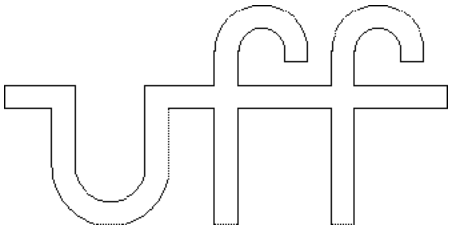
Apostila de Teoria para Circuitos Digitais

(Versão A2025M06D25)



Universidade Federal Fluminense

Alexandre Santos de la Vega

Departamento de Engenharia de Telecomunicações – TET

Escola de Engenharia – TCE

Universidade Federal Fluminense – UFF

Junho – 2025

.

621.3192 (*) D278 (*) 2025	de la Vega, Alexandre Santos Apostila de Teoria para Circuitos Digitais / Alexandre Santos de la Vega. – Niterói: UFF/TCE/TET, 2025. 349 (sem romanos) ou 377 (com romanos) p. (*) Apostila de Teoria – Graduação, Engenharia de Telecomunicações, UFF/TCE/TET, 2025. 1. Circuitos Digitais. 2. Técnicas Digitais. 3. Telecomunicações. I. Título.

(*) OBTER INFO NA BIBLIOTECA, ATUALIZAR E PEDIR NOVO REGISTRO !!!

Aos meus alunos.

Prefácio

O trabalho em questão cobre os tópicos abordados na disciplina Circuitos Digitais.

O presente volume apresenta um conteúdo teórico. O conteúdo prático, utilizando códigos de programas demonstrativos, pode ser encontrado no volume entitulado Apostila com Códigos de Programas Demonstrativos usando a linguagem VHDL para Circuitos Digitais.

As apostilas foram escritas com o intuito de servir como uma referência rápida para os alunos do curso de graduação em Engenharia de Telecomunicações da Universidade Federal Fluminense (UFF).

O material básico utilizado foram as minhas notas de aula que, por sua vez, originaram-se em uma coletânea de livros sobre os assuntos abordados.

A motivação principal foi a de aumentar o dinamismo das aulas. Portanto, deve ficar bem claro que esta apostila não pretende substituir os livros textos ou outros livros de referência. Muito pelo contrário, ela deve ser utilizada apenas como ponto de partida para estudos mais aprofundados, utilizando-se a literatura existente.

Espero conseguir manter o presente texto em constante atualização e ampliação.

Correções e sugestões são sempre bem-vindas.

Alexandre Santos de la Vega
UFF/TCE/TET

Agradecimentos

Àqueles alunos do Curso de Engenharia de Telecomunicações e professores do Departamento de Engenharia de Telecomunicações (TET), da Escola de Engenharia (TCE), da Universidade Federal Fluminense (UFF), que colaboraram com críticas e sugestões bastante úteis à finalização da versão inicial deste trabalho.

Aos ex-funcionários do TET, Arlei, Carmen Lúcia, Eduardo, Francisco e Jussara, pelo apoio constante.

Aos meus alunos, que, além de servirem de motivação principal, obrigam-me sempre a me tentar melhorar, em todos os sentidos.

Mais uma vez, e sempre, aos meus pais, por tudo.

Alexandre Santos de la Vega
UFF/TCE/TET

Conteúdo

Prefácio v

Agradecimentos vii

Sumário ix

Lista de Tabelas xix

Lista de Figuras xxv

I Apresentação 1

1 Descrição do documento 3
1.1 Introdução . . . 3
1.2 Circuitos digitais combinacionais <28 horas> . . . 3
1.3 Circuitos digitais seqüenciais <16 horas> . . . 4
1.4 Circuitos digitais programáveis <8 horas> . . . 4
1.5 Aplicações . . . 4

II Conceitos básicos 5

2 Conceitos básicos 7
2.1 Modelagem matemática . . . 7
2.2 Teoria de sistemas . . . 7
2.3 Definição de sinal . . . 8
2.4 Definição de sistema . . . 8
2.5 Discretização de sinais . . . 9
2.5.1 Definição e classificações . . . 9
2.5.2 Amostragem . . . 10
2.5.3 Amostragem uniforme . . . 11
2.5.4 Quantização . . . 11
2.5.5 Quantização uniforme . . . 12
2.5.6 *Sampling rate*, *bit depth* e *bit rate* . . . 12
2.6 Classificações de sinais . . . 12
2.7 Classificações de sistemas . . . 13
2.8 Estado e variáveis de estado de um sistema dinâmico . . . 13
2.9 Exemplo de aplicação de processamento digital . . . 14
2.9.1 Sistemas de comunicação . . . 14

2.9.2 Subsistemas de comunicação . . . 14
2.9.3 Sinais e meios de transmissão . . . 15
2.10 Processamento de sinais . . . 16
2.11 Arquitetura de sistemas de processamento digital . . . 16
2.12 Implementações analógicas e digitais . . . 19
2.12.1 Definição do problema . . . 19
2.12.2 Características das implementações analógicas e digitais . . . 20
2.13 Exercícios propostos . . . 22

III Circuitos digitais combinacionais 25

3 Circuitos combinacionais: conceitos básicos 27
3.1 Introdução . . . 27
3.2 Hierarquia em *hardware* e em *software* . . . 27
3.3 Codificação . . . 27
3.4 Elementos codificados: informação e quantidade . . . 28
3.5 Representação de informação . . . 28
3.6 Representação de quantidade . . . 28
3.7 Representações em circuitos digitais . . . 29

4 Funções lógicas 31
4.1 Introdução . . . 31
4.2 Mecanismos básicos de raciocínio . . . 31
4.2.1 Raciocínio por indução (ou por analogia) . . . 32
4.2.2 Raciocínio por dedução . . . 32
4.2.3 Estruturas axiomáticas . . . 32
4.2.4 Classificação das lógicas dedutivas . . . 33
4.3 Exemplo introdutório usando lógica clássica . . . 34
4.4 Conceitos básicos . . . 35
4.5 Formulação lógica clássica . . . 35
4.6 Operadores lógicos clássicos . . . 36
4.6.1 Tabela verdade (*truth table*) . . . 36
4.6.2 Funções de 1 variável . . . 37
4.6.3 Funções de 2 variáveis . . . 38
4.6.4 Operadores lógicos básicos . . . 38
4.6.5 Funções de $N_v > 2$ variáveis . . . 39
4.7 Exemplo de modelagem com relações de implicação . . . 39
4.8 Tautologia e equivalência lógica . . . 39
4.9 Teoremas de De Morgan . . . 41
4.10 Exemplos de verificação de equivalências lógicas . . . 41
4.11 Conjunto funcionalmente completo de operadores . . . 42
4.12 Decomposição em funções canônicas . . . 44
4.13 Conectivos de ordem superior . . . 46
4.14 Técnica de *bundling* . . . 46
4.15 Exemplos de aplicação direta de portas lógicas . . . 47
4.15.1 Uso de operador lógico como elemento de controle . . . 47
4.15.2 Identificador de paridade e gerador de paridade . . . 48
4.15.3 Identificador de igualdade entre padrões binários . . . 48

4.16 Blocos funcionais fundamentais . . . 48
4.17 Manipulação algébrica de blocos . . . 49
4.18 Exercícios propostos . . . 52

5 Álgebra de Boole **55**
5.1 Introdução . . . 55
5.2 Postulados de Huntington . . . 56
5.3 Versão alternativa para os Postulados de Huntington . . . 57
5.4 Dualidade . . . 57
5.5 Lemas e teoremas fundamentais . . . 58
5.6 Definição de uma estrutura algébrica particular . . . 59
5.7 Exemplos de associação com a estrutura algébrica de Boole . . . 59
5.8 Isomorfismo . . . 60
5.9 Simplificação algébrica de expressões lógicas . . . 60
5.10 Exemplo de manipulação algébrica não sistemática: Postulados, Lemas, Teoremas 60
5.11 Exemplo de manipulação algébrica por isomorfismo: Diagrama de Venn . . . 63
5.12 Resumo das relações algébricas . . . 64
5.13 Exercícios propostos . . . 66

6 Formas padrões para representação de expressões booleanas **69**
6.1 Introdução . . . 69
6.2 Definições . . . 70
6.3 Obtenção de formas SOP e POS padrões . . . 71
6.3.1 Complementação da lista de termos canônicos . . . 71
6.3.2 Manipulação algébrica . . . 72
6.3.3 Utilização de tabela verdade . . . 74
6.3.4 Negação das formas SOP e POS . . . 76
6.4 Conjuntos de formas padrões . . . 77
6.4.1 Definições . . . 77
6.4.2 Obtenção . . . 78
6.4.3 Utilização . . . 78
6.5 Exercícios propostos . . . 80

7 Simplificação algébrica sistemática de expressões booleanas **83**
7.1 Introdução . . . 83
7.2 Definição do ponto final: forma mínima procurada . . . 83
7.3 Processo de expansão vs processo de simplificação . . . 84
7.3.1 Expansão sem redundância . . . 85
7.3.2 Expansão com redundância . . . 85
7.3.3 Análise das expansões . . . 86
7.4 Definição do ponto inicial: forma máxima padrão . . . 86
7.5 Simplificação sistemática de expressões booleanas a partir de SOP e POS padrões 87
7.5.1 Operações básicas: aglutinação e replicação . . . 87
7.5.2 Uso da aglutinação . . . 87
7.5.3 Uso da replicação . . . 88
7.6 Eliminação sistemática de literais . . . 88
7.7 Implicantes e implicados . . . 91
7.7.1 Implicantes . . . 91
7.7.2 Implicados . . . 91

7.7.3 Implicantes, implicados e o processo de simplificação . . . 92
7.8 Processo sistemático de simplificação . . . 92
7.9 Subjetividade, complexidade e formas alternativas . . . 93
7.10 Minimização com estruturas de ordem superior: fatoração . . . 93
7.11 Minimização vs *hazards* e *glitches* . . . 94
7.12 Exemplos de simplificação . . . 95
7.12.1 Funções de três variáveis . . . 95
7.12.2 Funções de quatro variáveis . . . 100
7.13 Exercícios propostos . . . 109

8 Mapa de Karnaugh **111**
8.1 Introdução . . . 111
8.2 Motivação para o desenvolvimento do mapa-K . . . 112
8.2.1 Complexidade da busca dos termos na simplificação de funções . . . 112
8.2.2 Simplificação de funções por meio da tabela verdade . . . 112
8.2.3 Redução da complexidade da busca: primeira tentativa . . . 112
8.2.4 Modificação na organização da tabela verdade . . . 114
8.2.5 Motivação final para a criação do mapa de Karnaugh . . . 117
8.3 Construção do mapa-K . . . 117
8.3.1 Funções de 1 variável . . . 117
8.3.2 Funções de 2 variáveis . . . 118
8.3.3 Funções de 3 variáveis . . . 119
8.3.4 Funções de 4 variáveis . . . 120
8.4 Preenchimento do mapa-K . . . 121
8.5 Mapa-K como forma de expressão . . . 121
8.6 Mapa-K na simplificação de expressões booleanas . . . 123
8.6.1 Adjacência lógica, aglutinação e replicação . . . 123
8.6.2 Seleção sistemática de termos (implicantes ou implicados) . . . 124
8.6.3 Mapa-K de funções com múltiplos mínimos e mapa cíclico . . . 125
8.6.4 Mapa-K de funções não completamente especificadas . . . 125
8.7 Exercícios propostos . . . 128

9 Sistemas de numeração **131**
9.1 Introdução . . . 131
9.2 Sistema de numeração posicional convencional . . . 133
9.2.1 Representação de números inteiros não negativos . . . 133
9.2.2 Representação de números fracionários não negativos . . . 134
9.2.3 Representação de números inteiros negativos . . . 135
9.2.4 Representação de números fracionários negativos . . . 146
9.2.5 Conversão entre bases . . . 148
9.2.6 Bases mais comuns em circuitos digitais . . . 150
9.3 Quantização . . . 152
9.3.1 Conceitos básicos . . . 152
9.3.2 SNPC: resolução, base e quantidade de dígitos . . . 152
9.3.3 Classificações . . . 154
9.4 Códigos numéricos . . . 154
9.4.1 Introdução . . . 154
9.4.2 Códigos numéricos comuns . . . 154
9.4.3 Outros códigos numéricos . . . 154

9.4.4 Códigos BCD . . . 156
9.5 Representação em ponto flutuante . . . 159
9.5.1 O padrão de ponto flutuante IEEE 754 . . . 159
9.5.2 Ponto flutuante × ponto fixo . . . 160
9.6 Aritmética binária . . . 162
9.6.1 Tabelas de operações básicas entre dígitos . . . 162
9.6.2 Escalamento por potência inteira da base . . . 164
9.6.3 Adição e subtração em complemento-a-2 . . . 165
9.6.4 Funções envolvidas na adição de dígitos binários . . . 168
9.7 Exercícios propostos . . . 171

10 Circuitos combinacionais básicos **173**
10.1 Introdução . . . 173
10.2 Interpretações dos circuitos combinacionais . . . 173
10.2.1 Exemplos de interpretações . . . 174
10.3 Uso de portas lógicas como elementos de controle . . . 175
10.4 Uso de elementos de controle para mascaramento . . . 175
10.5 Gerador de funções lógicas . . . 175
10.6 Conversor de códigos . . . 175
10.6.1 Exemplo de projeto para conversor de códigos . . . 175
10.7 Gerador e detector de paridade . . . 179
10.8 Multiplexador e demultiplexador . . . 179
10.8.1 Exemplos de projeto de multiplexador . . . 179
10.8.2 Exemplos de projeto de demultiplexador . . . 181
10.9 Codificador e decodificador de endereço . . . 184
10.10 Codificador de prioridade . . . 185
10.10.1 Exemplos de projeto de codificador de prioridade . . . 185
10.11 Ordenador, separador ou agrupador binário . . . 187
10.11.1 Célula básica para um separador binário . . . 187
10.11.2 Estrutura modular com isolamento de um *bit* por vez . . . 188
10.11.3 Estrutura modular com sucessivas trocas em paralelo . . . 188
10.12 Deslocadores (*shifters*) . . . 188
10.13 Somadores em binário puro . . . 189
10.13.1 *Half-adder* (HA) . . . 189
10.13.2 *Full-adder* (FA) . . . 189
10.13.3 *Ripple-carry adder* (RCA) ou *carry propagate adder* (CPA) . . . 190
10.13.4 *Carry lookahead adder* (CLA) . . . 190
10.14 Subtratores em binário puro . . . 192
10.14.1 *Half-subtractor* (HS) . . . 192
10.14.2 *Full-subtractor* (FS) . . . 192
10.14.3 *Ripple-borrow subtractor* (RBS) ou *borrow propagate subtractor* (BPS) . . . 192
10.15 Somadores/subtratores configuráveis (binário puro) . . . 192
10.15.1 *Full-adder-subtractor* . . . 192
10.16 Incrementador e decrementador em binário puro . . . 193
10.17 Complementadores . . . 194
10.17.1 Complementador-a-1 (*bitwise implementation*) . . . 194
10.17.2 Complementador-a-2 . . . 194
10.18 Multiplicadores em binário puro . . . 195
10.18.1 Multiplicador de 1 *bit* . . . 195

10.18.2 Multiplicador de N *bits* . . . 195
10.19 Detector de *overflow* em adição de valores codificados em complemento-a-2 . . . 195
10.20 Saturador (valor codificado em complemento-a-2) . . . 195
10.21 Comparadores da quantidade de dígitos . . . 196
10.21.1 Comparador da quantidade de dígitos em um operando . . . 196
10.21.2 Comparador da quantidade de dígitos em dois operandos . . . 196
10.22 Comparadores numéricos de dois operandos . . . 196
10.22.1 Identificador de igualdade . . . 196
10.22.2 Projeto modular (*bit scanning*) . . . 197
10.22.3 Projeto usando a técnica de complemento . . . 200

IV Circuitos digitais seqüenciais 201

11 Circuitos seqüenciais: conceitos básicos 203
11.1 Introdução . . . 203
11.2 Estados e variáveis de estado . . . 203
11.3 Tipos de variáveis e sua interações . . . 204
11.4 Modelo genérico para circuitos seqüenciais . . . 205
11.5 Classificação quanto à dependência do sinal de saída . . . 207
11.6 Classificação quanto ao tipo de controle da mudança de estado . . . 208
11.6.1 Circuitos seqüenciais *clock-mode* ou *clocked* . . . 208
11.6.2 Circuitos seqüenciais *pulsed* . . . 209
11.6.3 Circuitos seqüenciais *level-mode* . . . 210

12 Elementos básicos de armazenamento 211
12.1 Introdução . . . 211
12.2 Classificação quanto à funcionalidade . . . 212
12.3 Relacionamento entre os tipos básicos de *flip-flops* . . . 213
12.4 Mapas de excitação dos *flip-flops* . . . 215
12.5 Tipos de comportamento das saídas dos *flip-flops* . . . 215
12.6 Excitação × comportamento . . . 216
12.7 Funcionalidade × excitação × comportamento . . . 216
12.8 Circuitos seqüenciais × tabelas dos *flip-flops* . . . 217
12.9 Estruturas estáticas simétricas . . . 220
12.10 Exemplos de *flip-flops* . . . 221
12.10.1 *Flip-flops* do tipo *unclocked* . . . 221
12.10.2 *Flip-flops* do tipo *clocked* . . . 224
12.11 Variações de funcionalidade . . . 229
12.12 Diferenças de nomenclatura . . . 229

13 Circuitos seqüenciais *clock-mode* 231
13.1 Introdução . . . 231
13.2 Controle de circuitos do tipo *clock-mode* . . . 232
13.2.1 Características da estrutura *clock-mode* . . . 232
13.2.2 Controle de circuitos do tipo Moore . . . 232
13.2.3 Controle de circuitos do tipo Mealy . . . 233
13.3 Representação dos estados . . . 234
13.4 Estado inicial . . . 234

13.5 Classificação quanto à capacidade de memorização . . . 234
13.6 Análise de circuitos seqüenciais . . . 236
13.6.1 Etapas genéricas para análise . . . 236
13.6.2 Exemplos de análise . . . 236
13.7 Projeto de circuitos seqüenciais . . . 239
13.7.1 Opções de projeto e suas características . . . 239
13.7.2 Etapas genéricas para projeto . . . 240
13.7.3 Exemplos de projeto . . . 242
13.8 Minimização de estados . . . 252
13.8.1 Conceitos básicos . . . 252
13.8.2 Eliminação de estados redundantes por simples inspeção . . . 252
13.8.3 Método da partição em classes de estados indistinguíveis . . . 254
13.8.4 Método da tabela de implicação de estados . . . 257
13.9 Atribuição de estados . . . 259
13.9.1 Considerações iniciais . . . 259
13.9.2 Base teórica para as regras de atribuição de estados . . . 260
13.9.3 Exemplo de regras simples (Armstrong-Humphrey) . . . 266
13.9.4 Exemplo de regras mais refinadas . . . 267
13.10 Efeitos causados por estados extras . . . 267
13.10.1 Definição do problema . . . 267
13.10.2 Possíveis soluções . . . 268

14 Circuitos seqüenciais *pulsed* 269
14.1 Introdução . . . 269
14.2 Restrições de operação . . . 270
14.3 Classificação quanto aos pulsos de entrada . . . 271
14.4 Circuitos *pulse-mode* . . . 272
14.4.1 Motivação . . . 272
14.4.2 Mudanças nas representações . . . 272
14.4.3 Exemplos de projeto . . . 274
14.5 Circuitos *ripple-clock* . . . 275
14.5.1 Motivação . . . 275
14.5.2 Operação . . . 275
14.5.3 Desvantagens . . . 275
14.5.4 Técnica de projeto . . . 275
14.5.5 Exemplo . . . 276
14.6 Circuitos *controlled-clock* . . . 276

15 Circuitos seqüenciais *level-mode* 279
15.1 Introdução . . . 279
15.2 Problemas comuns em circuitos *level-mode* . . . 280
15.3 Exemplo de análise de circuito *level-mode* . . . 281
15.4 Exemplo de projeto de circuito *level-mode* . . . 281
15.5 Problemas causados pela realimentação contínua . . . 282
15.5.1 Problemas causados pelo bloco de lógica combinacional . . . 282
15.5.2 Problema natural dos circuitos *level-mode* . . . 282
15.6 Solução para as corridas: atribuição de estados . . . 283
15.6.1 Definição do problema . . . 283
15.6.2 Possíveis soluções . . . 285

15.7 Solução para os perigos . . . 288
15.8 Valores das saídas em estados instáveis . . . 289

V Circuitos digitais programáveis 291

16 Circuitos programáveis 293
16.1 Introdução . . . 293
16.2 Circuitos configuráveis externamente . . . 294
16.2.1 Configuração de portas lógicas . . . 294
16.2.2 Configuração de funções lógicas com multiplexador . . . 295
16.3 Circuitos configuráveis internamente: PLDs . . . 298
16.4 Exercícios propostos . . . 299

VI Apêndices 301

A Exemplos de expressões equivalentes, empregando NAND e NOR 303
A.1 Introdução . . . 303
A.2 Relações com o operador NAND . . . 303
A.2.1 Operador NOT . . . 303
A.2.2 Operador AND . . . 303
A.2.3 Operador OR . . . 304
A.2.4 Operador XOR . . . 304
A.3 Relações com o operador NOR . . . 305
A.3.1 Operador NOT . . . 305
A.3.2 Operador AND . . . 305
A.3.3 Operador OR . . . 306
A.3.4 Operador XOR . . . 306

B Noções básicas sobre implementação de funções lógicas 309
B.1 Introdução . . . 309
B.2 Conceitos básicos . . . 309
B.3 Famílias lógicas . . . 310
B.4 Modelo de chaves . . . 310
B.4.1 Conceitos básicos . . . 310
B.4.2 Arranjos série e paralelo de chaves . . . 312
B.4.3 Modelo de chaves para a função NOT . . . 313
B.4.4 Modelo de chaves para a função NAND . . . 315
B.4.5 Modelo de chaves para a função NOR . . . 317
B.4.6 Modelo de chaves para arranjos AOI e OAI . . . 319
B.4.7 Modelo de chaves complementares genérico . . . 322
B.4.8 Saída em *Tri-State* (3S) . . . 323

C Tópicos sobre divisão de números inteiros 325
C.1 Algoritmo de divisão inteira . . . 325
C.2 Quociente . . . 325
C.3 Resto ou resíduo . . . 325
C.4 Congruência . . . 325
C.5 Relações úteis . . . 326

D Minimização de tabela de estados **327**
D.1 Introdução . . . 327
D.2 Tabelas de estados completamente especificadas . . . 328
D.2.1 Relações de equivalência . . . 328
D.2.2 Estados e circuitos equivalentes . . . 328
D.2.3 Determinação de classes de estados indistinguíveis . . . 329
D.2.4 Circuito de classes de equivalência . . . 329
D.3 Tabelas de estados não completamente especificadas . . . 330
D.3.1 Introdução . . . 330
D.3.2 Noções básicas de compatibilidade . . . 330
D.3.3 Formalização dos conceitos de compatibilidade e de cobertura . . . 331
D.3.4 Sistematização do processo de minimização . . . 332

E Linguagens de descrição de *hardware* **333**
E.1 Introdução . . . 333
E.2 Abordagem hierárquica . . . 333
E.3 Níveis de abstração . . . 334
E.4 Linguagens de descrição de *hardware* . . . 335

F Introdução à linguagem VHDL **337**
F.1 Histórico da linguagem VHDL . . . 337
F.2 VHDL como linguagem . . . 337
F.2.1 Considerações gerais . . . 337
F.2.2 Palavras reservadas . . . 338
F.2.3 Identificadores definidos pelo usuário . . . 338
F.2.4 Elementos sintáticos . . . 338
F.3 Conceitos básicos sobre o código VHDL . . . 340
F.3.1 Elementos básicos . . . 340
F.3.2 Tipos de execução . . . 340
F.3.3 Mecanismo genérico de simulação . . . 341
F.4 Estrutura do código VHDL . . . 342
F.4.1 Bibliotecas e pacotes . . . 342
F.4.2 Entidade . . . 343
F.4.3 Arquitetura . . . 343
F.5 Algumas regras sintáticas de VHDL . . . 344
F.5.1 Regras para biblioteca . . . 344
F.5.2 Regras para pacote . . . 344
F.5.3 Regras para entidade . . . 344
F.5.4 Regras para arquitetura . . . 345
F.5.5 Regras para processo . . . 345
F.5.6 Regras para componente . . . 346
F.6 Exemplos de declarações genéricas . . . 346
F.6.1 Exemplos de biblioteca e de pacote . . . 346
F.6.2 Exemplos de entidade . . . 346
F.6.3 Exemplos de arquitetura . . . 347
F.6.4 Exemplos de processo . . . 348

Referências bibliográficas **349**

Lista das Tabelas

2.1 Tabela Verdade das funções $f_1(A, B, C)$ e $f_2(A, B, C)$. 23

4.1 Exemplo de tabela verdade completamente especificada 37
4.2 Exemplo de tabela verdade não completamente especificada 37
4.3 Exemplo de tabela verdade simplificada 37
4.4 Tabela de funções lógicas de uma variável: $X_i = f_i(A)$, para $0 \leq i \leq 3$. 37
4.5 Tabela de operadores lógicos de uma variável. 37
4.6 Tabela de funções lógicas de duas variáveis: $X_i = f_i(A, B)$, para $0 \leq i \leq 15$. . . 38
4.7 Tabela de operadores lógicos de duas variáveis. 38
4.8 Exemplos de Tautologias ou Leis da Lógica. 40
4.9 Pares de proposições equivalentes, definindo os operadores OR, IMPLICA e XNOR, em função dos operadores NOT e AND. 40
4.10 Pares de proposições equivalentes, definindo os operadores AND, IMPLICA e XNOR, em função dos operadores NOT e OR. 40
4.11 Pares de proposições equivalentes, definindo os operadores AND, OR e XNOR, em função dos operadores NOT e IMPLICA. 40
4.12 Verificação da equivalência lógica $(p \rightarrow q) \equiv (\neg q \rightarrow \neg p)$. 41
4.13 Verificação da equivalência lógica $(p \vee q) \equiv \neg (\neg p \wedge \neg q)$. 41
4.14 Verificação da equivalência lógica $(p \wedge q) \equiv \neg (\neg p \vee \neg q)$. 41
4.15 Verificação da equivalência lógica $(A \uparrow B) \equiv (\neg A \vee \neg B)$. 42
4.16 Verificação da equivalência lógica $(A \downarrow B) \equiv (\neg A \wedge \neg B)$. 42
4.17 Funções lógicas de duas variáveis e suas expressões equivalentes, empregando os operadores AND, OR e NOT. 43
4.18 Tabela de funções canônicas (mintermos e maxtermos) para duas variáveis. . . 44
4.19 Exemplo de decomposição em funções canônicas (mintermos e maxtermos). . . 45
4.20 Tabela de funções canônicas (mintermos e maxtermos) para uma variável. . . . 45
4.21 Uso de operador lógico como elemento de controle. 47

5.1 Tabela de mapeamento: Cálculo Proposicional × Álgebra de Boole. 59
5.2 Tabela de mapeamento: Teoria de Conjuntos × Álgebra de Boole. 59
5.3 Resumo dos Postulados de Huntington para a estrutura algébrica de Boole. . . . 64
5.4 Resumo dos lemas para a estrutura algébrica de Boole. 64
5.5 Resumo dos teoremas para a estrutura algébrica de Boole. 65
5.6 Resumo da definição de uma estrutura algébrica de Boole particular. 65
5.7 Resumo das relações de isomorfismo. 65

6.1 Definição de mintermos para três variáveis (A,B,C). 71
6.2 Definição de maxtermos para três variáveis (A,B,C). 71
6.3 Exemplo de função X, proposta de decomposição de X em funções Y_k e associação das funções Y_k com os mintermos m_k. 74

6.4 Exemplo de função X, proposta de decomposição de X em funções Z_k e associação das funções Z_k com os maxtermos M_k. 75
6.5 Exemplo da obtenção do grupo AND-OR para a função XOR. 79
6.6 Exemplo da mudança de grupo para a função XOR. 79
6.7 Exemplo da obtenção do grupo OR-AND para a função XOR. 79

7.1 Tabela verdade genérica para funções de 3 variáveis. 89

8.1 Tabela verdade da Equação (8.1). 113
8.2 Relações de possibilidade de simplificação entre termos: funções de duas variáveis. 114
8.3 Relações de possibilidade de simplificação entre termos: funções de três variáveis. 114
8.4 Código Gray para 1 variável. 115
8.5 Código Gray para 2 variáveis. 115
8.6 Código Gray para 3 variáveis. 115
8.7 Tabela verdade da Equação (8.1), reorganizada. 116
8.8 Relações de possibilidade de simplificação entre termos: funções de duas variáveis. Tabela verdade reorganizada. 116
8.9 Relações de possibilidade de simplificação entre termos: funções de três variáveis. Tabela verdade reorganizada. 116
8.10 Tabela verdade para funções de 1 variável. 117
8.11 Tabela verdade para funções de 2 variáveis. 118
8.12 Tabela verdade para funções de 3 variáveis. 119
8.13 Tabela verdade para funções de 4 variáveis. 120
8.14 Tabela verdade relativa à Equação (8.4). 122
8.15 Tabela verdade de função incompletamente especificada. 126

9.1 Tabela de sinal-e-magnitude, para número inteiros, $b = 2$ e $N = 4$. 141
9.2 Tabela de complemento-a-1, para número inteiros, $b = 2$ e $N = 4$. 143
9.3 Tabela de complemento-a-2, para número inteiros, $b = 2$ e $N = 4$. 145
9.4 Tabela de sinal-e-magnitude, para números puramente fracionários, $b = 2$ e $N = 4$. 146
9.5 Tabela de complemento-a-1, para números puramente fracionários, $b = 2$ e $N = 4$. 147
9.6 Tabela de complemento-a-2, para números puramente fracionários, $b = 2$ e $N = 4$. 147
9.7 Alguns valores de níveis em um SNPC, em função do número de dígitos empregados, para as bases $b = 2$ e $b = 10$. 152
9.8 Alguns valores de resolução numérica em um SNPC, em função do número de dígitos empregados, para as bases $b = 2$ e $b = 10$. 153
9.9 Efeito do pequeno aumento da resolução numérica em um SNPC, em função do número de dígitos empregados, para a base $b = 2$. 153
9.10 Relação entre os códigos numéricos decimal, binário, octal e hexadecimal, para um código binário com 4 *bits*. 155
9.11 Relação entre os códigos numéricos decimal, binário, Gray, *One-hot* e Johnson, para um código binário com 4 *bits*. 155
9.12 Relação entre os dígitos decimais e os códigos BCD 8421, *Excess-3* e Mid-Gray-4. 156
9.13 Diferentes versões do código numérico BCD 631(-1). 156
9.14 Códigos numéricos BCD com mais de 4 *bits*: *2-out-of-5* e *Biquinary*. 157
9.15 Outros códigos numéricos BCD ponderados: 7421, 5421, 5311, 4221 e 2421. . . . 158
9.16 Exemplos de códigos numéricos BCD não ponderados. 158
9.17 Conjunto de representações possíveis com o padrão de ponto flutuante IEEE 754, onde: max $= (2^X - 1)$, X é o total de *bits* de E e NaN (*Not a Number*) indica que q não é um número. 159

9.18 Exemplos para uma representação numérica em ponto fixo, codificado em Sinal-e-Magnitude, para alguns fatores de escala M_{fix} diferentes. 161
9.19 Exemplo para uma representação numérica em ponto flutuante, sem valores reservados, com X=3, R=2 e V=1. 162
9.20 Forma 1 para representar a detecção de *overflow* na adição em complemento-a-2. 167
9.21 Forma 2 para representar a detecção de *overflow* na adição em complemento-a-2. 168
9.22 Tabela de conversão do Código 1 para o Código 2. 172

10.1 Relação entre os códigos numéricos decimal, binário e BCD 8421, para um código binário com 4 *bits*. 176
10.2 Tabela verdade do MUX 2x1. 179
10.3 Tabela verdade simplificada do MUX 2x1. 180
10.4 Tabela verdade simplificada do MUX 4x1. 181
10.5 Tabela verdade do DEMUX 2x1. 181
10.6 Tabela verdade do DEMUX 4x1. 183
10.7 Tabela verdade simplificada do codificador de prioridade com 2 entradas e 2 saídas. 185
10.8 Tabela verdade simplificada do codificador de prioridade com 3 entradas e 3 saídas. 186
10.9 Tabela verdade simplificada do codificador de prioridade com R entradas e R saídas. 186
10.10 Tabela verdade do bloco básico para o codificador de prioridade com R entradas e R saídas modular, com prioridade crescente. 187
10.11 Tabela Verdade do módulo básico de comparação. Sinais maior (G) e menor (L). 198
10.12 Mapa de Karnaugh para o sinal G_k do módulo básico de comparação. 198
10.13 Mapa de Karnaugh para o sinal L_k do módulo básico de comparação. 198
10.14 Tabela Verdade do módulo básico de comparação. Sinais igual (E) e maior (G). 199
10.15 Mapa de Karnaugh para o sinal G_k do módulo básico de comparação. 199
10.16 Mapa de Karnaugh para o sinal E_k do módulo básico de comparação. 199

11.1 Tipos de interações entre sinais dos tipos nível e pulso. 204
11.2 Excitação e saída em função de estado e entrada: forma expandida. 206
11.3 Estado e saída em função de estado e entrada: forma expandida. 206
11.4 Excitação e saída em função de estado e entrada: forma compacta. 206
11.5 Estado e saída em função de estado e entrada: forma compacta. 206

12.1 Transformações envolvendo *flip-flops* dos tipos JK, D, T_1 e T_2. 214
12.2 Definição dos tipos de comportamento apresentados pela saída de um *flip-flop*. . 215
12.3 Tabela resumo de funcionalidade-excitação-comportamento para os *flip-flops* SR, JK, D e T_2. 216
12.4 Tabela de mudanças de estado e de comportamento dos elementos de memória para um contador binário, crescente, de três *bits*. 218
12.5 Operação das estruturas de armazenamento estáticas e simétricas controladas por meio de portas lógicas NOR e NAND. 223
12.6 Diferentes nomenclaturas para *flip-flops*. 229

13.1 *Transition table* do exemplo de análise de uma Máquina de Mealy. 237
13.2 Possível Tabela de Atribuição de Estados para o exemplo de análise de uma Máquina de Mealy. 237
13.3 *State table* do exemplo de análise de uma Máquina de Mealy. 238
13.4 *Transition table* do exemplo de análise de uma Máquina de Moore. 238

13.5 Possível Tabela de Atribuição de Estados para o exemplo de análise de uma Máquina de Moore. . . . 239
13.6 *State table* do exemplo de análise de uma Máquina de Moore. . . . 239
13.7 *State table* do exemplo de projeto de uma Máquina de Moore - Contador. . . . 243
13.8 Possível Tabela de Atribuição de Estados para o exemplo de projeto de uma Máquina de Moore - Contador. . . . 243
13.9 *Transition table* do exemplo de projeto de uma Máquina de Moore - Contador. . . . 243
13.10 Tabela de excitação para o *flip-flop* JK. . . . 244
13.11 Tabela verdade de cada variável de excitação, para o exemplo de projeto com *flip-flop* JK, de uma Máquina de Moore - Contador. . . . 245
13.12 Tabela de excitação para o *flip-flop* D. . . . 246
13.13 Tabela verdade de cada variável de excitação, para o exemplo de projeto, com *flip-flop* D, de uma Máquina de Moore - Contador. . . . 246
13.14 *State table* do exemplo de projeto de uma Máquina de Mealy - Contador. . . . 248
13.15 Possível Tabela de Atribuição de Estados para o exemplo de projeto de uma Máquina de Mealy - Contador. . . . 248
13.16 *Transition table* do exemplo de projeto de uma Máquina de Mealy - Contador. . . . 248
13.17 Tabela de excitação para o *flip-flop* JK. . . . 249
13.18 Tabela verdade de cada variável de excitação, para o exemplo de projeto com *flip-flop* JK, de uma Máquina de Mealy - Contador. . . . 250
13.19 Tabela de excitação para o *flip-flop* D. . . . 251
13.20 Tabela verdade de cada variável de excitação, para o exemplo de projeto, com *flip-flop* D, de uma Máquina de Mealy - Contador. . . . 251
13.21 Número de atribuições de estados efetivamente diferentes. . . . 259

15.1 Atribuição de estados universal, usando shared-row, para tabelas de 3 estados. . . . 286
15.2 Atribuição de estados universal, usando multiple-row, para tabelas de 4 estados. 287
15.3 Atribuição de estados universal, usando shared-row, para tabelas de 5 a 8 estados. 287
15.4 Atribuição de estados universal, usando shared-row, para tabelas de 9 a 12 estados. 287
15.5 Atribuição de estados padrão, usando shared-row, para tabelas de 5 estados. . . . 287

16.1 Classificação de circuitos digitais de acordo com a função realizada. . . . 294
16.2 Comportamento funcional dos dispositivos NC7SZ57 e NC7SZ58. . . . 295
16.3 Opções de configuração dos dispositivos NC7SZ57 e NC7SZ58. . . . 295

B.1 Tipos básicos de associação entre valores de tensão e valores lógicos binários. . . . 310
B.2 Comportamento elétrico do modelo de chaves NOT. . . . 314
B.3 Comportamento lógico do modelo de chaves NOT. . . . 314
B.4 Comportamento elétrico do modelo de chaves NAND. . . . 315
B.5 Comportamento lógico do modelo de chaves NAND. . . . 316
B.6 Comportamento elétrico do modelo de chaves NOR. . . . 317
B.7 Comportamento lógico do modelo de chaves NOR. . . . 318
B.8 Comparação do comportamento dos blocos de chaves N e P no modelo de chaves da função lógica NOT. (a) Chave-N. (b) Chave-P. (c) Arranjo complementar. . . . 322
B.9 Comparação do comportamento dos blocos de chaves N e P no modelo de chaves da função lógica NAND. (a) Chave-N. (b) Chave-P. (c) Arranjo complementar. 322
B.10 Comparação do comportamento dos blocos de chaves N e P no modelo de chaves da função lógica NOR. (a) Chave-N. (b) Chave-P. (c) Arranjo complementar. . . . 323

E.1 Lista de fabricantes, produtos e funções, que lidam com HDL. . . . 336

F.1 Operadores de VHDL. 340

Lista das Figuras

2.1 Diagrama de caixa preta de um sistema. 9
2.2 Exemplo de discretização uniforme, para dois valores de resolução diferentes, dentro de uma faixa unitária contínua. 10
2.3 Sistema de comunicação genérico. 14
2.4 Sistema de comunicação com sistema de transmissão genérico. 14
2.5 Sistema de comunicação com sistema de transmissão digital genérico. 15
2.6 Cadeia de pré-processamento analógico. 18
2.7 Processador de Sinal Digital. 18
2.8 Cadeia de pós-processamento analógico. 18

4.1 Exemplos de tabela verdade para uma função lógica de duas variáveis, com duas ordenações de linhas diferentes. 36
4.2 Uso de operador lógico como elemento de controle: simbologia genérica. 47
4.3 Blocos funcionais fundamentais, associados aos operadores lógicos básicos (IEEE). 49
4.4 Blocos funcionais fundamentais, associados aos operadores lógicos básicos (IEC). 49
4.5 Manipulação algébrica de blocos: exemplo 1. 50
4.6 Manipulação algébrica de blocos: exemplo 2. 51

5.1 Mapeamento entre uma função genérica de duas variáveis e um Diagrama de Venn. 63

7.1 Eliminações de 1 literal em combinações de 2 mintermos. 89
7.2 Eliminações de 2 literais em combinações 4 de mintermos. 90
7.3 Eliminações de 1 literal em combinações de 2 maxtermos. 90
7.4 Eliminações de 2 literais em combinações 4 de maxtermos. 90

8.1 Exemplos de mapas de Karnaugh para funções de 1 variável. 117
8.2 Exemplos de mapas de Karnaugh para funções de 2 variáveis. 118
8.3 Exemplos de mapas de Karnaugh para funções de 3 variáveis. 119
8.4 Exemplos de mapas de Karnaugh para funções de 4 variáveis. 120
8.5 Mapa de Karnaugh relativo à Equação (8.1), considerando apenas os mintermos. 121
8.6 Mapa de Karnaugh relativo à Equação (8.1), considerando apenas os maxtermos. 121
8.7 Mapa de Karnaugh relativo à Equação (8.4). 122
8.8 Mapa de Karnaugh relativo à Equação (8.9). 123
8.9 Mapa de Karnaugh relativo à Equação (8.10). 123
8.10 Mapa de Karnaugh relativo à Equação (8.11). 124
8.11 Mapa de Karnaugh com múltiplas formas mínimas. 125
8.12 Mapa de Karnaugh com ciclo. 125
8.13 Mapa de Karnaugh da Tabela 8.15. 126
8.14 Mapa de Karnaugh dos mintermos da Tabela 8.15. 126

8.15 Mapa de Karnaugh dos maxtermos da Tabela 8.15. 127

9.1 Representação de quantidades $q < b$, para $b = 3$. 133
9.2 Representação de quantidades $q \geq b$, para $b = 3$, com ambigüidade. 134
9.3 Representação de quantidades $q \geq b$, para $b = 3$, com eliminação da ambigüidade através da justaposição dos dígitos. 134
9.4 Uso repetido da técnica de justaposição de dígitos para representação de quantidades $q \geq b$, para $b = 3$, sem ambigüidade. 134
9.5 Mapeamento decimal-binário para números não negativos e $N = 4$, sem sinal. . . 138
9.6 Mapeamento decimal-binário para números não negativos e $N = 4$, com sinal. . 138
9.7 Mapeamento decimal-binário para números negativos e $N = 4$, com codificação Sinal-e-Magnitude. 139
9.8 Mapeamento decimal-binário para números negativos e $N = 4$, com codificação Complemento-a-1. 139
9.9 Mapeamento decimal-binário para números negativos e $N = 4$, com codificação Complemento-a-2. 139
9.10 Comparação dos mapeamentos decimal-binário para números negativos e $N = 4$. 140
9.11 Tabelas de operações entre dígitos para $b = 2$: (a) adição e (b) multiplicação. . . 163
9.12 Tabelas de operações entre dígitos para $b = 3$: (a) adição e (b) multiplicação. . . 163
9.13 Tabelas de operações entre dígitos para $b = 4$: (a) adição e (b) multiplicação. . . 163
9.14 Tabelas que definem a adição entre dois dígitos binários (A e B): (a) adição completa, (b) "vai-um" (*carry out* - C_o) e (c) soma (S). 168
9.15 Tabelas que definem a adição entre três dígitos binários (A, B e C_i): (a) adição completa, (b) "vai-um" (*carry out* - C_o) e (c) soma (S). 169
9.16 Tabelas que definem três funções úteis na implementação de somadores binários com algoritmos complexos: (a) *Generate* (G), (b) *Propagate* (P) e (c) *Kill* (K). . 170

11.1 Modelo genérico para circuitos seqüenciais. 205
11.2 Exemplo de máquina de Mealy. 207
11.3 Exemplo de máquina de Moore. 208
11.4 Modelo genérico para circuitos seqüenciais *clock-mode*. 209
11.5 Modelo genérico para circuitos seqüenciais *pulsed*. 209
11.6 Modelo genérico para circuitos seqüenciais *level-mode*. 210

12.1 Tabelas de operação básica para os *flip-flops* SR, JK, D e T_2. 213
12.2 Mapas de excitação para os *flip-flops* SR, JK, D e T_2. 215
12.3 Tipos de comportamento e respectivas excitações para os *flip-flops* SR, JK, D e T_2. 216
12.4 Mapas-K de transição para os elementos de memória de um contador binário, crescente, de três *bits*. 218
12.5 Mapas-K de excitação para os *flip-flops* JK de um contador binário, crescente, de três *bits*. 219
12.6 Estrutura de armazenamento estática e simétrica, não controlável. 220
12.7 Estruturas de armazenamento estáticas e simétricas, controláveis por chaves. . . 221
12.8 Uso de portas lógicas NOR na implementação de controle em uma estrutura de armazenamento estática e simétrica. 222
12.9 Uso de portas lógicas NAND na implementação de controle em uma estrutura de armazenamento estática e simétrica. 222
12.10 Exemplo de implementação de *flip-flop* SR do tipo *clocked* elementar, usando portas lógicas NOR. 224

12.11Exemplo de implementação de *flip-flop* SR do tipo *clocked* elementar, usando portas lógicas NAND. . . . 224
12.12Exemplo de implementação de *flip-flop* D do tipo *clocked* elementar, com base em um *flip-flop* SR. . . . 224
12.13Técnica de *pipelining*: (a) Bloco funcional original e (b) Bloco com *pipelining*. . . . 226
12.14Exemplo de implementação de *flip-flop* D do tipo *master-slave*, com base em *flip-flops* SR. . . . 226
12.15Exemplo de implementação de *flip-flop* JK, a partir de *flip-flop* SR *unclocked*, com problema de oscilação. . . . 227
12.16Exemplo de implementação de *flip-flop* JK, a partir de *flip-flop* SR *clocked*, com problema de oscilação. . . . 227
12.17Exemplo de implementação de *flip-flop* JK, a partir de *flip-flop* SR *clocked*, sem problema de oscilação, devido ao uso de estrutura *master-slave*. . . . 227
12.18Exemplo 1 de implementação de *flip-flop* JK do tipo *master-slave*. . . . 228
12.19Exemplo 2 de implementação de *flip-flop* JK do tipo *master-slave*. . . . 228

13.1 Modelo genérico para circuitos seqüenciais *clock-mode*. . . . 231
13.2 Modelo genérico para circuitos com memória finita. . . . 235
13.3 Modelo genérico para circuitos com memória finita de entrada. . . . 235
13.4 Modelo genérico para circuitos com memória finita de saída. . . . 236
13.5 Fluxos de projeto para circuitos seqüenciais *clock-mode*: (a) Fluxo genérico, (b) Caso particular de Máquina de Moore e (c) Máquina de Memória Finita (MMF). . . . 240
13.6 Diagrama de estados do exemplo de projeto de uma Máquina de Moore - Contador. 242
13.7 Mapas de Karnaugh das variáveis de saída para o exemplo de projeto de uma Máquina de Moore - Contador. . . . 244
13.8 Mapas de Karnaugh das variáveis de excitação, para o exemplo de projeto com *flip-flop* JK, de uma Máquina de Moore - Contador. . . . 245
13.9 Mapas de Karnaugh das variáveis de excitação, para o exemplo de projeto com *flip-flop* D, de uma Máquina de Moore - Contador. . . . 246
13.10Diagrama de estados do exemplo de projeto de uma Máquina de Mealy - Contador. 247
13.11Mapas de Karnaugh das variáveis de saída para o exemplo de projeto de uma Máquina de Moore - Contador. . . . 249
13.12Mapas de Karnaugh das variáveis de excitação, para o exemplo de projeto com *flip-flop* JK, de uma Máquina de Mealy - Contador. . . . 250
13.13Mapas de Karnaugh das variáveis de excitação, para o exemplo de projeto com *flip-flop* D, de uma Máquina de Mealy - Contador. . . . 251
13.14Eliminação de estados redundantes através da inspeção da tabela de estados. . . . 253
13.15Exemplo de minimização positiva em um passo. . . . 255
13.16Exemplo de minimização negativa em um passo. . . . 255
13.17Exemplo de minimização positiva em mais de um passo. . . . 256
13.18Tabela de implicação genérica do método de Paul-Unger. . . . 258
13.19Célula genérica da tabela do método de Paul-Unger. . . . 258
13.20Análise de minimização para as equações de excitação e de saída: mapa de Karnaugh simbólico. . . . 261
13.21Análise de minimização para as equações de excitação e de saída: tabela de atribuição de estados hipotética. . . . 262
13.22Análise de minimização para as equações de excitação: casos de estados atuais com mesmo próximo estado. . . . 263

13.23Análise de minimização para as equações de excitação: casos de estado atual com próximos estados diferentes. . . . 264
13.24Análise de minimização para as equações de saída. . . . 265
13.25Ilustração das regras de Armstrong-Humphrey. . . . 266

14.1 Modelo genérico para circuitos seqüenciais *pulsed*. . . . 269
14.2 Equivalência de notações para mapa de Karnaugh utilizado na síntese de variáveis pulsadas. . . . 273
14.3 Tabelas de estados para circuitos *pulse-mode* Mealy e Moore. . . . 273
14.4 Mapas de Karnaugh para síntese de variáveis pulsadas, considerando-se duas entradas pulsadas: (a) Mapa completo e (b) Mapa simplificado. . . . 274
14.5 Mapas de Karnaugh para síntese de variáveis pulsadas, considerando-se três entradas pulsadas: (a) Mapa completo e (b) Mapa simplificado. . . . 274
14.6 Exemplo 1 de controle de sinal de *clock*. . . . 276
14.7 Exemplo 2 de controle de sinal de *clock*. . . . 276
14.8 Modelo genérico para circuitos seqüenciais *controlled-clock*. . . . 278

15.1 Modelo genérico para circuitos seqüenciais *level-mode*. . . . 279
15.2 Padrões de identificação de perigo essencial em tabelas de fluxo. . . . 283
15.3 Quadro resumo das mudanças de estado nos circuitos seqüenciais *level-mode*, operando em modo fundamental. . . . 284

16.1 Mapas de Karnaugh da função exemplo, para implementação usando MUX. . . . 297

B.1 Chaves simples ou *Single-Pole Single-Throw* (SPST). (a) Chave-N. (b) Chave-P. 311
B.2 Modelo comportamental para a relação de complementariedade no acionamento, entre as chaves N e P. . . . 311
B.3 Arranjo série de chaves simples. (a) Chave-N. (b) Chave-P. . . . 312
B.4 Arranjo paralelo de chaves simples. (a) Chave-N. (b) Chave-P. . . . 312
B.5 Modelo de chaves simples para uma implementação da função lógica NOT, onde: (a) chave N e (b) chave P. . . . 313
B.6 Modelo de chaves complementares para uma implementação da função lógica NOT. . . . 313
B.7 Modelo de chaves simples para uma implementação da função lógica NAND, onde: (a) chave N e (b) chave P. . . . 315
B.8 Modelo de chaves complementares para uma implementação da função lógica NAND. . . . 316
B.9 Modelo de chaves simples para uma implementação da função lógica NOR, onde: (a) chave N e (b) chave P. . . . 317
B.10 Modelo de chaves complementares para uma implementação da função lógica NOR. . . . 318
B.11 Modelo de chaves complementares para uma implementação do arranjo AOI definido por $f(A,B,C,D) = \neg(\ (A\ \wedge\ B)\ \vee\ (C\ \wedge\ D)\ )$. . . . 320
B.12 Modelo de chaves complementares para uma implementação do arranjo OAI definido por $f(A,B,C,D) = \neg(\ (A\ \vee\ B)\ \wedge\ (C\ \vee\ D)\ )$. . . . 321
B.13 Modelo de chaves complementares para uma implementação de função lógica NOT com saída em *Tri-State* (3S). . . . 324

F.1 Palavras reservadas de VHDL. . . . 339
F.2 Símbolos especiais de VHDL. . . . 339

Parte I

Apresentação

Capítulo 1

Descrição do documento

1.1 Introdução

- Este é um documento em constante atualização.
- Ele consta de tópicos desenvolvidos em sala de aula.
- Na preparação das aulas são utilizados os seguintes livros:
  - Livros indicados pela ementa da disciplina: [IC08], [Tau82].
  - Livros indicados pelo professor: [HP81], [Rhy73], [TWM07], [Uye02].
- Este documento aborda os seguintes assuntos:
  - Circuitos digitais combinacionais.
  - Circuitos digitais seqüenciais.
  - Circuitos digitais programáveis.
- Um resumo do conteúdo programático é apresentado a seguir, assumindo um período letivo de 15 semanas, com uma carga horária de 60 horas.

1.2 Circuitos digitais combinacionais <28 horas>

- Conceitos básicos: busca contextualizar o tópico no âmbito do curso de graduação e apresentar os conceitos que serão necessários ao longo do texto. <2 horas>
- Funções lógicas: que define as bases para a representação de informações não numéricas em circuitos digitais tradicionais. <6 horas>
- Álgebra de Boole: que apresenta um formalismo matemático para a estrutura algébrica da lógica implementada em circuitos digitais tradicionais. <2 horas>
- Formas padrões para representação de expressões booleanas: que define formas de expressões booleanas adequadas a um processo sistemático de simplificação das mesmas. <4 horas>
- Simplificação algébrica de expressões booleanas: que ilustra um processo algébrico para a simplificação sistemática de expressões booleanas. <4 horas>

- Mapa de Karnaugh: que apresenta uma ferramenta tabular para a simplificação sistemática de expressões booleanas. <4 horas>
- Sistemas de numeração: que define as bases para a representação de quantidades numéricas em circuitos digitais. <6 horas>
- Circuitos combinacionais básicos: que apresenta os blocos funcionais combinacionais básicos utilizados em sistemas digitais. <Distribuídos ao longo da apresentação do conteúdo e exercitados na forma de trabalhos>

1.3 Circuitos digitais seqüenciais <16 horas>

- Conceitos básicos: busca contextualizar o tópico no âmbito do curso de graduação e apresentar os conceitos que serão necessários ao longo do texto. <2 horas>
- Elementos básicos de armazenamento: apresenta os elementos de armazenamento utilizados nos circuitos seqüenciais abordados neste texto. <6 horas>
- Circuitos seqüenciais do tipo *clock-mode*: define as características dessa classe de circuitos e aborda os procedimentos, as técnicas e as ferramentas de análise e de projeto para circuitos da classe. <8 horas>
- Circuitos seqüenciais do tipo *pulsed*: define as características dessa classe de circuitos e aborda os procedimentos, as técnicas e as ferramentas de análise e de projeto para circuitos da classe. <Apenas contextualizado em Conceitos básicos>
- Circuitos seqüenciais do tipo *level-mode*: define as características dessa classe de circuitos e aborda os procedimentos, as técnicas e as ferramentas de análise e de projeto para circuitos da classe. <Apenas contextualizado em Conceitos básicos>

1.4 Circuitos digitais programáveis <8 horas>

- Circuitos programáveis: apresenta uma breve descrição sobre circuitos digitais que podem ser configurados para cumprir diferentes funções, combinacionais e/ou seqüenciais. <2 horas>
- Noções básicas sobre Linguagem de Descrição de *Hardware* (*Hardware Description Language* ou HDL). <2 horas>
- Noções básicas sobre as HDLs comumente utilizadas (VHDL e Verilog). <4 horas>

1.5 Aplicações

- Aplicações: exemplos de aplicações são distribuídos ao longo da apresentação do conteúdo e exercitados na forma de trabalhos.

