Principios y métodos combinatoriales en sistemas automáticos digitales

Germán Andrés Holguín Londoño,
(Pereira, Risaralda, Colombia, 1977).

Magister en Ingeniería Eléctrica e Ingeniero Electricista de la Universidad Tecnológica de Pereira. Profesor Titular de la Facultad de Ingenierías Eléctrica, Electrónica, Física y Ciencias de la Computación de la Universidad Tecnológica de Pereira.

Autor de los libros: *Manipuladores Paralelos: Síntesis, Análisis y aplicaciones* (2018), *Fundamentos Teóricos Para Los Automatas Industriales* (2010) y *Medidas Eléctricas* (2004)

Ha publicado artículos en revistas especializadas nacionales e internacionales.

Miembro del grupo de investigación en Gestión de Sistemas Eléctricos, Electrónicos y Automáticos.

gahol@utp.edu.co

Mauricio Holguín Londoño,
(Pereira, Risaralda, Colombia, 1974).

Ph.D. en Ingeniería e Ingeniero Electricista de la Universidad Tecnológica de Pereira. Profesor Titular de la Facultad de Ingenierías Eléctrica, Electrónica, Física y Ciencias de la Computación de la Universidad Tecnológica de Pereira.

Autor de los libros: *Pronóstico De Vida Útil Remanente En Rodamientos Con Base En La Estimación De La Probabilidad De la Degradación* (2019), *Fundamentos Teóricos Para Los Automatas Industriales* (2010) y *Automatismos Industriales* (2008).

Ha publicado artículos en revistas especializadas nacionales e internacionales.

Director del grupo de investigación en Gestión de Sistemas Eléctricos, Electrónicos y Automáticos.

mau.hol@utp.edu.co

Principios y métodos combinatoriales en sistemas automáticos digitales

Germán Andrés Holguín Londoño
Mauricio Holguín Londoño

Facultad de Ingenierías
Colección Textos Académicos
2021

Holguín Londoño, Germán Andrés
Principios y métodos combinatoriales en sistemas automáticos digitales / Germán Andrés Holguín Londoño y Mauricio Holguín Londoño. -- Pereira : Universidad Tecnológica de Pereira, 2021.

257 páginas. -- (Colección Textos académicos).

e-ISBN: 978-958-722-597-6

1. Sistemas de numeración ponderados 2. Codificación aritmética 3. Sistema binario 4. Lógica proposicional 5. Funciones lógicas 6. Sistemas automáticos y de control

CDD. 629.8

Principios y métodos combinatoriales en sistemas automáticos digitales

© Germán Andrés Holguín Londoño

© Mauricio Holguín Londoño

© Universidad Tecnológica de Pereira

eISBN: 978-958-722-597-6

Universidad Tecnológica de Pereira
Vicerrectoría de Investigaciones, Innovación y Extensión
Editorial Universidad Tecnológica de Pereira
Pereira, Colombia

Coordinador editorial:

Luis Miguel Vargas Valencia

luismvargas@utp.edu.co

Teléfono 313 7381

Edificio 9, Biblioteca Central "Jorge Roa Martínez"

Cra. 27 No. 10-02 Los Álamos, Pereira, Colombia

www.utp.edu.co

Montaje y producción:

María Alejandra Henao Jiménez

Universidad Tecnológica de Pereira

Pereira

CONTENIDO

Prefacio	11
-----------------------	----

CAPÍTULO UNO

1. ARITMÉTICA Y CODIFICACIÓN BINARIA	15
1.1. DEFINICIONES	16
1.2. SISTEMAS DE NUMERACIÓN PONDERADOS.....	17
1.2.1. Representaciones.....	17
1.2.2. Conteo	20
1.3. CONVERSIÓN ENTRE BASES	22
1.3.1. Conversión de cualquier base a base decimal.....	22
1.3.2. Conversión de base decimal a cualquier base.....	22
1.3.3. Conversión de cualquier base a cualquier base	26
1.3.4. Conversión entre bases A y B cuando $B = A_k$	27
1.4. AGRUPACIONES BINARIAS	29
1.5. REPRESENTACIÓN DE NÚMEROS CON SIGNO.....	29
1.5.1. Signo-magnitud.....	29
1.5.2. Introducción a sistemas por complemento	30
1.5.3. Complemento disminuido a la base	31
1.5.4. Complemento a la base	33
1.5.5. Representación sesgada.....	34
1.5.6. Comparación de representaciones con signo.....	35
1.5.7. Números en punto flotante IEEE Std 754-1985.....	35
1.5.8. Rango de los datos	37
1.6. ARITMÉTICA BINARIA	38
1.6.1. Suma y resta.....	38
1.6.2. Desbordamiento.....	39
1.6.3. Multiplicación	40
1.6.4. División	46
1.7. CÓDIGOS BINARIOS.....	52
1.7.1. Códigos alfanuméricos.....	53
1.7.2. Código BCD	53
1.7.3. Suma y resta en BCD8421	54
1.7.4. Código Gray	58
1.8. CÓDIGOS PARA DETECCIÓN DE ERRORES.....	60
1.8.1. Características de los códigos para detección de errores.....	61
1.8.2. Códigos básicos de paridad.....	62
1.8.3. Códigos de repetición	63
1.8.4. Códigos Hamming.....	64
1.9. EJERCICIOS PROPUESTOS.....	67

CAPÍTULO DOS

2. LÓGICA DE PROPOSICIONES	73
2.1. DEFINICIONES	73
2.1.1. Proposiciones	74
2.1.2. Tipos de proposiciones.....	75
2.2. LÓGICA PROPOSICIONAL FORMAL	75
2.2.1. Alfabeto	75
2.2.2. Notaciones simbólicas.....	77
2.2.3. Jerarquía de conectivas lógicas.....	77
2.2.4. Fórmulas lógicas	78
2.2.5. Conjunciones.....	80
2.2.6. Proceso de formalización.....	80
2.3. TABLAS DE VERDAD	84
2.3.1. Definición	85
2.3.2. Tablas de verdad de las conectivas lógicas.....	85
2.3.3. Tablas de verdad de proposiciones moleculares	85
2.4. CLASIFICACIÓN DE LAS FÓRMULAS LÓGICAS	86
2.5. FÓRMULAS EQUIVALENTES	87
2.5.1. Relación entre equivalencias y tautologías	87
2.5.2. Tautologías fundamentales	88
2.6. CONECTIVAS LÓGICAS EN SISTEMAS DIGITALES.....	89
2.7. EJERCICIOS PROPUESTOS.....	90

CAPÍTULO TRES

3. FUNCIONES LÓGICAS	95
3.1. DEFINICIONES	95
3.1.1. Funciones lógicas básicas.....	97
3.1.2. Funciones lógicas derivadas	100
3.2. REPRESENTACIÓN DE FUNCIONES LÓGICAS.....	104
3.2.1. Representación implícita.....	104
3.2.2. Representación explícita	104
3.2.3. Diagrama de tiempo	104
3.2.4. Diagrama lógico	105
3.3. MÉTODOS ALGEBRAICOS DE ANÁLISIS Y SÍNTESIS DE FUNCIONES LÓGICAS.....	106
3.3.1. Diagramas de Venn	107
3.3.2. Principio de la dualidad	109
3.3.3. Teoremas del álgebra Booleana	110
3.3.4. Suficiencia de la NAND	115
3.3.5. Suficiencia de la NOR.....	116
3.3.6. Álgebra de la XOR y la XNOR	116
3.4. FORMAS ALGEBRAICAS ESTÁNDARES.....	118
3.4.1. Suma de productos (SOP)	118
3.4.2. Producto de sumas (POS).....	118
3.4.3. Forma SOP canónica.....	119
3.4.4. Forma POS canónica.....	121
3.4.5. Deducción de canónicas por numeración.....	123
3.4.6. Equivalencias entre formas canónicas.....	123

3.5. CIRCUITOS LÓGICOS ESTÁNDARES.....	124
3.5.1. Universalización.....	126
3.5.2. Funciones con especificación incompleta.....	127
3.6. EJERCICIOS PROPUESTOS.....	128

CAPÍTULO CUATRO

4.MINIMIZACIÓN Y SÍNTESIS DE FUNCIONES LÓGICAS	133
4.1. DEFINICIONES.....	133
4.2. MAPAS DE VEITCH-KARNAUGH.....	134
4.2.1. Mapa VK para función de dos variables.....	135
4.2.2. Mapa VK para función de tres variables.....	136
4.2.3. Mapa VK para función de cuatro variables.....	137
4.2.4. Mapa VK para funciones de cinco, o más, variables.....	138
4.2.5. Reglas de construcción para mapas VK.....	138
4.3. MINIMIZACIÓN POR MAPAS VK.....	139
4.3.1. Criterios de agrupación en mapas VK.....	140
4.3.2. Implicantes	142
4.3.3. Algoritmo de minimización por vecindades.....	144
4.3.4. Algoritmo de minimización por I.P.E.	145
4.3.5. Implicados.....	146
4.3.6. Mapas VK con especificación incompleta	147
4.3.7. Mapas VK con ingreso de variables.....	149
4.4. MINIMIZACIÓN POR QUINE - McCLUSKEY - PETRICK (Q-M-P)	154
4.4.1. Algoritmo para la minimización de Quine-McCluskey (Q-M)	154
4.4.2. Algoritmo de Petrick.....	159
4.4.3. Algoritmo Q-M-P con prescindibles.....	160
4.5. LA FUNCIÓN MÍNIMA	162
4.6. EJERCICIOS PROPUESTOS.....	165

CAPÍTULO CINCO

5.PARÁMETROS DE COMPUERTAS Y COMPLEJIDAD CIRCUITAL	169
5.1. PARÁMETROS DE VOLTAJE Y CORRIENTE.....	169
5.1.1. Parámetros de voltaje	170
5.1.2. Márgenes de ruido	171
5.1.3. Parámetros de corriente.....	172
5.2. PARÁMETROS DINÁMICOS.....	173
5.2.1. Tiempos de retardo y de propagación.....	173
5.2.2. Disipación de potencia.....	174
5.3. FAMILIAS LÓGICAS Y COMPATIBILIDAD	174
5.3.1. Series de familias.....	174
5.3.2. Compatibilidad entre familias lógicas.....	176
5.4. CIRCUITOS INTEGRADOS - IC	178
5.5. COMPLEJIDAD Y PROFUNDIDAD CIRCUITAL.....	172
5.6. LA IMPLEMENTACIÓN MÍNIMA GLOBAL	183
5.7. COMPLEJIDAD, MINIMIZACIÓN Y UNIVERSALIZACIÓN.....	189
5.8. RIESGOS DE TIEMPO	191

5.8.1. Riesgo estático y riesgo dinámico	191
5.8.2. Corrección de riesgos de tiempo	192
5.9. EJERCICIOS PROPUESTOS.....	193

CAPÍTULO SEIS

6. DISEÑO Y SÍNTESIS PARA ARITMÉTICA COMBINACIONAL	197
6.1. CIRCUITOS PARA ARITMÉTICA BINARIA	197
6.1.1. Sumador medio - Half Adder (HA)	198
6.1.2. Sumador completo - Full Adder (FA)	199
6.1.3. Sumador en cascada - Ripple Carry Adder (RCA)	201
6.1.4. Sumador/Restador en cascada	202
6.1.5. Verificador de desbordamiento.....	204
6.2. COMPARADOR BINARIO.....	205
6.3. CIRCUITOS SOBRE SISTEMAS NO PONDERADOS	207
6.3.1. Sumador BCD natural.....	207
6.3.2. Restador BCD natural	209
6.3.3. Codificador - Decodificador de paridad.....	209
6.4. SÍNTESIS POR EXPANSIÓN.....	214
6.5. EJERCICIOS PROPUESTOS.....	215

CAPÍTULO SIETE

7. TRATAMIENTO Y ENRUTAMIENTO DE DATOS	219
7.1. CODIFICADORES Y DECODIFICADORES	219
7.1.1. Decodificadores.....	220
7.1.2. Codificadores.....	227
7.2. MULTIPLEXORES Y DEMULTIPLEXORES	232
7.2.1. Multiplexores.....	233
7.2.2. Demultiplexores	239
7.3. APLICACIÓN: UNIDAD LÓGICO ARITMÉTICA	241
7.4. EJERCICIOS PROPUESTOS.....	247

Referencias	249
-------------------	-----

Lista de figuras

1.1. Señal analógica vs señal digital	16
1.2. Señal digital vs señal binaria	17
1.3. Fundamento de las representaciones por complemento	31
1.4. Codificadores Binario (Izq.) y Gray (Der.) con 3 bits	60
1.5. Modelo de sistema de control de errores	61
2.1. Sensores contra incendio	82
3.1. Sistema lógico de 1 entrada y 1 salida.....	96
3.2. Sistema lógico con 2 entradas y 1 salida.....	97
3.3. Simbología estándar para la AND	98
3.4. Diagrama de contactos y tabla de verdad para la AND.....	98
3.5. Simbología estándar para la OR	99
3.6. Diagrama de contactos y tabla de verdad para la OR.....	99
3.7. Simbología estándar para la NOT	99
3.8. Diagrama de contactos y tabla de verdad para la NOT	100

3.9. Simbología estándar para la NAND.....	100
3.10. Diagrama de contactos y tabla de verdad para la NAND	101
3.11. Simbología estándar para la NOR.....	101
3.12. Diagrama de contactos y tabla de verdad para la NOR.....	102
3.13. Simbología estándar para la XOR.....	102
3.14. Diagrama de contactos y tabla de verdad para la XOR	103
3.15. Simbología estándar para la XNOR.....	103
3.16. Diagrama de contactos y tabla de verdad para la XNOR	104
3.17. Ejemplo de diagrama de tiempo.....	105
3.18. Ejemplo de diagrama lógico.....	105
3.19. Diagrama de tiempos para $Z = (xy) \quad (xz + x)$	106
3.20. Diagrama lógico para $Z = (xy) \quad (xz + x)$	106
3.21. Analogía entre diagramas de Venn y álgebra de Boole	108
3.22. Función NOT implementada con NAND.....	115
3.23. Función OR implementada con NAND	115
3.24. Función AND implementada con NAND	115
3.25. Función NOT implementada con NOR	116
3.26. Función AND implementada con NOR.....	116
3.27. Función OR implementada con NOR	116
3.28. Circuitos lógicos estándares.....	125
3.29. Circuito lógico universalizado.....	127
4.1. Diagrama de Venn para función de 2 variables.....	135
4.2. Mapas VK para 2 variables	135
4.3. Diagrama de Venn para función de 3 variables.....	136
4.4. Mapas VK para 3 variables	136
4.5. Diagrama de Venn para función de 4 variables.....	137
4.6. Mapas VK para 4 variables	137
4.7. Mapas VK para 5 y 6 variables.....	138
4.8. Ejemplo básico de mapa VK	139
4.9. Agrupación en mapa VK	140
4.10. Ejemplo de simplificación usando mapa VK, para forma SOP.....	141
4.11. Ejemplo de simplificación usando mapa VK, para forma POS.....	142
4.12. Implicantes de $f(A;B;C;D) = Pm(0; 2; 4; 5; 7; 9; 10; 13; 15)$	142
4.13. Simplificación por algoritmo de vecindades.....	145
4.14. Simplificación por algoritmo de I.P.E	146
4.15. Simplificación por implicados para $f(A;B;C;D) = QM$ $(0; 4; 5; 6; 7; 8; 9; 13; 15)$	147
4.16. Mapa VK para $f(A;B;C;D) = Pm(0; 5; 7; 9; 10) + Pd(2; 8; 13; 14; 15)$	148
4.17. Mapa VK para $f(A;B;C;D) = QM(1; 3; 4; 6; 11; 12) QD(2; 8; 13; 14; 15) \dots$	149
4.18. Mapas VK de 2 variables, con ingreso de una variable.....	150
4.19. Mapas VK de 3 variables, con ingreso de una variable.....	151
4.20. Mapa VK de 2 variables, con ingreso de dos variables.....	152
4.21. Mapa VK de 3 variables, con ingreso de una variable y prescindibles	153
4.22. I.P. sobre mapa VK y Q-M	155
4.23. I.P.E. sobre mapa VK y Q-M.....	156
4.24. Solución sobre mapa VK Vs QM	157
4.25. Obtención de función mínima	164
5.1. Parámetros de voltaje para una compuerta lógica.....	170

5.2. Rangos de voltaje para una compuerta lógica.....	171
5.3. Parámetros de tiempo en una compuerta lógica.....	173
5.4. Gráfico acíclico dirigido para formas SOP canónicas.....	180
5.5. Gráfico acíclico dirigido con reutilización	181
5.6. Ejemplo de implementación no mínima, con dos funciones	184
5.7. Ejemplo de implementación no mínima, con tres funciones	186
5.8. Gráfico acíclico para minimización global no universalizada.....	190
5.9. Gráfico acíclico para minimización global universalizada	190
5.10. Riesgo de tiempo	192
5.11. Riesgo dinámico	192
5.12. Solución, con mapas VK, libre de riesgo estático.....	193
6.1. Mapa VK para la suma en un sumador medio.....	198
6.2. Circuito lógico para sumador medio.....	199
6.3. Mapa VK para la suma y acarreo en un sumador completo	200
6.4. Circuito lógico para sumador completo.....	201
6.5. Sumador en cascada	201
6.6. Circuito lógico de un sumador/restador en cascada.....	203
6.7. Circuito lógico para verificador de desbordamiento	205
6.8. Mapa VK para las tres funciones de un comparador de dos palabras de 2 bits cada una.....	200
6.9. Mapa VK para función de corrección de número no BCD natural.....	207
6.10. Circuito para sumar dos palabras de 4 bits en BCD natural.....	208
6.12. Codificador - Decodificador de paridad	210
6.11. Circuito para restador BCD natural.....	211
6.13. Mapas VK para contador de unos	212
6.14. Mapa VK para agregador de paridad	213
7.1. Diagrama general de un decodificador.....	220
7.2. Circuito lógico de un decodificador 3x8	221
7.3. Circuito lógico de un decodificador BCD a decimal.....	223
7.4. Decodificador 2x4 con habilitación	224
7.5. Despliegue de dígitos decimales en 7 segmentos lumínicos	224
7.6. Decodificador 4 16 como expansión de dos decodificadores.....	226
7.7. Contador de unos para palabras de 3 bits, empleando decodificador como generador de mintérminos o maxtérminos.....	228
7.8. Diagrama general de un codificador sin prioridad.....	228
7.9. Circuito lógico de un codificador 8x3 sin prioridad.....	229
7.10. Circuito de un codificador Decimal a BCD sin prioridad	232
7.11. Diagrama general de un multiplexor N a 1.....	233
7.12. Circuito lógico de un multiplexor 4 a 1	234
7.13. Multiplexor 16 a 1 con base en expansión de multiplexores 4 a 1	235
7.14. Cuádruple multiplexor 2 a 1.....	236
7.15. Contador de unos implementado con multiplexores 8 a 1	237
7.16. Contador de unos implementado con multiplexores 4 a 1	238
7.17. Contador de unos implementado con multiplexores 2 a 1	239
7.18. Diagrama general de un demultiplexor 1 a N.....	240
7.19. Circuito lógico de un demultiplexor 1 a 4.....	241
7.20. Símbolo lógico de una ALU de N bits.....	242
7.21. Esquema modular de una ALU con 3 bits de selección	243

7.22. Módulos de una ALU de 1-bit	243
7.23. Unidad lógica de 1-bit.....	244
7.24. Módulos de una AU	245
7.25. Circuito lógico de B-CONTROL con multiplexor 4 a 1	246

Lista de tablas

1.1. Comparación de conteo entre diferentes bases	21
1.2. Agrupaciones binarias y sus límites de representación	29
1.3. Comparación de representaciones para 4 bits	35
1.4. Resumen de códigos BCD.....	54
1.5. Equivalencias Binario - Gray para 4 bits	58
1.6. Códigos básicos de paridad.....	63
1.7. Códigos de repetición	63
1.8. Ejemplo de código Hamming(7,4)	65
1.9. Ejemplo de comprobación Hamming(7,4).....	65
1.10. Listado de polinomios generadores para códigos CRC.....	66
2.1. Tabla de notaciones simbólicas	77
2.2. Conversión de conjunciones comunes a conectivas lógicas	81
2.3. Tablas de verdad de las conectivas lógicas.....	85
2.4. Tautologías fundamentales	89
2.5. Conectivas lógicas en sistemas digitales	89
3.1. Tabla de verdad para $Z = (xy) \quad (xz + x)$	105
4.1. Tabla de verdad reducida para $f(A;B;C) = Pm(1; 3; 4; 5; 6)$	149
4.2. Tabla de verdad reducida, ejemplo Mapa VK de 3 variables, con ingreso de una variable	150
4.3. Tabla de verdad reducida, ejemplo Mapa VK de 2 variables, con ingreso de dos variables	151
4.4. Tabla de verdad reducida, ejemplo Mapa VK de 3 variables, con ingreso de una variable y prescindibles	153
4.5. Ejemplo Quine McCluskey por minterminos, tabla de vecindades.....	156
4.6. Ejemplo Quine McCluskey por minterminos, tabla de I.P.E.	157
4.7. Ejemplo Quine McCluskey por minterminos, minterminos cubiertos.....	157
4.8. Ejemplo Quine McCluskey por maxtérminos, tabla de vecindades	158
4.9. Ejemplo Quine McCluskey por maxtérminos, tabla de cubierta	158
4.10. Ejemplo Q-M-P con prescindibles en forma SOP, tabla de vecindades	160
4.11. Ejemplo Q-M-P con prescindibles en forma SOP, tabla de cubierta	160
4.12. Ejemplo Q-M-P con prescindibles en forma POS, tabla de vecindades	161
4.13. Ejemplo Q-M-P con prescindibles en forma POS, tabla de I.P.E.	162
4.14. Ejemplo Q-M-P con prescindibles en forma POS, tabla de cubierta	162
4.15. Tabla de verdad para BCD natural 5.....	164
5.1. Principales parámetros de las familias lógicas	176
5.2. Circuitos integrados comunes para compuertas lógicas	178
5.3. Tabla de verdad circuito que suma tres bits	179
5.4. Ejemplo Quine McCluskey para implementación mínima global con dos funciones, tabla de I.P.	185
5.5. Ejemplo Quine McCluskey implementación mínima global con dos funciones, tabla de I.P.E.	185

5.6. Ejemplo Quine McCluskey para implementación mínima global con tres funciones, tabla de I.P.	187
5.7. Ejemplo Quine McCluskey para implementación mínima global con tres funciones, tabla de I.P.E.	188
6.1. Representación de la operación del sumador medio y su tabla de verdad	198
6.2. Representación de la operación del sumador completo y su tabla de verdad	199
6.3. Complejidad y profundidad para un sumador completo, según base de operaciones booleanas	200
6.4. Tabla de verdad para control del sustraendo en sumador/restador	203
6.5. Tabla de verdad para verificador de desbordamiento	204
6.6. Tabla de verdad para comparador de dos palabras de 2 bits cada una	205
6.7. Tabla de verdad para corrección de número no BCD natural	207
6.8. Tabla de verdad para contador de unos	212
6.9. Tabla de verdad para agregador de paridad	213
6.10. Tabla de verdad del verificador de recepción	214
6.11. Relaciones entre etapas para la síntesis de un comparador de 4 bits	214
7.1. Tabla de verdad para un decodificador 3x8	221
7.2. Tabla con ecuaciones de salida de un decodificador 3x8	221
7.3. Tabla de verdad para un decodificador BCD a decimal	222
7.4. Tabla con ecuaciones de salida para un descodificador 3x8	222
7.5. Tabla de verdad para decodificador 2x4 con habilitación	223
7.6. Tabla de verdad de activación para un decodificador BCD a 7 segmentos	225
7.7. Solución por mapas VK para decodificador BCD a 7 segmentos	225
7.8. Tabla de verdad para un decodificador 4x16	226
7.9. Tabla de verdad para contador de unos en palabras de 3 bits	227
7.10. Tabla de verdad para un codificador 8x3 sin prioridad	229
7.11. Tabla de verdad para un codificador 8x3 con prioridad	230
7.12. Tabla de verdad para un codificador Decimal a BCD sin prioridad	231
7.13. Tabla de verdad para un codificador Decimal a BCD con prioridad	231
7.14. Operación de la salida en un multiplexor 4 a 1	234
7.15. Operación de la salida en un multiplexor 16 a 1	235
7.16. Tabla de verdad para las salidas de un contador de unos, con multiplexor 4 a 1	238
7.17. Tabla de verdad para las salidas de un contador de unos, con multiplexor 2 a 1	239
7.18. Tabla de verdad para las salidas en un demultiplexor 1 a 4	240
7.19. Operación de las salidas en un demultiplexor 1 a 4	240
7.20. Tabla de funciones en una ALU con 3 bits de selección	242
7.21. Tabla de funciones de una LU	244
7.22. Tabla de funciones de una AU	244
7.23. Tabla de funciones y acciones de una AU	245
7.24. Tabla de verdad para B-CONTROL	245
7.25. Tabla de verdad para B-CONTROL con multiplexor 4 a 1	246
7.26. Tabla de complejidad y profundidad para los módulos en una ALU de N bits	246

Prefacio

Los sistemas digitales son ubicuos en el mundo moderno, y para un ingeniero en formación la comprensión y dominio del tema es de fundamental importancia en su desarrollo profesional. En los últimos 30 años, el avance tecnológico de esta área del conocimiento ha sido vertiginoso, tanto en tecnologías de materiales y fabricación de circuitos digitales como en las aplicaciones de software y firmware que se pueden desarrollar sobre las plataformas de hardware digital flexibles y de bajo costo que están disponibles en el mercado hoy en día.

Para el grupo de investigación en Gestión de Sistemas Eléctricos, Electrónicos y Automáticos, del programa de Ingeniería Eléctrica de la Universidad Tecnológica de Pereira, es un gusto presentar a sus estudiantes y a la comunidad académica en general, este documento resultado de la experiencia combinada de los autores de más de 30 años de labor docente e investigativa en ingeniería y en sistemas digitales en particular.

Para los próximos años, el ritmo de crecimiento de esta área del conocimiento no será menor. Por el contrario, se esperan nuevos y mejores sistemas de desarrollo que requieran por parte del usuario conocimientos fundamentales más sólidos, con una conceptualización de más bajo nivel, donde la teoría de los sistemas digitales se pueda fácilmente extrapolar en muchas de las áreas de la automática como la lógica de contactos, el diseño de máquinas de estados, la programación de controladores lógicos, microcontroladores, DSPs, la programación de computadores en general, entre otros. La visión integradora es posiblemente la más importante para un ingeniero de diseño y desarrollo moderno.

Por tal motivo, este documento no pretende ser un texto más de fundamentos de sistemas digitales combinatoriales. El lector tal vez puede percatarse que existen muchos documentos sobre el tema. Consideramos que la educación moderna en ingeniería requiere de tres aspectos a incorporar en el desarrollo de los contenidos de cursos fundamentales: 1) Conocimiento Pedagógico del contenido, 2) Transversalidad y 3) Completitud.

CPC, Conocimiento Pedagógico del Contenido. El conocimiento pedagógico del contenido (PCK por sus siglas en inglés), es una tendencia mundial en el desarrollo curricular y especialmente en ingeniería. Este documento está preparado teniendo en cuenta la experiencia docente de más de 30 años de los autores y la retroalimentación de muchos grupos de estudiantes. Los ejemplos resueltos y ejercicios propuestos han sido cuidadosamente seleccionados para atacar los problemas de aprendizaje más comunes en los estudiantes que están comenzando en este tema.

Transversalidad. A través de la historia, varios autores han dejado un legado significativo al desarrollo de esta área del conocimiento. Apellidos muy importantes como Boole, Veitch, Karnaugh, Shannon, Quine, McCluskey, Petrick, por nombrar solo algunos, han sentado bases que con el tiempo han hecho conexiones entre diferentes áreas que ahora sabemos son totalmente interdependientes. El presente documento busca ayudar al lector a encontrar esas interdependencias, para poder aplicar los mismos conceptos en diferentes aspectos de la ingeniería y la automatización de procesos, entre las cuales se puede mencionar, el diseño computacional de sistemas combinatoriales, el análisis de algoritmos desde la teoría de la complejidad, y la programación general desde los métodos y modelos computacionales.

Complejidad. Se pretende que el lector disponga de todas las herramientas, métodos y competencias para afrontar y dar solución a un problema de ingeniería de automatización que requiera de los principios y métodos combinatoriales, independientemente de las tecnologías particulares que se deseen utilizar o estén de moda en algún momento particular.

La visión de este documento es la de integrar estos tres aspectos, con el fin de brindar al lector una comprensión de los temas con base en una línea pedagógica, que muestra la transversalidad de los sistemas digitales y con el fin de desarrollar y obtener las competencias y metodologías bases para afrontar la conceptualización, diseño e implementación de sistemas que involucran los métodos combinatoriales. El lector encuentra que los temas se desarrollan desde la necesidad y pertinencia, con aplicación en áreas como la electrónica, el control, la automatización y la programación de computadores.

1

CAPÍTULO
UNO

Aritmética y codificación binaria

Con este capítulo se da inicio al recorrido por el mundo combinacional en los sistemas digitales. Como punto de partida se introduce la diferencia entre las señales análogas, binarias y digitales y se justifica el uso de cada una desde la necesidad de representar mejor la señal origen, de realizar operaciones de guardado y lectura de datos en medios físicos, el tamaño mismo de la información y el uso final. Las anteriores necesidades dan cabida a la presentación de sistemas de numeración ponderados y no ponderados. Los sistemas ponderados se usan principalmente para realizar operaciones aritméticas, por lo que se presentan las principales bases de numeración y los procedimientos de conversión entre bases de cualquier tipo, las representaciones de números con y sin signo y la forma de realizar aritmética binaria. Los sistemas no ponderados se usan normalmente para el tratamiento de información de interfaces entre el hombre y la máquina, entre la máquina y el hombre, con el almacenamiento de información, con los procedimientos para guardar y recuperar los datos, así como con el aseguramiento de la integridad de los mismos datos en todos los procesos, entre otros usos. Las representaciones vistas en este capítulo se pueden extender al diseño eléctrico y electrónico, a la automatización de procesos, al diseño de algoritmos y a los sistemas de comunicación, entre otros; por lo que se presentan ejemplos y ejercicios que contextualizan estas aplicaciones.

1.1. DEFINICIONES

Sistema:	<i>Es una combinación de dos o más elementos que realizan, o están en capacidad de realizar, una tarea o función específica.</i>
Dígito:	<i>Unidad que representa una cifra dentro de un sistema numérico.</i>

Se habla de una señal análoga para hacer referencia a cualquier tipo de variable, o función física, continua que definida en un dominio (o valores admisibles de entrada), su rango (o valores que puede producir) puede tomar infinitos valores entre posibles límites. A su vez, se habla de una señal digital para hacer referencia a una variable, o función física, donde su rango solo entrega un conjunto finito de posibles valores discretos.

La figura 1.1 compara una posible señal análoga producto de medir la temperatura en un día típico tropical, contra la misma señal pero de naturaleza digital.

Figura 1.1: Señal análoga vs señal digital

Las señales análogas tienen la ventaja de poder representar mejor el proceso al que están asociadas, además de ser el registro en tiempo real de la variable de interés. Sin embargo, las señales análogas tiene la desventaja que en presencia de ruido, este modifica la señal verdadera y, si además, la señal se guarda y recupera repetidamente, el ruido y el mismo proceso de escritura lectura la puede deteriorar considerablemente. A su vez, la señal digital, es una representación discretizada, donde solo se emplean algunos registros para su rango, lo cual hace que la señal sea una aproximación; pero al seleccionar adecuadamente la separación entre valores discretos, es posible reducir la afectación por ruido o por los procesos repetidos de guardar y recuperar la señal en algún medio de almacenamiento.

Otra forma para reducir aún más las afectaciones por ruido es que la señal digital solo posea dos valores de representación. En este caso, se dice que la señal es binaria, como se muestra en la figura 1.2. Ya que la señal binaria solo posee dos valores mutuamente excluyentes, estos se ajustan para tener la mayor separación posible que permita reducir la influencia del ruido en la señal. Posibles valores binarios son Verdadero-Falso, Frío-Caliente, Si-No, los dígitos 0-1, los valores de voltaje 0V-5V, entre otros. Sin embargo, una señal netamente binaria es una mala representación de una señal digital, y aún más de una señal analógica. Como forma alternativa, se pueden numerar los valores discretos de la señal digital empleando solo valores binarios, obteniendo una escala como la que se muestra entre las señales digital y binaria de la figura 1.2.

Figura 1.2: Señal digital vs señal binaria

De lo anterior, un sistema digital es aquel que utiliza dígitos binarios para el procesamiento de la información. El término *digital* en electrónica, normalmente está asociado a la utilización del sistema de numeración binaria.

Una señal eléctrica analógica puede ser representada por algún código binario mediante un proceso que se denomina **digitalización**. Este consiste de dos etapas básicas:

- Muestreo: proceso por el cual se toma un valor de la señal analógica a intervalos definidos de tiempo (tiempo de muestreo), lo cual implica una discretización del dominio del tiempo.
- Cuantización: proceso por el cual el valor de una señal analógica, en un instante dado de tiempo, se discretiza de acuerdo a su amplitud, la que se divide en niveles o quantums. El tipo más común de cuantización es la uniforme, donde cada nivel tiene la misma amplitud. A cada nivel se le asigna un identificador abstracto que usualmente consiste de una codificación binaria.

Un dispositivo capaz de realizar la digitalización se conoce como **convertidor análogo - digital**.

1.2. SISTEMAS DE NUMERACIÓN PONDERADOS

1.2.1. Representaciones

Un sistema de numeración consiste de un conjunto ordenado de normas y procedimientos con el que se puede expresar cualquier cantidad.

Está conformado por un conjunto de cifras básicas representadas por símbolos o caracteres denominados **dígitos**. Su combinación es la que permite representar una cantidad específica.

En la mayoría de los sistemas de numeración el valor de un dígito, en una cantidad determinada, depende de su posición. Estos son denominados **sistemas de valor posicional o ponderados**. Normalmente el dígito de menor valor se ubica a la derecha (LSD del inglés *less significant digit*) y el de mayor valor posicional se ubica a la izquierda (MSD del inglés *most significant digit*).

$$N = (8163)_{10} \Rightarrow \underbrace{8}_{MSD} \ 1 \ 6 \ \underbrace{3}_{LSD}$$

Existe una gran variedad de sistema de numeración, pero solo a algunos se les ha asignado nombre. A continuación se listan los más conocidos y se indica el número de dígitos propios, para lo cual se dice que un sistema numérico es base b cuando está conformado por b dígitos:

- Binario $b = 2$
- Ternario $b = 3$
- Quinario $b = 5$
- Octal $b = 8$
- Decimal $b = 10$
- Hexadecimal $b = 16$
- Sexagesimal $b = 60$

Los dígitos de los sistemas de numeración más frecuentemente empleados para sistemas digitales son los siguientes:

$b = 2$	0 1
$b = 8$	0 1 2 3 4 5 6 7
$b = 10$	0 1 2 3 4 5 6 7 8 9
$b = 16$	0 1 2 3 4 5 6 7 8 9 A B C D E F

Estos cuatro sistemas son de valor posicional y el peso de un dígito en la cifra N está dado en potencias de b .

Si un número N tiene M dígitos enteros en base b , entonces su **representación posicional** es:

$$(N)_b = \underbrace{D_{M-1}}_{MSD} D_{M-2} \cdots D_2 D_1 \underbrace{D_0}_{LSD}$$

16

Donde el peso de cada dígito está dado por:

$$\underbrace{D_{M-1}}_{b^{M-1}} \underbrace{D_{M-2}}_{b^{M-2}} \cdots \underbrace{D_2}_{b^2} \underbrace{D_1}_{b^1} \underbrace{D_0}_{b^0}$$

Ahora, la cantidad representada por $(N)_b$ también se puede escribir en su **representación polinomial** por:

$$(N)_b = \sum_{i=0}^{M-1} D_i * b^i$$

$$(N)_b = D_0 b^0 + D_1 b^1 + D_2 b^2 + \cdots + D_{M-2} b^{M-2} + D_{M-1} b^{M-1}$$

Siendo el máximo valor posible de $(N)_b$:

$$(N)_{b_{MAX}} = b^M - 1$$

Esta expresión supone que N es un número entero, sin embargo, el concepto puede ser extendido a un número con parte fraccionaria, donde el peso de las posiciones a la derecha del separador (punto) será de exponente negativo, así:

$$(N)_b = \underbrace{D_{M-1}}_{MSD} D_{M-2} \cdots D_2 D_1 D_0 \underbrace{\cdots}_{\text{separador}} \underbrace{D_{-1} D_{-2} \cdots D_{-L}}_{LSD}$$

$$(N)_b = \sum_{i=-L}^{M-1} D_i * b^i = \sum_{i=M-1}^{-L} D_i * b^i$$

Esta definición establece, además, la metodología básica para tomar una cantidad expresada en cualquier base y convertirla a base 10.

EJEMPLOS

1. Un número (N) en base 10 es:

$$(N)_{10} = (1498)_{10} \Rightarrow$$

$$\underbrace{1}_{10^3} \underbrace{4}_{10^2} \underbrace{9}_{10^1} \underbrace{8}_{10^0}$$

Siendo el peso 10^3 el MSD y el peso 10^0 el LSD.

$$\begin{aligned} \text{Luego } (N)_{10} &= 1 * 10^3 + 4 * 10^2 + 9 * 10^1 + 8 * 10^0 \\ (N)_{10} &= 1000 + 400 + 90 + 8 \\ (N)_{10} &= (1498)_{10} \end{aligned}$$

Para cuatro dígitos en base 10 el máximo valor posible es:

$$(N)_{10_{MAX}} = 10^4 - 1 = 9999$$

2. Un ejemplo con parte fraccionaria es:

$$(N)_{10} = (682,739)_{10} \Rightarrow$$

$$(N)_{10} = 6 * 10^2 + 8 * 10^1 + 2 * 10^0 + 7 * 10^{-1} + 3 * 10^{-2} + 9 * 10^{-3}$$

$$(N)_{10} = 600 + 80 + 2 + 0,7 + 0,03 + 0,009$$

$$(N)_{10} = (682,739)_{10}$$

1.2.2. Conteo

Los sistemas de numeración de valor posicional son cíclicos, por lo que agregando dígitos se puede expresar cantidades cada vez más grandes. A continuación se muestra un ejemplo del conteo en algunas bases de interés.

Al examinar la tabla 1.1, se puede observar que el $(12)_{10} = (14)_8 = (C)_{16} = (40)_3 = (13)_9 = (1100)_2$. Además, la base decimal es la base natural para los seres humanos, mientras la base binaria es la base en la que trabajan la mayoría de los sistemas informáticos, aunque crece rápidamente en dígitos requeridos para sus representaciones; a su vez, la base hexadecimal se distingue por ser la preferida para codificar información para ser almacenada, debido a que ocupa menos espacio que las demás y ciertas operaciones es preferible realizarlas en base octal. A continuación, se muestran secuencias de conteos para números mayores, donde se puede observar aspectos de los descritos.

DECIMAL

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, ..., 19, 20, 21, 22, ..., 29, 30, 31, ..., 98, 99, 100, 101, ..., 999, 1000, ..., 9999, 10000, ...

OCTAL

0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 22, ..., 77, 100, 101, ..., 177, 200, ..., 777, 1000, ..., 1777, 2000, ..., 7777, 10000, ...

HEXADECIMAL

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, ..., 1F, 20, ..., 2F, 30, 31, ..., FF, 100, ..., 1FF, 200, ..., FFF, 1000, ..., FFFF, 10000, ...

BINARIO

0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, ..., 1111, 10000, 10001, ..., 11111, ...

Por todo lo anterior, y ante la imposibilidad de hacer una tabla para todos los números, se hace necesario conocer métodos de conversión entre diferentes bases.

Tabla 1.1: Comparación de conteo entre diferentes bases

Decimal	Octal	Hexadecimal	Ternario	Base 9	Binario
0	0	0	0	0	0
1	1	1	1	1	1
2	2	2	2	2	10
3	3	3	10	3	11
4	4	4	11	4	100
5	5	5	12	5	101
6	6	6	20	6	110
7	7	7	21	7	111
8	10	8	22	8	1000
9	11	9	30	10	1001
10	12	<i>A</i>	31	11	1010
11	13	<i>B</i>	32	12	1011
12	14	<i>C</i>	40	13	1100
13	15	<i>D</i>	41	14	1101
14	16	<i>E</i>	42	15	1110
15	17	<i>F</i>	50	16	1111
16	20	10	51	17	10000
17	21	11	52	18	10001
18	22	12	60	20	10010
19	23	13	61	21	10011
20	24	14	62	22	10100
21	25	15	70	23	10101
22	26	16	71	24	10110
23	27	17	72	25	10111
24	30	18	80	26	11000
25	31	19	81	27	11001
26	32	1 <i>A</i>	82	28	11010
27	33	1 <i>B</i>	90	30	11011
28	34	1 <i>C</i>	91	31	11100

1.3. CONVERSIÓN ENTRE BASES

1.3.1. Conversión de cualquier base a base decimal

Para obtener el equivalente de una cifra en cualquier base r , o $(N)_r$, en base 10, o $(N)_{10}$, se expresa la cifra en notación polinomial y se lee el resultado del polinomio en base 10. En general:

$$(N)_{10} = (N)_r = \left(\sum_{i=M-1}^{-L} D_i * r^i \right)_{10}$$

EJEMPLOS

1. Encontrar $(N)_{10}$ si $(N)_8 = (265)_8$

$$(N)_{10} = (2 * 8^2 + 6 * 8^1 + 5 * 8^0)_{10}$$

$$(N)_{10} = (2 * 64 + 6 * 8 + 5)_{10}$$

$$(N)_{10} = (181)_{10}$$

$$(265)_8 = (181)_{10}$$

2. Encontrar $(N)_{10}$ si $(N)_{16} = (4FB)_{16}$

$$(N)_{10} = (4 * 16^2 + F * 16^1 + B * 16^0)_{10}$$

$$(N)_{10} = (4 * 16^2 + 15 * 16^1 + 11 * 16^0)_{10}$$

$$(N)_{10} = (4 * 256 + 15 * 16 + 11)_{10}$$

$$(N)_{10} = (1275)_{10}$$

$$(4FB)_{16} = (1275)_{10}$$

3. Encontrar $(N)_{10}$ si $(N) = (101011)_2$

$$(N)_{10} = (1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0)_{10}$$

$$(N)_{10} = (1 * 32 + 1 * 8 + 1 * 2 + 1)_{10}$$

$$(N)_{10} = (43)_{10}$$

$$(101011)_2 = (43)_{10}$$

4. Encontrar $(N)_{10}$ si $(N) = (302)_5$

$$(N)_{10} = (3 * 5^2 + 0 * 5^1 + 2 * 5^0)_{10}$$

$$(N)_{10} = (3 * 25 + 0 * 5 + 2 * 1)_{10}$$

$$(N)_{10} = (75 + 0 + 2)_{10}$$

$$(N)_{10} = (77)_{10}$$

$$(302)_5 = (77)_{10}$$

5. Encontrar $(N)_{10}$ si $N = (110, 11)_2$

$$(N)_{10} = (1 * 2^2 + 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1} + 1 * 2^{-2})_{10}$$

$$(N)_{10} = (4 + 2 + 0, 5 + 0, 25)_{10}$$

$$(N)_{10} = (6, 75)_{10}$$

$$(110, 11)_2 = (6, 75)_{10}$$

6. Encontrar $(N)_{10}$ si $N = (212, 012)_3$

$$(N)_{10} = (2 * 3^2 + 1 * 3^1 + 2 * 3^0 + 0 * 3^{-1} + 1 * 3^{-2} + 2 * 3^{-3})_{10}$$

$$(N)_{10} = (18 + 3 + 2 + 0, 11 + 0, 074)_{10}$$

$$(N)_{10} = (23, 184)_{10}$$

$$(212, 012)_3 = (23, 184)_{10}$$

1.3.2. Conversión de base decimal a cualquier base

CONVERSIÓN DE PARTE ENTERA

Las propiedades del sistema de numeración decimal permiten convertir cualquier cantidad entera a otra base, aplicando lo que se conoce como el **método de las divisiones sucesivas**, donde el número que se desea convertir es dividido sucesivamente por la base a la que quiere ser llevado, hasta que se obtenga como resultado de la división un cero. El entero en la nueva base se logra de leer todos los residuos desde el último obtenido (MSD) hacia el primero de ellos (LSD).

EJEMPLOS

1. Encontrar $(N)_2$ si $(N)_{10} = 19$

$$\begin{array}{r}
 19 \overline{)2} \\
 \uparrow \\
 1 \quad 9 \overline{)2} \\
 \uparrow \\
 \text{LSD} \quad 1 \quad 4 \overline{)2} \\
 \quad \quad 0 \quad 2 \overline{)2} \\
 \quad \quad \quad 0 \quad 1 \overline{)2} \\
 \text{MSD} \longrightarrow 1 \quad 0
 \end{array}$$

$$(19)_{10} = (10011)_2$$

2. Encontrar $(N)_8$ si $(N)_{10} = 250$

$$\begin{array}{r}
 250 \overline{)8} \\
 \text{LSD} \longrightarrow 2 \quad 31 \overline{)8} \\
 \quad \quad 7 \quad 3 \overline{)8} \\
 \text{MSD} \longrightarrow 3 \quad 0
 \end{array}$$

$$(250)_{10} = (372)_8$$

3. Encontrar $(N)_{16}$ si $(N)_{10} = 1357$

$$\begin{array}{r}
 1357 \overline{)16} \\
 \text{LSD} \longrightarrow 13 \quad 84 \overline{)16} \\
 \quad \quad 4 \quad 5 \overline{)16} \\
 \text{MSD} \longrightarrow 5 \quad 0
 \end{array}
 \quad (13)_{10} = (D)_{16}$$

$$(1357)_{10} = (54D)_{16}$$

CONVERSIÓN DE PARTE FRACCIONARIA

La parte fraccionaria de un número se obtiene aplicando el **método de las multiplicaciones sucesivas**, que consiste en multiplicar la fracción inicial por la nueva base, y sucesivamente multiplicar la fracción del nuevo producto obtenido por la nueva base, hasta lograr como producto ceros o alcanzar una cantidad de cifras significativas determinadas. La fracción en la nueva base se logra de leer las partes enteras de cada multiplicación desde la primera obtenida (MSD) hacia la última de ellas (LSD). Al cambiar de base, es posible que una fracción exacta en una base no lo sea en la nueva, o una fracción no exacta ahora sea exacta; por ello el método de las multiplicaciones sucesivas podría no tener fin, y en ocasiones se ejecuta hasta obtener una cierta cantidad de cifras significativas.

EJEMPLOS

1. Encontrar $(N)_2$ si $(N)_{10} = 0,1875$

$$\begin{array}{rcl}
 & \text{MSD} & \\
 & \downarrow & \\
 0,1875 * 2 = & \underline{0},375 & \\
 0,375 * 2 = & 0,75 & \\
 0,75 * 2 = & 1,5 & \\
 0,5 * 2 = & \underline{1},0 & \\
 & \uparrow & \\
 & \text{LSD} &
 \end{array}
 \quad (0,1875)_{10} = (0,0011)_2$$

2. Encontrar $(N)_8$ si $(N)_{10} = 0,824$

$$\begin{array}{rcl}
 & \text{MSD} & \\
 & \downarrow & \\
 0,824 * 8 = & \underline{6},592 & \\
 0,592 * 8 = & 4,736 & \\
 0,736 * 8 = & 5,888 & \\
 0,888 * 8 = & 7,104 & \\
 0,104 * 8 = & 0,832 & \\
 0,832 * 8 = & \underline{6},656 & \\
 \vdots & & \\
 & \uparrow & \\
 & \text{LSD} &
 \end{array}
 \quad (0,824)_{10} = (0,645706\cdots)_8$$

CONVERSIÓN GENERAL

En general, para convertir una cantidad con parte entera y fraccionaria de la base 10 a otra distinta, se deben separar las partes, aplicar el método respectivo a cada una de ellas y finalmente combinar los resultados.

CONVERSIÓN POR ESTRUCTURA DE PESOS

Es posible convertir números decimales a otras bases teniendo en cuenta la estructura de los pesos o la misma representación posicional. Este método es muy útil para pasar de decimal a binario números relativamente pequeños. Se debe recordar que el máximo número en base b con M dígitos está dado por la expresión $(N)_{b_{MAX}} = b^M - 1$, de la cual se puede encontrar el número de dígitos necesarios para una cantidad dada.

EJEMPLOS

1. Encontrar $(N)_2$ si $(N)_{10} = 27$

Para el decimal 27 en base 2 se debe tener $M = 5$, ya que $(N)_{b_{MAX}} = (2^5 - 1 = 31)$ y $(N)_{b_{MAX}} = (2^4 - 1 = 15)$; es decir, en binario se requiere de cinco dígitos:

$$\begin{array}{cccccc}
 \underbrace{0}_{2^5=32} & \underbrace{1}_{2^4=16} & \underbrace{1}_{2^3=8} & \underbrace{0}_{2^2=4} & \underbrace{1}_{2^1=2} & \underbrace{1}_{2^0=1} \\
 (27)_{10} = (11011)_2
 \end{array}$$

2. Encontrar $(N)_2$ si $(N)_{10} = 105$

Para el decimal 105 se debe tener $M = 7$, ya que $(N)_{b_{MAX}} = (2^7 - 1 = 127)$; es decir, en binario se requiere de siete dígitos:

$$\begin{array}{ccccccc} \underbrace{1}_{2^6=64} & \underbrace{1}_{2^5=32} & \underbrace{0}_{2^4=16} & \underbrace{1}_{2^3=8} & \underbrace{0}_{2^2=4} & \underbrace{0}_{2^1=2} & \underbrace{1}_{2^0=1} \end{array}$$

$$(105)_{10} = (1101001)_2$$

1.3.3. Conversión de cualquier base a cualquier base

En general, para pasar de una base cualquiera a otra cualquiera, primero se pasa la base inicial a base decimal y finalmente se pasa la base decimal a la segunda base deseada.

EJEMPLOS

1. Encontrar $(N)_5$ si $(N)_{13} = 81,32$

$$(N)_{10} = (8 * 13^1 + 8 * 13^0 + 3 * 13^{-1} + 2 * 13^{-2})_{10}$$

$$(N)_{10} = (104 + 8 + 0,230769 + 0,011834)_{10}$$

$$(N)_{10} = (112,242603)_{10}$$

$$\begin{array}{r} 112 \overline{)5} \\ \text{LSD} \rightarrow 2 \quad 22 \overline{)5} \\ \quad \quad 2 \quad 4 \overline{)5} \\ \text{MSD} \longrightarrow 4 \quad 0 \end{array}$$

$$\begin{array}{c} \text{MSD} \\ \downarrow \\ 0.242603 * 5 = \textcircled{1}213015 \\ 0.213015 * 5 = 1.065075 \\ 0.065075 * 5 = 0.325375 \\ 0.325375 * 5 = 1.626875 \\ 0.626875 * 5 = \textcircled{3}134375 \\ \uparrow \\ \text{LSD} \end{array}$$

$$(81,32)_{13} = (422,11013...)_{5}$$

2. Encontrar $(N)_9$ si $(N)_3 = 201$

$$(N)_{10} = 2 * 3^2 + 1 * 3^0$$

$$(N)_{10} = 18 + 1$$

$$(N)_{10} = (19)_{10}$$

$$\begin{array}{rcl} & 19 & \overline{)9} \\ \text{LSD} \rightarrow & 1 & 2 \quad \overline{)9} \\ \text{MSD} \longrightarrow & 2 & 0 \end{array}$$

$$(201)_3 = (21)_9$$

1.3.4. Conversión entre bases A y B cuando $B = A^k$

Existen bases A y B que se relacionan por $B = A^k$, donde $k = 2, 3, 4, \dots$. Un ejemplo son las bases 2 y 8, donde $8 = 2^3$. Este método se puede aplicar a cifras con parte entera y fracción de forma simultánea.

CONVERSIÓN DE LA BASE A HACIA LA BASE B

En este caso, se convierte desde la base menor hacia la base mayor. Para lo anterior, se inicia desde el separador de partes entera y fraccionaria, y hacia derecha e izquierda, se realizan agrupaciones de k dígitos; las agrupaciones en los extremos a derecha e izquierda deben poseer k dígitos, para lo cual en caso de ser necesario se debe completar con ceros. Finalmente, cada grupo de k dígitos en base A se escribe como su equivalente de un dígito en base B.

EJEMPLOS

1. Encontrar $(N)_9$ si $(N)_3 = 201,22101$

Para este caso $9 = 3^2$, por lo que $k = 2$:

$$\underbrace{02}_2, \underbrace{01}_1, \underbrace{22}_8, \underbrace{10}_3, \underbrace{10}_3$$

$$(201,22101)_3 = (21,833)_9$$

De la tabla 1.1, se puede verificar fácilmente que: $(22)_3 = (8)_9$ y $(10)_3 = (3)_9$

2. Encontrar $(N)_8$ si $(N)_2 = 1011101,11001101$

Para este caso $8 = 2^3$, por lo que $k = 3$:

$$\underbrace{001}_1, \underbrace{011}_3, \underbrace{101}_5, \underbrace{110}_6, \underbrace{011}_3, \underbrace{010}_2$$

$$(1011101,11001101)_2 = (135,632)_8$$

3. Encontrar $(N)_{16}$ si $(N)_2 = 1011101,11001101$

Para este caso $16 = 2^4$, por lo que $k = 4$:

$$\underbrace{0101}_5, \underbrace{1101}_D, \underbrace{1100}_C, \underbrace{1101}_D$$

$$(1011101,11001101)_2 = (5D.CD)_{16}$$

De la tabla 1.1, se puede verificar fácilmente que: $(1101)_2 = (D)_{16}$ y $(1100)_2 = (C)_{16}$

CONVERSIÓN DE LA BASE B HACIA LA BASE A

En este caso, se convierte desde la base mayor hacia la base menor. Para lo anterior, se inicia en el separador de partes entera y fraccionaria, y hacia derecha e izquierda, cada dígito en base B se escribe por su equivalente de k dígitos en base A .

EJEMPLOS

1. Encontrar $(N)_2$ si $(N)_{16} = CA,2D$

Para este caso $16 = 2^4$, por lo que $k = 4$:

$$\underbrace{C}_{1100}, \underbrace{A}_{1010}, \underbrace{2}_{0010}, \underbrace{D}_{1101}$$

$$(CA,2D)_{16} = (11001010,00101101)_2$$

2. Encontrar $(N)_2$ si $(N)_8 = 1546,72$

Para este caso $8 = 2^3$, por lo que $k = 3$:

$$\underbrace{1}_{001}, \underbrace{5}_{101}, \underbrace{4}_{100}, \underbrace{6}_{110}, \underbrace{7}_{111}, \underbrace{2}_{010}$$

$$(CA,2D)_{16} = (1101100110,11101)_2$$

Se debe observar que los ceros sobrantes, a derecha e izquierda, se pueden omitir.

3. Encontrar $(N)_2$ y $(N)_8$ si $(N)_{16} = 38F,1C$

Inicialmente se pasa la base 16 a base 2:

$$\underbrace{3}_{0011}, \underbrace{8}_{1000}, \underbrace{F}_{1111}, \underbrace{1}_{0001}, \underbrace{C}_{1100}$$

$$(38F,1C)_{16} = (1110001111,000111)_2$$

Luego se pasa la base 2 a base 8:

$$\underbrace{001}_1, \underbrace{110}_6, \underbrace{001}_1, \underbrace{111}_7, \underbrace{000}_0, \underbrace{111}_7$$

$$(38F,1C)_{16} = (1110001111,000111)_2 = (1617,07)_8$$

1.4. AGRUPACIONES BINARIAS

En el sistema de numeración binario es común asignar nombres a grupos de dígitos que, según su cantidad, permiten representar números hasta un límite dado. Los grupos más comunes, y su capacidad de representación, son los que se muestran en la tabla 1.2:

Tabla 1.2: Agrupaciones binarias y sus límites de representación		
Grupo	Nombre	Capacidad
1	Bit	0, 1
4	Nibble	$([0 \dots 15])_{10} = ([0 \dots F])_{16}$
8	Byte	$([0 \dots 255])_{10} = ([0 \dots FF])_{16}$
16	Word	$([0 \dots 65535])_{10} = ([0 \dots FFFF])_{16}$
32	Doble Word, DWord	$([0 \dots FFFFFFFF])_{16}$
64	Quad Word, QWord	$([0 \dots FFFFFFFFFFFFFFFF])_{16}$
128	Doble Quad Word, DQWord	$([0 \dots FFFFFFFFFFFFFFFFFFFFFFFF])_{16}$

1.5. REPRESENTACIÓN DE NÚMEROS CON SIGNO

En el sistema de numeración decimal, al realizar operaciones de números con signo es frecuente ajustar el signo de una cifra de acuerdo con la operación a realizar o a conveniencia, por ejemplo:

$$-(107)_{10} - (34)_{10} + (15)_{10} = (-107)_{10} - (+34)_{10} + (+15)_{10} = (-107)_{10} + (-34)_{10} - (-15)_{10}$$

En los sistemas de numeración, especialmente el binario, es conveniente encontrar representaciones donde el signo de una cifra pueda ser incluido como parte de la misma cifra. En busca de este objetivo, a continuación se estudian las representaciones con signo más comunes.

1.5.1. Signo-magnitud

Representación donde se agrega un bit S a la izquierda de la magnitud para indicar el signo; para un sistema en base b , se debe agregar:

$$\begin{aligned} S = 0 : & \quad \text{Si la magnitud es positiva} \\ S = b - 1 : & \quad \text{Si la magnitud es negativa} \end{aligned}$$

La representación en signo-magnitud, notada como $(N)_{bsm}$, toma la siguiente forma general:

$$\pm(N)_b = \pm(D_{M-1} \dots D_1 D_0 . D_{-1} \dots D_{-L})_b = (S D_{M-1} \dots D_1 D_0 . D_{-1} \dots D_{-L})_{bsm}$$

Se debe observar que en este tipo de representación, el signo hace parte íntegra de la magnitud.

EJEMPLOS

1. Encontrar $(N)_{10sm}$ si $(N)_{10} = -(19)_{10}$
Para este caso $S = 10 - 1 = 9$:

$$-(19)_{10} = (919)_{10sm}$$

2. Encontrar $(N)_{5sm}$ si $(N)_5 = -(314)_5$
Para este caso $S = 5 - 1 = 4$:

$$-(314C)_5 = (4314)_{5sm}$$

3. Encontrar $(N)_{2sm}$ si $(N)_2 = \pm(110110)_2$
Para el caso negativo $S = 2 - 1 = 1$, y para el caso positivo $S = 0$:

$$\begin{aligned} + (110110)_2 &= (0110110)_{2sm} \\ - (110110)_2 &= (1110110)_{2sm} \end{aligned}$$

4. Encontrar $(N)_{2sm}$ si $(N)_2 = \pm(0)_2$

$$\begin{aligned} + (0)_2 &= (00)_{2sm} \\ - (0)_2 &= (10)_{2sm} \end{aligned}$$

En base 2 y en representación signo-magnitud toda cantidad negativa inicia con un bit 1 y toda cantidad positiva con un bit 0. Como caso especial está el cero, que en cualquier base posee dos representaciones, toda vez que $+(0)_b = -(0)_b = (00)_{2sm} = (10)_{2sm} = (00)_{bsm} = (\underbrace{b-1\ 0})_{bsm}$.

1.5.2. Introducción a sistemas por complemento

Los métodos de representación de números con signo, buscan facilitar los procedimientos para realizar operaciones aritméticas. En particular, es de interés los procedimientos cuando la base del sistema es la binaria.

Para explorar el fundamento de los métodos basados en el complemento, se debe observar la figura 1.3-1, donde se muestra de forma vectorial la operación aritmética de resta $45 - 20 = 25$. En la operación anterior, a 45 se le denomina el minuendo (M), a 20 el sustraendo (S) y a 25 la resta (R), dando la forma general conocida para la resta de $M - S = R$. Ahora, en la figura 1.3-2, se adiciona el vector A, tal que $A + S + R = 100$, o $A + R = 100 - S$. En este contexto, $A + R$ es lo que le falta al valor absoluto del sustraendo (S) para ser igual a 100, o en otras palabras, $A + R$ es el complemento a 100 de S. Finalmente, en la figura 1.3-3, se observa que si se suman los vectores $M + A + R$, o $M + \text{complemento a 100 de S}$, se obtiene como resultado $100 + R$. Por tanto, $R = M + (A + R) - 100$, o $R = M + (100 - S) - 100$, que lleva a la igualdad inicial de $R = M - S$. De todo lo descrito, se puede concluir que en lugar de restar S en $R = M - S$, se puede sumar el complemento a 100 de S y luego restar 100 ($R = M + (100 - S) - 100$), o que, en cierta forma, el complemento a 100 de S (como cifra positiva) se comporta como si fuera S (la cifra negativa) más 100.

Una anotación importante para los sistemas de representación basados en el complemento, es que el vector $A + R$ debe ser mínimo el complemento a la cantidad que involucre los mínimos bits necesarios para representar a S ; o si se tienen M dígitos en la base b , el complemento se calcula mínimo a b^M . Para el caso en estudio, como el sustraendo posee dos dígitos, el complemento es calcula como mínimo a $10^2 = 100$. Entonces, generalizando $R = M + (b^M - S) - b^M$.

Figura 1.3: Fundamento de las representaciones por complemento

1.5.3. Complemento disminuido a la base

De la relación $R = M + (100 - S) - 100$, obtenida en la sección 1.5.2, es fácilmente comprobable que $R = M + (99 - S) - 100 + 1$, en la cual al complemento se le resta uno, o se dice que es el complemento disminuido a la base. Una resta general en complemento disminuido a la base es $R = M + (b^M - 1 - S) - b^M + 1$. Esta última relación es muy importante, toda vez que permite realizar restas que no involucren prestamos (o hacer restar sin llevar). Esto se logra ya que la resta $99 - S$, o en general $b^M - 1 - S$, es una operación de resta en la cual cada dígito del sustraendo es menor o igual al máximo dígito de la base en cuestión.

