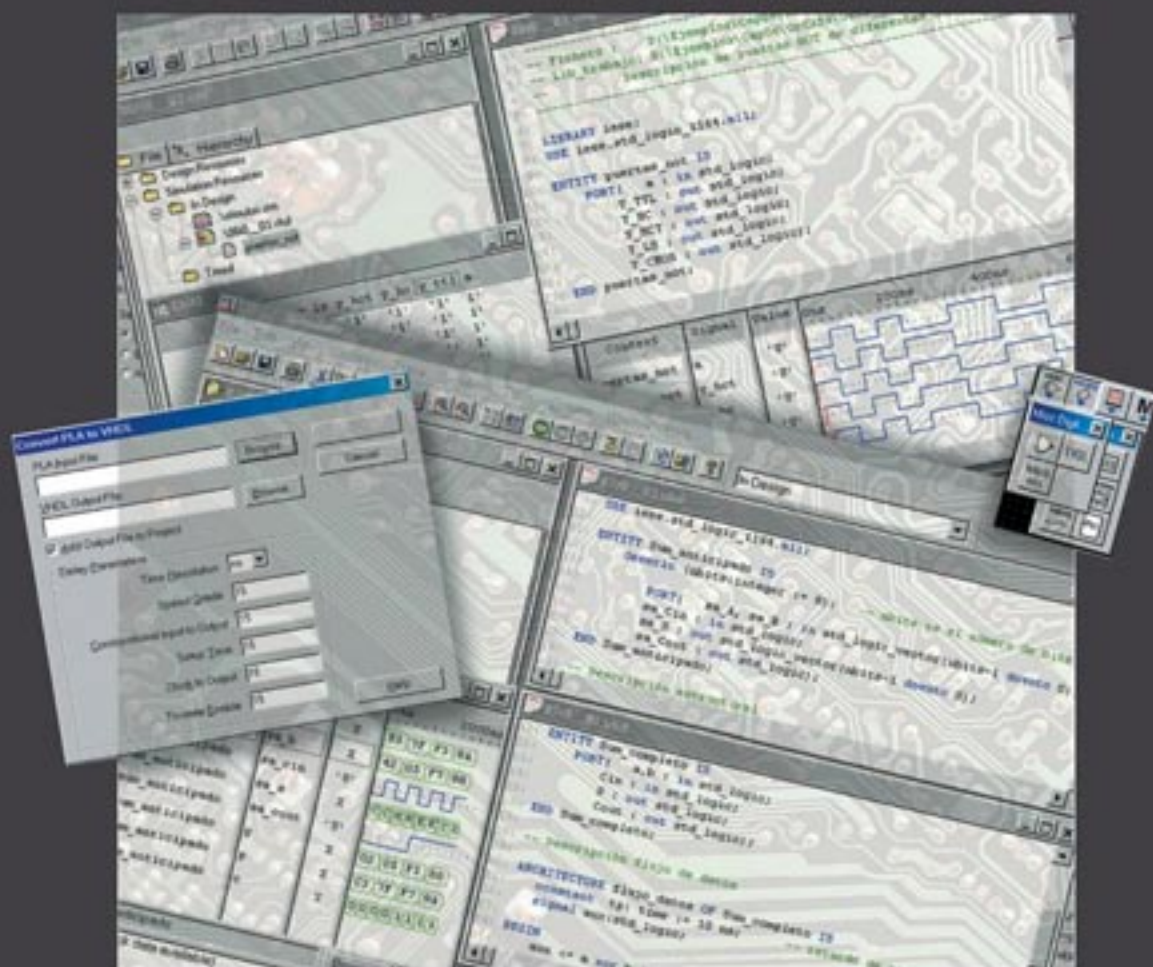


# Electrónica Digital

## Introducción a la Lógica Digital

### Teoría, Problemas y Simulación



**Santiago Acha • Manuel A. Castro**  
**Julio Pérez • Miguel A. Rioseras**



Incluye  
dos CD-ROM  
con programas  
y los ejemplos  
del libro

Alfaomega  Ra-Ma®

**Electrónica Digital:**  
**Introducción a la**  
**Lógica Digital**  
**Teoría, Problemas y Simulación**

# **Electrónica Digital: Introducción a la Lógica Digital**

## **Teoría, Problemas y Simulación**

Santiago Acha Alegre (Coordinador)

Julio Pérez Martínez (Coordinador)

Manuel-Alonso Castro Gil (Coordinador)

Miguel Ángel Riostras Gómez (Coordinador)

Adolfo Hilario Caballero

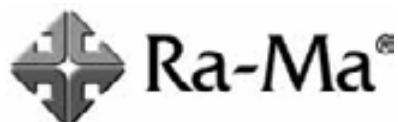
Rafael Sebastián Fernández

África López-Rey García-Rojas

Francisco Mur Pérez

Fernando Yeves Gutiérrez

Juan Peire Arroba



**ELECTRÓNICA DIGITAL.**

**INTRODUCCIÓN A LA**

**LÓGICA DIGITAL.**

**TEORÍA, PROBLEMAS Y**

**SIMULACIÓN**

# **ELECTRÓNICA DIGITAL. INTRODUCCIÓN A LA LÓGICA DIGITAL. TEORÍA, PROBLEMAS Y SIMULACIÓN**

**Santiago Acha Alegre (Coordinador)  
Julio Pérez Martínez (Coordinador)  
Manuel-Alonso Castro Gil (Coordinador)  
Miguel Angel Rioseras (Coordinador)  
Adolfo Hilario Caballero  
Rafael Sebastián Fernández  
África López-Rey García-Rojas  
Francisco Mur Pérez  
Fernando Yeves Gutiérrez  
Juan Peire Arroba**

*A mi esposa M<sup>a</sup> Ángeles e hijos Santi y Pablo,  
que juntos buscamos aventuras y  
recordamos momentos de felicidad.*

**Santiago**

*A mis padres, Manuel y Aurora,  
por su constante apoyo y entrega de cariño.  
Un recuerdo muy especial a la memoria de mi padre,  
por su alto valor humano como persona y como padre.*

**Julio**

*A los alumnos de la UNED, Universidad de Burgos y Escuela Politécnica  
Superior de Alcoy, y en especial a José Luis Beatobe, Rosa María  
Calleja, Jesús Castellano, Santiago Monteso, Antonio Nevado, Enrique  
Téllez y José Antonio Vernia.*

*Y a los profesores del DIEEC de la UNED por sus aportaciones.*

**Los autores**

## AUTORES

---

La presente obra ha sido desarrollada por un equipo de profesores y colaboradores del Departamento de Ingeniería Eléctrica, Electrónica y de Control de la Escuela Técnica Superior de Ingenieros Industriales de la Universidad Nacional de Educación a Distancia (UNED), (<http://www.ieec.uned.es/>).

### **Santiago Acha Alegre**

Ingeniero Industrial por la Escuela Técnica Superior de Ingenieros Industriales de la UNED, especialidad Electrónica y Automática e Ingeniero Técnico en Electricidad por la Escuela Universitaria Politécnica de Valladolid, especialidad Electrónica Industrial. Ha obtenido el Premio a los mejores Materiales Didácticos en Ciencias Experimentales del Consejo Social de la UNED en 1999.

Actualmente es Profesor Titular del Departamento de Electricidad y Electrónica en el I.E.S. Simón de Colonia de Burgos y Profesor Asociado en el Área de Tecnología Electrónica en el Departamento de Ingeniería Electromecánica de la Escuela Politécnica Superior de la Universidad de Burgos.

### **Julio Pérez Martínez**

Ingeniero Industrial por la Escuela Técnica Superior de Ingenieros Industriales de la UNED, especialidad Electrónica y Automática e Ingeniero Técnico Industrial por la Escuela Universitaria de Ingeniería Técnica Industrial de la Universidad de León, especialidad Electricidad, intensificación Electrónica, Regulación y Automatismos. Está realizando el Doctorado en el Departamento de Ingeniería Eléctrica, Electrónica y de Control de la ETSII de

la UNED. Ha obtenido el Premio Extraordinario de Estudios Fin de Carrera de la UNED. Ha obtenido el Premio a los mejores Materiales Didácticos en Ciencias Experimentales del Consejo Social de la UNED en 1999.

Actualmente trabaja en DMR Consulting - Estrategia y Tecnología de la Información, en el desarrollo e implantación de soluciones tecnológicas y de negocio en empresas líderes en su sector. Es colaborador del Departamento de Ingeniería Eléctrica, Electrónica y de Control, ETSII de la UNED. Ha trabajado como Profesor de Informática Aplicada en el C.E.A. San Bruno de Burgos, y en el Departamento de Mantenimiento Electrónico de ENDESA en La Coruña. Es miembro del IEEE.

## **Manuel-Alonso Castro Gil**

Doctor Ingeniero Industrial por la Escuela Técnica Superior de Ingenieros Industriales de la Universidad Politécnica de Madrid e Ingeniero Industrial, especialidad Electricidad, intensificación Electrónica y Automática por la misma Escuela. Ha obtenido el Premio Extraordinario de Doctorado de la UPM así como el Premio Viesgo 1988 a la Tesis Doctoral por la aportación a la Investigación Científica sobre Aplicaciones de la Electricidad en los Procesos Industriales. Ha obtenido el Premio a los mejores Materiales Didácticos en Ciencias Experimentales del Consejo Social de la UNED en 1997 y 1999.

Actualmente es Profesor Titular del Área de Tecnología Electrónica en el Departamento de Ingeniería Eléctrica, Electrónica y de Control de la UNED. Ha sido Director del Centro de Servicios Informáticos de la UNED y actualmente es Subdirector de Gestión Académica de la Escuela Técnica Superior de Ingenieros Industriales de la UNED. Ha trabajado cinco años en Digital Equipment Corporation. Es miembro Senior del IEEE, ACM, NYAS, ISES y del consejo de dirección de ISES España.

## **Miguel Ángel Rioseras Gómez**

Ingeniero Técnico de Telecomunicación por la Escuela Universitaria Politécnica de Alcalá de Henares, especialidad Equipos Electrónicos.

Es Profesor Titular del departamento de Electricidad y Electrónica en el I.E.S. Simón de Colonia de Burgos, en Ciclos Formativos de Grado Superior, especialidad Telecomunicación. Actualmente dedica su actividad profesional al desarrollo de proyectos de control y comunicación industrial en el ámbito I+D.

## **Adolfo Hilario Caballero**

Ingeniero Industrial, especialidad Electrónica y Automática por la Universidad Nacional de Educación a Distancia.

Actualmente es Profesor Titular de Escuela Universitaria en el Departamento de Ingeniería de Sistemas y Automática de la Universidad Politécnica de Valencia, Escuela Politécnica Superior de Alcoy. Miembro del Grupo de Investigación de Control de Sistemas Complejos en el mismo departamento. Colaborador en el Departamento de Ingeniería Eléctrica, Electrónica y de Control de la UNED.



## **Rafael Sebastián Fernández**

Ingeniero Industrial por la Escuela Técnica Superior de Ingenieros Industriales de la Universidad Politécnica de Madrid en las especialidades de Técnicas Energéticas y Electricidad, intensificación Electrónica y Automática.

Actualmente es Profesor Asociado en el Departamento de Ingeniería Eléctrica, Electrónica y de Control de la UNED. Ha sido profesor ayudante del mismo departamento durante los años 1983 a 1988. Posteriormente se incorporó desde 1988 hasta 1998 en ATHEL-EFANSA donde su actividad se centró principalmente en el diseño de sistemas electrónicos de control en los campos de automatización naval y producción de energía. Posteriormente se incorporó a Thyssen Ingeniería y Sistemas hasta principios de 2001 en el departamento de I+D trabajando en desarrollos electrónicos en los campos de elevación, automatización de almacenes y buses de campo.

## **África López-Rey García-Rojas**

Ingeniero/a Industrial por la Escuela Técnica Superior de Ingenieros Industriales de la Universidad Nacional de Educación a Distancia, especialidad Electrónica y Automática. Ha obtenido el Premio al mejor Material Didáctico en Ciencias Experimentales del Consejo Social de la UNED en 1999.

Actualmente es profesora ayudante de Ingeniería de Sistemas y Automática en el Departamento de Ingeniería Eléctrica, Electrónica y de Control de la ETSII de la UNED. Ha participado como colaboradora en diversos proyectos de investigación que han dado lugar a artículos y comunicaciones presentadas en congresos y revistas, en los que ha tomado parte activa tanto de ponente como coautora.

## **Francisco Mur Pérez**

Doctor Ingeniero Industrial por la Escuela Técnica Superior de Ingenieros Industriales de la UNED e Ingeniero Industrial, especialidad Electricidad, intensificación Electrónica y Automática por la Escuela Técnica Superior de Ingenieros Industriales de la Universidad Politécnica de Madrid. Ha obtenido el Premio Extraordinario de Doctorado de la UNED.

Actualmente es Profesor Titular en el Departamento de Ingeniería Eléctrica, Electrónica y de Control, ETSII de la UNED.

## **Fernando Yeves Gutiérrez**

Doctor Ingeniero Industrial por la Escuela Técnica Superior de Ingenieros Industriales de la Universidad Politécnica de Madrid e Ingeniero Industrial, especialidad Electricidad, intensificación Electrónica y Automática por la misma Escuela.

Actualmente es Profesor Titular del Área de Tecnología Electrónica en el Departamento de Ingeniería Eléctrica, Electrónica y de Control, ETSII de la UNED. Es miembro del IEEE.

## Juan Peire Arroba

Doctor Ingeniero Industrial por la Escuela Técnica Superior de Ingenieros Industriales de la Universidad Politécnica de Madrid e Ingeniero Industrial, especialidad Electricidad por la misma Escuela. Es Licenciado en Derecho por la Universidad Complutense de Madrid. Ha obtenido el Premio a los mejores Materiales Didácticos en Ciencias Experimentales del Consejo Social de la UNED en 1997 y 1999. Ha recibido el premio a la *“Innovative Excellence in Teaching, Learning & Technology”* del *“Center for the Advancement of Teaching and Learning”* del año 1999.

Actualmente es Catedrático de Universidad del Área de Tecnología Electrónica en el Departamento de Ingeniería Eléctrica, Electrónica y de Control de la UNED, habiendo sido Director del Departamento. Ha trabajado varios años como Consultor especializado en la creación de Empresas Tecnológicas, así como ha dirigido y dirige diversos proyectos de investigación tanto nacionales como internacionales. Es miembro del IEEE.

# CONTENIDO

---

---

PRÓLOGO.....	XXI
INTRODUCCIÓN.....	XXIII
LISTA DE TABLAS .....	XXVII
SIMBOLOGÍA Y NOMENCLATURA.....	XXXI
PROGRAMAS UTILIZADOS .....	XXXV

## **CAPÍTULOS:**

1. FUNDAMENTOS GENERALES DE LA ELECTRÓNICA DIGITAL ....	1
2. CODIFICACIÓN DE LA INFORMACIÓN.....	61
3. ÁLGEBRA DE CONMUTACIÓN Y SU REPRESENTACIÓN .....	95

4. FUNCIONES LÓGICAS BÁSICAS .....	133
5. SIMPLIFICACIÓN DE FUNCIONES LÓGICAS.....	229
6. TECNOLOGÍAS DE CIRCUITOS INTEGRADOS DIGITALES .....	307
7. CIRCUITOS ARITMÉTICOS .....	423
8. CONVERTIDORES A/D Y D/A.....	471
9. CONEXIÓN CON LÓGICA DIGITAL INTEGRADA .....	545

## APÉNDICES:

A. FUENTES Y GENERADORES.....	547
B. MANUAL BREVE DEL LENGUAJE VHDL Y NOTACIÓN BNF.....	571
C. SIMULADORES VHDL .....	627
D. SIMULACIÓN VHDL CON <i>ORCAD DEMO V9</i> .....	637
E. GUÍA DE <i>ELECTRONICS WORKBENCH 5.0</i> .....	659
F. GUÍA DE <i>ORCAD DEMO V9</i> .....	673
G. CONTENIDO DE LOS CD-ROM .....	685
H. LISTA DE EJEMPLOS.....	691
I. LISTA DE HOJAS DE DATOS .....	699
J. LECTOR DE DOCUMENTOS ELECTRÓNICOS ADOBE ACROBAT..	709
K. BIBLIOGRAFÍA.....	717
L. ÍNDICE ALFABÉTICO .....	719

# ÍNDICE

---

---

<b>PRÓLOGO.....</b>	<b>XXI</b>
<b>INTRODUCCIÓN .....</b>	<b>XXIII</b>
<b>LISTA DE TABLAS .....</b>	<b>XXVII</b>
<b>SIMBOLOGÍA Y NOMENCLATURA.....</b>	<b>XXXI</b>
<b>PROGRAMAS UTILIZADOS.....</b>	<b>XXXV</b>
<b>CAPÍTULO 1. FUNDAMENTOS GENERALES DE LA ELECTRÓNICA DIGITAL .....</b>	<b>1</b>
<b>1.1 Sistemas analógicos y digitales.....</b>	<b>2</b>
<b>1.2 Representación de la información .....</b>	<b>7</b>
1.2.1 Sistemas de numeración .....	7
1.2.1.1 Elección del sistema de numeración.....	9
1.2.1.2 Consideraciones importantes sobre sistemas polinomiales .....	11
1.2.1.3 Conversión entre bases .....	13
1.2.1.4 Sistema binario .....	21
1.2.1.5 Sistema octal.....	25

- 1.2.1.6 Sistema hexadecimal ..... 26
- 1.2.2 Representaciones numéricas..... 32
- 1.2.3 Representación de números en coma fija sin signo..... 33
  - 1.2.3.1 Representación de números naturales en binario puro ..... 33
- 1.2.4 Representación de números en coma fija con signo..... 34
  - 1.2.4.1 Formato de números binarios en signo-magnitud..... 34
  - 1.2.4.2 Complementos ..... 37
  - 1.2.4.3 Complemento a la base ..... 37
  - 1.2.4.4 Complemento a la base menos uno..... 38
  - 1.2.4.5 Convenio del complemento a dos en números binarios ..... 39
  - 1.2.4.6 Convenio del complemento a uno en números binarios ..... 46
  - 1.2.4.7 Comparación entre las representaciones diferentes de números binarios con signo..... 48
- 1.2.5 Representación de los números reales en binario..... 49
  - 1.2.5.1 Coma flotante..... 51

**CAPÍTULO 2. CODIFICACIÓN DE LA INFORMACIÓN..... 61**

- 2.1 Definiciones y propiedades de la codificación ..... 62**
  - 2.1.1 Definiciones..... 62
  - 2.1.2 Propiedades de interés de los códigos ..... 63
- 2.2 Códigos binarios..... 68**
  - 2.2.1 Principales definiciones y propiedades de los códigos binarios..... 69
- 2.3 Tipos ..... 70**
  - 2.3.1 Códigos numéricos ..... 70
    - 2.3.1.1 Código binario natural ..... 70
    - 2.3.1.2 Códigos BCD..... 71
    - 2.3.1.3 Códigos continuos y cíclicos ..... 77
    - 2.3.1.4 Código Gray ..... 78
    - 2.3.1.5 Código Johnson ..... 81
  - 2.3.2 Códigos alfanuméricos ..... 83
    - 2.3.2.1 Código ASCII ..... 83
  - 2.3.3 Códigos detectores de error ..... 85
    - 2.3.3.1 Códigos de paridad ..... 85
    - 2.3.3.2 Códigos de peso fijo ..... 87
  - 2.3.4 Códigos correctores de error ..... 87

**CAPÍTULO 3. ÁLGEBRA DE CONMUTACIÓN Y SU REPRESENTACIÓN ..... 95**

- 3.1 Álgebra de boole..... 96**
  - 3.1.1 Definición de álgebra de Boole ..... 96
- 3.2 Teoremas del álgebra de Boole ..... 97**
- 3.3 Álgebra de Boole bivalente..... 105**

3.3.1 Variables y funciones lógicas.....	108
3.3.2 Representación de las funciones lógicas mediante tablas de verdad.....	110
3.3.3 Representación de las funciones lógicas en su forma canónica .....	113
3.3.4 Obtención de la función canónica a partir de la tabla de verdad. Teorema de expansión .....	114
3.3.5 Conversión entre expresiones canónicas en <i>minterms</i> y <i>maxterms</i> .....	122
3.3.6 Conversión de expresiones normalizadas a canónicas .....	123
3.3.7 Conjunto de funciones de dos variables .....	125
3.3.8 Función incompletamente definida .....	130

## **CAPÍTULO 4. FUNCIONES LÓGICAS BÁSICAS..... 133**

<b>4.1 Funciones lógicas básicas.....</b>	<b>134</b>
4.1.1 Función AND (puerta AND).....	136
4.1.2 Función OR (puerta OR) .....	159
4.1.3 Función NOT (puerta lógica inversora) .....	169
4.1.4 Función NAND (puerta NAND).....	178
4.1.5 Función NOR (puerta NOR) .....	188
4.1.6 Función SEGUIDOR o puerta <i>BUFFER</i> .....	199
4.1.7 Función XOR .....	207
4.1.8 Función XNOR .....	217

## **CAPÍTULO 5. SIMPLIFICACIÓN DE FUNCIONES LÓGICAS... 229**

<b>5.1 Simplificación de funciones lógicas .....</b>	<b>230</b>
5.1.1 Método algebraico de simplificación .....	231
5.1.2 Métodos sistemáticos de simplificación.....	237
5.1.2.1 Definiciones previas. Adyacencias.....	237
5.1.2.2 Definiciones y propiedades de funciones mínimas.....	240
5.1.2.3 Método sistemático de simplificación .....	243
5.1.3 Método de <i>Karnaugh</i> .....	243
5.1.3.1 Mapas de <i>Karnaugh</i> de más de cuatro variables .....	262
5.1.4 Método de <i>Quine-McCluskey</i> .....	269
5.1.4.1 Método de <i>Q-M</i> con coordenadas binarias .....	269
5.1.4.2 Método de <i>Q-M</i> con coordenadas decimales.....	271
5.1.4.3 Tabla de implicantes .....	273
5.1.4.4 Tabla de implicantes reducida .....	275
5.1.4.5 Tablas cíclicas y método de <i>Petrick</i> .....	276
<b>5.2 Conversión a puertas NAND y NOR.....</b>	<b>279</b>
5.2.1 Circuitos con dos niveles.....	279
5.2.2 Circuitos con más de dos niveles .....	281
5.2.3 Circuitos con cualquier número de niveles y tipo de puertas.....	284
<b>5.3 Simplificación de funciones incompletas o con indiferencias .....</b>	<b>285</b>
<b>5.4 Simplificación multifuncional.....</b>	<b>292</b>

<b>CAPÍTULO 6. TECNOLOGÍAS DE CIRCUITOS INTEGRADOS DIGITALES .....</b>	<b>307</b>
<b>6.1 Introducción a las características básicas de las puertas lógicas.....</b>	<b>308</b>
<b>6.2 Características generales de los circuitos integrados digitales .....</b>	<b>309</b>
6.2.1 Características estáticas de los circuitos integrados digitales .....	310
6.2.1.1 Características de transferencia y niveles lógicos .....	310
6.2.1.2 Características de entrada y salida .....	314
6.2.1.3 Inmunidad frente al ruido .....	322
6.2.1.4 Consumo o disipación de potencia .....	324
6.2.2 Características dinámicas de los circuitos integrados digitales .....	327
6.2.2.1 Retardos de propagación.....	327
6.2.2.2 Frecuencia máxima de funcionamiento .....	329
6.2.2.3 Producto consumo por tiempo de propagación .....	330
6.2.3 Otras características de los circuitos integrados digitales .....	330
6.2.3.1 Flexibilidad lógica .....	330
6.2.3.2 Margen de temperatura .....	331
6.2.3.3 Coste .....	332
6.2.4 Características ideales de una puerta lógica .....	332
<b>6.3 Familias lógicas .....</b>	<b>333</b>
<b>6.4 Familia lógica TTL.....</b>	<b>334</b>
6.4.1 Introducción .....	334
<b>6.5 Puerta TTL-Estándar .....</b>	<b>335</b>
6.5.1 Constitución .....	335
6.5.2 Transistor multiemisor .....	336
6.5.3 Funcionamiento de la puerta TTL-Estándar. Análisis en continua .....	336
6.5.3.1 Todas las entradas con nivel alto .....	336
6.5.3.2 Alguna entrada con nivel bajo .....	338
6.5.3.3 Entradas sin conexión .....	340
6.5.4 Consideraciones de diseño de las resistencias.....	342
6.5.5 Características funcionales de la familia TTL-Estándar .....	343
6.5.5.1 Características estáticas de la familia TTL-Estándar.....	343
6.5.5.2 Características dinámicas de la familia TTL-Estándar .....	351
6.5.6 Representación de las características mediante perfiles de entrada y salida ..	352
<b>6.6 Subfamilias TTL .....</b>	<b>353</b>
6.6.1 TTL de baja potencia (LTTL, serie 54L/74L).....	353
6.6.2 Puerta TTL tipo <i>Schottky</i> (STTL, serie 54S/74S) .....	353
6.6.3 TTL <i>Schottky</i> de bajo consumo (LSTTL, serie 54LS/74LS) .....	356
6.6.4 TTL <i>Schottky</i> de bajo consumo mejorada (ALSTTL, serie 54ALS/74ALS). 359	
6.6.5 TTL <i>Schottky</i> mejorada (ASTTL, serie 54AS/74AS) .....	360
6.6.6 TTL de alta velocidad (FTTL, serie 54F/74F) .....	362
<b>6.7 Comparación de subfamilias TTL.....</b>	<b>363</b>
<b>6.8 Puertas TTL con otro tipo de salidas .....</b>	<b>365</b>



6.8.1 Puertas TTL con salida en colector abierto .....	365
6.8.2 Puertas TTL con salida con control triestado .....	370
<b>6.9 Precauciones en el diseño de sistemas con tecnología TTL .....</b>	<b>374</b>
<b>6.10 Familia Lógica CMOS .....</b>	<b>375</b>
6.10.1 Introducción .....	375
6.10.2 Constitución del circuito básico CMOS .....	375
6.10.3 Características estáticas de la familia CMOS .....	377
6.10.3.1 Característica de transferencia .....	377
6.10.3.2 Característica de entrada y salida .....	379
6.10.3.3 Inmunidad frente al ruido .....	381
6.10.3.4 Consumo y disipación de potencia .....	381
6.10.4 Características dinámicas de la familia CMOS .....	383
6.10.4.1 Retardos de propagación .....	383
6.10.4.2 Frecuencia máxima de funcionamiento .....	383
6.10.4.3 Producto consumo por tiempo de propagación .....	383
6.10.5 Otras características de la familia CMOS .....	384
6.10.5.1 Flexibilidad lógica .....	384
6.10.5.2 Coste .....	384
<b>6.11 Subfamilias CMOS .....</b>	<b>384</b>
6.11.1 CMOS estándar (serie 4000, 4000A, 4000B y 4000UB) .....	384
6.11.2 CMOS - TTL (serie 54C/74C) .....	386
6.11.3 CMOS de alta velocidad (HCMOS, serie 54HC/74HC) .....	386
6.11.4 CMOS avanzada (ACL, series 74AC y 74ACT) .....	387
6.11.5 Otras subfamilias .....	387
<b>6.12 Comparativa de las subfamilias CMOS .....</b>	<b>388</b>
6.12.1 Resumen de los perfiles de las familias TTL y CMOS .....	389
6.12.2 Perfiles de entrada de familias TTL y CMOS .....	389
6.12.3 Perfiles de salida de familias TTL y CMOS .....	389
6.12.4 Producto consumo por tiempo de propagación .....	390
<b>6.13 Puertas CMOS con otro tipo de salidas .....</b>	<b>390</b>
6.13.1 Puertas CMOS con salida en drenador abierto .....	390
6.13.2 Puertas CMOS con salida triestado .....	392
<b>6.14 Precauciones en la manipulación de dispositivos CMOS .....</b>	<b>393</b>
<b>6.15 Interfaces entre familias lógicas .....</b>	<b>394</b>
6.15.1 Alimentaciones iguales en la puerta excitadora y excitada .....	396
6.15.1.1 Interfaz CMOS (VDD = 5 V) a TTL .....	396
6.15.1.2 Interfaz TTL a CMOS (VDD = 5 V) .....	397
6.15.1.3 Interfaces específicas HCT Y ACT .....	399
6.15.2 Alimentación de la puerta excitadora menor que la de la puerta excitada ...	400
6.15.2.1 Interfaz específica 4104 .....	402
6.15.3 Alimentación de la puerta excitadora mayor que la de la puerta excitada ...	403
6.15.3.1 Interfaces específicas 4049 y 4050 .....	405

6.15.4 Tabla resumen de interfaces mediante adaptadores de niveles entre familias lógicas .....	406
<b>6.16 Simulación en VHDL de una familia lógica .....</b>	<b>407</b>
6.16.1 Asignaciones con retrasos .....	407
6.16.2 Simulación de niveles lógicos .....	414
<b>CAPÍTULO 7. CIRCUITOS ARITMÉTICOS .....</b>	<b>423</b>
<b>7.1 Circuitos aritméticos. Introducción.....</b>	<b>424</b>
<b>7.2 Sumadores binarios.....</b>	<b>424</b>
7.2.1 Semisumador .....	424
7.2.2 Sumador completo.....	426
7.2.3 Sumador paralelo con acarreo serie.....	432
7.2.4 Sumador paralelo con acarreo paralelo .....	438
7.2.5 Sumador paralelo con acarreo mixto.....	445
<b>7.3 Restadores binarios.....</b>	<b>446</b>
7.3.1 Aritmética en complemento a dos .....	447
7.3.2 Circuito complementador a dos.....	452
7.3.3 Circuito sumador-restador en binario signo magnitud .....	456
<b>7.4 Unidad Aritmético-Lógica (ALU) .....</b>	<b>460</b>
<b>7.5 Aplicaciones de los circuitos aritméticos.....</b>	<b>467</b>
7.5.1 Comparadores de magnitud binarios .....	467
7.5.2 Convertidor de código BCD a Exceso-3 .....	469
<b>CAPÍTULO 8. CONVERTIDORES A/D Y D/A .....</b>	<b>471</b>
<b>8.1 Convertidores A/D y D/A. Introducción .....</b>	<b>472</b>
<b>8.2 Convertidor D/A.....</b>	<b>473</b>
8.2.1 Convertidor D/A. Generalidades.....	473
8.2.2 Especificaciones de los convertidores D/A .....	474
8.2.3 Circuitos convertidores D/A.....	477
8.2.3.1 Convertidores D/A con resistencias ponderadas .....	478
8.2.3.2 Convertidores D/A con red R-2R .....	481
8.2.4 Funcionamiento bipolar de los convertidores D/A.....	488
8.2.5 El convertidor comercial DAC 0800.....	492
8.2.5.1 Funcionamiento unipolar .....	493
8.2.5.2 Funcionamiento bipolar .....	495
<b>8.3 Convertidor A/D.....</b>	<b>501</b>
8.3.1 Convertidor A/D. Generalidades.....	501
8.3.2 Especificaciones de los convertidores A/D .....	508
8.3.3 Circuitos convertidores A/D.....	509
8.3.3.1 Convertidor A/D instantáneo.....	510

8.3.3.2 Convertidor A/D de rampa .....	513
8.3.3.3 Convertidor A/D de doble rampa .....	518
8.3.3.4 Convertidor A/D por contador.....	526
8.3.3.5 Convertidor A/D por aproximaciones sucesivas .....	531
8.3.3.6 Convertidores A/D Sigma-Delta.....	535
8.3.4 El convertidor comercial ADC 0805.....	538

## **CAPÍTULO 9. CONEXIÓN CON LÓGICA DIGITAL INTEGRADA ..... 545**

### **APÉNDICES:**

<b>A. FUENTES Y GENERADORES .....</b>	<b>547</b>
<b>B. MANUAL BREVE DEL LENGUAJE VHDL Y NOTACIÓN BNF.....</b>	<b>571</b>
<b>C. SIMULADORES VHDL .....</b>	<b>627</b>
<b>D. SIMULACIÓN VHDL CON ORCAD DEMO V9.....</b>	<b>637</b>
<b>E. GUÍA DE ELECTRONICS WORKBENCH 5.0 .....</b>	<b>659</b>
<b>F. GUÍA DE ORCAD DEMO V9 .....</b>	<b>673</b>
<b>G. CONTENIDO DE LOS CD-ROM.....</b>	<b>685</b>
<b>H. LISTA DE EJEMPLOS .....</b>	<b>691</b>
<b>I. LISTA DE HOJAS DE DATOS .....</b>	<b>699</b>
<b>J. LECTOR DE DOCUMENTOS ELECTRÓNICOS ADOBE ACROBAT .....</b>	<b>709</b>
<b>K. BIBLIOGRAFÍA .....</b>	<b>717</b>
<b>L. ÍNDICE ALFABÉTICO .....</b>	<b>719</b>

## PRÓLOGO

---

---

La Electrónica es una disciplina que desde sus orígenes a comienzos del siglo XX ha sufrido un avance espectacular, más aún desde la aparición de los semiconductores a finales de los años cuarenta, principio de los cincuenta.

Este avance se caracteriza por una evolución constante de la tecnología de componentes y dispositivos electrónicos, que ha permitido un aumento extraordinario de la complejidad de los circuitos. La complejidad tiene a su vez un gran impacto en los métodos y procedimientos de diseño, no siendo posible diseñar circuitos mediante la simple interconexión de bloques funcionales conocidos, o mediante la prueba y error en el montaje de un prototipo, según se van aumentando las funcionalidades y características de los circuitos. Este hecho se manifiesta claramente en la Electrónica Digital.

Hoy la simulación de circuitos se ha convertido en un paso obligado en cualquier metodología de diseño por las innumerables ventajas que comporta su utilización. Los simuladores han evolucionado muy rápidamente, junto a los modelos de componentes de todo tipo, lo que unido a la evolución de los ordenadores sobre los que funcionan estos programas, han convertido la simulación en una herramienta muy eficaz y por lo tanto imprescindible.

Este libro aborda el estudio de la Electrónica Digital integrando en su metodología de trabajo la utilización sistemática de las herramientas de simulación. Cualquier libro de Electrónica moderno debe abordarse desde esta perspectiva, aprovechando las

enormes posibilidades que ofrece la simulación, pero sin olvidar que los aspectos conceptuales continúan siendo el objetivo básico del aprendizaje.

En este trabajo se sirve de este propósito, siendo clásico en lo conceptual y moderno en lo metodológico, de modo y manera que sirve a su objetivo fundamental, ayudar a la formación de estudiantes en Electrónica Digital.

Por todo ello, la obra resultará de gran utilidad a los estudiantes universitarios que se acerquen por primera vez a esta disciplina, aportando una visión conceptual y práctica, en castellano, que supone una contribución valiosa a la bibliografía en Electrónica, así como a los profesionales de este ámbito que quieran reforzar sus conocimientos o adquirir un nuevo aspecto en el enfoque de las herramientas aplicadas a las mismas.

Los autores

# INTRODUCCIÓN

---

---

Cuando se comenzó la realización de la presente obra, el objetivo era abarcar dentro de la misma la mayor parte de los contenidos de la Electrónica Digital, comenzando por el estudio de los aspectos que sientan la base de esta disciplina, y finalizando con la descripción de la lógica digital integrada, habiendo pasado, en todo este gran salto, por la descripción de los distintos componentes que forman parte de este campo. Observando la gran extensión que constituía este ambicioso proyecto, se decidió desarrollar dos tomos, que sean a la vez complementarios en su contenido e independientes en su estudio, en función de los conocimientos y del interés que tenga cada lector, en centrarse más en el primero, en el segundo o en ambos.

Así, el resultado final de la obra son dos libros: “Electrónica Digital. Introducción a la Lógica Digital. Teoría, Problemas y Simulación” y “Electrónica Digital. Lógica Digital Integrada. Teoría, Problemas y Simulación”. El primero de ellos, es la presente obra, tiene un carácter más analítico y de introducción a los fundamentos de diseño, mientras que el segundo está más orientado a la síntesis e implementación de sistemas electrónicos.

Centrándose en este primer tomo, el objetivo que se persigue en el mismo es dotar al lector de una forma distinta de acercarse al mundo de la Electrónica Digital, ya que en él se muestran los aspectos teóricos propios de la Electrónica Digital, pero tratados de una forma práctica, basándose en el uso de las herramientas y aplicaciones informáticas de simulación más utilizadas en el mundo profesional.

En este primer libro, se pretende que el lector adquiriera los conocimientos teóricos propios de un curso de Introducción a la Lógica Digital Integrada a la vez que se

introduce y se familiariza en el manejo de estas herramientas de simulación, que cada vez resultan más imprescindibles en el proceso de diseño de circuitos electrónicos asistido por ordenador. La apuesta por la herramienta de simulación se basa en que ésta permite evaluar el funcionamiento del circuito antes de construir el primer prototipo real con componentes físicos, lo cual conlleva una serie de ventajas que hacen imprescindible su utilización: ahorro de costes, reducción de tiempos en diseño y pruebas, facilidad en la experimentación, etc.

Las herramientas que se utilizan en este libro para realizar la simulación de los circuitos son las siguientes:

- *Electronics Workbench 5 Demo*.

Se trata de una versión limitada para estudiantes del programa profesional *Electronics Workbench* en castellano. Esta versión, de distribución gratuita, permite el diseño y simulación de circuitos analógicos, digitales y mixtos.

- *OrCAD Demo v9*.

Se trata de una versión limitada del programa *OrCAD* en inglés y de distribución gratuita, que permite el diseño y simulación de circuitos analógicos, digitales y mixtos, y la descripción lógica de componentes electrónicos mediante el lenguaje VHDL.

- *VeriBest V99*.

Se trata de una versión limitada del programa *VeriBest* en inglés y de distribución gratuita, que permite el diseño y simulación de componentes electrónicos en lenguaje VHDL.

Estas aplicaciones se encuentran en el CD-ROM que se incluye en este libro (a excepción de VeriBest); además, es posible actualizar estas versiones en las siguientes direcciones de Internet:

- Electronics Workbench. URL Internet: <http://www.interactive.com/>
- OrCAD. URL Internet: <http://www.orcad.com/>
- VeriBest. URL Internet: <http://www.mentor.com/>

Este libro se compone de nueve capítulos, los ocho primeros están dedicados al estudio de distintos temas de la Electrónica Digital y el noveno es un nexo de unión con la segunda parte de esta obra, que trata temas avanzados de la Lógica Digital Integrada.

En el Capítulo 1, “Fundamentos generales de la electrónica digital”, se describen las diferencias entre los sistemas analógicos y digitales, analizando las ventajas y los inconvenientes que presenta cada uno de ellos; se estudia la representación de la información numérica en los sistemas de numeración que más se utilizan habitualmente (binario, octal, hexadecimal, etc.), el manejo de representaciones numéricas en coma fija y en coma flotante, así como los cambios entre las distintas bases. Se utiliza la calculadora de Windows en modo “Científica” para obtener las

representaciones de números enteros con signo, según el convenio del complemento a dos, en las bases: binaria, octal, hexadecimal y decimal.

En el Capítulo 2, “Codificación de la información”, se describe la representación de la información mediante el conocimiento de las propiedades y principales aplicaciones de los códigos binarios; se utilizan los códigos de numeración más empleados para almacenar y transmitir información (BCD, Gray, Johnson, ASCII, etc.), así como los códigos de detección y corrección de errores.

En el Capítulo 3, “Álgebra de conmutación y su representación”, se estudia el álgebra de Boole como herramienta matemática básica para el análisis y síntesis de circuitos digitales, analizando las definiciones y teoremas de la misma. Se realiza la representación e interpretación de las funciones lógicas mediante su expresión canónica y su tabla de verdad. Mediante el programa de simulación *Electronics Workbench 5.0* se comprueban los principales teoremas y leyes del álgebra de Boole.

En el Capítulo 4, “Funciones lógicas básicas”, se describen las funciones lógicas desde sus aspectos más característicos, tales como: definición, operación asociada, diagrama de Venn, conexionado eléctrico, tabla de verdad, expresión algebraica, cronograma, simbología y circuitos integrados comerciales. Mediante los programas de simulación *Electronics Workbench 5.0*, *OrCAD Demo v9* y el simulador de VHDL *VeriBest VB99* se comprueban las distintas funciones lógicas, sus tablas de verdad y cronogramas.

En el Capítulo 5, “Simplificación de funciones lógicas”, se describen los métodos de simplificación de funciones lógicas múltiples: algebraico, *Karnaugh* y *Quine-McCluskey*, analizando cuál de ellos más se ajusta mejor para cada caso con el fin de optimizar los diseños, reducir el número de componentes o puertas lógicas necesarias en la realización de un sistema digital. Mediante el programa de simulación *Electronics Workbench 5.0* se comprueban los resultados obtenidos con los diferentes métodos de simplificación.

En el Capítulo 6, “Tecnologías de circuitos integrados digitales”, se realiza una introducción a las características internas de las puertas lógicas, de sus características funcionales, tanto estáticas como dinámicas y de las diferentes tecnologías digitales o familias lógicas que existen actualmente. Se describe la evolución de las tecnologías digitales, la familia lógica TTL y la familia lógica CMOS, así como realizar la interpretación de las hojas características de los circuitos integrados digitales, y elegir la tecnología más adecuada dependiendo de las especificaciones que deba reunir el sistema digital a realizar (consumo, velocidad, inmunidad al ruido, etc.). Mediante los programas de simulación *Electronics Workbench 5.0*, *OrCAD Demo v9* y *VeriBest VB99*, se comprueban las características funcionales de las puertas lógicas, tanto a partir de su esquema como utilizando el lenguaje de descripción hardware VHDL.

En el Capítulo 7, “Circuitos aritméticos”, se describen los principios básicos sobre los que se fundamenta la lógica aritmética, los distintos procedimientos para la realización de circuitos sumadores y restadores, y se analizan los distintos circuitos



aritméticos, sus especificaciones, conexionado y aplicaciones típicas. Se utilizarán los programas de simulación *Electronics WorkBench 5.0* y *OrCAD Demo v9* para la realización de circuitos aritméticos, así como para la comprobación de los ejercicios resueltos.

En el Capítulo 8, “Convertidores A/D y D/A”, se describe el principio de funcionamiento de los distintos convertidores A/D y D/A existentes, sus ventajas e inconvenientes, así como los distintos procedimientos utilizados para realizar la conversión A/D y D/A, sus especificaciones básicas y los procedimientos de análisis. Se analizan las especificaciones típicas de los convertidores A/D y D/A y se realiza el análisis y diseño de circuitos convertidores A/D y D/A. Se utilizarán los programas de simulación *Electronics WorkBench 5.0* y *OrCAD Demo v9* para la realización de convertidores A/D y D/A, así como para la comprobación de ejercicios resueltos.

En el Capítulo 9, “Conexión con Tomo II: Lógica Digital Integrada”, se realiza una breve descripción del contenido de cada tomo, así como una indicación de a qué perfil de lector, en función de su conocimiento e interés, puede interesar más cada uno de ellos.

De forma complementaria a los capítulos se incluyen una serie de apéndices. En el Apéndice A se incluye una descripción de los distintos tipos de Fuentes y Generadores de señales que habitualmente se utilizan en Electrónica. En el Apéndice B se incluye un manual breve del lenguaje VHDL y la notación BNF, en el que se realiza una descripción de este lenguaje de descripción cada vez más utilizado hoy en día. En el Apéndice C se realiza una descripción de los principales simuladores VHDL que actualmente se encuentran disponibles en el mercado. En el Apéndice D se incluye una guía para llevar a cabo la instalación del programa de simulación en castellano “*Electronics Workbench*”, uno de los pocos programas de este tipo existentes en nuestro idioma. En el Apéndice E se incluye una guía para llevar a cabo la instalación del programa de simulación en inglés “*OrCAD*”.

En el Apéndice F se realiza una descripción del contenido de los CD-ROM que acompañan al libro. En el Apéndice G se encuentra una lista de ejemplos en la que se indica la ruta y el nombre del archivo que contiene el circuito que se ha utilizado con cada programa para realizar las simulaciones, así como una breve descripción de los mismos. En el Apéndice H se muestra, a modo de ejemplo, una lista de hojas de datos que muestra las características de un componente y que sirve de referencia al lector para facilitar la forma de cómo consultar las hojas de datos que, en formato electrónico, se incluyen en el CDROM#2 que también acompaña al libro. En el Apéndice I se describe la aplicación *Adobe Acrobat Reader*, que permite visualizar los documentos, en formato PDF, que se incluyen en los CD-ROM que acompañan al libro. En el Apéndice J se detalla la bibliografía que se ha consultado durante la realización de la presente obra y que puede permitir al lector ampliar conocimientos. Y finalmente, en el Apéndice K se incluye un índice cruzado de apariciones de términos y palabras clave necesarios para la búsqueda de temas clave en la obra.

## LISTA DE TABLAS

---

---

Tabla 1.1. Ejemplos de sistemas de numeración .....	7
Tabla 1.2. Proceso de conversión de base 10 a base b2 en parte entera de un número.....	14
Tabla 1.3. Conversión del número 324(10 de decimal a binario.....	15
Tabla 1.4. Proceso de conversión de base 10 a base b2 en parte fraccionaria de un número ....	18
Tabla 1.5. Conversión del número 0,375(10 de decimal a binario.....	18
Tabla 1.6. Números binarios de 4 bits.....	24
Tabla 1.7. Números octales y su relación con decimales y binarios .....	25
Tabla 1.8. Números hexadecimales y su relación con decimales y binarios.....	27
Tabla 1.9. Representación de números binarios de 4 bits mediante el convenio de complemento a dos.....	42
Tabla 1.10. Representación de números binarios de 4 bits mediante el convenio de complemento a uno .....	47
Tabla 1.11. Números binarios con signo de cuatro bits representados en las tres diferentes formas estudiadas.....	49
Tabla 1.12. Casos especiales de representación del número N, en coma flotante, según el estándar IEEE 754.....	56
Tabla 2.1. Ejemplo de código.....	63
Tabla 2.2. Ejemplo de código uniforme y no singular .....	63
Tabla 2.3. Extensión de orden dos de un código.....	64
Tabla 2.4. Ejemplos de códigos unívocamente decodificables .....	65
Tabla 2.5. Código A y sus prefijos .....	66
Tabla 2.6. Código B y sus prefijos .....	66
Tabla 2.7. Código C y sus prefijos .....	67

Tabla 2.8. Número de códigos distintos que se pueden formar para codificar un alfabeto fuente dado.....	68
Tabla 2.9. Código binario natural de 4 bits.....	70
Tabla 2.10. Propiedades del código binario natural.....	71
Tabla 2.11. Código BCD Natural o BCD 8421.....	72
Tabla 2.12. Propiedades del código BCD Natural o BCD 8421.....	73
Tabla 2.13. Códigos BCD Aiken.....	73
Tabla 2.14. Propiedades de los códigos BCD Aiken.....	74
Tabla 2.15. Código BCD 642-3.....	74
Tabla 2.16. Propiedades del código BCD 642-3.....	75
Tabla 2.17. Código BCD de exceso 3.....	75
Tabla 2.18. Propiedades de los códigos BCD de exceso 3.....	76
Tabla 2.19. Construcción del código Gray o código reflejado.....	78
Tabla 2.20. Código Gray de cuatro bits.....	79
Tabla 2.21. Propiedades del código Gray.....	79
Tabla 2.22. Código Johnson.....	82
Tabla 2.23. Propiedades del código Johnson.....	82
Tabla 2.24. Resumen de las propiedades de los códigos.....	83
Tabla 2.25. Código alfanumérico ASCII.....	84
Tabla 2.26. Ejemplo de código de paridad correspondiente al código base BCD natural.....	86
Tabla 2.27. Códigos detectores de error de palabra fija: 2 entre 5 y biquinario.....	87
Tabla 2.28. Palabra de test de paridad para el código Hamming.....	89
Tabla 2.29. Código corrector de error Hamming generado a partir del código BCD natural... ..	92
Tabla 3.1. Definición de las operaciones suma lógica + y producto lógico ·.....	106
Tabla 3.2. Complemento lógico.....	106
Tabla 3.3. Comprobación de la ley distributiva del producto lógico sobre la suma lógica.....	107
Tabla 3.4. Comprobación de que el álgebra bivalente cumple el postulado quinto.....	108
Tabla 3.5. Representación de la función f3 mediante su tabla de verdad.....	111
Tabla 3.6. Tabla de minterms y maxterms para una función de tres variables.....	114
Tabla 3.7. Tabla de verdad de las dieciséis funciones distintas con dos variables.....	125
Tabla 3.8. Tabla resumen de las funciones que se pueden formar con dos variables.....	127
Tabla 3.9. Tabla de verdad de una función incompletamente definida.....	130
Tabla 4.1. Tabla de verdad de una puerta AND de dos variables de entrada.....	137
Tabla 4.2. Tabla de verdad de una puerta OR de dos variables de entrada.....	160
Tabla 4.3. Tabla de verdad de una puerta NOT.....	170
Tabla 4.4. Tabla de verdad de una puerta NAND de dos variables de entrada.....	179
Tabla 4.5. Tabla de verdad de una puerta NOR de dos variables de entrada.....	190
Tabla 4.6. Tabla de verdad de una puerta BUFFER.....	200
Tabla 4.7. Tabla de verdad de una puerta XOR de dos variables de entrada.....	208
Tabla 4.8. Tabla de verdad de una puerta XNOR de dos variables de entrada.....	218
Tabla 5.1. Asociación de minterms adyacentes y su simplificación.....	233
Tabla 5.2. Otra asociación de minterms adyacentes y su simplificación.....	234
Tabla 5.3. Ejemplos de representación algebraica y vectorial de términos canónicos de tres variables (c, b, a).....	238
Tabla 5.4. Ejemplos de índices de términos canónicos de tres variables (c, b, a).....	238

Tabla 5.5. Tabla de verdad de $f_1$ y $f_2$ .....	241
Tabla 5.6. Tabla de verdad de la función $f$ .....	244
Tabla 5.7. Tabla de adyacencias obtenidas por el método Q-M con coordenadas binarias.....	270
Tabla 5.8. Tabla de adyacencias obtenidas por el método Q-M con coordenadas decimales.....	272
Tabla 5.9. Tabla de adyacencias.....	278
Tabla 5.10. Tabla de conversión de cualquier puerta lógica a puertas NAND.....	284
Tabla 5.11. Tabla de conversión de cualquier puerta lógica a puertas NOR.....	285
Tabla 5.12. Tabla de verdad.....	287
Tabla 5.13. Tabla de adyacencias para minterms y términos indiferentes.....	290
Tabla 5.14. Tabla de adyacencias para maxterms y términos indiferentes.....	291
Tabla 5.15. Tabla de adyacencias para minterms y términos indiferentes.....	299
Tabla 6.1. Escalas de integración.....	320
Tabla 6.2. Consideraciones de diseño de las resistencias de una puerta NAND.....	355
Tabla 6.3. Características de la familia lógica TTL estándar.....	356
Tabla 6.4. Comparación de subfamilias TTL.....	376
Tabla 6.5. Tabla de verdad de la función $F$ obtenida al realizar el cableado lógico de las salidas FA y FB en colector abierto.....	379
Tabla 6.6. Tabla de verdad de la puerta NAND triestado.....	384
Tabla 6.7. Comparación de subfamilias CMOS con $V_{DD} = 5\text{ V}$ .....	400
Tabla 6.8. Tabla de verdad de la función NAND triestado.....	405
Tabla 6.9. Resumen de interfaces mediante adaptadores de niveles entre familias lógicas....	418
Tabla 6.10. Operadores predefinidos en VHDL, en el paquete Std_logic_1164.....	427
Tabla 7.1. Función de salida de un semisumador.....	436
Tabla 7.2. Función de salida de un sumador completo.....	439
Tabla 7.3. Función de salida de acarreo parcial.....	440
Tabla 7.4. Función de salida de un semirestador.....	458
Tabla 7.5. Función de conversión del circuito complementador a dos.....	464
Tabla 7.6. Función OR-EXCLUSIVA.....	464
Tabla 7.7. Función de conversión binario signo magnitud a complemento a dos.....	468
Tabla 7.8. Selección de operaciones en la ALU 74381-382.....	473
Tabla 7.9. Selección de operaciones en la ALU 74181 con entradas y salidas activas a nivel alto.....	477
Tabla 7.10. Función de un comparador binario.....	479
Tabla 7.11. Función de comparación.....	480
Tabla 8.1. Función de transferencia del DAC0800 unipolar.....	510
Tabla 8.2. Comparación entre la función de transferencia del DAC0800 en modo unipolar y bipolar.....	512
Tabla 8.3. Código Binario Natural aplicado a un cuantificador unipolar.....	519
Tabla 8.4. Código Complemento a dos aplicado sobre un cuantificador bipolar.....	520
Tabla 8.5. Código Binario Natural desplazado aplicado sobre un cuantificador bipolar.....	520
Tabla 8.6. ADC instantáneo de 3 bits.....	523

# SIMBOLOGÍA Y NOMENCLATURA

---

---

En el texto escrito a lo largo de este libro se han utilizado distintos tipos de letra dependiendo de qué se pretende expresar en cada caso.

Así, el texto plano tiene el aspecto de este mismo párrafo, mientras que los tipos especiales que se han utilizado en otros casos son los que se muestran como ejemplo en los siguientes términos.

- Analysis*** Se trata de un elemento de menú o de una opción de un cuadro de diálogo. Se representa en cursiva y negrita, con la letra correspondiente subrayada.
- Modelo** Se trata de un término o una definición que se desea resaltar dentro de un párrafo. Se representa en negrita.
- .MODEL** Cuando se hace referencia a un término o expresión que debe aparecer de forma literal en la aplicación o herramienta que se esté utilizando en ese momento, se expresará en el tipo de letra Courier-New.
- Duty cycle* Cuando sea necesario incluir un término en otro idioma, normalmente en inglés, éste se expresará en cursiva. Este estilo también se ha utilizado para resaltar alguna definición o concepto que se utiliza principalmente en electrónica.

A lo largo del libro también se han utilizado una serie de iconos representativos del contenido de la sección en la que se encuentran, de manera que resulte más fácil para el lector reconocer a primera vista determinadas secciones dentro del texto.

Estos iconos y su significado son los que se indican a continuación.



Deducción o introducción teórica del circuito que se analiza



Simulación del circuito con la herramienta *Electronics Workbench*



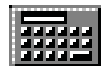
Simulación del circuito con la herramienta OrCAD (Digital)



Simulación del circuito con la herramienta OrCAD (*Simulate Demo*)



Simulación del circuito con la herramienta *VeriBest VHDL Simulator*



Simulación del circuito con la calculadora de Windows

A continuación, se describen los distintos parámetros que se han utilizado en este libro, así como el significado de cada uno de ellos.

$V_i$	Tensión de entrada a un sistema o a un dispositivo
$V_o$	Tensión de salida de un sistema o de un dispositivo
$V_{iH}$	Tensión de entrada a nivel alto
$V_{iL}$	Tensión de entrada a nivel bajo
$V_{oH}$	Tensión de salida a nivel alto
$V_{oL}$	Tensión de salida a nivel bajo
$I_i$	Corriente de entrada a un sistema o a un dispositivo
$I_o$	Corriente de salida de un sistema o de un dispositivo
$I_{iH}$	Corriente de entrada a nivel alto

$I_{iL}$	Corriente de entrada a nivel lógico bajo
$I_{oH}$	Corriente de salida a nivel lógico alto
$I_{oL}$	Corriente de salida a nivel bajo
$V_{iHmin}$	Tensión de entrada a nivel alto mínima que garantiza un nivel constante a la salida
$V_{iLmax}$	Tensión de entrada a nivel bajo máxima que garantiza un nivel constante a la salida
$V_{oHmin}$	Tensión de salida a nivel alto mínima
$V_{oLmax}$	Tensión de salida a nivel bajo máxima
$V_{DD}$	Tensión de alimentación positiva, también se representa como $V_{EE}$
$V_{NMH}$	Margen de ruido a nivel lógico alto
$V_{NML}$	Margen de ruido a nivel lógico bajo
$I_{iHmax}$	Corriente máxima de entrada a nivel alto
$I_{iLmax}$	Corriente máxima de entrada con nivel bajo
$I_{oHmax}$	Corriente máxima de salida con nivel alto
$I_{oLmax}$	Corriente máxima de salida con nivel bajo
$I_{oS}$	Intensidad de salida en cortocircuito
$I_{CC}$	Corriente media estática absorbida por un dispositivo
$I_{CCH}$	Corriente absorbida por un dispositivo en el nivel lógico alto
$I_{CCL}$	Corriente absorbida por un dispositivo en el nivel lógico bajo
$t_{pLH}$	Puesta en ON o tiempo de propagación en el flanco de subida de la señal de salida $V_o$
$t_{pD}$	Tiempo de propagación medio
$t_{pHL}$	Puesta en OFF o tiempo de propagación en el flanco de bajada de la señal de salida $V_o$
$t_r$	Tiempo de subida ( <i>rise time</i> ) de la señal de entrada
$t_f$	Tiempo de bajada ( <i>fall time</i> ) de la señal de entrada
$t_{TLH}$	Tiempo de subida de la señal de salida
$t_{THL}$	Tiempo de bajada de la señal de salida
$T_{DLH}$	Tiempo de retraso en el flanco de subida de la señal de salida
$T_{DHL}$	Tiempo de retraso en el flanco de bajada de la señal de salida

---

$f_{max}$	Frecuencia máxima de trabajo
$P_D$	Potencia media estática
$P_T$	Consumo dinámico de potencia
$P_D \cdot t_{pD}$	Producto potencia disipada-tiempo de propagación
$\theta_{JA}$	Resistencia térmica del dispositivo



# PROGRAMAS UTILIZADOS

---

---

## Electronics Workbench

*Electronics Workbench Demo* es la versión de demostración de Electronics Workbench 5, una herramienta para el análisis y la simulación de circuitos electrónicos analógicos y digitales asistida por ordenador. El sistema está integrado por tres módulos, un editor de esquemas con una interfaz gráfica muy fácil de utilizar, un simulador SPICE y un visualizador de señales eléctricas en pantalla.

Electronics Workbench permite implementar circuitos con gran facilidad, ya que está basado en el entorno gráfico Microsoft Windows, funcionando bajo Windows 9x, NT, 2xxx y mE.

La versión de demostración se diferencia de la versión profesional en los siguientes aspectos:

- Tan sólo permite realizar la simulación de los circuitos que acompañan a la misma.
- Permite realizar nuevos diseños de circuitos, pero no permite guardarlos.
- Tiene limitación de tiempo, transcurrido el mismo, el programa se cierra y es necesario arrancarlo de nuevo para poder continuar trabajando.

Se ha elegido este programa para la simulación de circuitos digitales por las siguientes razones:

- Se trata de una versión de educación, de distribución gratuita, y además se puede adquirir fácilmente.
- Se encuentra en lengua castellana.
- Contiene también una serie de ejemplos de circuitos analógicos y digitales.
- Es bastante popular en la enseñanza de electrónica en las Escuelas Técnicas, debido a la facilidad de manejo que presenta.

Para solicitar información adicional sobre la versión de educación o sobre la versión profesional y completa del programa, se puede dirigir a:

PRODEL, S.A.

Av. de Manoteras, 22

Edif. Alfa I, Ofic. 97

28050 Madrid

Tel. 91 383 83 35

Fax 91 383 04 90

E-mail: [prodel@ctv.es](mailto:prodel@ctv.es)

URL Internet: <http://www.prodel.es>

Para facilitar la evaluación y posterior posible adquisición de esta aplicación, se ha incluido en el CDRom#1 que acompaña a este libro la versión demo. Además, es posible actualizar las versiones de la misma en la siguiente dirección de Internet:

Electronics Workbench.

URL Internet: <http://www.electronicsworkbench.com/>

URL Internet: <http://www.interactiv.com/>

## OrCAD Demo v9

La aplicación *OrCAD Demo v9* es un paquete informático en el que se pueden diferenciar básicamente dos partes, una dedicada al diseño y simulación de circuitos electrónicos analógicos, digitales y mixtos, y otra que permite trabajar en el campo de la lógica programable.

Esta aplicación, un entorno gráfico que permite trabajar mediante ventanas y menús desplegables, es una versión limitada del programa OrCAD para profesionales que a modo de versión para educación se distribuye de forma gratuita.

Las principales diferencias entre esta versión de evaluación y la versión para profesionales, son las siguientes:

- En la versión de evaluación está limitado el número de componentes que se pueden incluir en un circuito que se desea simular. Así, un diseño no puede contener más de 30 instancias, no se pueden simular diseños que tengan más de 64 nodos, 10 transistores, 65 primitivas, 10 líneas de transmisión, 4 pares de líneas de transmisión acopladas, ficheros VHDL que tengan más de 200 líneas de código, etc. Estas limitaciones en la versión profesional son mucho menos restrictivas, siendo el número de componentes que se puede incluir en cada circuito bastante elevado.
- En la versión de evaluación existen determinados comandos y opciones que no se encuentran disponibles, tales como el tamaño de las páginas para diseñar los esquemas de los circuitos, la estructura jerárquica de los circuitos que permite la versión para profesionales, etc.
- El número de componentes (símbolos) disponibles en la versión de evaluación es bastante reducido frente al número de componentes disponibles en la versión profesional, si bien a modo de evaluación el número de los mismos es aceptable.

Se ha elegido este programa para la simulación de circuitos digitales, por las siguientes razones:

- Se trata de una aplicación que funciona bajo Windows, con todas las ventajas que esto conlleva: comunicación con todas las aplicaciones Windows (copiar, pegar, etc.), interfaz con el usuario más cómoda y universal, etc.
- Se trata de una versión de distribución gratuita que se puede adquirir fácilmente, o incluso copiar de otro usuario ya que es un programa shareware.
- La versión de CD-ROM contiene hipertextos con los manuales de la versión profesional y un tutorial que muestra el manejo de todas las herramientas.
- Contiene también una serie de ejemplos de circuitos tanto analógicos como digitales.
- Es bastante popular tanto en la enseñanza de Electrónica en las Escuelas Técnicas, como en la utilización del mismo para diseños de carácter profesional.

Para facilitar la evaluación y posterior posible adquisición de esta aplicación, se ha incluido en el CDROM#1 que acompaña a este libro. Además, es posible actualizar las versiones de la misma en la siguiente dirección de Internet:

SIDSA

URL Internet: <http://www.sidsapcb.com/>

Para adquirir el programa o solicitar cualquier tipo de información sobre el mismo puede dirigirse:

**Distribuidor en España:**

SIDSA  
Parque Tecnológico de Madrid  
Torres Quevedo, 1, 2ª planta  
28760 Tres Cantos - Madrid  
Tel. 91 803 50 52  
Fax. 91 803 95 57  
<http://www.sidsapcb.com/>

**Fabricante en USA:**

CADENCE Design Systems, Inc.  
13221 SW 68th Parkway, Suite 200  
Portland, OR 97223-8328  
USA  
[salesinfo@cadence.com](mailto:salesinfo@cadence.com)  
<http://www.pcb.cadence.com>  
<http://www.pspice.com/>

## VeriBest VHDL

La aplicación *VeriBest VHDL V99.0* permite realizar el diseño y simulación de sistemas en el campo de la lógica programable, a partir de la especificación de las características particulares que definen a la familia lógica y que la diferencia de las demás, como son principalmente: sus retardos, los distintos niveles lógicos que sean posibles adoptar y las salidas especiales que pueden presentar.

*VeriBest VHDL V99.0* está basada en un entorno gráfico que permite trabajar mediante ventanas y menús desplegados, es una versión limitada del programa VeriBest para profesionales que a modo de versión para educación se distribuye de forma gratuita.

La versión de evaluación se diferencia de la versión profesional en los siguientes aspectos:

- El número de componentes (descripciones) que es posible incluir en la simulación está limitado.
- Existen determinados comandos y opciones que no se encuentran disponibles en esta versión.

Se ha elegido este programa para la simulación de circuitos digitales por las siguientes razones:

- Se trata de una versión de educación, de distribución gratuita y además se puede adquirir fácilmente.
- Su entorno de trabajo, basado en ventanas, hace que la complejidad de su manejo sea más reducida.
- Contiene también una serie de ejemplos de sistemas que se pueden simular.

- Es bastante popular en la enseñanza de VHDL debido a su facilidad de instalación y manejo.

Para adquirir el programa o solicitar cualquier tipo de información sobre el mismo puede dirigirse:

**En España:**

Mentor Graphics España SL  
Parque Empresarial San Fernando  
Edificio Francia P.B.  
28830 San Fernando de Henares  
Tel.: 91 677 57 85  
Fax.: 91 677 58 79  
E-mail: [sales\\_spain@mentor.com](mailto:sales_spain@mentor.com)

**En USA (Oregón):**

Mentor Graphics Inc.  
8005 SW Doeckman road  
Willsonville, OR 97090 - USA  
Tel.: 1503 685 7000  
Fax: 1503 685 1204

Cabe señalar que no se incluye el Kit de instalación de la herramienta *VeriBest V99* en el CD-ROM que se incluyen con el libro, tal como inicialmente se había planteado, debido a que tras la compra del mismo por la compañía Mentor, ha cambiado el nombre del producto y su estrategia de comercialización. Para poder disponer de la nueva aplicación es necesario descargarla accediendo a la página de Internet de la compañía Mentor Graphics:

URL Internet: <http://www.mentor.com/>

Por otra parte, no está comprobada la funcionalidad de los ficheros que se han desarrollado con la aplicación *VeriBest V99* en la nueva aplicación que actualmente se comercializa, denominada *ModelSim*.

## Herramienta de Hojas de Datos de Fairchild

Además de las herramientas de simulación que se han citado anteriormente, en el CDROM#2 que acompaña a este libro, y por cortesía de Fairchild Semiconductor, se ha incluido una aplicación en la que se muestran las hojas de características de circuitos integrados lógicos de dicha casa comercial.

A través de esta herramienta se puede acceder a las características de los distintos componentes electrónicos que se han incluido, entre los que cabe señalar: componentes analógicos, mediante la opción: *Analog & Mixed Signal*, componentes digitales, mediante la opción: *Logic*, interfaces, mediante la opción: *Interface*, memorias no volátiles, mediante la opción: *Non-Volatile Memory*.

Para cada uno de estos componentes se describe el nombre de los mismos, el fabricante, la fecha de creación, una descripción general, sus características más importantes, su código, el diagrama de conexionado, los valores límite de trabajo, las características estáticas y dinámicas, las gráficas de características típicas de funcionamiento, las ondas de las señales en conmutación, las dimensiones físicas, etc.

Se ha elegido esta herramienta para la descripción de hojas de datos por las siguientes razones:

- Se trata de una versión de distribución gratuita y además se puede adquirir fácilmente.
- Su entorno de trabajo, basado en ventanas, hace que la complejidad de su manejo sea más reducida.
- Se trata de una de las casas comerciales más importantes del sector.

Para solicitar cualquier información sobre esta herramienta puede dirigirse:

**En España:**

EuroInger, SL.  
C/ Almendralejos, 4  
Fuente el Saz, 28140 Madrid  
Tel.: 91 620 09 21  
Fax.: 91 620 06 12  
Mov.: 607 52 83 74  
E-mail: [rruano@euroinger.com](mailto:rruano@euroinger.com)

**En USA (Texas):**

Customer Response Center  
Fairchild Semiconductor  
222 West Las Colinas Blvd., Ste. 380  
Irving, TX 75039 - USA  
Tel.: 888-522-5372  
Fax: 972-910-8036

Para facilitar la evaluación y posterior posible adquisición de esta aplicación, se ha incluido en el CDROM#2 que acompaña a este libro. Además, es posible ampliar información sobre de la misma en la siguiente dirección de Internet:

Fairchild Semiconductor

URL Internet: <http://www.fairchildsemi.com/>

## CAPÍTULO 1

# FUNDAMENTOS GENERALES DE LA ELECTRÓNICA DIGITAL

---

---

### *Objetivos:*

- Diferenciar entre sistemas digitales y analógicos. Conocer sus ventajas e inconvenientes.
- Saber representar información numérica en los sistemas de numeración más utilizados habitualmente (binario, octal, hexadecimal, etc.) y realizar cambios de base.
- Manejar representaciones numéricas en coma fija y en coma flotante.

*Contenido:* En este capítulo se exponen: las diferencias entre los sistemas analógico y digital; la representación de la información mediante el conocimiento de los sistemas de numeración.

*Simulación:* Se utiliza la calculadora de Windows en modo “Científica” para obtener las representaciones de números enteros con signo, según el convenio del complemento a dos, en las bases: binaria, octal, hexadecimal y decimal.

## 1.1 SISTEMAS ANALÓGICOS Y DIGITALES



Los circuitos electrónicos se dividen, según la naturaleza de los valores que toman las señales o magnitudes que intervienen en el sistema, en dos categorías: analógicos y digitales.

La electrónica analógica utiliza magnitudes con valores continuos, mientras que la electrónica digital emplea magnitudes con valores discretos.

En este apartado se analiza esta clasificación y se establecen las ventajas e inconvenientes entre los sistemas que tratan la información de forma analógica o digital.

Una **señal analógica** es aquella cuya magnitud, en cada instante de tiempo, puede tomar cualquiera de los infinitos valores del rango donde esté definida, pudiendo cambiar de valor en cantidades arbitrariamente pequeñas. La mayoría de las magnitudes que se pueden medir cuantitativamente se presentan en la naturaleza en forma analógica. Ejemplos de magnitudes analógicas son: presión, humedad, temperatura, tensión eléctrica, etc.

En el **Apéndice A** se describen y se simulan los principales generadores y fuentes de señales analógicas.

En la Figura 1.1 se muestra un ejemplo de señal analógica. En este caso la magnitud representada es la humedad relativa del aire en función del tiempo. Se aprecia cómo, a lo largo de un periodo de tiempo, la humedad relativa varía de forma continua en un rango de valores, es decir, entre dos puntos cualesquiera, como por ejemplo 25% y 50%, no lo hace de forma instantánea, sino que va tomando los infinitos valores que hay en ese rango o intervalo.

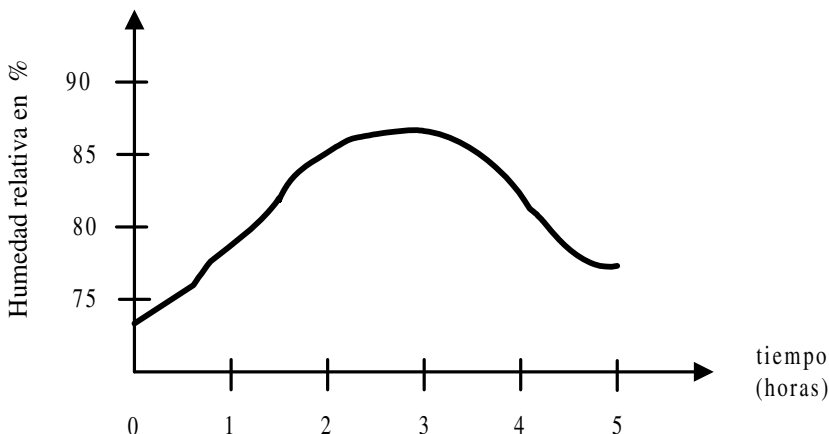


Figura 1.1. Representación de una magnitud analógica

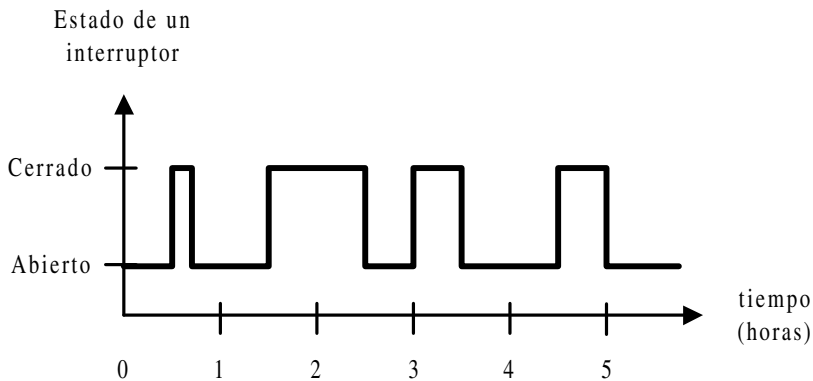


Las **señales digitales** son aquellas cuya magnitud, en cada instante de tiempo, sólo pueden tomar un valor de entre un conjunto finito de  $q$  valores discretos. En el paso de un valor a otro se produce una discontinuidad al no existir valores intermedios.

Si  $q = 2$  la magnitud presenta dos estados bien diferenciados: cerrado o abierto, alto (*High*) y bajo (*Low*), nivel de tensión alto ( $V_H$ ) o bajo ( $V_L$ ), valor numérico 1 o 0, etc. A las señales digitales con dos estados se las denominan binarias, y constituyen la base de la electrónica digital.

En el **Apéndice A** se describen y simulan los principales generadores y fuentes de señales digitales.

En la Figura 1.2 se muestra un ejemplo de señal digital binaria, donde la magnitud representa el estado de un interruptor utilizado para activar/desactivar el alumbrado de una habitación en función del tiempo. Se aprecia que dicho estado sólo puede tomar dos valores posibles: abierto o cerrado, por lo que habrá instantes en los que se producirá una discontinuidad al pasar de un estado a otro, por no existir valores intermedios entre el estado abierto y el estado cerrado. Por tanto, las señales digitales vienen caracterizadas por un número finito de posibles valores y las discontinuidades asociadas a dichos valores.



*Figura 1.2. Representación de una magnitud digital*

Un **sistema** es un conjunto de elementos con alguna característica en común. A los elementos de un sistema que asimismo tienen estructura de sistema, se les denomina **subsistema**. De lo anterior se deduce que según la escala que se emplee en la observación de un elemento, éste puede ser considerado como subsistema o como sistema. Por ejemplo, en el universo, los astros que orbitan alrededor del sol (incluido éste), son elementos que forman el denominado Sistema Solar. Si se aumenta la escala de observación, se verá que, por ejemplo, los planetas tienen estructura de sistema ya que están formados por elementos más simples. Por lo tanto, los planetas pueden ser considerados subsistemas del Sistema Solar.

Los sistemas que preferentemente se van a tratar en esta publicación, son los de naturaleza eléctrica, dentro de los cuales se encuentran los electrónicos y en especial, los sistemas digitales. Un ejemplo de sistema, sería el **circuito** o conjunto de mayor magnitud de todos los que se describan, por ejemplo: un reloj de sobremesa digital. A su vez estos sistemas estarán compuestos de subsistemas o **dispositivos** que realizan una operación o función electrónica por sí mismos, como por ejemplo: bloque de visualización, contadores, generador de la señal de reloj, etc. Asimismo, los dispositivos están formados por elementos más simples denominados **componentes**, siendo éstos los de menor magnitud de todos los que se describan, como por ejemplo: una puerta lógica, un display, un pulsador, un zumbador, etc.

Un **sistema analógico** es aquel en el que sus señales son de tipo analógico. Sus componentes suelen trabajar en su zona lineal, en la que la relación que existe entre las señales de entrada y salida es constante, denominada zona de trabajo. Dichas señales pueden tomar cualquier valor dentro de unos límites determinados.

### EJEMPLO:

Un ejemplo de sistema analógico es el que se muestra en la Figura 1.3. Se trata de un medidor analógico de la velocidad de un motor. El sistema está compuesto de un dispositivo, denominado dinamo, que transforma linealmente la magnitud física velocidad,  $v$ , en magnitud eléctrica tensión,  $V_S = K \cdot v$ , y de un elemento para visualizar la medida de velocidad, pudiendo ser éste por ejemplo un voltímetro analógico, conectado a la salida de la dinamo. Para que la medida sea directa, la carátula del voltímetro se debe tabular en velocidad.

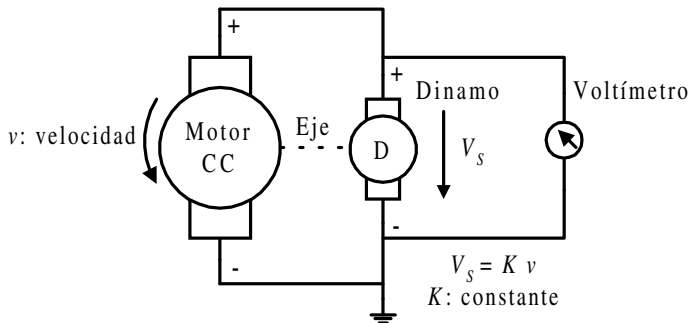


Figura 1.3. Ejemplo de sistema analógico

Un **sistema digital** es aquel en el que sus señales son de tipo digital. Sus componentes trabajan en las zonas de saturación (sus señales de salidas no tienen una relación lineal respecto de sus entradas). Las señales de estos sistemas suelen ser próximas a los potenciales de la alimentación, presentando dos estados diferenciados, correspondiendo cada uno de ellos a un nivel o valor de la magnitud binaria.

### Ejemplo:

Un sistema digital es el que se muestra en la Figura 1.4, se trata de un cronómetro digital compuesto por:

- Un reloj (onda cuadrada que corresponde a una señal digital binaria) de periodo igual a una centésima de segundo.
- Un bloque contador y un decodificador (elementos que se estudiarán en capítulos posteriores, aquí basta con conocer sus funciones), que cuenta los ciclos de reloj incrementándose cada centésima de segundo y proporcionando las señales digitales necesarias para el bloque de visualización.
- Un bloque de control de mandos que gobierna mediante un teclado la puesta a cero, bloqueo e inicio del cronómetro.
- Un bloque de visualización que proporciona numéricamente el periodo transcurrido entre dos intervalos de tiempo con una precisión de una centésima de segundo.

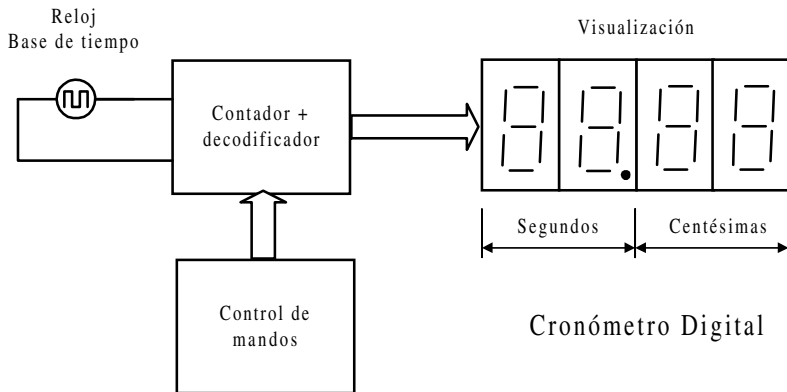


Figura 1.4. Ejemplo de sistema digital

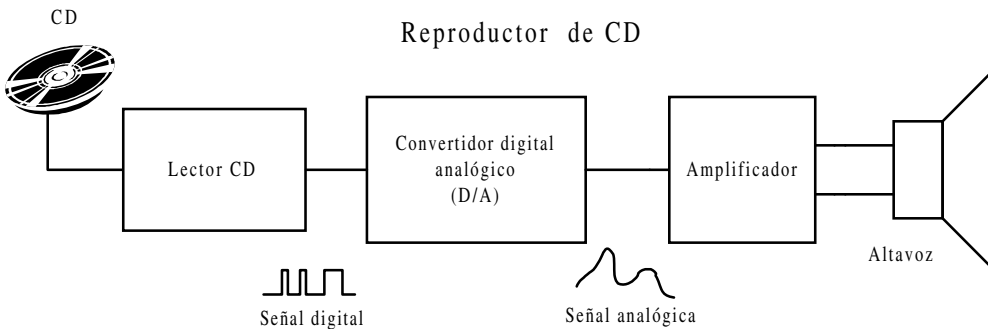
Un **sistema analógico-digital** es aquel en el que intervienen tanto señales analógicas como señales digitales; es decir, está compuesto de subsistemas analógicos y subsistemas digitales.

### Ejemplo:

Un ejemplo de sistema analógico-digital es el que se muestra en la Figura 1.5, que corresponde a un reproductor de *compact disc* (CD), en el que la información musical grabada en formato digital es recuperada por el lector de CD, compuesto de un sistema óptico con diodo láser que lee los datos digitales del disco cuando éste gira. Un convertidor digital-analógico (D/A) transforma

las señales digitales en analógicas, necesarias para excitar el altavoz, una vez hayan sido amplificadas en el bloque amplificador.

Se puede observar cómo el sistema analógico-digital de la Figura 1.5 está formado por un subsistema digital compuesto por el lector de CD y un subsistema analógico compuesto por el amplificador y el altavoz. El convertidor digital-analógico (D/A) permite la transferencia de información entre ellos. En la grabación de información en el CD se realizó el proceso inverso, utilizándose un convertidor analógico-digital (A/D).



*Figura 1.5. Ejemplo de sistema analógico-digital*

Un mismo diseño de sistema electrónico puede tener solución utilizando tanto técnicas digitales como analógicas, aunque se debe tener en cuenta que los sistemas digitales presentan ciertas **ventajas** con respecto a los analógicos, principalmente en el procesamiento y transmisión de la información. Estas ventajas son:

- El número de operaciones básicas con variables digitales es muy reducido frente al número de operaciones con variables analógicas, resultando por ello un número pequeño de circuitos básicos a utilizar que se repiten muchas veces, hasta formar un sistema digital.
- Mayor precisión y versatilidad en el procesamiento de la información.
- Menor sensibilidad al ruido y alta fiabilidad, al haber sido realizado el diseño con componentes que trabajan en conmutación.
- Alta capacidad de almacenamiento de información, pudiendo realizar accesos directos con tiempos reducidos (del orden de nanosegundos).
- Posibilidad de detección y corrección de errores en la transmisión de información.

## 1.2 REPRESENTACIÓN DE LA INFORMACIÓN



Los sistemas digitales tratan información binaria, siendo importante conocer los fundamentos de los sistemas de numeración y en especial en base dos. Otro tipo de información tratada en esta sección son los números binarios con signo y los códigos binarios.

### 1.2.1 Sistemas de numeración

La mayoría de los sistemas de numeración utilizados en la actualidad son del tipo **polinomial**. En este sistema un número viene definido por una cadena de dígitos, estando afectados cada uno de ellos por un factor de escala que depende de la posición que ocupa en la cadena. Un sistema de numeración polinomial tiene las siguientes características:

- Un número o cantidad se representa por una sucesión ordenada de símbolos, llamados **dígitos o cifras**, situados a izquierda y derecha de un punto de referencia (la coma en los países latinos y el punto en los anglosajones). Por ejemplo: 145,57 representa a un número o cantidad.
- Cada uno de estos dígitos tiene un valor fijo y diferente de los demás.
- El número de posibles dígitos distintos a utilizar en un determinado sistema de numeración constituye su **base**. Así, el sistema actualmente más empleado, llamado sistema decimal hindú-arábigo (tomado de los hindúes por los árabes, probablemente en el siglo VIII), tiene diez dígitos o guarismos. En la **Tabla 1.1** se muestran algunos ejemplos de los sistemas de numeración más empleados.

*Tabla 1.1. Ejemplos de sistemas de numeración*

SISTEMA	BASE	DÍGITOS
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8 y 9
Binario	2	0 y 1
Octal	8	0, 1, 2, 3, 4, 5, 6 y 7
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F

- El **valor numérico** expresado por una determinada combinación de dígitos, en una base de numeración dada, depende de dos factores:
  - ◆ del valor de los dígitos o cifras que lo componen, y,
  - ◆ de la posición de cada uno de ellos respecto del punto de referencia.

- Cada posición del dígito tiene un valor intrínseco, denominado **peso**, que aumenta de derecha a izquierda según potencias sucesivas de la base  $b$  del sistema de numeración empleado. Dichas potencias corresponden a la posición  $i$  del dígito dentro de la sucesión, siendo su valor cero aquella posición situada a la izquierda del punto de referencia, tomando valores enteros crecientes a medida que el desplazamiento es en posiciones situadas a la izquierda o valores enteros decrecientes si el desplazamiento es a la derecha. En este caso, el valor del peso es  $b^i$ .

Según lo anteriormente indicado, la representación de un número  $N$ , expresado en base  $b$ , mediante una sucesión de dígitos  $a_i$ , siendo  $p$  enteros y  $q$  fraccionarios, será la que se muestra en la expresión [1.1],

$$N_{(b)} = a_{p-1}a_{p-2} \dots a_i \dots a_2a_1a_0, a_{-1}a_{-2} \dots a_{-q} \quad (b) \quad [1.1]$$

Número	$a_{p-1}$	$a_{p-2}$	....	$a_i$	....	$a_2$	$a_1$	$a_0$	,	$a_{-1}$	$a_{-2}$	....	$a_{-q}$
Posición	$p-1$	$p-2$	....	$i$	....	2	1	0		-1	-2	....	$-q$
Peso	$b^{p-1}$	$b^{p-2}$	....	$b^i$	....	$b^2$	$b^1$	$b^0$		$b^{-1}$	$b^{-2}$	....	$b^{-q}$

siendo:

$$p > i \geq -q,$$

$$b > 1,$$

$$b > a_i \geq 0,$$

$a_i$  valor decimal del dígito o guarismo situado en la posición  $i$ .

Cualquier número real decimal  $N$  con  $p$  dígitos enteros y  $q$  fraccionarios, expresado en base  $b$ , adopta el desarrollo polinomial representado en la expresión [1.2], denominada **ecuación general decimal de los sistemas de numeración**.

$$N_{(10)} = a_{p-1}b^{p-1} + a_{p-2}b^{p-2} + \dots + a_i b^i + \dots + a_1 b^1 + a_0 b^0 + a_{-1} b^{-1} + \dots + a_{-q} b^{-q} \quad [1.2]$$

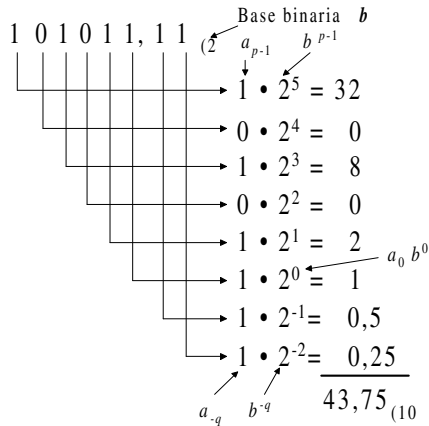
## PROBLEMA RESUELTO 1-1

Calcular el valor decimal del número binario  $N_{(2)} = 101011,11_{(2)}$

### Solución:

El número del enunciado es de base dos o binaria ( $b = 2$ ), con seis cifras enteras ( $p = 6$ ) y dos fraccionarias ( $q = 2$ ).

$$N_{(10)} = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} = 43,75_{(10)}$$



### PROBLEMA RESUELTO 1-2

Calcular el valor decimal del número octal  $N_{(8)} = 547_{(8)}$

**Solución:**

$$N_{(10)} = 5 \cdot 8^2 + 4 \cdot 8^1 + 7 \cdot 8^0 = 320 + 32 + 7 = 359_{(10)}$$

### PROBLEMA RESUELTO 1-3

Calcular el valor decimal del número hexadecimal  $N_{(16)} = 27AF,8_{(16)}$

**Solución:**

$$\begin{aligned} N_{(10)} &= 2 \cdot 16^3 + 7 \cdot 16^2 + 10 \cdot 16^1 + 15 \cdot 16^0 + 8 \cdot 16^{-1} = \\ &= 8192 + 1792 + 160 + 15 + 0,5 = 10159,5_{(10)} \end{aligned}$$

#### 1.2.1.1 ELECCIÓN DEL SISTEMA DE NUMERACIÓN

Cuanto mayor sea la base del sistema, mayor será el número de símbolos o guarismos diferentes que se pueden utilizar (tantos como el valor numérico de la base) y menor el número de cifras necesarias para representar una cantidad. Así, la mayor cantidad que se puede expresar con  $n$  dígitos, en una base  $b$ , será la indicada en la expresión [1.3].

$$N_{\max(b)} = b^n - 1 \quad [1.3]$$

Sin embargo, cuanto menor es la base más simples son las reglas que rigen los cálculos aritméticos y menos símbolos o niveles diferentes de una magnitud eléctrica son necesarios detectar en los circuitos electrónicos, consiguiéndose que éstos sean también más simples. Además, para una determinada tensión de alimentación, cuantos menos niveles lógicos se precisen habrá una mayor separación de tensión entre ellos, pudiendo el sistema admitir mayores fluctuaciones de tensión en un nivel lógico sin que lleguen a producirse errores en el sistema al adquirir otro nivel lógico adyacente.

Otra consideración es que la tensión es una magnitud de naturaleza analógica, por lo que un cambio de nivel lógico  $V_n$  a  $V_{n+2}$  necesariamente debe tomar valores intermedios como  $V_{n+1}$ . Si esta tensión fuera un nivel lógico válido, el sistema presentaría errores, ya que no podría realizar transiciones directamente de  $V_n$  a  $V_{n+2}$ .

Desde el punto de vista económico, el mejor sistema de numeración será el que menos componentes precise para la representación de un número. Si se necesitara representar  $d$  dígitos distintos en un sistema de base  $b$  mediante conmutadores, sería preciso utilizar  $n$  conmutadores de  $b$  posiciones, cumpliéndose la expresión [1.4].

$$d = b^n \quad [1.4]$$

Considerando que el coste del sistema  $P$  es proporcional al número de conmutadores  $n$  y al número de posiciones  $b$  de cada conmutador, se tiene que:

$$P = K \cdot b \cdot n \quad [1.5]$$

De las expresiones [1.4] y [1.5] se obtiene la expresión [1.6].

$$P = K \cdot b \cdot \frac{\ln d}{\ln b} \quad [1.6]$$

Para calcular la base idónea que haga que el sistema tenga un coste mínimo, se halla el valor mínimo de la función  $P$ . Para ello se calcula la derivada de la expresión [1.6] con respecto a  $b$  y se iguala a cero, despejando posteriormente el valor de  $b$ , como se muestra en la expresión [1.7].

$$\begin{aligned} \frac{dP}{db} &= \frac{d}{db} \left( K \cdot b \cdot \frac{\ln d}{\ln b} \right) = K \left( \frac{\ln d}{\ln b} - \frac{\ln d}{(\ln b)^2} \right) = 0 \\ \ln d \cdot \ln b - \ln d &= \ln d \cdot (\ln b - 1) = 0 \\ \ln b - 1 &= 0 \Rightarrow \ln b = 1 \\ b &= e = 2,718 \end{aligned} \quad [1.7]$$



Por tanto, el sistema de numeración más económico es aquel cuya base es dos o tres. Teniendo en cuenta todas las consideraciones apuntadas anteriormente se concluye que el **sistema binario** es el más idóneo como sistema de numeración de circuitos digitales.

### 1.2.1.2 CONSIDERACIONES IMPORTANTES SOBRE SISTEMAS POLINOMIALES

Como complemento al estudio de los sistemas polinomiales, se indican las siguientes consideraciones importantes:

- La base  $b$  suele ser un número natural, pero también son interesantes, aunque poco empleados, los sistemas de representación con **base negativa**, ya que permite representar todas las cantidades positivas y negativas sin tener que añadir el signo al número.

#### Ejemplo:

$$134_{(-10)} = 1 \cdot (-10)^2 + 3 \cdot (-10)^1 + 4 \cdot (-10)^0 = 100 + (-30) + 4 = 74_{(10)}$$

$$86_{(-10)} = 8 \cdot (-10)^1 + 6 \cdot (-10)^0 = (-80) + 6 = -74_{(10)}$$

- Se suele imponer la restricción de que todos los dígitos  $a_i$  sean positivos y menores que la base, lo que hace que:

$$0 \leq a_i < b, \quad \text{o} \quad a_i = b-1, b-2, \dots, 2, 1, 0$$

siendo interesante también utilizar los dígitos siguientes:

$$-b \leq a_i < b, \quad \text{o} \quad a_i = b-1, b-2, \dots, 1, 0, -1, \dots, 2-b, 1-b$$

lo que permite representar todas las cantidades positivas y negativas sin añadir signo al número. La representación no es única como se aprecia en el ejemplo siguiente.

#### Ejemplos:

$$\bar{1}10_{(2)} = (-4) + 2 + 0 = -2_{(10)}$$

$$010_{(2)} = 0 + 2 + 0 = 2_{(10)}$$

$$0\bar{1}0_{(2)} = 0 + (-2) + 0 = -2_{(10)}$$

$$\bar{1}\bar{1}0_{(2)} = 4 + (-2) + 0 = 2_{(10)}$$

- Una propiedad importante de los sistemas de representación con base es la siguiente:

Si se tienen dos bases  $\alpha$ , y  $\beta$ , tales que  $\alpha = \beta^k$ , se cumple que:

$$\dots a_2 a_1 a_0, a_{-1} a_{-2} \dots (\beta = \dots c_2 c_1 c_0, c_{-1} c_{-2} \dots)_\alpha$$

donde:

$$c_j = a_{kj+k-1} \dots a_{kj+1} a_{kj} (\alpha)$$

esto es, los dígitos de la base  $\alpha = \beta^k$  se obtienen agrupando los dígitos de la base  $\beta$  en grupos de longitud  $k$ . En los sistemas binario, octal y hexadecimal se cumple esta regla. Se pasa del binario al octal, agrupando los bits de tres en tres, y de binario al hexadecimal, agrupando de cuatro en cuatro.

### Ejemplo:

$$1001010101_{(2)} = 1125_{(8)} = 255_{(16)}$$

Un número racional puede tener una representación exacta en una base y una representación periódica, en otra base.

### Ejemplo:

$$\frac{1}{5_{(10)}} = 0,2_{(10)}$$

siendo en la base binaria:

$$\frac{1}{5_{(10)}} = 0,00\overline{11}_{(2)}$$

esto plantea un problema, ya que, al tener que almacenar este número en una cadena finita de  $n$  bits, hay que truncar el periodo, resultando una representación no exacta, como puede verse en el siguiente ejemplo:

$$\frac{1}{5_{(10)}} = 0,00110011_{(2)}$$

Esto justifica los errores que se producen en los resultados de los sistemas digitales, al convertir decimales a fracciones binarias, puesto que se necesita para representar ciertos valores, un número con infinitos dígitos, y éstos sólo pueden ser almacenados en un número finito de bits, por lo que su representación no será exacta. El resultado del ejemplo anterior, operando en el sistema binario, dará un cierto error (por truncar el periodo) que en el sistema decimal no se produciría.

### 1.2.1.3 CONVERSIÓN ENTRE BASES

La conversión entre bases permite representar el mismo número en bases distintas, es decir, dado un número  $N$  en base  $b_1$  expresar dicho número  $N$  en base  $b_2$ .

Se pueden realizar diferentes estrategias para efectuar un cambio de base, aunque es recomendable seguir los pasos indicados en la Figura 1.6, siendo éstos:

- convertir en un primer paso el número  $N_{(b_1)}$  de base  $b_1$  a decimal  $N_{(10)}$ ,
- y en un segundo paso convertir el número decimal a base  $b_2$ , obteniéndose el número  $N_{(b_2)}$ .

Esto permite realizar todas las operaciones en base 10, resultando más sencilla la conversión por ser la base más familiar.

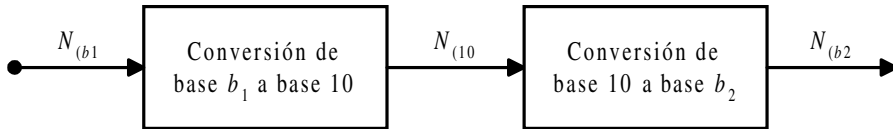


Figura 1.6. Pasos recomendables para la conversión entre bases

El primer paso de **conversión de  $N_{(b_1)}$  a  $N_{(10)}$**  se realiza, de forma sencilla, aplicando la expresión [1.2], descrita anteriormente. Esta ecuación se denomina ecuación general de los sistemas de numeración, y viene dada por la expresión [1.8].

$$N_{(10)} = a_{p-1}b^{p-1} + a_{p-2}b^{p-2} + \dots + a_i b^i + \dots + a_1 b^1 + a_0 b^0 + a_{-1} b^{-1} + \dots + a_{-q} b^{-q} \quad [1.8]$$

Este número  $N_{(10)}$ , en base 10, se obtendrá mediante la expresión [1.9].

$$N_{(10)} = d_{m-1}d_{m-2} \dots d_1 d_0, d_{-1} \dots d_{-n} \quad (10) = a_{p-1} \cdot b_1^{p-1} + a_{p-2} \cdot b_1^{p-2} + \dots \quad [1.9]$$

$$\dots + a_1 \cdot b_1^1 + a_0 \cdot b_1^0 + a_{-1} \cdot b_1^{-1} + \dots + a_{-q} \cdot b_1^{-q}$$

En el segundo paso de **conversión de  $N_{(10)}$  a  $N_{(b_2)}$**  se trata por separado la parte entera y la parte fraccionaria del número en base 10.

#### 1) PARTE ENTERA:

Considerando un número entero en base 10:

$$N_{(10)} = d_{m-1}d_{m-2} \dots d_1 d_0 \quad (10) \quad [1.10]$$

Su equivalente en base  $b_2$  es:

$$N_{(b_2)} = c_{k-1}c_{k-2} \dots c_1c_0 \quad (b_2) \quad [1.11]$$

La relación entre ambos es:

$$N_{(10)} = d_{m-1}d_{m-2} \dots d_1d_0 \quad (10) = c_{k-1} \cdot b_2^{k-1} + c_{k-2} \cdot b_2^{k-2} + \dots + c_1 \cdot b_2^1 + c_0 \cdot b_2^0 \quad [1.12]$$

siendo necesario calcular los coeficientes  $c_i$ , es decir,  $(c_{k-1} c_{k-2} \dots c_1 c_0)$  que son los dígitos desconocidos del número expresado en la nueva base  $b_2$ . Para ello, en la expresión [1.12], se dividen sus dos miembros entre la base  $b_2$ , con la aritmética de base 10.

$$\frac{N_{(10)}}{b_2} = c_{k-1} \cdot b_2^{k-2} + c_{k-2} \cdot b_2^{k-3} + \dots + c_1 + \frac{c_0}{b_2} = Q_0 + \frac{c_0}{b_2} \quad [1.13]$$

Como  $c_i < b_2$ ,  $c_0$  será igual al resto de la división anterior. El dígito  $c_1$  es el resto obtenido al dividir el cociente  $Q_0$  entre  $b_2$ .

$$\frac{Q_0}{b_2} \quad (10) = c_{k-1} \cdot b_2^{k-3} + c_{k-2} \cdot b_2^{k-4} + \dots + \frac{c_1}{b_2} = Q_1 + \frac{c_1}{b_2} \quad [1.14]$$

Reiterando este procedimiento se obtienen los demás coeficientes. El proceso finaliza con el cálculo del coeficiente  $c_{k-1}$ , lo que se cumple cuando se obtenga un cociente menor que  $b_2$ , siendo  $c_{k-1}$  precisamente este último cociente. La Tabla 1.2 resume este proceso, consistente en realizar divisiones sucesivas de  $N_{(10)}$  y la base  $b_2$ , hasta obtener un cociente  $Q_i$  menor que la base  $b_2$ . Los coeficientes del número expresado en la base  $b_2$  ( $N_{b_2}$ ) están formados por este último cociente (dígito con mayor peso  $c_{k-1}$ ) y los restos de las divisiones realizadas, tomados en orden inverso, como se indica en la Figura 1.7.

Tabla 1.2. Proceso de conversión de base 10 a base  $b_2$  en parte entera de un número

Operación	Cociente $Q_i$	Resto $c_i$
$N_{(10)} : b_2$	$Q_0$	$c_0$
$Q_0 : b_2$	$Q_1$	$c_1$
$Q_1 : b_2$	$Q_2$	$c_2$
$\vdots$	$\vdots$	$\vdots$
$Q_{k-3} : b_2$	$Q_{k-2} < b_2$	$c_{k-2}$
$Q_{k-2} : b_2$	$Q_{k-1} = 0$	$Q_{k-2} = c_{k-1}$

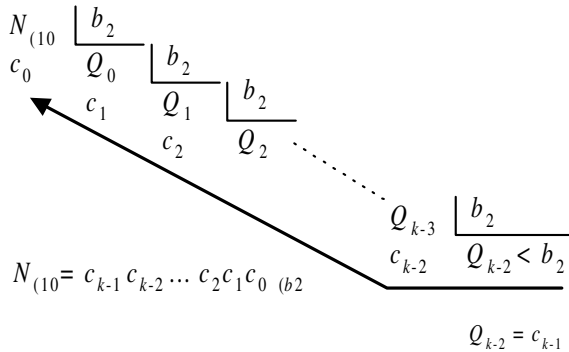


Figura 1.7. Conversión de base 10 a base  $b_2$  de la parte entera de un número

**PROBLEMA RESUELTO 1-4**

Convertir de **decimal a binario** el número  $324_{(10)}$

**Solución:**

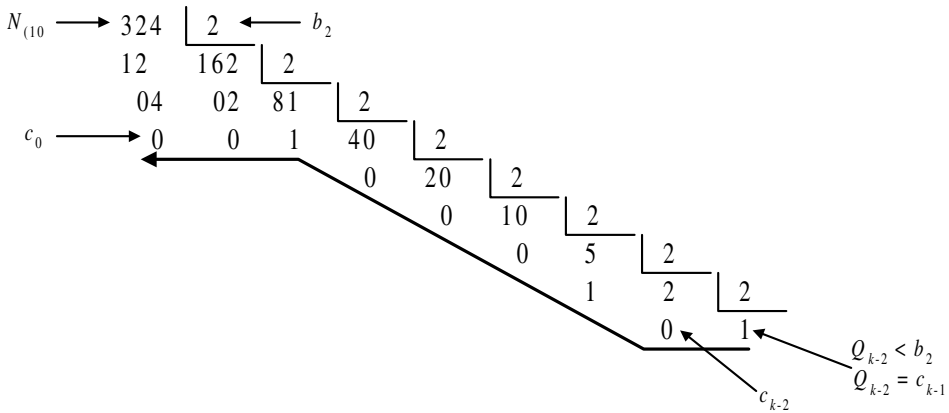
Aplicando el proceso indicado anteriormente en la Tabla 1.2 y considerando que  $N_{(10)} = 324_{(10)}$ , y  $b_2 = 2$ , se obtiene dicha conversión, tal como se muestra en la Tabla 1.3.

Tabla 1.3. Conversión del número  $324_{(10)}$  de decimal a binario

Operación	Cociente $Q_i$	Resto $c_i$
$324 : 2$	162	0
$162 : 2$	81	0
$81 : 2$	40	1
$40 : 2$	20	0
$20 : 2$	10	0
$10 : 2$	5	0
$5 : 2$	2	1
$2 : 2$	$1 < 2$	0
$1 : 2$	0	1

El resultado obtenido en la Tabla 1.3 es:  $324_{(10)} = 101000100_{(2)}$

El mismo proceso de conversión pero utilizando la representación de las divisiones sucesivas de la Figura 1.7, es el que se muestra a continuación:



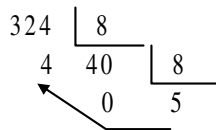
El resultado obtenido es:  $324_{(10)} = 101000100_{(2)}$

Se aprecia que el proceso es el mismo pero con otra representación, obteniéndose el mismo resultado en ambos casos.

**PROBLEMA RESUELTO 1-5**

Convertir de **decimal a octal** el número  $324_{(10)}$

**Solución:**



El resultado **obtenido** es:  $324_{(10)} = 504_{(8)}$

**PROBLEMA RESUELTO 1-6**

Convertir de **decimal a hexadecimal** el número  $324_{(10)}$

**Solución:**

$$\begin{array}{r}
 324 \quad | \quad 16 \\
 4 \quad | \quad 20 \quad | \quad 16 \\
 \quad \quad | \quad 4 \quad | \quad 1
 \end{array}$$

El resultado **obtenido** es:  $324_{(10)} = 144_{(16)}$

**2) PARTE FRACCIONARIA:**

Considerando un número **real** en base 10 con sólo parte fraccionaria.

$$N_{(10)} = d_{-1}d_{-2}\dots d_{-n} \quad (10) \quad [1.15]$$

Su equivalente en base  $b_2$  es:

$$N_{(b_2)} = c_{-1}c_{-2}\dots c_{-s} \quad (b_2) \quad [1.16]$$

La relación entre ambos es:

$$N_{(10)} = d_{-1}d_{-2}\dots d_{-n} \quad (10) = c_{-1} \cdot b_2^{-1} + c_{-2} \cdot b_2^{-2} + \dots + c_{-s} \cdot b_2^{-s} \quad [1.17]$$

siendo necesario calcular los coeficientes  $c_i$  de  $N_{(b_2)} = c_{-1} c_{-2} \dots c_{-s}$ , que son los dígitos desconocidos del número expresado en la nueva base  $b_2$ . Para ello en la expresión [1.17] se multiplican sus dos miembros por la base  $b_2$ , utilizando la aritmética de base 10,

$$N_{(10)} \cdot b_2 = c_{-1} + c_{-2} \cdot b_2^{-1} + \dots + c_{-s} \cdot b_2^{-s+1} = c_{-1} + R_0 \quad [1.18]$$

donde  $c_{-1}$  es la parte entera del producto.

Para calcular  $c_{-2}$  se multiplica, de nuevo, la parte fraccionaria restante,  $R_0$ , por  $b_2$ .

$$R_0 \cdot b_2 = c_{-2} + c_{-3} \cdot b_2^{-1} + \dots + c_{-s} \cdot b_2^{-s+2} = c_{-2} + R_1 \quad [1.19]$$

El proceso se repite hasta que la parte fraccionaria restante,  $R_i$  sea cero o se obtenga el número de dígitos  $c_i$  deseados. La Tabla 1.4 resume este proceso.

Tabla 1.4. Proceso de conversión de base 10 a base  $b_2$  de la parte fraccionaria de un número

Operación	Resto $R_i$	Parte entera $c_{-i}$
$N_{(10)} \cdot b_2$	$R_0$	$c_{-1}$
$R_0 \cdot b_2$	$R_1$	$c_{-2}$
$R_1 \cdot b_2$	$R_2$	$c_{-3}$
$\vdots$	$\vdots$	$\vdots$

**Observación:** No siempre este proceso es finito y concluye cuando la parte fraccionaria se hace cero. Hay casos en los que la parte fraccionaria de un número en base  $b_1$  no puede ser representada en base  $b_2$ , con un número finito de dígitos. Cuando se dé esta circunstancia, para representar el número, se debe buscar una serie de dígitos que se repitan periódicamente y en el caso de que no exista esta serie periódica, se deberá limitar la cantidad de dígitos a calcular en función de la exactitud que se requiera en la conversión.

### PROBLEMA RESUELTO 1-7

Convertir de **decimal a binario** el número  $0,375_{(10)}$

#### Solución:

Aplicando el proceso indicado anteriormente, en la Tabla 1.4, considerando que  $N_{(10)} = 0,375_{(10)}$ , y  $b_2 = 2$ , se obtiene que dicha conversión es la representada en la Tabla 1.5.

Tabla 1.5. Conversión del número  $0,375_{(10)}$  de decimal a binario

Operación	Parte fraccionaria $R_i$	Parte entera $c_{-i}$
$0,375 \cdot 2 = 0,75$	0,75	$c_{-1} = 0$
$0,75 \cdot 2 = 1,5$	0,5	$c_{-2} = 1$
$0,5 \cdot 2 = 1,0$	0 (Resto nulo)	$c_{-3} = 1$

El resultado obtenido en la Tabla 1.5 es:  $0,375_{(10)} = 0,011_{(2)}$

A continuación se realiza el mismo proceso de conversión representado de otra forma.



$$\begin{array}{r}
 0,375 \cdot 2 = 0,75 \\
 0,75 \cdot 2 = 1,5 \\
 0,5 \cdot 2 = 1,0
 \end{array}
 \begin{array}{l}
 | \\
 | \\
 \downarrow
 \end{array}$$

El resultado numérico de la conversión es igual a la serie de dígitos de la parte entera obtenida en los productos, en el orden indicado por la flecha. Es decir,  $0,375_{(10)} = 0,011_{(2)}$

### PROBLEMA RESUELTO 1-8

Convertir de **decimal a octal** el número  $0,176_{(10)}$

**Solución:**

$$\begin{array}{r}
 0,176 \cdot 8 = 1,408 \\
 0,408 \cdot 8 = 3,264 \\
 0,264 \cdot 8 = 2,112 \\
 0,112 \cdot 8 = 0,896 \\
 0,896 \cdot 8 = 7,168 \\
 0,168 \cdot 8 = 1,344 \\
 0,344 \cdot 8 = 2,752 \\
 0,752 \cdot 8 = 6,016
 \end{array}
 \begin{array}{l}
 | \\
 | \\
 | \\
 | \\
 | \\
 | \\
 \downarrow
 \end{array}$$

La parte fraccionaria del resultado obtenido tiene infinitos dígitos, por lo que se finaliza la conversión una vez que se alcanza la precisión deseada. El resultado es el siguiente:  $0,176_{(10)} = 0,13207126..._{(8)}$ .

### PROBLEMA RESUELTO 1-9

Convertir de **decimal a hexadecimal** el número  $0,145_{(10)}$

**Solución:**

$$\begin{array}{r}
 0,145 \cdot 16 = 2,32 \\
 0,32 \cdot 16 = 5,12 \\
 0,12 \cdot 16 = 1,92 \\
 0,92 \cdot 16 = 14,72 \\
 0,72 \cdot 16 = 11,52 \\
 0,52 \cdot 16 = 8,32 \\
 0,32 \cdot 16 = 5,12
 \end{array}
 \begin{array}{l}
 | \\
 | \\
 | \\
 | \\
 | \\
 \downarrow
 \end{array}$$

La parte fraccionaria del resultado obtenido es periódica, siendo:

$$0,145_{(10)} = 0,251\overline{EB8}_{(16)}$$

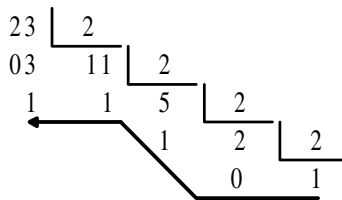
El resultado de **convertir un número decimal con parte entera y fraccionaria**, en un número de otra base, se obtendrá al convertir por separado su parte entera y su parte fraccionaria y sumar los resultados.

### PROBLEMA RESUELTO 1-10

Convertir de **decimal a binario** el número  $23,625_{(10)}$

#### Solución:

a) Se convierte primero la parte entera.



La parte entera tiene como resultado:  $23_{(10)} = 10111_{(2)}$ .

b) Se convierte posteriormente la parte fraccionaria.

$$\begin{array}{r} 0,625 \cdot 2 = 1,25 \\ 0,25 \cdot 2 = 0,5 \\ 0,5 \cdot 2 = 1 \end{array} \quad \downarrow$$

La parte fraccionaria tiene como resultado:  $0,625_{(10)} = 0,101_{(2)}$ .

El resultado final de la conversión es la suma de las partes entera y fraccionaria:  $23,625_{(10)} = 10111_{(2)} + 0,101_{(2)} = 10111,101_{(2)}$ .

### PROBLEMA RESUELTO 1-11

Convertir el número  $121,02_3$  a base nueve.

#### Solución:

a) El primer paso es convertir el número a decimal.

$$\begin{array}{r}
 1 \ 2 \ 1, \ 0 \ 2 \quad (3) \\
 \begin{array}{l}
 \longleftarrow 1 \cdot 3^2 = 9 \\
 \longleftarrow 2 \cdot 3^1 = 6 \\
 \longleftarrow 1 \cdot 3^0 = 1 \\
 \longleftarrow 0 \cdot 3^{-1} = 0 \\
 \longleftarrow 2 \cdot 3^{-2} = 0,2 \\
 \hline
 16,2 \quad (10)
 \end{array}
 \end{array}$$

- b) El segundo paso consiste en representar el número decimal en base nueve. Primero se convierte la parte entera y posteriormente la parte fraccionaria.

La parte entera:

$$\begin{array}{r}
 16 \ \underline{) \ 9} \\
 7 \ \underline{) \ 1} \\
 \longleftarrow
 \end{array}$$

Resultado:  $16_{(10)} = 17_{(9)}$ .

La parte fraccionaria:

$$\begin{array}{r}
 0,2 \cdot 9 = 1,9 \\
 0,9 \cdot 9 = 8,9 \\
 0,9 \cdot 9 = 8,9
 \end{array}
 \quad \downarrow$$

Resultado:  $0,2_{(10)} = 0,1\bar{8}_{(9)}$

El resultado final de la conversión es la suma de las partes entera y fraccionaria:  $16,2_{(10)} = 17_{(9)} + 0,1\bar{8}_{(9)} = 17,1\bar{8}_{(9)}$

Los sistemas de numeración más utilizados en Electrónica Digital se reducen al binario, hexadecimal y a veces octal, lo que posibilita, en las conversiones entre ellos, una serie de métodos o simplificaciones que se detallan en los apartados siguientes.

#### 1.2.1.4 SISTEMA BINARIO

El sistema binario es un sistema de numeración en base dos, introducido por Leibniz en el siglo XVII. Está basado en la utilización exclusiva, en cada dígito

binario denominado **bit**, de dos símbolos distintos, el 0 y el 1, para expresar cualquier magnitud numérica.

La importancia de este sistema radica en la sencillez de sus reglas aritméticas y en que los componentes electrónicos que se emplean para la realización de los circuitos digitales (transistores, diodos, etc.) presentan dos estados estables perfectamente diferenciados, que hacen que sea el sistema idóneo para su uso en Electrónica Digital. Las ventajas anteriores compensan el inconveniente de necesitar, para representar una misma cantidad, un mayor número de dígitos que en aquellos otros sistemas cuya base es mayor.

Como ya se ha indicado anteriormente, en el apartado 1.2.1, los pesos binarios de  $a_n \dots a_2 a_1 a_0$ , son respectivamente,  $2^n, \dots, 4, 2, 1$ . Sumar en decimal los pesos de los dígitos binarios cuyo valor sea la unidad es un procedimiento rápido para convertir un número binario a decimal.

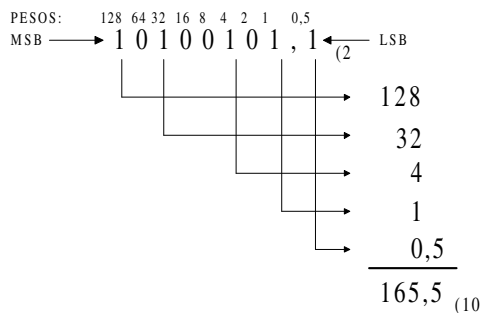
El bit de menor peso o menos significativo se denomina **LSB** (*Least Significant Bit*) y el bit de mayor peso o más significativo **MSB** (*Most Significant Bit*).

También, inversamente, se puede utilizar este mismo procedimiento para convertir números decimales a binarios. Para ello se comienza representando los pesos de cada posición binaria, poniendo a 1 el conjunto de dígitos binarios cuya suma dé el valor decimal. Es decir, se pone a 1 el dígito de mayor peso que no supere el valor a convertir; se resta este valor del número a convertir y con el resto, de forma iterativa, se vuelve a poner a 1 otro dígito cuyo peso sea lo mayor posible, restándose nuevamente, y así hasta que el valor de la resta sea cero.

## PROBLEMA RESUELTO 1-12

Utilizando el procedimiento rápido explicado anteriormente, convertir de binario a decimal el número  $10100101,1_{(2)}$ .

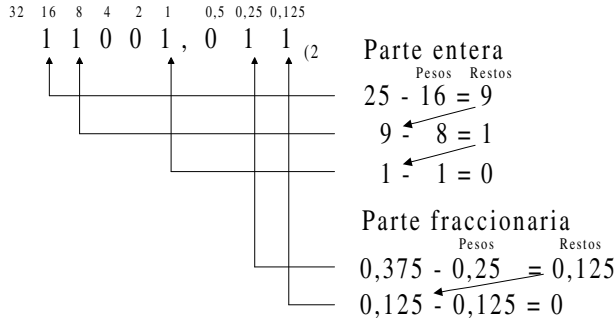
**Solución:**



### PROBLEMA RESUELTO 1-13

Convertir el número  $25,375_{(10)}$  de decimal a binario, utilizando el procedimiento rápido explicado anteriormente.

#### Solución:



Conociendo los anteriores procedimientos de conversión, es fácil construir la Tabla 1.6, que relaciona la formación o construcción ordenada de números binarios de cuatro bits y sus correspondientes valores en decimal.

En la misma Tabla 1.6 se puede apreciar que la construcción de números binarios y decimales siguen el mismo procedimiento. Conviene que las personas poco familiarizadas con los sistemas de numeración de base distinta a la decimal se den cuenta que los procedimientos y reglas son los mismos en las distintas bases. Así, cuando se cuenta en decimal y se llega al máximo valor que se puede expresar con un dígito (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), se siguen construyendo números con dos dígitos, incrementándose el dígito de la izquierda cada vez que se completa la serie (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), en el dígito de la derecha. Cuando se llega al número 99, se añade una cifra más a la izquierda y se sigue el mismo procedimiento. Cada vez que un dígito completa la serie, al comenzar otra vez la serie (pasa de 9 a 0), se incrementa el dígito situado a su izquierda.

En el sistema binario ocurre exactamente igual, lo único que sólo tiene los guarismos 0 y 1. Con un dígito sólo se podrá contar (0, 1), para continuar contando se necesitan números con dos dígitos (10, 11), posteriormente se deberá pasar a tres dígitos (100, 101, 110, 111) y así sucesivamente. Cada vez que un dígito completa la serie (0, 1), al comenzar otra vez la serie (pasa de 1 a 0), se incrementa el dígito situado a su izquierda.

Tabla 1.6. Números binarios de 4 bits

Decimal	Binario	Decimal	Binario
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

**Ejemplo:**

Hodómetro:

Todo lo indicado anteriormente recuerda al funcionamiento de un hodómetro o cuentakilómetros formado por una serie de discos en el que están impresos los guarismos del alfabeto decimal (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Cuando el disco de las unidades da una vuelta completa, incrementa en una posición ( $360^\circ$ ) o valor numérico al disco de las decenas, situado a su izquierda. Lo mismo ocurre con el disco de las centenas, y así sucesivamente.

De la misma forma se podría contar los kilómetros en otra base distinta, por ejemplo para  $b = 4$ , los discos tendrían cuatro posiciones (0, 1, 2, 3) y cada disco al dar una vuelta incrementa al disco situado a su izquierda.

El contador binario tendría discos con 2 posiciones (0, 1) y en el caso de que fueran cuatro discos, la lectura coincidiría con la Tabla 1.6.

Con  $n$  dígitos binarios se pueden representar todos los enteros cuya magnitud  $N$  esté comprendida en el rango  $0 \leq N \leq 2^n - 1$ . En la Tabla 1.6 se aprecia que con cuatro dígitos binarios ( $n = 4$ ) se pueden representar valores decimales desde 0 a  $2^4 - 1 = 15$ . En general, un número  $N$ , en un sistema de base  $b$  con  $n$  dígitos enteros se pueden representar  $b^n$  valores distintos, comprendidos entre  $0 \leq N \leq b^n - 1$ .

En el caso de una representación puramente fraccionaria con  $q$  dígitos binarios, el rango de la magnitud es  $0 \leq N \leq 1 - 2^{-q}$ . En general, para un número  $N$ , en un sistema de base  $b$  con  $q$  dígitos fraccionarios, se pueden representar  $b^q$  valores distintos, comprendidos en el rango:  $0 \leq N \leq 1 - b^{-q}$ .

### 1.2.1.5 SISTEMA OCTAL

Aunque los circuitos electrónicos digitales utilizan exclusivamente el sistema binario, éste resulta engorroso para el usuario por dos razones:

- Es laborioso por la gran cantidad de dígitos que emplea para expresar un valor.
- Es peligroso por la facilidad que existe de cometer un error, cuando el número está formado por muchos dígitos con sólo dos símbolos.

El uso del sistema octal, así como el hexadecimal, permite la conversión de números binarios con numerosos dígitos a una forma más compacta de la información, simple y conveniente para su lectura.

En la Tabla 1.7 se expresan los primeros números decimales y su correspondiente representación en octal y en binario.

Hay que tener en cuenta que este sistema es de base 8 y utiliza por tanto, ocho símbolos o guarismos diferentes: 0, 1, 2, 3, 4, 5, 6 y 7 que pueden ser representados con un único dígito. Completada la serie anterior se pasa a representar los siguientes números con dos dígitos y así sucesivamente, con tres dígitos, cuatro, etc.

*Tabla 1.7. Números octales y su relación con decimales y binarios*

Decimal	Octal	Binario	Decimal	Octal	Binario
0	0	000	9	11	1 001
1	1	001	10	12	1 010
2	2	010	11	13	1 011
3	3	011	⋮	⋮	⋮
4	4	100	15	17	1 111
5	5	101	16	20	10 000
6	6	110	17	21	10 001
7	7	111	⋮	⋮	⋮
8	10	1 000	63	77	111 111

Este sistema presenta la ventaja de permitir una fácil conversión de binario a octal y viceversa, debido a que su base es potencia entera de dos,  $2^3 = 8$ , lo que implica que cada dígito octal tiene una correspondencia con tres dígitos binarios o bits.

Para obtener la **conversión de binario a octal y viceversa** se realiza una partición del número binario en grupos de tres dígitos o bits, a derecha e izquierda del punto de referencia (separación de la parte entera y fraccionaria). Cada dígito octal es igual al

valor decimal de los grupos de tres bits anteriormente formados, como se aprecia en los siguientes ejercicios resueltos.

### PROBLEMA RESUELTO 1-14

Convertir el número binario  $1100101,011_2$  a octal.

#### Solución:

Se divide el número binario en grupos de tres bits a la derecha y a la izquierda de la coma y se hace la conversión decimal de cada grupo.

$$\begin{array}{cccc} \underline{001} & \underline{100} & \underline{101} & \underline{,011} \\ 1 & 4 & 5 & 3 \end{array}$$

El resultado de la conversión del número binario  $1100101,011_2$  a octal es  $145,3_8$ .

Al formar los grupos, si es necesario, se añaden ceros a la derecha de la parte fraccionaria o a la izquierda de la parte entera para completar un grupo de tres bits.

### PROBLEMA RESUELTO 1-15

Convertir el número octal  $257,4_8$  a binario.

#### Solución:

Se descompone cada dígito octal en grupos de tres bits con valor decimal igual al valor del dígito octal correspondiente.

$$\begin{array}{cccc} \underline{2} & \underline{5} & \underline{7} & \underline{,4} \\ 010 & 101 & 111 & 100 \end{array}$$

El resultado de la conversión del número octal  $257,4_8$  a binario es  $10101111,1_2$ .

Se pueden desprestigiar los ceros situados en los extremos derecho e izquierdo, obtenidos en la conversión.

## 1.2.1.6 SISTEMA HEXADECIMAL

En el sistema hexadecimal o sistema de base 16, se utilizan como símbolos los diez dígitos decimales y las seis primeras letras del alfabeto (mayúsculas): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F.



El uso del sistema hexadecimal permite la conversión de números binarios con numerosos dígitos a una forma más comprimida de información, simple y conveniente para su lectura.

En la Tabla 1.8 se expresan los primeros números decimales y su correspondiente representación en hexadecimal y binario. Hay que tener en cuenta que este sistema tiene base 16 y utiliza por tanto, dieciséis símbolos o dígitos diferentes: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, que pueden ser representados con un único dígito. Completada la serie anterior se pasa a representar los siguientes números con dos dígitos y así sucesivamente, con tres dígitos, cuatro, etc.

Este sistema presenta la ventaja de permitir una fácil conversión de binario a hexadecimal y viceversa, debido a que su base es potencia entera de dos,  $2^4 = 16$ , lo que implica que cada dígito hexadecimal tiene una correspondencia con cuatro dígitos binarios o bits.

Para realizar la **conversión de binario a hexadecimal y viceversa** se realiza una partición del número binario en grupos de cuatro dígitos o bits, a derecha e izquierda del punto de referencia (separación entre parte entera y fraccionaria). Cada dígito hexadecimal es igual al valor decimal de los grupos de cuatro bits anteriormente formados, como se aprecia en los siguientes problemas resueltos.

*Tabla 1.8. Números hexadecimales y su relación con decimales y binarios*

Decimal	Hexadecimal	Binario	Decimal	Hexadecimal	Binario
0	0	0000	11	B	1011
1	1	0001	12	C	1100
2	2	0010	13	D	1101
3	3	0011	14	E	1110
4	4	0100	15	F	1111
5	5	0101	16	10	1 0000
6	6	0110	17	11	1 0001
7	7	0111	⋮	⋮	⋮
8	8	1000	31	1F	1 1111
9	9	1001	32	20	10 0000
10	A	1010	⋮	⋮	⋮

### PROBLEMA RESUELTO 1-16

Convertir el número binario 11011001001,1101<sub>2</sub> a hexadecimal.

**Solución:**

Se divide el número binario en grupos de cuatro bits a la derecha y a la izquierda de la coma y se hace la conversión decimal de cada grupo.

$$\underbrace{01101}_{6} \underbrace{1001}_{C} \underbrace{001}_{9}, \underbrace{1101}_{D}$$

El resultado de la conversión del número binario  $11011001001,1101_{(2)}$  a hexadecimal es  $6C9,D_{(16)}$ .

Al formar los grupos, si es necesario, se añaden ceros a la derecha de la parte fraccionaria o a la izquierda de la parte entera para completar un grupo de cuatro bits.

**PROBLEMA RESUELTO 1-17**

Convertir el número hexadecimal  $7A5,6_{(16)}$  a binario.

**Solución:**

Se descompone cada dígito hexadecimal en grupos de cuatro bits con valor decimal igual al valor del dígito hexadecimal correspondiente.

$$\underbrace{7}_{0111} \underbrace{A}_{1010} \underbrace{5}_{0101}, \underbrace{6}_{0110}$$

El resultado de la conversión del número hexadecimal  $7A5,6_{(16)}$  a binario es  $11110100101,011_{(2)}$ .

Se pueden despreciar los ceros situados en los extremos derecho e izquierdo, obtenidos en la conversión.

Para la **conversión de octal a hexadecimal y viceversa**, lo más sencillo es convertir primero el número en binario y luego a hexadecimal o a octal, según corresponda.

**PROBLEMA RESUELTO 1-18**

Convertir el número octal  $37,6_{(8)}$  a hexadecimal.

**Solución:**

Se convierte primero el número octal a binario y posteriormente a hexadecimal.

$$\underbrace{3}_{011111}, \underbrace{7}_{110}, \underbrace{6}_{110} = \underbrace{00011111}_1, \underbrace{1100}_{FC}$$

El resultado de convertir el número octal  $37,6_{(8)}$  a hexadecimal es  $1F,C_{(16)}$ .

### PROBLEMA RESUELTO 1-19

Convertir el número hexadecimal  $6A,D_{(16)}$  a octal.

#### Solución:

Se convierte primero el número hexadecimal a binario y posteriormente a octal.

$$\underbrace{6}_{01101010}, \underbrace{A}_{1101}, \underbrace{D}_{110100} = \underbrace{001101010}_1, \underbrace{110100}_{64}$$

El resultado de la conversión del número hexadecimal  $6A,D_{(16)}$  a octal es  $152,64_{(8)}$ .

### PROBLEMAS PROPUESTOS

- 1-1) Hacer una tabla que relacione los 20 primeros números de los sistemas de numeración decimal, binaria, octal y hexadecimal.
- 1-2) Convertir el número binario  $10111,01_{(2)}$  a decimal.
- 1-3) Convertir el número decimal  $52,375_{(10)}$  a binario.
- 1-4) Convertir el número binario  $101011,101_{(2)}$  a octal y hexadecimal.
- 1-5) Convertir el número octal  $654,40625_{(8)}$  a decimal.
- 1-6) Convertir el número decimal  $123,45_{(10)}$  a octal.
- 1-7) Convertir el número octal  $365,6_{(8)}$  a binario y hexadecimal.
- 1-8) Convertir el número hexadecimal  $1B3,2_{(16)}$  a decimal.
- 1-9) Convertir el número decimal  $251,625_{(10)}$  a hexadecimal.
- 1-10) Convertir el número hexadecimal  $1B3,2_{(16)}$  a binario y octal.



Se utiliza la calculadora de Windows en modo “Científica”, para obtener las representaciones de números enteros sin signo en las bases: binaria, octal, hexadecimal y decimal.

Los problemas resueltos (propuestos) se pueden comprobar (resolver), mediante el uso de la calculadora de Windows 9x, NT, 2xxx y mE que se encuentra en el menú: **Inicio/Programas/Accesorios/Calculadora**. Dentro de esta utilidad hay que trabajar en modo calculadora científica mediante la opción: **Ver/Científica**.

Así por ejemplo se puede comprobar el Problema resuelto 1-4 y siguientes, que consisten en convertir el número decimal  $324_{(10)}$  a binario, octal y hexadecimal. Para ello se activa, en la calculadora, el sistema numérico **Dec** y se introduce el número decimal 324, a través del teclado del ordenador o mediante la pulsación, con el ratón, sobre el teclado numérico de la calculadora, apareciendo dicho número en el *display*, como se muestra en la Figura 1.8 (calculadora de Windows 9x).



Figura 1.8. Introducción del número decimal 324 en la calculadora de Windows

Para convertir este número decimal  $324_{(10)}$  a binario (Problema resuelto 1-4), se activara la opción **Bin**, obteniéndose en el *display* su valor equivalente en binario, tal como se muestra en la Figura 1.9. Conviene tener activada la opción **Dword**, lo cual permite trabajar con registros de doble palabra (32 bits) y, por tanto, hace posible conversiones de números con más dígitos (mayores). El resultado obtenido es:  $101000100_{(2)}$ .

Para convertir este número decimal  $324_{(10)}$  a octal (Problema resuelto 1-5), se activara la opción **Oct**, obteniéndose en el *display* su valor equivalente en octal, como se muestra en la Figura 1.10. Conviene tener activada la opción **Dword**, lo cual permite trabajar con registros de doble palabra (11 dígitos octales, uno de ellos sólo puede representar valores entre 0 y 3) y, por tanto, hace posible conversiones de números con más dígitos (mayores). El resultado obtenido es:  $504_{(8)}$ .



Figura 1.9. Conversión a binario del número decimal 324 utilizando la calculadora de Windows



Figura 1.10. Conversión a octal del número decimal 324 utilizando la calculadora de Windows

Para convertir este número decimal  $324_{(10)}$  a hexadecimal (Problema resuelto 1-6), se activará la opción **Hex**, obteniéndose en el *display* su valor equivalente en hexadecimal, como puede apreciarse en la Figura 1.11. Conviene tener activada la opción **Dword**, lo cual permite trabajar con registros de doble palabra (8 dígitos hexadecimales) y, por tanto, hace posible conversiones de números con más dígitos (mayores). El resultado obtenido es:  $144_{(16)}$ .

Como se puede apreciar, en la calculadora se han obtenido los mismos resultados que los calculados en los problemas resueltos, anteriormente indicados.



Figura 1.11. Conversión a hexadecimal del número decimal 324 utilizando la calculadora de Windows

## 1.2.2 Representaciones numéricas

Los **números reales** se clasifican en:

- **Naturales:** 0, 1, 2, 3, ...
- **Enteros** (positivos y negativos): ... -3, -2, -1, 0, 1, 2, 3, ...
- **Racionales.** Se expresan como cociente entre dos enteros. Estos números se representan con un número finito de decimales o mediante forma periódica:

$$\frac{1}{5} = 0,2$$

$$\frac{1}{3} = 0,333\dots = 0,\hat{3}$$

- **Irracionales.** No tienen una correspondencia con las clases anteriores. Se representan con un número infinito de decimales no periódicos:

$$\sqrt{2} \text{ y } \sqrt{3}$$

Dentro de los números irracionales destacan los **transcendentes**, como el número  $\pi$  y el número  $e$ .

Los conjuntos anteriores son infinitos, mientras que el espacio material de representación de los sistemas digitales es finito, esto implica que no se puede representar todos los números. En el caso de los números irracionales, ninguno de éstos se puede representar, por requerir un número infinito de bits para ello.

Normalmente, en los sistemas digitales se asigna un número fijo  $n$  de bits para representar un número, donde  $n$  es la longitud de una **palabra (Word)**, media palabra, **doble palabra (Dword)**, etc. Una palabra está compuesta por un número de bits que pueden ser tratados en su conjunto o simultáneamente en una operación del sistema digital, siendo sus longitudes más corrientes: 8, 16, 32 o 64 bits, según el sistema. Así por ejemplo, cuando  $n = 16$ , la media palabra tendrá una longitud de un byte (8 bits), la palabra tiene dos bytes (16 bits) y la doble palabra cuatro bytes (32 bits).

Con  $n$  bits se pueden representar  $2^n$  combinaciones distintas y por lo tanto  $2^n$  números diferentes, por lo que existirán dos valores extremos, un máximo y un mínimo, que acotarán a todos los números representables.

Se denomina **rango de representación** al intervalo comprendido entre el menor y el mayor número representable.

Se denomina **resolución de la representación** a la mayor diferencia que existe entre un número representable y su inmediato siguiente o sucesor. Este parámetro determina el máximo error que se puede cometer al representar un número.

Mediante los sistemas polinomiales se pueden representar los números:

- **Naturales** (incluido el cero), mediante representaciones en coma fija sin signo.
- **Enteros**, mediante representaciones en coma fija con signo.
- **Racionales**, mediante representaciones en coma flotante.

### 1.2.3 Representación de números en coma fija sin signo

Los sistemas estudiados anteriormente en el apartado 1.2.1 son de coma fija sin signo. Con este formato un número se representa mediante dos partes separadas mediante una coma (punto en notación anglosajona), que ocupa una posición fija dentro de los dígitos que componen el número. A la izquierda de la coma se encuentra la parte entera, mientras que a su derecha está la parte fraccionaria. Cada una de estas partes tienen un número de bits fijo o prefijado de antemano. Es este hecho, que la parte entera y fraccionaria tenga un número fijo de bits, el que obliga a que la coma se encuentre en una posición fija, de ahí la denominación de **coma fija**.

#### 1.2.3.1 REPRESENTACIÓN DE NÚMEROS NATURALES EN BINARIO PURO

El sistema de representación en binario puro se realiza mediante coma fija sin signo, siendo un sistema polinomial de base  $b = 2$  y sin parte fraccionaria. Los  $n$  dígitos binarios forman una cadena  $(a_{n-1} a_{n-2} \dots a_2 a_1 a_0)$ , tal que:

$$N_{(10)} = a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \dots + a_22^2 + a_12 + a_0 = \sum_{i=0}^{n-1} a_i \cdot 2^i \quad [1.20]$$

Este sistema permite representar, con palabras de  $n$  bits, todos los enteros positivos desde 0 hasta  $2^n-1$ . Por tanto su rango es de  $[0, 2^n-1]$  y su resolución es la unidad.

## 1.2.4 Representación de números en coma fija con signo

Al igual que en la vida cotidiana se efectúan operaciones tanto con números positivos como con negativos, los sistemas digitales deben ser capaces de procesar también este tipo de números, es decir, información numérica con signo.

En la representación habitual de números se añade un signo a su izquierda, + en los números positivos (aunque en la práctica se omite y se sobreentiende que cuando no lo lleva es positivo) y - en los números negativos.

Si se aplicara este mismo procedimiento para representar, en sistemas digitales, números binarios con signo, sería necesario trabajar con tres niveles lógicos: el 0, el 1 y el signo, lo que complicaría considerablemente los circuitos de estos sistemas.

La solución adoptada para evitar la inclusión de un nuevo nivel para el signo, es añadir un dígito más que indique el signo del número. Este bit denominado **dígito de signo**, se encuentra situado en el extremo izquierdo de la representación del número, y toma el valor 0 cuando se trate de números positivos y el valor 1 para los negativos.

Los formatos más habituales de representación de los números con signo son: signo-magnitud, complemento a la base y complemento a la base menos uno. A continuación se detalla cada uno de estos sistemas de representación.

### 1.2.4.1 FORMATO DE NÚMEROS BINARIOS EN SIGNO-MAGNITUD

Este tipo de representación utiliza uno de los dígitos, el situado más a la izquierda del número, para indicar su signo. Recibe el nombre de signo-magnitud, porque un dígito se dedica al signo y los demás a la magnitud.

El formato de representación binaria utiliza  $n$  bits ( $a_{n-1} a_{n-2} \dots a_1 a_0$ ), siendo el más significativo ( $a_{n-1}$ ) el que indica el signo del número: 0 indica número positivo y 1 negativo. Los restantes  $n-1$  bits expresan la magnitud del número, como puede verse en la Figura 1.12.

Por tanto, con  $n$  bits, el valor decimal que se puede representar estará comprendido en el **rango simétrico** de  $\pm (2^{n-1}-1)$ .



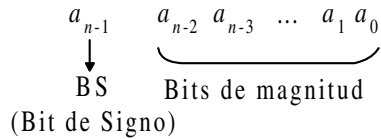


Figura 1.12. Formato de números binarios en signo-magnitud

Un **inconveniente** que tiene el sistema de representación en signo-magnitud, es la necesidad de utilizar circuitos diferentes para realizar las operaciones de suma y de resta.

### PROBLEMA RESUELTO 1-20

Un formato muy corriente es representar números binarios en ocho bits incluido el signo. Calcular su rango en representación decimal.

**Solución:**

$$[-(2^7-1), +(2^7-1)] = [-127, +127]$$

### PROBLEMA RESUELTO 1-21

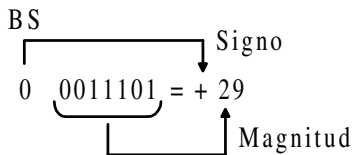
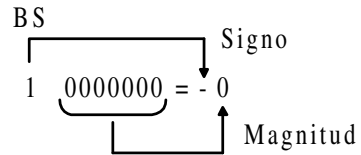
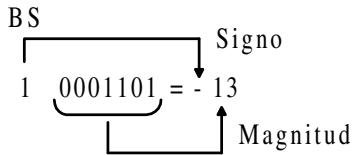
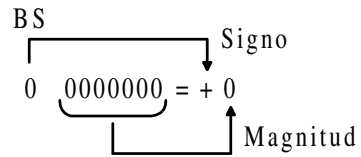
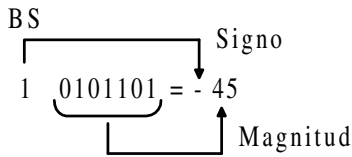
Dados los números binarios de ocho bits, representados en signo-magnitud,  $10101101_{(2)}$ ,  $10001101_{(2)}$ ,  $00011101_{(2)}$ ,  $00000000_{(2)}$ ,  $10000000_{(2)}$ , indicar:

- Cuáles son positivos y cuáles negativos.
- Determinar su valor decimal.

**Solución:**

- Los números primero, segundo y quinto, tienen el bit de signo igual a 1 y por lo tanto son negativos. El tercer y cuarto número son positivos por tener el bit de signo igual a 0.
- La magnitud está representada por los 7 bits menos significativos (situados a la derecha), siendo:

Los dos ejemplos de la derecha ponen de manifiesto que el valor cero tiene dos posibles representaciones, una con signo positivo y otra con negativo.

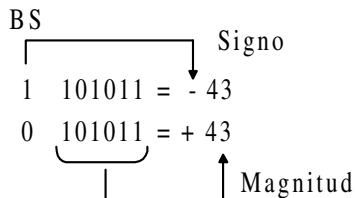


## PROBLEMA RESUELTO 1-22

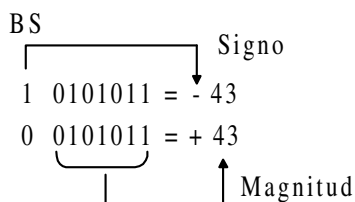
Representar en binario y en signo-magnitud, los números decimales -43 y 43.

### Solución:

Ambos números tienen la misma magnitud  $43_{(10)}$ , que expresada en binario utilizando seis bits es  $101011_{(2)}$ , y a la que habrá que añadirle el signo para obtener los dos números.



La representación de ambos números utilizando ocho bits incluido el bit de signo es:



### 1.2.4.2 COMPLEMENTOS

Para evitar el inconveniente del sistema de representación en signo-magnitud se utilizan los **complementos**, que son transformaciones en la representación de números que realizan **la función de cambio de signo del número**. Es utilizado por las máquinas digitales para convertir restas en sumas (se resta mediante la operación suma); permitiendo aplicar el mismo circuito para sumar y restar, y por lo tanto simplificando considerablemente los circuitos.

Los complementos pueden ser de dos clases: **complemento a la base  $b$**  y complemento a la base menos uno  $b-1$  o también denominado **restricción de la base**. En el sistema decimal será complemento a diez y complemento a nueve; en el sistema binario complemento a dos y complemento a uno, etc.

### 1.2.4.3 COMPLEMENTO A LA BASE

Dado un número positivo  $N$ , de  $n$  dígitos enteros y representado en base  $b$ , se define su complemento a la base, como el número  $C_b(N)$  que cumple:

$$N + C_b(N) = b^n \quad [1.21]$$

Como observación, cabe decir que  $N$  puede tener parte fraccionaria y que  $b^n$  es la potencia de la base inmediatamente superior a  $N$ . Para  $N > 0$  el valor de su complemento a la base es igual a:

$$C_b(N) = b^n - N \quad [1.22]$$

Para  $N = 0$  su complemento vale  $C_b(N) = 0$ , es decir, que el número cero tiene la misma representación que la de su complemento a la base. De la expresión [1.22] se puede deducir que el complemento del complemento a la base del número  $N$ , es dicho número, tal como puede verse en la expresión [1.23]. Esto es evidente si se considera que la función del complemento es la de cambio de signo en el número y que si se realiza dos cambios de signo se vuelve a tener el número inicial.

$$C_b(C_b(N)) = b^n - (b^n - N) = N \quad [1.23]$$

En el sistema decimal, el complemento a la base recibe el nombre de **complemento a diez** y en el sistema binario se denomina **complemento a dos**.

### PROBLEMA RESUELTO 1-23

Calcular el complemento a la base de los números:

- a)  $72_{(10)}$

- b)  $28_{(10)}$
- c)  $110,01_{(2)}$
- d)  $0_{(b)}$

**Solución:**

- a) El complemento a diez de  $72_{(10)}$  es igual a:  
 $C_{10}(72) = 10^2 - 72 = 100 - 72 = 28_{(10)}$
- b)  $C_{10}(28) = 10^2 - 28 = 100 - 28 = 72$ . Observando los apartados a) y b) se aprecia que se cumple que el complemento del complemento es el número original.
- c)  $C_2(110,01) = 2^3 - 110,01 = 1000 - 110,01 = 001,11$ .  
 El número binario ( $b = 2$ ),  $N = 110,01$  tiene tres dígitos enteros ( $n = 3$ ) y dos fraccionarios ( $m = 2$ ), luego  $C_2(N) = b^n - N = 2^3 - 110,01 = 1000 - 110,01 = 001,11$ . Obsérvese que  $b^n$  es la potencia de la base inmediatamente superior a  $N$ .
- d) Por definición, el complemento a la base de un número igual a cero vale cero,  $C_b(0) = 0$ . Se puede sacar la siguiente conclusión: el cero tiene la misma representación que la de su complemento a la base.

### 1.2.4.4 COMPLEMENTO A LA BASE MENOS UNO

Dado un número positivo  $N$  en base  $b$ , compuesto por  $n$  dígitos en la parte entera y  $m$  dígitos en la parte fraccionaria, se define su complemento a la base menos uno, como el número  $C_{b-1}(N)$  que cumple:

$$N + C_{b-1}(N) = b^n - b^{-m} \quad [1.24]$$

Cuando la parte fraccionaria es cero,  $b^{-m} = b^0 = 1$ , siendo en este caso el complemento a la base menos uno igual a:

$$C_{b-1}(N) = b^n - 1 - N \quad [1.25]$$

Se debe observar que  $b^n - 1$  es el valor máximo que se puede representar en la base  $b$  con  $n$  dígitos enteros.

Para  $N > 0$ , el valor de su complemento es igual a:  $C_b(N) = b^n - 1 - N$ .

Para  $N = 0$  su complemento vale  $C_{b-1}(N) = b^n - 1 - N = b^n - 1$ , es decir, que el número cero tiene distinta representación que la de su complemento a la base menos uno.

A partir de la expresión [1.24] se puede deducir que el complemento del complemento (a la base menos uno) del número  $N$ , es dicho número, como así se deduce en la expresión [1.26]. Esto es evidente si se considera que la función del complemento es la de cambio de signo en el número y que si se realizan dos cambios de signo se obtiene el número inicial.

$$C_{b-1}(C_{b-1}(N)) = b^n - b^{-m} - (b^n - b^{-m} - N) = N \quad [1.26]$$

En el sistema decimal el complemento a la base menos uno recibe el nombre de **complemento a nueve** y en el sistema binario se denomina **complemento a uno**.

### PROBLEMA RESUELTO 1-24

Calcular el complemento a la base menos uno de los siguientes números:

- $72_{(10)}$
- $27_{(10)}$
- $110,01_{(2)}$
- $0_{(b)}$

#### Solución:

- El complemento a nueve de  $72_{(10)}$  es igual a:  
 $C_9(72) = 10^2 - 1 - 72 = 99 - 72 = 27$ .
- $C_9(27) = 10^2 - 1 - 27 = 99 - 27 = 72$ . Observando los ejemplos a) y b) se aprecia que se cumple que el complemento del complemento es el número original.
- El número binario ( $b = 2$ ),  $N = 110,01$  tiene tres dígitos enteros ( $n = 3$ ) y dos fraccionarios ( $m = 2$ ), luego  $C_1(N) = (b^n - b^{-m}) - N = (2^3 - 2^{-2}) - 110,01 = (111,11) - 110,01 = 001,10$ . Obsérvese que  $(b^n - b^{-m})$  es igual al máximo valor numérico que se puede representar con  $n$  dígitos enteros y  $m$  fraccionarios en la base  $b$ .
- Por definición, el complemento a la base menos uno de un número igual a cero con un dígito entero ( $n = 1$ ) vale:  $C_{b-1}(0) = b^n - 1 - 0 = b - 1$  y con  $n$  dígitos enteros sería:  $C_{b-1}(00..00) = b^n - 1 - 0 = b^n - 1$ .

### 1.2.4.5 CONVENIO DEL COMPLEMENTO A DOS EN NÚMEROS BINARIOS

El complemento a la base, estudiado anteriormente en el apartado 1.2.4.3, cuando se aplica al caso particular del sistema binario ( $b = 2$ ) se denomina complemento a

dos, permite la representación de números negativos y cumple todas las propiedades incluidas en dicho apartado.

De la definición de complemento a la base, aplicada al caso particular del sistema binario (complemento a dos), se deduce que un número sumado con su complemento da como resultado cero, siempre y cuando se desprece el acarreo producido tal y como puede verse en la expresión [1.27].

$$N + C_2(N) = N + (2^n - N) = 2^n + (N - N) = 2^n + 0 \quad (\text{acarreo} + 0) \quad [1.27]$$

Es posible pues, representar los números negativos como sus correspondientes positivos complementados a dos, debido a que al sumar un número con su complemento a dos da como resultado cero.

Esta forma de representación de los números permite utilizar también la operación suma en las restas entre números, según se muestra en la expresión [1.28].

$$\begin{aligned} M - N &= M + (-N) = M + C_2(N) = M + (2^n - N) = \\ &= 2^n + (M - N) = 2^n + R \quad (\text{acarreo} + \text{Resultado\_resta}) \end{aligned} \quad [1.28]$$

Despreciando  $2^n$  (acarreo), que es un uno seguido de  $n$  ceros, es decir, no considerando dicho uno situado en el bit más significativo, se obtiene el resultado de la resta  $M - N$ .

La representación numérica de números binarios mediante el **convenio de complemento a dos** es la siguiente:

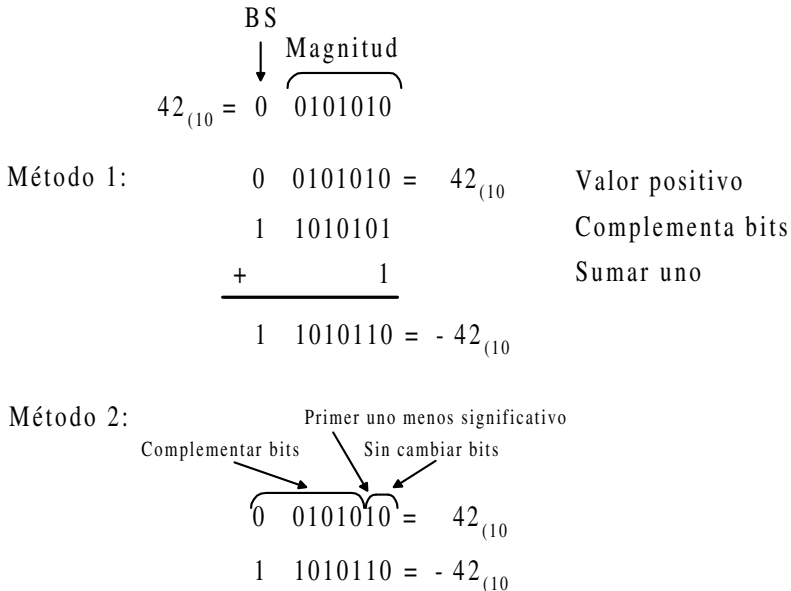
- **Números positivos.** Se representan por el bit de signo *BS* igual a 0, seguido de los bits de magnitud codificados en binario natural. Esto es, igual que en la representación en signo-magnitud, según se aprecia en la Figura 1.13.
- **Números negativos.** Se obtienen hallando el complemento a dos del valor absoluto del número, mediante la expresión [1.22].

Otros métodos más sencillos o prácticos de obtener la representación de los números negativos son:

**Método 1:** Representar el valor absoluto del número, cambiando todos los bits uno por cero y los bits cero por uno (operación de complementación) y sumarle uno.

**Método 2:** Representar de derecha a izquierda el valor absoluto del número, dejando todos los ceros y el primer uno menos significativo sin cambio y cambiando unos por ceros y ceros por unos en el resto de los bits más significativos.

**Ejemplo:**



*Figura 1.13. Representación mediante el convenio de complemento a dos de los números decimales 42 y -42*

En la Tabla 1.9 se muestran los números binarios de cuatro bits incluido el signo, representados mediante el convenio del complemento a dos. Como puede verse en dicha tabla los números negativos se obtienen hallando el complemento a dos de sus correspondientes positivos (a excepción del -8). También se observa que al aplicar el convenio del complemento a dos, el bit más significativo, toma el valor 0 en los números positivos y 1 en los números negativos, por esta razón recibe el nombre de **bit de signo**.

Se aprecia también en la Tabla 1.9, que la **magnitud de los números positivos** está representada en binario natural. Sin embargo, los números negativos no tienen estructura posicional de pesos. Para obtener la **magnitud de los números negativos** hay que complementarlos para obtener así su correspondiente positivo.

El valor -8, en binario 1000 (ocupa cuatro bits), es un caso especial, ya que, al complementarlo a dos se obtiene de nuevo el mismo resultado y no el esperado 8, en binario 01000 (que ocupa cinco bits). Esto se debe a que este sistema de representación tiene una peculiaridad y es que para un formato de cuatro bits con signo incluido, el número negativo -8 (en binario 1000), cuyo correspondiente positivo 8 (en binario 01000), no se puede representar pues son necesarios cinco bits.

En el convenio del complemento a dos con formato de  $n$  bits, incluido el signo, se pueden representar sólo números comprendidos en el **rango asimétrico**  $[-2^{n-1}, 2^{n-1}-1]$ .

*Tabla 1.9. Representación de números binarios de 4 bits mediante el convenio de complemento a dos*

Decimal	Convenio del complemento a dos
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

### Ejemplo:

En el caso de la Tabla 1.9 con formato de cuatro bits ( $n = 4$ ), el rango es  $[-2^3, 2^3 - 1] = [-8, 7]$ .



Se utiliza la calculadora de *Windows*, en modo científica, para obtener las representaciones de números enteros con signo (con ciertas limitaciones) en las bases: binaria, octal, hexadecimal y decimal.

La calculadora de *Windows* permite representar números en convenio de complemento a dos, pero con ciertas limitaciones. Los números negativos se deberán introducir en decimal y al activar la representación binaria, se visualizarán con signo (en complemento a dos).



En el sentido contrario no lo permite, es decir, al introducir el número con signo en binario la utilidad de calculadora no considera al bit más significativo como bit de signo sino como bit de magnitud y por lo tanto al activar la representación decimal, se obtiene siempre un número decimal positivo (formato sin signo).

Los problemas resueltos (propuestos) se pueden comprobar (resolver), con las limitaciones ya indicadas, mediante el uso de la calculadora de Windows 9x, NT, 2xxx o mE que se encuentra en el menú: **Inicio/Programas/Accesorios/Calculadora**. Dentro de esta utilidad hay que trabajar en modo calculadora científica mediante la opción: **Ver/Científica**.

Se puede comprobar, mediante simulación, que se cumple la expresión [1.27] mostrada anteriormente, que indica que “la suma de un número con su complemento a dos es cero”. Así, en el ejemplo representado en la Figura 1.13 se puede hallar mediante la calculadora de Windows la representación de los números  $42_{(10)}$  y  $-42_{(10)}$  en binario (almacenados en un byte, con signo incluido), mediante el convenio del complemento a dos.

Para ello, se introducen, en la calculadora, cada número decimal con la opción **Dec** y posteriormente, activando las opciones **Bin** y **Byte** se visualiza en el *display*, su correspondiente representación en convenio del complemento a dos. Como pueden verse en la Figura 1.14 y en la Figura 1.15 se obtienen los resultados 00101010 y 11010110 correspondientes al número  $42_{(10)}$  y  $-42_{(10)}$  respectivamente.



*Figura 1.14. Obtención del número  $42_{(10)}$  representado según el convenio del complemento a dos, mediante la calculadora de Windows*

Sumados ambos números binarios con la calculadora da como resultado cero (Figura 1.16), cumpliéndose la expresión [1.27].



Figura 1.15. Obtención del número  $-42_{(10)}$  representado según el convenio del complemento a dos, mediante la calculadora de Windows

Se puede comprobar que con el convenio del complemento a dos con formato de un byte, incluido el signo, se pueden representar números comprendidos en el **rango asimétrico**  $[-2^7, 2^7-1] = [-128, 127]$ . Luego el mayor número positivo, representado en convenio del complemento a dos (almacenados en un byte incluido el signo), será aquel cuyo bit de signo sea cero y el resto de los bits de magnitud sean unos; es decir, introduciendo en la calculadora el número binario 01111111, al activar la opción **Dec** se obtiene su valor decimal:  $127_{(10)}$ , como puede verse en la Figura 1.17.

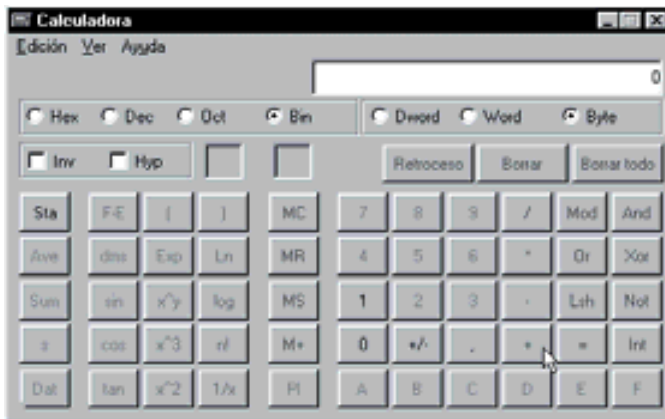


Figura 1.16. Resultado cero, obtenido al sumar el número  $42_{(10)} = 00101010$  y su complemento a dos  $-42_{(10)} = 11010110$



*Figura 1.17. Valor decimal del mayor número positivo que se puede almacenar en un byte, incluido el signo, mediante el convenio del complemento a dos*

La representación del caso especial o mayor número negativo:  $-128_{(10)}$ , representado en convenio del complemento a dos (almacenados en un byte incluido el signo), será aquel cuyo bit de signo sea uno y el resto de los bits de magnitud sean ceros, es decir, introduciendo en la calculadora el número decimal  $-128$ , al activar la opción **Bin** se obtiene su representación binaria en convenio del complemento a dos:  $10000000$ , como puede verse en la Figura 1.18.



*Figura 1.18. Representación binaria del mayor número negativo que se puede almacenar en un byte, incluido el signo, mediante el convenio del complemento a dos*

### 1.2.4.6 CONVENIO DEL COMPLEMENTO A UNO EN NÚMEROS BINARIOS

El complemento a la base menos uno, estudiado anteriormente en el apartado 1.2.4.5 y aplicado al caso particular del sistema binario ( $b = 2$ ), se denomina complemento a uno. Permite la representación de números negativos y cumple todas las propiedades apuntadas en dicho apartado.

De la definición de complemento a la base menos uno, aplicado al caso particular del sistema binario (complemento a uno), se deduce que un número sumado con su complemento da como resultado cero, siempre y cuando se sume el bit del acarreo producido al conjunto restante del resultado, según se muestra en la expresión [1.29].

$$\begin{aligned} N + C_1(N) &= N + (2^n - 1 - N) = 2^n - 1 + (N - N) = \\ &= 2^n - 1 + 0 \quad (\text{acarreo} - 1 + 0) \end{aligned} \quad [1.29]$$

Es posible pues, representar los números negativos como sus correspondientes positivos complementados a uno. Esta forma de representación de los números permite utilizar también la operación suma en las restas entre números, como muestra la expresión [1.30].

$$\begin{aligned} M - N &= M + (-N) = M + C_1(N) = M + (2^n - 1 - N) = \\ &= 2^n - 1 + (M - N) = 2^n - 1 + R \quad (\text{acarreo} - 1 + \text{Resultado}_{\text{resta}}) \end{aligned} \quad [1.30]$$

Sumando el bit del acarreo producido (cuyo valor es 1), al resto del resultado que es una unidad inferior a la diferencia entre  $M$  y  $N$ , se obtiene el resultado de la resta  $M - N$ .

La representación numérica de números binarios mediante el **convenio de complemento a uno** es la siguiente:

- **Números positivos.** Se representan por el bit de signo  $BS$  igual a 0, seguido de los bits de magnitud codificados en binario natural. Esto es, igual que en la representación en signo-magnitud, tal y como puede verse en la Figura 1.19.
- **Números negativos.** Se obtienen hallando el complemento a uno del valor absoluto (valor positivo) del número, mediante la expresión [1.25].

Otro método más sencillo o práctico de obtener la representación de los números negativos es, a partir del valor absoluto (valor positivo) del número cambiar todos los bits uno por cero y los bits cero por uno (operación de complementación).

$$\begin{array}{r}
 \text{BS} \\
 \downarrow \\
 42_{(10)} = 0 \quad \overbrace{0101010}^{\text{Magnitud}} \\
 \\
 0 \quad 0101010 = 42_{(10)} \\
 1 \quad 1010101 = -42_{(10)} \text{ (Complementa bits)}
 \end{array}$$

*Figura 1.19. Representación mediante el convenio de complemento a uno de los números decimales 42 y -42*

En la Tabla 1.10 se muestran los números binarios de cuatro bits incluido el signo, representados mediante el convenio del complemento a uno.

*Tabla 1.10. Representación de números binarios de 4 bits mediante el convenio de complemento a uno*

Decimal	Convenio del complemento a uno
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
-0	1111
-1	1110
-2	1101
-3	1100
-4	1011
-5	1010
-6	1001
-7	1000

Se aprecia en la Tabla 1.10 que los números negativos se obtienen hallando el complemento a uno de sus correspondientes positivos (cambiando ceros por unos y unos por ceros, operación de complementación). También se observa que al aplicar el convenio del complemento a uno, el bit más significativo toma el valor 0 en los números positivos y 1 en los números negativos, por esta razón recibe el nombre de **bit de signo**.

Se observa también, que la **magnitud de los números positivos** está representada en binario natural. Para obtener la **magnitud de los números negativos** hay que complementarlos para obtener así su correspondiente positivo.

En el convenio del complemento a uno existe la representación del 0 y del -0, como en el sistema signo-magnitud.

En el convenio del complemento a uno con formato de  $n$  bits, incluido el signo, se pueden representar números comprendidos en el **rango simétrico**  $[-(2^{n-1}-1), 2^{n-1}-1]$ .

### **Ejemplo:**

En el caso de la Tabla 1.10 con formato de cuatro bits ( $n = 4$ ), el rango es  $[-(2^3-1), 2^3-1] = [-7, 7]$ .

## **1.2.4.7 COMPARACIÓN ENTRE LAS REPRESENTACIONES DIFERENTES DE NÚMEROS BINARIOS CON SIGNO**

Los complementos presentan una gran ventaja sobre el sistema de signo-magnitud, ya que permiten realizar las operaciones de suma y resta mediante un mismo circuito electrónico sumador.

El complemento a dos tiene la ventaja, en las operaciones de sumas y restas, de no necesitar sumar uno al resultado si se produce acarreo, como así lo requiere el convenio del complemento a uno.

El complemento a uno es más fácil de implementar utilizando componentes digitales, ya que sólo hay que cambiar los ceros por unos y viceversa, mediante la operación de complementación. Pero presenta la desventaja de tener dos representaciones del cero, el +0 y -0, lo que complica los circuitos electrónicos capaces de detectar el cero.

Por todas las razones anteriores, la mayoría de las máquinas digitales utilizan el complemento a dos.

En la Tabla 1.11 se muestran los números binarios con signo, de cuatro bits, en las tres distintas representaciones estudiadas.

*Tabla 1.11. Números binarios con signo de cuatro bits representados en las tres diferentes formas estudiadas*

Decimal	Signo-magnitud	Convenio del complemento a uno	Convenio del complemento a dos
7	0111	0111	0111
6	0110	0110	0110
5	0101	0101	0101
4	0100	0100	0100
3	0011	0011	0011
2	0010	0010	0010
1	0001	0001	0001
0	0000	0000	0000
-0	1000	1111	...
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	...	...	1000

### 1.2.5 Representación de los números reales en binario

Todos los números reales binarios, que se han representado anteriormente, tienen un formato de coma fija. Con este formato un número real se representa mediante dos partes separadas mediante una coma. A la izquierda de la coma se encuentra la parte entera, mientras que a su derecha está la parte fraccionaria. Cada una de estas partes tienen un número de bits fijo o prefijado de antemano. Es este hecho, que la parte entera y fraccionaria tenga un número fijo de bits, el que obliga a que la coma se encuentre en una posición fija, de ahí la **denominación de coma fija**.

Modificar la posición de la coma tiene un efecto de multiplicación por un factor que es potencia de dos, negativa  $2^{-n}$  o positiva  $2^n$ , según se desplace  $n$  posiciones a la izquierda o  $m$  posiciones a la derecha respectivamente.

## PROBLEMA RESUELTO 1-25

Representar el número  $12,25_{(10)}$  en binario con formato de coma fija. Desplazar, en la representación binaria, la coma una posición a derecha e izquierda y calcular los nuevos valores que toma en cada caso el número binario modificado.

### Solución:

$$12,25_{(10)} = 1100,01_{(2)}$$

Desplazando la coma  $m$  posiciones a la derecha el valor del número queda multiplicado por  $2^m$ . En este ejemplo  $m = 1$ , por tanto, el nuevo valor del número queda multiplicado por dos.

$$11000,1_{(2)} = 24,5_{(10)}$$

Desplazando la coma  $n$  posiciones a la izquierda el valor del número queda multiplicado por  $2^n$ . En este ejemplo  $n = 1$ , por tanto, el nuevo valor del número queda multiplicado por  $2^{-1}$ , o lo que es lo mismo, queda multiplicado por 0,50.

$$110,001_{(2)} = 6,125_{(10)}$$

Es importante tener en cuenta que cuando se representa un número real en el formato de coma fija, el **error absoluto** que se comete siempre será inferior a  $2^{-q}$ , siendo  $q$  el número de bits de la parte fraccionaria.

El formato de coma fija tiene la **ventaja** de requerir sistemas digitales simples, pero el **inconveniente** de permitir un rango de valores muy limitado, en la representación de números reales. Una solución, aunque poco empleada, para aumentar el rango de representación, es decir, para representar números más largos en coma fija, es dividirlos en porciones de una palabra (número de bits que pueden ser tratados en su conjunto o simultáneamente en una operación del sistema digital, siendo los más corrientes: 8, 16, 32 o 64 bits, según el sistema), cada una de las cuales se almacenan en posiciones contiguas de memoria, procesándose secuencialmente.

### Ejemplo:

El número  $-1846384741_{(10)}$ , es equivalente al número (representado en complemento a dos)  $10010001111100100110011110011011_{(2)}$ , este resultado se puede obtener o comprobar haciendo uso de la calculadora de Windows, introduciendo primero el número decimal  $-1846384741_{(10)}$  y activando posteriormente la opción **Bin** de representación binaria, según se muestra en la Figura 1.20.

Para almacenar este número de 32 bits en una memoria típica organizada en bytes (8 bits), se debe descomponer en cuatro bytes, almacenándose en posiciones consecutivas, de la forma:



10011011  
01100111  
11110010  
10010001

Este bloque de cuatro bytes representa al número original. Tiene el inconveniente de que para operar con este número se necesitan cuatro lecturas de memoria. Además, el máximo rango de representación en convenio de complemento a dos, para números almacenados en cuatro bytes es el comprendido entre  $[-2^{n-1}, 2^{n-1}-1]$ , donde  $n = 32$ , siendo por tanto el límite o rango de representación  $[-2^{31}, 2^{31}-1] = [-2147483648, 2147483647]$ .



Figura 1.20. Conversión a binario, en convenio de complemento a dos, del número decimal -1846384741, utilizando la calculadora de Windows

Una mejor solución empleada en la mayoría de los sistemas digitales, para la representación de números reales, es utilizar una notación científica denominada **coma flotante**. Esta notación permite aumentar el rango de valores, en la representación de números reales, a cambio de una mayor complejidad y coste de los circuitos del sistema digital.

### 1.2.5.1 COMA FLOTANTE

Los sistemas digitales operan con números binarios, almacenados en palabras de memoria o registros de microprocesadores, de tamaño demasiado pequeño como para

poder representar números reales en coma fija. Esto limita considerablemente el rango de representación de los números reales.

La **representación de números en coma flotante** es equivalente a la denominada notación científica, utilizada normalmente para representar tanto cantidades numéricas muy grandes como muy pequeñas, y por lo tanto, aumentando ampliamente el rango de representación de los reales en los sistemas digitales.

### Ejemplo:

La representación del número cuatro billones quinientos veinte mil millones, en coma fija es:

4 520 000 000 000

y representado, de forma más abreviada, mediante notación científica, es:

$4,52 \cdot 10^{12}$

De la misma forma la representación del número cuatro coma cincuenta y dos billonésimas, en coma fija es:

0,000 000 000 004 52

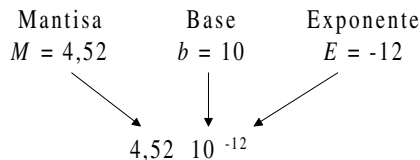
y representado, de forma más abreviada, mediante notación científica, es:

$4,52 \cdot 10^{-12}$

En la notación en coma flotante, un número tiene tres componentes: una mantisa  $M$ , un exponente  $E$  y una base  $b$ .

### Ejemplo:

Así en el ejemplo anterior cuyo número estaba representado por  $4,52 \cdot 10^{-12}$ , las tres componentes de la notación en coma flotante son:



Generalizando, un número  $N$  en coma flotante tiene la siguiente composición:

$$N = S M (b)^E \quad [1.31]$$

donde,

$S$ : es el signo del número;

$M$ : es el valor absoluto de la mantisa;

$E$ : es el valor del exponente, y,

$b$ : es la base del sistema de numeración utilizado.

Las cantidades numéricas representadas en coma flotante se almacenan mediante dos números de coma fija, en uno de ellos se representa el signo y la mantisa del número y en el otro el exponente. Observando la expresión [1.31], la base  $b$ , único dato que falta para componer el número  $N$  en coma flotante, se considera implícito en los circuitos de cálculo digitales, siendo en los sistemas binarios igual a dos o potencia de dos.

Dependiendo del número de dígitos que puedan almacenar estos dos números de coma fija se tendrá mayor **precisión** cuantos más dígitos tenga la mantisa y mayor **rango de representación** cuantos más dígitos tenga el exponente.

La representación de un número en coma flotante no es única, pues depende del lugar donde se coloque la coma en la mantisa.

### Ejemplo:

El número  $6,25_{(10)}$  representado en coma flotante (en base binaria), puede escribirse de formas diferentes:

$$0110,01 \cdot 2^{000}$$

$$011,001 \cdot 2^{001}$$

$$01,1001 \cdot 2^{010}$$

$$0,11001 \cdot 2^{011}$$

Para unificar la representación de los números en coma flotante se aplica el convenio de situar la coma en una posición fija de la mantisa. Además, para obtener una mayor precisión se almacenan en la mantisa el mayor número posible de dígitos significativos, mediante un proceso de **normalización de la mantisa**. El método más extendido de normalización consiste en calcular la mantisa fraccionaria de un número distinto de cero comprendida entre  $0,1 \geq |M| \geq 1$ , estando estos límites expresados en binarios. Este formato se caracteriza por situar la coma de la mantisa delante de la primera cifra significativa distinta de cero, es decir, la mantisa es un número real cuyo valor es inferior a 1 y cuya primera cifra después de la coma siempre es distinta de cero (excepto cuando el número es cero).

Todo proceso de normalización requiere el ajuste del exponente para que el valor del número no se vea modificado.

El cero es un caso especial en el proceso de normalización puesto que no tiene ninguna primera cifra significativa distinta de cero, además la expresión,

$$0 = 0 \cdot 2^E$$

se cumple con cualquier valor del exponente  $E$ . Esto implica que el cero tendrá una representación específica; se utiliza normalmente la convención de representarlo con una mantisa igual a cero y un exponente con el mayor valor negativo posible.

### Ejemplo:

Normalización de mantisas:

Números	Números con mantisa normalizada
$110,01 \cdot 2^{000}$	$0,11001000 \cdot 2^{011}$
$0,00110011 \cdot 2^{011}$	$0,11001100 \cdot 2^{001}$
	<div style="border-top: 1px solid black; width: 100px; margin: 0 auto; position: relative;"> <span style="position: absolute; top: -5px; left: 50%; transform: translate(-50%, -50%); font-size: small;">Mantisa de 8 bits</span> </div>

Aunque existen numerosos convenios para el **almacenamiento de números binarios en coma flotante**, es usual situar las informaciones más significativas en cabeza, es decir, primero el signo  $S$ , posteriormente el exponente  $E$  y, por último, la mantisa  $M$ .

Un formato de coma flotante muy extendido en los sistemas digitales es el estándar **IEEE 754**, propuesto por el *Institute of Electrical and Electronics Engineers* para la representación de números de 32 bits. Su formato está compuesto por:

- un bit de signo  $S$ , que es el signo de la mantisa,
- el campo del exponente  $E$  de 8 bits (incluido implícitamente el signo del exponente), y,
- el campo de la mantisa  $m$  de 23 bits;

tal como se muestra en la Figura 1.21.

Al tener la mantisa normalizada, el bit más significativo siempre es igual a 1, dado que ( $M = 1,m$ ), este bit no es necesario almacenarlo en memoria (pues siempre se guardaría el mismo valor 1); sin embargo, los circuitos digitales que posteriormente procesen el número deberán restaurarlo.

Los números binarios  $E$  y  $M$  están representados en coma fija. La mantisa  $M$  representa una fracción binaria, cuyo rango está comprendido entre  $1,00\dots0$  y  $1,11\dots1$ . La mantisa  $M$  junto con el signo  $S$ , forman un número con representación en signo-magnitud.

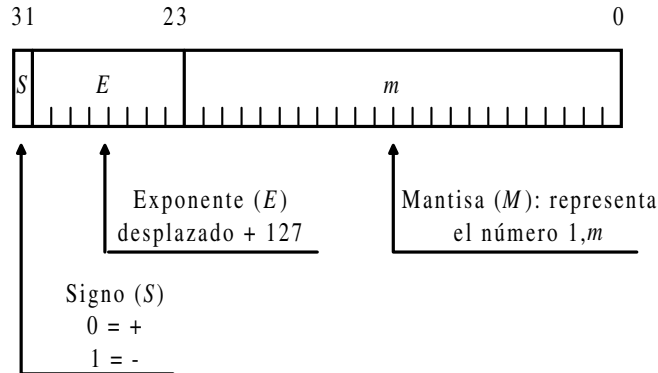


Figura 1.21. Formato de almacenamiento de números binarios en coma flotante, según el estándar IEEE 754

El campo del exponente  $E$  representa un entero positivo sin signo que al restarle 127, se obtiene el verdadero exponente  $E-127$ , de ahí que se llame **exponente en exceso** o también **exponente desplazado**, siendo 127 el valor del desplazamiento o exceso. El valor del exponente verdadero varía entre los valores  $-127$  y  $+128$  (implícitamente se almacena el signo del exponente). El hecho de almacenar en el campo del exponente un entero positivo simplifica algunas operaciones que se realizan con el mismo.

Resumiendo, un número  $N$ , definido mediante una palabra de 32 bits, o de **simple precisión**, según el estándar IEEE 754, cumple la expresión [1.32].

$$N = (-1)^S \cdot 2^{E-127} \cdot (1,m) \quad [1.32]$$

El estándar IEEE 754 también establece un formato de 64 bits, o de **doble precisión**, 11 bits en el campo del exponente y 52 bits en el campo de la mantisa, cumpliéndose la expresión [1.33].

$$N = (-1)^S \cdot 2^{E-1023} \cdot (1,m) \quad [1.33]$$

## PROBLEMA RESUELTO 1-26

Representar el número  $-6,125_{(10)}$  según el estándar IEEE 754 con formato de 32 bits.

### Solución:

El número  $-6,125_{(10)}$  se representa, su valor positivo en binario, como  $6,125_{(10)} = 110,001$ . Para normalizar la mantisa se debe desplazar a este número binario dos lugares a la izquierda (división de la mantisa entre  $2^2 = 4$ ) obteniéndose





- mediante rutinas de software que realizan operaciones en coma flotante, o,
- mediante hardware implementado en el procesador (circuitos aritméticos que realizan las operaciones en coma flotante). Existen, comercialmente, procesadores aritméticos que constituyen un medio eficiente, sencillo y económico de dotar a un sistema del hardware de coma flotante. Un ejemplo de estos sistemas son los DSPs denominados Procesadores Digitales de Señal.

### PROBLEMA RESUELTO 1-27

Determinar el valor decimal del número siguiente expresado en el formato binario de coma flotante de 32 bits, según el estándar IEEE 754:

0000 0000 0001 1101 0100 0000 0000 0000

#### Solución:

El número representado en coma flotante es un caso especial de los representados en la Tabla 1.12, donde el exponente  $E = 0$  y la mantisa  $M \neq 0$ . Aplicando la expresión [1.34] correspondiente a este caso especial, se obtiene:

$$\begin{aligned} N &= (-1)^0 \cdot 2^{-126} \cdot (0,001110101_{(2)}) = 1,175494 \cdot 10^{-38} \cdot 0,228515 = \\ &= 2,686188 \cdot 10^{-39} \end{aligned}$$

### PROBLEMAS PROPUESTOS

- 1-11) Expresar en el formato binario de 8 bits en signo-magnitud los números decimales:  
-25, 85,-100 y 125.
- 1-12) Expresar en el formato binario de 8 bits en complemento a uno los números decimales:  
-27, 88,-99 y 126.
- 1-13) Expresar en el formato binario de 8 bits en complemento a dos los números decimales:  
-10, 78,-101 y 123.
- 1-14) Determinar el valor decimal de cada uno de los siguientes números expresados en el formato de signo-magnitud:  
10010011, 01011010, 11111110 y 00000111.
- 1-15) Determinar el valor decimal de cada uno de los siguientes números expresados en el formato del convenio de complemento a uno:  
10010011, 01011010, 11111110 y 00000111.



1-16) Determinar el valor decimal de cada uno de los siguientes números expresados en el formato del convenio de complemento a dos:

10010011, 01011010, 11111110 y 00000111.

1-17) Expresar en el formato binario de coma flotante de 32 bits, según el estándar IEEE 754, los números decimales:

$-1023 \cdot 10^{-24}$ ;  $78,545 \cdot 10^{16}$  y  $-123,25$ .

1-18) Determinar el valor decimal de los números siguientes expresados en el formato binario de coma flotante de 32 bits, según el estándar IEEE 754:

0100 0100 1000 1100 0000 0000 0000 0000

0111 1111 1100 0000 0000 0000 0000 0000

0111 1111 1000 0000 0000 0000 0000 0000

1111 1111 1000 0000 0000 0000 0000 0000

0000 0000 0000 0000 0000 0000 0000 0000

0000 0000 0000 1100 0000 0000 0000 0000

# CODIFICACIÓN DE LA INFORMACIÓN

---

---

***Objetivos:***

- Utilizar los códigos de numeración más empleados para almacenar y transmitir información (BCD, Gray, Johnson, ASCII, etc.) y los códigos de detección y corrección de errores.

***Contenido:*** En este capítulo se exponen: la representación de la información mediante el conocimiento de las propiedades y principales aplicaciones de los códigos binarios.

***Simulación:*** Este capítulo no tiene ejercicios de simulación por su carácter teórico basado en definiciones, propiedades y representación de los principales códigos binarios.

## 2.1 DEFINICIONES Y PROPIEDADES DE LA CODIFICACIÓN



En este capítulo se da la definición de código. Se estudian los códigos que, al cumplir determinadas propiedades, tienen ciertas aplicaciones, particularizando en el caso de los códigos binarios (codificación binaria). Por último se trata el problema de la detección y corrección de errores en la transmisión digital.

### 2.1.1 Definiciones

La codificación de la información es una necesidad que surge como consecuencia del estudio de la naturaleza de la información y de su transmisión (comunicación).

La **información** es todo aquello que es captado por los sentidos y llega al cerebro, produciendo un incremento de nuestros conocimientos o una reacción.

El funcionamiento de las sociedades animales y humanas es posible gracias a la **comunicación**. Ésta se define como un acto mediante el cual un individuo establece con otro, u otros, un contacto que le permite transmitirles una información. Para que la comunicación se produzca, es necesario un código, es decir, un conjunto limitado y moderadamente extenso de símbolos que se combinan mediante ciertas reglas, conocidas por el emisor y el receptor. Se denomina **símbolo** a un objeto material (audible, visible, etc.) que representa a otro objeto (material o inmaterial), y que se utiliza para recibir, conservar o transmitir una información relativa al objeto representado.

Se entiende por **código** la correspondencia que asigna a cada símbolo  $\{F_1, F_2, \dots, F_r\}$ , de un alfabeto dado, denominado **alfabeto fuente**, una determinada combinación de símbolos  $\{C_1, C_2, \dots, C_n\}$ , de otro alfabeto, denominado **alfabeto código**, y viceversa.

A cada secuencia de símbolos que se pueden formar con el alfabeto código se le llama **palabra código**. Al número de símbolos  $C_i$  que contiene la palabra código se le llama **longitud de la palabra**. Y al número de los distintos símbolos que componen el alfabeto código se le llama **base del código**.

#### Ejemplo:

Si el alfabeto fuente está formado por los símbolos  $\{\alpha, \beta, \chi, \delta\}$  y el alfabeto código por  $\{0, 1\}$ , un posible código, con palabras de longitudes dos y tres simultáneamente, y con base de código dos, es el representado en la Tabla 2.1.

La aplicación que hace corresponder a cada símbolo del alfabeto fuente  $F_i$  con una palabra código se denomina **codificación**. Y al proceso inverso, mediante el cual se obtiene un símbolo del alfabeto fuente  $F_i$  conociendo la correspondiente palabra

código se denomina **decodificación**. En el ejemplo anterior,  $\delta$  se codifica con la palabra código 010, e inversamente, la decodificación de la palabra código 010 es  $\delta$ .

*Tabla 2.1. Ejemplo de código*

Símbolos fuente		Palabras código
$\alpha$	→	01
$\beta$	→	10
$\chi$	→	101
$\delta$	→	010

## 2.1.2 Propiedades de interés de los códigos

Dada la gran cantidad de códigos diferentes que se pueden crear atendiendo a la definición general de código efectuada anteriormente, se limita el estudio a aquellos códigos que presenten ciertas propiedades suplementarias que les hacen útiles.

Estas propiedades son:

- **UNIFORMIDAD:** Un código es uniforme si a cada símbolo fuente le corresponde una palabra código. A los códigos que cumplen esta propiedad también se les denomina código bloque.
- **NO SINGULARIDAD:** Un código uniforme es no singular si a cada símbolo fuente le corresponde palabras de código distintas.

### Ejemplo:

El código representado en la Tabla 2.1 y en la Tabla 2.2, cumple las dos propiedades anteriores, son uniformes y no singulares.

*Tabla 2.2. Ejemplo de código uniforme y no singular*

Símbolos fuente		Palabras código
$\alpha$	→	0
$\beta$	→	1
$\chi$	→	00
$\delta$	→	11

Las palabras que resultan de codificar todas las posibles parejas de símbolos del alfabeto fuente de la Tabla 2.2 están representadas en la Tabla 2.3. A este código resultante se le denomina **extensión de orden dos** del código de partida.

Se aprecia que aunque todas las palabras de código de partida son diferentes, como se puede ver en la Tabla 2.2, es posible encontrar una secuencia de palabras código en la Tabla 2.3 con origen indefinido. Así, por ejemplo, la secuencia 111 puede corresponder a la pareja de símbolos fuente  $\beta\delta$  o a  $\delta\beta$ . De esto se deduce que el código de partida, aun siendo no singular, da origen a secuencias de símbolos del alfabeto código iguales. Es decir, se producen extensiones de código singulares.

Tabla 2.3. Extensión de orden dos de un código

Símbolos fuente	Palabras código	Símbolos fuente	Palabras código
$\alpha\alpha$	00	$\chi\alpha$	000
$\alpha\beta$	01	$\chi\beta$	001
$\alpha\chi$	000	$\chi\chi$	0000
$\alpha\delta$	011	$\chi\delta$	0011
$\beta\alpha$	10	$\delta\alpha$	110
$\beta\beta$	11	$\delta\beta$	111
$\beta\chi$	100	$\delta\chi$	1100
$\beta\delta$	111	$\delta\delta$	1111

Lo anterior demuestra que en una transmisión digital no se puede obtener, de forma unívoca, la secuencia de símbolos fuente partiendo de la secuencia de símbolos código recibidos y cuestiona la necesidad de definir una nueva condición, más restrictiva que la uniformidad y la no singularidad para que el código sea utilizable, llamada:

- **DECODIFICACIÓN UNÍVOCA:** Un código es unívocamente decodificable si, y sólo si, su extensión de orden  $n$  es no singular para cualquier valor finito  $n$ .

Esta propiedad asegura que dos secuencias cualesquiera de símbolos fuente, de la misma longitud, dan lugar a secuencias distintas de símbolos de código. De la definición de esta propiedad, se deduce que también será necesario que la condición se cumpla en secuencias de símbolos fuente de longitud distinta.

### Ejemplos:

En la Tabla 2.4 se representan tres ejemplos de códigos unívocamente decodificables.

Tabla 2.4. Ejemplos de códigos unívocamente decodificables

Símbolos fuente	Código A	Código B	Código C
$\alpha$	00	0	0
$\beta$	01	10	01
$\chi$	10	110	011
$\delta$	11	1110	0111

El código *A* es la forma más sencilla y utilizada de crear códigos unívocamente decodificables. Este código es no singular y con palabras de longitud fija. Estas dos propiedades son una condición suficiente para garantizar la decodificación unívoca.

El código *B* también es unívocamente decodificable. Es no singular y aunque no tiene una longitud fija, el símbolo 0 separa una palabra de la siguiente, por lo que implícitamente se conoce la longitud de cada palabra. Este tipo de códigos reciben el nombre de códigos coma. Así, el símbolo coma puede interpretarse como el lugar donde termina una palabra y comienza la siguiente.

El código *C* es también unívocamente decodificable. Se diferencia de los códigos *A* y *B* en el siguiente aspecto importante: si se reciben secuencias binarias en el código *C* no se pueden identificar las palabras de la secuencia según se van recibiendo los dígitos binarios. Por ejemplo al recibir 01, primero se recibe el 0 y posteriormente el 1, no pudiéndose asegurar que el símbolo fuente es  $\beta$  mientras no se reciba el bit siguiente. Si el bit siguiente es 0 efectivamente el símbolo fuente es  $\beta$  y si es 1 hay que esperar a otro bit para asegurar que el símbolo fuente es  $\chi$ , puesto que si el bit es 1, se ha recibido la secuencia (0111) y el símbolo fuente correspondiente sería  $\delta$ .

Por consiguiente, el código *C* sufre un retraso de un dígito en el proceso de decodificación, a diferencia de los códigos *A* y *B* que permiten la decodificación, sin retraso, según se recibe la secuencia de bits.

La diferencia que presenta el código *C* con respecto a los códigos *A* y *B*, denota la necesidad de identificarla definiendo una nueva propiedad.

- **DECODIFICACIÓN INSTANTÁNEA:** Se denomina instantáneo, a un código unívocamente decodificable, cuando éste permite decodificar sin ambigüedad las palabras contenidas en una secuencia de símbolos del alfabeto código, sin necesitar el conocimiento de los símbolos que les suceden.

### Ejemplo:

En la Tabla 2.4 los códigos *A* y *B* son instantáneos y el *C*, aunque es unívoco, no es instantáneo.

No siempre es sencillo determinar cuándo un código permite decodificación instantánea, como en los casos anteriores de los códigos:  $A$ ,  $B$  y  $C$ . Es necesario disponer de una regla general que permita fácilmente identificar a los códigos instantáneos.

Sea una palabra de código cualquiera  $C_1 C_2 C_3 \dots C_s$ . Se denomina **prefijo** de esta palabra a una secuencia de símbolos  $C_1 C_2 C_3 \dots C_i$ , donde  $i$  es menor o igual que  $s$ .

**Ejemplo:**

La palabra 1010 tiene cuatro prefijos: 1010, 101, 10 y 1.

La **condición necesaria y suficiente** para que un **código sea instantáneo** es que ninguna de las palabras sea prefijo de otra.

**Ejemplo:**

A partir de la Tabla 2.4 se calculan los prefijos de cada palabra código. Para los códigos  $A$  y  $B$  se prueba fácilmente que ninguna de sus palabras código son prefijos del resto de palabras código. Así lo muestran la Tabla 2.5 y la Tabla 2.6 respectivamente.

*Tabla 2.5. Código A y sus prefijos*

Código $A$	Prefijos de $A$		
00	00	0	
01	01	0	
10	10	1	
11	11	1	

*Tabla 2.6. Código B y sus prefijos*

Código $B$	Prefijos de $B$			
0	0			
10	10	1		
110	110	11	1	
1110	1110	111	11	1

Sin embargo, en la Tabla 2.7, que representa el código  $C$  y sus prefijos, se aprecia cómo la palabra código 0 es prefijo de todas las demás palabras código; la palabra 01 es prefijo de la 011 y 0111; la palabra código 0111 es prefijo de la palabra código 01111. Por lo que este código no es instantáneo, como ya se había observado anteriormente.

Tabla 2.7. Código  $C$  y sus prefijos

Código $C$	Prefijos de $C$			
0	0			
01	01	0		
011	011	01	0	
0111	0111	011	01	0

En la Figura 2.1 se resumen las propiedades de los códigos definidas en este apartado. Están representadas gráficamente las cinco diferentes subclases de códigos que corresponden cada una de ellas con las terminaciones de las ramificaciones.

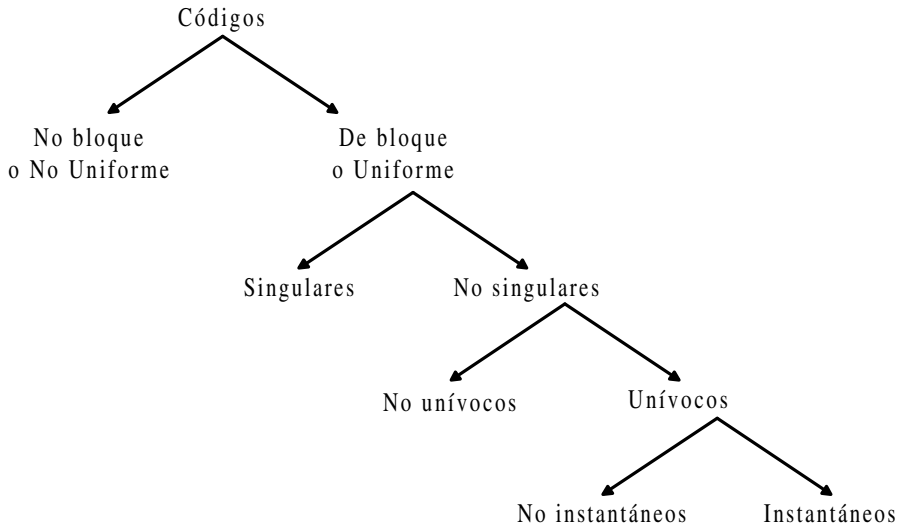


Figura 2.1. Resumen de las propiedades de los códigos



## 2.2 CÓDIGOS BINARIOS

Los códigos binarios se caracterizan por tener un alfabeto código fijo, formado por dos símbolos  $\{0, 1\}$ .

Los códigos que se estudian en este apartado son instantáneos de longitud de palabra fija. Ahora bien, los códigos instantáneos de longitud de palabra variable tienen una importante aplicación en la transmisión de señales, donde existen diferentes probabilidades de que aparezcan los distintos símbolos fuente. En estos casos, se asignan las palabras de código más cortas a los símbolos fuente más probables, minimizándose el número de símbolos a transmitir. Un ejemplo de código instantáneo de longitud variable es el código Morse.

En este apartado se estudian, principalmente, los códigos empleados en el proceso de información en vez de los utilizados para la transmisión, por lo que es más práctico utilizar códigos de longitud fija, siendo usual volver a codificar de nuevo esta información en códigos de longitud variable cuando se transmita la información.

Como la codificación consiste en establecer una correspondencia entre las palabras de código y los símbolos fuente, con un código binario de  $n$  bits ( $b_{n-1} b_{n-2} \dots b_1 b_0$ ) se pueden obtener  $2^n$  combinaciones (palabras de código) distintas. Lógicamente, cada combinación o palabra código se asigna a un símbolo del alfabeto fuente. Pero estas asignaciones, a su vez, se pueden realizar de diferentes formas, siendo el número posible de éstas las permutaciones de  $2^n$  combinaciones.

Cada forma de hacer las asignaciones dará lugar a un código diferente, es decir, habrá  $(2^n)(2^n-1)(2^n-2)\dots 1 = 2^n!$  posibles códigos.

### Ejemplo:

Para codificar el alfabeto fuente  $\{A, B, C, D\}$  con el alfabeto código  $\{0, 1\}$ , son necesarias palabras de longitud  $n = 2$ , ya que  $2^n = 2^2 = 4$ , siendo éste el número de símbolos del alfabeto fuente. El número de códigos distintos que se pueden formar es  $2^n! = 2^2! = 24$ , los cuales se representan en la Tabla 2.8.

*Tabla 2.8. Número de códigos distintos que se pueden formar para codificar un alfabeto fuente dado*

Símbolos fuente	Palabras código
A A A A A A B B B B B B C C C C C C D D D D D D	00
B B C C D D A A C C D D A A B B D D A A B B C C	01
C D B D B C C D A D A C B D A D A B B C A C A B	10
D C D B C B D C D A C A D B D A B A C B C A B A	11

## 2.2.1 Principales definiciones y propiedades de los códigos binarios

- **PONDERADOS:** Son aquellos códigos que a cada dígito binario se le asigna un peso y a cada palabra código la suma de los pesos de los dígitos binarios con valor uno, siendo el resultado igual al número decimal al que representan.

### Ejemplo:

Considerando que la palabra código 1101 está representada en el sistema binario, su valor decimal se obtiene al sumar los pesos:  $1 \cdot 8 + 1 \cdot 4 + 1 \cdot 1 = 13$ , representando dicha palabra código al símbolo fuente 13.

- La **DISTANCIA** entre dos palabras de código, se define como el número de dígitos que deben ser invertidos en una de ellas para obtener la otra.

### Ejemplo:

La distancia entre 1011 y 1100 es tres, ya que, las dos palabras de código se diferencian en tres bits.

- **DISTANCIA DEL CÓDIGO BINARIO:** se define como la menor de las distancias entre dos cualesquiera de sus palabras código.

Dos palabras de código son **adyacentes** si su distancia es uno, es decir, sólo difieren en un bit.

### Ejemplo:

La palabra código 1100 es adyacente a 1101.

- **CONTINUOS:** Son aquellos códigos cuyas palabras consecutivas son adyacentes; es decir, si dos cualesquiera de sus palabras de código consecutivas sólo difieren en un bit.
- **CÍCLICOS:** Son aquellos códigos que además de ser continuos, la primera y última palabra de código también son adyacentes.
- **DENSO:** Se define a un código como denso si teniendo una longitud de palabra de  $n$  bits está formado por  $2^n$  palabras de código.
- **AUTOCOMPLEMENTARIOS AL NÚMERO  $N$ :** Son aquellos códigos, cuya palabra de código y su complementada suman  $N$ . Los códigos con esta propiedad posibilitan efectuar más fácilmente las operaciones de resta mediante el complemento a  $N$ .

## 2.3 TIPOS

En los sistemas digitales se utilizan diversos códigos. Unos son estrictamente **numéricos** y otros son **alfanuméricos**.

### 2.3.1 Códigos numéricos

En este apartado se estudian los códigos numéricos más empleados para representar la información digital, como son: el código binario natural, los códigos BCD, los códigos continuos y cíclicos, el código Gray y el código Johnson.

#### 2.3.1.1 CÓDIGO BINARIO NATURAL

Este código representa los valores decimales en el sistema de base dos.

En la Tabla 2.9 se representa, a modo de ejemplo, el código binario natural, para el caso de longitud de palabra de cuatro bits.

*Tabla 2.9. Código binario natural de 4 bits*

Decimal	Binario natural				Decimal	Binario natural			
	$B_3$	$B_2$	$B_1$	$B_0$		$B_3$	$B_2$	$B_1$	$B_0$
0	0	0	0	0	8	1	0	0	0
1	0	0	0	1	9	1	0	0	1
2	0	0	1	0	10	1	0	1	0
3	0	0	1	1	11	1	0	1	1
4	0	1	0	0	12	1	1	0	0
5	0	1	0	1	13	1	1	0	1
6	0	1	1	0	14	1	1	1	0
7	0	1	1	1	15	1	1	1	1

En la Tabla 2.10 se resumen las principales propiedades del código binario natural.

El código binario natural es autocomplementario al número  $2^n-1$ , siendo  $n$  el número de bits empleados en la representación.

Tabla 2.10. Propiedades del código binario natural

Propiedad	Binario natural
Ponderado	Sí
Distancia de código	1
Continuo	No
Cíclico	No
Denso	Sí
Autocplementario	$2^n - 1$

**Ejemplo:**

Para  $n = 4$ , como ocurre en la Tabla 2.9, las palabras de código representado es autocplementario a  $2^n - 1 = 2^4 - 1 = 15$ . Así el complemento a 15 de 6, es:

$$15 - 6 = 9$$

y viceversa, el complemento a 15 de 9:

$$15 - 9 = 6$$

Si el código es autocplementario a  $N$ , basta cambiar, al número, los ceros por unos y unos por ceros para obtener su complementario. Esto puede apreciarse entre el número 6 y su complementario a 15, que es el 9.

$$\overbrace{0110}^6 \longleftarrow \text{Autocplementarios a 15} \longrightarrow \overbrace{1001}^9$$

**2.3.1.2 CÓDIGOS BCD**

En aplicaciones de introducción de información digital en forma decimal y en su visualización, como son por ejemplo los *displays*, resulta aconsejable, para simplificar los circuitos digitales, el empleo de códigos que representen por separado cada uno de los dígitos del número decimal. Este tipo de códigos se denomina decimales codificados en binario (*Binary Coded Decimal*, códigos BCD en lo sucesivo).

En estos códigos, se representan los diez guarismos 0, 1, 2, ..., 8 y 9 del sistema decimal mediante una cierta codificación binaria. El número de dígitos binarios

necesarios para la codificación es cuatro, pues con tres bits sólo se pueden representar del 0 al 7. Si bien, al utilizar cuatro bits se pueden formar  $2^4 = 16$  combinaciones, y únicamente se usan diez, existen seis combinaciones no utilizadas.

Los códigos BCD pueden ser: códigos ponderados y no ponderados.

Dentro de los códigos ponderados se pueden destacar el **BCD Natural** o **BCD 8421**. En este código cada dígito decimal, se representa mediante un código binario de cuatro bits, con asignación de pesos igual que en el sistema de numeración binario natural (pesos: 8, 4, 2, 1), de ahí que se denomine BCD natural o BCD 8421. En la Tabla 2.11, se representa este código.

*Tabla 2.11. Código BCD Natural o BCD 8421*

<b>Decimal</b>	<b>BCD Natural o BCD 8421</b>			
<b>Pesos →</b>	<b>8</b>	<b>4</b>	<b>2</b>	<b>1</b>
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

En la Tabla 2.12 se resumen las principales propiedades del código BCD Natural o BCD 8421.

Dentro de los códigos BCD y dependiendo del valor del peso asignado a cada bit, se definen los códigos **BCD Aiken 2421** y **BCD Aiken 5421**.

Se observa, que en principio, la representación de los números decimales en estos códigos no es única. Por ejemplo, para representar el número 6, en código Aiken 2421 se puede hacer de dos maneras diferentes, 1100 o 0110. Se adopta la asignación indicada en la Tabla 2.13, porque de esta forma el código Aiken 2421 es autocomplementario a nueve.

Tabla 2.12. Propiedades del código BCD Natural o BCD 8421

Propiedad	BCD Natural o BCD 8421
Ponderado	Sí
Distancia de código	1
Continuo	No
Cíclico	No
Denso	No
Autocplementario	No

El código Aiken 5421 al igual que el anterior código (Aiken 2421), asigna un cero al bit de mayor peso en los cinco primeros símbolos fuente (del 0 al 4) y un 1 a los cinco últimos símbolos fuente (del 5 al 9), tomando el resto de los bits el valor adecuado para que la suma de los pesos de la palabra código sea el número decimal. Así, la representación de estos códigos es única, como se muestra en la Tabla 2.13.

Tabla 2.13. Códigos BCD Aiken

Decimal	BCD Aiken 2421				BCD Aiken 5421			
	2	4	2	1	5	4	2	1
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	0
3	0	0	1	1	0	0	1	1
4	0	1	0	0	0	1	0	0
5	1	0	1	1	1	0	0	0
6	1	1	0	0	1	0	0	1
7	1	1	0	1	1	0	1	0
8	1	1	1	0	1	0	1	1
9	1	1	1	1	1	1	0	0

En la Tabla 2.14 se resumen las principales propiedades de los códigos BCD Aiken 2421 y BCD Aiken 5421.

Tabla 2.14. Propiedades de los códigos BCD Aiken

<b>Propiedad</b>	<b>BCD Aiken 2421</b>	<b>BCD Aiken 5421</b>
Ponderado	Sí	Sí
Distancia de código	1	1
Continuo	No	No
Cíclico	No	No
Denso	No	No
Autocplementario	a 9	No

Se puede demostrar que es condición necesaria para que un código BCD ponderado sea autocplementario a nueve que la suma de los pesos de sus dígitos sea igual a nueve. Esta condición se puede comprobar en los códigos representados en la Tabla 2.14. Existen cuatro posibles códigos BCD ponderados autocplementarios cuya distribución de pesos es: 2421, 3321 4311 y 5211. Si se consideran pesos positivos y negativos, se pueden formar trece códigos BCD autocplementarios, como por ejemplo, el BCD 642-3, representado en la Tabla 2.15.

Tabla 2.15. Código BCD 642-3

<b>Decimal</b>	<b>BCD 642-3</b>			
<b>Pesos →</b>	<b>6</b>	<b>4</b>	<b>2</b>	<b>-3</b>
0	0	0	0	0
1	0	1	0	1
2	0	0	1	0
3	1	0	0	1
4	0	1	0	0
5	1	0	1	1
6	0	1	1	0
7	1	1	0	1
8	1	0	1	0
9	1	1	1	1

En la Tabla 2.16 se resumen las principales propiedades del código BCD 642-3.

Tabla 2.16. Propiedades del código BCD 642-3

Propiedad	BCD 642-3
Ponderado	Sí
Distancia de código	1
Continuo	No
Cíclico	No
Denso	No
Autocplementario	a 9

El **código BCD de exceso 3** se forma sumando tres a cada palabra del código BCD natural, por lo que no será ponderado y sí autocplementario a nueve, como puede verse en la Tabla 2.17.

Tabla 2.17. Código BCD de exceso 3

Decimal	BCD de exceso 3
0	0 0 1 1
1	0 1 0 0
2	0 1 0 1
3	0 1 1 0
4	0 1 1 1
5	1 0 0 0
6	1 0 0 1
7	1 0 1 0
8	1 0 1 1
9	1 1 0 0

En la Tabla 2.18 se resumen las principales propiedades del código BCD de exceso 3.

El código BCD de exceso 3, al ser autocplementario, permite ejecutar más fácilmente las operaciones de resta aplicando el procedimiento del complemento a nueve.



Tabla 2.18. Propiedades de los códigos BCD de exceso 3

Propiedad	BCD de exceso 3
Ponderado	No
Distancia de código	1
Continuo	No
Cíclico	No
Denso	No
Autocplementario	a 9

La **conversión de un número decimal a código BCD** se realiza expresando cada dígito decimal mediante la combinación binaria correspondiente del código BCD elegido.

La **conversión del código BCD a un número decimal** se realiza dividiendo el número, a partir de la coma, en grupos de cuatro bits, expresando en cada grupo su valor decimal correspondiente del código BCD elegido.