Parte II

Conceitos básicos

Capítulo 2

Conceitos básicos

Nesse capítulo, são brevemente abordados alguns itens genéricos sobre modelagem matemática, teoria de sistemas, sistemas de comunicação, processamento de sinais e implementações analógicas e digitais, porém sem o rigor matemático exigido por uma literatura especializada no assunto. Primeiramente, a modelagem matemática é brevemente citada, bem como são apresentadas uma definição e algumas classificações sobre sinais e sistemas. Para auxiliar em tais classificações, a discretização de sinais é brevemente apresentada, bem como os conceitos de estado e de variáveis de estado são abordados. Em seguida, é apresentada uma visão geral dos sistemas de comunicação, do processamento de sinais e da arquitetura de sistemas de processamento digital. Finalmente, é apresentada uma comparação entre as implementações analógicas e digitais.

2.1 Modelagem matemática

Na solução de um problema (físico ou matemático) é comumente empregado o processo sistemático descrito a seguir. Inicialmente, o problema original é mapeado em um sistema matemático equivalente. Em seguida, o problema matemático equivalente é resolvido. Por fim, é realizado um mapeamento do resultado matemático em uma solução para o problema original. As motivações mais comuns para o emprego desta metodologia são as seguintes: simplificar a formulação do problema original, buscando simplificar a sua solução, ou mapear o problema original em um problema matemático com soluções já conhecidas.

2.2 Teoria de sistemas

O ramo matemático da Teoria de Sistemas trata do estudo de elementos abstratos denominados de Sinais e de Sistemas. Ela representa um poderoso sistema formal de modelagem para os mais diversos tipos de sistemas (físicos ou matemáticos).

Tanto na análise quanto na síntese (ou projeto) de sistemas físicos, é comum que, em uma primeira etapa, eles sejam matematicamente modelados. Em seguida, o trabalho em questão é realizado sobre o modelo matemático adotado. A Teoria de Sistemas é largamente empregada nesses casos.

2.3 Definição de sinal

Em relação aos sinais, podem-se destacar as seguintes definições e considerações iniciais:

- Sinal: entidade que carrega informação.
- Visão matemática de sinal: variável funcionalmente dependente de uma ou mais variáveis independentes. Ex.: $y = f(x)$, $z = f(x, y)$, $w = f(x, y, z)$.
- Visão física de sinal: grandeza física.
- Tipos de sinais de acordo com o número de variáveis independentes:
  - Unidimensional. Ex.: áudio $= f(t)$.
  - Bidimensional. Ex.: imagem $= f(x, y)$.
  - Tridimensional. Ex.: vídeo $= f(x, y, t)$.
  - Multidimensional. Ex.: tomografia/sismologia $= f(v_1(t), v_2(t), \cdots, v_V(t), t)$.

2.4 Definição de sistema

Entre as inúmeras definições de sistemas existentes, pode-se destacar a seguinte:

> **"Um sistema é um conjunto de elementos que interagem entre si, a fim de realizar uma determinada tarefa."**

Notam-se, na definição acima, três partes: os elementos, a interação e o objetivo.

Primeiramente, a existência do sistema está vinculada à existência de uma função a ser realizada. Porém, para alcançar o objetivo desejado, não basta apenas que se reunam elementos. Pelo contrário, eles devem, sobretudo, apresentar alguma forma de interação. E, para que os elementos interajam entre si e com o exterior, alguma forma de conexão interna e externa deve existir. Portanto, a interação carrega três significados: a existência de uma informação a ser trocada, a troca de informação em si e a presença de conexões que possibilitem a troca.

Em resumo, um sistema, como definido acima, pode ser caracterizado pelos seguintes itens:

- **Elementos**: que interagem, trocando informação, para cumprir o objetivo do sistema.
- **Conexão**: que possibilita a comunicação entre os elementos.
- **Variáveis**: que armazenam as informações manipuladas pelos elementos.
- **Função**: que é razão pela qual o sistema deve ser implementado.

Uma descrição pictórica de um sistema é mostrada na Figura 2.1. Tal descrição é definida como Diagrama de Caixa Preta (*Black Box Diagram*), uma vez que não apresenta qualquer descrição interna do sistema. Nesse caso, definem-se apenas as relações entre as variáveis externas do sistema (entradas e saídas).

Figura 2.1: Diagrama de caixa preta de um sistema.

Os sistemas físicos elétricos costumam ser definidos da seguinte forma:

- **Variáveis**: usualmente a informação é codificada nas grandezas físicas tensão e corrente. A tensão é associada ao acúmulo de carga elétrica ou à energia potencial. A corrente, por sua vez, é associada ao movimento de carga elétrica ou à energia cinética. A associação de ambas permite definir potência e energia.

- **Elementos**: a fim de modelar as fontes físicas de energia elétrica e os efeitos físicos de resistência, capacitância e indutância, são utilizados os respectivos elementos ideais de circuito: fontes independentes, fontes dependentes, resistor, capacitor, auto-indutância e indutância mútua. A forma como cada elemento manipula as variáveis do circuito é descrita na equação de definição de cada elemento.

- **Conexão**: em circuitos de baixa freqüência, são utilizadas as leis de tensão e de corrente de *Kirchoff* para descrever as conexões entre os elementos. A fim de sistematizar a análise e o projeto de circuitos, é comum que se utilize a teoria matemática de grafos para descrever matematicamente os circuitos e, portanto, a conexão existente nos mesmos.

Os sistemas matemáticos digitais costumam ser definidos da seguinte forma:

- **Variáveis**: a informação mais fundamental é codificada em números, os quais são representados em um determinado Sistema de Numeração, com um número finito de símbolos. Tipicamente, é utilizado o Sistema de Numeração Posicional Convencional (SNPC), com base $b = 2$ e codificação em complemento a 2.

- **Elementos**: os elementos mais fundamentais são as denominadas portas lógicas, que implementam as funções básicas da lógica clássica. A partir delas, são construídos todos os demais elementos, inclusive os elementos armazenadores de informação.

- **Conexão**: são utilizadas equações lógicas para descrever as conexões entre os elementos.

2.5 Discretização de sinais

2.5.1 Definição e classificações

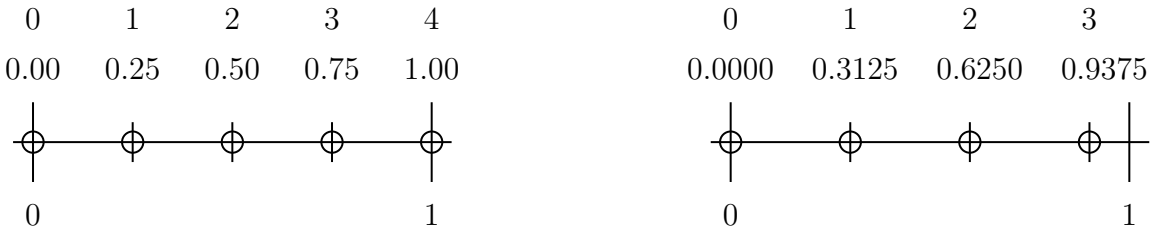
A discretização pode ser definida como a seleção de alguns valores individuais dentro de uma extensa faixa de valores (contínua ou já discretizada).

De acordo com a forma segundo a qual a seleção de cada valor é realizada, a discretização pode ser classificada como: uniforme (ou periódica) e não uniforme (ou não periódica). Na discretização uniforme, existe uma regularidade (periódica) na seleção de cada valor. Nesse caso, há um intervalo único, fixo e mínimo, que representa um período fundamental, segundo o qual cada valor é selecionado. Esse intervalo fundamental é denominado de resolução da discretização. Em outras palavras, dentro da faixa de valores considerada, são selecionados apenas aqueles valores que são múltiplos inteiros da resolução, ignorando-se os demais valores.

Por sua vez, na discretização não uniforme (ou não periódica), não há um período fundamental na seleção de cada valor individual. Nela, a seleção pode ser realizada nos mais diversos padrões.

Na discretização uniforme, os valores individuais são múltiplos inteiros do valor da resolução ($v = n\, v_{res}; n \in \mathbb{Z}$). Caso seja realizada uma normalização sobre o conjunto de valores discretos, empregando-se o valor da resolução, será obtido um novo conjunto de valores, que serão inteiros e adimensionais ($v_{norm} = v/v_{res} = n$). Por essa razão, costuma-se dizer que a discretização uniforme pode transformar uma faixa de valores em uma grade (*grid*) de valores inteiros e adimensionais. Nesses casos, é importante não perder a informação sobre o valor da resolução.

A Figura 2.2 ilustra graficamente o processo de discretização uniforme, para dois valores de resolução diferentes, dentro de uma faixa unitária contínua. Na Figura 2.2.(a), é obtido um número inteiro de intervalos na discretização da faixa considerada. Na Figura 2.2.(b), isso não é possível, fazendo com que a parte final da faixa encontre-se dentro de um intervalo de valores não selecionados. Em ambos os casos, dados o comprimento total L e a resolução R, o número de segmentos S de comprimento R é dado por $S = int\left(\frac{L}{R}\right)$, onde $int(x)$ é a parte inteira de x. Por sua vez, a quantidade de pontos P é dada por $P = (S + 1)$.



(a) Número inteiro de intervalos.

(b) Número não inteiro de intervalos.

Figura 2.2: Exemplo de discretização uniforme, para dois valores de resolução diferentes, dentro de uma faixa unitária contínua.

Em uma relação funcional do tipo $y = f(x_1, x_2, x_3, \ldots)$, definem-se y como variável dependente e x_k como variáveis independentes. O processo de discretização das variáveis independentes x_k é denominado de amostragem. Por sua vez, o processo de discretização da variável dependente y é denominado de quantização. Isso é brevemente descrito a seguir.

2.5.2 Amostragem

Dada a função $y(x)$, o processo de amostragem da variável dependente y envolve as seguintes etapas: a discretização de todos os possíveis valores da variável independente x e a seleção dos valores de y em tais valores discretos de x.

Comumente, é adotada a seguinte convenção: no caso de uma função contínua $y(x)$, se a amostragem for efetuada em um ponto x_d onde ocorre uma eventual descontinuidade, o valor da amostra deverá ser $y(x_d) = \lim_{\epsilon \to 0} y(x_d + \epsilon)$, $\epsilon > 0$.

É importante observar que a amostragem realiza uma conexão entre a função no seu domínio original e a nova função no domínio discreto. Para que a função no seu domínio original tenha uma relação biunívoca com a nova função no domínio discreto, alguns requisitos devem ser atendidos. Tais requisitos serão discutidos posteriormente.

2.5.3 Amostragem uniforme

Considerando a função $y(x)$, onde x é uma variável qualquer, a amostragem é denominada de uniforme (ou periódica) quando cada valor discreto de x é obtido de acordo com um intervalo periódico de discretização. Nesse caso, o intervalo mínimo (período fundamental) é a resolução da amostragem x_{res}. Desse modo, obtém-se a amostragem uniforme $y(n\ x_{res})$ da função $y(x)$.

Os valores discretos $x_D = n\ x_{res}$ ainda podem ser normalizados pela resolução, gerando um novo conjunto de valores $x_{Dnorm} = x_D/x_{res} = n$, que serão inteiros e adimensionais. Com isso, obtém-se uma seqüência de valores indexados $y[n]$ a partir da amostragem uniforme $y(n\ x_{res})$ da função $y(x)$.

A amostragem pode ser matematicamente modelada por um processo de multiplicação de uma função $y(x)$ por um trem de impulsos unitários $\delta_R(x)$, com período igual à resolução x_{res}, seguida do cálculo da área (por cálculo integral) de cada impulso resultante.

Embora a variável t seja comumente associada ao tempo, ela também pode ser considerada uma variável genérica. Assim, independentemente do significado de t, na amostragem uniforme (ou periódica) de uma função $y(t)$, a resolução t_{res} é denominada de intervalo mínimo ou período fundamental de amostragem T_S (*sampling interval* ou *sampling period*). Da mesma forma, pode ser definida uma taxa ou freqüência de amostragem $F_S = \frac{1}{T_S}$ (*sampling rate* ou *sampling frequency*).

A quantidade de pontos armazenados N_{rec}, considerando-se uma amostragem uniforme, com intervalo T_S, durante uma faixa de tempo T_{rec}, é dada por

$$N_{rec} = \left[int\left(\frac{T_{rec}}{T_S}\right) + 1 \right] \text{ amostras,}$$

onde $int(x)$ é a parte inteira de x.

A Figura 2.2 pode ser usada para ilustrar graficamente o cálculo de N_{rec}. Na Figura 2.2.(a), podem ser considerados $T_{rec} = 1$ e $T_S = 0.25$, o que fornece

$$N_{rec} = int\left(\frac{T_{rec}}{T_S}\right) + 1 = int\left(\frac{1}{0.25}\right) + 1 = int\,(4.0) + 1 = (4) + 1 = 5 \text{ amostras.}$$

Na Figura 2.2.(b), podem ser considerados $T_{rec} = 1$ e $T_S = 0.3125$, o que fornece

$$N_{rec} = int\left(\frac{T_{rec}}{T_S}\right) + 1 = int\left(\frac{1}{0.3125}\right) + 1 = int\,(3.2) + 1 = (3) + 1 = 4 \text{ amostras.}$$

2.5.4 Quantização

Dada a função $y(x)$, o processo de quantização $Q\{\cdot\}$ da variável dependente y envolve as seguintes etapas: a discretização da faixa de possíveis valores ocupados por y e a aproximação de um determinado valor de y para um dos dois valores discretos mais próximos, que são o valor imediatamente inferior $Q_{inf}\{y\}$ e o valor imediatamente superior $Q_{sup}\{y\}$ em relação ao valor original de y. A representação de y por $Q_{inf}\{y\}$ ou $Q_{sup}\{y\}$ vai depender da técnica de aproximação utilizada.

Uma vez que a quantização substitui o valor real de y por um valor aproximado $y_Q = Q\{y\}$, ela sempre vai apresentar um denominado erro de quantização $e_Q = (y - y_Q)$, definido pela técnica de aproximação utilizada.

As técnicas de aproximação mais comumente utilizadas são o truncamento e o arredondamento.

2.5.5 Quantização uniforme

A quantização $Q\{\cdot\}$ é denominada de uniforme quando cada valor quantizado $y_Q = Q\{y\}$ é obtido de acordo com um intervalo periódico de discretização. Nesse caso, o intervalo mínimo (período fundamental) é a resolução da quantização Q_{res}.

Os valores quantizados $y_Q = m\ Q_{res}$ ainda podem ser normalizados pela resolução, gerando um novo conjunto de valores $y_{Qnorm} = y_Q/Q_{res} = m$, que serão inteiros e adimensionais.

2.5.6 *Sampling rate*, *bit depth* e *bit rate*

Os termos *sampling rate*, *bit depth* e *bit rate* representam parâmetros associados com funções onde tanto as variáveis independentes quanto a variável dependente são discretizadas, tal como a seqüência $y_Q[n]$.

Assumindo que $y_Q[n]$ é uma seqüência indexada, obtida por meio da amostragem uniforme $y(n\ t_{res})$ da função $y(t)$, o parâmetro *sampling rate* ou *sampling frequency* (taxa ou freqüência de amostragem) $F_S = \frac{1}{T_S}$ é associado à resolução da amostragem uniforme $t_{res} = T_S$, que é o *sampling interval* ou *sampling period* (intervalo ou período de amostragem). No SI [Wik], a unidade de medida de T_S é o segundo (s) e de F_S é o hertz (Hz), onde 1 Hz = 1 ciclo/s. Por exemplo, um dos padrões comerciais usados para amostragem de um sinal de áudio é uma *sampling rate* de $F_S = 44,1$ kHz.

Assumindo que $y_Q[n]$ é uma seqüência indexada, obtida por meio da quantização uniforme $Q\{\cdot\}$ da seqüência indexada $y[n]$, o parâmetro *bit depth* (profundidade de *bits*) D_{bits} é associado com a quantidade de símbolos binários (*binary digits* ou *bits*) usados na representação de cada valor de $y_Q[n]$, assumindo o uso de um Sistema de Numeração Posicional Convencional (SNPC), com base $b = 2$. Por exemplo, um dos padrões comerciais usados para quantização de um sinal de áudio é uma *bit depth* de $D_{bits} = 16$ *bits*.

Pode-se dizer que o parâmetro *bit rate* (taxa ou freqüencia de *bits*) F_B representa uma associação dos parâmetros *sampling rate* e *bit depth*. Dada uma *sampling rate* F_S, tem-se a ocorrência de F_S valores (amostras) a cada segundo. Por sua vez, para uma *bit depth* D_{bits}, emprega-se D_{bits} *bits* para representar cada valor (amostra). Portanto, pode-se definir uma *bit rate* dada por $F_B = F_S \cdot D_{bits}$ com unidade dada por (amostras/s) $\cdot$ (*bits*/amostra) ou (*bits*/s) ou bps (*bits* por segundo). Por exemplo, para um padrão comercial onde $F_S = 44,1$ kHz e $D_{bits} = 16$ *bits*, obtém-se $F_B = F_S \cdot D_{bits} = 44,1 \cdot 16 = 705,6$ kbps.

2.6 Classificações de sinais

De acordo com os tipos das variáveis envolvidas, podem-se identificar as seguintes classes:

- Sinal analógico: todas as variáveis são contínuas.
- Sinal amostrado: discretização das variáveis independentes (amostragem).
- Sinal quantizado: discretização da variável dependente (quantização).
- Sinal digital: todas as variáveis são discretas (amostragem + quantização).

Para um sinal amostrado e um sinal digital, cabe ressaltar que ambos podem ser representados por um conjunto ordenado, cujos elementos são valores numéricos. Portanto, nesses casos, o sinal é interpretado como uma seqüência indexada (ou um vetor) de números. Para um sinal amostrado, pode-se considerá-lo um vetor com índice inteiro e elementos reais. Para um sinal digital, pode-se considerá-lo um vetor com índice inteiro e elementos inteiros.

2.7 Classificações de sistemas

Ao se classificar um sistema, procura-se por alguma característica que possa ser útil na sua análise e/ou no seu projeto e/ou na sua aplicação. Algumas classificações possíveis são:

- Tipo de sinal manipulado: Analógico × Amostrado × Quantizado × Digital × Híbrido.
- Causalidade: Causal × Não causal.
- Variância ao tempo: Variante × Invariante ao tempo.
- Linearidade: Linear × Não linear.
- Armazenamento: Instantâneo (sem memória) × Dinâmico (com memória).

Os sistemas abordados no presente texto serão classificados, na sua grande maioria, como sistemas dinâmicos, lineares, invariantes ao tempo, causais e amostrados/digitais.

2.8 Estado e variáveis de estado de um sistema dinâmico

O estado de um sistema, em um determinado instante, pode ser descrito como o conjunto formado pelos valores que cada uma de suas variáveis assume nesse instante. Porém, assim como todos os elementos de um determinado espaço vetorial podem ser descritos por uma combinação linear dos elementos de uma de suas bases, pode-se mostrar que todas as variáveis de um sistema podem ser definidas por uma combinação linear de um conjunto mínimo de suas variáveis. Logo, o estado de um sistema pode ser determinado por um dos possíveis conjuntos mínimos de suas variáveis. Surge, então, o conceito de variáveis de estado, definidas por:

> **"Variáveis de estado de um sistema é o conjunto mínimo de variáveis (linearmente independentes entre si) do sistema que, conjuntamente com as entradas do sistema no instante t_0, é capaz de definir as saídas do sistema para $t \geq t_0$."**

Define-se como ordem de um sistema o número de variáveis que compõem o conjunto de variáveis de estado do sistema.

Assim como as bases dos espaços vetoriais não são únicas e algumas delas apresentam características especiais (bases canônicas, bases ortogonais, bases ortonormais), o conjunto de variáveis de estado também não é único e alguns deles apresentam características especiais que facilitam tanto a análise quanto o projeto de circuitos.

Nos circuitos elétricos, os capacitores e os indutores são os elementos capazes de armazenar energia. Logo, as tensões dos capacitores e as correntes dos indutores são candidatas naturais para o conjunto das variáveis de estado de um circuito analógico. Deve-se ter atenção para os seguintes casos: malhas que contenham apenas capacitores e nós onde estejam ligados apenas indutores. Em ambos os casos, uma das variáveis será uma combinação linear das demais, não podendo fazer parte do conjunto de variáveis de estado do circuito.

De acordo com a capacidade de armazenar informação, os circuitos digitais podem ser divididos em dois grandes grupos: circuitos combinacionais e circuitos seqüenciais. Como o nome indica, a saída de um circuito combinacional é função de uma simples combinação lógica das suas entradas. Logo, essa classe de circuitos não consegue armazenar informação. Por sua vez, a denominação *seqüencial* indica que a ordem segundo a qual as entradas foram aplicadas ao longo do tempo influencia a saída do circuito. Isso indica que, nessa classe, existem elementos que conseguem armazenar informação. Portanto, as saídas dos elementos armazenadores são

candidatas naturais para o conjunto das variáveis de estado de um circuito digital. Porém, assim como no caso analógico, pode haver redundância em relação aos elementos armazenadores de informação, sendo necessário garantir a escolha de um conjunto mínimo.

2.9 Exemplo de aplicação de processamento digital

A seguir, é apresentada uma visão geral para um exemplo de aplicação de processamento digital em sistemas de comunicação.

2.9.1 Sistemas de comunicação

O ato de comunicar pode ser definido como o transporte de uma informação, através de um meio adequado, de uma fonte a um destino. Por sua vez, telecomunicação significa comunicar à distância, onde deve ser ressaltado o caráter relativo que possui o conceito de distância. A Figura 2.3 ilustra o processo de comunicação através de um sistema de comunicação genérico.

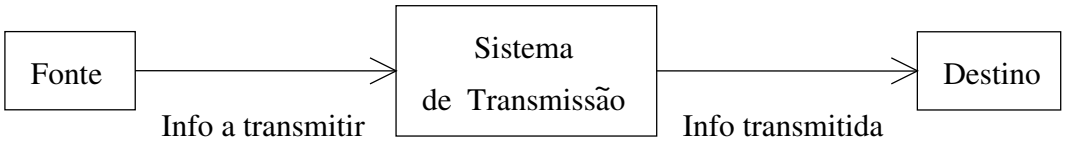


Figura 2.3: Sistema de comunicação genérico.

2.9.2 Subsistemas de comunicação

A Figura 2.4 ilustra uma possível composição para um sistema de comunicação com um sistema de transmissão genérico. A divisão apresentada para o sistema de transmissão é apenas didática e genérica. Porém, alguns pontos merecem destaque, os quais são apresentados a seguir.

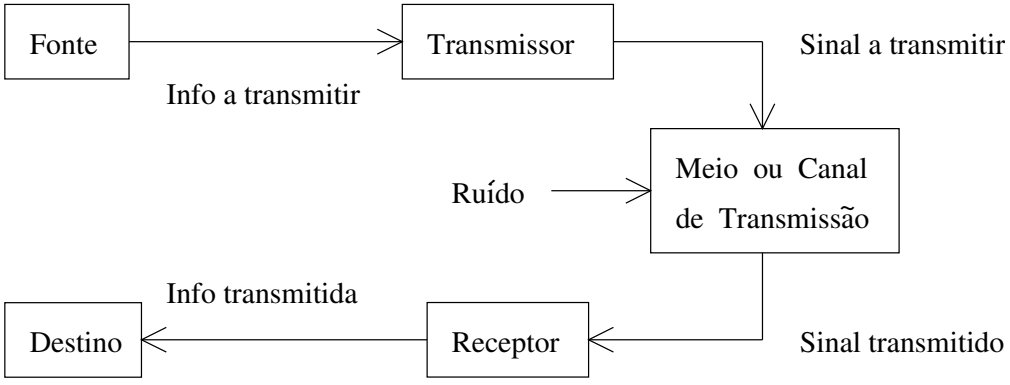


Figura 2.4: Sistema de comunicação com sistema de transmissão genérico.

Na Figura 2.4, deve-se observar a presença de um novo elemento denominado de sinal, que é definido como uma entidade que carrega informação. Assim, a função do transmissor é, basicamente, adequar a informação a ser transmitida ao meio de transmissão escolhido. Isso é feito através da codificação da informação em um sinal que o meio possa transmitir eficientemente. A fim de modelar as imperfeições do processo de transmissão, um sinal não desejado (ruído) é anexado ao meio. Finalmente, a função do receptor é tentar recuperar a informação original, através da decodificação do sinal transmitido.

Dependendo do sistema de transmissão modelado, inúmeras outras divisões podem ser propostas, onde vários subsistemas diferentes podem ser utilizados, possuindo os mais diversos nomes e funções. A título de exemplo, a Figura 2.5 apresenta o diagrama de blocos de um sistema de comunicação com um sistema de transmissão digital genérico.

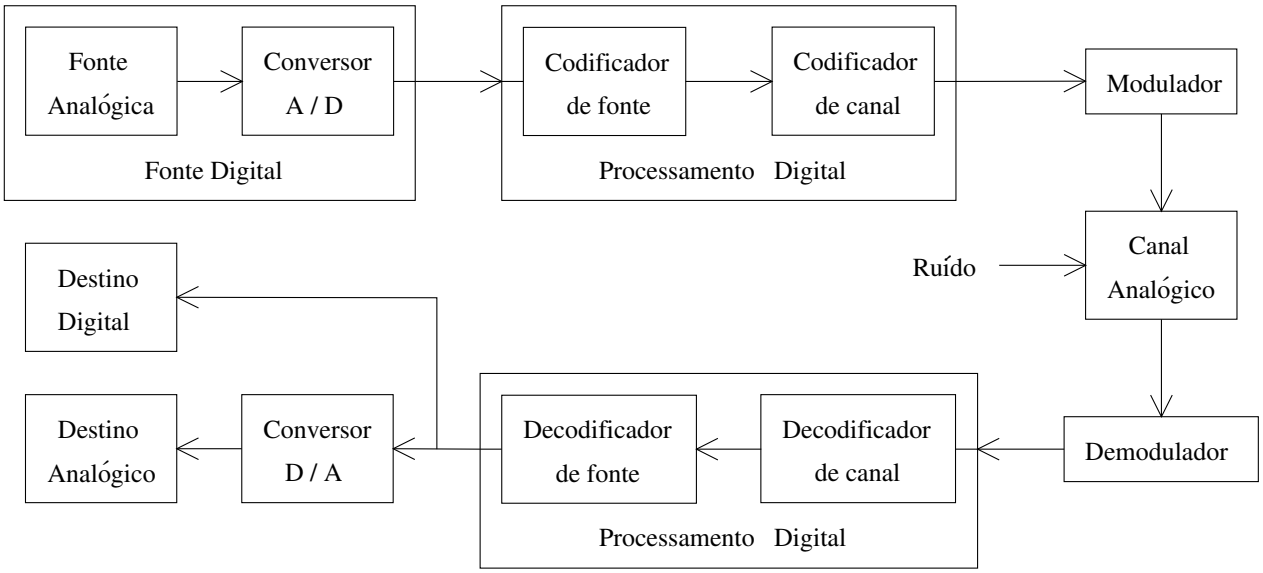


Figura 2.5: Sistema de comunicação com sistema de transmissão digital genérico.

2.9.3 Sinais e meios de transmissão

Diversas são as possibilidades de implementação, tanto dos sinais que carregam a informação desejada quanto do meio ou canal que transmitirá tais sinais.

Naturalmente, existe uma grande relação entre o sinal a transmitir e o meio ou canal de transmissão. Assim, a escolha de determinado sinal implicará a utilização de uma determinada classe de canais. Da mesma forma, a escolha de um determinado meio de transmissão possibilitará o uso de uma classe específica de sinais.

Nas situações onde não é possível um casamento adequado entre sinal e canal, um ou mais subsistemas devem ser adicionados, com a finalidade de gerar um sinal mais adequado ao meio disponível, a partir do sinal original. Isso é ilustrado na Figura 2.5, onde os blocos Modulador e Demodulador são responsáveis por compatibilizar a transmissão de sinais digitais através de um canal analógico.

Nos casos onde diversas técnicas são empregadas, com o intuito de melhorar a qualidade e a eficiência da transmissão, diversos sinais intermediários podem ser gerados. Isso também é exemplificado na Figura 2.5, onde os conversores A/D (Analógico/Digital) e D/A (Digital/Analógico) são utilizados a fim de permitir que técnicas de processamento de sinal digital possam ser empregadas em um sinal originalmente analógico.

2.10 Processamento de sinais

Algumas das definições básicas em processamento de sinais são as seguintes:

- Objeto do processamento: sinal (definido como uma entidade que carrega informação).
- Agente do processamento: sistema.
  - "Um sistema é um cojunto de elementos, que interagem entre si, com o objetivo de realizar uma determinada função".
  - Arquitetura de um sistema: variáveis, elementos, topologia e função.
- Domínio do processamento: domínio no qual a função do agente é definida.
  - Tempo/espaço (forma) × freqüência (composição espectral).
- Ação do processamento: função exercida pelo agente sobre o objeto.
  - Conformação (tempo/espaço) × Alteração espectral (freqüência).
- Arquitetura genérica do processamento:
  - Sinal de entrada (ou estímulo ou excitação ou perturbação).
  - Condições iniciais ou estado inicial.
  - Sistema.
  - Sinal de saída (ou resposta).
- Nomenclatura usual: "Sinal" (sinal desejado) × "Ruído" (sinal indesejado).

2.11 Arquitetura de sistemas de processamento digital

A arquitetura genérica de um sistema híbrido de processamento digital de sinais é formada pelos seguintes elementos:

- Sinal de entrada analógico $x_a(t)$: comumente, um sinal elétrico (tensão ou corrente).
- Pré-processamento analógico:
  - Filtro analógico *anti-aliasing*: com seletividade em freqüência do tipo passa-baixa.
  - Sinal de entrada analógico filtrado $x_f(t)$: sinal analógico, gerado pelo filtro *anti-aliasing*, a partir do sinal $x_a(t)$.
  - Amostragem e retenção (*Sample-and-Hold* ou S/H): responsável por obter amostras do sinal de entrada analógico e por manter fixo o valor de cada amostra durante um determinado tempo. Ele é utilizado para que a amostra seja diretamente processada por um sistema a dados amostrados ou para que ela seja convertida em um número a ser processado por um sistema digital.
  - Sinal de entrada analógico (amostrado e retido) $x_f(nT_S)$: sinal analógico, gerado pelo S/H, a partir de amostras do sinal $x_f(t)$. Também é denominado de sinal de dados amostrados (*sampled-data signal*).

  - Quantizador: condiciona os valores das amostras a valores representáveis no sistema digital.
  - Codificador: converte o valor de cada amostra em uma representação válida no Sistema de Numeração utilizado no sistema digital. Tipicamente, é utilizado o Sistema de Numeração Posicional Convencional, com base $b = 2$ e codificação em complemento a 2.
  - Conversor Analógico-Digital (*Analog to Digital Converter* ou ADC ou A/D): definido como um bloco funcional, formado pela união "Quantizador + Codificador" ou "S/H + Quantizador + Codificador". O sinal SOC (*Start Of Conversion*) provoca o início de uma conversão, enquanto o sinal EOC (*End Of Conversion*) notifica o seu término.

- Sinal de entrada digital $\{x[n]\}_Q$: seqüência numérica quantizada que possui uma representação numérica computacional.
- Processador de sinal digital (*Digital Signal Processor* ou DSP): implementado por um circuito digital, que pode ser fixo, configurável ou programável.
- Sinal de saída digital $\{y[n]\}_Q$: seqüência numérica quantizada que possui uma representação numérica computacional.
- Pós-processamento analógico:
  - Decodificador: converte uma representação válida no Sistema de Numeração utilizado no sistema digital em um valor de amostra do sinal de saída analógico.
  - Gerador de impulso/pulso: responsável por gerar um elemento para cada amostra. No caso de um impulso, o valor da amostra é relacionado com a sua intensidade (*strength*). No caso de um pulso, o valor da amostra é relacionado com a sua amplitude.
  - Conversor Digital-Analógico (D/A): (*Digital to Analog Converter* ou DAC ou D/A): definido como um bloco funcional, formado pela união "Decodificador + Gerador de impulso/pulso".
  - Sinal de saída quantizado $\{y(nT_S)\}_Q$: sinal quantizado, gerado pelo D/A, a partir da seqüência numérica quantizada $\{y[n]\}_Q$. Comumente, um sinal elétrico (tensão ou corrente).
  - Filtro analógico de suavização (*smoothing*): com seletividade em freqüência do tipo passa-baixa.
- Sinal de saída analógico $y_a(t)$: comumente, um sinal elétrico (tensão ou corrente).

A cadeia de processamento descrita acima pode ser visualizada nas Figuras 2.6 a 2.8.

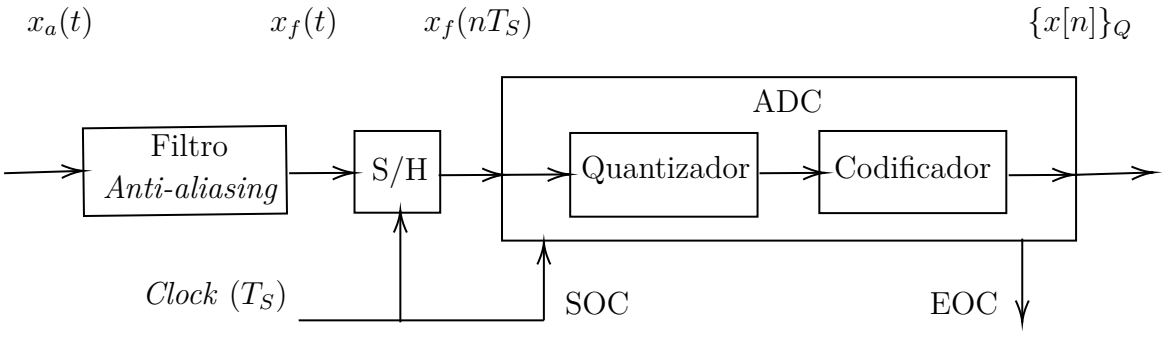


Figura 2.6: Cadeia de pré-processamento analógico.

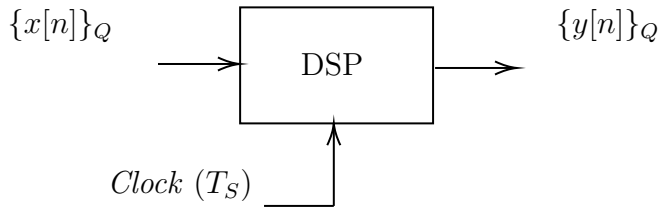


Figura 2.7: Processador de Sinal Digital.

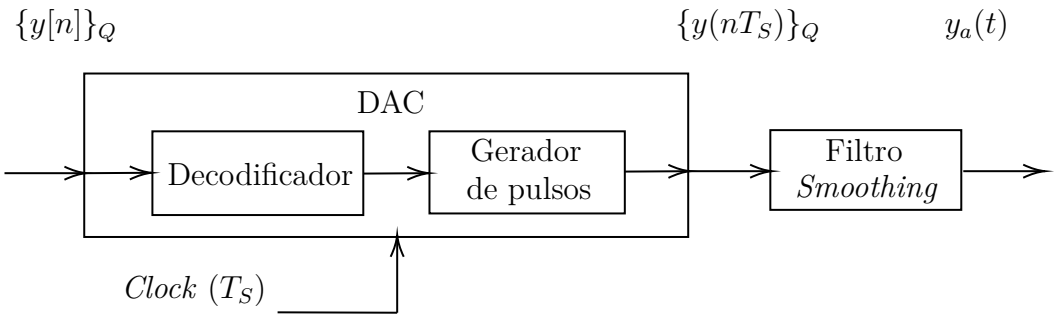


Figura 2.8: Cadeia de pós-processamento analógico.

2.12 Implementações analógicas e digitais

A seguir, é realizada uma breve discussão sobre a escolha de uma solução analógica ou digital para o problema de implementação de um sistema de processamento de sinais. Inicialmente, o problema é definido. Em seguida, é realizada uma comparação entre as características das implementações analógicas e digitais.

2.12.1 Definição do problema

Dado que um mesmo sistema pode ser implementado por sinais/componentes analógicos e digitais, naturalmente surge a questão de qual das opções é a melhor.

Diversas são as comparações encontradas na literatura técnica entre implementações de sistemas que empregam sinais/componentes analógicos e digitais.

Adeptos de ambos os tipos de implementação facilmente definem parâmetros que os favorecem e normalmente estabelecem comparações tentando mostrar que a sua implementação de preferência é, realmente, a melhor opção.

Antes de tecer meras comparações absolutas, muitas vezes passionalmente polarizadas, por um tipo ou outro de implementação, deve-se lembrar que estão sendo comparadas duas alternativas de implementação essencialmente diferentes. Logo, os parâmetros de comparação normalmente empregados fornecem apenas características individuais de cada tipo de implementação, ao invés de estabelecer bases reais de comparação.

Por um lado, uma implementação analógica mapeia sinais matemáticos em sinais físicos e realiza as operações matemáticas através de componentes físicos que possuem equações de definição capazes de realizar os cálculos necessários, isolada ou conjuntamente com outros componentes. Pode-se dizer que o problema matemático é transformado em um problema físico e implementado por sinais/componentes físicos.

Por sua vez, uma implementação digital, apesar de obviamente empregar componentes físicos, não utiliza os seus sinais e as suas equações diretamente. Ao invés disso, é realizada uma codificação mais complexa, de tal forma que a implementação simula diretamente as operações e os operandos matemáticos. Nesse caso, pode-se dizer que o problema matemático não é mapeado em um problema físico, sendo apenas simulado em uma implementação física, mas que é virtualmente matemática.

A comparação direta torna-se ainda mais sem sentido a partir da constatação de que sistemas digitais podem ser implementados por *software*, por *hardware* digital ou por ambos simultaneamente.

Com tais conceitos em mente, não é difícil perceber que qualquer tentativa para estabelecer parâmetros de comparação entre implementações analógicas e digitais conduz apenas a definições de característica relativas de cada uma das opções consideradas.

Também não é difícil perceber que, devido às características próprias de cada uma das implementações, cada uma delas poderá ser proposta como a melhor solução para problemas específicos, os quais necessitem de tais características. Por vezes, até mesmo uma solução que misture ambos os tipos de implementação pode ser a melhor escolha.

Finalmente, é importante ressaltar que o levantamento de parâmetros comparativos é importante e deve ser efetuado. Não para que se defina qual das duas opções de implementação é absolutamente a melhor, mas para que se possa estabelecer uma base de dados que fundamente a decisão de projeto diante de cada problema diferente.

2.12.2 Características das implementações analógicas e digitais

A seguir, levando-se em consideração uma implementação eletrônica, são apresentadas algumas características da implementação de sistemas para os casos analógico e digital.

- Sinais
  - Analógico: valores contínuos de tensão e de corrente.
  - Digital: seqüências de números, representados por uma determinada codificação, contendo um número finito de símbolos. Normalmente, é utilizada uma codificação binária.
- Componentes básicos
  - Analógico: componentes passivos (resistor, capacitor e indutor) e componentes ativos (transistor, OpAmp e OTA).
  - Digital: atrasador unitário (registrador), multiplicador e somador.
- Armazenamento de sinal por longo prazo
  - Analógico: por ser um processo que envolve valores contínuos de grandezas físicas, sofre grande degradação.
  - Digital: uma vez que, geralmente, envolve codificação binária, sofre pequena degradação.
- Ocupação de área
  - Analógico: de forma geral, menor ocupação.
  - Digital: de forma geral, maior ocupação.
- Repetitibilidade/reprodutibilidade
  - Analógico: uma vez que os parâmetros são físicos e contínuos, necessita de um bom processo de fabricação.
  - Digital: dado que os parâmetros são matemáticos, a fabricação é repetitível por construção.
- Variabilidade na fabricação
  - Analógico: os componentes possuem um valor nominal e uma incerteza associada ao processo de fabricação. Devem ser utilizadas técnicas de projeto de sistemas que controlem a sensibilidade à variação dos valores dos componentes.
  - Digital: valor matemático fixo, associado à quantidade símbolos utilizados na codificação numérica.
- Variabilidade na operação
  - Analógico: os componentes são influenciados por fatores ambientais (temperatura, humidade, etc.), podem ser sujeitos a envelhecimento (*aging*) e podem sofrer desgaste por uso. Devem ser utilizadas técnicas de projeto de sistemas que controlem a sensibilidade à variação dos valores dos componentes.
  - Digital: operação baseada em valores matemáticos fixos, representados por uma quantidade finita de símbolos utilizados na codificação numérica.

- Variabilidade com as freqüências envolvidas nos sinais
  - Analógico: as dimensões e a funcionalidade dos componentes podem ser fortemente afetadas pela faixa de valores de freqüência utilizada.
  - Digital: não afetado.
- Fontes de erro
  - Analógico: além da variação provocada por fatores ambientais, os componentes são diretamente afetados por ruídos dos mais variados tipos, provocados pelos mais diversos mecanismos, sendo intrínsecos aos próprios componentes e/ou induzidos por fontes externas.
  - Digital: devido ao número finito de símbolos usado na codificação numérica, surgem os seguintes problemas de aproximação numérica: quantização dos valores das seqüências, quantização dos valores dos coeficientes dos multiplicadores e aproximação dos valores finais das operações (soma e multiplicação).
- Programabilidade
  - Analógico: componentes podem ser fixos e/ou variáveis. No caso de componentes fixos, devem ser desenvolvidas técnicas de projeto de sistemas que permitam o controle da variação de um ou mais parâmetros do sistema.
  - Digital: naturalmente programável.
- Complexidade funcional
  - Analógico: implementada com dificuldade.
  - Digital: facilmente implementada.
- Multiplexação temporal de sinais
  - Analógico: difícil implementação.
  - Digital: fácil implementação.

2.13 Exercícios propostos

1. Uma barra de cobre mede $L = 3,7$ m. Para avaliar o perfil da distribuição de temperatura sobre a barra, são realizadas medições a cada $0,4$ cm, a partir de uma das suas extremidades. A temperatura pode variar de -40 ^{o}C até 85 ^{o}C. O instrumento de medição fornece valores múltiplos de 0,3 ^{o}C. A fim de dimensionar a memória necessária para armazenar os dados, atenda aos seguintes itens:

   (a) Quantos registros serão necessários para armazenar o total de amostras?

   (b) Quantos níveis de quantização serão necessários para representar cada amostra?

2. Um sinal de voz é uniformemente amostrado com uma taxa de $F_s = 8\ kHz$, durante um intervalo de tempo de $T_{rec} = 1\ min$. Quantos registros são necessários para armazenar o total de amostras?

3. Um sinal de música é uniformemente amostrado com uma taxa de $F_s = 40\ kHz$, durante um intervalo de tempo de $T_{rec} = 1\ min$. Quantos registros são necessários para armazenar o total de amostras?

4. Uma fotografia de dimensões $L_x = 9\ cm$ e $L_y = 11\ cm$ é uniformemente amostrada, em um padrão retangular, com intervalos de amostragem dados por $T_{S_x} = 7\ mm$ e $T_{S_y} = 3\ mm$. Quantos registros são necessários para armazenar o total de amostras?

5. Valores de temperatura na faixa de $T = [110; 127]\ K$ são representados com uma resolução de $r_T = 0,7\ K$. Quantos valores são representados?

6. Precisa-se medir mais do que $N_v = 759$ valores de uma dada corrente elétrica na faixa $I = [-12; 12]\ A$. Qual o valor máximo de resolução para realizar tais medidas?

7. Um aluno de Processamento Digital de Sinais, decidiu digitalizar o conteúdo musical de alguns dos seus discos de vinil, do tipo LP (*Long Play*). Os dados provenientes da digitalização, além de serem armazenados, também serão transmitidos a alguns de seus amigos. Os LPs 1, 2 e 3, contêm 14, 16 e 18 músicas, respectivamente. Cada música tem uma duração média de um minuto e trinta segundos. Ele decidiu realizar uma amostragem uniforme com freqüência de amostragem $F_S = 44\ kHz$ e converter as amostras para um padrão digital com 10 *bits*. O sistema de transmissão que o aluno vai utilizar trabalha com quadros de 8 *bits* de informação mais 3 *bits* de controle, para cada transmissão. A taxa de transmissão é de 9600 *bits/s*. Suponha que os dados de amostras diferentes não possam ser combinados para realizar uma única transmissão. Suponha ainda que o custo de transmissão é de $1,20\ R\$/min$. Calcule os seguintes parâmetros:

   (a) O total de amostras a serem armazenadas. (Resposta: 190.080.048 amostras.)

   (b) O total de *bits* a serem armazenados. (Resposta: 1.900.800.480 *bits*.)

   (c) O tempo mínimo para transmissão de todos os dados, expresso em horas, minutos e segundos (HH:MM:SS). (Resposta: $t_{Tx} = 435.600\ s = 7.260\ min = 121\ h$.)

   (d) O custo total mínimo para transmissão de todos os dados. (Resposta: $R\$\ 8.712,00$.)

8. A memória de um computador digital convencional é composta por R registros, onde cada registro ocupa B dígitos binários (*bits*). Logo, ela pode ser especificada como um dispositivo de armazenamento de $R \times B$ (*bits*). Um sinal de voz inteligível possui componentes senoidais com freqüências na faixa $0 \leq f \leq 3.9\ kHz$. Suponha que um sinal de voz seja amostrado com $F_S = 8\ kHz$, durante um intervalo de $T_{rec} = 40\ s$. Suponha que o sinal foi normalizado antes da amostragem, de tal forma que a sua amplitude reside na faixa $0 \leq |A| < 1$. Considere um *grid* de quantização uniforme com intervalo mínimo (resolução) igual a 0.003 e que as amostras quantizadas são representadas em complemento a dois. Especifique o espaço de armazenamento ($R \times B$) necessário.

9. Suponha as funções lógicas $f_1(A, B, C)$ e $f_2(A, B, C)$, definidas pela Tabela Verdade que é apresentada pela Tabela 2.1. Considere as seguintes associações: $F = 0$ e $T = 1$. Considere o tempo contínuo t e o índice k. Suponha que os valores do índice k são trocados a cada intervalo constante T_S. Esboce os seguintes gráficos:

   - $A \times t$.
   - $B \times t$.
   - $C \times t$.
   - $f_1(A, B, C) \times t$.
   - $f_2(A, B, C) \times t$.

k	A	B	C	$f_1(A, B, C)$	$f_2(A, B, C)$
0	F	F	F	F	F
1	F	F	T	F	T
2	F	T	F	F	T
3	F	T	T	T	F
4	T	F	F	F	T
5	T	F	T	T	F
6	T	T	F	T	T
7	T	T	T	T	T

Tabela 2.1: Tabela Verdade das funções $f_1(A, B, C)$ e $f_2(A, B, C)$.

10. Suponha o sinal analógico $x(t)$ dado por

$$x(t) = \begin{cases} 0 & , \quad -\infty < t \leq -2.5 \text{ s} \\ 4t + 10 & , \quad -2.5 \leq t \leq 0 \text{ s} \\ -4t + 10 & , \quad 0 \leq t \leq 2.5 \text{ s} \\ 0 & , \quad 2.5 \leq t < \infty \text{ s} \end{cases} .$$

    Suponha o sinal analógico periódico $\tilde{x}(t)$, com período fundamental $T_f = 6$ s, criado pela extensão periódica de $x(t)$. Suponha uma amostragem uniforme, com período de amostragem $T_S = 0.5$ s. Esboce os seguintes gráficos:

    (a) $x(t) \times t$, $-9 \leq t \leq 9$ s.
    (b) $\tilde{x}(t) \times t$, $-9 \leq t \leq 9$ s.
    (c) $\tilde{x}(nT_S) \times t$, $-9 \leq t \leq 9$ s.
    (d) $\tilde{x}[n] \times n$, $-18 \leq n \leq 18$.

11. Suponha que o sinal discreto definido por $x[n] = n^2$ foi uniformemente quantizado, com resolução r e usando a técnica de aproximação por arredondamento, considerando a metade do intervalo de representação como referência, gerando o sinal $\{x[n]\}_Q$.

    Considerando $n = [0, \pm1, \pm2, \pm3, \pm4, \pm5, \pm6]$, esboce os seguintes gráficos:

    (a) $x[n] \times n$.
    (b) $\{x[n]\}_Q \times n$, para $r = 3$.
    (c) $\{x[n]\}_Q \times n$, para $r = 2$.

12. Suponha que quantidades numéricas são representadas por um Sistema de Numeração Posicional Convencional (SNPC), com base $b = 2$ e nenhuma forma de codificação adicional. Suponha ainda que, nesse caso, as quantidades numéricas são representadas com 8 dígitos binários (*bits*). Atenda aos seguintes itens:

    (a) Calcule quantos números poderão ser representados.
    (b) Calcule, para a faixa de valores $\boldsymbol{q} = [7; 23]$, quais valores numéricos serão representados e qual será a resolução.

13. Suponha que o sinal contínuo $y(x)$ foi uniformemente amostrado com uma resolução R_x, produzindo o sinal amostrado $y(nR_x)$. Suponha que o sinal amostrado teve sua abscissa normalizada por R_x, gerando a seqüência discreta $y[n]$. Suponha que a seqüência discreta foi uniformemente quantizada, com resolução R_y (e usando a técnica de aproximação por arredondamento, considerando a metade do intervalo de representação como referência), gerando a seqüência digital $y_D[n] = y_Q[n] = \{y[n]\}_Q$. Suponha que a seqüência digital teve sua ordenada normalizada por R_y, gerando a seqüência digital normalizada $y_N[n]$.

    Dado $y(x) = x^2$, esboce os seguintes gráficos:

    (a) $y(x) \times x$, para $-13 \leq x \leq 13$.
    (b) $y(nR_x) \times nR_x$, para $R_x = 2$ e $-6 \leq n \leq 6$.
    (c) $y[n] \times n$, para $-6 \leq n \leq 6$.
    (d) $y_D[n] \times n$, para $R_y = 7$ e $-6 \leq n \leq 6$.
    (e) $y_N[n] \times n$, para $R_y = 7$ e $-6 \leq n \leq 6$.

    Em continuação, suponha que, a partir da seqüência digital normalizada $y_N[n]$, e usando uma nova resolução $\hat{R}_x$, foi obtido um novo sinal amostrado $\hat{y}_N(n\hat{R}_x)$. Suponha que, a partir do novo sinal amostrado, e usando algum tipo de interpolação, foi obtido um novo sinal analógico $\hat{y}_N(x)$.

    Dado $\hat{R}_x = 1$, esboce os seguintes gráficos:

    (a) $y_N[n] \times n$, para $-6 \leq n \leq 6$.
    (b) $\hat{y}_N(n\hat{R}_x)$, $-6 \leq n \leq 6$.
    (c) $\hat{y}_N(x)$, para $-13 \leq x \leq 13$.

    Dado $\hat{R}_x = 4$, esboce os seguintes gráficos:

    (a) $y_N[n] \times n$, para $-6 \leq n \leq 6$.
    (b) $\hat{y}_N(n\hat{R}_x)$, $-6 \leq n \leq 6$.
    (c) $\hat{y}_N(x)$, para $-25 \leq x \leq 25$.

Parte III

Circuitos digitais combinacionais

Capítulo 3

Circuitos combinacionais: conceitos básicos

3.1 Introdução

- Circuitos combinacionais × circuitos seqüenciais.
- Circuitos combinacionais são sistemas instantâneos ou sem memória.
- Circuitos seqüenciais são sistemas dinâmicos ou com memória.
- Por serem sistemas instantâneos, os circuitos combinacionais respondem sempre da mesma forma, em qualquer momento, para os mesmos valores das variáveis de entrada.
- Por sua vez, por serem sistemas dinâmicos, dependendo da informação que se encontre armazenada, os circuitos seqüenciais podem responder de formas diferentes, em diferentes momentos, para os mesmos valores das variáveis de entrada.
- Circuitos seqüenciais também podem ser denominados de máquinas de estados ou de autômatos.

3.2 Hierarquia em *hardware* e em *software*

- Elementos básicos.
- Blocos funcionais de complexidade básica, média e elevada.
- Sistemas de complexidade básica, média e elevada.

3.3 Codificação

- Código: sintaxe (símbolos) e semântica (significado).
- No caso geral: representação de idéias.
- No caso de computação: representação de dados.
- No caso de transmissão digital: representação de informação baseada nas características da fonte (exemplo: compressão de dados) e do canal (exemplo: redução da taxa de erro).

3.4 Elementos codificados: informação e quantidade

- Informação (fatos, classificações) × Quantidade (números, contagem).
- Informação pode ser codificada em quantidade.
- Quantidade pode ser codificada como informação.
- Ambos podem ser representados e manipulados como um único elemento.
- Ambos são um único elemento.
- Ambos significam idéias codificadas.

3.5 Representação de informação

- Uma vez que a informação é multivalorada, pode-se utilizar, para representá-la, um único dispositivo com múltiplos estados ou vários dispositivos com um número reduzido de estados.
- O número total de estados, ou de condições às quais os dispositivos podem ser ajustados, deve ser igual ou maior ao número de valores possíveis que a informação a ser representada pode assumir.
- Na tentativa de minimizar o número de dispositivos e o número de diferentes estados em cada dispositivo, foi demonstrado, baseado em um dado modelo matemático, que o número de estados ótimo para cada dispositivo é o número $e = 2.718281828459$ [Ric56].
- Embora a melhor aproximação seja um total de 3 estados por dispositivo, a disponibilidade de dispositivos eletro-eletrônicos que apresentam 2 estados de operação (chaves, relés, diodos, transistores), aliada à facilidade e à confiabilidade da implementação (estabilidade e robustez), têm levado à escolha da representação através de mecanismos envolvendo 2 estados por dispositivo.
- Assim, para representar informações envolvendo N estados, são necessários M dispositivos de 2 estados, tal que $2^M \geq N$.

