

Principios de Diseño de Sistemas Digitales

Guía Práctica
para alumnos de primer curso de
Grado en Ingeniería Informática
de Gestión
y Sistemas de Información

**GUILLERMO BOSQUE PEREZ
PABLO FERNANDEZ RODRIGUEZ**

Teknologia Elektronika Saila
Departamento de Tecnología Electrónica

ARGITALPEN ZERBITZUA
SERVICIO EDITORIAL

www.argitalpenak.ehu.es
ISBN: 978-84-9082-021-6



Universidad
del País Vasco
Euskal Herriko
Unibertsitatea



Principios de Diseño de Sistemas Digitales

Guía Práctica

PARA ALUMNOS DE PRIMER CURSO DE

GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

Guillermo Bosque Perez
Pablo Fernandez Rodriguez

Teknologia Elektronika Saila / Departamento de Tecnología Electrónica

Principios de Diseño de Sistemas Digitales

Guillermo Bosque Perez
Pablo Fernandez Rodriguez



Universidad
del País Vasco Euskal Herriko
Unibertsitatea

ARGITALPEN
ZERBITZUA
SERVICIO EDITORIAL

guillermo.bosque@ehu.es
pablo.fernandezr@ehu.es
www.ehu.es/guillermo.bosque

Índice general

Índice de figuras	9
Índice de cuadros	15
PREFACIO	17
OBJETIVOS Y ESTRUCTURA DEL TRABAJO	19
GLOSARIO	23
TÉRMINOS y ACRÓNIMOS	23
BIBLIOGRAFÍA - Observaciones	27
Capítulo 1. REPRESENTACIÓN DE LA INFORMACIÓN	29
1.1. INTRODUCCIÓN	29
1.2. SISTEMAS DE NUMERACIÓN POSICIONAL	33
1.3. REPRESENTACIÓN DE NÚMEROS CON SIGNO	40
1.4. NÚMEROS EN COMA FLOTANTE	48
1.5. CÓDIGOS ALFANUMÉRICOS	53
EJERCICIOS	57
Capítulo 2. INTRODUCCIÓN AL ÁLGEBRA DE BOOLE Y PUERTAS LÓGICAS	61
2.1. ÁLGEBRA DE BOOLE	61
2.2. PUERTAS LÓGICAS	74
EJERCICIOS	84
Capítulo 3. BLOQUES COMBINACIONALES	89
3.1. INTRODUCCIÓN	89
3.2. MULTIPLEXOR	91
3.3. DEMULTIPLEXOR	99
3.4. PUERTAS TRI-ESTADO	101
3.5. SISTEMAS COMBINACIONALES ARITMÉTICOS	102
3.6. CODIFICADOR	107
3.7. DECODIFICADOR	108
3.8. APLICACIONES: CONVERTIDOR DE CÓDIGO	112
EJERCICIOS	113

Capítulo 4. BLOQUES SECUENCIALES	115
4.1. INTRODUCCIÓN	115
4.2. CIRCUITOS SECUENCIALES	116
4.3. REGISTROS Y CONTADORES	128
4.4. SISTEMAS SECUENCIALES SÍNCRONOS	134
EJERCICIOS	152
Capítulo 5. MEMORIAS	155
5.1. BLOQUE DE REGISTROS	156
5.2. MEMORIA	158
5.3. MEMORIA ESTÁTICA SRAM	160
5.4. MEMORIA DINÁMICA DRAM	163
5.5. CLASES DE MEMORIAS	165
5.6. SOPORTES	170
5.7. ORGANIZACION DEL ALMACENAMIENTO DE DATOS	170
EJERCICIOS	171
Capítulo 6. INTRODUCCIÓN A LA METODOLOGÍA DE DISEÑO DE SISTEMAS DIGITALES	173
6.1. INTRODUCCIÓN	173
6.2. DISEÑO DE SISTEMAS DIGITALES COMPLEJOS	174
6.3. SISTEMA DIGITAL PROGRAMABLE	176
6.4. DIAGRAMA ASM	177
6.5. EJEMPLOS DE DISEÑO	179
EJERCICIOS	188
Bibliografía	191
Índice alfabético	193
Apéndice A. CÓDIGOS VHDL DE LOS CIRCUITOS COMBINACIONALES DESARROLLADOS EN EL CAPÍTULO 3	195
Multiplexor 4 a 1	195
Demultiplexor 1 a 4	195
Sumador de 4 bits	196
Sumador-Restador de 4 bits	196
Puerta triestado	197
Codificador de 4 bits	197
Decodificador de 2 bits	197
Capítulo B. CÓDIGOS VHDL DE LOS CIRCUITOS SECUENCIALES DESARROLLADOS EN EL CAPÍTULO 4	199
Biestable Asíncrono	199
Biestable Asíncrono (latch) D	200
Biestable Síncrono (Flip-Flop) D	201
Biestable Síncrono (Flip-Flop) JK	202

Biestable Síncrono (Flip-Flop) T	203
Registro de Almacenamiento	204
Registro de Desplazamiento (Shift Register)	205
Contadores	206
Diseño de Sistemas Secuenciales	207
Capítulo C. RESOLUCIÓN DE LOS EJERCICIOS	209
Tema 1	209
Tema 2	214
Tema 3	224
Tema 4	233
Tema 5	236
Tema 6	238

Índice de figuras

1.	Elementos de tecnología electrónica	30
2.	Función continua	31
3.	Función continua y función discreta	31
4.	Un circuito con un interruptor deja dos valores de tensión e intensidad	32
5.	Valores de tensión en Electrónica Digital	33
6.	Sistema de numeración posicional	34
7.	Binario, octal y hexadecimal	36
8.	Números positivos y sistema de magnitud con signo	41
9.	Complemento a uno	42
10.	Complemento a uno y complemento a dos	43
11.	Suma binaria con números negativos	46
12.	Código ASCII	54
13.	Los primeros 256 caracteres de UNICODE	55
1.	Mapa de Karnaugh genérico para 2 variables	69
2.	Mapa de Karnaugh genérico para 3 variables	69
3.	Mapa de Karnaugh genérico para 4 variables	70
4.	Mapa de Karnaugh para simplificar una función de 3 variables	70
5.	Mapas de Karnaugh que muestran los términos implicantes primos	71
6.	Mapa de Karnaugh (Maxtérminos) para simplificar una función de 4 variables	72
7.	Mapas de Karnaugh (mitérminos) para simplificar una función de 4 variables incompleta, (a) una elección, (b) otra elección	73
8.	Transistor MOSFET como interruptor	75
9.	Circuito NOT	75
10.	Circuito AND	76
11.	Circuito OR	76
12.	Cronograma de las puertas lógicas básicas	77
13.	Símbolos de otras puertas lógicas y sus tablas de verdad	77
14.	Símbolo del Buffer y su tabla de verdad	78

15. Síntesis de circuitos lógicos	78
16. Circuito de dos niveles	79
17. Equivalencias NAND y NOR	79
18. Definición de t_P	80
19. Circuito con riesgos	80
20. Riesgo en Y	81
21. (a) Detección de los riesgos y (b) su eliminación por Karnaugh	81
22. Circuito con riesgos corregidos	82
23. Circuito ejemplo y su descripción en VHDL	83
1. Circuito combinacional	89
2. Diseño jerárquico	91
3. Multiplexor	92
4. Multiplexor 2 a 1, puertas lógicas y bloque	92
5. Mux 4 a 1	93
6. Mux 4 a 1 con habilitación	95
7. Circuitos integrados multiplexores	95
8. Implementación de una función de 3 variables por medio de un mux 4 a 1.	97
9. Implementación de una función de 3 variables por medio de un mux 8 a 1.	99
10. Demultiplexor	100
11. Demultiplexor 1 a 4: puertas lógicas y bloque	101
12. Esquema de bloques con puertas tri-estado	101
13. Puertas tri-estado no inversoras: control = 1, control = 0	102
14. Puertas tri-estado inversoras: control = 1, control = 0	102
15. Circuito aritmético	102
16. Implementación del semisumador	103
17. Sumador completo	105
18. Sumador de operandos de 4 bits	105
19. Restador-Sumador de cuatro bits	107
20. Codificador	107
21. Esquema de bloques de un Codificador con prioridad	108
22. Decodificador	109
23. Decodificador de 2 bit con entrada de habilitación	110
24. Decodificador de 4 bits	110
25. Aplicación de un Decodificador en implementación de funciones	111
26. Convertidor de código	112

27. Visualizador de 7 segmentos	112
1. Circuito secuencial básico	117
2. Circuito secuencial básico con memoria	117
3. Esquema del latch S-R con puertas NOR	118
4. Esquema del latch S-R con puertas NAND	120
5. Cronogramas de latchs con puertas NOR y NAND	120
6. Esquema del latch S-R con puertas NAND y entrada de Control	121
7. Cronograma de un latch SR con entrada de control	121
8. Esquema del latch D con puertas NAND y entrada de Control	122
9. Cronograma de un latch D con entrada de control	122
10. Activación por nivel alto de un latch	123
11. Activación por flanco de un latch	123
12. Flip-Flop D	124
13. Cronograma de un FF-D activado por flanco de reloj	124
14. Flip-Flop JK	125
15. Cronograma de un FF-JK activado por flanco y señal de Clear (CLR)	126
16. Flip-Flop T	126
17. Cronograma de un FF-T	127
18. Flip-Flop D con entrada asíncrona de Reset	127
19. Cronograma de un FF-D activado por flanco de reloj y entrada asíncrona de Reset	128
20. Registro de almacenamiento	129
21. Registro de desplazamiento	130
22. Registro de desplazamiento - entrada serie y paralelo	130
23. Cronograma de un registro de desplazamiento derecha/izquierda	131
24. Cronograma de un registro de desplazamiento derecha/izquierda y entrada paralelo	131
25. Contador síncrono	132
26. Cronograma de un contador síncrono	133
27. Contador asíncrono	133
28. Cronograma de un contador asíncrono	134
29. Esquema de bloques de un sistema secuencial síncrono	134
30. Ejemplo de sistema secuencial	135
31. Diagrama de estados de un sistema secuencial síncrono - Entradas: XY / Salida: Z	137
32. Sistema secuencial síncrono modelo Mealy	138
33. Sistema secuencia síncrono modelo Moore	138
34. Cronograma de un circuito secuencial síncrono contador de 2 bits	139

35. Diagrama de estados de un circuito secuencial síncrono contador de 2 bits modelo Moore	140
36. Simplificación por Karnaugh para la obtención de los valores J_i K_i	144
37. Simplificación por Karnaugh para la obtención de los valores Z_i	145
38. Esquema de un contador de 2 bits modelo Moore	145
39. Diagrama de estados de un circuito secuencial síncrono contador de 2 bits modelo Mealy - Entrada: X / Salidas: Z_1Z_0	146
40. Simplificación por Karnaugh para la obtención de los valores D_i	150
41. Simplificación por Karnaugh para la obtención de los valores Z_i	151
42. Esquema de un contador de 2 bits modelo Mealy	151
1. Mapa de memoria.	156
2. Bloque de registros.	157
3. Memoria.	158
4. Ciclo de escritura.	159
5. Ciclo de lectura.	159
6. Celda binaria SRAM	161
7. Memoria SRAM de 4 palabras x 4 bits cada una.	162
8. Simulación funcional de la memoria SRAM	163
9. Celda DRAM.	164
10. Memoria ROM.	166
11. Conexión programable.	167
12. Dispositivo lógico programable (PLD)	169
1. Diagrama de bloques de un sistema digital	175
2. Diagrama de bloques de un sistema digital programable	176
3. Diagrama de estado Moore y diagrama ASM	177
4. Elementos del diagrama ASM	178
5. Diagrama ASM del contador de 1s.	180
6. Ruta de datos del contador de 1s.	181
7. Diagrama de bloques del contador de 1s.	181
8. Diagrama ASM de la unidad de control.	182
9. División en base 10.	183
10. Ejemplo de división binaria de cuatro bits.	184
11. Diagrama ASM del divisor	185
12. Ruta de datos del divisor.	186
13. Diagrama de bloques del divisor.	187

14. Diagrama ASM de la unidad de control del divisor.	187
15. Diagrama ASM para analizar	188
1. Mapa de Karnaugh - Mux 4a1 genérico - Función implementada	224
2. Función implementada con Mux 8a1	225
3. Función implementada con Decodificador 4a16	227
4. Bloque decodificador 2 a 4	227
5. Bloque decodificador 4 a 16	228
6. Conexiones del CI 74151 para la función $f = Sm(0,2,4,6)$.	229
7. Restador del ejercicio 9	230
8. Sumador binario natural a BCD	232
9. Sistema de memoria de $4K \times 8$	237
10. Diagrama ASM para analizar	238
11. Cronograma del sistema del ejercicio 1	239
12. Ruta de datos del multiplicador	241
13. Diagrama ASM del multiplicador: unidad de control	242
14. Multiplicador: diseño completo	242

Índice de cuadros

1.	Sistemas de numeración en diferentes bases	35
2.	Sistema decimal codificado en binario (BCD)	39
3.	Desbordamiento en suma binaria	47
4.	Números en coma fija con $n = 4$ bits	49
5.	Excepciones de la norma IEEE 754	52
6.	Valores extremos de la norma IEEE 754	53
1.	Tablas de la verdad de las operaciones básicas del álgebra de Boole	62
2.	Postulados	62
3.	Identidades básicas del álgebra de Boole	63
4.	Propiedades básicas del álgebra de Boole	63
5.	Leyes de De Morgan	63
6.	Demostración de la propiedad distributiva	63
7.	Operación suma exclusiva - XOR	64
8.	Demostración de la propiedad asociativa para la XOR	64
9.	Desarrollo de una función con mintérminos	65
10.	Desarrollo de una función con Maxtérminos	66
11.	Tabla dual	67
12.	Símbolos de las puertas lógicas básicas y sus tablas de verdad	76
1.	Tabla de verdad del mux 2 a 1	92
2.	Tabla de verdad del mux 4 a 1	93
3.	Tabla de verdad del mux 4 a 1 con habilitación	94
4.	Tabla de verdad de la implementación de una función de 3 variables con un mux de 4 a 1.	97
5.	Tabla de verdad de la implementación de una función de 4 variables con un mux.	98
6.	Tabla de verdad de un Demultiplexor 1 a 4	100
7.	Tabla de verdad de un semisumador	103
8.	Tablas de verdad del sumador completo	104
9.	Ejemplo básico de resta de 2 bits	106

10.	Tabla de verdad de un Codificador con prioridad	108
11.	Tabla de verdad del decodificador de 2 bits	109
12.	Tabla de verdad de un decodificador unido a un codificador	112
1.	Tabla de verdad de un latch S-R	118
2.	Estados prohibidos e inestables	119
3.	Tabla de verdad de un latch S-R con puertas NAND y puerta de control	121
4.	Tabla de verdad de un latch D	122
5.	Tabla de verdad de un flip-flop D	124
6.	Tabla de verdad de un flip-flop JK	125
7.	Tabla de verdad de un flip-flop T	127
8.	Tabla de verdad de un flip-flop D con entrada asíncrona de Reset	128
9.	Tabla de estados	136
10.	Evolución de estados	137
11.	Tabla de estados del modelo Moore	140
12.	Tabla de estados - asignación de estados	141
13.	Tabla de estados - asignación de estados final	141
14.	Tabla de verdad de un flip-flop JK	142
15.	Tabla conteniendo los valores $J_i K_i$ necesarios	143
16.	Tabla de salidas	144
17.	Tabla de estados del modelo Mealy	146
18.	Tabla de estados - asignación de estados	147
19.	Tabla de estados - asignación de estados final	148
20.	Tabla de verdad de un flip-flop D	148
21.	Tabla conteniendo los valores D_i necesarios	148
22.	Tabla simplificada conteniendo los valores D_i necesarios	149
23.	Tabla de salidas Z_i	150
1.	Mux 4a1	225
2.	Mux 8a1	226
3.	Decodificador 4a16	226
4.	Tabla de verdad de un Decodificador 2 a 4	227
5.	Tabla de verdad de la función requerida	228

PREFACIO

Esto solo sé: que no sé nada (Sócrates)

Grados recientes como el de Ingeniería Informática de Gestión y Sistemas de Información, y dada la orientación profesional que va a obtener el/la egresado/a para su incorporación en el mundo de la industria, requieren una adecuada fundamentación de las bases sobre las que se estructura el diseño de un computador, cómo se representa la información de manera binaria, como se opera con esa información, qué es un registro, porqué un registro es una memoria, como se estructura una memoria, cuando hay un tratamiento combinacional de la información y cuando depende del tiempo (tratamiento secuencial), etc.; dando al alumnado una visión básica no exhaustiva (propia de otras titulaciones) pero si rigurosa tanto técnica como científica.

El/la estudiante, en este primer curso, es donde se va a acercar a la estructura hardware de un computador, alejándose de ella en próximos cursos para adentrarse en lenguajes de programación y metodologías de gestión. Es por ello por lo que se hace necesario el exponer las bases electrónicas sobre las que se sustenta un computador y, dadas estas bases y la estructuración requerida, “vivenciar” como se dispone la información en un computador según códigos o normas internacionales.

Éste es el espíritu que ha guiado la elaboración del presente trabajo y esperamos que su lectura resulte amena y anime al lector/a a seguir profundizando en las diversas materias tratadas.

OBJETIVOS Y ESTRUCTURA DEL TRABAJO

La sabiduría sirve de freno a la juventud, de consuelo a los viejos, de riqueza a los pobres y de ornato a los ricos.
(Diógenes Laercio)

RESUMEN. Teniendo en cuenta la visión aportada en el Prefacio, procedemos a presentar los objetivos y la estructura de la presente guía.

La asignatura Principios de Diseño de Sistemas Digitales se imparte en el primer curso del Grado de Ingeniería Informática de Gestión y Sistemas de Información. El carácter de esta asignatura es obligatoria por lo que deberán cursarla todos los alumnos matriculados en este grado. Los objetivos a alcanzar por esta asignatura, teniendo en cuenta el perfil que se desea que los alumnos obtengan una vez finalizada la carrera, se pueden resumir en los siguientes puntos:

1. Objetivo General: será dotar al alumnado de conocimientos amplios y generales acerca de la representación de la información en un computador así como de los dispositivos (combinacionales y secuenciales) que configuran un computador y su funcionamiento.
2. Objetivos específicos:
 - a) Manejar la información numérica en diferentes formatos.
 - b) Operar aritméticamente la información numérica.
 - c) Ser capaz de analizar y diseñar un Sistema Digital sencillo Combinacional.
 - d) Ser capaz de analizar y diseñar un Sistema Digital sencillo Secuencial.
 - e) Aplicar las capacidades anteriores al diseño tanto con circuitos integrados de mercado como al diseño por medio de bloques funcionales.
 - f) Ser capaz de sintetizar Sistemas Digitales por medio de lenguajes de descripción Hardware como VHDL.
 - g) Ser capaz de seleccionar los circuitos integrados adecuados.

- h) Adquirir la capacidad de Análisis de circuitos integrados en base a la hoja de características.

Desde un punto de vista docente, el programa propuesto y su carga aproximada, para la parte teórica, es el siguiente:

- Capítulo 1. Representación de la información (6 horas)
- Capítulo 2. Introducción al álgebra de Boole y puertas lógicas (4+2 horas)
- Capítulo 3. Bloques combinacionales (5 horas)
- Capítulo 4. Bloques secuenciales (6 horas)
- Capítulo 5. Memorias (2 horas)
- Capítulo 6. Introducción a la metodología de diseño de sistemas digitales (2 horas)

El contenido de los capítulos mencionados es el siguiente:

- Glosario: Nomenclatura utilizada, tanto a lo largo del presente trabajo, como de otros términos de uso extendido.
- Capítulo 1: Tema introductorio que muestra los sistemas de representación de la información, tanto numérica como alfanumérica, más habituales en los sistemas digitales.
- Capítulo 2: El álgebra de Boole es la teoría matemática que describe al comportamiento de los circuitos digitales; por ello, antes de pasar a describir el funcionamiento y el diseño de estos circuitos, en este tema se presentan los conceptos básicos de dicha álgebra. Se estudian los componentes más simples de los sistemas digitales, esto es, las puertas lógicas, así como la forma de usarlas para construir sistemas digitales que realicen un a determinada función lógica.
- Capítulo 3: Se presentan los bloques combinacionales más utilizados. Se analiza su funcionamiento, así como la utilización de los mismos en el diseño de sistemas más complejos, con lo que pasaremos a diseñar en un nivel de complejidad superior al del tema anterior.
- Capítulo 4: En la misma línea que el tema anterior, aquí se estudian los dispositivos secuenciales síncronos más habituales y la metodología para desarrollar circuitos secuenciales más complejos.
- Capítulo 5: Este tema presenta uno de los componentes principales de los computadores: la memoria. Se describe el funcionamiento lógico de las memorias más habituales en los sistemas digitales.
- Capítulo 6: Establece los principios de una metodología de diseño de sistemas digitales. Más que ahondar en el desarrollo de diseños complejos, se trata de establecer las bases para entender diseños relativamente sencillos.

- Bibliografía: Muestra la Bibliografía consultada a lo largo del presente trabajo.

El programa propuesto para la parte práctica (Laboratorio) se explicita en cuaderno aparte de esta publicación: Cuaderno Docente de la Asignatura Principios de Diseño de Sistemas Digitales.

GLOSARIO

RESUMEN. En este capítulo se detallan todos los términos y acrónimos empleados a lo largo del presente trabajo, así como aquellos de uso extendido, con el fin de facilitar la lectura de todo lo expuesto.

TÉRMINOS y ACRÓNIMOS

- A/D:** Analógico/Digital
ACK: Acknowledge
ALU: Arithmetic Logic Unit
ARM: Advanced RISC Machine (Procesador embebido por Altera en sus dispositivos FPGA)
ASCII: American Standard Code for Information Interchange
ASIC: Application Specific Integrated Circuits
BEDO: Burst Extended Data Output RAM
BIOS: Basic Input Output System
BUS: Agrupación de Señales Eléctricas
CAD: Computer-Aided Design
CA/D: Conversión Analógico/Digital
CAE: Computer-Aided Engineering
CAM: Content Addressable Memory
CD: Compact Disk
CD/A: Conversión Digital/Analógico
CDROM: Compact Disk Read Only Memory
CPLD: Complex PLD
CPU: Central Processor Unit
CRC: Código de Redundancia Cíclica
CRT: Cathode Ray Tube
CTS: Clear To Send
D/A: Digital/Analógico
DCD: Data Carrier Detect
DCE: Data Communication Equipment
DDRRAM: Double Data Rate DRAM
DMA: Direct Memory Access
DRAM: Dynamic RAM
DSM: Distributed Shared Memory

- DSP:** *Digital Signal Processor*
- DSR:** *Data Set Ready*
- DTE:** *Data Terminal Equipment*
- DTR:** *Data Terminal Ready*
- DVD:** *Digital Vídeo Disk*
- E/S:** Entradas/Salidas
- ECC:** *Error Correcting Code*
- EDA:** *Electronic Design Automation*
- EDO:** *Extended Data Output RAM*
- EEPROM:** *Electrically Erasable Programmable Read Only Memory*
- EIA:** *Electronics Industries Association*
- EIDE:** *Extended IDE*
- EISA:** *Extended Industry Standard Architecture*
- EOT:** *End Of Transmisión*
- EPROM:** *Erasable Programmable Read Only Memory*
- FF-D:** *Flip-Flop D*
- FF-JK:** *Flip-Flop JK*
- FF-T:** *Flip-Flop T*
- FIFO:** *First In First Out*
- FPD:** *Field Programmable Device*
- FPGA:** *Field Programmable Gate Array*
- FPMRAM:** *Fast Page Mode RAM*
- FSK:** *Frequency Shift Keying*
- GAL:** *Generic Array Logic*
- GND:** *Ground*
- GMT:** *Grant*
- HW:** *Hardware*
- IBM:** *International Business Machines*
- IDE:** *Integrated Device Electronics*
- INT:** *Interrupt*
- INTA:** *Interrupt Acknowledge*
- IP:** *Intellectual Property*
- IRQ:** *Interrupt Request*
- ISA:** *Industry Standard Architecture*
- LATCH-D:** *Latch (cerrojo) D (Delay)*
- LATCH-SR:** *Latch (cerrojo) Set Reset*
- LCD:** *Liquid Crystal Display*
- LED:** *Light Emission Diode*
- LFU:** *Least Frequently Used*
- LRU:** *Least Recently Used*
- LUT:** *Look-up Table*
- μP:** Micro Procesador
- μC:** Micro Controlador
- MESI:** *(MESI protocol) Modified Exclusive Shared Invalid*

- MICROBLAZE:** Procesador RISC instanciado en dispositivos de Xilinx
- MIMD:** *Multiple Instruction Multiple Data*
- MISD:** *Multiple Instruction Single Data*
- MODEM:** Modulador/Demodulador
- MSI:** *Medium Scale Integration*
- MSYN:** *Master SYNchronitation*
- NACK:** *No Acknowledge*
- NIOS:** *National Institute of Open Schooling* (Procesador RISC instanciado en dispositivos de Altera)
- OTP:** *One Time Programmable*
- PAL:** *Programmable Array Logic*
- PBSRAM:** *Pipeline Burst Static RAM*
- PC:** *Personal Computer*
- PCI:** *Peripheral Component Interconnect*
- PIO:** *Peripheral Input Output*
- PLA:** *Programmable Logic Array*
- PLB:** *Processor Local Bus*
- PLD:** *Programmable Logic Device*
- PPC:** *Power PC* (Procesador de IBM)
- QDR:** *Quad Data Rate SRAM*
- PROM:** *Programmable Read Only Memory*
- RAID:** *Redundant Array Inexpensive Disks*
- RAM:** *Random Access Memory*
- RDRAM:** *Rambus DRAM*
- REQ:** *Request*
- RISC:** *Reduced Instruction Set Computer*
- RTS:** *Request To Send*
- RWM:** *Read Writable Memory*
- S/H:** *Sample and Hold*
- SAM:** *Secuential Acces Memory*
- SCSI:** *Small Computer System Interface*
- SDRAM:** *Synchronous DRAM*
- SGRAM:** *Synchronous Graphics RAM*
- SISD:** *Single Instruction Single Data*
- SIMD:** *Single Instruction Multiple Data*
- SLDRAM:** *Synchronous Link DRAM*
- SPLD:** *Simple PLD*
- SoC:** *System on Chip*
- SOH:** *Start Of Head*
- SoPC:** *System on a Programmable Chip*
- SRAM:** *Static RAM*
- SSYN:** *Slave SYNchronitation*
- SW:** *Software*

TN: *Twisted Nematic*

USART: *Universal Synchronous Asynchronous Receiver Transmitter*

USB: *Universal Serial Bus*

VHDL: *Very High Speed Hardware Description Language* (Lenguaje de Descripción Hardware)

VLSI: *Very Large Scale Integration*

VME: *VERSAmodule Eurocard o Versa Module Europa*

VRAM: Video DRAM

ZBT: *Zero Bus Turnaround SRAM*

BIBLIOGRAFÍA - Observaciones

Más libros, más libres. (Enrique Tierno Galván)

La Bibliografía disponible, para asignaturas que traten temas relacionados con el diseño digital, es muy extensa. Debido a ello, el objeto de este apartado no es ofrecer una recopilación enciclopédica de títulos, sino más bien al contrario, nombrar y comentar exclusivamente los textos que han sido seleccionados para la confección del programa en sus diversos contenidos. Como ya se comentará, unos títulos han sido más determinantes que otros en la elaboración de esta extensa guía, pero no por ello deben dejar de consultarse los restantes porque siempre nos aportarán una visión o complemento a lo expuesto y enriquecerán nuestro bagaje técnico.

La selección se ha realizado atendiendo a diversos criterios, a saber, la adecuación de los contenidos del texto al programa propuesto, la adecuación del nivel de complejidad presentado por los contenidos, la disponibilidad de los volúmenes (algunos solo en biblioteca), la calidad pedagógica de la exposición de la materia y la posibilidad que ofrecen algunos textos para “profundizar y ampliar conocimientos”.

En varias de estas referencias se ha valorado también la manera de tratar los temas por parte de los autores, el enfoque y lenguaje utilizado motivarán que el alumnado se acerque con menos temor que el que inspiran normalmente los libros especializados. Además, la bibliografía está organizada según prioridad.

1. Materiales de uso obligatorio

- G. Bosque, P. Fernandez, “Principios de Diseño de Sistemas Digitales - Guía Práctica para Alumnos de Primer Curso de Grado en Ingeniería Informática de Gestión y Sistemas de Información”. Ed. UPV-EHU. 2014 [1]. Guía-libro adaptada al temario aportando una visión clara y extensa de los temas tratados.
- O. Arbelaitz, O. Arregi, A. Aruabarrena, I. Etxeberria, A. Ibarra y T. Ruiz, “Principios de Diseño de Sistemas Digitales: Conceptos básicos y ejemplos”. Ed. Prentice-Hall [2]. Libro básico para esta asignatura, trata todos los temas con rigor, profusión y amenidad.

2. Bibliografía básica

- T.L. Floyd, “Fundamentos de Sistemas Digitales”. Ed. Prentice Hall. 2000. [3].
- M. Morris Mano, “Diseño Digital”. Ed. Prentice Hall. 2003. [4].
- D. D. Gajski, “Principios de Diseño Digital”. Ed. Prentice Hall. 1997. [5].
- J. P. Hayes, “Introducción al Diseño Lógico Digital”. Ed. Addison-Wesley Iberoamericana. 1996. [6].
- A. Lloris, A. Prieto, L. Parrilla, “Sistemas Digitales”. Ed. McGraw-Hill. 2003. [7].

3. Bibliografía de profundización

- J.P. Uyemura, “Diseño de Sistemas Digitales. Un Enfoque Integrado”. Ed. Thomson. 2000. [8].
- D.A. Patterson, J. L. Hennessy, “Organización y Diseño de Computadores”. Ed. McGraw-Hill, 1994. [9].
- M. Ercegovac, T. Lang, J. H. Moreno, “Introduction to Digital Systems”. Ed. John Wiley and Sons, 1999. [10].
- P.J. Ashenden, “Digital Design. An Embedded Systems Approach Using VHDL”. Ed. Morgan Kaufmann. 2008. [11].
- J.O. Hamblen, T.S.Hall, M.D. Furman, “Rapid Prototyping of Digital Systems. SoPC Edition”. Ed. Springer. 2008. [12].
- M. Morris Mano, C.R. Kime. “Fundamentos de Diseño Lógico y Computadoras”. Ed. Prentice-Hall. 3^a Edición. 2005 [13]. Énfasis en el diseño Digital de un Computador y el tratamiento de las instrucciones a nivel máquina. Excelente consulta para la memoria Virtual y Caché.

Capítulo 1

REPRESENTACIÓN DE LA INFORMACIÓN

La mente es como un paracaídas, si no lo abres, no sirve para nada (Albert Einstein)

RESUMEN. En los sistemas digitales es necesario representar la información que procesa el sistema. Para ello se utilizan diferentes métodos, todos basados en el sistema de numeración de base dos, conocido como sistema binario.

1.1. INTRODUCCIÓN

Un sistema es un conjunto de elementos organizados que interactúan entre sí para conseguir un objetivo. En el caso de los sistemas digitales, los elementos que forman el sistema son de *tecnología electrónica*, es decir, su funcionamiento está basado en los fenómenos electromagnéticos. Concretamente, en los sistemas digitales usamos los fenómenos electromagnéticos pero de forma *discreta* con sólo dos valores.

La tecnología electrónica está muy extendida (ver Fig. 1) en multitud de aplicaciones: los motores eléctricos, los elementos de iluminación, los altavoces, los mandos a distancia, los reproductores MP3 y por supuesto, los ordenadores. Todas estas aplicaciones incluyen sistemas que utilizan elementos de tecnología electrónica.

Los fenómenos electromagnéticos incluyen todas las posibles interacciones (fuerzas) que aparecen entre objetos que contienen cargas o corrientes eléctricas, y se estudian mediante los campos electromagnéticos. Los campos electromagnéticos están alrededor nuestro y hemos aprendido a detectarlos y cuantificarlos, pero pueden aparecer con cualquier valor (como la cantidad de agua que puede entrar en una botella, o la longitud de un cordel). Esta propiedad hace que llamemos *continuos* a los campos electromagnéticos, y a la tecnología electrónica que aprovecha todos los posibles valores en que aparecen las magnitudes asociadas a estos campos, *Electrónica analógica*. Muchos de los

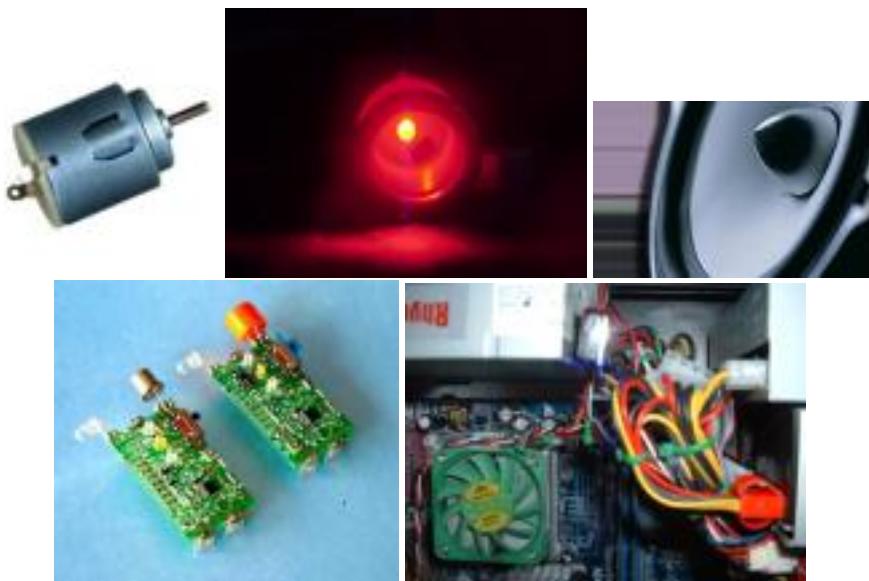


FIGURA 1. Elementos de tecnología electrónica

elementos de tecnología electrónica, antes citados, forman parte de este campo, como los altavoces o los motores.

Sin embargo, en los últimos cincuenta años, se ha desarrollado una tecnología electrónica diferente, que utiliza las magnitudes electromagnéticas (tensión, intensidad) sólo en dos valores. Este tipo de tecnología es la que llamamos *Electrónica digital*. El hecho de usar sólo dos valores de las magnitudes electromagnéticas nos permite simplificar el diseño de los elementos electrónicos y, por tanto, hacerlos más pequeños, rápido y eficientes. La rápida evolución en la tecnología digital han supuesto la sustitución de muchos sistemas analógicos por otros digitales, con lo que hoy en día, la mayoría de los sistemas electrónicos son en su mayor parte sistemas digitales.

Pero, ¿cómo se pueden usar sólo dos valores de una magnitud que es continua? Para entenderlo, vamos a profundizar en la diferencia entre continuo y discreto. El volumen del sonido emitido por un altavoz (relacionado con la tensión y la intensidad eléctrica del altavoz) es una magnitud continua, que puede aparecer en una amplia gama de posibles valores (Ver Fig. 2). En general, podemos decir que una función del tiempo es continua cuando, para un intervalo de tiempo determinado, aparecen los infinitos valores que están contenidos en el intervalo de valores posibles de la función. Recordad que dentro de un intervalo de la recta real, siempre hay un número infinito de números.

La mayoría de las magnitudes que nos rodean son continuas, las magnitudes eléctricas también lo son. Pero, ¿cómo es una función discreta? Si le damos la

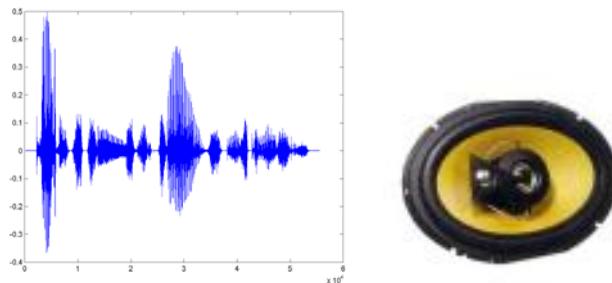


FIGURA 2. Función continua

vuelta a la definición de función continua, podemos decir que una función es discreta cuando, a una serie de valores concretos de tiempo, les corresponde una serie de valores de la función. Podemos ver en la Figura 3, cómo la función continua relaciona todos los posibles valores de tiempo entre 0 y 30 s con todos los valores de la función, desde 0 hasta 800. La función discreta relaciona sólo una serie de instantes del tiempo 0, 10, 25, 50 s... con una serie de valores de la función: 200, 400, 600.

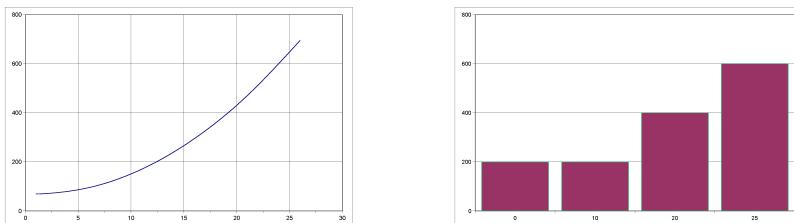


FIGURA 3. Función continua y función discreta

¿Qué relación hay entre las dos funciones de la Figura 3? En ambos casos se describe la relación entre el tiempo y una magnitud. La función discreta la hemos construido a partir de la función continua, empleando un método. Dividimos en subintervalos el intervalo de tiempo para el que está definida la función continua, y tomamos un sólo instante de cada uno. El valor de la función continua para ese instante de tiempo, será el valor de la función discreta. Hemos convertido una función que relaciona un número infinito de instantes de tiempo en una relación entre un número determinado de instantes de tiempo y sus correspondientes valores concretos de función. Esta transformación se llama discretizar, y es lo que aplicaremos a las magnitudes eléctricas analógicas para convertirlas en magnitudes de electrónica digital.

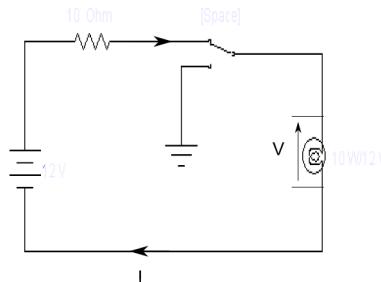


FIGURA 4. Un circuito con un interruptor deja dos valores de tensión e intensidad

En la Figura 4 aparece la representación gráfica de un circuito eléctrico, lo que llamamos el diagrama del circuito. Este circuito tiene un interruptor de tres vías, lo que quiere decir que permite conectar eléctricamente (permitir el paso de la corriente eléctrica) el conductor que viene de la bombilla, o bien con el conductor que viene de la fuente de tensión, o bien con el conductor de tierra. De este modo, el circuito tiene dos posibilidades de funcionamiento, con el interruptor conectando la bombilla a la fuente de tensión, o conectando la bombilla al punto de tierra (0 V).

¿Qué podemos decir entonces respecto a la tensión que aparece en la bombilla? Desde luego, si el interruptor está conectando la bombilla con la tierra, la tensión V en la bombilla serán 0V. En este caso, no circulará corriente por el circuito ($I=0$) porque la tensión es cero. La otra posibilidad es que el interruptor conecte la bombilla a la fuente de tensión. En este caso, la fuente de tensión alimenta la corriente a través de la bombilla, por lo que tendremos intensidad y tensión no nulas en la bombilla (la bombilla se encenderá). El interruptor reduce la tensión/intensidad a *sólo dos valores: alto/bajo*, en función de la posición del interruptor. Hemos conseguido un circuito electrónico discreto de sólo dos valores y esto es lo que llamamos un circuito electrónico digital.

En general, y dependiendo de la configuración del circuito y de la tecnología empleada para la comutación, los valores de tensión estarán comprendidos en un intervalo de tensiones, en vez de ser un sólo valor discreto. Por tanto, cualquier valor que esté dentro de ese intervalo se considerará como perteneciente al intervalo, ya sea el de tensión alta como el de tensión baja. En la Figura 5, podemos ver una representación aproximada de los intervalos de tensión que se aceptan habitualmente para los valores alto y bajo de tensión (de manera similar se definen los valores alto y bajo de intensidad). Cualquier circuito que produzca valores de tensión que no se correspondan con estos dos intervalos funcionará incorrectamente, pues no será reconocido como valor digital posible.

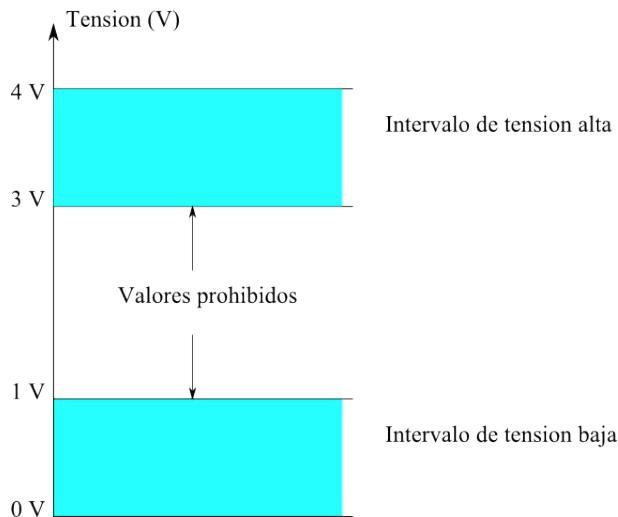


FIGURA 5. Valores de tensión en Electrónica Digital

Puesto que todas las funciones de tensión de la Electrónica Digital sólo pueden ser del intervalo alto (H o *high*) o del intervalo bajo (L o *low*), parece obvio representarlas mediante dos números, que son el 0 y el 1. Todas las funciones que aparezcan en los sistemas digitales tendrán como valor el 0 y el 1. Esto parece práctico para algunas aplicaciones, como apagar o encender la calefacción de una habitación o abrir o cerrar la puerta de un garaje, pero ¿cómo representamos acciones más complejas como presentar el resultado de una operación de una calculadora o producir el sonido del timbre de un teléfono móvil?

Existen métodos para representar números sólo basándonos en el cero y el uno y, para entenderlos, tendremos que entender mejor nuestro método para representar números: el *sistema de numeración posicional*.

1.2. SISTEMAS DE NUMERACIÓN POSICIONAL

En los sistemas digitales, la información está codificada en ceros y unos, ya que utilizan tensiones digitales que sólo pueden ser de valor alto y bajo. Para entender cómo se hace, empezaremos por estudiar la representación numérica. Y para ello vamos a utilizar en mismo método que utilizamos habitualmente nosotros. Este método se llama sistema de numeración posicional, ya que el principio de este sistema es que cada cifra que escribimos tiene un valor en función de la posición que ocupa en el número.

Pongamos, por ejemplo, que queremos escribir el número dos mil trece (ver Figura 6). Este número lo descomponemos en cifras, y cada cifra se escribe en una posición determinada. Primero escribimos la cifra de las unidades, en

2 0 1 3

millar centena decena unidad

$$2013 = 2 \cdot 1000 + 1 \cdot 10 + 3 \cdot 1$$

FIGURA 6. Sistema de numeración posicional

este caso 3. A continuación, escribimos la cifra de las decenas (1) y luego la cifra de las centenas, que es 0 porque no hay ninguna centena en este número. Finalmente, escribimos la cifra correspondiente a los millares, que es 2. Este proceso lo hemos realizado sin pensar mucho y lo hacemos siempre que escribimos un número, pero fijémonos en que estamos descomponiendo la cantidad que queremos expresar en una serie de múltiplos de unidades, decenas, centenas, etc. Pero, ¿porqué unidades, decenas, centenas...? Se trata de potencias de 10, con exponentes crecientes a medida que nos colocamos es posiciones más altas dentro del número. Vamos a describir este proceso mediante una expresión general:

$$(1.2.1) \quad N = \sum_{i=0}^{n-1} d_i \cdot 10^i$$

donde d_i son cada una de las cifras que escribimos en las diferentes posiciones, y la i representa la posición en la que está la cifra. El número que multiplica a cada cifra, que es una potencia de 10, se llama peso y es diferente para cada posición. Como la posición empieza en cero, empezamos buscando el número de unidades ($10^0 = 1$), luego el número de decenas ($10^1 = 10$), las centenas... etc. Como los d_i son todos los posibles valores entre 0 y 9, en cuanto la cantidad a expresar no cabe en la posición de las unidades, basta con pasar a la siguiente, la de las decenas, que empieza justo en la siguiente al 9, que es el 10. Esta relación se sigue manteniendo entre decenas y centenas, y así sucesivamente, de modo que N puede ser cualquier valor entero, siempre que podamos escribir las suficientes posiciones.

Este sistema funciona porque las cifras que se pueden expresar en cada posición son diez (de 0 a 9) y porque la base del peso correspondiente a cada posición también es 10. ¿Podemos generalizar este sistema a otras bases? La respuesta es que sí, siempre que tengamos en cuenta que el número de las diferentes posibilidades de cada cifra sea igual a la base de los pesos de las posiciones de las cifras.

El sistema de numeración posicional que utilizamos, el de base 10, encaja en una definición más general que nos va a permitir utilizarlo con diferente número de cifras posibles en cada posición:

$$(1.2.2) \quad N = \sum_{i=0}^{n-1} d_i \cdot b^i$$

donde b es la base del peso de la cifra en la posición i . Por tanto, es también el número de diferentes valores que puede tener d_i . Hasta ahora, siempre hemos considerado la base 10, pero en la electrónica digital nos interesa que el número de diferentes valores de d_i sea sólo dos, para que podamos utilizar funciones de tensión digital para representar las cifras de cualquier número. El sistema de numeración posicional de base 2 se llama *sistema binario*.

1.2.1. Sistema binario, octal, hexadecimal.

En electrónica digital representamos las señales mediante dos valores. El sistema de numeración de base 2, el sistema binario, sólo usa dos cifras. Por tanto, usaremos este sistema para representar información numérica en los sistemas de electrónica digital.

TABLA 1. Sistemas de numeración en diferentes bases

	b	d_i	N
<i>Binario</i>	2	0-1	11111011101
<i>Octal</i>	8	0-7	3735
<i>Decimal</i>	10	0-9	2013
<i>Hexadecimal</i>	16	0-F	7DD

En la Tabla 1 aparece en número dos mil trece escrito en diferentes bases, entre ellas la base 2. Es fácil observar cómo el número de posiciones que necesitamos para escribir el mismo número aumenta a medida que se reduce la base. Las cifras en base 2 se llaman *bits* y hacen falta 11 bits para expresar un número que en base 10 se puede escribir con cuatro cifras. Esto hace engorroso y difícil manejar los números binarios, ya que cuanto mayor es el número de cifras, más difícil se hace recordarlos, por lo que los primeros ingenieros de sistemas digitales propusieron un método para escribir los números binarios más rápido: usar la base 8, conocido como *sistema octal*. A medida que se hizo mayor la capacidad de tratamiento de datos de los sistemas digitales, se hizo necesario representar números más grandes, por lo que se pasó al sistema de base 16 o *sistema hexadecimal*. La conversión del sistema binario al octal es sencilla (ver Figura 7), basta con escribir una cifra octal por cada tres bits,

lo mismo que la conversión de binario a hexadecimal, se trata de escribir una cifra hexadecimal por cada cuatro bits del número binario original.

Hexadecimal	7	B	A	3
Binario	0111	1011	1010	0011
Octal	7	5	6	4

FIGURA 7. Binario, octal y hexadecimal

El sistema posicional es también el que empleamos para escribir números fraccionarios. La parte menor que uno es la que se escribe en las posiciones a la derecha de la coma decimal:

$$14.751 = 1 \cdot 10 + 4 \cdot 1 + 7 \cdot 0.1 + 5 \cdot 0.01 + 1 \cdot 0.001$$

El peso de cada cifra es también menor que uno, pero observemos que la primera posición a la izquierda de la coma corresponde a las décimas, la segunda a las centésimas, la tercera a las milésimas...etc. Es decir, de nuevo son potencias de la base 10, pero ahora los exponentes son números negativos crecientes: $0.1 = 10^{-1}$, $0.01 = 10^{-2}$... De este modo se puede generalizar la expresión del sistema posicional a cualquier número real:

$$(1.2.3) \quad N = \sum_{i=0}^{n-1} d_i \cdot b^i + \sum_{i=-1}^{-k} d_i \cdot b^i$$

donde podemos observar que hay n cifras a la izquierda de la coma (de 0 a $n-1$, con exponentes positivos) y k cifras a la derecha de la coma (de -1 a $-k$, con exponentes negativos). Si la base es 2 obtenemos el sistema binario para todos los números, enteros y fraccionarios.

$$\begin{aligned} 1110.11 &= 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} = \\ &= 8 + 4 + 2 + 0 + \frac{1}{2} + \frac{1}{4} = 14.75 \end{aligned}$$

1.2.2. Conversiones entre sistemas de numeración:

Una vez hemos introducido el concepto de base del sistema de numeración, está claro que es posible escribir números en cualquier base, siempre que siga siendo igual la cantidad de números posibles en cada cifra y la base. Cuando se utilizan números en diferentes bases, es habitual distinguir en qué

base esta escrito cada número mediante un subíndice que indica la base. Pero, ¿cómo sabemos qué número binario representa una cantidad en base 10 y viceversa? ¿Cómo se puede transformar un número de una base a otra?

Para ello, vamos a examinar la expresión 1.2.3 genérica de un número en el sistema de numeración posicional, primero sólo de la parte entera:

$$(1.2.4) \quad N = \sum_{i=0}^{n-1} d_i \cdot b^i = d_{n-1} \cdot b^{n-1} + d_{n-2} \cdot b^{n-2} + \dots + d_1 \cdot b^1 + d_0 \cdot b^0$$

Si dividimos ambos términos de esta expresión por la base, obtenemos:

$$\begin{aligned} \frac{N}{b} &= d_{n-1} \cdot \frac{b^{n-1}}{b} + d_{n-2} \cdot \frac{b^{n-2}}{b} + \dots + d_1 \cdot \frac{b^1}{b} + d_0 \cdot \frac{b^0}{b} = \\ &= d_{n-1} \cdot b^{n-2} + d_{n-2} \cdot b^{n-1} + \dots + d_1 \cdot b^0 + \frac{d_0}{b} \end{aligned}$$

De todos los sumandos de la parte derecha de esta expresión, sólo uno es menor que uno: el último, ya que cualquier d_i es siempre menor que b . Eso quiere decir que el resultado de la división de N entre b es siempre un número entero más un resto que es la cifra más significativa d_0 del número N en la base b . Repitiendo la división por b con el cociente que hemos obtenido, podemos obtener la cifra d_1 :

$$\begin{aligned} &\frac{d_{n-1} \cdot b^{n-2} + d_{n-2} \cdot b^{n-1} + \dots + d_1 \cdot b^0}{b} = \\ &= d_{n-1} \cdot b^{n-3} + d_{n-2} \cdot b^{n-2} + \dots + d_2 \cdot b^0 + \frac{d_1}{b}, \end{aligned}$$

ya que será el resto de la división realizada. Si repetimos el proceso, cuando el cociente entero sea cero, el resto será d_{n-1} , la cifra más significativa del número N en la base b .

La parte fraccionaria requiere un método diferente, ya que los exponentes negativos no decrecen al dividir por la base. Volvamos a la expresión 1.2.3, pero ahora tomemos sólo la parte menor que uno:

$$(1.2.5) \quad N = \sum_{i=-1}^{-k} d_i \cdot b^i = d_{-1} \cdot b^{-1} + d_{-2} \cdot b^{-2} + \dots + d_{-k} \cdot b^{-k}$$

Si multiplicamos ambos términos de esta expresión por la base, tenemos:

$$\begin{aligned} N \cdot b &= d_{-1} \cdot b^{-1} \cdot b + d_{-2} \cdot b^{-2} \cdot b + \dots + d_{-k} \cdot b^{-k} \cdot b = \\ &= d_{-1} \cdot b^0 + d_{-2} \cdot b^{-1} + \dots + d_{-k} \cdot b^{-(k-1)} \end{aligned}$$

Ahora, de todos los sumandos de la parte derecha de la expresión, sólo uno es mayor que uno: $d_{-k} \cdot b^0 = d_{-k}$. Es decir, la parte entera del resultado de multiplicar un número (menor que uno) por la base, es la cifra más significativa d_{-k} del número en la nueva base. Repitiendo este proceso con la parte fraccionaria del resultado de la multiplicación, iremos obteniendo cifras del número en la nueva base, hasta que nos dé un número entero, que será la cifra menos significativa d_{-1} del número N en la base b .

Tanto en el caso de la parte entera como de la parte fraccionaria, este proceso nos lleva a un número de cifras que no está definido de antemano, dependerá del número de veces que debemos repetir la multiplicación o la división por b . Para la parte mayor que uno, el proceso termina cuando el cociente de dividir por la base es cero. El número de veces que hemos tenido que repetir la división para llegar a este resultado será n , el número de cifras a la izquierda de la coma. Para la parte menor que uno, el proceso termina cuando el resultado de multiplicar por la base es un número entero, y el número de veces que hemos repetido la multiplicación es k , el número de cifras a la derecha de la coma.

1.2.3. Precisión finita.

En el papel es posible representar cualquier número mediante el sistema de numeración posicional, siempre que nos quede espacio para escribirlo (no olvidemos que existen números que *no pueden ser escritos*, porque el número de sus cifras decimales es infinito: $\frac{1}{3}, \pi\dots$). Sin embargo, en los sistemas digitales el número de cifras es fijo, pues las cifras se implementan mediante señales de tensión eléctrica que aparecen en determinados puntos de los circuitos digitales. No podemos cambiar el número de esas señales sin cambiar el diseño del circuito, por lo que es un número que habrá que fijar según cuál sea la aplicación del circuito y la magnitud de los números que queramos representar.