A continuación, se muestra un procedimiento de resta bajo los parámetros descritos, usando la base decimal:

Minuendo $\Rightarrow$	536	$b^M - 1 = 1000 - 1 \Rightarrow$	999
Sustraendo $\Rightarrow$	-372	Sustraendo $\Rightarrow$	-372
Resultado $\Rightarrow$	$\overline{164}$	Compl. disminuido a la base $\Rightarrow$	$\overline{627}$

Se observa que, la resta del lado derecho no involucra la necesidad de préstamos; en este caso se dice que 627 es el complemento disminuido a la base del número -372 , por lo que:

$$-(372)_{10} = [627]_{10-1}$$

Ahora la operación de resta se puede realizar de la siguiente forma $(536)_{10} - (372)_{10} = (536)_{10} + [627]_{10-1} - 1000 + 1$:

$$\begin{array}{rcl} \text{Minuendo} & \Rightarrow & 536 \\ \text{Sustraendo} & \Rightarrow & +627 \\ \text{Resultado parcial} & \Rightarrow & \overline{+1163} \\ & & -1000 \\ & & + \quad 1 \\ & & \hline & & 164 \end{array}$$

Otra forma de ver el resultado final, es que se obtiene de sumar el dígito adicional (acarreo) del resultado parcial al resto de la cifra, es decir: Resultado = $163 + 1 = 164$.

En la representación complemento disminuido a la base, las cifras positivas se representan igual que en signo-magnitud, pero la representación negativa es el complemento disminuido a la base. La notación es:

$$\begin{aligned} -(N)_b &= [N]_{b-1} = [b^M - 1 - (N)_b]_{b-1} \\ +(N)_b &= [N]_{b-1} = [0N]_{b-1} \end{aligned}$$

EJEMPLOS

1. Encontrar $[N]_{10-1}$ si $(N)_{10} = (3174)_{10}$
En este caso la cifra es positiva, por lo que:

$$(3174)_{10} = [03174]_{10-1}$$

2. Encontrar $[N]_{10-1}$ si $(N)_{10} = -(3174)_{10}$
En este caso la cifra es negativa, por lo que:

$$-(3174)_{10} = [b^M - 1 - (N)_b]_{10-1} = [9999 - 3174]_{10-1} = [7936]_{10-1}$$

3. Encontrar $[N]_{2-1}$ si $(N)_2 = -(11011001)_2$
La operación a realizar es: $[b^M - 1 - (N)_b]_{2-1}$

$$\begin{array}{rcl} b^M - 1 & \Rightarrow & 11111111 \\ (N)_b & \Rightarrow & -11011001 \\ \text{Compl. disminuido} & \Rightarrow & \overline{00100110} \end{array}$$

Para el caso especial de complemento disminuido a la base 2, se debe notar del último de los ejemplos presentados como el resultado se puede obtener de forma directa a partir del mismo número negativo, cambiando cada bit 1 por un 0 y cada bit 0 por un 1. En base 2, el complemento disminuido recibe el nombre de complemento a 1, o C1.

En complemento disminuido también existen dos representaciones para el número cero:

$$Para + (000) = [0000]_{2-1}$$

$$Para - (000) = [111]_{2-1}$$

1.5.4. Complemento a la base

Cuando se realiza una resta en la representación complemento disminuido a la base, hace falta sumar +1 al resultado parcial. La representación de complemento a la base corrige este faltante sumando 1 al complemento disminuido. La representación de complemento a la base se obtiene partiendo inicialmente de la representación signo-magnitud, donde para el caso de cifras negativas el signo se deja fuera de los paréntesis, luego se aplica complemento disminuido a la base y finalmente se suma 1; los números positivos son iguales a la presentación signo-magnitud.

$$\begin{aligned} -(N)_b &= -[0N]_{bsm} = [\{b^M - 1 - (N)_{bsm}\} + 1]_b \\ +(N)_b &= +[0N]_{bsm} = [0N]_b \end{aligned}$$

Es importante resaltar que es necesario realizar las operaciones en la secuencia descrita, es decir, primero se debe hacer $\{b^M - 1 - (N)_{bsm}\}$ y luego sumar 1; si se hace $\{b^M - 1 - (N)_{bsm}\} + 1 = \{b^M - (N)_{bsm}\}$ la resta requiere préstamos, lo cual es exactamente lo que se quiere evitar.

EJEMPLOS

1. Encontrar $[N]_2$ si $(N)_2 = -(11100100)_2 = -(011100100)_{2sm}$

La operación a realizar es: $[\{b^M - 1 - (N)_b\} + 1]_2 = [[N]_{2-1} + 1]_2$

$$\begin{array}{ll} \text{Compl. disminuido} \Rightarrow & 100011011 \\ +1 \Rightarrow & +1 \\ \text{Compl. a la base} \Rightarrow & \overline{100011100} \end{array}$$

2. Encontrar $[N]_2$ si $(N)_2 = -(1010010)_2 = -(01010010)_2$

$$\begin{array}{ll} \text{Compl. disminuido} \Rightarrow & 10101101 \\ +1 \Rightarrow & +1 \\ \text{Compl. a la base} \Rightarrow & \overline{10101110} \end{array}$$

3. Encontrar $[N]_2$ si $(N)_2 = (100110)_2$

Como la cifra es positiva: $(100110)_2 = [0100110]_2$

En base 2 la representación complemento a la base recibe el nombre de complemento a 2, o C2. De los ejemplos anteriores, se puede observar que existe un forma de obtener la representación de complemento a la base sin tener que pasar por el complemento disminuido, cuando $b = 2$. Para ello, se debe tomar el número en su representación signo magnitud y luego del bit LSD al MSD, o de derecha a izquierda, se busca el primer uno. Esto uno, y los bits a su derecha, se escriben iguales, pero se invierten los bits a su izquierda. Esto implica que en complemento a 2, ni siquiera es necesario hacer la resta sin llevar.

EJEMPLO

Encontrar $[N]_2$ si $(N)_2 = -(11100100)_2 = -(011100100)_{2sm}$

De izquierda a derecha se deja sin cambios: 100

Se continua a derecha cambiando: 100011100

La respuesta es: $-(11100100)_2 = [100011100]_2$

Una gran ventaja de la representación complemento a la base es que posee una representación única para el cero:

$$\text{Para } +(000) = +(0000)_{2sm} = [0000]_2$$

$$\text{Para } -(000)_2 = -(0000)_{2sm} = [0000]_2$$

Otra ventaja del complemento a la base es la de ser una representación exacta de números negativos, ya que respeta la estructura de pesos del sistema ponderado de numeración, al aplicar por segunda vez el complemento a la base se obtiene la cifra positiva y al sumar un número positivo con su negativo se obtiene cero:

Para el número $-(54)_{10} = -(110110)_2 = -(0110110)_{2sm}$

-54	-0110110	$-(-54)$	-1001010	-54	1001010
Compl. disminuido	1001001	Compl. disminuido	0110101	+54	+0110110
+1	+1	+1	+1		
Compl. a la base	<u>1001010</u>	Compl. a la base	<u>0110110</u>		<u>0000000</u>

Para la estructura de pesos, el primer bit asigna siempre signo negativo, lo que permite utilizar esta representación aritméticamente.

$$\begin{aligned} -(54)_{10} = [1001010]_2 &= -1 * 2^6 + 1 * 2^3 + 1 * 2^1 \\ &= -64 + 8 + 2 = -54 \end{aligned}$$

Se debe notar que para las cifras positivas el bit de signo es 0 y, por tanto, el signo menos no resta, con el fin de entregar una cantidad positiva.

1.5.5. Representación sesgada

La representación sesgada, o por exceso, se obtiene de:

$$\pm(N)_b = [2^{B-1} - 1 \pm N]_{bx}$$

Para 4 bits la constante $2^{B-1} - 1 = (7)_{10} = (111)_2$, para 8 bits vale $2^{B-1} - 1 = (127)_{10} = (1111111)_2$.

EJEMPLOS

1. Encontrar $[N]_{2x}$ si $(N)_2 = (01010)_2 = (10)_{10}$
Entonces: $(15 + 10)_{10} = [11001]_{2x}$
2. Encontrar $[N]_{2x}$ si $(N)_2 = -(01010)_2 = -(10)_{10}$
Entonces: $(15 - 10)_{10} = [00101]_{2x}$

1.5.6. Comparación de representaciones con signo

En la tabla 1.3 se muestra una comparación entre representaciones con signo para 4 bits. En ella se observa que las representaciones signo-magnitud y complemento a 1 pueden guardar números en el rango desde $[-7 \dots 7]$ pero almacenan dos veces el número cero; la representación complemento a 2 guarda en el rango $[-8 \dots 7]$ y la representación sesgada en el rango $[-7 \dots 8]$.

Tabla 1.3: Comparación de representaciones para 4 bits

Binario estándar	Signo-magnitud	Compl a 1	Compl a 2	Sesgada
0000	+0	+0	+0	-7
0001	+1	+1	+1	-6
0010	+2	+2	+2	-5
0011	+3	+3	+3	-4
0100	+4	+4	+4	-3
0101	+5	+5	+5	-2
0110	+6	+6	+6	-1
0111	+7	+7	+7	0
1000	-0	-7	-8	1
1001	-1	-6	-7	2
1010	-2	-5	-6	3
1011	-3	-4	-5	4
1100	-4	-3	-4	5
1101	-5	-2	-3	6
1110	-6	-1	-2	7
1111	-7	-0	-1	8

1.5.7. Números en punto flotante IEEE Std 754-1985

Esta representación define los formatos en los cuales se guardan los números de punto flotante y valores especiales como Infinito (*Inf*) y No es un número (*NaN*). Este estándar define especialmente tres formatos en 32, 64 y 80 bits, los valores para 32 bits son obligatorios y son los que se muestran a continuación, los valores para los otros formatos son opcionales.

Para cualquier número en formato de punto flotante se distinguen tres partes principales, a saber:

$$N = S \times F \times b^E$$

Signo: (S) representa con un 1 bit el signo de la cifra usando para positivos 0 y negativos 1.

Mantisa: (F) representa la magnitud

Exponente: (E) representa la posición del punto flotante

Por ejemplo, el número decimal 1438007,50 es igual a $0,14380075 \times 10^7$ o a $1,4380075 \times 10^6$. La última representación se denomina número normalizado, donde la mantisa inicia con el dígito más significativo seguido de la coma y a continuación el resto de la magnitud. La representación de número normalizado garantiza una forma estándar para una cifra. Un número normalizado en sistema binario siempre es un 1, por lo que este se omite a la hora de guardar la cantidad.

Para el formato de punto flotante con 32 bits se emplea 1 bit para guardar el signo, 8 bits para guardar el exponente en representación sesgada y los 23 bits restantes para guardar la mantisa normalizada sin el bit más significativo.

S	E	F
1 bit	8 bits	23 bits

Como el exponente se guarda en representación sesgada con 8 bits, la constante $2^{B-1} - 1 = (127)_{10} = (1111111)_2$ y permite guardar exponentes en el rango $[-127 \dots +128]$. El estándar define el uso libre del rango $[-127 \dots +127]$ y deja el +128, que sumando la constante se obtiene $128 + 127 = 255$, para los valores especiales así:

Infinito: Exponente $= (255)_{10} = (11111111)_2$ y con $F = 0$

NaN: Exponente $= (255)_{10} = (11111111)_2$ y con $F \neq 0$

EJEMPLOS

1. Almacenar $N = (5,4372)_{10}$

$$N = (5,4372)_{10} = (101,0110111111101)_2 = (1,010110111111101 \times 2^2)_2$$

A partir del número normalizado se tienen los siguientes datos:

$$S = 0$$

$$E = (2 + 127)_{10} = (129)_{10} = [10000001]_{2x}$$

$$F = 010110111111101$$

0	10000001	01011011111110100000000
---	----------	-------------------------

2. Almacenar $N = -(0,0001101101)_2$

$$N = -(0,0001101101)_2 = -(1,101101 \times 2^{-4})_2$$

A partir del número normalizado se tienen los siguientes datos:

$$S = 1$$

$$E = (-4 + 127)_{10} = (123)_{10} = [01111011]_{2x}$$

$$F = 101101$$

1	01111011	1011010000000000000000
---	----------	------------------------

3. Recuperar el binario almacenado:

1	01101001	1111000100000000000000
---	----------	------------------------

$S = 1$, el número es negativo

$E = [01101001]_{2x} = (105 - 127)_{10} = - (22)_{10}$, se resta 127, ya que al guardar se sumaron

$F = 11110001$

Por tanto el número es $N = - (1, 11110001 \times 2^{-22})$

1.5.8. Rango de los datos

Según se desee tener un número con signo o sin signo, el tipo de representación que se emplee y la cantidad de bits a utilizar, se tienen diferentes rangos de almacenamiento para datos numéricos. Los más comunes son:

Números sin signo (Naturales) $[0 \dots 2^N - 1]$

U8	Unsigned Byte	$[0 \dots 255]$
U16	Unsigned Word	$[0 \dots 65535]$
U32	Unsigned Long	$[0 \dots 4294967295]$

Números con signo (Enteros) $[-2^{N-1} \dots 2^{N-1} - 1]$

I8	Signed Byte	$[-128 \dots 127]$
I16	Signed Word	$[-32768 \dots 32767]$
I32	Signed Long	$[-2147483648 \dots 2147483647]$

Números de punto flotante (Reales)

SGL	32 bits	$Min = \pm 1,4 * 10^{-45}$	$Max = \pm 3,4 * 10^{38}$
DBL	64 bits	$Min = \pm 4,94 * 10^{-324}$	$Max = \pm 1,79 * 10^{308}$
EXT	128 bits	$Min = \pm 6,48 * 10^{-4966}$	$Max = \pm 1,19 * 10^{4932}$

Representación compleja (Complejos)

Se almacena independientemente la parte real y la compleja.

CSG	2*SGL
CDD	2*DBL
CXT	2*EXT

1.6. ARITMÉTICA BINARIA

En las operaciones aritméticas es muy importante escoger inicialmente la representación adecuada de los términos y de la respuesta, ya que en realidad los dispositivos electrónicos que realizan estas operaciones lo hacen con un número de bits fijo. Además, esto simplifica los procedimientos aritméticos.

1.6.1. Suma y resta

Las operaciones de suma y resta se realizan utilizando la presentación de complemento a la base 2, donde las cantidades positivas tienen un formato igual a signo-magnitud y las negativas están en complemento a 2. Al usar esta representación, la respuesta de cada operación también está en complemento a la base 2, lo cual garantiza poder seguir usando un resultado como el inicio de una operación subsiguiente. Las operaciones de suma y resta se pueden resumir en los siguientes casos:

SUMA DE DOS POSITIVOS

Las dos cantidades a sumar son positivas, el resultado debe ser positivo. A modo de ejemplo se presenta la operación $57 + 36$, usando como rango I8. En este caso $(57)_{10} = [00111001]_2$ y $(36)_{10} = [00100100]_2$.

$$\begin{array}{r} +57 \quad 00111001 \\ +36 \quad 00100100 \\ \hline 93 \quad 01011101 \end{array}$$

Como el rango es I8, se completa con ceros para tener 8 bits en cada cifra, además el primer cero en cada número indica que la cantidad es positiva, tal como lo pide la representación de complemento a la base, así la respuesta es: $[01011101]_2 = -0 * 2^7 + 1 * 2^6 + 1 * 2^4 + 1 * 2^3 + 1 * 2^2 + 1 * 2^0 = (93)_{10}$.

POSITIVO MENOS NEGATIVO

En este caso a un positivo se le resta un negativo que en valor absoluto es menor que el positivo, y el resultado debe ser positivo. A modo de ejemplo se presenta la operación $57 - 36$, usando como rango I8. En este caso $-(36)_{10} = [11011100]_2$.

$$\begin{array}{r} +57 \quad 00111001 \\ -36 \quad 11011100 \\ \hline 21 \quad 100010101 \end{array}$$

Ahora la cantidad negativa se representa en complemento a 2, y como el rango es I8, de la respuesta solo se toman 8 bits y se desprecia el noveno (acarreo). El bit MSD en 1, de -36 , indica que es una cantidad negativa. La respuesta es: $[00010101]_2 = -0 * 2^7 + 1 * 2^4 + 1 * 2^2 + 1 * 2^0 = (21)_{10}$, donde el bit MSD es 0, indicando que la respuesta es positiva.

NEGATIVO MÁS POSITIVO

En este caso a un positivo se le resta un negativo que en valor absoluto es mayor que el positivo, y el resultado debe ser negativo. A modo de ejemplo se presenta la operación $-57 + 36$, usando como rango I8. En este caso $-(57)_{10} = [11000111]_2$.

$$\begin{array}{r} -57 \quad 11000111 \\ +36 \quad 00100100 \\ \hline -21 \quad 11101011 \end{array}$$

La respuesta es: $[11101011]_2 = -1 * 2^7 + 1 * 2^6 + 1 * 2^5 + 1 * 2^3 + 1 * 2^1 + 1 * 2^0 = (-21)_{10}$, la cual es negativa.

NEGATIVO MÁS NEGATIVO

En este caso a un negativo se le suma otro negativo, el resultado debe ser negativo. A modo de ejemplo se presenta la operación $-57 - 36$, usando como rango I8.

$$\begin{array}{r} -57 \quad 11000111 \\ -36 \quad 11011100 \\ \hline -93 \quad 110100011 \end{array}$$

La respuesta es: $[10100011]_2 = -1 * 2^7 + 1 * 2^5 + 1 * 2^1 + 1 * 2^0 = (-93)_{10}$, la cual es negativa.

1.6.2. Desbordamiento

Ocorre desbordamiento cuando, definido un rango para los datos, al sumar dos positivos o dos negativos el resultado queda fuera del rango empleado. Esto no sucede cuando a un positivo se suma un negativo, o viceversa, ya que en valor absoluto el resultado es menor que el mayor de los operandos. A modo de ejemplo se presentan las operaciones $57 + 94$ y $-54 - 94$; ambas operaciones usando un rango I8 el cual permite operaciones en el intervalo $[-128 \dots 127]$. Para este ejemplo $(57)_{10} = [00111001]_2$, $-(57)_{10} = [11000111]_2$, $(94)_{10} = [01011110]_2$ y $-(94)_{10} = [10100010]_2$.

$$\begin{array}{r} +57 \quad 00111001 \\ +94 \quad 01011110 \\ \hline 151 \quad 10010111 \end{array} \quad \begin{array}{r} -57 \quad 11000111 \\ -94 \quad 10100010 \\ \hline -151 \quad 101101001 \end{array}$$

Cuando se suman los dos positivos, el resultado obtenido es $[10010111]_2 = -1 * 2^7 + 1 * 2^4 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = (-105)_{10}$ lo que difiere de la respuesta correcta que es $+(151)_{10}$; en el caso de la suma de negativos pasa igual, ya que el resultado obtenido es $[01101001]_2 = -0 * 2^7 + 1 * 2^6 + 1 * 2^5 + 1 * 2^3 + 1 * 2^0 = +(105)_{10}$ lo que difiere de la respuesta correcta que es $+(151)_{10}$. En ambos casos ocurren dos situaciones a tener presentes, la primera es que el signo obtenido difiere del correcto, ya que era de esperar que al sumar dos positivos se debe tener como resultado otro positivo y al sumar dos negativos el resultado debe ser igualmente negativo; la segunda situación es que el resultado correcto excede el rango permitido para I8. En definitiva, si ambos operandos inician con el mismo bit, la respuesta debe iniciar igualmente con el mismo bit, de lo contrario la respuesta excede el rango y se debe emplear un rango con más bits, por ejemplo I16 para el caso.

1.6.3. Multiplicación

MÉTODO TRADICIONAL

En el método tradicional se inicia con las cantidades en representación signo-magnitud, luego se realiza la multiplicación con el mismo procedimiento empleado en base decimal, pero sin tener en cuenta los bits de signo. Para el signo, se sabe que el resultado de multiplicar dos cantidades de igual signo arroja una cantidad positiva, y el resultado de multiplicar dos cantidades de diferentes signo arroja una cantidad negativa. Por lo anterior, de forma independiente de la multiplicación, se evalúa el signo del resultado mediante la suma de los bits de signo, sin tener en cuenta acarreos, así:

$$\begin{array}{cccc}
 + \Rightarrow 0 & + \Rightarrow 0 & - \Rightarrow 1 & - \Rightarrow 1 \\
 + \Rightarrow 0 & - \Rightarrow 1 & + \Rightarrow 0 & - \Rightarrow 1 \\
 \hline
 0 & 1 & 1 & 0
 \end{array}$$

EJEMPLOS

1. Encontrar $(10110)_2 \times (01011)_2$

Inicialmente se expresan las cantidades en formato signo-magnitud:

$(010110)_{2sm} \times (001011)_{2sm}$, se realiza la multiplicación sin los signos:

$$\begin{array}{r}
 10110 \\
 \times 01011 \\
 \hline
 10110 \\
 00000 \\
 10110 \\
 00000 \\
 \hline
 11110010
 \end{array}$$

El signo se obtiene de sumar $0 + 0 = 0$, es decir el resultado es positivo, por lo tanto la respuesta es:

$$(011110010)_{2sm} = +(242)_{10}$$

2. Encontrar $(10011)_2 \times -(11011)_2$

Inicialmente se expresan las cantidades en formato signo-magnitud:

$(010011)_{2sm} \times (111011)_{2sm}$, se realiza la multiplicación sin los signos:

$$\begin{array}{r}
 10011 \\
 \times 11011 \\
 \hline
 10011 \\
 00000 \\
 10011 \\
 10011 \\
 \hline
 100000001
 \end{array}$$

El signo se obtiene de sumar $0 + 1 = 1$, es decir el resultado es negativo, por lo tanto la respuesta es:

$$(11000000001)_{2sm} = -(513)_{10}$$

Durante el procedimiento de multiplicación, la respuesta requiere necesariamente un rango con mínimo el doble de bits del rango de los operandos. El procedimiento de multiplicación por el método tradicional no se puede realizar con números en representación de complemento a la base. En los ejemplos vistos, el procedimiento de sumas inicia por el bit menos significativo del multiplicador y con desplazamientos a izquierda de los productos parciales; sin embargo, también se puede realizar con desplazamientos a derecha si las sumas se generan iniciando por el bit más significativo del multiplicador.

EJEMPLO

Encontrar $(10110)_2 \times (01011)_2$

Inicialmente se expresan las cantidades en formato signo-magnitud:

$(010110)_{2sm} \times (001011)_{2sm}$, se realiza la multiplicación sin los signos iniciando por el bit más significativo del multiplicador::

$$\begin{array}{r} 10110 \\ \times 01011 \\ \hline 00000 \\ 10110 \\ 00000 \\ 10110 \\ 10110 \\ \hline 011110010 \end{array}$$

El signo se obtiene de sumar $0 + 0 = 0$, es decir el resultado es positivo, por lo tanto la respuesta es:

$$(011110010)_{2sm} = +(242)_{10}$$

MÉTODO DE LAS SUMAS SUCESIVAS

Otro procedimiento natural de la multiplicación consiste en realizar sumas sucesivas. En este procedimiento, las cantidades deben estar en representación signo-magnitud y los signos se tratan igual que en el método tradicional.

EJEMPLO

Encontrar $-(10110)_2 \times (100)_2$

Inicialmente se expresan las cantidades en formato signo-magnitud:

$(110110)_{2sm} \times (0100)_{2sm}$, se realiza la multiplicación como sumas sucesivas sin los signos:

$$\begin{array}{r} 10110 \quad \text{suma número } (1)_2 \\ 10110 \quad \text{suma número } (10)_2 \\ \hline 101100 \end{array}$$

$$\begin{array}{r} 101100 \\ 10110 \quad \text{suma número } (11)_2 \\ \hline 1000010 \end{array}$$

$$\begin{array}{r} 1000010 \\ 10110 \quad \text{suma número } (100)_2 \\ \hline 1011000 \end{array}$$

El signo se obtiene de sumar $1 + 0 = 1$, es decir el resultado es negativo, por lo tanto la respuesta es:

$$(11011000)_{2sm} = -(88)_{10}$$

Este procedimiento tiene el inconveniente de ser impráctico para cantidades grandes, e igualmente de no tratar con los números en representación de complemento a la base.

MULTIPLICACIÓN POR ALGORITMO DE BOOTH

Procedimiento de multiplicación, desarrollado por Andrew Donald Booth, el cual permite realizar la operación de multiplicación con cantidades en representación de complemento disminuido y además con menos operaciones que los métodos vistos previamente.

Se basa en el principio de poder dividir cualquier cifra binaria como la diferencia de otros dos números binarios: por ejemplo, el número $(0111110)_2$ puede ser reemplazado por $(0111110)_2 = (1000000)_2 - (10)_2$, ahora, si esta cifra es el multiplicador de otra, la cantidad de operaciones en comparación con los métodos previo se reduce ya que solo es necesario hacer dos multiplicaciones y la resta; sin embargo, las multiplicaciones a realizar no son más que procedimientos de corrimiento del multiplicando.

Sea la multiplicación $Q \times R$, donde Q es el multiplicando y R el multiplicador. Entonces esta multiplicación se puede escribir como:

$$Q \times R = R \times \sum_{i=0}^{M_q-1} q_i 2^i = Rq_n 2^n + Rq_{n-1} 2^{n-1} + \dots + Rq_1 2^1 + Rq_0 2^0$$

Si Q es positivo, en representación complemento a la base debe iniciar con cero ($q_n = 0$) y el producto se puede formar de las siguientes sumas:

$$\begin{aligned}
 \blacksquare P^{(0)} &= 0 + Rq_02^0 \\
 \blacksquare P^{(1)} &= P^{(0)} + Rq_12^1 \\
 \blacksquare P^{(2)} &= P^{(1)} + Rq_22^2 \\
 \blacksquare \dots \\
 \blacksquare P^{(n-1)} &= P^{(n-2)} + Rq_{n-1}2^{n-1} \\
 \blacksquare P^{(n)} &= P^{(n-1)} + Rq_n2^n
 \end{aligned}$$

De lo anterior, cada q_i toma valores 0 o 1, si es 0 el término no se suma, si es 1 se realiza la suma; $R2^i$ no es más que el multiplicador desplazado i posiciones a la izquierda.

Pero, como Q en general puede ser negativo, se puede escribir como:

$$Q = -q_n2^n + q_{n-1}2^{n-1} + \dots + q_12^1 + q_02^0$$

lo que equivale a:

$$Q = - (q_n2^n + q_{n-1}2^{n-1} + \dots + q_12^1 + q_02^0) + 2 (q_{n-1}2^{n-1} + \dots + q_12^1 + q_02^0)$$

$$Q = - (q_n2^n + q_{n-1}2^{n-1} + \dots + q_12^1 + q_02^0) + (q_{n-1}2^n + \dots + q_12^2 + q_02^1)$$

Ahora, se procede a agrupar los factores con iguales potencias de 2:

$$Q = (q_{n-1} - q_n) 2^n + (q_{n-2} - q_{n-1}) 2^{n-1} + \dots + (q_0 - q_1) 2^1 + (0 - q_0) 2^0$$

Entonces, el producto $P = Q \times R$ se puede expresar como las sumas:

$$\begin{aligned}
 \blacksquare P^{(0)} &= 0 + R (0 - q_0) \\
 \blacksquare P^{(1)} &= P^{(0)} + R (q_0 - q_1) 2^1 = P^{(0)} / 2 + R (q_0 - q_1) \\
 \blacksquare P^{(2)} &= P^{(0)} / 2 + R (q_1 - q_2) \\
 \blacksquare \dots \\
 \blacksquare P^{(n-1)} &= P^{(n-2)} / 2 + R (q_{n-2} - q_{n-1}) \\
 \blacksquare P^{(n)} &= P^{(n-1)} / 2 + R (q_{n-1} - q_n)
 \end{aligned}$$

De lo anterior, se puede concluir lo siguiente:

- Si $q_{i-1} = q_i$ entonces no es necesario realizar la operación de suma y se desplaza el producto un bit a la derecha.
- Si $q_{i-1} = 1$ y $q_i = 0$ entonces se suma R al producto parcial $P(i)$ y se desplaza un bit a la derecha.
- Si $q_{i-1} = 0$ y $q_i = 1$ entonces se resta R al producto parcial $P(i)$ y se desplaza un bit a la derecha.

Lo anterior da origen al algoritmo de Booth, que se puede resumir de la siguiente forma:

1. Sean Q y R el multiplicando y el multiplicador respectivamente, M_q y M_r los números de bits de cada uno, A y S valores iniciales del algoritmo y P el valor inicial del producto.
2. Se crea la cantidad inicial A con $(M_q + M_r + 1)$ bits, en los primeros q bits se ubica el valor de Q , los restantes bits se llenan con ceros. El valor inicial va en complemento a la base, es decir, en signo-magnitud si es positivo o complemento a 2 si es negativo.
3. Se crea la cantidad inicial S con $(M_q + M_r + 1)$ bits, en los primeros q bits se ubica el valor de $-Q$ en representación de complemento a la base, los restantes bits se llenan con ceros.
4. Se crea la cantidad inicial P con $(M_q + M_r + 1)$ bits, los primeros q bits se llenan con ceros, en los siguientes r bits se ubica R y en bit final se pone cero.
5. Iterar M_r veces sobre:
 - a) Determinar nuevo valor de P así:
 - 1) Si los dos últimos bits (los menos significativos) de P son 00 o 11, se conserva el mismo P actual
 - 2) Si los dos últimos bits de P son 01 $P = P + A$ ignorando acarreos
 - 3) Si los dos últimos bits de P son 10 $P = P + S$ ignorando acarreos
 - b) Hacer desplazamiento aritmético de P a la derecha (en el desplazamiento aritmético a derecha se conserva el bit más significativo)
6. Luego de terminar las iteraciones, se retira el bit menos significativo de P para obtener la respuesta deseada en complemento a la base.

EJEMPLOS

1. Encontrar $(10110)_2 \times -(100)_2$

$$a) \quad Q = (010110)_{2sm}, \quad Q = [101010]_2, \quad M_q = 6, \quad R = -(0100)_{2sm} = [1100]_2, \quad M_r = 4, \\ M_q + M_r + 1 = 11$$

- b) $A = 010110\ 0000\ 0$
- c) $S = 101010\ 0000\ 0$
- d) $P = 000000\ 1100\ 0$
- e) Se itera por $M_r = 4$ veces
- $P = 000000\ 1100\ 0$, los dos últimos bits son 00
Desplazamiento a derecha de P : $P = 000000\ 0110\ 0$
 - $P = 000000\ 0110\ 0$, los dos últimos bits son 00
Desplazamiento a derecha de P : $P = 000000\ 0011\ 0$
 - $P = 000000\ 0011\ 0$, los dos últimos bits son 10
 $P = P + S = 000000\ 0011\ 0 + 101010\ 0000\ 0 = 101010\ 0011\ 0$
Desplazamiento a derecha de P : $P = 110101\ 0001\ 1$
 - $P = 110101\ 0001\ 1$, los dos últimos bits son 11
Desplazamiento a derecha de P : $P = 111010\ 1000\ 1$
- f) $P = 1110101000 = -1 * 2^9 + 1 * 2^8 + 1 * 2^7 + 1 * 2^5 + 1 * 2^3 = -(88)_{10}$. Esta operación se encontró haciendo únicamente una suma.
2. Encontrar $-(101)_2 \times -(10101)_2$
- a) $Q = -(0101)_{2sm}$, $Q = [1011]_2$, $M_q = 4$, $R = -(010101)_{2sm} = [101011]_2$, $M_r = 6$,
 $M_q + M_r + 1 = 11$
- b) $A = 1011\ 000000\ 0$
- c) $S = 0101\ 000000\ 0$
- d) $P = 0000\ 101011\ 0$
- e) Se itera por $M_r = 6$ veces
- $P = 0000\ 101011\ 0$, los dos últimos bits son 10
 $P = P + S = 0000\ 101011\ 0 + 0101\ 000000\ 0 = 0101\ 101011\ 0$
Desplazamiento a derecha de P : $P = 0010\ 110101\ 1$
 - $P = P = 0010\ 110101\ 1$, los dos últimos bits son 11
Desplazamiento a derecha de P : $P = P = 0001\ 011010\ 1$
 - $P = 0001\ 011010\ 1$, los dos últimos bits son 01
 $P = P + A = 0001\ 011010\ 1 + 1011\ 000000\ 0 = 1100\ 011010\ 1$
Desplazamiento a derecha de P : $P = 1110\ 001101\ 0$
 - $P = 1110\ 001101\ 0$, los dos últimos bits son 10
 $P = P + S = 1110\ 001101\ 0 + 0101\ 000000\ 0 = 0011\ 001101\ 0$
Desplazamiento a derecha de P : $P = 0001\ 100110\ 1$
 - $P = 0001\ 100110\ 1$, los dos últimos bits son 01
 $P = P + A = 0001\ 100110\ 1 + 1011\ 000000\ 0 = 1100\ 100110\ 1$
Desplazamiento a derecha de P : $P = 1110\ 010011\ 0$
 - $P = 1110\ 010011\ 0$, los dos últimos bits son 10
 $P = P + S = 1110\ 010011\ 0 + 0101\ 000000\ 0 = 0011\ 010011\ 0$
Desplazamiento a derecha de P : $P = 0001\ 101001\ 1$

f) $P = 0001101001 = 1 * 2^6 + 1 * 2^5 + 1 * 2^3 + 1 * 2^0 = (105)_{10}$. Esta operación necesitó 5 sumas, sin embargo si se invierten el multiplicando y el multiplicador se puede realizar en menos pasos, tal como se muestra a continuación.

3. Encontrar $-(10101)_2 \times -(101)_2$

a) $Q = -(010101)_{2sm}$, $Q = [101011]_2$, $M_q = 6$, $R = -(0101)_{2sm} = [1011]_2$, $M_r = 4$,
 $M_q + M_r + 1 = 11$

b) $A = 101011\ 0000\ 0$

c) $S = 010101\ 0000\ 0$

d) $P = 000000\ 1011\ 0$

e) Se itera por $M_r = 4$ veces

- $P = 000000\ 1011\ 0$, los dos últimos bits son 10
 $P = P + S = 000000\ 1011\ 0 + 010101\ 0000\ 0 = 010101\ 1011\ 0$
Desplazamiento a derecha de P : $P = 001010\ 1101\ 1$
- $P = P = 001010\ 1101\ 1$, los dos últimos bits son 11
Desplazamiento a derecha de P : $P = P = 000101\ 0110\ 1$
- $P = 000101\ 0110\ 1$, los dos últimos bits son 01
 $P = P + A = 101011\ 0000\ 0 + 000101\ 0110\ 1 = 110000\ 0110\ 1$
Desplazamiento a derecha de P : $P = 111000\ 0011\ 0$
- $P = 111000\ 0011\ 0$, los dos últimos bits son 10
 $P = P + S = 111000\ 0011\ 0 + 010101\ 0000\ 0 = 001101\ 0011\ 0$
Desplazamiento a derecha de P : $P = 000110\ 1001\ 1$

f) $P = 0001101001 = 1 * 2^6 + 1 * 2^5 + 1 * 2^3 + 1 * 2^0 = (105)_{10}$. Esta operación necesitó 3 sumas.

De los ejemplos anteriores se observa que, una posible mejora del algoritmo de Booth consiste en identificar qué operando debe ir como multiplicando y cual como multiplicador, para ello basta con agregar un cero a la derecha de los operandos e identificar de izquierda a derecha el que tenga menos transiciones de 1 a 0 o de 0 a 1, ya que en estos casos es donde se debe sumar a P ya sea con A o con S . El operando con menos transiciones debe ser el multiplicador.

1.6.4. División

La operación de división se puede expresar de la siguiente forma:

$$R/Q = C + Residuo/Q$$

Donde R es el dividendo, Q es el divisor y C es el cociente.

MÉTODO DE LAS RESTAS SUCESIVAS

Este método consiste en restar sucesivamente el divisor del dividendo hasta que la resta sea un número menor que el divisor. El cociente será igual al número de iteraciones realizadas y el residuo será el resultado de la última resta. Como el residuo es una fracción del dividendo expresado como $Residuo/Q$, también podría ser aproximado mediante otro procedimiento de división sobre fracciones. Los signos se tratan igual que en el método de sumas sucesivas de la multiplicación y las restas se realizan con el divisor en complemento a la base.

EJEMPLO

Encontrar $-(10011)_2 / (00101)_2$

Inicialmente se expresan las cantidades en formato signo-magnitud usando igual cantidad de bits:

$(110011)_{2sm} / (00101)_{2sm}$, se realiza la división como restas sucesivas sin los signos:

10011	dividendo
11011	menos divisor
<u>01110</u>	resta sin acarreo
	iteración $(01)_2$
01110	
11011	
<u>01001</u>	resta sin acarreo
	iteración $(10)_2$
01001	
11011	
<u>00100</u>	iteración $(11)_2$

Se itera por $(11)_2 = (3)_{10}$ ya que el residuo 00100 es menor que el divisor. El signo se obtiene de sumar $1 + 0 = 1$, es decir el resultado es negativo, por lo tanto la respuesta es:

$$-(10011)_2 / (00101)_2 = -\{(11)_2 + (100)_2 / (101)_2\}$$

La división $Residuo/Q = (100)_2 / (101)_2$ también se puede realizar, sin embargo como es una fracción propia se debe agregar un cero a derecha, por lo que la división ahora es: $(01000)_2 / (00101)_2$:

01000	
11011	
<u>00011</u>	iteración $(01)_2$

Se itera por $(01)_2 = (1)_{10}$ ya que el residuo 00011 es menor que el divisor. La respuesta de esta división es:

$$(01000)_2 / (00101)_2 = (1)_2 + (11)_2 / (101)_2$$

La división $Residuo/Q = (11)_2 / (101)_2$ también se puede realizar, sin embargo como es una fracción propia se debe agregar un cero a derecha, por lo que la división ahora es: $(0110)_2 / (0101)_2$:

$$\begin{array}{r} 0110 \\ 1011 \\ \hline 0001 \end{array} \text{ iteración } (01)_2$$

Se itera por $(01)_2 = (1)_{10}$ ya que el residuo 0001 es menor que el divisor. La respuesta de esta división es:

$$(0110)_2 / (0101)_2 = (1)_2 + (1)_2 / (101)_2$$

Ahora, la división $(1)_2 / (101)_2$ es propia y se debe agregar un cero a la derecha, por lo que queda: $(10)_2 / (101)_2$, esta también es propia y se debe agregar otro cero: $(100)_2 / (101)_2$, pero esta última también es propia y se debe agregar otro cero más: $(1000)_2 / (101)_2$ que fue la primera de las divisiones impropias realizadas.

En definitiva la respuesta planteada inicialmente como:

$$-(10011)_2 / (00101)_2 = -\{(11)_2 + (100)_2 / (101)_2\}$$

se puede expresar de la siguiente forma:

$$-(10011)_2 / (00101)_2 = -(11, 1100\overline{1100})_2$$

El método de las restas sucesivas resulta impráctico en un sistema digital, debido a la gran cantidad de operaciones que se deben realizar. Se desea, entonces, encontrar métodos donde el número de operaciones se reduzca y, por ende, también el tiempo de respuesta en un sistema digital.

MÉTODO TRADICIONAL

En el método tradicional se inicia con las cantidades positivas en representación signo-magnitud, luego se realiza la división con el mismo procedimiento empleado en base decimal y con el divisor en complemento a la base. De forma independiente se evalúa el signo del resultado mediante la suma de los bits de signo sin tener en cuenta acarreo, tal como en la multiplicación. La ventaja de este procedimiento radica en la facilidad de determinar las veces que el divisor está en el dividendo, ya que en base binaria solo se puede estar o no estar, es decir 1 o 0.

EJEMPLO

Encontrar $(111001)_2 / -(101)_2$

Inicialmente se expresan las cantidades en formato signo-magnitud:

$(0111001)_{2sm} \times -(0101)_{2sm}$, se realiza la división tradicional asumiendo dividendo y divisor como positivos:

	Cociente	1011
	0101	$\overline{0111001}$
Restar divisor en Complemento a la base		1011
		$\overline{001000}$
		11011
		$\overline{00111}$
		1011
Residuo que es menor al divisor		$\overline{0010}$

El cociente se determina del bit más significativo al menos significativo así: $C_3 = 1$ ya que la primera resta se pudo realizar, $C_2 = 0$ por que el divisor no está en 100, $C_1 = 1$ por que el divisor está en 1000 y $C_0 = 1$ por que el divisor está en 111. El signo se obtiene de sumar $0 + 1 = 1$, es decir el resultado es negativo, por lo tanto el resultado es:

$$(111001)_2 / -(101)_2 = -\{(1011)_2 + (10)_2 / (101)_2\}$$

También es posible continuar con la expansión de la fracción, tal como se muestra en el método de las restas sucesivas.

MÉTODO DE DIVISIÓN CON RESTAURACIÓN

En general, el dividendo se puede expresar en función del divisor, cociente y residuo como:

$$R = Q \times C + Residuo$$

Las magnitudes de las cantidades R y $Residuo$ no se alteran con los signos de entrada, además los signos son derivables completamente a partir de los signos de R y Q , ya que el signo del residuo siempre es igual al signo del dividendo y el signo del cociente es positivo si dividendo y divisor tiene signo igual o es negativo si tiene signos contrarios. En el método de división con restauración se realiza la resta sucesiva del divisor en complemento a la base, y si la resta es negativa es debido a que el divisor no cabe en el dividendo, por lo que se retoma la resta previa (restauración) y se adiciona un bit. El procedimiento se realiza haciendo corrimiento del dividendo a izquierda.

EJEMPLO

Encontrar $(1110101)_2 / - (1010)_2$

Se parte de cantidades en formato signo-magnitud: $(01110101)_{2sm} / - (01010)_{2sm}$

	Cociente	1011
	01010	$\overline{001110101}$
Inicio $R^{(0)}$		001110101
Corrimiento $2 \times R^{(0)}$		01110101
Restar divisor en $C2 - 2^4 \times Q$		10110
Resultado positivo, cabe $R^{(1)}$	$C_3 = 1$	$\overline{00100101}$
Corrimiento $2 \times R^{(1)}$		0100101
Restar divisor en $C2 - 2^4 \times Q$		10110
Resultado negativo, no cabe	$C_2 = 0$	$\overline{1111101}$
Restauración y del resultado previo $R^{(2)} = 2 \times R^{(1)}$		0100101
Corrimiento $2 \times R^{(2)}$		100101
Restar divisor en $C2 - 2^4 \times Q$		10110
Resultado positivo, cabe $R^{(3)}$	$C_1 = 1$	$\overline{010001}$
Corrimiento $2 \times R^{(3)}$		10001
Restar divisor en $C2 - 2^4 \times Q$		10110
Resultado positivo, cabe $R^{(4)}$	$C_0 = 1$	$\overline{00111}$

Se para proceso al tener un residuo menor que el divisor. El cociente es $C = 1011$. El signo se obtiene de sumar $0 + 1 = 1$, es decir, el resultado es negativo:

$$(1110101)_2 / - (1010)_2 = - \{ (1011)_2 + (111)_2 / (1010)_2 \}$$

MÉTODO DE DIVISIÓN SIN RESTAURACIÓN

En el método de división sin restauración se controla el signo adecuado de una resta con el signo del divisor así: si el resultado de una resta es positivo, entonces el divisor entra en el dividendo; pero si el resultado es negativo no entra y se corrige el signo sumando, y no restando, el divisor. Este método tiene como ventaja sobre el anterior la necesidad de no necesitar la operación adicional para realizar la restauración.

EJEMPLO

Encontrar $(1110101)_2 / -(1010)_2$

Se parte de cantidades en formato signo-magnitud: $(01110101)_{2sm} / -(01010)_{2sm}$

	Cociente	1011
	01010	$\overline{001110101}$
Inicio $R^{(0)}$		001110101
Corrimiento $2 \times R^{(0)}$		01110101
Restar divisor en $C2 - 2^4 \times Q$		10110
Resultado positivo, cabe $R^{(1)}$	$C_3 = 1$	$\overline{00100101}$
Corrimiento $2 \times R^{(1)}$		0100101
Restar divisor en $C2 - 2^4 \times Q$		10110
Resultado negativo, no cabe $R^{(2)}$	$C_2 = 0$	$\overline{1111101}$
Corrimiento $2 \times R^{(2)}$		111101
Corregir, sumando divisor en $C2 + 2^4 \times Q$		01010
Resultado positivo, cabe $R^{(3)}$	$C_1 = 1$	$\overline{010001}$
Corrimiento $2 \times R^{(3)}$		10001
Sumar divisor en $C2 - 2^4 \times Q$		10110
Resultado positivo, cabe $R^{(4)}$	$C_0 = 1$	$\overline{00111}$

El cociente es $C = 1011$. El signo se obtiene de sumar $0 + 1 = 1$, es decir, el resultado es negativo:

$$(1110101)_2 / -(1010)_2 = -\{(1011)_2 + (111)_2 / (1010)_2\}$$

El método de división sin restauración se puede emplear para realizar la división incluyendo los signos de los operandos, donde cada uno debe ir en el formato complemento a la base. En este método con signos, si los signos del residuo y del divisor antes de una resta son iguales entonces el bit del cociente es 1 y se hace una resta, pero si son diferentes el bit del cociente es 0 y se hace una suma. Si al final, el residuo tiene signo diferente del dividendo se debe sumar/restar, según el caso, una vez más el divisor para corregir el signo y se debe sumar/restar 1 del cociente, el cual se obtiene en formato signo-magnitud.

EJEMPLO

Encontrar $(100001)_2 / -(111)_2$
Se parte de cantidades en formato complemento a la base: $(0100001)_{2sm} \times (1001)_{2sm}$

Cociente	1011
11001	$\overline{00100001}$
$R^{(0)}$	000100001
$2 \times R^{(0)}$	00100001
Signos diferentes $+ 2^4 \times Q$ $C_3 = 0$	11001
$R^{(1)}$	$\overline{11101001}$
$2 \times R^{(1)}$	1101001
Signos iguales $- 2^4 \times Q$ $C_2 = 1$	00111
$R^{(2)}$	$\overline{0000101}$
$2 \times R^{(2)}$	000101
Signos diferentes $+ 2^4 \times Q$ $C_1 = 0$	11001
$R^{(3)}$	$\overline{110111}$
$2 \times R^{(3)}$	10111
Signos iguales $- 2^4 \times Q$ $C_0 = 1$	00111
$R^{(4)}$	$\overline{111110}$
	00111
	$\overline{00101}$

El cociente es $C = 0101$, como se realizó corrección restando el divisor al residuo por error en signo, se debe restar 1 del cociente: $C = -(0101 - 1)_2 = -(0100)_{2sm} = [1100]_2$. Así, la respuesta es:

$$(100001)_2 / -(111)_2 = -\{(0100)_2 + (0101)_2 / (111)_2\}$$

1.7. CÓDIGOS BINARIOS

Código:	<i>Sistema de signos y reglas que permite formular y comprender mensajes y cantidades.</i>
----------------	--

En un código binario se busca representar cantidades o caracteres mediante la combinación unívoca de bits. Un código cambia los símbolos, más no el significado, de los elementos de información que representa. El uso frecuente de los códigos binarios está relacionado con el tratamiento de información entre el hombre y la máquina, entre la máquina y el hombre, con el almacenamiento de información, con los procedimientos para guardar y recuperar los datos, así como con el aseguramiento de la integridad de los datos en todos los procesos, entre otros usos.

1.7.1. Códigos alfanuméricos

Se emplean para codificar números, letras y símbolos mediante dígitos binarios. El más importante es el ASCII (*American Standard Code for Information Interchange*).

El código ASCII constó inicialmente de 7 bits (128 caracteres), pero hoy en día es de 8 bits (256 caracteres).

Está dividido en:

Caracteres de control:

32 Desde el $(00)_{16}$ hasta el $(1F)_{16}$

Caracteres de número:

0 es el $(30)_{16}$

9 es el $(39)_{16}$

Alfabeto en mayúsculas (inglés)

A es el $(41)_{16}$

Z es el $(5A)_{16}$

Alfabeto en minúsculas

A es el $(61)_{16}$

Z es el $(7A)_{16}$

Grupo extendido para soporte multilingüe:

Desde $(80)_{16}$ hasta $(FF)_{16}$

1.7.2. Código BCD

El sistema **Decimal Codificado en Binario** (o BCD del inglés *Binary-Coded Decimal*), consiste en representar un número decimal por su equivalente aritmético en binario usando 4 bits. Esta representación es muy útil para procesos simples y para números enteros. La codificación más utilizada es el BCD Natural o BCD8421, pero también se manejan otros sistemas como el BCD2421, el BCD 84-2-1 y el BCD+3.

El código BCD8421 (también BCD natural, o simplemente BCD) coincide con la numeración binaria del dígito decimal relacionado. El código BCD84-2-1 da ponderación de 8, 4, 2, 1 a cada posición binaria desde la más significativa, pero la mitad superior de la tabla inicia en cero y la mitad inferior inicia en uno. El código BCD2421 da ponderación de 2, 4, 2 y 1 a cada posición binaria desde la más significativa, pero la mitad superior de la tabla inicia en cero y la mitad inferior inicia en uno. El código BCD+3 se obtiene directamente del BCD8421 sumando +3. Estos códigos se observan en la tabla 1.4, donde es evidente que los códigos diferentes del natural son autocomplementados, o que dentro de la misma codificación existe el complemento disminuido a la base de cada código. Por ejemplo, para el sistema BCD+3, el código del decimal 0 es 0011, y su complemento disminuido a la base es 1100 (complemento a 1 en base 2), el cual es el código del mismo sistema para el decimal 9.

Tabla 1.4: Resumen de códigos BCD

Decimal	BCD8421	BCD84-2-1	BCD2421	BCD+3
0	0000	0000	0000	0011
1	0001	0111	0001	0100
2	0010	0110	0010	0101
3	0011	0101	0011	0110
4	0100	0100	0100	0111
5	0101	1011	1011	1000
6	0110	1010	1100	1001
7	0111	1001	1101	1010
8	1000	1000	1110	1011
9	1001	1111	1111	1100

1.7.3. Suma y resta en BCD8421

SUMA BCD8421

La suma de dos cifras en BCD8421 se puede realizar bit a bit, sin embargo, al sumar es posible obtener un código mayor que $(9)_{10} = (1001)_{BCD8421}$. En estos casos, por cada código que se obtenga mayor a 9, se debe sumar $(6)_{10} = (0110)_{BCD8421}$ con el fin de ajustar el código a una representación válida. El anterior ajuste también se puede interpretar como la necesidad de restar $-(1010)_2 = [0110]_2$.

EJEMPLOS

1. Realizar $(52)_{10} + (34)_{10}$ en BCD8421

Se expresan las cantidades en formato BCD8421:

$$(0101\ 0010)_{BCD8421} + (0011\ 0100)_{BCD8421}:$$

$$\begin{array}{rcl} 52 & \Rightarrow & 0101\ 0010 \\ +34 & \Rightarrow & 0011\ 0100 \\ \hline 86 & \Rightarrow & \begin{array}{cc} \overline{1000} & \overline{0110} \\ 8 & 6 \end{array} \end{array}$$

El resultado es entonces:

$$(0101\ 0010)_{BCD8421} + (0011\ 0100)_{BCD8421} = (1000\ 0110)_{BCD8421}$$

2. Realizar $(76)_{10} + (81)_{10}$ en BCD8421

Se expresan las cantidades en formato BCD8421:

$$(0111\ 0110)_{BCD8421} + (1000\ 0001)_{BCD8421}:$$

$$\begin{array}{rcl} 76 & \Rightarrow & 0111\ 0110 \\ +81 & \Rightarrow & 1000\ 0001 \\ \hline 157 & \Rightarrow & \overline{1111\ 0111} \\ & & +0110 \\ & & \hline & & \overline{1\ 0101\ 0111} \\ & & \substack{1\qquad\quad 5\qquad\quad 7} \end{array}$$

El resultado se muestra a continuación, donde cada cantidad BCD8421 se expresa con sus 4 bits:

$$(0111\ 0110)_{BCD8421} + (1000\ 0001)_{BCD8421} = (0001\ 0101\ 0111)_{BCD8421}$$

3. Realizar $(348)_{10} + (927)_{10}$ en BCD8421

Se expresan las cantidades en formato BCD8421:

$$(0011\ 0100\ 1000)_{BCD8421} + (1001\ 0010\ 0111)_{BCD8421}:$$

$$\begin{array}{rcl}
 348 & \Rightarrow & 0011\ 0100\ 1000 \\
 +927 & \Rightarrow & 1001\ 0010\ 0111 \\
 \hline
 \overline{1275} & \Rightarrow & \overline{1100\ 0110\ 1111} \\
 & & +0110 \qquad 0110 \\
 & & \hline
 & & \overline{1\ 0010\ 0111\ 0101} \\
 & & \substack{1 \quad 2 \quad 7 \quad 5}
 \end{array}$$

El resultado se muestra a continuación, donde cada cantidad BCD8421 se expresa con sus 4 bits:

$$(0011\ 0100\ 1000)_{BCD8421} + (1001\ 0010\ 0111)_{BCD8421} = (0001\ 0010\ 0111\ 0101)_{BCD8421}$$

RESTA BCD8421

La resta de dos cifras en BCD8421 se realiza, igual que en el sistema binario, como una suma. Para lo anterior, las cantidades negativas se representan en complemento a 9 (complemento disminuido a la base) o en complemento a 10 (complemento a la base).

Para realizar una resta BCD8421 en complemento a 9, se debe pasar la cantidad negativa a su representación de complemento a 9, sumar teniendo presente un bit de signo y corregir cada valor superior a 9. Se debe recordar que al hacer una resta en complemento disminuido a la base, el posible acarreo en el bit de signo se suma a la respuesta.

EJEMPLOS

1. Realizar $(5239)_{10} - (3296)_{10}$ en BCD8421

Se expresan las cantidades en formato BCD8421 y se encuentra el complemento a 9 del negativo:

$(0101\ 0010\ 0011\ 1001)_{BCD8421} - (0011\ 0010\ 1001\ 0110)_{BCD8421}$ y el complemento a 9 de $(0011\ 0010\ 1001\ 0110)$ es $(0110\ 0111\ 0000\ 0011)$

$$\begin{array}{rcl}
 5239 & \Rightarrow & 0\ 0101\ 0010\ 0011\ 1001 \\
 -3296 & \Rightarrow & 1\ 0110\ 0111\ 0000\ 0011 \\
 1943 & \Rightarrow & \overline{1\ 1011\ 1001\ 0011\ 1100} \\
 & + & \begin{array}{r} 0110 \qquad \qquad 0110 \\ \hline 10\ 0001\ 1001\ 0100\ 0010 \\ \hline \qquad \qquad \qquad 1 \\ \hline 0\ 0001\ 1001\ 0100\ 0011 \end{array}
 \end{array}$$

El resultado es entonces:

$$(0101\ 0010\ 0011\ 1001)_{BCD8421} - (0011\ 0010\ 1001\ 0110)_{BCD8421} = (0001\ 1001\ 0100\ 0011)_{BCD8421}$$

2. Realizar $(3196)_{10} - (5239)_{10}$ en BCD8421

Se expresan las cantidades en formato BCD8421 y se encuentra el complemento a 9 del negativo:

$(0011\ 0001\ 1001\ 0110)_{BCD8421} - (0101\ 0010\ 0011\ 1001)_{BCD8421}$ y el complemento a 9 de $(0101\ 0010\ 0011\ 1001)$ es $(0100\ 0111\ 0110\ 0000)$

$$\begin{array}{rcl}
 3196 & \Rightarrow & 0\ 0011\ 0001\ 1001\ 0110 \\
 -5239 & \Rightarrow & 1\ 0100\ 0111\ 0110\ 0000 \\
 -2043 & \Rightarrow & \overline{1\ 0111\ 1000\ 1111\ 0110} \\
 & + & \begin{array}{r} \qquad \qquad \qquad 0110 \\ \hline \overline{1\ 0111\ 1001\ 0101\ 0110} \end{array}
 \end{array}$$

El resultado es negativo, por lo que está en complemento a 9 y se debe regresar a BCD:

$(1001\ 1001\ 1001\ 1001)_{BCD8421} - (0111\ 1001\ 0101\ 0110)_{BCD8421}$ lo que entrega como resultado: $-(0010\ 0000\ 0100\ 0011)_{BCD8421}$.

Para realizar una resta BCD8421 en complemento a 10, se debe pasar la cantidad negativa a su representación de complemento a 10, para lo cual primero se obtiene complemento a 9 y se suma 1, luego se realiza la suma teniendo presente un bit de signo y se corrige cada valor superior a 9. Finalmente, el posible acarreo en el bit de signo se desprecia.

EJEMPLOS

1. Realizar $(5239)_{10} - (3296)_{10}$ en BCD8421

Se expresan las cantidades en formato BCD8421 y se encuentra el complemento a 10 del negativo:

$(0101\ 0010\ 0011\ 1001)_{BCD8421} - (0011\ 0010\ 1001\ 0110)_{BCD8421}$ y el complemento a 10 de $(0011\ 0010\ 1001\ 0110)$ es $(0110\ 0111\ 0000\ 0100)$

$$\begin{array}{rcl}
 5239 & \Rightarrow & 0\ 0101\ 0010\ 0011\ 1001 \\
 -3296 & \Rightarrow & 1\ 0110\ 0111\ 0000\ 0100 \\
 \hline
 1943 & \Rightarrow & 1\ 1011\ 1001\ 0011\ 1101 \\
 & + & 0110\qquad\qquad\qquad 0110 \\
 & \hline
 & 10\ 0001\ 1001\ 0100\ 0011
 \end{array}$$

El resultado es entonces:

$$(0101\ 0010\ 0011\ 1001)_{BCD8421} - (0011\ 0010\ 1001\ 0110)_{BCD8421} = (0001\ 1001\ 0100\ 0011)_{BCD8421}$$

2. Realizar $(3196)_{10} - (5239)_{10}$ en BCD8421

Se expresan las cantidades en formato BCD8421 y se encuentra el complemento a 10 del negativo:

$(0011\ 0001\ 1001\ 0110)_{BCD8421} - (0101\ 0010\ 0011\ 1001)_{BCD8421}$ y el complemento a 10 de $(0101\ 0010\ 0011\ 1001)$ es $(0100\ 0111\ 0110\ 0001)$

$$\begin{array}{rcl}
 3196 & \Rightarrow & 0\ 0011\ 0001\ 1001\ 0110 \\
 -5239 & \Rightarrow & 1\ 0100\ 0111\ 0110\ 0001 \\
 \hline
 -2043 & \Rightarrow & 1\ 0111\ 1000\ 1111\ 0111 \\
 & + & 0110 \\
 & \hline
 & 1\ 0111\ 1001\ 0101\ 0111
 \end{array}$$

El resultado es negativo, por lo que está en complemento a 10 y se debe regresar a BCD: $(1001\ 1001\ 1001\ 1001)_{BCD8421} - (0111\ 1001\ 0101\ 0111)_{BCD8421} + (0001)_{BC8421}$ lo que entrega como resultado: $-(0010\ 0000\ 0100\ 0011)_{BCD8421}$.

1.7.4. Código Gray

Es un código no aritmético, es decir, sin pesos específicos asignados a las posiciones de los bits. La característica más importante de este código es que sólo varía un bit de una cantidad a la siguiente.

El código GRAY puede utilizar cualquier número de bits. La tabla 1.5 muestra las equivalencias para 4 bits.

Tabla 1.5: Equivalencias Binario - Gray para 4 bits

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

Este código define su aplicación en situaciones donde la transición entre códigos puede generar errores.

Por ejemplo, en la transición del código binario del número 11 al código binario del número 12 se tiene:

11 1011
 1010 → 10
 1000 → 8
12 1100

Luego, si un sistema de registro está llevando el conteo, puede ser que grabe:

$$11 - 10 - 8 - 12$$

Cuando el registro real debe grabar:

$$11 - 12$$

Como el código GRAY no es aritmético, se hace necesario establecer una metodología para su conversión hacia y desde binario.

CONVERSIÓN BINARIO A GRAY

Se rige por las siguientes reglas:

1. El bit MSD es igual

$$Gray_{MSD} = Binario_{MSD}$$

2. De izquierda a derecha, se suma cada par adyacente para obtener el siguiente bit GRAY, se descartan los acarreos.

$$Gray_i = (Binario_{i+1} + Binario_i) \mid_{\text{sin acarreos}}$$

EJEMPLO

Encontrar $(N)_G = (1101)_2$

$$Gray_{MSD} = Binario_{MSD} = 1$$

$$Gray_2 = Binario_3 + Binario_2 = 1 + 1 = 0$$

$$Gray_1 = Binario_2 + Binario_1 = 1 + 0 = 1$$

$$Gray_0 = Binario_0 + Binario_1 = 0 + 1 = 1$$

El anterior procedimiento se puede observar gráficamente a continuación:

$(1101)_2$	$Binario$	1	+	1	+	0	+	1
	↓	↓		↓		↓		↓
	$Gray$	1		0		1		1

Luego, el binario $(1101)_2$ equivale al código GRAY $(1011)_G$

CONVERSIÓN GRAY A BINARIO

Se rige por las siguientes reglas:

1. El bit MSD es igual.

$$Binario_{MSD} = Gray_{MSD}$$

2. De izquierda a derecha, a cada bit binario calculado se le suma el bit GRAY adyacente, se descartan los acarreos.

$$Binario_i = (Binario_{i+1} + Gray_i) \mid_{\text{sin acarreos}}$$

EJEMPLOS

1. Encontrar $(N)_2 = (1011)_G$
 $Binario_{MSD} = Gray_{MSD} = 1$
 $Binario_2 = Binario_3 + Gray_2 = 1 + 0 = 1$
 $Binario_1 = Binario_2 + Gray_1 = 1 + 1 = 0$
 $Binario_0 = Binario_1 + Gray_0 = 0 + 1 = 1$
El anterior procedimiento se puede observar gráficamente a continuación:

$(1011)_G$

Gray

Binario

1	0	1	1
↗ ⁺	↓ ⁺	↗ ⁺	↓ ⁺
1	1	0	1

Luego, el Gray $(1011)_G$ equivale al Binario $(1101)_2$

2. Aplicación del código GRAY.
- El código GRAY puede ser utilizado para solucionar el problema de la codificación de la posición de un eje, tal que de un código al siguiente sólo cambia un bit a la vez y por tanto se puede hacer seguimiento y verificación de la posición. En la figura 1.4 se observa a la izquierda la codificación en Binario y a la derecha la codificación en Gray.

Figura 1.4: Codificadores Binario (Izq.) y Gray (Der.) con 3 bits

1.8. CÓDIGOS PARA DETECCIÓN DE ERRORES

Error:	Desde el punto de vista digital, es la distorsión de un valor binario correcto.
Error Simple:	Desde el punto de vista digital, es la distorsión de un sólo bit correcto.
Error Múltiple:	Desde el punto de vista digital, es la distorsión de uno o más bit correctos.