3.6 Representação de quantidade

- Representação numérica digital de uma informação analógica ou discreta.
- Representação digital de informação analógica: amostragem e quantização.
- Representação digital de informação discreta: quantização.
- Fontes de erro:
  - Acurácia da medida.
  - Resolução da medida (precisão).
  - Limites da representação da medida (máximo e mínimo).
  - Erros de conversão A/D.

  - Capacidade de armazenamento da amostra.

- Quantidades são naturalmente multivaloradas e representadas por símbolos.
- Novamente, dois extremos são possíveis. Por um lado, pode-se usar um único símbolo variável, cujas variações representam todos os valores numéricos desejados. De outra forma, pode-se optar por uma combinação de símbolos, pertencentes a um conjunto que é capaz de representar apenas uma faixa de valores.
- Exemplos de representação numérica:
  - Um sistema de numeração com 4 dígitos e resolução de 0.001 pode representar números positivos de 0.000 até 9.999, num total de 10.000 valores diferentes.
  - Para representar 100 valores diferentes, podem-se utilizar 100 símbolos fixos diferentes ou X símbolos variáveis, com Y valores cada.

3.7 Representações em circuitos digitais

Para representar e manipular as informações não numéricas e as quantidades numéricas, são comumente utilizados, respectivamente, um sistema lógico e um sistema de numeração.

Se for possível empregar uma simbologia única, que atenda a ambas as representações, podem-se modelar sistemas que manipulem, sem distinção, informações não numéricas e quantidades numéricas. Em uma solução comumente empregada, os valores lógicos são reinterpretados como valores numéricos e são propostas funções lógicas que podem ser interpretadas como operações numéricas. Tais sistemas, e seus circuitos, que se utilizam de um sistema lógico para modelar unificadamente informações não numéricas e quantidades numéricas, são denominados de sistemas, e circuitos, digitais.

Capítulo 4

Funções lógicas

4.1 Introdução

- O objetivo deste documento não é trabalhar o ensino da área matemática denominada de lógica. Pelo contrário, busca-se apenas aproveitar os resultados desse ramo da matemática na geração de circuitos eletro-eletrônicos que implementem funções lógicas.

- Porém, é recomendável que se discutam, pelo menos, algumas questões básicas sobre a chamada lógica simbólica ou lógica matemática. Isso é feito a seguir, antes de se tratar da implementação das funções lógicas.

- Em todas as áreas de atuação profissional pode-se encontrar algum tipo de manipulação de informações.

- A manipulação de informações pode ser dividida em três partes básicas: a obtenção dos dados, o processamento desses dados e a geração de novos dados.

- Toda e qualquer ação envolvida nesses processos requer, de certa forma, tomadas de decisão.

- Compreender o raciocínio humano que rege as tomadas de decisão possibilita que tal mecanismo seja implantado em sistemas artificiais.

- A lógica pode ser vista como um ramo de estudos da matemática que fornece elementos para a tentativa de modelagem do raciocínio humano.

- A lógica formal fornece uma linguagem estruturada para a definição e a manipulação de argumentos.

4.2 Mecanismos básicos de raciocínio

Podem-se identificar dois mecanismos, básicos e distintos, que são comumente utilizados no raciocínio humano, quais sejam: a indução (ou analogia) e a dedução. Ambos são brevemente discutidos a seguir.

4.2.1 Raciocínio por indução (ou por analogia)

- O raciocínio por indução (ou por analogia) é baseado em métodos empíricos.
- Nesse caso, conclui-se que algo é sempre verdadeiro a partir de um número limitado de exemplos. Tais conclusões empíricas são, na realidade, generalizações, baseadas em um número limitado de observações ou de experimentos.
- Assim, a partir de alguns exemplos, definem-se procedimentos padrões para a obtenção de algum resultado, ao invés de serem realizadas deduções ou demonstrações
- As conclusões estabelecidas por meio de um raciocínio por indução não são completamente garantidas pelos fatos. Pelo contrário, os exemplos utilizados apenas levam a crer que tais conclusões são sempre verdadeiras. Portanto, há apenas uma chance de que as conclusões estejam corretas.
- Dessa forma, não se pode dizer que o mecanismo de raciocínio por indução seja absolutamente confiável.
- Cabe ressaltar que o raciocínio por indução (ou por analogia) não deve ser confundido com o Princípio de Indução Matemática, o qual surge naturalmente como um teorema, dentro de um processo de raciocínio por dedução.

4.2.2 Raciocínio por dedução

- O raciocínio por dedução envolve um conjunto de proposições e um conjunto ou sistema de regras.
- As proposições são conjuntos de enunciados. Por sua vez, um determinado conjunto ou sistema de regras também é denominado de um sistema de lógica ou simplesmente de uma lógica.
- As proposições iniciais são denominadas de premissas. Aplicando-se as regras sobre as premissas, procura-se demonstrar, ou deduzir, uma proposição final, que é denominada de conclusão.
- Deve ficar claro que, no raciocínio por dedução, não se procura saber se a conclusão é Verdadeira ou Falsa. O objetivo do raciocínio por dedução é alcançar a conclusão, a partir das premissas, utilizando-se as regras definidas. Se a conclusão puder ser deduzida pelas regras, a partir das premissas, então o raciocínio é dito Válido (ou Correto ou Legítimo). Caso contrário, ele é dito Inválido (ou Incorreto ou Ilegítimo).

4.2.3 Estruturas axiomáticas

- As denominadas estruturas axiomáticas são diretamente relacionadas com o raciocínio dedutivo.
- Ao se realizar uma argumentação por raciocínio dedutivo deve-se tomar cuidado para que não se estabeleça uma circularidade no discurso, onde A é definido a partir de B, que é definido a partir de C, que é definido a partir de A.
- O emprego de uma estrutura axiomática é uma forma de se impedir que ocorra uma circularidade na discurso dedutivo.

- Uma estrutura axiomática possui os seguintes elementos padrões:
  - A apresentação dos termos primitivos, que são os termos básicos utilizados no discurso, mas que não possuem uma definição.
  - A definição de termos derivados dos termos primitivos.
  - A apresentação de axiomas ou postulados, que se utilizam dos termos primitivos e que são declarações primárias, sem definição. Pode-se ainda fazer uma distinção entre axioma e postulado, considerando-se o primeiro como algo que parece intuitivo e o segundo como uma mera imposição.
  - A definição de um conjunto ou sistema de regras, que também é denominado de lógica ou sistema de lógica.
  - A definição de declarações derivadas dos axiomas/postulados, que se utilizam dos termos primitivos e dos derivados, sendo obtidas por meio da aplicação das regras sobre os axiomas/postulados. Tais declarações derivadas são chamadas de Teoremas.
- Quando um discurso possui a estrutura acima, mas os elementos são entidades abstratas, sem significado real, ele define um Ramo de Matemática Pura. Nesse caso, tem-se a denominada Axiomática Formal.
- Quando os elementos definidos acima possuem um significado real, o discurso é denominado de Modelo do Ramo de Matemática Pura, ou apenas um Ramo de Matemática Aplicada. Aqui, tem-se a chamada Axiomática Material.
- Por um lado, os axiomas/postulados e os teoremas podem ser identificados como declarações afirmativas ou proposições. Por outro lado, os termos primitivos podem ser interpretados como variáveis. Assim, dado que os axiomas/postulados são declarações que se utilizam dos termos primitivos, eles podem ser vistos como funções proposicionais de tais variáveis. Da mesma forma, como os teoremas são implicações lógicas dos axiomas/postulados, eles também podem ser identificados como funções proposicionais.
- Deve-se notar ainda que a própria lógica pode ser definida como uma estrutura axiomática, possuindo: termos primitivos, termos derivados, axiomas/postulados, um conjunto ou sistema de regras e teoremas.

4.2.4 Classificação das lógicas dedutivas

- As lógicas dedutivas podem ser divididas em: clássica, complementar e não-clássica.
- A lógica clássica também recebe outras denominações, tais como: Lógica Binária, Lógica Bivalente, Cálculo Proposicional e Cálculo de Predicados de Primeira Ordem.
- Exemplos de lógica complementar são: modal, deôntica e epistêmica.
- Alterando-se os princípios da lógica clássica, surgem as lógicas não-clássicas. Alguns exemplos são: paracompletas, intuicionistas, não-aléticas, não-reflexivas, probabilísticas, polivalentes, *fuzzy*.
- Tais conjuntos de lógicas não serão aqui apresentados, uma vez que isso foge ao objetivo principal do documento.
- Como foi dito anteriormente, apenas serão utilizados os resultados da lógica binária para a implementação de funções lógicas por circuitos eletro-eletrônicos.

4.3 Exemplo introdutório usando lógica clássica

- Definição do problema e suas motivações:
  - Modelagem de um sistema automático de tomada de decisão.
  - Verificação do uso de cinto de segurança em um automóvel.
- Objetivo da solução: O sonorizador deverá emitir um sinal de alarme se, e somente se, a ignição for acionada e a marcha for engatada, uma vez que os assentos frontais estejam ocupados e os respectivos cintos de segurança não estejam engatados.
- Infraestrutura existente:
  - Sensor de ignição.
  - Sensor de engate de marcha.
  - Sensor de presença em assentos dianteiros.
  - Sensor de engate de cintos de segurança dianteiros.
  - Atuador de sonorizador de alarme.
- Valores condicionais (mutuamente excludentes): F (*False*) e T (*True*).
- Declarações condicionais básicas (sentenças declarativas) e variáveis associadas:
  - Alarme deve soar: A.
  - Ignição está acionada: I.
  - Marcha está engatada: M.
  - Banco do motorista está ocupado: B_M.
  - Cinto do motorista está engatado: C_M.
  - Banco do carona está ocupado: B_C.
  - Cinto do carona está engatado: C_C.
- Declarações derivadas
  - Negação
    - Função: NOT ($\neg$).
    - Exemplo: Banco não está ocupado ($\neg B$).
  - Composição (ou conexão ou combinação)
    - Conjunção:
      - Função: AND ($\wedge$).
      - Exemplo: Ignição AND Marcha ($I \wedge M$).
    - Disjunção:
      - Função: OR ($\vee$).
      - Exemplo: Banco do motorista OR banco do carona ($B_M \vee B_C$).
    - Equivalência:
      - Função: XNOR ($\equiv$).
      - Exemplo: Alarme XNOR "Modelo proposto" ($A \equiv MP$).
- Proposta de modelo: $A \equiv I \wedge \{M \wedge [(B_M \wedge \neg C_M) \vee (B_C \wedge \neg C_C)]\}$.

4.4 Conceitos básicos

Os elementos utilizados em uma formulação funcional são os seguintes: valores fixos, variáveis e funções.

Os valores fixos representam os estados definidos na formulação e são descritos por uma simbologia adequada.

As variáveis carregam informação. A codificação da informação depende do significado que lhes é atribuído e dos estados que elas venham a assumir.

As funções realizam um mapeamento entre variáveis. De uma forma geral, os estados correntes de determinadas variáveis são utilizados para especificar um estado que será atribuído a uma determinada variável.

As funções são comumente descritas das seguintes formas: tabelas (para uma pequena quantidade de pontos), gráficos (para uma grande quantidade de pontos) e equações (para uma lei de formação explícita).

Deve ser ressaltado que os valores fixos, as variáveis e as funções, podem ser associados a qualquer tipo de informação, não necessariamente representando quantidades.

4.5 Formulação lógica clássica

- Nesse texto, é abordado apenas um tipo de lógica: binária, bivalente ou clássica.
- Os argumentos são formados por proposições.
- Uma proposição é uma sentença afirmativa declarativa (ou uma afirmação declarativa ou uma assertiva ou um *statement*), sobre a qual faz sentido afirmar-se que a mesma é verdadeira ou falsa.
- Variáveis e valores fixos na lógica binária:
  - As variáveis representam as assertivas (ou as proposições).
  - Só existem dois valores fixos que podem ser atribuídos a uma variável.
  - Os dois valores devem ser, do ponto de vista lógico, mutuamente excludentes.
- Modelagem lógica de um problema real:
  - A formulação de um problema real envolve diversas representações ou codificações.
  - Problema real → um sistema formado por um conjunto de assertivas (*statements*), associadas por meio de conectivos (operadores ou funções).
  - Conectivo → elemento de conexão que é modelado por uma função lógica.
  - Assertiva → afirmação declarativa (*statement*) sobre algum elemento do problema, a qual é representada por uma variável de asserção do sistema.
  - Variável de asserção → variável do sistema associada a uma assertiva, à qual será atribuído um valor fixo lógico (*truth value*).
  - Valor fixo lógico → representação do estado de uma variável de asserção por meio de um símbolo com significado lógico Por exemplo: F/T, F/V, 0/1, 0/5 ou +12/-12.
- Uma vez que os dois valores fixos possíveis são mutamente excludentes, naturalmente surge a idéia de negação (da associação de assertivas, do conectivo, da assertiva, da variável de asserção, do valor ou do símbolo).

4.6 Operadores lógicos clássicos

Um operador lógico clássico pode ser matematicamente definido por meio de uma função de variáveis lógicas (*truth function*). A função realiza um mapeamento entre um dado conjunto de variáveis lógicas em uma nova variável lógica, de tal forma que $V = f(V_1, V_2, \cdots, V_{N_v})$. Uma vez que as variáveis envolvidas em tais funções lógicas representam proposições, pode-se dizer que elas são, na realidade, funções proposicionais. Logo, a formalização matemática da lógica que se utiliza de tais funções lógicas/proposicionais (lógica clássica) recebe as seguintes denominações: cálculo de funções lógicas ou cálculo sentencial ou cálculo proposicional ou tautologia. Devido à pequena quantidade de valores assumidos pelas funções, a representação mais eficiente para funções lógicas é uma tabela, denominada de Tabela Verdade (*truth table*). Os dois valores lógicos possíveis para cada variável são simbolicamente representados por F (*false*) e T (*true*). Uma vez que as variáveis só podem asumir 2 valores, então, para um total de N_v variáveis, obtém-se um total de $N_c = 2^{N_v}$ combinações de valores e um total de $N_f = 2^{N_c}$ funções.

4.6.1 Tabela verdade (*truth table*)

Em uma tabela verdade, as colunas representam uma função lógica e as suas variáveis, enquanto as linhas representam os diversos padrões assumidos pelas variáveis e o valor da função para cada padrão, como é exemplificado na Figura 4.1. Como é ilustrado na mesma figura, não existe uma ordenação obrigatória para as linhas de uma tabela verdade. A ordenação é livre. Porém, a relação entre os padrões e os valores da função para cada padrão deve ser corretamente caracterizada,

Linha	A	B	f(A,B)
0	F	F	f(0)
1	F	T	f(1)
2	T	F	f(2)
3	T	T	f(3)

Ordenação 1.

Linha	A	B	f(A,B)
2	T	F	f(2)
0	F	F	f(0)
3	T	T	f(3)
1	F	T	f(1)

Ordenação 2.

Figura 4.1: Exemplos de tabela verdade para uma função lógica de duas variáveis, com duas ordenações de linhas diferentes.

Em alguns problemas, as funções podem ser completamente especificadas, de tal forma que todas as suas avaliações retornam apenas os valores F ou T, como é exemplificado na Tabela 4.1.

Em alguns outros problemas, as funções lógicas podem ser não completamente especificadas. Nesses casos, duas situações podem ocorrer. Na primeira delas, para uma dada combinação de valores das variáveis, o valor da função não é relevante, caracterizando uma indeterminação por *don't-care*. Por outro lado, pode acontecer que uma determinada combinação de valores das variáveis nunca ocorra, caracterizando uma indeterminação por *can't-happen*. Em ambas as situações, pode-se especificar livremente qualquer um dos valores lógicos para a função. Para caracterizar uma indeterminação, costuma-se atribuir um valor lógico indeterminado X. A Tabela 4.2 exemplifica uma função lógica com indeterminação.

Um valor lógico indeterminado X também pode ser utilizado para simplificar a representação de uma tabela verdade longa. Isso é ilustrado na Tabela 4.3, que apresenta a Tabela 4.1 na sua forma simplificada.

Linha	A	B	f(A,B)
0	F	F	T
1	F	T	T
2	T	F	F
3	T	T	F

Tabela 4.1: Exemplo de tabela verdade completamente especificada

Linha	A	B	f(A,B)
0	F	F	T
1	F	T	T
2	T	F	X
3	T	T	F

Tabela 4.2: Exemplo de tabela verdade não completamente especificada

Linha	A	B	f(A,B)
0,1	F	X	T
2,3	T	X	F

Tabela 4.3: Exemplo de tabela verdade simplificada

4.6.2 Funções de 1 variável

Nesse caso, o mapeamento realizado é definido por $X = f(A)$, de forma que $N_v = 1$ variável, $N_c = 2^{N_v} = 2$ combinações de valores e $N_f = 2^{N_c} = 4$ funções. Cada uma das funções lógicas de uma variável $X_i = f_i(A)$, para $0 \leq i \leq 3$, é definida por sua própria Tabela Verdade, as quais são agrupadas na Tabela 4.4. Os operadores lógicos associados às funções $X_i = f_i(A)$ são definidos na Tabela 4.5.

A
F
T

X_0
F
F

X_1
F
T

X_2
T
F

X_3
T
T

Tabela 4.4: Tabela de funções lógicas de uma variável: $X_i = f_i(A)$, para $0 \leq i \leq 3$.

Função	Operador		
	Operação	Notação	Nomenclatura
X_0	(F)	(F)	Contradição
X_1	(A)	(A)	Identidade lógica
X_2	$NOT(A)$	$\neg(A)$ ou $\sim(A)$ ou $\overline{(A)}$ ou $(A)'$ ou $(A)^*$ ou $!(A)$	Negação lógica
X_3	(T)	(T)	Tautologia

Tabela 4.5: Tabela de operadores lógicos de uma variável.

4.6.3 Funções de 2 variáveis

Aqui, o mapeamento realizado é definido por $X = f(A, B)$, de forma que $N_v = 2$ variáveis, $N_c = 2^{N_v} = 4$ combinações de valores e $N_f = 2^{N_c} = 16$ funções. Cada uma das funções lógicas de duas variáveis $X_i = f_i(A, B)$, para $0 \leq i \leq 15$, é definida por sua própria Tabela Verdade, as quais são agrupadas na Tabela 4.6. Os operadores lógicos associados às funções $X_i = f_i(A, B)$ são definidos na Tabela 4.7.

A	B	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	X_{12}	X_{13}	X_{14}	X_{15}
F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T
F	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T
T	F	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T
T	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T

Tabela 4.6: Tabela de funções lógicas de duas variáveis: $X_i = f_i(A, B)$, para $0 \leq i \leq 15$.

Função	Operador	
	Operação	Notação
X_0	(F)	(F)
X_1	$(A\ AND\ B)$	$(A \wedge B)$
X_2	$NOT(A\ IMPLICA\ B)$	$\neg(A \rightarrow B)$ ou $\neg(A \supset B)$
X_3	(A)	(A)
X_4	$NOT(B\ IMPLICA\ A)$	$\neg(A \leftarrow B)$ ou $\neg(A \subset B)$
X_5	(B)	(B)
X_6	$(A\ XOR\ B)$	$(A \veebar B)$
X_7	$(A\ OR\ B)$	$(A \vee B)$
X_8	$NOT(A\ OR\ B) \equiv (A\ NOR\ B)$	$\neg(A \vee B)$ ou $(A \downarrow B)$
X_9	$NOT(A\ XOR\ B) \equiv (A\ XNOR\ B)$	$\neg(A \veebar B)$ ou $(A \barwedge B)$ ou $(A \leftrightarrow B)$ ou $(A \equiv B)$
X_{10}	$NOT(B)$	$\neg(B)$
X_{11}	$(B\ IMPLICA\ A)$	$(A \leftarrow B)$ ou $(A \subset B)$
X_{12}	$NOT(A)$	$\neg(A)$
X_{13}	$(A\ IMPLICA\ B)$	$(A \rightarrow B)$ ou $(A \supset B)$
X_{14}	$NOT(A\ AND\ B) \equiv (A\ NAND\ B)$	$\neg(A \wedge B)$ ou $(A \uparrow B)$
X_{15}	(T)	(T)

Tabela 4.7: Tabela de operadores lógicos de duas variáveis.

É possível que se estabeleça uma relação entre a Lógica Binária e a Teoria de Conjuntos. Porém, deve-se ressaltar que, na equivalência entre os símbolos $\leftarrow$ e $\subset$, o significado do símbolo $\subset$ não é associado à relação de inclusão (*inclusion* ou *containment*) entre conjuntos. Na simbologia em questão, ele pode ser interpretado como uma versão estilizada da seta ou, ainda, como uma adaptação da letra "C", associada à relação de "conseqüência de".

4.6.4 Operadores lógicos básicos

Com base nas funções lógicas identificadas nas Tabelas 4.4 e 4.6, podem ser definidos os operadores lógicos encontrados nas Tabelas 4.5 e 4.7. Podem-se destacar os seguintes operadores unários: identidade lógica e NOT (negação lógica). Por sua vez, podem-se destacar os seguintes operadores (ou conectivos) binários: AND (E lógico), OR (OU-inclusivo lógico), XOR (OU-eXclusivo), IMPLICA (implicação lógica), NAND (NOT-AND), NOR (NOT-OR), XNOR (NOT-XOR ou bi-implicação ou equivalência lógica), e NOT-IMPLICA.

4.6.5 Funções de $N_v > 2$ variáveis

Pode-se mostrar que as funções de $N_v > 2$ variáveis podem ser definidas através da combinação das variáveis lógicas e das operações identificadas nas funções de 1 e 2 variáveis.

4.7 Exemplo de modelagem com relações de implicação

Podem-se definir as seguintes relações de implicação de uma assertiva precedente A para uma assertiva conseqüente B:

1. Condição necessária:
   "SOMENTE SE $(A = T)$ ENTÃO $(B = T)$"
   ou
   "$(B = T)$ SOMENTE SE $(A = T)$".

2. Condição suficiente:
   "SE (A = T) ENTÃO (B = T)"
   ou
   "(B = T) SE (A = T)".

3. Condição necessária e suficiente:
   "SE E SOMENTE SE (A = T) ENTÃO (B = T)"
   ou
   "(B = T) SE E SOMENTE SE (A = T)".

Portanto, pode-se estabelecer a seguinte modelagem:

1. Condição necessária: $(A \leftarrow B)$.

2. Condição suficiente: $(A \rightarrow B)$.

3. Condição necessária e suficiente: $(A \rightarrow B) \wedge (A \leftarrow B) \equiv (A \leftrightarrow B) \equiv (A \equiv B)$.

4.8 Tautologia e equivalência lógica

Uma proposição fundamental é aquela associada a uma variável lógica. A combinação de proposições fundamentais (variáveis lógicas), por meio de conectores, conectivos ou operadores lógicos (funções lógicas), gera uma nova proposição.

Uma proposição que possui todos os seus valores iguais a T, independentemente dos valores das suas proposições fundamentais, é dita uma Tautologia ou Lei da Lógica. Algumas tautologias são apresentadas na Tabela 4.8.

Se as tabelas verdade de duas proposições P_1 e P_2 forem iguais, as proposições são ditas logicamente equivalentes e a proposição $P_1 \leftrightarrow P_2$ é uma tautologia. Assim sendo, em qualquer proposição onde apareça P_1, essa última pode ter trocada por P_2, e vice-versa. As Tabelas 4.9, 4.10 e 4.11, ilustram alguns pares de proposições logicamente equivalentes, envolvendo os operadores lógicos NOT, AND, OR, IMPLICA e XNOR. Tais equivalências mostram que existe um interdefinição entre tais operadores. Portanto, alguns deles podem ser dispensados pelo uso dos demais. Isso é tratado a seguir.

Lei	Nome
$(p \vee \neg p)$	Lei do meio excluído
$\neg\,(p \wedge \neg p)$	Lei da contradição
$[(p \to q) \wedge (q \to r)] \to (p \to r)$	Lei do silogismo
$(p) \leftrightarrow \neg\,(\neg p)$	Lei da dupla negação
$(p \to q) \leftrightarrow (\neg q \to \neg p)$	Lei da contraposição

Tabela 4.8: Exemplos de Tautologias ou Leis da Lógica.

P_1	P_2
$(p \vee q)$	$\neg\,(\neg p \wedge \neg q)$
$(p \to q)$	$\neg\,(p \wedge \neg q)$
$(p \leftrightarrow q)$	$\neg\,(p \wedge \neg q) \wedge \neg\,(\neg p \wedge q)$

Tabela 4.9: Pares de proposições equivalentes, definindo os operadores OR, IMPLICA e XNOR, em função dos operadores NOT e AND.

P_1	P_2
$(p \wedge q)$	$\neg\,(\neg p \vee \neg q)$
$(p \to q)$	$(\neg p \vee q)$
$(p \leftrightarrow q)$	$\neg\,[\neg\,(\neg p \vee q) \vee \neg\,(p \vee \neg q)]$

Tabela 4.10: Pares de proposições equivalentes, definindo os operadores AND, IMPLICA e XNOR, em função dos operadores NOT e OR.

P_1	P_2
$(p \wedge q)$	$\neg\,(p \to \neg q)$
$(p \vee q)$	$(\neg p \to q)$
$(p \leftrightarrow q)$	$\neg\,[(p \to q) \to \neg\,(q \to p)]$

Tabela 4.11: Pares de proposições equivalentes, definindo os operadores AND, OR e XNOR, em função dos operadores NOT e IMPLICA.

4.9 Teoremas de De Morgan

Os teoremas de De Morgan descrevem uma relação direta de equivalência entre os operadores NOT, AND e OR. Eles podem ser enunciados da seguinte forma:

- (A NAND B) ≡ NOT (A AND B) ≡ (NOT A) OR (NOT B)
  ou
  $(A \uparrow B) \equiv \neg(A \wedge B) \equiv (\neg A) \vee (\neg B)$

- (A NOR B) ≡ NOT (A OR B) ≡ (NOT A) AND (NOT B)
  ou
  $(A \downarrow B) \equiv \neg(A \vee B) \equiv (\neg A) \wedge (\neg B)$

4.10 Exemplos de verificação de equivalências lógicas

As Tabelas 4.12 a 4.16 ilustram exemplos de verificação para algumas das equivalências lógicas apresentadas anteriormente.

p	q	$\neg q$	$\neg p$	$(\neg q \rightarrow \neg p)$	$(p \rightarrow q)$
F	F	T	T	T	T
F	T	F	T	T	T
T	F	T	F	F	F
T	T	F	F	T	T

Tabela 4.12: Verificação da equivalência lógica $(p \rightarrow q) \equiv (\neg q \rightarrow \neg p)$.

p	q	$\neg p$	$\neg q$	$(\neg p \wedge \neg q)$	$\neg\,(\neg p \wedge \neg q)$	$(p \vee q)$
F	F	T	T	T	F	F
F	T	T	F	F	T	T
T	F	F	T	F	T	T
T	T	F	F	F	T	T

Tabela 4.13: Verificação da equivalência lógica $(p \vee q) \equiv \neg\,(\neg p \wedge \neg q)$.

p	q	$\neg p$	$\neg q$	$(\neg p \vee \neg q)$	$\neg\,(\neg p \vee \neg q)$	$(p \wedge q)$
F	F	T	T	T	F	F
F	T	T	F	T	F	F
T	F	F	T	T	F	F
T	T	F	F	F	T	T

Tabela 4.14: Verificação da equivalência lógica $(p \wedge q) \equiv \neg\,(\neg p \vee \neg q)$.

A	B	$\neg A$	$\neg B$	$(\neg A \vee \neg B)$	$(A \uparrow B)$
F	F	T	T	T	T
F	T	T	F	T	T
T	F	F	T	T	T
T	T	F	F	F	F

Tabela 4.15: Verificação da equivalência lógica $(A \uparrow B) \equiv (\neg A \vee \neg B)$.

A	B	$\neg A$	$\neg B$	$(\neg A \wedge \neg B)$	$(A \downarrow B)$
F	F	T	T	T	T
F	T	T	F	F	F
T	F	F	T	F	F
T	T	F	F	F	F

Tabela 4.16: Verificação da equivalência lógica $(A \downarrow B) \equiv (\neg A \wedge \neg B)$.

4.11 Conjunto funcionalmente completo de operadores

Deve-se notar, nas Tabelas 4.4 a 4.7, que metade da funções X_k pode ser obtida através da aplicação do operador de negação lógica (NOT) sobre a outra metade. Além disso, deve-se notar que alguns operadores (conectivos) binários podem ser descritos por meio da combinação de outros operadores, conforme ilustrado nas Tabelas 4.9, 4.10 e 4.11, bem como nos Teoremas de De Morgan.

Portanto, as seguintes questões surgem naturalmente:

1. É possível descrever todos os demais operadores a partir de um determinado conjunto básico de operadores (conjunto completo)?

2. Todos os operadores de um dado conjunto completo são absolutamente necessários (operadores independentes entre si)?

3. Existe um conjunto mínimo de operadores que forme um conjunto completo de operadores independentes entre si (conjunto completo mínimo)?

As seguintes respostas podem ser demonstradas:

- Tentativa 1: {AND} → Não!
- Tentativa 2: {OR} → Não!
- Tentativa 3: {AND, OR} → Não!
- Tentativa 4: {AND, OR, NOT} → OK! → Conjunto completo, mas não mínimo...
- Tentativa 5: {AND, NOT} ou {NAND} → OK! → Conjunto completo e mínimo!
- Tentativa 6: {OR, NOT} ou {NOR} → OK! → Conjunto completo e mínimo!

Conforme indicado pela Tentativa 4, a Tabela 4.17 ilustra as funções lógicas de duas variáveis e suas expressões equivalentes, empregando os operadores AND, OR e NOT.

Função lógica $X_k = f_k(A, B)$	Expressão equivalente {AND, OR, NOT}
X_0	$(A \wedge \neg A) = (B \wedge \neg B) = F$
X_1	$(A \wedge B)$
X_2	$(A \wedge \neg B)$
X_3	A
X_4	$(\neg A \wedge B)$
X_5	B
X_6	$(A \wedge \neg B) \vee (\neg A \wedge B)$
X_7	$(A \vee B)$
X_8	$\neg (A \vee B) = (\neg A \wedge \neg B)$
X_9	$(\neg A \wedge \neg B) \vee (A \wedge B)$
X_{10}	$\neg B$
X_{11}	$(A \vee \neg B)$
X_{12}	$\neg A$
X_{13}	$(\neg A \vee B)$
X_{14}	$\neg (A \wedge B) = (\neg A \vee \neg B)$
X_{15}	$(A \vee \neg A) = (B \vee \neg B) = T$

Tabela 4.17: Funções lógicas de duas variáveis e suas expressões equivalentes, empregando os operadores AND, OR e NOT.

Uma vez que as Tentativas 5 e 6 apontam as funções NAND e NOR, de duas entradas, isoladamente como conjuntos completos mínimos, estas funções são comumente denominadas de funções (ou operadores ou portas ou *gates*) lógicas universais.

A partir da Tentativa 4, a Tentativa 5 pode ser exemplificada por meio da relação

$$(A \vee B) = \neg (\neg A \wedge \neg B) = \neg (\neg (A \wedge A) \wedge \neg (B \wedge B)) = (A \uparrow A) \uparrow (B \uparrow B) \ .$$

A partir da Tentativa 4, a Tentativa 6 pode ser exemplificada por meio da relação

$$(A \wedge B) = \neg (\neg A \vee \neg B) = \neg (\neg (A \vee A) \vee \neg (B \vee B)) = (A \downarrow A) \downarrow (B \downarrow B) \ .$$

Exemplos de expressões equivalentes, empregando os operadores binários NAND e NOR, podem ser encontrados no Apêndice A.

Os operadores NOT, AND e OR são naturalmente utilizados nas expressões lógicas elaboradas pelo ser humano. Além disso, os operadores AND e OR possuem a propriedade de associatividade. Os operadores NOT, NAND e NOR são facilmente implementados por dispositivos eletro-eletrônicos. Porém, os operadores NAND e NOR não são associativos. Assim sendo, é comum que se definam as expressões lógicas utilizando os operadores do conjunto {NOT, AND, OR} e, em seguida, se for desejado, que elas sejam convertidas em expressões equivalentes, empregando os operadores do conjunto alternativo {NOT, NAND, NOR}.

4.12 Decomposição em funções canônicas

Uma vez que as funções NOT, OR e AND, formam um conjunto funcionalmente completo, sabe-se que uma função lógica genérica pode ser decomposta em uma combinação qualquer de tais funções. Porém, pode-se definir uma decomposição sistemática e bem estruturada, empregando-se funções canônicas, dos tipos m e M.

Em uma função canônica do tipo m, apenas um dos valores da sua Tabela Verdade é T, enquanto todos os demais são F. Uma vez que, na decomposição de uma função genérica, ela será um termo com um número mínimo de valores T, a função do tipo m é denominada de mintermo. A função AND possui um único valor T na sua Tabela Verdade, quando ambas as entradas assumem o valor T. Logo, um mintermo pode ser montado a partir uma função AND, por meio de uma adequação das variáveis de entrada.

Em uma função canônica do tipo M, apenas um dos valores da sua Tabela Verdade é F, enquanto todos os demais são T. Uma vez que, na decomposição de uma função genérica, ela será um termo com um número máximo de valores T, a função do tipo M, é denominada de maxtermo. A função OR possui um único valor F na sua Tabela Verdade, quando ambas as entradas assumem o valor F. Logo, um maxtermo pode ser montado a partir uma função OR, por meio de uma adequação das variáveis de entrada.

Para funções de duas variáveis, $f(A, B)$, os mintermos m_i são definidos por:

- $m_0(A, B) = (\neg A \wedge \neg B)$;
- $m_1(A, B) = (\neg A \wedge B)$;
- $m_2(A, B) = (A \wedge \neg B)$;
- $m_3(A, B) = (A \wedge B)$.

Para funções de duas variáveis, $f(A, B)$, os maxtermos M_i são definidos por:

- $M_0(A, B) = (A \vee B)$;
- $M_1(A, B) = (A \vee \neg B)$;
- $M_2(A, B) = (\neg A \vee B)$;
- $M_3(A, B) = (\neg A \vee \neg B)$.

A Tabela 4.18 apresenta as funções canônicas (mintermos e maxtermos) para duas variáveis.

Linha
0
1
2
3

A	***B***
F	*F*
F	*T*
T	*F*
T	*T*

m_0	m_1	m_2	m_3
T	*F*	*F*	*F*
F	*T*	*F*	*F*
F	*F*	*T*	*F*
F	*F*	*F*	*T*

M_0	M_1	M_2	M_3
F	*T*	*T*	*T*
T	*F*	*T*	*T*
T	*T*	*F*	*T*
T	*T*	*T*	*F*

Tabela 4.18: Tabela de funções canônicas (mintermos e maxtermos) para duas variáveis.

Para sintetizar os diversos valores T da Tabela Verdade de uma função genérica, utilizam-se os mintermos correspondentes, combinados pelo operador lógico OR.

Para sintetizar os diversos valores F da Tabela Verdade de uma função genérica, utilizam-se os maxtermos correspondentes, combinados pelo operador lógico AND.

A título de exemplo, a Tabela 4.19 ilustra a decomposição da função $X(A,B) = (A \veebar B)$ de duas formas, que são as seguintes:

1. $X(A,B) = (A \veebar B) = m_1 \vee m_2 = (\neg A \wedge B) \vee (A \wedge \neg B)$.
2. $X(A,B) = (A \veebar B) = M_0 \wedge M_3 = (A \vee B) \wedge (\neg A \vee \neg B)$.

A	B	$A \veebar B$	m_1	m_2	M_0	M_3
F	F	F	F	F	F	T
F	T	T	T	F	T	T
T	F	T	F	T	T	T
T	T	F	F	F	T	F

Tabela 4.19: Exemplo de decomposição em funções canônicas (mintermos e maxtermos).

Deve-se ressaltar que, para uma determinada função alvo $X(\cdot)$, cada função m_i (ou M_i) utilizada na sua decomposição é responsável por sintetizar apenas um dos valores T (ou F) de $X(\cdot)$. Logo, o total de termos m somado ao total de termos M, usados na decomposição de $X(\cdot)$, é igual ao total de valores da sua Tabela Verdade. Deve-se perceber ainda que:

- É válida a seguinte equivalência: $\neg(m_i) = M_i$.
- Portanto, as relações $\neg m_3 = M_3$ e $\neg M_0 = m_0$ representam, ao mesmo tempo, uma prova para os Teoremas de De Morgan e uma outra forma de enunciá-los.

Por fim, deve-se notar que:

- Os mintermos são aplicações dos operadores NOT e AND sobre as variáveis. Por sua vez, as funções lógicas genéricas são aplicações dos operadores OR sobre os mintermos.
- Os maxtermos são aplicações dos operadores NOT e OR sobre as variáveis. Por sua vez, as funções lógicas genéricas são aplicações dos operadores AND sobre os maxtermos.
- Logo, a decomposição de uma função lógica genérica em mintermos ou em maxtermos é apenas um arranjo dos operadores NOT, OR e AND, por meio de uma estrutura bem definida.

No caso particular de funções de uma variável $X(A)$, a Tabela 4.20 mostra que são válidas as seguintes relações: $m_0 = \neg M_0 = \neg A$ e $m_1 = \neg M_1 = A$.

A	m_0	m_1	M_0	M_1
F	T	F	F	T
T	F	T	T	F

Tabela 4.20: Tabela de funções canônicas (mintermos e maxtermos) para uma variável.

4.13 Conectivos de ordem superior

Os operadores lógicos básicos anteriormente definidos são associados a funções que dependem de um ou dois parâmetros. Assim, os conectivos básicos são ditos operadores binários.

Apesar de um operador lógico *Op*(A,B,C,D) ser, por definição, um bloco funcional genérico, pode-se denominá-lo de um conectivo *Op* de ordem superior (a dois). Assim, um bloco funcional associado ao operador *Op*(A,B,C,D) é representado por uma porta *Op* de quatro entradas.

Por exemplo, o operador AND(A,B,C,D) gera o valor T para a combinação A=B=C=D=T e o valor F para as demais combinações. Possíveis implementações para o operador AND(A,B,C,D), empregando-se apenas blocos fundamentais do tipo AND (X,Y), são dadas por

$$AND(A,B,C,D) = AND(D, AND(C, AND(B,A))) \tag{4.1}$$

e por

$$AND(A,B,C,D) = AND(AND(D,C), AND(B,A)) \ . \tag{4.2}$$

4.14 Técnica de *bundling*

A técnica de *bundling* é uma operação elementar, que permite simplificar expressões de funções lógicas que contenham uma associação em três planos (ou níveis) de lógica com as seguintes seqüências de operadores: NAND $\rightarrow$ NOT $\rightarrow$ AND, NAND $\rightarrow$ NOT $\rightarrow$ NAND, NOR $\rightarrow$ NOT $\rightarrow$ OR e NOR $\rightarrow$ NOT $\rightarrow$ NOR.

Nesses casos, é possível agrupar (*bundle*) as variáveis do primeiro operador e passá-las diretamente para o segundo operador, dispensando o operador NOT intermediário.

Tal modificação pode acarretar um ganho no tempo de resposta e/ou na quantidade do circuito empregado na implementação.

Para as associações NAND $\rightarrow$ NOT $\rightarrow$ AND e NAND $\rightarrow$ NOT $\rightarrow$ NAND, podem ser citados os seguintes exemplos:

$$X(A,B,C) = A \wedge [\neg(B \uparrow C)] = A \wedge \{\neg[\neg(B \wedge C)]\} = A \wedge (B \wedge C) = (A \wedge B \wedge C) \ ,$$

$$X(A,B,C) = A \uparrow [\neg(B \uparrow C)] = \neg(A \wedge \{\neg[\neg(B \wedge C)]\}) = \neg[A \wedge (B \wedge C)] = (A \uparrow B \uparrow C)$$

e

$$\begin{cases} X(A,B,C) = A \uparrow [\neg(B \uparrow C)] = \neg(A \wedge \{\neg[\neg(B \wedge C)]\}) = \neg[A \wedge (B \wedge C)] = (A \uparrow B \uparrow C) \\ Y(B,C,D) = D \vee (B \uparrow C) \end{cases} .$$

Por sua vez, para as associações NOR $\rightarrow$ NOT $\rightarrow$ OR e NOR $\rightarrow$ NOT $\rightarrow$ NOR, podem ser citados os seguintes exemplos:

$$X(A,B,C) = A \vee [\neg(B \downarrow C)] = A \vee \{\neg[\neg(B \vee C)]\} = A \vee (B \vee C) = (A \vee B \vee C) \ ,$$

$$X(A,B,C) = A \downarrow [\neg(B \downarrow C)] = \neg(A \vee \{\neg[\neg(B \vee C)]\}) = \neg[A \vee (B \vee C)] = (A \downarrow B \downarrow C)$$

e

$$\begin{cases} X(A,B,C) = A \downarrow [\neg(B \downarrow C)] = \neg(A \vee \{\neg[\neg(B \vee C)]\}) = \neg[A \vee (B \vee C)] = (A \downarrow B \downarrow C) \\ Y(B,C,D) = D \wedge (B \downarrow C) \end{cases} .$$

4.15 Exemplos de aplicação direta de portas lógicas

As portas lógicas são elementos básicos na construção de circuitos digitais. Assim sendo, elas podem ser pensadas como os circuitos digitais mais simples. A seguir, são apresentados alguns exemplos de aplicação direta de portas lógicas.

4.15.1 Uso de operador lógico como elemento de controle

No projeto de circuitos digitais, é comum que se necessite de alguns elementos básicos de controle, os quais podem ser implementados através dos operadores lógicos.

Considerando-se que as variáveis lógicas A e B, bem como a função do operador $X(A,B)$, sejam respectivamente mapeadas nas variáveis de entrada E, de controle $CTRL$ e de saída $S(E,CTRL)$, e que elas assumam apenas os valores lógicos F e T, podem-se definir as ações de controle apresentadas na Tabela 4.21.

Uma simbologia genérica para tais operações pode ser visualizada na Figura 4.2.

Operador lógico: $X(A,B)$	Ação de controle: $S(E,CTRL)$
AND	$S = (CTRL \wedge E) = \begin{cases} F & , \ CTRL = F \\ E & , \ CTRL = T \end{cases}$
$NAND$	$S = (CTRL \uparrow E) = \begin{cases} T & , \ CTRL = F \\ \neg E & , \ CTRL = T \end{cases}$
OR	$S = (CTRL \vee E) = \begin{cases} E & , \ CTRL = F \\ T & , \ CTRL = T \end{cases}$
NOR	$S = (CTRL \downarrow E) = \begin{cases} \neg E & , \ CTRL = F \\ F & , \ CTRL = T \end{cases}$
XOR	$S = (CTRL \veebar E) = \begin{cases} E & , \ CTRL = F \\ \neg E & , \ CTRL = T \end{cases}$
$XNOR$	$S = (CTRL \equiv E) = \begin{cases} \neg E & , \ CTRL = F \\ E & , \ CTRL = T \end{cases}$
$IMPLICA$	$S = (CTRL \rightarrow E) = \begin{cases} T & , \ CTRL = F \\ E & , \ CTRL = T \end{cases}$
	$S = (CTRL \leftarrow E) = \begin{cases} \neg E & , \ CTRL = F \\ T & , \ CTRL = T \end{cases}$
$NOT\ IMPLICA$	$S = \neg(CTRL \rightarrow E) = \begin{cases} F & , \ CTRL = F \\ \neg E & , \ CTRL = T \end{cases}$
	$S = \neg(CTRL \leftarrow E) = \begin{cases} E & , \ CTRL = F \\ F & , \ CTRL = T \end{cases}$

Tabela 4.21: Uso de operador lógico como elemento de controle.

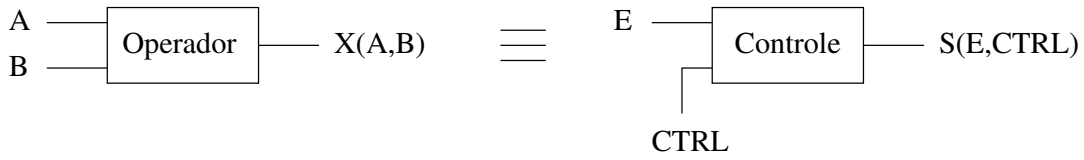


Figura 4.2: Uso de operador lógico como elemento de controle: simbologia genérica.

4.15.2 Identificador de paridade e gerador de paridade

De uma forma genérica, o termo PARIDADE pode ser tecnicamente utilizado para designar a contagem de valores F ou T em um conjunto de valores $\boldsymbol{V} = \{V_1, V_2, \cdots, V_N\}$, onde $V_k \in \{F, T\}$. Nesse sentido, paridade ímpar e paridade par indicam que o conjunto de valores $\boldsymbol{V}$ possui um número ímpar e um número par de valores F ou T, respectivamente.

Em algumas aplicações, pode ser necessário que se identifique a paridade de um dado conjunto de valores $\boldsymbol{V} = \{V_1, V_2, \cdots, V_N\}$. Em outras, um novo valor V_{N+1} deve ser gerado e agregado a um dado conjunto $\boldsymbol{V}$, de tal forma que o novo conjunto aumentado $\boldsymbol{V_P} = \{\boldsymbol{V}, V_{N+1}\}$ tenha a paridade desejada.

Analisando-se a Tabela Verdade dos operadores lógicos XOR(A,B) e XNOR(A,B), pode-se constatar que eles assumem o valor T quando existe em $\boldsymbol{V} = \{V_1, V_2\}$, respectivamente, um número ímpar de valores T e um número par de valores F.

Portanto, para o conjunto $\boldsymbol{V} = \{V_1, V_2\}$, os operadores XOR(A,B) e XNOR(A,B) podem ser usados como identificadores de paridade ímpar de valores T e paridade par de valores F, respectivamente.

Por outro lado, para o conjunto $\boldsymbol{V} = \{V_1, V_2\}$, os operadores XOR(A,B) e XNOR(A,B) podem ser usados como geradores do valor V_3, a fim de que o conjunto $\boldsymbol{V_P} = \{V_1, V_2, V_3\}$ apresente paridade par de valores T e paridade ímpar de valores F, respectivamente.

4.15.3 Identificador de igualdade entre padrões binários

Dadas duas cadeias de valores binários, $\boldsymbol{V} = \{V_1, V_2, \cdots, V_N\}$ e $\boldsymbol{W} = \{W_1, W_2, \cdots, W_N\}$, onde V_k e $W_k \in \{F, T\}$, pode ser necessário identificar se ambos os padrões são iguais.

Analisando-se a Tabela Verdade dos operadores lógicos XOR(A,B) e XNOR(A,B), pode-se constatar que eles assumem o valor T quando os valores de A e de B são diferentes ou iguais, respectivamente.

Portanto, para os conjuntos $\boldsymbol{V} = \{V_1\}$ e $\boldsymbol{W} = \{W_1\}$, os operadores XOR(A,B) e XNOR(A,B) podem ser usados como identificadores de diferença e de igualdade, respectivamente.

4.16 Blocos funcionais fundamentais

Para um operador lógico genérico, pode-se definir um bloco funcional que realize a operação lógica correspondente. Posteriormente, pode-se propor um sistema físico que implemente o bloco funcional desejado.

No caso dos operadores lógicos básicos, é adotada uma nomenclatura espefícica para os blocos funcionais a eles associados. Para a identidade lógica, o bloco é denominado de BUFFER. O bloco NOT é associado à negação lógica. Os demais operadores (AND, OR, XOR, NAND, NOR, XNOR) são associados a blocos funcionais que recebem a denominação genérica de porta lógica (*logic gate*).

Blocos funcionais fundamentais, associados aos operadores lógicos básicos, são ilustrados na Figura 4.3. Os símbolos usados nessa figura, que apresentam distinção de formato, são definidos no padrão IEEE Standard No.91 (ANSI Y 32.14, 1973). Um outro conjunto de símbolos, ilustrados na Figura 4.4 e que apresentam um formato uniforme, foi estabelecido pelo *International Electrotechnical Comission* (Publicação IEC 117-15) e foi incluído no mesmo padrão.

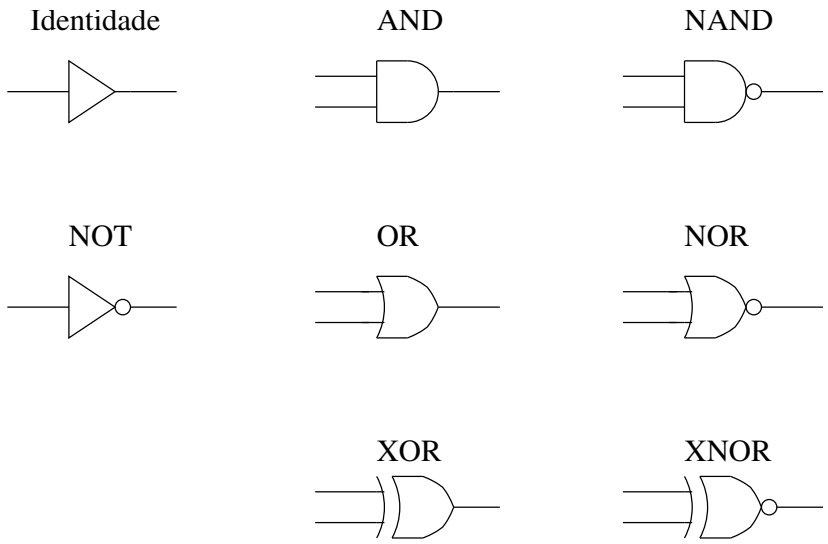


Figura 4.3: Blocos funcionais fundamentais, associados aos operadores lógicos básicos (IEEE).

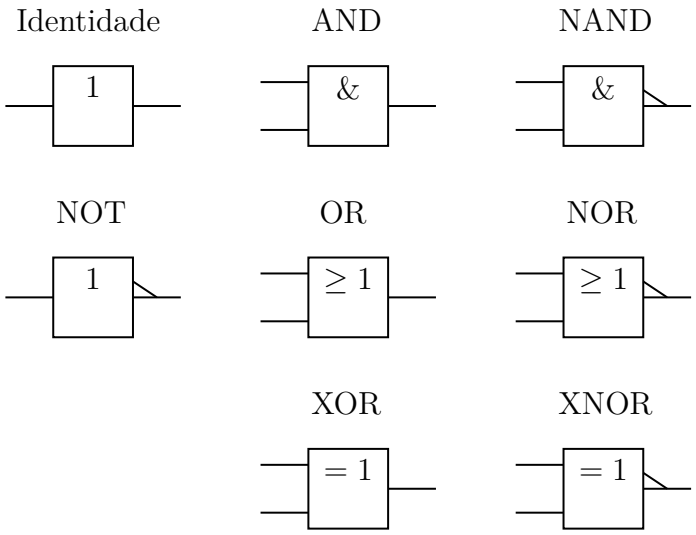


Figura 4.4: Blocos funcionais fundamentais, associados aos operadores lógicos básicos (IEC).