La aritmética en la que el número de cifras está definido se llama aritmética de *precisión finita* o de *coma fija*. Puesto que el número de cifras está fijado de antemano y no puede cambiarse, no será posible representar todos los números, puesto que en el proceso de calcular las cifras binarias de cada número, los valores de n y k no están definidos y dependen de cuál sea el número N que se quiere representar. Es importante, por tanto, definir cuáles son los números más grandes y más pequeños que podemos representar para un número fijo de cifras decimales y enteras. En base 2, con n cifras a la izquierda de la coma y k cifras a la derecha de la coma, tenemos el número más grande cuando todas las d_i , tanto a la izquierda como a la derecha de la coma son 1. En ese caso:

$$N_{\max} = 2^n - 1 + (1 - 2^{-k})$$

El número más pequeño (aparte del cero, que serán el que tiene 0 en todas sus cifras) que podemos representar será el que tenga todas sus cifras igual a 0

salvo la menos significativa, la de la posición $-k$, que será un 1. En ese caso, el número mínimo es:

$$N_{\min} = 2^{-k}$$

Todos los números que sean más grandes que N_{\max} o más pequeños (salvo el cero) que N_{\min} están fuera de la precisión que corresponde a los números binarios de $n + k$ bits, y no pueden ser representados en este sistema. La aparición de estos números como producto de alguna operación provocará un error en el sistema digital que se llama *error de desbordamiento (overflow)*. Estudiaremos más adelante los diferentes casos de desbordamiento y cómo detectarlos.

1.2.4. Sistema decimal codificado en binario BCD.

El sistema binario es el más adecuado para los sistemas digitales, pero obliga a un proceso de conversión a la base 10 cuando se trata de presentar los resultados al usuario del sistema. En este caso, hay sistemas que permiten representar números en base 10 (los displays y las pantallas representan números en base 10), pero existe un método intermedio que es muy útil para realizar el cambio de base.

CUADRO 2. Sistema decimal codificado en binario (BCD)

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
Código no usados	1010 1011 1100 1101 1110 1111

Se trata del *sistema decimal codificado en binario (Binary coded decimal BCD)*. Este sistema (ver 2) es un código binario en el que se representan los números decimales cifra por cifra. Como para representar la cifra decimal más

alta hacen falta cuatro bits, a cada cifra de un número decimal le corresponden cuatro bits que representan en binario la cifra decimal.

Como las seis cifras binarias 1010 a 1111 no tienen una cifra decimal correspondiente, no se usan. Por tanto existen combinaciones binarias que no se corresponden con el código BCD. Esto obliga a realizar las operaciones aritméticas con reglas especiales: si el resultado de sumar cada grupo de cuatro bits supera el 9, se “saltan” los seis números no usados sumando 6 a la cifra de cuatro bits que ha resultado inválida.

Las expresiones de la precisión en función del número de bits tampoco son válidas. El siguiente número escrito en BCD:

$$396_{10} = 0011\ 1001\ 0110_{BCD}$$

requiere 12 cifras binarias, pero serían suficiente con nueve bits, ya que $2^8 - 1 \leq 396 \leq 2^9 - 1$; es decir, 396 es más alto que el $N_{máx}$ que se puede representar con ocho bits, pero más pequeño que el que se puede representar en binario natural con 9 bits. Esto indica que utilizar BDC requiere más capacidad de almacenamiento que el binario natural y esta desventaja ha llevado a que el código BCD sólo se utilice en los sistemas de presentación para el usuario.

1.3. REPRESENTACIÓN DE NÚMEROS CON SIGNO

El sistema que hemos descrito hasta ahora es útil para representar números reales positivos, pero ¿qué hacemos con los números negativos? Con n bits se pueden representar 2^n números enteros (el cero incluido), ya que éste es el número de combinaciones de n elementos con dos posibles valores cada uno. Para poder representar números enteros positivos y negativos con n bits, tendremos que dividir el conjunto de todos los números posibles en dos mitades. 2^{n-1} números positivos y otros 2^{n-1} números negativos. Existen diversas maneras de realizar esa división, por lo que cada una tendrá diferentes propiedades en la implementación de operaciones aritméticas.

1.3.1. Magnitud con signo.

El sistema de magnitud con signo es el más sencillo (ver Figura 8 para $n = 4$ bits). Consiste en utilizar la cifra de la posición $n - 1$ para indicar si el número formado por el resto de las cifras es negativo o positivo. El valor 1 en la cifra más significativa indica que el número es negativo y el cero indica que es positivo, mientras que las $n - 1$ cifras restantes representan el valor absoluto mediante el sistema binario natural.

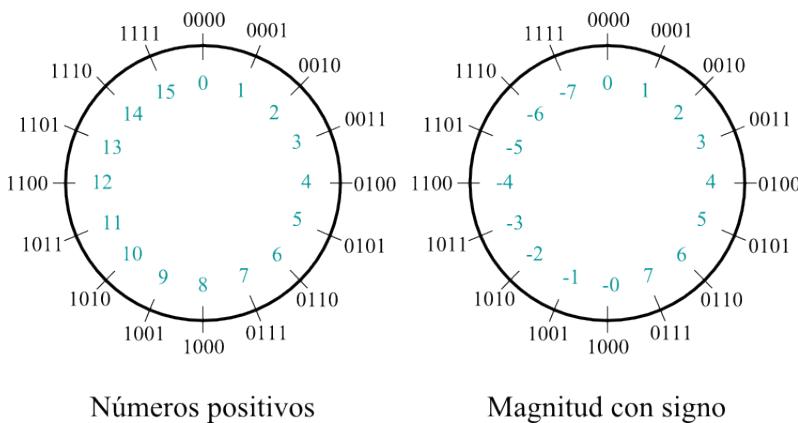


FIGURA 8. Números positivos y sistema de magnitud con signo

Este sistema es fácil de interpretar y de implementar, pero tiene varias desventajas. El cero tiene dos símbolos para representar la misma magnitud, dado que sea cual sea el signo, si el valor absoluto es cero, la cantidad es la misma. Por otro lado, en el conjunto de todos los números representados con n bits, los números positivos están en posiciones inferiores a los negativos, lo que conlleva dificultades a la hora de implementar la comparación de dos números.

Las operaciones aritméticas no pueden realizarse del mismo modo que con los números decimales, ya que no se utiliza el mismo sistema de numeración. Su expresión general sería:

$$(1.3.1) \quad N = (-1)^{d_{n-1}} \cdot \sum_{i=0}^{n-2} d_i \cdot 2^i$$

en donde podemos ver que la cifra $n-1$ no tiene peso ni sentido dentro del sistema de numeración posicional. Por esta razón no podemos realizar las sumas de manera directa, sino que habrá que considerar el signo para decidir si la operación a realizar es una suma o una resta. La implementación de suma y resta en un sistema digital implica un diseño diferente, por lo que habrá que utilizar circuitos diferentes según se trate de una u otra operación. La necesidad de un hardware más complejo ha arrinconado a este sistema en la mayoría de los sistemas digitales.

Si consideramos un número en el sistema de magnitud con signo, tendremos que el número positivo más alto posible con n bits es:

$$N_{\max} = 2^{n-1} - 1$$

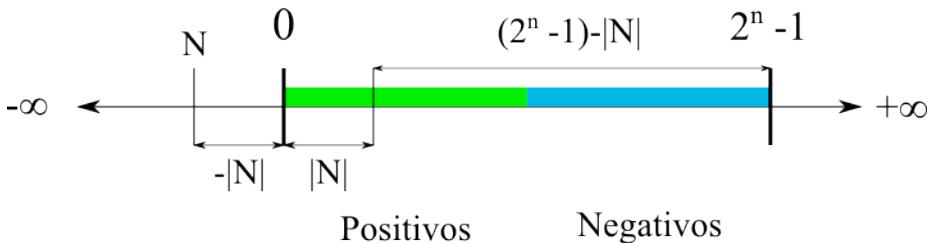


FIGURA 9. Complemento a uno

mientras que el número negativo de mayor valor absoluto será:

$$N_{\min} = -(2^{n-1} - 1)$$

1.3.2. Complemento a uno y complemento a dos.

Como acabamos de ver, el sistema de magnitud con signo nos obliga a implementar sistemas separados para la suma y para la resta. Sería deseable que el sistema que realiza la suma fuese capaz de realizar también la resta, teniendo en cuenta que es la misma operación si consideramos los signos de cada operando. Existen sistemas de representación de números negativos que incluyen esta posibilidad.

Consideremos antes la naturaleza de la representación de magnitud con signo. ¿Qué significa que una cantidad sea -3? Pues significa que esa cantidad está tres unidades por debajo del cero, es decir, que nos faltan tres unidades para llegar a cero. Si consideramos los números de esta manera, el signo sólo indica si se está por delante o por detrás del cero en la recta real. Podríamos representar la misma cantidad expresándola con referencia a otro punto de la recta real, pero parece lógico que sea el cero, ya que es el único punto definido en el conjunto de los números reales. Sin embargo, en nuestros sistemas digitales utilizaremos la precisión finita, es decir, el conjunto de los números que podemos utilizar no es ilimitado. Por ello, aparece otro número “especial” que puede servirnos como referencia para indicar dónde está una cantidad.

Igual que podemos representar un número (ver Figura 9) indicando cuántas unidades nos faltan para llegar al cero (en el sistema de magnitud con signo), podemos expresar ese mismo número representando cuántas unidades nos faltan hasta el número más alto $N_{\max} = 2^n - 1$ que se puede representar con la cantidad n de cifras de que disponemos. Este sistema, en el caso de los números binarios, se llama complemento a uno, y se define así:

$$(1.3.2) \quad N^{(1)} = (2^n - 1) - |N|$$

es decir, expresamos el número negativo N restando a $2^n - 1$ el valor absoluto del número N . Este método tiene como ventaja principal que es posible implementar suma y resta con un único sistema, como veremos más adelante. Para los números positivos no hay que hacer ninguna operación, ya que seguimos manteniendo la referencia en el cero. Es decir, este sistema expresa de diferente manera los números positivos y los negativos.

Sin embargo, en la definición de este sistema hay una resta, por tanto ¿dónde está la ventaja de operar sin restas si hay que hacer una resta antes de operar? Pues en que esta resta de $2^n - 1$ tiene una propiedad especial en binario que la va a hacer innecesaria. Recordad que el número $2^n - 1$ es el número máximo que se puede escribir con n bits y por lo tanto, todas sus cifras son 1 (la cifra más alta en binario). Por tanto, para todas las cifras de la resta $(2^n - 1) - |N|$ sólo hay dos posibilidades:

$$1 - 1 = 0$$

$$1 - 0 = 1$$

es decir, el resultado de esta resta es el número $|N|$ cambiando cada cero por un uno y cada uno por un cero. Es interesante observar (ver Figura 10) que todos los números negativos tienen un 1 en la cifra más significativa, ya que todos los números menores que 2^{n-1} tienen un cero en esa posición.

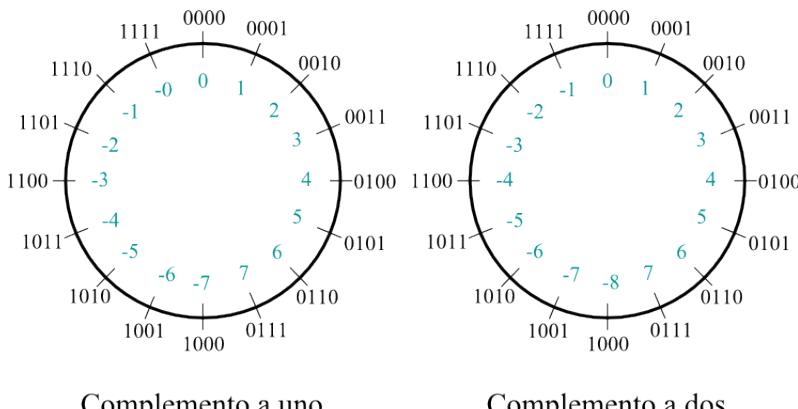


FIGURA 10. Complemento a uno y complemento a dos

Para conocer el valor absoluto de un número negativo en complemento a uno, basta con obtener su complemento a uno, ya que:

$$\left(N^{(1)}\right)^{(1)} = ((2^n - 1) - |N|)^{(1)} = (2^n - 1) - (2^n - 1 - |N|) = |N|,$$

así que el valor absoluto de un número negativo en complemento a uno se obtiene sustituyendo cada 0 por un 1 y cada 1 por un 0.

La expresión de un número en complemento a uno es:

$$(1.3.3) \quad N = -(2^{n-1} - 1) \cdot d_{n-1} + \sum_{i=0}^{n-2} d_i \cdot 2^i$$

En este sistema, igual que en el de magnitud con signo, el número positivo más alto posible con n bits es:

$$N_{\max} = 2^{n-1} - 1$$

mientras que el número negativo de mayor valor absoluto será:

$$N_{\min} = -(2^{n-1} - 1)$$

Del mismo modo, podemos observar que hay dos números que representan el cero: el cero positivo, con todas las cifras cero y el complemento a uno de cero, es decir todas las cifras igual a uno.

Una solución para este problema es el sistema de complemento a dos. En este sistema elegimos como referencia para los números negativos 2^n , es decir, el primero que no pertenece a los números que se pueden representar con n bits. Los números negativos en este sistema se definen así:

$$(1.3.4) \quad N^{(2)} = 2^n - |N|$$

y los positivos se quedan, igual que antes, referenciados al cero. Esta expresión también puede escribirse así:

$$(1.3.5) \quad N^{(2)} = (2^n - 1) - |N| + 1 = N^{(1)} + 1$$

donde se puede observar que el complemento a dos de un número negativo es igual a su complemento a uno más uno.

Como nuestra referencia está fuera de nuestro intervalo posible de números, ya no hay un valor para definir el cero negativo y, por tanto, sólo se asigna un código al cero (el cero positivo). El hacer desaparecer el cero negativo provoca que uno de los números no tenga significado, por lo que se le asigna un número negativo más: -2^{n-1} .

La expresión general de un número en complemento a dos es:

$$(1.3.6) \quad N = -2^{n-1} \cdot d_{n-1} + \sum_{i=0}^{n-2} d_i \cdot 2^i$$

Así pues, en el sistema de complemento a dos, el número positivo más alto posible con n bits es:

$$N_{\max} = 2^{n-1} - 1,$$

mientras que el número negativo de mayor valor absoluto será:

$$N_{\min} = -2^{n-1}$$

Igual que con el complemento a uno, el valor absoluto de un número negativo en complemento a dos es el complemento a dos de ese número, por lo que sólo hay que invertir ceros por unos y viceversa y sumarle uno.

1.3.3. Resta binaria.

La razón principal para introducir los sistemas de complemento a uno y a dos es su comportamiento frente a la resta. En ambos sistemas se puede incluir a la resta como un caso particular de la suma cuando uno de los sumandos es negativo. Así, el diseño de un sistema digital que realice operaciones aritméticas es más sencillo, ya que cualquier operación puede ser descompuesta en serie de sumas y restas.

En complemento a uno, la resta de A menos B (es decir, la suma de un número positivo A más un número negativo B), se hace sumando a A el complemento a uno de B ($B^{(1)}$), ya que es un número negativo:

$$(1.3.7) \quad A - B = A + B^{(1)} = A + (2^n - 1 - |B|) = 2^n + (A - |B|) - 1$$

Si la resta de A menos B es positiva, este valor queda fuera del rango de los números que se pueden expresar en n bits, pero si descartamos 2^n (que como tendría que aparecer en la cifra $n+1$ aparece como acarreo de la última cifra), el resultado queda correcto salvo una unidad. Es decir, sumando el acarreo que sale de la última cifra al resultado de la suma, ésta es correcta (ver Figura 11).

Si la resta es negativa, el resultado es menor que $2^n - 1$ y se corresponde con el complemento a uno de restar A del valor absoluto de B :

Sumandos	0	0	1	1
	+ 0	+ 1	+ 0	+ 1
Suma	0	1	1	0
Acarreo	0	0	0	1

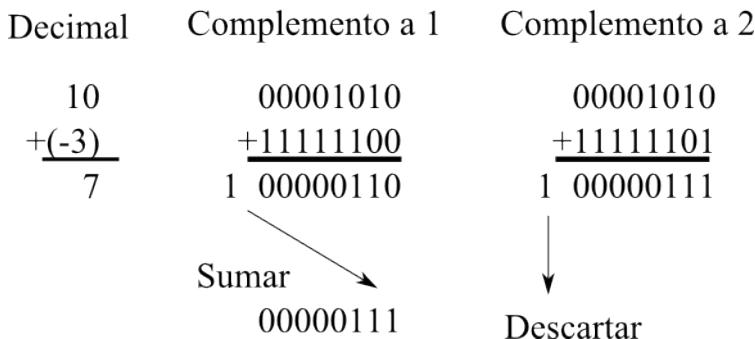


FIGURA 11. Suma binaria con números negativos

(1.3.8)

$$A - B = A + B^{(1)} = A + (2^n - 1 - |B|) = 2^n - 1 - (|B| - A) = (|B| - A)^{(1)}$$

es decir, es el resultado correcto sin modificación alguna. Para observar el valor absoluto del resultado, basta con hacer el complemento a uno de éste.

En este aspecto, el resultado de usar el complemento a dos es todavía más ventajoso:

$$(1.3.9) \quad A - B = A + B^{(2)} = A + (2^n - |B|) = 2^n + (A - |B|)$$

Si la resta de A menos B es positiva, de nuevo estamos fuera del rango posible de números y sigue habiendo un acarreo que hay que descartar, pero no hay que modificar el resultado para que sea correcto. De nuevo, si la resta es negativa no hay que descartar ningún acarreo y el resultado es el valor correcto de la resta en complemento a dos:

$$(1.3.10) \quad A - B = A + B^{(2)} = A + (2^n - |B|) = 2^n - (|B| - A) = (|B| - A)^{(2)}$$

CUADRO 3. Desbordamiento en suma binaria

Operando A	Operando B	Desbordamiento en A+B
Positivo	Negativo	No
Negativo	Positivo	No
Positivo	Positivo	Si es negativo
Negativo	Negativo	Si es positivo

Por tanto, en complemento a dos la resta queda totalmente integrada con la suma usando números negativos, y no es necesario utilizar ningún sistema especial para la resta. Este hecho, junto con la existencia de un sólo valor para el cero, ha llevado a que el complemento a dos sea el método más utilizado para la representación numérica en los sistemas digitales.

Aunque los operandos estén escritos correctamente en sistema binario en complemento a dos, no es garantía de que el resultado de una operación aritmética sea un número representable para el número de cifras binarias disponibles. Por ello, es necesario considerar las posibilidades de desbordamiento en la suma de números en complemento a dos..

Al sumar números de distinto signo, es seguro que el resultado es menor que cualquiera de los dos operandos. Por tanto, si en los operandos no aparece desbordamiento, en el resultado tampoco. Sólo es posible que aparezca desbordamiento si los dos operandos de la suma son del mismo signo, ya que en ese caso el resultado siempre será mayor que los dos operandos, tanto si son son positivos como si son negativos (el resultado será de mayor valor absoluto).

Existen varios métodos para saber si el resultado es incorrecto, ya que si el número supera el intervalo posible por la precisión asociada con el número n de bits, la cantidad no estará correctamente representada en el sistema de numeración en complemento a dos y el valor que obtengamos no será el resultado correcto. El método más sencillo (ver Tabla 3) consiste en observar que el resultado debe ser del mismo signo que los operandos: la suma de números positivos produce cantidades positivas, mientras que la suma de números negativos sólo puede producir cantidades negativas. Por tanto, si el bit de signo del resultado es diferente al de los operandos, sabemos que hay desbordamiento.

1.3.4. Exceso a 2^{n-1} .

Existen otros métodos para representar números binarios, pero vamos a estudiar sólo uno más, ya que se aplica en la expresión de los números en coma flotante. Se trata del sistema de desplazamiento o exceso a 2^{n-1} .

El sistema de exceso a 2^{n-1} consiste en sumar a todos los números (positivos y negativos) 2^{n-1} , de modo que el resultado siempre es positivo. El resultado

es expresado en binario natural y ése es nuestro número. Por ello, el valor más alto que se puede escribir con n bits es $2^{n-1} - 1$, ya que:

$$(2^{n-1} - 1) + 2^{n-1} = 2^n - 1 = N_{máx},$$

mientras que el número negativo de mayor valor absoluto será -2^{n-1} , ya que:

$$-(2^{n-1}) + 2^{n-1} = 0$$

Como el primer número positivo es el cero, su expresión será:

$$0 + 2^{n-1} = 2^{n-1}$$

expresión que es única (sólo un símbolo para el cero). Por tanto, la cifra más significativa del cero y de todos los números positivos mayores que cero en exceso a 2^{n-1} es 1 y por la misma razón, ya que su expresión en este sistema es más pequeña que 2^{n-1} , todos los números negativos tienen 0 en la cifra más significativa. Este hecho supone una ventaja en la ordenación de números en este sistema, ya que, a diferencia de los demás sistemas de numeración, los códigos correspondientes a números positivos son mayores que sus correspondientes números negativos.

Para saber el valor de un número cualquiera tenemos que restarle 2^{n-1} , lo que supone una gran desventaja para la suma, pues siempre aparece un 2^{n-1} de más. Para la suma de dos números A y B en exceso a 2^{n-1} tenemos:

$$A + 2^{n-1} + B + 2^{n-1} = ((A + B) + 2^{n-1}) + 2^{n-1}$$

La parte entre paréntesis es el resultado correcto de la suma en exceso a 2^{n-1} , pero aún hay que restarle otro 2^{n-1} para que sea coherente con este sistema. Debido a esto, no utilizaremos este sistema para operaciones aritméticas.

A continuación tenéis un cuadro (ver Tabla 4) que recoge todos los números enteros con $n = 4$ bits en todos los sistemas de coma fija que hemos descrito hasta ahora.

1.4. NÚMEROS EN COMA FLOTANTE

Los números en coma fija son útiles en muchas aplicaciones, pero cuando es preciso representar magnitudes muy grandes o muy pequeñas, la limitación del intervalo válido de representación puede suponer un problema. Por ello, se plantea la posibilidad de utilizar números que indiquen por un lado una cantidad, y por otra expresen un factor de escala, un valor que multiplica al primer número para precisar la cantidad que queremos representar. Cambiando

CUADRO 4. Números en coma fija con n = 4 bits

Decimal	Magn. con signo	Comp. a uno	Comp. a dos	Exceso a 2^{n-1}
-8	-	-	1000	0000
-7	1110	1000	1001	0001
-6	1110	1001	1010	0010
-5	1101	1010	1011	0011
-4	1100	1011	1100	0100
-3	1011	1100	1101	0101
-2	1010	1101	1110	0110
-1	1001	1110	1111	0111
-0	1000	1111	-	-
0	0000	0000	0000	1000
1	0001	0001	0001	1001
2	0010	0010	0010	1010
3	0011	0011	0011	1011
4	0100	0100	0100	1100
5	0101	0101	0101	1101
6	0110	0110	0110	1110
7	0111	0111	0111	1111

el factor de escala, es posible acceder a zonas de la recta real que no serían accesibles con el número de cifras de que disponemos. Este concepto es el que se utiliza en la *notación científica* o *notación en coma flotante*.

1.4.1. Notación científica.

El concepto de notación científica consiste en expresar una cantidad mediante dos números, uno llamado fracción f o mantisa y el exponente e de una potencia de 10, que es el factor de escala que multiplica a la fracción. Por ejemplo:

$$15,000 = 0.15 \times 10^5 \rightarrow f = 0.15; e = 5$$

$$0,00035 = 3.5 \times 10^{-4} \rightarrow f = 0.35; e = -4$$

donde se puede observar que los números pueden ser muy grandes con exponentes positivos y muy pequeños con exponentes negativos. La expresión de un número en notación científica no es unívoca, es decir, hay varios números que representan la misma cantidad, así:

$$15,000 = 0.15 \times 10^5 = 15 \times 10^3 = 0.0015 \times 10^7 \rightarrow \begin{cases} f = 0.15 & e = 5 \\ f = 15 & e = 3 \\ f = 0.0015 & e = 7 \end{cases}$$

tres números diferentes en notación científica que representan la misma cantidad. Podemos ver cómo existe una relación entre las tres fracciones, el número es el mismo pero la posición de la coma varía a medida que el valor del exponente aumente (desplaza la coma a la izquierda) o disminuya (desplaza la coma a la derecha).

En general:

$$N = f \cdot 10^e$$

Este sistema se utiliza en los sistemas digitales expresando en binario la fracción y el exponente, y usando 2 como base del exponente, así:

$$(1.4.1) \quad N = f \cdot 2^e,$$

y se llaman números en coma flotante a los datos numéricos de los sistemas digitales que usan este sistema. En general, para evitar duplicidad en las representaciones de los números, se suele elegir una forma llamada *normalizada*, que es la que tenemos cuando la primera cifra a la izquierda de la coma decimal es un 1. Por ejemplo, con 17 bits para la fracción (16 para el valor absoluto y uno para el signo) y 7 para el exponente (un total de 24 bits para el número) tenemos:

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
±	exponente										fracción												

La fracción se expresa en el sistema de magnitud con signo, mientras que el exponente se expresa en exceso a 2^{n-1} . El siguiente número está escrito en coma flotante:

$$\left| \begin{array}{c|ccccc} 23 & 22 & & & & 16 & | & 15 & & & & & & & & & & & & & & & & & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{array} \right|$$

Donde podemos ver que la fracción es $2^{-11} + 2^{-12} + 2^{-14} + 2^{-16}$, mientras que el exponente es: $2^6 + 2^4 + 2^2 - 2^6 = 20$, ya que está escrito en exceso a 2^6 . Por tanto:

$$(2^{-12} + 2^{-13} + 2^{-15} + 2^{-15}) \times 2^{20} = 432$$

Este número no está normalizado, la primera cifra diferente de cero de la fracción es la posición -11. Si queremos que la primera cifra a la derecha de la coma sea 1, debemos desplazar la coma hacia la derecha desde la posición 0 hasta la posición -10. Para ello basta con disminuir el exponente en tantas unidades como posiciones queremos desplazar, en nuestro caso $e = 20 - 10 = 10$ y el número queda:

23	22											16	15											0	
0	1											0	1											0	

La fracción es ahora $2^{-1} + 2^{-2} + 2^{-4} + 2^{-5}$, mientras que el exponente es: $(2^6 + 2^3 + 2^0) - 2^6 = 9$. Así, el valor del número no cambia:

$$(2^{-1} + 2^{-2} + 2^{-4} + 2^{-5}) \times 2^{10} = 432$$

Los números en coma flotante de los sistemas digitales se expresan mediante este sistema, pero en el caso de los computadores existe un método de representación con algunas modificaciones que ha sido aceptado por la industria informática. Este método se recoge en la norma IEEE 754.

1.4.2. Norma IEEE 754.

Se trata de un método para representar números en coma flotante reconocido por el instituto de normalización estadounidense ANSI (American National Standards Institute). Esta norma surge de la asociación estadounidense de ingenieros eléctricos y electrónicos IEEE (Institute of Electrical and Electronical Engineers) después de un largo proceso de estudio que llevó a su adopción por la mayoría de los fabricantes informáticos. Su nomenclatura es ANSI/IEEE Std. 754 (1985), ya que fue adoptada como norma en ese año, y sus principales diferencias con la coma flotante son que el exponente se expresa en exceso a $2^{n-1} - 1$, que los valores para los que el exponente tiene todas sus cifras 0 y todas sus cifras 1 son excepciones con significado específico y que la fracción normalizada tiene un 1 a la izquierda de la coma que se suprime (se conoce como “1” implícito).

La estructura del número, de 32 bits con 1 de signo y 23 de magnitud para la fracción y 8 bits para el exponente, será la siguiente:

31	30 29 28 27 26 25 24 23	22 21 20 19 18 17 ... 10 9 8 7 6 5 4 3 2 1 0	
<i>s</i>	<i>e</i>	<i>f</i>	

Con esta estructura, para calcular el valor del número, bastará con aplicar:

$$(1.4.2) \quad N = (-1)^s \times 2^{(e-127)} \times (1+f)$$

ya que, como hemos dicho más arriba, en la fracción normalizada hay un 1 a la izquierda de la coma que no aparece y el exponente está en exceso a $2^{n-1} - 1$, por lo que hay que restar 127 ($2^{8-1} - 1$), al exponente para obtener su valor. Este tipo de dato numérico se conoce como flotante de *precisión sencilla*.

Existe otro tipo de dato de 64 bits, con 1 + 52 bits para la fracción y 11 bits para el exponente. Se llama flotante de *doble precisión* y su valor se obtiene de:

$$(1.4.3) \quad N = (-1)^s \times 2^{(e-1023)} \times (1+f)$$

ya que, como el número de bits del exponente es 11, el exceso es 1023 ($2^{11-1} - 1$).

CUADRO 5. Excepciones de la norma IEEE 754

Exponente	Fracción	Valor del dato
Todo 0s	Distinto de 0	No normalizado
Todo 0s	Todo 0s	Cero
Todo 1s	Todo 0s	Infinito
Todo 1s	Distinto de 0	No es dato numérico (NaN)

Cuando el exponente presenta todas sus cifras a 0 o a 1 (lo que correspondería a -127 o a 128 en precisión sencilla y a -1023 o 1024 en doble precisión), el número representado no se ajusta a este método, sino que tiene valores especiales que pueden ser útiles de representar fuera del sistema de coma flotante (ver Tabla 5). Las excepciones son las siguientes:

- Si el exponente es todo 0s y la fracción es distinta de cero, el número *no está normalizado*. Por tanto, la fracción no tiene un 1 antes de la coma.
- Si tanto exponente como fracción son todo 0s, el valor representado es un *cero exacto*. El que todas las cifras sean 0 sólo significa que el valor representado es más pequeño que el número mínimo que podemos representar con la cantidad de bits que tenemos. En este caso, salimos del método para indicar el cero.
- Si el exponente es todo 1s y la fracción es todo 0s, el valor es *infinito*.
- Si el exponente es todo 1s y la fracción es distinta de cero, el dato *no es un número (not a number)*. Este tipo de dato se suele abbreviar *NaN* y se puede usar cuando el resultado de una operación no es un dato

numérico, por ejemplo si es una raíz imaginaria o si es una indeterminación.

Dejando aparte las excepciones ($0, \infty, \text{NaN}$), los valores que podemos representar mediante la norma IEEE 754 de precisión sencilla y doble precisión están comprendidos entre los siguientes valores extremos (Tabla 6):

CUADRO 6. Valores extremos de la norma IEEE 754

Concepto	Precisión sencilla	Doble precisión
Normalizado más alto	$1 + (1 - 2^{-23}) \times 2^{127} \approx 10^{38}$	$1 + (1 - 2^{-52}) \times 2^{1023} \approx 10^{308}$
Normalizado más pequeño	$1 \times 2^{-126} \approx 10^{-38}$	$1 \times 2^{-1022} \approx 10^{-308}$
No normalizado más pequeño	$2^{-23} \times 2^{-126} \approx 10^{-45}$	$2^{-52} \times 2^{-1022} \approx 10^{-324}$

Un ejemplo de precisión sencilla es el siguiente: dado un nº expresado en coma flotante y precisión sencilla $N=C0500000H$, determinar su valor decimal.

1. Primero se expresa en binario:
1 0 1 1 0 0 0 0 0 1 0 1 0
 2. Segundo se realizan las divisiones de los campos de información:
1|0 1 1 0 0 0 0 0| 1 0 1 0
 3. Se obtiene el signo:
 $s = (-1)^1 = -1$, es un nº negativo.
 4. Se obtiene el exponente:
 $e = 2^5 + 2^6 = 96$, teniendo en cuenta que está expresado en exceso 127, $e = 96 - 127 = -29$
 5. Se obtiene la fracción o mantisa:
 $f = 1 + 2^{-1} + 2^{-3} = 1 + 0.5 + 0.125 = 1.725$
 6. El nº en binario es:
 $N = -1.725 \cdot 2^{-29}$
 7. El nº en decimal es:
 $N = -3.21306 \cdot 10^{-9}$

1.5. CÓDIGOS ALFANUMÉRICOS

Los datos que se utilizan en los sistemas digitales no siempre son numéricos. En ocasiones, nos interesa representar conceptos que luego serán presentados al usuario como texto, por tanto, es preciso representar en números binarios símbolos alfabéticos. Los sistemas binarios destinados a representar letras se llaman *códigos alfanuméricicos*.

1.5.1. Código ASCII.

Los códigos alfanuméricos se llevan utilizando desde el principio de la tecnología digital, pero uno de los primeros códigos de uso estándar es el ASCII (American Standard Code for Infomation Interchange). Se trata de un código de 7 cifras binarias, en el que cada una de las 128 combinaciones representa un carácter (ver Figura 12). Inicialmente el octavo bit se usaba como bit de paridad para detectar la existencia de errores en la transmisión de un código. Los códigos 00_{16} a $1F_{16}$ (en decimal del 0 al 31) se asignaron a una serie de comandos pensados para la comunicación entre una terminal remota y el sistema central (por ejemplo el carácter 04_{16} = EOT *end of transmission*), por tanto no son caracteres imprimibles y no tienen significado alfabético. El resto representan los símbolos que utilizamos habitualmente para comunicar información en forma de texto.

Dado que muchas lenguas europeas usan otros símbolos que no están definidos en el código ASCII, se modificó el código añadiendo otros 128 símbolos al extender el código a 8 bits. Los primeros 128 símbolos se mantuvieron sin cambios, mientras que los siguientes 128 números binarios (MSB=1) se asignaron a caracteres acentuados o símbolos especiales. (*Latin-1*) La Figura 12 es para $b_8=0$, con $b_8=1$ aparecen el resto de caracteres del código Latin-1.

b_7	b_6	b_5	b_4	b_3	b_2	b_1	Bits	0	0	0	0	1	0	1	0	1	0	0	1	0	1	1	0	1	1	1	0	1	1	1
								Columna ↓	Fila ↓	0	1	2	3	4	5	6	7													
0	0	0	0	0	0	0	NULL			DLE		SP		0	@	P	'	p												
0	0	0	0	1	1	1	SOH			DC1		!		1	A	Q	a	q												
0	0	1	0	2	2	2	STX			DC2		"		2	B	R	b	r												
0	0	1	1	3	3	3	ETX			DC3		#		3	C	S	c	s												
0	1	0	0	4	4	4	EOT			DC4		\$		4	D	T	d	t												
0	1	0	1	5	5	5	ENQ			NAK		%		5	E	U	e	u												
0	1	1	0	6	6	6	ACK			SYN		&		6	F	V	f	v												
0	1	1	1	7	7	7	BEL			ETB		'		7	G	W	g	w												
1	0	0	0	8	8	8	BS			CAN		(8	H	X	h	x												
1	0	0	1	9	9	9	HT			EM)		9	I	Y	i	y												
1	0	1	0	10	10	10	LF			SUB		*		:	J	Z	j	z												
1	0	1	1	11	11	11	VT			ESC		+		;	K	[k	{												
1	1	0	0	12	12	12	FF			FC		,		<	L	\	l													
1	1	0	1	13	13	13	CR			GS		-		=	M]	m	}												
1	1	1	0	14	14	14	SO			RS		.		>	N	^	n	~												
1	1	1	1	15	15	15	SI			US		/		?	O	_	o	DEL												

FIGURA 12. Código ASCII

Control		ASCII						Control		Latin 1						
000	001	002	003	004	005	006	007	008	009	00A	00B	00C	00D	00E	00F	
0	CTRL	CTRL	[SPACE]	0	@	P	`	p	CTRL	CTRL	[NBSpace]	°	À	Ð	à	Ð
1	CTRL	CTRL	!	1	A	Q	a	q	CTRL	CTRL	¡	±	Á	Ñ	á	ñ
2	CTRL	CTRL	"	2	B	R	b	r	CTRL	CTRL	¢	²	Â	Ò	â	ò
3	CTRL	CTRL	#	3	C	S	c	s	CTRL	CTRL	£	³	Ã	Ó	ã	ó
4	CTRL	CTRL	\$	4	D	T	d	t	CTRL	CTRL	¤	'	Ä	Ö	ä	ö
5	CTRL	CTRL	%	5	E	U	e	u	CTRL	CTRL	¥	µ	À	Ö	à	ö
6	CTRL	CTRL	&	6	F	V	f	v	CTRL	CTRL	¦	¶	Æ	Ö	æ	ö
7	CTRL	CTRL	,	7	G	W	g	w	CTRL	CTRL	§	·	Ç	×	ç	÷
8	CTRL	CTRL	(8	H	X	h	x	CTRL	CTRL	“	”	È	Ø	è	ø
9	CTRL	CTRL)	9	I	Y	i	y	CTRL	CTRL	®	º	É	Ù	é	ù
A	CTRL	CTRL	*	:	J	Z	j	z	CTRL	CTRL	ª	º	Ê	Ú	ê	ú
B	CTRL	CTRL	+	:	K	[k	{	CTRL	CTRL	«	»	Ë	Û	ë	û
C	CTRL	CTRL	,	<	L	\	l		CTRL	CTRL	¬	½	Ì	Û	ì	û
D	CTRL	CTRL	-	=	M]	m]	CTRL	CTRL	-	½	Í	Ý	í	ý
E	CTRL	CTRL	.	>	N	^	n	~	CTRL	CTRL	®	¾	Ì	þj	í	þj
F	CTRL	CTRL	/	?	O	_	o	CTRL	CTRL	-	¸	Í	Þ	í	þ	

FIGURA 13. Los primeros 256 caracteres de UNICODE

A medida que se fue extendiendo el campo de aplicación de este código, apareció la necesidad de adaptarlo a otras lenguas no europeas. Este problema se resolvió definiendo de antemano en qué grupo de idiomas se realiza la operación del sistema digital, y a continuación seleccionando un código en el que se definan los caracteres necesarios para esa lengua. Este método se conoce como *página de código*, ya que la idea es definir los datos de 8 bits en una tabla diferente, cada una con 256 caracteres adecuados a la lengua en que opera el sistema. Esto conlleva la operación previa de definir en qué página nos colocamos para poder realizar la comunicación, aparte de la imposibilidad de combinar más de un idioma en cada operación.

Por otro lado, existen sistemas de escritura que no utilizan letras, sino símbolos que representan ideas o palabras, como los caracteres kanji usados en China y Japón. Estos símbolos son varios miles, por lo que una página de código apenas puede reproducir una fracción muy pequeña de los que pueden aparecer en un texto sencillo. Este tipo de problemas han llevado a la adopción de un código alfanumérico de mayor número de bits, el código *UNICODE*.

1.5.2. Código UNICODE.

El sistema UNICODE asigna a cada carácter un único número binario (*punto de código*) de 16 bits al que se asigna a un carácter determinado ($2^{16} \approx 64,000$ caracteres). Los grupos de caracteres asociados a una lengua concreta aparecen como números consecutivos en el código, definiendo zonas del código correspondientes a los idiomas utilizados. Con el fin de facilitar la transición de un sistema a otro, el código UNICODE asigna a los primeros 256 números los mismos símbolos que Latin-1.

Se puede ver (Figura 13) que los códigos ASCII 61_{16} a $7A_{16}$ (en decimal del 97 al 122) correspondientes a las letras minúsculas, se transforman en códigos UNICODE 0061_{16} a $007A_{16}$ (los mismos, pero ahora con 16 bits) que son las misma letras minúsculas. Los teclados de los sistemas digitales realizan la función de codificar el valor binario UNICODE correspondiente a la tecla pulsada (si tecleamos el código UNICODE en base 10 en el teclado numérico de un PC mientras mantenemos pulsada la tecla Alt, introducimos el carácter correspondiente a ese código).

El sistema se inició con 16 bits (UTF-16), pero actualmente también existe una versión con puntos de código de 32 bits (UTF-32) para cada carácter. Los $2^{32} \approx 4 \times 10^9$ caracteres son asignados por un consorcio creado por Apple, Microsoft y Sun, entre otras empresas, desde 1991.

EJERCICIOS

1. ¿En un sistema de numeración posicional cómo representarías los números $N = 2443_{10}$ y $N = A7D6H$?
2. Convertir al sistema decimal los siguientes números en sistema binario:
 - a) 11
 - b) 100
 - c) 111
 - d) 1000
 - e) 11101
 - f) 11,011
3. ¿Cuál es el número decimal más alto que se puede expresar con los siguientes número de bits?
 - a) 2 bit
 - b) 7 bit
 - c) 10 bit
4. ¿Cuántos bits son necesarios para expresar los siguientes números en binario?
 - a) 17
 - b) 81
 - c) 35
 - d) 32
5. Convertir al sistema decimal:
 - a) $E5_{16}$
 - b) $B2F8_{16}$
6. Convierte al sistema decimal el siguiente nmero en base ocho: 2374₈
7. Conversión binario-hexadecimal
 - a) 1100101001010111
 - b) 01101001101
8. Conversión hexadecimal-binario
 - a) $10A4_{16}$
 - b) $CF8E_{16}$
 - c) 9742_{16}
9. Conversión decimal-hexadecimal
 - a) 650_{10}
 - b) 4025_{10}
10. Convierte el número $N=234,56_{10}$ a binario, estando expresada la parte decimal sobre 5 bits
11. Convierte de decimal a binario (Máximo cuatro cifras a la derecha de la coma, si la conversión no es completa, indica el error relativo):
 - a) 177.625
 - b) 78.4375
 - c) 113.7

12. Conversión de binario a decimal y a hexadecimal y octal:
- 10011100.1001
 - 110111.001
 - 1001001.001
13. Expresa el número N= - 57 en binario complemento a 2 sobre 8 bits
14. Expresión de números negativos. Escribe con 8 bits el siguiente número decimal en magnitud con signo, complemento a 1, complemento a 2 y exceso a 128:
- 113
 - 78
15. ¿Qué número decimal representan los siguientes números binarios en cada sistema? Rellena la tabla y utiliza los valores para comprobar si el resultado de las sumas propuestas puede ser correcto en ocho bit (para cada sistema existen unos límites en los resultados posibles, superarlos produce desbordamiento: resultado en 9 bit o signo opuesto).

	Binario natural	Magnitud con signo	Compl. a 1	Compl. a 2	Exceso a 128
A	01001010				
B	00101010				
C	01001100				
D	01010100				
E	10100010				
F	11101110				
G	11000001				
H	10111001				

- a) Realiza las siguientes operaciones en binario natural:
 $0 \leq N \leq 2^n - 1 \rightarrow n = 8 \text{ bit} \rightarrow [0, 255]$
- A+B
 - C+D
 - E+F
 - G+H
- b) Realiza las siguientes operaciones en complemento a 2:
 $-2^{n-1} \leq N \leq 2^{n-1} - 1 \rightarrow n = 8 \text{ bit} \rightarrow [-128, 127]$
- C+D
 - E+F
 - G+H
 - B+G
16. Números decimales codificados en binario: Convertir de BCD a decimal y a binario natural.
- 0010 0101 0111
 - 0110 0011 1000

17. Coma flotante IEEE Std. 754: Convierte los siguientes números de hexadecimal a decimal (están escritos en coma flotante y precisión sencilla según el estándar IEEE 754):
- a) 42E48000H
 - b) 3F880000H
 - c) 00800000H
 - d) C7F00000H

Capítulo 2

INTRODUCCIÓN AL ÁLGEBRA DE BOOLE Y PUERTAS LÓGICAS

Las revoluciones no se hacen por menudencias, pero nacen por menudencias. (Aristóteles de Estagira)

RESUMEN. En el Capítulo 1 se han sentado las bases de la representación numérica para abordar operaciones aritméticas. La representación se realiza sobre una base binaria, 0 y 1. En el presente Capítulo, partiendo de la representación binaria, se desarrollan diferentes relaciones y operaciones, de tipo lógico, entre variables binarias. Agrupando variables binarias y estableciendo relaciones lógicas entre ellas, se llegará al concepto de función lógica binaria. Abordando métodos que permiten simplificar expresiones que, por consiguiente, redundan en menores costos de diseño.

2.1. ÁLGEBRA DE BOOLE

2.1.1. Introducción.

En 1938, Shannon propuso aplicar al diseño de circuitos digitales un método matemático para tratar funciones digitales. Dicho método había sido propuesto en el siglo XIX por George Boole¹ para analizar proposiciones lógicas de tipo Verdadero/Falso: el álgebra de la lógica o álgebra Booleana. Esta lógica está basada en dos operaciones: unión y disyunción. Otra operación es la negación.

2.1.2. Representación de funciones lógicas.

Este método se basa en el uso de variables de dos valores y tres operaciones entre ellas. Las relaciones se definen mediante tablas que contienen todos los valores posibles de las variables y de la función → Tablas de la verdad.

La Tabla 1 muestra las tres operaciones básicas: a) disyunción AND (Y lógica), b) unión OR (O lógica) y c) negación NOT (NO lógico). La Tabla 1(a) muestra que el resultado de la operación AND es 1 si y solo si las dos variables

¹Lincoln (Gran Bretaña) 1815 - Ballintemple (Irlanda) 1864

valen 1. La Tabla 1(b) muestra que el resultado de la operación OR es 1 cuando cualquiera de las dos variables vale 1. La Tabla 1(c) muestra que el operador NOT complementa el valor de la variable.

CUADRO 1. Tablas de la verdad de las operaciones básicas del álgebra de Boole

AND			OR			NOT	
X	Y	$Z = X \cdot Y$	X	Y	$Z = X + Y$	X	$Z = \bar{X}$
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

(a)

(b)

(c)

2.1.3. Postulados e Identidades del Álgebra de Boole.

2.1.3.1. Postulados.

Partimos de una serie de afirmaciones que no necesitan ser demostradas, estas se denominan “postulados” o “axiomas”. La Tabla 2 muestra los diferentes postulados fundamentales en el desarrollo del álgebra Booleana. El símbolo “.” representa la operación AND y se denomina producto lógico, el símbolo “+” representa la operación OR y se denomina suma lógica y el símbolo “ \bar{X} ” representa la operación NOT (sobre la variable X).

CUADRO 2. Postulados

$1 \cdot 1 = 1$	$0 + 0 = 1$	$X = 0 \text{ si } X \neq 1$
$0 \cdot 0 = 0$	$1 + 1 = 1$	$X = 1 \text{ si } X \neq 0$
$1 \cdot 0 = 0$	$0 + 1 = 1$	$\bar{X} = 0 \text{ si } X = 1$
$0 \cdot 1 = 0$	$1 + 0 = 1$	$\bar{X} = 1 \text{ si } X = 0$

Utilizando estas afirmaciones (que son la definición de los números y las operaciones de éste álgebra), podemos crear nuevas proposiciones y relaciones.

2.1.3.2. Identidades, propiedades básicas y leyes.

La Tablas 3, 4 y 5 muestran las identidades, propiedades y leyes que afectan a las variables booleanas respectivamente. Comprobando que cumplen los postulados, sabemos que las relaciones que se muestran en las tablas mencionadas, son correctas.

CUADRO 3. Identidades básicas del álgebra de Boole

$X + 0 = X$	$\bar{X} = X$	$X \cdot 1 = X$
$X + 1 = 1$		$X \cdot 0 = 0$
$X + \bar{X} = X$		$X \cdot X = X$
$X + X = X$		$X \cdot \bar{X} = 0$

CUADRO 4. Propiedades básicas del álgebra de Boole

$X + Y = Y + X$	$X \cdot Y = Y \cdot X$	Commutativa
$X + (Y + Z) = (X + Y) + Z$	$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$	Asociativa
$X \cdot (Y + Z) = X \cdot Y + X \cdot Z$	$X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$	Distributiva
$(X + Y) \cdot (X + \bar{Y}) = X$	$X \cdot Y + X \cdot \bar{Y} = X$	Combinación

CUADRO 5. Leyes de De Morgan

$$\overline{X + Y} = \overline{X} \cdot \overline{Y} \quad \overline{X \cdot Y} = \overline{X} + \overline{Y}$$

- Las identidades, las propiedades y las leyes se pueden comprobar definiendo la tabla de la verdad de las funciones a ambos lados de la igualdad y comprobando que son iguales.

Como ejemplo de demostración, la Tabla 6 demuestra la propiedad distributiva del producto respecto a la suma:

CUADRO 6. Demostración de la propiedad distributiva

X	Y	Z	$Y + X$	$X \cdot (Y + Z)$	$X \cdot Y$	$X \cdot Z$	$X \cdot Y + Z \cdot Y$
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	1	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

Aunque no forma parte de las operaciones básicas, existe otra operación que se define por su utilidad: la O (OR) exclusiva o suma exclusiva (XOR).

CUADRO 7. Operación suma exclusiva - XOR

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

El resultado es “1” si los operandos son distintos.

La suma exclusiva también tiene la propiedad asociativa (vease la Tabla 8). Por tanto, la función que define la operación XOR sobre tres o más variables, es tal que vale 1 si el número de 1s en la entrada es impar. Esta función se conoce como detector de paridad impar.

CUADRO 8. Demostración de la propiedad asociativa para la XOR

X	Y	Z	$Y \oplus Z$	$X \oplus (Y \oplus Z)$	$X \oplus Y$	$(X \oplus Y) \oplus Z$
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	1	1	1	1
0	1	1	0	0	1	0
1	0	0	0	1	1	1
1	0	1	1	0	1	0
1	1	0	1	0	0	0
1	1	1	0	1	0	1

2.1.4. Expresión Canónica de una Función. Mediante el álgebra booleana y las tablas de la verdad, podemos crear nuevas funciones, esto es:

- De la tabla de la verdad de una función lógica, podemos obtener su expresión algebraica, esta se denomina: expresión canónica.
- La expresión canónica contiene todas las variables de la función en todos sus términos.
- Los términos de la expresión canónica pueden ser productos (**mintérminos**) o sumas (**Maxtérminos**).

Por lo que concluimos que una función lógica puede ser expresada en su forma canónica de mintérminos o Maxtérminos. A continuación, se desarrollan estos conceptos.

2.1.4.1. Expresión canónica en mintérminos.

Veamos el desarrollo de la tabla de la verdad para una función de 3 variables, esta muestra los términos denominados “*minterm*” o mintérminos. La Tabla 9 muestra este desarrollo:

CUADRO 9. Desarrollo de una función con mintérminos

			Término de									
X	Y	Z	producto	Símbolo	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0	0	0	$\bar{X} \cdot \bar{Y} \cdot \bar{Z}$	m_0	1	0	0	0	0	0	0	0
0	0	1	$\bar{X} \cdot \bar{Y} \cdot Z$	m_1	0	1	0	0	0	0	0	0
0	1	0	$\bar{X} \cdot Y \cdot \bar{Z}$	m_2	0	0	1	0	0	0	0	0
0	1	1	$\bar{X} \cdot Y \cdot Z$	m_3	0	0	0	1	0	0	0	0
1	0	0	$X \cdot \bar{Y} \cdot \bar{Z}$	m_4	0	0	0	0	1	0	0	0
1	0	1	$X \cdot \bar{Y} \cdot Z$	m_5	0	0	0	0	0	1	0	0
1	1	0	$X \cdot Y \cdot \bar{Z}$	m_6	0	0	0	0	0	0	1	0
1	1	1	$X \cdot Y \cdot Z$	m_7	0	0	0	0	0	0	0	1

- Un **mintérmino** es un producto que vale 1 sólo para una combinación de valores de las variables y se anula para todas las demás, p. ej., $X \cdot \bar{Y} \cdot Z$.
- En la expresión del **mintérmino**, si la variable que no anula el producto vale 1, aparece sin negar, y si vale 0 aparece negada.

2.1.4.2. Expresión canónica en Maxtérminos.

Desarrollando la tabla de la verdad para una función de 3 variables, esta muestra los términos denominados “*Maxterm*” o Maxtérminos. La Tabla 10 muestra este desarrollo:

- Un **Maxtérmino** es una suma que vale 0 sólo para una combinación de valores de las variables.
- En la expresión del **Maxtérmino**, si la variable que anula la suma vale 0, aparece sin negar, y si vale 1 aparece negada.
- Cada **Maxtérmino** es la negación de su **mintérmino** correspondiente
 $\rightarrow M_i = \bar{m}_i$.

CUADRO 10. Desarrollo de una función con Maxtérminos

Término de			producto	Símbolo	m ₀							
X	Y	Z			m ₁	m ₂	m ₃	m ₄	m ₅	m ₆	m ₇	
0	0	0	$X + Y + Z$	M_0	0	1	1	1	1	1	1	1
0	0	1	$X + Y + Z$	M_1	1	0	1	1	1	1	1	1
0	1	0	$X + \bar{Y} + Z$	M_2	1	1	0	1	1	1	1	1
0	1	1	$X + \bar{Y} + \bar{Z}$	M_3	1	1	1	0	1	1	1	1
1	0	0	$\bar{X} + Y + Z$	M_4	1	1	1	1	0	1	1	1
1	0	1	$\bar{X} + Y + \bar{Z}$	M_5	1	1	1	1	1	0	1	1
1	1	0	$\bar{X} + \bar{Y} + Z$	M_6	1	1	1	1	1	1	0	1
1	1	1	$\bar{X} + \bar{Y} + \bar{Z}$	M_7	1	1	1	1	1	1	1	0

2.1.4.3. Dualidad de las expresiones canónicas.

Como ya se ha expuesto, una función puede ser expresada en forma canónica de mintérminos o Maxtérminos indistintamente:

- La expresión canónica es la suma de todos los mintérminos correspondientes a las combinaciones de valores de variables que hacen 1 la función.
- También es expresión canónica el producto de los Maxtérminos correspondientes a las combinaciones de valores de variables que hacen 0 la función.

De este modo, cada función tiene asociadas dos expresiones canónicas → Suma de mintérminos/Producto de Maxtérminos.

- Dada una tabla de la verdad conocida, se puede obtener la función algebraica expresada en sus dos formas canónicas.

Veamos a continuación la demostración de la dualidad de los términos canónicos. Partimos de una función expresada inicialmente en la tabla de la verdad. Como ejemplo, sea ésta la expresada en la Tabla 11:

CUADRO 11. Tabla dual

X	Y	Z	F	\bar{F}
0	0	0	1	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	1	0
1	1	0	0	1
1	1	1	1	0

La forma algebráica de F y \bar{F} será la siguiente:

$$F = \bar{X} \cdot \bar{Y} \cdot \bar{Z} + \bar{X} \cdot Y \cdot \bar{Z} + X \cdot \bar{Y} \cdot Z + X \cdot Y \cdot Z = m_0 + m_2 + m_5 + m_7$$

$$\bar{F} = \bar{X} \cdot \bar{Y} \cdot Z + \bar{X} \cdot Y \cdot Z + X \cdot \bar{Y} \cdot \bar{Z} + X \cdot Y \cdot \bar{Z} = m_1 + m_3 + m_4 + m_6$$

Complementando esta última función y a través de las leyes de De Morgan, se demuestra que las dos expresiones canónicas son idénticas:

$$F = \overline{m_1 + m_3 + m_4 + m_6} = \bar{m}_1 \cdot \bar{m}_3 \cdot \bar{m}_4 \cdot \bar{m}_6$$

Teniendo en cuenta que:

$$M_i = \bar{m}_i,$$

obtenemos la expresión siguiente:

$$F = M_1 \cdot M_3 \cdot M_4 \cdot M_6$$

2.1.5. Simplificación de Funciones Lógicas. Las expresiones canónicas nos permiten expresar algebraicamente cualquier función lógica. Estas expresiones suelen ser largas

- Mediante el álgebra booleana, es posible transformar una expresión en otra equivalente de menos términos, que sea más sencilla de implementar → Simplificación.