### Ejemplo:

La representación del número decimal 37,6 en el código BCD natural es:

$$\begin{array}{ccc} \overbrace{0011}^3 \overbrace{0111}^7, \overbrace{0110}^6 & \text{Decimal} \\ 00110111,0110 & \text{BCD natural} \end{array}$$

El valor decimal del código BCD natural: 1001010011,011 es:

$$\begin{array}{ccc} \overbrace{1001}^2 \overbrace{0100}^5 \overbrace{11,01}^3 \overbrace{10}^6 & \text{BCD natural} \\ 1001010011,0110 & \text{Decimal} \end{array}$$

## PROBLEMA RESUELTO 2-1

Representar el número decimal 127,25 en los códigos BCD natural, Aiken 2421, exceso 3 y en binario natural.

### Solución:

En la tabla siguiente se muestra la representación en los cuatro códigos.

Código	Palabras código
BCD natural	0001 0010 0111,0010 0101
BCD Aiken 2421	0001 0010 1101,0010 1011
BCD exceso 3	0100 0101 1010,0101 1000
Binario natural	1111111,01

### PROBLEMA RESUELTO 2-2

Determinar el número decimal del código: 0100 0101 1000,0011 cuando está expresado en: BCD natural, Aiken 2421, exceso 3 y en binario natural.

#### Solución:

Código	Valor decimal
BCD natural	458,3
BCD Aiken 2421	-----
BCD exceso 3	125,0
Binario natural	1112,1875

En el caso de la obtención del número decimal del código: 0100 0101 1000,0011 expresada en BCD Aiken 2421, las palabras código 0101 y 1000 no pertenecen al alfabeto de dicho código, por lo que este apartado del problema no tiene solución, indicándose este hecho mediante una línea de trazos.

La **ventaja** que presentan los códigos BCD, como ya se ha indicado anteriormente, es que al efectuarse codificaciones independientes para cada dígito, se facilita la conversión decimal-binario. Por el contrario, la **desventaja** que presentan es que se necesitan más bits para ser representados. Así, una palabra de  $n$  bits puede representar en binario natural  $2^n$  números distintos. En cambio en BCD, con  $n$  bits, sólo es posible representar  $10^{n/4} = 2^{0,8305 \cdot n}$  números distintos. Por esta misma razón se requieren circuitos más complejos cuando se opera con códigos BCD.

### 2.3.1.3 CÓDIGOS CONTINUOS Y CÍCLICOS

Los códigos cíclicos por definición son continuos, por lo que garantizan que entre dos palabras de código adyacentes solamente cambiará un bit, lo cual evita la aparición de palabras transitorias de código debidas a la imposibilidad de conmutación de dos o más dígitos. Por ejemplo, al pasar de la palabra código 000 a la 011, puede

que se realice este paso, siguiendo la secuencia:  $000 \rightarrow 010 \rightarrow 011$  o mediante  $000 \rightarrow 001 \rightarrow 011$ . Entre la palabra código inicial y la final, aparece una palabra de código intermedia o transitoria, debido a la conmutación que produce perturbaciones y efectos perjudiciales en el buen funcionamiento de los sistemas digitales. Por ello en muchas aplicaciones, como codificadores de posición angular (*encoders* ópticos), se emplean códigos cíclicos, ya que éstos sólo cambian un bit entre posiciones adyacentes, eliminándose la posibilidad de que aparezcan palabras código transitorias erróneas que den lugar a interpretaciones incorrectas del posicionamiento angular.

Una de las aplicaciones importantes de estos códigos está en los sistemas de conversión de digital a analógico y de analógico a digital.

Dos de los códigos continuos y cíclicos más empleados son el código Gray y el código Johnson.

### 2.3.1.4 CÓDIGO GRAY

El código Gray es uno de los códigos cíclicos más usados. También recibe el nombre de **código reflejado**, debido al reflejo que se debe realizar en las palabras código al construirlo. La formación del código Gray de  $n$  bits se realiza por reflexión del código de  $n-1$  bits, repitiendo simétricamente las combinaciones de éste y añadiendo a la izquierda un bit, que será cero en las  $2^{n-1}$  primeras palabras código (primera mitad de las filas) y un uno en las  $2^{n-1}$  filas restantes (última mitad de las filas), que es la parte reflejada, como puede verse en Tabla 2.19.

En la Tabla 2.20 se representan las palabras del código Gray de cuatro bits.

En la Tabla 2.21 se resumen las principales propiedades del código Gray.

Tabla 2.19. Construcción del código Gray o código reflejado

1 bit	2 bits	3 bits
0	0 0	0 0 0
1	0 1	0 0 1
	1 1	0 1 1
	1 0	0 1 0
		1 1 0
		1 1 1
		1 0 1
		1 0 0

Tabla 2.20. Código Gray de cuatro bits

Decimal	Código Gray	Decimal	Código Gray
0	0 0 0 0	8	1 1 0 0
1	0 0 0 1	9	1 1 0 1
2	0 0 1 1	10	1 1 1 1
3	0 0 1 0	11	1 1 1 0
4	0 1 1 0	12	1 0 1 0
5	0 1 1 1	13	1 0 1 1
6	0 1 0 1	14	1 0 0 1
7	0 1 0 0	15	1 0 0 0

Tabla 2.21. Propiedades del código Gray

Propiedad	Código Gray
Ponderado	No
Distancia de código	1
Continuo	Sí
Cíclico	Sí
Denso	Sí
Autocplementario	No

Por ser cíclico, el código Gray se utilizará para la simplificación de funciones lógicas en capítulos posteriores.

Además de ser continuo y cíclico, el código Gray utiliza al máximo la capacidad de codificación de los dígitos que lo forman y, por lo tanto, también es denso. Asimismo su conversión a binario y viceversa es muy simple.

La **conversión del código binario natural a Gray** se realiza teniendo presente que:

- El bit más significativo (MSB) en el código Gray,  $G_{n-1}$ , y en el código Binario natural,  $B_{n-1}$ , son idénticos.
- El resto de los bits del código Gray,  $G_i$ , se obtienen sumando de izquierda a derecha cada par adyacente de los bits en código binario  $B_i$  y  $B_{i+1}$ , siendo su resultado el siguiente bit del código Gray, descartando los acarreos; o

mediante la función lógica **XOR** (cuyo símbolo es  $\oplus$ ). Esta operación da como resultado un uno lógico cuando sus operandos son distintos y un cero lógico cuando son iguales (en capítulos posteriores se estudiará esta función con más detalle).

$$G_i = B_i \oplus B_{i+1}$$

donde:

$$0 \leq i \leq n-2,$$

$$B_{n-1} = G_{n-1},$$

si  $B_i \neq B_{i+1}$  entonces  $G_i = 1$  y

si  $B_i = B_{i+1}$  entonces  $G_i = 0$ .

### Ejemplo:

Conversión del número 101011, en código binario natural, a código Gray.

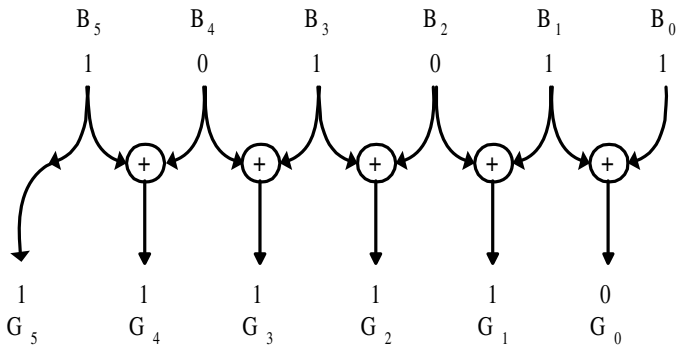


Figura 2.2. Ejemplo de conversión de código binario natural a Gray

La **conversión de código Gray a binario natural** se realiza teniendo presente que:

Los bits más significativos (MSB) en el código binario natural  $B_{n-1}$  y en el código Gray  $G_{n-1}$  son idénticos.

El resto de los bits del código binario natural  $B_i$  se obtienen sumando de izquierda a derecha a cada bit del código generado  $B_{i+1}$  el bit en código Gray  $G_i$  de la siguiente posición adyacente y descartando los acarrees; o mediante la función lógica XOR (cuyo símbolo es  $\oplus$ ), anteriormente señalada.

$$B_i = B_{i+1} \oplus G_i$$

donde:

$$0 \leq i \leq n-2,$$

$$B_{n-1} = G_{n-1},$$

si  $B_{i+1} \neq G_i$  entonces  $B_i = 1$  y

si  $B_{i+1} = G_i$  entonces  $B_i = 0$ .

**Ejemplo:**

Conversión del número 111110, en código Gray, a código binario natural.

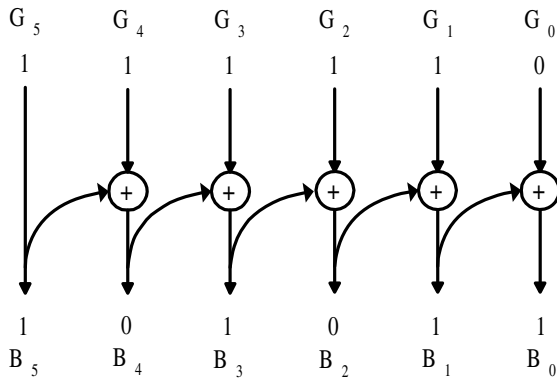


Figura 2.3. Ejemplo de conversión de código Gray a binario natural

**2.3.1.5 CÓDIGO JOHNSON**

El código Johnson es continuo y cíclico. Este código recibe también el nombre de **código progresivo**, debido a que el número de unos aumenta y disminuye progresivamente de una combinación a la siguiente. En la Tabla 2.22 se muestra el código Johnson de cinco bits.

El código Johnson, además de ser continuo y cíclico, presenta la **desventaja** de tener una capacidad de codificación para  $n$  bits de tan sólo  $2 \cdot n$  símbolos fuentes distintos, por lo que no es denso. Por ejemplo, en la Tabla 2.22, se representa este código con cinco bits pudiéndose sólo codificar  $2 \cdot n = 10$  símbolos fuentes distintos (los números decimales del 0 al 9).

En la Tabla 2.23 se resumen las principales propiedades del código Johnson.

Una **ventaja** que presenta el código Johnson es que es muy fácil de generar mediante circuitos digitales.

*Tabla 2.22. Código Johnson*

<b>Decimal</b>	<b>Código Johnson</b>
0	0 0 0 0 0
1	0 0 0 0 1
2	0 0 0 1 1
3	0 0 1 1 1
4	0 1 1 1 1
5	1 1 1 1 1
6	1 1 1 1 0
7	1 1 1 0 0
8	1 1 0 0 0
9	1 0 0 0 0

*Tabla 2.23. Propiedades del código Johnson*

<b>Propiedad</b>	<b>Código Johnson</b>
Ponderado	No
Distancia de código	1
Continuo	Sí
Cíclico	Sí
Denso	No
Autocplementario	No

En la Tabla 2.24 se muestra, a modo de resumen, las principales propiedades de los códigos estudiados anteriormente, pudiendo verse las similitudes y deferencias entre los mismos.

Tabla 2.24. Resumen de las propiedades de los códigos

Propiedad	Códigos					
	Binario natural	BCD			Gray	Johnson
		Natural 8421	Aiken 2421	Exceso 3		
Ponderado	Sí	Sí	Sí	No	No	No
Distancia código	1	1	1	1	1	1
Continuo	No	No	No	No	Sí	Sí
Cíclico	No	No	No	No	Sí	Sí
Denso	Sí	No	No	No	Sí	No
Autocplementario	$2^n-1$	No	a 9	a 9	No	No

## 2.3.2 Códigos alfanuméricos

Estos códigos se caracterizan porque permiten representar tanto números como caracteres alfabéticos. También suelen incluir caracteres especiales y de control, necesarios, estos últimos, para la transferencia de información.

Entre los diversos códigos alfanuméricos existentes, son de destacar el código EBCDIC (*Extended BCD Interchange Code*) y el código ASCII (*American Standard Comitee on Information Interchange*), estando el último adoptado internacionalmente como código alfanumérico estándar para el intercambio de información entre sistemas digitales.

### 2.3.2.1 CÓDIGO ASCII

El código alfanumérico ASCII, tiene palabras código de  $n = 7$  bits, pudiendo por lo tanto representar  $2^n = 2^7 = 128$  símbolos fuente distintos, los cuales corresponden a: 26 letras mayúsculas, 26 letras minúsculas, 10 dígitos decimales, signos de puntuación, caracteres especiales y caracteres de control no imprimibles (utilizados para la comunicación y el control de los sistemas digitales).

En la Tabla 2.25 se muestra el código ASCII de 7 bits, siendo LSD el dígito hexadecimal menos significativo (*Least Significant Digit*) y MSD el dígito hexadecimal más significativo (*Most Significant Digit*).



Tabla 2.25. Código alfanumérico ASCII

LSD \ MSD	0	1	2	3	4	5	6	7	
	000	001	010	011	100	101	110	111	
0	0000	NUL	DLE	(Esp)	0	@	P	'	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	EXT	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	\$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	_	o	(Borr)

Las dos primeras columnas, de código, de la citada Tabla 2.25 corresponden a los caracteres de control, cuyos significados son:

NUL: Nulo.	VT: Tabulación vertical.	SYN: Sincronización.
SOH: Inicio encabezamiento.	FF: Pasar página.	ETB: Fin bloque transmisión.
STX: Inicio de texto.	CR: Retorno de carro.	CAN Anulación.
EXT: Fin de texto.	SO: Fuera de código.	EM: Fin de soporte.
EOT: Fin de transmisión.	SI: En el código.	SUB: Sustituir.
ENQ: Petición.	DLE: Eludir la transmisión.	ESC: Escape.
ACK Acuse de recibo.	DC1: Control dispositivo 1.	FS: Separador archivo.
BEL: Campanilla.	DC2: Control dispositivo 2.	GS: Separador equipo.
BS: Backspace.	DC3: Control dispositivo 3.	RS: Separador registro.
HT: Tabulación horizontal.	DC4: Control dispositivo 4.	US: Separador de unidad.
LF: Pasar línea.	NAK: Acuse recibo negativo.	

Actualmente se utiliza el **código ASCII extendido** de ocho bits, el cual dispone de un segundo mapa de 128 palabras de código, las cuales corresponden a símbolos gráficos y a los códigos del alfabeto internacional. Esta extensión del código ASCII

permite la compatibilidad con los registros de ocho bits utilizados ampliamente en los sistemas digitales.

### 2.3.3 Códigos detectores de error

Cuando se transmite información digital por un medio físico, tal como cable, radio, fibra óptica, etc., se pueden producir **errores** (recibir información distinta a la transmitida), debido a la presencia de ruido, interferencias electromagnéticas, fallo de componentes, falsos contactos, etc. En estos casos, interesa conocer cuándo se ha producido el error, e incluso, una vez detectado el dato erróneo, llegar a corregirlo. Para ello se utilizan códigos especiales que permiten resolver este problema.

En un código denso no es posible la detección de un error, porque una palabra código, después del error, se transforma en otra palabra que también pertenece al código. Por tanto, para detectar errores en el control, proceso y transmisión digital es necesario que las palabras código no presenten todas las posibles combinaciones. Esta **condición**, es **necesaria**, pero **no suficiente**, ya que la probabilidad de que se produzcan dos o más errores simultáneamente (cambio de dos o más bits), aun no siendo cero, es muy reducida. Considerando por tanto los errores simples o con mayor probabilidad de producirse, en los que sólo se modifica un bit, la **condición necesaria y suficiente** para que un código binario permita detectar errores en un bit es que su distancia sea superior a la unidad. Para ello, se añade **información redundante** (bits de chequeo) a la palabra a transmitir aumentando su distancia. En general, para poder ser detectados  $E$  errores simultáneos, la distancia mínima del código será  $E + 1$ .

Entre los diversos códigos detectores de errores, se destacan los **códigos de paridad** y los **códigos de peso fijo de palabra** (entendiendo por **peso de una palabra de código** el número de unos lógicos que contiene).

#### 2.3.3.1 CÓDIGOS DE PARIDAD

Se define la **paridad** de una combinación o palabra de código binario, como el número de unos que contiene. Si el número de unos es par, la configuración tendrá **paridad par** y en caso contrario, tendrá **paridad impar**.

Los códigos de paridad se forman partiendo de cualquier código (denominado código base) cuya distancia mínima sea uno. A cada combinación del código base se le añade un bit llamado **bit de paridad**. El bit de paridad toma un valor tal que hace que el número total de unos en el grupo sea siempre par o impar. Si se desea obtener un código de paridad par, dicho bit será tal que el número de unos en cada palabra del nuevo código sea par. Por el contrario, para obtener un código de paridad impar, dicho bit será tal que el número de unos en cada palabra del nuevo código sea impar.

En la Tabla 2.26 se muestra cómo a partir del código base, BCD natural, se obtiene el código de paridad, de distancia mínima dos, capaz de detectar errores en un bit.

*Tabla 2.26. Ejemplo de código de paridad correspondiente al código base BCD natural*

Dígito decimal	Código BCD natural	Bit de paridad impar	Bit de paridad par
0	0000	1	0
1	0001	0	1
2	0010	0	1
3	0011	1	0
4	0100	0	1
5	0101	1	0
6	0110	1	0
7	0111	0	1
8	1000	0	1
9	1001	1	0

La detección de errores requiere que el transmisor genere el código de paridad, a partir del código base, añadiendo el bit de paridad (par o impar) y enviando esta información por el medio de transmisión. El receptor, en el otro extremo del medio de transmisión, debe comprobar si la paridad se mantiene igual a la prefijada en el transmisor (par o impar), detectando el error cuando ésta no se cumpla.

### PROBLEMA RESUELTO 2-3

Determinar en las siguientes palabras con código de paridad par cuáles son erróneas:

- a) 11011
- b) 1110011

#### Solución:

- a) La palabra código tiene paridad par, siendo correcta.
- b) La palabra código tiene un número impar de unos, siendo errónea.

### PROBLEMA RESUELTO 2-4

Añadir un bit de paridad a la derecha de las palabras de código del problema anterior para formar codificaciones de paridad impar.

**Solución:**

110111

11100110

**2.3.3.2 CÓDIGOS DE PESO FIJO**

Entre los **códigos detectores de error de peso fijo** más representativos, cabe destacar el código **2 entre 5** y el código **biquinario**. Este último es ponderado y consta de dos partes, una de dos bits y otra de cinco bits, de ahí su nombre.

Los dos códigos indicados se caracterizan por tener una distancia de código igual a dos (lo que permite la detección de un bit de error) y todas sus palabras código tienen exactamente dos unos (paridad par), como se muestra en la Tabla 2.27.

*Tabla 2.27. Códigos detectores de error de palabra fija: 2 entre 5 y biquinario*

<b>Dígito decimal</b>	<b>Código 2 entre 5</b>	<b>Código biquinario</b>
		(Pesos) → <b>50 43210</b>
0	01100	01 00001
1	11000	01 00010
2	10100	01 00100
3	10010	01 01000
4	01010	01 10000
5	00110	10 00001
6	10001	10 00010
7	01001	10 00100
8	00101	10 01000
9	00011	10 10000

**2.3.4 Códigos correctores de error**

Los **códigos correctores de error** además de detectar la presencia de un error, proporcionan información, indicando los bits en los que se ha producido el error. Por tanto, una vez identificados los bits erróneos basta con invertir su valor (si es 0 se pone 1, y viceversa) y así obtener el valor correcto de los datos. Estos códigos se utilizan principalmente en la transmisión de información, y en especial en aquellos

casos donde la transmisión se realiza una sola vez, existiendo la imposibilidad de volver a repetirla cuando se detecta que se ha producido el error. Tal es el caso de los **sistemas que trabajan en tiempo real**, en los que la información que se transmite es utilizada por el sistema receptor en el mismo instante en el que se recibe.

Los códigos de distancia dos estudiados anteriormente (en el Apartado 2.3.3 Códigos detectores de error), no permiten la corrección de errores, ya que al producirse un error simple de un bit, la combinación obtenida posee dos palabras código adyacentes pertenecientes al código, no pudiendo conocer de cuál de las dos procede.

### Ejemplo:

Considerando el código BCD natural con bit de paridad impar, añadido a la derecha (representado en la Tabla 2.26), si se produce una combinación, como por ejemplo: 00011, ésta se puede detectar como errónea ya que tiene un número par de unos. Ahora bien, no es posible conocer qué bit es el erróneo ya que al invertir cualquiera de ellos se obtiene una palabra de código correcta. Por ejemplo, si se considera como bit erróneo el representado en negrita: 000**1**1, la combinación correcta, una vez corregida, sería 00010; pero también se hubiera podido considerar como bit erróneo, al bit siguiente (representado en negrita): 000**1**1 y la combinación correcta, una vez corregida, hubiera sido otra, 00001.

La **condición necesaria y suficiente** para que un código permita corregir errores en un bit es que la distancia mínima debe ser superior a dos. Así, por ejemplo, si un código es de distancia mínima tres, cualquier combinación que contenga un bit erróneo es adyacente a una sola combinación del código, y esto permite conocer cuál es el bit incorrecto. En general, para que un código permita corregir  $F$  bits erróneos simultáneos, la distancia mínima debe ser igual a  $2 \cdot F + 1$ .

Existen numerosos códigos correctores de errores, como lo son, los de paridad bidimensional, códigos cíclicos, etc., siendo uno de los más utilizados el **código Hamming**.

Los principios básicos para la **construcción de un código Hamming** que sea capaz de corregir errores de un bit, partiendo de un código de  $n$  bits de distancia unidad, son los siguientes:

- A cada palabra de  $n$  dígitos ( $B_n \dots B_2 B_1$ ) se le añaden  $k$  dígitos más ( $D_k \dots D_2 D_1$ ), generados a partir de los  $n$  primeros, formando una palabra código de longitud  $n + k$ .
- Los  $k$  dígitos añadidos se generan de forma que  $k$  tests de paridad, elegidos convenientemente, devuelvan una palabra binaria ( $T_k \dots T_2 T_1$ ) que indique la posición del bit erróneo o un cero cuando no se produzca error.
- Al ser la palabra transmitida de longitud  $n + k$  y existir la posibilidad de error en cualquiera de los  $n + k$  dígitos, más la posibilidad de ausencia de error, se

deben considerar  $n + k + 1$  casos. Dado que con la palabra de test ( $T_k \dots T_2 T_1$ ), se deben codificar los  $n + k + 1$  casos posibles, se tiene que cumplir la inecuación [2.1]:

$$2^k \geq n + k + 1 \quad [2.1]$$

Un código de control de paridad de Hamming se dice que es **óptimo** cuando cumple la igualdad [2.2]:

$$2^k = n + k + 1; \quad 2^k - 1 = n + k \quad [2.2]$$

### Ejemplo:

Construcción de un código Hamming a partir del código BCD natural.

Sea el código BCD natural cuya palabra código está compuesta por los bits  $B_4 B_3 B_2 B_1$ . En este caso la longitud de palabra de código es  $n = 4$ , y por tanto para que se cumpla la expresión [2.1], se toma  $k = 3$  (o mayor). El código resultante tiene una longitud de palabra  $n + k$  de siete bits y además es óptimo al cumplir la ecuación  $n + k = 2^{k-1} - 1$ , o sea,  $7 = n + k = 2^{3-1} - 1$ . En la Tabla 2.28 se establece el valor que ha de tomar la palabra resultante de los tests de paridad ( $T_3 T_2 T_1$ ) en función de la posición del error.

*Tabla 2.28. Palabra de test de paridad, en función de la posición del error, para el código Hamming*

Posición errónea ( $P_i$ )	Test de paridad ( $T_j$ )		
	$T_3$	$T_2$	$T_1$
0 (sin error)	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Se denomina  $P_i$  (para  $i = 1, 2, \dots, 7$ ) a cada una de las posiciones binarias de las palabras del código Hamming, de siete bits,  $(P_7 P_6 \dots P_2 P_1)$ . Considerando sólo la posibilidad de que se produzca error en un único bit. De la Tabla 2.28 se deduce que el bit de test de paridad  $T_1$  toma el valor 1 si se produce error en la posición  $P_1, P_3, P_5$  o  $P_7$ . El bit de test de paridad  $T_2$  toma el valor 1 si se produce error en la posición  $P_2, P_3, P_6$  o  $P_7$ . Y por último, el bit de test de paridad  $T_3$  toma el valor 1 si se produce error en la posición  $P_4, P_5, P_6$  o  $P_7$ .

Se observa cómo las posiciones  $P_1, P_2$  y  $P_4$  aparecen sólo una vez en una de las listas  $(P_1 P_3 P_5 P_7)$ ,  $(P_2 P_3 P_6 P_7)$  y  $(P_4 P_5 P_6 P_7)$  asociadas a cada bit de test  $T_j$  de paridad. Por lo tanto se pueden situar en estas posiciones los bits añadidos  $(C_3 C_2 C_1)$  y generarlos de modo que la paridad de las listas, indicadas anteriormente y asociadas a cada bit de test de paridad, sea siempre par.

Las palabras del código Hamming resultante tendrán la forma:

$$B_4 B_3 B_2 C_3 B_1 C_2 C_1$$

y los dígitos añadidos  $(C_3 C_2 C_1)$  se obtendrán mediante un **generador Hamming**, cuya descripción es la siguiente:

- **Generación de  $C_1$ .** El bit añadido  $C_1$  tomará un valor tal que el conjunto  $(P_7, P_5, P_3$  y  $C_1)$  o lo que es lo mismo, los bits  $(B_4, B_2, B_1$  y  $C_1)$  tengan paridad par.
- **Generación de  $C_2$ .** El bit añadido  $C_2$  tomará un valor tal que el conjunto  $(P_7, P_6, P_3$  y  $C_2)$  o lo que es lo mismo, los bits  $(B_4, B_3, B_1$  y  $C_2)$  tengan paridad par.
- **Generación de  $C_3$ .** El bit añadido  $C_3$  tomará un valor tal que el conjunto  $(P_7, P_6, P_5$  y  $C_3)$  o lo que es lo mismo, los bits  $(B_4, B_3, B_2$  y  $C_3)$  tengan paridad par.

De esta forma la generación de  $(C_3 C_2 C_1)$  asegura la paridad par (triple generación de paridad par) de los grupos de posiciones mencionados anteriormente.

Como se estudiará posteriormente, los bits  $(C_3 C_2 C_1)$  se pueden generar mediante puertas lógicas XOR (cuyo símbolo es  $\oplus$ ). Siendo su expresión lógica:

#### Generador Hamming:

$$C_1 = P_7 \oplus P_5 \oplus P_3 = B_4 \oplus B_2 \oplus B_1$$

$$C_2 = P_7 \oplus P_6 \oplus P_3 = B_4 \oplus B_3 \oplus B_1$$

$$C_3 = P_7 \oplus P_6 \oplus P_5 = B_4 \oplus B_3 \oplus B_2$$

Los bits de test de paridad, que indican la posición del bit erróneo (ver Tabla 2.28), se obtendrán por triple detección de paridad par, mediante un circuito o bloque denominado **corrector Hamming** cuya descripción es la siguiente:

- **Obtención de  $T_1$ .** El bit de test de paridad  $T_1$  tomará un valor tal que el conjunto ( $P_7, P_5, P_3, P_1$  y  $T_1$ ) o lo que es lo mismo, los bits ( $B_4, B_2, B_1, C_1$  y  $T_1$ ) tengan paridad par.
- **Obtención de  $T_2$ .** El bit de test de paridad  $T_2$  tomará un valor tal que el conjunto ( $P_7, P_6, P_3, P_2$  y  $T_2$ ) o lo que es lo mismo, los bits ( $B_4, B_3, B_1, C_2$  y  $T_2$ ) tengan paridad par.
- **Obtención de  $T_3$ .** El bit de test de paridad  $T_3$  tomará un valor tal que el conjunto ( $P_7, P_6, P_5, P_4$  y  $T_3$ ) o lo que es lo mismo, los bits ( $B_4, B_3, B_2, C_3$  y  $T_3$ ) tengan paridad par.

Como se estudiará posteriormente, los bits de test de paridad ( $T_3, T_2, T_1$ ) se pueden obtener mediante puertas lógicas XOR (cuyo símbolo es  $\oplus$ ). Siendo su expresión lógica:

#### Corrector Hamming:

$$T_1 = P_7 \oplus P_5 \oplus P_3 \oplus P_1 = B_4 \oplus B_2 \oplus B_1 \oplus C_1$$

$$T_2 = P_7 \oplus P_6 \oplus P_3 \oplus P_2 = B_4 \oplus B_3 \oplus B_1 \oplus C_2$$

$$T_3 = P_7 \oplus P_6 \oplus P_5 \oplus P_4 = B_4 \oplus B_3 \oplus B_2 \oplus C_3$$

En la Tabla 2.29 se muestra el código Hamming que resulta al aplicar las expresiones anteriores a los bits ( $B_4, B_3, B_2, B_1$ ) del código BCD natural. Se comprueba en dicha tabla que dicho código Hamming tiene distancia tres.

#### Ejemplo:

Para comprobar el funcionamiento del código corrector Hamming. Supóngase que al transmitir el número 6 cuyo código Hamming es 0110011 se produce un error en el bit de posición tres, por lo que la información recibida es:

Posiciones	$P_7$	$P_6$	$P_5$	$P_4$	$P_3$	$P_2$	$P_1$
Mensaje recibido	0	1	1	0	1	1	1

Aplicando las condiciones del circuito corrector Hamming, se obtienen los bits de test siguientes:

Dado ( $P_7, P_5, P_3, P_1, T_1$ ) = (0111X), para que el conjunto tenga paridad par,

$$T_1 = 1.$$

Dado ( $P_7, P_6, P_3, P_2, T_2$ ) = (0111X), para que el conjunto tenga paridad par,

$$T_2 = 1.$$



Dado  $(P_7 P_6 P_5 P_4 T_3) = (0110X)$ , para que el conjunto tenga paridad par,  
 $T_3 = 0$ .

*Tabla 2.29. Código corrector de error Hamming generado a partir del código BCD natural*

Decimal Posiciones →	Código Hamming						
	$P_7$ $B_4$	$P_6$ $B_3$	$P_5$ $B_2$	$P_4$ $C_3$	$P_3$ $B_1$	$P_2$ $C_2$	$P_1$ $C_1$
0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1
2	0	0	1	1	0	0	1
3	0	0	1	1	1	1	0
4	0	1	0	1	0	1	0
5	0	1	0	1	1	0	1
6	0	1	1	0	0	1	1
7	0	1	1	0	1	0	0
8	1	0	0	1	0	1	1
9	1	0	0	1	1	0	0

El valor señalado por los bits de test, representa la posición  $(T_3 T_2 T_1) = (011)$ ; es decir, el bit de la posición tercera del mensaje recibido (0110111) se debe invertir, obteniéndose el mensaje correcto 0110011 del dígito decimal 6.

Si se hubiera recibido un mensaje sin error, el bloque corrector de Hamming hubiera entregado los bits de test  $(T_3 T_2 T_1) = (000)$ , significando mensaje correcto.

### PROBLEMA RESUELTO 2-5

Determinar si el dato 0011110, recibido en código Hamming, es correcto o bien corregirlo si es necesario.

#### Solución:

Posiciones	$P_7$	$P_6$	$P_5$	$P_4$	$P_3$	$P_2$	$P_1$
Mensaje recibido	0	0	1	1	1	1	0

Los bits de test de paridad serán:

$(P_7 P_5 P_3 P_1 T_1) = (0110X)$ , para que el conjunto tenga paridad par,  $T_1 = 0$ .

$(P_7 P_6 P_3 P_2 T_2) = (0011X)$ , para que el conjunto tenga paridad par,  $T_2 = 0$ .

$(P_7 P_6 P_5 P_4 T_3) = (0011X)$ , para que el conjunto tenga paridad par,  $T_3 = 0$ .

El obtener los bits de test  $(T_3 T_2 T_1) = (000)$ , significa que la palabra código recibida es correcta.

## PROBLEMA RESUELTO 2-6

Determinar, en el problema anterior, las palabras de código originales.

**Solución:**

Posiciones	$P_7$	$P_6$	$P_5$	$P_4$	$P_3$	$P_2$	$P_1$
Significado de los bits	$D_4$	$D_3$	$D_2$	$C_3$	$D_1$	$C_2$	$C_1$
Mensaje recibido	0	0	1	1	1	1	0

El dato original transmitido es  $(D_4 D_3 D_2 D_1) = 0011$

## PROBLEMAS PROPUESTOS

2-1) Convertir a código Gray los números binarios:

1011

1100101

1110001110

2-2) Convertir a binario los números en código Gray:

1011

1100101

1110001110

2-3) Convertir a decimal los números en código BCD de exceso 3:

1011

110011

10011001010,01

2-4) Convertir a código BCD de exceso 3 los números decimales:

5

99

199,05

2-5) Decodificar el siguiente mensaje codificado en ASCII:

10011110 1101001 11101110 1100101 11011100 0100000 0100010 0110001  
0100010

2-6) Escribir en hexadecimal el mensaje del problema anterior.

2-7) Convertir a código ASCII la siguiente sentencia de programa:

10 MOV A,#20H ;Valor inicial

2-8) Determinar los siguientes códigos de paridad par erróneos:

1011  
110011  
10011001000

2-9) Añadir un bit de paridad a la derecha de las palabras de código siguientes para formar codificaciones de paridad par.

01011  
11001  
1001100100

2-10) Determinar los siguientes códigos erróneos de paridad impar:

1011  
110011  
10011001000

2-11) Añadir un bit de paridad a la derecha de las palabras de código siguientes para formar codificaciones de paridad impar.

01011  
11001  
1001100100

2-12) De los datos siguientes recibidos en código Hamming (óptimo) detectar cuáles son erróneos y corregirlos si es necesario.

101  
100  
111  
000  
1001100  
1001000

2-13) Determinar, en el problema anterior, las palabras de código originales.

# ÁLGEBRA DE CONMUTACIÓN Y SU REPRESENTACIÓN

---

---

**Objetivos:**

- Estudiar el álgebra de Boole como herramienta matemática básica para el análisis y síntesis de circuitos digitales.
- Saber representar e interpretar las funciones lógicas mediante su expresión canónica y su tabla de verdad.

**Contenido:** Definiciones y teoremas del álgebra de Boole. Representaciones de funciones lógicas mediante su expresión canónica y su tabla de verdad.

**Simulación:** Mediante el programa de simulación *Electronics Workbench 5.0* se comprueban los principales teoremas y leyes del álgebra de Boole.

## 3.1 ÁLGEBRA DE BOOLE



En esta sección se realiza el estudio del álgebra de Boole, siendo ésta la herramienta matemática que posteriormente servirá de base en el análisis y síntesis de circuitos digitales.

El álgebra de Boole fue introducida por el matemático inglés George Boole en 1854, desarrollando un método simbólico para el análisis de la lógica humana en su tratado *An Investigation of the Laws of Thought*. Posteriormente, en 1939, Claude E. Shannon, en su tratado *A Symbolic Analysis of Relay and Switching Circuits*, aplicó el álgebra de Boole en el estudio de los circuitos eléctricos con dos estados posibles, denominados circuitos de conmutación. Estos estudios han proporcionado las bases matemáticas para el diseño de los circuitos básicos digitales.

### 3.1.1 Definición de álgebra de Boole

Una estructura matemática, como es el álgebra de Boole, se construye a partir de un conjunto de **elementos** sobre los que se definen unos **operadores** que permiten realizar operaciones en ellos, estableciendo unos **postulados** o **axiomas** que relacionan tanto al conjunto de elementos como al conjunto de operadores.

En cualquier estructura matemática, los postulados son las hipótesis iniciales que la definen y que no se demuestran. Estos postulados son el punto de partida para deducir los teoremas y propiedades de dicha estructura.

Se pueden utilizar diferentes conjuntos de postulados para definir un álgebra de Boole, aunque uno de los más utilizados es el propuesto por Huntington en 1904.

Para la construcción de un álgebra de Boole, se parte de una estructura algebraica  $(B, +, \cdot)$ , formada por un conjunto de elementos  $B$  y dos operaciones definidas en el mismo, denominadas  $+$  y  $\cdot$  (suma y producto). Se dice que es un álgebra de Boole si cumple los siguientes axiomas, también conocidos como **postulados de Huntington**:

**Postulado I.** El conjunto  $B$  es cerrado con respecto a las dos operaciones.

Es decir, se cumple que  $\forall a, b \in B$ :

$$\begin{aligned} a + b &\in B \\ a \cdot b &\in B \end{aligned} \quad [3.1]$$

**Postulado II.** Existe un **elemento identidad** en las dos operaciones.

En la operación  $+$  el elemento identidad es el 0 y en la operación  $\cdot$  es el 1, cumpliéndose que  $\forall a \in B$ :

$$\begin{aligned} a + 0 &= a \\ a \cdot 1 &= a \end{aligned} \tag{3.2}$$

**Postulado III.** Las dos operaciones cumplen la **propiedad conmutativa**.

Es decir, se cumple que  $\forall a, b \in B$ :

$$\begin{aligned} a + b &= b + a \\ a \cdot b &= b \cdot a \end{aligned} \tag{3.3}$$

**Postulado IV.** Cada operación es **distributiva** con respecto a la otra.

Es decir, se cumple que  $\forall a, b, c \in B$ :

$$\begin{aligned} a \cdot (b + c) &= (a \cdot b) + (a \cdot c) \\ a + (b \cdot c) &= (a + b) \cdot (a + c) \end{aligned} \tag{3.4}$$

**Postulado V.** Existe un **elemento complementario**.

Se cumple que  $\forall a \in B$  existe otro elemento de  $B$  llamado “complementario de  $a$ ” que se representa por  $\bar{a}$  (la línea horizontal indica **complemento** o **negación** de  $a$ ), siendo:

$$\begin{aligned} a + \bar{a} &= 1 \\ a \cdot \bar{a} &= 0 \end{aligned} \tag{3.5}$$

**Postulado VI.** **Número de elementos.**

En el conjunto  $B$  existen al menos dos elementos diferentes, cumpliéndose que  $\forall a, b \in B$ :

$$a \neq b \tag{3.6}$$

Se debe tener en cuenta la generalidad de este postulado, que sólo establece el número mínimo de elementos de  $B$ , no precisando ni su número total, ni el tipo de éstos.

### 3.2 TEOREMAS DEL ÁLGEBRA DE BOOLE

De los postulados anteriores se deducen un conjunto de propiedades del álgebra de Boole que se indican a continuación en forma de leyes y teoremas.

- **Principio de Dualidad.** Sea  $E$  una igualdad entre dos expresiones booleanas y  $E^D$  otra igualdad obtenida a partir de  $E$  intercambiado los operadores  $+$  y  $\cdot$ , y

los elementos de identidad 0 y 1. Si  $E$  es una identidad (igualdad que se verifica para cualquier valor de sus variables),  $E^D$ , denominada dual de  $E$ , también lo es.

**Nota:** El teorema del principio de dualidad es consecuencia de la simetría de los postulados con respecto a las dos operaciones  $+$  y  $\cdot$ , y a los dos elementos de identidad 0 y 1. Cada axioma se define doblemente mediante dos expresiones duales entre sí.

En los siguientes teoremas se omiten las demostraciones de una de las dos partes duales, dado que es fácil su obtención aplicando el principio de dualidad.

- **Ley de idempotencia.** Para cualquier elemento  $a$  en un álgebra de Boole, se verifica que:

$$\begin{aligned} a + a &= a \\ a \cdot a &= a \end{aligned} \quad \begin{array}{l} \\ \text{(identidad dual)} \end{array} \quad [3.7]$$

Demostración:

$$\begin{aligned} a + a &= (a + a) \cdot 1 = && \text{(postulado II)} \\ &= (a + a) \cdot (a + \bar{a}) = && \text{(postulado V)} \\ &= a + (a \cdot \bar{a}) = && \text{(postulado IV)} \\ &= a + 0 = && \text{(postulado V)} \\ &= a && \text{(postulado II)} \end{aligned}$$

La segunda expresión de este teorema se demuestra igualmente utilizando los postulados duales, como ya se ha indicado en el principio de dualidad. Se desarrolla a continuación este proceso, demostrando la segunda expresión (identidad dual) del teorema de idempotencia (expresión [3.7]), para que sirva de ejemplo.

$$\begin{aligned} a \cdot a &= (a \cdot a) + 0 = && \text{(postulado II)} \\ &= (a \cdot a) + (a \cdot \bar{a}) = && \text{(postulado V)} \\ &= a \cdot (a + \bar{a}) = && \text{(postulado IV)} \\ &= a \cdot 1 = && \text{(postulado V)} \\ &= a && \text{(postulado II)} \end{aligned}$$

Observese cómo a partir de una de las expresiones ya demostradas de la ley de idempotencia, si se intercambian los operadores  $+$  y  $\cdot$ , y los elementos de

identidad 0 y 1, se obtiene la demostración de la otra expresión dual de la ley de idempotencia.

- **Operaciones con elementos identidad.** Para cualquier elemento  $a$  en un álgebra de Boole, se cumple que:

$$\begin{aligned} a + 1 &= 1 \\ a \cdot 0 &= 0 \end{aligned} \quad \text{(identidad dual)} \quad [3.8]$$

Demostración:

$$\begin{aligned} a + 1 &= (a + 1) \cdot 1 = && \text{(postulado II)} \\ &= (a + 1) \cdot (a + \bar{a}) = && \text{(postulado V)} \\ &= a + (1 \cdot \bar{a}) = && \text{(postulado IV)} \\ &= a + \bar{a} = && \text{(postulado II)} \\ &= 1 && \text{(postulado V)} \end{aligned}$$

- **TEOREMA:** El complemento de cada elemento es único.

Demostración: Si  $a + b = 1$  y  $a \cdot b = 0$  (aplicación del postulado V), entonces  $b = \bar{a}$ .

$$\begin{aligned} \bar{a} &= \bar{a} + 0 = && \text{(postulado II)} \\ &= \bar{a} + a \cdot b = && \text{(hipótesis)} \\ &= (\bar{a} + a) \cdot (\bar{a} + b) = && \text{(postulado IV)} \\ &= 1 \cdot (\bar{a} + b) = && \text{(postulado V)} \\ &= (a + b) \cdot (\bar{a} + b) = && \text{(hipótesis)} \\ &= (a \cdot \bar{a}) + b = && \text{(postulado IV)} \\ &= 0 + b = && \text{(postulado V)} \\ &= b && \text{(postulado II)} \end{aligned}$$

- **Ley de involución.** Para todo elemento  $a$  en un álgebra de Boole, se verifica:

$$\overline{\bar{a}} = a \quad [3.9]$$

Demostración: Por el postulado V se sabe que  $a + \bar{a} = 1$  y  $a \cdot \bar{a} = 0$ , lo que permite definir el complemento de  $a$ . De dicho postulado se deduce que el complemento de  $a$  es  $\bar{a}$  y de la misma manera el complemento de  $\bar{a}$  es  $\overline{(\bar{a})}$ .

Como el complemento es único, se deduce que  $\overline{(\bar{a})} = a$ .



- **Ley de absorción.** Para cada par de elementos  $a$  y  $b$  de un álgebra de Boole se verifica que:

$$\begin{aligned} a + a \cdot b &= a \\ a \cdot (a + b) &= a \end{aligned} \quad \text{(identidad dual)} \quad [3.10]$$

Demostración:

$$\begin{aligned} a + a \cdot b &= (a \cdot 1) + (a \cdot b) = && \text{(postulado II)} \\ &= a \cdot (1 + b) = && \text{(postulado IV)} \\ &= a \cdot (b + 1) = && \text{(postulado III)} \\ &= a \cdot 1 = && \text{(Expresión [3.8])} \\ &= a && \text{(postulado II)} \end{aligned}$$

- En el álgebra de Boole se verifica que:

$$\begin{aligned} a + (\bar{a} \cdot b) &= a + b \\ a \cdot (\bar{a} + b) &= a \cdot b \end{aligned} \quad \text{(identidad dual)} \quad [3.11]$$

Demostración:

$$\begin{aligned} a + (\bar{a} \cdot b) &= (a + \bar{a}) \cdot (a + b) = && \text{(postulado IV)} \\ &= 1 \cdot (a + b) = && \text{(postulado V)} \\ &= a + b && \text{(postulado II)} \end{aligned}$$

- En un álgebra de Boole las **operaciones + y  $\cdot$  son asociativas**. Para toda terna de elementos  $a$ ,  $b$  y  $c$  se verifica que:

$$\begin{aligned} a + (b + c) &= (a + b) + c \\ a \cdot (b \cdot c) &= (a \cdot b) \cdot c \end{aligned} \quad \text{(identidad dual)} \quad [3.12]$$

- **Leyes de De Morgan.** En un álgebra de Boole se verifica que:

$$\begin{aligned} \overline{a + b + c + d + \dots} &= \bar{a} \cdot \bar{b} \cdot \bar{c} \cdot \bar{d} \cdot \dots \\ \overline{a \cdot b \cdot c \cdot d \cdot \dots} &= \bar{a} + \bar{b} + \bar{c} + \bar{d} + \dots \end{aligned} \quad \text{(identidad dual)} \quad [3.13]$$

- **Teorema.** El complemento de una función se obtiene intercambiando las operaciones + y  $\cdot$ , y reemplazando cada variable por su complementario.

$$\overline{f(a, b, c, d, +, \cdot)} = f(\bar{a}, \bar{b}, \bar{c}, \bar{d}, \cdot, +) \quad [3.14]$$

- **Teorema de expansión de Shannon.** Toda función del álgebra de Boole se puede expresar de la siguiente forma:

$$f(\dots, d, c, b, a) = a \cdot f(\dots, d, c, b, 1) + \bar{a} \cdot f(\dots, d, c, b, 0) \quad [3.15]$$

Y su identidad dual:

$$f(\dots, d, c, b, a) = [a + f(\dots, d, c, b, 0)] \cdot [\bar{a} + f(\dots, d, c, b, 1)] \quad [3.16]$$

Una función se expande o desarrolla respecto de una variable cuando se aplica el teorema de expansión. Así, en las expresiones anteriores, la función se ha desarrollado respecto a la variable  $a$ . De igual forma se puede seguir expandiendo la función respecto a las demás variables. Se comprueba la igualdad de la expresión [3.15] haciendo  $a = 1$  y  $\bar{a} = 0$ . Para  $a = 0$  y  $\bar{a} = 1$  se comprueba la igualdad [3.16].

Las demostraciones de los últimos teoremas son bastante extensas y a la vez fáciles de comprobar mediante una tabla de verdad (como se verá posteriormente), por lo que se omiten en este apartado.

Para **evaluar una expresión del álgebra de Boole** se procede (al igual que en el álgebra ordinaria) de izquierda a derecha, realizando las operaciones según el siguiente orden: paréntesis, complemento, operador  $\cdot$  y por último el operador  $+$ .

Al **comparar el álgebra de Boole ( $B, +, \cdot$ ) con el cuerpo de los números reales ( $\mathfrak{R}, +, \cdot$ )**, se encuentran las siguientes diferencias:

- En los postulados del álgebra de Boole no se incluye la propiedad asociativa y, sin embargo, en los postulados de la estructura de cuerpo sí.
- En el álgebra de Boole la propiedad distributiva es doble. En la estructura de cuerpo solamente del operador  $\cdot$  respecto al operador  $+$ .
- En el álgebra de Boole se define un operador llamado complemento lógico que no existe en la estructura de cuerpo.
- El álgebra de Boole no tiene inversos aditivos ni multiplicativos y por lo tanto no tiene operaciones de sustracción ni división.



Utilizando el convertidor lógico del programa de simulación *Electronics Workbench*, se pueden comprobar los teoremas correspondientes a las expresiones [3.7], [3.9], [3.10], [3.11], [3.12] y [3.13].

Para realizar la simulación con la aplicación *Electronics Workbench*, se arrastra el convertidor lógico, situado en la barra de componentes e instrumentación del banco de instrumentos, al área de trabajo.

Posteriormente, haciendo doble clic sobre el icono del convertidor lógico se expande su carátula en la que se pueden ver las opciones disponibles para

realizar la conversión. En la Figura 3.1 se muestra dicho convertidor lógico colocado en el área de trabajo.

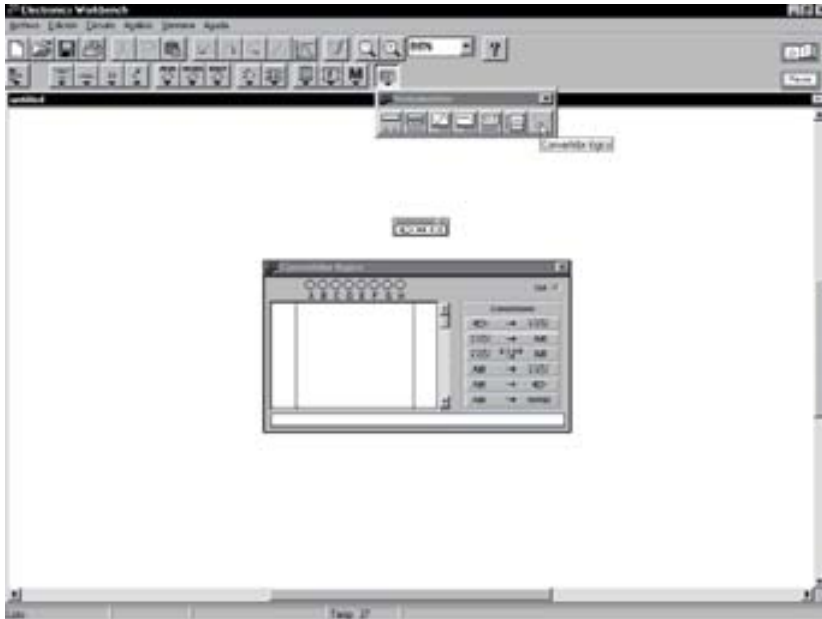


Figura 3.1. Obtención y colocación del convertidor lógico en el área de trabajo

El procedimiento de simulación a seguir es el siguiente:

- Se introduce en el convertidor lógico el primer término de la igualdad de cada una de las expresiones algebraicas. Para introducir el **complemento de una variable** se escribe después de ella el símbolo ' (comilla simple).
- Se realiza la conversión de expresión algebraica a tabla de verdad, cuyo icono es el siguiente:



- Se obtiene el resultado o segundo término de la igualdad al activar la opción de simplificación SIMP del convertidor, cuyo icono es el siguiente:



Siguiendo este procedimiento se comprueba a continuación una de las igualdades de la expresión [3.7] o ley de idempotencia, siendo ésta:

$$a + a = a$$

Se introduce  $A+A$  en el convertidor lógico y se activa la opción de conversión de expresión algebraica a tabla de verdad obteniéndose el resultado representado en la Figura 3.2.



Figura 3.2. Comprobación de la ley de idempotencia (expresión [3.7]) introduciendo primeramente  $A+A$  en el convertidor lógico

Posteriormente se comprueba el resultado del teorema, activando la opción de simplificación, como se muestra en la Figura 3.3.

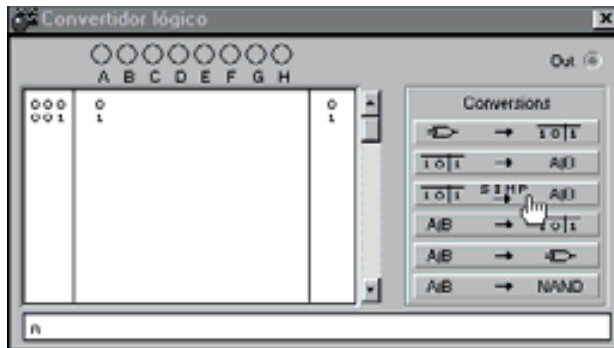


Figura 3.3. Comprobación de la ley de idempotencia al obtener el segundo término de la igualdad mediante la opción de simplificación del convertidor lógico

Siguiendo el mismo procedimiento se deja al lector que compruebe, mediante el programa de simulación *Electronics Workbench*, el resto de las expresiones algebraicas correspondientes a los teoremas propuestos.

Los pasos que deben realizarse son los siguientes:

- Para comprobar la segunda expresión (identidad dual) de la ley de idempotencia (expresión [3.7]) se introduce  $AA$  en el convertidor lógico y

se activa la opción de conversión de expresión algebraica a tabla de verdad y, posteriormente la opción de simplificación obteniendo el resultado A.

- Para comprobar la ley de involución (expresión [3.9]) se introduce  $A''$  (doble negación o complemento de  $a$  negada) en el convertidor lógico. Para ello se activa la opción de conversión de expresión algebraica a tabla de verdad y posteriormente la opción de simplificación, obteniendo el resultado A.
- Para comprobar la ley de absorción (expresión [3.10]) se introduce  $A+AB$  o su expresión dual  $A(A+B)$  en el convertidor lógico. Para ello se activa la opción de conversión de expresión algebraica a tabla de verdad y posteriormente la opción de simplificación, obteniendo el resultado A.
- Se puede comprobar la expresión [3.11] introduciendo  $A+(A'B)$  o su expresión dual  $A(A'+B)$  en el convertidor lógico. Se activa la opción de conversión de expresión algebraica a tabla de verdad y posteriormente la opción de simplificación, obteniendo los resultados  $A+B$  y  $AB$  respectivamente.
- Se puede comprobar que la operación  $+$  es asociativa introduciendo  $A+(B+C)$  y seleccionando la opción de conversión de expresión algebraica a tabla de verdad. Posteriormente se comprueba que la expresión  $(A+B)+C$  también tiene la misma tabla de verdad, verificando la igualdad de la expresión [3.12]. Se puede comprobar, por el anterior procedimiento que la operación  $\cdot$  es asociativa verificando si las expresiones  $A(BC)$  y  $(AB)C$  tienen las mismas tablas de verdad.
- Para comprobar las leyes de De Morgan (expresión [3.13]) se introduce  $(A+B+C+D)'$  en el convertidor lógico. Se activa la opción de conversión de expresión algebraica a tabla de verdad y posteriormente la opción de simplificación, obteniendo el resultado  $A'B'C'D'$ .

La otra expresión de las leyes de De Morgan,

$$\overline{a \cdot b \cdot c \cdot d \cdot \dots} = \bar{a} + \bar{b} + \bar{c} + \bar{d} + \dots$$

se comprueba introduciendo  $(A \cdot B \cdot C \cdot D)'$  en el convertidor lógico.

A continuación, se activa la opción de conversión de expresión algebraica a tabla de verdad, con lo que se obtiene el resultado que se muestra en la Figura 3.4.

Posteriormente, se activa la opción de simplificación SIMP comprobando el resultado de la ley de De Morgan, como se muestra en la Figura 3.5.



Figura 3.4. Comprobación de la ley de De Morgan (expresión [3.13]) introduciendo primeramente  $(A \cdot B \cdot C \cdot D)'$  en el convertidor lógico

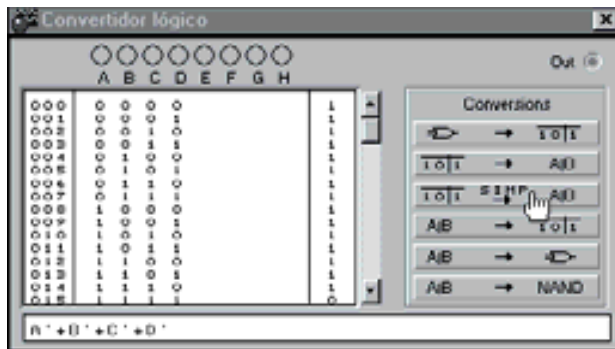


Figura 3.5. Comprobación de la ley de De Morgan al obtener el segundo término de la igualdad mediante la opción de simplificación del convertidor lógico

### 3.3 ÁLGEBRA DE BOOLE BIVALENTE

Dependiendo del conjunto  $B$  elegido y de cómo se especifiquen las operaciones  $+$  y  $\cdot$  se pueden definir numerosas álgebras de Boole. Entre ellas, la de mayor interés, en el diseño de circuitos digitales (desarrollada por Claude E. Shannon), es el **álgebra de Boole Bivalente o de conmutación**, denominada así por estar definida sobre un conjunto con dos elementos  $B = \{0, 1\}$  y las **operaciones suma lógica  $+$  y producto lógico  $\cdot$** , determinadas en la Tabla 3.1. En la Tabla 3.2 se incluye la operación de complemento definida en el postulado quinto.

Este tipo de tablas en las que se expresa, en cada fila, el valor que toma la expresión para cada una de las posibles combinaciones de valores de sus variables, se denomina **tabla de verdad**.

Tabla 3.1. Definición de las operaciones suma lógica  $+$  y producto lógico  $\cdot$ 

$a$	$b$	$a + b$	$a \cdot b$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Tabla 3.2. Complemento lógico

$a$	$\bar{a}$
0	1
1	0

Se demuestra que la estructura algebraica bivalente  $(B, +, \cdot)$ , desarrollada por Claude E. Shannon, es un álgebra de Boole, al cumplirse los seis postulados de Huntington, así:

- Se cumple el primer postulado ya que el conjunto  $B$  es cerrado para las dos operaciones definidas.
- Los postulados segundo y tercero se pueden comprobar directamente observando la Tabla 3.1 y la Tabla 3.2 mostradas anteriormente.
- El postulado cuarto, correspondiente a la ley distributiva de la operación  $\cdot$  sobre  $+$ , queda demostrado mediante la Tabla 3.3 (columnas sombreadas), comprobando que se cumple la expresión:  $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ .
- El postulado quinto, correspondiente a la existencia de un elemento complementario que verifique que  $a + \bar{a} = 1$  y que  $a \cdot \bar{a} = 0$ , queda demostrado en la Tabla 3.4.
- El postulado sexto se cumple al considerar la estructura algebraica bivalente formada por un conjunto  $B$  de dos elementos.

### PROBLEMA RESUELTO 3-1

Demostrar las leyes de De Morgan mediante las tablas de verdad, para funciones de dos variables.

a)  $\overline{a + b} = \bar{a} \cdot \bar{b}$

b)  $\overline{a \cdot b} = \bar{a} + \bar{b}$

**Solución:**

a)  $\overline{a+b} = \bar{a} \cdot \bar{b}$

$a b$	$a+b$	$\overline{a+b}$	$\bar{a}$	$\bar{b}$	$\bar{a} \cdot \bar{b}$
0 0	0	1	1	1	1
0 1	1	0	1	0	0
1 0	1	0	0	1	0
1 1	1	0	0	0	0

b)  $\overline{a \cdot b} = \bar{a} + \bar{b}$

$a b$	$a \cdot b$	$\overline{a \cdot b}$	$\bar{a}$	$\bar{b}$	$\bar{a} + \bar{b}$
0 0	0	1	1	1	1
0 1	0	1	1	0	1
1 0	0	1	0	1	1
1 1	1	0	0	0	0

Obsérvese, en los dos casos, las columnas sombreadas que justifican la igualdad de las ecuaciones y la dualidad existente entre las dos tablas.

*Tabla 3.3. Comprobación de la ley distributiva del producto lógico  $\cdot$  sobre la suma lógica  $+$*

$a b c$	$b + c$	$a \cdot (b + c)$	$a \cdot b$	$a \cdot c$	$(a \cdot b) + (a \cdot c)$
0 0 0	0	0	0	0	0
0 0 1	1	0	0	0	0
0 1 0	1	0	0	0	0
0 1 1	1	0	0	0	0
1 0 0	0	0	0	0	0
1 0 1	1	1	0	1	1
1 1 0	1	1	1	0	1
1 1 1	1	1	1	1	1



Tabla 3.4. Comprobación de que el álgebra bivalente cumple el postulado quinto

$a$	$\bar{a}$	$a + \bar{a}$	$a \cdot \bar{a}$
0	1	1	0
1	0	1	0

### PROBLEMAS PROPUESTOS

- 3-1) Comprobar, mediante tablas de verdad (de tres variables), que en un álgebra de Boole bivalente las operaciones  $+$  y  $\cdot$  son asociativas.
- 3-2) Aplicar el teorema de Shannon a la expresión:  $\overline{a+b \cdot c}$  y comprobar la igualdad del resultado mediante tabla de verdad.
- 3-3) Aplicar el teorema de expansión, respecto de la variable  $a$ , en la expresión:  $a \cdot b \cdot c + a \cdot \bar{b} \cdot c + \bar{a} \cdot b \cdot c$ , en sus dos partes duales.

### 3.3.1 Variables y funciones lógicas

Se define una **variable** como un símbolo, por ejemplo  $a$ , que representa a cualquiera de los elementos de un conjunto  $B$  sobre el que se ha definido un álgebra de Boole. Así, en el álgebra de conmutación la variable  $a$  puede tomar los valores 0 y 1, de ahí que se le designe como **variable binaria**.

Se define una **función booleana** como una correspondencia entre  $B^n$  y  $B$ , de tal forma que a cada  $n$ -upla de  $B^n$  se le hace corresponder con un elemento de  $B$ . Matemáticamente se expresa como:

$$\begin{aligned} f : B^n &\longrightarrow B \\ (a_1, a_2, \dots, a_n) &\longrightarrow a \end{aligned} \quad [3.17]$$

donde,

$$\begin{aligned} (a_1, a_2, \dots, a_n) &\in B^n \text{ y} \\ a &\in B \end{aligned}$$

Tanto a las variables como a las funciones booleanas se las conoce como variables o funciones lógicas, ya que, el álgebra de Boole es una teoría matemática usada para formalizar el pensamiento y de la que se deriva la lógica simbólica.

Una **función de conmutación** o **función lógica**  $f$  es una función booleana definida en  $B^n$ , cuya imagen pertenece al conjunto  $B = \{0, 1\}$ , siendo su valor igual al de una expresión algebraica de variables lógicas unidas mediante las operaciones de suma lógica  $+$ , producto lógico  $\cdot$  y el operador complemento.

Las **funciones lógicas se representan** como:

$$f = f(a_n, \dots, a_2, a_1) = f(\dots, c, b, a) \quad [3.18]$$

donde, el valor lógico de  $f$  depende de las variables binarias:  $\dots, c, b, a$ .

Entre las variables, el símbolo  $\cdot$ , correspondiente a la operación producto lógico, puede ser omitido.

### Ejemplo:

Algunas expresiones de funciones lógicas son las siguientes:

$$\begin{aligned} f_1 &= f_1(b, a) = b a + b \bar{a} \\ f_2 &= f_2(c, b, a) = c b + a \\ f_3 &= f_3(c, b, a) = b a + c b \bar{a} + \bar{b} a + a \\ f_4 &= f_4(c, b, a) = (b + a)(c + b + \bar{a})(\bar{b} + a) \\ f_5 &= f_5(e, d, c, b, a) = \bar{e} b a + \bar{d} c b \bar{a} + \bar{b} \end{aligned}$$

El **valor de una función** se determina sustituyendo las variables por sus valores en la expresión algebraica y aplicando las reglas definidas para las operaciones  $+$  y  $\cdot$ .

### Ejemplo:



Determinar el valor de la función anterior  $f_3$ , para  $a = 1$ ,  $b = 0$  y  $c = 1$  y comprobar el resultado mediante el programa de simulación *Electronics Workbench*.

La función  $f_3$  es la siguiente:

$$f_3 = f_3(c, b, a) = b a + c b \bar{a} + \bar{b} a + a$$

Sustituyendo en la expresión algebraica  $f_3$  las variables por sus valores ( $a = 1$ ,  $b = 0$  y  $c = 1$ ), se obtiene el resultado de la función:

$$f_3 = f_3(1, 0, 1) = 0 \cdot 1 + 1 \cdot 0 \cdot \bar{1} + \bar{0} \cdot 1 + 1 = 0 + 0 + 1 + 1 = 1$$

Utilizando el convertidor lógico del programa de simulación *Electronics Workbench*, se comprueba el resultado obtenido anteriormente de forma teórica. Para ello se introduce la expresión algebraica y se selecciona la conversión de expresión algebraica a circuito lógico, cuyo icono es:



Obtenido el circuito lógico se aplican los niveles lógicos correspondientes a sus entradas ( $a = 1$ ,  $b = 0$  y  $c = 1$ ) y se conecta la salida a un piloto (sonda) que indica el valor que toma la función.

Posteriormente se activa la simulación, con lo que se obtiene el resultado que se muestra en la Figura 3.6, en el que la salida del circuito toma el valor lógico alto al estar el piloto encendido.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap03\Ewb\03W0\_16.ewb**

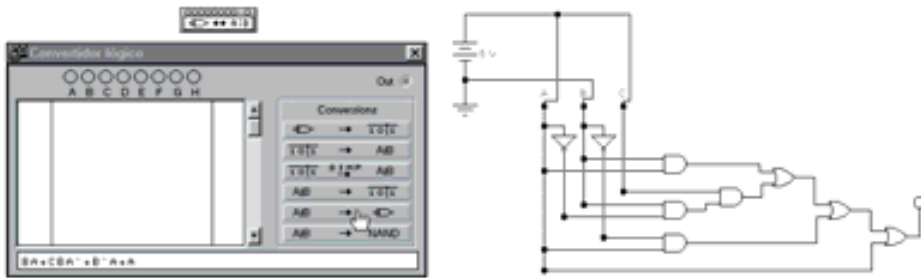


Figura 3.6. Comprobación del valor de una función mediante el convertidor lógico

### 3.3.2 Representación de las funciones lógicas mediante tablas de verdad

Otra forma de representar una función lógica es mediante una tabla, llamada **tabla de verdad**, que indique el valor que toma la función para cada una de las combinaciones de los valores de las variables de entrada.

La **construcción de la tabla de verdad** de una función se realiza representando en la columna de la izquierda, de la tabla, todas las posibles combinaciones de las variables de entrada y en la columna de la derecha los valores asignados a la función de salida  $f$  para cada combinación de las variables de entrada.

#### Ejemplo:

Representación de la anterior función  $f_3$ , mediante su tabla de verdad.

$$f_3 = f_3(c, b, a) = b a + c b \bar{a} + \bar{b} a + a$$

Calculando el valor que toma la función para cada una de las posibles combinaciones de valores de las variables de entrada y representándolo en una tabla, se obtiene su tabla de verdad.

El número de posibles combinaciones de valores de las variables de entrada es  $2^n$ , siendo  $n$  el número de variables de entrada. En este ejemplo, en el que se tienen tres variables, habrá ocho posibles combinaciones de las variables de entrada, tal como puede verse en la Tabla 3.5.

Tabla 3.5. Representación de la función  $f_3$  mediante su tabla de verdad

$c b a$	$f_3$
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	1



Utilizando el convertidor lógico del programa de simulación *Electronics Workbench*, se obtiene y comprueba la tabla de verdad de la función  $f_3$  del ejemplo anterior.

Para ello se introduce la función  $BA+CBA'+B'A+A$  en el convertidor lógico del programa y se activa la opción de conversión de expresión algebraica a tabla de verdad, con lo que se obtiene el resultado que se muestra en la parte inferior de la Figura 3.7.

Se hace la **observación**, para evitar confusiones, que en este programa de simulación las variables de entrada se representan en mayúscula  $A, B, C, \dots$  y están dispuestas en orden inverso en la asignación de pesos con respecto al criterio prefijado en este texto, en el que la variable  $a$  (en minúscula) es la de menor peso y según sea de mayor orden alfabético tendrá mayor peso.

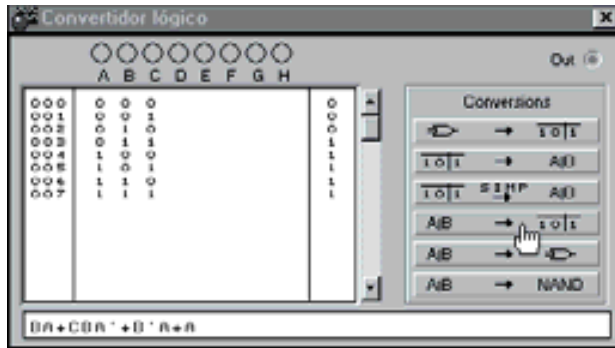


Figura 3.7. Obtención de la tabla de verdad de una función mediante el convertidor lógico

Una misma función lógica puede ser representada por expresiones algebraicas diferentes. Por ejemplo, las funciones:

$$f_2 = f_2(c, b, a) = cb + a$$

$$f_3 = f_3(c, b, a) = ba + cb\bar{a} + \bar{b}a + a$$

tienen la misma tabla de verdad que la representada anteriormente en la Tabla 3.5, dejando al lector, como ejercicio, su comprobación.

Se dice que dos funciones lógicas, como por ejemplo  $f_2$  y  $f_3$ , son **funciones equivalentes**,  $f_2 = f_3$ , si ambas tienen la misma tabla de verdad y por lo tanto describen la misma función de conmutación.



Utilizando el convertidor lógico del programa de simulación *Electronics Workbench*, se comprueba la equivalencia de las funciones lógicas  $f_2$  y  $f_3$ .

Para ello se introduce la función  $CB+A$  en el convertidor lógico y activando la opción de conversión de expresión algebraica a tabla de verdad se obtiene su tabla de verdad.

Se procede igualmente con la función  $BA+CBA'+B'A+A$  y se observa en la Figura 3.8 que ambas funciones son equivalentes por tener la misma tabla de verdad.

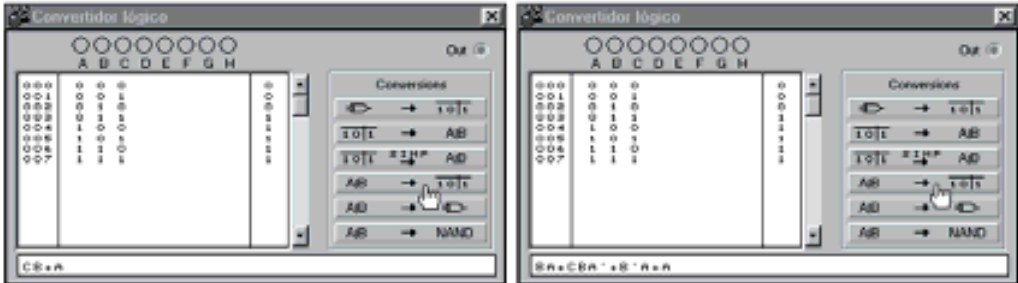


Figura 3.8. Obtención de la tabla de verdad de dos funciones equivalentes

### 3.3.3 Representación de las funciones lógicas en su forma canónica

Entre las múltiples expresiones algebraicas con las que se puede representar una función lógica, destacan dos tipos según la expresión esté formada por:

- sumas de productos, como por ejemplo:

$$f_3 = f_3(c, b, a) = b a + c b \bar{a} + \bar{b} a + a$$

- productos de sumas, como por ejemplo:

$$f_4 = f_4(c, b, a) = (b + a)(c + b + \bar{a})(\bar{b} + a)$$

Se define como **término canónico** de una función lógica a todo producto o suma en el que aparecen todas las variables en su forma directa  $a$  o complementada  $\bar{a}$ . Por ejemplo, en una función de tres variables, son términos canónicos, entre otros:  $c b \bar{a}$  y  $\bar{c} + b + \bar{a}$ .

A los términos producto se les llama **productos canónicos o minitérminos** (*miniterms*). Esta denominación se debe al hecho de que, este término, toma el valor 1 para una sola combinación de las variables de entrada, de ahí el prefijo mini. A los términos suma se les llama **sumas canónicas o maxitérminos** (*Maxterms*), denominándose así por el hecho de que, este término, toma el valor 1 tantas veces como lo hagan los sumandos que lo forman, de ahí el prefijo maxi.

Una función formada, exclusivamente, por términos de sumas canónicas o bien de productos canónicos recibe el nombre de **función canónica**. Si esta función tiene  $n$  variables, cada uno de sus productos o sumas canónicas tendrá  $n$  variables. Como cada variable se puede representar en su forma directa o complementada, el número de productos canónicos posibles será  $2^n$ , al igual que el de sumas canónicas.

**Ejemplo:**

Una función de tres variables tiene un máximo de  $2^3 = 8$  minterminos e igual número de maxiterminos.

Un método sencillo para la **determinación de los términos canónicos**, con  $n$  variables, consiste en representar en una tabla (como se muestra en la Tabla 3.6) las combinaciones que pueden formarse con las  $n$  variables. Para ello, en una columna, debajo de las  $n$  variables (...  $c b a$ ), se listan los números binarios comprendidos entre 0 y  $2^n - 1$ . Cada mintermino o *minterm* se obtiene multiplicando las  $n$  variables en su forma directa si toman el valor 1 y complementada si tienen el valor 0.

Asimismo, aplicando el principio de dualidad, cada maxitermino o *maxterm* se obtiene sumando las  $n$  variables en su forma directa si toman el valor 0 y complementada si tienen el valor 1.

Los *minterms* se representan por  $m_i$  y los *maxterms* por  $M_i$ , siendo el subíndice  $i$  igual al valor decimal del número binario que corresponde al término canónico.

Tabla 3.6. Tabla de minterms y maxterms para una función de tres variables

Decimal	$c b a$	<i>Minterms</i>	<i>Maxterms</i>
0	0 0 0	$\bar{c} \bar{b} \bar{a}$ $m_0$	$c + b + a$ $M_7$
1	0 0 1	$\bar{c} \bar{b} a$ $m_1$	$c + b + \bar{a}$ $M_6$
2	0 1 0	$\bar{c} b \bar{a}$ $m_2$	$c + \bar{b} + a$ $M_5$
3	0 1 1	$\bar{c} b a$ $m_3$	$c + \bar{b} + \bar{a}$ $M_4$
4	1 0 0	$c \bar{b} \bar{a}$ $m_4$	$\bar{c} + b + a$ $M_3$
5	1 0 1	$c \bar{b} a$ $m_5$	$\bar{c} + b + \bar{a}$ $M_2$
6	1 1 0	$c b \bar{a}$ $m_6$	$\bar{c} + \bar{b} + a$ $M_1$
7	1 1 1	$c b a$ $m_7$	$\bar{c} + \bar{b} + \bar{a}$ $M_0$

### 3.3.4 Obtención de la función canónica a partir de la tabla de verdad. Teorema de expansión

**Teorema de expansión o desarrollo de Shannon (primera fórmula):** Cualquier función de  $n$  variables puede expresarse, mediante un desarrollo único, como suma de *minterms*.

Dada una función de una variable  $f(a)$ , se verifica que:

$$f(a) = \bar{a} \cdot f(0) + a \cdot f(1) \quad [3.19]$$

Demostración:

Si  $a = 0$  entonces  $\bar{a} = 1$  y,

$$f(0) = 1 \cdot f(0) + 0 \cdot f(1) = f(0)$$

Si  $a = 1$  entonces  $\bar{a} = 0$  y,

$$f(1) = 0 \cdot f(0) + 1 \cdot f(1) = f(1)$$

cumpléndose la expresión [3.19].

La igualdad anterior puede hacerse extensiva a funciones con  $n$  variables,

$$f(a_n, \dots, a_i, \dots, a_1) = \bar{a}_i \cdot f(a_n, \dots, 0, \dots, a_1) + a_i \cdot f(a_n, \dots, 1, \dots, a_1) \quad [3.20]$$

Demostración:

Si  $a_i = 0$  entonces  $\bar{a}_i = 1$  y,

$$\begin{aligned} f(a_n, \dots, 0, \dots, a_1) &= 1 \cdot f(a_n, \dots, 0, \dots, a_1) + 0 \cdot f(a_n, \dots, 1, \dots, a_1) = \\ &= f(a_n, \dots, 0, \dots, a_1) \end{aligned}$$

Si  $a_i = 1$  entonces  $\bar{a}_i = 0$  y,

$$\begin{aligned} f(a_n, \dots, 1, \dots, a_1) &= 0 \cdot f(a_n, \dots, 0, \dots, a_1) + 1 \cdot f(a_n, \dots, 1, \dots, a_1) = \\ &= f(a_n, \dots, 1, \dots, a_1) \end{aligned}$$

cumpléndose la expresión [3.20].

Expandiendo dos variables (por ejemplo las dos primeras), se obtiene la siguiente expresión [3.21].

$$\begin{aligned} f(a_n, \dots, a_1) &= \bar{a}_2 \bar{a}_1 \cdot f(a_n, \dots, a_3, 0, 0) + \bar{a}_2 a_1 \cdot f(a_n, \dots, a_3, 0, 1) + \\ &+ a_2 \bar{a}_1 \cdot f(a_n, \dots, a_3, 1, 0) + a_2 a_1 \cdot f(a_n, \dots, a_3, 1, 1) \end{aligned} \quad [3.21]$$

El proceso puede repetirse para las  $n$  variables de la función, de forma inductiva, hasta obtener la función canónica.

$$f(a_n, \dots, a_1) = \bar{a}_n \dots \bar{a}_1 \cdot f(0, \dots, 0) + \dots + a_n \dots a_1 \cdot f(1, \dots, 1) \quad [3.22]$$

Cada sumando de la expresión [3.22] está constituido por el producto de las  $n$  variables o *minterm*,  $m_i$ , multiplicado por una subfunción  $f_i$ , que es igual al valor de la



función en dicho *minterm*. Otra forma de representar dicha función es la que se muestra en la expresión [3.23].

$$f(a_n, \dots, a_1) = m_0 \cdot f_0 + \dots + m_{2^n-1} \cdot f_{2^n-1} = \sum_{i=0}^{2^n-1} m_i \cdot f_i \quad [3.23]$$

Al expandir las  $n$  variables de la función aparecen  $2^n$  sumandos en los que las subfunciones  $f_0 = f(0, \dots, 0, 0)$ ,  $f_1 = f(0, \dots, 0, 1)$  hasta  $f_{2^n-1} = f(1, \dots, 1, 1)$  toman un valor binario constante (0 o 1). Los sumandos en los que la subfunción  $f_i$  sea cero serán nulos y sólo los sumandos que tengan la subfunción igual a uno formarán parte de la expresión.

La **expresión canónica de la función a partir de su tabla de verdad** se obtiene sumando los *minterms* en los que la función vale uno. En la expresión canónica habrá tantos *minterms* como unos tenga la tabla de verdad de la función.

### Ejemplo:

Aplicación del teorema de expansión de Shannon (primera fórmula o por *minterms*) a una función de tres variables.

$$\begin{aligned} f(c, b, a) &= \bar{a} \cdot f(c, b, 0) + a \cdot f(c, b, 1) = \\ &= \bar{b} [\bar{a} \cdot f(c, 0, 0) + a \cdot f(c, 0, 1)] + b [\bar{a} \cdot f(c, 1, 0) + a \cdot f(c, 1, 1)] = \\ &= \bar{c} \{ \bar{b} [\bar{a} \cdot f(0, 0, 0) + a \cdot f(0, 0, 1)] + b [\bar{a} \cdot f(0, 1, 0) + a \cdot f(0, 1, 1)] \} + \\ &+ c \{ \bar{b} [\bar{a} \cdot f(1, 0, 0) + a \cdot f(1, 0, 1)] + b [\bar{a} \cdot f(1, 1, 0) + a \cdot f(1, 1, 1)] \} \end{aligned} \quad [3.24]$$

desarrollando los paréntesis y operando se obtiene la expresión [3.25].

$$\begin{aligned} f(c, b, a) &= \\ &= \bar{c} \bar{b} \bar{a} f(0, 0, 0) + \bar{c} \bar{b} a f(0, 0, 1) + \bar{c} b \bar{a} f(0, 1, 0) + \bar{c} b a f(0, 1, 1) + \\ &+ c \bar{b} \bar{a} f(1, 0, 0) + c \bar{b} a f(1, 0, 1) + c b \bar{a} f(1, 1, 0) + c b a f(1, 1, 1) \end{aligned} \quad [3.25]$$

Dicha expresión [3.25] indica que una función lógica puede expresarse como suma de los productos formados por los *minterms* y el valor de la función en cada *minterm*, tal como puede se muestra en la expresión [3.26].

$$\begin{aligned} f(c, b, a) &= \\ &= m_0 f(0, 0, 0) + m_1 f(0, 0, 1) + m_2 f(0, 1, 0) + m_3 f(0, 1, 1) + \\ &+ m_4 f(1, 0, 0) + m_5 f(1, 0, 1) + m_6 f(1, 1, 0) + m_7 f(1, 1, 1) = \\ &= \sum_{i=0}^7 m_i f_i \end{aligned} \quad [3.26]$$

**PROBLEMA RESUELTO 3-2**

Obtener la función canónica a partir de la tabla de verdad siguiente:

$c b a$	$f$
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	1

**Solución:**

La función canónica se obtiene sumando los *minterms* en los que la función vale uno. Obsérvese que la expresión canónica tiene tantos *minterms* como unos tenga la columna del valor de la función en la tabla de verdad.

$$f = f(c, b, a) = m_1 + m_3 + m_4 + m_6 + m_7 = \bar{c}\bar{b}a + \bar{c}ba + c\bar{b}\bar{a} + c\bar{b}a + cba$$



Utilizando el convertidor lógico del programa de simulación *Electronics Workbench*, se obtiene la función, en forma canónica en *minterms*, a partir de su tabla de verdad.

Se introduce en el convertidor lógico la tabla de verdad del Problema resuelto 3-2. Para ello, se activan las columnas de las variables de entrada  $A$ ,  $B$ ,  $C$ , haciendo clic sobre los círculos grises; y situados encima de ellas se rellenan los códigos de las columnas de la tabla de verdad correspondientes a las variables seleccionadas. Posteriormente, se rellena la columna de salida situada a la derecha (Figura 3.9).

Para interpretar el resultado, es importante tener en cuenta la observación realizada anteriormente en lo que al convenio de las variables se refiere.

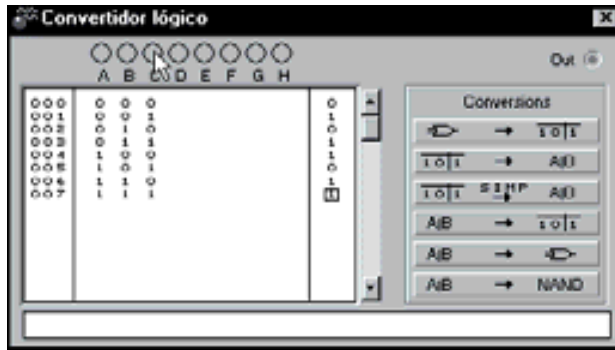


Figura 3.9. Introducción de la tabla de verdad de la función propuesta en el convertidor lógico del simulador

Una vez completada la tabla de verdad se puede obtener su función canónica, en forma de *minterms*. Para ello se debe hacer clic en la siguiente opción del convertidor lógico:



La expresión así obtenida coincide con el resultado del Problema resuelto 3-2, como puede verse en la parte inferior de la Figura 3.10, teniendo en cuenta que las variables negadas se representan con una comilla simple a su derecha.

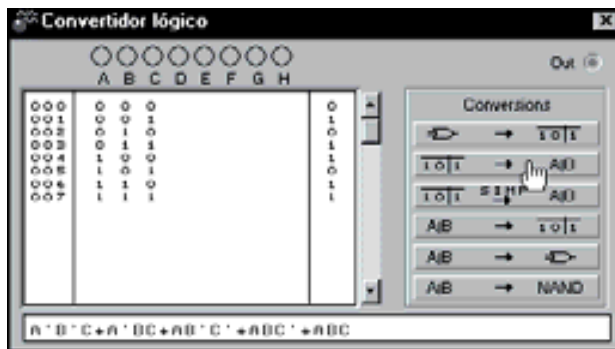


Figura 3.10. Función canónica en forma de minterms correspondiente a la tabla de verdad introducida en el convertidor lógico

Se puede tener una **representación más compacta de las funciones canónicas** expresadas en *minterms*. La función anterior se puede expresar como:

$$f = f(c, b, a) = \sum_3 (1, 3, 4, 6, 7)$$

Donde el símbolo sumatorio  $\Sigma$  representa la suma de los *minterms* cuyos números se encuentren dentro del paréntesis. Debajo de  $\Sigma$  se indica el número  $n$  de variables que tiene la función.

Teniendo en cuenta el principio de dualidad, se puede aplicar el teorema de Shannon para desarrollar una expansión de la función por *maxterms*.

**Teorema de expansión o desarrollo de Shannon (segunda fórmula):** Cualquier función de  $n$  variables puede expresarse, mediante un desarrollo único, como producto de *maxterms*.

Dada una función de una variable  $f(a)$  se verifica que:

$$f(a) = [\bar{a} + f(1)][a + f(0)] \quad [3.27]$$

Demostración:

Si  $a = 0$ , entonces  $\bar{a} = 1$  y,

$$f(0) = [1 + f(1)][0 + f(0)] = f(0)$$

Si  $a = 1$ , entonces  $\bar{a} = 0$  y,

$$f(1) = [0 + f(1)][1 + f(0)] = f(1)$$

cumpléndose la expresión [3.27].

La igualdad anterior puede hacerse extensiva a funciones con  $n$  variables,

$$f(a_n, \dots, a_i, \dots, a_1) = [\bar{a}_i + f(a_n, \dots, 1, \dots, a_1)][a_i + f(a_n, \dots, 0, \dots, a_1)] \quad [3.28]$$

Demostración:

Si  $a_i = 0$ , entonces  $\bar{a}_i = 1$  y,

$$\begin{aligned} f(a_n, \dots, 0, \dots, a_1) &= [1 + f(a_n, \dots, 1, \dots, a_1)][0 + f(a_n, \dots, 0, \dots, a_1)] = \\ &= f(a_n, \dots, 0, \dots, a_1) \end{aligned}$$

Si  $a_i = 1$ , entonces  $\bar{a}_i = 0$  y,

$$\begin{aligned} f(a_n, \dots, 1, \dots, a_1) &= [0 + f(a_n, \dots, 1, \dots, a_1)][1 + f(a_n, \dots, 0, \dots, a_1)] = \\ &= f(a_n, \dots, 1, \dots, a_1) \end{aligned}$$

cumpléndose la expresión [3.28].

Expandiendo dos variables cualesquiera (por ejemplo las dos primeras), se obtiene la expresión [3.29].

$$f(a_n, \dots, a_1) = [\bar{a}_2 + \bar{a}_1 + f(a_n, \dots, a_3, 1, 1)] [\bar{a}_2 + a_1 + f(a_n, \dots, a_3, 1, 0)] \\ [a_2 + \bar{a}_1 + f(a_n, \dots, a_3, 0, 1)] [a_2 + a_1 + f(a_n, \dots, a_3, 0, 0)] \quad [3.29]$$

El proceso puede repetirse para las  $n$  variables de la función, de forma inductiva, hasta obtener la función canónica.

$$f(a_n, \dots, a_1) = [\bar{a}_n + \dots + \bar{a}_1 + f(1, \dots, 1)] \dots [a_n + \dots + a_1 + f(0, \dots, 0)] \quad [3.30]$$

Cada producto de la expresión [3.30] está constituido por la suma de las  $n$  variables o *maxterm*  $M_i$ , más el valor de la función en dicho *maxterm*. Otra forma de representar dicha función es la que se muestra en la expresión [3.31].

$$f(a_n, \dots, a_1) = [M_{2^n-1} + f_0] \dots [M_0 + f_{2^n-1}] = \prod_{i=0}^{2^n-1} (M_{2^n-1-i} + f) \quad [3.31]$$

Al expandir las  $n$  variables de la función aparecen  $2^n$  productos en los que las subfunciones  $f_0 = f(0, \dots, 0, 0)$ ,  $f_1 = f(0, \dots, 0, 1)$  hasta  $f_{2^n-1} = f(1, \dots, 1, 1)$  toman un valor binario constante (0 o 1). Los productos en los que la subfunción  $f_i$  sea uno serán identidad y sólo los sumandos que tengan la subfunción igual a cero formarán parte de la expresión.

La **expresión canónica de la función a partir de su tabla de verdad** se obtiene multiplicando los *maxterms* en los que la función vale cero. En la expresión canónica habrá tantos *maxterms* como ceros tenga la tabla de verdad de la función.

### Ejemplo:

Aplicación del teorema de expansión de Shannon (segunda fórmula o por *maxterms*) a una función de tres variables.

$$f(c, b, a) = [\bar{a} + f(c, b, 1)] [a + f(c, b, 0)] = \\ = \{ \bar{b} + [\bar{a} + f(c, 1, 1)] [a + f(c, 1, 0)] \} \{ b + [\bar{a} + f(c, 0, 1)] [a + f(c, 0, 0)] \} = \\ = \langle \bar{c} + \{ \bar{b} + [\bar{a} + f(1, 1, 1)] [a + f(1, 1, 0)] \} \rangle \{ b + [\bar{a} + f(1, 0, 1)] [a + f(1, 0, 0)] \} \rangle \\ \langle c + \{ \bar{b} + [\bar{a} + f(0, 1, 1)] [a + f(0, 1, 0)] \} \rangle \{ b + [\bar{a} + f(0, 0, 1)] [a + f(0, 0, 0)] \} \rangle$$

desarrollando los paréntesis y operando se obtiene que:

$$\begin{aligned}
 f(c, b, a) &= \\
 &= [\bar{c} + \bar{b} + \bar{a} + f(1,1,1)][\bar{c} + \bar{b} + a + f(1,1,0)][\bar{c} + b + \bar{a} + f(1,0,1)] \\
 &[\bar{c} + b + a + f(1,0,0)][c + \bar{b} + \bar{a} + f(0,1,1)][c + \bar{b} + a + f(0,1,0)] \\
 &[c + b + \bar{a} + f(0,0,1)][c + b + a + f(0,0,0)]
 \end{aligned} \tag{3.32}$$

La expresión [3.32] indica que una función lógica puede expresarse como producto de los sumandos formados por los *maxterms* más el valor de la función en cada *maxterm*.

En la expresión [3.33] se muestra dicho resultado, en la que se ha invertido el orden de los términos.

$$\begin{aligned}
 f(c, b, a) &= \\
 &= [M_7 + f(0,0,0)][M_6 + f(0,0,1)][M_5 + f(0,1,0)][M_4 + f(0,1,1)] \\
 &[M_3 + f(1,0,0)][M_2 + f(1,0,1)][M_1 + f(1,1,0)][M_0 + f(1,1,1)] = \\
 &= \prod_{i=0}^7 (M_{7-i} + f_i)
 \end{aligned} \tag{3.33}$$

### PROBLEMA RESUELTO 3-3

Obtener la función canónica a partir de la tabla de verdad siguiente:

<i>c b a</i>	<i>f</i>
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	1

#### Solución:

La función canónica se obtiene multiplicando los *maxterms* en los que la función vale cero. Obsérvese que la expresión canónica tiene tantos *maxterms* como ceros tenga la columna del valor de la función de la tabla de verdad.

$$f = f(c, b, a) = M_2 \cdot M_5 \cdot M_7 = (\bar{c} + b + \bar{a})(c + \bar{b} + a)(c + b + a)$$

Se puede tener una **representación más compacta de las funciones canónicas** expresadas en *maxterms*. La función anterior se puede expresar como:

$$f = f(c, b, a) = M_2 \cdot M_5 \cdot M_7 = \prod_3(2, 5, 7)$$

Donde el símbolo  $\Pi$  representa al producto de los *maxterms* cuyos números se encuentren dentro del paréntesis. Debajo de  $\Pi$  se indica el número  $n$  de variables que tiene la función.

### Resumiendo:

- Para obtener una **expresión canónica en suma de productos** (*minterms*) se usarán las combinaciones de variables binarias en las que la **función vale uno**.
- En el caso de querer obtener una **expresión canónica en producto de sumas** (*maxterms*) se usarán las combinaciones de variables binarias en las que la **función vale cero**.

## 3.3.5 Conversión entre expresiones canónicas en minterms y maxterms

Para  $n$  variables, utilizando las leyes de De Morgan, se deduce la relación existente entre el complemento de un *minterm* y su *maxterm* equivalente, que se muestra en la expresión [3.34].

$$\bar{m}_i = M_{2^n - 1 - i} \quad [3.34]$$

o viceversa, la relación existente entre el complemento de un *maxterm* y su *minterm* equivalente, que se muestra en la expresión [3.35].

$$\bar{M}_i = m_{2^n - 1 - i} \quad [3.35]$$

### Ejemplo:

Para  $n = 3$ , se cumple:

$$\begin{aligned} \bar{m}_2 &= M_{2^3 - 1 - 2} = M_5 \\ \bar{c} \bar{b} \bar{a} &= c + \bar{b} + a \end{aligned}$$

o viceversa,

$$\overline{M}_5 = m_{2^3-1-5} = m_2$$

$$\overline{c + \overline{b} + a} = \overline{c} \overline{b} \overline{a}$$

El proceso se puede generalizar al conjunto de términos de una función, considerando que,

$$f = \sum_{i=0}^{2^n-1} m_i$$

donde,  $m_i$  son los términos en los que la función vale uno. La función  $\overline{f}$  está formada por los *minterms* que no pertenecen a  $f$  y por lo tanto la hacen cero. Utilizando las leyes de involución (doble negación) y De Morgan en  $f$  se obtiene la expresión canónica en *maxterms*.

### Ejemplo:

La función,

$$f(c, b, a) = \sum_3 (1, 3, 4, 6, 7)$$

vale uno en cada *minterm*.

La función complemento de  $f$ ,

$$\overline{f(c, b, a)} = \sum_3 (0, 2, 5)$$

está formada por los *minterms* que no pertenecen a  $f$  y por lo tanto la hacen cero.

Utilizando las leyes involución y De Morgan en  $f$  se obtiene la expresión [3.36] en *maxterms*.

$$\begin{aligned} f(c, b, a) &= \overline{\overline{f(c, b, a)}} = \overline{\sum_3 (0, 2, 5)} = \overline{m_0 + m_2 + m_5} = \\ &= \overline{m_0} \cdot \overline{m_2} \cdot \overline{m_5} = M_7 \cdot M_5 \cdot M_2 = \prod_3 (2, 5, 7) \end{aligned} \quad [3.36]$$

### 3.3.6 Conversión de expresiones normalizadas a canónicas

Las **expresiones normalizadas** son aquellas en las que no todos sus términos son canónicos y están únicamente formadas por suma de productos o por producto de sumas.



**Ejemplo:**

Son funciones normalizadas,

$$f_1(c, b, a) = cb + \bar{c}b\bar{a}$$

$$f_2(c, b, a) = (c + b)(\bar{c} + b + \bar{a})$$

sin embargo no es normalizada,

$$f_3(c, b, a) = c(b + \bar{b}a) + \bar{b}$$

pudiéndose normalizar si se opera sobre ella (desarrollando sus paréntesis),

$$f_3(c, b, a) = c(b + \bar{b}a) + \bar{b} = cb + c\bar{b}a + \bar{b}$$

Para **convertir una expresión normalizada a canónica:**

- En el caso de **suma de productos**, se multiplica cada término producto no canónico por la variable que falta más ella misma negada.
- En el caso de **producto de sumas**, se suma en cada factor no canónico la variable que falta por ella misma negada.

En ambos casos, el proceso se repite por cada variable que falte en cada término.

**Ejemplo:**

La conversión de la función normalizada  $f_3$ , del ejemplo anterior, a expresión canónica se realiza del siguiente modo:

$$f_3(c, b, a) = cb + c\bar{b}a + \bar{b} = cb(a + \bar{a}) + c\bar{b}a + \bar{b}(c + \bar{c})(a + \bar{a}) =$$

$$= cba + cb\bar{a} + \bar{c}\bar{b}a + c\bar{b}\bar{a} + \bar{c}\bar{b}\bar{a} + \bar{c}\bar{b}a + \bar{c}\bar{b}\bar{a} =$$

$$= cba + cb\bar{a} + \bar{c}\bar{b}a + \bar{c}\bar{b}\bar{a} + \bar{c}\bar{b}a + \bar{c}\bar{b}\bar{a} = \sum_3(0, 1, 4, 5, 6, 7)$$

Se puede afirmar que una función normalizada es una simplificación de la expresión canónica, pues utiliza menor número de variables, términos y operandos.



Utilizando el convertidor lógico del programa de simulación *Electronics Workbench*, se realiza la conversión de la función normalizada, del ejemplo anterior, a expresión canónica.

Para ello se introduce la función normalizada  $CB + CB'A + B'$  en el convertidor lógico y se obtiene su tabla de verdad activando la opción de conversión de expresión algebraica a tabla de verdad.



Dichas funciones se clasifican, agrupándolas según su complejidad en:

- Funciones **constantes**.

$f_0 = 0$       **función nula**. Siempre vale cero.

$f_{15} = 1$       **función unidad**. Siempre vale uno.

- Funciones **variables simples**.

$f_{10} = a$  y  $f_{12} = b$       **funciones de transferencia**. Transfiere a la salida una de las variables de entrada.

$f_3 = \bar{b}$  y  $f_5 = \bar{a}$       **funciones de complementación**. Transfiere a la salida una de las variables de entrada complementada.

- Funciones con la **operación producto**.

$f_8 = b a$       **función AND**.

$f_2 = \bar{b} a$       **función inhibición**. Se lee:  $a$  pero no  $b$ , y se simboliza  $a/b$ .

$f_4 = b \bar{a}$       **función inhibición**. Se lee:  $b$  pero no  $a$ , y se simboliza  $b/a$ .

$f_1 = \bar{b} \bar{a}$       **función NOR**. Siendo:  $f_1 = \bar{b} \bar{a} = \overline{b + a}$ .

- Funciones con la **operación suma**.

$f_{14} = b + a$       **función OR**.

$f_{11} = \bar{b} + a$       **función implicación**. Se lee: si  $b$  entonces  $a$ , y se simboliza como  $b \Rightarrow a$ .

$f_{13} = b + \bar{a}$       **función implicación**. Se lee: si  $a$  entonces  $b$ , ( $a \Rightarrow b$ ).

$f_7 = \bar{b} + \bar{a}$       **función NAND**. Siendo:  $f_7 = \bar{b} + \bar{a} = \overline{b a}$ .

- Funciones con la **operación producto y suma**.

$f_6 = b \bar{a} + \bar{b} a$       **función XOR**. Se lee:  $a$  distinta a  $b$ , y se simboliza como  $b \oplus a$ .

$f_9 = \bar{b} \bar{a} + b a$       **función XNOR**. Se lee:  $a$  igual a  $b$ , y se simboliza como  $b \odot a$ .

En la Tabla 3.8 se resumen las dieciséis funciones anteriores que se pueden formar con dos variables.

Tabla 3.8. Tabla resumen de las funciones que se pueden formar con dos variables

FUNCIÓN	NOMBRE	OPERADOR	OBSERVACIÓN
$f_0 = 0$	Nula		Constante binaria 0
$f_1 = \overline{b} \overline{a} = \overline{b+a}$	NOR	$\overline{b+a}$	No OR
$f_2 = \overline{b} a$	Inhibición	$a\overline{b}$	$a$ pero no $b$
$f_3 = \overline{b}$	Complemento	$\overline{b}$	No $b$
$f_4 = b \overline{a}$	Inhibición	$b\overline{a}$	$b$ pero no $a$
$f_5 = \overline{a}$	Complemento	$\overline{a}$	No $a$
$f_6 = \overline{b} a + b \overline{a}$	OR exclusiva	$b \oplus a$	$b$ distinta de $a$
$f_7 = \overline{b} + \overline{a} = \overline{b \cdot a}$	NAND	$\overline{b \cdot a}$	No AND
$f_8 = b a$	AND	$b \cdot a$	$b$ y $a$
$f_9 = \overline{b} \overline{a} + b a$	Equivalencia	$b \odot a$	$b$ igual a $a$
$f_{10} = a$	Transferencia		$a$
$f_{11} = \overline{b} + a$	Implicación	$b \Rightarrow a$	Si $b$ entonces $a$
$f_{12} = b$	Transferencia		$b$
$f_{13} = b + \overline{a}$	Implicación	$a \Rightarrow b$	Si $a$ entonces $b$
$f_{14} = b + a$	OR	$b + a$	$b$ o $a$
$f_{15} = 1$	Identidad		Constante binaria 1



Utilizando el convertidor lógico del programa de simulación *Electronics Workbench*, se obtienen las dieciséis funciones distintas que se pueden formar con dos variables.

El procedimiento que debe seguirse es el mismo en cada una de las dieciséis posibles funciones:

- Se introduce la tabla de verdad en el convertidor lógico.
- Se obtiene su expresión algebraica mediante la opción de conversión de tabla de verdad a expresión algebraica.
- Se simplifica la expresión algebraica mediante la opción SIMP del convertidor lógico.
- Se obtiene el circuito lógico mediante la opción de conversión de expresión algebraica a circuito lógico.

A continuación, como ejemplo de este tipo de simulación, se verifica la expresión algebraica y el circuito lógico de la función nula ( $f_0$ ) representada en

la Figura 3.12, de la función XOR ( $f_6$ ) representada en la Figura 3.13 y de la función OR ( $f_{14}$ ) representada en la Figura 3.14.

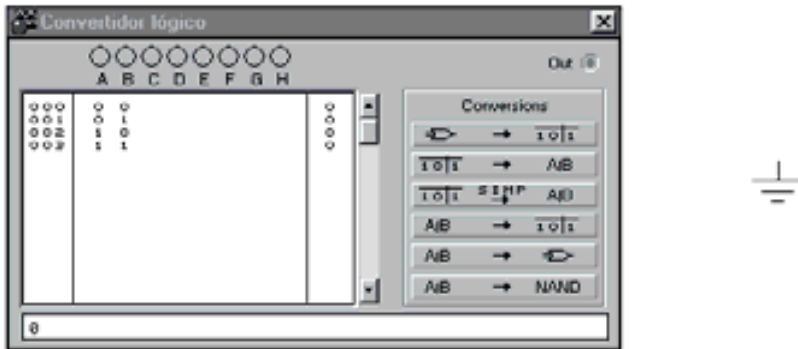


Figura 3.12. A partir de la tabla de verdad de la función nula ( $f_0$ ) se obtiene su expresión algebraica y su circuito lógico mediante el convertidor lógico

**Nota:** Cabe señalar, que cuando se trata de simular la función identidad  $f_{15} = 1$ , se produce un error en el convertidor lógico del programa de simulación, y aparece como resultado una masa que indica nivel bajo en vez de aparecer un nivel alto que sería el resultado correcto.

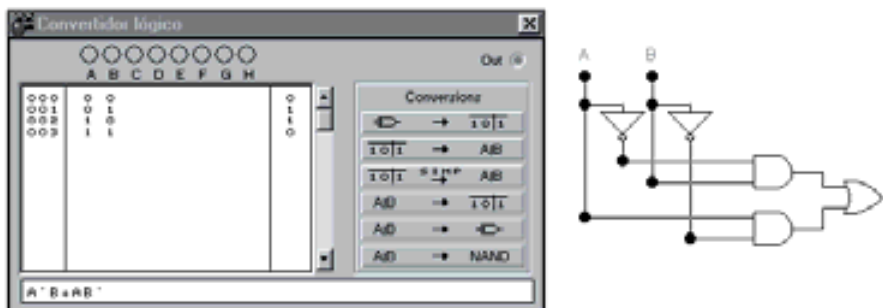


Figura 3.13. A partir de la tabla de verdad de la función XOR ( $f_6$ ) se obtiene su expresión algebraica y su circuito lógico mediante el convertidor lógico

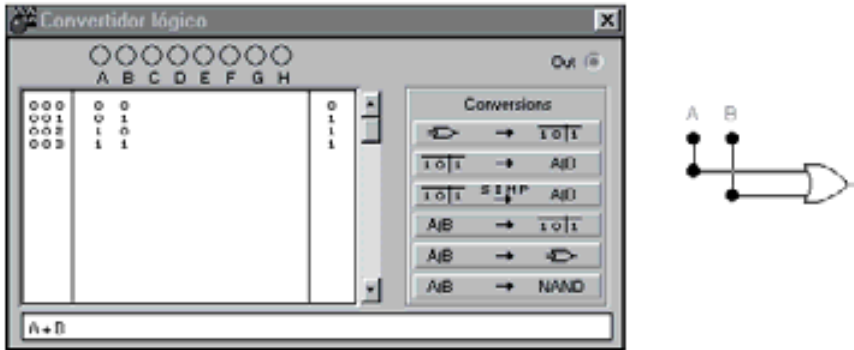


Figura 3.14. A partir de la tabla de verdad de la función OR ( $f_{14}$ ) se obtiene su expresión algebraica y su circuito lógico mediante el convertidor lógico

Un conjunto de operadores es funcionalmente completo, si cualquier función lógica se puede expresar mediante los operadores de este conjunto.

La función canónica puede expresar cualquier función lógica (en *minterms* o *maxterms*) y sólo utiliza operadores AND (operación producto  $\cdot$ ), OR (operación suma  $+$ ) y NOT (operación complemento  $\bar{\phantom{x}}$ ), por lo que el conjunto de operadores  $\{\cdot, +, \bar{\phantom{x}}\}$  es funcionalmente completo.

Utilizando las leyes de De Morgan se demuestra que el operador  $\cdot$ , puede ser sustituido por el conjunto de operadores  $\{+, \bar{\phantom{x}}\}$ , según se demuestra, a modo de ejemplo, en la siguiente expresión:

$$b \cdot a = \overline{\overline{b + a}}$$

Por consiguiente, el conjunto de operadores  $\{+, \bar{\phantom{x}}\}$  es funcionalmente completo.

Asimismo, mediante las leyes de De Morgan, el operador  $+$  puede ser sustituido por el conjunto de operadores  $\{\cdot, \bar{\phantom{x}}\}$ , según se demuestra en el siguiente ejemplo:

$$b + a = \overline{\overline{b \cdot a}}$$

Por consiguiente, el conjunto de operadores  $\{\cdot, \bar{\phantom{x}}\}$  es también funcionalmente completo.

Cabe indicar que los conjuntos  $\{+, \bar{\phantom{x}}\}$  y  $\{\cdot, \bar{\phantom{x}}\}$  se corresponden con los operadores NOR y NAND.

Se concluye que los operadores NOR y NAND son funcionalmente completos, y, si bien existen otros operadores funcionalmente completos, éstos son los más empleados.

### 3.3.8 Función incompletamente definida

Una función lógica está **completamente definida** si para cada una de las posibles combinaciones de sus variables, existe, y es único, el valor de la función. Por el contrario, se define una **función incompletamente definida o función incompleta** aquella que puede tomar indistintamente el valor 0 o 1 para una o más combinaciones de sus variables de entrada, también llamadas **términos indiferentes o indiferencias**.

Algunas razones que justifican que una función pueda ser incompleta, son:

- Combinaciones de variables que físicamente no se puedan producir.
- Combinaciones de variables para las cuales el valor de la función sea indiferente o no afecte al sistema digital.

La **expresión algebraica de las funciones incompletas** se efectúa añadiendo, a la expresión en términos canónicos, la lista de indiferencias precedida del símbolo  $X$ , que las identifica, como se muestra en la expresión [3.37].

$$f_1(c, b, a) = \sum_3(1, 2, 4, 7) + X(0, 5) \quad [3.37]$$

$$f_1(c, b, a) = \prod_3(1, 4) \cdot X(2, 7)$$

La representación de la **tabla de verdad de funciones incompletas** se realiza de la misma forma que para funciones completas, salvo en aquellos términos indiferentes que les asigna como valor de la función el símbolo  $X$ , indicando con ello que la función puede tomar indistintamente el valor 0 o 1.

En la Tabla 3.9 se representa mediante su tabla de verdad, la función incompleta definida en la expresión [3.37].

Tabla 3.9. Tabla de verdad de una función incompletamente definida

$c b a$	$f_3$
0 0 0	$X$
0 0 1	1
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	$X$
1 1 0	0
1 1 1	1

En muchos casos, poder asignar indistintamente el valor 0 o 1 a los términos indiferentes, favorece la obtención de una mayor simplificación, como se verá posteriormente en los métodos de simplificación.



Utilizando el convertidor lógico del programa de simulación *Electronics Workbench*, se puede introducir la tabla de verdad de una función incompletamente definida y obtener una de sus expresiones algebraicas o uno de sus posibles circuitos lógicos.

El procedimiento es el siguiente:

- a) Se introduce la tabla de verdad en el convertidor lógico, en el que al menos uno de los valores de la función será X.
- b) Se obtiene su expresión algebraica mediante la opción de conversión de tabla de verdad a expresión algebraica.
- c) Se simplifica la expresión algebraica mediante la opción SIMP del convertidor lógico.
- d) Se obtiene el circuito lógico mediante la opción de conversión de expresión algebraica a circuito lógico.

A continuación, con el objeto de que sirva como ejemplo de este tipo de simulación, se simula la función incompleta de la Tabla 3.9.

A partir de la tabla de verdad de una función incompletamente definida se obtiene su expresión algebraica y su circuito lógico mediante el convertidor lógico. En la Figura 3.15 se muestra la expresión algebraica (parte inferior del convertidor lógico) y el circuito lógico (parte derecha de la figura).

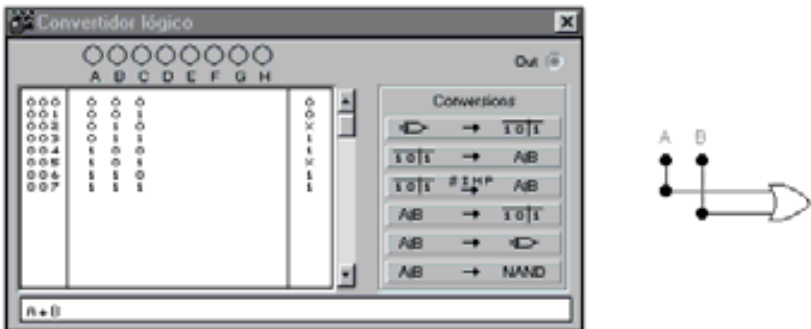


Figura 3.15. Tabla de verdad, expresión algebraica y circuito lógico de una función incompletamente definida



**PROBLEMAS PROPUESTOS**

- 3-4) Determinar la expresión algebraica y obtener un circuito lógico que satisfaga un sistema digital cuya tabla de verdad es la siguiente:

<i>c b a</i>	<i>f</i>
0 0 0	0
0 0 1	1
0 1 0	X
0 1 1	1
1 0 0	0
1 0 1	X
1 1 0	1
1 1 1	1

- 3-5) Determinar la tabla de verdad y obtener un circuito lógico que satisfaga un sistema digital cuya función incompletamente definida es la siguiente:

$$f(c, b, a) = \sum_3 (0, 6, 7) + X(1, 2, 5)$$

- 3-6) Determinar la tabla de verdad y obtener un circuito lógico que satisfaga un sistema digital cuya función incompletamente definida es la siguiente:

$$f(c, b, a) = \prod_3 (3, 4) \cdot X(2, 5, 6)$$

## FUNCIONES LÓGICAS BÁSICAS

---

---

**Objetivos:**

- Conocer las funciones lógicas desde sus aspectos más característicos, tales como: definición, diagrama de *Venn*, conexionado eléctrico, tabla de verdad, simbología, cronograma, expresión algebraica y circuitos comerciales.

**Contenido:** Representación de las funciones lógicas desde varios de sus aspectos característicos.

**Simulación:** Mediante los programas de simulación *Electronics Workbench 5.0*, *OrCAD Demo v9* y el simulador de VHDL *VeriBest VB99.0* se comprueban las distintas funciones lógicas, sus tablas de verdad y cronogramas, simulando una aplicación de cada una de ellas.

## 4.1 FUNCIONES LÓGICAS BÁSICAS



En esta sección se realiza el estudio de las funciones lógicas básicas desde varios aspectos: su definición, operación asociada, circuito eléctrico, tabla de verdad, cronogramas, expresión matemática, simbología, circuitos integrados representativos y ejemplo de aplicación mediante simulación.

La implementación práctica de funciones lógicas se realiza mediante dispositivos electrónicos denominados **puertas lógicas**, siendo éstas los componentes básicos de la electrónica digital.

Las puertas lógicas proporcionan, generalmente en su salida, unos niveles de tensión en función de las tensiones presentes en sus entradas. Estos niveles son diferentes según la tecnología que se haya empleado en el proceso de fabricación, variando de unos dispositivos a otros. El conocimiento preciso de estos valores de tensión no es significativo en las operaciones lógicas, sino los rangos de tensiones entre los que operan las entradas y salidas de una puerta lógica, llamados **niveles lógicos**. Así, existe un rango de tensiones alto (*High*), denominado **nivel lógico alto**  $V_H$  y un rango de tensiones bajo (*Low*), denominado **nivel lógico bajo**  $V_L$ .

El criterio de asignación de los estados cero lógico 0 y uno lógico 1 es totalmente arbitrario. Si se asigna el valor 1 a las tensiones más altas  $V_H$  y el valor 0 a las más bajas  $V_L$ , el convenio utilizado se denomina **lógica positiva**. Si por el contrario se asigna el valor 1 a las tensiones más bajas  $V_L$  y el valor 0 a las más altas  $V_H$ , el convenio utilizado se denomina **lógica negativa**.

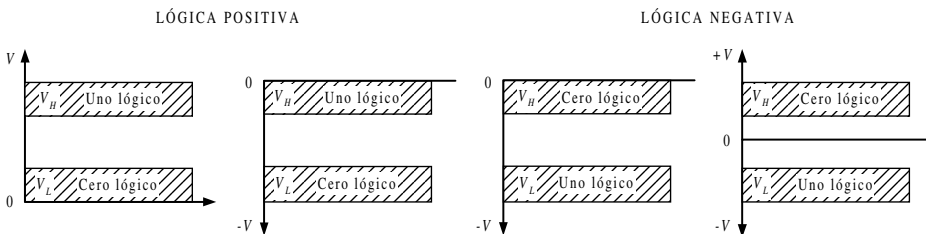


Figura 4.1. Ejemplos del convenio de lógica positiva y negativa

Para la representación gráfica de las funciones lógicas se aplican las normas IEEE 91-1973 (ANSI), que es la más empleada, y la IEEE 91-1984 (DIN o IEC).

En las ecuaciones lógicas, que definen a los sistemas digitales, intervienen varias operaciones para cuya comprensión es muy útil la **teoría de conjuntos**. Seguidamente se enumeran aquellas definiciones y representaciones de la teoría de conjuntos relacionadas con este tema.

Se llama **conjunto** a una reunión de elementos que se caracterizan todos ellos por poseer una propiedad común. Así, por ejemplo la constelación de la Osa Menor es un conjunto de estrellas y la Estrella Polar es un elemento del conjunto.

Se llama **conjunto universal**  $U$  o **conjunto unidad** "1" al que comprende la totalidad de los elementos considerados, por ejemplo todas las constelaciones.

Un conjunto  $A$  es una parte o **subconjunto** del  $U$  si y sólo si todo elemento de  $A$  pertenece a  $U$ . La **representación gráfica de un conjunto** se corresponde con los puntos contenidos en el interior de una figura cualquiera (normalmente suele ser circular o rectangular). En la Figura 4.2 se representa el conjunto universal y un subconjunto de él.

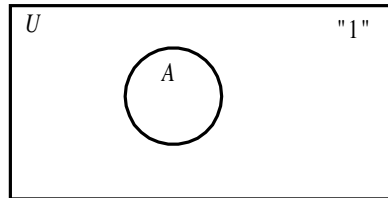


Figura 4.2. Representación de un conjunto universal con un subconjunto de él

La **representación eléctrica de un conjunto** cualquiera puede ser definida mediante un interruptor normalmente abierto, como el de la Figura 4.3. El cambio de estado del interruptor significa la pertenencia al conjunto  $A$  del elemento que se esté considerando, por el contrario la no pertenencia del elemento al conjunto no modifica el estado del interruptor. Por ejemplo, si el conjunto  $A$  representa a los minerales, el elemento gato no pertenece al conjunto y por lo tanto no se modifica el estado del interruptor, manteniéndose en la posición de abierto. Sin embargo, el elemento cuarzo, al pertenecer al conjunto, modifica el estado del interruptor adoptando la posición de cerrado, lo que produce la presencia de tensión en la salida.

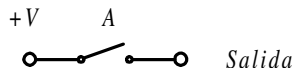


Figura 4.3. Representación eléctrica de un conjunto cualquiera

La **representación eléctrica del conjunto universal** es un interruptor siempre cerrado, como se muestra en la Figura 4.4.

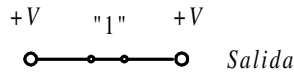


Figura 4.4. Representación eléctrica de un conjunto universal

La **representación eléctrica del conjunto vacío** es un interruptor siempre abierto, como se muestra en la Figura 4.5.

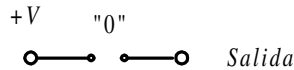


Figura 4.5. Representación eléctrica de un conjunto vacío

Las funciones implementadas en puertas lógicas normalizadas en el diseño digital son las siguientes: AND, OR, NOT, NAND, NOR, SEGUIDOR, XOR y XNOR. En los siguientes apartados se estudian en detalle cada una de ellas.

### 4.1.1 Función AND (puerta AND)

La salida de una puerta AND vale 1 sólo si todas y cada una de las variables de entrada son simultáneamente 1. Se puede considerar también, por el principio de dualidad, que la salida de una puerta AND vale 0 si una cualquiera de sus variables de entrada vale 0.

La función AND efectúa la operación de **producto o intersección de conjuntos**. El producto o intersección de varios conjuntos es otro conjunto formado por los elementos comunes a ellos, como se muestra en la Figura 4.6.

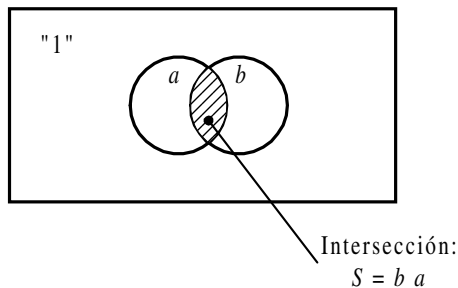


Figura 4.6. El área rayada representa el producto o intersección de los conjuntos  $a$  y  $b$

La función AND realiza la operación de **producto lógico**, siendo su símbolo algebraico “•”, que se lee “por” o también “y”. Es corriente omitir este símbolo, asumiéndose la operación de producto lógico cuando entre dos conjuntos no se especifique ninguna operación.

Desde el punto de vista del **conexionado eléctrico**, se representa la función AND o producto de conjuntos, colocando interruptores en **serie** que simbolizan los factores o elementos físicos considerados como variables de entrada.

En la Figura 4.7 se aprecia cómo la salida sólo presentará un nivel de tensión +V cuando todos los interruptores estén cerrados (hayan sido accionados). Es decir, el elemento considerado ha de pertenecer a la vez a todos los conjuntos.

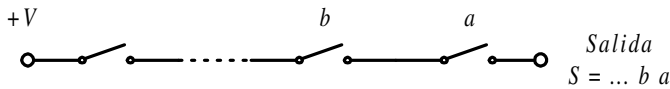


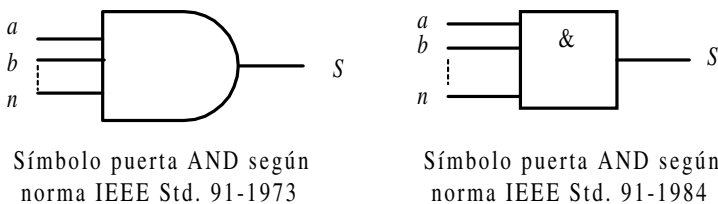
Figura 4.7. Representación eléctrica de la función AND (producto o intersección de conjuntos)

La **tabla de verdad** de una puerta AND de dos variables de entrada se representa en la Tabla 4.1, pudiéndose hacer extensivo a *n* variables (ver Problema resuelto 4-1).

Tabla 4.1. Tabla de verdad de una puerta AND de dos variables de entrada

<i>b a</i>	<i>S</i>
0 0	0
0 1	0
1 0	0
1 1	1

El **símbolo** de la puerta AND es el representado en la Figura 4.8.



Símbolo puerta AND según norma IEEE Std. 91-1973

Símbolo puerta AND según norma IEEE Std. 91-1984

Figura 4.8. Símbolos de la puerta lógica AND

El **cronograma** de la puerta AND de dos entradas, que muestra la relación existente, a lo largo del tiempo, entre sus entradas y su salida, se representa en la Figura 4.9.

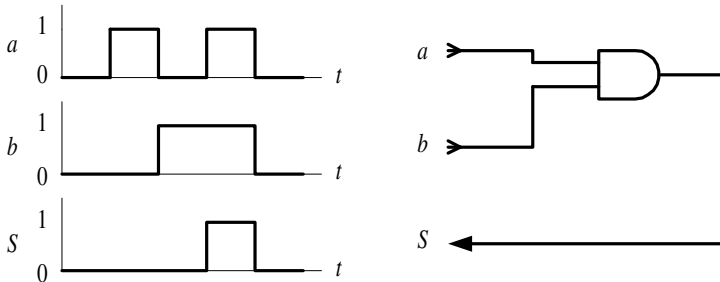


Figura 4.9. Cronograma de una puerta AND de dos entradas

La **expresión algebraica** de una puerta AND, considerando  $a, b, c, \dots$  a las variables de entrada y  $S$  a la salida, es la siguiente:

$$S = f(\dots, c, b, a) = \dots cba \quad [4.1]$$



Mediante el **lenguaje de descripción hardware VHDL** se puede modelar los sistemas digitales para su simulación o su síntesis, como por ejemplo una puerta AND.

El Apéndice B contiene un breve manual del lenguaje VHDL y el Apéndice C una breve descripción de herramientas de simulación VHDL. El **Apéndice D** contiene un breve tutorial en el que se describe cómo utilizar la herramienta *OrCAD Demo v9* para realizar simulaciones VHDL con dicha herramienta (se aconseja el uso de esta herramienta para aquellos lectores que no dispongan del programa *VeriBest VHDL*).

**Nota importante:** Si al simular un ejercicio en VeriBest V99.0 se produce un error al compilar, indicando: **[error] library ...\WORKLIB no found**, es debido a que esta librería no se encuentra en el mismo *path* o trayectoria que la realizada en la definición original. Para solucionarlo basta con ejecutar la opción **Workspace/Reinitialize lib Environment**, con lo que se crea en el subdirectorio por defecto una carpeta con esta librería.

Para los lectores que quieran simular este ejercicio, deben tener instalado el programa *VeriBest V99.0* y abrir el fichero cuya ruta es la siguiente:

**D:\Ejemplos\Cap04\VBv99\AND\_algo\AND\_algo.vpd**

Al abrir este fichero se obtiene el resultado que se muestra en la ventana de la Figura 4.11. Los lectores con conocimientos de este programa pueden saltarse los

párrafos siguientes en los que se describe la forma de crear el ejercicio de simulación propuesto.

En este primer ejemplo de simulación con el programa *VeriBest V99.0* se hace una breve descripción para aquellos lectores que prefieran crear la simulación del comportamiento de una puerta AND de tres entradas, debiendo realizar los siguientes pasos:

- Se abre la aplicación Inicio/Programas/*VeriBest V99.0/VeriBest VHDL simulador/VeriBest VHDL* y se crea un nuevo espacio de trabajo mediante el comando **File/New**, seleccionando *VHDL Workspace* y haciendo clic en la opción *OK*. Tras especificar el nombre del espacio de trabajo en la casilla *Workspace Name* se pulsa la opción *Create*.
- De la misma forma se crea un fichero fuente VHDL, mediante el comando **File/New** seleccionando *VHDL Source File* y haciendo clic en la opción *OK* se crea una ventana de nombre *VHDL1* en la que se puede escribir el código fuente VHDL.

En la Figura 4.10 se muestra el entorno del programa de simulación *VeriBest VHDL simulador* después de crear un espacio de trabajo y un fichero fuente VHDL.

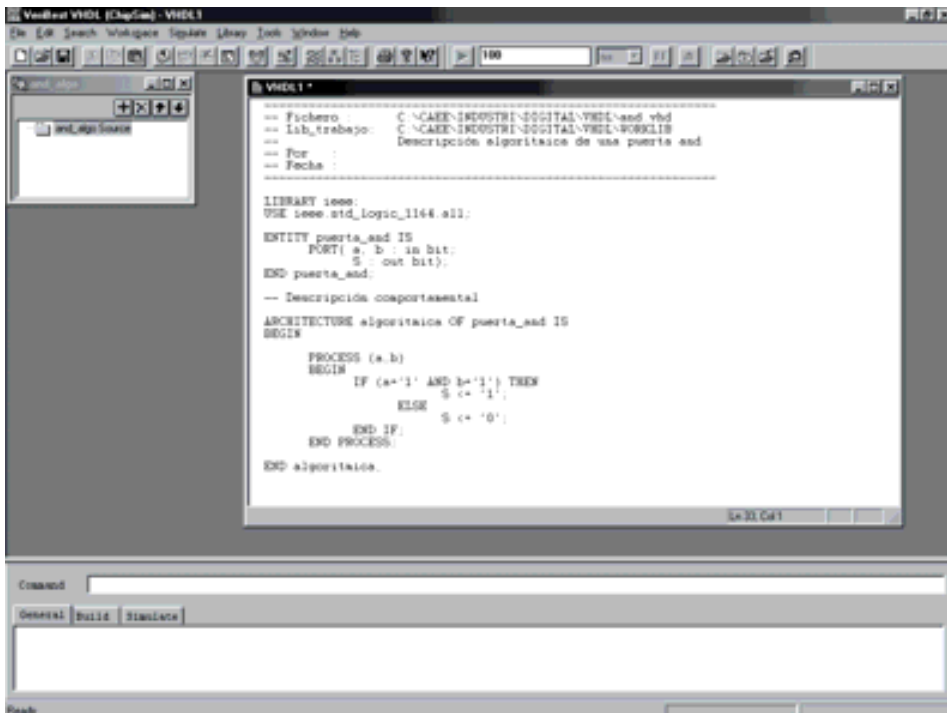


Figura 4.10. Programa de simulación *VeriBest VHDL simulador* después de crear un espacio de trabajo y un fichero fuente VHDL



El lenguaje VHDL presenta tres estilos diferentes de descripción de sistemas dependiendo del nivel de abstracción. A continuación, se desarrollan los tres estilos de descripción, con una explicación de las características principales de cada uno de ellos, siendo conveniente leer el Apéndice B (breve manual de VHDL) para comprender mejor estos conceptos.

Como ejemplo en esta primera descripción *hardware* de una puerta AND se desarrollan los tres estilos con objeto de que el lector pueda compararles, identificar su nivel de abstracción y sacar conclusiones. En el resto de descripciones *hardware* sólo se desarrollará el estilo más apropiado o uno diferente en cada caso para evitar que este capítulo resulte demasiado extenso.

- 1) **Descripción comportamental algorítmica.** Es la más cercana a un lenguaje convencional. Está formada por el bloque PROCESS, similar a una subrutina, cuyas instrucciones se ejecutan secuencialmente. En esta descripción no se indican ni componentes, ni interconexiones, sólo su comportamiento o funcionamiento. En la Figura 4.11 se muestra un ejemplo de aplicación de este tipo de descripción.

La ruta y el nombre del fichero que contiene esta descripción VHDL es la que se indica a continuación:

**D:\Ejemplos\Cap04\VBv99\AND\_algo\AND\_algo.vpd**



```
-- Fichero : C:\CAEE\INDUSTRI\DIGITAL\VHDL\and.vhd
-- Lib_trabajo: C:\CAEE\INDUSTRI\DIGITAL\VHDL\WORKLIB
-- Descripción algorítmica de una puerta and
-- Por :
-- Fecha :
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY puerta_and IS
    PORT( a, b : in bit;
          S : out bit);
END puerta_and;

-- Descripción comportamental

ARCHITECTURE algorítmica OF puerta_and IS
BEGIN

    PROCESS (a,b)
    BEGIN
        IF (a='1' AND b='1') THEN
            S <= '1';
        ELSE
            S <= '0';
        END IF;
    END PROCESS;

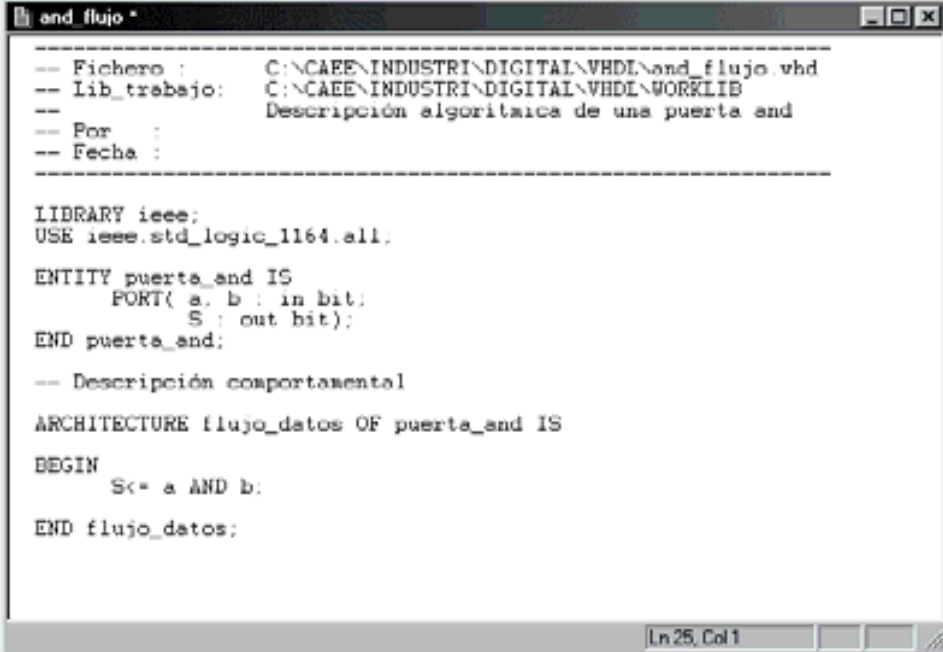
END algorítmica;
```

Figura 4.11. Descripción comportamental algorítmica de una puerta AND mediante VHDL

- 2) **Descripción comportamental por flujo de datos** o también llamada de **transferencia entre registros (RTL)**. Es más cercana a la realización física o estructural aunque sin llegar a serlo ya que si bien describe componentes y asigna señales, no constituye una lista de componentes e interconexiones. Permite realizar varias instrucciones en paralelo, siendo una descripción concurrente. En la Figura 4.12 se muestra un ejemplo de este tipo de aplicación.

La ruta y el nombre del fichero que contiene esta descripción VHDL es la que se indica a continuación:

**D:\Ejemplos\Cap04\VBv99\AND\_flujo\AND\_flujo.vpd**



```
-- Fichero : C:\CAEE\INDUSTRI\DIGITAL\VHDL\and_flujo.vhd
-- Lib_treabajo: C:\CAEE\INDUSTRI\DIGITAL\VHDL\WORKLIB
-- Por : Descripción algorítmica de una puerta and
-- Fecha :
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY puerta_and IS
    PORT( a, b : in bit;
          S : out bit);
END puerta_and;

-- Descripción comportamental

ARCHITECTURE flujo_datos OF puerta_and IS

BEGIN
    S<= a AND b;

END flujo_datos;
```

Figura 4.12. Descripción comportamental por flujo de datos de una puerta AND mediante VHDL

- 3) **Descripción estructural**. Es la que más se acerca a la realización física de un circuito y puede considerarse como una lista de componentes e interconexiones (*Netlist*). La descripción es mucho más larga y menos clara que las anteriormente indicadas. Los componentes son entidades definidas en bibliotecas y para las conexiones se utilizan señales internas definidas al principio. A cada componente se le asigna un símbolo, hecho este que se denomina **replicación**. En la Figura 4.13 se muestra un ejemplo de este tipo de descripción.

La ruta y el nombre del fichero que contiene esta descripción VHDL es la que se indica a continuación:

**D:\Ejemplos\Cap04\VBv99\AND\_estr\AND\_estr.vpd**

```

and_estr
-----
-- Fichero :      C:\CAEE\INDUSTRI\DIGITAL\VHDL\and_estr.vhd
-- Lib_trabajo:  C:\CAEE\INDUSTRI\DIGITAL\VHDL\WORKLIB
-- Por          Descripción estructural de una puerta and
-- Fecha
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Definición de entidades y arquitectura de cada componente
-- (en este caso sólo uno)

ENTITY componente_and IS
    PORT( ap, bp : in bit;
          Sp : out bit);
END componente_and;

ARCHITECTURE rtl OF componente_and IS
BEGIN
    Sp <= ap AND bp;
END rtl;

-- Definición de la entidad y arquitectura del conjunto o circuito (netlist)

ENTITY puerta_and IS
    PORT (a,b: IN bit;
          S: OUT bit);
END puerta_and;

ARCHITECTURE estructural OF puerta_and IS
    COMPONENT componente_and IS
        PORT (ap,bp: IN bit; Sp: OUT bit);
    END COMPONENT componente_and;

BEGIN
    u0: componente_and PORT MAP (ap=>a,bp=>b,Sp=>S);
END estructural;
Ln 24, Col 10

```

*Figura 4.13. Descripción estructural de una puerta AND mediante VHDL*

Se ha creado un nuevo fichero fuente VHDL para definir los estímulos y poder simular el comportamiento de una puerta AND. Este fichero, al que se le ha dado el nombre `and_stim.vhd` y que se muestra en la Figura 4.14, se ha incluido en la ruta indicada para cada uno de los directorios correspondientes a las tres descripciones VHDL anteriores (algorítmica, flujo de datos y estructural).

```

and_stim
-----
-- Fichero : C:\CAEE\INDUSTRI\DIGITAL\VHDL\and_stim.vhd
-- Lib_trabajo: C:\CAEE\INDUSTRI\DIGITAL\VHDL\WORKLIB
--
-- Por :
-- Fecha :
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY estímulos IS
END estímulos;

ARCHITECTURE and_stim OF estímulos IS

    COMPONENT puerta_and
        PORT (a,b: IN bit;
              S: OUT bit);
    END COMPONENT;

    SIGNAL a,b,S: bit;

BEGIN

    puerta_and1: puerta_and PORT MAP (a,b,S);

    estímulos: PROCESS
    BEGIN
        a <= '0';
        b <= '0';
        WAIT FOR 50 ns;
        a <= '1';
        WAIT FOR 50 ns;
        a <= '0';
        b <= '1';
        WAIT FOR 50 ns;
        a <= '1';
        WAIT FOR 50 ns;
    END PROCESS;

END and_stim;
Ln 44, Col 1

```

Figura 4.14. Fichero de estímulos para la simulación de una puerta AND, mediante lenguaje VHDL

Como cualquier lenguaje de programación, en la simulación mediante VHDL, se deben realizar los siguientes pasos: compilar el fichero fuente creado, ejecutar el simulador y visualizar los resultados obtenidos.

- La **compilación** se realiza mediante el comando *Workspace/Compile All*, o pulsando la combinación [SHIFT+F8], o haciendo clic sobre el botón de la barra de herramientas cuyo icono es el siguiente:



- Una vez compilado sin errores se puede simular, pero antes, para evitar errores al ejecutar el simulador, se debe **elegir la entidad y arquitectura raíz** para la

simulación mediante el comando **Workspace/Settings**, o haciendo clic sobre el botón de la barra de herramientas cuyo icono es el siguiente:



- Así, aparecerá el menú que se muestra en la Figura 4.15 con varias solapas de las cuales se debe seleccionar **Simulate** y abrir la carpeta **WORK**, que es la biblioteca donde se ha compilado el diseño. Haciendo clic sobre los nombres de las arquitecturas y entidades que aparecen a la izquierda se selecciona la arquitectura que se quiere simular y se hace clic sobre el botón **Set** y sobre el botón **Aceptar**, quedando especificada la arquitectura a simular.

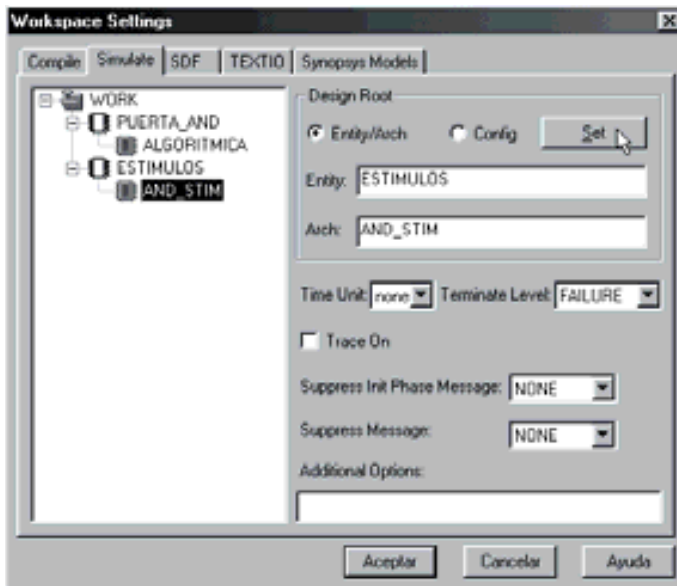


Figura 4.15. Ventana de configuración de la entidad y arquitectura a simular

- Para **simular** hay que seleccionar el comando **Workspace/Execute Simulator**, o pulsar la combinación de teclas [SHIFT+F5], o hacer clic sobre el botón de la barra de herramientas cuyo icono se muestra a continuación. En el caso de que no se disponga de licencia, mientras se ejecuta el simulador aparece un mensaje indicando que éste funciona con limitaciones.



- Para **definir las señales a visualizar** se selecciona el comando **Tools/New WaveForm Window**, o hacer clic sobre el botón de la barra de herramientas cuyo icono es el siguiente:



- En la ventana que aparece se deben especificar las señales a visualizar, para ello se hace clic sobre el botón siguiente:



- Aparece una lista con todas las señales, pudiéndose seleccionar las que más interesen o todas mediante la opción **Add All**, terminando con la opción **Close** de este menú, como se muestra en la Figura 4.16.

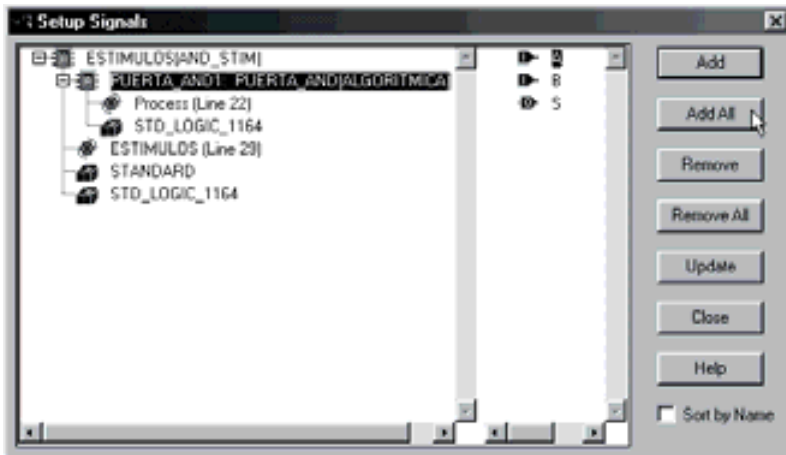


Figura 4.16. Selección de las señales a visualizar en el simulador

- El último paso consiste en **ejecutar el simulador** para ver las señales que se producen durante un determinado periodo de tiempo que se especifica en la barra de herramientas cuyo icono es el siguiente:



Posteriormente, se selecciona el comando **Simulate/Run**, o se pulsa la tecla [F5], o se hace clic sobre el botón de la barra de herramientas cuyo icono es el siguiente:



Con lo que se obtiene el cronograma representado en la Figura 4.17, cuyo resultado es el mismo para las diferentes descripciones de la puerta AND, como es de esperar.

- Para **salir de la simulación** se selecciona el comando **Simulate/Quit**, o se pulsa la combinación [CTRL+Q], o se hace clic sobre el botón de la barra de herramientas cuyo icono es:



Los **circuitos comerciales** más representativos de puertas AND son los siguientes:

- 7408: Cuádruple puerta AND de dos entradas.
- 7409: Cuádruple puerta AND de dos entradas con salidas colector abierto.
- 7411: Triple puerta AND de tres entradas.
- 7415: Triple puerta AND de tres entradas con salidas colector abierto.
- 7421: Doble puerta AND de cuatro entradas.

Mediante el programa de Fairchild Semiconductor para **Data Sheets** que se incluye en el CDROM#2 que acompaña a este libro se pueden buscar las características de los circuitos comerciales de puertas AND, en especial aquéllos con las referencias anteriormente indicadas.

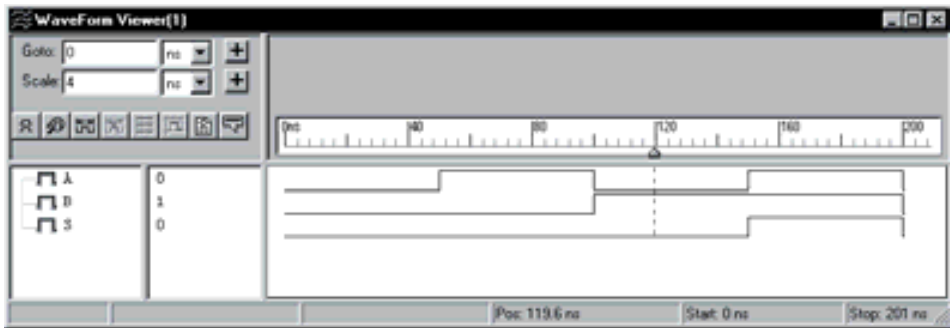


Figura 4.17. Resultado de la simulación de una puerta AND con VHDL

A continuación, se desarrollan ejercicios y **ejemplos de aplicación** de puertas AND utilizando los programas de simulación *Electronics Workbench 5.0* (EWB) y *OrCAD Demo v9*.

### PROBLEMA RESUELTO 4-1

Comprobar mediante los programas de simulación el comportamiento de la puerta lógica AND.



Simular el comportamiento de una puerta lógica AND de tres entradas y obtener su tabla de verdad y cronograma, utilizando para ello el programa *Electronics Workbench*.

### Solución:

Para comprender mejor este proceso se ha elaborado el manual y tutorial del programa de simulación *Electronics Workbench 5.0* en el Apéndice D.

Para aquellos lectores que quieran directamente simular este ejercicio sin necesidad de crearlo, sólo deben tener instalado el programa *Electronics Workbench 5.0* y abrir el fichero cuya ruta es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W0\_01.ewb**

con lo que se obtiene la ventana de la Figura 4.18. Los lectores familiarizados con el manejo de este programa pueden saltarse los párrafos siguientes en los que se describen la forma de crear el ejercicio de simulación propuesto.

En este primer ejemplo de simulación con el programa *Electronics Workbench 5.0* se hace una breve descripción para aquellos lectores que prefieran crear la simulación del comportamiento de una puerta AND de tres entradas, debiendo realizar los siguientes pasos:

- Desde *Windows 95 o 98* se abre la **aplicación** Inicio/*Electronics Workbench 5.0/Electronics Workbench*.
- Se crea un nuevo diseño mediante el comando Archivo/Nuevo apareciendo una ventana o área de trabajo sin título (*untitled*).
- Sobre el área de trabajo se colocan los componentes, seleccionando en el banco de componentes el elemento deseado. En este ejemplo sólo se requiere la puerta lógica AND situada en el banco de componentes denominado **Compuertas lógicas**, cuyo icono es el siguiente:



y también el convertidor lógico situado en el banco de componentes de *Instrumentos*.

- Se asignan **valores a los componentes**. En el ejemplo sólo se requiere definir que la puerta AND tenga tres entradas haciendo doble clic en dicho componente. Al abrirse una ventana denominada **Propiedades Puerta AND de 2 entradas**, se selecciona la pestaña **Número de entradas** y se asigna el valor **3**.
- Se realizan las conexiones entre componentes; para ello se aproxima el puntero del ratón a uno de los terminales del componente hasta que aparezca un círculo o punto de conexión. En ese instante se pulsa el botón izquierdo del ratón y sin soltarlo, arrastrarlo hacia el otro terminal a unir hasta que igualmente presente un círculo o punto de conexión, momento en el que se puede soltar el botón del ratón, con lo que se produce la conexión. Se aprecia cómo al arrastrar con el ratón aparece una línea que



representa el cable de conexión. En el ejemplo se han conectado las entradas y salidas de la puerta AND al convertidor lógico. En la Figura 4.18 se muestra cómo queda el esquema después de realizar los pasos anteriores (abrir la aplicación, crear un diseño nuevo, colocar los componentes, asignar valores a los componentes y realizar las conexiones).

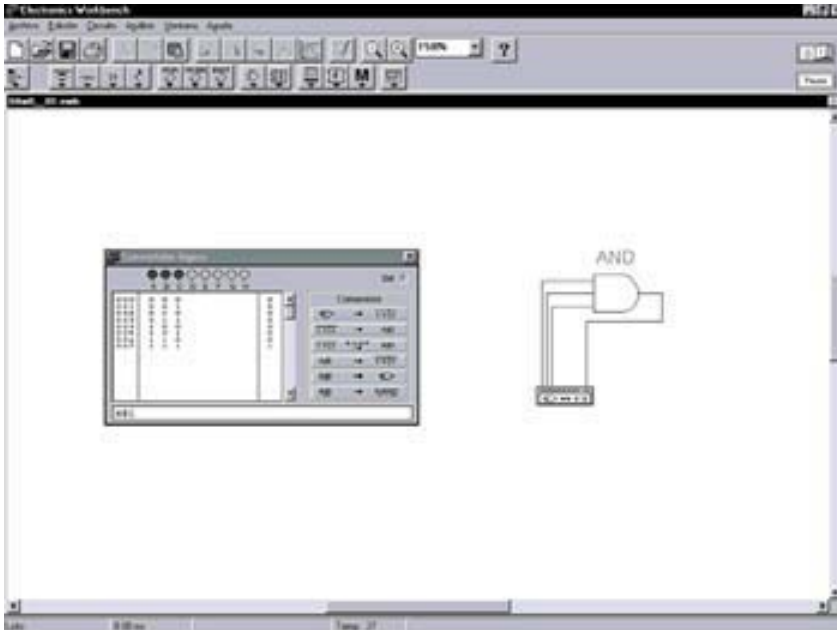
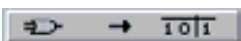


Figura 4.18. Obtención de la tabla de verdad de una puerta AND de tres entradas utilizando Electronics Workbench

- En el convertidor lógico del programa de simulación *Electronics Workbench*, se puede obtener la tabla de verdad de una puerta AND de tres entradas de dos formas diferentes: introduciendo su expresión algebraica o representando su circuito lógico y conectando las entradas y salidas de la puerta AND al convertidor lógico, como se muestra en la Figura 4.18. Posteriormente, según la conversión que se elija, se activa el botón de conversión de expresión algebraica a tabla de verdad, cuyo icono es el siguiente,



o la conversión de circuito lógico a tabla de verdad, cuyo icono es:



Para obtener el cronograma de una puerta AND de tres entradas se deben realizar los siguientes pasos:

- Se coloca en el área de trabajo y se conecta a las tres entradas de la puerta AND el generador de palabras, disponible en el banco de componentes denominado **Instrumentos**. Se define, en dicho generador de palabras, las posibles combinaciones diferentes que pueden tomar las señales de entrada.
- Se coloca en el área de trabajo y se conectan las entradas y salidas de la puerta AND al analizador lógico, disponible en el banco de componentes denominado **Instrumentos**.
- Por último, se activa la simulación mediante el comando **Análisis/Activar**, o pulsando la combinación [CTRL+G], o accionando el interruptor de activar simulación cuya representación en la barra de herramientas es la siguiente,



con lo cual se visualiza en el analizador lógico el cronograma de una puerta AND de tres entradas, según se muestra en la Figura 4.19.

La ruta y el nombre del fichero que contiene el esquema de este circuito es el que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W1\_\_01.ewb**



Simular con el programa *OrCAD Demo* el comportamiento de una puerta lógica AND de tres entradas y obtener su cronograma. A partir de éste se puede obtener de forma inmediata su tabla de verdad.

### Solución:

Para comprender mejor este proceso se ha elaborado el manual y tutorial del programa de simulación *OrCAD Demo* en el Apéndice E.

Para aquellos lectores que quieran directamente simular este ejercicio sin necesidad de crearlo, sólo deben tener instalado el programa *OrCAD Demo* y abrir el fichero cuya ruta es la que se indica a continuación:

**D:\Ejemplos\Cap04\OrCAD9\04R0\_\_01\04R0\_\_01.opj**

con lo que se obtiene la ventana de la Figura 4.21. Los lectores con conocimientos de este programa pueden saltarse los párrafos siguientes en los que se describen la forma de crear el ejercicio de simulación propuesto.

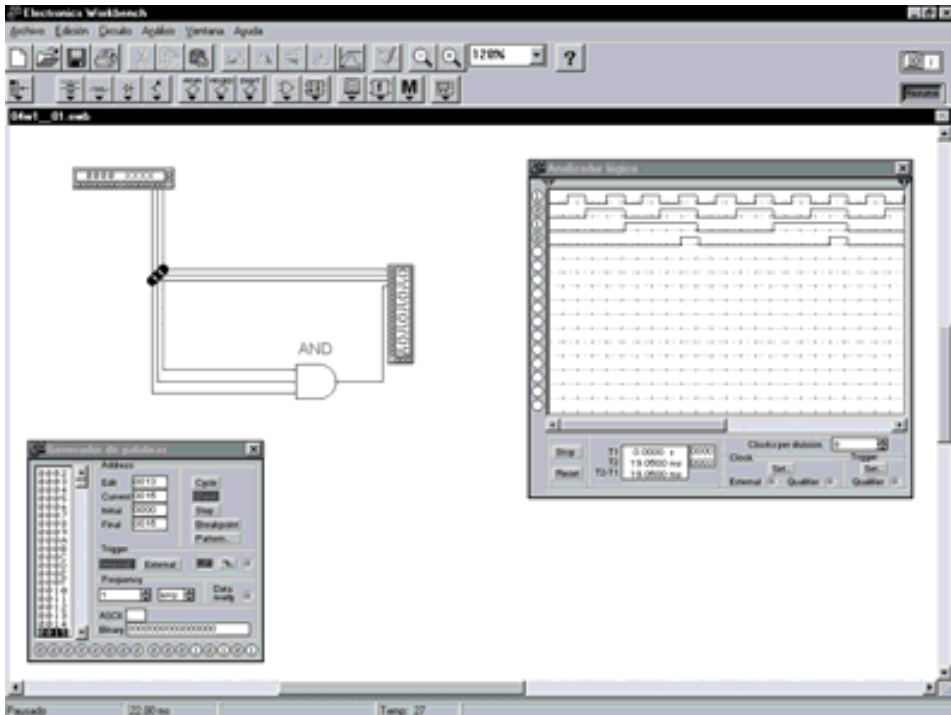


Figura 4.19. Obtención del cronograma de una puerta AND de tres entradas utilizando Electronics Workbench

En este primer ejemplo de simulación con el programa *OrCAD Demo* se hace una breve descripción para aquellos lectores que prefieran crear la simulación del comportamiento de una puerta AND de tres entradas, debiendo realizar los siguientes pasos:

- Se abre la aplicación *OrCAD Capture* y se crea un nuevo proyecto mediante el comando **File/New/Project...**. En la ventana que aparece al activar este comando (Figura 4.20) se asigna un nombre de proyecto en la casilla Name, se activa la opción **Analog or Mixed-Signal Circuit Wizard**, se selecciona el directorio donde se quiere guardar el proyecto y se pulsa **OK**.
- En la ventana que aparece, en la que realiza la captura de esquemáticos y que se denomina (SCHEMATIC1:PAGE1) se procede a colocar los componentes mediante el comando **Place/Part...**, o pulsando la combinación [SHIFT+P], o haciendo clic sobre el botón de la barra de herramientas cuyo icono es el siguiente:



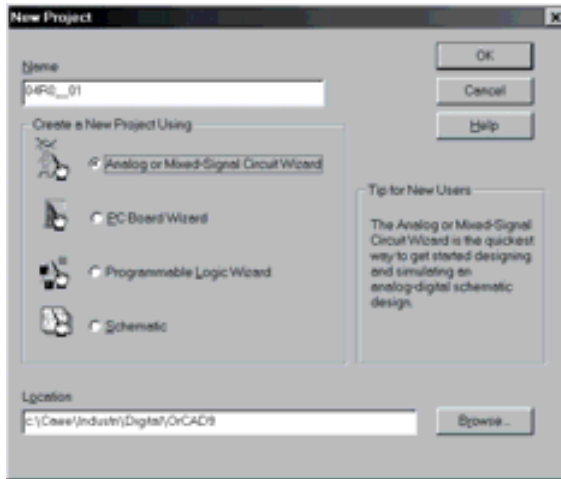


Figura 4.20. Ventana para definir un nuevo proyecto en OrCAD Capture

- A continuación, se selecciona el modelo de una puerta AND de tres entradas, cuyo circuito comercial es el 7411 que incorpora la librería de *PSpice* denominada EVAL y los dispositivos de estímulos digitales DigStim1 de la librería de *PSpice* denominada SOURCSTM.
- Se realizan las conexiones entre componentes mediante el comando **Place/Wire**, o pulsando la combinación [SHIFT+W] o haciendo clic sobre el botón de la barra de herramientas, cuyo icono es el siguiente:



- Se definen los puntos de prueba en los que se quiere visualizar las señales del circuito, que en este caso son la entrada y la salida del mismo. Para ello se selecciona el botón de la barra de herramientas, cuyo icono es el siguiente:



- Se etiqueta el nombre de las señales *a*, *b*, *c* y *S*, para su posterior visualización en la aplicación *OrCAD PSpice A/D Demo*. El botón de la barra de herramientas que asigna una etiqueta o alias a una conexión tiene el siguiente icono:



- En la Figura 4.21 se muestra cómo queda el esquema después de realizar los pasos anteriores (colocar componentes, realizar las conexiones, definir puntos de prueba y etiquetar señales).

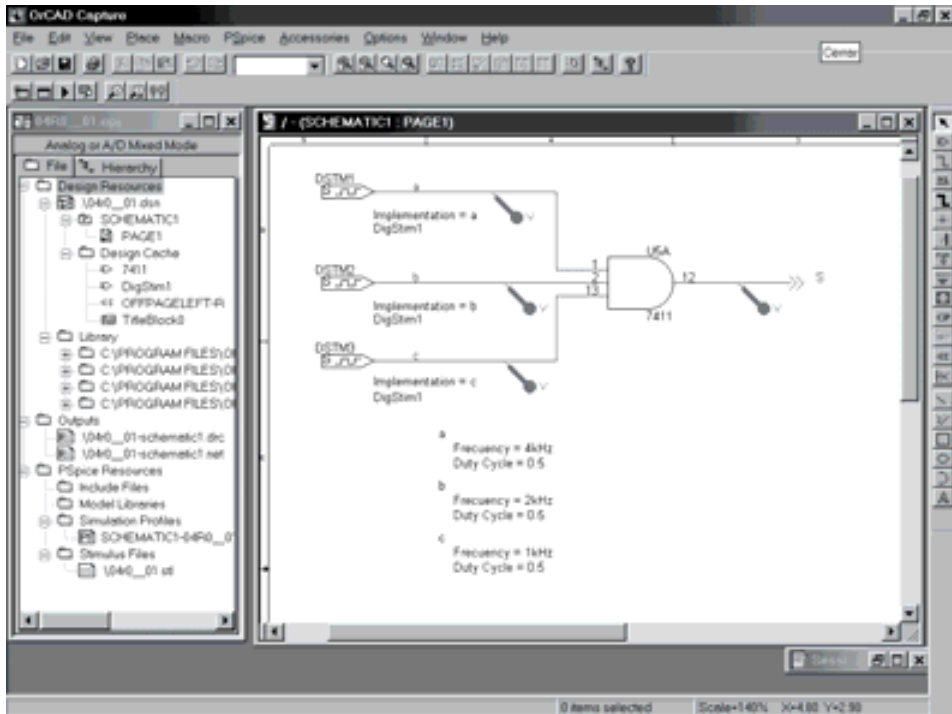


Figura 4.21. Esquema del circuito con una puerta AND de tres entradas con OrCAD Capture

- Se introducen los estímulos de entrada mediante el modelo DigStim1, en los componentes DSTM1, DSTM2 y DSTM3. En cada uno de los componentes DSTM se definen un nombre de estímulo haciendo doble clic sobre la palabra **Implementation**, apareciendo una ventana con la casilla **Value** en la que se asigna el valor a para DSTM1, b para DSTM2 y c para DSTM3, que posteriormente se concretarán en la aplicación *PSpice Stimulus Editor*.
- Mediante la aplicación *PSpice Stimulus Editor* se definen las características de los estímulos implementados en los modelos DSTM1, DSTM2 y DSTM3 como a, b y c respectivamente. Para ello se abre la aplicación *PSpice Stimulus Editor* y se ejecuta el comando **File/New** que abre una ventana para visualizar los estímulos. Posteriormente, se ejecuta el comando **Stimulus/New**, con lo que se abre una nueva ventana en la que se puede introducir el nombre del estímulo en la casilla **Name** (por ejemplo el estímulo a). Se selecciona el tipo **clock** en las opciones de estímulos de **digital** y se acepta la selección pulsando **OK**, abriéndose otra ventana en la que se puede definir la frecuencia de reloj mediante la opción **Frequency (Hz)** y el ciclo de trabajo mediante la opción **Duty cycle (%)**. En la Figura 4.22 se muestra la aplicación *PSpice Stimulus*

*Editor* con los tres estímulos, y el cuadro de diálogo que muestra las características para el estímulo *a*. Como puede observarse en dicha figura, para el estímulo *a*: Frecuencia = 4 kHz y *Duty cycle* = 0.5; para el estímulo *b*: Frecuencia = 2 kHz y *Duty cycle* = 0.5; y para el estímulo *c*: Frecuencia = 1 kHz y *Duty cycle* = 0.5.

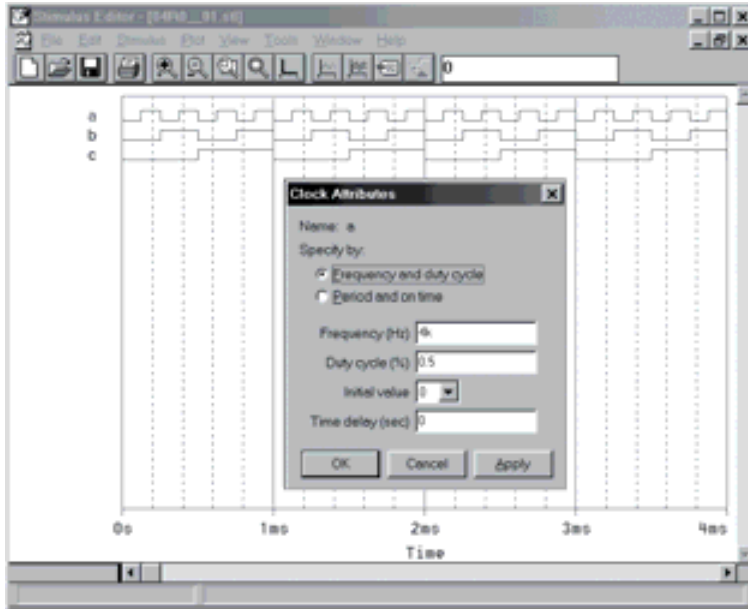


Figura 4.22. Definición de los estímulos en la aplicación PSpice Stimulus Editor

- Se define un perfil de simulación mediante el comando **PSpice/New Simulation Profile** o seleccionando en la barra de herramientas el icono que se muestra a continuación, que permite dar un nombre a la simulación en la casilla Name.



- Mediante el comando **PSpice/Edit Simulation Setting** o seleccionando en la barra de herramientas el icono que se muestra a continuación, se puede seleccionar el tipo de análisis a simular.



- Para ver el comportamiento de la puerta AND se realiza un análisis transitorio **Time Domain (Transient)**. Los ajustes realizados en el cuadro de diálogo de dicho comando se muestran en la Figura 4.23.

También hay que definir el nombre del fichero de estímulos en la ficha *Stimulus* de este mismo cuadro, como se muestra en la Figura 4.24.

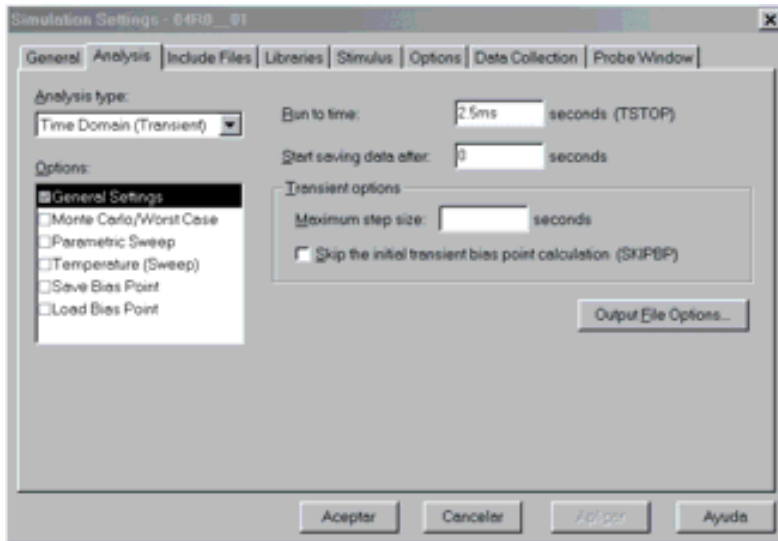


Figura 4.23. Valores utilizados para el análisis transitorio de la puerta AND

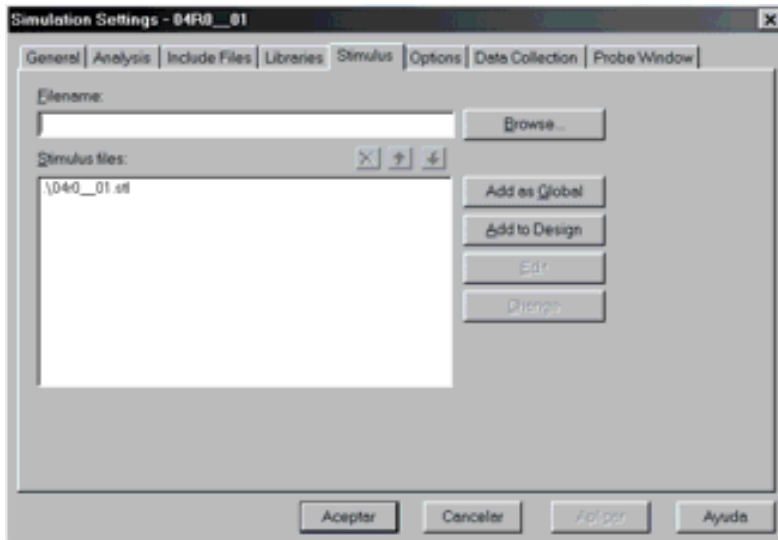


Figura 4.24. Definición del nombre del fichero de estímulos

- Ejecutando el comando **PSpice/Run** o seleccionando en la barra de herramientas el icono que se muestra a continuación, se abre la aplicación *OrCAD PSpice A/D Demo* y se obtienen los resultados de la Figura 4.25.



En dicha aplicación se ha activado el cursor, con lo que se puede ver, al lado del nombre de las señales, el valor que toma cada una de ellas en el instante en el cual se sitúa el mismo.

En el caso de la Figura 4.25 se pueden comprobar los valores de las señales en el instante en el que está situado el cursor, siendo:  $a = 1$ ,  $b = 0$ ,  $c = 0$ ,  $S = 0$ . Situando el cursor en cada uno de los puntos de interés, por ejemplo, en cada uno de los sucesivos valores que van tomando las señales se puede obtener la tabla de verdad del circuito. Además, en la ventana del mismo, aparece también el instante de tiempo correspondiente al punto del eje X en el que se encuentra situado el cursor, el valor de la señal seleccionada y el valor de la señal de salida.



Figura 4.25. Resultados obtenidos en OrCAD PSpice A/D Demo para la puerta AND de tres entradas



## PROBLEMA RESUELTO 4-2



Simular con el programa *Electronics Workbench* un circuito que permita habilitar o inhabilitar el paso de una señal de reloj (tren de impulsos) mediante una entrada de control (habilitación).

### Solución:

Este problema corresponde a una aplicación típica de las puertas AND, también llamadas puertas cerrojo por comportarse como una llave que permite, o no permite, el paso de una señal digital, según se muestra en la Figura 4.26.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W0\_\_02.ewb**

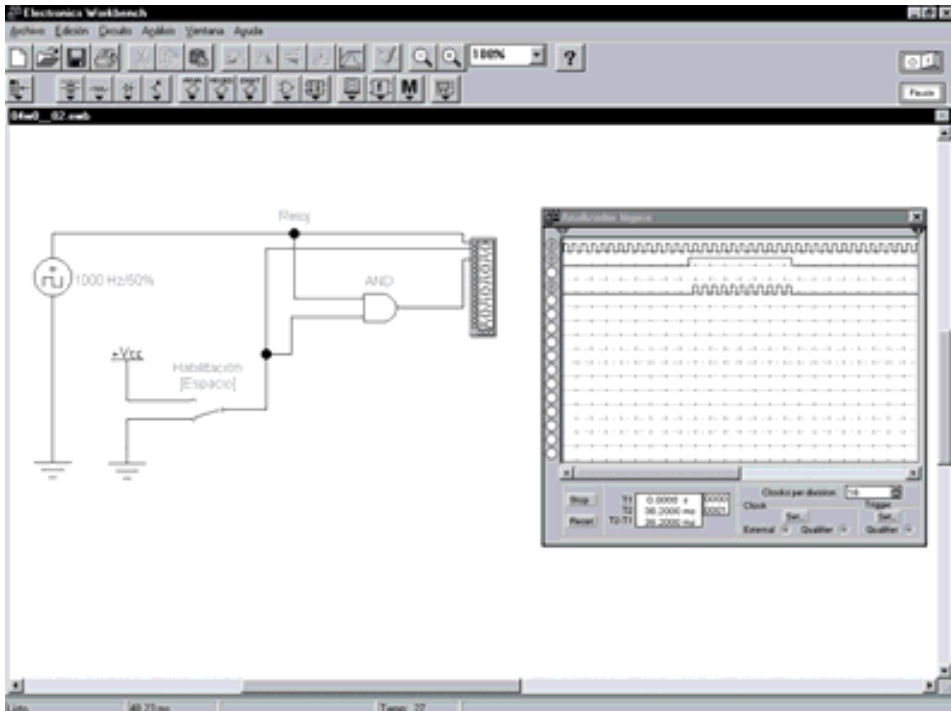
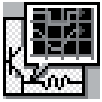


Figura 4.26. Circuito cerrojo utilizando la aplicación *Electronics Workbench*



Simular con el programa *OrCAD Demo* un circuito que permita habilitar o inhabilitar el paso de una señal de reloj (tren de impulsos) mediante una entrada de control (habilitación).

### Solución:

Para simular el comportamiento de este circuito se utiliza el esquema que se muestra en la Figura 4.27.

La ruta y el nombre del fichero que contiene el fichero proyecto de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\OrCAD9\04R0\_\_02\04R0\_\_02.opj**

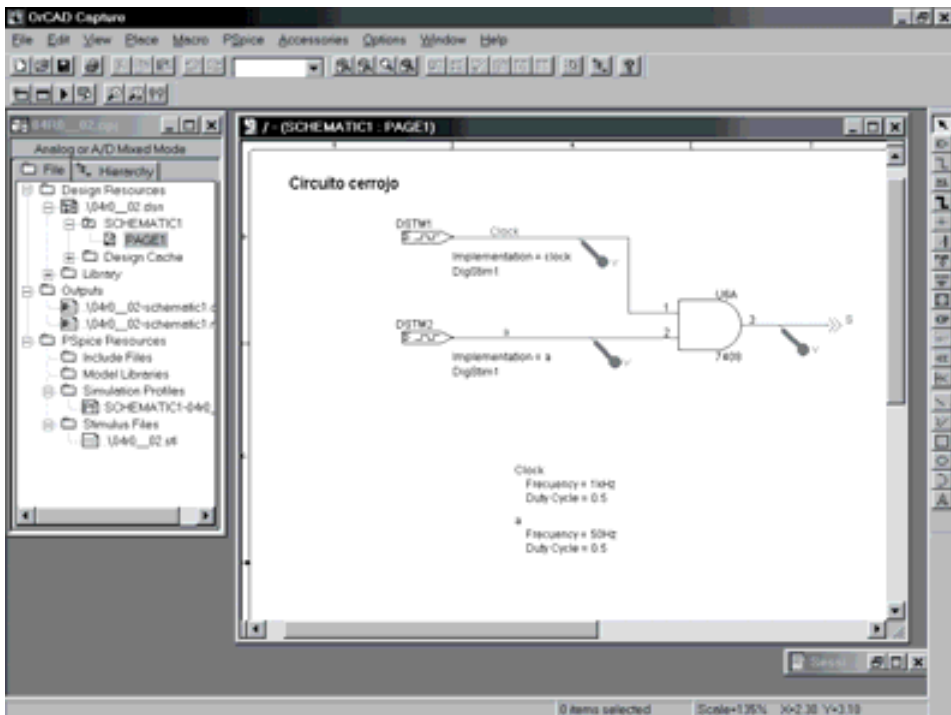


Figura 4.27. Esquema del circuito cerrojo utilizado la aplicación OrCAD Capture

En dicha figura se han representado los valores de las señales de entrada (*Clock* y  $U_c$ ), que deben definirse en la aplicación *PSpice Stimulus Editor* al igual que se ha hecho en el problema anterior.

Para estudiar el comportamiento de este circuito se realiza un análisis transitorio (***Transient...***). Los ajustes realizados en el cuadro de diálogo de dicho comando se muestran en la Figura 4.28. También hay que definir el

nombre del fichero de estímulos en la ficha *Stimulus* de este mismo cuadro de diálogo.

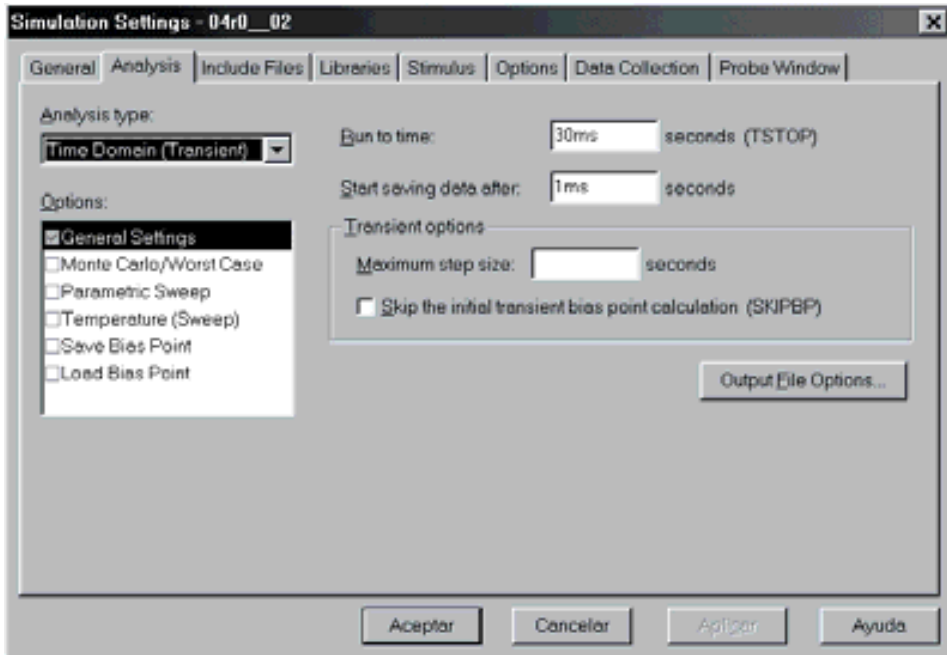


Figura 4.28. Configuración del análisis transitorio de la puerta AND de tres entradas

En la Figura 4.29 se muestran los resultados que se obtienen en la aplicación de visualización de los resultados, *OrCAD PSpice A/D Demo*, después de realizar la simulación.

## PROBLEMAS PROPUESTOS

- 4-1) Diseñar un circuito eléctrico que realice la función AND con tres interruptores. Comprobarlo utilizando los programas de simulación.
- 4-2) Diseñar un subcircuito que forme parte de una alarma que active una sirena  $S$  cuando la alarma esté conectada  $a$  y se produzca la activación de un sensor de rotura de cristal  $b$ .

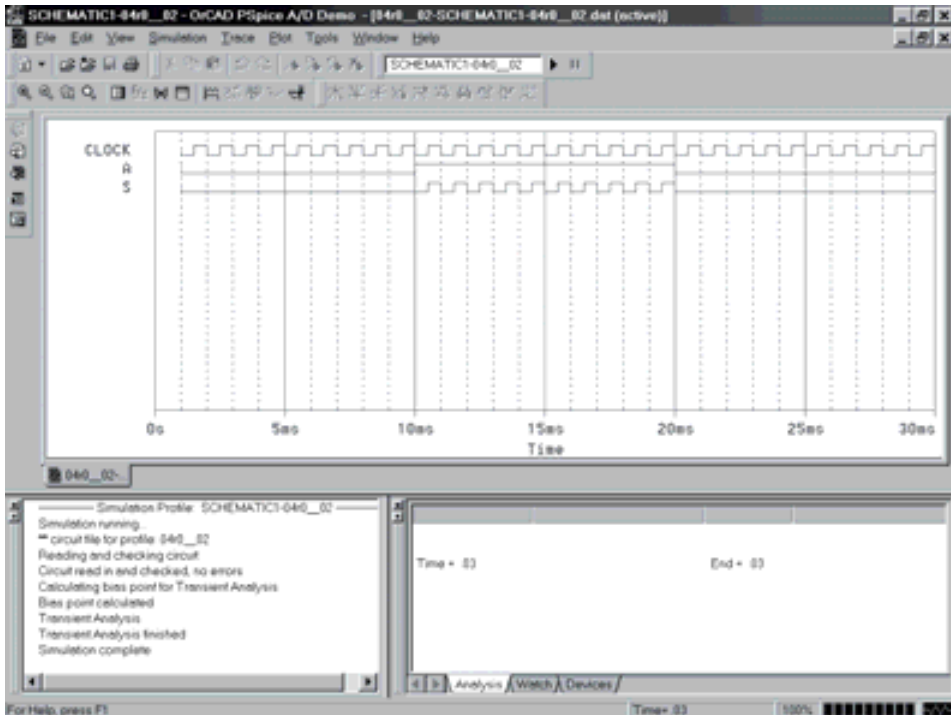


Figura 4.29. Resultados para el circuito cerrojo con OrCAD PSpice A/D Demo

### 4.1.2 Función OR (puerta OR)

La salida de una puerta OR vale 0 sólo si todas y cada una de las variables de entrada son simultáneamente 0. Se puede considerar también, por el principio de dualidad, que la salida de una puerta OR vale 1 si una cualquiera de sus variables de entrada vale 1.

La función OR efectúa la operación de **suma o unión de conjuntos**. La suma o unión de varios conjuntos es otro conjunto formado por todos los elementos de ellos, como se muestra en la Figura 4.30.

La función OR realiza la operación de **suma lógica**, y su símbolo algebraico es  $+$ . Se lee “más” o también “o”.

Desde el punto de vista del **conexionado eléctrico**, se representa la función OR o suma de conjuntos, colocando interruptores en **paralelo** que simbolizan los sumandos o elementos físicos considerados como variables de entrada.

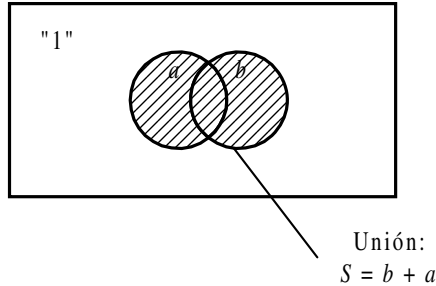


Figura 4.30. El área rayada representa la suma o unión de los conjuntos  $a$  y  $b$

En la Figura 4.31 se aprecia cómo la salida presentará un nivel de tensión  $+V$  cuando al menos uno de los interruptores esté cerrado (haya sido accionado). Es decir, el elemento considerado ha de pertenecer al menos a uno de los conjuntos.

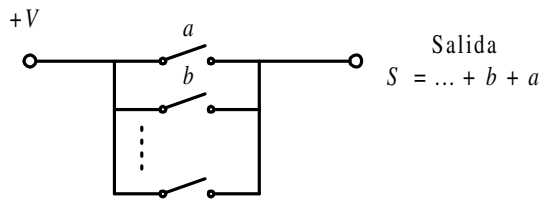


Figura 4.31. Representación eléctrica de la función OR (suma o unión de conjuntos)

La **tabla de verdad** de una puerta OR de dos variables de entrada se representa en la Tabla 4.2, pudiéndose hacer extensivo a  $n$  variables (ver Problema resuelto 4-3).

Tabla 4.2. Tabla de verdad de una puerta OR de dos variables de entrada

$b$	$a$	$S$
0	0	0
0	1	1
1	0	1
1	1	1

El **símbolo** de la puerta OR es el representado en la Figura 4.32.

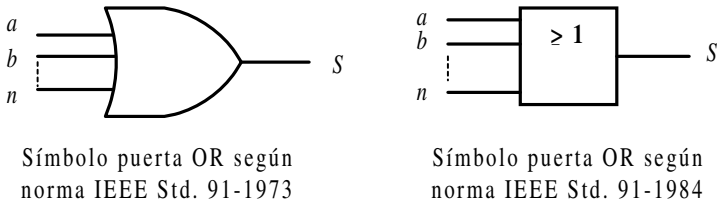


Figura 4.32. Símbolos de la puerta lógica OR

El **cronograma** de la puerta OR de dos entradas, que muestra la relación existente, a lo largo del tiempo, entre sus entradas y su salida, se representa en la Figura 4.33.

La **expresión algebraica** de una puerta OR, considerando  $a, b, c, \dots$  a las variables de entrada y  $S$  a la salida, viene dada por [4.2]:

$$S = f(\dots, c, b, a) = \dots + c + b + a \quad [4.2]$$

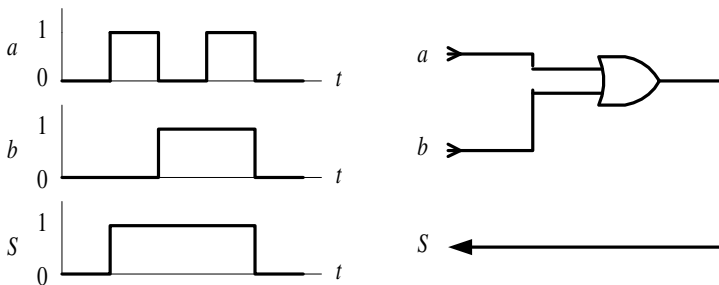


Figura 4.33. Cronograma de una puerta OR de dos entradas



Como aplicación práctica de lo expuesto anteriormente, mediante el **lenguaje VHDL** se define con descripción comportamental algorítmica una puerta OR y se simula su cronograma. En la Figura 4.34 se muestra el código correspondiente.

El Apéndice B contiene un breve manual del lenguaje VHDL que puede servir de referencia en la comprensión de la descripción *hardware*, para aquellos lectores poco familiarizados con este lenguaje.

La ruta y el nombre del fichero que contiene esta descripción VHDL es la que se indica a continuación:

**D:\Ejemplos\Cap04\VBv99\Or\OR\_algo.vpd**



```

-- OR_algo
-----
-- Fichero :      C:\CAEE\INDUSTRI\DIGITAL\VHDL\OR.vhd
-- lib_trabajo:  C:\CAEE\INDUSTRI\DIGITAL\VHDL\WORKLIB
-- Por :         Descripción algorítmica de una puerta OR
-- Fecha :
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY puerta_or IS
    PORT( a, b : in bit;
          S : out bit);
END puerta_or;

-- Descripción comportamental
ARCHITECTURE algorítmica OF puerta_or IS
BEGIN

    PROCESS (a,b)
    BEGIN
        IF (a='1' OR b='1') THEN
            S <= '1';
        ELSE
            S <= '0';
        END IF;
    END PROCESS;

END algorítmica;

```

Figura 4.34. Descripción comportamental algorítmica de una puerta OR mediante VHDL

En la Figura 4.35 se muestra el fichero de estímulos, en lenguaje VHDL, para simular el comportamiento de una puerta OR con descripción comportamental algorítmica.

La ruta y el nombre del fichero que contiene este fichero de estímulos es la que se indica a continuación:

**D:\Ejemplos\Cap04\VBv99\Or\OR\_stim.vhd**

En la Figura 4.36 se muestra el cronograma de la puerta OR obtenido por simulación a partir de la descripción comportamental algorítmica en lenguaje VHDL.

El **circuito integrado comercial** más representativo de puertas OR es el siguiente:

7432: Cuádruple puerta OR de dos entradas.

Mediante el programa de Fairchild de **Data Sheets** que se incluye en el CDRom#2 que acompaña a este libro se pueden buscar las características de los circuitos comerciales de puertas OR, en especial aquél con la referencia anteriormente indicada.

A continuación, se desarrollan ejercicios y **ejemplos de aplicación** de puertas OR utilizando los programas de simulación *Electronics Workbench 5.0 (EWB)* y *OrCAD Demo v9*.

```

OR_stim
-----
-- Fichero : C:\CAEE\INDUSTRI\DIGITAL\VHDL\OR_stim.vhd
-- Lib_trabajo: C:\CAEE\INDUSTRI\DIGITAL\VHDI\WORKLIB
--
-- Por :
-- Fecha :
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY estiaulos IS
END estiaulos;

ARCHITECTURE or_stim OF estiaulos IS
    COMPONENT puerta_or
        PORT (a,b: IN bit;
              S: OUT bit);
    END COMPONENT;
    SIGNAL a,b,S: bit;
BEGIN
    puerta_or1: puerta_or PORT MAP (a,b,S);

    estiaulos: PROCESS
    BEGIN
        a <= '0';
        b <= '0';
        WAIT FOR 50 ns;
        a <= '1';
        WAIT FOR 50 ns;
        a <= '0';
        b <= '1';
        WAIT FOR 50 ns;
        a <= '1';
        WAIT FOR 50 ns;
    END PROCESS;
END or_stim;
Ln 43, Col 1

```

Figura 4.35. Fichero de estímulos para la simulación de una puerta OR, con descripción comportamental, mediante lenguaje VHDL

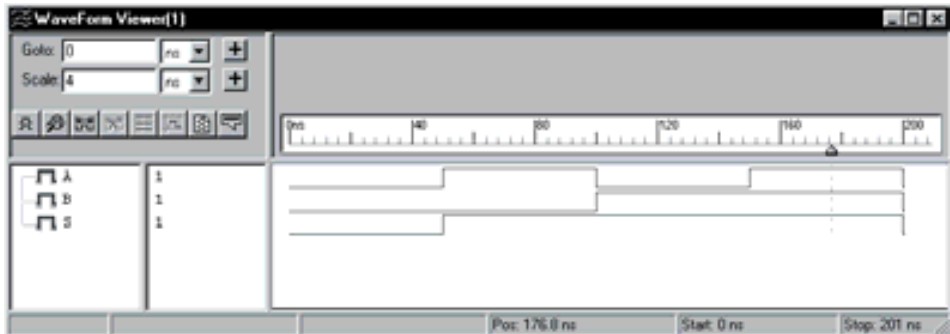


Figura 4.36. Resultado de la simulación, a partir de una descripción en VHDL, de una puerta OR.



### PROBLEMA RESUELTO 4-3



Simular con el programa *Electronics Workbench* el comportamiento de una puerta lógica OR de tres entradas obteniendo su tabla de verdad y su cronograma.

#### Solución:

Mediante el convertidor lógico disponible en el programa de simulación *Electronics Workbench*, se obtiene la tabla de verdad de una puerta OR de tres entradas de dos formas diferentes: introduciendo su expresión algebraica o representando su circuito lógico. Posteriormente, se activará el botón correspondiente según se quiera convertir, de expresión o de circuito lógico a tabla de verdad, como se muestra en la Figura 4.37.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W0\_\_03.ewb**

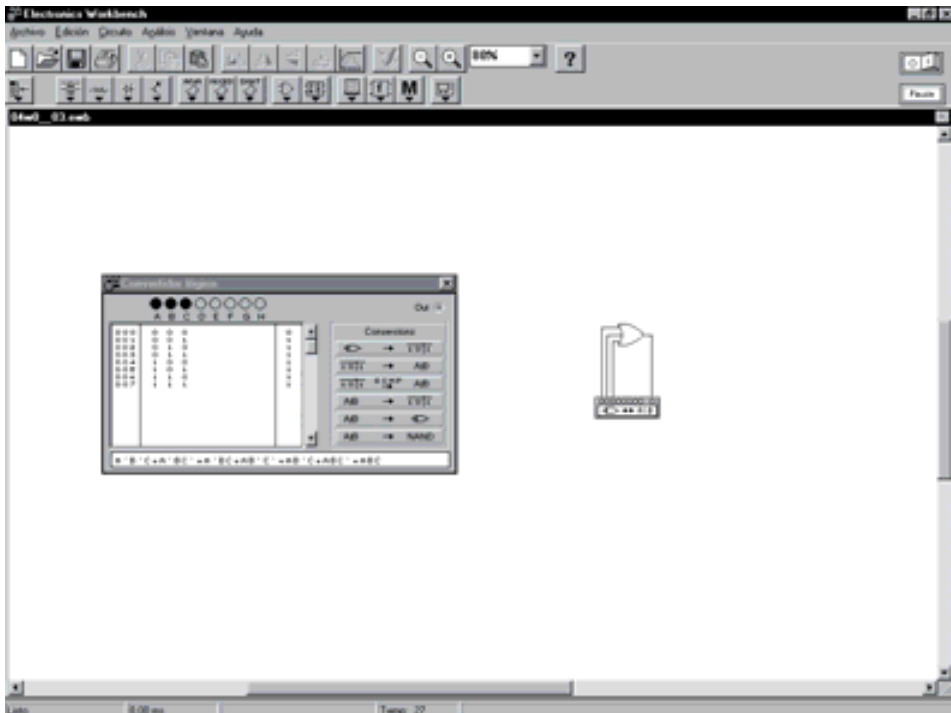


Figura 4.37. Obtención de la tabla de verdad de una puerta OR de tres entradas mediante simulación

Mediante el generador de palabras y el analizador lógico disponible en *Electronics Workbench*, se obtiene el cronograma de una puerta OR de tres entradas, según se muestra en la Figura 4.38.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W1\_\_03.ewb**

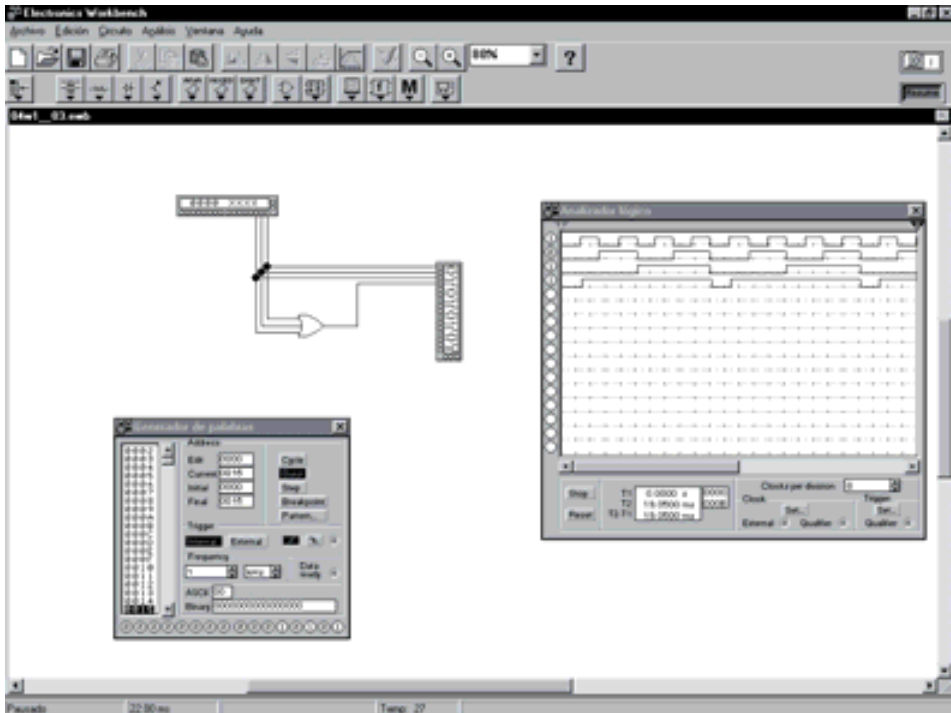


Figura 4.38. Cronograma de una puerta OR de tres entradas mediante simulación



Simular con el programa *OrCAD Demo* el comportamiento de una puerta lógica OR de tres entradas y obtener su cronograma. A partir de éste se puede obtener de forma inmediata su tabla de verdad.

### Solución:

Para simular el comportamiento de este circuito se utiliza el esquema que se muestra en la Figura 4.39. La puerta OR de tres entradas se ha construido a partir del circuito integrado 7432 compuesto de puertas OR de dos entradas.

La ruta y el nombre del fichero que contiene el fichero proyecto de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\OrCAD9\04R0\_03\04R0\_03.opj**

En dicha figura se han representado los valores de las señales de entrada (*a*, *b*, *c*), que deben definirse en la aplicación *PSpice Stimulus Editor*, como se explicó en el primer problema resuelto de este capítulo.

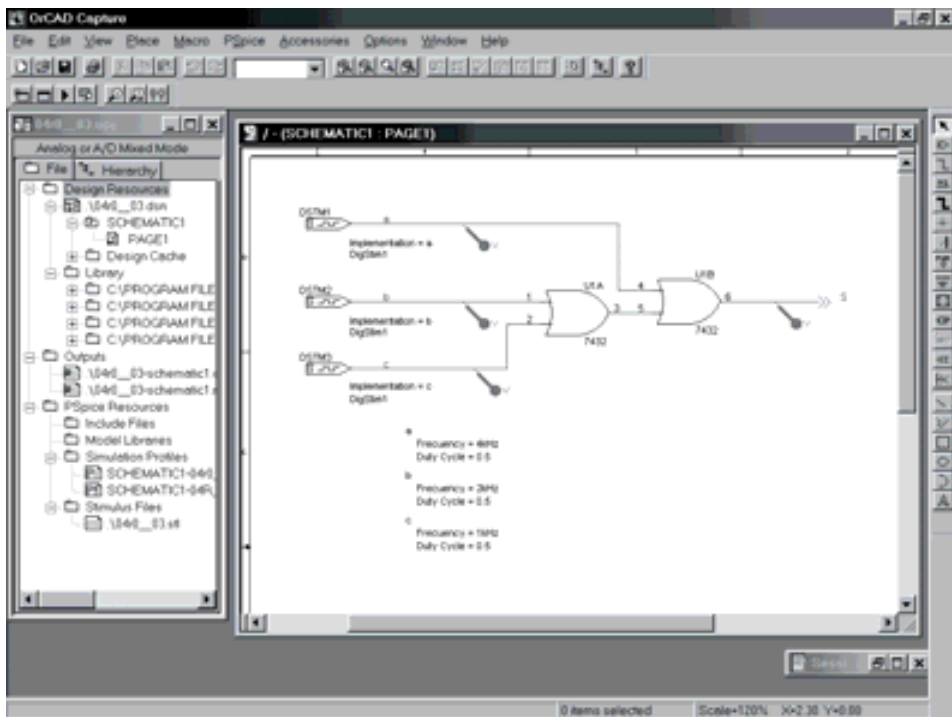


Figura 4.39. Esquema del circuito OR de tres entradas en OrCAD Capture

Para estudiar el comportamiento de este circuito se realiza un análisis transitorio (*Transient...*). Los ajustes realizados en el cuadro de diálogo de dicho comando se muestran en la Figura 4.40. También hay que definir el nombre del fichero de estímulos en la solapa *Stimulus* de este mismo cuadro.

En la Figura 4.41 se muestran los resultados que se obtienen en la aplicación de visualización de los resultados, *OrCAD PSpice A/D Demo*, después de realizar la simulación. En dicha aplicación se ha activado el cursor, que permite ver, al lado del nombre de las señales, el valor que toma cada una de ellas en el instante en el cual se sitúa el mismo. Situando el cursor en cada uno de los puntos de interés, por ejemplo en cada uno de los sucesivos valores que van tomando las señales, se puede obtener la tabla de verdad del circuito.

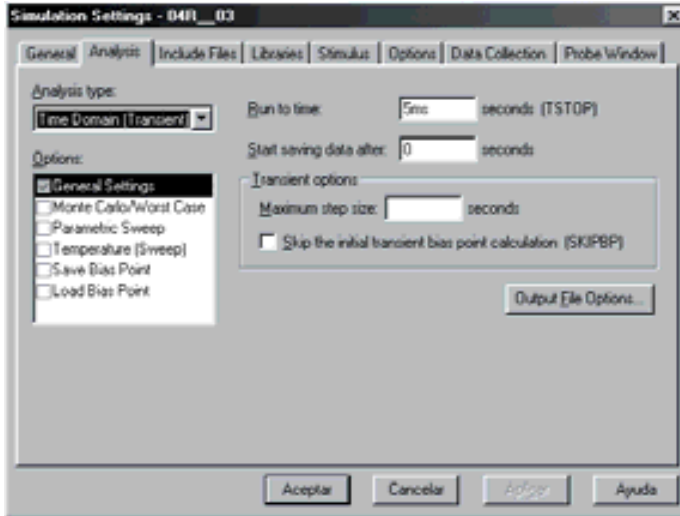


Figura 4.40. Configuración del análisis transitorio de la puerta OR de tres entradas

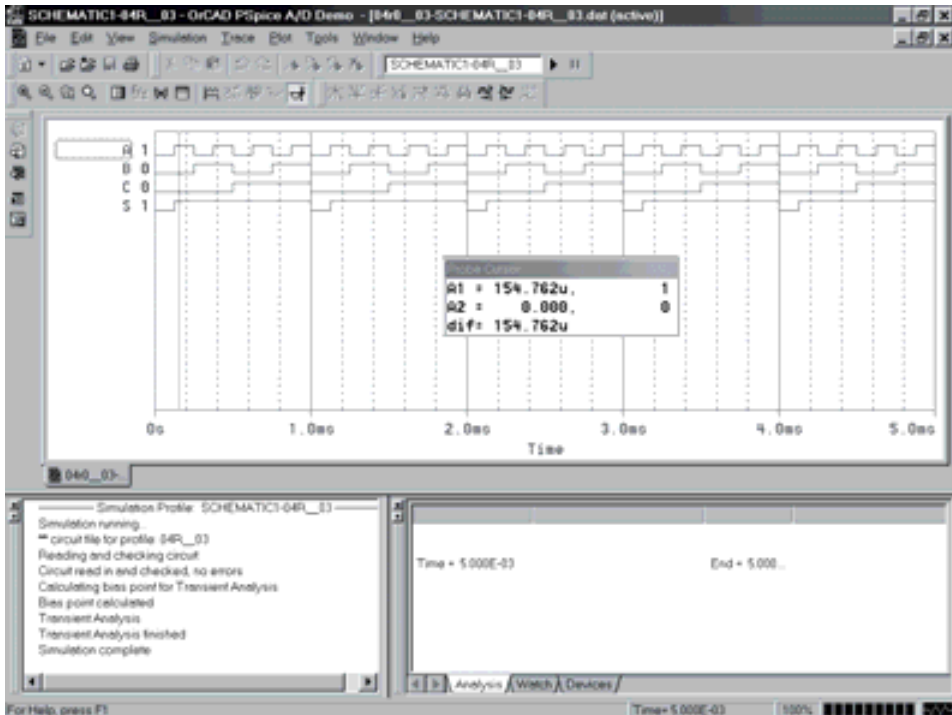


Figura 4.41. Resultados en OrCAD PSpice A/D Demo para la OR de tres entradas

## PROBLEMA RESUELTO 4-4



Diseñar un subcircuito, que forma parte de una alarma, que active una sirena  $S$  cuando cualquiera de los sensores situados en tres ventanas  $a$ ,  $b$ ,  $c$  y una puerta  $d$  detecten una intrusión.

### Solución:

Este problema corresponde a una aplicación típica de las puertas OR, según se muestra en la Figura 4.42. Se podría haber hecho también con una sola puerta OR de cuatro entradas.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W0\_\_04.ewb**

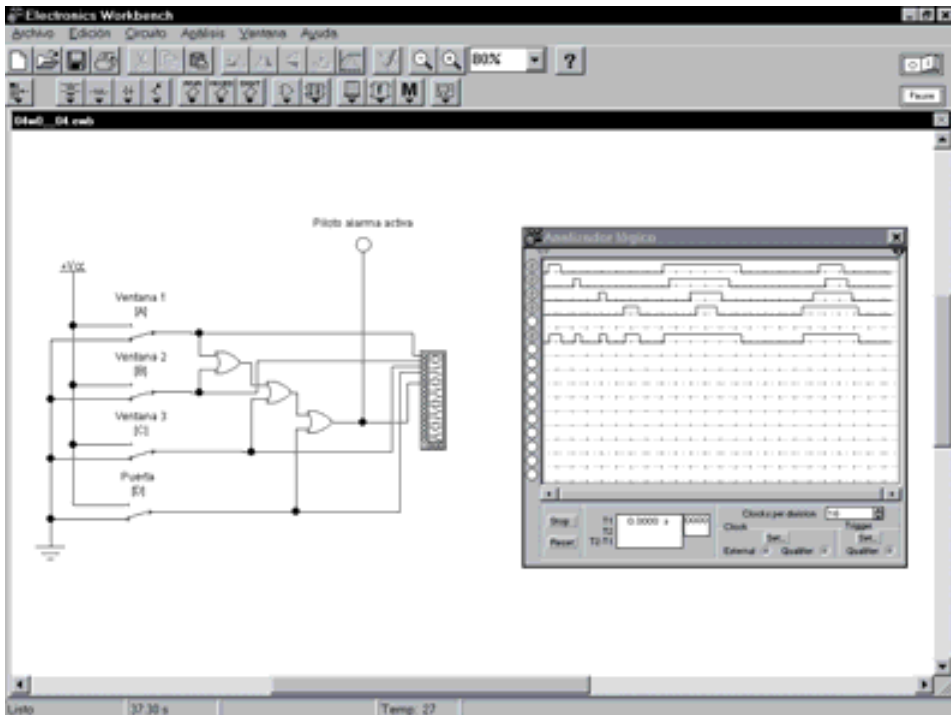


Figura 4.42. Aplicación de puerta OR: subcircuito de alarma

## PROBLEMAS PROPUESTOS

- 4-3) Diseñar un circuito eléctrico que realice la función OR con tres interruptores. Comprobarlo mediante el simulador.

### 4.1.3 Función NOT (puerta lógica inversora)

Una puerta lógica inversora sólo tiene una entrada. La salida es el complemento de la entrada, es decir, si la entrada vale 0 la salida vale 1 y si la entrada vale 1 la salida vale 0.

La función NOT efectúa la operación de **inversión o complemento de conjuntos**. Un conjunto es inverso, negado o complementario de otro conjunto, cuando está formado por los elementos del conjunto universal no contenidos en aquél, como se muestra en la Figura 4.43.

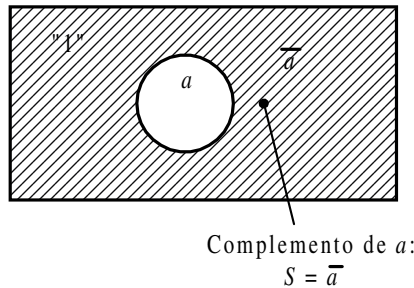


Figura 4.43. El área rayada representa al conjunto  $\bar{a}$  (complemento del conjunto  $a$ )

Desde el punto de vista del **conexionado eléctrico**, se representa la función NOT o conjunto inverso mediante un interruptor normalmente cerrado. En la Figura 4.44 se representa al conjunto  $a$  y su inverso o complementario  $\bar{a}$ .

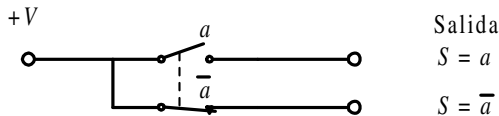


Figura 4.44. Representación eléctrica de la función NOT (inversión o complemento de un conjunto)

En la Figura 4.44 se aprecia cómo la salida sólo tendrá nivel de tensión +V cuando **no** se cambie el estado del interruptor  $\bar{a}$ , es decir, el elemento considerado no debe pertenecer al conjunto  $a$ .

La **tabla de verdad** de una puerta NOT se representa en la Tabla 4.3.

Tabla 4.3. Tabla de verdad de una puerta NOT

$a$	$S$
0	1
1	0

El **símbolo** de la puerta NOT es el representado en la Figura 4.45.

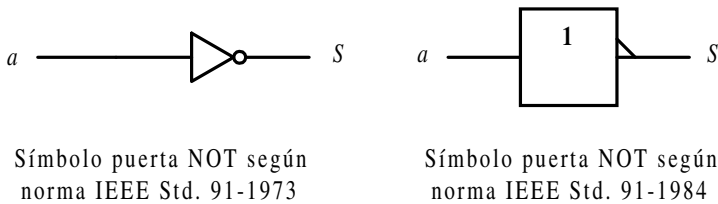


Figura 4.45. Símbolos de la puerta lógica NOT

El **cronograma** de la puerta NOT, que muestra la relación existente, a lo largo del tiempo, entre su entrada y su salida, se representa en la Figura 4.46.

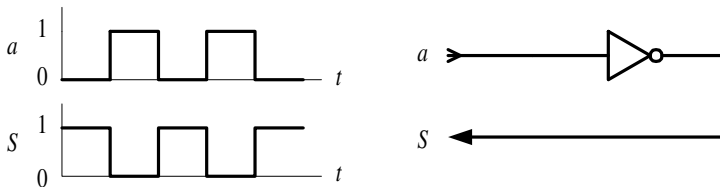


Figura 4.46. Cronograma que relaciona las señales de entrada y de salida de una puerta NOT

La **expresión algebraica** de una puerta NOT, considerando  $a$  a la variable de entrada y  $S$  a la salida, es la siguiente:

$$S = f(a) = \bar{a} \quad [4.3]$$



Como aplicación práctica de lo expuesto anteriormente, mediante el **lenguaje VHDL** se define con descripción comportamental de flujo de datos una puerta NOT (Figura 4.47) y se simula su cronograma.

El Apéndice B contiene un breve manual del lenguaje VHDL que puede servir de referencia en la comprensión de la descripción *hardware*, para aquellos lectores poco familiarizados con este lenguaje.