4.17 Manipulação algébrica de blocos

Os blocos funcionais são uma representação alternativa para os operadores lógicos. Por sua vez, os desenhos esquemáticos que contêm blocos funcionais, denominados de circuitos lógicos, são uma representação alternativa para as equações envolvendo operadores lógicos. Portanto, pode-se realizar uma manipulação diretamente sobre os blocos lógicos de um circuito, equivalentemente àquela realizada sobre os operadores lógicos de uma equação. Dois exemplos de manipulação algébrica de equações são mostrados nas Equações (4.3) e (4.4). Dois exemplos de manipulação algébrica de blocos, referentes às manipulações das Equações (4.3) e (4.4), são mostrados nas Figuras 4.5 e 4.6, respectivamente.

$$
\begin{aligned}
X(A,B) &= A \veebar B \\
&= (\neg A \wedge B) \vee (A \wedge \neg B) \\
&= \neg\left(\neg\left((\neg A \wedge B) \vee (A \wedge \neg B)\right)\right) \\
&= \neg\left((\neg A \wedge B) \downarrow (A \wedge \neg B)\right) \\
&= \neg\left((\neg\left(\neg A \wedge B\right)) \wedge (\neg\left(A \wedge \neg B\right))\right) \\
&= (\neg A \uparrow B) \uparrow (A \uparrow \neg B)\ . \qquad (4.3)
\end{aligned}
$$

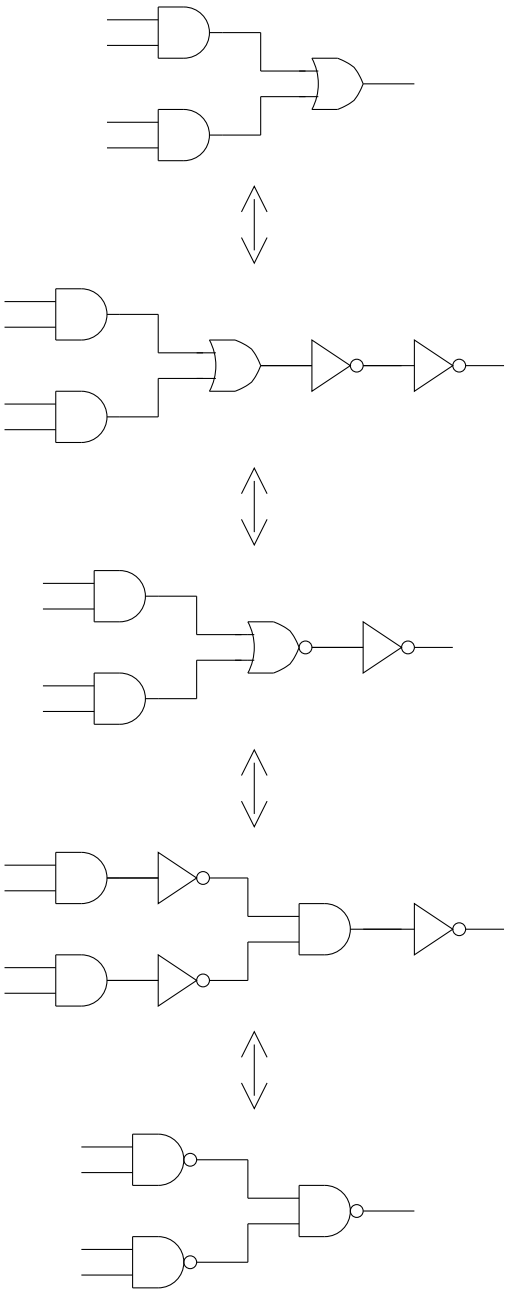


Figura 4.5: Manipulação algébrica de blocos: exemplo 1.

$$
\begin{aligned}
X(A,B) &= A \veebar B \\
&= (A \vee B) \wedge (\neg A \vee \neg B) \\
&= \neg\left(\neg\left((A \vee B) \wedge (\neg A \vee \neg B)\right)\right) \\
&= \neg\left((A \vee B) \uparrow (\neg A \vee \neg B)\right) \\
&= \neg\left((\neg\,(A \vee B)) \vee (\neg\,(\neg A \vee \neg B))\right) \\
&= (A \downarrow B) \downarrow (\neg A \downarrow \neg B)\ . \qquad (4.4)
\end{aligned}
$$

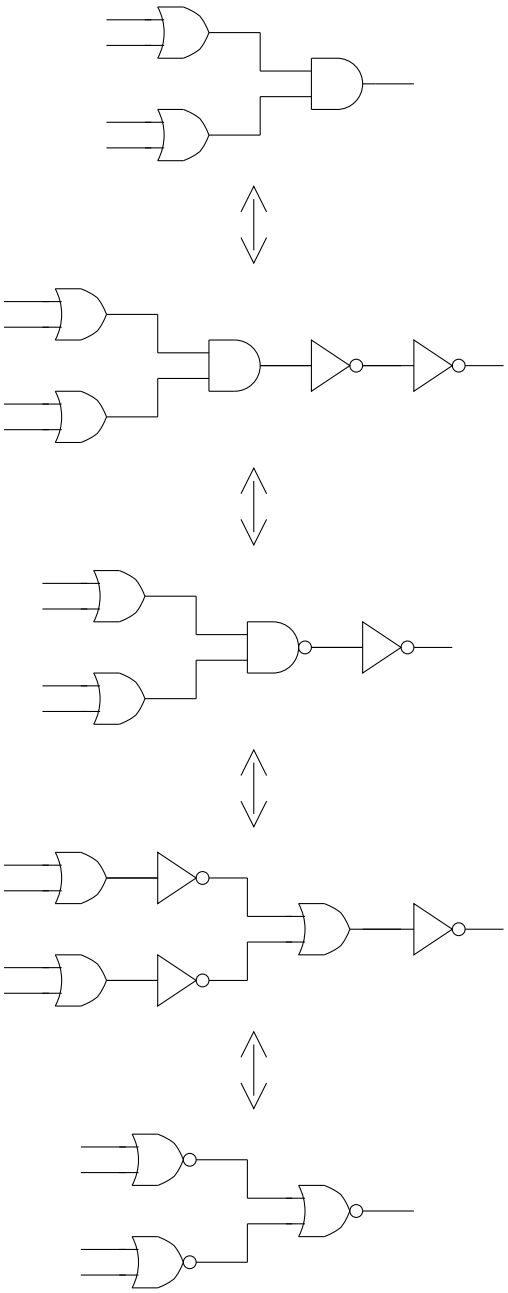


Figura 4.6: Manipulação algébrica de blocos: exemplo 2.

4.18 Exercícios propostos

1. Para cada operador binário $Op \in \{AND, OR, XOR, NAND, NOR, XNOR\}$, atenda aos seguintes itens:

   (a) Calcule o resultados das seguintes operações:

      i. $A\ Op\ F$.
      ii. $A\ Op\ T$.
      iii. $A\ Op\ A$.
      iv. $A\ Op\ \neg A$.

   (b) Mostre que, para tais operadores, valem as seguintes propriedades:

      i. Comutatividade: $A\ Op\ B = B\ Op\ A$.
      ii. Associatividade: $A\ Op\ (B\ Op\ C) = (A\ Op\ B)\ Op\ C$.
      iii. Distributividade com os demais operadores:
      $A\ Op_i\ (B\ Op_j\ C) = (A\ Op_i\ B)\ Op_j\ (A\ Op_i\ C)$. para $i \neq j$.

   (c) Calcule os seguintes elementos particulares:

      i. Elemento neutro (E_n): $A\ Op\ E_n = A$.
      ii. Elemento inverso (E_{inv}): $A\ Op\ E_{inv} = E_n$.

2. Considerando que todas as funções lógicas sejam descritas pela combinação dos operadores unário e binários, prove que:

   (a) Os conjuntos {AND}, {OR} e {AND, OR}, não são conjuntos completos.
   (b) O conjunto {AND, OR, NOT} é um conjunto completo.
   (c) O conjunto {AND, OR, NOT} não é um conjunto completo mínimo.
   (d) O conjunto {AND, NOT} e {NAND} é um conjunto completo mínimo.
   (e) O conjunto {OR, NOT} e {NOR} é um conjunto completo mínimo.

3. Escreva as funções de todos os conectivos binários utilizando apenas as seguintes funções básicas:

   (a) $\{AND, NOT\}$.
   (b) $\{OR, NOT\}$.
   (c) $NAND$.
   (d) NOR.

4. Prove as tautologias da Tabela 4.8.

5. Prove as equivalências das Tabelas 4.9, 4.10 e 4.11.

6. Prove os Teoremas de De Morgan.

7. Decomponha as funções de todos os conectivos binários como combinações das funções básicas m_i (mintermos), associadas pelo conectivo OR.

8. Decomponha as funções de todos os conectivos binários como combinações das funções básicas M_i (maxtermos), associadas pelo conectivo AND.

9. Dados os mintermos $m_i(A, B)$ e os maxtermos $M_i(A, B)$, atenda aos seguintes itens:
   (a) Descreva cada mintermo $m_i(A, B)$ em função dos demais mintermos $m_j(A, B)$, onde $i \neq j$.
   (b) Descreva cada mintermo $m_i(A, B)$ em função dos maxtermos $M_j(A, B)$.
   (c) Descreva cada maxtermo $M_i(A, B)$ em função dos demais maxtermos $M_j(A, B)$, onde $i \neq j$.
   (d) Descreva cada maxtermo $M_i(A, B)$ em função dos mintermos $m_j(A, B)$.

10. Para os exercícios listados abaixo, considere as equações lógicas apresentadas em seguida.

    Exercícios:

    (a) Desenhe um Diagrama de Blocos Funcionais equivalente, para cada uma das equações lógicas fornecidas.
    (b) Decomponha cada uma das equações lógicas fornecidas como combinação das funções básicas m_i (mintermos), associadas pelo conectivo OR.
    (c) Decomponha cada uma das equações lógicas fornecidas como combinação das funções básicas M_i (maxtermos), associadas pelo conectivo AND.
    (d) Desenhe um Diagrama de Blocos Funcionais equivalente, para cada uma das decomposições pedidas anteriormente.

    Equações lógicas:
    i. $F(A, B) = (A \vee B) \wedge (A \vee \neg B) \wedge (\neg A \vee B)$.
    ii. $F(A, B) = (\neg A \vee B) \wedge (A \vee \neg B) \wedge (\neg A \vee \neg B)$.
    iii. $F(A, B) = (A \wedge \neg B) \vee (\neg A \wedge B) \vee (A \wedge B)$.
    iv. $F(A, B) = (\neg A \wedge \neg B) \vee (\neg A \wedge B) \vee (A \wedge \neg B)$.
    v. $F(A, B, C, D) = (A \vee B) \wedge (\neg(C \wedge D))$.

11. Para cada operador binário $Op \in \{AND, OR, XOR, NAND, NOR, XNOR\}$ e $N \geq 2$, atenda aos seguintes itens:

    (a) Calcule o resultado da seguinte seqüência de operações:

    $$(V_N \; Op \; (\cdots (V_3 \; Op \; (V_2 \; Op \; V_1)) \cdots)) \; .$$

    (Sugestão: Calcule os casos particulares onde $N = 2, 3, 4, 5$ e aplique o Princípio de Indução Matemática para obter o resultado do caso geral.)
    (b) Considere que $Op(V_1, V_2, \cdots, V_N)$ é um operador de ordem N e verique a validade da seguinte relação:

    $$Op(V_1, V_2, \cdots, V_N) = (V_N \; Op \; (\cdots (V_3 \; Op \; (V_2 \; Op \; V_1)) \cdots)) \; .$$

12. Verifique a validade das seguintes relações:
    (a) $A \; XOR \; B = (\neg A \; AND \; B) \; OR \; (A \; AND \; \neg B) = (A \; OR \; B) \; AND \; (\neg A \; OR \; \neg B)$.
    (b) $A \; XNOR \; B = (A \; AND \; B) \; OR \; (\neg A \; AND \; \neg B) = (\neg A \; OR \; B) \; AND \; (A \; OR \; \neg B)$.
    (c) $A \; XOR \; B = \neg(A \; XNOR \; B) = \neg A \; XNOR \; B = A \; XNOR \; \neg B$.
    (d) $A \; XNOR \; B = \neg(A \; XOR \; B) = \neg A \; XOR \; B = A \; XOR \; \neg B$.

13. Dadas as seguintes relações:

    (a) $R_1 = (A\ XOR\ B)$,
    (b) $R_2 = (A\ XOR\ \neg B)$,
    (c) $R_3 = (\neg A\ XOR\ B)$,
    (d) $R_4 = (\neg A\ XOR\ \neg B)$,
    (e) $R_5 = (A\ XNOR\ B)$,
    (f) $R_6 = (A\ XNOR\ \neg B)$,
    (g) $R_7 = (\neg A\ XNOR\ B)$,
    (h) $R_8 = (\neg A\ XNOR\ \neg B)$,

    atenda aos seguintes itens:

    - calcule a **tabela verdade** para cada uma delas;
    - expresse cada uma delas por meio de uma combinação de **mintermos**;
    - expresse cada uma delas por meio de uma combinação de **maxtermos**;

    e identifique as equivalências existentes.

Capítulo 5

Álgebra de Boole

5.1 Introdução

- A implementação de um sistema digital apresenta um custo.
- Por razões óbvias, sempre é desejado o menor custo possível.
- A complexidade da implementação é um dos itens associados ao seu custo.
- Logo, deve-se minimizar a implementação a fim de se reduzir o seu custo.
- Na modelagem de um problema por equações lógicas, podem-se obter inúmeras equações lógicas equivalentes.
- Nesse caso, visando minimizar o custo da implementação, é interessante que se encontre a menor equação lógica possível, a fim de se alcançar a menor implementação possível.
- O Cálculo Proposicional não apresenta ferramentas adequadas para encontrar a função lógica mínima, dentro de um conjunto de equações lógicas equivalentes.
- Assim, torna-se necessário definir uma representação para a lógica empregada, a qual forneça ferramentas para a minimização das funções lógicas.
- Tais ferramentas podem ser encontradas na álgebra abstrata.
- A álgebra pode ser definida como o ramo da matemática que estuda as generalizações dos conceitos e das operações da aritmética.
- Em álgebra abstrata definem-se estruturas abstratas que representam, de uma forma global, diversas estruturas encontradas na prática.
- Uma estrutura algébrica adequada para a formulação, a manipulação e a minimização de funções lógicas foi inicialmente proposta por George Boole, a qual será tratada neste capítulo.

5.2 Postulados de Huntington

- Na definição de estruturas algébricas abstratas, é utilizado um processo axiomático.
- Em um sistema axiomático, são estabelecidos os seguintes itens:
  - um conjunto de elementos;
  - um determinado número de operações;
  - alguns elementos particulares;
  - algumas propriedades.
- Na associação de uma determinada estrutura abstrata com um determinado sistema existente são definidos:
  - os elementos que compõem os conjuntos;
  - o funcionamento das operações;
  - os elementos particulares.
- Um sistema axiomático pode ser definido de diversas maneiras equivalentes entre si.
- Entre as diversas formas de abordar a estrutura proposta por Boole, uma das mais utilizadas são os Postulados de Huntington, apresentados a seguir.
- Deve ser ressaltado que, nesses postulados, os símbolos denotam itens puramente abstratos. Assim sendo, as operações abstratas "+" e "·" não significam as operações aritméticas básicas de adição e multiplicação. Por sua vez, os símbolos "0" e "1" não representam quantidades, uma vez que não é definido um tipo particular para os elementos.
- Postulados de Huntington
  1. Existe um conjunto $\boldsymbol{K}$ de objetos ou elementos, sujeito a uma relação de equivalência, denotada pelo símbolo "=", que satisfaz ao princípio da substituição.
  2. É definida uma operação, denotada por "+", tal que, dados a e $b \in \boldsymbol{K}$, $(a+b) \in \boldsymbol{K}$.
     É definida uma operação, denotada por "·", tal que, dados a e $b \in \boldsymbol{K}$, $(a \cdot b) \in \boldsymbol{K}$.
  3. Existe um elemento $0 \in \boldsymbol{K}$, tal que, para cada $a \in \boldsymbol{K}$, $(a + 0) = a$.
     Existe um elemento $1 \in \boldsymbol{K}$, tal que, para cada $a \in \boldsymbol{K}$, $(a \cdot 1) = a$.
  4. As seguintes relações de comutatividade são válidas:
     $(a + b) = (b + a)$
     $(a \cdot b) = (b \cdot a)$
  5. As seguintes relações de distributividade são válidas:
     $a + (b \cdot c) = (a + b) \cdot (a + c)$
     $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
  6. Para cada elemento $a \in \boldsymbol{K}$ existe um elemento $\overline{a} \in \boldsymbol{K}$, tal que:
     $a + \overline{a} = 1$
     $a \cdot \overline{a} = 0$
  7. Deve haver, pelo menos, um total de dois elementos a e $b \in \boldsymbol{K}$, tal que $a \neq b$.

5.3 Versão alternativa para os Postulados de Huntington

- A Álgebra Booleana é um conjunto $B = \{a, b, c, \cdots\}$ com duas operações binárias, $\cup$ (*cup*) e $\cap$ (*cap*), satisfazendo os seguintes postulados:

  1. Cada uma das operações binárias é comutativa.
     Assim, dados a e $b \in \boldsymbol{B}$, tem-se que:
     $(a \cup b) = (b \cup a)$
     $(a \cap b) = (b \cap a)$

  2. Existem dois elementos distintos, z e $u \in B$, respectivamente relativos às operações $\cup$ e $\cap$, tal que:
     $(a \cup z) = a$
     $(a \cap u) = a$

  3. Cada uma das operações é relativamente distributiva à outra, de tal forma que:
     $a \cup (b \cap c) = (a \cup b) \cap (a \cup c)$
     $a \cap (b \cup c) = (a \cap b) \cup (a \cap c)$

  4. Para cada elemento $a \in \boldsymbol{B}$ existe um elemento $\overline{a} \in \boldsymbol{B}$, tal que:
     $a \cup \overline{a} = u$
     $a \cap \overline{a} = z$

- Deve ser ressaltado que, nesses postulados, os símbolos denotam itens puramente abstratos. Assim sendo, as operações abstratas "$\cup$" e "$\cap$" não significam as operações básicas sobre conjuntos conhecidas respectivamente como união e interseção. Por sua vez, os símbolos "z" e "u" não representam respectivamente os conjuntos Vazio e Universo.

5.4 Dualidade

- A dualidade é a característica daquilo que é dual, o que significa ser composto por duas unidades ou dois elementos.

- Pode-se observar que alguns dos Postulados de Huntington são apresentados em pares.

- Em cada par, um postulado pode ser obtido através do outro, efetuando-se a troca das operações "$+$" e "$\cdot$", bem como dos elementos 0 e 1.

- Cada teorema relacionado à estrutura algébrica de Boole possui um teorema dual.

- Ao usar a dualidade sobre a prova de um teorema, pode-se facilmente provar o seu dual.

- A dualidade é normalmente expressa por meio de um teorema. Sendo um teorema sobre teoremas, ela faz parte do que é chamado de metamatemática.

5.5 Lemas e teoremas fundamentais

- Nessa seção, apresentam-se lemas e teoremas para a estrutura algébrica de Boole.

- Os postulados podem ser entendidos como proposições fundamentais do sistema axiomático, enquanto os lemas e os teoremas são proposições derivadas a partir das proposições fundamentais, por meio de regras dedutivas.

- Os lemas são resultados intermediários das provas dos teoremas.

- Os teoremas podem ser usados como ferramentas para resolução de problemas.

- Os lemas e os teoremas, apresentados a seguir, podem ser demonstrados a partir dos Postulados de Huntington, definidos anteriormente.

- Lemas:

  1. Os elementos 0 e 1 são únicos.
  2. Para cada $a \in \boldsymbol{K}$, $(a + a) = a$ e $(a \cdot a) = a$.
  3. Para cada $a \in \boldsymbol{K}$, $(a + 1) = 1$ e $(a \cdot 0) = 0$.
  4. Os elementos 0 e 1 são distintos e $\overline{1} = 0$.
  5. Para cada par a e $b \in \boldsymbol{K}$, $a + (a \cdot b) = a$ e $a \cdot (a + b) = a$.
  6. O elemento $\overline{a}$, definido no Postulado 6, é único, para cada $a \in \boldsymbol{K}$.
  7. Para cada $a \in \boldsymbol{K}$, $a = \overline{(\overline{a})}$.
  8. Para quaisquer três elementos a, b e $c \in \boldsymbol{K}$, $a \cdot [(a + b) + c] = [(a + b) + c] \cdot a = a$.

- Teoremas:

  1. Para quaisquer três elementos a, b e $c \in \boldsymbol{K}$,
     $a + (b + c) = (a + b) + c$
     $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
  2. Para cada par a e $b \in \boldsymbol{K}$,
     $a + (\overline{a} \cdot b) = (a + b)$
     $a \cdot (\overline{a} + b) = (a \cdot b)$
  3. Para cada par a e $b \in \boldsymbol{K}$,
     $\overline{a + b} = \overline{a} \cdot \overline{b}$
     $\overline{a \cdot b} = \overline{a} + \overline{b}$
  4. Para quaisquer três elementos a, b e $c \in \boldsymbol{K}$, $(a \cdot b) + (\overline{a} \cdot c) + (b \cdot c) = (a \cdot b) + (\overline{a} \cdot c)$.

5.6 Definição de uma estrutura algébrica particular

De acordo com o Postulado 7, apresentado acima, o menor conjunto K possível é aquele que possui dois elementos. Por sua vez, uma estrutura algébrica que possa ser associada ao Cálculo Proposicional também deve possuir um conjunto K com dois elementos. Portanto, para modelar algebricamente o Cálculo Proposicional, pode-se definir a seguinte estrutura algébrica:

- Elementos: $\boldsymbol{K} = \{0, 1\}$.
- Complementos: $\begin{cases} 0 = \overline{1} \\ 1 = \overline{0} \end{cases}$.
- Identidades: $\begin{cases} 1 \cdot 1 = 1 + 1 = 1 + 0 = 0 + 1 = 1 \\ 0 + 0 = 0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0 \end{cases}$.

5.7 Exemplos de associação com a estrutura algébrica de Boole

- Cálculo Proposicional: a associação entre o Cálculo Proposicional e a estrutura algébrica de Boole é apresentada na Tabela 5.1.
- Teoria de conjuntos: a associação entre a teoria de conjuntos e a estrutura algébrica de Boole é apresentada na Tabela 5.2.
- Das Tabelas 5.1 e 5.2, pode-se concluir que o Cálculo Proposicional e Teoria de Conjuntos compartilham a mesma estrutura algébrica de Boole.

Cálculo Proposicional	Álgebra de Boole
$\wedge$	$\cdot$
$\vee$	$+$
F	0
T	1
$\neg(S)$	$\overline{S}$

Tabela 5.1: Tabela de mapeamento: Cálculo Proposicional × Álgebra de Boole.

Teoria de Conjuntos	Álgebra de Boole
$\cap$	$\cdot$
$\cup$	$+$
S_Z	0
S_U	1
$C(S)$	$\overline{S}$

Tabela 5.2: Tabela de mapeamento: Teoria de Conjuntos × Álgebra de Boole.

5.8 Isomorfismo

- Sistemas que são modelados pela mesma estrutura algébrica são ditos sistemas isomórficos.
- Sistemas isomórficos podem ser mapeados uns nos outros, por intermédio da estrutura compartilhada.
- As operações realizadas em cada sistema isomórfico são equivalentes e podem ser relacionadas entre si.
- As ferramentas existentes em um sistema isomófico podem ser usadas para resolver problemas nos sistemas equivalentes, através do mapeamento adequado.

5.9 Simplificação algébrica de expressões lógicas

- As expressões lógicas, provenientes de problemas reais e que devem ser implementadas com o menor custo possível, podem ser simplificadas por meio das ferramentas da álgebra abstrata (Postulados, Lemas e Teoremas).
- Inicialmente, deve-se mapear o sistema lógico (Cálculo Proposicional) em um sistema algébrico (estrutura algébrica de Boole).
- Em seguida, pode-se realizar a manipulação algébrica das expressões lógicas mapeadas, a fim de reduzí-las a formas mais simples e, conseqüentemente, reduzir o custo de sua implementação.
- As ferramentas algébricas utilizadas são os postulados, os lemas e os teoremas da estrutura algébrica com a qual se esteja trabalhando.
- A manipulação algébrica não sistemática depende da habilidade do profissional e não é diretamente automatizável, uma vez que não há uma definição inicial nem de qual conjunto de ferramentas nem de qual item do conjunto deva ser utilizado, nem mesmo em qual ordem.
- Portanto, faz-se necessário utilizar algum mecanismo mais adequado à atividade de simplificação das expressões.
- Um exemplo de manipulação algébrica não sistemática é apresentado a seguir.

5.10 Exemplo de manipulação algébrica não sistemática: Postulados, Lemas, Teoremas

Inicialmente, usando a Tabela 5.1, a expressão lógica $(A \vee B) \wedge (A \vee \neg B) \wedge (\neg A \vee B)$ é mapeada na expressão booleana $(A+B)\cdot(A+\overline{B})\cdot(\overline{A}+B)$. Em seguida, as Equações (5.1) a (5.3) ilustram possíveis manipulações algébricas, não sistemáticas, da expressão booleana, a fim de minimizá-la. É fácil perceber, por essas equações, que, dependendo das escolhas realizadas, há uma grande diferença no esforço dispendido. Além disso, não há qualquer garantia de que a expressão final seja a expressão mínima, ou de que a expressão mínima será alcançada.

$$
\begin{aligned}
&(A+B)\cdot(A+\overline{B})\cdot(\overline{A}+B) \\
&\qquad\downarrow\ P5 \\
&([(A+B)\cdot(A)]+[(A+B)\cdot(\overline{B})])\cdot(\overline{A}+B) \\
&\qquad\downarrow\ P5\ /\ P5 \\
&([(A\cdot A)+(B\cdot A)]+[(A\cdot\overline{B})+(B\cdot\overline{B})])\cdot(\overline{A}+B) \\
&\qquad\downarrow\ L2 \\
&([A+(B\cdot A)]+[(A\cdot\overline{B})+(B\cdot\overline{B})])\cdot(\overline{A}+B) \\
&\qquad\downarrow\ P6 \\
&([A+(B\cdot A)]+[(A\cdot\overline{B})+0])\cdot(\overline{A}+B) \\
&\qquad\downarrow\ P4 \\
&([A+(A\cdot B)]+[(A\cdot\overline{B})+0])\cdot(\overline{A}+B) \\
&\qquad\downarrow\ P3 \\
&([A+(A\cdot B)]+[(A\cdot\overline{B})])\cdot(\overline{A}+B) \\
&\qquad\downarrow\ L5 \\
&\left[A+(A\cdot\overline{B})\right]\cdot(\overline{A}+B) \\
&\qquad\downarrow\ P5 \\
&\left[A\cdot(\overline{A}+B)\right]+\left[(A\cdot\overline{B})\cdot(\overline{A}+B)\right] \\
&\qquad\downarrow\ P5\ /\ P5 \\
&(A\cdot\overline{A})+(A\cdot B)+(A\cdot\overline{B}\cdot\overline{A})+(A\cdot\overline{B}\cdot B) \\
&\qquad\downarrow\ P6 \\
&0+(A\cdot B)+(A\cdot\overline{B}\cdot\overline{A})+(A\cdot\overline{B}\cdot B) \\
&\qquad\downarrow\ P4 \\
&0+(A\cdot B)+(A\cdot\overline{A}\cdot\overline{B})+(A\cdot\overline{B}\cdot B) \\
&\qquad\downarrow\ P4 \\
&0+(A\cdot B)+(A\cdot\overline{A}\cdot\overline{B})+(A\cdot B\cdot\overline{B}) \\
&\qquad\downarrow\ P6 \\
&0+(A\cdot B)+(0\cdot\overline{B})+(A\cdot B\cdot\overline{B}) \\
&\qquad\downarrow\ P6 \\
&0+(A\cdot B)+(0\cdot\overline{B})+(A\cdot 0) \\
&\qquad\downarrow\ P4 \\
&0+(A\cdot B)+(\overline{B}\cdot 0)+(A\cdot 0) \\
&\qquad\downarrow\ L3 \\
&0+(A\cdot B)+(\overline{B}\cdot 0)+0 \\
&\qquad\downarrow\ L3 \\
&0+(A\cdot B)+0+0 \\
&\qquad\downarrow\ P4 \\
&(A\cdot B)+0+0+0 \\
&\qquad\downarrow\ P3 \\
&(A\cdot B)+0+0 \\
&\qquad\downarrow\ P3 \\
&(A\cdot B)+0 \\
&\qquad\downarrow\ P3 \\
&(A\cdot B)
\end{aligned}
\tag{5.1}
$$

$$
\begin{aligned}
&(A+B)\cdot(A+\overline{B})\cdot(\overline{A}+B) \\
&\qquad\qquad\downarrow\ P5 \\
&\big[A+(B\cdot\overline{B})\big]\cdot(\overline{A}+B) \\
&\qquad\qquad\downarrow\ P6 \\
&(A+0)\cdot(\overline{A}+B) \\
&\qquad\qquad\downarrow\ P3 \\
&A\cdot(\overline{A}+B) \\
&\qquad\qquad\downarrow\ P5 \\
&(A\cdot\overline{A})+(A\cdot B) \\
&\qquad\qquad\downarrow\ P6 \\
&0+(A\cdot B) \\
&\qquad\qquad\downarrow\ P4 \\
&(A\cdot B)+0 \\
&\qquad\qquad\downarrow\ P3 \\
&(A\cdot B)
\end{aligned}
\tag{5.2}
$$

$$
\begin{aligned}
&(A+B)\cdot(A+\overline{B})\cdot(\overline{A}+B) \\
&\qquad\qquad\downarrow\ L2 \\
&(A+B)\cdot(A+B)\cdot(A+\overline{B})\cdot(\overline{A}+B) \\
&\qquad\qquad\downarrow\ P4 \\
&\big[(A+B)\cdot(A+\overline{B})\big]\cdot\big[(A+B)\cdot(\overline{A}+B)\big] \\
&\qquad\qquad\downarrow\ P5 \\
&\big[A+(B\cdot\overline{B})\big]\cdot\big[(A+B)\cdot(\overline{A}+B)\big] \\
&\qquad\qquad\downarrow\ P5 \\
&\big[A+(B\cdot\overline{B})\big]\cdot\big[(A\cdot\overline{A})+B\big] \\
&\qquad\qquad\downarrow\ P6 \\
&[A+0]\cdot\big[(A\cdot\overline{A})+B\big] \\
&\qquad\qquad\downarrow\ P6 \\
&[A+0]\cdot[0+B] \\
&\qquad\qquad\downarrow\ P4 \\
&[A+0]\cdot[B+0] \\
&\qquad\qquad\downarrow\ P3 \\
&A\cdot[B+0] \\
&\qquad\qquad\downarrow\ P3 \\
&(A\cdot B)
\end{aligned}
\tag{5.3}
$$

5.11 Exemplo de manipulação algébrica por isomorfismo: Diagrama de Venn

- Na tentativa de sistematizar o processo de simplificação de uma expressão lógica, pode-se aproveitar o isomorfismo existente entre o Cálculo Proposicional e a Teoria de Conjuntos.

- Mapeando-se as operações e os elementos dos dois sistemas, o Diagrama de Venn pode ser usado na simplificação de expressões lógicas que envolvam 2 ou 3 variáveis.

- Nesse sentido, a cada variável da expressão lógica é associado um conjunto e para cada linha da Tabela Verdade da expressão lógica é associada uma região do Diagrama de Venn construído com os conjuntos definidos.

- A Figura 5.1 mostra um exemplo de mapeamento entre uma função genérica de duas variáveis e um Diagrama de Venn.

- Através da manipulação do Diagrama de Venn, pode-se tentar encontrar uma expressão simplificada, equivalente à expressão original.

- Ainda assim, embora seja um processo sistemático, a etapa final da simplificação através do Diagrama de Venn exige habilidade para encontrar a expressão mais simples.

- Além disso, para expressões que envolvam mais variáveis, o processo torna-se complexo e confuso.

- Portanto, devem ser utilizadas ferramentas mais eficientes, as quais serão tratadas a seguir.

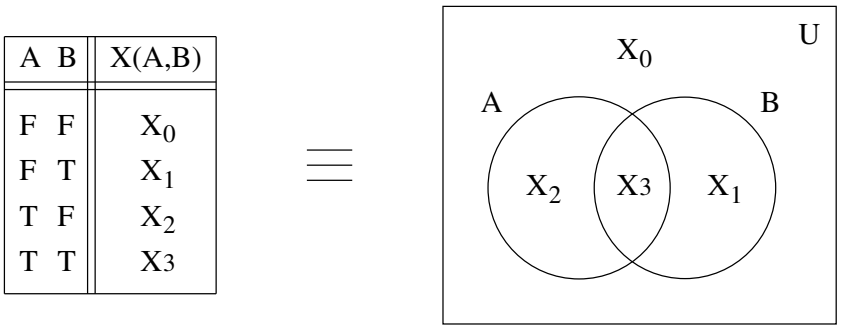


Figura 5.1: Mapeamento entre uma função genérica de duas variáveis e um Diagrama de Venn.

5.12 Resumo das relações algébricas

As Tabelas 5.3 a 5.7 apresentam um resumo das relações algébricas abordadas neste capítulo: os postulados, os lemas, os teoremas, a definição de uma estrutura algébrica de Boole particular e os isomorfismos.

Postulados de Huntington	
P3	$a+0 = a$ $a\cdot 1 = a$
P4	$(a+b) = (b+a)$ $(a\cdot b) = (b\cdot a)$
P5	$a+(b\cdot c) = (a+b)\cdot(a+c)$ $a\cdot(b+c) = (a\cdot b)+(a\cdot c)$
P6	$a+\overline{a} = 1$ $a\cdot\overline{a} = 0$

Tabela 5.3: Resumo dos Postulados de Huntington para a estrutura algébrica de Boole.

Lemas	
L2	$a+a = a$ $a\cdot a = a$
L3	$a+1 = 1$ $a\cdot 0 = 0$
L4	$0 = \overline{1}$ $\overline{0} = 1$
L5	$a+(a\cdot b) = a$ $a\cdot(a+b) = a$
L7	$a = \overline{(\overline{a})}$
L8	$a\cdot[(a+b)+c] = [(a+b)+c]\cdot a = a$

Tabela 5.4: Resumo dos lemas para a estrutura algébrica de Boole.

Teoremas	
T1	$\begin{cases} a+(b+c) = (a+b)+c \\ a\cdot(b\cdot c) = (a\cdot b)\cdot c \end{cases}$
T2	$\begin{cases} a+(\overline{a}\cdot b) = (a+b) \\ a\cdot(\overline{a}+b) = (a\cdot b) \end{cases}$
T3	$\begin{cases} \overline{(a+b)} = \overline{a}\cdot\overline{b} \\ \overline{(a\cdot b)} = \overline{a}+\overline{b} \end{cases}$
T4	$\begin{cases} (a\cdot b)+(\overline{a}\cdot c)+(b\cdot c) = (a\cdot b)+(\overline{a}\cdot c) \\ (a+b)\cdot(\overline{a}+c)\cdot(b+c) = (a+b)\cdot(\overline{a}+c) \end{cases}$

Tabela 5.5: Resumo dos teoremas para a estrutura algébrica de Boole.

Estrutura algébrica de Boole Particular	
Elementos	$\{K=\{0,1\}$
Complementos	$\begin{cases} 0 = \overline{1} \\ \overline{0} = 1 \end{cases}$
Identidades	$\begin{cases} 1\cdot 1 = 1+1 = 1+0 = 0+1 = 1 \\ 0+0 = 0\cdot 0 = 0\cdot 1 = 1\cdot 0 = 0 \end{cases}$

Tabela 5.6: Resumo da definição de uma estrutura algébrica de Boole particular.

Isomorfismos		
Teoria de Conjuntos	Cálculo Proposicional	Álgebra de Boole
$\cup$	$\vee$	$+$
$\cap$	$\wedge$	$\cdot$
S_Z	F	0
S_U	T	1
$C(S)$	$\neg(S)$	$\overline{S}$

Tabela 5.7: Resumo das relações de isomorfismo.

5.13 Exercícios propostos

1. Considerando o isomorfismo do Cálculo Proposicional com a Álgebra de Boole Binária:

   (a) Provar que os postulados, os lemas e os teoremas, da Álgebra de Boole, se aplicam para o Cálculo Proposicional.

   (b) Para cada um dos operadores lógicos (unário e binários), escrever sua Tabela Verdade usando a notação da Álgebra de Boole.

2. Para uma função genérica de duas variáveis, montar sua Tabela Verdade e identificar, em um Diagrama de Venn correspondente, cada uma das linhas da tabela.

3. Para uma função genérica de três variáveis, montar sua Tabela Verdade e identificar, em um Diagrama de Venn correspondente, cada uma das linhas da tabela.

4. Desenhar um Diagrama de Venn equivalente para cada uma das funções lógicas $X_i = f_i(A, B)$, onde $0 \leq i \leq 15$,

5. Para os exercícios listados abaixo, considerar as equações lógicas apresentadas em seguida.

   (a) Escrever as equações booleanas referentes às equações lógicas fornecidas.

   (b) Montar a Tabela Verdade para cada uma das equações lógicas fornecidas, usando a notação da Álgebra de Boole.

   (c) Aplicando os postulados, os lemas e os teoremas, da Álgebra Abstrata de Boole, apresentar uma simplificação para as equações booleanas referentes às equações lógicas fornecidas.

   (d) Utilizando o Diagrama de Venn, apresentar uma simplificação para as equações booleanas referentes às equações lógicas fornecidas.

   Equações lógicas:

   i. $F(A, B) = (A \vee B) \wedge (A \vee \neg B) \wedge (\neg A \vee B)$.
   ii. $F(A, B) = (\neg A \vee B) \wedge (A \vee \neg B) \wedge (\neg A \vee \neg B)$.
   iii. $F(A, B) = (A \wedge \neg B) \vee (\neg A \wedge B) \vee (A \wedge B)$.
   iv. $F(A, B) = (\neg A \wedge \neg B) \vee (\neg A \wedge B) \vee (A \wedge \neg B)$.
   v. $F(A, B, C, D) = (A \vee B) \wedge (\neg(C \wedge D))$.

6. Provar as relações dos conjuntos duais abaixo, utilizando a Tabela Verdade com a notação da Álgebra de Boole. Repetir o exercício, utilizando o Diagrama de Venn.

   (a) $A + 0 = A$ (P3)
   $A + \overline{A} = 1$ (P6)
   $A + A = A$ (L2)
   $A + 1 = 1$ (L3)

   (b) $A \cdot 1 = A$ (P3)
   $A \cdot \overline{A} = 0$ (P6)
   $A \cdot A = A$ (L2)
   $A \cdot 0 = 0$ (L3)

7. Provar as relações abaixo, relativas ao conectivo XOR ($\oplus$), utilizando a Tabela Verdade com a notação da Álgebra de Boole. Repetir os itens (a) a (n), utilizando o Diagrama de Venn.

   (a) $A \oplus 0 = A$

   (b) $A \oplus 1 = \overline{A}$

   (c) $A \oplus A = 0$

   (d) $A \oplus \overline{A} = 1$

   (e) $\overline{A} \oplus \overline{A} = 0$

   (f) $(A \oplus A) \oplus A = A$

   (g) $(A \oplus A) \oplus \overline{A} = \overline{A}$

   (h) $A \oplus (\overline{A} \oplus \overline{A}) = A$

   (i) $\overline{A} \oplus (\overline{A} \oplus \overline{A}) = \overline{A}$

   (j) $A \oplus B = (\overline{A} \cdot B) + (A \cdot \overline{B}) = (\overline{A} + \overline{B}) \cdot (A + B)$

   (k) $\overline{A \oplus B} = (\overline{A} \cdot \overline{B}) + (A \cdot B) = (\overline{A} + B) \cdot (A + \overline{B})$

   (l) $\overline{A \oplus B} = \overline{A} \oplus B = A \oplus \overline{B}$

   (m) $(A \oplus B) \oplus C = A \oplus (B \oplus C)$

   (n) $A \cdot (B \oplus C) = (A \cdot B) \oplus (A \cdot C)$

   (o) $f(C, D, E, F) = (C \cdot D) + (E \cdot F) = A \oplus B$, para $C = \overline{A}$, $D = B$, $E = A$ e $F = \overline{B}$.

   (p) $f(C, D, E, F) = (C \cdot D) + (E \cdot F) = \overline{A \oplus B}$, para $C = \overline{A}$, $D = \overline{B}$, $E = A$ e $F = B$.

   (q) $f(C, D, E, F) = (C + D) \cdot (E + F) = A \oplus B$, para $C = \overline{A}$, $D = \overline{B}$, $E = A$ e $F = B$.

   (r) $f(C, D, E, F) = (C + D) \cdot (E + F) = \overline{A \oplus B}$, para $C = \overline{A}$, $D = B$, $E = A$ e $F = \overline{B}$.

Capítulo 6

Formas padrões para representação de expressões booleanas

6.1 Introdução

- O projeto de sistemas digitais convencionais envolve a implementação de equações lógicas.
- Equações lógicas expressas pela Álgebra de Boole são denominadas equações booleanas.
- A minimização do custo de implementação de um projeto está associada à simplificação de suas equações booleanas.
- Um processo eficiente de simplificação deve ser simples de se entender, fácil de se operar, de rápida execução e completamente sistemático, a fim de permitir sua automatização.
- Os processos sistemáticos de simplificação que serão apresentados nos capítulos que se seguem trabalham sobre uma função expressa em formas padrões.
- Assim, para que se possa usar tais ferramentas de projeto, expressões booleanas genéricas devem ser inicialmente expandidas para tais formas.
- As formas padrões básicas são as decomposições da expressão booleana original em mintermos e maxtermos.
- Uma expansão em mintermos envolve realizar os mintermos necessários com operadores AND e, em seguida, combiná-los com operadores OR. Por essa razão, uma decomposição em mintermos é chamada de forma padrão AND-OR.
- Uma expansão em maxtermos envolve realizar os maxtermos necessários com operadores OR e, em seguida, combiná-los com operadores AND. Por essa razão, uma decomposição em maxtermos é chamada de forma padrão OR-AND.
- As demais formas padrões surgem como resultado de manipulações algébricas das formas padrões básicas.
- Podem ser identificados dois conjuntos de formas padrões:
  - Grupo AND-OR → {AND-OR, NAND-NAND, OR-NAND, NOR-OR}.
  - Grupo OR-AND → {AND-NOR, NAND-AND, OR-AND, NOR-NOR}.

6.2 Definições

- Literal: variável booleana ou seu complemento.
- Termo (combinação de literais):
  - Termo produto: combinação de literais através do operador AND.
  - Termo soma: combinação de literais através do operador OR.
  - Termo normal: termo produto ou termo soma onde nenhum literal aparece mais de uma vez.
  - Termo normal expandido: termo normal que contém todos os literais envolvidos na expressão booleana.
- Expansão (combinação de termos):
  - Soma de produtos (SOP): combinação de termos produto através do operador OR.
  - Produto de somas (POS): combinação de termos soma através do operador AND.
  - Soma de produtos normal: SOP onde os termos produto são termos normais.
  - Produto de somas normal: POS onde os termos soma são termos normais.
  - Forma normal expandida: forma normal (SOP ou POS) onde cada termo é um termo normal expandido.
- Expansão padrão:
  - Em uma forma SOP normal expandida, os termos produto são chamados de: produtos padrões, produtos canônicos ou mintermos.
  - A forma SOP normal expandida recebe as seguintes denominações: SOP padrão, SOP canônica, soma de mintermos, decomposição em mintermos ou forma normal disjuntiva completa.
  - Em uma forma POS normal expandida, os termos soma são chamados de: somas padrões, somas canônicas ou maxtermos.
  - A forma POS normal expandida recebe as seguintes denominações: POS padrão, POS canônica, produto de maxtermos, decomposição em maxtermos ou forma normal conjuntiva completa.
- Observações:
  - Uma vez que $(A \cdot A) = (A + A) = A$, $(A + \overline{A}) = 1$ e $(A \cdot \overline{A}) = 0$, conclui-se que múltiplas ocorrências de um literal em um termo soma ou em um termo produto acarretam: i) redundância ou ii) funções triviais.
    Portanto, pode-se dizer que, para fins de simplificação, a forma normal é a melhor forma de representação.
  - Além disso, observa-se que a forma normal expandida representa a maior forma de uma expressão sistematicamente construída, sem redundâncias.
    Por essa razão, ela será o ponto de partida para o processo de simplificação que será adotado, como será abordado a seguir.

- Exemplos da definição de mintermos e maxtermos, para três variáveis, são apresentados na Tabela 6.1 e na Tabela 6.2, respectivamente.
- Deve ser ressaltado que a chamada "Notação alternativa", ilustrada na Tabela 6.2, não será empregada neste texto.

Linha	A	B	C	*Produto*	*Mintermo*
0	0	0	0	$\left(\overline{A} \cdot \overline{B} \cdot \overline{C}\right)$	m_0
1	0	0	1	$\left(\overline{A} \cdot \overline{B} \cdot C\right)$	m_1
2	0	1	0	$\left(\overline{A} \cdot B \cdot \overline{C}\right)$	m_2
3	0	1	1	$\left(\overline{A} \cdot B \cdot C\right)$	m_3
4	1	0	0	$\left(A \cdot \overline{B} \cdot \overline{C}\right)$	m_4
5	1	0	1	$\left(A \cdot \overline{B} \cdot C\right)$	m_5
6	1	1	0	$\left(A \cdot B \cdot \overline{C}\right)$	m_6
7	1	1	1	$(A \cdot B \cdot C)$	m_7

Tabela 6.1: Definição de mintermos para três variáveis (A,B,C).

Linha	A	B	C	*Soma*	*Maxtermo*	Notação alternativa
0	0	0	0	$(A + B + C)$	M_0	M_7
1	0	0	1	$\left(A + B + \overline{C}\right)$	M_1	M_6
2	0	1	0	$\left(A + \overline{B} + C\right)$	M_2	M_5
3	0	1	1	$\left(A + \overline{B} + \overline{C}\right)$	M_3	M_4
4	1	0	0	$\left(\overline{A} + B + C\right)$	M_4	M_3
5	1	0	1	$\left(\overline{A} + B + \overline{C}\right)$	M_5	M_2
6	1	1	0	$\left(\overline{A} + \overline{B} + C\right)$	M_6	M_1
7	1	1	1	$\left(\overline{A} + \overline{B} + \overline{C}\right)$	M_7	M_0

Tabela 6.2: Definição de maxtermos para três variáveis (A,B,C).

6.3 Obtenção de formas SOP e POS padrões

Dada uma expressão booleana qualquer, pode-se obter uma forma padrão (SOP ou POS) por meio dos seguintes procedimentos ou de suas combinações: complementação da lista de termos canônicos, manipulação algébrica e utilização de tabela verdade. Cada um deles é abordado a seguir.

6.3.1 Complementação da lista de termos canônicos

Dada uma função lógica, cada termo canônico (mintermo ou maxtermo) de uma forma padrão que a represente é associado a uma linha da sua tabela verdade.

Uma vez que foi adotada uma numeração única para mintermos e maxtermos, associada com o número da linha da tabela verdade, pode-se facilmente obter a lista do termos canônicos de um dos dois tipos por complementação numérica da lista do outro tipo.

Por exemplo, dada a expressão $F(A,B,C) = \sum(1,3,5)$ para definir a SOP padrão, pode-se obter a expressão $F(A,B,C) = \prod(0,2,4,6,7)$ para definir a POS padrão, e vice-versa, por simples complementação numérica.

6.3.2 Manipulação algébrica

Para que se obtenha uma forma normal, a partir de uma equação genérica, deve-se atender aos seguintes itens:

- Inicialmente, se houver negação de algum termo que não seja um literal, deve-se aplicar o teorema de De Morgan.
- Quando houver negação apenas de literais, deve-se aplicar, repetidamente, as regras de distributividade.
- Finalmente, deve-se eliminar literais e/ou termos redundantes ou triviais.

Para que se obtenha a forma normal expandida, a partir de uma forma normal, deve-se atender aos seguintes itens:

- Primeiro, deve-se inserir os literais faltosos nos termos normais. Isso é feito aplicando-se os postulados, os lemas e os teoremas da álgebra de Boole sobre a forma normal.
- Em seguida, deve-se eliminar literais e/ou termos redundantes ou triviais.

Um exemplo do procedimento para a obtenção de uma forma POS padrão é apresentado na Equação (6.1), para $F = f(A,B,C,D)$.

$$
\begin{aligned}
F &= (A+B)\cdot\overline{(C\cdot D)} \\
&= (A+B)\cdot(\overline{C}+\overline{D}) \\
&= (A+B+0)\cdot(0+\overline{C}+\overline{D}) \\
&= [A+B+(C\cdot\overline{C})]\cdot[(B\cdot\overline{B})+\overline{C}+\overline{D}] \\
&= (A+B+C)\cdot(A+B+\overline{C})\cdot(B+\overline{C}+\overline{D})\cdot(\overline{B}+\overline{C}+\overline{D}) \\
&= (A+B+C+0)\cdot(A+B+\overline{C}+0)\cdot(0+B+\overline{C}+\overline{D})\cdot(0+\overline{B}+\overline{C}+\overline{D}) \\
&= [A+B+C+(D\cdot\overline{D})]\cdot[A+B+\overline{C}+(D\cdot\overline{D})]\cdot \\
&\quad [(A\cdot\overline{A})+B+\overline{C}+\overline{D}]\cdot[(A\cdot\overline{A})+\overline{B}+\overline{C}+\overline{D}] \\
&= (A+B+C+D)\cdot(A+B+C+\overline{D})\cdot(A+B+\overline{C}+D)\cdot \\
&\quad (A+B+\overline{C}+\overline{D})\cdot(A+B+\overline{C}+\overline{D})\cdot(\overline{A}+B+\overline{C}+\overline{D})\cdot \\
&\quad (A+\overline{B}+\overline{C}+\overline{D})\cdot(\overline{A}+\overline{B}+\overline{C}+\overline{D}) \\
&= (A+B+C+D)\cdot(A+B+C+\overline{D})\cdot(A+B+\overline{C}+D)\cdot \\
&\quad (A+B+\overline{C}+\overline{D})\cdot(\overline{A}+B+\overline{C}+\overline{D})\cdot \\
&\quad (A+\overline{B}+\overline{C}+\overline{D})\cdot(\overline{A}+\overline{B}+\overline{C}+\overline{D}) \\
&= \prod M(0,1,2,3,11,7,15)
\end{aligned}
\tag{6.1}
$$

Um exemplo do procedimento para a obtenção de uma forma SOP padrão é apresentado na Equação (6.2), para $F = f(A, B, C, D)$.

$$
\begin{aligned}
F &= (A+B)\cdot\overline{(C\cdot D)} \\
&= (A+B)\cdot(\overline{C}+\overline{D}) \\
&= [(A+B)\cdot\overline{C}]+[(A+B)\cdot\overline{D}] \\
&= [(A\cdot\overline{C})+(B\cdot\overline{C})]+[(A\cdot\overline{D})+(B\cdot\overline{D})] \\
&= (A\cdot\overline{C})+(B\cdot\overline{C})+(A\cdot\overline{D})+(B\cdot\overline{D}) \\
&= (A\cdot 1\cdot\overline{C})+(1\cdot B\cdot\overline{C})+(A\cdot 1\cdot\overline{D})+(1\cdot B\cdot\overline{D}) \\
&= [A\cdot(B+\overline{B})\cdot\overline{C}]+[(A+\overline{A})\cdot B\cdot\overline{C}]+ \\
&\quad [A\cdot(B+\overline{B})\cdot\overline{D}]+[(A+\overline{A})\cdot B\cdot\overline{D}] \\
&= (A\cdot B\cdot\overline{C})+(A\cdot\overline{B}\cdot\overline{C})+(A\cdot B\cdot\overline{C})+(\overline{A}\cdot B\cdot\overline{C})+ \\
&\quad (A\cdot B\cdot\overline{D})+(A\cdot\overline{B}\cdot\overline{D})+(A\cdot B\cdot\overline{D})+(\overline{A}\cdot B\cdot\overline{D}) \\
&= (A\cdot B\cdot\overline{C})+(A\cdot\overline{B}\cdot\overline{C})+(\overline{A}\cdot B\cdot\overline{C})+ \\
&\quad (A\cdot B\cdot\overline{D})+(A\cdot\overline{B}\cdot\overline{D})+(\overline{A}\cdot B\cdot\overline{D}) \\
&= (A\cdot B\cdot\overline{C}\cdot 1)+(A\cdot\overline{B}\cdot\overline{C}\cdot 1)+(\overline{A}\cdot B\cdot\overline{C}\cdot 1)+ \\
&\quad (A\cdot B\cdot 1\cdot\overline{D})+(A\cdot\overline{B}\cdot 1\cdot\overline{D})+(\overline{A}\cdot B\cdot 1\cdot\overline{D}) \\
&= [A\cdot B\cdot\overline{C}\cdot(D+\overline{D})]+[A\cdot\overline{B}\cdot\overline{C}\cdot(D+\overline{D})]+[\overline{A}\cdot B\cdot\overline{C}\cdot(D+\overline{D})]+ \\
&\quad [A\cdot B\cdot(C+\overline{C})\cdot\overline{D}]+[A\cdot\overline{B}\cdot(C+\overline{C})\cdot\overline{D}]+[\overline{A}\cdot B\cdot(C+\overline{C})\cdot\overline{D}] \\
&= (A\cdot B\cdot\overline{C}\cdot D)+(A\cdot B\cdot\overline{C}\cdot\overline{D})+(A\cdot\overline{B}\cdot\overline{C}\cdot D)+ \\
&\quad (A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D})+(\overline{A}\cdot B\cdot\overline{C}\cdot D)+(\overline{A}\cdot B\cdot\overline{C}\cdot\overline{D})+ \\
&\quad (A\cdot B\cdot C\cdot\overline{D})+(A\cdot B\cdot\overline{C}\cdot\overline{D})+(A\cdot\overline{B}\cdot C\cdot\overline{D})+ \\
&\quad (A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D})+(\overline{A}\cdot B\cdot C\cdot\overline{D})+(\overline{A}\cdot B\cdot\overline{C}\cdot\overline{D}) \\
&= (A\cdot B\cdot\overline{C}\cdot D)+(A\cdot B\cdot\overline{C}\cdot\overline{D})+(A\cdot\overline{B}\cdot\overline{C}\cdot D)+(A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D})+ \\
&\quad (\overline{A}\cdot B\cdot\overline{C}\cdot D)+(\overline{A}\cdot B\cdot\overline{C}\cdot\overline{D})+(A\cdot B\cdot C\cdot\overline{D})+(A\cdot\overline{B}\cdot C\cdot\overline{D})+ \\
&\quad (\overline{A}\cdot B\cdot C\cdot\overline{D}) \\
&= \sum m(13,12,9,8,5,4,14,10,6)
\end{aligned}
\tag{6.2}
$$

6.3.3 Utilização de tabela verdade

A partir de uma tabela verdade, pode-se obter diretamente as formas padrões na forma de decomposição em mintermos ou maxtermos. Ambas as formas são discutidas a seguir.

Decomposição em mintermos

- Dada uma expressão booleana, pode-se montar uma tabela verdade que a represente, como demonstrado na Tabela 6.3, para uma função $X = f(A, B, C)$.

A	B	C	X	Y_1	Y_2	Y_3	*Linha*	*Produto*	*Mintermo*
0	0	0	0	0	0	0	0	$\left(\overline{A} \cdot \overline{B} \cdot \overline{C}\right)$	m_0
0	0	1	1	1	0	0	1	$\left(\overline{A} \cdot \overline{B} \cdot C\right)$	m_1
0	1	0	0	0	0	0	2	$\left(\overline{A} \cdot B \cdot \overline{C}\right)$	m_2
0	1	1	0	0	0	0	3	$\left(\overline{A} \cdot B \cdot C\right)$	m_3
1	0	0	1	0	1	0	4	$\left(A \cdot \overline{B} \cdot \overline{C}\right)$	m_4
1	0	1	0	0	0	0	5	$\left(A \cdot \overline{B} \cdot C\right)$	m_5
1	1	0	1	0	0	1	6	$\left(A \cdot B \cdot \overline{C}\right)$	m_6
1	1	1	0	0	0	0	7	$(A \cdot B \cdot C)$	m_7

Tabela 6.3: Exemplo de função X, proposta de decomposição de X em funções Y_k e associação das funções Y_k com os mintermos m_k.

- Da Tabela 6.3, pode-se escrever que

$$\begin{array}{rcccccc} Y_1 & = & f_1(A,B,C) & = & \left(\overline{A} \cdot \overline{B} \cdot C\right) & = & m_1 \\ Y_2 & = & f_2(A,B,C) & = & \left(A \cdot \overline{B} \cdot \overline{C}\right) & = & m_4 \\ Y_3 & = & f_3(A,B,C) & = & \left(A \cdot B \cdot \overline{C}\right) & = & m_6 \end{array}$$

  e que $X = f(A, B, C) = (Y_1) + (Y_2) + (Y_3)$.