- El método de los mapas de Karnaugh nos permite sistematizar la simplificación de funciones.

2.1.5.1. Simplificación mediante mapas de Karnaugh mediante mintérminos.

Primeramente hay que introducir el concepto de adyacencia, o términos adyacentes, ya que es sobre este concepto sobre el que se apoya Karnaugh para llevar a cabo su sistema de simplificación gráfica.

DEFINICIÓN 1. Términos adyacentes: son aquellos términos canónicos que solo se diferencian en una variable complementada, es decir, en un término una variable se presenta complementada y en el otro término está sin complementar, p. ej., $\bar{X} \cdot \bar{Y} \cdot Z$ y $\bar{X} \cdot \bar{Y} \cdot \bar{Z}$.

Teniendo en cuenta esta definición, si tenemos la función:

$$F = \bar{X} \cdot \bar{Y} \cdot \bar{Z} + \bar{X} \cdot \bar{Y} \cdot Z,$$

la simplificación es la siguiente:

$$F = \bar{X} \cdot \bar{Y} \cdot (\bar{Z} + Z) = \bar{X} \cdot \bar{Y} \cdot 1 = \bar{X} \cdot \bar{Y},$$

por lo que concluimos que los términos adyacentes son simplificables eliminando la variable que se presenta complementada - no complementada. El mismo razonamiento aplica a las expresiones canónicas con Maxtérminos y que se verá en la Subsección 2.1.5.2.

Atendiendo a lo anteriormente expuesto, un mapa de Karnaugh presenta las siguientes características:

- Es una representación de la tabla de la verdad como tabla de doble entrada (atiende a las adyacencias de los **mintérminos** o **Maxtérminos**).
- Cada punto de intersección de fila y columna (casilla del mapa) es un valor de la función.
- Cada casilla se corresponde con un mintérmino (o Maxtérmino), se escribe un 1 en los mintérminos (o un 0 en los Maxtérminos) de la función.

La Figura 1 muestra la organización de una función de dos variables, de forma gráfica, sobre estructuras matriciales, atendiendo a los mintérminos.

		y	
		0	1
$x \setminus y$	0	$\bar{x} \cdot \bar{y}$	$\bar{x} \cdot y$
	1	$x \cdot \bar{y}$	$x \cdot y$

FIGURA 1. Mapa de Karnaugh genérico para 2 variables

DEFINICIÓN 2. Término implicante: es el término resultante de la simplificación.

Del ejemplo anterior, el término $\bar{X} \cdot \bar{Y}$ es un término implicante.

Podemos resumir lo anteriormente expuesto en los tres puntos siguientes:

- Segundo el número de variables de la función, los mapas de Karnaugh tienen 4 casillas (dos variables - ver Figura 1), 8 casillas (tres variables - ver Figura 2), 16 casillas (cuatro variables - ver Figura 3), es decir, 2^n casillas, siendo n el número de variables..
- Los mintérminos correspondientes a casillas adyacentes en el mapa de Karnaugh se pueden agrupar en un sólo producto (combinación).
- En la expresión de ese término (implicante) sólo aparecen las variables de valores comunes a las dos casillas.

		y			
		00	01	11	10
$x \setminus yz$	0	$\bar{x} \cdot \bar{y} \cdot \bar{z}$	$\bar{x} \cdot \bar{y} \cdot z$	$\bar{x} \cdot y \cdot z$	$x \cdot y \cdot \bar{z}$
	1	$x \cdot \bar{y} \cdot \bar{z}$	$x \cdot \bar{y} \cdot z$	$x \cdot y \cdot z$	$x \cdot y \cdot \bar{z}$

FIGURA 2. Mapa de Karnaugh genérico para 3 variables

				y				
				00	01	11	01	
wx \ yz				w	00	01	11	01
m_0	m_1	m_3	m_2	w	$\bar{w} \cdot \bar{x} \cdot \bar{y} \cdot \bar{z}$	$\bar{w} \cdot \bar{x} \cdot \bar{y} \cdot z$	$\bar{w} \cdot \bar{x} \cdot y \cdot z$	$\bar{w} \cdot x \cdot y \cdot \bar{z}$
m_4	m_5	m_7	m_6		$\bar{w} \cdot x \cdot \bar{y} \cdot \bar{z}$	$\bar{w} \cdot x \cdot \bar{y} \cdot z$	$\bar{w} \cdot x \cdot y \cdot z$	$\bar{w} \cdot x \cdot y \cdot \bar{z}$
m_{12}	m_{13}	m_{15}	m_{14}		$w \cdot x \cdot \bar{y} \cdot \bar{z}$	$w \cdot x \cdot \bar{y} \cdot z$	$w \cdot x \cdot y \cdot z$	$w \cdot x \cdot y \cdot \bar{z}$
m_8	m_9	m_{11}	m_{10}		$w \cdot \bar{x} \cdot \bar{y} \cdot \bar{z}$	$w \cdot \bar{x} \cdot \bar{y} \cdot z$	$w \cdot \bar{x} \cdot y \cdot z$	$w \cdot x \cdot y \cdot \bar{z}$

z

x

FIGURA 3. Mapa de Karnaugh genérico para 4 variables

Veamos, a continuación, un ejemplo de simplificación de una función de 3 variables:

$$F = m_1 + m_2 + m_3 + m_5 + m_7$$

La Figura 4 muestra las agrupaciones de términos adyacentes para su simplificación:

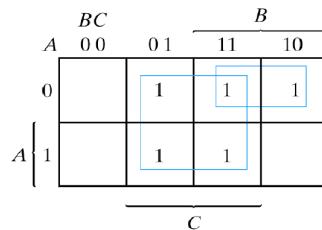


FIGURA 4. Mapa de Karnaugh para simplificar una función de 3 variables

- m_1, m_3, m_5 y m_7 se agrupan en un implicante único: C .
- m_3 y m_2 se agrupan en el implicante $\bar{A} \cdot B$.
- La suma de estos dos implicantes contiene a todos los mintérminos de la función, luego es equivalente a su expresión canónica $\rightarrow F = C + \bar{A} \cdot B$.

Introduzcamos ahora las siguientes definiciones:

DEFINICIÓN 3. **Celdas distinguidas:** son las celdas que sólo pertenecen a un implicante primo.

DEFINICIÓN 4. Implicante primo esencial: es aquel que contiene una celda esencial.

La **suma mínima** de una función es una suma de implicantes primos de esa función (puede que no todos), y siempre contiene a todos los implicantes primos esenciales.

Veamos éste concepto por medio de dos ejemplos:

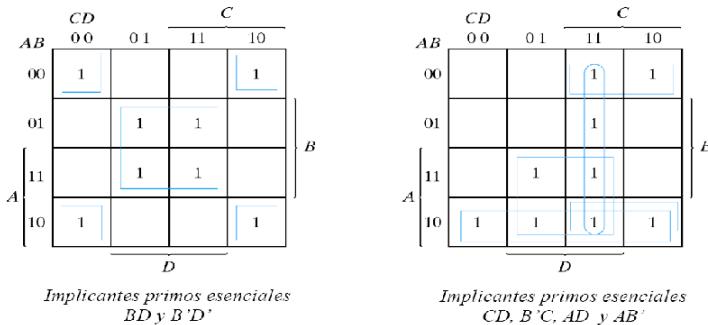


FIGURA 5. Mapas de Karnaugh que muestran los términos implicantes primos

2.1.5.2. Simplificación mediante mapas de Karnaugh con Maxtérminos.

Las celdas vacías (ceros de la función) corresponden a los Maxtérminos de la función.

- Se pueden agrupar (combinación) celdas de ceros adyacentes en un sólo implicante (suma) que contiene a los Maxtérminos correspondientes. En su expresión sólo aparecen las variables de valor común en todas las casillas.
- El producto de los implicantes de ceros que contiene a todos los Maxtérminos de la función, es equivalente a su expresión canónica.
- Llamaremos producto mínimo de una función al producto de menor número de términos y con menor número de variables en cada término que incluya a todos los Maxtérminos de la función.

Veamos el proceso de simplificación a través de un ejemplo. Sea la función expresada en términos canónicos de mintérminos:

$$F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$$

El mapa de karnaugh se corresponde con:

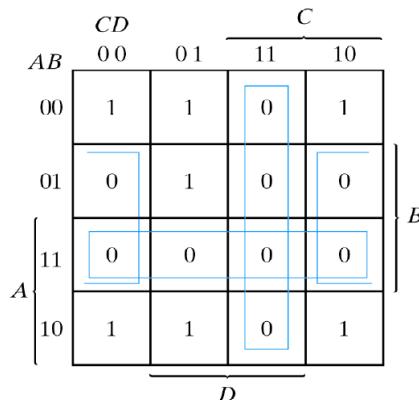


FIGURA 6. Mapa de Karnaugh (Maxtérminos) para simplificar una función de 4 variables

- M_4, M_6, M_{12} y M_{14} se agrupan en un sólo implicante primo: $\bar{B} + D$.
- Los otros dos implicantes son: $\bar{A} + \bar{B}$ y $\bar{C} + \bar{D}$.
- Puesto que todos los implicantes son esenciales, el producto mínimo es:

$$F = (\bar{B} + D) \cdot (\bar{A} + \bar{B}) \cdot (\bar{C} + \bar{D})$$

2.1.5.3. Simplificación de funciones de especificación incompleta.

Las funciones de especificación incompleta son aquéllas para las que no están asignados valores para todas las combinaciones de variables de entrada.

- En muchos casos, no todas las combinaciones de variables son posibles, por lo que no se define valor de la función para ellas, se indica con una X.
- El término asignado para estas combinaciones de variables se llama término “no importa”, y se nombra con “d” y el subíndice correspondiente a la combinación de variables para la que no está definida la función.
- En estos casos, se puede escoger para la función el valor que suponga mayor simplificación: 0 ó 1.

Veamos esta simplificación a través de un ejemplo:

$$F = \sum m(1, 3, 7, 11, 15) + \sum d(0, 2, 5)$$

Podemos realizar la elección de términos “no importa” como la siguiente:

- Si tomamos 0, 2 y 5 como ceros de la función, el implicante primo sólo contiene 1 y 3: $\bar{w} \cdot \bar{x} \cdot z$.
- Los términos 0 y 2 son tomados como 1 para formar el implicante primo: $\bar{w} \cdot \bar{x}$.
- Dejando 5 como 0, la expresión que representa la suma mínima es:

$$F = y \cdot z + \bar{w} \cdot \bar{x}$$

El mapa de Karnaugh correspondiente se muestra en la Figura 7 (a):

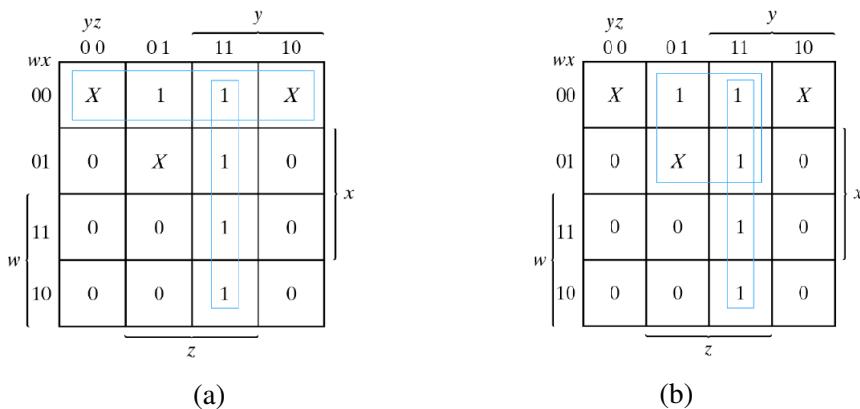


FIGURA 7. Mapas de Karnaugh (mitérminos) para simplificar una función de 4 variables incompleta, (a) una elección, (b) otra elección

Podemos realizar otra elección de términos “no importa”, como la siguiente:

- Si tomamos 0 y 2 como ceros y 5 como 1, el nuevo implicante primo es: $\bar{w} \cdot z$.
- De este modo, la suma mínima es:

$$F = y \cdot z + \bar{w} \cdot x$$

Esta función es diferente de la anterior pero responde a la misma especificación incompleta de F .

La Figura 7 (b) muestra esta elección.

2.1.5.4. *Observaciones y conclusiones.*

El método gráfico puede extenderse hasta 6 variables aunque no se expone en esta guía por no considerarlo necesario para cumplir los objetivos de esta guía.. De una manera sucinta, podemos resaltar las siguientes conclusiones:

- La simplificación se consigue sustituyendo la expresión canónica por la expresión mínima (puede ser la suma o el producto, la más sencilla de las dos).
- Para número de variables de la función mayor que seis, no se suelen usar los mapas de Karnaugh, sino métodos iterativos por computador.

2.2. PUERTAS LÓGICAS

2.2.1. **Introducción.**

El álgebra booleana nos permite diseñar funciones lógicas. Para aprovechar estas funciones en el diseño digital, necesitamos circuitos digitales que implementen las operaciones booleanas. Estos circuitos se llaman puertas lógicas y permiten generar tensiones según los valores (tomados como variables booleanas) de tensión aplicados al circuito. La relación entre las tensiones aplicadas y las tensiones que generan las puertas lógicas sigue la que existe entre los operandos y los resultados de la operaciones del álgebra de Boole. Es decir, las puertas lógicas realizan las operaciones del álgebra de Boole.

Según esto, las funciones que hemos diseñado según las reglas del álgebra de Boole se pueden convertir en circuitos digitales que implementan las mismas funciones, simplemente sustituyendo las variables por tensiones (entre conductores y tierra) y los operadores por puertas lógicas correspondientes a cada operador: suma, producto y negación.

2.2.2. **Puertas Lógicas.**

El elemento básico de la tecnología digital es el interruptor, es decir, un elemento que impide o no la circulación de corriente eléctrica (intensidad) por un conductor. La circulación o interrupción de intensidad por un conductor produce diferentes valores de tensión en los elementos conectados a ese conductor (relación tensión-intensidad).

Actualmente, el elemento más utilizado como interruptor es el transistor MOSFET (ver Figura 8), que permite interrumpir la corriente que lo atraviesa en función del valor de la señal de tensión aplicada al terminal de control. El transistor MOSFET tiene tres terminales, de los que dos actúan como entrada y salida para la intensidad (drenador D y fuente S) y el tercero (puerta G)

controla la conducción o no (corte) de la intensidad. La tensión aplicada en la puerta controla el estado del interruptor que constituye el transistor MOSFET, de modo que, cuándo ésta es baja, la intensidad que lo atraviesa es nula.

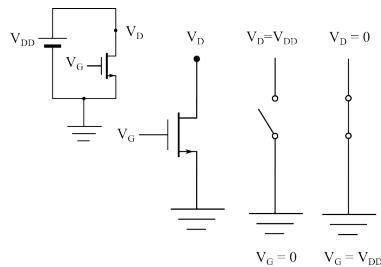


FIGURA 8. Transistor MOSFET como interruptor

La tecnología basada en este transistor es compleja, pero podemos entender el fundamento de las puertas lógicas mediante unos sencillos ejemplos de circuitos con transistores MOSFET. El circuito más sencillo (ver Figura 9) se compone de un único transistor que actúa sobre la tensión de entrada produciendo una única tensión de salida. Si consideramos los dos posibles valores de tensión a la salida, con el interruptor abierto ($V_x=0$) o cerrado ($V_x=\text{tensión alta}$) podemos ver cómo la relación entre las tensiones a la entrada y a la salida es la que existe entre el operando y el resultado de la operación booleana negación lógica.

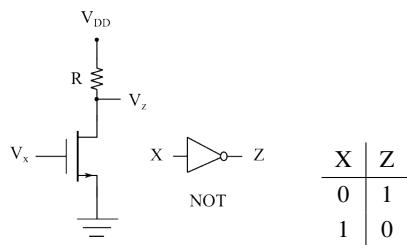


FIGURA 9. Circuito NOT

Utilizando tres transistores es posible construir circuitos que realizan las otras dos operaciones del álgebra de Boole. En estos circuitos tenemos dos tensiones de entrada (V_x y V_y) y podemos observar cómo considerando las cuatro posibles combinaciones para estos dos valores de tensión, podemos obtener en la tensión de salida V_z la función producto lógico de las dos tensiones de entrada, utilizando dos transistores MOSFET en serie (ver Figura 10).

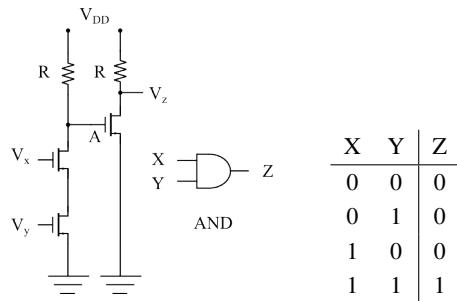


FIGURA 10. Circuito AND

Usando otro circuito de tres transistores, pero ahora con dos transistores en paralelo (ver Figura 11), se consigue un circuito en el cual la relación entre las tensiones de entrada y de salida es la de la operación suma lógica del álgebra de Boole.

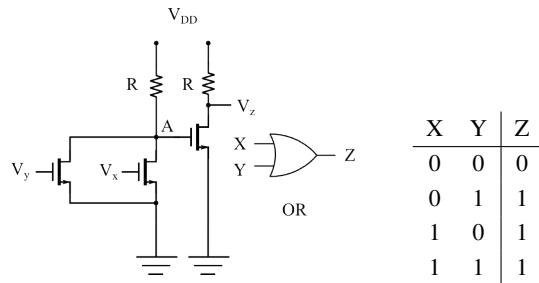
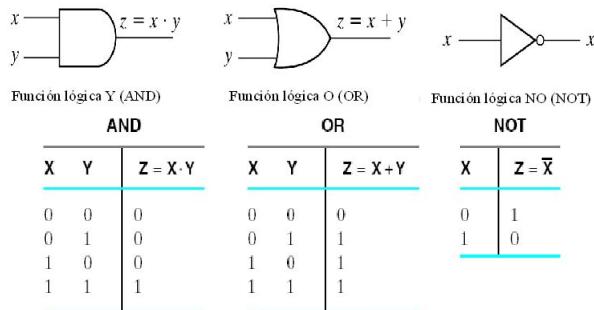


FIGURA 11. Circuito OR

Las puertas lógicas básicas se corresponden con las tres operaciones del álgebra booleana.



CUADRO 12. Símbolos de las puertas lógicas básicas y sus tablas de verdad

Cada variable booleana es sustituída por una señal de tensión: a 0 le corresponde la tensión baja (L), y a 1 la tensión alta (H)

La Figura 12 muestra las señales de entrada y de salida de circuitos digitales representadas frente al tiempo (cronogramas)

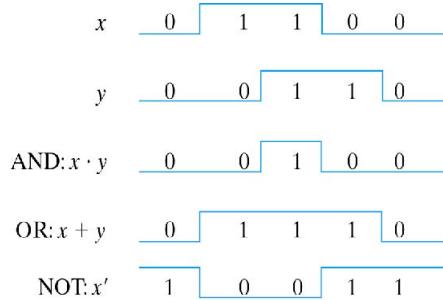


FIGURA 12. Cronograma de las puertas lógicas básicas

Por su utilidad, o por facilidad de fabricación, se han diseñado otras puertas lógicas que no son operadores básicos del álgebra booleana. La Figura 13 muestra estas puertas lógicas (4) de amplio uso en electrónica digital.

Nombre	Símbolo	Expresión algebraica	Tabla de verdad															
NAND		$F = (xy)'$	<table border="1"> <thead> <tr> <th>x</th><th>y</th><th>F</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = (x + y)'$	<table border="1"> <thead> <tr> <th>x</th><th>y</th><th>F</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
OR exclusivo (XOR)		$F = xy' + x'y = x \oplus y$	<table border="1"> <thead> <tr> <th>x</th><th>y</th><th>F</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
NOR exclusivo (equivalencia)		$F = xy + x'y' = (x \oplus y)'$	<table border="1"> <thead> <tr> <th>x</th><th>y</th><th>F</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

FIGURA 13. Símbolos de otras puertas lógicas y sus tablas de verdad

Hay un último dispositivo, que no podría considerarse propiamente como una puerta lógica, denominado buffer; no tiene función aparente, se usa para mantener los valores digitales de tensión e intensidad a través de sucesivas puertas lógicas. La Figura 14 muestra este dispositivo.

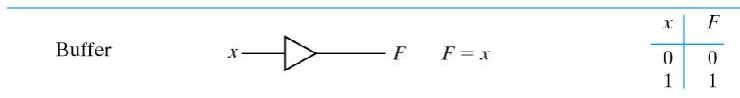


FIGURA 14. Símbolo del Buffer y su tabla de verdad

2.2.3. Síntesis de Circuitos Lógicos.

Cualquier expresión del álgebra de Boole se puede transformar en circuito mediante las puertas lógicas, sustituyendo cada operador de Boole por la puerta lógica correspondiente. Cuanto más sencilla sea la expresión, menor será el número de puertas y más sencillo, rápido y eficaz el circuito.

La Figura 15 muestra las expresiones algebraicas de dos funciones lógicas y su representación circuital por medio de puertas lógicas básicas.

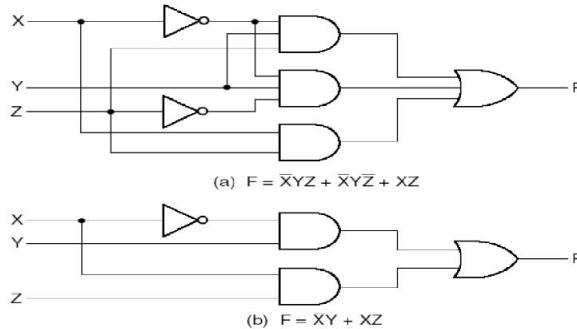


FIGURA 15. Síntesis de circuitos lógicos

2.2.3.1. Circuitos de dos niveles.

Si la expresión algebraica es de suma de productos o de producto de sumas, sólo hay dos puertas entre entrada y salida del circuito.

De forma genérica y atendiendo a las leyes de De Morgan, en el caso de suma de productos se puede implementar todo el circuito sólo con puertas NAND y en el caso de producto de sumas, se puede obtener el circuito sólo con puertas NOR, es decir, las puertas NAND y NOR son capaces de implementar por separado todas las operaciones del álgebra de Boole.

Esto se muestra en la Figura 16. La Figura 16(a) representa un circuito de dos niveles con puertas básicas. Si en la salida de las dos puertas AND se añaden dos inversores (negaciones), vease la Figura 16(b), la salida no cambia (vease la Tabla 3), de esta forma las puertas AND se han convertido en NAND y las dos negaciones en las entradas de la puerta OR convierten a esta puerta

en NAND (genéricamente por De Morgan: $\bar{X} + \bar{Y} = \bar{X} \cdot \bar{Y}$). La Figura 16(c) muestra el circuito resultante.

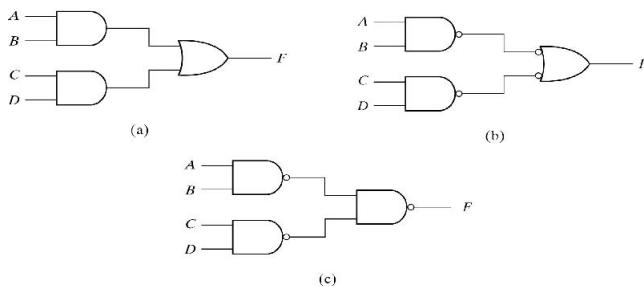


FIGURA 16. Circuito de dos niveles

2.2.3.2. Conjunto completo de puertas lógicas.

Un conjunto de puertas lógicas completo es aquel con el que se puede implementar cualquier función lógica. Las puertas NAND y NOR son capaces de implementar por separado todas las operaciones del álgebra de Boole, luego son un conjunto completo de puertas lógicas.

La Figura 17 muestra las diferentes equivalencias entre las puertas NAND y NOR en la ejecución de funciones AND y OR.

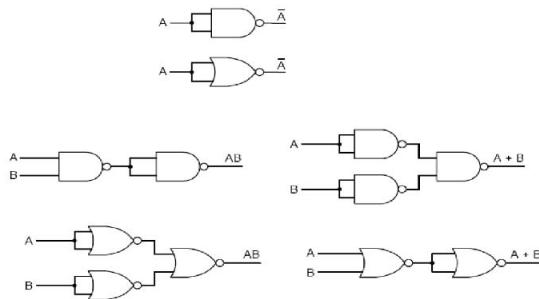


FIGURA 17. Equivalencias NAND y NOR

2.2.4. Tiempo en Circuitos Lógicos.

En la naturaleza que nos rodea, todos los procesos que en ella suceden presentan una velocidad de propagación, es decir, requieren de un tiempo para presentar cambios en su estado. La velocidad más rápida conocida es la de la luz; el resto de procesos en la naturaleza, presentan velocidades inferiores.

En los circuitos lógicos, los cambios en las variables requieren de un tiempo para reflejarse en la función de salida debido a la estructura física de los dispositivos.

2.2.4.1. *Tiempo de retardo.*

Al tiempo que pasa desde que cambia una entrada hasta que cambia la salida del circuito, le llamamos tiempo de retardo t_P .

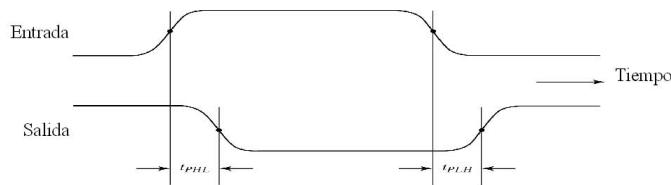


FIGURA 18. Definición de t_P

Cuando la señal cambia de L a H, se denomina t_{PLH} y si cambia de H a L, se denomina t_{PHL} .

2.2.4.2. *Riesgos.*

Cuando la salida de un circuito es la entrada de otro, el tiempo de retardo se acumula. Debido a esto, algunos circuitos pueden producir valores erróneos de la función durante pequeños intervalos de tiempo. Estos valores transitorios se llaman riesgos y se producen cuando los trayectos desde las variables hasta la función tienen diferentes tiempos de retardo al recorrer diferente número de puertas lógicas.

La Figura 19 muestra un circuito con riesgos debido a la presencia de una puerta inversora para la señal x_2 , teniendo en cuenta que la señal x_2 está presente en otra puerta sin pasar por la inversión, es decir, sin verse afectada por el retardo introducido por la puerta NOT.

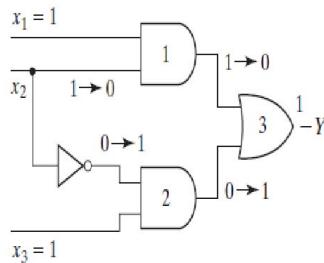


FIGURA 19. Circuito con riesgos

La ecuación que rige este circuito es: $Y = x_1 \cdot x_2 + \bar{x}_2 \cdot x_3$

Cuando x_2 cambia de 1 a 0, las dos entradas de la puerta OR tienen diferente retardo debido al retardo introducido por la puerta NOT. Por ello hay un intervalo en el que ambas son 0: Y vale 0 durante el intervalo de retardo de la puerta NOT (consideramos, por simplificar, que las puertas AND y OR no presentan retardos). La Figura 20 muestra el instante donde Y pasa a valer 0 debido al retardo de la puerta NOT.

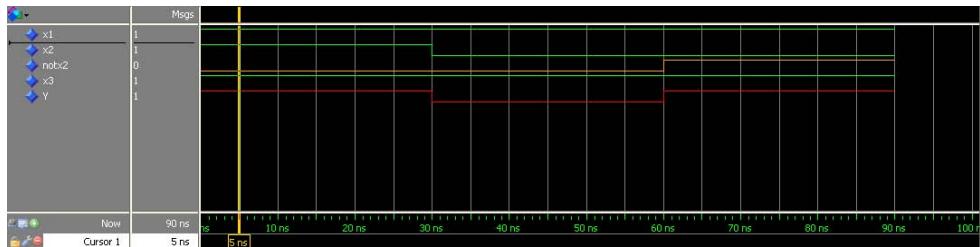


FIGURA 20. Riesgo en Y

Por medio de los mapas de Karnaugh se pueden detectar los riesgos e introducir los términos redundantes (se añaden los términos a la suma) que eliminan el riesgo. La Figura 21(a) muestra los minterms y su simplificación por Karnaugh que generan la función Y . Se observa que: los minterms 5 y 7 son adyacentes, están sin simplificar y difieren en la variable x_2 (m_5 con \bar{x}_2 y m_7 con x_2), es decir, cuando x_2 varía de 1 → 0, o de 0 → 1, la función Y pasa de tener presente el minterm m_5 al minterm m_7 o viceversa. Es entre estas dos celdas donde se produce el riesgo.

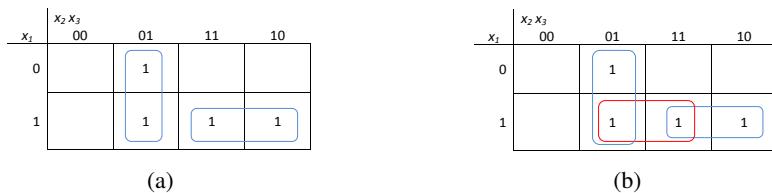


FIGURA 21. (a) Detección de los riesgos y (b) su eliminación por Karnaugh

La Figura 21(b) muestra la introducción de un término redundante $x_1 \cdot x_3$ que es el que elimina el riesgo. El término nuevo vale 1 mientras se produce el cambio de valor: Y sigue valiendo 1.

De forma genérica, cuando hay dos minterms adyacentes y no están simplificados, representan un riesgo pudiéndose simplificar para obtener un término redundante que elimine el riesgo.

La Figura 22 muestra el circuito con riesgos corregidos: $Y = x_1 \cdot x_2 + \bar{x}_2 \cdot x_3 + x_1 \cdot x_3$

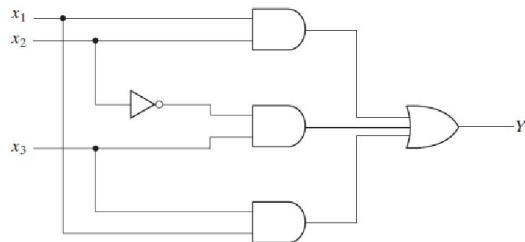


FIGURA 22. Circuito con riesgos corregidos

2.2.5. Lenguaje de Descripción del Hardware: VHDL.

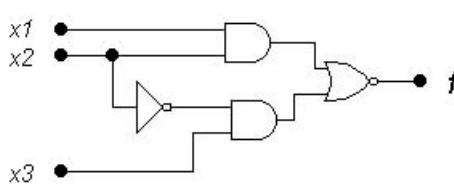
Existe otro método para representar los circuitos digitales: el Lenguaje de Descripción de Hardware (HDL: Hardware Description Language). Se trata de un texto que describe una o varias funciones lógicas. El fichero de texto se usa para la programación (configuración) o fabricación de un circuito integrado que cumple la función descrita.

El lenguaje de descripción de hardware más utilizado actualmente es el VHDL (Very High-Speed Integrated Circuit Hardware Description Language). Otro lenguaje de descripción hardware, ampliamente utilizado, es Verilog.

El texto VHDL se estructura en tres partes:

1. Librería: Especifica qué librerías y qué paquetes, de estas librerías, se van a utilizar.
2. Entidad: Especifica cuáles son las variables, su tipo (bit o vector) y si son de entrada, salida o bidireccionales.
3. Arquitectura: Se describen las funciones a realizar entre las variables definidas en la entidad.

La Figura 23 muestra un ejemplo de circuito y su descripción VHDL.



```

LIBRARY ieee;
USE      ieee.std_logic_1164.all;

ENTITY ejemplo IS
PORT( x1, x2, x3 : IN  BIT ;
      f          : OUT BIT );
END ejemplo ;

ARCHITECTURE LogicFunc OF ejemplo IS
BEGIN
      f <= (x1 AND x2) OR (NOT x2 AND x3) ;
END LogicFunc ;

```

FIGURA 23. Circuito ejemplo y su descripción en VHDL

En los Apéndices A y B se muestran diversos circuitos descritos en VHDL relativos a circuitos combinacionales (Capítulo 3) y secuenciales (Capítulo 4).

EJERCICIOS

- Ejercicio 1: Simplifica todo lo posible las siguientes expresiones:

 1. $Z = \bar{A} \cdot B \cdot C + \bar{A}$
 2. $Z = A + A \cdot B$
 3. $Z = A \cdot (B + C \cdot (B + A))$
 4. $Z = \bar{A} \cdot \bar{B} + A \cdot B + A \cdot \bar{B}$

Ejercicio 2: Comprueba las siguientes afirmaciones respecto a la suma exclusiva:

 1. $X \oplus \bar{X} = 1$
 2. $X \oplus 0 = X$
 3. $X \oplus 1 = \bar{X}$
 4. $X \oplus X = 0$

Ejercicio 3: Escribe la tabla de la verdad de las siguientes funciones lógicas:

- Ejercicio 4: Define por su expresión canónica las siguientes funciones:

 1. Función F que vale 1 si el número de 1s que aparecen en sus tres variables es mayor que el número de 0s.
$$F(A, B, C) = \Sigma m(1, 1, 1)$$
 2. Función lógica F de cuatro variables, que vale 1 cuando el número introducido en sus variables pertenezca al código BCD de 4 bits.
$$F(A, B, C, D) = \Sigma m(1, 1, 1, 1)$$
 3. Función F de cuatro variables A_3, A_2, A_1, A_0 que valga 0 sólo si el valor expresado por $A_3A_2A_1A_0$ es mayor que 10 o menor que 5.
$$F(A_3, A_2, A_1, A_0) = \Sigma m(0, 0, 0, 0, 1, 1, 1, 1)$$

Ejercicio 5: Repite el ejercicio 4, obteniendo la expresión mínima para las tres funciones

 1. $F(A, B, C) =$
 2. $F(A, B, C, D) =$

3. $F(A_3, A_2, A_1, A_0) =$

Ejercicio 6: Para las siguientes funciones, define la expresión mínima (suma o producto):

1. $F(A, B, C, D) = \Sigma m(1, 4, 5, 6, 7, 14, 15)$

$$F(A, B, C, D) =$$

2. $F(A, B, C, D) = \Sigma m(0, 1, 2, 4, 5, 6, 8, 10, 12)$

$$F(A, B, C, D) =$$

3. $F(A, B, C, D) = \Sigma m(0, 3, 5, 7, 8, 9, 10, 12, 13) + \Sigma d(1, 6, 11, 14)$

$$F(A, B, C, D) =$$

4. $F(A, B, C, D) = \Sigma m(1, 3, 4, 5, 11, 12, 13)$

$$F(A, B, C, D) =$$

5. $F(A, B, C, D) = \Sigma m(0, 2, 4, 6, 7, 8, 10, 13, 15)$

$$F(A, B, C, D) =$$

6. $F(A, B, C, D) = \Sigma m(2, 4, 8, 10, 11, 12)$

$$F(A, B, C, D) =$$

Ejercicio 7: Función lógica $Z_{A < B}$ que vale uno sólo si el número binario de dos bits $A(A_1, A_0)$ es menor que el número binario de dos bits $B(B_1, B_0)$, y función lógica $Z_{A > B}$ que vale uno sólo si el número binario de dos bits A es mayor que el número binario de dos bits B . Obtén su expresión mínima.

$$Z_{A < B}(A_1, A_0, B_1, B_0) =$$

$$Z_{A > B}(A_1, A_0, B_1, B_0) =$$

Ejercicio 8: Define una función Z que vale 1 sólo si el número en complemento a dos expresado por las cuatro variables de entrada E_3, E_2, E_1, E_0 , está entre -5 y 5, incluídos el 0 y ambos números. Obtén su expresión mínima.

$$Z(E_3, E_2, E_1, E_0) =$$

Ejercicio 9: Para las siguientes funciones, define la expresión mínima (suma o producto):

1. $F(A, B, C, D) = \Sigma m(0, 1, 2, 3, 7, 10, 14, 15)$

$$F(A, B, C, D)_m =$$

2. $F(A, B, C, D) = \Sigma m(0, 1, 5, 7, 8, 10, 14, 15)$

$$F(A, B, C, D)_m =$$

$$F(A, B, C, D)_M =$$

3. $F(A, B, C, D) = \Sigma m(0, 2, 6, 8, 10, 15) + \Sigma d(4, 9, 12, 13)$

$$F(A, B, C, D)_m =$$

$$F(A, B, C, D)_M =$$

Ejercicio 10: Utilizando puertas AND, OR y NOT, diseña un sencillo sistema de alarma para el cinturón de seguridad del coche, el cual

detecta cuándo el interruptor de arranque se ha activado y el cinturón de seguridad no está abrochado

Ejercicio 11: Utilizando puertas AND, OR y NOT, diseña un sencillo sistema de detección de intrusos en una habitación de una casa. Para ello tenemos sensores en dos ventanas y en la puerta. Estos sensores producen una salida de nivel alto cuando se abre la puerta (o ventana).

Ejercicio 12: Obtén la forma canónica de las siguientes expresiones:

1. $f_1 = \overline{A} \cdot \overline{C} + A \cdot \overline{C} \cdot D + \overline{A} \cdot B \cdot C \cdot D$
2. $f_2 = \overline{A} \overline{C} \overline{D} + AC + ABC$
3. $f_3 = \overline{A}B + AC$
4. $f_4 = (\overline{B} + D)(\overline{B} + A)(C + D)(D + A)$
5. $f_5 = (X_1 + X_3)(\overline{X}_3 + X_2 \overline{X}_4) + (X_3 X_5 + (\overline{X}_1 + X_4)(X_2 + \overline{X}_3))$

Ejercicio 13: Representa y simplifica las siguientes funciones mediante mapas de Karnaugh:

1. $f_1 = (1, 3, 9, 10, 12, 13, 14, 15)m$
2. $f_2 = (0, 2, 3, 4, 5, 7, 8, 10, 11)m$
3. $f_3 = (1, 6, 7)m$
4. $f_4 = ((0, 1, 6)M$
5. $f_5 = (0, 1, 2, 3, 7, 8, 9, 11, 15)m + K(6, 12)$
6. $f_6 = (3, 6, 7, 8, 10)m + K(12, 13, 14)$
7. $f_7 = (0, 1, 4, 5, 7, 8, 10, 15)m + K(2, 6, 14)$
8. $f_8 = (0, 1, 4, 5, 6, 7, 12, 13, 14, 16, 17, 28, 29)m + K(10, 11, 22, 23, 25, 26, 30, 31)$

Ejercicio 14: Implementar las expresiones siguientes mediante lógica NAND de 2 entradas:

1. $f_1 = A \cdot B \cdot C$
2. $f_2 = A \cdot B \cdot C + D \cdot E$
3. $f_3 = A \cdot B \cdot C + \overline{D} + \overline{E}$
4. $f_4 = (\overline{A} \cdot \overline{B}) + (\overline{C} \cdot \overline{D})$
5. $f_5 = (A + B) \cdot (C + D)$
6. $f_6 = A \cdot B \cdot (C \cdot ((\overline{D} \cdot \overline{E}) + (\overline{A} \cdot \overline{B})) + (\overline{B} \cdot \overline{C} \cdot \overline{E}))$
7. $f_7 = B \cdot (C \cdot \overline{D} \cdot E + \overline{E} \cdot F \cdot G) \cdot ((\overline{A} \cdot \overline{B}) + C)$

Ejercicio 15: Los semáforos de un cruce están gobernados por las condiciones:

1. Si hay un coche por la calle A, el semáforo de esa calle está en verde.
2. Si hay un coche por la calle B, el semáforo de esa calle está en verde si no hay coches en A.
3. Si hay un coche por la calle C, el semáforo de esa calle está en verde si no hay coches ni en A ni en B.
4. En ausencia de coches todos están en rojo.

A través de la tabla de la verdad, expresa la función lógica correspondiente.(rojo=1)

- Ejercicio 16: Diseña un circuito lógico con cuatro variables de entrada que sólo genera un 1 en la salida cuando tres variables de entrada son 1.
- Ejercicio 17: En una empresa hay 4 socios que tienen repartidas las acciones de la forma siguiente:
 $A = 35\% B = 30\% C = 25\% D = 10\%$
Describir una función lógica que indique, a la hora de votar una propuesta, si saldrá ésta adelante o no.
- Ejercicio 18: En una determinada planta de procesamiento químico, se usan tres diferentes elementos químicos líquidos en el proceso de fabricación. Los tres elementos químicos se almacenan en tres tanques diferentes. Un sensor de nivel en cada tanque genera una tensión a nivel alto cuando el nivel de líquido del tanque cae por debajo de un punto especificado.
Diseña un circuito para monitorizar el nivel del elemento químico en cada tanque, que indique cuándo el nivel de dos de los tanques cae por debajo del punto especificado.
- Ejercicio 19: Un circuito lógico tiene 5 entradas y 1 salida. 4 entradas representan un dígito decimal y la quinta es de control. Cuando ésta, está en 0 lógico, la salida será 0 lógico, si el número decimal es par., y 1 si es impar. Cuando el control está a 1, la salida será 0 cuando la entrada sea múltiplo de 3. Diseñar el circuito.
- Ejercicio 20: Diseña un circuito combinacional que gobierne el sistema de climatización de un recinto de acuerdo con las siguientes especificaciones:
 Cuando la temperatura interior sea superior a 25°C, se debe encender la refrigeración y cuando se inferior a 15°C, se debe poner en marcha la calefacción.
 Si la temperatura exterior es inferior a la interior en el primer caso del apartado a), es decir, en el caso de refrigeración, o superior a la interior en el segundo caso (calefacción), además del sistema correspondiente (refrigeración/calefacción), se deben poner en marcha los ventiladores de entrada de aire exterior.
 En días festivos, el sistema debe estar parado en cualquier condición. Implementar el circuito mediante puertas NAND de 2 y 3 entradas.

Capítulo 3

BLOQUES COMBINACIONALES

No mires a lo lejos, descuidando lo que tienes cerca.(Eurípides de Salamina)

RESUMEN. En capítulos anteriores se han sentado las bases del Álgebra de Boole así como se han introducido las puertas lógicas que nos permiten realizar funciones en base a la combinación de diferentes variables binarias. En este capítulo, basándonos en las puertas lógicas, se desarrollarán entidades funcionales de amplio uso en la confección de sistemas digitales combinacionales.

3.1. INTRODUCCIÓN

3.1.1. Sistemas Combinacionales Lógicos.

De manera genérica podemos decir que los circuitos combinacionales son aquéllos en los que una serie de variables X_i , definen una serie de funciones Z_j . Las principales restricciones que caracterizan el comportamiento funcional son las siguientes:

1. Cada combinación de valores de X_i define un sólo valor de Z_j .
2. Los valores de Z_j sólo pueden cambiar cuando cambian los valores de X_i .

La Figura 1 muestra un bloque combinacional genérico.

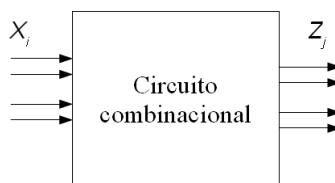


FIGURA 1. Circuito combinacional

En este capítulo se diseñarán sistemas combinacionales que han sido integrados por diversos fabricantes de dispositivos de estado sólido y están disponibles en el mercado. Genéricamente se conocen estos dispositivos como Circuitos Integrados. Cualquiera de estos dispositivos integrados va a poder ser incorporado en los diseños de funcionalidades más complejas.

3.1.2. Clasificación de los Circuitos Integrados.

La clasificación más general está basada en lo que se denomina “Escala de integración”. En ella intervienen los siguientes criterios:

1. Los circuitos integrados se clasifican por el número de puertas que forman parte de las funciones que realizan
2. Existen 4 grupos, llamados escalas de integración:
 - a) SSI (menos de 10 puertas).
 - b) MSI (entre 10 y 100 puertas).
 - c) LSI (entre 100 y 1000 puertas).
 - d) VLSI (a partir de 1000 puertas).

Los circuitos combinacionales se encuentran, fundamentalmente, en la escala MSI. Los microprocesadores y las memorias son circuitos VLSI con millones de puertas.

3.1.3. Diseño Jerárquico.

El principal problema que se encuentra un diseñador de sistemas electrónicos es cómo estructurar un diseño, la pregunta fundamental es ¿cómo se puede diseñar un circuito con miles de puertas lógicas? Para ello, vamos a exponer una serie de ideas básicas para enfrentarnos a este tipo de diseños:

1. La solución es dividir el circuito en funciones más pequeñas, de modo que podamos resolver éstas por el álgebra booleana.
2. Organizando los bloques funcionales, podemos resolver el problema más grande.
3. Este enfoque se denomina diseño jerárquico, porque se establecen niveles de diseño, el más bajo de los cuales se resuelve mediante álgebra booleana y puertas lógicas y los demás mediante las funciones definidas en niveles inferiores.

La Figura 2 muestra los pasos seguidos, en un proceso de diseño jerárquico, para obtener un sistema que determine si una función de 9 bits presenta un número impar de unos.

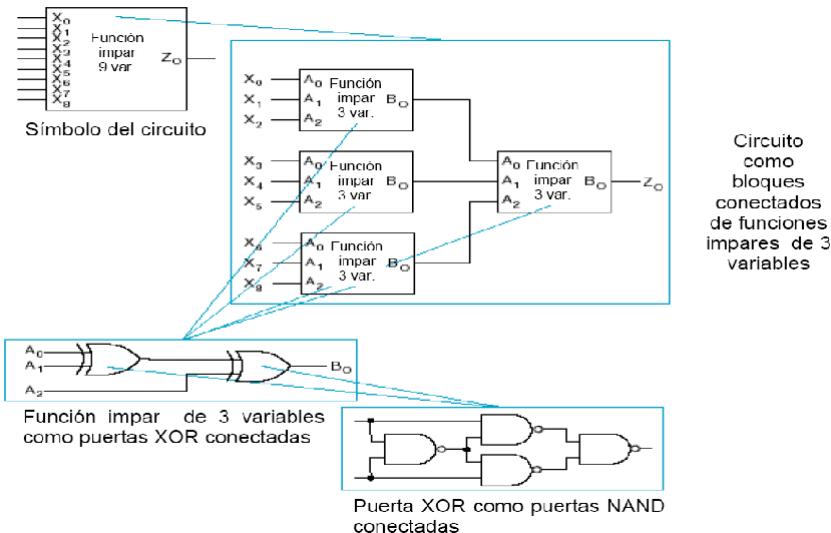


FIGURA 2. Diseño jerárquico

El circuito final tiene 9 variables, pero sólo hemos usado el álgebra de Boole para dos variables. Este circuito se presenta comercialmente con el código 74HC280¹.

Este modo de diseño se aplicará en este capítulo en el diseño de alguno de los bloques funcionales, siendo también válido para acometer diseños superiores que deban incorporar estos bloques funcionales.

3.2. MULTIPLEXOR

La finalidad de un multiplexor consiste en elegir una entre varias variables. La funcionalidad es la siguiente:

1. Según el valor de una variable de control, se elige una de entre varias variables de entrada al bloque.
2. La salida del bloque será la variable elegida.

La Figura 3 muestra un multiplexor que da servicio a 3 entradas por medio de una señal de control.

¹Nota aclaratoria: De forma genérica, los primeros dos dígitos (74 o 54) hacen referencia a características de voltaje y temperatura, las letras HC hacen referencia a la tecnología de fabricación (HC, LS, AL, etc.) que afecta al consumo, velocidad, etc., y los dos o tres últimos dígitos, en este caso tres (280), se corresponden con la funcionalidad del dispositivo.

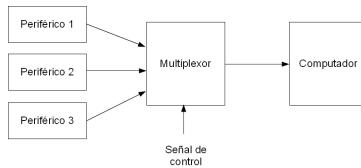


FIGURA 3. Multiplexor

3.2.1. Multiplexor 2 a 1.

Un Multiplexor 2 a 1 consta de una entrada de selección, S , dos entradas binarias, I_0I_1 , y una salida Y . Cuando $S = 0$, la salida Y presentará la entrada I_0 y cuando $S = 1$, Y presentará el valor de la entrada I_1 . La Tabla 1 muestra el comportamiento del multiplexor 2 a 1.

CUADRO 1. Tabla de verdad del mux 2 a 1

S	I_0	I_1	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Aplicando Karnaugh obtenemos la ecuación siguiente:

$$Y = \bar{S} \cdot I_0 + S \cdot I_1$$

La Figura 4 muestra el esquema circuital, con puertas lógicas, de la ecuación resultante.

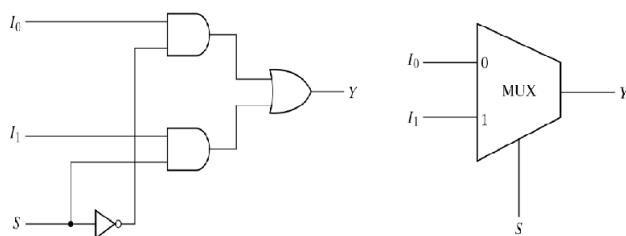


FIGURA 4. Multiplexor 2 a 1, puertas lógicas y bloque

Comercialmente, el circuito 74HC157 contiene cuatro bloques mux² 2 a 1, respondiendo todos a la misma señal de selección.

3.2.2. Multiplexor 4 a 1.

Un Multiplexor 4 a 1 consta de dos entradas de selección, S_1S_0 , cuatro entradas binarias, $I_0I_1I_2I_3$, y una salida Y . Cuando $S_1S_0 = 00$, la salida Y presentará la entrada I_0 , cuando $S_1S_0 = 01$, Y presentará el valor de la entrada I_1 y así sucesivamente. La Tabla 2 muestra el comportamiento del multiplexor 4 a 1:

CUADRO 2. Tabla de verdad del mux 4 a 1

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Observando la Tabla 2, la salida Y se expresará según la ecuación siguiente:

$$Y = \overline{S_1} \cdot \overline{S_0} \cdot I_0 + \overline{S_1} \cdot S_0 \cdot I_1 + S_1 \cdot \overline{S_0} \cdot I_2 + S_1 \cdot S_0 \cdot I_3$$

La Figura 5 muestra el esquema circuital, con puertas lógicas, de la ecuación resultante.

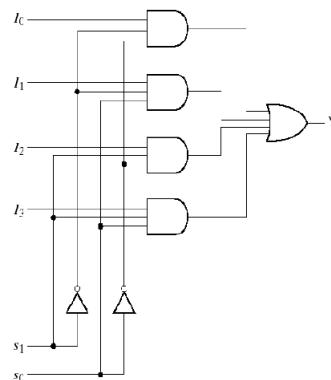


FIGURA 5. Mux 4 a 1

²Forma abreviada de expresar multiplexor

Para cualquier número de entradas 2^n , el número de señales de control es n , o dicho de otra forma, si se dispone de n señales de control, se pueden manejar 2^n señales.

El Anexo A muestra el desarrollo de este Multiplexor descrito por medio del lenguaje de descripción hardware VHDL.

3.2.3. Entrada de Habilitación en un Multiplexor.

La idea básica, al introducir una entrada de habilitación en un multiplexor, consiste en que la salida del mux presente la entrada seleccionada solo cuando el dispositivo esté habilitado. La decisión de introducir una entrada de habilitación se basa en las siguientes reflexiones:

1. Los bloques funcionales se usan como elementos de sistemas más grandes.
2. Esos sistemas combinan las funciones de estos bloques, usando sólo alguno de ellos en distintos momentos.
3. Para regular cuándo se usa cada bloque, se usa la entrada de habilitación.

Teniendo en cuenta estas reflexiones, cuando la entrada de habilitación está desactivada, el bloque no produce ninguna función, manteniendo la salida constante (a 0 o 1 según dispositivo). En general, la entrada de habilitación, suele nombrarse E o EN (abreviatura de ENABLE, habilitar en inglés).

La Tabla 3 muestra la funcionalidad de un Mux 4 a 1 con habilitación.

CUADRO 3. Tabla de verdad del mux 4 a 1 con habilitación

EN	S_1	S_0	F
0	X	X	0
1	0	0	I_0
1	0	1	I_1
1	1	0	I_2
1	1	1	I_3

Observando la Tabla 3, la salida Y se expresará según la ecuación siguiente:

$$Y = EN \cdot (\overline{S_1} \cdot \overline{S_0} \cdot I_0 + \overline{S_1} \cdot S_0 \cdot I_1 + S_1 \cdot \overline{S_0} \cdot I_2 + S_1 \cdot S_0 \cdot I_3)$$

La Figura 6 muestra el bloque funcional Mux 4 a 1 con entrada de habilitación.

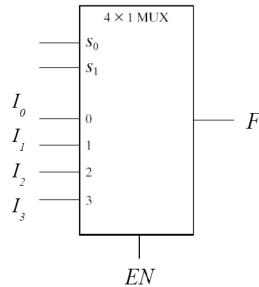


FIGURA 6. Mux 4 a 1 con habilitación

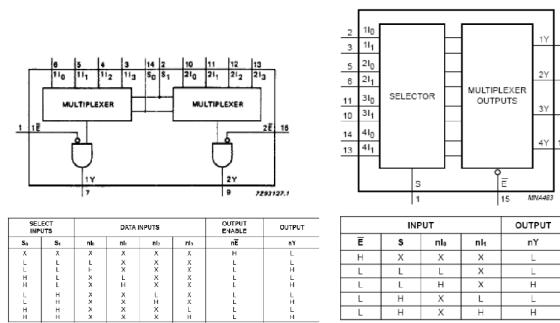
3.2.4. Aplicaciones de los Multiplexores.

3.2.4.1. Buses con Multiplexores.

Hemos visto el tratamiento de señales individuales por parte de los multiplexores, si tenemos en cuenta las siguientes consideraciones, se podrán realizar agrupaciones de señales para ser tratadas por multiplexores:

1. Un conjunto de señales eléctricas homogéneas se llama bus.
2. Los buses se utilizan para definir datos que supongan más de una variable binaria.
3. Tanto las entradas como las salidas de los multiplexores pueden ser buses con más de una variable binaria.
4. En ese caso, el número de variables de control será n si el cociente entre el número de señales de entrada y el número de señales de salida es 2^n (mux 4 a 1 $\Rightarrow 2^2$).