Los datos binarios son manipulados por elementos físicos como hardware, están bajo la influencia de efectos externos no controlables como el ruido, temperatura, presión, etc., y en conjunto pueden intervenir para ocasionar la distorsión de los datos binarios. Con el fin de tratar con estas distorsiones, se introducen codificaciones que permitan detectar e incluso corregir los errores en la información binaria. Existen muchos métodos para detección de errores, entre los destacados están los de paridad, Hamming y los de CRC (*Cyclic Redundancy Check*).

1.8.1. Características de los códigos para detección de errores

Para un cifra, o palabra, binaria A codificada en un rango determinado, su peso $w(A)$ es el número de bits que son iguales a 1. Para dos palabras binarias A y B , su distancia $d(A, B)$ es igual al número de posiciones donde sus bits difieren, o en otras palabras donde $A_i \neq B_i$.

EJEMPLO Determinar el peso de cada palabra y la distancia entre ellas si: $A = 10001101$ y

$B = 11011101$

$w(A) = 4$, por tener cinco bits con valor de 1

$w(B) = 6$, por tener tres bits con valor de 1

$d(A, B) = 2$, por tener dos posiciones donde el bit de una palabra difiere del bit de la otra

En el diseño de un sistema digital es de especial interés disminuir la probabilidad de recibir una palabra digital que difiera de la originalmente enviada, o también poder detectar la presencia de un error e incluso poder conocer que tipo de error se presenta. Si se define la palabra A , con M_A bits, como la palabra que se desea enviar y a $C(A)$ como un código, con M_C bits, como la codificación de la palabra A que se transmite en lugar de A , es claro que el código debe tener más bits que la palabra original, por lo que $M_C > M_A$ y los bits adicionales proveen el medio para detectar y/o corregir los posibles errores. La palabra codificada $C(A)$, pasa por el medio de transmisión de donde sale como $C'(A)$, que puede o no ser igual a la información previa. Luego $C'(A)$ pasa por un sistema que determina la integridad del dato recibido (decodificador) y entrega una posible palabra enviada A^t , la cual según la evaluación de integridad puede ser correcta, es decir se toma como la original A y se pasa a la recepción. Pero la posible palabra enviada A^t puede ser incorrecta, caso en el cual se pasa a una etapa donde se decide si el tipo de error es posible de corregir, o si es necesario volver a pedir la información. Esta etapas se puede visualizar en la figura 1.5.

Figura 1.5: Modelo de sistema de control de errores

Ya se ha definido A^t como la palabra recibida después de la decodificación, entonces A^t no posee errores si $d(A^t, C(A)) = 0$, pero posee errores si $k < d(A^t, C(A)) < 0$, donde k es el máximo posible de errores para el sistema. Un código para la detección de errores puede detectar hasta un máximo de k errores si $k < d(A^t, C(A)) < 0$ y el código recibido A^t no pertenece a $C(A)$. Para ello, un código $C(A)$ debe tener una distancia mínima entre dos palabras de $\text{dmin}C = k + 1$.

Si el objetivo es corregir errores, entonces en la recepción debe existir una función de decodificación $D(A^t)$ capaz de corregir hasta k errores, y por lo tanto $D(A^t) = A$. Para este proceso se debe cumplir que $\text{dmin}C = 2k + 1$.

En general, si se define s como el número de errores a detectar y t como el número de errores a corregir, se debe cumplir para un código C :

$$\text{dmin}C \geq 2t + s + 1$$

1.8.2. Códigos básicos de paridad

Los códigos básicos de paridad, C_P , ante una palabra de entrada A , entregan un código consistente en la misma palabra más un bit adicional P , o bit de paridad, según el peso de la palabra. Existen dos codificaciones diferentes, a saber:

Código de paridad par: en esta codificación, la palabra, más su bit de paridad, deben tener un peso par. El bit de paridad se selecciona para cumplir esta restricción, por lo que si $w(A)$ es par $P = 0$ para que $w(A, P)$ siga siendo par, y si $w(A)$ es impar $P = 1$ para que $w(A, P)$ sea ahora par.

Código de paridad impar: en esta codificación, la palabra, más su bit de paridad, deben tener un peso impar. El bit de paridad se selecciona para cumplir esta restricción, por lo que si $w(A)$ es par $P = 1$ para que $w(A, P)$ sea ahora impar, y si $w(A)$ es impar $P = 0$ para que $w(A, P)$ siga siendo impar.

Para estos sistemas, el dispositivo digital que transmite calcula el bit de paridad y lo adiciona al dato, luego, el dispositivo que recibe vuelve a calcular el bit de paridad y lo compara con el calculado por el dispositivo de transmisión. Si no coinciden, se considera que el dato es erróneo y se solicita su retransmisión. Estos métodos detectan que un bit no se transmitió adecuadamente, sin embargo, si dos o más bits del mismo valor son transmitidos erróneamente no se percatan, por lo que sólo son capaces de detectar un error y no corrigen, es decir $t = 0$ y $s = 1$ y $\text{dmin}C \geq 2 \cdot 0 + 1 + 1 = 2$.

En la tabla 1.6 se observa la codificación de paridad par e impar para un código binario de entrada de 3 bits. Por ejemplo, el código de entrada $A = 011$ posee peso par ya que $w(A) = 2$, entonces su bit de paridad par debe ser 0 de tal forma que $w(0110) = 2$ sigue siendo par, y su bit de paridad impar debe ser 1 de tal forma que $w(0111) = 3$ ahora es impar. Si A se envía con paridad impar por un medio de comunicación y sufre de alteraciones, por ejemplo llega 0011 el sistema de recepción evalúa que $w(0011) = 2$ es par y por tanto es incorrecto; sin embargo si durante el proceso de transmisión se alteran más de dos bits, como en el caso 1101, el peso es $w(1101) = 3$ y el sistema no se percató de los errores, aceptando la palabra.

Tabla 1.6: Códigos básicos de paridad

Código de entrada		Código	Bit par		Código	Bit impar
000		000	0		000	1
001		001	1		001	0
010		010	1		010	0
011		011	0		011	1
100		100	1		100	0
101		101	0		101	1
110		110	0		110	1
111		111	1		111	0

En la tabla 1.6 se observa como de una palabra aceptada a la siguiente existe una distancia constante de 2, lo cual hace que para estos códigos $dminC = 2$, como ya se dijo.

1.8.3. Códigos de repetición

Si en un código binario de entrada, cada bit se repite n veces, entonces se tiene una forma de verificar la distorsión de un valor. Si, por ejemplo, un bit 1 se repite tres veces, se tiene el grupo 111, por lo que en caso de llegar un grupo 011, 101 o 110 se puede decir que el valor verdadero es 1; sin embargo, si se cambian dos bits se puede tener un valor incorrecto. En la tabla 1.7 se muestra la codificación de un código de repetición con $n = 3$ para un código de entrada con 3 bits.

Tabla 1.7: Códigos de repetición

Código de entrada		Grupo 1	Grupo 2	Grupo 3
000		000	000	000
001		000	000	111
010		000	111	000
011		000	111	111
100		111	000	000
101		111	000	111
110		111	111	000
111		111	111	111

En la tabla 1.7 se observa que entre algunas palabras del código existe una distancia de 6 y entre otras de 3, por ello $dminC = 3$, y el sistema puede corregir un error, ya que con $t = 1$ y $s = 0$ se tiene que $dminC \geq 2*1+0+1 = 3$. Estos códigos requieren más bits de repetición para incrementar sus posibilidades de corrección, lo cual es ineficiente ya que incrementa la cantidad de información a ser tratada y transmitida.

1.8.4. Códigos Hamming

Es un código para el control de errores propuesto por R. W Hamming usando una distancia mínima $d_{min} = m$. En este método, por cada entero m existe un código de Hamming de $2^m - 1$ bits que contiene m bits de paridad y $2^m - 1 - m$ bits de información. Los bits de paridad y los bits de información se encuentran entremezclados de la siguiente forma: si se numeran las posiciones de los bits desde 1 hasta $2^m - 1$, los bits en las posiciones 2^k con $0 \leq k \leq m - 1$ son bits de paridad y los restantes son bits de información. El valor de cada bit de paridad se escoge de modo que el total de unos en un grupo específico de bits sea par, y cada grupo se selecciona de forma tal que ningún bit de información quede con la misma combinación de bits de paridad. Lo anterior es lo que proporciona al código su capacidad de corrección. Cada bit de paridad en la posición 2^k , comprueba los bits en las posiciones que tengan al bit k en su representación binaria.

El procedimiento para evaluación de los bits de paridad se describe a continuación:

1. Los bits de paridad van en posiciones potencia de dos (posiciones 1, 2, 4, 8, 16, 32, 64, etc.).
2. Los bits restantes son bits de información (posiciones 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, etc.).
3. La posición del bit de paridad determina la secuencia de los bits que alternativamente comprueba y salta, a partir de este, tal y como se explica a continuación.
 - Posición 1: $2^0 = 1$, salta 0, comprueba 1, salta 1, comprueba 1, etc. Es decir, comprueba los bits 3, 5, 7, 9, 11, etc. ya que en binario 0011, 0101, 0111, 1001, 1011, etc., tienen un 1 en la posición 1.
 - Posición 2: $2^1 = 2$, salta 1, comprueba 2, salta 2, comprueba 2, etc. Es decir, comprueba los bits 3, 6, 7, 10, 11, etc. ya que en binario 0011, 0110, 0111, 1010, 1011, etc., tienen un 1 en la posición 2.
 - Posición 4: $2^2 = 4$, salta 3, comprueba 4, salta 4, comprueba 4, etc. Es decir, comprueba los bits 5, 6, 7, 12, 13, 14, 15 etc. ya que en binario 0101, 0110, 0111, 1100, 1101, 1110, 1111 etc., tienen un 1 en la posición 3.
 - Posición 8: $2^3 = 8$, salta 7, comprueba 8, salta 8, comprueba 8, etc.
 - En general para la posición n : salta $n-1$ bits, comprueba n bits, salta n bits, comprueba n bits, etc.

CÓDIGO HAMMING(7,4)

El código Hamming(7,4) posee 7 bits de información y cuatro de paridad de acuerdo con las reglas generales vistas previamente.

EJEMPLO

Codificar con Hamming(7,4) la palabra 0111010

En la tabla 1.8 las posiciones 1, 2, 4 y 8 son bits de paridad y las restantes bits de información

El bit de paridad 1, en la posición 1, comprueba las posiciones 3, 5, 7, 9 y 11

El bit de paridad 2, en la posición 2, comprueba las posiciones 3, 6, 7, 10 y 11

El bit de paridad 3, en la posición 4, comprueba las posiciones 5, 6 y 7

El bit de paridad 4, en la posición 8, comprueba las posiciones 9, 10 y 11

Una vez se transmite una palabra codificada con Hamming(7,4) se comprueban los bits de paridad, para lo cual si una paridad está correcta se asigna un valor de 0 y en caso contrario un valor de 1 en una nueva palabra que identifica la posición del error. En la palabra de comprobación, el bit de paridad de mayor posición es el más significativo y el bit de paridad de menor posición es el menos significativo.

Tabla 1.8: Ejemplo de código Hamming(7,4)

	1	2	3	4	5	6	7	8	9	10	11
	p1	p2	d1	p3	d2	d3	d4	p4	d5	d6	d7
Palabra original			0		1	1	1		0	1	0
p1	0		0		1		1		0		0
p2		1	0			1	1			1	0
p3				1	1	1	1				
p4								1	0	1	0
Palabra codificada	0	1	0	1	1	1	1	1	0	1	0

EJEMPLO

Si en el ejemplo de codificación con Hamming(7,4) en vez de arribar la palabra 0101111010 llega la palabra 01011011010, identificar la posición del error.

En la tabla 1.9 se comprueban los bits de paridad:

Tabla 1.9: Ejemplo de comprobación Hamming(7,4)

	1	2	3	4	5	6	7	8	9	10	11	Prueba	Bit
	p1	p2	d1	p3	d2	d3	d4	p4	d5	d6	d7		
Recepción	0	1	0	1	1	0	1	1	0	1	0		
p1	0		0		1		1		0		0	Correcto	0
p2		1	0			0	1			1	0	Error	1
p3				1	1	0	1					Error	1
p4								1	0	1	0	Correcto	0

La palabra de comprobación es $(0110)_2 = (6)_{10}$, con lo cual la posición 6 tiene un valor incorrecto.

El código Hamming(7,4) puede corregir el error de un solo bit, pero cuando hay errores en más de un bit, la palabra recibida se confunde con otra de error en un sólo bit, ante lo cual se corrige pero de forma incorrecta.

EJEMPLO

- Codificar con Hamming(7,4) la palabra 0111010
- En la tabla 1.8 las posiciones 1, 2, 4 y 8 son bits de paridad y las restantes bits de información
- El bit de paridad 1, en la posición 1, comprueba las posiciones 3, 5, 7, 9 y 11
- El bit de paridad 2, en la posición 2, comprueba las posiciones 3, 6, 7, 10 y 11
- El bit de paridad 3, en la posición 4, comprueba las posiciones 5, 6 y 7
- El bit de paridad 4, en la posición 8, comprueba las posiciones 9, 10 y 11

Una vez se transmite una palabra codificada con Hamming(7,4) se comprueban los bits de paridad, para lo cual si una paridad está correcta se asigna un valor de 0 y en caso contrario un valor de 1 en una nueva palabra que identifica la posición del error. En la palabra de comprobación, el bit de paridad de mayor posición es el más significativo y el bit de paridad de menor posición es el menos significativo.

Los códigos CRC tienen su fundamentación en la evaluación del residuo de dividir un polinomio entre otro. Los polinomios emplean el formato GF(2), o polinomios de campo Galois con dos elementos, es decir, los polinomios son de una sola variable x con coeficientes 1 o 0. La suma y resta de los polinomio se hace sumando coeficientes sin tener presente acarreo, la multiplicación entrega un coeficiente en el producto siempre que exista ese coeficiente en los operandos y la división se realiza como las divisiones de polinomios sobre enteros.

Los códigos CRC tratan la información como si fuera un polinomio GF(2), por lo que una palabra de envío $A = 11010011$ es una representación del polinomio $x^7 + x^6 + x^4 + x^1 + 1$. Los sistemas de envío y recepción unifican sobre un polinomio que se denomina polinomio generador el cual es de grado r . El sistema adiciona r bits de valor cero a la palabra de envío que representa un polinomio de grado m . Así, se obtiene un nuevo polinomio de grado $m + r - 1$, el cual es dividido por el polinomio generador, entregando un residuo de grado máximo $r - 1$. El polinomio residuo de máximo r coeficientes son los bits de verificación. La palabra que se transmite es finalmente la palabra original de m bits más los r bits de verificación.

Una vez transmitida la palabra, existen dos formas de verificar su integridad: una primera forma es volver a calcular sobre la palabra los bits de verificación y asegurar que coincidan con los transmitidos; la segunda forma es dividir todos los $m + r$ bits recibidos por el polinomio generador y verificar que el residuo de r bits sea cero.

Existen varios estándares para códigos CRC, entre los cuales los más comunes son los listados en la tabla 1.10.

Tabla 1.10: Listado de polinomios generadores para códigos CRC

Nombre	r	Polinomio Generador
CRC-12	12	$x^{12} + x^{11} + x^3 + x^2 + x + 1$
CRC-16	16	$x^{16} + x^{15} + x^2 + 1$
CRC-CCITT	16	$x^{16} + x^{12} + x^5 + 1$
CRC-32	32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

El código CRC-12 se emplea en la transmisión de 6 bits por streaming, el código CRC-16 se emplea en el estándar de comunicación BISYNCH de IBM, el código CRC-CCITT se emplea en los protocolos XMODEM, X.25, SDLC de IBM y HDLC de ISO. El código CRC-32 se emplea en los protocolos PKZip, Ethernet, AAL5, FDDI, IEEE-802, etc.

EJEMPLO

Codificar el mensaje $M = 1011000100101010$ en un código CRC que posee un polinomio generador $GF = x^3 + 1$, representado en binario por 1001.

Como el polinomio generador es de grado 3, se adicionan 4 bits ceros al mensaje original:

$M = 1011000100101010\ 0000$.

Los bits de verificación se obtienen del residuo de dividir $M = 1011000100101010\ 0000$ entre 1001.

La división propuesta entrega como residuo 0011.

El mensaje transmitido es: 1011000100101010 0011. Para verificar en la recepción, se divide lo recibido entre el polinomio generador y el residuo debe ser cero.

1.9. EJERCICIOS PROPUESTOS

1. Convertir de la base indicada a la base decimal.

- a) $N_2 = 1101011$
- b) $N_8 = 54261$
- c) $N_{16} = 42F7CA$
- d) $N_7 = 6235$
- e) $N_3 = 211,0121$

2. Convertir de la base decimal a las bases indicadas.

- a) $N_{10} = 572$ pasar a bases 2, 8 y 16
- b) $N_{10} = 638,24$ pasar a bases 2, 8 y 16
- c) $N_{10} = 159$ pasar a bases 5 y 9
- d) $N_{10} = 45,71$ pasar a bases 3 y 6
- e) $N_{10} = 162,138$ pasar a base 2, 4 y 8

3. Convertir entre las bases indicadas.

- a) $N_2 = 11011, 1101101$ pasar a bases 10 y 16
- b) $N_2 = 1010, 01101$ pasar a bases 8 y 16
- c) $N_5 = 2341,41$ pasar a bases 10 y 7
- d) $N_7 = 1652,45$ pasar a bases 6 y 16
- e) $N_8 = 4671,13$ pasar a bases 2, 9 y 16

4. Hacer las siguientes operaciones.

- a) $(26)_{10} + (172)_{10}$ operación en base 2 y representación I8
- b) $(45)_{10} - (74)_{10}$ operaciones en base 2 y representación I16
- c) $-(264)_{10} - (387)_{10}$ operaciones en base 2 y representación I16
- d) $(462)_8 - (5A)_{16}$ operaciones en base 2, seleccione una representación adecuada
- e) $(00111001)_{BCD} + (00100110)_{BCD}$ hacer en BCD y en base 2.

5. Hacer las siguientes operaciones en base 2. Seleccione una representación adecuada.

- a) $(75)_{10} * (23)_{10}$
- b) $-(63)_{10} * (21)_{10}$
- c) $(41)_8 * -(C1)_{16}$
- d) $(473)_{10} \div -(23)_{10}$
- e) $(1011011)_2 \div (A)_{16}$

6. Almacenar los siguientes números en representación de punto flotante.

- a) $(3582)_{10}$
- b) $(733,02301)_{10}$
- c) $(0,004825)_{10}$
- d) $(11011,110001)_2$
- e) $(271,44)_8$

7. Hacer las siguientes conversiones.

- a) $N_2 = 101101$ pasar a Gray y BCD
- b) $N_8 = 624$ pasar a Gray y BCD
- c) $N_5 = 142$ pasar a Gray y BCD
- d) $N_{10} = 967$ pasar a Gray y BCD
- e) Gray = 110110 pasar a binario y BCD

8. Si un sistema digital recibe por medio de un canal de comunicación la cadena 10110010011

- a) Si la recepción es en codificación de paridad par, ¿la recepción es la correcta?
- b) Si la recepción es en codificación de paridad impar, ¿la recepción es la correcta?
- c) Si la recepción es en codificación Hamming(7,4), ¿la recepción es la correcta?

9. Codifique la siguiente cadena para ser enviada por un canal de comunicación: 1101110

- a) Si la codificación es en paridad par
- b) Si la codificación es en paridad impar
- c) Si la codificación es en repeticiones con $n = 3$
- d) Si la codificación es en Hamming(7,4)
- e) Si la codificación es en CRC con Polinomio Generador $GF = x^2 + 1$

10. Codifique la siguiente cadena para ser enviada por un canal de comunicación: 1101010

- a) Si la codificación es en paridad par
- b) Si la codificación es en paridad impar
- c) Si la codificación es en repeticiones con $n = 2$
- d) Si la codificación es en Hamming(7,4)
- e) Si la codificación es en CRC con Polinomio Generador $GF = x^3 + 1$

11. Adicionales

- a) ¿En qué consisten las etapas de cuantización y muestro en un proceso de digitalización?
- b) ¿Cómo se puede clasificar el sistema romano de numeración? Defina y justifique
- c) Realice la deducción de rangos para cada tipo de dato.
- d) Investigue sobre otros métodos para detección de errores.

2

CAPÍTULO DOS

Lógica de proposiciones

Los sistemas de numeración ponderados y no ponderados son el soporte sobre el cual se puede realizar cualquier planteamiento, diseño y/o análisis de un sistema digital. Sin embargo, antes de poder emprender estas tareas, en este capítulo se afronta el reto inicial de traducir los enunciados de contextos de aplicación, que usualmente se entregan en un lenguaje natural no estructurado, a expresiones matemáticas formalizadas que permitan su tratamiento. Estas expresiones formalizadas se trabajan mediante la denominada lógica de proposiciones, la cual establece las unidades básicas de información, las diferentes formas en las cuales se pueden conectar estas ideas entre sí, como se establecen los valores de verdad de ideas simples y de ideas complejas y como se expresa su representación formalizada.

2.1. DEFINICIONES

Oración:	<i>Palabra, o secuencia de palabras, que constituyen en conjunto la unidad más pequeña de comunicación.</i>
Proposición:	<i>Clase de oración que constituye por sí misma unidad de comunicación, y que puede ser verdadera o falsa.</i>

El lenguaje es un sistema compuesto por un conjunto de sonidos y grafías con sentido, que está sujeto a una determinada estructura jerárquica. La oración es una palabra, o secuencia de las mismas, que constituyen en conjunto la unidad más pequeña; según lo que quiera comunicar, se clasifican en:

Oraciones aseverativas o declarativas: oraciones que expresan ideas a las que se les puede asignar un valor de verdad.

Oraciones optativas o desiderativas: oraciones que expresan deseos, se pronuncian normalmente de forma exclamativa y no poseen un valor de verdad.

Oraciones interrogativas: oraciones para formular preguntas y no poseen un valor de verdad.

Oraciones exclamativas o admirativas: oraciones que expresan sorpresa, afectividad, emoción o admiración, tampoco tienen un valor de verdad.

Oraciones exhortativas o imperativas: oraciones que expresan consejo, ruego, mandato o prohibición, tampoco tienen un valor de verdad.

Oraciones dubitativas: oraciones que expresan suposición o probabilidad. Aunque pueden afirmar, su valor de verdad no se puede establecer.

2.1.1. Propositiones

Las proposiciones son oraciones aseverativas o declarativas en las que tiene sentido expresar un valor de verdad. Las oraciones de tipo optativas, interrogativas, exclamativas y exhortativas no son proposiciones ya que en ninguna de ellas se expresa verdad o falsedad, o su valor no se puede establecer, como en el caso de las dubitativas. En algunas oraciones aseverativas no tiene sentido asignar un valor de verdad, por lo que se denominan pseudoproposiciones, o falsas proposiciones.

EJEMPLO

Determinar el tipo de oración:

- Enero es el primer mes del año: oración aseverativa y por tanto proposición, con valor verdadero.
- La llave está abierta: oración aseverativa y por tanto proposición, con valor verdadero.
- El horno congela alimentos: oración aseverativa y por tanto proposición, con valor falso.
- Un cuadrilátero es un polígono de cinco lados: oración aseverativa y por tanto proposición, con valor falso.
- El automóvil es racional: oración aseverativa, más no proposición al no tener sentido asignarle valor de verdad.
- El exponente tiene tres hijos: oración aseverativa, más no proposición al no tener sentido asignarle valor de verdad.
- Ojalá llueva mañana: oración desiderativa y por tanto no proposición.
- ¿De cuánto tiempo dispongo?: oración interrogativa y por tanto no proposición.
- ¡Que alegría volver a verte!: oración admirativa y por tanto no proposición.
- ¡No puedes dejar de comer!: oración exhortativa y por tanto no proposición.
- Tal vez no sea tan rápido como pensaba: oración dubitativa y por tanto no proposición.
- ¿Acaso tengo razón?: oración dubitativa y por tanto no proposición.

2.1.2. Tipos de proposiciones

Cuando una proposición expresa una única idea en su forma más simple se dice que es una **proposición atómica**. Si expresa más de una idea, o proposición atómica, se dice que es una **proposición molecular**.

Para el estudio de las proposiciones se requiere de la definición de un lenguaje formal que indique el conjunto de símbolos y reglas que en conjunto forman el alfabeto de este lenguaje.

2.2. LÓGICA PROPOSICIONAL FORMAL

El lenguaje natural se emplea en la vida cotidiana, sirve para expresar información, deseos, sentimientos, dar órdenes, formular preguntas, dar consejos, etc.; son ejemplos de este tipo de lenguaje el español, el inglés, el alemán, el francés, etc. El lenguaje lógico sirven para formular conocimiento, es formalizado por que sus propiedades entregan la forma y estructura de las proposiciones.

2.2.1. Alfabeto

El alfabeto se compone de un conjunto de símbolos reunidos en las siguientes categorías:

Constantes: símbolos formados por las primeras letras en minúsculas, y que pueden tener subíndices, que se emplean para designar los objetos de los cuales se quiere hacer referencia. Por ejemplo, para designar un motor se puede hacer referencia a este con la constante m . Si existen dos motores se pueden designar respectivamente como m_1 y m_2 .

Variables proposicionales: símbolos formados por las últimas letras en minúsculas y que se emplean para designar objetos indeterminados.

Relatores: símbolos formados por las primeras letras en mayúsculas y que se emplean para describir las acciones o hechos. Si “estar caliente” se representa por C , entonces Cm significa el motor está caliente, o de igual forma Cm_1 significa el motor 1 está caliente. Otro posible relator es “ser idénticos” y que puede ser representado por I , y el cual en Im_1m_2 significa el que el motor 1 es idéntico al motor 2. En ambas situaciones previas se tienen ejemplos de proposiciones atómicas. Los dos relatores dados previamente se diferencian por requerir de un número diferente de complementos para tomar sentido, así, C sólo necesita de uno, mientras I necesita de dos. Cuando un relator requiere de sólo un complemento se dice es de rango uno o **monádico**, mientras si requiere de n complementos se dice que es un relator de rango n . En particular un relator de rango dos recibe el nombre de relator **diádico**. Un relator diádico de uso común es el igualador o = y que se lee como “ser igual a”. Ejemplos de otros relatores comunes son $>$, $<$, $\leq$, $\neq$, etc.

Conectivas lógicas: son el conjunto de símbolos que permiten relacionar proposiciones atómicas con el fin de formar proposiciones moleculares y establecer operaciones entre ellas y son:

1. Negador: se simboliza por $\neg$, aunque también se emplean símbolos como ' o $\cdot$. Se lee como “no”, “no es cierto que”, “es falso que”, “no es verdad que”. Si Cm significa el motor está caliente, entonces $\neg Cm$ significa el motor no está caliente.
2. Conjunción: se simboliza por $\wedge$, aunque también se emplean símbolos como & ó $\cdot$. Se lee como “y”. De esta forma $\neg Cm_1 \wedge Cm_2$ significa que el motor 1 no está caliente y el motor 2 está caliente. Será verdadera cuando ambas partes sean verdaderas.
3. Disyunción: se subdivide en dos tipos a saber: la disyunción inclusiva que se simboliza por $\vee$ o $+$ y que se lee como “o”, “lo uno, lo otro o ambos”; y en la disyunción exclusiva que se simboliza por $\oplus$ o $\leftrightarrow$ y que se lee como “lo uno, lo otro, pero no ambos”. Si el relator P significa “poder encender”, entonces $Pm_1 \vee Pm_2$ significa que se puede encender el motor 1 o se puede encender el motor 2, o se pueden encender ambos; pero $Pm_1 \oplus Pm_2$ significa que se puede encender el motor 1 o se puede encender el motor 2, pero no se pueden encender ambos. La disyunción inclusiva es verdadera cuando una cualquiera de las dos partes es verdadera, mientras que la disyunción exclusiva será verdadera cuando exactamente una de las partes sea verdadera.
4. Implicador o condicional: se simboliza por $\rightarrow$ y se lee “... implica que ...” o “si ... entonces ...”. La interpretación del condicional ha tenido varios puntos de vista en la historia, pero se opta por decir que $A \rightarrow B$ será verdadera si A es verdadera y B también lo es o si A es falsa sin importar que ocurre con B . Si se tiene $Cm_1 \rightarrow \neg Cm_2$ se lee como “si el motor 1 está caliente entonces el motor 2 no está caliente; esto será verdad si el motor 1 está caliente y el motor 2 no lo está, o si el motor 1 no está caliente, caso este último en el cual no importa como esté el motor 2. Sin embargo, si el motor 1 está caliente y el motor 2 está igual, se tiene que la implicación es falsa.
5. Coimplicador o bicondicional: se simboliza por $\leftrightarrow$ y se lee “... si y sólo si ...” o “... es condición necesaria y suficiente ...”. $Cm_1 \leftrightarrow Cm_2$ se interpreta como que el motor 1 está caliente si y sólo si el motor 2 está caliente.

El conjunto de símbolos para las conectivas lógicas que se seguirán empleando en el presente capítulo son $\neg, \vee, \wedge, \oplus$, pero en los siguientes capítulos se emplearán $+, \cdot, \oplus$ respectivamente. La negación es un operador monádico, ya que sólo afecta a una variable. Los demás operadores son diádicos por tener un doble alcance, es decir afectan a dos variables.

Cuantificadores: son dos, el cuantificador existencial, que se simboliza por $\vee y$ y se lee “existe” y el cuantificador universal $\wedge y$, que se lee “para todo”. Los cuantificadores se emplean conjuntamente con las variables. De lo anterior, $\vee y Cy$ se interpreta como que existe un y de forma que y está caliente, donde la variable y hace referencia a un motor. Análogamente, $\wedge y Cy$ se interpreta como que para todo y , y está caliente.

Descriptor: se simboliza por $| y$ y se lee “tal que”. $\vee y | \dots$ se interpreta como que existe un y tal que cumple una condición dada.

Signos de puntuación: son especialmente los paréntesis y la coma, que se emplean de la misma forma que en los lenguajes naturales para relacionar precedencias o separación.

Función: representada por las letras f , g o h . El número de argumentos de una función es igual a su *aridad*. Así, una función con tres argumentos tiene una aridad de 3.

2.2.2. Notaciones simbólicas

Las principales notaciones simbólicas se muestran en la tabla 2.1.

Tabla 2.1: Tabla de notaciones simbólicas

Sistema	Negación	Conjunción	Disyunción inclusiva	Disyunción exclusiva	Condicional	Bicondicional
Scholz	$\neg p$	$p \wedge q$	$p \vee q$	$p \oplus q$	$p \rightarrow q$	$p \leftrightarrow q$
Peano-Russell	$\sim p$	$p \bullet q$	$p \vee q$	$p \neq q$	$p \supset q$	$p = q$
Lukasiewicz	Np	Kpq	Apq	Jpq	Cpq	Epq

En la tabla 2.1, p y q representan variables proposicionales. Las principales características de cada sistema son:

Sistema Scholz: los operadores monádicos preceden las variables, los diádicos se escriben entre las variables, los operadores se representan con signos especiales y emplea paréntesis para determinar la jerarquía.

Sistema Peano-Russell: igual al sistema Scholz, con la diferencia de emplear puntos auxiliares para determinar la jerarquía.

Sistema Lukasiewicz: los operadores se escriben delante de las variables, los operadores se representan con letras mayúsculas y no emplea signos de agrupación.

Como la orientación del presente texto es el estudio de los sistemas digitales, la notación de Scholz es la que mejor se ajusta.

2.2.3. Jerarquía de conectivas lógicas

Se establece una jerarquía entre las conectivas lógicas y los signos de puntuación, de tal forma que listados de menor a mayor jerarquía son:

- La negación
- El condicional y bicondicional
- La conjunción y la disyunción, ya sea inclusiva o exclusiva
- Los paréntesis

Con base en la jerarquía establecida, $p \rightarrow (q \wedge (\neg r))$ es igual a $p \rightarrow q \wedge \neg r$. Se acostumbra omitir los paréntesis cuando la jerarquía quede plenamente establecida, con o sin ellos.

- Los paréntesis

Con base en la jerarquía establecida, $p \rightarrow (q \wedge (\neg r))$ es igual a $p \rightarrow q \wedge \neg r$. Se acostumbra omitir los paréntesis cuando la jerarquía quede plenamente establecida, con o sin ellos.

EJEMPLOS

1. Ejemplos de proposiciones atómicas son:
 - a) El motor está en operación.
 - b) El agua tiene nivel bajo.
 - c) La válvula está cerrada.
 - d) El horno tiene temperatura normal.
 - e) No hay fluido eléctrico.
2. Ejemplos de proposiciones moleculares son:
 - a) El motor está operando y el agua tiene nivel bajo.
 - b) El horno tiene temperatura normal o la válvula no está cerrada.
 - c) No hay fluido eléctrico y el horno no tiene temperatura normal.
 - d) Si hay fluido eléctrico entonces el motor está operando.
 - e) El motor está operando si y sólo si hay fluido eléctrico.
3. Ejemplos de proposiciones moleculares expresadas mediante el alfabeto definido son (se muestra las proposiciones moleculares del ejemplo previo):
 - a) Si m = motor, a = agua, O = estar operando y N = nivel bajo, se tiene: $Om \vee Na$.
 - b) Si h = horno, v = válvula, T = tener temperatura normal y C = estar cerrada, se tiene: $Th \wedge \neg Cv$.
 - c) Si f = fluido eléctrico y H = No haber, se tiene: $Hf \vee \neg Th$.
 - d) $\neg Hf \rightarrow Om$.
 - e) $Om \leftrightarrow \neg Hf$.

2.2.4. Fórmulas lógicas

<i>Sentencias:</i>	<i>Serie de signos que representan un enunciado declarativo o proposición atómica.</i>
<i>Fórmulas lógicas:</i>	<i>Cadenas de símbolos estructurados bajo una serie de reglas establecidas por la sintaxis lógica.</i>
<i>Sintaxis lógica:</i>	<i>Disciplina que estudia el lenguaje de la lógica interesándose solo por las relaciones entre los símbolos.</i>

Las sentencias se representan mediante letras, que reciben el nombre de **letras sentenciales** o **literales**. Su objeto es simplificar la representación de una proposición atómica compuesta por una constante o variable y un relator. También se denominan **fórmulas atómicas**.

Una **fórmula molecular** es una representación simbólica de una proposición molecular; posee varias fórmulas atómicas enlazadas mínimo por el símbolo de una conectiva lógica.

La sintaxis lógica permite la construcción de fórmulas lógicas, o fórmulas bien formadas (FBF), mediante el establecimiento de reglas para emplear y combinar símbolos. Las principales reglas para la formación de fórmulas bien formadas son las siguientes:

1. Toda variable proposicional es una FBF.
2. Si p es una FBF, entonces $\neg p$ es una FBF.
3. Si p y q son FBF, entonces $p \vee q$, $p \wedge q$, $p \rightarrow q$, $p \leftrightarrow q$ y $p \leftrightarrow q$ también son FBF.
4. Una cadena de símbolos es una FBF, si y sólo si se sigue de la aplicación de las reglas 1, 2 y 3.
5. Una fórmula lógica es bien formada si y sólo si existe una jerarquía establecida entre sus operadores.
6. El nombre de una FBF están dado por el operador de mayor jerarquía.
7. El operador de mayor jerarquía es aquel que está libre de los signos de agrupación.
8. Los signos de agrupación solo se aplican cuando su omisión es susceptible de una doble interpretación.
9. Los operadores diádicos tienen mayor jerarquía que los monádicos.
10. El operador de negación se escribe antes de una FBF.
11. El operador de negación no se escribe entre las FBF. Sin embargo, por facilidad de lectura, en sistemas digitales se acostumbra omitir esta regla.
12. Si un operador de negación antecede otro operador de negación, el operador de la izquierda tiene mayor jerarquía.

EJEMPLOS

1. $p \wedge q \vee r$, es una fórmula mal formada por la falta de signos de agrupación. Si se escribe $(p \wedge q) \vee r$ es una FBF y se denomina disyunción inclusiva, si se escribe $p \wedge (q \vee r)$ es una FBF y se denomina conjunción. En los sistemas digitales se acostumbra asignar una mayor jerarquía al operador de conjunción, con lo cual la fórmula original estaría bien formada y se interpreta como $p \wedge (q \vee r)$.

2. $p \leftrightarrow (q \vee r)$, es una FBF formada y se denomina bicondicional.
3. $\neg[p \leftrightarrow (\neg q \vee r)]$ es una FBF y se denomina negación.
4. $\neg[p \leftrightarrow (q \neg \vee r)]$ es una fórmula mal formada, por tener una negación a la izquierda de la variable.
5. $\neg[p \leftrightarrow (q \neg \vee r)] \rightarrow \neg s$ es una FBF y se denomina condicional.

2.2.5. Conjunciones

En un enunciado que se expresa en lenguaje normal, las conjunciones realizan la misma función de las conectivas, es decir, cumplen la tarea de enlazar las diferentes proposiciones atómicas. La tabla 2.2 es un listado de algunas de las conjunciones del lenguaje español y su traducción equivalente a conectivas lógicas, donde r y s son dos proposiciones atómicas cualesquiera.

2.2.6. Proceso de formalización

El proceso de formalizar una proposición molecular consiste en reescribirla en una nueva proposición donde su estructura sintáctica sea explícita, es decir, reescribirla empleando las conjunciones básicas de las conectiva lógicas. En el proceso de formalización, se reemplaza cada proposición atómica por una variable proposicional, o letra sentencial, y los conectivos lógicos por sus respectivos símbolos. Se puede emplear signos de agrupación para establecer la jerarquía entre operadores de una fórmula lógica, pero se deben omitir si la jerarquía queda plenamente establecida sin ellos.

EJEMPLOS

1. El motor está operando y el agua tiene nivel bajo. En este caso se emplean como letras sentenciales, o literales, a p = el motor está operando y q = el agua tiene nivel bajo. Lo cual se simplifica por $p \vee q$.
2. El horno tiene temperatura normal o la válvula no está cerrada. Usando h = el horno tiene temperatura normal y v = la válvula está cerrada, entonces se obtiene $h \wedge \neg v$. Se debe observar que se puede definir v = la válvula no está cerrada, caso en el cual se obtiene $h \wedge v$.
3. No hay fluido eléctrico y el horno no tiene temperatura normal. Usando f = hay fluido eléctrico y h = el horno tiene temperatura normal, se obtiene $\neg f \vee \neg h$.
4. Cuando un conductor arriba a una estación de servicio se le permite llenar combustible si no está fumando y:
 - a) Estando solo apaga el vehículo, o
 - b) Estando acompañado se bajan todos y apaga el vehículo.

Inicialmente se hace la asignación de letras sentenciales así:

Tabla 2.2: Conversión de conjunciones comunes a conectivas lógicas

CONJUNCIONES	CONECTIVAS
No r	$\neg r$
No es cierto que r	$\neg r$
Es falso que r	$\neg r$
No es verdad que r	$\neg r$
Jamás r	$\neg r$
Carece r de	$\neg r$
r y s	$r \wedge s$
Tanto r como s	$r \wedge s$
r tanto como s	$r \wedge s$
Ambas r y s	$r \wedge s$
r e igualmente s	$r \wedge s$
r además de s	$r \wedge s$
ni r ni s	$\neg r \wedge \neg s$
No ambas r y s	$\neg r \wedge \neg s$
r o s	$r \vee s$
r o s o ambas	$r \vee s$
alguna entre r y s	$r \vee s$
cualquiera entre r y s	$r \vee s$
ya sea r o sea s	$r \vee s$
r y/o s	$r \vee s$
no es cierto que r o s	$\neg (r \vee s)$
r o s pero no ambas	$r \oplus s$
r implica que s	$r \rightarrow s$
Si r entonces s	$r \rightarrow s$
r siempre y cuando s	$r \rightarrow s$
r con tal que s	$r \rightarrow s$
r si s	$r \rightarrow s$
r puesto que s	$r \rightarrow s$
r implica que no s	$r \rightarrow \neg s$
Si r entonces no s	$r \rightarrow \neg s$
r a menos que no s	$r \rightarrow \neg s$
Si no r entonces s	$\neg r \rightarrow s$
r no implica que s	$\neg (r \rightarrow s)$
r si y sólo si s	$r \leftrightarrow s$
r es condición necesaria y suficiente para que s	$r \leftrightarrow s$
r cuando y sólo cuando s	$r \leftrightarrow s$
si r entonces y sólo entonces no s	$r \leftrightarrow \neg s$
r es condición necesaria y suficiente para que no s	$r \leftrightarrow \neg s$
No r es condición necesaria y suficiente para que s	$\neg r \leftrightarrow s$
r no es condición necesaria y suficiente para que s	$\neg (r \leftrightarrow s)$

Letra sentencial	Sentencia
l	Llenar combustible
f	Estar fumando
s	Estar solo
b	Bajar del vehículo
a	Apagar el vehículo

El enunciado posee dos partes enlazadas entre sí por la conjunción “o”, pero ambas partes se ven afectadas por la imposibilidad de fumar. Por lo que la primera parte se puede relacionar como $\neg f \wedge (s \wedge a)$, y la segunda parte como $\neg f \wedge (\neg s \wedge b \wedge a)$. En definitiva:

$$l = [\neg f \wedge (s \wedge a)] \vee [\neg f \wedge (\neg s \wedge b \wedge a)]$$

La anterior expresión no es la única forma de relacionar el enunciado, ya que se puede definir la letra sentencial s como estar acompañado, caso en el cual se debe cambiar el sentido de la negación aplicada. Además, mediante aplicación de algunas reglas de álgebra, se puede expresar de forma diferente. Todos estos temas se tratan en el siguiente capítulo.

5. Una alarma contra incendio en un almacén posee una configuración de sensores como la que se muestra en la figura 2.1. Para que la alarma se active, se debe cumplir alguna de las siguientes condiciones:
 - a) Se active el sensor central y uno cualquiera de los otros.
 - b) Se activen tres sensores cualesquiera.
 - c) Se activen todos los sensores.

Figura 2.1: Sensores contra incendio

Se hace asignación de sentencias así:

Letra sentencial	Sentencia
s	Sonar alarma
i	Sensor de la izquierda activo
d	Sensor de la derecha activo
c	Sensor del centro activo
a	Sensor de abajo activo

La alarma sonará si se cumple una cualquiera de las tres condiciones, las cuales, desde un punto de vista simple, se pueden representar como un listado de posibilidades así:

$$s = (c \wedge i) \vee (c \wedge d) \vee (c \wedge a) \vee (c \wedge d \wedge i) \vee (c \wedge d \wedge a) \vee (c \wedge a \wedge i) \vee (d \wedge a \wedge i) \vee (c \wedge d \wedge i \wedge a)$$

6. Las condiciones para que un banco apruebe un crédito de libre inversión son:

- a) Lo solicitado sea inferior al 30 % de lo devengado.
- b) No tener otros créditos activos.
- c) No haber estado en mora con otros créditos.
- d) Tener menos de 70 años de edad.

Haciendo asignación de sentencias así:

Letra sentencial	Sentencia
c	Crédito aprobado
s	Solicitar menos del 30 %
o	Tener otros créditos activos
m	Haber estado en mora
e	Edad inferior a 70 años

En este caso el listado representa condiciones que se deben cumplir todas, por lo que la expresión lógica para este enunciado es la siguiente:

$$c = s \wedge \neg o \wedge \neg m \wedge e$$

7. El llenado de un tanque con dos reactivos se termina (se cierran llaves de llenado) si:

- a) El nivel llega al tope permitido.
- b) Se vierten simultáneamente los dos reactivos.
- c) Si la salida del tanque está abierta, se vierte uno cualquiera de los reactivos y el nivel llega al tope.

La asignación de sentencias es:

Letra sentencial	Sentencia
t	Terminar llenado
n	Nivel a tope
r_1	Se vierte reactivo 1
r_2	Se vierte reactivo 2
s	Salida abierta

La expresión lógica para este enunciado es la siguiente:

$$t = n \vee (r_1 \wedge r_2) \vee (s \wedge n \wedge (r_1 \vee r_2))$$

2.3. TABLAS DE VERDAD

2.3.1. Definición

Una **tabla de verdad** es la representación explícita de verdad o falsedad de una proposición molecular. Se fundamenta en el hecho que toda proposición atómica es verdadera o falsa, (V o F, 1 o 0, Si o No), y en que los valores de verdad de una proposición molecular se pueden determinar por los valores de verdad de las proposiciones atómicas que la componen y las conectivas lógicas que las relacionan. El formato tabular conocido en nuestros días lo introdujo Ludwig Wittgenstein en su trabajo *Tractatus logico-philosophicus*, publicado en 1921, pero inicialmente fue desarrollada por Charles Sanders Peirce hacia 1880.

Una tabla de verdad consiste en un listado de todas las posibles combinaciones de los valores de verdad que se les puede asignar a cada una de las proposiciones atómicas que componen una proposición molecular. De lo anterior, si una fórmula lógica está compuesta por dos variables lógicas, entonces las combinaciones de sus valores de verdad son 4 en total, si una fórmula lógica está compuesta por tres variables lógicas entonces las posibles combinaciones son 8. En general, el número de posibles combinaciones para n variables lógicas es de 2^n .

En una tabla de verdad, a la izquierda se listan todas las posibles combinaciones de las variables lógicas y la derecha la evaluación respectiva del valor de verdad. Una forma práctica de obtener todas las combinaciones, sin que falte alguna, consiste en asignar a cada variable el peso en una cifra binaria, así, para tres variables p , q y r , se representan como la cifra binaria pqr donde p es el bit más significativo y r el menos significativo. Como el peso de r es $2^0 = 1$, su columna alterna entre valores de verdadero (V) y falso (F); para q su peso es $2^1 = 2$ por lo que su columna alterna cada dos valores y finalmente para p su peso es $2^2 = 4$ por lo que alterna cada 4 valores. En los sistemas digitales, se acostumbra iniciar la tabla con valor de falso y terminar con valor de verdadero.

2.3.2. Tablas de verdad de las conectivas lógicas

La tabla de verdad para cada una de las conectivas lógicas se muestra a continuación en la tabla 2.3, donde s y t son variables lógicas.

Tabla 2.3: Tablas de verdad de las conectivas lógicas

<table><tr><th>t</th><th>$\neg t$</th></tr><tr><td>F</td><td>V</td></tr><tr><td>V</td><td>F</td></tr></table>	t	$\neg t$	F	V	V	F	<table><tr><th>s</th><th>t</th><th>$s \wedge t$</th></tr><tr><td>F</td><td>F</td><td>F</td></tr><tr><td>F</td><td>V</td><td>F</td></tr><tr><td>V</td><td>F</td><td>F</td></tr><tr><td>V</td><td>V</td><td>V</td></tr></table>	s	t	$s \wedge t$	F	F	F	F	V	F	V	F	F	V	V	V	<table><tr><th>s</th><th>t</th><th>$s \vee t$</th></tr><tr><td>F</td><td>F</td><td>F</td></tr><tr><td>F</td><td>V</td><td>V</td></tr><tr><td>V</td><td>F</td><td>V</td></tr><tr><td>V</td><td>V</td><td>V</td></tr></table>	s	t	$s \vee t$	F	F	F	F	V	V	V	F	V	V	V	V
t	$\neg t$																																					
F	V																																					
V	F																																					
s	t	$s \wedge t$																																				
F	F	F																																				
F	V	F																																				
V	F	F																																				
V	V	V																																				
s	t	$s \vee t$																																				
F	F	F																																				
F	V	V																																				
V	F	V																																				
V	V	V																																				
Negador	Conjunción	Disyunción inclusiva																																				

<table><tr><th>s</th><th>t</th><th>$s \oplus t$</th></tr><tr><td>F</td><td>F</td><td>F</td></tr><tr><td>F</td><td>V</td><td>V</td></tr><tr><td>V</td><td>F</td><td>V</td></tr><tr><td>V</td><td>V</td><td>F</td></tr></table>	s	t	$s \oplus t$	F	F	F	F	V	V	V	F	V	V	V	F	<table><tr><th>s</th><th>t</th><th>$s \rightarrow t$</th></tr><tr><td>F</td><td>F</td><td>V</td></tr><tr><td>F</td><td>V</td><td>V</td></tr><tr><td>V</td><td>F</td><td>F</td></tr><tr><td>V</td><td>V</td><td>V</td></tr></table>	s	t	$s \rightarrow t$	F	F	V	F	V	V	V	F	F	V	V	V	<table><tr><th>s</th><th>t</th><th>$s \leftrightarrow t$</th></tr><tr><td>F</td><td>F</td><td>V</td></tr><tr><td>F</td><td>V</td><td>F</td></tr><tr><td>V</td><td>F</td><td>F</td></tr><tr><td>V</td><td>V</td><td>V</td></tr></table>	s	t	$s \leftrightarrow t$	F	F	V	F	V	F	V	F	F	V	V	V
s	t	$s \oplus t$																																													
F	F	F																																													
F	V	V																																													
V	F	V																																													
V	V	F																																													
s	t	$s \rightarrow t$																																													
F	F	V																																													
F	V	V																																													
V	F	F																																													
V	V	V																																													
s	t	$s \leftrightarrow t$																																													
F	F	V																																													
F	V	F																																													
V	F	F																																													
V	V	V																																													
Disyunción exclusiva	Implicador	Coimplicador																																													

En la tabla 2.3 se puede observar la forma como opera cada una de las conectivas, así: la negación invierte el valor de verdad, la conjunción es verdad cuando sus variables son ambas verdad, la disyunción inclusiva es falsa cuando sus variables son ambas falsas, la disyunción exclusiva es falsa cuando sus variables tienen valor de verdad igual, el implicador es falso únicamente cuando s es verdad y t es falso, el coimplicador es verdad cuando sus variables tienen valor de verdad igual.

2.3.3. Tablas de verdad de proposiciones moleculares

Para la evaluación de una proposición molecular se realiza la evaluación de proposiciones atómicas relacionadas por conectivas lógicas, teniendo presente la jerarquía impuesta por los mismos operadores o por los signos de agrupación. Se debe evaluar inicialmente los operadores de menor jerarquía y avanzar evaluando hacia los operadores de mayor jerarquía.

EJEMPLOS

1. Evaluar la tabla de verdad para $\neg s \vee t$:

s	t	$\neg s$	$\neg s \vee t$
F	F	V	V
F	V	V	V
V	F	F	F
V	V	F	V

Nótese como este último resultado es igual al implicador, por lo que $\neg s \vee t = s \rightarrow t$.

2. Evaluar la tabla de verdad para $(s \leftrightarrow t) \vee (s \oplus t)$:

s	t	$(s \leftrightarrow t)$	$(s \oplus t)$	$(s \leftrightarrow t) \vee (s \oplus t)$
F	F	V	F	V
F	V	F	V	V
V	F	F	V	V
V	V	V	F	V

3. Evaluar la tabla de verdad para $(r \wedge s) \vee t$:

r	s	t	$(r \wedge s)$	$(r \wedge s) \vee t$
F	F	F	F	F
F	F	V	F	V
F	V	F	F	F
F	V	V	F	V
V	F	F	F	F
V	F	V	F	V
V	V	F	V	V
V	V	V	V	V

4. Evaluar la tabla de verdad para $(s \wedge t) \wedge (s \oplus t)$:

s	t	$(s \wedge t)$	$(s \oplus t)$	$(s \wedge t) \wedge (s \oplus t)$
F	F	F	F	F
F	V	F	V	F
V	F	F	V	F
V	V	V	F	F

2.4. CLASIFICACIÓN DE LAS FÓRMULAS LÓGICAS

Las tablas de verdad de los ejemplos previos muestran tres clases diferentes de fórmulas lógicas, a saber:

De verdad indeterminada: también denominadas fórmulas lógicas de verdad consistente, son aquellas donde algunos de los valores de verdad de la fórmula lógica son verdad y otros son falsos.

Contradicción: también denominadas fórmulas lógicas inconsistentes, son aquellas donde todos los valores de verdad de la fórmula lógica son falsos. En estas fórmulas lógicas su valor de verdad FALSO no depende de los valores de verdad de las proposiciones que la forman.

Tautología: también denominada ley lógica, son aquellas donde todos los valores de verdad de la fórmula lógica son verdad. En estas fórmulas lógicas su valor de verdad VERDADERO no depende de los valores de verdad de las proposiciones que la forman.

2.5. FÓRMULAS EQUIVALENTES

Dos fórmulas lógicas son equivalentes si y sólo si tienen los mismos valores de verdad para toda combinación de valores de verdad de sus componentes atómicos. En caso contrario no son equivalentes.

EJEMPLO

Determinar si las siguientes fórmulas lógicas son equivalentes:

1. $(\neg p \vee r) \vee (p \rightarrow \neg q)$
2. $\neg p \vee \neg q \vee r$

Para verificar si son equivalentes se realiza la tabla de verdad de ambas fórmulas:

p	q	r	$\neg p$	$\neg q$	$\neg r$	$(\neg p \vee r)$	$(p \rightarrow \neg q)$	$(\neg p \vee r) \vee (p \rightarrow \neg q)$	$\neg p \vee \neg q$	$\neg p \vee \neg q \vee r$
F	F	F	V	V	V	V	V	V	V	V
F	F	V	V	V	F	V	V	V	V	V
F	V	F	V	F	V	V	V	V	V	V
F	V	V	V	F	F	V	V	V	V	V
V	F	F	F	V	V	F	V	V	V	V
V	F	V	F	V	F	V	V	V	V	V
V	V	F	F	F	V	F	F	F	F	F
V	V	V	F	F	F	V	F	V	F	V

De donde se comprueba que ambas fórmulas tienen los mismos valores de verdad para cada combinación de las variables lógicas.

2.5.1. Relación entre equivalencias y tautologías

Si dos fórmulas lógicas son equivalentes, la fórmula que se obtiene de operarlas con la conectiva bicondicional es una tautología. Además, si la fórmula lógica bicondicional es una tautología, las fórmulas lógicas componentes son equivalentes.

EJEMPLOS

1. En el ejemplo de la sección 2.5 se encontró que las fórmulas lógicas $(\neg p \vee r) \vee (p \rightarrow \neg q)$ y $\neg p \vee \neg q \vee r$ son equivalentes, entonces al operarlas con la conectiva bicondicional se debe obtener una tautología, como se muestra a continuación:

p	q	r	$(\neg p \vee r) \vee (p \rightarrow \neg q)$	$\neg p \vee \neg q \vee r$	$[(\neg p \vee r) \vee (p \rightarrow \neg q)] \leftrightarrow [\neg p \vee \neg q \vee r]$
F	F	F	V	V	V
F	F	V	V	V	V
F	V	F	V	V	V
F	V	V	V	V	V
V	F	F	V	V	V
V	F	V	V	V	V
V	V	F	F	F	V
V	V	V	V	V	V

2. La fórmula lógica $[(\neg p \wedge q) \wedge \neg p] \leftrightarrow \neg(q \rightarrow p)$ es una tautología, comprobar que sus fórmulas lógicas componentes $(\neg p \wedge q) \wedge \neg p$ y $\neg(q \rightarrow p)$ son equivalentes:

p	q	$\neg p$	$\neg q$	$(\neg p \wedge q)$	$(\neg p \wedge q) \wedge \neg p$	$\neg(q \rightarrow p)$	$[(\neg p \wedge q) \wedge \neg p] \leftrightarrow \neg(q \rightarrow p)$
F	F	V	V	F	F	F	V
F	V	V	F	V	V	V	V
V	F	F	V	F	F	F	V
V	V	F	F	F	F	F	V

2.5.2. Tautologías fundamentales

Razonamiento: Proceso que se realiza para obtener una demostración.

Demostración: Encadenamiento de proposiciones iniciales supuestas verdaderas, denominadas premisas, que permiten obtener una nueva proposición denominada conclusión.

Inferencia lógica: Conclusiones obtenidas luego de realizar un razonamiento.

La tabla 2.4 muestra una lista de tautologías que reciben nombres especiales. Estas tautologías se conocen como **reglas de la inferencia**. El razonamiento producto que da lugar a las inferencias lógicas solo es verdadero si se cumplen las siguientes condiciones:

1. Las premisas deben ser verdaderas.
2. Durante el proceso de deducción, las premisas se deben relacionar sujetas a las leyes de la lógica.

Tabla 2.4: Tautologías fundamentales

Tautología	Nombre
$p \vee \neg p$	Ley del medio excluido
$\neg (p \wedge \neg p)$	Ley de no contradicción
$((p \rightarrow q) \wedge p) \rightarrow q$	Modus ponendo ponens
$((p \rightarrow q) \wedge \neg q) \rightarrow \neg p$	Modus tollendo tollens
$((p \vee q) \wedge \neg p) \rightarrow q$	Silogismo Disyuntivo
$((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow r)$	Silogismo Hipotético

3. El conocimiento obtenido de proposiciones verdaderas preestablecidas (premisas), y aplicando las leyes de la lógica a esas premisas, se denomina conclusión.

2.6. CONECTIVAS LÓGICAS EN SISTEMAS DIGITALES

En los sistemas digitales se acostumbra usar la notación de Scholz, donde se emplean paréntesis, llaves y corchetes como símbolos de agrupación, los operadores monádicos preceden las variables y los diádicos se escriben entre las variables. Sin embargo, es usual emplear simbología diferentes para las conectivas lógicas, tal como se describe en la tabla 2.5.

Tabla 2.5: Conectivas lógicas en sistemas digitales

Conectiva	Notación Scholz	Sistemas Digitales
Conjunción	$p \wedge q$	$p \cdot q$ o pq
Disyunción inclusiva	$p \vee q$	$p + q$
Disyunción exclusiva	$p \leftrightarrow q$	$p \oplus q$
Coimplicador	$p \leftrightarrow q$	$p \odot q$
Negación	$\neg q$	$\bar{q}$

En el resto de este texto se hace uso de los símbolos para sistemas digitales, tal como se indican en la tabla 2.5. Se debe notar como el símbolo para la negación se posiciona sobre la variable lógica que afecta, los demás símbolos diádicos permanecen entre las variables tal como en el sistema Scholz; además, la conectiva implicador no se usa. La conectiva de conjunción se acostumbra omitir cuando no presenta posibilidad a doble interpretación, en similitud con el operador de multiplicación en la aritmética.

2.7. EJERCICIOS PROPUESTOS

1. Determinar el tipo de oración de los siguientes enunciados:
 - a) El número cinco es primo, pero el número 8 es par.
 - b) El sensor se activa ya sea que la temperatura esté caliente o la presión baje.
 - c) Estudiaré en la universidad a pesar de tener calificaciones deficientes.
 - d) ¡Casi me ganó la lotería!
 - e) ¿Estás en tu casa?
 - f) ¿Por qué no ha salido?
 - g) ¡Nos vemos en cinco minutos!
 - h) Abre la llave, así el tanque esté lleno.
 - i) Ya era el momento.
 - j) La impresora tiene buenos sentimientos.
2. Identificar las proposiciones atómicas en los siguientes enunciados:
 - a) El motor está encendido y la banda transportadora opera normalmente.
 - b) El ascensor funciona si y sólo si se presiona el piso deseado y la puerta se cierra.
 - c) Una sola motobomba operando es condición necesaria y suficiente para tener presión.
 - d) Temperatura muy alta, velocidad muy baja o flujo de gas no continuo implican producto no aceptable.
 - e) La planta eléctrica opera si y solo si tiene combustible, no está obstruida la refrigeración, tiene aceite lubricante en cantidad y condición correcta y la batería está cargada.
3. Identificar cada una de las conectivas lógicas que enlazan las proposiciones atómicas de los ejercicios del punto anterior.
4. Continuando con los enunciados del punto 1, plantearlos usando el alfabeto definido.
5. Encontrar la tabla de verdad para cada una de las expresiones lógicas halladas en la Sección 2.2.6.
6. Encontrar la fórmula lógica que representa el siguiente enunciado. La alarma de prevención en un ascensor suena en alguno de los siguientes casos:
 - a) Si existe sobre cupo.
 - b) Si con orden de cerrar puerta, la puerta permanece abierta y se ha solicitado viaje a un piso.
 - c) Si se presiona el botón de emergencia en cualquier momento.
 - d) Si estando en viaje a un piso se pide abrir puerta.
 - e) Si estando en viaje a un piso se abre la puerta.

7. Encontrar la fórmula lógica que representa el siguiente enunciado. En una impresora, la luz de prevención se enciende si:
 - a) Se atasca el papel, o
 - b) Si no hay papel en la bandeja de alimentación, o
 - c) Si se cierra la bandeja sin terminar la impresión, o
 - d) Si queda poca tinta.
8. Encontrar la fórmula lógica que representa el siguiente enunciado. La alarma auditiva de una UPS se activa en alguno de los siguientes casos:
 - a) Si no hay alimentación y queda poca carga en las baterías.
 - b) Si no hay alimentación.
 - c) Si no funciona el sistema de carga de baterías.
 - d) Si se presiona apagar con carga presente.
 - e) Si se pierde la carga presente.
9. Encontrar la fórmula lógica que representa el siguiente enunciado. El funcionamiento de una banda transportadora depende de tres botones, A, B y C. Si se presiona A pero no B la operación es a izquierda, pero si se presiona B y no A la operación es a derecha, si se presionan ambos se para la banda. Si se presionan los tres botones simultáneamente todo sigue como esté en ese instante.
10. Comprobar que las fórmulas lógicas de la tabla 2.4 son tautologías.
11. Comprobar que al aplicar el operador de negación a las tautologías de la tabla 2.4 se obtiene una contradicción.
12. Comprobar la clase de fórmula lógica que se obtiene al operar con las conectivas diádicas, a dos tautologías de la tabla 2.4.
13. Comprobar la clase de fórmula lógica que se obtiene al operar con las conectivas diádicas, a dos contradicción de las obtenidas en la pregunta 11.
14. Comprobar la clase de fórmula lógica que se obtiene al operar con las conectivas diádicas, a una tautología con una contradicción de las obtenidas en la pregunta 11.