La ruta y el nombre del fichero que contiene esta descripción VHDL es la que se indica a continuación:

**D:\Ejemplos\Cap04\VBv99\Not\NOT\_flujo.vpd**



```
not_flujo
-----
-- Fichero : C:\CAEE\INDUSTRI\DIGITAL\VHDL\NOT\not_flujo.vhd
-- Lib_trabajo: C:\CAEE\INDUSTRI\DIGITAL\VHDL\WORKLIB
-- Por : Descripción flujo de datos de una puerta not
-- Fecha :
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY puerta_not IS
    PORT( a : in bit;
          S : out bit);
END puerta_not;

-- Descripción comportamental

ARCHITECTURE flujo_datos OF puerta_not IS
--    SIGNAL ex: bit;
BEGIN
    S<= NOT a;
END flujo_datos;
```

*Figura 4.47. Descripción comportamental de flujo de datos de una puerta NOT mediante VHDL*

En la Figura 4.48 se muestra el fichero de estímulos, en lenguaje VHDL, para simular el comportamiento de una puerta NOT con descripción comportamental de flujo de datos.

La ruta y el nombre del fichero que contiene este fichero de estímulos es la que se indica a continuación:

**D:\Ejemplos\Cap04\VBv99\Not\NOT\_stim.vhd**

En la Figura 4.49 se muestra el cronograma de la puerta NOT obtenido por simulación a partir de la descripción comportamental de flujo de datos en lenguaje VHDL.



```

not_stim
-----
-- Fichero :      C:\CAEE\INDUSTRI\DIGITAL\VHDL\NOT\not_stim.vhd
-- Lib_trabajo:  C:\CAEE\INDUSTRI\DIGITAL\VHDL\WORKLIB
-- Por :         Estimulos de prueba para una puerta not
-- Fecha :
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY estmulos IS
END estmulos;

ARCHITECTURE not_stim OF estmulos IS

    COMPONENT puerta_not
        PORT (a: IN bit;
              S: OUT bit);
    END COMPONENT;

    SIGNAL a,S: bit;

BEGIN

    puerta_not1: puerta_not PORT MAP (a,S);

    estmulos: PROCESS
    BEGIN
        a <= '0';
        WAIT FOR 50 ns;
        a <= '1';
        WAIT FOR 50 ns;
    END PROCESS;

END not_stim;

```

"C:\CAee\Industri\Digita\VHDL\NOT\not\_stim.vhd" saved      Ln 37, Col 8

Figura 4.48. Fichero de estímulos para la simulación de una puerta NOT, con descripción comportamental, mediante lenguaje VHDL

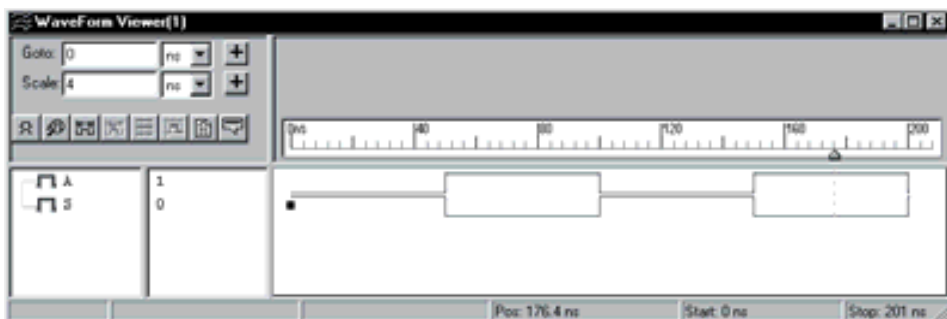


Figura 4.49. Resultado de la simulación, a partir de una descripción en VHDL, de una puerta NOT.

Los **circuitos integrados comerciales** más representativos de puertas NOT son:

7404: Inversor séxtuple.

7405/6/16: Inversor séxtuple con salidas colector abierto.

Mediante el programa de Fairchild de **Data Sheets** que se incluye en el CDROM#2 que acompaña a este libro se pueden buscar las características de los circuitos comerciales de puertas NOT, en especial aquéllos con las referencias anteriormente indicadas.

A continuación, se desarrollan ejercicios y **ejemplos de aplicación** de puertas NOT utilizando los programas de simulación *Electronics Workbench 5.0 (EWB)* y *OrCAD Demo v9*.

### PROBLEMA RESUELTO 4-5



Utilizando el programa *Electronics Workbench* simular el comportamiento de una puerta lógica NOT, obteniendo su tabla de verdad y su cronograma.

#### Solución:

Mediante el convertidor lógico disponible en el simulador *Electronics Workbench*, se obtiene la tabla de verdad de una puerta NOT de dos formas diferentes: bien introduciendo su expresión algebraica o bien representando su circuito lógico. Posteriormente se activará el botón correspondiente según se quiera convertir, de expresión o de circuito lógico a tabla de verdad, como se muestra en la Figura 4.50.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W0\_\_05.ewb**

Mediante un generador de onda cuadrada y el analizador lógico disponible en el simulador *Electronics Workbench*, se obtiene el cronograma de una puerta NOT, según se muestra en la Figura 4.51.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W1\_\_05.ewb**

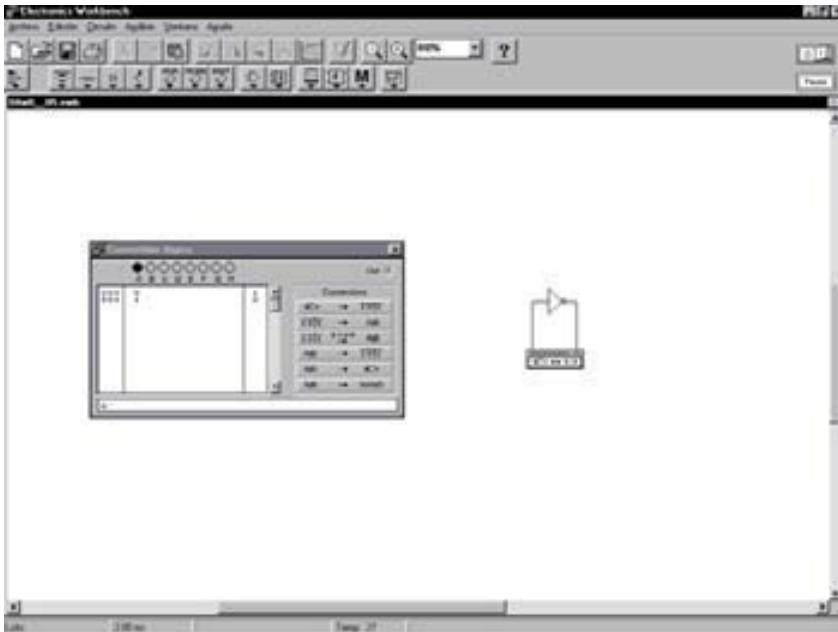


Figura 4.50. Obtención de la tabla de verdad de una puerta NOT mediante EWB

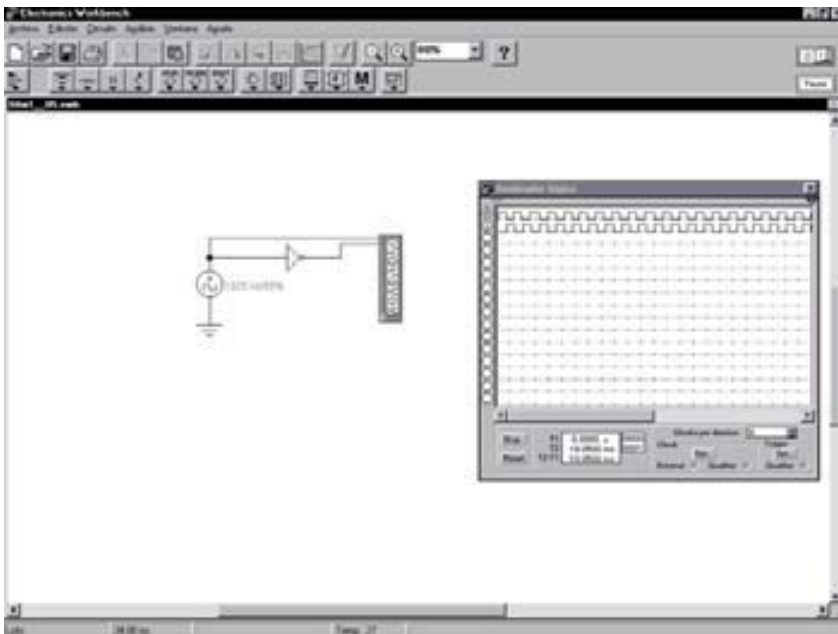


Figura 4.51. Obtención del cronograma de una puerta NOT mediante EWB



Simular con el programa *OrCAD Demo* el comportamiento de una puerta lógica NOT y obtener su cronograma. A partir de éste se puede obtener de forma inmediata su tabla de verdad.

### Solución:

Para simular el comportamiento de este circuito se utiliza el esquema que se muestra en la Figura 4.52.

La ruta y el nombre del fichero que contiene el fichero proyecto de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\OrCAD9\04R0\_\_05\04R0\_\_05.opj**

En la Figura 4.52 se han representado los valores de la señal de entrada *a*, que deben definirse en la aplicación *PSpice Stimulus Editor*, como se explicó en el primer problema resuelto de este capítulo.

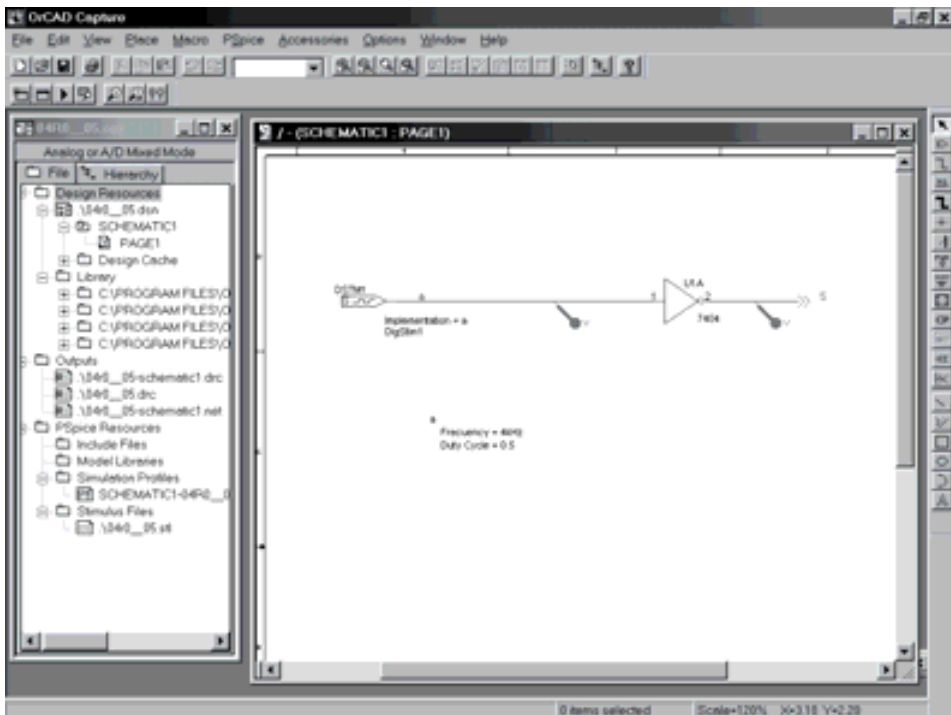


Figura 4.52. Esquema del circuito NOT utilizando la aplicación *OrCAD Capture*

Para estudiar el comportamiento de este circuito se realiza un análisis transitorio (*Transient...*). Se ha definido en el menú *PSpice/Edit Simulation Settings/Analysis* el valor de **5ms** para *Run to time*. También está definido en

**PSpice/Edit Simulation Settings/Stimulus** el fichero **04R0\_05.st1** de estímulos.

En la Figura 4.53 se muestran los resultados que se obtienen en la aplicación de visualización de los resultados, *OrCAD PSpice A/D Demo*, después de realizar la simulación. En dicha aplicación se ha activado el cursor, que permite ver, al lado del nombre de las señales, el valor que toma cada una de ellas en el instante en el cual se sitúa el mismo.

Situando el cursor en cada uno de los puntos de interés, por ejemplo en cada uno de los sucesivos valores que van tomando las señales se puede obtener la tabla de verdad del circuito.

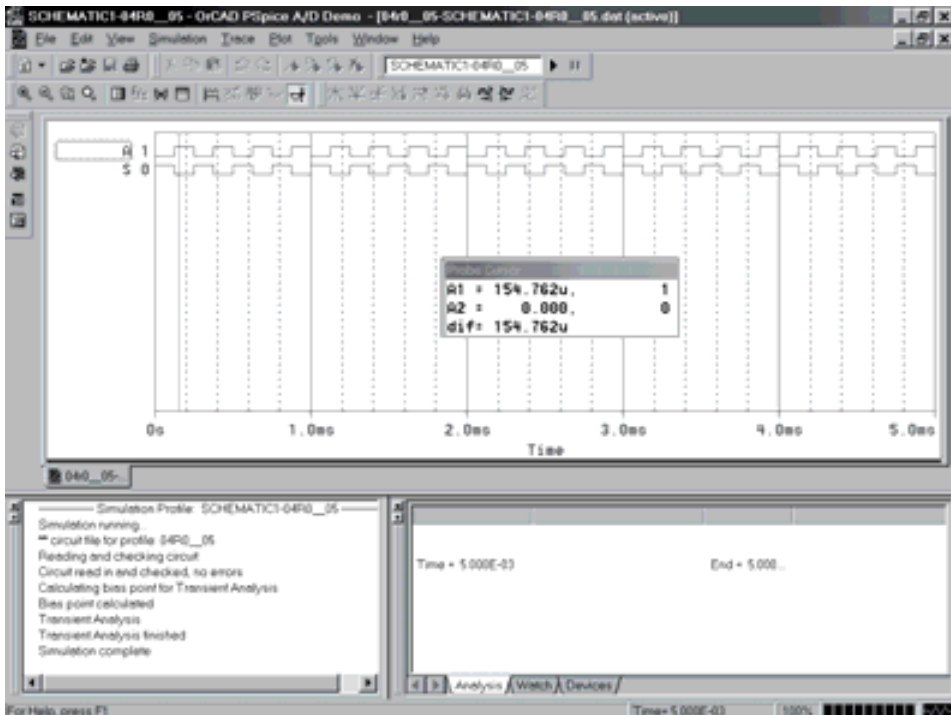


Figura 4.53. Resultados en OrCAD PSpice A/D Demo para el circuito NOT

## PROBLEMA RESUELTO 4-6



Utilizando el programa de simulación *Electronics Workbench*, diseñar un circuito que realice el complemento a uno de un número binario de ocho bits.

**Solución:**

Este problema corresponde a una aplicación típica de las puertas NOT. Se comprueba el resultado para el caso particular del dato de entrada en hexadecimal igual a A3, según se muestra en la Figura 4.54.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W0\_\_06.ewb**

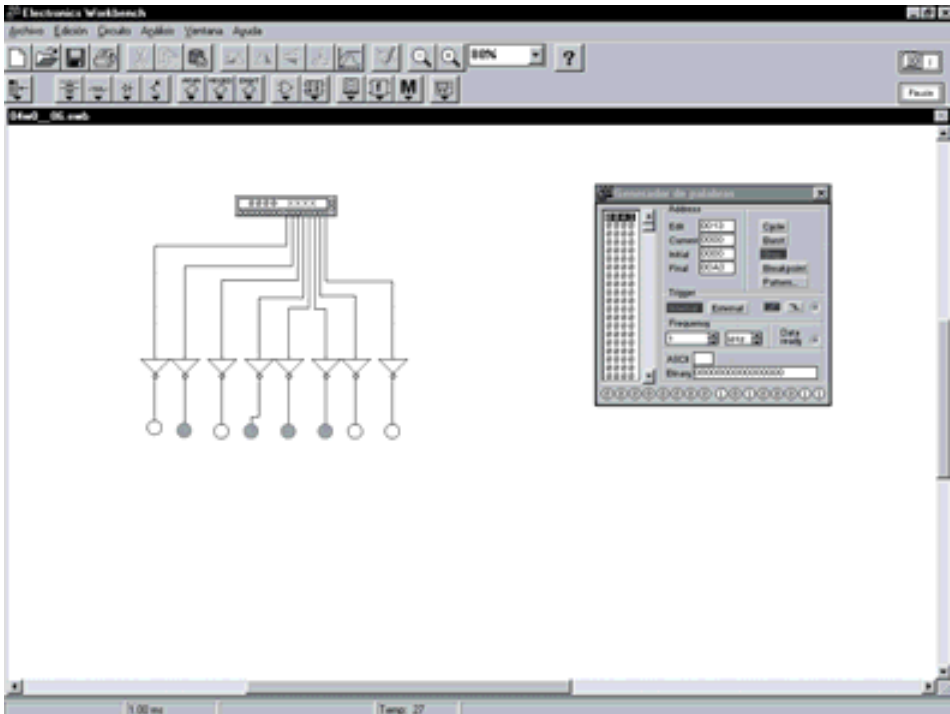


Figura 4.54. Aplicación de puerta NOT: circuito que realiza el complemento a uno de un número binario de ocho bits

**PROBLEMAS PROPUESTOS**

- 4-4) Utilizando el programa de simulación *OrCAD Demo*, diseñar un circuito que realice el complemento a uno de un número binario de ocho bits.
- 4-5) Diseñar un circuito eléctrico, con relés, que realice la función NOT. Comprobarlo mediante los programas de simulación.

### 4.1.4 Función NAND (puerta NAND)

La puerta NAND es el complemento de la puerta AND.

La salida de una puerta NAND vale 0 sólo si todas y cada una de las variables de entrada son simultáneamente 1. Se puede considerar también, por el principio de dualidad, que la salida de una puerta NAND vale 1 si una cualquiera de sus variables de entrada vale 0.

La operación NAND produce el resultado inverso o complementado del producto de varios conjuntos. La **operación complemento del producto de varios conjuntos** es otro conjunto formado por los elementos no comunes a ellos, como se muestra en la Figura 4.55.

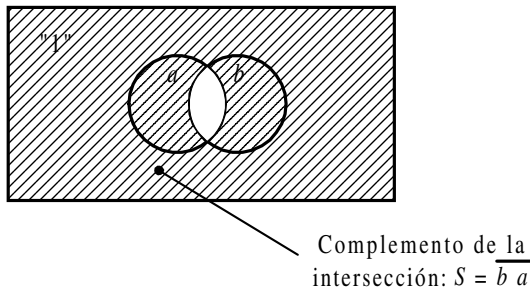


Figura 4.55. El área rayada representa el complemento del producto o intersección de los conjuntos  $a$  y  $b$

La función NAND realiza la operación de **complemento del producto** lógico de varios conjuntos  $a_1, a_2, \dots$ , siendo su símbolo algebraico  $\overline{a_1 a_2 \dots}$ . Se lee “inverso del producto  $a_1$  por  $a_2$  por ...”.

Desde el punto de vista del **conexionado eléctrico**, se representan el complemento del producto de conjuntos, colocando los interruptores en serie y agregando un elemento que complemente el resultado, como se muestra en la Figura 4.56.

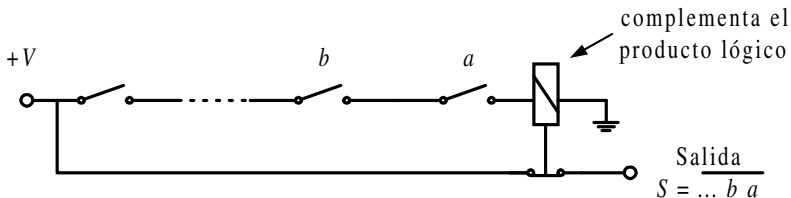


Figura 4.56. Representación eléctrica de la función NAND; complemento del producto o intersección de conjuntos

Otro circuito, resultado de aplicar el teorema de De Morgan a la función NAND, es decir,  $f = \overline{\dots b \cdot a} = \dots + \overline{b} + \overline{a}$ , es el mostrado en la Figura 4.57.

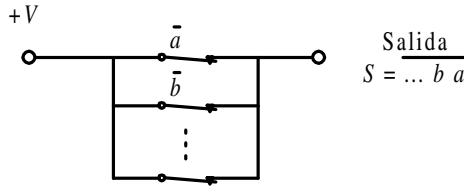


Figura 4.57. Otra representación eléctrica de la función NAND

En dichas figuras se aprecia cómo la salida siempre tendrá nivel de tensión +V excepto cuando todos los interruptores simultáneamente cambien de estado, es decir, el elemento considerado no ha de pertenecer a la vez a todos los conjuntos.

La **tabla de verdad** de una puerta NAND de dos variables de entrada se representa en la Tabla 4.4, pudiéndose hacer extensivo a  $n$  variables (Problema resuelto 4-7).

Tabla 4.4. Tabla de verdad de una puerta NAND de dos variables de entrada

$b$	$a$	$S$
0	0	1
0	1	1
1	0	1
1	1	0

El **símbolo** de la puerta NAND es el representado en la Figura 4.58.

El **cronograma** de la puerta NAND de dos entradas, que muestra la relación existente, a lo largo del tiempo, entre sus entradas y su salida, se representa en la Figura 4.59.

La **expresión algebraica** de una puerta NAND, considerando  $a, b, c, \dots$  a las variables de entrada y  $S$  a la salida, es:

$$S = f(\dots, c, b, a) = \overline{\dots cba} \tag{4.4}$$



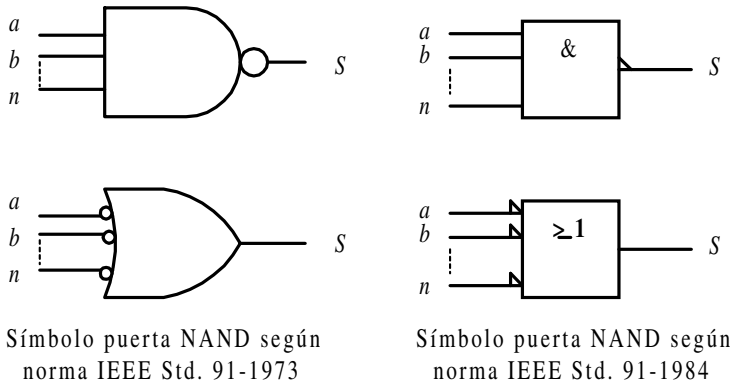


Figura 4.58. Símbolos de la puerta lógica NAND

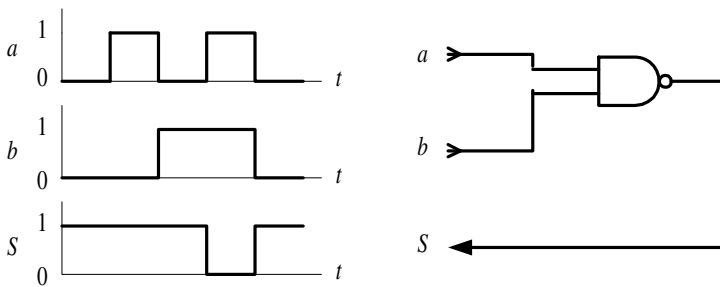


Figura 4.59. Cronograma que relaciona las señales de entradas y salida de una puerta NAND de dos entradas



Como aplicación práctica de lo expuesto anteriormente, mediante el lenguaje VHDL se define con descripción estructural una puerta NAND y se simula su cronograma. Para ello se parte del fichero de la Figura 4.60.

El Apéndice B contiene un breve manual del lenguaje VHDL que puede servir de referencia en la comprensión de la descripción *hardware*, para aquellos lectores poco familiarizados con este lenguaje.

La ruta y el nombre del fichero que contiene esta descripción VHDL es la que se indica a continuación:

**D:\Ejemplos\Cap04\VBv99\Nand\NAND\_estr.vpd**

En la Figura 4.61 se muestra el fichero de estímulos, en lenguaje VHDL, para simular el comportamiento de una puerta NAND con descripción estructural.

En la Figura 4.62 se muestra el cronograma de la puerta NAND obtenido por simulación a partir de la descripción estructural en lenguaje VHDL.

```

Nand_estr
-----
-- Fichero : C:\CAEE\INDUSTRI\DIGITAL\VHDL\Nand\Nand_estr\Nand_estr.vhd
-- Lib_trabajo: C:\CAEE\INDUSTRI\DIGITAL\VHDL\WORKLIB
-- Descripción estructural de una puerta Nand
-- Por :
-- Fecha :
-----
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Definición de entidades y arquitectura de cada componente
-- (en este caso sólo uno)

ENTITY puerta_nand IS
    PORT( ap, bp : in bit;
          Sp : out bit);
END puerta_nand;

ARCHITECTURE rtl OF puerta_nand IS
BEGIN
    Sp <= ap NAND bp;
END rtl;

-- Definición de la entidad y arquitectura del conjunto o circuito
-- (netlist)

ENTITY circuito_nand IS
PORT (a,b: IN bit;
      S: OUT bit);
END circuito_nand;

ARCHITECTURE estructural OF circuito_nand IS
    COMPONENT puerta_nand IS
        PORT (ap,bp: IN bit; Sp: OUT bit);
    END COMPONENT puerta_nand;

BEGIN
    u0: puerta_nand PORT MAP (ap=>a,bp=>b,Sp=>S);
END estructural;
-----
Ln 22, Col 25

```

Figura 4.60. Descripción estructural de una puerta NAND mediante VHDL

Los **circuitos comerciales** más representativos de puertas NAND son:

- 7400: Cuádruple puerta NAND de dos entradas.
- 7401/03/26/38/39: Cuádruple puerta NAND de dos entradas con salidas colector abierto.
- 7410: Triple puerta NAND de tres entradas.
- 7412: Triple puerta NAND de tres entradas con salidas colector abierto.
- 7420: Doble puerta NAND de cuatro entradas.
- 7430: Puerta NAND de ocho entradas.
- 74133: Puerta NAND de trece entradas.

```

Nand_stim_estr
-----
-- Fichero : C:\CAEE\INDUSTRI\DIGITAL\VHDL\Nand\Nand_stia_estr.vhd
-- Lib_trabajo: C:\CAEE\INDUSTRI\DIGITAL\VHDL\WORKLIB
--          Estioulos para la simulación de una puerta Nand
--          con descripción estructural.
-- Por :
-- Fecha :
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY estiaulos IS
END estiaulos;

ARCHITECTURE nand_stia OF estiaulos IS

    COMPONENT circuito_nand
        PORT (a,b: IN bit;
              S: OUT bit);
    END COMPONENT;

    SIGNAL a,b,S: bit;

BEGIN

    circuito_nand1: circuito_nand PORT MAP (a,b,S);

    estiaulos: PROCESS
    BEGIN
        a <= '0';
        b <= '0';
        WAIT FOR 50 ns;
        a <= '1';
        WAIT FOR 50 ns;
        a <= '0';
        b <= '1';
        WAIT FOR 50 ns;
        a <= '1';
        WAIT FOR 50 ns;
    END PROCESS;

END nand_stia;

"C:\Cae\Industr\Digita\VHDL\Nand\Nand_estru\Nand_stim_estr.vhd" saved |Ln 9, Col 1

```

Figura 4.61. Fichero de estímulos para la simulación de una puerta NAND, con descripción estructural, mediante lenguaje VHDL

Mediante el programa de Fairchild de **Data Sheets** que se incluye en el CDROM#2 que acompaña a este libro se pueden buscar las características de los circuitos comerciales de puertas NAND, en especial aquéllos con las referencias anteriormente indicadas.

A continuación, se desarrollan ejercicios y **ejemplos de aplicación** de puertas NAND utilizando los programas de simulación *Electronics Workbench 5.0 (EWB)* y *OrCAD Demo v9*.

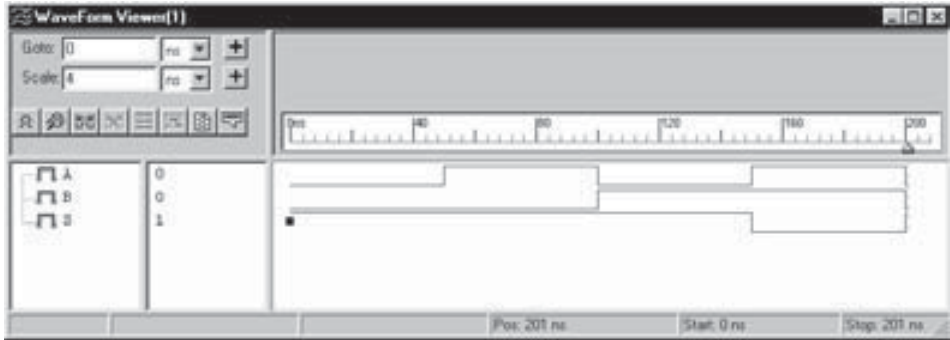


Figura 4.62. Resultado de la simulación, a partir de una descripción en VHDL, de una puerta NAND

## PROBLEMA RESUELTO 4-7



Simular con el programa *Electronics Workbench* el comportamiento de una puerta lógica NAND de tres entradas, obteniendo su tabla de verdad y su cronograma.

### Solución:

Mediante el convertidor lógico disponible en el programa de simulación *Electronics Workbench*, se obtiene la tabla de verdad de una puerta NAND de tres entradas de dos formas diferentes: introduciendo su expresión algebraica o representando su circuito lógico. Posteriormente se activará el botón correspondiente según se quiera convertir, de expresión o de circuito lógico a tabla de verdad, como se muestra en la Figura 4.63.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W0\_\_07.ewb**

Mediante el generador de palabras y el analizador lógico disponible en el programa de simulación *Electronics Workbench*, se obtiene el cronograma de una puerta NAND de tres entradas, según se muestra en la Figura 4.64.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W1\_\_07.ewb**

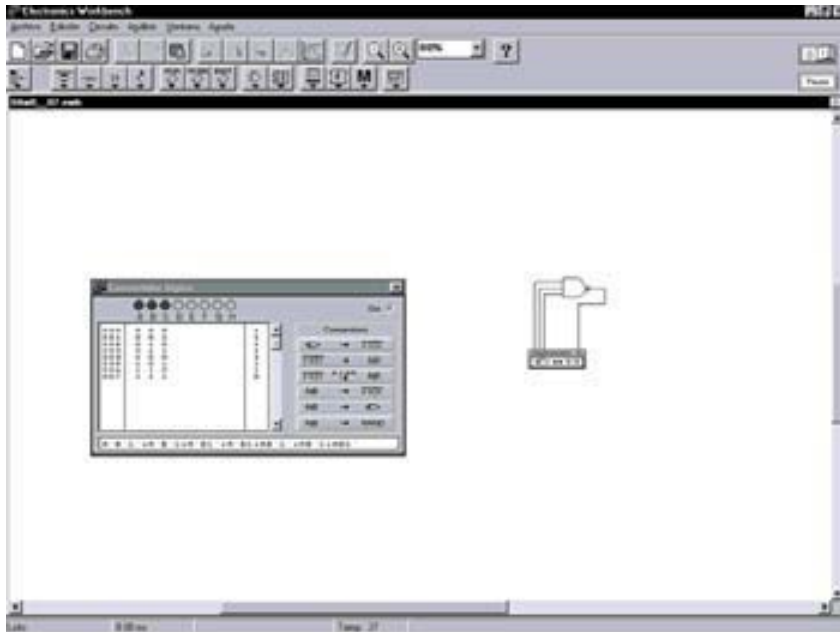


Figura 4.63. Tabla de verdad de una puerta NAND de tres entradas en EWB

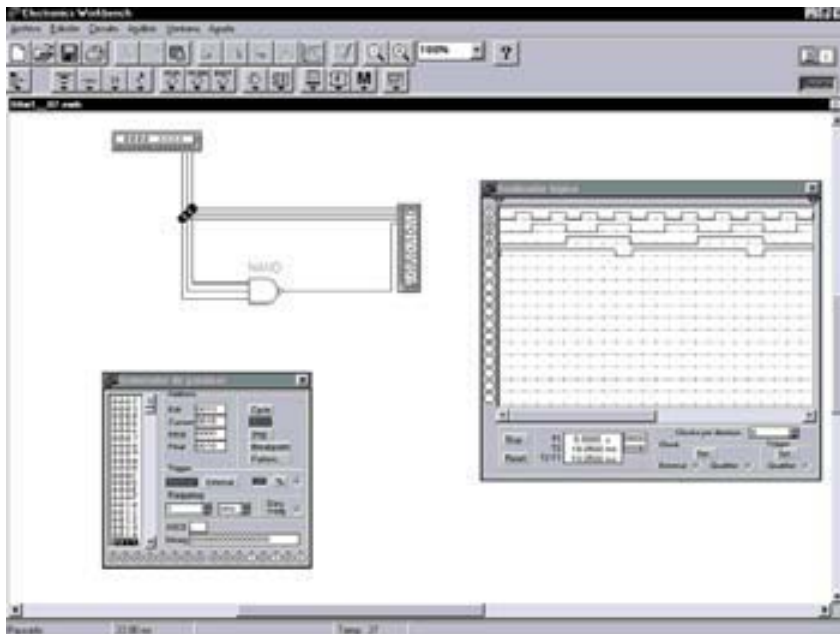


Figura 4.64. Cronograma de una puerta NAND de tres entradas en EWB



Simular con el programa *OrCAD Demo* el comportamiento de una puerta lógica NAND de tres entradas y obtener su cronograma. A partir de éste se puede obtener de forma inmediata su tabla de verdad.

### Solución:

Para simular con *OrCAD Demo* el comportamiento de este circuito se utiliza el esquema que se muestra en la Figura 4.65.

La ruta y el nombre del fichero que contiene el fichero proyecto de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\OrCAD9\04R0\_\_07\04R0\_\_07.opj**

En dicha figura se han representado los valores de las señales de entrada ( $a$ ,  $b$ ,  $c$ ) que deben definirse en la aplicación *Stimulus Editor*, como se explicó en el primer problema resuelto de este capítulo.

Para estudiar el comportamiento de este circuito se realiza un análisis transitorio (*Transient...*). Se ha definido en el menú *PSpice/Edit Simulation Settings/Analysis* el valor de **2ms** en *Run to time*. En *PSpice/Edit Simulación Settings/Stimulus* el fichero **04R0\_\_07.st1** contiene los estímulos.

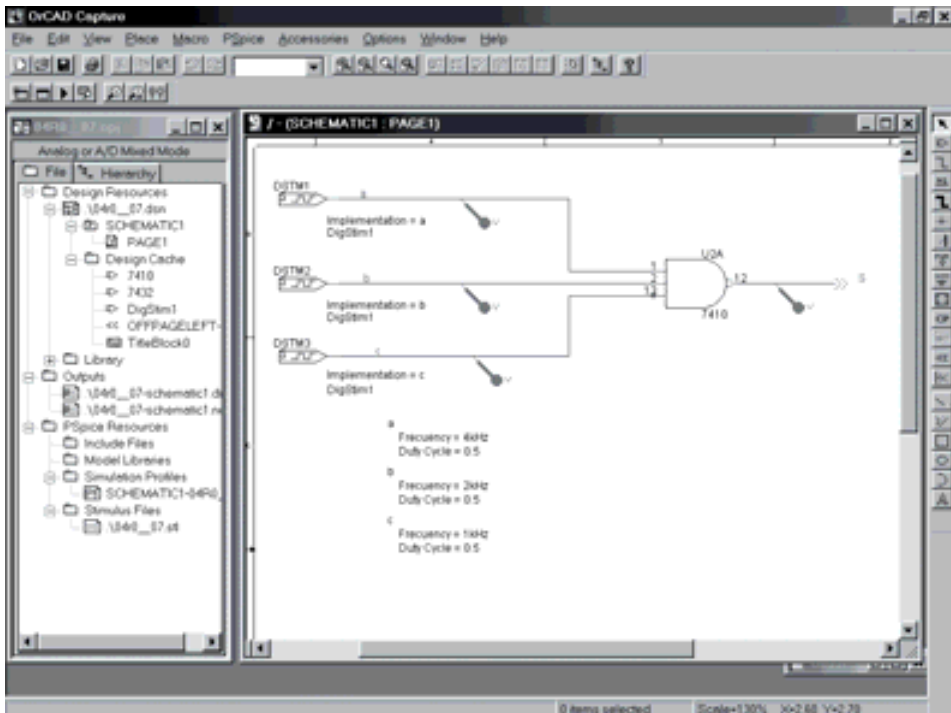


Figura 4.65. Esquema del circuito NAND de tres entradas en OrCAD Capture

En la Figura 4.66 se muestran los resultados que se obtienen en la aplicación de visualización de *OrCAD PSpice A/D Demo*, después de realizar la simulación. En dicha aplicación se ha activado el cursor, que permite ver, el valor que toma cada una de las señales en el instante en el cual se sitúa el mismo. Situando el cursor en cada uno de los puntos de interés, por ejemplo en cada uno de los sucesivos valores que van tomando las señales se puede obtener la tabla de verdad del circuito.

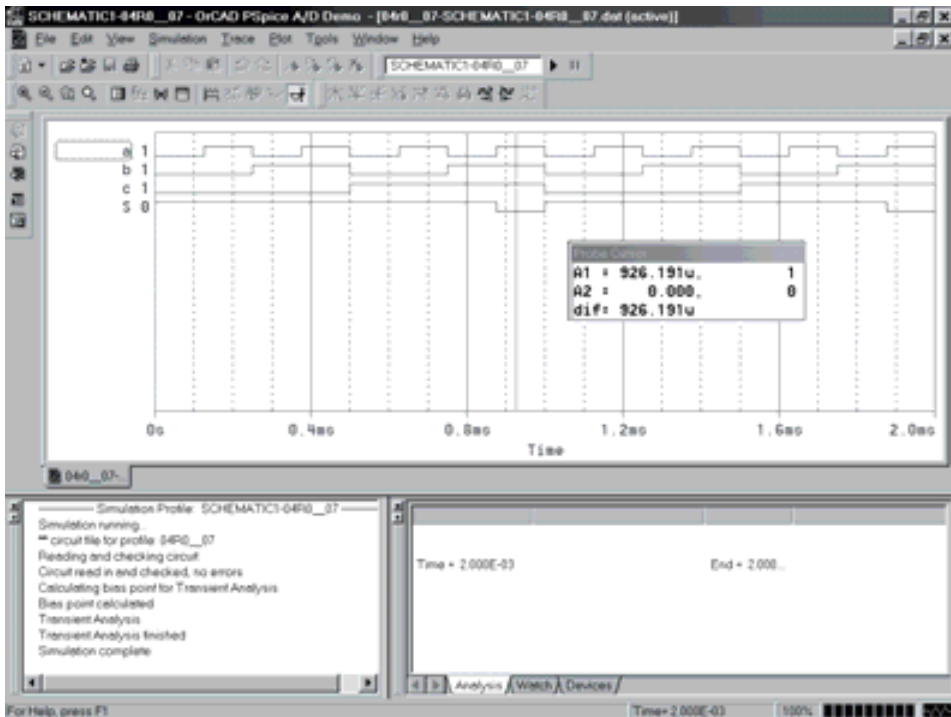


Figura 4.66. Resultados obtenidos en *OrCAD PSpice A/D Demo* para el circuito NAND

## PROBLEMA RESUELTO 4-8



Se trata de diseñar, utilizando el programa de simulación *Electronics Workbench*, un circuito que detecta el nivel, por debajo de un mínimo, en dos depósitos.

En un proceso de fabricación, que requiere la mezcla de dos líquidos almacenados en dos depósitos, se quiere diseñar un circuito que detecte cuándo alguno de los depósitos se encuentra por debajo del 20 % de su capacidad, visualizándose en un LED de color rojo esta situación.

Los sensores de nivel de líquidos entregan un 1 cuando el depósito está por encima del 20% y un 0 en caso contrario.

### Solución:

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W0\_\_08.ewb**

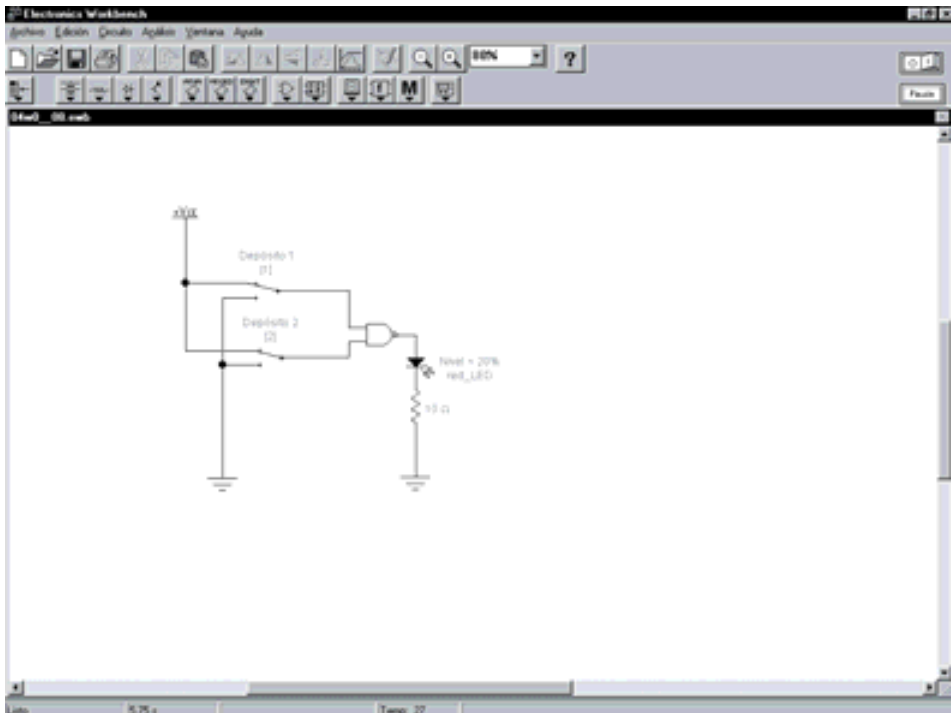


Figura 4.67. Aplicación de puerta NAND: circuito que detecta niveles por debajo de un mínimo en dos depósitos

### PROBLEMAS PROPUESTOS

- 4-6) Diseñar, utilizando el programa de simulación *Electronics Workbench*, un circuito que detecta el nivel, por debajo de un mínimo, en dos depósitos.
- 4-7) Modificar el problema anterior para que la señalización se realice mediante un LED verde que indique la situación en la que los dos depósitos se encuentran por encima del 20%. Comprobarlo utilizando las herramientas de simulación.



- 4-8) Representar el cronograma de salida de la puerta NAND (que opera mediante una puerta Negativa-OR) de tres entradas a la que se les aplica las señales de la Figura 4.68. Indicar cómo se modifica el cronograma de salida si se invierte la entrada  $a$ . Comprobarlo utilizando las herramientas de simulación.

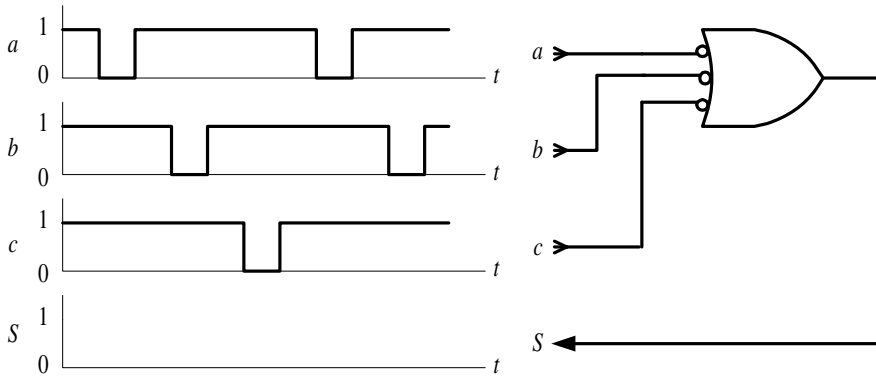


Figura 4.68. Problema propuesto con puerta NAND, representada mediante una puerta Negativa-OR

### 4.1.5 Función NOR (puerta NOR)

La puerta NOR es el complemento de la puerta OR.

La salida de una puerta NOR vale 1 sólo si todas y cada una de las variables de entrada son simultáneamente 0. Se puede considerar también, por el principio de dualidad, que la salida de una puerta NOR vale 0 si una cualquiera de sus variables de entrada vale 1.

La operación NOR produce el resultado inverso o complementado de la unión de varios conjuntos. La **operación complemento de la unión de varios conjuntos** es otro conjunto formado por los elementos que no pertenecen a ninguno de ellos, como se muestra en la Figura 4.69.

La función NOR realiza la operación de **complemento de la suma** lógica de varios conjuntos  $a_1, a_2, \dots$ , siendo su símbolo algebraico  $\overline{a_1 + a_2 + \dots}$ . Se lee “inverso de la suma de  $a_1$  más  $a_2$  más ...”.

Desde el punto de vista del **conexión eléctrico**, se representan el complemento de la suma de conjuntos, colocando los interruptores en paralelo y agregando un elemento que complemente el resultado, como se muestra en la Figura 4.70.

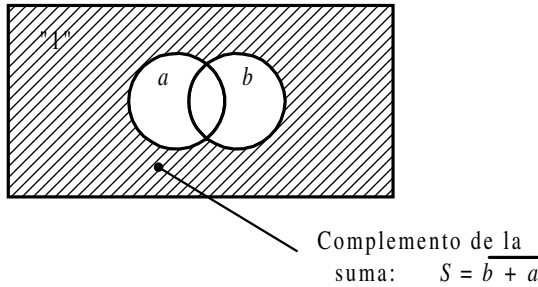


Figura 4.69. El área rayada representa el complemento de la suma o unión de los conjuntos *a* y *b*

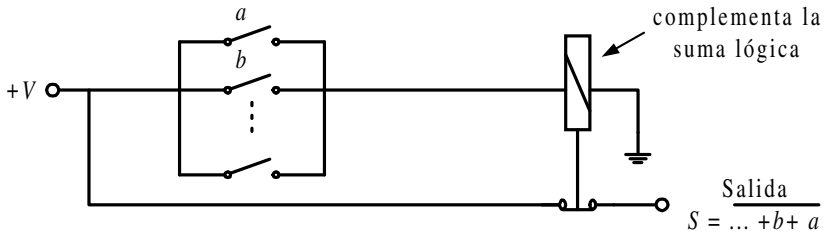


Figura 4.70. Representación eléctrica de la función NOR; complemento de la suma o unión de conjuntos

Otro circuito, resultado de aplicar el teorema de De Morgan a la función NOR,  $f = \dots + b + a = \dots \cdot \overline{b} \cdot \overline{a}$ , es el mostrado en la Figura 4.71.

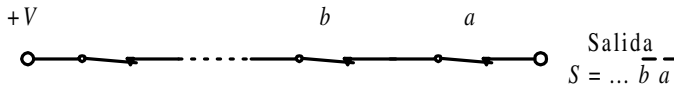


Figura 4.71. Otra representación eléctrica de la función NOR

En las figuras anteriores se aprecia cómo la salida siempre tendrá nivel de tensión +V excepto cuando uno de los interruptores cambie de estado, es decir, el elemento considerado no ha de pertenecer a ninguno de los conjuntos.

La **tabla de verdad** de una puerta NOR de dos variables de entrada se representa en la Tabla 4.5, pudiéndose hacer extensivo a *n* variables (ver Problema resuelto 4-9).

Tabla 4.5. Tabla de verdad de una puerta NOR de dos variables de entrada

$b$	$a$	$S$
0	0	1
0	1	0
1	0	0
1	1	0

El **símbolo** de la puerta NOR es el representado en la Figura 4.72.

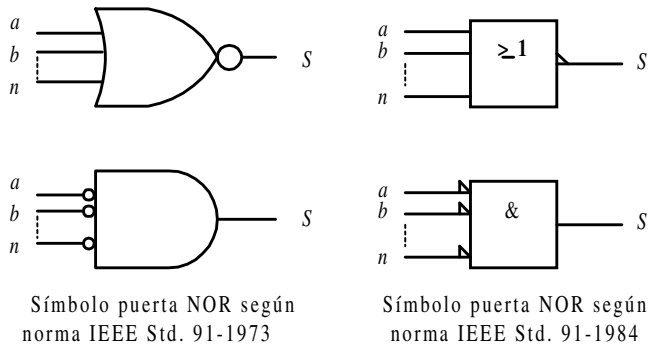


Figura 4.72. Símbolos de la puerta lógica NOR

El **cronograma** de la puerta NOR de dos entradas, que muestra la relación existente, a lo largo del tiempo, entre sus entradas y su salida, se representa en la Figura 4.73.

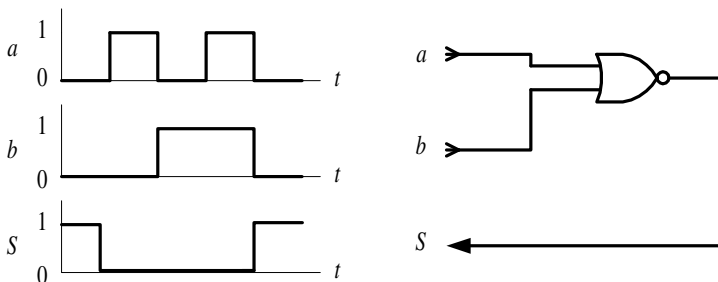


Figura 4.73. Cronograma que relaciona las señales de entrada y salida de una puerta NOR de dos entradas

La **expresión algebraica** de una puerta *NOR*, considerando  $a$ ,  $b$ ,  $c$ ,... a las variables de entrada y  $S$  a la salida, es la siguiente:

$$S = f(\dots, c, b, a) = \overline{\dots + c + b + a} \quad [4.5]$$



Como aplicación práctica de lo expuesto anteriormente, mediante el **lenguaje VHDL** se define con descripción comportamental algorítmica una puerta *NOR* y se simula su cronograma. Se parte del fichero de la Figura 4.60.

El Apéndice B contiene un breve manual del lenguaje VHDL que puede servir de referencia en la comprensión de la descripción *hardware*, para aquellos lectores poco familiarizados con este lenguaje.

La ruta y el nombre del fichero que contiene esta descripción VHDL es la que se indica a continuación:

**D:\Ejemplos\Cap04\VBv99\Nor\NOR\_algo.vpd**

```

nor_algo
-----
-- Fichero :      C:\CAEEN\INDUSTRI\DIGITAL\VHDL\NOR\nor_algo.vhd
-- Lib_trabajo:  C:\CAEEN\INDUSTRI\DIGITAL\VHDL\WORKLIB
-- Por :         Descripción algorítmica de una puerta nor
-- Fecha :
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY puerta_nor IS
    PORT( a, b : in bit;
          S : out bit);
END puerta_nor;

-- Descripción comportamental

ARCHITECTURE algoritmica OF puerta_nor IS
BEGIN

    PROCESS (a,b)
    BEGIN
        IF (a='1' OR b='1') THEN
            S <= '0';
        ELSE
            S <= '1';
        END IF;
    END PROCESS;

END algoritmica;

"C:\CAEEN\Industria\Digital\VHDL\NOR\nor_algo.vhd" saved      Ln 32, Col 1
  
```

Figura 4.74. Descripción comportamental algorítmica de una puerta *NOR* mediante VHDL

En la Figura 4.75 se muestra el fichero de estímulos, en lenguaje VHDL, para simular el comportamiento de una puerta NOR con descripción comportamental algorítmica.

```

nor_stim
-----
-- Fichero :      C:\CAEE\INDUSTRI\DIGITAL\VHDL\NOR\nor_stim.vhd
-- Lib_trabajo:  C:\CAEE\INDUSTRI\DIGITAL\VHDL\WORKLIB
-- Por :         Estimulos de prueba para una puerta nor
-- Fecha :
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY estiaulos IS
END estiaulos;

ARCHITECTURE nor_stim OF estiaulos IS
    COMPONENT puerta_nor
        PORT (a,b: IN bit;
              S: OUT bit);
    END COMPONENT;

    SIGNAL a,b,S: bit;

BEGIN

    puerta_nor1: puerta_nor PORT MAP (a,b,S);

    estiaulos: PROCESS
    BEGIN
        a <= '0';
        b <= '0';
        WAIT FOR 50 ns;
        a <= '1';
        WAIT FOR 50 ns;
        a <= '0';
        b <= '1';
        WAIT FOR 50 ns;
        a <= '1';
        WAIT FOR 50 ns;
    END PROCESS;

END nor_stim;

```

"C:\Cae\Industri\Digita\VHDL\NOR\nor\_stim.vhd" saved Ln 35, Col 24

Figura 4.75. Fichero de estímulos para la simulación de una puerta NOR, con descripción comportamental, mediante lenguaje VHDL

En la Figura 4.76 se muestra el cronograma de la puerta NOR obtenido por simulación a partir de la descripción comportamental algorítmica en lenguaje VHDL.

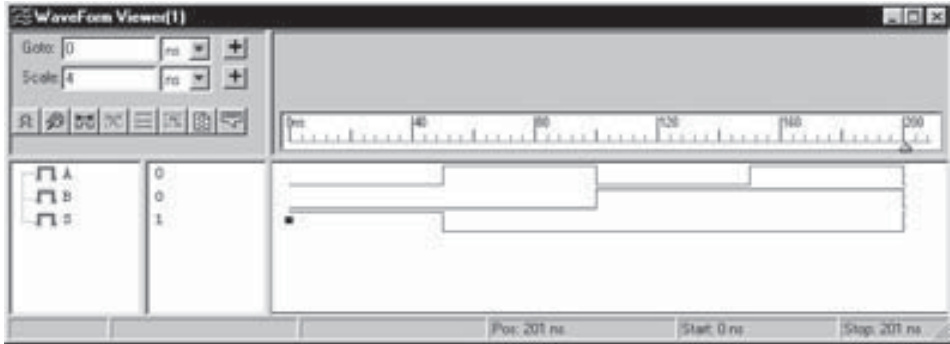


Figura 4.76. Resultado de la simulación, a partir de una descripción en VHDL, de una puerta NOR.

Los **circuitos comerciales** más representativos de puertas NOR son los siguientes:

- 7402: Cuádruple puerta NOR de dos entradas.
- 7427: Cuádruple puerta NOR de tres entradas.
- 7425: Doble puerta NOR de cuatro entradas.
- 74260: Doble puerta NOR de cinco entradas.

Mediante el programa de Fairchild de **Data Sheets** que se incluye en el CDRom#2 que acompaña a este libro se pueden buscar las características de los circuitos comerciales de puertas NOR, en especial aquéllos con las referencias anteriormente indicadas.

A continuación, se desarrollan ejercicios y **ejemplos de aplicación** de puertas NOR utilizando los programas de simulación *Electronics Workbench 5.0 (EWB)* y *OrCAD Demo v9*.

## PROBLEMA RESUELTO 4-9



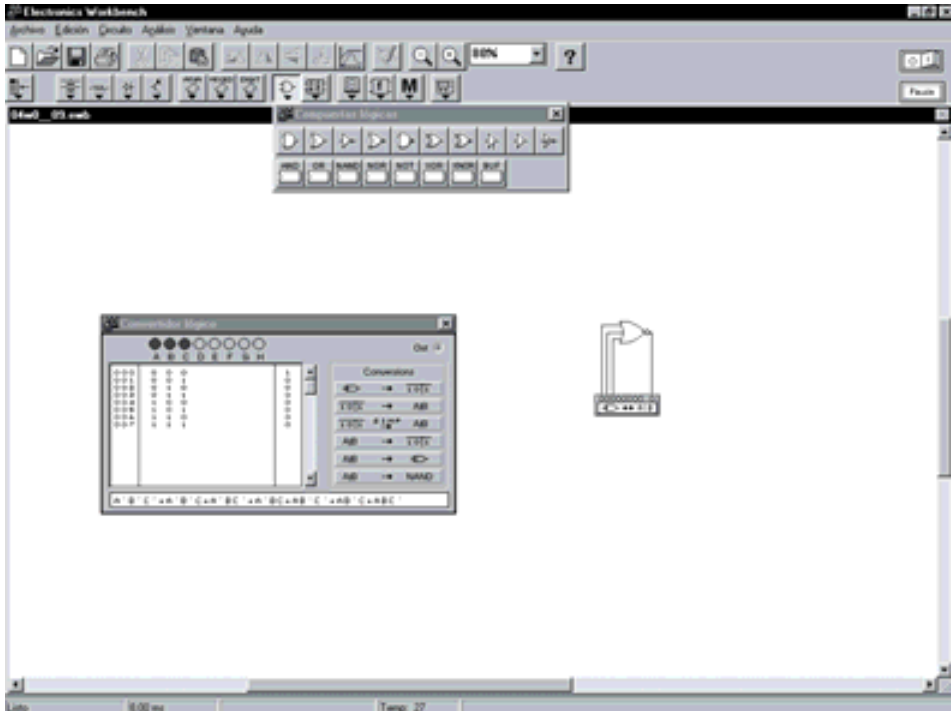
Simular con el programa *Electronics Workbench* el comportamiento de una puerta lógica NOR de tres entradas obteniendo su tabla de verdad y su cronograma.

### Solución:

Mediante el convertidor lógico disponible en el simulador *Electronics Workbench*, se obtiene la tabla de verdad de una puerta NOR de tres entradas de dos formas diferentes: introduciendo su expresión algebraica o representando su circuito lógico. Posteriormente se activará el botón correspondiente según se quiera convertir, de expresión o de circuito lógico a tabla de verdad, como se muestra en la Figura 4.77.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W0\_09.ewb**



*Figura 4.77. Obtención de la tabla de verdad de una puerta NAND de tres entradas mediante EWB*

Mediante el generador de palabras y el analizador lógico disponible en el simulador *Electronics Workbench*, se obtiene el cronograma de una puerta NOR de tres entradas, según se muestra en la Figura 4.78.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W1\_09.ewb**

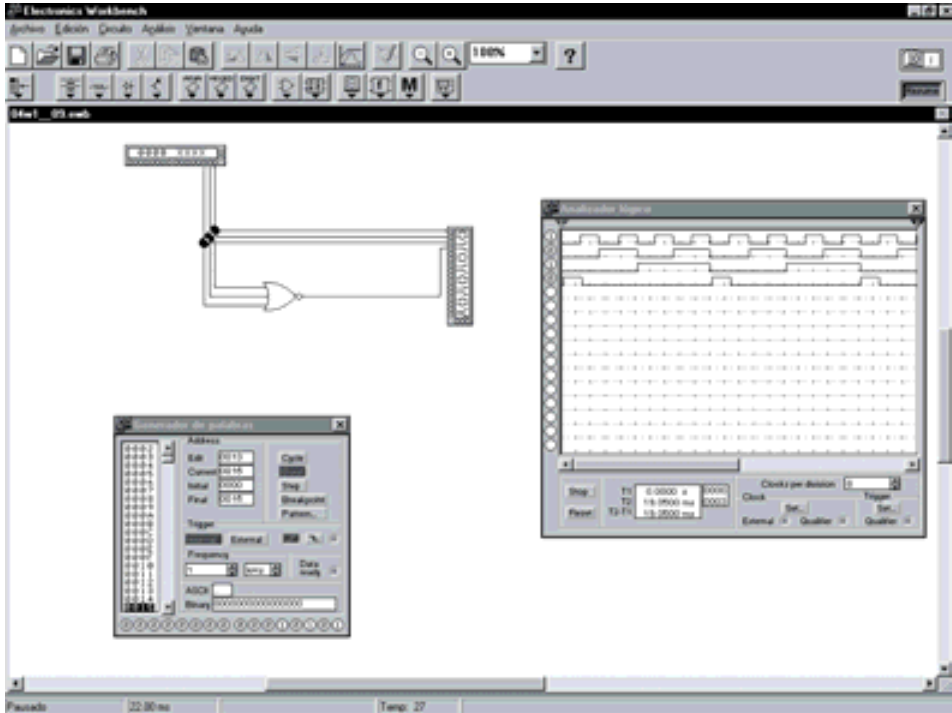


Figura 4.78. Obtención del cronograma de una puerta NOR de tres entradas mediante EWB



Simular con el programa *OrCAD Demo* el comportamiento de una puerta lógica NOR de tres entradas y obtener su cronograma. A partir de éste se puede obtener de forma inmediata su tabla de verdad.

**Solución:**

Para simular el comportamiento de este circuito se utiliza el esquema que se muestra en la Figura 4.79.

La ruta y el nombre del fichero proyecto que contiene este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\OrCAD9\04R0\_09\04R0\_09.opj**

En dicha figura se han representado los valores de las señales de entrada (*a*, *b*, *c*) que deben definirse en la aplicación *Stimulus Editor*, como se explicó en el primer problema resuelto de este capítulo.



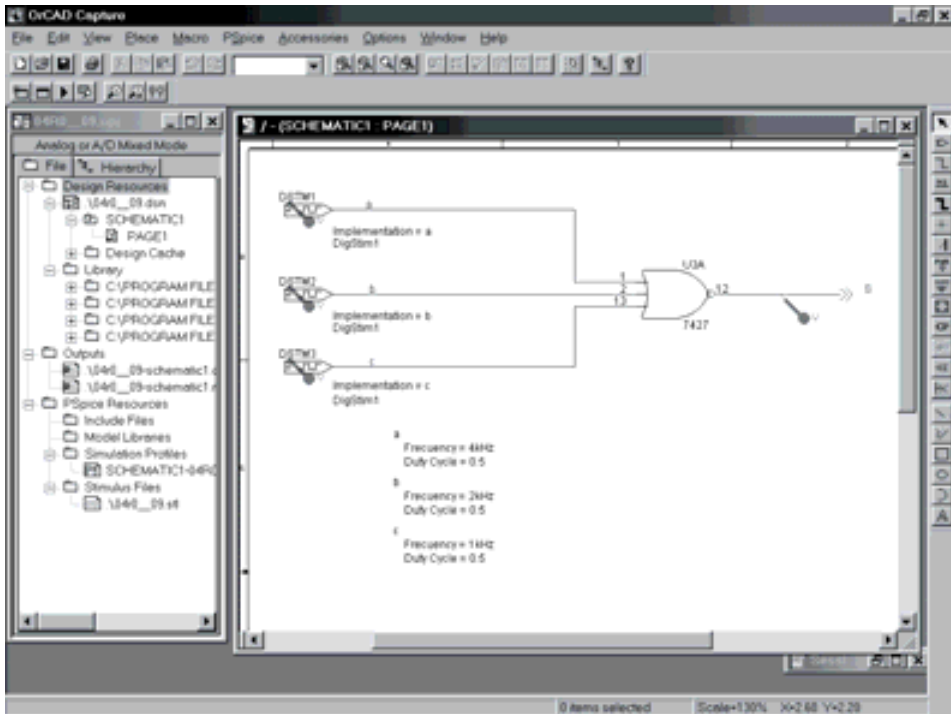


Figura 4.79. Esquema del circuito NOR de tres entradas utilizando la aplicación OrCAD Capture

Para estudiar el comportamiento de este circuito se realiza un análisis transitorio (***Transient...***). Se ha definido en el menú ***PSpice/Edit Simulation Settings/Analysis*** el valor de **2ms** en ***Run to time***. También se ha definido en ***PSpice/Edit Simulation Settings/Stimulus*** el fichero **04R0\_09.st1** de estímulos.

En la Figura 4.80 se muestran los resultados que se obtienen en la aplicación de visualización de *OrCAD PSpice A/D Demo*, después de realizar la simulación.

En dicha aplicación se ha activado el cursor, que permite ver, al lado del nombre de las señales, el valor que toma cada una de ellas en el instante en el que se sitúa el cursor.

Colocando el cursor en cada uno de los puntos de interés, por ejemplo en cada uno de los sucesivos valores que van tomando las señales se puede obtener la tabla de verdad del circuito.

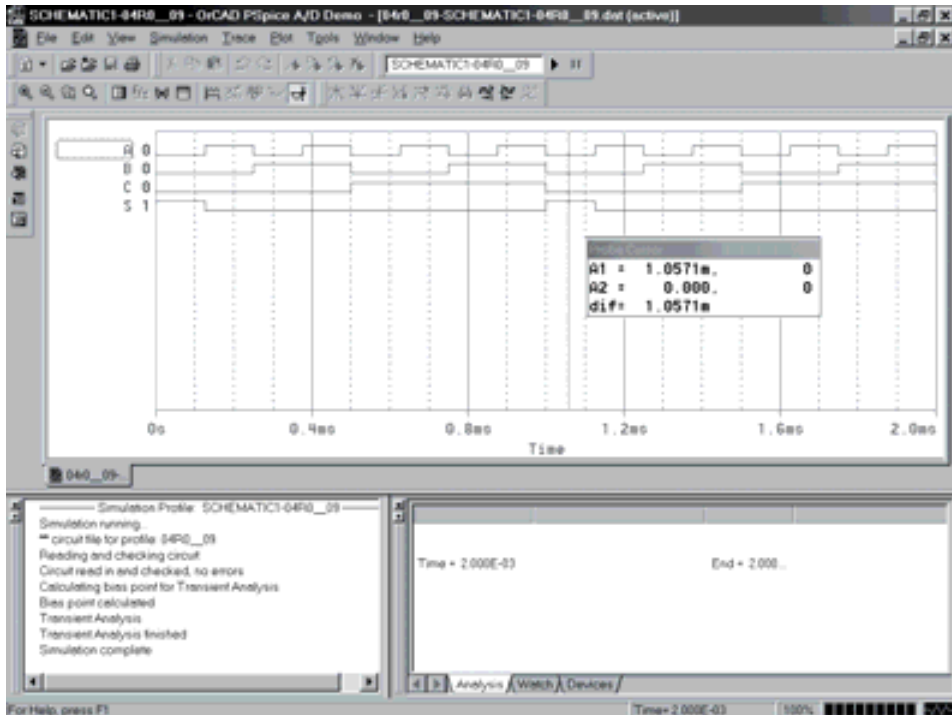


Figura 4.80. Resultados obtenidos en OrCAD PSpice A/D Demo para el circuito NOR

## PROBLEMA RESUELTO 4-10



Con el programa de simulación *Electronics Workbench* diseñar un sistema que indique cuando un automóvil circula con las puertas mal cerradas.

El sistema de detección del estado de las puertas  $p$  de un automóvil entrega un nivel bajo si se detecta alguna puerta mal cerrada. Una señal  $m$  presenta el nivel bajo si el automóvil supera la velocidad de 10 km/h.

### Solución:

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

D:\Ejemplos\Cap04\Ewb5\04W0\_\_10.ewb

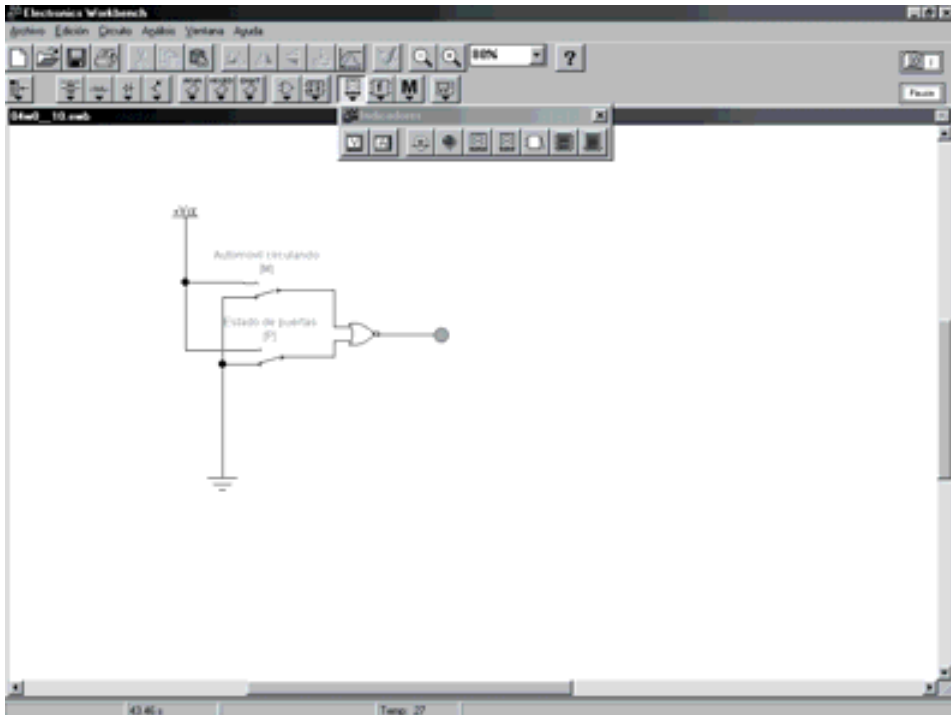


Figura 4.81. Aplicación de puerta NOR: circuito que indica cuándo un automóvil circula con las puertas mal cerradas

## PROBLEMAS PROPUESTOS

- 4-9) Con el programa de simulación *OrCAD Demo* diseñar un sistema que indique cuándo un automóvil circula con las puertas mal cerradas.
- 4-10) Modificar el problema anterior para que la señalización se realice mediante un LED verde que indique la situación en la que el automóvil circula con las puertas bien cerradas. Comprobarlo utilizando las herramientas de simulación.
- 4-11) Representar el cronograma de salida de la puerta NOR (que opera mediante una puerta Negativa-AND) de tres entradas a la que se les aplica las señales de la Figura 4.82. Indicar cómo se modifica el cronograma de salida si se invierte la entrada *a*. Comprobarlo utilizando las herramientas de simulación.

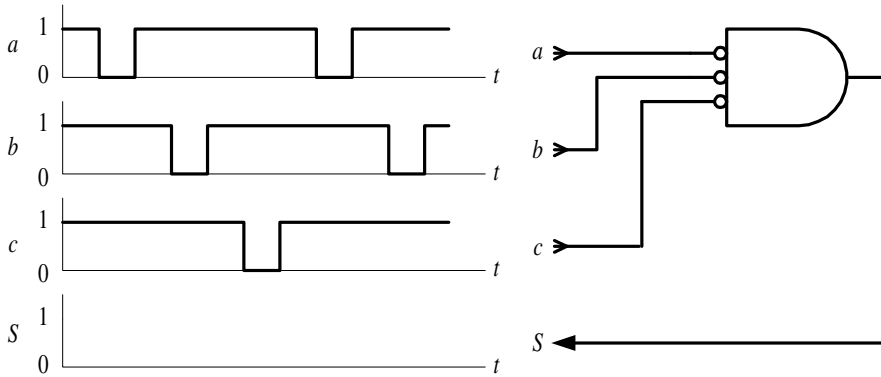


Figura 4.82. Problema propuesto con puerta NOR, representada mediante una puerta Negativa-AND

### 4.1.6 Función SEGUIDOR o puerta BUFFER

Una función lógica seguidor o puerta BUFFER sólo tiene una entrada. La salida es igual a la entrada, es decir, si la entrada vale 0 la salida vale 0 y si la entrada vale 1 la salida es 1.

Aunque la función seguidor no realiza ninguna operación lógica sobre la entrada, se justifica su utilización en aquellas aplicaciones en las que se requiere aumentar la corriente para excitar a dispositivos que así lo requieran.

La función seguidor representa en sí al **conjunto**  $a$ , estando formada por los elementos del conjunto como se muestra en la Figura 4.83.

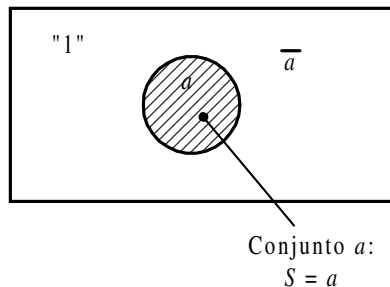


Figura 4.83. El área rayada representa al conjunto  $a$

Desde el punto de vista del **conexionado eléctrico**, se representa la función seguidor mediante un interruptor normalmente abierto. En la Figura 4.84 se representa al conjunto  $a$ .

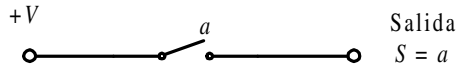


Figura 4.84. Representación eléctrica de la función seguidor

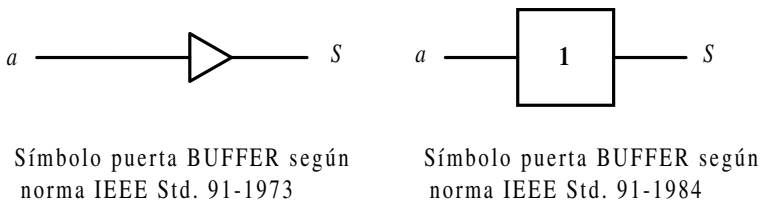
En la Figura 4.44 se aprecia cómo la salida sólo tendrá nivel de tensión  $+V$  cuando se cambie el estado del interruptor  $a$ , es decir, el elemento considerado debe pertenecer al conjunto  $a$ .

La **tabla de verdad** de una puerta BUFFER se representa en la Tabla 4.6.

Tabla 4.6. Tabla de verdad de una puerta BUFFER

$a$	$S$
0	0
1	1

El **símbolo** de la puerta BUFFER es el representado en la Figura 4.85.



Símbolo puerta BUFFER según norma IEEE Std. 91-1973

Símbolo puerta BUFFER según norma IEEE Std. 91-1984

Figura 4.85. Símbolos de la puerta lógica seguidor

El **cronograma** de la puerta BUFFER, que muestra la relación existente, a lo largo del tiempo, entre su entrada y su salida, se representa en la Figura 4.86.

La **expresión algebraica** de una puerta BUFFER, considerando  $a$  como la variable de entrada y  $S$  como salida, es la siguiente:

$$S = f(a) = a \quad [4.6]$$

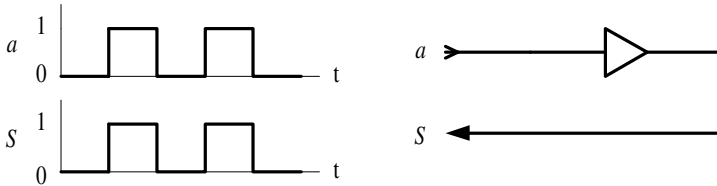


Figura 4.86. Cronograma que relaciona las señales de entradas y de salida de una puerta BUFFER



Como aplicación práctica de lo expuesto anteriormente, mediante el **lenguaje VHDL** se define con descripción comportamental de flujo de datos una puerta BUFFER y se simula su cronograma. Se parte del fichero de la Figura 4.87.

El Apéndice B contiene un breve manual del lenguaje VHDL que puede servir de referencia en la comprensión de la descripción *hardware*, para aquellos lectores poco familiarizados con este lenguaje.

La ruta y el nombre del fichero que contiene esta descripción VHDL es la que se indica a continuación:

**D:\Ejemplos\Cap04\VBv99\Buffer\BUFFER\_flujo.vpd**

```

buffer_flujo *
-----
-- Fichero : C:\CAE\INDUSTRI\DIGITAL\VHDL\buffer\buffer_flujo.vhd
-- Lib_trabajo: C:\CAE\INDUSTRI\DIGITAL\VHDL\WORKLIB
-- Por : Descripción flujo de datos de una puerta buffer
-- Fecha :
-----
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY puerta_buffer IS
    PORT ( a : in bit;
          S : out bit);
END puerta_buffer;

-- Descripción comportamental
ARCHITECTURE flujo_datos OF puerta_buffer IS
    -- SIGNAL ax: bit;
BEGIN
    S<= a;
END flujo_datos;
-----
Ln 22, Col 7

```

Figura 4.87. Descripción comportamental de una puerta BUFFER en VHDL

En la Figura 4.88 se muestra el fichero de estímulos, en lenguaje VHDL, para simular el comportamiento de una puerta BUFFER con descripción comportamental de flujo de datos.

En la Figura 4.89 se muestra el cronograma de la puerta BUFFER obtenido por simulación a partir de la descripción comportamental de flujo de datos en VHDL.

Los **circuitos integrados comerciales** más representativos de puertas BUFFER son los siguientes:

7407/17: BUFFER séxtuple.

Mediante el programa de Fairchild de **Data Sheets** que se incluye en el CDROM#2 que acompaña a este libro se pueden buscar las características de los circuitos comerciales de puertas BUFFER, en especial aquéllos con las referencias anteriormente indicadas.

A continuación, se desarrollan ejercicios y **ejemplos de aplicación** de puertas BUFFER utilizando los programas de simulación *Electronics Workbench 5.0 (EWB)* y *OrCAD Demo v9*.



```

-- Fichero : C:\CAEE\INDUSTRI\DIGITAL\VHDL\buffer\buffer_stim.vhd
-- Lib_trabajo: C:\CAEE\INDUSTRI\DIGITAL\VHDL\WORKLIB
-- Por : Estimulos de prueba para una puerta buffer
-- Fecha :
-----
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY estimulos IS
END estimulos;

ARCHITECTURE buffer_stim OF estimulos IS
    COMPONENT puerta_buffer
        PORT (a: IN bit;
              S: OUT bit);
    END COMPONENT;
    SIGNAL a,S: bit;

BEGIN
    puerta_buffer1: puerta_buffer PORT MAP (a,S);
    estimulos: PROCESS
    BEGIN
        a <= '0';
        WAIT FOR 50 ns;
        a <= '1';
        WAIT FOR 50 ns;
    END PROCESS;
END buffer_stim;

```

"C:\Case\Industri\Digita\VHDL\Buffer\buffer\_stim.vhd" saved. Ln 31, Col 22

Figura 4.88. Fichero de estímulos para simular una puerta BUFFER en VHDL

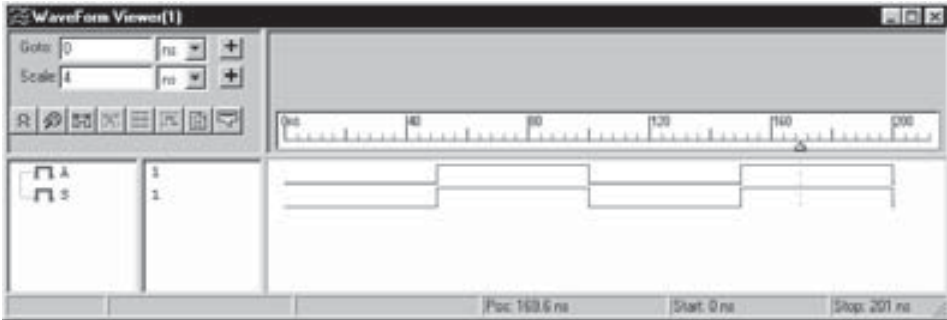


Figura 4.89. Resultado de la simulación, a partir de una descripción en VHDL, de una puerta BUFFER

### PROBLEMA RESUELTO 4-11



Simular el comportamiento de una puerta lógica BUFFER obteniendo su tabla de verdad y su cronograma.

#### Solución:

Mediante el convertidor lógico disponible en el simulador *Electronics Workbench*, se obtiene la tabla de verdad de una puerta BUFFER de dos formas diferentes: introduciendo su expresión algebraica y representando su circuito lógico. Posteriormente, se activará el botón correspondiente según se quiera convertir, de expresión o de circuito lógico a tabla de verdad, como se muestra en la Figura 4.90.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W0\_\_11.ewb**

Mediante un generador de onda cuadrada y el analizador lógico disponible en el simulador *Electronics Workbench*, se obtiene el cronograma de una puerta BUFFER, según se muestra en la Figura 4.91.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W1\_\_11.ewb**



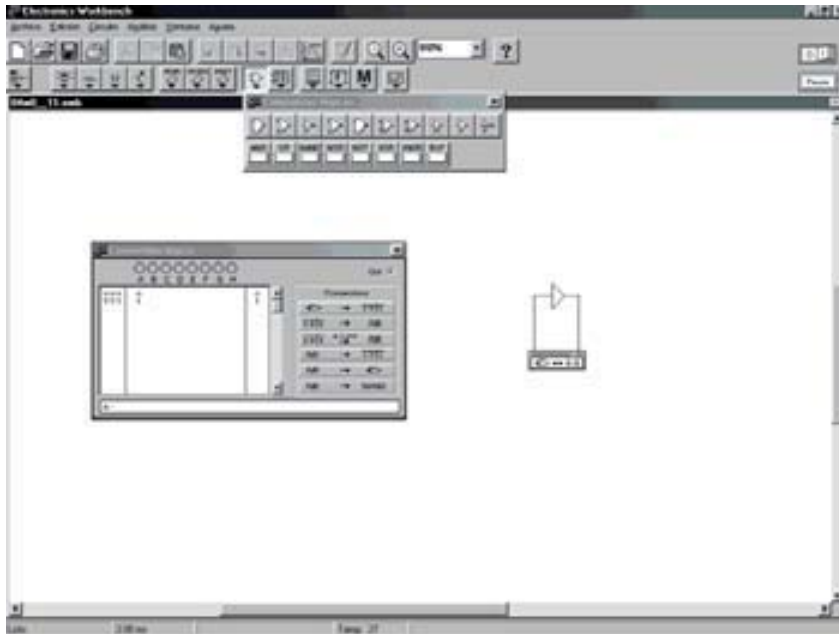


Figura 4.90. Obtención de la tabla de verdad de una puerta BUFFER en EWB

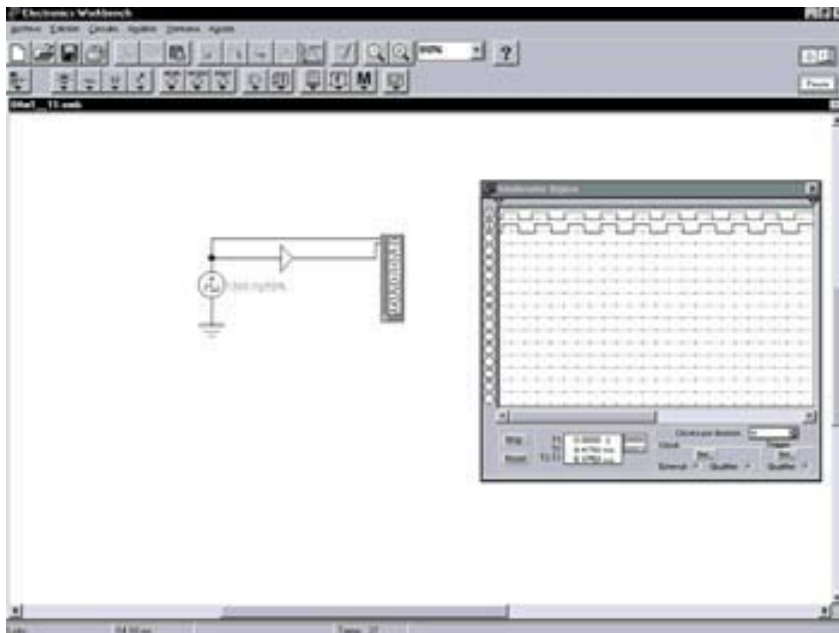


Figura 4.91. Obtención del cronograma de una puerta BUFFER en EWB



Simular con el programa *OrCAD Demo* el comportamiento de una puerta lógica BUFFER y obtener su cronograma. A partir de éste se puede obtener de forma inmediata su tabla de verdad.

### Solución:

Para simular el comportamiento de este circuito se utiliza el esquema que se muestra en la Figura 4.92.

**Nota:** Es necesario colocar una resistencia *pull-up* para polarizar el transistor de salida, al ser esta puerta de colector abierto. En el capítulo que trata sobre las tecnologías de circuitos integrados digitales se explicará este concepto.

La ruta y el nombre del fichero proyecto que contiene este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\OrCAD9\04R0\_\_11\04R0\_\_11.opj**

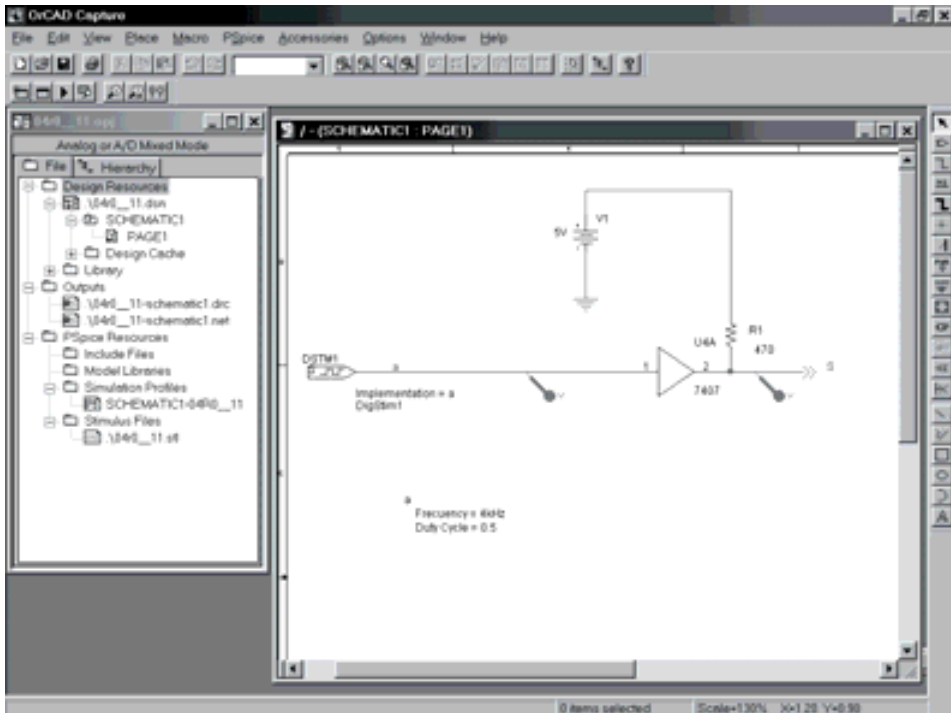


Figura 4.92. Esquema del circuito BUFFER utilizando la aplicación *OrCAD Capture*

En dicha figura se han representado los valores de la señal de entrada *a*, que deben definirse en la aplicación *Stimulus Editor*, como se explicó en el primer problema resuelto de este capítulo.

Para estudiar el comportamiento de este circuito se realiza un análisis transitorio (*Transient...*). Se ha definido en el menú *PSpice/Edit Simulation Settings/Analysis* el valor de **5ms** a *Run to time*. También está definido en *PSpice/Edit Simulation Settings/Stimulus* el fichero **04R0\_11.st1** de estímulos.

En la Figura 4.93 se muestran los resultados que se obtienen en la aplicación de visualización de *OrCAD PSpice A/D Demo*, después de realizar la simulación. En dicha aplicación se ha activado el cursor, que permite ver, al lado del nombre de las señales, el valor que toma cada una de ellas en el instante en el cual se sitúa el mismo. Situando el cursor en cada uno de los puntos de interés, por ejemplo en cada uno de los sucesivos valores que van tomando las señales se puede obtener la tabla de verdad del circuito.

**Nota:** La salida de la puerta, que es de colector abierto, es tratada y representada por el simulador como señal analógica.

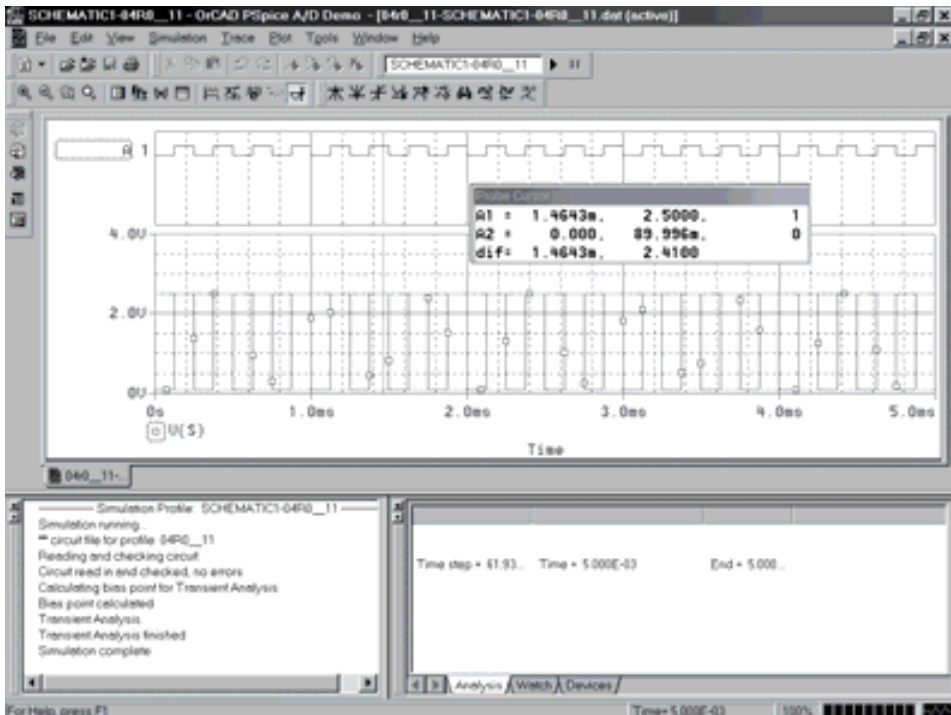


Figura 4.93. Resultados obtenidos en OrCAD PSpice A/D Demo para el circuito *BUFFER*

### PROBLEMAS PROPUESTOS

4-12) Diseñar un circuito eléctrico, con relés, que realice la función BUFFER. Comprobarlo utilizando las herramientas de simulación.

#### 4.1.7 Función XOR

La salida de una puerta XOR vale 1 cuando el número de entradas con valor igual a 1 sea impar y su salida vale 0 cuando el número de entradas con valor igual a 1 sea par. Para el caso particular de puertas XOR de dos entradas, su salida vale 1 cuando una de sus entradas vale 1 y la otra vale 0 (esto es, las variables de entrada tomen valores distintos).

La función XOR de dos conjuntos efectúa la operación **b o a pero no ambas** siendo el conjunto formado por los elementos que pertenecen a b o pertenecen a a pero no a ambas ( $S = \bar{a}b + a\bar{b}$ ), como se muestra en la Figura 4.94.

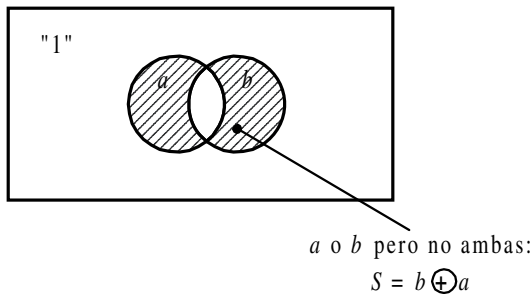


Figura 4.94. El área rayada representa la función XOR de los conjuntos a y b

La función XOR de varios conjuntos  $a_1, a_2, \dots$ , tiene por **símbolo** algebraico  $a_1 \cdot \oplus \cdot a_2 \cdot \oplus \dots$  y se lee “ $a_1$  or-exclusiva  $a_2$  or-exclusiva ...”.

Desde el punto de vista del **conexionado eléctrico**, la representación de  $S = \bar{a}b + a\bar{b}$  se muestra en la Figura 4.95. Se aprecia en dicha figura que la salida siempre tendrá nivel de tensión +V cuando sólo uno de los interruptores cambie de estado, es decir, el elemento considerado ha de pertenecer a uno de los conjuntos y al otro no.

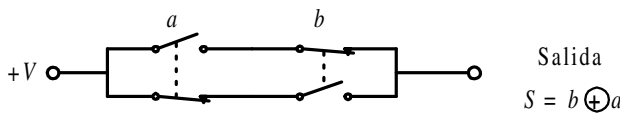


Figura 4.95. Representación eléctrica de la función XOR

La **tabla de verdad** de una puerta XOR de dos variables de entrada se representa en la Tabla 4.7, pudiéndose hacer extensivo a  $n$  variables (Problema resuelto 4-12).

Tabla 4.7. Tabla de verdad de una puerta XOR de dos variables de entrada

$b$	$a$	$S$
0	0	0
0	1	1
1	0	1
1	1	0

El **símbolo** de la puerta XOR es el representado en la Figura 4.96.

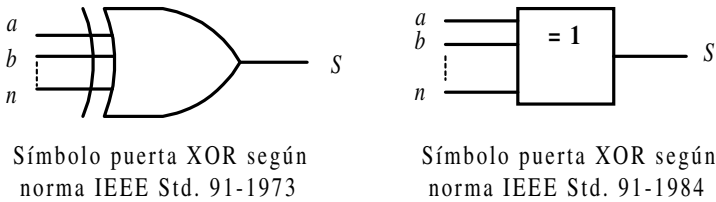


Figura 4.96. Símbolos de la puerta lógica XOR

El **cronograma** de la puerta XOR de dos entradas, que muestra la relación existente, a lo largo del tiempo, entre sus entradas y su salida, se representa en la Figura 4.97.

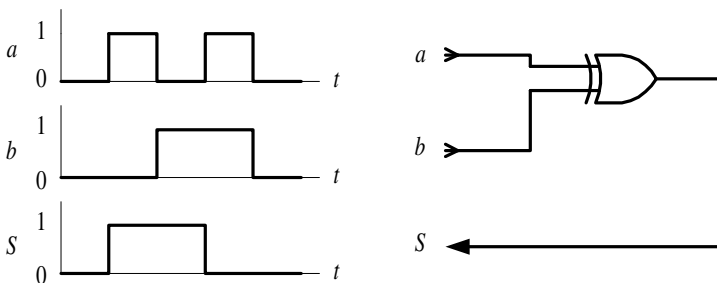


Figura 4.97. Cronograma que relaciona las señales de entrada y salida de una puerta XOR de dos entradas

La **expresión algebraica** de una puerta XOR, considerando  $a$ ,  $b$ ,  $c$ ,... a las variables de entrada y  $S$  a la salida, es la siguiente:

$$S = f(\dots, c, b, a) = \dots c \oplus b \oplus a \quad [4.7]$$

Una función XOR de dos variables se puede expresar como función canónica según se indica en la expresión [4.8].

$$S = f(\dots, c, b, a) = b \oplus a = \bar{b} a + b \bar{a} = \overline{(b + \bar{a})} \overline{(\bar{b} + a)} \quad [4.8]$$

El circuito lógico de la expresión [4.8] correspondiente a una puerta XOR es el representado en la Figura 4.98.

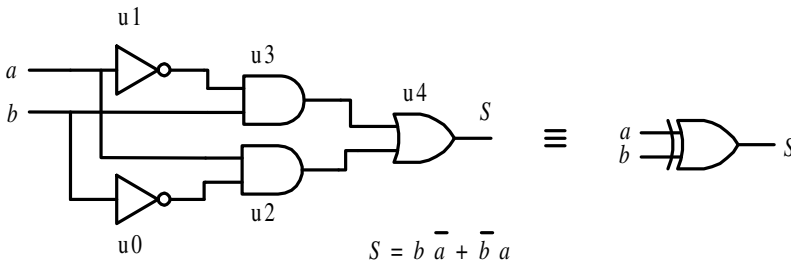


Figura 4.98. Circuitos lógicos de una puerta XOR



Como aplicación práctica, mediante el **lenguaje VHDL** se realiza la descripción estructural de la Figura 4.98, con las referencias de puertas indicadas (u0 a u4) correspondiente a una puerta XOR. En la Figura 4.99 se muestra el contenido de esta descripción.

El Apéndice B contiene un breve manual del lenguaje VHDL que puede servir de referencia en la comprensión de la descripción *hardware*, para aquellos lectores poco familiarizados con este lenguaje.

La ruta y el nombre del fichero que contiene esta descripción VHDL es la que se indica a continuación:

**D:\Ejemplos\Cap04\VBv99\Xor\XOR\_estr.vpd**

En la Figura 4.100 se muestra el contenido del fichero de estímulos, en lenguaje VHDL, para simular el comportamiento de una puerta XOR con descripción estructural.

En la Figura 4.101 se muestra el cronograma de la puerta XOR obtenido por simulación a partir de la descripción estructural en lenguaje VHDL.

```

-----
-- Fichero : C:\CAEE\INDUSTRI\DIGITAL\VHDL\XOR\Xor_estr.vhd
-- Lib_trabajo: C:\CAEE\INDUSTRI\DIGITAL\VHDL\WORKLIB
--           Descripción estructural de una puerta XOR
-- Por :
-- Fecha :
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Definición de entidades y arquitectura de cada componente

ENTITY inv IS PORT(e:IN bit; y:OUT bit);END inv;
ENTITY and2 IS PORT(e1,e2:IN bit; y:OUT bit);END and2;
ENTITY or2 IS PORT(e1,e2:IN bit; y:OUT bit);END or2;

ARCHITECTURE rtl OF inv IS BEGIN y<=NOT e;END rtl;
ARCHITECTURE rtl OF and2 IS BEGIN y<=e1 AND e2;END rtl;
ARCHITECTURE rtl OF or2 IS BEGIN y<=e1 OR e2;END rtl;

-- Definición de la entidad y arquitectura del conjunto o circuito
-- (netlist)

ENTITY circuito_xor IS
PORT (a,b: IN bit;
      S: OUT bit);
END circuito_xor;

ARCHITECTURE estructural OF circuito_xor IS
    COMPONENT inv IS PORT(e:IN bit; y:OUT bit);END COMPONENT;
    COMPONENT and2 IS PORT(e1,e2:IN bit; y:OUT bit);END COMPONENT;
    COMPONENT or2 IS PORT(e1,e2:IN bit; y:OUT bit);END COMPONENT;

    SIGNAL na,nb,anb,bna:bit;

BEGIN
    u0: inv PORT MAP (e=>b,y=>nb);
    u1: inv PORT MAP (e=>a,y=>na);
    u2: and2 PORT MAP (e1=>a,e2=>nb,y=>anb);
    u3: and2 PORT MAP (e1=>b,e2=>na,y=>bna);
    u4: or2 PORT MAP (e1=>anb,e2=>bna,y=>S);

END estructural;

-- configuración

CONFIGURATION estruc OF circuito_xor IS
    FOR estructural
        FOR ALL: inv USE ENTITY work.inv; END FOR;
        FOR ALL: and2 USE ENTITY work.and2; END FOR;
        FOR ALL: or2 USE ENTITY work.or2; END FOR;
    END FOR;

END CONFIGURATION estruc;

```

*Figura 4.99. Descripción estructural de una puerta XOR mediante VHDL*

```

-----
-- Fichero : C:\CAEE\INDUSTRI\DIGITAL\VHDL\XOR\Xor_stim.vhd
-- Lib_trabajo: C:\CAEE\INDUSTRI\DIGITAL\VHDL\WORKLIB
--           Estimulos de prueba para una puerta XOR
-- Por      :
-- Fecha   :
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY estimulos IS
END estimulos;

ARCHITECTURE xor_stim OF estimulos IS

    COMPONENT circuito_xor
        PORT (a,b: IN bit;
              S: OUT bit);
    END COMPONENT;

    SIGNAL a,b,S: bit;

BEGIN

    circuito_xor1: circuito_xor PORT MAP (a,b,S);

    estimulos: PROCESS
    BEGIN
        a <= '0';
        b <= '0';
        WAIT FOR 50 ns;
        a <= '1';
        WAIT FOR 50 ns;
        a <= '0';
        b <= '1';
        WAIT FOR 50 ns;
        a <= '1';
        WAIT FOR 50 ns;
    END PROCESS;

END xor_stim;

```

*Figura 4.100. Fichero de estímulos para la simulación de una puerta XOR, con descripción estructural, mediante lenguaje VHDL*

Los **circuitos comerciales** más representativos de puertas XOR son los siguientes:

7486: Cuádruple puerta XOR de dos entradas.

74136: Cuádruple puerta XOR de dos entradas con salidas colector abierto.

Mediante el programa de Fairchild de **Data Sheets** que se incluye en el CDRom#2 que acompaña a este libro se pueden buscar las características de los circuitos comerciales de puertas XOR, en especial aquéllos con las referencias anteriormente indicadas.



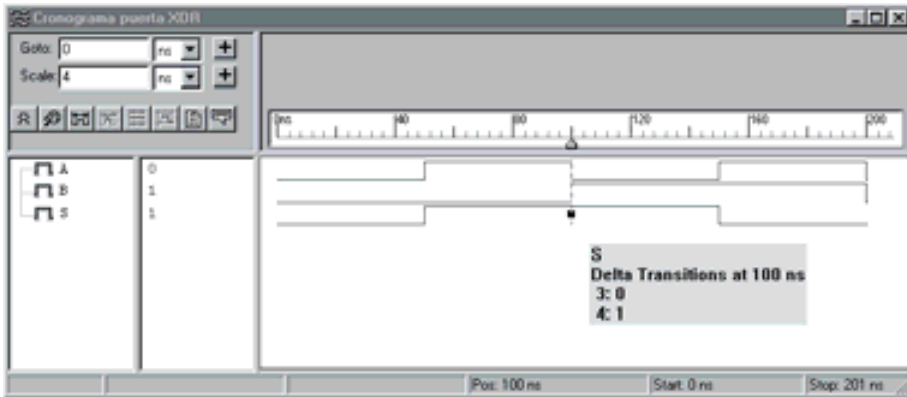


Figura 4.101. Resultado de la simulación, a partir de una descripción en VHDL, de una puerta XOR.

A continuación, se desarrollan ejercicios y **ejemplos de aplicación** de puertas XOR con los programas *Electronics Workbench 5.0 (EWB)* y *OrCAD Demo v9*.

## PROBLEMA RESUELTO 4-12



Simular con el programa EWB el comportamiento de una puerta lógica XOR de tres entradas obteniendo su tabla de verdad y su cronograma. Verificar que esta puerta se comporta como un generador de paridad par.

### Solución:

Mediante el convertidor lógico disponible en el simulador *Electronics Workbench*, se obtiene la tabla de verdad de una puerta XOR de tres entradas de dos formas diferentes: introduciendo su expresión algebraica y representando su circuito lógico. Posteriormente, se activará el botón correspondiente según se quiera convertir, de expresión o de circuito lógico a tabla de verdad, como se muestra en la Figura 4.102.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W0\_\_12.ewb**

Mediante el generador de palabras y el analizador lógico disponible en el simulador *Electronics Workbench*, se obtiene el cronograma de una puerta XOR de tres entradas, según se muestra en la Figura 4.103.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W1\_\_12.ewb**

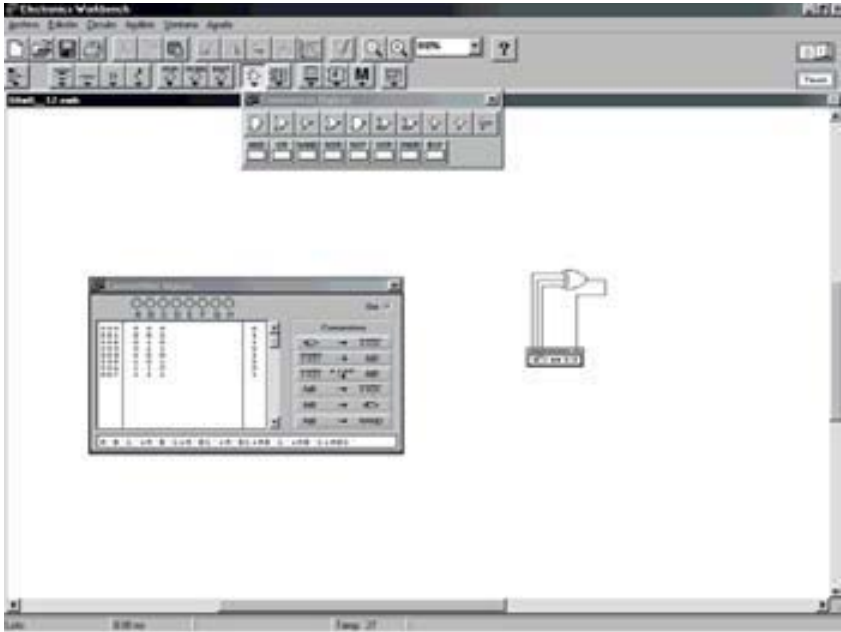


Figura 4.102. Tabla de verdad de una puerta XOR de tres entradas en EWB

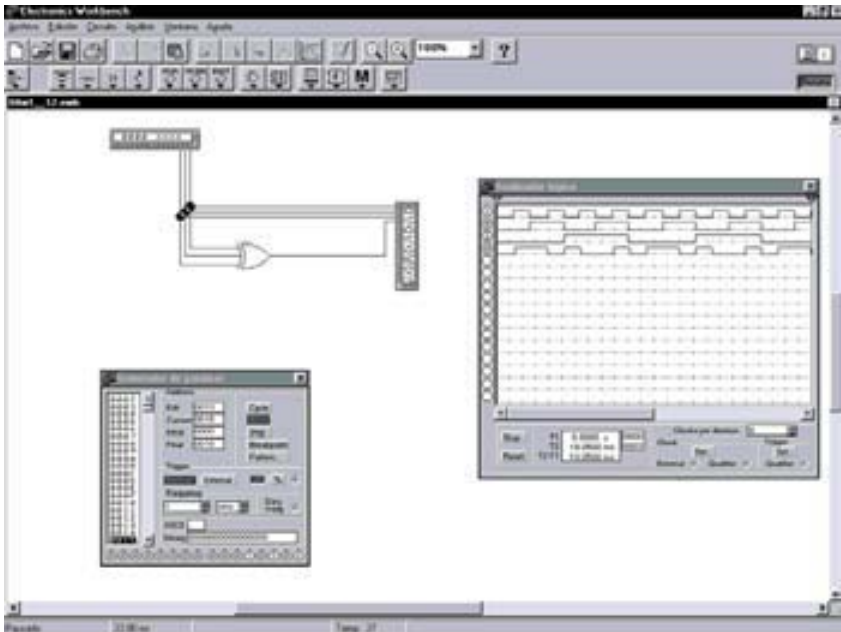


Figura 4.103. Cronograma de una puerta XOR de tres entradas en EWB



Simular con el programa *OrCAD Demo* el comportamiento de una puerta lógica XOR de tres entradas y obtener su cronograma. A partir de éste se puede obtener de forma inmediata su tabla de verdad. Verificar que esta puerta se comporta como un generador de paridad par.

### Solución:

Para simular el comportamiento de este circuito se utiliza el esquema que se muestra en la Figura 4.104. La puerta XOR de tres entradas se ha construido a partir del circuito integrado 7486 compuesto de puertas XOR de dos entradas.

La ruta y el nombre del fichero proyecto que contiene este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\OrCAD9\04R0\_\_12\04R0\_\_12.opj**

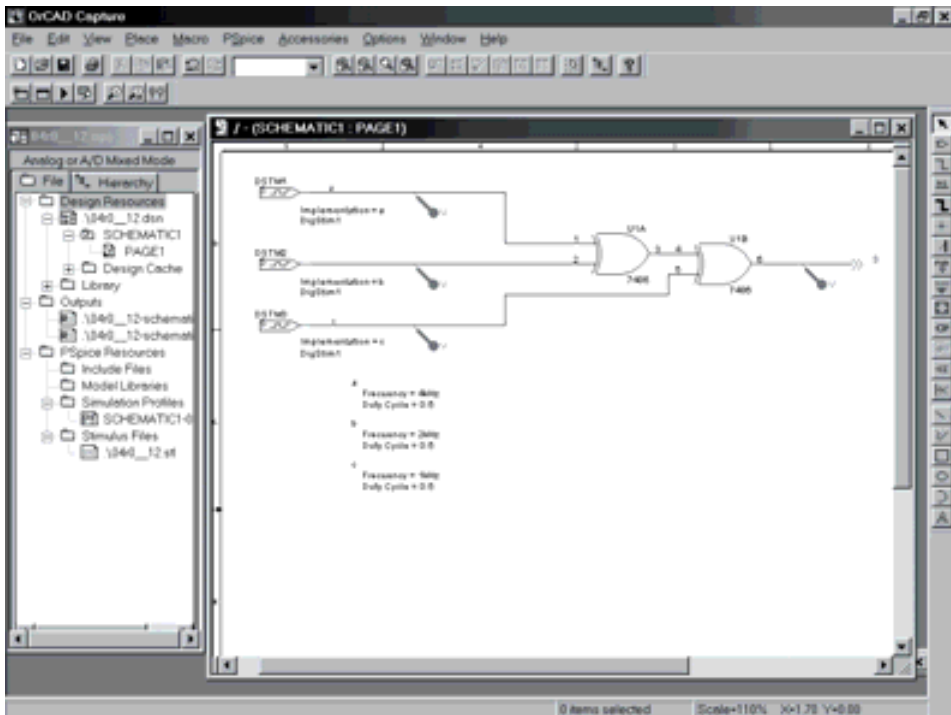


Figura 4.104. Esquema del circuito XOR de tres entradas utilizando la aplicación *OrCAD Capture*

En dicha figura se han representado los valores de las señales de entrada (*a*, *b*, *c*) que deben definirse en la aplicación *Stimulus Editor*, como se explicó en el primer problema resuelto de este capítulo.

Para estudiar el comportamiento de este circuito se realiza un análisis transitorio (*Transient...*). Se ha definido en el menú *PSpice/Edit Simulation Settings/Analysis* el valor de **2ms** en *Run to time*. También está definido en *PSpice/Edit Simulation Settings/Stimulus* el fichero **04R0\_12.st1** de estímulos.

En la Figura 4.105 se muestran los resultados que se obtienen en la aplicación de visualización de *OrCAD PSpice A/D Demo*, después de realizar la simulación. En dicha aplicación se ha activado el cursor, que permite ver, al lado del nombre de las señales, el valor que toma cada una de ellas en el instante en el cual se sitúa el mismo. Situando el cursor en cada uno de los puntos de interés, por ejemplo en cada uno de los sucesivos valores que van tomando las señales se puede obtener la tabla de verdad del circuito.

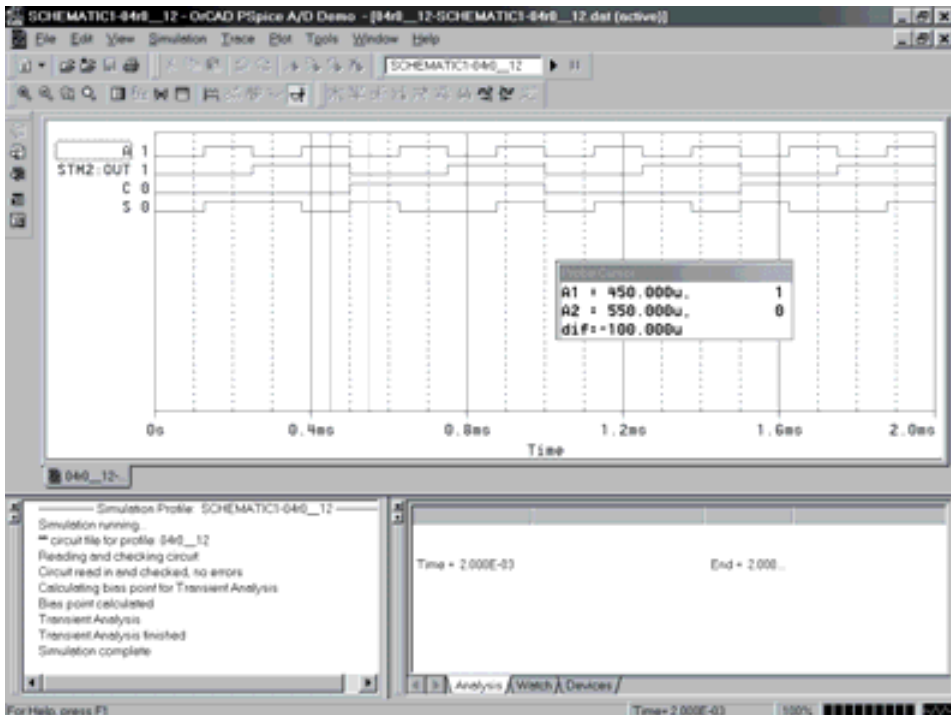


Figura 4.105. Resultados obtenidos en OrCAD PSpice A/D Demo para el circuito XOR

### PROBLEMA RESUELTO 4-13



Se trata de diseñar, utilizando el programa *Electronics Workbench*, un circuito que permita realizar el control de calidad en una cadena de producción.

La salida de cada equipo fabricado en la cadena de producción es comparada con la de un equipo patrón para detectar si presenta algún defecto, es decir, si no son iguales. Esto indicaría un fallo en el equipo fabricado.

#### Solución:

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W0\_\_13.ewb**

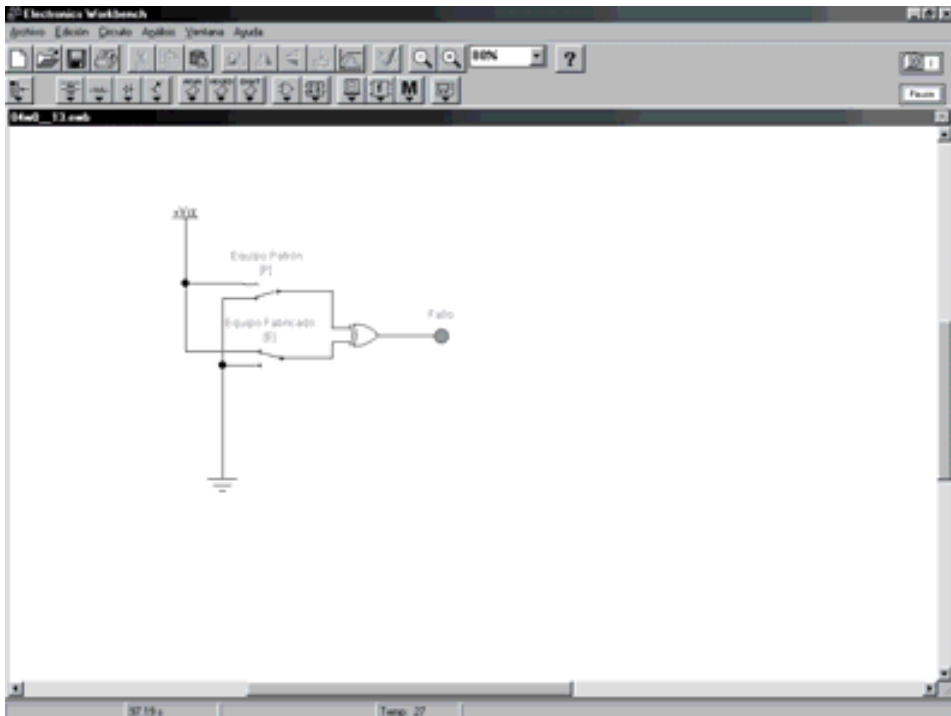


Figura 4.106. Aplicación de puerta XOR: como control de calidad de un equipo

## PROBLEMAS PROPUESTOS

- 4-13) Diseñar, utilizando el programa *OrCAD Demo*, un circuito que permita realizar el control de calidad en una cadena de producción.
- 4-14) Comprobar mediante simulación tres funciones diferentes y equivalentes de una puerta XOR representadas en la expresión [4.8]. Definir los tres circuitos lógicos correspondientes y comprobar que tienen la misma tabla de verdad (la de la función XOR).
- 4-15) Diseñar un generador de paridad par de 8 bits de datos más el de paridad, utilizando las herramientas de simulación.

### 4.1.8 Función XNOR

La puerta XNOR es el complemento de la puerta XOR.

La salida de una puerta XNOR vale 1 cuando el número de entradas con valor igual a 1 sea par y su salida vale 0 cuando el número de entradas con valor igual a 1 sea impar. Para el caso particular de puertas XNOR de dos entradas, su salida vale 1 cuando ambas entradas valen 1 o ambas entradas valen 0 (esto es, las variables de entrada tomen valores iguales).

La función XNOR de dos conjuntos efectúa la operación **b igual a a** siendo el conjunto formado por los elementos que pertenecen a  $a$  y  $b$  o no pertenecen a  $a$  y  $b$  ( $S = ab + \bar{a}\bar{b}$ ), como se muestra en la Figura 4.107.

La función XNOR en dos conjuntos efectúa la operación **b y a, o no ambas**, como se muestra en la Figura 4.107.

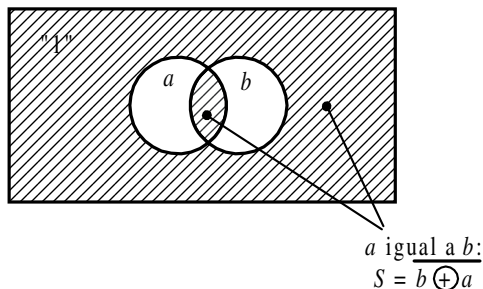


Figura 4.107. El área rayada representa la función XNOR de los conjuntos  $a$  y  $b$

La función XNOR de varios conjuntos  $a_1, a_2, \dots$ , tiene por **símbolo** algebraico  $\overline{a_1 \oplus a_2 \oplus \dots}$  y se lee “negación de  $a_1$  or-exclusiva  $a_2$  or-exclusiva ...”.

Desde el punto de vista del **conexionado eléctrico**, la representación de la función XNOR  $S = a b + \bar{a} \bar{b}$  se muestra en la Figura 4.108. Se aprecia en dicha figura que la salida siempre tendrá nivel de tensión +V cuando ninguno de los dos interruptores cambien de estado o cambien los dos, es decir, el elemento considerado ha de pertenecer a los dos conjuntos o a ninguno de ellos.

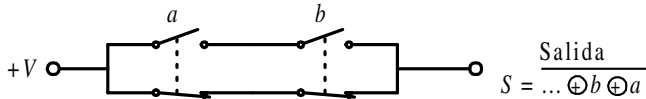


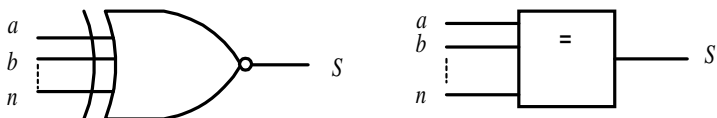
Figura 4.108. Representación eléctrica de la función XNOR; complemento de la suma o unión de conjuntos

La **tabla de verdad** de una puerta XNOR de dos variables de entrada se representa en la Tabla 4.8, pudiéndose hacer extensivo a  $n$  variables (Problema resuelto 4-14).

Tabla 4.8. Tabla de verdad de una puerta XNOR de dos variables de entrada

$b$	$a$	$S$
0	0	1
0	1	0
1	0	0
1	1	1

El **símbolo** de la puerta XNOR es el representado en la Figura 4.109.



Símbolo puerta XNOR según norma IEEE Std. 91-1973

Símbolo puerta XNOR según norma IEEE Std. 91-1984

Figura 4.109. Símbolos de la puerta lógica XNOR

El **cronograma** de la puerta XNOR de dos entradas, que muestra la relación existente, a lo largo del tiempo, entre sus entradas y su salida, se representa en la Figura 4.110.

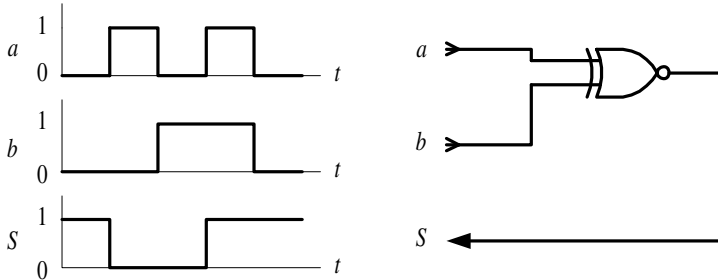


Figura 4.110. Cronograma que relaciona las señales de entradas y salida de una puerta XNOR de dos entradas

La **expresión algebraica** de una puerta XNOR, considerando  $a, b, c, \dots$  a las variables de entrada y  $S$  a la salida, es la siguiente:

$$S = f(\dots, c, b, a) = \overline{\dots c \oplus b \oplus a} = \dots \odot c \odot b \odot a \quad [4.9]$$

Una función XNOR de dos variables se puede expresar como función canónica según se indica en la expresión [4.10].

$$S = f(\dots, c, b, a) = \overline{b \oplus a} = \bar{b} \bar{a} + b a = \overline{(b + a)} \overline{(\bar{b} + \bar{a})} \quad [4.10]$$



Como aplicación de lo descrito anteriormente, mediante el **lenguaje VHDL** se define con descripción estructural de una puerta XNOR y se simula su cronograma. En la Figura 4.99 se muestra la descripción estructural.

El Apéndice B contiene un breve manual del lenguaje VHDL que puede servir de referencia en la comprensión de la descripción *hardware*, para aquellos lectores poco familiarizados con este lenguaje.

La ruta y el nombre del fichero que contiene esta descripción VHDL es la que se indica a continuación:

**D:\Ejemplos\Cap04\VBv99\Xnor\XNOR\_estr.vpd**

En la Figura 4.112 se muestra el fichero de estímulos, en lenguaje VHDL, para simular el comportamiento de una puerta XNOR con descripción estructural.

La ruta y el nombre del fichero que contiene el fichero de estímulos es la que se indica a continuación:

**D:\Ejemplos\Cap04\VBv99\Xnor\XNOR\_stim.vhd**



```

Xnor_estr *
-----
-- Fichero :      C:\CAEE\INDUSTRI\DIGITAL\VHDL\XNOR\Xnor_estr.vhd
-- Lib_trabajo:  C:\CAEE\INDUSTRI\DIGITAL\VHDL\WORKLIB
-- Por :         Descripción estructural de una puerta XNOR
-- Fecha :
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Definición de entidades y arquitectura de cada componente

ENTITY puerta_xnor IS
    PORT( ap, bp : in bit;
          Sp : out bit);
END puerta_xnor;

ARCHITECTURE rtl OF puerta_xnor IS
BEGIN
    Sp <= ap XNOR bp;
END rtl;

-- Definición de la entidad y arquitectura del conjunto o circuito (

ENTITY circuito_xnor IS
PORT (a,b: IN bit;
      S: OUT bit);
END circuito_xnor;

ARCHITECTURE estructural OF circuito_xnor IS
    COMPONENT puerta_xnor IS
        PORT (ap,bp: IN bit; Sp: OUT bit);
    END COMPONENT puerta_xnor;

BEGIN
    u0: puerta_xnor PORT MAP (ap=>a,bp=>b,Sp=>S);
END estructural;
-----
Ln 4, Col 59 CAP

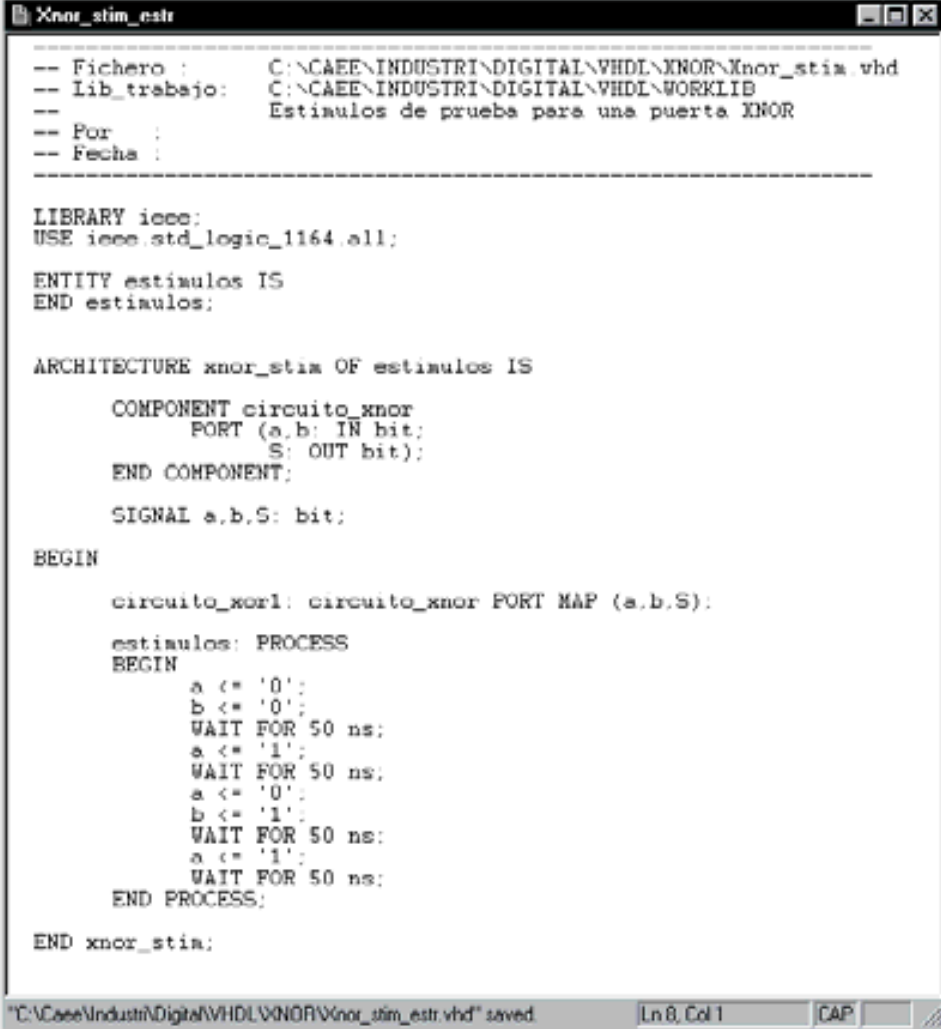
```

Figura 4.111. Descripción estructural de una puerta XNOR mediante VHDL

En la Figura 4.113 se muestra el cronograma de la puerta XNOR obtenido por simulación a partir de la descripción estructural en lenguaje VHDL.

Los **circuitos comerciales** más representativos de puertas XNOR son los siguientes:

74266: Cuádruple puerta XNOR de dos entradas con salidas colector abierto.



```
-- Fichero :      C:\CAEE\INDUSTRI\DIGITAL\VHDL\XNOR\Xnor_stia.vhd
-- Lib_trabajo:  C:\CAEE\INDUSTRI\DIGITAL\VHDL\WORKLIB
--
-- Por :         Estimulos de prueba para una puerta XNOR
-- Fecha :
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY estiaulos IS
END estiaulos;

ARCHITECTURE xnor_stia OF estiaulos IS

    COMPONENT circuito_xnor
        PORT (a,b: IN bit;
              S: OUT bit);
    END COMPONENT;

    SIGNAL a,b,S: bit;

BEGIN

    circuito_xor1: circuito_xnor PORT MAP (a,b,S);

    estiaulos: PROCESS
    BEGIN
        a <= '0';
        b <= '0';
        WAIT FOR 50 ns;
        a <= '1';
        WAIT FOR 50 ns;
        a <= '0';
        b <= '1';
        WAIT FOR 50 ns;
        a <= '1';
        WAIT FOR 50 ns;
    END PROCESS;

END xnor_stia;
```

Figura 4.112. Fichero de estímulos para la simulación de una puerta XNOR, con descripción estructural, mediante lenguaje VHDL

Mediante el programa de Fairchild de **Data Sheets** que se incluye en el CDRom#2 que acompaña a este libro se pueden buscar las características de los circuitos comerciales de puertas XNOR, en especial aquéllos con las referencias anteriormente indicadas.

A continuación, se desarrollan ejercicios y **ejemplos de aplicación** de puertas XNOR utilizando los programas de simulación *Electronics Workbench (EWB)* y *OrCAD Demo*.

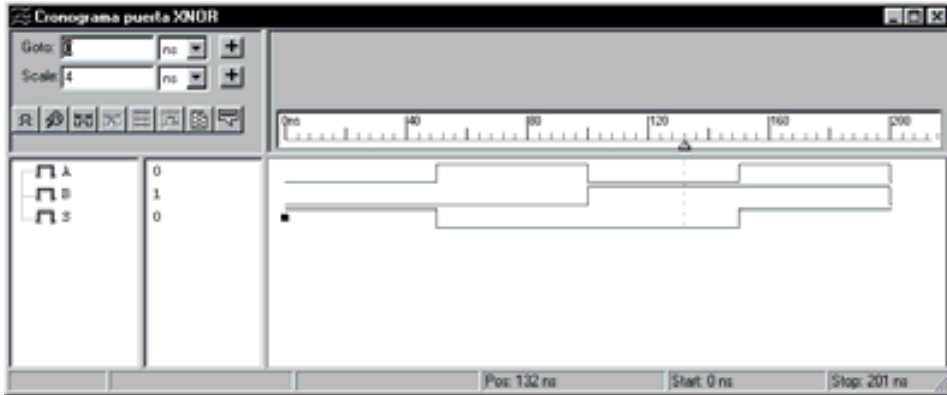


Figura 4.113. Resultado de la simulación, a partir de una descripción en VHDL, de una puerta XNOR

## PROBLEMA RESUELTO 4-14



Simular con el programa EWB el comportamiento de una puerta lógica XNOR de tres entradas obteniendo su tabla de verdad y su cronograma. Verificar que esta puerta se comporta como un generador de paridad impar.

### Solución:

Mediante el convertidor lógico disponible en el programa *Electronics Workbench*, se obtiene la tabla de verdad de una puerta XNOR de tres entradas de dos formas diferentes: introduciendo su expresión algebraica y representando su circuito lógico. Posteriormente, se activará el botón correspondiente según se quiera convertir, de expresión o de circuito lógico a tabla de verdad, como se muestra en la Figura 4.114.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W0\_\_14.ewb**

Mediante el generador de palabras y el analizador lógico disponible en el simulador *Electronics Workbench*, se obtiene el cronograma de una puerta NOR de tres entradas, según se muestra en la Figura 4.115.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W1\_\_14.ewb**

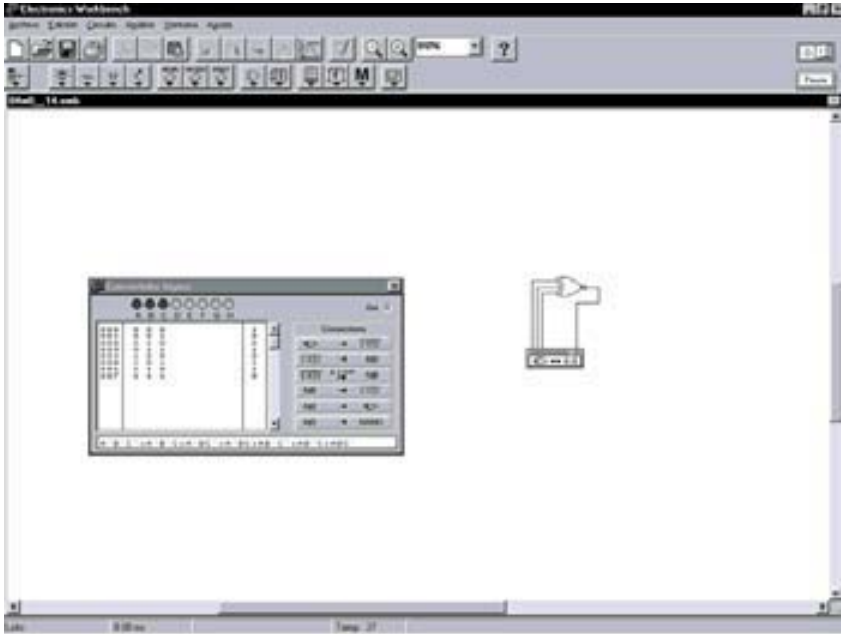


Figura 4.114. Tabla de verdad de una puerta XNOR de tres entradas en EWB

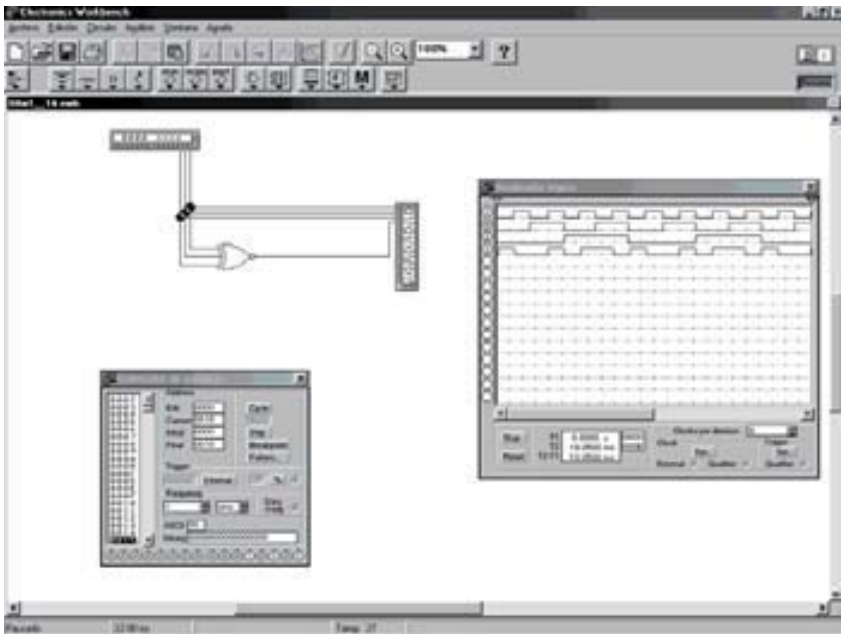


Figura 4.115. Cronograma de una puerta XNOR de tres entradas en EWB



Simular con el programa *OrCAD Demo* el comportamiento de una puerta lógica XNOR de tres entradas y obtener su cronograma. A partir de éste se puede obtener de forma inmediata su tabla de verdad. Verificar que esta puerta se comporta como un generador de paridad impar.

### Solución:

Para simular el comportamiento de este circuito se utiliza el esquema que se muestra en la Figura 4.116. La puerta XNOR de tres entradas se ha construido a partir del circuito integrado 7486 compuesto de puertas XOR de dos entradas y un inversor.

La ruta y el nombre del fichero proyecto que contiene este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\OrCAD9\04R0\_\_14\04R0\_\_14.opj**

En dicha figura se han representado los valores de las señales de entrada (*a*, *b*, *c*) que deben definirse en la aplicación *Stimulus Editor*, como se explicó en el primer problema resuelto de este capítulo.

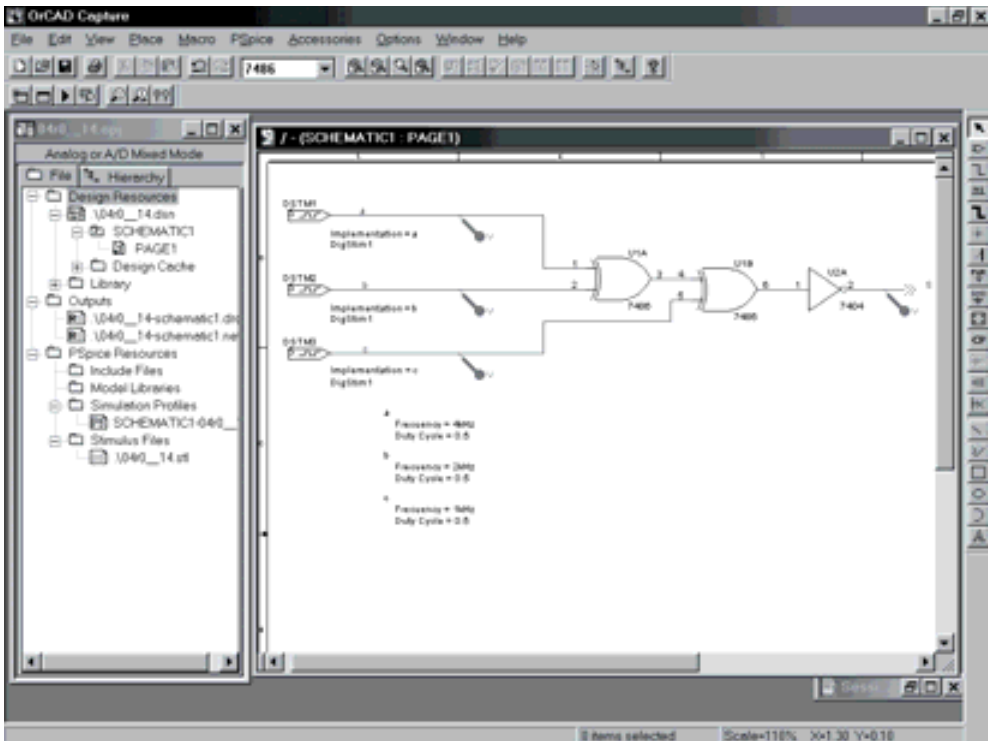


Figura 4.116. Esquema del circuito XNOR de tres entradas en OrCAD Capture

Para estudiar el comportamiento de este circuito se realiza un análisis transitorio (*Transient...*). Se ha definido en el menú *PSpice/Edit Simulation Settings/Analysis* el valor de **2ms** en *Run to time*. También está definido en *PSpice/Edit Simulation Settings/Stimulus* el fichero **04R0\_14.stl** de estímulos.

En la Figura 4.117 se muestran los resultados que se obtienen en la aplicación de visualización de *OrCAD PSpice A/D Demo*, después de realizar la simulación.

En dicha aplicación se ha activado el cursor, que permite ver, al lado del nombre de las señales, el valor que toma cada una de ellas en el instante en el cual se sitúa el mismo. Situando el cursor en cada uno de los puntos de interés, por ejemplo en cada uno de los sucesivos valores que van tomando las señales se puede obtener la tabla de verdad del circuito.

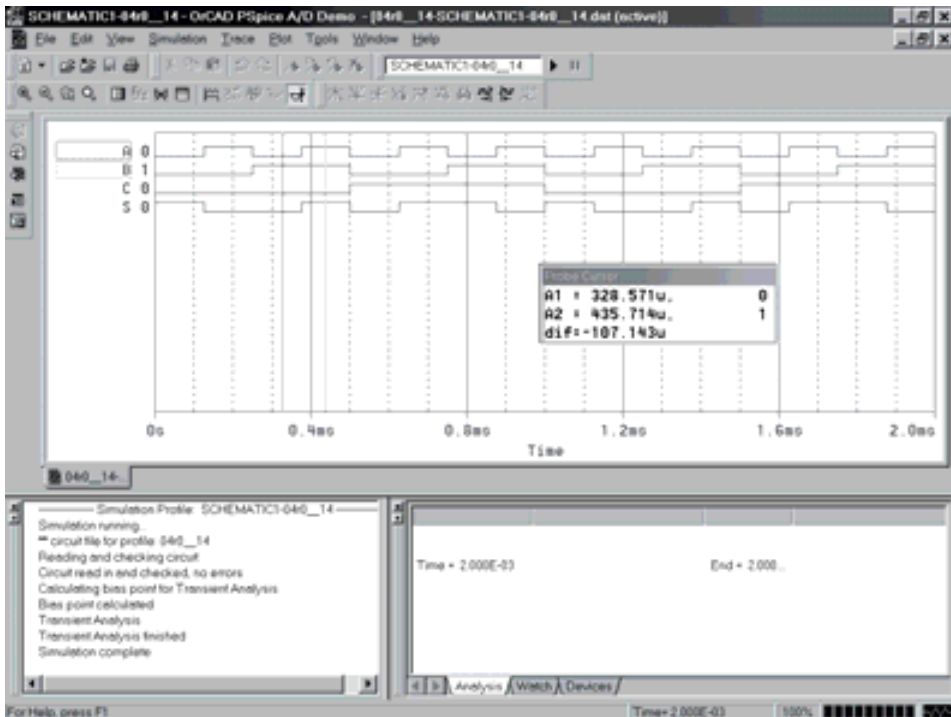


Figura 4.117. Resultados obtenidos en *OrCAD PSpice A/D Demo* para el circuito *XNOR*

## PROBLEMA RESUELTO 4-15



Determinar la expresión algebraica del circuito eléctrico de la Figura 4.118. Simplificarla mediante álgebra de *Boole* y representar este circuito eléctrico simplificado. Simular con el programa EWB ambos circuitos eléctricos, demostrando que son equivalentes.

### Solución:

Teniendo en cuenta lo indicado al estudiar la función producto AND cuya representación eléctrica corresponde a interruptores conectados en serie y que la función suma OR tiene una representación eléctrica con interruptores conectados en paralelo. Y, además considerando que los interruptores abiertos corresponden a variables binarias y los interruptores cerrados a variables binarias negadas (función NOT), se puede deducir la expresión [4.11].

$$S = [(a + \bar{a})bc + \bar{b}](\bar{d} + c) \quad [4.11]$$

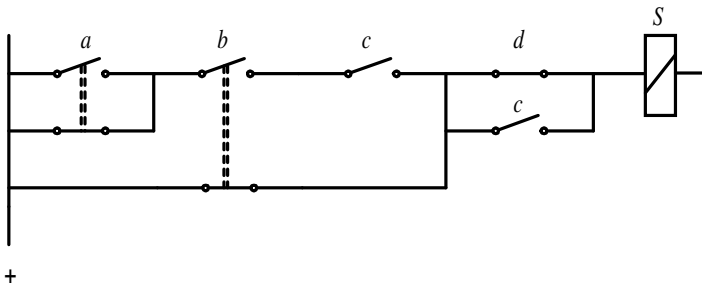


Figura 4.118. Esquema del circuito eléctrico propuesto

Operando, aplicando las propiedades del álgebra de Boole, y en especial la propiedad distributiva, se simplifica  $S$  como puede verse en la expresión [4.12]

$$\begin{aligned} S &= [(a + \bar{a}) \cdot b \cdot c + \bar{b}](\bar{d} + c) = (1 \cdot b \cdot c + \bar{b})(\bar{d} + c) = \\ &= (b \cdot c + \bar{b})(\bar{d} + c) = (b + \bar{b})(c + \bar{b})(\bar{d} + c) = (c + \bar{b})(\bar{d} + c) = \\ &= c + \bar{b} \cdot \bar{d} \end{aligned} \quad [4.12]$$

Por consiguiente, el circuito lógico correspondiente a la expresión simplificada, que se muestra de nuevo en [4.13],

$$S = c + \bar{b} \cdot \bar{d} \quad [4.13]$$

es el representado en la Figura 4.119.

Mediante el programa de simulación *Electronics Workbench*, se comprueba la equivalencia de los dos circuitos de  $S$ , el correspondiente al enunciado y el simplificado. Para ello, en el entorno de simulación correspondiente a la Figura 4.120 se van accionando las distintas combinaciones posibles, es decir, las distintas posiciones de los interruptores, pulsando en el teclado A, B, C y D, comprobando que los pilotos de salida etiquetados como  $S$ , en todo momento tienen el mismo valor binario (encendido/apagado), lo que demuestra la equivalencia y simplificación del circuito eléctrico.

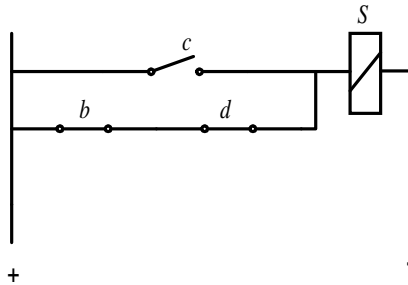


Figura 4.119. Esquema del circuito eléctrico simplificado de  $S$

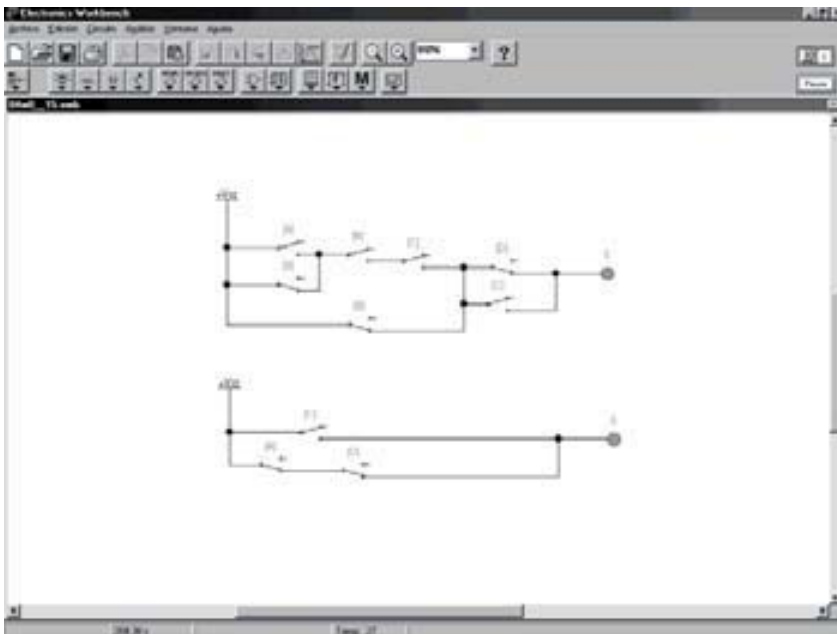


Figura 4.120. Comprobación de la equivalencia de los dos circuitos eléctricos de  $S$



La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap04\Ewb5\04W0\_15.ewb**

## **PROBLEMAS PROPUESTOS**

- 4-16) Determinar la expresión algebraica del circuito eléctrico de la Figura 4.118. Simplificarla mediante álgebra de *Boole* y representar este circuito eléctrico simplificado. Simular con el programa OrCAD Demo ambos circuitos eléctricos, demostrando que son equivalentes.
- 4-17) Comprobar mediante simulación tres funciones diferentes y equivalentes de una puerta XNOR representadas en la expresión [4.10]. Definir los tres circuitos lógicos correspondientes y comprobar que tienen la misma tabla de verdad (la de la función XNOR).
- 4-18) Diseñar un generador de paridad impar de 8 bits de datos más el de paridad utilizando para ello las herramientas de simulación.
- 4-19) Diseñar un control de calidad en una cadena de producción. La salida de cada equipo fabricado en la cadena de producción es comparada con la de un equipo patrón para detectar si son iguales. Esto indicaría que el equipo fabricado es correcto al cumplir las condiciones de funcionamiento. Realizar el diseño de un circuito que realice esta función, utilizando para ello las herramientas de simulación.

## CAPÍTULO 5

# SIMPLIFICACIÓN DE FUNCIONES LÓGICAS

---

---

**Objetivos:**

- Saber aplicar los distintos métodos de simplificación de funciones lógicas para optimizar los diseños, reduciendo el número de componentes o puertas lógicas necesarias en la realización de un sistema digital.

**Contenido:** Métodos de simplificación de funciones lógicas: algebraico, *Karnaugh* y *Quine-McCluskey*. Tratamiento de simplificación en funciones lógicas múltiples.

**Simulación:** Mediante el programa de simulación *Electronics Workbench 5.0* se comprueban los resultados obtenidos con los diferentes métodos de simplificación.

## 5.1 SIMPLIFICACIÓN DE FUNCIONES LÓGICAS



En esta sección se estudian los distintos métodos de simplificación de funciones lógicas: algebraico, *Karnaugh* y *Quine-McCluskey*. También se tratan las funciones lógicas múltiples.

Existe una relación directa entre la complejidad de la red de puertas que constituyen un circuito lógico determinado y la complejidad de su expresión booleana. Por ello, el **objetivo de la simplificación** de un circuito lógico consiste en minimizar su expresión para conseguir su implementación utilizando un número mínimo de puertas lógicas conectadas adecuadamente.

Para **valorar las prestaciones de un diseño digital** se tienen en consideración principalmente dos factores: la velocidad de respuesta y el coste. **La velocidad de respuesta** de un sistema digital disminuye con el retardo que sufre la señal al propagarse por los **niveles** o número de puertas que compone el camino más largo entre las entradas y las salidas del sistema, buscando aquel diseño que tenga el menor número de niveles de puertas.

La reducción de su **coste** se consigue utilizando un número mínimo de puertas, lo que determina una disminución de interconexiones, diseños de circuitos impresos más simples, mayor facilidad de mantenimiento, etc. En la valoración del coste no se incluye los posibles complementos de las variables de entrada que se supone que van a estar disponibles. Sin embargo, sí hay que tener en cuenta, principalmente con componentes discretos, las conexiones de las entradas y las conexiones necesarias entre las puertas del primer nivel y las puertas de salida o de segundo nivel.

El **criterio para comparar el coste** de dos circuitos digitales es: dado dos circuitos digitales se considera de menor coste aquel que tenga menos puertas, y a igual número de puertas el que necesite menos conexiones.

Los métodos de simplificación más empleados se materializan con dos niveles de puertas, si bien no siempre esta topología es la más adecuada ya que los circuitos con más de dos niveles dan más posibilidades de ubicación para una superficie dada, frente a los de dos niveles en los que uno de sus niveles puede crecer en gran medida con respecto al otro y no permitir un buen aprovechamiento de la superficie.

Además, al usar los métodos de simplificación más empleados para dos niveles de puertas, la velocidad de respuesta, para una determinada tecnología, es igual en cualquier diseño que se obtenga, pasando a ser sólo relevante su coste.

No existe un método único de simplificación. Actualmente, se está produciendo una gran evolución en este sentido, debido al avance de sistemas que permiten la síntesis de circuitos lógicos a partir de su descripción, sus expresiones booleanas, tablas de verdad, diagramas de estados, etc. Dicha síntesis se realiza mediante lenguajes de alto nivel de descripción de hardware (como el VHDL), que posibilitan la realización física de cualquier función lógica a partir de sistemas funcionales complejos implementados en circuitos integrados.

Para determinar cuándo una expresión booleana es la más simple de todas las equivalentes a ella, se adopta el **criterio de expresión minimizada o función mínima**. Este criterio establece que una expresión está minimizada cuando, expresada en productos de sumas o en sumas de productos, tenga el mínimo número de términos y el mínimo número de variables en cada término.

Se denomina **minimización de una función** al proceso por el que se obtiene una función mínima.

Cabe destacar, en función de la complejidad o número de variables de la función a simplificar, los siguientes métodos de minimización de una función:

- **Método algebraico.** Consiste en la aplicación analítica de los teoremas y axiomas del álgebra de Boole (principalmente la propiedad distributiva a los términos de la función), con el objetivo de eliminar términos y variables. Tiene el inconveniente de ser poco sistemático, muy subjetivo y por lo tanto no siempre se llega de forma fácil a la expresión minimizada, e incluso a identificarla cuando se obtiene ésta.
- **Método de Karnaugh.** Es un método gráfico y sistemático, muy eficiente en funciones de hasta seis variables.
- **Método de Quine-McCluskey.** Es un método sistemático que utiliza un algoritmo de simplificación que posibilita su programación en un ordenador. Se utiliza, preferentemente, en la simplificación de funciones complejas, con más de seis variables.

### 5.1.1 Método algebraico de simplificación

Este método consiste en buscar dos **términos canónicos adyacentes** de  $n$  variables, es decir, aquellos que sólo se diferencien en el estado de una de sus variables (apareciendo en uno de los términos negada y en el otro sin negar). Al aplicar la propiedad distributiva y los postulados 5 y 2 de *Huntington* (apartado 3.1.1) se simplifica dicha variable, como se representa en la expresión [5.1] y su expresión dual [5.2].

$$a_n \cdot \dots \cdot a_3 \cdot a_2 \cdot a_1 + a_n \cdot \dots \cdot a_3 \cdot a_2 \cdot \overline{a_1} = a_n \cdot \dots \cdot a_3 \cdot a_2 \quad [5.1]$$

y la expresión dual,

$$(a_n + \dots + a_3 + a_2 + a_1)(a_n + \dots + a_3 + a_2 + \overline{a_1}) = a_n + \dots + a_3 + a_2 \quad [5.2]$$

siendo la demostración de la expresión [5.1],

$$\begin{aligned}
 a_n \cdot \dots \cdot a_3 \cdot a_2 \cdot a_1 + a_n \cdot \dots \cdot a_3 \cdot a_2 \cdot \bar{a}_1 &= (a_n \cdot \dots \cdot a_3 \cdot a_2)(a_1 + \bar{a}_1) && \text{(p. distributiva)} \\
 &= (a_n \cdot \dots \cdot a_3 \cdot a_2) \cdot 1 && \text{(postulado 5)} \\
 &= (a_n \cdot \dots \cdot a_3 \cdot a_2) && \text{(postulado 2)}
 \end{aligned}$$

La demostración de la expresión [5.2] se obtiene por dualidad.

La propiedad indicada anteriormente es la base de muchos de los métodos de simplificación. Así, como se verá posteriormente, el método de *Karnaugh* determina, mediante un procedimiento gráfico, los términos canónicos adyacentes, de forma más cómoda y sistemática para ser simplificados; y el método de *Quine-McCluskey* también determina los términos canónicos adyacentes mediante una tabla (algoritmo) que ordena y compara dichos términos.

### Ejemplo:

$$\begin{aligned}
 f &= \bar{c}ba + cba = (\bar{c} + c)ba = 1ba = ba \\
 f &= (c + \bar{b} + a)(\bar{c} + \bar{b} + a) = (c\bar{c}) + \bar{b} + a = 0 + \bar{b} + a = \bar{b} + a
 \end{aligned} \tag{5.3}$$

La primera expresión del ejemplo indica que la suma de dos productos canónicos adyacentes (que difieren en el estado de una de sus variables, en este caso la  $c$ ), se reduce a un único producto en el cual se ha suprimido dicha variable (variable que cambia). La segunda expresión es dual de la primera y por lo tanto el razonamiento es el mismo para el caso de sumas canónicas.

Las expresiones simplificadas obtenidas se denominan no canónicas por no disponer de alguna de las variables.

El siguiente problema resuelto ilustra este procedimiento de simplificación algebraica y sirve para mostrar el concepto de expresión irreducible, mínima y la no unicidad en la minimización.

### PROBLEMA RESUELTO 5-1

Simplificar algebraicamente la función:

$$f(c, b, a) = \sum_3 (1, 2, 3, 4, 5, 6) \tag{5.4}$$

### SOLUCIÓN:

La función propuesta es:

$$f(c, b, a) = \bar{c}\bar{b}a + \bar{c}b\bar{a} + \bar{c}ba + c\bar{b}\bar{a} + c\bar{b}a + cb\bar{a} \quad [5.5]$$

Asociando todos los *minterms* de la función en pares adyacentes (pudiéndose asignar un *minterm* en dos o más asociaciones) y aplicando la expresión [5.1], se produce la simplificación representada en la Tabla 5.1.

Tabla 5.1. Asociación de *minterms* adyacentes y su simplificación

	<i>Minterms</i>		<i>Simplificación</i>
1	$\bar{c}\bar{b}a$	}	$\bar{c}a$
3	$\bar{c}ba$		
2	$\bar{c}b\bar{a}$	}	$\bar{c}b$
3	$\bar{c}ba$		
4	$c\bar{b}\bar{a}$	}	$c\bar{b}$
5	$c\bar{b}a$		
4	$c\bar{b}\bar{a}$	}	$c\bar{a}$
6	$cb\bar{a}$		

La función obtenida es más simple que la inicial, tal como puede verse en la expresión [5.6].

$$f(c, b, a) = \bar{c}a + \bar{c}b + c\bar{b} + c\bar{a} \quad [5.6]$$

Esta expresión es irreducible ya que no hay ningún par de términos adyacentes que permitan la simplificación, y no es mínima ya que se pueden buscar otras asociaciones que, utilizando todos los *minterms*, permitan una mayor reducción, como por ejemplo la representada en la Tabla 5.2.

En este caso se obtiene la función de la expresión [5.7].

$$f(c, b, a) = \bar{c}a + c\bar{b} + b\bar{a} \quad [5.7]$$

Esta función además de irreducible es mínima ya que no existen otras asociaciones de términos que permitan una mayor reducción de variables y/o de términos.

Tabla 5.2. Otra asociación de minterms adyacentes y su simplificación

	<i>Minterms</i>		<i>Simplificación</i>
1	$\bar{c}\bar{b}a$	}	$\bar{c}a$
3	$\bar{c}ba$		
4	$c\bar{b}\bar{a}$	}	$c\bar{b}$
5	$c\bar{b}a$		
2	$\bar{c}b\bar{a}$	}	$b\bar{a}$
6	$cb\bar{a}$		

Existe otra posible asociación que posibilita otra minimización de la función propuesta, dada por la expresión [5.8].

$$f(c, b, a) = \bar{b}a + \bar{c}b + c\bar{a} \quad [5.8]$$

A la vista del Problema resuelto 5-1 se puede concluir que no siempre una expresión simplificada es mínima y que la minimización de una función lógica no tiene por qué ser única.

## PROBLEMA RESUELTO 5-2



Mediante el programa de simulación *Electronics Workbench* se obtiene y se comprueba la minimización de la función propuesta en el Problema resuelto 5-1.

### Solución:

Para realizar la simulación con la aplicación *Electronics Workbench*, se debe colocar (arrastrando) el convertidor lógico, situado en la barra de componentes e instrumentación del banco de instrumentos, en el área de trabajo, como se muestra en la Figura 5.1.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es el que se indica a continuación:

**D:\Ejemplos\Cap05\Ewb5\05W0\_\_01.ewb**

Se introduce en dicho convertidor lógico la tabla de verdad del Problema resuelto 5-1. Para ello, se activan las columnas de las variables de entrada *A*, *B*, *C*, haciendo clic sobre los círculos grises, situados encima de ellas. Los círculos activados pasan a color blanco rellenándose los códigos de las columnas de la tabla de verdad correspondientes a las variables seleccionadas.

Posteriormente, se rellena la columna de salida situada a la derecha poniendo un 1 en aquellos *minterms* que definen la función propuesta, es decir:

$$f(c, b, a) = \bar{c}\bar{b}a + \bar{c}b\bar{a} + \bar{c}ba + c\bar{b}\bar{a} + c\bar{b}a + cb\bar{a} \quad [5.9]$$

Se debe hacer la **observación**, para evitar confusiones, que en este simulador las variables de entrada se representan en mayúscula *A, B, C, ...* y dispuestas en orden inverso en la asignación de pesos con respecto al criterio prefijado en este texto, en el que siempre la variable *a* (en minúscula) es la de menor peso y según sea de mayor orden alfabético la variable tendrá mayor peso.

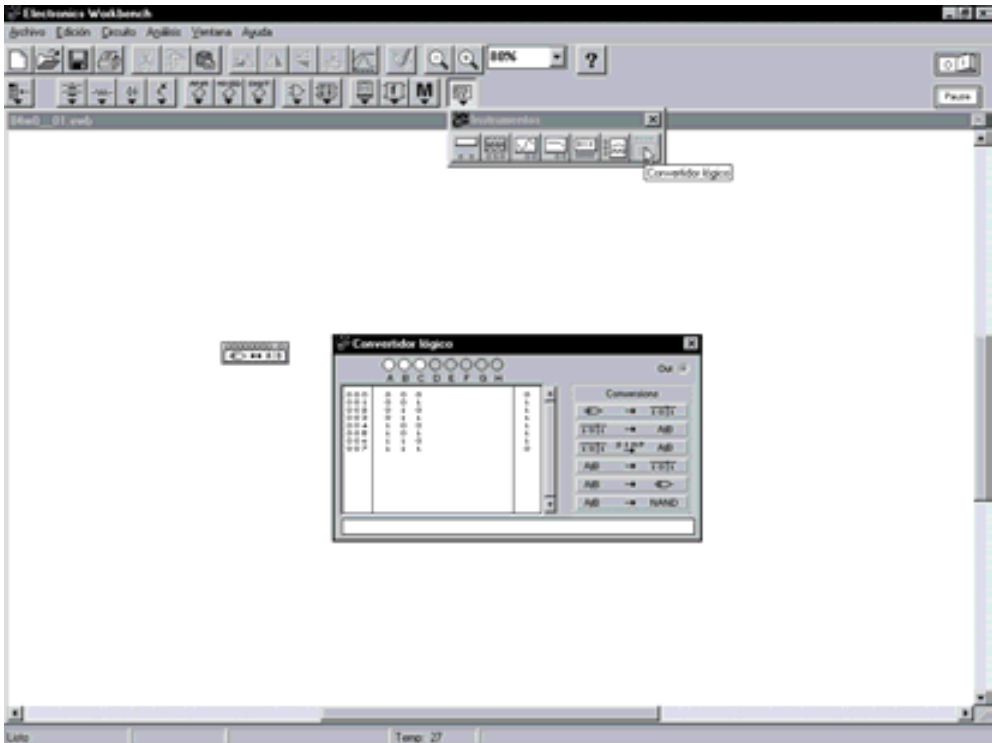


Figura 5.1. Introducción de la tabla de verdad de la función propuesta en el convertidor lógico del programa Electronics Workbench

Una vez completada la tabla de verdad se puede obtener su función canónica, en forma de *minterms*, pulsando la opción del convertidor lógico siguiente.





La expresión así obtenida debe coincidir con la función propuesta, expresión [5.9], como se muestra en la Figura 5.2, teniendo en cuenta que las variables negadas se representan con una tilde a su derecha.

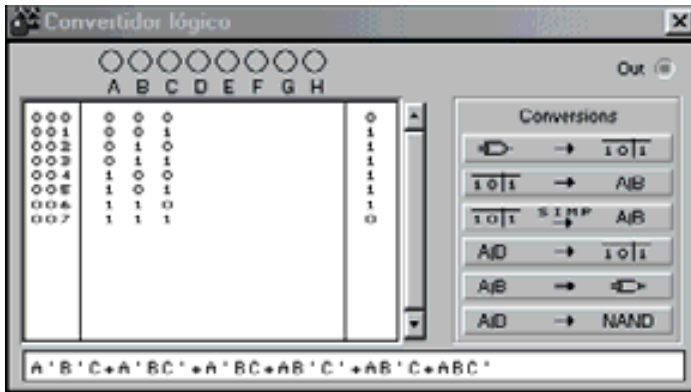


Figura 5.2. Función canónica en forma de minterms correspondiente a la tabla de verdad introducida en el convertidor lógico

Para minimizar una función se pulsa la opción de simplificación del convertidor lógico:



obteniéndose la función propuesta minimizada que coincide con la expresión [5.8], como se muestra en la Figura 5.3.

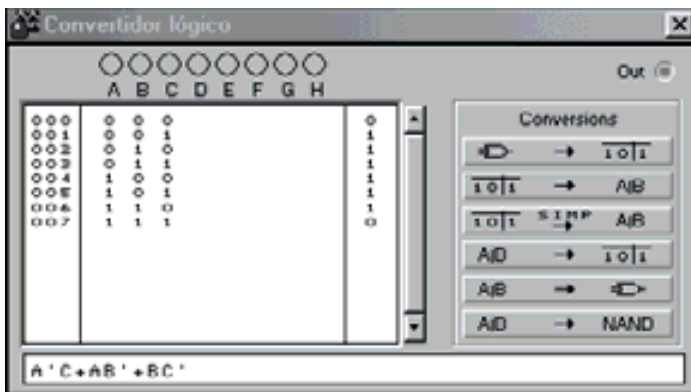


Figura 5.3. Resultado de la minimización de una función mediante simulación

## 5.1.2 Métodos sistemáticos de simplificación

En el apartado de simplificación algebraica se han expuesto los conceptos fundamentales y condiciones a cumplir en la minimización de funciones lógicas. A partir de ello, es más fácil establecer definiciones más rigurosas que sirvan de base a las reglas, procedimientos y propiedades que deben cumplir los métodos sistemáticos de simplificación, que se estudian posteriormente.

### 5.1.2.1 DEFINICIONES PREVIAS. ADYACENCIAS

Las **adyacencias**, a diferencia de los términos canónicos, están formadas por términos productos o sumas en los que no tienen por qué aparecer todas las variables de la función lógica. El **orden de la adyacencia** representa el número de variables que faltan en el término. Así, una adyacencia de orden cero representa a un término canónico.

Un término canónico puede tener una **representación vectorial** indicando en binario el estado de las variables que lo forman. Considerando la posición u orden establecido en las variables de una función  $f(\dots, c, b, a)$  como si fueran coordenadas de un vector, en la representación del término canónico:

- 1) ***minterm***, los ceros designan variables negadas y los unos se corresponden con variables directas o sin negar. Teniendo en cuenta que la posición de los unos (nivel activo alto) definen un peso, el valor decimal del número binario obtenido en la representación vectorial se corresponde con el número o subíndice del término canónico *minterm*.
- 2) ***maxterm***, los unos designan variables negadas y los ceros se corresponden con variables directas o sin negar. Teniendo en cuenta que la posición de los ceros (nivel activo bajo) definen un peso, el valor decimal del número binario obtenido en la representación vectorial se corresponde con el número o subíndice del término canónico *maxterm*.

#### Ejemplo:

Representación en la Tabla 5.3 de varios términos canónicos correspondientes a una tabla de verdad de tres variables (c, b, a) de forma algebraica y vectorial en *minterms* y *maxterms*.

Se define como **índice de un *minterm*** al número de variables sin negar (o número de unos en la representación vectorial) que tiene dicho término canónico. Por tanto, los *minterms* de  $n$  variables se pueden agrupar según su índice en  $n+1$  grupos, desde el índice cero, con el término (0 ... 0), hasta el índice  $n$  con el término (1 ... 1). El **índice de un *maxterm*** se puede obtener por dualidad, definiéndose como el número de variables negadas (o número de ceros en su representación vectorial) que tiene dicho *maxterm*.

Tabla 5.3. Ejemplos de representación algebraica y vectorial de términos canónicos de tres variables ( $c, b, a$ )

Tabla de verdad Representación vectorial $c b a$	Representación Minterms			Representación Maxterms		
	algebraica	Valor decimal, nivel H	$m_i$	algebraica	Valor decimal, nivel L	$M_i$
0 0 0	$\bar{c}\bar{b}\bar{a}$	0	$m_0$	$c + b + a$	7	$M_7$
0 1 1	$\bar{c}ba$	3	$m_2$	$c + \bar{b} + \bar{a}$	4	$M_4$
1 1 1	$cba$	7	$m_3$	$\bar{c} + \bar{b} + \bar{a}$	0	$M_0$

### Ejemplo:

En la Tabla 5.4 se representan, algebraicamente y vectorialmente, los términos canónicos de tres variables ( $c, b, a$ ), así como los correspondientes índices.

Tabla 5.4. Ejemplos de índices de términos canónicos de tres variables ( $c, b, a$ )

$c b a$	Representación Minterms	Índice $m_i$	Representación Maxterms	Índice $M_i$
0 0 0	$\bar{c}\bar{b}\bar{a}$	$m_0$	$c + b + a$	$M_7$
0 1 1	$\bar{c}ba$	$m_3$	$c + \bar{b} + \bar{a}$	$M_4$
1 1 1	$cba$	$m_7$	$\bar{c} + \bar{b} + \bar{a}$	$M_0$

Obsérvese que los índices de un mismo término canónico expresado en *minterm* y en *maxterm* son complementarios a  $n$ , siendo  $n$  el número de variables de la función.

Dados dos términos canónicos con el mismo número de variables  $n$ , se dice que forman una **adyacencia de primer orden** si ambos tienen la misma expresión salvo en una variable, que en uno aparece sin complementar y en el otro complementada.

**Corolario:** Para que dos términos canónicos puedan formar una adyacencia de primer orden es necesario que sus índices difieran en una unidad.

No se debe confundir adyacencia con términos adyacentes. Dos términos canónicos adyacentes forman una adyacencia de primer orden.

Se **representa una adyacencia de primer orden** mediante un término que incluya las  $n-1$  variables que coinciden en los dos términos canónicos (*minterms* o *maxterms*) de origen y con la misma forma de complementación en que aparezcan.

Se puede utilizar la **representación vectorial en adyacencias de primer orden** asignando las mismas coordenadas 0 o 1 que tengan los términos canónicos que la forman en aquellas  $n-1$  posiciones coincidentes, y colocando un guión (-) en la posición en que difieren, también llamada **coordenada vacua**.

**Ejemplo:**

Los *minterms*,

$$d\bar{c}\bar{b}a \quad (1001)$$

$$d\bar{c}ba \quad (1011)$$

cuyas representaciones vectoriales se indican a su derecha, forman una adyacencia de primer orden (con una coordenada vacua) de la forma,

$$d\bar{c}a \quad (10-1)$$

Dos adyacencias de primer orden forman una **adyacencia de segundo orden** si una de ellas se puede obtener a partir de la otra complementando una de sus variables (difieren sólo en una de sus variables y coincide la posición de la única coordenada vacua que tienen).

Se **representa una adyacencia de segundo orden** mediante un término que incluya las  $n-2$  variables que coinciden en las dos adyacencias de primer orden de origen y con la misma forma de complementación y coordenada vacua en que aparezcan.

**Ejemplo:**

Las dos adyacencias de primer orden,

$$d\bar{c}a \quad (10-1)$$

$$d\bar{c}\bar{a} \quad (10-0)$$

cuyas representaciones vectoriales se indican a su derecha, forman una adyacencia de segundo orden (con dos coordenadas vacuas) de la forma,

$$d\bar{c} \quad (10--)$$

Se observa cómo una adyacencia de primer orden tiene una coordenada vacua y se minimiza una variable. Una adyacencia de segundo orden tiene dos coordenadas vacuas, por lo que se minimizan dos variables, además puede ser generada mediante dos parejas diferentes de adyacencias de primer orden.

**Ejemplo:**

La adyacencia de segundo orden,

$$d\bar{c} \quad (10--)$$

puede ser generada por la pareja de adyacencias de primer orden,

$$d\bar{c}a \quad (10-1)$$

$$d\bar{c}\bar{a} \quad (10-0)$$

o por la pareja de adyacencias,

$$d\bar{c}\bar{b} \quad (100-)$$

$$d\bar{c}b \quad (101-)$$

Dos adyacencias de segundo orden pueden formar una **adyacencia de tercer orden**, y así sucesivamente. Una adyacencia de tercer orden puede proceder de tres parejas diferentes de adyacencias de segundo orden, en general, una **adyacencia de orden  $n$  puede proceder** de  $n$  parejas diferentes de orden  $n-1$ .

En la **agrupación de adyacencias** se comienza identificando las de orden cero (términos canónicos), posteriormente se van formando adyacencia de orden uno agrupando dos de orden cero, se continúa formando adyacencias de orden dos agrupando dos de orden uno o cuatro de orden cero y así sucesivamente.

Aquellos productos canónicos o adyacencias de orden cero que formen parte de una adyacencia de orden superior pueden ser sustituidos por ella, simplificándose así la función booleana.

La **simplificación sistemática** se basa en encontrar el menor número de adyacencias de mayor orden que cubran a todos los términos (*minterms* o *maxterms*).

### 5.1.2.2 DEFINICIONES Y PROPIEDADES DE FUNCIONES MÍNIMAS

En este apartado se definen las propiedades que debe satisfacer una función para que además de ser irreducible sea mínima. Para no extender la explicación, sólo se hace referencia a *minterms*, pudiéndose razonar un desarrollo paralelo para *maxterms* por dualidad.

Sean  $f_1(a_n, \dots, a_2, a_1)$  y  $f_2(a_n, \dots, a_2, a_1)$  dos funciones lógicas con las mismas variables. Se dice que una función  $f_1$  **cubre a**  $f_2$ , representándose ( $f_1 \supseteq f_2$ ), si siempre que  $f_2$  toma el valor 1 la función  $f_1$  también vale 1, es decir,  $f_1$  cubre a  $f_2$  si la tabla de verdad de  $f_1$  tienen al menos todos los unos que tiene la de  $f_2$ . De esta definición se

deduce que si  $f_1$  cubre a  $f_2$ , y además  $f_2$  cubre a  $f_1$ , entonces ambas funciones tendrán la misma tabla de verdad y por lo tanto serán **funciones equivalentes**.

**Corolario:** Una función lógica y su función simplificada deben tener la misma tabla de la verdad y por lo tanto serán equivalentes.

**Ejemplo:**

Las funciones:

$$\begin{aligned} f_1(c, b, a) &= \bar{b}\bar{a} + b\bar{a} + ca \\ f_2(c, b, a) &= \bar{a} + c \end{aligned} \quad [5.10]$$

cumplen que  $f_1$  cubre a  $f_2$  y además que  $f_2$  cubre a  $f_1$ , como se comprueba al examinar sus tablas de verdad representadas en la Tabla 5.5. De ello se deduce que ambas funciones son equivalentes.

Tabla 5.5. Tabla de verdad de  $f_1$  y  $f_2$

$c b a$	$f_1$	$f_2$
0 0 0	1	1
0 0 1	0	0
0 1 0	1	1
0 1 1	0	0
1 0 0	1	1
1 0 1	1	1
1 1 0	1	1
1 1 1	1	1

Sean  $f_1(a_n, \dots, a_2, a_1)$  una función Booleana y  $f_3(a_n, \dots, a_2, a_1)$  un producto de variables booleanas. Si  $f_1$  cubre a  $f_3$  se dice que  $f_3$  es un **implicado de  $f_1$**  o que  $f_3$  implica a  $f_1$ , expresándose de la forma  $f_3 \rightarrow f_1$ .

**Ejemplo:**

Las funciones:

$$\begin{aligned} f_1(d, c, b, a) &= d\bar{b}\bar{a} + b\bar{a} + ca \\ f_3(d, c, b, a) &= cb\bar{a} \end{aligned} \quad [5.11]$$

cumplen que  $f_1$  es una función Booleana y  $f_3$  es un producto de variables booleanas. Además, si  $f_3$  toma el valor 1 para  $c = 1$ ,  $b = 1$  y  $a = 0$ , la función  $f_1$  también vale 1 para esos mismos valores de las variables, pues su segundo término producto es igual a 1. Por lo que se cumple que  $f_3$  implica a  $f_1$ .

Como consecuencia de lo anterior se deduce que todos los *minterms* de una función son implicados de ella.

Se define como **implicado primo de  $f$**  a un producto de variables booleanas que cumpla que cualquier subproducto de él no es implicado de  $f$ . Los implicados primos de  $f$  son también las adyacencias que se pueden formar en  $f$  que no estén incluidas en ninguna otra de mayor orden.

### Ejemplo:

La función:

$$f_1(d, c, b, a) = d\bar{b}\bar{a} + b\bar{a} + ca \quad [5.12]$$

tiene como uno de los implicados primos al producto  $ca$ , dado que este producto implica a  $f_1$  y sus subproductos  $a$  y  $c$  no son implicados de  $f_1$ .

**Teorema de expresión irreducible.** Si  $f$  es una función irreducible y está expresada como sumas de productos, entonces está formada por la unión de sus implicados primos o adyacencias de mayor orden que se pueden formar en  $f$ .

### Demostración:

Suponiendo que se tiene una función  $f$  irreducible, la función  $f$  tiene que estar formada por la unión de sus implicados primos, porque si  $f$  contiene un término producto que no es implicado primo, se puede obtener un subproducto de ella con menos variables que implique a  $f$  en contra de la hipótesis del teorema.

Cualquier función mínima de una función  $f$  ha de incluir sólo implicados primos, pero cualquier simplificación de  $f$  que incluya sólo implicados primos no tiene por qué ser mínima, pues ésta debe tener el menor número de implicados primos que cubran a la función  $f$ .

Un **implicado primo esencial** de  $f$ , o también llamada **adyacencia esencial**, es aquel/la que cubre por lo menos a un *minterm* de la función  $f$  que no cubre ningún otro implicado primo o adyacencia de mayor orden. Debido a que todos los *minterms* de la función  $f$  deben ser cubiertos, los implicados primos esenciales (adyacencias esenciales) deberán pertenecer a la expresión irreducible equivalente a  $f$ , y por esta misma razón, los implicados primos (adyacencias de mayor orden) cubiertos por los implicados primos esenciales (adyacencias esenciales) no pertenecerán a la expresión irreducible.

### 5.1.2.3 MÉTODO SISTEMÁTICO DE SIMPLIFICACIÓN

Los pasos generales que se deben realizar en la aplicación de un método sistemático de simplificación son los siguientes:

- 1) Obtención del conjunto de implicados primos (adyacencias de mayor orden).
- 2) Determinación de los implicados primos esenciales (adyacencias esenciales), que forman parte de la expresión mínima  $f_2$ .
- 3) Eliminar de la relación de implicados primos (adyacencias de mayor orden) aquellos que son cubiertos por implicados primos esenciales (adyacencias esenciales).
- 4) Se concluye observando si la expresión  $f_2$  obtenida en el paso 2) cubre a todos los *minterms* de  $f_1$ . En este caso  $f_2$  es la expresión mínima y además única. En caso contrario se deben añadir a  $f_2$  los implicados primos o adyacencias (no esenciales) necesarios para cubrir a todos los *minterms* de  $f_1$ , teniendo en cuenta, en todo momento, que el total de implicados primos (adyacencias de mayor orden) añadidos y su tamaño sean mínimos.

### 5.1.3 Método de Karnaugh

Este método fue enunciado en 1952 por *E. W. Veitch* y modificado, presentando la forma actual de mapa, por el ingeniero de IBM *Maurice Karnaugh* en 1953.

El mapa de *Karnaugh* es otra forma de representar la tabla de verdad de un sistema lógico, presentando la ventaja de ofrecer una disposición espacial apropiada para la simplificación.

El mapa de *Karnaugh* **está formado** por una matriz de cuadros, en los que cada uno de ellos representa a un término canónico. Su disposición es tal, que los cuadros contiguos entre sí, horizontal y verticalmente (no en diagonal) representan a **términos canónicos adyacentes** (términos que difieren sólo en un bit) o lo que es lo mismo forman una adyacencia de primer orden pudiéndose simplificar una variable.

En cada fila y columna de la matriz de cuadros se representa, con ceros y unos, los valores que toman las variables. En cada cuadro se representa el valor (0 o 1) que toma la función para cada término canónico.

#### Ejemplo:

Sea una función de dos variables representada en *minterms*,

$$f(b, a) = \sum_2 (0, 1, 3) = \bar{b}\bar{a} + \bar{b}a + ba \quad [5.13]$$



Su tabla de verdad se representa en la Tabla 5.6, y su mapa de *Karnaugh* es el representado en la Figura 5.4.

Tabla 5.6. Tabla de verdad de la función *f*

<i>b a</i>	<i>f</i>	<i>m<sub>i</sub>/M<sub>i</sub></i>
0 0	1	<i>m<sub>0</sub></i>
0 1	1	<i>m<sub>1</sub></i>
1 0	0	<i>m<sub>2</sub></i>
1 1	1	<i>m<sub>3</sub></i>

Obsérvese, en la Figura 5.4, la identificación de cada *minterm* en función de los valores que toman las variables representadas en cada fila y en cada columna.

Usualmente se simplifica la representación de los *minterms* dejando en blanco las casillas que tienen el valor 0.

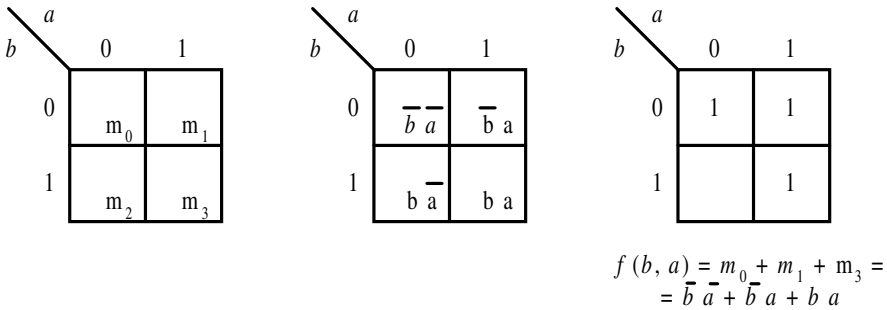


Figura 5.4. Mapa de *Karnaugh* de dos variables en forma de *minterms*

La misma función anterior expresada en *maxterms* viene dada por la expresión [5.14].

$$f_1(b, a) = \prod_2 M(i) = \bar{b} + a \tag{5.14}$$

La tabla de verdad es la misma que la expresada anteriormente en la Tabla 5.6 y su mapa de *Karnaugh* se muestra en la Figura 5.5.

En caso, usualmente se simplifica la representación de los *maxterms* dejando en blanco las casillas que tienen el valor 1.

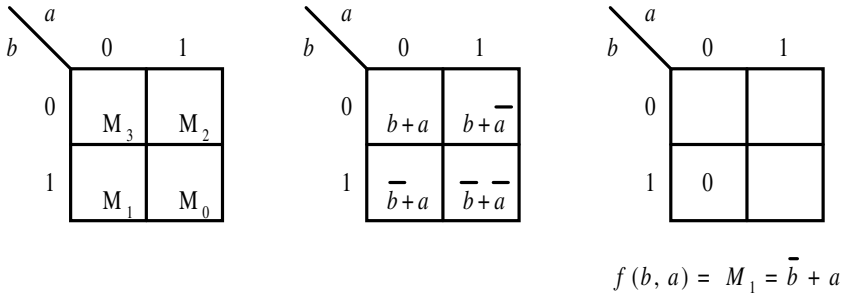


Figura 5.5. Mapa de Karnaugh de dos variables en forma de maxterms

La configuración de los mapas de *Karnaugh* puede tener distintas representaciones, aunque siempre debe cumplir que sus cuadros contiguos sean términos canónicos adyacentes. En el presente libro se ha optado por el siguiente **criterio de representación del mapa de Karnaugh**:

- Se divide el número de variables  $n$  en dos grupos iguales, uno se asociará a las filas  $f$  y el otro a las columnas  $c$ , siendo  $n = f + c$  y cumpliéndose que  $f = c$  en el caso de un número par de variables. En caso de que el número de variables sea impar, en este texto se ha optado por asignar una variable más a las columnas. El mapa estará formado por  $2^f$  filas y  $2^c$  columnas.
- Se utiliza el código Gray para asignar el valor a cada variable en su disposición de filas y columnas, asegurando de esta forma la adyacencia de los términos entre cuadros contiguos, por ser continuo el código Gray.

Para la correcta identificación de los términos canónicos asociados a cada cuadro es necesario establecer un peso a cada variable. Como ya se ha indicado, en este texto, se asigna normalmente a la variable  $a$  el menor peso y según sea de mayor orden alfabético la variable, ésta tendrá mayor peso. También se puede indicar el peso de cualquier variable según el orden en el que se representen en la notación  $f(\dots, \delta, \chi, \beta, \alpha)$ , siendo los pesos de estas variables  $(\dots, 8, 4, 2, 1)$  respectivamente.

- En cada cuadro se indica, en el vértice inferior derecho, el número del *minterm/maxterm* que representa.
- En las filas y columnas, los intervalos en los que la variable toma el valor 1 se representa con una raya con el símbolo de la variable.

Cada mapa de *Karnaugh* se puede considerar descompuesto, con respecto a cada variable, en dos mitades. En una de las mitades la variable vale 0 y en la otra mitad del mapa la variable vale 1, como se aprecia en la representación de los mapas de *Karnaugh* de la Figura 5.6 y siguientes.

Con el criterio indicado se representan los mapas de *Karnaugh* de dos, tres y cuatro variables, en las siguientes figuras.

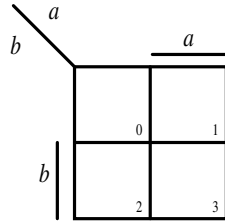


Figura 5.6. Mapa de Karnaugh de dos variables

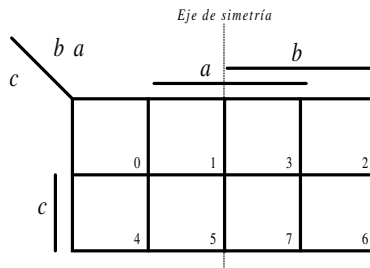


Figura 5.7. Mapa de Karnaugh de tres variables

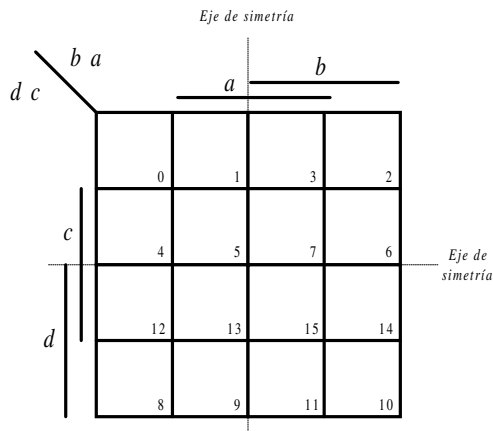


Figura 5.8. Mapa de Karnaugh de cuatro variables

En los mapas de Karnaugh de  $n$  variables, cada cuadro (término canónico) es adyacente a  $n$  cuadros (términos canónicos), es decir, un cuadro de un mapa de dos

variables sólo tiene dos cuadros adyacentes, cada cuadro de tres variables tiene tres cuadros adyacentes y así sucesivamente.

Para localizar los **cuadros adyacentes a un cuadro determinado**, además de cumplirlo los cuadros horizontales y verticales vecinos, también son adyacentes los situados simétricamente respecto a un eje horizontal o vertical que divida el mapa en dos mitades iguales. De ello se deduce y se puede comprobar que los cuadros pertenecientes a las filas/columnas extremas son adyacentes. Esta última observación será más útil y necesaria cuando se trabaje con mapas de cinco y seis variables.

El **procedimiento** a seguir para la minimización de funciones mediante mapas de *Karnaugh* es el siguiente:

- 1) Se dibuja el correspondiente mapa de *Karnaugh* según el número de variables que tenga la función.
- 2) Según como esté representada la función canónica a simplificar, en *minterms* o *maxterms*, se escribirá un **1** en los cuadros correspondientes a los *minterms* de la función o un **0** en el caso de los *maxterms*. Como sólo se requiere simplificar la función, o por *minterms*, o por *maxterms*, normalmente se elige aquella representación que tenga menor número de términos canónicos.
- 3) Para obtener una minimización de la función se deben elegir correctamente las adyacencias, o lo que es lo mismo, cumplir las siguientes reglas que aseguren la **agrupación correcta y óptima** de los términos (*minterms* o *maxterms*) de la función:
  - a) Para funciones con  $n$  variables, se formarán las adyacencias agrupando unos o ceros (según se esté simplificando respectivamente por *minterms* o *maxterms*) en potencia de dos, es decir  $2^m$  términos, donde  $0 \leq m \leq n$ , pudiendo ser las agrupaciones de  $(1, 2, 4, \dots, 2^m, \dots, 2^n)$  términos, formándose adyacencias de orden  $m$ , y por tanto  $m$  es el número de variables que se simplifican. En resumen, cada grupo de  $2^m$  términos o cuadros corresponde a una adyacencia de orden  $m$ .

La simplificación será máxima cuando se definan el mínimo número de adyacencias de mayor orden o, lo que es lo mismo, el **menor número de grupos con el mayor número de términos** en cada uno de ellos.
  - b) Para formar una adyacencia de orden  $m$  se debe cumplir que cada uno de los  $2^m$  cuadros incluidos en un grupo deben ser adyacentes a otros  $m$  cuadros del mismo grupo. Esta condición se cumple únicamente cuando los cuadrados que forman el grupo tengan una **disposición cuadrada o rectangular**. La forma de representar a un grupo de unos o ceros, es abarcándolos mediante una curva cerrada.
  - c) Las adyacencias deben cubrir a todos los términos de la función. Todos los términos unos o ceros (según se simplifique por *minterms* o *maxterms*) deben pertenecer a un grupo, es decir, no se pueden dejar términos de simplificación sueltos.

- d) Se pueden incluir en un grupo términos, que ya pertenecen a otros, con el fin de formar grupos lo más grandes posibles o, lo que es lo mismo, adyacencias de mayor orden.
- e) Aquellas adyacencias que son las únicas que pueden cubrir a un término canónico se denominan **adyacencias esenciales** (o implicado primo esencial). Dado que la función simplificada debe ser equivalente a la función de partida, todos los términos canónicos deben estar cubiertos por las adyacencias, de lo que se deduce que las adyacencias esenciales deben pertenecer a la función simplificada. En el mapa de *Karnaugh* las adyacencias esenciales se identifican por aquella agrupación posible con mayor número de términos que sea la única que puede abarcar a un término canónico o cuadro.
- f) Se deben eliminar grupos en los que todos sus términos pertenecen a otros grupos, es decir, que no contengan al menos un término que sólo pertenezca a dicho grupo.

El **procedimiento** para satisfacer las reglas anteriores es comenzar buscando **grupos de un término** (cubierto por adyacencias de orden cero que no permiten simplificación) que, en caso de haberlos, lo formarán aquellos términos que no son adyacentes con ningún otro. Posteriormente, se buscarán **grupos de dos términos** (cubiertos por adyacencias de orden uno que permiten simplificar una variable), comprobando que no puedan formar grupos de cuatro. Se buscarán **grupos de cuatro términos** (cubiertos por adyacencias de orden dos que permiten simplificar dos variables) que no puedan formar grupos de ocho. Así sucesivamente hasta agrupar a todos los términos (*minterms* o *maxterms*) del mapa.

- 4) Cada grupo señalado da lugar a una adyacencia o **término simplificado** en el que se ha eliminado la variable o variables cuyo valor es 1 en la mitad de los cuadros del grupo y 0 en la otra mitad. Esto tiene como justificación lo expuesto sobre formación y representación de adyacencias.
  - a) Si se realiza la **simplificación de una función canónica dada en *minterms***, las variables que en todo el grupo permanecen como 1 no se simplifican y se representan sin negar. Si las variables en todo el grupo permanecen como 0 no se simplifican y se representan negadas.

Todas las variables de un grupo no simplificadas se multiplican entre sí y los términos simplificados correspondientes a cada grupo se suman entre sí. Es decir, la función simplificada será una suma de productos con tantos sumandos como grupos se hayan formado en el mapa y en cada producto correspondiente a un grupo de  $2^m$  términos se habrán simplificado  $m$  variables.

- b) Si se realiza la **simplificación a una función canónica dada en *maxterms***, las variables que en todo el grupo permanecen como 1 no se

simplifican y se representan negadas. Si las variables en todo el grupo permanecen como 0 no se simplifican y se representan sin negar.

Todas las variables de un grupo no simplificadas se suman entre sí y los términos simplificados correspondientes a cada grupo se multiplican entre sí. Es decir, la función simplificada será un producto de sumas con tantos factores como grupos se hayan formado en el mapa y en cada suma correspondiente a un grupo de  $2^m$  términos se habrán simplificado  $m$  variables.

**Observación:** En general, para la realización práctica de un circuito con el mínimo número de puertas, se aconseja realizar la simplificación en *minterms* y en *maxterms*, para ver cuál es la función simplificada que resulta más sencilla.

### PROBLEMA RESUELTO 5-3



Utilizando el programa de simulación *Electronics Workbench*, comprobar el resultado de la simplificación y obtener el circuito lógico con puertas NAND para el enunciado siguiente.

Simplificar mediante el método de *Karnaugh* para obtener el circuito lógico más eficiente que cumpla la función:

$$f(c, b, a) = \sum_3 (1, 2, 3, 4, 5) \quad [5.15]$$

#### Solución:

Se aplica el procedimiento descrito anteriormente para la minimización de funciones mediante mapas de *Karnaugh*. Primero se simplifica la función canónica dada en *minterms*, después se simplifica en *maxterms* y por último se elige, entre ellas, la solución del circuito lógico más eficiente para ser implementado con el menor número de puertas.

La función propuesta expresada en *maxterms* (apartado 5.1.4.4) es:

$$f(c, b, a) = \prod_3 (0, 1, 7) \quad [5.16]$$

- Se dibuja el mapa de *Karnaugh* de tres variables, como el representado anteriormente en la Figura 5.7.
- Se simplifica la función canónica expresada en *minterms/maxterms*, por lo que se escribe un **1** en los cuadros correspondientes a los *minterms* o un **0** en el resto de los cuadros (que corresponden a los *maxterms*) del mapa, como se muestra en la Figura 5.9

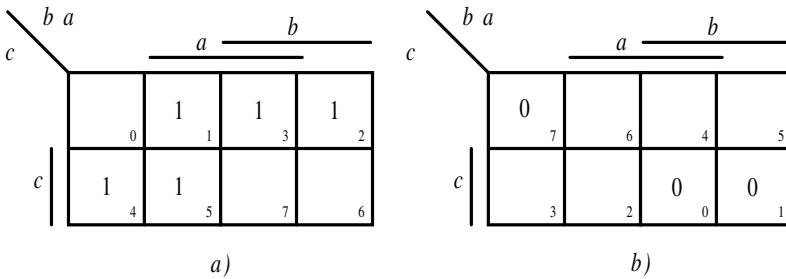


Figura 5.9. Mapa de Karnaugh de la función propuesta en: a) minterms, b) maxterms

- Se realiza una agrupación correcta y óptima de los términos *minterms/maxterms* de la función para obtener su minimización. Se agrupan los unos/ceros en potencia de dos, formando el **menor número de grupos con el mayor número de términos**, abarcándolos mediante una curva cerrada con una **disposición cuadrada o rectangular**.

Para ello se comienza buscando **grupos de un término** (cubierto por adyacencias de orden cero que no permiten simplificación), siendo aquellos términos que no son adyacentes con ningún otro. Posteriormente, se buscarán **grupos de dos términos** (cubiertos por adyacencias de orden uno que permiten simplificar una variable), comprobando que no puedan formar grupos de cuatro. Se buscarán **grupos de cuatro términos** (cubiertos por adyacencias de orden dos que permiten simplificar dos variables) que no puedan formar grupos de ocho. Así sucesivamente hasta agrupar a todos los términos *minterms* o *maxterms* del mapa, con lo que se obtienen las agrupaciones que se muestran en la Figura 5.10.

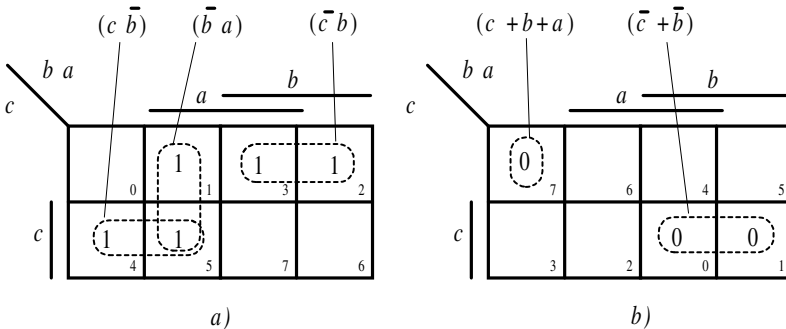


Figura 5.10. Agrupación de los términos: a) minterms, b) maxterms de la función propuesta

Las funciones simplificadas para los dos casos *minterms* y *maxterms* son:

$$f(c, b, a) = c\bar{b} + \bar{b}a + \bar{c}b$$

$$f(c, b, a) = (c + b + a)(\bar{c} + \bar{b})$$
[5.17]

Introduciendo la tabla de verdad en el programa de simulación y seleccionando, en el convertidor lógico, la conversión de tabla de verdad a expresión booleana simplificada, cuyo icono es el siguiente:



se obtiene la expresión simplificada.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap05\Ewb5\05W0\_\_02.ewb**

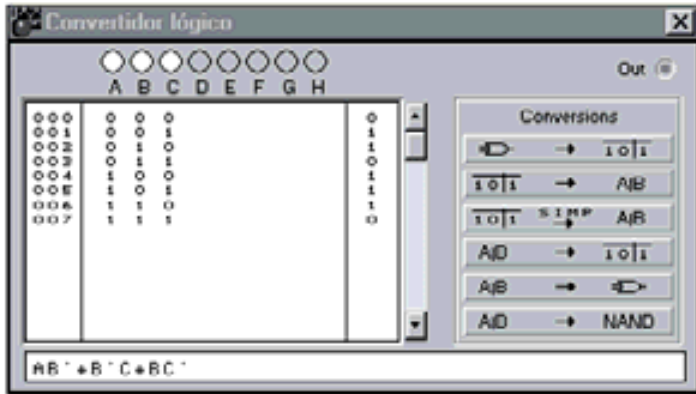


Figura 5.11. Comprobación de la minimización de la función propuesta mediante el programa de simulación Electronics Workbench

Seleccionando, en el convertidor lógico, la conversión de expresión booleana a esquema del circuito, cuyo icono es el siguiente:



se obtiene el circuito lógico de la Figura 5.12.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap05\Ewb5\05W1\_\_02.ewb**



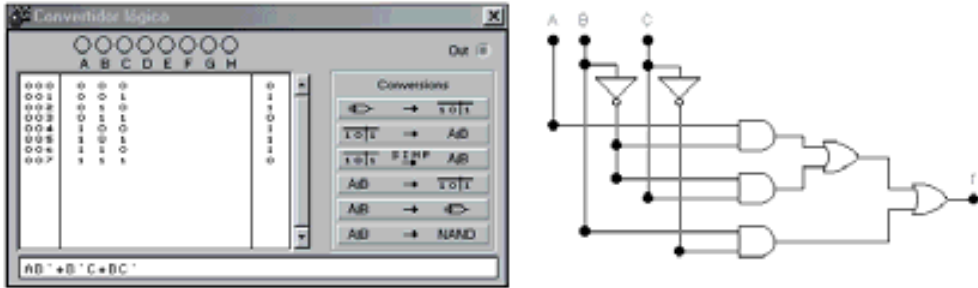


Figura 5.12. Circuito lógico de la expresión canónica en minterms simplificada

Introduciendo la expresión simplificada en *maxterms* en el programa de simulación, y seleccionando la conversión de expresión booleana simplificada a circuito lógico, se obtiene el circuito lógico que se muestra en la Figura 5.13.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap05\Ewb5\05W2\_\_02.ewb**

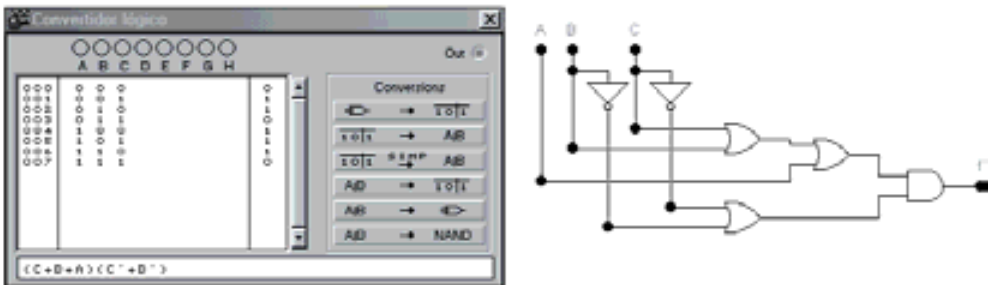


Figura 5.13. Circuito lógico de la expresión canónica en maxterms simplificada

Comparando los dos circuitos anteriores se aprecia que el correspondiente a la expresión canónica en *maxterms* simplificada tiene una puerta menos.

El circuito lógico con puertas NAND correspondientes a la expresión canónica en *minterms* simplificada, se obtiene en el convertidor lógico mediante la conversión de la expresión booleana a puertas NAND, cuyo icono es,



siendo el resultado el representado en la Figura 5.14.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap05\Ewb5\05W3\_\_02.ewb**

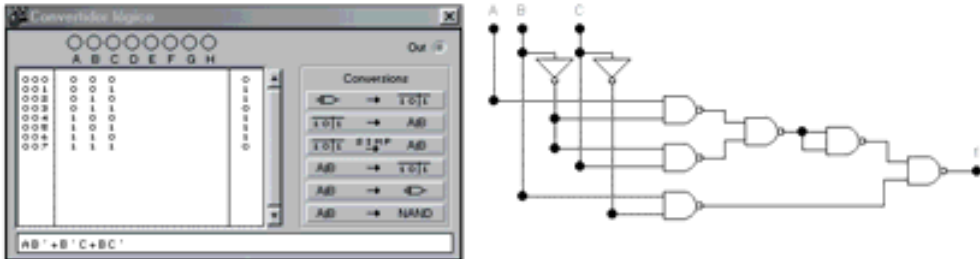


Figura 5.14. Circuito lógico con puertas NAND de la expresión canónica en minterms simplificada

De igual forma, el circuito lógico con puertas NAND correspondientes a la expresión canónica en *maxterms* simplificada, se obtiene en el convertidor lógico mediante la opción de conversión de expresión booleana a puertas NAND, obteniéndose el circuito de la Figura 5.15.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es el que se indica a continuación:

**D:\Ejemplos\Cap05\Ewb5\05W4\_\_02.ewb**

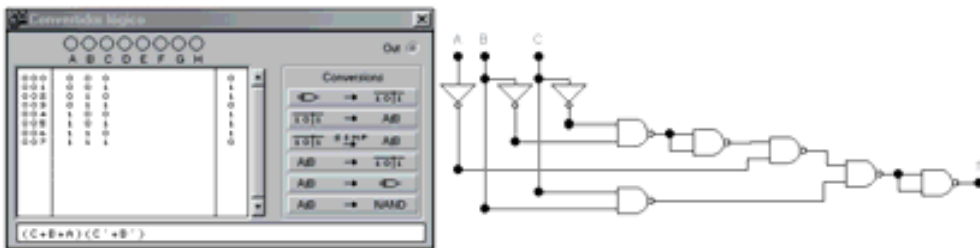


Figura 5.15. Circuito lógico con puertas NAND de la expresión canónica en maxterms simplificada

**PROBLEMA RESUELTO 5-4**



Simplificar por unos y ceros, mediante el método de *Karnaugh*, la siguiente función lógica de cuatro variables y posteriormente comprobar el resultado de la minimización mediante el programa de simulación *Electronics Workbench*.

$$f(d, c, b, a) = \sum_4 (0, 1, 2, 5, 7, 8, 9, 10, 13, 15) \tag{5.18}$$

**Solución:**

El problema no tiene solución única. Como se muestra en la Figura 5.16 existen dos adyacencias esenciales y éstas no cubren a todos los términos canónicos, pudiéndose abarcar al resto de términos de dos formas diferentes.

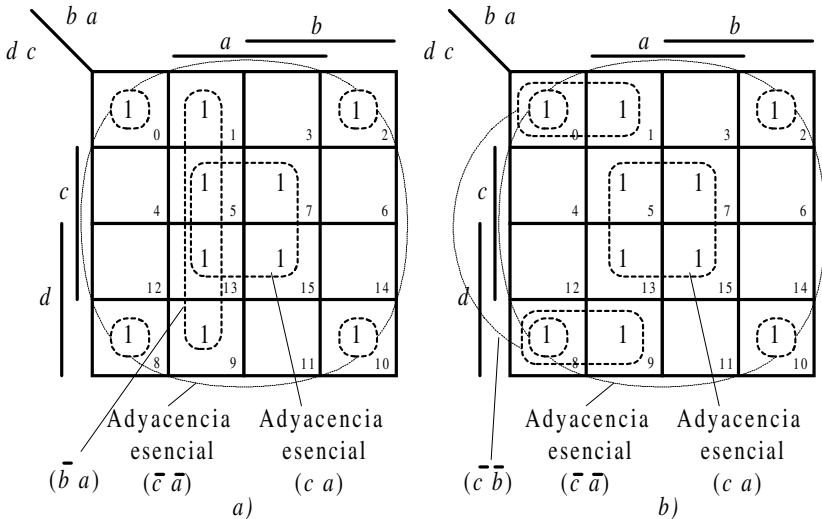


Figura 5.16. Dos posibles agrupamientos o soluciones al problema: a) y b)

De los grupos o adyacencias posibles se obtienen las dos soluciones distintas siguientes. Obsérvese que la función simplificada no es única y que las adyacencias esenciales pertenecen a dicha función simplificada.

$$f_1(d, c, b, a) = \bar{b}a + \bar{c}\bar{a} + ca \tag{5.19}$$

$$f_2(d, c, b, a) = \bar{c}\bar{b} + \bar{c}\bar{a} + ca$$

Representando en el mapa de *Karnaugh* los ceros de la función, se puede obtener la expresión en *maxterms*. Dicha expresión está formada por los *maxterms* cuyo valor decimal corresponde con el código binario de sus variables, considerando el nivel activo (con peso) a los ceros.