- Pelas definições apresentadas, as funções auxiliares Y_i são mintermos e a função X pode ser descrita pela forma SOP padrão $X = m_1 + m_4 + m_6 = \sum m(1, 4, 6)$.

- Analisando-se as funções auxiliares Y_i, pode-se observar que, para cada combinação das variáveis, apenas um dos termos produto apresenta um valor lógico 1, enquanto todos os outros assumem o valor lógico 0. Essa é a razão pela qual tais termos são denominados produtos canônicos ou mintermos.

- Uma vez que toda expressão booleana é completamente representada por uma tabela verdade e, a partir da tabela verdade, sempre é possível se obter uma forma SOP padrão, pode-se enunciar o teorema a seguir.

- **Teorema**: Qualquer expressão booleana de N variáveis, $y = f(x_1, x_2, \cdots, x_N)$, pode ser expressa por uma forma SOP padrão.

Decomposição em maxtermos

- Dada uma expressão booleana, pode-se montar uma tabela verdade que a represente, como demonstrado na Tabela 6.4, para uma função $X = f(A, B, C)$.

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

X
0
1
0
0
1
0
1
0

Z_1	Z_2	Z_3	Z_4	Z_5
0	1	1	1	1
1	1	1	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1
1	1	1	0	1
1	1	1	1	1
1	1	1	1	0

Linha	*Soma*	*Maxtermo*
0	$(A+B+C)$	M_0 (*ou* M_7)
1	$\left(A+B+\overline{C}\right)$	M_1 (*ou* M_6)
2	$\left(A+\overline{B}+C\right)$	M_2 (*ou* M_5)
3	$\left(A+\overline{B}+\overline{C}\right)$	M_3 (*ou* M_4)
4	$\left(\overline{A}+B+C\right)$	M_4 (*ou* M_3)
5	$\left(\overline{A}+B+\overline{C}\right)$	M_5 (*ou* M_2)
6	$\left(\overline{A}+\overline{B}+C\right)$	M_6 (*ou* M_1)
7	$\left(\overline{A}+\overline{B}+\overline{C}\right)$	M_7 (*ou* M_0)

Tabela 6.4: Exemplo de função X, proposta de decomposição de X em funções Z_k e associação das funções Z_k com os maxtermos M_k.

- Da Tabela 6.4, pode-se escrever que

$$\begin{array}{rcrcrcl} Z_1 & = & f_1(A,B,C) & = & (A+B+C) & = & M_0 \\ Z_2 & = & f_2(A,B,C) & = & \left(A+\overline{B}+C\right) & = & M_2 \\ Z_3 & = & f_3(A,B,C) & = & \left(A+\overline{B}+\overline{C}\right) & = & M_3 \\ Z_4 & = & f_4(A,B,C) & = & \left(\overline{A}+B+\overline{C}\right) & = & M_5 \\ Z_5 & = & f_5(A,B,C) & = & \left(\overline{A}+\overline{B}+\overline{C}\right) & = & M_7 \end{array}$$

  e que $X = f(A, B, C) = (Z_1) \cdot (Z_2) \cdot (Z_3) \cdot (Z_4) \cdot (Z_5)$.

- Pelas definições apresentadas, as funções auxiliares Z_i são maxtermos e a função X pode ser descrita pela forma POS padrão $X = M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_7 = \prod M(0, 2, 3, 5, 7)$.

- Analisando-se as funções auxiliares Z_i, pode-se observar que, para cada combinação das variáveis, apenas um dos termos soma apresenta um valor lógico 0, enquanto todos os outros assumem o valor lógico 1. Essa é a razão pela qual tais termos são denominados somas canônicas ou maxtermos.

- Uma vez que toda expressão booleana é completamente representada por uma tabela verdade e, a partir da tabela verdade, sempre é possível se obter uma forma POS padrão, pode-se enunciar o teorema a seguir.

- **Teorema**: Qualquer expressão booleana de N variáveis, $y = f(x_1, x_2, \cdots, x_N)$, pode ser expressa por uma forma POS padrão.

6.3.4 Negação das formas SOP e POS

Dadas uma função genérica $F(A, B, C, \ldots) = F(\cdot)$ e a sua negação $\overline{F}(A, B, C, \ldots) = \overline{F}(\cdot)$, podem-se estabelecer relações simples entre as suas formas padrões SOP e POS.

Primeiramente, deve-se lembrar a relação entre mintermos e maxtermos, dada por

$$m_k = \overline{M_k} \ .$$

Em seguida, deve-se levar em consideração o Teorema de De Morgan para os casos genéricos dados por

$$\overline{(X \cdot Y \cdot Z \cdot \ldots)} = \left(\overline{X} + \overline{Y} + \overline{Z} + \ldots\right)$$

e

$$\overline{(X + Y + Z + \ldots)} = \left(\overline{X} \cdot \overline{Y} \cdot \overline{Z} \cdot \ldots\right) \ .$$

Por fim, definindo-se, sem perda de generalidade, as formas padrões SOP e POS de $F(\cdot)$ respectivamente como

$$F(\cdot)_{SOP} = (m_i + m_j + m_k)$$

e

$$F(\cdot)_{POS} = (M_l \cdot M_m \cdot M_n \cdot \ldots) \ ,$$

bem como as formas padrões SOP e POS de $\overline{F}(\cdot)$ respectivamente como

$$\overline{F}(\cdot)_{SOP} = (m_l + m_m + m_n + \ldots)$$

e

$$\overline{F}(\cdot)_{POS} = (M_i \cdot M_j \cdot M_k) \ ,$$

obtêm-se as seguintes relações:

$$\begin{aligned} \overline{F}(\cdot)_{POS} &= (M_i \cdot M_j \cdot M_k) \\ &= (\overline{m}_i \cdot \overline{m}_j \cdot \overline{m}_k) \\ &= \overline{(m_i + m_j + m_k)} \\ &= \overline{F(\cdot)_{SOP}} \end{aligned} \tag{6.3}$$

e

$$\begin{aligned} \overline{F}(\cdot)_{SOP} &= (m_l + m_m + m_n + \ldots) \\ &= \left(\overline{M}_l + \overline{M}_m + \overline{M}_n + \ldots\right) \\ &= \overline{(M_l \cdot M_m \cdot M_n \cdot \ldots)} \\ &= \overline{F(\cdot)_{POS}} \ . \end{aligned} \tag{6.4}$$

Como exemplo, pode-se usar a função $X = F(A, B, C)$, definida nas Tabelas 6.3 e 6.4. Nesse caso, as formas padrões SOP e POS de $F(\cdot)$ dadas por

$$F(\cdot)_{SOP} = (m_1 + m_4 + m_6)$$

e

$$F(\cdot)_{POS} = (M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_7) \ ,$$

bem como as formas padrões SOP e POS de $\overline{F}(\cdot)$ dadas por

$$\overline{F}(\cdot)_{SOP} = (m_0 + m_2 + m_3 + m_5 + m_7)$$

e

$$\overline{F}(\cdot)_{POS} = (M_1 \cdot M_4 \cdot M_6) \ ,$$

e, finalmente, podem-se obter as seguintes relações:

$$\begin{aligned} \overline{F}(\cdot)_{POS} &= (M_1 \cdot M_4 \cdot M_6) \\ &= (\overline{m}_1 \cdot \overline{m}_4 \cdot \overline{m}_6) \\ &= \overline{(m_1 + m_4 + m_6)} \\ &= \overline{F(\cdot)_{SOP}} \end{aligned}$$

e

$$\begin{aligned} \overline{F}(\cdot)_{SOP} &= (m_0 + m_2 + m_3 + m_5 + m_7) \\ &= \left(\overline{M}_0 + \overline{M}_2 + \overline{M}_3 + \overline{M}_5 + \overline{M}_7\right) \\ &= \overline{(M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_7)} \\ &= \overline{F(\cdot)_{POS}} \ . \end{aligned}$$

6.4 Conjuntos de formas padrões

6.4.1 Definições

- Uma expressão booleana pode ser representada por um total de oito formas padrões.
- Uma soma de mintermos (SOP padrão) é identificada como uma forma AND-OR.
- Um produto de maxtermos (POS padrão) é identificado como uma forma OR-AND.
- A partir da forma AND-OR, pode-se obter o seguinte grupo de formas padrões: Grupo AND-OR = {AND-OR, NAND-NAND, OR-NAND, NOR-OR}.
- A partir da forma OR-AND, pode-se obter o seguinte grupo de formas padrões: Grupo OR-AND = {OR-AND, NOR-NOR, AND-NOR, NAND-AND}.

6.4.2 Obtenção

Dentro de um mesmo grupo, as formas podem ser obtidas através da aplicação sucessiva dos Teoremas de De Morgan.

A mudança de grupo pode ser realizada aplicando-se a regra de distributividade entre as formas AND-OR e OR-AND.

As Tabelas 6.5 a 6.7, exemplificam, para a função XOR, a obtenção do grupo AND-OR, a mudança de grupo e a obtenção do grupo OR-AND, respectivamente.

Deve ser lembrado ainda que, partindo-se da tabela verdade, podem-se obter diretamente algumas formas padrões, empregando-se os seguintes conceitos:

- Por definição, a soma de mintermos da função $F(\cdot)$ fornece a forma AND-OR.
- Por definição, o produto de maxtermos da função $F(\cdot)$ fornece a forma OR-AND.
- Pela Equação (6.3), os mintermos da função $F(\cdot)$ podem ser usados para que se obtenha a forma AND-NOR (grupo OR-AND).
- Pela Equação (6.4), os maxtermos da função $F(\cdot)$ podem ser usados para que se obtenha a forma OR-NAND (grupo AND-OR).

Assim, uma outra técnica para mudança de grupo é montar a tabela verdade da função, a partir de uma dada forma. De posse da tabela verdade, pode-se obter uma forma do outro grupo.

6.4.3 Utilização

O projeto e a análise de circuitos digitais convencionais são baseados na formação, na manipulação e na implementação de funções lógicas/booleanas. Por dois motivos básicos, as formas AND-OR e OR-AND são as formas mais utilizadas na representação de tais funções. Primeiro, ela são diretamente obtidas no processo de especificação do problema. Segundo, elas são mais próximas da forma como se processa o pensamento (expressão lógica) do ser humano.

Por outro lado, as formas NAND-NAND e NOR-NOR são as formas básicas de operação dos circuitos eletro-eletrônicos usados para implementar as funções lógicas/booleanas.

Portanto, transformações entre tais formas são freqüentemente realizadas.

As formas padrões possuem dois grandes atrativos. Por um lado, e de uma forma geral, elas apresentam o menor retardo de operação, uma vez que são compostas apenas por dois planos de lógica. Além disso, elas são o ponto de partida para um processo de simplificação sistemático e eficiente, conforme será abordado em seguida.

Expressão booleana					Forma padrão
$F(A,B)$	$=$	$A \oplus B$			
	$=$	$\Big(\overline{A} \cdot B\Big) + \Big(A \cdot \overline{B}\Big)$			AND-OR
	$=$	$\overline{\overline{\Big(\overline{A} \cdot B\Big) + \Big(A \cdot \overline{B}\Big)}}$			
	$=$	$\overline{\overline{\Big(\overline{A} \cdot B\Big)} \cdot \overline{\Big(A \cdot \overline{B}\Big)}}$	$=$	$(\overline{A} \uparrow B) \uparrow (A \uparrow \overline{B})$	NAND-NAND
	$=$	$\overline{\Big(A + \overline{B}\Big) \cdot \Big(\overline{A} + B\Big)}$	$=$	$\Big(A + \overline{B}\Big) \uparrow \Big(\overline{A} + B\Big)$	OR-NAND
	$=$	$\overline{\Big(A + \overline{B}\Big) + \Big(\overline{A} + B\Big)}$	$=$	$\Big(A \downarrow \overline{B}\Big) + \Big(\overline{A} \downarrow B\Big)$	NOR-OR
	$=$	$\Big(\overline{A} \cdot B\Big) + \Big(A \cdot \overline{B}\Big)$			AND-OR

Tabela 6.5: Exemplo da obtenção do grupo AND-OR para a função XOR.

$F(A,B)$	$=$	$A \oplus B$
	$=$	$\Big(\overline{A} \cdot B\Big) + \Big(A \cdot \overline{B}\Big)$
	$=$	$\Big[\Big(\overline{A} \cdot B\Big) + A\Big] \cdot \Big[\Big(\overline{A} \cdot B\Big) + \overline{B}\Big]$
	$=$	$\Big[(\overline{A} + A) \cdot (B + A)\Big] \cdot \Big[\Big(\overline{A} + \overline{B}\Big) \cdot \Big(B + \overline{B}\Big)\Big]$
	$=$	$[(1) \cdot (B + A)] \cdot \Big[\Big(\overline{A} + \overline{B}\Big) \cdot (1)\Big]$
	$=$	$(B + A) \cdot \Big(\overline{A} + \overline{B}\Big)$
	$=$	$\Big(\overline{A} + \overline{B}\Big) \cdot (A + B)$

Tabela 6.6: Exemplo da mudança de grupo para a função XOR.

Expressão booleana					Forma padrão
$F(A,B)$	$=$	$A \oplus B$			
	$=$	$\Big(\overline{A} + \overline{B}\Big) \cdot (A + B)$			OR-AND
	$=$	$\overline{\overline{\Big(\overline{A} + \overline{B}\Big) \cdot (A + B)}}$			
	$=$	$\overline{\Big(\overline{A} + \overline{B}\Big)} + \overline{(A + B)}$	$=$	$(\overline{A} \downarrow \overline{B}) \downarrow (A \downarrow B)$	NOR-NOR
	$=$	$\overline{(A \cdot B) + \Big(\overline{A} \cdot \overline{B}\Big)}$	$=$	$(A \cdot B) \downarrow \Big(\overline{A} \cdot \overline{B}\Big)$	AND-NOR
	$=$	$\overline{(A \cdot B)} \cdot \overline{\Big(\overline{A} \cdot \overline{B}\Big)}$	$=$	$(A \uparrow B) \cdot \Big(\overline{A} \uparrow \overline{B}\Big)$	NAND-AND
	$=$	$\Big(\overline{A} + \overline{B}\Big) \cdot (A + B)$			OR-AND

Tabela 6.7: Exemplo da obtenção do grupo OR-AND para a função XOR.

6.5 Exercícios propostos

1. Um aluno de Circuitos Digitais desenvolveu o seguinte raciocínio:

   - A Tabela Verdade de uma função lógica $F(\cdot)$, com notação booleana, possui apenas os valores algébricos 0 e 1.
   - Equacionando-se cada valor 1 de $F(\cdot)$ isoladamente, obtém-se uma SOP padrão $F(\cdot)_{SOP}$.
   - Equacionando-se cada valor 0 de $F(\cdot)$ isoladamente, obtém-se uma POS padrão $F(\cdot)_{POS}$.
   - Dado que $0 = \overline{1}$ e $1 = \overline{0}$:
     - Pode-se obter a POS de $F(\cdot)$ por meio da SOP da negação de $F(\cdot)$: $F(\cdot)_{POS} = \overline{F}(\cdot)_{SOP}$.
     - Pode-se obter a SOP de $F(\cdot)$ por meio da POS da negação de $F(\cdot)$: $F(\cdot)_{SOP} = \overline{F}(\cdot)_{POS}$.

   Você concorda com ele? **JUSTIFIQUE !!!**

   Observação:

   Deve-se entender como justificativa, uma explicação conceitual, acompanhada ou não de uma prova matemática.

   Uma simples demonstração matemática, provando a veracidade ou a falsidade do raciocínio, sem uma argumentação conceitual, não será aceita como justificativa.

   De forma simples, devem ser apresentadas as razões que tornam o raciocínio verdadeiro ou falso.

2. Para os exercícios listados abaixo, considerar as equações booleanas apresentadas em seguida.

   (a) Algebricamente, obter a forma SOP normal da equação fornecida.
   (b) Algebricamente, obter a forma SOP padrão da equação fornecida.
   (c) Expressar a função por uma lista de mintermos.
   (d) A partir da SOP padrão, obter as demais formas do seu grupo.
   (e) Algebricamente, obter a forma POS normal da equação fornecida.
   (f) Algebricamente, obter a forma POS padrão da equação fornecida.
   (g) Expressar a função por uma lista de maxtermos.
   (h) A partir da POS padrão, obter as demais formas do seu grupo.
   (i) Expressar a função por uma tabela verdade, com notação booleana.

   Equações boolenas:

   i. $F(A,B,C) = \Big\{B \cdot \Big[\Big(\overline{A} \cdot \overline{C}\Big) + (A \cdot C)\Big]\Big\} + \overline{\Big\{B + \Big[\Big(\overline{A} + \overline{C}\Big) \cdot (A + C)\Big]\Big\}}$.

   ii. $F(A,B,C) = \overline{\Big\{\Big[\overline{(A+B)} + C\Big] \cdot \Big[\overline{A} + \overline{\Big(B + \overline{C}\Big)}\Big]\Big\}}$.

   iii. $F(A,B,C) = \Big[\overline{\Big(A \cdot \overline{B}\Big)} + C\Big] \cdot \overline{[(A+C) \cdot B]}$.

3. Dados os conectivos lógicos XOR ($\oplus$) e XNOR ($\odot$), bem como as seguintes relações:

   (a) $R_1 = (A \ \oplus \ B)$,
   (b) $R_2 = (A \ \oplus \ \neg B)$,
   (c) $R_3 = (\neg A \ \oplus \ B)$,
   (d) $R_4 = (\neg A \ \oplus \ \neg B)$,
   (e) $R_5 = (A \ \odot \ B)$,
   (f) $R_6 = (A \ \odot \ \neg B)$,
   (g) $R_7 = (\neg A \ \odot \ B)$,
   (h) $R_8 = (\neg A \ \odot \ \neg B)$,

   atenda aos seguintes itens:

   - calcule a **tabela verdade** para cada uma delas, usando a notação booleana;
   - expresse cada uma delas por meio de uma combinação de **mintermos**;
   - expresse cada uma delas por meio de uma combinação de **maxtermos**;

   e identifique as equivalências existentes.

Capítulo 7

Simplificação algébrica sistemática de expressões booleanas

7.1 Introdução

Para que se possa realizar uma simplificação algébrica sistemática de expressões booleanas, empregando-se as ferramentas da Álgebra Booleana (Postulados, Lemas e Teoremas), é necessário que se definam três elementos: o ponto de partida, o método sistemático e o ponto de chegada. Como será justificado a seguir, o ponto de partida escolhido são as formas SOP padrão e POS padrão. O ponto de chegada e o método sistemático são definidos e descritos a seguir. Inicialmente, é definido o ponto final desejado. Em seguida, é analisado o processo de expansão de uma expressão booleana. A partir disso, torna-se praticamente natural tanto a escolha do ponto de partida quanto a definição do processo de simplificação sistemática de expressões booleanas.

7.2 Definição do ponto final: forma mínima procurada

Para que se possa propor um processo sistemático destinado à minimização de expressões booleanas, é necessário que se defina, primeiramente, o que será aceito como expressão mínima.

Ao se definir a expressão mínima, serão considerados os seguintes requisitos:

1. a menor quantidade de circuito necessário à implementação;

2. o circuito com o menor tempo de resposta;

3. a existência de procedimentos sistemáticos, simples e práticos, que levem à expressão mínima.

No intuito de atender aos três requisitos acima simultaneamente, serão consideradas apenas as expressões com dois planos de operações lógicas: SOP (AND-OR) e POS (OR-AND). Esses dois tipos de expressões são conhecidos também por: estruturas em dois planos, estruturas em dois níveis e estruturas de segunda ordem.

Uma expressão com dois planos de operações lógicas será considerada mínima se:

1. não existir outra expressão desse tipo com um número menor de termos;

2. não existir outra expressão desse tipo com mesmo número de termos, porém com um número menor de literais.

Deve ser ressaltado que, se for levado em consideração apenas a quantidade de circuito necessário à implementação, a definição acima não garante que a expressão final será um mínimo global. Em alguns casos, embora aceita como mínima, a expressão encontrada pode representar um mínimo local. Por exemplo, a função $F(A,B,C,D) = \sum m(5,6,9,10,13,14)$ pode ser expressa por

$$F(A,B,C,D) = (A+B)\cdot\Big[(C\cdot\overline{D})+(\overline{C}\cdot D)\Big] \tag{7.1}$$

e

$$F(A,B,C,D) = (A\cdot\overline{C}\cdot D)+(A\cdot C\cdot\overline{D})+(B\cdot\overline{C}\cdot D)+(B\cdot C\cdot\overline{D})\ . \tag{7.2}$$

Na Equação (7.1), a função é representada por uma expressão booleana genérica. Essa expressão apresenta três planos de operações lógicas, sendo considerada uma estrutura de três níveis ou de ordem superior (a dois). Na Equação (7.2), a expressão booleana encontra-se na forma SOP. Embora a expressão da Equação (7.1) seja a menor entre as duas, ela apresenta, funcionalmente, um tempo de resposta maior, uma vez que envolve três planos de operações lógicas. Além disso, não há um processo sistemático, simples e prático, para sua obtenção. Logo, a expressão da Equação (7.2) será considerada a expressão mínima procurada pelo processo de minimização descrito a seguir, ainda que seja um mínimo local.

Com base nessa definição de forma mínima (local), conclui-se que a uma determinada função lógica podem ser atribuídas diversas expressões mínimas (locais) equivalentes.

Além disso, não se pode garantir que, para uma dada função, a expressão mínima (local) seja da forma SOP ou da forma POS. É necessário minimizar ambas as formas e escolher a menor delas. Por exemplo, a função $F(A,B,C,D) = \sum m(0,1,8,9)$ apresenta formas mínimas SOP e POS equivalentes entre si. Por sua vez, no caso das funções $F(A,B,C,D) = \sum m(4,5,8,9)$ e $F(A,B,C,D) = \sum m(2,7,8,13)$, a forma POS mínima é menor que a forma SOP mínima. Finalmente, para a função $F(A,B,C,D) = \sum m(6,7,8,9)$, a forma SOP mínima é menor que a forma POS mínima. Deve-se observar que, em todos os exemplos, o número de mintermos é o mesmo e que ele é menor do que o número de maxtermos.

Por fim, vale a pena exemplificar uma situação comum, onde uma tentativa de simplificação adicional sobre uma forma mínima (SOP ou POS) conduz não a uma forma menor, mas acarreta uma mudança de forma. Por exemplo, a tentativa de simplificação da SOP mínima

$$f(A,B,C) = (A\cdot B)+(A\cdot C)$$

conduz à POS mínima

$$f(A,B,C) = (A)\cdot(B+C)\ .$$

Do mesmo modo, a tentativa de simplificação da POS mínima

$$f(A,B,C) = (A+B)\cdot(A+C)$$

conduz à SOP mínima

$$f(A,B,C) = (A)+(B\cdot C)\ .$$

7.3 Processo de expansão vs processo de simplificação

O processo sistemático de simplificação que será apresentado pode ser visto como a inversão de um processo de expansão da forma mínima definida acima. Portanto, é interessante analisar o processo de expansão, na tentativa de facilitar o entendimento do processo de simplificação. Isso é feito a seguir.

7.3.1 Expansão sem redundância

Aplicando-se os Postulados P3, P6 e P5, bem como o Postulado P4, da Álgebra de Boole, pode-se demonstrar as seguintes relações:

$$f(A,B) = (B) = (B)\cdot(1) = (B)\cdot(A+\overline{A}) = (B\cdot A)+(B\cdot\overline{A}) = (A\cdot B)+(\overline{A}\cdot B)\ , \quad (7.3)$$

$$f(A,B) = (B) = (B)+(0) = (B)+(A\cdot\overline{A}) = (B+A)\cdot(B+\overline{A}) = (A+B)\cdot(\overline{A}+B)\ , \quad (7.4)$$

$$f(A,B,C) = (A\cdot B) = (A\cdot B)\cdot(1) = (A\cdot B)\cdot(C+\overline{C}) = (A\cdot B\cdot C)+(A\cdot B\cdot\overline{C})\ , \quad (7.5)$$

$$f(A,B,C) = (A+B) = (A+B)+(0) = (A+B)+(C\cdot\overline{C}) = (A+B+C)\cdot(A+B+\overline{C})\ , \quad (7.6)$$

$$\begin{aligned} f(A,B,C) &= (B) \\ &= (B)\cdot(1) \\ &= (B)\cdot(A+\overline{A}) \\ &= (B\cdot A)+(B\cdot\overline{A}) \\ &= (A\cdot B)+(\overline{A}\cdot B) \\ &= [(A\cdot B)\cdot(1)]+[(\overline{A}\cdot B)\cdot(1)] \\ &= [(A\cdot B)\cdot(C+\overline{C})]+[(\overline{A}\cdot B)\cdot(C+\overline{C})] \\ &= (A\cdot B\cdot C)+(A\cdot B\cdot\overline{C})+(\overline{A}\cdot B\cdot C)+(\overline{A}\cdot B\cdot\overline{C}) \end{aligned} \quad (7.7)$$

e

$$\begin{aligned} f(A,B,C) &= (B) \\ &= (B)+(0) \\ &= (B)+(A\cdot\overline{A}) \\ &= (B+A)\cdot(B+\overline{A}) \\ &= (A+B)\cdot(\overline{A}+B) \\ &= [(A+B)+(0)]\cdot[(\overline{A}+B)+(0)] \\ &= [(A+B)+(C\cdot\overline{C})]\cdot[(\overline{A}\cdot B)+(C\cdot\overline{C})] \\ &= (A+B+C)\cdot(A+B+\overline{C})\cdot(\overline{A}+B+C)\cdot(\overline{A}+B+\overline{C})\ . \end{aligned} \quad (7.8)$$

7.3.2 Expansão com redundância

Aplicando-se os Postulados P3, P6 e P5, bem como o Postulado P4 e o Lema L2, da Álgebra de Boole, pode-se demonstrar as seguintes relações:

$$\begin{aligned} f(A,B,C) &= (A\cdot B)+(A\cdot C) \\ &= [(A\cdot B)\cdot(1)]+[(A\cdot C)\cdot(1)] \\ &= [(A\cdot B)\cdot(C+\overline{C})]+[(A\cdot C)\cdot(B+\overline{B})] \\ &= [(A\cdot B\cdot C)+(A\cdot B\cdot\overline{C})]+[(A\cdot C\cdot B)+(A\cdot C\cdot\overline{B})] \\ &= [(A\cdot B\cdot C)+(A\cdot B\cdot\overline{C})]+[(A\cdot B\cdot C)+(A\cdot\overline{B}\cdot C)] \\ &= (A\cdot B\cdot C)+(A\cdot B\cdot\overline{C})+(A\cdot\overline{B}\cdot C) \end{aligned} \quad (7.9)$$

e

$$\begin{aligned} f(A,B,C) &= (A+B)\cdot(A+C) \\ &= [(A+B)+(0)]\cdot[(A+C)+(0)] \\ &= [(A+B)+(C\cdot\overline{C})]\cdot[(A+C)+(B\cdot\overline{B})] \\ &= [(A+B+C)\cdot(A+B+\overline{C})]\cdot[(A+C+B)\cdot(A+C+\overline{B})] \\ &= [(A+B+C)\cdot(A+B+\overline{C})]\cdot[(A+B+C)\cdot(A+\overline{B}+C)] \\ &= (A+B+C)\cdot(A+B+\overline{C})\cdot(A+\overline{B}+C)\ . \end{aligned} \tag{7.10}$$

7.3.3 Análise das expansões

As Equações (7.3) a (7.10) mostram que:

- O mesmo termo inicial simplificado pode gerar termos finais expandidos que são do tipo mintermo (Termo Soma Padrão) ou maxtermo (Termo Produto Padrão).
- A expansão de 1 literal pode gerar 2 termos (somas ou produtos) padrões de 2 literais.
- A expansão de 1 literal pode gerar 4 termos (somas ou produtos) padrões de 3 literais.
- Pode-se mostrar que a expansão de 1 literal pode gerar 2^L termos (somas ou produtos) padrões de $L+1$ literais.
- Algumas expansões geram termos (somas ou produtos) padrões iguais, o que significa uma redundância na equação expandida final. Dito de outra forma, pode-se pensar que tais redundâncias devem necessariamente aparecer para que a expansão seja completa.
- A expansão sem redundâncias de um termo qualquer segue o procedimento geral dado pela aplicação das seguintes ferramentas da Álgebra de Boole: P3 $\rightarrow$ P6 $\rightarrow$ P5.
- A expansão com redundâncias de um termo qualquer segue o procedimento geral dado pela aplicação das seguintes ferramentas da Álgebra de Boole: P3 $\rightarrow$ P6 $\rightarrow$ P5 $\rightarrow$ L2.
- Eventualmente, durante a expansão, pode ser necessário aplicar o Postulado P4.

7.4 Definição do ponto inicial: forma máxima padrão

A partir da definição da forma mínima procurada e da análise do processo de expansão, torna-se natural pensar em um processo de simplificação fundamentado na inversão do processo de expansão.

No caso da expansão, o ponto de partida é uma forma mínima. Aplicando-se uma seqüência de postulados, obtém-se uma forma expandida máxima, sem redundância de termos. A forma máxima será uma SOP padrão ou uma POS padrão.

No caso da simplificação, pode-se propor que a equação lógica/booleana seja, inicialmente, colocada em uma forma máxima padrão, SOP ou POS. Em seguida, aplicando-se a seqüência de postulados na ordem inversa, chega-se à forma mínima, SOP ou POS. Finalmente, pode-se comparar as duas formas mínimas, SOP e POS, e decidir-se por uma delas.

7.5 Simplificação sistemática de expressões booleanas a partir de SOP e POS padrões

As formas SOP (AND-OR) e POS (OR-AND) apresentam duas grandes vantagens: a facilidade de descrição do problema durante a sua modelagem e a quantidade de planos de operações lógicas utilizados na sua implementação. Porém, nem sempre tais formas estão em sua expressão mais simples.

A aplicação das operações de aglutinação e de replicação em funções lógicas expressas nas formas padrões SOP e POS é a base para um processo sistemático de simplificação. Isso é discutido a seguir.

7.5.1 Operações básicas: aglutinação e replicação

No processo sistemático de simplificação algébrica de expressões booleanas, duas operações são fundamentais: a aglutinação e a replicação, as quais são definidas nas Equações (7.11) e (7.12), respectivamente. Da Álgebra de Boole, a aglutinação utiliza os Postulados P5, P6 e P3, enquanto a replicação emprega o Lema L2.

$$\text{Aglutinação} \begin{cases} (A \cdot B) + (A \cdot \overline{B}) = A \cdot (B + \overline{B}) = A \cdot 1 = A \\ (A + B) \cdot (A + \overline{B}) = A + (B \cdot \overline{B}) = A + 0 = A \end{cases} \tag{7.11}$$

$$\text{Replicação} \begin{cases} A = A + A + A + \ldots \\ A = A \cdot A \cdot A \cdot \ldots \end{cases} \tag{7.12}$$

7.5.2 Uso da aglutinação

A propriedade de distributividade mostra que, se dois termos diferem de apenas um literal (L e $\overline{L}$), eles podem ser fatorados. Surgem, desse modo, combinações do literal com seu complemento. Tais combinações geram valores lógicos 0 ou 1, os quais podem ser eliminados da expressão. Isto é exemplificado nas Equações (7.11) , (7.13) e (7.14).

$$\begin{aligned} F(A,B,C) &= \sum m(4,6) \\ &= m_4 + m_6 \\ &= (A \cdot \overline{B} \cdot \overline{C}) + (A \cdot B \cdot \overline{C}) \\ &= (A \cdot \overline{C} \cdot \overline{B}) + (A \cdot \overline{C} \cdot B) \\ &= [(A \cdot \overline{C}) \cdot \overline{B})] + [(A \cdot \overline{C}) \cdot B)] \\ &= (A \cdot \overline{C}) \cdot (\overline{B} + B) \\ &= (A \cdot \overline{C}) \cdot (1) \\ &= (A \cdot \overline{C}) \end{aligned} \tag{7.13}$$

$$\begin{aligned} F(A,B,C) &= \prod(0,4) \\ &= M_0 \cdot M_4 \\ &= (A+B+C)\cdot(\overline{A}+B+C) \\ &= [A+(B+C)]\cdot[\overline{A}+(B+C)] \\ &= (A\cdot\overline{A})+(B+C) \\ &= (0)+(B+C) \\ &= (B+C) \end{aligned} \tag{7.14}$$

7.5.3 Uso da replicação

Em algumas expressões, um mesmo termo tem a possibilidade de se combinar com diversos outros termos por aglutinação. Porém, formalmente, apenas dois termos podem ser combinados por vez. Logo, a operação de replicação possibilita que um mesmo termo seja combinado com diversos outros por aglutinação, possibilitando diversas simplificações. Isto é exemplificado na Equação (7.15).

$$\begin{aligned} F(A,B,C) &= \sum m(0,1,2) \\ &= m_0+m_1+m_2 \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C})+(\overline{A}\cdot\overline{B}\cdot C)+(\overline{A}\cdot B\cdot\overline{C}) \\ &= m_0+m_1+m_2+m_0 \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C})+(\overline{A}\cdot\overline{B}\cdot C)+(\overline{A}\cdot B\cdot\overline{C})+(\overline{A}\cdot\overline{B}\cdot\overline{C}) \\ &= [(\overline{A}\cdot\overline{B}\cdot\overline{C})+(\overline{A}\cdot\overline{B}\cdot C)]+[(\overline{A}\cdot B\cdot\overline{C})+(\overline{A}\cdot\overline{B}\cdot\overline{C})] \\ &= [(\overline{A}\cdot\overline{B})\cdot(\overline{C}+C)]+[(\overline{A}\cdot\overline{C})\cdot(\overline{B}+B)] \\ &= (\overline{A}\cdot\overline{B})+(\overline{A}\cdot\overline{C}) \\ &= (\overline{A})\cdot(\overline{B}+\overline{C}) \end{aligned} \tag{7.15}$$

7.6 Eliminação sistemática de literais

Aplicando-se as operações de aglutinação e de replicação às formas padrões SOP e POS, vários termos podem ser combinados, gerando um número menor de termos mais simples, por meio da eliminação de literais. A quantidade de literais eliminados depende do número de termos combinados e da configuração de literais em cada termo.

- Eliminação de 1 literal: exceto 1 literal, o qual será eliminado, todos os demais literais são idênticos em uma combinação de 2 termos normais.
- Eliminação de 2 literais: exceto 2 literais, os quais serão eliminados, todos os demais literais são idênticos em uma combinação de 4 termos normais.
- Eliminação de 3 literais: exceto 3 literais, os quais serão eliminados, todos os demais literais são idênticos em uma combinação de 8 termos normais.
- Eliminação de N literais: exceto N literais, os quais serão eliminados, todos os demais literais são idênticos em uma combinação de 2^N termos normais.

A título de exemplo, a Tabela 7.1 apresenta uma tabela verdade genérica para funções de 3 variáveis. Para tais funções, as Figuras 7.1 e 7.2 ilustram as possibilidades de eliminação de 1 e 2 literais em combinações de 2 e 4 mintermos, respectivamente. Por sua vez, as Figuras 7.3 e 7.4 ilustram as possibilidades de eliminação de 1 e 2 literais em combinações de 2 e 4 maxtermos, respectivamente.

Linha	A	B	C	$F(A,B,C)$
0	0	0	0	F_0
1	0	0	1	F_1
2	0	1	0	F_2
3	0	1	1	F_3
4	1	0	0	F_4
5	1	0	1	F_5
6	1	1	0	F_6
7	1	1	1	F_7

Tabela 7.1: Tabela verdade genérica para funções de 3 variáveis.

$$
\begin{aligned}
m_0 + m_1 &= (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (\overline{A} \cdot \overline{B} \cdot C) = (\overline{A} \cdot \overline{B}) \\
m_2 + m_3 &= (\overline{A} \cdot B \cdot \overline{C}) + (\overline{A} \cdot B \cdot C) = (\overline{A} \cdot B) \\
m_4 + m_5 &= (A \cdot \overline{B} \cdot \overline{C}) + (A \cdot \overline{B} \cdot C) = (A \cdot \overline{B}) \\
m_6 + m_7 &= (A \cdot B \cdot \overline{C}) + (A \cdot B \cdot C) = (A \cdot B)
\end{aligned}
$$

$$
\begin{aligned}
m_0 + m_2 &= (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (\overline{A} \cdot B \cdot \overline{C}) = (\overline{A} \cdot \overline{C}) \\
m_1 + m_3 &= (\overline{A} \cdot \overline{B} \cdot C) + (\overline{A} \cdot B \cdot C) = (\overline{A} \cdot C) \\
m_4 + m_6 &= (A \cdot \overline{B} \cdot \overline{C}) + (A \cdot B \cdot \overline{C}) = (A \cdot \overline{C}) \\
m_5 + m_7 &= (A \cdot \overline{B} \cdot C) + (A \cdot B \cdot C) = (A \cdot C)
\end{aligned}
$$

$$
\begin{aligned}
m_0 + m_4 &= (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (A \cdot \overline{B} \cdot \overline{C}) = (\overline{B} \cdot \overline{C}) \\
m_1 + m_5 &= (\overline{A} \cdot \overline{B} \cdot C) + (A \cdot \overline{B} \cdot C) = (\overline{B} \cdot C) \\
m_2 + m_6 &= (\overline{A} \cdot B \cdot \overline{C}) + (A \cdot B \cdot \overline{C}) = (B \cdot \overline{C}) \\
m_3 + m_7 &= (\overline{A} \cdot B \cdot C) + (A \cdot B \cdot C) = (B \cdot C)
\end{aligned}
$$

Figura 7.1: Eliminações de 1 literal em combinações de 2 mintermos.

$$
\begin{aligned}
&m_0 + m_1 + m_2 + m_3 = (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (\overline{A} \cdot \overline{B} \cdot C) + (\overline{A} \cdot B \cdot \overline{C}) + (\overline{A} \cdot B \cdot C) = (\overline{A}) \\
&m_4 + m_5 + m_6 + m_7 = (A \cdot \overline{B} \cdot \overline{C}) + (A \cdot \overline{B} \cdot C) + (A \cdot B \cdot \overline{C}) + (A \cdot B \cdot C) = (A) \\
&\\
&m_0 + m_1 + m_4 + m_5 = (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (\overline{A} \cdot \overline{B} \cdot C) + (A \cdot \overline{B} \cdot \overline{C}) + (A \cdot \overline{B} \cdot C) = (\overline{B}) \\
&m_2 + m_3 + m_6 + m_7 = (\overline{A} \cdot B \cdot \overline{C}) + (\overline{A} \cdot B \cdot C) + (A \cdot B \cdot \overline{C}) + (A \cdot B \cdot C) = (B) \\
&\\
&m_0 + m_2 + m_4 + m_6 = (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (\overline{A} \cdot B \cdot \overline{C}) + (A \cdot \overline{B} \cdot \overline{C}) + (A \cdot B \cdot \overline{C}) = (\overline{C}) \\
&m_1 + m_3 + m_5 + m_7 = (\overline{A} \cdot B \cdot \overline{C}) + (\overline{A} \cdot \overline{B} \cdot C) + (A \cdot \overline{B} \cdot C) + (A \cdot B \cdot C) = (C)
\end{aligned}
$$

Figura 7.2: Eliminações de 2 literais em combinações 4 de mintermos.

$$
\begin{aligned}
&M_0 \cdot M_1 = (A + B + C) \cdot (A + B + \overline{C}) = (A + B) \\
&M_2 \cdot M_3 = (A + \overline{B} + C) \cdot (A + \overline{B} + \overline{C}) = (A + \overline{B}) \\
&M_4 \cdot M_5 = (\overline{A} + B + C) \cdot (\overline{A} + B + \overline{C}) = (\overline{A} + B) \\
&M_6 \cdot M_7 = (\overline{A} + \overline{B} + C) \cdot (\overline{A} + \overline{B} + \overline{C}) = (\overline{A} + \overline{B}) \\
&\\
&M_0 \cdot M_2 = (A + B + C) \cdot (A + \overline{B} + C) = (A + C) \\
&M_1 \cdot M_3 = (A + B + \overline{C}) \cdot (A + \overline{B} + \overline{C}) = (A + \overline{C}) \\
&M_4 \cdot M_6 = (\overline{A} + B + C) \cdot (\overline{A} + \overline{B} + C) = (\overline{A} + C) \\
&M_5 \cdot M_7 = (\overline{A} + B + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) = (\overline{A} + \overline{C}) \\
&\\
&M_0 \cdot M_4 = (A + B + C) \cdot (\overline{A} + B + C) = (B + C) \\
&M_1 \cdot M_5 = (A + B + \overline{C}) \cdot (\overline{A} + B + \overline{C}) = (B + \overline{C}) \\
&M_2 \cdot M_6 = (A + \overline{B} + C) \cdot (\overline{A} + \overline{B} + C) = (\overline{B} + C) \\
&M_3 \cdot M_7 = (A + \overline{B} + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) = (\overline{B} + \overline{C})
\end{aligned}
$$

Figura 7.3: Eliminações de 1 literal em combinações de 2 maxtermos.

$$
\begin{aligned}
&M_0 \cdot M_1 \cdot M_2 \cdot M_3 = (A + B + C) \cdot (A + B + \overline{C}) \cdot (A + \overline{B} + C) \cdot (A + \overline{B} + \overline{C}) = (A) \\
&M_4 \cdot M_5 \cdot M_6 \cdot M_7 = (\overline{A} + B + C) \cdot (\overline{A} + B + \overline{C}) \cdot (\overline{A} + \overline{B} + C) \cdot (\overline{A} + \overline{B} + \overline{C}) = (\overline{A}) \\
&\\
&M_0 \cdot M_1 \cdot M_4 \cdot M_5 = (A + B + C) \cdot (A + B + \overline{C}) \cdot (\overline{A} + B + C) \cdot (\overline{A} + B + \overline{C}) = (B) \\
&M_2 \cdot M_3 \cdot M_6 \cdot M_7 = (A + \overline{B} + C) \cdot (A + \overline{B} + \overline{C}) \cdot (\overline{A} + \overline{B} + C) \cdot (\overline{A} + \overline{B} + \overline{C}) = (\overline{B}) \\
&\\
&M_0 \cdot M_2 \cdot M_4 \cdot M_6 = (A + B + C) \cdot (A + \overline{B} + C) \cdot (\overline{A} + B + C) \cdot (\overline{A} + \overline{B} + C) = (C) \\
&M_1 \cdot M_3 \cdot M_5 \cdot M_7 = (A + B + \overline{C}) \cdot (A + \overline{B} + \overline{C}) \cdot (\overline{A} + B + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) = (\overline{C})
\end{aligned}
$$

Figura 7.4: Eliminações de 2 literais em combinações 4 de maxtermos.

7.7 Implicantes e implicados

Quando uma função é expressa na forma SOP (AND-OR) ou POS (OR-AND), seus termos recebem uma denominação adicional, de acordo com o valor lógico que eles geram na tabela verdade da função: implicantes ou implicados. Tal denominação é descrita a seguir.

7.7.1 Implicantes

Quando uma função é expressa na forma SOP (AND-OR), cada termo produto é denominado de implicante (*implicant*). O nome se deve ao fato de que, caso o termo produto (implicante) assuma o valor lógico 1, isso implicará em um valor lógico 1 para a função.

No caso de uma SOP padrão, os implicantes são os próprios mintermos. Caso contrário, eles são o resultado de simplificações provenientes de combinações de mintermos.

A Equação (7.16) apresenta um exemplo de implicantes. Na primeira expressão, ela apresenta 3 implicantes, que são os mintermos responsáveis pelas 3 combinações lógicas de literais que fazem a função assumir o valor lógico 1. A segunda expressão apresenta 2 implicantes. O primeiro deles, sendo uma combinação de 2 mintermos, representa 2 combinações lógicas de literais capazes de produzir um valor lógico 1 para a função. O segundo deles, sendo um dos mintermos, representa a terceira combinação lógica de literais capaz de produzir um valor lógico 1 para a função.

$$\begin{aligned} F(A,B,C) &= \sum m(0,1,7) \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}) + (\overline{A}\cdot\overline{B}\cdot C) + (A\cdot B\cdot C) \\ &= (\overline{A}\cdot\overline{B}) + (A\cdot B\cdot C) \end{aligned} \tag{7.16}$$

7.7.2 Implicados

Quando uma função é expressa na forma POS (OR-AND), cada termo soma é denominado de implicado (*implicate*). O nome se deve ao fato de que, caso o termo soma (implicado) assuma o valor lógico 0, isso implicará em um valor lógico 0 para a função.

No caso de um POS padrão, os implicados são os próprios maxtermos. Caso contrário, eles são o resultado de simplificações provenientes de combinações de maxtermos.

A Equação (7.17) apresenta um exemplo de implicados. Na primeira expressão, ela apresenta 3 implicados, que são os maxtermos responsáveis pelas 3 combinações lógicas de literais que fazem a função assumir o valor lógico 0. A segunda expressão apresenta 2 implicados. O primeiro deles, sendo uma combinação de 2 maxtermos, representa 2 combinações lógicas de literais capazes de produzir um valor lógico 0 para a função. O segundo deles, sendo um dos maxtermos, representa a terceira combinação lógica de literais capaz de produzir um valor lógico 0 para a função.

$$\begin{aligned} F(A,B,C) &= \prod M(2,3,5) \\ &= (A+\overline{B}+C)\cdot(A+\overline{B}+\overline{C})\cdot(\overline{A}+B+\overline{C}) \\ &= (A+\overline{B})\cdot(\overline{A}+B+\overline{C}) \end{aligned} \tag{7.17}$$

7.7.3 Implicantes, implicados e o processo de simplificação

Pela definição apresentada, os termos normais expandidos das formas SOP padrão e POS padrão são implicantes e implicados que geram, respectivamente, apenas um único valor lógico "1" e "0" na tabela verdade da função por eles especificada.

A associação de 2^n implicantes ou implicados quaisquer, com o intuito de eliminar n literais, resulta em um único implicante ou implicado, acarretando a diminuição do número de tais termos.

Além disso, cada novo implicante e implicado, gerado pela associação de 2^n termos normais expandidos, passa a ser responsável, respectivamente, pela geração de 2^n valores lógicos "1" e "0" na tabela verdade da função por eles especificada.

Assim sendo, o processo de simplificação definido acima pode ser pensado como a busca do menor número possível de implicantes e implicados, cada um deles apresentando o menor número possível de literais e, conseqüentemente, cada um deles gerando, respectivamente, o maior número possível de valores lógicos "1" e "0" na tabela verdade da função por eles especificada.

7.8 Processo sistemático de simplificação

O ponto de partida do processo é expressar a função lógica nas formas SOP padrão e POS padrão. Em seguida, a operação de aglutinação é aplicada sucessivamente. Sempre que possível, a operação de replicação deve ser empregada, para maximizar a simplificação das expressão. Quando mais nenhuma aglutinação puder ser efetuada, a expressão restante será, naturalmente, a expressão definida anteriormente como mínima.

Em uma seleção sistemática, duas definições são de grande auxílio na escolha de termos a serem agrupados para simplificação: termo essencial e termo primo.

Quando um termo original é coberto por um único agrupamento possível, o termo resultante do agrupamento é denominado de termo essencial. Isso indica que os termos essenciais devem ser incluídos na expressão mínima equivalente à função desejada.

Um termo original que não tenha sido coberto por qualquer agrupamento anterior deve ser incluído em um agrupamento máximo, o qual será denominado de termo primo. Isso indica que os termos primos devem ser incluídos na expressão mínima equivalente à função desejada.

Deve ser ressaltado que nem todo agrupamento máximo representa um termo primo.

Assim, uma forma sistemática de escolha de termos é:

S1 - Identificar todas as possibilidades de agrupamento, através dos maiores grupos possíveis.

S2 - Marcar todos os termos originais cobertos por apenas 1 agrupamento. Tais agrupamentos formam os termos essenciais.

S3 - Listar todos os termos essenciais.

S4 - Usar os maiores agrupamentos possíveis para cobrir os termos originais não cobertos pelos termos essenciais, formando termos primos.

S5 - Listar apenas tais termos primos.

S6 - Montar a expressão mínima, a partir das duas listas.

No caso da existência de diversas formas mínimas equivalentes, em uma SOP ou em uma POS, deve-se aplicar algum critério extra para a escolha final.

Dada uma função, e suas formas SOP e POS, nada se pode garantir em relação a qual das duas conduzirá à expressão mais simples. Assim, é necessário encontrar a forma mínima de ambas e decidir qual delas é a mais simples.

7.9 Subjetividade, complexidade e formas alternativas

A existência de Postulados, Lemas e Teoremas, possibilita, mas não facilita, o processo de simplificação de equações booleanas. O emprego aleatório de tais ferramentas induz uma imensa subjetividade no processo, ao mesmo tempo que não garante que a simplificação máxima seja encontrada.

O processo sistemático de simplificação, usando replicação e aglutinação, acarreta uma grande redução na subjetividade do processo. Porém, para equações com muitos literais e/ou muitos termos, o processo ainda irá apresentar um certo grau de subjetividade e um bom grau de complexidade, no que se refere: à escolha dos termos a serem replicados e à escolha dos termos a serem aglutinados.

A fim de se tornar o processo de minimização ainda menos subjetivo e menos complexo, pode-se realizá-lo não diretamente sobre as equações, mas, alternativamente, sobre uma forma pictórica de representação ou através de um procedimento computacional. Em ambos os casos, as operações básicas são as mesmas, porém realizadas sobre outras formas de expressão. As alternativas comumente empregadas são o Mapa de Karnaugh e o Algoritmo Tabular de Quine-McCluskey.

7.10 Minimização com estruturas de ordem superior: fatoração extra

Deve-se notar que a fatoração de termos em uma expressão booleana sempre acarreta um aumento no número de planos de lógica, gerando estruturas com mais níveis ou de ordens maiores. Este aumento de níveis traduz-se em um aumento no tempo de propagação da estrutura.

No caso particular onde surge uma simplificação após a fatoração, o número de planos de lógica não se altera, mantendo o número de níveis ou a ordem da estrutura. Nesses casos, o que ocorre é uma compensação causada pelas relações $\left(L \cdot \overline{L}\right) = 0$ e $\left(L + \overline{L}\right) = 1$, que acabam por suprimir o plano extra criado pela fatoração. Por exemplo, em uma fatoração com supressão de termo, tem-se que

$$\begin{array}{rcll} f(A,B,C) & = & (A \cdot B) + (A \cdot \overline{B}) + (B \cdot C) & \rightarrow \ 2 \text{ níveis} \\ & = & [A \cdot (B + \overline{B})] + (B \cdot C) & \rightarrow \ 3 \text{ níveis} \\ & = & [A \cdot (1)] + (B \cdot C) & \rightarrow \ 2 \text{ níveis} \\ & = & (A) + (B \cdot C) & \rightarrow \ 2 \text{ níveis} \ , \end{array}$$

e, em uma fatoração com supressão de termo e de entrada, tem-se que

$$\begin{array}{rcll} f(A,B,C,D) & = & (A \cdot B \cdot C) + (A \cdot \overline{B} \cdot C) + (\overline{A} \cdot C \cdot \overline{D}) + (\overline{A} \cdot \overline{C} \cdot \overline{D}) & \rightarrow \ 2 \text{ níveis} \\ & = & [(A \cdot C) \cdot (B + \overline{B})] + [(\overline{A} \cdot \overline{D}) \cdot (C + \overline{C})] & \rightarrow \ 3 \text{ níveis} \\ & = & [(A \cdot C) \cdot (1)] + [(\overline{A} \cdot \overline{D}) \cdot (1)] & \rightarrow \ 2 \text{ níveis} \\ & = & (A \cdot C) + (\overline{A} \cdot \overline{D}) & \rightarrow \ 2 \text{ níveis} \ . \end{array}$$

Em alguns casos, partindo-se de uma expressão booleana padrão (SOP ou POS) e aplicando-se as fatorações adequadas (replicação e aglutinação), chega-se a uma expressão de segunda

ordem absolutamente mínima (SOP ou POS). Em alguns outros casos, a expressão de segunda ordem mínima representa um mínimo local e ainda permite fatorações, mas que não acarretarão simplificações e gerarão uma estrutura de ordem superior. Em casos extremos, a própria expressão padrão já é uma expressão de segunda ordem mínima e, ainda que seja possível realizar fatorações, elas também não acarretarão simplificações e também gerarão uma estrutura de ordem superior.

Em aplicações onde o tempo de resposta é um parâmetro importante, é recomendado o emprego de um estrutura de segunda ordem mínima, ainda que ela não represente a estrutura mínima absoluta. Por outro lado, quando o tempo de resposta não é decisivo, pode-se tentar encontrar uma fatoração favorável que leve à estrutura de mínimo absoluto. Nesse caso, a dificuldade é encontrar um modo simples e prático de se realizar tais fatorações em uma expressão booleana qualquer.

7.11 Minimização vs *hazards* e *glitches*

Dado um circuito digital combinacional, sempre que algum sinal de entrada é modificado, ocorre um período transitório até que o sinal de saída seja considerado estável. Durante tais períodos transitórios, podem ocorrer transições inesperadas, nos sinais internos e no sinal de saída, denominadas de *hazards*. Por sua vez, *hazards* podem gerar breves pulsos inesperados, denominados de *glitches*.

Hazards e *glitches* podem causar, pelo menos, dois problemas. Por um lado, eles representam mudanças inesperadas nos sinais e, portanto, podem causar algum tipo de funcionamento errôneo do circuito. Por outro lado, nas implementações mais comumente empregadas, o consumo de energia ocorre durante as transições dos sinais. Assim, em aplicações que requerem baixo consumo, as variações de sinais desnecessárias representam um consumo de energia desnecessário.