La Figura 7 muestra dos circuitos comerciales conteniendo multiplexores que permiten agrupaciones en buses.



74HC/HCT153

74AHC157

FIGURA 7. Circuitos integrados multiplexores

3.2.4.2. Implementación de funciones con multiplexores.

En el Capítulo 2 vimos como se podían implementar funciones de varias variables binarias por medio de tablas de verdad, expresándolas en sus formas canónicas (minterm o Maxterm) y posteriormente acudir a los mapas de Karnaugh para su simplificación. Ahora vamos a implementar funciones por medio de bloques funcionales como los multiplexores, atendiendo a las siguientes consideraciones:

1. La salida de un multiplexor se expresa siempre como suma de productos multiplicado por cada entrada.
2. La expresión canónica de cualquier función puede expresarse como una suma de productos.
3. Si utilizamos las señales de control como variables de entrada, podemos representar la expresión canónica de cualquier función.
4. Podemos usar las entradas del multiplexor para completar los minterminos con otra variable.
5. La salida del multiplexor representará la función buscada.

Veamos dos ejemplos aclaratorios del modo de proceder expuesto:

Ejemplo de función de 3 variables

Supongamos que queremos implementar la siguiente función, expresada en términos canónicos de suma de productos (minterm):

$$F = \bar{x} \cdot \bar{y} \cdot z + \bar{x} \cdot y \cdot \bar{z} + x \cdot y \cdot \bar{z} + x \cdot y \cdot z$$

Llevamos esta función a la tabla de verdad, organizando las filas de manera que dos, de las tres columnas x, y, z , representen 2 entradas de selección de un multiplexor y la otra represente una variable de entrada al multiplexor. La Tabla 4 muestra la separación con las variables x, y como entradas de selección de un mux 4 a 1 y la variable z como variable a ser utilizada en las entradas $I_0 I_1 I_2 I_3$.

CUADRO 4. Tabla de verdad de la implementación de una función de 3 variables con un mux de 4 a 1.

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Se observa que:

1. Cuando $x, y = 00$ (selección de I_0) si $z = 0, F = 0$ y si $z = 1, F = 1$, es decir, la entrada $I_0 = z$.
2. Cuando $x, y = 01$ (selección de I_1) si $z = 0, F = 1$ y si $z = 1, F = 0$, es decir, la entrada $I_1 = \bar{z}$.
3. Cuando $x, y = 10$ (selección de I_2) si $z = 0, F = 0$ y si $z = 1, F = 0$, es decir, la entrada $I_2 = 0$.
4. Cuando $x, y = 11$ (selección de I_3) si $z = 0, 1$ y si $z = 1, F = 1$, es decir, la entrada $I_3 = 1$.

Por lo tanto, el cableado de las señales en el mux 4 a 1, quedará tal y como se muestra en la Figura 8:

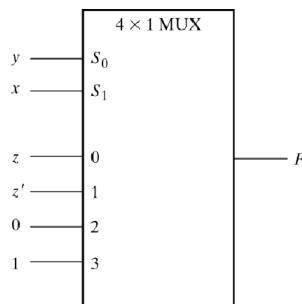


FIGURA 8. Implementación de una función de 3 variables por medio de un mux 4 a 1.

□ **Ejemplo de función de 4 variables**

Supongamos que queremos implementar la siguiente función, expresada en términos canónicos de suma de productos (minterm):

$$F = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D + \overline{A} \cdot \overline{B} \cdot C \cdot D + \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + A \cdot \overline{B} \cdot C \cdot D + \\ A \cdot B \cdot \overline{C} \cdot \overline{D} + A \cdot B \cdot \overline{C} \cdot D + A \cdot B \cdot C \cdot \overline{D} + A \cdot B \cdot C \cdot D$$

Llevamos esta función a la tabla de verdad, organizando las filas de manera que tres, de las cuatro columnas A, B, C, D , representen 3 entradas de selección de un multiplexor y la otra represente una variable de entrada al multiplexor. La Tabla 5 muestra la separación con las variables A, B , y C como entradas de selección de un mux 8 a 1 y la variable D como variable a ser utilizada en las entradas $I_0I_1I_2I_3I_4I_5I_6I_7$. La tabla 5 muestra la separación de las variables para implementar la función con un mux 8 a 1.

CUADRO 5. Tabla de verdad de la implementación de una función de 4 variables con un mux.

A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Se observa que:

1. Cuando $A, B, C = 000$ (selección de I_0) si $D = 0, F = 0$ y si $D = 1, F = 1$, es decir, la entrada $I_0 = D$.
2. Ídem con $A, B, C = 001$ (selección de I_1), la entrada $I_1 = D$.
3. Cuando $A, B, C = 010$ (selección de I_2) si $D = 0, F = 1$ y si $D = 1, F = 0$, es decir, la entrada $I_2 = \overline{D}$.

4. Cuando $A, B, C = 011$ (selección de I_3) si $D = 0, F = 0$ y si $D = 1, F = 0$, es decir, la entrada $I_3 = 0$.
5. Ídem con $A, B, C = 100$ (selección de I_4), la entrada $I_4 = 0$.
6. Cuando $A, B, C = 101$ (selección de I_5) si $D = 0, F = 0$ y si $D = 1, F = 1$, es decir, la entrada $I_5 = D$.
7. Cuando $A, B, C = 110$ (selección de I_6) si $D = 0, 1$ y si $D = 1, F = 1$, es decir, la entrada $I_6 = 1$.
8. Ídem $A, B, C = 111$ (selección de I_7), es decir, la entrada $I_7 = 1$.

La Figura 9 muestra el cableado del mux 8 a 1 que satisface esta implementación.

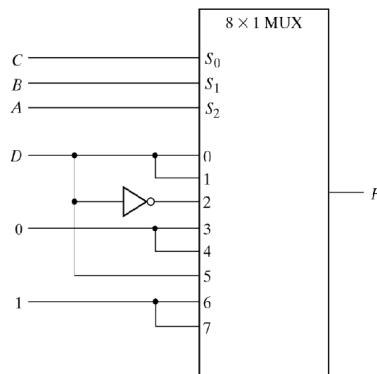


FIGURA 9. Implementación de una función de 3 variables por medio de un mux 8 a 1.

3.3. DEMULTIPLEXOR

La finalidad de un demultiplexor consiste en direccionar una variable hacia una de entre varias funciones. La funcionalidad es la siguiente:

1. Según el valor de una variable de control, se transfiere el valor de la variable de entrada al bloque a una de entre varias funciones.
2. Las salidas del bloque serán nuevas variables para otros tantos bloques.

La Figura 10 muestra un demultiplexor de una entrada que da servicio a 3 salidas por medio de una señal de control.

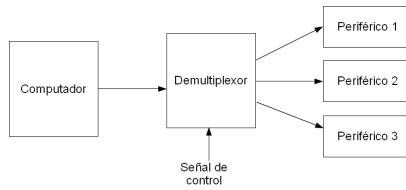


FIGURA 10. Demultiplexor

El Anexo A muestra el desarrollo de este Demultiplexor descrito por medio del lenguaje de descripción hardware VHDL.

3.3.1. Demultiplexor 1 a 4.

Un Demultiplexor 1 a 4 consta de dos entradas de selección, S_1S_0 , una entrada E , y cuatro salidas binarias, $D_0D_1D_2D_3$. Cuando $S_1S_0 = 00$, la salida D_0 presentará la entrada E , cuando $S_1S_0 = 01$, D_1 presentará el valor de la entrada E , y así sucesivamente. La Tabla 6 muestra el comportamiento del demultiplexor 1 a 4:

CUADRO 6. Tabla de verdad de un Demultiplexor 1 a 4

E	S_1	S_0	D_0	D_1	D_2	D_3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

De la Tabla 6 podemos extraer las siguientes ecuaciones que reflejan el comportamiento del demux³ 1 a 4:

$$\begin{aligned}
 D_0 &= E \cdot \overline{S_1} \cdot \overline{S_0} \\
 D_1 &= E \cdot \overline{S_1} \cdot S_0 \\
 D_2 &= E \cdot S_1 \cdot \overline{S_0} \\
 D_3 &= E \cdot S_1 \cdot S_0
 \end{aligned}$$

³abreviatura de demultiplexor

La Figura 11 muestra el esquema circuitual con puertas lógicas y el bloque funcional:

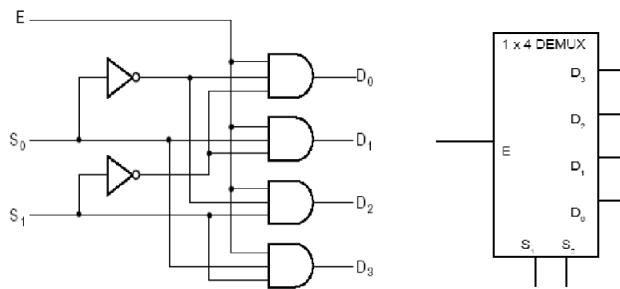


FIGURA 11. Demultiplexor 1 a 4: puertas lógicas y bloque

3.4. PUERTAS TRI-ESTADO

La existencia de puertas Tri-estado⁴ se debe a las siguientes consideraciones:

1. Los buses son a menudo compartidos por varios bloques funcionales, pero el uso de una sola línea por dos bloques a la vez puede llevar a un cortocircuito.
2. En ese caso, es preciso asegurar que sólo accedan al bus una pareja de bloques (emisor y receptor de la señal).
3. La conexión de un bloque con el bus se puede cortar eléctricamente (alta impedancia - Z) mediante una puerta tri-estado.

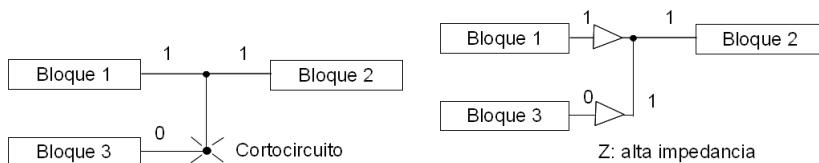


FIGURA 12. Esquema de bloques con puertas tri-estado

Las Figuras 13 y 14, muestran puertas tri-estado con señal de control tanto por alto como por bajo. Además, estas puertas pueden ser inversoras o no inversoras.

⁴Tri-state en inglés

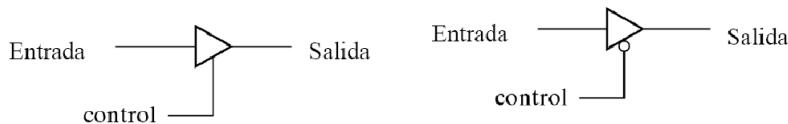


FIGURA 13. Puertas tri-estado no inversoras: control = 1, control = 0

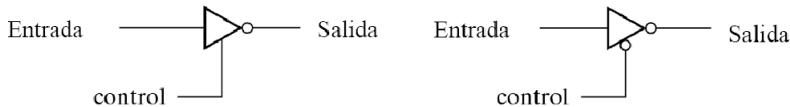


FIGURA 14. Puertas tri-estado inversoras: control = 1, control = 0

El Anexo A muestra el desarrollo de una puerta tri-estado descrita por medio del lenguaje de descripción hardware VHDL.

3.5. SISTEMAS COMBINACIONALES ARITMÉTICOS

Los sistemas combinacionales aritméticos son aquéllos que usan números en entradas o salidas y realizan alguna operación aritmética. Para el desarrollo de estos sistemas establecemos que:

1. Los números se representen mediante variables lógicas binarias, cuyos dos valores representan la cifra 0 o la cifra 1.
2. Las variables numéricas se representen mediante una letra mayúscula y una serie de subíndices para diferenciar cada cifra binaria.

La Figura 15 muestra el esquema de bloques de un circuito aritmético:

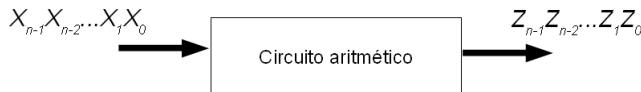


FIGURA 15. Circuito aritmético

CUADRO 7. Tabla de verdad de un semisumador

			x	y	s	c
	0	0	0	0	0	0
+	0	+	1	1	1	0
	0	0	0	1	1	0
					1	1
					1	1

3.5.1. Semisumador.

Veamos, a continuación, el desarrollo de un sumador básico que atiende a dos operandos (x, y) de un bit y muestra el resultado de su operación. Para ello debemos tener en cuenta las siguientes observaciones:

1. La suma aritmética de dos cifras binarias es una función lógica de dos variables binarias.
2. Para incluir todas las posibilidades de la suma, el resultado son dos funciones: s (suma) y c (acarreo).

La Tabla 7 muestra la relación entre los operandos, x e y , y el resultado de la suma, s y c :

La expresión algebraica de las dos funciones s y c de la suma es la siguiente:

$$\begin{cases} s = x \cdot \bar{y} + \bar{x} \cdot y = x \oplus y \\ c = x \cdot y \end{cases}$$

La Figura 16 muestra las ecuaciones anteriores expresadas mediante puertas lógicas:

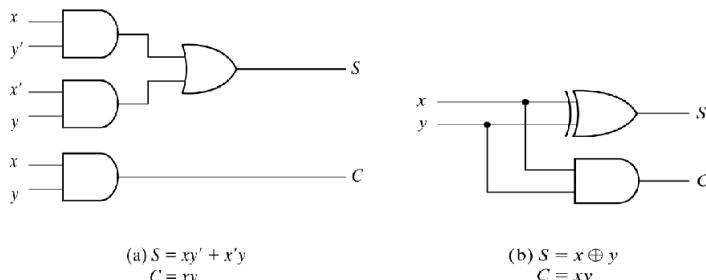


FIGURA 16. Implementación del semisumador

3.5.2. Sumador Completo.

Para que sea práctica la implementación del semisumador, debemos generalizarla a la suma de n bit. Esto nos lleva a la realización de un sumador completo o *full-adder* en donde deberemos tener en cuenta que:

1. Para realizar sumas de n bit, podemos realizar sumas de un bit para cada cifra de los dos sumandos.
2. En cada cifra sumaremos dos bit de cada operando y el acarreo de la cifra anterior.
3. Por ello definiremos una función que sume tres operandos de un bit: sumador completo.

La Figura 8 muestra un ejemplo de suma y la tabla de verdad de un sumador completo:

CUADRO 8. Tablas de verdad del sumador completo

Operación:			3	+	6	x	y	$z(c_{i-1})$	s	c_i
Operandos	Acarreos:		1	1	0	0	0	0	0	0
3			0	0	1	0	1	0	1	0
6	+		0	1	1	0	1	1	0	1
			1	0	0	1	0	1	0	1
						1	0	1	0	1
						1	1	0	0	1
						1	1	1	1	1

Simplificando la Tabla 8 por medios algebraicos o mapas de Karnaugh⁵, obtenemos las siguientes expresiones:

$$\begin{cases} s = z \oplus (x \oplus y) \\ c_i = (x \cdot y) + (x \oplus y) \cdot z \end{cases}$$

La Figura 17 muestra la implementación de un sumador completo como combinación de dos semisumadores:

⁵Según el método escogido, hay una cierta diferencia en la expresión final de c_i , la diferencia es entre el término $(x \oplus y)$ y $(x + y)$.

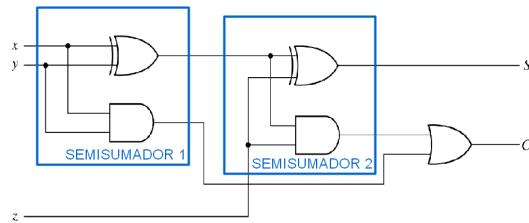


FIGURA 17. Sumador completo

La realización de un sumador que tenga en cuenta operandos con más de un bit, se realiza añadiendo en serie “cajas” de sumadores completos. Un ejemplo para la suma de operandos de 4 bits, se muestra en la Figura 18:

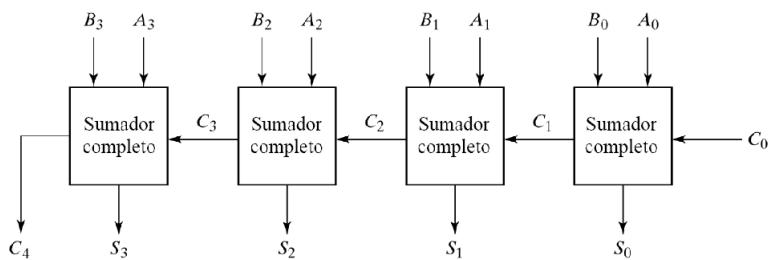


FIGURA 18. Sumador de operandos de 4 bits

De forma generalizada:

- Combinando n sumadores completos, se obtiene un sumador de n bits.
- Para obtener la última cifra S_{n-1} , hace falta el acarreo C_{n-1} , obtenido del bloque anterior, y éste del anterior a él.
- Este modo de proceder genera acumulación de retardos.

El Anexo A muestra el desarrollo de un Sumador completo, para operandos de 4 bits, descrito por medio del lenguaje de descripción hardware VHDL.

3.5.3. Restador.

Aritméticamente la resta presenta la misma importancia que la suma. Nuestro natural modo de proceder, cuando realizamos restas en base 10, no puede ser trasladado a un sistema de cálculo binario. Como ya se ha visto en el Capítulo 1, las operaciones de resta están sujetas a determinadas restricciones. Vamos a realizar ciertas observaciones a tener en cuenta para el desarrollo de un bloque restador:

1. La resta binaria puede definirse del mismo modo que la suma binaria.
2. Sin embargo, si el minuendo es menor que el sustraendo, produce un resultado negativo.
3. Si el resultado de la resta produce un número negativo, la resta se realiza invirtiendo el orden de los operadores, y el resultado se marca como negativo con una cifra binaria añadida (0 si es positivo y 1 si es negativo).
4. El sistema de resta en magnitud con signo añadido no se utiliza por presentar una complejidad innecesaria.
5. El sistema de resta en complemento a 1 presenta ventajas con respecto al anterior pero es más complejo que el complemento a 2.
6. El sistema de resta en complemento a 2 es el más adecuado para su implementación en un sistema digital.

La Tabla 9 muestra el principio de la operación de la resta con dos operandos de un bit:

CUADRO 9. Ejemplo básico de resta de 2 bits

$$\begin{array}{r}
 0 & 1 & 1 & 0 \\
 - 0 & - 0 & - 1 & - 1 \\
 \hline
 0 & 1 & 0 & 1
 \end{array}$$

3.5.3.1. Resta en complemento a dos.

En el Capítulo 1, se sentaron las bases de las operaciones aritméticas para desarrollar las operaciones de resta entre números binarios. El realizar la resta en complemento a 2, requiere prestar atención a los siguientes puntos:

1. La resta en complemento a dos consiste en aplicar el complemento a dos al sustraendo, y luego realizar la suma: $A - B = A + (-B)$.
2. La operación es $2^n - B$, y en la práctica consiste en sustituir 0s por 1s y viceversa (complemento a 1), y luego sumar 1 al resultado del complemento a 1.
3. El resultado es coherente con el sistema utilizado (si es negativo, está en complemento a dos) y permite unificar suma y resta en el mismo operador.
4. Si el resultado es negativo, realizando la misma operación de complemento a dos, se obtiene la magnitud absoluta.
5. Si el resultado es positivo, aparece siempre un acarreo en la última cifra, que descartamos.

La expresión circuital de un Restador-Sumador se muestra en la Figura 19 donde la señal M determina si es Resta ($M = 1$) o Suma ($M = 0$). Las puertas

XOR realizaran el complemento a 1 cuando $M = 1$ y con este valor, $c_0 = 1$ lo que produce la suma de 1 al complemento a 1.

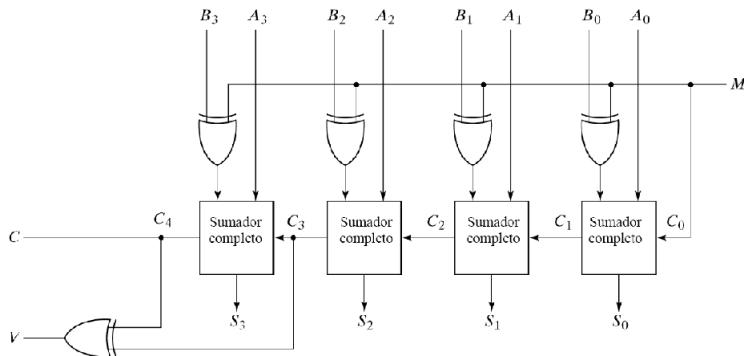


FIGURA 19. Restador-Sumador de cuatro bits

Las señales C y V tienen el siguiente significado:

1. La señal V indica si ha habido desbordamiento en la resta, es decir, si el resultado se puede expresar en 4 bit.
2. La señal C indica si ha habido desbordamiento en la suma.

El Anexo A muestra el desarrollo de un Restador-Sumador de 4 bits, descrito por medio del lenguaje de descripción hardware VHDL.

3.6. CODIFICADOR

1. El codificador produce como salida el número (código) correspondiente a la variable que está activada.
2. Si el código es el binario natural, el número de entradas es 2^n , siendo n el número de salidas.

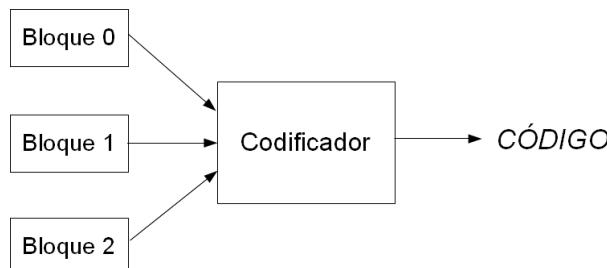


FIGURA 20. Codificador

3.6.1. Codificador con Prioridad.

Bla.

1. Si se activan dos señales al mismo tiempo, la salida debe corresponder sólo a una, por lo que se establece prioridad entre las entradas.
2. Normalmente, la entrada asociada al número mayor es la de mayor prioridad.
3. La salida adicional V indica si hay alguna entrada activada.

CUADRO 10. Tabla de verdad de un Codificador con prioridad

D_0	D_1	D_2	D_3	x	y	v
0	0	0	0	0	0	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

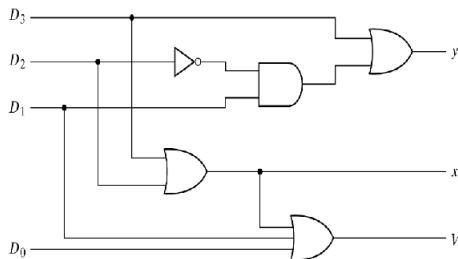


FIGURA 21. Esquema de bloques de un Codificador con prioridad

El Anexo A muestra el desarrollo de un Codificador con prioridad, descrito por medio del lenguaje de descripción hardware VHDL.

3.7. DECODIFICADOR

El Decodificador elige de entre varias funciones cuál se activa al aplicar en las variables de entrada el número (código) asignado a esa función. La funcionalidad presenta ciertas similitudes con el Demultiplexor.

Un circuito Decodificador con n variables de entrada tiene 2^n salidas, siendo el código binario natural.

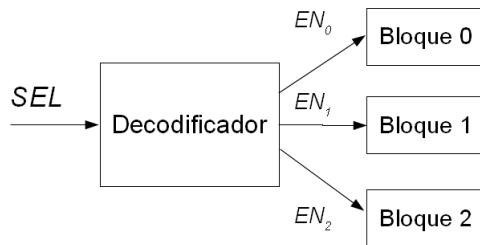


FIGURA 22. Decodificador

3.7.1. Decodificador de 2 bits con Entrada de Habilitación.

Veamos a continuación el desarrollo de un decodificador de 2 bits con entrada de habilitación. La entrada de habilitación presenta la misma funcionalidad que en el Multiplexor. La Tabla 11 muestra el comportamiento de este Decodificador:

CUADRO 11. Tabla de verdad del decodificador de 2 bits

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Observando la Tabla 11, al estar expresadas las salidas activas a 0, podemos expresar las salidas como Maxtérminos.

$$\begin{aligned}
 D_0 &= \overline{\bar{E} \cdot \bar{A} \cdot \bar{B}} = E + A + B \\
 D_1 &= \overline{\bar{E} \cdot \bar{A} \cdot B} = E + A + \bar{B} \\
 D_2 &= \overline{\bar{E} \cdot A \cdot \bar{B}} = E + \bar{A} + B \\
 D_3 &= \overline{\bar{E} \cdot A \cdot B} = E + \bar{A} + \bar{B}
 \end{aligned}$$

La Figura 23 muestra estas ecuaciones en su expresión circuital:

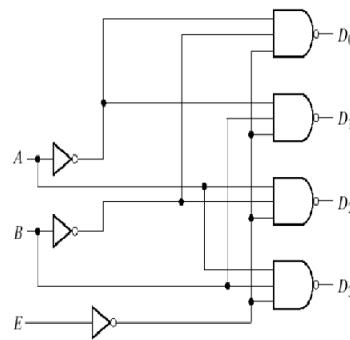


FIGURA 23. Decodificador de 2 bit con entrada de habilitación

El Anexo A muestra el desarrollo de un Decodificador descrito por medio del lenguaje de descripción hardware VHDL.

Nota: Queda para el lector el desarrollar un decodificador de 3 bits siguiendo las directrices mostradas con el de 2 bits.

3.7.2. Decodificador de 4 bits Mediante dos Decodificadores de 3 bits.

Atendiendo al concepto de diseño jerárquico, podemos diseñar decodificadores de mayor entidad basándonos en decodificadores que presenten un menor número de bits. La Figura 24 muestra el caso del diseño de un decodificador de 4 bits con dos decodificadores de 3 bits. En este caso, el diseño se ha realizado con entrada de habilitación (E) activa a “1”, es decir, cuando la variable w valga “0”, estará habilitado el decodificador superior ($D_7..D_0$).

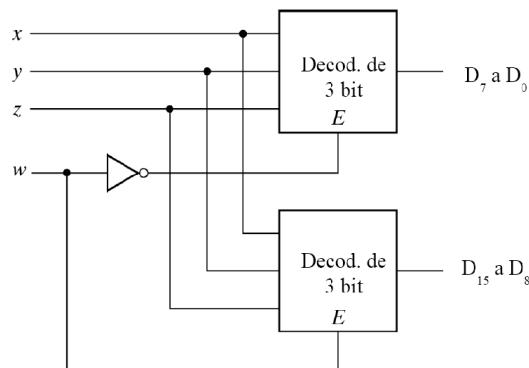


FIGURA 24. Decodificador de 4 bits

3.7.3. Aplicaciones de los Decodificadores.

3.7.3.1. Implementación de funciones: sumador completo.

Lo primero, consideraremos que el Decodificador presenta las salidas activas a 1, a diferencia de lo expresado en la Tabla 11 (decodificador de 2 bits) y su posterior desarrollo. Cada salida del decodificador es, por lo tanto, un mintérmino, por lo que podemos usarla para representar cualquier expresión canónica, sumando las salidas correspondientes a mintérminos de la función con una puerta OR.

Observando la Tabla 8, si expresamos las funciones s y c_i en términos canónicos, tendrémos las siguientes expresiones:

$$\begin{aligned}s &= \bar{x} \cdot \bar{y} \cdot z + \bar{x} \cdot y \cdot \bar{z} + x \cdot \bar{y} \cdot \bar{z} + x \cdot y \cdot z \\c_i &= \bar{x} \cdot y \cdot z + x \cdot \bar{y} \cdot z + x \cdot y \cdot \bar{z} + x \cdot y \cdot z\end{aligned}$$

Expresado como:

$$\begin{aligned}s &= \sum(m_1, m_2, m_4, m_7) \\c_i &= \sum(m_3, m_5, m_6, m_7)\end{aligned}$$

Escogemos las salidas 1, 2, 4 y 7 para representar la función suma, s , y las salidas 3, 5, 6 y 7 para representar la función de acarreo, c_i . Finalmente añadimos una puerta OR para agrupar cada 4 mintérminos. La Figura 25 muestra la implementación del sumador completo.

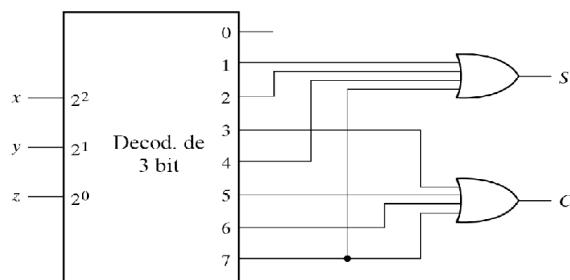


FIGURA 25. Aplicación de un Decodificador en implementación de funciones

Nota: Como ejercicio para el lector, queda el representar el sumador completo tal y como se ha desarrollado con salidas activas como Maxtérminos.

3.8. APLICACIONES: CONVERTIDOR DE CÓDIGO

Una aplicación típica de decodificador+codificador es el convertidor de código.

1. Un dato numérico es introducido al decodificador para activar la salida correspondiente.
2. La señal activada es la entrada para un codificador en otro código, obteniendo a la salida el dato codificado en el nuevo código.

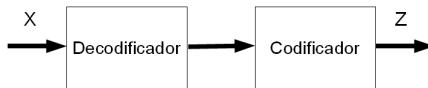


FIGURA 26. Convertidor de código

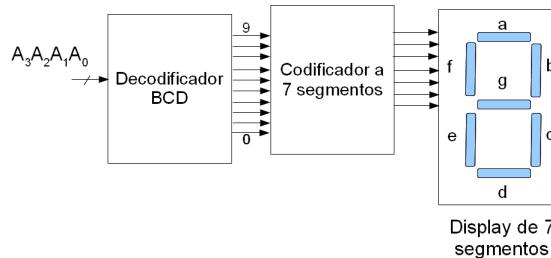


FIGURA 27. Visualizador de 7 segmentos

La Tabla 12 muestra el proceso de decodificación y el de codificación.

CUADRO 12. Tabla de verdad de un decodificador unido a un codificador

A_3	A_2	A_1	A_0	S	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0	0	0	0
0	0	1	0	2	1	1	0	1	1	0	1
0	0	1	1	3	1	1	1	1	0	0	1
0	1	0	0	4	0	1	1	0	0	1	1
0	1	0	1	5	1	0	1	1	0	1	1
0	1	1	0	6	1	0	1	1	1	1	1
0	1	1	1	7	1	1	1	0	0	0	0
1	0	0	0	8	1	1	1	1	1	1	1
1	0	0	1	9	1	1	1	1	0	1	1

EJERCICIOS

- Ejercicio 1: Utiliza multiplexores 74151 y cualquier otra lógica necesaria para multiplexar 16 líneas de datos en una única línea de salida.
- Ejercicio 2: Utilizando un MUX de 4 a 1, implementa $f(a, b, c) = ab + ac$.
- Ejercicio 3: Utilizando un MUX de 8 a 1, implementa la función:
 $f = \Sigma m(1, 2, 5, 6, 7, 8, 10, 12, 13, 15)$.
- Ejercicio 4: Utilizando el decodificador 74154, implementa un circuito para decodificar un número de 5 bits. Determina la salida que se activa al introducir el código binario de entrada 10101.
- Ejercicio 5: Diseña un decodificador de 2 a 4, con entrada de activación E , que tenga salidas y entradas activadas a nivel alto. Utilizando el decodificador diseñado, decodifica de 4 a 16.
- Ejercicio 6: Utiliza el 74151 para implementar la función:
 $f = \Sigma m(0, 2, 4, 6)$.
- Ejercicio 7: Utiliza el 74151 para implementar la función:
 $f = \Sigma m(0, 1, 2, 3, 4, 9, 13, 14, 15)$.
- Ejercicio 8: Diseña un circuito que sume uno al número que tengamos.
- Ejercicio 9: Diseña un restador ($A - B$) en complemento a 2. El resultado final debe representar el valor absoluto y su signo, este se reflejará sobre un led. Las variables A y B son de 4 bits y positivas.
- Ejercicio 10: Diseña un circuito que multiplique un número de 4 bits por dos, tres y cinco.
- Ejercicio 11: Realiza un sumador binario natural a BCD.

Capítulo 4

BLOQUES SECUENCIALES

El tiempo revela todas las cosas: es un charlatán y habla hasta cuando no se le pregunta. (Eurípides de Salamina)

RESUMEN. En el Capítulo 3, hemos visto que un sistema combinacional presenta en sus salidas un resultado función de las entradas binarias, por lo tanto, no permite secuenciar diferentes acciones y que las salidas del sistema dependan de situaciones anteriores. Estas situaciones anteriores se van a conocer como “estados” del sistema digital y van a marcar la evolución a estados futuros. Esto requiere introducir el concepto de memorización de estados. El diseño de sistemas secuenciales se basa en dispositivos que memoricen valores y su aplicación para construir estructuras complejas. En este capítulo se introducirán los dispositivos de memorización y las bases para realizar el análisis y la síntesis de los sistemas secuenciales.

4.1. INTRODUCCIÓN

Veamos una reflexión previa a la presentación de los sistemas secuenciales. Teniendo en cuenta que en la Sección 3 se han tratado los circuitos combinacionales, podemos resaltar que:

- Los circuitos combinacionales no pueden mantener valores en el tiempo, ni producir listas ordenadas de valores, es decir, no se puede realizar, p. ej., un “contador” ($1 \rightarrow 2 \rightarrow 3 \rightarrow \dots \rightarrow n$) porque no se puede “guardar” el valor previo de la cuenta.

Este tipo de funciones las van a realizar los denominados circuitos secuenciales, que constituyen la base de las memorias y de los microprocesadores. Ejemplos de circuitos secuenciales que nos rodean en nuestra vida cotidiana pueden ser: ascensores, semáforos, etc.

Teniendo en cuenta lo expuesto, la característica que va a diferenciar un circuito secuencial de un circuito combinacional es la siguiente:

- En los circuitos secuenciales, en un instante de tiempo determinado, la salida depende de los valores de las entradas externas y de la información almacenada, presente en ese instante.

Los diferentes dispositivos que constituyen las memorias pueden estar regidos, o no, por una señal periódica pulsante (reloj), por lo tanto se dividirán en:

- Asíncronos: los cambios en las salidas vienen regidos por el cambio de las variables de entrada.
- Síncronos: Los cambios en las salidas vienen regidos por las transiciones de una señal digital periódica denominada reloj.

Los dispositivos de memorización se conocen genéricamente como “básculas” al cambiar su valor de $0 \rightarrow 1$ o de $1 \rightarrow 0$, permaneciendo en el nuevo valor hasta que se presenten las nuevas condiciones que determinen que “bascule” la salida.

4.2. CIRCUITOS SECUENCIALES

4.2.1. Concepto de Estado.

Un circuito digital, de tipo secuencial, genera una función que es reenviada como nueva entrada al circuito, esto se conoce como “realimentación”.

- La señal realimentada, antes de ser memorizada, presenta el “estado futuro” del circuito. Las variables que conforman el estado futuro se denominan “variables de excitación”.
- El “estado futuro” del circuito constituye la entrada de los elementos que van a memorizar el estado.
- La señal realimentada, una vez memorizada, se denomina “señal de estado”, y a cada combinación actual de valores se le denomina “estado del circuito”. Las variables que conforman el estado se denominan “variables de estado”.

La estructura básica de un circuito secuencial se observa en la Figura 1:

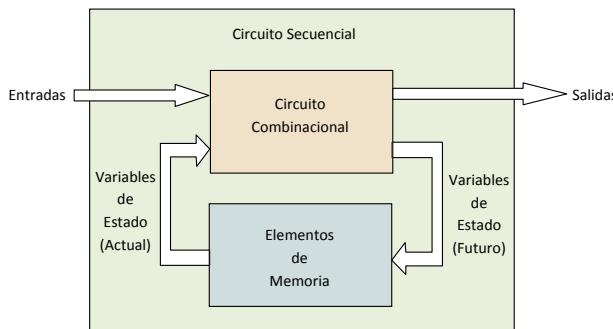


FIGURA 1. Circuito secuencial básico

- Cada vez que el estado del circuito cambie, la salida tendrá nuevos valores aunque el valor de la entrada sea el mismo. Es decir, un circuito secuencial puede evolucionar sin que sus entradas varíen (característica de los dispositivos síncronos).
- La variable de estado debe permanecer aunque la entrada cambie, pues de lo contrario se perderá el estado del circuito¹. La permanencia se lleva a cabo por los elementos de memoria.

Estas características las podemos ver en la Figura 2 que muestra la evolución de un circuito contador:

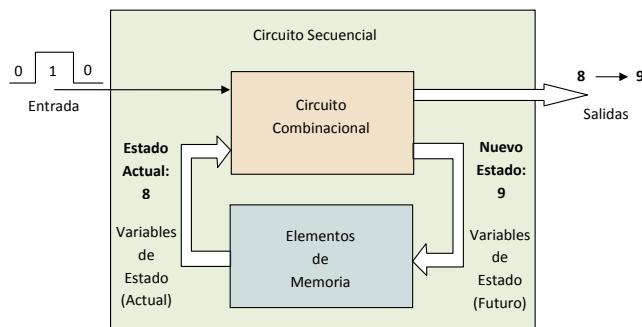


FIGURA 2. Circuito secuencial básico con memoria

Se observa que cuando se produce un pulso en la entrada, si la salida del contador es “8”, ésta evolucionará a “9”.

Debido a este comportamiento, este tipo de circuitos se denomina Máquina de Estados Finitos² (MEF), o también Autómata de Estados Finitos.

¹Esta afirmación será matizada con posterioridad al tratar los dispositivos asíncronos y síncronos por presentar características diferenciadas (estos últimos regidos los cambios por reloj).

²En terminología anglosajona “Finite-State Machine” (FSM)

4.2.2. Biestables Asíncronos (latch).

4.2.2.1. Tipo S-R.

Un latch, o cerrojo, S-R (Set-Reset) es un elemento de memoria de un bit que presenta dos estados estables (1 o 0), siendo capaz de mantener el valor de este bit indefinidamente. La funcionalidad es la siguiente:

- Un 1 en S cambia el valor de la salida a 1, y un 1 en R cambia el valor de la salida a 0.
- Un 0 en S y un 0 en R, dejan el valor de la salida inalterado (se mantiene el valor anterior).

La Figura 3 muestra un latch S-R y las características funcionales las podemos ver en la Tabla 1:

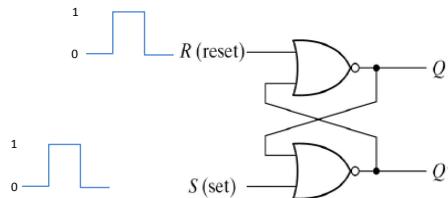


FIGURA 3. Esquema del latch S-R con puertas NOR

CUADRO 1. Tabla de verdad de un latch S-R

S	R	Q	\bar{Q}	
1	0	1	0	
0	0	1	0	Después de que S = 1
0	1	0	1	
0	0	0	1	Después de que R = 1
1	1	0	0	Estado prohibido

En la Tabla 1 se denomina estado prohibido a aquel en que $Q = 0$ y $\bar{Q} = 0$ ya que se presenta una contradicción entre dos salidas que deben ser complementarias y son iguales.

Continuando con la interpretación de la Tabla 1, vemos que:

- Cuando las entradas S y R valgan 0, el valor de la salida (estado) se mantendrá hasta que vuelva a cambiar a 1 alguna de las entradas (S o R).

- Si las dos entradas valen 1 y pasan simultáneamente a valer 0, vemos que $Q = 0$ y $\bar{Q} = 0$, este valor de estado no se puede mantener, se trata de un estado inestable.

La explicación es la siguiente: Al pasar las dos entradas simultáneamente a valer 0 ($S = 0, R = 0$), las dos salidas Q y \bar{Q} pasan a valer 1 ($Q = 1, \bar{Q} = 1$), lo que lleva a que, debido a la realimentación de Q y \bar{Q} , las salidas Q y \bar{Q} valgan 0 ($Q = 0, \bar{Q} = 0$) y este proceso se extendería al infinito, de no ser por las características físicas de los componentes. Alguna de las ramas va a presentar un retardo ligeramente diferente, siendo alguna de las salidas (Q o \bar{Q}) quien adelante su nuevo estado y fije el latch definitivamente.

Estas características las podemos ver en la Tabla 2, donde Q representa la salida actual y Q^* la nueva salida:

CUADRO 2. Estados prohibidos e inestables

Q	S	R	Q^*	\bar{Q}^*	
0	0	0	0	1	
0	0	1	0	1	
0	1	0	1	0	
0	1	1	0	0	Estado prohibido
1	0	0	1	0	
1	0	1	0	1	
1	1	0	1	0	
1	1	1	0	0	Estado prohibido

Hemos visto la implementación del latch S-R con puertas NOR, veamos otro tipo de implementación:

- El latch S-R se puede implementar con puertas NAND.
- Este circuito funciona con entradas a nivel bajo: S y R se activan con 0s.
- El estado prohibido (que genera un estado inestable) aparece con dos 0s en las entradas .

Estas características las podemos ver en la Figura 4 (queda para el lector el realizar la tabla de verdad de este latch):

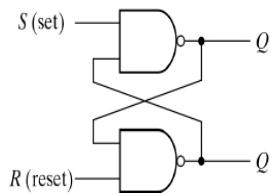


FIGURA 4. Esquema del latch S-R con puertas NAND

La Figura 5 muestra los cronogramas de un latch SR con puertas NOR y NAND:

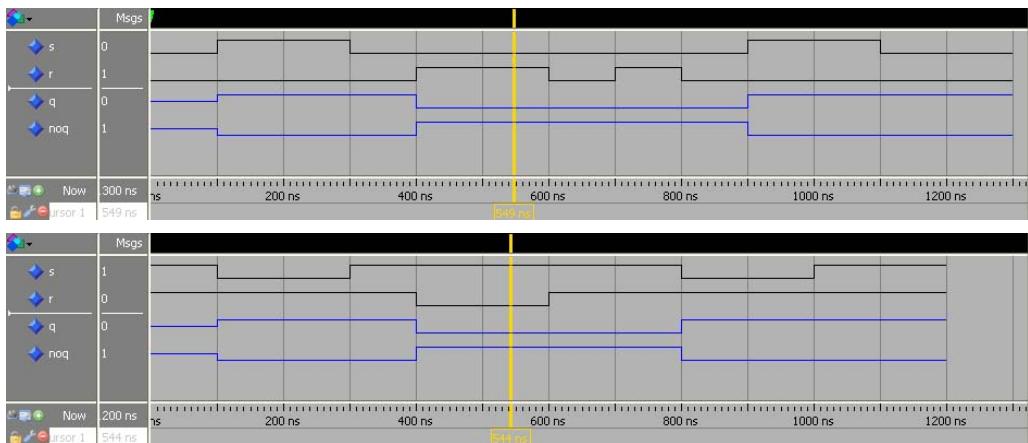


FIGURA 5. Cronogramas de latchs con puertas NOR y NAND

Observando las características del latch S-R se constata que:

- Los cambios de estado se pueden suceder en cascada.
- Estos cambios los podemos evitar limitándolos en el tiempo.

El Anexo 1 muestra el desarrollo de este latch descrito por medio del lenguaje de descripción hardware VHDL.

Introduzcamos ahora una modificación al latch S-R representado en la Figura 4. Si se le añade una entrada de “control”, puede producirse el siguiente efecto:

- Sólo cambiará el estado cuando la entrada de control sea 1.

La Figura 6 muestra un latch SR con puertas NAND y entrada de control:

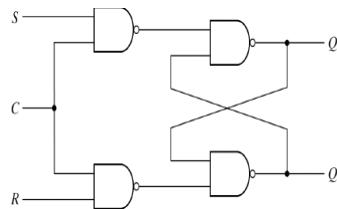


FIGURA 6. Esquema del latch S-R con puertas NAND y entrada de Control

La Tabla 3 muestra el comportamiento del latch SR con puertas NAND y entrada de control. Se observa que la introducción de la puerta de control no resuelve el problema del estado prohibido.

CUADRO 3. Tabla de verdad de un latch S-R con puertas NAND y puer-
ta de control

C	S	R	Q	\bar{Q}	
0	X	X	Q	\bar{Q}	
1	0	0	Q	\bar{Q}	
1	0	1	0	1	
1	1	0	1	1	
1	1	1	1	1	Estado prohibido

La Figura 7 muestra los cronogramas de un latch SR con entrada de control:

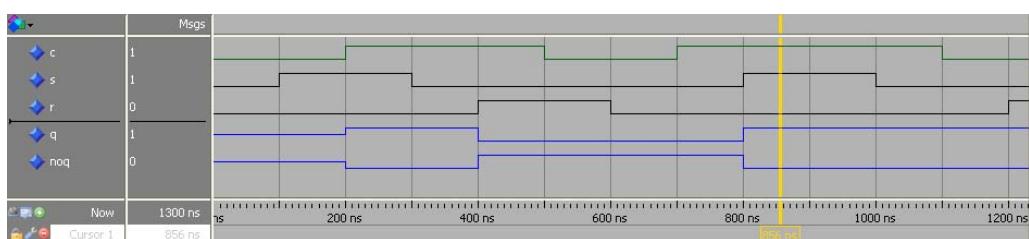


FIGURA 7. Cronograma de un latch SR con entrada de control

El Anexo 2 muestra el desarrollo de este latch descrito por medio del lenguaje de descripción hardware VHDL.

4.2.2.2. Tipo D.

Un latch de tipo D presenta el siguiente comportamiento:

- El valor de la entrada se repite en la salida, salvo si la entrada de control está a 0 → Memoria del último estado.

El latch de tipo D presenta la siguiente ventaja con respecto al latch S-R:

- Un latch D no presenta el estado inestable del latch S-R.

La Figura 8 muestra un biestable asíncrono D con puertas NAND y entrada de control. La Tabla 4 muestra el comportamiento de un latch D:

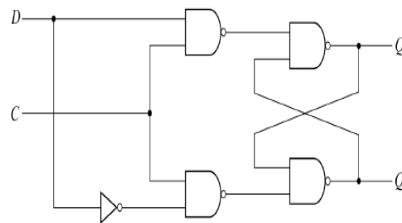


FIGURA 8. Esquema del latch D con puertas NAND y entrada de Control

CUADRO 4. Tabla de verdad de un latch D

C	D	Q^*	\bar{Q}^*
0	X	Q	\bar{Q}
1	0	0	1
1	1	1	0

La Figura 9 muestra los cronogramas de un latch D con entrada de control:

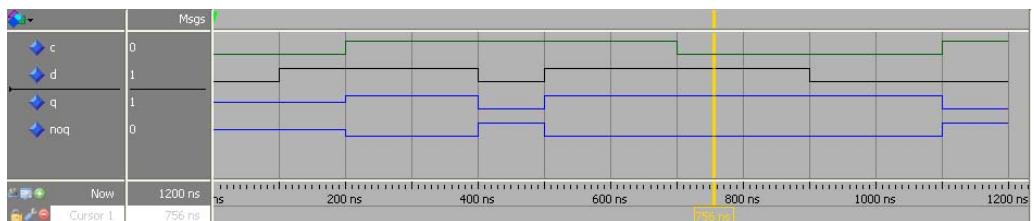


FIGURA 9. Cronograma de un latch D con entrada de control

El Anexo B muestra el desarrollo de este latch descrito por medio del lenguaje de descripción hardware VHDL.

4.2.3. Modos de Activación.

1. Activación por nivel:

- La señal de control (activación por nivel) requiere definir un intervalo de tiempo en el que se puede cambiar el estado.
- Durante el intervalo de activación del elemento de memoria, los cambios de estado se pueden dar en cascada.

La Figura 10 muestra la activación por nivel:



FIGURA 10. Activación por nivel alto de un latch

2. Activación por flanco:

- Se puede reducir este intervalo a sólo el necesario para que una señal cambie de valor → Activación por flanco.
- Si la señal de activación es de frecuencia constante, los cambios de estado estarán sincronizados a esa frecuencia → Reloj.

La Figura 11 muestra la activación por flanco:

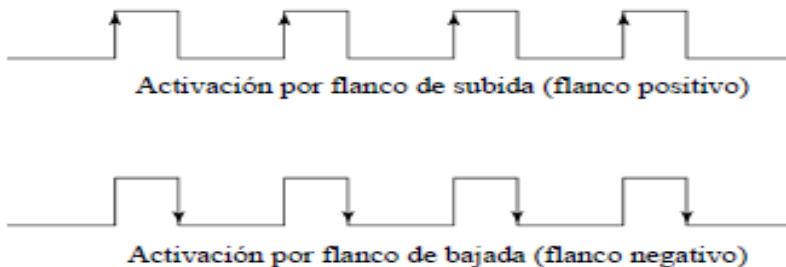


FIGURA 11. Activación por flanco de un latch

4.2.4. Biestables Síncronos (flip-flop).

4.2.4.1. Tipo D.

Un flip-flop D (FF-D) es el elemento de memoria equivalente al cerrojo D con entrada de control, pero siendo esta entrada de control activada por flanco.

- La entrada de control pasa a denominarse reloj (clock o clk).
- Con cada flanco del reloj la entrada aparece en la salida.

La Figura 12 muestra el flip-flop D y la Tabla 5 su comportamiento:

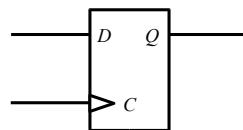


FIGURA 12. Flip-Flop D

CUADRO 5. Tabla de verdad de un flip-flop D

clk	D	Q^*	\bar{Q}^*
X	X	Q	\bar{Q}
\uparrow	0	0	1
\uparrow	1	1	0

Cuando el clk presenta un flanco ascendente (positivo) \uparrow , la entrada D se muestra en la salida Q.

La Figura 13 muestra el cronograma de un FF-D activado por flanco:

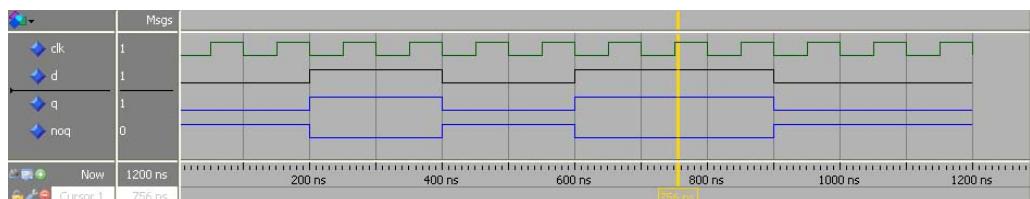


FIGURA 13. Cronograma de un FF-D activado por flanco de reloj

El Anexo B muestra el desarrollo de este FF descrito por medio del lenguaje de descripción hardware VHDL.

4.2.4.2. Tipo JK³.

Un flip-flop JK (FF-JK) puede verse como una modificación del flip-flop D⁴ pero provisto de dos entradas.

- Su funcionalidad es semejante al latch SR, pero si se activan las dos entradas, la salida cambia ($1 \rightarrow 0$ o $0 \rightarrow 1$). Esta situación se conoce como “Toggle”. El FF-JK, debido a esta última funcionalidad, resuelve el problema de las entradas iguales del latch SR.

La Figura 14 muestra el FF-JK y la Tabla 6 su comportamiento:

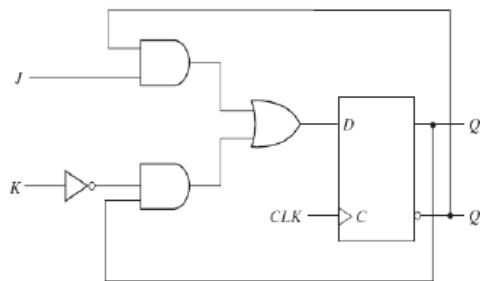


FIGURA 14. Flip-Flop JK

La ecuación que rige el comportamiento del FF-D, de la estructura del FF-JK es: $D = J \cdot \bar{Q} + \bar{K} \cdot Q$.

CUADRO 6. Tabla de verdad de un flip-flop JK

clk	J	K	Q^*	\bar{Q}^*
X	X	X	Q	\bar{Q}
↑	0	0	Q	\bar{Q}
↑	0	1	0	1
↑	1	0	1	0
↑	1	1	\bar{Q}	Q

“Toggle”

La Figura 15 muestra el cronograma de un FF-JK activado por flanco:

³El nombre proviene de su inventor Jack S. Kilby

⁴Esto es una visión pedagógica pero no constituye el dispositivo electrónico de origen

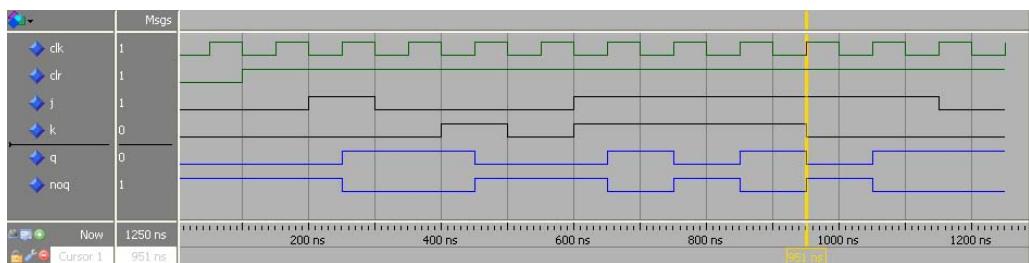


FIGURA 15. Cronograma de un FF-JK activado por flanco y señal de Clear (CLR)

El Anexo B muestra el desarrollo de este FF descrito por medio del lenguaje de descripción hardware VHDL.

4.2.4.3. *Tipo T.*

El nombre T, del biestable síncrono, viene dado por su comportamiento “Toggle”⁵.

- Se modifica el FF-JK para que disponga de solo una entrada.

Su funcionalidad es la siguiente:

- Mientras la entrada T sea 1, la salida cambia con cada flanco del reloj.
- Si la entrada T es 0, la salida no cambia con los flancos del reloj.

Se aplica en contadores, ya que el cambio de una variable es la cuenta de una cifra binaria.

La Figura 16 muestra el FF-T y la Tabla 7 su comportamiento:

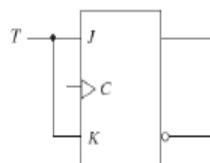


FIGURA 16. Flip-Flop T

⁵En su inicio era conocido como “Trigger”, ya que era la entrada de “disparo” conectada a las bases de dos transistores NPN

CUADRO 7. Tabla de verdad de un flip-flop T

clk	T	Q^*	\bar{Q}^*
X	X	Q	\bar{Q}
\uparrow	0	Q	\bar{Q}
\uparrow	1	\bar{Q}	Q

“Toggle”

La Figura 17 muestra el cronograma de un FF-T activado por flanco:

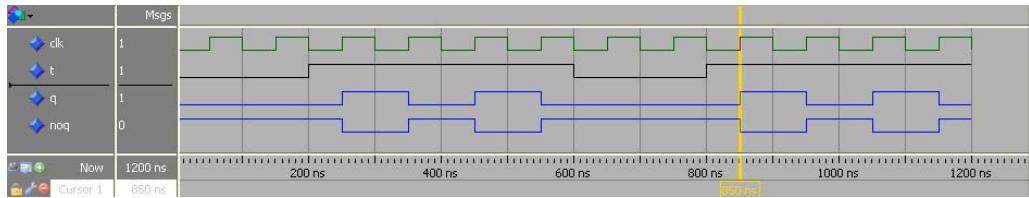


FIGURA 17. Cronograma de un FF-T

El Anexo B muestra el desarrollo de este FF descrito por medio del lenguaje de descripción hardware VHDL.