Por ejemplo, en la Figura 5.17, el cero situado más arriba en el mapa de *Karnaugh* tiene sus variables (*d, c, b, a*) igual a (0011). Considerando el peso (8421) correspondiente a los ceros, su valor decimal es 12, y por lo tanto representa el *maxterms*  $M_{12}$ .

Siguiendo este procedimiento con el resto de los ceros del mapa, se obtiene:

$$f(d, c, b, a) = \prod_4 (1, 3, 4, 9, 11, 12) \tag{5.20}$$

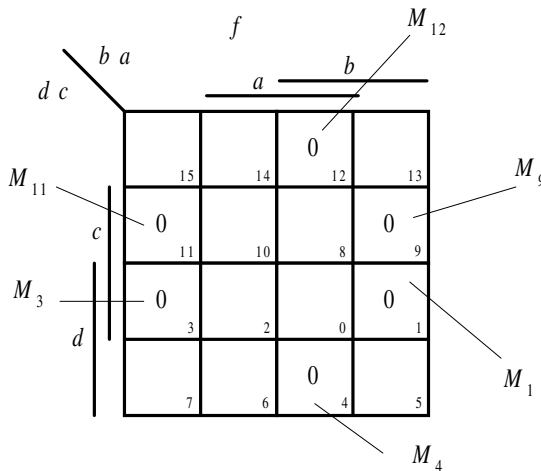


Figura 5.17. Representación de *maxterms* o ceros en el mapa de *Karnaugh*

El mismo resultado se obtiene algebraicamente (apartado 5.1.4.4) al considerar que *f*, expresada en *minterms*, representa a los términos cuya salida vale 1, por lo que  $\bar{f}$  representa a los *minterms* que no son 1, es decir, los términos que faltan en *f*, que son 0.

$$\overline{f(d, c, b, a)} = \sum_4 (3, 4, 6, 11, 12, 14)$$

Considerando que  $\bar{m}_i = M_{2^n - i - 1}$ , se obtiene la expresión,

$$\begin{aligned}
 f &= \overline{\overline{\overline{\overline{f(d,c,b,a)}}}} = \overline{\sum_4(3,4,6,11,12,14)} = \\
 &= \overline{m_3 + m_4 + m_6 + m_{11} + m_{12} + m_{14}} = \\
 &= \overline{m_3 \cdot m_4 \cdot m_6 \cdot m_{11} \cdot m_{12} \cdot m_{14}} = \\
 &= M_{12} \cdot M_{11} \cdot M_9 \cdot M_4 \cdot M_3 \cdot M_1
 \end{aligned}
 \tag{5.21}$$

y finalmente ordenando los términos se obtiene el resultado,

$$f(d,c,b,a) = \prod_4(1,3,4,9,11,12) \tag{5.22}$$

el cual coincide con el calculado anteriormente en la expresión [5.20].

Simplificando por ceros mediante el método de *Karnaugh*, según se muestra en la Figura 5.18, se obtiene el resultado:

$$f_3 = (\bar{c} + a)(c + \bar{b} + \bar{a}) \tag{5.23}$$

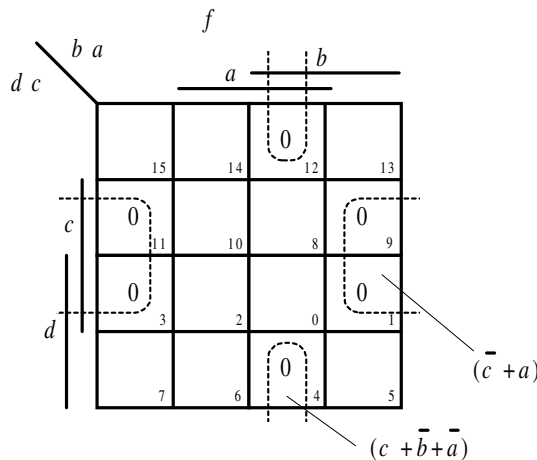


Figura 5.18. Simplificación por ceros por el método de Karnaugh

Mediante el programa de simulación *Electronics Workbench* se puede comprobar el resultado de la simplificación. Para ello se introducen en el convertidor lógico los datos del enunciado: definiendo la tabla de verdad o la expresión canónica.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap05\Ewb5\05W0\_\_03.ewb**

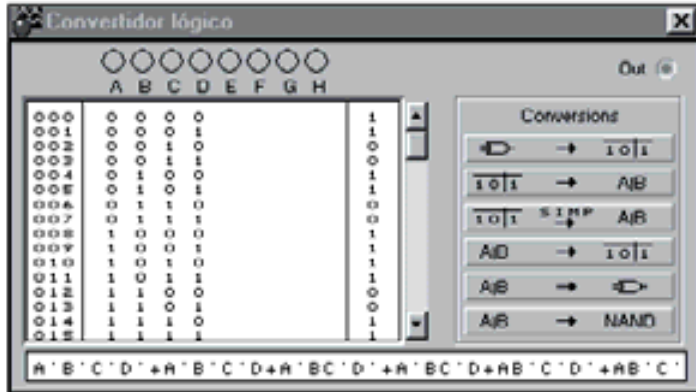


Figura 5.19. Introducción de la función lógica a simplificar en el convertidor lógico

Seleccionando, en el convertidor lógico, la conversión de tabla de verdad a expresión booleana simplificada y, posteriormente, seleccionando la conversión de expresión booleana a circuito lógico, cuyos iconos son respectivamente,



se obtiene una de las expresiones simplificadas, la función  $f_2$  y su circuito lógico, como se muestra en la Figura 5.20.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap05\Ewb5\05W1\_\_03.ewb**

Es obvio, y se puede comprobar en el convertidor lógico, que la función original y las funciones simplificadas  $f_1$ ,  $f_2$  y  $f_3$  tienen la misma tabla de verdad, es decir, son equivalentes.

Se deja al lector que realice el mismo procedimiento realizado en  $f_2$ , introduciendo las expresiones booleanas de  $f_1$  y/o  $f_3$  en el convertidor lógico del simulador *Electronics Workbench*, activando posteriormente la conversión de expresión booleana a tabla de verdad para comprobar que tienen todas ellas la misma tabla de verdad.

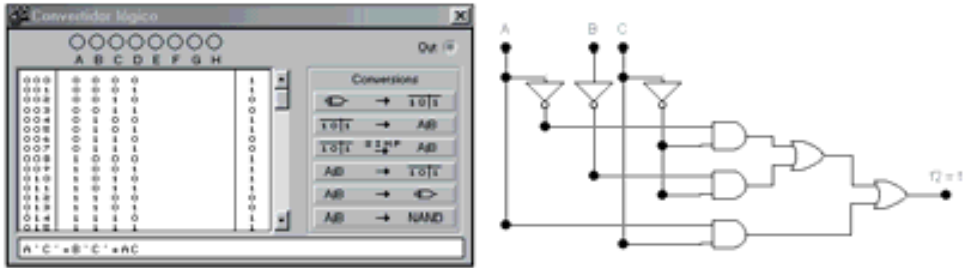


Figura 5.20. Obtención de la función simplificada  $f_2$  y su circuito lógico

También se puede convertir, por el mismo procedimiento indicado en la función  $f_2$ , las otras funciones booleanas equivalentes  $f_1$  y  $f_3$  cuyas expresiones vienen dadas por [5.24] y sus circuitos lógicos en la Figura 5.21.

$$\begin{aligned}
 f_1(d, c, b, a) &= \bar{b}a + \bar{c}\bar{a} + ca \\
 f_3 &= (\bar{c} + a)(c + \bar{b} + \bar{a})
 \end{aligned}
 \tag{5.24}$$

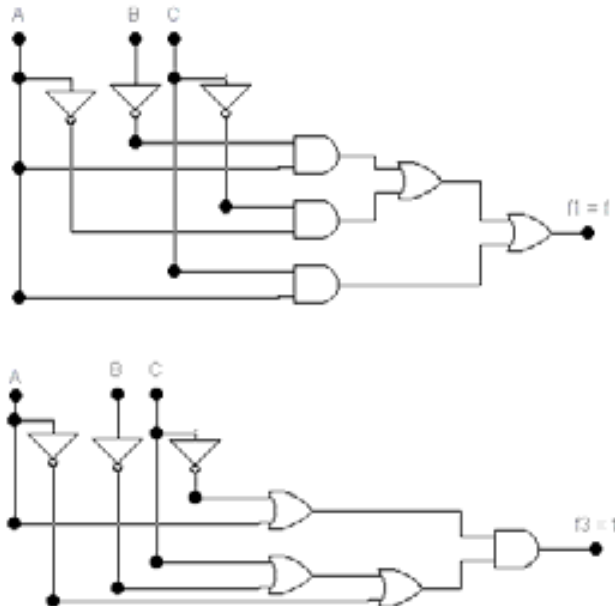


Figura 5.21. Circuitos lógicos de  $f_1$  y  $f_3$  equivalentes a  $f$

**Nota:** La única representación que es única al definir el comportamiento de un sistema digital es su tabla de verdad (invariante). Además, como se aprecia en este ejemplo, a un sistema digital se le puede definir con diferentes expresiones booleanas y/o distintos circuitos lógicos, siendo todos ellos equivalentes.

### PROBLEMA RESUELTO 5-5



Dado el circuito lógico de la Figura 5.22 obtener: su tabla de verdad, su función lógica, su función lógica minimizada por el método de *Karnaugh* y su circuito lógico minimizado con puertas NAND mediante el programa de simulación *Electronics Workbench*.

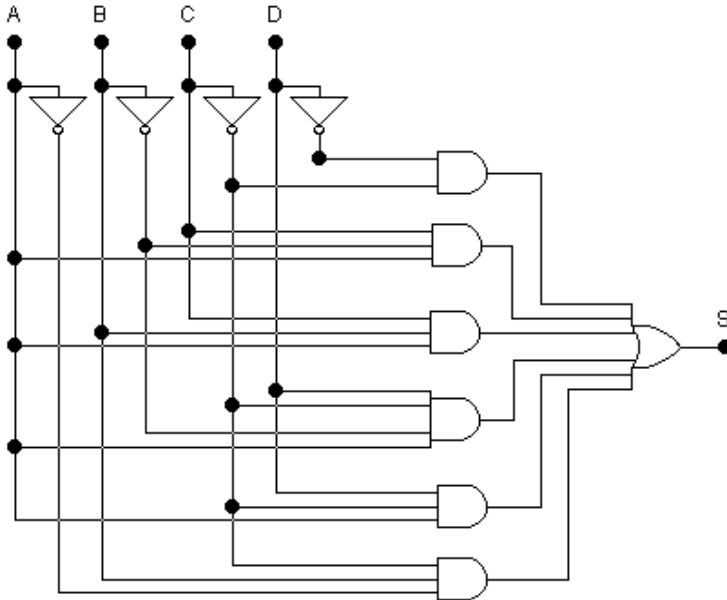


Figura 5.22. Circuito lógico para problema resuelto

### Solución:

A partir del circuito, considerando las operaciones producto (puertas AND) y suma (puertas OR) que se realizan sobre las variables de entrada  $d$ ,  $c$ ,  $b$  y  $a$ , se obtiene la siguiente función lógica en la salida  $S$ .

$$S = \bar{d}\bar{c} + \bar{c}\bar{b}a + cba + d\bar{c}\bar{b}a + d\bar{c}a + d\bar{c}\bar{a}$$



Para obtener su tabla de verdad en el programa de simulación *Electronics Workbench*, se puede proceder de dos formas:

- a) Introduciendo la expresión booleana en el convertidor lógico, Figura 5.23, y seleccionando la conversión de expresión booleana a tabla de verdad, cuyo icono es el siguiente:



- b) Conectando las entradas *d*, *c*, *b* y *a* y salida *S* del circuito al convertidor lógico, Figura 5.24, y seleccionando la conversión de circuito lógico a tabla de verdad, cuyo icono es el siguiente:



Posteriormente, se puede obtener la expresión booleana (en *minterms*) de dicho circuito, Figura 5.24, seleccionando la conversión de tabla de verdad a expresión booleana, cuyo icono es el siguiente:



Para el caso a) la ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap05\Ewb5\05W0\_\_04.ewb**

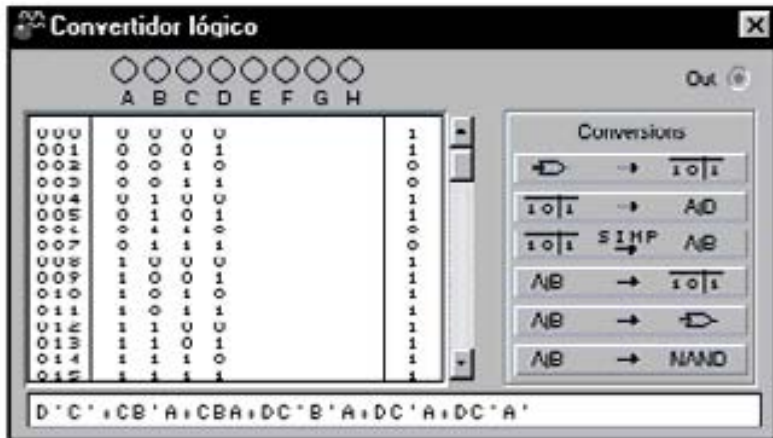


Figura 5.23. Expresión booleana para obtener su tabla de verdad

Para el caso b) la ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap05\Ewb5\05W1\_\_04.ewb**

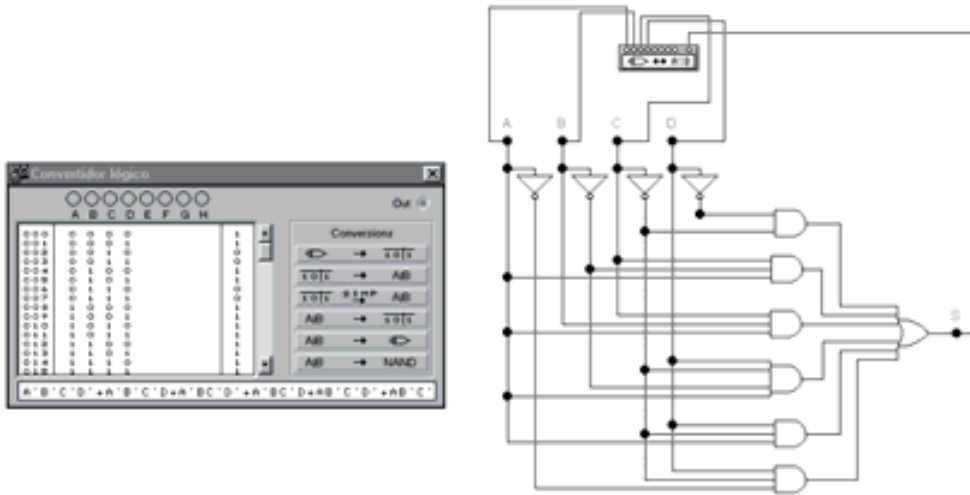
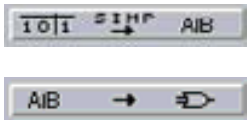


Figura 5.24. Conexión del circuito al convertidor lógico para obtener su tabla de verdad y expresión booleana

Seleccionando, en el convertidor lógico, la conversión de tabla de verdad a expresión booleana simplificada y, posteriormente, seleccionando la conversión de expresión booleana a circuito lógico, cuyos iconos son respectivamente:



se obtiene una de la funciones simplificada y el circuito lógico de la Figura 5.25.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es el que se indica a continuación:

**D:\Ejemplos\Cap05\Ewb5\05W2\_\_04.ewb**

El circuito lógico de la función simplificada *S* con puertas NAND se obtiene en el convertidor lógico seleccionando la conversión de la expresión booleana a puertas NAND, cuyo icono es el siguiente:



siendo el resultado el representado en la Figura 5.26.

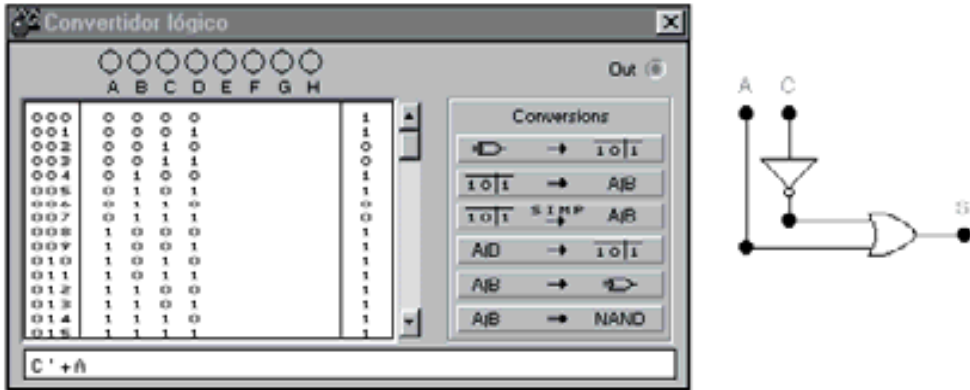


Figura 5.25. Obtención de la función simplificada S y su circuito lógico

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap05\Ewb5\05W3\_\_04.ewb**

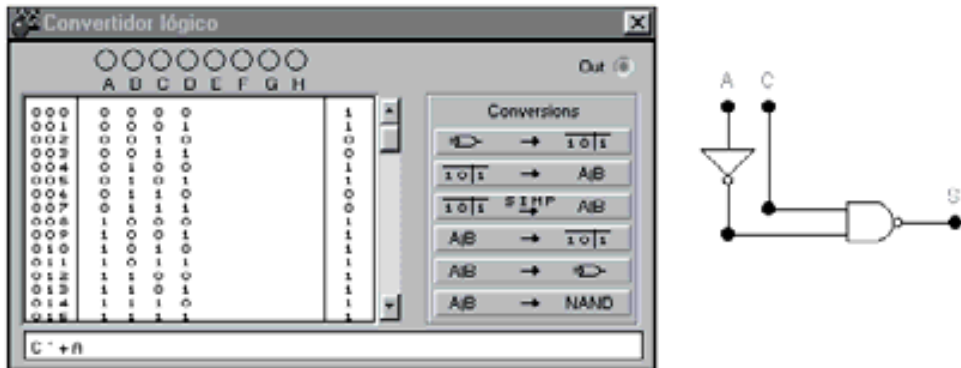


Figura 5.26. Circuito lógico de la función simplificada S con puertas NAND

Se puede también deducir este resultado considerando que una puerta NAND es equivalente a una puerta negativa-OR.

### 5.1.3.1 MAPAS DE KARNAUGH DE MÁS DE CUATRO VARIABLES

El procedimiento descrito del método de *Karnaugh* puede aplicarse también a cinco y seis variables. Para mayor número de variables, siete o más, el método es complicado y poco operativo.

La **formación de mapas de Karnaugh de cinco y seis variables** sigue las mismas reglas generales utilizadas en los mapas de menor número de variables.

Para mapas de cinco variables se asignan dos variables en las filas,  $f = 2$  y tres variables en las columnas,  $c = 3$ , obteniéndose un mapa de  $2^f = 4$  filas y  $2^c = 8$  columnas (Figura 5.27). En el caso de seis variables se asignan tres variables en las filas,  $f = 3$  y tres variables en las columnas,  $c = 3$ , obteniéndose un mapa de  $2^f = 8$  filas y  $2^c = 8$  columnas (Figura 5.28).

Con cinco variables,  $n = 5$ , el mapa tiene  $2^n = 32$  cuadros o términos canónicos (*minterms* o *maxterms*) y se puede considerar formado por dos mapas de cuatro variables unidos en horizontal en el que en uno de ellos existe una variable que siempre vale 0 (mapa izquierdo  $c = 0$ ) y en el otro siempre vale 1 (mapa derecho  $c = 1$ ), siguiendo las mismas reglas de formación de adyacencias o grupos descritos en mapas de cuatro variables. De la misma forma, con seis variables,  $n = 6$ , el mapa tiene  $2^n = 64$  cuadros y se puede considerar formado por dos mapas de cinco variables, o cuatro mapas de cuatro.

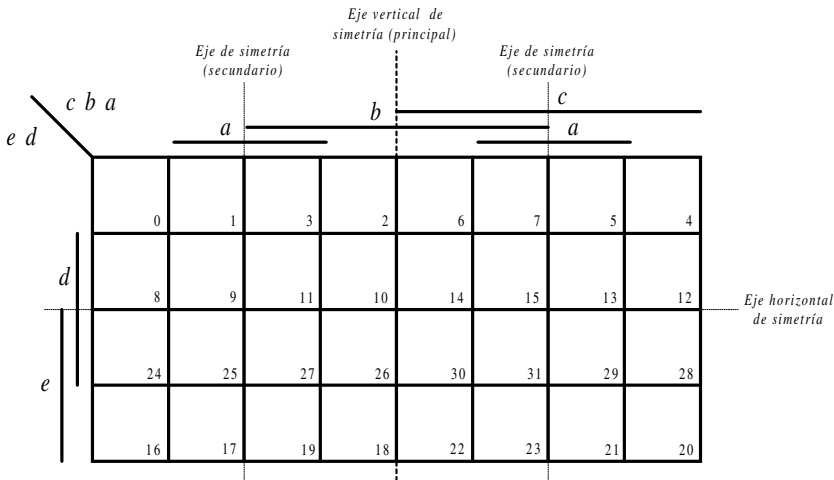


Figura 5.27. Mapa de Karnaugh de cinco variables

Es importante considerar las **simetrías de los mapas de Karnaugh** teniendo en cuenta que cada cuadro (término canónico) puede ser adyacente a  $n$  cuadros, siendo  $n$  el número de variables del mapa. Así, en los mapas de Karnaugh de cinco variables cuatro de los cuadros adyacentes serán los horizontales y verticales vecinos a él (cumpliendo las condiciones de simetría indicadas en mapas de cuatro variables) y el quinto, es el cuadro simétrico respecto al eje vertical de simetría principal.

En la Figura 5.29 se representan varios ejemplos de los posibles cuadros o términos canónicos adyacentes, respecto a uno dado, que se pueden formar en mapas de cinco variables. Cada arco define una adyacencia de primer orden.

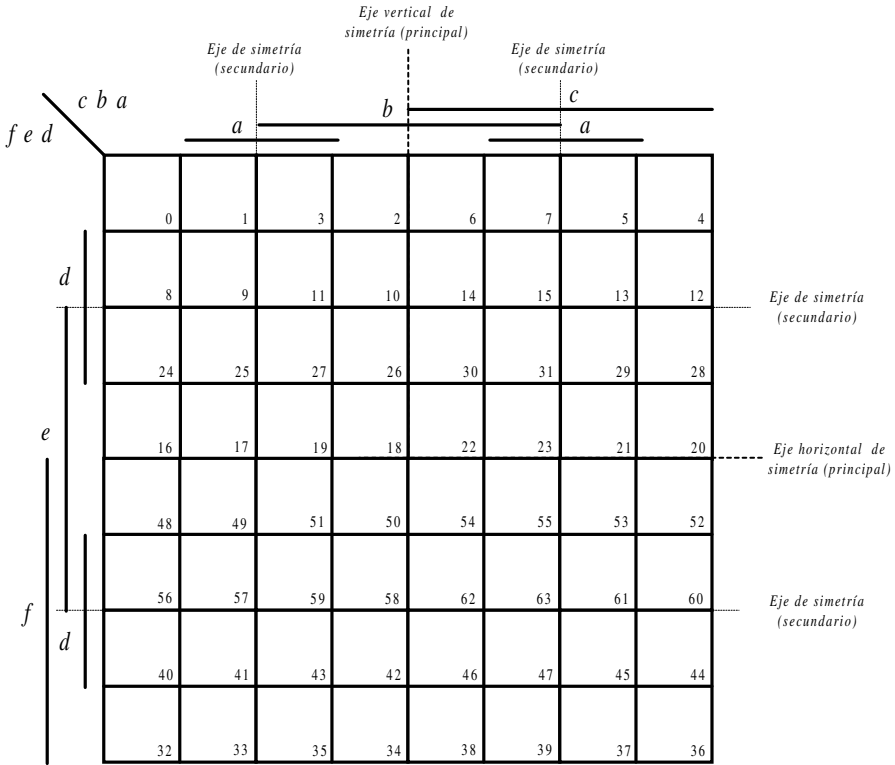


Figura 5.28. Mapa de Karnaugh de seis variables

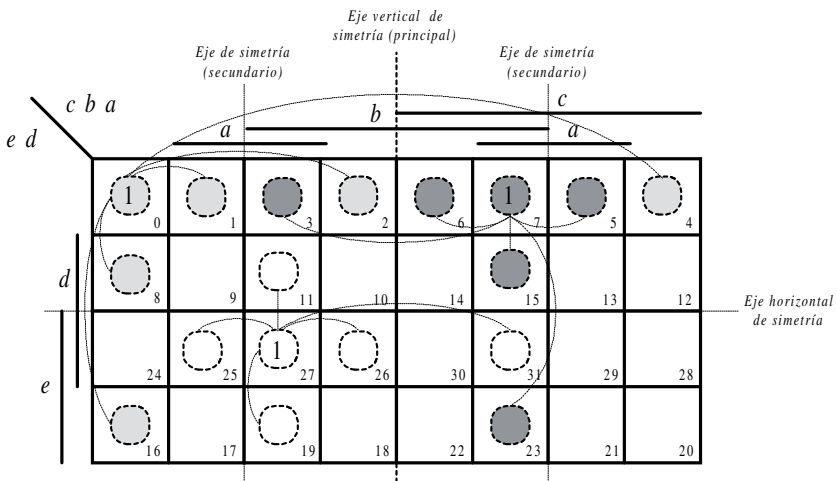


Figura 5.29. Ejemplos términos adyacentes en mapa de Karnaugh de cinco variables

Los grupos estarán formados por  $2^m$  unos, siendo  $0 < m < n$  y  $n$  el número de variables del mapa. Pueden estar todos los unos del mismo grupo a un lado del eje de simetría, o en caso de que estén a ambos lados deberán ser simétricos respecto a dicho eje, es decir, habrá el mismo número de unos a cada lado (Figura 5.30).

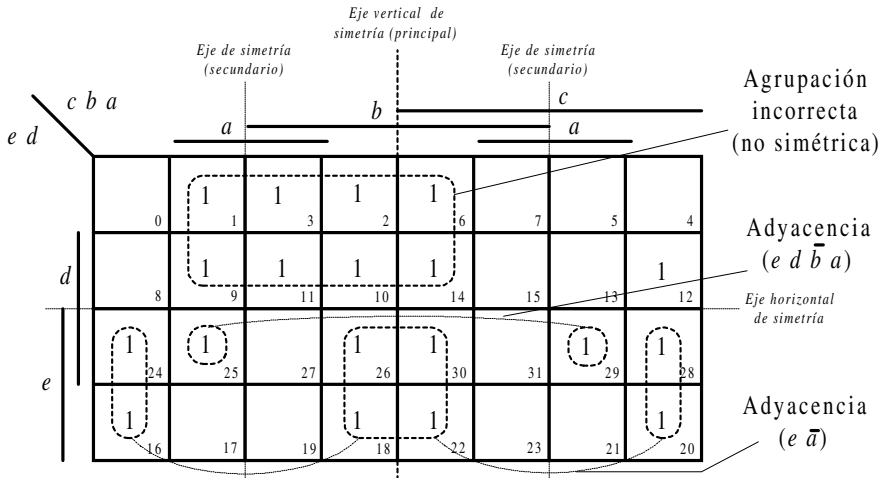


Figura 5.30. Ejemplos de formación de grupos o adyacencias en un mapa de Karnaugh de cinco variables

Lo indicado anteriormente sobre simetrías se hace extensivo a mapas de Karnaugh de seis variables, teniendo en cuenta que estos mapas también tienen un eje horizontal de simetría principal.

**PROBLEMA RESUELTO 5-6**



Simplificar, por el método de Karnaugh, la siguiente función lógica de cinco variables y posteriormente comprobar el resultado de la minimización mediante el programa de simulación Electronics Workbench.

$$S = \sum_5 (0, 2, 4, 5, 6, 7, 9, 11, 12, 13, 15, 18, 19, 21, 23, 25, 27, 29, 31) \quad [5.25]$$

**Solución:**

Cumpliendo con las reglas de simetrías y formación de grupos o adyacencias ya descritas, se han definido las siguientes agrupaciones (Figura 5.31).

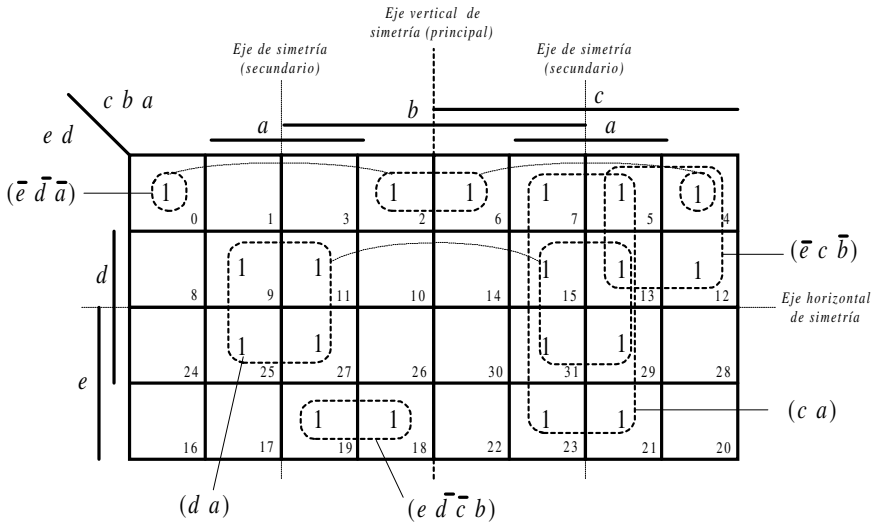


Figura 5.31. Formación de grupos o adyacencias de la función lógica de cinco variables del enunciado

De la Figura 5.31 se obtiene la siguiente función simplificada:

$$S = \bar{e}\bar{d}\bar{a} + d a + c a + e\bar{d}\bar{c}b + \bar{e}c\bar{b} \quad [5.26]$$

Mediante el programa de simulación *Electronics Workbench* se puede comprobar el resultado de la simplificación. Para ello se introducen los datos del enunciado, términos *minterms* que definen la tabla de verdad o su expresión booleana en el convertidor lógico.

Seleccionando la conversión de tabla de verdad a expresión booleana simplificada, se comprueba el resultado de la simplificación obtenida (Figura 5.32).

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap05\Ewb5\05W0\_\_05.ewb**

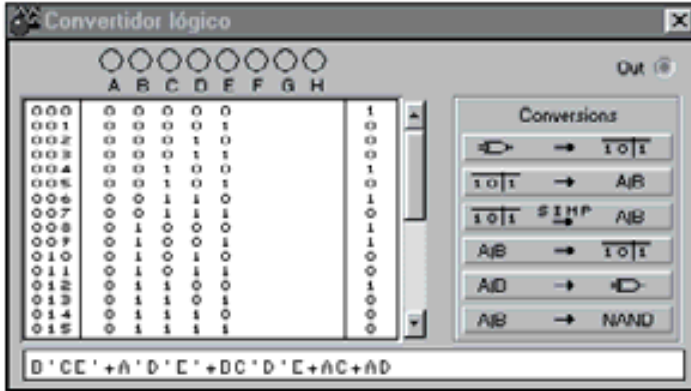


Figura 5.32. Introducción de los datos del enunciado, minterms de la función lógica para obtener su simplificación en el convertidor lógico

**PROBLEMA RESUELTO 5-7**



Simplificar, por el método de *Karnaugh*, la siguiente función lógica de seis variables y posteriormente comprobar el resultado de la minimización mediante el programa de simulación *Electronics Workbench*.

$$S = \sum_6 \left( \begin{matrix} 0,10,11,12,14,15,17,20,21,25,26,27,29,30,31,42,43, \\ 46,47,49,52,56,57,58,59,60,61,62,63 \end{matrix} \right) \quad [5.27]$$

**Solución:**

Cumpliendo con las reglas de simetrías y formación de grupos o adyacencias ya descritas, se han definido las siguientes agrupaciones (Figura 5.33).

A partir de la Figura 5.33 se obtiene la función simplificada dada por [5.28].

$$S = \overline{f} \overline{e} \overline{d} \overline{c} \overline{b} \overline{a} + db + \overline{f} \overline{e} \overline{b} a + \overline{e} \overline{d} \overline{c} \overline{b} a + f \overline{e} d + \overline{e} \overline{c} \overline{b} a + \overline{f} \overline{e} d \overline{c} a \quad [5.28]$$

Mediante el programa de simulación *Electronics Workbench* se puede comprobar el resultado de la simplificación. Para ello, se introducen los datos del enunciado, términos *minterms* que definen la tabla de verdad o su expresión booleana en el convertidor lógico.

Seleccionando la conversión de tabla de verdad a expresión booleana simplificada, se comprueba el resultado de la simplificación obtenida. En la Figura 5.34 se muestra la introducción de los datos del enunciado, *minterms* de la función lógica, para obtener su simplificación en el convertidor lógico.



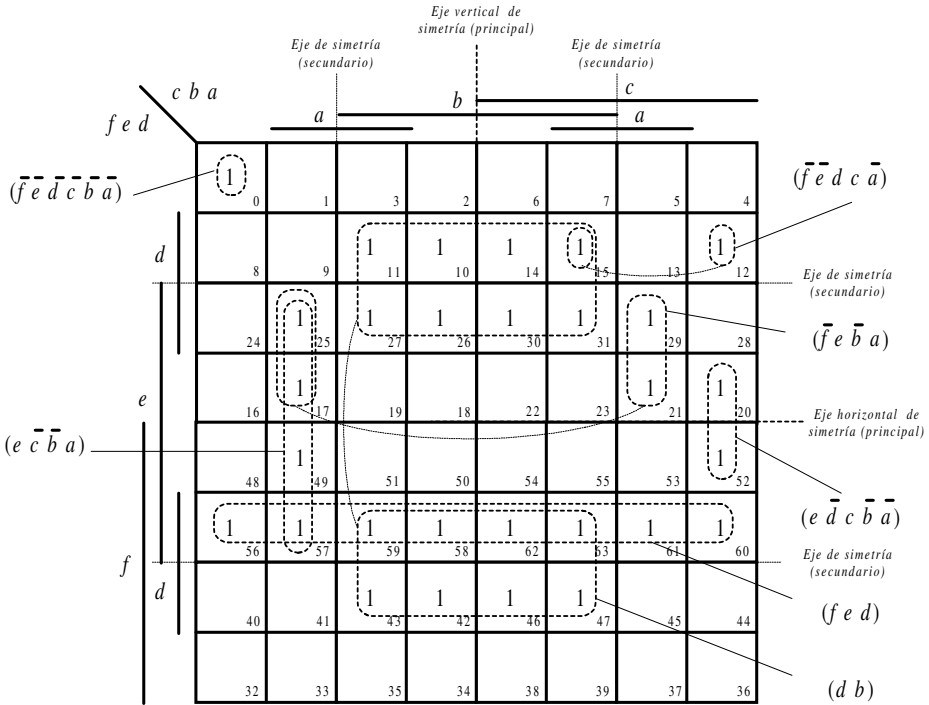


Figura 5.33. Formación de grupos o adyacencias de la función de seis variables

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap05\Ewb5\05W0\_\_06.ewb**

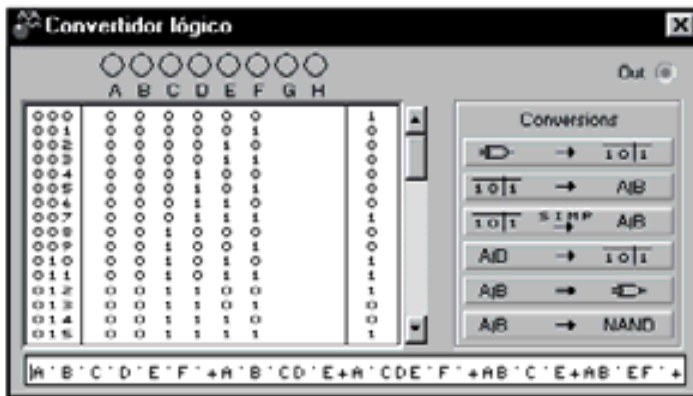


Figura 5.34. Datos del enunciado para simplificar en el convertidor lógico

### 5.1.4 Método de Quine-McCluskey

El método de *Karnaugh* es útil en funciones de menos de seis variables o cuando hay pocos unos o ceros, siendo un método que depende de la habilidad o experiencia de quien lo aplica.

Uno de los métodos sistemáticos que no presenta las limitaciones indicadas en el método de *Karnaugh* y que permite su programación y uso en un sistema informático es el método de *Quine-McCluskey* (en abreviatura *Q-M*).

El método *Q-M* consiste en obtener (de forma sistemática) las adyacencias de órdenes crecientes, hasta llegar a las de mayor orden posible, que son los ya estudiados implicantes primos. Para ello se parte de las adyacencias de orden cero o términos canónicos de la función, posteriormente, con este método y de forma sistemática, se obtienen todas las adyacencias de primer orden, y así hasta obtener todas las de mayor orden posible.

#### 5.1.4.1 MÉTODO DE *Q-M* CON COORDENADAS BINARIAS

Para facilitar la comprensión del método se realiza, simultáneamente con la explicación teórica, una simplificación práctica de una función.

Sea la función dada por [5.29]:

$$f(d, c, b, a) = d(b+a) + \bar{d}c\bar{b}a + \bar{d}c\bar{b}\bar{a} + \bar{d}\bar{c}\bar{a} + \bar{c}\bar{b}\bar{a} + d\bar{c}\bar{a} \quad [5.29]$$

Para aplicar el método *Q-M* se realizan los siguientes **pasos**:

- 1) Dada una función lógica cualquiera se debe obtener su función canónica en *minterms* (o *maxterms*).

Operando en la expresión [5.29] y convirtiendo a expresión canónica en *minterms*, se obtiene la expresión [5.30]:

$$f(d, c, b, a) = \sum_4 (0, 2, 4, 5, 8, 9, 10, 11, 13, 14, 15) \quad [5.30]$$

- 2) Se realiza una primera tabla, denominada de **orden cero**, en la que los términos estén clasificados por su índice (número de unos/ceros que tenga la representación vectorial de cada *minterm/maxterm*). Con ello se favorece la búsqueda de adyacencias superiores (de orden uno), ya que sólo se pueden formar a partir de adyacencias con índices consecutivos, como se puede ver en la Tabla 5.7 a). Ejemplo: (0010: 2) es un término *minterm* de índice uno y el *minterm* (1110: 14) tiene índice tres.
- 3) Para realizar la tabla de **orden uno** se van comparando los términos de índice  $i$  con los de índice consecutivo  $i+1$  de la tabla de orden cero. Para ello se buscan las adyacencias de orden uno (términos que sólo difieren en un bit),

produciéndose una coordenada vacua representada por un guión, como se puede ver en la Tabla 5.7 b). Ejemplo: (0000: 0) y (0010: 2) son dos términos o adyacencia de orden cero y de índice consecutivo que forman la adyacencia de primer orden (00-0: 0,2). Obsérvese que el único término de orden cero es siempre adyacente con todos los términos de orden uno.

Tabla 5.7. Tabla de adyacencias obtenidas por el método Q-M con coordenadas binarias

Índice	Orden 0 minterms	Orden 1 minterms	Orden 2 minterms
0	0000 : 0 ✓	00-0 : 0, 2 ✓	-0-0 : 0, 2, 8, 10 <i>d</i>
	0010 : 2 ✓	0-00 : 0, 4 <i>g</i>	10-- : 8, 9, 10, 11 <i>c</i>
1	0100 : 4 ✓	-000 : 0, 8 ✓	1--1 : 9, 11, 13, 15 <i>b</i>
	1000 : 8 ✓	-010 : 2, 10 ✓	1-1- : 10, 11, 14, 15 <i>a</i>
2	0101 : 5 ✓	010- : 4, 5 <i>f</i>	
	1001 : 9 ✓	100- : 8, 9 ✓	
	1010 : 10 ✓	10-0 : 8, 10 ✓	
3	1011 : 11 ✓	-101 : 5, 13 <i>e</i>	
	1101 : 13 ✓	10-1 : 9, 11 ✓	
	1110 : 14 ✓	1-01 : 9, 13 ✓	
4	1111 : 15 ✓	101- : 10, 11 ✓	
		1-10 : 10, 14 ✓	
		1-11 : 11, 15 ✓	
		11-1 : 13, 15 ✓	
		111- : 14, 15 ✓	

a)

b)

c)

- 4) Para realizar la tabla de **orden dos** se van comparando los términos de índice  $i$  con los de índice consecutivo  $i+1$ . Para ello, en la tabla de orden uno, se buscan las adyacencias de orden dos (términos que sólo difieren en un bit y que además tienen la coordenada vacua o guión en la misma posición), produciéndose una segunda coordenada vacua representada por otro guión, como se puede ver en la Tabla 5.7 c). Por ejemplo: (00-0: 0,2) y (10-0: 8,10)

son dos términos o adyacencias de orden uno y de índice consecutivo que forman la adyacencia de segundo orden (-0-0: 0,2,8,10).

- 5) Por el mismo procedimiento se van obteniendo adyacencias de órdenes crecientes, hasta llegar a las de mayor orden posible.
- 6) Al generarse las adyacencias de órdenes superiores se marcan (por ejemplo con ✓), aquéllas de órdenes inferiores que son cubiertas por ellas. Ejemplo: la adyacencia (00-0: 0,2) cubre a (0000: 0) y al (0010: 2), por lo que estas últimas adyacencias están marcadas con (✓). Con esto se consigue que al final del proceso se tenga, de forma sistemática, la lista de **implicados primos** compuesta por todas las adyacencias de mayor orden posible (adyacencias sin marcar) no cubiertas por ninguna adyacencia de orden superior. Estas adyacencias que definen a los implicados primos se enumeran, por ejemplo con letras, partiendo de las de orden superior hasta la de menor orden. En el ejercicio de la Tabla 5.7 están enumeradas de la *a* a la *g*.

#### 5.1.4.2 MÉTODO DE *Q-M* CON COORDENADAS DECIMALES

El procedimiento de coordenadas binarias suele ser fuente de errores al manejar gran cantidad de unos y ceros, creciendo esta dificultad al aumentar el número de variables o de adyacencias a tratar. Un proceso equivalente, pero más eficaz, para buscar las adyacencias de orden superior, que permitan identificar a los implicados primos, es mediante **coordenadas decimales**.

Los **pasos** a seguir para el método *Q-M* con coordenadas decimales son idénticos a los descritos para coordenadas binarias, salvo que su notación es más compacta al ser decimal.

Los pasos 1) y 2) son los mismos que se han expuesto para coordenadas binarias, excepto que no se escribe la columna de dichas coordenadas binarias, tal como puede verse en la Tabla 5.8 a).

- 3) Para realizar la tabla de **orden uno** se van comparando los términos de índice *i* con los de índice consecutivo *i+1* de la tabla de orden cero, buscando las adyacencias de orden uno (términos que sólo difieren en un bit). Desde el punto de vista de coordenadas decimales esto ocurre cuando al restar el valor decimal del término de índice *i+1* menos el del término de índice *i* dé como resultado positivo una potencia de dos (1, 2, 4, 8, ...). En estos casos en la Tabla 5.8 b) se representan el valor decimal de los términos que intervienen y entre paréntesis el resultado de la resta, la cual define el peso binario o posición de la coordenada vacua. Por ejemplo: el término 0 y el 2 son dos adyacencias de orden cero y de índice consecutivo que al restarse ( $2 - 0 = 2$ ) da como resultado positivo potencia de dos y por lo tanto forman una adyacencia de primer orden representada por 0, 2 coordenada vacua (2). Obsérvese que el único término de orden cero (valor decimal 0) es siempre adyacente con todos los términos de orden uno (valor decimal potencia de 2).

Tabla 5.8. Tabla de adyacencias obtenidas por el método Q-M con coordenadas decimales

Índice	Orden 0 minterms		Orden 1 minterms		Orden 2 minterms	
0	0	✓	0, 2	(2) ✓	0, 2, 8, 10	(2, 8) <i>d</i>
	2	✓	0, 4	(4) <i>g</i>	8, 9, 10, 11	(1, 2) <i>c</i>
1	4	✓	0, 8	(8) ✓	9, 11, 13, 15	(2, 4) <i>b</i>
	8	✓	2, 10	(8) ✓	10, 11, 14, 15	(1, 4) <i>a</i>
	5	✓	4, 5	(1) <i>f</i>		
2	9	✓	8, 9	(1) ✓		
	10	✓	8, 10	(2) ✓		
	11	✓	5, 13	(8) <i>e</i>		
3	13	✓	9, 11	(2) ✓		
	14	✓	9, 13	(4) ✓		
4	15	✓	10, 11	(1) ✓		
			10, 14	(4) ✓		
			11, 15	(4) ✓		
			13, 15	(2) ✓		
			14, 15	(1) ✓		

a)

b)

c)

- 4) Para realizar la tabla de **orden dos** se van restando los términos de índice  $i+1$  de los de índice  $i$ , de la tabla de orden uno, buscando las adyacencias de orden dos (términos que sólo difieren en un bit y que además tienen la coordenada vacua o guión en la misma posición). Desde el punto de vista de coordenadas decimales, Tabla 5.8, esto ocurre cuando en la tabla de orden uno, las dos adyacencias tengan el mismo valor entre paréntesis y al restar cualquiera de los términos situados en la misma posición (derecha o izquierda) de índice  $i+1$  menos el del término de índice  $i$  dé como resultado positivo una potencia de dos (1, 2, 4, 8, ...). En estos casos, en la tabla de orden dos, se representan el valor decimal de los términos que intervienen y entre paréntesis, al ser una adyacencia de orden dos aparecerán dos valores (dos coordenadas vacuas): el que tenían en el orden uno y el resultado de la resta. Por ejemplo: 0,2 y 8,10 son dos términos o adyacencia de orden uno y de índice consecutivo que tienen la misma coordenada vacua (2); al restar los valores de sus izquierdas

( $8 - 0 = 8$ ) o de sus derechas ( $10 - 2 = 8$ ) da el valor 8 que es una potencia de dos, y por tanto, forman la adyacencia de segundo orden 0, 2, 8, 10 con coordenadas vacuas (2, 8).

**Nota:** en lo sucesivo, en adyacencias de orden superior con numerosos valores, sólo será necesario restar los dos valores de cada término situados más a la izquierda.

- 5) Por el mismo procedimiento se van obteniendo adyacencias de órdenes crecientes, hasta llegar a las de mayor orden posible.
- 6) Al generarse las adyacencias de órdenes crecientes se marcan, por ejemplo con (✓), aquéllas de órdenes inferiores que son cubiertas por ellas. Ejemplo: la adyacencia de orden uno 0, 2 cubre a las de orden cero 0 y 2, por lo que estas últimas adyacencias están marcadas con (✓). Con esto se consigue que al final del proceso se tenga, de forma sistemática, la lista de **implicados primos** compuesta por todas las adyacencias de mayor orden posible (adyacencias sin marcar) no cubiertas por ninguna adyacencia de orden superior. Estas adyacencias que definen a los implicados primos se enumeran, por ejemplo con letras, partiendo de las de orden superior hasta la de menor orden. En el ejercicio de la Tabla 5.8 están enumeradas de la *a* a la *g*.

### 5.1.4.3 TABLA DE IMPLICANTES

Como ya se indicó anteriormente (apartado 5.1.2.2, definiciones y propiedades de las funciones mínimas) en el Teorema de expresión irreducible, cualquier función mínima debe incluir sólo implicantes primos, pero cualquier función que incluya sólo implicantes primos no tiene por qué ser mínima, ha de incluir el menor número de implicantes primos. Además, aquellos implicados primos de la función que sólo ellos pueden cubrir a un término canónico, se les denomina **implicados primos esenciales** y deben forzosamente pertenecer a la función mínima. Otros implicados primos cubiertos por los implicados primos esenciales no pertenecerán a la expresión irreducible.

Para facilitar la tarea de búsqueda de los implicados primos, que permitan la realización mínima de una función lógica, se crea la **tabla de implicantes** de la Figura 5.35 correspondiente a la función del ejemplo dado por [5.29]. En sus filas se enumeran los implicantes primos clasificados por coste (de mayor orden a menor orden) y en sus columnas todos los términos canónicos (*minterms* o *maxterms*) de la función. En cada intersección de la matriz que se forma se señala con una marca (●) aquellos términos canónicos cubiertos por cada implicante primo.

En las filas, al lado de las letras que identifican los implicantes primos, conviene escribir sus coordenadas binarias. Esto facilita la posterior obtención de la expresión de la función mínima. Si el procedimiento se ha realizado mediante coordenadas binarias, esta información está disponible directamente en la tabla de adyacencias (Tabla 5.7). Si por el contrario el procedimiento se ha realizado mediante coordenadas

decimales, esta información es fácil de deducir. Por ejemplo, el implicante primo *a* según la Tabla 5.8 está definido por: 10, 11, 14, 15 (1, 4); los términos entre paréntesis indican las posiciones (expresado en pesos binarios) de las coordenadas vacuas o guiones (variables que se simplifican). En el ejemplo, los guiones estarán situados en la posición cero ( $2^0 = 1$ , de menor peso, situada más a la derecha) y la posición dos ( $2^2 = 4$ ), es decir, en las posiciones: X-X-. Los valores binarios (X) son el resto de las posiciones, que no son coordenadas vacuas, comunes a los números de los términos que cubren: 10, 11, 14, 15, por lo que escribiendo uno cualquiera de ellos en X-X-, se obtiene 1-1-.

Esta tabla permite una rápida detección de los implicantes primos esenciales. Observando las columnas de la tabla, cada columna con una única marca determina un **implicante primo esencial** en la fila correspondiente. En el ejemplo, el implicante primo *a* es el único que cubre al *minterm* 14 y el implicante primo *d* al *minterm* 2.

El **proceso de cobertura** de los términos canónicos consiste en marcar con un asterisco (\*), en las filas, los implicantes primos esenciales y señalar en las columnas con una marca (✓) los términos canónicos a los que cubren. Estos datos se obtienen de las tablas de adyacencias (Tabla 5.7 y Tabla 5.8). Por ejemplo: en la Figura 5.35 los implicantes primos esenciales son *a* y *d*.

Si faltan términos por cubrir, hay que seleccionar entre los implicantes primos que quedan (no esenciales), el menor número de ellos con el menor coste (mayor orden) que cubran los términos que faltan. Esto se puede realizar en la mayoría de los casos por simple observación o mediante la tabla de implicantes reducida.

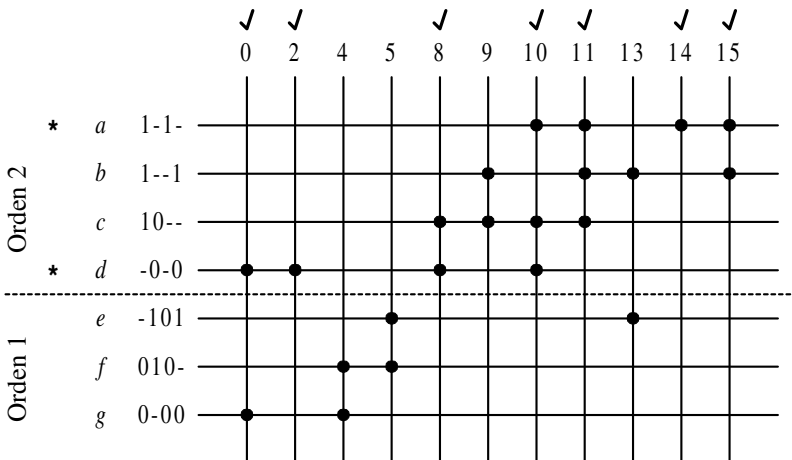


Figura 5.35. Tabla de implicantes primos

### 5.1.4.4 TABLA DE IMPLICANTES REDUCIDA

Cuando a simple vista no se encuentre una cobertura mínima de los términos canónicos mediante los implicantes primos, es conveniente construir una tabla de implicantes reducida, en la que sólo aparezcan aquellos que no son esenciales y los términos canónicos no cubiertos.

En el ejemplo, la tabla reducida correspondería a la de la Figura 5.36. Se pueden aplicar conceptos de **equivalencia** y **dominancia** entre las filas para descartar alguna de ellas.

Dos filas de una tabla de implicantes primos reducida son **equivalentes** si cubren los mismos términos (marcas en las mismas posiciones). En tal caso se eliminarán las de mayor coste (menor orden) y si son de igual coste se elegirá una cualquiera, existiendo varias soluciones de la función mínima.

En una tabla de implicantes primos reducida, una fila *i* **domina** a otra *j* si la fila *i* cubre a todos los términos canónicos de la fila *j* y alguno más.

En el ejemplo (Figura 5.36) se aprecia cómo el implicante primo *b* domina al implicante *c* y el implicante primo *f* al *g*.

Eliminando de la tabla los implicantes dominados *c* y *g* se obtiene la tabla reducida representada a la derecha de la Figura 5.36. En ella se aprecia cómo los implicantes *b* y *f* son primos esenciales (marcados con \*). Es decir, los implicantes primos *b* y *f* cubren a todos los términos de la tabla reducida y van a pertenecer a la expresión de la función mínima.

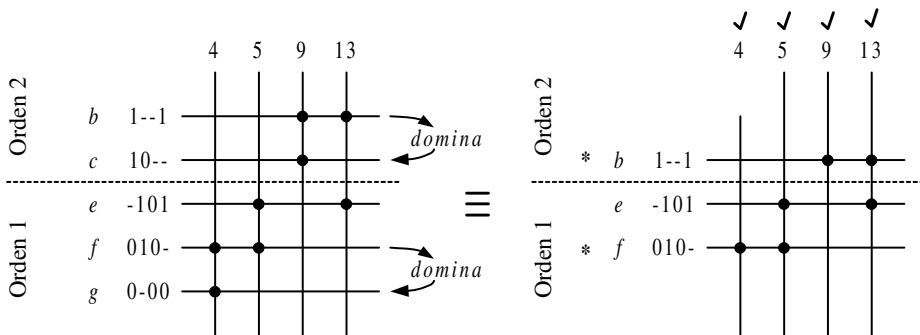


Figura 5.36. Tabla de implicantes reducida

En sistemas complejos se pueden crear tablas de implicantes primos reducidas secundaria, terciaria, etc., aplicando los criterios anteriores de implicante primo esencial, equivalencia y dominancia para la elección de implicantes.

Como consecuencia de esto pueden aparecer implicantes primos esenciales secundarios, terciarios, etc.



Una vez obtenidos todos los implicantes primos (esenciales y no esenciales) que formarán parte de la minimización de la función, se procede a obtener la **expresión de la función mínima**. A partir de la tabla de implicantes de la Figura 5.35 y de la Figura 5.36, se van escribiendo las expresiones de cada implicante primo, teniendo como referencia sus coordenadas binarias.

En la Figura 5.37 se han representado los implicantes que pertenecen a la función mínima y la obtención de las expresiones de cada uno de ellos.

		Variables	Expresión
		<i>d c b a</i>	
Implicantes	*	<i>a</i> 1-1-	<i>d b</i>
	*	<i>b</i> 1--1	<i>d a</i>
	*	<i>d</i> -0-0	$\bar{c} \bar{a}$
	*	<i>f</i> 010-	$\bar{d} c \bar{b}$

Figura 5.37. Obtención de la expresión de la función mínima

De la Figura 5.37 se tiene que la expresión de la función mínima es:

$$f(d, c, b, a) = d b + d a + \bar{c} \bar{a} + \bar{d} c \bar{b} \tag{5.31}$$

### 5.1.4.5 TABLAS CÍCLICAS Y MÉTODO DE PETRICK

Un caso particular en el que no se pueden aplicar los criterios de implicado primo esencial, equivalencia y dominancia es en las tablas cíclicas. En la Figura 5.38 se muestra una tabla de implicantes cíclica, en la que se aprecia que ninguno de los implicantes primos es esencial, ni equivalente, ni tampoco dominante.

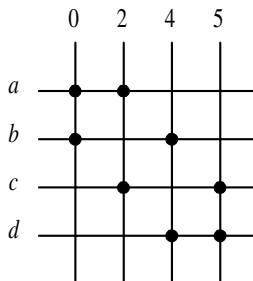


Figura 5.38. Tabla de implicantes cíclica

Para determinar en este caso el menor número de implicantes primos, de menor coste y que cubran a todos los términos canónicos (representados en las columnas) se utiliza el **método de Petrick**.

Este método se deduce de la tabla de la Figura 5.38 en la que el *minterm* 0 puede ser cubierto por los implicantes ( $a$  o  $b$ ), el *minterm* 2 por los implicantes ( $a$  o  $c$ ), etc. Para cubrir todos los *minterms* se deberá plantear la función:

$$Z = (a + b)(a + c)(b + d)(c + d) \quad [5.32]$$

Operando hasta llegar a expresarlo en suma de productos, se obtiene:

$$Z = da + cba + dcb + cb \quad [5.33]$$

Cada uno de los cuatro términos de la expresión [5.33] son soluciones, por cubrir a todos los términos canónicos de la tabla de implicantes cíclica. Los términos  $cba$  y  $dcb$  se descartan por ser de mayor coste que los términos  $da$  o  $cb$ , siendo uno cualquiera de estos últimos la solución.

### PROBLEMA RESUELTO 5-8

Simplificar, por el método de *Quine-McCluskey*, mediante ceros (*maxterms*) la función de la expresión [5.30], anteriormente simplificada por unos (*minterms*). Su expresión es:

$$f(d, c, b, a) = \sum_4 (0, 2, 4, 5, 8, 9, 10, 11, 13, 14, 15) \quad [5.34]$$

#### Solución:

Primeramente se convierte a *maxterms*:

$$\begin{aligned} \overline{f(d, c, b, a)} &= \sum_4 (1, 3, 6, 7, 12) \\ f(d, c, b, a) &= \overline{\overline{f(d, c, b, a)}} = \overline{\sum_4 (1, 3, 6, 7, 12)} = \overline{m_1 + m_3 + m_6 + m_7 + m_{12}} = \\ &= \overline{m_1} \cdot \overline{m_3} \cdot \overline{m_6} \cdot \overline{m_7} \cdot \overline{m_{12}} = M_{14} \cdot M_{12} \cdot M_9 \cdot M_8 \cdot M_3 \end{aligned} \quad [5.35]$$

Ordenando y expresando la función en *maxterms*, se obtiene la obtiene la expresión [5.36].

$$f(d, c, b, a) = \prod_4 (3, 8, 9, 12, 14) \quad [5.36]$$

Siguiendo el proceso descrito en la simplificación por el método *Q-M* con coordenadas decimales, se construye la siguiente tabla de adyacencias. Se observa en dicha tabla de *maxterms* que el índice se define como el número de ceros presentes en las coordenadas binarias del término *maxterm*.

Tabla 5.9. Tabla de adyacencias

Índice	Orden 0 <i>maxterms</i>		Orden 1 <i>maxterms</i>
1	8	✓	8, 9 (1) <i>c</i>
	3	<i>d</i>	8, 12 (4) <i>b</i>
2	9	✓	12, 14 (2) <i>a</i>
	12	✓	
3	14	✓	

La tabla de implicantes primos es la representada en la Figura 5.39. En dicha Figura 5.39 los implicantes primos esenciales *a*, *c* y *d* cubren a todos los términos canónicos, por lo que el proceso de la búsqueda de implicantes primos ha terminado.

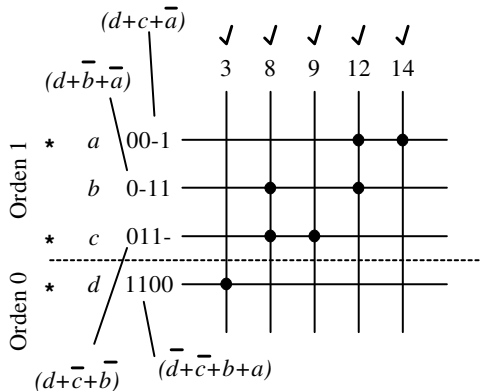


Figura 5.39. Tabla de implicantes primos

La expresión, en productos de sumas, de la función mínima:

$$f(d, c, b, a) = (d + c + \bar{a})(d + \bar{c} + \bar{b})(\bar{d} + \bar{c} + b + a) \quad [5.37]$$

## 5.2 CONVERSIÓN A PUERTAS NAND Y NOR

La representación de funciones de conmutación en *minterm* y *maxterm* determinan dos niveles de puertas. En *minterm* (función formada por sumas de términos productos) el primer nivel está formado por puertas AND y el segundo por una puerta OR de salida. Después de la simplificación por unos se conserva esta estructura llamada **realización AND a OR**.

Por dualidad, en *maxterm* (función formada por productos de términos suma) el primer nivel está formado por puertas OR y el segundo por una puerta AND de salida. Después de la simplificación por ceros se conserva esta estructura llamada **realización OR a AND**.

Se ha demostrado en el Capítulo 3 que los operadores NAND y NOR forman, cada uno de ellos por separado, un conjunto de operadores funcionalmente completo, es decir, cualquier función lógica puede expresarse mediante uno de ellos. Esto permite la síntesis de circuitos lógicos mediante un único tipo de puertas (NAND o NOR), lo que facilita el mantenimiento y reduce el coste de implementación.

### 5.2.1 Circuitos con dos niveles

En los métodos sistemáticos de simplificación estudiados en este capítulo se obtienen resultados de **circuitos con dos niveles**, con expresiones algebraicas en sumas de productos (puertas AND a OR) o productos de sumas (puertas OR a AND).

**Teorema:** En todo circuito de dos niveles AND a OR en el que todas las entradas actúan sobre puertas AND y la salida es una puerta OR, se pueden sustituir cada una de las puertas del circuito por puertas NAND de igual número de entradas, sin que la función se modifique.

**Demostración:** Considerando una expresión cualquiera en suma de los términos producto  $X, Y, Z$ , etc., por ejemplo la dada en [5.38]:

$$f = X + Y + Z + \dots \quad [5.38]$$

Si se realiza una doble negación sobre cada término producto y aplicando la ley de *De Morgan*, se obtiene la expresión [5.39].

$$f = \overline{\overline{X}} + \overline{\overline{Y}} + \overline{\overline{Z}} + \dots = \overline{\overline{X} \cdot \overline{\overline{Y}} \cdot \overline{\overline{Z}} \cdot \dots} \quad [5.39]$$

Como puede observarse, se trata de una implementación exclusivamente con puertas NAND.

Gráficamente (mediante circuito lógico) también se puede demostrar el teorema tal y como se representa en la Figura 5.40.

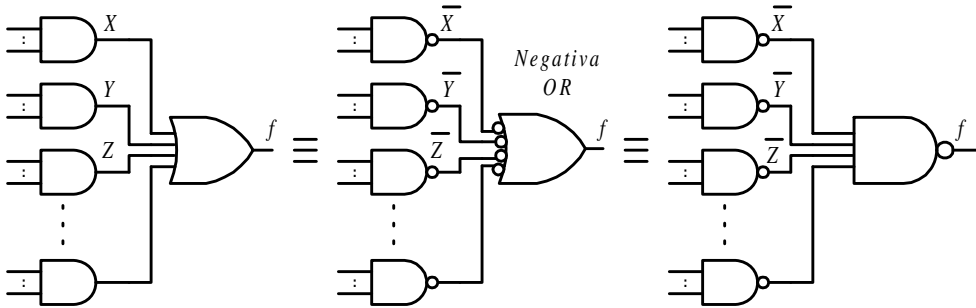


Figura 5.40. Proceso de conversión de expresiones en sumas de productos a puertas NAND

**Teorema** (dual al anterior teorema): En todo circuito de dos niveles OR a AND en el que todas las entradas actúan sobre puertas OR y la salida es una puerta AND, se pueden sustituir cada una de las puertas del circuito por puertas NOR de igual número de entradas, sin que la función se modifique.

**Demostración:** Considerando una expresión cualquiera en productos de los términos suma  $U, V, W$ , etc., por ejemplo la dada en [5.40].

$$f = U \cdot Y \cdot W \cdot \dots \quad [5.40]$$

Si se realiza una doble negación sobre cada término suma y aplicando la ley de *De Morgan*, se obtiene la expresión [5.41].

$$f = \overline{\overline{U}} \cdot \overline{\overline{V}} \cdot \overline{\overline{W}} \cdot \dots = \overline{\overline{\overline{U}} + \overline{\overline{V}} + \overline{\overline{W}} + \dots} \quad [5.41]$$

Como puede observarse se trata de una implementación exclusivamente con puertas NOR.

Gráficamente (mediante circuito lógico) también se puede demostrar tal y como se representa en la Figura 5.41.

En resumen, una función expresada en dos niveles puede ser sintetizada con un solo tipo de puertas.

Si la función está expresada en sumas de productos la conversión a puertas NAND es la de menor coste y consiste en sustituir cada puerta AND del primer nivel y cada puerta OR del segundo nivel por puertas NAND con el mismo número de entradas, como se muestra en la Figura 5.40.

Si la función está expresada en productos de sumas la conversión a puertas NOR es la de menor coste y consiste en sustituir cada puerta OR del primer nivel y cada puerta AND del segundo nivel por puertas NOR con el mismo número de entradas, como se muestra en la Figura 5.41.

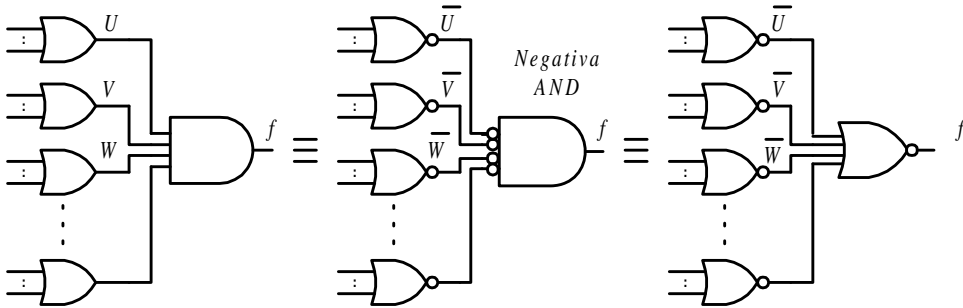


Figura 5.41. Proceso de conversión de expresiones en productos de sumas a puertas NOR

## 5.2.2 Circuitos con más de dos niveles

En primer lugar se estudia la solución de **entradas externas al segundo nivel**. En realidad es un caso particular en el que algunos de los términos producto (suma), del primer nivel, están formados por una variable ( $a$ ,  $b$ , ...), como se muestra en la expresión [5.42].

$$f = X + Y + \dots + a + b + \dots \quad [5.42]$$

Realizando una doble negación sobre cada término producto y aplicando la ley de *De Morgan* se obtiene la expresión [5.43].

$$f = \overline{\overline{X}} + \overline{\overline{Y}} + \dots + \overline{\overline{a}} + \overline{\overline{b}} + \dots = \overline{\overline{X} \cdot \overline{\overline{Y}} \cdot \dots \cdot \overline{\overline{a}} \cdot \overline{\overline{b}} \cdot \dots} \quad [5.43]$$

Se comprueba en dicha expresión, que se puede seguir sustituyendo las puertas del primer y segundo nivel por puertas NAND (NOR) si las entradas externas a la puerta del segundo nivel o de salida se complementan, como se muestra en la Figura 5.42.

Cuando el circuito está compuesto por puertas AND y OR, con más de dos niveles, se deben considerar los siguientes casos:

### 1) Más de dos niveles alternados AND a OR (OR a AND)

Estos circuitos se pueden convertir a sólo puertas NAND (NOR), sustituyendo cada una de sus puertas a NAND (NOR) con el mismo número de entradas y complementando las entradas externas que estén conectadas en los niveles impares, como se muestra en la Figura 5.43.

### 2) Más de dos niveles no alternados

Sea el circuito de la Figura 5.44 con más de dos niveles no alternados.

En este caso es recomendable descomponer el circuito en bloques de puertas AND a OR si se quiere convertir a puertas NAND (o bloques de puertas OR a AND si se quiere convertir a puertas NOR), como se muestra en la Figura 5.45.

Se realiza la conversión a NAND (NOR) de cada bloque por el procedimiento ya descrito en el apartado anterior, para posteriormente unir estos bloques, teniendo en cuenta, la complementación de las señales externas, donde corresponda, según se muestra en la Figura 5.46.

Uniendo los bloques convertidos se obtiene la conversión del circuito con más de dos niveles no alternados a puertas NAND que se muestra en la Figura 5.47.

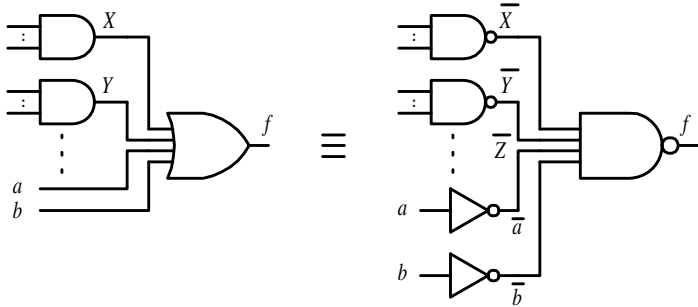


Figura 5.42. Entradas externas al segundo nivel

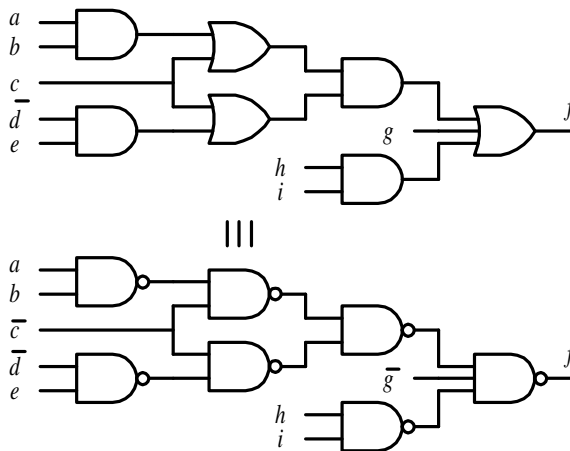


Figura 5.43. Conversión a puertas NAND en circuitos con más de dos niveles alternados

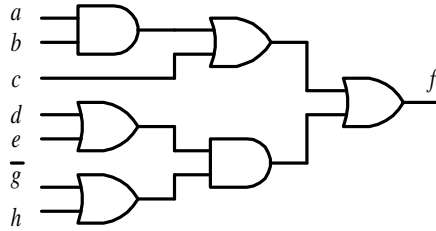


Figura 5.44. Circuito con más de dos niveles no alternados

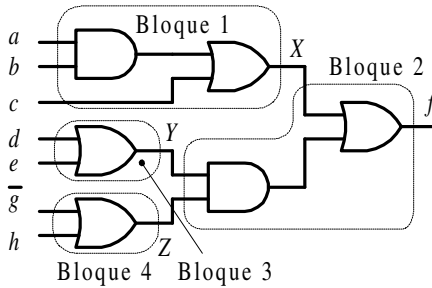


Figura 5.45. Selección de bloques de puertas AND a OR para convertir el circuito a puertas NAND

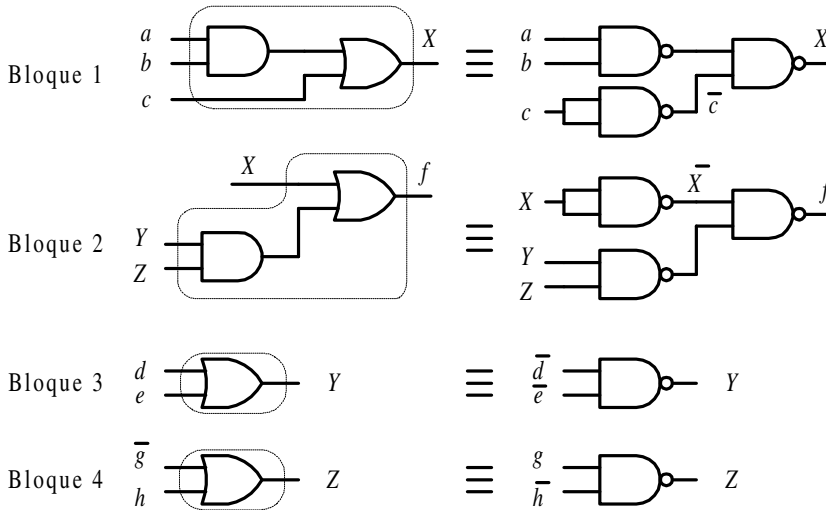


Figura 5.46. Conversión a puertas NAND de los bloques seleccionados



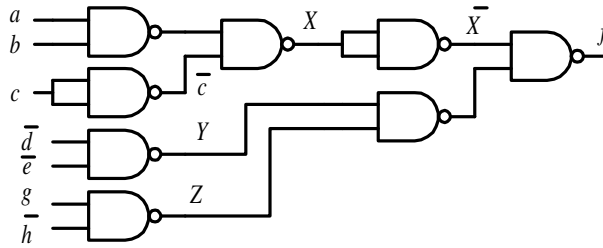


Figura 5.47. Conversión a puertas NAND para más de dos niveles no alternados

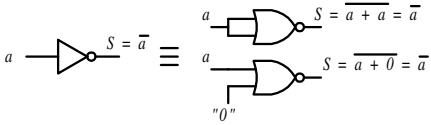
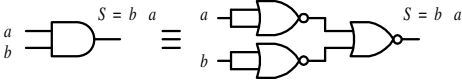
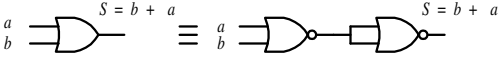
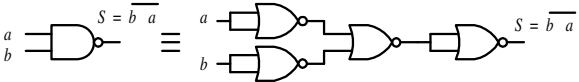
### 5.2.3 Circuitos con cualquier número de niveles y tipo de puertas

Siempre es posible aplicar un procedimiento más general en circuitos con cualquier número de niveles y tipo de puertas descomponiendo dicho circuito en tantos bloques como puertas lógicas tenga y realizando la conversión de cada una de estas puertas (bloques) a puertas NAND o NOR, como se indica respectivamente en la Tabla 5.10 y en la Tabla 5.11. Posteriormente, se unen todos los bloques convertidos y se eliminan aquellas partes del circuito que tengan un número par de puertas inversoras conectadas en niveles consecutivos (ley de involución).

Tabla 5.10. Tabla de conversión de cualquier puerta lógica a puertas NAND

Puerta	Demostración	Conversión a NAND
NOT	$S = \bar{a} = \overline{a \cdot a} = \overline{a \cdot 1}$	
AND	$S = b \cdot a = \overline{\overline{b \cdot a}}$	
OR	$S = b + a = \overline{\overline{b + a}} = \overline{\overline{b} \cdot \overline{a}}$	
NOR	$S = \overline{b + a} = \overline{\overline{\overline{b + a}}} = \overline{\overline{\overline{b} \cdot \overline{a}}}$	

Tabla 5.11. Tabla de conversión de cualquier puerta lógica a puertas NOR

Puerta	Demostración	Conversión a NOR
NOT	$S = \bar{a} = \overline{a + a} = \overline{a + 0}$ $S = \overline{a + a}$ $S = \overline{a + 0}$	
AND	$S = b \cdot a = \overline{\overline{b \cdot a}} = \overline{\overline{b} + \overline{a}}$	
OR	$S = b + a = \overline{\overline{b + a}} = \overline{\overline{b} \cdot \overline{a}}$	
NAND	$S = \overline{b \cdot a} = \overline{b} + \overline{a} = \overline{\overline{\overline{b} + \overline{a}}}$	

### 5.3 SIMPLIFICACIÓN DE FUNCIONES INCOMPLETAS O CON INDIFERENCIAS

En **funciones incompletas o con indiferencias** (apartado 3.3.8) el procedimiento de simplificación sigue siendo el mismo, salvo las siguientes consideraciones:

- En el **método de Karnaugh** se incluyen tanto los términos canónicos *minterms* (*maxterms*) como los términos indiferentes, representándose estos últimos mediante X. El procedimiento es el mismo: consiste en formar el mínimo número de grupos, estando éstos compuestos por el mayor número de unos (ceros) que sea potencia de dos. La diferencia consiste en incluir los términos indiferentes para aumentar la simplificación, añadiéndoles en aquellos grupos que aumentan el número de unos (ceros).

**Nota:** Sólo es necesario cubrir a todos los unos, es decir, términos canónicos *minterms* (o ceros, términos canónicos *maxterms*). No es necesario que sean cubiertos los términos indiferentes pues sólo se utilizarán si conviene y tomarán el valor que más interese (cero o uno) para aumentar la simplificación.

- En el **método de Q-M** se incluyen tanto los términos canónicos *minterms* (*maxterms*) como los términos indiferentes. El procedimiento de simplificación es el mismo, es decir, consiste en obtener el conjunto de

implicantes primos. La diferencia consiste en que al construir la tabla de implicantes primos no se incluyen los términos indiferentes siendo sólo necesario buscar los implicantes primos que cubren a los términos canónicos *minterms* (*maxterms*) de la función.

**Nota:** Los implicantes primos obtenidos por el método *Q-M* podrán ser de mayor orden al estar incluidos los términos indiferentes, obteniéndose por ello una mayor simplificación. Al incluir sólo los términos *minterms* (*maxterms*) en la tabla de implicantes se descartan a los implicantes primos que sólo incluyan indiferencias.

### PROBLEMA RESUELTO 5-9

Diseñar un sistema digital con cuatro entradas en código BCD natural que permita detectar aquellas codificaciones del código de entrada que tengan dos o tres unos (índice dos o tres).

Determinar:

- Tabla de verdad.
- Expresión compacta ( $\Sigma$ ) en *minterms*.
- Expresión compacta ( $\Pi$ ) en *maxterms*.
- Expresión simplificada por unos mediante *Karnaugh*.
- Expresión simplificada por ceros mediante *Karnaugh*.
- Conversión en puertas NAND de dos entradas.
- Conversión en puertas NOR de dos entradas.
- Dibujar el circuito lógico con el menor número de puertas iguales de dos entradas.
- Comprobar la simplificación anteriormente obtenida por el método de *Quine-McCluskey* (resolviendo por ceros y por unos).

#### Solución:

- Tabla de verdad.

A partir del enunciado se obtiene la Tabla 5.12.

- Expresión compacta ( $\Sigma$ ) en *minterms*.

Considerando los unos de la tabla de verdad y los términos indiferentes  $X$ , se obtiene la expresión [5.44].

$$f(d, c, b, a) = \sum_4 (3, 5, 6, 7, 9) + X(10, 11, 12, 13, 14, 15) \quad [5.44]$$

Tabla 5.12. Tabla de verdad

$d c b a$	$f_2$
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	1
0 1 0 0	0
0 1 0 1	1
0 1 1 0	1
0 1 1 1	1
1 0 0 0	0
1 0 0 1	1
1 0 1 0	X
1 0 1 1	X
1 1 0 0	X
1 1 0 1	X
1 1 1 0	X
1 1 1 1	X

- c) Expresión compacta ( $\Pi$ ) en *minterms*.

Considerando los ceros de la tabla de verdad y los términos indiferentes  $X$ , se obtiene la expresión [5.45].

$$f(d, c, b, a) = \prod_4 (7, 11, 13, 14, 15) \cdot X(0, 1, 2, 3, 4, 5) \quad [5.45]$$

- d) Expresión simplificada por unos mediante *Karnaugh*.

En la Figura 5.48 se representa el mapa de *Karnaugh* para la simplificación por unos. Obsérvese cómo para los términos indiferentes 10 y 12 se ha tomado el valor cero y para los términos 11, 13, 14 y 15 el valor uno para obtener una mayor simplificación. La función simplificada por unos se muestra en la expresión [5.46].

$$f(d, c, b, a) = b a + c a + d a + c b \quad [5.46]$$

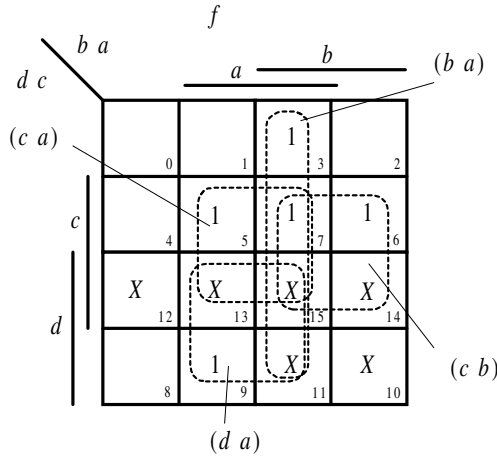


Figura 5.48. Simplificación por unos por el método de Karnaugh

- e) Expresión simplificada por ceros mediante *Karnaugh*.

En la Figura 5.49 se representa el mapa de *Karnaugh* para la simplificación por ceros. Obsérvese cómo para los términos indiferentes 3 y 5 se ha tomado el valor cero y para los términos 0, 1, 2 y 4 el valor uno para obtener una mayor simplificación. La función simplificada por ceros se muestra en la expresión [5.47].

$$f(d, c, b, a) = (b + a)(c + a)(d + c + b) \tag{5.47}$$

- f) Conversión en puertas NAND de dos entradas.

$$\begin{aligned} f(d, c, b, a) &= b a + c a + d a + c b = \overline{\overline{b a + c a + d a + c b}} = \\ &= \overline{\overline{b a} \cdot \overline{c a} \cdot \overline{d a} \cdot \overline{c b}} = \overline{\overline{b a} \cdot \overline{c a} \cdot \overline{d a} \cdot \overline{c b}} \end{aligned} \tag{5.48}$$

Se deja al lector que mediante el procedimiento estudiado en el apartado 5.2 Conversión a puertas NAND y NOR, obtenga este resultado resolviéndolo a través del circuito lógico.

- g) Conversión en puertas NOR de dos entradas.

$$\begin{aligned} f(d, c, b, a) &= (b + a)(c + a)(d + c + b) = \overline{\overline{b + a}} \overline{\overline{c + a}} \overline{\overline{d + c + b}} = \\ &= \overline{\overline{b + a} + \overline{c + a} + \overline{d + c + b}} = \overline{\overline{b + a} + \overline{c + a} + \overline{d + c + b}} \end{aligned} \tag{5.49}$$

Se deja al lector que mediante el procedimiento estudiado en el apartado 5.2 Conversión a puertas NAND y NOR, obtenga este resultado resolviéndolo a través del circuito lógico.

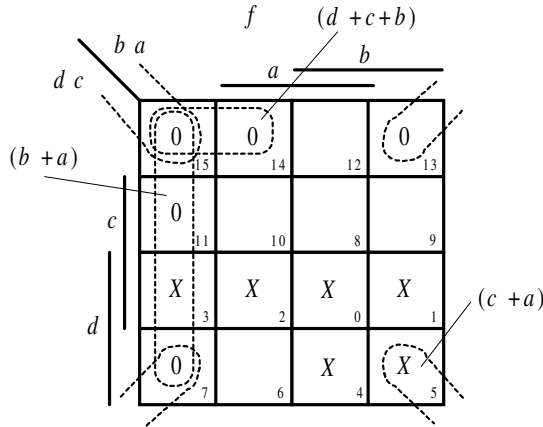


Figura 5.49. Simplificación por ceros por el método de Karnaugh

- h) Dibujar el circuito con el menor número de puertas iguales de 2 entradas.

En la expresión [5.48] se deduce que para implementar dicho circuito lógico se requieren nueve puertas NAND y en la expresión [5.49] el número de puertas es de ocho puertas NOR, siendo esta última expresión la que corresponde al circuito lógico con el menor número de puertas iguales de dos entradas representado en la Figura 5.50.

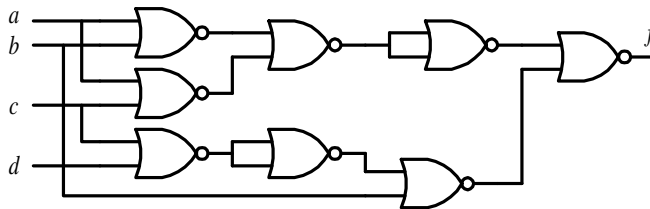


Figura 5.50. Circuito lógico con el menor número de puertas iguales de dos entradas

- i) Comprobar la simplificación anteriormente obtenida por método de *Quine-McCluskey* (resolviendo por ceros y por unos).

La **simplificación por unos** mediante el método de *Q-M* con coordenadas decimales, consiste en construir la Tabla 5.13 de adyacencias para *minterms* y términos indiferentes.

Tabla 5.13. Tabla de adyacencias para minterms y términos indiferentes

Índice	Orden 0 minterms	Orden 1 minterms	Orden 2 minterms
2	3 ✓	3, 7 (4) ✓	3, 7, 11, 15 (4, 8) <i>f</i>
	5 ✓	3, 11 (8) ✓	5, 7, 13, 15 (2, 8) <i>e</i>
	6 ✓	5, 7 (2) ✓	6, 7, 14, 15 (1, 8) <i>d</i>
	9 ✓	5, 13 (8) ✓	9, 11, 13, 15 (2, 4) <i>c</i>
	10 ✓	6, 7 (1) ✓	10, 11, 14, 15 (1, 4) <i>b</i>
	12 ✓	6, 14 (8) ✓	12, 13, 14, 15 (1, 2) <i>a</i>
3	7 ✓	9, 11 (2) ✓	
	11 ✓	9, 13 (4) ✓	
	13 ✓	10, 11 (1) ✓	
	14 ✓	10, 14 (4) ✓	
4	15 ✓	12, 13 (1) ✓	
		12, 14 (2) ✓	
		7, 15 (8) ✓	
		11, 15 (4) ✓	
		13, 15 (2) ✓	
		14, 15 (1) ✓	

En la Figura 5.51 se representa la tabla de implicantes primos. En dicha figura se observa que los implicantes primos *a* y *b* sólo cubren términos indiferentes por lo que no intervienen en la simplificación. Los implicantes primos esenciales *c*, *d*, *e* y *f* cubren a todos los términos canónicos, por lo que el proceso de la búsqueda de implicantes primos ha terminado.

La expresión de la función mínima, en sumas de productos, coincide con la obtenida anteriormente por el método de *Karnaugh*, expresión [5.46], que se muestra de nuevo en [5.50].

$$f(d, c, b, a) = ba + ca + da + cb \quad [5.50]$$

La **simplificación por ceros** mediante el método de *Q-M* con coordenadas decimales, consiste en construir la Tabla 5.14 de adyacencias para *maxterms* y términos indiferentes.

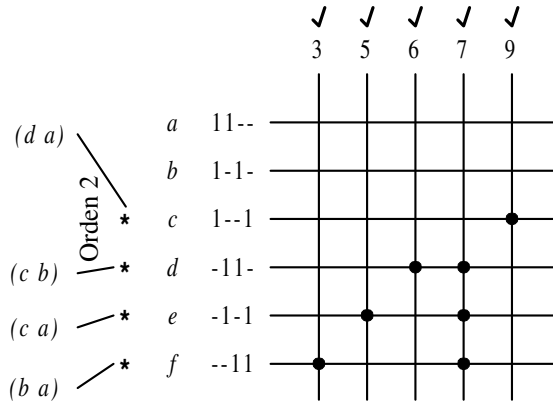


Figura 5.51. Tabla de implicantes primos

Tabla 5.14. Tabla de adyacencias para maxterms y términos indiferentes

Índice	Orden 0 maxterms		Orden 1 maxterms		Orden 2 maxterms	
4	0	✓	0, 1	(1) ✓	0, 1, 2, 3	(1, 2) e
	1	✓	0, 2	(2) ✓	0, 1, 4, 5	(1, 4) d
3	2	✓	0, 4	(4) ✓	1, 3, 5, 7	(2, 4) c
	4	✓	1, 3	(2) ✓	3, 7, 11, 15	(4, 8) b
2	3	✓	1, 5	(4) ✓	5, 7, 13, 15	(2, 8) a
	5	✓	2, 3	(1) ✓		
	7	✓	4, 5	(1) ✓		
1	11	✓	3, 7	(4) ✓		
	13	✓	3, 11	(1) ✓		
	14	✓	5, 7	(4) ✓		
0	15	✓	5, 13	(1) ✓		
			7, 15	(8) ✓		
			11, 15	(4) ✓		
			13, 15	(2) ✓		
			14, 15	(1) f		



En la Figura 5.52 se muestra la tabla de implicantes primos.

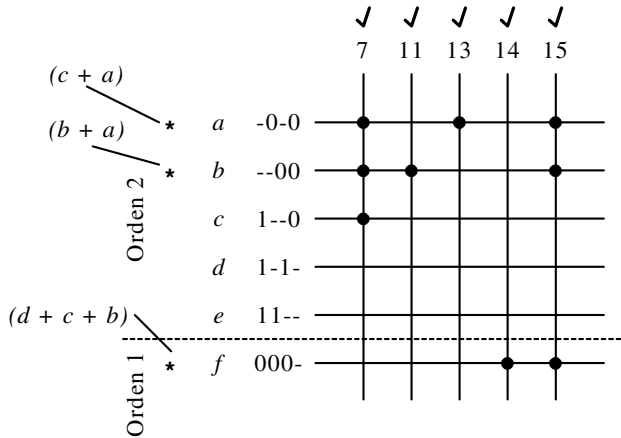


Figura 5.52. Tabla de implicantes primos

Se observa en la Figura 5.52 que los implicantes primos *d* y *e* sólo cubren términos indiferentes por lo que no intervienen en la simplificación. Los implicantes primos esenciales *a*, *b* y *f* cubren a todos los términos canónicos, por lo que el proceso de la búsqueda de implicantes primos ha terminado.

La expresión de la función mínima, en productos de sumas, coincide con la anteriormente obtenida por el método de *Karnaugh*, expresión [5.47], que se muestra de nuevo en la expresión [5.51].

$$f(d, c, b, a) = (b + a)(c + a)(d + c + b) \tag{5.51}$$

## 5.4 SIMPLIFICACIÓN MULTIFUNCIONAL

En los sistemas digitales es bastante usual que existan varias salidas (funciones), siendo necesario en este caso evitar la realización de tantos circuitos independientes como salidas tenga el circuito. Es preferible elaborar un diseño global que comparta puertas pertenecientes a las distintas funciones para reducir costes.

Para comprender más fácilmente la explicación, se consideran las funciones siguientes a las que se quiere optimizar su implementación mediante una simplificación multifuncional desarrollada en el Problema resuelto 5-10.

$$\begin{aligned}
 f(d, c, b, a) &= \sum_4 (2, 3, 4, 5, 6, 7, 12, 13) \\
 g(d, c, b, a) &= \sum_4 (2, 3, 7, 11, 12, 13, 15) \\
 h(d, c, b, a) &= \sum_4 (0, 1, 2, 3, 4, 5, 8, 9, 12, 13)
 \end{aligned}
 \tag{5.52}$$

El procedimiento de simplificación sigue siendo el mismo, salvo las siguientes consideraciones:

- El procedimiento por el **método de Karnaugh** consiste en realizar los siguientes pasos:
  - a) Crear un **mapa de Karnaugh en el que se representen todas las funciones**. En este mapa se indica, con una letra identificativa de la función, cada uno de los términos canónicos de las funciones ( $f, g, h, \dots$ ) o con la letra y el subíndice  $x$  ( $f_x, g_x, h_x, \dots$ ) sus términos indiferentes. En la Figura 5.56 se pueden ver los términos de las funciones  $f, g$  y  $h$  correspondientes al Problema resuelto 5-10.
  - b) Buscar todos los **términos canónicos que sean únicos**, es decir, aquellas celdas del mapa con una única letra identificativa de una función. Estos términos se agrupan como si del mapa de la función particular se tratase. En la Figura 5.57 se pueden ver los grupos  $f, g$  y  $h$  formados, correspondientes al Problema resuelto 5-10.
  - c) Buscar todos los **términos canónicos que sean comunes a dos funciones**, que no lo sean a tres y no hayan sido cubiertos en su totalidad en el apartado anterior. Estos términos se agrupan formando adyacencias que sean comunes a ambas funciones y en consecuencia, las funciones, comparten puertas en su realización física.
  - d) Repetir el proceso cubriendo, si existen, los **términos canónicos comunes a tres funciones**, y así sucesivamente. En la Figura 5.58 se pueden ver los grupos  $f, g$  y  $h$  formados, correspondientes al Problema resuelto 5-10.
- El procedimiento por el **método de Q-M** consiste, en primer lugar, en ordenar los términos canónicos (*minterms* o *maxterms*) según su índice, añadiendo una columna adicional que indique la función a la que pertenece el término. Posteriormente el procedimiento es el utilizado en el método *Q-M* para una sola función, es decir, se buscan las adyacencias de primer orden, después las de segundo orden, etc., pero cumpliendo la **restricción** de que dos adyacencias de un determinado orden pueden formar una adyacencia de orden superior si ambas tienen al menos una función común, indicando en la columna correspondiente las funciones que son comunes (Problema resuelto 5-10).

La simplificación multifuncional por este método permite realizar sistemas con numerosas variables.

### PROBLEMA RESUELTO 5-10

Un sistema digital tiene las siguientes funciones de salida:

$$\begin{aligned} f(d, c, b, a) &= \sum_4 (2, 3, 4, 5, 6, 7, 12, 13) \\ g(d, c, b, a) &= \sum_4 (2, 3, 7, 11, 12, 13, 15) \\ h(d, c, b, a) &= \sum_4 (0, 1, 2, 3, 4, 5, 8, 9, 12, 13) \end{aligned} \quad [5.53]$$

Realizar una simplificación individual de cada función y una simplificación multifuncional mediante los métodos de *Karnaugh* y *Quine-McCluskey*.

#### Solución:

- a) Simplificación individual de cada función por el método de *Karnaugh*.

Se resuelve, de forma individual, la simplificación de las funciones de salida del sistema como se muestra en la Figura 5.53, en la Figura 5.54 y en la Figura 5.55, obteniéndose las funciones simplificadas de la expresión [5.54].

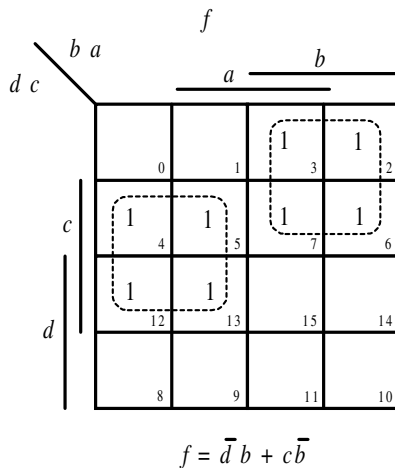


Figura 5.53. Simplificación individual de la función  $f$

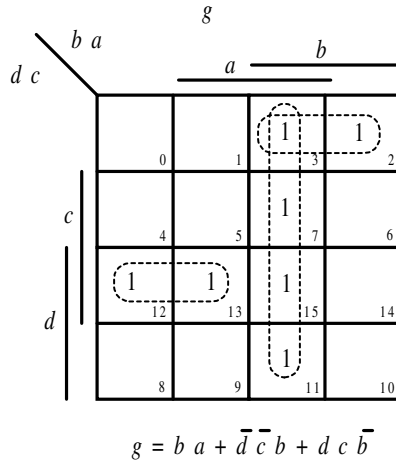


Figura 5.54. Simplificación individual de la función  $g$

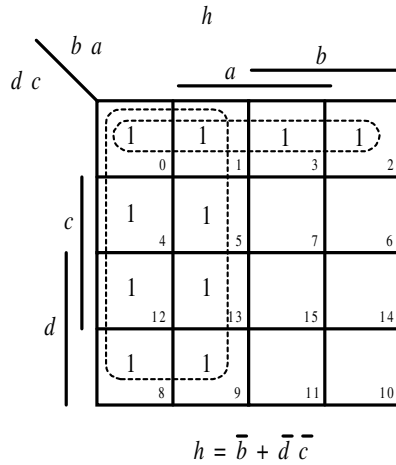


Figura 5.55. Simplificación individual de la función  $h$

Las expresiones de las funciones simplificadas son las siguientes:

$$\begin{aligned}
 f &= \bar{d} b + c \bar{b} \\
 g &= b a + \bar{d} \bar{c} b + d c \bar{b} \\
 h &= \bar{b} + \bar{d} \bar{c}
 \end{aligned}
 \tag{5.54}$$

b) Simplificación multifuncional por el método de *Karnaugh*.

- Creación del mapa de *Karnaugh* en el que se representan todas las funciones, tal como se muestra en la Figura 5.56.

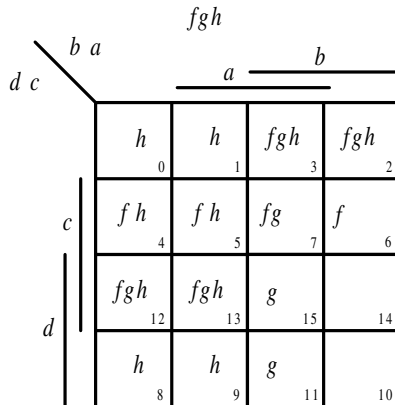


Figura 5.56. Mapa de *Karnaugh* en el que se representan todas las funciones

- Agrupación de los términos canónicos que sólo pertenecen a una función.

En este ejercicio los términos que pertenecen a una sola función son: para la función  $f$  el *minterm*  $m_6$ ; para la función  $g$  los *minterms*  $m_{11}$  y  $m_{15}$ ; y para la función  $h$  los *minterms*  $m_0$ ,  $m_1$ ,  $m_8$  y  $m_9$ . Se agrupan de la forma indicada en la Figura 5.57.

Obsérvese que al formarse los grupos, además de considerar los términos a los que abarca el grupo también hay que tener en cuenta las funciones cubiertas. Por ejemplo, el *minterm* 7 está cubierto por un grupo de  $f$  y por el  $g$ , por lo que sus dos funciones  $f$  y  $g$  están cubiertas. Sin embargo, el *minterm* 3 está cubierto sólo por el grupo  $g$  por lo que sus funciones  $f$  y  $h$  no están cubiertas.

- Agrupación de términos canónicos que son comunes a 2 funciones.

En este ejercicio no existen, pues aunque primeramente pudieran parecer serlo, los *minterms*  $m_4$ ,  $m_5$  y  $m_7$ . Dichos términos pertenecen a grupos de  $f$ ,  $g$ , y  $h$  ya creados, como puede verse en la Figura 5.57.

- Agrupación de términos canónicos que son comunes a 3 funciones.

En este ejercicio, los términos que pertenecen simultáneamente a tres funciones son: los *minterms*  $m_2$ ,  $m_3$ ,  $m_{12}$ ,  $m_{13}$  y se agrupan de la forma indicada en la Figura 5.58.

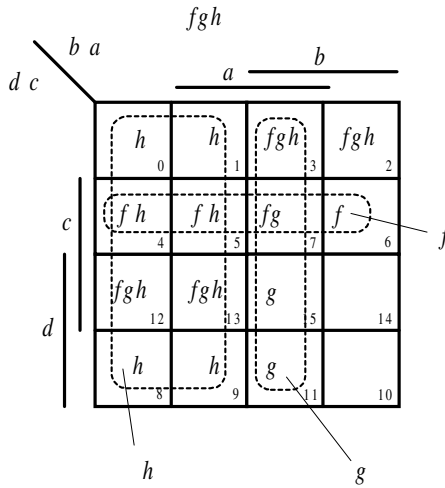


Figura 5.57. Grupos con términos de una única función

Con las agrupaciones obtenidas en el mapa de *Karnaugh* para términos comunes a una, dos y a las tres funciones, se representan las expresiones de las funciones de salida del sistema. Se debe tener en cuenta que las adyacencias obtenidas a partir de los términos que sólo pertenecen a una función sólo aparecerán en la expresión de dicha función, las adyacencias obtenidas a partir de términos que pertenecen a dos funciones aparecerán en la expresión de ambas funciones y las adyacencias obtenidas a partir de términos que pertenecen a las tres funciones aparecerán en la expresión de las tres funciones, y así sucesivamente, como puede verse en la Figura 5.59.

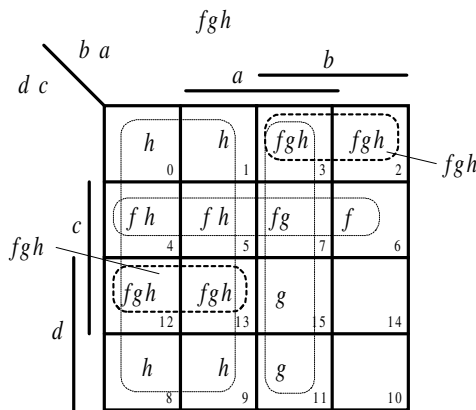


Figura 5.58. Grupos con términos que son comunes a las tres funciones

$$\begin{array}{l}
 \text{Elementos} \\
 \text{comunes} \\
 f = \bar{d}c + \boxed{\bar{d}\bar{c}b + dc\bar{b}} \\
 g = \bar{b}a + \boxed{\bar{d}\bar{c}b + dc\bar{b}} \\
 h = \bar{b} + \boxed{\bar{d}\bar{c}b + dc\bar{b}}
 \end{array}$$

Figura 5.59. Expresiones de las funciones de salida del sistema resuelto por el método de Karnaugh multifuncional

Las expresiones obtenidas por este procedimiento tendrán elementos comunes que permitirán diseñar un circuito más reducido, ya que se pueden compartir componentes entre las distintas funciones.

El diseño del circuito que se obtiene aplicando este método se muestra en la Figura 5.60.

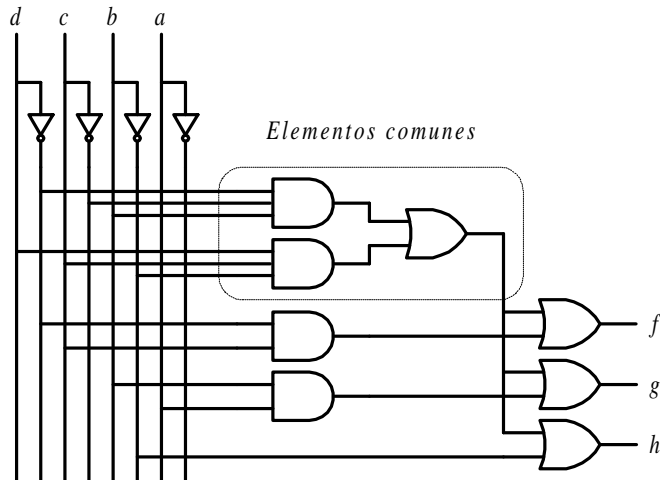


Figura 5.60. Circuito del sistema resuelto por el procedimiento multifuncional

- c) Simplificación multifuncional por el método de *Quine-McCluskey*.

En la Tabla 5.15 se muestra la tabla de adyacencias que se ha construido para *minterms* (de orden 0, 1, 2 y 3) y de términos indiferentes.

Tabla 5.15. Tabla de adyacencias para minterms y términos indiferentes

Índice	Orden 0 minterms	Orden 1 minterms	Orden 2 minterms	Orden 3 minterms
0	0 <i>h</i> ✓	0, 1 (1) <i>h</i> ✓	0, 1, 2, 3 (1, 2) <i>h f</i>	0, 1, 4, 5, 8, 9, 12, 13 (1, 2, 4) <i>h a</i>
1	1 <i>h</i> ✓	0, 2 (2) <i>h</i> ✓	0, 1, 4, 5 (1, 4) <i>h</i> ✓	
	2 <i>fgh</i> ✓	0, 4 (4) <i>h</i> ✓	0, 1, 8, 9 (1, 8) <i>h</i> ✓	
	4 <i>fh</i> ✓	0, 8 (8) <i>h</i> ✓	0, 4, 8, 12 (4, 8) <i>h</i> ✓	
	8 <i>h</i> ✓	1, 3 (2) <i>h</i> ✓	1, 5, 9, 13 (4, 8) <i>h</i> ✓	
	3 <i>fgh</i> ✓	1, 5 (4) <i>h</i> ✓	2, 3, 6, 7 (1, 4) <i>f e</i>	
	5 <i>fh</i> ✓	1, 9 (8) <i>h</i> ✓	4, 5, 6, 7 (1, 2) <i>f d</i>	
2	6 <i>f</i> ✓	2, 3 (1) <i>fgh k</i>	4, 5, 12, 13 (1, 8) <i>fh c</i>	
	9 <i>h</i> ✓	2, 6 (4) <i>f j</i>	8, 9, 12, 13 (1, 4) <i>h</i> ✓	
	12 <i>fgh</i> ✓	4, 5 (1) <i>fh</i> ✓	3, 7, 11, 15 (4, 8) <i>g b</i>	
	7 <i>fg</i> ✓	4, 6 (2) <i>f</i> ✓		
3	11 <i>g</i> ✓	4, 12 (8) <i>fh</i> ✓		
	13 <i>fgh</i> ✓	8, 9 (1) <i>h</i> ✓		
4	15 <i>g</i> ✓	8, 12 (4) <i>h</i> ✓		
		3, 7 (4) <i>fg i</i>		
		3, 11 (8) <i>g</i> ✓		
		3, 7 (2) <i>f</i> ✓		
		5, 13 (8) <i>fh</i> ✓		
		6, 7 (1) <i>f</i> ✓		
		9, 13 (4) <i>h</i> ✓		
		12, 13 (1) <i>fgh h</i>		
		7, 15 (8) <i>g</i> ✓		
		11, 15 (4) <i>g</i> ✓		
		13, 15 (2) <i>g g</i>		

En la Figura 5.61 se muestra la tabla de implicantes primos, y en la Figura 5.62 la tabla de implicantes reducida.



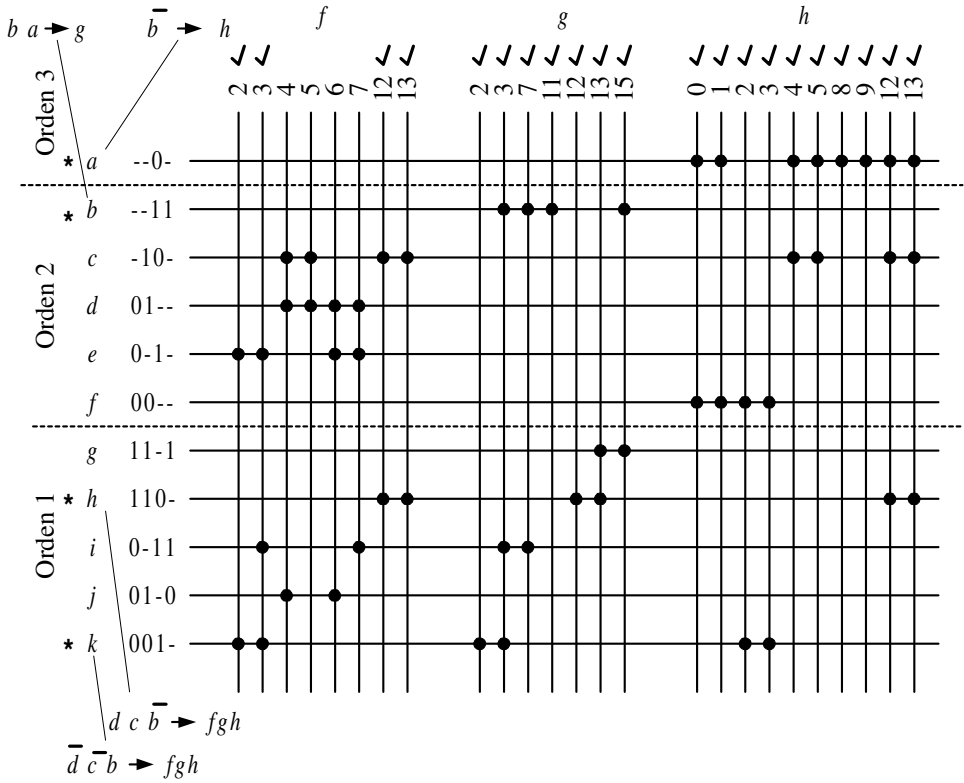


Figura 5.61. Tabla de implicantes primos

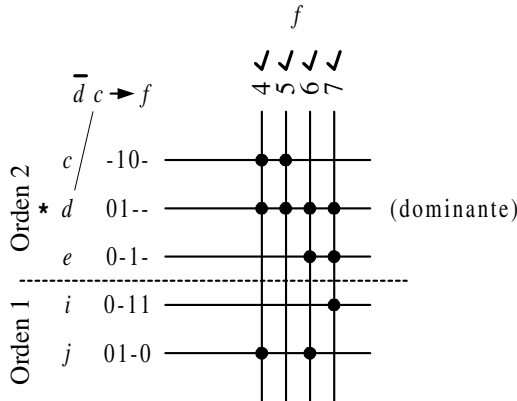


Figura 5.62. Tabla de implicantes reducida

Las expresiones de las funciones de salida del sistema resuelto por el método Q-M multifuncional se muestran en la Figura 5.63.

$$\begin{array}{r}
 \text{Elementos} \\
 \text{comunes} \\
 f = \bar{d}c + \boxed{\bar{d}\bar{c}b + dc\bar{b}} \\
 g = \bar{b}a + \boxed{\bar{d}\bar{c}b + dc\bar{b}} \\
 h = \bar{b} + \boxed{\bar{d}\bar{c}b + dc\bar{b}}
 \end{array}$$

Figura 5.63. Funciones de salida obtenidas por el método Q-M multifuncional

Como puede observarse, estas expresiones coinciden con las de la Figura 5.59, obtenidas anteriormente por el método de *Karnaugh* multifuncional.

## PROBLEMAS PROPUESTOS

5-1) Obtener las adyacencias de mayor orden de las siguientes funciones.

$$\begin{aligned}
 f(d, c, b, a) &= m_0 + m_1 + m_2 + m_3 + m_4 + m_5 + m_6 \\
 g(e, d, c, b, a) &= \sum_5 (4, 5, 8, 9, 11, 13, 22, 24, 26, 28) \\
 h(d, c, b, a) &= \sum_4 (9, 12, 13, 14, 15) + X(0, 1, 2, 5, 7, 8, 10) \\
 k(d, c, b, a) &= \bar{c}b\bar{a} + dc\bar{a} + \bar{d}cb\bar{a} + dca
 \end{aligned}$$

5-2) Obtener las adyacencias de mayor orden de las siguientes funciones.

$$\begin{aligned}
 u(d, c, b, a) &= M_3 \cdot M_4 \cdot M_7 \cdot M_9 \cdot M_{12} \cdot M_{13} \cdot M_{15} \\
 v(e, d, c, b, a) &= \prod_5 (3, 4, 5, 6, 7, 12, 13, 14, 19, 21, 23, 29) \\
 w(d, c, b, a) &= \prod_4 (0, 1, 2, 3, 4, 5, 6) \cdot X(7, 9, 11, 13, 15) \\
 y(d, c, b, a) &= (c + a)(c + \bar{a})\bar{c}
 \end{aligned}$$

5-3) Dada la función:

$$F = dc + \bar{d}\bar{c} + \bar{d}cb\bar{a} + \bar{d}ca$$

- Representarla sobre el mapa de *Karnaugh* indicando los grupos que dan origen a implicantes primos e implicantes primos esenciales.
- Obtener la expresión normalizada mínima en forma de sumas de productos.

- Obtener la expresión normalizada mínima en forma de productos de sumas.
- Representar el circuito lógico con puertas NOR de dos entradas.
- Representar el circuito lógico con puertas NAND de dos entradas.

5-4) Minimizar las funciones siguientes utilizando el método de *Karnaugh* y el de *Quine-McCluskey*. Comprobar los resultados obtenidos utilizando para ello la herramienta de simulación *Electronics Workbench*.

$$F = c a + c \bar{a} + \bar{c}$$

$$G = c b \bar{a} + d c \bar{b} + c a + \bar{d} c \bar{b} \bar{a}$$

$$H = (\bar{c} + \bar{b} + \bar{a})(\bar{c} + \bar{b} + a)(c + b)(c + \bar{b} + \bar{a})(c + \bar{b} + a)$$

$$f_1(d, c, b, a) = m_3 + m_6 + m_7 + m_9 + m_{12} + m_{14}$$

$$f_2(e, d, c, b, a) = m_3 + m_4 + m_5 + m_6 + m_7 + m_{12} + m_{13} + m_{14} + m_{19} + \\ + m_{21} + m_{23} + m_{29}$$

$$f_3(d, c, b, a) = M_0 \cdot M_2 \cdot M_4 \cdot M_5 \cdot M_7 \cdot M_{10} \cdot M_{11} \cdot M_{13} \cdot M_{14} \cdot M_{15}$$

$$f_4(d, c, b, a) = M_4 \cdot M_5 \cdot M_7 \cdot M_8 \cdot M_9 \cdot M_{10} \cdot M_{11} \cdot M_{12} \cdot M_{13} \cdot M_{14} \cdot M_{15}$$

$$f_5(d, c, b, a) = \sum_4 (1, 2, 3, 5, 7, 8, 9, 12, 14)$$

$$f_6(d, c, b, a) = \sum_4 (0, 1, 2, 3, 4, 6, 8, 9, 10, 11)$$

$$f_7(e, d, c, b, a) = \sum_5 (2, 8, 11, 12, 24, 25, 29) + X(0, 6, 31)$$

$$f_8(d, c, b, a) = \prod_4 (2, 3, 5, 6, 7, 8, 10, 11, 12, 14, 15)$$

$$f_9(d, c, b, a) = \prod_4 (8, 9, 10, 12, 13, 14)$$

$$f_{10}(e, d, c, b, a) = \prod_5 (1, 4, 7, 9, 12, 16, 18, 19) \cdot X(2, 3, 11)$$

$$f_{11} = b a + (c \oplus b)$$

$$f_{12} = (c \oplus b) + (\bar{c} \bar{b} + c b)$$

5-5) Minimizar por el método de *Karnaugh* y *Q-M* las siguientes funciones, obteniendo la expresión simplificada en sumas de productos.

$$f(c, b, a) = \sum_3 (1, 2, 3, 4, 6) + X(0)$$

$$g(e, d, c, b, a) = \sum_5 (1, 3, 9, 11, 13, 15, 17, 19, 24, 25, 28, 29)$$

$$h(d, c, b, a) = \prod_4 (1, 3, 4, 5, 7, 8, 9, 11, 13, 14)$$

$$k(e, d, c, b, a) = \prod_5 (3, 4, 5, 9, 11, 16, 17, 25, 26) \cdot X(0, 2, 7, 24, 31)$$

- 5-6) Minimizar por el método de *Karnaugh* y *Q-M* las siguientes funciones, obteniendo la expresión simplificada en productos de sumas.

$$f(c, b, a) = \sum_3 (0, 1, 2, 3, 4, 5, 6)$$

$$g(d, c, b, a) = \sum_4 (1, 3, 6, 8, 10, 11) + X(0, 2, 4, 12, 13)$$

$$h(d, c, b, a) = \sum_4 (0, 1, 2, 8, 9, 10) + X(5, 7, 13, 15)$$

$$k(e, d, c, b, a) = \prod_5 (4, 5, 8, 9, 11, 13, 22, 24, 26, 28)$$

- 5-7) Simplificar y convertir a puertas NAND los siguientes sistemas.

$$F = (c + b + a)(d + c + b)(c + a)(c + b)$$

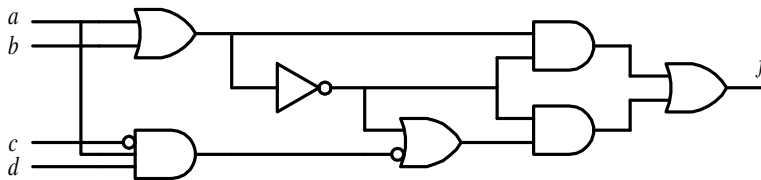
$$f_1(d, c, b, a) = m_9 + m_{12} + m_{13} + m_{14} + m_{15}$$

$$f_2(e, d, c, b, a) = M_1 \cdot M_3 \cdot M_9 \cdot M_{11} \cdot M_{13} \cdot M_{15} \cdot M_{17} \cdot M_{19} \cdot M_{24} \cdot M_{25} \cdot M_{28} \cdot M_{29}$$

$$f_3(d, c, b, a) = \sum_4 (0, 5, 7, 9, 11, 12, 13, 14)$$

$$f_4(d, c, b, a) = \prod_4 (1, 3, 5, 7, 8, 9, 12, 13, 14, 15)$$

- 5-8) Convertir a puertas NAND el siguiente esquema lógico.



- 5-9) Simplificar y convertir a puertas NOR los siguientes sistemas.

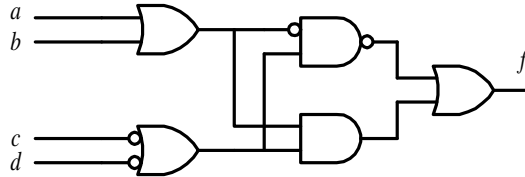
$$F = \bar{c} b \bar{a} + d c \bar{a} + \bar{d} c b \bar{a} + d c a$$

$$f_1(d, c, b, a) = m_3 + m_4 + m_5 + m_6 + m_7 + m_{11} + m_{12} + m_{13} + m_{14} + m_{15}$$

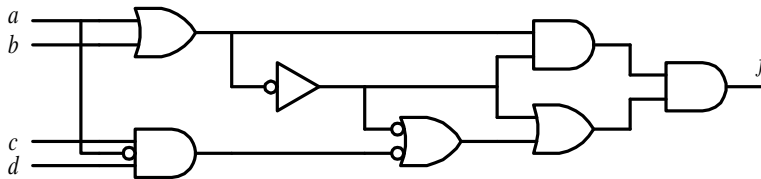
$$f_2(d, c, b, a) = \sum_4(4, 5, 8, 9, 10, 11, 12, 13, 14, 15)$$

$$f_3(d, c, b, a) = \prod_4(0, 1, 2, 3, 8, 10)$$

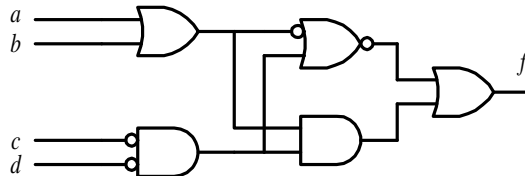
5-10) Convertir a puertas NOR el siguiente esquema lógico.



5-11) Convertir a dos niveles (AND a OR) el siguiente circuito.



5-12) Convertir a dos niveles (OR a AND) el siguiente circuito.



5-13) Diseñar un circuito con cuatro entradas ( $d, c, b, a$ ) que satisfaga la siguiente tabla de verdad:

$b a$	$F$	$G$
0 0	$c$	$d$
0 1	0	$c d$
1 0	$c + d$	0
1 1	1	1

5-14) Las decisiones en una junta de accionistas se reparten en grupos de 10%, 15%, 35% y 40%. Cada grupo dispone de un botón para votar y el resultado de la votación es por mayoría. Diseñar un sistema digital que indique si se aprueba la propuesta votada.

5-15) Diseñar, con el menor número de componentes, un circuito lógico de cinco entradas ( $e, d, c, b, a$ ) y dos salidas  $F$  y  $G$ , en el que la salida  $F$  toma un nivel alto cuando la mayoría de las entradas sean ceros y la salida  $G$  toma un nivel alto cuando las entradas tengan dos o tres unos.

5-16) Realizar una simplificación multifuncional de las funciones:

$$f(d, c, b, a) = \prod_4(0, 1, 2, 4, 5, 7, 8, 11, 14)$$

$$g(d, c, b, a) = \sum_4(0, 2, 4, 8, 9, 12, 13)$$

5-17) Realizar una simplificación multifuncional de las funciones:

$$f(e, d, c, b, a) = \prod_5(2, 4, 7, 11, 14, 18, 19, 27, 28, 29) \cdot X(3, 8, 9, 16, 26, 31)$$

$$g(e, d, c, b, a) = \prod_5(5, 6, 7, 11, 13, 18, 19, 27, 28) \cdot X(1, 2, 4, 9, 26)$$

5-18) Realizar una simplificación multifuncional de las funciones:

$$f(e, d, c, b, a) = \sum_5(1, 3, 5, 7, 11, 12, 25, 27, 28, 29) + X(0, 9, 20, 26)$$

$$g(e, d, c, b, a) = \sum_5(2, 3, 4, 9, 11, 13, 14, 25, 27, 31) + X(1, 15, 23, 28)$$

$$h(e, d, c, b, a) = \sum_5(4, 5, 6, 12, 17, 18, 25, 27) + X(0, 1, 3, 8, 15)$$

# TECNOLOGÍAS DE CIRCUITOS INTEGRADOS DIGITALES

---

---

**Objetivos:**

- Interpretar las características que definen a los circuitos integrados digitales.
- Elegir la tecnología más adecuada dependiendo de las especificaciones que deba reunir el sistema digital a realizar (consumo, velocidad, inmunidad al ruido, etc.).

**Contenido:** Introducción a las características internas de las puertas lógicas. Características funcionales de las puertas lógicas: estáticas y dinámicas. Descripción de las diferentes tecnologías digitales o familias lógicas. Evolución de las tecnologías digitales. Interpretación de las hojas características de los circuitos integrados digitales. Familia lógica TTL. Familia lógica CMOS.

**Simulación:** Mediante los programas de simulación *Electronics Workbench 5.0*, *OrCAD Demo v9* y *VeriBest VB99.0*, se comprueban las características funcionales de las puertas lógicas, tanto a partir de su esquema como utilizando el lenguaje de descripción *hardware* VHDL.

## 6.1 INTRODUCCIÓN A LAS CARACTERÍSTICAS BÁSICAS DE LAS PUERTAS LÓGICAS



En esta sección se introducen los conceptos y definiciones generales necesarios para comprender las características de una puerta lógica, con independencia de la tecnología digital utilizada.

Se denomina **circuito integrado** (C.I.) a un componente electrónico constituido por un soporte físico (pastilla de silicio) sobre el que están integrados componentes (transistores, resistencias, condensadores, etc.) interconectados entre sí, formando un circuito más o menos complejo.

Desde que en 1961 la empresa Fairchild comercializó el primer circuito integrado, las puertas lógicas se han desarrollado en este tipo de soporte y han evolucionado hacia la miniaturización, la mayor complejidad, la mayor fiabilidad y el menor coste. Esto ha contribuido al rápido desarrollo que han experimentado hoy en día las telecomunicaciones, la electrónica de consumo e industrial, la informática, etc.

Las tecnologías de fabricación han ido solucionando problemas de disipación de potencia y miniaturización, estableciéndose una clasificación, denominada **escala de integración**, que indica la cantidad de puertas lógicas que contiene un circuito integrado.

La Tabla 6.1 muestra las distintas escalas de integración que cronológicamente se han ido desarrollando.

Tabla 6.1. Escalas de integración

<i>Escalas de integración</i>		<i>Núm. puertas</i>	<i>Funciones</i>
<b>SSI</b>	<i>(Small Scale of Integration)</i> Pequeña escala de integración	1 a 10	Puertas lógicas y biestables
<b>MSI</b>	<i>(Medium Scale of Integration)</i> Mediana escala de integración	10 a 100	Sumadores, registros, contadores, etc.
<b>LSI</b>	<i>(Large Scale of Integration)</i> Gran escala de integración	100 a 1.000	Memorias, micropocesadores y periféricos
<b>VLSI</b>	<i>(Very Large Scale of Integration)</i> Muy alta escala de integración	1.000 a 10.000	Memorias, micropocesadores y periféricos
<b>ULSI</b>	<i>(Ultra Large Scale of Integration)</i> Ultra alta escala de integración	10.000 a 100.000	Microcomputadores, memorias, periféricos
<b>GLSI</b>	<i>(Giga Large Scale of Integration)</i> Giga alta escala de integración	mayor de 100.000	Microcomputadores, memorias, periféricos



Actualmente los sistemas digitales presentan la ventaja de estar formados por bloques funcionales interconectados (constituidos por C.I.), **prácticamente sin componentes discretos**, facilitando el diseño, reduciendo el espacio, aumentando la fiabilidad y reduciendo el coste, permitiendo de esta forma tiempos de desarrollo de productos y puesta de los mismos en el mercado mucho menores.

Las características de las puertas lógicas (consumo, velocidad, niveles lógicos, inmunidad al ruido, alimentación, etc.) dependen de la tecnología empleada en su construcción.

El objetivo de este capítulo es presentar estas características para, posteriormente, analizar cada una de ellas, según la tecnología empleada y así proporcionar al diseñador las distintas alternativas y soluciones que tiene a su disposición.

## 6.2 CARACTERÍSTICAS GENERALES DE LOS CIRCUITOS INTEGRADOS DIGITALES

En este apartado se describe el comportamiento de los circuitos integrados digitales desde el punto de vista de su funcionamiento y características eléctricas, siendo necesario que el diseñador adquiera estos conocimientos para la correcta interpretación de las hojas características aportadas por el fabricante del circuito integrado.

Los parámetros a destacar son:

- **Características estáticas**
  - a) Característica de transferencia y niveles lógicos.
  - b) Característica de entrada y salida.
  - c) Inmunidad frente al ruido.
  - d) Consumo y disipación de potencia.
- **Características dinámicas**
  - e) Retardos de propagación.
  - f) Frecuencia máxima de funcionamiento.
  - g) Producto consumo por tiempo de propagación.
- **Otras características**
  - h) Flexibilidad lógica.
  - i) Margen de temperatura.
  - j) Coste.

## 6.2.1 Características estáticas de los circuitos integrados digitales

### 6.2.1.1 CARACTERÍSTICAS DE TRANSFERENCIA Y NIVELES LÓGICOS

La **característica de transferencia** de una puerta lógica representa la señal de salida  $V_o$  en función de la señal de entrada  $V_i$ . En la Figura 6.1 se expresa la característica típica de una puerta *NOT*; en ella se puede observar cómo dentro de unos márgenes de tensión de entrada  $V_i$  la salida  $V_o$  se mantiene a un nivel prácticamente constante. Estos márgenes de entrada y sus correspondientes valores de salida definen los niveles lógicos altos *H* y bajos *L*.

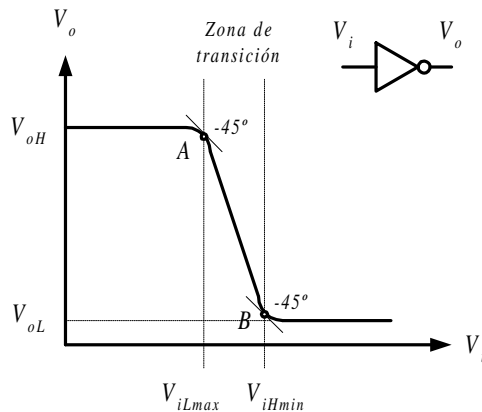


Figura 6.1. Característica de transferencia de un inversor

Los puntos A y B de la curva de la característica de transferencia, que tienen una pendiente de  $45^\circ$ , limitan la zona en la que se considera que la salida se mantiene constante y definen los siguientes parámetros característicos.

- $V_{iLmax}$ : Tensión de entrada (*input*) a nivel bajo (*Low*), máxima (*max*) que garantiza un nivel constante a la salida.
- $V_{iHmin}$ : Tensión de entrada (*input*) a nivel alto (*High*), mínima (*min*) que garantiza un nivel constante a la salida.
- $V_{oH}$ : Tensión de salida (*output*) a nivel alto (*High*).
- $V_{oL}$ : Tensión de salida (*output*) a nivel bajo (*Low*).

Las curvas de transferencia de los circuitos digitales no son únicas pues se ven afectadas por la temperatura, las tolerancias de fabricación, la tensión de alimentación, las cargas, etc. Por ello, es más útil considerar una familia de curvas características y en especial aquellas extremas o **envolventes**, representadas en la Figura 6.2.

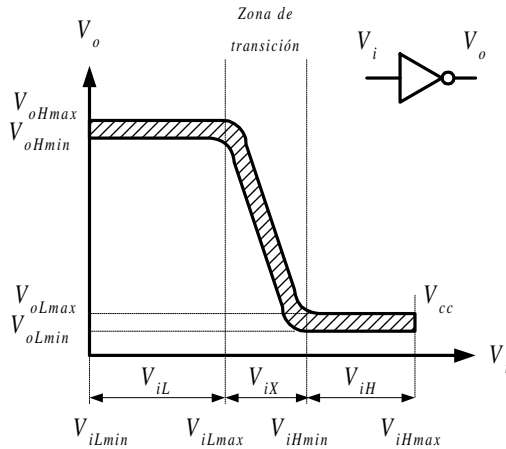


Figura 6.2. Envolventes de la familia de características de transferencia del inversor

A partir de la familia de características de transferencia se definen nuevamente los parámetros de tensión de entrada  $V_{iLmax}$  y  $V_{iHmin}$  para los casos más desfavorables. Los valores de tensión de entrada extremos  $V_{iLmin}$  (normalmente cero voltios) y  $V_{iHmax}$  (próximo a la tensión de alimentación  $V_{cc}$ ), definen el **intervalo de tensiones de entrada**,  $V_{iLmin} < V_i < V_{iHmax}$  fuera del cual existe riesgo de destrucción del circuito.

En esta familia de curvas se representan unos márgenes de tensiones de salida a nivel alto, comprendidos entre  $V_{oHmin}$  y  $V_{oHmax}$ , y a nivel bajo, entre  $V_{oLmin}$  y  $V_{oLmax}$ . También se representan los márgenes de tensiones de entrada a nivel bajo, comprendidos entre  $V_{iLmin}$  y  $V_{iLmax}$ , y a nivel alto, entre  $V_{iHmin}$  y  $V_{iHmax}$ .

Se define como **margen del uno**  $V_{iH}$  al rango de variación de la tensión de entrada de la puerta que es reconocido como nivel lógico alto; se muestra en la Figura 6.2.

$$V_{iHmin} < V_{iH} < V_{iHmax} \tag{6.1}$$

De forma análoga, se define como **margen del cero**  $V_{iL}$  al rango de variación de la tensión de entrada de la puerta que es reconocido como nivel lógico bajo; tal y como se ha representado en la Figura 6.2.

$$V_{iLmin} < V_{iL} < V_{iLmax} \tag{6.2}$$

Los valores de tensión de entrada comprendidos entre  $V_{iLmax}$  y  $V_{iHmin}$  no pertenecen ni al margen del cero, ni al margen del uno. Dicho espacio  $V_{iX}$ , en el que no está definido el nivel lógico, se denomina **zona de transición**.

$$V_{iLmax} < V_{iX} < V_{iHmin} \quad [6.3]$$

Normalmente las salidas de las puertas lógicas excitan a entradas de otras puertas, siendo útil el estudio de las **características de transferencia de una puerta conectada a otra**. Para analizar la característica de transferencia que se produce al conectar la salida de una puerta a la entrada de otra, se va a estudiar el circuito formado por la conexión de la salida de una puerta inversora conectada a la entrada de otra puerta inversora, como se muestra en la Figura 6.3. En dicho circuito se observa que  $V_{o2}$  es la doble negación de  $V_{i1}$  (ley de involución), siendo  $V_{o2} = V_{i1}$ , por lo que se puede considerar un circuito aproximado formado por las dos puertas inversoras con realimentación, y así poder representar en una misma gráfica las funciones de transferencias de ambas y sus áreas de intersección que definen los márgenes de variación de los niveles lógicos (en condiciones normales).

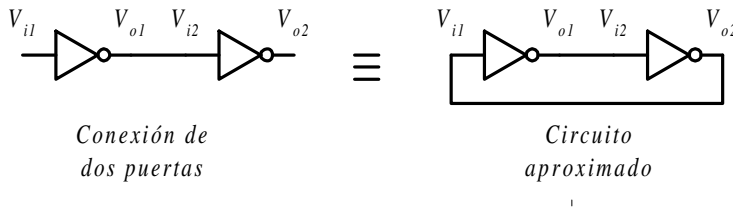


Figura 6.3. Conexión de dos puertas inversoras

Si sobre la familia de curvas de transferencia del primer inversor se trazan las del segundo inversor considerando que  $V_{o1} = V_{i2}$  y  $V_{o2} = V_{i1}$ , se obtiene la Figura 6.4, en la que se han destacado las zonas de funcionamiento normal y las de funcionamiento indeseado.

En la Figura 6.4, las zonas o áreas de intersección A y C definen el funcionamiento normal del circuito. En estas zonas el **funcionamiento normal a nivel bajo** está limitado por los puntos de tensión mínima de estado bajo  $V_{Lmin}$  y tensión máxima de estado bajo  $V_{Lmax}$  y en el **funcionamiento normal a nivel alto** por los puntos de tensión mínima de estado alto  $V_{Hmin}$ , y tensión máxima de estado alto  $V_{Hmax}$ .

La **máxima variación posible del nivel bajo**, en la Figura 6.4, está limitada por los puntos 0 V y  $V_{iLmax}$  (peor caso) y la **máxima variación posible del nivel alto** está limitada por los puntos  $V_{iHmin}$  (peor caso) y  $V_{cc}$ .

La zona B define el **funcionamiento indeseado** y corresponde a los valores que no tienen definido un nivel lógico.

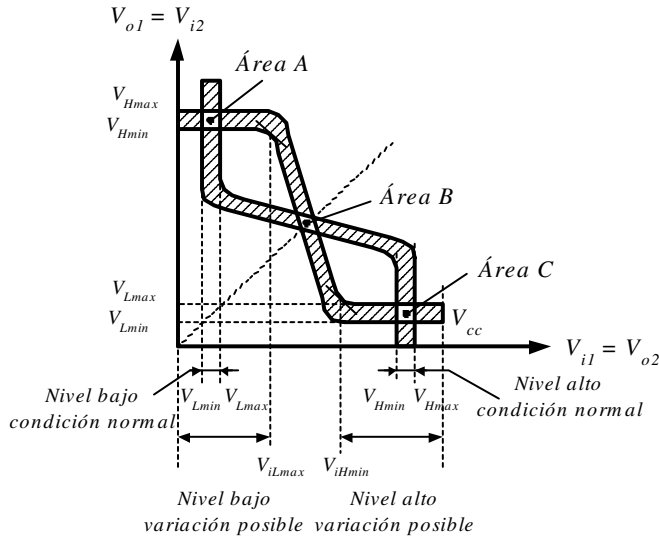


Figura 6.4. Características de transferencia de una puerta conectada a otra

También se fabrican puertas con una función de transferencia particular, como la representada en la Figura 6.5, denominadas **puertas Trigger-Schmitt**.

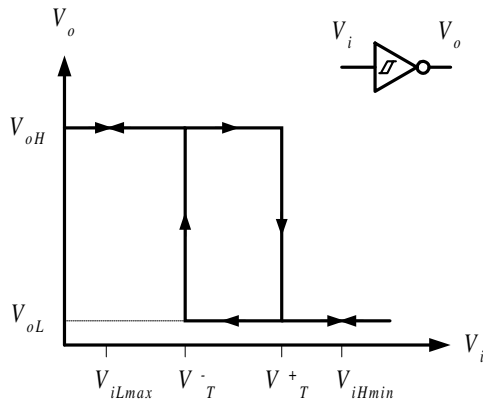


Figura 6.5. Característica de transferencia de una puerta inversora Trigger-Schmitt

Las características más notables de este tipo de puertas son las siguientes:

- Las transiciones entre niveles tienen una pendiente muy elevada.
- Las transiciones en la salida, de nivel alto a bajo ( $H \rightarrow L$ ) y de bajo a alto ( $L \rightarrow H$ ), se producen siguiendo el camino de las flechas indicado en la Figura

6.5, cuando la entrada  $V_i$  pasa por las **tensiones umbrales**  $V_T^+$  o  $V_T^-$  sin importar la velocidad de variación de la señal de entrada  $V_i$ .

### 6.2.1.2 CARACTERÍSTICAS DE ENTRADA Y SALIDA

Las características de entrada y de salida representan la relación que existe, respectivamente, entre la tensión y la corriente en la entrada  $V_i = f(I_i)$ , o bien en la salida  $V_o = f(I_o)$ .

En la Figura 6.6 se definen las corrientes de entrada y salida de un circuito digital en sus dos posibles niveles lógicos, con la notación:

$I_{iL}$ : Corriente de entrada a nivel bajo.

$I_{iH}$ : Corriente de entrada a nivel alto.

$I_{oL}$ : Corriente de salida a nivel bajo.

$I_{oH}$ : Corriente de salida a nivel alto.

Es bastante usual utilizar el convenio que se sigue en este libro de considerar positivas las corrientes entrantes al circuito, tal como se muestra en la Figura 6.6.

Si bien las características de entrada y las de salida pueden ser expresadas mediante gráficas, como por ejemplo las representadas en la Figura 6.7, preferentemente los fabricantes en sus hojas características dan mediante tablas los valores de las corrientes de entrada y salida límites, garantizadas bajo las condiciones más desfavorables (por ejemplo, para la mayor tensión de alimentación  $V_{CCmax}$ ).

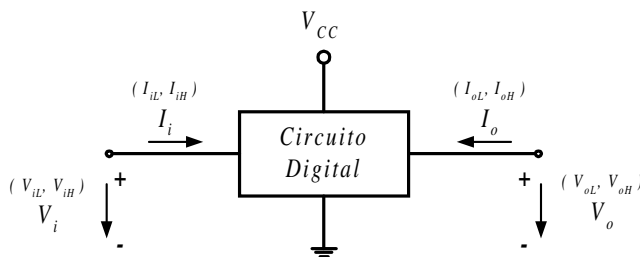


Figura 6.6. Notación de corrientes y tensiones, de entrada y salida de un circuito digital

Otra forma práctica de definir las características de entrada y salida de las puertas lógicas y sus diferentes zonas de trabajo, a partir de los datos de las hojas características proporcionadas por los fabricantes (valores límites que garantizan los niveles lógicos), es mediante los **perfiles de entrada y salida** de los circuitos lógicos, representados en la Figura 6.8. Esta representación gráfica establece las zonas de

funcionamiento mediante rectángulos, llamados **ventanas** de nivel alto ( $H$ ), indefinida y nivel bajo ( $L$ ). Estas ventanas quedan definidas por sus valores límites de corriente y tensión en su entrada y salida.

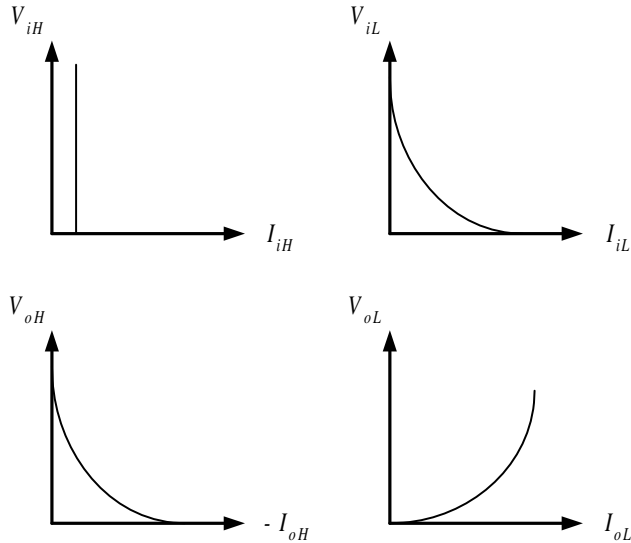


Figura 6.7. Gráficas de las características de entrada y salida de una puerta lógica

Es muy importante conocer las características de entrada y salida de los circuitos integrados digitales para evitar errores lógicos (indefinición o falta de garantía de los niveles  $L$  y  $H$  en las condiciones de trabajo elegidas), el excesivo aumento de disipación de potencia en la salida de la puerta, etc. Siendo este conocimiento imprescindible cuando se conecten puertas de diferente tecnología.

Al interconexionar varias puertas o circuitos entre sí se debe garantizar su **compatibilidad** eléctrica, tanto desde el punto de vista de sus tensiones como de sus corrientes, asegurando la garantía de los niveles  $L$  y  $H$  en las condiciones de trabajo elegidas. Para ello se deben tener presentes las siguientes condiciones:

### 1) Compatibilidad de tensiones

Al conectar la salida de un circuito con la entrada de otro, como se muestra en la Figura 6.9, se debe cumplir:

$$\begin{aligned} V_{oLmax} &\leq V_{iLmax} \\ V_{oHmin} &\geq V_{iHmin} \end{aligned} \quad [6.4]$$

Esta condición se puede establecer también de manera más gráfica, a partir de los perfiles lógicos de entrada y salida, considerando que, al superponer las

ventanas de nivel alto y bajo del perfil de salida sobre sus ventanas homólogas del perfil de entrada, deben estar contenidos todos sus puntos en estas últimas, como se muestra en la Figura 6.10.

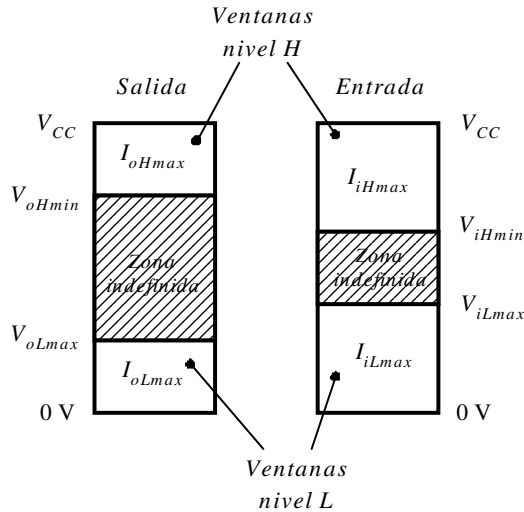


Figura 6.8. Perfiles de entrada y salida de un circuito lógico

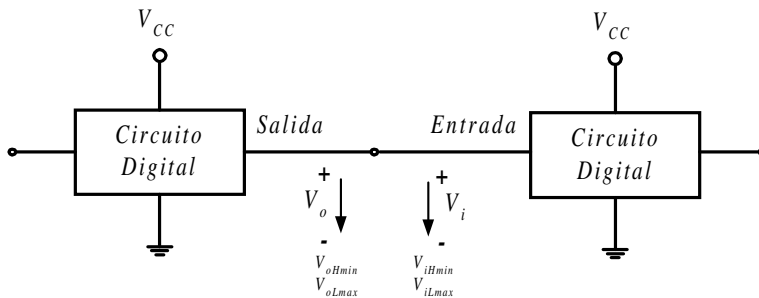


Figura 6.9. Interconexión de puertas lógicas desde el punto de vista de las tensiones

## 2) Compatibilidad de corrientes

Al conectar la salida de un circuito con la entrada de otro, como se muestra en la Figura 6.11, se debe cumplir que:

- Si la corriente de salida de un circuito es entrante (positiva), en la entrada del otro circuito conectado deberá ser saliente (negativa), o viceversa.



- b) El circuito que ataca o excitador debe suministrar la suficiente corriente en su salida como demande la entrada del circuito atacado o excitado. Es decir, se deben cumplir las siguientes condiciones en los niveles lógicos alto y bajo:

$$\begin{aligned} |I_{oHmax}| &\geq |I_{iHmax}| && \text{(nivel lógico alto)} \\ |I_{oLmax}| &\geq |I_{iLmax}| && \text{(nivel lógico bajo)} \end{aligned} \quad [6.5]$$

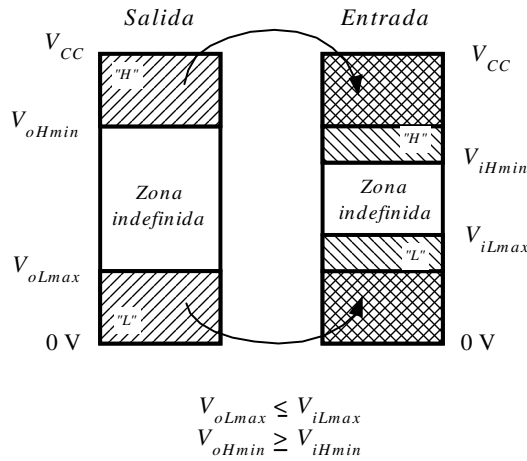


Figura 6.10. Compatibilidad de tensiones en la interconexión de puertas lógicas

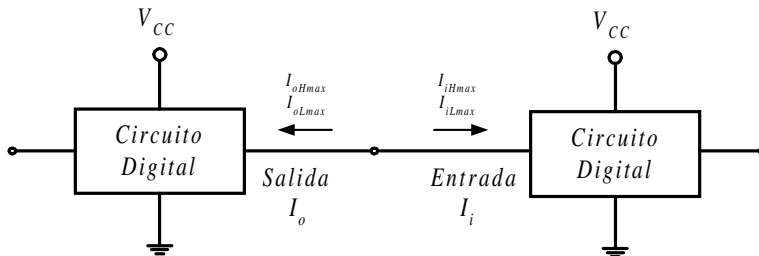


Figura 6.11. Interconexión de puertas lógicas desde el punto de vista de las corrientes

Estas condiciones se pueden fácilmente comprobar en las representaciones de los perfiles de entrada y salida de circuitos lógicos, verificando que:

- las corrientes de entrada y salida para un determinado nivel lógico deben tener signos contrarios y que,
- el perfil de salida, en cada nivel lógico, debe tener un valor absoluto de corriente mayor que el perfil de entrada, como muestra la Figura 6.12.

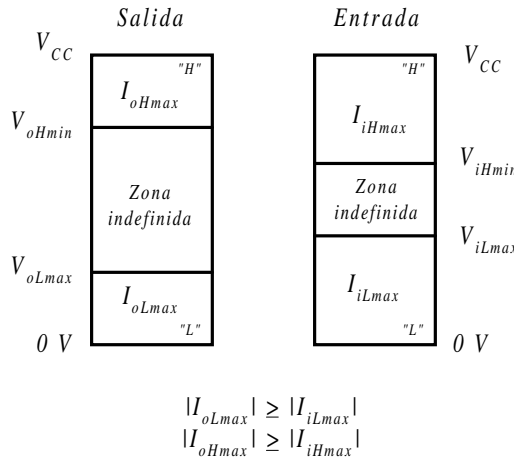


Figura 6.12. Condiciones de compatibilidad de las corrientes máximas en la interconexión de puertas lógicas

**PROBLEMA RESUELTO 6-1**

Estudiar la compatibilidad de dos familias lógicas cuyas características son:

<i>Parámetro</i>	<i>Familia Lógica (V<sub>CC</sub> = 5 V)</i>	
	<i>A</i>	<i>B</i>
<i>V<sub>oHmin</sub></i>	2,4 V	4,9 V
<i>V<sub>oLmax</sub></i>	0,4 V	0,3 V
<i>V<sub>iHmin</sub></i>	2 V	3,5 V
<i>V<sub>iLmax</sub></i>	0,8 V	1,5 V
<i>I<sub>oHmax</sub></i>	- 400 μA	-0,5 mA
<i>I<sub>oLmax</sub></i>	16 mA	0,5 mA
<i>I<sub>iHmax</sub></i>	40 μA	0,1 μA
<i>I<sub>iLmax</sub></i>	-1,6 mA	-0,1 μA

**Solución:**

Dibujando los perfiles de entrada y salida de cada familia lógica:

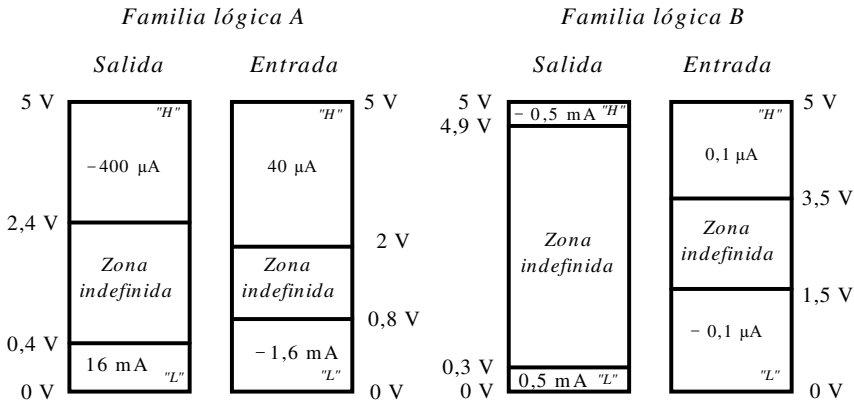


Figura 6.13. Perfiles de las dos familias lógicas del enunciado

Se deduce que individualmente, cada familia lógica, cumple la expresión [6.4] de compatibilidad de tensiones y la expresión [6.5] de compatibilidad de corrientes.

Para considerar **compatibles entre sí dos familias lógicas** se debe verificar que la salida de una de ellas es compatible (en tensión y corriente) con la entrada de la otra y a la vez también que la salida de esta última sea compatible con la entrada de la primera.

De los perfiles lógicos se deduce que desde el punto de vista de corrientes las salidas de la familia A pueden atacar a las entradas de la familia B, sin embargo, en cuanto a tensiones las ventanas (tanto a nivel H como L) de salida de la familia A no están contenidas en la ventana de entrada de la familia B por lo que no son compatibles (en el margen posible de 2,4 V a 3,5 V de la salida de A, la entrada de B es indefinida).

De igual forma desde el punto de vista de corrientes las salidas de la familia B no pueden atacar a las entradas en nivel bajo de la familia A por lo que no es compatible, aunque sí lo sea en cuanto a tensiones al estar las ventanas de salida de la familia B contenidas en las ventanas de entrada de la familia A. Resumiendo, las familias A y B no son compatibles, necesitando una **interfaz** para poder ser conectadas entre sí.

Para realizar interconexiones correctas y compatibles entre elementos lógicos es necesario conocer los valores de sus corrientes de entrada y salida. Los fabricantes lo suelen expresar en sus hojas características (para una familia lógica determinada),

indicando el número máximo de entradas de puertas que pueden ser conectadas a la salida de otra. Para ello se utilizan los parámetros *fan-in* y *fan-out*.

Se define el ***fan-in*** o **abanico de entrada**, como el valor absoluto de la corriente máxima que circula por una entrada de un circuito lógico expresado en unidades de carga (*u.c.*). Siendo la **unidad de carga (*u.c.*)** el máximo valor absoluto de la corriente que circula por la entrada de una puerta estándar de una familia lógica determinada. El ***fan-in*** se calcula en los dos niveles lógicos *H* y *L*, considerando aquel que sea más desfavorable, o lo que es lo mismo, el de mayor valor. Para ello se divide la corriente máxima de entrada de un circuito lógico entre la corriente máxima de entrada de la puerta lógica estándar, tanto en el nivel *H* como en el *L*, siendo el mayor resultado obtenido el *fan-in* de dicho circuito.

$$\begin{aligned} fan - in_L &= \frac{|I_{iLmax}|}{|I_{iL uc}|} \\ fan - in_H &= \frac{|I_{iHmax}|}{|I_{iH uc}|} \end{aligned} \quad [6.6]$$

Se define el ***fan-out*** o **abanico de salida**, como el valor absoluto de la corriente máxima que circula por una salida de un circuito lógico y que asegura los valores de tensión de los niveles lógicos (perfil de salida), expresado en unidades de carga (*u.c.*). Es decir, expresa el número máximo de entradas de puertas estándar que se pueden conectar a la salida de un circuito. El ***fan-out*** se calcula en los dos niveles lógicos *H* y *L* considerando aquel que sea más desfavorable, o lo que es lo mismo, el de menor valor. Para ello, se divide la corriente máxima de salida de un circuito lógico entre la corriente máxima de entrada de la puerta lógica estándar, tanto en el nivel *H* como en el *L*, siendo el menor resultado obtenido el *fan-out* de dicho circuito.

$$\begin{aligned} fan - out_L &= \frac{|I_{oLmax}|}{|I_{iLmax}|} \\ fan - out_H &= \frac{|I_{oHmax}|}{|I_{iHmax}|} \end{aligned} \quad [6.7]$$

Para aumentar el *fan-out* cuando se requiere mayor corriente de salida en una puerta se puede recurrir a conectar en paralelo las entradas y salidas de dos o más puertas como se muestra en la Figura 6.14.

Ambas puertas deben pertenecer al mismo encapsulado para garantizar que sus características sean lo más semejantes posibles y a la vez evitar diferencias de temperatura entre ellas. También se debe considerar el inconveniente de la generación de transitorios por las diferencias de retardos existentes en las puertas conectadas en paralelo.

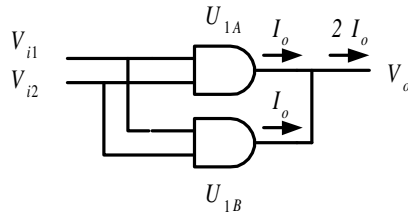


Figura 6.14. Aumento del fan-out

### PROBLEMA RESUELTO 6-2

Calcular el *fan-in* de un sistema cuyas corrientes de entrada son:

$$I_{iHmax} = 480 \mu A$$

$$I_{iLmax} = -16 \text{ mA}$$

Considerando que el valor de la unidad de carga *u.c.* (corrientes de entrada estándar de la familia lógica considerada) son:

$$I_{iHuc} = -40 \mu A$$

$$I_{iLuc} = 1,6 \text{ mA}$$

**Solución:**

$$\text{fan-in}_L = \frac{|I_{iLmax}|}{|I_{iLuc}|} = \frac{16 \text{ mA}}{1,6 \text{ mA}} = 10 \text{ u.c.}$$

$$\text{fan-in}_H = \frac{|I_{iHmax}|}{|I_{iHuc}|} = \frac{480 \mu A}{40 \mu A} = 12 \text{ u.c.}$$

[6.8]

El *fan-in* del sistema considerado es el mayor valor de los dos calculados, es decir, 12 *u.c.*

### PROBLEMA RESUELTO 6-3

Calcular el *fan-out* de la familia A del Problema resuelto 6-1.

**Solución:**

Mediante los datos de la familia A indicados en el enunciado o su perfil representado en la Figura 6.13 se calcula el *fan-out*:

$$fan - out_L = \frac{|I_{oLmax}|}{|I_{iLmax}|} = \frac{16 \text{ mA}}{1,6 \text{ mA}} = 10 \text{ u.c.}$$

$$fan - out_H = \frac{|I_{oHmax}|}{|I_{iHmax}|} = \frac{400 \mu\text{A}}{40 \mu\text{A}} = 10 \text{ u.c.}$$
[6.9]

El *fan-out* de la familia considerada es el menor valor de los dos calculados, en este caso al ser iguales, es 10 *u.c.*

### 6.2.1.3 INMUNIDAD FRENTE AL RUIDO

El **ruido** es una perturbación libre, no voluntaria, que puede modificar los niveles lógicos de los sistemas digitales, originando un mal funcionamiento del sistema.

Las **fuentes de ruido** más comunes suelen ser:

- Radiaciones de elementos eléctricos próximos al sistema digital, como contactores, motores de escobillas, interruptores, acoplos de líneas cercanas, etc.
- Ruidos provocados por elementos externos acoplados a través de la fuente de alimentación del sistema digital.
- Picos en la fuente de alimentación debidos a cambios bruscos de consumo en las líneas de suministro eléctrico.
- Ruidos debidos a reflexiones y oscilaciones de líneas no adaptadas.

Los **tipos de acoplo** más frecuentes entre la fuente de ruido y el sistema digital suelen ser:

- Magnético (inductivo).
- Electrostático (capacitivo).
- Radiación electromagnética.
- Por impedancia común.

La **manifestación del ruido**, como se puede apreciar en la Figura 6.15, puede ser de dos formas diferentes:

- **Ruido en continua o analógico (D.C.).** Es una variación aleatoria con una componente de muy baja frecuencia que se suma algebraicamente a los niveles lógicos y que puede modificar dichos niveles.
- **Ruido de alterna (A.C.).** Es una variación aleatoria en forma de impulsos de corta duración, con componente de frecuencias altas producido principalmente por acoplos capacitivos que pueden modificar los niveles lógicos del sistema.

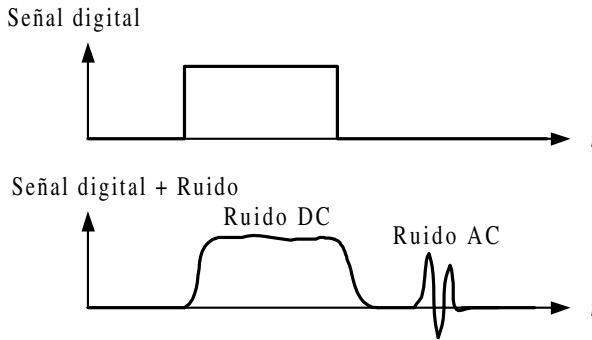


Figura 6.15. Manifestación del ruido sobre una señal digital

Se define la **inmunidad al ruido** o **margen de ruido en continua**  $V_{NM}$  (*Noise Margin*) al máximo valor de tensión que puede sumarse algebraicamente al nivel de salida de un circuito digital para que en el caso más desfavorable la entrada de otro circuito conectado a él no interprete el nivel lógico contrario.

En la Figura 6.16 se definen gráficamente los márgenes de ruido en el nivel alto y bajo.

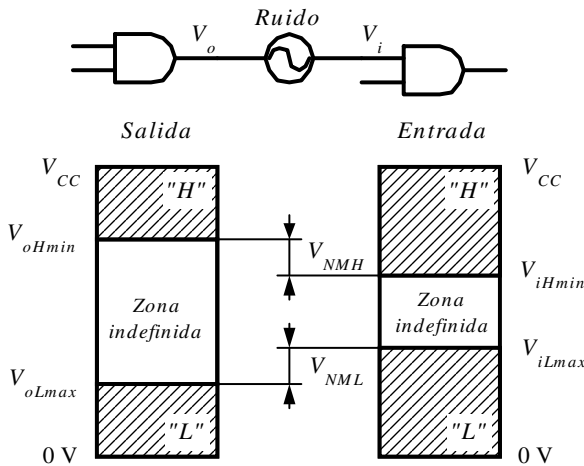


Figura 6.16. Definición gráfica de los márgenes de ruido en el nivel alto  $V_{NMH}$  y bajo  $V_{NML}$

Conociendo los perfiles lógicos de entrada y salida de un circuito se pueden determinar el margen de ruido, tanto en el nivel lógico alto  $V_{NMH}$  como en el nivel bajo  $V_{NML}$ , siendo:

$$V_{NMH} = V_{oHmin} - V_{iHmin} \quad [6.10]$$

$$V_{NML} = V_{iLmax} - V_{oLmax}$$

El margen de ruido  $V_{NM}$  del circuito es el menor de los valores de  $V_{NMH}$  y  $V_{NML}$ .

### PROBLEMA RESUELTO 6-4

Estudiar la inmunidad al ruido de las familias *A* y *B* del Problema resuelto 6-1.

#### Solución:

Teniendo en cuenta las características de la familia lógica *A* y las ecuaciones de la expresión [6.10] se tiene que:

$$V_{NMH} = V_{oHmin} - V_{iHmin} = 2,4 - 2 = 0,4 \text{ V} \quad [6.11]$$

$$V_{NML} = V_{iLmax} - V_{oLmax} = 0,8 - 0,4 = 0,4 \text{ V}$$

Ambos márgenes de ruido coinciden, por lo que la inmunidad al ruido de la familia *A* es de 0,4 V (en caso de no coincidir se considera el menor valor).

Teniendo en cuenta las características de la familia lógica *A* y las ecuaciones de la expresión [6.10] se tiene que:

$$V_{NMH} = V_{oHmin} - V_{iHmin} = 4,9 - 3,5 = 1,4 \text{ V} \quad [6.12]$$

$$V_{NML} = V_{iLmax} - V_{oLmax} = 1,5 - 0,3 = 1,2 \text{ V}$$

La inmunidad al ruido de la familia *B* es de 1,2 V.

En la Figura 6.17 se muestra el cálculo de este valor, utilizando para ello la representación gráfica de sus perfiles lógicos.

En los sistemas analógicos sólo se puede atenuar la señal de ruido, no se puede suprimir. En cambio, en los sistemas digitales el ruido es anulado si no supera los márgenes de ruido  $V_{NM}$  del sistema. Sin embargo, si el ruido supera el valor  $V_{NM}$  y modifica el nivel lógico en una puerta, este error se propaga por el circuito originando un incorrecto funcionamiento del sistema digital.

### 6.2.1.4 CONSUMO O DISIPACIÓN DE POTENCIA

Los circuitos lógicos requieren, para su funcionamiento, ser alimentados mediante una tensión continua  $V_{CC}$  o  $V_{DD}$  suministrada por una fuente de alimentación, lo cual implica un **consumo** o **disipación de potencia**.



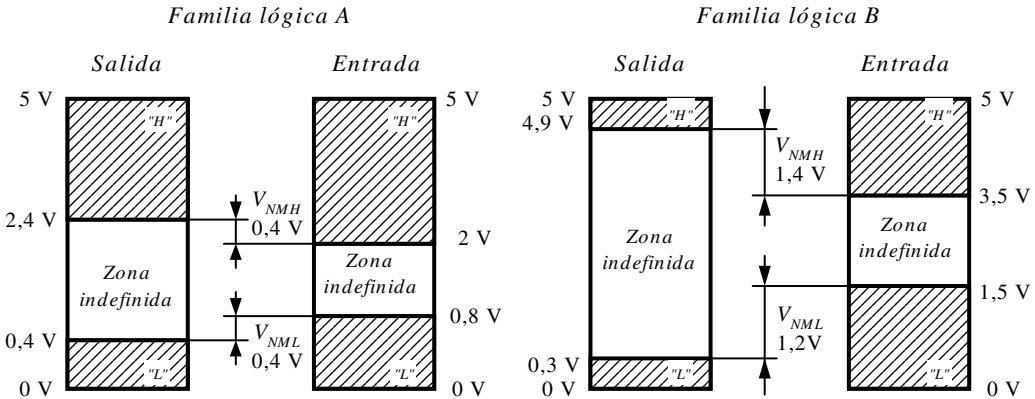


Figura 6.17. Inmunidad al ruido de las dos familias lógicas del enunciado

El consumo o disipación de potencia por puerta lógica es una característica importante al evaluar la bondad de una determinada tecnología, buscando siempre minimizar este parámetro para obtener ventajas como:

- Fuentes de alimentación de menor tamaño, menos costosas, con sistemas de refrigeración más simples, con mayor autonomía; siendo esta característica importante en equipos portátiles alimentados por batería.
- El nivel de integración está directamente relacionado con el consumo por puerta, pudiéndose implementar más puertas por unidad de superficie cuanto menor sea este consumo para una determinada tecnología.

La corriente absorbida por una puerta suele ser diferente según su nivel lógico de salida, siendo  $I_{CCH}$  e  $I_{CCL}$  la corriente absorbida en el nivel alto y bajo respectivamente. El consumo medio en **condiciones estáticas** se define para un ciclo de trabajo (*duty cycle*) del 50%, (ciclo simétrico en el que la mitad del periodo está a nivel alto y la otra mitad a nivel bajo), siendo la corriente media estática  $I_{CC}$ :

$$I_{CC} = \frac{I_{CCH} + I_{CCL}}{2} \quad [6.13]$$

y la potencia media estática:

$$P_D = I_{CC} V_{CC} \quad [6.14]$$

Existen tecnologías de circuitos integrados digitales (por ejemplo, *CMOS*) en las que el consumo de potencia se produce en las transiciones entre niveles lógicos, denominado **consumo dinámico de potencia**  $P_T$ . Este consumo puede ser incluso más elevado que los consumos estáticos, ya que el consumo dinámico de potencia en una

puerta lógica es proporcional a la frecuencia de transiciones de nivel que se producen en su salida.

Como resumen a lo anteriormente indicado cabe señalar que para el cálculo del consumo o disipación de potencia se debe distinguir y tener en cuenta la diferencia entre **consumo en condiciones estáticas**, en aquellos circuitos digitales con transiciones entre niveles lógicos poco frecuentes, y **consumo en condiciones dinámicas**, en aquellos circuitos con continuas transiciones en sus niveles lógicos.

Otro parámetro a tener en cuenta, proporcionado por el fabricante, es la **intensidad de salida en cortocircuito**  $I_{OS}$ , que se define como la intensidad que circula por la salida de una puerta que está a nivel alto cuando se produce un cortocircuito entre ella y masa.

El valor de  $I_{OS}$  se determina para las peores condiciones, es decir, con la máxima tensión de alimentación aplicada  $V_{CCmax}$ .

Esta intensidad de salida en cortocircuito tendrá un valor relativamente elevado, incrementando el consumo y la disipación normal de la puerta lógica, por lo que se deberá tener muy en cuenta, en los cálculos del diseño, en previsión de posibles anomalías y cortocircuitos en las salidas de las puertas lógicas.

### PROBLEMA RESUELTO 6-5

Calcular la disipación de potencia estática de las familias alimentadas con una tensión  $V_{CC} = 5\text{ V}$  y cuyos consumos son:

<i>Familias Lógicas (<math>V_{CC} = 5\text{ V}</math>)</i>		
	$I_{CCL}$	$I_{CCH}$
<b>A</b>	22 mA	8 mA
<b>B</b>	0,5 nA	0,5 nA

#### Solución:

La familia lógica *A* tiene una disipación de potencia de:

$$I_{CC} = \frac{I_{CCH} + I_{CCL}}{2} = \frac{8 + 22}{2} = 15\text{ mA}$$

$$P_D = I_{CC} V_{CC} = 15\text{ mA} \cdot 5\text{ V} = 75\text{ mW}$$

Y la familia lógica *B* tiene una disipación de potencia de:

$$I_{CC} = \frac{I_{CCH} + I_{CCL}}{2} = \frac{0,5 + 0,5}{2} = 0,5 \text{ nA}$$

$$P_d = I_{CC} V_{CC} = 0,5 \text{ nA} \cdot 5 \text{ V} = 2,5 \text{ nW}$$

## 6.2.2 Características dinámicas de los circuitos integrados digitales

### 6.2.2.1 RETARDOS DE PROPAGACIÓN

Los cambios en los niveles que se producen en las salidas de las puertas lógicas, al cambiar los niveles lógicos de las entradas, no son instantáneos. Esto es debido a las variaciones de carga (efectos capacitivos) que se originan en los semiconductores que las componen, dando lugar a los **retardos de propagación**.

A continuación, se definen los distintos retardos de propagación que, para mayor claridad se han representado, de forma gráfica, en la Figura 6.18.

#### a) Tiempos de propagación:

$t_{pLH}$ : **Puesta en ON** o tiempo de propagación en el flanco de subida de la señal de salida  $V_o$ . Es el intervalo de tiempo que se produce en la transición del nivel bajo  $L$  al nivel alto  $H$ , medido desde que la señal de entrada  $V_i$  alcanza el 50% de su valor final hasta que la señal de salida alcanza el 50% de su valor final.

$t_{pHL}$ : **Puesta en OFF** o tiempo de propagación en el flanco de bajada de la señal de salida  $V_o$ . Es el intervalo de tiempo que se produce en la transición del nivel alto  $H$  al nivel bajo  $L$ , medido desde que la señal de entrada  $V_i$  alcanza el 50% de su valor final hasta que la salida alcanza el 50% de su valor final.

$t_{pD}$ : **Tiempo de propagación medio**. Es la media aritmética de los dos tiempos anteriores, es decir:

$$t_{pD} = \frac{t_{pLH} + t_{pHL}}{2} \quad [6.15]$$

#### b) Tiempos de transición:

$t_r$ : **Tiempo de subida (rise time) de la señal de entrada** o tiempo de transición necesario para que la señal de entrada  $V_i$  pase, en el flanco de subida, del 10% al 90% de su valor final.

$t_f$ : **Tiempo de bajada (fall time) de la señal de entrada** o tiempo de transición necesario para que la señal de entrada  $V_i$  pase, en el flanco de bajada, del 90% al 10% de su valor inicial.

$t_{TLH}$ : **Tiempo de subida de la señal de salida** o tiempo de transición necesario para que la señal de salida  $V_o$  pase, en el flanco de subida, del 10% al 90% de su valor final.

$t_{THL}$ : **Tiempo de bajada de la señal de salida** o tiempo de transición necesario para que la señal de salida  $V_o$  pase, en el flanco de bajada, del 90% al 10% de su valor inicial.

c) **Tiempos de retraso:**

$t_{DLH}$ : **Tiempo de retraso en el flanco de subida de la señal de salida** (transición del nivel bajo  $L$  al nivel alto  $H$ ) medido desde que la señal de entrada  $V_i$  alcanza el 10% de su valor final hasta que la salida alcanza el 10% de su valor final.

$t_{DHL}$ : **Tiempo de retraso en el flanco de bajada de la señal de salida** (transición del nivel alto  $H$  al nivel bajo  $L$ ) medido desde que la señal de entrada  $V_i$  alcanza el 90% de su valor inicial hasta que la salida decrece al 90% de su valor inicial.

En un sistema digital las señales de entrada son aplicadas a las entradas de las puertas denominadas de primer nivel. Las señales de salida de las puertas de primer nivel se aplican a las entradas de otras puertas que constituyen el segundo nivel, y así sucesivamente.

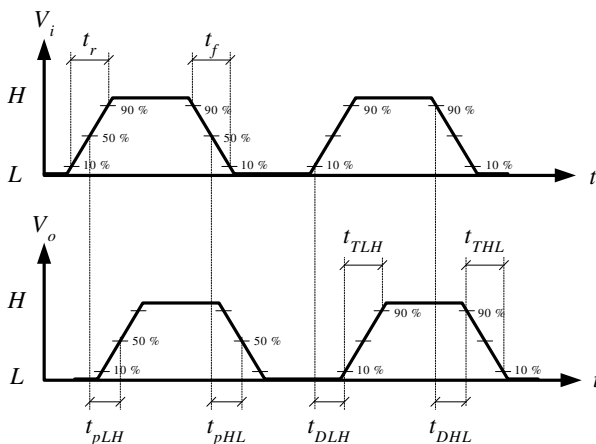


Figura 6.18. Tiempos de retardos en las ondas de entrada y salida de un dispositivo no inversor

Para una determinada tecnología se puede considerar que aproximadamente todos los niveles o puertas tienen el mismo retardo, por lo que en un **sistema digital formado por  $n$  niveles**, el retardo de propagación que se produce entre su entrada y su salida es  $n$  veces el retardo de propagación de una puerta. Se debe tener en cuenta, en el diseño de un circuito digital, que cuando en sus entradas están conectadas salidas de otros circuitos con niveles diferentes, unas señales estarán retrasadas respecto a otras, pudiéndose producir funcionamientos anómalos (guiños, cambios de nivel indeseados, etc.).

Los retardos de propagación limitan la frecuencia máxima de trabajo de los sistemas digitales.

### 6.2.2.2 FRECUENCIA MÁXIMA DE FUNCIONAMIENTO

La **frecuencia máxima** ( $f_{max}$ ) de una puerta es la mayor frecuencia que asegura que en la conmutación de la puerta se alcanzan los niveles lógicos necesarios. Para una determinada familia lógica, este valor se calcula a partir del tiempo de propagación medio  $t_{pD}$  (considerando una puerta NAND de dicha familia lógica), bajo las condiciones de salida en vacío o sin carga.

Un criterio utilizado para calcular la frecuencia máxima de trabajo, es considerar la frecuencia de entrada que produce un retardo de medio semiciclo en la salida, es decir, que tenga un periodo mínimo de cuatro veces el retardo  $t_{pD}$ , tal como se muestra en la Figura 6.19.

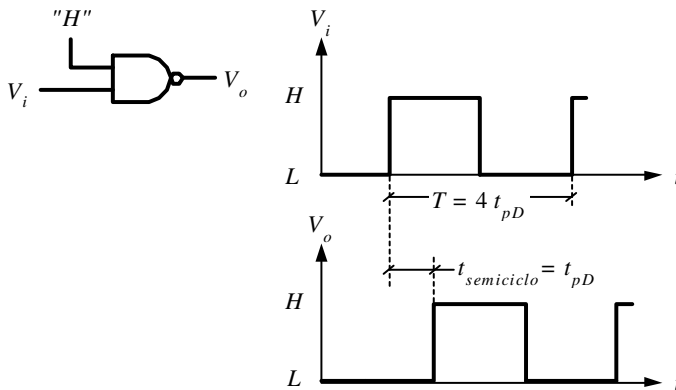


Figura 6.19. Determinación de la frecuencia máxima de funcionamiento

$$T_{min} = 4 t_{pD} \quad [6.16]$$

Siendo la frecuencia máxima de trabajo:

$$f_{max} = \frac{1}{4 t_{pD}} \quad [6.17]$$

Los fabricantes indican en sus hojas características una  $f_{max}$  de la familia lógica como la máxima frecuencia de reloj que permiten los circuitos secuenciales biestables (circuitos que se estudiarán posteriormente) construidos con puertas de la familia lógica considerada.

### 6.2.2.3 PRODUCTO CONSUMO POR TIEMPO DE PROPAGACIÓN

El consumo de potencia y el tiempo de propagación son dos parámetros relacionados en proporción inversa. El producto consumo por tiempo de propagación es un factor de mérito de gran utilidad para comparar familias lógicas entre sí. Dicho producto, representado en la expresión [6.18], indica la energía consumida por una determinada puerta lógica expresada en picojulios (pJ). Se busca siempre que este parámetro sea lo menor posible.

$$potencia \times retardo = P_D T_{pD} \quad [6.18]$$

## 6.2.3 Otras características de los circuitos integrados digitales

Para elegir una determinada familia lógica en el diseño de sistemas digitales, además de los factores anteriormente expuestos (retardos, inmunidad al ruido, consumo, etc.), se deben valorar otras características, tales como: flexibilidad lógica, margen de temperatura, potenciales de alimentación y su margen de tolerancia, tamaño, encapsulado, coste, etc.

### 6.2.3.1 FLEXIBILIDAD LÓGICA

La flexibilidad lógica es una medida de la versatilidad, capacidad o posibilidad de implementación de sistemas digitales con una determinada tecnología.

Los factores que caracterizan la flexibilidad de una determinada familia lógica son:

- **Cableado lógico:** en un apartado posterior se estudiará la posibilidad de conectar varias salidas de puertas lógicas entre sí (colector o drenador abierto). Se utiliza en el diseño de buses o implementación de funciones que sin gasto adicional no son posibles con puertas simples.

- **Capacidad de excitación:** directamente relacionado con el parámetro *fan-out* indica la capacidad que tienen las salidas de la familia lógica de atacar a un número determinado de entradas sin necesidad de circuitos adicionales (*buffers*).
- **Variedad en las salidas:** disponibilidad de puertas lógicas con salidas complementarias (con dos salidas, una directa y otra inversa). Permite el ahorro de inversores, y además, ambas salidas tienen retardos similares cuando se utilizan en líneas de transmisión de datos a distancia.

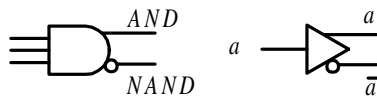


Figura 6.20. Variedad de salidas en un circuito digital

- **Variedad de bloques funcionales:** disponibilidad de circuitos integrados con funciones específicas (codificadores, contadores, sumadores, etc.), que permite una mayor facilidad, menor tamaño y coste en el diseño de sistemas digitales.
- **Compatibilidad con otras tecnologías:** posibilidad de conexión directa de entradas y salidas entre puertas de distintas tecnologías sin necesidad de circuitos adicionales, con adaptadores muy simples o con circuitos diseñados para este fin.

### 6.2.3.2 MARGEN DE TEMPERATURA

Este parámetro especifica el rango de temperatura dentro del cual el circuito integrado puede trabajar sin que se produzca su deterioro o un mal funcionamiento.

Según el tipo de aplicaciones, se han estandarizado principalmente dos **rangos de temperatura de trabajo** (con el dispositivo en funcionamiento):

- **Aplicaciones civiles** o de consumo: de 0 a 70 °C.
- **Aplicaciones militares:** de -55 a 125 °C.

Otros datos que el fabricante establece, relacionados con la temperatura son:

- El **rango de temperatura de almacenamiento** (con el dispositivo no operativo), no existiendo distinción entre el uso civil o militar y cuyo valor es: de -60 a 150 °C.
- **Resistencia térmica del dispositivo  $\theta_{JA}$** , utilizada en el cálculo de disipación del dispositivo o análisis térmico. Este parámetro está determinado por la

capacidad de disipación del tipo de encapsulado (DIP, LCC, PLCC, SOP) y los materiales utilizados (plásticos o cerámicos).

### 6.2.3.3 COSTE

El coste económico de los componentes de un sistema digital, en la mayoría de los casos, es un factor determinante en su diseño y realización.

## 6.2.4 Características ideales de una puerta lógica

Las características que debe reunir una puerta lógica ideal son las siguientes:

- **Alimentación** única  $V_{CC}$ .
- **Curva de transferencia** con niveles lógicos de salida iguales a los potenciales de alimentación. Transición entre niveles con pendiente infinita para una tensión de entrada igual a la mitad de la alimentación y con envolventes (curvas extremas de transferencia) coincidentes, como se muestra en la Figura 6.21.

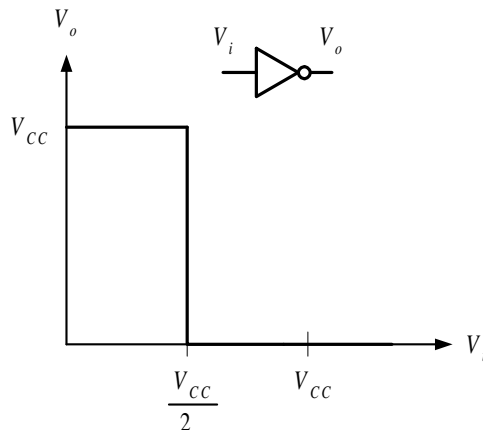


Figura 6.21. Curva de transferencia de una puerta inversora ideal

- **Impedancia de entrada** infinita para presentar una carga despreciable a las fuentes de excitación.
- **Impedancia de salida** nula para suministrar en la salida elevadas corrientes sin reducción de su tensión por la caída de potencial en la impedancia de salida.



- **Tiempo de propagación** nulo, pudiendo trabajar con cualquier frecuencia por muy alta que ésta sea.

Los avances tecnológicos en el campo de los circuitos integrados digitales, así como las distintas familias lógicas a las que han dado origen, tienden a que sus características se acerquen cada vez más a las de la puerta ideal.

## 6.3 FAMILIAS LÓGICAS

Una **familia lógica** está constituida por un conjunto de dispositivos lógicos contruidos con la misma tecnología (basada en transistores) y que por lo tanto tienen las mismas características, además de ser compatibles al poder conectarse entre sí.

Las familias lógicas evolucionan dando origen a **subfamilias lógicas** que conservan las características generales de las primeras, potenciando algunas de ellas (como consumo, velocidad, etc., acercándose al comportamiento de la puerta ideal) lo que permite que cada subfamilia se adapta mejor en determinadas aplicaciones.

En la Figura 6.22 se muestra una **clasificación de las principales familias lógicas**. Se puede apreciar una primera distinción tecnológica en base a la utilización de transistores: bipolares, unipolares o ambos (BICMOS), heredando en cada caso las siguientes características:

- **Tecnología bipolar:** tiene como ventaja su velocidad y como desventaja su consumo.

Las principales familias con esta tecnología son las siguientes:

- ◆ **TTL (*Transistor Transistor Logic*):** es la más popular. Sus subfamilias han mejorado progresivamente el producto consumo por tiempo de propagación.
- ◆ **ECL (*Emitter-Coupled Logic*):** de mayor velocidad que la familia TTL pero también con mayor consumo, estando limitado su uso en muchos casos por requerir sistemas de refrigeración.
- ◆ **I<sup>2</sup>L (*Integrated Injection Logic*):** la que más se aproxima a la familia ideal, aunque más cara al ser su fabricación bastante compleja.
- **Tecnología unipolar:** tiene como ventaja su gran densidad de integración y su reducido consumo y como desventaja su velocidad (son lentas). La familia más popular con esta tecnología es la **CMOS**.
- **Tecnología BICMOS:** combina las anteriores tecnologías (Bipolar y CMOS) en el mismo circuito integrado, consiguiendo las ventajas de ambas, como son: gran capacidad de excitación, buena velocidad, bajo consumo y considerable inmunidad al ruido.

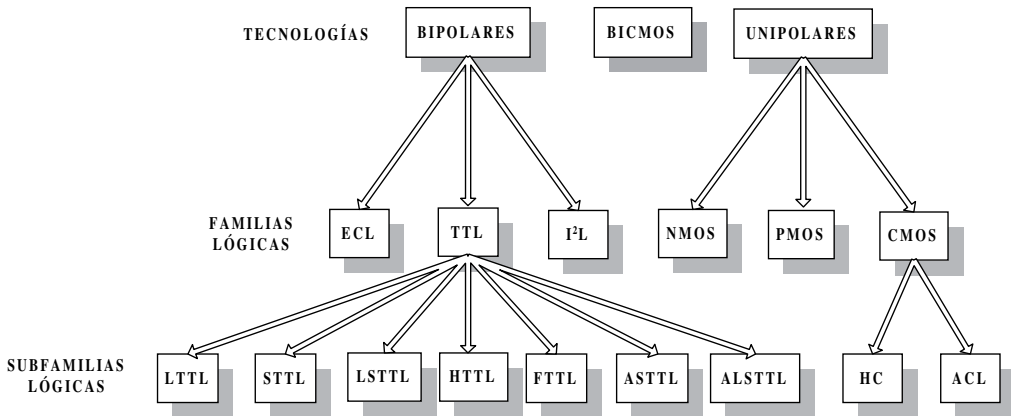


Figura 6.22. Clasificación de las principales familias lógicas

Actualmente el desarrollo tecnológico ha revolucionado el diseño mediante la utilización de **dispositivos lógicos programables** (PROM, PAL, PLA, etc.), permitiendo que el usuario realice diseños a medida a partir de estructuras predefinidas en las que se especifican sus interconexiones o circuitos integrados a medida **ASIC** (*Application Specific Integrated Circuits*).

## 6.4 FAMILIA LÓGICA TTL

### 6.4.1 Introducción

La familia lógica TTL (*Transistor Transistor Logic*) es una de las más populares, siendo ampliamente utilizada en el rango SSI y MSI debido a su buena velocidad, a su flexibilidad lógica y a su gran número de bloques funcionales integrados disponibles.

Es una familia de **lógica saturable** construida con tecnología bipolar en la que sus transistores trabajan en corte y saturación. Para saturar a un transistor bipolar se requiere una considerable corriente de base lo cual aumenta el consumo. Para que un transistor bipolar pase del estado de saturación al de corte se debe eliminar previamente el exceso de portadores de carga acumulada en su base, lo que aumenta los tiempos de conmutación.

## 6.5 PUERTA TTL-ESTÁNDAR

### 6.5.1 Constitución

En la Figura 6.23 se representa la estructura interna de la puerta NAND, puerta lógica básica representativa de la familia TTL estándar, en la que se pueden apreciar tres etapas:

- 1) **Etapa de entrada:** constituida por el transistor  $Q_1$  de tipo multiemisor que realiza la función AND (considerando como entradas de dicha función sus dos terminales de emisor y como salida su colector). Los diodos  $D_1$  y  $D_2$ , denominados CAMP, protegen las entradas ante tensiones negativas que a nivel bajo producen un incremento de  $I_{IL}$ , que pueden destruir la puerta. Estos diodos CAMP empiezan a conducir cuando la tensión de entrada supera una tensión negativa correspondiente a su tensión de codo  $-V_D$ , evitando que esta corriente circule internamente por la puerta.

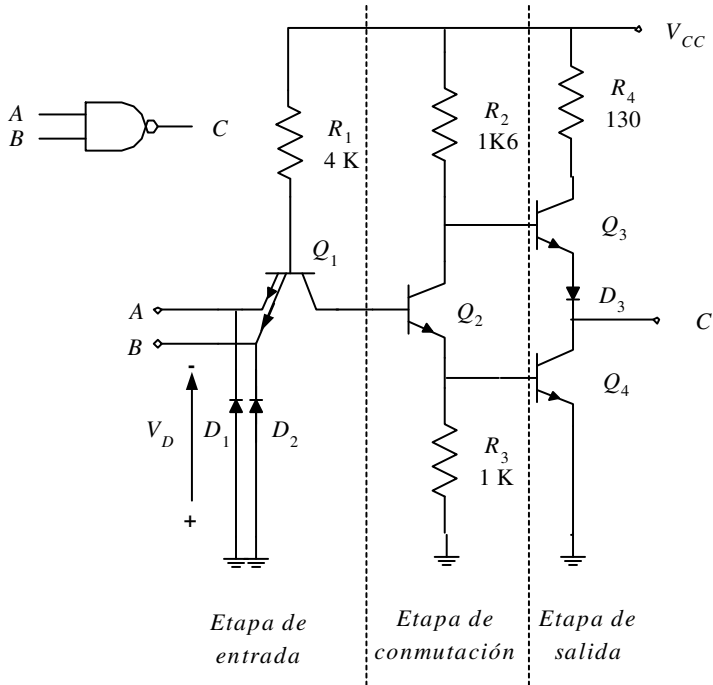


Figura 6.23. Estructura de la puerta NAND, representativa de la familia TTL estándar

- 2) **Etapa de conmutación:** constituida por el transistor  $Q_2$  y las resistencias de carga  $R_2$  y  $R_3$ , de colector y emisor respectivamente. Este circuito es un inversor de fase.
- 3) **Etapa de salida:** constituida por los transistores  $Q_3$ , denominado **transistor de salida pull-up** (tirar hacia arriba) que pone a nivel alto la salida y  $Q_4$  denominado **transistor de salida pull-down** (tirar hacia abajo) que pone a nivel bajo la salida, y que, junto con el diodo  $D_3$  forman la configuración denominada etapa de salida en **Totem-Pole** o par activo. Con esta configuración se consigue que la capacidad presente en la salida se cargue a través de  $Q_3$  (polarizado en la zona activa) y se descargue a través de  $Q_4$  (polarizado en saturación). El diodo  $D_3$  limita el pico de corriente que se produce en la transición en la que  $Q_3$  empieza a conducir estando aún  $Q_4$  en saturación.

La tensión de alimentación  $V_{CC}$  de la familia TTL estándar es de  $5\text{ V} \pm 5\%$ , es decir, debe estar comprendida entre  $V_{CCmin} = 4,75\text{ V}$  y  $V_{CCmax} = 5,25\text{ V}$ .

## 6.5.2 Transistor multiemisor

Una de las características de la familia TTL, y que ha dado el significado a dichas siglas, es la utilización de un **transistor de entrada multiemisor**, representado en la Figura 6.24, junto con su circuito equivalente aproximado desde el punto de vista de sus uniones semiconductoras.

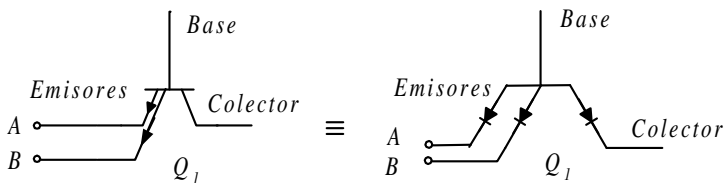


Figura 6.24. Transistor multiemisor: símbolo y circuito equivalente aproximado

## 6.5.3 Funcionamiento de la puerta TTL-Estándar. Análisis en continua

### 6.5.3.1 TODAS LAS ENTRADAS CON NIVEL ALTO

En la Figura 6.25 se representa el comportamiento de una puerta NAND de la familia TTL estándar con todas las entradas con nivel alto ( $2\text{ V} < V_{IH} < 5\text{ V}$ ). En estas condiciones las uniones base-emisor del transistor multiemisor  $Q_1$ , están inversamente

polarizadas, de forma que por los emisores circula una corriente de fugas  $I_{IH}$ , cuyo valor es, aproximadamente,  $40 \mu A$ .

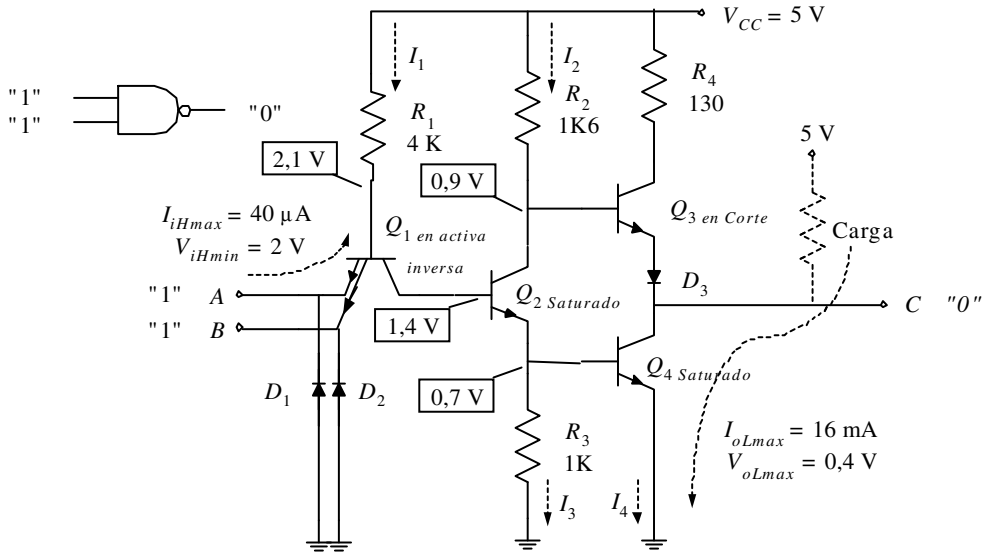


Figura 6.25. Comportamiento eléctrico de una puerta NAND de la familia TTL estándar con todas las entradas con nivel alto

El transistor  $Q_1$  trabaja en la región activa inversa, estando su unión base-colector polarizada directamente y por consiguiente dicha unión será equivalente a un diodo en conducción. Esta corriente que atraviesa la unión base-colector de  $Q_1$  circula por la base de  $Q_2$  provocando su conducción.

La conducción de  $Q_2$  aumenta el potencial de su emisor y proporciona la corriente de base a  $Q_4$  que le lleva a la saturación, originando en la salida  $C$  un nivel bajo. Asimismo, la conducción de  $Q_2$  disminuye el potencial de su colector hasta  $\approx 0,9$  V, poniendo en corte a  $Q_3$ .

Obsérvese que cuando el potencial de emisor de  $Q_2$  aumenta, el potencial de su colector disminuye y viceversa, de ahí que se le denomine a este transistor inversor de fase.

A continuación, se detalla el cálculo de las señales de interés para el circuito descrito anteriormente.

Cálculo de tensiones:

$$\begin{aligned}
 V_{B4} &= V_{E2} = V_{BE4} = 0,7 \text{ V} \\
 V_{CE2sat} &= 0,2 \text{ V} \\
 V_{C2} = V_{B3} &= V_{BE3} + V_{CE2sat} = 0,7 + 0,2 = 0,9 \text{ V} \\
 V_{B2} = V_{C1} &= V_{E2} + V_{BE2} = 0,7 + 0,7 = 1,4 \text{ V} \\
 V_{B1} = V_{C1} + V_{BC1} &= 1,4 + 0,7 = 2,1 \text{ V} \\
 V_{oL} = V_{C4sat} &\approx 0,2 \text{ V}
 \end{aligned} \tag{6.19}$$

Se demuestra que  $Q_3$  está en corte considerando que:

$$V_{BE3} = V_{C2} - (V_{oL} + V_{D3}) = 0,9 - (0,2 + 0,7) = 0 \text{ V} < 0,7 \text{ V} \tag{6.20}$$

Cálculo de intensidades:

$$\begin{aligned}
 I_1 &= \frac{V_{CC} - V_{B1}}{R_1} = \frac{5 \text{ V} - 2,1 \text{ V}}{4 \text{ k}\Omega} = 725 \mu\text{A} \\
 I_2 &= \frac{V_{CC} - V_{C2}}{R_2} = \frac{5 \text{ V} - 0,9 \text{ V}}{1 \text{ k}\Omega} = 2,56 \text{ mA} \\
 I_3 &= \frac{V_{B4}}{R_3} = \frac{0,7 \text{ V}}{1 \text{ k}\Omega} = 700 \mu\text{A} \\
 I_{B4} &= I_1 + I_2 - I_3 = 0,725 + 2,56 - 0,700 = 2,585 \text{ mA}
 \end{aligned} \tag{6.21}$$

La corriente de salida  $I_{oL}$  depende de la carga que se conecte, siendo la corriente máxima que soporta  $Q_4$  de  $I_{oLmax} = 16 \text{ mA}$ .

### 6.5.3.2 ALGUNA ENTRADA CON NIVEL BAJO

En la Figura 6.26 se representa el comportamiento de una puerta NAND de la familia TTL estándar con alguna de sus entradas a nivel bajo.

Cuando alguna de las entradas  $A$ ,  $B$  o ambas se encuentran a nivel bajo ( $0 \text{ V} < V_{iL} < 0,8 \text{ V}$ ), la corriente de entrada circula por la unión base-emisor del transistor  $Q_1$ , el cual entra en saturación. Considerando el valor máximo de la tensión que se puede aplicar a una entrada  $V_{iLmax} = 0,8 \text{ V}$ , se tiene que la tensión en el colector de  $Q_1$  es:

$$V_{C1max} = V_{B2max} = V_{iLmax} + V_{CE1sat} = 0,8 + 0,2 = 1 \text{ V} \tag{6.22}$$

El transistor  $Q_2$  pasa a corte, pues para su conducción es necesario que tenga un potencial en su base  $V_{B2}$  de  $1,4 \text{ V}$  y sólo tiene  $1 \text{ V}$ . Si  $Q_2$  en el instante anterior estaba en saturación la pequeña resistencia que presenta  $Q_1$  entre su colector-emisor favorece

la descarga de la capacidad base-emisor de  $Q_2$  ( $C_{BE2}$ ), ayudando a la transición del estado de saturación al de corte de  $Q_2$  más rápidamente. El **proceso de descarga** descrito proporciona la velocidad de conmutación característica de la familia TTL estándar.

Al pasar  $Q_2$  a corte, no circula corriente por  $R_3$  y por tanto  $Q_4$  también pasa a corte por tener su tensión de base-emisor nula. Por el mismo motivo el colector de  $Q_2$  se acerca al valor de la alimentación  $V_{CC}$  haciendo conducir a  $Q_3$ .

El transistor  $Q_3$  actúa de seguidor de emisor, proporcionando una baja impedancia y una corriente de salida con nivel alto  $I_{oH}$ . Esta corriente está limitada por la resistencia  $R_4$ , siendo la máxima de salida con nivel alto  $I_{oHmax} = - 400 \mu A$ .

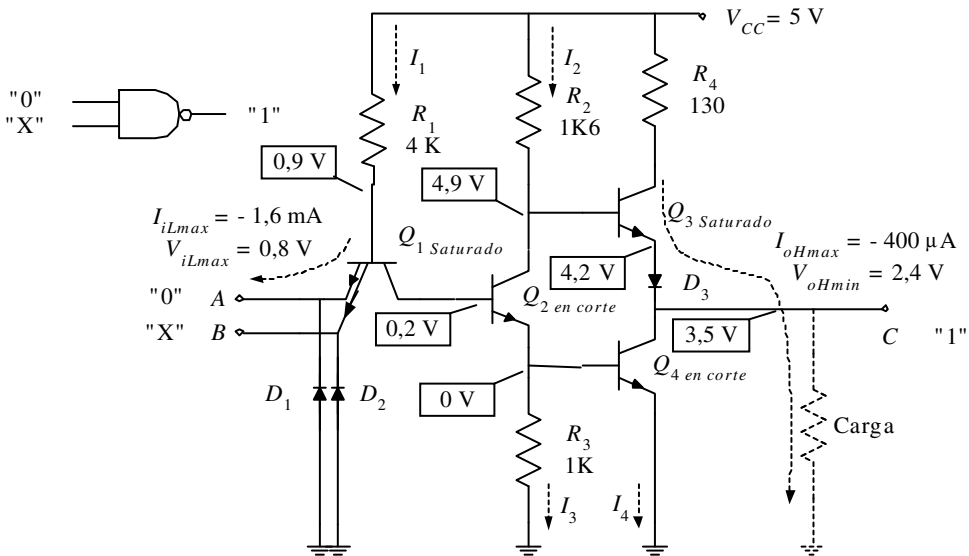


Figura 6.26. Comportamiento eléctrico de una puerta NAND de la familia TTL estándar con alguna entrada con nivel bajo

A continuación se detalla el cálculo del circuito descrito.

En la explicación anterior se ha indicado que,

$$\begin{aligned}
 V_{C1} &= V_{B2} = 1 \text{ V} \\
 V_{E2} &= V_{B4} = 0 \text{ V} \\
 V_{C2} &= V_{B3} \approx 4,90 \text{ V}
 \end{aligned}
 \tag{6.23}$$

El cálculo de  $V_{B1}$  considerando  $V_{iL} = 0,2 \text{ V}$  es:

$$V_{B1} = V_{iL} + V_{BE1} = 0,2 + 0,7 = 0,9 \text{ V} \quad [6.24]$$

Cálculo de intensidades:

$$I_1 = I_{iL} = \frac{V_{CC} - V_{B1}}{R_1} = \frac{5 \text{ V} - 0,9 \text{ V}}{4 \text{ K}\Omega} = 1,025 \text{ mA}$$

$$I_2 = \frac{V_{CC} - V_{C2}}{R_2} = \frac{5 \text{ V} - 4,90 \text{ V}}{1 \text{ K}6 \Omega} \approx 60 \mu\text{A} \quad [6.25]$$

$$I_3 = I_4 = 0 \mu\text{A}$$

La tensión de salida con nivel alto:

$$V_{oH} = V_{C4} = V_{C2} - V_{BE3} - V_{D1} = 4,9 - 0,7 - 0,7 = 3,5 \text{ V} \quad [6.26]$$

$V_{oH}$  depende de la temperatura y de la carga conectada a la salida pudiendo variar desde 3,5 V hasta 4,5 V.

### 6.5.3.3 ENTRADAS SIN CONEXIÓN

Cuando las entradas  $A$  y  $B$  se dejan sin conectar, la unión base-colector de  $Q_1$  quedaría polarizada directamente y sería igual al caso estudiado anteriormente en el que todas las entradas estaban a nivel alto.

Como conclusión, cuando una entrada en TTL no se conecta (se encuentra al aire) se interpreta como si estuviera a nivel alto.

#### PROBLEMA RESUELTO 6-6



Simular el funcionamiento de una puerta NAND TTL-Estándar utilizando para ello el programa de simulación *Electronics Workbench*, comprobando los resultados analíticos descritos en el apartado 6.5.3 Funcionamiento de la puerta TTL-Estándar. Análisis en continua.

#### Solución:

Se introduce en el simulador *Electronics Workbench* el circuito, que se ha analizado anteriormente en la Figura 6.23, correspondiente a la estructura de la puerta NAND representativa de la familia TTL-Estándar.

En dicho circuito se deben comprobar los valores de tensión en los nodos y las corrientes en sus componentes, para los diferentes niveles lógicos presentes en

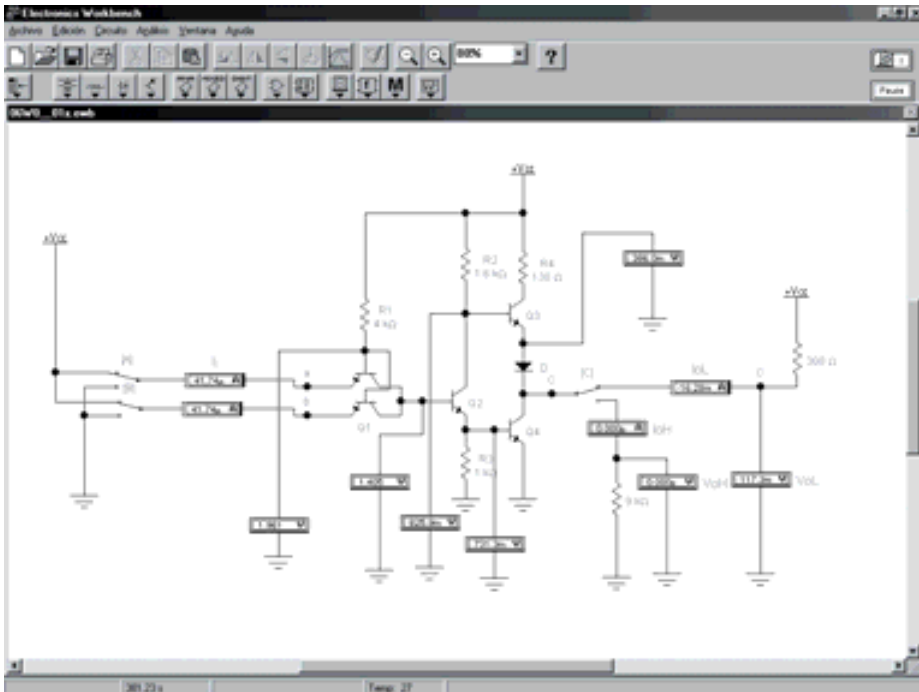


sus entradas. En la Figura 6.27 se muestra el caso en el que todas las entradas de la puerta NAND están a nivel alto.

Como puede observarse el transistor multiemisor  $Q_1$  se ha simulado con el montaje de dos transistores.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap06\Ewb5\06W0\_\_01.ewb**



*Figura 6.27. Simulación de una puerta NAND de la familia TTL estándar con todas las entradas a nivel alto*

Pulsando en el teclado [A] o [B] se pueden cambiar las entradas a nivel bajo obteniendo la simulación que se muestra en la Figura 6.28. También se debe pulsar [C] para conectar la carga entre la salida y masa.

Se deja al lector que compruebe cómo los resultados obtenidos con el programa de simulación son similares a los obtenidos de forma teórica en las expresiones [6.19] a [6.26].