3

CAPÍTULO
TRES

Funciones lógicas

Este capítulo emplea los temas vistos previamente con el fin de poder llevar enunciados no formalizados a enunciados formalizados y poder explorar todas sus posibles formas de representación. Las posibles representaciones incluyen formas gráficas, explícitas e implícitas que permiten expresar soluciones según el contexto donde se empleen. Así, por ejemplo, en electrónica es común requerir una implementación mediante compuertas lógicas que dan origen a sistemas electrónicos físicos; en automatización y en algoritmia normalmente se requiere de una expresión en forma de ecuación para implementar como código; en aplicaciones de análisis sobre diseños existentes, con el fin de explorar su funcionamiento, puede ser necesario encontrar tablas o gráficos de comportamiento. Lo anterior demuestra la necesidad de saber combinar todas las posibles formas de representación, poder manipularlas algebraicamente e incluso poder expresarlas en formas estandarizadas, de ser necesario. Al entrar en este capítulo, el lector se percatará que los sistemas de numeración ponderados y el álgebra booleana, tienen un papel importante con el fin de permitir establecer equivalencias entre formas de representación de sistemas lógicos.

3.1. DEFINICIONES

-
- Variable lógica:** *Es un símbolo algebraico que en cualquier instante de tiempo t puede tomar uno y sólo uno de dos valores posibles, 0 ó 1. También se denomina variable proposicional, variable sentencial o literal.*
- Función lógica:** *Expresión que describe las operaciones lógicas entre sus variables lógicas. Del mismo modo, una función lógica sólo puede ser igual a 0 ó 1.*
-

Al igual que el álgebra aritmética, el álgebra booleana se basa en un conjunto de reglas que se derivan de un pequeño número de supuestos básicos. Los valores lógicos implican elementos que toman uno de dos valores, por lo tanto, una variable lógica es mutuamente excluyente. Ejemplos de valores mutuamente excluyentes son:

- Encendido - Apagado
- Verdadero - Falso
- SI - NO
- 1 - 0
- Abierto - Cerrado
- En marcha - Detenido
- Caliente - Frío

La interacción entre variables lógicas define las funciones lógicas, que toman la forma:

$$Z = f(A)$$

Donde Z y A son variables lógicas y f la función que las relaciona. A diferencia del álgebra aritmética, en el álgebra booleana no existen infinitas funciones f . Numéricamente, las variables Z y A están representadas en base 2, luego, para $Z = f(A)$ sólo existen 2^x funciones, siendo x el número de combinaciones de valores de verdad entre variables, que es igual a $2^{\#variables}$. Resumiendo:

$$\#funciones = 2^{2^{\#variables}}$$

Los sistemas lógicos vistos como cajas negras están representados por el esquema de la figura 3.1.

Figura 3.1: Sistema lógico de 1 entrada y 1 salida

Este es un esquema simple con una sola variable de entrada y una sola salida. Como $Z = f(A)$ tiene 1 variable dependiente, sólo existen 4 funciones posibles f que relacionan Z y A , las cuales se describen a continuación:

A	Z_1	Z_2	Z_3	Z_4
0	0	0	1	1
1	0	1	0	1

En este caso, es fácil deducir que $Z_1 = f(A) = 0$, $Z_2 = f(A) = A$, $Z_3 = f(A) = \bar{A}$ y $Z_4 = f(A) = 1$. Z_1 es una contradicción, Z_2 sigue la variable de entrada, Z_3 es la negación y Z_4 es una tautología. Un ejemplo de sistema lógico con dos entradas y una salida se muestra en la figura 3.2.

Figura 3.2: Sistema lógico con 2 entradas y 1 salida

Donde $Z = f(A, B)$ y f es una función de dos variables. Luego, existen $2^2 = 4$ posibles combinaciones de entrada y $2^4 = 16$ posibles funciones, como se puede observar a continuación:

A	B	Z ₁	Z ₂	Z ₃	Z ₄	Z ₅	Z ₆	Z ₇	Z ₈	Z ₉	Z ₁₀	Z ₁₁	Z ₁₂	Z ₁₃	Z ₁₄	Z ₁₅	Z ₁₆
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

En este último caso, se puede observar que Z_1 es una contradicción, Z_2 es la AND, Z_7 es la XOR, Z_8 es la OR, Z_9 es la NOR, Z_{10} es la XNOR, Z_{11} es la NOT de la variable B, Z_{13} es la NOT de la variable A, Z_{15} es la NAND y Z_{16} es una tautología. Se observa, que solo con dos variables de entrada, ya algunas de las posibles funciones no tienen nombre, o no se les puede relacionar de forma directa con conectivos lógicos y es necesario describirlas por su función lógica, que relaciona la interacción entre las variables involucradas, como por ejemplo, es fácilmente comprobable que para $Z_3 = f(A, B) = A \cdot \bar{B}$. El estudio de estas interacciones se desarrolla en los apartados siguientes de este capítulo.

Similarmente, para un sistema lógica con tres entradas y una salida, existen $2^3 = 8$ posibles combinaciones de entrada y $2^8 = 256$ posibles funciones f .

La representación explícita de la función f se hace mediante una tabla de verdad, pero también puede ser representada implícitamente, utilizando una representación simbólica en forma de fórmula lógica, donde las proposiciones atómicas son representadas por sus literales y las conectivas lógicas con sus símbolos respectivos. Todas las posibles funciones para un determinado número de variables de entrada se pueden describir en función de tres funciones básicas que se describen a continuación.

3.1.1. Funciones lógicas básicas

Existen tres funciones lógicas básicas, a partir de las cuales se pueden expresar todas las demás funciones, a saber: **AND**, **OR**, y **NOT**. A continuación se describen, para cada una de estas tres funciones, sus aspectos más relevantes.

AND

Se muestra la función lógica de la AND y su simbología estándar en la figura 3.3.

$$\begin{aligned} Z = f(A, B) &= A \cdot B \\ &= A \& B = A \wedge B = AB \end{aligned}$$

Figura 3.3: Simbología estándar para la AND

En general: $Z = f(A, B, C, \dots) = ABC\dots$, y cumple la ley asociativa y conmutativa:

$$\begin{aligned} Si \ Z = f(A, B, C) &= ABC = (AB)C \\ Cumple \ leyes &= A(CB) = A(BC) \\ Asociativa &= C(BA) = CBA \\ y \ Conmutativa &= CAB \end{aligned}$$

La representación en diagrama de contactos (o equivalente circuital por contactos de conmutación) se hace mediante contactos normalmente abiertos en serie, ya que para dar continuidad a un circuito se requiere de la activación simultánea de los contactos, como se observa en la figura 3.4. Esta última figura también contiene la tabla de verdad respectiva.

#	A	B	Z
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

Figura 3.4: Diagrama de contactos y tabla de verdad para la AND

OR

Se muestra la función lógica de la OR y su simbología estándar en la figura 3.5.

$$Z = f(A, B) = A + B = A \vee B$$

Figura 3.5: Simbología estándar para la OR

En general: $Z = f(A, B, C, \dots) = A + B + C \dots$, y cumple la ley asociativa y conmutativa:

$$\begin{aligned} \text{Si } Z = f(A, B, C) &= A + B + C = (A + B) + C \\ \text{Cumple leyes} &= A + (C + B) = A + (B + C) \\ \text{Asociativa} &= C + (B + A) = C + B + A \\ \text{y Conmutativa} &= C + A + B \end{aligned}$$

La representación en diagrama de contactos se hace mediante contactos normalmente abiertos en paralelo, ya que para dar continuidad a un circuito se requiere de la activación de uno cualquiera de los contactos, según la tabla de verdad y diagrama que se observa en la figura 3.6.

Figura 3.6: Diagrama de contactos y tabla de verdad para la OR

NOT

Ahora, se muestra la función lógica de la NOT y su simbología estándar en la figura 3.7. Como esta es una función de una sola variable, no se aplican las propiedades asociativa y conmutativa.

$$Z = f(A) = \neg A = \overline{A} \text{ (Negación)}$$

Figura 3.7: Simbología estándar para la NOT

La representación en diagrama de contactos se hace mediante un contacto normalmente cerrado, ya que la continuidad del circuito es inversa a la activación del mismo contacto, como se observa en la figura 3.8 y la respectiva tabla de verdad.

Figura 3.8: Diagrama de contactos y tabla de verdad para la NOT

3.1.2. Funciones lógicas derivadas

Son cuatro funciones lógicas que se obtienen por combinación simple de las tres básicas y que por su utilidad reciben nombre. Ellas son: **NAND**, **NOR**, **XOR**, **XNOR**. A continuación se describe, para cada una de estas cuatro funciones, su función lógica, la simbología estándar, algunas propiedades, el diagrama de contactos y su tabla de verdad.

NAND

Se muestra la función lógica de la NAND y su simbología estándar en la figura 3.9.

$$Z = f(A, B) = \overline{A \cdot B}$$

Figura 3.9: Simbología estándar para la NAND

En general: $Z = f(A, B, C, \dots) = \overline{A \cdot B \cdot C \dots}$ y la propiedad conmutativa se cumple, ya que:

$$Z = f(A, B, C, \dots) = \overline{A \cdot B \cdot C \dots} = \overline{A \cdot C \cdot B \dots} = \overline{B \cdot A \cdot C \dots}$$

La propiedad asociativa no se cumple, ya que:

$$Z = f(A, B, C, \dots) = \overline{A \cdot B \cdot C \dots} \neq \overline{(\overline{A \cdot B}) \cdot C} \neq \overline{(A \cdot B) \cdot C}$$

Para poder aplicar la propiedad asociativa, se debe compensar la afectación que ocasiona asociar términos negados, teniendo presente que la negación de una negación es la misma expresión original no negada:

$$Z = f(A, B, C, \dots) = \overline{A \cdot B \cdot C \dots} = \overline{\overline{\overline{A \cdot B}} \cdot \overline{C}} = \overline{\overline{A \cdot B}} \cdot \overline{\overline{C}}$$

La representación en lógica conmutada consta de contactos normalmente cerrados en paralelo, ya que para dar continuidad a un circuito se requiere la no activación de alguno de los contactos. La figura 3.10 muestra el respectivo diagrama y la tabla de verdad asociada.

Figura 3.10: Diagrama de contactos y tabla de verdad para la NAND

NOR

La figura 3.11 muestra la función lógica de la NOR y su simbología estándar.

$$Z = f(A, B) = \overline{A + B}$$

Figura 3.11: Simbología estándar para la NOR

En general: $Z = f(A, B, C, \dots) = \overline{A + B + C + \dots}$ y la propiedad conmutativa se cumple, ya que:

$$Z = f(A, B, C, \dots) = \overline{A + B + C + \dots} = \overline{A + C + B + \dots} = \overline{B + A + C + \dots}$$

La propiedad asociativa no se cumple, ya que:

$$Z = f(A, B, C, \dots) = \overline{A + B + C \dots} \neq \overline{\overline{(A + B)} + C} \neq \overline{(A + B) + \overline{C}}$$

Para poder aplicar la propiedad asociativa, se debe compensar la afectación que ocasiona asociar términos negados, en igual forma que se realizó con la NAND:

$$Z = f(A, B, C, \dots) = \overline{A + B + C \dots} = \overline{\overline{\overline{A + B} + \overline{C}}}$$

La representación en lógica conmutada consiste de contactos normalmente cerrados en serie, ya que para dar continuidad a un circuito se requiere la no activación de ambos contactos, tal como se puede ver en la tabla de verdad y diagrama de la figura 3.12.

#	A	B	Z
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	0

Figura 3.12: Diagrama de contactos y tabla de verdad para la NOR

XOR (OR exclusiva)

A continuación, en la figura 3.13, se muestra la función lógica de la XOR y su simbología estándar.

$$Z = f(A, B) = A \oplus B$$

Figura 3.13: Simbología estándar para la XOR

En general: $Z = f(A, B, C, \dots) = A \oplus B \oplus C \oplus \dots$, y cumple la ley asociativa y conmutativa:

$$\begin{aligned} \text{Si } Z = f(A, B, C) &= A \oplus B \oplus C = (A \oplus B) \oplus C \\ \text{Cumple leyes} &= A \oplus (C \oplus B) = A \oplus (B \oplus C) \\ \text{Asociativa} &= C \oplus (B \oplus A) = C \oplus B \oplus A \\ \text{y Conmutativa} &= C \oplus A \oplus B \end{aligned}$$

La representación en lógica conmutada es el paralelo de dos series, en una serie un contacto es normalmente abierto y el otro normalmente cerrado, y en la otra serie se invierten. Para dar continuidad a un circuito se requiere la no activación de una variable y la activación de la otra, tal como lo enseña la tabla de verdad y diagrama de la figura 3.14.

Figura 3.14: Diagrama de contactos y tabla de verdad para la XOR

XNOR

Finalmente, se muestra la función lógica de la XNOR y su simbología estándar en la figura 3.15.

$$Z = f(A, B) = \overline{A \oplus B} = A \odot B$$

Figura 3.15: Simbología estándar para la XNOR

En general: $Z = f(A, B, C, \dots) = A \odot B \odot C \odot \dots = \overline{A \oplus B \oplus C}$ y la propiedad conmutativa se cumple, ya que:

$$Z = f(A, B, C, \dots) = \overline{A \oplus B \oplus C} = \overline{A \oplus B \oplus C} = \overline{A \oplus B \oplus C}$$

La propiedad asociativa no se cumple, ya que:

$$Z = f(A, B, C, \dots) = \overline{A \oplus B \oplus C \dots} \neq \overline{\overline{(A \oplus B) \oplus C}} \neq \overline{(A \oplus B) \oplus C}$$

Para poder aplicar la propiedad asociativa, se debe compensar la afectación que ocasiona asociar términos negados, así:

$$Z = f(A, B, C, \dots) = \overline{A \oplus B \oplus C \dots} = \overline{\overline{\overline{(A \oplus B) \oplus C}}}$$

La representación en lógica conmutada es el paralelo de dos series, en una serie los contactos son normalmente abiertos y en la otra serie son normalmente cerrados. Para dar continuidad a un circuito se requiere ya sea la activación, o la no activación, simultánea de los contactos. Lo descrito se puede observar en la tabla de verdad y diagrama de contactos de la figura 3.16.

Figura 3.16: Diagrama de contactos y tabla de verdad para la XNOR

3.2. REPRESENTACIÓN DE FUNCIONES LÓGICAS

3.2.1. Representación implícita

La misma representación simbólica corresponde con la representación implícita de las funciones lógicas, donde cada proposición molecular se representa por su letra sentencial, cada conectiva lógica por su símbolo y se emplean signos de agrupación con el fin de establecer jerarquías entre operadores; todo en estricto cumplimiento de las reglas de la sintaxis lógica, que permite la elaboración de fórmulas bien formadas.

3.2.2. Representación explícita

La representación por tabla de verdad corresponde con la representación explícita de las funciones lógicas, donde de forma unívoca se asigna un valor de verdad para la función lógica de acuerdo con cada posible combinación de los valores de verdad de las variables proposicionales.

3.2.3. Diagrama de tiempo

Los diagramas de tiempo muestran el comportamiento de la salida de una función lógica ante diferentes valores de entrada, que varían en el dominio del tiempo. Normalmente, para realizar un diagrama de tiempo se colocan a la entrada del sistema todas y cada una de las posibles combinaciones de entrada, preferiblemente en orden y cada determinado intervalo de tiempo, y se grafican paralelamente con la salida que cada combinación de entrada produce. La figura 3.17 muestra un diagrama de tiempo para la función lógica AND.

Figura 3.17: Ejemplo de diagrama de tiempo

3.2.4. Diagrama lógico

Una función lógica compleja puede ser representada algebraicamente por una combinación lineal de las funciones básicas o derivadas. También puede ser representada gráficamente utilizando los símbolos estandarizados o los símbolos de compuerta. Al diagrama resultante se le denomina **diagrama lógico**. Un diagrama lógico representa una función que puede ser implementada físicamente en un **circuito lógico**. La figura 3.18 muestra un diagrama lógico para la función dada.

Figura 3.18: Ejemplo de diagrama lógico

EJEMPLO

Encontrar la representación explícita, el diagrama de tiempo y el diagrama lógico para la siguiente función:

$$Z = f(x, y, z) = (xy) \odot (xz + \bar{x})$$

La representación explícita, o tabla de verdad, se muestra en la tabla 3.1:

 Tabla 3.1: Tabla de verdad para $Z = (xy) \odot (xz + \bar{x})$

x	y	z	$\bar{x}$	xy	xz	$xz + \bar{x}$	$(xy) \odot (xz + \bar{x})$
0	0	0	1	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	1	0	0	1	0
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	1	0	0	1	1	0
1	1	0	0	1	0	0	0
1	1	1	0	1	1	1	1

El diagrama de tiempos se muestra en la figura 3.19:

Figura 3.19: Diagrama de tiempos para $Z = (xy) \odot (xz + \bar{x})$

El diagrama lógico se muestra en la figura 3.20.

Figura 3.20: Diagrama lógico para $Z = (xy) \odot (xz + \bar{x})$

3.3. MÉTODOS ALGEBRAICOS DE ANÁLISIS Y SÍNTESIS DE FUNCIONES LÓGICAS

Álgebra	Es un conjunto de postulados que resume los elementos y propiedades básicas del razonamiento lógico.
Booleana :	
Postulado:	También conocido como axioma, es una fórmula bien formada que se admite sin demostración y es punto de partida para otros razonamientos y demostraciones.
Teorema:	Afirmación demostrable mediante un marco lógico consistente de un conjunto de postulados y un proceso de inferencia. Un teorema se puede derivar desde postulados u otros teoremas ya demostrados.

En 1849 George Boole publicó el trabajo *An investigation into the laws of thought, on which are founded the mathematical theories of logic and probability* donde se sentaron las bases de lo que hoy se conoce como **Álgebra Booleana**.

Por este mismo trabajo, se les ha dado a las variables lógicas el nombre de **variables booleanas**. Los postulados del sistema algebraico Booleano son los siguientes:

Postulado 1: Definición

Sea el sistema algebraico $(\mathbb{B}, +, \cdot)$ un conjunto cerrado, distributivo y complementado tal que si $\{a, b \in \mathbb{B}\}$, entonces $\{a + b \in \mathbb{B}\}$ y $\{a \cdot b \in \mathbb{B}\}$.

Postulado 2: Elementos Neutros

$$\{\bigwedge a \in \mathbb{B} \mid a + 0 = a\}$$

$$\{\bigwedge a \in \mathbb{B} \mid a \cdot 1 = a\}$$

Postulado 3: Conmutatividad

$$\{\bigwedge (a, b) \in \mathbb{B} \mid a + b = b + a\}$$

$$\{\bigwedge (a, b) \in \mathbb{B} \mid a \cdot b = b \cdot a\}$$

Postulado 4: Asociatividad

$$\{\bigwedge (a, b, c) \in \mathbb{B} \mid a + (b + c) = (a + b) + c\}$$

$$\{\bigwedge (a, b, c) \in \mathbb{B} \mid a \cdot (b \cdot c) = (a \cdot b) \cdot c\}$$

Postulado 5: Distributividad

De $+$ sobre $\cdot$ y de $\cdot$ sobre $+$

$$\{\bigwedge (a, b, c) \in \mathbb{B} \mid a + (b \cdot c) = (a + b) \cdot (a + c)\}$$

$$\{\bigwedge (a, b, c) \in \mathbb{B} \mid a \cdot (b + c) = (a \cdot b) + (a \cdot c)\}$$

Postulado 6: Complemento

$$\{\bigwedge a \in \mathbb{B} \bigvee \bar{a} \in \mathbb{B} \mid a + \bar{a} = 1\}$$

$$\{\bigwedge a \in \mathbb{B} \bigvee \bar{a} \in \mathbb{B} \mid a \cdot \bar{a} = 0\}$$

3.3.1. Diagramas de Venn

Un diagrama de Venn es la representación en contorno cerrado de un conjunto, utilizando círculos, cuadrados, elipses, etc. En analogía con el álgebra de Boole, el conjunto universal representa un valor de verdad verdadero, el conjunto vacío representa un valor de verdad falso, una variable lógica a se indica por el área dentro de su contorno y la negación, o complemento, de la misma variable lógica se indica por el área fuera de su contorno, tal como se indica en la figura 3.21. La operación de intersección de conjuntos es análoga a la operación de conjunción (AND) y la operación de unión de conjuntos es análoga a la disyunción inclusiva (OR). Con base en las anteriores analogías, debe ser posible observar sobre diagramas de Venn representaciones de las funciones lógicas, toda vez que las tres operaciones básicas AND, OR, NOT son análogas respectivamente a la intersección, unión y complemento de conjuntos; además toda operación lógica derivada se puede expresar en términos de estas tres funciones básicas.

La figura 3.21 también muestra una comprobación del postulado 6, donde $a + \bar{a} = 1$ y $a\bar{a} = 0$.

Figura 3.21: Analogía entre diagramas de Venn y álgebra de Boole

EJEMPLO

Verificación del Postulado 5. Para ello, se debe demostrar que $A + BC = (A + B)(A + C)$. Se inicia mostrando las áreas relacionadas para los conjuntos A y B .

Ahora para tres variables, se muestra el área equivalente de $A + BC$, que es el lado izquierdo de la igualdad en estudio:

Luego, se comprueba el área de $(A + B)(A + C)$, o el lado derecho de la igualdad:

Por tanto, se demuestra que las áreas de los lados izquierdo y derecho son iguales, y que $A + BC = (A + B)(A + C)$.

3.3.2. Principio de la dualidad

Sea p una expresión algebraica booleana válida, luego existe pd la **expresión dual** de p , que también es una expresión booleana válida, o formalmente:

$$\left\{ \bigwedge p \in \mathbb{B} \bigvee pd \in \mathbb{B} \mid pd \in \mathbb{B} \right\}$$

La existencia de estas dos expresiones es lo que se denomina el principio de la dualidad, en el cual las dos expresiones, p y pd , son válidas, pero no necesariamente iguales entre sí. La expresión dual se obtiene:

- Reemplazando los operadores $\cdot$ por $+$
- Reemplazando los operadores $+$ por $\cdot$
- Reemplazando los 0 por 1
- Reemplazando los 1 por 0
- Preservando los paréntesis

EJEMPLOS

1. Si $p : \{a + (bc) = (a + b)(a + c)\}$
Entonces $pd : \{a \cdot (b + c) = (a \cdot b) + (a \cdot c)\}$
2. Si $p : \{a + 0 = a\}$
Entonces $pd : \{a \cdot 1 = a\}$
3. Si $p : \{a + \bar{a} = 1\}$
Entonces $pd : \{a \cdot \bar{a} = 0\}$

Se debe observar que en los Postulados 2 a 6, los pares de resultados son obtenidos mediante la aplicación del principio de la dualidad. Además es importante resaltar nuevamente que el principio de dualidad solo asegura la validez del postulado dual, más no su equivalencia con el postulado original; esto se puede observar con claridad en el ejemplo 3 visto previamente, donde las dos expresiones $a + \bar{a} = 1$ y $a \cdot \bar{a} = 0$ son válidas más no equivalentes ($a + \bar{a} \neq a \cdot \bar{a}$); para el ejemplo 2 las expresiones $a + 0 = a \cdot 1 = a$ sí son equivalentes.

3.3.3. Teoremas del álgebra Booleana**T1: Idempotencia**

$$\text{a) } a + a = a \quad \text{b) } a \cdot a = a$$

Demostración parte a:

$$\begin{aligned} a + a &= 1 \cdot (a + a) \\ &= (a + \bar{a})(a + a) \\ &= a + a\bar{a} \\ &= a + 0 \\ &= a \end{aligned}$$

T2: Modulación

$$\text{a) } a + 1 = 1 \quad \text{b) } a \cdot 0 = 0$$

Demostración parte a:

$$\begin{aligned} a + 1 &= 1 \cdot (a + 1) \\ &= (a + \bar{a})(a + 1) \\ &= a + \bar{a} \cdot 1 \\ &= a + \bar{a} \\ &= 1 \end{aligned}$$

T3: Involución

$$\bar{\bar{a}} = a$$

T4: Absorción, Forma 1

a) $a + ab = a$

b) $a(a + b) = a$

Demostración parte a:

$$\begin{aligned} a + ab &= a(1 + b) \\ &= a \cdot 1 \\ &= a \end{aligned}$$

T5: Absorción, Forma 2

a) $a + \bar{a}b = a + b$

b) $a(\bar{a} + b) = ab$

T6: Absorción, Forma 3

a) $ab + a\bar{b} = a$

b) $(a + b)(a + \bar{b}) = a$

T7: Absorción, Forma 4

a) $ab + a\bar{b}c = ab + ac$

b) $(a + b)(a + \bar{b} + c) = (a + b)(a + c)$

T8: Teorema de DeMorgan

a) $\overline{a + b} = \bar{a} \cdot \bar{b}$ y de forma generalizada $\overline{a + b + c + \dots} = \bar{a} \cdot \bar{b} \cdot \bar{c} \cdot \dots$

b) $\overline{a \cdot b} = \bar{a} + \bar{b}$ y de forma generalizada $\overline{a \cdot b \cdot c \cdot \dots} = \bar{a} + \bar{b} + \bar{c} + \dots$

Demostración parte a.

Sea $X = a + b$ y $Y = \bar{a} \cdot \bar{b}$, luego se debe demostrar que $\overline{X} = Y$; para ello se debe cumplir:
 $X \cdot Y = 0$ y $X + Y = 1$, dado que $\overline{X \cdot X} = 0$ y $X + \overline{X} = 1$.

$$\begin{aligned} X + Y &= (a + b) + (\bar{a}\bar{b}) \\ &= a + b + \bar{a}\bar{b} \\ &= a + \bar{a}\bar{b} + b \\ &= a + \bar{b} + b \\ &= a + 1 \\ &= 1 \end{aligned}$$

$$\begin{aligned} X \cdot Y &= (a + b)(\bar{a}\bar{b}) \\ &= a\bar{a}\bar{b} + \bar{a}b\bar{b} \\ &= 0 \end{aligned}$$

T9: Consenso

$$\text{a) } ab + \bar{a}c + bc = ab + \bar{a}c$$

$$\text{b) } (a + b)(\bar{a} + c)(b + c) = (a + b)(\bar{a} + c)$$

Demostración parte b.

$$\begin{aligned} (a + b)(\bar{a} + c)(b + c) &= (a + b)(\bar{a} + c)(b + c) \cdot 1 \\ &= (a + b)(\bar{a} + c)(b + c)(a + \bar{a}) \\ &= (a + b)(\bar{a} + c)(a + b + c)(\bar{a} + b + c) \\ &= (a + b)(a + b + c)(\bar{a} + c)(\bar{a} + b + c) \\ &= (a + b)(\bar{a} + c) \end{aligned}$$

T10: Teorema de desarrollo de Shannon

$$\text{a) } f(x_1, x_2, x_3, \dots, x_n) = x_1 \cdot f(1, x_2, x_3, \dots, x_n) + \bar{x}_1 \cdot f(0, x_2, x_3, \dots, x_n)$$

$$\text{b) } f(x_1, x_2, x_3, \dots, x_n) = [x_1 + f(0, x_2, x_3, \dots, x_n)] \cdot [\bar{x}_1 + f(1, x_2, x_3, \dots, x_n)]$$

EJEMPLOS

$$1. Z = ab + \bar{b}ac$$

$$\begin{aligned} Z &= ab + \bar{b}ac \\ &= ab + \bar{a}bc \\ &= ab + c \end{aligned}$$

$$2. r = \bar{x}(x + y) + \bar{z} + \bar{z}y$$

$$\begin{aligned} r &= \bar{x}(x + y) + \bar{z} + \bar{z}y \\ &= \bar{x}x + \bar{x}y + \bar{z} + \bar{z}y \\ &= \bar{x}x + \bar{x}y + \bar{z} + y \\ &= \bar{x}y + \bar{z} + y \\ &= y + \bar{x}y + \bar{z} \\ &= y + \bar{z} \end{aligned}$$

$$3. Z = AB + A(B + C) + B(B + C)$$

$$\begin{aligned} Z &= AB + A(B + C) + B(B + C) \\ &= AB + AB + AC + BB + BC \\ &= AB + AC + B + BC \\ &= AB + AC + B \\ &= B + AC \end{aligned}$$

$$4. U = \overline{AB + AC} + \overline{ABC}$$

$$\begin{aligned} U &= \overline{AB + AC} + \overline{ABC} \\ &= \overline{AB} \cdot \overline{AC} + \overline{ABC} \\ &= (\overline{A} + \overline{B})(\overline{A} + \overline{C}) + \overline{A}\overline{B}\overline{C} \\ &= \overline{A}\overline{A} + \overline{A}\overline{C} + \overline{A}\overline{B} + \overline{B}\overline{C} + \overline{A}\overline{B}\overline{C} \\ &= \overline{A} + \overline{A}\overline{C} + \overline{A}\overline{B} + \overline{B}\overline{C} \\ &= \overline{A} + \overline{A}\overline{B} + \overline{B}\overline{C} \\ &= \overline{A} + \overline{B}\overline{C} \end{aligned}$$

$$5. t = (a + b) (a + b + c) (\bar{a} + c + cb) (b + c + a + \bar{a})$$

$$\begin{aligned} t &= (a + b) (a + b + c) (\bar{a} + c + cb) (b + c + a + \bar{a}) \\ &= (a + b) (\bar{a} + c + cb) (b + c + a + \bar{a}) \\ &= (a + b) (\bar{a} + c) (b + c + a + \bar{a}) \\ &= (a + b) (\bar{a} + c) (b + c + 1) \\ &= (a + b) (\bar{a} + c) \end{aligned}$$

$$6. X = (\overline{AB + AB}) + (\overline{AB + AB})$$

$$\begin{aligned} X &= (\overline{AB + AB}) + (\overline{AB + AB}) \\ &= A \cdot f(1, B) + \bar{A} \cdot f(0, B) \\ &= A \cdot [(\bar{1}B + 1\bar{B}) + (\overline{\bar{1}B + 1\bar{B}})] + \bar{A} \cdot [(\bar{0}B + 0\bar{B}) + (\overline{\bar{0}B + 0\bar{B}})] \\ &= A \cdot [\bar{B} + B] + \bar{A} \cdot [B + \bar{B}] \\ &= A \cdot [1] + \bar{A} \cdot [1] \\ &= A + \bar{A} \\ &= 1 \end{aligned}$$

$$7. s = AB + A\overline{CB} + A\overline{BCD} + AC\overline{D} + A\overline{B}$$

$$\begin{aligned} s &= AB + A\overline{CB} + A\overline{BCD} + AC\overline{D} + A\overline{B} \\ &= AB + A\overline{CB} + A\overline{BC} + AC\overline{D} + A\overline{B} \\ &= A + A\overline{B} + AC\overline{D} \\ &= A \end{aligned}$$

3.3.4. Suficiencia de la NAND

Toda función lógica puede ser representada por una combinación lineal de sólo funciones NAND. Para demostrar esto, basta con implementar las tres funciones básicas utilizando sólo la función NAND, tal como se muestra en las figuras 3.22, 3.23 y 3.24.

1. NOT

$$\bar{A} = \bar{A} + \bar{A} = \overline{A \cdot A}$$

Entonces,

$$A \text{ --- } \neg \text{ --- } \bar{A} \quad \equiv \quad A \text{ --- } \text{NAND} \text{ --- } \bar{A}$$

Figura 3.22: Función NOT implementada con NAND

2. OR

$$A + B = \overline{\overline{A + B}} = \overline{\overline{A} \cdot \overline{B}}$$

Entonces,

$$\begin{matrix} A \\ B \end{matrix} \text{ --- } \vee \text{ --- } A + B \quad \equiv \quad \begin{matrix} A \\ B \end{matrix} \text{ --- } \text{NAND} \text{ --- } \text{NAND} \text{ --- } A + B$$

Figura 3.23: Función OR implementada con NAND

3. AND

$$A \cdot B = \overline{\overline{A \cdot B}}$$

Entonces,

$$\begin{matrix} A \\ B \end{matrix} \text{ --- } \wedge \text{ --- } A \cdot B \quad \equiv \quad \begin{matrix} A \\ B \end{matrix} \text{ --- } \text{NAND} \text{ --- } \neg \text{ --- } A \cdot B$$

Figura 3.24: Función AND implementada con NAND

3.3.5. Suficiencia de la NOR

Toda función lógica puede ser representada por una combinación lineal de sólo funciones NOR. Para demostrar esto, basta con implementar las tres funciones básicas utilizando sólo la función NOR, tal como se muestra en las figuras 3.25, 3.26 y 3.27.

1. NOT

$$\bar{A} = \bar{A} \cdot \bar{A} = \overline{A + A}$$

Entonces,

$$A \text{ --- } \neg \text{ --- } \bar{A} \quad \equiv \quad A \text{ --- } \text{NOR} \text{ --- } \bar{A}$$

Figura 3.25: Función NOT implementada con NOR

2. AND

$$A \cdot B = \overline{\overline{A} \cdot \overline{B}} = \overline{\overline{A} + \overline{B}}$$

Entonces,

$$\begin{matrix} A \\ B \end{matrix} \text{ --- } \text{AND} \text{ --- } A \cdot B \quad \equiv \quad \begin{matrix} A \\ B \end{matrix} \text{ --- } \text{NOR} \text{ --- } \text{NOR} \text{ --- } A \cdot B$$

Figura 3.26: Función AND implementada con NOR

3. OR

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

Entonces,

$$\begin{matrix} A \\ B \end{matrix} \text{ --- } \text{NOR} \text{ --- } A + B \quad \equiv \quad \begin{matrix} A \\ B \end{matrix} \text{ --- } \text{NOR} \text{ --- } \text{NOR} \text{ --- } A + B$$

Figura 3.27: Función OR implementada con NOR

3.3.6. Álgebra de la XOR y la XNOR

Aunque los operadores XOR y XNOR se pueden expresar usando las funciones básicas y aplicar los teoremas del álgebra de Boole para su análisis, existen expresiones válidas que permiten su manipulación conveniente y directa.

TEOREMAS DE LA XOR

Entre los principales teoremas están:

- $X \oplus 0 = X$, ya que $\overline{X}0 + X\overline{0} = X$
- $X \oplus 1 = \overline{X}$, ya que $\overline{X}1 + X\overline{1} = \overline{X}$
- $X \oplus X = 0$, ya que $\overline{X}X + X\overline{X} = 0$
- $X \oplus \overline{X} = 1$, ya que $\overline{X}\overline{X} + X\overline{\overline{X}} = \overline{X} + X = 1$
- Ley asociativa: $X \oplus Y \oplus Z = (X \oplus Y) \oplus Z = X \oplus (Y \oplus Z)$
- Ley conmutativa: $X \oplus Y \oplus Z = X \oplus (Y \oplus Z) = X \oplus (Z \oplus Y) = (X \oplus Z) \oplus Y = Z \oplus X \oplus Y$
- Ley distributiva: $X(Y \oplus Z) = (XY) \oplus (XZ)$
- Teorema de absorción: $X(\overline{X} \oplus Y) = XY$
- Teorema del consenso: $(XY) \oplus (\overline{X}Z) + (YZ) = (XY) \oplus (\overline{X}Z)$
- Teorema de DeMorgan: $X \oplus Y = \overline{X} \oplus \overline{Y} = \overline{X \odot Y}$

TEOREMAS DE LA XNOR

- $X \odot 1 = X$, ya que $\overline{\overline{X}1 + X\overline{1}} = X$
- $X \odot 0 = \overline{X}$, ya que $\overline{\overline{X}0 + X\overline{0}} = \overline{X}$
- $X \odot X = 1$, ya que $\overline{\overline{X}X + X\overline{X}} = \overline{0} = 1$
- $X \odot \overline{X} = 0$, ya que $\overline{\overline{X}\overline{X} + X\overline{\overline{X}}} = \overline{1} = 0$
- Ley asociativa: $X \odot Y \odot Z = (X \odot Y) \odot Z = X \odot (Y \odot Z)$
- Ley conmutativa: $X \odot Y \odot Z = X \odot (Y \odot Z) = X \odot (Z \odot Y) = (X \odot Z) \odot Y = Z \odot X \odot Y$
- Ley distributiva: $X + (Y \odot Z) = (X + Y) \odot (X + Z)$
- Teorema de absorción: $X + (\overline{X} \odot Y) = X + Y$
- Teorema del consenso: $(X + Y) \odot (\overline{X} + Z)(Y + Z) = (X + Y) \odot (\overline{X} + Z)$
- Teorema de DeMorgan: $X \odot Y = \overline{X} \odot \overline{Y} = \overline{X \oplus Y}$

PRINCIPIO DE LA DUALIDAD

Los teoremas vistos para la XOR y la XNOR también se pueden obtener con base en el principio de la dualidad, donde además de cambiar OR por AND, AND por OR, 1 por 0, 0 por 1 y dejar paréntesis, se debe cambiar la operación XOR por XNOR y la XNOR por XOR. Igualmente, el principio garantiza la validez de la expresión dual, más no su equivalencia con la original.

EJEMPLOS

1. Si $p : \{a + (b \odot c) = (a + b) \odot (a + c)\}$
Entonces $pd : \{a \cdot (b \oplus c) = (a \cdot b) \oplus (a \cdot c)\}$
2. Si $p : \{a \oplus 0 = a\}$
Entonces $pd : \{a \odot 1 = a\}$
3. Si $p : \{a \oplus a = 0\}$
Entonces $pd : \{a \odot a = 1\}$

3.4. FORMAS ALGEBRAICAS ESTÁNDARES

Hasta ahora se ha visto la representación simbólica, o representación implícita, de las funciones lógicas, donde el uso de las funciones básicas, las funciones derivadas y el álgebra booleana, dan paso a muchas formas de expresar una misma función. Incluso, la aplicación simple de las leyes conmutativa y asociativa permiten variar la escritura de una función. Con el fin dar paso a procesos de análisis y diseño de circuitos y expresiones lógicas, es necesario unificar la escritura mediante formas algebraicas estandarizadas, las cuales se describen a continuación.

3.4.1. Suma de productos (SOP)

Expresión donde cada término es un producto de variables independientes y los términos están agrupados mediante el operador de disyunción inclusiva. El operador de negación solo puede afectar una variable a la vez.

$$\begin{aligned} &AB + ABC \\ &\overline{A}\overline{B} + CDA + F\overline{H}G \\ &\overline{A}BC + \overline{A}\overline{B} + ABC \end{aligned}$$

3.4.2. Producto de sumas (POS)

Expresión donde cada término es una suma de variables independientes y los términos están agrupados mediante el operador de conjunción. El operador de negación solo puede afectar una variable a la vez.

$$\begin{aligned} &(\overline{A} + B)(A + \overline{B} + C) \\ &(A + B + D)(\overline{A} + \overline{B}) \\ &(A + \overline{B} + \overline{C})(A + C + H)(H + B) \end{aligned}$$

3.4.3. Forma SOP canónica

Forma SOP donde cada término producto contiene exactamente una vez todas las variables de la función.

$$\begin{aligned} & ABC\overline{C} + A\overline{B}C \\ & A\overline{B}F\overline{G}H + A\overline{B}F\overline{G}H + \overline{A}BF\overline{G}H \\ & \overline{A}BC + \overline{A}\overline{B}\overline{C} + ABC \end{aligned}$$

Cada término de una forma SOP estándar canónica se denomina **mintérmino**. Para toda función $f(x_1, x_2, x_3, \dots, x_n)$ existe una y sólo una forma SOP canónica.

Para llevar una función en forma SOP a una en forma SOP canónica, proceso denominado deducción de mintérminos, se puede aplicar de forma sistemática los postulados 2, 5 y 6, así como el teorema de idempotencia. A continuación se muestra este proceso para la expresión lógica $AB + A\overline{C} + B$.

$$\begin{aligned} & AB + A\overline{C} + B \\ & AB \cdot 1 + A\overline{C} \cdot 1 + B \cdot 1 \cdot 1 \\ & AB(C + \overline{C}) + A\overline{C}(B + \overline{B}) + B(A + \overline{A})(C + \overline{C}) \\ & ABC + AB\overline{C} + A\overline{C}B + A\overline{C}\overline{B} + ABC + AB\overline{C} + \overline{A}BC + \overline{A}B\overline{C} \\ & ABC + AB\overline{C} + \overline{A}B\overline{C} + \overline{A}BC + \overline{A}B\overline{C} \end{aligned}$$

La deducción de mintérminos también se puede obtener al aplicar el teorema de desarrollo de Shannon conjuntamente con el teorema de idempotencia. Se muestra este proceso para la misma expresión lógica anterior $AB + A\overline{C} + B$.

$$\begin{aligned} f(A, B, C) &= AB + A\overline{C} + B \\ &= A \cdot f(1, B, C) + \overline{A} \cdot f(0, B, C) \\ &= A[1 \cdot B + 1 \cdot \overline{C} + B] + \overline{A}[0 \cdot B + 0 \cdot \overline{C} + B] \\ &= AB + A\overline{C} + AB + \overline{A}B \\ &= AB + A\overline{C} + \overline{A}B \\ &= B \cdot f(A, 1, C) + \overline{B} \cdot f(A, 0, C) \\ &= B[A \cdot 1 + A\overline{C} + \overline{A} \cdot 1] + \overline{B}[A \cdot 0 + A\overline{C} + \overline{A} \cdot 0] \\ &= AB + AB\overline{C} + \overline{A}B + \overline{A}B\overline{C} \\ &= C \cdot f(A, B, 1) + \overline{C} \cdot f(A, B, 0) \\ &= C[AB + AB\overline{1} + \overline{A}B + \overline{A}B\overline{1}] + \overline{C}[AB + AB\overline{0} + \overline{A}B + \overline{A}B\overline{0}] \\ &= ABC + \overline{A}BC + AB\overline{C} + AB\overline{C} + \overline{A}B\overline{C} + \overline{A}B\overline{C} \\ &= ABC + \overline{A}BC + AB\overline{C} + \overline{A}B\overline{C} + \overline{A}B\overline{C} \end{aligned}$$

En toda forma SOP canónica, la expresión toma un valor lógico verdadero (1 lógico) cuando uno cualquiera de los mintérminos toma un valor de 1 lógico. Como un mintérmino es un producto de todas las variables de la función, para que tome un valor lógico de 1 es necesario que cada variable no negada del mintérmino valga 1 lógico y cada variable negada valga 0 lógico.

La función $Z = f(a, b, c) = ab\bar{c} + \bar{a}\bar{b}c + \bar{a}bc$ vale 1 cuando $ab\bar{c}$, o cuando $\bar{a}\bar{b}c$, o cuando $\bar{a}bc$ toman un valor de 1. Para que $ab\bar{c} = 1$ se necesita que $a = 1$, $b = 1$ y $c = 0$; para que $\bar{a}\bar{b}c = 1$ se necesita que $a = 0$, $b = 0$ y $c = 1$; para que $\bar{a}bc = 1$ se necesita que $a = 0$, $b = 1$ y $c = 1$. Con base en esta interpretación, y teniendo presente el orden de las variables, donde a es la más significativa y c la menos significativa, se puede representar cada mintérmino numerado en binario con su respectivo decimal, así:

$$\begin{array}{ccc}
 \begin{array}{c} MSD \\ \downarrow \\ Z = f(a, b, c) \\ \uparrow \\ LSD \end{array} & = & ab\bar{c} + \bar{a}\bar{b}c + \bar{a}bc \\
 & & \begin{array}{ccc} \begin{array}{c} \downarrow\downarrow\downarrow \\ 110 \\ \hline 6 \end{array} & \begin{array}{c} \downarrow\downarrow\downarrow \\ 000 \\ \hline 0 \end{array} & \begin{array}{c} \downarrow\downarrow\downarrow \\ 011 \\ \hline 3 \end{array} \end{array}
 \end{array}$$

De lo anterior, dos formas compactas de representar una función son las siguientes:

$$Z = f(a, b, c) = ab\bar{c} + \bar{a}\bar{b}c + \bar{a}bc = m_0 + m_3 + m_6 = \sum m(0, 3, 6)$$

Si se realiza la representación explícita de una función en forma SOP canónica, los unos de la tabla de verdad coinciden con la presentación binaria de los mintérminos y por tanto con la numeración decimal de las combinaciones:

#	a	b	c	Z
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

3.4.4. Forma POS canónica

Forma POS donde cada término suma contiene exactamente una vez todas las variables de la función.

$$\begin{aligned} & (\overline{A} + B + \overline{C}) (A + \overline{B} + C) \\ & (A + B + C + \overline{D}) (\overline{A} + \overline{B} + C + D) \\ & (A + \overline{B} + \overline{C} + H) (A + B + \overline{C} + \overline{H}) (\overline{A} + \overline{C} + H + B) \end{aligned}$$

Cada término de una forma POS estándar canónica se denomina **maxtérmino**. Para toda función $f(x_1, x_2, x_3, \dots, x_n)$ existe una y solo una forma POS canónica.

Para llevar una función en forma POS a una en forma POS canónica, proceso denominado deducción de maxtérminos, se aplica de forma sistemática los apartados duales a los aplicados en la deducción de mintérminos. A continuación, se muestra este proceso para la expresión lógica $A(\overline{B} + C)$ haciendo uso de los postulados 2, 5 y 6, así como el teorema de idempotencia.

$$\begin{aligned} & A(\overline{B} + C) \\ & (A + 0)(\overline{B} + C + 0) \\ & (A + B\overline{B})(\overline{B} + C + A\overline{A}) \\ & (A + B)(A + \overline{B})(A + \overline{B} + C)(\overline{A} + \overline{B} + C) \\ & (A + B + C\overline{C})(A + \overline{B} + C\overline{C})(A + \overline{B} + C)(\overline{A} + \overline{B} + C) \\ & (A + B + C)(A + B + \overline{C})(A + \overline{B} + C)(A + \overline{B} + \overline{C})(A + \overline{B} + C)(\overline{A} + \overline{B} + C) \\ & (A + B + C)(A + B + \overline{C})(A + \overline{B} + C)(A + \overline{B} + \overline{C})(\overline{A} + \overline{B} + C) \end{aligned}$$

La deducción de maxtérminos también se puede obtener al aplicar el teorema de desarrollo de Shannon, conjuntamente con el teorema de idempotencia, en sus versiones duales a las aplicadas para la deducción de mintérminos. Se muestra este proceso para la misma expresión lógica planteada $A(\overline{B} + C)$.

$$\begin{aligned} f(A, B, C) &= A(\overline{B} + C) \\ &= [A + f(0, B, C)] \cdot [\overline{A} + f(1, B, C)] \\ &= (A)(\overline{A} + \overline{B} + C) \\ &= [B + f(A, 0, C)] \cdot [\overline{B} + f(A, 1, C)] \\ &= [B + (A)(\overline{A} + 1 + C)] \cdot [\overline{B} + (A)(\overline{A} + C)] \\ &= (A + B) \cdot (A + \overline{B})(\overline{A} + \overline{B} + C) \\ &= [C + f(A, B, 0)] \cdot [\overline{C} + f(A, B, 1)] \\ &= [C + (A + B) \cdot (A + \overline{B})(\overline{A} + \overline{B})] \cdot [\overline{C} + (A + B) \cdot (A + \overline{B})] \\ &= (A + B + C)(A + \overline{B} + C)(\overline{A} + \overline{B} + C)(A + B + \overline{C})(A + \overline{B} + \overline{C}) \\ &= (A + B + C)(A + \overline{B} + C)(\overline{A} + \overline{B} + C)(A + B + \overline{C})(A + \overline{B} + \overline{C}) \end{aligned}$$

En toda forma POS canónica, la expresión toma un valor lógico falso (0 lógico) cuando uno cualquiera de los maxtérminos toma un valor de 0 lógico. Como un maxtérmino es una suma de todas las variables de la función, para que tome un valor lógico de 0 es necesario que cada variable no negada del mintérmino valga 0 lógico y cada variable negada valga 1 lógico.

La función $Z = f(a, b, c) = (\bar{a} + b + c) (\bar{a} + \bar{b} + \bar{c}) (\bar{a} + b + \bar{c})$ vale 0 cuando $(\bar{a} + b + c) = 0$, o cuando $(\bar{a} + \bar{b} + \bar{c}) = 0$, o cuando $(\bar{a} + b + \bar{c}) = 0$. Para que: $(\bar{a} + b + c) = 0$ se necesita que $a = 1$, $b = 0$ y $c = 0$; para que $(\bar{a} + \bar{b} + \bar{c}) = 0$ se necesita que $a = 1$, $b = 1$ y $c = 1$; para que $(\bar{a} + b + \bar{c}) = 0$ se necesita que $a = 1$, $b = 0$ y $c = 1$. Con base en esta interpretación, y teniendo presente el orden de las variables, donde a es la más significativa y c la menos significativa, se puede representar cada maxtérmino numerado en binario con su respectivo decimal, así:

$$Z = f(a, b, c) = (\bar{a} + b + c) (\bar{a} + \bar{b} + \bar{c}) (\bar{a} + b + \bar{c})$$

$$\underbrace{\begin{matrix} \downarrow & \downarrow & \downarrow \\ 1 & 0 & 0 \end{matrix}}_4 \underbrace{\begin{matrix} \downarrow & \downarrow & \downarrow \\ 1 & 1 & 1 \end{matrix}}_7 \underbrace{\begin{matrix} \downarrow & \downarrow & \downarrow \\ 1 & 0 & 1 \end{matrix}}_5$$

De lo anterior, dos formas compactas de representar una función son las siguientes:

$$Z = f(a, b, c) = (\bar{a} + b + c) (\bar{a} + \bar{b} + \bar{c}) (\bar{a} + b + \bar{c}) = M_4 \cdot M_5 \cdot M_7 = \prod M(4, 5, 7)$$

Si se realiza la representación explícita de una función en forma POS canónica, los ceros de la tabla de verdad coinciden con la presentación binaria de los maxtérminos y por tanto con la numeración decimal de las combinaciones:

#	a	b	c	Z
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

3.4.5. Deducción de canónicas por numeración

La numeración de mintérminos o maxtérminos también se puede emplear como metodología rápida para llevar una forma SOP o POS a una forma estándar canónica. El proceso consiste en, para cada mintérmino de la expresión, representar el valor binario de cada variable y para las variables faltantes tomar las dos opciones posibles (1 o 0). Los números binarios resultantes representan a los mintérminos de la función. Se debe tener presente conservar de forma estricta el orden asignado a las variables y aplicar el Teorema de Idempotencia, el cual elimina los términos repetidos.

EJEMPLOS

1. Encontrar la forma SOP estándar canónica para $Z = f(A, B, C) = AB + A\overline{C} + B$:

$$\begin{aligned} Z &= AB + A\overline{C} + B \\ &= 11_0^1 + 1_0^1 0 + _0^1 1_0^1 \\ &= m_7 + m_6 + m_6 + m_4 + m_7 + m_6 + m_3 + m_2 \\ &= \sum m(2, 3, 4, 6, 7) \end{aligned}$$

2. Encontrar la forma POS estándar canónica para $Z = f(A, B, C) = A(\overline{B} + C)$:

$$\begin{aligned} Z &= A(\overline{B} + C) \\ &= (A + _1^0 + _1^0) (_1^0 + \overline{B} + C) \\ &= M_0 \cdot M_1 \cdot M_2 \cdot M_3 \cdot M_2 \cdot M_6 \cdot \\ &= \prod M(0, 1, 2, 3, 6) \end{aligned}$$

3.4.6. Equivalencias entre formas canónicas

Toda forma SOP canónica tiene una forma POS canónica equivalente. En una tabla de verdad, los 1 lógicos de una función indican los mintérminos para la forma SOP canónica y los 0 lógicos indican los maxtérminos para la forma POS canónica. Además, los 1 lógicos de una función f son los ceros lógicos de la misma función negada $\overline{f}$, y los ceros lógicos de f son los unos lógicos de $\overline{f}$. Lo anterior permite pasar entre formas SOP y POS canónicas y entre una función y su negación.

EJEMPLOS

1. De la sección 3.4.3, la función $Z = f(a, b, c) = ab\overline{c} + \overline{a}\overline{b}c + \overline{a}bc = \sum m(0, 3, 6)$ se puede expresar en forma POS canónica así: $Z = f(a, b, c) = \prod M(1, 2, 4, 5, 7) = M_1 \cdot M_2 \cdot M_4 \cdot M_5 \cdot M_7$, ahora como en cada maxtérmino numerado en binario, un 1 representa una variable negada y un cero una variable sin negar:

$$Z = f(a, b, c) = (a + b + \overline{c})(a + \overline{b} + c)(\overline{a} + b + c)(\overline{a} + b + \overline{c})(\overline{a} + \overline{b} + \overline{c})$$

2. De la sección 3.4.4, la función $Z = f(a, b, c) = (\bar{a} + b + c)(\bar{a} + \bar{b} + \bar{c})(\bar{a} + b + \bar{c}) = \prod M(4, 5, 7)$ se puede expresar en forma SOP canónica así: $Z = f(a, b, c) = \sum m(0, 1, 2, 3, 6) = m_0 + m_1 + m_2 + m_3 + m_6$, ahora como en cada mintérmino numerado en binario, un 1 representa una variable sin negar y un cero una variable negada:

$$Z = f(a, b, c) = \bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}c + \bar{a}b\bar{c} + \bar{a}bc + ab\bar{c}$$

3. Para la función $Z = f(A, B, C) = \sum m(2, 3, 4, 6, 7)$ se cumple que:

$$f(A, B, C) = \sum m(2, 3, 4, 6, 7) = \prod M(0, 1, 5)$$

$$\text{Como } f(A, B, C) = \sum m(2, 3, 4, 6, 7) \text{ entonces } \bar{f}(A, B, C) = \sum m(0, 1, 5)$$

$$\text{Como } f(A, B, C) = \prod M(0, 1, 5) \text{ entonces } \bar{f}(A, B, C) = \prod M(2, 3, 4, 6, 7)$$

4. Si una función $Z = f(A, B, C) = (A + C)(B + \bar{C})$ encontrar su representación como suma de sus mintérminos.

Como de forma inicial la representación es POS, se inicia encontrando los maxtérminos de la función:

$$\begin{aligned} Z = f(A, B, C) &= (A + C)(B + \bar{C}) \\ &= (0 + \overset{0}{1} + 0)(\overset{0}{1} + 0 + 1) \\ &= \prod M(0, 1, 2, 5) \end{aligned}$$

Finalmente, como $Z = f(A, B, C) = (A + C)(B + \bar{C}) = \prod M(0, 1, 2, 5)$ entonces $Z = f(A, B, C) = \sum m(3, 4, 6, 7)$, por lo que $Z = f(A, B, C) = \bar{A}BC + A\bar{B}\bar{C} + AB\bar{C} + ABC$.

3.5. CIRCUITOS LÓGICOS ESTÁNDARES

La implementación física del circuito lógico de una función se puede realizar a partir de la expresión lógica SOP canónica, de la POS canónica o de una SOP o POS estándar mínima. Las formas mínimas son formas SOP o POS simplificadas, pero que lógicamente conservan la forma estándar respectiva.

EJEMPLO

Sea la función $Z = f(A, B, C) = B + AC$

La anterior es una representación SOP estándar mínima para Z . Sin embargo, Z puede tener muchas otras formas de representación. Por ejemplo, la representación explícita de Z esta dada por su tabla de verdad, así:

#	A	B	C	Z
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

La representación algebraica de Z puede estar dada por la SOP canónica o la POS canónica:

$$Z = \sum m(2, 3, 5, 6, 7) = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

$$Z = \prod M(0, 1, 4) = (A + B + C)(A + B + \bar{C})(\bar{A} + B + C)$$

La figura 3.28 muestra la implementación de Z con en sus versiones SOP canónica, POS canónica y SOP estándar mínima.

Figura 3.28: Circuitos lógicos estándares

Es evidente concluir entonces que, para la implementación de una función lógica, es preferible utilizar una expresión algebraica mínima ya que utiliza un menor número de términos, permitiendo una estructura de circuito más simple. Además, es preferible también que la expresión algebraica mínima a utilizar esté en alguna representación estandarizada, para permitir que la estructura del circuito lógico, además de ser simple, sea también estándar. Por otro lado, una estructura estandarizada permite realizar fácilmente un procedimiento conocido como **universalización**, el cual se expone a continuación.

3.5.1. Universalización

El proceso de universalización consiste en expresar una función lógica en términos de un sólo tipo de operador. Para ello, todas las funciones lógicas básicas se deben poder expresar en términos del operador seleccionado. Sólo las funciones NAND y NOR permiten desarrollar las funciones básicas en términos de ellas, tal como se presentó en las secciones 3.3.4 y 3.3.5.

UNIVERSALIZACIÓN CON NAND

Cuando se utiliza una expresión algebraica SOP para implementar una función, el circuito lógico resultante tiene una estructura AND-OR, o primero va una serie de compuertas AND que finalmente se agrupan bajo el operador OR. Esta estructura es equivalente a la estructura NAND-NAND luego de aplicar el teorema de DeMorgan. Por ejemplo:

$$\begin{aligned} Z = \sum m(2, 3, 5, 6, 7) &= \overline{\overline{ABC} + \overline{ABC} + \overline{ABC} + \overline{ABC} + \overline{ABC}} \\ &= \overline{\overline{ABC} \cdot \overline{ABC} \cdot \overline{ABC} \cdot \overline{ABC} \cdot \overline{ABC}} \end{aligned}$$

UNIVERSALIZACIÓN CON NOR

Cuando se utiliza una expresión algebraica POS para implementar una función, el circuito lógico resultante tiene una estructura OR-AND, o primero va una serie de compuertas OR que finalmente se agrupan bajo el operador AND. Esta estructura es equivalente a la estructura NOR-NOR luego de aplicar el teorema de DeMorgan. Por ejemplo:

$$\begin{aligned} Z = \prod M(0, 1, 4) &= \overline{\overline{(A + B + C)} \cdot \overline{(A + B + \overline{C})} \cdot \overline{(\overline{A} + B + C)}} \\ &= \overline{\overline{(A + B + C)} + \overline{(A + B + \overline{C})} + \overline{(\overline{A} + B + C)}} \end{aligned}$$

Adicionalmente, en ocasiones es conveniente universalizar utilizando funciones que tengan un número de entradas limitado, es decir, que cuando se utilizan las funciones NAND o NOR, estas tengan un número finito e igual de entradas. Por ejemplo, se puede universalizar una expresión algebraica utilizando sólo funciones NAND de dos entradas. Es importante recordar que las funciones NAND y NOR cumplen la propiedad conmutativa, más no la asociativa.

EJEMPLO

La figura 3.29 muestra los circuitos universalizados para las expresiones SOP canónica y POS canónica de la figura 3.28.

Figura 3.29: Circuito lógico universalizado

3.5.2. Funciones con especificación incompleta

Un término **prescindible** es aquel que, para una determinada expresión, puede ser considerado como mintérmino o maxtérmino sin que esto afecte el comportamiento de la función lógica que la expresión representa. Esto ocurre, por ejemplo, en casos mutuamente excluyentes donde una combinación de entrada NO SE PRESENTA, o cuando sólo es importante la respuesta para ciertas combinaciones de entrada. Por ejemplo, si en un tanque de reserva de agua existen dos sensores de nivel, uno para nivel bajo y otro para nivel alto, y si estos sensores entregan un uno lógico cuando el nivel es igual o supera la posición del sensor, entonces se tienen las siguientes posibilidades:

Sensor nivel bajo	Sensor nivel alto	Interpretación
0	0	Tanque vacío
0	1	Sin interpretación
1	0	Tanque medio
1	1	Tanque lleno

De la anterior tabla, la combinación 01 (sensor nivel bajo no activo y sensor nivel alto activo) no se presenta, ya que es una contradicción, por tanto esta combinación es prescindible. Los términos prescindibles también se conocen como términos **No importa** o *Don't care*.

EJEMPLO

Una función Z puede ser representada por:

$$Z = f(a, b, c) = \sum m(0, 1, 2, 3) + \sum d(6, 7)$$

$$Z = f(a, b, c) = \prod M(4, 5) \cdot \prod D(6, 7)$$

Esto quiere decir que 6 y 7 pueden ser considerados como mintérminos o maxtérminos según convenga.

Normalmente, los términos prescindibles son utilizados en los procedimientos de minimización de las funciones lógicas, donde se pueden tomar individualmente como mintérminos o maxtérminos, tomarlos todos, tomar algunos o no tomar ninguno.

3.6. EJERCICIOS PROPUESTOS

- Para una función cualquiera $Z = f(A, B)$:
 - Obtenga el número total de combinaciones de entrada y de posibles funciones existentes.
 - Implemente una tabla de verdad para todas las posibles funciones.
 - Dentro del conjunto de posibles funciones, identifique las funciones lógicas básicas.
 - Dentro del conjunto de posibles funciones, identifique las funciones lógicas derivadas.
- Para cada una de las siguientes funciones obtenga el diagrama de tiempo, el diagrama lógico y la tabla de verdad:
 - $f(a, b, c) = a\bar{c} + \bar{b}c + ab$
 - $f(a, b, c) = (\bar{a} + \bar{b} + c)(\bar{a} + b)(a + b + \bar{c})$
 - $f(A, B, C) = A\bar{B} + C(\bar{A} + B + \bar{C})$
 - $f(A, B, C, D) = \overline{(A \odot B) + (C \oplus D)} + AC(A + B + D)$
 - $f(w, x, y, z) = \{w + w(\bar{x} + y)\} \{\bar{x} + y + z\} + w\bar{x} + \bar{w}yz$
- Usando exclusivamente diagramas de Venn:
 - Demuestre que $(A \odot B) + (A \oplus B) = 1$
 - Comprobar si $\bar{A}B(C\bar{A} + B\bar{C}) \stackrel{?}{=} \bar{A}B$
 - Comprobar si $\bar{A}B + (\bar{C}\bar{A} + B\bar{C}) \stackrel{?}{=} \bar{A} + B$
 - Simplificar $ABC\{AB + \bar{C}(BC + AC)\}$
 - Simplificar $(w + wx)(\bar{w} + y)$
- Simplifique las siguientes expresiones empleando sólo postulados o teoremas del álgebra de Boole:
 - $f(A, B, C, D) = (\bar{A} + B + D)(C + \bar{D})(\bar{B} + \bar{C} + D)(A + \bar{D})$
 - $f(a, b, c, d) = \overline{(a \oplus b)(b \oplus c)} + \overline{\bar{a} + \bar{b} + a + c}$
 - $f(a, b, c, d) = abc\{ab + \bar{c}(bc + ac)\}$
 - $f(a, b, c, d, e) = \bar{a}b + \bar{a}b\bar{c} + \bar{a}bcd + \bar{a}b\bar{c}d\bar{e}$
 - $f(a, b, c, d) = (a\bar{b} + \bar{a}c(\bar{b} + \bar{c}))(d + a\bar{c})\overline{(c + d)(\bar{a}b\bar{c} + a\bar{d})}$

5. Para la función $f(a, b, c, d) = \sum m(1, 3, 5, 7)$:
 - a) Encontrar la función en forma SOP canónica
 - b) Encontrar la función en forma POS canónica
 - c) Encontrar la función en forma SOP mínima
 - d) Encontrar la función en forma POS mínima
 - e) Encontrar la función $f(d, c, b, a)$ en forma de suma de mintérminos
 - f) Encontrar la función $f(c, d, a, b)$ en forma producto de maxtérminos
 - g) Encontrar $\bar{f}(a, b, c, d)$ en forma SOP canónica
6. Realizar los siguientes circuitos lógicos:
 - a) Circuito de una XOR de 2 entradas mediante una XOR de cuatro entradas.
 - b) Circuito de una XOR de cuatro entradas mediante compuertas XOR de dos entradas
7. Llevar cada una de las siguientes expresiones lógicas a su forma canónica, implementar el circuito lógico estándar asociado y el circuito lógico universalizado con compuertas de dos entradas:
 - a) $f(x, y, z) = \bar{y}z + \bar{x}y\bar{z} + xyz + \bar{x}\bar{y}z$
 - b) $f(A, B, C, D) = (A + \bar{B} + C)(C + \bar{D})(\bar{A} + C + \bar{D})(B + \bar{D})$
 - c) $f(a, b, c, d) = \bar{a}bcd + \bar{a}\bar{c}d + b\bar{c}d + \bar{a}b\bar{c}\bar{d}$
 - d) $f(A, B, C, D) = \overline{(A \odot B) + (C \oplus D)}$
8. Si la entrada a una función son los cuatro dígitos binarios de la representación BCD natural, y la salida es un uno lógico cuando la entrada está en el rango 5 a 9, determine:
 - a) La forma SOP canónica sin tomar y tomando los términos prescindibles.
 - b) La forma POS canónica sin tomar y tomando los términos prescindibles.
 - c) La forma SOP mínima sin tomar y tomando los términos prescindibles.
 - d) La forma POS mínima sin tomar y tomando los términos prescindibles.
 - e) La forma mínima de todas las anteriores.