4.2.4.4. Entradas Asíncronas.

Una entrada es asíncrona cuando al producirse su activación, el comportamiento del FF cambia (según la funcionalidad asociada a esta entrada) sin esperar a la llegada del reloj.

- Los flip-flop pueden incorporar el comportamiento del cerrojo (latch)
 - El estado cambia cuando cambia la entrada.
- La entrada asíncrona R (RESET) lleva al estado a valer 0, y la entrada asíncrona S (SET) o P (PRESET) lleva al estado a valer 1.
- Las entradas asíncronas tienen preferencia sobre las entradas de los flip-flop.

La Figura 18 muestra la entrada asíncrona de Reset y la Tabla 8 su comportamiento:

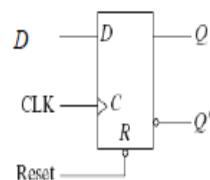


FIGURA 18. Flip-Flop D con entrada asíncrona de Reset

CUADRO 8. Tabla de verdad de un flip-flop D con entrada asíncrona de Reset

R	clk	D	Q^*	\bar{Q}^*
0	X	X	0	1
1	X	X	Q	\bar{Q}
1	↑	0	0	1
1	↑	1	1	0

La Figura 19 muestra el cronograma de un FF-D activado por flanco y con entrada asíncrona de Reset:

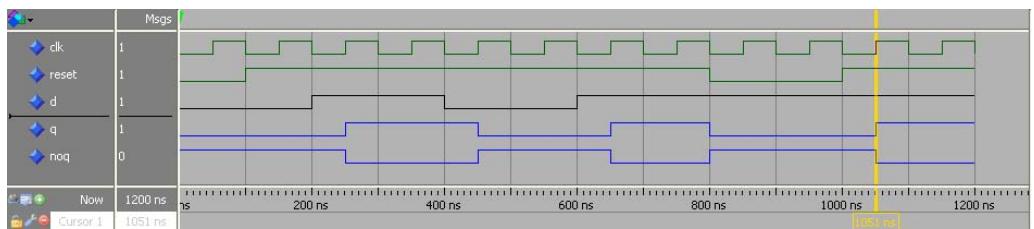


FIGURA 19. Cronograma de un FF-D activado por flanco de reloj y entrada asíncrona de Reset

4.3. REGISTROS Y CONTADORES

4.3.1. Registros.

Entendemos los registros como una agrupación ordenada de elementos de memoria unitarios (bits).

- Las aplicaciones más sencillas de los elementos de memoria son los registros y los contadores.
- Los registros permiten mantener el valor de un dato binario (registro de almacenamiento) y operar sobre él (registro de desplazamiento).
- Los contadores producen un dato numérico de una secuencia predefinida, a cada flanco de reloj.

4.3.1.1. Registro de Almacenamiento.

Partiendo del registro presentado en la Figura 20, realizado con FFs-D, el comportamiento será el siguiente:

- Mantiene el dato binario de I desde que aparece el flanco de reloj hasta el nuevo flanco.

- Durante ese tiempo, y aunque cambie I, en A se mantiene el dato introducido.
- Al activar Reset, el valor de A vuelve a 0.

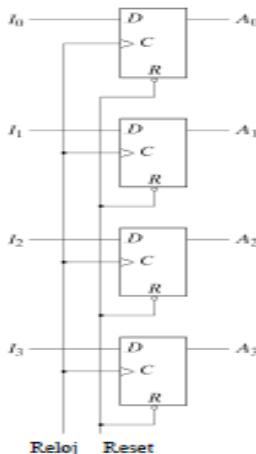


FIGURA 20. Registro de almacenamiento

El Anexo B muestra el desarrollo de este registro descrito por medio del lenguaje de descripción hardware VHDL.

4.3.1.2. Registro de Desplazamiento.

El registro de desplazamiento modifica el dato introducido, escribiendo cada bit del dato en la siguiente posición, es decir, la salida de un FF es la entrada de otro FF. teniendo en cuenta la posición de los FFs, los registros de desplazamiento pueden agruparse en dos categorías:

- Registros de desplazamiento de Izquierda a Derecha: La salida de un FF se conecta con la entrada del FF inmediato de su derecha y así sucesivamente.
- Registros de desplazamiento de Derecha a Izquierda: La salida de un FF se conecta con la entrada del FF inmediato de su izquierda y así sucesivamente.

El dato puede ser introducido en formato serie (un bit cada período de reloj) o en formato paralelo (todos los bits se introducen a la vez). Ver Figura 22 .

La Figura 21 muestra un registro de desplazamiento realizado con FFs-D con desplazamiento de izquierda a derecha:

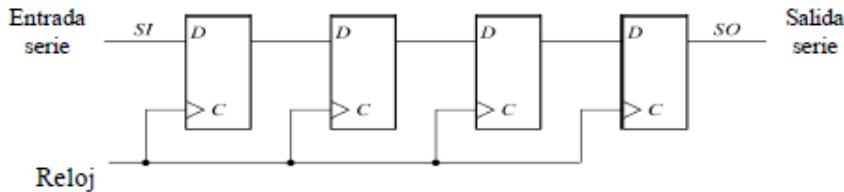


FIGURA 21. Registro de desplazamiento

- Existen diseños con desplazamiento a la izquierda o incluso en los que se puede elegir el desplazamiento a la izquierda o a la derecha.
- En el diseño presentado en la Figura 22, se puede elegir entrada serie o paralelo.

La Figura 22 muestra un registro de desplazamiento de 4 bits con entrada serie y paralelo:

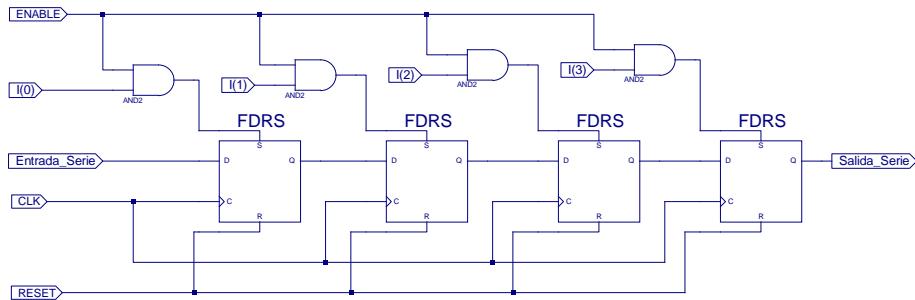


FIGURA 22. Registro de desplazamiento - entrada serie y paralelo

El Anexo B muestra el desarrollo de este registro descrito por medio del lenguaje de descripción hardware VHDL.

La Figura 23 muestra los cronogramas de un registro de desplazamiento de 4 bits, siendo:

- k: selección de izda. (1) o dcha. (0).
- ei: entrada izda.
- ed: entrada dcha.
- s: salida, pudiendo observarse el desplazamiento de los bits de cada FF.

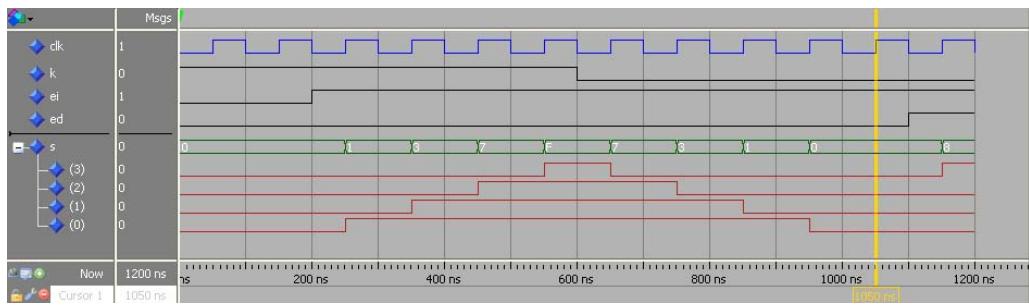


FIGURA 23. Cronograma de un registro de desplazamiento derecha/izquierda

La Figura 24 muestra los cronogramas de un registro de desplazamiento de 4 bits con entrada paralelo. En este caso, las señales son:

- k: selección de izda. (01), dcha. (10), almacenamiento (00) y carga paralelo (11).
- ep: entrada paralelo.
- ei: entrada izda.
- ed: entrada dcha.
- s: salida, pudiendo observarse el desplazamiento de los bits de cada FF.

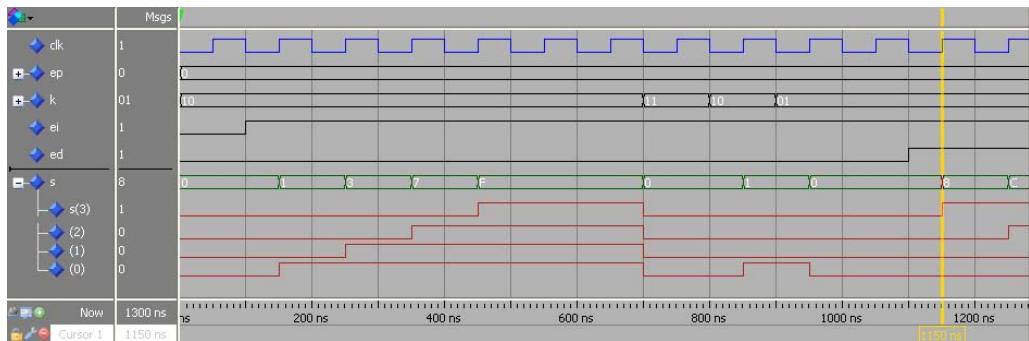


FIGURA 24. Cronograma de un registro de desplazamiento derecho/izquierdo y entrada paralelo

4.3.2. Contadores.

Los contadores producen un dato binario que va cambiando a cada flanco de reloj, de una secuencia prefijada.

- La secuencia suele ser la binaria ascendente o descendente, pero puede ser también BCD.

- Los contadores se llaman síncronos si todos los bits del dato producido cambian al mismo tiempo.
- Si no ocurre así, se llaman asíncronos.

4.3.2.1. Contador Síncrono.

- Este contador produce, en A, la secuencia binaria ascendente, a cada pulso de CLK.
- Cada bit cambia cuando los anteriores son 1.
- La secuencia se detiene si Conteo vuelve a 0.

La Figura 25 muestra un contador síncrono realizado con FFs-JK trabajando como FFs-T:

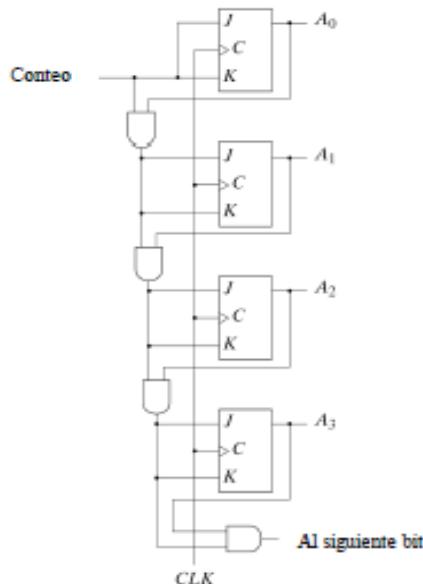


FIGURA 25. Contador síncrono

La Figura 26 muestra el cronograma de un contador síncrono, siendo:

- k: cuenta ascendente (01), cuenta descendente (10), almacenamiento (00) y carga paralelo (11).
- ep: entrada paralelo.
- s: salida del contador.

:

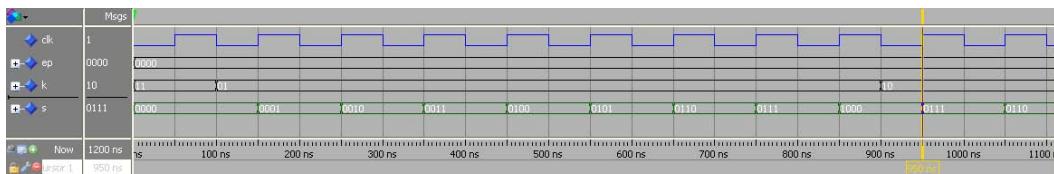


FIGURA 26. Cronograma de un contador síncrono

El Anexo B muestra el desarrollo de un contador síncrono descrito por medio del lenguaje de descripción hardware VHDL.

4.3.2.2. Contador Asíncrono.

- Cada flip-flop tiene una señal de reloj diferente, por lo que cambia en diferente instante.
- La secuencia es la binaria ascendente a cada flanco de *CLK*, pero hay pequeños intervalos erróneos.
- Al activar Reset, A vuelve a 0 sin esperar al flanco de *CLK*.

La Figura 27 muestra un contador asíncrono realizado con FFs-JK trabajando como FFs-T:

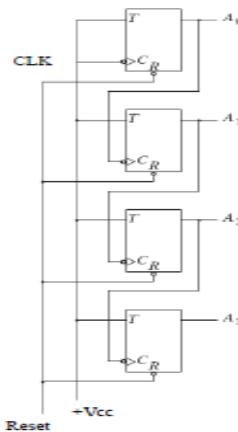


FIGURA 27. Contador asíncrono

La Figura 28 muestra el cronograma de un contador asíncrono, siendo:

- reset: entrada asíncrona de RESET.
- s: salida del contador.

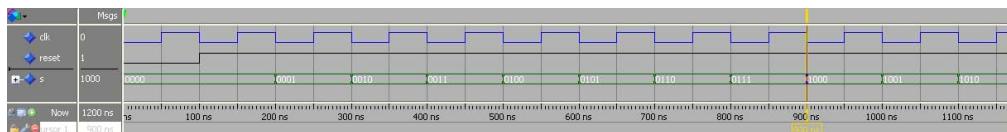


FIGURA 28. Cronograma de un contador asíncrono

4.4. SISTEMAS SECUENCIALES SÍNCRONOS

Los circuitos secuenciales, cuyos elementos de memoria son flip-flops con la misma señal de reloj, constituyen los sistemas secuenciales síncronos.

- En estos sistemas, de forma genérica, la salida es función del estado actual y de la entrada.
- Junto con la salida, se define el valor del estado siguiente, este estado es almacenado en los flip-flops.
- El estado sólo puede cambiar cuando lo hace la señal de reloj.

La Figura 29 muestra un esquema de bloques de un sistema secuencial síncrono:

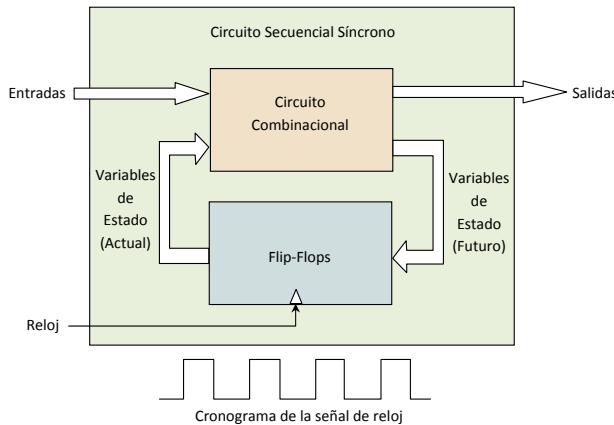


FIGURA 29. Esquema de bloques de un sistema secuencial síncrono

4.4.1. Análisis y Síntesis de Sistemas Secuenciales Síncronos.

Aclaremos, primeramente el concepto de los términos “Análisis” y “Síntesis”.

- Análisis: Estudio de un sistema para describir su funcionamiento.

- Síntesis: Diseño de un circuito a partir de la descripción de su funcionamiento.

4.4.2. Análisis de Sistemas Secuenciales.

- Para analizar un sistema secuencial, hay que conocer el valor del nuevo estado para cada combinación de valores de entrada y estado actual
→ Ecuaciones de nuevo estado.
- De este modo, y con las ecuaciones de respuesta de los flip-flops, conoceremos los cambios de estado (transiciones) del sistema para cada posible valor de entrada.

La Figura 30 muestra un sistema secuencial sobre el que se realizará el análisis y la Tabla 9 muestra la evolución de los estados según el estado actual y el valor de las variables de entrada.

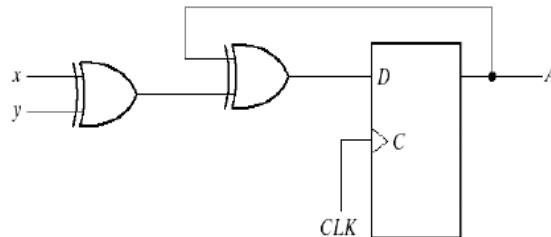


FIGURA 30. Ejemplo de sistema secuencial

Las ecuaciones que rigen este sistema son las que se muestran a continuación:

$$\begin{aligned} D &= X \oplus Y \oplus A \\ Q^* &= D \\ A &= Q \end{aligned}$$

Notese que, dada la sencillez del circuito propuesto, la salida A coincide con el estado Q .

CUADRO 9. Tabla de estados

Estado actual	Entradas		Estado siguiente
A	x	y	A
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

4.4.2.1. Diagramas de Estado.

Un diagrama de estado es una representación gráfica de la evolución de un sistema secuencial. Se apoya en dos elementos gráficos: burbuja y línea direccional. La burbuja determina el estado actual y la línea direccional representa la transición de un estado (burbuja) a otro estado (burbuja) según ciertas condiciones.

- La evolución de un estado a otro viene determinada por la aparición de un flanco de reloj. O bien se evoluciona con flanco positivo o bien con flanco negativo.
- Con la aparición del flanco de reloj y teniendo en cuenta el estado actual, son evaluadas las entradas del sistema, determinándose hacia qué estado evoluciona. Esta evolución viene reflejada por las líneas direccionales.
- Los diagramas de estado son representaciones gráficas de todas las posibles transiciones del sistema secuencial, con los correspondientes valores de salida → Ecuaciones de salida.

Podemos concluir que:

- El diagrama de estado permite predecir el comportamiento del sistema ante cualquier sucesión de variables de entrada (conocido el estado inicial).

Como ejemplo de lo anteriormente expuesto, la Figura 31 muestra el diagrama de estados correspondiente al esquema de bloques de la Figura 30 y la Tabla 10 refleja, para unos valores de las entradas (X, Y), el estado actual y cual será

el estado siguiente (marcado por el reloj), así como la salida Z asociada a la transición⁶.

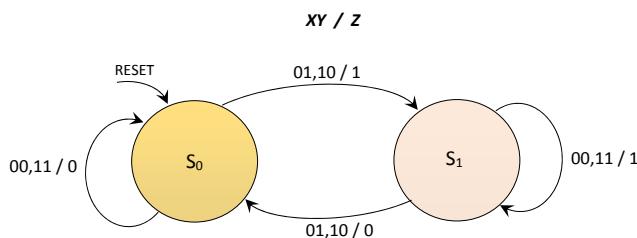


FIGURA 31. Diagrama de estados de un sistema secuencial síncrono - Entradas: XY / Salida: Z

CUADRO 10. Evolución de estados

X	0	0	1	1	0	0	0
Y	0	1	1	0	0	1	0
Salida Actual A	0	0	1	1	0	0	1
Estado Actual	0	0	1	1	0	0	1
Estado Siguiente	S_0	S_1	S_1	S_0	S_0	S_1	S_1

No importa cuándo cambia la entrada, los cambios de estado se realizan en el instante del flanco de reloj.

4.4.3. Síntesis de Sistemas Secuenciales.

Para diseñar un sistema secuencial, podemos considerar dos modelos para las salidas del sistema: Mealy y Moore.

1. Modelo Mealy

Las funciones de salida dependen tanto del estado actual como de la variable de entrada. La Figura 32 muestra el esquema de bloques de un sistema secuencial modelo Mealy.

⁶Como veremos en 4.4.3, podemos asociar las salidas a la transición o al estado

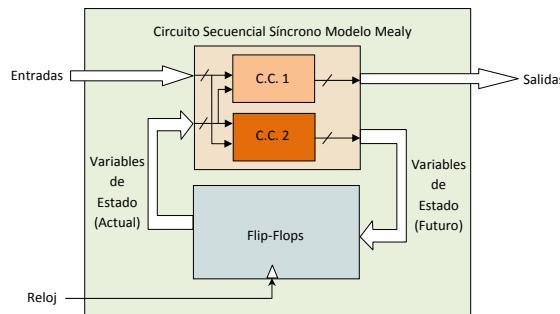


FIGURA 32. Sistema secuencial síncrono modelo Mealy

2. Modelo Moore

Podemos simplificar el diseño, tomando las funciones de salida como dependientes sólo del estado actual. Las entradas junto con el estado actual, afectan al cambio del estado. De este modo, cuando cambia el estado, cambia la salida. La Figura 33 muestra el esquema de bloques de un sistema secuencial modelo Moore.

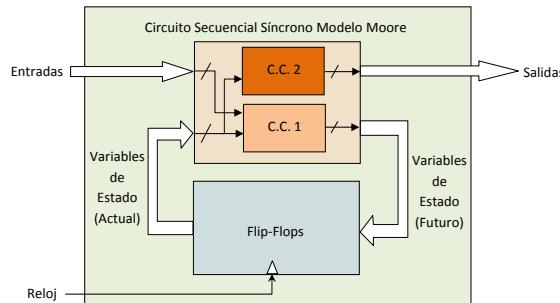


FIGURA 33. Sistema secuencial síncrono modelo Moore

La síntesis, de forma absolutamente genérica, requiere de las siguientes consideraciones previas:

- Partimos de una descripción lingüística de un problema concreto.
- A partir de la descripción del comportamiento del sistema, elaboraremos un diagrama de estados que refleje este comportamiento.

En el diagrama de estados debemos incluir todas las posibles transiciones y considerar todos los estados necesarios para crear el comportamiento descrito.

- La representación del diagrama de estados será llevada a una tabla de estados.
- En esta tabla se buscarán los estados redundantes (que presentan la misma funcionalidad) eliminándose estos⁷.
- Como partimos de denominaciones abstractas de los estados, será necesario concretar, primero, el número de FFs necesarios y, segundo, los valores binarios que van a adoptar los FFs en representación de todos los estados.
 - Los valores concretos serán llevados a la tabla de estado en sustitución de los estados abstractos.
 - Escogeremos el FF con el que vamos a realizar la síntesis (D-T-JK).
 - Teniendo en cuenta la funcionalidad del FF escogido, realizaremos la tabla de excitación de los FFs.
 - De la tabla de excitación extraeremos las tablas de cada entrada de los FFs que serán tratadas como mapas de Karnaugh para su simplificación.

Es aconsejable dibujar un cronograma acorde con la descripción, suele ayudar a entender el comportamiento que preveamos para nuestro circuito.

4.4.3.1. Síntesis según el modelo Moore.

Atendiendo a las consideraciones mencionadas previamente, pasemos a desarrollarlas.

1. Descripción del comportamiento:

Diseñar un circuito contador de dos bit que cuente en sentido ascendente cuando la señal X sea 0, y que cuente en sentido descendente cuando la señal X sea 1. La Figura 34 muestra el cronograma del comportamiento del circuito a diseñar.

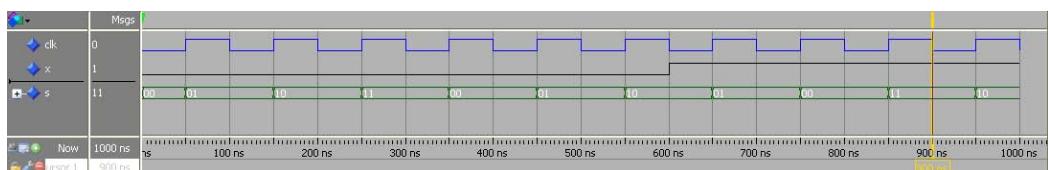


FIGURA 34. Cronograma de un circuito secuencial síncrono contador de 2 bits

⁷Este modo de trabajo no se contempla en esta guía por considerarlo relacionado con otro tipo de formación.

2. Diagrama de estados:

La Figura 35 muestra el “diagrama de estados” del circuito a diseñar:

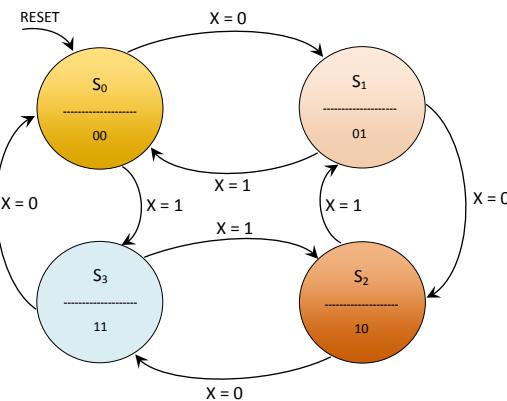


FIGURA 35. Diagrama de estados de un circuito secuencial síncrono contador de 2 bits modelo Moore

El Anexo B muestra la resolución del circuito planteado bajo lenguaje de descripción hardware VHDL.

3. Tabla de estados:

Una vez que hayamos reflejado el comportamiento del sistema deseado sobre el diagrama de estados, procederemos de la siguiente forma:

- La información reflejada por el diagrama de estados la escribiremos en forma de “tabla de estados”, con una fila por cada estado y una columna por cada valor de entrada posible, en cada columna se reflejará el estado siguiente correspondiente a esa entrada; escribiremos los valores de salida uno por cada fila (estado).

La Tabla 11 muestra la confección de la tabla de estados:

CUADRO 11. Tabla de estados del modelo Moore

Estado actual	$x = 0$	$x = 1$	Z_1	Z_0
S_0	S_1	S_3	0	0
S_1	S_2	S_0	0	1
S_2	S_3	S_1	1	0
S_3	S_0	S_2	1	1

Al ser un modelo Moore, las salidas están asociadas al estado actual, es decir, al estado S_0 le corresponde una salida 00, al estado S_1 le corresponde una salida 01 y así sucesivamente.

4. Asignación de estados:

A continuación, se realiza la “asignación de estados” que consiste en dar valores concretos a los estados abstractos.

- Los estados se codifican en variables binarias (asignación de estados), que serán nuestras variables de estado.
- El número de variables binarias vendrá dado por la potencia de 2 que más se ajuste al número de estados requeridos.
- El número de FFs requeridos se corresponde con el número de variables binarias.
- El bit de menor peso se corresponde con el FF-0 y la salida será Q_0 , el siguiente bit se corresponde con el FF-1 y la salida será Q_1 y así sucesivamente.

La Tabla 12 refleja esta asignación. Al tratarse de 4 estados se requieren 2 variables binarias. En este caso vemos que la asignación de estados coincide con las salidas del sistema, esta situación se dá normalmente en diseño de contadores que cuentan en binario natural. La asignación podría haber sido otra muy diferente. El comportamiento final del sistema será el mismo, solamente variará la lógica combinacional que conforma el bloque combinacional.

Los criterios que rigen asignaciones diferentes al binario natural, se escapan del objetivo de esta guía.

CUADRO 12. Tabla de estados - asignación de estados

Estado actual	Q_1	Q_0
S_0	0	0
S_1	0	1
S_2	1	0
S_3	1	1

La asignación realizada, se lleva a la tabla de estados general. La Tabla 13 refleja la asignación final:

CUADRO 13. Tabla de estados - asignación de estados final

Q_1	Q_0	$x = 0$		$x = 1$		Z_1	Z_0
		Q_1^*	Q_0^*	Q_1^*	Q_0^*		
0	0	0	1	1	1	0	0
0	1	1	0	0	0	0	1
1	0	1	1	0	1	1	0
1	1	0	0	1	0	1	1

5. Síntesis con el FF escogido (D - JK - T):

En este punto debemos escoger el FF con el que se completará la implementación que satisfaga los requisitos expuestos en la descripción inicial del sistema. Como observación general, la síntesis con los FF-D y T generarán un circuito combinacional más complejo que con los FF-JK. La existencia de dos entradas (J-K) permiten una mayor flexibilidad en la síntesis, lo que redunda en un menor número de puertas lógicas final. Por contra, el diseño con FF-JK es más complejo que con los FF-D y T. Como apunte final, la síntesis con FF-D es la más sencilla y transparente.

Vamos a realizar la síntesis con FF-JK y en diversos ejercicios, introduciremos la síntesis con FF-D y FF-T. Primeramente, vamos a fijarnos en el comportamiento de un FF-JK. La Tabla 14 muestra, a su izquierda, el comportamiento de un FF-JK y, a su derecha, el valor de la salida actual (Q), el valor futuro de la salida (Q^* - valor que adquirirá en el siguiente flanco de reloj) y los valores de las entradas para que se dé la evolución de Q a Q^* :

CUADRO 14. Tabla de verdad de un flip-flop JK

clk	J	K	Q^*	\bar{Q}^*	Q	Q^*	J	K
X	X	X	Q	\bar{Q}			0	X
↑	0	0	Q	\bar{Q}			0	X
↑	0	1	0	1			1	X
↑	1	0	1	0			X	1
↑	1	1	\bar{Q}	Q			X	0

a) Asignación de valores a $J_i K_i$:

La síntesis requiere el determinar qué valores deben tener las entradas para que, en el siguiente flanco de reloj, las salidas de los FF se correspondan con el “estado siguiente”. Cuando una de las entradas (J o K) adquiere el valor “X” (no importa), significa que puede valer “0” o “1”, no afectando al valor de la salida, que viene determinada por el valor de la otra entrada.

Observando la Tabla 14 derecha, si $Q = 0$ y va a evolucionar a $Q^* = 0$, según muestra la Tabla de la izquierda, si $J = K = 0$ o $J = 0$ y $K = 1$, se puede determinar que no importa qué valor adquiera K con tal que $J = 0$. Este razonamiento es extensivo al resto de casos.

Ya estamos en condiciones de, tomando como referencia la Tabla 13, generar la tabla que contenga los valores de J y de K que determinen la evolución del sistema de un estado actual a un estado futuro. La Tabla 15 muestra las asignaciones de J y de K :

CUADRO 15. Tabla conteniendo los valores J_iK_i necesarios

Q_1	Q_0	$x = 0$				$x = 1$				Z_1	Z_0
		J_1K_1	$Q_1^*Q_0^*$	J_0K_0	$Q_1^*Q_0^*$	J_1K_1	$Q_1^*Q_0^*$	J_0K_0	$Q_1^*Q_0^*$		
(a)	0	0	0X	(0-)	1X	(-1)	1X	(1-)	1X	(-1)	0 0
(b)	0	1	1X	(1-)	X1	(-0)	0X	(0-)	X1	(-0)	0 1
(c)	1	0	X0	(1-)	1X	(-1)	X1	(0-)	1X	(-1)	1 0
(d)	1	1	X1	(0-)	X1	(-0)	X0	(1-)	X1	(-0)	1 1

Con el fin de facilitar la síntesis, hemos introducido columnas adicionales a la derecha de las columnas J_1K_1 y J_0K_0 , que nos indiquen cual es el estado siguiente, es decir, cual es el valor de la salida Q_1^* y Q_0^* , que va a corresponder a los valores J_1K_1 y J_0K_0 , teniendo en cuenta que J_1K_1 afecta a Q_1^* y J_0K_0 afecta a Q_0^* con la llegada del próximo flanco de reloj.

Con el fin de facilitar la comprensión de la tabla, vamos a revisar alguna de las filas para ver el porqué se han tomado esos y no otros valores de J_iK_i .

- Tomemos la fila (a), vemos que $Q_1 = 0 \rightarrow Q_1^* = 0$ (tomemos la flecha \rightarrow como indicación de evolución) para $x = 0$, por lo tanto $J_1K_1 = 0X$, y $Q_0 = 0 \rightarrow Q_0^* = 1$, por lo tanto $J_0K_0 = 1X$. Para $x = 1$ tenemos que $Q_1 = 0 \rightarrow Q_1^* = 1$, por lo tanto $J_1K_1 = 1X$ y $Q_0 = 0 \rightarrow Q_0^* = 1$, por lo tanto $J_0K_0 = 1X$.
- Tomemos la fila (c), vemos que $Q_1 = 1 \rightarrow Q_1^* = 1$ para $x = 0$, por lo tanto $J_1K_1 = X0$, y $Q_0 = 0 \rightarrow Q_0^* = 1$, por lo tanto $J_0K_0 = 1X$. Para $x = 1$ tenemos que $Q_1 = 1 \rightarrow Q_1^* = 0$, por lo tanto $J_1K_1 = X1$ y $Q_0 = 0 \rightarrow Q_0^* = 1$, por lo tanto $J_0K_0 = 1X$.

b) Asignación de valores a Z_1Z_0 :

Teniendo en cuenta que las salidas solo dependen del estado actual (modelo Moore) tal y como se refleja en la Tabla 15, construimos una tabla independiente que refleje este hecho (ver Tabla 16).

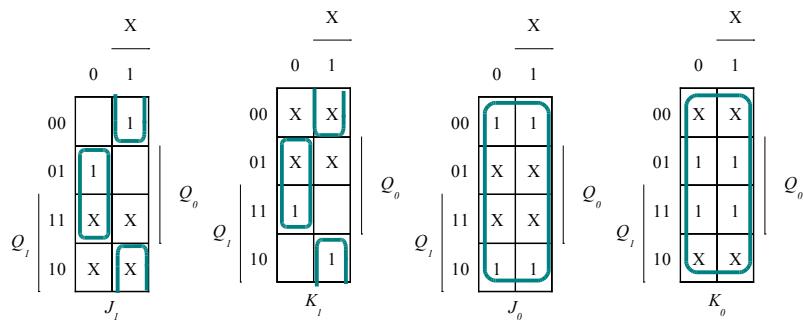
CUADRO 16. Tabla de salidas

Q_1	Q_0	Z_1	Z_0
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

c) Extracción de las ecuaciones $J_i K_i$:

En este punto es donde debemos fijarnos en que los valores de $J_1 K_1$ y $J_0 K_0$, reflejados en la tabla 15, son las entradas de los FF-JK antes del flanco de reloj y que los valores de $J_i K_i$ vienen dados exclusivamente por los valores de x y $Q_1 Q_0$. Por lo tanto, podemos mostrar esta dependencia sobre un mapa de Karnaugh y realizar las simplificaciones permitidas.

La Figura 36 muestra las tablas y la simplificación.

FIGURA 36. Simplificación por Karnaugh para la obtención de los valores $J_i K_i$

$$J_1 = Q_0 \cdot \bar{X} + \bar{Q}_0 \cdot X = Q_0 \oplus X$$

$$K_1 = Q_0 \cdot \bar{X} + \bar{Q}_0 \cdot X = Q_0 \oplus X$$

$$J_0 = 1$$

$$K_0 = 1$$

/RESET = $\overline{\text{RESET}}$

d) Extracción de las ecuaciones Z_i :

Fijándonos en la Tabla 16 el paso al mapa de Karnaugh es inmediato. La Figura 37 muestra la simplificación.

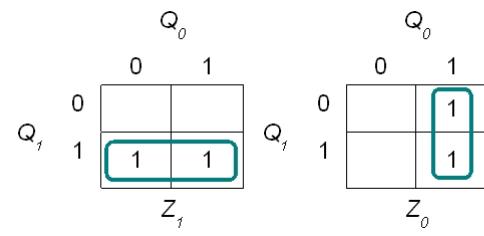


FIGURA 37. Simplificación por Karnaugh para la obtención de los valores Z_i

$$Z_1 = Q_1$$

$$Z_0 = Q_0$$

e) **Esquema del circuito secuencial:**

La Figura 38 muestra la realización del circuito con los FF-JK según el modelo Moore.

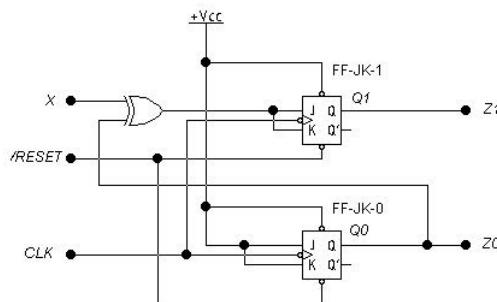


FIGURA 38. Esquema de un contador de 2 bits modelo Moore

4.4.3.2. Síntesis según el modelo Mealy.

1. Descripción del comportamiento:

El circuito a diseñar es el mismo que el planteado en la Subsección 4.4.3.1.

2. Diagrama de estados:

La Figura 39 muestra el “diagrama de estados” del circuito a diseñar:

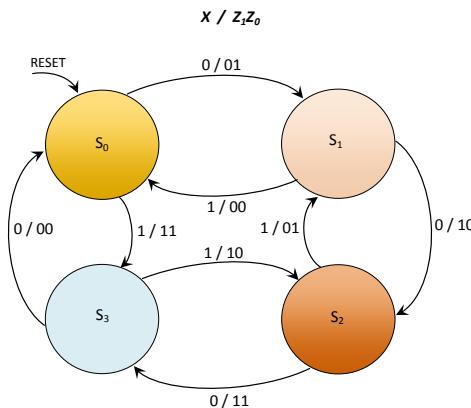


FIGURA 39. Diagrama de estados de un circuito secuencial síncrono contador de 2 bits modelo Mealy - Entrada: X / Salidas: Z_1Z_0

3. Tabla de estados:

Una vez que hayamos reflejado el comportamiento del sistema deseado sobre el diagrama de estados, procederemos de la siguiente forma:

- La información reflejada por el diagrama de estados la escribiremos en forma de “tabla de estados”, con una fila por cada estado, una doble columna por cada valor de entrada contenido en el estado siguiente y el valor de salida correspondiente a la transición al estado siguiente.

La Tabla 17 muestra la confección de la tabla de estados:

CUADRO 17. Tabla de estados del modelo Mealy

Estado actual	$x = 0$		$x = 1$	
	Z_1	Z_0	Z_1	Z_0
S_0	S_1	0 1	S_3	1 1
S_1	S_2	1 0	S_0	0 0
S_2	S_3	1 1	S_1	0 1
S_3	S_0	0 0	S_2	1 0

Al ser un modelo Mealy, las salidas están asociadas a la transición de estados, es decir, a la transición del estado S_0 al estado S_1 le corresponde una salida 01; a la transición del estado S_1 al estado S_2 le corresponde una salida 10 y así sucesivamente.

4. Asignación de estados⁸:

A continuación, se realiza la “asignación de estados” que consiste en dar valores concretos a los estados abstractos.

- Los estados se codifican en variables binarias (asignación de estados), que serán nuestras variables de estado.
- El número de variables binarias vendrá dado por la potencia de 2 que más se ajuste al número de estados requeridos.
- El número de FFs requeridos se corresponde con el número de variables binarias.
- El bit de menor peso se corresponde con el FF-0 y la salida será Q_0 , el siguiente bit se corresponde con el FF-1 y la salida será Q_1 y así sucesivamente.

La tabla 18 refleja esta asignación. Al tratarse de 4 estados se requieren 2 variables binarias. En este caso vemos que la asignación de estados coincide con las salidas del sistema, esta situación se dá normalmente en diseño de contadores que cuentan en binario natural. La asignación podría haber sido otra muy diferente. El comportamiento final del sistema será el mismo, solamente variará la lógica combinacional que conforma el bloque combinacional.

Los criterios que rigen asignaciones diferentes al binario natural, se escapan del objetivo de esta guía.

CUADRO 18. Tabla de estados - asignación de estados

Estado actual	Q_1	Q_0
S_0	0	0
S_1	0	1
S_2	1	0
S_3	1	1

La asignación realizada, se lleva a la tabla de estados general. La Tabla 19 refleja la asignación final:

⁸Se repiten los mismos criterios que los dados en el mismo apartado del modelo Moore, con el fin de poder realizar lecturas independientes.

CUADRO 19. Tabla de estados - asignación de estados final

Q_1	Q_0	$x = 0$				$x = 1$			
		Q_1^*	Q_0^*	Z_1	Z_0	Q_1^*	Q_0^*	Z_1	Z_0
0	0	0	1	0	1	1	1	1	1
0	1	1	0	1	0	0	0	0	0
1	0	1	1	1	1	0	1	0	1
1	1	0	0	0	0	1	0	1	0

5. Síntesis con el FF escogido (D - JK - T):

Se realizan las mismas consideraciones que en el punto 5 de la Subsección 4.4.3.1.

Vamos a realizar la síntesis con FF-D. Primeramente, vamos a fijarnos en el comportamiento de un FF-D, la Tabla 20 muestra, a su izquierda, el comportamiento de un FF-D y, a su derecha, el valor de la salida actual (Q), el valor futuro de la salida (Q^* - valor que adquirirá en el siguiente flanco de reloj) y los valores de las entradas para que se dé la evolución de Q a Q^* :

CUADRO 20. Tabla de verdad de un flip-flop D

clk	D	Q^*	\bar{Q}^*	Q	\bar{Q}	Q^*	D
		0	0				
↑	0	0	1				
↑	1	1	0				

a) Asignación de valores a D_i :

La síntesis requiere el determinar qué valores deben tener las entradas para que, en el siguiente flanco de reloj, las salidas de los FF se correspondan con el “estado siguiente”.

Tomando como referencia la Tabla 19, vamos a generar la tabla que contenga los valores de D_i que determinen la evolución del sistema de un estado actual a un estado futuro. La Tabla 21 muestra los valores de D_i y la evolución de los estados.

CUADRO 21. Tabla conteniendo los valores D_i necesarios

Q_1	Q_0	$x = 0$						$x = 1$						
		D_1	$Q_1^*Q_0^*$	D_0	$Q_1^*Q_0^*$	Z_1	Z_0	D_1	$Q_1^*Q_0^*$	D_0	$Q_1^*Q_0^*$	Z_1	Z_0	
(a)	0	0	0	(0-)	1	(-1)	0	1	1	(1-)	1	(-1)	1	1
(b)	0	1	1	(1-)	0	(-0)	1	0	0	(0-)	0	(-0)	0	0
(c)	1	0	1	(1-)	1	(-1)	1	1	0	(0-)	1	(-1)	0	1
(d)	1	1	0	(0-)	0	(-0)	0	0	1	(1-)	0	(-0)	1	0

Con el fin de facilitar la síntesis, hemos introducido columnas adicionales a la derecha de las columnas D_1 y D_0 , que nos indiquen cual es el estado siguiente, es decir, cual es el valor de la salida Q_1^* y Q_0^* , que va a corresponder a los valores D_1 y D_0 , teniendo en cuenta que D_1 afecta a Q_1^* y D_0 afecta a Q_0^* con la llegada del próximo flanko de reloj.

Con el fin de facilitar la comprensión de la tabla, vamos a revisar alguna de las filas para ver el porqué se han tomado esos y no otros valores de D_i .

- Tomemos la fila (a), vemos que $Q_1 = 0 \rightarrow Q_1^* = 0$ (tomemos la flecha \rightarrow como indicación de evolución) para $x = 0$, por lo tanto $D_1 = 0$, y $Q_0 = 0 \rightarrow Q_0^* = 1$, por lo tanto $D_0 = 1$. Para $x = 1$ tenemos que $Q_1 = 0 \rightarrow Q_1^* = 1$, por lo tanto $D_1 = 1$ y $Q_0 = 0 \rightarrow Q_0^* = 1$, por lo tanto $D_0 = 1$.
- Tomemos la fila (c), vemos que $Q_1 = 1 \rightarrow Q_1^* = 1$ para $x = 0$, por lo tanto $D_1 = 1$, y $Q_0 = 0 \rightarrow Q_0^* = 1$, por lo tanto $D_0 = 1$. Para $x = 1$ tenemos que $Q_1 = 1 \rightarrow Q_1^* = 0$, por lo tanto $D_1 = 0$ y $Q_0 = 0 \rightarrow Q_0^* = 1$, por lo tanto $D_0 = 1$.

Como conclusión de la elección de valores para D_i , podemos simplificar la Tabla 21 ya que el estado siguiente se corresponde con el valor de D_i escogido para la evolución. Por lo tanto, tomando la Tabla 19 y cambiando las variables $Q_1^*Q_0^*$ por D_1D_0 , está resuelta la asignación de valores. La Tabla 22 muestra esta asignación:

CUADRO 22. Tabla simplificada conteniendo los valores D_i necesarios

Q_1	Q_0	$x = 0$				$x = 1$			
		D_1	D_0	Z_1	Z_0	D_1	D_0	Z_1	Z_0
0	0	0	1	0	1	1	1	1	1
0	1	1	0	1	0	0	0	0	0
1	0	1	1	1	1	0	1	0	1
1	1	0	0	0	0	1	0	1	0

b) Asignación de valores a Z_1Z_0 :

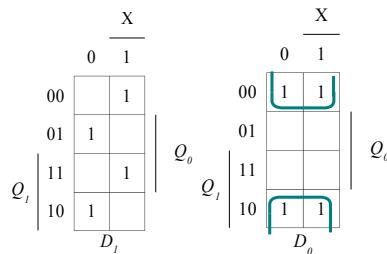
Teniendo en cuenta que las salidas dependen del estado actual y de la entrada, la Tabla 23 refleja esta situación. Esto llevará a dos mapas de Karnaugh de 3 variables, uno para sintetizar la salida Z_1 y otro para sintetizar la salida Z_0 .

CUADRO 23. Tabla de salidas Z_i

Q_1	Q_0	$X = 0$		$X = 1$	
		Z_1	Z_0	Z_1	Z_0
0	0	0	1	1	1
0	1	1	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0

c) Extracción de las ecuaciones D_i :

En este punto es donde debemos fijarnos en que los valores de D_1 y D_0 , reflejados en la tabla 21, son las entradas de los FF-D antes del flanco de reloj y que los valores de D_i vienen dados exclusivamente por los valores de x y Q_1Q_0 . Por lo tanto, podemos mostrar esta dependencia sobre un mapa de Karnaugh y realizar las simplificaciones permitidas.

FIGURA 40. Simplificación por Karnaugh para la obtención de los valores D_i

$$\begin{aligned}D_1 &= X \oplus Q_0 \oplus Q_1 \\D_0 &= \bar{Q}_0 \\/\text{RESET} &= \overline{\text{RESET}}\end{aligned}$$

d) Extracción de las ecuaciones Z_i :

Fijándonos en la Tabla 23 el paso al mapa de Karnaugh es ligeramente más complejo que en el modelo Moore. La Figura 41 muestra la simplificación.

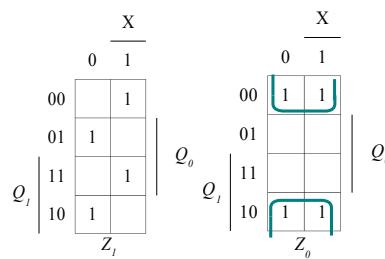


FIGURA 41. Simplificación por Karnaugh para la obtención de los valores Z_i

$$Z_1 = D_1$$

$$Z_0 = D_0$$

e) Esquema del circuito secuencial:

La Figura 42 muestra la realización del circuito con los FF-D según el modelo Mealy.

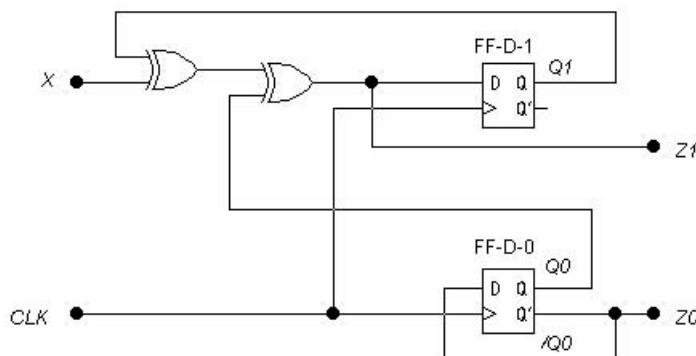
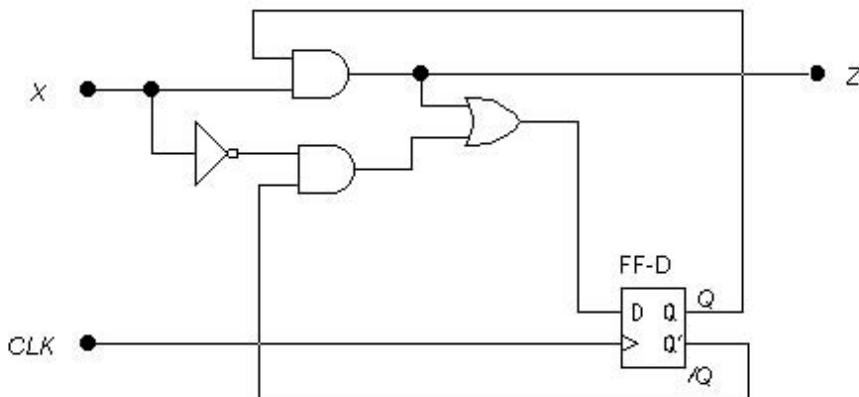


FIGURA 42. Esquema de un contador de 2 bits modelo Mealy

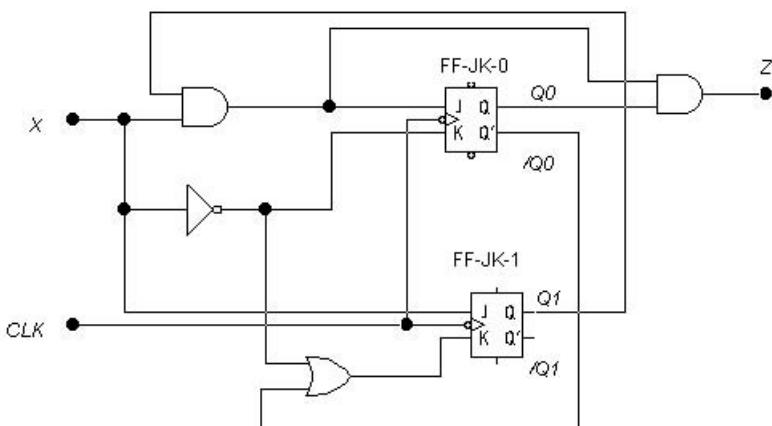
EJERCICIOS

Ejercicio 1: Realizad el análisis de los siguientes circuitos secuenciales síncronos:

a. ¿Modelo Moore o Mealy?



b. ¿Modelo Moore o Mealy?



Ejercicio 2: Diseñad un circuito secuencial síncrono con una señal de entrada X y una señal de salida Z . La señal de salida Z será 1 cuando los valores de X durante los cuatro anteriores períodos de reloj sean 0, 0, 1 y 1. La señal de salida volverá a 0 el período de reloj siguiente a que X vuelva a valer 0. Utiliad flip-flops J-K e includid una señal asíncrona RST que obligará al circuito a reiniciar el ciclo.

- Ejercicio 3: Diseñad un circuito secuencial síncrono con una señal de entrada X por la que recibe datos en formato serie. El circuito produce una señal de salida Z que activa a 1 cada vez que el valor del bit recibido por X cambia, ya sea de 0 a 1 como de 1 a 0. Realizad el diseño utilizando flip-flops tipo J-K e incluid una entrada asíncrona RST que obligará al circuito a reiniciar el ciclo.
- Ejercicio 4: Diseñad un circuito secuencial síncrono con una entrada X y una salida Z , de modo que dicha salida Z se ponga a 1 cuando la entrada X valga 1 durante tres o más flancos de reloj consecutivos. La salida Z sólo vuelve a uno si se rompe la secuencia, es decir si la entrada X vale 0 durante uno o más flancos de reloj. Resolved con flip-flops D e incluid una entrada asíncrona RST que obligará al circuito a reiniciar el ciclo.
- Ejercicio 5: Diseñad un circuito secuencial síncrono que controle el uso de un bus por parte de dos periféricos, A y B. Cuando alguno de ambos periféricos precisa el uso del bus, activa a 1 la señal de petición de bus correspondiente (AR y BR , respectivamente), señales que funcionan como entrada al circuito controlador. Éste concede el uso del bus al periférico que lo ha solicitado, activando a 1 la señal de concesión de bus adecuada (AG y BG), y manteniéndola en este valor hasta que AR o BR vuelvan a 0. La petición y el uso del bus puede pasar de uno a otro periférico en cualquier momento. En caso de que los dos periféricos soliciten el uso del bus al mismo tiempo, (AR y $BR = 11$), el controlador sólo concederá el bus al periférico A. Utilizad flip-flops D e incluid una entrada asíncrona RST que obligará al circuito a reiniciar el ciclo.
- Ejercicio 6: Diseñad un circuito secuencial síncrono con una entrada X y una salida Z , de modo que dicha salida Z se ponga a 1 durante un período de reloj, cuando en la entrada X aparezca la siguiente secuencia sin interrupción: primero 0, luego 1 y después 0. La salida Z vuelve a 0 en el siguiente período de reloj independientemente del resto de valores de X , reiniciando la lectura de la secuencia. Una señal asíncrona I reinicia el circuito en cualquier momento al subir a 1. Resolved con flip-flops J-K.
- Ejercicio 7: Diseñad un circuito secuencial síncrono que funcione como un contador de dos bits Z_1Z_0 , cuya cuenta sólo se inicia si la entrada E vale 1. El contador se quedará en el valor 3 hasta que E vuelva a valer 0, reiniciando su función. El circuito debe reiniciarse también cuando se active a 1 la señal asíncrona RST. Utilizad flip-flops D.
- Ejercicio 8: Diseñad un circuito secuencial síncrono que tenga una salida Z y una entrada X . La salida Z se pone a 1 cuando el valor de

X se repite (dos ceros o dos unos), manteniéndose en ese valor hasta que el valor de X sea distinto al anterior. El circuito debe reiniciarse cuando se active a 1 la señal asíncrona RST. Utilizad flip-flops J-K.

Capítulo 5

MEMORIAS

La memoria es el centinela del cerebro
(William Shakespeare)

RESUMEN. En computación se presenta la necesidad de almacenar la información necesaria tanto para realizar un proceso como para la información generada por dicho proceso. El almacenamiento se va a realizar sobre unos dispositivos denominados memorias. Las memorias van a presentar diversos formatos, tamaños y tecnologías. en este capítulo se realiza una visión exhaustiva de estos dispositivos.