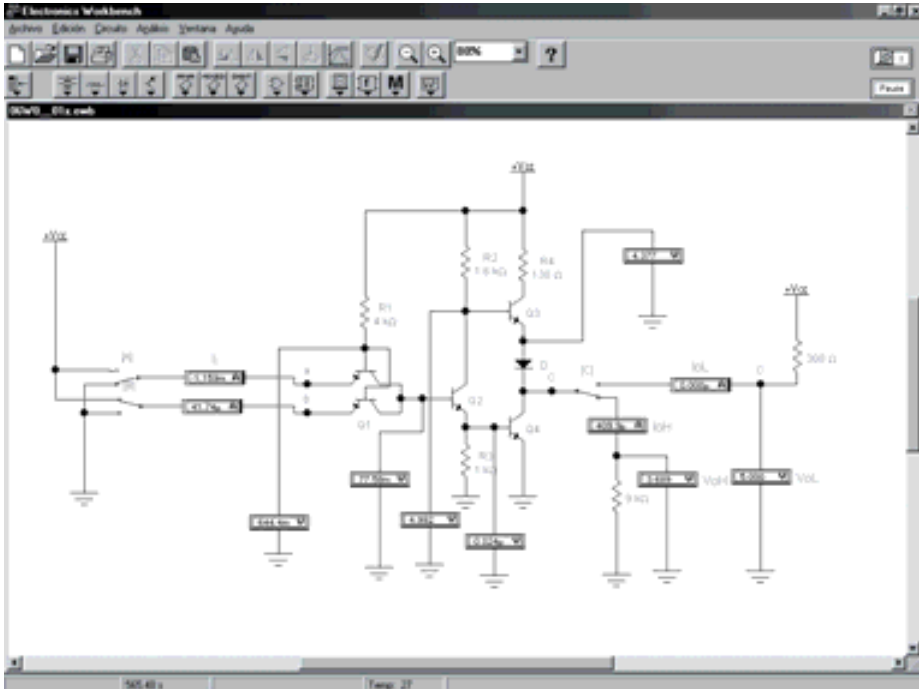


Figura 6.28. Simulación de una puerta NAND de la familia TTL estándar con alguna entrada con nivel bajo

## 6.5.4 Consideraciones de diseño de las resistencias

Se debe tener en cuenta que si  $R_1$  es muy pequeña, aumenta la corriente de entrada con nivel bajo cargando en exceso las salidas de las puertas que excitan a estas entradas. Esto también origina un aumento del consumo y una disminución del *fan out* de las puertas excitadoras. Por el contrario, si  $R_1$  es grande la corriente de base de  $Q_2$  sería pequeña, aumentando el tiempo de carga de la capacidad  $C_{BE}$  de  $Q_2$ , y por consiguiente disminuyendo la velocidad de la puerta.

Con  $R_2$  se prefija la corriente necesaria en la base de  $Q_4$  que permite su saturación.

El objeto de  $R_3$  es proporcionar un camino para eliminar la carga almacenada en la base de  $Q_4$  cuando este transistor pasa de saturación a corte. Desde este punto de vista su valor tendría que ser bajo, pero ello conlleva un mayor consumo; por lo que se opta por un valor intermedio como solución de compromiso entre ambas condiciones contrapuestas.

La resistencia  $R_4$  limita los picos de corriente que se producen en las transiciones de la salida de nivel bajo a nivel alto (flanco de subida o positivo) en las que  $Q_3$

empieza a conducir estando aún  $Q_4$  en saturación. También limita la corriente de salida en el nivel alto  $I_{oH}$  cuando  $Q_3$  conduce y  $Q_4$  está en corte.

En la Tabla 6.2 se resumen los efectos que tienen las resistencias sobre las características de la puerta.

*Tabla 6.2. Consideraciones de diseño de las resistencias de una puerta NAND*

<i>Componente</i>	<i>Parámetro que controla</i>
$R_1$	Limita la corriente de entrada a nivel bajo $I_{iL}$ e influye sobre el <i>fan out</i> de las puertas excitadoras. Limita el consumo.
$R_2$	Asegura la saturación de $Q_4$ proporcionando $V_{oL}$ adecuado.
$R_3$	Elimina la carga de la base de $Q_4$ en saturación, mejorando la velocidad de la puerta. Limita el consumo.
$R_4$	Limita la disipación de potencia en las transiciones positivas de la salida. Limita la corriente de salida en el nivel alto $I_{oH}$ .

## 6.5.5 Características funcionales de la familia TTL-Estándar

### 6.5.5.1 CARACTERÍSTICAS ESTÁTICAS DE LA FAMILIA TTL-ESTÁNDAR

Las hojas de características de los fabricantes proporcionan la información necesaria sobre los parámetros estudiados, siendo imprescindible su conocimiento en el diseño de sistemas digitales.

Como ejemplo, en el programa de Fairchild Semiconductor para **Data Sheets** que se incluye en el CDRom#2 que acompaña a este libro se pueden buscar las características de los circuitos comerciales que suministra dicho fabricante, pudiendo comparar los valores suministrados para sus circuitos lógicos con los valores típicos que se indican a lo largo de este capítulo.

En la Tabla 6.3 se muestran las características típicas de la familia lógica TTL-Estándar. Se trata de una puerta NAND de dos entradas, correspondiente al circuito integrado 7400.

Tabla 6.3. Características de la familia lógica TTL estándar

<b>7400 Cuádruple puertas NAND de dos entradas</b>			
<i>Parámetro</i>	<i>Valor</i>	<i>Unidades</i>	<i>Condiciones</i>
$V_{oHmin}$	2,4	V	$V_{CCmin}$
$V_{oLmax}$	0,4	V	$V_{CCmin}$
$V_{iHmin}$	2	V	$V_{CCmax}$
$V_{iLmax}$	0,8	V	$V_{CCmax}$
$V_{NMH}$	400	mV	
$V_{NML}$	400	mV	
$I_{oHmax}$	- 400	$\mu$ A	$V_{CCmin}$
$I_{oLmax}$	16	mA	$V_{CCmin}$
$I_{iHmax}$	40	$\mu$ A	$V_{CCmax}$ , $V_{iH} = 2,4$ V
$I_{iLmax}$	-1,6	mA	$V_{CCmax}$ , $V_{iL} = 0,4$ V
$I_{Osmín}$	- 20	mA	$V_{CCmax}$
$I_{Osmáx}$	- 100	mA	$V_{CCmax}$
$I_{CCH}$	4	mA	$V_{CCmax}$
$I_{CCL}$	11	mA	$V_{CCmax}$
$t_{pLH}$	11	ns	
$t_{pHL}$	7	ns	
$t_{TLH}$	12	ns	
$t_{THL}$	5	ns	
$f_{max}$	25	MHz	
<i>Fan out (referido a TTL estándar)</i>	10	u.c.	
$P_D$	40	mW	$V_{CCmax}$ , 10 mW/puerta
<i>Producto <math>P_D \cdot t_{pD}</math></i>	100	pJ	

Gráficamente también se pueden obtener todos los datos del comportamiento estático de una puerta mediante las curvas de: entrada, salida y transferencia.

- **Características de entrada**

En la Figura 6.29 se representan las curvas típicas de la característica de entrada de una puerta TTL estándar.

El fabricante garantiza que las entradas presentarán unos niveles máximos de corriente a nivel bajo de  $I_{iLmax} = -1,6 \text{ mA}$  (signo negativo debido a que es una corriente saliente) y a nivel alto de  $I_{iHmax} = 40 \mu\text{A}$ . Véase también la Tabla 6.3.

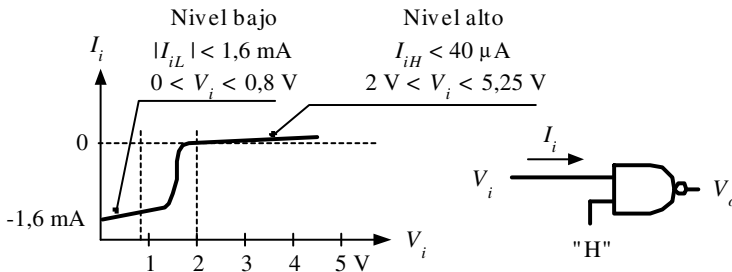


Figura 6.29. Curva típica de la característica de entrada de una puerta TTL estándar

## PROBLEMA RESUELTO 6-7



Obtener la curva característica de entrada de una puerta NAND TTL-Estándar actuando como inversor, utilizando para ello el programa de simulación *Electronics Workbench*.

### Solución:

Para realizar esta simulación se debe crear, en el simulador, el circuito que se muestra en la Figura 6.30. Dicho circuito es equivalente al representado anteriormente en la Figura 6.23, que corresponde a la estructura de la puerta NAND representativa de la familia TTL-Estándar. Para este circuito se realiza en *Electronics Workbench* un subcircuito que se denomina TTL.

Compruébese en el osciloscopio que se representa la curva característica de entrada de una puerta inversora TTL estándar, como se muestra en la Figura 6.30. Se debe observar que debido a las referencias de masa que son necesarias tomar en los canales del osciloscopio, el eje de representación de abscisas se encuentra invertido.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap06\Ewb5\06W0\_04.ewb**

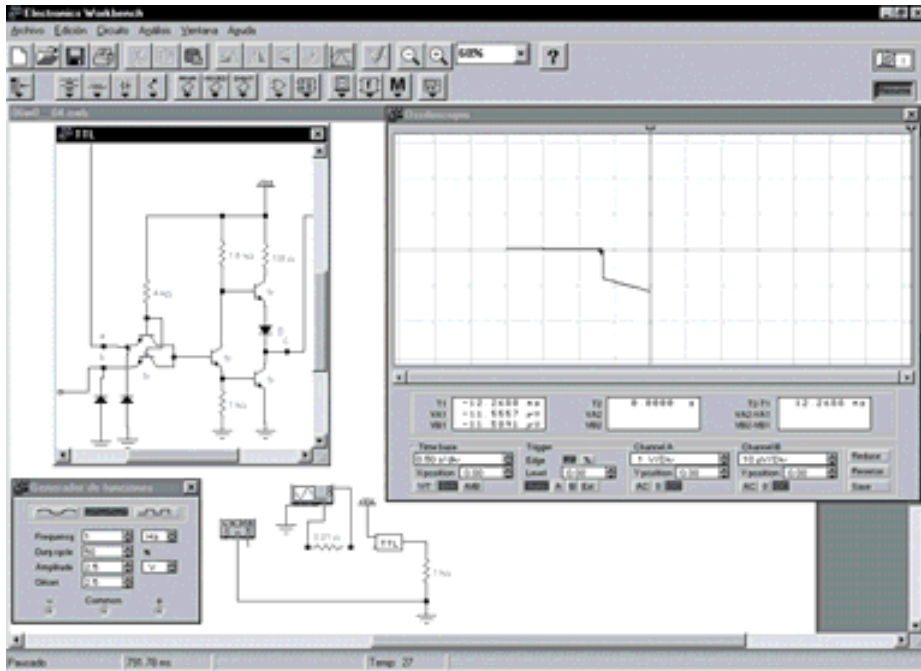


Figura 6.30. Simulación de la característica de salida a nivel bajo de una puerta NAND TTL-Estándar actuando como inversor

- **Características de salida**

En la Figura 6.31 se representan las curvas típicas de la característica de salida de una puerta TTL estándar según el nivel lógico que presente la salida.

En la curva de **salida a nivel bajo** se aprecia cómo la tensión de salida  $V_{OL}$  aumenta con la carga. El cambio brusco de pendiente en la curva, que indica un gran aumento de  $V_{OL}$ , se produce cuando el transistor  $Q_4$  deja de trabajar en la zona de saturación.

En la curva de **salida a nivel alto** se obtiene el parámetro de intensidad de salida en cortocircuito.

El fabricante garantiza que la salida puede absorber a nivel bajo una corriente de  $I_{OLmax} = 16 \text{ mA}$  y suministrar a nivel alto una corriente de  $I_{OHmax} = -400 \mu\text{A}$  (Tabla 6.3). Con estos datos de corrientes de salida y los valores de corriente de entrada obtenidos en las curvas características de entrada se puede calcular el *fan-out*, tal como se indica en la expresión [6.27].

Como puede comprobarse, este cálculo de *fan-out* coincide con el indicado por el fabricante en la Tabla 6.3.

$$fan - out_H = \frac{|I_{oHmax}|}{|I_{iHmax}|} = \frac{|-400 \mu A|}{|40 \mu A|} = 10$$

$$fan - out_L = \frac{|I_{oLmax}|}{|I_{iLmax}|} = \frac{|16 \text{ mA}|}{|-1,6 \text{ mA}|} = 10$$
[6.27]

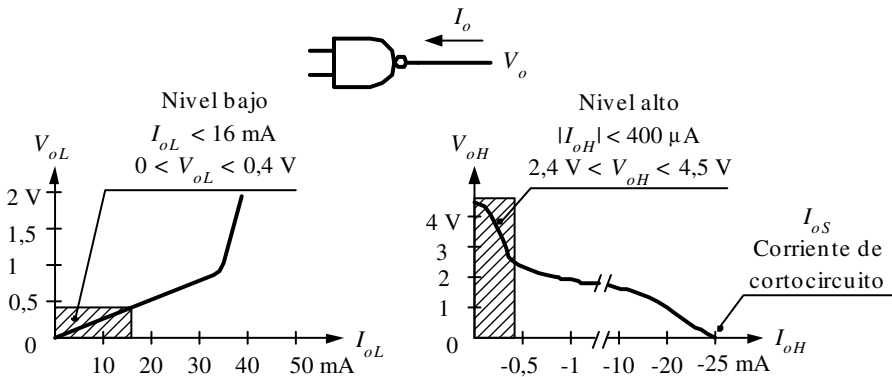


Figura 6.31. Curvas típicas de la característica de salida de una puerta TTL estándar

### PROBLEMA RESUELTO 6-8



Obtener la curva característica de salida de una puerta NAND TTL-Estándar actuando como inversor, utilizando para ello el programa de simulación *Electronics Workbench*.

#### Solución:

Para obtener la curva característica de salida de una puerta NAND TTL-Estándar, se van a simular dos montajes: uno para conseguir dicha característica con la salida a nivel alto como se muestra en la Figura 6.32 y otro para conseguir la característica con la salida a nivel bajo como se muestra en la Figura 6.33. En ambos montajes se introduce en el programa de simulación *Electronics Workbench* el circuito que se ha representado en la Figura 6.23 correspondiente a la estructura de la puerta NAND representativa de la familia TTL-Estándar, realizando un subcircuito denominado TTL.

Utilizando el osciloscopio, se puede comprobar la curva típica de la característica de salida a nivel bajo y a nivel alto de una puerta inversora TTL estándar, como se muestra en la Figura 6.32 y en la Figura 6.33, respectivamente. Se debe observar, en ambas características, que debido a las

referencias de masa en los canales del osciloscopio, algunos de los ejes de representación se encuentran invertidos.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema, representado en la Figura 6.32, es la que se indica a continuación:

**D:\Ejemplos\Cap06\Ewb5\06W0\_\_03.ewb**

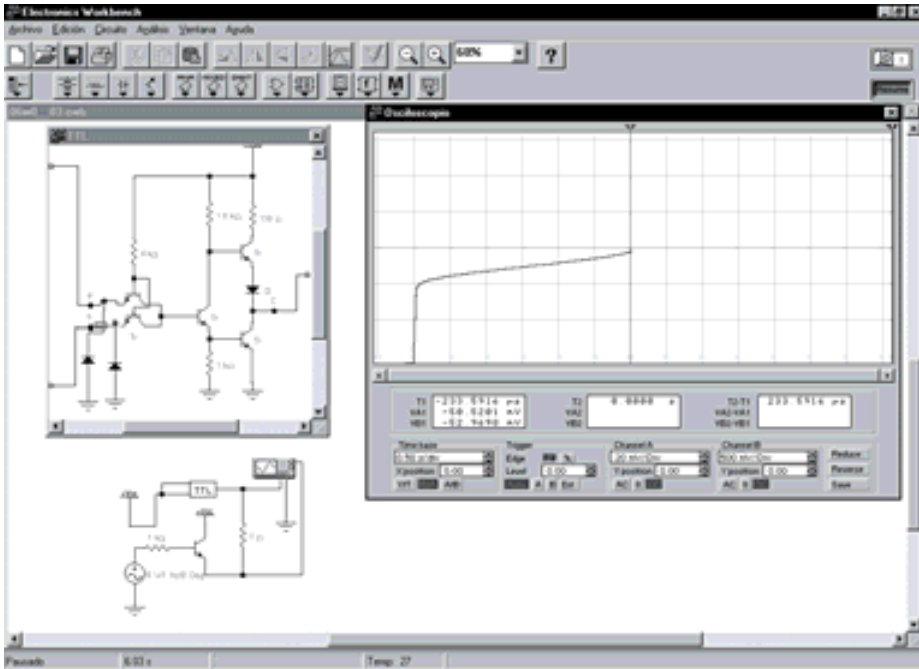


Figura 6.32. Simulación de la característica de salida a nivel alto de una puerta NAND TTL-Estándar actuando como inversor

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema, representado en la Figura 6.33, es la que se indica a continuación:

**D:\Ejemplos\Cap06\Ewb5\06W1\_\_03.ewb**

- **Características de transferencia**

En la Figura 6.34 se representan las curvas típicas de la característica de transferencia de una puerta inversora TTL estándar para tres temperaturas diferentes:  $-55\text{ }^{\circ}\text{C}$ ,  $25\text{ }^{\circ}\text{C}$  y  $75\text{ }^{\circ}\text{C}$ .

En esta curva se pueden obtener los márgenes o perfiles de tensiones de entrada y salida de los niveles lógicos bajo y alto ( $V_{iLmax}=0,8\text{ V}$ .;  $V_{iHmin}=2\text{ V}$ .;  $V_{oLmax}=0,4\text{ V}$ .;  $V_{oHmin}=2,4\text{ V}$ ). Estos valores también se pueden ver en la Tabla 6.3.



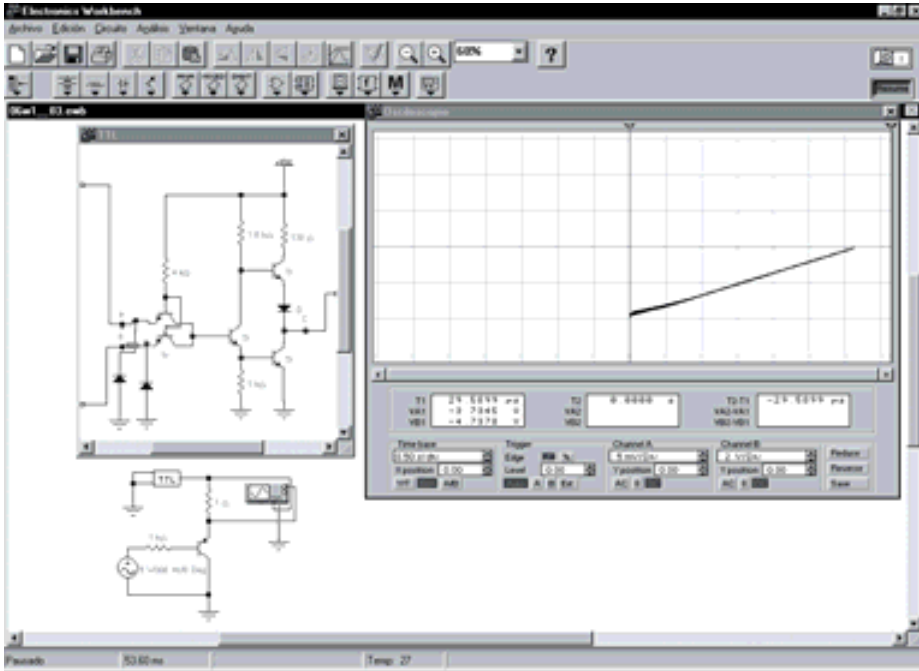


Figura 6.33. Simulación de la característica de salida a nivel bajo de una puerta NAND TTL-Estándar actuando como inversor

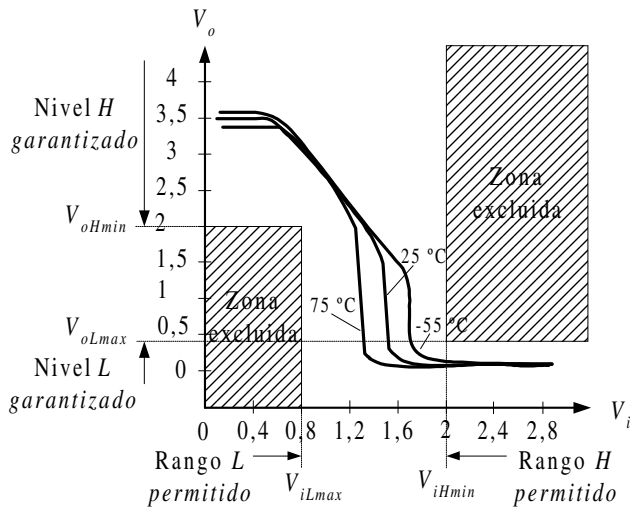


Figura 6.34. Curvas típicas de la característica de transferencia de una puerta TTL estándar

## PROBLEMA RESUELTO 6-9



Obtener la función de transferencia de una puerta NAND TTL-Estándar actuando como inversor, utilizando para ello el programa de simulación *Electronics Workbench*, y comprobando la teoría descrita anteriormente.

### Solución:

Se debe realizar el montaje que muestra en la Figura 6.35. Para ello se introduce en el programa de simulación *Electronics Workbench* el circuito que se ha representado en la Figura 6.23 correspondiente a la estructura de la puerta NAND representativa de la familia TTL-Estándar, realizando con él un subcircuito denominado TTL.

Mediante el osciloscopio se puede ver la curva típica de la característica de transferencia de una puerta inversora TTL estándar, como se muestra en la Figura 6.35.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap06\Ewb5\06W0\_\_02.ewb**

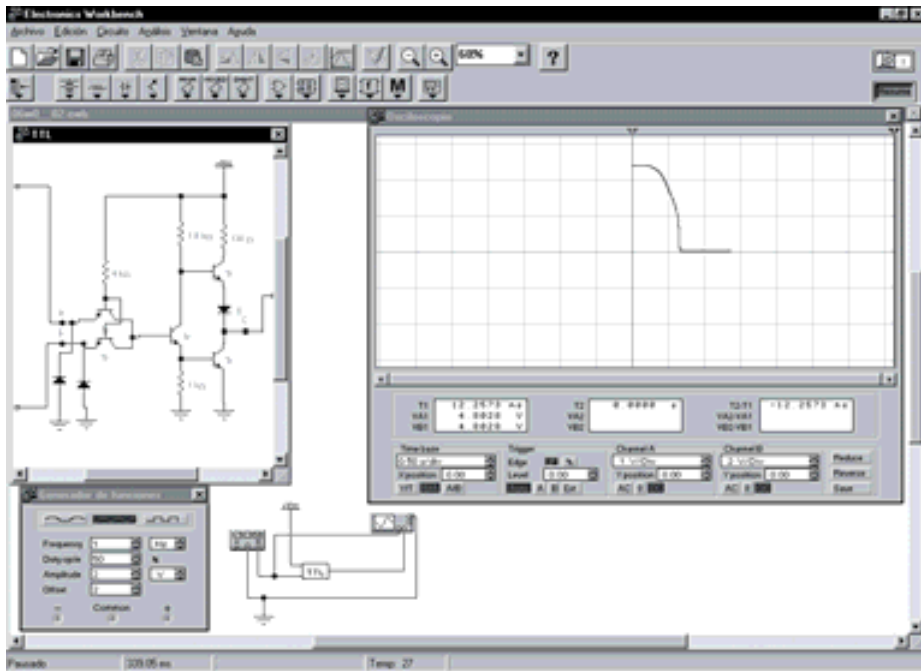


Figura 6.35. Simulación de la función de transferencia de una puerta NAND TTL-Estándar actuando como inversor

- **Inmunidad al ruido**

Con estos datos de tensión se puede calcular la inmunidad al ruido con nivel bajo  $V_{NML}$  y alto  $V_{NMH}$ .

$$\begin{aligned} V_{NML} &= V_{iLmax} - V_{oLmax} = 0,8 - 0,4 = 0,4 \text{ V} \\ V_{NMH} &= V_{oHmin} - V_{iHmin} = 2,4 - 2 = 0,4 \text{ V} \end{aligned} \quad [6.28]$$

Compruébese este cálculo de  $V_{NM} = 0,4 \text{ V}$  con el indicado por el fabricante en la Tabla 6.3.

- **Disipación de potencia**

A partir de los datos proporcionados por el fabricante (Tabla 6.3) donde  $I_{CCH} = 4 \text{ mA}$  e  $I_{CCL} = 11 \text{ mA}$  y para un valor de alimentación máxima de  $V_{CCmax} = 5,25 \text{ V}$ , se puede calcular la disipación de potencia, según se muestra en [6.29].

$$\begin{aligned} I_{CC} &= \frac{I_{CCH} + I_{CCL}}{2} = \frac{4 \text{ mA} + 11 \text{ mA}}{2} = 7,5 \text{ mA} \\ P_D &= I_{CC} V_{CC} = 7,5 \text{ mA} \cdot 5,25 \text{ V} \approx 40 \text{ mW} \end{aligned} \quad [6.29]$$

Al tener el circuito integrado 7400 cuatro puertas el consumo es de 10 mW/puerta.

Compruébese este cálculo de  $P_D$  con el indicado por el fabricante que se ha dado en la Tabla 6.3.

### 6.5.5.2 CARACTERÍSTICAS DINÁMICAS DE LA FAMILIA TTL-ESTÁNDAR

- **Retardos de propagación y de transición**

De la Tabla 6.3 se obtiene el valor de los retardos indicados por el fabricante de una puerta TTL estándar.

$$\begin{aligned} T_{pLH} &= 11 \text{ ns} \\ T_{pHL} &= 7 \text{ ns} \end{aligned}$$

Por consiguiente, el retardo de propagación medio vendrá dado por:

$$t_{pD} = \frac{t_{pLH} + t_{pHL}}{2} = \frac{11 \text{ ns} + 7 \text{ ns}}{2} \approx 10 \text{ ns} \quad [6.30]$$

- **Frecuencia máxima de funcionamiento**

La frecuencia máxima de funcionamiento viene dada por la expresión [6.31].

$$f_{max} = \frac{1}{4 t_{pD}} = \frac{1}{4 \cdot 10 \text{ ns}} = 25 \text{ MHz} \quad [6.31]$$

Compruébese este cálculo de la frecuencia máxima de funcionamiento con el indicado por el fabricante en la Tabla 6.3.

- **Producto potencia disipada-retardo de propagación**

Este producto viene dado por la expresión [6.32].

$$P_D t_{pD} = 10 \text{ mW} \cdot 10 \text{ ns} = 100 \text{ pJ} \quad [6.32]$$

Compruébese este cálculo del producto potencia disipada-retardo de propagación con el indicado por el fabricante en la Tabla 6.3.

## 6.5.6 Representación de las características mediante perfiles de entrada y salida

Otra forma práctica de definir las principales características estáticas de una familia lógica es mediante los perfiles de entrada y salida.

Por ejemplo, la representación de las características estáticas de la familia TTL estándar por este método se puede ver en la Figura 6.36.

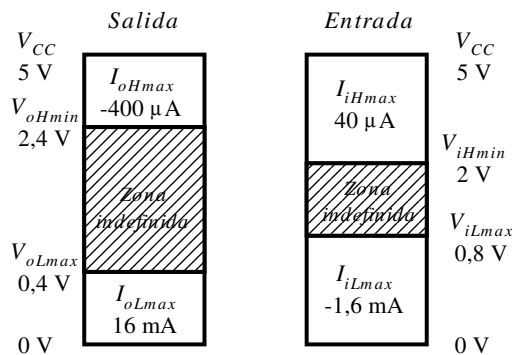


Figura 6.36. Perfiles de entrada y salida de una puerta TTL estándar

Mediante los datos incluidos en esta representación y aplicando las expresiones correspondientes se puede calcular: la inmunidad al ruido  $V_{NM}$ , el *fan-in*, el *fan-out*, la compatibilidad con otros dispositivos lógicos, etc.

## 6.6 SUBFAMILIAS TTL

Las subfamilias TTL se han desarrollado mejorando las características de la familia TTL estándar, sobre todo en lo que a consumo y velocidad se refiere.

### 6.6.1 TTL de baja potencia (LTTL, serie 54L/74L)

La subfamilia de bajo consumo *Low-power-LTTL*, se diferencia de la TTL estándar en que el valor de las resistencias es mayor. En este caso al valor de dichas resistencias es el siguiente:

$$\begin{aligned} R_1 &= 40 \text{ k}\Omega & R_3 &= 12 \text{ k}\Omega \\ R_2 &= 20 \text{ k}\Omega & R_4 &= 500 \text{ }\Omega \end{aligned}$$

Como resultado se obtiene una menor corriente de alimentación  $I_{CC}$  y por tanto menor consumo. Esta disminución de la corriente presente en el circuito tiene la desventaja de reducir la velocidad de conmutación al ser mayor la constante de tiempo de carga y descarga de las capacidades parásitas.

Esta subfamilia es apropiada en aquellas aplicaciones en las que el menor consumo es más importante que la velocidad de conmutación.

#### Características:

Según se puede ver en la Tabla 6.4, las principales características de este tipo de puertas son las siguientes:

Consumo típico por puerta, 1 mW.

Tiempo de propagación, 33 ns.

Frecuencia operativa, 8 MHz.

### 6.6.2 Puerta TTL tipo Schottky (STTL, serie 54S/74S)

Cuando un transistor pasa de saturación a corte se produce un retardo debido a la necesidad de evacuar el exceso de carga almacenada en la región de base durante la saturación. Una solución utilizada para aumentar la velocidad es evitar que los transistores trabajen en la zona de saturación, impidiendo que su unión base-colector ( $B-C$ ) se polarice directamente,  $V_{BCsat} \approx 0,5 \text{ V}$ . Para ello, como se muestra en la Figura 6.37, se utilizan diodos *Schottky* conectados durante el proceso de fabricación entre  $B-C$ , llamándose al conjunto transistor con barrera *Schottky* o simplemente **transistor Schottky**. El diodo *Schottky* se caracteriza por ser rápido y presentar una caída de tensión en sus extremos ( $V_D$ ) de 0,2 V a 0,3 V inferior a la necesaria en la unión  $B-C$  para que el transistor se sature, dando origen a la lógica no saturable.

En el transistor *Schottky*, representado en la Figura 6.37, se verifica:

$$\begin{aligned}
 V_D &= 0,2 V \\
 V_{BE} &= 0,7 V \\
 V_{CEsat} &= 0,2 V \\
 V_{CE} &= V_{BE} - V_D = 0,7 V - 0,2 V = 0,5 V \\
 V_{CEsat} &< V_{CE} \quad (\text{transistor no saturado})
 \end{aligned}
 \tag{6.33}$$

En el transistor *Schottky* gran parte de la corriente  $I_1$  se deriva al colector a través del diodo *Schottky*,  $I_D$ , y sólo una porción de  $I_1$  circula por la base del transistor,  $I_B$ , evitando el exceso de carga en la región de base que, como anteriormente se ha indicado, produce un retardo al pasar el transistor de saturación a corte.

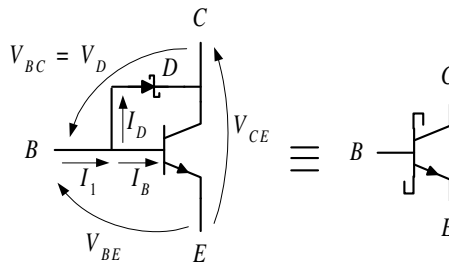


Figura 6.37. Conexión y símbolo del transistor *Schottky*

En la Figura 6.38 se muestra el circuito de una puerta NAND de tipo STTL. En dicha figura se han sombreado las diferencias o evolución de la subfamilia STTL respecto a TTL estándar, representándose las siguientes modificaciones:

- Se ha **disminuido el valor de las resistencias** para obtener mayores corrientes de carga de las capacidades parásitas de los transistores, ofreciendo una mayor velocidad de conmutación, pero en contrapartida presentando un mayor consumo.
- Se ha modificado la configuración de salida incluyendo el **par Darlington** formado por los transistores  $Q_4$  y *Schottky*  $Q_5$  que proporcionan una  $\beta_T$  igual al producto de las  $\beta_4$  y  $\beta_5$  individuales, como se muestra en la expresión [6.34]. Con ello se obtiene: una impedancia de salida menor que en la TTL estándar, una mayor velocidad de conmutación y un mejor *fan-out*.

La mayor velocidad de conmutación se debe a que cuando  $Q_2$  entra en conducción (nivel alto en las entradas) evacua rápidamente la carga en la base de  $Q_4$  facilitando la transición al corte del par *Darlington* a la vez que se incrementa la corriente en la base de  $Q_6$ . Este incremento de corriente en la base de  $Q_6$  le permite alcanzar más rápidamente una conducción cerca de la

saturación (sin llegar a ella por ser de tipo *Schottky*) obteniéndose un reducido retardo en la transición a nivel bajo en la salida.

$$\beta_r = \beta_4 \beta_5 \tag{6.34}$$

El transistor  $Q_5$ , además, cumple la misma función del diodo presente en la salida de TTL estándar. La resistencia  $R_5$  limita el exceso de corriente de  $Q_5$  evitando que este transistor adquiriera demasiada carga almacenada en la base durante la saturación.

- La resistencia  $R_3 = 1\text{ k}\Omega$  de la puerta TTL estándar de la Figura 6.23 se ha sustituido en la subfamilia STTL por la red  $R_3, R_4$  y  $Q_3$  denominada **red de encuadre** o *Squaring Network* que presenta una impedancia mucho menor que  $1\text{ k}\Omega$ , facilitando la evacuación de carga de base de  $Q_6$ . Esta red también proporciona una característica de transferencia más simétrica haciendo que la pendiente de la función de transferencia sea más pronunciada; de ahí el nombre de red de encuadre.
- El transistor  $Q_6$  al ser de tipo *Schottky* eleva la tensión de salida a nivel bajo  $V_{iH}$  respecto de la obtenida en el caso de la TTL estándar.

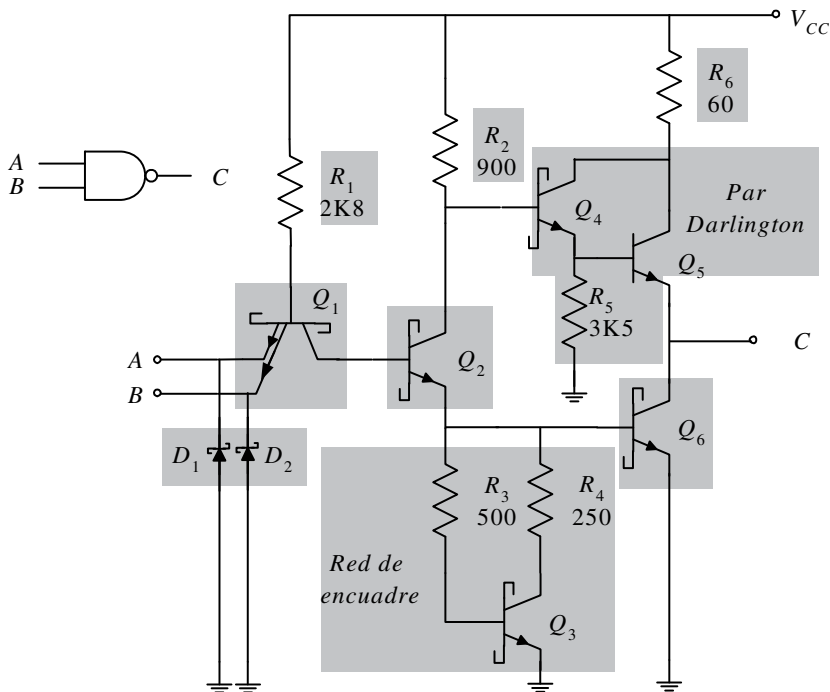


Figura 6.38. Puerta NAND STTL

**Características:**

Según se puede ver en la Tabla 6.4, las principales características de este tipo de puertas son las siguientes:

Consumo típico por puerta, 19 mW.

Tiempo de propagación, 3 ns.

Frecuencia operativa, 95 MHz.

### 6.6.3 TTL Schottky de bajo consumo (LSTTL, serie 54LS/74LS)

La subfamilia LSTTL (*Low power Schottky*) es una evolución de la STTL con una reducción de consumo. Como se muestra en la Figura 6.39, el circuito de la subfamilia LSTTL es prácticamente igual al de la subfamilia STTL.

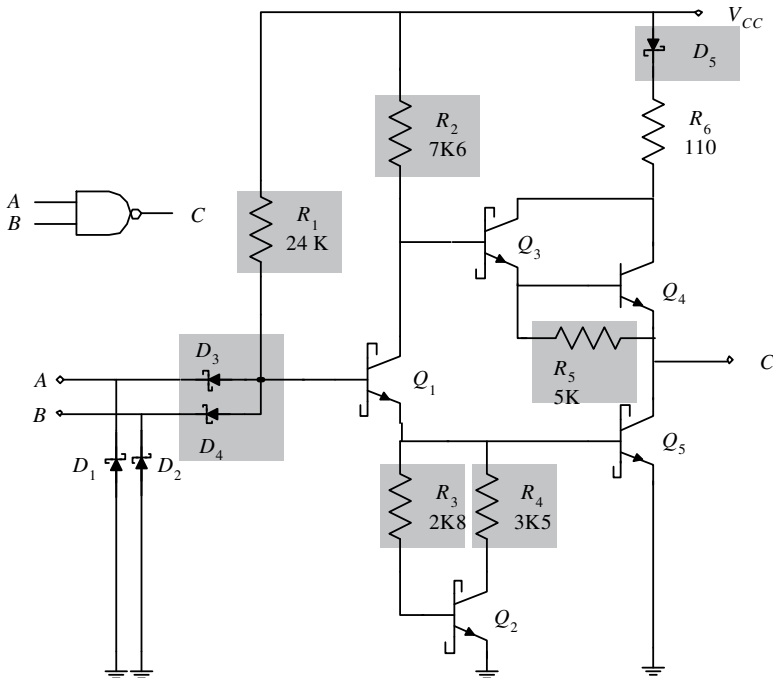


Figura 6.39. Puerta NAND LSTTL

En dicha figura se han sombreado las diferencias o evolución de la subfamilia LSTTL respecto a STTL, representándose las siguientes modificaciones:



- Aumento de las **resistencias  $R_1$ ,  $R_2$ ,  $R_3$  y  $R_4$**  para disminuir la corriente de alimentación y, por tanto, el consumo.
- Sustitución del transistor multiemisor  $Q_1$  por **una etapa de entrada de lógica de diodos *Schottky*** que proporciona: mayor velocidad de conmutación, una impedancia de entrada alta tanto a nivel bajo como alto y una capacidad parásita de entrada pequeña (aproximadamente de 3 pF).
- La **resistencia  $R_5$** , conectada entre base y emisor de  $Q_4$ , en lugar de la conexión de base y tierra en STTL tiene una caída de tensión menor de 0,7 V, mientras la carga en la salida sea pequeña, haciendo que  $Q_4$  no conduzca. Esto produce una tensión de salida  $V_{oH}$  más próxima a la tensión de alimentación  $V_{CC}$  cuando la salida requiere muy poca corriente a nivel alto.
- El **diodo  $D_5$**  situado en el colector del par *Darlington* permite realizar un *pull-up* que, como se estudia posteriormente, consiste en conectar una resistencia entre la salida y una tensión  $V_{DD}$  (comprendida entre  $V_{CC}$  y +10 V), en lugar de la alimentación fija  $V_{CC}$  de la puerta, proporcionando un valor distinto de tensión de salida a nivel alto próxima a  $V_{DD}$ . Esto permite la interconexión y compatibilidad con otras familias cuyo rango de tensiones en el nivel lógico alto contiene el valor  $V_{DD}$ .

La subfamilia LSTTL es muy utilizada por proporcionar un buen compromiso entre consumo y velocidad, es decir, bajo producto de consumo por tiempo de propagación. También se caracteriza por su baja generación de ruido y alta fiabilidad.

#### **Características:**

Según se puede ver en la Tabla 6.4, las principales características de este tipo de puertas son las siguientes:

Consumo típico por puerta, 2 mW.

Tiempo de propagación, 8 ns.

Frecuencia operativa, 33 MHz.

Se fabrican también para esta subfamilia **otras configuraciones de entradas**, conocidos como diodos *cluster* y transistores PNP.

En la Figura 6.40 se muestra la **configuración de entrada en diodo *cluster*** (tres diodos agrupados  $D_{A2}$ ,  $D_{A3}$ ,  $D_{A4}$ ). Cuando ambas entradas A y B están a nivel alto, los diodos  $D_{A2}$  y  $D_{B2}$  no conducen y los  $D_{A3}$  y  $D_{B3}$  conducen, circulando corriente y aplicando un nivel alto en la base del transistor  $Q_1$ . Cuando una o ambas entradas A y B están a nivel bajo, los diodos  $D_{A2}$  y/o  $D_{B2}$  conducen y los  $D_{A3}$  y/o  $D_{B3}$  no conducen, no circulando corriente y aplicando un nivel bajo de tensión en la base del transistor  $Q_1$  a través de  $D_{A4}$  y/o  $D_{B4}$ . Por consiguiente, se concluye que esta configuración de diodo *cluster* realiza la función AND.

En la Figura 6.41 se muestra la **configuración de entrada en transistor PNP**. Se aprecia que se han sustituido los diodos  $D_{A2}$  y  $D_{B2}$  de la configuración en diodo *cluster*

por transistores PNP. El hecho de incorporar transistores reduce la corriente de entrada a nivel bajo en proporción a la ganancia en corriente  $\beta$  del transistor PNP. Las consideraciones de su funcionamiento son las mismas que las indicadas en la configuración en diodo *cluster*, teniendo en cuenta que los diodos  $D_{A2}$  y  $D_{B2}$  han sido sustituidos por las uniones B-E de los transistores PNP  $Q_{A2}$  y  $Q_{B2}$ , realizando también el conjunto la función AND.

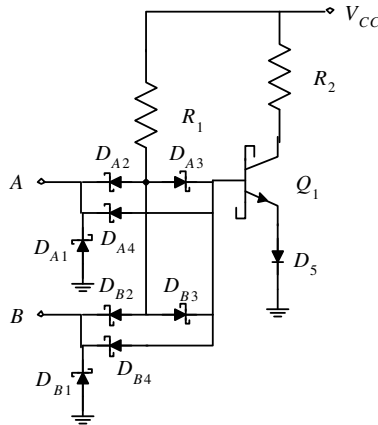


Figura 6.40. Configuración de entrada con diodo *cluster*

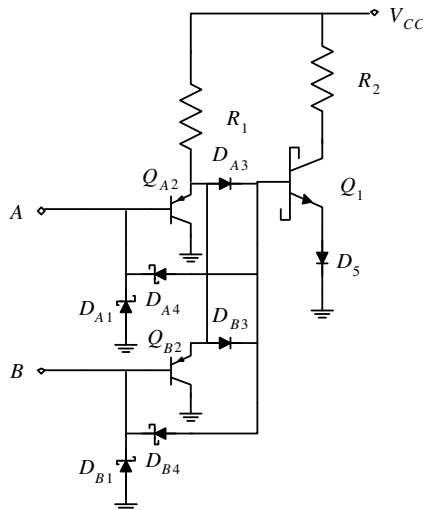


Figura 6.41. Configuración de entrada en transistor PNP

### 6.6.4 TTL Schottky de bajo consumo mejorada (ALSTTL, serie 54ALS/74ALS)

La subfamilia *Schottky de baja potencia avanzada*, ALSTTL (*Advanced Low power Schottky*) es una evolución de la LSTTL con un aumento de velocidad del doble y una reducción a la mitad en el consumo, lo que mejora el producto consumo velocidad (*power-speed*) a la cuarta parte y a la veintea parte con respecto a la TTL estándar, siendo útil en aquellas aplicaciones en las que la velocidad y el consumo son importantes.

Es compatible con TTL estándar, STTL y LSTTL, necesitando una corriente de entrada cuyo valor es la mitad respecto de la TTL estándar.

Como se muestra en la Figura 6.42, el circuito de la subfamilia ALSTTL es una evolución la LSTTL diferenciándose en:

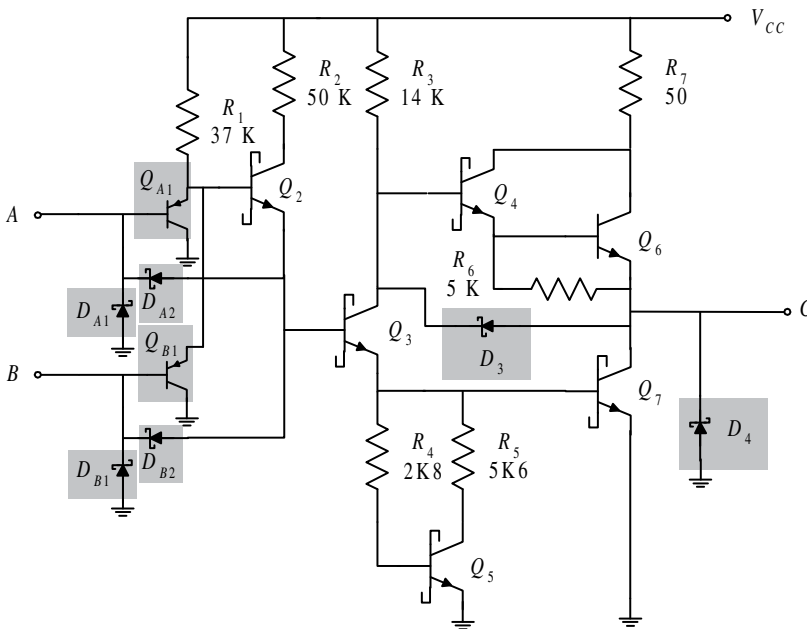


Figura 6.42. Puerta NAND ALSTTL

- La sustitución de los **diodos de entrada**  $D_3$  y  $D_4$  por los transistores  $Q_{A1}$  y  $Q_{B1}$  que reducen la corriente de entrada  $I_{IL}$  y que permiten evacuar más rápidamente la carga de la base de  $Q_2$ , aumentando así la velocidad de conmutación.

- Los **diodos CLAMPING**  $D_{A1}$  y  $D_{B1}$  limitan los impulsos negativos que se producen en la entrada al pasar del nivel alto a nivel bajo.
- Los **diodos**  $D_{A2}$  y  $D_{B2}$  favorecen la descarga de la capacidad de la base de  $Q_3$  alcanzando este transistor el corte más rápidamente, lo que aumenta la velocidad de conmutación.
- El **diodo**  $D_3$  descarga más rápidamente la capacidad presente en la salida, aumentando así la velocidad de conmutación. La corriente de descarga de  $D_3$  circula por  $Q_3$  aumentando el potencial de su emisor, y también la corriente de base de  $Q_7$ , lo que reduce el retardo de transición de nivel alto a nivel bajo de la salida (flanco de bajada).

#### **Características:**

Según se puede ver en la Tabla 6.4, las principales características de este tipo de puertas son las siguientes:

Consumo típico por puerta, 1 mW.

Tiempo de propagación, 4 ns.

Frecuencia operativa, 70 MHz.

### **6.6.5 TTL Schottky mejorada (ASTTL, serie 54AS/74AS)**

La subfamilia *Schottky* avanzada, ASTTL (*Advanced Schottky*) reduce a la mitad la potencia disipada y a la cuarta parte el producto potencia-velocidad (*power-speed*) respecto de la subfamilia STTL. También se caracteriza por tener corrientes de entrada más reducidas que las subfamilias anteriormente estudiadas y menores retardos de propagación, pudiendo trabajar a altas frecuencias con bajo consumo. Como desventaja cabe destacar el mayor coste.

En la Figura 6.43 se muestra el circuito interno de una puerta NAND ASTTL. Su **etapa de entrada** es parecida a la de la subfamilia ALSTTL, caracterizándose por:

- La sustitución de los diodos **CLAMPING**  $D_{A1}$  y  $D_{B1}$ , (Figura 6.42), por los **transistores Schottky**  $Q_{A1}$  y  $Q_{B1}$ . Estos transistores tienen unidos los terminales de  $B$  y  $E$  siendo su circuito equivalente el de su unión  $B-C$ , la cual se comporta como los diodos **CLAMP** protegiendo las entradas contra tensiones negativas y descargas electrostáticas.
- El **transistor**  $Q_3$  y el **diodo**  $D_3$  favorecen la transición rápida del nivel alto al nivel bajo (flanco de bajada) de la salida, evacuando la carga almacenada en la base de  $Q_{10}$  cuando éste conduce (nivel alto de la salida). Al mismo tiempo proporciona corriente de base a  $Q_4$  haciendo más rápida su puesta en conducción. También  $Q_3$  aumenta el rango de cero elevando el nivel de la tensión umbral de entrada  $V_{iLmax}$ , con lo que se obtiene una mayor inmunidad al ruido.

La **etapa de salida** de la subfamilia ASTTL se caracteriza por:

- La sustitución del diodo  $D_3$  de la subfamilia ALSTTL (Figura 6.42), por el **transistor  $Q_7$** , que realiza la misma función, favorecer el flanco de bajada descargando la capacidad de la carga conectada a la salida a nivel alto.

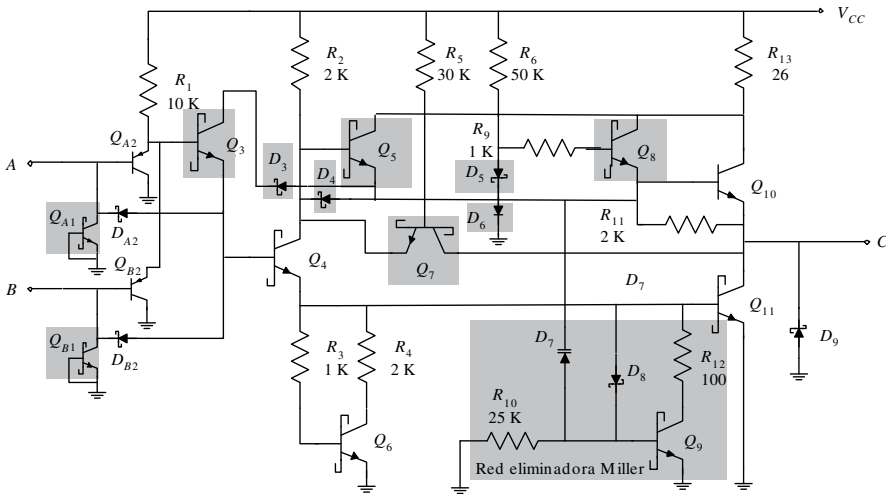


Figura 6.43. Puerta NAND ASTTL

- La incorporación del **transistor  $Q_8$**  para aumentar la corriente en la base de  $Q_{10}$  y a su vez los **diodos  $D_5$  y  $D_6$**  que favorecen la transición al corte de  $Q_8$ .
- La inclusión de una **red eliminadora Miller** (*The Miller Killer Network*) formada por  $Q_9$ ,  $R_{10}$ ,  $R_{12}$ ,  $D_7$  y  $D_8$ . Esta red reduce el tiempo de subida  $t_{DLH}$  de la tensión de salida, disminuyendo la potencia de conmutación a frecuencias altas. Cuando  $Q_4$  pasa al corte, el diodo varicap  $D_7$  comienza a cargarse gracias a la corriente suministrada por el emisor del transistor en conducción  $Q_5$ . La corriente de carga del diodo varicap se aplica a la base de  $Q_9$  haciéndole conducir. La conducción de  $Q_9$  facilita la transición al corte de  $Q_{11}$ , reforzando y realizando una de las funciones que tiene la red de encuadre formada por  $Q_6$ ,  $R_3$  y  $R_4$ . Una vez cargado el diodo varicap  $D_7$  se anula su intensidad, permaneciendo en esta situación hasta que  $Q_4$  entre en conducción lo que produce la descarga de  $D_7$  a través del camino  $Q_4$ ,  $D_4$  y  $D_8$ . Obsérvese que sin los componentes  $Q_9$  y  $D_7$ , la carga almacenada en  $Q_{11}$  haría que éste condujera y también durante un instante el transistor  $Q_{10}$ , con lo que se produciría una elevada corriente (cortocircuito) en la alimentación a través del camino  $R_{13}$ ,  $Q_{10}$  y  $Q_{11}$ .
- La conexión del **diodo Schottky  $D_9$**  protege la salida ante tensiones transitorias negativas.

## 6.6.6 TTL de alta velocidad (FTTL, serie 54F/74F)

La subfamilia FTTL (donde el significado de las siglas sería FAST- *FAIRCHILD Advanced Schottky* TTL) fue desarrollada por la compañía *FAIRCHILD* basándose en la técnica Isoplanar II, que posibilita llevar a cabo la fabricación de transistores de dimensiones extremadamente pequeñas. Esto permite conseguir capacidades parásitas muy pequeñas y por tanto mayores velocidades de conmutación.

Esta subfamilia se caracteriza por presentar una elevada impedancia de entrada, una alta velocidad de conmutación (aumento del 50%) y la reducción a la cuarta parte de la potencia disipada con respecto de la subfamilia STTL. Por consiguiente, esta subfamilia presenta un menor producto potencia-velocidad (*power-speed*).

Los circuitos con componentes lógicos STTL pueden ser sustituidos por sus equivalentes en FTTL.

En la Figura 6.44 se muestra el circuito interno de una puerta NAND FTTL. Como puede observarse en dicha figura, su circuito es bastante parecido al de otras subfamilias avanzadas de tecnología TTL, como ALSTTL y ASTTL que ya han sido descritas en apartados anteriores.

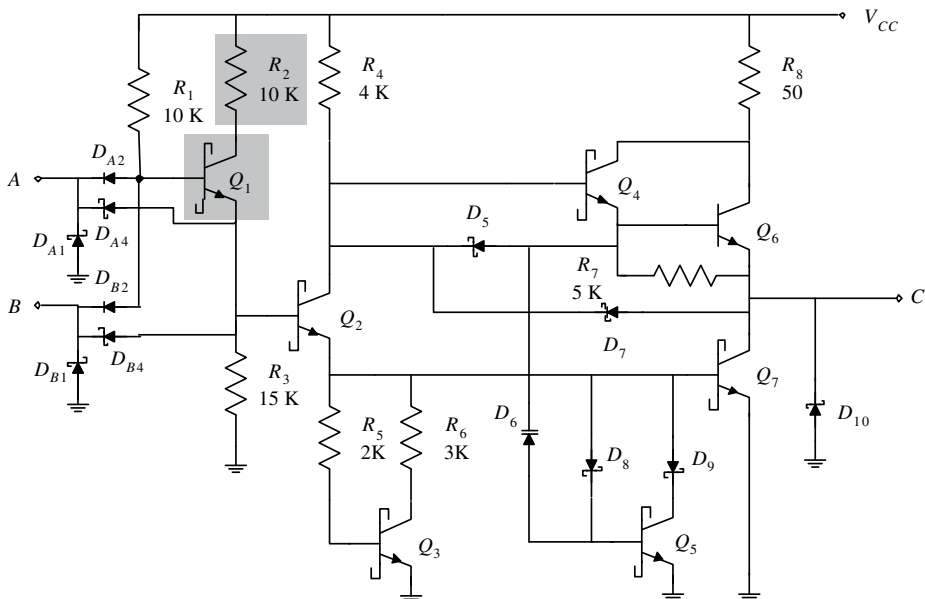


Figura 6.44. Puerta NAND FTTL

La **etapa de entrada** se caracteriza por:

- Tener una configuración de entrada en diodo *cluster* (Figura 6.40). Se ha sustituido el diodo  $D_{A3}$  por el transistor  $Q_1$  con su emisor conectado a tierra a través de  $R_3$ , cuya misión es establecer una corriente en la base de  $Q_2$  que favorezca la transición de este transistor al corte cuando la entrada de la puerta cambia a nivel bajo.
- Incluir la resistencia  $R_2$  y el transistor  $Q_1$  para establecer la corriente necesaria en las bases de  $Q_2$  y  $Q_7$  en los niveles altos de la tensión de entrada.

La **etapa de salida** de la subfamilia FTTL tiene elementos ya estudiados en la subfamilia ASTTL y que se indican a continuación a modo de resumen de los circuitos utilizados.

- Red de encuadre formada por  $Q_3$ ,  $R_5$  y  $R_6$ .
- Red eliminadora *Killer* formada por  $Q_5$ ,  $D_6$ ,  $D_8$ ,  $D_9$ .
- Par *Darlington* formado por los transistores  $Q_6$  y *Schottky*  $Q_4$ .
- Diodos  $D_{A4}$ ,  $D_{B4}$ ,  $D_5$  y  $D_7$  que aumentan la velocidad del circuito descargando las capacidades del mismo, y en concreto  $D_8$  que descarga la capacidad presente en la salida.
- Resistencia  $R_7$ , que para corrientes de salidas  $I_{oH}$  pequeñas, acerca el nivel  $V_{oH}$  a  $V_{CC}$ .
- El diodo  $D_{10}$  que protege la salida ante tensiones transitorias negativas.

## 6.7 COMPARACIÓN DE SUBFAMILIAS TTL

En la Tabla 6.4 se incluye una comparación entre distintas subfamilias TTL, según diversos parámetros de funcionamiento (tensiones, corrientes, tiempos, etc.).

Las subfamilias que se incluyen en dicha comparación son las siguientes:

- TTL-Estándar
- TTL-S
- TTL-LS
- TTL-ALS
- TTL\_AS
- TTL-F

Tabla 6.4. Comparación de subfamilias TTL

<b>Subfamilias TTL</b> (Parámetros referidos a puerta NAND)							
<b>Parámetro</b>	<b>Estándar</b>	<b>S</b>	<b>LS</b>	<b>ALS</b>	<b>AS</b>	<b>F</b>	<b>Unidad</b>
$V_{DD}$	4,75-5,25	4,75-5,25	4,75-5,25	4,5-5,5	4,5-5,5	4,5-5,5	V
$V_{oHmin}$	2,4	$V_{CC}-2$	2,7	$V_{CC}-2$	2,5	2,5	V
$V_{oLmax}$	0,4	0,5	0,5	0,5	0,5	0,5	V
$V_{iHmin}$	2	2	2,2	2,2	2,2	2,2	V
$V_{iLmax}$	0,8	0,8	0,9	0,9	0,8	0,9	V
$V_{NMH}$	400	700	700	700	700	700	mV
$V_{NML}$	400	300	300	300	300	300	mV
$I_{oHmax}$	-0,4	-1	-0,4	-0,4	-2	-1	mA
$I_{oLmax}$	16	20	8	8	20	20	mA
$I_{iHmax}$	40	50	20	20	20	20	$\mu$ A
$I_{iLmax}$	-1,6	-2	-0,36	-0,2	-0,5	-0,6	mA
$I_{oS}$	-100	-100	-100	-112	-112	-150	mA
$I_{CCH}$	4	16	1,6	0,85	3,2	2,8	mA
$I_{CCL}$	11	36	4,4	3	2,7	10,2	mA
$t_{pLH}$	11	3	8	4	2,5	3,7	ns
$t_{pHL}$	7	3	8	4	3,7	3,2	ns
$t_{TLH}$	12	6	13	10	5	2,5	ns
$t_{THL}$	5	3	3	6	3	2,5	ns
$f_{max}$	25	95	33	70	125	125	MHz
$Fan\ out$	10-TTL	10-S	20-LS	20-ALS	20-AS	20-F	u.c.
$P_D$	10	19	2	1	2	2	mW/ puerta
$Producto$							
$P_D \cdot t_{pD}$	100	60	20	4	13	12	pJ



## 6.8 PUERTAS TTL CON OTRO TIPO DE SALIDAS

Además de la configuración de salida en *Totem-Pole*, estudiada en los apartados anteriores, también se fabrican otros tipos de salidas que añaden nuevas funciones. Estas configuraciones son:

- Salida en **colector abierto** (OC - *Open Collector outputs*)
- Salida con control **tri-estado** (*Tri-State outputs control*)

### 6.8.1 Puertas TTL con salida en colector abierto

Las puertas con salida *Totem-Pole* no permiten la realización de un **cableado lógico**, es decir, unir dos o más salidas para obtener una nueva función. Si dos salidas en *Totem-Pole* se unen entre sí y en algún momento tienen nivel lógico distinto se produce una circulación de corriente excesiva, como la representada en la Figura 6.45, capaz de destruir las puertas.

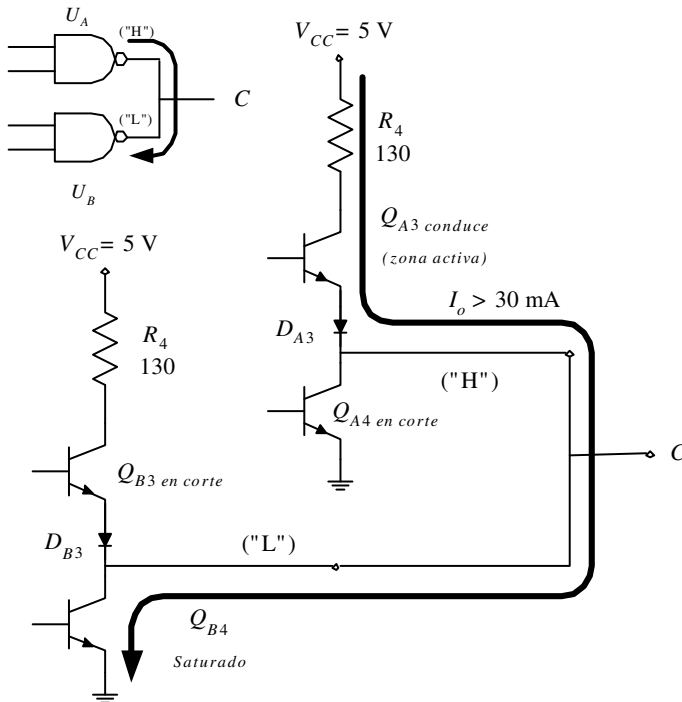


Figura 6.45. Imposibilidad de realizar cableado lógico con salidas Totem-Pole

Además, esta corriente elevada puede sacar al transistor  $Q_{B4}$  de la zona de saturación aumentando el potencial de salida por encima del rango de cero ( $V_{oL} > V_{oLmax}$ ). Considerando en la Figura 6.45 por ejemplo que la puerta  $U_A$  tiene salida a nivel alto ( $Q_{A3}$  en conducción en la zona activa y  $Q_{A4}$  en corte) y la puerta  $U_B$  con salida a nivel bajo ( $Q_{B3}$  en corte y  $Q_{B4}$  en saturación), esto ocasiona una corriente elevada en los transistores que conducen  $Q_{A3}$  y  $Q_{B4}$ , capaz de destruirlos por sobrepasar su disipación de potencia máxima.

Eliminando en la salida *Totem-Pole* la etapa *pull-up* (constituida por  $Q_3$  y  $D_3$ ) y conectado el colector de  $Q_4$  al terminal de salida, se construye la denominada **salida en colector abierto** mostrada en la Figura 6.46.

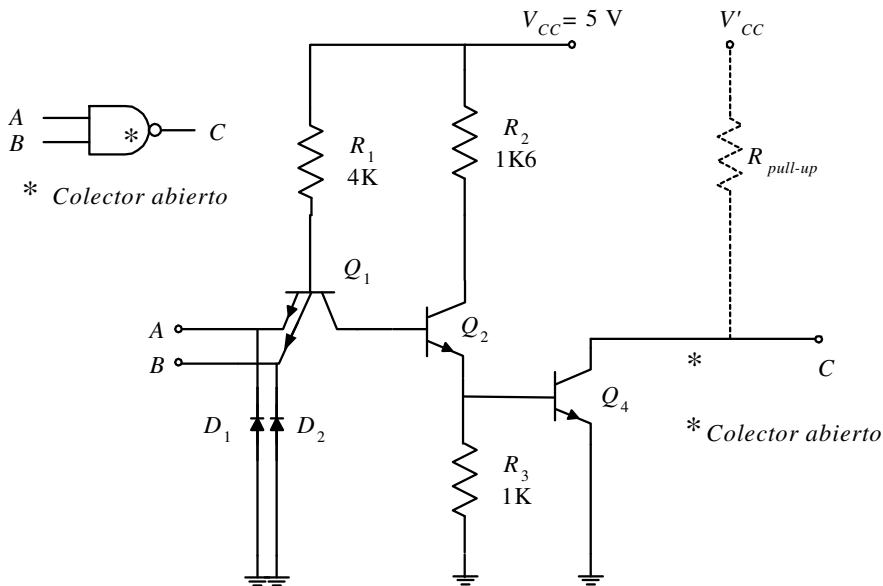


Figura 6.46. Puerta TTL estándar con salida en colector abierto

Como se muestra en el circuito de la Figura 6.47, se puede realizar un cableado lógico mediante puertas con salida en colector abierto. En dicho circuito se aprecia que la salida  $F$  toma el nivel lógico bajo si cualquiera de las salidas ( $F_A$ ,  $F_B$ , etc.) que forman el cableado lógico tiene nivel bajo.

En la Tabla 6.5 se representa el valor de  $F$  en función de los posibles niveles lógicos que tomen  $F_A$  y  $F_B$ , que como puede observarse, se corresponde con la tabla de verdad de la función AND. Por consiguiente se puede concluir que la nueva función que se obtiene al realizar un cableado lógico es la **función AND** de todas las salidas conectadas en colector abierto.

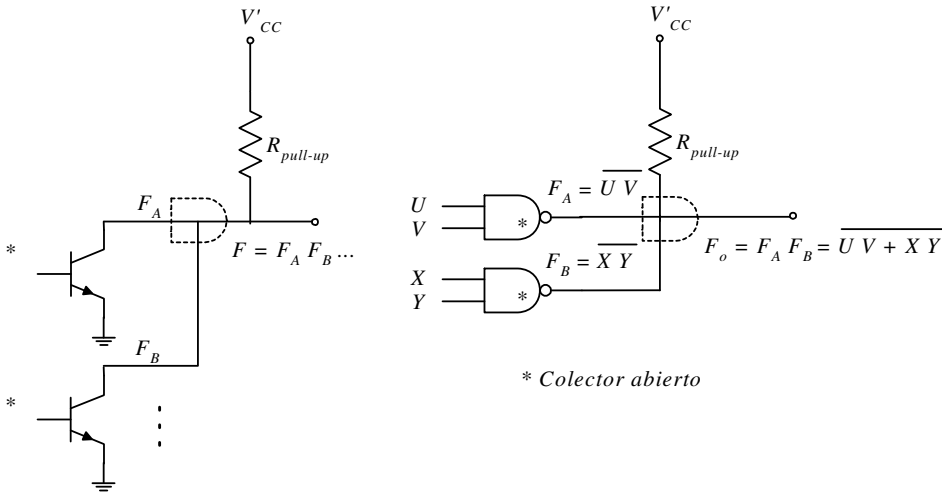


Figura 6.47. Cableado lógico realizado con puertas con salida en colector abierto de la familia TTL

Tabla 6.5. Tabla de verdad de la función  $F$  obtenida al realizar el cableado lógico de las salidas  $F_A$  y  $F_B$  en colector abierto

$F_A$	$F_B$	$F$
0	0	0
0	1	0
1	0	0
1	1	1

Para obtener, en la salida de una puerta en colector abierto, el potencial correspondiente al nivel alto y proporcionar un camino de carga de las capacidades de las puertas conectadas a la salida  $F$  del montaje que se representa en la Figura 6.48, es necesario conectar una resistencia entre dicha salida y un potencial positivo de alimentación, denominada **resistencia pull-up**  $R_{pull-up}$ .

El valor de  $R_{pull-up}$  es función del *fan-in* de las puertas a excitar (expresado en unidades de carga, *u.c.*) y del número de salidas de dispositivos con los que se ha realizado el cableado lógico. Este valor de la resistencia  $R_{pull-up}$  está acotado por un mínimo  $R_{pull-up\ min}$  que garantiza que no se exceda el *fan-out* cuando una sola salida de las que componen el cableado lógico está a nivel bajo y por un máximo  $R_{pull-up\ max}$  que garantiza el nivel alto  $V_{oH}$  de la salida  $F$  del cableado lógico. Dentro del margen

permitido a  $R_{pull-up}$  es importante tener en cuenta que los valores altos de  $R_{pull-up}$  favorecen un menor consumo con un mayor retardo y, al contrario, para valores bajos de  $R_{pull-up}$  el retardo será menor pero con mayor consumo.

En la expresión [6.35] se muestra el cálculo de la resistencia  $R_{pull-up}$ .

$$R_{pull-up \min} = \frac{V_{CCmax} - V_{oL}}{I_{oL} - N_{2(Low)} I_{iL}} \quad [6.35]$$

$$R_{pull-up \max} = \frac{V_{CCmin} - V_{oH}}{N_1 I_{oH} + N_{2(High)} I_{iH}}$$

Donde,  $N_1$  es el número de salidas pertenecientes al cableado lógico y  $N_2$  el número de cargas conectadas expresados en unidades de carga (*u.c.*), como se muestra en la Figura 6.48.

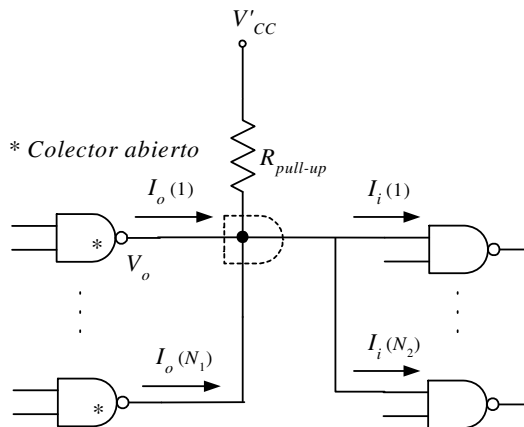


Figura 6.48. Resistencia  $R_{pull-up}$  en el cableado lógico

Otras ventajas que aportan las salidas en colector abierto son las siguientes:

- Permitir tensiones de salida, a nivel alto, superiores a la de alimentación de 5 V mediante la conexión de la resistencia  $R_{pull-up}$  a un potencial superior  $V'_{CC}$ , siempre que éste no exceda a la tensión de ruptura del transistor de salida en colector abierto. Esta posibilidad, como se verá posteriormente, se usa para conectar entre sí familias lógicas distintas.
- Menor consumo y disminución del tiempo de propagación total al disponer la puerta de menor número de transistores y también de menor número de niveles del circuito lógico.

**PROBLEMA RESUELTO 6-10**



Realizar, mediante el simulador *Electronics Workbench*, la función OR-exclusiva expresada en términos *maxterm* utilizando en los productos el cableado lógico para reducir el número de niveles del circuito lógico.

**Solución:**

La función XOR expresada en *maxterm* tiene como ecuación lógica:

$$S = (b + a)(\bar{b} + \bar{a})$$

Como se muestra en la Figura 6.49, el circuito se puede realizar mediante dos puertas OR en colector abierto cuyas salidas se unen para obtener la función AND mediante cableado lógico. Se puede comprobar que la salida del circuito corresponde a una función XOR utilizando el convertidor lógico de EWB.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap06\Ewb5\06W0\_\_06.ewb**

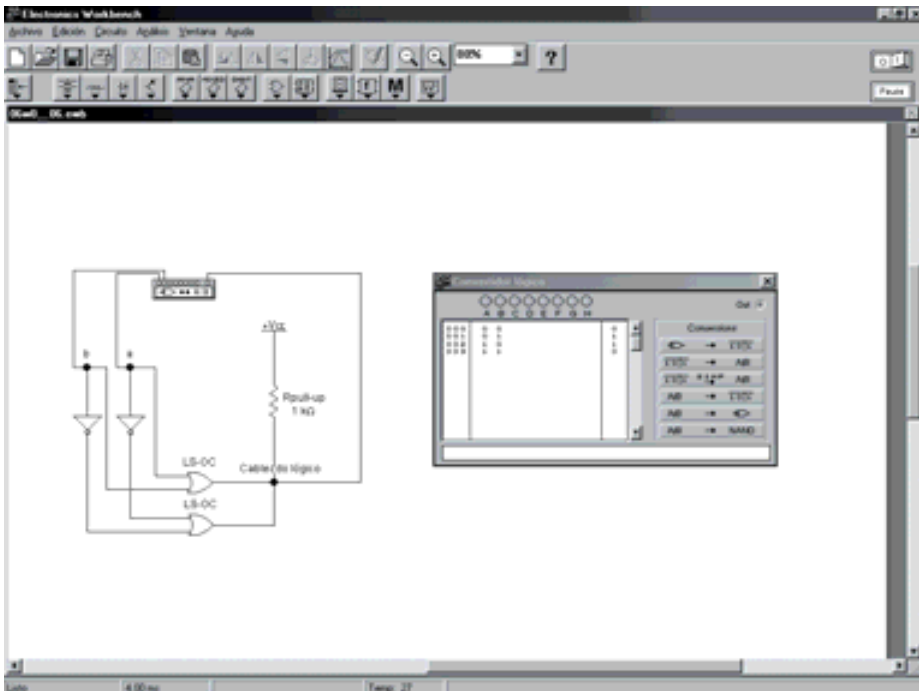


Figura 6.49. Simulación de la puerta XOR utilizando cableado lógico

## 6.8.2 Puertas TTL con salida con control triestado

La salida triestado permite, mediante un circuito de control, el corte simultáneo de los dos transistores de salida  $Q_3$  (transistor *pull-up*) y  $Q_4$  (transistor *pull-down*) de la configuración *Totem-Pole*. A este nuevo estado o tercer estado se le conoce como **estado de alta impedancia**. Las salidas triestado posibilitan la conexión de varias salidas de puertas a una línea común o a un *bus* de datos.

En la Figura 6.50 se representa el circuito de una puerta TTL estándar con salida triestado. Como puede observarse este circuito es el mismo que el de una puerta TTL estándar al que se le ha incorporado una entrada de inhabilitación  $\bar{I}$  (*disable*), un diodo  $D_4$  y un emisor más en el transistor multiemisor  $Q_1$ .

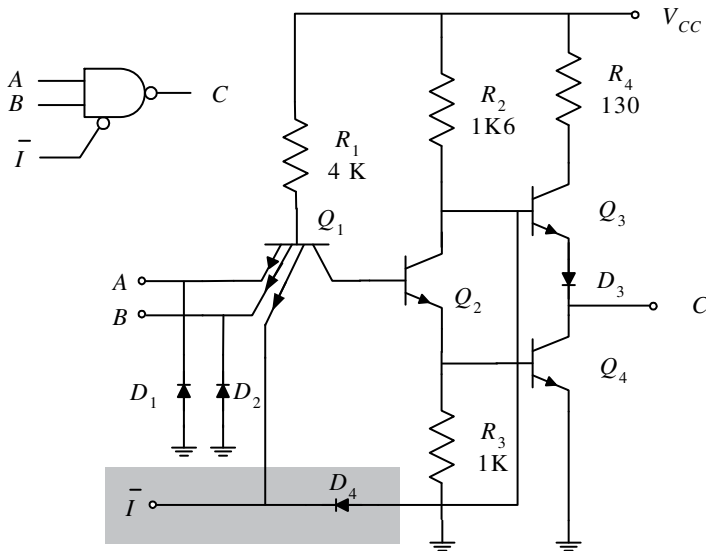


Figura 6.50. Puerta TTL estándar con salida triestado

En la Figura 6.51 se muestra el comportamiento de una puerta con salida triestado cuando la **señal de inhabilitación  $\bar{I}$  es inactiva** (nivel alto).

Este nivel alto de la señal  $\bar{I}$  no influye en la conducción de  $Q_1$ , dependiendo este transistor del estado de los otros emisores o entradas de la puerta (al comportarse el colector de  $Q_1$  como una función AND, respecto del estado de sus emisores).

El diodo  $D_4$  tampoco conduce, por lo que en estas condiciones el circuito se comporta como el de una puerta TTL estándar con salida *Totem-Pole*, entregando niveles lógicos altos o bajos, siendo su salida la función NAND de sus entradas.

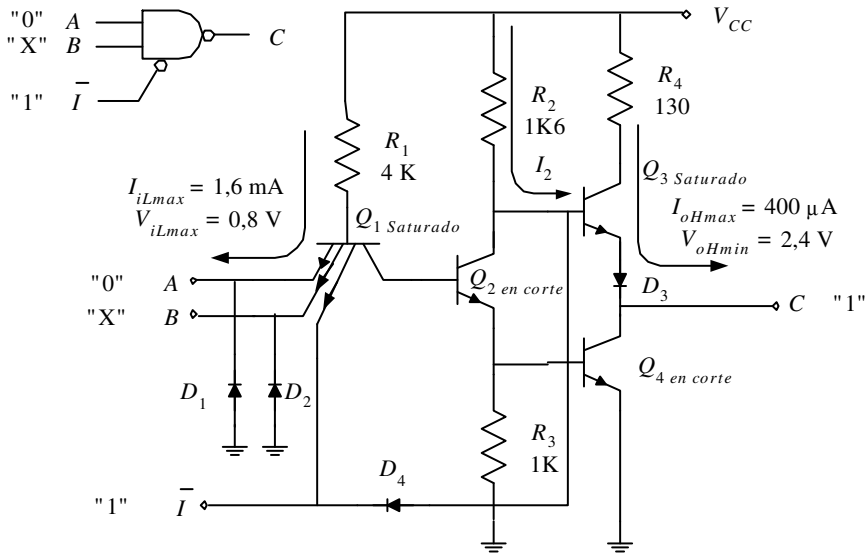


Figura 6.51. Comportamiento de una puerta con salida triestado cuando la señal de inhabilitación  $\bar{I}$  es inactiva (nivel alto)

En la Figura 6.52 se muestra el comportamiento de una puerta con salida triestado cuando la **señal de inhabilitación  $\bar{I}$  es activa** (nivel bajo).

Este nivel bajo de la señal  $\bar{I}$  aplica un potencial cercano a cero al emisor del transistor  $Q_1$ , por lo que este transistor se satura, dejando en corte a  $Q_2$  y al transistor *pull-down*  $Q_4$ .

Por otra parte el nivel bajo de  $\bar{I}$  pone en conducción a  $D_4$ , dejando sin corriente a la base de  $Q_3$ , por lo que este transistor *pull-up* queda en corte. Como ambos transistores *pull-up* y *pull-down* están en corte el terminal de salida queda aislado.

La impedancia vista desde el terminal de salida de la puerta es muy alta (denominándose a esta situación, estado de alta impedancia).

Otros nombres utilizados para las señales de control de la salida triestado son el permiso o habilitación (*Strobe*), que es complementaria a la de inhabilitación (*Disable*) que se ha definido anteriormente.

En la Tabla 6.6 se resume el comportamiento de la puerta triestado estudiada en este apartado. En dicha tabla se muestran la señal de inhabilitación, las dos entradas de la puerta y el valor de la salida para cada caso.

Tabla 6.6. Tabla de verdad de la puerta NAND triestado

$\bar{I}$	B	A	C
0	X	X	Z
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

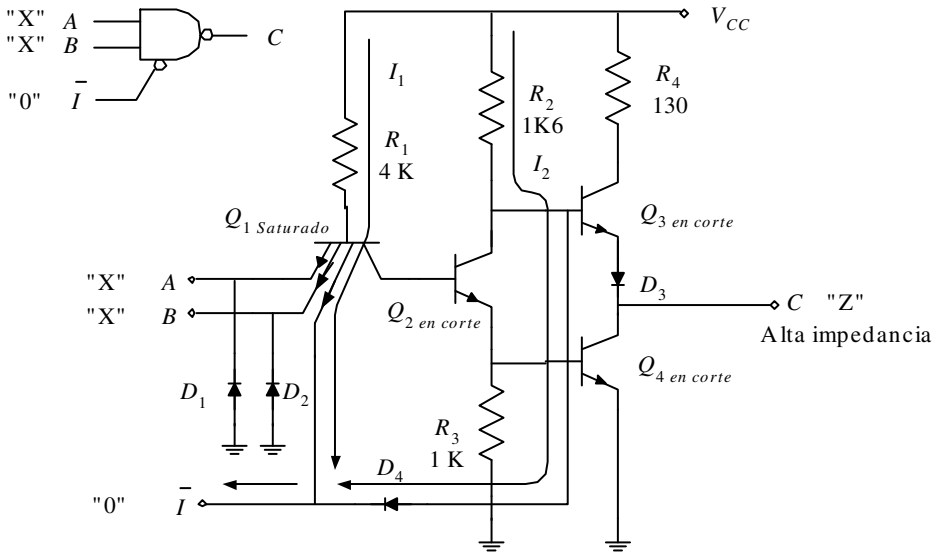


Figura 6.52. Comportamiento de una puerta con salida triestado cuando la señal de inhabilitación  $\bar{I}$  es activa (nivel bajo)

**PROBLEMA RESUELTO 6-11**



Realizar un circuito que permita aplicar sobre una misma línea de un bus dos señales de reloj procedente de las salidas de dos puertas, pudiendo elegir la señal de reloj presente en el bus mediante una señal de selección, utilizando para ello el programa de simulación *Electronics Workbench*.



### Solución:

Para poder conectar las salidas de dos puertas a una misma línea de *bus*, dichas puertas deberán tener salidas triestado. Cada salida de estas puertas triestado aplicarán una señal de reloj al bus y sólo una de ellas estará habilitada mediante una señal de selección.

En la Figura 6.53 se muestra el circuito que se utilizará para resolver este problema. La selección del reloj presente en el bus se realiza cambiando el estado del conmutador mediante la barra espaciadora. Las señales de habilitación de las puertas triestado son complementarias, por lo que siempre sólo una de ellas está habilitada en cada momento evitándose así la contención del bus.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap06\Ewb5\06W0\_\_07.ewb**

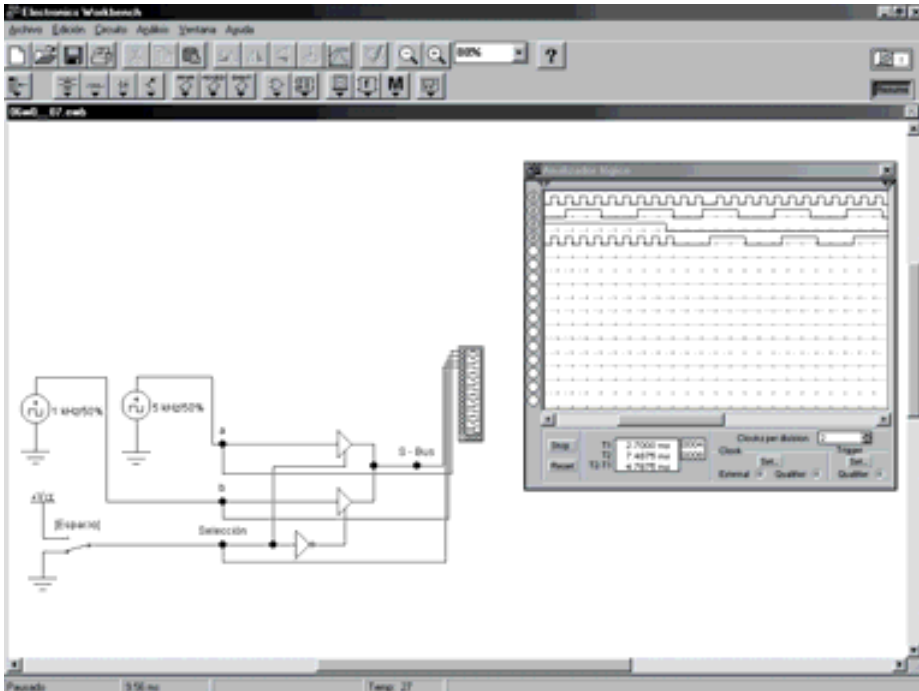


Figura 6.53. Simulación con puertas triestado

## 6.9 PRECAUCIONES EN EL DISEÑO DE SISTEMAS CON TECNOLOGÍA TTL

El diseño de sistemas digitales con tecnología TTL requiere una serie de precauciones en su alimentación, conexión, entradas no utilizadas, desacoplo, etc. que deben tenerse en cuenta. Estas precauciones son las siguientes:

- **Alimentación.** La alimentación debe ser regulada para que esté comprendida entre 4,75 V y 5,25 V. Las líneas de alimentación serán de baja inductancia e impedancia; y la masa, preferentemente, se debe realizar mediante plano de masa, evitando lazos cerrados de corriente.
- **Desacoplo.** En sistemas combinacionales se debe colocar un condensador de desacoplo comprendido entre 10 y 100 nF por cada cinco encapsulados. En sistemas secuenciales se debe colocar un condensador de desacoplo en cada encapsulado y lo más cercano posible a sus pines de alimentación.
- **Hilos o pistas de conexión.** Estos hilos o pistas deben tener una longitud inferior a 25 cm. Entre 25 cm y 50 cm es necesario usar un plano de masa. Por encima de 50 cm se debe emplear cable coaxial o de par trenzado, colocando una resistencia *pull-up*, al final de la línea, para aumentar el margen de ruido. Además, si las señales son de alta frecuencia se deben evitar líneas paralelas en las pistas de circuito impreso.
- **Entradas no utilizadas.** Nunca se deben dejar al aire o sin conectar entradas de dispositivos lógicos ya que son fuentes de ruido al actuar como antenas.

En las puertas de funciones lógicas básicas (OR, NOR, NAND) una solución sencilla consiste en unir las entradas no utilizadas a otras que sí están conectadas en el circuito, pero teniendo en cuenta que al añadir estas nuevas entradas no se exceda el *fan-out*. Otra solución consiste en poner las entradas no utilizadas de las puertas OR y NOR a nivel bajo, para que la salida no se modifique. De la misma forma para el caso de entradas no utilizadas en las puertas AND y NAND se pondrán a nivel alto.

- **Entradas con nivel fijo.** Para forzar una entrada a nivel bajo sólo se debe conectar a masa, sin embargo para forzarla a nivel alto no es aconsejable conectarla directamente a  $V_{CC}$  ya que si esta alimentación supera el valor de 5,5 V puede provocar la destrucción de la puerta por ruptura de la entrada. Lo más aconsejable es colocar una resistencia limitadora de 1 k $\Omega$  entre la entrada a forzar y  $V_{CC}$ ; en este caso, además, con una única resistencia limitadora se pueden conectar hasta cincuenta entradas a nivel alto.

## 6.10 FAMILIA LÓGICA CMOS

### 6.10.1 Introducción

La familia CMOS, constituida por transistores unipolares MOS de canal N y canal P, se caracteriza, principalmente, frente a la familia TTL, por tener un consumo más bajo, y como contrapartida es más lenta.

### 6.10.2 Constitución del circuito básico CMOS

En la Figura 6.54 se representa el circuito básico de un inversor CMOS. Como puede observarse está constituido por dos transistores MOS, uno de canal N y el otro de canal P, alimentados por los potenciales  $V_{DD}$  y  $GND$ .

Se debe destacar la simplicidad del circuito, con sólo dos transistores, la carencia de resistencias y otros componentes respecto a la familia TTL. Esto último, unido a su bajo consumo, favorece un elevado nivel de integración y un coste más reducido de fabricación, haciendo que la familia CMOS sea ampliamente utilizada.

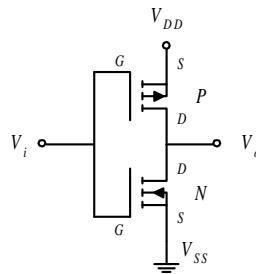


Figura 6.54. Circuito básico de un inversor CMOS

En la Figura 6.55 se muestra el funcionamiento del inversor CMOS con entrada a nivel alto.

Cuando la entrada del inversor  $V_{iH} \approx V_{DD}$ , se cumple que el potencial  $V_{GS1} \approx 0$  V, deduciéndose de la curva de transconductancia de  $Q_1$  que este transistor no conduce (estado de corte) presentando un camino de alta impedancia entre la salida y  $V_{DD}$ . Asimismo, el nivel alto en la entrada hace que  $V_{GS2} > V_{T2}$ , deduciéndose de la curva de transconductancia de  $Q_2$  que este transistor conduce, presentando un camino de muy baja impedancia entre la salida y  $GND$ , por lo que la salida del circuito toma un nivel lógico bajo  $V_{oL} \approx 0$  V.

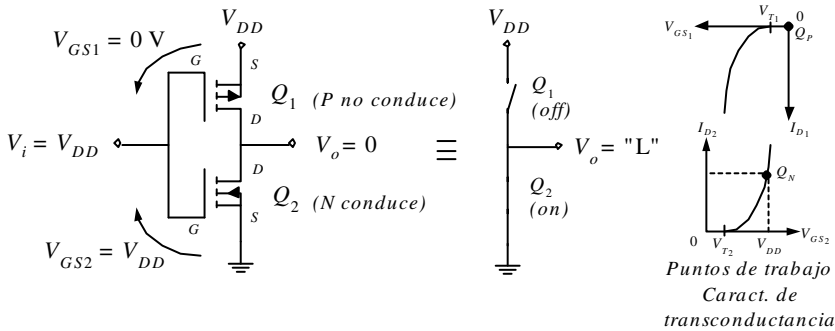


Figura 6.55. Funcionamiento de un inversor CMOS con entrada a nivel alto

En la Figura 6.56 se muestra el funcionamiento del inversor CMOS con entrada a nivel bajo.

Cuando en la entrada del inversor,  $V_{iL} \approx 0\text{ V}$ , se cumple que el potencial  $V_{GS2} \approx 0\text{ V}$  deduciéndose de la curva de transconductancia de  $Q_2$  que este transistor no conduce (estado de corte) presentando un camino de alta impedancia entre la salida y  $GND$ .

Asimismo, el nivel bajo en la entrada hace que  $V_{GS1} < V_{T1}$ , deduciéndose de la curva de transconductancia de  $Q_1$  que este transistor conduce, presentando un camino de muy baja impedancia entre la salida y  $V_{DD}$ , por lo que la salida del circuito toma un nivel lógico alto  $V_{oH} \approx V_{DD}$ .

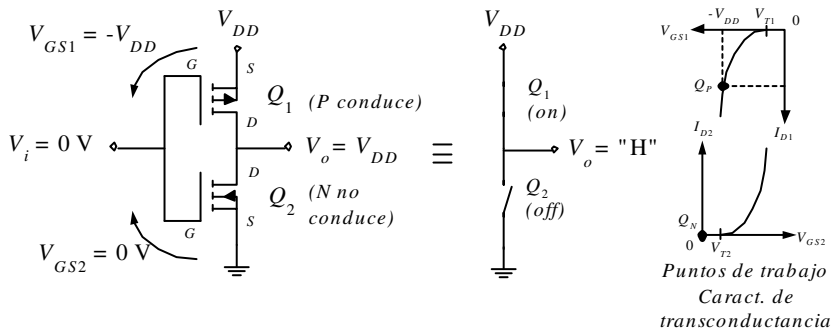


Figura 6.56. Funcionamiento de un inversor CMOS con entrada a nivel bajo

Este mismo análisis se puede realizar en los circuitos que se muestran en la Figura 6.57, dejando al lector la comprobación de que los circuitos corresponden a puertas a) NOR y b) NAND, respectivamente.

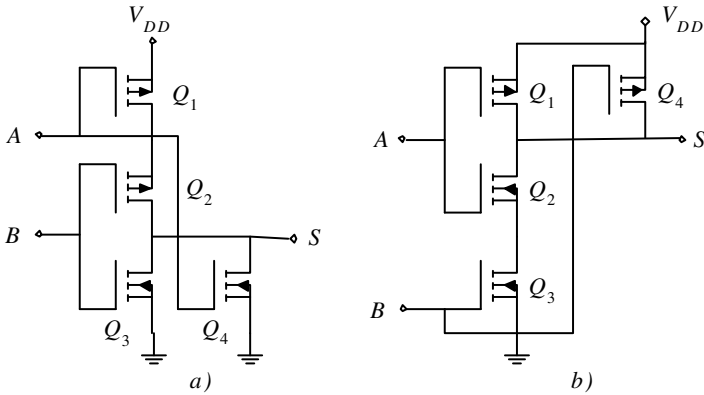


Figura 6.57. Ejemplos de circuitos de puertas CMOS: a) NOR y b) NAND

### 6.10.3 Características estáticas de la familia CMOS

#### 6.10.3.1 CARACTERÍSTICA DE TRANSFERENCIA

Como se puede apreciar en la Figura 6.58, la función de transferencia del inversor CMOS se aproxima a la de una puerta ideal ya que presenta una transición entre niveles con una pendiente elevada, con una tensión de entrada igual a la mitad de la alimentación  $V_{DD}/2$  y con envolventes (curvas extremas de transferencia) muy cercanas.

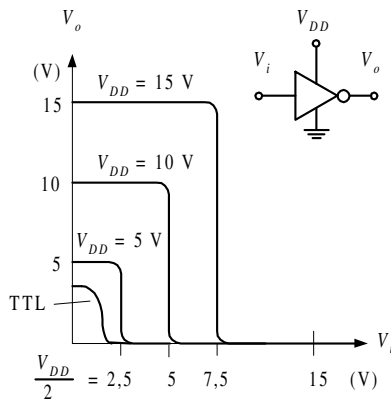


Figura 6.58. Característica de transferencia del inversor CMOS

Se debe destacar, en la función de transferencia de las puertas CMOS, que los niveles lógicos de salida son cercanos a los potenciales de alimentación  $V_{OL} \approx 0 \text{ V}$  y  $V_{OH} \approx V_{DD}$  y que se pueden aplicar un amplio margen de potenciales  $V_{DD}$  de alimentación. En la Figura 6.58 también se incluye como referencia la característica de transferencia del inversor TTL, con el fin de poder compararla con la de CMOS.

## PROBLEMA RESUELTO 6-12



Obtener la función de transferencia de una puerta inversora CMOS, utilizando para ello el programa de simulación *Electronics Workbench*.

### Solución:

Para simular la función de transferencia de esta puerta, se parte del esquema representado en la Figura 6.59. Para ello se introduce en el simulador *Electronics Workbench* el circuito mostrado anteriormente en la Figura 6.54, correspondiente a la estructura de la puerta inversora CMOS, y se realiza con este diseño un subcircuito denominado *cmos*.

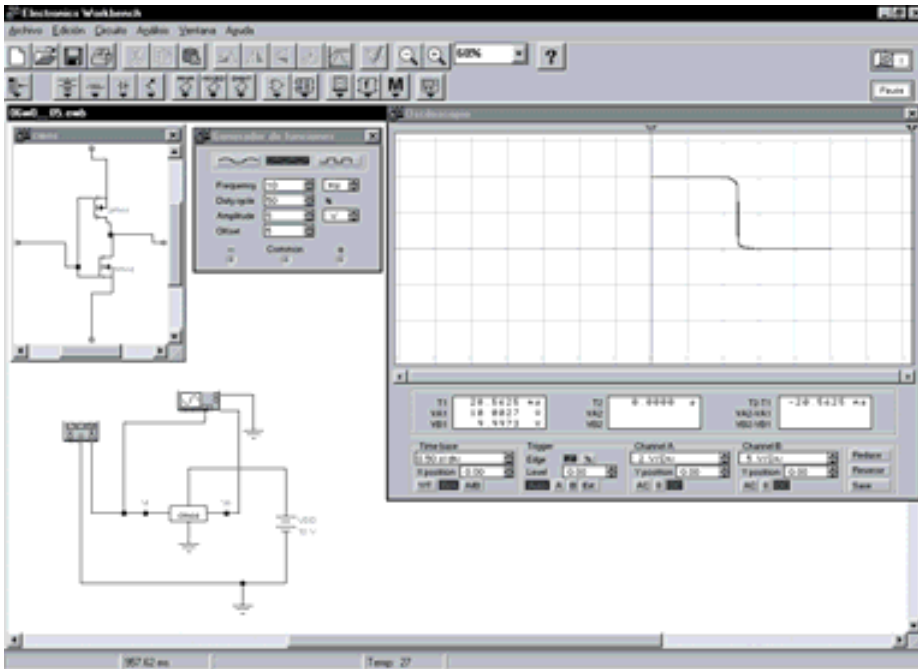


Figura 6.59. Función de transferencia de una puerta inversora CMOS

Utilizando el osciloscopio que tiene disponible la aplicación, se puede comprobar la representación de la curva característica de transferencia de la puerta inversora CMOS, que se muestra en la Figura 6.59. Para ver las distintas curvas de la función de transferencia según la alimentación  $V_{DD}$  aplicada a la puerta, como se ha mostrado anteriormente en la Figura 6.58, se debe variar en esta simulación el valor de  $V_{DD}$  a 5 V, 10 V y 15 V.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap06\Ewb5\06W0\_05.ewb**

### 6.10.3.2 CARACTERÍSTICA DE ENTRADA Y SALIDA

La familia CMOS se caracteriza por tener un valor reducido de la corriente de entrada  $I_i$  siendo inferior a  $\pm 1$  mA y corrientes de salida  $I_o$  entre  $\pm 10$  mA y  $\pm 50$  mA, dependientes en función directa de la tensión de alimentación  $V_{DD}$ . Además, los niveles lógicos dependen de la alimentación, así  $V_{iHmin}$  suele ser el 70 % de  $V_{DD}$  y  $V_{iLmax}$  el 30 % de  $V_{DD}$  como se muestra en la Figura 6.60.

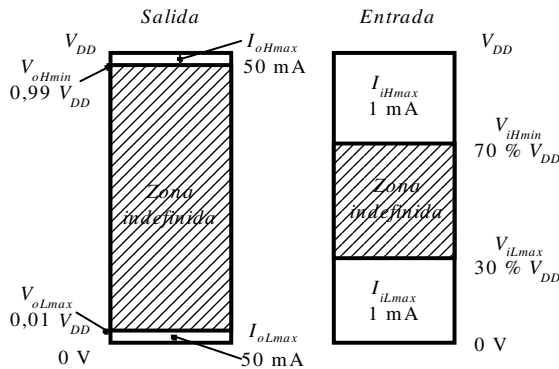


Figura 6.60. Perfiles lógicos de entrada y salida de un circuito lógico CMOS

De los perfiles lógicos mostrados en la Figura 6.60 se deduce que la familia CMOS tiene mayor *fan-out* que la TTL, siendo típico un valor de 50. Sin embargo, cuanto mayor sea el número de entradas que se conecten a una salida determinada mayor es la capacidad de carga presente en esta salida (aproximadamente 5 pF por cada entrada conectada), aumentando considerablemente los tiempos de conmutación. En este sentido el procedimiento que siguen los diseñadores es considerar el retardo de propagación máximo que se quiere en la puerta y en función de éste, obtener la capacidad de carga máxima permitida en la salida (normalmente mediante gráficas). Como se muestra en la expresión [6.36], si se divide la capacidad de carga máxima

permitida en la salida  $C_{o\ max}$  entre la capacidad que presenta cada entrada (*u.c.* unidad de carga) conectada a ella  $C_{i\ (u.c.)} \approx (5\ \text{pF})$ , se obtiene el *fan-out* de la puerta que garantiza o limita el retardo de propagación máximo permitido.

$$\text{fan-out} = \frac{C_{o\ max}}{C_{i\ (u.c.)}} \quad [6.36]$$

### Ejemplo:

En un circuito CMOS, cuyo retardo de propagación en función de la capacidad de carga es el representado en la curva característica que se muestra en la Figura 6.61, se calculará el *fan-out* para que el retardo de propagación no sea superior a 15 ns.

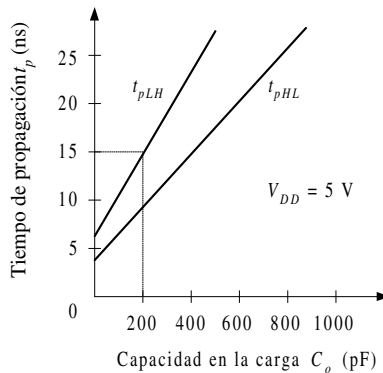


Figura 6.61. Retardo de propagación en función de la capacidad de carga en un circuito CMOS

De la curva característica del circuito se obtiene que la capacidad de carga máxima en la salida que garantiza un retardo de propagación inferior a 15 ns, para el caso más desfavorable (curva superior correspondiente al  $t_{pLH}$ ), es de 200 pF. Siendo el *fan-out*:

$$\text{fan-out} = \frac{C_{o\ max}}{C_{i\ (u.c.)}} = \frac{200\ \text{pF}}{5\ \text{pF}} = 40\ \text{u.c.}$$

Al igual que en TTL, se deben evitar las **entradas flotantes**, pues a la vez que son fuentes de ruido, en el caso de la tecnología CMOS, producen un consumo excesivo que puede llegar a ser destructivo, al trabajar ambos transistores MOS de canal  $p$  y  $n$



en la zona lineal (en conducción), presentado una baja impedancia entre los terminales de alimentación.

La constitución de la puerta aislada del transistor MOS mediante una finísima película (aproximadamente  $1.000 \text{ \AA}$ ), muy aislante, de dióxido de silicio,  $\text{SiO}_2$ , con rigidez dieléctrica aproximada de  $7 \cdot 10^6 \text{ V/cm}$ , puede ser perforada por **descargas electrostáticas**, presentes al ser manipulada en condiciones normales (como por ejemplo  $70 \text{ V}$ ). Para evitar este inconveniente, propio de la tecnología CMOS, se integran en el circuito **diodos de protección**, tal como se muestra en la Figura 6.62, que, cuando conducen, limitan el potencial en el terminal de puerta de los transistores MOS a potenciales seguros, comprendidos entre  $V_{DD}$  y  $V_{SS}$  de alimentación (entre  $0$  y  $18 \text{ V}$ ).

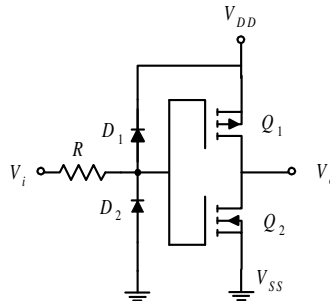


Figura 6.62. Diodos de protección ante descargas electrostáticas en la entrada de los transistores MOS

### 6.10.3.3 INMUNIDAD FRENTE AL RUIDO

Se deduce de los perfiles de entrada y salida de un circuito lógico CMOS, representados anteriormente en la Figura 6.60, que la inmunidad al ruido es aproximadamente el 30% de la tensión de alimentación  $V_{DD}$ .

### 6.10.3.4 CONSUMO Y DISIPACIÓN DE POTENCIA

Uno de los motivos por el que en los diseños se elige la familia CMOS, frente a otras soluciones, es por su bajo consumo, siendo este requisito imprescindible, por ejemplo, en equipos portátiles.

En la tecnología de circuitos integrados digitales CMOS el **consumo dinámico de potencia**  $P_T$  que se produce en las transiciones entre niveles lógicos (conmutación), puede ser incluso más elevado que los **consumos estáticos**  $P_D$ . Si en una puerta lógica

se producen transiciones de nivel en su salida con una determinada frecuencia  $f$  el consumo dinámico de potencia será proporcional a dicha frecuencia. Por tanto, para el cálculo del consumo o disipación de potencia se debe considerar el consumo estático en aquellos circuitos digitales con transiciones entre niveles lógicos poco frecuentes y consumo dinámico en aquellos circuitos con continuas transiciones entre niveles lógicos.

- **Consumo estático**

Como ya se indicó en el apartado 6.2.1.4 el consumo medio estático o **potencia media estática**  $P_D$ , viene dada, para un ciclo de trabajo (*duty cycle*) del 50%, por la expresión [6.37].

$$P_D = I_{CC} V_{CC} \quad [6.37]$$

Siendo la corriente media estática  $I_{CC}$ , que viene dada por la expresión [6.38].

$$I_{CC} = \frac{I_{CCH} + I_{CCL}}{2} \quad [6.38]$$

El valor de la potencia media estática es muy bajo, del orden de nW.

- **Consumo dinámico**

El consumo dinámico se produce en las transiciones principalmente por **dos causas**:

- ◆ La capacidad que existe en paralelo con la salida necesita ser cargada y descargada para variar los niveles lógicos de la salida (considérese que cada entrada conectada a la salida de la puerta incrementa su capacidad entre 3 y 5 pF). Como la tensión en una capacidad no puede variar en un tiempo nulo se producen picos de corriente en la alimentación  $I_{DD}$  que incrementan el consumo, siendo éste mayor al aumentar la frecuencia.
- ◆ Durante las transiciones en las que uno de los transistores MOS de salida debe pasar del estado de conducción al de corte y el otro transistor de salida del corte a la conducción, se produce un breve instante en el que ambos transistores conducen, produciéndose picos de corriente en la alimentación  $I_{DD}$  que incrementan el consumo, siendo éste mayor al aumentar la frecuencia.

De lo anteriormente indicado se deduce que la **potencia dinámica**  $P_T$  depende de la capacidad presente en la salida, de la frecuencia de conmutación y de la alimentación aplicada a la puerta, como se indica en la expresión [6.39].

$$P_T = f C V_{DD}^2 \quad [6.39]$$

La potencia dinámica  $P_T$  puede llegar a ser excesiva cuando se trabaja con frecuencias elevadas.

## 6.10.4 Características dinámicas de la familia CMOS

### 6.10.4.1 RETARDOS DE PROPAGACIÓN

Si bien la ventaja que caracteriza a la familia CMOS con respecto a la TTL es su bajo consumo a bajas frecuencias, su desventaja más notable es la de presentar retardos elevados en la conmutación (superiores a 100 ns). Por ello, el desarrollo de las subfamilias CMOS ha estado orientado en reducir los retardos de propagación hasta acercarse a los de las subfamilias TTL, alcanzando valores cercanos a los 3 ns.

Los retardos de propagación en la familia CMOS dependen de la alimentación  $V_{DD}$  aplicada a la puerta y de la capacidad de carga conectada a la salida  $C_o$  (como ya se ha indicado anteriormente en el ejemplo del apartado 6.10.3.2 al calcular el *fan-out*).

### 6.10.4.2 FRECUENCIA MÁXIMA DE FUNCIONAMIENTO

La **frecuencia máxima** de una puerta CMOS es la mayor frecuencia que asegura que en la conmutación de la puerta se alcanzan los niveles lógicos. Su cálculo se realiza como en cualquier puerta lógica aplicando la expresión [6.17], anteriormente estudiada.

$$f_{max} = \frac{1}{4 t_{pD}}$$

### 6.10.4.3 PRODUCTO CONSUMO POR TIEMPO DE PROPAGACIÓN

El consumo de potencia y el tiempo de propagación es un factor de mérito de gran utilidad para comparar, entre sí, familias y subfamilias lógicas. Dicho producto, que ya fue estudiado e indicado en la expresión [6.18], representa la energía consumida por una determinada puerta lógica expresada en picojulios (pJ). Se busca siempre que este parámetro sea lo menor posible.

$$potencia \times retardo = P_D T_{pD}$$

## 6.10.5 Otras características de la familia CMOS

### 6.10.5.1 FLEXIBILIDAD LÓGICA

La flexibilidad lógica es una medida de la versatilidad, capacidad o posibilidad de implementación de sistemas digitales con una determinada tecnología. Al igual que la familia TTL, la familia CMOS dispone de factores que caracterizan la flexibilidad lógica, como son:

- posibilidad de realizar cableado lógico, mediante puertas con drenador abierto,
- capacidad de excitación mediante *buffers*,
- variedad en las salidas mediante puertas con dos salidas, una directa y otra inversa,
- variedad de bloques funcionales o disponibilidad de circuitos integrados con funciones específicas (codificadores, contadores, sumadores, etc.), y
- compatibilidad con otras tecnologías mediante conexión directa, con adaptadores muy simples o con circuitos diseñados para este fin.

### 6.10.5.2 COSTE

El coste de los componentes de un sistema digital es un factor determinante en su diseño y realización, siendo las puertas CMOS, con menor número de componentes, menor tamaño y más fáciles de fabricar que las puertas TTL, las que presentan una mayor integración a menor coste.

## 6.11 SUBFAMILIAS CMOS

La evolución de las subfamilias CMOS ha estado orientada a mejorar su mayor desventaja, su lentitud de respuesta respecto de otras familias como la TTL, es decir, reducir los retardos de propagación.

### 6.11.1 CMOS estándar (serie 4000, 4000A, 4000B y 4000UB)

La familia CMOS 4000 es la estándar de esta tecnología, de la cual han evolucionado otras subfamilias con mejores prestaciones.

Las principales características, estáticas y dinámicas de la familia CMOS, ya han sido apuntadas en los apartados anteriores. De entre todas ellas, cabe destacar que tienen un amplio margen de tensión de alimentación, alta inmunidad al ruido, bajo

consumo, amplio margen de temperatura, un *fan-out* casi ideal al presentar una muy alta impedancia de entrada. Su mayor inconveniente es la menor velocidad respecto a la familia TTL.

Como referencia, se indican a continuación las principales características de la familia CMOS 4000 considerando sus tres series disponibles: 4000A, 4000B y 4000UB.

- **4000A:**

Alimentación comprendida entre 3 y 12 V. Corriente de salida  $I_{oL} = 0,5 \text{ mA}$  e  $I_{oH} = -0,5 \text{ mA}$ . Tiempo de propagación  $t_p = 125 \text{ ns}$ .

- **4000B:**

Alimentación comprendida entre 3 y 18 V. La sigla B (4000 *Buffered*) representa su principal modificación al incorporar un *buffer* en la salida que permite corrientes  $I_{oL} = 1 \text{ mA}$  e  $I_{oH} = -1 \text{ mA}$ . En la parte izquierda de la Figura 6.63 se muestra un esquema interno de este tipo de puertas.

- **4000UB:**

El significado de las siglas UB (4000 *UnBuffered*) indica que se ha suprimido el *buffer* en la salida reduciéndose el retardo de propagación  $t_p$  a 90 ns, aunque también se reduce la inmunidad al ruido. En la parte derecha de la Figura 6.63 se muestra un esquema interno de este tipo de puertas.

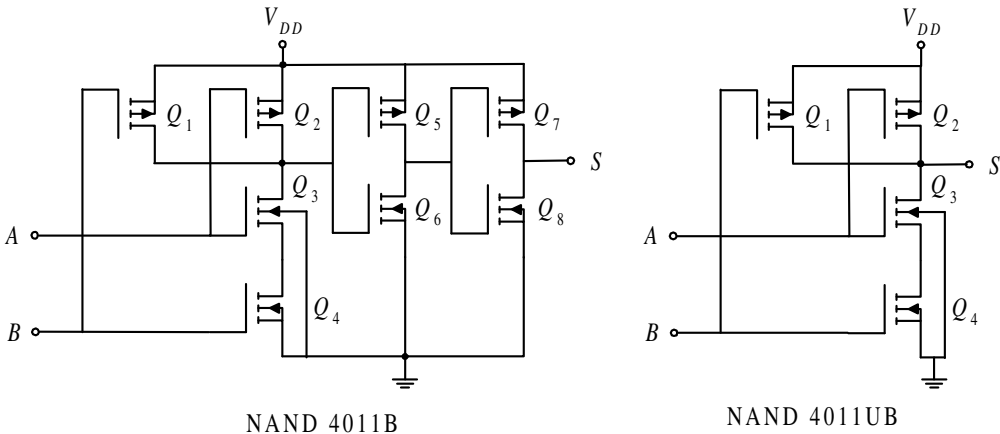


Figura 6.63. Puerta NAND correspondientes al integrado 4011B (*Buffered*) y 4011UB (*UnBuffered*)

### 6.11.2 CMOS - TTL (serie 54C/74C)

La subfamilia 54C/74C surge de la necesidad de sustituir los circuitos TTL por elementos CMOS, en aquellas aplicaciones en las que la tecnología CMOS presenta mayores ventajas. De esta manera, la tecnología CMOS empezó a desplazar el uso masivo hasta hace pocos años de la tecnología bipolar TTL.

Esta subfamilia dispone de una gama de componentes homólogos y compatibles *pin a pin* con los existentes en TTL. Además, no sólo se pueden alimentar a 5 V sino que soportan mayor margen de alimentación, entre 3 y 15 V.

Según el margen de temperatura de trabajo que permitan se distinguen dos tipos: uno, cuyo nombre comienza por 74 con un rango entre  $-40\text{ }^{\circ}\text{C}$  y  $85\text{ }^{\circ}\text{C}$  y que es de uso comercial; y otro, cuyo nombre comienza por 54 con un rango entre  $-55\text{ }^{\circ}\text{C}$  y  $125\text{ }^{\circ}\text{C}$ , que es de uso militar.

### 6.11.3 CMOS de alta velocidad (HCMOS, serie 54HC/74HC)

La subfamilia CMOS de alta velocidad (*High-speed* CMOS: HCMOS) ha alcanzado velocidades comparables a la subfamilia LSTTL pero con consumos muy inferiores, del orden de una millonésima parte en régimen estático.

Si a estas propiedades se le añade la compatibilidad *pin a pin* con la familia TTL, se justifica la gran difusión de la serie HCMOS en el mercado y la clara sustitución de la familia TTL por ella.

Según el margen que permitan de temperatura de trabajo se distinguen dos tipos: uno, cuyo nombre comienza por 74, con un rango entre  $-40\text{ }^{\circ}\text{C}$  y  $85\text{ }^{\circ}\text{C}$ , que es de uso comercial; y otro, cuyo nombre comienza por 54, con un rango entre  $-55\text{ }^{\circ}\text{C}$  y  $125\text{ }^{\circ}\text{C}$ , que es de uso militar.

Existen tres series dentro de HCMOS:

- **54HC/74HC:**

Los niveles lógicos de entrada son reconocibles por la familia CMOS 4000. Su alimentación está comprendida entre 2 y 6 V y dispone de un *buffer* en la salida que permite corrientes  $I_{oL} = 4\text{ mA}$  e  $I_{oH} = -4\text{ mA}$ . El tiempo de propagación es,  $t_p = 6\text{ ns}$ .

- **54HCT/74HCT:**

Las siglas HCT proceden de HCMOS-TTL, debido a que los niveles lógicos de entrada son reconocibles por la familia TTL. Su alimentación es de 5 V  $\pm 10\%$ . Tiene tanto las entradas como las salidas con *buffer*, y su tiempo de propagación es,  $t_p = 6\text{ ns}$ .

- **54HCU/74HCU:**

Las características son iguales a las de la serie HC con la diferencia de que sus salidas no están *bufereadas* (*Unbuffered*), por lo que tiene menor corriente de salida, pero mejor tiempo de propagación,  $t_p = 5$  ns.

#### 6.11.4 CMOS avanzada (ACL, series 74AC y 74ACT)

La subfamilia lógica CMOS avanzada (*Advanced CMOS Logic*, ACL) presenta una apreciable mejora ya que en ella se reducen los dos factores del producto: consumo por tiempo de propagación. Como referencia, cabe indicar que el retardo típico de propagación por puerta es de 3 ns y la disipación de potencia, que una puerta alimentada con 5 V, cargada con 50 pF y trabajando a una frecuencia de conmutación de 100 kHz, es de 0,2 mW y a 10 MHz es de 20 mW, pudiendo trabajar a más de 150 MHz.

La inmunidad al ruido es tres veces superior que en la familia TTL.

Según el margen de temperatura de trabajo que permitan se distinguen dos tipos: uno, cuyo nombre comienza por 74, con un rango entre  $-40$  °C y  $85$  °C, que es de uso comercial, y otro, cuyo nombre comienza por 54, con un rango entre  $-55$  °C y  $125$  °C, que es de uso militar.

Existen dos series dentro de ACL:

- **54AC/74AC:**

Los niveles lógicos de entrada son compatibles con CMOS. La alimentación está comprendida entre 3 y 5,5 V. Dispone de un *buffer* en la salida que permite corrientes  $I_{oL} = 24$  mA e  $I_{oH} = -24$  mA, y su tiempo de propagación es,  $t_p = 3$  ns.

- **54ACT/74ACT:**

Los niveles lógicos de entrada son compatibles con TTL. La alimentación es de  $5$  V  $\pm 10\%$ . Dispone de un *buffer* en la salida que permite corrientes  $I_{oL} = 24$  mA e  $I_{oH} = -24$  mA, y su tiempo de propagación es,  $t_p = 3$  ns.

#### 6.11.5 Otras subfamilias

Aunque las subfamilias se acercan cada vez más a la puerta ideal, aún se continúa investigando en esta línea. La empresa TOSHIBA ha desarrollado la subfamilia VHCT cuyas características son: retardo de propagación  $t_p = 3,5$  ns (frecuencia máxima de funcionamiento 150 MHz) y corrientes de salida de hasta 1 A por puerta.

## 6.12 COMPARATIVA DE LAS SUBFAMILIAS CMOS

Tabla 6.7. Comparación de subfamilias CMOS con  $V_{DD} = 5 V$

<i>Subfamilias CMOS</i>							<i>Unidad</i>
<i>(Parámetros referidos a puerta NAND)</i>							
<i>Parámetro</i>	<i>4000A</i>	<i>4000B</i>	<i>HC</i>	<i>HCT</i>	<i>AC</i>	<i>ACT</i>	
$V_{DD}$	3-12	3-18	2-6	3-6	2-6	4,5-5,5	
$V_{oHmin}$	$V_{CC}-0,01$	$V_{CC}-0,01$	4,9	4,9	4,4	4,4	V
$V_{oLmax}$	0,01	0,01	0,1	0,1	0,1	0,1	V
$V_{iHmin}$	$70\% \cdot V_{CC}$	$70\% \cdot V_{CC}$	3,5	2,2	3,4	2,2	V
$V_{iLmax}$	$30\% \cdot V_{CC}$	$30\% \cdot V_{CC}$	1,5	1,2	1,2	0,8	V
$V_{NMH}$	$30\% \cdot V_{CC}$	$30\% \cdot V_{CC}$	1,4	2,7	1	2,2	V
$V_{NML}$	$30\% \cdot V_{CC}$	$30\% \cdot V_{CC}$	1,4	1,1	1,1	0,7	V
$I_{oHmax}$	-0,5	-1	-4	-4	-24	-24	mA
$I_{oLmax}$	0,5	1	4	4	24	24	mA
$I_{iHmax}$	1	1	1	1	1	1	mA
$I_{iLmax}$	-1	-1	-1	-1	-1	-1	mA
$I_{oS}$	-5	-5	-50	-50	-150	-150	mA
$t_{pLH}$	120	125	18	20	6	5,5	ns
$t_{pHL}$	120	125	18	20	6	5,5	ns
$t_{TLH}$	50	50	7	7	6	6	ns
$t_{THL}$	40	40	7	7	6	6	ns
$f_{max}$	2	2	75	60	125	125	MHz
<i>Fan out</i>	4-LS	2-LS	8-LS	8-LS	50-LS	50-LS	u.c.
$P_D$	0,1	0,1	0,6	0,6	0,6	0,8	mW
<i>Producto</i>							
$P_D t_{pD}$ (100 kHz)	11	1,4	10,8	10,8	3,9	3,6	pJ



### 6.12.1 Resumen de los perfiles de las familias TTL y CMOS

Mediante el conjunto de datos que aportan los perfiles de entrada y salida de las familias TTL y CMOS, que se muestran a continuación, se puede llegar a determinar diferentes características, por ejemplo: compatibilidad, *fan-in*, *fan-out*, inmunidad al ruido, etc.

### 6.12.2 Perfiles de entrada de familias TTL y CMOS

En la Figura 6.64 se comparan los perfiles de entrada de las familias lógicas estudiadas: TTL y CMOS (alimentada a 5 V), clasificados por el valor descendente de  $V_{iLmax}$ .

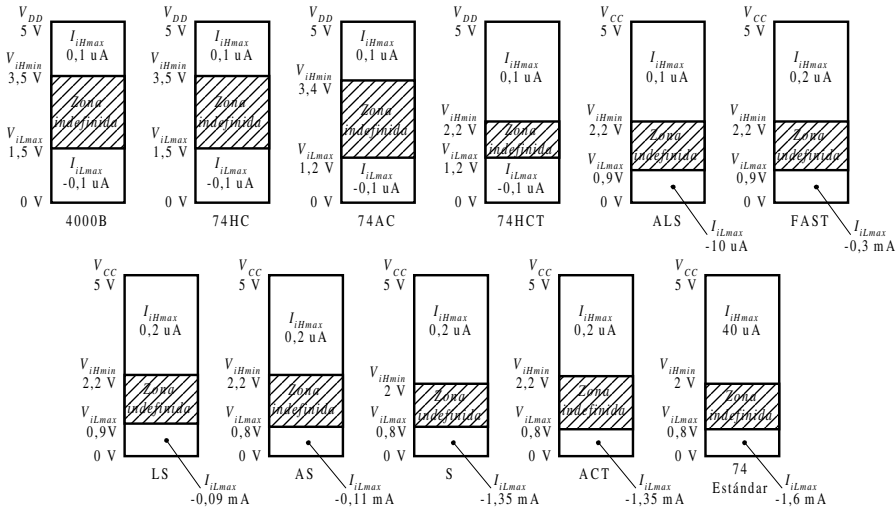


Figura 6.64. Perfiles de entrada de las familias lógicas TTL y CMOS, clasificados por su valor de  $V_{iLmax}$

### 6.12.3 Perfiles de salida de familias TTL y CMOS

En la Figura 6.65 se comparan los perfiles de salida de las familias lógicas estudiadas: TTL y CMOS (alimentada a 5 V), clasificados por el valor descendente de  $V_{oLmax}$ .

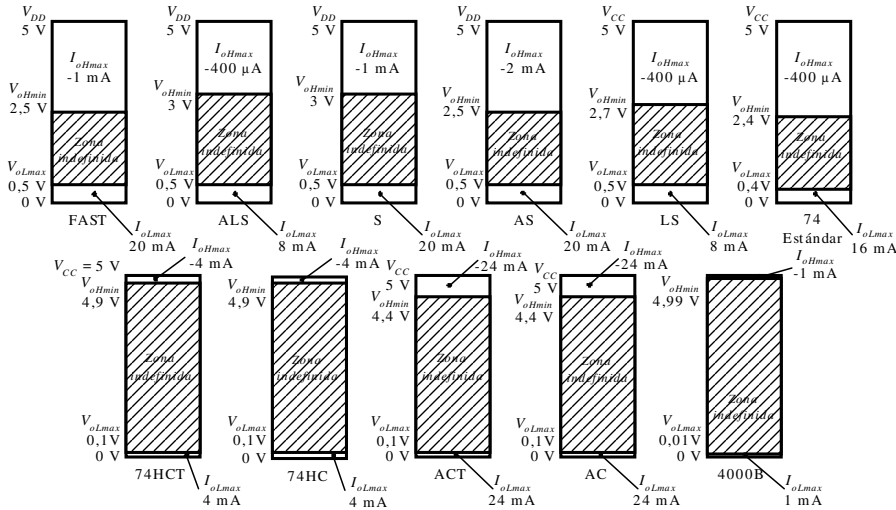


Figura 6.65. Perfiles de salida de las familias lógicas TTL y CMOS, clasificados por su valor de  $V_{oLmax}$

### 6.12.4 Producto consumo por tiempo de propagación

En la Figura 6.66 se compara el valor que toma el producto consumo por tiempo de propagación para cada una de las familias lógicas, TTL y CMOS, estudiadas anteriormente.

## 6.13 PUERTAS CMOS CON OTRO TIPO DE SALIDAS

### 6.13.1 Puertas CMOS con salida en drenador abierto

Al igual que ocurre en la familia TTL, se deduce que en el circuito básico de salida de una puerta CMOS no es posible conectar otras salidas, pues ello provocaría una corriente destructiva cuando conduce, en una de las salidas el transistor *pull-up* y en otra cualquiera el transistor *pull-down*.

En la Figura 6.67 se muestra el circuito de salida CMOS en drenador abierto, mediante transistor de canal N, el cual se puede utilizar para realizar este tipo de conexiones.

La forma de realizar el cableado lógico con este tipo de circuito, es conectando varias salidas de **puertas CMOS en drenador abierto** con una resistencia externa de

polarización *pull-up* cuyo valor se calcula de manera análoga a lo descrito en el caso de la familia TTL, según la expresión [6.35].

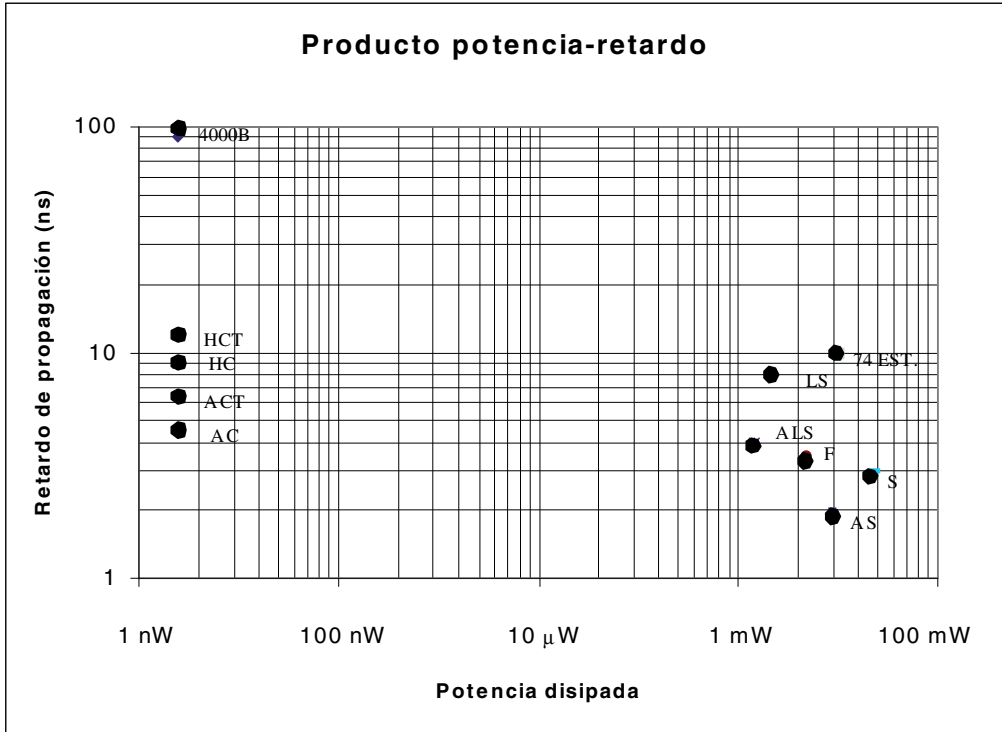


Figura 6.66. Comparación del producto consumo por tiempo de propagación de las familias lógicas TTL y CMOS estudiadas

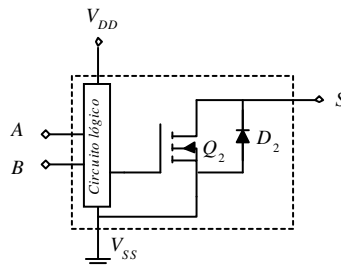


Figura 6.67. Circuito de salida CMOS en drenador abierto

En el caso de sólo conectar en el cableado lógico entradas CMOS, en dicha expresión [6.35], se pueden despreciar las corrientes  $I_{iL}$  e  $I_{iH}$ , por ser estas corrientes en CMOS muy pequeñas, obteniéndose, en este caso, la expresión [6.40].

$$R_{pull-up\ min} = \frac{V_{DDmax} - V_{oL}}{I_{oL}} \quad [6.40]$$

$$R_{pull-up\ max} = \frac{V_{DDmin} - V_{oH}}{N_1 I_{oH}}$$

Donde,  $N_1$  es el número de salidas pertenecientes al cableado lógico.

### 6.13.2 Puertas CMOS con salida triestado

De la misma forma que en la tecnología TTL, CMOS dispone de puertas que permiten triestado (nivel alto, bajo y estado de alta impedancia). Esto se consigue mediante las denominadas **puertas de transmisión**.

El circuito equivalente de una puerta de transmisión puede considerarse como el de un interruptor que al estar abierto o cerrado, aísla o conecta, respectivamente, la salida de la puerta al resto del circuito. Mediante tecnología CMOS, como se muestra en la Figura 6.68, se puede realizar una puerta de transmisión. En la Figura 6.69 la puerta de transmisión está gobernada mediante una señal de control que cuando está a nivel alto conducen ambos transistores unipolares siendo este circuito equivalente al de un interruptor cerrado; y cuando la señal de control está a nivel bajo ambos transistores se encuentran en corte, proporcionando una alta impedancia en la salida.

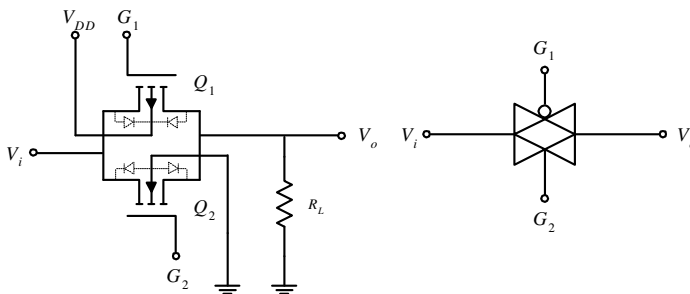


Figura 6.68. Puerta CMOS de transmisión y su símbolo.

Cualquier puerta CMOS puede ser de salida triestado conectando una puerta transmisión como se muestra en la Figura 6.69.

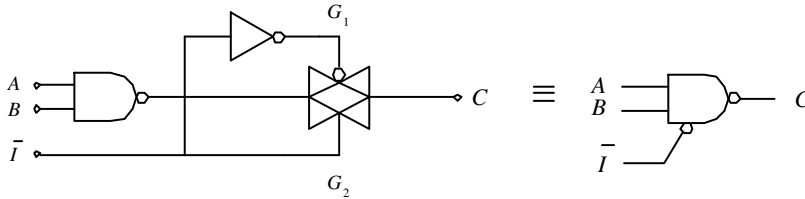


Figura 6.69. Puerta NAND CMOS triestado obtenida mediante una puerta de transmisión

En la Tabla 6.8 se resume el comportamiento de la puerta NAND triestado estudiada.

Tabla 6.8. Tabla de verdad de la función NAND triestado

$\bar{I}$	$B$	$A$	$C$
0	X	X	Z
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

### 6.14 PRECAUCIONES EN LA MANIPULACIÓN DE DISPOSITIVOS CMOS

Las principales precauciones a tener en cuenta en los dispositivos CMOS son relativas a su manipulación por su vulnerabilidad ante cargas electrostáticas, evitar transiciones espurias que elevan la potencia de disipación, y asegurar que tengan alimentación cuando se aplican señales de entrada.

- **Cargas electrostáticas.** La constitución de la puerta aislada del transistor MOS mediante una película muy fina (aproximadamente 1.000 Å), muy aislante, de dióxido de silicio,  $\text{SiO}_2$ , con rigidez dieléctrica aproximada de  $7 \cdot 10^6$  V/cm, puede ser perforada por **descargas electrostáticas** que pueden estar presentes al ser manipulada en condiciones normales (como por ejemplo 70 V). Para evitarlo se deben almacenar los circuitos CMOS pinchados en espuma o bolsas conductoras para cortocircuitar los terminales de las entradas y evitar así la presencia de cargas electrostáticas.

- **Alimentación.** Debe estar presente cuando se aplican señales de entrada para evitar que los diodos de protección entren en conducción. Además es importante no equivocarse los potenciales de alimentación, invirtiéndolos, ya que se produce la conducción de todas las uniones internas, como se muestra en la Figura 6.70.

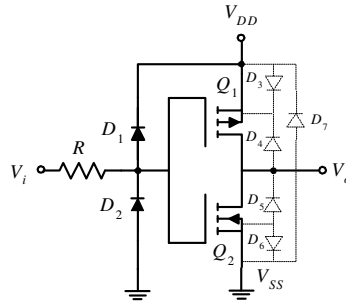


Figura 6.70. Uniones internas de una puerta CMOS

- **Transitorios.** Dado que los dispositivos CMOS tienen una potencia estática despreciable y consumen en las transiciones, siendo su potencia mayor con la frecuencia, se debe:

- ◆ Evitar dejar al aire o sin conectar entradas no utilizadas de dispositivos lógicos, ya que son fuentes de ruido, al actuar como una antena que provoca transiciones espurias.

En las puertas de funciones lógicas básicas (OR, NOR, NAND) una solución sencilla consiste en unir las entradas no utilizadas a otras conectadas en el circuito. Otra solución consiste en poner las entradas no utilizadas de las puertas OR y NOR a nivel bajo  $V_{SS}$ , para que la salida no se modifique. De la misma forma, para el caso de entradas no utilizadas en las puertas AND y NAND se pondrán a nivel alto  $V_{DD}$  sin que se modifique su salida.

- ◆ Los fabricantes especifican el valor máximo de la duración de los flancos de las señales de entrada que evitan una excesiva disipación de potencia, buscando siempre que las señales de entrada tengan flancos lo más abruptos posible.

## 6.15 INTERFACES ENTRE FAMILIAS LÓGICAS

Hay situaciones en las que se deben conectar entre sí diferentes dispositivos, bien para comunicar equipos lógicos diseñados con distinta tecnología, o porque se

requiera aprovechar las diferentes ventajas que aportan las distintas familias lógicas conectándolas entre sí.

Aunque en este apartado sólo se exponen las diferentes situaciones que se pueden presentar al conectar las familias TTL y CMOS y su solución de compatibilidad mediante interfaz, cuando se necesite conectar otras familias lógicas diferentes a las estudiadas se podrán seguir los mismos procedimientos que se describen aquí. En cualquier caso, la interfaz se puede deducir a partir del conocimiento de los perfiles de salida de la puerta excitadora y de los perfiles de entrada de la puerta excitada.

Para conectar distintas familias lógicas se debe garantizar su **compatibilidad**, tanto desde el punto de vista de sus tensiones como de sus corrientes, asegurando que se cumplen las siguientes condiciones:

- **Compatibilidad de tensiones**

Al conectar la salida de un circuito (puerta excitadora) con la entrada de otro (puerta excitada) se debe cumplir:

$$\begin{aligned}V_{oLmax} &\leq V_{iLmax} \\V_{oHmin} &\geq V_{iHmin}\end{aligned}$$

Esta última condición, la del nivel alto, es la que principalmente se debe considerar, ya que, el nivel bajo al ser próximo a cero voltios suele cumplirse en la mayoría de los casos. La interfaz consiste normalmente en disminuir o elevar la tensión de salida  $V_{oH}$  para cumplir dicha condición.

- **Compatibilidad de corrientes**

Al conectar la salida de un circuito (puerta excitadora) con la entrada de otro (puerta excitada) se debe cumplir que:

- ◆ Si la corriente de salida de un circuito es entrante (positiva), en la entrada del otro circuito interconectado deberá ser saliente (negativa), o viceversa.
- ◆ El circuito que ataca o excitador debe suministrar la suficiente corriente en su salida como demande la entrada del circuito atacado o excitado. Es decir, se deben cumplir las siguientes condiciones en los niveles lógicos alto y bajo:

$$\begin{aligned}|I_{oHmax}| &\geq |I_{iHmax}| && \text{(nivel lógico alto)} \\|I_{oLmax}| &\geq |I_{iLmax}| && \text{(nivel lógico bajo)}\end{aligned}$$

Todas estas condiciones de compatibilidad se pueden comprobar fácilmente en las representaciones de los perfiles de entrada y salida de circuitos lógicos, verificando que:

- ◆ para que exista compatibilidad de tensiones, al superponer las ventanas de nivel alto y bajo del perfil de salida de la puerta excitadora sobre sus ventanas homólogas del perfil de entrada de la puerta excitada deben estar contenidos todos sus puntos en esta última,
- ◆ las corrientes de entrada de la puerta excitada y de salida de la puerta excitadora, para un determinado nivel lógico, deben tener signos contrarios y,
- ◆ el perfil de salida de la puerta excitadora, en cada nivel lógico, debe tener un valor absoluto de corriente mayor que el perfil de entrada de la puerta excitada.

Según sea la alimentación de la puerta excitadora: igual, menor o mayor respecto de la alimentación de la puerta excitada existen tres casos de interfaces, que se estudian en los apartados siguientes.

## 6.15.1 Alimentaciones iguales en la puerta excitadora y excitada

### 6.15.1.1 INTERFAZ CMOS (VDD = 5 V) A TTL

Los perfiles de tensión de salida de una puerta CMOS alimentada a 5 V son compatibles con los perfiles de entrada de cualquier subfamilia TTL, por lo que se puede realizar dicha conexión directamente, siempre que la puerta excitadora CMOS sea capaz de suministrar la corriente que demande el circuito excitado.

#### PROBLEMA RESUELTO 6-13

Estudiar la interfaz necesaria para que una puerta CMOS de la serie 74AC excite a una puerta TTL estándar. Calcular el *fan-out* de esta conexión.

#### Solución:

Los perfiles de salida de la puerta excitadora y los perfiles de entrada de la puerta excitada son los representados en la Figura 6.71.

De estos perfiles se puede deducir que se cumplen las condiciones de compatibilidad:

$$V_{oLmax} \leq V_{iLmax} \Rightarrow 0,1 \text{ V} < 0,8 \text{ V}$$

$$V_{oHmin} \geq V_{iHmin} \Rightarrow 4,9 \text{ V} > 2 \text{ V}$$



$$|I_{oHmax}| \geq |I_{iHmax}| \Rightarrow 24 \text{ mA} > 40 \mu\text{A} \quad (\text{nivel lógico alto})$$

$$|I_{oLmax}| \geq |I_{iLmax}| \Rightarrow 24 \text{ mA} > 1,6 \text{ mA} \quad (\text{nivel lógico bajo})$$

El cálculo del *fan-out* será:

$$fan-out_L = \frac{|I_{oLmax}|}{|I_{iLmax}|} = \frac{24 \text{ mA}}{1,6 \text{ mA}} = 15 \quad [6.41]$$

$$fan-out_H = \frac{|I_{oHmax}|}{|I_{iHmax}|} = \frac{24 \text{ mA}}{40 \mu\text{A}} = 600$$

Resultando un *fan-out* de 15; por lo que el máximo número de entradas TTL que pueden ser conectadas a la salida de la puerta CMOS serie 74AC es de 15.

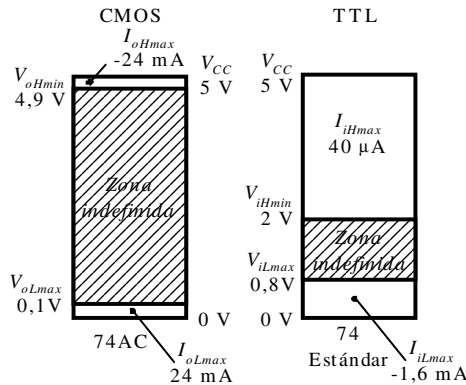


Figura 6.71. Perfiles de salida CMOS serie 74AC y de entrada TTL estándar

### 6.15.1.2 INTERFAZ TTL A CMOS (VDD = 5 V)

En el laboratorio, en condiciones favorables sin la presencia de ruidos, la salida de una puerta TTL ( $V_{oH} \approx 3,6 \text{ V}$ ) puede excitar a una entrada CMOS alimentada a 5 V ( $V_{iHmin} = 3,5 \text{ V}$ ), pero en ambientes industriales o ruidosos el nivel de tensión de la salida TTL disminuye, presentando una incompatibilidad al cumplirse que  $V_{oHmin} < V_{iHmin}$ .

La interfaz necesaria consiste en conectar una resistencia elevadora de nivel *pull-up* entre la salida TTL y la alimentación  $V_{CC}$ , como se muestra en la Figura 6.72, de valor comprendido entre  $1 \text{ k}\Omega$  y  $10 \text{ k}\Omega$  y en cuyo cálculo se aplica la expresión [6.42].

$$R_{pull-up} = \frac{V_{CC} - V_{oL}}{I_{oL}} \quad [6.42]$$

Donde,  $V_{oL}$  es la tensión de salida a nivel bajo TTL que es interpretada como nivel bajo en la entrada CMOS, e  $I_{oL}$  es la corriente, a nivel bajo, que circula por la salida TTL (despreciando la corriente de entrada de la puerta CMOS) que produce la tensión  $V_{oL}$  en la salida TTL.

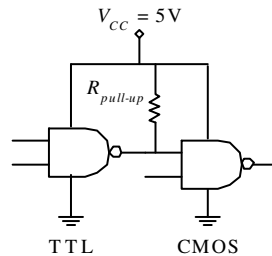


Figura 6.72. Interfaz TTL a CMOS con alimentación 5 V mediante resistencia elevadora pull-up

## PROBLEMA RESUELTO 6-14



Realizar una interfaz TTL a CMOS con alimentación 5 V mediante resistencia elevadora *pull-up*, utilizando para ello el programa de simulación *Electronics Workbench*.

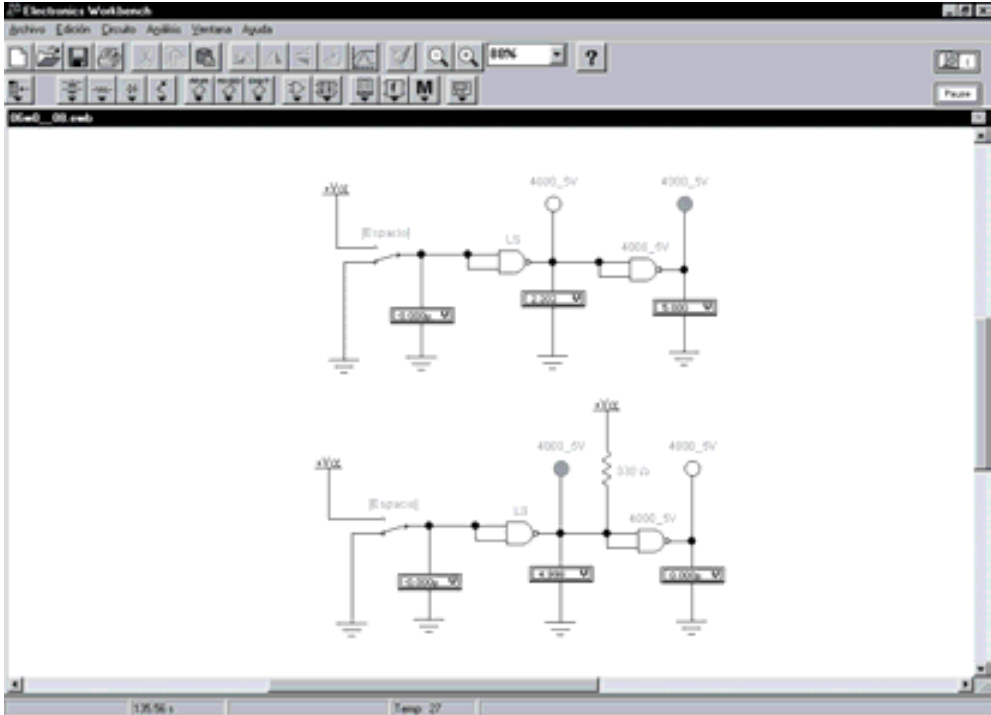
### Solución:

Se introduce en el programa de simulación *Electronics Workbench* el circuito interfaz TTL a CMOS con alimentación 5 V. Para comprobar el funcionamiento de la interfaz se ha simulado el mismo circuito con y sin resistencia *pull-up* como se muestra en la Figura 6.73, verificando la necesidad de dicha resistencia para mantener los correctos niveles lógicos de la interfaz. El valor de la resistencia *pull-up* viene determinado por la expresión [6.42], considerando que la salida de la puerta TTL tiene una unidad de carga TTL,  $I_{oL} = 1,6$  mA.

$$R_{pull-up} = \frac{V_{CC} - V_{oL}}{I_{oL}} = \frac{5 \text{ V} - 0,4 \text{ V}}{1,6 \text{ mA}} = 2875 \Omega \cong 2\text{k}7 \Omega$$

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap06\Ewb5\06W0\_08.ewb**



*Figura 6.73. Simulación de la Interfaz TTL a CMOS con alimentación 5 V mediante resistencia elevadora pull-up*

### 6.15.1.3 INTERFACES ESPECÍFICAS HCT Y ACT

Las entradas de las subfamilias HCT y ACT, de tecnologías CMOS, han sido diseñadas para ser compatibles con las diferentes subfamilias TTL y CMOS, por lo que son otra alternativa para realizar una interfaz, de forma que una salida TTL excite a una entrada HCT o ACT y la salida de ésta a una entrada CMOS.

En la Figura 6.74 se muestra la interfaz TTL a CMOS con alimentación 5 V intercalando una puerta HCT o ACT entre ambas.

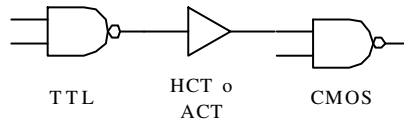


Figura 6.74. Interfaz TTL a CMOS intercalando una puerta HCT o ACT

### 6.15.2 Alimentación de la puerta excitadora menor que la de la puerta excitada

La interfaz necesaria en esta situación deberá aumentar el nivel de salida de la puerta excitadora para que sea reconocido como nivel alto por la entrada de la puerta excitada. Se deben considerar los siguientes casos:

- La **puerta excitadora soporta la tensión de alimentación de la puerta excitada**. La interfaz necesaria consiste en conectar una resistencia elevadora de nivel entre la salida de la puerta excitadora y la alimentación  $V_{CC}$  de la puerta excitada, tal como se muestra en la Figura 6.75. El cálculo de la resistencia elevadora de nivel *pull-up* es el mismo que se ha indicado anteriormente en la expresión [6.42].

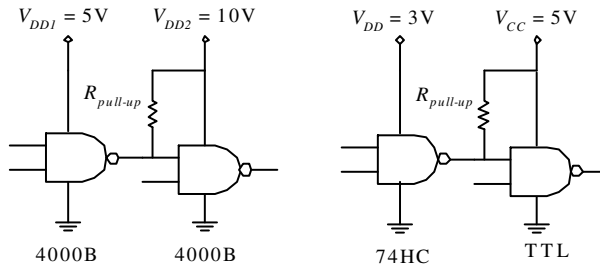


Figura 6.75. Interfaz cuando la alimentación de la puerta excitadora es menor que la de la excitada y la puerta excitadora soporta la alimentación de la puerta excitada

- La **puerta excitadora no soporta la tensión de alimentación de la puerta excitada**, lo que provoca la destrucción (por tensión de ruptura) del transistor de salida de la puerta excitadora al no soportar una tensión superior a la de su  $V_{CCmax}$ .

La interfaz necesaria consiste en utilizar puertas en colector o drenador abierto (según sea la puerta excitadora TTL o CMOS respectivamente) y polarizar el transistor de esta puerta mediante una resistencia *pull-up* conectada entre su

salida, en colector o drenador abierto, y la alimentación  $V_{CC}$  de la puerta excitada, como se muestra en la Figura 6.76.

El cálculo de la resistencia *pull-up* es el mismo que se ha indicado anteriormente en la expresión [6.42].

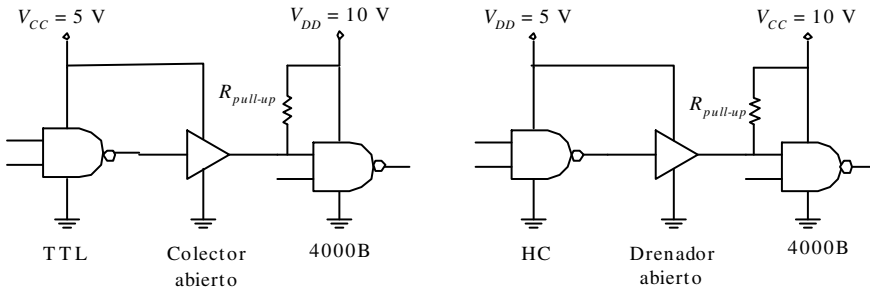


Figura 6.76. Interfaz cuando la alimentación de la puerta excitadora es menor que la de la excitada y la puerta excitadora no soporta la alimentación de la puerta excitada

### PROBLEMA RESUELTO 6-15



Realizar una interfaz HC a CMOS con alimentación 10 V mediante puerta en drenador abierto y resistencia elevadora *pull-up*, utilizando para ello el programa de simulación *Electronics Workbench*.

#### Solución:

Para realizar la simulación de este circuito se introduce el diseño del mismo en el programa *Electronics Workbench*, tal como se muestra en la Figura 6.77. La interfaz está compuesta por una puerta en drenador abierto HC-OD y una resistencia *pull-up* conectada al potencial de 10 V.

Para comprobar el funcionamiento de la interfaz se modifica el estado del nivel lógico de la entrada (cambiar la posición del interruptor mediante la barra espaciadora), verificando que los valores correspondientes de tensión en la interfaz son los correctos.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap06\Ewb5\06W0\_\_09.ewb**

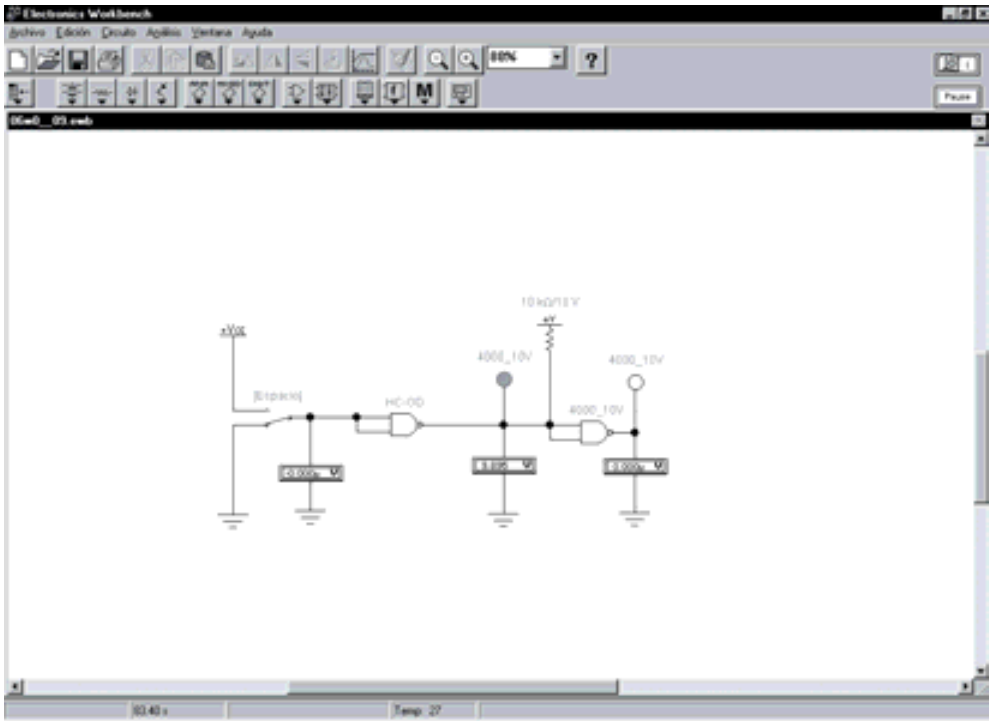


Figura 6.77. Simulación de la Interfaz HC a CMOS con alimentación 10 V mediante puerta en drenador abierto (HC-OD) y resistencia elevadora pull-up

### 6.15.2.1 INTERFAZ ESPECÍFICA 4104

No obstante, es importante tener presente que existen en el mercado circuitos integrados adaptadores de nivel que sirven de interfaz cuando la alimentación de la puerta excitadora es menor que la de la puerta excitada, sustituyendo a interfaces con puertas en colector o drenador abierto y resistencias *pull-up*.

El circuito integrado 4104, mostrado en la Figura 6.78, dispone de dos alimentaciones,  $V_{CCinput}$  y  $V_{DDoutput}$ , definidas externamente por el usuario. Esto permite adaptar los niveles de tensión, elevando el de salida respecto al de entrada cuando  $V_{CCinput} < V_{DDoutput}$  o reduciéndolo cuando  $V_{CCinput} > V_{DDoutput}$ .

Si se desea obtener más información sobre las características del circuito integrado 4104 se puede consultar el programa de Fairchild Semiconductors de **Data Sheets** que se incluye en el CDROM#2 que acompaña a este libro.

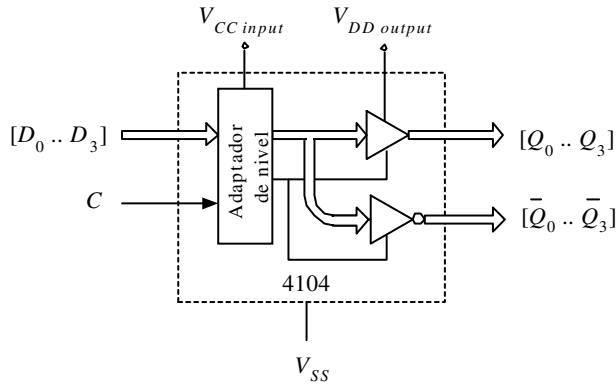


Figura 6.78. Circuito integrado adaptador de nivel 4104

### 6.15.3 Alimentación de la puerta excitadora mayor que la de la puerta excitada

La interfaz necesaria para esta situación deberá disminuir el nivel de salida de la puerta excitadora para que sea reconocido como nivel alto por la entrada de la puerta excitada.

Una posible interfaz consiste en el circuito mostrado en la Figura 6.79 que consta de un diodo tipo *Schottky* y una resistencia *pull-up*. Cuando la salida de la puerta excitadora está a nivel alto el diodo no conduce y a través de la resistencia *pull-up* se aplica, a la entrada de la puerta excitada, un nivel apropiado de tensión alto, próximo al de alimentación de la puerta excitada. Cuando la salida de la puerta excitadora está a nivel bajo el diodo conduce aplicando, a la entrada de la puerta excitada, un nivel de tensión bajo, de valor próximo a su tensión de codo, por ello conviene que esta tensión de codo sea pequeña como ocurre en los diodos *Schottky*.

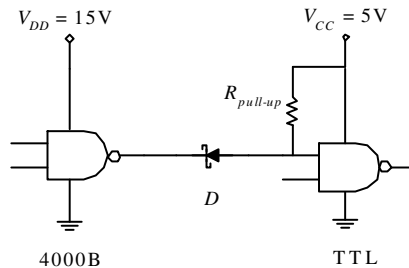


Figura 6.79. Interfaz mediante resistencia pull-up y diodo cuando la alimentación de la puerta excitadora es mayor que la de la puerta excitada

Otra solución, ya estudiada anteriormente, consiste en utilizar puertas de colector o drenador abierto, tal como se muestra en la Figura 6.80.

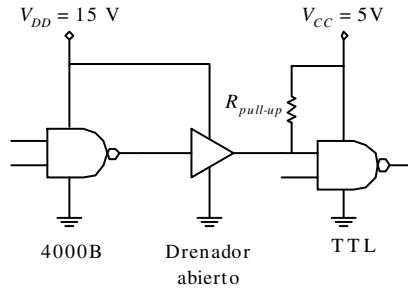


Figura 6.80. Interfaz mediante colector o drenador abierto y resistencia *pull-up* cuando la alimentación de la puerta excitadora es mayor que la de la puerta excitada

## PROBLEMA RESUELTO 6-16



Realizar una interfaz CMOS con alimentación de 15 V a TTL mediante una puerta en drenador abierto (OD a 15 V) y una resistencia *pull-up*, utilizando para ello el programa de simulación *Electronics Workbench*.

### Solución:

Para realizar la simulación de este circuito se introduce el diseño del mismo en el programa *Electronics Workbench*, tal como se muestra en la Figura 6.81. La interfaz, que se corresponde con la Figura 6.80, está compuesta por una puerta en drenador abierto OD alimentada a 15 V y una resistencia *pull-up* conectada al potencial de 5 V.

Para comprobar el funcionamiento de la interfaz, se modifica el estado del nivel lógico de la entrada (cambiar la posición del interruptor mediante la barra espaciadora) y verificando que los valores de tensión en la interfaz son los correctos.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap06\Ewb5\06W0\_10.ewb**



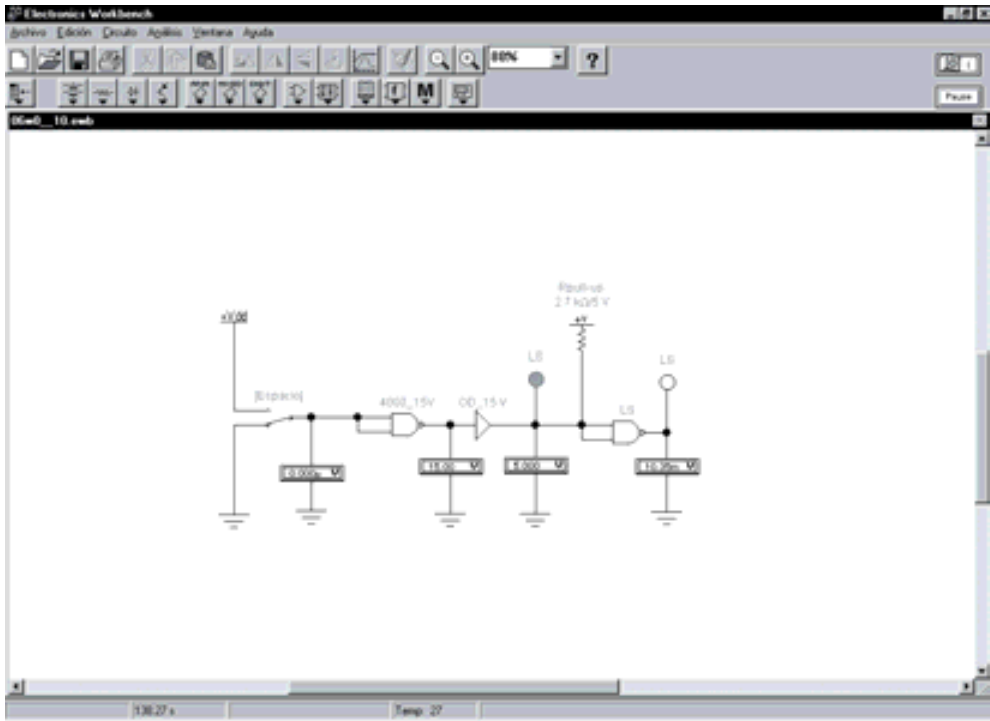


Figura 6.81. Simulación de la Interfaz CMOS de 15 V a TTL mediante puerta en drenador abierto (OD a 15 V) y resistencia elevadora pull-up

### 6.15.3.1 INTERFACES ESPECÍFICAS 4049 Y 4050

Al igual que en el caso anterior, también existen en el mercado circuitos integrados adaptadores de nivel que sirven de interfaz cuando la alimentación de la puerta excitadora es mayor que la de la excitada, sustituyendo a interfaces con puertas en drenador abierto, diodos y resistencias *pull-up*.

Los circuitos integrados 4049 y 4050, que se muestran en la Figura 6.82, permiten adaptar los niveles de tensión, reduciendo el de salida respecto al de entrada al ser  $V_{input} > V_{DD}$ .

Si se desea obtener más información sobre las características de los circuitos integrados 4049 y 4050 se puede consultar el programa de Fairchild Semiconductors de **Data Sheets** que se incluye en el CDROM#2 que acompaña a este libro.

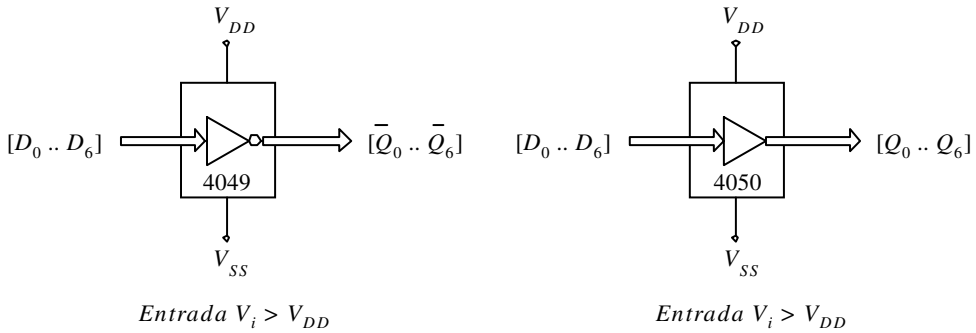


Figura 6.82. Circuitos integrado adaptadores de nivel alto a bajo 4049 y 4050

### 6.15.4 Tabla resumen de interfaces mediante adaptadores de niveles entre familias lógicas

Tabla 6.9. Resumen de interfaces mediante adaptadores de niveles entre familias lógicas

Puerta excitadora	Puerta excitada						
	TTL, S, LS, ALS, AS y F	4000B 5 V	4000B 3-4,5 V	4000B 6-15 V	HC/AC 5 V	HC/AC 3 V	HCT/ACT
TTL, S, LS, ALS, AS y F	Directa	Resist. pull-up	4050	4104	Resist. pull-up	4104 4050	Directa
CMOS4000B 5 V	Directa	Directa	4050	4104	Directa	4104 4050	Directa
CMOS4000B 3-4,5 V	4104	4104	(1)	4104	4104	4104 4050	4104
CMOS4000B 6-15 V	4050	4050	4050	(1)	4050	4104 4050	4050
HC/AC 5 V	Directa	Directa	4050	4104	Directa	Directa	Directa
HCT/ACT	Directa	Directa	4050	4104	Directa	4104 4050	Directa

**Notas:**

(1) Directa en el caso de que la alimentación sea igual en la puerta excitadora y en la puerta excitada. En caso contrario, se utiliza un circuito, 4104 o 4050, según corresponda.

Como ya se ha estudiado anteriormente, los circuitos integrados 4104 y 4050 pueden ser sustituidos por puertas en colector o drenador abierto y resistencia *pull-up*.

**PROBLEMA RESUELTO 6-17**

Realizar una interfaz CMOS con alimentación de 15 V a TTL mediante una puerta en drenador abierto (OD a 15 V) y el circuito interfaz específico, utilizando para ello el programa de simulación *Electronics Workbench*.

**Solución:**

Introducir en el programa de simulación *Electronics Workbench* el circuito correspondiente al Problema resuelto 6-16 añadiendo el circuito interfaz 4050 alimentado a 5 V como se muestra en la Figura 6.83. Para comprobar el funcionamiento de la interfaz se modifica el estado del nivel lógico de la entrada (cambiar la posición del interruptor mediante la barra espaciadora), y verificar que los valores de tensión en la interfaz son los correctos.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap06\Ewb5\06W0\_\_11.ewb**

## 6.16 SIMULACIÓN EN VHDL DE UNA FAMILIA LÓGICA

Para **simular una familia lógica mediante el lenguaje de descripción *hardware* VHDL** se deben especificar las características particulares que definen a dicha familia lógica y que la diferencia de las demás, como son: los retardos, los distintos niveles lógicos y las salidas especiales que pueden adoptar, etc.

### 6.16.1 Asignaciones con retrasos

En primer lugar, en VHDL es importante conocer la **diferencia entre variables y señales**:

- Las **variables** son elementos abstractos a los que se puede modificar su valor instantáneamente dentro de un proceso secuencial.

- Las **señales**, con significado físico en procesos secuenciales y concurrentes, tienen dos partes: una, donde se escribe (denominada fuente) y otra, donde se lee que no tiene por qué ser igual a lo escrito. Existe conexión entre ellas, siendo posible su desconexión mediante la palabra clave **NULL**. Pueden ser de tipo: **normal** (cuando no se pueden desconectar las fuentes), **bus** (con valor por defecto cuando todas las fuentes están desconectadas) y **register** (si conserva el último valor escrito cuando están desconectadas todas las fuentes).

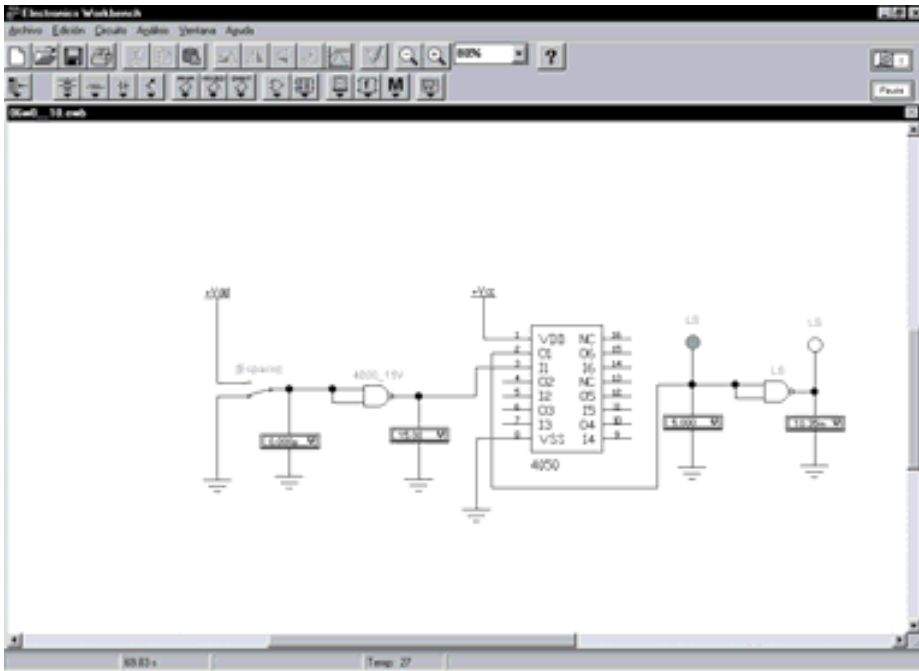


Figura 6.83. Simulación de la interfaz CMOS con alimentación de 15 V a TTL mediante circuito específico 4050

Cuando en VHDL se asigna un retardo a una señal, lo que se hace es anotar en una lista (tipo cola) los valores futuros que tomará la señal ordenados por tiempo. En cada asignación se van añadiendo nuevos eventos a la lista, ordenados según sus retardos asociados.

La **forma de asignar los retardos** puede clasificarse en: retardos inerciales y retardos transportados.

Considérese un inversor con un retardo de propagación  $t_p$  de 10 ns. Cada vez que su señal de entrada  $a$  cambie, se introduce en la lista de eventos de la señal de salida  $S$  el valor de la entrada complementado, con un retardo de 10 ns. Es importante estudiar aquí lo que ocurre si la salida  $S$  debe cambiar antes de los 10 ns, por haberse aplicado

en su entrada *a* un impulso de duración menor que su retardo, por ejemplo de 5 ns. En esta situación se deben considerar dos casos:

- 1) que el segundo evento se añada a la lista, la cual ahora estaría formada por dos eventos, produciéndose el denominado **retardo transportado**, o,
- 2) que el segundo evento se añada a la lista eliminando al anterior, produciéndose el denominado **retardo inercial**.

En la Figura 6.84 se representan los cronogramas a un retardo inercial y a un retardo transportado, correspondientes a un inversor con un retardo  $t_p$  de 10 ns.

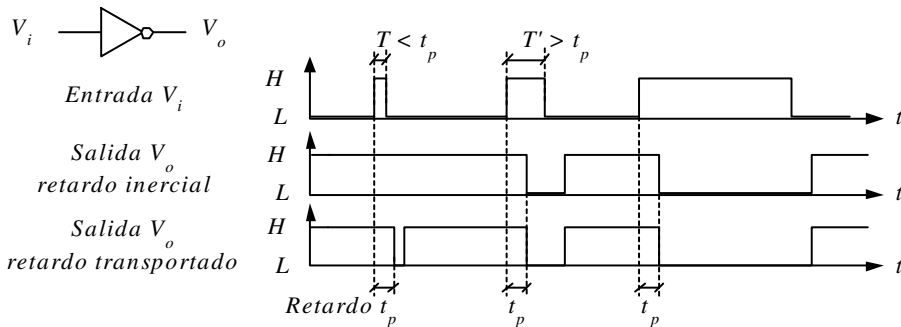


Figura 6.84. Cronogramas de retardo inercial y transportado

En simulación mediante VHDL, la **sintaxis completa de la asignación** con retardos es:

```
[identificador_asignacion]
señal <= [TRANSPORT | [REJECT tiempo]] INERTIAL]
        forma_onda { , forma_onda } | UNAFFECTED;
```

Cuando se realiza una asignación con retardo en una señal, su valor se asigna a su fuente y transcurrido el retardo se transfiere a la señal en sí, pudiéndose leer este valor. La palabra clave es **AFTER**, como por ejemplo:

```
S <= '1' AFTER 10 ns;
```

La forma de realizar varias asignaciones a una misma señal, en la misma sentencia, es mediante una lista de valores futuros ordenados en el tiempo, como por ejemplo:

```
S <= '1' AFTER 20 ns, '0' AFTER 25 ns, '1' AFTER 30 ns;
```

### Retardos inerciales

Muchos dispositivos lógicos requieren que sus señales de entrada tengan una duración mínima para que sean reconocidas o tengan efecto, es decir, ciertos

impulsos de entrada de corta duración no tienen efecto sobre la salida, pudiéndose considerar que son filtrados por el dispositivo (eliminados debido a su frecuencia).

En VHDL los retardos por defecto son inerciales. Por ejemplo:

```
S <= not a AFTER 10 ns;
```

y

```
S <= INERTIAL not a AFTER 10 ns;
```

son sentencias equivalentes.

Mediante la palabra clave **REJECT** se puede especificar el intervalo de tiempo durante el cual es posible que un nuevo evento elimine a los que existen en la cola, sin necesidad de que coincida con su retardo de propagación  $t_p$ , tal y como ocurre en los retardos inerciales.

Por ejemplo:

```
S <= REJECT 3 ns INERTIAL not a AFTER 10 ns;
```

En este caso los impulsos en  $a$  inferiores a 3 ns son filtrados y no modifican la señal de salida  $S$ .

### Retardos transportados

Mediante la palabra clave **TRANSPORT** la salida del dispositivo siempre es afectada por un pulso de la señal de entrada, con independencia de su duración, creándose una lista de eventos y procesándose en el tiempo, en función de su retardo. Por ejemplo:

```
S <= TRANSPORT not a AFTER 10 ns;
```

En este caso todos los impulsos presentes en  $a$  modifican la señal de salida  $S$ . Ningún impulso es filtrado.

En el caso de asignaciones múltiples, sólo la primera es inercial, siendo el resto transportadas, pues de lo contrario se perdería la información de toda la lista al ser sólo la última asignación válida, eliminándose las anteriores.

```
S <= '0' AFTER 5 ns, '1' AFTER 20 ns;
```

### PROBLEMA RESUELTO 6-18



Describir y simular mediante el programa *VeriBest VB99.0* la respuesta de la salida  $S$  de un inversor con retardo de propagación de 10 ns que no responda a impulsos de entrada inferiores a 3 ns. Asimismo, considerar la salida  $Y$  de un inversor con retardo de 5 ns que complementa impulsos de cualquier duración.

### Solución:

En la Figura 6.85 se muestra la descripción en lenguaje VHDL de las puertas inversoras con los retardos indicados en el enunciado del problema, donde  $a$  es la señal de entrada común a ambos inversores y  $S$  e  $Y$  sus respectivas salidas.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap06\VB99\06V0\_\_01\06V0\_\_01.vpd**



```

-- Fichero :      D:\Ejemplos\Cap06\VB99\06V0__01\06V0__01.vhd
-- Lib_trabajo:  D:\Ejemplos\Cap06\VB99\06V0__01\WORKLIB
--              Descripción de puertas NOT con retardos
--              inerciales y transportados
--
-----
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY puerta_not IS
    PORT( a : in bit;
          S : out bit;
          Y : out bit);
END puerta_not;

-- Descripción coportamental
ARCHITECTURE flujo_datos OF puerta_not IS

BEGIN
    S<= REJECT 3 ns INERTIAL NOT a AFTER 10 ns;
    Y<= TRANSPORT NOT a AFTER 5 ns;

END flujo_datos;

```

Figura 6.85. Descripción en lenguaje VHDL de puertas inversoras con los retardos

En la Figura 6.86 se muestran los resultados que se obtienen con el programa *VeriBest VB99.0* al realizar la simulación de la descripción en lenguaje VHDL de la figura anterior.

Obsérvese cómo se puede describir que ciertos impulsos de entrada de corta duración no tengan efecto sobre la salida  $S$ , pudiéndose considerar que son filtrados por el dispositivo.

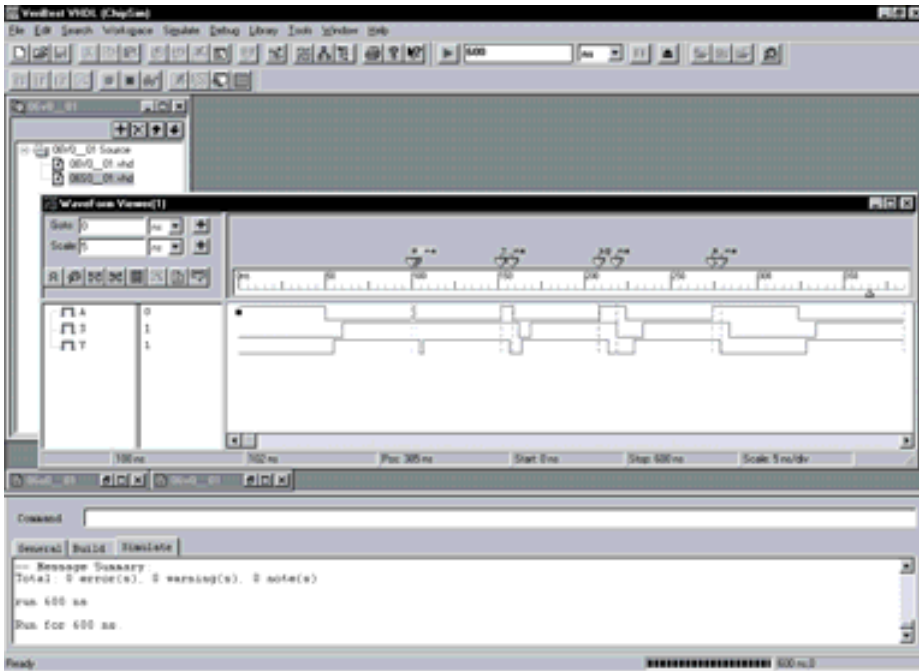


Figura 6.86. Simulación a partir de descripción en lenguaje VHDL de puertas inversoras con retardos

## PROBLEMA RESUELTO 6-19



Describir y simular las respuestas de salida de inversores correspondientes a las subfamilias: TTL-estándar, HC, HCT, LS y CMOS, utilizando para ello el módulo *Simulate Demo* del programa *OrCAD Demo v9*. Este fichero se debe abrir directamente desde el módulo indicado para evitar posibles problemas durante su ejecución.

### Solución:

En la Figura 6.87 se muestra la descripción en lenguaje VHDL de las puertas inversoras con sus correspondientes retardos, donde  $a$  es la señal de entrada común a todas las puertas e  $Y_{TTL}$ ,  $Y_{HC}$ ,  $Y_{HCT}$ ,  $Y_{LS}$ ,  $Y_{CMOS}$  las salidas correspondientes a cada uno de ellos.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap06\OrCAD\06R0\_\_03\06R0\_\_03.opj**

En la Figura 6.88 se muestran los retardos que se obtienen en *OrCAD Demo v9* al realizar dicha simulación.



```
06_03
2:
3: -- Fichero : D:\Ejemplos\Cap06\OrCAD9\06RO_03\06RO_03.vhd
4: -- Lib_trabajo: D:\Ejemplos\Cap06\OrCAD9\06RO_03\WORKLIB
5: -- Descripción de puertas NOT de diferentes subfamilias
6: ==
7: -----
8:
9: LIBRARY ieee;
10: USE ieee.std_logic_1164.all;
11:
12: ENTITY puertas_not IS
13:   PORT(
14:     a : in std_logic;
15:     Y_TTL : out std_logic;
16:     Y_NC : out std_logic;
17:     Y_NCT : out std_logic;
18:     Y_LS : out std_logic;
19:     Y_CMOS : out std_logic);
20: END puertas_not;
21:
22: == Descripción comportamental
23:
24: ARCHITECTURE flujo_datos OF puertas_not IS
25:
26: BEGIN
27:   Y_TTL <= NOT a AFTER 22 ns;
28:   Y_NC <= NOT a AFTER 1500 ps;
29:   Y_NCT <= NOT a AFTER 800 ps;
30:   Y_LS <= NOT a AFTER 10 ns;
31:   Y_CMOS <= NOT a AFTER 80 ns;
32:
33: END flujo_datos;
```

Figura 6.87. Descripción en VHDL de puertas inversoras de distintas subfamilias

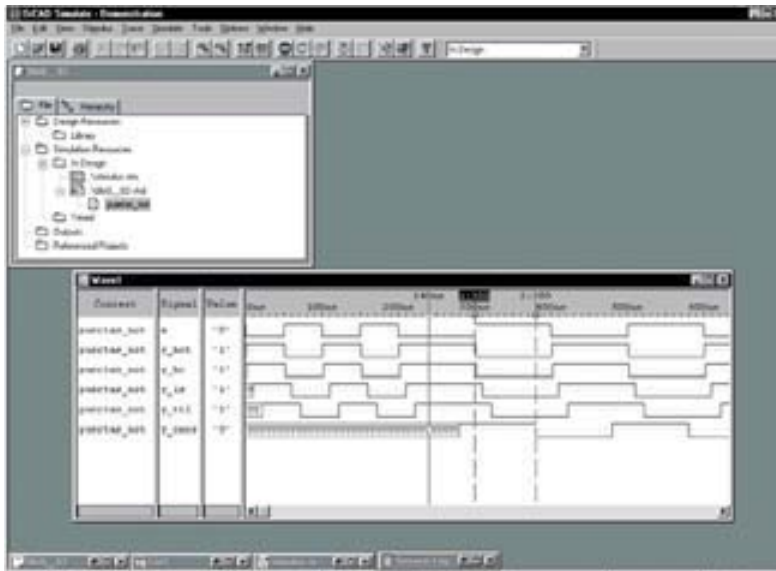


Figura 6.88. Simulación en VHDL de puertas inversoras de diferentes subfamilias

Obsérvese los efectos (zonas de niveles indefinidos 'U') que producen los retardos, a determinadas frecuencias, en las respuestas de las puertas según la subfamilia, en particular en las tecnologías más lentas, como por ejemplo, TTL-estándar y CMOS.

### 6.16.2 Simulación de niveles lógicos

Los tipos básicos **bit** cuyos valores lógicos sólo pueden ser '1' o '0' no se corresponden con la realidad en simulación, ya que no contemplan el estado de alta impedancia (salidas en triestado) o en colector/drenador abierto, ni determinadas situaciones reales que se producen cuando varias señales están conectadas a una misma línea de un *bus*.

Para modelar con mayor detalle físico una señal o línea de un *bus*, existe un modelo basado en **fuerzas y valores lógicos** que define los posibles estados lógicos que pueden adoptar una señal o nodo de un circuito lógico. Mediante los parámetros de fuerza y el valor lógico se puede modelar cualquier familia lógica.

Los valores que se pueden tomar en cada caso son los siguientes:

- Los **valores lógicos** pueden ser tres: **1** (alto), **0** (bajo) y **X** (desconocido).
- Las **Fuerzas** pueden ser:

- F:** **Conexión directa.** Conexión de la salida a la alimentación o masa mediante una conexión directa. Es la de mayor fuerza.
- S:** **Strong.** Conexión de la salida a la alimentación  $V_{CC}$  ( $V_{DD}$ ) o a masa GND mediante un transistor (salida en *Totem-Pole*).
- R:** **Resistiva.** Conexión de la salida a alimentación o a masa mediante una resistencia (salida en colector/drenador o emisor/fuente abierto/a).
- Z:** **Alta impedancia.** Conexión de la salida a alimentación o masa mediante una alta impedancia (salida triestado).
- I:** **Indeterminada.** No se conoce la fuerza que hay en la línea.

Para evitar la asignación de una señal o línea de *bus* con dos cantidades (fuerza y valor lógico), en VHDL se puede expresar mediante una cantidad única, definida en un tipo enumerado **std\_logic** del paquete **std\_logic\_1164** de la biblioteca **IEEE**, representado en la Tabla 6.10. En dicha tabla se observa que en este tipo enumerado no está definida la fuerza *F*.

Se denomina **contención** cuando varios dispositivos escriben al mismo tiempo sobre una misma línea o señal. En contenciones se evalúa primero el valor de la

fuerza, siendo: **S** fuerte; **R** débil y **Z** prácticamente nula. El resultado de la contención consiste en tomar la fuerza más fuerte y su valor lógico.

Tabla 6.10. Operadores predefinidos en VHDL, en el paquete *Std\_logic\_1164*

<code>std_logic</code>	Fuerza-valor	Descripción	Puerta típica
'U'		Condiciones iniciales no definidas	
'X'	SX	Desconocido fuerte	<i>Totem-Pole</i>
'0'	S0	'0' fuerte	<i>Totem-Pole</i>
'1'	S1	'1' fuerte	<i>Totem-Pole</i>
'W'	RX	Desconocido resistivo	Emisor/fuente o colector/drenador abierto
'L'	R0	'0' resistivo	Emisor/fuente abierto/a
'H'	R1	'1' resistivo	Colector/drenador abierto
'Z'	ZX	Alta impedancia	Triestado
'_'		No importa	

Para **evitar la contención**, los dispositivos, escriben en los *buses* a través de *buffers* de tipo:

- **colector abierto** (AND cableada), de señal resistiva débil o,
- **triestado** (alta impedancia), de señal prácticamente nula.

Si las fuerzas son iguales se mira su valor lógico. En caso de que sean iguales sus valores lógicos se toma cualquiera de ellos y, si son distintos depende de la tecnología del *bus*.

En VHDL se definen **funciones de resolución** que, como su nombre indica, determinan el nivel lógico en todas las posibles contenciones que se presenten en el *bus*. Los valores asignados por estas funciones de resolución se establecen mediante la utilización de **tipos resueltos**.

Un **tipo resuelto** se define mediante una declaración de subtipo y el nombre de la función de resolución, que calcula el valor de la fuente, en función de todas las asignaciones que se están produciendo.

Dentro de los tipos predefinidos en VHDL se encuentran: **bit** y **bit\_vector**, que son no resueltos y por lo tanto no se les puede usar en *buses* ni en señales en los que varios procesos escriban al mismo tiempo. **Std\_logic** y **std\_logic\_vector** poseen nueve niveles lógicos y función de resolución en el paquete **std\_logic\_1164** de la biblioteca **IEEE** ya indicados en la Tabla 6.10. Sin embargo, **std\_ulogic** y **std\_ulogic\_vector** son los mismos que los anteriores, pero sin función de resolución (no resueltos).

## PROBLEMA RESUELTO 6-20



Describir y simular la función OR-exclusiva expresada en términos *maxterm* usando en los productos el cableado lógico mediante descripciones de modelo basado en **fuerzas y valores lógicos**, utilizando para ello el programa *VeriBest VB99.0*.

### Solución:

Como referencia para resolverlo se puede ver también el Problema resuelto 6-10 en el que se ha realizado este mismo problema mediante el programa de simulación *Electronics Workbench*.

La función XOR expresada en *maxterm* tiene como ecuación lógica:

$$S = (b + a)(\bar{b} + \bar{a})$$

El circuito se puede realizar mediante el lenguaje VHDL utilizando una descripción estructural en la que en dos puertas OR en colector abierto se unen sus salidas para obtener la función AND mediante cableado lógico, como se muestra en la Figura 6.89.

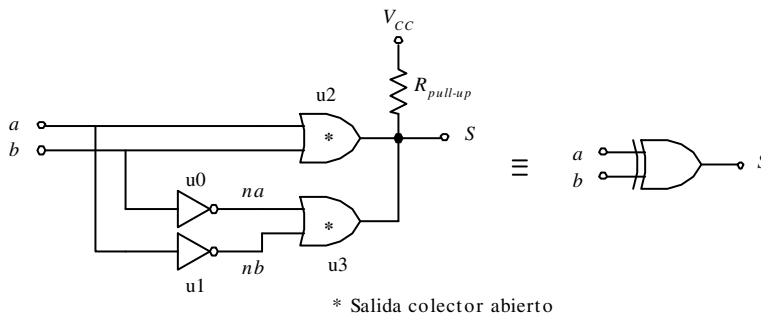


Figura 6.89. Circuito de una puerta XOR con puertas con salida en colector abierto

El contenido de los ficheros, en lenguaje VHDL, que describen de forma estructural cada componente de una puerta XOR utilizando puertas NOT y OR con salidas en colector abierto es el que se muestra en la Figura 6.90, en la Figura 6.91 y en la Figura 6.92 que se muestran a continuación.

```

-- Fichero : D:\Ejemplos\Cap06\VBv99\06V0_02\06V1_02.vhd
-- Lib_trabajo: D:\Ejemplos\Cap06\VBv99\06V0_02\WORKLIB
-- Descripción del componente puerta NOT con
-- asignación de niveles lógicos y retardos
--
-----
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-- Definición de entidades y arquitectura del componente
ENTITY inv IS PORT(e:IN std_logic; y:OUT std_logic);END inv;

ARCHITECTURE rtl OF inv IS BEGIN y<=NOT e AFTER 10 ns;END rtl;

"F:\Cae\Industri\Digital\Ejemplos\Cap06\VBv99\06V0_02\06V1_02.vhd Ln 14, Col 47

```

Figura 6.90. Descripción en lenguaje VHDL del componente puerta inversora con asignación de niveles lógicos y retardos

```

-- Fichero : D:\Ejemplos\Cap06\VBv99\06V0_02\06V2_02.vhd
-- Lib_trabajo: D:\Ejemplos\Cap06\VBv99\06V0_02\WORKLIB
-- Descripción del componente puerta OR en
-- colector abierto y con asignación de retardos
-- de 10 ns
--
-----
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Definición de entidades y arquitectura del componente

ENTITY or2 IS PORT(e1,e2: IN std_logic; y: OUT std_logic);END or2;

ARCHITECTURE rtl OF or2 IS BEGIN
  process (e1,e2)
  begin
    if e1='1' or e2='1' THEN y<='H' AFTER 10 ns;
    ELSE y<='0' AFTER 10 ns;
    END if;
  END process;
END rtl;

"F:\Cae\Industri\Digital\Ejemplos\Cap06\VBv99\06V0_02\06V2_02.vhd" saved. Ln 20, Col 22

```

Figura 6.91. Descripción en lenguaje VHDL del componente puerta OR con salida en colector abierto y retardo de 10 ns

```

06v0_02*
-----
-- Fichero : D:\Ejemplos\Cap06\VBv99\06V0_02\06V0_02.vhd
-- Lib_trabajo: D:\Ejemplos\Cap06\VBv99\06V0_02\WORKLIB
-- Descripción estructural de una puerta XOR
-- utilizando puertas NOT y OR en colector abierto
-- con retardos de 10 ns
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Definición de la entidad y arquitectura del conjunto o circuito (netlist)

ENTITY circuito_xor IS
PORT (a,b: IN std_logic;
      S: OUT std_logic);
END circuito_xor;

ARCHITECTURE estructural OF circuito_xor IS

    COMPONENT or2 IS PORT(e1,e2:IN std_logic; y:OUT std_logic);END COMPONENT;
    COMPONENT inv IS PORT(e:IN std_logic; y:OUT std_logic);END COMPONENT;

    SIGNAL na,nb:std_logic;

BEGIN
    u0: inv PORT MAP (e=>b,y=>nb);
    u1: inv PORT MAP (e=>a,y=>na);
    u2: or2 PORT MAP (e1=>a,e2=>b,y=>S);
    u3: or2 PORT MAP (e1=>na,e2=>nb,y=>S);

END estructural;

-- configuración

CONFIGURATION estruc OF circuito_xor IS
    FOR estructural
        FOR ALL: inv USE ENTITY work.inv; END FOR;
        FOR ALL: or2 USE ENTITY work.or2; END FOR;
    END FOR;

END CONFIGURATION estruc;
Ln 5, Col 44

```

Figura 6.92. Descripción estructural en lenguaje VHDL de una puerta XOR utilizando componentes con asignación de niveles (colector abierto) y retardos

El contenido del fichero, en lenguaje VHDL, que define los estímulos de entrada que se utilizan para probar el comportamiento de la puerta XOR es el que se muestra en la Figura 6.93.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap06\VBv99\06V0\_02\06V0\_02.vpd**

En la Figura 6.94 se muestran los resultados obtenidos en el programa *VeriBest VB v99* una vez realizada la simulación de las descripciones en lenguaje VHDL definidas anteriormente.

En dicha figura se puede comprobar que la salida *S* corresponde a una función XOR. Debido a que el retardo asignado en todos los componentes es de 10 ns, durante los primeros 10 ns el nivel lógico es ‘U’, indefinido. Además, también se puede observar que en la salida *S* se producen retardos de 10 ns y 20 ns, según sea uno o dos el número de niveles presentes entre la entrada y la salida, como se puede deducir en el circuito de la Figura 6.89.

```
06a0_02
-- Fichero : D:\Ejemplos\Cap06\VBr99\06V0_02\06S0_02.vhd
-- Lib_trabajo: D:\Ejemplos\Cap06\VBr99\06V0_02\WORKLIB
--
--
-----
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY estmulos IS
END estmulos;

ARCHITECTURE xor_stim OF estmulos IS
    COMPONENT circuito_xor
        PORT (a,b : IN std_logic;
              S : OUT std_logic);
    END COMPONENT;
    SIGNAL a,b,S : std_logic;
BEGIN
    circuito_xor1: circuito_xor PORT MAP (a,b,S);
    estmulos: PROCESS
    BEGIN
        a <= '0';
        b <= '0';
        WAIT FOR 100 ns;
        a <= '1';
        WAIT FOR 100 ns;
        a <= '0';
        b <= '1';
        WAIT FOR 100 ns;
        a <= '1';
        WAIT FOR 100 ns;
    END PROCESS;
END xor_stim;
```

Figura 6.93. Descripción en lenguaje VHDL de los estímulos de prueba

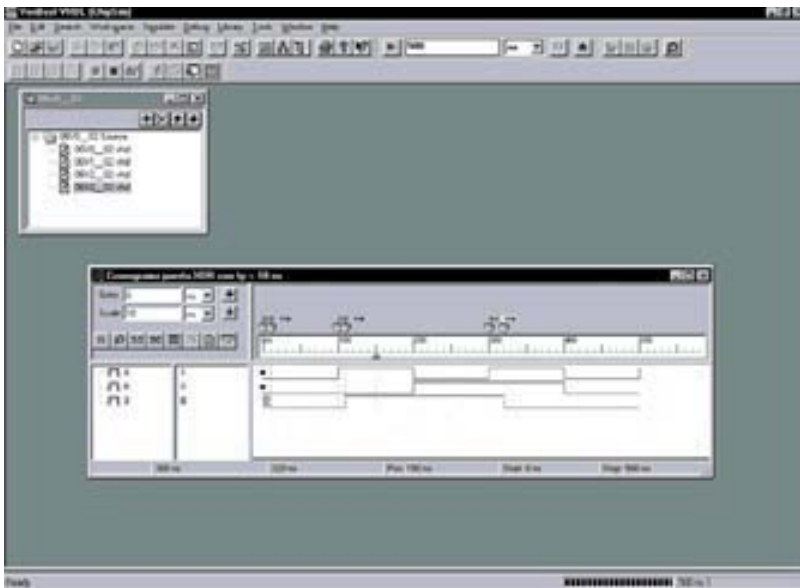
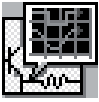


Figura 6.94. Simulación de una puerta XOR con puertas de salida en colector abierto

## PROBLEMA RESUELTO 6-21



Describir y simular la respuesta de un inversor con salida triestado y retardo de 10 ns. Utilizando para ello el módulo *Simulate Demo* del programa *OrCAD Demo v9*. Este fichero se debe abrir directamente desde el módulo indicado, para evitar posibles problemas durante su ejecución.

### Solución:

En la Figura 6.95 se muestra la descripción en lenguaje VHDL de una puerta inversora con salida triestado y retardo de 10 ns, donde *a* es la señal de entrada, *en* la señal de habilitación (*enable*) y *S* es la señal de salida.

```

06r0_04
-----
1:
2: -- Fichero : D:\Ejemplos\Cap06\OrCAD9\06R0_04\
3: -- Lib_trabajo: D:\Ejemplos\Cap06\VBv99\06R0_04\W
4: -- Descripción de puertas triestado
5: --
6: library ieee;
7: use ieee.std_logic_1164.all;
8: -- use ieee.numeric_std.all;
9:
10: entity tri_state is
11:   port(a, en : in std_logic;
12:        S : out std_logic);
13: end;
14:
15: architecture behavior of tri_state is
16: begin
17:
18:   process (a, en)
19:   begin
20:     if en = '1' then
21:       S <= not a after 10 ns;
22:     else
23:       S <= 'Z' after 10 ns;
24:     end if;
25:   end process;
26:
27: end;

```

Figura 6.95. Descripción en lenguaje VHDL de una puerta inversora con salida triestado y retardo de 10 ns

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap06\OrCAD\06R0\_04\06R0\_04.opj**



En la Figura 6.96 se muestran los resultados que se obtienen en *OrCAD Demo v9* al realizar la simulación de la descripción en lenguaje VHDL anterior.

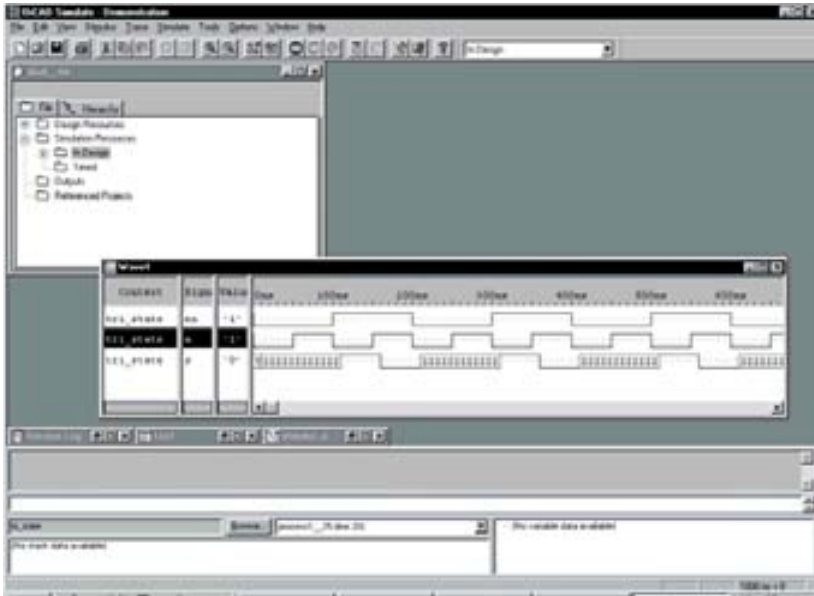


Figura 6.96. Simulación de una puerta inversora con salida triestado y retardo 10 ns

Obsérvese, en la Figura 6.96, que cuando la señal de habilitación *en* es inactiva (a nivel cero) la salida tiene un nivel de alta impedancia 'Z', cuya fuerza es prácticamente nula. Se puede comprobar que se produce un retardo de propagación de 10 ns.

**PROBLEMAS PROPUESTOS**

- 6-1) Calcular el *fan-out* de la familia *B* del Problema resuelto 6-1.
- 6-2) Calcular la disipación de potencia estática de las familias TTL estándar y TTL AS, alimentadas con una tensión  $V_{CC} = 5\text{ V}$  y cuyos consumos se muestran en la tabla siguiente.

<i>Familias Lógicas (VCC = 5 V)</i>		
	$I_{CCL}$	$I_{CCH}$
<i>TTL Estándar</i>	11 mA	4 mA
<i>TTL AS</i>	2,7 mA	3,2 mA

- 6-3) Estudiar la compatibilidad de dos familias lógicas, TTL estándar y CMOS avanzada (ACT), cuyas características se muestran en la siguiente tabla.

<i>Familia Lógica (<math>V_{CC} = 5\text{ V}</math>)</i>		
<i>Parámetro</i>	<i>TTL estándar</i>	<i>CMOS (ACT)</i>
$V_{oHmin}$	2,4 V	4,4 V
$V_{oLmax}$	0,4 V	0,1 V
$V_{iHmin}$	2 V	2,2 V
$V_{iLmax}$	0,8 V	0,8 V
$I_{oHmax}$	- 400 $\mu\text{A}$	-24 mA
$I_{oLmax}$	16 mA	24 mA
$I_{iHmax}$	40 $\mu\text{A}$	1 mA
$I_{iLmax}$	-1,6 mA	-1 mA

- 6-4) Utilizando el módulo *Simulate Demo* del programa *OrCAD Demo v9*, simular la respuesta de un inversor con salida triestado y retardo de 5 ns. Repetir la simulación con un retardo de 50 ns.
- 6-5) Utilizando el programa *VeriBest VB99.0* simular la respuesta de la salida *S* de un inversor con retardo de propagación de 5 ns que no responda a impulsos de entrada inferiores a 1 ns. Asimismo, considerar la salida *Y* de un inversor con retardo de 2 ns que complementa impulsos de cualquier duración.
- 6-6) Utilizando el programa *Electronics Workbench* obtener la curva característica de entrada de una puerta NAND CMOS-Estándar (serie 4000) actuando como inversor.

## CIRCUITOS ARITMÉTICOS

---

---

**Objetivos:**

- Comprender los principios básicos sobre los que se fundamenta la lógica aritmética.
- Conocer los distintos circuitos aritméticos, sus especificaciones, conexionado y aplicaciones típicas.
- Saber analizar y diseñar circuitos aritméticos.

**Contenido:** En este capítulo se exponen los distintos procedimientos para la realización de circuitos sumadores y restadores.

**Simulación:** Se utilizarán los programas de simulación *Electronics WorkBench 5.0* y *OrCAD Demo v9* para la realización de circuitos aritméticos, así como para la comprobación de ejercicios resueltos.

## 7.1 CIRCUITOS ARITMÉTICOS. INTRODUCCIÓN



Los circuitos aritméticos posibilitan las operaciones de cálculo en la tecnología digital y representan la base para el desarrollo de los sistemas computacionales.

El desarrollo de los circuitos aritméticos se realiza a partir de la combinación adecuada de puertas lógicas, lo que implica una gran rapidez en el cálculo.

El dispositivo lógico básico es el sumador, a partir de éste y mediante la representación de números negativos, se obtiene el dispositivo restador. Las operaciones más complejas de multiplicación y división se obtienen a partir de algoritmos basados en sumas y restas sucesivas.

La combinación de todos estos dispositivos da lugar a la Unidad Aritmético-Lógica, **ALU**, núcleo esencial en la arquitectura de un microprocesador.

## 7.2 SUMADORES BINARIOS

En esta apartado se estudia el semisumador, el sumador completo, el sumador paralelo con acarreo serie, el sumador paralelo con acarreo paralelo, y el sumador paralelo con acarreo mixto.

### 7.2.1 Semisumador

Se denomina **semisumador** al circuito combinacional capaz de realizar la suma aritmética binaria de dos únicos bits  $A$  y  $B$ , proporcionando a su salida un bit resultado de suma  $S$  y un bit de acarreo  $C$ . En la Tabla 7.1 se muestra la función de transferencia de este tipo de circuitos.

Tabla 7.1. Función de salida de un semisumador

Entradas	Salidas
$B A$	$C S$
0 0	0 0
0 1	0 1
1 0	0 1
1 1	1 0

A partir de la Tabla 7.1 se obtienen las funciones lógicas  $S$  y  $C$  que se muestran en la expresión [7.1].

$$S = \bar{A}B + A\bar{B} = A \oplus B \quad [7.1]$$

$$C = A \cdot B$$

En la Figura 7.1 se muestra el símbolo del *semisumador* y su realización mediante puertas lógicas básicas.

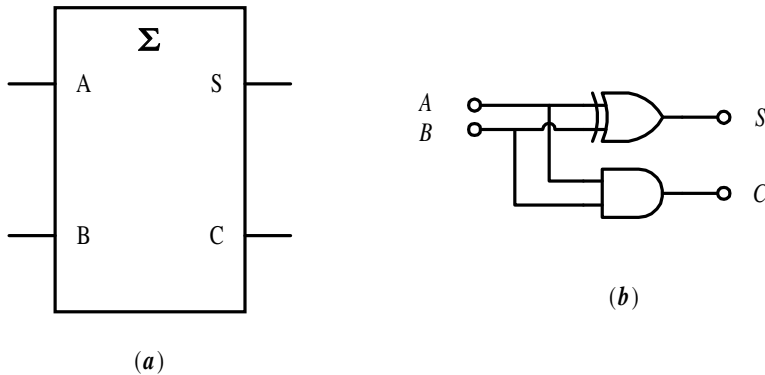


Figura 7.1. Símbolo del semisumador (a) y circuito lógico (b)

### PROBLEMA RESUELTO 7-1



Simular el funcionamiento del semisumador que se ha representado en la Figura 7.1, utilizando para ello el programa *Electronics Workbench*.

#### Solución:

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap07\Ewb5\07W0\_01.ewb**

Para realizar la comprobación de la función de salida descrita en la Tabla 7.1, se debe modificar la posición de los conmutadores A y B. La posición de estos conmutadores se realiza utilizando las teclas activas [1] y [2], que se han asociado a cada uno de ellos respectivamente.

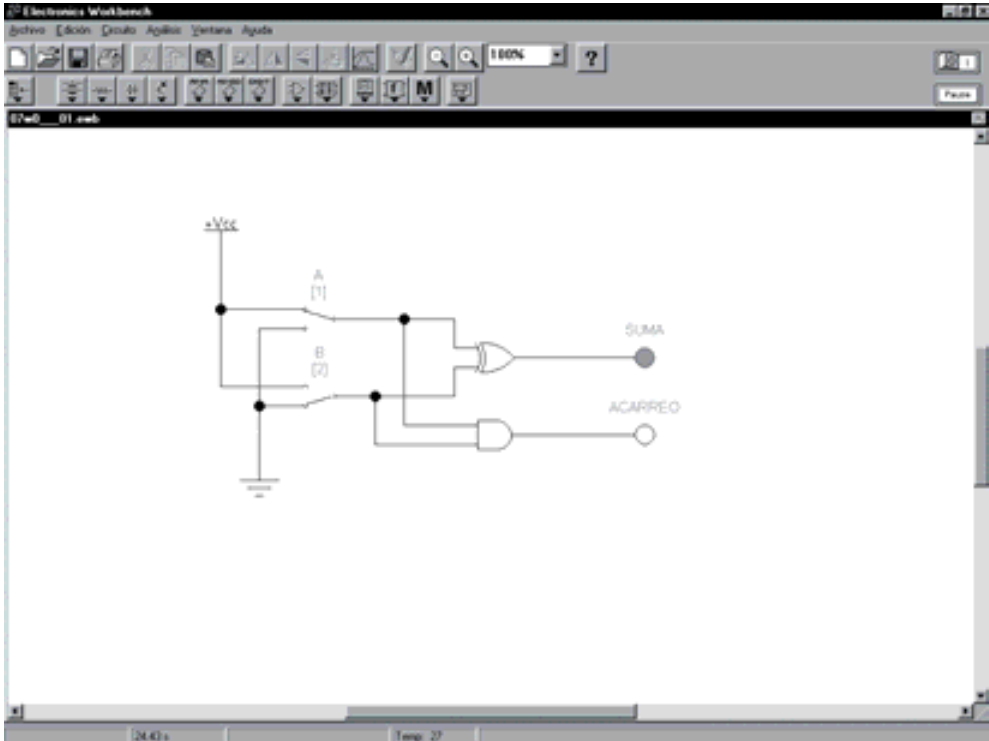


Figura 7.2. Circuito para simular el funcionamiento de un semisumador

## 7.2.2 Sumador completo

En la práctica, los circuitos sumadores manejan informaciones binarias con una longitud de palabra superior a un bit, por lo que es preciso ampliar la funcionalidad del semisumador a un dispositivo capaz de realizar sumas binarias de  $n$  bits.