4

CAPÍTULO CUATRO

Minimización y síntesis de funciones lógicas

Aunque el álgebra booleana permite realizar la minimización de ecuaciones lógicas para llevarlas a expresiones convenientes, esta minimización es un procedimiento manual y depende de la habilidad de quien la realiza para encontrar formas adecuadas y reorganizar términos. Una manera de realizar estos requerimientos de minimización es emplear un procedimiento gráfico denominado mapas de Veitch-Karnaugh que es práctico en sistemas de hasta máximo 5 variables, y aún con cierto nivel de incertidumbre en su solución. Para el tratamiento de sistemas con cualquier número de variables, e incluso para minimizar varias funciones de forma simultánea, este capítulo muestra el algoritmo de minimización tabular de Quine-McCluskey-Petrack, que es un método completamente determinístico. Estos procedimientos de minimización gráficos y tabulares permiten tratar con funciones en las cuales es posible que ciertos valores de entrada no se presentan, con funciones de más de 6 variables, con múltiples funciones simultáneas y hasta con funciones donde se mezclan estas demandas.

4.1. DEFINICIONES

Síntesis:	<i>Procedimiento de análisis de un circuito complejo mediante la separación e identificación de sus componentes básicos.</i>
Minimización:	<i>Procedimiento mediante el cual se encuentra la representación de una función que implique el menor costo de la función.</i>
Costo de función:	<i>Se refiere al número de términos y variables lógicas que componen una función, además del número total de elementos necesarios para su implementación.</i>

La minimización de una función se refiere a encontrar una representación, normalmente estandarizada, que implique el menor costo en referencia a obtener una función que cumpla los siguientes criterios:

- Menor número de términos. Si la forma obtenida posee el menor número de términos posible, entonces se necesitan menos elementos para su implementación.
- Menor número de variables. Si luego del criterio anterior, se elige la forma que elimine más variables, entonces cada elemento usado es lo más simple posible.
- Menor número posible de elementos diferentes. Si se realiza la universalización de una función lógica, se obtiene una forma donde sólo se emplea un único tipo de elemento. Además es importante unificar el número de entradas en cada elemento.
- Menor número de circuitos. Comercialmente, los elementos lógicos se obtienen en grupos, o paquetes de ellos, bajo lo se que denomina **circuitos integrados**. Ya que estos circuitos traen cantidades definidas de elementos, es importante planificar su uso eficiente. Normalmente la universalización y reutilización de señales permiten este objetivo.
- Menor espacio físico requerido en la implementación física.
- Menor número de capas de circuito. Cada señal sufre de retardos en su propagación a medida que pasa por un elemento lógico, por lo que es necesario disminuir el número de retardos.
- Siempre se debe garantizar el menor consumo de un circuito.

En general, estos criterios aplican tanto para implementaciones físicas como para implementaciones programadas. Para el caso de una implementación programada, los mismos criterios llevan a ecuaciones lógicas con menor complejidad, en relación a ser evaluadas con menos operaciones de computadora y con menos uso de memoria.

4.2. MAPAS DE VEITCH-KARNAUGH

Método gráfico diseñado originalmente por Edward Veitch en 1952 y mejorado sustancialmente por Maurice Karnaugh en 1953. Es una representación en forma de tabla de los diagramas de Venn, donde se concatenan conceptos sobre mintérminos, maxtérminos, términos prescindibles, numeración decimal en binario, código gray y álgebra booleana. Normalmente se simplifica su notación como mapa VK. Ya que los diagramas de Venn son gráficos de contorno cerrado, los mapas de Veitch-Karnaugh también lo son.

4.2.1. Mapa VK para función de dos variables

Un diagrama de Venn de dos variables consiste de dos posibles conjuntos (variables) que tienen intersección. La figura 4.1 indica para cada área posible la pertenencia, o no, a cada uno de los conjuntos; así, por ejemplo, la intersección es un área donde están presentes ambos conjuntos, por lo que se indica como AB , mientras que al área externa a ambos conjuntos se indica por $\overline{A}\overline{B}$. Si para cada uno de los términos que identifican un área en particular, la pertenencia se indica con un uno lógico y su no pertenencia con un cero lógico, entonces, la intersección es $AB = 11$ y corresponde con el mintermino m_3 , igualmente $\overline{A}\overline{B} = 00$ y corresponde con el mintermino m_0 . Este procedimiento consiste básicamente en identificar la numeración del mintermino asociado con el área respectiva en el diagrama de Venn. En todo diagrama de Venn, la línea del lado izquierdo del diagrama realmente es la continuación de la línea del lado derecho, por lo que el diagrama se cierra sobre sí mismo para que estos dos lados se unan. Igualmente sucede con los lados superior e inferior, dando origen a lo que se denomina como un gráfico de contorno cerrado.

Figura 4.1: Diagrama de Venn para función de 2 variables

En la figura 4.1 se puede observar que, entre áreas vecinas sólo cambia un bit de la numeración binaria, además cada área es vecina con otras dos. Estas características se aprovechan para realizar representaciones rectangulares del diagrama de Venn de dos variables, o mapa VK, tal como se muestra en la figura 4.2. En los mapas VK de dos variables, cada área es vecina a dos áreas y entre ellas sólo cambia un bit. Ya que se trata con gráficos de contorno cerrado, en el mapa MK del lado derecho, el área del mintermino 0 es vecina a las áreas de los minterminos 1 y 2, y a su vez el área del mintermino 2 es vecina a las áreas de los minterminos 0 y 3, conservando así dos áreas vecinas.

Figura 4.2: Mapas VK para 2 variables

4.2.2. Mapa VK para función de tres variables

La figura 4.3 muestra un diagrama de Venn para tres variables. Igualmente se sigue cumpliendo que entre áreas vecinas sólo cambia un bit de la numeración binaria, pero ahora cada área es vecina con otras tres. Con estas características, se construye la representación rectangular asociada.

Figura 4.3: Diagrama de Venn para función de 3 variables

En la figura 4.4 se puede observar que cada celda es vecina a otras tres, por ejemplo, la celda 0 es adyacente a las celdas 1, 2 y 4, tal como lo son los minterminos respectivos en el diagrama de Venn. Se debe notar que la numeración decimal de cada celda se obtiene leyendo la numeración binaria para filas y columnas, la cual sigue el código gray, y conservando la significancia de las variables. Así, por ejemplo, para un orden de significancia donde la variable A es la más significativa y la variable C la menos significativa, la celda 5 es producto de leer la numeración para una función $f(A, B, C)$ donde $A = 1$, $B = 0$ y $C = 1$, lo cual entrega el binario 101, que es el decimal 5.

Figura 4.4: Mapas VK para 3 variables

4.2.3. Mapa VK para función de cuatro variables

La figura 4.5 muestra un diagrama de Venn para cuatro variables. Se sigue cumpliendo que entre áreas vecinas sólo cambia un bit de la numeración binaria, pero ahora cada área es vecina con otras cuatro. Con estas características se construye la representación rectangular asociada, como se muestra en la figura 4.6. Un diagrama de Venn para cuatro variables se obtiene dividiendo cada área del diagrama de tres variables en dos áreas, este mismo proceso es extensible a más variables. Los diagramas de Venn usualmente solo se emplean para representar tres, o máximo cuatro, variables.

Figura 4.5: Diagrama de Venn para función de 4 variables

Figura 4.6: Mapas VK para 4 variables

En la figura 4.6 se puede observar que cada celda es vecina a otras cuatro, por ejemplo, la celda 7 es vecina a las celdas 3, 5, 6 y 15 tal como lo son los minterminos respectivos en el diagrama de Venn. Igualmente, la celda 10 es vecina a las celdas 2, 8, 11 y 14. El número de filas y de columnas de un mapa VK es siempre una potencia de 2.

4.2.4. Mapa VK para funciones de cinco, o más, variables

Los mapas VK para más de cinco variables implican tener siempre presente que el número de vecindades para una celda debe ser igual al número de variables de la función. La figura 4.7 muestra ejemplos de mapas VK para 5 y 6 variables. En el caso particular del mapa VK de 5 variables, una celda debe poseer cinco celdas vecinas, por lo que el mapa se construye como la superposición por capas de dos mapas VK de 4 variables y, así, una celda posee 4 vecindades en una capa y otra vecindad que se encuentra en la misma posición del mapa de la otra capa. Para el caso del mapa VK de 6 variables, los mapas se superponen igualmente en capas, y una celda posee cuatro vecindades en el mismo mapa y otras dos que se encuentran en la misma posición de los mapas en capas continuas. En los mapas VK de 6 variables, los mapas en los extremos se consideran en contorno cerrado.

Los mapas VK para siete, o más, variables se consideran imprácticos, debido a la complejidad de graficarlos y, por consiguiente, de determinar las vecindades de una celda.

Figura 4.7: Mapas VK para 5 y 6 variables

4.2.5. Reglas de construcción para mapas VK

A partir de los mapas VK descritos previamente, se tiene el siguiente conjunto de reglas generales de construcción:

- Un mapa VK debe tener tantas celdas como áreas diferentes tiene el análogo diagrama de Venn, por tanto, para n variables el mapa VK tiene 2^n celdas.
- En un mapa VK cada celda debe tener n celdas vecinas. Para cumplimiento de lo anterior, los contornos del mapa se consideran cerrados.
- El mapa VK es una tabla donde el número de filas y de columnas es potencia de dos, pero la representación tiene tendencia cuadrada.
- En un mapa VK las filas y columnas siguen una numeración binaria en código gray.
- Las variables se ubican para que su lectura sea en orden, desde la más significativa hacia la menos significativa.

- La numeración decimal de celdas se obtiene de leer en orden de precedencia las variables según la numeración en código gray asignada.
- Cada celda representa un posible mintérmino o maxtérmino de la función.

Volviendo a la figura 4.6, ya que existen 4 variables, el mapa VK debe ser de $2^4 = 16$ celdas, donde cada celda debe tener 4 vecinos. El número de filas y columnas debe ser potencia de dos, por lo que se toma una representación de 4 filas y 4 columnas. Se ubican las variables para lectura en orden desde la más significativa hacia la menos significativa; y tanto filas como columnas se numeran en binario siguiendo código gray, por lo que la numeración sigue el orden 00, 01, 11, 10. Finalmente, se procede a realizar la numeración decimal de las celdas. En esta figura, se puede observar que hay dos formas de ubicar las variables, y que cada una lleva a una numeración diferente del mapa VK.

4.3. MINIMIZACIÓN POR MAPAS VK

Como en los mapas VK las celdas representan a cada uno de los posibles mintérminos de una función, las expresiones lógicas deben estar en una forma canónica para poder determinar los unos, o los ceros, de la función respectiva. Por ejemplo, si se tiene la función $ab + a\bar{b}c$ esta se debe pasar a una expresión SOP canónica, con lo cual toma la forma $ab + a\bar{b}c = abc + ab\bar{c} + a\bar{b}c$ o en forma compacta $abc + ab\bar{c} + a\bar{b}c = \sum m(5, 6, 7)$. Igualmente, la función se puede representar en forma POS canónica, o $ab + a\bar{b}c = (a + b + c)(a + b + \bar{c})(a + \bar{b} + c)(a + \bar{b} + \bar{c})(\bar{a} + b + c) = \prod M(0, 1, 2, 3, 4)$. El diagrama de Venn y mapa VK, asociados a la forma SOP canónica, se muestran en la figura 4.8.

Figura 4.8: Ejemplo básico de mapa VK

De la figura 4.8, se puede observar en el diagrama de Venn y en el mapa VK que las áreas de los mintérminos 5 y 7 son vecinas e igual las áreas de los mintérminos 6 y 7. Dichas áreas vecinas se pueden agrupar y formar un área mayor con lo cual, por ejemplo, las áreas 5 y 7 son la intersección ac y las áreas 6 y 7 la intersección ab , llevando a que la expresión original $ab + a\bar{b}c$ se pueda escribir también como $ab + ac$. En el mapa VK estas vecindades también se pueden representar como se muestra en la figura 4.9. Para saber en un mapa VK como representar estas nuevas agrupaciones, se procede así: en el grupo de los mintérminos 5 y 7 la variable a siempre vale 1 lógico, la variable b cambia de valor y la variable c siempre vale 1 lógico; por tanto, como la variable b cambia de valor, se elimina al agrupar estos dos mintérminos, y el nuevo grupo se denomina ac . Igualmente, en la agrupación de los mintérminos 6 y 7, a siempre vale 1 lógico, b siempre vale 1 lógico y c cambia de valor; por tanto, como la variable c cambia de valor, se elimina y el nuevo grupo se denomina ab . Como se trabaja con mintérminos, el área total es la suma de las áreas parciales y la función

$ab + a\bar{b}c = ab + ac$, la cual es una expresión SOP mínima. En definitiva, en cada agrupación que se realiza, se eliminan todas las variables que cambian de valor.

		AB			
		00	01	11	10
C	0				
	1		1	1	1

Figura 4.9: Agrupación en mapa VK

4.3.1. Criterios de agrupación en mapas VK

Al minimizar una expresión en un diagrama de Venn, siempre es de menor costo una expresión que se componga de menos términos (ya sean producto o suma) y donde cada término tenga la menor cantidad posible de variables. Una menor cantidad de términos se logra agrupando todas las áreas con la menor cantidad posible de nuevos grupos y una menor cantidad posible de variables por grupo se logra haciendo cada grupo lo más grande posible. Al cumplir con los anteriores criterios, la expresión resultante es mínima para la función en análisis. Para desarrollar este procedimiento con éxito sobre un mapa VK, se debe tener presente los siguientes criterios:

- Dos celdas en un mapa VK son vecinas si y sólo si en la numeración binaria de cada una, sólo cambia un bit entre ellas para el mintérmino, o maxtérmino, que representan.
- Con celdas vecinas se pueden formar nuevos grupos con tantas celdas como potencia de dos, es decir, se pueden agrupar 1, 2, 4, 8, 16, etc. celdas entre sí.
- Cada agrupación sigue los mismos criterios de formación que para un mapa VK, siendo representaciones cuadradas con números de filas y columnas potencia de dos, y recordando que son de contorno cerrado.
- El número de agrupaciones debe ser el menor posible, lo cual garantiza que la expresión simplificada tiene la menor cantidad posible de términos.
- Cada agrupación debe ser lo más grande posible, lo cual garantiza que en cada término se simplifica la mayor cantidad posible de variables.
- En cada grupo, si una variable cambia de valor, entonces se simplifica. Si en la agrupación la variable vale siempre uno lógico, o cero lógico, no se simplifica. Si la forma de la función es SOP, cada término es producto, donde las variables que valen 1 lógico se escriben sin negar y las variables que valen 0 lógico se escriben negadas. Si la forma de la función es POS, cada término es suma, donde las variables que valen 1 lógico se escriben negadas y las variables que valen 0 lógico se escriben sin negar. Se debe observar que las formas SOP y POS son expresiones duales entre sí.

- En cada grupo se simplifican tantas variables como potencia de dos del número de unos agrupados, es decir, para un grupo de 8 celdas, su potencia de dos es 3 y por tanto se simplifican 3 variables.
- Si en una mapa VK todas las celdas poseen un mintermino, entonces su agrupación arroja un 1 lógico, producto de simplificar todas las variables. Igualmente, si en una mapa VK todas las celdas poseen un maxtérmino, entonces su agrupación arroja un 0 lógico, producto de simplificar todas las variables.
- La expresión simplificada se obtiene de concatenar cada término obtenido, por cada agrupación realizada, en la forma SOP o POS respectiva.

EJEMPLO

Simplificar, usando un mapa VK, la siguiente expresión:

$$f(A, B, C, D) = \sum m(0, 2, 4, 5, 7, 9, 10, 13, 15)$$

Partiendo de una forma SOP, se debe realizar la agrupación de minterminos asegurando la menor cantidad posible de grupos y cada grupo lo más grande posible, así los grupos obtenidos en la figura 4.10 son los siguientes:

Figura 4.10: Ejemplo de simplificación usando mapa VK, para forma SOP

- Grupo 0 y 4: A vale 0, B cambia, C vale 0, D vale 0, y el grupo es $\overline{A}\overline{C}\overline{D}$
- Grupo 2 y 10: A cambia, B vale 0, C vale 1, D vale 0, y el grupo es $\overline{B}C\overline{D}$
- Grupo 9 y 13: A vale 1, B cambia, C vale 0, D vale 1, y el grupo es $A\overline{C}D$
- Grupo 5, 7, 13 y 15: A cambia, B vale 1, C cambia, D vale 1, y el grupo es BD

Entonces la expresión SOP mínima es:

$$f(A, B, C, D) = \sum m(0, 2, 4, 5, 7, 9, 10, 13, 15) = \overline{A}\overline{C}\overline{D} + \overline{B}C\overline{D} + A\overline{C}D + BD$$

Esta misma función se puede simplificar en un mapa VK desde su forma POS

$$f(A, B, C, D) = \prod (1, 3, 6, 8, 11, 12, 14):$$

Figura 4.11: Ejemplo de simplificación usando mapa VK, para forma POS

- Grupo 6 y 14: A cambia, B vale 1, C vale 1, D vale 0, y el grupo es $(\overline{B} + \overline{C} + D)$
- Grupo 8 y 12: A vale 1, B cambia, C vale 0, D vale 0, y el grupo es $(\overline{A} + C + D)$

Entonces, la expresión POS mínima es:

$$f(A, B, C, D) = \prod (1, 3, 6, 8, 11, 12, 14) = (A + B + \overline{D}) (B + \overline{C} + \overline{D}) (\overline{B} + \overline{C} + D) (\overline{A} + C + D).$$

4.3.2. Implicantes

Se denomina **implicante** (I), a todo posible grupo con el cual se pueda cubrir mintérminos de una función. Por ejemplo, para la función $f(A, B, C, D) = \sum m(0, 2, 4, 5, 7, 9, 10, 13, 15)$ se pueden realizar los grupos de 1, 2 y 4 mintérminos que se muestran en la figura 4.12 y se listan a continuación.

 Figura 4.12: Implicantes de $f(A, B, C, D) = \sum m(0, 2, 4, 5, 7, 9, 10, 13, 15)$

Posibles grupos de un mintérmino:

1. $m_0 = \overline{A}\overline{B}\overline{C}\overline{D}$
2. $m_2 = \overline{A}\overline{B}C\overline{D}$
3. $m_4 = \overline{A}B\overline{C}\overline{D}$
4. $m_5 = \overline{A}B\overline{C}D$
5. $m_7 = \overline{A}BCD$
6. $m_9 = A\overline{B}\overline{C}\overline{D}$

$$7. m_{10} = \overline{A}\overline{B}\overline{C}\overline{D}$$

$$8. m_{13} = \overline{A}\overline{B}\overline{C}D$$

$$9. m_{15} = \overline{A}\overline{B}\overline{C}D$$

Posibles grupos de dos mintérminos:

$$1. m_0, m_4 = \overline{A}\overline{C}\overline{D}$$

$$2. m_0, m_2 = \overline{A}\overline{B}\overline{D}$$

$$3. m_2, m_{10} = \overline{B}\overline{C}\overline{D}$$

$$4. m_4, m_5 = \overline{A}\overline{B}\overline{C}$$

$$5. m_5, m_7 = \overline{A}\overline{B}D$$

$$6. m_5, m_{13} = \overline{B}\overline{C}D$$

$$7. m_7, m_{15} = \overline{B}CD$$

$$8. m_{13}, m_{15} = \overline{A}BD$$

$$9. m_9, m_{13} = \overline{A}\overline{C}D$$

Posibles grupos de cuatro mintérminos

$$1. m_5, m_7, m_{13}, m_{15} = BD$$

De lo anterior, se tiene un total de 19 diferentes agrupaciones para los mintérminos de la función. En estos posibles grupos, existen unos que contienen a los otros, lo cual, desde una perspectiva de un diagrama de Venn, quiere decir que existen unos grupos que son subconjuntos de otros grupos. Los grupos que son subconjunto de otros no deben ser parte de la respuesta, toda vez que existe un grupo más grande que a su vez los contiene. Se define, entonces, **implicante primo** (I.P.) a todo grupo que no es subconjunto de otro grupo; en el ejemplo que se está examinando, todos los grupos de un uno están contenidos en algún grupo de dos unos, a su vez los grupos de dos mintérminos: $\overline{A}BD$, $\overline{B}\overline{C}D$, $\overline{B}CD$ y $\overline{A}\overline{C}D$ están contenidos en el grupo de cuatro mintérminos BD , por lo tanto, son implicantes primos los siguientes:

$$1. \overline{A}\overline{C}\overline{D}$$

$$2. \overline{A}\overline{B}\overline{D}$$

$$3. \overline{B}\overline{C}\overline{D}$$

$$4. \overline{A}\overline{B}\overline{C}$$

$$5. \overline{A}\overline{C}D$$

$$6. BD$$

Con los implicantes primos se tiene un conjunto de grupos que cubren a todos los minterminos de la función, por tanto, solo basta encontrar el menor conjunto posible de estos que cubran todos los minterminos. Para encontrar este conjunto mínimo, se hace necesario inicialmente buscar entre los implicantes primos aquellos que cubren de forma exclusiva algún mintermino de la función. Para el ejemplo, el grupo BD cubre de forma exclusiva los minterminos m_7 y m_{15} , y el grupo $A\bar{C}D$ cubre exclusivamente al mintermino m_9 . Como estos implicantes cubren de forma exclusiva algunos minterminos, deben ir en la respuesta y reciben el nombre de **implicantes primos esenciales** (I.P.E.).

La respuesta de una función minimizada (que también recibe el nombre de **cubierta**) sobre un mapa VK, siempre consta de la suma de dos partes, a saber:

- Todos los implicantes primos esenciales.
- El mínimo conjunto de implicantes primos necesarios para cubrir los minterminos no cubiertos por los implicantes primos esenciales.

Aunque la minimización consiste en encontrar estas dos partes descritas para la cubierta de la función, se han diseñado algoritmos encaminados a guiar este proceso, los cuales se presentan a continuación.

4.3.3. Algoritmo de minimización por vecindades

En busca de un procedimiento que permita entregar determinísticamente la respuesta mínima de una función, mediante el empleo de mapas VK, se plantea inicialmente un algoritmo que busca los implicantes primos esenciales desde la perspectiva de encontrar primero los minterminos más solos, lo cual posibilita que sean I.P.E. El procedimiento es el siguiente:

1. Listar los minterminos de la función ordenados de acuerdo con el número de minterminos vecinos que posee cada uno.
2. Seleccionar uno de los minterminos más solos.
3. Formar el grupo más grande posible con dicho mintermino. Si existe la posibilidad de formar más de un grupo, se debe seleccionar aquel que cubra más minterminos aún no cubiertos.
4. Iterar en 2 y 3 hasta cubrir todos los minterminos de la función.

EJEMPLO

Aplicar el algoritmo de minimización por vecindades para simplificar la expresión:

$$f(A, B, C, D) = \sum m(0, 2, 4, 5, 7, 9, 10, 13, 15)$$

En la figura 4.13, se observa la disposición inicial de los minterminos de la función. Aplicando el algoritmo se tiene:

Paso 1 Lista de minterminos según número de vecinos:

- Con un vecino: m_9, m_{10}
- Con dos vecinos: $m_0, m_2, m_4, m_7, m_{15}$
- Con tres vecinos: m_5, m_{13}

Paso 2 y 3, Iteración a Seleccionado m_9 , solo es posible hacer el grupo $m_9, m_{13} = \overline{A}\overline{C}D$

Paso 2 y 3, Iteración b Seleccionando m_{10} solo es posible hacer el grupo $m_2, m_{10} = \overline{B}C\overline{D}$

Paso 2 y 3, Iteración c Como ya están cubiertos todos los minterminos de un vecino, se continúa con los de dos vecinos: $m_0, m_2, m_4, m_7, m_{15}$. De este grupo m_2 ya está cubierto, y entre los demás m_7 entrega la posibilidad del grupo con más minterminos no cubiertos, así: $m_5, m_7, m_{15}, m_{13} = BD$.

Paso 2 y 3, Iteración d En los minterminos con dos vecinos faltan m_0 y m_4 , que se pueden agrupar entre sí para formar el grupo más grande posible y con más minterminos aún no cubiertos. $m_0, m_4 = \overline{A}\overline{C}\overline{D}$

Paso 4 Finalmente, con todos los minterminos cubiertos, la función simplificada es:

$$f(A, B, C, D) = \overline{A}\overline{C}D + \overline{B}C\overline{D} + BD + \overline{A}\overline{C}\overline{D}$$

Figura 4.13: Simplificación por algoritmo de vecindades

4.3.4. Algoritmo de minimización por I.P.E.

Se fundamenta en la misma definición de los implicantes primos, usando el siguiente procedimiento:

1. Dibujar todos los implicantes de la función.
2. Seleccionar todos los implicantes primos de la función.
3. Seleccionar todos los implicantes primos esenciales.
4. Sobre un diagrama con sólo los implicantes primos esenciales, seleccionar el menor conjunto de implicantes primos que permitan terminar de cubrir los minterminos aún no cubiertos por los implicantes primos esenciales.

EJEMPLO

Aplicar el algoritmo de minimización por I.P.E. para simplificar la expresión:

$$f(A, B, C, D) = \sum m(0, 2, 4, 5, 7, 9, 10, 13, 15)$$

En la figura 4.14, se observa el procedimiento, según la siguiente descripción:

Paso 1 y 2 En el mapa de la izquierda, se muestran todos los implicantes primos de la función.

Paso 3 En el mapa del centro, se muestran solo los implicantes primos esenciales de la función.

Paso 4 En el mapa de la derecha, se completa el mapa del centro, mediante la selección del menor conjunto de implicantes primos (tomados desde el mapa de la izquierda) que permiten terminar de cubrir los minterminos no cubiertos por los implicantes primos esenciales. Finalmente, con todos los minterminos cubiertos la función simplificada es:

$$f(A, B, C, D) = A\overline{C}D + \overline{B}C\overline{D} + BD + \overline{A}\overline{C}\overline{D}$$

Figura 4.14: Simplificación por algoritmo de I.P.E.

4.3.5. Implicados

Se denomina **implicado** (también I), a todo posible grupo con el cual se pueda cubrir maxtérminos de una función. A semejanza de los implicantes, pueden existir implicados que contienen a otros implicados, con lo cual algunos grupos son subconjuntos de otros grupos. Los grupos que son subconjunto de otros no deben ser parte de la respuesta, toda vez que existe un grupo más grande que a su vez los contiene. Se define entonces, **implicado primo** (también I.P.), como a todo grupo que no es subconjunto de otro grupo.

Con los implicados primos se tiene un conjunto de grupos que cubren a todos los maxtérminos de la función, por tanto, solo basta encontrar el menor conjunto posible de estos implicados primos que cubran todos los maxtérminos. Para encontrar este conjunto mínimo, se hace necesario inicialmente buscar entre los implicados primos aquellos que cubren de forma exclusiva algún maxtérmino de la función, por lo que deben ir en la respuesta y reciben el nombre de **implicados primos esenciales** (también I.P.E.).

El procedimiento de minimización es igual al seguido con minterminos, e igualmente se pueden aplicar los algoritmos de minimización por vecindades y por implicados primos esenciales, teniendo presente que la respuesta se debe expresar en una forma POS, donde las variables con valor lógico 0 se escriben sin negar y las variables con valor lógico 1 se escriben negadas.

EJEMPLO

Aplicar el algoritmo de minimización por I.P.E. para simplificar la expresión:

$$f(A, B, C, D) = \prod M(0, 4, 5, 6, 7, 8, 9, 13, 15)$$

En la figura 4.15 se observa el procedimiento, según la siguiente descripción:

Paso 1 y 2 En el mapa de la izquierda, se muestran todos los implicados primos de la función.

Paso 3 En el mapa del centro, se muestran solo los implicados primos esenciales de la función.

Paso 4 En el mapa de la derecha, se completa el mapa del centro mediante la selección del menor conjunto de implicados primos que permiten terminar de cubrir los maxtérminos no cubiertos por los implicados primos esenciales. Finalmente, con todos los maxtérminos cubiertos la función simplificada es:

$$f(A, B, C, D) = (A + \overline{B})(\overline{B} + \overline{C})(\overline{A} + B + C)(B + C + D)$$

Figura 4.15: Simplificación por implicados para $f(A, B, C, D) = \prod M(0, 4, 5, 6, 7, 8, 9, 13, 15)$

4.3.6. Mapas VK con especificación incompleta

Para las funciones con especificación incompleta, los términos prescindibles se pueden considerar a conveniencia como minterminos, como maxtérminos, como ambos, o como ninguno. Los minterminos (o maxtérminos) de la función siempre se deben agrupar, los prescindibles se agrupan exclusivamente como una contribución a formar un grupo más grande.

EJEMPLO

Simplificar la función

$$f(A, B, C, D) = \sum m(0, 5, 7, 9, 10) + \sum d(2, 8, 13, 14, 15)$$

El mapa VK respectivo se muestra en la figura 4.16, donde se puede observar lo siguiente:

- El prescindible d_{13} se agrupa con el mintérmino m_9 , con la intención de formar un grupo más grande.
- Los prescindible d_2 y d_8 se agrupan con los mintérminos m_0 y m_{10} , con la intención de formar un grupo más grande. En este grupo se resalta el hecho que, efectivamente las celdas de las esquinas del mapa VK son vecinas entre sí, toda vez que la numeración de una a otra solo cambia en un bit y, además, forman contorno cerrado.
- Los prescindible d_{13} y d_{15} se agrupan con los mintérminos m_5 y m_7 , con la intención de formar un grupo más grande.
- El prescindible d_{14} no es necesario, ya que su uso implica un grupo adicional que no posee nuevos mintérminos sin cubrir.

		<i>AB</i>			
		00	01	11	10
<i>CD</i>	00	1			<i>d</i>
	01		1	<i>d</i>	1
	11		1	<i>d</i>	
	10	<i>d</i>		<i>d</i>	1

Figura 4.16: Mapa VK para $f(A, B, C, D) = \sum m(0, 5, 7, 9, 10) + \sum d(2, 8, 13, 14, 15)$

La simplificación de la función es:

$$f(A, B, C, D) = BD + \overline{B}\overline{D} + A\overline{C}D$$

EJEMPLO

La misma función anterior se puede simplificar en su forma POS, con lo cual

$$f(A, B, C, D) = \prod M(1, 3, 4, 6, 11, 12) \cdot \prod D(2, 8, 13, 14, 15)$$

El mapa VK respectivo se muestra en la figura 4.17, donde nuevamente se debe observar que los prescindibles solo se usan con la finalidad de conseguir grupos de mayor tamaño. Se resalta el empleo de la letra d para prescindibles que pueden ser tomados como 1 lógico, y de la letra D para prescindibles que pueden ser tomados como 0 lógico. Sin embargo, es común emplear la letra d , o el símbolo X , indistintamente de si se trata de prescindibles tomados como mintérminos o como maxtérminos. En el presente documento, la letra D no se seguirá usando para prescindibles, con el fin de evitar confusión con la variable D .

		<i>AB</i>			
		00	01	11	10
<i>CD</i>	00		0	0	<i>D</i>
	01	0		<i>D</i>	
	11	0		<i>D</i>	0
	10	<i>D</i>	0	<i>D</i>	

Figura 4.17: Mapa VK para $f(A, B, C, D) = \prod M(1, 3, 4, 6, 11, 12) \cdot \prod D(2, 8, 13, 14, 15)$

La simplificación de la función es:

$$f(A, B, C, D) = (\overline{B} + D)(A + B + \overline{D})(\overline{A} + \overline{C} + \overline{D})$$

4.3.7. Mapas VK con ingreso de variables

Los mapas VK son una herramienta útil para la simplificación de funciones de máximo 5 o 6 variables. Incluso mapas de 6 variables puede presentar dificultades para encontrar vecindades y formar agrupaciones. Una forma de solventar este problema es el uso de los denominados **mapas VK con ingreso de variables**.

La idea fundamental tras los mapas VK con ingreso de variables, consiste en llevar una función a una presentación en tabla de verdad donde el valor de la función se expresa en términos de una o más variables. Por ejemplo, para la función de tres variables $f(A, B, C) = \sum m(1, 3, 4, 5, 6)$ se puede expresar su tabla de verdad en función únicamente de dos de las tres variables. Para ello, la función debe estar en una forma SOP o POS canónica y se evalúa para cada una de las combinaciones de una tabla de verdad que depende solo de dos variables. Para el caso de la función de ejemplo, se puede obtener una tabla de verdad reducida sobre las variables A y B , tal como se muestra en la tabla 4.1.

Tabla 4.1: Tabla de verdad reducida para $f(A, B, C) = \sum m(1, 3, 4, 5, 6)$

A	B	f
0	0	C
0	1	C
1	0	1
1	1	$\overline{C}$

$$f(A, B, C) = \overline{A}\overline{B}C + \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C + A\overline{B}C$$

$$f(0, 0, C) = C$$

$$f(0, 1, C) = C$$

$$f(1, 0, C) = \overline{C} + C = 1$$

$$f(1, 1, C) = \overline{C}$$

La tabla de verdad que se obtiene se puede llevar a un mapa VK de dos variables (A y B), tal como se muestra en la figura 4.18. En la figura se ha empleado la igualdad $1 = C + \overline{C}$. Cada celda del mapa VK con ingreso de variables, se comporta como si fuera un mapa VK para una variable, es decir, como un mapa de dos celdas.

Figura 4.18: Mapas VK de 2 variables, con ingreso de una variable

Al igual que en un mapa VK normal, al realizar la simplificación se deben agrupar todas las variables ingresadas más los minterminos (o maxtérminos, si la forma es POS). La simplificación, para el caso del ejemplo, usando el mapa VK central de la figura 4.18 es:

$$f(A, B, C) = \overline{A}C + \overline{B}C + A\overline{C}$$

Esta simplificación también se puede realizar usando el mapa VK de la derecha en la figura 4.18, donde se sabe que $1 = C + \overline{C}$:

$$f(A, B, C) = \overline{A}C + A\overline{B} + A\overline{C}$$

EJEMPLO

Simplificar, usando mapas VK con ingreso de variables, la función:

$$f(A, B, C, D) = \sum m(0, 1, 2, 3, 5, 7, 8, 9, 12, 13) = \prod (4, 6, 10, 11, 14, 15)$$

Se obtiene la tabla de verdad reducida en función únicamente de las variables A , B y C , como se muestra en la tabla 4.2.

Tabla 4.2: Tabla de verdad reducida, ejemplo Mapa VK de 3 variables, con ingreso de una variable

A	B	C	f	$f(A, B, C, D) = \overline{A}\overline{B}C + \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C + ABC\overline{C}$
0	0	0	1	$f(0, 0, 0, D) = 1$
0	0	1	1	$f(0, 0, 1, D) = 1$
0	1	0	D	$f(0, 1, 0, D) = D$
0	1	1	D	$f(0, 1, 1, D) = D$
1	0	0	1	$f(1, 0, 0, D) = 1$
1	0	1	0	$f(1, 0, 1, D) = 0$
1	1	0	1	$f(1, 1, 0, D) = 1$
1	1	1	0	$f(1, 1, 1, D) = 0$

La figura 4.19 muestra, al lado izquierdo, el mapa para minterminos y donde se emplea la igualdad $1 = D + \overline{D}$. El mapa VK del lado derecho, es el mapa para los maxtérminos, donde se emplea la igualdad $0 = D \cdot \overline{D}$.

Figura 4.19: Mapas VK de 3 variables, con ingreso de una variable

De los mapas de la figura 4.19 se obtienen las expresiones SOP mínima y POS mínima:

$$\begin{aligned} f(A, B, C) &= \overline{A}\overline{B} + \overline{A}D + A\overline{C} \\ &= (A + \overline{B} + D)(\overline{A} + \overline{C}) \end{aligned}$$

Los mapas VK con ingreso de variables no solamente pueden quedar en términos de una variable. En la función del ejemplo anterior, se puede realizar una tabla de verdad donde los valores de la función queden en términos de dos variables, por ejemplo C , y D .

EJEMPLO

Simplificar, usando mapas VK con ingreso de dos variables, la función:

$$f(A, B, C, D) = \sum m(0, 1, 2, 3, 5, 7, 8, 9, 12, 13) = \prod (4, 6, 10, 11, 14, 15)$$

En la tabla 4.3 se muestra la tabla de verdad respectiva:

Tabla 4.3: Tabla de verdad reducida, ejemplo Mapa VK de 2 variables, con ingreso de dos variables

A	B	f	$f(A, B, C, D) = \overline{A}\overline{B}C + \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C + A\overline{B}C$
0	0	1	$f(0, 0, C, D) = 1$
0	1	$\overline{D}$	$f(0, 1, C, D) = D$
1	0	$\overline{C}$	$f(1, 0, C, D) = \overline{C}$
1	1	$\overline{C}$	$f(1, 1, C, D) = \overline{C}$

Al llevar esta tabla de verdad a un mapa VK con ingreso de variables para la forma SOP, queda un mapa como el que se muestra en la figura 4.20. En la simplificación se emplea la igualdad $1 = D + \overline{D}$.

Figura 4.20: Mapa VK de 2 variables, con ingreso de dos variables

El mapa VK entrega como simplificación:

$$f(A, B, C) = \overline{A}\overline{B} + \overline{A}D + A\overline{C}$$

Que es el mismo resultado para la forma SOP obtenido previamente.

Los mapas VK con ingreso de variables también se pueden emplear para la simplificación de funciones con especificación incompleta. En estos casos, se deben tener presente las siguientes igualdades:

- $1 = X + \overline{X}$
- $0 = X \cdot \overline{X}$
- $X + d = X + d\overline{X}$. La variable X OR un prescindible, debe entregar X si el prescindible es 0, o X si el prescindible es 1.
- $Xd = X(d + \overline{X})$. La variable X AND un prescindible, debe entregar X si el prescindible es 1, o X si el prescindible es 0.

EJEMPLO

Simplificar, empleando mapas VK con ingreso de variables, la función con prescindibles:

$$f(A, B, C, D) = \sum m(3, 6, 9, 10, 11) + \sum d(0, 1, 4, 7, 8)$$

La tabla de verdad reducida, y en función de tres variables, se muestra en la tabla 4.4.

Tabla 4.4: Tabla de verdad reducida, ejemplo Mapa VK de 3 variables, con ingreso de una variable y prescindibles

A	B	C	f	$f(A, B, C, D) = \overline{A}\overline{B}CD + \overline{A}B\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}CD + d\overline{A}\overline{B}\overline{C}\overline{D} + d\overline{A}\overline{B}C\overline{D} + d\overline{A}B\overline{C}\overline{D} + d\overline{A}BC\overline{D} + d\overline{A}\overline{B}C\overline{D}$
0	0	0	d	$f(0, 0, 0, D) = d\overline{D} + dD = d$
0	0	1	D	$f(0, 0, 1, D) = D$
0	1	0	$d\overline{D}$	$f(0, 1, 0, D) = d\overline{D}$
0	1	1	$d + \overline{D}$	$f(0, 1, 1, D) = dD + \overline{D} = d + \overline{D}$
1	0	0	$d + D$	$f(1, 0, 0, D) = d\overline{D} + D = d + D$
1	0	1	1	$f(1, 0, 1, D) = \overline{D} + D = 1$
1	1	0	0	$f(1, 1, 0, D) = 0$
1	1	1	0	$f(1, 1, 1, D) = 0$

El mapa VK respectivo, con ingreso de variables y términos prescindibles, se muestra en la figura 4.21.

		AB			
		00	01	11	10
C	0	d	$d\overline{D}$		$d + D$
	1	D	$d + \overline{D}$		$\overline{D} + D$

Figura 4.21: Mapa VK de 3 variables, con ingreso de una variable y prescindibles

En la figura 4.21 se debe tener presente las siguientes observaciones:

- El término $d\overline{D}$ no se emplea, ya que puede ser considerado como un maxtérmino, con $d = 0$.
- El término $d + \overline{D}$ se considera como 1 lógico, con $d = 1$.
- La agrupación de cuatro celdas se realiza sobre la variable D , con $d = D$.
- La agrupación de dos celdas es un grupo con dos mintérminos, o unos lógicos, con $d = 1$ y recordando que $\overline{D} + D = 1$.

Con base en las observaciones anteriores, la simplificación es:

$$f(A, B, C, D) = \overline{B}D + \overline{A}\overline{B} + \overline{A}BC$$

4.4. MINIMIZACIÓN POR QUINE - McCLUSKEY - PETRICK (Q-M-P)

Los algoritmos de minimización por vecindades y por I.P.E. basan su operación en identificar los I.P.E. y luego completar la solución mediante la búsqueda del menor conjunto de I.P. para formar la cubierta de la función. Este último paso descrito no es determinístico, y por tanto dificulta la programación e implementación de estos métodos en ordenador. Además, los métodos de minimización descritos, y los mapas Veitch-Karnaugh, se caracterizan por estar limitados a funciones de máximo 5 variables.

El método tabular de simplificación presentado originalmente por Willard Van Orman Quine y Edward J. McCluskey, o método (Q-M), no está limitado al número de variables de una función, es un método que posibilita la implementación algorítmica y tiene la ventaja de realizar simplificación simultánea de varias funciones que comparten el mismo conjunto de variables de entrada. En general, la minimización por el método Q-M, operando sobre varias funciones, entrega una solución mínima global de la implementación, en lugar de una solución mínima por función (mínimos locales).

4.4.1. Algoritmo para la minimización de Quine-McCluskey (Q-M)

Este método de minimización se basa en el algoritmo por I.P.E., ya visto en la sección 4.3.4. Sigue los siguientes pasos:

1. Ya que la numeración de vecindades en un diagrama de Venn sigue un código binario en formato Gray, inicialmente, se debe listar en una columna los minterminos (o maxtérminos) de la función agrupados según el número de unos de su representación binaria. En la columna que se obtiene, los minterminos (o maxtérminos) con cero unos, solo pueden ser vecinos de los que tienen un uno. Los minterminos (o maxtérminos) con un uno, solo pueden ser vecinos de los que tienen cero o dos unos, y así sucesivamente.
2. Evaluar las vecindades entre grupos adyacentes de la primera columna. Para lo anterior, se debe verificar la vecindad de cada mintermino (o máxtermino) de un grupo con sus grupos adyacentes. Si existe vecindad, entonces se anota en una nueva columna los minterminos (o maxtérminos) que se agrupan y se especifica con un guion (-) el bit que cambia y con un asterisco (*) los elementos agrupados de la columna previa. Este proceso se realiza hasta que no se encuentren más vecindades entre grupos adyacentes. Las nuevas columnas se identifican sucesivamente como 1-Cubo, 2-Cubo, etc.
3. Realizar el mismo proceso de comparación del punto 2, con los elementos de la nueva columna. Este proceso de comparación se continua hasta que no existan nuevas vecindades en la última de las columnas. Al final, todos los elementos no agrupados (sin señalar con un *) no están contenidos en ningún otro grupo y por tanto son implicants (o implicados) primos.
4. Realizar una nueva tabla donde las filas contienen cada uno de los implicants (o implicados) primos identificados en el punto 3, y las columnas cada uno de los minterminos (o maxtérminos) de la función. En esta tabla, identificar para cada impicante (o implicado) primo los minterminos (o maxtérminos) que cubre. Con lo anterior, los implicants (o implicados) primos que cubren de forma exclusiva un mintermino (o maxtérmino) son implicants (o implicados) primos esenciales y hacen parte de la respuesta.

5. Con base en los implicantes (o implicados) primos esenciales, identificar el total de minterminos (o maxtérminos) cubiertos y no cubiertos por ellos.
6. Para los minterminos (o maxtérminos) no cubiertos en el punto 5, identificar el mínimo conjunto de implicantes (o implicados) primos necesarios para cubrirlos.
7. La respuesta está compuesta del conjunto de implicantes (o implicados) primos esenciales identificados en el punto 4 y del conjunto de implicantes (o implicados) primos identificado en el punto 6.

EJEMPLO

Minimizar mediante el método de Quine-McCluskey la siguiente función:

$$f(A, B, C, D) = \sum m(2, 4, 6, 8, 9, 10, 12, 13, 15)$$

Para este ejemplo, además, se muestra la solución simultánea usando mapas VK con el método de I.P.E., con el fin de hacer el paralelo entre los dos algoritmos.

Partiendo con los pasos 1 y 2 del algoritmo de minimización por I.P.E., se obtienen todos los I.P., tal como se indica en la figura 4.22.

Por Q-M, los I.P. se obtienen producto de aplicar los pasos 1, 2 y 3 del respectivo algoritmo, así: inicialmente se lista en una columna los minterminos agrupados según el número de unos. Para este caso, los minterminos se agrupan en grupos de 1, 2, 3 y 4 unos. Seguidamente, se evalúan las vecindades, por ejemplo, el grupo 1 contra el grupo 2, donde m_2 es vecino

Figura 4.22: I.P. sobre mapa VK y Q-M

de m_6 formando la vecindad m_2m_6 , identificada en binario como 0 – 10. En esta vecindad, el guion identifica el bit que cambia y en la primera columna se identifica con el asterisco los minterminos agrupados. En la tabla 4.5, la columna denominada 1-Cubo es el resultado final de comparación de vecindades de la columna de minterminos. La columna denominada 2-Cubo, es el resultado de comparación de vecindades de la columna 1-Cubo. En la columna 2-Cubo se destaca que la vecindad $m_8m_9m_{12}m_{13}$ se obtiene dos veces, pero solo se anota una. Lo anterior sucede ya que, al agrupar en la columna 1-Cubo las vecindades m_8m_9 y $m_{12}m_{13}$, se obtiene lo mismo que al agrupar m_8m_{12} con m_9m_{13} .

Numero de 1's	minterminos	1-Cubo	2-Cubo
1	$m_2 0010^*$ $m_4 0100^*$ $m_8 1000^*$	$m_2 m_6 0 - 10IP2$ $m_2 m_{10} - 010IP3$ $m_4 m_6 01 - 0IP4$ $m_4 m_{12} - 100IP5$ $m_8 m_9 100 - ^*$ $m_8 m_{10} 10 - 0IP6$ $m_8 m_{12} 1 - 00^*$	$m_8 m_9 m_{12} m_{13} 1 - 0 - IP1$
2	$m_6 0110^*$ $m_9 1001^*$ $m_{10} 1010^*$ $m_{12} 1100^*$	$m_9 m_{13} 1 - 01^*$ $m_{12} m_{13} 1 - 01^*$	
3	$m_{13} 1101^*$	$m_{13} m_{15} 11 - 1IP7$	
4	$m_{15} 1111^*$		

Al final del proceso anterior, se identifican las vecindades no agrupadas (sin asterisco) lo cual las convierte en implicantes primos. Estos I.P. coinciden con los mismos identificados sobre el mapa VK de la figura 4.22.

Siguiendo con el paso 3 del algoritmo de minimización por I.P.E., se obtienen todos los implicantes primos que contienen un uno exclusivo, es decir, se identifican los I.P.E., tal como se muestra en la figura 4.23.

Figura 4.23: I.P.E. sobre mapa VK y Q-M

Este proceso de identificar los I.P.E. se realiza en los pasos 4 y 5 del algoritmo Q-M. Para lo anterior, se procede a realiza una nueva tabla con los implicantes primos en las filas y los minterminos en las columnas. Se pone una “X” para identificar los minterminos que cubre cada implicante primo. En la tabla 4.6, se puede observar que los minterminos m_9 y m_{15} solo están cubiertos por un único implicante primo, IP1 e IP7 en este caso, lo cual los vuelve esenciales.

Si IP1 e IP7 son esenciales, entonces hacen parte de la respuesta y se identifica con un asterisco los mintérminos que agrupan, que son m_8 , m_9 , m_{12} , m_{12} y m_{15} .

Tabla 4.6: Ejemplo Quine McCluskey por mintérminos, tabla de I.P.E.

IP	IPE	m_2	m_4	m_6	m_8	m_9	m_{10}	m_{12}	m_{13}	m_{15}
					*	*		*	*	*
IP1, $m_8m_9m_{12}m_{13}1 - 0-$	IPE				X	X		X	X	
IP2, $m_2m_60 - 10$		X		X						
IP3, $m_2m_{10} - 010$		X					X			
IP4, $m_4m_601 - 0$			X	X						
IP5, $m_4m_{12} - 100$			X					X		
IP6, $m_8m_{10}10 - 0$					X		X			
IP7, $m_{13}m_{15}11 - 1$	IPE								X	X

Ahora, la figura 4.24 muestra la solución completa por mapa VK, la cual consta de los I.P.E. más el mínimo conjunto de I.P. necesarios para cubrir todos los mintérminos faltantes.

Figura 4.24: Solución sobre mapa VK Vs QM

Para identificar, por Q-M, los implicantes primos restantes, y obtener la solución respectiva, se siguen los pasos 6 y 7 del algoritmo. Este conjunto mínimo está formado por los implicantes primos IP3 e IP4, tal como se puede observar en la tabla 4.7, con los cuales se termina de cubrir todos los mintérminos.

Tabla 4.7: Ejemplo Quine McCluskey por mintérminos, mintérminos cubiertos

IP	IPE	m_2	m_4	m_6	m_8	m_9	m_{10}	m_{12}	m_{13}	m_{15}
		*	*	*	*	*	*	*	*	*
IP1, $m_8m_9m_{12}m_{13}1 - 0-$	IPE				X	X		X	X	
IP2, $m_2m_60 - 10$		X		X						
IP3, $m_2m_{10} - 010$	+	X					X			
IP4, $m_4m_601 - 0$	+		X	X						
IP5, $m_4m_{12} - 100$			X					X		
IP6, $m_8m_{10}10 - 0$					X		X			
IP7, $m_{13}m_{15}11 - 1$	IPE								X	X

De la tabla anterior, se obtiene la solución completa así:

$$f = IP1 + IP7 + IP3 + IP4 = A\overline{C} + ABD + \overline{BCD} + \overline{ABD}$$

Una observación que se puede realizar sobre el algoritmo de minimización Q-M radica en que, en el paso 1, la numeración de mintérminos (o maxtérminos) se puede hacer tanto por la cantidad de unos que contiene su representación binaria, así como por la cantidad de ceros. Lo anterior radica en el hecho que, numerar por la cantidad de ceros lo único que hace es listar en sentido contrario al realizado por cantidad de unos.

A continuación, se muestra el procedimiento de minimización para una función en término de sus maxtérminos.

EJEMPLO

Minimizar mediante el método de Quine-McCluskey la siguiente función:

$$f(A, B, C, D) = \prod M(0, 5, 6, 7, 8, 10, 13, 15)$$

Inicialmente, se lista en una columna los maxtérminos según el número de unos. Para este caso, los maxtérminos se agrupan en grupos de 0, 1, 2, 3 y 4 unos. Seguidamente, se evalúan las vecindades y se determinan los implicados primos, como se observa en la tabla 4.8.

Tabla 4.8: Ejemplo Quine McCluskey por maxtérminos, tabla de vecindades

Numero de 1's	mintérminos	1-Cubo	2-Cubo
0	M_0000^*	$M_0M_8 - 000$ IP2	$M_5M_7M_{13}M_{15} - 1 - 1IP1$
1	M_81000^*	$M_8M_{10}10 - 0$ IP3	
2	M_50101^* M_60110^* $M_{10}1010^*$	$M_5M_701 - 1^*$ $M_5M_{13} - 101^*$ $M_6M_7011 - IP4$	
3	M_70111^* $M_{13}1101^*$	$M_7M_{15} - 111^*$ $M_{13}M_{15}11 - 1^*$	
4	$M_{15}1111^*$		

Ahora, se realiza una nueva tabla con los implicados primos en las filas y los maxtérminos en las columnas. Se pone una "X" para identificar los maxtérminos que cubren cada implicado primo. En la tabla 4.9 se puede observar que todos los implicados primos son esenciales.

Tabla 4.9: Ejemplo Quine McCluskey por maxtérminos, tabla de cubierta

IP	IPE	M_0	M_5	M_6	M_7	M_8	M_{10}	M_{13}	M_{15}
		*	*	*	*	*	*	*	*
IP1, $M_5M_7M_{13}M_{15} - 1 - 1$	IPE		X		X			X	X
IP2, $M_0M_8 - 000$	IPE	X				X			
IP3, $M_8M_{10}10 - 0$	IPE					X	X		
IP4, $M_6M_7011 -$	IPE			X	X				

De la tabla anterior, se obtiene la solución completa así:

$$F = IP1 + IP2 + IP3 + IP4 = (\overline{B} + \overline{D})(B + C + D)(\overline{A} + B + D)(A + \overline{B} + \overline{C})$$

4.4.2. Algoritmo de Petrick

En la solución obtenida desde la tabla 4.6, el conjunto mínimo de implicants primos se obtiene producto de un proceso de búsqueda no determinística. Lo ideal es contar con un procedimiento que permita inequívocamente encontrar este conjunto mínimo. El algoritmo de Petrick es un procedimiento que permite este objetivo; fue presentado por Stanley R. Petrick y mejorado luego por Insley B. Pyne y Edward Joseph McCluskey. La suma de los algoritmos Q-M y de Petrick, da origen al algoritmo Q-M-P. Los pasos del algoritmo de Petrick se describen a continuación:

1. Escribir una forma POS con tantos términos suma como mintérminos (o maxtérminos) falten por cubrir.
2. Cada término de la forma POS es la suma de los implicants (o implicados) primos que sirven para cubrir a cada uno de los mintérminos (o maxtérminos) faltantes.
3. Desarrollar los productos de la forma POS con el fin de llegar a una forma SOP, aplicando idempotencia.
4. Seleccionar el término de la forma SOP que implique el menor costo, es decir el que tenga menos elementos. Si hay más de uno, se debe seleccionar el término que elimine más variables.

EJEMPLO

Para la función $f(A, B, C, D) = \sum m(2, 4, 6, 8, 9, 10, 12, 13, 15)$, que condujo a la tabla 4.6, determinar el conjunto mínimo de implicants primos mediante el algoritmo de Petrick.

1. Con los mintérminos faltantes por cubrir m_2, m_4, m_6 y m_{10} , se debe realizar una forma POS de cuatro términos suma. En cada término suma van los implicants primos que cubren cada uno de los mintérminos. Por ejemplo, para cubrir el mintérmino m_2 se pueden usar los implicants primos $IP2$ e $IP3$, para el mintérmino m_4 se pueden usar $IP4$ e $IP5$, y así sucesivamente, arrojando:

$$Petrick = (IP2 + IP3)(IP4 + IP5)(IP2 + IP4)(IP3 + IP6)$$

2. Ahora se procede a llevar la forma POS a una SOP y se eliminan elementos repetidos (teorema de idempotencia):

$$\begin{aligned} Petrick &= (IP2IP4 + IP2IP5 + IP3IP4 + IP3IP5)(IP2IP3 + IP2IP6 + IP3IP4 + IP4IP6) \\ &= IP2IP3IP4 + IP2IP4IP6 + IP2IP3IP5 + IP2IP5IP6 + IP2IP3IP4IP5 + IP2IP4IP5IP6 + \\ &= IP2IP3IP4IP6 + IP3IP4 + IP3IP4IP6 + IP2IP3IP5IP6 + IP3IP4IP5 + IP3IP4IP5IP6 \end{aligned}$$

3. El término SOP de menor costo es $IP3IP4$, dado que solo requiere de dos implicants primos. Esta solución coincide claramente con la solución planteada previamente para esta misma función.

4.4.3. Algoritmo Q-M-P con prescindibles

Para la aplicación del algoritmo de simplificación Q-M-P, cuando una función posee términos prescindibles, se debe tener presente que estos términos solo interesan para formar grupos más grandes. Por tanto, los prescindibles se tienen en cuenta para la identificación de I.P., pero no como mintérminos que se deben cubrir necesariamente.

EJEMPLO

Minimizar, mediante el método Q-M-P, la siguiente función en forma SOP con prescindibles:

$$g(A, B, C, D) = \sum m(2, 3, 6, 8, 9, 10, 13) + \sum d(0, 7)$$

Inicialmente, se lista los mintérminos y prescindibles según el número de unos. Para este caso, se tienen presente los prescindibles con el fin de identificar grupos más grandes a los cuales ellos puedan contribuir. Seguidamente, se evalúan las vecindades y se determinan los implicantes primos, como se observa en la tabla 4.10.

Tabla 4.10: Ejemplo Q-M-P con prescindibles en forma SOP, tabla de vecindades

Numero de 1's	mintérminos	1-Cubo	2-Cubo
0	m_00000^*	$m_0m_200 - 0^*$ $m_0m_8 - 000^*$	$m_0m_2m_8m_{10} - 0 - 0$ IP1
1	m_20010^* m_81000^*	$m_2m_3001 - ^*$ $m_2m_60 - 10^*$ $m_2m_{10} - 010^*$ $m_8m_9100 -$ IP3 $m_8m_{10}10 - 0^*$	$m_2m_3m_6m_70 - 1 -$ IP2
2	m_30011^* m_60110^* m_91001^* $m_{10}1010^*$	$m_3m_70 - 11^*$ $m_6m_7011 - ^*$ $m_9m_{13}1 - 01$ IP4	
3	m_70111^* $m_{13}1101^*$		

Ahora, se realiza la tabla para identificar los I.P.E. y el mínimo conjunto necesario de I.P. para terminar de cubrir los mintérminos aún no cubiertos, como se observa en la tabla 4.11. En este caso, se debe observar que los términos prescindibles no se tienen en cuenta, ya que no es necesario cubrirlos todos, solo si aportan a formar un grupo más grande.

Tabla 4.11: Ejemplo Q-M-P con prescindibles en forma SOP, tabla de cubierta

IP	Solución	m_2	m_3	m_6	m_8	m_9	m_{10}	m_{13}
		*	*	*	*	*	*	*
IP1, $m_0m_2m_8m_{10} - 0 - 0$	IPE	X			X		X	
IP2, $m_2m_3m_6m_70 - 1 -$	IPE	X	X	X				
IP3, $m_8m_9100 -$					X	X		
IP4, $m_9m_{13}1 - 01$	IPE					X		X

Finalmente, la respuesta SOP mínima es:

$$g(A, B, C, D) = \overline{B} \overline{D} + \overline{A} C + A \overline{C} D$$

EJEMPLO

Minimizar, mediante el método Q-M-P, la siguiente función en forma POS con prescindibles:

$$g(A, B, C, D) = \prod M(1, 4, 5, 11, 12, 14, 15) + \prod D(0, 7)$$

Inicialmente, se lista los maxtérminos y prescindibles según el número de unos. Para este caso, se tienen presente los prescindibles con el fin de identificar grupos más grandes a los cuales ellos puedan contribuir. Seguidamente, se evalúan las vecindades y se determinan los implicados primos, como se observa en la tabla 4.12.

Tabla 4.12: Ejemplo Q-M-P con prescindibles en forma POS, tabla de vecindades

Numero de 1's	maxtérminos	1-Cubo	2-Cubo
0	$M_0 0000^*$	$M_0 M_1 000 - *$ $M_0 M_4 0 - 00 *$	$M_0 M_1 M_4 M_5 0 - 0 -$ IP1
1	$M_1 0001^*$ $M_4 0100^*$	$M_1 M_5 0 - 01 *$ $M_4 M_5 010 - *$ $M_4 M_{12} - 100$ IP2	
2	$M_5 0101$ $M_{12} 1100$	$M_5 M_7 01 - 1$ IP3 $M_{12} M_{14} 11 - 0$ IP4	
3	$M_7 0111$ $M_{11} 1011$ $M_{14} 1110$	$M_7 M_{15} - 111$ IP5 $M_{11} M_{15} 1 - 11$ IP6 $M_{14} M_{15} 111 -$ IP7	
4	$M_{15} 1111$		

Ahora, se realiza la tabla para identificar los I.P.E. y el mínimo conjunto necesario de I.P. para terminar de cubrir los maxtérminos aún no cubiertos, como se observa en la tabla 4.13. En este caso, se debe observar que los términos prescindibles no se tienen en cuenta, ya que no es necesario cubrirlos todos, solo si aportan a formar un grupo más grande.

Tabla 4.13: Ejemplo Q-M-P con prescindibles en forma POS, tabla de I.P.E.

IP	Solución	M_1	M_4	M_5	M_{11}	M_{12}	M_{14}	M_{15}
		*	*	*	*			*
IP1, $M_0M_1M_4M_50 - 0-$	IPE	X	X	X				
IP2, $M_4M_{12} - 100$			X			X		
IP3, $M_5M_701 - 1$				X				
IP4, $M_{12}M_{14}11 - 0$						X	X	
IP5, $M_7M_{15} - 111$								X
IP6, $M_{11}M_{15}1 - 11$	IPE				X			X
IP7, $M_{14}M_{15}111-$							X	X

En la tabla 4.13, se observa que faltan por cubrir los maxtérminos M_{12} y M_{14} , por lo que se realiza el procedimiento de Petrick con el fin de identificar el mínimo conjunto de I.P. necesarios para terminar de cubrir estos maxtérminos. La solución de Petrick entrega:

$$Petrick = (IP2 + IP4)(IP4 + IP7) = IP2 * IP4 + IP2 * IP7 + IP4 + IP4 * IP7$$

En esta solución, el conjunto mínimo está formado únicamente por el $IP4$, con lo cual se termina la cubierta, como se muestra en la tabla 4.14.