Lo primero que debemos definir es el concepto de “memoria”: “dispositivo capaz de almacenar información”. No se precisa el tipo de dispositivo (soporte), si es mecánico, de estado sólido, magnético u óptico, etc. Tampoco importa el tipo de información, qué es, cómo está codificada, etc...

El funcionamiento básico de la memoria consiste en mantener en el tiempo uno o varios datos y para eso ya conocemos un dispositivo, el registro. La diferencia entre registro y memoria es la cantidad de datos que se pueden almacenar. En los registros, la unidad de información es el dato, es decir, siempre es posible escribir o leer un conjunto único de bits que llamamos dato. En el caso de la memoria, el objetivo del sistema es igualmente escribir o leer un dato, pero ahora éste es sólo un elemento de un conjunto de datos que constituyen la totalidad de la información almacenada en la memoria. Cada uno de estos datos necesita una de referencia que nos permita acceder a él individualmente, esta referencia es lo que llamamos *dirección*. Cada dato, que a partir de ahora llamaremos *palabra*, tendrá unida una dirección que nos permitirá identificarlo. Al contenido de cada dirección de memoria se le llama *posición* de memoria. Las palabras suelen tener un número de bits múltiplo de 8 (octeto o *byte*). El número de bits de la dirección define el número de diferentes direcciones posibles, por lo que fija también la capacidad de la memoria:

$$\text{Capacidad} = \text{nºbitspalabra} \times 2^{\text{nºbitsdirección}}$$

En un sistema digital podemos tener diferentes dispositivos de memoria; el conjunto de palabras que representan toda la información que contienen todos

los dispositivos de memoria del sistema (ver Fig. 1), con cada palabra referenciada con su dirección, representa lo que se llama *mapa de memoria* del sistema digital.

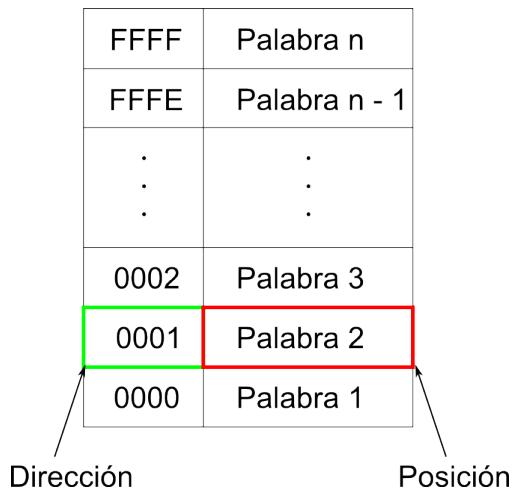


FIGURA 1. Mapa de memoria.

5.1. BLOQUE DE REGISTROS

Los sistemas digitales utilizan datos para realizar operaciones con ellos. A menudo, las operaciones se realizan entre varios operandos, y el resultado es un dato que hay que almacenar, ya sea para presentarlo en el elemento de salida correspondiente o para utilizarlo como dato para la siguiente operación a realizar. Para mantener todos estos datos en el tiempo en que se realizan las operaciones (tiempo de retardo), es preciso mantenerlos en varios registros. Una manera de unificar estos registros en un único dispositivo es el bloque de registros (ver Fig. 2), que consiste en un número reducido de registros que se utilizan como una unidad de memoria con una capacidad limitada, pero de reducido tiempo de acceso a cada dato.

Todos los registros comparten señales de entrada. Aunque todos los datos se colocan en la misma señal de entrada, cada dato se mantiene en un registro diferente. Para ello es necesario activar el proceso de escritura o *carga* en cada registro mediante una señal de habilitación de carga, esta señal permite que el registro se actualice a cada pulso de la señal de reloj (no se ha representado en el diagrama) sólo cuando esté activada (a 1 si es de lógica positiva). Para asegurar que sólo se actualiza un registro y todos los demás continúan con su último valor, las señales de carga las produce un decodificador de tantas salidas como registros tenemos. Al introducir en el decodificador (“Selección

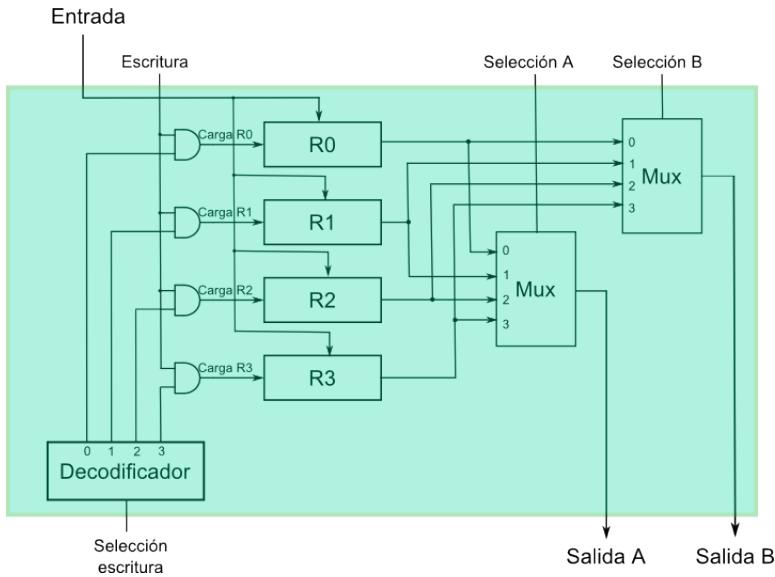


FIGURA 2. Bloque de registros.

escritura") el número correspondiente al registro que queremos actualizar, sólo se activará la salida correspondiente a ese registro, quedando las demás a cero.

En ocasiones es necesario realizar operaciones cuyo resultado no queremos que afecte a los datos guardados en los registros. Para realizar una comparación entre dos datos numéricos, un método muy extendido es realizar la resta de los dos datos y comprobar si el resultado es cero, positivo o negativo. En este caso no interesa guardar el resultado de la operación, por lo que no deseamos que varíe el contenido de ningún resigistro. Las puertas AND en las señales de carga nos permiten asegurar que no se va a actualizar el valor de ningún registro hasta que la señal "Escritura" esté a uno ($0 \cdot X = 0$; $1 \cdot X = X$).

Por otro lado, en este bloque de registros tenemos dos multiplexores dedicados a presentar el contenido de cualquiera de los registros individuales de que disponemos. En cada uno de ellos, la señal de control del multiplexor "Selección" permite mostrar el contenido del registro seleccionado en dos grupos de señales de salida, el bus de salida A y el bus de salida B. De este modo, en cualquier período de reloj basta con activar la señal de selección para obtener dos datos diferentes.

Activando las señales de control correspondientes, este bloque de registros permite en un sólo período de reloj producir dos datos de los cuatro registros posibles, y escribir el valor que esté en el bus de entrada al final del período en el registro seleccionado para escritura (ese registro sólo cambiará al principio del período siguiente).

5.2. MEMORIA

La memoria funciona como un bloque de registros que almacena un número de datos mucho mayor. En la memoria (ver Fig.3) tenemos por tanto un bus de entrada con n bits individuales y un bus de salida con el mismo número de bits, así como un bus de k bits que aporta el número de referencia de cada dato, la dirección. Por otro lado, necesitamos una serie de señales de control para elegir cuál será la operación que realizamos, obtener un dato (“Lectura”) o asegurar que un dato se conserva en el tiempo con la dirección indicada (“Escritura”). La operación de obtener un dato almacenado sólo requiere aportar la dirección del dato, mientras que la operación de guardar un dato en memoria siempre requiere la presencia tanto de la dirección con la que se referencia el dato, como con el dato que vamos a guardar.

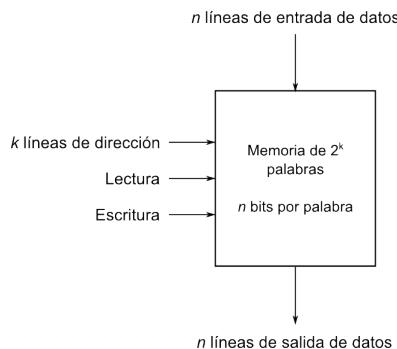


FIGURA 3. Memoria.

5.2.1. Operaciones de memoria: Lectura/Escritura. Las operaciones de memoria son dos: Lectura y Escritura (ver Figs. 4 y 5)

- *Ciclo de escritura:* Para cada dirección, el dato escrito en el bus de entrada de datos será el que se almacene. En esta operación no se utiliza el bus de salida de datos, es decir, no se produce ningún cambio en la salida de la memoria. Al iniciar la escritura, la dirección aparecerá en el bus de direcciones, el dato que guardamos estará en el bus de entrada de datos y activaremos las señales de control: habilitación del dispositivo de memoria y la señal de escritura (en este caso, se comparten las señales para escritura y para lectura; se trata de una única señal que indica la lectura cuando está a “1” y la escritura cuando está a “0”). La operación de escritura requiere un intervalo de tiempo (en este caso transcurren tres períodos de reloj) que llamamos *tiempo de escritura*, al término del cual podemos dar por terminada la operación, desactivando las señales de control y retirando de los buses la dirección y el dato, de este modo es posible reiniciar el ciclo de operación

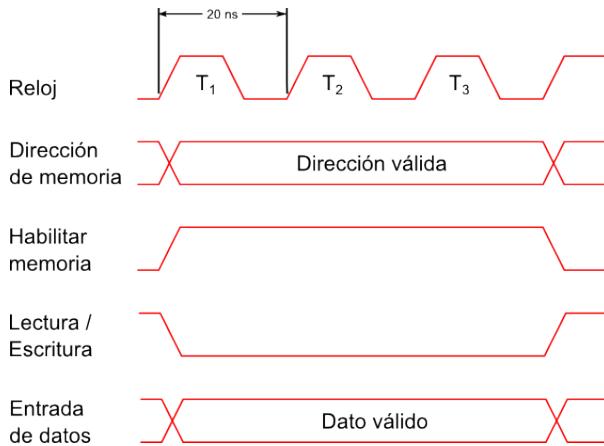


FIGURA 4. Ciclo de escritura.

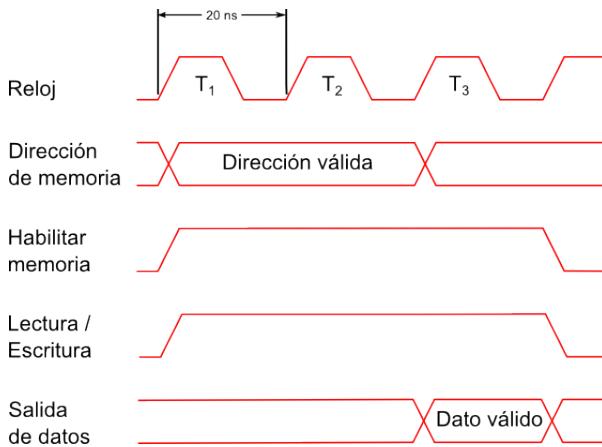


FIGURA 5. Ciclo de lectura.

de memoria con la siguiente operación. Es importante asegurar que se mantienen todas las señales, dirección y dato que guardamos durante todo el tiempo de escritura, pues de lo contrario no aseguramos que se ha realizado correctamente. Esto quiere decir que, si intentamos acceder al contenido de la dirección que hemos utilizado durante la escritura en una operación de lectura, no hay seguridad de que el dato que aparezca en el bus de salida sea el que hemos guardado.

- *Ciclo de lectura:* Para cada dirección, el dato correspondiente aparecerá en el bus de datos. El inicio de la operación de lectura se produce cuando aparece en el bus de direcciones la dirección correspondiente

al dato que queremos leer y producimos las señales de control correspondientes, habilitación del dispositivo de memoria y señal de control de lectura (en este caso la señal Lectura/Escritura se pone a “1” para indicar la lectura). A partir de este momento, el dispositivo está listo para producir el dato que aparecerá en el bus de salida de datos, completando la operación y quedando la memoria lista para reiniciar el ciclo con la siguiente operación de memoria. Es importante resaltar que completar este ciclo requiere un intervalo de tiempo (en este caso es inferior a dos períodos de reloj) que es el transcurrido desde que cambian las señales a la entrada de la memoria (dirección y señales de control) hasta que se observa el cambio del dato en el bus de salida de datos, lo que llamamos el *tiempo de lectura*.

Los tiempos de lectura y escritura conjuntamente constituyen el llamado *tiempo de latencia* o de acceso a memoria, habitualmente el más largo entre los tiempos de retardo en un sistema digital y por ello, el más significativo a la hora de valorar la frecuencia máxima de reloj en un sistema secuencial.

5.3. MEMORIA ESTÁTICA SRAM

La memoria estática permite mantener un dato indefinidamente sin realizar ninguna operación. Este tipo de memoria suele tener una organización interna en la que los datos de todas las posiciones de memoria tienen el mismo tiempo de acceso, de modo que el orden en el que se accede a ellos es irrelevante y puede ser cualquiera, por lo que se la conoce como memoria estática de acceso aleatorio (en inglés *Static Random Access Memory*) o SRAM. La unidad básica de almacenamiento de datos se llama celda binaria porque sólo puede almacenar un bit (ver Fig. 6).

Cada bit es almacenado en un flip-flop SR. Las entradas S y R están unidas a la señal de entrada de datos de modo que un “1” en Entrada provoca el cambio del contenido de la celda a “1”, mientras que un “0” en la entrada lleva al flip-flop al valor “0”. Las dos puertas AND que producen las entradas del flip-flop impiden que se produzca cualquier cambio de estado, salvo si está activada la señal Seleccionar y la señal Lectura/Escritura está a “0”. Del mismo modo, la puerta AND que produce Salida estará siempre a “0” salvo si está activada Seleccionar y la señal Lectura/Escritura está a “1”. En este caso en la salida tendremos el bit que contenga el flip-flop, ya sea “0” o “1”. Por tanto, esta estructura sólo permite cambios del bit almacenado o producir el contenido de la celda si la señal Seleccionar correspondiente a esta celda está activada, eligiendo si la operación es de lectura o de escritura mediante la señal Lectura/Escritura (“1” lectura y “0” escritura).

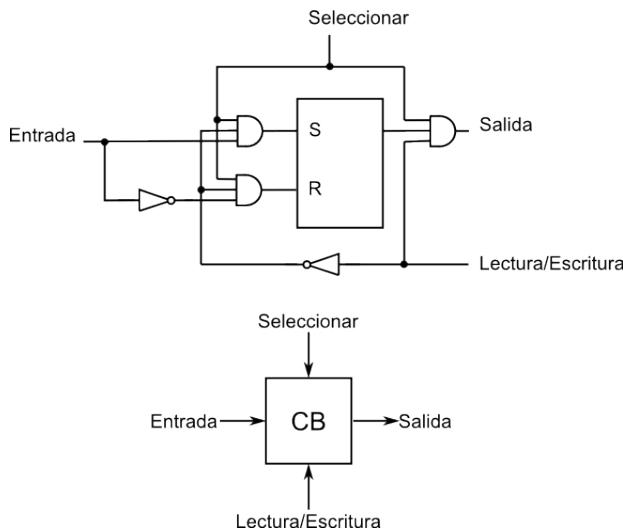


FIGURA 6. Celda binaria SRAM

Partiendo de la celda binaria, es relativamente sencillo diseñar una estructura que permite la escritura en un conjunto de celdas binarias (posición) para un valor numérico concreto (dirección) y que, del mismo modo, produzca el contenido de un conjunto de celdas binarias cuando se indica el valor numérico que referencia ese conjunto de celdas. Esta estructura, que permite la escritura de una palabra de datos en una dirección determinada y que produce una dato concreto al suministrar la dirección correspondiente, es lo que llamamos memoria SRAM (ver Fig. 7)

En este caso se han unido las señales Selección de cada conjunto de cuatro celdas binarias (palabra de 4 bits de longitud), de modo que la activación de cada celda sea simultánea con la de las otras tres celdas que constituyen la posición de memoria. Todas las señales Lectura/Escritura están unidas entre sí, por lo que todas las celdas serán escritas o leídas a la vez. Como las señales Selección son activadas mediante la salida de un decodificador, para cada valor de entrada en él (dirección), sólo un conjunto de celdas está seleccionado y todos los demás no lo están. Todas las salidas de las celdas están unidas a las entradas de una puerta OR, la celda que esté seleccionada y en operación de lectura fijará el valor del correspondiente bit de la salida de datos.

El decodificador tiene una señal de habilitación que permite asegurar que no se producirá ningún cambio en el contenido ni en la salida de la memoria mientras esté desactivada.

A continuación tenéis la misma memoria SRAM, en este caso de 16 palabras con 16 bits de longitud de palabra, descrita en VHDL. Se ha utilizado la función conv_integer para expresar el número entero indicado por la dirección, ya

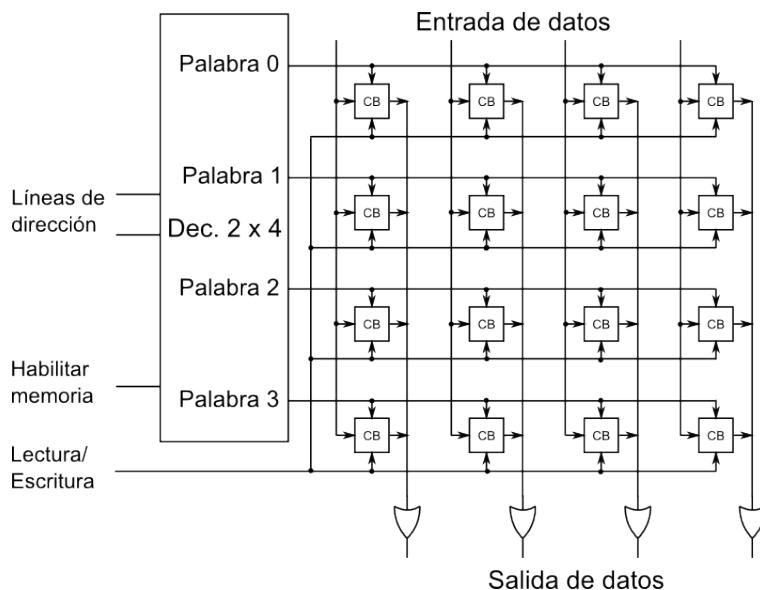


FIGURA 7. Memoria SRAM de 4 palabras x 4 bits cada una.

que lo utilizamos como posición de memoria (índice del array que representan las 16 posiciones de memoria):

```

use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
entity memoria is
Port (
    entrada : in STD_LOGIC_VECTOR (15 downto 0);
    salida : out STD_LOGIC_VECTOR (15 downto 0);
    lect_escr : in STD_LOGIC;
    direc : in STD_LOGIC_VECTOR (3 downto 0);
    habil, clk : in STD_LOGIC
);
end memoria;

architecture a of memoria is
type ram is array (15 downto 0) of std_logic_vector (15 downto 0);
signal estado, estado_nuevo: ram;

```

```

begin
    estado_nuevo(conv_integer(direc))<=entrada when lect_escr='0' and habil='1' else
        estado(conv_integer(direc));
    salida<=estado(conv_integer(direc)) when lect_escr='1' and habil='1' else (OTHERS =>'0');

    process (clk)
    begin
        if clk'event and clk='1' then
            estado<=estado_nuevo;
        end if;
    end process;

end a;

```

En la figura 8 tenéis una simulación funcional (no hay retardos) de la memoria RAM sintetizada por VHDL.

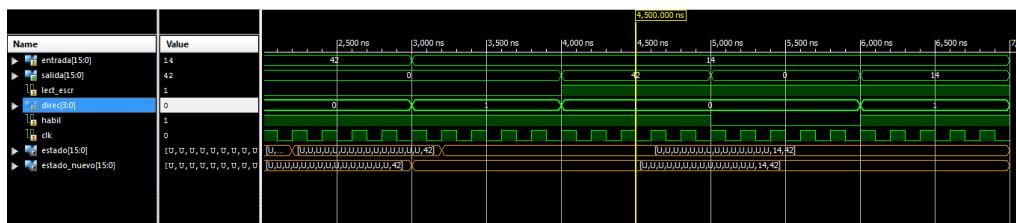


FIGURA 8. Simulación funcional de la memoria SRAM

5.4. MEMORIA DINÁMICA DRAM

Las memorias dinámicas de acceso aleatorio DRAM (en inglés *Dynamic Random Access Memory*) se basan en el uso de un condensador para almacenar información. Puesto que la tensión en el condensador va disminuyendo hasta 0, el dato almacenado en cada celda debe reescribirse periódicamente. Cada celda (ver Fig. 9) está compuesta únicamente de un transistor y un condensador (se usa normalmente el comportamiento capacitivo de un transistor MOSFET), por lo que es mucho más sencilla y ocupa mucho menos espacio de silicio que la celda SRAM, formada por un flip-flop SR y tres puertas AND.

La operación de esta celda depende del estado de conducción o corte del transistor. Si tenemos tensión alta en la puerta del transistor (Dirección a “1”), éste entra en conducción y por tanto no hay caída de tensión en el transistor ni

obstáculo para la circulación de carga a través de él. En este caso, si Entrada dato está a “1”, el condensador se cargará a través del transistor y alcanzará la tensión de alimentación V_{DD} , lo que supone un valor de “1” que se puede observar a través del conector Salida dato. De igual manera, si la señal Entrada dato está a “0”, el condensador se descarga a través del transistor y terminará a 0V, lo que también se refleja en el valor de la señal Salida dato. Si la señal Dirección está a cero, el transistor está en corte, de modo que no permite la circulación de carga a través de él y no es posible alterar la tensión del condensador mediante Entrada dato. Tampoco es posible observar cuál es el valor de tensión del condensador, por lo que no es posible leer el valor de la celda.

La tensión en el condensador cargado es un valor variable en el tiempo que va decreciendo hasta cero durante un tiempo definido por la capacidad y la resistencia del circuito (constante de tiempo $\tau = R \cdot C$), por lo que es preciso realizar periódicamente la lectura de cada celda y la reescritura de un “1” antes de que el contenido de la celda se convierta en un “0” de manera natural. Este proceso se llama *refresco* y todas las memorias DRAM tienen sistemas encargados de realizarlo automáticamente. Una memoria DRAM, por tanto, estará formada por una serie de celdas DRAM, un sistema de refresco y los elementos necesarios para realizar la lectura y la escritura de cada posición de memoria, de la misma forma que en las memorias SRAM.

Por su sencillez, las memorias DRAM permiten la integración de un mayor número de celdas binarias en la misma superficie de silicio, y por ello han desplazado a las memorias SRAM como memoria principal de los computadores.

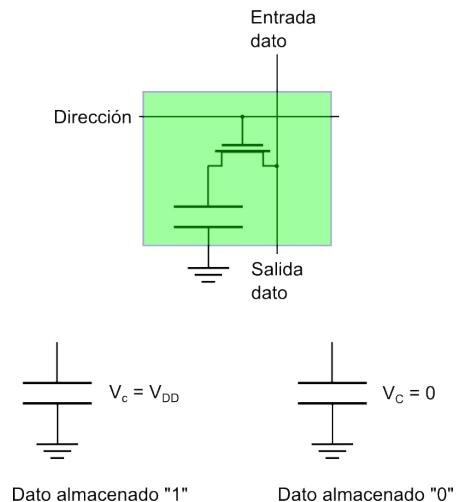


FIGURA 9. Celda DRAM.

5.5. CLASES DE MEMORIAS

A medida que la tecnología de los sistemas digitales, especialmente la tecnología de los computadores, ha ido evolucionando, se han desarrollado una gran cantidad de dispositivos de memoria, basados en diferentes tecnologías y de características muy diversas. En consecuencia, resulta útil plantear una serie de comparativas que permiten conocer mejor sus características.

5.5.1. Mantenimiento de la información. En cuanto a la duración de la información contenida en una memoria, podemos clasificar las memorias de este modo:

- Memoria volátil: se pierden los datos al retirar la alimentación del dispositivo. En este apartado podemos considerar dos tipos:
 - ▷ Memoria estática o regenerativa (SRAM): Los datos se mantienen indefinidamente. Basadas en biestables síncronos, la información se almacenada gracias a la estructura realimentada del elemento de memoria, que alarga indefinidamente el efecto memoria del tiempo de retardo de las puertas lógicas.
 - ▷ Memoria dinámica (DRAM): El mantenimiento de los datos precisa su lectura y reescritura periódica, lo que llamamos refresco de datos. Se usa en aplicaciones que requieren alta capacidad de almacenamiento, ya que la simplicidad de su celda binaria permite integrar un mayor número de celdas en el mismo espacio de silicio (menor coste) que la memoria estática. Su tiempo de acceso es mayor que el de la SRAM.
- Memoria no volátil NVRAM (en inglés *Non Volatile RAM*): los datos se mantienen aunque no haya alimentación. Su implementación se puede realizar mediante un sistema autónomo de alimentación como una pila o mediante la tecnología desarrollada a partir de las memorias sólo de lectura ROM (en inglés *Read Only Memory*), que estudiaremos más adelante.

5.5.2. Modo de acceso a los datos. En cuanto al tiempo de acceso a los datos existe una distinción clara entre dos clases principales:

- Memoria de acceso aleatorio (RAM): El tiempo de acceso es el mismo para toda las posiciones de memoria, por lo que el orden en el que se accede a la información es irrelevante.
- Memoria de acceso secuencial: El tiempo de acceso es función de las posiciones del acceso anterior y el actual. Esta característica aparece en los dispositivos de almacenamiento basados en partículas magnéticas y en los elementos ópticos, ya que las posiciones de memoria se

encuentran distribuidas espacialmente sobre la superficie del disco o cinta. El tiempo necesario para acceder a una posición determinada es función de la posición ocupada por la cabeza lectora al iniciar la operación de lectura, ya que si se encuentra en una localización próxima a la posición de memoria a la que queremos acceder, el desplazamiento será menor que si está alejada de la localización de dicha posición de memoria.

Como en las memorias de acceso secuencial el tiempo de acceso depende de la posición de memoria a la que se accede en la operación anterior, no es posible definir este parámetro en este tipo de memorias. En este caso lo habitual es definir los valores medios de tiempos de acceso a lo largo de un alto número de operaciones de memoria.

5.5.3. Capacidad de lectura y escritura. Aunque hemos definido las operaciones que se pueden realizar con la memoria como de lectura y de escritura, existe un tipo de tecnología en la que sólo es posible la lectura y en la que la información no desaparece cuando se desconecta la alimentación. A partir de este diseño se han desarrollado varias modalidades con diferentes capacidades en cuanto al proceso de escritura.

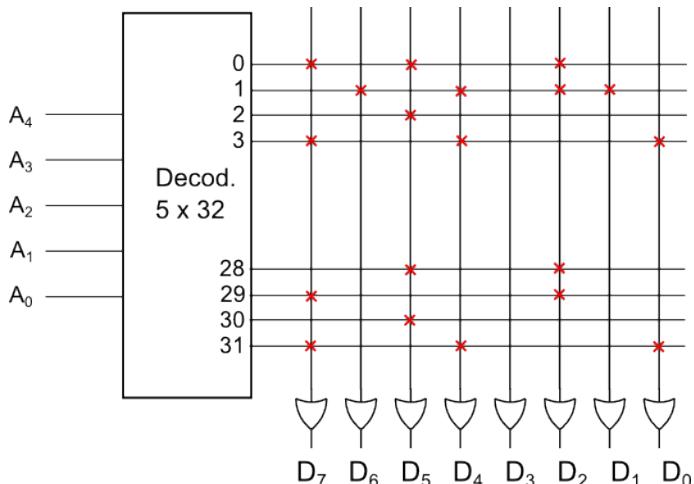


FIGURA 10. Memoria ROM.

- Memoria de sólo lectura ROM: La estructura (ver Fig.10) de este tipo de memoria es muy semejante a la de otras memorias, pero renunciando a la celda binaria que puede almacenar un “1” o un “0”. Si la señal de selección activada por la dirección está conectada al bus de salida de datos, aparecerá un “1” cuando se accede a esta posición de memoria, mientras que si no está conectada aparecerá un “0”.

Al no existir celda binaria, la información no desaparece al desconectar la alimentación. Del mismo modo, no es posible realizar escritura alguna, y la información se introduce en el proceso de fabricación del circuito integrado, según exista o no la conexión entre la línea de selección y la de salida de datos. Las “X” en los puntos en los que se cruzan las salidas del decodificador con las señales de salida representan conexiones eléctricas entre ambas líneas. Para cada dirección de memoria, si existe conexión eléctrica, tendremos un “1” en el bit correspondiente del bus de salida de datos.

- Memoria programable de sólo lectura (en inglés *Programmable Read Only Memory*) PROM: Se trata de memorias ROM en las que se ha introducido una conexión programable en los puntos de cruce entre línea de selección y de salida de datos. En este caso es posible para el usuario definir cuál será la información que escribamos en el dispositivo.

La conexión se realiza mediante una pista conductora que se establece en todos los cruces (todas las palabras de la memoria contienen todo los bits a “1”), pero si aplicamos una intensidad lo suficientemente alta a través de esta conexión, el calor generado por efecto Joule es suficiente para romper el conductor y destruir la conexión, grabando un “0”. Esta tecnología no permite el borrado y escritura de la memoria.

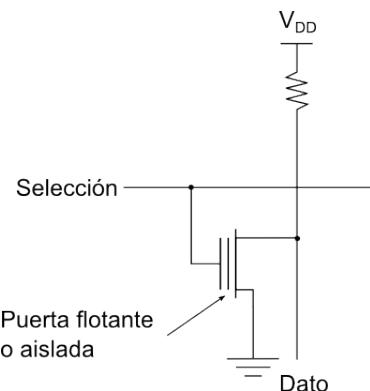


FIGURA 11. Conexión programable.

- Memorias programables borrables de sólo lectura (en inglés *Erasable Programmable Read Only Memory*) EPROM: La conexión se realiza mediante un transistor de puerta flotante que permite mantener la conexión indefinidamente (ver Fig. 11) sin destruir el elemento al programarlo. Se trata de un transistor MOSFET en el que existe una segunda puerta, por debajo del conector de puerta, recubierta totalmente

por un aislante de modo que no tiene conexión eléctrica (por ello se la llama puerta aislada o flotante).

Al aplicar una tensión lo suficientemente alta a la puerta a través de la señal de selección, se aporta carga a la puerta aislada a través de la polarización del aislante, de modo que esta puerta aislada lleva al transistor al estado de conducción. De este modo, la señal de salida de datos queda conectada a través del transistor con tierra y por tanto, en este punto tendremos un “0”. La carga almacenada en la puerta aislada puede ser eliminada mediante luz ultravioleta o mediante la aplicación de tensión eléctrica de polaridad contraria a la utilizada para cargarla (memoria eléctricamente borrable y programable de sólo lectura EEPROM). Las actuales memorias flash utilizan esta tecnología, con la ventaja respecto a las EEPROM de que no es necesario realizar el borrado de toda la memoria al mismo tiempo.

5.5.4. Dispositivos lógicos programables. Aunque no se trata de memorias en el sentido estricto, los dispositivos lógicos programables (Programmable Logic Devices), constituyen una interesante aportación derivada de la tecnología asociada a las memorias ROM y EEPROM. En este caso, se utiliza la matriz de conexiones programables que constituyen el núcleo de las memorias programables como un elemento de creación de relaciones entre las variables de entrada y de salida del dispositivo, es decir, se trata de un circuito capaz de implementar diferentes funciones booleanas, funciones que el usuario puede modificar mediante las conexiones programables.

A diferencia de las puertas lógicas, cada una de las cuales puede realizar una única función booleana, definida en su fabricación, los PLDs pueden realizar la función que diseñemos mediante la modificación de las conexiones, y ésta puede ser modificada cuando lo deseemos. La idea parte de la ROM (ver fig. 10), en la que el conjunto de variables que llamamos dirección ($A_4 - A_0$), produce unos valores determinados en las variables de salida que llamamos dato ($D_7 - D_0$). Si consideramos esas variables como valores lógicos de salida y de entrada, no como dirección o dato, tenemos que cada ROM puede implementar una serie de funciones booleanas definidas por las conexiones de las líneas de selección y de datos, donde cada bit de salida de datos es una función booleana de las variables de dirección. Por ejemplo, como el bit D_0 sólo está conectado con las líneas de selección de la dirección 3 y la 31, sólo valdrá 1 cuando en las variables $A_4 - A_0$ aparezcan dichos valores. Estos valores se corresponden con los valores binarios $A = 00011$ y $A = 11111$, de modo que podemos escribir la función así: $D_0 = \bar{A}_4 \cdot \bar{A}_3 \cdot \bar{A}_2 \cdot A_1 \cdot A_0 + A_4 \cdot A_3 \cdot A_2 \cdot A_1 \cdot A_0$. De manera análoga tenemos las funciones asociadas a los demás líneas de salida $D_7 - D_6$. Estas funciones quedan definidas en la fabricación y por tanto, no se pueden cambiar por el usuario.

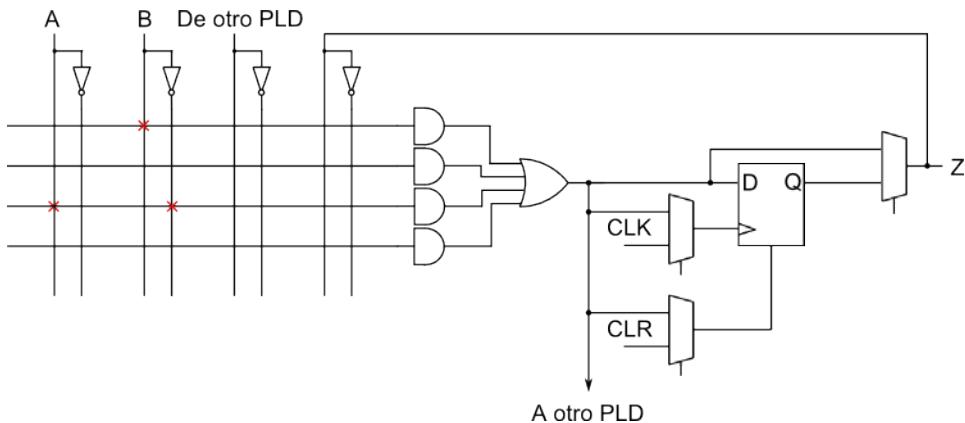


FIGURA 12. Dispositivo lógico programable (PLD)

El siguiente paso es introducir la tecnología de conexión programable para las conexiones (ver fig:11), de modo que sea el usuario quien define la función en suma de productos que aparece en cada línea de salida, función que puede ser borrada; y añadiendo un elemento de memoria a la salida de dichas líneas es posible construir funciones combinacionales y secuenciales en un único circuito integrado que además permite configurar la función como deseemos. Este esquema (ver fig.12) es la base de los PLDs actuales, en el que la conexión programable permite definir una función por suma de productos (puertas AND), que luego podemos almacenar en un flip-flop D. Los multiplexores nos permiten elegir la función combinacional o secuencial, y también cómo implementamos el reloj y la entrada asíncrona del flip-flop. La realimentación de la salida nos permite crear una función secuencial y las entradas adicionales permiten utilizar varios dispositivos en cascada. Este dispositivo es muy semejante a las FPGAs (*Field Programmable Gate Arrays*), en las cuales se ha sustituido la conexión programable por una memoria RAM en la que se escriben los valores binarios que aparecen para cada valor de entrada. Este tipo de tecnología es muy semejante a la de las PLDs, pero como está basada en unidades de memoria estática, es volátil y debe programarse cada vez que se desconecta la alimentación.

Las actuales PLDs y FPGAs están compuestas de miles de celdas lógicas, cada una semejante al dispositivo que acabamos de describir, conectadas entre sí de forma configurable por el usuario, de modo que permiten la configuración de funciones combinacionales y secuenciales muy sofisticadas, en un sólo circuito integrado y con la capacidad de cambiarlas según las necesidades del usuario sin cambiar el diseño del sistema digital, sólo definiendo nuevas funciones para el dispositivo programable.

La configuración de los PLDs se realiza mediante programas de software que interpretan los lenguajes de descripción de hardware (VHDL) y producen las señales de configuración adecuadas en el circuito integrado, de modo que al concluir el proceso de configuración, nuestro PLD realizará la función deseada.

5.6. SOPORTES

Por la tecnología utilizada para el almacenamiento de los datos, podemos clasificarlas en:

- Memorias magnéticas: Se basan en la orientación de partículas magnéticas mediante un campo magnético. Son de este tipo los discos duros magnéticos y las cintas magnéticas.
- Memorias de discos ópticos: La información se graba mediante la definición de pequeñas zonas (“pits”) de diferente reflectividad que el resto de la superficie de los discos. En esta categoría tenemos los CD-ROMs y los DVDs.
- Memorias semiconductoras Circuitos integrados basados en semiconductores (millones de transistores o condensadores en un sólo chip). Actualmente, la memoria principal se basa en memoria semiconductor volátil y dinámica (DRAM)

5.7. ORGANIZACION DEL ALMACENAMIENTO DE DATOS

Existen dispositivos de memoria de diversos tiempos de acceso, capacidad de almacenamiento y coste. Dado que el tiempo de acceso a memoria condiciona el funcionamiento de todo el sistema digital, es habitual combinar varias memorias en un único mapa de memoria.

Existen varios métodos para organizar estos sistemas de memoria:

- Si aumentamos los bits de dirección, aumentamos la capacidad sin modificar la longitud de palabra. Al existir más direcciones, tenemos más posiciones de memoria y la capacidad es mayor.
- Si aumentamos la longitud de palabra sin modificar las direcciones el número de palabras es el mismo pero aumenta el número de bits del bus de salida y entrada de datos a la memoria.

La elección de uno u otro método depende de las características de nuestro sistema digital, no existe un método definido para elegir uno u otro.

EJERCICIOS

- Ejercicio 1: Explica la diferencia entre *dirección* de memoria y *posición* de memoria.
- Ejercicio 2: La capacidad de las siguientes unidades de memoria se especifica por el número de palabras multiplicado por el número de bits por palabra. ¿Cuántas líneas de dirección y líneas de entrada-salida de datos se necesitan en cada caso? ¿Cuál es el número de bytes de capacidad de cada una?
1. $4K \times 8$
 2. $2Gx \times 32$
 3. $16M \times 16$
 4. $256K \times 64$
- Ejercicio 3: En la memoria SRAM de 4×4 bits de capacidad de la Figura 1.7 se usa un decodificador binario natural de 2 entradas y cuatro puertas OR de cuatro entradas cada una. Si queremos expandir la capacidad de esa memoria hasta 256×8 bits, ¿cómo serían el decodificador y las puertas OR necesarias?
- Ejercicio 4: Diseña, utilizando los decodificadores y pueras lógicas que sean necesarios, un sistema de memoria de $4K \times 8$ (1 byte de longitud de palabra) que usa circuitos integrados de $1K \times 4$ bits de capacidad.

Capítulo 6

INTRODUCCIÓN A LA METODOLOGÍA DE DISEÑO DE SISTEMAS DIGITALES

Elegancia es la ciencia de no hacer nada igual que los demás, pareciendo que se hace todo de la misma manera que ellos. (Honoré de Balzac)

RESUMEN. Los diseños digitales complejos requieren un enfoque específico, basado en, primero, la división en ruta de datos, encargada del procesamiento de los datos y, segundo, unidad de control, dedicada a especificar el orden y el ritmo de las operaciones a realizar. Esta estructura es lo suficientemente versátil como para utilizarla en el diseño de sistemas digitales con alto número de estados, registros y elementos combinacionales con gran número de variables binarias. Este enfoque, en combinación con un elemento de memoria que permita almacenar datos y una secuencia ordenada de operaciones (programa), es la base del desarrollo de sistemas digitales programables, los cuales han dado lugar a la tecnología informática como la conocemos actualmente.

6.1. INTRODUCCIÓN

Los sistemas secuenciales sencillos se pueden diseñar mediante un número reducido de estados y elementos de memoria, descritos mediante los diagramas y las tablas de estado, como hemos explicado en el tema anterior. Sin embargo, en el diseño digital nos encontramos a menudo con problemas que implican tareas complejas, que hacen necesario combinar varias decenas de variables binarias con un gran número de estados del sistema. En este caso resulta complicado utilizar las tablas de la verdad y los diagramas de estado, es preciso buscar un enfoque más genérico que nos permita tratar con un esquema semejante los diferentes comportamientos que podemos esperar de un sistema digital.

6.2. DISEÑO DE SISTEMAS DIGITALES COMPLEJOS

En el proceso de buscar la descripción más simple de una tarea compleja, no debemos perder de vista la función final de nuestro sistema. Por un lado, debemos descomponerla hasta el nivel más básico, de modo que las funciones que la componen sean lo bastante sencillas como para que los sistemas digitales que conocemos sean capaces de realizarlas. Por otro, es preciso que estas funciones se realicen de una manera organizada para llegar a completar la tarea que debe cumplir nuestro sistema. Para ello es imprescindible no perder la referencia del conjunto, la visión global que le aporta sentido y permite especificar la secuencia en que deben realizarse las funciones que desarrolla cada uno de los elementos del sistema.

Un enfoque que recoje perfectamente estas dos necesidades, la de reducir nuestro diseño a subsistemas sencillos y la de mantener contacto con el objetivo del diseño definitivo, consiste en plantear el diseño en los siguientes pasos:

- Idear la secuencia de operaciones sencillas que componen nuestra tarea y cómo se relacionan entre ellas.
- Describir con precisión esta secuencia de acciones sencillas.
- Identificar los recursos que permitan realizar cada operación, tanto elementos de memoria como funciones combinacionales, así como los elementos de conexión entre ellos.
- Expressar en forma de circuito secuencial síncrono la secuencia de señales de control que activarán los recursos de memoria o combinacionales, de modo que al completar dicha secuencia tengamos terminada la tarea objetivo del diseño del sistema.

De acuerdo con este enfoque, el diseño de sistemas digitales complejos se puede considerar separado en dos apartados:

- El conjunto de recursos que ejecutan las operaciones sobre los datos, al que llamamos *ruta de datos*.
- Un sistema encargado de controlar la ruta de datos, produciendo las señales de control adecuadas a cada fase de la tarea que realiza el sistema, en la secuencia adecuada, la *unidad de control*.

La Fig. 1 muestra la estructura de bloques de éste planteamiento. De este modo es posible tratar por separado y con cierta independencia, las operaciones que se realizan y la secuencia en que éstas deben cumplirse.

6.2.1. Ruta de datos.

La ruta de datos se encarga de realizar todas y cada una de las operaciones sencillas en que dividimos la tarea que debe realizar el sistema completo.

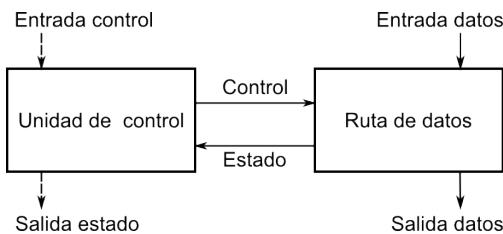


FIGURA 1. Diagrama de bloques de un sistema digital

Para ello, reúne todos los circuitos combinacionales y registros necesarios, así como los buses que permiten el intercambio de datos binarios entre estos elementos. Las señales de *control* de todos estos elementos las recibe de la unidad de control y le envía a ésta todos los resultados intermedios que puedan ser necesarios para determinar la secuencia en que deben realizarse las operaciones. Estos resultados se suelen denominar información de *estado*, pues describen la situación actual de la secuencia de operaciones (ver Fig. 1).

Para que la ruta de datos pueda realizar tareas aritméticas, lógicas y de desplazamiento, generalmente cuenta con registros de almacenamiento y de desplazamiento, UAL (unidad aritmético-lógica) y otros operadores como multiplicadores, operadores en coma flotante, comparadores... y además con una serie de buses y multiplexores encargados de regular el envío de datos binarios entre todos estos elementos.

No podemos hablar de un diseño genérico para la ruta de datos, ya que deberá adaptarse a la lista de funciones en que hemos dividido el objetivo del sistema completo. En cada caso, incluiremos en el diseño de la ruta de datos un circuito diferente por cada función que debe ser realizada.

6.2.2. Unidad de control.

La unidad de control define la secuencia en que se realizarán las funciones en que hemos dividido nuestro problema de diseño. A medida que la ruta de datos va realizando cada paso, la unidad de control habilita y secuencia las operaciones de la ruta de datos, produciendo las señales de control que activan los elementos de la ruta de datos. De este modo, determina la secuencia de ejecución de las operaciones.

En numerosos casos, el funcionamiento del sistema implica la actuación del usuario para controlar la secuencia de acciones: determina cuándo se cargan los operandos, fija cuándo empieza la operación, elige qué camino tomar en función de los resultados intermedios, define cuándo ha terminado la operación. Es la unidad de control la que se comunica con el usuario: recibe entradas de control y emite información sobre el estado de las operaciones.

Para ello definimos un sistema secuencial síncrono, cuyas entradas son las señales enviadas por el usuario y las informaciones de estado que envía la ruta de datos, mientras que las salidas que genera este sistema serán las señales de control que precisa la ruta de datos para realizar todas sus funciones, siempre en el orden adecuado, así como las señales enviadas al usuario para informarle del estado de la operación (ver Fig. 1). Este diseño se realiza como maquina de estados finitos a través del diagrama de estados, como estudiamos en el tema anterior.

6.3. SISTEMA DIGITAL PROGRAMABLE

Hasta ahora hemos supuesto un sistema con una única función, la cual define cómo será el diseño. Para cambiar de función, podemos ampliar el diseño para que abarque más funciones, eligiendo el usuario cuál se realizará en cada momento, mediante la señales de control enviadas a la unidad de control. Este diseño se conoce como *no programable*.

Sin embargo, existe la posibilidad de generalizar este concepto de modo que incluya un campo muy amplio de funciones, las cuales pueden ser especificadas en el orden deseado en una lista. De este modo, alterando la lista conseguiremos cambiar la función que realiza el sistema completo. Es decir, la lista ordenada de funciones que utilicemos en cada caso define cuál es la tarea que realizará nuestro sistema digital. Este tipo de diseño se llama *sistema digital programable*, y a la lista ordenada de funciones la llamamos *programa*. Cada una de las funciones que compone la lista ordenada se llama *instrucción*.

La unidad de control programable es capaz de obtener instrucciones (entradas de control) de una memoria, ejecutando un programa (ver Fig.2). Es éste tipo de diseño, en el que se crea un sistema digital (*hardware*) cuyo propósito es genérico, y que es definido por el programa (*software*) que utilice, el que ha dado lugar a los microprocesadores actuales.

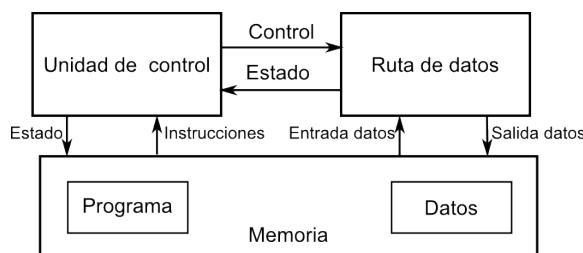


FIGURA 2. Diagrama de bloques de un sistema digital programable

6.4. DIAGRAMA ASM

Los diseño complejos con muchas entradas y salidas son difíciles de abarcar mediante las tablas y los diagramas de estado. Los circuitos secuenciales complejos se pueden describir mediante diagramas ASM (*Algorithmic State Machine*, máquina de estado algorítmica).

Los diagramas ASM representan el comportamiento de los circuitos secuenciales, igual que los diagramas de estado. En la Fig. 3 podemos ver un ejemplo en el que usamos ambos métodos para describir el mismo circuito secuencial.

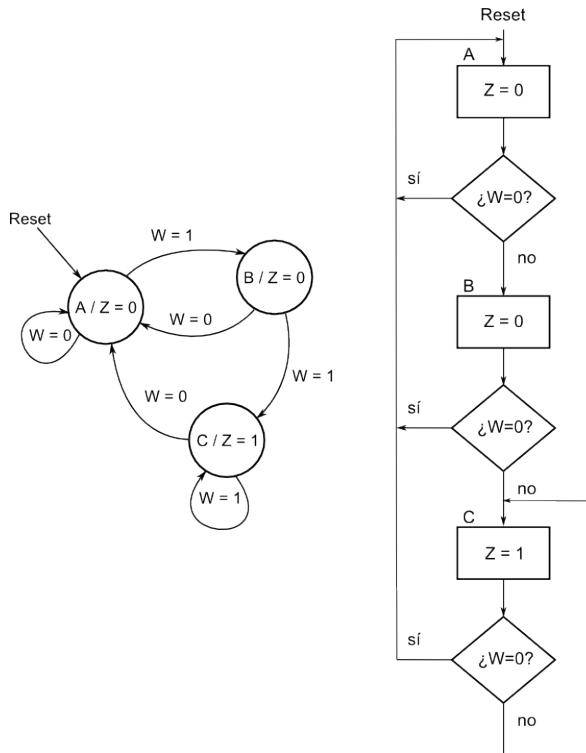


FIGURA 3. Diagrama de estado Moore y diagrama ASM

En el diagrama ASM podemos ver cómo se representa el comportamiento del circuito mediante una serie de bloques unidos por flechas, cada uno de ellos con diferentes funciones (estado, decisión, salida condicional), lo que es representado con bloques de diferentes formas. La Fig. 4 muestra los tres bloques de funciones comentados.

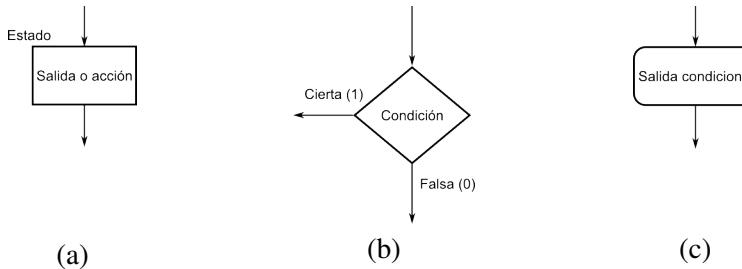


FIGURA 4. Elementos del diagrama ASM

A continuación veremos las funciones de cada elemento y la manera de reconocerlos.

6.4.1. Estado.

Representa un estado y los valores asociados a él. Es de forma rectangular, en su interior se escriben los valores de la salida asociados al estado (modelo Moore) y encima a la izquierda se escribe el nombre del estado asociado (ver Fig. 4(a)). También se pueden usar descripciones abreviadas con acciones completas asociadas al estado actual.

6.4.2. Decisión.

Define las posibles transiciones de estado, así como valores de variables o acciones, en función del cumplimiento o no de una condición. La condición a testear se establece en función del valor de una variable de entrada a nuestro circuito secuencial. En función de la condición se elige una flecha u otra, cada una llevará a diferentes estados. Se representa con forma de rombo dentro del que se escribe la condición para la decisión (ver Fig. 4(b)).

6.4.3. Salida condicional.

Establece un valor de una variable de salida, en el caso que su valor dependa de una variable de entrada. Por ello, siempre está al final de un elemento de decisión. Tiene forma de rectángulo con los vértices redondeados (ver Fig. 4(c)). En su interior se escribe el valor que adopta la salida condicional (modelo Mealy).

6.5. EJEMPLOS DE DISEÑO

A continuación vamos a plantear algunos diseños de sistemas digitales en los que utilizaremos los diagramas ASM y el enfoque ruta de datos/unidad de control. Para ello, partiendo de la descripción de las funciones que debe realizar nuestro circuito, estableceremos las siguientes fases:

- Empezamos por una lista de acciones o algoritmo.
- Siguiendo el algoritmo, realizamos un diagrama de bloques de la ruta de datos.
- Diseñamos un diagrama de conexión entre la ruta de datos y la unidad de control, indicando las señales de control.
- Realizaremos un diagrama ASM de la unidad de control, con todas las señales.

6.5.1. Contador de unos.

Diseñar un circuito que cuente el número de unos de un dato binario de n bits. Al inicio de cualquier proceso de diseño debe realizarse un análisis que permita dividir una operación en los elementos más simples posibles. Es intuitivo entender este caso: se trata de ir observando uno por uno todos los bits del dato y aumentar la cuenta sólo cuando el bit observado sea 1. Cuando hayamos observado todos los bits del dato, el número al que hayamos llegado en nuestra cuenta será el número de unos del dato.

Llamaremos A ($A_{n-1}A_{n-2} \dots A_0$) al dato cuyo número de unos queremos contar (y al registro que lo mantiene durante la operación) y B al resultado de nuestro conteo (y también al registro que lo mantendrá mientras el usuario lo lee). En ese caso, una representación de la secuencia de acciones que requiere esta operación es:

$$B \leftarrow 0$$

Repetir si $A \neq 0$:

Si $A_0 = 1$,

$$B \leftarrow B + 1$$

$A \leftarrow$ desplazar derecha A

Desplazando a la derecha todos los bits de A y observando siempre el bit menos significativo A_0 , podemos comprobar uno a uno si los bits del dato A son uno o cero. Si hemos desplazado a la derecha todos los bits del dato, en el registro sólo tendremos ceros, por lo que ha terminado la operación.

Teniendo en cuenta una señal externa S que indique cuándo se ha terminado de escribir el dato A y se desea iniciar el conteo de los unos (mientras sea '1' no tenemos nuevo dato), podemos reflejar el anterior esquema de operaciones en el siguiente diagrama ASM (ver Fig. 5).

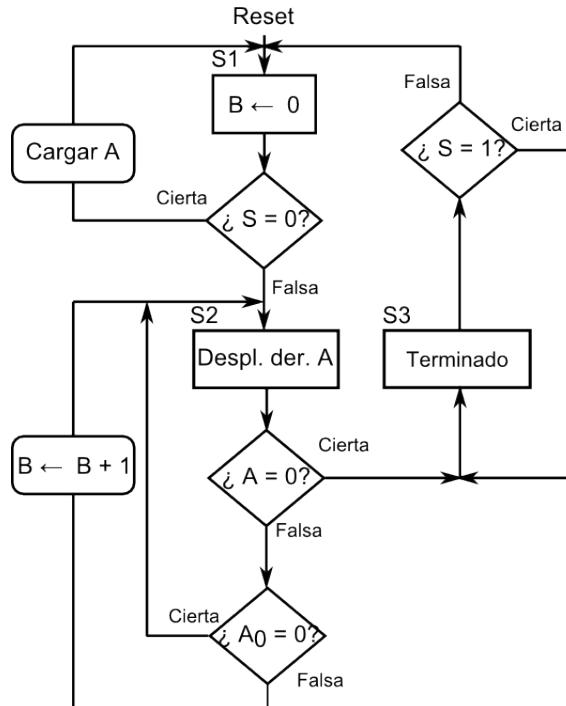


FIGURA 5. Diagrama ASM del contador de 1s.