El proceso a seguir en la realización de una suma binaria es muy similar al empleado habitualmente en la aritmética decimal. Desde un punto de vista genérico, sean  $A$  y  $B$  dos informaciones binarias de  $n$  bits:

$$A = A_{n-1} A_{n-2} \dots A_2 A_1 A_0 \quad B = B_{n-1} B_{n-2} \dots B_2 B_1 B_0 \quad [7.2]$$

La suma de  $A + B$  será un proceso de  $n$  **sumas parciales**, comenzando por los dos *bits* de menor peso. El resultado de esta operación  $A_0 + B_0$  será un bit de suma  $S_0$  y un *bit* de acarreo  $C_0$  que se propagará como elemento integrante de la suma, en los siguientes dos bits, en orden ascendente de peso  $A_1$  y  $B_1$ . A partir de este punto el resultado de todas las operaciones  $A_i + B_i + C_{i-1}$  incorporan un bit de suma  $S_i$  y un bit

de acarreo  $C_i$  que se propaga como elemento integrante de la suma en los dos bits siguientes y así sucesivamente.

Por tanto, el resultado final de la suma de  $A + B$  será:

$$S = C_{n-1} S_{n-1} S_{n-2} \dots S_2 S_1 S_0 \quad [7.3]$$

Como se ha podido apreciar, la operación básica realizada en el proceso de la suma ha sido:  $A_i + B_i + C_{i-1}$  dando como resultado  $S_i$  y  $C_i$ , con la única excepción de la primera operación realizada que no considera el acarreo inicial. Al circuito que desarrolla esta funcionalidad se le denomina **sumador completo**, y su función de salida se presenta en la Tabla 7.2.

Tabla 7.2. Función de salida de un sumador completo

Entradas	Salidas
$C_{i-1} B_i A_i$	$C_i S_i$
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	0 1
1 0 1	1 0
1 1 0	1 0
1 1 1	1 1

A partir de la Tabla 7.2 se deduce la ecuación lógica a la que responde el sumador completo:

$$S_i = \overline{A}BC_{i-1} + A\overline{B}C_{i-1} + \overline{A}B\overline{C}_{i-1} + ABC_{i-1} = A \oplus B \oplus C_{i-1} \quad [7.4]$$

$$C_i = AB + BC_{i-1} + AC_{i-1}$$

En la Figura 7.3 se muestra el circuito **sumador completo** realizado a partir de dos semisumadores. La función resultante de suma  $S_i$  resulta de la aplicación directa de los dos semisumadores  $A_i + B_i + C_{i-1}$  y la función del acarreo de salida  $C_i$  se obtiene a partir de la tabla de verdad de los acarreos parciales  $C_{s1}$  y  $C_{s2}$  (Tabla 7.3).

Tabla 7.3. Función de salida de acarrees parciales

Entradas		Salidas	Acarrees parciales		
$C_{in}$	$B A$		$C_{out}$	$C_{s1}$	$C_{s2}$
0	0 0	0	0	0	0
0	0 1	1	0	0	0
0	1 0	1	0	0	0
0	1 1	0	1	1	0
1	0 0	1	0	0	0
1	0 1	0	1	0	1
1	1 0	0	1	0	1
1	1 1	1	1	1	0

$$S = A + B + C_{in}$$

[7.5]

$$C_{out} = C_{s1} + C_{s2} = C_{s1} \oplus C_{s2}$$

Obsérvese que el acarreo de salida se puede definir como una función OR u OR-EXCLUSIVA aprovechando que la combinación de variables  $C_{s1} = C_{s2} = 1$  es imposible y por tanto indefinida.

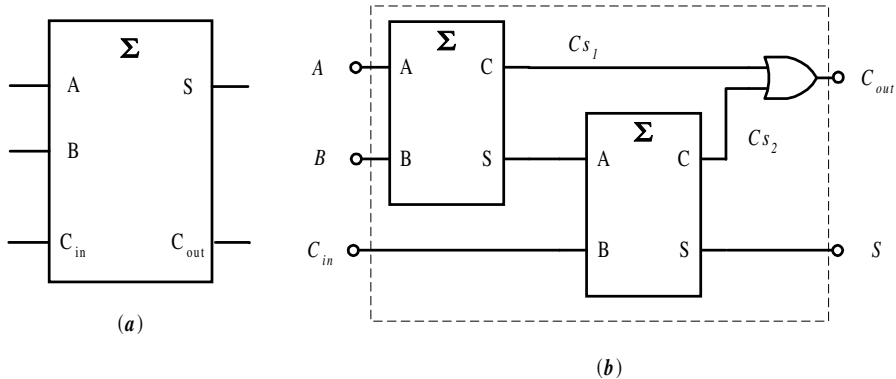


Figura 7.3. (a) Símbolo del sumador completo y (b) Circuito lógico



## PROBLEMA RESUELTO 7-2



Simular el funcionamiento del circuito sumador completo representado anteriormente en la Figura 7.3, utilizando para ello el programa *Electronics Workbench*.

### Solución:

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap07\Ewb5\07W0\_\_02.ewb**

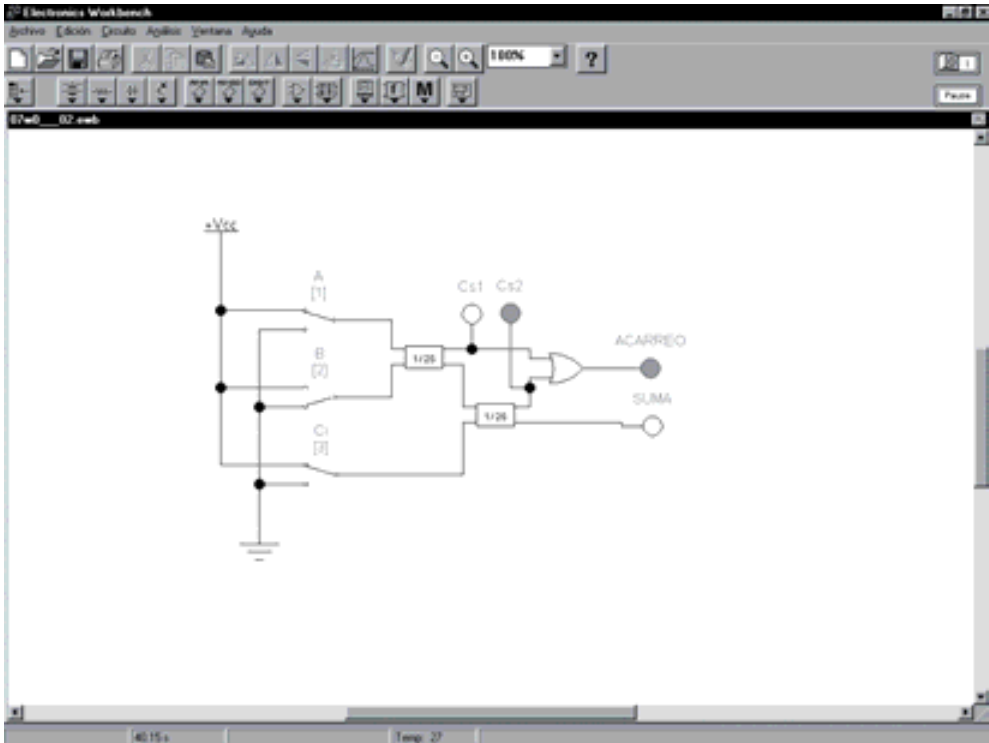


Figura 7.4. Circuito para simular el funcionamiento de un sumador completo

Mediante el empleo de los conmutadores A, B y C, que se encuentran asociados a las teclas activas [1], [2] y [3], respectivamente, se puede realizar la comprobación de las funciones de salida descritas en la Tabla 7.2 y en la Tabla 7.3.

### PROBLEMA RESUELTO 7-3



Simular, describiendo en lenguaje VHDL, el funcionamiento del sumador completo representado en la Figura 7.3, utilizando para ello el módulo *Simulate Demo* del programa *OrCAD Demo v9*. Este fichero se debe abrir directamente desde el módulo indicado para evitar posibles problemas durante su ejecución.

#### Solución:

Mediante el lenguaje VHDL se hace una descripción funcional de un sumador completo y se simula su cronograma y tabla de verdad.

La ruta y el nombre del fichero de proyecto que contiene esta descripción en lenguaje VHDL es la que se indica a continuación:

**D:\Ejemplos\Cap07\OrCAD9\07R0\_\_01\07R0\_\_01.opj**

Dentro del proyecto existen varias ventanas que se describen a continuación. Una de estas ventanas contiene el fichero de texto en el que se describe el código VHDL del sumador completo, y que se ha representado en la Figura 7.5.

```

-----
-- Fichero : D:\Ejemplos\Cap07\OrCAD9\07R0__01\07R0__01.vhd
--           Modelo de sumador completo mediante flujo de datos
--
--
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Sum_completo IS
    PORT(
        a,b : in std_logic;
          Cin : in std_logic;
          S : out std_logic;
          Cout : out std_logic);
END Sum_completo;

-- Descripción flujo de datos

ARCHITECTURE flujo_datos OF Sum_completo IS
    constant tp: time := 10 ns;          -- retardo de propagación
    signal aux:std_logic;

BEGIN
    aux <= a xor b AFTER tp;
    S <= Cin xor aux AFTER tp;
    Cout <= (a and b) or (aux and cin) AFTER 2*tp;    -- 2 niveles

END flujo_datos;

```

Figura 7.5. Descripción funcional mediante VHDL de un sumador completo

En la Figura 7.6 se muestran las ventanas que se generan al abrir el proyecto, en las que se relacionan:

- ◆ los documentos que integran el proyecto (ventana 07r0\_\_01),
- ◆ el listado de niveles lógicos que toman las señales en función del tiempo (ventana *list1*, equivalente a la tabla de verdad con retardos),
- ◆ la descripción VHDL anteriormente indicada y,
- ◆ el cronograma de las señales de entrada y salida del sumador completo (ventana *Wave1*).

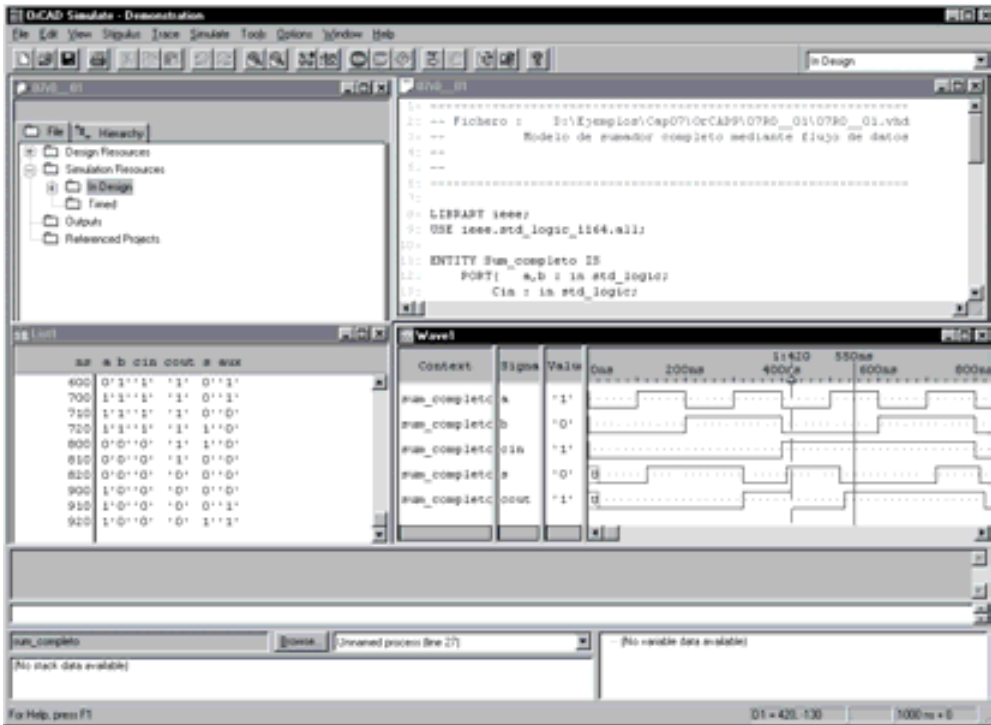


Figura 7.6. Circuito de simulación de un sumador completo

Situando el cursor sobre cada uno de los puntos de interés, por ejemplo en cada uno de los sucesivos valores que van tomando las señales, se puede verificar la tabla de verdad del circuito. Se debe tener en cuenta y se aconseja comprobar en el cronograma los retardos de propagación  $t_p = 10$  ns por cada nivel lógico, definidos en la descripción VHDL, es decir, el resultado del sumador es correcto tras un retardo de 20 ns.

### 7.2.3 Sumador paralelo con acarreo serie

Un sumador paralelo de dos informaciones binarias  $A+B$  de  $n$  bits necesita realizar  $n$  sumas parciales, empleando para ello  $n$  sumadores completos.

Cuando las palabras  $A$ ,  $B$  y  $C_{-1}$  son presentadas en la entrada del sumador, transcurrido el tiempo de operación, aparece de forma simultánea en la palabra de salida  $S$  la información que se obtiene como resultado de la suma y los acarreos de salida  $C$ , teniendo en cuenta que sólo los bits  $S_0$  y  $C_0$  son estables. A partir de este momento, la información correspondiente al acarreo de menor peso  $C_0$  se propaga en modo serie por todos los sumadores en orden creciente de peso, con la posibilidad de alterar los resultados obtenidos en las sumas parciales hasta que finaliza la propagación estabilizando el bit  $C_{n-1}$ .

Este circuito es muy simple e intuitivo pero presenta el serio inconveniente de tener que esperar un tiempo igual a  $n$  **tiempos de propagación** antes de obtener un resultado estable y fiable en la suma, por lo que no es aconsejable utilizarlo en sistemas que precisen una cierta rapidez, y que necesiten el uso de palabras grandes de información.

Se debe observar en el circuito de la Figura 7.7 que al disponer de un bit de acarreo inicial  $C_{-1}$ , es posible realizar dos operaciones en función del estado de dicho bit:

Si  $C_{-1} = 0$ , el resultado será:  $S = A + B$

Si  $C_{-1} = 1$ , el resultado será:  $S = A + B + 1$

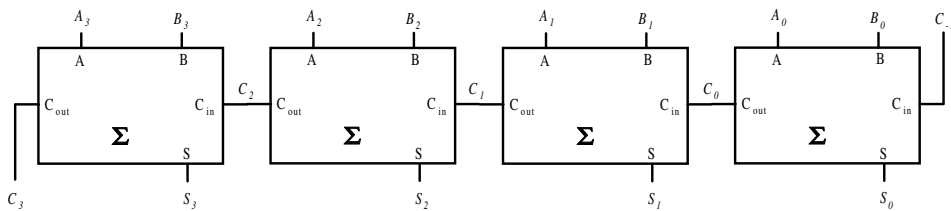


Figura 7.7. Circuito lógico de un sumador paralelo con acarreo serie de 4 bits

En la Figura 7.8 se muestra el símbolo de un circuito sumador paralelo con acarreo serie de cuatro bits.

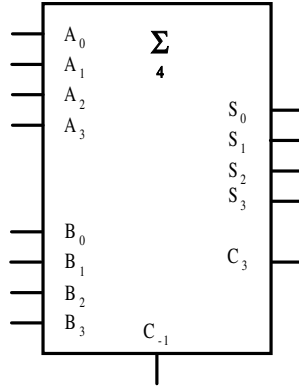


Figura 7.8. Símbolo de un sumador paralelo de 4 bits

A partir del símbolo de la Figura 7.8 representado como un bloque funcional completo, la asociación necesaria para crear sumadores de orden superior se muestra en la Figura 7.9.

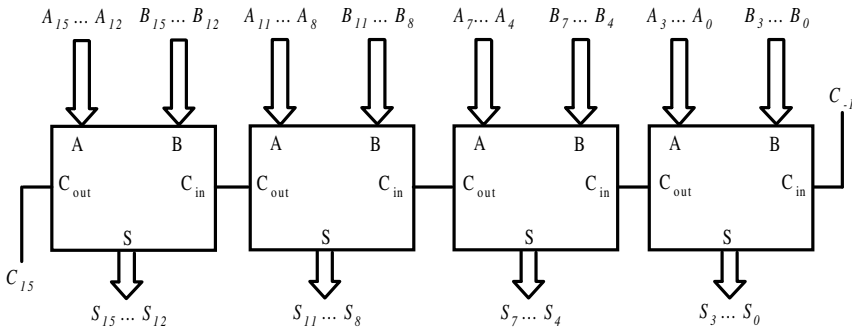


Figura 7.9. Circuito lógico de un sumador de 16 bits con acarreo serie

### PROBLEMA RESUELTO 7-4



Simular el funcionamiento de un sumador paralelo con acarreo serie de dos bits, similar al representado en la Figura 7.7 utilizando para ello el programa *Electronics Workbench*.

#### Solución:

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap07\Ewb5\07W0\_03.ewb**

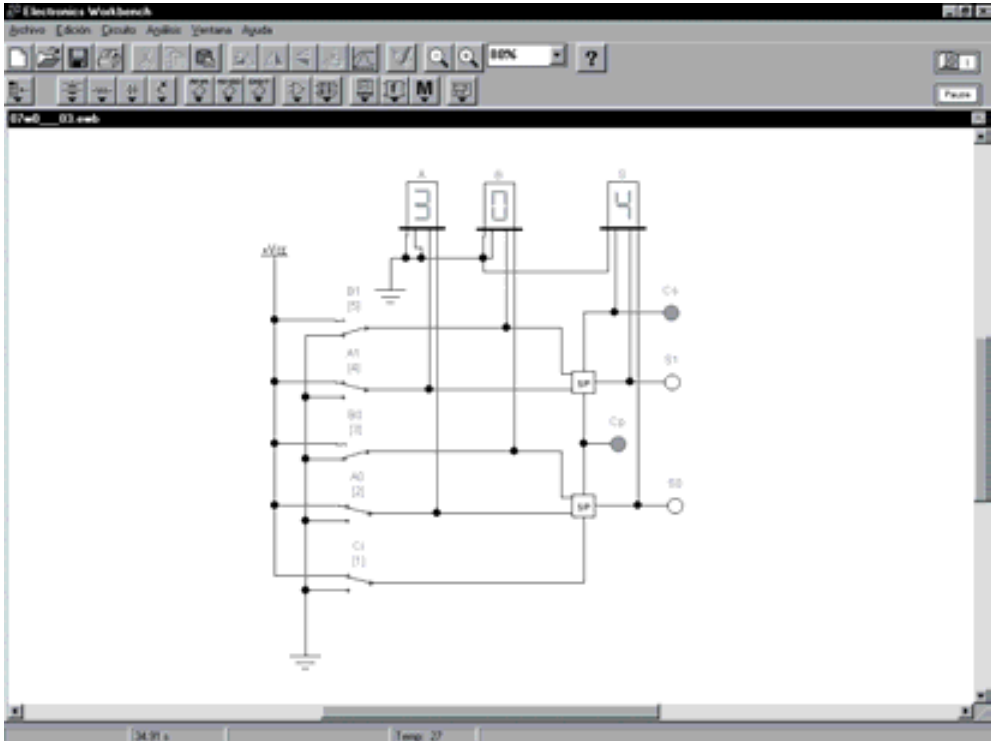


Figura 7.10. Circuito para simular el funcionamiento de un sumador paralelo de dos bits

Mediante el empleo de los conmutadores A0, B0, ..., A1, B1 y Ci, que se encuentran asociados a la teclas activas [1], [2], [3], [4] y [5] respectivamente, realizar la comprobación de la función suma, observando la evolución del acarreo parcial Cp.

### PROBLEMA RESUELTO 7-5



Simular el funcionamiento del sumador paralelo con acarreo serie de cuatro bits que se ha representado anteriormente en la Figura 7.7, utilizando para ello el programa de simulación *Electronics Workbench*.

#### Solución:

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap07\Ewb5\07W0\_\_04.ewb**

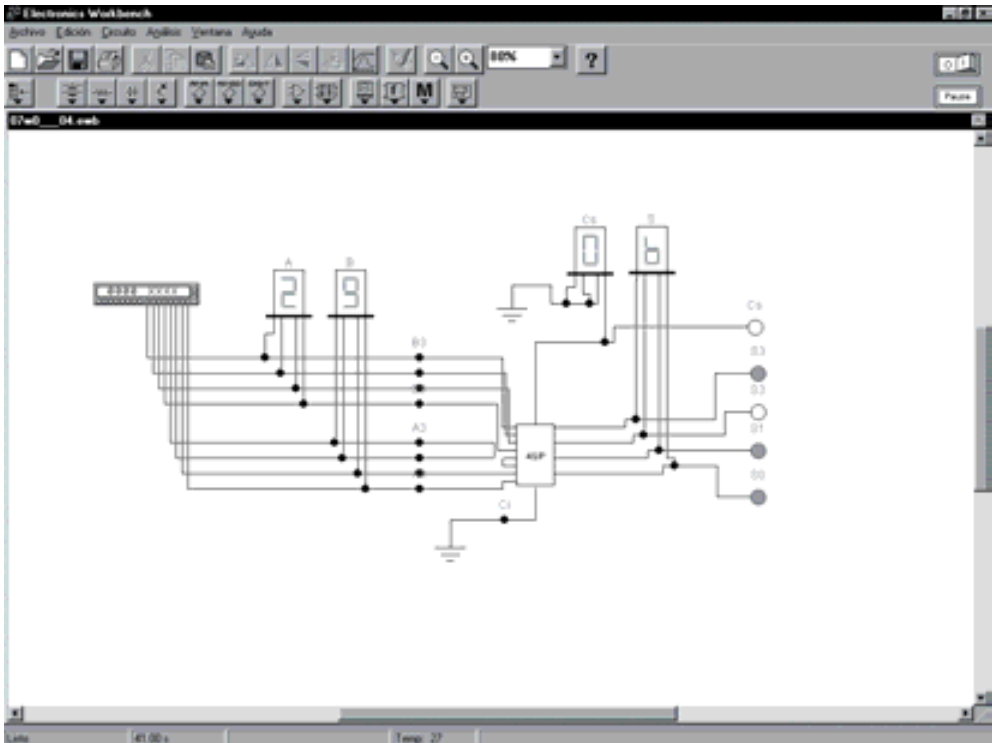


Figura 7.11. Circuito para simular el funcionamiento de un sumador paralelo de cuatro bits

Mediante el empleo del generador de palabras realizar la comprobación de la función suma sin acarreo inicial. Observar que la representación en los indicadores de siete segmentos se realiza en notación hexadecimal.

### PROBLEMA RESUELTO 7-6



Simular, describiendo en VHDL, el funcionamiento del sumador paralelo con acarreo serie de ocho bits, utilizando para ello el módulo *Simulate Demo* del programa *OrCAD Demo v9*. Véase el sumador de cuatro bits representado anteriormente en la Figura 7.7. Este fichero se debe abrir directamente desde el módulo indicado para evitar posibles problemas durante su ejecución.

### Solución:

Mediante el lenguaje VHDL se realiza una descripción funcional de un sumador paralelo con acarreo serie de ocho bits y se realiza la simulación de algunas operaciones, representadas mediante cronograma.

La ruta y el nombre del fichero de proyecto que contiene esta descripción VHDL es la que se indica a continuación:

**D:\Ejemplos\Cap07\OrCAD9\07R0\_02\07R0\_02.opj**

Dentro del proyecto existen varias ventanas que se describen a continuación. Una de estas ventanas contiene el fichero de texto en el que se describe el código VHDL del sumador paralelo con acarreo serie de ocho bits, y que se muestra de forma completa en la Figura 7.12.

El componente Sum\_completo, incluido en este proyecto, se ha descrito en el Problema resuelto 7-3, (Figura 7.5. Descripción funcional mediante VHDL de un sumador completo).

```

-----
-- Fichero : D:\Ejemplos\Cap07\OrCAD9\07R0_02\07R0_02.vhd
-- Modelo estructural de sumador de nbits con acarreo serie
-----
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Sum_serie IS
    Generic (nbits:integer := 8); -- nbits es el número de bits
                                   -- del sumador

    PORT( ss_A, ss_B : in std_logic_vector(nbts-1 downto 0);
          ss_Cin : in std_logic;
          ss_S : out std_logic_vector(nbts-1 downto 0);
          ss_Cout : out std_logic);
END Sum_serie;

-- Descripción estructural

ARCHITECTURE estructural OF Sum_serie IS
    component Sum_completo
    PORT( a,b : in std_logic;
          Cin : in std_logic;
          S : out std_logic;
          Cout : out std_logic);
    END component;
    signal aux_acarreo: std_logic_vector(nbts-1 downto -1);
BEGIN
    aux_acarreo(-1) <= ss_Cin;

    sum_serie: for i in 0 to nbts-1 generate
        elemento_sumador: sum_completo
        port map( a => ss_A(i),
                 b => ss_B(i),
                 Cin => aux_acarreo(i-1),
                 S => ss_S(i),
                 Cout => aux_acarreo(i));
    End generate sum_serie;
    ss_Cout <= aux_acarreo(nbts-1);
END estructural;

```

*Figura 7.12. Descripción funcional mediante VHDL de un sumador paralelo con acarreo serie de ocho bits*



En la Figura 7.13 se muestran las ventanas que se generan al abrir el proyecto y que relacionan:

- ◆ los documentos que integran el proyecto (ventana 07r0\_\_02),
- ◆ el cronograma de las señales de entrada y salida del sumador paralelo con acarreo serie (ventana Wave1),
- ◆ la descripción VHDL del sumador paralelo con acarreo serie y,
- ◆ la descripción VHDL de componente Sum\_completo (ventana 07r0\_\_01), en la que se define un retardo de propagación de 10 ns.

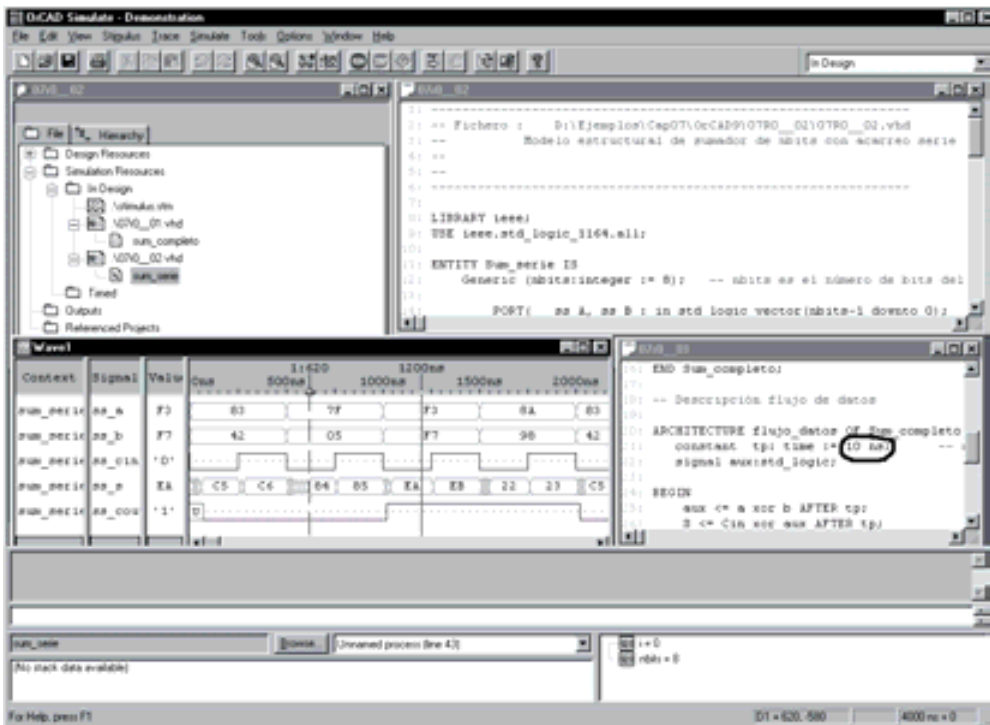


FIGURA 7.13. Circuito de simulación de un sumador paralelo con acarreo serie de ocho bits con retardos

Situando el cursor sobre cada uno de los puntos de interés, por ejemplo en cada uno de los sucesivos valores que van tomando las señales, se obtienen los diferentes resultados de las sumas. Se puede comprobar en el cronograma que la velocidad de respuesta de los retardos de propagación de las señales de salida son debidas al retardo de cada sumador completo ( $2 \cdot t_p$ ) por el número de ellos que componen el sumador paralelo con acarreo serie ( $nbits$ ), definidos en la descripción VHDL.

Se deja al lector que modifique alguna característica del sumador paralelo con acarreo serie, como por ejemplo: su número de bits ( $nbits$ ), los retardos de los sumadores completos, etc., comprobando cómo se ve afectada la velocidad de respuesta en las señales de salida. Por ejemplo si se define un retardo nulo en el sumador completo,  $tp = 0$  ns, se obtiene el cronograma de la Figura 7.14, correspondiente a un sumador paralelo con acarreo serie ideal.

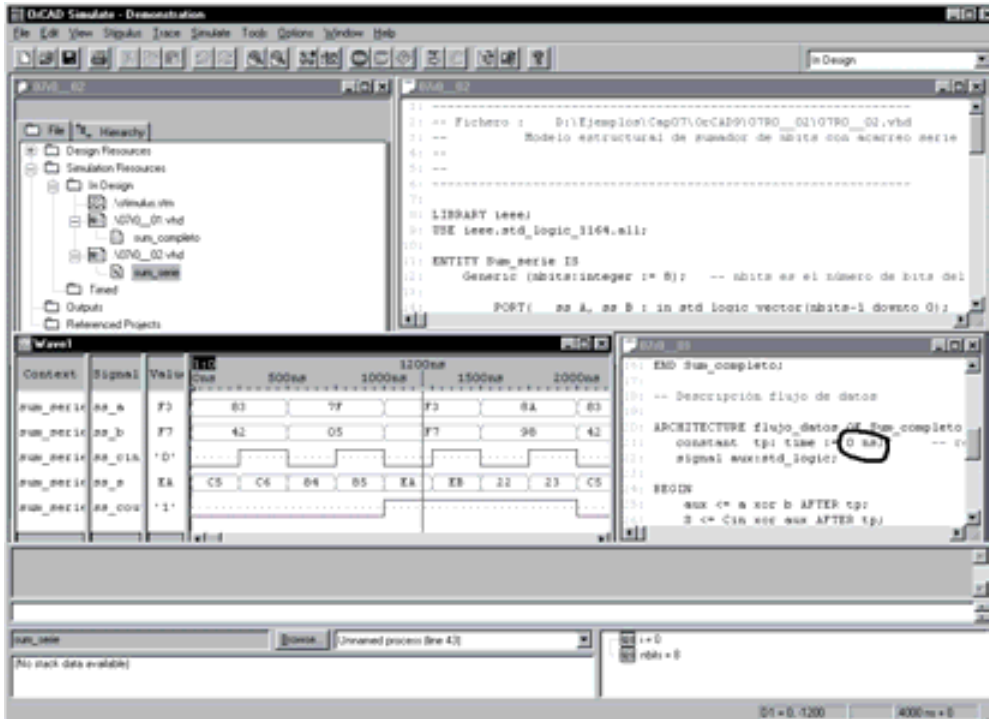


Figura 7.14. Circuito de simulación de un sumador paralelo con acarreo serie de ocho bits sin retardos

## 7.2.4 Sumador paralelo con acarreo paralelo

Este tipo de sumador, también llamado **sumador paralelo con acarreo anticipado**, realiza la suma de dos informaciones  $A$  y  $B$  de  $n$  bits aumentando la velocidad de proceso respecto del sumador paralelo con acarreo serie. Este circuito permite la generación simultánea de todos los bits de acarreo en el mismo proceso de cálculo de las sumas parciales.

Cuando se suman dos informaciones  $A_i$  y  $B_i$  se obtendrá acarreo por alguna de las dos posibles circunstancias:

- a) Porque se ha generado un acarreo en la propia etapa del sumador. En este caso se denomina **acarreo generado**  $G_i$  y sólo aparecerá si:  $A_i = B_i = 1$ .

$$G_i = A_i \cdot B_i \quad [7.6]$$

- b) Porque el acarreo proviene de la etapa anterior  $C_{i-1}$  y se propaga nuevamente al llegar a esta etapa. En este caso se denomina **acarreo propagado**  $P_i$  y sólo aparecerá si:  $A_i \neq 0$  o  $B_i \neq 0$ .

$$P_i = A_i + B_i \quad [7.7]$$

Por tanto, el acarreo producido en la etapa  $i$ -ésima  $C_i$  será porque **se genera** o porque **se propaga** y se expresará, en función de [7.6] y [7.7], según la ecuación [7.8].

$$C_i = G_i + P_i C_{i-1} = A_i \cdot B_i + (A_i + B_i) \cdot C_{i-1} \quad [7.8]$$

Obsérvese que la expresión [7.8] es idéntica a la obtenida anteriormente en [7.4].

Desarrollando la expresión [7.8] para un sumador de cuatro bits ( $i = 0, \dots, 3$ ) se tiene que:

$$C_0 = G_0 + P_0 (C_{-1})$$

$$C_1 = G_1 + P_1 (C_0) = G_1 + P_1 G_0 + P_1 P_0 (C_{-1}) \quad [7.9]$$

$$C_2 = G_2 + P_2 (C_1) = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 (C_{-1})$$

$$C_3 = G_3 + P_3 (C_2) = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 (C_{-1})$$

A partir de la expresión [7.9] se puede observar que el cálculo de  $C_i$  puede realizarse simultáneamente, pues sólo depende de las variables que se aplican a la entrada del sumador.

La Figura 7.15 representa el **circuito lógico simplificado de un sumador con generación de  $P$  y  $G$**  para el cálculo del acarreo anticipado. Obsérvese que para simplificar el circuito el propagador de acarreos  $P$  se ha obtenido mediante una puerta XOR existente, en vez de incorporar una OR. Esto es válido considerando que, según la expresión [7.8], cuando existe generación de acarreo  $G$  la condición de propagación es indiferente,  $P = 'X'$ .

El circuito integrado 74182, representado en la Figura 7.16, es un generador de acarreo anticipado para sumadores completos de cuatro bits, fácilmente ampliable a órdenes de bits superiores, manteniendo la generación anticipada del acarreo.

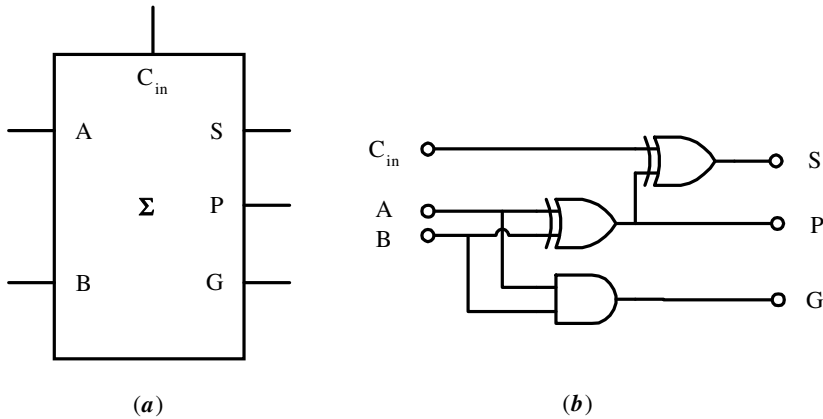


Figura 7.15. (a) Símbolo y (b) circuito lógico de un sumador con generación de  $P$  y  $G$

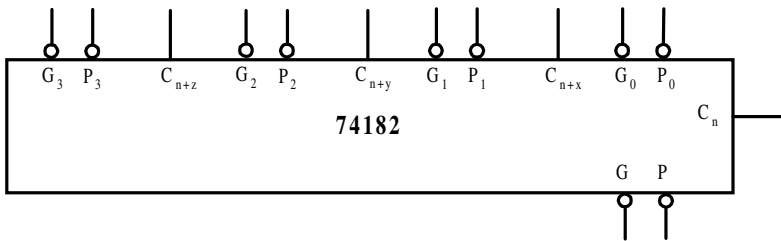


Figura 7.16. Símbolo del IC 74182

Las salidas de este circuito integrado presentan las funciones que se muestran en la expresión [7.10]:

$$\begin{aligned}
 C_{n+x} &= G_0 + P_0 (C_n) \\
 C_{n+y} &= G_1 + P_1 G_0 + P_1 P_0 (C_n) \\
 C_{n+z} &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 (C_n) \\
 \overline{G} &= \overline{G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 (C_n)} \\
 \overline{P} &= \overline{P_3 P_2 P_1 P_0}
 \end{aligned}
 \tag{7.10}$$

En la Figura 7.17 se muestra un sumador de cuatro bits con acarreo anticipado. Debe observarse que el acarreo inicial  $C_{-1}$  se aplica tanto al sumador de menor peso como a la entrada  $C_n$  del circuito generador de acarreo anticipado y el acarreo final  $C_3$  puede obtenerse mediante la operación de suma lógica  $G_3 + P_3$ .

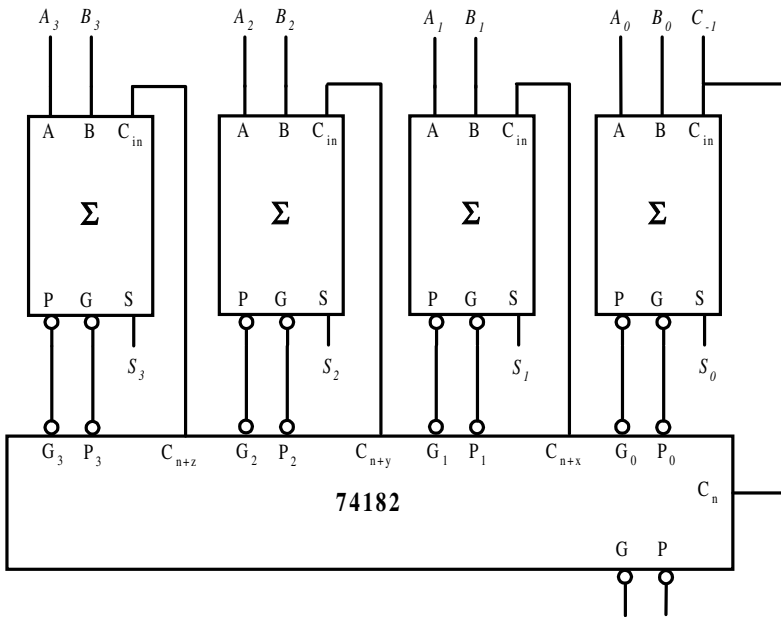


Figura 7.17. Circuito lógico de un sumador de 4 bits con acarreo anticipado

En la Figura 7.18 se muestra la asociación necesaria para crear sumadores de orden superior, considerando el circuito de la Figura 7.17 como un bloque funcional completo.

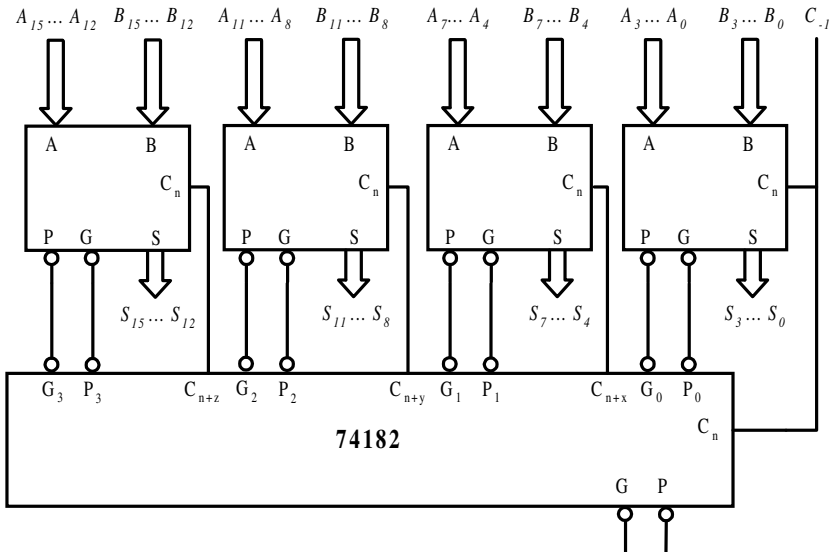


Figura 7.18. Circuito lógico de un sumador de 16 bits con acarreo anticipado

## PROBLEMA RESUELTO 7-7



Simular el funcionamiento del sumador completo con generación de acarreo paralelo representado en la Figura 7.15, utilizando para ello el programa de simulación *Electronics Workbench*.

### Solución:

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap07\Ewb5\07W0\_\_05.ewb**

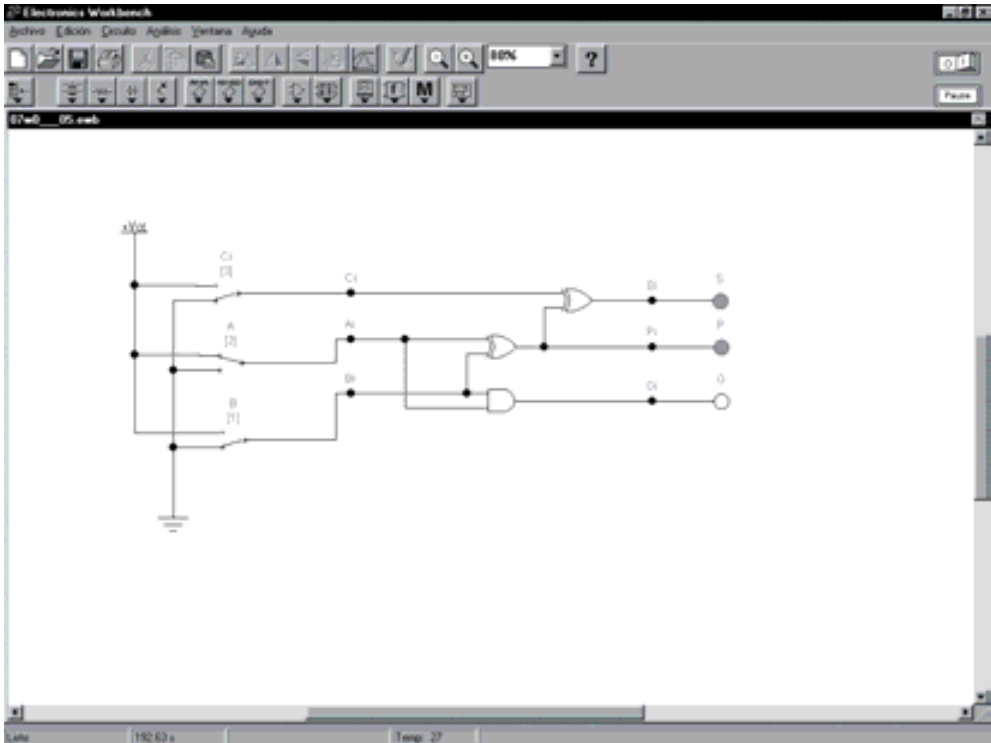


Figura 7.19. Circuito para simular el funcionamiento de un sumador completo con generación de acarreo anticipado

Mediante el empleo de los conmutadores A, B y C con las teclas activas 1, 2 y 3 realizar la comprobación de las funciones de suma (S), acarreo propagado (P) y acarreo generado (G), descritas en las expresiones [7.6] y [7.7].

## PROBLEMA RESUELTO 7-8



Simular, describiendo en VHDL, el funcionamiento del sumador paralelo con acarreo anticipado de ocho bits, utilizando para ello el módulo *Simulate Demo* del programa de simulación *OrCAD Demo v9*. Este fichero se debe abrir directamente desde el módulo indicado para evitar posibles problemas durante su ejecución.

### Solución:

Mediante el lenguaje VHDL se realiza una descripción funcional de un sumador paralelo con acarreo anticipado de ocho bits y se simulan algunas operaciones mediante cronograma.

La ruta y el nombre del fichero de proyecto que contiene esta descripción VHDL es la que se indica a continuación:

**D:\Ejemplos\Cap07\OrCAD9\07R0\_\_03\07R0\_\_03.opj**

Dentro del proyecto existen varias ventanas que se describen a continuación. Una de estas ventanas contiene el fichero de texto en el que se describe el código VHDL del sumador paralelo con acarreo anticipado de ocho bits, y que se muestra de forma completa en la Figura 7.20.

El componente `Sum_completo`, incluido en este proyecto, se ha descrito en el Problema resuelto 7-3 (Figura 7.5. Descripción funcional mediante VHDL de un sumador completo).

En la Figura 7.21 se muestran las ventanas que componen el proyecto, en las que se relacionan:

- ◆ los documentos que integran el proyecto (ventana `07r0__03`),
- ◆ el cronograma de las señales de entrada y salida del sumador paralelo con acarreo anticipado (ventana *Wave1*),
- ◆ la descripción VHDL del sumador paralelo con acarreo anticipado mostrada en la Figura 7.20 (ventana `07r0__03`) y,
- ◆ la descripción VHDL del `Sum_completo` (ventana `07r0__01`).

Colocando el cursor sobre cada uno de los puntos de interés, por ejemplo en cada uno de los sucesivos valores que van tomando las señales, se pueden obtener los diferentes resultados de las sumas. En los cronogramas de la Figura 7.21 y de la Figura 7.13, se puede comprobar la principal ventaja del sumador con acarreo anticipado respecto al de acarreo serie, comparando los retardos que se producen. El sumador con acarreo anticipado asegura un retardo máximo de  $5 \cdot t_p$  independiente del número de bits que éste tenga.

Se deja como ejercicio para el lector, la realización de modificaciones en las distintas características del sumador paralelo con acarreo anticipado, como por

ejemplo: su número de bits (*nbits*), los retardos de los sumadores completos, etc., comprobando cómo se ve afectada la velocidad de respuesta.

```

-----
-- Fichero : D:\Ejemplos\Cap07\OrCAD9\07R0_03\07R0_03.vhd
-- Modelo estructural de sumador de nbits con acarreo anticipado
-----
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Sum_anticipado IS
    Generic (nbits:integer := 8); -- nbits número de bits sumador

    PORT( sa_A, sa_B : in std_logic_vector(nbits-1 downto 0);
          sa_Cin : in std_logic;
          sa_S : out std_logic_vector(nbits-1 downto 0);
          sa_Cout : out std_logic);
END Sum_anticipado;

-- Descripción estructural

ARCHITECTURE estructural OF Sum_anticipado IS
    constant tp: time := 10 ns; -- retardo de propagación
    signal C: std_logic_vector(nbits-1 downto -1);
    signal G, P:std_logic_vector(nbits-1 downto 0);

    component Sum_completo
    PORT( a,b : in std_logic;
          Cin : in std_logic;
          S : out std_logic;
          Cout : out std_logic);
    END component;

BEGIN
    C(-1) <= sa_Cin;
    G <= sa_A and sa_B after tp;
    P <= sa_A or sa_B after tp;

    -- Generación de acarreos anticipados

    acarreo:for j in 0 to nbits-1 generate
        C(j) <= G(j) or (P(j) and C(j-1)); -- Expresión del acarreo
    end generate acarreo;
    sa_Cout <= C(nbits-1) after tp;

    sum_ant: for i in 0 to nbits-1 generate
        elemento_sumador:sum_completo
        port map( a => sa_A(i),
                 b => sa_B(i),
                 Cin => C(i-1),
                 S => sa_S(i),
                 Cout => open);
    End generate sum_ant;

END estructural;

```

Figura 7.20. Descripción funcional mediante VHDL de un sumador paralelo con acarreo anticipado de ocho bits



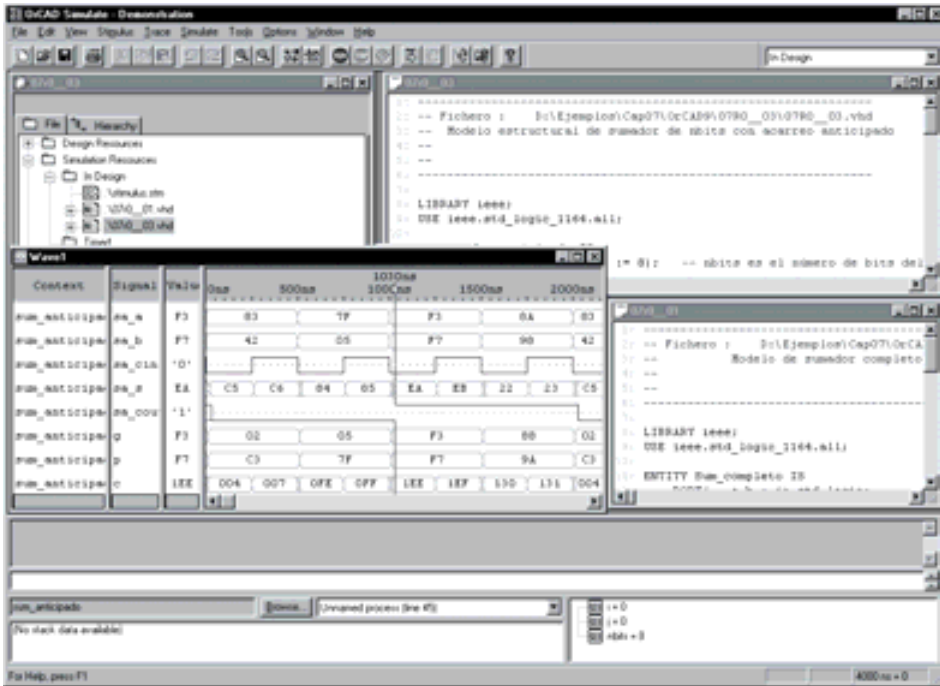


Figura 7.21. Circuito de simulación de un sumador paralelo con acarreo anticipado

### 7.2.5 Sumador paralelo con acarreo mixto

También es posible realizar un acoplamiento mixto, donde la generación de acarreo en cada sumador se realiza en paralelo y la propagación entre sumadores en serie. El IC 7483 es un sumador de cuatro bits con generación de acarreo anticipado interno proporcionando un acarreo de salida para el acoplamiento serie con otros sumadores. La Figura 7.22 muestra el circuito de un sumador de dieciséis bits con acarreo mixto.

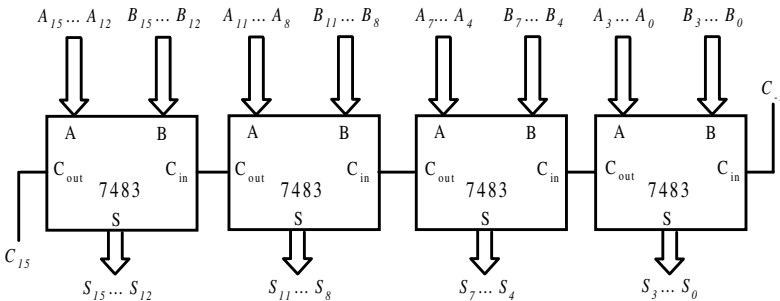


Figura 7.22. Sumador de 16 bits con acarreo mixto

## 7.3 RESTADORES BINARIOS

El desarrollo de los circuitos restadores binarios puede realizarse siguiendo el procedimiento expuesto para los sumadores, así se debería realizar el semirrestador, restador completo y restadores paralelos con acarreo serie y/o anticipado. De acuerdo con este procedimiento, a continuación se expone la tabla de verdad del semirrestador como función  $R = A - B$  y su circuito lógico.

Tabla 7.4. Función de salida de un semirrestador

Entradas		Salidas	
B	A	C	R
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

A partir de la Tabla 7.4 se obtienen las funciones lógicas  $R$  y  $C$ , que se muestran en la expresión [7.11].

$$R = \bar{B} A + B \bar{A} = B \oplus A \quad [7.11]$$

$$C = \bar{B} A$$

En la Figura 7.23 se muestra su realización a partir de puertas lógicas básicas.

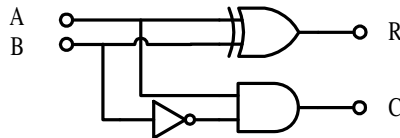


Figura 7.23. Circuito lógico del semirrestador

Sin embargo, este dispositivo no se emplea en la práctica pues es preferible realizar una codificación de números con signo y emplear sumadores basados en el principio:

$$A - B = A + (-B) \quad [7.12]$$

Para la conversión del operando  $B$  a un valor con signo se puede emplear la codificación en **complemento a uno** o **complemento a dos**, siendo esta última la más utilizada en la práctica.

La codificación en complemento a uno presenta el inconveniente de la duplicación del código cero con signo positivo ( $00 \dots 00$ ) y negativo ( $11 \dots 11$ ) y la propagación de los acarreos.

La codificación en complemento a dos resuelve el problema de la duplicidad del código cero y ciertas consideraciones sobre los acarreos, pero presenta el inconveniente de la asimetría en la codificación respecto del valor central. Para más información, véase el apartado 1.2.4. Representación de números en coma fija con signo.

### PROBLEMA RESUELTO 7-9



Simular el funcionamiento del semirrestador representado en la Figura 7.23, utilizando para ello el programa *Electronics Workbench*.

#### Solución:

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap07\Ewb5\07W0\_\_06.ewb**

Mediante el empleo de los conmutadores A y B, asociados a las teclas activas [1] y [2] respectivamente, realizar la comprobación de la función de salida descrita en la Tabla 7.4.

### 7.3.1 Aritmética en complemento a dos

Como ya se ha visto en la expresión [7.12], los restadores pueden realizarse a partir de sumadores empleando codificadores con criterio de signo.

$$S = \pm A + (\pm B) \quad [7.13]$$

No obstante una forma más genérica de abordar el problema consiste en reducir las operaciones a sumas en complemento a dos, donde las variables  $A$  y  $B$  pueden tener cualquier signo. De este modo se obtendrá un **sumador-restador en complemento a dos**.

En la Figura 7.25 se muestra el símbolo asociado a un circuito sumador-restador de cuatro bits en complemento a dos.

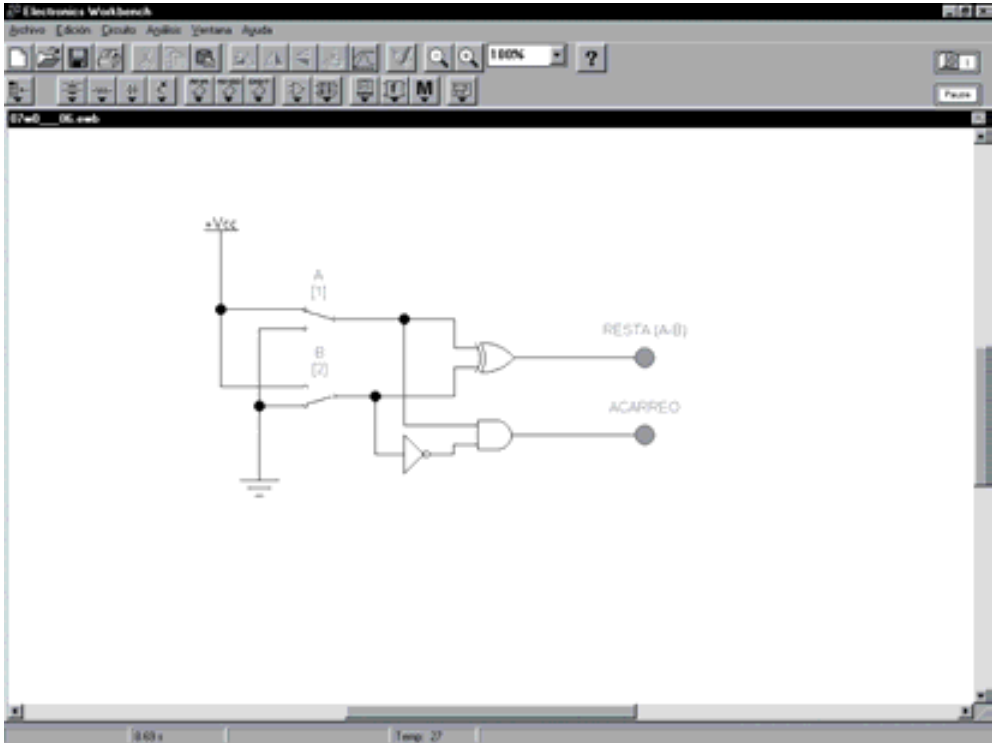


Figura 7.24. Circuito de simulación de un semisumador

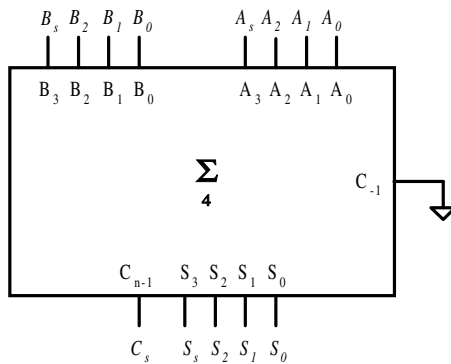


Figura 7.25. Circuito sumador-restador de 4 bits en complemento a dos

A partir de las combinaciones de signos de las variables  $A$  y  $B$  se pueden observar una serie de circunstancias o casos que se describen a continuación.

- **A y B positivos:**  $S = A + B$

Si se aplica la información  $A$  y  $B$  al sumador completo de la Figura 7.25 incluyendo los dos bits de signo ( $A_s$  y  $B_s$ ), el resultado deberá ser positivo  $S_s = 0$ .

**Ejemplo:**

$A = +3$       0011  
 $B = +2$       0010  
 $A + B = +5$     0101

$$\begin{array}{r} A \ 0011 \\ B \ 0010 \\ \hline S \ 0101 \end{array}$$

El resultado  $S = 0101$  se corresponde al valor  $+5$  y no se genera acarreo.

El desbordamiento  $Ov$  (*overflow*) se producirá cuando exista un acarreo parcial del bit  $S_{n-2}$  sobre  $S_{n-1}$  que representa el bit de signo, denominado  $S_s$ , resultando negativo (igual a "1"). El acarreo  $C_{n-1}$  no interviene en la operación y por tanto se desprecia.

**Ejemplo:**

$A = +3$       0011  
 $B = +6$       0110  
 $A + B = +9$     1001    Desbordamiento

$$\begin{array}{r} A \ 0011 \\ B \ 0110 \\ \hline S \ 1001 \end{array}$$

El resultado  $S = 1001$  se corresponde al valor  $-7$ , en lugar del resultado correcto.

Por consiguiente, se puede definir la condición de desbordamiento  $Ov$  para esta circunstancia a partir de la expresión:

$$Ov = \overline{A_s} \cdot \overline{B_s} \cdot S_s \quad [7.14]$$

- **A positivo y B negativo:**  $S = A + B$

Si se aplica la información  $A$  y  $B$  al sumador completo de la Figura 7.25, incluyendo los dos bits de signo  $A_s$  y  $B_s$ , el signo del resultado  $S_s$  se deberá corresponder con el signo asociado a la información de mayor valor absoluto y en ningún caso se producirá desbordamiento.

**Ejemplo:**

$A = +3$       0011  
 $B = -5$       1011  
 $A + B = -2$     1110

$$\begin{array}{r} A \ 0011 \\ B \ 1011 \\ \hline S \ 1110 \end{array}$$

El resultado  $S = 1110$  se corresponde al valor  $-2$ , siendo un resultado correcto.

- **A negativo y B positivo:**  $S = A + B$

De este caso se pueden extraer las mismas observaciones que en el caso anterior.

- **A negativo y B negativo:**  $S = A + B$

Si se aplica la información  $A$  y  $B$  al sumador completo de la Figura 7.25, incluyendo los dos bits de signo, el resultado deberá ser negativo. El sumador genera un bit de acarreo  $C_{n-1}$  que al igual que en los casos anteriores no interviene en la operación y, por tanto, se desprecia.

**Ejemplo:**

$A = -2$       1110  
 $B = -3$       1101  
 $A + B = -5$     1011

$$\begin{array}{r} A \ 1110 \\ B \ 1101 \\ \hline S \ 11011 \end{array}$$

El resultado  $S = 11011$ , genera un “1” de acarreo  $C_{n-1}$ . Despreciando el acarreo, se corresponde al valor  $-5$ .

El desbordamiento ( $Ov$ ) se producirá cuando exista un acarreo parcial del bit  $S_{n-2}$  sobre  $S_{n-1}$  que representa el bit de signo  $S_s$  resultando positivo (igual a “0”).

**Ejemplo:**

$A = -3$             1101  
 $B = -6$             1010  
 $A + B = -9$         0111    Desbordamiento

$A$  1101  
 $B$  1010  


---

 $S$  10111

El resultado  $S = 10111$ , despreciando el acarreo  $C_{n-1}$ , se corresponde al valor +7 en lugar del resultado correcto.

Por tanto, se puede definir la condición de desbordamiento para esta circunstancia a partir de la expresión:

$$Ov = A_s \cdot B_s \cdot \overline{S_s} \tag{7.15}$$

En términos generales se puede definir el desbordamiento  $Ov$  a partir de las expresiones [7.14] y [7.15], según se indica en la Figura 7.21.

$$Ov = \overline{A_s} \cdot \overline{B_s} \cdot S_s + A_s \cdot B_s \cdot \overline{S_s} \tag{7.16}$$

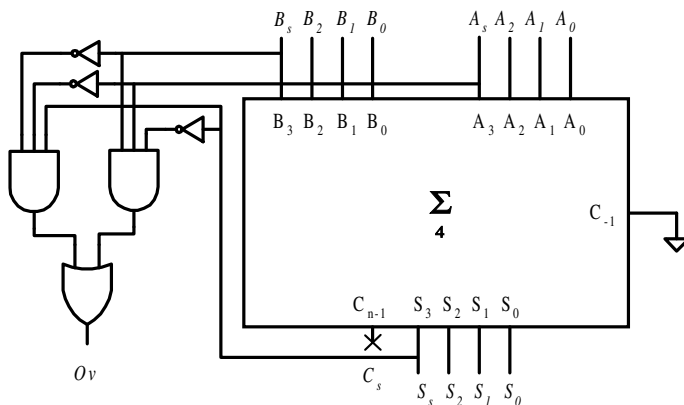


Figura 7.26. Circuito sumador-restador de 4 bits en complemento a dos con detector de desbordamiento  $Ov$

### 7.3.2 Circuito complementador a dos

Conforme a la expresión [7.12] la función que debe realizar un circuito complementador a dos debe ser la de convertir una información codificada **en binario natural a complemento a dos**.

La función de conversión se realiza mediante la operación aritmética que se muestra en la expresión [7.17].

$$\text{Complemento a dos de } (B) = \overline{B} + 1 \quad [7.17]$$

Sin embargo, para desarrollar un circuito sumador-restador sobre informaciones numéricas codificadas en binario natural se requiere implementar un circuito complementador a dos que disponga de un terminal de control que permita una doble función dependiente de la operación de suma o resta según la Tabla 7.5.

Tabla 7.5. Función de conversión del circuito complementador a dos

Terminal de control	Información binaria de entrada	Función de salida
$C$		
0	$B$	$S = B$
1	$B$	$S = B_{(C2)} = \overline{B} + 1$

Por consiguiente, ante la entrada de control  $C = 0$  el circuito es **transparente** a la información de entrada, realizando verdaderamente la función de complemento a dos ante la entrada de control  $C = 1$ . El circuito de la Figura 7.27 realiza la función descrita en la Tabla 7.5 mediante el uso de puertas OR-EXCLUSIVA como inversores dependientes de la variable  $C$ , según se puede observar en la Tabla 7.6, donde se obtiene la **función inversión/no inversión** de la variable  $B$  en función del estado del terminal  $C$ .

Tabla 7.6. Función OR-EXCLUSIVA

Entradas		Salidas	
$A$	$B$	$S$	
0	0	0	$S = B$
0	1	1	No invierte $B$
1	0	1	$S = \overline{B}$
1	1	0	Invierte $B$



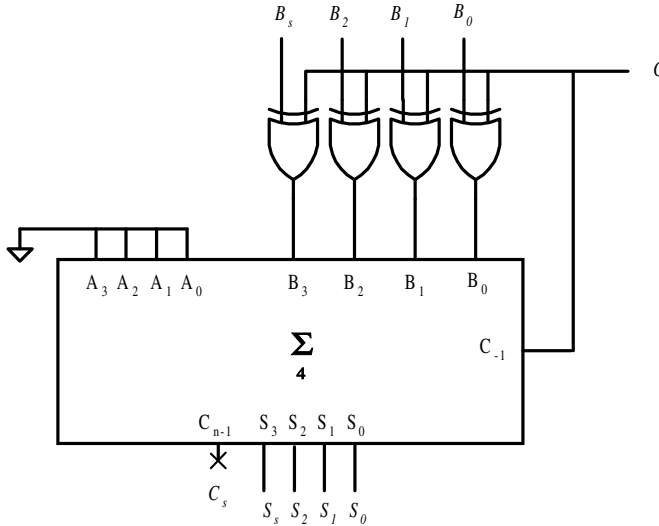


Figura 7.27. Circuito complementador a dos de 4 bits

- La función realizada por el circuito de la Figura 7.27 para  $C = 0$ , será:

$$S = A + B + C_{-1} = 0 + B + 0 = B \tag{7.18}$$

- La función realizada por el circuito de la Figura 7.27 para  $C = 1$ , será:

$$S = A + \bar{B} + C_{-1} = 0 + \bar{B} + 1 = \bar{B} + 1 \tag{7.19}$$

En ambos casos, con el bit de signo incluido.

**PROBLEMA RESUELTO 7-10**



Simular el funcionamiento de un circuito complementador a dos representado en la Figura 7.27, utilizando para ello el programa *Electronics Workbench*.

**Solución:**

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap07\Ewb5\07W0\_\_07.ewb**

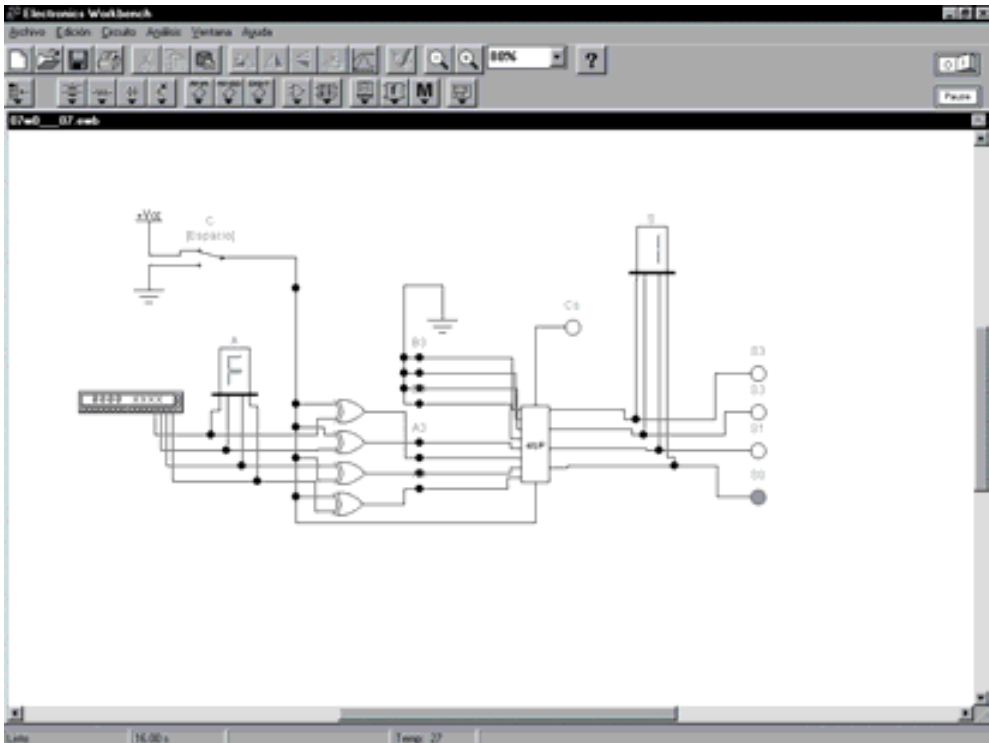


Figura 7.28. Simulación del circuito complementador a dos de cuatro bits

Mediante el empleo del terminal de control  $C$ , que se encuentra asociado a la tecla [ESPACIO], se realiza la función complemento a dos. Para el caso en el que  $C = 1$  hace referencia a la información de cuatro bits proporcionada por el generador de palabras y para el caso  $C = 0$ , hace referencia a la función transparencia de información. Se deja como ejercicio para el lector:

- Realizar la comprobación de la conversión de código binario natural a complemento a dos mediante la lectura de los indicadores siete segmentos en notación hexadecimal, en la posición del conmutador  $C = 1$ .
- Realizar la comprobación de la función transparencia de información en la posición del conmutador  $C = 0$ .

### PROBLEMA RESUELTO 7-11



Simular el funcionamiento de un circuito sumador-restador de cuatro bits que realice la función  $S = A \pm B$ , siendo  $A$  y  $B$  dos operandos codificados en binario natural, utilizando para ello el programa *Electronics Workbench*. La función suma-resta se realizará mediante un terminal de control  $C$  que ha descrito anteriormente.

**Solución:**

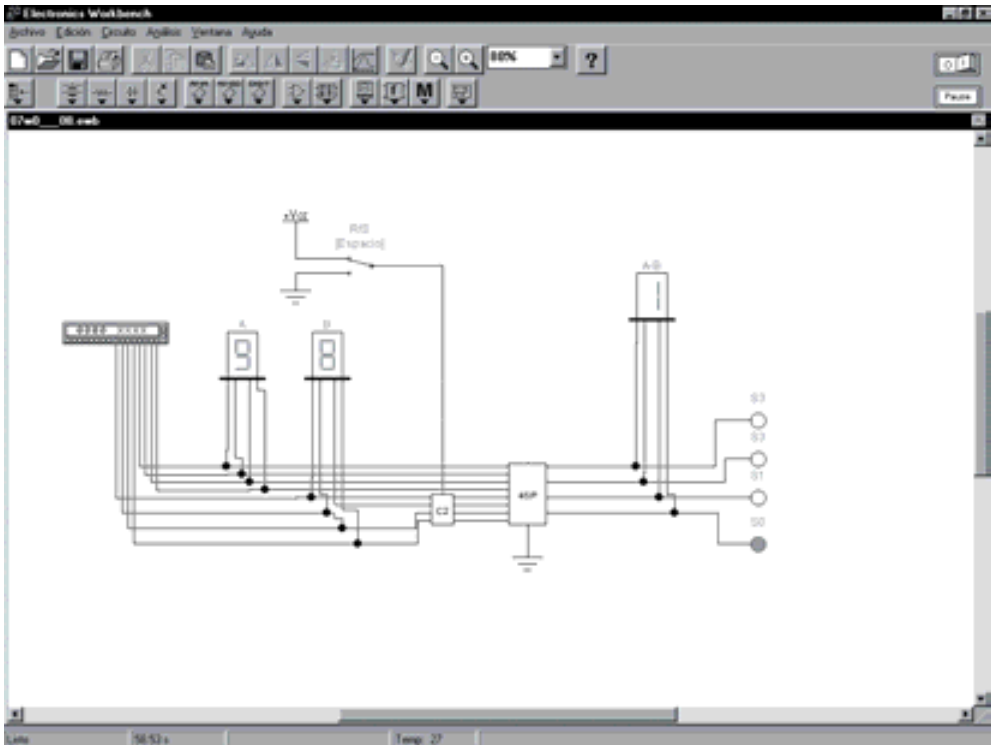
La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap07\Ewb5\07W0\_08.ewb**

Puesto que las informaciones A y B están representadas en binario natural, la operación de resta se debe realizar a partir del complemento a dos de B.

$$A - B = A + (-B)$$

Puesto que también se debe realizar la operación de suma, el circuito complementador a dos debe de incluir la función de transparencia como un modo de funcionamiento.



*Figura 7.29. Circuito para simular el funcionamiento de un sumador-restador de 4 bits*

Mediante el empleo del conmutador *R/S* con la tecla activa espacio, realizar la comprobación de la función de salida  $S = A \pm B$ .

El conmutador  $R/S$  actúa sobre el terminal de control  $C$  realizando la función complemento a dos ( $R/S = 1$ ) de la información  $B$  o la función transparencia de información ( $R/S = 0$ ). Se deja como ejercicio para el lector:

- Realizar la comprobación de la función de salida  $S = A + B$  en la posición del conmutador  $R/S = 0$ . Observar el desbordamiento para valores de  $S$  superior al valor  $F$ , en hexadecimal.
- Realizar la comprobación de la función de salida  $S = A - B$  en la posición del conmutador  $R/S = 1$ . Observar que el resultado para valores de  $S$  negativos viene expresado en complemento a dos.

### 7.3.3 Circuito sumador-restador en binario signo magnitud

Cuando la información de entrada se encuentra codificada en binario signo magnitud es preciso realizar una conversión de código a complemento a dos para operar en aritmética de complemento a dos. La función de conversión se puede obtener a partir de la Tabla 7.7 donde se observa que la codificación de los números positivos es coincidente en ambos códigos y para la conversión de los números negativos se aplica el siguiente procedimiento básico:

- 1) Se parte de la **codificación positiva** en complemento a dos o binario signo magnitud.
- 2) Se aplica la función **complemento a dos C2** a la información, bit de signo incluido.

Dicha función se obtiene a partir de un **circuito complementador a dos** conectado según se muestra en la Figura 7.30 (a).

Tabla 7.7. Función de conversión binario signo magnitud a complemento a dos

Binario Natural con signo magnitud	Complemento a dos	Valor decimal
0 1 1	0 1 1	+ 3
0 1 0	0 1 0	+ 2
0 0 1	0 0 1	+ 1
0 0 0	0 0 0	+ 0
1 0 1	1 1 1	- 1
1 1 0	1 1 0	- 2
1 1 1	1 0 1	- 3
---	1 0 0	- 4

Puesto que el resultado del circuito sumador-restador debe expresarse en el mismo sistema de codificación que las variables de entrada, es preciso disponer de un **convertidor de complemento a dos a binario signo magnitud**. La función de conversión se puede obtener a partir de la Tabla 7.7 aplicando el siguiente procedimiento básico cuando la información resultante es negativa:

- 1) El bit de signo  $A_s$  en C2 se aplica directamente como bit de signo del código binario signo magnitud.
- 2) Se aplica la función *complemento a dos* C2 a la información, ignorando el bit de signo obtenido.

Dicha función se obtiene a partir de un *circuito complementador a dos* C2, conexionado según se muestra en la Figura 7.30 (b).

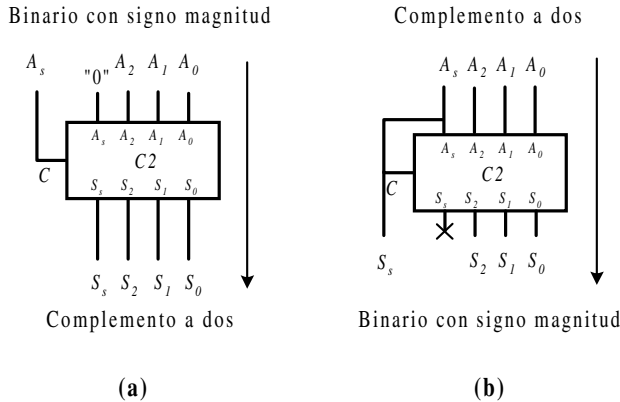


Figura 7.30. Circuito convertidor de código: (a) Binario con signo magnitud a C2 y (b) C2 a Binario con signo magnitud

El circuito de la Figura 7.31 está basado en el **sumador-restador en complemento a dos** y realiza la función  $S = \pm A + (\pm B)$  siendo  $A$  y  $B$  información numérica positiva o negativa codificada en binario signo magnitud. Los bloques convertidores a *complemento a dos*, C2, realizan la conversión siempre que el bit de signo sea negativo, presentando la salida  $S$  en **Binario signo magnitud** cuando el bit de signo de la salida  $S_s$  sea negativo. El sumador completo siempre realizará la función de suma en complemento a dos.

Debe de observarse que si se realiza la función C2 sobre una información negativa en complemento a dos, ésta queda codificada positivamente. Por esta razón, el bit de signo del resultado  $S_s$  se incorpora directamente como bit de signo en la salida.

El circuito de la Figura 7.32, basándose en la misma idea anterior, realiza la función  $S = \pm A \pm (\pm B)$ , siendo  $A$  y  $B$  información numérica positiva o negativa codificada en binario signo magnitud, presentando la salida  $S$  codificada en binario signo magnitud.

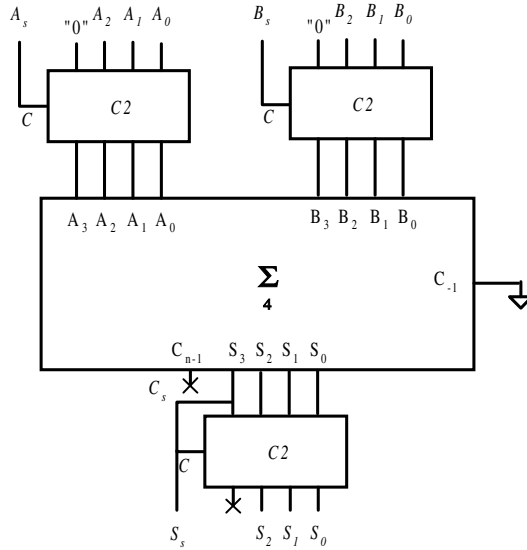


Figura 7.31. Circuito sumador de 4 bits para códigos de entrada y salida en binario natural con signo

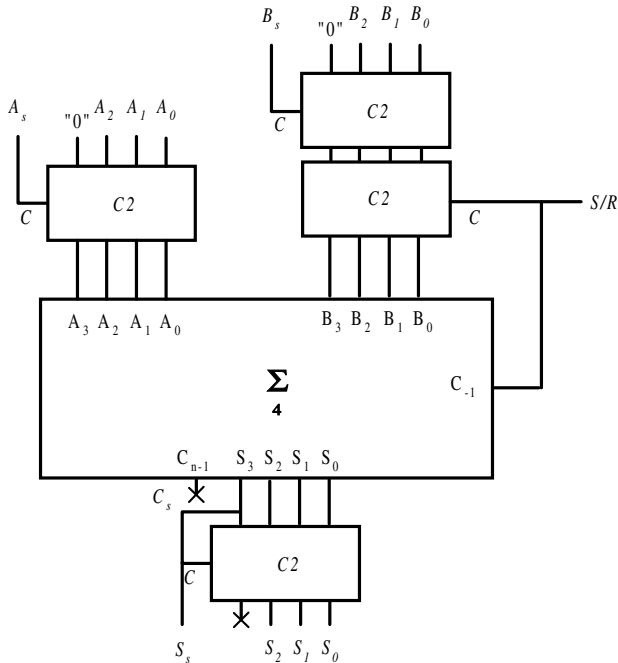


Figura 7.32. Circuito sumador-restador de 4 bits para códigos de entrada y salida en binario signo magnitud

## PROBLEMA RESUELTO 7-12



Simular el funcionamiento de un circuito sumador-restador de cuatro bits que realice la función  $S = \pm A + (\pm B)$ , siendo  $A$  y  $B$  dos operandos codificados en binario natural con signo magnitud. Utilizar para ello el programa *Electronics Workbench*.

### Solución:

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap07\Ewb5\07W0\_10.ewb**

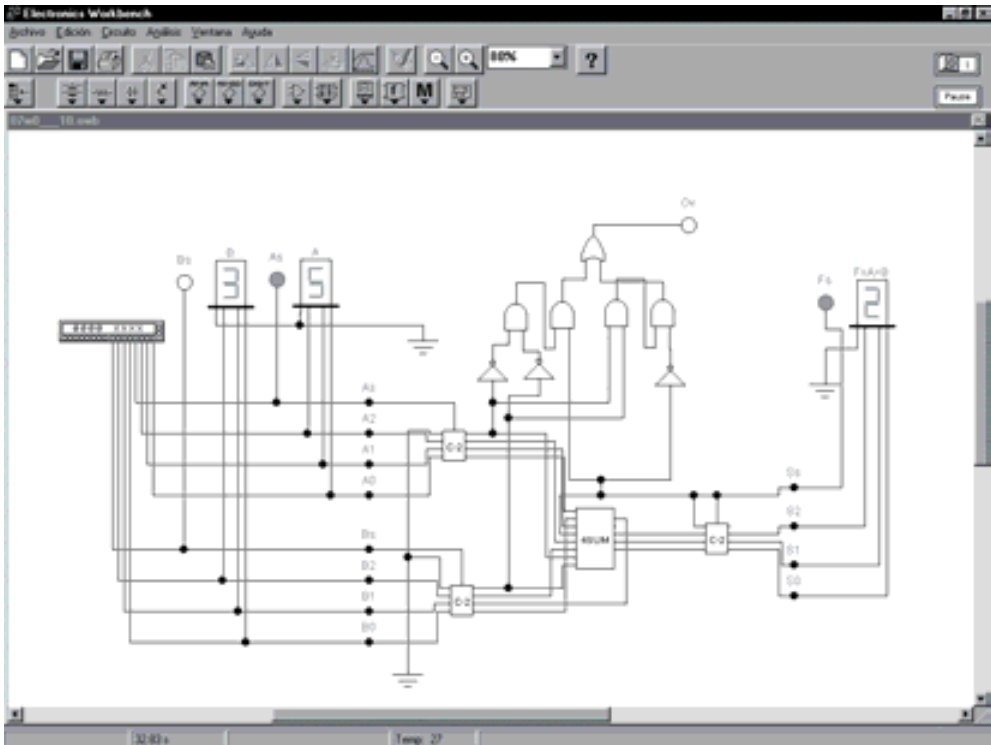


Figura 7.33. Circuito para simular el funcionamiento de un sumador-restador de 4 bits codificado en binario signo magnitud

Puesto que la información  $A$  y  $B$  está representada en binario signo magnitud, la operación de sumador-restador se debe realizar a partir del complemento a dos de las variables  $A$  y  $B$  condicionado al signo de la información numérica.

Por tanto, el circuito complementador a dos debe de incluir la función de transparencia como un modo de funcionamiento.

$$A - B = \pm A + (\pm B)$$

Mediante el empleo del generador de palabras binarias se obtiene la información de las variables  $A$  y  $B$  codificadas en binario signo magnitud. El valor de la magnitud queda representado por los tres bits de menor peso y se visualiza en los indicadores de siete segmentos. El valor del signo queda representado por el bit de mayor peso, visualizándose en el indicador luminoso situado junto al indicador de siete segmentos. El circuito de simulación incluye un indicador de *overflow* o desbordamiento descrito en la expresión [7.16].

- Realizar la comprobación de la función de salida  $S = \pm A + (\pm B)$ . Observar el resultado, signo incluido y las condiciones de desbordamiento para valores de  $S$  superior a  $\pm 7$ .

## 7.4 UNIDAD ARITMÉTICO-LÓGICA (ALU)

Una **Unidad Aritmético-Lógica** (*Arithmetic-Logic Unit - ALU*) es un dispositivo capaz de realizar operaciones lógicas y aritméticas entre dos informaciones binarias aplicadas a su entrada.

La Figura 7.34 muestra el diagrama lógico de este dispositivo, donde se pueden apreciar los siguientes terminales:

- **Operandos de entrada**  $A$  y  $B$  generalmente de cuatro bits.
- **Operando de salida** de resultado  $F$ .
- **Selector de operación**  $S$  mediante el cual se selecciona la operación a realizar en los operandos de entrada.
- **Entradas y salidas auxiliares**. En este grupo se encuentran los terminales de acarreo inicial y de salida, indicación de desbordamiento, etc.

Las ALUs comerciales generalmente operan con un formato de palabra binaria de cuatro bits, siendo fácilmente ampliables a órdenes superiores. Los circuitos integrados más utilizados en la familia TTL son el 74381 y el 74382. El primero opera bajo la técnica de generación de acarreo anticipado y el segundo bajo la técnica de propagación de acarreo serie.

La Tabla 7.8 muestra las condiciones de funcionamiento de la ALU, donde la operación a realizar se fija mediante la combinación de bits  $S_2$ ,  $S_1$  y  $S_0$ . Con el fin de diferenciar claramente las operaciones lógicas de las aritméticas, generalmente se emplea la notación *plus/más* y *minus/menos* para referenciar las sumas y restas aritméticas y a la vez diferenciar la suma aritmética *plus/más* de la suma lógica representada mediante el signo +.



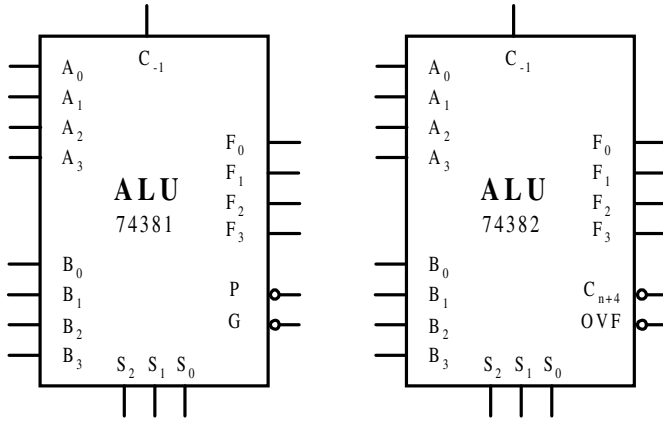


Figura 7.34. Diagrama lógico de las ALUs comerciales 74381 y 74382

Tabla 7.8. Selección de operaciones en la ALU 74381-382

Selector de operación			Operación	$C_{in}$	Comentario
$S_2$	$S_1$	$S_0$			
0	0	0	Clear		
0	0	1	$B \text{ plus } \overline{A}$	0	$B \text{ minus } A \text{ minus } 1$
0	0	1	$B \text{ plus } \overline{A}$	1	$B \text{ minus } A$
0	1	0	$A \text{ plus } \overline{B}$	0	$A \text{ minus } B \text{ minus } 1$
0	1	0	$A \text{ plus } \overline{B}$	1	$A \text{ minus } B$
0	1	1	$A \text{ plus } B$	0	$A \text{ plus } B$
0	1	1	$A \text{ plus } B$	1	$A \text{ plus } B \text{ plus } 1$
1	0	0	$A \oplus B$		$A \text{ XOR } B$
1	0	1	$A + B$		$A \text{ OR } B$
1	1	0	$A \cdot B$		$A \text{ AND } B$
1	1	1	Preset		

Es importante destacar que la condición de acarreo activo en una operación de resta se produce con un cero lógico.

La asociación de ALUs para operaciones con palabras de orden superior a cuatro bits se realiza del mismo modo que los sumadores empleando las técnicas de generación de acarreo anticipado o de propagación de acarreo serie.

## PROBLEMA RESUELTO 7-13



Simular, mediante la descripción en VHDL, la ALU 74382 representada en la Figura 7.34, utilizando para ello el módulo *Simulate Demo* del programa *OrCAD Demo v9*. Este fichero se debe abrir directamente desde el módulo indicado para evitar posibles problemas durante su ejecución.

### Solución:

Mediante el lenguaje VHDL se realiza una descripción en forma funcional de la ALU 74382 y se simulan algunas operaciones mediante cronograma.

La ruta y el nombre del fichero de proyecto que contiene esta descripción VHDL es la que se indica a continuación:

**D:\Ejemplos\Cap07\OrCAD9\07R0\_\_04\07R0\_\_04.opj**

Dentro del proyecto existen varias ventanas que se describen a continuación. Una de estas ventanas contiene el fichero de texto en el que se describe el código VHDL de la ALU 74382, y que se ha representado en la Figura 7.35 (cabe indicar, que el contenido de esta figura se extiende a dos páginas).

```

-----
-- Fichero :          D:\Ejemplos\Cap07\OrCAD9\07R0__04\07R0__04.vhd
--          Modelo funcional de la unidad aritmética y lógica (ALU) 74382
-----
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
ENTITY ALU_74382 IS

    PORT(
        A,B : in std_logic_vector(3 downto 0); -- datos A3..A0 y B3..B0
        S : in std_logic_vector(2 downto 0); -- selector de operación S2..S0
        Cin : in std_logic;                -- acarreo de entrada
        F : out std_logic_vector(3 downto 0); -- resultado F3..F0
        Cout : out std_logic;              -- acarreo de salida
        OVF : out std_logic;               -- flags overflow
    );
END ALU_74382;

-- Descripción funcional
ARCHITECTURE funcional OF ALU_74382 IS

    begin
        process (A, B, S, Cin)
            variable aux_F: std_logic_vector(4 downto 0);
            variable aux_A, aux_B: std_logic_vector(4 downto 0);
            variable aux_Cout, aux_OVF: std_logic;
            variable tp_aritmetica: time := 20 ns; -- retardo aritméticas
            variable tp_logica: time := 10 ns; -- retardo lógicas
            variable retardo: time;
            begin
                aux_A:= ('0' & A);
                aux_B:= ('0' & B);
                case S is
                    when "000" =>

```

```

aux_F := "00000";           -- Clear
    retardo:= tp_logica;
when "001" =>
    if (Cin = '0') then aux_F:= aux_B-aux_A-1;
-- B plus /A con Cin = 0, (en complemento a dos: B minus A minus
    else aux_F:= aux_B-aux_A;
-- B plus /A con Cin = 1, (en complemento a dos: B minus A)
    end if;
    aux_cout:= aux_F(4);
    retardo:= tp_aritmetica;
when "010" =>
    if (Cin = '0') then aux_F:= aux_A-aux_B-1;
-- A plus /B con Cin = 0, (en complemento a dos: A minus B minus 1)
    else aux_F:= aux_A-aux_B;
-- A plus /B con Cin = 1, (en complemento a dos: A minus B)
    end if;
    aux_cout:= aux_F(4);
    retardo:= tp_aritmetica;
when "011" =>
    if (Cin = '0') then aux_F:= aux_A+aux_B;    -- A plus B
    else aux_F:= aux_A+aux_B+1;    -- A plus B plus 1
    end if;
    aux_cout:= aux_F(4);
    retardo:= tp_aritmetica;
when "100" =>
    aux_F := aux_A xor aux_B;           -- A XOR B
    retardo:= tp_logica;
when "101" =>
    aux_F := aux_A or aux_B;           -- A OR B
    retardo:= tp_logica;
when "110" =>
    aux_F := aux_A and aux_B;         -- A AND B
    retardo:= tp_logica;
when "111" =>
    aux_F := "11111";                 -- Preset
    retardo:= tp_logica;
when others => -- error
    aux_F:= (others =>'X');
end case;
-- asignación del resultado, acarreo de salida y su retardo
F <= aux_F(3 downto 0) after retardo;
-- determinación de los flags
if (s="000" or s(2) = '1') then aux_OVF:='X';
    aux_Cout:= 'X';
    elsif ((aux_A(3)='0') and (aux_B(3)='0') and (aux_F(3)='1')) or
    ((aux_A(3)='1') and (aux_B(3)='1') and (aux_F(3)='0')) then
        aux_OVF := '1';
    else
        aux_OVF := '0';
    end if;
OVF<= aux_OVF;
Cout<= aux_Cout;
end process;
end funcional;

```

Figura 7.35. Descripción funcional mediante VHDL de la ALU 74382

En la Figura 7.36 se muestran las ventanas que componen el proyecto, en las que se relacionan:

- ◆ los documentos que integran el proyecto (ventana 07r0\_\_04),
- ◆ el cronograma de las señales de entrada y salida de la ALU 74382 (ventana *Wave1*) y,
- ◆ su descripción en VHDL (ventana 07r0\_\_04).

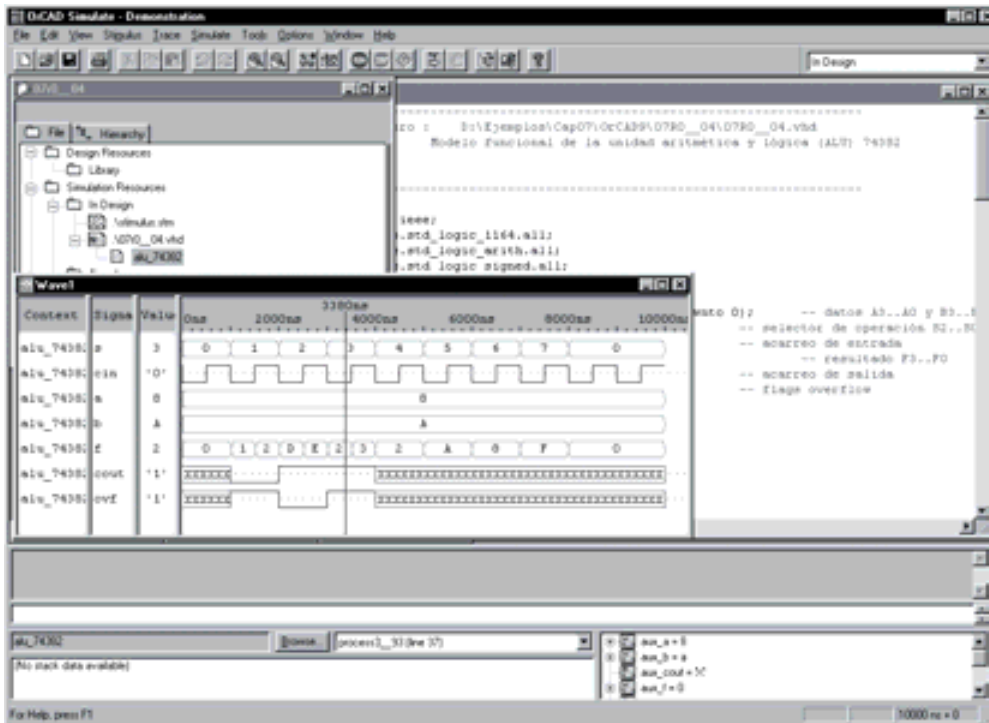


Figura 7.36. Circuito de simulación de la ALU 74382

Situando el cursor sobre cada uno de los puntos de interés, por ejemplo, en cada uno de los sucesivos valores que van tomando las señales, se pueden obtener los diferentes resultados de las operaciones lógicas y aritméticas que puede realizar esta ALU. Obsérvese que los datos de entrada *a* y *b* son constantes y que se consideran indistintos 'X' los niveles lógicos de los flags de acarreo *C<sub>out</sub>* y *overflow* en las operaciones lógicas.

Se puede comprobar en el cronograma los retardos de propagación en las señales de salida *t<sub>p</sub>* que se han definido con valor diferente en las operaciones aritméticas y lógicas.

**PROBLEMA RESUELTO 7-14**



Simular el funcionamiento de una unidad Aritmético-Lógica (ALU) de cuatro bits, utilizando para ello el programa *Electronics Workbench*.

**Solución:**

El programa *Electronics Workbench* solamente dispone de la posibilidad de realizar la simulación de la ALU 74181. La descripción de este circuito a partir de sus terminales es la siguiente:

- **Operandos de entrada A y B.** Indicados como  $A_3, \dots, A_1$  y  $B_3, \dots, B_1$ .
- **Operando de salida de resultado F.** Indicado como  $F_3, \dots, F_1$ .
- **Selector de operación S.** Indicado como  $S_3, \dots, S_0$ , cuya función se describe en la Tabla 7.9 que se muestra a continuación.

*Tabla 7.9. Selección de operaciones en la ALU 74181 con entradas y salidas activas a nivel alto*

Selector de operación				Lógica	Aritmética*
$S_3$	$S_2$	$S_1$	$S_0$	$M=H$	$M=L, C_n = H$
0	0	0	0	$\overline{A}$	A
0	0	0	1	$\overline{A+B}$	$A+B$
0	0	1	0	$\overline{A} \cdot B$	$A + \overline{B}$
0	0	1	1	Lógico 0	menos 1 (Compl. a 2)
0	1	0	0	$\overline{A \cdot B}$	A más $A \cdot \overline{B}$
0	1	0	1	$\overline{B}$	$(A+B)$ más $A \cdot \overline{B}$
0	1	1	0	$A \oplus B$	A menos B menos 1
0	1	1	1	$A \cdot \overline{B}$	$A \cdot \overline{B}$ menos 1
1	0	0	0	$\overline{A+B}$	A más $A \cdot B$
1	0	0	1	$\overline{A \oplus B}$	A más B
1	0	1	0	B	$(A + \overline{B})$ más $A \cdot B$
1	0	1	1	$A \cdot B$	$A \cdot B$ menos 1
1	1	0	0	Lógico 1	A más A
1	1	0	1	$A + \overline{B}$	$(A + B)$ más A
1	1	1	0	$A + B$	$(A + \overline{B})$ más A
1	1	1	1	A	A menos 1

(\*) Todas las operaciones aritméticas se realizan en complemento a dos

- **Entradas y salidas auxiliares:**

- ◆  $C_n$ . Acarreo de entrada activo a nivel bajo.
- ◆  $C_{n+4}$ . Acarreo de salida activo a nivel bajo.
- ◆  $M$ . Selector de operaciones Lógicas o Aritméticas. A nivel bajo, los acarrees internos quedan habilitados para la realización de las funciones aritméticas. A nivel alto, los acarrees internos quedan inhibidos para la realización de las funciones lógicas.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap07\Ewb5\07W0\_\_11.ewb**

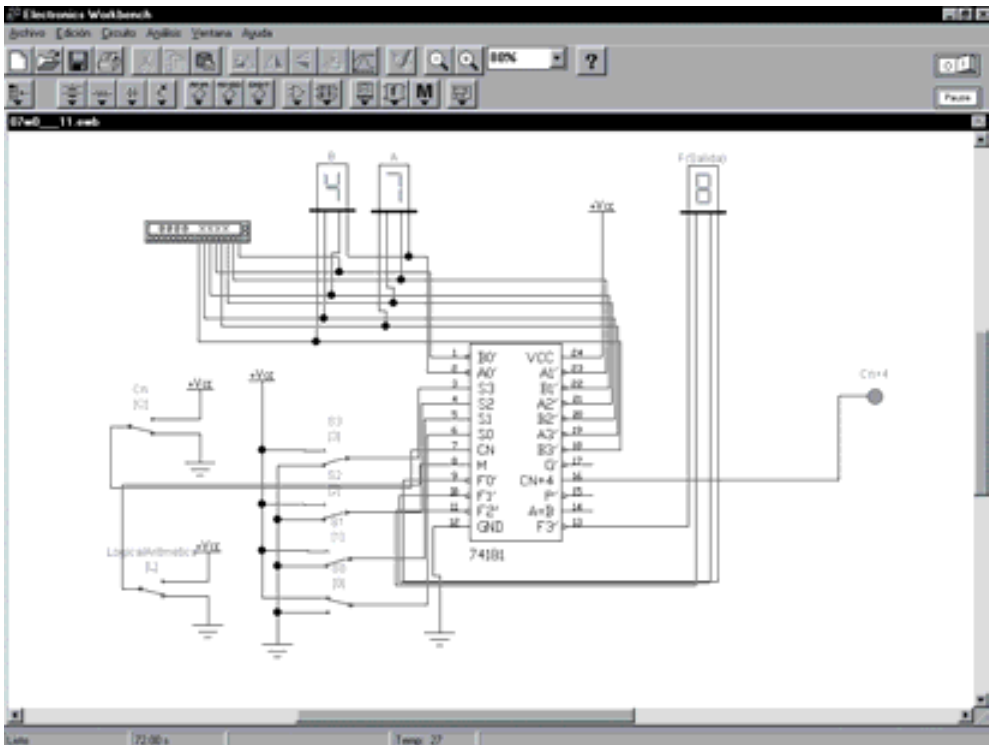


Figura 7.37. Circuito para simular el funcionamiento de la ALU 74181

Mediante el conmutador  $C_n$ , asociado a la tecla activa [C], fijar el valor del acarreo inicial, recordando que el acarreo activo es a nivel bajo. Con el conmutador *Lógica/Aritmética* cuya tecla activa es [L], fijar el tipo de operación: Lógica a nivel alto y aritmética a nivel bajo.

Mediante el empleo del conmutador  $S_0, \dots, S_3$ , cuyas teclas activas son: [0], [1], [2] y [3] respectivamente, seleccionar la función a realizar según se ha definido en la Tabla 7.9.

- Realizar la comprobación de diversas funciones lógicas con la posición del conmutador *Lógica/Aritmética* = 1. Observar la influencia del acarreo inicial  $C_n$ .
- Realizar la comprobación de diversas funciones aritméticas con la posición del conmutador *Lógica/Aritmética* = 0. Observar la influencia del acarreo inicial  $C_n$ . Recordar que todas las operaciones aritméticas se realizan en complemento a dos.

## 7.5 APLICACIONES DE LOS CIRCUITOS ARITMÉTICOS

### 7.5.1 Comparadores de magnitud binarios

Un comparador de magnitud es un dispositivo lógico que proporciona a su salida información relativa a la comparación entre dos informaciones binarias  $A$  y  $B$  de  $n$  bits según la tabla de verdad (Tabla 7.10) que se muestra a continuación.

Tabla 7.10. Función de un comparador binario

Entradas	Salidas		
	$A > B$	$A = B$	$A < B$
$A > B$	1	0	0
$A = B$	0	1	0
$A < B$	0	0	1

La función de comparación se puede obtener a partir de un restador binario en complemento a uno, dado que:

$$A + \bar{A} = 1 \quad [7.20]$$

Si a partir de la expresión [7.20] se realiza la suma ( $A + \text{complemento a uno de } B$ ) se obtienen las condiciones para la salida del sumador completo en función de las distintas magnitudes de las variables de entrada. Estas condiciones se muestran en la Tabla 7.11.

Tabla 7.11. Función de comparación

Entradas		Salidas				
A	B	$C_s$	$S_3$	$S_2$	$S_1$	$S_0$
$A > B$		1	$S \neq 1$	1	1	1
$A = B$		0	1	1	1	1
$A < B$		0	$S \neq 1$	1	1	1

En la Figura 7.38 se muestra el esquema de un circuito comparador binario de cuatro bits mediante un restador binario en complemento a uno.

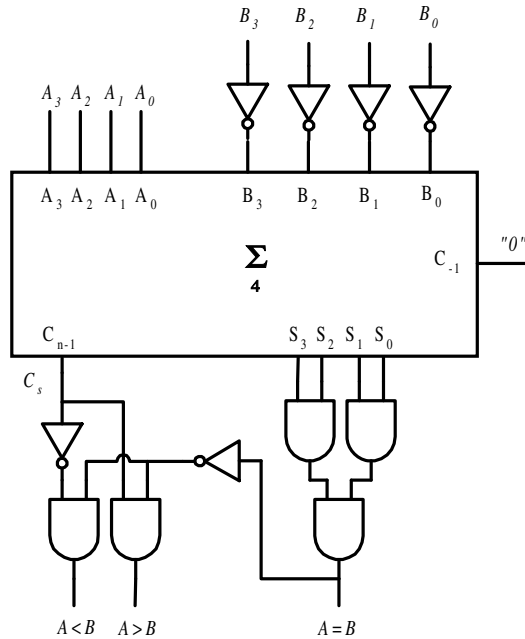


Figura 7.38. Circuito comparador binario de cuatro bits mediante restador binario en complemento a uno

De igual forma la función de comparación también puede obtenerse a partir de un restador binario en complemento a dos.

Se propone al lector la realización de un circuito comparador basado en un restador binario en complemento a dos.



## 7.5.2 Convertidor de código BCD a Exceso-3

Puesto que la relación de conversión entre el **código BCD Natural y Exceso-3** es la suma aritmética de tres unidades, la aplicación del convertidor resulta inmediata mediante el empleo de un sumador completo.

En la Figura 7.39 se muestra el esquema de un circuito que realiza la conversión de código BCD Natural a Exceso-3.

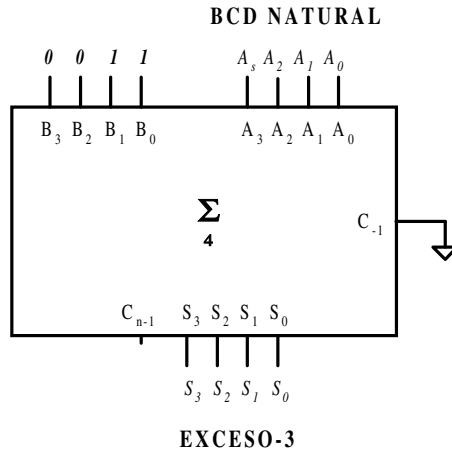


Figura 7.39. Circuito convertidor de código BCD a Exceso-3

### PROBLEMAS PROPUESTOS

- 7-1) Diseñar un circuito que realice el complemento a nueve de datos codificados en BCD natural. Idem para el complemento a diez.
- 7-2) Realizar el circuito de un comparador de magnitudes, para palabras binarias de cuatro bits, basado en un restador en complemento a dos.
- 7-3) Diseñar un circuito semisumador empleando sólo puertas NAND de dos entradas.
- 7-4) Diseñar un circuito sumador completo con sólo dos niveles de puertas.
- 7-5) Diseñar el circuito de un sumador para datos codificados en BCD exceso 3.
- 7-6) Diseñar el circuito convertidor de BCD exceso 3 a BCD natural empleando un sumador completo de cuatro bits del tipo 7483.
- 7-7) Diseñar el circuito convertidor de BCD natural a código *Gray* utilizando un sumador de cuatro bits del tipo 7483.

- 7-8) Diseñar el circuito convertidor de BCD *Aiken* al código BCD natural utilizando sólo sumadores de cuatro bits del tipo 7483.
- 7-9) Describir en VHDL y simular mediante el módulo *Simulate Demo* del programa *OrCAD Demo v9*, las funciones de la ALU 74181 representadas en la Tabla 7.9. Idem para la ALU 74381.
- 7-10) Describir en VHDL y simular mediante el módulo *Simulate Demo* del programa *OrCAD Demo v9*, el circuito del apartado 7.3.3 Circuito sumador-restador en binario signo magnitud.

## CONVERTIDORES A/D Y D/A

---

---

**Objetivos:**

- Comprender el principio de funcionamiento de los distintos convertidores A/D y D/A existentes. Ventajas e inconvenientes.
- Conocer las especificaciones típicas de los convertidores A/D y D/A.
- Saber analizar y diseñar circuitos convertidores A/D y D/A.

**Contenido:** En este capítulo se exponen los distintos procedimientos para la conversión A/D y D/A, sus especificaciones básicas y los procedimientos de análisis.

**Simulación:** Se utilizarán los programas de simulación *Electronics WorkBench 5.0* y *OrCAD Demo v9* para la realización de convertidores A/D y D/A, así como para la comprobación de ejercicios resueltos.

## 8.1 CONVERTIDORES A/D Y D/A. INTRODUCCIÓN



Los convertidores Analógico-Digital (A/D) y Digital-Analógico (D/A) representan los elementos de enlace entre los sistemas analógicos y los sistemas digitales.

El notable desarrollo experimentado en los sistemas digitales, en especial los sistemas basados en microprocesador, ha impulsado un cambio en las técnicas del tratamiento de la información, sustituyendo paulatinamente los sistemas analógicos por nuevos sistemas digitales. Esto se debe fundamentalmente a la mejora de prestaciones obtenida sobre la base de un coste más reducido, aumentando sustancialmente la potencia de cálculo, la fiabilidad en los sistemas e introduciendo mejoras en la facilidad y flexibilidad en el diseño. La información digital puede ser procesada, transmitida y almacenada con gran facilidad de forma indestructible y sin errores.

Sin embargo, el mundo físico está íntimamente ligado a las magnitudes analógicas con valores continuos, mientras que los sistemas digitales emplean magnitudes con valores discretos, razón por la cual se requieren sistemas que permitan llevar a cabo la transformación entre ambas magnitudes.

En la Figura 8.1 se muestra el diagrama de bloques general para el tratamiento digital de señales analógicas proporcionando una variable de salida igualmente analógica.

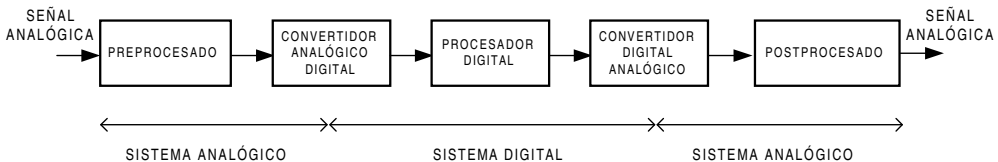


Figura 8.1. Diagrama de bloques de un sistema de procesamiento digital

**Preprocesado.** Antes de realizar el procesamiento digital de una información analógica se requiere un acondicionamiento de la señal eliminando el ruido y adaptando el rango de amplitudes. Este bloque estará basado en circuitos preamplificadores, filtros, fijadores de nivel, etc.

**Convertidor A/D.** Este circuito proporciona un valor codificado digitalmente en función de la amplitud de la señal analógica.

**Procesado digital.** Mediante este bloque se proporciona la capacidad de cálculo para obtener la funcionalidad deseada. Está basado en un sistema microprocesador ( $\mu\text{P}$ ), microcontrolador ( $\mu\text{C}$ ), procesador digital de señal (DSP), etc.

**Convertidor D/A.** Este circuito realiza la conversión de la información digital en una variable de magnitud analógica cuya amplitud es función directa de los códigos digitales aplicados a la entrada del convertidor.

**Postprocesado.** Formado básicamente por un filtro paso-bajo, su función es la de suavizar los cambios bruscos de nivel producidos por la secuencia de códigos digitales aplicados al convertidor D/A compuesto por valores discretos.

## 8.2 CONVERTIDOR D/A

### 8.2.1 Convertidor D/A. Generalidades

Este capítulo comienza con el estudio de los convertidores digitales analógicos (DAC) por ser circuitos más sencillos desde el punto de vista conceptual y parte integrante de algunos tipos de convertidores analógicos digitales (ADC).

Un **convertidor digital analógico (DAC)** es un dispositivo capaz de proporcionar una señal analógica, en forma de corriente o de tensión, proporcional al código digital aplicado a su entrada, según se muestra en la expresión [8.1].

$$V_s(I_s) = K D \quad [8.1]$$

siendo:

- $K$  una constante de proporcionalidad y
- $D$  el valor equivalente decimal del código digital.

En la Figura 8.2 se muestra, de forma gráfica, la función de transferencia de un convertidor analógico digital de 3 bits.

La entrada digital es una variable discreta que proporciona un conjunto de niveles discretos en la variable de salida.

La estructura general de un DAC es la que se muestra en la Figura 8.3, donde se observa una fuente de referencia, una red resistiva que entrega una corriente proporcional al código digital aplicado y los bloques necesarios para aislar y adaptar el sistema a los distintos tipos de tecnologías digitales.

Como puede observarse en dicha figura, en este tipo de sistemas es posible obtener la magnitud analógica de salida en forma de tensión o de corriente; con ello, el diseñador podrá seleccionar la que resulte más aconsejable en cada situación, dependiendo de las necesidades del diseño.

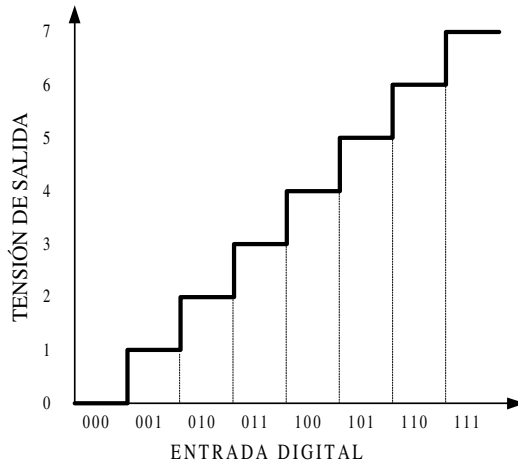


Figura 8.2. Función de transferencia de un DAC de 3 bits

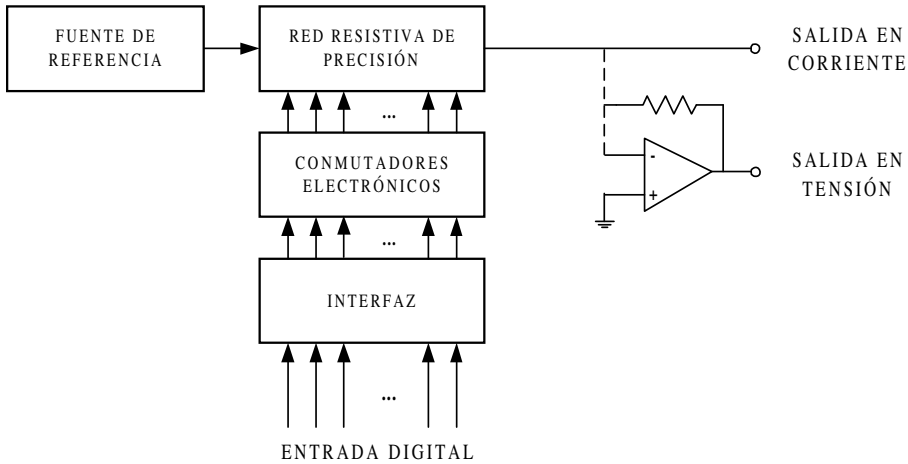


Figura 8.3. Diagrama de bloques de un DAC

### 8.2.2 Especificaciones de los convertidores D/A

**Resolución.** Representa la mínima variación incremental posible en la magnitud de salida. De este modo un convertidor de  $n$  bits resolverá en una parte de  $2^n$  partes. Comercialmente, existen DACs de 8, 10, 12, etc. bits, cuya resolución depende del margen dinámico de la magnitud de salida.

Un DAC de  $n$  bits para  $2^n$  códigos ( $0, \dots, 2^n - 1$ ) proporciona  $2^n$  niveles de la variable de salida dividida en  $2^n - 1$  intervalos.

Sean  $V_2$  y  $V_1$  dos tensiones analógicas de tal modo que  $V_2 > V_1$ . A partir de la Figura 8.4 se define *FS como rango a fondo de escala* de la variable de salida, un valor que define el margen de salida de la función de transferencia de un DAC.

La expresión [8.2] representa la resolución de un convertidor D/A, que como puede observarse es función del rango a fondo de escala *FS*.

$$Resolución = a = \frac{V(I)}{2^n} = \frac{V_2 - V_1}{2^n} = \frac{FS}{2^n} \tag{8.2}$$

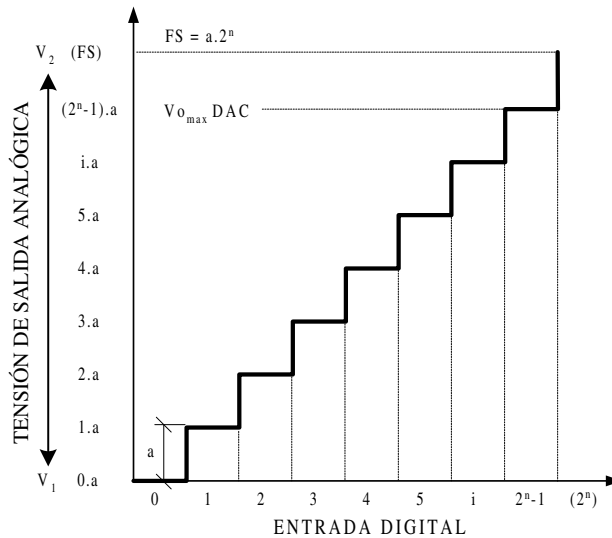


Figura 8.4. Función de transferencia detallada de un DAC de  $n$  bits

En los sistemas cuyo rango a fondo de escala *FS* varía entre cero y  $V_2$ , la resolución se corresponde con la variación incremental asociada al bit de menos peso (*LSB: Least Significant Bit*), según se muestra en la expresión [8.3].

$$Resolución = a = \frac{V_2}{2^n} = \frac{FS}{2^n} = LSB \tag{8.3}$$

En los sistemas cuyo rango a fondo de escala *FS* varía entre  $+V_2$  y  $-V_2$ , la resolución se corresponde con la expresión [8.4].

$$\text{Resolución} = a = \frac{V_2 - (-V_2)}{2^n} = \frac{2 \cdot FS}{2^n} \quad [8.4]$$

Definiendo la **constante de proporcionalidad**  $K = a$  según la expresión [8.1], la magnitud de salida varía a intervalos regulares múltiplos enteros de  $a$ , es decir,  $0, a, 2 \cdot a, \dots, (2^n - 2) \cdot a, (2^n - 1) \cdot a$ , alcanzando su valor máximo de salida  $V_{omax}$  de valor  $(2^n - 1) \cdot a$  para el mayor código posible  $(2^n - 1)$ . Por tanto también es posible definir la resolución como:

$$\text{Resolución} = a = \frac{V_{omax}}{2^n - 1} \quad [8.5]$$

La variable  $V_{omax}$  representa la máxima tensión de salida posible del convertidor DAC.

$$V_{omax} = a \cdot (2^n - 1) = FS - a \quad [8.6]$$

Debe observarse que la diferencia entre fondo de escala  $FS$  y  $V_{omax}$  es siempre igual al valor de la resolución  $a$ , tal y como se representa en la Figura 8.4.

**Precisión.** Es la desviación, tomada en el peor caso, de la magnitud de salida respecto al valor teórico esperado. Esta especificación incluye los errores de *offset*, ganancia, linealidad, ruido, etc., por tanto, es un parámetro que define el grado de precisión o imprecisión en el establecimiento de un valor analógico de salida. Generalmente se suele expresar como % de FS o % de LSB.

**Tiempo de conversión o establecimiento (*setting time*).** Es el tiempo que transcurre desde que se aplica un código estable a la entrada del convertidor hasta que la magnitud analógica de salida alcanza su valor estable. Es lógico suponer que este tiempo dependerá del incremento en la magnitud de salida, por ello este parámetro se especifica en la peor condición posible, la que implica un incremento desde su valor mínimo hasta un valor próximo a FS, expresado en % de FS.

**Sensibilidad frente a las variaciones de temperatura.** Representa la variación de la magnitud de *offset*, ganancia o linealidad frente a los incrementos de temperatura ambiente. La magnitud de *offset* suele expresarse en  $\mu V/^\circ C$  ( $\mu A/^\circ C$ ) o en partes por millón por grado centígrado (ppm/ $^\circ C$ ). La variación de la ganancia se suele expresar en ppm/ $^\circ C$  y la variación en la linealidad en partes por millón por grado centígrado aplicado sobre FS (ppm/ $^\circ C$  FS).

**Sensibilidad frente a las variaciones de la tensión de alimentación.** Representa la variación de la magnitud de *offset*, ganancia o linealidad frente a los incrementos de tensión de la fuente de alimentación. Al igual que la sensibilidad frente a las variaciones de temperatura, las distintas magnitudes afectadas se expresan en  $\mu V/V$ , ppm/V y ppm/V FS.



**Error de Offset.** Es el valor de la magnitud de salida que hace que la función de transferencia no pase por el origen. El dato indicado por el fabricante será el correspondiente al código de entrada (00...00) expresado en milivoltios (mV), nanoamperios (nA) o en un porcentaje del fondo de escala (% de FS). Normalmente este error es debido a las tensiones o corrientes de *offset* del dispositivo de salida y con frecuencia puede ser ajustado con potenciómetros externos.

**Error de ganancia o escala.** Representa la diferencia en las pendientes de las funciones de transferencia real e ideal. El fabricante proporciona este dato como la diferencia entre ambas rectas a fondo de escala expresado en % sobre FS, como se muestra en la parte izquierda de la Figura 8.5. Este error se puede corregir de forma externa.

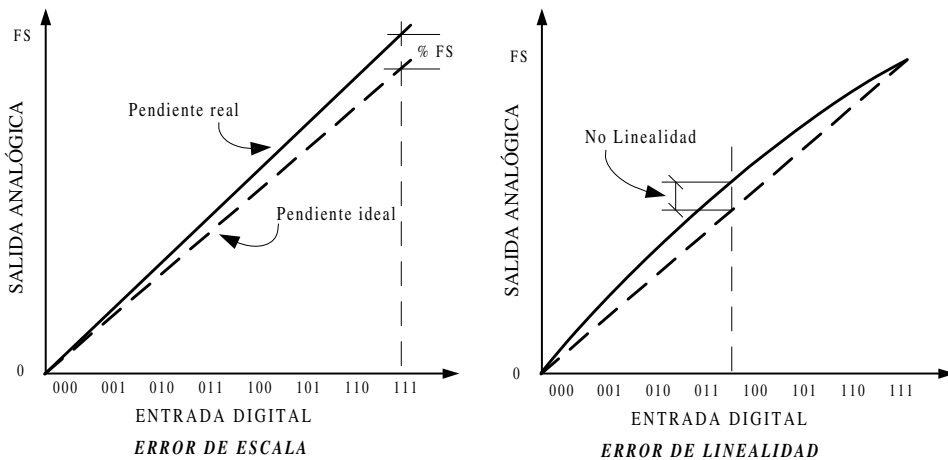


Figura 8.5. Representación de los errores de Escala y Linealidad

**Error de linealidad.** En realidad es más bien un parámetro que describe la no-linealidad y representa la máxima desviación de la función de transferencia real respecto de la teórica (una línea recta), como se muestra en la parte derecha de la Figura 8.5. Este valor generalmente se expresa como % de LSB puesto que en ningún caso deberá ser superior en valor absoluto a  $\frac{1}{2}$  LSB para que se cumpla la condición de monotonicidad en la función de salida.

### 8.2.3 Circuitos convertidores D/A

La diferencia fundamental entre los distintos tipos de convertidores reside en el tipo de red utilizada y la fuente de referencia. Básicamente existen dos tipos de redes a emplear en los DACs: las redes ponderadas al peso binario de cada bit y las redes en escalera del tipo R-2R.

### 8.2.3.1 CONVERTIDORES D/A CON RESISTENCIAS PONDERADAS

En la Figura 8.6 se muestra un convertidor de este tipo donde se puede apreciar la fuente de referencia de tensión  $V_{ref}$ , la red de conmutadores electrónicos y el conjunto de resistencias ponderadas con un valor proporcional al peso de cada bit que le corresponde a las entradas digitales  $S_0, \dots, S_{n-1}$ . Finalmente, un sumador basado en un amplificador operacional convierte la magnitud de corriente en tensión de salida.

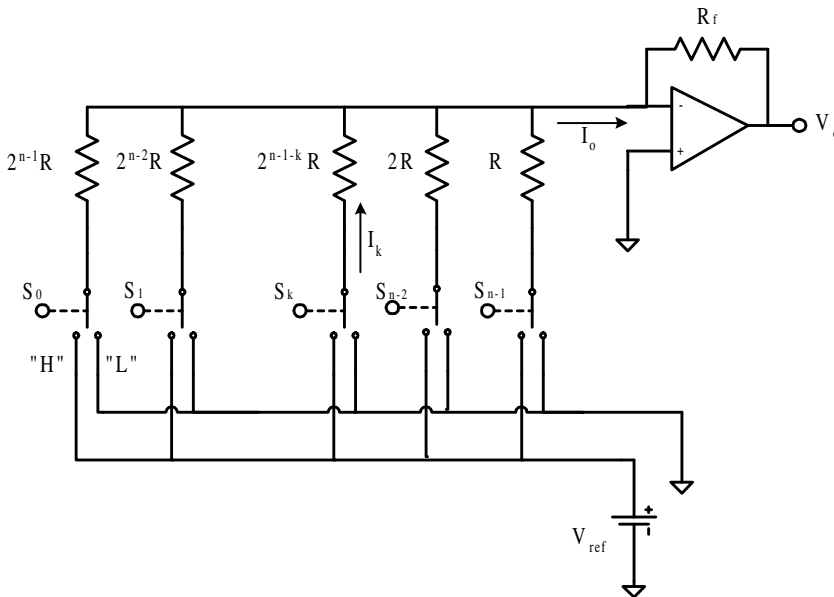


Figura 8.6. Esquema de un DAC basado en red de resistencias ponderadas

Los conmutadores  $S_0, \dots, S_{n-1}$  conectan cada resistencia ponderada a  $V_{ref}$  o masa dependiendo de que la entrada digital sea “0” o “1” lógico. Las entradas binarias que sean “0” no aportan ninguna corriente a la corriente total  $I_o$ , mientras que las entradas que estén a “1” aportan una corriente dependiente de su peso,

$$I_k = S_k \frac{V_{ref}}{2^{n-1-k} \cdot R} \quad [8.7]$$

donde  $n$  es igual al número de bits del convertidor y  $k$  es la posición del bit de entrada, pudiendo variar entre  $0 \leq k \leq n-1$ .

Sumando todas las corrientes aportadas por cada bit de información digital se obtiene la expresión:

$$I_o = \sum_{k=0}^{k=n-1} I_k \quad [8.8]$$

$$I_o = \frac{V_{ref}}{R} \left( S_{n-1} + S_{n-2} \frac{1}{2} + S_{n-3} \frac{1}{4} + \dots + S_2 \frac{1}{2^{n-3}} + S_1 \frac{1}{2^{n-2}} + S_0 \frac{1}{2^{n-1}} \right)$$

De igual modo se puede obtener la tensión de salida  $V_o$ :

$$V_o = -I_o \cdot R_f \quad [8.9]$$

$$V_o = \frac{-V_{ref} \cdot R_f}{R} \left( S_{n-1} + S_{n-2} \frac{1}{2} + S_{n-3} \frac{1}{4} + \dots + S_2 \frac{1}{2^{n-3}} + S_1 \frac{1}{2^{n-2}} + S_0 \frac{1}{2^{n-1}} \right) \quad [8.10]$$

$$V_o = \frac{-V_{ref} \cdot R_f}{2^{n-1} R} [D]$$

$$a = LSB = \frac{-V_{ref} \cdot R_f}{2^{n-1} R} \quad [8.11]$$

Para el valor mínimo  $V_{omin}$ , donde  $S_{n-1} = S_{n-2} = \dots = S_1 = S_0 = 0$ , el margen dinámico de la salida del convertidor será:

$$V_{omin} = 0 \quad [8.12]$$

Para el valor máximo de  $V_{omax}$ , donde  $S_{n-1} = S_{n-2} = \dots = S_1 = S_0 = 1$ , el margen dinámico de la salida del convertidor será:

$$V_{omax} = \frac{-V_{ref} \cdot R_f}{R} \left( \frac{2^n - 1}{2^{n-1}} \right) \quad [8.13]$$

$$V_{omax} = a \cdot (2^n - 1)$$

Estos convertidores tienen la ventaja de la simplicidad, pero tienen varios inconvenientes a considerar, entre los que cabe destacar los dos que se indican a continuación:

- No es posible ajustar correctamente las corrientes de *offset* de los terminales de entrada del amplificador operacional al no disponer la red resistiva de una impedancia constante.

- En la práctica resulta complicado encontrar resistencias precisas con los valores requeridos:  $R$ ,  $2R$ ,  $4R$ ,  $8R$ , etc.

### PROBLEMA RESUELTO 8-1



Simular el funcionamiento del DAC de 8 bits de resistencias ponderadas representado en la Figura 8.6, utilizando para ello el programa *Electronics Workbench*.

Calcular el valor de los parámetros de resolución  $a$ , fondo de escala  $FS$  y tensión máxima de salida del convertidor  $V_{omax}$ .

#### Datos para el cálculo:

$$V_{ref} = 10 \text{ V}; R = 10 \text{ k}\Omega \text{ y } R_f = 5 \text{ k}\Omega.$$

#### SOLUCIÓN:

La resolución representa el mínimo valor incremental de la magnitud de salida, y queda determinado por la expresión [8.11]:

$$LSB = a = \frac{-V_{ref} \cdot R_f}{2^{n-1} R} = -39,06 \text{ mV}$$

El cálculo de  $FS$  se determina a partir de [8.3] y  $V_{omax}$  a partir de [8.6]:

$$FS = a \cdot 2^n = -39,06 \cdot 10^{-3} \cdot 256 = -10 \text{ V}$$

$$V_{omax} = FS - a = 10 - 39,06 \cdot 10^{-3} = -9,96 \text{ V}$$

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap08\Ewb5\08W0\_\_01.ewb**

Mediante el empleo de los conmutadores asociados a las teclas [0], [1], ... , [6] y [7], que representan la posición de cada bit, aplicar las palabras binarias comprobando el valor obtenido en la tensión de salida  $V_o$ .

- Entrada = 00000001 para obtener el valor  $LSB = a$ .
- Entrada = 11111111 para obtener el valor  $V_{omax}$ .
- Introducir otras combinaciones y comprobar el valor obtenido en  $V_o$ .

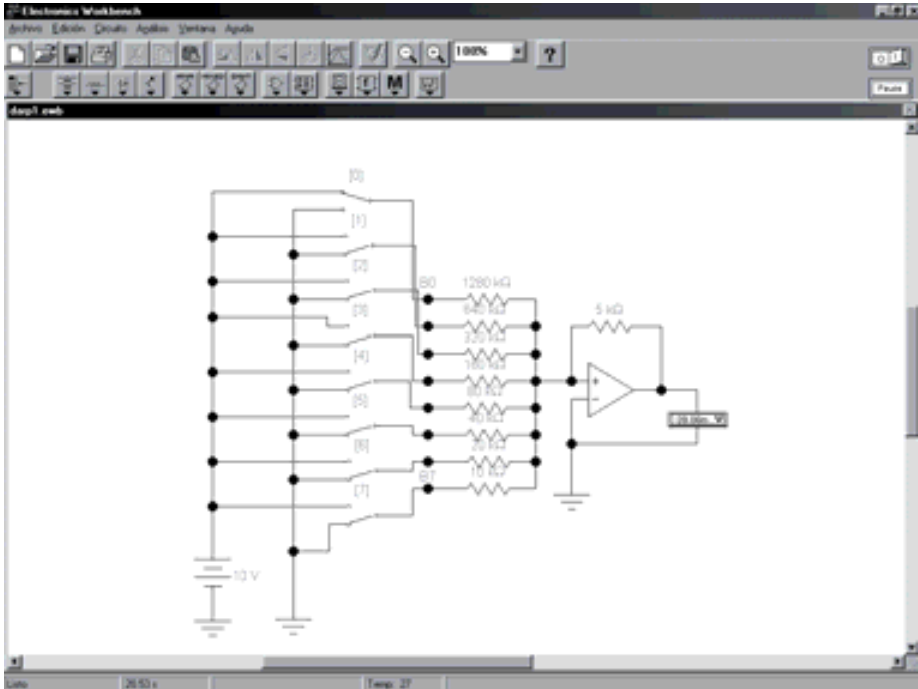


Figura 8.7. Circuito para simular el funcionamiento de un DAC basado en una red de resistencias ponderadas

### 8.2.3.2 CONVERTIDORES D/A CON RED R-2R

La **red resistiva en escalera R-2R** resuelve los problemas mencionados anteriormente para los DAC con resistencia ponderada, presentando dos únicos valores resistivos ( $R-2R$ ) y una impedancia de entrada y salida constante, lo que posibilita un correcto ajuste de las corrientes de *offset* del operacional. En la Figura 8.8 se muestra el esquema de un convertidor digital analógico basado en una red R-2R.

Para comprender el funcionamiento de la red  $R-2R$ , se deben tener en cuenta algunas propiedades de esta red basadas en su simetría. Si se considera un nudo cualquiera de corrientes de la parte superior de la red, por ejemplo el nudo de  $b$  de la Figura 8.9, se puede observar que la impedancia equivalente sobre este punto, vista sobre cualquiera de las direcciones, toma siempre el valor de  $R_o = (2R//2R)+R = 2R$ .

De igual modo se puede observar que la impedancia de salida de la red es de valor constante  $R_s = 3R$  con independencia del estado de los conmutadores electrónicos, como se muestra en la Figura 8.10.

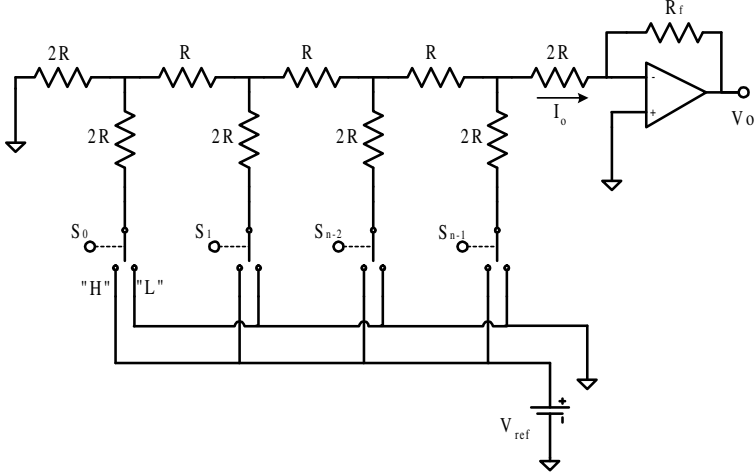


Figura 8.8. Esquema de un DAC basado en la red R-2R

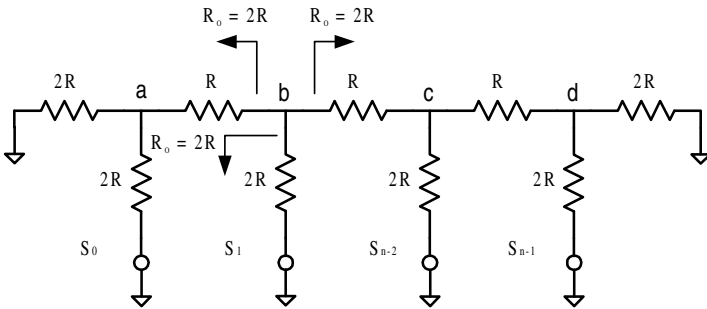


Figura 8.9. Impedancia vista desde cada nudo en una red R-2R

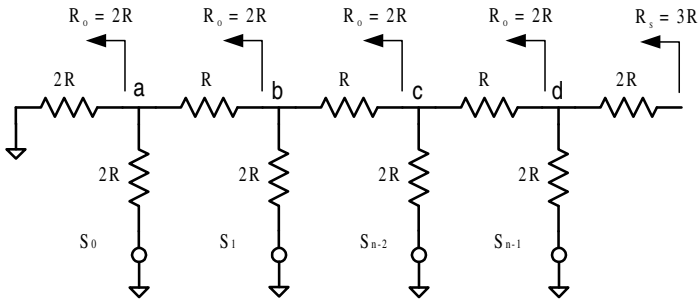


Figura 8.10. Impedancia de salida de la red R-2R

La impedancia presente en las entradas de los conmutadores es constante y de valor  $R_i = 3R$ , como se muestra en la Figura 8.11.

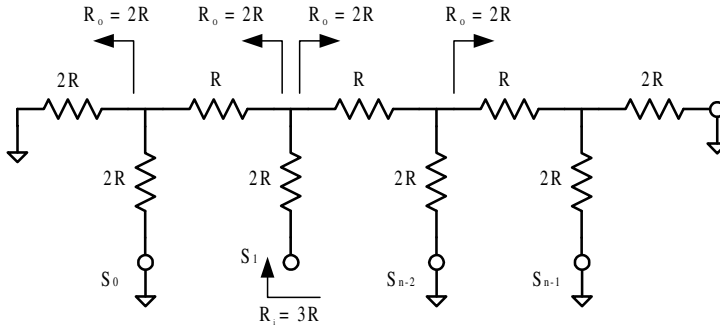


Figura 8.11. Impedancia de entrada de la red R-2R

Debido a la simetría de impedancias, el principio de funcionamiento de la red R-2R es el de ir dividiendo sucesivamente entre dos la corriente que llega a cada nudo. De este modo, la corriente aportada por el terminal  $S_{n-2}$  será la mitad de la aportada por el terminal  $S_{n-1}$  sobre la corriente total  $I_o$ , como puede observarse en la Figura 8.12 que se muestra a continuación.

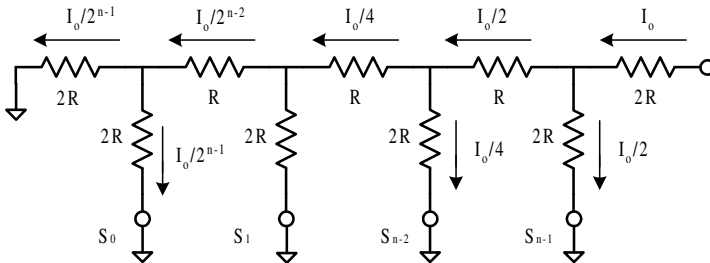


Figura 8.12. División de corrientes en una red R-2R

Aplicando estas observaciones sobre el circuito convertidor A/D basado en la red R-2R y considerando los efectos individuales producidos al aplicar la tensión de referencia  $V_{ref}$  sobre cada una de las entradas lógicas, se deduce que cuando se aplica un nivel lógico alto a la entrada  $S_{n-1}$ , representada en la Figura 8.13, la corriente de entrada sobre este terminal contribuye a una corriente  $I_o$  en una cantidad igual a la mitad de su valor.

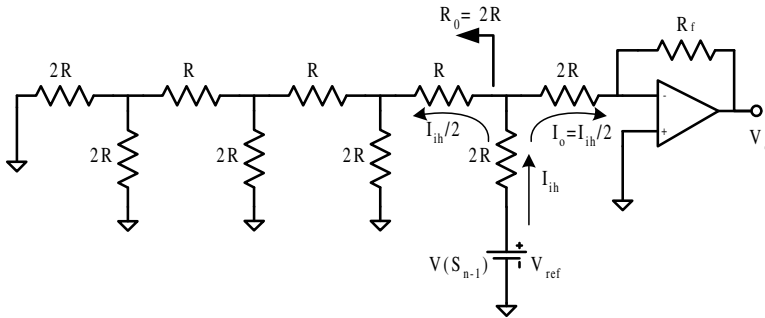


Figura 8.13. Efecto de  $V(S_{n-1})$  sobre  $I_o$

De igual modo la corriente de entrada aplicada sobre el terminal  $S_{n-2}$  (Figura 8.14), contribuye sobre la corriente total con un valor igual a una cuarta parte de su valor y así sucesivamente siguiendo una relación inversa de potencias de dos.

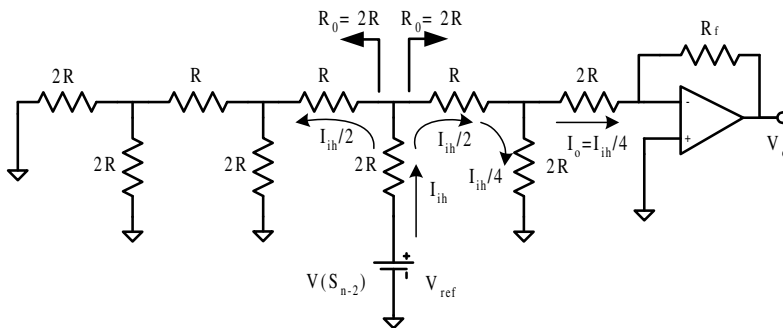


Figura 8.14. Efecto de  $V(S_{n-2})$  sobre  $I_o$

Puesto que la impedancia de entrada en cualquier terminal es constante, la corriente de entrada será también constante y de valor:

$$I_{iH} = \frac{V_{ref}}{3R} \tag{8.14}$$

Aplicando el **teorema de superposición de efectos** sobre la corriente total, se obtiene:

$$I_o = \frac{V_{ref}}{3R} \left( S_{n-1} \frac{1}{2} + S_{n-2} \frac{1}{4} + S_{n-3} \frac{1}{8} + \dots + S_2 \frac{1}{2^{n-2}} + S_1 \frac{1}{2^{n-1}} + S_0 \frac{1}{2^n} \right) \tag{8.15}$$



donde,  $n$  es el número de bits del convertidor.

De igual modo se puede obtener la tensión de salida  $V_o$ :

$$V_o = -I_o \cdot R_f \quad [8.16]$$

$$V_o = \frac{-V_{ref} \cdot R_f}{3R} \left( S_{n-1} \frac{1}{2} + S_{n-2} \frac{1}{4} + S_{n-3} \frac{1}{8} + \dots + S_1 \frac{1}{2^{n-1}} + S_0 \frac{1}{2^n} \right) \quad [8.17]$$

$$V_o = \frac{-V_{ref} \cdot R_f}{2^n \cdot 3R} [D]$$

$$a = LSB = \frac{-V_{ref} \cdot R_f}{2^n \cdot 3R} \quad [8.18]$$

El valor mínimo de salida  $V_{omin}$ , cuando  $S_{n-1} = S_{n-2} = \dots = S_1 = S_0 = 0$ , es:

$$V_{omin} = 0 \quad [8.19]$$

El valor máximo de salida  $V_{omax}$ , cuando  $S_{n-1} = S_{n-2} = \dots = S_1 = S_0 = 1$ , es:

$$V_{omax} = \frac{-V_{ref} \cdot R_f}{3R} \left( \frac{2^n - 1}{2^n} \right) \quad [8.20]$$

$$V_{omax} = a(2^n - 1)$$

## PROBLEMA RESUELTO 8-2



Simular el funcionamiento de la red en escalera  $R$ - $2R$  representada en la Figura 8.8, visualizando la división de corrientes ponderadas en cada bit. Utilizar para ello el programa de simulación *Electronics Workbench*.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap08\Ewb5\08W0\_02.ewb**

En la Figura 8.15 se muestra el circuito que se ha diseñado en *Electronics Workbench* para simular el funcionamiento del reparto de corrientes de la red en escalera  $R$ - $2R$ .

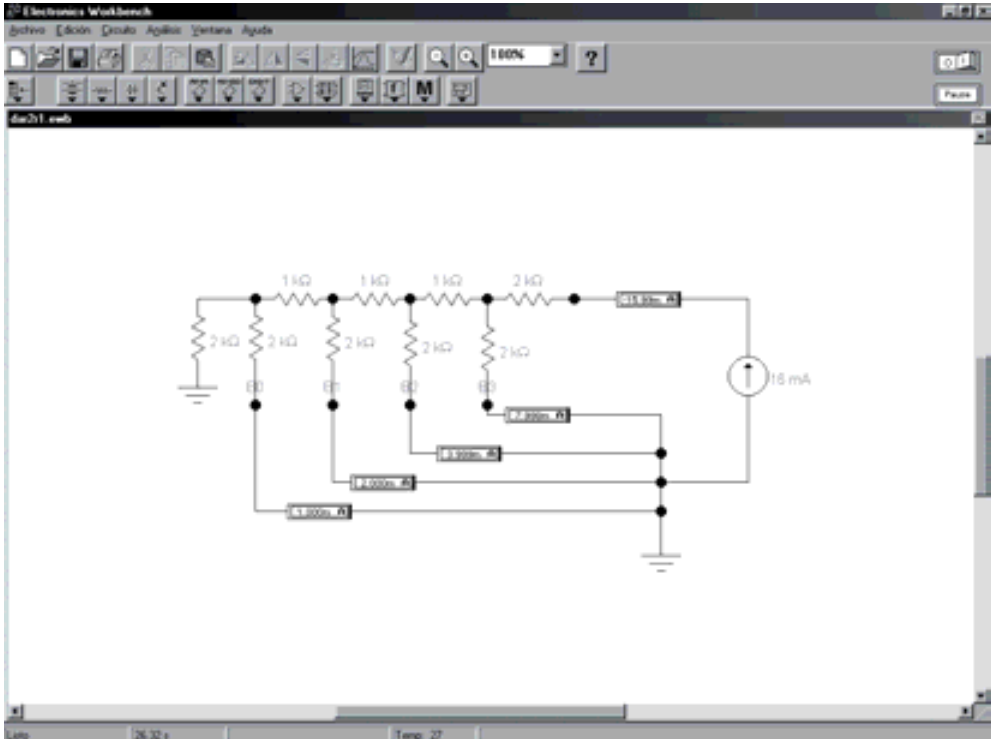


Figura 8.15. Circuito para simular el funcionamiento del reparto de corrientes en una red R-2R

### PROBLEMA RESUELTO 8-3



Simular el funcionamiento de un DAC de 8 bits basado en la red R-2R representado en la Figura 8.6, utilizando para ello el programa *Electronics Workbench*.

Calcular el valor de la resistencia  $R_f$  para obtener la tensión de salida máxima del convertidor  $V_{omax} = 5V$ . Obtener el parámetro de resolución  $a$ .

#### Datos para el cálculo:

$$V_{ref} = -10 V; R = 10 k\Omega.$$

#### Solución:

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap08\Ewb5\08W0\_\_03.ewb**

$V_{omax}$  representa el valor máximo de la tensión de salida del DAC, que se produce ante la entrada binaria (11 ... 11). A partir de la ecuación [8.20] se tiene que:

$$V_{omax} = \frac{-V_{ref} \cdot R_f}{3R} \left( \frac{2^n - 1}{2^n} \right)$$

$$R_f = \frac{V_{omax} \cdot 3R}{-V_{ref}} \left( \frac{256}{255} \right) = 15,058 \text{ k}\Omega$$

El cálculo de la resolución se determina a partir de la expresión [8.5]:

$$\text{Resolución} = a = \frac{V_{omax}}{2^n - 1} = \frac{5}{255} = 19,60 \text{ mV}$$

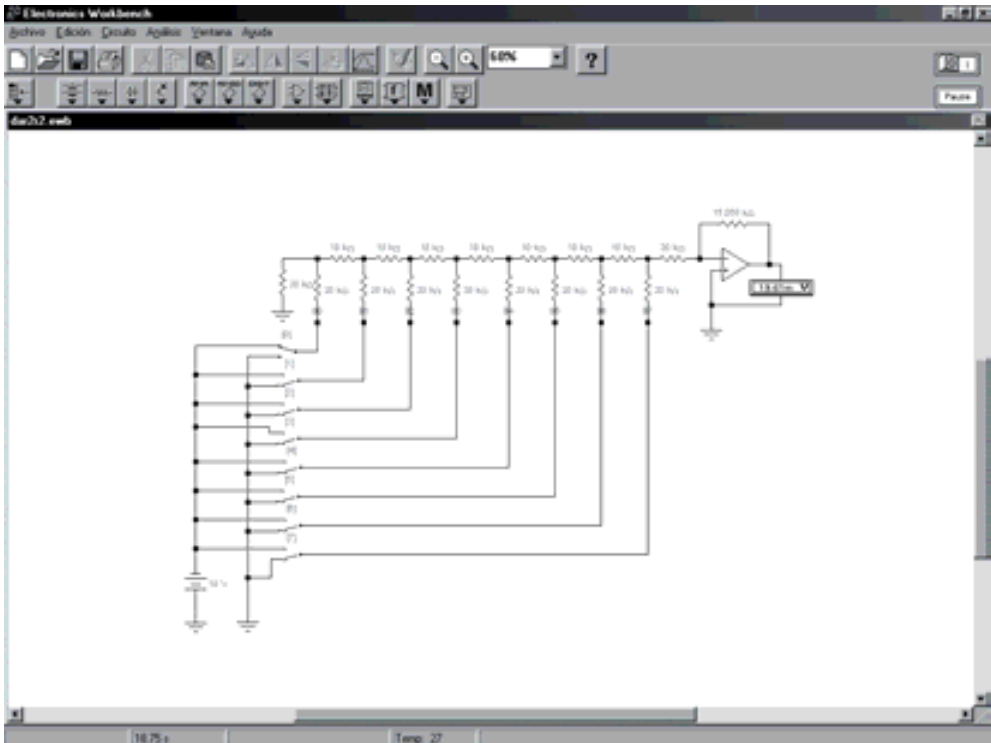


Figura 8.16. Circuito para simular el funcionamiento de un DAC unipolar basado en una red R-2R

Mediante el empleo de los conmutadores asociados a las teclas [0], [1], ... , [6] y [7], que representan la posición de cada bit, aplicar las palabras binarias comprobando el valor obtenido en la tensión de salida  $V_o$ .

- Entrada = 00000001 para obtener el valor  $LSB = a$ .
- Entrada = 11111111 para obtener el valor  $V_{o_{max}}$ .
- Introducir otras combinaciones y comprobar el valor obtenido de  $V_o$ .

## 8.2.4 Funcionamiento bipolar de los convertidores D/A

Los distintos DACs descritos hasta este momento proporcionan corrientes o tensiones de salida unipolares, sólo valores positivos o negativos, con un rango desde cero hasta fondo de escala  $FS$ .

Cuando se desea que la magnitud de salida cubra un margen dinámico desde  $+FS$  a  $-FS$ , funcionamiento **bipolar**, es necesario incorporar a los circuitos estudiados una modificación que permita un desplazamiento en el punto inicial de la función de transferencia,

$$V_s(I_s) = K \cdot D + C \quad [8.21]$$

donde,

- $K$  es una constante de proporcionalidad,
- $D$  es el valor equivalente decimal del código digital, y
- $C$  la constante que determina el valor inicial para el código cero.

Una forma de conseguir este desplazamiento consiste en agregar un valor constante ( $I_{offset}$ ) a la corriente  $I_o$  de tal modo que:

$$I_o' = I_o + I_{offset} \quad [8.22]$$

Tomando como ejemplo un DAC basado en una red R-2R, se obtiene a partir de la expresión [8.17]:

$$V_o = \frac{-V_{ref} \cdot R_f}{3R \cdot 2^n} (D) - \frac{V_{offset} \cdot R_f}{R_{offset}} \quad [8.23]$$

donde,  $D$  representa el valor equivalente decimal del código aplicado.

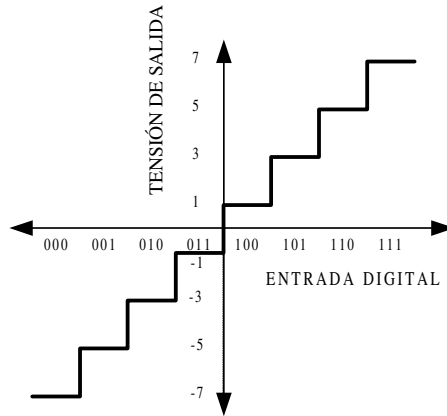


Figura 8.17. Función de transferencia de un DAC bipolar

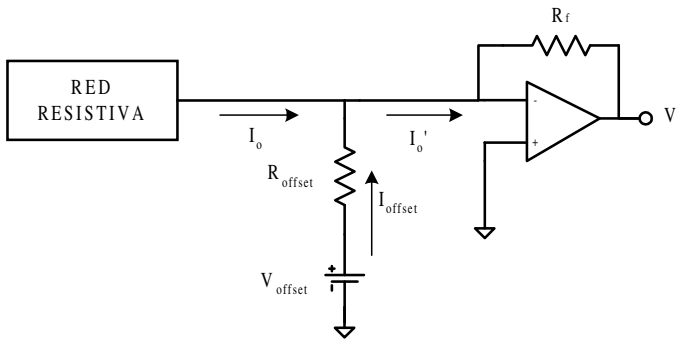


Figura 8.18. Circuito de offset

El valor mínimo de salida  $V_{o_{min}}$  cuando el equivalente decimal  $D = 0$ , es:

$$V_{o_{min}} = \frac{-V_{offset} R_f}{R_{offset}} \tag{8.24}$$

El valor máximo de salida  $V_{o_{max}}$ , cuando el equivalente decimal  $D = 2^n - 1$ , es:

$$V_{o_{max}} = \frac{-V_{ref} R_f}{3R 2^n} (2^n - 1) - \frac{V_{offset} R_f}{R_{offset}} \tag{8.25}$$

La simetría de  $\pm FS$  sólo se consigue si se cumple que para el valor decimal del código  $D = 2^n/2 = 2^{n-1}$  el resultado de  $V_o = 0$ .

Los valores de  $V_{offset}$  y  $R_{offset}$  que posibilitan esta simetría deben cumplir esta condición y por tanto, partiendo de [8.23] se obtiene:

$$0 = \frac{-V_{ref} R_f}{3R 2^n} \left( \frac{2^n}{2} \right) - \frac{V_{offset} R_f}{R_{offset}} \quad [8.26]$$

$$\frac{V_{ref}}{6R} = \frac{-V_{offset}}{R_{offset}}$$

### PROBLEMA RESUELTO 8-4



Simular el funcionamiento de un DAC bipolar de 8 bits basado en la red R-2R, representado en la Figura 8.6, utilizando para ello el programa *Electronics Workbench*.

- Calcular el valor de  $V_{ref}$  para obtener un voltaje de salida unipolar de valor  $V_{o_{max}} = 10 \text{ V}$ .
- Calcular el circuitos de *offset* (de la Figura 8.18) para obtener un voltaje de salida bipolar  $V_{o_{max}} = \pm 5 \text{ V}$ .

#### Datos para el cálculo:

$R = 10 \text{ k}\Omega$ ,  $R_f = 30 \text{ k}\Omega$  y  $V_{offset} = +2,5 \text{ V}$ .

#### Solución:

- En primer lugar se debe observar que la tensión  $V_{offset}$  ha de ser de polaridad opuesta a  $V_{ref}$ .

Considerando que el rango de salida  $V_o$  es 0/10 V, se obtiene el valor de  $V_{ref}$  a partir de la expresión [8.20].

$$V_{o_{max}} = \frac{-V_{ref} R_f}{3R} \left( \frac{255}{256} \right)$$

$$V_{ref} = \frac{-V_{o_{max}} 3R}{R_f} \left( \frac{256}{255} \right) = 10,04 \text{ V}$$

- Se debe observar que trasladar el rango de salida de 0/10 V a +5/-5 V es equivalente a introducir un decalaje en la función de transferencia de  $-5 \text{ V}$ , como constante de *offset*.

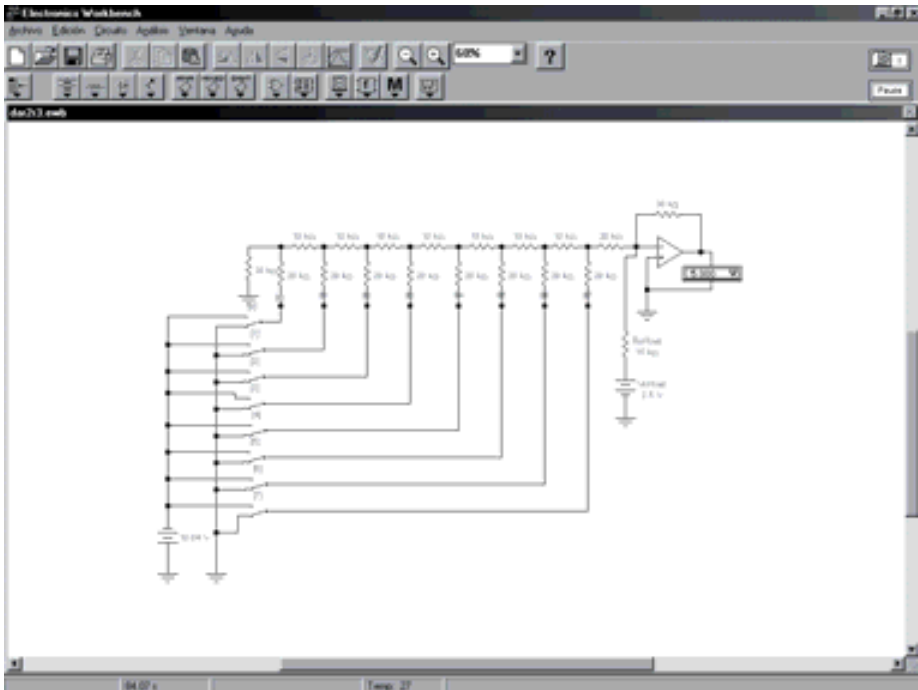
Para que se cumpla la condición de simetría en el rango bipolar, es preciso que se aplique la razón expuesta en [8.26].

$$\frac{V_{ref}}{6R} = \frac{-V_{offset}}{R_{offset}}$$

$$R_{offset} = \frac{-V_{offset}}{V_{ref}} 6R = 15k\Omega$$

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap08\Ewb5\08W0\_\_04.ewb**



*Figura 8.19. Circuito de simulación de un DAC bipolar basado en red R-2R*

Mediante el empleo de los conmutadores [0], [1], ... , [6] y [7], que representan la posición de cada bit, aplicar las palabras binarias y comprobar el valor de la salida  $V_o$ .

- Entrada = 0000000 para obtener el valor  $V_{omin}$ .
- Entrada = 1111111 para obtener el valor  $V_{omax}$ .
- Introducir otras combinaciones y comprobar el valor obtenido de  $V_o$ .

## 8.2.5 El convertidor comercial DAC 0800

En el CDRom#2 que acompaña a este libro, se incluye un **Data Sheets** de la empresa Fairchild Semiconductors, en el que se muestra la estructura interna del convertidor DAC0800. Este convertidor se corresponde con un tipo de convertidor de 8 bits en escalera  $R-2R$ , que proporciona una salida en forma de corriente  $I_o$ .

El circuito está formado por un conjunto de transistores cuya corriente de colector se encuentra ponderada debido a la red en escalera  $R-2R$  conectada a los emisores. En la Figura 8.20 se muestran los distintos terminales de conexión de este circuito integrado.

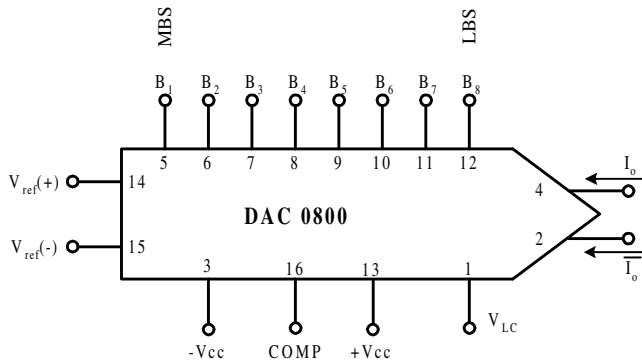


Figura 8.20. Circuito integrado DAC0800

La descripción de los distintos terminales que se han representado en el chip es la siguiente:

- $V_{ref}(+)$  y  $V_{ref}(-)$ . Terminales de tensión de referencia.
- $I_o$  e  $\bar{I}_o$ . Terminales de salida de corriente complementaria.
- $V_{lc}$ . Terminal de configuración de la interfaz lógica para la correcta adaptación de las distintas familias (TTL, CMOS, ECL, etc.).
- **COMP**. Terminal de compensación. El fabricante recomienda conectar un condensador de valor de  $0,1\mu\text{F}$  entre este terminal y el terminal identificado como  $-V_{cc}$ .
- **+Vcc** y **-Vcc**. Para el correcto funcionamiento del convertidor se requiere una alimentación simétrica  $+V_{cc}$  y  $-V_{cc}$  cuyos valores deben estar comprendidos entre  $\pm 4,5$  y  $\pm 18$  V.

Las corrientes de salida  $I_o$  e  $\bar{I}_o$  quedan determinadas por la expresión:



$$I_o = \frac{I_{ref}}{256} (2^7 B_1 + 2^6 B_2 + 2^5 B_3 + \dots + 2^1 B_7 + 2^0 B_8) = \frac{I_{ref}}{256} [D]$$

$$\bar{I}_o = \frac{I_{ref}}{256} (2^7 \bar{B}_1 + 2^6 \bar{B}_2 + 2^5 \bar{B}_3 + \dots + 2^1 \bar{B}_7 + 2^0 \bar{B}_8) = \frac{I_{ref}}{256} [\bar{D}]$$
[8.27]

donde,  $B_8 = LSB$  y  $B_1 = MSB$

Relacionando ambas corrientes, se obtiene:

$$I_o + \bar{I}_o = \frac{I_{ref}}{256} (D + \bar{D}) = I_{ref} \frac{255}{256}$$
[8.28]

La corriente de referencia  $I_{ref}$ , teniendo en cuenta el esquema que se muestra en la Figura 8.21, se determina a partir de la siguiente expresión:

$$I_{ref} = \frac{V_{ref}}{R_{ref}}$$
[8.29]

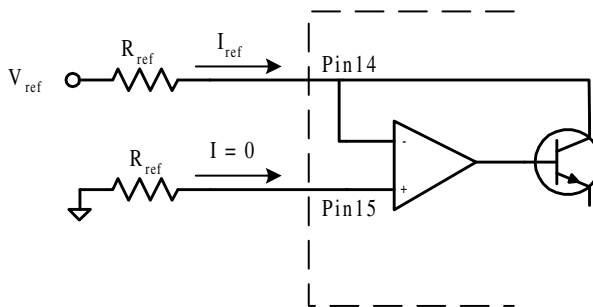


Figura 8.21. Circuito que determina la corriente de referencia  $I_{ref}$  del convertidor DAC0800

### 8.2.5.1 FUNCIONAMIENTO UNIPOLAR

Para obtener una salida en tensión en modo unipolar, es preciso un convertidor de corriente-tensión basado en un amplificador operacional, tal como se muestra en el esquema de la Figura 8.22.

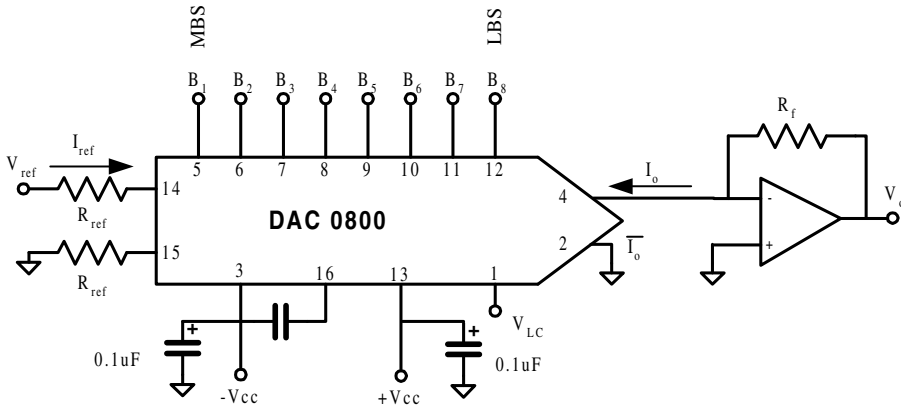


Figura 8.22. Convertidor DAC0800 en funcionamiento unipolar

En estas condiciones la tensión de salida  $V_o$  será:

$$V_o = -I_o R_f \quad [8.30]$$

$$V_o = \frac{I_{ref} \cdot R_f}{256} (2^7 B_1 + 2^6 B_2 + 2^5 B_3 + \dots + 2^1 B_7 + 2^0 B_8) \quad [8.31]$$

$$V_o = \frac{I_{ref} R_f}{256} [D]$$

Donde,  $B_8 = LSB$  y  $B_1 = MSB$ .

$$LSB = a = \frac{I_{ref} R_f}{256}$$

$$a = \frac{FS}{256} \quad [8.32]$$

$$FS = I_{ref} R_f$$

El valor mínimo de la tensión de salida  $V_{omin}$ , cuando  $B_1 = B_2 = \dots = B_7 = B_8 = 0$ , viene dado por la expresión [8.33].

$$V_{omin} = 0 \quad [8.33]$$

El valor máximo de la tensión de salida  $V_{omax}$ , cuando  $B_1 = B_2 = \dots = B_7 = B_8 = 1$ , viene dado por la expresión [8.34].

$$V_{omax} = I_{ref} R_f \frac{255}{256} \tag{8.34}$$

### 8.2.5.2 FUNCIONAMIENTO BIPOLAR

Para obtener una salida en tensión en modo bipolar, es preciso un convertidor de corriente-tensión cuyo rango dinámico varíe en  $\pm FS$ . En la Figura 8.23 se muestra el esquema de conexión para este modo de funcionamiento.

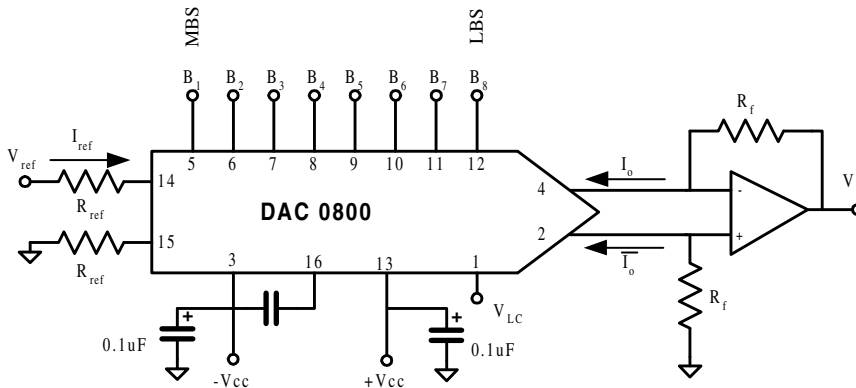


Figura 8.23. Convertidor DAC0800 en funcionamiento bipolar

En estas condiciones la tensión de salida  $V_o$  será:

$$V_o = I_o R_f - \bar{I}_o R_f = (I_o - \bar{I}_o) R_f \tag{8.35}$$

Teniendo en cuenta que:

$$I_o + \bar{I}_o = I_{ref} \frac{255}{256} \tag{8.36}$$

$$V_o = \left( 2I_o - \frac{255}{256} I_{ref} \right) R_f \quad [8.37]$$

Para obtener el rango dinámico de la tensión de salida, se sustituye en la expresión [8.37] el código aplicado al convertidor por las palabras binarias 00, ..., 00 y 11, ..., 11. Así, el valor mínimo de la tensión de salida  $V_{omin}$ , cuando  $B_1 = B_2 = \dots = B_7 = B_8 = 0$ , viene dado por la expresión [8.38].

$$V_{omin} = -I_{ref} R_f \frac{255}{256} \quad [8.38]$$

El valor máximo de la tensión de salida  $V_{omax}$ , cuando  $B_1 = B_2 = \dots = B_7 = B_8 = 1$ , viene dado por la expresión [8.39].

$$V_{omax} = \left( 2I_{ref} \frac{255}{256} - I_{ref} \frac{255}{256} \right) R_f = I_{ref} R_f \frac{255}{256} \quad [8.39]$$

### PROBLEMA RESUELTO 8-5



Simular el funcionamiento del convertidor DAC0800 unipolar de 8 bits basado en la Figura 8.22, utilizando para ello el programa *Electronics Workbench*.

- Calcular el valor de  $R_{ref}$  para obtener una corriente  $I_{ref} = 1$  mA.
- Calcular el valor de  $R_f$  para obtener un margen de salida  $V_{omax} = 5$  V.
- Calcular la resolución  $a$  y el rango a fondo de escala  $FS$ .
- Obtener la tabla de la función de transferencia del convertidor relacionando los códigos aplicados a la entrada y la tensión de salida  $V_o$  correspondientes a los códigos decimales: 0, 1, 2, ..., 128, ..., 253, 254 y 255.

#### Datos para el cálculo:

$$V_{ref} = +2.5 \text{ V.}$$

#### Solución:

- El cálculo de  $I_o$  se realiza a partir de la expresión [8.29]:

$$I_{ref} = \frac{V_{ref}}{R_{ref}}$$

$$R_{ref} = \frac{V_{ref}}{I_{ref}} = 2,5 \text{ k}\Omega$$

- b) Considerando que el rango de salida  $V_{omax}$  es 0-5 V y aplicando la expresión [8.34], se obtiene  $R_f$ .

$$V_{omax} = I_{ref} R_f \frac{255}{256}$$

$$R_f = \frac{V_{omax}}{I_{ref}} \frac{256}{255} = 5019 \Omega$$

- c) El cálculo de la resolución  $a$  y  $FS$  se realiza a partir de la expresión [8.32]:

$$LSB = a = \frac{I_{ref} R_f}{256} = \frac{1 \cdot 10^{-3} \cdot 5019}{256} = 19,605 \text{ mV}$$

$$FS = I_{ref} R_f = 1 \cdot 10^{-3} \cdot 5019 = 5,019 \text{ V}$$

- d) La función de salida de un DAC puede expresarse matemáticamente mediante la ecuación [8.21] (donde,  $K$  representa la *resolución*,  $D$  el valor equivalente decimal del código aplicado y  $C$  el valor inicial para el código decimal cero).

$$V_s = K D + C$$

donde,

$$C = 0 \text{ V}$$

$$K = LSB = a = 19,605 \cdot 10^{-3} \text{ V}$$

Por consiguiente, la tabla de la función de transferencia será la representada en la Tabla 8.1.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap08\Ewb5\08W0\_\_05.ewb**

Tabla 8.1. Función de transferencia del DAC0800 unipolar

Código Decimal	$V_o(V)$
0	0
1	0,0196
2	0,0393
...	...
127	2,4898
128	2,5094
...	...
253	4,9607
254	4,971
255	5

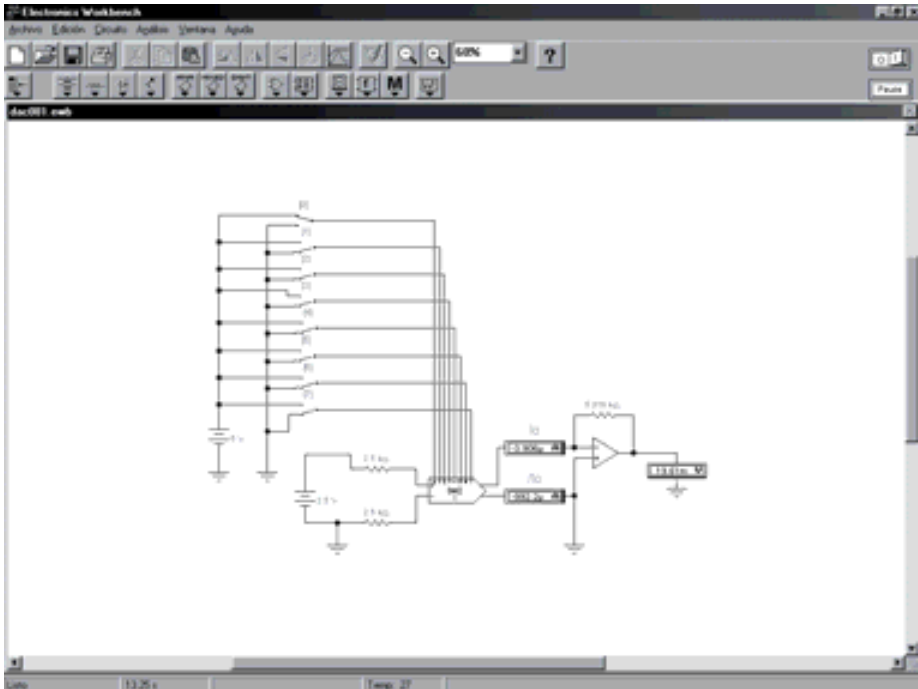


Figura 8.24. Circuito para simular el funcionamiento de un convertidor DAC0800 unipolar

Mediante el empleo de los conmutadores asociados a las teclas [0], [1], ... , [6] y [7], que representan la posición de cada bit, aplicar las palabras binarias siguientes comprobando el valor obtenido en la tensión de salida  $V_o$ .

- Entrada = 00000000 para obtener el valor  $V_{omin}$ .
- Entrada = 11111111 para obtener el valor  $V_{omax}$ .
- Introducir otras combinaciones y comprobar el valor obtenido de  $V_o$ , comparándolo con los valores de la Tabla 8.1.

### PROBLEMA RESUELTO 8-6



Simular el funcionamiento de un convertidor DAC0800 bipolar de 8 bits basado en la Figura 8.23, utilizando para ello el programa *Electronics Workbench*.

- a) Realizar las modificaciones necesarias sobre el Problema resuelto 8-5 para cambiar el modo de funcionamiento de unipolar a bipolar y obtener un margen de salida  $\pm V_{omax} = \pm 5V$ .
- b) Calcular el nuevo valor de resolución  $a$  y rango a fondo de escala  $FS$ .
- c) Obterer la tabla de la función de transferencia del convertidor, en modo unipolar y bipolar, relacionando los códigos aplicados a la entrada y la tensión de salida  $V_o$  correspondientes a los códigos decimales: 0, 1, 2, ... , 128, ... , 253, 254 y 255.

#### Solución:

- a) Las modificaciones necesarias para cambiar el modo de funcionamiento a bipolar en un DAC0800 consisten en introducir una resistencia  $R$  de valor  $R_f$  entre el terminal  $V^+$  del amplificador operacional (señal  $I_0$  negada) y masa, según se muestra en el esquema de la Figura 8.23.
- b) En el modo de funcionamiento bipolar, el margen dinámico del convertidor según [8.38] y [8.39] es  $\pm V_{omax} = \pm 5 V$ . Para calcular la tensión de salida correspondiente a cada código, se debe conocer en primer lugar la resolución.

$$\text{Resolución modo bipolar} = \frac{2FS}{256} = \frac{2 \cdot 5,019}{256} = 39,210 \text{ mV}$$

- c) La función de salida de un DAC puede expresarse matemáticamente mediante la ecuación [8.21] (donde,  $K$  representa la resolución,  $D$  el valor equivalente decimal del código aplicado y  $C$  el valor inicial para el código decimal cero).

$$V_s = K D + C$$

$$C = -5V$$

$$K = LSB = a = 39,210 \cdot 10^{-3} V$$

Tabla 8.2. Comparación entre la función de transferencia del DAC0800 en modo unipolar y bipolar

Código Decimal	$V_o(V)$	$V_o(V)$
	Modo Unipolar	Modo Bipolar
0	0	-5
1	0,0196	-4,9608
2	0,0393	-4,9216
...	...	...
127	2,4898	-0,0203
128	2,5094	0,01888
...	...	...
253	4,9607	4,9201
254	4,971	4,9593
255	5	5

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap08\Ewb5\08W0\_\_06.ewb**

Mediante el empleo de los conmutadores asociados a las teclas [0], [1], ... , [6] y [7], que representan la posición de cada bit, aplicar las palabras binarias que se muestran a continuación comprobando el valor obtenido en la tensión de salida  $V_o$ .

- Entrada = 00000000 para obtener el valor  $V_{omin}$ .
- Entrada = 00000001 para obtener el valor  $LSB$ .
- Entrada = 00111111 para obtener el valor negativo más próximo a cero.
- Entrada = 10000000 para obtener el valor positivo más próximo a cero.
- Entrada = 11111111 para obtener el valor  $V_{omax}$ .



Introducir otras combinaciones y verificar el valor obtenido de  $V_o$ , comparándolo con los valores calculados en la Tabla 8.2.

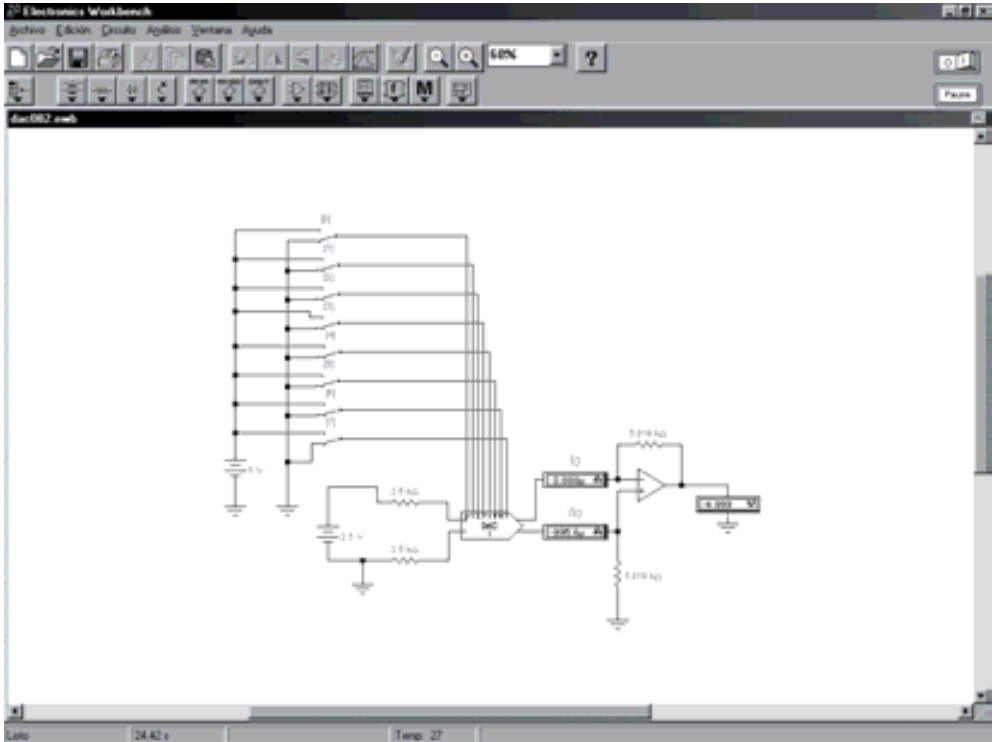


Figura 8.25. Circuito para simular el funcionamiento de un DAC0800 bipolar

### 8.3 CONVERTIDOR A/D

#### 8.3.1 Convertidor A/D. Generalidades

Para convertir una señal analógica en digital se requiere de un proceso basado en cuatro fases: **muestreo, retención, cuantificación y codificación.**

##### Muestreo

Se trata de una técnica que consiste en tomar una serie de muestras de la amplitud instantánea de la señal a intervalos regulares de tiempo obteniendo de este modo una serie de **pulsos modulados en amplitud (PAM)** (Figura 8.26). La frecuencia de muestreo  $f_c$  debe permitir la reconstrucción completa de la señal original a partir de las

muestras tomadas, lo que determina que dicha frecuencia sea al menos, el doble de la frecuencia máxima  $f_{max}(x(t))$  de la señal a muestrear  $x(t)$ , condición impuesta por el **teorema de Nyquist** y fundamento básico de la **teoría de muestreo**, que se representa en la expresión [8.40]. Donde,

- A.- Señal a muestrear  $x(t)$
- B.- Señal muestreadora de frecuencia  $f_c$
- C.- Señal A tras el proceso de muestreo
- D.- Señal C tras el proceso de retención

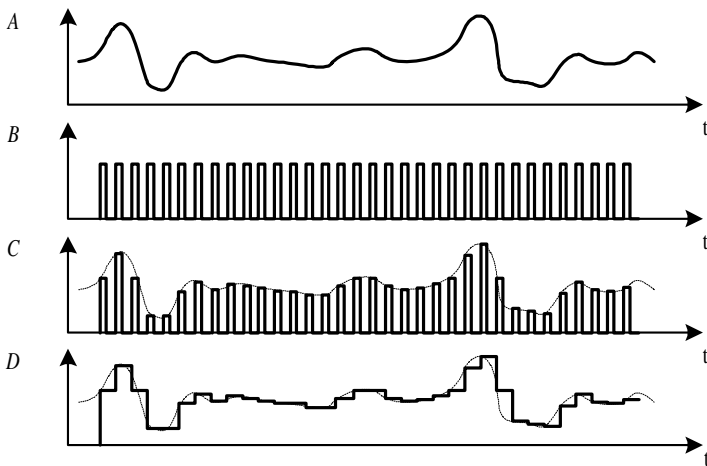


Figura 8.26. Cronograma del proceso de muestreo y retención

$$f_c \geq 2 f_{max}(x(t)) \quad [8.40]$$

### Retención

La tensión instantánea muestreada debe ser retenida entre dos impulsos consecutivos de muestreo para permitir que el circuito convertidor analógico-digital realice la conversión en el tiempo que precisa  $T_c$ . Las operaciones de muestreo y retención se obtienen a partir de circuitos específicos denominados de muestreo y retención (*Sample and Hold*) gobernados a partir del terminal  $V_{s/h}$ , como se muestra en la Figura 8.27.

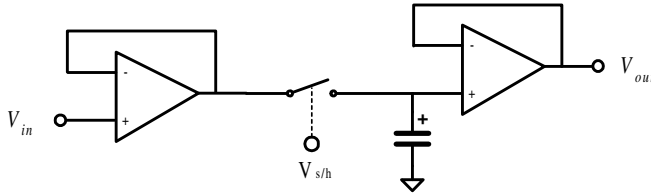


Figura 8.27. Circuito básico de muestreo y retención

### Cuantificación

El proceso de cuantificación consiste en asignar una función que determina qué margen de valores de la tensión de entrada deberá proporcionar un único nivel de la función de salida. Si dicha función es lineal, entonces el **intervalo de cuantificación** ( $q$ ) es constante y se determina a partir de la siguiente expresión:

$$q = \frac{V_{inmax} - V_{inmin}}{N} \quad [8.41]$$

donde,  $N$  representa el número de niveles del cuantificador.

Este valor,  $q$ , representa el intervalo  $V_{k+1} - V_k$  al cual se le hace corresponder como función de salida el nivel  $k \cdot q$  para valores de  $k = 0, 1, 2, 3, \dots, (N-1)$ .

La Figura 8.28 representa la función de transferencia de un cuantificador unipolar de 8 intervalos,  $N = 8$ , sobre un fondo de escala  $FS$  determinado. La función de salida del cuantificador, constará de  $N$  niveles de cuantificación divididos en  $N-1$  tramos o escalones. Por tanto, para un rango de entrada  $V_{in}$  entre  $V_{inmax} - V_{inmin}$  se obtendrá un rango de salida  $V_{out}$  entre 0 y  $(N - 1) q$ .

Es importante señalar que siempre que se aplica una función de cuantificación se produce un error (**error de cuantificación**) que depende básicamente del intervalo de cuantificación y de la función de salida del cuantificador.

El valor del error de cuantificación viene dado por la expresión [8.42] y su representación gráfica se muestra en la Figura 8.29.

$$\text{Error de cuantificación} = V_{out} - V_{in} \quad [8.42]$$

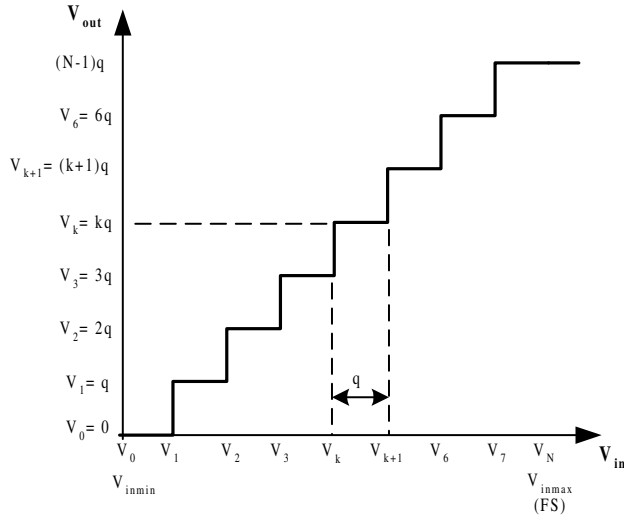


Figura 8.28. Función de transferencia de un cuantificador

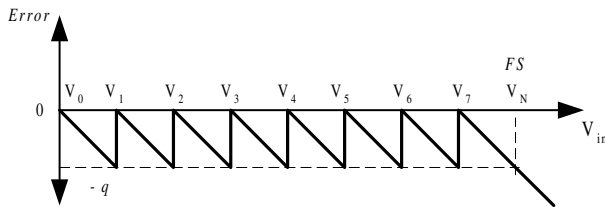


Figura 8.29. Error de cuantificación

Generalmente se emplean dos procedimientos de cuantificación:

- **Cuantificación por truncamiento.** Para el intervalo  $V_{k+1} - V_k$  se le asigna una función de salida  $(k \cdot q)$ . En estas condiciones se produce un error máximo de cuantificación de valor  $q$ . En la Figura 8.30 se representa un cuantificador de 8 intervalos y se observa que el rango de entrada es simétrico y el de salida asimétrico respecto del valor central.
- **Cuantificación por redondeo.** Para el intervalo  $V_{(k+q/2)} - V_{(k-q/2)}$  se le asigna una función de salida  $(k \cdot q)$ . En estas condiciones se produce un error máximo de cuantificación de valor  $\pm q/2$ . En la Figura 8.31 se observa que tanto el rango de entrada como el de salida son asimétricos respecto del valor medio, y por tanto el fondo de escala superior ( $FS_1$ ) e inferior ( $FS_2$ ) tienen distinto valor.

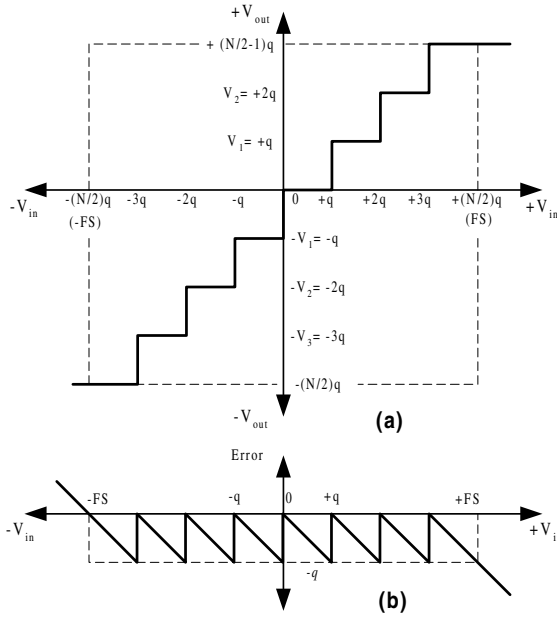


Figura 8.30. (a) Cuantificador por truncamiento. (b) Error de cuantificación

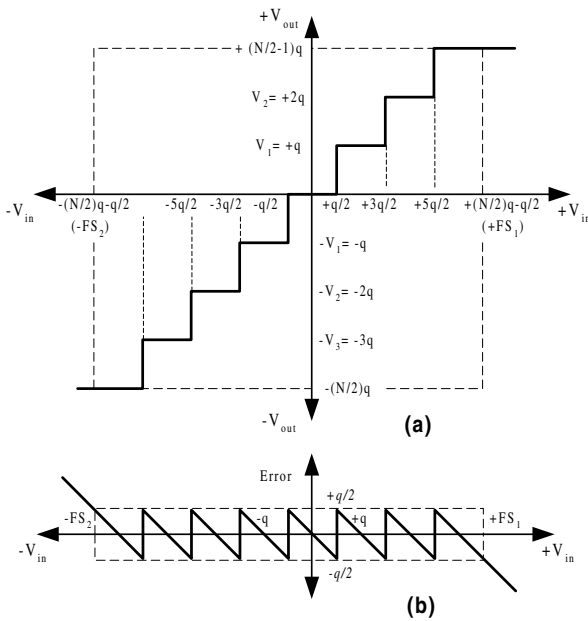


Figura 8.31. (a) Cuantificador por redondeo. (b) Error de cuantificación

Desde el punto de vista estadístico, **el error instantáneo**  $\varepsilon_i$  que se producirá en una determinada conversión está delimitado por su valor máximo  $q/2$ . La **función densidad de error**  $\varphi_\varepsilon$  tomará el valor  $1/q$ , para todo  $-q/2 \leq \varepsilon_i \leq +q/2$ .

$$\varphi_{(\varepsilon)} = \frac{1}{q} \quad [8.43]$$

El valor de la **media**  $\mu$  o valor central, será:

$$\mu = \int_{-\infty}^{+\infty} \varphi_{(\varepsilon)} \varepsilon d\varepsilon = 0 \quad [8.44]$$

La **desviación típica**  $\sigma$  respecto de la media, representa el valor del error probable en un cuantificador *EC*:

$$\sigma^2 = \int_{-\infty}^{+\infty} (\varepsilon - \mu)^2 \varphi_{(\varepsilon)} d\varepsilon = \int_{-q/2}^{+q/2} (\varepsilon - 0)^2 \frac{1}{q} d\varepsilon = \frac{q^2}{12} \quad [8.45]$$

$$\sigma = EC = \frac{q}{\sqrt{12}}$$

**Codificación.** El proceso de codificación consiste en asignar un código digital a cada uno de los niveles de salida del cuantificador.

Puesto que  $q$  representa el intervalo mínimo de cuantificación, este valor debe ser asignado a la variación mínima entre dos códigos consecutivos, equivalente al valor del bit menos significativo *LSB* del código empleado en la codificación. De este modo se puede establecer la relación de la expresión [8.46], a partir de [8.41].

$$q = \frac{V_{inmax} - V_{inmin}}{N} = LSB \quad [8.46]$$

Cuando se aplica una codificación binaria a la salida de un cuantificador, se hace preciso que el número  $N$  de intervalos de cuantificación coincida con una potencia de dos, con el fin de obtener todas las combinaciones posibles en el código de salida formado a partir de  $n$  bits.

$$q = \frac{V_{inmax} - V_{inmin}}{2^n} = LSB \quad [8.47]$$

Los códigos digitales más utilizados son los siguientes:

**Códigos unipolares:** Son aquellos destinados a representar únicamente valores positivos o negativos, correspondientes a la tensión de entrada en el cuantificador.

- **Binario Natural (BN).** Es uno de los códigos de salida más utilizados en los ADCs, donde cada palabra binaria representa una fracción del rango a fondo de escala  $FS$ . En la Tabla 8.3 se muestra el código binario natural aplicado a un cuantificador unipolar.
- **Binario Codificado en Decimal (BCD).**

Tabla 8.3. Código Binario Natural aplicado a un cuantificador unipolar

$V_{in}$	$V_{out}$	Código de Salida	Valor Decimal
$0 - q/2$	0	00000000	0
$q/2 - 3 q/2$	q	00000001	1
$5 q/2 - 7 q/2$	2q	00000010	2
...	...	...	...
$((N/2-1) q - q/2) - ((N/2) q - q/2)$	$(N/2-1) q$	01111111	127
$((N/2) q - q/2) - ((N/2+1) q - q/2)$	$(N/2) q$	10000000	128
...	...	...	....
$((N-2) q - q/2) - ((N-1) q - q/2)$	$(N-2) q$	11111110	254
$((N-1) q - q/2) - (N q - q/2)$	$(N-1) q$	11111111	255

**Códigos bipolares:** Son aquellos destinados a representar valores positivos y negativos, correspondientes a la tensión de entrada del cuantificador.

- **Binario natural con signo magnitud.**
- **Binario natural en complemento a dos.** Es el código de salida que mejor se adapta a las representaciones con variación de signo y centradas sobre cero. En la Tabla 8.4 se muestra el código binario natural en complemento a dos sobre un cuantificador bipolar.
- **Binario natural desplazado (OB).** Este código aporta la solución a la aplicación del código Binario Natural a un convertidor ADC bipolar, agregando un *offset* para conseguir que a la palabra binaria 100...000 le corresponda el valor 0 V de entrada, quedando la mitad superior de la tabla asociada a las tensiones positivas y la mitad inferior a las negativas de forma continua. En la Tabla 8.5 se muestra el código binario natural desplazado aplicado sobre un cuantificador bipolar.

Tabla 8.4. Código Complemento a dos aplicado sobre un cuantificador bipolar

$V_{in}$	$V_{out}$	Código de Salida	Valor Decimal
$((N/2-1)q - q/2) - ((N/2)q - q/2)$	$+(N/2-1)q$	01111111	+127
$((N/2-2)q - q/2) - ((N/2-1)q - q/2)$	$+(N/2-2)q$	01111110	+126
...	...	...	...
$q/2 - 3q/2$	$+q$	00000001	+1
$(-q/2) - (+q/2)$	0	00000000	0
$-3q/2 - (-q/2)$	$-q$	11111111	-1
...	...	...	...
$((-N/2-1)q - q/2) - ((-N/2-2)q - q/2)$	$-(N/2-1)q$	10000001	-127
$((-N/2)q - q/2) - ((-N/2-1)q - q/2)$	$-(N/2)q$	10000000	-128

Tabla 8.5. Código Binario Natural desplazado aplicado sobre un cuantificador bipolar

$V_{in}$	$V_{out}$	Código de Salida	Valor Decimal
$((N/2-1)q - q/2) - ((N/2)q - q/2)$	$+(N/2-1)q$	11111111	255
$((N/2-2)q - q/2) - ((N/2-1)q - q/2)$	$+(N/2-2)q$	11111110	254
...	...	...	...
$q/2 - 3q/2$	$+q$	10000001	129
$(-q/2) - (q/2)$	0	10000000	128
$-3q/2 - (-q/2)$	$-q$	01111111	127
...	...	...	...
$((-N/2-2)q - q/2) - ((-N/2-1)q - q/2)$	$-(N/2-1)q$	00000001	1
$((-N/2-1)q - q/2) - (-N/2q - q/2)$	$-(N/2)q$	00000000	0

### 8.3.2 Especificaciones de los convertidores A/D

**Resolución.** Especifica el número de bits del código digital de salida en relación con el intervalo de cuantificación  $q$  y el valor del rango de fondo de escala  $FS$ .



**Tiempo de conversión.** Es el tiempo que transcurre entre la orden de inicio de conversión (*SOC*, *Start Of Conversion*) y la salida estable del código equivalente a la señal de entrada, generalmente asociada a una señal *EOC* (*End Of Conversion*).

**Margen de tensión analógica de entrada.** Define el rango admisible de tensión de entrada al ADC.

**Código de salida.** Especifica el o los tipos de códigos de salida del ADC. Lo más común es que la salida se codifique en binario natural (BN) y/o binario natural en complemento a dos (BN-C-2).

**Salida digital.** Hace referencia al tipo de tecnología digital empleado en los *buffers* de salida (TTL, CMOS, ECL, etc.) y el tipo de salida (colector abierto, *Totem-Pole*, triestado, etc.).

En cuanto a los **errores en la conversión:**

**Error de cuantificación.** Este tipo de error ya ha sido mencionado en la expresión [8.42] y generalmente se asocia al valor  $\pm 1/2$  LBS o superior.

**Error de linealidad diferencial.** Si se produce un *error de linealidad* y éste es tal que impide la generación de determinados códigos a la salida, entonces se genera este error, también denominado de *códigos omitidos*.

El resto de los errores mantienen una definición muy similar a los estudiados en los DAC (apartado 8.2.2 Especificaciones de los convertidores D/A). Como son: error de precisión, *Linealidad*, *ganancia*, *offset* y las distintas variantes de la *sensibilidad* con la temperatura.

### 8.3.3 Circuitos convertidores A/D

Estos convertidores, también llamados ADCs, pueden construirse mediante muy diversas técnicas, que pueden ser agrupadas en dos tipos básicos: **ADCs de bucle abierto** y **ADCs de bucle cerrado**. En los primeros, como se muestra en la Figura 8.32, no existe realimentación interna, obteniendo la información digital de forma directa.



Figura 8.32. Esquema de un ADC en bucle abierto

El segundo tipo de ADCs, lo forman los que poseen un lazo de realimentación interno a partir de un convertidor DAC. En este caso, los procesos de cuantificación y codificación se producen de forma simultánea, obteniéndose una secuencia de códigos

digitales que a su vez son reconvertidos a un valor analógico y comparados con la señal de entrada. El código de salida se obtiene finalmente mediante técnicas de aproximación/comparación, como se muestra en la Figura 8.33.

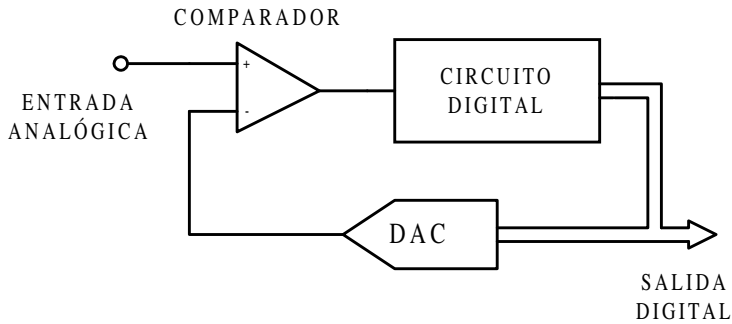


Figura 8.33. ADC en bucle cerrado

### 8.3.3.1 CONVERTIDOR A/D INSTANTÁNEO

En este tipo de convertidor los procesos de cuantificación y codificación están claramente separados y diferenciados como se muestra en la Figura 8.34.

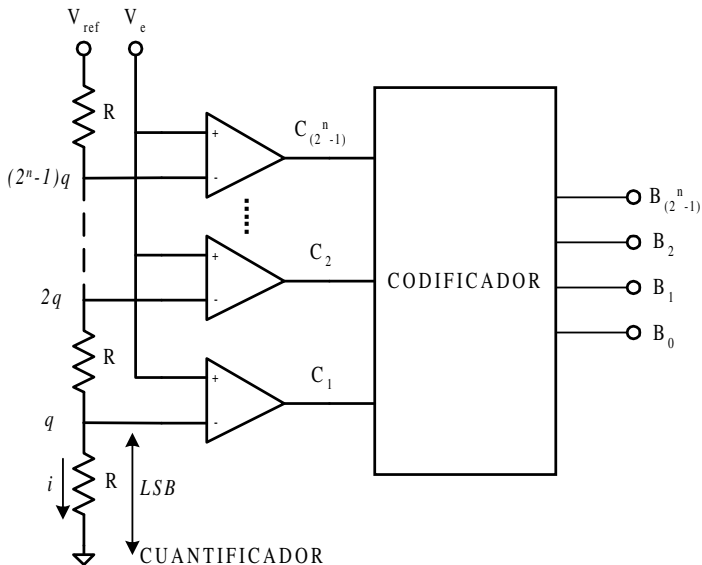


Figura 8.34. Esquema de un ADC instantáneo

El primero se lleva a cabo mediante comparadores que discriminan entre un número finito de niveles de tensión. Estos comparadores reciben en sus entradas la señal analógica junto con una tensión de referencia, distinta para cada uno de ellos. Finalmente, un bloque **codificador** obtendrá el código de salida mediante técnicas combinatoriales a partir de los distintos niveles de salida de los comparadores.

La Tabla 8.6 muestra el código de salida y el estado de las salidas del cuantificador en función de la variable de entrada, en un circuito ADC que tiene tres bits de resolución.

Tabla 8.6. ADC instantáneo de 3 bits

	CUANTIFICADOR	CODIFICADOR
	$C_7, C_6, C_5, C_4, C_3, C_2, C_1$	$B_2, B_1, B_0$
$0 \leq V_e < q$	0 0 0 0 0 0 0	0 0 0
$q \leq V_e < 2q$	0 0 0 0 0 0 1	0 0 1
$2q \leq V_e < 3q$	0 0 0 0 0 1 1	0 1 0
$3q \leq V_e < 4q$	0 0 0 0 1 1 1	0 1 1
$4q \leq V_e < 5q$	0 0 0 1 1 1 1	1 0 0
$5q \leq V_e < 6q$	0 0 1 1 1 1 1	1 0 1
$6q \leq V_e < 7q$	0 1 1 1 1 1 1	1 1 0
$7q \leq V_e \leq FS$	1 1 1 1 1 1 1	1 1 1

Este tipo de convertidor recibe el nombre de **instantáneo** porque es del tipo de alta velocidad, ya que el tiempo de conversión será la suma de los tiempos de conmutación de los comparadores más el tiempo de asentamiento del codificador.

Observando la red resistiva que proporciona las tensiones de referencia a los comparadores y sabiendo que dichas tensiones deben estar escalonadas para proporcionar los distintos niveles de cuantificación,  $q, 2q, 3q, \dots, 2^n-1q$ , se puede establecer que dicha red consta de  $2^n-1$  comparadores y  $2^n$  resistencias. Por consiguiente, a partir de la expresión [8.46] que se ha mostrado anteriormente, se obtiene la expresión [8.48]:

$$q = \frac{FS}{2^n} = LSB = i R = \frac{V_{ref}}{2^n} \quad [8.48]$$

y a partir de ésta se obtiene la expresión [8.49], que relaciona la tensión de referencia con el fondo de escala  $FS$ .

$$V_{ref} = FS \quad [8.49]$$

Este tipo de convertidor tiene la ventaja de la simplicidad en el diseño y la rapidez en el tiempo de conversión, para las aplicaciones de baja resolución. En la actualidad la máxima resolución aplicable a este tipo de ADCs sólo queda limitada por la tecnología empleada para conseguir un alto nivel de integración de componentes semiconductores, debido al gran número de comparadores implicado en el circuito.

### PROBLEMA RESUELTO 8-7



Simular el funcionamiento de un ADC instantáneo de 3 bits basado en el circuito de la Figura 8.34, utilizando para ello el programa *Electronics Workbench*.

- Calcular el valor de  $V_{ref}$  para obtener un rango a fondo de escala de valor  $FS = 4\text{ V}$ .
- Calcular el valor del intervalo de cuantificación  $q$ .

#### Datos para el cálculo:

$R = 10\text{ k}\Omega$ .

#### Solución:

- El cálculo de  $V_{ref}$  se realiza a partir de la expresión [8.49].

$$V_{ref} = FS = 4\text{ V}$$

- El valor del intervalo de cuantificación  $q$  se calcula a partir de la expresión [8.48].

$$q = LSB = \frac{FS}{2^n} = \frac{V_{ref}}{2^n} = \frac{4}{2^2} = 1\text{ V}$$

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap08\Ewb5\08W0\_\_07.ewb**

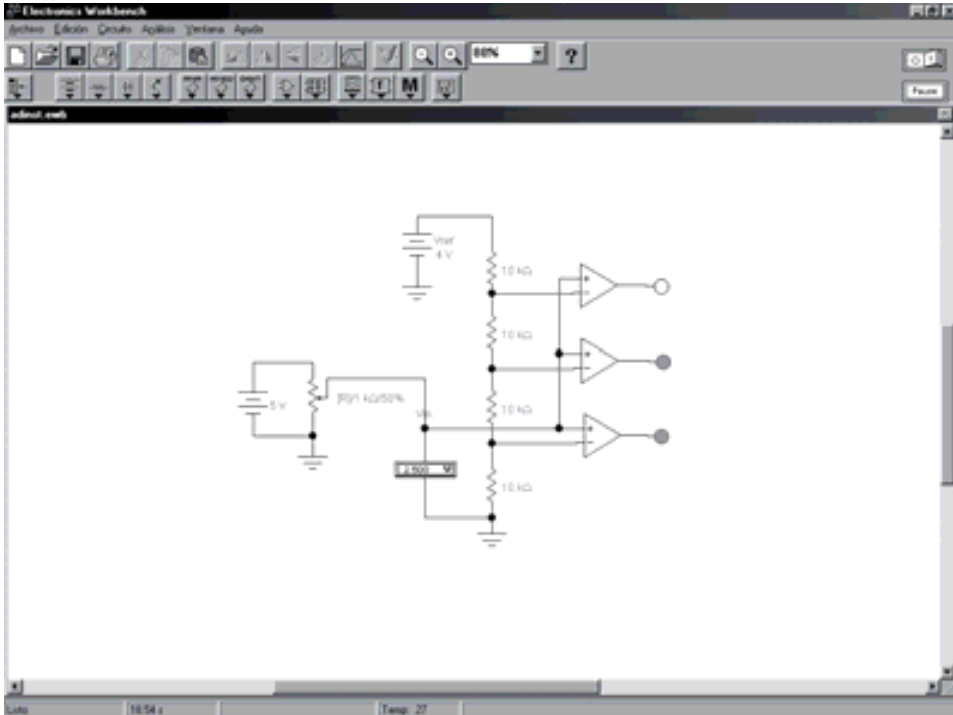


Figura 8.35. Circuito para simular el funcionamiento de un ADC instantáneo de 3 bits

Mediante la modificación del valor del potenciómetro [R] variar la tensión  $V_{in}$  aplicada al convertidor, observando la respuesta en la salida del cuantificador.