Tabla 4.14: Ejemplo Q-M-P con prescindibles en forma POS, tabla de cubierta

IP	Solución	M_1	M_4	M_5	M_{11}	M_{12}	M_{14}	M_{15}
		*	*	*	*	+	+	*
IP1, $M_0M_1M_4M_50 - 0-$	IPE	X	X	X				
IP2, $M_4M_{12} - 100$			X			X		
IP3, $M_5M_701 - 1$				X				
IP4, $M_{12}M_{14}11 - 0$	IP +					X	X	
IP5, $M_7M_{15} - 111$								X
IP6, $M_{11}M_{15}1 - 11$	IPE				X			X
IP7, $M_{14}M_{15}111-$							X	X

Finalmente, la respuesta POS mínima es:

$$g(A, B, C, D) = IP1 + IP6 + IP4 = (A + C) * (\bar{A} + \bar{C} + \bar{D}) * (\bar{A} + \bar{B} + D)$$

4.5. LA FUNCIÓN MÍNIMA

De la aplicación de los métodos de simplificación vistos, se puede obtener una forma SOP mínima, o POS mínima, según el caso. Sin embargo, lo ideal es encontrar siempre entre las formas mínimas la que presente el costo mínimo, la cual se denomina la función mínima. Este proceso es más evidente y crítico cuando se trata con funciones que contienen prescindibles.

Para una función con especificación incompleta, se debe verificar la forma mínima para las representaciones SOP y POS sin, y con, términos prescindibles. Este proceso de comparación se facilita sobre un método gráfico de simplificación como los mapas VK, aunque el método de Q-M-P también incluirá automáticamente los prescindibles que sean favorables a la minimización.

En los ejemplos anteriores para la sección 4.4.3, se simplificó la misma función con prescindibles tanto en forma SOP, como en forma POS, con los siguientes resultados:

$$\begin{aligned} g(A, B, C, D) &= \sum m(2, 3, 6, 8, 9, 10, 13) + \sum d(0, 7) \\ &= \prod M(1, 4, 5, 11, 12, 14, 15) + \prod D(0, 7) \end{aligned}$$

Forma SOP mínima: $g(A, B, C, D) = \overline{B}\overline{D} + \overline{A}C + A\overline{C}D$, posee tres términos producto y un total de 7 literales

Forma POS mínima: $g(A, B, C, D) = (A + C) * (\overline{A} + \overline{C} + \overline{D}) * (\overline{A} + \overline{B} + D)$, posee tres términos suma y un total de 8 literales

Función mínima: Aunque ambas formas poseen igual número de términos (primer criterio de comparación), la forma SOP mínima posee menos literales (segundo criterio de comparación), por tanto, la forma SOP mínima es la función mínima.

EJEMPLO

Encontrar la función mínima para una implementación que determine si un número en formato BCD natural es mayor o igual que 5. Inicialmente, se muestran las representaciones SOP y POS para esta función, además, la tabla 4.15 enseña la tabla de verdad respectiva.

$$\begin{aligned} f(A, B, C, D) &= \sum m(5, 6, 7, 8, 9) + \sum d(10, 11, 12, 13, 14, 15) \\ &= \prod M(0, 1, 2, 3, 4) \cdot \prod D(10, 11, 12, 13, 14, 15) \end{aligned}$$

En la figura 4.25, se observa la simplificación por mintérminos y por maxtérminos. La forma SOP mínima requiere de tres términos producto, mientras la forma POS mínima solo requiere de dos términos suma, con los cual la forma POS sin prescindibles es la función mínima.

Tabla 4.15: Tabla de verdad para BCD natural ≥ 5

#	A	B	C	D	f
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	d
11	1	0	1	1	d
12	1	1	0	0	d
13	1	1	0	1	d
14	1	1	1	0	d
15	1	1	1	1	d

Figura 4.25: Obtención de función mínima

La función mínima es:

$$f(A, B, C, D) = (A + C + D)(A + B)$$

4.6. EJERCICIOS PROPUESTOS

1. Obtener la forma SOP mínima de las siguientes funciones:

a) $f(A, B) = \overline{A}\overline{B} + A\overline{B} + AB$

b) $f(A, B, C) = \overline{A}\overline{B}\overline{C} + \overline{A}B\overline{C} + AB\overline{C} + \overline{A}\overline{B}C + A\overline{B}C$

c) $f(A, B, C, D) = \sum m(0, 2, 4, 8, 9, 10, 11, 13, 15)$

d) $f(A, B, C, D) = \sum m(0, 1, 4, 6, 8, 9, 12, 14)$

2. Obtener la forma POS mínima de las siguientes funciones:

a) $f(A, B) = (A + B)(A + \overline{B})(\overline{A} + B)$

b) $f(A, B, C) = (A + \overline{B} + C)(\overline{A} + \overline{B} + C)(A + B + \overline{C})(A + \overline{B} + \overline{C})(\overline{A} + B + \overline{C})$

c) $f(A, B, C, D) = \prod M(1, 3, 4, 5, 6, 7, 9, 11, 12, 13, 14, 15)$

d) $f(A, B, C, D) = \prod M(0, 1, 5, 7, 8, 10, 12, 14, 15)$

3. Obtener la forma mínima, según la función corresponda a una forma SOP o POS:

a) $f(A, B, C, D) = \sum m(1, 5, 8, 10, 12, 14) + \sum d(7, 9, 11, 15)$

b) $f(A, B, C, D) = \sum m(2, 3, 5, 7, 8) + \sum d(10, 11, 12, 13, 14, 15)$

c) $f(A, B, C, D) = \prod M(1, 2, 3, 5, 7) \cdot \prod D(0, 8, 10, 13, 15)$

d) $f(A, B, C, D) = \prod M(0, 1, 7, 8, 10, 11, 12, 14) \cdot \prod D(3, 5, 13, 15)$

4. Simplificar usando ingreso de variables:

a) La función del punto 2b, ingresando una variable

b) La función del punto 3b, ingresando una variable

c) La función del punto 2c, ingresando una variable

d) La función del punto 3c, ingresando una variable

e) La función del punto 2d, ingresando dos variables

f) La función del punto 3d, ingresando dos variables

5. Simplificar usando el algoritmo de Q-M-P:

a) $f(A, B, C, D) = \sum m(0, 1, 3, 5, 8, 11, 13, 14)$

b) $f(A, B, C, D) = \sum m(0, 1, 7, 8, 9, 10, 12, 14)$

c) $f(A, B, C, D) = \prod M(3, 5, 7, 9, 11, 13, 15)$

d) $f(A, B, C, D) = \prod M(2, 3, 6, 8, 9, 10, 11, 12, 13)$

6. Encontrar la función mínima, mediante el método indicado:

a) $f(A, B, C, D) = \sum m(0, 1, 4, 6, 9, 10, 12, 14) + \sum d(3, 5, 7, 11, 13, 15)$, usando mapas VK

b) $f(A, B, C, D, E) = \prod M(0, 2, 5, 8, 10, 13, 16, 18, 21, 29, 30, 31) \cdot \prod D(7, 23, 24, 26)$, usando mapas VK

c) $f(A, B, C, D) = \sum m(1, 2, 4, 5, 7, 9, 10) + \sum d(3, 8, 15)$, usando Q-M-P

d) $f(A, B, C, D, E) = \prod M(0, 1, 4, 5, 8, 9, 10, 11, 16, 17, 20, 21, 23, 27) \cdot \prod D(15, 31, 26)$, usando Q-M-P

5

CAPÍTULO
CINCO

Parámetros de compuertas y complejidad circuital

Debido a que el alcance del presente libro cubre diversas áreas de la ingeniería y de aplicación, este capítulo explora el comportamiento de los sistemas lógicos según su contexto de aplicación. En esos comportamientos se incluyen los niveles de aceptación de umbrales para valores lógicos y los retardos de tiempo, o propagación de señales, a través de elementos físicos. Estos comportamientos dan origen a las denominadas familias lógicas, y se exploran sus características y compatibilidades. Para todo tipo de aplicación, es fundamental poder diferenciar y clasificar entre soluciones convenientes, relacionadas con el tiempo de retardo acumulado, el tamaño de elementos o porciones de código empleadas, entre otras; para lo cual este capítulo enseña la complejidad, la profundidad, el control de riesgos de tiempo e implementaciones mínimas globales para determina la solución conveniente.

5.1. PARÁMETROS DE VOLTAJE Y CORRIENTE

Circuito lógico :	<i>Sistema formado por un cierto conjunto de dispositivos, que hacen operaciones lógicas sobre las señales de entrada para generar un determinado número de señales lógicas de salida.</i>
Nivel lógico :	<i>Rango de valores de voltaje, corriente o cualquier otra cantidad física, que representa el valor de una variable lógica (1 - 0 lógico).</i>
Compuerta lógica:	<i>Dispositivo que implementa una función booleana, de aritmética lógica o de funcionalidad lógica.</i>

En el desarrollo de circuitos lógicos, funciones lógicas o sentencias lógicas, las compuertas son los dispositivos o elementos que representan las funciones booleanas básicas, derivadas o compuestas

Figura 5.1: Parámetros de voltaje para una compuerta lógica

de las anteriores, con el fin de implementar la función o funciones deseadas. Las entradas a estos circuitos, y sus salidas, son señales de naturaleza lógica (1 - 0) con rangos aceptados de voltaje o corriente para cada uno de los valores lógicos. Estos rangos y parámetros se exponen a continuación.

5.1.1. Parámetros de voltaje

Ya que los dispositivos electrónicos físicos no son ideales, el valor aceptado para el 1 lógico no es exactamente un valor de voltaje o corriente, pero sí un rango; por ejemplo, para un 1 lógico un posible rango es que la señal esté entre 4 V y 5 V, y para un 0 lógico un posible rango es que la señal esté entre 1 V y 0 V. Al rango con valores mayores se le denomina el **Nivel Alto** (H, del inglés *high*) y al rango con los valores menores se le denomina el **Nivel Bajo** (L, del inglés *low*).

En la figura 5.1, se observa una señal de entrada a una compuerta negadora (NOT) y su respectiva señal de salida donde, para un nivel H a la entrada, se entrega un nivel L a la salida; y para un nivel L a la entrada, se entrega un nivel H a la salida. En la señal de entrada, existe un voltaje máximo de operación para el nivel H (V_{Hmax}) y un voltaje mínimo de operación para el nivel L (V_{Lmin}). A su vez, en la señal de entrada, existe un nivel hasta el cual el voltaje se puede reducir con el fin de seguir siendo aceptado como H (V_{IHmin}); y como esta es una compuerta negadora, este nivel H mínimo aceptado produce a la salida un nivel L máximo asegurado (V_{OLmax}). Igualmente, en la señal de entrada, existe un nivel hasta el cual el voltaje se puede incrementar con el fin de seguir siendo aceptado como L (V_{ILmax}); y como se tiene una compuerta negadora, este nivel L máximo aceptado produce a la salida un nivel H mínimo asegurado (V_{OHmin}). En lugar de representar cada vez las señales de entrada y salida, como en el lado izquierdo de la figura 5.1, se suele presentar el diagrama del lado derecho, donde se resumen los niveles de voltaje descritos.

Figura 5.2: Rangos de voltaje para una compuerta lógica

5.1.2. Márgenes de ruido

Los valores anteriores de voltaje definen intervalos en los cuales las compuertas presentan una operación confiable en presencia de ruido o para permitir acoplar compuertas entre sí. Es así como, para el nivel H, ya que una compuerta asegura a la salida un voltaje V_{OHmin} , este valor puede decaer hasta un voltaje igual a V_{IHmin} y aún se garantiza la correcta operación. A la diferencia de estos valores se le denomina margen de ruido para el nivel H (NM_H) con $NM_H = V_{OHmin} - V_{IHmin}$. A su vez, para el nivel L, ya que una compuerta asegura a la salida un voltaje V_{OLmax} , este valor puede crecer hasta un voltaje igual a V_{ILmax} y aún se garantiza la correcta operación. A este último intervalo se le denomina margen de ruido para el nivel L (NM_L) con $NM_L = V_{ILmax} - V_{OLmax}$. La zona comprendida entre V_{IHmin} y V_{ILmax} no garantiza un nivel H o L, y se le conoce como la zona de rango indeterminado. Además, se define como margen de ruido, al valor menor entre NM_H y NM_L , o $NM = \min(NM_H; NM_L)$, y es el valor absoluto de ruido que puede afectar en todo instante a las señales de voltaje para garantizar la operación confiable de los niveles lógicos. Estos intervalos de operación garantizada o indeterminada, se puede observar en la figura 5.2.

En todas las compuertas, aspectos físicos de operación como la temperatura, procesos de fabricación, etc., pueden alterar los umbrales de conmutación que, en todo caso, deben quedar cubiertos por los márgenes de ruido descritos.

EJEMPLO

Determinar los márgenes de ruido para una compuerta que posee los siguientes parámetros de voltaje: $V_{OHmin} = 2,7$ V, $V_{IHmin} = 2,1$ V, $V_{OLmax} = 0,4$ V y $V_{ILmax} = 0,8$ V.

- $NM_H = V_{OHmin} - V_{IHmin} = 2,7 - 2,1 = 0,6$ V. Es decir, esta compuerta permite una caída de hasta 0,6 V para el nivel lógico H, en relación con el voltaje que ella entrega para ese nivel. En otras palabras, una vez la compuerta entrega 2,7 V en su salida, este valor puede decaer en 0,6 V, hasta 2,1 V, antes de ser entregado a la compuerta siguiente del mismo tipo.

- $NM_L = V_{ILmax} - V_{OLmax} = 0,8 - 0,3 = 0,5$ V. Es decir, esta compuerta permite un incremento de hasta 0,5 V para el nivel lógico L, en relación con el voltaje que ella entrega para ese nivel. En otras palabras, una vez la compuerta entrega 0,3 V en su salida, este valor puede crecer en 0,5 V, hasta 0,8 V, antes de ser entregado a la compuerta siguiente del mismo tipo.
- $NM = \min(NM_H; NM_L) = \min(0,6; 0,5) = 0,5$ V. Es decir, esta es la máxima variación (ruido admisible) en todo momento en los valores de voltaje dados por la compuerta, antes de ser entregados a la compuerta siguiente.

5.1.3. Parámetros de corriente

Además de la compatibilidad en los niveles lógicos, para poder realizar una conexión entre compuertas, es necesario determinar la cantidad de elementos que se pueden conectar como entrada a esa compuerta y la cantidad de elementos que dicha compuerta puede tener a su salida. Estos dos parámetros son los siguientes:

Fan-In: es el número máximo de entradas que puede tener una compuerta lógica sin degradar los niveles lógicos; es decir, sin que las señales se salgan de los márgenes de ruido admisibles.

Fan-Out: es el número máximo de compuertas que se pueden conectar a la salida sin degradar los niveles lógicos; es decir, sin que las señales se salgan de los márgenes de ruido admisibles.

Dependiendo del nivel lógico, una compuerta solicita, o entrega, los siguientes valores de corriente:

I_{IH} : valor de corriente pasando por la entrada, para el nivel lógico H.

I_{IL} : valor de corriente pasando por la entrada, para el nivel lógico L.

I_{OH} : valor de corriente pasando por la salida, para el nivel lógico H.

I_{OL} : valor de corriente pasando por la salida, para el nivel lógico L.

U_{LH} : unidad de carga para el nivel H, o corriente de la compuerta en el estado H.

U_{LL} : unidad de carga para el nivel L, o corriente de la compuerta en el estado L.

Para determinar el Fan-In se toma $\min\left(\frac{I_{IH}}{U_{LH}}, \frac{I_{IL}}{U_{LL}}\right)$ y, similarmente, para determinar el Fan-Out se toma $\min\left(\frac{I_{OH}}{U_{LH}}, \frac{I_{OL}}{U_{LL}}\right)$. Es decir, se busca la limitante, entre los dos niveles lógicos, de la corriente que fluye contra la corriente demanda en el nivel lógico, tanto a la entrada como a la salida.

EJEMPLO

Para una determinada compuerta, $U_{LL} = 1$ mA y $U_{LH} = 30$ mA. Además: $I_{OH} = 300$ mA, $I_{IH} = 150$ mA, $I_{OL} = 8$ mA e $I_{IL} = 4$ mA. Determinar los valores de Fan-In y Fan-Out para esta compuerta.

- Fan-In: $\min\left(\frac{I_{IH}}{U_{LH}}, \frac{I_{IL}}{U_{LL}}\right) = \min\left(\frac{150}{30}, \frac{4}{1}\right) = 4$. Por tanto, esta compuerta puede tener hasta 4 entradas conectadas.
- Fan-Out: $\min\left(\frac{I_{OH}}{U_{LH}}, \frac{I_{OL}}{U_{LL}}\right) = \min\left(\frac{300}{30}, \frac{8}{1}\right) = 8$. Por tanto, esta compuerta puede tener hasta 8 compuertas conectadas a su salida.

5.2. PARÁMETROS DINÁMICOS

5.2.1. Tiempos de retardo y de propagación

La respuesta de una compuerta, ante un cambio en las entradas, no es inmediata. Por ejemplo, para una compuerta negadora, el comportamiento en el tiempo de las señales de entrada y salida son como los que se muestran en la figura 5.3. Este comportamiento es igual si, en lugar de una negadora, se usa una compuerta NAND con una de las dos entradas a 1 lógico y la otra libre.

Figura 5.3: Parámetros de tiempo en una compuerta lógica

También, la señal de salida no responde de forma instantánea a una solicitud de cambio en su nivel lógico, en su lugar, le toma un tiempo hacer estos cambios. En general, los tiempos para cambio de un nivel lógico a otro son evidentes en la señal de salida y se denominan **tiempos de retardo**. Con el fin de evaluar estos tiempos en zonas de pendiente aproximadamente constante, se toman entre el 10% y el 90% del valor de voltaje de la salida, así:

t_f : tiempo de retardo que le toma a señal de salida caer desde de su nivel alto, hacia su nivel bajo.

t_r : tiempo de retardo que le toma a señal de salida subir desde su nivel bajo, hacia su nivel alto.

Además, de los anteriores tiempos, está la diferencia en respuesta de la compuerta desde el instante en que la señal de entrada ocasiona un cambio en la señal de salida. Ya que en este ejemplo se está analizado una compuerta negadora, cuando la señal de entrada pasa de nivel L a nivel H, la señal de salida debe pasar de nivel H a nivel L; igualmente, cuando la señal de entrada pasa de nivel H a nivel L, la señal de salida debe pasar de nivel L a nivel H. Estos cambios toman un tiempo y se conocen como tiempos de propagación de la respuesta, o simplemente **tiempos de propagación**. Su medición se realiza desde el 50% del voltaje de entrada que solicita cambio, hasta el 50% del voltaje de salida donde efectivamente se ve el cambio, así:

t_{pHL} : tiempo de propagación de la señal de entrada hacia la señal de salida, para que esta última pase de un nivel H a un nivel L.

t_{pLH} : tiempo de propagación de la señal de entrada hacia la señal de salida, para que esta última pase de un nivel L a un nivel H.

Para efectos de análisis posteriores, en lugar de trabajar con estos dos parámetros, es común tomar su promedio o **tiempo de propagación promedio**:

$$t_{pmean} = \frac{t_{pHL} + t_{pLH}}{2}$$

El tiempo de propagación en una compuerta limita la frecuencia máxima a la que puede trabajar, ya que $f_{max} = 1/t_{pmean}$. Entre menor sea el retardo de una compuerta, mayor será la frecuencia de trabajo y, por tanto, podrá realizar mayor cantidad de operaciones por unidad de tiempo.

5.2.2. Disipación de potencia

Para una compuerta, la disipación de potencia es la potencia promedio demandada en los ciclos de trabajo en nivel alto y nivel bajo. Se define así:

I_{CH} : corriente de alimentación de la compuerta cuando la salida está en nivel H.

I_{CL} : corriente de alimentación de la compuerta cuando la salida está en nivel L.

DP : en un ciclo de trabajo, la disipación media de potencia es: $DP = V_{CC} * (I_{CH} + I_{CL})/2$.

En ciertas aplicaciones, se debe ponderar aspectos de velocidad, de consumo, o ambos. Para estos casos, existe el parámetro **producto retardo-potencia**, que es el producto del tiempo de propagación promedio y la disipación promedio de potencia. Cuanto menor es este producto, mejor será la tecnología que se emplea, pues tiene un menor tiempo promedio de retardo y una menor disipación de potencia.

5.3. FAMILIAS LÓGICAS Y COMPATIBILIDAD

5.3.1. Series de familias

Para la implementación de circuitos físicos, ya se ha visto que toda compuerta discrimina entre dos valores lógicos para los niveles H y L, mediante valores de voltaje y/o corriente. Estos valores se estandarizan por grupos de compuertas que implementan las funciones lógicas, recibiendo el nombre de **familias lógicas**. Estas compuertas, además, se entregan en paquetes de varias de ellas, denominados **circuitos integrados** (IC) y que comparten características entre la misma familia.

El criterio de uniformidad, es decir, reducir al mínimo las diferencias entre las diferentes compuertas lógicas, es la base de la definición de una familia lógica, donde existe una estructura básica a partir de la cual se construyen las compuertas y circuitos integrados de esa familia. En esta estructura, se involucran los componentes internos y los valores de los parámetros de tal forma que los integrantes de una familia son compatibles entre sí. Para dar cumplimiento a las exigencias anteriores, actualmente se hace uso de dispositivos semiconductores, por lo que una posible clasificación de familias es:

Familias bipolares: emplean transistores bipolares y diodos, es decir, dispositivos de unión. Las familias más representativas son TTL y ECL.

Familias MOS: emplean transistores MOSFET, es decir, transistores de efecto de campo. Las familias más representativas son NMOS y CMOS.

La familia TTL ha sido de las más representativas, se basa en lógica bipolar transistor-transistor. Se introdujo en 1964 y ha marcado influencia en el desarrollo de características y parámetros de todas las familias existentes hoy en día. Esta familia nació con la Serie 74 que ya está obsoleta, luego se desprendió la Serie 74S la cual redujo los tiempos de retardo y también está obsoleta, posteriormente desde esta última serie se desprendieron las Series 74AS y 74LS. La Serie 74AS presenta mejoras en los procesos de fabricación, reducción en los tiempos de respuesta y en la disipación de potencia, está actualmente activa y es la más rápida de esta familia. La Serie 74LS es la misma serie 74S con menor consumo de potencia, pero más lenta. De la Serie 74LS se desprende la Serie 74ALS con mejoras en fabricación, velocidad y disipación de potencia. De esta última, se desprende la Serie 74F con mejoras en su diseño y fabricación.

La familia TTL basa su diseño en el BJT operando en modo saturado y con velocidades de conmutación limitadas. La familia ECL mejora lo anterior basando su diseño en el BJT operando en modo no saturado, por lo que hace la conmutación por corriente y mejora la velocidad. Para lograr lo anterior, esta familia requiere de polarización negativa, posee alta disipación de potencia y bajos márgenes de ruido.

La familia MOS basa su estructura en transistores de efecto de campo de metal-óxido. Algunas subfamilias relacionadas están ya obsoletas, como lo son las familias NMOS y PMOS. La familia actualmente distintiva es la CMOS (MOS Complementaria). La familia CMOS se distingue por ser de fácil fabricación a bajos costos, posee alta velocidad de operación, bajo consumo de potencia y da mayor posibilidad de integración. Esto se logra al no poseer elementos resistivos y tener muy bajo consumo de potencia, aunque es susceptible a daños por electro descargas. Este grupo de familias nació con la Serie 4000, con altos márgenes de ruido y baja velocidad, y ya está obsoleta. Luego, se desprendió la Serie 74C, que es de baja velocidad, compatible con las familias TTL, pero ya obsoleta. De la anterior, se derivaron las Series 74HC y 74HCT, las cuales se caracterizan por tener mejora significativa en velocidad y mayor capacidad de fan-out. La familia 74HCT posee niveles lógicos compatibles con TTL. De las series 74HC y 74HCT se derivan las Series 74AC, 74ACT, 74AHC, 74AHCT, BiCMOS, 74LVC, 74ALVC, 74LV y 74AVC. Estas familias se caracterizan por disminuir las pérdidas dinámicas y permitir reducir los voltajes de suministro, pasando desde originalmente 12 V, a valores de 5 V; 3,3 V; 2,5 V; 1,8 V; 1,5 V y 1,2 V.

A continuación, la tabla 5.1 muestra un resumen de parámetros de varias de las familias expuestas.

Tabla 5.1: Principales parámetros de las familias lógicas

Serie Familia	t_{pmean} [ns]	$max(t_f, t_r)$ [ns]	V_{IHmin} [V]	V_{ILmax} [V]	V_{OHmin} [V]	V_{OLmax} [V]	NM [V]
74	22		2.0	0.8	2.4	0.4	0.4
74LS	15		2.0	0.8	2.7	0.5	0.3
74F	5	2.3	2.0	0.8	2.7	0.5	0.3
74AS	4.5	1.5	2.0	0.8	2.7	0.5	0.3
74ALS	11	2.3	2.0	0.8	2.5	0.5	0.3
ECL	1.45	0.35	-1.165	-1.475	-1.025	-1.610	0.135
4000	250	90	3.5	1.5	4.95	0.05	1.45
74C	90		3.5	1.5	4.5	0.5	1.0
74HC	18	3.6	3.5	1.0	4.9	0.1	0.9
74HCT	23	3.9	2.0	0.8	4.9	0.1	0.7
74AC	9	1.5	3.5	1.5	4.9	0.1	1.4
74ACT	9	1.5	2.0	0.8	4.9	0.1	0.7
74AHC	3.7		3.85	1.65	4.4	0.44	0.55

5.3.2. Compatibilidad entre familias lógicas

Para que una familia sea compatible con otra, los niveles lógicos de salida de una de ellas (familia maestra) deben ser aceptados por los niveles lógicos de entrada de la siguiente (familia esclava). En general se deben cumplir las siguientes dos relaciones:

$$V_{OHmin} |_{familia\ maestra} \geq V_{IHmin} |_{familia\ esclava}$$

$$V_{OLmax} |_{familia\ maestra} \leq V_{ILmax} |_{familia\ esclava}$$

EJEMPLO

1. Verificar la compatibilidad de la Serie 74HC como familia maestra y la familia 74AS como familia esclava. Desde la tabla 5.1, se puede constatar que:

$$4,9 \geq 2,0$$

$$0,1 \leq 0,8$$

Ya que ambas relaciones se cumplen, la conexión de la familia Serie 74HC que alimenta a la familia Serie 74AS, es posible.

2. Verificar la compatibilidad de la Serie 74ALS como familia maestra y la familia 74HC como familia esclava. Desde la tabla 5.1, se puede constatar que:

$$2,5 \not\leq 3,5$$

$$0,5 \leq 1,0$$

Ya que una de las dos relaciones no se cumple, la conexión de la familia Serie 74ALS que alimenta a la familia Serie 74HC, no es posible.

5.4. CIRCUITOS INTEGRADOS - IC

Un circuito integrado es un encapsulado que posee cierta cantidad de elementos lógicos que pertenecen a la misma familia. La tabla 5.2 lista algunos de los circuitos integrados más comunes relacionados con compuertas lógicas booleanas, especificando el número de compuertas que trae y la cantidad de entradas disponibles por elemento.

Tabla 5.2: Circuitos integrados comunes para compuertas lógicas

Referencia	Función booleana	Número de compuertas	Entradas por componente
7400	NAND	4	2
7402	NOR	4	2
7404	NOT	6	1
7408	AND	4	2
7410	NAND	3	3
7420	NAND	2	4
7421	AND	2	4
7430	NAND	1	8
7432	OR	4	2
7451	2 AND a 1 NOR	2 Juegos	2 por AND
74H52	4 AND a 1 OR	1 Juego	2 por AND
7454	4 AND a 1 NOR	1 Juego	2 por AND
7486	XOR	4	2
74LS266	XNOR	4	2

5.5. COMPLEJIDAD Y PROFUNDIDAD CIRCUITAL

Al hablar del diseño de circuitos digitales, también se quiere hacer referencia a la implementación de algoritmos lógicos. Las compuertas lógicas juegan un papel importante en el diseño de circuitos digitales y, con ellas, la complejidad del hardware obtenido. Siempre que cualquier circuito digital se diseña usando compuertas lógicas, se enfrentan desafíos relacionados con el número de componentes individuales, número de IC empleados, tipos diferentes de elementos e IC, retados de tiempo en respuesta de los circuitos, etc.

Los desafíos descritos para hardware también tienen sus contrapartes cuando se está interesado en el diseño de algoritmos lógicos eficientes. Comparar dos algoritmos para una misma función, examinando el tiempo dedicado a cada uno no es la forma correcta, ya que este tiempo es relativo a las personas o a la tecnología de las máquinas. Para realizar procesos de comparación, se requiere además conocer si se desea una operación netamente secuencial, es decir, un paso a la vez, o si el hardware o software disponibles tiene un determinado paralelismo. Incluso, hoy se dispone de ordenadores que constan de muchos procesadores con procesamiento paralelo.

Para estos análisis en el sistema Booleano $\mathbb{B}$, se desea describir el conjunto de funciones $(f_n)_{n \in \mathbb{B}}$: $\{0, 1\}^n \rightarrow \{0, 1\}^m$, es decir, las funciones booleanas con n entradas que producen m salidas. Para la implementación de estas funciones, los circuitos constituyen un modelo para efectuar cálculos secuenciales y en paralelo, donde la complejidad de las funciones no difiere significativamente entre la complejidad del circuito y la complejidad del algoritmo. Ahora, la reducción del tiempo requerido para el cálculo de una función en particular se puede reducir, ya sea usando mejores ordenadores o mejores algoritmos, pero, como ya se dijo, es preferible determinar la complejidad de una función independientemente de la etapa de desarrollo de una tecnología.

Para una función f_n , se tienen 2^n diferentes combinaciones de entrada y un total de 2^{2^n} posibles funciones diferentes. Con el fin de aclarar, y recordando de la sección 3.1, f_2 es una función de dos variables, luego, existen $2^2 = 4$ posibles combinaciones de entrada y $2^{2^2} = 16$ posibles funciones diferentes, como se puede observar a continuación:

A	B	Z ₁	Z ₂	Z ₃	Z ₄	Z ₅	Z ₆	Z ₇	Z ₈	Z ₉	Z ₁₀	Z ₁₁	Z ₁₂	Z ₁₃	Z ₁₄	Z ₁₅	Z ₁₆
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Para toda f_2 , se puede observar que las 16 funciones posibles se pueden describir diciendo que Z_1 es una contradicción, Z_2 es la AND, Z_7 es la XOR, Z_8 es la OR, Z_9 es la NOR, Z_{10} es la XNOR, Z_{11} es la NOT de la variable B, Z_{13} es la NOT de la variable A, Z_{15} es la NAND y Z_{16} es una tautología.

Para representaciones de funciones lógicas f_n en formas algebraicas estándares, una revisión simple muestra que estas pueden tener una extensión exponencial de términos, ya que el máximo número de términos producto, o suma, es igual al máximo número posible de combinaciones de entrada, o 2^n .

Cuando se realizan operaciones aritméticas, se confía en un conjunto reducido de operaciones bien definidas, como la suma de dígitos, el uso de tablas de multiplicación, comparación entre dígitos y pruebas excluyentes SI-NO. Para el caso de la evaluación de funciones booleanas, se selecciona un conjunto finito (Ω) de operaciones base con una única salida. Las entradas a estas evaluaciones pueden ser variables y constantes lógicas. Un **paso de evaluación** es la aplicación de una de las operaciones base $\omega \in \Omega$ a datos ya evaluados, o a alguna de las entradas con datos ya evaluados.

Con base en lo anterior, un circuito C- Ω es aquel que opera sobre un número n fijo de entradas booleanas $(x_1, \dots, x_n)$ y que posee un número finito b de compuertas $(G(1), \dots, G(b))$, donde cada compuerta $G(i)$ implementa una de las operaciones base $\omega_i \in \Omega$. Las m salidas de un circuito C- Ω se representan por el vector $y = (y_1, \dots, y_m)$, que describe los cómputos realizados por el circuito.

Algunas veces, fuera de usar un diagrama lógico, es conveniente representar un circuito mediante un **gráfico acíclico dirigido**, que es un grafo que no posee ciclos; esto significa que, para cada vértice, no hay un camino directo que empiece y termine en él. Las entradas al C- Ω son las fuentes en este gráfico, los vértices hacen las veces de cada una de las compuertas $G(i)$ con $n(i)$ puntos de entrada desde las compuertas predecesoras. Esta definición se puede ilustrar mediante un circuito lógico que realiza la suma binaria de tres bits, $f(x_1, x_2, x_3) = (y_1, y_0)$, como se indica en la tabla 5.3.

Para diseñar este circuito, se puede obtener la forma SOP canónica para ambas salidas. Los gráficos acíclicos dirigidos que representan estas dos funciones se muestran en la figura 5.4, donde los vértices $G(1)$, $G(2)$ y $G(3)$ implementan compuertas negadoras para cada una de las tres variables y son las

Tabla 5.3: Tabla de verdad circuito que suma tres bits

x_1	x_2	x_3	y_1	y_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

mismas tanto para el gráfico del lado izquierdo (y_0) como para el del lado derecho (y_1). Los vértices $G(4)$ a $G(7)$ y $G(9)$ a $G(12)$ son compuertas AND que implementan los mintérminos respectivos y finalmente los vértices $G(8)$ y $G(13)$ son compuertas OR que implementan la suma en los términos SOP. Las funciones de salida son:

$$y_0 = \overline{x_1} \overline{x_2} x_3 + \overline{x_1} x_2 \overline{x_3} + x_1 \overline{x_2} \overline{x_3} + x_1 x_2 x_3$$

$$y_1 = \overline{x_1} x_2 x_3 + x_1 \overline{x_2} x_3 + x_1 x_2 \overline{x_3} + x_1 x_2 x_3$$

Figura 5.4: Gráfico acíclico dirigido para formas SOP canónicas

También es posible desarrollar, desde las formas SOP, usando álgebra booleana, expresiones donde se pueda reutilizar partes evaluadas en una salida, para obtener las otras. En este caso, la expresión $x_1 \oplus x_2$ se puede reutilizar, como se observa a continuación.

$$\begin{aligned} y_0 &= \overline{x_1} \overline{x_2} x_3 + \overline{x_1} x_2 \overline{x_3} + x_1 \overline{x_2} \overline{x_3} + x_1 x_2 x_3 \\ &= \overline{x_1} (\overline{x_2} x_3 + x_2 \overline{x_3}) + x_1 (\overline{x_2} \overline{x_3} + x_2 x_3) \\ &= \overline{x_1} (x_2 \oplus x_3) + x_1 (\overline{x_2} \oplus x_3) \\ &= x_1 \oplus x_2 \oplus x_3 = (x_1 \oplus x_2) \oplus x_3 \end{aligned}$$

$$\begin{aligned} y_1 &= \overline{x_1} x_2 x_3 + x_1 \overline{x_2} x_3 + x_1 x_2 \overline{x_3} + x_1 x_2 x_3 \\ &= x_3 (\overline{x_1} x_2 + x_1 \overline{x_2}) + x_1 x_2 (\overline{x_3} + x_3) \\ &= (x_1 \oplus x_2) x_3 + x_1 x_2 \end{aligned}$$

La figura 5.5 muestra el gráfico acíclico dirigido para los desarrollos con álgebra booleana, donde sobre un mismo gráfico se indican las dos funciones y los vértices $G(1)$ y $G(4)$ son la compuerta XOR.

Figura 5.5: Gráfico acíclico dirigido con reutilización

Fuera de las implementaciones mostradas en las figuras 5.4 y 5.5, existen muchos otros circuitos y expresiones algebraicas que evalúan la misma función. Lo importante es obtener los circuitos óptimos, para lo cual se requiere de criterios para comparar la eficiencia, y decidir en consecuencia. Si un circuito se emplea para evaluaciones secuenciales, el número de compuertas mide el tiempo de evaluación y si se asume que, por simplicidad, cada compuerta de las operaciones base $\omega_i \in \Omega$ tiene el mismo tiempo de retardo, entonces el tiempo de evaluación secuencial es el número de compuertas de la implementación. Sin embargo, el hardware de los sistemas de cómputo moderno tiene diferentes grados de paralelismo, e igual el software de despliegue para implementaciones embebidas. En la figura 5.5, los vértices $G(1)$ y $G(2)$ se pueden evaluar en paralelo, luego los vértices $G(3)$ y $G(4)$ y finalmente $G(5)$. Para este último caso, se dice que se requiere de 3 **pasos**, o **capas de evaluación**. Si se comparan las dos figuras en estudio, se puede observar que ambas requieren de tres pasos de evaluación, pero la gráfica para las formas SOP canónicas requiere de más compuertas, en este caso.

De lo expuesto previamente, se dice que la **complejidad** $C(S)$, de un circuito S , es igual al número de compuertas requeridas para su implementación. Esta complejidad de circuito de una función f_n se relaciona también a la base Ω de operaciones booleanas y se define como $C_\Omega(f_n)$, que es el menor número de compuertas de la base que permiten evaluar la función. De forma análoga, la **profundidad** $D(S)$ del circuito S es la ruta de mayor longitud en S y equivale al número de capas de evaluación, o pasos. Igualmente, la profundidad de f_n se relaciona a la base Ω y se define como $D_\Omega(f_n)$, que es la mínima profundidad de la base de compuertas que implementan f_n .

Para evaluaciones secuenciales, la complejidad circuital corresponde al tiempo de evaluación. Para sistemas con paralelismo, la complejidad mide el costo de la construcción, ya sea en elementos de circuito o instrucciones de código, y la profundidad corresponde al tiempo de evaluación. Sin importar cual sea el caso, es de interés tratar de minimizar simultáneamente tanto el tamaño (complejidad) como la profundidad. Las medidas de complejidad no dependen realmente de la base subyacente, si la base es lo suficientemente grande.

Las medidas de complejidad y profundidad definidas se basan sobre las operaciones base sin más restricciones. Sin embargo, estas medidas se pueden realizar imponiendo algunas restricciones adicionales relacionadas con la implementación de hardware, ya que no se dispone libremente de compuertas individuales, si no de grupos de ellas tal como se muestra en la tabla 5.2, además su disponibilidad se limita a un número de entradas determinado y la posibilidad de alimentar otras compuertas, lo cual impone restricciones dadas por el fan-in y el fan-out.

EJEMPLO

1. Presentar las medidas de complejidad y profundidad para las dos funciones de la figura 5.4. ¿Cómo se alteran estas medidas si se restringe el fan-in de las compuertas a dos entradas?

Para las dos funciones de la figura 5.4:

Complejidad: $C(S) = 13$, que es el número de compuertas requeridas para implementar el circuito S , compuesto por las funciones y_0 y y_1 . Se debe tener presente que las compuertas $G(1)$, $G(2)$ y $G(3)$ son las mismas para ambas funciones.

Profundidad: $D(S) = 3$, que es el número de capas de evaluación del circuito S , compuesto por las funciones y_0 y y_1 , y es además la ruta de mayor longitud en S .

Base de la complejidad $C_\Omega(f_n)$: en este caso, la base de funciones booleanas es $\{\text{NOT}, \text{AND}, \text{OR}\}$.

Para las dos funciones de la figura 5.4, pero restringiendo fan-in a dos entradas:

Complejidad: 25, que es el número de compuertas requeridas para implementar el circuito S , compuesto por las funciones y_0 y y_1 . En este caso, se debe tener presente que las compuertas $G(1)$, $G(2)$ y $G(3)$ son las mismas para ambas funciones. Las compuertas $G(4)$ a $G(7)$ y $G(9)$ a $G(12)$ son nodos con tres entradas, pero como se restringe el uso a compuertas de solo dos entradas, es necesario aplicar propiedad asociativa, con lo cual $xyz = (xy)z$, obligando a primero evaluar un par de entradas y luego la respuesta con la variable faltante; esto lleva a que cada una de las 8 compuertas en discusión se deban reemplazar con dos compuertas, o 16 nuevas compuertas de dos entradas. Las compuertas $G(8)$ y $G(13)$ son nodos de cuatro entradas, y al restringir a dos entradas y aplicar propiedad asociativa se tiene que $w + x + y + z = (w + x) + (y + z)$, obligando a evaluar dos pares de entradas y luego sumar sus resultados; esto último lleva a que cada una de las dos compuertas en discusión se deban reemplazar con tres compuertas, o 6 nuevas compuertas de dos entradas. Al sumar el total, se obtienen 25 compuertas.

Profundidad: $D(S) = 5$, que es el número de capas de evaluación del circuito S , compuesto por las funciones y_0 y y_1 . En este caso, y producto de usar solo compuertas de dos entradas, el reemplazo de las compuertas $G(4)$ a $G(7)$ y $G(9)$ a $G(12)$ necesita de una capa adicional e igual el reemplazo de las compuertas $G(8)$ y $G(13)$, por lo que a las 3 capas previas se debe sumar las dos capas nuevas por usar solo compuertas de dos entradas.

Base de la complejidad $C_\Omega(f_n)$: en este caso, la base de funciones booleanas es $\{\text{NOT}, \text{AND de 2 entradas}, \text{OR de 2 entradas}\}$.

2. Presentar las medidas de complejidad y profundidad para las dos funciones de la figura 5.5.

Complejidad: $C(S) = 5$, que es el número de compuertas requeridas para implementar el circuito S , compuesto por las funciones y_0 y y_1 .

Profundidad: $D(S) = 3$, que es el número de capas de evaluación del circuito S , compuesto por las funciones y_0 y y_1 , y es además la ruta de mayor longitud en S .

Claramente, estas medidas de complejidad y profundidad son mejores que las presentadas para el punto 1 y, por tanto, la implementación de la figura 5.5 es de menos costo.

Base de la complejidad $C_\Omega(f_n)$: en este caso, la base de funciones booleanas es $\{\text{AND de 2 entradas}, \text{OR de 2 entradas}, \text{XOR de 2 entradas}\}$.

5.6. LA IMPLEMENTACIÓN MÍNIMA GLOBAL

La implementación simultánea de varias funciones lógicas, con el mismo conjunto de variables de entrada, requiere de encontrar la forma mínima global, más no la mínima por función. Simplificar de forma individual cada función sólo garantiza una forma mínima para cada una, más no una simplificación global de las funciones, lo cual se puede verificar al revisar las medidas de complejidad y profundidad relacionadas. Mediante el algoritmo de Quine-McCluskey-Petrick (Q-M-P), se puede realizar una simplificación global.

Para el proceso de minimización de múltiples funciones con las mismas entradas, el algoritmo Q-M-P adicionan las siguientes observaciones:

- Dos mintérminos son vecinos, si fuera de tener solo el cambio de un bit además tienen funciones en común.
- Un mintérmino se considera agrupado sólo si se incluyen todas las funciones a que pertenece.

Para examinar, y mostrar la forma de obtener una implementación mínima global, se presenta el siguiente ejemplo.

EJEMPLO

Obtener la implementación mínima global para las siguientes dos funciones lógicas que comparten el mismo conjunto de variables de entrada. Comparar la solución con la simplificación individual por mapas VK, indicando la complejidad y profundidad si se restringe el fan-in a dos entradas.

$$\begin{aligned} f_{\alpha}(A, B, C, D) &= \sum m(4, 6, 7, 11, 12) + \sum d(14, 15) \\ f_{\beta}(A, B, C, D) &= \sum m(4, 5, 6, 11, 14) + \sum d(8, 12) \end{aligned}$$

Para una realización, simplificando de forma individual cada función, se obtiene el resultado que se muestra en la figura 5.6. En esta figura, el mapa de la izquierda muestra la simplificación de la función f_{α} y el mapa de la derecha la simplificación para f_{β} .

Los resultados de esta simplificación son:

$$\begin{aligned} f_{\alpha}(A, B, C, D) &= B\overline{D} + BC + ACD \\ f_{\beta}(A, B, C, D) &= B\overline{D} + \overline{A}B\overline{C} + A\overline{B}CD \end{aligned}$$

La implementación de estas funciones requiere de los siguientes elementos, restringiendo el fan-in a dos entradas y con $C_{\Omega}(f_n) = \{\text{NOT, AND de 2 entradas, OR de 2 entradas}\}$:

- La función f_α requiere de una compuerta NOT, dos compuertas OR de dos entradas y cuatro compuertas AND de dos entradas. Esta función se implementa con cuatro retardos de compuerta, o cuatro capas de evaluación: dos capas por el término ACD y dos capas para los términos suma. En este caso, como la negación está en un término de dos variables, no crea una capa adicional de retardo.
- La función f_β requiere de tres compuertas NOT, dos compuertas OR de dos entradas y cuatro compuertas AND de dos entradas. En este total de compuertas, se ha tenido

Figura 5.6: Ejemplo de implementación no mínima, con dos funciones

presente que los términos de dos literales $B\overline{D}$ y CD , implementados para la función f_α , se pueden reutilizar para la función f_β , así como la negación de la variable D . Esta función se implementa con cinco retardos de compuerta, o cinco capas de evaluación.

- El total de elementos es de cuatro compuertas NOT, cuatro compuertas OR y ocho compuertas AND. Por tanto, $C(S) = 16$ y $D(S) = 5$, que es la profundidad de la ruta más larga. Además, si se piensa en términos de circuitos integrados, de la tabla 5.2, se puede comprobar que se requiere de un IC 7404 (NOT), un IC 7432 (OR de 2 entradas) y dos IC 7408 (AND de 2 entradas), para un total de 4 IC. Este número de IC es una forma alterna de ver la complejidad, para implementaciones físicas.

Para la simplificación simultánea de las funciones usando el algoritmo de Q-M-P, se realiza la agrupación de todos los mintérminos involucrados, incluso los prescindibles, según su número de unos. Dos mintérminos son vecinos si fuera de tener solo el cambio de un bit, tienen funciones en común. Un mintérmino se considera agrupado sólo si se incluyen todas las funciones a que pertenece. La tabla 5.4 muestra el proceso descrito.

Tabla 5.4: Ejemplo Quine McCluskey para implementación mínima global con dos funciones, tabla de I.P.

Numero de 1's	mintérminos	1-Cubo	2-Cubo
1	$m_4 0100 f_\alpha f_\beta^*$ $m_8 1000 f_\beta^*$	$m_4 m_5 010 - f_\beta \text{IP3}$ $m_4 m_6 01 - 0 f_\alpha f_\beta^*$ $m_4 m_{12} - 100 f_\alpha f_\beta^*$ $m_8 m_{12} 1 - 00 f_\beta \text{IP4}$	$m_4 m_6 m_{12} m_{14} - 1 - 0 f_\alpha f_\beta \text{IP1}$ $m_6 m_7 m_{14} m_{15} - 11 - f_\alpha \text{IP2}$
2	$m_5 0101 f_\beta^*$ $m_6 0110 f_\alpha f_\beta^*$ $m_{12} 1100 f_\alpha f_\beta^*$	$m_6 m_7 011 - f_\alpha^*$ $m_6 m_{14} - 110 f_\alpha f_\beta^*$ $m_{12} m_{14} 11 - 0 f_\alpha f_\beta^*$	
3	$m_7 0111 f_\alpha^*$ $m_{11} 1011 f_\alpha f_\beta \text{IP6}$ $m_{14} 1110 f_\alpha f_\beta^*$	$m_7 m_{15} - 111 f_\alpha^*$ $m_{11} m_{15} 1 - 11 f_\alpha \text{IP5}$ $m_{14} m_{15} 111 - f_\alpha^*$	
4	$m_{15} 1111 f_\alpha^*$		

Ahora, se realiza la tabla con los implicantes primos en las filas y mintérminos en las columnas. En la tabla 5.5, se muestra la identificación de los implicantes primos esenciales, donde se puede observar que solo se han tenido presente los mintérminos y no los prescindibles, ya que estos últimos solo se usan en caso de poder realizar agrupaciones mayores. El implicante primo IP4 solo cubre prescindibles, por lo que no cubre ningún mintérmino.

Tabla 5.5: Ejemplo Quine McCluskey implementación mínima global con dos funciones, tabla de I.P.E.

IP	IPE	f_α					f_β				
		m_4	m_6	m_7	m_{11}	m_{12}	m_4	m_5	m_6	m_{11}	m_{14}
		*	*	*	*	*	*	*	*	*	*
IP1, $m_4 m_6 m_{12} m_{14} - 1 - 0 f_\alpha f_\beta$	IPE	X	X			X	X		X		X
IP2, $m_6 m_7 m_{14} m_{15} - 11 - f_\alpha$	IPE		X	X							
IP3, $m_4 m_5 010 - f_\beta$	IPE						X	X			
IP4, $m_8 m_{12} 1 - 00 f_\beta$											
IP5, $m_{11} m_{15} 1 - 11 f_\alpha$					X						
IP6, $m_{11} 1011 f_\alpha f_\beta$	IPE				X					X	

Como con implicantes primos esenciales se cubren todos los mintérminos, no se requiere emplear el algoritmo de Petrick, y las funciones resultantes son:

$$\begin{aligned}
 f_\alpha(A, B, C, D) &= IP1 + IP2 + IP6 = B\overline{D} + BC + A\overline{B}CD \\
 f_\beta(A, B, C, D) &= IP1 + IP3 + IP6 = B\overline{D} + \overline{A}B\overline{C} + A\overline{B}CD
 \end{aligned}$$

La implementación de estas funciones requiere de los siguientes elementos, con

$C_\Omega(f_n) = \{\text{NOT, AND de 2 entradas, OR de 2 entradas}\}$:

- La función f_α requiere de dos compuertas NOT, dos compuertas OR de dos entradas y cinco compuertas AND de dos entradas. Este proceso se realiza en cuatro capas de evaluación. Sin embargo, si se suman primero los términos $B\overline{D} + A\overline{B}CD$ para ser reutilizados en f_β , el proceso requiere cinco capas, que es como se toma, en definitiva.
- La función f_β requiere de dos compuertas NOT, una compuerta OR de dos entradas y dos compuertas AND de dos entradas. En este total de compuertas, se ha tenido presente que la expresión $B\overline{D} + A\overline{B}CD$ implementa para la función f_α se pueden reutilizar para la función f_β . El proceso requiere de cinco capas de evaluación.

- El total de elementos es de cuatro compuertas NOT, tres compuertas OR y siete compuertas AND. Por tanto, ahora $C(S) = 14$ y $D(S) = 5$. Además, si se piensa en términos de circuitos integrados, de la tabla 5.2, se puede comprobar que se requiere de un IC 7404 (NOT), un IC 7432 (OR de 2 entradas) y dos IC 7408 (AND de 2 entradas), para un total de 4 IC.
- Al comparar las dos soluciones, la profundidad de ambas es igual, pero la complejidad de la solución por Q-M-P es menor. Lo anterior demuestra que, la simplificación simultánea entrega una solución que permite implementaciones globales mínimas. En esta ocasión, se observa que, desde el punto de vista de IC, las dos soluciones requieren de la misma cantidad, aunque en la solución por Q-M-P sobran más elementos sin usar en los IC.

A continuación, se muestra un nuevo ejemplo, donde se observa que el algoritmo Q-M-P entrega la solución con la menor complejidad y profundidad, tanto desde el punto de vista de compuertas como desde IC.

EJEMPLO

Obtener la implementación mínima global para las siguientes tres funciones lógicas que comparten el mismo conjunto de variables de entrada. Comparar la solución con la simplificación individual por mapas VK, indicando la complejidad y profundidad si se restringe el fan-in a dos entradas.

$$\begin{aligned}
 f_{\alpha}(A, B, C, D) &= \sum m(0, 2, 7, 10) + \sum d(12, 15) \\
 f_{\beta}(A, B, C, D) &= \sum m(2, 4, 5) + \sum d(6, 7, 8, 10) \\
 f_{\gamma}(A, B, C, D) &= \sum m(2, 7, 8) + \sum d(0, 5, 13)
 \end{aligned}$$

Para una realización, simplificando de forma individual cada función, se obtiene el resultado que se muestra en la figura 5.7. En esta figura, el mapa de la izquierda muestra la simplificación de la función f_{α} , el mapa del centro la simplificación de la función f_{β} y el mapa de la derecha la simplificación para f_{γ} .

Los resultados de esta simplificación son:

Figura 5.7: Ejemplo de implementación no mínima, con tres funciones

$$\begin{aligned}
 f_\alpha(A, B, C, D) &= \overline{A}\overline{B}\overline{D} + \overline{B}C\overline{D} + BCD \\
 f_\beta(A, B, C, D) &= \overline{A}B + \overline{A}C\overline{D} \\
 f_\gamma(A, B, C, D) &= \overline{A}BD + \overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{D}
 \end{aligned}$$

La implementación de estas funciones requiere de los siguientes elementos, restringiendo el fan-in a dos entradas y con $C_\Omega(f_n) = \{\text{NOT, AND de 2 entradas, OR de 2 entradas}\}$:

- La función f_α requiere de tres compuertas NOT, dos compuertas OR de dos entradas y cinco compuertas AND de dos entradas, donde el término $\overline{B}\overline{D}$ se reutiliza dos veces. Esta función se implementa con cuatro retardos de compuerta, o cuatro capas de evaluación: dos capas en términos producto y dos capas para los términos suma.
- La función f_β no requiere NOT, pues se reutilizan desde f_α ; necesita una compuerta OR de dos entradas y tres compuertas AND de dos entradas. Se tiene presente que no se puede reutilizar términos previos, ya que se reutilizó $\overline{B}\overline{D}$, en lugar de $\overline{A}\overline{D}$, en la misma f_α . Esta función se implementa con cuatro retardos de compuerta.
- La función f_γ requiere una compuerta NOT para $\overline{C}$, necesita dos compuertas OR de dos entradas y dos compuertas AND de dos entradas. Se tiene presente que los términos $\overline{A}\overline{B}\overline{D}$, $\overline{B}\overline{D}$ y BD ya estaban totalmente obtenidos desde f_α . Esta función se implementa con cinco retardos de compuerta, donde tres retardos se deben a la espera del término $\overline{A}\overline{B}\overline{D}$ y los otros dos retardos se requieren para hacer la suma.
- El total de elementos es de cuatro compuertas NOT, cinco compuertas OR y diez compuertas AND. Por tanto, $C(S) = 19$ y $D(S) = 5$, que es la profundidad de la ruta más larga. Además, si se piensa en términos de circuitos integrados, de la tabla 5.2, se puede comprobar que se requiere de un IC 7404 (NOT), dos IC 7432 (OR de 2 entradas) y tres IC 7408 (AND de 2 entradas), para un total de 6 IC.

Para la simplificación simultánea de las funciones usando el algoritmo de Q-M-P, se inicia identificando todos los implicantes primos e involucrando los prescindibles, tal como se muestra en la tabla 5.6.

Tabla 5.6: Ejemplo Quine McCluskey para implementación mínima global con tres funciones, tabla de I.P.

I's	mintérminos	1-Cubo	2-Cubo
0	$m_0000\alpha\gamma^*$	$m_0m_200 - 0\alpha\gamma\text{IP2}$ $m_0m_8 - 000\gamma\text{IP3}$	$m_4m_5m_6m_701 - \beta\text{IP1}$
1	$m_20010\alpha\beta\gamma\text{IP10}$ $m_40100\beta^*$ $m_81000\beta\gamma\text{IP11}$	$m_2m_60 - 10\beta\text{IP4}$ $m_2m_{10} - 010\alpha\beta\text{IP5}$ $m_4m_5010 - \beta^*$	
2	$m_50101\beta\gamma^*$ $m_60110\beta^*$ $m_{10}1010\alpha\beta^*$ $m_{12}1100\alpha\text{IP12}$	$m_4m_601 - 0\beta^*$ $m_8m_{10}10 - 0\beta\text{IP6}$	
3	$m_70111\alpha\beta\gamma\text{IP13}$ $m_{13}1101\gamma^*$	$m_5m_701 - 1\beta\gamma\text{IP7}$ $m_5m_{13} - 101\gamma\text{IP8}$ $m_6m_7011 - \beta^*$	
4	$m_{15}1111\alpha^*$	$m_7m_{15} - 111\alpha\text{IP9}$	

Ahora, la tabla 5.7 muestra el proceso para identificar los I.P.E. y desde la cual se encuentra el conjunto mínimo de I.P., para terminar de cubrir los minterminos aún no cubiertos, usando el algoritmo de Petrick.

Tabla 5.7: Ejemplo Quine McCluskey para implementación mínima global con tres funciones, tabla de I.P.E.

IP	Solución	f_α				f_β			f_γ		
		m_0 *	m_2 *	m_7 +	m_{10} *	m_2 *	m_4 *	m_5 *	m_2 *	m_7 +	m_8 -
IP1, $m_4m_5m_6m_701 - \beta$	IPE						X	X			
IP2, $m_0m_200 - 0\alpha\gamma$	IPE	X	X						X		
IP3, $m_0m_8 - 000\gamma$	IP -										X
IP4, $m_2m_60 - 10\beta$						X					
IP5, $m_2m_{10} - 010\alpha\beta$	IPE		X		X	X					
IP6, $m_8m_{10}10 - 0\beta$											
IP7, $m_5m_701 - 1\beta\gamma$								X		X	
IP8, $m_5m_{13} - 101\gamma$											
IP9, $m_7m_{15} - 111\alpha$				X							
IP10, $m_20010\alpha\beta\gamma$			X			X			X		
IP11, $m_81000\beta\gamma$											X
IP12, $m_{12}1100\alpha$											
IP13, $m_70111\alpha\beta\gamma$	IP +			X						X	

$$\begin{aligned}
 Petrick &= (IP9 + IP13)(IP7 + IP13)(IP3 + IP11) \\
 &= (IP9IP7 + IP9IP13 + IP7IP13 + IP13)(IP3 + IP11) \\
 &= IP3IP9IP7 + IP11IP9IP7 + IP3IP9IP13 + IP11IP9IP13 + IP3IP7IP13 \\
 &\quad + IP11IP7IP13 + IP3IP13 + IP11IP13
 \end{aligned}$$

Los resultados de esta simplificación son:

$$\begin{aligned}
 f_\alpha(A, B, C, D) &= \overline{A}\overline{B}\overline{D} + \overline{B}C\overline{D} + \overline{A}BCD \\
 f_\beta(A, B, C, D) &= \overline{A}B + \overline{B}C\overline{D} \\
 f_\gamma(A, B, C, D) &= \overline{A}BCD + \overline{B}C\overline{D} + \overline{A}B\overline{D}
 \end{aligned}$$

La implementación de esta solución requiere de los siguientes elementos, restringiendo el fan-in a dos entradas y con $C_\Omega(f_n) = \{\text{NOT, AND de 2 entradas, OR de 2 entradas}\}$:

- La función f_α requiere de tres compuertas NOT, dos compuertas OR de dos entradas y seis compuertas AND de dos entradas, donde el término $\overline{B}\overline{D}$ se reutiliza dos veces. Esta función se implementa con cinco retardos de compuerta, o cuatro capas de evaluación: tres capas para el término producto de cuatro literales y dos capas para los términos suma.

- La función f_β no requiere NOT, pues se reutilizan desde f_α ; necesita una compuerta OR de dos entradas y no requiere compuertas AND, ya que todos sus términos producto se pueden reutilizar desde f_α . Esta función se implementa con cuatro retardos de compuerta: tres retardos por cada término producto en f_α y un retardo para su suma.
- La función f_γ requiere una compuerta NOT para $\overline{C}$; necesita una compuerta OR de dos entradas, ya que los términos $\overline{A}BCD + \overline{A}\overline{B}\overline{D}$ se evalúan y suma desde f_α ; y una compuerta AND de dos entradas, teniendo presente que se reutiliza el producto $\overline{B}\overline{D}$ desde f_α . Esta función se implementa con cinco retardos de compuerta: cuatro retardos por los términos producto en f_α con su respectiva suma y un retardo para sumar el término faltante.
- El total de elementos es de cuatro compuertas NOT, cuatro compuertas OR y siete compuertas AND. Por tanto, $C(S) = 15$ y $D(S) = 5$, que es la profundidad de la ruta más larga. Además, si se piensa en términos de circuitos integrados, de la tabla 5.2, se puede comprobar que se requiere de un IC 7404 (NOT), un IC 7432 (OR de 2 entradas) y dos IC 7408 (AND de 2 entradas), para un total de 4 IC.
- Esta solución de simplificación mínima global ofrece los mejores valores de complejidad y profundidad. Incluso, desde una perspectiva de circuitos integrados, también ofrece una implementación con menos elementos.

5.7. COMPLEJIDAD, MINIMIZACIÓN Y UNIVERSALIZACIÓN

El proceso de universalización de una función también aporta a obtener soluciones de costo mínimo, junto con la minimización global de funciones. Para ello, es importante tener presente la universalización al momento de evaluar la complejidad y profundidad de una solución.

EJEMPLO

Para el último ejemplo de minimización global con tres funciones, analizar y comparar la solución mínima sin universalizar y universalizada.

La figura 5.8 muestra el gráfico acíclico dirigido de la solución mínima previa, donde:

$$\begin{aligned} f_\alpha(A, B, C, D) &= \overline{A}\overline{B}\overline{D} + \overline{B}C\overline{D} + \overline{A}BCD \\ f_\beta(A, B, C, D) &= \overline{A}B + \overline{B}C\overline{D} \\ f_\gamma(A, B, C, D) &= \overline{A}BCD + \overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{D} \end{aligned}$$

Figura 5.8: Gráfico acíclico para minimización global no universalizada

En este caso, dado que las formas son SOP, se usa universalización por NAND, lo cual conduce a expresiones de la siguiente forma, ajustadas de una vez para implementación restringida a fan-in de dos entradas.

$$f_{\alpha}(A, B, C, D) = \overline{\overline{\overline{A\overline{B}D}} \left(\overline{\overline{\overline{A}B\overline{C}D}} \right) \left(\overline{\overline{B}D\overline{C}} \right)}$$

$$f_{\beta}(A, B, C, D) = \overline{\overline{A}B} \left(\overline{\overline{\overline{B}D}C} \right)$$

$$f_{\gamma}(A, B, C, D) = \overline{\overline{\overline{A}B}D} \left(\overline{\overline{\overline{A}B\overline{C}D}} \right) \left(\overline{\overline{B}D\overline{C}} \right)$$

La figura 5.9 muestra el diagrama acíclico orientado para esta solución mínima universalizada con NAND y donde todos los vértices representan compuertas NAND.