El diagrama ASM se corresponde con el algoritmo que hemos desarrollado, incluyendo además una señal *Terminado* que indica al usuario que el proceso de conteo ha terminado y el contenido de B es el resultado (observe que $B = B + 1$ está en una caja "condicional" ya que depende de la variable A_0). No olvidemos que, en cualquier circuito secuencial, la actualización de los registros se produce después del cambio de período, cuando ha aparecido un nuevo flanko activo de reloj. En el estado S2 no tenemos desplazamiento del contenido del registro A hasta que termine el período de reloj actual. En el esquema de la Figura 6, se pueden observar los circuitos que realizan las operaciones con los datos que hemos especificado en el algoritmo, circuitos que constituyen la ruta de datos de nuestro sistema.

Esta ruta de datos necesita un desplazador a la derecha para observar todos los bits de A mediante el bit menos significativo del desplazador A_0 , un

contador que va incrementando B cada vez que A_0 es uno y una señal combinatorial Z que compruebe si $A = 0$ (es decir, si todos sus bits son cero, para lo que basta una puerta NOR de todos los bits de A). En la Figura 6 podemos ver la estructura de este sistema, compartiendo la misma señal de reloj de modo que el desplazamiento y el conteo se coordinen, y con una serie de señales de control que permiten realizar la operación siguiendo los pasos indicados en el diagrama ASM segúan la secuencia indicada:

- Para el desplazador: una señal de habilitación EA, una señal de carga paralelo LA y una entrada serie ES, en este caso siempre a cero.
- Para el contador: una señal de habilitación EB y una señal de carga paralelo LB.

Para producir las señales de control necesitamos un sistema digital que vaya generando la secuencia adecuada de dichas señales, es decir, una unidad de control.

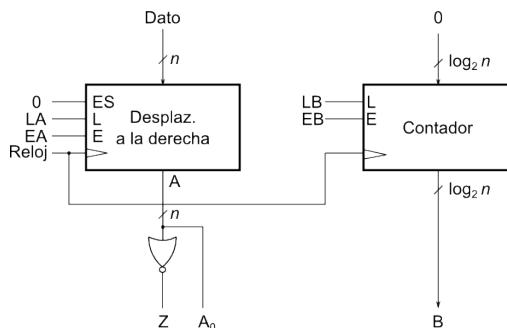


FIGURA 6. Ruta de datos del contador de 1s.

En la Figura 7 podemos ver el diagrama de bloques de nuestro sistema completo, en él aparece la unidad de control mandando las señales de control a la ruta de datos y recibiendo de ésta la información necesaria para cumplir la secuencia en el orden correcto (cuándo es 1 el bit A_0 y cuándo hemos comprobado todos los bits de A).

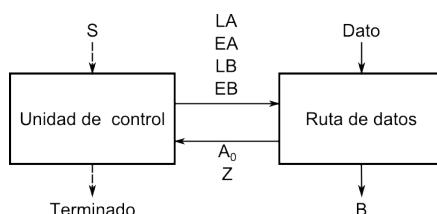


FIGURA 7. Diagrama de bloques del contador de 1s.

El diseño detallado de la unidad de control debe incluir todas las señales de control que producirán la secuencia de tareas que requiere la operación, así como las señales de control del sistema completo (S) y las que informen al usuario (*Terminado*). En la Figura 8 podemos ver cómo este esquema se corresponde con el algoritmo con el que iniciamos el diseño, pero ahora sólo aparecen las señales de entrada y de salida. La señal *Reset* permite regresar al sistema al estado inicial y dejar a cero el registro B que presenta el resultado, con todo el sistema preparado para recibir el siguiente dato.

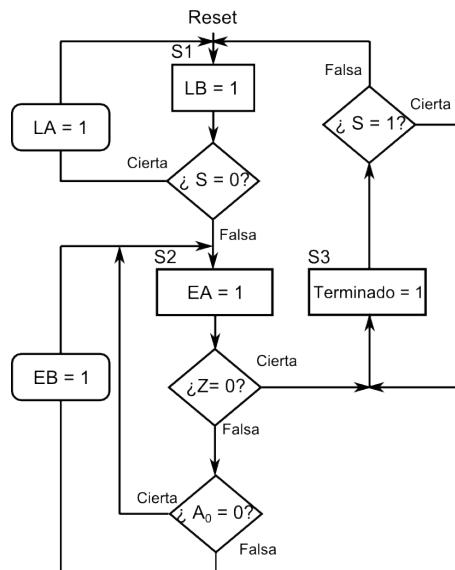


FIGURA 8. Diagrama ASM de la unidad de control.

6.5.2. Divisor binario.

Realizaremos el diseño de un divisor binario en el que supondremos tanto al dividendo como al divisor como números sin signo de n bits. Para analizar las tareas de que se compone esta operación, recordemos el método para la división en decimal (ver Fig. 9), para lo cual escogemos dividendo y divisor con todas sus cifras 0 ó 1:

Dividendo		Divisor
1 0 0 1 0 1 0		1000
- 1 0 0 0		1001
<hr/>		Cociente (q)
1 0		
1 0 1		
1 0 1 0		
- 1 0 0 0		
<hr/>		
1 0	Resto	

FIGURA 9. División en base 10.

Si nos fijamos en la división de la Fig. 9 observamos que el dividendo se va desplazando hacia la izquierda, dejando un dígito (bit) hasta poder realizar la resta con el divisor. La idea de este sistema es ir desplazando el dividendo hacia la izquierda partiendo de cero y añadiendo la cifra más significativa, cada vez que el grupo de cifras que se va obteniendo de este modo es mayor que el divisor, se resta y se añade un 1 al cociente (que empieza en cero y se va desplazando hacia la izquierda) en la posición menos significativa (de lo contrario se añade un 0); al resultado de la resta se le siguen añadiendo cifras del dividendo y se repite la comparación con el divisor y la resta, este proceso continúa hasta que hemos recorrido todas. Podemos resumir el proceso de la siguiente manera:

Cargar *Divisor*, *Dividendo* en *Resto_{bajo}*

Si *Divisor* = 0, salir

Repetir desde $i = n - 1$ hasta $i = 0$

Resto \leftarrow Desp. izquierda *Resto*

Si *Resto_{alto}* − *Divisor* ≥ 0 ,

Resto_{alto} \leftarrow *Resto_{alto}* − *divisor*

Entrada *Cociente* = 1 (se carga bit a bit (q_i) desplazando a la izda.)

Si *Resto_{alto}* − *Divisor* < 0,

Entrada *Cociente* = 0 (se carga bit a bit (q_i) desplazando a la izda.)

$i \leftarrow i - 1$

El resto de la división es el resultado de la última resta almacenada. Si utilizamos un registro de desplazamiento a la izquierda de $2n$ bits para el dividendo, tenemos un espacio de n bits que se va vaciando a medida que avanzamos en la operación y no vuelve a utilizarse hasta la siguiente operación, cuando se

escribirá el nuevo divisor. Ese espacio puede utilizarse para almacenar el cociente, ya que se va desplazando a la izquierda a medida que más cifras del dividendo entran en la resta. Este algoritmo considera la posibilidad de que el divisor fuese cero, en ese caso evitamos ejecutar cualquier acción.

Analicemos cómo se desarrolla un ejemplo de división realizada por este método, suponiendo datos de cuatro bits (ver Fig. 10). En cada paso se realiza el desplazamiento del dividendo a la izquierda y la resta de la parte desplazada del dividendo menos el divisor, que se realiza mediante la suma del complemento a dos del divisor, pero este resultado sólo se guarda cuando el resultado es positivo (bit de signo a 0, en azul). Al mismo tiempo se introduce un bit nuevo (un 1 si la resta es positiva, un 0 si es negativa) en la parte baja de Resto, lo que va conformando el cociente.

		Dividendo: 7	Divisor: 3	Comp. a 2 de 3
		Restoalto	Restobajo	
<i>Resto:</i> registro de 8 bits				
Divisor en Comp. a 2:	+ 1 1 0 1			
Si <i>MSB</i> = 1 $\Rightarrow q_3 = 0$	1 1 0 1			
Si <i>MSB</i> = 1 $\Rightarrow q_2 = 0$	0 0 0 1	1 1 0 0	Despl. izda.	Cuenta: 1
	+ 1 1 0 1			
Si <i>MSB</i> = 1 $\Rightarrow q_1 = 1$	1 1 1 0	↑ <i>q</i> ₀	(cociente)	
	0 0 1 1	1 0 0 0	Despl. izda.	Cuenta: 2
	+ 1 1 0 1	↑ <i>q</i> ₀	↑ <i>q</i> ₁	(cociente)
Si <i>MSB</i> = 0 $\Rightarrow q_1 = 1$	0 0 0 0	1 0 0 0	Despl. izda.	Cuenta: 3
	+ 1 1 0 1	↑ <i>q</i> ₀	↑ <i>q</i> ₁	(cociente)
Si <i>MSB</i> = 1 $\Rightarrow q_0 = 0$	1 1 1 0	0 0 0 1	Despl. izda.	Cuenta: 4
Comp. a 2 del resto:	0 0 1 0	↑ <i>q</i> ₀	↑ <i>q</i> ₁	(cociente)
	+ 1 1 0 1	↑ <i>q</i> ₀	↑ <i>q</i> ₁	
	Resto: 1	↑ <i>q</i> ₃	↑ <i>q</i> ₂	Despl. izda.
		↑ <i>q</i> ₁	↑ <i>q</i> ₀	(cociente)
			Cociente: 2	

FIGURA 10. Ejemplo de división binaria de cuatro bits.

Observad cómo es introducido un dígito (en rojo) en Resto durante el primer desplazamiento, pero no tiene relación con el signo de la resta, por lo que es erróneo. Esta cifra incorrecta se queda en el resto al acabar la operación, por lo que del resultado correcto del resto hay que descartar el bit menos significativo. Esto quiere decir que el resto tiene $n - 1$ bits, pero tened en cuenta que el resto es siempre menor que el divisor, por lo que no habrá desbordamiento en el resto.

Es interesante observar también que la actualización de Resto cuando su signo es positivo se produce antes de que acabe el período de reloj (carga

asíncrona) para que se pueda desplazar el nuevo resto al inicio del siguiente período de reloj. El último paso se produce cuando el número de posiciones desplazadas es el número n de bits del dividendo (4 en este caso), dejando en la parte baja del registro Resto el cociente, y en la parte alta (descartando el quinto bit) aparece el resto de la división.

La Figura 11 nos muestra el diagrama ASM correspondiente a nuestro diseño del divisor. Como se puede ver, incluye un registro Divisor, el registro de desplazamiento a la izquierda Resto, con la parte alta y la parte baja diferenciadas, y la señal S que indica cuándo el usuario ha terminado de escribir el dividendo y el divisor en la entrada paralela de ambos registros. Es necesario un contador Cuenta que utilizaremos para saber cuándo hemos terminado de recorrer los n bits del dividendo.

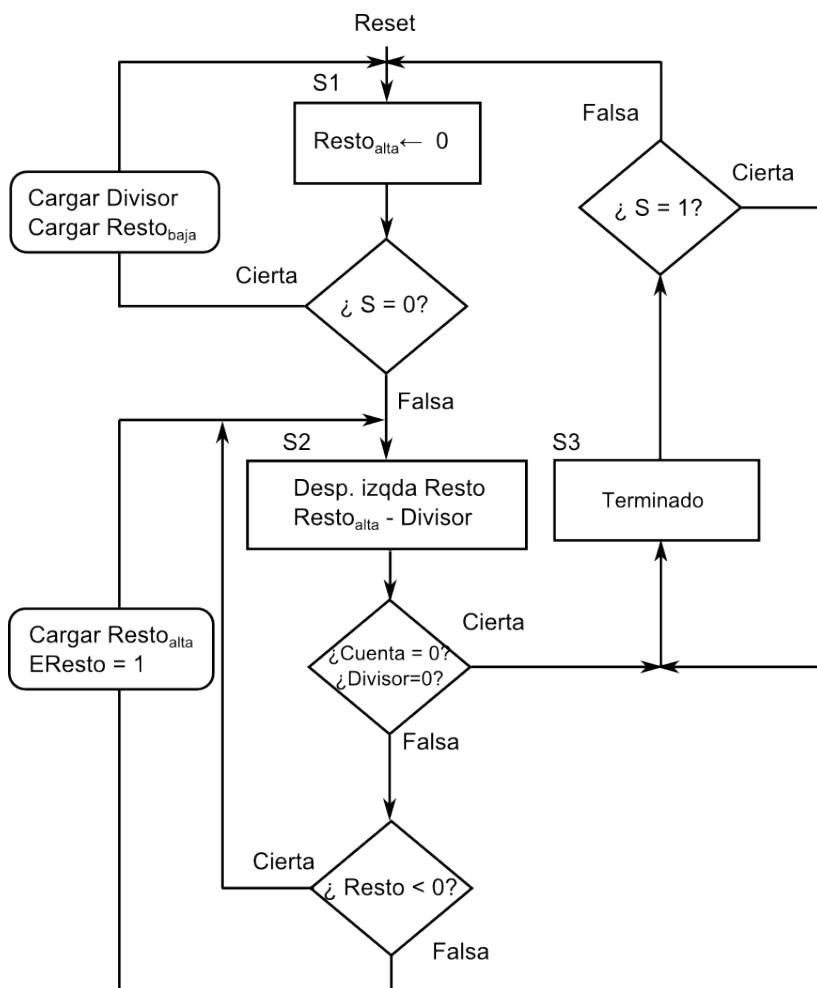


FIGURA 11. Diagrama ASM del divisor

Como en diseño del contador de unos, el siguiente paso es desarrollar el diseño de la ruta de datos que realizará las acciones que acabamos de especificar.

Podemos ver el esquema completo en la Figura 12, en donde hemos incluído, además de los registros Resto y Divisor, el contador para Cuenta y las señales que indican la aparición del cero (basta con una puerta NOR de n entradas) en Cuenta (*Fin*) y en Divisor (*Z*). Junto con el bit de signo de la resta (*Signo Resto*), constituyen la información de estado que requiere la unidad de control. Las señales de control son:

- Para el registro: habilitacion ED y carga paralelo LD.
- Para el registro de desplazamiento a la izquierda: habilitacion ER, carga paralelo LR y entrada serie ES (controlada por la negación de *Signo Resto*).
- Para el contador: habilitación EC, carga paralelo LC.
- Para el multiplexor la señal de selección *Sel*.

En este esquema las flechas indican buses de n bits (salvo *Resto*, que son sólo $n-1$ bits).

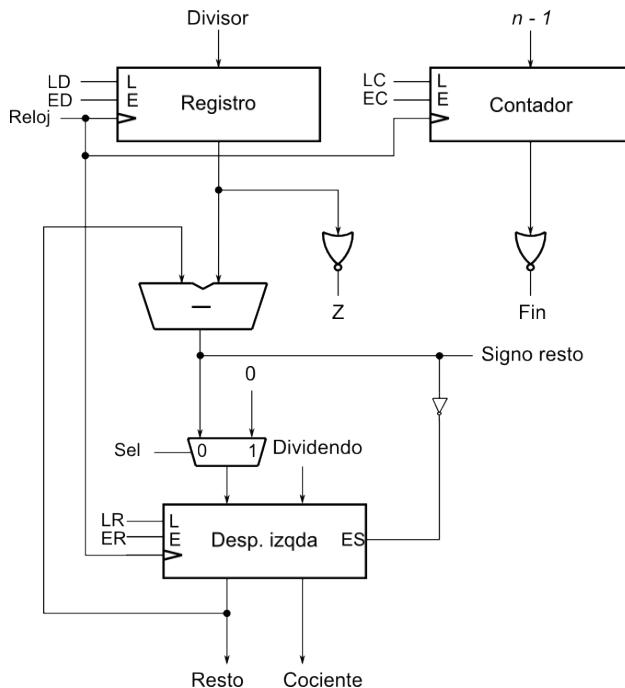


FIGURA 12. Ruta de datos del divisor.

En la Figura 13 aparece el sistema divisor completo, con todas las señales que comunican la unidad de control y la ruta de datos, así como las señales del usuario.

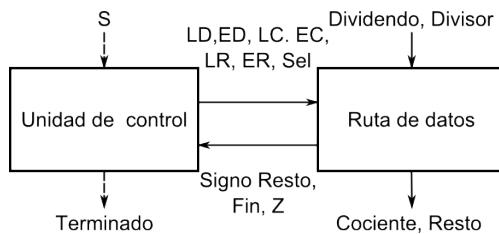


FIGURA 13. Diagrama de bloques del divisor.

Finalmente, hay que realizar el diseño definitivo de la unidad de control. La Figura 14 recoge el diagrama ASM de la unidad de control, con todas las señales asignadas. Este sistema secuencial permite realizar todas las tareas necesarias para la división, mediante la ruta de datos que hemos propuesto.

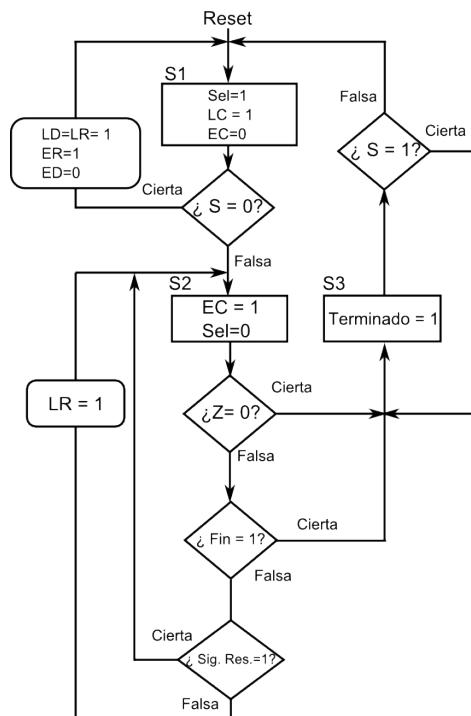


FIGURA 14. Diagrama ASM de la unidad de control del divisor.

EJERCICIOS

Ejercicio 1: Realiza, con la ayuda de un cronograma que contenga todas las señales, el análisis del funcionamiento del sistema descrito por el siguiente diagrama ASM (ver Fig. 15), en el que a partir de una señal S, se controla el funcionamiento de un contador de salida A ($A_2A_1A_0$, 3 bits) y las señales de salida E y F:

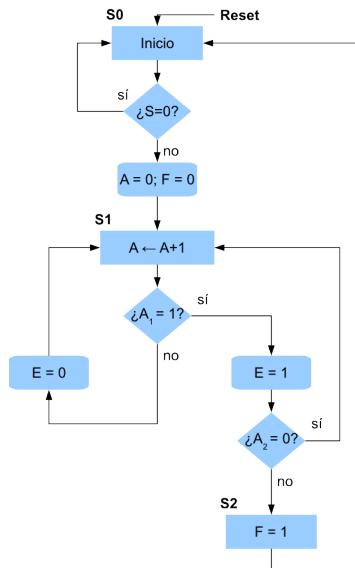


FIGURA 15. Diagrama ASM para analizar

Ejercicio 2: Diseñad un circuito secuencial síncrono que realice la función de multiplicar dos números enteros sin signo de 4 bit. El resultado aparecerá en un dato de 8 bits, junto con un bit que indique cuándo se ha terminado la operación.

- NOTA: El diseño puede ser semejante a la multiplicación en papel, es decir, por cada cifra del multiplicador, se va sumando el multiplicando desplazado una posición a la izquierda al resultado anterior. Tened en cuenta que cada cifra del multiplicador sólo puede ser 0 ó 1. El bit para indicar que se ha terminado la operación se puede obtener con un registro de desplazamiento a la derecha del multiplicador que sea 1 cuando todos los bits del registro sean 0.

$$\begin{array}{r} & 1 & 1 & 0 & 1 & \text{Multiplicador} \\ \times & 1 & 0 & 1 & 1 & \text{Multiplicando} \\ \hline & 1 & 1 & 0 & 1 \\ & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ \hline & 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \text{Resultado} \end{array}$$

Bibliografía

- [1] G. Bosque, P. Fernandez, Principios de Diseño de Sistemas Digitales - Guía Práctica para Alumnos de Primer Curso de Grado en Ingeniería Informática de Gestión y Sistemas de Información, UPV-EHU, 2014.
- [2] O. Arbelaitz, O. Arregi, A. Aruabarrena, I. Etxeberria, A. Ibarra, T. Ruiz, Principios de Diseño de Sistemas Digitales: Conceptos básicos y ejemplos, UPV/EHU, 2008.
- [3] T. Floyd, Fundamentos de Sistemas Digitales, Prentice Hall, 2000.
- [4] M. Morris-Mano, Diseño Digital, Prentice Hall, 2003.
- [5] D. Gajski, Principios de Diseño Digital, Prentice Hall, 1997.
- [6] J. Hayes, Introducción al Diseño Lógico Digital, Addison-Wesley Iberoamericana, 1996.
- [7] A. Lloris, A. Prieto, L. Parrilla, Sistemas Digitales, McGraw-Hill, 2003.
- [8] J. Uyemura, Diseño de Sistemas Digitales. Un Enfoque Integrado, Thomson, 2000.
- [9] D. A. Patterson, J. L. Hennesy, Organización y Diseño de Computadores, McGraw-Hill, 1994.
- [10] M. Ercegovac, T. Lang, J. H. Moreno, Introduction to Digital Systems, John Wiley and Sons, 1999.
- [11] P. Ashenden, Digital Design. An Embedded Systems Approach Using VHDL, Morgan Kaufmann, 2008.
- [12] J. Hamblen, T.S.Hall, M. Furman, Rapid Prototyping of Digital Systems. SoPC Edition, Springer, 2008.
- [13] M. Morris-Mano, C. R. Kime, Fundamentos de Diseño Lógico y Computadoras, Prentice-Hall, 2005.

Índice alfabético

- Algebra de Boole, Identidades, 62
- Algebra de Boole, Postulados, 62
- Análisis y Síntesis de Sistemas
 - Secuenciales Síncrono
 - Modelo Mealy, 137
 - Modelo Moore, 138
 - Síntesis de Sistemas Secuenciales Síncronos, 137
 - Análisis y Síntesis de Sistemas Secuenciales Síncronos
 - Análisis de Sistemas Secuenciales Síncronos, 135
- Aplicaciones, codificador-decodificador, 112
- Biestables asíncronos
 - Latch D, 122
 - Latch S-R, 118
- Biestables síncronos
 - Entradas asíncronas, 127
 - FF-D, 123
 - FF-JK, 125
 - FF-T, 126
- Bloque de registros, 156
- Bloques combinacionales - Introducción, 89
- Códigos alfanuméricos, 53
 - Código ASCII, 54
 - Código UNICODE, 55
 - Latin-1, 54
 - página de código, 55
 - punto de código, 55
- Circuitos Secuenciales, 116
 - Biestables asíncronos, 118
 - Biestables síncronos, 123
 - Entradas Asíncronas, 127
 - Estado, 116
 - Flip-Flop D, 123
- Flip-Flop JK, 125
- Flip-Flop T, 126
- Latch D, 122
- Latch S-R, 118
- Modos de activación, 123
- Clases de memorias, 165
 - Capacidad de lectura y escritura, 166
 - Mantenimiento de la información, 165
 - Modo de acceso a los datos, 165
- Codificador, 107
- Codificador con prioridad, 108
- Decodificador, 108
- Decodificador: 2 bits con habilitación, 109
- Decodificador: 4 bits con habilitación, 110
- Decodificadores, aplicaciones, 111
- Demultiplexor, 99
- Demultiplexor 1 a 4, 100
- Diagrama ASM
 - Decisión, 178
 - Estado, 178
 - Salida condicional, 178
- Diseño de sistemas digitales complejos, 174
 - Ruta de datos, 174
 - Unidad de control, 175
- Dispositivos lógicos programables (PLD), 168
- Ejemplos de diseño, 179
 - Contador de unos, 179
- Formas canónicas, 64
- Glosario, Términos y Acrónimos, 23
- Karnaugh, especificación incompleta, 72
- Karnaugh, Maxterminos, 71
- Karnaugh, mintérminos, 68

- Lenguaje de descripción de hardware:
VHDL, 82
- Max términos, 65
- Memoria, 158
Operaciones de memoria:
Lectura/Escritura, 158
- Memoria dinámica DRAM, 163
- Memoria estática SRAM, 160
- mintérminos, 65
- Multiplexor, 91
- Multiplexor 2 a 1, 92
- Multiplexor 4 a 1, 93
- Multiplexor, entrada de habilitación, 94
- Multiplexores, aplicaciones, 95
- Multiplexores, buses, 95
- Multiplexores, implementación de funciones, 96
- Números en coma flotante, 48
Norma IEEE 754, 51
Doble precisión, 52
NaN, 52
Precisión sencilla, 52
Notación científica, 49
- Organización del almacenamiento de datos, 170
- Puertas lógicas, 74
- Registros y Contadores, 128
Contadores, 131
Registros, 128
- Representación de números con signo, 40
Complemento a dos, 44
Complemento a uno, 42
desbordamiento, 47
Exceso a 2^{n-1} , 47
Magnitud con signo, 40
suma binaria, 45
- Resolución de los ejercicios, 209
Tema 1, 209
Tema 2, 214
Tema 3, 224
Tema 4, 233
Tema 5, 236
Tema 6, 238
- Resta en complemento a dos, 106
- Restador, 105
- Síntesis de circuitos lógicos, 78
Circuitos de dos niveles, 78
- Conjunto completo de puertas lógicas, 79
- Síntesis de Sistemas Secuenciales
Asignación de Estados según el modelo Mealy, 147
Asignación de Estados según el modelo Moore, 141
Comportamiento según el modelo Mealy, 146
Comportamiento según el modelo Moore, 139
Diagrama de estados según el modelo Mealy, 146
Diagrama de estados según el modelo Moore, 140
FF escogido según el modelo Mealy, 148
FF escogido según el modelo Moore, 142
Síntesis según modelo Mealy, 146
Síntesis según modelo Moore, 139
Tabla de Estados según el modelo Mealy, 146
Tabla de Estados según el modelo Moore, 140
- Semisumador, 103
- Sistema de numeración posicional
Coma fija, 38
Conversión entre sistemas de numeración, 37
Precisión finita, 38
Sistema decimal codificado en binario (BCD), 39
- Sistema digital programable, 176
- Sistemas de numeración posicional, 33
Precisión finita, 38
Sistema binario, octal, hexadecimal, 35
Sistema decimal codificado en binario BCD, 39
- Sistemas Secuenciales Síncronos, 134
Análisis y Síntesis, 134
- Soportes, 170
- Sumador completo, 104
- Tiempo en circuitos lógicos, 79
Riesgos, 80
Tiempo de retardo, 80
- Transistor MOSFET, 74
- Tri-estado, puertas, 101

Apéndice A

CÓDIGOS VHDL DE LOS CIRCUITOS COMBINACIONALES DESARROLLADOS EN EL CAPÍTULO 3

Multiplexor 4 a 1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux4to1 IS
PORT ( w0,w1,w2,w3 : IN std_logic;
        s           : IN std_logic_vector (1 DOWNTO 0);
        f           : OUT std_logic_vector (1 DOWNTO 0));
END mux4to1;

ARCHITECTURE codmux4a1 OF mux4to1 IS
BEGIN
    WITH s SELECT
        f <= w0 WHEN "00",
        w1 WHEN "01",
        w2 WHEN "10",
        w3 WHEN OTHERS
END codmux4a1;
```

Demultiplexor 1 a 4

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY demux1to4 IS
PORT ( w0,w1,w2,w3 : OUT std_logic;
        f           : IN std_logic;
        s           : IN std_logic_vector (1 DOWNTO 0));
END demux1to4;

ARCHITECTURE coddemux1a4 OF demux1to4 IS
BEGIN
    w0 <= f WHEN s = "00" ELSE '0',
    w1 <= f WHEN s = "01" ELSE '0',
    w2 <= f WHEN s = "10" ELSE '0',
    w3 <= f WHEN s = "11" ELSE '0';
END coddemux1a4;
```

Sumador de 4 bits

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY fulladder IS
PORT ( x,y      : IN  std_logic_vector(3 DOWNTO 0)    -- Operandos
       ,cin     : IN  std_logic;                      -- Carry in
       ,Suma   : OUT std_logic_vector (3 DOWNTO 0);   -- Resultado
       ,cout   : OUT std_logic);                     -- Carry out
END fulladder;

ARCHITECTURE codfulladder OF fulladder IS
  SIGNAL s_int:      std_logic_vector (4 DOWNTO 0);
BEGIN
  s_int <= ("0"&x) + y + cin;  -- Se agrega un "0" por la izquierda para pasar de 3 a 4 bits
  Suma <= s_int(3 downto 0);
  cout <= s_int(4);           -- Carry out es el bit 4
END codfulladder;

```

Sumador-Restador de 4 bits

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY restsum IS
PORT ( S_R      : IN  std_logic;      -- Señal de Suma (0) o Resta (1)
       ,A,B     : IN  std_logic_vector (3 DOWNTO 0);  -- Operandos
       ,R       : OUT std_logic_vector (3 DOWNTO 0)); -- Resultado
END restsum;

ARCHITECTURE codrestsum OF restsum IS
BEGIN
  PROCESS (S_R, A, B)
  BEGIN
    CASE S_R IS
      WHEN '0'      => F <= A + B;
      WHEN OTHERS   => F <= A - B;
    END CASE;
  END PROCESS;
END codrestsum;

```

Puerta triestado

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY tri_state IS
PORT ( s : OUT std_logic; -- Salida
        e : IN  std_logic; -- Entrada
        c : IN  std_logic); -- Control
END tri_state;

ARCHITECTURE codtri_state OF tri_state IS
BEGIN
    s <= e WHEN c = '1' ELSE 'Z'; -- 'Z' representa la "Alta Impedancia"
END codtri_state;

```

Codificador de 4 bits

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY codif4to2 IS
PORT ( E : IN  std_logic_vector(3 DOWNTO 0);
        S : OUT std_logic_vector(1 DOWNTO 0);
        C : OUT std_logic_vector );
END codif4to2;

ARCHITECTURE codcodif4to2 OF codif4to2 IS
BEGIN
    S <= "11" WHEN E(3) = '1' ELSE
              "10" WHEN E(2) = '1' ELSE
              "01" WHEN E(1) = '1' ELSE
              "00";
    C <= '0' WHEN E = "0000" ELSE '1';
END codcodif4to2;

```

Decodificador de 2 bits

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY decodif2to4 IS
PORT ( w : IN  std_logic_vector(1 DOWNTO 0);
        y : OUT std_logic_vector(0 TO 3));
END decodif2to4;

ARCHITECTURE coddecodif2to4 OF decodif2to4 IS
BEGIN
    WITH s SELECT
        y <= "0111" WHEN "00",
                      "1011" WHEN "01",
                      "1101" WHEN "10",
                      "1111" WHEN OTHERS
END coddecodif2to4;

```


Capítulo B

CÓDIGOS VHDL DE LOS CIRCUITOS SECUENCIALES DESARROLLADOS EN EL CAPÍTULO 4

Biestable Asíncrono

1. Sin Entrada de Control

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY biasinSR IS
PORT ( S,R:  IN std_logic;
        Q, noQ: OUT std_logic);
END biasinSR;

ARCHITECTURE codigo OF biasinSR IS
    SIGNAL B, B1:   std_logic;
BEGIN
    B1  <=  S NOR B;
    B   <=  R NOR B1;
    Q   <=  B;
    noQ <=  B1;
END codigo;
```

2. Con Entrada de Control

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY biasincSR IS
PORT ( S,R,C:  IN std_logic;
        Q, noQ: OUT std_logic);
END biasincSR;

ARCHITECTURE codigo OF biasincSR IS
    SIGNAL B, B1:   std_logic;
BEGIN
    B   <=  (S NAND C) NAND B1;
    B1  <=  (R NAND C) NAND B;
    Q   <=  B;
    noQ <=  B1;
END codigo;
```

Biestable Asíncrono (latch) D

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY biasinD IS
PORT ( D,C:  IN std_logic;
        Q, noQ: OUT std_logic);
END biasinD;

ARCHITECTURE codigo OF biasinD IS
    SIGNAL B, B1: std_logic;
BEGIN
    PROCESS (D, C)
    BEGIN
        IF C='0' THEN
            B      <= B;
            B1    <= B1;
        ELSE
            B      <= D;
            B1    <= NOT D;
        END IF;
    END PROCESS;

    Q      <= B;
    noQ   <= B1;

END codigo;
```

Biestable Síncrono (Flip-Flop) D

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY bisinFFD IS
PORT ( D,CLK: IN std_logic;
        Q, noQ:OUT std_logic);
END bisinFFD;

ARCHITECTURE codigo OF bisinFFD IS
BEGIN
    PROCESS (CLK)
    BEGIN
        IF CLK'EVENT AND CLK='1' THEN -- Si '1' Flanco positivo
            -- Si '0' Flanco neg.
            Q      <= D;
            noQ   <= NOT D;
        END IF;
    END PROCESS;
END codigo;
```

Biestable Síncrono (Flip-Flop) JK

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY bisinFFJK IS
PORT ( PRE, CLR, J, K, CLK: IN std_logic;
        Q, Q1: OUT std_logic);
END bisinFFJK;

ARCHITECTURE codigo OF bisinFFJK IS
    SIGNAL D, B, B1: std_logic;
BEGIN
    PROCESS (PRE, CLR, CLK)
    BEGIN

        IF PRE='1' THEN
            B <='1';
            B1 <='0';
        ELSIF clr='1' THEN
            B <='0';
            B1 <='1';
        IF CLK'EVENT AND CLK='1' THEN -- Si '1' Flanco positivo
                                         -- Si '0' Flanco neg.
            Q <= D;
            Q1 <= NOT D;
        END IF;
    END PROCESS;

    D <= (J AND B1) OR ((NOT K) AND B);
    Q <= B;
    Q1<= B1;

END codigo;

```

Biestable Síncrono (Flip-Flop) T

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY bisinFFT IS
PORT ( T, CLK:    IN      std_logic;
        Q, noQ:   OUT     std_logic);
END bisinFFT;

ARCHITECTURE codigoT OF bisinFFT IS
    SIGNAL B:      std_logic;
BEGIN
    PROCESS (CLK)
    BEGIN
        IF CLK'EVENT AND CLK='1' THEN -- Si '1' Flanco positivo
            -- Si '0' Flanco neg.
            IF T='1' THEN
                B <= NOT B;
            END IF;
        END IF;
    END PROCESS;

    Q      <=  B;
    noQ   <= NOT B;

END codigoT;
```

Registro de Almacenamiento

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY registro IS
PORT ( D, CLK:  IN  std_logic;
        Q:        OUT std_logic);
END registro;

ARCHITECTURE codREG OF registro IS
BEGIN
    PROCESS (CLK)
    BEGIN
        IF CLK'EVENT AND CLK='1' THEN
            Q <= D;
        END IF;
    END PROCESS;
END codREG;
```

Registro de Desplazamiento (Shift Register)

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY registrodesp IS
PORT ( Es, CLK, IN: IN std_logic;
        E:         IN std_logic_vector (3 DOWNTO 0);
        K:         IN std_logic_vector (1 DOWNTO 0);
        S:         OUT std_logic_vector (3 DOWNTO 0));
END registrodesp;

ARCHITECTURE codREGdesp OF registrodesp IS
    SIGNAL B: std_logic_vector (3 DOWNTO 0);
BEGIN
    PROCESS (K, CLK)
    BEGIN
        IF K = "11" THEN B <= E;           -- Carga paralela
        ELSEIF CLK'EVENT AND CLK='1' THEN
            CASE K IS
                WHEN "01" => -- Desplz. Izda.
                    B(0) <= B(1);
                    B(1) <= B(2);
                    B(2) <= B(3);
                    B(3) <= Es;
                WHEN "10" => -- Desplz. Dcha.
                    B(0) <= IN;
                    B(1) <= B(0);
                    B(2) <= B(1);
                    B(3) <= B(2);
                WHEN OTHERS => B <= B;
            END CASE;
        END IF;
    END PROCESS;
    S <= B;
END codREGdesp;

```

K Funcionamiento

- 00 Almacenamiento
- 01 Desp. a la izquierda
- 10 Desp. a la derecha
- 11 Carga paralela

Contadores

```

LIBRARY      ieee;
USE         ieee.std_logic_1164.all;
USE         ieee.std_logic_unsigned.all;

ENTITY contador IS PORT (
    CLK:  IN   std_logic;
    E:    IN   std_logic_vector (3 DOWNTO 0);
    K:    IN   std_logic_vector (1 DOWNTO 0);
    S:    OUT  std_logic_vector (3 DOWNTO 0));
END contador;

ARCHITECTURE codconta OF contador IS
    SIGNAL B:   std_logic_vector (3 DOWNTO 0);
BEGIN
    PROCESS (K, CLK)
    BEGIN
        IF K = "11" THEN
            B <= E;
        ELSEIF CLK'EVENT AND CLK='1' THEN
            CASE K IS
                WHEN "01"      => B <= B + 1;
                WHEN "10"      => B <= B - 1;
                WHEN OTHERS   => B <= B;
            END CASE;
        END IF;
    END IF;
    END PROCESS;

    S <= B;

END codconta;

```

K Funcionamiento

- 00 Almacenamiento
- 01 Ascendente
- 10 Descendente
- 11 Carga paralela

Diseño de Sistemas Secuenciales

Atendiendo al ejercicio propuesto en 4.4.3.1 veamos su desarrollo bajo VHDL

```

LIBRARY ieee;
USE     ieee.std_logic_1164.all;

ENTITY moore IS PORT(
    clk:      IN      std_logic;
    x:        IN      std_logic;
    reset:    IN      std_logic;
    Z:        OUT     std_logic_vector(1 DOWNTO 0));
END moore;

ARCHITECTURE cont2bits OF moore IS

    TYPE estado IS (s0, s1, s2, s3);
    SIGNAL cuenta: estado;

BEGIN
    PROCESS (clk, reset)
    BEGIN
        IF reset = '1' THEN cuenta <= s0;
        ELSIF (clk'EVENT AND clk='1') THEN
            CASE cuenta IS
                WHEN s0 =>
                    IF x = '0' THEN cuenta <= s1;
                    ELSE cuenta <= s3;
                    END IF;
                WHEN s1 =>
                    IF X = '0' THEN cuenta <= s2;
                    ELSE cuenta <= s0;
                    END IF;
                WHEN s2 =>
                    IF x = '0' THEN cuenta <= s3;
                    ELSE cuenta <= s1;
                    END IF;
                WHEN s3 =>
                    IF x = '0' THEN cuenta <= s0;
                    ELSE cuenta <= s2;
                    END IF;
            END CASE;
        END IF;
    END PROCESS;