Uma das formas de eliminar a possibilidade de ocorrência de *hazards* e *glitches*, em um circuito digital combinacional, é a inserção de termos adicionais na sua equação mínima, que passam a representar redundâncias necessárias. Nesta solução, as redundâncias mascaram a ocorrência do *hazard*, que é provocada pelos caminhos não equalizados dos sinais. Por exemplo, supondo a função booleana dada pela seguinte forma SOP padrão:

$$f(A,B,C) = (\overline{A}\cdot B\cdot C) + (A\cdot B\cdot C) + (A\cdot B\cdot \overline{C}) + (A\cdot \overline{B}\cdot \overline{C}) = m_3 + m_7 + m_6 + m_4 \ ,$$

podem-se formar as seguintes aglutinações: $(m_3 - m_7)$, $(m_6 - m_4)$ e $(m_7 - m_6)$. Porém, a aglutinação $(m_7 - m_6)$ é redundante. Assim, a SOP mínima de $f(A,B,C)$ é dada por

$$f(A,B,C) = (m_3 - m_7) + (m_6 - m_4) = (B\cdot C) + (A\cdot \overline{C}) \ . \tag{7.18}$$

Supondo a implementação da Equação (7.18) com o conjunto { 1 NOT, 1 OR, 2 ANDs }, as entradas $ABC = 111$ e $ABC = 110$ geram a saída $f(A,B,C) = 1$. Logo, do ponto de vista comportamental, na transição $ABC = 111 \rightarrow 110$, a saída deveria permanecer $f(A,B,C) = 1$. Porém, no circuito, na transição $ABC = 111 \rightarrow 110$, o implicante $(B\cdot C)$ torna-se 0 antes que o implicante $(A\cdot \overline{C})$ torne-se 1, devido ao atraso de propagação do inversor. Isto faz com que a saída apresente o comportamento inesperado dado por $f(A,B,C) = 1 \rightarrow 0 \rightarrow 1$. Para evitar este *glitch* na saída, a solução, usando redundância, é a adicionar o implicante $(m_7 - m_6) = (A\cdot B)$ na Equação (7.18), que se torna

$$f(A,B,C) = (m_3 - m_7) + (m_6 - m_4) + (m_7 - m_6) = (B\cdot C) + (A\cdot \overline{C}) + (A\cdot B) \ . \tag{7.19}$$

Dessa forma, na transição $ABC = 111 \rightarrow 110$, o implicante $(A\cdot B)$ mantém-se em 1, enquanto os demais implicantes estão em transição, antes de se estabilizarem. Portanto, a saída é mantida fixa no valor 1, durante todo o transitório, evitando o *glitch* na saída.

7.12 Exemplos de simplificação

A seguir, são apresentados alguns exemplos de simplificação algébrica sistemática, para funções de três e quatro variáveis, empregando as técnicas de aglutinação e, quando necessário, de replicação.

7.12.1 Funções de três variáveis

Exemplo 1

Seja a função especificada por $f(A,B,C) = \sum m(0,3,5,6)$.

A equação literal da SOP padrão é dada por

$$f(A,B,C) = \sum m(0,3,5,6) = (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (\overline{A} \cdot B \cdot C) + (A \cdot \overline{B} \cdot C) + (A \cdot B \cdot \overline{C}) \ .$$

Analisando-se os mintermos, verifica-se que não há possibilidades de aglutinação. Portanto, a SOP padrão já é a SOP mínima.

Por sua vez, a equação literal da POS padrão é dada por

$$f(A,B,C) = \prod M(1,2,4,7) = (A + B + \overline{C}) \cdot (A + \overline{B} + C) \cdot (\overline{A} + B + C) \cdot (\overline{A} + \overline{B} + \overline{C}) \ .$$

Analisando-se os maxtermos, verifica-se que não há possibilidades de aglutinação. Portanto, a POS padrão já é a POS mínima.

Exemplo 2

Seja a função especificada por $f(A,B,C) = \sum m(0,2,5,7)$.

A equação literal da SOP padrão é dada por

$$f(A,B,C) = \sum m(0,2,5,7) = (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (\overline{A} \cdot B \cdot \overline{C}) + (A \cdot \overline{B} \cdot C) + (A \cdot B \cdot C) \ .$$

Analisando-se os mintermos, verifica-se que há as seguintes possibilidades de aglutinação: $(m_0 - m_2)$ e $(m_5 - m_7)$. Portanto, a SOP mínima é dada por

$$\begin{aligned} f(A,B,C) &= \sum m(0,2,5,7) \\ &= \Big[(\overline{A} \cdot \overline{B} \cdot \overline{C}) + (\overline{A} \cdot B \cdot \overline{C})\Big] + \Big[(A \cdot \overline{B} \cdot C) + (A \cdot B \cdot C)\Big] \\ &= \Big[(\overline{A} \cdot \overline{C}) \cdot (\overline{B} + B)\Big] + \Big[(A \cdot C) \cdot (\overline{B} + B)\Big] \\ &= \Big[(\overline{A} \cdot \overline{C}) \cdot (1)\Big] + [(A \cdot C) \cdot (1)] \\ &= (\overline{A} \cdot \overline{C}) + (A \cdot C) \ . \end{aligned}$$

Por sua vez, a equação literal da POS padrão é dada por

$$f(A,B,C) = \prod M(1,3,4,6) = (A+B+\overline{C})\cdot(A+\overline{B}+\overline{C})\cdot(\overline{A}+B+C)\cdot(\overline{A}+\overline{B}+C)\ .$$

Analisando-se os maxtermos, verifica-se que há as seguintes possibilidades de aglutinação: $(M_1 - M_3)$ e $(M_4 - M_6)$. Portanto, a POS mínima é dada por

$$\begin{aligned} f(A,B,C) &= \prod M(1,3,4,6) \\ &= \Big[(A+B+\overline{C})\cdot(A+\overline{B}+\overline{C})\Big]\cdot\Big[(\overline{A}+B+C)\cdot(\overline{A}+\overline{B}+C)\Big] \\ &= \Big[(A+\overline{C})+(B\cdot\overline{B})\Big]\cdot\Big[(\overline{A}+C)+(B\cdot\overline{B})\Big] \\ &= \Big[(A+\overline{C})+(0)\Big]\cdot\Big[(\overline{A}+C)+(0)\Big] \\ &= (A+\overline{C})\cdot(\overline{A}+C)\ . \end{aligned}$$

Exemplo 3

Seja a função especificada por $f(A,B,C) = \sum m(2,5,6,7)$.
A equação literal da SOP padrão é dada por

$$f(A,B,C) = \sum m(2,5,6,7) = (\overline{A}\cdot B\cdot\overline{C}) + (A\cdot\overline{B}\cdot C) + (A\cdot B\cdot\overline{C}) + (A\cdot B\cdot C)\ .$$

Analisando-se os mintermos, verifica-se que há as seguintes possibilidades de aglutinação: $(m_2 - m_6)$, $(m_5 - m_7)$ e $(m_6 - m_7)$. Observando-se que as aglutinações $(m_2 - m_6)$ e $(m_5 - m_7)$ são essenciais e já cobrem todos os valores booleanos "1" da função, a aglutinação $(m_6 - m_7)$ torna-se redundante, não precisando ser realizada. Portanto, a SOP mínima é dada por

$$\begin{aligned} f(A,B,C) &= \sum m(2,5,6,7) \\ &= \Big[(\overline{A}\cdot B\cdot\overline{C}) + (A\cdot B\cdot\overline{C})\Big] + \Big[(A\cdot\overline{B}\cdot C) + (A\cdot B\cdot C)\Big] \\ &= \Big[(B\cdot\overline{C})\cdot(\overline{A}+A)\Big] + \Big[(A\cdot C)+(\overline{B}+B)\Big] \\ &= \Big[(B\cdot\overline{C})\cdot(1)\Big] + [(A\cdot C)\cdot(1)] \\ &= (B\cdot\overline{C}) + (A\cdot C)\ . \end{aligned}$$

Por sua vez, a equação literal da POS padrão é dada por

$$f(A,B,C) = \prod M(0,1,3,4) = (A+B+C)\cdot(A+B+\overline{C})\cdot(A+\overline{B}+\overline{C})\cdot(\overline{A}+B+C)\ .$$

Analisando-se os maxtermos, verifica-se que há as seguintes possibilidades de aglutinação: $(M_0 - M_1)$, $(M_1 - M_3)$ e $(M_0 - M_4)$. Observando-se que as aglutinações $(M_0 - M_4)$ e $(M_1 - M_3)$ são essenciais e já cobrem todos os valores booleanos "0" da função, a aglutinação $(M_0 - M_1)$ torna-se redundante, não precisando ser realizada. Portanto, a POS mínima é dada por

$$\begin{aligned} f(A,B,C) &= \prod M(0,1,3,4) \\ &= \Big[(A+B+C)\cdot(\overline{A}+B+C)\Big]\cdot\Big[(A+B+\overline{C})\cdot(A+\overline{B}+\overline{C})\Big] \\ &= \Big[(B+C)+(A\cdot\overline{A})\Big]\cdot\Big[(A+\overline{C})+(B\cdot\overline{B})\Big] \\ &= [(B+C)+(0)]\cdot\Big[(A+\overline{C})+(0)\Big] \\ &= (B+C)\cdot(A+\overline{C})\ . \end{aligned}$$

Exemplo 4

Seja a função especificada por $f(A,B,C) = \sum m(2,6,7)$.

A equação literal da SOP padrão é dada por

$$f(A,B,C) = \sum m(2,6,7) = (\overline{A} \cdot B \cdot \overline{C}) + (A \cdot B \cdot \overline{C}) + (A \cdot B \cdot C) \ .$$

Analisando-se os mintermos, verifica-se que há as seguintes possibilidades de aglutinação: $(m_2 - m_6)$ e $(m_6 - m_7)$. Observando-se que as aglutinações $(m_2 - m_6)$ e $(m_6 - m_7)$ são essenciais, deve-se duplicar o mintermo m_6 para que se obtenha a simplificação máxima. Portanto, a SOP mínima é dada por

$$\begin{aligned}
f(A,B,C) &= \sum m(2,6,7) \\
&= \Big[(\overline{A} \cdot B \cdot \overline{C}) + (A \cdot B \cdot \overline{C})\Big] + \Big[(A \cdot B \cdot \overline{C}) + (A \cdot B \cdot C)\Big] \\
&= \Big[(B \cdot \overline{C}) \cdot (\overline{A} + A)\Big] + \Big[(A \cdot B) + (\overline{C} + C)\Big] \\
&= \Big[(B \cdot \overline{C}) \cdot (1)\Big] + [(A \cdot B) + (1)] \\
&= (B \cdot \overline{C}) + (A \cdot B) \ .
\end{aligned}$$

Por sua vez, a equação literal da POS padrão é dada por

$$f(A,B,C) = \prod M(0,1,3,4,5) = (A+B+C)\cdot(A+B+\overline{C})\cdot(A+\overline{B}+\overline{C})\cdot(\overline{A}+B+C)\cdot(\overline{A}+B+\overline{C}) \ .$$

Analisando-se os maxtermos, verifica-se que há as seguintes possibilidades de aglutinação: $(M_0 - M_1)$, $(M_0 - M_4)$, $(M_1 - M_3)$, $(M_1 - M_5)$ e $(M_4 - M_5)$. Observando-se que a aglutinação $(M_1 - M_3)$ é essencial, as seguintes aglutinações devem ser feitas para cobrir todos os valores booleanos "0" e para simplificar ao máximo a função: $\{(M_1 - M_3), (M_0 - M_1), (M_4 - M_5)\}$ ou $\{(M_1 - M_3), (M_0 - M_4), (M_1 - M_5)\}$. Em qualquer das duas opções, deve-se duplicar o maxtermo M_1.

Empregando-se as aglutinações $(M_1 - M_3)$, $(M_0 - M_1)$ e $(M_4 - M_5)$, a POS mínima é dada por

$$\begin{aligned}
f(A,B,C) &= \prod M(0,1,3,4,5) \\
&= \Big[(A + B + \overline{C}) \cdot (A + \overline{B} + \overline{C})\Big] \cdot \\
&\quad \Big[(A + B + C) \cdot (A + B + \overline{C})\Big] \cdot \Big[(\overline{A} + B + C)(\overline{A} + B + \overline{C})\Big] \\
&= \Big[(A + \overline{C}) + (B \cdot \overline{B})\Big] \cdot \Big[(A + B) + (C \cdot \overline{C})\Big] \cdot \Big[(\overline{A} + B) + (C \cdot \overline{C})\Big] \\
&= \Big[(A + \overline{C}) + (0)\Big] \cdot [(A + B) + (0)] \cdot \Big[(\overline{A} + B) + (0)\Big] \\
&= (A + \overline{C}) \cdot (A + B) \cdot (\overline{A} + B) \\
&= (A + \overline{C}) \cdot \Big[B + (A \cdot \overline{A})\Big] \\
&= (A + \overline{C}) \cdot [B + (0)] \\
&= (A + \overline{C}) \cdot (B) \ .
\end{aligned}$$

Nesse caso, ocorre a dupla aglutinação $[(M_0 - M_1) - (M_4 - M_5)]$. Inicialmente, a variável C é dispensada. Em seguida, isso acontece com a variável A.

Empregando-se as aglutinações $(M_1 - M_3)$, $(M_0 - M_4)$ e $(M_1 - M_5)$, a POS mínima é dada por

$$\begin{aligned}
f(A,B,C) &= \prod M(0,1,3,4,5) \\
&= \Big[(A+B+\overline{C})\cdot(A+\overline{B}+\overline{C})\Big]\cdot \\
&\quad \Big[(A+B+C)\cdot(\overline{A}+B+C)\Big]\cdot\Big[(A+B+\overline{C})\cdot(\overline{A}+B+\overline{C})\Big] \\
&= \Big[(A+\overline{C})+(B\cdot\overline{B})\Big]\cdot\Big[(B+C)+(A\cdot\overline{A})\Big]\cdot\Big[(B+\overline{C})+(A\cdot\overline{A})\Big] \\
&= \Big[(A+\overline{C})+(0)\Big]\cdot[(B+C)+(0)]\cdot\Big[(B+\overline{C})+(0)\Big] \\
&= (A+\overline{C})\cdot(B+C)\cdot(B+\overline{C}) \\
&= (A+\overline{C})\cdot\Big[B+(C\cdot\overline{C})\Big] \\
&= (A+\overline{C})\cdot[B+(0)] \\
&= (A+\overline{C})\cdot(B)\ .
\end{aligned}$$

Nesse caso, ocorre a dupla aglutinação $[(M_0 - M_4) - (M_1 - M_5)]$. Inicialmente, a variável A é dispensada. Em seguida, isso acontece com a variável C.

Exemplo 5

Seja a função especificada por $f(A,B,C) = \sum m(2,4,6,7)$.
A equação literal da SOP padrão é dada por

$$f(A,B,C) = \sum m(2,4,6,7) = (\overline{A}\cdot B\cdot\overline{C}) + (A\cdot\overline{B}\cdot\overline{C}) + (A\cdot B\cdot\overline{C}) + (A\cdot B\cdot C)\ .$$

Analisando-se os mintermos, verifica-se que há as seguintes possibilidades de aglutinação: $(m_2 - m_6)$, $(m_4 - m_6)$ e $(m_6 - m_7)$. Observando-se que as aglutinações $(m_2 - m_6)$, $(m_4 - m_6)$ e $(m_6 - m_7)$ são essenciais, deve-se triplicar o mintermo m_6 para que se obtenha a simplificação máxima. Portanto, a SOP mínima é dada por

$$\begin{aligned}
f(A,B,C) &= \sum m(2,4,6,7) \\
&= \Big[(\overline{A}\cdot B\cdot\overline{C})+(A\cdot B\cdot\overline{C})\Big]+ \\
&\quad \Big[(A\cdot\overline{B}\cdot\overline{C})+(A\cdot B\cdot\overline{C})\Big]+\Big[(A\cdot B\cdot\overline{C})+(A\cdot B\cdot C)\Big] \\
&= \Big[(B\cdot\overline{C})\cdot(\overline{A}+A)\Big]+\Big[(A\cdot\overline{C})\cdot(\overline{B}+B)\Big]+\Big[(A\cdot B)+(\overline{C}+C)\Big] \\
&= \Big[(B\cdot\overline{C})\cdot(1)\Big]+\Big[(A\cdot\overline{C})\cdot(1)\Big]+[(A\cdot B)+(1)] \\
&= (B\cdot\overline{C})+(A\cdot\overline{C})+(A\cdot B)\ .
\end{aligned}$$

Por sua vez, a equação literal da POS padrão é dada por

$$f(A,B,C) = \prod M(0,1,3,5) = (A+B+C)\cdot(A+B+\overline{C})\cdot(A+\overline{B}+\overline{C})\cdot(\overline{A}+B+\overline{C})\ .$$

Analisando-se os maxtermos, verifica-se que há as seguintes possibilidades de aglutinação: $(M_0 - M_1)$, $(M_1 - M_3)$ e $(M_1 - M_5)$. Observando-se que as aglutinações $(M_0 - M_1)$, $(M_1 - M_3)$ e $(M_1 - M_5)$ são essenciais, deve-se triplicar o maxtermo M_1 para que se obtenha a simplificação máxima. Portanto, a POS mínima é dada por

$$
\begin{aligned}
f(A,B,C) &= \prod M(0,1,3,5) \\
&= \Big[(A+B+C)\cdot(A+B+\overline{C})\Big]\cdot \\
&\quad \Big[(A+B+\overline{C})\cdot(A+\overline{B}+\overline{C})\Big]\cdot\Big[(A+B+\overline{C})\cdot(\overline{A}+B+\overline{C})\Big] \\
&= \Big[(A+B)+(C\cdot\overline{C})\Big]\cdot\Big[(A+\overline{C})+(B\cdot\overline{B})\Big]\cdot\Big[(B+\overline{C})+(A\cdot\overline{A})\Big] \\
&= [(A+B)+(0)]\cdot\Big[(A+\overline{C})+(0)\Big]\cdot\Big[(B+\overline{C})+(0)\Big] \\
&= (A+B)\cdot(A+\overline{C})\cdot(B+\overline{C})\ .
\end{aligned}
$$

Exemplo 6

Seja a função especificada por $f(A,B,C)=\sum m(2,4,5,6,7)$.

A equação literal da SOP padrão é dada por

$$f(A,B,C)=\sum m(2,4,5,6,7)=(\overline{A}\cdot B\cdot\overline{C})+(A\cdot\overline{B}\cdot\overline{C})+(A\cdot\overline{B}\cdot C)+(A\cdot B\cdot\overline{C})+(A\cdot B\cdot C)\ .$$

Analisando-se os mintermos, verifica-se que há as seguintes possibilidades de aglutinação: (m_2-m_6), (m_4-m_5), (m_4-m_6), (m_5-m_7) e (m_6-m_7). Observando-se que a aglutinação (m_2-m_6) é essencial, as seguintes aglutinações devem ser feitas para cobrir todos os valores booleanos "1" e para simplificar ao máximo a função: $\{(m_2-m_6),(m_4-m_5),(m_6-m_7)\}$ ou $\{(m_2-m_6),(m_4-m_6),(m_5-m_7)\}$. Em qualquer das duas opções, deve-se duplicar o mintermo m_6.

Empregando-se as aglutinações (m_2-m_6), (m_4-m_5) e (m_6-m_7), a SOP mínima é dada por

$$
\begin{aligned}
f(A,B,C) &= \sum m(2,4,5,6,7) \\
&= \Big[(\overline{A}\cdot B\cdot\overline{C})+(A\cdot B\cdot\overline{C})\Big]+ \\
&\quad \Big[(A\cdot\overline{B}\cdot\overline{C})+(A\cdot\overline{B}\cdot C)\Big]+\Big[(A\cdot B\cdot\overline{C})+(A\cdot B\cdot C)\Big] \\
&= \Big[(B\cdot\overline{C})\cdot(\overline{A}+A)\Big]+\Big[(A\cdot\overline{B})\cdot(\overline{C}+C)\Big]+\Big[(A\cdot B)\cdot(\overline{C}+C)\Big] \\
&= \Big[(B\cdot\overline{C})\cdot(1)\Big]+\Big[(A\cdot\overline{B})\cdot(1)\Big]+[(A\cdot B)\cdot(1)] \\
&= (B\cdot\overline{C})+(A\cdot\overline{B})+(A\cdot B) \\
&= (B\cdot\overline{C})+\Big[A\cdot(\overline{B}+B)\Big] \\
&= (B\cdot\overline{C})+[A\cdot(1)] \\
&= (B\cdot\overline{C})+(A)\ .
\end{aligned}
$$

Nesse caso, ocorre a dupla aglutinação $[(m_4-m_5)-(m_6-m_7)]$. Inicialmente, a variável C é dispensada. Em seguida, isso acontece com a variável B.

Empregando-se as aglutinações $(m_2 - m_6)$, $(m_4 - m_6)$ e $(m_5 - m_7)$, a SOP mínima é dada por

$$\begin{aligned} f(A,B,C) &= \sum m(2,4,5,6,7) \\ &= \Big[(\overline{A}\cdot B\cdot \overline{C}) + (A\cdot B\cdot \overline{C})\Big] + \\ &\quad \Big[(A\cdot \overline{B}\cdot \overline{C}) + (A\cdot B\cdot \overline{C})\Big] + \Big[(A\cdot \overline{B}\cdot C) + (A\cdot B\cdot C)\Big] \\ &= \Big[(B\cdot \overline{C})\cdot(\overline{A}+A)\Big] + \Big[(A\cdot \overline{C})\cdot(\overline{B}+B)\Big] + \Big[(A\cdot C)\cdot(\overline{B}+B)\Big] \\ &= \Big[(B\cdot \overline{C})\cdot(1)\Big] + \Big[(A\cdot \overline{C})\cdot(1)\Big] + [(A\cdot C)\cdot(1)] \\ &= (B\cdot \overline{C}) + (A\cdot \overline{C}) + (A\cdot C) \\ &= (B\cdot \overline{C}) + \Big[A\cdot(\overline{C}+C)\Big] \\ &= (B\cdot \overline{C}) + [A\cdot(1)] \\ &= (B\cdot \overline{C}) + (A)\ . \end{aligned}$$

Nesse caso, ocorre a dupla aglutinação $[(m_4 - m_6) - (m_5 - m_7)]$. Inicialmente, a variável B é dispensada. Em seguida, isso acontece com a variável C.

Por sua vez, a equação literal da POS padrão é dada por

$$f(A,B,C) = \prod M(0,1,3) = (A+B+C)\cdot(A+B+\overline{C})\cdot(A+\overline{B}+\overline{C})\ .$$

Analisando-se os maxtermos, verifica-se que há as seguintes possibilidades de aglutinação: $(M_0 - M_1)$ e $(M_1 - M_3)$. Observando-se que as aglutinações $(M_0 - M_1)$ e $(M_1 - M_3)$ são essenciais, deve-se duplicar o maxtermo M_1 para que se obtenha a simplificação máxima. Portanto, a POS mínima é dada por

$$\begin{aligned} f(A,B,C) &= \prod M(0,1,3) \\ &= \Big[(A+B+C)\cdot(A+B+\overline{C})\Big]\cdot\Big[(A+B+\overline{C})\cdot(A+\overline{B}+\overline{C})\Big] \\ &= \Big[(A+B)+(C\cdot\overline{C})\Big]\cdot\Big[(A+\overline{C})+(B\cdot\overline{B})\Big] \\ &= [(A+B)+(0)]\cdot\Big[(A+\overline{C})+(0)\Big] \\ &= (A+B)+(A+\overline{C})\ . \end{aligned}$$

7.12.2 Funções de quatro variáveis

Exemplo 1

Seja a função especificada por $f(A,B,C,D) = \sum m(0,5,7,10,14)$.

A equação literal da SOP padrão é dada por

$$\begin{aligned} f(A,B,C,D) &= \sum m(0,5,7,10,14) \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \\ &\quad (\overline{A}\cdot B\cdot\overline{C}\cdot D) + (\overline{A}\cdot B\cdot C\cdot D) + (A\cdot\overline{B}\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot\overline{D})\ . \end{aligned}$$

Analisando-se os mintermos, verifica-se que há as seguintes possibilidades de aglutinação: $(m_5 - m_7)$ e $(m_{10} - m_{14})$. Portanto, a SOP mínima é dada por

$$\begin{aligned} f(A,B,C,D) &= \sum m(0,5,7,10,14) \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \\ &\quad \Big[(\overline{A}\cdot B\cdot\overline{C}\cdot D) + (\overline{A}\cdot B\cdot C\cdot D)\Big] + \Big[(A\cdot\overline{B}\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot\overline{D})\Big] \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \\ &\quad \Big[(\overline{A}\cdot B\cdot D)\cdot(\overline{C}+C)\Big] + \Big[(A\cdot C\cdot\overline{D})\cdot(\overline{B}+B)\Big] \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \\ &\quad \Big[(\overline{A}\cdot B\cdot D)\cdot(1)\Big] + \Big[(A\cdot C\cdot\overline{D})\cdot(1)\Big] \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot D) + (A\cdot C\cdot\overline{D})\ . \end{aligned}$$

Exemplo 2

Seja a função especificada por $f(A,B,C,D) = \sum m(0,7,10,14,15)$.
A equação literal da SOP padrão é dada por

$$\begin{aligned} f(A,B,C,D) &= \sum m(0,7,10,14,15) \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \\ &\quad (\overline{A}\cdot B\cdot C\cdot D) + (A\cdot\overline{B}\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot D)\ . \end{aligned}$$

Analisando-se os mintermos, verifica-se que há as seguintes possibilidades de aglutinação: $(m_7 - m_{15})$, $(m_{10} - m_{14})$ e $(m_{14} - m_{15})$. Observando-se que as aglutinações $(m_7 - m_{15})$ e $(m_{10} - m_{14})$ são essenciais e já cobrem todos os valores booleanos "1" da função, a aglutinação $(m_{14} - m_{15})$ torna-se redundante, não precisando ser realizada. Portanto, a SOP mínima é dada por

$$\begin{aligned} f(A,B,C,D) &= \sum m(0,7,10,14,15) \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \\ &\quad \Big[(\overline{A}\cdot B\cdot C\cdot D) + (A\cdot B\cdot C\cdot D)\Big] + \Big[(A\cdot\overline{B}\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot\overline{D})\Big] \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \\ &\quad \Big[(B\cdot C\cdot D)\cdot(\overline{A}+A)\Big] + \Big[(A\cdot C\cdot\overline{D})\cdot(\overline{B}+B)\Big] \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \\ &\quad [(B\cdot C\cdot D)\cdot(1)] + \Big[(A\cdot C\cdot\overline{D})\cdot(1)\Big] \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + (B\cdot C\cdot D) + (A\cdot C\cdot\overline{D})\ . \end{aligned}$$

Exemplo 3

Seja a função especificada por $f(A,B,C,D) = \sum m(0,6,10,14,15)$.

A equação literal da SOP padrão é dada por

$$\begin{aligned} f(A,B,C,D) &= \sum m(0,6,10,14,15) \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \\ &\quad (\overline{A}\cdot B\cdot C\cdot\overline{D}) + (A\cdot\overline{B}\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot D)\ . \end{aligned}$$

Analisando-se os mintermos, verifica-se que há as seguintes possibilidades de aglutinação: $(m_6 - m_{14})$, $(m_{10} - m_{14})$ e $(m_{14} - m_{15})$. Observando-se que as aglutinações $(m_6 - m_{14})$, $(m_{10} - m_{14})$ e $(m_{14} - m_{15})$ são essenciais, deve-se triplicar o mintermo m_{14} para que se obtenha a simplificação máxima. Portanto, a SOP mínima é dada por

$$\begin{aligned} f(A,B,C,D) &= \sum m(0,6,10,14,15) \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \Big[(\overline{A}\cdot B\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot\overline{D})\Big] + \\ &\quad \Big[(A\cdot\overline{B}\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot\overline{D})\Big] + \Big[(A\cdot B\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot D)\Big] \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \Big[(B\cdot C\cdot\overline{D})\cdot(\overline{A}+A)\Big] + \\ &\quad \Big[(A\cdot C\cdot\overline{D})\cdot(\overline{B}+B)\Big] + \Big[(A\cdot B\cdot C)\cdot(\overline{D}+D)\Big] \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \Big[(B\cdot C\cdot\overline{D})\cdot(1)\Big] + \\ &\quad \Big[(A\cdot C\cdot\overline{D})\cdot(1)\Big] + [(A\cdot B\cdot C)\cdot(1)] \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + (B\cdot C\cdot\overline{D}) + (A\cdot C\cdot\overline{D}) + (A\cdot B\cdot C)\ . \end{aligned}$$

Exemplo 4

Seja a função especificada por $f(A,B,C,D) = \sum m(0,5,7,10,11,14,15)$.

A equação literal da SOP padrão é dada por

$$\begin{aligned} f(A,B,C,D) &= \sum m(0,5,7,10,11,14,15) \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \\ &\quad (\overline{A}\cdot B\cdot\overline{C}\cdot D) + (\overline{A}\cdot B\cdot C\cdot D) + \\ &\quad (A\cdot\overline{B}\cdot C\cdot\overline{D}) + (A\cdot\overline{B}\cdot C\cdot D) + (A\cdot B\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot D)\ . \end{aligned}$$

Analisando-se os mintermos, verifica-se que há as seguintes possibilidades de aglutinação: $(m_5 - m_7)$, $(m_{10} - m_{11})$, $(m_{14} - m_{15})$, $(m_{10} - m_{14})$ e $(m_{11} - m_{15})$. Observando-se que a aglutinação $(m_5 - m_7)$ é essencial, as seguintes aglutinações devem ser feitas para cobrir todos os valores booleanos "1" e para simplificar ao máximo a função: $\{(m_5 - m_7), (m_{10} - m_{11}), (m_{14} - m_{15})\}$ ou $\{(m_5 - m_7), (m_{10} - m_{14}), (m_{11} - m_{15})\}$.

Empregando-se as aglutinações $(m_5 - m_7)$, $(m_{10} - m_{11})$ e $(m_{14} - m_{15})$, a SOP mínima é dada por

$$\begin{aligned}
f(A,B,C,D) &= \sum m(0,5,7,10,11,14,15) \\
&= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \Big[(\overline{A}\cdot B\cdot\overline{C}\cdot D) + (\overline{A}\cdot B\cdot C\cdot D)\Big] + \\
&\quad \Big[(A\cdot\overline{B}\cdot C\cdot\overline{D}) + (A\cdot\overline{B}\cdot C\cdot D)\Big] + \Big[(A\cdot B\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot D)\Big] \\
&= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \Big[(\overline{A}\cdot B\cdot D)\cdot(\overline{C}+C)\Big] + \\
&\quad \Big[(A\cdot\overline{B}\cdot C)\cdot(\overline{D}+D)\Big] + \Big[(A\cdot B\cdot C)\cdot(\overline{D}+D)\Big] \\
&= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \Big[(\overline{A}\cdot B\cdot D)\cdot(1)\Big] + \\
&\quad \Big[(A\cdot\overline{B}\cdot C)\cdot(1)\Big] + [(A\cdot B\cdot C)\cdot(1)] \\
&= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot D) + \\
&\quad (A\cdot\overline{B}\cdot C) + (A\cdot B\cdot C) \\
&= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot D) + \\
&\quad \Big[(A\cdot C)\cdot(\overline{B}+B)\Big] \\
&= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \Big[(\overline{A}\cdot B\cdot D)\Big] + \\
&\quad [(A\cdot C)\cdot(1)] \\
&= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot D) + (A\cdot C)\ .
\end{aligned}$$

Nesse caso, ocorre a dupla aglutinação $[(m_{10} - m_{11}) - (m_{14} - m_{15})]$. Inicialmente, a variável D é dispensada. Em seguida, isso acontece com a variável B.

Empregando-se as aglutinações $(m_5 - m_7)$, $(m_{10} - m_{14})$ e $(m_{11} - m_{15})$, a SOP mínima é dada por

$$\begin{aligned}
f(A,B,C,D) &= \sum m(0,5,7,10,11,14,15) \\
&= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \Big[(\overline{A}\cdot B\cdot\overline{C}\cdot D) + (\overline{A}\cdot B\cdot C\cdot D)\Big] + \\
&\quad \Big[(A\cdot\overline{B}\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot\overline{D})\Big] + \Big[(A\cdot\overline{B}\cdot C\cdot D) + (A\cdot B\cdot C\cdot D)\Big] \\
&= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \Big[(\overline{A}\cdot B\cdot D)\cdot(\overline{C}+C)\Big] + \\
&\quad \Big[(A\cdot C\cdot\overline{D})\cdot(\overline{B}+B)\Big] + \Big[(A\cdot C\cdot D)\cdot(\overline{B}+B)\Big] \\
&= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \Big[(\overline{A}\cdot B\cdot D)\cdot(1)\Big] + \\
&\quad \Big[(A\cdot C\cdot\overline{D})\cdot(1)\Big] + [(A\cdot C\cdot D)\cdot(1)] \\
&= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot D) + \\
&\quad (A\cdot C\cdot\overline{D}) + (A\cdot C\cdot D) \\
&= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot D) + \\
&\quad \Big[(A\cdot C)\cdot(\overline{D}+D)\Big] \\
&= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot D) + \\
&\quad [(A\cdot C)\cdot(1)] \\
&= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot D) + (A\cdot C)\ .
\end{aligned}$$

Nesse caso, ocorre a dupla aglutinação $[(m_{10} - m_{14}) - (m_{11} - m_{15})]$. Inicialmente, a variável B é dispensada. Em seguida, isso acontece com a variável D.

Exemplo 5

Seja a função especificada por $f(A,B,C,D) = \sum m(0,4,5,10,11,14,15)$.
A equação literal da SOP padrão é dada por

$$\begin{aligned} f(A,B,C,D) &= \sum m(0,4,5,10,11,14,15) \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot\overline{C}\cdot D) + \\ &\quad (A\cdot\overline{B}\cdot C\cdot\overline{D}) + (A\cdot\overline{B}\cdot C\cdot D) + (A\cdot B\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot D)\ . \end{aligned}$$

Analisando-se os mintermos, verifica-se que há as seguintes possibilidades de aglutinação: $(m_0 - m_4)$, $(m_4 - m_5)$, $(m_{10} - m_{11})$, $(m_{14} - m_{15})$, $(m_{10} - m_{14})$ e $(m_{11} - m_{15})$. Observando-se que as aglutinações $(m_0 - m_4)$ e $(m_4 - m_5)$ são essenciais, as seguintes aglutinações devem ser feitas para cobrir todos os valores booleanos "1" e para simplificar ao máximo a função: $\{(m_0 - m_4), (m_4 - m_5), (m_{10} - m_{11}), (m_{14} - m_{15})\}$ ou $\{(m_0 - m_4), (m_4 - m_5), (m_{10} - m_{14}), (m_{11} - m_{15})\}$. Em qualquer das duas opções, deve-se duplicar o mintermo m_4.

Empregando-se as aglutinações $(m_0 - m_4)$, $(m_4 - m_5)$, $(m_{10} - m_{11})$ e $(m_{14} - m_{15})$, a SOP mínima é dada por

$$\begin{aligned} f(A,B,C,D) &= \sum m(0,4,5,10,11,14,15) \\ &= \Big[(\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot\overline{C}\cdot\overline{D})\Big] + \Big[(\overline{A}\cdot B\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot\overline{C}\cdot D)\Big] + \\ &\quad \Big[(A\cdot\overline{B}\cdot C\cdot\overline{D}) + (A\cdot\overline{B}\cdot C\cdot D)\Big] + \Big[(A\cdot B\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot D)\Big] \\ &= \Big[(\overline{A}\cdot\overline{C}\cdot\overline{D})\cdot(\overline{B}+B)\Big] + \Big[(\overline{A}\cdot B\cdot\overline{C})\cdot(\overline{D}+D)\Big] + \\ &\quad \Big[(A\cdot\overline{B}\cdot C)\cdot(\overline{D}+D)\Big] + \Big[(A\cdot B\cdot C)\cdot(\overline{D}+D)\Big] \\ &= \Big[(\overline{A}\cdot\overline{C}\cdot\overline{D})\cdot(1)\Big] + \Big[(\overline{A}\cdot B\cdot\overline{C})\cdot(1)\Big] + \\ &\quad \Big[(A\cdot\overline{B}\cdot C)\cdot(1)\Big] + [(A\cdot B\cdot C)\cdot(1)] \\ &= (\overline{A}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot\overline{C}) + \\ &\quad (A\cdot\overline{B}\cdot C) + (A\cdot B\cdot C) \\ &= (\overline{A}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot\overline{C}) + \\ &\quad \Big[(A\cdot C)\cdot(\overline{B}+B)\Big] \\ &= (\overline{A}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot\overline{C}) + \\ &\quad [(A\cdot C)\cdot(1)] \\ &= (\overline{A}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot\overline{C}) + (A\cdot C)\ . \end{aligned}$$

Nesse caso, ocorre a dupla aglutinação $[(m_{10} - m_{11}) - (m_{14} - m_{15})]$. Inicialmente, a variável D é dispensada. Em seguida, isso acontece com a variável B.

Empregando-se as aglutinações $(m_0 - m_4)$, $(m_4 - m_5)$, $(m_{10} - m_{14})$ e $(m_{11} - m_{15})$, a SOP mínima é dada por

$$\begin{aligned}
f(A,B,C,D) &= \sum m(0,4,5,10,11,14,15) \\
&= \Big[(\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot\overline{C}\cdot\overline{D})\Big] + \Big[(\overline{A}\cdot B\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot\overline{C}\cdot D)\Big] + \\
&\quad \Big[(A\cdot\overline{B}\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot\overline{D})\Big] + \Big[(A\cdot\overline{B}\cdot C\cdot D) + (A\cdot B\cdot C\cdot D)\Big] \\
&= \Big[(\overline{A}\cdot\overline{C}\cdot\overline{D})\cdot(\overline{B}+B)\Big] + \Big[(\overline{A}\cdot B\cdot\overline{C})\cdot(\overline{D}+D)\Big] + \\
&\quad \Big[(A\cdot C\cdot\overline{D})\cdot(\overline{B}+B)\Big] + \Big[(A\cdot C\cdot D)\cdot(\overline{B}+B)\Big] \\
&= \Big[(\overline{A}\cdot\overline{C}\cdot\overline{D})\cdot(1)\Big] + \Big[(\overline{A}\cdot B\cdot\overline{C})\cdot(1)\Big] + \\
&\quad \Big[(A\cdot C\cdot\overline{D})\cdot(1)\Big] + [(A\cdot C\cdot D)\cdot(1)] \\
&= (\overline{A}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot\overline{C}) + \\
&\quad (A\cdot C\cdot\overline{D}) + (A\cdot C\cdot D) \\
&= (\overline{A}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot\overline{C}) + \\
&\quad \Big[(A\cdot C)\cdot(\overline{D}+D)\Big] \\
&= (\overline{A}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot\overline{C}) + \\
&\quad [(A\cdot C)\cdot(1)] \\
&= (\overline{A}\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot\overline{C}) + (A\cdot C)\ .
\end{aligned}$$

Nesse caso, ocorre a dupla aglutinação $[(m_{10} - m_{14}) - (m_{11} - m_{15})]$. Inicialmente, a variável B é dispensada. Em seguida, isso acontece com a variável D.

Exemplo 6

Seja a função especificada por $f(A,B,C,D) = \sum m(4,5,8,10,11,14,15)$.
A equação literal da SOP padrão é dada por

$$\begin{aligned}
f(A,B,C,D) &= \sum m(4,5,8,10,11,14,15) \\
&= (\overline{A}\cdot B\cdot\overline{C}\cdot\overline{D}) + (\overline{A}\cdot B\cdot\overline{C}\cdot D) + (A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}) + \\
&\quad (A\cdot\overline{B}\cdot C\cdot\overline{D}) + (A\cdot\overline{B}\cdot C\cdot D) + (A\cdot B\cdot C\cdot\overline{D}) + (A\cdot B\cdot C\cdot D)\ .
\end{aligned}$$

Analisando-se os mintermos, verifica-se que há as seguintes possibilidades de aglutinação: $(m_4 - m_5)$, $(m_8 - m_{10})$, $(m_{10} - m_{11})$, $(m_{14} - m_{15})$, $(m_{10} - m_{14})$ e $(m_{11} - m_{15})$. Observando-se que as aglutinações $(m_4 - m_5)$ e $(m_8 - m_{10})$ são essenciais, as seguintes aglutinações devem ser feitas para cobrir todos os valores booleanos "1" e para simplificar ao máximo a função: $\{(m_4 - m_5), (m_8 - m_{10}), (m_{10} - m_{11}), (m_{14} - m_{15})\}$ ou $\{(m_4 - m_5), (m_8 - m_{10}), (m_{10} - m_{14}), (m_{11} - m_{15})\}$. Em qualquer das duas opções, deve-se duplicar o mintermo m_{10}.

Empregando-se as aglutinações $(m_4 - m_5)$, $(m_8 - m_{10})$, $(m_{10} - m_{11})$ e $(m_{14} - m_{15})$, a SOP mínima é dada por

$$\begin{aligned}
f(A,B,C,D) &= \sum m(4,5,8,10,11,14,15) \\
&= \Big[(\overline{A}\cdot B\cdot \overline{C}\cdot \overline{D}) + (\overline{A}\cdot B\cdot \overline{C}\cdot D)\Big] + \Big[(A\cdot \overline{B}\cdot \overline{C}\cdot \overline{D}) + (A\cdot \overline{B}\cdot C\cdot \overline{D})\Big] + \\
&\quad \Big[(A\cdot \overline{B}\cdot C\cdot \overline{D}) + (A\cdot \overline{B}\cdot C\cdot D)\Big] + \Big[(A\cdot B\cdot C\cdot \overline{D}) + (A\cdot B\cdot C\cdot D)\Big] \\
&= \Big[(\overline{A}\cdot B\cdot \overline{C})\cdot (\overline{D}+D)\Big] + \Big[(A\cdot \overline{B}\cdot \overline{D})\cdot (\overline{C}+C)\Big] + \\
&\quad \Big[(A\cdot \overline{B}\cdot C)\cdot (\overline{D}+D)\Big] + \Big[(A\cdot B\cdot C)\cdot (\overline{D}+D)\Big] \\
&= \Big[(\overline{A}\cdot B\cdot \overline{C})\cdot (1)\Big] + \Big[(A\cdot \overline{B}\cdot \overline{D})\cdot (1)\Big] + \\
&\quad \Big[(A\cdot \overline{B}\cdot C)\cdot (1)\Big] + [(A\cdot B\cdot C)\cdot (1)] \\
&= (\overline{A}\cdot B\cdot \overline{C}) + (A\cdot \overline{B}\cdot \overline{D}) + \\
&\quad (A\cdot \overline{B}\cdot C) + (A\cdot B\cdot C) \\
&= (\overline{A}\cdot B\cdot \overline{C}) + (A\cdot \overline{B}\cdot \overline{D}) + \\
&\quad \Big[(A\cdot C)\cdot (\overline{B}+B)\Big] \\
&= (\overline{A}\cdot B\cdot \overline{C}) + (A\cdot \overline{B}\cdot \overline{D}) + \\
&\quad [(A\cdot C)\cdot (1)] \\
&= (\overline{A}\cdot B\cdot \overline{C}) + (A\cdot \overline{B}\cdot \overline{D}) + (A\cdot C)\ .
\end{aligned}$$

Nesse caso, ocorre a dupla aglutinação $[(m_{10} - m_{11}) - (m_{14} - m_{15})]$. Inicialmente, a variável D é dispensada. Em seguida, isso acontece com a variável B.

Empregando-se as aglutinações $(m_4 - m_5)$, $(m_8 - m_{10})$, $(m_{10} - m_{14})$ e $(m_{11} - m_{15})$, a SOP mínima é dada por

$$\begin{aligned}
f(A,B,C,D) &= \sum m(4,5,8,10,11,14,15) \\
&= \Big[(\overline{A}\cdot B\cdot \overline{C}\cdot \overline{D}) + (\overline{A}\cdot B\cdot \overline{C}\cdot D)\Big] + \Big[(A\cdot \overline{B}\cdot \overline{C}\cdot \overline{D}) + (A\cdot \overline{B}\cdot C\cdot \overline{D})\Big] + \\
&\quad \Big[(A\cdot \overline{B}\cdot C\cdot \overline{D}) + (A\cdot B\cdot C\cdot \overline{D})\Big] + \Big[(A\cdot \overline{B}\cdot C\cdot D) + (A\cdot B\cdot C\cdot D)\Big] \\
&= \Big[(\overline{A}\cdot B\cdot \overline{C})\cdot (\overline{D}+D)\Big] + \Big[(A\cdot \overline{B}\cdot \overline{D})\cdot (\overline{C}+C)\Big] + \\
&\quad \Big[(A\cdot C\cdot \overline{D})\cdot (\overline{B}+B)\Big] + \Big[(A\cdot C\cdot D)\cdot (\overline{B}+B)\Big] \\
&= \Big[(\overline{A}\cdot B\cdot \overline{C})\cdot (1)\Big] + \Big[(A\cdot \overline{B}\cdot \overline{D})\cdot (1)\Big] + \\
&\quad \Big[(A\cdot C\cdot \overline{D})\cdot (1)\Big] + [(A\cdot C\cdot D)\cdot (1)] \\
&= (\overline{A}\cdot B\cdot \overline{C}) + (A\cdot \overline{B}\cdot \overline{D}) + \\
&\quad (A\cdot C\cdot \overline{D}) + (A\cdot C\cdot D) \\
&= (\overline{A}\cdot B\cdot \overline{C}) + (A\cdot \overline{B}\cdot \overline{D}) + \\
&\quad \Big[(A\cdot C)\cdot (\overline{D}+D)\Big] \\
&= (\overline{A}\cdot B\cdot \overline{C}) + (A\cdot \overline{B}\cdot \overline{D}) + \\
&\quad [(A\cdot C)\cdot (1)] \\
&= (\overline{A}\cdot B\cdot \overline{C}) + (A\cdot \overline{B}\cdot \overline{D}) + (A\cdot C)\ .
\end{aligned}$$

Nesse caso, ocorre a dupla aglutinação $[(m_{10} - m_{14}) - (m_{11} - m_{15})]$. Inicialmente, a variável B é dispensada. Em seguida, isso acontece com a variável D.

Exemplo 7

Seja a função especificada por $f(A,B,C,D) = \sum m(6,8,10,11,13,14,15)$.
A equação literal da SOP padrão é dada por

$$\begin{aligned} f(A,B,C,D) &= \sum m(6,8,10,11,13,14,15) \\ &= (\overline{A}\cdot B\cdot C\cdot \overline{D}) + (A\cdot \overline{B}\cdot \overline{C}\cdot \overline{D}) + (A\cdot \overline{B}\cdot C\cdot \overline{D}) + \\ &\quad (A\cdot \overline{B}\cdot C\cdot D) + (A\cdot B\cdot \overline{C}\cdot D) + (A\cdot B\cdot C\cdot \overline{D}) + (A\cdot B\cdot C\cdot D)\ . \end{aligned}$$

Analisando-se os mintermos, verifica-se que há as seguintes possibilidades de aglutinação: $(m_6 - m_{14})$, $(m_8 - m_{10})$, $(m_{13} - m_{15})$, $(m_{10} - m_{11})$, $(m_{14} - m_{15})$, $(m_{10} - m_{14})$ e $(m_{11} - m_{15})$. Observando-se que as aglutinações $(m_6 - m_{14})$, $(m_8 - m_{10})$, $(m_{13} - m_{15})$ são essenciais, as seguintes aglutinações devem ser feitas para cobrir todos os valores booleanos "1" e para simplificar ao máximo a função: $\{(m_6 - m_{14}), (m_8 - m_{10}), (m_{13} - m_{15}), (m_{10} - m_{11}), (m_{14} - m_{15})\}$ ou $\{(m_6 - m_{14}), (m_8 - m_{10}), (m_{13} - m_{15}), (m_{10} - m_{14}), (m_{11} - m_{15})\}$. Em qualquer das duas opções, devem-se duplicar os mintermos m_{10}, m_{14} e m_{15}.

Empregando-se as aglutinações $(m_6 - m_{14})$, $(m_8 - m_{10})$, $(m_{13} - m_{15})$, $(m_{10} - m_{11})$, $(m_{14} - m_{15})$, a SOP mínima é dada por

$$\begin{aligned} f(A,B,C,D) &= \sum m(6,8,10,11,13,14,15) \\ &= \Big[(\overline{A}\cdot B\cdot C\cdot \overline{D}) + (A\cdot B\cdot C\cdot \overline{D})\Big] + \\ &\quad \Big[(A\cdot \overline{B}\cdot \overline{C}\cdot \overline{D}) + (A\cdot \overline{B}\cdot C\cdot \overline{D})\Big] + \Big[(A\cdot B\cdot \overline{C}\cdot D) + (A\cdot B\cdot C\cdot D)\Big] + \\ &\quad \Big[(A\cdot \overline{B}\cdot C\cdot \overline{D}) + (A\cdot \overline{B}\cdot C\cdot D)\Big] + \Big[(A\cdot B\cdot C\cdot \overline{D}) + (A\cdot B\cdot C\cdot D)\Big] \\ &= \Big[(B\cdot C\cdot \overline{D})\cdot (\overline{A} + A)\Big] + \\ &\quad \Big[(A\cdot \overline{B}\cdot \overline{D})\cdot (\overline{C} + C)\Big] + \Big[(A\cdot B\cdot D)\cdot (\overline{C} + C)\Big] + \\ &\quad \Big[(A\cdot \overline{B}\cdot C)\cdot (\overline{D} + D)\Big] + \Big[(A\cdot B\cdot C)\cdot (\overline{D} + D)\Big] \\ &= \Big[(B\cdot C\cdot \overline{D})\cdot (1)\Big] + \Big[(A\cdot \overline{B}\cdot \overline{D})\cdot (1)\Big] + [(A\cdot B\cdot D)\cdot (1)] + \\ &\quad \Big[(A\cdot \overline{B}\cdot C)\cdot (1)\Big] + [(A\cdot B\cdot C)\cdot (1)] \\ &= (B\cdot C\cdot \overline{D}) + (A\cdot \overline{B}\cdot \overline{D}) + (A\cdot B\cdot D) + \\ &\quad (A\cdot \overline{B}\cdot C) + (A\cdot B\cdot C) \\ &= (B\cdot C\cdot \overline{D}) + (A\cdot \overline{B}\cdot \overline{D}) + (A\cdot B\cdot D) + \\ &\quad \Big[(A\cdot C)\cdot (\overline{B} + B)\Big] \\ &= (B\cdot C\cdot \overline{D}) + (A\cdot \overline{B}\cdot \overline{D}) + (A\cdot B\cdot D) + \\ &\quad [(A\cdot C)\cdot (1)] \\ &= (B\cdot C\cdot \overline{D}) + (A\cdot \overline{B}\cdot \overline{D}) + (A\cdot B\cdot D) + (A\cdot C)\ . \end{aligned}$$

Nesse caso, ocorre a dupla aglutinação $[(m_{10} - m_{11}) - (m_{14} - m_{15})]$. Inicialmente, a variável D é dispensada. Em seguida, isso acontece com a variável B.

Empregando-se as aglutinações $(m_6 - m_{14})$, $(m_8 - m_{10})$, $(m_{13} - m_{15})$, $(m_{10} - m_{14})$ e $(m_{11} - m_{15})$, a SOP mínima é dada por

$$\begin{aligned}
f(A,B,C,D) &= \sum m(6,8,10,11,13,14,15) \\
&= \Big[(\overline{A}\cdot B\cdot C\cdot \overline{D}) + (A\cdot B\cdot C\cdot \overline{D})\Big] + \\
&\quad \Big[(A\cdot \overline{B}\cdot \overline{C}\cdot \overline{D}) + (A\cdot \overline{B}\cdot C\cdot \overline{D})\Big] + \Big[(A\cdot B\cdot \overline{C}\cdot D) + (A\cdot B\cdot C\cdot D)\Big] + \\
&\quad \Big[(A\cdot \overline{B}\cdot C\cdot \overline{D}) + (A\cdot B\cdot C\cdot \overline{D})\Big] + \Big[(A\cdot \overline{B}\cdot C\cdot D) + (A\cdot B\cdot C\cdot D)\Big] \\
&= \Big[(B\cdot C\cdot \overline{D})\cdot (\overline{A} + A)\Big] + \\
&\quad \Big[(A\cdot \overline{B}\cdot \overline{D})\cdot (\overline{C} + C)\Big] + \Big[(A\cdot B\cdot D)\cdot (\overline{C} + C)\Big] + \\
&\quad \Big[(A\cdot C\cdot \overline{D})\cdot (\overline{B} + B)\Big] + \Big[(A\cdot C\cdot D)\cdot (\overline{B} + B)\Big] \\
&= \Big[(B\cdot C\cdot \overline{D})\cdot (1)\Big] + \Big[(A\cdot \overline{B}\cdot \overline{D})\cdot (1)\Big] + [(A\cdot B\cdot D)\cdot (1)] + \\
&\quad \Big[(A\cdot C\cdot \overline{D})\cdot (1)\Big] + [(A\cdot C\cdot D)\cdot (1)] \\
&= (B\cdot C\cdot \overline{D}) + (A\cdot \overline{B}\cdot \overline{D}) + (A\cdot B\cdot D) + \\
&\quad (A\cdot C\cdot \overline{D}) + (A\cdot C\cdot D) \\
&= (B\cdot C\cdot \overline{D}) + (A\cdot \overline{B}\cdot \overline{D}) + (A\cdot B\cdot D) + \\
&\quad \Big[(A\cdot C)\cdot (\overline{D} + D)\Big] \\
&= (B\cdot C\cdot \overline{D}) + (A\cdot \overline{B}\cdot \overline{D}) + (A\cdot B\cdot D) + \\
&\quad [(A\cdot C)\cdot (1)] \\
&= (B\cdot C\cdot \overline{D}) + (A\cdot \overline{B}\cdot \overline{D}) + (A\cdot B\cdot D) + (A\cdot C)\ .
\end{aligned}$$

Nesse caso, ocorre a dupla aglutinação $[(m_{10} - m_{14}) - (m_{11} - m_{15})]$. Inicialmente, a variável B é dispensada. Em seguida, isso acontece com a variável D.