Figura 5.9: Gráfico acíclico para minimización global universalizada

Comparando las soluciones sin universalizar y universalizada se tienen las siguientes observaciones y con $C_{\Omega}(f_n) = \{\text{NAND de 2 entradas}\}$:

- La solución no universalizada requiere de cuatro compuertas NOT, cuatro OR y siete AND, con $C(S) = 15$ y $D(S) = 5$. En términos de IC se requiere de un IC 7404 (NOT), un IC 7432 (OR de 2 entradas) y dos IC 7408 (AND de 2 entradas), para un total de 4 IC, donde se usan 15 de 18 elementos disponibles, sobrando 3 elementos (2 compuertas NOT y una compuerta AND).
- La solución universalizada requiere de 19 compuertas NAND, con lo cual $C(S) = 19$ y $D(S) = 7$. Las 2 capas adicionales son producto de aplicar de forma adecuada la ley asociativa para realizar NAND de 2 entradas. En términos de IC, se requiere de 5 IC 7400 (NAND de 2 entradas), donde se usan 19 de 20 elementos disponibles, sobrando 1 elemento (1 compuerta NAND).
- Aunque la complejidad y profundidad de la solución universalizada es mayor en comparación con la solución no universalizada, se debe observar que, al universalizar todas las compuertas, e IC, son iguales, con lo cual el costo por comprar más elementos de igual naturaleza puede jugar a favor. Además, los elementos que sobran de la implementación universalizada, obviamente se pueden usar en otra implementación universalizada. Esto último no es seguro en la implementación sin universalizar, pues sobran elementos diferentes entre sí, que no es seguro si se pueden usar en otras implementaciones.
- Una observación importante para destacar en implementaciones físicas es que los fabricantes producen todas las compuertas con base en la NAND, por lo que una compuerta como la OR está internamente construida con base en la NAND, y por consiguiente la NAND es la compuerta más rápida de todas. Esto hace que, aunque existan más capas, sea posible obtener una implementación universalizada más rápida, al ser construida con base en la compuerta madre. Valores típicos para la familia 74LSXX son: para la 74LS00 10 ns, para la 74LS32 14 ns, para la 74LS08 12 ns y para la 74LS04 10 ns.
- Se debe recordar que los valores de complejidad y profundidad de un circuito, $C_{\Omega}(f_n)$, son relativos a la base de funciones empleadas.

5.8. RIESGOS DE TIEMPO

5.8.1. Riesgo estático y riesgo dinámico

Debido a los retardos de propagación de las compuertas, se pueden presentar los siguientes dos casos, que se muestran en la figura 5.10.

- Se examina el caso de una expresión SOP de la forma $z(x_1, x_2, x_3) = x_1x_2 + \overline{x_1}x_3$, donde una variable está en dos términos y cambia de valor, ya que en uno está negada y en el otro sin negar, ver diagrama (a).
- Si los retardos de propagación (D) son iguales en todos los componentes de la red, lo cual es equivalente a que los retardos sean ideales, la salida se comporta como en el diagrama (b). En este diagrama, los cambios en las señales internas y_1 y y_2 ocurren en una sola de ellas o en ambas de forma simultánea, lo cual no altera la salida z .

- Si los retardos de propagación son tales que $G1$ es la compuerta más lenta y $G3$ (compuerta OR) la más rápida, la salida se comporta como en el diagrama (c). En este caso, los cambios

Figura 5.10: Riesgo de tiempo

en las señales internas y_1 y y_2 ocurren a tiempo distintos, lo cual ocasiona que en la salida z se observe una conmutación temporal de 1 a 0 y luego nuevamente a 1 lógico, que se denomina **riesgo estático 1 $\rightarrow$ 0 $\rightarrow$ 1**.

- Análogamente, en las redes POS de la forma $z(x_1, x_2, x_3) = (x_1 + x_2)(\overline{x_1} + x_3)$, si la compuerta AND es la más rápida, los cambios en las señales internas ocurren a tiempos distintos, lo cual ocasiona que en la salida z se observe una conmutación temporal de 0 a 1 y luego a 0 lógico, que se denomina riesgo estático 0 $\rightarrow$ 1 $\rightarrow$ 0.

Cuando un circuito lógico con riesgo estático alimenta a otro circuito lógico con riesgo estático, a la salida del segundo circuito se produce una acumulación de riesgos, denominado **riesgo dinámico**, como se enseña en la figura 5.11.

Figura 5.11: Riesgo dinámico

5.8.2. Corrección de riesgos de tiempo

Para corregir una expresión lógica que posee riesgo estático de tiempo, se debe adicionar un término a la ecuación de tal forma que no afecte su evaluación. Este término se logra involucrando las dos señales que acompañan la variable que cambia y se consigue de aplicar el teorema del consenso. La forma original con riesgo $z(x_1, x_2, x_3) = x_1x_2 + \overline{x_1}x_3$ se corrige adicionando el término de consenso $z(x_1, x_2, x_3) = x_1x_2 + \overline{x_1}x_3 + x_2x_3$.

En un mapa VK, este término de consenso implica adicionar a la respuesta minimizada los implicantes/implicados primos que conecten grupos no conectados de la solución. En Q-M-P, la solución es adicionar los términos de consenso manualmente o poner todos los I.P. en la solución.

EJEMPLO

Obtener una solución libre de riesgo estático para la siguiente función:

$$f(A, B, C, D) = \sum m(0, 5, 7, 9, 10) + \sum d(2, 8, 13, 14, 15)$$

La solución, vista sobre un mapa VK, se puede observar en la figura 5.12, donde el grupo de los minterminos m_8 y m_9 se adiciona con el fin de conectar dos agrupaciones vecinas no conectadas entre sí, lo cual genera el riesgo estático. La solución con riesgo estático es: $f(A, B, C, D) = BD + \overline{B}\overline{D} + A\overline{C}D$. La solución sin riesgo estático es: $f(A, B, C, D) = BD + \overline{B}\overline{D} + A\overline{C}D + A\overline{B}\overline{C}$, donde el término que se adiciona $A\overline{B}\overline{C}$ es el consenso de $\overline{B}\overline{D} + A\overline{C}D$, donde D es la variable que cambia.

Figura 5.12: Solución, con mapas VK, libre de riesgo estático

5.9. EJERCICIOS PROPUESTOS

1. Empleando los datos consignados en la tabla 5.1:
 - a) Determine que todas las series de la familia TTL son compatibles entre sí.
 - b) Determine que todas las series de la familia CMOS son compatibles entre sí.
 - c) Determine la compatibilidad de las series TTL como compuertas maestras y de las series CMOS como esclavas.
 - d) Determine la compatibilidad de las series CMOS como compuertas maestras y de las series TTL como esclavas.
2. Presentar las medidas de complejidad, profundidad y la base de la complejidad $C_{\Omega}(f_n)$, para la función que determina si una señal de entrada BCD natural es mayor o igual que 5.
3. Presentar las medidas de complejidad, profundidad y la base de la complejidad $C_{\Omega}(f_n)$, para la función del punto 2, pero restringiendo todas las compuertas a 2 entradas.
4. Presentar las medidas de complejidad, profundidad y la base de la complejidad $C_{\Omega}(f_n)$, para la función del punto 2, pero restringiendo todas las compuertas a 2 entradas y universalizadas con NAND. Para este punto, realice el procedimiento desde la forma SOP.

5. Presentar las medidas de complejidad, profundidad y la base de la complejidad $C_{\Omega}(f_n)$, para la función del punto 2, pero restringiendo todas las compuertas a 2 entradas y universalizadas con NOR. Para este punto, realice el procedimiento desde la forma POS.
6. Comparar las medidas de complejidad y profundidad de las soluciones de los puntos 2 a 5.
7. Presentar las medidas de complejidad, profundidad y la base de la complejidad $C_{\Omega}(f_n)$, para las dos funciones que determinan si una señal de entrada BCD natural es un número primo (f_1) o si es un número entre 3 y 6 inclusive (f_2). Realizar la comparación de complejidad para la solución que minimiza cada función de forma independiente, para la solución que minimiza de forma simultánea las dos funciones y para esta última solución universalizada con la compuerta de su preferencia.

6

CAPÍTULO
SEIS

Diseño y síntesis para aritmética combinacional

Este capítulo se centra en emplear lo presentado en capítulos previos con el fin de diseñar toda variedad de circuitos con funcionalidad aritmética. Se realiza la presentación de situaciones de diseño que se encadenan con el fin de lograr funcionalidades de mayor envergadura. Debido a que el número de variables de un problema repercute directamente en el tamaño de una tabla de verdad, se trabajan varias situaciones donde se debe proceder por síntesis, o por diseñar un problema mayor mediante partes menores funcionales y luego determinar las relaciones que gobiernan esas partes.

6.1. CIRCUITOS PARA ARITMÉTICA BINARIA

Diseño combinacional :	<i>Proceso sistemático de toma de decisiones que produce una estrategia donde recursos, técnicas y herramientas combinacionales se integran para resolver una necesidad dada.</i>
Síntesis combinacional :	<i>Procedimiento de análisis de un circuito combinacional complejo mediante la identificación y separación de su función, en componentes combinacionales básicos.</i>
Síntesis por expansión:	<i>Procedimiento de análisis y diseño de un dispositivo de mayor capacidad, empleando dispositivos de igual funcionalidad, pero de menor capacidad.</i>

6.1.1. Sumador medio - Half Adder (HA)

Un sumador medio realiza la suma binaria de dos bits x_i y y_i . Ya que la máxima suma posible requiere de dos bits, se requieren dos salidas, una para la suma (S_i) y otra para el acarreo (C_i). La representación de la operación aritmética que se implementa y la respectiva tabla de verdad se muestran en la tabla 6.1.

Tabla 6.1: Representación de la operación del sumador medio y su tabla de verdad

OPERACIÓN	TABLA DE VERDAD			
	x_i	y_i	C_i	S_i
$\begin{array}{r} x_i \\ + y_i \\ \hline C_i S_i \end{array}$	0	0	0	0
	0	1	0	1
	1	0	0	1
	1	1	1	0

La simplificación de estas dos funciones que comparten el mismo conjunto de entradas no requiere del método Q-M-P ya que, desde una inspección de la tabla de verdad, se puede observar que las dos funciones no tienen mintérminos en común, además como la función C_i solo tiene un mintérmino, este es directamente la solución. De lo anterior, el mapa VK para la salida S_i es el que se muestra en la figura 6.1 y donde no hay riesgo de tiempo.

		X_i	
		0	1
Y_i	0		1
	1	1	

Figura 6.1: Mapa VK para la suma en un sumador medio

La solución para las dos salidas de un sumador medio es:

$$C_i = x_i y_i$$

$$S_i = \bar{x}_i y_i + x_i \bar{y}_i = x_i \oplus y_i$$

El circuito lógico que implementa estas dos funciones se muestra en la figura 6.2, donde se puede constatar que para $C_\Omega(f_n) = \{\text{AND, OR, NOT de 2 entradas}\}$, $C(S) = 6$ y $D(S) = 3$. Aunque la salida C_i tiene retardo de propagación de 1 tiempo de compuerta ($t_{carry} = 1t_g$), la salida S_i tiene retardo de propagación de 3 tiempos de compuerta ($t_{add} = 3t_g$), lo cual deriva en que el circuito, o algoritmo, es tan lento como su salida más lenta, o $D(S) = 3$. Si $C_\Omega(f_n) = \{\text{XOR, AND de 2 entradas}\}$, en este caso $C(S) = 2$ y $D(S) = 1$, ya que no es necesario pasar las señales de entrada por las compuertas negadoras. Como existen diversos $C_\Omega(f_n)$ para implementar funciones, es común referir la complejidad y profundidad a $C_\Omega(f_n) = \{\text{AND, OR, NOT de 2 entradas}\}$ con el fin de comparar sistemas aritméticos.

Figura 6.2: Circuito lógico para sumador medio

6.1.2. Sumador completo - Full Adder (FA)

Un sumador completo realiza la suma binaria de tres bits, con dos bits de suma x_i y y_i más un posible acarreo previo C_{i-1} . La máxima suma posible requiere dos bits de salida, uno para la suma (S_i) y otro para el acarreo (C_i). La representación de la operación aritmética que se implementa y la respectiva tabla de verdad se muestran en la tabla 6.2.

Tabla 6.2: Representación de la operación del sumador completo y su tabla de verdad

x_i	y_i	C_{i-1}	C_i	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

La simplificación de estas dos funciones que comparten el mismo conjunto de entradas no requiere del método Q-M-P, ya que de una inspección de la tabla de verdad y visualización en mapa VK, se puede observar que las dos funciones solo tienen el mintermino m_7 en común, que no es conveniente dejar como un grupo solo en la minimización de la salida C_i . Teniendo lo anterior presente, el mapa VK para las dos salidas se muestra en la figura 6.3 y donde el posible riesgo de tiempo queda cubierto por el mismo proceso de minimización.

Figura 6.3: Mapa VK para la suma y acarreo en un sumador completo

La solución para las dos salidas de un sumador completo es:

$$\begin{aligned}
 C_i &= x_i y_i + y_i C_{i-1} + x_i C_{i-1} \\
 S_i &= \bar{x}_i \bar{y}_i C_{i-1} + \bar{x}_i y_i \bar{C}_{i-1} + x_i \bar{y}_i \bar{C}_{i-1} + x_i y_i C_{i-1} \\
 &= C_{i-1}(\bar{x}_i \bar{y}_i + x_i y_i) + \bar{C}_{i-1}(\bar{x}_i y_i + x_i \bar{y}_i) \\
 &= C_{i-1} \oplus (x_i \oplus y_i)
 \end{aligned}$$

Si se parte de no simplificar a C_i , estas funciones se pueden expresar como a continuación, donde el término $x_i \oplus y_i$ se puede reutilizar entre ambas salidas:

$$\begin{aligned}
 C_i &= \bar{x}_i y_i C_{i-1} + x_i \bar{y}_i C_{i-1} + x_i y_i \bar{C}_{i-1} + x_i y_i C_{i-1} \\
 &= C_{i-1}(\bar{x}_i y_i + x_i \bar{y}_i) + x_i y_i (\bar{C}_{i-1} + C_{i-1}) \\
 &= C_{i-1}(x_i \oplus y_i) + x_i y_i
 \end{aligned}$$

La complejidad y profundidad para estas soluciones varían según la base de operaciones booleanas, como se puede observar en la tabla 6.3 y donde se tiene presente la reutilización de términos ya evaluados.

Tabla 6.3: Complejidad y profundidad para un sumador completo, según base de operaciones booleanas

$C_{\Omega}(f_n)$	Solución			$C(S)$	$D(S)$
{AND, OR, NOT}	C_i	$x_i y_i + y_i C_{i-1} + x_i C_{i-1}$	$t_{carry} = 2t_g$	12	3
	S_i	$\bar{x}_i \bar{y}_i C_{i-1} + \bar{x}_i y_i \bar{C}_{i-1} + x_i \bar{y}_i \bar{C}_{i-1} + x_i y_i C_{i-1}$	$t_{add} = 3t_g$		
{AND, OR, NOT 2 entradas}	C_i	$x_i y_i + y_i C_{i-1} + x_i C_{i-1}$	$t_{carry} = 2t_g$	15	5
	S_i	$\bar{x}_i \bar{y}_i C_{i-1} + \bar{x}_i y_i \bar{C}_{i-1} + x_i \bar{y}_i \bar{C}_{i-1} + x_i y_i C_{i-1}$	$t_{add} = 5t_g$		
{AND, OR, XOR}	C_i	$C_{i-1}(x_i \oplus y_i) + x_i y_i$	$t_{carry} = 2t_g$	5	3
	S_i	$C_{i-1} \oplus (x_i \oplus y_i)$	$t_{add} = 3t_g$		

A continuación, en la figura 6.4, se muestra el circuito lógico de menor complejidad y profundidad visto para un sumador completo.

Figura 6.4: Circuito lógico para sumador completo

6.1.3. Sumador en cascada - Ripple Carry Adder (RCA)

El sumador en cascada realiza la suma binaria sobre palabras de n -bits, mediante la propagación del acarreo a través de todas las etapas intermedias de suma. En esta operación, la suma en la etapa menos significativa se realiza mediante un HA, ya que no hay acarreo previo, pero la suma en las demás etapas se realiza mediante un FA con el fin de propagar los acarreos entre etapas. La suma final tiene un bit adicional, producto de un posible acarreo final. La representación de la operación aritmética que se implementa es la siguiente:

$$\begin{array}{r}
 x_{n-1} x_{n-2} \dots x_2 \ x_1 \ x_0 \\
 + \ y_{n-1} y_{n-2} \dots y_2 \ y_1 \ y_0 \\
 \hline
 Z_n \ Z_{n-1} Z_{n-2} \dots Z_2 \ Z_1 \ Z_0
 \end{array}$$

Si cada una de las palabras a sumar posee 4 bits, el diseño posee una tabla de verdad de 8 entradas (4 por cada palabra), arrojando 256 combinaciones. El tamaño de esta tabla es el que dirige, inicialmente, las opciones de diseño hacia un enfoque por síntesis modular, usando HA y FA, como se observa en la figura 6.5.

Figura 6.5: Sumador en cascada

Como el diseño de este sumador en cascada se basa en un enfoque modular, la complejidad está directamente relacionada con la cantidad de bits en cada palabra. Ya que un HA posee $C(S) = 6$ y $D(S) = 3$ con $t_{carry} = 1t_g$ y $t_{add} = 3t_g$, y un FA se puede tomar con $C(S) = 5$ y $D(S) = 3$ con $t_{carry} = 2t_g$ y $t_{add} = 3t_g$, entonces, la complejidad del RCA es igual a la complejidad de un HA (por la etapa menos significativa) más $n - 1$ veces la complejidad de un FA (por las demás etapas), arrojando que $C(S) = 6 + (n - 1) * 5 = 5n + 1$. Igualmente, para determinar la profundidad del RCA, se debe tener presente que la segunda etapa debe esperar por el acarreo del HA (donde $t_{carry} = 1t_g$), las demás etapas (menos la más significativa) esperan por acarreos provenientes de un FA (donde $t_{carry} = 2t_g$) y en la etapa más significativa se debe esperar por el resultado de la suma, que es mas lenta que el acarreo, con $t_{add} = 3t_g$; finalmente $D(S) = 1t_g + (n - 2) * 2t_g + 3t_g = 2n * t_g$. De lo anterior, en un RCA $t_{add} = 2n * t_g$ y $t_{carry} = (2n - 1)t_g$. Si la suma se hace sobre dos palabras de 8 bits, entonces se tiene que $C(S) = 5n + 1 = 41$ compuertas y $D(S) = 2n * t_g = 16t_g$.

6.1.4. Sumador/Restador en cascada

Un sumador/restador en cascada realiza, según la entrada de selección S/R , las operaciones de suma o resta sobre palabras de n -bits, mediante la propagación del acarreo a través de todas las etapas intermedias. En esta operación, la suma se realiza como en un RCA y la resta se hace empleando complemento a 2 para el número negativo. Según la operación, el sumando 1 o el minuendo, permanecen inalterados; en cambio, en una suma el sumando 2 no se altera, pero el sustraendo se pasa a complemento a 1 y se le suma +1 para tener su complemento a 2, como se indica a continuación:

PARA SUMAR						PARA RESTAR							
Sumando 1	$x_{n-1}x_{n-2} \dots x_2 x_1 x_0$					Minuendo	$x_{n-1}x_{n-2} \dots x_2 x_1 x_0$						
Sumando 2	+	$y_{n-1}y_{n-2} \dots y_2 y_1 y_0$					Sustraendo	+	$\overline{y_{n-1}}\overline{y_{n-2}} \dots \overline{y_2} \overline{y_1} \overline{y_0}$				
Resultado		Z_n	Z_{n-1}	Z_{n-2}	Z_2	Z_1	Z_0						
		Z_n	Z_{n-1}	Z_{n-2}	Z_2	Z_1	Z_0						
		 						
		Z_n											

Para implementar las dos operaciones se toma un RCA y se tiene presente las siguientes observaciones:

- Ya sea una suma o resta, el número positivo (sumando 1 o minuendo) permanece sin cambio.
- Se reemplaza el HA de la etapa menos significativa por un FA. La entrada adicional de este FA menos significativo se emplea para sumar el +1 que se requiere al hacer restas y, se tiene presente que, para hacer sumas esta entrada debe ser cero. Por ello, convenientemente se toma la entrada de selección de operación como $S/R = 0$ para sumar y $S/R = 1$ para restar. Para describir esta entrada de selección de forma más expedita, se acostumbra simplemente escribir $\overline{S/R}$, con lo cual se entiende que la negación en la suma indica que esta opera se asocia al cero lógico.

- Para el sustraendo, o sumando 2 según la operación deseada, se debe diseñar un circuito adicional que permita pasar la palabra sin alteraciones cuando lo que se desea es sumar y que invierta cada bit cuando lo que se desea es restar. La tabla 6.4 muestra la tabla de verdad para la función de control sobre el sustraendo, donde y_i^* es la salida con el bit ajustado según la operación deseada. Claramente, $y_i^* = y_i \oplus \bar{S}/R$.

Tabla 6.4: Tabla de verdad para control del sustraendo en sumador/restador

SELECCIÓN $\bar{S}/R$			
y_i	$\bar{S}/R$	y_i^*	
0	0	0	Para sumar: $y_i^* = y_i$ Para restar: $y_i^* = \bar{y}_i$
0	1	1	
1	0	1	
1	1	0	

La implementación de este circuito se muestra en la figura 6.6, donde se puede observar que la entrada de selección de operación se conecta al acarreo de entrada del FA menos significativo y a cada una de las compuertas XOR, para controlar el sustraendo. Ya que este sumador/restador se construye con base en un RCA, su complejidad es igual a la de un RCA más n compuertas XOR requeridas para cada uno de los FA, por lo que $C(S) = 5n + 1 + n = 6n + 1$. Para determinar la profundidad del sumador/restador, se debe tener presente que la segunda etapa debe esperar por la evaluación de la primera XOR más el acarreo del primer FA ($1t_g + 2t_g$); las demás etapas, excepto la más significativa, esperan por acarreos provenientes de un FA (donde $t_{carry} = 2t_g$) y en la etapa más significativa se debe esperar por el resultado de la suma, donde $t_{add} = 3t_g$; finalmente $D(S) = 3t_g + (n - 2) * 2t_g + 3t_g = (2n + 2)t_g$. De lo anterior, en un sumador/restador $t_{add} = (2n + 2)t_g$ y $t_{carry} = (2n + 1)t_g$. Si este diseño opera sobre dos palabras de 8 bits, entonces se tiene que $C(S) = 6n + 1 = 49$ y $D(S) = (2n + 2)t_g = 18t_g$.

Figura 6.6: Circuito lógico de un sumador/restador en cascada

6.1.5. Verificador de desbordamiento

El desbordamiento ocurre cuando la suma de dos números positivos da un número negativo, o cuando la suma de dos negativos da uno positivo, lo cual conlleva que el resultado no entre en la misma representación de los operandos. Una forma de verificar este desbordamiento es comprobar que si se suman dos números de igual signo, el resultado debe ser del mismo signo. La tabla 6.5 muestra la tabla de verdad respectiva, donde x_{n-1} y y_{n-1} son los bits de signo (bits más significativos) de los operandos y C_{n-2} es el bit de acarreo de la etapa previa, los cuales sumados arrojan el último bit de suma (Z_{n-1}) y el bit de acarreo final (C_{n-1}). La señal V es la advertencia de desbordamiento, entregando 1 lógico cuando ello ocurre. En esta misma tabla se puede verificar que el desbordamiento también ocurre cuando las señales C_{n-1} y C_{n-2} son diferentes. Esto hace que el verificador se pueda implementar a partir de dos expresiones lógicas, una con base en los bits de signo y otra con base en los acarreos, así:

$$V = \overline{x_{n-1}} \cdot \overline{y_{n-1}} \cdot Z_{n-1} + x_{n-1} \cdot y_{n-1} \cdot \overline{Z_{n-1}}$$

$$V = C_{n-1} \oplus C_{n-2}$$

Tabla 6.5: Tabla de verdad para verificador de desbordamiento

x_{n-1}	y_{n-1}	C_{n-2}	C_{n-1}	Z_{n-1}	V
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

La expresión que se basa en los bits de signo siempre es posible implementarla, pero la expresión que se basa en los acarreos requiere el acceso a la señal C_{n-2} , la cual, en un IC que implementa este sumador/restador, no siempre es accesible para el usuario.

La profundidad y complejidad de la primera implementación son $D(S) = 3t_g$ y $C(S) = 6$ si $C_\Omega(f_n) = \{\text{AND, OR, NOT}\}$, o $D(S) = 4t_g$ y $C(S) = 8$ si $C_\Omega(f_n) = \{\text{AND, OR, NOT de 2 entradas}\}$; para la segunda implementación $D(S) = 1t_g$ y $C(S) = 1$ con $C_\Omega(f_n) = \{\text{XOR}\}$.

Debido que este verificador opera en conjunto con un sumador/restador, se debe adicionar la complejidad respectiva, según la opción implementada. Además, la profundidad de la opción que se basa en los bits de signo sin restricción de fan-in no se suma, ya que se ejecuta en paralelo con el FA más significativo; la profundidad de esta misma opción con restricción de fan-in suma un retardo; y la profundidad de la opción que se basa en acarreos no suma, ya que se ejecuta luego del acarreo final, que es una capa más rápido que el bit de suma respectivo. La figura 6.7 muestra la implementación de ambas funciones.

Figura 6.7: Circuito lógico para verificador de desbordamiento

6.2. COMPARADOR BINARIO

Fuera de poder realizar operaciones aritméticas binarias, las operaciones de comparación son muy comunes, como medio para toma de decisiones y de control del flujo de un algoritmo. Un comparador binario básico determina relaciones de magnitud entre dos palabras binarias, con 2 bits cada una (A_1A_0 y B_1B_0), tal como se indica en la tabla 6.6. En esta tabla de verdad, se puede verificar que la salida de igualdad se puede obtener fácilmente desde las salidas mayor que o menor que, así: $f_{A=B} = \overline{f_{A>B}} + \overline{f_{A<B}}$. Sin embargo, es común la implementación de esta salida de forma directa, con el fin de reducir la profundidad del circuito o algoritmo.

Tabla 6.6: Tabla de verdad para comparador de dos palabras de 2 bits cada una

A_1	A_0	B_1	B_0	$f_{A>B}$	$f_{A=B}$	$f_{A<B}$
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
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	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Para simplificar las funciones de salida no es necesario emplear el método de Q-M-P, toda vez que ninguna de las tres salidas posee mintérminos (o maxtérminos) comunes con las demás, es decir, todas las salidas son mutuamente excluyentes entre sí. La simplificación entonces se puede realizar recurriendo a un mapa VK por cada una de las salidas, como se muestra en la figura 6.8, y donde el riesgo de tiempo queda mitigado directamente por la simplificación, ya que todos los grupos vecinos están conectados entre sí.

Figura 6.8: Mapa VK para las tres funciones de un comparador de dos palabras de 2 bits cada una

El resultado de la simplificación de las tres funciones de comparación se muestra a continuación:

$$\begin{aligned}
 f_{A>B} &= A_1 \cdot \overline{B_1} + A_0 \cdot \overline{B_1} \cdot \overline{B_0} + A_1 \cdot A_0 \cdot \overline{B_0} \\
 f_{A=B} &= \overline{A_1} \cdot \overline{A_0} \cdot \overline{B_1} \cdot \overline{B_0} + \overline{A_1} \cdot A_0 \cdot \overline{B_1} \cdot B_0 + A_1 \cdot \overline{A_0} \cdot B_1 \cdot \overline{B_0} + A_1 \cdot A_0 \cdot B_1 \cdot B_0 \\
 f_{A<B} &= \overline{A_1} \cdot B_1 + \overline{A_1} \cdot \overline{A_0} \cdot B_0 + \overline{A_0} \cdot B_1 \cdot B_0
 \end{aligned}$$

La profundidad y complejidad de este comparador son $D(S) = 3t_g$ y $C(S) = 17$ si $C_\Omega(f_n) = \{\text{AND, OR, NOT}\}$, o $D(S) = 5t_g$ y $C(S) = 27$ si $C_\Omega(f_n) = \{\text{AND, OR, NOT de 2 entradas}\}$.

Adicionalmente, existen otras relaciones que se pueden obtener a partir de las anteriores, como son:

$$\begin{aligned}
 f_{A \neq B} &= \overline{f_{A=B}} = f_{A<B} + f_{A>B} = f_{A<>B} \\
 f_{A \leq B} &= f_{A<B} + f_{A=B} \\
 f_{A \geq B} &= f_{A>B} + f_{A=B}
 \end{aligned}$$

6.3. CIRCUITOS SOBRE SISTEMAS NO PONDERADOS

6.3.1. Sumador BCD natural

Al sumar dos palabras en BCD natural en un RCA de 4 bits, la respuesta es máximo el decimal $9+9=18$, o 19 si existe acarreo previo. Se debe recordar que, al hacer la suma en una RCA es posible obtener como resultado una palabra que no es BCD natural, lo cual se soluciona sumando el decimal 6 (o el binario 0110) si la salida es mayor o igual a 10. La tabla 6.7 muestra la tabla de verdad que indica si se debe corregir la salida de la suma ($f_{Z \geq 10}$) en un RCA, y donde Z_3, Z_2, Z_1, Z_0 son el resultado de la suma y C el acarreo; además esta tabla posee prescindibles, o don't cares, desde la posición 20 hacia adelante.

Tabla 6.7: Tabla de verdad para corrección de número no BCD natural

#	C	Z_3	Z_2	Z_1	Z_0	$f_{Z \geq 10}$	#	C	Z_3	Z_2	Z_1	Z_0	$f_{Z \geq 10}$
0	0	0	0	0	0	0	10	0	1	0	1	0	1
1	0	0	0	0	1	0	11	0	1	0	1	1	1
2	0	0	0	1	0	0	12	0	1	1	0	0	1
3	0	0	0	1	1	0	13	0	1	1	0	1	1
4	0	0	1	0	0	0	14	0	1	1	1	0	1
5	0	0	1	0	1	0	15	0	1	1	1	1	1
6	0	0	1	1	0	0	16	1	0	0	0	0	1
7	0	0	1	1	1	0	17	1	0	0	0	1	1
8	0	1	0	0	0	0	18	1	0	0	1	0	1
9	0	1	0	0	1	0	19	1	0	0	1	1	1

La simplificación para la única función de salida $f_{Z \geq 10}$ se realiza sobre un mapa VK de 5 variables, el cual se muestra en la figura 6.9.

Figura 6.9: Mapa VK para función de corrección de número no BCD natural

La simplificación de la función de corrección de un número no BCD natural es $f_{Z \geq 10} = C + Z_3 \cdot Z_1 + Z_3 \cdot Z_2$ y no existe riesgo de tiempo, toda vez que los grupos vecinos están conectados entre sí. La profundidad y complejidad de esta función son $D(S) = 2t_g$ y $C(S) = 3$ si $C_{\Omega}(f_n) = \{\text{AND, OR}\}$, o $D(S) = 3t_g$ y $C(S) = 4$ si $C_{\Omega}(f_n) = \{\text{AND, OR, de 2 entradas}\}$. Ahora, para diseñar el circuito que realiza la suma completa sobre dos palabras de 4 bits en BCD natural, se emplea la señal $f_{Z \geq 10}$ para compensar las sumas que no son BCD, así:

- Si la suma de las dos palabras en BCD natural arroja que $f_{Z \geq 10} = 1$, entonces la respuesta de la suma es incorrecta (no es BCD natural) y hay necesidad de hacer corrección. Para ello, la respuesta incorrecta se debe volver a sumar con el binario 0110 (decimal 6).
- Si la suma de las dos palabras en BCD natural arroja que $f_{Z \geq 10} = 0$, entonces la respuesta de la suma es correcta (es BCD natural) y no hay necesidad de hacer corrección. Sin embargo, con el fin de unificar un diseño, esto equivale a volver a sumar la respuesta correcta con el binario 0000 (decimal 0).
- De las dos anotaciones previas, se puede concluir que siempre se vuelve a sumar el binario $0f_{Z \geq 10}f_{Z \geq 10}0$, o 0110 para cuando la respuesta es incorrecta y 0000 para cuando la respuesta es correcta. El diseño completo, se muestra en la figura 6.10, donde las salidas S_3, S_2, S_1, S_0 son el resultado de la suma luego de esta corrección final.

Figura 6.10: Circuito para sumar dos palabras de 4 bits en BCD natural

Partiendo del hecho que en un RCA para palabras de 4 bits $C(S) = 5n + 1 = 21$ compuertas y $D(S) = 2n * t_g = 8t_g$, y que en la figura 6.10 se puede observar como la salida del primer RCA se requiere para evaluar la función $f_{Z \geq 10}$ y esta última para poder hacer la compensación en el segundo RCA, entonces la complejidad y profundidad totales del circuito son: $C(S) = 2 * 21 + 3 = 45$ compuertas y $D(S) = (2 * 8 + 2)t_g = 18t_g$ con $C_{\Omega}(f_n) = \{\text{AND}, \text{OR}\}$.

6.3.2. Restador BCD natural

Para el diseño de un circuito restador de dos palabras de 4 bits en BCD natural ($A - B$), se hace uso de la representación de complemento a base 10 para la palabra B . De lo anterior, la operación a realizar es: $A + (10 - B)$, y este resultado debe ser corregido en caso de no ser un número BCD, es decir se suma el decimal 6 (o binario 0110) si no se obtiene un BCD natural. Finalmente, si no hubo necesidad de corregir el BCD, es por que la resta arroja un resultado negativo y se debe volver a tomar el complemento a la base 10; pero si ya previamente se corrigió el BCD, es por que la resta arroja un resultado positivo y no hay necesidad de volver a tomar este complemento. Esto se ilustra con los siguientes dos casos:

- La resta $5-3=2$, se hace como $5+(10-3) = 5+7 = 12$. Ahora, el acarreo (1) indica que el resultado (2) es positivo, por lo que el signo debe ser 0 (se niega el acarreo) y al resultado se le suma cero ($2+0=2$).
- La resta $3-7=-4$, se hace como $3+(10-7) = 3+3 = 06$. Ahora, el acarreo (0) indica que el resultado (6) es negativo y está en complemento a la base, por lo que el signo debe ser 1 (se niega el acarreo) y al resultado se le debe tomar su complemento a la base, o $10-6=4$.

En definitiva, para hacer la operación ($A - B$) en BCD se deben seguir los siguientes pasos que, además, se muestran en el circuito de la figura 6.11:

- Realizar complemento a 10 del sustraendo: $10 - B$.
- Sumar el resultado anterior con A : $\text{Resta parcial} = A + (10 - B)$.
- Si en la operación anterior el acarreo es 1, el resultado es positivo y se debe hacer $0 + \text{Resta parcial}$; pero si el acarreo es 0, el resultado es negativo y se debe hacer $10 - \text{Resta parcial}$. El signo final es el inverso del acarreo en Resta parcial .

6.3.3. Codificador - Decodificador de paridad

El control de errores por códigos de paridad, en los procesos de comunicación, permite detectar hasta el cambio (error) en un bit, tal como se explicó previamente en la sección 1.6.

Para el desarrollo de un sistema de control de errores para enviar una palabra A de n bits, codificada en una paridad dada, se tiene en cuenta los siguientes pasos, y los cuales se pueden observar en la figura 6.12:

- Con base en el peso de la palabra $w(A)$ se agrega el bit de paridad según la paridad deseada, para obtener $C(A)$ que es la palabra codificada. Este proceso se hace en dos etapas:
 - Se cuenta el número de unos de la palabra A , o $w(A)$.
 - Se agrega el bit de paridad.
- Luego, la palabra codificada $C(A)$ viaja por el medio.
- Al destino llega la palabra $C'(A)$, a la cual se le debe determinar su integridad.
- En el destino, el proceso de verificación inicia con decodificar lo recibido, para lo cual se evalúa el peso de la palabra recibida, contando el número de unos lógicos $w(D)$.
- Con base en $w(D)$ y la paridad definida inicialmente, se decide si A llegó bien.

Figura 6.12: Codificador - Decodificador de paridad

Figura 6.11: Circuito para restador BCD natural

A continuación, se desarrolla cada uno de los módulos contenidos en la figura 6.12, suponiendo que la palabra A es de 3 bits. Inicialmente se diseña un contador de unos para $n + 1$ bits (es decir, 4 bits), de tal forma que sirva para determinar tanto $w(A)$ como $w(D)$, donde llega un bit adicional (el de paridad). La tabla de verdad respectiva se muestra en la tabla 6.8, donde X_3 a X_0 son los 4 bits de la palabra y w_2 , w_1 y w_0 son el formato binario del número de unos contenidos en A .

Tabla 6.8: Tabla de verdad para contador de unos

X_3	X_2	X_1	X_0	w_2	w_1	w_0	X_3	X_2	X_1	X_0	w_2	w_1	w_0
0	0	0	0	0	0	0	1	0	0	0	0	0	1
0	0	0	1	0	0	1	1	0	0	1	0	1	0
0	0	1	0	0	0	1	1	0	1	0	0	1	0
0	0	1	1	0	1	0	1	0	1	1	0	1	1
0	1	0	0	0	0	1	1	1	0	0	0	1	0
0	1	0	1	0	1	0	1	1	0	1	0	1	1
0	1	1	0	0	1	0	1	1	1	0	0	1	1
0	1	1	1	0	1	1	1	1	1	1	0	0	0

La solución implica simplificar tres funciones que comparten las mismas entradas, por lo que la solución se debe realizar mediante el algoritmo de Q-M-P. Sin embargo, en la figura 6.13, se muestra la solución usando mapas VK y donde se omite el respectivo mapa para la solución trivial $w_2 = X_3 \cdot X_2 \cdot X_1 \cdot X_0$. Se debe observar que, en el mapa VK para la salida w_1 , existe riesgo de tiempo y fue necesario adicionar agrupaciones que unen grupos vecinos no conectados previamente.

Figura 6.13: Mapas VK para contador de unos

La solución para las tres salidas del contador de unos es:

$$w_2 = X_3 \cdot X_2 \cdot X_1 \cdot X_0$$

$$w_1 = X_3 \cdot X_2 \cdot \overline{X_1} + X_3 \cdot X_2 \cdot \overline{X_0} + \overline{X_3} \cdot X_1 \cdot X_0 + \overline{X_2} \cdot X_1 \cdot X_0 + X_2 \cdot \overline{X_1} \cdot X_0 + X_3 \cdot \overline{X_1} \cdot X_0 + \\ + X_2 \cdot X_1 \cdot \overline{X_0} + X_3 \cdot X_1 \cdot \overline{X_0} + \overline{X_3} \cdot X_2 \cdot X_0 + \overline{X_3} \cdot X_2 \cdot X_1 + X_3 \cdot \overline{X_2} \cdot X_0 + X_3 \cdot \overline{X_2} \cdot X_1$$

$$w_0 = \overline{X_3} \cdot \overline{X_2} \cdot \overline{X_1} \cdot X_0 + \overline{X_3} \cdot \overline{X_2} \cdot X_1 \cdot \overline{X_0} + \overline{X_3} \cdot X_2 \cdot \overline{X_1} \cdot \overline{X_0} + \overline{X_3} \cdot X_2 \cdot X_1 \cdot X_0 + \\ + X_3 \cdot \overline{X_2} \cdot \overline{X_1} \cdot \overline{X_0} + X_3 \cdot \overline{X_2} \cdot X_1 \cdot X_0 + X_3 \cdot X_2 \cdot \overline{X_1} \cdot X_0 + X_3 \cdot X_2 \cdot X_1 \cdot \overline{X_0}$$

Ahora, se diseña el módulo agregador de paridad, donde el bit de paridad a agregar (p) se evalúa según el peso de la palabra (w_2, w_1, w_0) y la paridad deseada de envío P/I ($\overline{p\bar{a}r}$ /impar), tal como se indica en la tabla 6.9.

Tabla 6.9: Tabla de verdad para agregador de paridad

P/I	w_2	w_1	w_0	p	P/I	w_2	w_1	w_0	p
0	0	0	0	0	1	0	0	0	1
0	0	0	1	1	1	0	0	1	0
0	0	1	0	0	1	0	1	0	1
0	0	1	1	1	1	0	1	1	0
0	1	0	0	0	1	1	0	0	1
0	1	0	1	1	1	1	0	1	0
0	1	1	0	0	1	1	1	0	1
0	1	1	1	1	1	1	1	1	0

La simplificación para la salida p se muestra en la figura 6.14, y donde no existe riesgo de tiempo.

		P/Iw_2			
		00	01	11	10
w_1w_0	00			1	1
	01	1	1		
	11	1	1		
	10			1	1

Figura 6.14: Mapa VK para agregador de paridad

La solución queda como se muestra a continuación:

$$p = \overline{P/I} \cdot w_0 + P/I \cdot \overline{w_0} = P/I \oplus w_0$$

Finalmente, se muestra el diseño del módulo verificador de recepción. La integridad MAL? (0 bien, 1 mal) se evalúa con base en el peso de la palabra recibida (w_2, w_1, w_0) y la paridad en que se hizo el envío (P/I). La tabla 6.10 muestra la tabla de verdad respectiva y donde se puede comprobar que dicha tabla y su salida MAL? son exactamente iguales al agregador de paridad, o:

$$MAL? = p = \overline{P/I} \cdot w_0 + P/I \cdot \overline{w_0} = P/I \oplus w_0$$

Tabla 6.10: Tabla de verdad del verificador de recepción

P/I	w_2	w_1	w_0	MAL?	P/I	w_2	w_1	w_0	MAL?
0	0	0	0	0	1	0	0	0	1
0	0	0	1	1	1	0	0	1	0
0	0	1	0	0	1	0	1	0	1
0	0	1	1	1	1	0	1	1	0
0	1	0	0	0	1	1	0	0	1
0	1	0	1	1	1	1	0	1	0
0	1	1	0	0	1	1	1	0	1
0	1	1	1	1	1	1	1	1	0

Una observación importante es que el diseño se realizó tal que, el circuito en el dispositivo que emite la información es igual al circuito en el dispositivo que recibe, lo cual hace que el mismo diseño permita emitir y recibir.

6.4. SÍNTESIS POR EXPANSIÓN

La síntesis por expansión es el procedimiento de diseño de un dispositivo de mayor capacidad, empleado dispositivos de igual funcionalidad pero de menor capacidad. Durante este proceso, primero se diseña un dispositivo básico que cumple la funcionalidad y luego se hace la síntesis de las partes constitutivas de un dispositivo de mayor capacidad, que emplea como fundamento el de menor capacidad, y se determinan los elementos adicionales requeridos para conectar esas partes.

EJEMPLO

Realizar síntesis por expansión para el diseño de un comparador para palabras de 4 bits, usando el comparador para palabras de 2 bits de la sección 6.2.

La justificación de proceder mediante síntesis por expansión radica en que la tabla de verdad de un comparador para dos palabras de 4 bits implica la existencia de $2^8 = 256$ combinaciones, lo cual resulta en tablas imprácticas. En contraste, es una mejor concepción emplear dos comparadores de 2 bits para su implementación, donde un comparador se encarga de determinar las relaciones entre los bits menos significativos (etapa LSD) y el otro comparador de las relaciones entre los bits más significativos (etapa MSD), tal como se muestra en la tabla 6.11.

Tabla 6.11: Relaciones entre etapas para la síntesis de un comparador de 4 bits

ETAPA MSD		‡	ETAPA LSD	
A_3	A_2	‡	A_1	A_0
B_3	B_2	‡	B_1	B_0
$f_{A=B}$	IGUALES	‡	IGUALES	$f_{A=B} = f_{A=B}^{MSD} * f_{A=B}^{LSD}$
$f_{A>B}$	$f_{A>B}$	‡	NO IMPORTA	$f_{A>B} = f_{A>B}^{MSD} + f_{A=B}^{MSD} * f_{A>B}^{LSD}$
	$f_{A=B}$	‡	$f_{A>B}$	$+ f_{A=B}^{MSD} * f_{A>B}^{LSD}$
$f_{A<B}$	$f_{A<B}$	‡	NO IMPORTA	$f_{A<B} = f_{A<B}^{MSD} + f_{A=B}^{MSD} * f_{A<B}^{LSD}$
	$f_{A=B}$	‡	$f_{A<B}$	$+ f_{A=B}^{MSD} * f_{A<B}^{LSD}$
$t_{comp} = [3 + 2]t_g = 5t_g$				

De la tabla 6.11 se pueden observar que:

- Dos palabras de 4 bits son iguales si y sólo si los 2 bits de la etapa menos significativa son iguales y los 2 bits de la etapa más significativa son iguales.
- En dos palabras de 4 bits, la palabra $A > B$ si y sólo si:
 - La etapa MSD de la palabra A es la mayor, o
 - La parte MSD de ambas palabras es igual y en la etapa LSD la palabra $A > B$
- En dos palabras de 4 bits, la palabra $A < B$ si y sólo si:
 - La etapa MSD de la palabra A es la menor, o
 - La parte MSD de ambas palabras es igual y en la etapa LSD la palabra $A < B$

La implementación de estas operaciones entre etapas para un comparador de 4 bits implica que, la profundidad de la relación de igualdad en contraposición con un comparador de 2 bits crece en un retardo. Igualmente, la profundidad de las salidas $f_{A>B}$ o $f_{A<B}$ crecen en dos retardos. Por lo tanto, para un comparador de 4 bits implementado mediante síntesis por expansión, la profundidad es $D(S) = 5t_g$ si $C_\Omega(f_n) = \{\text{AND, OR, NOT}\}$, o $D(S) = 7t_g$ si $C_\Omega(f_n) = \{\text{AND, OR, NOT de 2 entradas}\}$. La complejidad aumenta en 5, con lo cual $C(S) = 22$ si $C_\Omega(f_n) = \{\text{AND, OR, NOT}\}$, o $C(S) = 32$ si $C_\Omega(f_n) = \{\text{AND, OR, NOT de 2 entradas}\}$.

6.5. EJERCICIOS PROPUESTOS

1. Hacer la solución de la tabla 6.13 usando el algoritmo Q-M-P y comparar su complejidad y profundidad con la solución obtenida por mapas VK en la sección 6.3.3.
2. Implementar el circuito comparador para palabras de 4 que se diseñó en la sección 6.4.
3. Diseñar por síntesis por expansión un circuito comparador genérico para palabras de n bits, con base en el comparador para palabras de 4 bits. Realizar los análisis de profundidad y complejidad para su diseño.
4. Diseñar un circuito y presentar su respectivo algoritmo, para un sumador/restador de dos palabras en BCD natural.
5. Diseñar un circuito y presentar su respectivo algoritmo, para un multiplicador tradicional para dos palabras binarias de 4 bits. Un bit adicional por palabra y respuesta se emplea para evaluar el signo. Emplear FA, HA o RCA como bases del diseño. Hacer los análisis de complejidad y profundidad respectivos.
6. Diseñar un circuito y presentar su respectivo algoritmo, para pasar palabras binarias de 4 bits a palabras en código Gray. Hacer los análisis de complejidad y profundidad respectivos.
7. Diseñar un circuito y presentar su respectivo algoritmo, para pasar palabras en Gray de 4 bits a palabras en binario. Hacer los análisis de complejidad y profundidad respectivos.
8. Diseñar un circuito y presentar su respectivo algoritmo, para pasar palabras binarias de 4 bits a palabras en BCD + 3. Hacer los análisis de complejidad y profundidad respectivos.

9. Diseñar un circuito y presentar su respectivo algoritmo, para un codificador Hamming 7,4.
Hacer los análisis de complejidad y profundidad respectivos.
10. Diseñar un circuito y presentar su respectivo algoritmo, para un decodificador Hamming 7,4.
Hacer los análisis de complejidad y profundidad respectivos.

7

CAPÍTULO SIETE

Tratamiento y enrutamiento de datos

Este capítulo final cubre el diseño y síntesis de sistemas lógicos combinacionales que incorporan el tratamiento de datos, o procesos de conversión y/o representación de los mismos; así como el enrutamiento, o procesos para el control de flujo de información a través de diseños. Estos tratamientos y enrutamientos de datos, permiten soluciones que cubren varias funcionalidades y son ajustables en su configuración. El alcance de los sistemas que se presentan en este libro cubre exclusivamente los sistemas combinacionales. La adición de capacidad de memoria, pila o diseño en cinta hace parte de los denominados sistemas secuenciales y de otras máquinas formales, que serán parte de subsecuentes entregas.

7.1. CODIFICADORES Y DECODIFICADORES

Codificar/ Decodificar:	<i>Codificar es el proceso por el cual la información se convierte a un formato adecuado para su procesamiento. Decodificar es el proceso de invertir la codificación, usualmente con fines de desplegar información al usuario.</i>
Multiplexar/ Demultiplexar :	<i>Multiplexar es el proceso de tomar varias fuentes de información y conducirlas por un único medio. Demultiplexar es el proceso de tomar una única fuente de información y dirigirla a varios destinos.</i>
Interfaz hombre/máquina:	<i>Medio que permite el intercambio de información entre el hombre y la máquina, con el fin de permitir al hombre introducir información y visualizar datos generados por la máquina.</i>

Para el tratamiento de la información, y su control de flujo, en circuitos o algoritmos que involucran los sistemas digitales, se emplean códigos binarios con el fin de realizar, entre otras acciones: tomar información desde el entorno para ser entendida por la máquina, cambiar el formato de datos para su procesamiento, llevar los datos desde formatos binarios hacia formatos para despliegue visual amigable con el hombre y para permitir la interfaz hombre/máquina. Estas funciones las realizan los decodificadores y codificadores que se exponen a continuación.

7.1.1.1. Decodificadores

Un decodificador es un dispositivo combinacional que toma un código binario de entrada de N bits y activa una, o varias, de M líneas de salida en representación del código de entrada. En estos dispositivos, el número de salidas es un entero menor o igual a 2^N .

Los decodificadores más comunes activan solo una de las M líneas de salida para cada una de las posibles combinaciones de las N líneas de entrada, con exactamente $M = 2^N$. En lógica de nivel alto, para una combinación de entrada solo se activa una línea de salida en 1 lógico y las demás líneas permanecen en 0 lógico. Pero, también es posible hacer el diseño para lógica de nivel bajo, o donde solo se activa una salida en 0 lógico y las demás permanecen en 1 lógico. Para este último tipo de activación, cada salida se muestra negada, indicando el comportamiento inverso a lógica de nivel alto. La figura 7.1 muestra el diagrama general de un decodificador de N entradas y M salidas para lógica de nivel alto.

Figura 7.1: Diagrama general de un decodificador

Para el diseño de un decodificador, por ejemplo de 3 entradas y $2^3 = 8$ salidas, se emplea la tabla 7.1, donde se puede observar que para cada código binario de entrada, solo se activa una línea de salida.

A un decodificador donde $M = 2^N$ se le denomina decodificador $N \times M$. Para el ejemplo, se tiene un decodificador 3x8 o también decodificador binario a octal, lo cual especifica que existen 8 líneas de salida y que el código binario de entrada es de 3 bits. Las ecuaciones de las salidas de este decodificador se muestran en la tabla 7.2, donde es claro que basta con implementar el mintermino respectivo para cada salida, siendo mutuamente excluyente y haciendo innecesario una simplificación global.

Tabla 7.1: Tabla de verdad para un decodificador 3x8

I_2	I_1	I_0	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Tabla 7.2: Tabla con ecuaciones de salida de un decodificador 3x8

Salida	Ecuación	Salida	Ecuación
O_0	$\bar{I}_2 \bar{I}_1 \bar{I}_0$	O_4	$I_2 \bar{I}_1 \bar{I}_0$
O_1	$\bar{I}_2 \bar{I}_1 I_0$	O_5	$I_2 \bar{I}_1 I_0$
O_2	$\bar{I}_2 I_1 \bar{I}_0$	O_6	$I_2 I_1 \bar{I}_0$
O_3	$\bar{I}_2 I_1 I_0$	O_7	$I_2 I_1 I_0$

El circuito respectivo para un decodificador 3x8 se muestra en la figura 7.2.

Figura 7.2: Circuito lógico de un decodificador 3x8

La complejidad de un decodificador $N \times M$ siempre será igual a $C(S) = N + M$, ya que se requiere de una negación por cada entrada y una compuerta AND por cada salida, siempre y cuando no se fijen restricciones en el número de entradas de las compuertas, o $C_{\Omega}(f_n) = \{\text{AND}, \text{NOT}\}$. La profundidad, igualmente sin restricciones en el fan-in, es de $D(S) = 2t_g$, o una capa por las negaciones y otra capa por las AND.

También existen decodificadores que no emplean todos los 2^N códigos posibles de entrada. Este es el caso de un decodificador con entrada BCD y salida decimal, ya que posee un código de entrada de 4 bits, del cual sólo se usan 10 de las 16 posibles codificaciones (recordar que un código BCD natural codifica los números decimales del 0 al 9). En estos dispositivos puede suceder que al tratar de decodificar un código no válido de entrada que se toma como prescindible, la respuesta ponga las salidas en un código igual a uno válido. Para dar solución a lo anterior, se acostumbra diseñar el decodificar tal que, si se presenta un código no válido a la entrada, todas las salidas permanezcan inactivas. La tabla 7.3 muestra la tabla de verdad para un decodificador BCD a decimal donde los códigos no válidos ponen todas las salidas inactivas.

Tabla 7.3: Tabla de verdad para un decodificador BCD a decimal

I_3	I_2	I_1	I_0	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9
0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1
1	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0

Las ecuaciones de las salidas para un decodificador BCD a decimal se muestran en la tabla 7.4, donde nuevamente basta con implementar el mintermino respectivo para cada salida, y siendo innecesario una simplificación global.

Tabla 7.4: Tabla con ecuaciones de salida para un decodificador 3x8

Salida	Ecuación	Salida	Ecuación
O_0	$I_3 \bar{I}_2 \bar{I}_1 \bar{I}_0$	O_5	$\bar{I}_3 I_2 I_1 \bar{I}_0$
O_1	$I_3 \bar{I}_2 \bar{I}_1 I_0$	O_6	$\bar{I}_3 I_2 I_1 I_0$
O_2	$I_3 \bar{I}_2 I_1 \bar{I}_0$	O_7	$\bar{I}_3 I_2 I_1 I_0$
O_3	$I_3 \bar{I}_2 I_1 I_0$	O_8	$I_3 \bar{I}_2 \bar{I}_1 \bar{I}_0$
O_4	$I_3 \bar{I}_2 \bar{I}_1 I_0$	O_9	$I_3 \bar{I}_2 \bar{I}_1 I_0$

La figura 7.3 enseña el diagrama lógico para un decodificador BCD a decimal.

Figura 7.3: Circuito lógico de un decodificador BCD a decimal

La complejidad y profundidad de un decodificador BCD a decimal es igual a la de un decodificador $N \times M$, o $C(S) = N + M = 4 + 10 = 14$ compuertas y $D(S) = 2t_g$, con $C_{\Omega}(f_n) = \{\text{AND}, \text{NOT}\}$.

En ciertas circunstancias se puede desear que un decodificador no opere, es decir que ponga todas sus salidas en 0 lógico (para activación en nivel alto) o en 1 lógico (para activación en nivel bajo). Para dar solución a ello, la mayoría de decodificadores incluyen una línea de habilitación que permite, o no, la operación del decodificador. La tabla 7.5 muestra la tabla de verdad para un decodificador 2x4 con salidas activas en nivel alto o nivel bajo. La entrada E (del inglés *Enable*) es la habilitación y cuando vale 1 lógico el decodificador no opera. Una forma simple de obtener un decodificador con entrada de habilitación es adicionar la entrada E a cada una de las compuertas AND que implementan las ecuaciones de salida con base en mintérminos. El cambio de activación por nivel se puede obtener simplemente negando las salidas.

Tabla 7.5: Tabla de verdad para decodificador 2x4 con habilitación

E	I_1	I_0	Salidas activas nivel bajo				Salidas activas nivel alto			
			O_0	O_1	O_2	O_3	O_0	O_1	O_2	O_3
1	d	d	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0
0	0	1	1	0	1	1	0	1	0	0
0	1	0	1	1	0	1	0	0	1	0
0	1	1	1	1	1	0	0	0	0	1

La figura 7.4 muestra el circuito lógico para un decodificador 2x4 con habilitación, que se obtiene de implementar cada uno de los minterminos de las salidas.

Figura 7.4: Decodificador 2x4 con habilitación

Si $C_{\Omega}(f_n) = \{\text{AND}, \text{NOT}\}$, la complejidad de un decodificador con habilitación suma una compuerta a la complejidad de un decodificador $N \times M$, o $C(S) = N + M + 1$ debido a la NOT de la habilitación; pero la profundidad sigue siendo $D(S) = 2t_g$, ya que la nueva compuerta NOT se puede evaluar en paralelo con las negaciones de las entradas.

Con base en obtener una tabla de verdad que relacione la activación de cada una de las M líneas de salida, según los códigos de entrada, se pueden diseñar decodificadores para aplicaciones específicas. Además, pueden existir decodificadores que activan más de una línea de salida, por código de entrada.

EJEMPLOS

1. Diseñar un decodificador BCD a 7 segmentos, el cual toma una entrada BCD natural y produce como salida la activación de las líneas necesarias para dibujar el decimal respectivo con base en un despliegue de 7 segmentos lumínicos, como se observa en la figura 7.5. Por ejemplo, para mostrar el número decimal 6, se deben activar los segmentos c, d, e, f y g (poner a 1 lógico).

Figura 7.5: Despliegue de dígitos decimales en 7 segmentos lumínicos

En este decodificador, cada salida se activa para más de una combinación de entrada. Es así que, el segmento “ d ” se activa para los dígitos 0, 2, 3, 5, 6 y 8, como se muestra en la tabla 7.6, que es la tabla de verdad que representa la activación de segmentos según la entrada BCD. En esta tabla, los códigos no BCD se han tomado como prescindibles.

Tabla 7.6: Tabla de verdad de activación para un decodificador BCD a 7 segmentos

I_3	I_2	I_1	I_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	0	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	d	d	d	d	d	d	d
1	0	1	1	d	d	d	d	d	d	d
1	1	0	0	d	d	d	d	d	d	d
1	1	0	1	d	d	d	d	d	d	d
1	1	1	0	d	d	d	d	d	d	d
1	1	1	1	d	d	d	d	d	d	d

La solución libre de riesgo, empleando mapas VK, para las 8 salidas se muestra a continuación en la tabla 7.7; sin embargo, esta solución amerita implementar una minimización global, que no se enseña en esta oportunidad.

Tabla 7.7: Solución por mapas VK para decodificador BCD a 7 segmentos

Salida	Ecuación	Salida	Ecuación
a	$I_1 + I_3 + I_2 I_1 + I_2 \bar{I}_0$	e	$\bar{I}_2 I_0 I_1 \bar{I}_0$
b	$I_2 + I_1 \bar{I}_0 + I_1 I_0$	f	$I_3 + I_1 \bar{I}_0 + I_2 \bar{I}_1 + I_2 \bar{I}_0$
c	$I_3 + I_2 + \bar{I}_1 + I_0$	g	$I_3 + I_2 \bar{I}_1 + I_1 \bar{I}_0 + I_2 I_1$
d	$\bar{I}_2 \bar{I}_0 + I_2 I_1 + I_2 I_1 I_0 + I_1 \bar{I}_0$	—	—

Existen variantes de este decodificador que, en lugar de tomar los prescindibles, asignan patrones específicos a los códigos no BCD.

2. Realizar la síntesis por expansión de un decodificador 4x16 empleando 2 decodificadores 3x8, con entrada de habilitación.

La tabla 7.8 es la tabla de verdad para las salidas de un decodificador 4x16, donde se puede observar que el bit más significativo del código de entrada (I_3) se puede emplear como la línea de habilitación que controla dos decodificadores 3x8 sobre los tres bits menos significativos. Así, si $I_3 = 0$ opera el decodificador menos significativo, y si $I_3 = 1$ opera el decodificador más significativo, como lo resalta la letra en negrita de la misma tabla.

El circuito resultante se muestra en la figura 7.6.

Tabla 7.8: Tabla de verdad para un decodificador 4x16

I_3	I_2	I_1	I_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9	O_{10}	O_{11}	O_{12}	O_{13}	O_{14}	O_{15}	O_{16}
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figura 7.6: Decodificador 4×16 como expansión de dos decodificadores 3×8

Los decodificadores $N \times M$, con $M = 2^N$, también pueden implementar funciones lógicas de forma directa. Lo anterior ocurre ya que, para un código de entrada, un decodificador activa solo una salida, por lo que los códigos de entrada se pueden asimilar como los renglones de una tabla de verdad y las salidas como la activación de posibles minterminos o maxtérminos, que solo basta agrupar para obtener la función en su forma estándar canónica, SOP o POS. De lo anterior, se dice que los decodificadores generan minterminos o maxtérminos.

EJEMPLO

Diseñar un contador de unos para palabras de tres bits, empleando un decodificador como generador de los mintérminos y los maxtérminos de las funciones de salida.

La tabla de verdad para un contador de unos en palabras de tres bits (I_2, I_1, I_0) requiere entregar los binarios desde el número $(0)_2$ hasta el $(11)_2$, por lo que necesita de dos salidas binarias (w_1, w_0) , tal como se indica en la tabla 7.9.

Tabla 7.9: Tabla de verdad para contador de unos en palabras de 3 bits

I_2	I_1	I_0	w_1	w_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

A partir de esta tabla de verdad se puede deducir claramente que las ecuaciones de salida, en forma canónica, son:

$$w_1(I_2, I_1, I_0) = \sum m(3, 5, 6, 7) = \prod M(0, 1, 2, 4)$$

$$w_0(I_2, I_1, I_0) = \sum m(1, 2, 4, 7) = \prod M(0, 3, 5, 6)$$

De las anteriores expresiones, esta implementación se puede realizar ya sea mediante las formas SOP o las formas POS, como se muestra en la figura 7.7.