- Relacionar las variaciones en la salida del cuantificador con el intervalo de cuantificación  $q$ .

### 8.3.3.2 CONVERTIDOR A/D DE RAMPA

También denominado *ADC con integrador de simple pendiente*, este convertidor proporciona un impulso de anchura variable, directamente proporcional al valor de tensión analógica aplicado a la entrada, pudiendo ser convertido a una magnitud digital mediante un reloj y un contador de pulsos.

La Figura 8.36 muestra el diagrama de bloques básico de este convertidor compuesto por:

- **circuito de reloj** construido a partir de un generador de pulsos de frecuencia fija y estable,
- **generador de rampa** construido a partir de un circuito integrador,

- **biestable** cuya misión es la de fijar y retener un determinado estado lógico a su salida y
- **contador de pulsos** cuya misión consiste en contar el número de pulsos aplicado a su terminal de entrada en código binario natural.

Un impulso aplicado al terminal  $SOC$  determina el inicio de conversión mediante el comienzo en la generación de la rampa y la puesta a '1' lógico de la salida  $Q$  del biestable. Este nivel se mantendrá hasta que la tensión instantánea de la rampa supere a la entrada analógica, momento en que la salida del biestable volverá a '0' lógico y en consecuencia se active la señal  $EOC$  indicando el fin de conversión. El contador recibirá los impulsos de frecuencia fija del reloj mientras la salida  $Q$  del biestable permanezca a nivel alto. La Figura 8.37 muestra el cronograma de este proceso.

En consecuencia, al ser la duración del impulso en la salida del biestable directamente proporcional a la tensión analógica aplicada, la salida del contador será una representación digital de la misma.

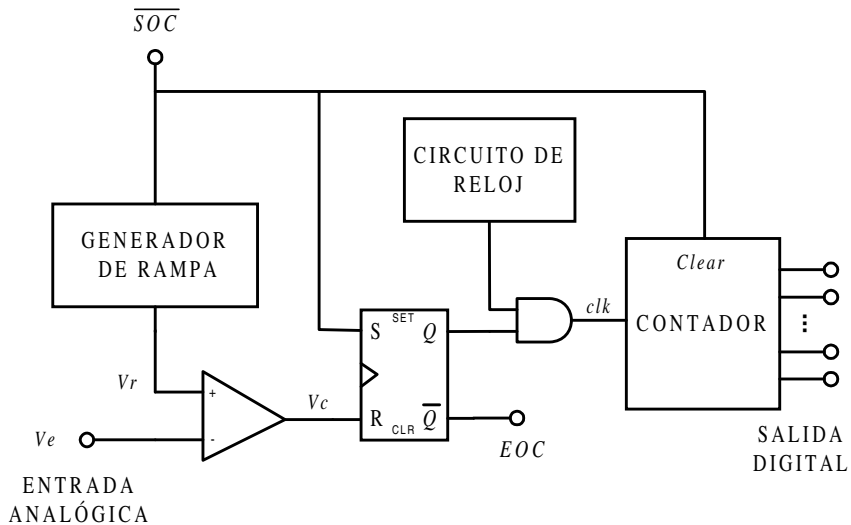


Figura 8.36. Esquema de un ADC de simple rampa

A efectos de diseño, se debe generar una rampa con una pendiente tal que coincida un intervalo de cuantificación  $q$  con un impulso completo de reloj  $T_{clk}$ , entonces la salida digital proporcionará directamente el código equivalente al número de intervalos de cuantificación de la que consta la señal analógica. En la expresión [8.50] se muestra la relación entre la pendiente de dicha rampa, los intervalos de cuantificación y el periodo de los impulsos de reloj.

$$Tg \alpha = \frac{q}{T_{clk}} = q F_{clk} \tag{8.50}$$

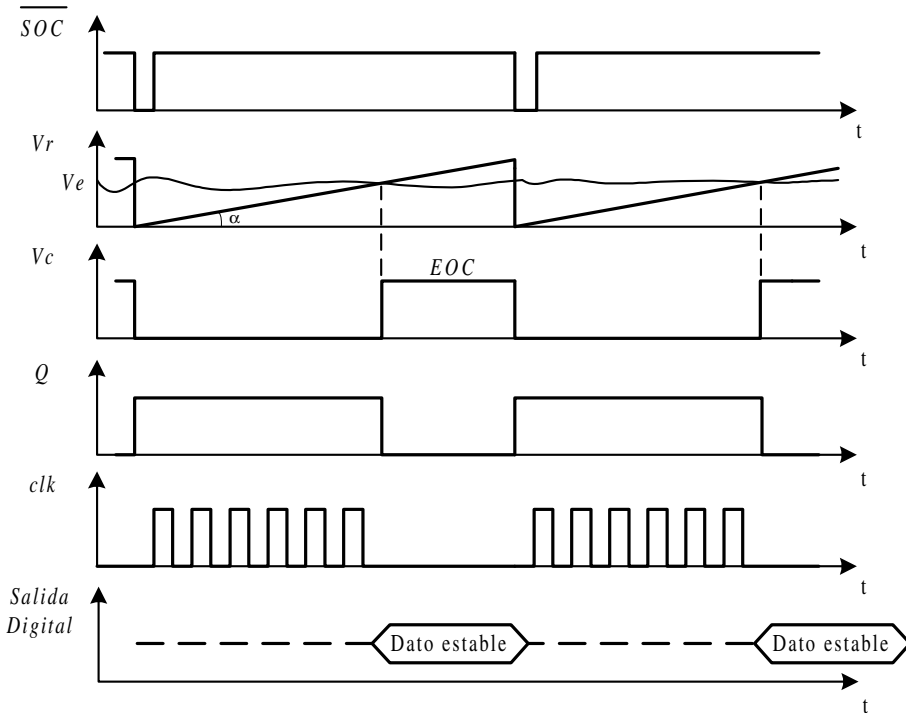


Figura 8.37. Cronograma del ADC de simple rampa

En estas condiciones, la amplitud de la rampa deberá ser superior o igual al fondo de escala  $FS$  del convertidor para el tiempo máximo de conversión  $T_{cmax}$ .

$$T_{cmax} = T_{clk} (2^n - 1) \tag{8.51}$$

La frecuencia máxima de los impulsos de inicio de conversión  $SOC$  determina el número máximo de muestras de la señal que se pueden garantizar para la peor de las circunstancias. En un sistema genérico de adquisición de datos, esto representa la máxima frecuencia de muestreo  $F_c$ .

$$F_c \leq \frac{1}{T_{cmax}} \leq \frac{1}{T_{clk} (2^n - 1)} \quad [8.52]$$

Las limitaciones de este tipo de convertidor son varias:

- Errores por falta de linealidad en la rampa.
- Puesta a cero de la rampa con el inicio de conversión.
- Tiempo de conversión variable.
- La precisión depende de la estabilidad del reloj.

### PROBLEMA RESUELTO 8-8



Simular el funcionamiento de un DAC de simple rampa de 8 bits basado en el circuito de la Figura 8.36, utilizando para ello el programa *Electronics Workbench*.

- a) Calcular el valor del intervalo de cuantificación  $q$ .
- b) Calcular el valor de la frecuencia de reloj  $F_{clk}$  para obtener una relación directa entre el valor de la tensión analógica aplicada al convertidor y la salida binaria del contador digital de pulsos.

#### Datos para el cálculo:

Fondo de escala  $FS = 10$  V.

Pendiente de la rampa del integrador  $g(\alpha) = 1$  V/s.

#### Solución:

- a) El valor del intervalo de cuantificación  $q$  se determina a partir de la expresión [8.41].

$$q = \frac{V_{inmax} - V_{inmin}}{N} = \frac{FS}{2^n} = \frac{10}{256} = 39,06 \text{ mV}$$

- b) El cálculo de  $F_{clk}$  se realiza a partir de la expresión [8.50].

$$Tg \alpha = \frac{q}{T_{clk}} = q F_{clk}$$

$$F_{clk} = \frac{Tg \alpha}{q}$$



$$F_{clk} = \frac{\operatorname{tg} \alpha}{q} = \frac{1}{39,06 \cdot 10^{-3}} = 25,6 \text{ Hz}$$

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap08\Ewb5\08W0\_08.ewb**

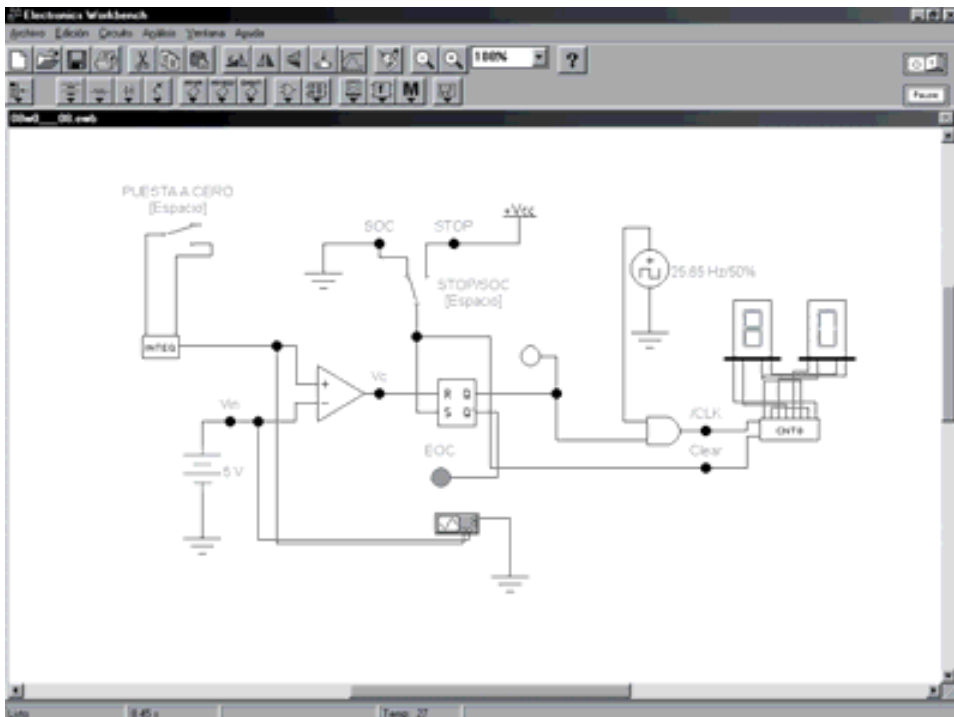


Figura 8.38. Circuito para simular el funcionamiento de un ADC de simple rampa

El circuito de simulación incluye un doble conmutador para el control del convertidor. La situación inicial de dicho conmutador debe ser la posición **STOP**, lo que implica la puesta a cero de la rampa del integrador, la puesta a cero del contador y la puesta a uno lógico del biestable. El cambio del conmutador a la posición **SOC** determina el inicio de la conversión hasta la activación de la señal **EOC**, indicadora de la finalización del proceso. Para realizar una nueva conversión es preciso repetir el proceso de **STOP/SOC**. En la Figura 8.39 se muestra la señal en forma de rampa que se aplica al circuito representado en la Figura 8.38.

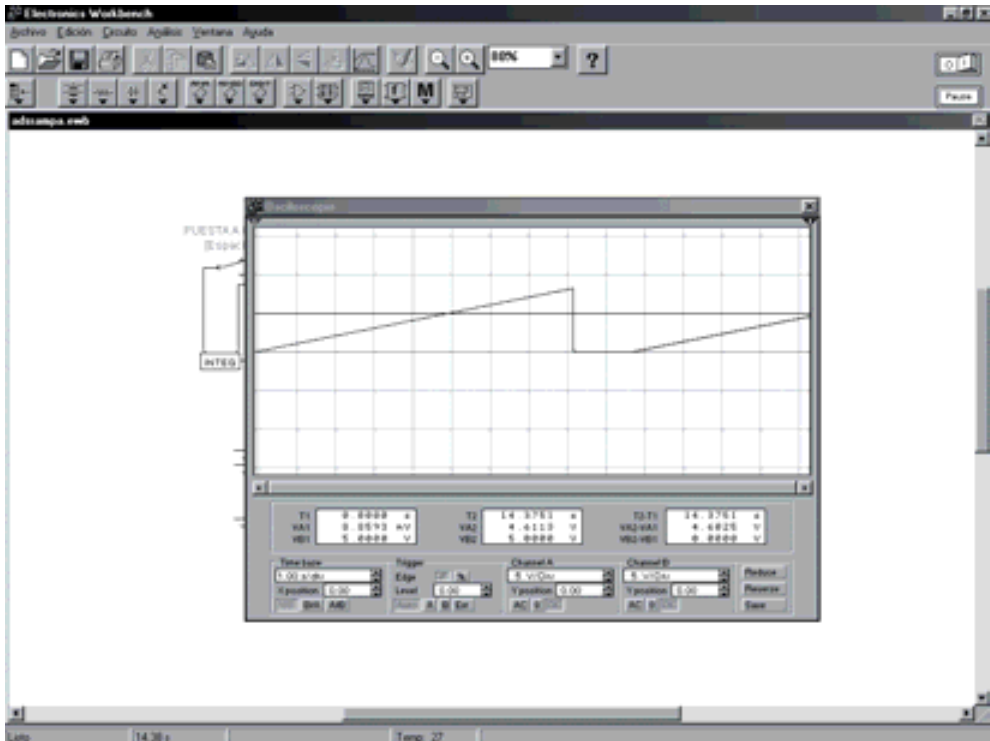


Figura 8.39. Oscilograma de la rampa

Variar la tensión aplicada al convertidor  $V_{in}$ , modificando el valor de la fuente de tensión. Iniciar la conversión mediante el proceso *STOP/SOC* observando la evolución del contador de pulsos sobre los indicadores de siete segmentos. Observar la indicación final coincidiendo con la activación de la señal *EOC*.

Se debe tener en cuenta que el contador de pulsos siempre introduce una imprecisión de  $\pm 1$  pulso de reloj, razón por la cual el indicador puede fluctuar en  $\pm 1$  unidad.

- Entrada = +10 V para obtener el valor FFh (255 pulsos).
- Entrada = +5 V para obtener el valor 80 (127 pulsos).
- Introducir otros valores de  $V_{in}$ , y comprobar el valor obtenido a la salida del contador.

### 8.3.3.3 CONVERTIDOR A/D DE DOBLE RAMPA

Existen dos tipos de convertidores A/D de doble rampa: simétrica y asimétrica, ambos se describen en este apartado.

### Convertidor A/D de doble rampa simétrica

Este tipo de convertidor es uno de los más utilizados, especialmente en aplicaciones donde se requiere cierta precisión a bajo coste como es el caso de los voltímetros digitales, donde el tiempo de conversión no es una característica importante. El principio básico de funcionamiento sigue siendo la comparación de la señal de entrada con una rampa en forma de señal triangular.

En la Figura 8.40 se muestra el diagrama de bloques básico de este tipo de convertidor, donde la duración del impulso en la salida del comparador es directamente proporcional a la tensión analógica aplicada y al igual que en el caso anterior, la salida del contador será una fiel representación digital de la misma. El **inicio de conversión** se produce automáticamente cuando la tensión instantánea de la entrada analógica supera a la rampa en su tramo descendente, momento en que la salida del comparador pasará a '1' lógico. Este nivel se mantiene hasta que la tensión instantánea de la rampa supera a la entrada analógica en su tramo ascendente, momento en que la salida del comparador vuelve a '0' lógico. El contador recibe los impulsos de frecuencia fija del reloj mientras la salida del comparador permanezca a nivel alto.

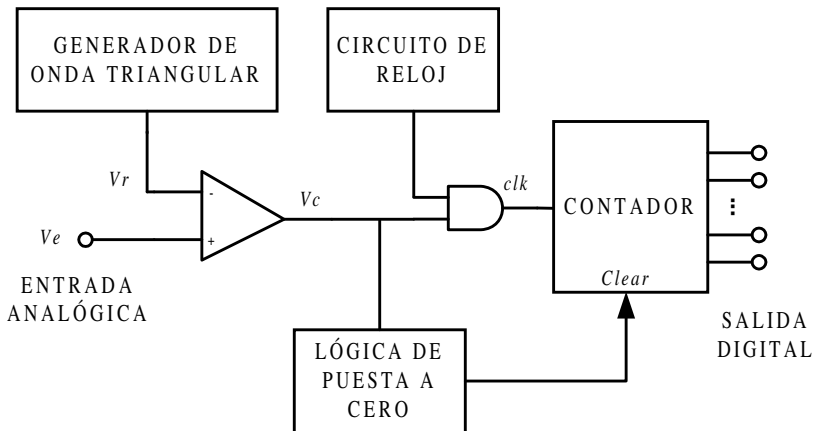


Figura 8.40. Esquema del ADC de doble rampa simétrica

En la Figura 8.41 se muestra el cronograma del proceso descrito anteriormente en forma repetitiva.

A efectos de diseño, se debe generar una señal triangular simétrica con una pendiente tal que coincidan dos intervalos de cuantificación  $q$ , por cada impulso completo de reloj  $T_{clk}$ , como se muestra en la Figura 8.45.

De este modo, la salida digital proporcionará directamente el código equivalente al número de intervalos de cuantificación de la que consta la señal analógica.

$$Tg \alpha = \frac{2q}{T_{clk}} = 2q F_{clk} \quad [8.53]$$

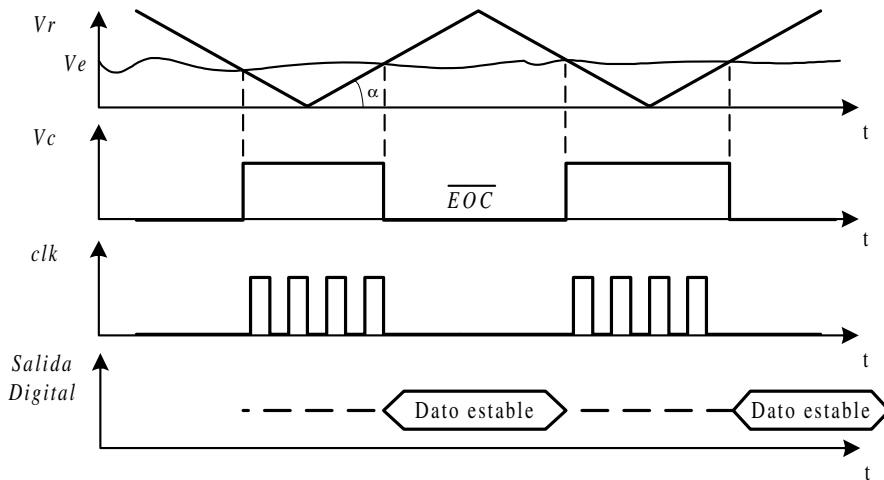


Figura 8.41. Cronograma del ADC de doble rampa simétrica

En estas condiciones, la amplitud de la rampa deberá ser superior o igual al fondo de escala  $FS$  del convertidor, siendo el tiempo máximo de conversión  $T_{cmax}$ :

$$T_{cmax} = T_{clk} (2^n - 1) \quad [8.54]$$

El número máximo de capturas por segundo que puede realizar este convertidor, depende únicamente de la frecuencia de la señal triangular  $F_{triangular}$ .

$$F_c = F_{triangular} \quad [8.55]$$

### Convertidor A/D de doble rampa asimétrica

Otro convertidor muy extendido es el que se muestra en la Figura 8.42, conocido como ADC de doble rampa asimétrica.

El proceso de conversión se inicia conectando la tensión de entrada analógica al integrador durante un tiempo fijo  $T_1$ , el que necesita el contador para pasar desde el estado inicial de conteo 00 ... 00 hasta el estado final de desbordamiento, mientras

cuenta  $2^n$  impulsos de reloj. La indicación de desbordamiento (*overflow*) provoca la aplicación de la tensión de referencia  $V_{ref}$  a la entrada del integrador lo que hace que éste evolucione con pendiente positiva hacia la tensión de cero voltios. Durante este periodo se realiza una nueva cuenta de los impulsos de reloj hasta la indicación del detector de paso por cero, donde el resultado de la cuenta será directamente proporcional a la tensión analógica aplicada al integrador. El gráfico de la Figura 8.43 muestra la salida del integrador en las dos fases del conmutador.

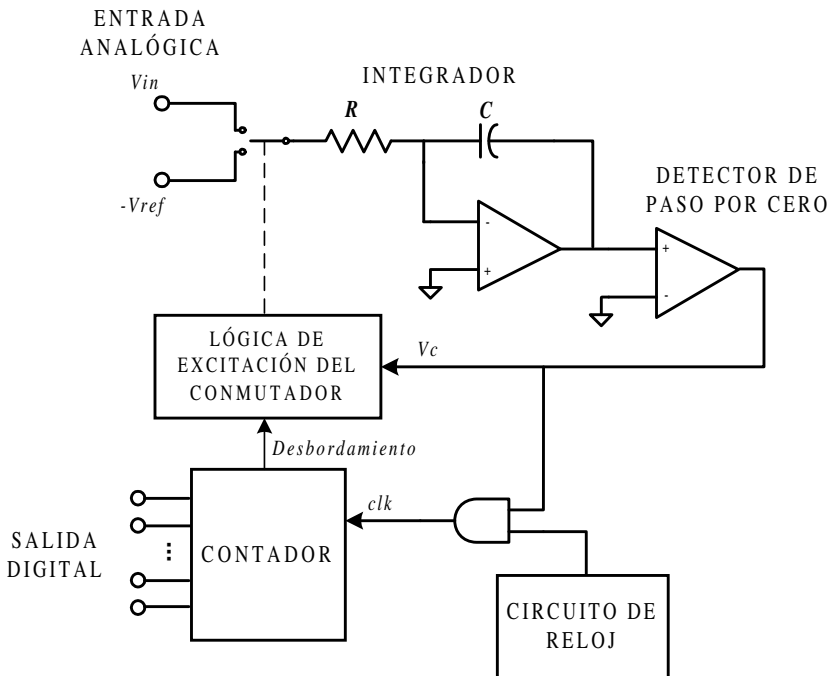


Figura 8.42. Esquema del ADC de doble rampa asimétrica

Como puede verse en la Figura 8.43  $T_X < T_1$ , lo que implica que  $V_{in} < V_{ref}$ , siendo el valor de  $T_1$  y  $T_X$  el expresado en [8.56].

$$\begin{aligned}
 T_1 &= 2^n T \\
 T_X &= \beta T
 \end{aligned}
 \tag{8.56}$$

Partiendo del punto común ( $-V_X$ ) se tiene que:

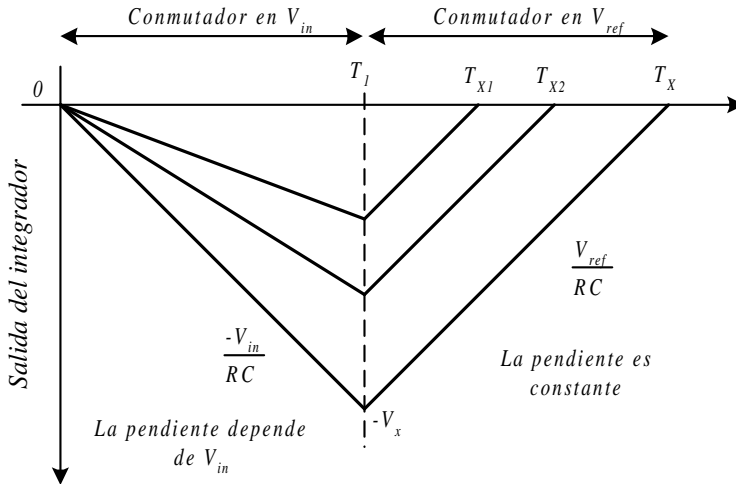


Figura 8.43. Salida del integrador de doble rampa asimétrica

$$-V_x = -\frac{V_{in}}{RC} T_1 = -\frac{V_{ref}}{RC} T_x$$

$$T_x = \beta T = T_1 \frac{V_{in}}{V_{ref}} = \frac{V_{in}}{V_{ref}} 2^n T \quad [8.57]$$

Si se calibra  $V_{ref} = 2^n$

$$\beta = V_{in}$$

Lo que implica que la cuenta numérica de pulsos aplicados al contador es igual a la tensión analógica aplicada a  $V_{in}$ .

La característica más notable que presenta este circuito es la precisión como resultado de la independencia de su salida respecto de la estabilidad del valor del condensador o de la frecuencia de reloj, con tal de que se mantengan constantes durante el periodo de conversión.

Este tipo de convertidor supera algunas de las limitaciones indicadas en el convertidor de simple rampa. Algunas de estas características son las siguientes:

- Mejora la linealidad en la rampa.
- No requiere una puesta a cero de la rampa con el inicio de conversión.
- Mejora la precisión.
- Como contrapartida, se mantiene un tiempo de conversión largo y variable.

### PROBLEMA RESUELTO 8-9



Simular el funcionamiento de un ADC bipolar de doble rampa simétrica de 8 bits basado en el circuito de la Figura 8.40, utilizando para ello el programa *Electronics Workbench*.

- Calcular los valores de amplitud y frecuencia del generador de señal triangular.
- Calcular el valor del intervalo de cuantificación  $q$ .
- Calcular el valor de la frecuencia de reloj  $F_{clk}$  para obtener una relación directa entre el valor de la tensión analógica aplicada al convertidor y la salida binaria del contador digital de pulsos.

#### Datos para el cálculo:

Fondo de escala  $\pm FS = \pm 5 \text{ V}$ .

Pendiente de la rampa del integrador  $Tg(\alpha) = 2 \text{ V/s}$ .

#### Solución:

- Para que el  $FS$  del convertidor tenga el valor  $\pm 5 \text{ V}$  se requiere generar una onda triangular de amplitud  $\pm 5 \text{ V}$ .

$$V_{\text{triangular}} = \pm 5 \text{ V}$$

Para que la onda triangular tenga una pendiente de  $2 \text{ V/s}$ , con las amplitudes mencionadas, se requiere que el valor de la frecuencia sea:

$$p = \frac{V_{\max} - V_{\min}}{\frac{T}{2}}$$

$$T = \frac{2(V_{\max} - V_{\min})}{p}$$

$$F_{\text{triangular}} = \frac{1}{T} = \frac{p}{2(V_{\max} - V_{\min})} = \frac{2}{20} = 0,1 \text{ Hz}$$

- El valor del intervalo de cuantificación  $q$  se determina a partir de la expresión [8.41].

$$q = \frac{V_{i\max} - V_{i\min}}{N} = \frac{FS}{2^n} = \frac{10}{256} = 39,06 \text{ mV}$$

c) El cálculo de  $F_{clk}$  se realiza a partir de la expresión [8.53].

$$Tg \alpha = \frac{2q}{T_{clk}} = 2q F_{clk}$$

$$F_{clk} = \frac{Tg \alpha}{2q}$$

$$F_{clk} = \frac{Tg \alpha}{2q} = \frac{2}{2 \cdot 0,03906} = 25,6 \text{ Hz}$$

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap08\Ewb5\08W0\_09.ewb**

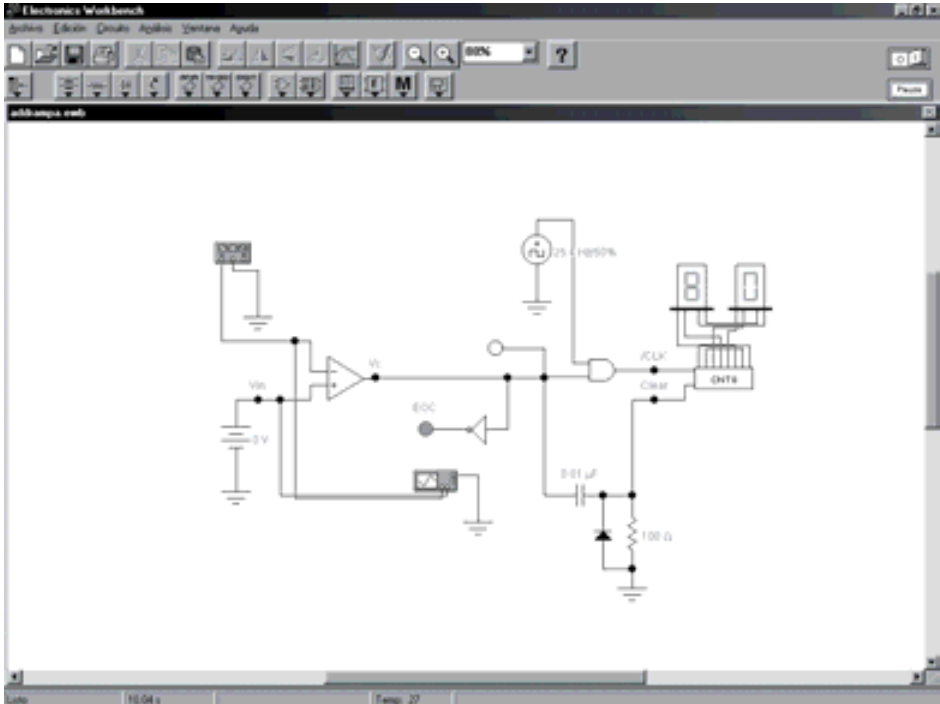


Figura 8.44. Circuito para simular un ADC de doble rampa simétrica



El circuito de simulación incluye un circuito diferenciador ( $RC$ ) como circuito de **Lógica de puesta cero (*clear*) del contador** al inicio de cada nueva conversión. La conversión en este circuito es cíclica de tal modo que el resultado final sólo es válido ante la señal activa *EOC*.

En la Figura 8.45 se muestra, de forma gráfica, la señal triangular que se introduce en la entrada del circuito.

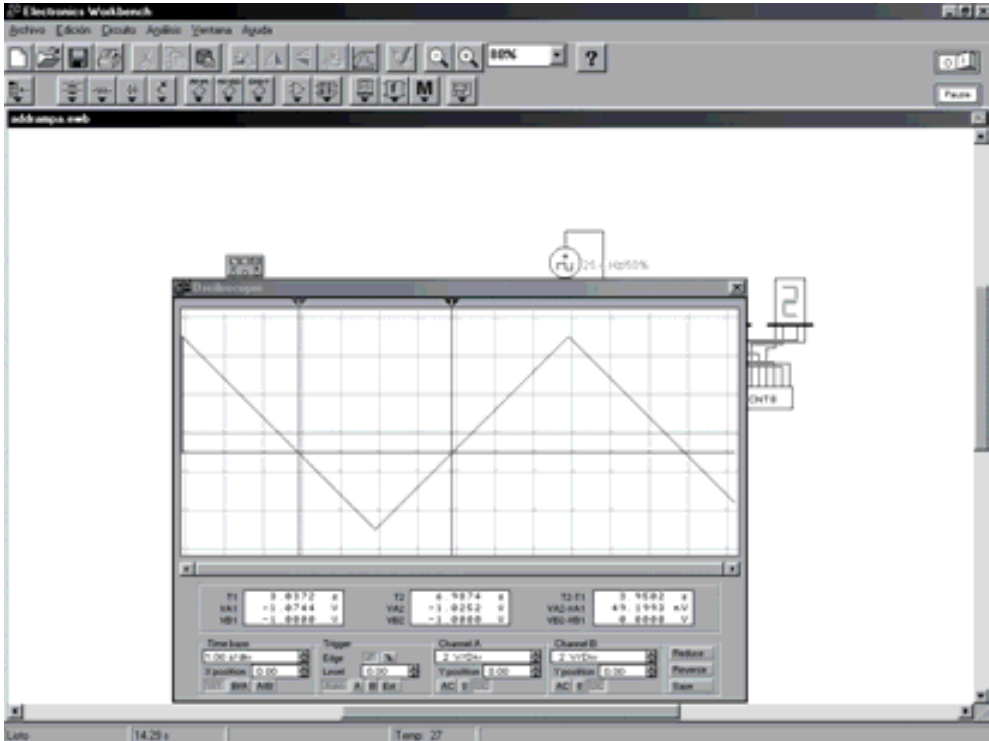


Figura 8.45. Oscilograma de la señal triangular

En el circuito anterior, variar la tensión aplicada en la entrada del convertidor  $V_{in}$ , modificando el valor de la fuente de tensión. Iniciar la conversión mediante la puesta en marcha del simulador. Observar la indicación final coincidiendo con la activación de la señal *EOC*.

Se debe tener en cuenta que el contador de pulsos siempre introduce una imprecisión de  $\pm 1$  pulso de reloj, razón por la cual el indicador puede fluctuar en  $\pm 1$  unidad.

- Entrada = +4,99 V para obtener el valor FFh (255 pulsos).
- Entrada = 0V para obtener el valor 80 (127 pulsos).

- Entrada = -2,5 V para obtener el valor 40 (64 pulsos).
- Introducir otros valores de  $V_{in}$ , y comprobar el valor obtenido a la salida del contador.
- Obsérvese que la frecuencia de reloj  $F_{clk}$  es independiente del valor de salida del convertidor.

### 8.3.3.4 CONVERTIDOR A/D POR CONTADOR

El convertidor ADC por contador es la evolución natural del convertidor de simple rampa realizado a partir de técnicas digitales. Construido sobre la estructura de un bucle cerrado, consta de un comparador, un reloj, un contador y un DAC como dispositivo de realimentación. El principio básico de funcionamiento sigue siendo la comparación de la señal analógica de entrada con una rampa escalonada generada a partir de un contador y el DAC.

La Figura 8.46 muestra el diagrama de bloques básico de este tipo de convertidor, donde la duración del impulso en la salida del comparador es directamente proporcional a la tensión analógica aplicada y al igual que en los casos anteriores, la salida del contador será una fiel representación digital de la misma. El inicio de conversión  $SOC$  se produce al aplicar una señal de  $CLEAR$  al contador y en consecuencia iniciar una rampa escalonada en la salida del DAC, instante en que la salida del comparador pasa a '1' lógico. Este nivel se mantiene hasta que la tensión instantánea de la rampa supera a la entrada analógica, momento en que la salida del comparador vuelve a '0' lógico. El contador recibe los impulsos de frecuencia fija del reloj mientras la salida del comparador permanezca a nivel alto.

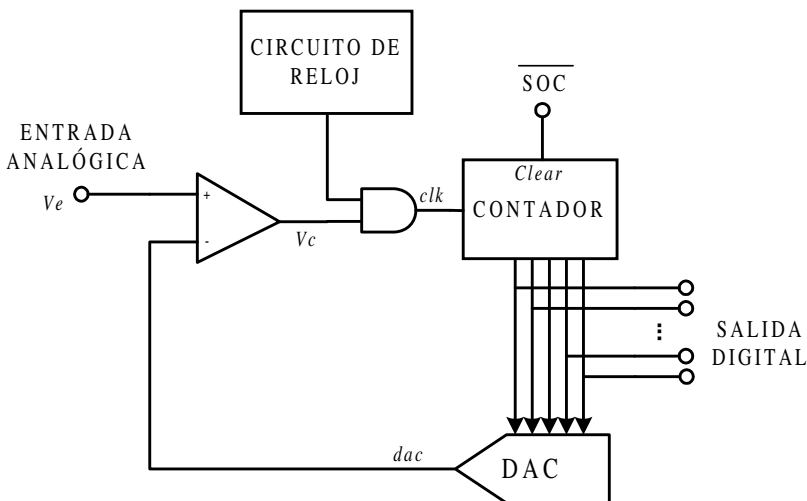


Figura 8.46. Esquema de un ADC por rampa en escalera

En la Figura 8.47 se muestra, de forma gráfica, el cronograma del proceso descrito anteriormente.

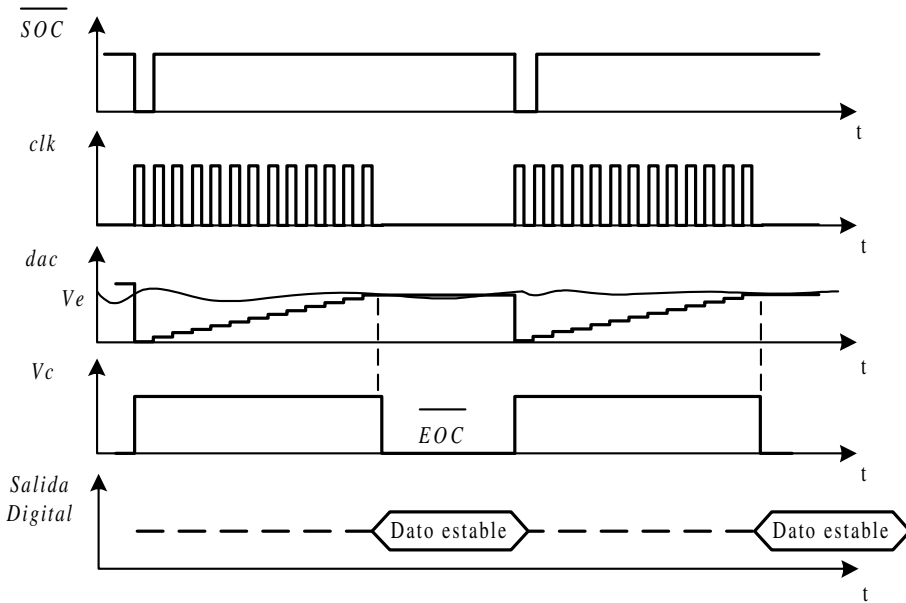


Figura 8.47. Cronograma del ADC por rampa en escalera

A efectos de diseño, se debe generar una rampa escalonada cuya resolución  $a$  coincida con un intervalo de cuantificación  $q$ , como se muestra en la expresión [8.58]; con esta condición la salida digital proporciona directamente el código equivalente al número de intervalos de cuantificación de la que consta la señal analógica.

$$a = q = LSB \tag{8.58}$$

La frecuencia máxima de reloj  $F_{clk}$  debe tener un valor máximo impuesto por la función inversa de la suma de los retardos y tiempos de conmutación acumulados a lo largo del bucle cerrado, por lo que el periodo máximo viene dado por la expresión [8.59].

$$T_{cmax} = T_{clk} (2^n - 1) \tag{8.59}$$

Este tipo de convertidor presenta la ventaja de la simplicidad y el empleo de técnicas exclusivamente digitales, aunque presenta un grave inconveniente al tener

que pasar forzosamente por todos los estados, desde el inicial hasta obtener el valor de conversión buscado, lo que implica tiempos de conversión largos y variables.

Una variante en este tipo de ADC es el **Convertidor A/D Continuo por Contador** donde se pretende suplir algunas de las deficiencias indicadas con anterioridad, y cuyo esquema de conexión se muestra en la Figura 8.48.

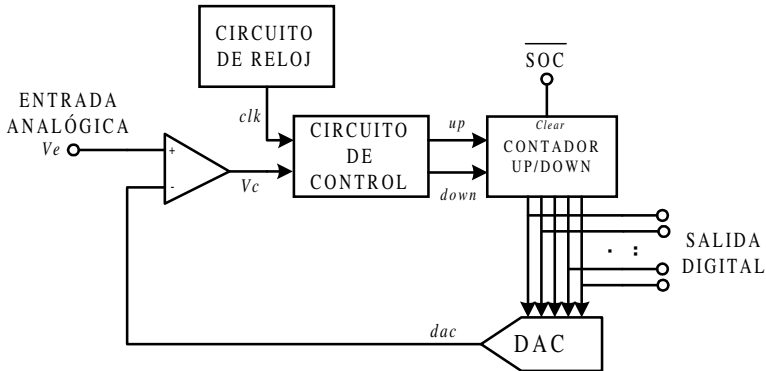


Figura 8.48. Esquema del ADC continuo

La Figura 8.49 muestra el cronograma de este proceso de forma repetitiva.

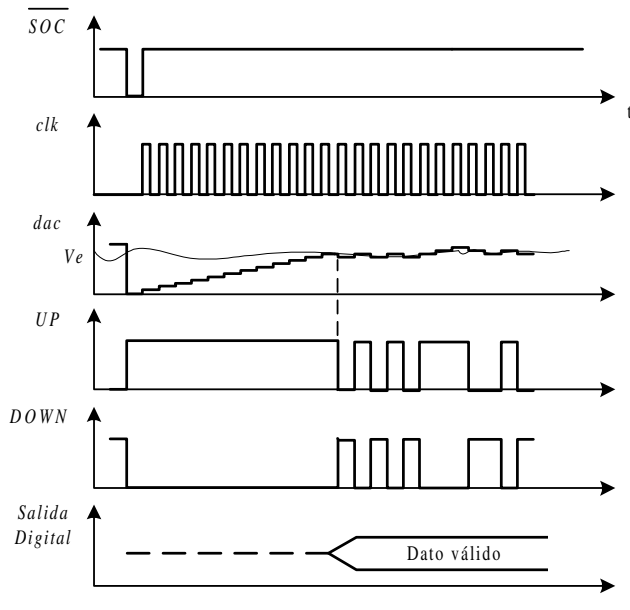


Figura 8.49. Cronograma del ADC continuo

El principio de funcionamiento es el mismo que el descrito para el caso anterior, pero en lugar de iniciar una nueva rampa por cada periodo de conversión, ésta tratará de seguir continuamente las variaciones de la señal analógica mediante el empleo de un contador reversible. Este modo de funcionamiento reduce el tiempo de conversión a partir de la primera muestra de señal, puesto que existe una gran probabilidad de que la siguiente muestra se encuentre próxima a la anterior.

### PROBLEMA RESUELTO 8-10



Simular el funcionamiento de un ADC de rampa escalonada por contador de 8 bits basado en el circuito de la Figura 8.46, empleando el convertidor DAC0800 unipolar del Problema resuelto 8-5, utilizando para ello el programa *Electronics Workbench*.

- Calcular el valor del intervalo de cuantificación  $q$ .
- Determinar el fondo de escala  $FS$  del convertidor.

#### Datos para el cálculo:

Resolución del DAC = 19,605 mV.

$V_{omax}$  del DAC = 5 V.

#### Solución:

- El valor del intervalo de cuantificación  $q$  se determina a partir de la expresión [8.58].

$$a = q = LSB = 19,605 \text{ mV}$$

- El valor de  $FS$  del ADC coincide con el valor de referencia del comparador, es decir, con el valor de  $V_{omax}$  del DAC.

$$FS = V_{omax}(DAC) = 5V$$

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap08\Ewb5\08W0\_\_10.ewb**

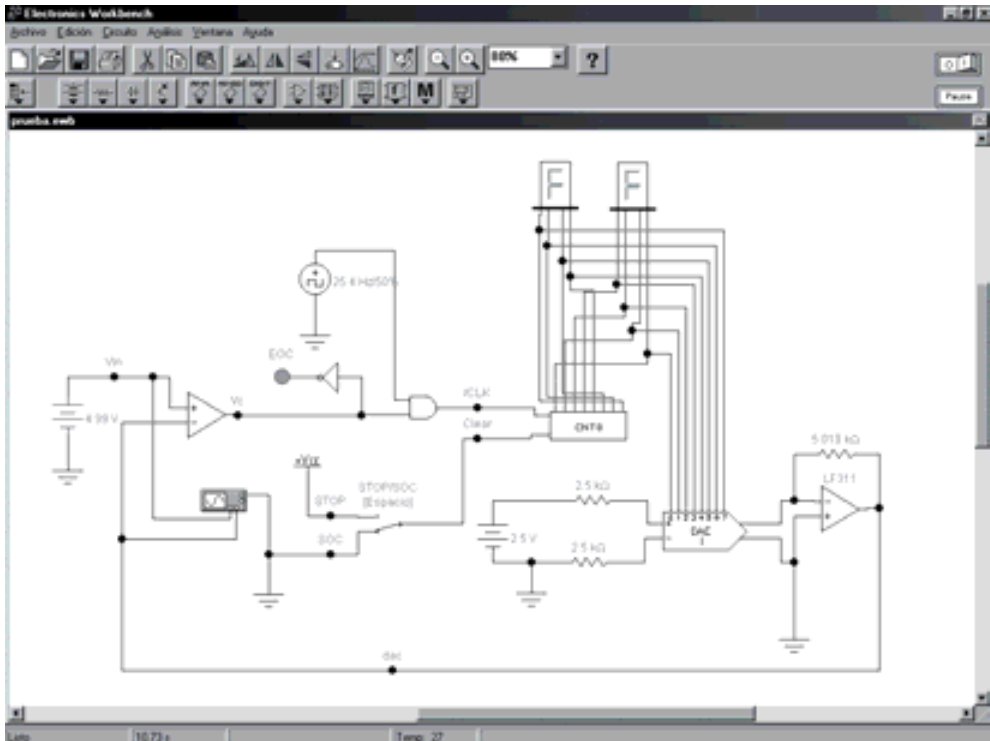


Figura 8.50. Circuito para simular el funcionamiento de un ADC de rampa en escalera

El circuito de simulación incluye un conmutador para el control del convertidor. La situación inicial debe ser la posición **STOP** que implica la puesta a cero del contador. El cambio del conmutador a la posición **SOC** determina el inicio de la conversión hasta la activación de la señal **EOC**, indicadora de la finalización del proceso. Para realizar una nueva conversión es preciso repetir el proceso de **STOP/SOC**.

En la Figura 8.51 se muestra, de forma gráfica, la señal que introduce en la entrada del circuito.

Se deja como ejercicio para el lector, variar, en dicho circuito la tensión aplicada a la entrada del convertidor  $V_{in}$ , modificando el valor de la fuente de tensión. Iniciar la conversión mediante el proceso **STOP/SOC** observando la evolución del contador de pulsos sobre los indicadores de siete segmentos. Observar la indicación final coincidiendo con la activación de la señal **EOC**.

Se debe tener en cuenta que el contador de pulsos siempre introduce una imprecisión de  $\pm 1$  pulso de reloj, razón por la cual el indicador puede fluctuar en  $\pm 1$  unidad.

- Entrada  $V_{in} = +4,99 \text{ V}$  para obtener el valor FFh (255 pulsos).
- Entrada  $V_{in} = +2,5 \text{ V}$  para obtener el valor 80 (127 pulsos).
- Introducir otros valores de  $V_{in}$ , y comprobar el valor obtenido a la salida del contador.
- Observese que la frecuencia de reloj  $F_{clk}$  es independiente del valor de salida del convertidor.

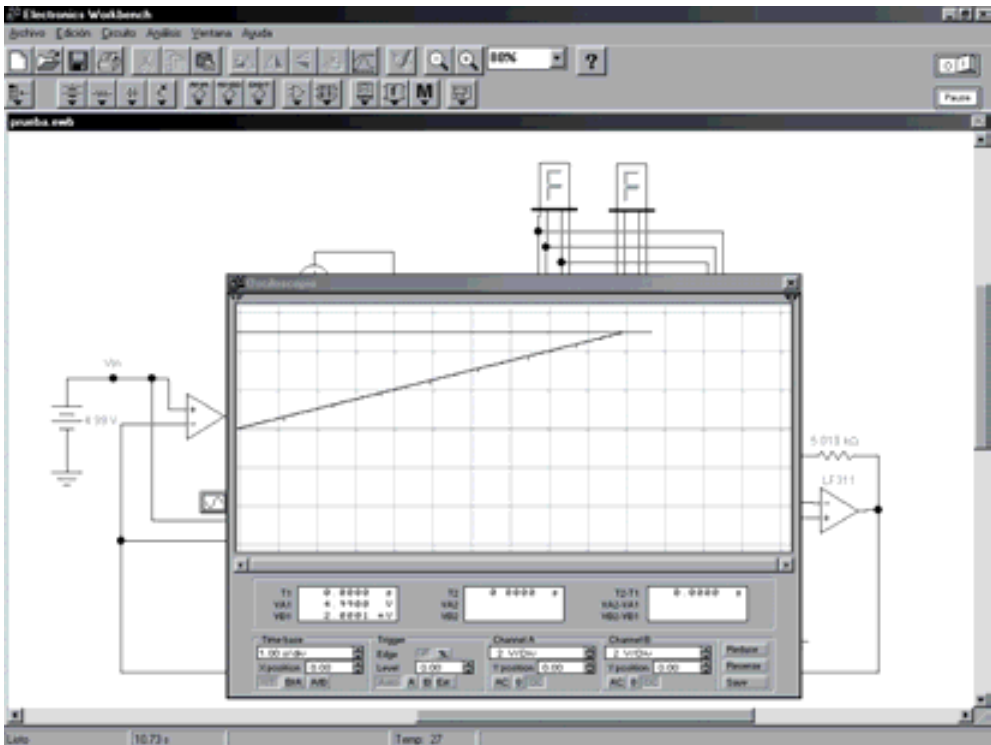


Figura 8.51. Oscilograma del ADC de rampa en escalera

### 8.3.3.5 CONVERTIDOR A/D POR APROXIMACIONES SUCESIVAS

Éste es el tipo de convertidor más utilizado en aplicaciones generales donde se requieren medias y altas velocidades de conversión. Basándose en un principio de búsqueda del valor de la muestra por el método de aproximaciones sucesivas, presenta la importante ventaja de un tiempo de conversión fijo y rápido, característica fundamental en determinadas aplicaciones donde se precisa rapidez e intervalos

regulares de tiempos entre muestras. En la Figura 8.52 se muestra el esquema de un convertidor de este tipo.

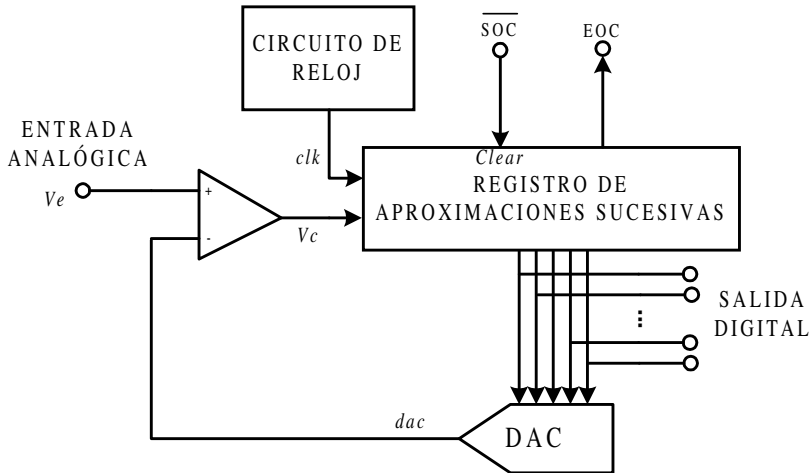


Figura 8.52. ADC por aproximaciones sucesivas

El principio de búsqueda por aproximaciones sucesivas se basa en dividir el rango de búsqueda entre dos para eliminar una de las dos mitades (la que no cumple la condición especificada) y continuar el procedimiento de forma recursiva sobre la otra mitad hasta obtener el código digital correspondiente a  $V_e$ .

El registro de aproximaciones sucesivas, elemento esencial de este convertidor, está compuesto internamente por un registro de desplazamiento  $RD$  y una lógica de control asociada. En la Figura 8.53 se muestra la evolución del registro de desplazamiento en función del resultado del comparador para el ejemplo que se muestra a continuación sobre un convertidor ADC de tres bits:

### Ejemplo:

Para el valor equivalente  $V_e = 101$ ; el proceso de comparación sería el siguiente:

- 1) Valor inicial del  $RD = 100$  (Bit  $MSB = 1$ , resto a cero).

Se compara (Valor = 101)  $\geq$  100.

Si el resultado de la comparación es **verdadero** el nuevo rango de búsqueda es 100 - 111.

Si el resultado de la comparación es **falso** el nuevo rango de búsqueda es 000 - 011.

- 2) Al ser verdadero, se modifica el bit ( $MSB - 1$ ) = 1 en  $RD = 110$ . Nuevo rango 100 - 111.



Se compara (Valor = 101)  $\geq$  110.

Si el resultado de la comparación es **verdadero** el nuevo rango de búsqueda es 110 - 111.

Si el resultado de la comparación es **falso** el nuevo rango de búsqueda es 100 - 101.

- 3) Al ser falso, se modifica el bit ( $MSB - 1$ ) = 0 y ( $MSB - 2$ ) = 1 en  $RD = 101$ . Nuevo rango 100 - 101.

Se compara (Valor = 101)  $\geq$  101.

Si el resultado de la comparación es **verdadero** el resultado que se obtiene es 101.

Si el resultado de la comparación es **falso** el resultado que se obtiene es 100.

**Verdadero: Resultado final =101.**

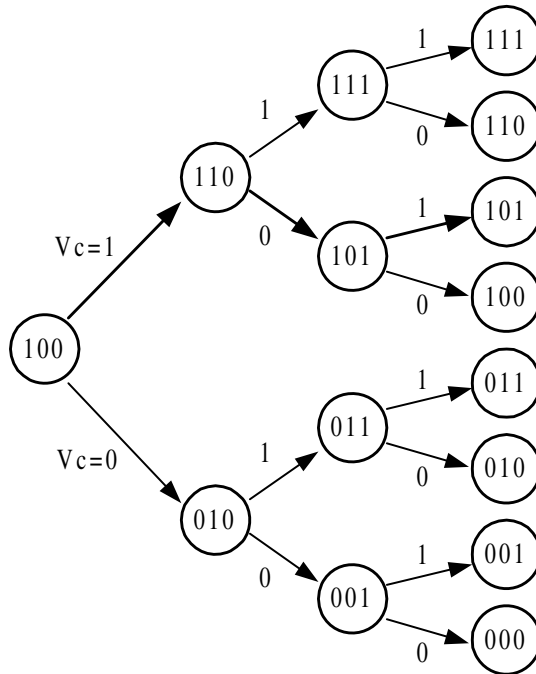


Figura 8.53. Diagrama de evolución del RD en un registro de aproximaciones sucesivas de tres bits

En términos generales, el funcionamiento interno del registro de desplazamiento RD opera del siguiente modo:

- 1) Registro de desplazamiento de  $N$  bits a cero.  $RD = 0$ .
- 2) Se fija el valor del bit  $MSB = B(N-1) = 1$ .  
Se realiza la comparación. ( $V_e > dac$ ):
  - a) Si el resultado de la comparación es verdadero,  $B(N-1) = 1$ .
  - b) Si el resultado de la comparación es falso,  $B(N-1) = 0$ .
  - c) Seguidamente se fija el bit  $B(N-2) = 1$  y se realizan los pasos a), b) y c) descritos anteriormente.
- 3) Para el proceso genérico sobre el bit  $B(i)$ ; se fija inicialmente el valor  $B(i) = 1$ .  
Se realiza la comparación:
  - a) Si el resultado de la comparación es verdadero,  $B(i) = 1$ .
  - b) Si el resultado de la comparación es falso,  $B(i) = 0$ .
  - c) Seguidamente se fija el bit  $B(i-1) = 1$  y se realizan los pasos a), b) y c) descritos anteriormente hasta que  $i = 0$ .

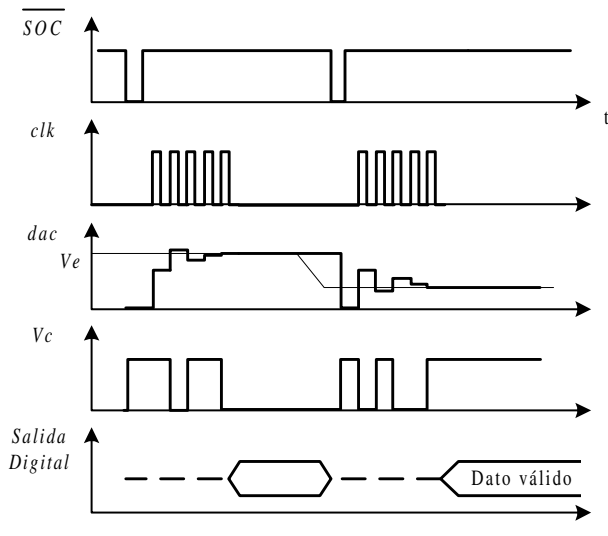


Figura 8.54. Cronograma del ADC por aproximaciones sucesivas

A efectos de diseño, la resolución  $a$  del DAC debe coincidir con un intervalo de cuantificación  $q$ , para que la salida digital proporcione directamente el código equivalente al número de intervalos de cuantificación de las que consta la señal analógica. La frecuencia máxima de reloj  $F_{clk}$  debe tener un valor máximo impuesto por la función inversa de la suma de los retardos y tiempos de conmutación acumulados a lo largo del bucle cerrado.

$$a = q = LSB \quad [8.60]$$

$$T_c = N T_{clk} = cte$$

$$F_c \leq \frac{1}{T_c} \leq \frac{1}{N T_{clk}} \quad [8.61]$$

siendo,  $N$  el número de bits del convertidor.

Este tipo de convertidor presenta la ventaja del empleo de técnicas exclusivamente digitales y un tiempo de conversión fijo y relativamente rápido. De hecho es el tipo de convertidor A/D más empleado en aplicaciones de propósito general.

### 8.3.3.6 CONVERTIDORES A/D SIGMA-DELTA

A principios de la década de los noventa irrumpió este nuevo tipo de convertidores A/D que aportaba un mayor número de bits (mayor resolución) a un coste menor. Además, y debido a técnicas provenientes del tratamiento digital de señales se obtiene una reducción espectacular del ruido de cuantificación en la banda pasante de estos convertidores. Por otra parte, cabe comentar que no permiten multiplexación, aunque esta desventaja se compensa con su coste reducido que facilita la utilización de un convertidor por canal.

Debido a que internamente utilizan unas frecuencias de muestreo unas 20 veces superior a la de los convertidores clásicos, su utilización se está centrando principalmente en la banda de audio. En estas aplicaciones se pueden encontrar convertidores de un número de bits mayor de 16 bits; 18 bits y 24 bits son valores muy habituales.

En la Figura 8.55 se muestra el esquema de un ADC Sigma-Delta, que como puede verse se compone de las siguientes partes:

#### **Muestreador con realimentación**

La estructura del muestreador de 1 bit de tipo continuo con realimentación, se compone de un integrador de una señal de error: integral de (señal de entrada - señal de salida). La salida de este integrador se lleva a un comparador cuya salida solamente puede tomar los valores 0 o 1. Finalmente, esta salida se realimenta hacia la entrada a través de un DAC de 1 bit ( $+U$  o  $-U$ ).

Esta estructura genera una cadena de bits (salida en serie) a una frecuencia muy alta. La frecuencia de muestreo a utilizar con un convertidor clásico es  $f_s$  viene dada por la expresión [8.62].

$$f_s \geq 2BW \quad [8.62]$$

Los convertidores Sigma-Delta utilizan una técnica denominada sobremuestreo y que trabaja a una frecuencia de muestreo  $f_{sm}$ , dada por la expresión [8.63].

$$f_{sm} = k f_s \quad [8.63]$$

donde,  $m$  es un número mayor de 20.

Esta frecuencia de muestreo tan elevada permite simplificar el filtro *anti-aliasing* de entrada, siendo suficiente para la mayoría de los casos la utilización de un filtro de primer orden para evitar dicho efecto. Este bloque sobremuestreador, también se conoce como muestreador de 1 bit. Su salida es una señal modulada en anchura de pulso cuyo ancho es proporcional a la amplitud de la señal de entrada. Por ejemplo, cuando la entrada es continua y de amplitud igual a media escala, la salida está compuesta por igual número de unos que de ceros a una frecuencia  $m \cdot f_s$ .

### Filtro digital y diezmador

La salida serie a frecuencia  $k \cdot F_s$  se lleva a un sistema digital compuesto por un filtro digital y por un subsistema denominado diezmador que se encarga de reducir la frecuencia de  $k \cdot f_s$  a  $f_s$ . Esta reducción de frecuencia de muestreo tiene un alto interés, pues es la que permite reducir el ruido de cuantificación en la banda pasante,  $f_s/2$ .

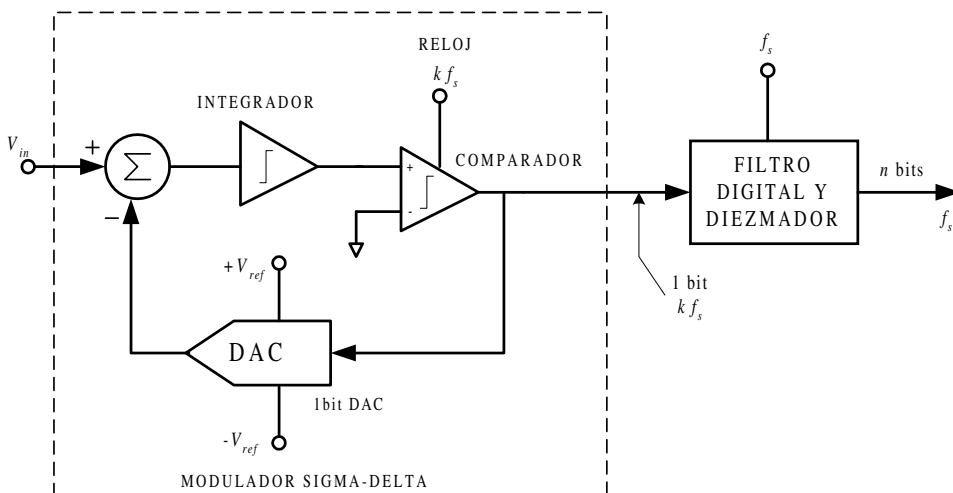


Figura 8.55. Convertidor A/D Sigma-Delta

La reducción de ruido de cuantificación que se consigue, se debe principalmente a los dos factores siguientes:

- En primer lugar, dicho ruido se reduce elevando el número de bits de la conversión y estos convertidores siempre utilizan resoluciones mayores de 16 bits.
- En segundo lugar, y una vez fijada la resolución, se tiene que el ruido de cuantificación se distribuye de una forma aproximadamente uniforme sobre toda la banda pasante.

En la expresión [8.64] se muestra la raíz cuadrática media del ruido de cuantificación.

$$\text{RMS del ruido de cuantificación} = \frac{q}{\sqrt{12}} \quad [8.64]$$

En la Figura 8.56 se muestra de forma gráfica el ruido de cuantificación.

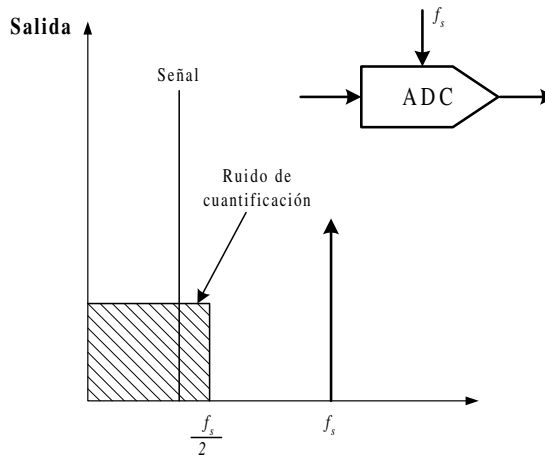


Figura 8.56. Ruido de cuantificación

Como se puede observar, al aumentar la frecuencia de muestreo en un factor  $k$  se consigue reducir en esa misma proporción el ruido de cuantificación, tal como se muestra en la Figura 8.57. Además, la actuación del filtro digital permite reducir todavía más dicho ruido.

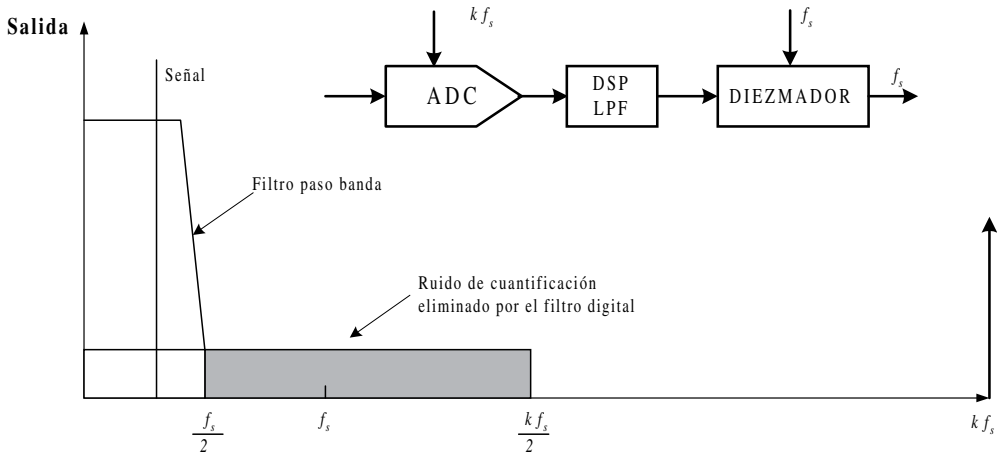


Figura 8.57. Reducción del ruido de cuantificación aumentando la frecuencia de muestreo

### 8.3.4 El convertidor comercial ADC 0805

En la Figura 8.58 se muestra la estructura interna del convertidor ADC0805 que se corresponde con un tipo de convertidor de 8 bits con estructura de aproximaciones sucesivas y un tiempo de conversión  $T_c$  de 100  $\mu$ s aproximadamente.

El chip dispone de los siguientes terminales de conexión:

- **/CS (Chip Select)**. Terminal de **Selección de chip** como dispositivo periférico bajo arquitectura de un microprocesador  $\mu P$  o microcontrolador  $\mu C$ . Activo a nivel bajo, se requiere que este terminal esté seleccionado para su correcto funcionamiento. En caso contrario, todas las entradas no serán operativas y las salidas permanecerán en estado de alta impedancia (HZ).
- **/WR (Write)**. Terminal de escritura en dispositivo. La orden de **inicio de conversión** se realiza activando este terminal a nivel bajo.
- **/INTR**. Coincidiendo con el inicio de conversión, esta salida pasa a nivel alto y el flanco de bajada indica el **fin de conversión**.
- **/RD (Read)**. Terminal de lectura en dispositivo. Mientras este terminal esté inactivo, nivel alto, el *bus* de datos de salida se encuentra en estado de alta impedancia. Al aplicar un nivel bajo sobre este terminal se habilitan los *buffers* de salida y la información binaria está disponible para su lectura, momento en que la señal **/INTR** pasa nuevamente a nivel alto.
- **CLK (Clock)**. El reloj del sistema puede aplicarse como un reloj externo sobre el terminal 4 y masa o bien mediante un circuito RC aplicando la resistencia

entre los terminales 19 y 4; y el condensador entre el terminal 4 y masa. En tal caso la frecuencia de reloj  $CLK$  viene determinada por la expresión [8.65] que se muestra a continuación.

$$F_{clk} = \frac{1}{Ln3 \cdot RC} \tag{8.65}$$

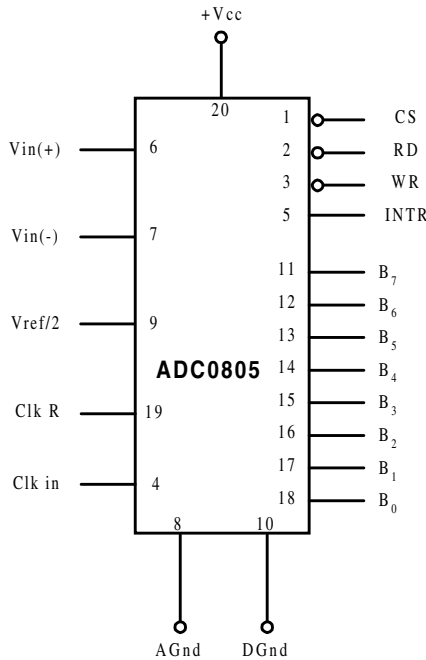


Figura 8.58. Circuito integrado ADC0805

- **Entrada analógica.** Los terminales  $V_{in}(+)$  y  $V_{in}(-)$  representan los valores máximos y mínimos del rango de fondo de escala  $FS$ . La tensión aplicada sobre ambos terminales no debe exceder los límites de la alimentación:  $V_{CC}$  y 0 V. Esto implica que se trata de un ADC unipolar.

$$FS = V_{in}(+) - V_{in}(-)$$

$$q = \frac{FS}{256} = \frac{V_{in}(+) - V_{in}(-)}{256} \tag{8.66}$$

- **Tensión de referencia.** El terminal 9 ( $V_{ref}/2$ ) fija internamente una referencia igual a  $V_{CC}/2$ . Mediante la aplicación de una tensión de referencia externa

pueden ser modificados los límites del fondo de escala  $FS$  establecidos en la expresión [8.66].

La Figura 8.59 muestra la aplicación típica de este convertidor en su conexión a un circuito microprocesador.

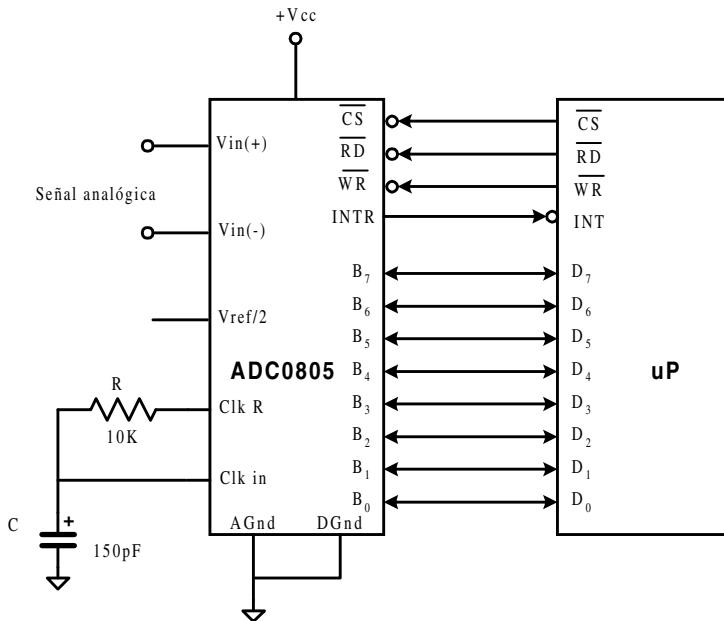


Figura 8.59. Circuito de aplicación típica del ADC0805

Comercialmente existe también el ADC0808 que dispone de ocho canales de entrada multiplexados, siendo su funcionamiento muy similar al ADC 0805 aquí descrito.

### PROBLEMA RESUELTO 8-11



Simular el funcionamiento de un ADC genérico de 8 bits similar al ADC0805 mostrado en el esquema de la Figura 8.58, utilizando para ello el programa *Electronics Workbench*.

- a) Calcular el valor del intervalo de cuantificación  $q$ .

#### Datos para el cálculo:

Fondo de escala  $FS = 5 \text{ V}$ .



**Solución:**

- a) El valor del intervalo de cuantificación  $q$  se determina a partir de la expresión [8.41].

$$q = \frac{V_{imax} - V_{imin}}{N} = \frac{FS}{2^n} = \frac{5}{256} = 19,53 \text{ mV}$$

El dispositivo genérico ADC incluido en el simulador *Electronics Workbench* dispone de los siguientes terminales de control:

- ◆ **VIN**. Terminal de entrada de la tensión analógica.
- ◆ **VREF+**, **VREF-**. Margen dinámico de conversión. En el ejemplo  $VREF+ = 5 \text{ V}$  y  $VREF- = 0 \text{ V}$ , lo que determina un rango unipolar con fondo de escala  $FS = 5 \text{ V}$ .
- ◆ **SOC**. Terminal de control de inicio de conversión. Terminal activo a nivel alto, '1' lógico.
- ◆ **DE**. Terminal de habilitación de los *buffers* de salida del contador. Terminal activo a nivel alto, '1' lógico.
- ◆ **EOC**. Señal indicadora de fin de conversión.
- ◆ **D0, ... , D7**. Terminales de salida de los *buffers* del contador de impulsos, codificado en binario natural.

La ruta y el nombre del fichero que contiene el esquema de este circuito es la que se indica a continuación:

**D:\Ejemplos\Cap08\Ewb5\08W0\_\_11.ewb**

Como ejercicio para el lector se debe variar en dicho circuito la tensión aplicada a la entrada del convertidor  $V_{in}$ , modificando el valor de la fuente de tensión. Iniciar la conversión mediante el proceso *STOP/SOC* observando la evolución del contador de pulsos sobre los indicadores de siete segmentos.

Se debe tener en cuenta que el contador de pulsos siempre introduce una imprecisión de  $\pm 1$  pulso de reloj, razón por la cual el indicador puede fluctuar en  $\pm 1$  unidad.

- Entrada = +5 V para obtener el valor FFh (255 pulsos).
- Entrada = +2,5 V para obtener el valor 80 (128 pulsos).
- Entrada = +1,25 V para obtener el valor 40 (64 pulsos).
- Introducir otros valores de  $V_{in}$ , y comprobar el valor obtenido a la salida del contador.

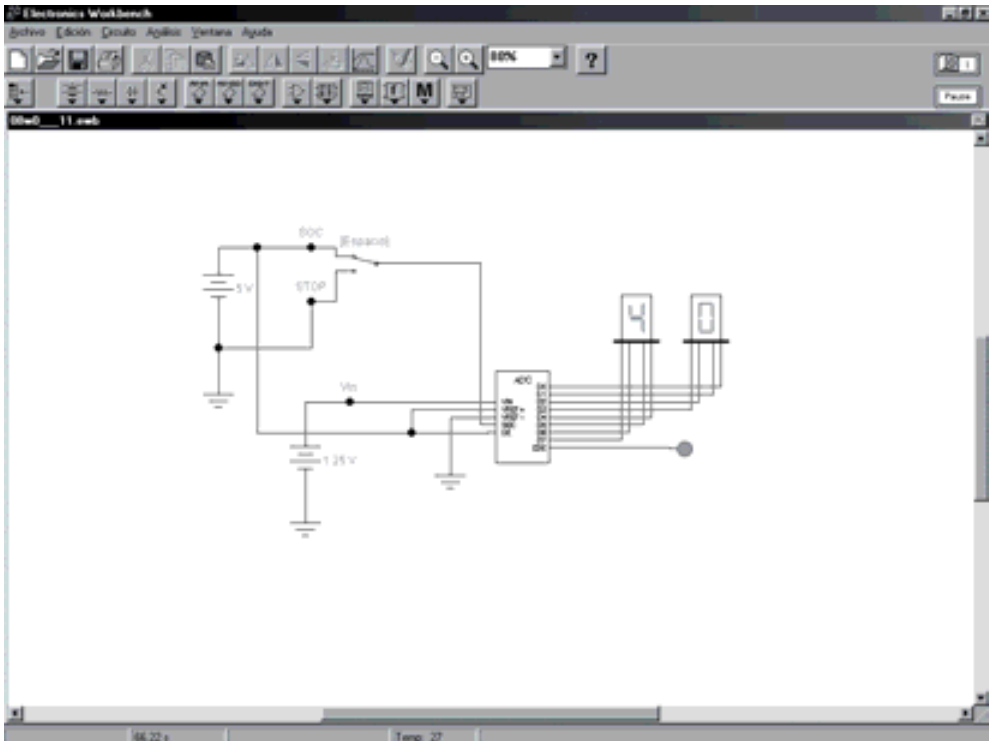


Figura 8.60. Circuito para simular el funcionamiento de un ADC genérico integrado

## PROBLEMAS PROPUESTOS

- 8-1) Calcular en un convertidor DAC de 10 *bits* de resistencias ponderadas, el valor de los parámetros de resolución  $a$ , fondo de escala  $FS$  y tensión máxima de salida del convertidor  $V_{omax}$ . Considérese los datos que se indican a continuación:  $V_{ref} = -10\text{ V}$ ;  $R = 10\text{ k}\Omega$  y  $R_f = 5\text{ k}\Omega$ .
- 8-2) En el esquema de la red en escalera de la Figura 8.12 la corriente que circula por la resistencia asociada al bit de menor peso (*LSB*) es de  $10\text{ }\mu\text{A}$ . Calcular el valor de la corriente correspondiente a la resistencia del bit de mayor peso (*MSB*).
- 8-3) En el convertidor DAC basado en red de resistencias ponderadas de la Figura 8.6, determinar el circuito de *offset* que permita una salida bipolar en el rango  $\pm 5\text{ V}$ . Considérese los siguientes datos:  $V_{ref} = 10\text{ V}$  y  $R_f = 5\text{ k}\Omega$ .
- 8-4) Calcular en un convertidor ADC por rampa en escalera, de 12 bits y una frecuencia de reloj  $f_{clk}$  de 1 MHz:
  - a) Tiempo máximo de conversión.

- b) Tiempo medio de conversión.
  - c) Frecuencia máxima de conversión.
- 8-5) Calcular en un convertidor ADC por rampa en escalera, de 10 bits la frecuencia de reloj  $f_{clk}$  necesaria para obtener un mínimo de 10.000 muestras digitalizadas de la señal analógica de entrada.
- 8-6) Partiendo de un convertidor ADC por rampa en escalera, de 10 bits realizar las modificaciones necesarias para un funcionamiento continuo a partir de un primer y único pulso  $sc$  (*start of conversion*).
- 8-7) Calcular la máxima pendiente de seguimiento de la señal analógica de entrada en un convertidor ADC continuo de 10 bits, con un valor de intervalo de cuantificación  $q$  de 1 mV y una frecuencia de reloj  $f_{clk}$  de 1 MHz.
- 8-8) Calcular el tiempo de conversión y el número máximo de muestras por segundo de la señal analógica de entrada, en un convertidor ADC de aproximaciones sucesivas de 12 bits, cuando la frecuencia de reloj  $f_{clk}$  es de 1 MHz.

## CONEXIÓN CON LÓGICA DIGITAL INTEGRADA

---

---

En el desarrollo de los contenidos genéricos de los materiales que abarcan la **“Electrónica Digital. Teoría, Problemas y Simulación”** se ha visto la necesidad de dividir el texto en **dos tomos**. El objetivo de tal división es hacerlos más manejables, ya que el número de páginas y contenidos es considerable; además de los textos teóricos, se incluyen problemas resueltos y propuestos, comprobados a partir del circuito esquemático o/y descritos en lenguaje VHDL para ser simulados e implementados.

Así, se ha considerado que este **primer tomo**, titulado **“Electrónica Digital. Introducción a la Lógica Digital. Teoría, Problemas y Simulación”**, presente un carácter más analítico y de introducción a los fundamentos de diseño de electrónica digital y el **segundo tomo**, titulado **“Electrónica Digital. Lógica Digital Integrada. Teoría, Problemas y Simulación”**, esté más orientado a la síntesis e implementación, mediante el uso de varias herramientas existentes actualmente en el mercado.

Desde el punto de vista de las herramientas utilizadas para resolver problemas y describir el *hardware* de los circuitos lógicos mediante código VHDL, en el primer tomo están orientadas sólo a la simulación, mientras que en el segundo tomo las descripciones y soluciones adoptadas están orientadas al diseño y la síntesis; es decir, condicionadas a que sea fácil y/o posible la realización física del sistema digital.

Dependiendo de los conocimientos que tenga el lector y el uso o aplicación que le quiera dar a la Electrónica Digital, le puede ser útil un tomo u otro, o los dos. Así, este **primer tomo** está orientado a personas sin conocimientos de electrónica digital o que, teniendo estos conocimientos, quieran profundizar en los fundamentos más teóricos,

que son la base de la Lógica Digital: sistemas de numeración, codificación de la información, álgebra de *Boole*, funciones o puertas lógicas, simplificación de funciones, tecnologías de circuitos integrados digitales, aritmética binaria y convertidores A/D y D/A. Según el lector vaya adquiriendo estos conocimientos los podrá ir verificando con los ejercicios propuestos a lo largo del libro, utilizando las herramientas de simulación y de descripción que se le proponen e incluyen en el CDROM#1 que acompaña a este libro, en componentes lógicos de pequeña escala de integración (SSI). Igualmente, en el CDROM#2 podrá encontrar circuitos digitales comerciales que le permitirán aproximar los contenidos teóricos y simulados del libro a componentes reales y comerciales.

El **segundo tomo**, titulado “**Electrónica Digital. Lógica Digital Integrada. Teoría, Problemas y Simulación**”, está orientado a aquellos lectores que, con unos conocimientos básicos de Lógica Digital, estén más interesados en el diseño y su realización física. El lector irá adquiriendo con los ejercicios propuestos, mediante el uso de las herramientas de simulación y síntesis, los conocimientos y métodos procedimentales, de implementación de sistemas digitales, con componentes de media y gran escala de integración (MSI y LSI).

Los **temas tratados en este segundo tomo** son: circuitos combinatoriales, circuitos secuenciales, memorias de semiconductores y dispositivos lógicos programables (PLD).

## FUENTES Y GENERADORES

---

---

***Objetivos:***

- Conocer los parámetros de ajuste y configuración de las fuentes y generadores básicos de señal.

***Contenido:*** Relación de fuentes y generadores de señal habituales en los laboratorios de electrónica, con la descripción de los principales parámetros para el ajuste de la señal.

***Simulación:*** Enumeración de las distintas fuentes y generadores básicos de señal disponibles en el simulador *Electronics Workbench*, describiendo su uso.



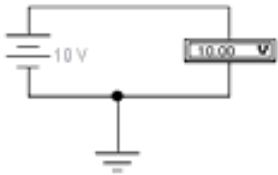
## A.1 FUENTES ANALÓGICAS

### A.1.1 Fuente de continua

- **De tensión**

En la Tabla A.1 se representa el símbolo de una fuente de tensión continua, su único parámetro ajustable, el valor de la tensión, y su comprobación mediante un voltímetro de continua en el simulador *Electronics Workbench*.

Tabla A.1. Fuentes de tensión continua

Símbolo	Parámetros ajustables	Simulación
		

- **De corriente**

En la Tabla A.2 se representa el símbolo de una fuente de corriente continua, su único parámetro ajustable, el valor de la corriente, y su comprobación mediante un amperímetro de continua en el simulador *Electronics Workbench*.

### A.1.2 Fuentes y generadores de alterna

- **Fuente de tensión senoidal**

En la Tabla A.3 se representa el símbolo de una fuente de tensión senoidal. Sus parámetros ajustables son: amplitud, frecuencia y fase.

La comprobación de la amplitud (en valores eficaces 10 V) se realiza mediante un voltímetro de alterna. Para medir todos sus parámetros de ajuste se utiliza un osciloscopio. Los resultados de la simulación de estas dos medidas, con voltímetro y osciloscopio, se obtienen con el simulador

*Electronics Workbench* y están representadas en la columna de simulación de la Tabla A.3.

Se aprecia cómo el valor de la tensión pico visualizada en el osciloscopio es la  $\sqrt{2}$  de su valor eficaz.

Tabla A.2. Fuentes de corriente continua



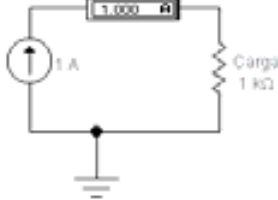


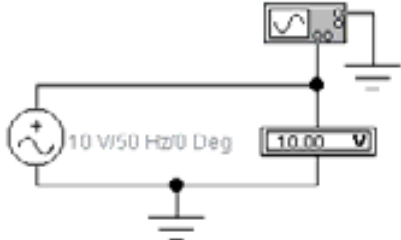
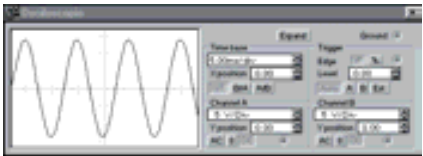
Símbolo	Parámetros ajustables	Simulación
		

Tabla A.3. Fuentes de tensión senoidal

Símbolo	Parámetros ajustables	Simulación
		 



- **Fuente de corriente senoidal**


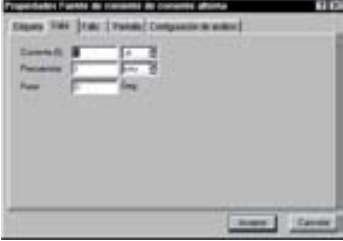
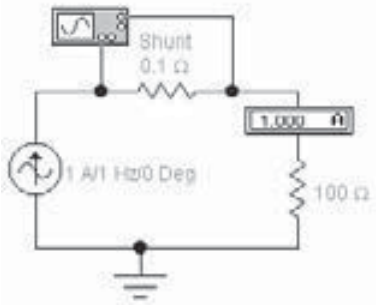
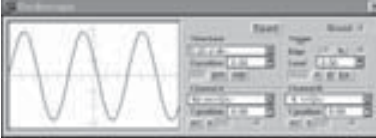
En la Tabla A.4 se representa el símbolo de una fuente de corriente senoidal. Sus parámetros ajustables son: corriente, frecuencia y fase.

La comprobación de la corriente (en valores eficaces) se realiza mediante un amperímetro de alterna. Para medir todos sus parámetros de ajuste se coloca un *shunt* (resistencia de muy bajo valor para no influir en el circuito) situada en la rama en la que se quiere medir la corriente. El osciloscopio, en los extremos de la **resistencia *shunt*** mide una tensión proporcional a la corriente que circula por ella. Los resultados de la simulación de estas dos medidas, con amperímetro y osciloscopio, se realiza con el simulador *Electronics Workbench* y están representadas en la columna de simulación de la Tabla A.4.

La corriente tiene la misma forma de onda que la tensión visualizada en el osciloscopio pero con diferente escala, debido a que la corriente es igual a la tensión dividida entre el valor constante de la resistencia *shunt*.

Se observa en el osciloscopio un valor de pico de tensión de 0,1414 V, que dividido entre la resistencia *shunt* de 0,1  $\Omega$  permite obtener el valor pico de corriente de 1,414 A, el cual corresponde a un valor eficaz de 1 A.

Tabla A.4. Fuentes de corriente senoidal

Símbolo	Parámetros ajustables	Simulación
		 

- **Generador de funciones**

El generador de funciones es un equipo de instrumentación de laboratorio capaz de entregar en su salida una señal periódica, cuyos parámetros ajustables, accesibles en la carátula del instrumento, son los siguientes:

- ◆ **Tipo de función** (*Function*). Generalmente se puede obtener ondas senoidales, triangulares y rectangulares.
- ◆ **Frecuencia** (*Frequency*). Este ajuste determina el número de ciclos generados por segundo.
- ◆ **Ciclo de trabajo** (*Duty cycle*). Controla la simetría de las formas de onda. En el simulador *Electronics Workbench* sólo tiene efecto en las ondas rectangulares y triangulares.

Las ondas rectangulares se explican en el apartado A.2 Fuentes digitales. En ondas triangulares, el ajuste controla el tanto por ciento del ciclo en el que se produce la pendiente de subida. Un 50% de ciclo de trabajo proporciona ondas triangulares con igual pendiente de subida que de bajada. Para valores extremos de este parámetro, 1% y 99%, se producen ondas en diente de sierra. En la Figura A.1 se ilustran tres valores del ciclo de trabajo: 1%, 50% y 99%.

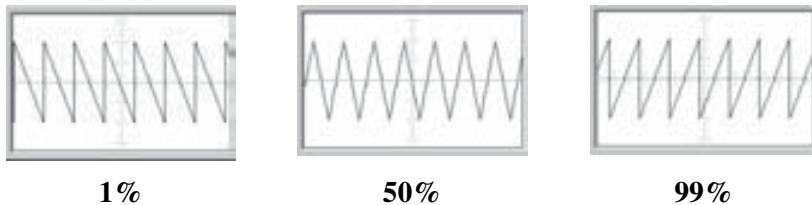


Figura A.1. Onda triangular con tres valores distintos de ciclo de trabajo

- ◆ **Amplitud** (*Amplitude*). Determina el nivel de tensión o amplitud de la onda seleccionada.
- ◆ **Desplazamiento** (*Offset*). Añade una componente continua produciendo un desplazamiento positivo o negativo en la onda alterna con respecto al nivel de referencia 0 V.

En la Figura A.2 se representa una señal senoidal de periodo  $T$  o frecuencia  $f = 1/T = 1$  kHz, con un ciclo de trabajo del 50%, una amplitud de 10 V y un desplazamiento de 5 V. Se puede apreciar la comprobación de todos los parámetros ajustables con la ayuda del osciloscopio del simulador *Electronics Workbench*.

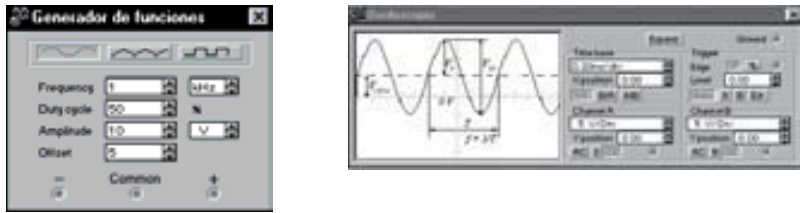


Figura A.2. Detalle de una onda senoidal

En la Tabla A.5 se representa el símbolo de un generador de tensión senoidal, sus parámetros ajustables y su comprobación mediante el osciloscopio en el simulador *Electronics Workbench*.

Tabla A.5. Generador de tensión senoidal

Símbolo	Parámetros ajustables	Simulación

De la misma forma se pueden obtener ondas triangulares. En la Tabla A.6 se representa un generador de tensión triangular con una componente continua o desplazamiento de 5 V.

## A.2 FUENTES DIGITALES

### Fuente de onda cuadrada (Reloj)

- En la Tabla A.7 se representa el símbolo de un generador de una onda cuadrada o reloj. Los parámetros ajustables son: frecuencia, ciclo de trabajo y amplitud. Su comprobación se realiza mediante el osciloscopio del simulador *Electronics Workbench*.

Tabla A.6. Generador de tensión triangular

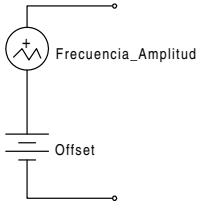




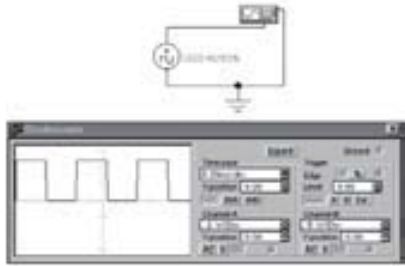
Símbolo	Parámetros ajustables	Simulación
		

Tabla A.7. Onda cuadrada. Señal digital de reloj

Símbolo	Parámetros ajustables	Simulación
		

- **Señal de reloj mediante el generador de funciones**

En el generador de funciones se puede obtener también una señal digital o de reloj, seleccionando el tipo de onda rectangular, ajustando la frecuencia, la amplitud (considerando sólo dos estados de tensión, el valor máximo  $H$  y el valor mínimo  $L$ ), el desplazamiento y el ciclo de trabajo.

El ciclo de trabajo (*Duty cycle*), en las ondas rectangulares, determina la proporción del ciclo en que la onda está a nivel alto. Un 50% de ciclo de trabajo proporciona ondas rectangulares con semiperiodos iguales, es decir, ondas cuadradas. El rango se puede ajustar desde 1% hasta 99%. En la Figura A.3 se ilustran tres valores del ciclo de trabajo: 1% , 50% y 99%.

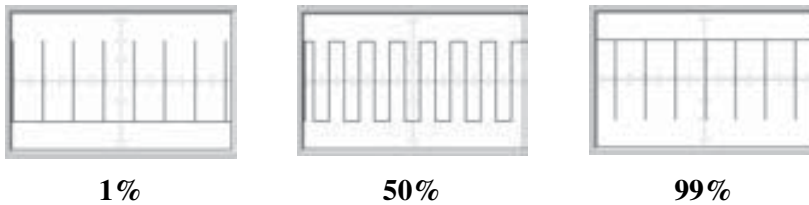


Figura A.3. Señal rectangular con tres valores distintos de ciclo de trabajo

En la Tabla A.8 se representa un generador de tensión rectangular con un ciclo de trabajo del 80%.

Tabla A.8. Generador de tensión rectangular

Símbolo	Parámetros ajustables	Simulación

### A.3 FUENTES DEPENDIENTES



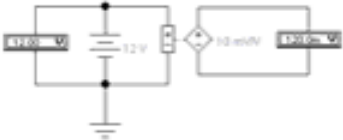
- Fuente de tensión dependiente de tensión (V/V)

En una fuente de tensión-controlada por tensión, el voltaje de salida depende del voltaje aplicado en los terminales de entrada. La relación entre el voltaje de salida y entrada define el parámetro de **ganancia de voltaje  $E$**  cuyo rango puede variar entre relaciones de mV/V hasta kV/V.

$$E = \frac{V_{out}}{V_{in}}$$

En la Tabla A.9 se representa el símbolo de una fuente de tensión controlada por tensión. El parámetro ajustable es la ganancia de voltaje  $E$ . Su comprobación se realiza mediante el voltímetro del simulador *Electronics Workbench*.

Tabla A.9. Fuente de tensión controlada por tensión

Símbolo	Parámetros ajustables	Simulación
		

- Fuente de corriente dependiente de tensión (I/V)**

En una fuente de corriente-controlada por tensión, la corriente de salida depende del voltaje aplicado a los terminales de entrada. La relación entre la corriente de salida y el voltaje de entrada define el parámetro de **transconductancia  $G$** . Su unidad es el **mhos** (también conocido como **siemens**) y puede tener cualquier rango comprendido entre mmhos y kmhos.

$$G = \frac{I_{out}}{V_{in}}$$

En la Tabla A.10 se representa el símbolo de una fuente de corriente controlada por tensión. El parámetro ajustable es el de transconductancia  $G$ . Su comprobación se realiza mediante el voltímetro y amperímetro del simulador *Electronics Workbench*.

- Fuente de tensión dependiente de corriente (V/I)**

En una fuente de tensión-controlada por corriente, el voltaje de salida depende de la corriente aplicada a los terminales de entrada. La relación entre el voltaje de salida y la corriente de entrada define el parámetro **transresistivo  $H$**  cuyo rango puede variar entre mV/A y kV/A.

$$H = \frac{V_{out}}{I_{in}}$$

En la Tabla A.11 se representa el símbolo de una fuente de tensión controlada por corriente. El parámetro ajustable es la transresistencia  $H$ . Su comprobación se realiza mediante el voltímetro y amperímetro del simulador *Electronics Workbench*.

Tabla A.10. Fuente de corriente controlada por tensión



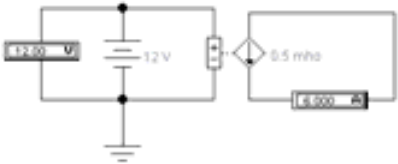


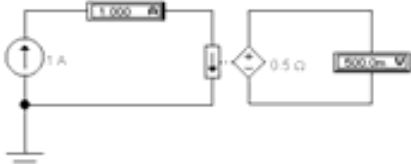
Símbolo	Parámetros ajustables	Simulación
		

Tabla A.11. Fuente de tensión controlada por corriente

Símbolo	Parámetros ajustables	Simulación
		

- **Fuente de corriente dependiente de corriente (I/I)**



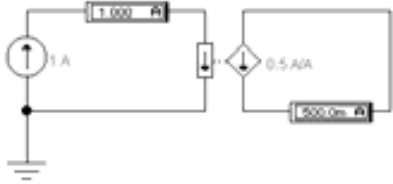
En una fuente de corriente-controlada por corriente, la corriente de salida de este dispositivo depende de la corriente aplicada a los terminales de entrada. La relación entre la corriente de salida y la corriente de entrada depende del parámetro denominado **ganancia de corriente**  $F$  cuyo rango puede variar entre mA/A y kA/A.

$$F = \frac{I_{out}}{I_{in}}$$

En la Tabla A.12 se representa el símbolo de una fuente de corriente controlada por corriente. El parámetro ajustable es la ganancia de corriente  $F$ .

Su comprobación se realiza mediante los amperímetros del simulador *Electronics Workbench*.

Tabla A.12. Fuente de corriente controlada por corriente

Símbolo	Parámetros ajustables	Simulación
		

- **Fuente senoidal dependiente de tensión (VCO Senoidal)**

Este dispositivo, denominado oscilador controlado por tensión (VCO), considera la entrada de señal alterna (AC) o continua (DC) como variable independiente. Su salida es una señal sinusoidal de amplitud constante y frecuencia directamente proporcional a la amplitud instantánea de la señal aplicada a su entrada. Esta entrada es muy útil considerándola como voltaje de mando o de control, aplicando diferentes tipos de señales, tales como:

- una **señal continua** (DC), pudiendo esta fuente ser usada como **generador de frecuencia sinusoidal** gobernado a través de un potenciómetro que proporciona la tensión continua.
- una **señal alterna** (AC), obteniendo en la salida de la fuente una señal portadora modulada en frecuencia (**FM, Frequency Modulation**). Tiene aplicación en técnicas de modulación analógica.
- una **onda cuadrada**, obteniendo en la salida de la fuente una señal portadora modulada digital en frecuencia (**FSK, Frequency Shift Keying**). Tiene aplicación en técnicas de modulación digital.
- una **onda triangular o diente de sierra**, obteniendo en la salida de la fuente un **barrido lineal de frecuencias**. Tiene aplicación en instrumentación para determinar la respuesta de los circuitos en función de la frecuencia y conocer el ancho de banda del dispositivo.
- la señal de **salida de un oscilador enganchado en fase (PLL, Phase-Locked Loop)**, obteniendo en la salida de la fuente una reproducción exacta de la señal aplicada a la entrada del PLL. Tiene aplicación en sintetizadores de frecuencia, demoduladores síncronos, etc.



La relación entre la frecuencia de salida y la tensión de entrada depende del parámetro  $K$ , cuyo rango puede variar entre Hz/V y kHz/V.

$$K = \frac{F_{out}}{V_{in}}$$

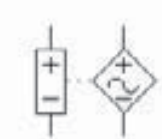

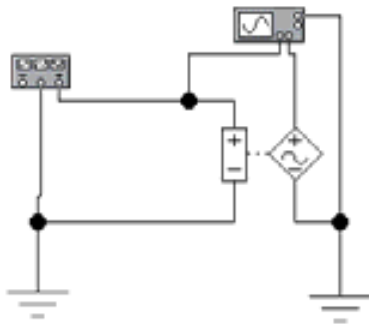
En la Tabla A.13 se representa el símbolo de un generador sinusoidal controlado por tensión. Los parámetros ajustables se encuentran en el menú: **Circuito/Propiedades de componentes/Modelos/Edición**, en la Hoja 1 y Hoja 2.

La amplitud de la onda de salida se expresa en voltios mediante los parámetros  $L$  y  $H$ .

La función  $K$ , que representa la relación entre frecuencia y tensión, se define mediante una tabla de  $N$  pares de valores  $C1-F1, C2-F2, \dots, Cn-Fn$ , donde  $F_i$  representa la frecuencia correspondiente a la tensión  $C_i$ .

Su comprobación se realiza mediante el osciloscopio del simulador *Electronics Workbench*.

Tabla A.13. Generador sinusoidal controlado por tensión (VCO Senoidal)

Símbolo	Parámetros ajustables	Simulación
		

En la Figura A.4 se representa un barrido de frecuencia sinusoidal generado a partir de una señal de control en diente de sierra. Se puede apreciar la comprobación de todos los parámetros ajustables con la ayuda del osciloscopio del simulador *Electronics Workbench*.

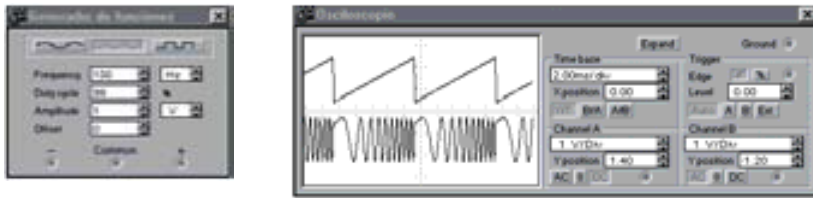


Figura A.4. Detalle de un barrido de frecuencia senoidal obtenido mediante una señal de control en diente de sierra

- **Fuente triangular dependiente de tensión (VCO Triangular)**

Este dispositivo es idéntico al VCO senoidal con la única diferencia de que la forma de onda generada en la salida es una señal triangular. La fuente VCO triangular toma una señal de corriente alterna (AC) o continua (DC) en la entrada como variable independiente, generando en su salida una señal triangular de amplitud constante y frecuencia directamente proporcional a la amplitud instantánea de la señal aplicada a su entrada. Esta entrada es muy útil considerándola como voltaje de mando o de control, aplicando diferentes tipos de señales, tales como:

- una **señal continua** (DC), pudiendo esta fuente ser usada como **generador de frecuencia triangular** gobernado a través de un potenciómetro.
- una **señal alterna** (AC), obteniendo en la salida de la fuente una señal portadora modulada en frecuencia (**FM, Frequency Modulation**).
- una **onda cuadrada**, obteniendo en la salida de la fuente una señal portadora modulada digital en frecuencia (**FSK, Frequency Shift Keying**).

La relación entre la frecuencia de salida y la tensión de entrada depende del parámetro  $K$ , cuyo rango puede variar entre Hz/V y kHz/V.

$$K = \frac{F_{out}}{V_{in}}$$

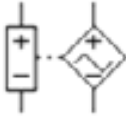

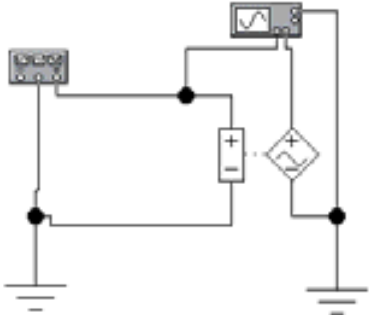
En la Tabla A.14 se representa el símbolo de un generador triangular controlado por tensión. Los parámetros ajustables se encuentran en el menú: **Circuito/Propiedades de componentes/Modelos/Edición**, en la Hoja 1 y Hoja 2.

La amplitud de la onda de salida se expresa en voltios mediante los parámetros  $L$  y  $H$ .

La función  $K$  que representa la relación entre frecuencia y tensión, se define mediante una tabla de  $N$  pares de valores  $C1-F1, C2-F2, \dots, Cn-Fn$ , donde  $F_i$  representa la frecuencia correspondiente a la tensión  $C_i$ .

Su comprobación se realiza mediante el osciloscopio del simulador *Electronics Workbench*.

Tabla A.14. Generador triangular controlado por tensión (VCO Triangular)

Símbolo	Parámetros ajustables	Simulación
		

En la Figura A.5 se representa una señal de salida con modulación FSK generada a partir de una señal de control en onda cuadrada. Se puede comprobar los parámetros ajustables con la ayuda del osciloscopio del simulador *Electronics Workbench*.

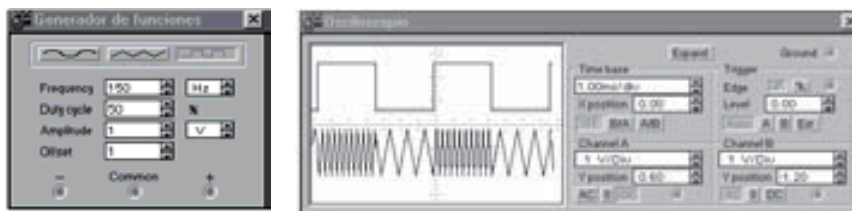


Figura A.5. Detalle con una onda de control cuadrada

- **Fuente rectangular dependiente de tensión (VCO Rectangular)**

Este dispositivo es idéntico al VCO senoidal con la única diferencia que la forma de onda generada en la salida es una señal rectangular. La fuente VCO rectangular toma una señal de corriente alterna (AC) o continua (DC) en la

entrada como variable independiente, generando en su salida una señal rectangular de amplitud constante y frecuencia directamente proporcional a la amplitud instantánea de la señal aplicada a su entrada. Esta entrada es muy útil considerándola como voltaje de mando o de control, aplicando diferentes tipos de señales, tales como:

- una **señal continua** (DC), pudiendo esta fuente ser usada como **generador de frecuencia rectangular** gobernado a través de un potenciómetro.
- una **señal alterna** (AC), obteniendo en la salida de la fuente una señal portadora modulada en frecuencia (**FM, Frequency Modulation**).
- una **onda cuadrada**, obteniendo en la salida de la fuente una señal portadora modulada digital en frecuencia (**FSK, Frequency Shift Keying**).

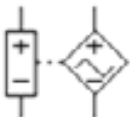

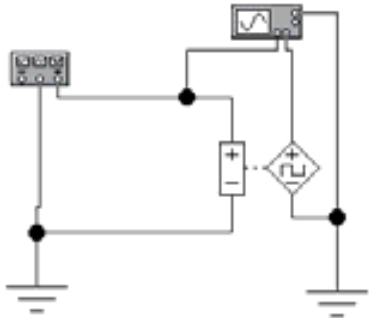
La relación entre la frecuencia de salida y la tensión de entrada depende del parámetro  $K$ , cuyo rango puede variar entre Hz/V y kHz/V.

$$K = \frac{F_{out}}{V_{in}}$$

En la Tabla A.15 se representa el símbolo de un generador rectangular controlado por tensión. Los parámetros ajustables se encuentran en el menú: **Circuito/Propiedades de componentes/Modelos/Edición**, en la Hoja 1 y Hoja 2.

La amplitud de la onda de salida se expresa en voltios mediante los parámetros  $L$  y  $H$ .

Tabla A.15. Generador rectangular controlado por tensión (VCO Rectangular)

Símbolo	Parámetros ajustables	Simulación
		

La función  $K$ , que representa la relación entre frecuencia-tensión, se define mediante una tabla de  $N$  pares de valores  $C1-F1, C2-F2, \dots, Cn-Fn$ , donde  $F_i$  representa la frecuencia correspondiente a la tensión  $C_i$ .

Otros parámetros ajustables son el ciclo de trabajo  $D$  y los tiempos de subida  $TR$  y bajada  $TF$  de la onda rectangular

Su comprobación se realiza mediante el osciloscopio del simulador *Electronics Workbench*.

En la Figura A.6 se representa una señal de salida con modulación FM generada a partir de una señal de control senoidal. Se puede comprobar los parámetros ajustables con la ayuda del osciloscopio del simulador *Electronics Workbench*.

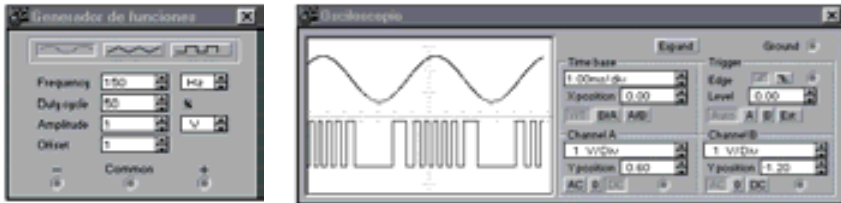


Figura A.6. Detalle con una onda de control senoidal

- **Fuente dependiente no lineal**

Una fuente dependiente no lineal genera una tensión o corriente controlada por varias entradas de tensión  $V(1), V(2), V(3), V(4)$  y de corriente  $I(5), I(6)$ . Su función de transferencia es una expresión matemática, que puede llegar a ser compleja al permitir usar los siguientes operadores:

+ - \* / ^ unary

y las funciones predefinidas:

<i>abs</i>	<i>asin</i>	<i>atanh</i>	<i>exp</i>	<i>sin</i>	<i>tan</i>
<i>acos</i>	<i>asinh</i>	<i>cos</i>	<i>ln</i>	<i>sinh</i>	<i>u</i>
<i>acosh</i>	<i>atan</i>	<i>cosh</i>	<i>log</i>	<i>sqrt</i>	<i>uramp</i>

La función no lineal  $u$  representa al **escalón unitario**, siendo su definición matemática la siguiente:

$$u(x) = \begin{cases} 1 & \text{para } x \geq 0 \\ 0 & \text{para } x < 0 \end{cases}$$

La función no lineal *uramp* representa la **rampa unitaria**. La función *uramp* es la integral de *u*, respecto del tiempo y su definición matemática es:

$$uramp(x) = \begin{cases} x & \text{para } x \geq 0 \\ 0 & \text{para } x < 0 \end{cases}$$

Si los argumentos de las funciones *ln*, *log* y *sqrt* fueran negativos se realiza el cálculo considerando el valor absoluto de dichos argumentos.

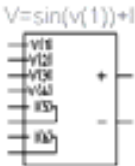

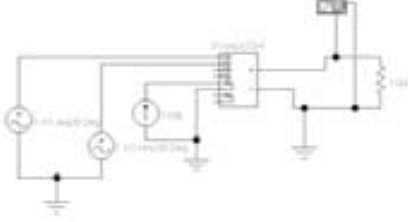
Si la variable dependiente se define con la letra *V* la salida será una tensión y si se define con la letra *I* la salida será una corriente.

En la Tabla A.16 se representa el símbolo de la fuente dependiente no lineal. Los parámetros ajustables se encuentran en el menú: **Circuito/Propiedades de componentes**, en la hoja denominada **Valor** (o haciendo doble clic sobre el símbolo de la fuente), donde se especifica la expresión matemática que define la función de transferencia dependiente de las tensiones *V*(1), *V*(2), *V*(3), *V*(4) y de las corrientes *I*(5), *I*(6).

La expresión definida para el ejemplo es la siguiente:

$$v = \sin(V(1)) + \log(V(2) + 1) + u(I(5) \cdot 100)$$

Tabla A.16. Fuente dependiente no lineal

Símbolo	Parámetros ajustables	Simulación
		

En la Figura A.7 se representa la señal de salida de la fuente dependiente no lineal obtenida en el osciloscopio del programa de simulación *Electronics Workbench*.

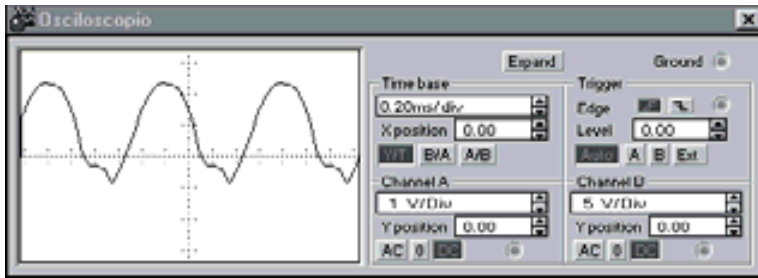


Figura A.7. Señal de salida de la fuente dependiente no lineal

- **Fuente polinomial**

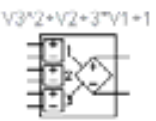

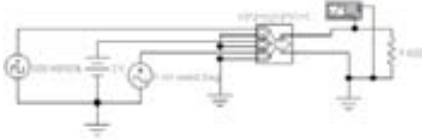
Una fuente polinomial genera una tensión controlada por tensión, cuya función de transferencia es polinomial.

El modelo usado por el programa de simulación *Electronics Workbench* define dicha función de transferencia polinomial mediante tres controles de entrada de tensión:  $V_1$ ,  $V_2$  y  $V_3$ .

En la Tabla A.17 se representa el símbolo de la fuente polinomial. Los parámetros ajustables se encuentran en el menú: **Circuito/Propiedades de componentes**, en la hoja denominada **Valor** (o haciendo doble clic sobre el símbolo de la fuente), donde se especifican los coeficientes  $A$  hasta  $K$  de los distintos términos del polinomio. Para la expresión del ejemplo se ha definido una tensión de salida mediante el polinomio:

$$S = V_3^2 + V_2 + 3 \cdot V_1 + 1$$

Tabla A.17. Fuente polinomial

Símbolo	Parámetros ajustables	Simulación
		

En la Figura A.8 se representa la señal de salida de la fuente polinomial obtenida en el osciloscopio del programa de simulación *Electronics Workbench*.

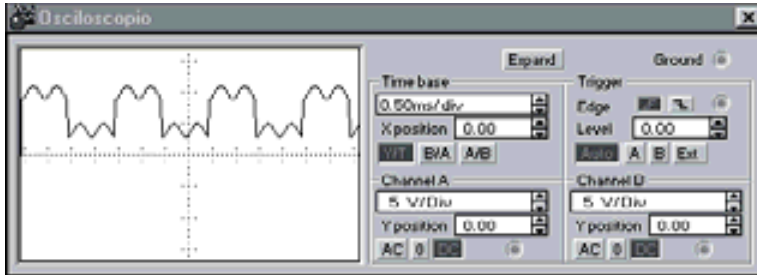


Figura A.8. Señal de salida de una fuente polinomial

## A.4 FUENTES CON MODULACIÓN

- **Fuente de amplitud modulada, AM**

Este dispositivo es un generador de portadora sinusoidal modulada en amplitud (**AM, Amplitude Modulation**). Internamente opera sobre la expresión analítica que define una portadora modulada en amplitud por un tono único, y que se muestra a continuación:

$$V_{out} = V_a \cdot \sin(2 \cdot \pi \cdot f_c \cdot TIME) \cdot (1 + m \cdot \sin(2 \cdot \pi \cdot f_m \cdot TIME)) \quad [A.1]$$

Donde las variables representan:

$V_a$ , amplitud de la señal portadora expresada en voltios (V).

$f_c$ , frecuencia de la señal portadora expresada en hercios (Hz).

$f_m$ , frecuencia de la señal moduladora expresada en hercios (Hz).

$m$ , índice de modulación ( $m = V_m/V_a$ ), donde  $V_m$  es la amplitud de la señal moduladora expresada en voltios.

$TIME$ , variable para prefija el tiempo de análisis del simulador *Electronics Workbench*.

En la Tabla A.18 se representa el símbolo de un generador AM. Los parámetros ajustables se encuentran en el menú: **Circuito/Propiedades de componentes**, en la hoja denominada **Valor** (o haciendo doble clic sobre el símbolo de la fuente), siendo éstos:

$V_C$ , amplitud de la señal portadora.





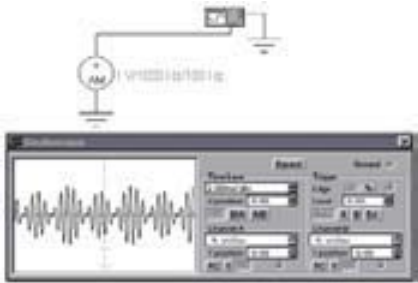
FC, frecuencia de la señal portadora.

M, índice de modulación.

FM, frecuencia de la señal moduladora.

Su comprobación se realiza mediante el osciloscopio del simulador *Electronics Workbench*.

Tabla A.18. Generador de amplitud modulada (AM)

Símbolo	Parámetros ajustables	Simulación
		

- **Fuente de frecuencia modulada, FM**

Este dispositivo es un generador de portadora sinusoidal modulada en frecuencia (**FM, *Frequency Modulation***). Internamente opera sobre la expresión analítica que define una portadora modulada en frecuencia (modulación del ángulo de fase) por un tono único, que se muestra a continuación:

$$V_{out} = V_a \cdot \sin(2 \cdot \pi \cdot f_c \cdot TIME + m \cdot \sin(2 \cdot \pi \cdot f_m \cdot TIME)) \quad [A.2]$$

Donde las variables representan:

$V_a$ , amplitud de la señal portadora expresada en voltios (V).

$f_c$ , frecuencia de la señal portadora expresada en hercios (Hz).

$f_m$ , frecuencia de la señal moduladora expresada en hercios (Hz).




$m$ , índice de modulación. Obsérvese que  $m$  es la amplitud de la señal moduladora expresada en voltios. Este parámetro también puede ser representado por:  $m = \text{Inc}f_c/f_m(\text{max})$ .

$TIME$ , variable del tiempo de análisis empleado por el simulador *Electronics Workbench*.

En la Tabla A.19 se representa el símbolo de un generador FM. Los parámetros ajustables se encuentran en el menú: **Circuito/Propiedades de componentes**, en la hoja denominada **Valor** (o haciendo doble clic sobre el símbolo de la fuente), siendo éstos:

- VA, amplitud de la señal portadora.
- FC, frecuencia de la señal portadora.
- M, índice de modulación.
- FM, frecuencia de la señal moduladora.

Tabla A.19. Generador de frecuencia modulada (FM)

Símbolo	Parámetros ajustables	Simulación
		

- **Fuente de modulación digital de frecuencia, FSK**

Este dispositivo es un modulador digital de frecuencia (**FSK, Frequency Shift Keying**). Se utiliza en transmisiones telegráficas o por teletipo, introduciendo una portadora senoidal y un desplazamiento de frecuencia correspondiente a **MARCA** cuando la información binaria es un uno lógico a la entrada y **ESPACIO** cuando es un cero lógico.

Este generador puede emular el funcionamiento de un módem de baja velocidad operando sobre la norma Bell 202. La **MARCA** se corresponde con un tono único de portadora de 1.200 Hz y el **ESPACIO** con un tono único de portadora de 2.200 Hz.

Internamente opera sobre la expresión analítica que define una portadora modulada en frecuencia (modulación del ángulo de fase) por un tono único que se muestra a continuación:

$$V_{out} = V_a \cdot \sin(2 \cdot \pi \cdot (f_c + \Delta f) \cdot TIME) \tag{A.3}$$

Donde las variables representan:

$V_a$ , amplitud de la señal portadora expresada en voltios (V).

$f_c$ , frecuencia de la señal portadora expresada en hercios (Hz).

$f_m$ , frecuencia de la señal moduladora expresada en hercios (Hz).

$TIME$ , variable del tiempo de análisis empleado por el programa de simulación *Electronics Workbench*.

En la Tabla A.20 se representa el símbolo de un generador FSK. Los parámetros ajustables se encuentran en el menú: **Circuito/Propiedades de componentes/**, en la hoja denominada **Valor** (o haciendo doble clic sobre el símbolo de la fuente), siendo éstos:


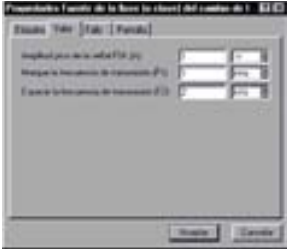
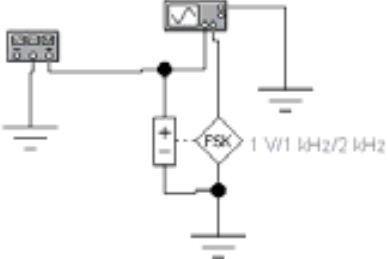
$A$ , amplitud de la señal modulada.

$F_1$ , frecuencia correspondiente a MARCA (“1” lógico).

$F_2$ , frecuencia correspondiente a ESPACIO (“0” lógico).

Se puede comprobar el efecto de los parámetros ajustables con la ayuda del osciloscopio del programa de simulación *Electronics Workbench*.

Tabla A.20. Generador de modulación digital de frecuencia (FSK)

Símbolo	Parámetros ajustables	Simulación
		

En la Figura A.9 se representa una modulación FSK correspondiente a una transmisión binaria de 200 b/s con una frecuencia de MARCA de 1 kHz y una frecuencia de ESPACIO de 2 kHz.

- **Fuente lineal de voltaje controlado (PWL)**

Una fuente lineal de voltaje controlado PWL permite definir la forma de onda de la señal mediante un fichero de texto en el que se especifican pares de

valores tensión-tiempo de la señal. Los valores intermedios, entre dos valores especificados se determinan mediante interpolación lineal.

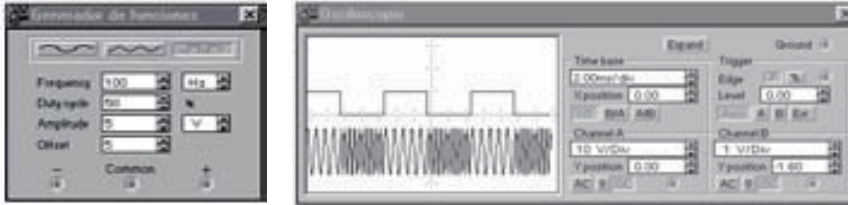


Figura A.9. Detalle de una señal con modulación FSK

La estructura del archivo de texto de especificación de la señal debe cumplir las siguientes condiciones:

- Cada línea del archivo representa a un punto de la señal.
- Su sintaxis o formato debe ser:  
Tiempo <espacio/s> voltaje
- Si no se especifica como primer punto de la señal el par 0 0 se considera éste como valor inicial.


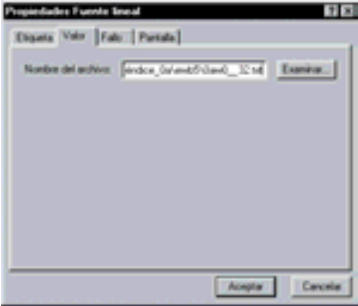
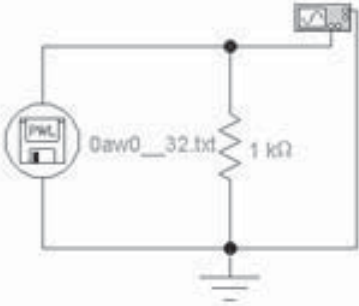
En la Figura A.10 se muestra un ejemplo de archivo de texto de especificación de una fuente PWL.

Time	Voltage
0	0
1e-03	0
1.01e-03	1
2e-03	0
3e-03	0
3.01e-03	1
4e-03	1
4.01e-03	2
5e-03	2
5.01e-03	1
6e-03	1
6.01e-03	0
7e-03	0
7.01e-03	-1
8e-03	-1
8.01e-03	-2
9e-03	-2
9.01e-03	-1
10e-03	-1
10.01e-03	0
11e-03	0
12e-03	-1
12.01e-03	0
13e-03	0

Figura A.10. Ejemplo de archivo de texto de especificación de fuente PWL

En la Tabla A.21 se representa el símbolo de la fuente dependiente no lineal. Los parámetros ajustables se encuentran en el menú: **Circuito/Propiedades de componentes**, en la hoja denominada **Valor** (o haciendo doble clic sobre el símbolo de la fuente), debiéndose indicar dónde se encuentra almacenado el fichero de texto de especificación de la fuente PWL.

Tabla A.21. Fuente lineal de voltaje controlado (PWL)

Símbolo	Parámetros ajustables	Simulación
		

En la Figura A.11 se representa la señal de salida de la fuente PWL definida en el ejemplo y obtenida en el osciloscopio del simulador *Electronics Workbench*.

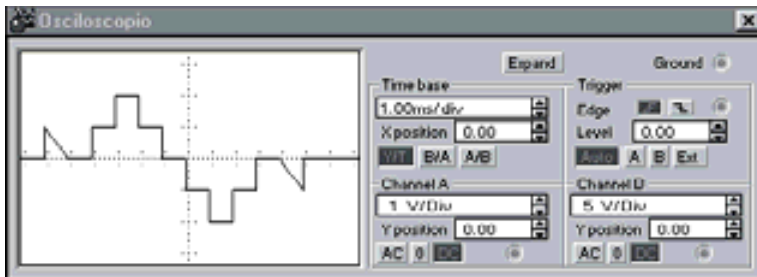


Figura A.11. Detalle de una onda senoidal

# MANUAL BREVE DEL LENGUAJE VHDL Y NOTACIÓN BNF

---

---

***Objetivos:***

- Conocer los entornos HDLs y BNF.
- Conocer las características metodológicas de diseño para HDL y BNF.
- Conocer los elementos sintácticos del lenguaje VHDL.
- Conocer la notación BNF del lenguaje VHDL.

***Contenido:*** Descripción del entorno y características de las metodologías de diseño más utilizadas actualmente y las aportaciones de los HDLs en el proceso de diseño; así como del lenguaje VHDL: sus orígenes, características, elementos sintácticos, unidades básicas de diseño, etc. Al final del apéndice, se utiliza la notación BNF para especificar las reglas sintácticas del lenguaje VHDL.

***Simulación:*** Este apéndice no tiene ejercicios de simulación por su carácter teórico, si bien se realizan descripciones utilizables en aplicaciones de simulación VHDL.

## B.1 METODOLOGÍAS DE DISEÑO Y APORTACIONES DE LOS HDLS AL PROCESO DE DISEÑO

Las **metodologías de diseño descendente**, *top-down* utilizan los **lenguajes de descripción de hardware** y las herramientas de simulación y síntesis, con el **objetivo** de concentrar el esfuerzo inicial en el aspecto funcional o arquitectural de los sistemas, para poder obtener un abanico de soluciones alternativas antes de establecer el diseño detallado y su realización física. El uso de los HDLs en el proceso de diseño es una característica del denominado **diseño de alto nivel**.

La **globalización del diseño electrónico** ha permitido el desarrollo en paralelo del diseño asistido por ordenador **CAD** (*Computer Aided Design*) y de la automatización del diseño electrónico **EDA** (*Electronics Design Automation*), posibilitando, en sistemas complejos, una **ingeniería concurrente** formada por equipos de trabajo que realizan, por separado y simultáneamente, aspectos de diseños *hardware* y *software* de un sistema.

En el desarrollo electrónico, tanto *hardware* como *software*, se utiliza **estaciones de trabajo** conectadas en red para tener acceso a todos los recursos distribuidos. Los ordenadores personales **PCs**, gracias a las nuevas posibilidades que ofrecen sus sistemas operativos (*Windows*, etc.), la potencia de los nuevos procesadores (Pentium, etc.), los sistemas multimedia y las comunicaciones (redes, Internet, correo electrónico, grupos de trabajo, etc.), y todo ello con un coste razonable, están integrados en esta red informática como puntos de acceso al entorno informático y para el proceso de diseño y almacenamiento local.

Aunque en principio los lenguajes más extendidos de descripción *hardware*, **Verilog** y **VHDL**, fueron creados con una semántica y sintaxis dirigida al modelado de simulación *hardware*, rápidamente se extendió su **uso en todas las fases del proceso de diseño** y muy en especial en las etapas de síntesis. Otro lenguaje de descripción *hardware*, menos utilizado es el **UDL/I** desarrollado por los japoneses.

Las principales **ventajas** que presentan los HDLs en el proceso de diseño son:

- Los lenguajes Verilog y especialmente el VHDL son **lenguajes de dominio público**, que no están sometidos a ninguna patente ni firma comercial. Están especificados y **mantenidos por el IEEE**, quien garantiza una continuidad, mejora y soporte documental.
- Los HDLs, al ser lenguajes de alto nivel, son de **fácil interpretación** para las personas como para los equipos informáticos, por ello se utilizan tanto en los procesos específicos de diseño (modelado, simulación, síntesis y verificación), como en los intercambios de modelos, documentación y mantenimiento de los diseños entre distintos equipos de trabajo.
- Permiten **descripciones con diferentes niveles de abstracción**, pudiéndose combinar módulos funcionales con módulos a nivel de puertas (descritos más detalladamente). Esto posibilita diseños independientes de otros módulos y

componentes, pudiéndose distribuir diferentes tareas entre **distintos equipos**, todo ello mediante un lenguaje común en simulación, síntesis y verificación.

- Proporcionan **independencia en la metodología utilizada, la tecnología y las herramientas de diseño**. Esta independencia permiten la **reutilización del código**. Una descripción VHDL pensada en principio para una determinada tecnología (TTL, CMOS, etc.) y para una determinada implementación (ASIC, FPGA, etc.) se puede con facilidad utilizar en otros diseños con tecnologías, implementaciones y herramientas CAD diferentes, o el mismo diseño puede ser modificado, adaptándolo o mejorándolo con nuevas tecnologías.
- La utilización de un lenguaje común en todo el proceso de diseño **facilita su gestión** y reduce los formatos, convertidores de formatos y herramientas de diseño.
- Los HDLs son los lenguajes estándar con una sintaxis y semántica definida para el modelado en simulación, lo cual permite la portabilidad entre las diferentes herramientas CAD. Esta **portabilidad no se cumple en la síntesis ni en la verificación**, por no estar especificada la semántica de estos procesos en el IEEE, existiendo herramientas CAD, que según la empresa que las ha desarrollado, presentan diferencias de interpretación en el proceso de síntesis.

Algunas de las **limitaciones** que presentan los HDLs son:

- Son lenguajes generales (en especial el VHDL) que han sido desarrollados por universidades y empresas mediante acuerdos que satisfacen diversos y variados aspectos y por lo tanto suelen ser **lenguajes complejos**.
- La **evolución** de los lenguajes es lenta. En VHDL se producen revisiones importantes cada cinco o más años en la comisión correspondiente del IEEE.
- La **carencia de semántica formal para la síntesis** complica la portabilidad de los diseños entre los diferentes entornos de síntesis y verificación.

## B.2 RESEÑA HISTÓRICA DE LOS HDLS

El diseño de sistemas digitales está fundamentado en la:

- **Tecnología** de realización de componentes.
- Metodología de **diseño** de circuitos.

En los **años sesenta** además de las **tecnologías** bipolares, aparece la Tecnología unipolar MOS (*Metal Oxide Semiconductor*), la cual produce una gran evolución e incremento de los procesos de fabricación de circuitos integrados. Esta época se caracteriza por **diseños manuales** en los que primaban el asegurar los niveles eléctricos de interconexión entre componentes basados en transistores apareciendo los



primeros simuladores con modelos altamente dependientes de la tecnología utilizada, como *PSpice*.

La evolución de los procesos tecnológicos ha experimentado un gran crecimiento en complejidad, aumentando el nivel de integración, los cuales son más difíciles de depurar y mantener. Los circuitos MSI (*Medium Scale Integration*) y LSI (*Large Scale Integration*) se diseñaron mediante **prototipos** de puertas lógicas previamente probados. Este rápido crecimiento experimentado por la tecnología junto con los métodos de diseño altamente manuales (obsoletos) en los **años ochenta**, manifestó un gran **desfase entre la tecnología y el diseño**.

El aumento de la complejidad de los circuitos integrados que se pueden fabricar, implica unos grandes **riesgos y costes de diseño** imposibles de asumir por la mayoría de las empresas.

La metodología actual de diseño de circuitos, es la especificación y descripción del sistema. La captura de esquemas ha quedado obsoleta si el sistema es complejo (con miles de componentes). Esto justifica la necesidad de **utilizar un lenguaje para la especificación y descripción del hardware**. Partiendo de esta idea se formaron varios grupos de investigación con el objeto de crear y desarrollar lenguajes de descripción *hardware*. Entre éstos destacan el lenguaje IDL desarrollado por IBM, TI-HDL de *Texas Instruments*, ZEUS de *General Electric*, etc. Las universidades también desarrollaron lenguajes buscando una solución a los problemas del diseño de sistemas complejos.

Los anteriores lenguajes no alcanzaron la difusión y consolidación necesaria debido a que los desarrollados en la industria permanecieron encerrados en las empresas por ser de su propiedad y no estar disponibles para su estandarización, y los desarrollados en las universidades no se difundieron al no disponer de soporte ni mantenimiento adecuado.

En 1981 el Departamento de Defensa de los Estados Unidos desarrolla un proyecto llamado VHSIC (*Very High Speed Integrated Circuit*) con el objeto de rentabilizar las inversiones en *hardware* haciendo más sencillo su mantenimiento y resolver el problema de modificar el *hardware* diseñado en un proyecto para utilizarlo en otro.

### B.3 LENGUAJE DE DESCRIPCIÓN FORMAL VHDL

En 1983, IBM, *Texas Instruments* e *Intermetrics* comienzan a trabajar conjuntamente en el desarrollo de un lenguaje de diseño que permita la estandarización, facilitando con ello, el mantenimiento de los diseños y la depuración de los algoritmos. Para ello el IEEE propuso su estándar en 1984.

Posteriormente se realizaron varias versiones en las que colaboraron la industria y las universidades. El IEEE publicó en diciembre de 1987 el **estándar IEEE std 1076-1987** que constituyó el punto de partida de lo que después de cinco años sería denominado VHDL.

El conjunto de **propiedades del lenguaje VHDL** son:

- la **independencia en la metodología, tecnología, herramientas de diseño y fabricantes**,
- su capacidad descriptiva en **múltiples dominios y niveles de abstracción**,
- su **versatilidad** para la descripción de sistemas complejos, y
- su posibilidad de **reutilización**.

Todas estas propiedades han hecho que VHDL se convierta en el lenguaje de descripción *hardware* más utilizado.

La primera versión del año 1987 (denominada VHDL'87) presenta carencias en la síntesis de circuitos principalmente debidas a la evolución de las herramientas de diseño. Recogidas las experiencias, problemas y gracias a la rápida difusión del lenguaje en la comunidad científica se crea una segunda versión en el año 1993 cuya referencia es **IEEE Std 1076-1993**, y que es conocida como VHDL'93.

Esta versión guarda la compatibilidad con la anterior, siendo el VHDL'87 un subconjunto del VHDL'93. Las nuevas posibilidades que ofrece esta última versión están relacionadas con las declaraciones y uso de los ficheros.

La versión VHDL'93 es más fácil de utilizar, por sus menores restricciones, aunque no todas las herramientas de simulación y síntesis son capaces de entender esta versión.

## B.4 ELEMENTOS SINTÁCTICOS DEL LENGUAJE VHDL

El lenguaje VHDL, al igual que cualquier otro lenguaje, tiene su sintaxis, sus tipos de datos y sus estructuras. Debido a que este tipo de lenguaje permite una descripción *hardware* le diferencia de otros lenguajes en la necesidad de ejecutar instrucciones de forma **concurrente**.

En los siguientes subapartados se describen los principales elementos sintácticos.

### B.4.1 Objetos

Un objeto en VHDL es un elemento que tiene asignado un valor de un tipo determinado y un conjunto de operaciones que se puede utilizar.

Es importante señalar que en general, no se pueden realizar operaciones entre dos objetos que sean de distinto tipo, a no ser que previamente se efectúe una conversión de tipos.

## B.4.2 Identificadores

Los **identificadores** son conjuntos de caracteres que cumplen unas determinadas normas propias del lenguaje y sirven para dar nombre a los elementos en VHDL, como: señales variables, etiquetas, etc.

Las reglas a tener en cuenta al elegir un identificador son:

- Nombres alfanuméricos, pudiendo incluir el símbolo de subrayado `_`.
- Deben comenzar con un carácter alfabético, no pudiendo terminar con un carácter de subrayado, ni tener dos o más caracteres subrayados seguidos.
- VHDL no es sensible a las mayúsculas y minúsculas, siendo iguales los identificadores `Salida` y `SALIDA`.
- No deben contener símbolos especiales, ni coincidir con las palabras claves o reservadas de VHDL.

## B.4.3 Palabras reservadas

Las **palabras reservadas** son un conjunto de identificadores que tienen significado específico en VHDL al ser empleadas dentro del lenguaje en la descripción del diseño. Por esta razón no pueden ser empleadas en los identificadores definidos por el usuario.

Las palabras reservadas en VHDL'87 son las siguientes:

ABS	ELSE	NAND	REPORT
ACCESS	ELSIF	NEW	RETURN
AFTER	END	NEXT	SELECT
ALIAS	ENTITY	NOR	SEVERITY
ALL	EXIT	NOT	SIGNAL
AND	FILE	NULL	SUBTYPE
ARCHITECTURE	FOR	OF	THEN
ARRAY	FUNCTION	ON	TO
ASSER	GENERATE	OPEN	TRANSPORT
ATTRIBUTE	GENERIC	OR	TYPE
BEGIN	GUARDED	OTHERS	UNITS
BLOCK	IF	OUT	UNTIL
BODY	IN	PACKAGE	USE

BUFFER	INOUT	PORT	VARIABLE
BUS	IS	PROCEDURE	WAIT
CASE	LABEL	PROCESS	WHEN
COMPONENT	LIBRARY	RANGE	WHILE
CONFIGURATION	LINKAGE	RECORD	WITH
CONSTANT	LOOP	REGISTER	XOR
DISCONNECT	MAP	REM	
DOWNTO	MOD		

En VHDL'93 se incluyen además las siguientes palabras reservadas:

GROUP	POSTPONED	ROR	SRA
IMPURE	PURE	SHARED	SRL
INERTIAL	REJECT	SLA	UNAFFECTED
LITERAL	ROL	SLL	XNOR

## B.4.4 Símbolos especiales

El lenguaje VHDL utiliza símbolos para realizar funciones específicas. Pueden ser de un solo carácter:

+ - / \* ( ) . , : ; & ` " < > = | #

y de dos caracteres:

\*\* => := /= >= <= <> --

Un símbolo muy empleado por el programador es el ; debido a que con él se finalizan cada una de las líneas de código, señalando la terminación de cada sentencia del programa.

## B.4.5 Comentarios

El símbolo -- se emplea para que el usuario realice comentarios que aclaren o hagan más comprensible el código del programa. El programa compilador ignora lo escrito después de dicho símbolo.

## B.4.6 Números

Por defecto se consideran en base decimal a las representaciones numéricas. Es posible definir números en otras bases utilizando el símbolo del sostenido #. Ejemplo: 2#11100# y 16#1C# representan al número decimal 28.

También se admite la notación científica en coma flotante.

## B.4.7 Caracteres

Son cualquier letra o carácter entre comillas simples.

Ejemplo: '5', 'x', '\$'.

## B.4.8 Cadenas

Son conjuntos de caracteres entre comillas dobles.

Ejemplo: "Ejemplo de cadena".

## B.4.9 Tipos de datos

El **tipo de datos** es un elemento básico que delimita los valores que puede tener un objeto en VHDL y las operaciones que se pueden efectuar con él.

La sintaxis es rigurosa puesto que todos los objetos en VHDL deben pertenecer a un determinado tipo de datos.

A diferencia de otros lenguajes, en VHDL no existen tipos de datos propios como por ejemplo los enteros (*integer*) o reales (*real*), aunque hay medios para especificar cualquier tipo de datos.

De hecho existen unos **tipos de datos predefinidos**, presentes en una biblioteca que generalmente está presente en todas las herramientas VHDL: compiladores, simuladores, etc. y que al no tener que definirles el usuario parece que pertenecen al lenguaje, pero no es cierto.

A continuación, se muestran los tipos de datos que se encuentran predefinidos en el lenguaje VHDL:

### **boolean**

Puede tomar dos valores: verdadero (*true*) o falso (*false*). Por ejemplo la salida de un comparador puede ser verdadero si los números comparados son iguales y falso si son distintos.



Cada tipo es diferente e incompatible con los demás, aunque tenga una declaración idéntica, por lo que no se puede asignar a un objeto de un tipo otro de tipo distinto, a menos que se le aplique una función de conversión.

Los **tipos se clasifican** en:

- **escalares:** enumerados, enteros, reales, y físicos; y en,
- **compuestos:** vectores o matrices y registros.

### B.4.9.1 TIPOS ENUMERADOS

Está formado por una lista que define el conjunto de valores que puede tomar este tipo de datos. El primer identificador es el nombre del tipo y sirve para referenciarlo, y entre paréntesis y separados por comas se enumeran la lista de todos los valores legales del tipo. Ejemplo:

```
type orientación is (Norte, Sur, Este, Oeste);
```

En caso de no especificar ningún valor inicial, por defecto el objeto toma el valor situado más a la izquierda de los especificados en la declaración del tipo. Es decir, en el objeto de tipo orientación toma el valor Norte por defecto.

### B.4.9.2 TIPOS ENTEROS/REALES

Como ya se ha indicado anteriormente son tipos de datos predefinidos. El tipo **integer** tiene un rango comprendido entre -2147483647 y 2147483647. El tipo **real** su rango está comprendido entre -1.0e38 y 1.0e38.

Para definirlos hay que especificar el rango de valores que puede tener el objeto, como por ejemplo:

```
type meses is range 12 downto 0;
```

```
type byte is range 0 to 255;
```

Si no se ha especificado ningún valor inicial, por defecto el objeto, toma el valor especificado más a la izquierda en la declaración del tipo. En los ejemplos anteriores el objeto **meses** se inicializa por defecto con el valor 12 y el objeto **byte** con el valor 0.

### B.4.9.3 TIPOS FÍSICOS

Tipos de datos que sirven para representar magnitudes del mundo real como la presión, energía, fuerza, tiempo, etc., por lo que están formados por un número y una magnitud física.

Se pueden asignar unidades auxiliares a la definida, como por ejemplo:

```
type condensador is rango 1 to 1e12;
units
pF;
nF=1000 pF;
uF=1000 nF;
mF=1000 uF;
F=1000 mF;
end units;
```

Si no se ha especificado ningún valor inicial, por defecto el objeto, toma el valor inicial especificado más a la izquierda en la declaración del tipo. En el ejemplo el objeto **condensador** se inicializa por defecto con el valor 1.

#### B.4.9.4 VECTORES Y MATRICES

Es un tipo de datos compuesto, formado por un conjunto de **objetos del mismo tipo** ordenados mediante uno o más índices, los cuales además sirven para señalar la posición del objeto dentro del vector.

Los vectores con sólo un índice se les denomina **vectores unidimensionales** o simplemente **vectores** y con varios índices **vectores multidimensionales** o **matrices**.

La sintaxis en VHDL es:

```
type identificador is array (rango{, ...}) of tipo objetos;
```

como por ejemplo:

```
type vector_unidimensional is array (0 to 2) of integer;
```

```
type matriz_memoria is array (0 to 7, 0 to 256) of bit;
```

#### B.4.9.5 REGISTROS

Es un tipo de datos compuesto, que a diferencia de los vectores puede estar formado por un conjunto de **objetos de distinto tipo** (denominados campos), los cuales en vez de referenciarse mediante índices, se hace mediante identificador.

La sintaxis en VHDL es:

```
type identificador is record
  identificador {, ...}: tipo;
```



```
{...}
```

```
end record;
```

como por ejemplo:

```
type ficha is record
```

```
  Nombre_apellidos: string;
```

```
  dni: integer;
```

```
  sexo: bit;
```

```
end record;
```

## B.4.10 Operadores y expresiones

En la Tabla B.1 se muestran los distintos operadores predefinidos más empleados en lenguaje VHDL y clasificados según el tipo de datos utilizado.

*Tabla B.1. Operadores predefinidos en VHDL*

Clasificación	Operadores	Operandos / Resultado	
Operadores lógicos	AND, OR, NOT, NAND, NOR, XOR	Operandos:	bit, bit_vector, boolean
		Resultado:	bit, bit_vector, boolean
Operadores relacionales	= / < <= > >=	Operandos:	cualquier tipo
		Resultado:	boolean
Operadores aritméticos	+ - * / ** MOD, REM, ABS	Operandos:	integer, real, signal
		Resultado:	integer, real, signal
Operadores de desplazamiento (sólo en VHDL'93)	SLL, SRL, SLA, SRA, SRA, ROL, ROR	Operandos:	bit_vector
		Resultado:	bit_vector
Operador concatenación	&	Operandos:	vector o matiz
		Resultado:	vector o matriz

Los **operadores lógicos** pueden ser empleados con los tipos predefinidos `bit`, `bit_vector` y `boolean`, dando como resultado un valor del mismo tipo que los operadores.

Los **operadores relacionales** generan un resultado de tipo booleano sin importar el tipo de los operandos.

Los **operandos aritméticos** deben ser del mismo tipo (`integer`, `real`, `signal`) y la operación devuelve un resultado del mismo tipo de datos.

El **operador concatenación** `&` es empleado para unir o concatenar vectores o matrices, obteniéndose una matriz resultante cuya dimensión es la suma de las dimensiones de las matrices de los operandos. Los elementos a concatenar deben ser del mismo tipo y el resultado que nos devuelve esta operación es un vector con los elementos de los operandos concatenados.

Este operador es empleado en la descripción de *buses* y registros.

### B.4.11 Precedencia de operadores

En la Tabla B.2 se relaciona la precedencia de los operadores ordenados de mayor precedencia (arriba) a menor (abajo). Los operadores situados en la misma fila tienen la misma precedencia. Una expresión se evaluará de izquierda a derecha según la precedencia de los operadores. El uso de paréntesis permite evaluar primero operadores con menor precedencia.

Tabla B.2. Operadores predefinidos en VHDL

Precedencias		Operadores				
Mayor	**	ABS	NOT			
	*	/	MOD	REM		
	Signo +	signo -				
	+	-	&			
	=	/=	<	>	<=	>=
Menor	AND	OR	NAND	NOR	XOR	XNOR

## B.4.12 Atributos

Los **atributos** proporcionan información adicional sobre cualquier elemento de un modelo VHDL, tales como: objetos, tipos de datos, etc.

Los atributos se representan mediante el símbolo de comilla simple ‘ en los objetos del lenguaje.

La sintaxis en VHDL es:

```
identificador_elemento'identificador_atributo
```

### B.4.12.1 ATRIBUTOS PREDEFINIDOS

Los atributos predefinidos en VHDL se indican a continuación:

Suponiendo que **a** es un elemento de tipo escalar (enumerado, entero, flotante, o físico), se le puede aplicar los atributos predefinidos de la Tabla B.3.

*Tabla B.3. Atributos predefinidos para un elemento a de tipo enumerado, entero, flotante, o físico*

Atributo	Significado
a`left	Límite izquierdo del tipo a
a`right	Límite derecho del tipo a
a`low	Límite inferior del tipo a
a`high	Límite superior del tipo a

Suponiendo que el tipo del atributo **a** es el elemento anteriormente indicado, **i** un miembro de ese tipo, y **N** un entero, se le puede aplicar los atributos predefinidos de la Tabla B.4.

Suponiendo **m** es de tipo compuesto (vector o matriz) y **N** un entero entre 1 y el número de dimensiones de la matriz, se pueden aplicar los atributos predefinidos de la Tabla B.5.

Suponiendo que **s** es una señal, se pueden aplicar los atributos predefinidos de la Tabla B.6.

*Tabla B.4. Atributos predefinidos para un elemento  $a$  del tipo anterior,  $i$  un miembro de ese tipo, y  $N$  un entero*

<b>Atributo</b>	<b>Significado</b>
$a'pos(i)$	Posición de $i$ dentro del tipo $a$
$a'val(N)$	Elemento $N$ del tipo $a$
$a'leftof(i)$	Elemento que está a la izquierda de $i$ en $a$
$a'rightof(i)$	Elemento que está a la derecha de $i$ en $a$
$a'pred(i)$	Elemento que está delante de $i$ en $a$
$a'succ(i)$	Elemento que está detrás de $i$ en $a$

*Tabla B.5. Atributos predefinidos para un elemento  $m$  de tipo matriz*

<b>Atributo</b>	<b>Significado</b>
$m'left(N)$	Límite izquierdo del rango de dimensión $N$ de $m$
$m'right(N)$	Límite derecho del rango de dimensión $N$ de $m$
$m'lowf(N)$	Límite inferior del rango de dimensión $N$ de $m$
$m'high(N)$	Límite superior del rango de dimensión $N$ de $m$
$m'range(N)$	Rango de índice de dimensión $N$ de $m$
$m'length(N)$	Longitud del índice de dimensión $N$ de $m$

*Tabla B.6. Atributos predefinidos para una señal  $s$*

<b>Atributo</b>	<b>Significado</b>
$s'event$	Devuelve true si se produce un cambio o evento en la señal $s$ . Sirve para detectar flancos de $s$
$s'stable(tiempo)$	Devuelve true si la señal permaneció estable durante el último período igual a tiempo

### B.4.12.2 ATRIBUTOS DEFINIDOS POR EL USUARIO

Mediante atributos definidos por el usuario se puede agregar información adicional a los objetos definidos en VHDL, como por ejemplo transferir información a las herramientas de diseño y síntesis utilizadas en VHDL, como parámetros de control en la síntesis o tipos de encapsulados en herramientas de diseño de PCBs.

Los atributos definidos por el usuario devuelven siempre una constante.

La **definición de un atributo** se realiza con la palabra reservada **ATTRIBUTE** y consta de:

- **declaración**, indicando el tipo de elemento que devuelve,  
-- Declaración  
**ATTRIBUTE** nombre: tipo;
- **especificación**, indicando el valor que devuelve en cada caso,  
-- Especificación  
**ATTRIBUTE** nombre **OF** id\_elemento: clase\_elemento **IS** valor;

### B.4.13 Constantes, variables y señales

Al igual que en los **Lenguajes de Descripción Software (SDL)**, los **Lenguajes de Descripción Hardware (HDL)**, disponen de objetos denominados **variables** que contienen un valor inicial que puede ser modificado mediante una asignación secuencial. También disponen de objetos denominados **constantes** que tienen valores prefijados y fijos a lo largo de la ejecución del programa. Sin embargo, en los lenguajes de descripción *hardware*, como el VHDL, es necesario la utilización de un nuevo tipo de objeto, denominado **señales**, para poder emular las asignaciones concurrentes propias de los circuitos eléctricos reales.

#### B.4.13.1 CONSTANTES

Una constante es un elemento que se inicializa con un valor y que no puede ser modificado. Ejemplo:

```
CONSTANT pi: real := 3.141592;
```

```
CONSTANT inicio_contador: natural;
```

En la última sentencia no se ha dado valor a la constante inicio\_contador, esto es posible siempre que se declare en otra parte del programa.

### B.4.13.2 VARIABLES

A las variables se las pueden asignar un valor inicial y modificarlas en cualquier instante. Ejemplo:

**VARIABLE** minuendo: bit\_vector(15 **DOWNTO** 0);

**VARIABLE** acumulador: natural := 0;

La **asignación a una variable** sólo se puede definir en un proceso o un subprograma (ejecuciones secuenciales). Las variables son locales y, por tanto, no mantiene su valor dentro de un subprograma después de cada llamada, es decir una variable se reinicializa cada vez que se llama al subprograma. Para hacer uso de una variable fuera de un proceso, se debe asignar su valor a una señal y operar fuera del proceso con la señal.

La asignación de una variable se realiza con el operador **:=**, como por ejemplo:

b:=12;           -- La variable b toma el valor 12

a:=b;           -- La variable a toma el valor de la variable b

b:=a;           -- La variable b toma de nuevo el valor de la variable a

Para poder intercambiar el valor de las señales a y b, se debe usar otra variable (variable temporal):

temporal:=a;   -- La variable temporal toma el valor de la variable a

a:=b;           -- La variable a toma el valor de la variable b

b:=temporal;   -- La variable b toma de nuevo el valor de la variable temporal

El asignar un valor a una variable no tiene un retardo asociado, es decir, la variable toma el nuevo valor justo en el momento de la asignación. Ésta es una de las diferencias que tienen las variables respecto a las señales, como se indica a continuación.

### B.4.13.3 SEÑALES

Una señal se diferencia de una variable en que el valor que guarda no es visible en el instante. Es decir, una **señal tiene dos partes**:

- **donde se escribe** o se guarda el valor y
- **donde se lee**, que puede que no coincida con lo que se acaba de escribir.

Internamente hay una **conexión** entre estas dos partes, siendo posible su desconexión de manera que al leer la señal no se tenga acceso a lo que se escribió mediante la palabra reservada **NULL**.

Las señales pueden ser de **tipo normal, register y bus**. En caso de no especificarlo será por defecto de tipo **normal**; para las otras dos opciones se emplean las palabras reservadas **register** o **bus**. Estos tres tipos de señal se diferencian en su **comportamiento al desconectarse**:

- la señal de tipo **normal** no es posible su desconexión;
- la señal de tipo **bus** adquiere un valor por defecto cuando todas las fuentes de la señal se encuentran desconectadas,
- la señal de tipo **register** mantiene el último valor que se escribió cuando todas las fuentes de la señal se encuentran desconectadas.

A las señales se las puede inicializar. Por ejemplo:

```
SIGNAL reset: bit := '1';
```

```
SIGNAL Bus_datos: bit_vector(7 DOWNTO 0) := B"00001100";
```

Para **asignar un valor a una señal**, ésta debe haber sido declarada anteriormente en una entidad (en las declaraciones de puertos), o haber sido definida en un proceso (process).

Para realizar una asignación a una señal se usa el operador **<=**. Su sintaxis es:

```
Señal <= valor;
```

La asignación a una señal dentro de un proceso es secuencial, es decir, no se modifica hasta que se haya evaluado dicho proceso. En el caso de no pertenecer a un proceso, el cambio es concurrente, la asignación siempre se actualiza instantáneamente. En el siguiente ejemplo se aclara esta particularidad:

```
PROCESS
```

```
BEGIN
```

```
  a <= b;
```

```
  b <= a;
```

```
  wait on a,b;
```

```
END PROCESS;
```

En el anterior proceso las dos señales intercambian sus valores, pues cuando se ejecuta la segunda asignación (**b <= a;**), el valor de la señal **a** no ha cambiado aún en la primera asignación (**a <= b;**), puesto que ninguna señal cambia de valor hasta que no se evalúe todo el proceso, y es en ese momento cuando **a** y **b** toman el valor asignado.

Las señales representan el mecanismo que permite ejecutar en paralelo las instrucciones concurrentes, debido a que en VHDL se **ejecutan las instrucciones concurrentes** cuando se produce un **evento o cambio en las señales**.

Las señales son conexiones entre componentes pueden ser de entradas o salidas, aunque ambas son declaradas como señales se tiene que considerar las siguientes diferencias:

- a las entradas no se las pueden dar valores en su descripción y,
- a las salidas no se las pueden leer, es decir, no pueden ser argumento de una asignación.

#### B.4.13.4 RESUMEN DE DIFERENCIAS ENTRE VARIABLES Y SEÑALES

Las **variables** sólo tienen sentido en entornos de ejecución serie como en procesos y subprogramas. Son elementos abstractos que no tienen por qué tener un significado físico. El valor asignado a una variable se hace visible de forma inmediata.

Las **señales** se pueden declarar, como más adelante se indicará, en bloques concurrentes, arquitecturas y paquetes. El valor asignado a una señal no se hace visible hasta el siguiente paso de simulación. Poseen un significado puramente físico al representar las conexiones de un circuito.

### B.4.14 Unidades básicas de diseño en VHDL

En este apartado se describen las unidades básicas de diseño utilizando la **notación BNF (*Backus Naur Form*)**, por ser una notación muy extendida en la descripción de lenguajes. En el último apartado de este apéndice se puede encontrar una explicación de la notación BNF y la descripción completa de la sintaxis del VHDL'93.

#### B.4.14.1 DECLARACIÓN DE ENTIDADES

Todas las descripciones VHDL comienzan con la definición del símbolo o entidad ENTITY del circuito. Como si de una caja negra se tratara, se definen las entradas, salidas y sus dimensiones tales como, su tamaño (0 a  $n$  bits), modo (IN, OUT, BUFFER, etc.) y tipo (integer, bit, etc.).

La declaración de entidades es análoga al símbolo esquemático que se quiere implementar, el cual describe las conexiones de un componente al resto del circuito.

La sintaxis de entidades en VHDL tiene la forma:

**ENTITY** nombre **IS**

[**GENERIC** (lista de parámetros);]

[**PORT** (lista de puertos);]



```

[declaraciones]
[BEGIN
    sentencias]
END [ENTITY] [nombre];

```

La instrucción **GENERIC** es opcional y sirve para definir las propiedades y pasar parámetros o constantes a la entidad.

Mediante la palabra opcional **PORT**, se definen el puerto de entradas y salidas del módulo.

La **declaración de un puerto** consiste en indicar el nombre del puerto, y seguido por dos puntos se indica el modo de conexión y el tipo de datos del puerto.

Nombre\_puerto: modo tipo;      Declaración genérica de un puerto

Ejemplos de declaraciones de puerto son:

Puerto_uno: <b>IN</b> bit;	Puerto de un bit de entrada de nombre Puerto_uno.
Puerto_dos: bit_vector(3 <b>DOWNTO</b> 0);	Puerto de cuatro bits de entrada, siendo el MSB el puerto_dos(3) y el LSB el puerto_dos(0)
Puerto_tres: <b>OUT</b> bit_vector(0 <b>TO</b> 7);	Puerto de ocho bits de salida, siendo el MSB el puerto_tres(7) y el LSB el puerto_tres(0)
Puerto_cuatro: <b>BUFFER</b> bit;	<i>Buffer</i> de un bit.
Puerto_cinco: <b>INOUT</b> std_logic;	Puerto de un bit de entrada y salida del tipo estándar lógico.

El **modo** describe la dirección en la que la información se transmite por el puerto, pudiendo ser **IN**, **OUT**, **BUFFER**, **INOUT** y **LINKAGE**. Si no se especifica se toma por defecto el modo **IN**.

## **IN**

La información del puerto entra a la entidad. Se usa como entradas de reloj, y señales del bus de control (reset, escritura, lectura, selección de chip, etc.) y datos o buses de entrada unidireccionales como el de direcciones.

## OUT

La información de la entidad sale por el puerto. Este modo no permite la realimentación puesto que el compilador no permite su lectura, aunque en principio pueda parecer un inconveniente tiene la ventaja de consumir menos recursos en la síntesis de dispositivos lógicos programables.

## BUFFER

Es un modo muy similar al OUT pero con la diferencia de que al poderse leer permite la realimentación. Es **unidireccional** y sólo puede ser conectado directamente a una señal interna o a otro puerto en modo *buffer* de otra entidad.

Una aplicación típica de este modo es la de salida de un contador, en la que es necesario conocer la salida en el estado presente para determinar la salida en el estado futuro.

## INOUT

Es un modo para señales **bidireccionales**, es decir, la información del puerto entra o sale de la entidad. Permite la realimentación interna y puede reemplazar a cualquiera de los anteriores modos, pero con la desventaja de dificultar la comprensión del código y consumir los recursos disponibles al realizar la síntesis del diseño de los dispositivos programables.

## LINKAGE

Es un modo menos usado, similar a INOUT. Se utiliza para enlazar con otros módulos que no tienen por qué estar escritos en VHDL, es decir, como interfaz con otros modelos de otras herramientas.

El apartado opcional de **declaraciones** sirve para expresar constantes.

En el bloque opcional **BEGIN** es donde se incluyen sentencias de test, monitorización y detección de errores en la simulación.

## Ejemplo:

Declaración de entidades correspondientes a un selector de datos.

```
ENTITY selector_datos IS
```

```
    PORT (enable: IN bit;
```

```
           seleccion: IN bit;
```

```
           dato_0: IN bit_vector(3 DOWNTO 0);
```

```
           dato_1: IN bit_vector(3 DOWNTO 0);
```

```
           salida: OUT bit_vector(3 DOWNTO 0) -- No lleva ;
```

```
END selector_datos;
```

La entidad es análoga al símbolo esquemático del componente, correspondiendo la declaración de la entidad anterior con el esquema que se representa a continuación en la Figura B.1.

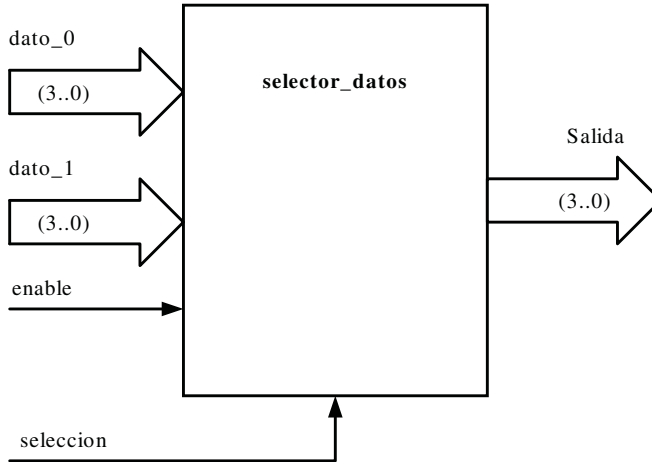


Figura B.1. Esquema correspondiente a la entidad de un selector de datos

### Ejemplo:

Declaración de entidades correspondientes a un microprocesador.

**ENTITY** microprocesador **IS**

**GENERIC**(frec\_max: frecuencia := 100 MHz);

**PORT** (Relej,Rst: **IN** std\_logic;

Bus\_direcciones: **OUT** std\_logic\_vector(15 **DOWNTO** 0);

Bus\_datos: **INOUT** std\_logic\_vector(7 **DOWNTO** 0);

r\_w: **OUT** std\_logic);

**END** microprocesador;

El símbolo del microprocesador declarado en la entidad anterior se representa en la Figura B.2.

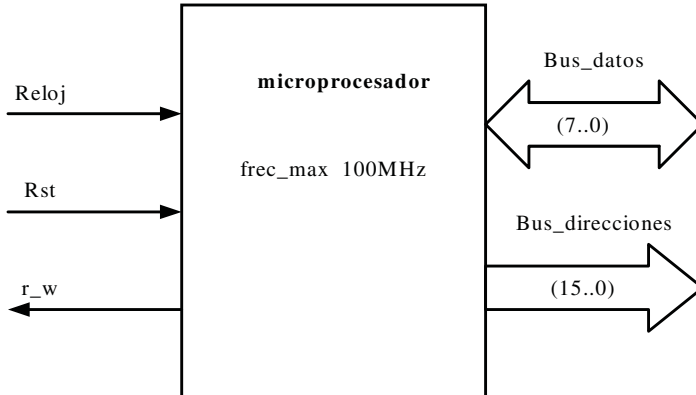


Figura B.2. Esquema correspondiente a la entidad de un procesador

#### B.4.14.2 DECLARACIÓN DE ARQUITECTURAS

En la arquitectura se describe el funcionamiento del módulo definido en la entidad, es decir, se especifica el procesado que se realiza con la información de las señales de entrada para obtener las señales correspondientes a los puertos de salida (especificados en la entidad).

Considerando a la entidad como una caja negra que lo único importante son sus entradas y salidas, la arquitectura es el conjunto de detalles interiores de esta caja negra.

A cada arquitectura le corresponde una entidad concreta, es decir, carece de sentido hacer declaraciones de arquitectura sin haber especificado la entidad. Sin embargo, una misma entidad puede tener diferentes arquitecturas, y es en el momento de la simulación o la síntesis cuando se puede elegir la arquitectura que se va a simular o sintetizar.

Su sintaxis VHDL es:

```

ARCHITECTURE nombre OF nombre_entidad IS
    [declaraciones]
BEGIN
    [sentencias concurrentes]
END [ARCHITECTURE] [nombre];
```

En la parte [declaraciones] de la arquitectura se definen: los subprogramas (funciones, procedimientos, etc.), componentes, tipos de datos, constantes y además es uno de los pocos apartados donde se pueden declarar las señales.

En el bloque **BEGIN ... END** están presentes las instrucciones que definen el comportamiento, estructura y funcionalidad del circuito. Estas **instrucciones** pueden ser de dos tipos:

- **concurrentes** y,
- **referencia de componentes**, que son **descripciones estructurales** que definen componentes y las conexiones entre ellos.

Una de las instrucciones concurrentes especiales es el bloque `.` cuyas instrucciones internas se ejecutan secuencialmente. Esto último permite **descripciones algorítmicas** con instrucciones muy parecidas a las utilizadas en los lenguajes de descripción software (SDL).

### Ejemplo:

Se indica a continuación la arquitectura de un selector de datos, siendo ésta una de las múltiples formas de describirle en VHDL.

-- La declaración de entidad se realizó en el anterior ejemplo

**ARCHITECTURE** arq\_selector\_datos **OF** selector\_datos **IS**

-- Este programa no necesita señales

**BEGIN**

**PROCESS** (enable, dato\_0, dato\_1)

**IF** enable = '0' **THEN** salida <= "1111";

**ELSIF** enable = '1' **THEN**

**IF** selección = '0' **THEN** salida <= dato\_0;

**ELSIF** selección = '1' **THEN** salida <= dato\_1;

**END IF;**

**END IF;**

**END PROCESS;**

**END** arq\_selector\_datos;

## B.4.15 Paquetes en VHDL

Los paquetes permiten agrupar un conjunto de declaraciones para que puedan ser usadas en el diseño de diferentes circuitos sin tener que ser repetidas en la declaración de cada uno.

La estructura básica de declaración de un paquete tiene dos partes:

- **Declaración del paquete**

Representa una visión externa del componente.

Su sintaxis VHDL es:

-- Declaración de paquete

**PACKAGE** nombre **IS**

declaraciones

**END [PACKAGE] [nombre];**

- **Cuerpo del paquete**

Especifica los tipos, subprogramas (funciones y procedimientos), etc., de forma similar a la arquitectura.

Su sintaxis VHDL es:

-- Declaración del cuerpo del paquete

**PACKAGE BODY** nombre **IS**

declaraciones

subprogramas, etc.

**END [PACKAGE BODY] [nombre];**

Una vez declarados los subprogramas dentro del paquete, éstos se puedan utilizar únicamente haciéndoles una llamada con la lista de parámetros necesarios.

Para **llamar a un paquete** ya creado, se debe incluir la palabra reservada **USE** en la parte declarativa de la arquitectura y especificar: el nombre de la biblioteca donde se encuentra el paquete, el nombre del paquete y el nombre del componente a habilitar.

Cuando no se conozca el nombre del componente a usar del paquete o no se tenga claro cuál de ellos se necesitarán en la descripción se pueden **habilitar todos los componentes** mediante la palabra reservada **ALL**.

La sintaxis VHDL es:

**USE** nombre\_biblioteca.nombre\_paquete.nombre\_componente;

**USE** nombre\_biblioteca.nombre\_paquete.**ALL**;

## B.4.16 Bibliotecas en VHDL

Las bibliotecas pueden almacenar diseños para posteriores usos. Esta característica de las bibliotecas permite la compartición y la **reutilización** del código en diferentes diseños.

Las estructuras estudiadas, entidad y arquitectura, se las denomina **unidades**. Estas unidades se incluyen en varios archivos que se compilan antes de la simulación o síntesis llamados **ficheros de diseño o de trabajo**, cuyo conjunto forma una única biblioteca, denominada **biblioteca de diseño**, con las descripciones de todos los elementos del circuito, que tiene por nombre **WORK**.

La síntesis o simulación del circuito se realiza sobre la biblioteca **WORK**.

Para incorporar en la biblioteca del diseño elementos de otras bibliotecas se utiliza la palabra reservada **LIBRARY** seguido de la lista de bibliotecas que se desea que sean accesibles.

**LIBRARY** nombre\_biblioteca;

Mediante la palabra reservada **USE** se pueden seleccionar los elementos internos de las bibliotecas: paquetes y componentes que se desea que sean accesibles.

**USE** nom\_biblioteca.nom\_paquete.nom\_elemento;

Ejemplos:

<b>LIBRARY</b> digital;	-- Hace visible la biblioteca digital
<b>USE</b> digital.puertas.nor2;	-- Hace visible la puerta nor2 del paquete puertas
<b>USE</b> digital.combinacionales. <b>ALL</b> ;	-- Hace visible todos los elementos del paquete combinacionales

En VHDL hay dos bibliotecas que no necesitan ser incorporadas puesto que son implícitamente accesibles, una es la biblioteca **WORK** que contiene las unidades de diseño resultado de la compilación y la otra es la **biblioteca STD** con dos paquetes, el **standard** y el **textio**. El paquete **standard** contiene las definiciones de tipos y constantes y el paquete **textio** contiene tipos y funciones para el acceso a ficheros de texto.

Otra biblioteca que se ha convertido en un estándar es IEEE con su paquete **std\_logic\_1164** que contiene elementos, tipos y funciones para trabajar con nueve

niveles lógicos, como: `std_ulogic` y `std_logic`, este último tiene asociada una función de resolución.

## B.4.17 Descripción de la estructura y comportamiento de circuitos en VHDL

Para **describir un circuito** normalmente se utilizan dos formas:

- **Descripción estructural**, definiendo los componentes y su interconexión. De esta forma se tiene especificado el circuito y su funcionamiento. Es la manera tradicional de descripción de circuitos, siendo las herramientas utilizadas las de **captura de esquemas** y las de descripción *netlist*.
- **Descripción comportamental**, indicando lo que hace el circuito y cómo funciona. De esta forma se describe su comportamiento, siendo muy interesante para el diseñador puesto que lo que realmente le interesa es el funcionamiento del circuito más que sus componentes o la tecnología empleada. Por otro lado, al ser en muchos casos esta descripción independiente de la tecnología a emplear, se pueden producir problemas a la hora de la síntesis.

VHDL permite los dos tipos de descripciones anteriormente indicadas: estructural y comportamental.

La **descripción estructural** soportada por VHDL se parece a un *netlist*, donde se especifican componentes y sus conexiones.

La **descripción comportamental** soportada por el VHDL es útil en simulación ya que permite definir el comportamiento del sistema sin conocer o comprometerse en su estructura interna. Este tipo de descripción va adquiriendo cada día más importancia y es más utilizada porque las actuales herramientas de síntesis permiten la **creación automática de circuitos** a partir de una descripción de su funcionamiento.

Dependiendo del nivel de abstracción y del modo en que se ejecutan las instrucciones, la descripción comportamental se divide a su vez en dos estilos: algorítmica y flujo de datos.

Por lo tanto VHDL, dependiendo del **nivel de abstracción**, presenta tres estilos de descripción de circuitos. El menos abstracto es la descripción puramente estructural. Los otros dos estilos de descripción funcional o comportamental presentan las siguientes diferencias. El **estilo algorítmico** utiliza procesos cuyas instrucciones internas se ejecutan en serie. El **estilo flujo de datos** establece, como su nombre indica, un flujo de datos entre distintos módulos que contienen funciones y operadores que se ejecutan de forma concurrente.

Se ha de tener en cuenta que el código VHDL escrito no siempre va a describir un circuito de forma óptima al ser traducido por la herramienta de compilación.



Diferentes descripciones producen diferentes, aunque equivalentes, ecuaciones de diseño pudiéndose dar, sin embargo, disposiciones diferentes de los recursos.

En los siguientes apartados se indican las principales características de estos tres estilos de descripción. Para que se puedan apreciar dichas características, se describe el mismo circuito (comparador de cuatro bits) en los tres estilos de arquitectura.

Se indica a continuación la entidad, para evitar su repetición, al ser común a las tres descripciones.

```

ENTITY comparador IS
    PORT (a,b: IN bit_vector(3 DOWNTO 0);
           igual: OUT bit
           );
END comparador;

```

### B.4.17.1 DESCRIPCIÓN ALGORÍTMICA

La descripción behavioral o comportamental algorítmica es la más cercana a un lenguaje convencional de descripción *software* (SDL). Está formada por el bloque **PROCESS**, que es una sentencia concurrente, similar a una subrutina, cuyas instrucciones se ejecutan secuencialmente. Por esta razón, una arquitectura con un solo proceso es equivalente a un algoritmo ejecutado secuencialmente.

En esta descripción no se hace ninguna referencia a la estructura a seguir para implementar el algoritmo en *hardware*, es decir, no se indican ni componentes, ni interconexiones, ni tecnologías, sólo su comportamiento o funcionamiento.

El siguiente fragmento de código corresponde a un circuito comparador de cuatro bits, en estilo comportamental algorítmica o behavioral (su entidad se ha descrito anteriormente).

```

ARCHITECTURE behavioral OF comparador IS
BEGIN
    comp: PROCESS (a,b)
        BEGIN
            IF a=b THEN igual <= '1';
                ELSE igual <= '0';
            END IF;
        END PROCESS;
END behavioral;

```

### B.4.17.2 DESCRIPCIÓN DE FLUJO DE DATOS

**Descripción comportamental por flujo de datos** (*dataflow*), o también llamada de **transferencia entre registros (RTL)**, debido a que la información es transferida entre señales, de las entradas a las salidas sin el uso de asignaciones secuenciales. Permite la paralelización de instrucciones siendo una descripción concurrente.

La descripción de flujo de datos es cercana a la realización física o estructural aunque sin llegar a serlo, ya que si bien describe componentes y asigna señales, no constituye una lista de componentes e interconexiones.

Los dos siguientes fragmentos de código son equivalentes y corresponden a un circuito comparador de cuatro bits, en estilo comportamental por flujo de datos (su entidad se ha descrito anteriormente).

```
-- Estilo flujo de datos. La información pasa a la salida sin el uso de
-- sentencias secuenciales
```

```
ARCHITECTURE dataflow1 OF comparador IS
```

```
BEGIN
```

```
    Igual<='1' WHEN (a=b) ELSE '0';
```

```
END dataflow1;
```

```
-- Estilo flujo de datos. La información pasa a la salida sin el uso de
-- sentencias secuenciales
```

```
ARCHITECTURE dataflow2 OF comparador IS
```

```
BEGIN
```

```
    Igual<= not(a(0) xor b(0))
           and not(a(1) xor b(1))
           and not(a(2) xor b(2))
           and not(a(3) xor b(3));
```

```
END dataflow2;
```

### B.4.17.3 DESCRIPCIÓN ESTRUCTURAL

La descripción estructural es la más cercana a la realización física de un circuito. Puede considerarse como una lista de componentes e interconexiones (*Netlist*). La descripción es menos clara y mucho más extensa que las anteriormente indicadas. Los componentes `xnor2` y `and4` (en el ejemplo siguiente) se suelen definir en una biblioteca y para las conexiones se utilizan señales internas definidas al principio de la arquitectura. A cada componente se le asigna un identificador, hecho que se denomina **replicación**.

El estilo estructural permite descomponer un diseño en unidades más pequeñas y manejables, pudiéndose repartir cada una de estas unidades en diferentes equipos de trabajo para su desarrollo. El estilo estructural se usa cuando se quiere un mayor control sobre la síntesis.

### Ejemplo:

Fragmento de código corresponde a un circuito comparador de cuatro bits, en estilo estructural (su entidad se ha descrito anteriormente).

**ARCHITECTURE** estructural **OF** comparador **IS**

**SIGNAL** x: bit\_vector(0 TO 3);

**BEGIN**

u0: xnor2 **PORT MAP** (a(0), b(0), x(0));

u1: xnor2 **PORT MAP** (a(1), b(1), x(1));

u2: xnor2 **PORT MAP** (a(2), b(2), x(2));

u3: xnor2 **PORT MAP** (a(3), b(3), x(3));

u4: and4 **PORT MAP** (x(0), x(1), x(2), x(3), igual);

**END** estructural;

## B.4.17.4 DESCRIPCIÓN MIXTA

VHDL permite descripciones compuestas por dos o más de los estilos indicados anteriormente.

## B.4.18 Sentencias secuenciales

La mayoría de los lenguajes de descripción *software* (SDL) se caracterizan porque todas sus sentencias de asignación son de naturaleza secuencial. Esto implica que la ejecución del programa se efectúa siguiendo el orden en el que se hayan escrito dichas sentencias en el programa, siendo importante su posición.

VHDL realiza las asignaciones secuenciales a señales dentro del cuerpo de un proceso (**PROCESS**), por lo que en la compilación se tendrá en cuenta el orden en el que aparezcan estas asignaciones. Mediante este procedimiento el VHDL se comporta como cualquier otro lenguaje (SDL) de programación (C, Pascal, etc.).

### B.4.18.1 SENTENCIA PROCESS

La sentencia **PROCESS** es una construcción de VHDL usada para agrupar algoritmos. Esta sentencia está compuesta opcionalmente por una etiqueta seguida de dos puntos, la palabra reservada **PROCESS** y una lista de variables sensibles que cuando se modifican provocan la ejecución del proceso.

Dentro del proceso (**PROCESS**) se encuentran sentencias secuenciales (no concurrentes). Esto hace que el orden de las sentencias dentro del proceso sea importante, ya que se ejecuta una después de otra y los posibles cambios que se efectúen en las señales no serán visibles hasta después de evaluar todo el proceso. Esta característica define una de las particularidades de VHDL. Es importante considerar que las variables toman instantáneamente el valor asignado y que sólo tienen sentido de existencia dentro de un proceso. Las señales cambian su valor solamente cuando el proceso termina.

Su sintaxis VHDL es:

```
[identificador_proceso]
PROCESS [(lista_sensible)] [IS]
    [declaraciones]
BEGIN
    Instrucciones serie
END PROCESS [identificador_proceso];
```

Las sentencias que se pueden incluir dentro de los procesos, con ejecución secuencial son las habitualmente empleadas en los lenguajes SDL. Estas sentencias son:

- **Sentencia condicional:** **IF...THEN...ELSE**

Selecciona un conjunto de sentencias u otras a ejecutar según la evaluación de una condición booleana (**true** o **false**).

- **Sentencia de selección:** **CASE**

Especifica una serie de acciones según el valor que tenga una señal de selección.

- **Bucles:** **FOR** y **WHILE LOOP**

Ejecuta un grupo de sentencias un número determinado de veces o siempre que se cumpla una condición, respectivamente.

- **Interrupciones de bucles:** **EXIT** y **NEXT**

**EXIT** permite salir de un bucle. **NEXT** permite saltar una o más de las ejecuciones programadas en el bucle.

- **Sentencia de espera: WAIT**

Permite detener la ejecución de un proceso hasta que no se cumpla:

- ◆ un cambio en las señales de la lista sensible,
- ◆ una condición, o
- ◆ un tiempo límite.

Su sintaxis VHDL es:

[identificador\_wait]

**WAIT** [ON lista\_sensible)] [UNTIIL condición] [FOR tiempo\_límite]

WAIT sin parámetros detiene el proceso para siempre.

## B.4.19 Sentencias concurrentes

La propia naturaleza de los circuitos eléctricos hace necesario que VHDL soporte un tipo de asignación de señales que puedan establecer simultáneamente o en paralelo. En una asignación concurrente la señal que esté a la izquierda de la asignación es evaluada siempre que alguna de las señales de la derecha modifique su valor.

Su sintaxis VHDL es:

señal\_a <= señal\_b;

### B.4.19.1 ASIGNACIÓN CONCURRENTE CONDICIONAL

Es la asignación concurrente equivalente a la sentencia IF.

Su sintaxis VHDL es:

señal\_a <= señal\_b **WHEN** condición **ELSE** señal\_c;

### B.4.19.2 ASIGNACIÓN CONCURRENTE CON SELECCIÓN

Es la asignación concurrente equivalente a la sentencia CASE.

Su sintaxis VHDL es:

**WITH** expresión **SELECT**

señal\_a <= señal\_b **WHEN** valor\_expresion\_1,

señal\_c **WHEN** valor\_expresion\_2;

## B.4.20 Sentencias estructurales

Otra manera de realizar llamadas a subprogramas en VHDL es mediante sentencias estructurales, también llamadas **sentencias de instantación**. Con ellas se pueden utilizar componentes o circuitos, anteriormente definidos, sin tener que incluir éstos en la descripción actual. Para ello sólo se necesita realizar una llamada a dicho componente para usarlo con las especificaciones correspondientes del diseño actual. Su operatividad se parece a la de una biblioteca.

Las sentencias estructurales son concurrentes, situadas en la arquitectura, permiten llamar a un modelo fuera de cualquier proceso. Su principal elemento es **COMPONENT** (el componente), con su función **PORT MAP** (mapa puertos).

Para realizar la descripción estructural de un sistema es necesario conocer qué subsistemas o componentes lo forman y la interconexión entre ellos.

Antes de hacer referencia a los componentes, éstos se deben declarar. Si esta declaración se realiza en la arquitectura podrán ser utilizados en cualquier parte de la misma. Si la declaración de los componentes se hace en un paquete, se podrán utilizar en todas las arquitecturas que llamen a ese paquete.

El estilo estructural se reconoce fácilmente puesto que el aspecto funcional del programa no se puede deducir del código, por estar formado exclusivamente por componentes y las señales de conexión con otros componentes. Para comprender el comportamiento del sistema habría que conocer también el funcionamiento de los componentes, los cuales normalmente se encuentran en los paquetes.

### B.4.20.1 SUBPROGRAMAS

Los **subprogramas** describen algoritmos que pueden ser utilizados varias veces mediante llamadas. Un subprograma consta de una parte declarativa en el que se definen el nombre y los datos de entrada y salida, y una parte de sentencias o cuerpo del subprograma en el que se indica las operaciones que se realizan con los datos.

Hay dos tipos de subprogramas: las **funciones** y los **procedimientos**.

### B.4.20.2 FUNCIONES

Las funciones realizan cálculos y definen nuevos operadores. Pueden tener varios parámetros de entrada pero sólo devuelven un único valor o resultado en una expresión. Sus variables internas son locales, no siendo visibles en otras operaciones.

Una **función consta** de las siguientes partes:

- **Parte declarativa.** En ella se indican los argumentos a introducir en la función que sólo son de entrada y dentro de la cual se consideran constantes pues no se pueden modificar.

Se puede definir tipos, constantes, variables, etc., pero sólo existen cuando la función es llamada, creándose e inicializándose cuando esto ocurre. Éste es el motivo de **no poder incluir señales** en la parte declarativa.

Comienza con la palabra reservada **FUNCTION**, seguido del nombre de la función y entre paréntesis los argumentos de entrada y sus correspondientes tipos. Se termina con la palabra reservada **RETURN** y el tipo de resultado que entrega la función.

- **Cuerpo.** En él se describe y se obtiene el valor de salida en función de los argumentos. Se pueden usar señales y variables externas, pero no pueden modificarse, ni tampoco utilizar la sentencia **WAIT**.

Las primeras líneas del cuerpo de la función coinciden con la parte declarativa pero terminando con la palabra **IS**. A continuación se declaran tipos, subtipos, variables, etc., entre las que se incluye la variable del resultado que devuelve la función.

Entre las palabras **BEGIN** y **END** se incluye el algoritmo por el que se obtiene el resultado de la función y la sentencia **RETURN** que es el mecanismo para pasar el valor de la función.

### Ejemplo:

Se describe una función que realiza la suma de dos números binarios de cuatro bits.

La parte declarativa es:

```
FUNCTION sumar(a,b:std_logic_vector(3 DOWNT0 0))
RETURN std_logic_vector(3 DOWNT0 0);
```

El cuerpo de la función es:

```
FUNCTION sumar(a,b:std_logic_vector(3 DOWNT0 0))
RETURN std_logic_vector(3 DOWNT0 0) IS;
    VARIABLE suma:std_logic_vector;
    BEGIN
        suma:=a+b;
    RETURN suma;
END FUNCTION sumar;
```

**Ejemplo:**

Llamada a la función sumar.

**PROCESS**

```
VARIABLE dato_a, dato_b:std_logic_vector(3 DOWNTO 0);
```

```
VARIABLE salida: std_logic_vector(3 DOWNTO 0);
```

**BEGIN**

```
...
```

```
salida:=sumar(dato_a,dato_b);
```

```
...
```

**END PROCESS;**

En el ejemplo anterior se ha hecho una **asignación por posición** de las entradas, de tal forma que el compilador usa el valor dato\_a como a, ya que dato\_a es el primer argumento que se introduce en la función y a su vez a es el primer argumento incluido en la declaración de la función. Y por la misma razón dato\_b y b tienen la misma posición. Se puede también realizar una **asignación por nombre** de cualquiera de las siguientes formas, siendo ambas equivalentes:

```
Salida:=sumar(dato_a=>a, dato_b=>b);
```

```
Salida:=sumar(dato_b=>b, dato_a=>a);
```

Los argumentos asignados a la función deben corresponder con los tipos declarados en ella. De la misma forma la variable o señal a la que se asigna el valor de la función debe ser del mismo tipo que el objeto que se haya asignado como salida de la función.

**B.4.20.3 PROCEDIMIENTOS**

Los procedimientos pueden alterar los datos a los que tienen acceso, tanto internos como externos. Pueden devolver más de un valor y modificar los valores de los argumentos introducidos. Por estas razones se debe tener cuidado en su uso pues se pueden producir efectos colaterales.

Un **procedimiento consta** de:

- **Parte declarativa.** En ella se indican los argumentos a introducir en el procedimiento, pudiendo ser de tres modos: IN, OUT e INOUT.

**IN** No se puede modificar el valor del argumento, solamente usarlo, ya que dentro del procedimiento se considera como constante.



**OUT** Puede modificarse el valor del argumento pero no se puede leer su valor.

**INOUT** Puede modificarse y leerse el valor del argumento.

Como ocurre en las funciones se pueden declarar tipos, contantes, variables, etc., pero éstas serán locales, es decir, sólo existen dentro del procedimiento y se crearán y reiniciarán nuevamente cada vez que se llame al procedimiento.

Su estructura es muy parecida a la de la entidad, empieza con la palabra reservada **PROCEDURE** seguida del nombre procedimiento y entre paréntesis se declaran los objetos de entrada como si de una entidad se tratase. En los procedimientos no hace falta usar la palabra **RETURN**, ya que se especifica cuál de las señales son de entrada y salida.

- **Cuerpo del procedimiento.** En él se pueden modificar la señales y variables tanto internas como externas al procedimiento. Se puede usar la sentencia **WAIT**.

Su estructura comienza repitiendo la parte declarativa y entre las palabras reservadas **BEGIN** y **END**, se describen los algoritmos que constituyen el procedimiento.

### Ejemplo:

Se realiza anterior ejemplo cuyo enunciado es: suma de dos números binarios de cuatro bits, pero realizándose mediante un procedimiento.

Su parte declarativa es:

```
PROCEDURE sumar(a,b:std_logic_vector(3 DOWNT0 0),
suma: OUT std_logic_vector(3 DOWNT0 0));
```

El cuerpo de la función es:

```
PROCEDURE sumar(a,b:std_logic_vector(3 DOWNT0 0),
Suma: OUT std_logic_vector(3 DOWNT0 0));
BEGIN
Suma<=a+b;
END PROCEDURE sumar;
```

Una vez definido el procedimiento, éste se puede usar realizando una llamada en cualquier parte del programa, como por ejemplo:

```
PROCESS
VARIABLE dato_a, dato_b:std_logic_vector(3 DOWNT0 0);
VARIABLE salida: std_logic_vector(3 DOWNT0 0);
```

**BEGIN**

```
...
    sumar(dato_a,dato_b, salida);
...
```

**END PROCESS;**

Al igual que en una función, los tipos de los argumentos de entrada y salida, en la llamada a un procedimiento, deben coincidir con los declarados en el procedimiento.

Se debe tener en cuenta que la llamada a un procedimiento es una sentencia.

En el ejemplo anterior se ha hecho una asignación en los argumentos por posición, ya que si no se indica nada, el compilador asigna el primer argumento introducido en la llamada al procedimiento con el primer objeto en la declaración de dicho procedimiento, y así sucesivamente con el resto de los argumentos. Se puede hacer asignaciones por nombre, como por ejemplo:

```
sumar(salida=>suma, dato_a=>a, dato_b=>b);
```

**B.4.20.4 LLAMADA CONCURRENTE A SUBPROGRAMA**

La llamada a un subprograma (función o procedimiento) puede realizarse tanto en entornos concurrentes (arquitectura) como en entornos serie (dentro de un proceso). En la llamada concurrente a un subprograma su ejecución es como la de un proceso, considerando como lista sensible los argumentos del subprograma de tipo IN y INOUT.

Los subprogramas a usar deben ser visibles por el compilador, es decir, la biblioteca que los contenga debe estar declarada.

Su sintaxis VHDL es:

```
señal<=nombre_funcion(entradas);           -- Llamada a función
nombre_procedimiento(entradas,salidas);    -- Llamada a procedimiento
```

**B.4.21 Elementos avanzados de VHDL****B.4.21.1 BUSES Y RESOLUCIÓN DE SEÑALES**

Una de las características físicas que presentan los *buses* es la **contingencia**, producida cuando varias fuentes escriben al mismo tiempo sobre una misma señal o línea de *bus*. Para **evitar la contención**, los dispositivos, escriben en el *bus* a través de un *buffers* que puede ser:

- **triestado** (Z), de señal débil o,
- **colector abierto** (AND cableada), de señal fuerte.

Para considerar la posibilidad de que varias fuentes escriban sobre una misma línea, como ocurre en los *buses*, se debe decidir qué valor de todos los presentes en la línea se asigna. Para ello se utilizan los tipos resueltos.

Un **tipo resuelto** se define mediante una declaración de subtipo y el nombre de la función de resolución, que calcula el valor de la fuente, en función de todas las asignaciones que se están produciendo.

Dentro de los tipos predefinidos se encuentran **bit** y **bit\_vector** que son no resueltos y no se les puede usar en *buses* ni en señales en los que varios procesos escriban a un mismo tiempo. **Std\_logic** y **std\_logic\_vector** poseen nueve niveles lógicos y función de resolución en el paquete **std\_logic\_1164** de la biblioteca **IEEE**, sin embargo **std\_ulogic** y **std\_ulogic\_vector** son los mismos anteriores pero sin función de resolución (no resueltos).

### B.4.21.2 PUNTEROS EN VHDL

VHDL posibilita la creación de variables de forma dinámica, reservando espacio en memoria. Hay estructuras con tamaño difícil de concretar inicialmente, siendo éste variable y modificable con el tiempo, como listas de colas (FIFO), pilas (LIFO). Estos elementos se pueden definir de forma dinámica, de manera que en cada instante sólo se utilice la capacidad necesaria, optimizando los recursos del sistema, sin compromisos previos y sin el peligro de desbordamiento.

Como en los lenguajes SDL, el VHDL realiza las asignaciones de memoria dinámica mediante **punteros** o direcciones de memoria que apuntan a una variable de nueva creación. Las variables a almacenar se organizan mediante estructuras de tipo registro, enlazándose entre ellas mediante punteros.

Mediante la palabra reservada **ACCESS** se define un tipo de datos que permite la reserva de memoria dinámica. Se debe tener en cuenta que los punteros no pueden ser señales, sólo son variables y siempre con ejecución serie, dentro de los procesos (PROCESS).

La palabra reservada **NEW** crea un nuevo puntero. Si se crea un puntero sin definir su valor éste toma por defecto **NULL**. Para liberar la memoria de una variable que no se necesita se utiliza el procedimiento **deallocate**.

### B.4.21.3 FICHEROS EN VHDL

VHDL permite trabajar con ficheros, siendo una de sus aplicaciones como banco de pruebas en el entorno de la simulación. Los procedimientos implícitos asociados a un fichero son: lectura, escritura y final de fichero.

## B.5 NOTACIÓN BNF DEL LENGUAJE VHDL'93

### B.5.1 Introducción

**Jhon Backus** y **Peter Naur** fueron los primeros que desarrollaron una notación formal, denominada BNF, para describir la sintaxis de un lenguaje, siendo actualmente la notación más utilizada para especificar las reglas sintácticas de un lenguaje de programación. **BNF** es el acrónimo de "*Backus Naur Form*".

### B.5.2 Notación BNF

Los **símbolos de la notación BNF** son los siguientes:

- ::=** significa "se define como"
- |** significa "o" y es utilizado para indicar varias opciones posibles.
- [ ]** Los elementos opcionales se ponen entre corchetes.
- { }** Los elementos que se repitan se cierran entre llaves.

**Tipo de letra:** Se puede distinguir los terminales y no terminales según el tipo de letra. Por ejemplo, es muy normal que las palabras clave (terminales) se pongan en negrilla mientras que los no-terminales se dejen con la letra normal.

### B.5.3 Descripción completa de la sintaxis en notación BNF

```
abstract_literal ::= decimal_literal | based_literal
```

```
access_type_definition ::= access subtype_indication
```

```
actual_designator ::=  
    expression  
    | signal_name  
    | variable_name  
    | file_name  
    | open
```

```
actual_parameter_part ::= parameter_association_list
```

```
actual_part ::=  
    actual_designator
```

```

    | function_name ( actual_designator )
    | type_mark ( actual_designator )
adding_operator ::= + | - | &
aggregate ::=
    ( element_association { , element_association } )
alias_declaration ::=
    alias alias_designator [ : subtype_indication ] is name [ signature ] ;
alias_designator ::= identifier | character_literal | operator_symbol
allocator ::=
    new subtype_indication
    | new qualified_expression
architecture_body ::=
    architecture identifier of entity_name is
        architecture_declarative_part
    begin
        architecture_statement_part
    end [ architecture ] [ architecture_simple_name ] ;
architecture_declarative_part ::=
    { block_declarative_item }
architecture_statement_part ::=

    { concurrent_statement }
array_type_definition ::=
    unconstrained_array_definition | constrained_array_definition
assertion ::=
    assert condition
        [ report expression ]
        [ severity expression ]
assertion_statement ::=      [ label : ] assertion ;
association_element ::=
    [ formal_part => ] actual_part
association_list ::=
    association_element { , association_element }
attribute_declaration ::=
    attribute identifier : type_mark ;
attribute_designator ::= attribute_simple_name
attribute_name ::=
    prefix [ signature ] ' attribute_designator [ ( expression ) ]

```

```
attribute_specification ::=
    attribute attribute_designator of entity_specification is expression ;

base ::= integer

base_specifier ::= B | O | X

base_unit_declaration ::= identifier ;

based_integer ::=
    extended_digit { [ underline ] extended_digit }

based_literal ::=
    base # based_integer [ . based_integer ] # [ exponent ]

basic_character ::=
    basic_graphic_character | format_effector

basic_graphic_character ::=
    upper_case_letter | digit | special_character | space_character

basic_identifier ::=
    letter { [ underline ] letter_or_digit }

binding_indication ::=
    [ use entity_aspect ]
    [ generic_map_aspect ]
    [ port_map_aspect ]

bit_string_literal ::= base_specifier " bit_value "

bit_value ::= extended_digit { [ underline ] extended_digit }

block_configuration ::=
    for block_specification
        { use_clause }
        { configuration_item }
    end for ;

block_declarative_item ::=
    subprogram_declaration
    | subprogram_body
    | type_declaration
    | subtype_declaration
    | constant_declaration
    | signal_declaration
    | shared_variable_declaration
    | file_declaration
    | alias_declaration
    | component_declaration
    | attribute_declaration
    | attribute_specification
    | configuration_specification
    | disconnection_specification
```

```
| use_clause
| group_template_declaration
| group_declaration

block_declarative_part ::=
    { block_declarative_item }

block_header ::=
    [ generic_clause
    [ generic_map_aspect ; ] ]
    [ port_clause
    [ port_map_aspect ; ] ]

block_specification ::=
    architecture_name
    | block_statement_label
    | generate_statement_label [ ( index_specification ) ]

block_statement ::=
    block_label :
        block [ ( guard_expression ) ] [ is ]
            block_header
            block_declarative_part
        begin
            block_statement_part
        end block [ block_label ] ;

block_statement_part ::=
    { concurrent_statement }

case_statement ::=
    [ case_label : ]
        case expression is
            case_statement_alternative
            { case_statement_alternative }
        end case [ case_label ] ;

case_statement_alternative ::=
    when choices =>
        sequence_of_statements

character_literal ::= ' graphic_character '

choice ::=
    simple_expression
    | discrete_range
    | element_simple_name
    | others

choices ::= choice { | choice }
```

```
component_configuration ::=
    for component_specification
        [ binding_indication ; ]
        [ block_configuration ]
    end for ;

component_declaration ::=
    component identifier [ is ]
        [ local_generic_clause ]
        [ local_port_clause ]
    end component [ component_simple_name ] ;

component_instantiation_statement ::=
    instantiation_label :
        instantiated_unit
            [ generic_map_aspect ]
            [ port_map_aspect ] ;

component_specification ::=
    instantiation_list : component_name

composite_type_definition ::=
    array_type_definition
    | record_type_definition

concurrent_assertion_statement ::=
    [ label : ] [ postponed ] assertion ;

concurrent_procedure_call_statement ::=
    [ label : ] [ postponed ] procedure_call ;

concurrent_signal_assignment_statement ::=
    [ label : ] [ postponed ] conditional_signal_assignment
    | [ label : ] [ postponed ] selected_signal_assignment

concurrent_statement ::=
    block_statement
    | process_statement
    | concurrent_procedure_call_statement
    | concurrent_assertion_statement
    | concurrent_signal_assignment_statement
    | component_instantiation_statement
    | generate_statement

condition ::= boolean_expression

condition_clause ::= until condition

conditional_signal_assignment ::=
    target <= options conditional_waveforms ;

conditional_waveforms ::=
    { waveform when condition else }
```



```
    waveform [ when condition ]  
configuration_declaration ::=  
    configuration identifier of entity_name is  
        configuration_declarative_part  
        block_configuration  
    end [ configuration ] [ configuration_simple_name ] ;  
configuration_declarative_item ::=  
    use_clause  
    | attribute_specification  
    | group_declaration  
configuration_declarative_part ::=  
    { configuration_declarative_item }  
configuration_item ::=  
    block_configuration  
    | component_configuration  
configuration_specification ::=  
    for component_specification binding_indication ;  
constant_declaration ::=  
    constant identifier_list : subtype_indication [ := expression ] ;  
constrained_array_definition ::=  
    array index_constraint of element_subtype_indication  
constraint ::=  
    range_constraint  
    | index_constraint  
context_clause ::= { context_item }  
context_item ::=  
    library_clause  
    | use_clause  
decimal_literal ::= integer [ . integer ] [ exponent ]  
declaration ::=  
    type_declaration  
    | subtype_declaration  
    | object_declaration  
    | interface_declaration  
    | alias_declaration  
    | attribute_declaration  
    | component_declaration  
    | group_template_declaration  
    | group_declaration  
    | entity_declaration
```

```

    | configuration_declaration
    | subprogram_declaration
    | package_declaration
delay_mechanism ::=
    transport
    | [ reject time_expression ] inertial
design_file ::= design_unit { design_unit }
design_unit ::= context_clause library_unit
designator ::= identifier | operator_symbol
direction ::= to | downto
disconnection_specification ::=
    disconnect guarded_signal_specification after time_expression ;
discrete_range ::= discrete_subtype_indication | range
element_association ::=
    [ choices => ] expression
element_declaration ::=
    identifier_list : element_subtype_definition ;
element_subtype_definition ::= subtype_indication
entity_aspect ::=
    entity entity_name [ ( architecture_identifier ) ]
    | configuration configuration_name
    | open
entity_class ::=
    entity      | architecture | configuration
    | procedure | function   | package
    | type     | subtype    | constant
    | signal    | variable    | component
    | label   | literal     | units
    | group  | file
entity_class_entry ::=          entity_class [ <> ]
entity_class_entry_list ::=
    entity_class_entry { , entity_class_entry }
entity_declaration ::=
    entity identifier is
        entity_header
        entity_declarative_part
    [ begin
        entity_statement_part ]
    end [ entity ] [ entity_simple_name ] ;

```

```
entity_declarative_item ::=
    subprogram_declaration
    | subprogram_body
    | type_declaration
    | subtype_declaration
    | constant_declaration
    | signal_declaration
    | shared_variable_declaration
    | file_declaration
    | alias_declaration
    | attribute_declaration
    | attribute_specification
    | disconnection_specification
    | use_clause
    | group_template_declaration
    | group_declaration

entity_declarative_part ::=
    { entity_declarative_item }

entity_designator ::= entity_tag [ signature ]

entity_header ::=
    [ formal_generic_clause ]
    [ formal_port_clause ]

entity_name_list ::=
    entity_designator { , entity_designator }
    | others
    | all

entity_specification ::=
    entity_name_list : entity_class

entity_statement ::=
    concurrent_assertion_statement
    | passive_concurrent_procedure_call_statement
    | passive_process_statement

entity_statement_part ::=
    { entity_statement }

entity_tag ::= simple_name | character_literal | operator_symbol

enumeration_literal ::= identifier | character_literal

enumeration_type_definition ::=
    ( enumeration_literal { , enumeration_literal } )

exit_statement ::=
    [ label : ] exit [ loop_label ] [ when condition ] ;

exponent ::= E [ + ] integer | E - integer
```

```
expression ::=
    relation { and relation }
  | relation { or relation }
  | relation { xor relation }
  | relation [ nand relation ]
  | relation [ nor relation ]
  | relation { xnor relation }

extended_digit ::= digit | letter

extended_identifier ::=
    \ graphic_character { graphic_character } \

factor ::=
    primary [ ** primary ]
  | abs primary
  | not primary

file_declaration ::=
    file identifier_list : subtype_indication file_open_information ] ;

file_logical_name ::= string_expression

file_open_information ::=
    [ open file_open_kind_expression ] is file_logical_name

file_type_definition ::=
    file of type_mark

floating_type_definition ::= range_constraint

formal_designator ::=
    generic_name
  | port_name
  | parameter_name

formal_parameter_list ::= parameter_interface_list

formal_part ::=
    formal_designator
  | function_name ( formal_designator )
  | type_mark ( formal_designator )

full_type_declaration ::=
    type identifier is type_definition ;

function_call ::=
    function_name [ ( actual_parameter_part ) ]

generate_statement ::=
    generate_label :
        generation_scheme generate
            [ { block_declarative_item }
        begin ]
```

```

        { concurrent_statement }
    end generate [ generate_label ] ;

generation_scheme ::=
    for generate_parameter_specification
    | if condition

generic_clause ::=
    generic ( generic_list ) ;

generic_list ::= generic_interface_list

generic_map_aspect ::=
    generic map ( generic_association_list )

graphic_character ::=
    basic_graphic_character | lower_case_letter | other_special_character

group_constituent ::= name | character_literal

group_constituent_list ::= group_constituent { , group_constituent }

group_template_declaration ::=
    group identifier is ( entity_class_entry_list ) ;

group_declaration ::=
    group identifier : group_template_name ( group_constituent_list ) ;

guarded_signal_specification ::=
    guarded_signal_list : type_mark

identifier ::=
    basic_identifier | extended_identifier

identifier_list ::= identifier { , identifier }

if_statement ::=
    [ if_label : ]
        if condition then
            sequence_of_statements
        { elsif condition then
            sequence_of_statements }
        [ else
            sequence_of_statements ]
        end if [ if_label ] ;

incomplete_type_declaration ::= type identifier ;

index_constraint ::= ( discrete_range { , discrete_range } )

index_specification ::=
    discrete_range
    | static_expression

index_subtype_definition ::= type_mark range <>

```

```

indexed_name ::= prefix ( expression { , expression } )
instantiated_unit ::=
    [ component ] component_name
    | entity entity_name [ ( architecture_identifier ) ]
    | configuration configuration_name
instantiation_list ::=
    instantiation_label { , instantiation_label }
    | others
    | all
integer ::= digit { [ underline ] digit }
integer_type_definition ::= range_constraint
interface_constant_declaration ::=
    [ constant ] identifier_list : [ in ] subtype_indication
    [ := static_expression ]
interface_declaration ::=
    interface_constant_declaration
    | interface_signal_declaration
    | interface_variable_declaration
    | interface_file_declaration
interface_element ::= interface_declaration
interface_file_declaration ::=
    file identifier_list : subtype_indication
interface_list ::=
    interface_element { ; interface_element }
interface_signal_declaration ::=
    [signal] identifier_list : [ mode ] subtype_indication [ bus ] [ :=
static_expression ]
interface_variable_declaration ::=
    [variable] identifier_list : [ mode ] subtype_indication [ := static_expression ]
iteration_scheme ::=
    while condition
    | for loop_parameter_specification
label ::= identifier
letter ::= upper_case_letter | lower_case_letter
letter_or_digit ::= letter | digit
library_clause ::= library logical_name_list ;
library_unit ::=
    primary_unit
    | secondary_unit

```

```

literal ::=
    numeric_literal
    | enumeration_literal
    | string_literal
    | bit_string_literal
    | null

logical_name ::= identifier

logical_name_list ::= logical_name { , logical_name }

logical_operator ::= and | or | nand | nor | xor | xnor

loop_statement ::=
    [ loop_label : ]
        [ iteration_scheme ] loop
            sequence_of_statements
        end loop [ loop_label ] ;

miscellaneous_operator ::= ** | abs | not

mode ::= in | out | inout | buffer | linkage

multiplying_operator ::= * | / | mod | rem

name ::=
    simple_name
    | operator_symbol
    | selected_name
    | indexed_name
    | slice_name
    | attribute_name

next_statement ::=
    [ label : ] next [ loop_label ] [ when condition ] ;

null_statement ::= [ label : ] null ;

numeric_literal ::=
    abstract_literal
    | physical_literal

object_declaration ::=
    constant_declaration
    | signal_declaration
    | variable_declaration
    | file_declaration

operator_symbol ::= string_literal

options ::= [ guarded ] [ delay_mechanism ]

package_body ::=
    package body package_simple_name is
        package_body_declarative_part

```

```

end [ package body ] [ package_simple_name ] ;
package_body_declarative_item ::=
    subprogram_declaration
    | subprogram_body
    | type_declaration
    | subtype_declaration
    | constant_declaration
    | shared_variable_declaration
    | file_declaration
    | alias_declaration
    | use_clause
    | group_template_declaration
    | group_declaration
package_body_declarative_part ::=
    { package_body_declarative_item }
package_declaration ::=
    package identifier is
        package_declarative_part
    end [ package ] [ package_simple_name ] ;
package_declarative_item ::=
    subprogram_declaration
    | type_declaration
    | subtype_declaration
    | constant_declaration
    | signal_declaration
    | shared_variable_declaration
    | file_declaration
    | alias_declaration
    | component_declaration
    | attribute_declaration
    | attribute_specification
    | disconnection_specification
    | use_clause
    | group_template_declaration
    | group_declaration
package_declarative_part ::=
    { package_declarative_item }
parameter_specification ::=
    identifier in discrete_range
physical_literal ::= [ abstract_literal ] unit_name
physical_type_definition ::=
    range_constraint
    units
        base_unit_declaration

```



```
        { secondary_unit_declaration }
    end units [ physical_type_simple_name ]

port_clause ::=
    port ( port_list ) ;

port_list ::= port_interface_list

port_map_aspect ::=
    port map ( port_association_list )

prefix ::=
    name
    | function_call

primary ::=
    name
    | literal
    | aggregate
    | function_call
    | qualified_expression
    | type_conversion
    | allocator
    | ( expression )

primary_unit ::=
    entity_declaration
    | configuration_declaration
    | package_declaration

procedure_call ::= procedure_name [ ( actual_parameter_part ) ]

procedure_call_statement ::=
    [ label : ] procedure_call ;

process_declarative_item ::=
    subprogram_declaration
    | subprogram_body
    | type_declaration
    | subtype_declaration
    | constant_declaration
    | variable_declaration
    | file_declaration
    | alias_declaration
    | attribute_declaration
    | attribute_specification
    | use_clause
    | group_template_declaration
    | group_declaration

process_declarative_part ::=
    { process_declarative_item }
```

```

process_statement ::=
    [ process_label : ]
        [ postponed ] process [ ( sensitivity_list ) ] [ is ]
            process_declarative_part
        begin
            process_statement_part
        end [ postponed ] process [ process_label ] ;

process_statement_part ::=
    { sequential_statement }

qualified_expression ::=
    type_mark ' ( expression )
    | type_mark ' aggregate

range ::=
    range_attribute_name
    | simple_expression direction simple_expression
    range_constraint ::= range range

record_type_definition ::=
    record
        element_declaration
        { element_declaration }
    end record [ record_type_simple_name ]

relation ::=
    shift_expression [ relational_operator shift_expression ]

relational_operator ::=      = | /= | < | <=      | > | >=

report_statement ::=
    [ label : ]
        report expression
            [ severity expression ] ;

return_statement ::=
    [ label : ] return [ expression ] ;

scalar_type_definition ::=
    enumeration_type_definition | integer_type_definition
    | floating_type_definition   | physical_type_definition

secondary_unit ::=
    architecture_body
    | package_body

secondary_unit_declaration ::=      identifier = physical_literal ;

selected_name ::= prefix . suffix

selected_signal_assignment ::=
    with expression select
        target <= options selected_waveforms ;

```

```

selected_waveforms ::=
    { waveform when choices , }
    waveform when choices

sensitivity_clause ::= on sensitivity_list

sensitivity_list ::= signal_name { , signal_name }

sequence_of_statements ::=
    { sequential_statement }

sequential_statement ::=
    wait_statement
    | assertion_statement
    | report_statement
    | signal_assignment_statement
    | variable_assignment_statement
    | procedure_call_statement
    | if_statement
    | case_statement
    | loop_statement
    | next_statement
    | exit_statement
    | return_statement
    | null_statement

shift_expression ::=
    simple_expression [ shift_operator simple_expression ]

shift_operator ::= sll | srl | sla | sra | rol | ror

sign ::= + | -

signal_assignment_statement ::=
    [ label : ] target <= [ delay_mechanism ] waveform ;

signal_declaration ::=
    signal identifier_list : subtype_indication [ signal_kind ] [ := expression ] ;

signal_kind ::=      register | bus

signal_list ::=
    signal_name { , signal_name }
    | others
    | all

signature ::= [ [ type_mark { , type_mark } ] [ return type_mark ] ]

simple_expression ::=
    [ sign ] term { adding_operator term }

simple_name ::=      identifier

slice_name ::=      prefix ( discrete_range )

string_literal ::= " { graphic_character } "

```

```
subprogram_body ::=
    subprogram_specification is
        subprogram_declarative_part
    begin
        subprogram_statement_part
    end [ subprogram_kind ] [ designator ] ;

subprogram_declaration ::=
    subprogram_specification ;

subprogram_declarative_item ::=
    subprogram_declaration
    | subprogram_body
    | type_declaration
    | subtype_declaration
    | constant_declaration
    | variable_declaration
    | file_declaration
    | alias_declaration
    | attribute_declaration
    | attribute_specification
    | use_clause
    | group_template_declaration
    | group_declaration

subprogram_declarative_part ::=
    { subprogram_declarative_item }

subprogram_kind ::= procedure | function

subprogram_specification ::=
    procedure designator [ ( formal_parameter_list ) ]
    | [ pure | impure ] function designator [ ( formal_parameter_list ) ]
    return type_mark

subprogram_statement_part ::=
    { sequential_statement }

subtype_declaration ::=
    subtype identifier is subtype_indication ;

subtype_indication ::=
    [ resolution_function_name ] type_mark [ constraint ]

suffix ::=
    simple_name
    | character_literal
    | operator_symbol
    | all

target ::=
    name
```

```

    | aggregate
term ::=
    factor { multiplying_operator factor }
timeout_clause ::= for time_expression
type_conversion ::= type_mark ( expression )
type_declaration ::=
    full_type_declaration
    | incomplete_type_declaration
type_definition ::=
    scalar_type_definition
    | composite_type_definition
    | access_type_definition
    | file_type_definition
type_mark ::=
    type_name
    | subtype_name
unconstrained_array_definition ::=
    array ( index_subtype_definition { , index_subtype_definition } )
    of element_subtype_indication
use_clause ::=
    use selected_name { , selected_name } ;
variable_assignment_statement ::=
    [ label : ] target := expression ;
variable_declaration ::=
    [ shared ] variable identifier_list : subtype_indication [ := expression ] ;
wait_statement ::=
    [ label : ] wait [ sensitivity_clause ] [ condition_clause ] [ timeout_clause ] ;
waveform ::=
    waveform_element { , waveform_element }
    | unaffected
waveform_element ::=
    value_expression [ after time_expression ]
    | null [ after time_expression ]

```

## SIMULADORES VHDL

---

---

**Objetivos:**

- Conocer cómo se realiza la simulación VHDL en una familia lógica.
- Conocer cómo se realiza la notificación en la simulación de sucesos.
- Conocer los métodos utilizados en los bancos de pruebas.