    PROCESS (cuenta)
    BEGIN
        CASE cuenta IS
            WHEN s0 => z <= "00";
            WHEN s1 => z <= "01";
            WHEN s2 => z <= "10";
            WHEN s3 => z <= "11";
        END CASE;
    END PROCESS;
END cont2bits;
```


Capítulo C

RESOLUCIÓN DE LOS EJERCICIOS

Tema 1

1. ¿En un sistema de numeración posicional cómo representarías los números $N = 2443_{10}$ y $N = A7D6H$?

$$N = 2443_{10} = 2 \cdot 10^3 + 4 \cdot 10^2 + 4 \cdot 10^1 + 3 \cdot 10^0$$

$$N = A7D6H = A \cdot 16^3 + 7 \cdot 16^2 + D \cdot 16^1 + 6 \cdot 16^0$$

2. Convertir al sistema decimal los siguientes números en sistema binario:

a) $11 = 1 \cdot 2^1 + 1 \cdot 2^0 = 2 + 1 = 3$

b) $100 = 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 4$

c) $111 = 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 4 + 2 + 1 = 7$

d) $1000 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 8$

e) $11101 = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 16 + 8 + 4 + 1 = 29$

f) $11,011 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 - 1 + 1 \cdot 2^0 = 16 + 8 + 0 - 1 + 1 = 24$

$0,25 + 0,125 = 3,375$

3. ¿Cuál es el número decimal más alto que se puede expresar con los siguientes número de bits?

a) 2 bit $\rightarrow 2^2 - 1 = 3$

b) 7 bit $\rightarrow 2^7 - 1 = 127$

c) 10 bit $\rightarrow 2^{10} - 1 = 1023$

4. ¿Cuántos bits son necesarios para expresar los siguientes números binarios?

a) $17 \rightarrow 4$ bit $\rightarrow 2^4 - 1 = 15 \rightarrow$ No bastan 4 bit. 5 bit $\rightarrow 2^5 - 1 = 31$
 \rightarrow Con 5 bit.

b) $81 \rightarrow 6$ bit $\rightarrow 2^6 - 1 = 63 \rightarrow$ No bastan 6 bit. 7 bit $\rightarrow 2^7 - 1 = 127$
 \rightarrow Con 7 bit.

c) $35 \rightarrow 5$ bit $\rightarrow 2^5 - 1 = 31 \rightarrow$ No bastan 5 bit. 6 bit $\rightarrow 2^6 - 1 = 63$
 \rightarrow Con 6 bit.

d) $32 \rightarrow 5$ bit $\rightarrow 2^5 - 1 = 31 \rightarrow$ No bastan 5 bit. 6 bit $\rightarrow 2^6 - 1 = 63$
 \rightarrow Con 6 bit.

5. Convertir al sistema decimal:

$$a) E5_{16} = E \cdot 16^1 + 5 \cdot 16^0 = 14 \cdot 16 + 5 = 224 + 5 = 229_{10}$$

$$b) B2F8_{16} = B \cdot 16^3 + 2 \cdot 16^2 + 15 \cdot 16^1 + 8 \cdot 16^0 = 11 \cdot 4096 + 2 \cdot 256 + 15 \cdot 16 + 8 = 45816_{10}$$

6. Convierte al sistema decimal el siguiente número en base ocho:

$$2374_8 = 2 \cdot 8^3 + 3 \cdot 8^2 + 7 \cdot 8^1 + 4 \cdot 8^0 = 2 \cdot 512 + 3 \cdot 64 + 7 \cdot 8 + 4 = 1276_{10}$$

7. Conversión binario-hexadecimal

$$a) 1100101001010111 \rightarrow 1100\ 1010\ 0101\ 0111 \rightarrow \\ 12\ 10\ 5\ 7 \rightarrow C\ A\ 5\ 7 \rightarrow$$

$$1100101001010111_2 = CA57_{16}$$

$$b) 01101001101 \rightarrow 011\ 0100\ 1101 \rightarrow 0011\ 0100\ 1101 \rightarrow \\ 3\ 4\ 13 \rightarrow 3\ 4\ D \rightarrow 01101001101_2 = 34D_{16}$$

8. Conversión hexadecimal-binario

$$a) 10A4_{16} \rightarrow 0001\ 0000\ 1010\ 0100 \rightarrow 1000010100100_2$$

$$b) CF8E_{16} \rightarrow 1100\ 1111\ 1000\ 1110 \rightarrow 1100111110001110_2$$

$$c) 9742_{16} \rightarrow 1001\ 0111\ 0100\ 0010 \rightarrow 1001011101000010_2$$

9. Conversión decimal-hexadecimal

$$a) 650_{10}$$

$$\begin{array}{r} 650 \\ \hline 16 \\ \boxed{10} \quad 40 \\ \hline 16 \\ \boxed{8} \quad 2 \\ \hline 16 \\ \boxed{2} \quad 0 \\ \hline \end{array}$$

N= 2 8 A

$$b) 4025_{10}$$

$$\begin{array}{r} 4025 \\ \hline 16 \\ 82 \quad \boxed{251} \\ \hline 16 \\ 25 \quad \boxed{91} \quad \boxed{15} \\ \hline 16 \\ \boxed{9} \quad \boxed{11} \quad \boxed{15} \quad 0 \\ \hline \end{array}$$

N= F B 9

10. Convierte el número $N = 234,56_{10}$ a binario, estando expresada la parte decimal sobre 5 bits. Parte entera: 234:

$$\begin{array}{ccccccccc}
 234 & & 2 & & & & & & \\
 & 117 & 2 & & & & & & \\
 03 & & & 58 & 2 & & & & \\
 14 & 17 & & & 29 & 2 & & & \\
 \mathbf{0} & \mathbf{1} & 18 & & & & & & \\
 & & & 09 & 14 & 2 & & & \\
 & & & \mathbf{1} & \mathbf{0} & 7 & 2 & & \\
 & & & & & 1 & 3 & 2 & \\
 & & & & & & 1 & 1 & 2 \\
 & & & & & & & 1 & 0 \\
 \text{N} = & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0
 \end{array}$$

Parte decimal 0,56:

$$\begin{aligned}
 0,56 \cdot 2 &= 1,12 \\
 0,12 \cdot 2 &= 0,24 \\
 0,24 \cdot 2 &= 0,48 \\
 0,48 \cdot 2 &= 0,96 \\
 0,96 \cdot 2 &= 1,92 \\
 0,56_{10} &= 0,10001_2
 \end{aligned}$$

$$N = 234,56_{10} = 11101010,10001_2$$

11. Convierte de decimal a binario (Máximo cuatro cifras a la derecha de la coma, si la conversión no es completa, indica el error relativo):
- 177,625 → 10110001,101
 - 78,4375 → 1001110,0111
 - 113,7 → 1110001,1011**0011001**
113,6875 → $E_{abs} = 0,0125$;
 $E_{rel} = 100 \cdot E_{abs} / 113,7 = 0,0109938 < 0,011\%$

12. Conversión de binario a decimal y a hexadecimal y octal:

- 10011100,1001₂ → 156,5625₁₀ → 9C,9₁₆ → 234,44₈
- 110111,001₂ → 55,125₁₀ → 37,216 → 67,18
- 1001001,001₂ → 73,125₁₀ → 49,2₁₆ → 111,1₈

13. Expresa el número $N = -57$ en binario complemento a 2 sobre 8 bits

En valor absoluto 57:

$$\begin{array}{r}
 57 \quad | \quad 2 \\
 17 \quad 28 \quad | \quad 2 \\
 \mathbf{1} \quad 08 \quad 14 \quad | \quad 2 \\
 \mathbf{0} \quad \mathbf{0} \quad 7 \quad | \quad 2 \\
 \mathbf{1} \quad 3 \quad | \quad 2 \\
 \mathbf{1} \quad 1 \quad | \quad 2 \\
 \mathbf{1} \quad 0
 \end{array}$$

$N = \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1$

Complemento a 1 en 8 bit: $00111001 \rightarrow 11000110$ Complemento a 2: $11000110 + 1 = 11000111 \rightarrow N = 11000111$

14. Expresión de números negativos. Escribe con 8 bits el siguiente número decimal en magnitud con signo, complemento a 1, complemento a 2 y exceso a 128:

- a) $-113 \rightarrow S\&M: 11110001 \rightarrow \text{Comp. a1: } 10001110 \rightarrow \text{Comp. a2: } 10001111 \rightarrow \text{Exc.: } 00001111$
 b) $-78 \rightarrow S\&M: 11001110 \rightarrow \text{Comp. a1: } 10110001 \rightarrow \text{Comp. a2: } 10110010 \rightarrow \text{Exc.: } 00110010$

15. ¿Qué número decimal representan los siguientes números binarios en cada sistema? Rellena la tabla y utiliza los valores para comprobar si el resultados de las sumas propuestas puede ser correcto en ocho bit (para cada sistema existen unos límites en los resultados posibles, superarlos produce desbordamiento: resultado en 9 bit o signo opuesto).

	Binario natural	Magnitud con signo	Compl. a 1	Comp. a 2	Exceso a 128
A	01001010	+74	+74	+74	-54
B	00101010	+42	+42	+42	-86
C	01001100	+76	+76	+76	-52
D	01010100	+84	+84	+84	-44
E	10100010	+162	-34	-93	+34
F	11101110	+238	-110	-17	+110
G	11000001	+193	-65	-62	+65
H	10111001	+185	-57	-70	+57

- a) Binario natural: $0 \leq N \leq 2^n - 1 \rightarrow n = 8$ bit $\rightarrow [0, 255]$

- 1) A+B: $74+42=113$
- 2) C+D: $76+84=160$
- 3) E+F: $162+238=400 \rightarrow$ desbordamiento
- 4) G+H: $193+185=378 \rightarrow$ desbordamiento

- b) Complemento a 2: $-2^{n-1} \leq N \leq 2^{n-1} - 1 \rightarrow n = 8 \text{ bit} \rightarrow [-128, 127]$
- 1) C+D: $76 + 84 = 160$ La suma binaria es negativa \rightarrow desbordamiento
 - 2) E+F: $-94 + -18 = -112$
 - 3) G+H: $(-63) + (-71) = (-134)$ La suma binaria es positiva \rightarrow desbordamiento
 - 4) B+G: $42 + (-63) = -21$
16. Números decimales codificados en binario: Convertir de BCD a decimal y a binario natural.
- a) $001001010111_{BCD} \rightarrow 257_{10} \rightarrow 100000001_2$
 - b) $011000111000_{BCD} \rightarrow 638_{10} \rightarrow 100111110_2$
17. Coma flotante: IEEE Std. 754: Convierte los siguientes números de hexadecimal a decimal (están escritos en coma flotante y precisión sencilla según el estándar IEEE 754):
- a) $42E48000H \rightarrow 01000010111001001000000000000000_2 \rightarrow +1,78515625 \cdot 2^6 = +114,25$
 - b) $3F880000H \rightarrow 00111111000100000000000000000000_2 \rightarrow +1,0625 \cdot 2^0 = +1,0625$
 - c) $00800000H \rightarrow 00000001000000000000000000000000_2 \rightarrow +1,0 \cdot 2^{-126} = +1,17549435 \cdot 10^{-38}$
 - d) $C7F00000H \rightarrow 11000111111000000000000000000000_2 \rightarrow -1,875 \cdot 2^{16} = -122,880$

Tema 2

1. Simplifica todo lo posible las siguientes expresiones:
 - a) $Z = \bar{A} \cdot B \cdot C + \bar{A} = \bar{A} \cdot (B \cdot C + 1) = \bar{A}$
 - b) $Z = A + A \cdot B = A$
 - c) $Z = A \cdot (B + C \cdot (B + A)) = A \cdot (B + C)$
 - d) $Z = \bar{A} \cdot \bar{B} + A \cdot B + A \cdot \bar{B} = A \cdot \bar{B}$
2. Comprueba las siguientes afirmaciones respecto a la suma exclusiva:
 - a) $X \oplus \bar{X} = 1 \Rightarrow X \oplus \bar{X} = X \cdot \bar{X} + \bar{X} \cdot X = X + \bar{X} = 1$
 - b) $X \oplus 0 = X \Rightarrow X \oplus 0 = X \cdot \bar{0} + \bar{X} \cdot 0 = X \cdot 1 + 0 = X$
 - c) $X \oplus 1 = \bar{X} \Rightarrow X \oplus 1 = X \cdot \bar{1} + \bar{X} \cdot 1 = \bar{X}$
 - d) $X \oplus X = 0 \Rightarrow X \oplus X = X \cdot \bar{X} + \bar{X} \cdot X = 0 + 0 = 0$
3. Escribe la tabla de la verdad de las siguientes funciones lógicas:

a) $F = A \cdot B + C$				b) $F = \bar{A} \cdot C + B$				c) $F = A \cdot B + C \cdot (A \oplus B)$			
A	B	C	F	A	B	C	F	A	B	C	F
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	1	0	0	1	0
0	1	0	0	0	0	1	0	0	1	0	0
0	1	1	1	0	1	1	1	0	1	1	1
1	0	0	0	1	0	0	0	1	0	0	0
1	0	1	1	1	0	1	0	1	0	1	1
1	1	0	1	1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1

4. Define por su expresión canónica las siguientes funciones:

Los 3 ejercicios siguientes se plantearán por medio de la Tabla de Verdad (TV) y se plasmarán los minterm. Se muestra la solución y se deja al alumno/a el planteamiento en la TV.

 - a) Función F que vale 1 si el número de 1s que aparecen en sus tres variables es mayor que el número de 0s.

$$F(A, B, C) = \sum m(3, 5, 6, 7)$$
 - b) Función lógica F de cuatro variables, que vale 1 cuando el número introducido en sus variables pertenezca al código BCD de 4 bits.

$$F(A, B, C, D) = \sum m(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)$$
 - c) Función F de cuatro variables A_3, A_2, A_1, A_0 que valga 0 sólo si el valor expresado por $A_3A_2A_1A_0$ es mayor que 10 o menor que 5.

$$F(A_3, A_2, A_1, A_0) = \sum m(5, 6, 7, 8, 9, 10)$$
5. Repite el ejercicio 4, obteniendo la expresión mínima para las tres funciones.
 Se resuelven varios ejercicios.

a) $F(A, B, C) = A \cdot C + A \cdot B + B \cdot C$

		BC			
		00	01	11	10
A	0	0	0	1	0
	1	0	1	1	1

b) $F(A, B, C, D) = \overline{A} + \overline{B} \cdot \overline{C}$

c) $F(A_3, A_2, A_1, A_0) = (A_3 + A_2) \cdot (\overline{A}_3 + \overline{A}_2) \cdot (A_3 + A_1 + A_0) \cdot (\overline{A}_3 + \overline{A}_1 + \overline{A}_0)$

		A ₁ A ₀			
		00	01	11	10
A ₃	A ₂	00	0	0	0
		01	0	1	1
11		0	0	0	0
10		1	1	0	1

6. Para las siguientes funciones, define la expresión mínima (suma o producto):

a) $F(A, B, C, D) = \sum m(1, 4, 5, 6, 7, 14, 15)$
 $F(A, B, C, D) = (B + D) \cdot (B + \overline{C}) \cdot (\overline{A} + C)$

b) $F(A, B, C, D) = \sum m(0, 1, 2, 4, 5, 6, 8, 10, 12)$
 $F(A, B, C, D) = (\overline{A} + \overline{D}) \cdot (\overline{C} + \overline{D}) \cdot (\overline{A} + \overline{B} + \overline{C})$

c) $F(A, B, C, D) = \sum m(0, 3, 5, 7, 8, 9, 10, 12, 13) + \Sigma d(1, 6, 11, 14)$
 $F(A, B, C, D) = A \cdot D + \overline{B} \cdot \overline{C} + A \cdot \overline{D} + \overline{C} \cdot D$

d) $F(A, B, C, D) = \sum m(1, 3, 4, 5, 11, 12, 13)$
 $F(A, B, C, D) = (\overline{B} + \overline{C}) \cdot (B + D) \cdot (\overline{A} + B + C)$

e) $F(A, B, C, D) = \sum m(0, 2, 4, 6, 7, 8, 10, 13, 15)$
 $F(A, B, C, D) = (\overline{B} + \overline{C}) \cdot (B + D) \cdot (\overline{A} + B + C)$

f) $F(A, B, C, D) = \sum m(2, 4, 8, 10, 11, 12)$
 $F(A, B, C, D) = (C + \overline{D}) \cdot (A + \overline{D}) \cdot (\overline{B} + \overline{C}) \cdot (A + B + C)$

7. Función lógica $Z_{A < B}$ que vale uno sólo si el número binario de dos bits $A(A_1, A_0)$ es menor que el número binario de dos bits $B(B_1, B_0)$, y función lógica $Z_{A > B}$ que vale uno sólo si el número binario de dos bits A es mayor que el número binario de dos bits B . Obtén su expresión mínima.

A_1	A_0	B_1	B_0	$Z_{A < B}$	$Z_{A > B}$
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	0	0
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	1	0	1
1	0	1	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	0	1
1	1	1	0	0	1
1	1	1	1	0	0

Plantea dos mapas de Karnaugh, uno por función, y simplifica.

a) $Z_{A < B}(A_1, A_0, B_1, B_0) = \bar{A}_1 \cdot B_1 + \bar{A}_1 \cdot \bar{A}_0 \cdot B_0 + \bar{A}_0 \cdot B_1 \cdot B_0$

b) $Z_{A > B}(A_1, A_0, B_1, B_0) = A_1 \cdot \bar{B}_1 + A_0 \cdot \bar{B}_1 \cdot \bar{B}_0 + A_1 \cdot A_0 \cdot \bar{B}_0$

8. Define una función Z que vale 1 sólo si el número en complemento a dos expresado por las cuatro variables de entrada E_3, E_2, E_1, E_0 , está entre -5 y 5, incluidos el 0 y ambos números. Obtén su expresión mínima.

Plantea la TV, llévalo a un mapa de Karnaugh y simplifica por minterms.

$$Z(E_3, E_2, E_1, E_0) = \bar{E}_3 \cdot \bar{E}_2 + E_3 \cdot E_2 + E_3 \cdot \bar{E}_2 \cdot E_1 + \bar{E}_3 \cdot E_1$$

9. Para las siguientes funciones, define la expresión mínima (suma o producto):

Lleva a un mapa de Karnaugh los minterms indicados y simplifica por minterms o Maxterms.

a) $F(A, B, C, D) = \sum m(0, 1, 2, 3, 7, 10, 14, 15)$
 $F(A, B, C, D)_m = \bar{A} \cdot \bar{B} + A \cdot C \cdot \bar{D} + B \cdot C \cdot D$

b) $F(A, B, C, D) = \sum m(0, 1, 5, 7, 8, 10, 14, 15)$
 $F(A, B, C, D)_m = \bar{A} \cdot B \cdot \bar{C} + \bar{A} \cdot B \cdot D + A \cdot B \cdot C + A \cdot \bar{B} \cdot \bar{D}$
 $F(A, B, C, D)_M = (A + B + \bar{C}) \cdot (A + \bar{B} + D) \cdot (\bar{A} + \bar{B} + C) \cdot (\bar{A} + B + \bar{D})$

c) $F(A, B, C, D) = \sum m(0, 2, 6, 8, 10, 15) + \sum d(4, 9, 12, 13)$
 $F(A, B, C, D)_m = \bar{B} \cdot \bar{D} + A \cdot \bar{D} + A \cdot B \cdot D$
 $F(A, B, C, D)_M = (A + \bar{D}) \cdot (B + \bar{D}) \cdot (\bar{A} + \bar{B} + D)$

10. Utilizando puertas AND, OR y NOT, diseña un sencillo sistema de alarma (A) para el cinturón (C) de seguridad del coche, el cual detecta cuándo el interruptor (I) de arranque se ha activado y el cinturón de seguridad no está abrochado.

C	I	A
0	0	0
0	1	1
1	0	0
1	1	0

$$A = \bar{C} \cdot I$$

11. Utilizando puertas AND, OR y NOT, diseña un sencillo sistema de detección de intrusos en una habitación de una casa. Para ello tenemos sensores en dos ventanas (V_1 y V_2) y en la puerta (P). Estos sensores producen una salida de nivel alto (A) cuando se abre la puerta (o ventana).

Consideramos que puerta o ventanas abiertas es 1. Resolvemos por Maxterm

V_1	V_2	P	A
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$$A = V_1 + V_2 + P$$

12. Obtén la forma canónica de las siguientes expresiones:

- a) $f_1 = \bar{A} \cdot \bar{C} + A \cdot \bar{C} \cdot D + \bar{A} \cdot B \cdot C \cdot D$
- b) $f_2 = \bar{A} \cdot \bar{C} \cdot \bar{D} + A \cdot C + A \cdot B \cdot C$
- c) $f_3 = \bar{A} \cdot B + A \cdot C$
- d) $f_4 = (\bar{B} + D) \cdot (\bar{B} + A) \cdot (C + D) \cdot (D + A)$
- e) $f_5 = (X_1 + X_3) \cdot (\bar{X}_3 + \bar{X}_2 \cdot X_4) + (X_3 \cdot X_5 + (\bar{X}_1 + X_4) \cdot (X_2 + \bar{X}_3))$

13. Representa y simplifica las siguientes funciones mediante mapas de Karnaugh:

Se Resuelven varios ejercicios.

a) $f_1 = (1, 3, 9, 10, 12, 13, 14, 15)m$

		CD				
		00	01	11	10	
A	B	00	0	1	1	0
		01	0	0	0	0
11		1	1	1	1	
10		0	1	0	0	

1) $f_1 = A \cdot B + A \cdot \bar{C} \cdot D + \bar{A} \cdot \bar{B} \cdot D$

b) $f_2 = (0, 2, 3, 4, 5, 7, 8, 10, 11)m$

c) $f_3 = (1, 6, 7)m$

d) $f_4 = ((0, 1, 6)M$

		BC				
		00	01	11	10	
A	B	00	0	0		
		01				0
1						0

1) $f_4 = (A + B) \cdot (\bar{A} + \bar{B} + C)$

e) $f_5 = (0, 1, 2, 3, 7, 8, 9, 11, 15)m + K(6, 12)$

f) $f_6 = (3, 6, 7, 8, 10)m + K(12, 13, 14)$

g) $f_7 = (0, 1, 4, 5, 7, 8, 10, 15)m + K(2, 6, 14)$

		CD				
		00	01	11	10	
A	B	00	1	1	0	X
		01	1	1	1	X
11		0	0	1	X	
10		1	0	0	1	

1) $f_7 = \bar{A} \cdot \bar{C} + B \cdot C + \bar{B} \cdot \bar{D}$

h) $f_8 = (0, 1, 4, 5, 6, 7, 12, 13, 14, 16, 17, 28, 29)m + K(10, 11, 22, 23, 25, 26, 30, 31)$

14. Implementar las expresiones siguientes mediante lógica NAND de 2 entradas:

Se Resuelven varios ejercicios.

a) $f_1 = A \cdot B \cdot C = \overline{\overline{A} \cdot \overline{B} \cdot C}$

b) $f_2 = A \cdot B \cdot C + D \cdot E = \overline{\overline{A} \cdot \overline{B} \cdot C} + \overline{\overline{D} \cdot \overline{E}} = \overline{(\overline{A} \cdot \overline{B} \cdot C)} \cdot \overline{(\overline{D} \cdot \overline{E})}$

c) $f_3 = A \cdot B \cdot C + \overline{D} + \overline{E}$

d) $f_4 = (\overline{A} \cdot \overline{B}) + (\overline{C} \cdot \overline{D})$

e) $f_5 = (A + B)(C + D) = \overline{\overline{(A + B)} \cdot \overline{(C + D)}} = \overline{(\overline{A} \cdot \overline{B})} \cdot \overline{(\overline{C} \cdot \overline{D})}$

f) $f_6 = A \cdot B \cdot (C \cdot (\overline{D} \cdot \overline{E}) + (\overline{A} \cdot \overline{B})) + (\overline{B} \cdot C \cdot \overline{E})$

$$g) f_7 = B \cdot (C \cdot \overline{D} \cdot E + \overline{E} \cdot F \cdot G) \cdot ((\overline{A} \cdot \overline{B}) + C)$$

15. Los semáforos de un cruce están gobernados por las condiciones:

- a) Si hay un coche por la calle A, el semáforo de esa calle S_A está en verde.
- b) Si hay un coche por la calle B, el semáforo de esa calle S_B está en verde si no hay coches en A.
- c) Si hay un coche por la calle C, el semáforo de esa calle S_C está en verde si no hay coches ni en A ni en B.
- d) En ausencia de coches todos están en rojo.

A través de la tabla de la verdad, expresa la función lógica correspondiente (rojo=1). Resolvemos por Maxterms

A	B	C	S_A	S_B	S_C
0	0	0	1	1	1
0	0	1	1	1	0
0	1	0	1	0	1
0	1	1	1	0	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	0	1	1

S_A		S_B		S_C	
		C	C	C	
AB	0 1	AB	0 1	AB	0 1
00				00	
01			0 0	01	
11	0 0			11	
10	0 0	10		10	

$$S_A = \overline{A}$$

$$S_B = A + \overline{B}$$

$$S_B = A + B + \overline{C}$$

16. Diseña un circuito lógico con cuatro variables de entrada que sólo genera un 1 en la salida cuando tres variables de entrada son 1.

A	B	C	D	S	S	CD
					AB	00 01 11 10
0	1	1	1	1	00	0 0 0 0
1	0	1	1	1	01	0 0 1 0
1	1	0	1	1	11	0 1 0 1
1	1	1	0	1	10	0 0 1 0
X	X	X	X	0		

minterms: $A = (A \oplus B) \cdot C \cdot D + A \cdot B \cdot (C \oplus D)$

17. En una empresa hay 4 socios que tienen repartidas las acciones de la forma siguiente:

$$A = 35\% \quad B = 30\% \quad C = 25\% \quad D = 10\%$$

Describe una función lógica que indique, a la hora de votar una propuesta, si saldrá ésta adelante (S) o no.

A	B	C	D	S
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

minterms: $S = A \cdot B + B \cdot C + A \cdot C$

Maxterms: $S = (A + B) \cdot (B + C) \cdot (A + C)$

18. En una determinada planta de procesamiento químico, se usan tres diferentes elementos químicos líquidos en el proceso de fabricación. Los tres elementos químicos se almacenan en tres tanques diferentes (A , B y C). Un sensor de nivel en cada tanque (N_A , N_B y N_C) genera una tensión a nivel alto cuando el nivel de líquido del tanque cae por debajo de un punto especificado.

Diseña un circuito para monitorizar el nivel del elemento químico en cada tanque, que indique cuándo el nivel de dos de los tanques cae por debajo del punto especificado.

N_A	N_B	N_C	A
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$A = A \cdot B + B \cdot C + A \cdot C$

19. Un circuito lógico tiene 5 entradas y 1 salida. 4 entradas representan un dígito decimal y la quinta es de control. Cuando ésta, está en 0 lógico, la salida será 0 lógico, si el número decimal es par., y 1 si es impar. Cuando el control está a 1, la salida será 0 cuando la entrada sea múltiplo de 3. Diseña el circuito.

Se divide en dos la TV por comodidad.

c	n_3	n_2	n_1	n_0	S	c	n_3	n_2	n_1	n_0	S
0	0	0	0	0	1	1	0	0	0	0	1
	0	0	0	1	1		0	0	0	1	1
	0	0	1	0	0		0	0	1	0	1
	0	0	1	1	1		0	0	1	1	0
	0	1	0	0	0		0	1	0	0	1
	0	1	0	1	1		0	1	0	1	1
	0	1	1	0	0		0	1	1	0	0
	0	1	1	1	1		0	1	1	1	1
	1	0	0	0	0		1	0	0	0	1
	1	0	0	1	1		1	0	0	1	0
x					x	x					x

Se plantea una simplificación de 5 variables por mapas de Karnaugh. Son necesarios 2 mapas de 4 variables cada uno y cada mapa representa el estado de la variable de mayor peso (S) cuando esta vale 0 (mapa izquierdo) y 1 (mapa derecho).

S				
		n_1n_0		
n_3n_2		00	01	11
00		1	1	1
01			1	1
11	X	X	X	X
10			1	X

$C = 0$

		n_1n_0		
n_3n_2		00	01	11
00		1	1	
01		1	1	1
11	X	X	X	X
10		1		X

$C = 1$

$$S = \bar{n}_3\bar{n}_2\bar{n}_1 + \bar{c}n_0 + \bar{c}n_2n_1 + n_2n_0 + c\bar{n}_1\bar{n}_0 + c\bar{n}_2\bar{n}_3$$

20. Diseña un circuito combinacional que gobierne el sistema de climatización de un recinto de acuerdo con las siguientes especificaciones:

- a) Cuando la temperatura interior sea superior a 25°C ($T_i > 25^\circ\text{C} \rightarrow T_{i>25}$), se debe encender la refrigeración (R) y cuando se inferior a 15°C ($T_i < 15^\circ\text{C} \rightarrow T_{i<15}$), se debe poner en marcha la calefacción (C).
- b) Si la temperatura exterior es inferior a la interior en el primer caso del apartado a), es decir, en el caso de refrigeración ($T_e < T_{i>25} \rightarrow T_{e < T_{i>25}}$), o superior a la interior en el segundo caso (calefacción) ($T_e > T_{i<15} \rightarrow T_{e > T_{i<15}}$), además del sistema correspondiente (refrigeración/calefacción), se deben poner en marcha los ventiladores (V) de entrada de aire exterior.
- c) En días festivos (F), el sistema debe estar parado en cualquier condición. Implementar el circuito mediante puertas NAND de 2 y 3 entradas.

F	$T_{i>25}$	$T_{i<15}$	$T_{e < T_{i>25}}$	$T_{e > T_{i<15}}$	R	C	V
	A	B	C	D			
0	0	0	0	0	0	0	0
	0	0	0	1	0	0	0
	0	0	1	0	0	0	0
	0	0	1	1	0	0	0
	0	1	0	0	0	1	0
	0	1	0	1	0	1	1
	0	1	1	0	0	0	0
	0	1	1	1	0	1	1
	1	0	0	0	1	0	0
	1	0	0	1	1	0	0
	1	0	1	0	1	0	1
	1	0	1	1	1	0	1
	-	-	-	-	X	X	X

Realizando 3 mapas de Karnaugh de 4 variables (no se contempla la variable F), obtendremos las tres funciones relativas a R , C y V .

Una vez obtenidas las 3 funciones (f_R , f_C y f_V), mediante el operador AND, las salidas del sistema estarán activas si no es Festivo, es decir, \overline{F} .

$$R = \overline{F} \cdot f_R = \overline{\overline{F} \cdot f_R}$$

$$C = \overline{F} \cdot f_C = \overline{\overline{F} \cdot f_C}$$

$$V = \overline{F} \cdot f_V = \overline{\overline{F} \cdot f_V}$$

Se resuelven las funciones f_R , f_C y f_V :

		f_R			
		CD			
AB		00	01	11	10
00		0	0	0	0
01		0	0	0	0
11		X	X	X	X
10		1	1	1	1

$$f_R = A \rightarrow R = \overline{F} \cdot A = \overline{\overline{F} \cdot A}$$

		f_C			
		CD			
AB		00	01	11	10
00		0	0	0	0
01		1	1	1	0
11		X	X	X	X
10		0	0	0	0

$$f_C = B \cdot (\overline{C} + D) \rightarrow C = \overline{F} \cdot B \cdot (\overline{C} + D) = \overline{\overline{F} \cdot B \cdot (\overline{C} + D)} = \overline{\overline{F} \cdot B \cdot (\overline{C} \cdot \overline{D})}$$

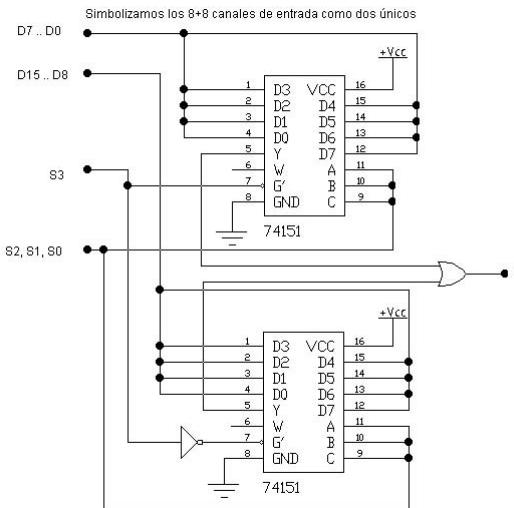
		f_V			
		CD			
AB		00	01	11	10
00		0	0	0	0
01		0	1	1	0
11		X	X	X	X
10		0	0	1	1

$$f_V = BD + AC \rightarrow V = \overline{\overline{\overline{F} \cdot (BD + AC)}} = \overline{\overline{F} \cdot \overline{(BD + AC)}}$$

Tema 3

1. Ejercicio 1: Utiliza multiplexores 74151 y cualquier otra lógica necesaria para multiplexar 16 líneas de datos en una única línea de salida.

Un mux. 74151 presenta 8E/1S y 3 bits de Selección (8 a 1).



2. Ejercicio 2: Utilizando un MUX 4 a 1, implementa

$$f(a, b, c) = ab + ac.$$

Acudiendo al mapa de Karnaugh, la función canónica es:

$$f(a, b, c) = abc + \bar{a}bc + ab\bar{c}.$$

Sobre el mapa de Karnaugh, de la Figura 1, se muestra cuales han sido las simplificaciones de los minter para obtener la función a implementar.

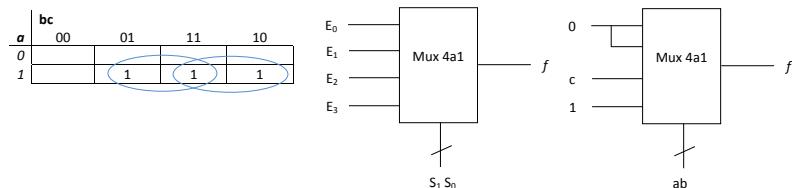


FIGURA 1. Mapa de Karnaugh - Mux 4a1 genérico - Función implementada

Sabiendo que la función f del Mux 4a1 se expresa de la siguiente forma:

$$f = E_0 \bar{S}_1 \bar{S}_0 + E_1 \bar{S}_1 S_0 + E_2 S_1 \bar{S}_0 + E_3 S_1 S_0,$$

sobre la Tabla 1 siguiente se muestra la selección de variables para la implementación, escogiendo a, b como $S_1 S_0$ y c entrada al Mux.:

CUADRO 1. Mux 4a1

a	b	c	f	Valor E_i
0	0	0	0	$E_0 = 0$
0	0	1	0	
0	1	0	0	$E_1 = 0$
0	1	1	0	
1	0	0	0	$E_2 = c$
1	0	1	1	
1	1	0	1	$E_3 = 1$
1	1	1	1	

Finalmente, la Figura 1 muestra la conexión de un Mux 4a1 para implementar la función propuesta.

3. Ejercicio 3: Utilizando un MUX de 8 a 1, implementa la función:

$$f = \Sigma m(1, 2, 5, 6, 7, 8, 10, 12, 13, 15).$$

Teniendo en cuenta que es una función de 4 variables ($A_3 A_2 A_1 A_0$), se necesitará un Mux 8a1, dedicando 3 variables a la selección y la cuarta variable como posible entrada al Mux. La Tabla 2 muestra las diferentes agrupaciones y la Figura 2 muestra las conexiones a realizar en el Mux 8a1 para realizar la función propuesta:

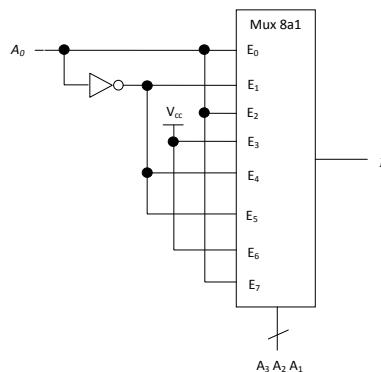


FIGURA 2. Función implementada con Mux 8a1

CUADRO 2. Mux 8a1

A_3	A_2	A_1	A_0	f	Valor E_i
0	0	0	0	0	$E_0 = A_0$
0	0	0	1	1	
0	0	1	0	1	$E_1 = \bar{A}_0$
0	0	1	1	0	
0	1	0	0	0	$E_2 = A_0$
0	1	0	1	1	
0	1	1	0	1	$E_3 = 1$
0	1	1	1	1	
1	0	0	0	1	$E_4 = \bar{A}_0$
1	0	0	1	0	
1	0	1	0	1	$E_5 = \bar{A}_0$
1	0	1	1	0	
1	1	0	0	0	$E_6 = 1$
1	1	0	1	1	
1	1	1	0	1	$E_7 = A_0$
1	1	1	1	1	

4. Ejercicio 4: Utilizando el decodificador 74154, implementa un circuito para decodificar un número de 5 bits. Determina la salida que se activa al introducir el código binario de entrada 10101.

Hay que encontrar la función f que se active a 1 cuando se dé la combinación de entradas 10101. Elaborando la tabla de verdad, según se muestra en la Tabla 3, pasamos a realizar las conexiones adecuadas en el dispositivo 74154 decodificador 4a16 que se muestran en la Figura 3.

CUADRO 3. Decodificador 4a16

D_4	D_3	D_2	D_1	D_0	f
0	0	0	0	0	0
...					
...					
0	1	1	1	1	0
1	0	0	0	0	0
...					
1	0	1	0	1	1 (5)
...					
1	1	1	1	1	0

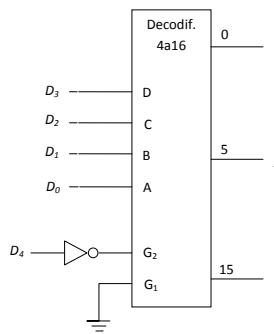


FIGURA 3. Función implementada con Decodificador 4a16

5. Ejercicio 5: Diseña un decodificador de 2 a 4, con entrada de activación E , que tenga salidas y entradas activadas a nivel alto. Utilizando el decodificador diseñado, decodifica de 4 a 16.

La Tabla 4 muestra la tabla de verdad de un decodificador 2 a 4 con salidas activas en alto:

CUADRO 4. Tabla de verdad de un Decodificador 2 a 4

E	B	A	D ₀	D ₁	D ₂	D ₃
			0	0	0	0
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

La Figura 4 muestra el esquema de bloques del Decodif. 2a4 que vamos a utilizar en el diseño del Decodif. 4 a 16 y la Figura 5 muestra el esquema de bloques del Decodif. 4a16.

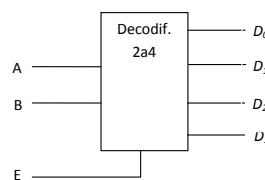


FIGURA 4. Bloque decodificador 2 a 4

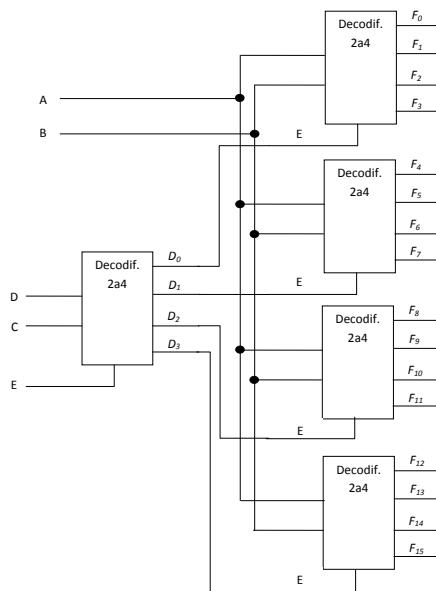


FIGURA 5. Bloque decodificador 4 a 16

6. Ejercicio 6: Utiliza el 74151 para implementar la función:

$$f = \Sigma m(0, 2, 4, 6).$$

La Tabla 5 muestra la función activa en los minterm requeridos y la Figura 6 muestra las conexiones en el CI 74151.

CUADRO 5. Tabla de verdad de la función requerida

S_2	S_1	S_0	f
0	0	0	1 $D_0 = 1$
0	0	1	0 $D_1 = 0$
0	1	0	1 $D_2 = 1$
0	1	1	0 $D_3 = 0$
1	0	0	1 $D_4 = 1$
1	0	1	0 $D_5 = 0$
1	1	0	1 $D_6 = 1$
1	1	1	0 $D_7 = 0$

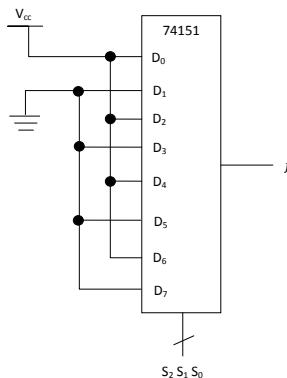


FIGURA 6. Conexiones del CI 74151 para la función $f = \Sigma m(0,2,4,6)$.

7. Ejercicio 7: Utiliza el 74151 para implementar la función:

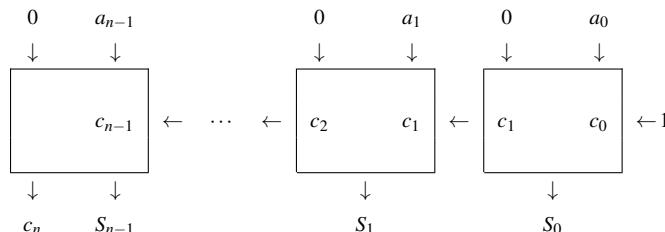
$$f = \Sigma m(0,1,2,3,4,9,13,14,15).$$

Sigue el mismo procedimiento que el empleado en el ejercicio 3.

8. Ejercicio 8: Diseña un circuito que sume uno al número que tengamos.

El nº será $A = a_{n-1}a_{n-2} \cdots a_1a_0$.

Escogemos los bloques full-adder y realizamos el montaje de bloques, introduciendo “1” en el c_0 del primer bloque. Un operando será el nº A y el otro operando será “0” en todos sus bits.



9. Ejercicio 9: Diseña un restador ($A - B$) en complemento a 2. El resultado final debe representar el valor absoluto y su signo, este se reflejará sobre un led. Las variables A y B son de 4 bits y positivas.

La primera consideración es que al ser números de 4 bits, y trabajar en complemento a 2, es necesario añadir un quinto bit (bit de signo) para poder realizar el complemento a 2 correctamente. Veamos un ejemplo:

Sean A y B las variables a operar, $R = A - B$, si $A = 0111$, es decir $A = 7$, y $B = 1111$, es decir, $B = 15$, el complemento a 1 de B es 0000 y el complemento a 2 es 0001, por lo que $R = A - B = 0111 + 0001 = 1000$, es decir, $R = 8$, cuando debería ser $R = -8$.

Debemos añadir un bit más para la operación (bit de signo), tendremos $A = 00111$ y $B = 01111$, por lo que el complemento a 2 de B , será 10001. Realizando la operación de suma, tenemos que $R = 00111 + 10001 = 11000$. como el bit de signo es 1, se trata de un número negativo, por lo que realizando el complemento a 2 tendrímos el valor absoluto e indicaremos que es negativo: $11000 \rightarrow 01000$, por lo tanto $R = -8$.

Una vez obtenido el valor de R , hay que determinar el signo con el fin de mostrar el valor absoluto de la operación. Si es negativo (*Signo* = 1) hay que volver a realizar el complemento a 2. El resultado final se obtendrá a través de un multiplexor 2a1 de 4 bits por variable de entrada, siendo la señal de control el signo.

El esquema viene reflejado en la Figura 7:

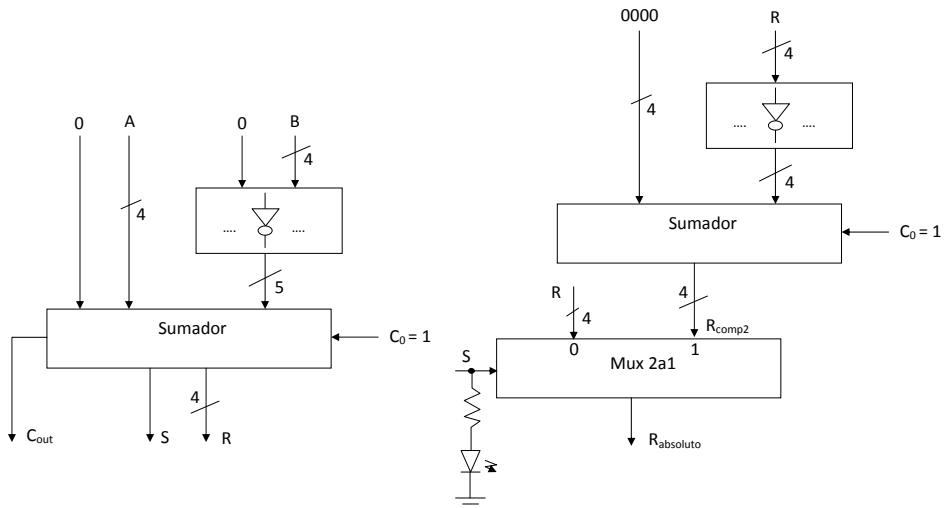


FIGURA 7. Restador del ejercicio 9

10. Ejercicio 10: Diseña un circuito que multiplique un número de 4 bits por dos, tres y cinco.

Las operaciones a realizar son las siguientes: $N = A \times 2$, $N = A \times 3$ y $N = A \times 5$.

$N = A \times 2$

$$N = (a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \dots + a_12^1a_02^0) \times 2 = a_{n-1}2^n + a_{n-2}2^{n-1} + \dots + a_12^2 + a_02^1$$

$N = A \times 3$

$$N = (a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \dots + a_12^1a_02^0) \times (2^1 + 2^0) = (a_{n-1}2^n + a_{n-2}2^{n-1} + \dots + a_12^2a_02^1) + (a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \dots + a_12^1 + a_02^0) =$$

$$a_{n-1}2^n + (a_{n-1} + a_{n-2})2^{n-1} + \dots + (a_1 + a_2)2^2 + (a_0 + a_1)2^1 + a_02^0$$

$N = A \times 5$: Sigue el mismo procedimiento.

11. Ejercicio 11: Realiza un sumador binario natural a BCD.

El planteamiento es realizar la suma en binario natural y posteriormente realizar la transformación a BCD.

Sean las variables A y B , la suma será $S = A + B$. El menor valor de la suma es $S = 0 + 0 = 0$ y el mayor $S = 9 + 9 = 18$, es decir 10010, ya que la suma se realiza en binario natural.

Si $S = A + B \leq 9$, no hay que realizar ninguna transformación del resultado, pero si $S = A + B \geq 10$, hay que transformar el resultado. Veamos los dos casos extremos para cuando $S \geq 10$:

$$\begin{array}{rcl} 10 = & 01010 & \rightarrow 1 \quad 0000 : \text{Se observa que sumando 6 (110) a 10 (01010)} \\ & +110 & \text{se obtiene el equivalente BCD} \\ \hline & 10000 & \end{array}$$

$$\begin{array}{rcl} 18 = & 10010 & \rightarrow 1 \quad 1000 : \text{Se observa que sumando 6 (110) a 18 (010010)} \\ & +110 & \text{se obtiene el equivalente BCD} \\ \hline & 11000 & \end{array}$$

Por lo tanto:

- Si $S < 10 \Rightarrow$ Resultado sin transformar
- Si $S \geq 10 \Rightarrow$ Resultado sin transformado (+6)

Hay que detectar qué números son ≥ 10 . Para ello podemos reflejar sobre un mapa de Karnaugh, de 5 variables, todos los posibles resultados que sean ≥ 10 , como minterms. El resultado de la suma será $S = S_3S_2S_1S_0$ y un bit de carry C . La función obtenida la llamaremos $F_{\geq 10}$.

		S_1S_0						S_1S_0			
		00	01	11	10			00	01	11	10
S_3S_2	00					S_3S_2	00	1	1	X	1
	01						01	X	X	X	X
	11	1	1	1	1		11	X	X	X	X
	10			1	1		10	X	X	X	X
		$C = 0$						$C = 1$			

Por lo tanto, después de realizar las agrupaciones de simplificación, tendremos que:

$F_{\geq 10} = C + S_3S_2 + S_3S_1 = C + S_3(S_2 + S_1)$, es decir, si $S \geq 10$ entonces: $F_{\geq 10} = 1$ si no $F_{\geq 10} = 0$.

- Si $F_{\geq 10} = 1 \Rightarrow +6 \Rightarrow +0110 \Rightarrow 0F_{\geq 10}F_{\geq 10}0$
- Si $F_{\geq 10} = 0 \Rightarrow +0 \Rightarrow +0000 \Rightarrow 0F_{\geq 10}F_{\geq 10}0$

De esta forma, al resultado del sumador, se le sumará 0 $F_{\geq 10} F_{\geq 10} 0$. La Figura 8 muestra el esquema de bloques del sumador binario natural a BCD.

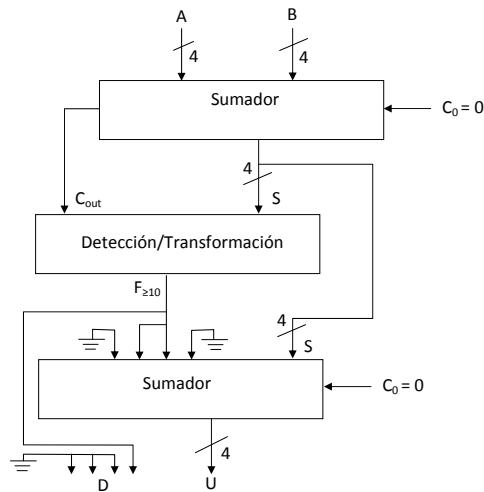


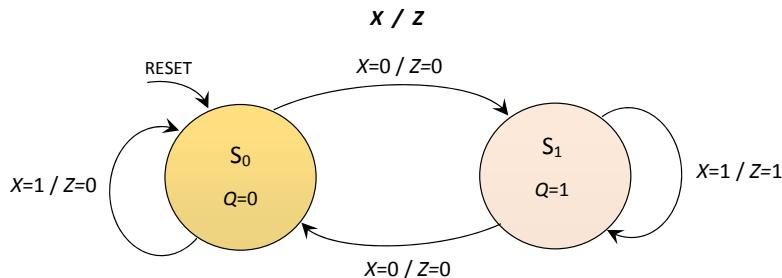
FIGURA 8. Sumador binario natural a BCD

Tema 4

1. Ejercicio 1:

a) Modelo Mealy

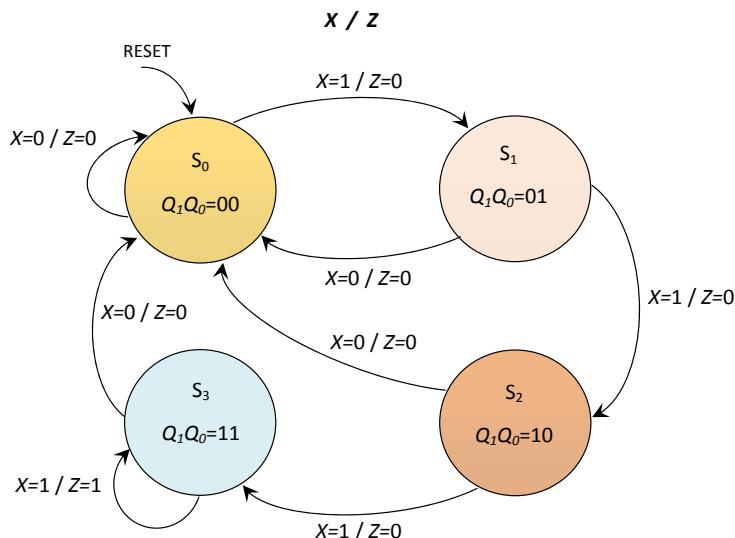
- 1) $Z = X \cdot Q$
- 2) $D = X \cdot Q + \bar{X} \cdot \bar{Q}$
- 3) $Q^* = D$



4)

b) Modelo Mealy

- 1) $Z = Q_1 \cdot Q_0 \cdot X$
- 2) $J_1 = X \cdot Q_0$
- 3) $K_1 = \bar{X}$
- 4) $J_0 = X$
- 5) $K_0 = \bar{X} + \bar{Q}_1$



6)

2. Ejercicio 2:

- a) $Z = Q_2$
- b) $J_2 = X \cdot Q_1 \cdot \bar{Q}_0$
- c) $K_2 = \bar{X}$
- d) $J_1 = \bar{X} \cdot \bar{Q}_0$
- e) $K_1 = \bar{X}$
- f) $J_0 = X$
- g) $K_0 = \bar{X}$
- h) $\overline{RST} = CLR_1 = CLR_0$

3. Ejercicio 3:

- a) $Z = Q_0$
- b) $J_1 = Q_0$
- c) $K_1 = Q_0$
- d) $J_0 = Q_1 \oplus X$
- e) $K_0 = Q_1 \oplus X$
- f) $\overline{RST} = CLR_1 = CLR_0$

4. Ejercicio 4:

- a) $Z = Q_1 \cdot \bar{Q}_0$
- b) $D_1 = X \cdot \underline{Q}_0 + X \cdot Q_1$
- c) $D_0 = X \cdot \bar{Q}_1$
- d) $\overline{RST} = CLR_1 = CLR_0$

5. Ejercicio 5:

- a) $AG = Q_1$
- b) $BG = Q_0$
- c) $D_1 = AR$
- d) $D_0 = \overline{AR} \cdot BR$
- e) $\overline{RST} = CLR_1 = CLR_0$

6. Ejercicio 6:

- a) $Z = Q_1 \cdot \bar{Q}_0$
- b) $J_1 = X \cdot \underline{Q}_0$
- c) $K_1 = X + \underline{Q}_0$
- d) $J_0 = \bar{X}$
- e) $K_0 = Q_1$
- f) $\overline{RST} = CLR_1 = CLR_0$

7. Ejercicio 7:

- a) $Z_1 = Q_1$
- b) $Z_0 = Q_0$
- c) $D_1 = Q_1 \oplus Q_0 + E \cdot Q_1$
- d) $D_0 = Q_1 \cdot Q_0 + E \cdot \bar{Q}_0$
- e) $\overline{RST} = CLR_1 = CLR_0$

8. Ejercicio 8:

- a) $Z = \overline{Q}_0$
- b) $J_1 = Q_0$
- c) $K_1 = Q_0$
- d) $J_0 = Q_1 \oplus X$
- e) $K_0 = Q_1 \oplus \overline{X}$
- f) $\overline{RST} = CLR_1 = CLR_0$

Tema 5

1. Ejercicio 1: Explica la diferencia entre *dirección* de memoria y *posición* de memoria.

La dirección de memoria es el *número binario* que especifica cada dato contenido en la memoria, de modo que podemos acceder a él aplicando ese valor binario al bus de direcciones de la memoria.

La posición de memoria es el *contenido de la memoria* especificado por una valor definido de dirección de memoria.

2. Ejercicio 2: La capacidad de las siguientes unidades de memoria se especifica por el número de palabras multiplicado por el número de bits por palabra. ¿Cuántas líneas de dirección y líneas de entrada-salida de datos se necesitan en cada caso? ¿Cuál es el número de bytes de capacidad de cada una?

a) $4K \times 8 = 4 \times 2^{10} \times 8 = 4 \times 1,024 \times 8 = 32,768 \text{ bits}; 32,738/8 = 4,096 \text{ bytes} = 4 \text{ K bytes}$

4 K palabras son $4 \cdot 2^{10} = 2^2 \cdot 2^{10} = 2^{12} \Rightarrow 12$ bits de dirección.

Longitud de palabra 8 bits $\Rightarrow 8$ bits de entrada-salida.

b) $2G \times 32 = 2 \times 2^{30} \times 32 = 2 \times 1,073,741,824 \times 32 = 68,719,476,736 \text{ bits}; 68,719,476,736/8 = 8,589,934,592 \text{ bytes} = 8 \text{ G bytes}$

2 G palabras son $2 \cdot 2^{30} = 2^1 \cdot 2^{30} = 2^{31} \Rightarrow 31$ bits de dirección.

Longitud de palabra 32 bits $\Rightarrow 32$ bits de entrada-salida.

c) $16M \times 16 = 16 \times 2^{20} \times 16 = 16 \times 1,048,576 \times 16 = 268,435,456 \text{ bytes}; 268,435,456/8 = 33,554,432 \text{ bytes} = 32 \text{ M bytes}$

16 M palabras son $16 \cdot 2^{20} = 2^4 \cdot 2^{20} = 2^{24} \Rightarrow 24$ bits de dirección.

Longitud de palabra 16 bits $\Rightarrow 16$ bits de entrada-salida.

d) $256K \times 64 = 256 \times 2^{10} \times 64 = 256 \times 1,024 \times 64 = 16,777,216 \text{ bits}; 16,777,216/8 = 2,097,152 \text{ bytes} = 2 \text{ G bytes}$

256 K palabras son $256 \cdot 2^{10} = 2^8 \cdot 2^{30} = 2^{38} \Rightarrow 38$ bits de dirección.

Longitud de palabra 64 bits $\Rightarrow 64$ bits de entrada-salida.

3. Ejercicio 3: En la memoria SRAM de 4×4 bits de capacidad de la Figura 7 se usa un decodificador binario natural de 2 entradas y cuatro puertas OR de cuatro entradas cada una. Si queremos expandir la capacidad de esa memoria hasta 256×8 bits, ¿cómo serían el decodificador y las puertas OR necesarias?

En la Figura 7 se observa cómo cada salida del decodificador corresponde a la señal *Seleccionar* de una serie de celdas binarias, las correspondientes a una palabra. Por tanto, si queremos 256 palabras, necesitamos 256 líneas a la salida del decodificador. Un decodificador binario natural de $256 = 2^8$ salidas tiene 8 bits de entrada, las líneas de dirección.

Además, como la señal *Salida* de cada celda binaria de una palabra está unida a una puerta OR, si tenemos 8 bits de longitud de palabra, necesitamos 8 puertas OR, cada una con 256 entradas, tantas como palabras tenemos.

- Ejercicio 4: Diseña, utilizando los decodificadores y puertas lógicas que sean necesarios, un sistema de memoria de $4K \times 8$ (1 byte de longitud de palabra) que usa circuitos integrados de $1K \times 4$ bits de capacidad.

Como vamos a aumentar la longitud de palabra, necesitamos duplicar los bits de entrada y de salida. Por tanto, necesitamos dos bloques de $4K \times 4$.

Además, como hemos aumentado el número de palabras, hay que aumentar el número de líneas de dirección de 10 ($1K = 2^{10}$, diez líneas de dirección) a 12 ($4K = 2^{12}$, doce líneas de dirección). Para habilitar cada uno de los cuatro nuevos bloques de memoria, se utiliza un decodificador binario natural de dos entradas.

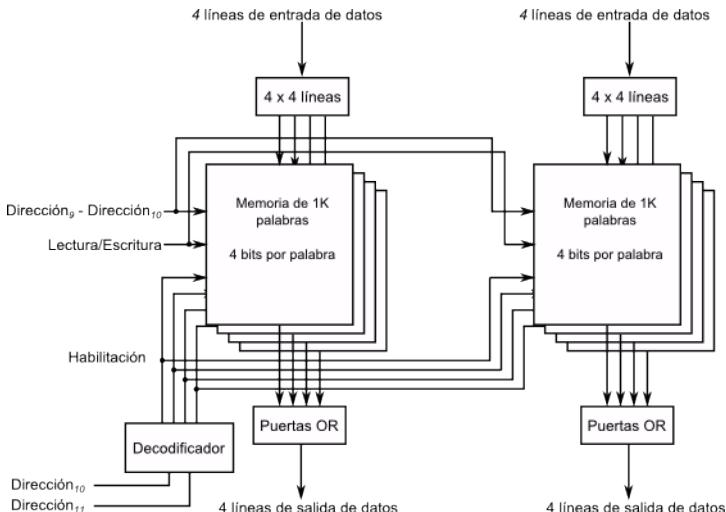


FIGURA 9. Sistema de memoria de $4K \times 8$

Tema 6

1. Ejercicio 1: Realiza, con la ayuda de un cronograma que contenga todas las señales, el análisis del funcionamiento del sistema descrito por el siguiente diagrama ASM (ver Fig. 10), en el que a partir de una señal S, se controla el funcionamiento de un contador de salida A ($A_2A_1A_0$, 3 bits) y las señales de salida E y F:

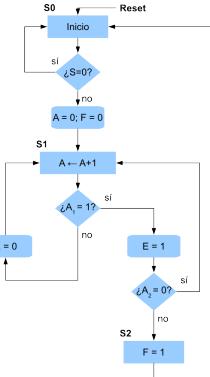


FIGURA 10. Diagrama ASM para analizar

Se trata de activar dos señales de salida E y F, en función del valor de un contador de tres bits, cuyo valor viene expresado por el número de tres bits A, la salida del contador. Aparte de los estados S_0 y S_2 , tenemos las serie de estados S_1 , uno por cada valor que adopte la cuenta (variable de estado) del contador.

Como siempre, todos los cambios de estado son sincronizados por una señal de reloj CLK, y además disponemos de una señal de inicio asíncrono Reset que nos lleva al estado inicial.

En el cronograma de la Fig. 11 aparece el comportamiento del sistema. Es importante notar que los cambios en la señales A, E y F se producen de modo asíncrono porque están asociados a un cambio en una variable. El cambio de F a uno, en cambio, se produce sincrónicamente porque se produce por un cambio de estado.

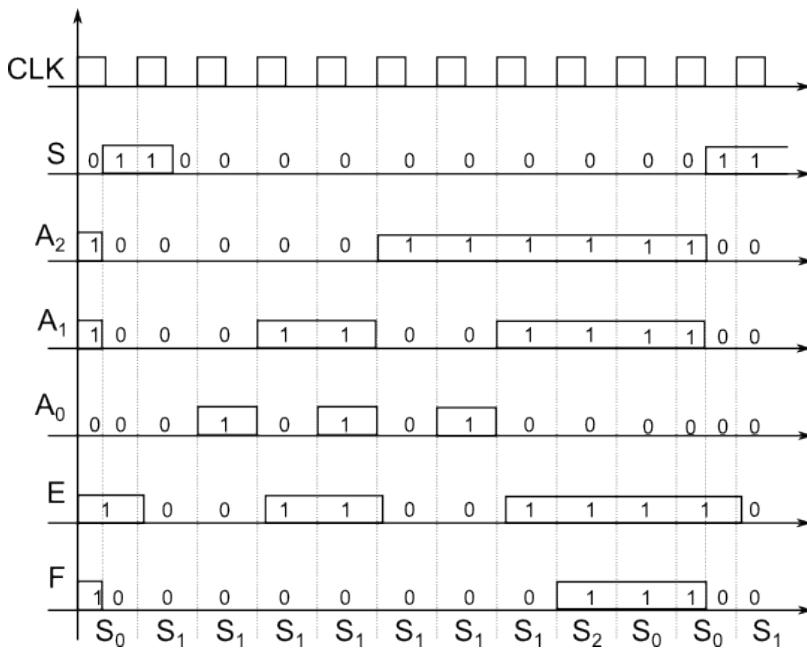


FIGURA 11. Cronograma del sistema del ejercicio 1

2. Ejercicio 2: Diseñad un circuito secuencial síncrono que realice la función de multiplicar dos números enteros sin signo de 4 bit. El resultado aparecerá en un dato de 8 bits, junto con un bit que indique cuándo se ha terminado la operación.

NOTA: El diseño puede ser semejante a la multiplicación en papel, es decir, por cada cifra del multiplicador, se va sumando el multiplicando desplazado una posición a la izquierda al resultado anterior. Tened en cuenta que cada cifra del multiplicador sólo puede ser 0 ó 1. El bit para indicar que se ha terminado la operación se puede obtener con un registro de desplazamiento a la derecha del multiplicador que sea 1 cuando todos los bits del registro sean 0.

$$\begin{array}{r}
 & 1 & 1 & 0 & 1 & \text{Multiplicador} \\
 \times & 1 & 0 & 1 & 1 & \text{Multiplicando} \\
 \hline
 & 1 & 1 & 0 & 1 \\
 & 1 & 1 & 0 & 1 \\
 & 0 & 0 & 0 & 0 \\
 & 1 & 1 & 0 & 1 \\
 \hline
 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & \text{Resultado}
 \end{array}$$

Si el multiplicador es A y el multiplicando B, lo único que hay que hacer es ir sumando en un registro, que será el producto P, el valor de A desplazado a la izquierda si hay un 1 en la cifra actual de B. Si no, se mantiene el último

valor. Desplazando B a la derecha, la cifra actual de B será siempre la menos significativa (b_0) y además sabremos que hemos recorrido todas las cifras de B porque serán todas cero (bit entrante = 0). Comprobando desde el inicio, evitamos resultados erróneos, ya que si $B = 0$, $P = 0$.

$$P = 0$$

Desde $i = 0$ hasta $i = 3$ repetir

Si $B = 0$, salir

Si $b_0 = 1, P = P + A$

Desplazar izquierda A

Desplazar derecha B

$$i \leftarrow i + 1$$

fin

Una vez definida la operación, diseñaremos una ruta de datos que ejecute la secuencia de acciones que necesitamos para completar la operación (ver Fig. 12). Necesitamos un registro desplazador a la izquierda para A, uno a la derecha para B y otro para guardar los resultados de la suma $P+A$. Inicialmente hay que cargar 0 en este registro, de modo que pondremos un multiplexor para elegir si se carga el último resultado o cero.

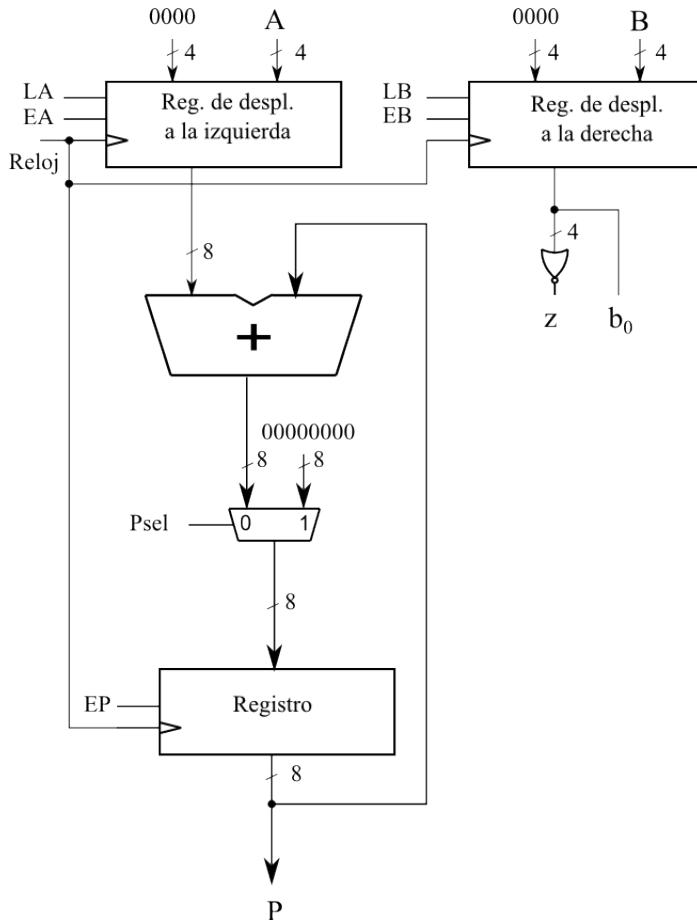


FIGURA 12. Ruta de datos del multiplicador

Finalmente, diseñamos un circuito secuencial síncrono que genera las señales de control de nuestra ruta de datos (unidad de control, ver Fig. 13), en la secuencia adecuada para cumplir con nuestra tarea. Supondremos que el usuario activa las cargas paralelas de los registros de desplazamiento (LA, LB) y que informa de cuándo están cargados los operandos A y B mediante una señal de control s. En este diseño tenemos tres estados: S0, S1 y S2. Las entradas son: s, z y b₀. Las salidas son EA, EB, EP y Psel.

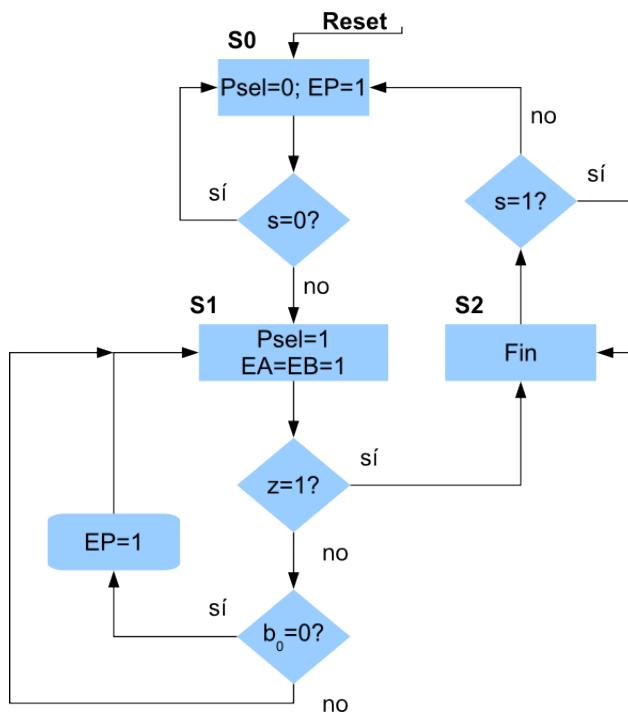


FIGURA 13. Diagrama ASM del multiplicador: unidad de control

El diseño final (ver Fig. 14, no se muestra el reloj) incluye la ruta de datos con la unidad de control y todas las señales del sistema completo:

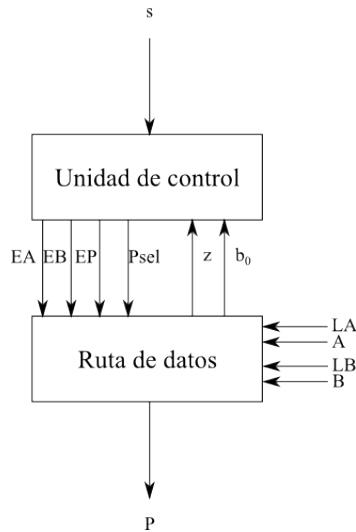


FIGURA 14. Multiplicador: diseño completo