7.13 Exercícios propostos

1. Obtenha, algebricamente, as formas SOP mínima e POS mínima para as seguintes equações booleanas:

   (a) $F(A,B,C) = (\overline{A}\cdot\overline{B}\cdot\overline{C}) + (\overline{A}\cdot B\cdot\overline{C}) + (A\cdot\overline{B}\cdot C) + (A\cdot B\cdot C)$.
   (b) $F(A,B,C) = (\overline{A}\cdot\overline{B}\cdot C) + (\overline{A}\cdot B\cdot C) + (A\cdot\overline{B}\cdot\overline{C}) + (A\cdot B\cdot\overline{C})$.
   (c) $F(A,B,C) = (\overline{A}\cdot\overline{B}\cdot\overline{C}) + (\overline{A}\cdot\overline{B}\cdot C) + (A\cdot B\cdot\overline{C}) + (A\cdot B\cdot C)$.
   (d) $F(A,B,C) = (\overline{A}\cdot B\cdot\overline{C}) + (\overline{A}\cdot B\cdot C) + (A\cdot\overline{B}\cdot\overline{C}) + (A\cdot\overline{B}\cdot C)$.
   (e) $F(A,B,C) = (\overline{A}\cdot\overline{B}\cdot\overline{C}) + (\overline{A}\cdot B\cdot C) + (A\cdot\overline{B}\cdot\overline{C}) + (A\cdot B\cdot C)$.
   (f) $F(A,B,C) = (\overline{A}\cdot\overline{B}\cdot C) + (\overline{A}\cdot B\cdot\overline{C}) + (A\cdot\overline{B}\cdot C) + (A\cdot B\cdot\overline{C})$.

2. Obtenha, algebricamente, as formas SOP mínima e POS mínima para as seguintes equações booleanas:

   (a) $F(A,B,C) = (A+B+\overline{C})\cdot(A+\overline{B}+\overline{C})\cdot(\overline{A}+B+C)\cdot(\overline{A}+\overline{B}+C)$.
   (b) $F(A,B,C) = (A+B+C)\cdot(A+\overline{B}+C)\cdot(\overline{A}+B+\overline{C})\cdot(\overline{A}+\overline{B}+\overline{C})$.
   (c) $F(A,B,C) = (A+\overline{B}+C)\cdot(A+\overline{B}+\overline{C})\cdot(\overline{A}+B+C)\cdot(\overline{A}+B+\overline{C})$.
   (d) $F(A,B,C) = (A+B+C)\cdot(A+B+\overline{C})\cdot(\overline{A}+\overline{B}+C)\cdot(\overline{A}+\overline{B}+\overline{C})$.
   (e) $F(A,B,C) = (A+B+\overline{C})\cdot(A+\overline{B}+C)\cdot(\overline{A}+B+\overline{C})\cdot(\overline{A}+\overline{B}+C)$.
   (f) $F(A,B,C) = (A+B+C)\cdot(A+\overline{B}+\overline{C})\cdot(\overline{A}+B+C)\cdot(\overline{A}+\overline{B}+\overline{C})$.

3. Para os exercícios listados abaixo, considerar as equações booleanas apresentadas em seguida.

   (a) Algebricamente, obter a forma SOP padrão da equação fornecida.
   (b) Algebricamente, obter a forma SOP mínima, a partir da SOP padrão.
   (c) Algebricamente, obter a forma POS padrão da equação fornecida.
   (d) Algebricamente, obter a forma POS mínima, a partir da POS padrão.
   (e) Apresentar a expressão mínima para função.

   Equações boolenas:

   i. $F(A,B,C) = \Big\{B\cdot\Big[\Big(\overline{A}\cdot\overline{C}\Big) + (A\cdot C)\Big]\Big\} + \overline{\Big\{B + \Big[\Big(\overline{A}+\overline{C}\Big)\cdot(A+C)\Big]\Big\}}$.
   ii. $F(A,B,C) = \overline{\Big\{\Big[\overline{(A+B)}+C\Big]\cdot\Big[\overline{A}+\overline{\Big(B+\overline{C}\Big)}\Big]\Big\}}$.
   iii. $F(A,B,C) = \Big[\overline{\Big(A\cdot\overline{B}\Big)}+C\Big]\cdot\overline{[(A+C)\cdot B]}$.

Capítulo 8

Mapa de Karnaugh

8.1 Introdução

O mapa de Karnaugh (mapa-K) é mais uma das possíveis expressões de uma função lógica, além das seguintes: uma equação lógica genérica, uma equação booleana genérica, uma forma do grupo SOP, uma forma do grupo POS, uma forma padrão do grupo SOP, uma forma padrão do grupo POS, uma lista de mintermos, uma lista de maxtermos e uma tabela verdade.

Além de representar uma simples expressão para uma função lógica, o mapa-K pode ser usado como ferramenta prática para a minimização da equação que a define.

O mapa-K pode ser interpretado como uma tabela verdade rearranjada ou como uma representação análoga ao Diagrama de Venn, da Teoria de conjuntos. Para cada linha da tabela verdade de uma equação booleana é associada uma posição no mapa-K. Cada linha da tabela verdade é associada a um mintermo ou a um maxtermo. Logo, a cada um desses termos também é associada uma posição do mapa.

A fim de que o mapa-K seja empregado como uma ferramenta prática, de operação simples e rápida, no processo de simplificação de equações booleanas, ele é arranjado da seguinte forma:

- De forma similar à uma tabela verdade, deve existir uma localização única no mapa para cada combinação das variáveis das quais a função lógica é dependente. Isso é equivalente a dizer que deve existir uma localização única no mapa para cada mintermo e para cada maxtermo da função lógica.

- As localizações devem ser arranjadas de tal forma que todos os grupos de termos que apresentam potencial para serem combinados em formas reduzidas devam ser facilmente encontrados por uma simples inspeção visual.

Devido a uma limitação prática, são construídos mapas-K para funções lógicas de até 6 variáveis. Para funções lógicas com um número superior a 6 variáveis, pode-se utilizar um algoritmo de minimização, tal como o algoritmo tabular de Quine-McCluskey.

8.2 Motivação para o desenvolvimento do mapa-K

8.2.1 Complexidade da busca dos termos na simplificação de funções

Para funções booleanas dependentes de poucas variáveis e equações booleanas com poucos termos, o emprego sistemático dos Postulados, Lemas e Teoremas da Álgebra de Boole, organizados na forma das operações de aglutinação e de replicação, não é uma tarefa muito complexa. Porém, com o aumento do número de variáveis e do número de termos, cresce também a complexidade do processo de simplificação de uma forma SOP/POS padrão em uma forma SOP/POS mínima. O aumento da complexidade reside no fato de que a procura dos termos que podem ser combinados e dos que precisam ser replicados torna-se mais custosa à medida que crescem o número de termos da expressão e a quantidade de literais em cada termo. Assim sendo, torna-se necessário tentar encontrar uma técnica que reduza a complexidade da busca dos termos que podem ser combinados.

8.2.2 Simplificação de funções por meio da tabela verdade

A tabela verdade pode ser utilizada para a simplificação de equações booleanas, caso seja empregado sobre ela o mesmo método sistemático que é aplicado às equações.

Nas equações, a aglutinação de dois termos mais complexos em um único termo mais simples, por meio da eliminação de uma variável, é conseguida quando todos as variáveis de um termo são iguais às do outro termo, exceto uma delas, que aparece na sua forma original em um termo e na sua forma barrada (negada) no outro termo. A variável que se modifica é a eliminada.

Na tabela verdade, o raciocínio equivalente acontece quando as variáveis e as suas formas barradas são trocadas pelos valores booleanos “0” e “1”. Nesse caso, a aglutinação é possível quando todos as variáveis de ambos os termos assumem o valor 0 (ou 1), exceto uma delas, que aparece como 0 em um termo e como 1 no outro termo. A variável que se modifica é a eliminada.

Como exemplo do uso da tabela verdade, sobre uma função de três variáveis $f(A, B, C)$, observa-se que tanto os mintermos $m_0 = (\overline{A} \cdot \overline{B} \cdot \overline{C})$ e $m_1 = (\overline{A} \cdot \overline{B} \cdot C)$ quanto os maxtermos $M_0 = (A + B + C)$ e $M_1 = (A + B + \overline{C})$ podem ser combinados, gerando $(\overline{A} \cdot \overline{B})$ e $(A + B)$, respectivamente, por meio da eliminação da variável “C”. Na tabela verdade, o processo equivalente é o seguinte: comparar os padrões “000” e “001”, eliminar a variável “C” e montar o termo final simplificado com as variáveis “A” e “B”.

8.2.3 Redução da complexidade da busca: primeira tentativa

Em um primeira tentativa de redução da complexidade da busca de termos, na simplificação de funções booleanas, pode-se pensar em utilizar a tabela verdade, dado que ela é uma representação visualmente mais simples e bem organizada. Por exemplo, a tabela verdade da equação booleana

$$\begin{aligned} F(A,B,C) &= \sum m(1,2,6,7) &&= (\overline{A} \cdot \overline{B} \cdot C) + (\overline{A} \cdot B \cdot \overline{C}) + (A \cdot B \cdot \overline{C}) + (A \cdot B \cdot C) \\ &= \prod M(0,3,4,5) &&= (A+B+C) \cdot (A+\overline{B}+\overline{C}) \cdot (\overline{A}+B+C) \cdot (\overline{A}+B+\overline{C}) \ . \end{aligned} \tag{8.1}$$

é definida na Tabela 8.1, onde são indicados os mintermos (m_k) e os maxtermos (M_k).

Linha	A	B	C	$F(A,B,C)$	m_k/M_k
0	0	0	0	0	M_0
1	0	0	1	1	m_1
2	0	1	0	1	m_2
3	0	1	1	0	M_3
4	1	0	0	0	M_4
5	1	0	1	0	M_5
6	1	1	0	1	m_6
7	1	1	1	1	m_7

Tabela 8.1: Tabela verdade da Equação (8.1).

A SOP mínima da Equação (8.1) pode ser obtida replicando-se o mintermo m_6 e aglutinando cada cópia com os mintermos m_2 e m_7, o que fornece

$$F(A,B,C) = (\overline{A} \cdot \overline{B} \cdot C) + (B \cdot \overline{C}) + (A \cdot B) \ . \tag{8.2}$$

Da Tabela 8.1, deve-se observar, para os mintermos, que:

- Os mintermos m_6 e m_7, que são próximos, combinam-se.
- Os mintermos m_6 e m_2, que são distantes, combinam-se.
- Os mintermos m_1 e m_2, que são próximos, não se combinam.
- Os mintermos m_1 e m_7, que são distantes, não se combinam.

Por sua vez, a POS mínima da Equação (8.1) pode ser obtida replicando-se o maxtermo M_4 e aglutinando cada cópia com os maxtermos M_0 e M_5, o que fornece

$$F(A,B,C) = (A + \overline{B} + \overline{C}) \cdot (B + C) \cdot (\overline{A} + B) \ . \tag{8.3}$$

Da Tabela 8.1, deve-se observar, para os maxtermos, que:

- Os maxtermos M_4 e M_5, que são próximos, combinam-se.
- Os maxtermos M_4 e M_0, que são distantes, combinam-se.
- Os maxtermos M_3 e M_4, que são próximos, não se combinam.
- Os maxtermos M_3 e M_0, que são distantes, não se combinam.

Fornecendo uma visão mais geral do processo, as Tabelas 8.2 e 8.3 ilustram as relações de possibilidade de simplificação entre termos (mintermos/maxtermos), respectivamente para funções de duas e três variáveis. Nessas tabelas, observa-se que, para cada termo em questão (*), alguns dos termos que podem se aglutinar (+) encontram-se em uma linha vizinha, enquanto outros termos que também podem se aglutinar (+) encontram-se em linhas diversas e não vizinhas.

Linha	A	B	T_k (m_k/M_k)			
			0	1	2	3
0	0	0	*	+	+	
1	0	1	+	*		+
2	1	0	+		*	+
3	1	1		+	+	*

Tabela 8.2: Relações de possibilidade de simplificação entre termos: funções de duas variáveis.

Linha	A	B	C	T_k (m_k/M_k)							
				0	1	2	3	4	5	6	7
0	0	0	0	*	+	+		+			
1	0	0	1	+	*		+		+		
2	0	1	0	+		*	+			+	
3	0	1	1		+	+	*				+
4	1	0	0	+				*	+	+	
5	1	0	1		+			+	*		+
6	1	1	0			+		+		*	+
7	1	1	1				+		+	+	*

Tabela 8.3: Relações de possibilidade de simplificação entre termos: funções de três variáveis.

Portanto, não é difícil concluir que, na sua forma original, apresentada nas Tabelas 8.1, 8.2 e 8.3, a tabela verdade não é uma ferramenta extremamente facilitadora para o processo de minimização. Na sua forma original, ela apresenta uma dificuldade similar àquela encontrada nas equações, na busca por combinações de termos.

8.2.4 Modificação na organização da tabela verdade

Uma modificação que pode ser executada sobre a tabela verdade, no sentido de torná-la uma ferramenta mais útil ao processo de otimização, é a reorganização das suas linhas.

A organização original da tabela verdade baseia-se na alteração do período P_k, relativo à troca de valores das variáveis. Assim, em funções de duas variáveis $f(A, B)$, tem-se $P_B = 1$ para B e $P_A = 2$ para A, como na Tabela 8.2. Por sua vez, em funções de três variáveis $f(A, B, C)$, tem-se $P_C = 1$ para C, $P_B = 2$ para B e $P_A = 4$ para A, como na Tabela 8.3.

Porém, na busca por termos combináveis, é mais importante que a organização da tabela verdade fundamente-se em uma alocação que estabeleça uma relação direta entre os padrões que se diferenciam em apenas uma das variáveis.

O denominado Código Gray apresenta o tipo de organização desejada. As Tabelas 8.4, 8.5 e 8.6 apresentam o Código Gray para funções de 1, 2 e 3 variáveis, respectivamente. O processo de geração do Código Gray pode ser interpretado como a realização de espelhamentos entre blocos de padrões, produzidos a partir de eixos de simetria, conforme indicado nas tabelas em questão. Nessas tabelas, podem-se observar espelhamentos entre os seguintes blocos: 0 e 1; 3 e 2; 6 e 7; 5 e 4; 0-1 e 3-2; 6-7 e 5-5; 0-1-3-2 e 6-7-5-4.

Os espelhamentos do Código Gray produzem dois efeitos. No primeiro deles, a cada duas linhas geometricamente adjacentes, apenas uma das variáveis sofre alteração no seu valor. Isso também acontece entre a primeira e a última linha, criando uma ligação geométrica circular entre elas e, assim, tornando-as geometricamente adjacentes. No segundo efeito, a alteração no valor de apenas uma das variáveis também acontece em padrões que se superpõem nos espelhamentos. Por exemplo, o padrão da linha 0 sofre espelhamento sobre os padrões das linhas 1, 2 e 4, alterando apenas as variáveis C, B e A, respectivamente.

Linha	*A*
0	0
1	1

Tabela 8.4: Código Gray para 1 variável.

Linha	*A*	*B*
0	0	0
1	0	1
3	1	1
2	1	0

Tabela 8.5: Código Gray para 2 variáveis.

Linha	*A*	*B*	*C*
0	0	0	0
1	0	0	1
3	0	1	1
2	0	1	0
6	1	1	0
7	1	1	1
5	1	0	1
4	1	0	0

Tabela 8.6: Código Gray para 3 variáveis.

Seguindo a modificação indicada pelo Código Gray, a tabela verdade da Equação (8.1), apresentada pela Tabela 8.1, é reorganizada e reapresentada na Tabela 8.7. Nessa tabela modificada, para o exemplo em questão, torna-se mais fácil verificar que o mintermo m_1 fica isolado, enquanto o mintermo m_6 combina-se com os mintermos m_2 e m_7. Da mesma forma, observa-se mais facilmente que o maxtermo M_3 fica isolado, enquanto o maxtermo M_4 combina-se com os maxtermos M_5 e M_0.

Fornecendo uma visão mais geral do processo, as Tabelas 8.8 e 8.9 ilustram a reorganização das Tabelas 8.2 e 8.3. Nessas tabelas modificadas, também torna-se mais fácil verificar, para cada termo em questão (*), com quais termos ele pode se aglutinar (+).

Linha	A	B	C	$F(A,B,C)$	m_k/M_k
0	0	0	0	0	M_0
1	0	0	1	1	m_1
3	0	1	1	0	M_3
2	0	1	0	1	m_2
6	1	1	0	1	m_6
7	1	1	1	1	m_7
5	1	0	1	0	M_5
4	1	0	0	0	M_4

Tabela 8.7: Tabela verdade da Equação (8.1), reorganizada.

Linha	A	B	T_k (m_k/M_k)			
			0	1	2	3
0	0	0	*	+	+	
1	0	1	+	*		+
3	1	1		+	+	*
2	1	0	+		*	+

Tabela 8.8: Relações de possibilidade de simplificação entre termos: funções de duas variáveis. Tabela verdade reorganizada.

Linha	A	B	C	T_k (m_k/M_k)							
				0	1	2	3	4	5	6	7
0	0	0	0	*	+	+		+			
1	0	0	1	+	*		+		+		
3	0	1	1		+	+	*				+
2	0	1	0	+		*	+			+	
6	1	1	0			+		+		*	+
7	1	1	1				+		+	+	*
5	1	0	1		+			+	*		+
4	1	0	0	+				*	+	+	

Tabela 8.9: Relações de possibilidade de simplificação entre termos: funções de três variáveis. Tabela verdade reorganizada.

A reorganização da estrutura da tabela verdade, empregando o Código Gray e definindo uma sistemática para a relação entre os termos, proporciona uma melhoria no processo de busca dos termos combináveis. Mesmo assim, ainda é necessário que os termos combináveis sejam procurados ao longo de toda a tabela. Logo, um esforço extra é necessário, a fim de que uma ferramenta mais eficaz seja proposta.

8.2.5 Motivação final para a criação do mapa de Karnaugh

A tentativa inicial de utilizar a tabela verdade a fim de facilitar a busca de termos para a simplificação das expressões booleanas não produz um bom resultado. Em seguida, a tentativa de modificação na organização da tabela verdade produz um resultado melhor, mas ainda requer um aperfeiçoamento. Finalmente, o resultado procurado aparece na forma do mapa de Karnaugh, que apresenta uma alocação geometricamente adjacente completa para todos os termos potencialmente combináveis.

8.3 Construção do mapa-K

A seguir, são apresentados alguns exemplos de construção para um mapa-K, relativos a funções de uma até quatro variáveis. Como deverá ser percebido, o mapa-K é criado a partir de um rearranjo da tabela verdade. O rearranjo é baseado em espelhamentos relativos aos valores das variáveis, gerando uma estrutura que lembra um Diagrama de Venn. Pode-se dizer que a estrutura vetorial da tabela verdade é transformada na estrutura matricial do mapa-K. Em função dos espelhamentos realizados, a indexação das linhas e das colunas do mapa-K assume os valores do Código Gray, incorporando seus efeitos.

8.3.1 Funções de 1 variável

$Linha$	A	$F(A)$
0	0	F_0
1	1	F_1

Tabela 8.10: Tabela verdade para funções de 1 variável.

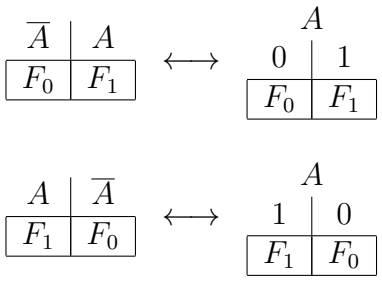


Figura 8.1: Exemplos de mapas de Karnaugh para funções de 1 variável.

8.3.2 Funções de 2 variáveis

Linha	A	B	$F(A,B)$
0	0	0	F_0
1	0	1	F_1
2	1	0	F_2
3	1	1	F_3

Tabela 8.11: Tabela verdade para funções de 2 variáveis.

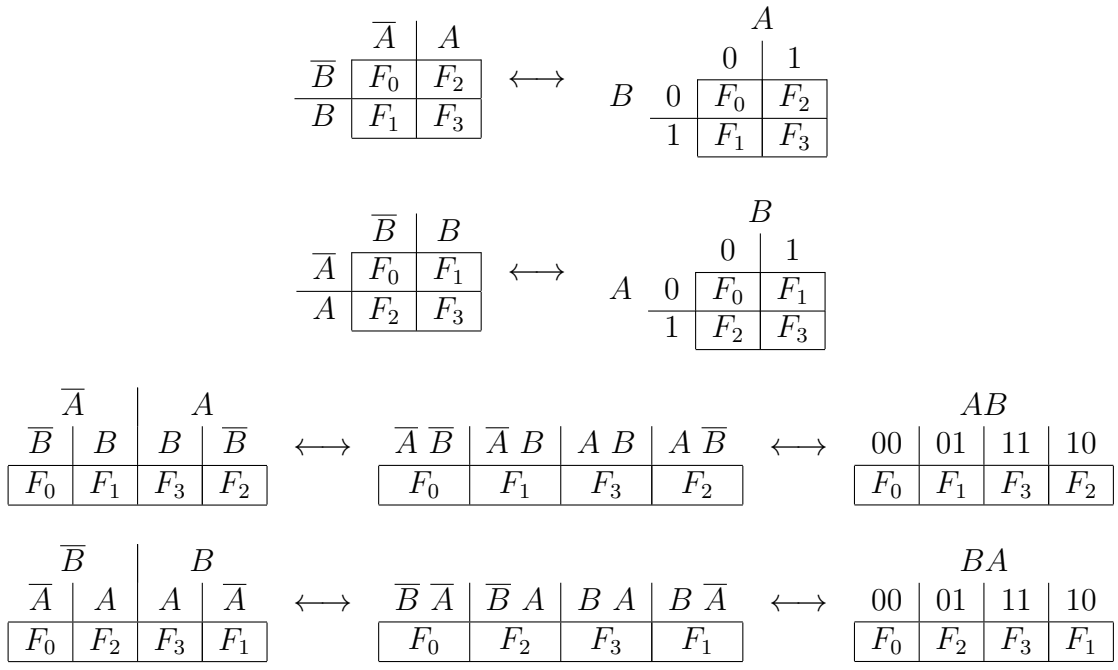


Figura 8.2: Exemplos de mapas de Karnaugh para funções de 2 variáveis.

8.3.3 Funções de 3 variáveis

Linha	A	B	C	$F(A,B,C)$
0	0	0	0	F_0
1	0	0	1	F_1
2	0	1	0	F_2
3	0	1	1	F_3
4	1	0	0	F_4
5	1	0	1	F_5
6	1	1	0	F_6
7	1	1	1	F_7

Tabela 8.12: Tabela verdade para funções de 3 variáveis.

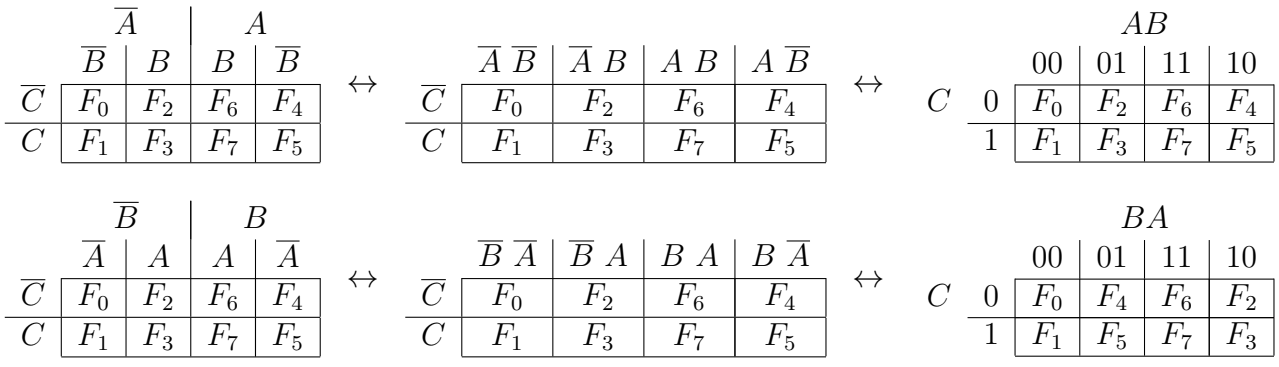


Figura 8.3: Exemplos de mapas de Karnaugh para funções de 3 variáveis.

8.3.4 Funções de 4 variáveis

Linha	A	B	C	D	$F(A,B,C,D)$
0	0	0	0	0	F_0
1	0	0	0	1	F_1
2	0	0	1	0	F_2
3	0	0	1	1	F_3
4	0	1	0	0	F_4
5	0	1	0	1	F_5
6	0	1	1	0	F_6
7	0	1	1	1	F_7
8	1	0	0	0	F_8
9	1	0	0	1	F_9
10	1	0	1	0	F_{10}
11	1	0	1	1	F_{11}
12	1	1	0	0	F_{12}
13	1	1	0	1	F_{13}
14	1	1	1	0	F_{14}
15	1	1	1	1	F_{15}

Tabela 8.13: Tabela verdade para funções de 4 variáveis.

		$\overline{A}$		A	
		$\overline{B}$	B	B	$\overline{B}$
$\overline{C}$	$\overline{D}$	F_0	F_4	F_{12}	F_8
	D	F_1	F_5	F_{13}	F_9
C	D	F_3	F_7	F_{15}	F_{11}
	$\overline{D}$	F_2	F_6	F_{14}	F_{10}

$\longleftrightarrow$

	$\overline{A}\ \overline{B}$	$\overline{A}\ B$	$A\ B$	$A\ \overline{B}$
$\overline{C}\ \overline{D}$	F_0	F_4	F_{12}	F_8
$\overline{C}\ D$	F_1	F_5	F_{13}	F_9
$C\ D$	F_3	F_7	F_{15}	F_{11}
$C\ \overline{D}$	F_2	F_6	F_{14}	F_{10}

$\updownarrow$

		AB			
		00	01	11	10
CD	00	F_0	F_4	F_{12}	F_8
	01	F_1	F_5	F_{13}	F_9
	11	F_3	F_7	F_{15}	F_{11}
	10	F_2	F_6	F_{14}	F_{10}

Figura 8.4: Exemplos de mapas de Karnaugh para funções de 4 variáveis.

8.4 Preenchimento do mapa-K

Por construção, cada uma das localizações do mapa é associada a cada uma das combinações das variáveis das quais a função é dependente. Isso equivale a dizer que cada uma das localizações do mapa é associada a uma linha da tabela verdade da função. Logo, cada uma das localizações do mapa deve ser preenchida com o respectivo valor da função (0 ou 1).

Para montar o mapa de uma função lógica e empregá-lo para simplificar uma forma SOP, deve-se manter os valores 1 (mintermos) no mapa e ignorar os valores 0 (maxtermos), a fim de facilitar a identificação visual.

Para montar o mapa de uma função lógica e empregá-lo para simplificar uma forma POS, deve-se manter os valores 0 (maxtermos) no mapa e ignorar os valores lógicos 1 (mintermos), a fim de facilitar a identificação visual.

Por exemplo, considerando os mintermos e os maxtermos separadamente, as Figuras 8.5 e 8.6 ilustram os mapas-K para a Equação (8.1), respectivamente.

C \ AB	00	01	11	10
0		1	1	
1	1		1	

Figura 8.5: Mapa de Karnaugh relativo à Equação (8.1), considerando apenas os mintermos.

C \ AB	00	01	11	10
0	0			0
1		0		0

Figura 8.6: Mapa de Karnaugh relativo à Equação (8.1), considerando apenas os maxtermos.

8.5 Mapa-K como forma de expressão

Uma função de variáveis booleanas pode ser expressa por uma equação genérica, por uma forma do grupo SOP, por uma forma do grupo POS, por uma forma padrão do grupo SOP, por uma forma padrão do grupo POS por uma lista de mintermos, por uma lista de maxtermos e por uma tabela verdade.

Além de ser usado como ferramenta de minimização, o mapa-K pode ser visto como mais uma alternativa de representação para funções booleanas.

As transformações entre: i) uma equação genérica, ii) uma forma dos grupos SOP ou POS e iii) uma forma padrão dos grupos SOP ou POS, envolvem manipulação algébrica das equações.

Por outro lado, as transformações realizadas entre uma lista de mintermos ou maxtermos, uma tabela verdade, um mapa-K e as demais representações, envolvem catalogação direta.

Portanto, partindo-se de uma dada forma de representação, pode-se facilmente obter todas as demais, independentemente do tipo de mapeamento utilizado.

Um exemplo de tais relacionamentos pode ser obtido a partir da função dada por

$$F(A,B,C) = \overline{\left(\overline{A} + \overline{(B+C)}\right)} \ . \tag{8.4}$$

Após alguma manipulação algébrica, a Equação (8.4) pode gerar a forma POS

$$F(A,B,C) = (A) \cdot (B+C) \tag{8.5}$$

e a forma SOP

$$F(A,B,C) = (A \cdot B) + (A \cdot C) \ . \tag{8.6}$$

Expandindo-se os termos das Equações (8.5) e (8.6), obtém-se, respectivamente, a forma padrão POS

$$\begin{aligned} F(A,B,C) &= (A+B+C) \cdot (A+B+\overline{C}) \cdot (A+\overline{B}+C) \cdot (A+\overline{B}+\overline{C}) \cdot (\overline{A}+B+C) \\ &= \prod M(0,1,2,3,4) \end{aligned} \tag{8.7}$$

e a forma padrão SOP

$$\begin{aligned} F(A,B,C) &= (A \cdot \overline{B} \cdot C) + (A \cdot B \cdot \overline{C}) + (A \cdot B \cdot C) \\ &= \sum m(5,6,7) \ . \end{aligned} \tag{8.8}$$

Por sua vez, a tabela verdade referente à Equação (8.4) é apresentada na Tabela 8.14.

Linha	A	B	C	$F(A,B,C)$
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

Tabela 8.14: Tabela verdade relativa à Equação (8.4).

Finalmente, o mapa-K da função é mostrado na Figura 8.7.

C \ AB	00	01	11	10
0	0	0	1	0
1	0	0	1	1

Figura 8.7: Mapa de Karnaugh relativo à Equação (8.4).

8.6 Mapa-K na simplificação de expressões booleanas

8.6.1 Adjacência lógica, aglutinação e replicação

A simplificação algébrica de expressões booleanas baseia-se na utilização de duas operações: a aglutinação e a replicação.

Se dois termos diferem de apenas um literal (A e $\overline{A}$), a aplicação da aglutinação permite simplificá-los em um único termo, sem o literal em questão. Tais termos são ditos logicamente adjacentes. Isso pode ser exemplificado por

$$\begin{aligned} F(A,B,C) &= (\overline{A}\cdot\overline{B}\cdot\overline{C}) + (A\cdot\overline{B}\cdot\overline{C}) + (\overline{A}\cdot B\cdot C) + (A\cdot B\cdot C) \\ &= (\overline{B}\cdot\overline{C}) + (B\cdot C)\ . \end{aligned} \tag{8.9}$$

Por sua vez, a replicação permite que um mesmo termo seja utilizado em simplificações envolvendo diversos outros termos. Um exemplo de replicação é dado por

$$\begin{aligned} F(A,B,C) &= (\overline{A}\cdot\overline{B}\cdot\overline{C}) + (\overline{A}\cdot B\cdot\overline{C}) + (A\cdot\overline{B}\cdot\overline{C}) \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}) + (\overline{A}\cdot B\cdot\overline{C}) + (A\cdot\overline{B}\cdot\overline{C}) + (\overline{A}\cdot\overline{B}\cdot\overline{C}) \\ &= (\overline{A}\cdot\overline{C}) + (\overline{B}\cdot\overline{C})\ . \end{aligned} \tag{8.10}$$

Os mapas de Karnaugh são construídos de tal forma que as adjacências geométricas do mapa são equivalentes às adjacências lógicas dos termos das equações. Portanto, a combinação algébrica dos termos de uma equação é equivalente à combinação de termos adjacentes do mapa. Assim sendo, a equação pode ser simplificada através da leitura direta da informação do mapa. O mapa da Figura 8.8 exemplifica a Equação (8.9), onde são realizadas as combinações $(m_0 - m_4)$ e $(m_3 - m_7)$.

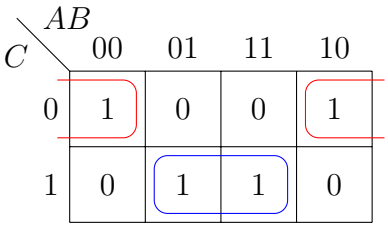


Figura 8.8: Mapa de Karnaugh relativo à Equação (8.9).

No mapa, a replicação é interpretada como a combinação de um termo com os demais geometricamente adjacentes. O mapa da Figura 8.9 exemplifica a Equação (8.10), onde o mintermo m_0 é replicado para as combinações $(m_0 - m_4)$ e $(m_0 - m_2)$.

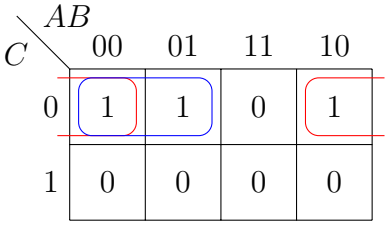


Figura 8.9: Mapa de Karnaugh relativo à Equação (8.10).

8.6.2 Seleção sistemática de termos (implicantes ou implicados)

Em uma seleção sistemática, duas definições são de grande auxílio na escolha de termos a serem agrupados para simplificação: termo essencial e termo primo.

Quando um termo original é coberto por um único agrupamento possível, o termo resultante do agrupamento é denominado de termo essencial. Isso indica que os termos essenciais devem ser incluídos na expressão mínima equivalente à função desejada.

Um termo original que não tenha sido coberto por qualquer agrupamento anterior deve ser incluído em um agrupamento máximo, o qual será denominado de termo primo. Isso indica que os termos primos devem ser incluídos na expressão mínima equivalente à função desejada.

Deve ser ressaltado que nem todo agrupamento máximo representa um termo primo.

A Figura 8.10 apresenta o mapa da função dada por

$$\begin{aligned} F(A,B,C) &= \sum m(0,2,3,5,6,7) \\ &= (\overline{A}\cdot\overline{B}\cdot\overline{C}) + (\overline{A}\cdot B\cdot\overline{C}) + (\overline{A}\cdot B\cdot C) + \\ &\quad (A\cdot\overline{B}\cdot C) + (A\cdot B\cdot\overline{C}) + (A\cdot B\cdot C)\ . \end{aligned} \tag{8.11}$$

Observam-se os seguintes grupamentos possíveis: $(m_0{-}m_2)$, $(m_5{-}m_7)$ e $(m_2{-}m_3{-}m_6{-}m_7)$.

Os agrupamentos $(m_0 - m_2)$ e $(m_5 - m_7)$ são considerados essenciais, porque são as únicas possibilidades de simplificação para os mintermos m_0 e m_5.

Após os agrupamentos essenciais, sobram os mintermos m_3 e m_6. Um agrupamento primo $(m_2 - m_3 - m_6 - m_7)$ pode ser realizado, duplicando-se os mintermos m_2 e m_7.

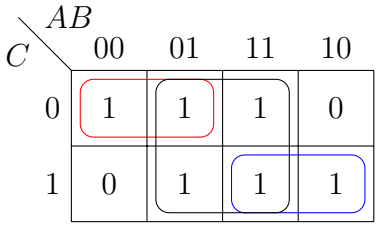


Figura 8.10: Mapa de Karnaugh relativo à Equação (8.11).

Assim, uma forma sistemática de escolha de termos é:

S1 - Identificar todas as possibilidades de agrupamento, através dos maiores grupos possíveis.

S2 - Marcar todos os termos originais cobertos por apenas 1 agrupamento. Tais agrupamentos formam os termos essenciais.

S3 - Listar todos os termos essenciais.

S4 - Usar os maiores agrupamentos possíveis para cobrir os termos originais não cobertos pelos termos essenciais, formando termos primos.

S5 - Listar apenas tais termos primos.

S6 - Montar a expressão mínima, a partir das duas listas.

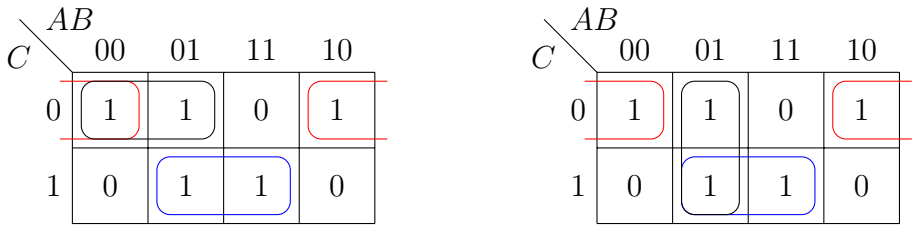
No caso da existência de diversas formas mínimas equivalentes, em uma SOP ou em uma POS, deve-se aplicar algum critério extra para a escolha final.

Dada uma função, e suas formas SOP e POS, nada se pode garantir em relação a qual das duas conduzirá à expressão mais simples. Assim, é necessário encontrar a forma mínima de ambas e decidir qual delas é a mais simples.

8.6.3 Mapa-K de funções com múltiplos mínimos e mapa cíclico

Algumas equações booleanas não possuem uma forma mínima única. Isso acontece porque, em um conjunto de termos da expressão, cada um deles é coberto por mais de um agrupamento de termos logicamente equivalentes. Assim sendo, não é possível selecionar um conjunto único de termos essenciais e/ou primos. Em tais casos, o que se deve fazer é avaliar as possíveis soluções e escolher a de menor custo. Caso ainda existam opções logicamente equivalentes, todas de mesmo custo, deve-se adotar algum critério extra de escolha.

A Figura 8.11 apresenta o mapa de uma função com múltiplas formas mínimas, envolvendo o termo m_2. As Figuras 8.11a e 8.11b respectivamente ilustram as duas soluções de mesmo custo, que são as seguintes: a) $(m_0 - m_4), (m_3 - m_7), (m_0 - m_2)$ e b) $(m_0 - m_4), (m_3 - m_7), (m_3 - m_2)$.

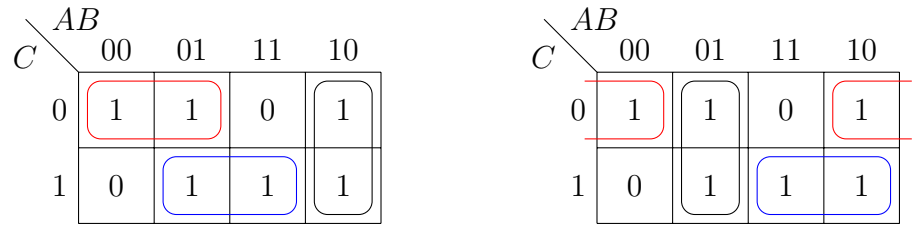


(a) Forma mínima 1. (b) Forma mínima 2.

Figura 8.11: Mapa de Karnaugh com múltiplas formas mínimas.

Em alguns casos particulares, todos os termos de um subconjunto dos termos da função são cobertos por mais de um agrupamento, todos de mesmo custo. Tal subconjunto de termos forma um ciclo. Mapas de funções com tal característica são denominados de mapas cíclicos. Nesses casos, deve-se adotar algum critério extra de escolha para quebrar o ciclo.

A Figura 8.12 apresenta um mapa com ciclo, que possui duas soluções de mesmo custo. As Figuras 8.12a e 8.12b respectivamente ilustram as duas soluções de mesmo custo, que são as seguintes: a) $(m_0 - m_2), (m_3 - m_7), (m_4 - m_5)$ e b) $(m_0 - m_4), (m_5 - m_7), (m_2 - m_3)$.



(a) Solução 1 para quebra de ciclo. (b) Solução 2 para quebra de ciclo.

Figura 8.12: Mapa de Karnaugh com ciclo.

8.6.4 Mapa-K de funções não completamente especificadas

Em alguns problemas, as funções booleanas podem ser não completamente especificadas. Nesses casos, duas situações podem ocorrer. Na primeira delas, para uma dada combinação de valores dos literais, o valor da função não é relevante, caracterizando uma indeterminação por *don't-care*. Por outro lado, pode acontecer que uma determinada combinação de literais nunca ocorra, caracterizando uma indeterminação por *can't-happen*. Em ambas as situações, pode-se especificar livremente qualquer um dos valores lógicos para a função. Para caracterizar uma indeterminação, costuma-se atribuir um valor lógico indeterminado X. Os valores indeterminados podem ser úteis no processo de simplificação de formas padrões SOP e POS.

A Tabela 8.15 exemplifica uma função incompletamente especificada, a qual também pode ser expressa por

$$F(A,B,C) = \sum m(0,3,4) + \sum d(2,7) = \prod M(1,5,6) \cdot \prod d(2,7)\ . \tag{8.12}$$

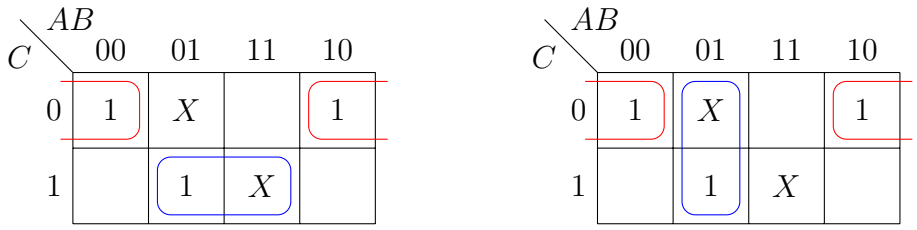
Linha	A	B	C	$F(A,B,C)$
0	0	0	0	1
1	0	0	1	0
2	0	1	0	X
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	X

Tabela 8.15: Tabela verdade de função incompletamente especificada.

As Figuras 8.13 a 8.15 ilustram, respectivamente, os mapas de Karnaugh da função definida na Equação (8.12), dos seus mintermos e dos seus maxtermos.

C \ AB	00	01	11	10
0	1	X	0	1
1	0	1	X	0

Figura 8.13: Mapa de Karnaugh da Tabela 8.15.

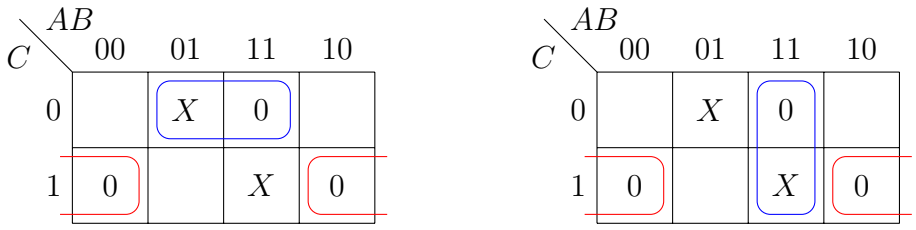


(a) Forma mínima 1. (b) Forma mínima 2.

Figura 8.14: Mapa de Karnaugh dos mintermos da Tabela 8.15.

Da configuração de mintermos apresentada no mapa da Figura 8.14, pode-se escrever que

$$\begin{aligned} F(A,B,C) &= \sum m(0,3,4) + \sum d(2,7) \\ &= (m_0 + m_4) + (m_3 + d_7) \;=\; (\overline{B} \cdot \overline{C}) + (B \cdot C) \\ &= (m_0 + m_4) + (m_3 + d_2) \;=\; (\overline{B} \cdot \overline{C}) + (\overline{A} \cdot B)\ . \end{aligned} \tag{8.13}$$



(a) Forma mínima 1. (b) Forma mínima 2.

Figura 8.15: Mapa de Karnaugh dos maxtermos da Tabela 8.15.

Da configuração de maxtermos apresentada no mapa da Figura 8.15, pode-se escrever que

$$\begin{aligned} F(A,B,C) &= \prod M(1,5,6) \cdot \prod d(2,7) \\ &= (M_1 \cdot M_5) \cdot (M_6 \cdot d_2) = (B + \overline{C}) \cdot (\overline{B} + C) \\ &= (M_1 \cdot M_5) \cdot (M_6 \cdot d_7) = (B + \overline{C}) \cdot (\overline{A} + \overline{B}) \ . \end{aligned} \tag{8.14}$$

As Equações (8.13) e (8.14) mostram que os valores lógicos indeterminados podem ser usados, ou não, no processo de simplificação. Elas ilustram ainda o papel relevante dos valores indeterminados na simplificação de funções booleanas.

Deve ser ressaltado que, uma vez escolhidos como "0" ou como "1", os valores indeterminados "X", bem como a função original, perdem a sua característica de indeterminação na expressão mínima. Assim sendo, a função minimizada final passa a ser completamente especificada.

8.7 Exercícios propostos

1. Obtenha, utilizando o Mapa de Karnaugh, as formas SOP mínima e POS mínima para as seguintes equações booleanas:

   (a) $F(A,B,C) = (\overline{A}\cdot\overline{B}\cdot\overline{C}) + (\overline{A}\cdot B\cdot\overline{C}) + (A\cdot\overline{B}\cdot C) + (A\cdot B\cdot C)$.
   (b) $F(A,B,C) = (\overline{A}\cdot\overline{B}\cdot C) + (\overline{A}\cdot B\cdot C) + (A\cdot\overline{B}\cdot\overline{C}) + (A\cdot B\cdot\overline{C})$.
   (c) $F(A,B,C) = (\overline{A}\cdot\overline{B}\cdot\overline{C}) + (\overline{A}\cdot\overline{B}\cdot C) + (A\cdot B\cdot\overline{C}) + (A\cdot B\cdot C)$.
   (d) $F(A,B,C) = (\overline{A}\cdot B\cdot\overline{C}) + (\overline{A}\cdot B\cdot C) + (A\cdot\overline{B}\cdot\overline{C}) + (A\cdot\overline{B}\cdot C)$.
   (e) $F(A,B,C) = (\overline{A}\cdot\overline{B}\cdot\overline{C}) + (\overline{A}\cdot B\cdot C) + (A\cdot\overline{B}\cdot\overline{C}) + (A\cdot B\cdot C)$.
   (f) $F(A,B,C) = (\overline{A}\cdot\overline{B}\cdot C) + (\overline{A}\cdot B\cdot\overline{C}) + (A\cdot\overline{B}\cdot C) + (A\cdot B\cdot\overline{C})$.

2. Obtenha, utilizando o Mapa de Karnaugh, as formas SOP mínima e POS mínima para as seguintes equações booleanas:

   (a) $F(A,B,C) = (A+B+\overline{C})\cdot(A+\overline{B}+\overline{C})\cdot(\overline{A}+B+C)\cdot(\overline{A}+\overline{B}+C)$.
   (b) $F(A,B,C) = (A+B+C)\cdot(A+\overline{B}+C)\cdot(\overline{A}+B+\overline{C})\cdot(\overline{A}+\overline{B}+\overline{C})$.
   (c) $F(A,B,C) = (A+\overline{B}+C)\cdot(A+\overline{B}+\overline{C})\cdot(\overline{A}+B+C)\cdot(\overline{A}+B+\overline{C})$.
   (d) $F(A,B,C) = (A+B+C)\cdot(A+B+\overline{C})\cdot(\overline{A}+\overline{B}+C)\cdot(\overline{A}+\overline{B}+\overline{C})$.
   (e) $F(A,B,C) = (A+B+\overline{C})\cdot(A+\overline{B}+C)\cdot(\overline{A}+B+\overline{C})\cdot(\overline{A}+\overline{B}+C)$.
   (f) $F(A,B,C) = (A+B+C)\cdot(A+\overline{B}+\overline{C})\cdot(\overline{A}+B+C)\cdot(\overline{A}+\overline{B}+\overline{C})$.

3. Para os exercícios listados abaixo, considerar as equações booleanas apresentadas em seguida.

   (a) Obter a forma SOP padrão da equação fornecida.
   (b) Obter a forma SOP mínima, utilizando o mapa de Karnaugh correspondente.
   (c) Obter a forma POS padrão da equação fornecida.
   (d) Obter a forma POS mínima, utilizando o mapa de Karnaugh correspondente.
   (e) Apresentar a expressão mínima para função.

   Equações booleanas:

   i. $F(A,B,C) = \Big\{B\cdot\Big[\Big(\overline{A}\cdot\overline{C}\Big) + (A\cdot C)\Big]\Big\} + \overline{\Big\{B + \Big[\Big(\overline{A}+\overline{C}\Big)\cdot(A+C)\Big]\Big\}}$.
   ii. $F(A,B,C) = \overline{\Big\{\Big[\overline{(A+B)}+C\Big]\cdot\Big[\overline{A}+\overline{\Big(B+\overline{C}\Big)}\Big]\Big\}}$.
   iii. $F(A,B,C) = \Big[\overline{\Big(A\cdot\overline{B}\Big)}+C\Big]\cdot\overline{[(A+C)\cdot B]}$.

4. Construa a Tabela Verdade e o Mapa de Karnaugh a partir das expressões booleanas mínimas apresentadas nas Equações (8.13) e (8.14). Perceba as diferenças entre tais representações, bem como as diferenças entre cada uma delas e a Tabela Verdade original e o Mapa de Karnaugh original, apresentados na Tabela 8.15 e na Figura 8.13, respectivamente.

5. Para os exercícios listados abaixo, considerar as equações booleanas apresentadas em seguida.

   (a) Obter a forma SOP mínima, utilizando o mapa de Karnaugh correspondente.

   (b) Obter a forma POS mínima, utilizando o mapa de Karnaugh correspondente.

   (c) Apresentar a expressão mínima para função.

   Equações booleanas:

   i. $F(A,B,C) = \sum m(0,2,7)$
   ii. $F(A,B,C) = \sum m(0,2,3)$
   iii. $F(A,B,C) = \sum m(0,2,4,7)$
   iv. $F(A,B,C) = \sum m(0,2,5,7)$
   v. $F(A,B,C) = \sum m(0,2,3,7)$
   vi. $F(A,B,C) = \sum m(1,2,3,7)$
   vii. $F(A,B,C) = \sum m(0,1,2,3)$
   viii. $F(A,B,C) = \sum m(0,1,3,4,5)$
   ix. $F(A,B,C) = \sum m(0,1,2,4,6,7)$
   x. $F(A,B,C) = \sum m(1,2,3,5,6,7)$
   xi. $F(A,B,C,D) = \sum m(5,6,12,13,14,15)$
   xii. $F(A,B,C,D) = \sum m(5,6,8,11,12,13,14,15)$
   xiii. $F(A,B,C,D) = \sum m(1,3,5,7,12,13,14,15)$
   xiv. $F(A,B,C,D) = \sum m(4,5,11,13,15)$
   xv. $F(A,B,C,D) = \sum m(0,1,5,6,7,14)$
   xvi. $F(A,B,C,D) = \sum m(0,1,2,6,7,8,9,10,14)$
   xvii. $F(A,B,C,D) = \sum m(0,1,2,5,6,7,8,9,10,14)$
   xviii. $F(A,B,C,D) = \sum m(0,1,2,6,7,8,9,10,14,15)$
   xix. $F(A,B,C,D) = \sum m(0,1,2,5,6,7,8,9,10,14,15)$
   xx. $F(A,B,C,D) = \sum m(0,1,2,5,6,7,8,9,10,13,14,15)$
   xxi. $F(A,B,C,D) = \sum m(0,2,6,7,8,9,10,12,13)$
   xxii. $F(A,B,C,D) = \sum m(0,2,6,7,8,9,10,12,13,15)$
   xxiii. $F(A,B,C,D) = \sum m(0,2,4,6,10,11,14,15)$
   xxiv. $F(A,B,C,D) = \sum m(0,1,2,3,5,7,15)$
   xxv. $F(A,B,C,D) = \sum m(0,1,2,3,5,7,14,15)$
   xxvi. $F(A,B,C,D) = \sum m(0,1,2,3,5,7,11,15)$
   xxvii. $F(A,B,C,D) = \sum m(0,1,2,3,5,7,13,15)$
   xxviii. $F(A,B,C,D) = \sum m(0,2,5,6,8,10,13)$
   xxix. $F(A,B,C,D) = \sum m(0,2,5,7,8,10,11,13,15)$
   xxx. $F(A,B,C,D) = \sum m(1,5,6,7,11,12,13,15)$
   xxxi. $F(A,B,C,D) = \sum m(2,3,4,5,6,7,10,12,13,15)$
   xxxii. $F(A,B,C,D) = \sum m(1,2,4,6,7,9,11,12,13,14,15)$

Capítulo 9

Sistemas de numeração

9.1 Introdução

- Sistema numérico (*number system*) × sistema de numeração (*numeral system*).
- Sistemas numéricos classificam o tipo da quantidade numérica: $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, etc..
- Sistemas de numeração tratam da representação da quantidade numérica: grupos de numerais (símbolos) que representam quantidades.
- Máquinas digitais possuem capacidade de armazenamento finito. Um registro só pode armazenar uma quantidade finita de elementos básicos de informação e a máquina só pode armazenar uma quantidade finita de registros. Portanto, toda quantidade armazenada será uma aproximação da quantidade original. O sistema de numeração utilizado pela máquina tem influência direta na qualidade dessa aproximação.
- Da mesma forma, a eficiência de uma determinada implementação para as operações aritméticas básicas (adição, subtração, multiplicação e divisão) também é influenciada pelo sistema de numeração utilizado pela máquina. Deve ser ressaltado que a eficiência é, geralmente, medida em relação ao tempo necessário para a realização da operação, à quantidade de elementos constituintes utilizados, aos tipos de tais elementos e ao consumo de energia.
- Assim, através da escolha adequada entre as diversas alternativas matemáticas para a representação de quantidades, bem como da sua implementação (máquina e linguagem de programação), procura-se reduzir o erro das aproximações e/ou tornar as operações aritméticas mais eficientes.
- Sistemas comumente usados em máquinas digitais:
  - Posicional.
  - Resíduos (ou resto).
  - Racional.
  - Logarítmico.

- Sistemas de numeração posicional
  - Sistema vetorial posicional.
  - É definido um conjunto básico de dígitos ou símbolos $\boldsymbol{S} = \{s_1, s_2, \cdots, s_M\}$.
  - Os números x são representados por grupos de dígitos (vetores) pertencentes a $\boldsymbol{S}$: $\boldsymbol{x} = [d_N, \cdots, d_2, d_1]_S$, onde $d_i \in S$.
  - A posição de cada dígito no vetor tem significado.
  - A cada posição i é associado um peso numérico w_i, o qual é multiplicado pelo dígito d_i correspondente: $\boldsymbol{w} = [w_N, \cdots, w_2, w_1]$.
  - Os dígitos d_i representam números inteiros, podendo ser positivos e/ou negativos.
  - Os pesos podem ser os mais diversos possíveis.