**Contenido:** Se indican determinados aspectos de VHDL en simulación, como son: la asignación de los retardos, la simulación guiada por eventos, los distintos niveles lógicos, las notificaciones de sucesos y las descripciones de los bancos de pruebas.

**Simulación:** Mediante los programas de simulación *OrCAD Demo v9* y el simulador de VHDL *VeriBest VB99.0* se realizan comprobaciones de los ejemplos que se presentan: puertas inversoras de distintas subfamilias, puertas inversoras con los retardos, etc.

## C.1 SIMULACIÓN EN VHDL DE UNA FAMILIA LÓGICA

Para **simular una familia lógica mediante el lenguaje de descripción *hardware* VHDL** se deben especificar las características particulares que definen a dicha familia lógica y que la diferencia de las demás, como son principalmente: sus retardos, los distintos niveles lógicos que sean posibles adoptar y las salidas especiales que pueden presentar.

### C.1.1 Asignaciones con retrasos

Es importante en este punto recordar la **diferencia entre variables y señales** que se indicó en el apartado B.4.13.4 Resumen de diferencias entre variables y señales del Apéndice B.

- Las **variables** son elementos abstractos a los que se puede modificar su valor instantáneamente dentro de un proceso secuencial.
- Las **señales** con significado físico se usan en procesos secuenciales y concurrentes. Tienen dos partes: una en la cual se escribe (denominada fuente) y otra en la que se lee, que no tiene por qué ser igual a lo escrito. Existe conexión entre ellas, siendo posible su desconexión mediante la palabra clave **NULL**. Pueden ser de tipo: **normal** (cuando no se pueden desconectar las fuentes), **bus** (con valor por defecto cuando todas las fuentes están desconectadas) y **register** (si conserva el último valor escrito cuando están desconectadas todas las fuentes).

Cuando en VHDL se asigna un retardo a una señal, lo que se hace es anotar en una lista (tipo cola) los valores futuros que tomará la señal ordenados en el tiempo. En cada asignación se van añadiendo nuevos eventos a la lista, ordenados según sus retardos asociados.

La **forma de asignar los retardos** puede clasificarse en retardos inerciales o retardos transportados.

Considérese un inversor con un retardo de propagación  $tp$  de 10 ns, cada vez que su señal de entrada  $a$  cambie, se introduce en la lista de eventos de la señal de salida  $S$  el valor de la entrada complementado, con un retardo de 10 ns. Es importante considerar si la salida  $S$  debe cambiar antes de los 10 ns por haberse aplicado en su entrada  $a$  un impulso de duración menor que su retardo, por ejemplo de 5 ns. En esta situación se deben considerar dos casos:

- que el segundo evento se añada a la lista, la cual ahora estaría formada por dos eventos, produciéndose el denominado **retardo transportado**, o,
- que el segundo evento se añada a la lista eliminando al anterior, produciéndose el denominado **retardo inercial**.

En la Figura C.1 se representan los cronogramas de un retardo inercial y transportado correspondientes a un inversor con un retardo  $t_p$  de 10 ns.

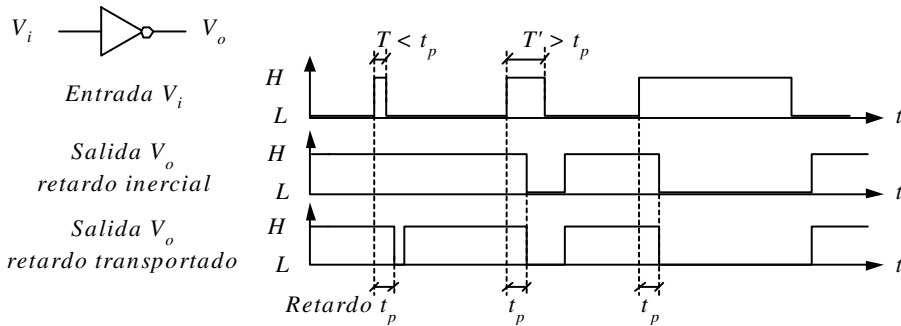


Figura C.1. Cronogramas de retardo inercial y transportado

Para la simulación mediante VHDL, la **sintaxis completa de la asignación** con retardos es la siguiente:

```
[identificador_asignacion]
señal <= [TRANSPORT | [REJECT tiempo] [INERTIAL]
        forma_onda {,forma_onda} | UNAFECTED;
```

Cuando se realiza una asignación con retardo en una señal, su valor se establece en su fuente y transcurrido el retardo se transfiere a la señal en sí, pudiéndose leer este valor. La palabra clave es **AFTER**, como por ejemplo:

```
S <= '1' AFTER 10 ns;
```

La forma de realizar varias asignaciones a una misma señal, en la misma sentencia, es mediante una lista de valores futuros ordenados en el tiempo, como por ejemplo:

```
S <= '1' AFTER 20 ns, '0' AFTER 25 ns, '1' AFTER 30 ns;
```

### Retardos inerciales

Muchos dispositivos lógicos requieren que sus señales de entrada tengan una duración mínima para que sean reconocidas o tengan efecto, es decir, ciertos



impulsos de entrada de corta duración no tienen efecto sobre la salida, pudiéndose considerar que son filtrados por el dispositivo (eliminados debido a su frecuencia).

En VHDL los retardos por defecto son inerciales. Ejemplo:

```
S <= not a AFTER 10 ns;
```

y

```
S <= INERTIAL not a AFTER 10 ns;
```

ambas sentencias son equivalentes.

Mediante la palabra clave **REJECT** se puede especificar el intervalo de tiempo durante el cual es posible que un nuevo evento elimine a los que existen en la cola, sin necesidad de que coincida con su retardo de propagación  $tp$ , tal y como ocurre en los retardos inerciales. Ejemplo:

```
S <= REJECT 3 ns INERTIAL not a AFTER 10 ns;
```

En este caso los impulsos en  $a$  inferiores a 3 ns son filtrados y no modifican la señal de salida  $S$ .

### Retardos transportados

Mediante la palabra clave **TRANSPORT** la salida del dispositivo siempre es afectada por un pulso de la señal de entrada, con independencia de su duración, creándose una lista de eventos y procesándose en el tiempo, en función de su retardo.

```
S <= TRANSPORT not a AFTER 10 ns;
```

En este caso todos los impulsos presentes en  $a$  modifican la señal de salida  $S$ . Ningún impulso es filtrado.

En el caso de asignaciones múltiples sólo la primera es inercial, siendo el resto transportadas, pues de lo contrario se perdería la información de toda la lista al ser sólo la última asignación válida, eliminándose las anteriores.

```
S <= '0' AFTER 5 ns, '1' AFTER 20 ns;
```

### Ejemplo:



Se describe y simula mediante el programa *VeriBest VB99.0* la respuesta de la salida *S* de un inversor con retardo de propagación de 10 ns que no responda a impulsos de entrada inferiores a 3 ns. Asimismo, se considera la salida *Y* de un inversor con retardo de 5 ns que complementa impulsos de cualquier duración. Declaración de entidades correspondientes a un selector de datos.

En la Figura C.2 se muestra la descripción en lenguaje VHDL de las puertas inversoras con los retardos indicados en el enunciado del problema, donde *a* es la señal de entrada común a ambos inversores y *S* e *Y* sus salidas. Este ejemplo se corresponde con el problema resuelto 6-18 del Capítulo 6, que se ha incluido de nuevo aquí por ser el más representativo de este caso.

```

06v0_01
-----
-- Fichero : D:\Ejemplos\Cap06\VBv99\06V0_01\06V0_01.vhd
-- Lib_trabajo: D:\Ejemplos\Cap06\VBv99\06V0_01\WORKLIB
-- Descripción de puertas NOT con retardos
-- inerciales y transportados
--
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY puerta_not IS
    PORT( a : in bit;
          S : out bit;
          Y : out bit);
END puerta_not;

-- Descripción comportamental

ARCHITECTURE flujo_datos OF puerta_not IS

BEGIN
    S<= REJECT 3 ns INERTIAL NOT a AFTER 10 ns;
    Y<= TRANSPORT NOT a AFTER 5 ns;

END flujo_datos;
Ln 29, Col 1

```

Figura C.2. Descripción en lenguaje VHDL de puertas inversoras con los retardos

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap06\VBv99\06V0\_01\06V0\_01.vpd**

En la Figura C.3 se muestra el resultado de la simulación que se obtiene en *VeriBest VB99.0*. Obsérvese que se puede describir que ciertos impulsos de entrada de corta duración no tengan efecto sobre la salida *S*, pudiéndose considerar que son filtrados por el dispositivo.

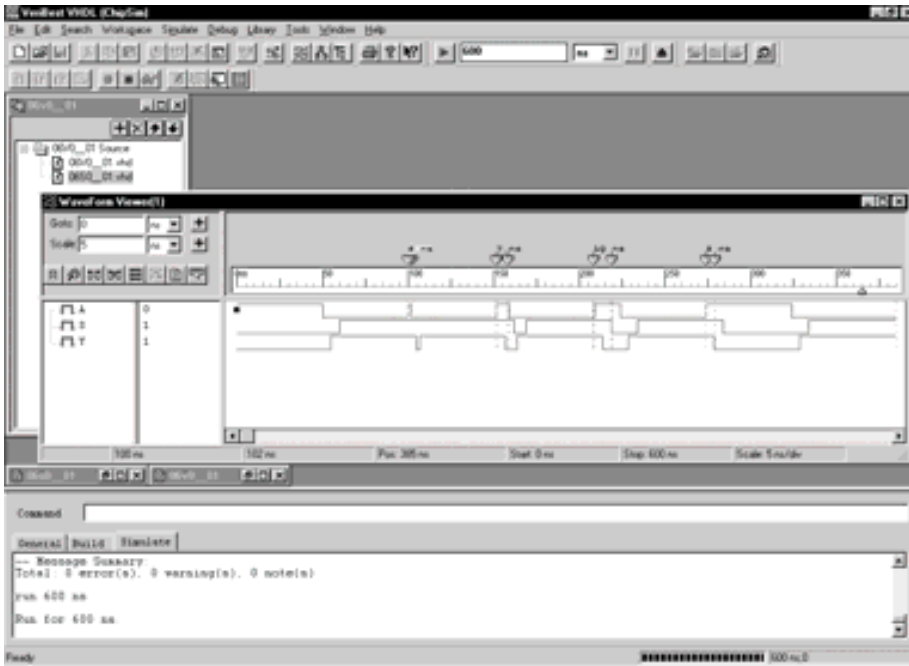


Figura C.3. Simulación a partir de descripción en lenguaje VHDL de puertas inversoras con retardos

### Ejemplo:



Se describe y simula mediante el programa *OrCAD Demo v9* las respuestas de salida de inversores correspondientes a las subfamilias: TTL-estándar, HC, HCT, LS y CMOS.

En la Figura C.4 se muestra la descripción en lenguaje VHDL de las puertas inversoras con sus correspondientes retardos, donde  $a$  es la señal de entrada común a todas las puertas e  $Y_{TTL}$ ,  $Y_{HC}$ ,  $Y_{HCT}$ ,  $Y_{LS}$ ,  $Y_{CMOS}$  sus salidas. Este ejemplo se corresponde con el problema resuelto 6-19 del Capítulo 6, incluido de nuevo por ser el más representativo de este caso.

La ruta y el nombre del fichero que contiene los datos de simulación de este sistema es la que se indica a continuación:

**D:\Ejemplos\Cap06\OrCAD\06R0\_03\06R0\_03.vpd**

En la Figura C.5 se muestra el resultado de la simulación que se obtiene con *OrCAD Demo v9*. Obsérvese los efectos (zonas de niveles indefinidos 'U') que producen los retardos, a determinadas frecuencias, en las respuestas de las puertas según la subfamilia, en particular en las tecnologías más lentas como por ejemplo TTL-estándar y CMOS.

```
06r0_03
-----
1:
2: -- Fichero : D:\Ejemplos\Cap06\Circuitos\06r0_03\06r0_03.vhd
3: -- Lib_trabajo: D:\Ejemplos\Cap06\Circuitos\06r0_03\WORKLIB
4: -- Descripción de puertas NOT de diferentes subfamilias
5: --
6: --
7: -----
8:
9: LIBRARY ieee;
10: USE ieee.std_logic_1164.all;
11:
12: ENTITY puertas_not IS
13:   PORT(
14:     a : in std_logic;
15:     Y_TTL : out std_logic;
16:     Y_BC : out std_logic;
17:     Y_NCT : out std_logic;
18:     Y_LS : out std_logic;
19:     Y_CMOS : out std_logic);
20: END puertas_not;
21:
22: -- Descripción comportamental
23:
24: ARCHITECTURE flujo_datos OF puertas_not IS
25:
26: BEGIN
27:   Y_TTL <= NOT a AFTER 22 ns;
28:   Y_BC <= NOT a AFTER 1500 ps;
29:   Y_NCT <= NOT a AFTER 800 ps;
30:   Y_LS <= NOT a AFTER 10 ns;
31:   Y_CMOS <= NOT a AFTER 80 ns;
32:
33: END flujo_datos;
```

Figura C.4. Descripción en VHDL de puertas inversoras de distintas subfamilias

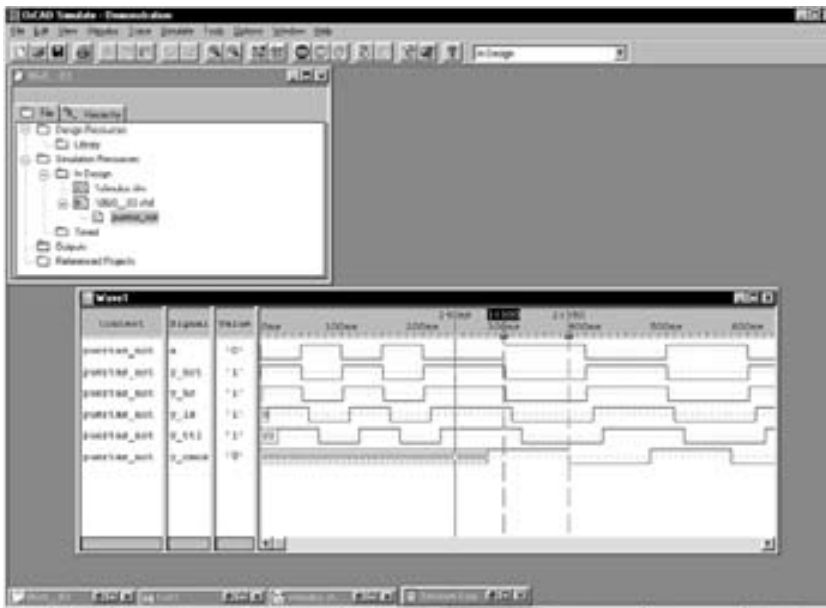


Figura C.5. Simulación en VHDL de puertas inversoras de diferentes subfamilias

## C.1.2 Simulación de niveles lógicos

Los tipos básicos **bit** cuyos valores lógicos sólo pueden ser '1' o '0' no se corresponden con la realidad en simulación, ya que no contemplan el estado de alta impedancia (salidas en triestado) o en colector/drenador abierto, ni determinadas situaciones reales que se producen cuando varias señales están conectadas a una misma línea de un *bus*.

Para modelar con una mayor aproximación a la realidad una señal o línea de un *bus*, se utiliza un modelo basado en **fuerzas y valores lógicos** que define los posibles estados lógicos que pueden adoptar una señal o nodo de un circuito lógico. Mediante los parámetros de fuerza y el valor lógico se puede modelar cualquier familia lógica.

En el modelo propuesto hay que especificar:

- El **valor lógico**, que puede ser: **1** (alto), **0** (bajo) y **X** (desconocido).
- y las **Fuerzas** que pueden ser:
  - F:** **Conexión directa.** Conexión de la salida a la alimentación o masa mediante una conexión directa. Es la de mayor fuerza.
  - S:** **Strong.** Conexión de la salida a la alimentación  $V_{CC}$  ( $V_{DD}$ ) o a masa GND mediante un transistor (salida en *Totem-Pole*).
  - R:** **Resistiva.** Conexión de la salida a alimentación o a masa mediante una resistencia (salida en colector/drenador o emisor/fuente abierto/a).
  - Z:** **Alta impedancia.** Conexión de la salida a alimentación o masa mediante una alta impedancia (salida triestado).
  - I:** **Indeterminada.** No se conoce la fuerza que hay en la línea.

Para evitar la asignación de una señal o línea de *bus* con dos cantidades (fuerza y valor lógico), en VHDL se puede expresar mediante una cantidad única, definida en un tipo enumerado **std\_logic** del paquete **std\_logic\_1164** de la biblioteca **IEEE**, representado en la Tabla C.1. Se observa en dicha tabla que en este tipo enumerado no está definida la fuerza *F*.

## C.2 NOTIFICACIÓN EN LA SIMULACIÓN DE SUCESOS

La notificación de sucesos consiste en avisar mediante un mensaje cuando se produce una circunstancia o suceso, como por ejemplo: la activación de una señal, la violación de retardos, etc. Esto es muy útil como medio de diagnóstico. Su palabra reservada es **ASSERT**.

Tabla C.1. Operadores predefinidos en VHDL, en el paquete *Std\_logic\_1164*

Std_logic	Fuerza-valor	Descripción	Puerta típica
'U'		No inicializado	
'X'	SX	Desconocido fuerte	<i>Totem-pole</i>
'0'	S0	0 fuerte	<i>Totem-pole</i>
'1'	S1	1 fuerte	<i>Totem-pole</i>
'Z'	ZX	Alta impedancia	Tiestado
'W'	RX	Desconocido resistivo	Em-Col abierto
'L'	R0	0 resistivo	Emisor abierto
'H'	R1	1 resistivo	Colector abierto
'_'		No importa	

Si no se cumple su condición asociada a **ASSERT** aparece el mensaje y el nivel de gravedad que tienen adjunto.

Su sintaxis VHDL es:

**ASSERT** condición [**REPORT** mensaje] [**SEVERITY** nivel\_gravedad];

Los niveles de gravedad son de tipo predefinido: *note*, *warning*, *error* y *failure*, siendo *error* por defecto.

Su ejecución puede ser concurrente o serie.

### C.3 BANCO DE PRUEBAS

Un banco de pruebas sirve para verificar y determinar la correcta funcionalidad del sistema diseñado. Tradicionalmente se realizaba mediante un prototipo físico, con alto riesgo y coste. VHDL permite acelerar este proceso con una exhaustiva simulación del modelo, que de ser correcta se pasa a su síntesis.

Los **bancos de pruebas** definen un conjunto de entradas, llamadas **patrones de test**, cuyo objetivo es comprobar el circuito o modelo VHDL. Hay herramientas CAD que definen estos vectores VHDL permitiendo modelar el banco de pruebas independiente de la herramienta de simulación y puede ser utilizado en cualquier fase del diseño con el consiguiente ahorro de tiempo, trabajo y coste.

Un banco de pruebas es una entidad sin entradas ni salidas y su arquitectura es de tipo estructural siendo sus señales internas las entradas/salidas del circuito. Este circuito a su vez es considerado como el único componente de la arquitectura del banco de pruebas a simular.

Al sintetizar, el mismo banco de pruebas utilizado en la pre-síntesis puede utilizarse en la post-síntesis.

Los bancos de pruebas se pueden construir según los siguientes métodos:

- **Método tabular.** Detecta las diferencias existentes entre la tabla de verdad que define el comportamiento del sistema (descrita de forma tabular) y su respuesta en simulación.
- Utilización de **ficheros de vectores de test**. Se basa en almacenar la tabla de verdad del comportamiento del sistema mediante vectores en un fichero y comparar posteriormente sus diferencias con la respuesta del sistema en simulación.
- **Metodología algorítmica.** Se utiliza principalmente cuando las tablas de verdad resultan demasiado extensas. La respuesta esperada del sistema se determina algorítmicamente y comparada, posteriormente, su diferencia con la respuesta del sistema en simulación.

## **SIMULACIÓN VHDL CON *ORCAD DEMO V9***

---

---

***Objetivos:***

- Conocer el procedimiento para realizar la simulación VHDL con la herramienta *OrCAD Demo v9*.
- Conocer los tipos de descripciones en VHDL: descripción comportamental algorítmica, descripción comportamental por flujo de datos y descripción estructural y su utilización con *OrCAD Demo v9*.
- Conocer cómo se generan ficheros de estímulos con esta herramienta.

***Contenido:*** Se describe cómo llevar a cabo la simulación VHDL utilizando la aplicación *OrCAD Demo v9*: la asignación de los retardos, la simulación guiada por eventos, los distintos niveles lógicos, las notificaciones de sucesos y las descripciones de los bancos de pruebas.

***Simulación:*** Mediante el programa de simulación *OrCAD Demo v9* se simula un circuito semisumador, descrito en lenguaje VHDL.



## D.1 CREACIÓN DE UN PROYECTO

**Pasos a realizar** para crear un nuevo proyecto:

- **Abrir la aplicación** Inicio/Programas/*OrCAD Demo/Simulate Demo*, visualizándose la ventana de la Figura D.1.



*Figura D.1. Pantalla inicial del simulador*

- **Crear un nuevo proyecto** mediante el comando ***File/New***, o el icono



mostrándose la ventana de la Figura D.2, en la que se solicita el tipo de fichero a crear, seleccionando *OrCAD Project*.

- En la ventana de la Figura D.3, el programa solicita: el **nombre del nuevo proyecto** y la localización de su **directorio**,

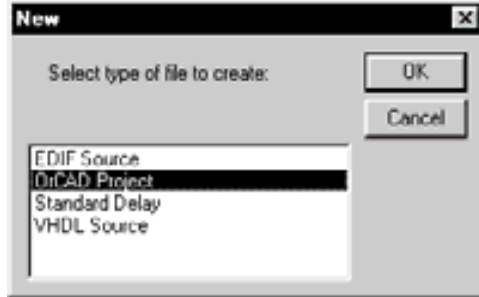


Figura D.2. Creación de un proyecto



Figura D.3. Asignación del nombre y directorio al proyecto

## D.2 AÑADIR DOCUMENTOS A UN PROYECTO

### Pasos a realizar:

- **Incluir en el proyecto ficheros VHDL** (\*.vhd, \*.vho) si éstos ya han sido previamente creados, como se muestra en la Figura D.4 y el **tipo de ficheros VHDL** (*Netlist*, *SimModel* o *Testbench*) indicados de la Figura D.5. En caso de no querer añadir documentos al proyecto seleccionar el botón **Cancelar**, pudiéndose crear posteriormente, mediante el editor del simulador, estos ficheros VHDL y añadirles al proyecto.

**Nota:** También pueden ser incluidos en el proyecto ficheros de tipo: estímulos, (\*.stm), *Netlist XILINX* (\*.xf), *netlist EDIF* (\*.edn, \*.edf) y de retardos (\*.sdf), como se muestra en la Figura D.4.

**Nota:** Los ficheros incluidos en la creación del proyecto se almacenan por defecto en la carpeta *In Design*, utilizada para guardar los ficheros fuentes en proceso de evaluación, que no han sido revisados o comprobados.

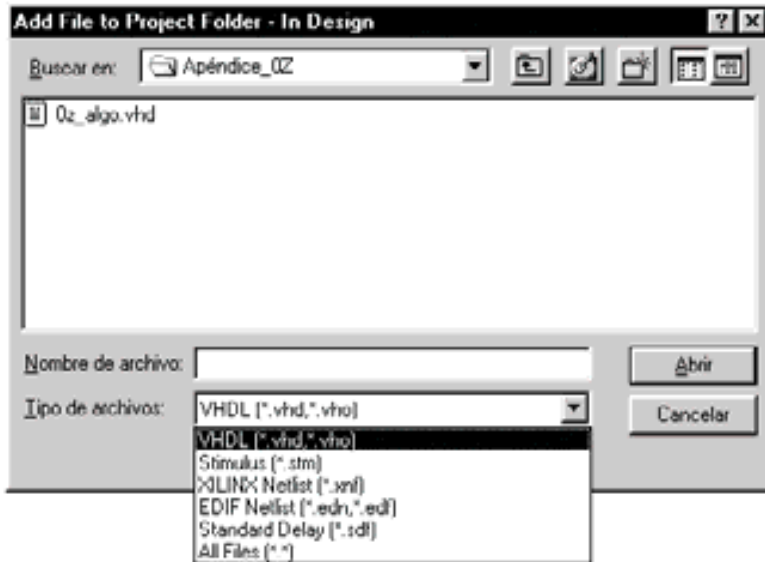


Figura D.4. Añadir documentos a un proyecto

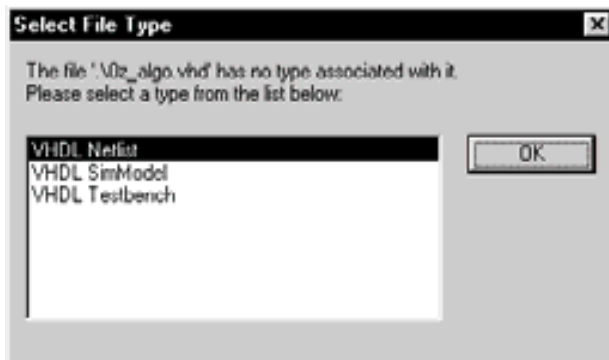


Figura D.5. Selección del tipo de fichero VHDL

- Si en los pasos anteriores se incluyeron en el proyecto ficheros previamente creados, el siguiente aviso de la Figura D.6 indica si se quiere compilar (registrando o cargando) dichos ficheros en el proyecto.

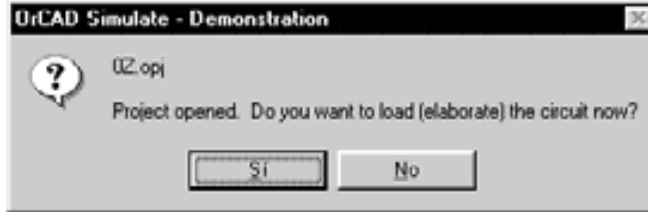


Figura D.6. Aviso solicitando si se quiere compilar los documentos añadidos al proyecto

- En la Figura D.7 se muestra el entorno del programa de simulación *OrCAD Simulate Demo* después de crear un nuevo proyecto y haber añadido un fichero fuente en lenguajes VHDL al proyecto, en la carpeta que se crea por defecto denominada *In Design*.

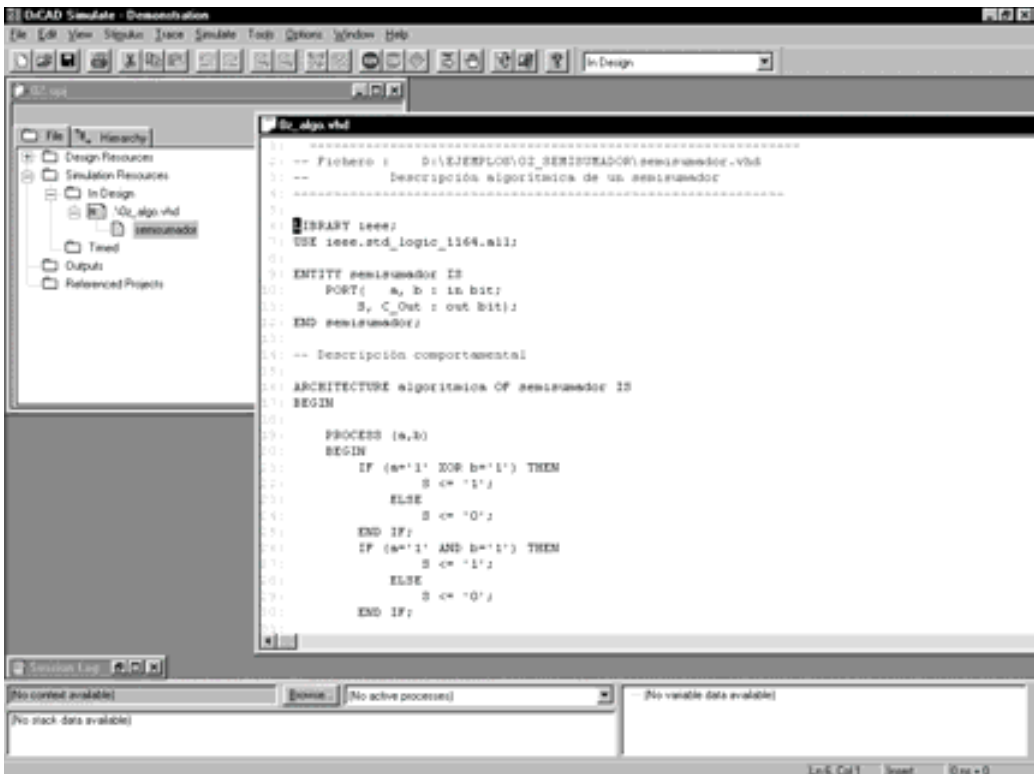
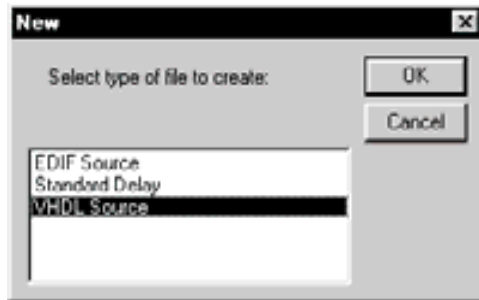


Figura D.7. Pantalla del programa del simulador, *Simulate* de *OrCAD Demo*, después de crear un proyecto y haberle añadido un fichero fuente VHDL

- Para crear en el proyecto nuevos ficheros fuente de tipo: VHDL, EDIF o Delay se utiliza el comando ***File/New***, o el icono



Mostrándose la pantalla de la Figura D.8, que permite elegir el tipo de fichero a crear en el proyecto, mediante el editor del simulador.



*Figura D.8. Selección del tipo de fichero nuevo a crear*

Ejemplo, si se elige la opción VHDL Source, se abre el editor de texto del simulador, como muestra la Figura D.9 donde se puede escribir el fichero fuente en VHDL. En el apartado D.2.1 Descripciones VHDL se indican los distintos tipos de descripciones VHDL.



*Figura D.9. Ventana del editor para crear un fichero VHDL*

- Para abrir e incluir ficheros en el proyecto, de los tipos indicados en la Figura D.10, se utiliza el comando ***File/Open***, o el icono



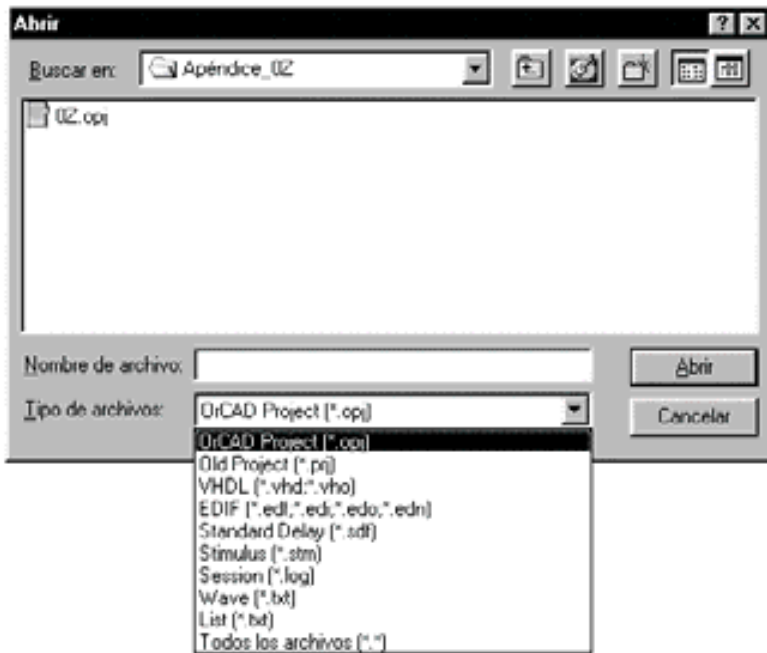


Figura D.10. Ventana de abrir ficheros para incluirlos en el proyecto

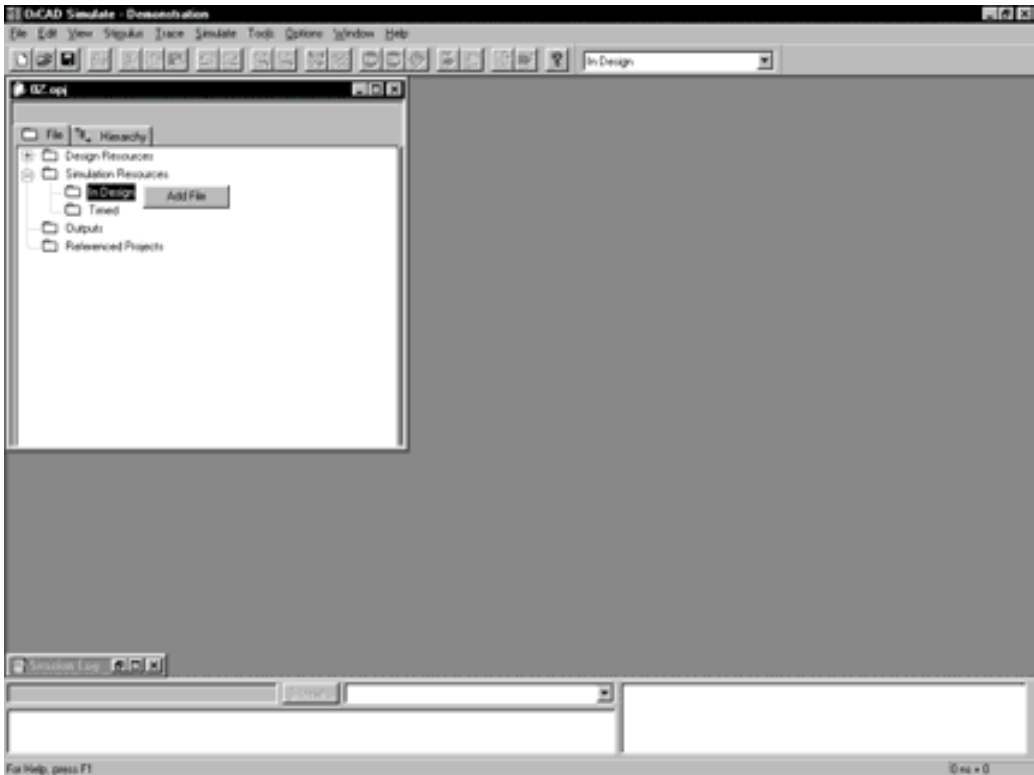
- **Otra forma de añadir ficheros en el proyecto** es mediante en menú contextual (pulsando el botón derecho del ratón) sobre un apartado del proyecto en el que se quiere añadir el fichero, apareciendo la opción **Add Files**, como se muestra en la Figura D.11.

**Nota:** Hay dos carpetas en las que se pueden incluir ficheros en el proyecto: **In Design** y **Timed**. Preferentemente, la carpeta **In Design** se usa para guardar los ficheros fuentes en proceso de evaluación y en la carpeta **Timed** para los ficheros con versiones definitivas por haber sido comprobados. Posteriormente en el proceso de simulación habrá que elegir dónde se encuentran los ficheros a simular: en la carpeta **In Design** o en **Timed**.

- **IMPORTANTE:** Todos los nuevos ficheros que se añaden en el proyecto, para que tengan efecto, deben ser **compilados** con el comando **Simulate/Reload Project**, o el icono



Una vez ejecutado el comando **Simulate/Reload Project** se puede observar que en el programa del simulador se habilitan los iconos que permiten generar los estímulos y la simulación.



*Figura D.11. Menú contextual sobre la carpeta In Design para añadir ficheros al proyecto*

## D.2.1 Descripciones VHDL

El lenguaje VHDL presenta tres estilos diferentes de descripción de sistemas dependiendo del nivel de abstracción. A continuación, se desarrollan los tres estilos de descripción de un circuito semisumador, con una explicación de las características principales de cada uno de ellos.

- 1) **Descripción comportamental algorítmica.** Es la más cercana a un lenguaje convencional. Está formada por el bloque PROCESS, similar a una subrutina, cuyas instrucciones se ejecutan secuencialmente. En esta descripción no se indican ni componentes, ni interconexiones, sólo su comportamiento o funcionamiento. En la Figura D.12 se muestra un ejemplo de aplicación de este tipo de descripción.

La ruta y el nombre del fichero que contiene esta descripción VHDL es la que se indica a continuación:

## D:\Ejemplos\Apend\_D\0D\_algo\0D\_algo.vhd

```

01: -----
02: -- Fichero :      D:\EJEMPLOS\Apendice_02\02_algo\02_algo.vhd
03: --              Descripción algorítmica de un semisumador
04: -----
05:
06: LIBRARY ieee;
07: USE ieee.std_logic_1164.all;
08:
09: ENTITY semisumador IS
10:     PORT(   a, b : in bit;
11:           S, C_Out : out bit);
12: END semisumador;
13:
14: -- Descripción comportamental
15:
16: ARCHITECTURE algoritmica OF semisumador IS
17: BEGIN
18:
19:     PROCESS (a,b)
20:     BEGIN
21:         IF ((a=1 AND b=0) OR (a=0 AND b=1)) THEN
22:             S <= '1';
23:         ELSE
24:             S <= '0';
25:         END IF;
26:         IF (a='1' AND b='1') THEN
27:             C_out <= '1';
28:         ELSE
29:             C_out <= '0';
30:         END IF;
31:     END PROCESS;
32:
33: END algoritmica;
34:
35:

```

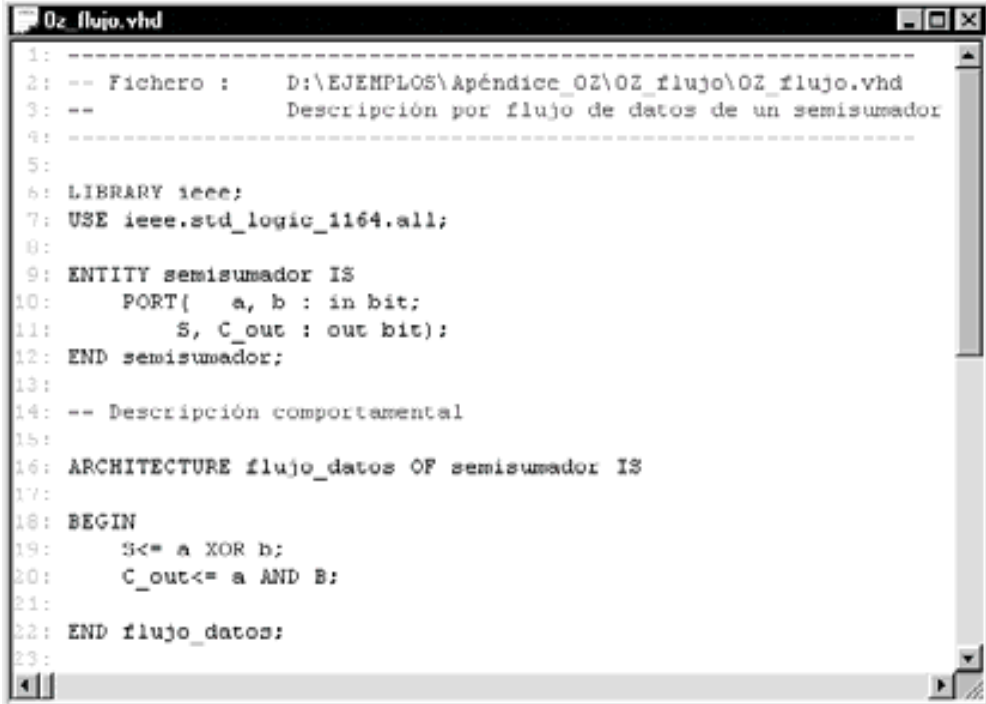
Figura D.12. Descripción comportamental algorítmica de un circuito semisumador mediante VHDL

- 2) **Descripción comportamental por flujo de datos** o también llamada de **transferencia entre registros (RTL)**. Es más cercana a la realización física o estructural aunque sin llegar a serlo ya que si bien describe componentes y asigna señales, no constituye una lista de componentes e interconexiones. Permite la paralelización de instrucciones siendo una descripción concurrente. En la Figura D.13 se muestra un ejemplo de este tipo de aplicación.



La ruta y el nombre del fichero que contiene esta descripción VHDL es la que se indica a continuación:

**D:\Ejemplos\Apend\_D\0D\_flujo\0D\_flujo.vhd**



```

1: -----
2: -- Fichero :    D:\EJEMPLOS\Apéndice_OZ\OZ_flujo\OZ_flujo.vhd
3: --           Descripción por flujo de datos de un semisumador
4: -----
5:
6: LIBRARY ieee;
7: USE ieee.std_logic_1164.all;
8:
9: ENTITY semisumador IS
10:     PORT(    a, b : in bit;
11:           S, C_out : out bit);
12: END semisumador;
13:
14: -- Descripción comportamental
15:
16: ARCHITECTURE flujo_datos OF semisumador IS
17:
18: BEGIN
19:     S<= a XOR b;
20:     C_out<= a AND b;
21:
22: END flujo_datos;
23:

```

*Figura D.13. Descripción comportamental por flujo de datos de un semisumador mediante VHDL*

- 3) **Descripción estructural.** Es la que más se acerca a la realización física de un circuito y puede considerarse como una lista de componentes e interconexiones (*Netlist*). La descripción es mucho más larga y menos clara que las anteriormente indicadas. Los componentes son entidades definidas en bibliotecas y para las conexiones se utilizan señales internas definidas al principio. A cada componente se le asigna un símbolo, hecho este que se denomina **replicación**. En la Figura D.14 se muestra un ejemplo de este tipo de descripción.

La ruta y el nombre del fichero que contiene esta descripción VHDL es la que se indica a continuación:

**D:\Ejemplos\Apend\_D\0D\_estr\0D\_estr.vhd**

```

-----
-- Fichero : D:\EJEMPLOS\Apéndice_0Z\0Z_estr\0Z_estr.vhd
-- Descripción estructural de un semisumador
-----

LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Definición de entidades y arquitectura de cada componente

ENTITY componente_xor IS
    PORT( ap, bp : in bit;
          Sp : out bit);
END componente_xor;

ENTITY componente_and IS
    PORT( aq, bq : in bit;
          C_outq : out bit);
END componente_and;

ARCHITECTURE rtl OF componente_xor IS
BEGIN
    Sp <= ap XOR bp;
END rtl;

ARCHITECTURE rtl OF componente_and IS
BEGIN
    C_outq <= aq AND bq;
END rtl;

-- Definición de la entidad y arquitectura del conjunto o circuito (netlist)

ENTITY semisumador IS
    PORT (a,b: IN bit;
          S, C_out: OUT bit);
END semisumador;

ARCHITECTURE estructural OF semisumador IS

    COMPONENT componente_xor IS
        PORT (ap,bp: IN bit; Sp: OUT bit);
    END COMPONENT componente_xor;

    COMPONENT componente_and IS
        PORT (aq,bq: IN bit; C_outq: OUT bit);
    END COMPONENT componente_and;

BEGIN
    u0: componente_xor PORT MAP (ap=>a,bp=>b,Sp=>S);
    u1: componente_and PORT MAP (aq=>a,bq=>b,C_outq=>C_out);

END estructural;

```

*Figura D.14. Descripción estructural de un semisumador mediante VHDL*

## D.3 GENERACIÓN DE ESTÍMULOS

Para definir los estímulos del circuito, el programa de simulación *OrCAD Demo v9*, permite **dos procedimientos**:

- mediante la descripción de los estímulos en un **fichero VHDL** que se añade al proyecto, o,
- mediante una **herramienta interactiva** de generación de estímulos propia del simulador.

### D.3.1 Generación de estímulos mediante fichero VHDL

Mediante un editor de texto se puede generar la definición de los estímulos de un circuito en lenguaje VHDL, como se muestra en la Figura D.15, en el que se han definido a modo de ejemplo los estímulos del semisumador. Este fichero se debe añadir al proyecto pulsando el botón derecho del ratón sobre el apartado del proyecto en el que se quiere añadir el fichero, apareciendo un menú contextual con la opción *Add Files*, como se mostró en la anterior Figura D.11. En la Figura D.16 se muestra el fichero de estímulos añadido al proyecto llamado *estímulos*, el cual posteriormente debe ser **compilado** con el comando *Simulate/Reload Project*, o el icono



### D.3.2 Generación de estímulos mediante la herramienta interactiva del simulador

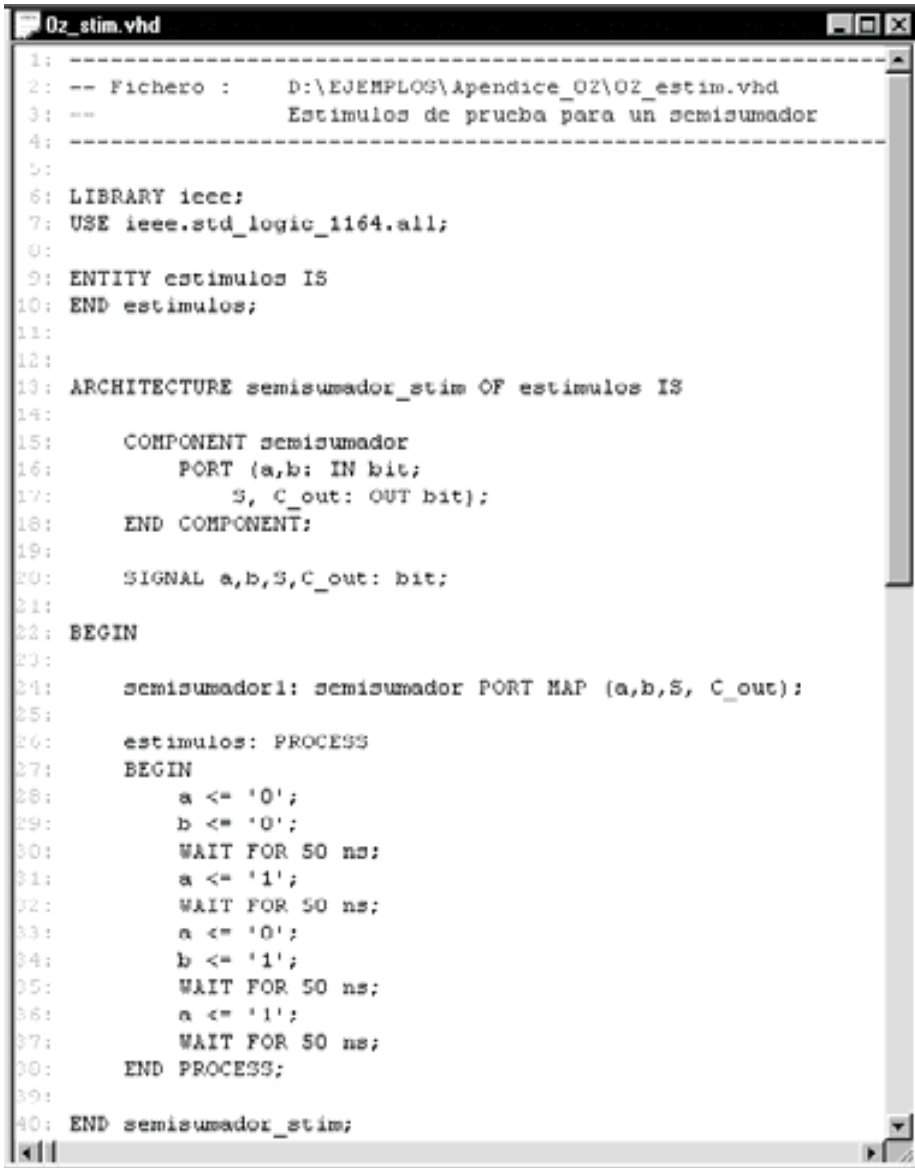
El comando *Stimulos/New Interactive...* o el icono,



abren la herramienta interactiva del simulador que genera los estímulos de un circuito, permitiendo definirlos de tres formas distintas:

- **Basic**: para generar impulsos. En el ejemplo de la Figura D.17 se define en la señal de entrada *a* un impulso de nivel “H” entre 50 y 100 ns, cuyo cronograma se muestra en Figura D.18.
- **Advanced**: para generar una serie de impulsos que se pueden repetir un número finito de veces. En el ejemplo de la Figura D.19. se define en la señal de entrada *b* dos impulsos de nivel “H” entre 25 - 50 ns, 90 - 100, que se repiten tres veces. En la Figura D.20 se muestra el cronograma de dicha señal.

- **Clock:** para generar relojes. En el ejemplo de la Figura D.21 se define en la señal de entrada *a* un reloj de periodo 100 ns y en la señal de entrada *b* un reloj de periodo 200 ns, cuyos cronogramas se muestran en la Figura D.20.



```
1: -----
2: -- Fichero :    D:\EJEMPLOS\apendice_OZ\OZ_estim.vhd
3: --            Estimulos de prueba para un semisumador
4: -----
5:
6: LIBRARY ieee;
7: USE ieee.std_logic_1164.all;
8:
9: ENTITY estimulos IS
10: END estimulos;
11:
12:
13: ARCHITECTURE semisumador_stim OF estimulos IS
14:
15:     COMPONENT semisumador
16:     PORT (a,b: IN bit;
17:          S, C_out: OUT bit);
18:     END COMPONENT;
19:
20:     SIGNAL a,b,S,C_out: bit;
21:
22: BEGIN
23:
24:     semisumador1: semisumador PORT MAP (a,b,S, C_out);
25:
26:     estimulos: PROCESS
27:     BEGIN
28:         a <= '0';
29:         b <= '0';
30:         WAIT FOR 50 ns;
31:         a <= '1';
32:         WAIT FOR 50 ns;
33:         a <= '0';
34:         b <= '1';
35:         WAIT FOR 50 ns;
36:         a <= '1';
37:         WAIT FOR 50 ns;
38:     END PROCESS;
39:
40: END semisumador_stim;
```

Figura D.15. Fichero de estímulos para la simulación de un semisumador, mediante lenguaje VHDL



Figura D.16. Fichero de estímulos añadido al proyecto llamado estímulos



Figura D.17. Herramienta de generación de estímulos de tipo Basic

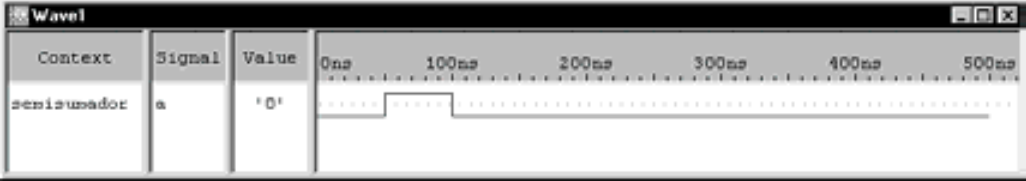


Figura D.18. Cronograma de la señal de entrada a definida de tipo Basic

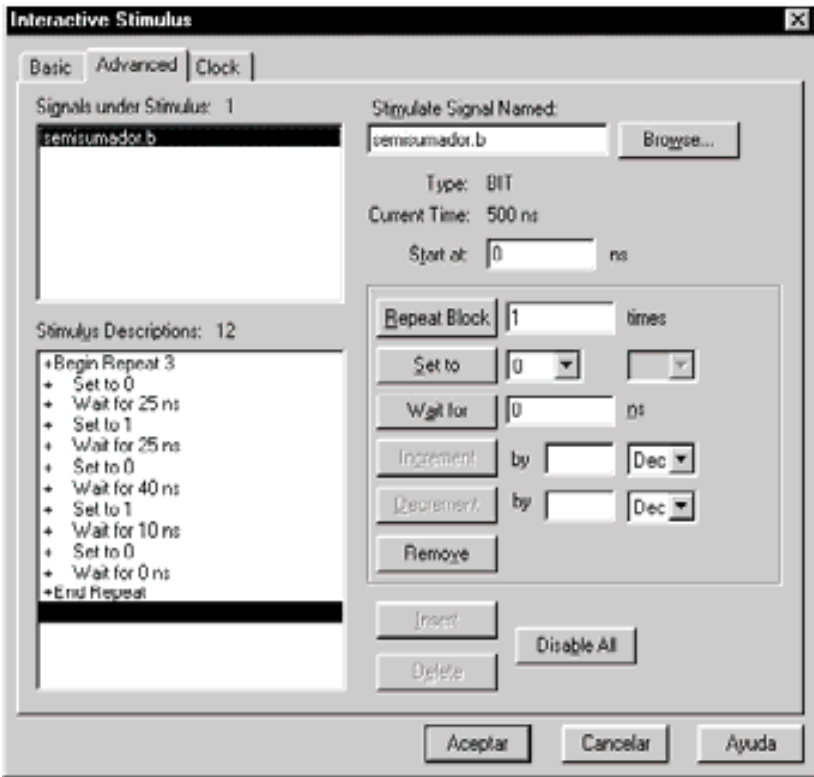


Figura D.19. Herramienta de generación de estímulos de tipo Advanced



Figura D.20. Cronograma de la señal de entrada b definida de tipo Advanced

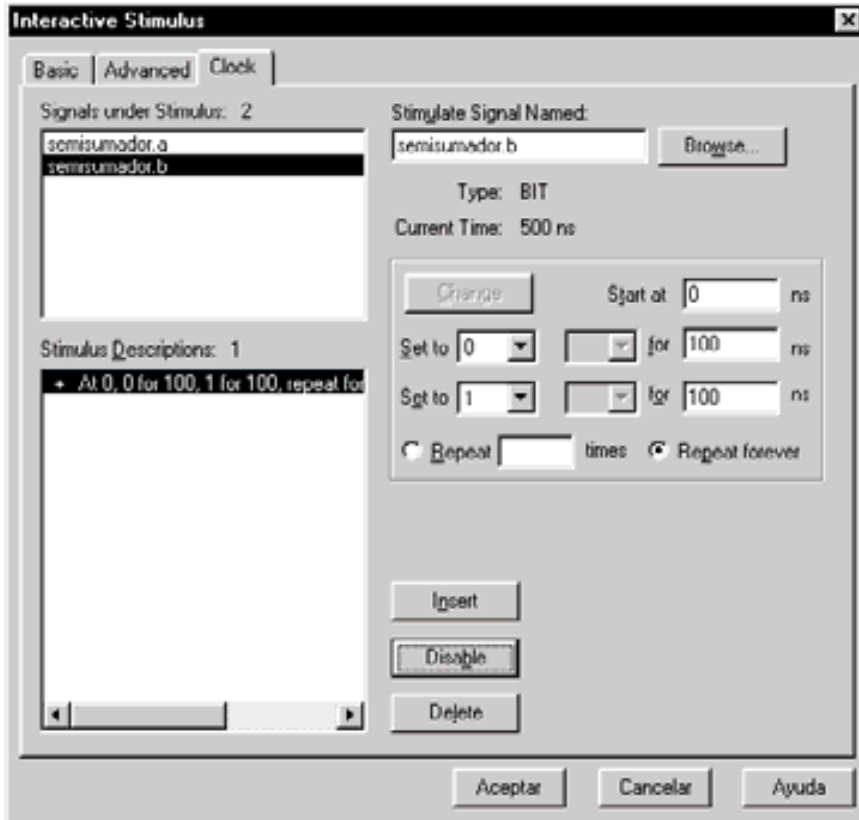


Figura D.21. Herramienta de generación de estímulos de tipo Clock

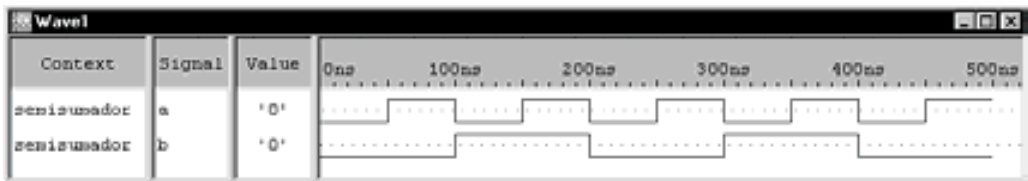


Figura D.22. Cronograma de la señal de entrada b definida de tipo Clock

En la Figura D.23 se muestran las señales de estímulos que han sido definidas mediante la herramienta interactiva del simulador y de qué tipo son (*Basic*, *Advanced* o *Clock*). En el caso de definir una misma señal con varios tipos (*Basic*, *Advanced* o *Clock*) de estímulos, como ocurre en dicha figura, el estímulo resultante será la suma lógica de todos ellos. Se pueden **habilitar o inhabilitar cada uno de los estímulos** o partes de ellos con los botones *Enable* y *Disable* respectivamente, en la ventana *Interactive Stimulus*. Cada elemento del estímulo habilitado tiene un signo + a su

izquierda, mientras que si está inhabilitado el signo es -. Ver, por ejemplo, en la Figura D.21 el botón *Disable* y que la definición del estímulo *semisumador.b* se encuentra habilitada al figurar en el apartado *Stimulus Descriptions* +At 0, 0 for 100,...



Figura D.23. Información de las señales de estímulos definidas mediante la herramienta interactiva del simulador y su tipo

## D.4 SIMULACIÓN

La simulación, que muestra el comportamiento del circuito, puede obtenerse mediante dos tipos de representación diferentes:

- *Wave*: Cronograma.
- *List*: Tabla de verdad.

El comando Trace/Edit Signal Traces... o el icono,



permiten seleccionar el tipo de representación, las señales a representar y su orden o posición, mediante las ventanas de la Figura D.24 y de la Figura D.25.

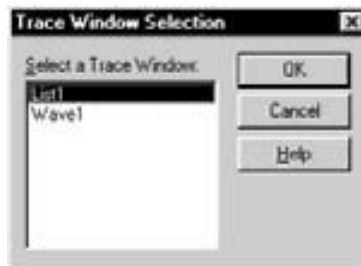


Figura D.24. Selección de las señales a representar según el tipo de representación (list o Wave)



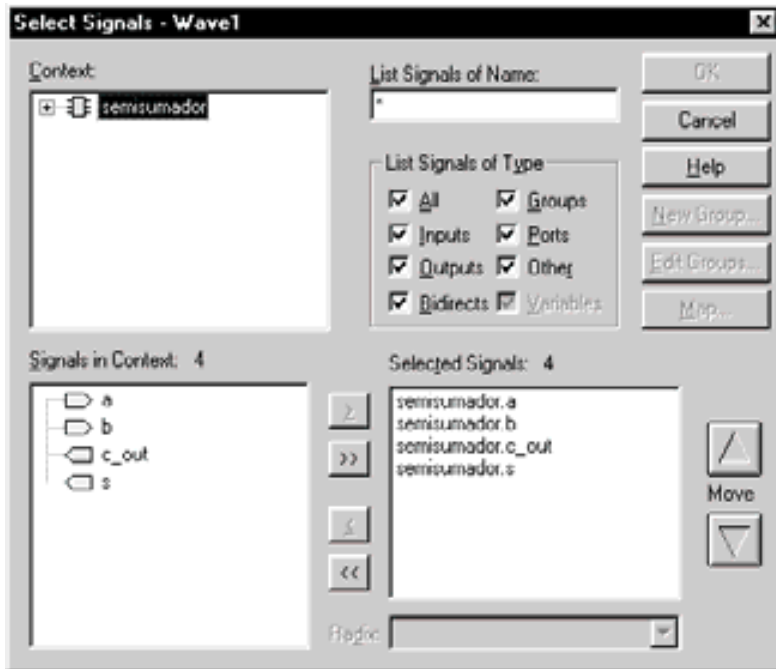


Figura D.25. Selección de las señales a representar y su posición

Para **activar la simulación** se debe:

- Prefijar las opciones del proyecto, siendo una de ellas la duración de la simulación. El comando **Option/Project**, abre la ventana de la Figura D.26, en la que se puede establecer dicha duración en la casilla **Run Duration**. Si se activan las opciones **Wave Window** y **List Window** el simulador presenta los resultados, del cronograma y tabla de verdad, en dos ventanas.
- Elegir la ubicación de los ficheros a simular, carpeta *In Design* o *Timed* mediante el icono,



**Nota:** En el ejemplo del semisumador los ficheros VHDL y de estímulos están almacenados en la carpeta *In Design*.

- Asegurarse de que todos los ficheros del proyecto están compilados con el comando **Simulate/Reload Project**, o el icono



- Inicializar o resetear la simulación con el comando ***Simulate/Restart***, o el icono,



- Por último poner en marcha la simulación con el comando ***Simulate/Run...***, o el icono,

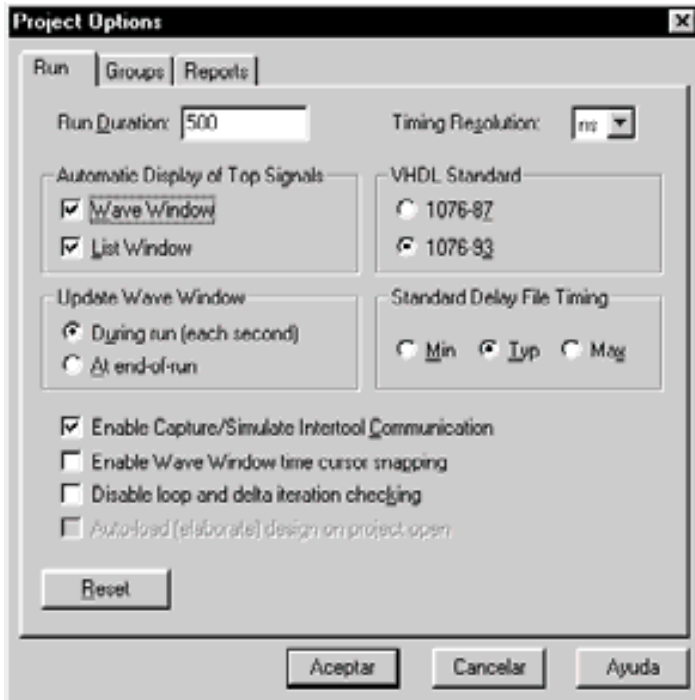


Figura D.26. Opciones del proyecto

En la Figura D.27 se muestra el comportamiento del semisumador. En la ventana ***Wave1*** se representa su cronograma y en la ventana ***List1*** su tabla de verdad. También se disponen de ventanas informativas como ***Sesion Log*** que indica los mensajes y errores en el proceso de simulación y ***Stimulus1*** que informa de las señales de estímulos y su tipo (en este ejemplo se utilizan relojes para las señales del semisumador *a* y *b*).

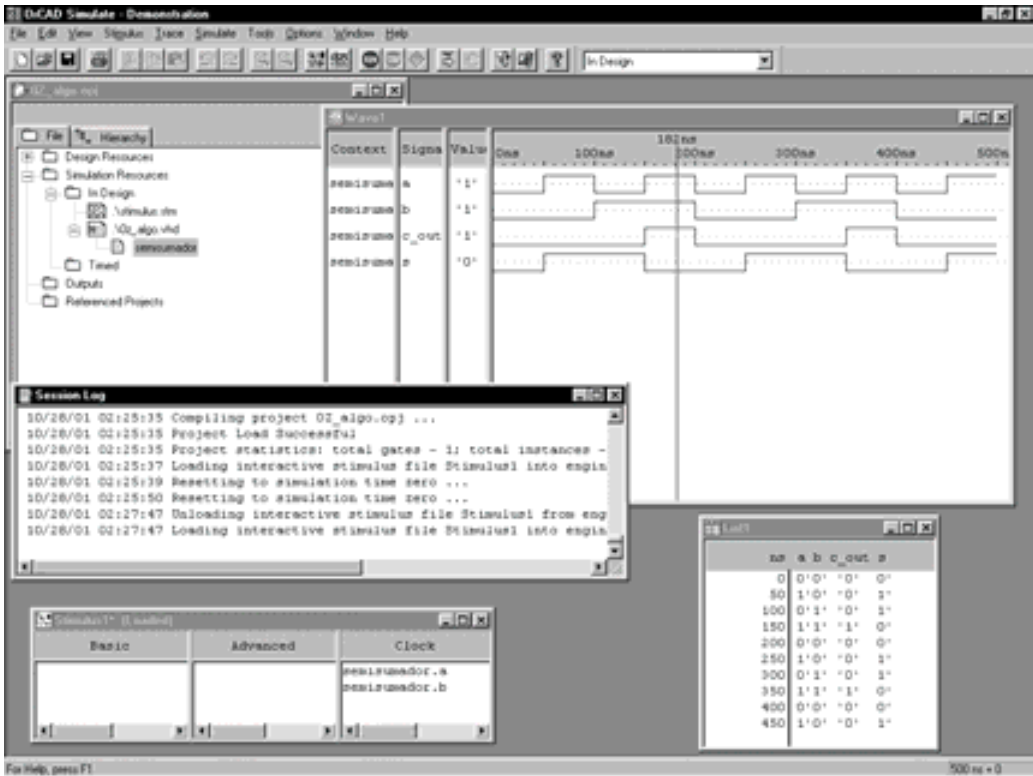


Figura D.27. Resultado de la simulación del semisumador

En el cronograma de la ventana **Wave1** se puede modificar la escala temporal de visualización (*zoom* en eje *x*), mediante los comandos **View/Zoom In**, **View/Zoom Out**, o los iconos,



## D.5 DEBUGGER

Como en otros lenguajes de programación, VHDL también permite herramientas de depuración como: ejecución paso a paso, puntos de ruptura, monitorización de señales, etc.

Mediante el comando **Simulate/Step**, o el icono,



se puede realizar una ejecución paso a paso señalando una flecha amarilla la instrucción a ejecutar en el listado del programa fuente en VHDL como se muestra en la Figura D.28. En esta misma Figura D.28 se aprecian dos puntos rojos que son puntos de ruptura que se crean con el comando *Simulate/Break on expr...*, *Simulate/Break on line...* o el icono,



previa colocación del cursor en la línea o expresiones en las que interese que la ejecución se pare, cuando pase por ellas.

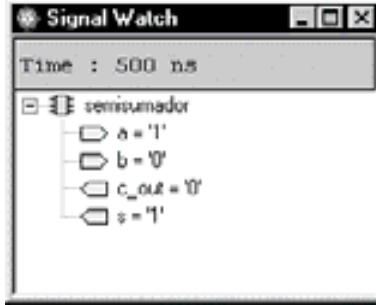
```

1: -----
2: -- Archivo :    DD:\EJEMPLOS\apendice_02\02_algo.vhd
3: --            Descripción algorítmica de un semisumador
4: -----
5:
6: LIBRARY ieee;
7: USE ieee.std_logic_1164.all;
8:
9: ENTITY semisumador IS
10:    PORT(  a, b : in bit;
11:          S, C_Out : out bit);
12: END semisumador;
13:
14: -- Descripción comportamental
15:
16: ARCHITECTURE algoritmica OF semisumador IS
17: BEGIN
18:
19:    PROCESS (a,b)
20:    BEGIN
21:        IF ((a=1 AND b=0) OR (a=0 AND b=1)) THEN
22:            S <= '1';
23:        ELSE
24:            S <= '0';
25:        END IF;
26:        IF (a='1' AND b='1') THEN
27:            C_out <= '1';
28:        ELSE
29:            C_out <= '0';
30:        END IF;
31:
32:    END PROCESS;
33:
34: END algoritmica;

```

Figura D.28. Ejecución paso a paso y puntos de ruptura en la simulación de un semisumador

Otras posibilidades del simulador son la monitorización de señales mediante el comando ***Trace/Watch Window***, que tras seleccionar las señales a visualizar, como en la Figura D.25 se obtiene su valor, como muestra la Figura D.29.



*Figura D.29. Monitorización de señales con el comando Trace/Watch Window*

Se incluyen tres proyectos del semisumador con sus diferentes tipos de descripción VHDL, que pueden ser abiertos desde el simulador *OrCAD Demo Simulate*.

Las rutas y los nombres de los ficheros que contienen los proyectos son los que se indican a continuación:

**D:\Ejemplos\Apend\_D\0D\_algo\0D\_algo.opj**

**D:\Ejemplos\Apend\_D\0D\_flujo\0D\_flujo.opj**

**D:\Ejemplos\Apend\_D\0D\_estr\0D\_estr.opj**

## **GUÍA DE *ELECTRONICS WORKBENCH 5.0***

---

---

**Objetivos:**

- Conocer las características de las distintas ediciones de la versión 5.0 de la herramienta *Electronics Workbench*
- Conocer cómo se lleva a cabo la instalación de la aplicación *Electronics Workbench 5.0*.

**Contenido:** Se describen las características de la herramienta de simulación *Electronics Workbench 5.0*, así como el procedimiento de instalación del programa de simulación de circuitos electrónicos, denominado *Electronics Workbench 5.0* versión estudiante “*Student Edition*”, indicando las distintas ediciones existentes (profesional, personal, educacional y estudiante) y sus limitaciones. Además, se hace referencia a la nueva aplicación multiSIM de EWW.

**Simulación:** Este capítulo no tiene ejercicios de simulación por su carácter teórico.

## E.1 INTRODUCCIÓN

*Electronics Workbench* es un **laboratorio virtual de electrónica** integrado en el ordenador. Permite construir fácilmente esquemas de circuitos analógicos, digitales o mixtos, a los que se les conecta instrumentos de prueba virtuales para determinar su comportamiento, sin tener que usar bancos de trabajo ni componentes o instrumentos reales.

Esta herramienta es capaz de simular el comportamiento de los circuitos de forma real, visualizando continuamente los resultados obtenidos en la pantalla del polímetro, osciloscopio, trazador de Bode, analizador lógico, mediante pilotos indicadores, etc. Además es integrado e **interactivo**, permite modificar el circuito en cualquier instante, lo que posibilita un análisis rápido y repetitivo de posibles situaciones, como por ejemplo, pueden abrirse interruptores o ajustarse potenciómetros mientras se simula el circuito. Las formas de onda están disponibles en todo momento durante la simulación, y no únicamente al terminar ésta o tras realizar una nueva compilación del esquema modificado.

*Electronics Workbench* tiene **simulación mixta** (analógica y digital) y un análisis gráfico de la forma de onda, que permite diseñar circuitos para analizarlos posteriormente con distintos instrumentos virtuales de medida y diferentes opciones de análisis.

La edición de componentes y su interconexión en la creación de esquemas es rápida y sencilla, mediante *clic* y arrastre del ratón, al estar basado en el entorno gráfico de *Microsoft Windows*, funcionando bajo *Windows-3.1/3.11/95/98/NT*.

## E.2 REQUISITOS DEL SISTEMA

### E.2.1 Windows 95

Requiere *Microsoft Windows 95* con ratón compatible *Microsoft*, CPU 486 o superior, con 8 MB de RAM (recomendables 16 MB) y 20 MB de espacio libre en el disco duro.

### E.2.2 Windows NT

Requiere *Microsoft Windows NT* con ratón compatible *Microsoft*, CPU 486 o superior, con 12 MB de RAM (recomendables 16 MB) y 20 MB de espacio libre en el disco duro.

## E.2.3 Windows 3.1

Requiere *Microsoft Windows 3.1* ó *3.11* con ratón compatible *Microsoft*, CPU 486 o superior, con 8 MB de RAM (recomendables 16 MB) y 20 MB de espacio libre en el disco duro.

## E.3 CARACTERÍSTICAS DE ELECTRONICS WORKBENCH VERSIÓN 5 “PROFESSIONAL EDITION”

Las características que se enumeran en este Apartado se refieren a la edición profesional de la versión 5 “*Professional Edition*”. Otras ediciones de la versión 5 presentan las limitaciones que se relacionan en el Apartado E.4.

### E.3.1 Generales

- 1) **Herramientas integradas:** Editor de esquemas, simulación *SPICE*, generación de formas de onda y análisis. Permite modificar el circuito durante la simulación. Realiza análisis de circuitos mediante instrumentos de prueba virtuales, o los catorce tipos de análisis enumerados en el Apartado E.3.2.
- 2) **Herramientas integradas:** Editor de esquemas, simulación *SPICE*, generación de formas de onda y análisis. Permite modificar el circuito durante la simulación. Realiza análisis de circuitos mediante instrumentos de prueba virtuales, o los catorce tipos de análisis enumerados en el Apartado E.3.2.
- 3) **Dispositivo de simulación:** *SPICE 3FS* de 32 bits interactivo, ampliado con soporte modo nativo e híbrido analógico/digital. Inserción automática de interfaz de traducción de señal. Soporte reutilizable de bloques jerarquizados. Escalamiento *GMIN* para mejorar la convergencia. Sin límites preestablecidos de dimensiones y complejidad de circuitos.
- 4) **Captura de esquemas:** Mediante ratón. Zona de trabajo jerarquizado. Cableado automático con ajuste manual. Designación automática de referencia. Sin límites preestablecidos de dimensiones de esquemas.
- 5) **Análisis:** Instrumentos de prueba virtuales de análisis. Catorce análisis gráficos en pantalla.
- 6) **Diseño encapsulado:** Toda la información del diseño, como: configuración del circuito, parámetros *SPICE*, instalación y copias de todos los modelos; se almacenan en un archivo para facilitar la reutilización y uso compartido del diseño por otras herramientas.
- 7) **Interoperatividad:** Importación y exportación de archivos de lista de red *SPICE* estándar para comunicarse con otros simuladores o reutilizar los



elementos de diseño existentes. Importación de modelos de fabricantes y componentes reutilizables de *Electronics Workbench*. Exportación a los principales paquetes de diseño de Circuitos Impresos (*ORCAD*, *Protel* y *Tango*).

### E.3.2 Análisis

- 1) **Punto de funcionamiento en continua:** Calcula el punto de funcionamiento en continua, obteniéndose información de la tensión presente en cada nodo del circuito.
- 2) **Transitorio:** Determina las corrientes y tensiones en el circuito en función del tiempo en cualquiera de los nodos. Se debe especificar el tiempo inicial y final.
- 3) **Respuesta en frecuencia:** Calcula la ganancia y fase para pequeña señal en régimen estacionario y en el dominio de la frecuencia, en cualquiera de los nodos del circuito. Se debe especificar el margen, tipo (década, octava o lineal) y resolución (número de puntos) de la frecuencia de barrido.
- 4) **Fourier:** Determina el módulo y fase de componentes espectrales de *Fourier* y su valor medio o componente continua. Se debe especificar la frecuencia fundamental y el número de armónicos.
- 5) **Ruido:** Determina la contribución del ruido generado por los semiconductores y resistencias, expresando su valor como suma eficaz de sus componentes. Se debe especificar el dispositivo de interés, nodo de salida, fuente de referencia, margen, tipo y resolución del barrido de frecuencia.
- 6) **Distorsión:** Calcula los productos de intermodulación y armónicos en régimen estacionario para pequeñas señales en un margen de frecuencias. Se debe especificar el nodo, margen, tipo y resolución del barrido de frecuencia.
- 7) **Respuesta paramétrica:** Determina el comportamiento transitorio, en un nodo del circuito, cuando varía un parámetro de uno de los dispositivos del circuito. Se debe especificar el nodo, el margen, tipo y resolución de la variación del parámetro.
- 8) **Respuesta térmica:** Determina el comportamiento transitorio, en un nodo del circuito, cuando varía la temperatura. Se debe especificar el nodo, el margen, tipo y resolución de la variación de temperatura.
- 9) **Polo-cero:** Calcula polos y ceros de la función de transferencia. Se debe especificar los nodos de entrada y salida.
- 10) **Función de transferencia:** Determina la relación salida/entrada, la impedancia entre los terminales de entrada y la impedancia entre los terminales de salida del circuito, proporcionando el circuito equivalente para

pequeña señal en continua entre los puntos definidos como entrada y salida en el circuito. Se deben especificar los nodos de entrada y salida.

- 11) **Sensibilidad de continua:** Determina la sensibilidad o dependencia del punto de trabajo de un dispositivo con respecto a los parámetros del circuito. Se debe especificar dispositivo y parámetros de interés.
- 12) **Sensibilidad de alterna:** Determina la sensibilidad o dependencia de los incrementos de voltaje o corriente de una variable de salida con respecto a los incrementos de un parámetro del circuito. Se debe especificar la variable de salida y el parámetro de interés.
- 13) **Peor caso:** Calcula la máxima desviación que sufre el punto de trabajo en continua, la variación de la respuesta en alterna o la variación de la respuesta transitoria de una variable de salida, cuando los parámetros de los dispositivos del circuito son variados dentro de su tolerancia. Se debe especificar la consideración de peor caso (ejemplo el valor máximo o mínimo).
- 14) **Monte Carlo:** Calcula la desviación que sufre el punto de trabajo en continua, la variación de la respuesta en alterna o la variación de la respuesta transitoria de una variable de salida, cuando los parámetros de los dispositivos del circuito son variados al azar dentro de su tolerancia. Se debe especificar el nodo de salida, el número de veces que se realiza el análisis (número de ejecuciones), la tolerancia, la semilla, el tipo de distribución uniforme o de *Gauss* y el tipo de respuesta a analizar: continua, transitoria o de frecuencia.

### E.3.3 Instrumentos de prueba virtual

- 1) **Multímetro Digital:** Autoescalable. Mide corriente y tensión tanto en continua como en alterna, medidas de resistencia y pérdidas en decibelios.
- 2) **Generador de Funciones:** Genera señales senoidales, triangulares y de onda cuadrada, desde 1 Hz a 999 MHz, con ciclo de trabajo, amplitud y *offset* ajustables.
- 3) **Osciloscopio:** Doble traza. Margen de base de tiempos desde nanosegundos a segundos. Disparo interno y externo; con flanco positivo y negativo. Despliegue temporal de dos cursores digitales. Almacenamiento de datos en archivos *ASCII*.
- 4) **Trazador de Bode:** Dibuja módulo y fase para un barrido de frecuencia. Soporta frecuencias desde MHz a GHz. Permite ejes en las gráficas con escala logarítmica y lineal.
- 5) **Generador de Palabra:** Actúa como editor de estímulos digitales para excitar al circuito, mediante 32 K de palabras de 16 bits. Presenta y edita datos en *ASCII*, binario y hexadecimales. Carga, guarda, corta y pega palabras. Excita

al circuito con series de datos de una palabra, un ciclo de palabras o modo continuo. Disparo externo con condiciones o patrón de inicio de adquisición.

- 6) **Analizador Lógico:** Soporta predisparo y posdisparo. Reloj interno o externo, mediante flanco negativo o positivo. Salida de reloj para sincronizar datos. Utilización de patrones de disparo, para comenzar la adquisición de datos.
- 7) **Convertidor Lógico:** Convierte representaciones de circuitos lógicos en su expresión algebraica *Booleana* o en tabla de verdad del circuito y viceversa.

### E.3.4 Componentes

- 1) **Fuentes:** De tensión continua o alterna, de corriente continua o alterna, de tensión controlada por tensión o corriente, de corriente controlada por tensión o corriente, AM, FM, Reloj, de pulsos modulados en anchura (PWM), Oscilador controlado por tensión (VCO), polinomiales y dependientes no lineales.
- 2) **Pasivos:** Resistencia, capacidad, inductancia, transformador, relé, conmutador, conmutador con retardo, conmutador controlado por tensión, resistencia variable, inductancia variable, inductancias acopladas y transformadores no lineales.
- 3) **Diodos:** Diodo rectificador, diodo zener, diodo emisor de luz (LED), diodo schotkky, diac, tiristor (SCR), triac, puente rectificador.
- 4) **Transistores:** De unión (BJT) de tipo NPN o PNP; de efecto campo de canal N o P (JFET); de puerta aislada (MOSFET), de enriquecimiento o empobrecimiento, de 3 o 4 terminales, de canal N o P.
- 5) **Circuitos integrados analógicos:** Amplificador operacional de 3 o 5 terminales, comparador y regulador de tensión.
- 6) **Circuitos integrados mixtos:** Convertidor analógico-digital (A/D), convertidor digital-analógico (D/A), convertidor de tensión-corriente, convertidor de corriente-tensión, temporizador 555 y monoestable.
- 7) **Puertas lógicas:** AND, OR, NOT, NAND, NOR, ORX, NORX, *buffer*, *buffer* triestado y disparador *Schmitt*.
- 8) **Digitales:** Básculas RS, JK, y D; sumadores completos; multiplexores, demultiplexores, codificadores y decodificadores.
- 9) **Indicadores:** Lámparas, pilotos de señalización, voltímetros, amperímetros, *display* de 7 segmentos, *display* de barras y zumbadores.
- 10) **Controles:** Diferenciador, integrador, bloque de ganancia, función de transferencia, limitador, sumador, multiplicador y divisor.

- 11) **Otros:** Fusibles, líneas de transmisión con y sin pérdidas, cristales, motores de continua, válvulas de vacío y convertidores.
- 12) **Circuitos Integrados:** series 74xx, 74xxx y 4xxx.

### E.3.5 Modelos

- 1) **Digitales:** De circuitos integrados de puertas y básculas en HC, *buffers* HC, *Drenador* abierto HC, LS, *buffers* LS y colector abierto LS.
- 2) **Diodos:** Más de 1300 modelos de diodos, diodos zener, diodos emisores de luz, diodos *schotkky* y diacs de Motorola, General Instruments, International Rectifier, Zetex y Phillips.
- 3) **Transistores:** Más de 1400 modelos de unión (BJT), JFET, MOSFET, tiristores, triacs y IGBT de Motorola, National Semiconductor, International Rectifier, Toshiba, Harris y Phillips.
- 4) **Integrados analógicos:** Más de 1200 modelos de amplificadores operacionales, comparadores y rectificadores de Motorola, Texas Instruments, Maxim, Elantec, Analog Devices, Zetex, Burr Brown y Linear Technology.
- 5) **Otros modelos:** Relés, transformadores, válvulas de vacío, líneas de transmisión y cristales.

## E.4 EDICIONES DE LA VERSIÓN 5 Y SUS LIMITACIONES

### E.4.1 Edición profesional “PROFESSIONAL EDITION”

Sin limitaciones, salvo las que imponga las características del equipo ordenador en el que está instalado (capacidad de memoria de almacenamiento, velocidad, etc.). Las características son las indicadas en el Apartado E.3.

### E.4.2 Edición personal “PERSONAL EDITION”

El número de análisis en esta edición está limitado a los seis primeros del Apartado E.3.2.

El número de modelos está limitado: en diodos a 850, en transistores a 400 y en circuitos integrados analógicos a 650 modelos.

### E.4.3 Versión educación “EDUCATIONAL EDITION”

El número de modelos está limitado: en diodos a 850, en transistores a 400 y en circuitos integrados analógicos a 650 modelos.

### E.4.4 Edición para estudiantes “STUDENT EDITION”

Es una versión limitada de *Electronics Workbench*, diseñada sólo para el uso del estudiante, la cual tiene una serie de limitaciones respecto a la versión para profesionales.

Las limitaciones de la misma son las siguientes:

- Permite realizar solamente cinco tipos de análisis: Punto de operación (CD), Frecuencia de corriente alterna (CA), Transitorio, *Fourier* y *Monte Carlo*.
- Los circuitos sólo pueden contener hasta 25 componentes analógicos activos (incluyendo diodos, rectificadores, rectificadores controlados de silicio (SCR), conductor de triodo bilateral, conmutador de diodo bilateral, diodos *schotkky*, transmisor bipolar de unión (BJT), transistores de efecto de campo de unión (JFET), transistores de efecto de campo de semiconductor a óxido metálico (MOSFET), GASFETs, amplificadores operacionales y líneas de transmisión con pérdidas o sin pérdidas.
- Los circuitos pueden contener un total de 100 componentes, sin contar con circuitos emisores de tierra o conectores.
- Todos los componentes están disponibles a excepción del componente *SPICE Netlist* (lista de red) que se encuentra en los archivos binarios varios.

### E.4.5 Programa demostración de la versión 5

- El circuito puede contener hasta 10 componentes análogos activos.
- No se permite guardar los circuitos creados, ni la impresión.
- Sólo permite cargar los circuitos ejemplo proporcionados en la demostración.
- Están disponibles aquellos análisis asociados a los circuitos ejemplo que se proporcionan en la demostración.
- No permite la importación ni exportación de *netlist*.
- No se pueden crear subcircuitos.
- Sólo se pueden utilizar los modelos ideales.
- Sólo están disponibles dos circuitos integrados de cada serie.

## E.5 INSTRUCCIONES DE INSTALACIÓN

Para realizar la instalación desde el CD-ROM de *Electronics Workbench 5.0 versión de estudiante* se deben efectuar los siguientes pasos:

- Introducir el CD-ROM en la unidad correspondiente.
- Visualizar con el explorador de *Windows* el contenido del directorio raíz del CD-ROM.
- Hacer doble clic con el ratón sobre el icono **Setup.exe**.

Al seleccionar dicho archivo se debe llegar a la ventana de instalación de la aplicación que se muestra en la Figura E.1.

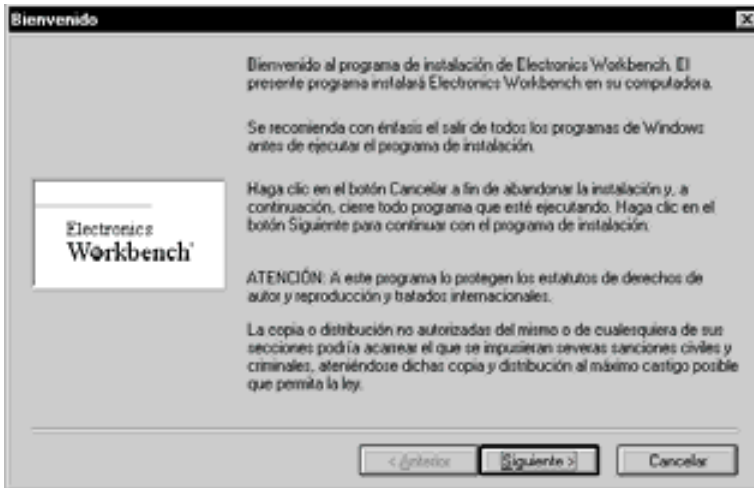


Figura E.1. Pantalla inicial de bienvenida

Haciendo *clic* sobre el botón denominado “Siguiente>” se pasa a la pantalla de la Figura E.2, en la que se solicita los datos del usuario. Una vez introducidos dichos datos se hace *clic* sobre el botón denominado “Siguiente>”, para continuar con la instalación.

Aparece la siguiente pantalla de la Figura E.3, en la que se debe introducir el número de serie que ha sido facilitado por el distribuidor. Una vez introducido dicho número se hace clic sobre el botón “Siguiente>”.

En la siguiente pantalla, que se muestra en la Figura E.4, se especifica el directorio del disco duro en el cual se almacenarán los archivos del programa. Si no se especifica ningún directorio, el programa guardará, haciendo clic sobre el botón “Siguiente>”, los archivos en el directorio por defecto, denominado C:\EBW5\ dentro del directorio raíz del disco duro.



Figura E.2. Pantalla de identificación de usuario



Figura E.3. Pantalla de introducción del número de serie

En la siguiente pantalla, que se muestra en la Figura E.5, se debe seleccionar el tipo de símbolos esquemáticos que se desean utilizar, ANSI (Instituto de normalización estadounidense) o DIN (Norma europea). El programa por defecto instalará la primera opción.

Haciendo clic sobre el botón "Siguiente>" se comienza la grabación de ficheros en el subdirectorio especificado, como se muestra en la Figura E.7, tras lo cual se finaliza la instalación.



Figura E.4. Pantalla para especificar el directorio donde se guarda el programa



Figura E.5. Pantalla de selección del tipo de símbolos esquemáticos ANSI o DIN

## E.6 COMPONENTES DE LA APLICACIÓN

Una vez realizada la instalación del software, tal y como se ha descrito anteriormente, aparecerá en el grupo **Programas** del botón **Inicio** un nuevo grupo de programas denominado *Electronics Workbench 5.0* y que se muestra en la Figura E.7.



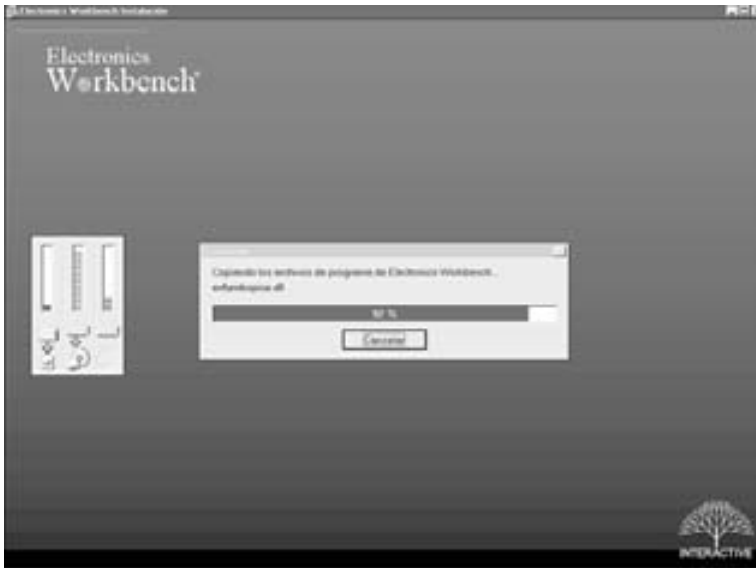


Figura E.6. Pantalla del proceso de grabación de ficheros en el subdirectorio especificado

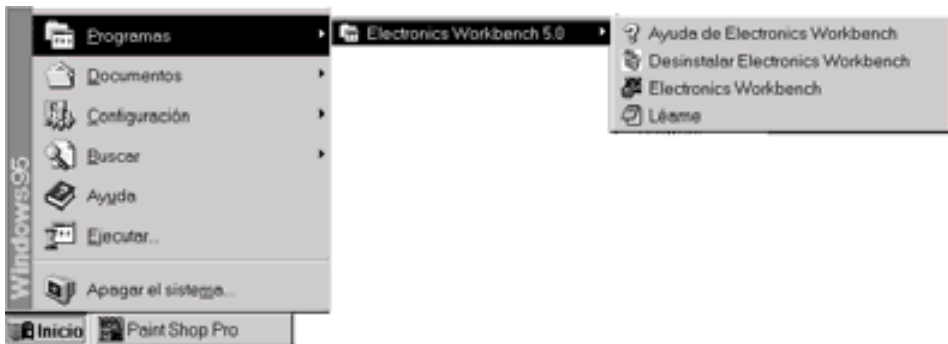


Figura E.7. Componentes de la aplicación *Electronics Workbench 5.0 Estudiante*

Como puede verse en dicha figura, los componentes que incorpora esta aplicación, son los siguientes:

- **Ayuda de Electronics Workbench:** Haciendo clic sobre este programa se arranca la ayuda que se encuentra disponible en un documento de hipertexto.
- **Desinstalar Electronics Workbench:** Permite desinstalar la aplicación completa del programa, tanto de los archivos y directorios grabado en el disco duro, como de los iconos y grupos de programas que se hayan creado durante la instalación.

- **Electronics Workbench:** Es el componente que arranca la aplicación propiamente dicha.
- **Léame:** Muestra un fichero de texto que contiene información de última hora acerca de la aplicación, las limitaciones de la misma, los errores detectados en el manual de usuario, etc.

## E.7 DESINSTALACIÓN

Para llevar a cabo la desinstalación de la aplicación, se debe pulsar en el menú de Windows: *Inicio/Programas/Electronics Workbench 5.0*, sobre el icono *Desinstalar Electronics Workbench*. Se irán abriendo sucesivamente las distintas ventanas de la Figura E.7. Posteriormente se efectúa el proceso de desinstalación como se muestra en la Figura E.8.

Sólo los archivos instalados con el programa serán eliminados. Todos los archivos que se hayan creado, por ejemplo, los de circuitos, no se verán afectados y los directorios donde se guardaron tampoco son eliminados.

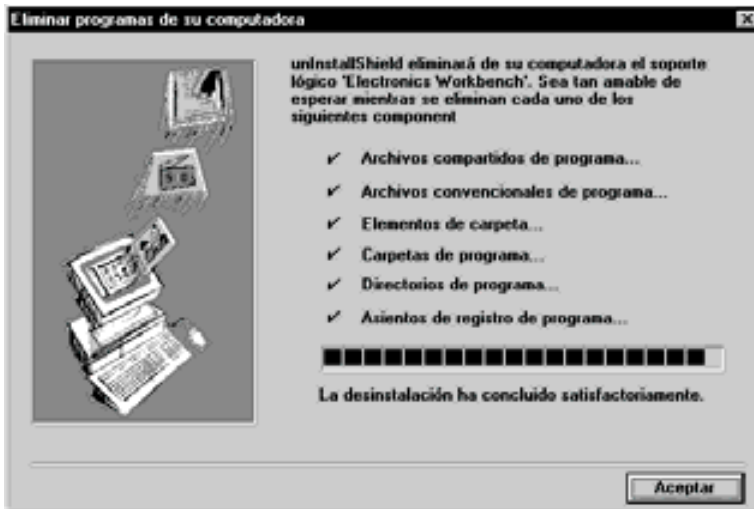


Figura E.8. Pantalla de desinstalación del programa

## E.8 MULTISIM. LA NUEVA APLICACIÓN DE EWB

La empresa canadiense *Electronics Workbench* (EWB) actualmente ha diseñado un sistema completo para el diseño de circuitos y placas de circuito impreso, compuesto por los programas *multiSIM* (simulación y análisis de circuitos), *ultiBOARD* (realización de placas de circuito impreso) y *ultiROUTE* (trazado de las pistas dentro del circuito). La unión de todos estos programas en un único paquete permite al usuario pasar de la idea al circuito electrónico real en un tiempo mínimo.

En los documentos que se incluyen en el CDRom#1 que acompaña a este libro se incluye un manual/tutorial en castellano de utilización de la aplicación de simulación de circuitos *multiSIM*.

Existe una versión Estudiante (*multiSIM* versión Estudiantes) de esta aplicación, que se puede adquirir al precio de: 18.800 ptas. (IVA incluido), a través del distribuidor (Prodel) de la misma, cuyos datos se describen en el Apartado de Programas Utilizados al comienzo del presente libro.

Si desea actualizar información o disponer de un tutorial más completo de esta herramienta, puede visitar las siguientes páginas web:

<http://www.electronicsworkbench.com/>

<http://www.prodel.es/>

En la primera de las páginas se dispone de versiones demostración de *multiSIM*, *ultiBOARD* y *ultiROUTE*, que le permitirán trabajar con un número limitado de componentes.

## GUÍA DE ORCAD DEMO V9

---

---

**Objetivos:**

- Conocer una introducción y descripción general de la aplicación *OrCAD Demo v9*.
- Conocer cómo se lleva a cabo la instalación de la aplicación *OrCAD Demo v9*.

**Contenido:** Se realiza una descripción de las principales características que presenta la herramienta *OrCAD Demo v9* y los principales componentes básicos de la parte de simulación analógica y digital que componen: el capturador de esquemáticos (*OrCAD Capture CIS Demo*) y el simulador analógico/digital y a la vez visualizador de los resultados (*OrCAD PSpice A/D Demo*), así como el procedimiento que se debe seguir para llevar a cabo su instalación.

**Simulación:** Este apéndice no tiene ejercicios de simulación por su carácter teórico.

## F.1 INTRODUCCIÓN

El paquete integrado *OrCAD Demo v9* es un conjunto de aplicaciones diseñadas para trabajar en Windows 95, Windows 98 y Windows NT (32 bits) que están dedicadas al diseño y simulación de circuitos electrónicos analógicos, digitales y mixtos. También permite el diseño de placas de circuito impreso (PCB). Cabe señalar que en versiones Demo superiores a *OrCAD Demo v9.1* ya no se incluye la aplicación *OrCAD Express*, dedicada al diseño y simulación de sistemas electrónicos digitales utilizando el lenguaje VHDL.

Con estas aplicaciones se puede realizar distintos tipos de análisis para cada circuito, como son: análisis temporal, análisis en continua, análisis en frecuencia, análisis de la variación de la temperatura, análisis de ruido, estudiar la variación de un parámetro dentro de un circuito, análisis de Fourier, análisis estadísticos de Monte Carlo, análisis de peor-caso, etc.

Se trata de un entorno gráfico que permite trabajar mediante ventanas y menús desplegables, en los que los comandos se pueden activar mediante el ratón o el teclado, pudiendo el usuario elegir en cada caso la forma más cómoda o rápida de hacerlo.

Al realizar una instalación completa del paquete integrado se crea en el disco duro un grupo de programas que contiene las aplicaciones necesarias para la simulación tanto analógica como digital denominado *OrCAD Demo*, de los que se tratará en este libro solamente la parte de analógica.

### F.1.1 Requerimientos del sistema

El sistema en el que se instale la aplicación deberá tener como mínimo las siguientes características:

#### Hardware

- Pentium 90 MHz compatible con IBM.
- 32 MB de RAM.
- 50-75 MB libre de disco duro.
- Tarjeta gráfica VGA 256 colores.
- Unidad de CD-ROM.
- Tarjeta de sonido de 16-bits (recomendado).

#### Sistema operativo

- Windows 95/98.
- Windows NT v 4.0 con *Service Pack 3* o 4.

## F.1.2 Limitaciones de esta versión de evaluación

Esta versión de evaluación tiene una serie de limitaciones respecto a la misma versión para profesionales.

A continuación, se citan algunas de las principales diferencias entre las dos versiones que existen para esta aplicación.

- En la versión de evaluación existen determinados comandos y opciones que no se encuentran disponibles.
- A la hora de crear el esquemático de un circuito con la aplicación *Capture CIS Demo v9*, se tienen las siguientes limitaciones:
  - ◆ El número máximo de símbolos (instancias) que se pueden colocar es 30.
  - ◆ En una librería de símbolos creada por el usuario, no se pueden tener más de 15 símbolos (instancias).
  - ◆ Las librerías de ejemplo solamente contienen 39 componentes analógicos y 134 digitales.
  - ◆ No permite exportar formatos EDIF.
  - ◆ El asistente de componentes de Internet no está disponible.
- A la hora de realizar la simulación del esquemático con la aplicación *PSpice A/D Demo*, las limitaciones que se tienen son las siguientes:
  - ◆ El número máximo de nodos que puede contener el circuito es 64.
  - ◆ El número máximo de transistores es 10.
  - ◆ El número máximo de primitivas de dispositivos digitales es 65.
  - ◆ El número máximo de líneas de transmisión ideales es 10 (ideales o no ideales).
  - ◆ El número máximo de pares acoplados de líneas de transmisión ideales es 4.
  - ◆ Solamente se pueden incluir (crear) estímulos de tipo senoidal (en la parte analógica) e impulsos de reloj (en la parte digital).
  - ◆ El editor de modelos sólo permite crear modelos para diodos.
  - ◆ No se pueden crear ficheros de datos con formato CSDF.
  - ◆ Solamente se pueden visualizar los resultados procedentes de simulaciones realizadas con la versión de evaluación.

### F.1.3 Instrucciones de instalación

Antes de comenzar la instalación de la aplicación se recomienda cerrar cualquier programa de antivirus residente que se tenga instalado en el ordenador, para así evitar posibles problemas durante la instalación.

Igualmente, se debe tener en cuenta que en el capítulo de Instalación de *OrCAD Demo v9*, se encuentra una explicación detallada del proceso de instalación completo, para aquellos lectores que quieran seguir de forma detallada dicho proceso.

Para realizar la instalación desde el CD-ROM de *OrCAD Demo v9* se debe introducir el mismo en la unidad de CD del ordenador y esperar unos instantes ya que dicho CD-ROM es autoejecutable. Transcurridos unos instantes aparecerá la primera pantalla de presentación en la que se muestran las opciones de instalación que se han definido para el mismo.

Si no se dispone del CD-ROM original de *OrCAD Demo v9* o bien ha pasado más de un minuto y el autoarranque no se ha iniciado, se debe realizar la instalación del mismo desde **Agregar o Quitar programas** del menú del **Panel de Control** de Windows 95/98 o bien en el botón **Inicio** seleccionar el comando **Ejecutar...** En el cuadro de diálogo de **Agregar o Quitar programas** o de **Ejecutar...** se debe seleccionar o escribir el archivo **D:\OrCADStart.exe**, donde **D:** indica la unidad de CD-ROM.

Realizando cualquiera de las formas anteriores, se debe llegar a la ventana de instalación de la aplicación de *OrCAD Demo v9* que se muestra en la Figura F.1. Si se desea saltar la *demo* que aparece al principio y llegar directamente a dicha ventana se debe pulsar la tecla [Escape]. Desde esta ventana puede comenzar la instalación del software, ver el contenido del CD-ROM o ver los archivos de información que se incluyen en el mismo, dependiendo de la opción que se elija.

En dicha pantalla se pueden realizar distintas acciones como visualizar ayuda sobre el capturador de esquemáticos, sobre FPGA, señales analógicas/digitales, sobre PCB, acceder al archivo *readme*, ver los archivos incluidos en el CD-ROM, ver el archivo de ayuda, ayuda para trabajar a través de la Web de OrCAD, instalar la aplicación o salir de la misma. En la Figura F.2 se muestra, a modo de ejemplo, la ayuda que aparece al seleccionar la opción *Schematic Entry/Component Info*, a través de la cual se puede navegar utilizando el botón *more*.

Para realizar la instalación se selecciona la opción *Install Demo Software* (Instalar Software de Evaluación), con lo que se pasará a la ventana que se muestra en la Figura F.2. En esta ventana se muestra una advertencia, indicando que se debe cerrar cualquier aplicación residente de antivirus para evitar problemas durante la instalación. En caso de tener aún alguno de estos programas instalado se recomienda cerrarlo antes de continuar.



Figura F.1. Pantalla inicial de instalación de OrCAD Demo v9

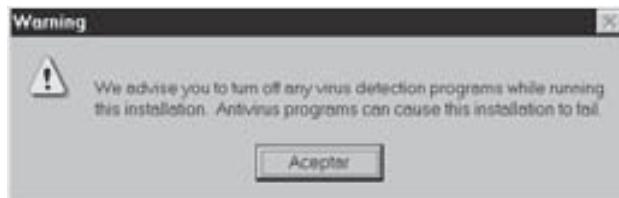


Figura F.2. Segunda pantalla de instalación de OrCAD Demo v9

Si no se tiene ningún programa de antivirus instalado en el ordenador, haciendo clic sobre el botón **Aceptar** se continúa con la instalación, con lo que se pasa a la ventana que se muestra en la Figura F.3. En dicha ventana se puede elegir una o varias de las cuatro aplicaciones disponibles:

- **Capture CIS**: para la captura de esquemáticos (esta aplicación es necesaria para poder trabajar con cualquiera de las dos siguientes).
- **Express**: para trabajar con VHDL.



- **PSpice A/D**: para realizar la simulación de circuitos analógicos, digitales y mixtos.
- **Layout Plus**: para realizar placas de circuito impreso.

En el caso que se indica en la Figura F.3 se han elegido las cuatro aplicaciones, si bien el lector podrá seleccionar las que más ajusten a sus necesidades.



Figura F.3. Pantalla de los componentes de OrCAD Demo v9

Al hacer clic en el botón *Next >* (Siguiente) se pasa a la siguiente ventana de instalación, y aceptando las opciones que aparecen por defecto en las distintas ventanas que van apareciendo se completa la misma; llegando a la ventana que se muestra en la Figura F.4 que es la última de la instalación. En esta ventana se tiene la opción de visualizar o no el fichero *Readme*, que contiene información de última hora acerca de la aplicación. Haciendo clic sobre el botón *Finish*, se finaliza la instalación.

Todos los ficheros y subdirectorios de la aplicación, si no se especifica lo contrario durante el proceso de instalación de la misma, son copiados en un directorio denominado **OrCAD Demo**, que cuelga del directorio **C:\Program Files**.

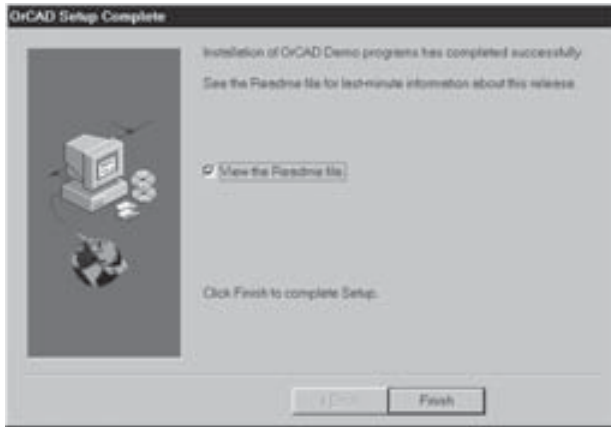


Figura F.4. Pantalla final de instalación de la aplicación OrCAD Demo v9

## F.1.4 Componentes de la aplicación OrCAD Demo v9

Una vez realizada la instalación del software, tal y como se ha descrito anteriormente, aparecerá en el grupo **Programas** del botón **Inicio** un nuevo grupo de programas denominado *OrCAD Demo*, en el que se encuentran los iconos correspondientes a las aplicaciones que componen este paquete integrado.

En la Figura F.5 se muestra el grupo de programas que se crea si se han seguido los pasos descritos anteriormente en el apartado de instrucciones de instalación.



Figura F.5. Programas que incorpora el grupo OrCAD Demo dedicado a la simulación analógica, digital y mixta

En la misma figura se muestran todas las aplicaciones, representadas por los iconos que las arrancan, que se incluyen en la parte de *OrCAD Demo v9*, dedicada a la simulación de circuitos electrónicos analógicos, digitales y mixtos.

A lo largo de este capítulo se describen con detalle las dos aplicaciones básicas que se utilizan para realizar la simulación de cualquier circuito. Estas aplicaciones son: la aplicación *Capture CIS Demo* para la realización del esquemático del circuito y la aplicación *PSpice A/D Demo* que es la encargada de realizar la simulación propiamente dicha y una vez finalizada la misma, a modo de osciloscopio software, permite ver de forma gráfica los resultados de la simulación, es decir, las formas de las señales en distintos puntos del circuito que se ha diseñado.

En el grupo de programas que se ha creado se encuentran todas las aplicaciones de *OrCAD Demo v9*, de las cuales en este libro se describirán las básicas y, a la vez, suficientes para realizar simulaciones en circuitos de carácter analógico, digitales y mixtos, que son las dos siguientes:

- ***Capture CIS Demo***. Es el capturador de esquemáticos, en el cual se crea el esquemático del circuito que se desea diseñar y simular, con los componentes, conexiones y valores de los mismos que se desee. También permite seleccionar el tipo de análisis que se va a realizar. Desde este programa se pueden ejecutar directamente el programa de simulación y visualización, sin necesidad de volver a la ventana en la que se encuentra su icono.
- ***PSpice AD Demo***. Es la aplicación que realiza la simulación propiamente dicha. Para ello utiliza una serie de algoritmos a través de los cuales realiza los cálculos y operaciones necesarios dependiendo del tipo de análisis que se haya seleccionado. Esta aplicación crea los ficheros que contienen los resultados de la simulación. Una vez finalizada la simulación, permite visualizar de forma gráfica en la pantalla los resultados que se han obtenido en la misma para las formas de onda de tensiones y corrientes en distintos puntos del circuito. Se puede considerar como el equivalente del osciloscopio en el laboratorio, por ello, se puede considerar como un osciloscopio software.

Estas dos aplicaciones serán tratadas con detalle a lo largo de este libro en apartados posteriores.

También se incluyen otras aplicaciones como son:

- ***Layout Plus Demo***. Esta aplicación permite el diseño y optimización de placas de circuito impreso, a partir del esquemático que se realice con la aplicación de captura de esquemáticos. Permite el diseño de los mismos de forma automática, manual o una mezcla de ambas. También permite importar y exportar el esquema a formato DXF.
- ***Layout Plus Demo SmartRoute Calibrate***. Esta aplicación realiza de forma automática el calibrado y ajuste del esquema para que cumpla las especificaciones estándar.

- ***PSpice Model Editor Demo.*** Es un editor de componentes, que permite crear y editar librerías de componentes, el símbolo que se desea que aparezca en el esquemático al colocar dicho componente y las características internas propias del mismo, es decir, el modelo asociado al mismo.
- ***PSpice Optimizer Demo.*** Esta aplicación permite optimizar los circuitos una vez que se han simulado, respecto al parámetro que se defina.
- ***PSpice Stimulus Editor Demo.*** Se trata de un editor de estímulos, en el cual se puede definir de una forma gráfica cualquier tipo de señal que posteriormente se desee utilizar como entrada para un circuito.
- ***Simulate Demo.*** A modo de presentación multimedia realiza una breve descripción de la aplicación, que muestra la forma de trabajo de la misma, así como las distintas funciones que permite realizar.

Estas aplicaciones, que si bien pueden resultar importantes en determinados casos particulares, no se describen con detalle en este libro, si bien se hace referencia a ellas e incluso alguna descripción breve de las mismas en el apartado dedicado a la aplicación *Capture CIS Demo*, desde la cual se puede acceder a las mismas a través de algunos de sus comandos.

Existen otras aplicaciones en las que se puede encontrar ayuda acerca del manejo y de la descripción de las herramientas de simulación que se incluyen en la aplicación, como son:

- ***Release Notes.*** Esta aplicación presenta, a través del navegador de Internet que se tenga instalado, una breve descripción de cada una de las aplicaciones que contiene el CD-ROM, la forma de llevar a cabo su instalación, las novedades que incorporan, etc.
- ***Otra documentación de ayuda.*** En el directorio que se crea durante la instalación, **C:\Program Files\OrCAD Demo\Document**, se encuentra una serie de archivos con formato PDF. A estos archivos se puede acceder bien desde la ayuda de cada una de las aplicaciones o bien de forma directa desde el explorador de Windows. Para poder visualizar dichos archivos es necesario tener instalada la aplicación *Acrobat Reader*.

En la Figura F.6 se muestra la pantalla completa de la aplicación *Adobe Reader*, en la que se puede visualizar a modo de hipertexto el fichero (**capqrc.pdf**) que contiene una guía rápida de la aplicación *OrCAD Capture*.

Como puede observarse en la misma, estos manuales se encuentran divididos en diferentes secciones según las diferentes aplicaciones y partes que componen el paquete software. Haciendo clic sobre cualquiera de las secciones, se accede directamente al índice de la misma, y desde éste se puede acceder al capítulo que se desee en cada caso.

También se puede encontrar ayuda de cada aplicación del paquete integrado en el menú **Help** (Ayuda) de cada una de ellas. Dentro de cada aplicación existen cursos de

aprendizaje que el lector puede realizar si lo desea. Por ejemplo, si se desea realiza el curso de la aplicación OrCAD Capture, se debe seleccionar en el menú *Help*, el comando *Learning Capture*, con lo que se comenzará dicho curso.

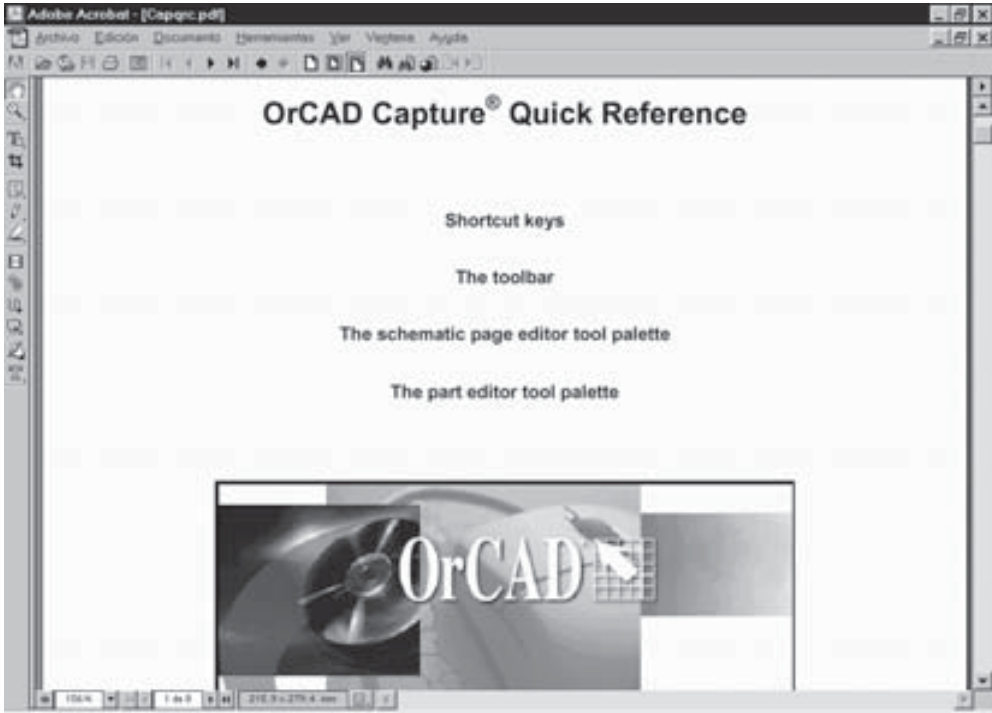


Figura F.6. Pantalla de la aplicación Adobe Reader en la que se muestra el contenido de un fichero de ayuda de OrCAD Demo

### F.1.5 Filosofía de trabajo con OrCAD Demo v9

La forma de trabajar con *OrCAD Demo v9* consiste en arrancar la aplicación *Capture CIS Demo* (capturador de esquemáticos) y en él crear o abrir el esquemático del circuito que se desea analizar. Si se trata de un circuito nuevo, lo primero será ir colocando en la hoja de trabajo los componentes que se van a necesitar para el circuito, a los cuales se les dará un nombre y un valor.

A continuación, se realizan las conexiones necesarias entre esos componentes mediante hilos (cables) y, por último, se seleccionan los distintos análisis que se desean realizar.

Una vez terminado el circuito y ajustado el tipo de análisis que desea se pasa a la simulación del mismo, para lo cual se ejecuta la aplicación *PSpice A/D Demo* (desde

el mismo programa capturador de esquemáticos), la cual, una vez realizados los cálculos necesarios, permite, automáticamente, visualizar la forma de onda de las señales que se desee.

## F.2 COMPONENTES DE LA APLICACIÓN ORCAD DEMO V9

Al arrancar el programa *Capture Demo* (capturador de esquemáticos), y después de seleccionar el tipo de proyecto, el nombre que se desea dar al mismo y las bibliotecas de símbolos que se utilizarán (una vez dentro del esquemático se pueden añadir nuevas bibliotecas) se abre automáticamente la pantalla que contiene los componentes y herramientas del mismo y que presenta el aspecto que se muestra en la Figura F.7. En dicha pantalla se pueden abrir simultáneamente varias ventanas en las que se pueden crear diferentes proyectos. Pero tan sólo puede estar activa una de ellas en cada momento.

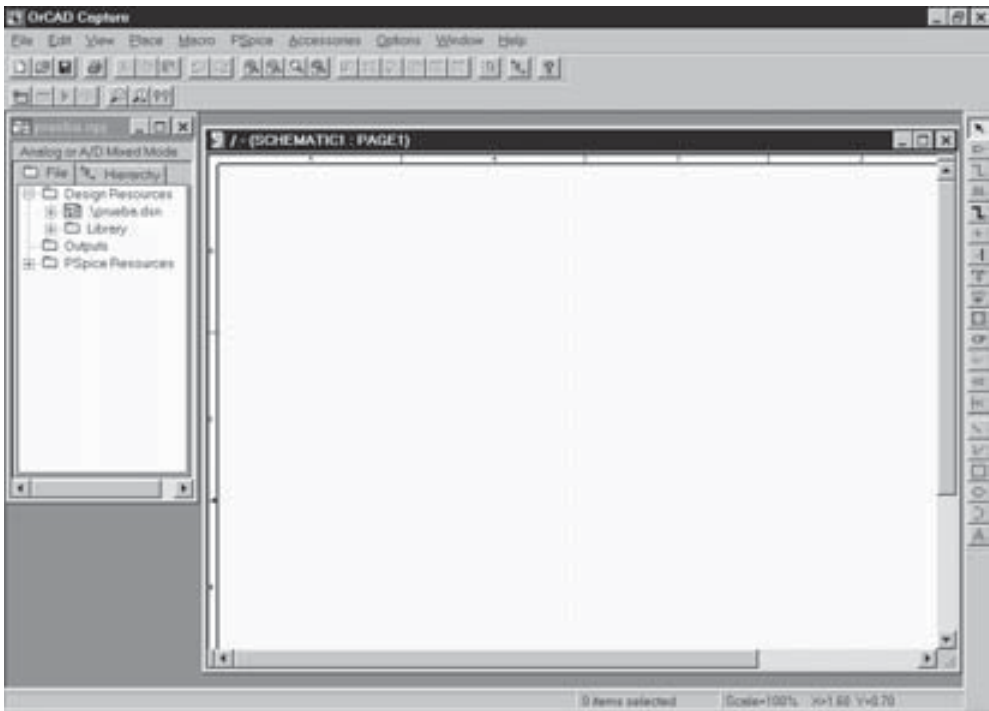


Figura F.7. Pantalla completa de la aplicación *Capture Demo*

En la ventana de la parte izquierda se muestra el nombre del proyecto (o circuito) que contiene el esquemático en cuestión. Este nombre puede ser cualquiera que el usuario desee, estando limitado por las condiciones impuestas por el sistema operativo

del PC en el que se esté trabajando. La extensión de estos archivos, que la añade el programa automáticamente, es **\*.opj**. En dicha ventana se muestran las distintas carpetas que contienen todos los archivos y elementos que componen el proyecto, entre los cabe destacar, los esquemáticos propiamente dichos, las bibliotecas de símbolos, las salidas generadas, otros recursos de PSpice, etc. Estas carpetas se pueden expandir o contraer haciendo doble clic sobre las mismas o bien haciendo clic sobre el signo + o -, que aparece en la parte derecha de su símbolo.

La ventana que aparece en la parte derecha corresponde al esquemático denominado, en este caso, **PAGE1**. Ésta es la zona de trabajo en la cual se colocan los componentes para generar el esquemático. Se debe tener en cuenta que es posible tener tantos esquemáticos como se desee.

Se puede observar en la Figura F.7, que dependiendo de si está activa la ventana del esquemático o la ventana del gestor de proyectos, la barra de menú y las barras de herramientas presentan pequeñas diferencias en uno y otro caso.

### F.3 MÁS INFORMACIÓN

Si desea actualizar información de esta herramienta, puede visitar las siguientes páginas web:

<http://www.pcb.cadence.com/>

<http://www.pspice.com/>

Igualmente, en España, también se puede acceder a través de la página web de su distribuidor:

<http://www.sidsapcb.com/>

## CONTENIDO DE LOS CD-ROM

---

---

**Objetivos:**

- Conocer el contenido que se incluye en los CD-ROM que acompañan a este libro.

**Contenido:** Los CD-ROM que se distribuyen junto con este libro, contienen los programas de simulación que se han utilizado en el libro, así como otras versiones de los mismos que actualmente también se encuentran en el mercado. También contiene documentos electrónicos con formato PDF, el programa para visualizar dichos documentos y ejemplos de esquemáticos de los circuitos tratados en el libro, así como documentos electrónicos con las características de los componentes de la casa comercial que lo ha facilitado.

**Simulación:** Este apéndice no tiene ejercicios de simulación por su carácter teórico.



## G.1 INTRODUCCIÓN

A continuación se describe la estructura y el contenido de los dos CD-ROM que acompañan a este libro, el CDROM#1 y el CDROM#2. Para cada directorio, se muestra en la parte izquierda, la estructura de directorios que se ha definido, y en la parte derecha se describe el contenido de cada uno de estos directorios.

## G.2 CONTENIDO DEL PRIMER CD-ROM (CDROM#1)

### G.2.1 Document

El directorio **Document** contiene los documentos electrónicos, en formato PDF, de los manuales y tutoriales desarrollados para las herramientas de simulación que se utilizan en el libro (también se incluyen los correspondientes a otras versiones), así como un manual de instalación del visualizador de PDF Adobe Acrobat Reader.

**Adobe** **Adobe Acrobat Reader** para Windows. El software de *Adobe*<sup>tm</sup> *Acrobat*<sup>tm</sup> proporciona acceso instantáneo a documentos en su forma original, independientemente de la plataforma informática que se esté utilizando. *Acrobat Reader* permite ver e imprimir cualquier archivo en formato *Portable Document Format* (PDF).

En este directorio se encuentra un documento que describe la forma de llevar a cabo la instalación de esta aplicación.

El nombre del archivo es:

`\I_Ar40.pdf`

**Car\_ESDL** **Características de Entrada y Salida de Dispositivos Lógicos.** En este directorio se encuentra un documento en el que se realiza una descripción de las características de entrada y de salida, con los distintos niveles de tensión posibles, de los dispositivos lógicos.

El nombre del archivo es:

`\Car_ESDL.pdf`

**Ewb** **Electronics Workbench Demo.** En este directorio se encuentran los manuales y tutoriales de tres versiones de esta aplicación, tal como se indica a continuación:

`\Ewb41\M_Ewb41.pdf`: Manual para la versión 4.1

`\Ewb50\M_Ewb50c.pdf`: Manual para la versión 5.0

`\Ewb50\M_Ewb50r.pdf`: Manual reducido para la versión 5.0

`\Ewb50\T_Ewb50c.pdf`: Tutorial para la versión 5.0

`\Ewb50\T_Ewb50r.pdf`: Tutorial reducido para la versión 5.0

`\Ewb_demo\T_Ewbdem.pdf`: Manual para la versión demo

`\Ewb_demo\T_Ewbdem.pdf`: Tutorial para la versión demo

**MultisIM** **MultisIM**. En este directorio se un documento en el que realiza una breve descripción y un tutorial que se ha desarrollado para esta aplicación:

El nombre del archivo es:

`\Car_ESDL.pdf`

**OrCAD** **OrCAD Demo v9**. En este directorio se encuentran el manual y el tutorial que se han desarrollado para esta aplicación:

`M_OrCAD.pdf`: Manual de OrCAD v9

`T_OrCAD.pdf`: Tutorial de OrCAD v9

## G.2.2 Ejemplos

Contiene los circuitos de todos los ejemplos que se han realizado en el libro, con el objeto de que el lector pueda disponer de ellos si lo desea y agilizar así su estudio.

Se trata de una estructura de directorios organizada por capítulos, que se corresponden con los capítulos de este libro, y dentro de cada capítulo por herramienta utilizada, *Ewb5*, *OrCAD9* o *VBv99*, en donde se encuentran todos los ejemplos implementados en este libro. A modo de ejemplo se describen a continuación cuatro de estos ejemplos con la ruta completa:

**Ejemplos** `\Cap03\Ewb5\03W0__16.ewb`: Se trata del esquema de un circuito que se estudia en el Capítulo 3 y se debe simular con la herramienta *Ewb*.

`\Cap04\OrCAD9\05R0__00.opj`: Se trata del esquema de un circuito que se estudia en el Capítulo 5 y que se debe simular con la herramienta *OrCAD*.

`\Cap06\VBv99\12V1__00.vhd`: Se trata del esquema de un circuito que se estudia en el Capítulo 6 y se debe simular con la herramienta *Vhdl VB99*.

Para trabajar con estos ejemplos se deberán seguir los siguientes pasos:

Copiar el directorio **Ejemplos** (con todos los subdirectorios y archivos que se incluyen dentro de él) en el disco duro. Una vez copiados en el disco duro, se debe quitar el atributo de “Sólo lectura” a todos los archivos que se incluyen. Para quitar este atributo de “Sólo lectura”, si está trabajando en Windows 95, debe hacer clic con el botón derecho sobre el archivo (o sobre varios) y en el comando **Propiedades** deseleccionar la casilla **Sólo lectura**; si está trabajando en Windows 3.x, deberá hacerlo a través del comando **atributos** de archivo.

Ir abriendo desde el programa correspondiente cada uno de los archivos y almacenándolo en el directorio del disco duro que se desee, para posteriormente poder realizar la simulación del mismo.

Siguiendo cualquiera de estos pasos anteriores, al realizar la primera simulación puede que se origine un mensaje de advertencia durante la misma. Dicho mensaje no es importante y se puede obviar sin más.

## G.2.3 Programa

Contiene los programas de simulación que se han utilizado para simular los circuitos descritos a lo largo del libro, así como el programa para visualizar los documentos electrónicos PDF.

**Adobe** **Adobe Acrobat Reader** para Windows. El software de *Adobe*<sup>™</sup> *Acrobat*<sup>™</sup> proporciona acceso instantáneo a documentos en su forma original, independientemente de la plataforma informática. *Acrobat Reader* permite ver e imprimir cualquier archivo en formato *Portable Document Format* (PDF).

En este directorio se encuentra una versión 4.0 de esta aplicación, que se puede ejecutar directamente desde el CD-ROM, si así se desea.

**Adobe\CD\Reader\Acrord32**

En el subdirectorio **Adobe\Instalers** se incluyen diferentes *kits* de instalación de la versión 4.0, para Windows 95 y 98, según el idioma que se desee. Para realizar la instalación del programa en versión castellano se debe seleccionar el siguiente archivo:

**Adobe\Instalers\Acrd4es.exe**

**Ewb** **Electronics Workbench Demo**. Contiene el software completo de la versión de evaluación de la aplicación de simulación *Electronics Workbench*, versión demo. La instalación de esta aplicación se realiza ejecutando el archivo **Rundemo.exe** de este directorio.

Igualmente se puede realizar la instalación desde el directorio **\Ewbdemo** utilizando el archivo **Setup.exe** que se encuentra en el mismo. El resultado que se obtiene en ambos casos es el mismo.

**MultiSIM MultiSIM Demo.** Contiene el software completo de la versión de demostración de la aplicación de simulación *MultiSIM*. La instalación de esta aplicación se puede realizar ejecutando el archivo **Start.pdf** que se encuentra en este directorio. Al hacer doble clic sobre dicho archivo se presenta una descripción de las aplicaciones que se incluyen en esta carpeta (*Multisim*, *Ultiboard* y *Ultiroute*) y la opción de realizar la instalación de cada una de ellas si se desea.

Igualmente se puede realizar la instalación ejecutando el correspondiente archivo **Setup.exe** de cada una de las aplicaciones indicadas que se encuentra en cada una de las siguientes rutas:

**MultiSIM\Msminst\Setup.exe** (Multisim)

**MultiSIM\Ubinst\Setup.exe** (Ultiboard)

**MultiSIM\Urinst\Setup.exe** (Ultiroute)

**OrCAD OrCAD Demo V9.1.** Contiene el software completo de la versión de evaluación de la aplicación de simulación *OrCAD*, versión 9. La instalación de esta aplicación se realiza ejecutando el archivo **OrCADStar.exe** que se encuentra en este directorio.

Igualmente se puede realizar la instalación desde el directorio **Software** utilizando el archivo **setup.exe**.

**Vhdl VHDL VB99.** No se incluye el Kit de instalación de la herramienta *VeriBest V99*, tal como inicialmente se había previsto, debido a que tras la compra de esta aplicación por parte de la compañía Mentor Graphics, ha cambiado el nombre del producto, así como su estrategia de comercialización. Para poder disponer de la nueva aplicación (*ModelSim*) es necesario descargarla accediendo a la página de Internet: <http://www.mentor.com>

### G.3 CONTENIDO DEL SEGUNDO CD-ROM (CDROM#2)

A continuación, se describe la estructura y el contenido del segundo CD-ROM, cuya etiqueta es **CDROM#2**. Para cada directorio, se muestra en la parte izquierda, la estructura de directorios según la cual se ha estructurado, y en la parte derecha se describe el contenido de cada uno de estos directorios.

### G.3.1 Fairchild

Contiene documentos electrónicos en formato PDF con las características de los componentes electrónicos de la casa comercial *Fairchild Semiconductor*, la cual ha facilitado cortésmente esta aplicación y la información que contiene.

**Acroread** **Adobe Acrobat Reader** para Windows. El software de *Adobe*<sup>™</sup> *Acrobat*<sup>™</sup> proporciona acceso instantáneo a documentos en su forma original, independientemente de la plataforma informática. *Acrobat Reader* permite ver e imprimir cualquier archivo en formato *Portable Document Format* (PDF). Este visualizador permite que el CDROM#2 sea navegable de forma independiente y sin necesidad de tener instalada la aplicación *Adobe Acrobat Reader* en el disco duro del ordenador.

**PDFs** **Descripción de componentes de Fairchild** para Windows. En este directorio se encuentra una serie de subdirectorios, en los que se organiza la información, distintos ficheros PDF, de los componentes electrónicos. Esta estructura de directorios se corresponde con la estructura de navegación que se ha definido para la aplicación. Para acceder a ventana principal o de entrada a la aplicación se debe hacer doble clic sobre el archivo:

**Welcome.pdf** que se encuentra en el directorio raíz del CDROM#2

El resto de archivos que se incluyen permiten acceder directamente a la información de los dispositivos electrónicos a los que hace referencia. También se puede acceder a ellos desde la ventana principal anterior.

Si se desea actualizar información se puede consultar la siguiente dirección de su página web: <http://www.fairchildsemi.com>

## LISTA DE EJEMPLOS

---

---

***Objetivos:***

- Facilitar la localización de los ejemplos desarrollados a lo largo del libro.

***Contenido:*** Se describe la ruta en la que se encuentran los distintos ejemplos que se han desarrollado en los capítulos del libro.

***Simulación:*** Este apéndice no tiene ejercicios de simulación, indica la ruta en la que se encuentran éstos.

D:\Ejemplos\Cap03\Ewb\03W0__16.ewb.....	110
Valor de una función de tres variables dada. EWB	
D:\Ejemplos\Cap04\VBv99\AND_algo\AND_algo.vpd.....	138
Descripción algorítmica de una puerta NAND. VeriBest	
D:\Ejemplos\Cap04\VBv99\AND_algo\AND_algo.vpd.....	140
Descripción comportamental de una puerta NAND. VeriBest	
D:\Ejemplos\Cap04\VBv99\AND_flujo\AND_flujo.vpd.....	141
Descripción comportamental por flujo de datos de una puerta AND. VeriBest	
D:\Ejemplos\Cap04\VBv99\AND_estr\AND_estru.vpd.....	142
Descripción estructural de una puerta NAND. VeriBest	
D:\Ejemplos\Cap04\Ewb5\04W0__01.ewb.....	147
Comportamiento de una puerta AND de tres entradas (tabla de verdad). EWB	
D:\Ejemplos\Cap04\Ewb5\04W1__01.ewb.....	149
Comportamiento de una puerta AND de tres entradas (cronograma). EWB	
D:\Ejemplos\Cap04\OrCAD9\04R0__01\04R0__01.opj.....	149
Comportamiento de una puerta AND de tres entradas. OrCAD	
D:\Ejemplos\Cap04\Ewb5\04W0__02.ewb.....	156
Simulación de un circuito cerrojo. EWB	
D:\Ejemplos\Cap04\OrCAD9\04R0__02\04R0__02.opj.....	157
Simulación de un circuito cerrojo. OrCAD	
D:\Ejemplos\Cap04\VBv99\Or\OR_algo.vpd.....	161
Descripción comportamental algorítmica y cronograma de una puerta OR. EWB	
D:\Ejemplos\Cap04\VBv99\Or\OR_stim.vhd.....	162
Fichero de estímulos para la simulación de una puerta OR. EWB	
D:\Ejemplos\Cap04\Ewb5\04W0__03.ewb.....	164
Comportamiento de una puerta lógica OR tres entradas (tabla de verdad). EWB	
D:\Ejemplos\Cap04\Ewb5\04W1__03.ewb.....	165
Comportamiento de una puerta lógica OR tres entradas (cronograma). EWB	
D:\Ejemplos\Cap04\OrCAD9\04R0__03\04R0__03.opj.....	166
Comportamiento de una puerta lógica OR tres entradas. OrCAD	

D:\Ejemplos\Cap04\Ewb5\04W0__04.ewb.....	168
Aplicación de puerta OR: subcircuito de alarma. EWB	
D:\Ejemplos\Cap04\VBv99\Not\NOT_flujo.vpd .....	171
Descripción comportamental por flujo de datos de una puerta NOT. VeriBest	
D:\Ejemplos\Cap04\VBv99\Not\NOT_stim.vhd.....	171
Fichero de estímulos para la simulación de una puerta NOT. VeriBest	
D:\Ejemplos\Cap04\Ewb5\04W0__05.ewb.....	173
Comportamiento de una puerta lógica NOT tres entradas (tabla de verdad). EWB	
D:\Ejemplos\Cap04\Ewb5\04W1__05.ewb.....	173
Comportamiento de una puerta lógica NOT tres entradas (cronograma). EWB	
D:\Ejemplos\Cap04\OrCAD9\04R0__05\04R0__05.opj .....	175
Comportamiento de una puerta lógica NOT tres entradas. OrCAD	
D:\Ejemplos\Cap04\Ewb5\04W0__06.ewb.....	177
Complemento a uno de un número binario de ocho bits. EWB	
D:\Ejemplos\Cap04\VBv99\Nand\NAND_estr.vpd.....	180
Descripción estructural de una puerta NAND. VeriBest	
D:\Ejemplos\Cap04\Ewb5\04W0__07.ewb.....	183
Comportamiento de puerta lógica NAND tres entradas (tabla de verdad). EWB	
D:\Ejemplos\Cap04\Ewb5\04W1__07.ewb.....	183
Comportamiento de puerta lógica NAND tres entradas (cronograma). EWB	
D:\Ejemplos\Cap04\OrCAD9\04R0__07\04R0__07.opj .....	185
Comportamiento de puerta lógica NAND tres entradas. OrCAD	
D:\Ejemplos\Cap04\Ewb5\04W0__08.ewb.....	187
Circuito que detecta niveles por debajo de un mínimo en dos depósitos. EWB	
D:\Ejemplos\Cap04\VBv99\Nor\NOR_algo.vpd .....	191
Descripción comportamental algorítmica de una puerta NOR. VeriBest	
D:\Ejemplos\Cap04\Ewb5\04W0__09.ewb.....	194
Comportamiento de puerta lógica NOR tres entradas (tabla de verdad). EWB	
D:\Ejemplos\Cap04\Ewb5\04W1__09.ewb.....	194
Comportamiento de puerta lógica NOR tres entradas (cronograma). EWB	



D:\Ejemplos\Cap04\OrCAD9\04R0__09\04R0__09.opj .....	195
Comportamiento de puerta lógica NOR tres entradas. OrCAD	
D:\Ejemplos\Cap04\Ewb5\04W0__10.ewb.....	197
Circuito que indica estado de puertas mal cerradas en un automóvil. EWB	
D:\Ejemplos\Cap04\VBv99\Buffer\BUFFER_flujo.vpd.....	201
Descripción comportamental de una puerta BUFFER. VeriBest	
D:\Ejemplos\Cap04\Ewb5\04W0__11.ewb.....	203
Comportamiento de una puerta lógica BUFFER (tabla de verdad). EWB	
D:\Ejemplos\Cap04\Ewb5\04W1__11.ewb.....	203
Comportamiento de una puerta lógica BUFFER (cronograma). EWB	
D:\Ejemplos\Cap04\OrCAD9\04R0__11\04R0__11.opj .....	205
Comportamiento de una puerta lógica BUFFER. OrCAD	
D:\Ejemplos\Cap04\VBv99\Xor\XOR_estr.vpd .....	209
Descripción estructural de una puerta XOR. VeriBest	
D:\Ejemplos\Cap04\Ewb5\04W0__12.ewb.....	212
Comportamiento de una puerta lógica XOR (tabla de verdad). EWB	
D:\Ejemplos\Cap04\Ewb5\04W1__12.ewb.....	212
Comportamiento de una puerta lógica XOR (cronograma). EWB	
D:\Ejemplos\Cap04\OrCAD9\04R0__12\04R0__12.opj .....	214
Comportamiento de una puerta lógica XOR. OrCAD	
D:\Ejemplos\Cap04\Ewb5\04W0__13.ewb.....	216
Circuito para realizar el control de calidad en una cadena de producción. EWB	
D:\Ejemplos\Cap04\VBv99\Xnor\XNOR_estr.vpd.....	219
Descripción estructural de una puerta XNOR. VeriBest	
D:\Ejemplos\Cap04\VBv99\Xnor\XNOR_stim.vhd.....	219
Fichero de estímulos para la simulación de una puerta XNOR. VeriBest	
D:\Ejemplos\Cap04\Ewb5\04W0__14.ewb.....	222
Comportamiento de una puerta lógica XNOR (tabla de verdad). EWB	
D:\Ejemplos\Cap04\Ewb5\04W1__14.ewb.....	222
Comportamiento de una puerta lógica XNOR (cronograma). EWB	

D:\Ejemplos\Cap04\OrCAD9\04R0__14\04R0__14.opj .....	224
Comportamiento de una puerta lógica XNOR. OrCAD	
D:\Ejemplos\Cap04\Ewb5\04W0__15.ewb.....	228
Expresión algebraica correspondiente a un circuito eléctrico dado. EWB	
D:\Ejemplos\Cap05\Ewb5\05W0__01.ewb.....	234
Minimización de una función lógica de tres variables. EWB	
D:\Ejemplos\Cap05\Ewb5\05W0__02.ewb.....	251
Minimización de una función lógica de tres variables (tabla de verdad). EWB	
D:\Ejemplos\Cap05\Ewb5\05W1__02.ewb.....	251
Circuito lógico de la expresión canónica en minterms simplificada. EWB	
D:\Ejemplos\Cap05\Ewb5\05W2__02.ewb.....	252
Circuito lógico de la expresión canónica en maxterms simplificada. EWB	
D:\Ejemplos\Cap05\Ewb5\05W3__02.ewb.....	253
Circuito lógico (puertas NAND) expresión canónica en minterms. EWB	
D:\Ejemplos\Cap05\Ewb5\05W4__02.ewb.....	253
Circuito lógico (puertas NAND) expresión canónica en maxterms. EWB	
D:\Ejemplos\Cap05\Ewb5\05W0__03.ewb.....	257
Simplificación de una función de tres variables (método de Karnaugh). EWB	
D:\Ejemplos\Cap05\Ewb5\05W1__03.ewb.....	257
Simplificación de una función de tres variables (circuito lógico). EWB	
D:\Ejemplos\Cap05\Ewb5\05W0__04.ewb.....	260
Simplificación de una función de cuatro variables (expresión booleana). EWB	
D:\Ejemplos\Cap05\Ewb5\05W1__04.ewb.....	260
Simplificación de una función de cuatro variables (circuito lógico). EWB	
D:\Ejemplos\Cap05\Ewb5\05W2__04.ewb.....	261
Función de cuatro variables simplificada y su circuito lógico. EWB	
D:\Ejemplos\Cap05\Ewb5\05W3__04.ewb.....	262
Circuito lógico de la función simplificada con puertas NAND. EWB	
D:\Ejemplos\Cap05\Ewb5\05W0__05.ewb.....	266
Simplificación de una función de cinco variables (minterms). EWB	

D:\Ejemplos\Cap05\Ewb5\05W0__06.ewb.....	268
Simplificación de una función de seis variables (minterms). EWB	
D:\Ejemplos\Cap06\Ewb5\06W0__01.ewb.....	353
Funcionamiento de una puerta NAND TTL-Estándar. EWB	
D:\Ejemplos\Cap06\Ewb5\06W0__04.ewb.....	357
Curva característica de entrada de una puerta NAND TTL-Estándar. EWB	
D:\Ejemplos\Cap06\Ewb5\06W0__03.ewb.....	360
Característica de salida a nivel alto de una puerta NAND TTL-Estándar. EWB	
D:\Ejemplos\Cap06\Ewb5\06W1__03.ewb.....	360
Característica de salida a nivel bajo de una puerta NAND TTL-Estándar. EWB	
D:\Ejemplos\Cap06\Ewb5\06W0__02.ewb.....	362
Función de transferencia de una puerta NAND TTL-Estándar. EWB	
D:\Ejemplos\Cap06\Ewb5\06W0__06.ewb.....	381
Función OR-exclusiva expresada en términos maxterm. EWB	
D:\Ejemplos\Cap06\Ewb5\06W0__07.ewb.....	385
Sistema con puertas triestado. EWB	
D:\Ejemplos\Cap06\Ewb5\06W0__05.ewb.....	391
función de transferencia de una puerta inversora CMOS. EWB	
D:\Ejemplos\Cap06\Ewb5\06W0__08.ewb.....	411
Interfaz TTL a CMOS mediante resistencia elevadora pull-up. EWB	
D:\Ejemplos\Cap06\Ewb5\06W0__09.ewb.....	413
Interfaz HC a CMOS mediante puerta en drenador abierto y pull-up. EWB	
D:\Ejemplos\Cap06\Ewb5\06W0__10.ewb.....	416
Interfaz CMOS a TTL mediante puerta en drenador abierto y pull-up. EWB	
D:\Ejemplos\Cap06\Ewb5\06W0__11.ewb.....	419
Interfaz CMOS a TTL mediante una puerta y el circuito interfaz específico. EWB	
D:\Ejemplos\Cap06\VBv99\06V0__01\06V0__01.vpd .....	423
Descripción en lenguaje VHDL de puertas inversoras con los retardos. VeriBest	
D:\Ejemplos\Cap06\OrCAD\06R0__03\06R0__03.opj .....	424
Descripción en VHDL de puertas inversoras de distintas subfamilias. OrCAD	

D:\Ejemplos\Cap06\VBv99\06V0__02\06V0__02.vpd .....	430
Descripción estructural de una puerta XOR (con puertas NOT y OR). VeriBest	
D:\Ejemplos\Cap06\OrCAD\06R0__04\06R0__04.opj .....	432
Puerta inversora con salida triestado y retardo. OrCAD	
D:\Ejemplos\Cap07\Ewb5\07W0__01.ewb.....	437
Circuito de un semisumador. EWB	
D:\Ejemplos\Cap07\Ewb5\07W0__02.ewb.....	441
Circuito sumador completo. EWB	
D:\Ejemplos\Cap07\OrCAD9\07R0__01\07R0__01.opj .....	442
Descripción funcional mediante VHDL de un sumador completo. OrCAD	
D:\Ejemplos\Cap07\Ewb5\07W0__03.ewb.....	445
Circuito de un sumador paralelo de dos bits. EWB	
D:\Ejemplos\Cap07\Ewb5\07W0__04.ewb.....	446
Circuito sumador paralelo con acarreo serie de cuatro bits. EWB	
D:\Ejemplos\Cap07\OrCAD9\07R0__02\07R0__02.opj .....	448
Circuito sumador paralelo con acarreo serie de ocho bits. OrCAD	
D:\Ejemplos\Cap07\Ewb5\07W0__05.ewb.....	454
Circuito sumador completo con generación de acarreo paralelo. EWB	
D:\Ejemplos\Cap07\OrCAD9\07R0__03\07R0__03.opj .....	455
Descripción VHDL de sumador paralelo con acarreo de ocho bits. OrCAD	
D:\Ejemplos\Cap07\Ewb5\07W0__06.ewb.....	459
Circuito semisumador. EWB	
D:\Ejemplos\Cap07\Ewb5\07W0__07.ewb.....	465
Circuito complementador a dos. EWB	
D:\Ejemplos\Cap07\Ewb5\07W0__08.ewb.....	467
Circuito sumador-restador de 4 bits. EWB	
D:\Ejemplos\Cap07\Ewb5\07W0__10.ewb.....	471
Sumador-restador de 4 bits codificado en binario signo magnitud. EWB	
D:\Ejemplos\Cap07\OrCAD9\07R0__04\07R0__04.opj .....	474
Descripción funcional mediante VHDL de la ALU 74382. OrCAD	

---

D:\Ejemplos\Cap07\Ewb5\07W0\_\_11.ewb..... 478

Circuito de una unidad Aritmético-Lógica (ALU) de cuatro bits

## LISTA DE HOJAS DE DATOS

---

---

**Objetivos:**

- Conocer cómo utilizar una herramienta de hojas de características de circuitos integrados.
- Conocer cómo interpretar los datos presentados en las hojas de características de circuitos integrados.

**Contenido:** Se describe la utilización e interpretación de las hojas de características de los circuitos integrados de electrónica digital. En primer lugar se muestran los pasos a seguir para acceder a las hojas de características de circuitos integrados lógicos de *FAIRCHILD Semiconductor*, presentes en el CDROM#2 que acompaña a esta publicación. Posteriormente, se indican los distintos apartados que componen las hojas de características o *Data Sheets* y su interpretación.

**Simulación:** Este apéndice no tiene ejercicios de simulación por su carácter teórico.

## I.1 ACCESO A LAS HOJAS CARACTERÍSTICAS DE C.I. DIGITALES PRESENTES EN EL CDROM

Por cortesía de Fairchild Semiconductor, en el CDROM#2 que acompaña a este libro se ha incluido una aplicación, en la que se muestran las hojas de características de circuitos integrados lógicos de dicha casa comercial.

Para acceder a dicha aplicación, se debe insertar el CDROM#2 en la unidad de CDROM del ordenador, abrir el explorador de Windows y en la ruta **D:\** hacer doble clic en el fichero **Welcome.pdf**. Una vez realizados estos pasos, se abrirá la pantalla de presentación o bienvenida de la aplicación, que se muestra en la Figura I.1.

Haciendo clic en cualquier punto de dicha ventana se accede a la ventana del menú principal de la aplicación, que se muestra en la Figura I.2, y desde la que se puede acceder a las características de los distintos componentes electrónicos: componentes analógicos, mediante la opción: *Analog & Mixed Signal*, a las de los componentes digitales, mediante la opción: *Logic*, a las de interfaces, mediante la opción: *Interface*, o a las de memorias no volátiles, mediante la opción: *Non-Volatile Memory*. También figuran las distintas oficinas internacionales de la empresa *FAIRCHILD Semiconductor* y la dirección de su página *Web*.



Figura I.1. Pantalla inicial del documento de hojas de características de circuitos integrados de FAIRCHILD Semiconductor



Figura I.2. Pantalla del menú principal de la aplicación

Haciendo clic con el ratón sobre la opción *Logic* se accede a las características de los componentes digitales, apareciendo la pantalla de la Figura I.3, en la que se puede seleccionar la subfamilia lógica correspondiente.



Figura I.3. Pantalla de componentes digitales donde seleccionar la subfamilia lógica

Haciendo clic con el ratón sobre la subfamilia lógica TTL Bipolar, se obtiene la pantalla que se muestra en la Figura I.4, donde se puede seleccionar una puerta o componente digital de dicha familia.



## I.2 APARTADOS E INTERPRETACIÓN DE LAS HOJAS DE CARACTERÍSTICAS (DATA SHEETS)

Siguiendo los pasos indicados en el anterior apartado se puede acceder a las características, por ejemplo, de la puerta NAND MM74C00 correspondiente a la subfamilia C (74C) CMOS, obteniéndose las hojas de características con los principales apartados que se explican a continuación.

La Figura I.5 corresponde a la primera hoja de características de un componente digital. En ella figuran los siguientes apartados:

- **Fabricante:** *FAIRCHILD Semiconductor.*
- **Fecha de creación:** Octubre de 1987 y de **revisión:** enero de 1999.
- **Nombre del componente:** MM74C00.
- **Descripción general** (*General Description*) donde se indica el tipo, ventajas y aspectos más relevantes del componente.
- **Características** (*Features*) más importante del componente.
- **Código** (*Ordering Code*) del componente en función del encapsulado.
- **Diagrama de conexionado** (*Connection Diagrams*) donde se describe las entradas y salidas del componente mediante la correspondencia entre su símbolo y los *pins* del circuito integrado.



Figura I.4. Pantalla donde se puede seleccionar el componente digital

La Figura I.6 corresponde a la segunda hoja de características, en la que figuran los siguientes apartados:

- **Valores máximos** (*Absolute Maximum Ratings*) o valores límite de trabajo del circuito integrado.

**FAIRCHILD**  
SEMICONDUCTOR

October 1987  
Revised January 1990

**MM74C00-MM74C02-MM74C04**  
**Quad 2-Input NAND Gate**  
**Quad 2-Input NOR Gate**  
**Hex Inverter**

**General Description**  
The MM74C00, MM74C02, and MM74C04 logic gates employ complementary MOS (CMOS) to achieve wide power supply operating range, low power consumption, high noise immunity and symmetric controlled rise and fall times. With features such as this the 74C logic family is close to ideal for use in digital systems. Function and pin out compatibility with series 74 devices minimizes design time for those designers already familiar with the standard 74 logic family.

All inputs are protected from damage due to static discharge by diode clamp to  $V_{CC}$  and GND.

**Features**

- Wide supply voltage range: 3V to 15V
- Guaranteed noise margin: 1V
- High noise immunity: 0.45V<sub>CC</sub> (typ.)
- Low power consumption: 10nW/package (typ.)
- Low power: TTL compatibility:  
Fanout driving 74L.

**Ordering Code:**

Order Number	Package Number	Package Description
MM74C00M	M14A	14-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-120, 0.150" Narrow
MM74C00N	N14A	14-Lead Plastic Dual In-Line Package (PDP), JEDEC MS-001, 0.300" Wide
MM74C02N	M14A	14-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-120, 0.150" Narrow
MM74C04M	M14A	14-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-120, 0.150" Narrow
MM74C04N	N14A	14-Lead Plastic Dual In-Line Package (PDP), JEDEC MS-001, 0.300" Wide

Devices are available in Tape and Reel. Specify tape packaging code in the order code.

**Connection Diagrams**  
Pin Assignments for DIP and SOIC

© 1990 Fairchild Semiconductor Corporation DS001477.pdf www.fairchildsemi.com

MM74C00-MM74C02-MM74C04 Quad 2-Input NAND Gate-Quad 2-Input NOR Gate-Hex Inverter

Figura I.5. Primera hoja de características de la puerta NAND 74C00

- Características eléctricas en corriente continua (*DC Electrical Characteristics*) o también denominadas **características estáticas**. Véase Capítulo 6, Apartado 6.2.1 Características estáticas de CI, donde se describen estos parámetros.
- Características eléctricas en corriente alterna (*AC Electrical Characteristics*) o también denominadas **características dinámicas**. Véase en el Capítulo 6 la misma referencia del punto anterior.

MM74C00-MM74C02-MM74C04

Absolute Maximum Ratings (Note 1)		Lead Temperature (Soldering, 10 seconds)				
Voltage Any Pin	-0.3V to +0.3V	300 °C				
Operating Temperature Range	-40°C to +85 °C					
Storage Temperature Range	-55°C to +150°C					
Operating V <sub>CC</sub> Range	3.0V to 15V	Note 1: "Absolute Maximum Ratings" are those values beyond which the reliability of the device cannot be guaranteed. Except for "Operating Temperature Range" they are not intended to imply that the device can actually be operated at these limits. The table of "Electrical Characteristics" provides conditions for actual device operation.				
Maximum V <sub>CC</sub> Voltage	18V					
Power Dissipation (P <sub>D</sub> )						
Dual-In-Line	700 mW					
Small Outline	500 mW					

DCElectricalCharacteristics						
Min/Max values are given for specified temperature range unless otherwise noted						
Symbol	Parameter	Conditions	Min	Typ	Max	Units
<b>CMOS TO CMOS</b>						
V <sub>IL1</sub>	Logical 1 Input Voltage	V <sub>CC</sub> = 0.8V V <sub>CC</sub> = 10V	0.8			V
V <sub>IL2</sub>	Logical 2 Input Voltage	V <sub>CC</sub> = 0.8V V <sub>CC</sub> = 10V			1.5	V
V <sub>OL1</sub>	Logical 1 Output Voltage	V <sub>CC</sub> = 0.8V, I <sub>O</sub> = -10 μA V <sub>CC</sub> = 10V, I <sub>O</sub> = -10 μA	0.8		2.0	V
V <sub>OL2</sub>	Logical 2 Output Voltage	V <sub>CC</sub> = 0.8V, I <sub>O</sub> = 10 μA V <sub>CC</sub> = 10V, I <sub>O</sub> = 10 μA			0.8	V
I <sub>IL1</sub>	Logical 1 Input Current	V <sub>CC</sub> = 10V, V <sub>IN</sub> = 15V		0.005	1.0	μA
I <sub>IL2</sub>	Logical 2 Input Current	V <sub>CC</sub> = 10V, V <sub>IN</sub> = 0V	-1.0	-0.005		μA
I <sub>IC</sub>	Supply Current	V <sub>CC</sub> = 10V		0.01	18	μA
<b>LOW POWER TO CMOS</b>						
V <sub>IL1</sub>	Logical 1 Input Voltage	4.5V, V <sub>CC</sub> = 4.75V			0.5	V
V <sub>IL2</sub>	Logical 2 Input Voltage	4.5V, V <sub>CC</sub> = 4.75V			0.5	V
V <sub>OL1</sub>	Logical 1 Output Voltage	4.5V, V <sub>CC</sub> = 4.75V, I <sub>O</sub> = -10 μA	0.4			V
V <sub>OL2</sub>	Logical 2 Output Voltage	4.5V, V <sub>CC</sub> = 4.75V, I <sub>O</sub> = 10 μA			0.4	V
<b>CMOS TO LOW POWER</b>						
V <sub>IL1</sub>	Logical 1 Input Voltage	4.5V, V <sub>CC</sub> = 4.75V	0.2			V
V <sub>IL2</sub>	Logical 2 Input Voltage	4.5V, V <sub>CC</sub> = 4.75V			1.0	V
V <sub>OL1</sub>	Logical 1 Output Voltage	4.5V, V <sub>CC</sub> = 4.75V, I <sub>O</sub> = -30 μA	0.4			V
V <sub>OL2</sub>	Logical 2 Output Voltage	4.5V, V <sub>CC</sub> = 4.75V, I <sub>O</sub> = 30 μA			0.4	V
<b>OUTPUT DRIVE (see Family Characteristics Data Sheet) (T<sub>A</sub> = 25 °C) (short-circuit current)</b>						
I <sub>OL1</sub>	Output Sourcing Current	V <sub>CC</sub> = 0.8V, V <sub>OL</sub> = 0V, V <sub>OH</sub> = 0V	-1.75			mA
I <sub>OL2</sub>	Output Sourcing Current	V <sub>CC</sub> = 10V, V <sub>OL</sub> = 0V, V <sub>OH</sub> = 0V	-8.0			mA
I <sub>OL3</sub>	Output Sourcing Current	V <sub>CC</sub> = 0.8V, V <sub>OL</sub> = 0.5V, V <sub>OH</sub> = 0V	-1.75			mA
I <sub>OL4</sub>	Output Sourcing Current	V <sub>CC</sub> = 10V, V <sub>OL</sub> = 10V, V <sub>OH</sub> = 0V	-8.0			mA

ACElectricalCharacteristics (Note 2)						
T <sub>A</sub> = 25°C, V <sub>CC</sub> = 10V, unless otherwise specified						
Symbol	Parameter	Conditions	Min	Typ	Max	Units
<b>MM74C00, MM74C02, MM74C04</b>						
t <sub>prop1</sub>	Propagation Delay Time	V <sub>CC</sub> = 0.8V		30	80	ns
t <sub>prop2</sub>	Logical 1 to 0	V <sub>CC</sub> = 10V		30	60	ns
C <sub>in</sub>	Input Capacitance	(Note 3)		6.0		pF
C <sub>OP</sub>	Power Dissipation Capacitance	Per Gate with fan-in (Note 4)		2		μF
Note 2: AC Parameter requirements are in 50% duty cycle unless otherwise noted. Note 3: Capacitance is measured at 1 kHz typical frequency. Note 4: C <sub>OP</sub> is the maximum thermal capacitance of the device. For complete information see Family Characteristics Application Note—AN-80.						

Figura I.6. Segunda hoja de características de la puerta NAND 74C00

La Figura I.7 muestra el apartado de **gráficas de características típicas de funcionamiento** del componente (*Typical Performance Characteristics*), tales como: funciones de transferencia, retardos, ruidos, frecuencias de trabajo, etc.

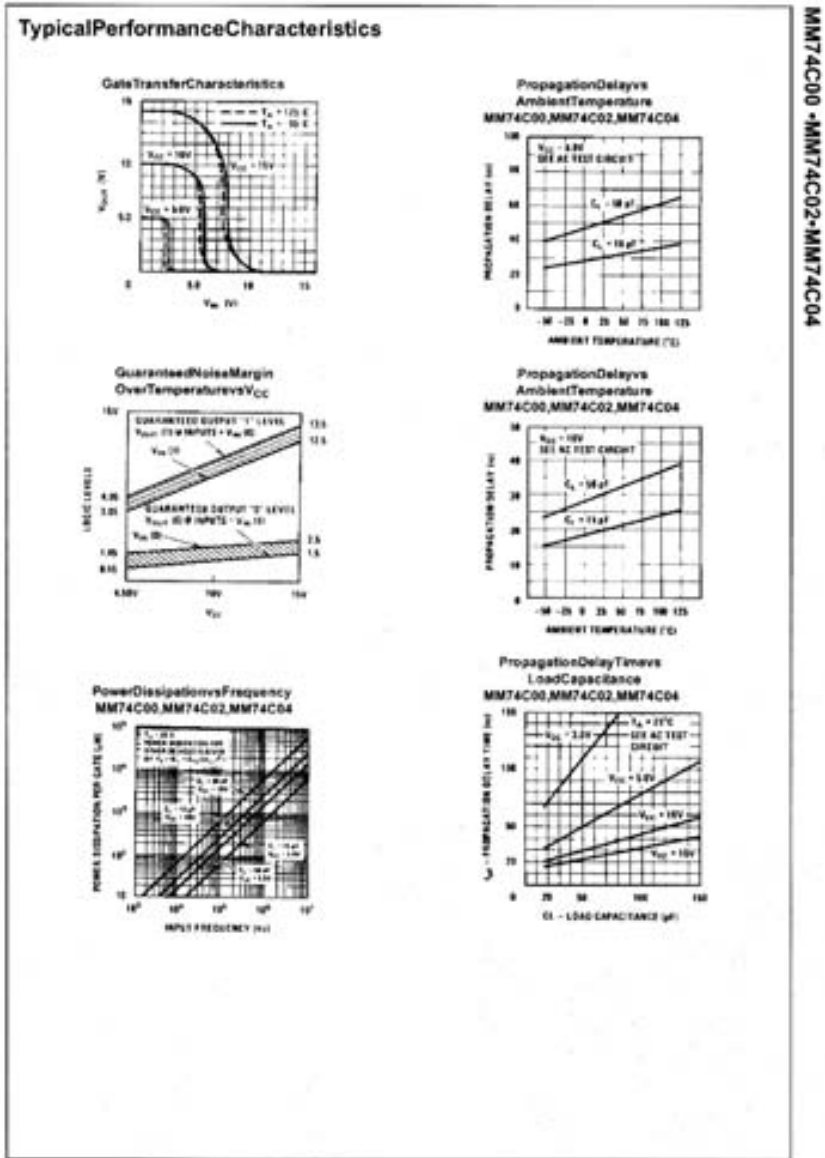


Figura I.7. Tercera hoja de características de la puerta NAND 74C00

La Figura I.8 muestra el apartado de las formas de ondas de **las señales en conmutación y el circuito de test utilizado** (*Switching Time Waveforms and AC Test Circuit*). Cronograma que representa el comportamiento dinámico del componente, sus retardos de propagación, retardos de subida y bajada, etc. También se representa las condiciones o circuito mediante el cual se ha realizado el test.

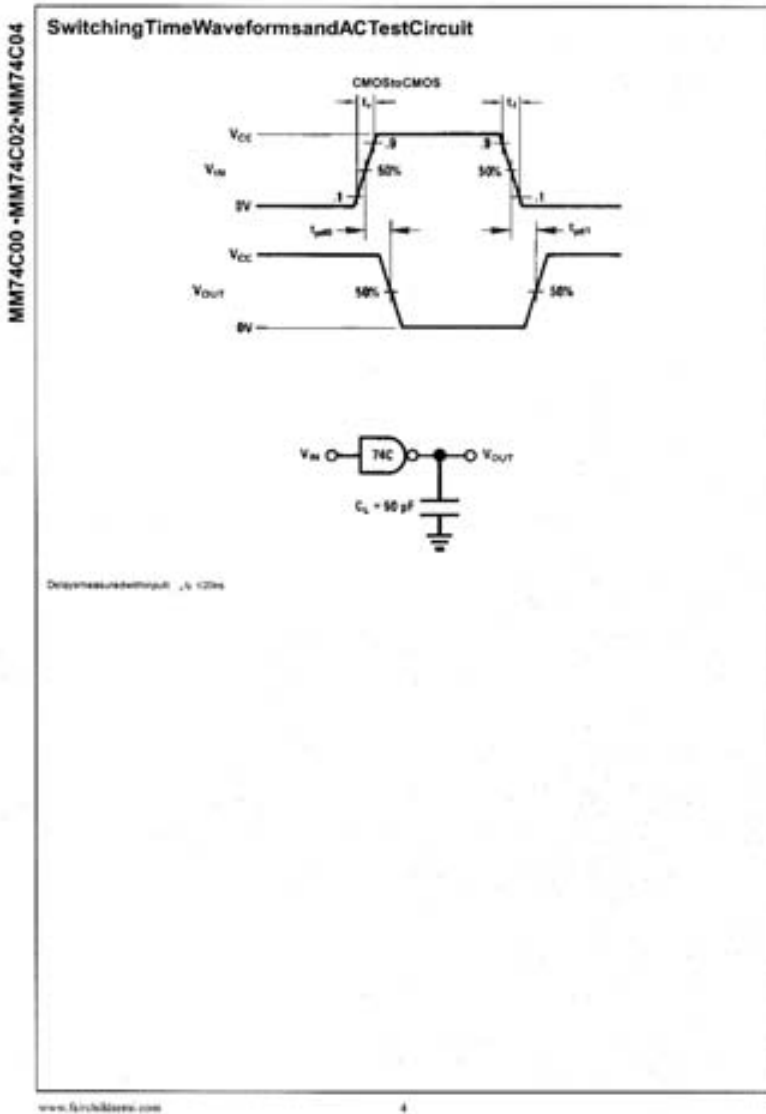


Figura I.8. Cuarta hoja de características de la puerta NAND 74C00

La Figura I.9 y la Figura I.10 muestran el apartado *Physical Dimensions*, donde se especifican las dimensiones físicas de los distintos encapsulados disponibles para esta puerta.

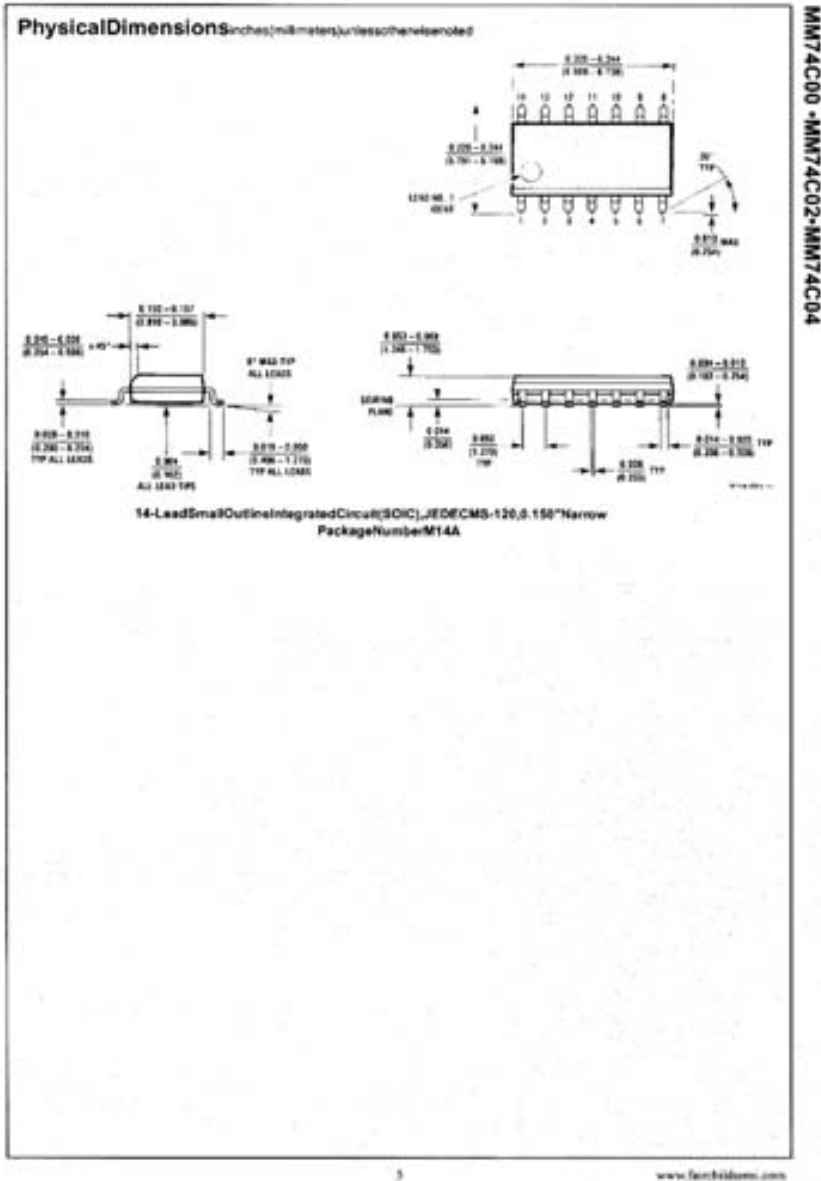


Figura I.9. Quinta hoja de características de la puerta NAND 74C00

MM74C00-MM74C02-MM74C04Quad2-InputNANDGate-Quad2-InputNORGate-HexInverter

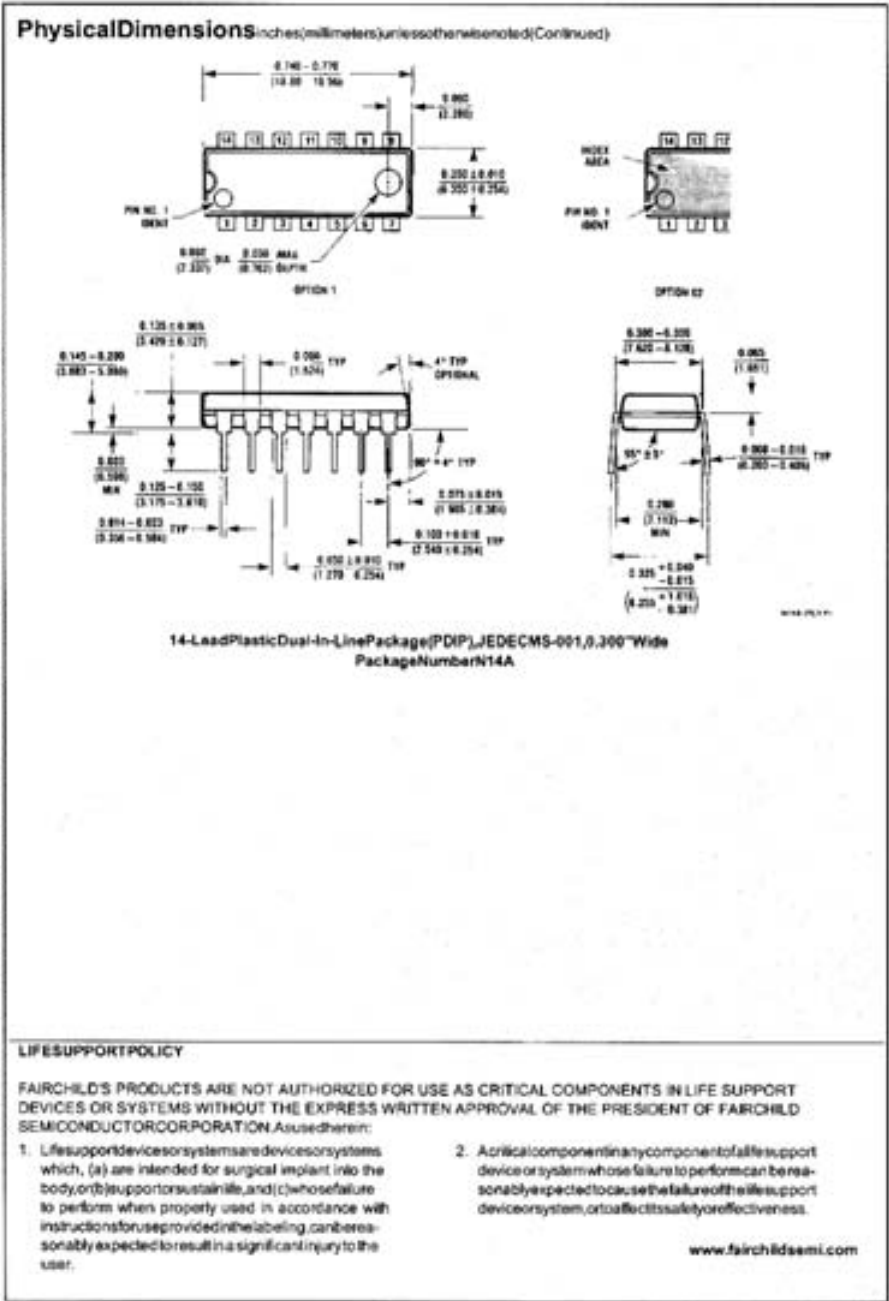


Figura I.10. Última hoja de características de la puerta NAND 74C00

## LECTOR DE DOCUMENTOS ELECTRÓNICOS ADOBE ACROBAT

---

---

**Objetivos:**

- Conocer los fundamentos de la aplicación *Adobe Acrobat Reader 4.0*, así como la forma de llevar a cabo su instalación.

**Contenido:** Se realiza una breve descripción de la aplicación *Adobe Acrobat Reader 4.0* para Windows 9x y 2xxx, que permite visualizar documentos electrónicos; se citan los requisitos hardware y software necesarios y se describe cómo llevar a cabo la instalación de la misma.

**Simulación:** Este apéndice por su carácter teórico no contiene ejercicios de simulación.



## J.1 INTRODUCCIÓN

La aplicación *Adobe Acrobat Reader 4.0* para Windows 9x y 2xxx permite visualizar en pantalla, e imprimir posteriormente si así se desea, documentos electrónicos en formato PDF (*Portable Document Format*). Se trata de una versión de libre distribución del programa para profesionales *Adobe Exchange 4.0*. La versión para profesionales permite, además de visualizar los documentos, crear nuevos documentos, modificar documentos ya creados, definir niveles de protección de los mismos, etc.

Los documentos electrónicos en general, aportan una serie de ventajas: por un lado, la facilidad de intercambio mediante comunicación electrónica que poseen y, por otro, la considerable reducción de costes (ahorro ecológico), al poder prescindirse completamente del papel si se desea. En particular, los documentos electrónicos con formato PDF aportan otras ventajas, además de las anteriormente citadas, ya que pueden visualizarse en cualquier plataforma, independientemente del software que se tenga disponible, pues el visor de estos documentos es de libre distribución. Además, estos documentos pueden ser posteriormente impresos con la impresora que se desee, sin que el formato se vea alterado, ya que éste es independiente de la misma.

## J.2 REQUISITOS DEL SISTEMA

A continuación se indican los requisitos, hardware y software, que debe tener el ordenador en el que se instale esta aplicación:

- Ordenador personal con procesador i486; se recomienda Pentium.
- Microsoft Windows 9x y 2xxx o Microsoft Windows NT 4.0 con Service Pack 3 o posterior.
- 8 MB de RAM para Windows 9x y 2xxx (recomendable 16 MB). 16 MB de RAM para Windows NT (se recomienda 24 MB).
- 10 MB de espacio en el disco duro, más 7 MB de espacio temporal disponibles durante la instalación.
- 50 MB de espacio en el disco duro, adicionales si se desean instalar fuentes asiáticas.
- Para visualizar Archivos PDF en un examinador de la Web: Netscape Navigator 4.0, incluyendo el componente Netscape Navigator 4.0 de Netscape Communicator 4.0, o Microsoft Internet Explorer 3.0 o posterior (también se puede usar Netscape Navigator 2.0.2, pero tiene ciertas limitaciones; por ejemplo, no podrá usarse para enviar un formulario PDF. También se podrían usar otros examinadores de la Web, siempre que sean totalmente compatibles con las API de Netscape).

## J.3 INSTALACIÓN

En el CDROM#1 que se adjunta con este libro es la versión de libre distribución de esta aplicación, *Adobe Acrobat Reader 4.0*.

La citada versión y la ruta en la que se encuentran sus archivos de instalación dentro del CDROM#1 es la siguiente (para la versión en castellano):

**CDROM#1:\Programa\Adobe\Installers\**

La instalación de la aplicación se debe realizar desde **Agregar o Quitar programas** del menú del **Panel de Control** de Windows 95. En el cuadro de diálogo de **Agregar o Quitar programas** se debe seleccionar el archivo **Acrd4es.exe** que se encuentra en la ruta del CDROM#1 que se ha descrito anteriormente, es decir:

**CDROM#1:\Programa\Adobe\Installers\Acrd4es.exe**

Una vez seleccionado el archivo y haciendo clic sobre el botón **Aceptar**, aparece un cuadro informativo indicando el progreso de la descompresión de los archivos. Una vez finalizada dicha descompresión en un directorio temporal, aparece la ventana de bienvenida que se muestra en la Figura J.1 en la cual se indica que es aconsejable cerrar todas las aplicaciones que se tengan abiertas, antes de continuar con la instalación.



Figura J.1. Primera pantalla de instalación de Adobe Acrobat Reader 4.0

Haciendo clic sobre el botón “Siguiente >”, se pasará a la segunda pantalla de la instalación (Figura J.2), que muestra el contrato de licencia de software.

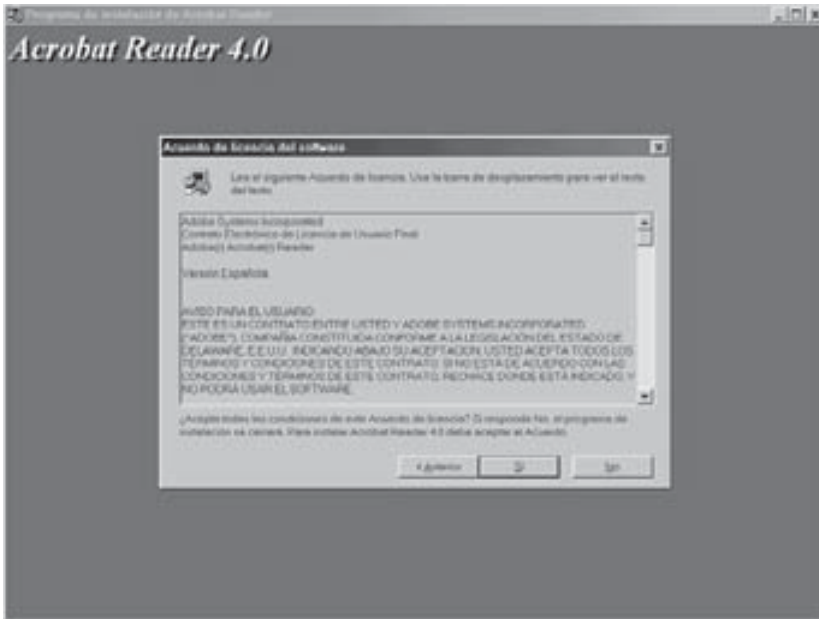


Figura J.2. Segunda pantalla de instalación. Contrato de licencia del software

Una vez leídas y aceptadas las condiciones del contrato, haciendo clic sobre el botón “Sí” se pasa a ventana que se muestra en la Figura J.3, en la que se muestra el directorio del disco duro en el que por defecto se instalarán los archivos del programa. Si se desea realizar la instalación en otro directorio distinto del que se muestra por defecto, se puede hacer seleccionándolo a través del botón **Examinar...** No obstante, se aconseja realizar la instalación en el directorio que el programa propone por defecto, que es el mostrado en dicha figura, ya que se facilitará el proceso de desinstalación, en caso de que se quiera llevar a cabo.

Haciendo clic sobre el botón “Siguiente >” comenzará a copiar los archivos de la instalación del programa en el directorio seleccionado anteriormente. En la Figura J.4 se muestra la ventana que aparece durante la fase de copiado de los archivos, en la cual se indica el grado de avance de la misma, así como la indicación de que la versión profesional tiene más funciones además de la visualización.

Una vez instalados todos los archivos, aparecerá la ventana que se muestra en la Figura J.5, agradeciendo haber elegido *Adobe Acrobat Reader*.

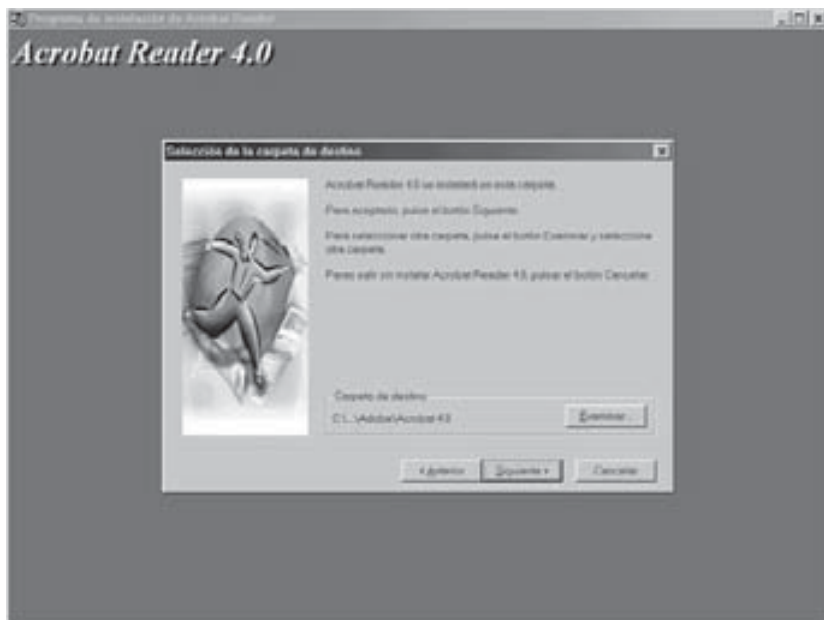


Figura J.3. Pantalla para seleccionar el directorio de destino

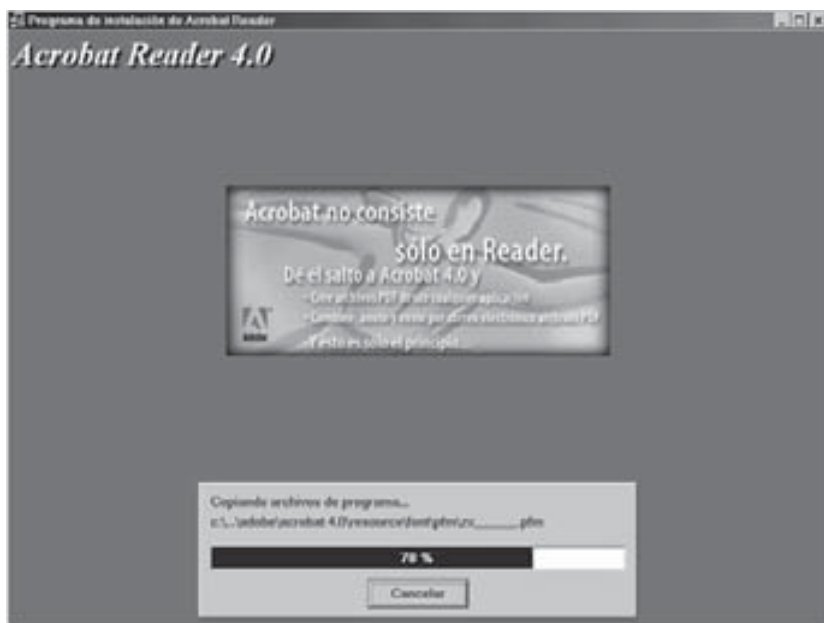


Figura J.4. Pantalla que indica grado de avance de la instalación



Figura J.5. Pantalla final de la instalación

Haciendo clic sobre el botón **Aceptar** se cerrará dicha ventana y se regresa a Windows, donde se habrá creado un nuevo grupo de programas dentro del botón **Inicio**, en la carpeta **Programas** que se denomina *Adobe Acrobat 4.0*, en el cual se encuentran los siguientes elementos:

- **Acrobat Reader 4.0.** Arranca la aplicación propiamente dicha.
- **Desinstalación Acrobat Reader 4.0.** Permite desinstalar la aplicación.
- **Léame de Acrobat Reader 4.0.** Muestra el archivo **Léame**, que contiene información de última hora sobre la aplicación.

## J.4 CARACTERÍSTICAS

En la Figura J.6 se muestra la pantalla completa de la aplicación *Adobe Acrobat Reader 4.0*, en la que se encuentra abierto, a modo de ejemplo, uno de los capítulos incluidos en el libro.

En esta aplicación se pueden realizar diferentes ajustes a la hora de visualizar el documento. Existen tres modos de presentar el documento:

- **Sólo la página.** Muestra solamente la página del documento, donde puede verse el texto y los gráficos que contenga.
- **Los marcadores y la página.** Esta opción divide la zona de visualización en dos partes. En la izquierda coloca los marcadores “índices” y en la derecha muestra la página con el texto y los gráficos.
- **Las miniaturas y la página.** Esta opción divide la zona de visualización en dos partes. En la parte de la izquierda coloca las páginas que componen el documento con tamaño reducido y numeradas y en la parte de la derecha muestra la página con el texto y los gráficos. Ésta es la opción que se ha seleccionado en el ejemplo de la Figura J.6.

En lo que a niveles de zoom se refiere, existen varias opciones: ver la página al tamaño real, ajustar el largo de la página a la ventana, ajustar el ancho de la página a la ventana, ajustar la zona de texto de la página al ancho de la ventana y, ampliar o reducir el tamaño de la página al valor que se desee.

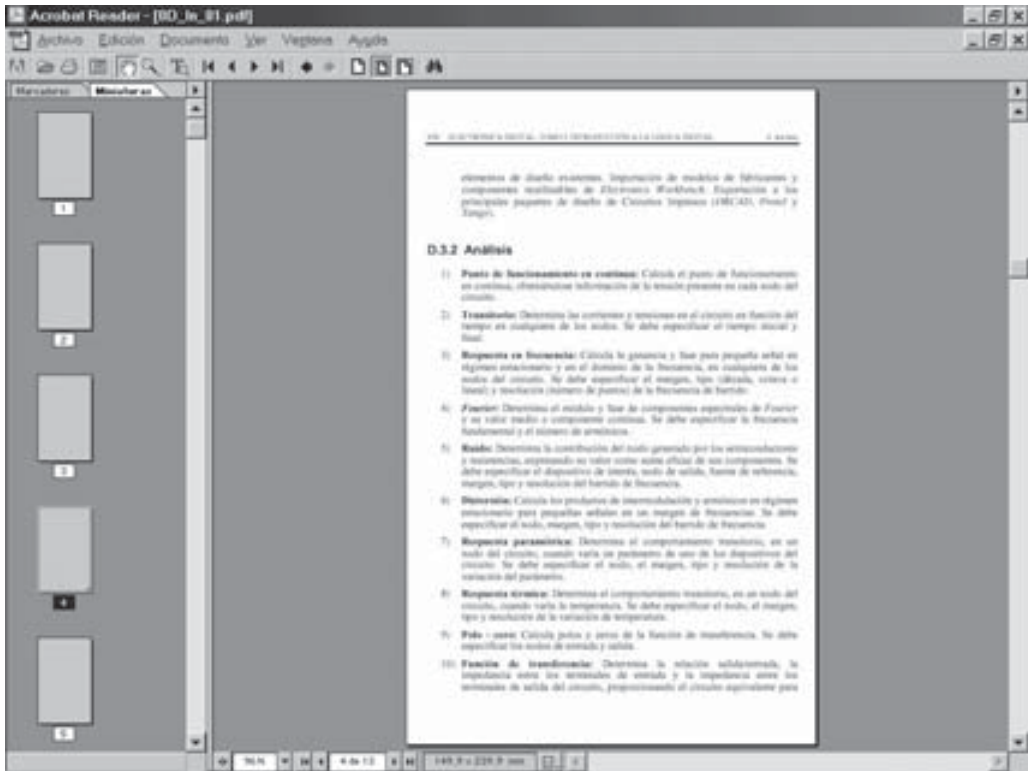


Figura J.6. Pantalla completa de la aplicación Adobe Acrobat Reader 4.0

En lo referente a la forma de presentar las páginas, existen tres opciones: las páginas de una en una, las páginas de forma continua, una tras otra y, las páginas de forma continua, con las páginas opuestas en paralelo.

Existen otras herramientas que permiten navegar por el documento: ir a la página anterior o siguiente, ir al principio o al final del documento, ir al modo de visualización anterior y siguiente, etc. Herramientas para realizar búsquedas de cadenas de caracteres en el documento, de comentarios, etc. Y la herramienta “mano” para mover la página dentro de la zona de visualización.

En los documentos que no se encuentren protegidos con contraseña, se podrá modificar, copiar, cortar, pegar, imprimir, etc. Los niveles de protección de cada documento se pueden ver dentro del comando **Seguridad...** que se encuentra en el menú **Archivo**, opción **Datos sobre el documento**.

## ÍNDICE ALFABÉTICO

---

---

### A

Abanico de entrada, 242  
Abanico de salida, 242  
Acarreo anticipado, 365, 382  
Acarreo paralelo, 346  
Acarreo serie, 346, 354, 357, 384  
ADC, 231, 265, 267, 270, 286, 290  
Adyacencia esencial, 146, 152  
Álgebra de Boole, 67, 71, 72, 188, 190  
ALSTTL, 281, 282  
ALU, 346, 382, 384  
ASIC, 256

### B

Base, 7, 12, 13, 14  
Base negativa, 11  
BCD, 265  
BICMOS, 255  
Biestable, 272, 275  
Binario, 25, 27, 29, 36, 50, 58

Binario Codificado en Decimal, 265  
Binario Natural, 265  
Binario puro, 33  
Bit de mayor peso, 22  
Bit de menor peso, 22  
Bit de signo, 48, 54  
Bit más significativo, 22, 79  
Bit menos significativo, 22, 264  
Bucle abierto, 267  
Bucle cerrado, 267, 284, 292

### C

Ciclo de trabajo, 247, 304  
Cifra, 7, 8, 9  
Circuito diferenciador, 283  
Circuito integrado, 230, 253  
Circuito sumador-restador, 369, 374, 379  
CMOS, 229, 255, 297, 302  
Codificación, 259, 264, 267  
Codificador, 269  
Código 2 entre 5, 86  
Código Aiken 2421, 72, 76, 81  
Código Aiken 5421, 72

Código ASCII, 82, 93  
 Código BCD 642-3, 73  
 Código BCD 8421, 71  
 Código BCD exceso 3, 74, 75, 92  
 Código BCD natural, 71, 74, 76, 87, 90, 392  
 Código BCD Natural, 391  
 Código binario natural, 69, 76, 81  
 Código biquinario, 86  
 Código denso, 68, 84  
 Código EBCDIC, 82  
 Código exceso 3, 76, 81  
 Código Gray, 69, 77, 78, 81, 92  
 Código Hamming, 87, 89, 90, 93  
 Código Johnson, 69, 77, 80, 81  
 Código natural 8421, 81  
 Código progresivo, 80  
 Código VHDL, 352, 358, 384  
 Códigos Aiken 2421, 72  
 Códigos autocomplementarios, 68  
 Códigos BCD, 69, 70  
 Códigos BCD Aiken, 71  
 Códigos BCD exceso 3, 75  
 Códigos binarios, 67  
 Códigos bipolares, 265  
 Códigos cíclicos, 68, 69, 76  
 Códigos continuos, 68, 69, 76  
 Códigos correctores de error, 86  
 Códigos de paridad, 84  
 Códigos detectores de error, 86  
 Códigos detectores de errores, 84  
 Códigos ponderados, 68, 71  
 Coma fija, 33, 34, 49  
 Coma flotante, 51, 52, 53, 54, 57, 59  
 Complementador, 374, 375, 377, 382  
 Complemento a diez, 37, 38  
 Complemento a dos, 37, 40, 42, 49, 369, 374, 378  
 Complemento a la base, 37, 39, 46  
 Complemento a nueve, 37, 39  
 Complemento a uno, 39, 46, 47, 49  
 Componente discreto, 231  
 Conversión, 13, 15, 75, 78  
 Convertidor A/D, 230, 241, 259, 276, 286, 293  
 Convertidor D/A, 231, 233, 235  
 Convertidor Sigma-Delta, 293, 294

Corrientes de offset, 235, 237, 239  
 Cronograma, 352, 365, 384  
 Cuantificación, 259, 261, 266

## D

DAC, 231, 239, 246, 284  
 Data Sheets, 265, 324, 327  
 Decimal, 16, 19, 25, 29, 49, 59  
 Desbordamiento, 371, 372, 382  
 Desviación típica, 264  
 Diezmador, 294  
 Dígito, 7, 8, 9, 14, 21, 23, 25  
 Diodo cluster, 279, 284  
 Diodo Schottky, 275, 283, 325  
 Dispositivos, 4  
 Doble precisión, 55, 56  
 DSP, 230  
 Duty cycle, 247, 304

## E

Electrónica analógica, 2  
 Electrónica digital, 2  
 Enteros, 8, 24, 29, 57  
 Error absoluto, 50  
 Error de cuantificación, 261, 267  
 Error de linealidad, 235, 267  
 Error de offset, 234, 235  
 Error de precisión, 267  
 Error instantáneo, 264  
 Estándar IEEE 754, 54, 55, 57, 58  
 Exponente, 53, 54, 58  
 Expresión booleana, 156  
 Expresión canónica, 82, 83, 86, 157  
 Expresiones normalizadas, 89

## F

Familia lógica, 229, 241, 265  
 Fan-in, 242, 243, 274, 289, 311  
 Fan-out, 242, 244, 264, 274, 296, 302  
 Filtro, 230, 294  
 Fondo de escala, 262, 281  
 FS, 233, 246, 281  
 FTTL, 284, 285  
 Función AND, 92, 98, 120, 257, 288



Función booleana, 74, 145, 162  
 Función canónica, 79, 80, 81, 86  
 Función de complementación, 92  
 Función de transferencia, 92, 232, 272, 299,  
 231, 233, 235  
 Función densidad de error, 264  
 Función equivalente, 145  
 Función implicación, 92  
 Función inhibición, 92  
 Función mínima, 135, 144, 177, 180  
 Función NAND, 92, 140, 263, 265  
 Función NOR, 92, 150  
 Función NOT, 131  
 Función nula, 92  
 Función OR, 92, 121, 361  
 Función seguidor, 161  
 Función unidad, 92  
 Función XNOR, 92, 179, 190  
 Función XOR, 79, 92, 169, 361  
 Funciones equivalentes, 78  
 Funciones incompletas, 189

## G

Ganancia, 234, 267

## H

HCMOS, 308  
 Hexadecimal, 16, 19, 27, 28, 29  
 Huntington, 135

## I

Impedancia, 254, 261, 297, 237, 241, 296  
 Implicados primos, 146, 175, 177

## K

Karnaugh, 134, 136, 147, 149, 158, 166, 173,  
 189

## L

Least Significant Bit, 233  
 LED, 149, 160  
 Lenguaje VHDL, 352, 357, 365  
 Leyes de De Morgan, 66, 88, 95

Linealidad, 234, 267, 274, 280  
 Lógica negativa, 96  
 Lógica positiva, 96  
 LSB, 22, 233, 252, 264  
 LSTTL, 278, 281  
 LTTL, 275

## M

Mantisa, 53, 54, 55, 58  
 Margen de ruido, 245, 296  
 Maxitérminos, 79  
 Maxterms, 79, 80, 86, 88, 89, 142, 157, 173  
 Media, 264  
 Método algebraico, 135  
 Método de Karnaugh, 135  
 Método de Petrick, 181  
 Método de Quine-McCluskey, 135  
 Microcontrolador, 230  
 Microprocesador, 346, 230  
 Minitérminos, 79  
 Minterms, 79, 80, 88, 89, 137, 146  
 Monotonicidad, 235  
 MSB, 22, 79, 252  
 Muestreador, 293  
 Muestreo, 259

## N

Normalización, 53, 54  
 Números Enteros, 32, 33  
 Números irracionales, 32  
 Números Naturales, 32, 33  
 Números Racionales, 33  
 Números Racionales, 32  
 Números Reales, 32  
 Nyquist, 260

## O

Octal, 16, 19, 25, 28, 29  
 Offset, 234, 267  
 Overflow, 371, 382, 386

## P

Par Darlington, 276, 279

Peso, 8, 14, 22  
 Polaridad, 248  
 Precisión, 5, 6, 19, 53, 57, 234, 274, 277, 280  
 Precisión simple, 57  
 Procesador digital de señal, 230  
 Producto de sumas, 88, 90  
 Producto lógico, 99, 140  
 Productos canónicos, 79  
 Puerta AND, 98, 183  
 Puerta BUFFER, 161  
 Puerta inversora, 131  
 Puerta NAND, 140, 150, 156, 165, 166, 185, 188  
 Puerta NOR, 150, 160, 185, 188  
 Puerta OR, 121, 163, 183  
 Puerta Trigger-Schmitt, 235  
 Puerta XNOR, 179, 184, 190  
 Puerta XOR, 169, 174, 179  
 Puertas lógicas, 96, 109

## Q

Q-M, 173, 175, 189  
 Quine-McCluskey, 134, 136, 173, 193

## R

Raíz cuadrática media, 295  
 Rango a fondo de escala, 233, 265  
 Rango de representación, 53  
 Red R-2R, 239, 241, 246  
 Red resistiva, 231, 239, 269  
 Representación binaria, 34  
 Resistencia térmica, 253  
 Resolución, 232, 233, 266, 270  
 Restador binario, 368, 389  
 Restador paralelo, 368  
 Retención, 259, 260  
 Ruido, 244, 246, 255, 230, 234, 293, 295  
 Ruido de cuantificación, 294, 295

## S

Sample and Hold, 260  
 Semirestador, 368, 369  
 Semisumador, 346, 347, 348  
 Señal analógica, 2

Señal digital, 3  
 Signo-magnitud, 34, 36, 46, 48, 49, 54, 58  
 Simple precisión, 57  
 Simple precisión, 55  
 Simplificación multifuncional, 209  
 Sistema, 3  
 Sistema analógico, 4, 6  
 Sistema binario, 11, 12, 21, 23, 25, 37, 39, 46  
 Sistema digital, 4, 5, 6, 33, 48, 50, 51  
 Sistema polinomial, 33  
 Sistemas de numeración, 7, 8  
 Sistemas polinomiales, 33  
 STTL, 276, 278, 279, 281  
 Subsistema, 3, 6  
 Suma de productos, 88, 90  
 Sumador, 348, 352, 361  
 Sumador paralelo, 346, 354, 355, 357, 360, 365  
 Sumador serie, 367  
 Sumas canónicas, 79

## T

Tabla cíclica, 180  
 Tensión umbral, 236, 282  
 Teorema de expansión, 80, 82, 85  
 Teorema de Shannon, 74, 85  
 Teorema de superposición, 242  
 Teoría de muestreo, 260  
 Término canónico, 79, 80  
 Términos canónicos, 135, 147, 167, 177  
 Tiempo de conversión, 234, 267, 270, 280, 296  
 Totem-Pole, 258, 287  
 Transistor Schottky, 275, 276  
 TTL, 255, 256

## U

Unidad Aritmético-Lógica, 346, 382, 387  
 Unidad de carga, 242, 243, 302

## V

Valor central, 262, 264  
 Valor numérico, 3, 7, 24, 39  
 VHDL, 329, 331