Una observación importante sobre la implementación de funciones con decodificadores radica en que la profundidad de estas implementaciones, cuando no existen restricciones de fan-in, siempre es de $D(S) = 3t_g$, donde dos tiempos de retardo se deben al propio decodificador y un tiempo adicional a la compuerta que agrupa los mintérminos o maxtérminos. Esto hace que esta forma de implementar funciones tenga la misma profundidad que una implementación directa con compuertas, que siempre tiene tres capas, pero reduciendo el número de elementos requeridos, si se toma el decodificador como uno solo. Entonces, la complejidad es $C(S) = 2$, uno por el decodificador y otro por la compuerta de agrupación.

7.1.2. Codificadores

Un codificador toma M líneas de entrada, donde solo una está activa a la vez y entrega un código binario de N bits de salida que representa la línea activa, con $M \leq 2^N$. Esta operación de un codificador es la contraria a la que realiza un decodificador y, cuando ocurre de la forma descrita, se

Figura 7.7: Contador de unos para palabras de 3 bits, empleando decodificador como generador de minterminos o maxtérminos

denomina codificador sin prioridad. La Figura 7.8, muestra el diagrama general de un codificador con M entradas y N salidas, o codificador $M \times N$, con lógica de nivel alto.

Figura 7.8: Diagrama general de un codificador sin prioridad

Un codificador sin prioridad, por ejemplo de $M = 8$ entradas y $N = 3$ salidas ($8 = 2^3$), entrega el código binario de tres bits que representa la entrada activa, tal como se muestra en la tabla 7.10. En esta tabla de verdad, se puede observar que, solo hay una entrada activa a la vez.

Tabla 7.10: Tabla de verdad para un codificador 8x3 sin prioridad

I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	O_2	O_1	O_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

A un codificador 3x8 sin prioridad también se le denomina codificador octal, con lo cual se especifica que existen 8 líneas de entrada sobre las que se produce un código de salida de 3 bits. Las ecuaciones de las salidas de este codificador son las siguientes, y donde no se ha empleado simplificación global con el fin de minimizar el tiempo de respuesta:

$$O_2 = I_4 + I_5 + I_6 + I_7$$

$$O_1 = I_2 + I_3 + I_6 + I_7$$

$$O_0 = I_1 + I_3 + I_5 + I_7$$

El circuito respectivo para un codificador 8x3 se muestra en la figura 7.9.

Figura 7.9: Circuito lógico de un codificador 8x3 sin prioridad

La complejidad de un codificador $M \times N$ siempre será igual a $C(S) = N$, ya que se requiere de una compuerta OR por cada salida, siempre y cuando no se fijen restricciones en el número de entradas de las compuertas, o $C_{\Omega}(f_n) = \{\text{OR}\}$. La profundidad, igualmente sin restricciones en el fan-in, es de $D(S) = 1t_g$, o una capa por las compuertas OR.

En los codificadores sin prioridad solo puede haber una de las entradas activa a la vez. Cuando puede haber más de una entrada activa a la vez, se recurre a un decodificador con prioridad, donde la codificación de salida corresponde a la línea de entrada más significativa activa. Además, si se adiciona una entrada de habilitación (E), con el fin de distinguir claramente los estados de operación, se adicionan dos salidas: GS (del inglés *Group signal*) que está activa si el codificador cumple su función y EO (del inglés *Enable Output*) que está activa si todas las señales de entrada están inactivas. La tabla 7.11 muestra el comportamiento de verdad de un decodificador 8x3 con prioridad, donde se distinguen los siguientes tres estados:

- $GS = 0$ y $EO = 0$: el codificador está inhabilitado.
- $GS = 0$ y $EO = 1$: el codificador está habilitado, sin entradas activas.
- $GS = 1$ y $EO = 0$: el codificador está habilitado, con entradas activas.

Tabla 7.11: Tabla de verdad para un codificador 8x3 con prioridad

E	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	O_2	O_1	O_0	GS	EO
1	d	d	d	d	d	d	d	d	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0	0	0	1	0
0	d	1	0	0	0	0	0	0	0	0	1	1	0
0	d	d	1	0	0	0	0	0	0	1	0	1	0
0	d	d	d	1	0	0	0	0	0	1	1	1	0
0	d	d	d	d	1	0	0	0	1	0	0	1	0
0	d	d	d	d	d	1	0	0	1	0	1	1	0
0	d	d	d	d	d	d	1	0	1	1	0	1	0
0	d	d	d	d	d	d	d	1	1	1	1	1	0

A partir de la tabla de verdad, se puede deducir que las ecuaciones de salida son las siguientes y donde no se ha empleado simplificación global con el fin de minimizar el tiempo de respuesta:

$$O_2 = I_4 + I_5 + I_6 + I_7$$

$$O_1 = I_2 + I_3 + I_6 + I_7$$

$$O_0 = I_1 + I_3 + I_5 + I_7$$

$$GS = \overline{E}(I_0 + I_1 + I_2 + I_3 + I_4 + I_5 + I_6 + I_7)$$

$$EO = \overline{E} + I_0 + I_1 + I_2 + I_3 + I_4 + I_5 + I_6 + I_7$$

La complejidad de un codificador con prioridad siempre será igual a $C(S) = N+5$, con $C_Q(f_n) = \{\text{AND}, \text{OR}, \text{NOT}\}$. Las cinco compuertas adicionales, sobre una implementación sin prioridad, se requieren para entregar las salidas GS y EO . La profundidad, sin restricciones en el fan-in, es de $D(S) = 2t_g$, donde GS es la salida más lenta.

Con base en una tabla de verdad que relacione el código en las N líneas de salida, según la activación en las M líneas de entrada, se pueden diseñar codificadores para aplicaciones específicas.

EJEMPLOS

1. Diseñar un codificador Decimal a BCD sin prioridad.

Un codificador con entrada decimal, posee 10 líneas de entrada ($I_0, \dots, I_9$), donde cada una representa a un dígito decimal, y 4 salidas que codifican el BCD natural (O_3 a O_0). Solo se admite una entrada activa a la vez. La tabla 7.12 muestra la tabla de verdad respectiva.

Tabla 7.12: Tabla de verdad para un codificador Decimal a BCD sin prioridad

I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	O_3	O_2	O_1	O_0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

A partir de la tabla de verdad se puede deducir que las ecuaciones de salida son las siguientes y donde no se ha empleado simplificación global con el fin de minimizar el tiempo de respuesta:

$$O_3 = I_8 + I_9$$

$$O_2 = I_4 + I_5 + I_6 + I_7$$

$$O_1 = I_2 + I_3 + I_6 + I_7$$

$$O_0 = I_1 + I_3 + I_5 + I_7 + I_9$$

De las anteriores expresiones, se obtiene el circuito lógico que se muestra en la figura 7.10.

Tabla 7.13: Tabla de verdad para un codificador Decimal a BCD con prioridad

E	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	O_3	O_2	O_1	O_0	GS	EO
1	d	d	d	d	d	d	d	d	d	d	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	d	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0
0	d	d	1	0	0	0	0	0	0	0	0	0	1	0	1	0
0	d	d	d	1	0	0	0	0	0	0	0	0	1	1	1	0
0	d	d	d	d	1	0	0	0	0	0	0	1	0	0	1	0
0	d	d	d	d	d	1	0	0	0	0	0	1	1	0	1	0
0	d	d	d	d	d	d	1	0	0	0	0	1	1	1	1	0
0	d	d	d	d	d	d	d	1	0	0	1	0	0	0	1	0
0	d	d	d	d	d	d	d	d	1	0	1	0	0	1	1	0

Figura 7.10: Circuito de un codificador Decimal a BCD sin prioridad

2. Diseñar un codificador Decimal a BCD con prioridad.

Es igual a un codificador con entrada decimal sin prioridad, con 10 líneas de entrada ($I_0, \dots, I_9$) que representan los dígitos decimales, 4 salidas que codifican el BCD natural (O_3 a O_0), pero se admite más de una entrada activa a la vez y la salida codifica el BCD que corresponde a la entrada más significativa activa. La tabla 7.13 muestra la tabla de verdad respectiva, a partir de la cual se puede deducir que las ecuaciones de salida y donde no se ha empleado simplificación global con el fin de minimizar el tiempo de respuesta:

$$O_3 = I_8 + I_9$$

$$O_2 = I_4 + I_5 + I_6 + I_7$$

$$O_1 = I_2 + I_3 + I_6 + I_7$$

$$O_0 = I_1 + I_3 + I_5 + I_7 + I_9$$

$$GS = \overline{E}(I_0 + I_1 + I_2 + I_3 + I_4 + I_5 + I_6 + I_7 + I_8 + I_9)$$

$$EO = \overline{E + I_0 + I_1 + I_2 + I_3 + I_4 + I_5 + I_6 + I_7 + I_8 + I_9}$$

7.2. MULTIPLEXORES Y DEMULTIPLEXORES

Para el control del flujo de datos y para el direccionamiento de los mismos, desde una fuente hacia varios destinos, o desde varias fuentes hacia un destino, se emplean los multiplexores y demultiplexores, los cuales se exponen a continuación.

7.2.1. Multiplexores

Un multiplexor, o también un selector de datos, es un diseño de circuito lógico, o de algoritmo, que implementa una especie de conmutador que direcciona una de entre varias entradas de datos ($I_{N-1}, \dots, I_0$) hacia un único destino (Z). El enrutamiento de la entrada hacia la salida, en todo instante de tiempo, está controlado por unas entradas de selección ($S_{M-1}, \dots, S_0$), o dirección, que reciben un código binario en representación de la entrada que se deja pasar. En general, se requiere que $2^M \geq N$ para que el código de selección pueda direccionar todas las entradas. La Figura 7.11 muestra el diagrama general de un multiplexor, el cual se nota como multiplexor N a 1.

Figura 7.11: Diagrama general de un multiplexor N a 1

Para diseñar un multiplexor 4 a 1 ($N = 4$ entradas enrutadas hacia una sola salida), se requiere de dos entradas de selección ($M = 2$), ya que $2^2 = 4$. El problema básico con este diseño radica en que requiere 6 entradas, lo cual produce una tabla de verdad de 64 combinaciones y hace que para acometer este diseño se deba recurrir a síntesis modular. En este sentido, la tabla 7.14 indica la operación de la salida en función de los valores de las líneas de selección, sin poner explícitamente los valores de las entradas. La salida es igual al valor de la entrada que representa el código de selección, por lo que el diseño se puede basar en habilitar la entrada que iguala el código de selección y en inhibir las demás. Para ello, se requieren 4 compuertas AND controladas por algún dispositivo que las habilite o inhiba. Ya que solo puede estar habilitada una entrada a la vez, esta operación coincide exactamente con la función que implementa un decodificador 2x4, como se muestra en la figura 7.12. Se puede concluir que, el diseño de un multiplexor N a 1 incluye un decodificador $M \times N$ y las salidas de este decodificador podrían estar disponibles para obtener en un mismo sistema las dos funcionalidades; sin embargo, en los IC esto no es común, toda vez que adicionar puertos de salida incrementa el tamaño del dispositivo.

Tabla 7.14: Operación de la salida en un multiplexor 4 a 1

Selección		Salida
S_1	S_0	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Figura 7.12: Circuito lógico de un multiplexor 4 a 1

La complejidad de un multiplexor N a 1 es igual a $C(S) = 2N + M + 1$, ya que requiere de $N + M$ compuertas por el decodificador interno y $N + 1$ para el control de inhibición de las entradas hacia la salida, bajo un escenario de $C_{\Omega}(f_n) = \{\text{AND, OR, Decodificador}\}$. La profundidad, igualmente sin restricciones en el fan-in, es de $D(S) = 4t_g$, o dos capas por la profundidad del decodificador y dos capas por la inhibición de entradas hacia la salida.

La técnica de expansión para multiplexores se puede emplear tanto para obtener un multiplexor de mayor capacidad, como para realizar multiplexación simultánea sobre varias palabras de entrada.

EJEMPLOS

1. Diseñar, por expansión, un multiplexor 16 a 1 con base en multiplexores 4 a 1.

La tabla 7.15 indica la operación de la salida en función de los valores de las líneas de selección. En esta tabla, se ha realizado una división en 4 grupos de filas. Los bits menos significativos (S_1, S_0) tienen los mismos cuatro valores sin importar el grupo y los bits más significativos (S_3, S_2) difieren de un grupo a otro, pero valen lo mismo al interior de cada grupo.

Tabla 7.15: Operación de la salida en un multiplexor 16 a 1

		Selección				Salida
		S_3	S_2	S_1	S_0	
Grupo 0		0	0	0	0	I_0
		0	0	0	1	I_1
		0	0	1	0	I_2
		0	0	1	1	I_3
Grupo 1		0	1	0	0	I_4
		0	1	0	1	I_5
		0	1	1	0	I_6
		0	1	1	1	I_7
Grupo 2		1	0	0	0	I_8
		1	0	0	1	I_9
		1	0	1	0	I_{10}
		1	0	1	1	I_{11}
Grupo 3		1	1	0	0	I_{12}
		1	1	0	1	I_{13}
		1	1	1	0	I_{14}
		1	1	1	1	I_{15}

De esta partición se puede concluir que, para realizar una multiplexación basta con hacer una primera capa de multiplexores 4 a 1 para los bits menos significativos y una segunda capa de multiplexación de las 4 salidas anteriores hacia una única salida usando otro multiplexor 4 a 1, tal como se muestra en la figura 7.13.

Figura 7.13: Multiplexor 16 a 1 con base en expansión de multiplexores 4 a 1

Esta solución, comparada con un multiplexor 4 a 1, multiplica por 5 la complejidad y dobla la profundidad.

2. Diseña un multiplexor que permita seleccionar entre dos palabras de 4 bits.

La entrada a este multiplexor la conforman dos palabras de 4 bits ($A_3A_2A_1A_0$ y $B_3B_2B_1B_0$) y la salida es una palabra de 4 bits ($Z_3Z_2Z_1Z_0$). La selección opera sobre las dos palabras de entrada dejando pasar una de ellas (S_0), por lo que el diseño se basa en seleccionar entre los bits de igual peso de las dos palabras y, para lo cual, se emplean cuatro multiplexores 2 a 1 (uno por cada par de bits de igual peso), tal como se muestra en la figura 7.14. El dispositivo resultante se conoce como un cuádruple multiplexor 2 a 1.

Este proceso de múltiple multiplexación no altera la profundidad del multiplexor empleado, toda vez que la multiplexación ocurre simultáneamente. La complejidad simplemente se incrementa en el número de multiplexores empleados.

Los multiplexores N a 1 también pueden implementar funciones lógicas de forma directa. Lo anterior ocurre ya que, para un código en la línea de selección, un multiplexor deja pasar una de las N líneas de entrada a su única salida, por lo que los códigos de selección de entrada se pueden asimilar como las filas de una tabla de verdad y las líneas de entrada como los valores de verdad de cada

Figura 7.14: Cuádruple multiplexor 2 a 1

combinación, o cada posible mintermino o maxtérmino. De lo anterior, se dice que los multiplexores generan tablas de verdad y por tanto, las funciones lógicas se deben expresar en forma canónica. Una ventaja adicional del uso de multiplexores, para implementar funciones, es que se puede usar multiplexores de tamaño menor al demandado por las variables de la tabla de verdad, para lo cual algunas variables se evalúan al interior de la forma canónica respectiva.

EJEMPLOS

1. Diseñar un contador de unos para palabras de tres bits, empleando multiplexores 8 a 1 como generadores de las tablas de verdad de las funciones de salida.

Como ya se ha visto, la tabla de verdad para un contador de unos en palabras de tres bits (I_2, I_1, I_0) requiere entregar los binarios desde el número $(0)_2$ hasta el $(11)_2$, por lo que necesita de dos salidas binarias (w_1, w_0), tal como se indica en la tabla 7.9.

Las ecuaciones de salida, en forma canónica, son:

$$w_1(I_2, I_1, I_0) = \sum m(3, 5, 6, 7) = \prod M(0, 1, 2, 4)$$

$$w_0(I_2, I_1, I_0) = \sum m(1, 2, 4, 7) = \prod M(0, 3, 5, 6)$$

Las anteriores expresiones se pueden implementar empleando multiplexores 8 a 1, (8 líneas de entrada, una por cada combinación en la tabla de verdad), tal como se muestra en la figura 7.15 y donde, explícitamente, las entradas de datos a cada multiplexor son los minterminos o maxtérminos de cada una de las dos funciones de salida y las líneas de selección son las variables de entrada en la tabla de verdad.

Figura 7.15: Contador de unos implementado con multiplexores 8 a 1

2. Diseñar el contador de unos anterior, empleando multiplexores 4 a 1 y 2 a 1 como generadores de las tablas de verdad de las dos funciones de salida (w_1 y w_0).

Las ecuaciones de salida, en forma canónica, son:

$$w_1(I_2, I_1, I_0) = \sum m(3, 5, 6, 7) = \overline{I_2}I_1I_0 + I_2\overline{I_1}I_0 + I_2I_1\overline{I_0} + I_2I_1I_0$$

$$w_0(I_2, I_1, I_0) = \sum m(1, 2, 4, 7) = \overline{I_2}\overline{I_1}I_0 + \overline{I_2}I_1\overline{I_0} + I_2\overline{I_1}\overline{I_0} + I_2I_1I_0$$

Para obtener las salidas empleando un multiplexor 4 a 1, se debe tomar una de las tres variables para ser evaluada en la expresión SOP y dejar las otras dos como parte de la tabla de verdad. Para este ejemplo, se toma la variable I_0 , tal como se indica en la tabla 7.16. En general, al reducir una variable en la tabla de verdad para ser evaluada en la expresión, las entradas al multiplexor solo tendrán como valor 1, 0, la variable, la variable negada. Esto permite reducir la complejidad del multiplexor a usar (ya que es un multiplexor con menos entradas y líneas de selección), pero se gana una capa adicional de retardo por las posibles compuertas negadoras, o se incrementa en una capa la profundidad, lo que se puede observar en la figura 7.16.

Tabla 7.16: Tabla de verdad para las salidas de un contador de unos, con multiplexor 4 a 1

I_2	I_1	$w_1(I_2, I_1, I_0) = \overline{I_2}I_1I_0 + I_2\overline{I_1}I_0 + I_2I_1\overline{I_0} + I_2I_1I_0$	$w_0(I_2, I_1, I_0) = \overline{I_2}\overline{I_1}I_0 + \overline{I_2}I_1\overline{I_0} + I_2\overline{I_1}\overline{I_0} + I_2I_1I_0$
0	0	$w_1(I_2, I_1, I_0) = 0 + 0 + 0 + 0 = 0$	$w_0(I_2, I_1, I_0) = I_0 + 0 + 0 + 0 = I_0$
0	1	$w_1(I_2, I_1, I_0) = I_0 + 0 + 0 + 0 = I_0$	$w_0(I_2, I_1, I_0) = 0 + I_0 + 0 + 0 = I_0$
1	0	$w_1(I_2, I_1, I_0) = 0 + I_0 + 0 + 0 = I_0$	$w_0(I_2, I_1, I_0) = 0 + 0 + \overline{I_0} + 0 = \overline{I_0}$
1	1	$w_1(I_2, I_1, I_0) = 0 + 0 + I_0 + I_0 = 1$	$w_0(I_2, I_1, I_0) = 0 + 0 + 0 + I_0 = I_0$

Figura 7.16: Contador de unos implementado con multiplexores 4 a 1

Para obtener las salidas empleando un multiplexor 2 a 1, se sigue un procedimiento igual al anterior, pero en esta ocasión se deben tomar dos de las tres variables para ser evaluadas en la expresión SOP y dejar la otra como parte de la tabla de verdad. Para este ejemplo, se toman las variables I_0 e I_1 , tal como se indica en la tabla 7.17. En general, al reducir dos variables en la tabla de verdad para ser evaluadas en la expresión lógica, las entradas al multiplexor son una función lógica de las dos variables reducidas. Esto disminuye la complejidad de multiplexor a usar, pero se obtienen las capas adicionales de retardo requeridas por la expresión de mayor profundidad en las entradas de datos al multiplexor. El circuito lógico respectivo se muestra en la figura 7.17.

Tabla 7.17: Tabla de verdad para las salidas de un contador de unos, con multiplexor 2 a 1

I_2	$w_1(I_2, I_1, I_0) = \overline{I_2}I_1I_0 + I_2\overline{I_1}I_0 + I_2I_1\overline{I_0} + I_2I_1I_0$	$w_0(I_2, I_1, I_0) = \overline{I_2}\overline{I_1}I_0 + \overline{I_2}I_1\overline{I_0} + I_2\overline{I_1}\overline{I_0} + I_2I_1I_0$
0	$w_1(I_2, I_1, I_0) = I_1I_0 + 0 + 0 + 0 = I_1I_0$	$w_0(I_2, I_1, I_0) = \overline{I_1}I_0 + I_1\overline{I_0} + 0 + 0 = I_0 \oplus I_1$
1	$w_1(I_2, I_1, I_0) = 0 + \overline{I_1}I_0 + I_1\overline{I_0} + I_1I_0 = I_1 + I_0$	$w_0(I_2, I_1, I_0) = 0 + 0 + \overline{I_1}\overline{I_0} + I_1I_0 = I_0 \odot I_1$

Figura 7.17: Contador de unos implementado con multiplexores 2 a 1

7.2.2. Demultiplexores

Un demultiplexor es un diseño de circuito lógico, o de algoritmo, que implementa una especie de distribuidor, que toma una entrada de datos (I) y la enruta hacia una de varias salidas ($Z_{N-1}, \dots, Z_0$). El enrutamiento de la entrada hacia las salidas, en todo instante de tiempo, está controlado por unas entradas de selección ($S_{M-1}, \dots, S_0$), o dirección, que reciben un código binario en representación de la salida a la que se desea conectar el dato de entrada. En general, se requiere que $2^M \geq N$ para que el código de selección pueda enrutar hacia todas las salidas disponibles. La Figura 7.18 muestra el diagrama general de un demultiplexor, el cual normalmente se nota como demultiplexor 1 a N .

Para diseñar un demultiplexor 1 a 4 ($N = 4$ salidas hacia las que puede ir el dato de entrada), se requiere de dos entradas de selección ($M = 2$), ya que $2^2 = 4$. La tabla 7.18 enseña la tabla de verdad con la operación de las salidas en función de los valores de las líneas de selección y el dato de entrada.

Tabla 7.18: Tabla de verdad para las salidas en un demultiplexor 1 a 4

Selección		Dato	Salidas			
S_1	S_0	I	Z_0	Z_1	Z_2	Z_3
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	0
1	1	1	0	0	0	1

La tabla 7.18 también se puede reescribir para indicar la operación de las salidas en función del dato de entrada, sin poner explícitamente sus valores, como se muestra en la tabla 7.19.

Tabla 7.19: Operación de las salidas en un demultiplexor 1 a 4

Selección		Salidas			
S_1	S_0	Z_0	Z_1	Z_2	Z_3
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I

De cualquiera de las dos tablas anteriores se pueden deducir las ecuaciones de salida, como sigue:

$$Z_0(S_1, S_0, I) = \overline{S_1} \overline{S_0} I$$

$$Z_1(S_1, S_0, I) = \overline{S_1} S_0 I$$

$$Z_2(S_1, S_0, I) = S_1 \overline{S_0} I$$

$$Z_3(S_1, S_0, I) = S_1 S_0 I$$

Se puede observar que cada una de las salidas se obtiene de hacer la AND lógica entre el mintermino que se genera con las líneas de selección (S_1, S_0) y el dato de entrada. Ya que un decodificador es un generador de minterminos, se puede realizar un diseño semejante al de un multiplexor, tal como se observa en la figura 7.19.

Se puede concluir que, el diseño de un demultiplexor 1 a N incluye un decodificador $M \times N$ y las salidas de este decodificador están siempre disponibles en las mismas salidas del demultiplexor si se hace $I = 1$. Además, la complejidad de un demultiplexor 1 a N es $C(S) = 2N + M$, ya que se requiere $N + M$ compuertas por el decodificador y N compuertas AND de salida. La profundidad es $D(S) = 3t_g$, o dos capas por el decodificador y una capa por las compuertas AND. Si se toma

Figura 7.19: Circuito lógico de un demultiplexor 1 a 4

$C_{\Omega}(f_n) = \{\text{AND}, \text{Decodificador}\}$, la complejidad se reduce a $C(S) = N + 1$ (N compuertas AND y el decodificador) y la profundidad a $D(S) = 2t_g$, o una capa por el decodificador y otra por las AND.

7.3. APLICACIÓN: UNIDAD LÓGICO ARITMÉTICA

Una unidad básica de procesamiento para computadora debe realizar un conjunto de operaciones lógicas y aritméticas básicas, a partir de las cuales se puedan realizar operaciones de mayor envergadura. Por ejemplo, si se diseña una unidad que realiza las tres operaciones lógicas básicas (AND, OR y NOT), es suficiente para realizar la evaluación de cualquier función lógica; además si se esta unidad tiene la capacidad de sumar, también se pueden realizar las operaciones aritméticas de resta, multiplicación y división. Esta unidad recibe el nombre de unidad lógico aritmética o ALU (del inglés *Arithmetic Logic Unit*).

Las entradas a esta unidad son dos palabras de N bits ($A = A_{N-1} \dots A_1 A_0$ y $B = B_{N-1} \dots B_1 B_0$) además de J líneas de selección de la operación lógico aritmética a realizar ($S_{J-1} S_{J-2} \dots S_1 S_0$), y las salidas son los N bits de la palabra (Z) resultante de la operación realizada ($Z = Z_{N-1} \dots Z_1 Z_0$) más un posible bit de acarreo de las operaciones aritméticas (Z_N o C_{N-1}). La figura 7.20 muestra el símbolo lógico con entradas y salidas para una ALU y donde las líneas gruesas representan a grupos de bits.

Figura 7.20: Símbolo lógico de una ALU de N bits

Una ALU básica funcional, con $J = 3$ líneas de selección, permite implementar 8 funciones, 4 lógicas y 4 aritméticas. La tabla 7.20 enseña las funciones lógicas a implementar, según el código de selección.

Tabla 7.20: Tabla de funciones en una ALU con 3 bits de selección

Líneas de selección			Función	Descripción
S_2	S_1	S_0		
0	0	0	$A + B$	Suma
0	0	1	$A - B$	Resta
0	1	0	$A + 1$	Incremento
0	1	1	$A - 1$	Decremento
1	0	0	$A \text{ AND } B$	AND lógica
1	0	1	$A \text{ OR } B$	OR lógica
1	1	0	$\text{NOT } A$	NOT lógica
1	1	1	$A \text{ XOR } B$	XOR lógica

El diseño de una ALU se realiza por medio de síntesis modular, con un enfoque similar al de un RCA, donde se dispone de varias ALU de 1-bit que se interconectan entre sí en cascada, con el fin de propagar los acarreo en las operaciones aritméticas. Las operaciones lógicas no requieren de acarreo, caso en el cual las líneas de acarreo no cumplen función alguna. Además, para las operaciones aritméticas, es necesario controlar el bit de acarreo inicial, con el fin de poder implementar restas por complemento a 2; a este bloque se le denomina C-CONTROL. Bajo el esquema descrito, y con el subíndice i haciendo referencia al i -ésimo bit de las palabras de entrada o salidas, una ALU de 1-bit tiene como entrada los bits A_i , B_i , el bit del acarreo de la etapa previa C_{i-1} y los 3 bits de selección de operación ($S_2S_1S_0$); las salidas son el bit producto de realizar la operación Z_i y el acarreo de propagación C_i . El esquema modular de una ALU se muestra en la figura 7.21.

Una ALU de 1-bit es una unidad que se divide en dos partes, una sección para las operaciones lógicas (LU, del inglés *Logic Unit*) y otra para las operaciones aritméticas (AU, del inglés *Arithmetic Unit*). Según la tabla 7.20, los bits S_1 y S_0 indican cuál de las 4 operaciones se hace en cada unidad LU y AU; además el bit de selección S_2 determina la salida de la ALU de 1-bit mediante un multiplexor 2 a 1, donde la operación de salida es aritmética si $S_2 = 0$ o lógica si $S_2 = 1$. La figura 7.22 muestra los módulos descritos para una ALU de 1-bit.

Figura 7.21: Esquema modular de una ALU con 3 bits de selección

Figura 7.22: Módulos de una ALU de 1-bit

La LU debe entregar la operación lógica indicada por las líneas de selección S_1 y S_0 . Una forma sencilla de implementar esta funcionalidad es usar un multiplexor 4 a 1, donde las dos líneas de selección de la operación lógica coinciden con las dos líneas de selección del multiplexor, tal como se indica en la tabla 7.21 y se muestra en la figura 7.23.

La LU de 1 bit, también se puede deducir desde una tabla de verdad, lo cual garantiza una profundidad de 3 capas, mejorando en solo una capa una implementación con base en el multiplexor 4 a 1.

La AU debe entregar la operación aritmética indicada por las líneas de selección S_1 y S_0 . Ya que las operaciones a diseñar son básicamente de suma o resta, se pueden implementar bajo el mismo

Tabla 7.21: Tabla de funciones de una LU

Selección		Función
S_1	S_0	
0	0	$A_i \text{ AND } B_i$
0	1	$A_i \text{ OR } B_i$
1	0	$\text{NOT } A_i$
1	1	$A_i \text{ XOR } B_i$

Figura 7.23: Unidad lógica de 1-bit

concepto de un RCA. Cada AU posee, entonces, un sumador completo (FA) que recibe a su vez el acarreo previo y entrega el resultado de la operación y un acarreo de salida. Las operaciones a realizar se indican en la tabla 7.22.

Tabla 7.22: Tabla de funciones de una AU

Selección		Función
S_1	S_0	
0	0	$A_i + B_i$
0	1	$A_i - B_i$
1	0	$A + 1$
1	1	$A - 1$

La operación de suma implica que $Z_i = A_i + B_i + C_{i-1}$, para la operación de resta $Z_i = A_i + \overline{B_i} + C_{i-1}$ y se debe sumar $+1$ en el acarreo C_{-1} , para el incremento $Z_i = A_i + 0$ y se debe sumar $+1$ en el acarreo C_{-1} . Para realizar el decremento se debe tener presente que, por ejemplo, usando 4 bits, se resta $(0001)_2$ que pasado a complemento a 2 es $[1111]_2$ y finalmente $Z_i = A_i + 1$. El resumen de las acciones a realizar para implementar las operaciones aritméticas en una AU se muestra en la tabla 7.23.

Tabla 7.23: Tabla de funciones y acciones de una AU

Selección		Función	A_i	B_i	C_{-1}
S_1	S_0				
0	0	$A_i + B_i$	A_i	B_i	0
0	1	$A_i - B_i$	A_i	$\overline{B_i}$	1
1	0	$A + 1$	A_i	0	1
1	1	$A - 1$	A_i	1	0

De la tabla 7.23 se puede concluir fácilmente que la caja C-CONTROL en la figura 7.21 tiene como salida $C_{-1} = S_1 \oplus S_0$, que la señal A_i que llega al FA siempre es igual sin importar la operación a realizar y que la señal B_i cambia según la operación. Por tanto, se requiere de una función que controle el bit B_i , B-CONTROL, para implementar la AU, como se indica en la figura 7.24.

Figura 7.24: Módulos de una AU

La tabla 7.24 muestra la tabla de verdad para B-CONTROL.

Tabla 7.24: Tabla de verdad para B-CONTROL

Selección		B_i	B-CONTROL
S_1	S_0		
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

Una forma de implementar B-CONTROL es mediante un multiplexor 4 a 1 como generador de tablas de verdad, tal como se indica en la tabla 7.25, y donde la forma canónica de B-CONTROL es $\sum m(1, 2, 6, 7) = \overline{S_1} \overline{S_0} B_i + \overline{S_1} S_0 \overline{B_i} + S_1 S_0 \overline{B_i} + S_1 S_0 B_i$.

Tabla 7.25: Tabla de verdad para B-CONTROL con multiplexor 4 a 1

S_1	S_0	B-CONTROL= $\overline{S_1} \overline{S_0} B_i + \overline{S_1} S_0 \overline{B_i} + S_1 S_0 \overline{B_i} + S_1 S_0 B_i$
0	0	B_i
0	1	$\overline{B_i}$
1	0	0
1	1	$B_i + \overline{B_i} = 1$

El circuito lógico para B-CONTROL se muestra en la figura 7.25.

Figura 7.25: Circuito lógico de B-CONTROL con multiplexor 4 a 1

Para evaluar la complejidad y la profundidad se tiene presente los datos para cada módulo implementado, según la tabla 7.26, y donde la base de funciones booleanas empleada está conformada por $C_{\Omega}(f_n)=\{\text{AND, OR, NOT, XOR, FA, MUX}\}$, de tal forma que los elementos modulares, como el FA y los multiplexores, se toman como elementos que solo producen un retardo.

Tabla 7.26: Tabla de complejidad y profundidad para los módulos en una ALU de N bits

Módulo	$C(S)$	Descripción de elementos	$D(S)$	Descripción de capas, o retardos
C-CONTROL	1	XOR	1	Capa XOR
B-CONTROL	2	Mux 4 a 1 y NOT	2	Capa NOT y MUX 2 a 1
LU	5	Mux 4 a 1 y 4 compuertas	2	Capa por compuertas y Mux 4 a 1
AU	3	FA y elementos B-CONTROL	3	2 capas en B-CONTROL y FA
ALU-1bit	9	Elementos LU, AU y Mux 2 a 1	4	3 capas en AU y Mux 2 a 1

Ya que las ALU de 1-bit se conectan en cascada, la propagación se realiza con el acarreo a través de las AU, con lo cual existen $N - 1$ etapas de ALU de 1-bit que deben esperar por el acarreo (que es un tiempo de compuerta más rápido que la salida aritmética, por no requerir pasar por el multiplexor 2 a 1), la etapa inicial debe esperar con C-CONTROL y en la etapa final se debe esperar por la salida aritmética que es más lenta que la salida de acarreo, con lo cual la profundidad de una ALU de N bits es $D(S) = \{(N - 1) * 3 + 1 + 4\}t_g = (3N + 2)t_g$. La complejidad de la ALU de N bits es igual a N veces la complejidad de una ALU de 1-bit más la complejidad por C-CONTROL, por lo que $C(S) = 9N + 1$ elementos en total. Se debe notar que, en una ALU de 1-bit, la salida en la LU es más rápida que la salida en la AU, por lo que esta primera no se tiene en cuenta para el análisis de profundidad.

7.4. EJERCICIOS PROPUESTOS

1. Diseñar un decodificador de entrada hexadecimal de 4 bits, para despliegue en 7 segmentos.
2. Diseñar un sumador completo de 2 bits, empleando un decodificador para generar los minterminos. Repetir con los maxtérminos.
3. Implementar, mediante síntesis por expansión, un decodificador 4x16 empleando decodificadores 2x4 con entrada de habilitación.
4. Implementar las siguientes dos funciones de forma simultánea empleando: a) un decodificador de salidas activas altas para generar los minterminos, b) un decodificador de salidas activas bajas para generar los minterminos, c) un decodificador de salidas activas altas para generar los maxtérminos y d) un decodificador de salidas activas bajas para generar los maxtérminos. Además, comparar la complejidad y profundidad de las cuatro soluciones al universalizar las compuertas adicionales requeridas y restringir el fan-in a dos entradas.

$$f_1(A, B, C) = \sum m(0, 1, 4, 6)$$

$$f_2(A, B, C) = \sum m(1, 3, 4, 5, 7)$$

5. Diseñar un circuito para un codificador Decimal a BCD+3 sin prioridad.
6. Diseñar un circuito para un codificador Decimal a Gray con prioridad.
7. Diseñar un circuito mediante multiplexores que seleccione una palabra, entre cuatro palabras de entrada de 4 bits cada una.
8. Implementar la función solicitada usando los siguientes tipos de multiplexores, como generadores de tablas de verdad: a) con un multiplexor 16 a 1, b) con un multiplexor 8 a 1, c) con un multiplexor 4 a 1 y d) con un multiplexor 2 a 1. Además, comparar la complejidad y profundidad de las cuatro soluciones al universalizar las compuertas adicionales requeridas y restringir el fan-in a dos entradas.

$$f_1(A, B, C, D) = \sum m(0, 1, 3, 5, 8, 9, 12, 15)$$

9. Implementar, mediante síntesis por expansión, un demultiplexor 1 a 16 empleando demultiplexores 1 a 4 con entrada de habilitación.
10. Realizar diseño modular para un circuito que toma 10 líneas de entrada, donde cada una representa un dígito decimal, y entrega, de acuerdo con unas líneas de selección, una salida codificada en una de las siguientes cuatro opciones: BCD natural, BCD+3, BCD2421 y Gray. Además, existe otra línea de entrada para determinar si las salidas son codificadas con prioridad o sin prioridad.

Referencias

- [1] F. Gray, "Pulse code communication," U.S. Patent 2,632,058, March 17, 1953.
- [2] D. J. Costello and S. Lin, *Error Control Coding*. Prentice Hall, 2004, 2nd edition. isbn 10: 0130426725.
- [3] K. Hwang, *Computer Arithmetic: Principles, Architecture and Design*. John Wiley & Sons, 1979, isbn: 9780471034964, New York.
- [4] F. Tinker, *Number Systems: The Electrical Engineering Handbook*. Academic Press, 2000, 2nd edition, isbn:-691295-5.
- [5] T. L. Floyd, *Fundamentos de Sistemas Digitales*. Pearson Education, 2006, 9na edicion. isbn 10:-085-7.
- [6] V. P. Nelson, B. D. Carroll, and J. D. Irwin, *Análisis y Diseño de Circuitos Lógicos Digitales*. Prentice Hall, 1996, isbn: 9789688807064.
- [7] J. F. Wakerly, *Diseño Digital, Principios y Prácticas*. Prentice Hall, 2006, 3ra edicion. isbn: 9789702607205.
- [8] H. Garner, "Number systems and arithmetic," *Advances in Computers*. vol. 6, pp. 131–194, 1965.
- [9] D. Booth, "A signed binary multiplication technique," *The Quarterly Journal of Mechanics and Applied Mathematics, Volume IV, Part 2*, 1951.
- [10] W. Stallings, *Computer Organization and Architecture: Designing for performance*. New Jersey: Prentice-Hall, Inc., 2000, 5th edition. isbn:-081294-3.
- [11] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press,, 2000, isbn-512583-5.
- [12] I. Koren, *Computer Arithmetic Algorithms*. A.K. Peters, 2002, 2nd edition. isbn:1-160-8.
- [13] J. Lloyd, *Foundations of Logic Programming*. Springer Verlag, 1987, 2nd edition. isbn:-642-96826-6.

- [14] G. Brookshear, *Teoría de la Computación: lenguajes formales, autómatas y complejidad*. Addison Wesley Iberoamericana, 1993, isbn:-60119-2.
- [15] J. Hopcroft, R. Motwani, and J. D. Ullman, *Introducción a la teoría de autómatas lenguajes y computación*. Pearson Educación S.A., Madrid, 2007, 3ra edición. isbn:-7829-088-8.
- [16] S. Sahni, *Concepts in Discrete Mathematics*. Camelot Publishing Co, 1985, 2nd edition. isbn-10: 0942450000.
- [17] B. Ramón, *Autómatas y Lenguajes, Un enfoque de diseño*. McGraw-Hill, 2014, isbn: 9781456220532.
- [18] J. E. Labra Gallo and D. Fernández Lanvin, *Lógica de Predicados, cuaderno didáctico*. Escuela Universitaria de Ingeniería Técnica en Informática de Oviedo. Universidad de Oviedo, 2016.
- [19] A. Baker, *Breve Introducción a la Teoría de Números*. Alianza Universidad, Madrid, 1986, isbn 2472-1.
- [20] J. Moster, *Lógica de Primer Orden*. Ariel, Barcelona, 1976, 2da edición. isbn: isbn 10: 8434439395.
- [21] J. Devlin, *The Joy of Sets. Fundamentals of contemporary set theory*. Springer, New York, 1993, isbn-4612-0903-4.
- [22] I. Copi, *Lógica Simbólica*. Compañía Editorial Continental, México, 2000, isbn: 9682601347.
- [23] M. Ben-Ari, *Mathematical Logic for Computer Science*. Springer-Verlag, 2001, 2nd edition. isbn3-319-7.
- [24] M. R. Genesereth and N. J. Nilsson, *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, Inc, 2014, isbn-10: 1493305980.
- [25] G. Boole, *An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities*. Walton and Maberly, 1854, vol. 2.
- [26] C. E. Shannon, "A symbolic analysis of relay and switching circuits," *Electrical Engineering*, vol. 57, no. 12, pp. 713–723, 1938.
- [27] E. J. McCluskey, *Introduction to the theory of switching circuits*. McGraw-hill, 1965, isbn-10: 0070448434.
- [28] —, *Logic design principles with emphasis on testable semicustom circuits*. Prentice-Hall, Inc. Englewood Cliffs, 1986, isbn: 0135397847.
- [29] J. Venn, "I. on the diagrammatic and mechanical representation of propositions and reasonings," *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, vol. 10, no. 59, pp. 1–18, 1880.

- [30] M. M. Mano and M. Ciletti, *Digital design: with an introduction to the Verilog HDL*. Pearson, 2013.
- [31] R. F. Tinder, *Digital Engineering Design: A Modern Approach*. USA: Prentice-Hall, Inc., 1991.
- [32] —, *Engineering Digital Design*. San Diego: Academic Press, 2000. [Online]. Available: <https://www.sciencedirect.com/book/9780126912951/engineering-digital-design>
- [33] F. J. Hill and G. R. Peterson, *Digital Logic and Microprocessors*. USA: John Wiley & Sons, Inc., 1984.
- [34] C. Roth and L. Kinney, *Fundamentals of Logic Design*. Cengage Learning, 2013.
- [35] P. Halmos, *Naive Set Theory*. Springer-Verlag, 2021.
- [36] E. W. Veitch, “A chart method for simplifying truth functions,” in *Proceedings of the 1952 ACM National Meeting (Pittsburgh)*, ser. ACM ’52. New York, NY, USA: Association for Computing Machinery, 1952, p. 127–133. [Online]. Available: <https://doi.org/10.1145/609784.609801>
- [37] M. Karnaugh, “The map method for synthesis of combinational logic circuits,” *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, vol. 72, no. 5, pp. 593–599, 1953.
- [38] S. R. Petrick, “A recognition procedure for transformational grammars,” Ph.D. dissertation, Massachusetts Institute of Technology, 1965.
- [39] L. Lind, N. Frederick, and J. C. Cunliffe, *Analysis and design of sequential digital systems*. Macmillan International Higher Education, 1977, isbn 0-333-19266-4.
- [40] A. M. Rushdi and H. A. Al-Yahya, “Variable-entered karnaugh map procedures for obtaining the irredundant disjunctive forms of a switching function from its complete sum,” *Journal of King Saud University - Engineering Sciences*, vol. 14, no. 1, pp. 13–26, 2002. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1018363918307414>
- [41] A. Rushdi and H. Albarakati, “Using variable-entered karnaugh maps in determining dependent and independent sets of boolean functions,” *Journal of King Abdulaziz University: Computers and Information Technology*, vol. 1, pp. 45–67, 01 2012.
- [42] A. Velarde-Martínez, “Generación de grafos dirigidos acíclicos mediante métodos aleatorios para la planificación de tareas,” *Conciencia Tecnológica*, 2018. [Online]. Available: <https://www.redalyc.org/articulo.oa?id=94455712004>
- [43] I. Wegener, *The Complexity of Boolean Functions*. USA: John Wiley & Sons, Inc., 1987.
- [44] A. V. Aho and J. E. Hopcroft, *The Design and Analysis of Computer Algorithms*, 1st ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1974.

- [45] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third edition*, 3rd ed. The MIT Press, 2009.
- [46] J. Plaza Castillo, *Guía de Electrónica Digital*. Universidad del Atlántico, Barranquilla, Colombia, 2011. [Online]. Available: https://www.uniatlantico.edu.co/uatlantico/pdf/arc_12399.pdf
- [47] S. Dormido, J. Mira, M. Canto, and A. Delgado, *Electrónica Digital (Teoría y problemas)*. España, Editorial Sanz y Torres, 2010.
- [48] N. Balabanian and B. Carlson, *Principios de Diseño Lógico Digital*. CESCO, 2003, 1ra edición.
- [49] R. Tocci, N. Widmer, and G. Moss, *Digital Systems*. Pearson Education Limited, 2017, 12th edition, isbn-10: 1292162007.
- [50] A. Elahi, *Computer Systems: Digital Design, Fundamentals of Computer Architecture and Assembly Language*. Springer International Publishing, 2018, 1st edition, isbn: 978-3-319-66774-4.
- [51] L. Li and M. Thornton, *Digital System Verification: A Combined Formal Methods and Simulation Framework*. Morgan and Claypool Publishers, 2010, isbn-10: 160845178X.
- [52] W. J. Dally and R. C. Harting, *Digital Design: A Systems Approach*. Cambridge University Press, 2012, 1st edition, isbn-10: 0521199506.
- [53] C. Unsalan and B. Tar, *Digital System Design with FPGA: Implementation Using Verilog and VHDL*. McGraw-Hill Education, 2017, 1st edition, isbn-10: 9781259837906.
- [54] R. Kamal, *Digital Systems: Principles and Design*. Prentice Hall, 2009, 1st edition, isbn-13: 978-8177585704.
- [55] J. Crisp, *Introduction to Digital Systems*. Newnes, 2000, 1st edition, isbn-13: 9780080535081.
- [56] G. Donzellini, L. Oneto, D. Ponta, and D. Anguita, *Introduction to Digital Systems Design*. Italia, Springer International Publishing, 2019, 1st edition, isbn-13: 978-3-319-92804-3.
- [57] J. Deschamps, E. Valderrama, and L. Terés, *Digital Systems: From Logic Gates to Processors*. Springer International Publishing, 2017, 1st edition, isbn-13: 978-3-319-41198-9.

Índice alfabético

Álgebra Booleana, 95, 96, 106, 133, 134, 178,
Agregador de paridad, 208, 211
Alfabeto, 53, 75, 78, 90
Algoritmo de Petrick, 157, 186,
Algoritmo Q-M-P con prescindibles, 158, 159, 160
Algoritmo de Quine-McCluskey (Q-M), 181
AND, 97, 98, 104, 107, 115, 116, 117, 126, 152, 177, 178, 179, 182, 184, 185, 186, 187,
188, 189, 191, 192, 198, 200, 204, 206, 208, 209, 215, 221, 223, 224, 230, 233, 234, 240,
241, 242, 244, 246
Aridad, 77
Aritmética binaria, 15, 38, 195,
BCD Natural, 53, 54, 129, 162, 164, 193, 194, 207, 208, 209, 211, 215, 222, 224, 230,
232, 248
BCD+3, 53, 54, 247, 248
BCD2421, 53, 54, 248
BCD84-2-1, 54
BCD8421, 53, 54, 55, 56, 57
Bicondicional, 76, 77, 80, 87, 88,
Binario, 17, 18, 20, 21, 25, 26, 29, 35, 36, 37, 42, 52, 53, 55, 58, 59, 60, 61,
Bit, 30, 31, 32, 33, 34, 35, 36, 38, 39, 40, 44, 45, 46, 48, 49, 51, 52, 53, 54, 55, 56, 57, 58,
59, 60, 61, 62, 63, 64, 65, 67, 84, 135, 136, 137, 148, 154, 155, 179, 183, 185, 198, 199,
201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 220, 222,
225, 227, 228, 229, 234, 235, 236, 237, 241, 242, 243, 244, 245, 246, 247
Byte, 29, 37
Capas de evaluación, 181, 182, 184, 185, 187, 188
CDD, 37
Circuito, 98, 99, 100, 101, 102, 103, 118, 126, 127, 133, 134, 174, 178, 179, 181, 182,
185, 186, 191, 197, 203, 205, 208, 209, 215, 220, 221, 225, 229, 232, 248
Circuito lógico, 105, 124, 125, 129, 169, 192, 198, 199, 201, 205, 221, 223, 224, 231,
233, 234, 239, 241, 246, 247
Circuitos integrados, 134, 174, 175, 177, 187, 189
Codificador con prioridad, 230
Codificador sin prioridad, 228
Codificador/Decodificador de paridad, 209, 210
Codificadores, 60, 220, 229, 230
Codificar, 20, 53, 65, 66, 67, 219
Coimplicador, 76, 85, 89

Comparador binario de 2 bits, 205
Comparador binario de 4 bits, 214
Compatibilidad entre familias lógicas, 176
Complejidad circuital, 169, 181
Complemento a 1, 32, 35, 54, 202
Complemento a 2, 33, 35, 38, 44, 202, 242, 244
Complemento a la base, 33, 34, 38, 41, 42, 43, 44, 47, 48, 49, 51, 52, 55, 209
Complemento disminuido a la base, 31, 32, 33, 54, 55, 56
Compuerta lógica, 169, 170, 171, 172, 173
Condicional, 76, 77, 80
Conectivas lógicas, 76, 77, 80, 81, 84, 85, 89, 90, 97
Conjunciones, 80, 81
Conjunción, 76, 77, 79, 82, 85, 89, 107, 118
Constantes, 75, 177
Contador de unos, 212, 227, 228, 237, 238, 239
Conteo, 20, 21, 58
Contradicción, 87, 89, 91, 97, 127, 178
Conversión Binario a Gray, 59
Conversión entre Bases, 15, 22, 27
Conversión Gray a Binario, 59
Convertidor análogo - digital, 18
Corrección de riesgos de tiempo, 192
Costo de función, 133
CRC-12, 66, 67
CRC-16, 66, 67
CRC-32, 66, 67
CRC-CCITT, 66, 67
Cuantificador existencial, 76
Cuantificador universal, 76
Cuantificadores, 76
Cuantización, 17, 69
Cubierta de una función, 141
CXT, 37
Código, 17, 52, 54, 61, 62, 64, 95, 169, 181, 220, 222, 224, 226, 227, 229, 230, 233, 236, 242
Código BCD, 53, 54
Código Gray, 59, 60, 134, 215
Código Hamming(7,4), 64, 65
Códigos alfanuméricos, 53
Códigos binarios, 52, 62, 154, 220, 228, 233, 239
Códigos básicos de paridad, 62, 63
Códigos CRC, 65, 66, 67
Códigos de repetición, 63
Códigos Hamming, 64
Códigos para detección de errores, 60, 61
Códigos polinomiales,
DBL, 37

Decimal, 20, 22, 23, 25, 26, 29, 31, 36, 40, 48, 53, 54, 58, 67, 120, 122, 134, 136, 207, 208, 209, 222, 223, 224, 230, 231, 236, 247, 248
 Decodificadores, 219, 220, 222, 223, 224, 225, 226, 227
 Decodificar, 210, 219, 222
 Deducción de maxtérminos, 121
 Deducción de mintérminos, 119, 121
 Demostración, 88, 106, 110, 111, 112
 Demultiplexar, 219
 Desbordamiento, 39, 204, 205
 Descriptor, 76
 Diagrama de tiempo, 104, 105, 106, 128
 Diagrama lógico, 105, 106, 128, 179, 222
 Diagramas de Venn, 107, 108, 128, 134, 137
 Digitalización, 17, 18, 69
 Diseño combinacional, 197
 Disipación de potencia, 174, 175
 Distancia entre palabras,
 Distancia mínima, 62, 64
 Disyunción, 76, 77, 79, 85, 89, 107, 118
 División binaria,
 División con restauración, 49
 División por restas sucesivas,
 División sin restauración, 50, 51
 División tradicional, 48
 Doble Quad Word, 29
 Doble Word, 29
 Dígito, 16, 18, 19, 27, 28, 31, 32, 36, 54, 230, 248
 Dígitos, 17, 18, 19, 20, 25, 26, 27, 28, 29, 31, 53, 129, 179, 224, 232
 Enrutamiento de datos, 219
 Error, 52, 60, 61, 62, 63, 65, 66
 Error múltiple, 60
 Error simple, 60
 Especificación incompleta, 127, 147, 152, 163
 Exponente, 19, 36, 74
 Expresión dual, 109, 117
 EXT
 FA - Full adder,
 Familia CMOS, 175, 193
 Familia ECL, 175
 Familia TTL, 175, 193
 Familias bipolares, 175
 Familias lógicas, 169, 174, 176
 Familias, 175
 Fan-In, 172, 181, 182, 183, 186, 188, 190, 204, 221, 227, 229, 230, 234, 247
 Fan-Out, 172, 175, 181
 Forma POS, 139, 147, 148, 159, 161, 162, 163, 165, 194
 Forma POS canónica, 121, 122, 123, 129, 149
 Forma SOP, 119, 123, 139, 151, 152, 159, 160, 162, 163, 165, 179, 193

Forma SOP canónica, 119, 120, 123, 124, 129, 149
Formas algebraicas estándarES, 118, 178
Funciones lógicas básicas, 97, 126, 128
Funciones lógicas derivadas, 100, 128
Función, 16, 49, 62, 77, 80, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 115, 116, 118, 119, 120, 121, 122, 123, 124, 126, 128, 129, 133, 134, 135, 136, 137, 138, 139, 140, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 158, 159, 160, 161, 163, 165, 169, 170, 177, 178, 181, 183, 184, 185, 186, 187, 188, 189, 193, 194, 197, 198, 203, 207, 208, 209, 226, 229, 233, 234, 239, 240, 241, 242, 244, 245, 247
Función lógica, 95, 98, 99, 100, 101, 102, 103, 104, 105, 115, 116, 126, 134
Función mínima, 162, 163, 164
Fórmula molecular, 79
Fórmulas atómicas, 79
Fórmulas bien formadas, 79, 104
Fórmulas equivalentes, 87
Fórmulas lógicas, 78, 79, 86, 87, 88, 81
Gráfico acíclico dirigido, 179, 180, 181, 189, 190
HA - Half adder, 198
Hexadecimal, 18, 20, 21, 245
IEEE Std 754-1985, 35
Igualador, 75
Implementación mínima global, 183, 185, 182, 187, 188
Implicado primo, 146, 158
Implicador, 76, 85, 89
Implicados, 146, 147, 154, 155, 158, 159, 161, 193
Implicados primos esenciales, 146, 147
Implicante, 159, 160, 185, 187, 193
Implicante primo, 141, 156, 185
Implicantes primos esenciales, 144, 145, 146, 185
Inferencia lógica, 88
Interfaz hombre/máquina, 219, 220
Letras sentenciales, 79, 80
Ley de no contradicción, 89
Ley del medio excluido, 89
Literales, 79, 80, 97, 163, 184, 188
Lógica de proposiciones, 73
Mantisa, 36
Mapas de Veitch-Karnaugh, 133, 134
Mapas VK, 135, 138, 144, 149, 151, 152, 155, 163, 165, 183, 184, 193, 212, 215, 225
Mapas VK con especificación incompleta, 147
Maxtérmino, 121, 122, 123, 124, 127, 128, 129, 134, 141, 144, 147, 148, 150, 151, 153, 154, 158, 161, 162, 163, 206, 226, 227, 228, 237, 247
Minimización, 133, 134, 144, 147, 154, 158, 163, 183, 189, 190, 199, 225
Minimización de funciones lógicas, 127
Minimización por I.P.E., 145, 146, 147, 154, 155
Minimización por vecindades, 144, 154
Mintérmino, 119, 121, 122, 123, 124, 127, 128, 129, 134, 135, 136, 137, 142, 143, 144, 145, 146, 147, 148, 150, 151, 153, 154, 155, 156, 157, 158, 177, 160, 163, 179, 183, 184,

185, 187, 188, 193, 198, 199, 206, 220, 222, 223, 226, 227, 228, 237, 240, 247
Modus ponendo ponens, 89
Modus tollendo tollens, 89
Muestreo, 17
Multiplexar, 219
Multiplicación Binaria,
Multiplicación por Algoritmo de Booth, 42
Multiplicación por restas sucesivas,
Multiplicación tradicional,
Márgenes de ruido, 171, 172, 175
Método de las divisiones sucesivas, 23
Método de las multiplicaciones sucesivas, 24
NAND, 100, 101, 102, 115, 126, 173, 177, 178, 190, 191
Negador, 76, 85, 170
Nibble, 29
Nivel alto, 127, 170, 173, 175, 220, 223, 228
Nivel bajo, 78, 80, 127, 171, 173, 174, 220, 223
Nivel lógico, 169, 171, 172, 173
NOR, 97, 100, 101, 102, 116, 126
NOT, 97, 99, 100, 107, 115, 116
Número normalizado, 36
Números en punto flotante, 35
Octal, 18, 20, 21, 220, 229
OR, 97, 99, 100, 107, 115, 116
OR exclusiva, 102
Oración, 73, 74, 90,
Parámetros de compuertas, 169
Parámetros de voltaje, 169, 170, 171
Paso de evaluación, 179
Pasos de evaluación, 181
Polinomio generador, 66, 67, 68, 69
Postulado, 106, 107, 108, 110, 119, 121, 128
Prescindible, 127, 128, 129, 134, 147, 148, 152, 153, 160, 161, 162, 163, 184, 185, 187,
207, 222, 224, 225
Principio de la Dualidad, 109, 119, 117
Producto de sumas, 118
Producto Retardo-Potencia, 174
Profundidad circuital, 178
Proposición, 73, 74, 75, 80, 88
Proposición atómica, 75, 78, 79, 84
Proposición molecular, 75, 79, 80, 84, 85, 104
Quad Word, 29
Quinario, 18
Rango de los datos, 37
Razonamiento, 88, 106
RCA - Ripple carry adder, 201
Reglas de la inferencia, 88
Relator diádico, 75

Relatores, 75
Representación de números con signo, 15, 29, 30, 36
Representación explícita, 84, 97, 104, 105, 120, 124
Representación implícita, 104, 118
Representación polinomial, 19
Representación por exceso,
Representación posicional, 19, 25
Representación sesgada, 34, 35, 36
Resta BCD8421, 55
Resta BCD8421 en complemento a 10, 57
Resta BCD8421 en complemento a 9, 56
Restador BCD natural, 209, 211
Riesgo dinámico, 191, 192
Riesgo estático, 191, 192, 193
Sentencias, 78, 79, 83, 84, 170,
Sexagesimal, 18
Señal análoga, 16, 17
Señal binaria, 17
Señal digital, 16, 17
Signed Byte, 37
Signed Long, 37
Signed Word, 37
Signo-magnitud, 29, 30, 32, 33, 35, 38, 40, 41, 42, 47, 48, 50, 51
Signos de puntuación, 77
Silogismo Disyuntivo, 89
Silogismo Hipotético, 89
Sintaxis lógica, 78, 79, 104
Sistema, 16, 17, 29, 30, 34, 36, 48, 52, 53, 54, 55, 58, 61, 62, 63, 68, 73, 77, 96, 97, 104, 107, 169, 178,
Sistemas de numeración, 17, 18, 23, 29
Sistemas de valor posicional, 18, 20
Suficiencia de la NAND, 115
Suficiencia de la NOR, 116
Suma BCD8421, 54
Suma binaria, 179, 198, 199, 201
Suma de productos, 118
Sumador BCD natural, 207, 211
Sumador completo, 199, 200, 201, 244, 247
Sumador en cascada, 200, 202
Sumador medio, 198, 199
Sumador/restador en cascada, 202, 203
Síntesis, 133, 197, 214, 219
Síntesis combinacional, 197
Síntesis de funciones lógicas, 133
Síntesis por expansión, 197, 214, 215, 225, 247, 248
Tablas de verdad, 84, 85, 86, 237, 238, 246, 247
Tautología, 87, 88, 91, 97, 178
Tautologías fundamentales, 88, 89

Teorema, 106
Teorema de Absorción, 117
Teorema del Consenso, 117, 192
Teorema de DeMorgan, 111, 117, 126
Teorema de desarrollo de Shannon, 112, 119, 121
Teorema de Idempotencia, 119, 121, 123, 159
Teoremas del álgebra Booleana, 110
Ternario, 18, 21
Tiempo de propagación, 172, 174
Tiempo de propagación promedio, 174
Tiempo de retardo, 169, 173, 181
Tratamiento de datos, 219
Términos no importa, 127
Unidad Lógico Aritmética, 241
Universalización, 126, 134, 189, 190
Universalización con NAND, 126
Universalización con NOR, 126
Unsigned Byte, 37
Unsigned Long, 37
Unsigned Word, 37
Variable lógica, 89, 95, 96, 107, 169
Variables Booleanas, 107
Variables lógicas, 84, 85, 87, 95, 96, 107, 133
Variables proposicionales, 75, 77, 104
Verdad indeterminada, 86
Verificador de desbordamiento, 204, 205, 210, 213, 214
Verificador de paridad en recepción, 220
VK con ingreso de variables, 149, 150, 151, 152
Word, 29, 37
XNOR, 97, 100, 103, 104, 116, 117, 177, 178
XOR, 97, 100, 102, 103, 116, 117, 129, 178, 180, 182, 198, 200, 203, 204

La Editorial de la Universidad
Tecnológica de Pereira tiene como
política la divulgación del saber
científico, técnico y humanístico para
fomentar la cultura escrita a través
de libros y revistas científicas
especializadas.

Las colecciones de este proyecto son:
Trabajos de Investigación, Ensayos,
Textos Académicos y Tesis Laureadas.

Este libro pertenece a la Colección
Textos Académicos.

Principios Y Métodos Combinatoriales En Sistemas Automáticos Digitales explora los sistemas digitales desde lo relacionado netamente con el mundo combinatorial, dejando los ámbitos secuenciales y de autómatas formales para ser tratados en subsecuentes entregas. El presente libro abarca temáticas sobre la aritmética y codificación binaria como fundamento para el procesamiento de la información digital, la lógica de proposiciones como medio para formalizar un lenguaje natural en una sentencia lógica que luego puede ser descrita, analizada y representada mediante funciones lógicas, gracias al álgebra booleana y métodos de minimización de funciones. Para analizar y diseñar problemas lógicos, se presentan los parámetros de las compuertas y su relación con la complejidad desde los puntos de vista circuital y/o sentencial; además para acometer problemas de mayor envergadura, se introduce la síntesis de problemas y el tratamiento y enrutamiento de datos.

En este libro, el grupo de investigación en Gestión de Sistemas Eléctricos, Electrónicos y Automáticos, del programa de Ingeniería Eléctrica de la Universidad Tecnológica de Pereira, presenta a la comunidad académica la experiencia combinada de sus autores en más de 30 años de labor docente e investigativa. Además, se entregan conocimientos fundamentales, con una conceptualización y profundidad que permiten ser extrapoladas a diversas áreas de aplicación, fuera de una visión integradora y un enfoque hacia una formación moderna en ingeniería, cubriendo los tres aspectos fundamentales en el desarrollo de competencias: el Conocimiento Pedagógico del Contenido, la Transversalidad y la Complejidad.