- Sistemas de numeração de resíduos (ou restos)
  - Sistema vetorial não posicional.
  - É definido um vetor de elementos primos entre si dois a dois: $\boldsymbol{m} = [m_1, m_2, \cdots, m_N]$.
  - São calculados os resíduos (restos) r_i da divisão de um número inteiro x por cada elemento m_i.
  - Os números x são representados por um vetor contendo os resíduos: $\boldsymbol{x} = [r_1, r_2, \cdots, r_N]_m$.
  - Nas operações aritméticas, os resíduos podem ser tratados independentemente, acelerando o processo de cálculo.

- Sistemas de numeração racional
  - Representação de números através de frações.
  - Numerador e denominador da fração são representados por números inteiros.
  - As operações aritméticas são realizadas sem erro, mesmo em uma máquina com precisão finita.

- Sistemas de numeração logarítmico
  - Um número real $\mu > 1$ é definido como base.
  - É gerado um conjunto de números reais $\boldsymbol{L_\mu} = \{x \mid |x| = \mu^i, i \in \mathbb{Z}\}\ U\ \{0\}$.
  - É objetivada uma melhoria de precisão na representação dos números, conseguida através de arredondamento geométrico.

9.2 Sistema de numeração posicional convencional

Nesta seção, são abordados os seguintes aspectos do sistema de numeração posicional convencional (SNPC):

- Representação de números com partes inteira e fracionária.
- Representação de números positivos, nulos e negativos.
- Tabelas de operações básicas entre dígitos.
- Escalamento por potência inteira da base.
- Conversão entre bases.
- Bases mais comuns em circuitos digitais.

9.2.1 Representação de números inteiros não negativos

Para representar quantidades numéricas inteiras, ordenadas e não negativas, o sistema de numeração posicional convencional utiliza um conjunto ordenado e não negativo de símbolos simples (dígitos) $d_i \in \boldsymbol{S} = \{s_1, s_2, \cdots, s_M\} = \{0, 1, 2, \cdots, (b-1)\}$, juntamente com uma técnica de poderação ou escala. O número de elementos de $\boldsymbol{S}$, $M = b$, é denominado **base** ou **radical** (*radix*) do sistema de numeração. Os pesos ou fatores de escala utilizados são potências inteiras da base $w_i \in \boldsymbol{W} = \{w_1, w_2, w_3, \cdots\} = \{b^0, b^1, b^2, \cdots\}$.

Uma visão geométrica modular do processo de representação pode ser encontrada nas Figuras 9.1 a 9.4, para $b = 3$. Para representar cada uma das quantidades $q < b$, é utilizado apenas um dos elementos de $\boldsymbol{S}$, como na Figura 9.1. Para as quantidades $q \geq b$, como não existem outros símbolos disponíveis, repetem-se os elementos de $\boldsymbol{S}$, em módulos de comprimento b, como exemplificado na Figura 9.2. Porém, isso gera ambigüidade na representação, a qual é resolvida através da combinação de símbolos, como ilustrado na Figura 9.3. Agora, a cada módulo de b símbolos, no nível básico de representação $L = 0$, são justapostos os elementos de $\boldsymbol{S}$, formando um novo nível de representação $L = 1$. Essa técnica é aplicada, sucessivamente, cada vez que o número de possibilidades de representação em um determinado nível L se esgota e uma nova ambigüidade é gerada pela repetição de símbolos, como é apresentado na Figura 9.4.

Em cada nível da representação existe um módulo formado pelos símbolos de $\boldsymbol{S}$. Devido à lei de formação empregada, o comprimento do módulo em cada nível é uma versão escalada dos comprimentos dos módulos dos níveis inferiores. Os fatores de escala são as potências inteiras da base $\boldsymbol{W} = \{b^0, b^1, b^2, \cdots\}$. As mudanças de símbolos, dentro de cada nível, são reguladas pelo fator de escala do nível. Dessa forma, dentro de cada nível $L = 0, 1, 2, ..., (N-1)$, ocorre uma mudança de símbolos a cada b^L unidades da quantidade representada.

0	1	2

Figura 9.1: Representação de quantidades $q < b$, para $b = 3$.

0	1	2	0	1	2	0	1	2

Figura 9.2: Representação de quantidades $q \geq b$, para $b = 3$, com ambigüidade.

0	1	2	0	1	2	0	1	2
0			1			2		

Figura 9.3: Representação de quantidades $q \geq b$, para $b = 3$, com eliminação da ambigüidade através da justaposição dos dígitos.

0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
0			1			2			0			1			2			0			1			2		
0									1									2								

Figura 9.4: Uso repetido da técnica de justaposição de dígitos para representação de quantidades $q \geq b$, para $b = 3$, sem ambigüidade.

Algebricamente, a idéia geométrica modular de uma combinação de níveis pode ser expressa por uma soma de níveis de valores, onde o valor numérico de cada nível k é expresso por um dígito $d_k \in \boldsymbol{S}$, ponderado por um fator $w_k \in \boldsymbol{W}$, conforme a Equação (9.1). A notação pode ser simplificada através da justaposição dos dígitos, acompanhada da especificação da base, como ilustrado na Equação (9.2). Nos casos onde se opera sempre com a mesma base, a sua indicação pode ser omitida, como na Equação (9.3).

$$(q_I)_b = (d_{N-1} \times b^{N-1}) + \cdots + (d_2 \times b^2) + (d_1 \times b^1) + (d_0 \times b^0) = \sum_{k=0}^{N-1} d_k b^k \ . \tag{9.1}$$

$$(q_I)_b = \sum_{k=0}^{N-1} d_k b^k = [d_{N-1} \cdots d_2 d_1 d_0]_b \ . \tag{9.2}$$

$$q_I = \sum_{k=0}^{N-1} d_k b^k = [d_{N-1} \cdots d_2 d_1 d_0] \ . \tag{9.3}$$

9.2.2 Representação de números fracionários não negativos

Para representar quantidades numéricas puramente fracionárias, ordenadas e não negativas, o sistema de numeração posicional convencional utiliza o mesmo mecanismo empregado com números inteiros. Nesse caso, os pesos w_i, usados para ponderar os dígitos d_i, são potências inteiras e negativas da base b.

Partindo-se das Equações (9.1) a (9.3), pode-se dizer que

$$(q_F)_b = b^{-N} \cdot (q_I)_b = (d_{N-1} \times b^{-1}) + \cdots + (d_1 \times b^{-N+1}) + (d_0 \times b^{-N}) = \sum_{k=0}^{N-1} d_k b^{-N+k} \tag{9.4}$$

ou, utilizando-se uma notação mais genérica, que

$$(q_F)_b = (d_{-1} \times b^{-1}) + (d_{-2} \times b^{-2}) + \cdots + (d_{-N} \times b^{-N}) = \sum_{k=-N}^{-1} d_k b^k = [d_{-1} d_{-2} \cdots d_{-N}]_b \ . \tag{9.5}$$

Na representação simplificada por um vetor de dígitos, para uso humano, emprega-se um símbolo extra para diferenciar as representações de números puramente inteiros, puramente fracionários e com partes inteira e fracionária. Normalmente é utilizado um ponto ou uma vírgula, como é ilustrado na Equação (9.6), para números puramente inteiros, na Equação (9.7), para números puramente fracionários, e na Equação (9.8), para números com partes inteira e fracionária.

$$(q_I)_b = \sum_{k=0}^{N_I-1} d_k b^k = [d_{N_I-1} \cdots d_1 d_0 \cdot\ ]_b = [d_{N_I-1} \cdots d_1 d_0 \ \cdot\ 0]_b \ . \tag{9.6}$$

$$(q_F)_b = \sum_{k=-N_F}^{-1} d_k b^k = [\ \cdot\ d_{-1} d_{-2} \cdots d_{-N_F}]_b = [0 \cdot d_{-1} d_{-2} \cdots d_{-N_F}]_b \ . \tag{9.7}$$

$$(q)_b = (q_I)_b + (q_F)_b = \sum_{k=-N_F}^{N_I-1} d_k b^k = [d_{N_I-1} \cdots d_2 d_1 d_0 \cdot d_{-1} d_{-2} \cdots d_{-N_F}]_b \ . \tag{9.8}$$

Na representação utilizada nos circuitos digitais o símbolo extra não é utilizado, uma vez que o conhecimento de quantos dígitos são empregados para as partes inteira e fracionária transformam-no em uma informação redundante e, portanto, dispensável.

Para o uso humano, a redundância é útil para facilitar a visualização das partes inteira e fracionária, bem como para sua manipulação.

9.2.3 Representação de números inteiros negativos

Na representação matemática para uso humano, uma forma de diferenciar números positivos e negativos é a adição dos símbolos "+" e "−", respectivamente. Tais símbolos também podem ser interpretados como operadores unários. Logo, a menos que seja necessário resolver alguma ambigüidade, o símbolo "+" é dispensado, uma vez que não realiza qualquer modificação sobre a quantidade original.

Na representação utilizada nos circuitos digitais, é necessário empregar um dos próprios símbolos utilizados na codificação de quantidades para diferenciar quantidades positivas e negativas, devido a não existência de outros símbolos.

Diversas formas de recodificação podem ser encontradas para os vetores de dígitos que representam as quantidades numéricas. As mais comuns são discutidas a seguir.

Conceitos básicos

- Representação numérica
  - Sistema: SNPC com base b.
  - Dígitos: $d_i \in S = \{0, 1, 2, \cdots, (b-1)\}$.
  - Representação: vetor de N dígitos.
- Significado dos N dígitos
  - O dígito mais significativo representa o sinal: $d_{N-1} = s_{N-1}$.
  - Os restantes $(N-1)$ dígitos representam a quantidade numérica.

- Números não negativos
  - Dígito mais significativo: $d_{N-1} = s_{N-1} = 0$.
  - Representação: $(q_{I+})_b = [s_{N-1} d_{N-2} \cdots d_2 d_1 d_0]_b = [0\ d_{N-2} \cdots d_2 d_1 d_0]_b$.
  - Codificação: sinal-e-magnitude.
- Números negativos
  - Dígito mais significativo: $d'_{N-1} = s'_{N-1} = (b-1)$.
  - Representação: $(q_{I-})_b = [s'_{N-1} d'_{N-2} \cdots d'_2 d'_1 d'_0]_b = [(b-1)\ d'_{N-2} \cdots d'_2 d'_1 d'_0]_b$.
  - Codificações:
    - Sinal-e-magnitude.
    - Sinal-e-complemento (magnitude recodificada):
      - Complemento à base (b).
      - Complemento à base diminuída ($b-1$).
- A seguir, são abordadas as codificações de números negativos para $b = 2$.

Visão geral das codificações

- As Figuras 9.5 a 9.10 apresentam uma visão geométrica das codificações, para $b = 2$.
- Na representação de números inteiros não negativos, é utilizada a justaposição de N dígitos para representar b^N valores consecutivos, na faixa $[0; (b^N - 1)]$.
  Por exemplo, para $b = 2$ e $N = 4$, o valor $V = (6)_{10}$ é representado por $V = (0110)_2$.
  Isso é ilustrado na Figura 9.5.
- Para possibilitar a representação de números inteiros negativos, o dígito mais significativo é utilizado para simbolizar os sinais "+" e "−". Normalmente, são adotados "0" e "$(b-1)$", respectivamente. Portanto, uma metade dos b^N possíveis padrões de dígitos é usada para representar os números positivos e o zero, enquanto a outra metade pode ser usada para representar números negativos.
- Para os valores não negativos, a associação entre números e padrões de dígitos é a mesma utilizada anteriormente, adotando-se o dígito mais significativo com valor "0" para representar o sinal "+".
  Por exemplo, para $b = 2$ e $N = 4$, o valor $V_+ = (+6)_{10}$ é representado por $V_+ = (0110)_2$.
  Isso é ilustrado na Figura 9.6.
- Por outro lado, para os valores negativos, três outros tipos de associação são comumente empregados, onde os valores binários originais são recodificados.
- Na codificação denominada de Sinal-e-Magnitude, a associação entre números negativos e padrões de dígitos é a mesma utilizada anteriormente, adotando-se o dígito mais significativo com valor "$(b-1)$", para representar o sinal "−".
  Por exemplo, para $b = 2$ e $N = 4$, o valor $V_- = (-6)_{10}$ é representado por $(V_-)_{SM} = (1110)_2$. Isso é ilustrado na Figura 9.7.
- Nas codificações denominadas de complementares, um valor negativo ($V_- = -|V|$) é representado por seu valor complementar (V_C) em relação a um determinado valor positivo de referência (V_R), de tal forma que $V_C = (V_R - |V|)$.

- Da mesma forma, o negativo de um valor negativo ($V_+ = -V_- = |V|$)) é representado por seu valor complementar (V_C) em relação ao valor de referência (V_R), de tal forma que $V_+ = (V_R - V_C) = [V_R - (V_R - |V|)] = |V|$.

- Comumente, utilizam-se duas codificações complementares, que são:

  – Complemento à base diminuída ($b - 1$) ou *Diminished Radix Complement* ou (b-1)s' *Complement*.
  Exemplo 1: Complemento-aos-9 ou Complemento-a-9 ou *nines' Complement*.
  Exemplo 2: Complemento-aos-1 ou Complemento-a-1 ou *ones' Complement*.

  – Complemento à base (b) ou *Radix Complement* ou b's *Complement*.
  Exemplo 1: Complemento-a-10 ou *ten's Complement*.
  Exemplo 2: Complemento-a-2 ou *two's Complement*.

- Na codificação denominada de Complemento à base diminuída ($b-1$), o valor de referência V_{Rd} é igual ao maior valor representável, de tal forma que $V_{Rd} = (b^N - 1)$.
Por exemplo, para $b = 2$ e $N = 4$, o valor $V_- = -(6)_{10} = -(0110)_2$ é representado pelo valor complementar $V_{C1} = (V_{R1} - |V_-|) = (15 - 6)_{10} = (9)_{10} = (1111 - 0110)_2 = (1001)_2$.
Isso é ilustrado na Figura 9.8.

- Na codificação denominada de Complemento à base (b), o valor de referência V_{Rb} é o valor seguinte ao maior valor representável, de tal forma que $V_{Rb} = (V_{Rd} + 1) = (b^N)$.
Por exemplo, para $b = 2$ e $N = 4$, o valor $V_- = -(6)_{10} = -(0110)_2$ é representado pelo valor complementar $V_{C2} = (V_{R2} - |V_-|) = (16 - 6)_{10} = (10)_{10} = (10000 - 0110)_2 = (1010)_2$.
Isso é ilustrado na Figura 9.9.

- Para uma melhor comparação entre elas, todas as codificações são reunidas na Figura 9.10.

Comentários sobre as operações

- A partir das definições apresentadas para as codificações complementares acima, deve-se notar que uma das representações pode ser convertida na outra pela simples adição ou subtração da unidade.
Por exemplo, para $b = 2$ e $N = 4$, o valor $V_- = -(6)_{10} = -(0110)_2$ pode ser representado por $V_{C1} = (1001)_2$ ou por $V_{C2} = (V_{C1} + 1) = (1010)_2$.

- Para o caso de $b = 2$, no Complemento-a-1, o valor de referência utilizado na subtração, dado por $V_{R1} = (2^N - 1) = (11 \cdots 11)_2$, faz com que, na prática, todos os dígitos do valor a ser complementado sejam trocados de "0" para "1" e vice-versa.
Por exemplo, para $b = 2$ e $N = 4$, o valor $V_- = -(6)_{10} = -(0110)_2$ é representado pelo valor complementar $V_{C1} = (V_{R1} - |V_-|) = (15 - 6)_{10} = (9)_{10} = (1111 - 0110)_2 = (1001)_2$.

- Para o caso de $b = 2$, no Complemento-a-2, o valor de referência utilizado na subtração, dado por $V_{R2} = 2^N = (100 \cdots 00)_2$, produz três efeitos distintos sobre os dígitos do valor a ser complementado. Os dígitos "0" menos significativos não são alterados. O dígito "1" menos significativo também não é alterado. Os demais dígitos, a partir do dígito "1" menos significativo, são trocados de "0" para "1" e vice-versa.
Por exemplo, para $b = 2$ e $N = 4$, o valor $V_- = -(6)_{10} = -(0110)_2$ é representado pelo valor complementar $V_{C2} = (V_{R2} - |V_-|) = (16 - 6)_{10} = (10)_{10} = (10000 - 0110)_2 = (1010)_2$.

Comentários sobre a nomenclatura

- Uma vez que as codificações complementares são geradas pelo complemento do valor a uma referência, e esta depende do número de dígitos N, é razoável pensar que as suas denominações deveriam mudar para cada valor de N. Nesse sentido, pensando na operação executada sobre o valor, as denominações Complemento à $2^{(b-1)}$ e Complemento à 2^b poderiam parecer mais adequadas. Porém, pensando na operação executada sobre cada dígito, eles são complementados em relação ao números "$(b-1)$", no Complemento à base diminuída, e "0" (que deve ser interpretado como "10" ou b), no Complemento à base. Assim, os nomes originais passam a fazer sentido.
  Por exemplo, para $b = 2$ e $N = 4$, o valor $V_- = -(6)_{10} = -(0110)_2$ é representado pelo valor complementar $V_{C1} = (V_{R1} - |V_-|) = (1111 - 0110)_2 = (1001)_2$. ou pelo valor complementar $V_{C2} = (V_{R2} - |V_-|) = (10000 - 0110)_2 = (1010)_2$.

- No caso do Complemento à base (b), pode-se pensar em uma versão escalada por b^{N-1} de todos os números envolvidos, o que não altera a codificação. Nesse caso, obtêm-se versões escaladas dos valores, dadas por $V < 1$ e uma versão escalada da própria referência, dada por $V_{Rb} = (b^N/b^{N-1}) = b$. Dessa forma, novamente pode-se pensar em manter um único nome, independente de N e do valor de referência.
  Por exemplo, para $b = 2$ e $N = 4$, o valor $V_- = -(6)_{10} = -(0110)_2$ é representado pelo valor complementar $V_{C16} = (V_{R16} - |V_-|) = (16. - 6.)_{10} = (10.)_{10} = (1010.)_2$. Mas, se for considerado o fator de escala $2^{N-1} = 8$, o valor de referência passa a ser $V_{R16/8} = 16/8 = 2 = V_{R2}$, enquanto o valor complementar de V_- passa a ser $V_{C2} = V_{C16/8} = (V_{R16/8} - |V_-/8|) = (V_{R2} - |V_-/8|) = (16/8 - 6/8)_{10} = (2.00 - 0.75)_{10} = (1.25)_{10} = (1.010)_2$. Portanto, a menos da semântica quantitativa, ambas as codificações conduzem a uma sintaxe similar, baseada no padrão binário "1010", podendo ser chamadas unicamente de Complemento-a-2 ou *two's Complement*.

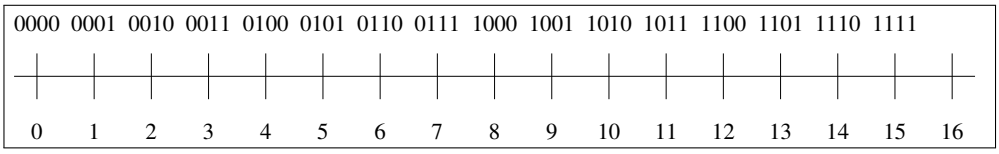


Figura 9.5: Mapeamento decimal-binário para números não negativos e $N = 4$, sem sinal.

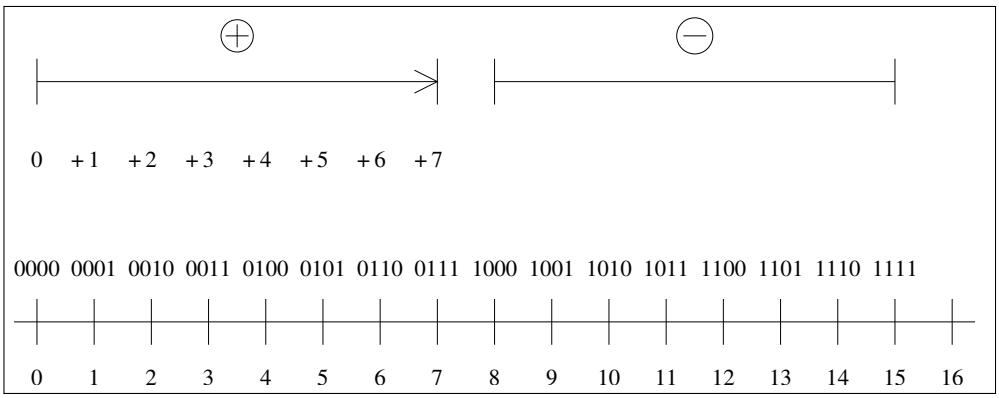


Figura 9.6: Mapeamento decimal-binário para números não negativos e $N = 4$, com sinal.

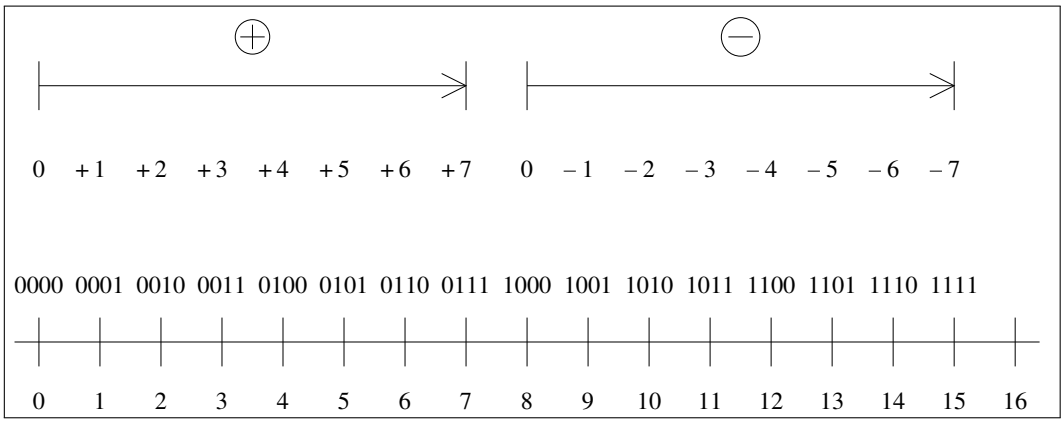


Figura 9.7: Mapeamento decimal-binário para números negativos e $N = 4$, com codificação Sinal-e-Magnitude.

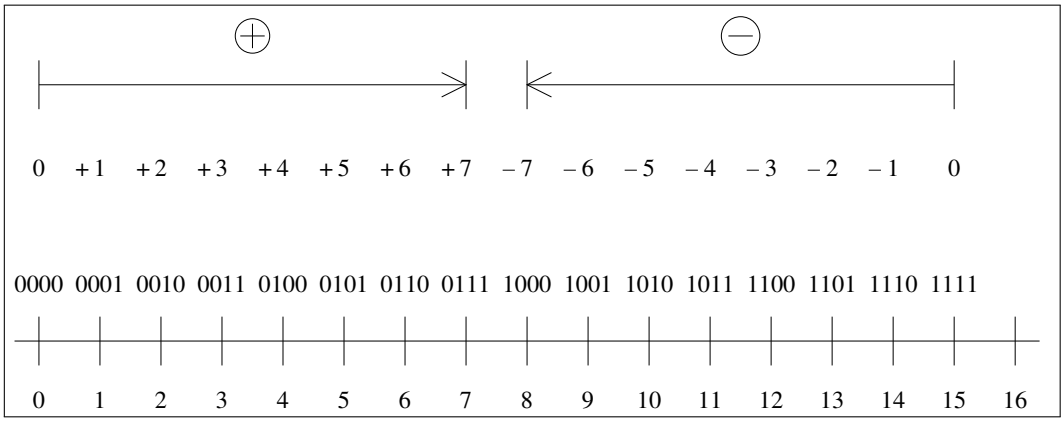


Figura 9.8: Mapeamento decimal-binário para números negativos e $N = 4$, com codificação Complemento-a-1.

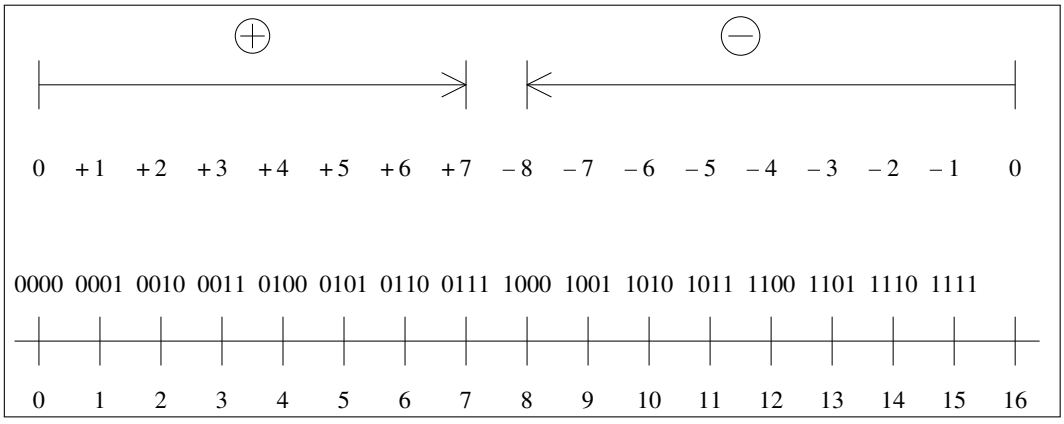


Figura 9.9: Mapeamento decimal-binário para números negativos e $N = 4$, com codificação Complemento-a-2.

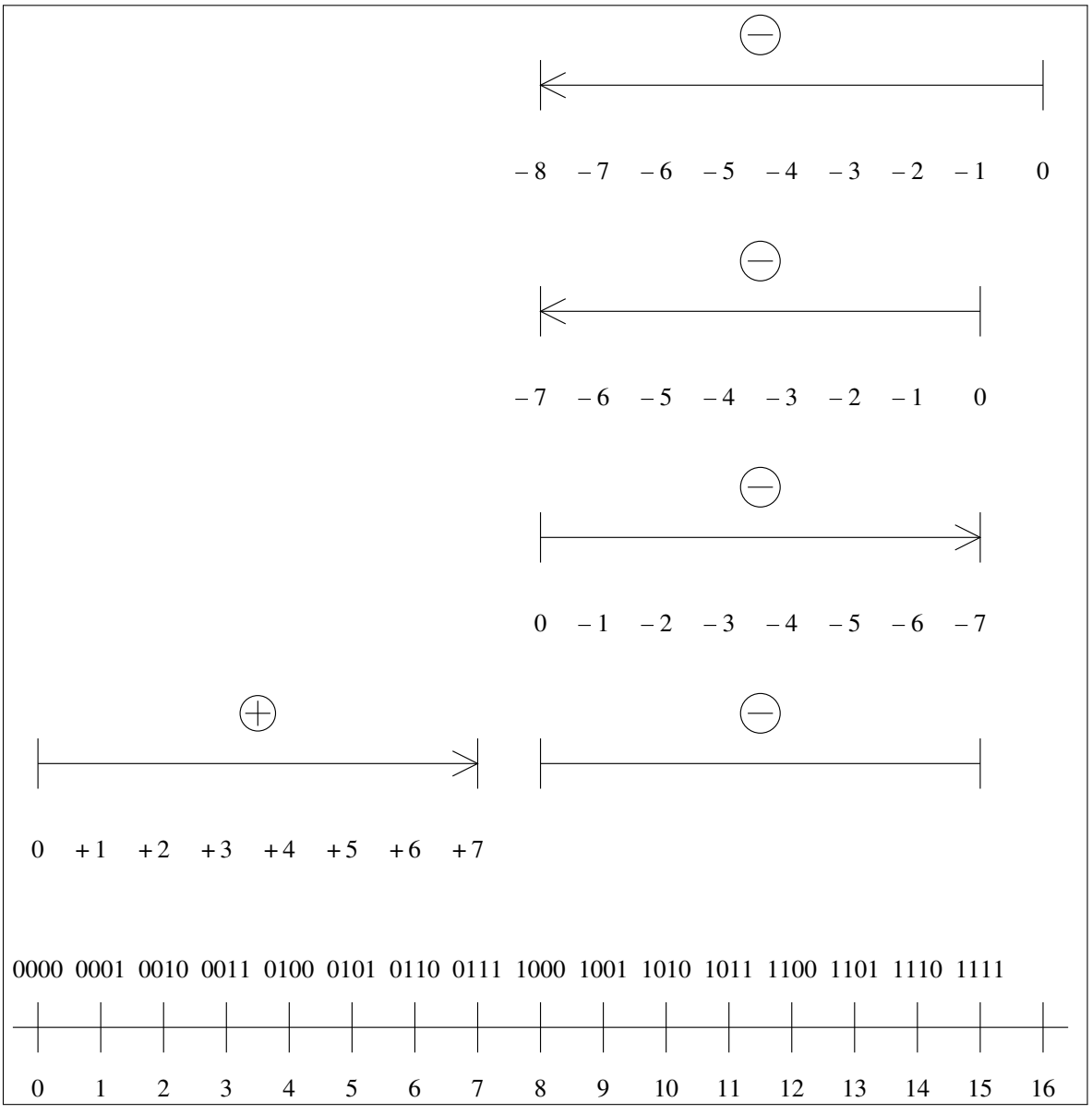


Figura 9.10: Comparação dos mapeamentos decimal-binário para números negativos e $N = 4$.

Codificação Sinal-e-magnitude

- Assim como na representação para uso humano, o dígito de sinal $d_{N-1} = s_{N-1} = 0$ ou $(b-1)$ pode ser interpretado como um operador unário ou como um dígito sem peso numérico, que indica apenas o valor do sinal.
- Os demais dígitos representam um valor numérico positivo.
- A representação possui dois padrões binários para o valor numérico nulo: $+(0)$ e $-(0)$.
- A Equação (9.9) apresenta uma interpretação numérica da representação, para $b = 2$.
- Um exemplo é apresentado na Tabela 9.1, para $b = 2$ e $N = 4$.

$$\begin{aligned}(q_I)_2 &= [0/1\ d_{N-2}\cdots d_2 d_1 d_0]_2 = [s_{N-1} d_{N-2}\cdots d_2 d_1 d_0]_2 \\ &= (-1)^{s_{N-1}} \times \left[(d_{N-2}\times 2^{N-2}) + \cdots + (d_2 \times 2^2) + (d_1 \times 2^1) + (d_0 \times 2^0)\right] \\ &= (-1)^{s_{N-1}} \times \left(\sum_{k=0}^{N-2} d_k 2^k\right) \end{aligned} \tag{9.9}$$

Binário				Decimal	Interpretação
0	1	1	1	7	$(+1)\cdot(7)$
0	1	1	0	6	$(+1)\cdot(6)$
0	1	0	1	5	$(+1)\cdot(5)$
0	1	0	0	4	$(+1)\cdot(4)$
0	0	1	1	3	$(+1)\cdot(3)$
0	0	1	0	2	$(+1)\cdot(2)$
0	0	0	1	1	$(+1)\cdot(1)$
0	0	0	0	0	$(+1)\cdot(0)$
1	0	0	0	0	$(-1)\cdot(0)$
1	0	0	1	−1	$(-1)\cdot(1)$
1	0	1	0	−2	$(-1)\cdot(2)$
1	0	1	1	−3	$(-1)\cdot(3)$
1	1	0	0	−4	$(-1)\cdot(4)$
1	1	0	1	−5	$(-1)\cdot(5)$
1	1	1	0	−6	$(-1)\cdot(6)$
1	1	1	1	−7	$(-1)\cdot(7)$

Tabela 9.1: Tabela de sinal-e-magnitude, para número inteiros, $b = 2$ e $N = 4$.

Codificação Complemento à base diminuída

- Para quantidades positivas, a codificação é sinal-e-magnitude, onde $d_{N-1} = s_{N-1} = 0$.
- Para quantidades negativas, a codificação é $d_{N-1} = s_{N-1} = (b-1)$ e os demais dígitos são recodificados.
- A representação possui dois padrões binários para o valor numérico nulo: $+(0)$ e $-(0)$.
- Interpretação 1:
  - Técnica: um valor positivo adequado é adicionado ao número negativo, de tal forma que o resultado seja positivo e que o dígito de sinal passe de 0 para $(b-1)$.
  - O número negativo é representado como o complemento do seu valor absoluto em relação a um módulo, cujo valor é a menor potência (inteira e positiva) da base que é maior do que a representação do valor absoluto a ser representado, reduzida de uma unidade.
  - Para $(q_{I-})_b$ representada por um vetor de N dígitos: $-|x| \leftrightarrow x_C = \left(b^N - 1\right) - |x|$.
  - As Equações (9.10) e (9.11) apresentam uma interpretação numérica da representação, para $b = 2$.

$$(q_{I-})_2 = -|x| = -\left(\sum_{k=0}^{N-1} d_k 2^k\right) = -\left[d_{N-1}d_{N-2}\cdots d_1 d_0\right]_2 = -\left[0\ d_{N-2}\cdots d_1 d_0\right]_2 \tag{9.10}$$

$$\begin{aligned} x_{C1} &= \left(2^N - 1\right) - |x| = \left(2^N - 1\right) - \left(\sum_{k=0}^{N-1} d_k 2^k\right) \\ &= [1\ 1\cdots 1\ 1]_2 - [0\ d_{N-2}\cdots d_1 d_0]_2 = \left[1\ d'_{N-2}\cdots d'_1 d'_0\right]_2 \end{aligned} \tag{9.11}$$

- Interpretação 2 (para $b = 2$):
  - O dígito de sinal tem peso negativo: $w_{N-1} = [-(2^{N-1} - 1)]$.
  - Os demais dígitos representam um valor numérico positivo que, somado ao valor negativo do dígito de sinal, fornece o valor negativo desejado.
  - As Equações (9.12) e (9.13) apresentam uma interpretação numérica da representação, para $b = 2$.

$$\begin{aligned} (q_{I+})_2 &= [0\ d_{N-2}\cdots d_2 d_1 d_0]_2 \\ &= [s_{N-1}\ d_{N-2}\cdots d_2 d_1 d_0]_2 \\ &= s_{N-1}\cdot[-(2^{N-1} - 1)] + [(d_{N-2}\times 2^{N-2}) + \cdots + (d_2\times 2^2) + (d_1\times 2^1) + (d_0\times 2^0)] \\ &= s_{N-1}\cdot[-(2^{N-1} - 1)] + \left(\sum_{k=0}^{N-2} d_k 2^k\right) \end{aligned} \tag{9.12}$$

$$\begin{aligned} (q_{I-})_2 &= [1\ d'_{N-2}\cdots d'_2 d'_1 d'_0]_2 \\ &= [s'_{N-1}d'_{N-2}\cdots d'_2 d'_1 d'_0]_2 \\ &= s'_{N-1}\cdot[-(2^{N-1} - 1)] + [(d'_{N-2}\times 2^{N-2}) + \cdots + (d'_2\times 2^2) + (d'_1\times 2^1) + (d'_0\times 2^0)] \\ &= s'_{N-1}\cdot[-(2^{N-1} - 1)] + \left(\sum_{k=0}^{N-2} d'_k 2^k\right) \end{aligned} \tag{9.13}$$

- Um exemplo é apresentado na Tabela 9.2, para $b = 2$ e $N = 4$.
- Para a base $b = 2$, pode-se definir o seguinte algoritmo para a conversão entre as representações de quantidades positivas e negativas, em complemento-a-1:
  - Dada uma representação numérica, em complemento-a-1, para se obter sua representação complementar basta que se troque os numerais 0 por 1 e que se troque os numerais 1 por 0.

Binário				Decimal	Interpretação 1	Interpretação 2
0	1	1	1	7	$(0) + (7)$	$(0) + (7)$
0	1	1	0	6	$(0) + (6)$	$(0) + (6)$
0	1	0	1	5	$(0) + (5)$	$(0) + (5)$
0	1	0	0	4	$(0) + (4)$	$(0) + (4)$
0	0	1	1	3	$(0) + (3)$	$(0) + (3)$
0	0	1	0	2	$(0) + (2)$	$(0) + (2)$
0	0	0	1	1	$(0) + (1)$	$(0) + (1)$
0	0	0	0	0	$(0) + (0)$	$(0) + (0)$
1	1	1	1	0	$(15) - (0)$	$(-7) + (7)$
1	1	1	0	-1	$(15) - (1)$	$(-7) + (6)$
1	1	0	1	-2	$(15) - (2)$	$(-7) + (5)$
1	1	0	0	-3	$(15) - (3)$	$(-7) + (4)$
1	0	1	1	-4	$(15) - (4)$	$(-7) + (3)$
1	0	1	0	-5	$(15) - (5)$	$(-7) + (2)$
1	0	0	1	-6	$(15) - (6)$	$(-7) + (1)$
1	0	0	0	-7	$(15) - (7)$	$(-7) + (0)$

Tabela 9.2: Tabela de complemento-a-1, para número inteiros, $b = 2$ e $N = 4$.

Codificação Complemento à base

- Para quantidades positivas, a codificação é sinal-e-magnitude, onde $d_{N-1} = s_{N-1} = 0$.
- Para quantidades negativas, a codificação é $d_{N-1} = s_{N-1} = (b-1)$ e os demais dígitos são recodificados.
- A representação possui apenas um padrão binário para o valor numérico nulo: 0.
- Interpretação 1:
  - Técnica: um valor positivo adequado é adicionado ao número negativo, de tal forma que o resultado seja positivo e que o dígito de sinal passe de 0 para $(b-1)$.
  - O número negativo é representado como o complemento do seu valor absoluto em relação a um módulo, cujo valor é a menor potência (inteira e positiva) da base que é maior do que a representação do valor absoluto a ser representado.
  - Para $(q_{I-})_b$ representada por um vetor de N dígitos: $-|x| \leftrightarrow x_C = b^N - |x|$.
  - As Equações (9.14) e (9.15) apresentam uma interpretação numérica da representação, para $b = 2$.

$$(q_{I-})_2 = -|x| = -\left(\sum_{k=0}^{N-1} d_k 2^k\right) = -\left[d_{N-1}d_{N-2}\cdots d_1 d_0\right]_2 = -\left[0\ d_{N-2}\cdots d_1 d_0\right]_2 \tag{9.14}$$

$$x_{C2} = 2^N - |x| = 2^N - \left(\sum_{k=0}^{N-1} d_k 2^k\right) = [1\ 0\ 0\cdots 0\ 0]_2 \ - \ [0\ d_{N-2}\cdots d_1 d_0]_2 = \left[1\ d'_{N-2}\cdots d'_1 d'_0\right]_2 \tag{9.15}$$

- Interpretação 2 (para $b = 2$):
  - O dígito de sinal tem peso negativo: $w_{N-1} = (-2^{N-1})$.
  - Os demais dígitos representam um valor numérico positivo que, somado ao valor negativo do dígito de sinal, fornece o valor negativo desejado.
  - As Equações (9.16) e (9.17) apresentam uma interpretação numérica da representação, para $b = 2$.

$$\begin{aligned}(q_{I+})_2 &= [0\ d_{N-2}\cdots d_2 d_1 d_0]_2 \\ &= [s_{N-1}\ d_{N-2}\cdots d_2 d_1 d_0]_2 \\ &= s_{N-1}\cdot[-(2^{N-1})] + [(d_{N-2}\times 2^{N-2}) + \cdots + (d_2\times 2^2) + (d_1\times 2^1) + (d_0\times 2^0)] \\ &= s_{N-1}\cdot[-(2^{N-1})] + \left(\sum_{k=0}^{N-2} d_k 2^k\right)\end{aligned} \tag{9.16}$$

$$\begin{aligned}(q_{I-})_2 &= [1\ d'_{N-2}\cdots d'_2 d'_1 d'_0]_2 \\ &= [s'_{N-1}d'_{N-2}\cdots d'_2 d'_1 d'_0]_2 \\ &= s'_{N-1}\cdot[-(2^{N-1})] + [(d'_{N-2}\times 2^{N-2}) + \cdots + (d'_2\times 2^2) + (d'_1\times 2^1) + (d'_0\times 2^0)] \\ &= s'_{N-1}\cdot[-(2^{N-1}] + \left(\sum_{k=0}^{N-2} d'_k 2^k\right)\end{aligned} \tag{9.17}$$

- Um exemplo é apresentado na Tabela 9.3, para $b = 2$ e $N = 4$.

- Para a base $b = 2$, podem-se definir os seguintes algoritmos para a conversão entre as representações de quantidades positivas e negativas, em complemento-a-2:
  - Algoritmo 1: Dada uma representação numérica, em complemento-a-2, para obter-se a sua representação complementar basta: i) que se troquem os numerais 0 por 1 e que se troquem os numerais 1 por 0 (complemento-a-1) e, em seguida, ii) que seja adicionado o valor 1 ao dígito menos significativo (*Least Significant Bit* ou LSB).
  - Algoritmo 2: Dada uma representação numérica, em complemento-a-2, para obter-se a sua representação complementar deve-se realizar uma busca a partir do dígito menos significativo (LSB). Durante a busca, os numerais 0 menos significativos não serão modificados até que seja encontrado o primeiro numeral 1, que também não será modificado. A partir deste ponto, basta que se troquem os numerais 0 por 1 e que se troquem os numerais 1 por 0.
  - Uma vez que ele trabalha com adição, o primeiro algoritmo é mais adequado para ambientes onde já se dispõe de um circuito somador.
  - Por sua vez, dado que ele envolve um processo de varredura, o segundo algoritmo é mais recomendado quando se deseja implementar um simples bloco funcional para realizar a complementação.
  - Nessa codificação, o valor mais negativo representável é dado por (-2^{N-1}). Por sua vez, o valor mais positivo representável é dado por $(2^{N-1} - 1)$. Assim, dado que o valor mais negativo não possui um valor positivo equivalente, a sua conversão para um valor positivo deve ser tratada como uma situação de "overflow". Nesse caso, tal situação deve ser detectada e, em seguida, o valor positivo deve ser saturado no maior valor representável.

Binário				Decimal	Interpretação 1	Interpretação 2
0	1	1	1	7	$(0) + (7)$	$(0) + (7)$
0	1	1	0	6	$(0) + (6)$	$(0) + (6)$
0	1	0	1	5	$(0) + (5)$	$(0) + (5)$
0	1	0	0	4	$(0) + (4)$	$(0) + (4)$
0	0	1	1	3	$(0) + (3)$	$(0) + (3)$
0	0	1	0	2	$(0) + (2)$	$(0) + (2)$
0	0	0	1	1	$(0) + (1)$	$(0) + (1)$
0	0	0	0	0	$(0) + (0)$	$(0) + (0)$
1	1	1	1	-1	$(16) - (1)$	$(-8) + (7)$
1	1	1	0	-2	$(16) - (2)$	$(-8) + (6)$
1	1	0	1	-3	$(16) - (3)$	$(-8) + (5)$
1	1	0	0	-4	$(16) - (4)$	$(-8) + (4)$
1	0	1	1	-5	$(16) - (5)$	$(-8) + (3)$
1	0	1	0	-6	$(16) - (6)$	$(-8) + (2)$
1	0	0	1	-7	$(16) - (7)$	$(-8) + (1)$
1	0	0	0	-8	$(16) - (8)$	$(-8) + (0)$

Tabela 9.3: Tabela de complemento-a-2, para número inteiros, $b = 2$ e $N = 4$.

9.2.4 Representação de números fracionários negativos

- O equacionamento utilizado para a representação de números inteiros negativos pode ser aproveitado para números negativos puramente fracionários.
- Uma quantidade puramente fracionária x_F pode ser obtida através da multiplicação de uma quantidade inteira x_I por um fator de escala FE adequado ($x_F = FE \cdot x_I$).
- Assim, para aproveitar o equacionamento anterior, basta utilizar um escalamento.
- Cabe ressaltar que, em circuitos digitais que manipulam números binários e que utilizam uma posição fixa para o separador das partes inteira e fracionária (aritmética de ponto fixo), é comum que se interprete todas as grandezas como números puramente fracionários $0 \leq |x_F| < 1$, codificados em complemento-a-2. Nesse caso, o separador encontra-se (virtualmente) entre o dígito de sinal (s_{N-1}) e os demais $N-1$ dígitos que representam a quantidade numérica. Partindo-se das Equações (9.14) e (9.15), não é difícil demonstrar a seguinte equivalência: $-|x_F| \leftrightarrow (x_F)_{C2} = 2 - |x_F|$. Finalmente, partindo-se das Equações (9.16) e (9.17), não é difícil demonstrar a seguinte notação: $(x_F)_{C2} = -s_{N-1} + \left(\sum_{k=1}^{N-1} d_{-k} 2^{-k}\right)$.
- A título de exemplo, as Tabelas 9.1 a 9.3, que representam números inteiros, são transformadas nas Tabelas 9.4 a 9.6, para números puramente fracionários, através do fator de escala $FE = 2^{-(N-1)} = 2^{-3} = 8^{-1}$.
- Comparando-se os conteúdos das Tabelas 9.1 a 9.6, destaca-se mais uma vez o fato de que um mesmo padrão de dígitos pode ser interpretado de diversas formas diferentes, dependendo do sistema de numeração, da forma de codificação e da posição do separador fracionário utilizados.

Binário				Decimal	Interpretação
0	1	1	1	0.875	$(+1) \cdot (0.875)$
0	1	1	0	0.750	$(+1) \cdot (0.750)$
0	1	0	1	0.625	$(+1) \cdot (0.625)$
0	1	0	0	0.500	$(+1) \cdot (0.500)$
0	0	1	1	0.375	$(+1) \cdot (0.375)$
0	0	1	0	0.250	$(+1) \cdot (0.250)$
0	0	0	1	0.125	$(+1) \cdot (0.125)$
0	0	0	0	0.000	$(+1) \cdot (0.000)$
1	0	0	0	0.000	$(-1) \cdot (0.000)$
1	0	0	1	−0.125	$(-1) \cdot (0.125)$
1	0	1	0	−0.250	$(-1) \cdot (0.250)$
1	0	1	1	−0.375	$(-1) \cdot (0.375)$
1	1	0	0	−0.500	$(-1) \cdot (0.500)$
1	1	0	1	−0.625	$(-1) \cdot (0.625)$
1	1	1	0	−0.750	$(-1) \cdot (0.750)$
1	1	1	1	−0.875	$(-1) \cdot (0.875)$

Tabela 9.4: Tabela de sinal-e-magnitude, para números puramente fracionários, $b = 2$ e $N = 4$.

Binário				Decimal	Interpretação 1	Interpretação 2
0	1	1	1	0.875	(0.000) + (0.875)	(0.000) + (0.875)
0	1	1	0	0.750	(0.000) + (0.750)	(0.000) + (0.750)
0	1	0	1	0.625	(0.000) + (0.625)	(0.000) + (0.625)
0	1	0	0	0.500	(0.000) + (0.500)	(0.000) + (0.500)
0	0	1	1	0.375	(0.000) + (0.375)	(0.000) + (0.375)
0	0	1	0	0.250	(0.000) + (0.250)	(0.000) + (0.250)
0	0	0	1	0.125	(0.000) + (0.125)	(0.000) + (0.125)
0	0	0	0	0.000	(0.000) + (0.000)	(0.000) + (0.000)
1	1	1	1	0.000	(1.875) − (0.000)	(−0.875) + (0.875)
1	1	1	0	−0.125	(1.875) − (0.125)	(−0.875) + (0.750)
1	1	0	1	−0.250	(1.875) − (0.250)	(−0.875) + (0.625)
1	1	0	0	−0.375	(1.875) − (0.375)	(−0.875) + (0.500)
1	0	1	1	−0.500	(1.875) − (0.500)	(−0.875) + (0.375)
1	0	1	0	−0.625	(1.875) − (0.625)	(−0.875) + (0.250)
1	0	0	1	−0.750	(1.875) − (0.750)	(−0.875) + (0.125)
1	0	0	0	−0.875	(1.875) − (0.875)	(−0.875) + (0.000)

Tabela 9.5: Tabela de complemento-a-1, para números puramente fracionários, $b = 2$ e $N = 4$.

Binário				Decimal	Interpretação 1	Interpretação 2
0	1	1	1	0.875	(0) + (0.875)	(0) + (0.875)
0	1	1	0	0.750	(0) + (0.750)	(0) + (0.750)
0	1	0	1	0.625	(0) + (0.625)	(0) + (0.625)
0	1	0	0	0.500	(0) + (0.500)	(0) + (0.500)
0	0	1	1	0.375	(0) + (0.375)	(0) + (0.375)
0	0	1	0	0.250	(0) + (0.250)	(0) + (0.250)
0	0	0	1	0.125	(0) + (0.125)	(0) + (0.125)
0	0	0	0	0.000	(0) + (0.000)	(0) + (0.000)
1	1	1	1	−0.125	(2) − (0.125)	(−1) + (0.875)
1	1	1	0	−0.250	(2) − (0.250)	(−1) + (0.750)
1	1	0	1	−0.375	(2) − (0.375)	(−1) + (0.625)
1	1	0	0	−0.500	(2) − (0.500)	(−1) + (0.500)
1	0	1	1	−0.625	(2) − (0.625)	(−1) + (0.375)
1	0	1	0	−0.750	(2) − (0.750)	(−1) + (0.250)
1	0	0	1	−0.875	(2) − (0.875)	(−1) + (0.125)
1	0	0	0	−1.000	(2) − (1.000)	(−1) + (0.000)

Tabela 9.6: Tabela de complemento-a-2, para números puramente fracionários, $b = 2$ e $N = 4$.

9.2.5 Conversão entre bases

A seguir, são consideradas as conversões de números não negativos (inteiros e puramente fracionários).

Números não negativos e inteiros

A conversão da base s para a base t é o processo onde, conhecendo-se os dígitos d'_i da Equação (9.18), deseja-se encontrar os dígitos d_i da Equação (9.19). Considerando-se todas as quantidades expressas na base s, podem-se definir as relações expressas na Equação (9.20). Assim, para que se encontrem os dígitos d_i, basta que se realizem divisões sucessivas do dividendo N_i pelo divisor t, gerando-se o quociente N_{i+1} e o resto d_i, e que, no final, os restos sejam posicionados na ordem adequada. Uma vez que o número de dígitos d_i é finito, é garantido que o algoritmo terá um número finito de passos.

$$(q)_s = [d'_J \cdots d'_1 d'_0]_s = (N_0)_s \ . \tag{9.18}$$

$$(q)_t = [d_K \cdots d_1 d_0]_t \ . \tag{9.19}$$

$$\begin{aligned}
N_0 &= \left(d_K \times t^K + \cdots + d_2 \times t^2 + d_1 \times t^1 + d_0 \times t^0\right) \\
&= \left(d_K \times t^{K-1} + \cdots + d_2 \times t^1 + d_1 \times t^0\right) \times t + \left(d_0 \times t^0\right) \\
&= N_1 \times t + d_0 \\
\\
N_1 &= \left(d_K \times t^{K-1} + \cdots + d_2 \times t^1 + d_1 \times t^0\right) \\
&= \left(d_K \times t^{K-2} + \cdots + d_2 \times t^0\right) \times t + \left(d_1 \times t^0\right) \\
&= N_2 \times t + d_1 \\
\\
&\vdots \\
\\
N_{K-1} &= \left(d_K \times t^1 + d_{K-1} \times t^0\right) \\
&= (d_K) \times t + \left(d_{K-1} \times t^0\right) \\
&= N_K \times t + d_{K-1} \\
\\
N_K &= d_K \ .
\end{aligned} \tag{9.20}$$

Números não negativos e puramente fracionários

A conversão da base s para a base t é o processo onde, conhecendo-se os dígitos d'_i da Equação (9.21), deseja-se encontrar os dígitos d_i da Equação (9.22). Considerando-se todas as quantidades expressas na base s, podem-se definir as relações expressas na Equação (9.23). Assim, para que se encontrem os dígitos d_i, basta que se realizem multiplicações sucessivas do multiplicando puramente fracionário N_i pelo multiplicador t, gerando-se o resultado N_{i-1}, que contém d_i como parte inteira, e que, no final, os restos sejam posicionados na ordem adequada. Uma vez que não se pode garantir que o número de dígitos d_i será finito, deve-se estabelecer um número máximo de passos para garantir que o algoritmo terá um término.

$$(q)_s = [d'_{-1}d'_{-2}\cdots d'_{-J}]_s = (N_{-1})_s \ . \tag{9.21}$$

$$(q)_t = [d_{-1}d_{-2}\cdots d_{-K}]_t \ . \tag{9.22}$$

$$\begin{aligned}
N_{-1}\times t &= \left(d_{-1}\times t^{-1} + d_{-2}\times t^{-2} + d_{-3}\times t^{-3} + \cdots + d_{-K}\times t^{-K}\right)\times t \\
&= \left(d_{-1}\times t^{0}\right) + \left(d_{-2}\times t^{-1} + d_{-3}\times t^{-2} + \cdots + d_{K}\times t^{-K+1}\right) \\
&= d_{-1} + N_{-2} \\
\\
N_{-2}\times t &= \left(d_{-2}\times t^{-1} + d_{-3}\times t^{-2} + \cdots + d_{-K}\times t^{-K+1}\right)\times t \\
&= \left(d_{-2}\times t^{0}\right) + \left(d_{-3}\times t^{-1} + \cdots + d_{K}\times t^{-K+2}\right) \\
&= d_{-2} + N_{-3} \\
\\
&\vdots \\
\\
N_{-K+1}\times t &= \left(d_{-K+1}\times t^{-1} + d_{-K}\times t^{-2}\right)\times t \\
&= \left(d_{-K+1}\times t^{0}\right) + \left(d_{-K}\times t^{-1}\right) \\
&= d_{-K+1} + N_{-K} \\
\\
N_{-K}\times t &= \left(d_{-K}\times t^{-1}\right)\times t \\
&= \left(d_{-K}\times t^{0}\right) \\
&= d_{-K}
\end{aligned} \tag{9.23}$$

9.2.6 Bases mais comuns em circuitos digitais

A notação em base $b = 2$ é a mais adequada para lidar com a implementação de circuitos digitais baseados em sistemas binários. Porém, dada uma base de valor reduzido, a representação terá um número elevado de dígitos. Para o uso humano, quanto maior é o número de dígitos, mais trabalhoso é a sua interpretação e a sua manipulação. Assim, a fim de simplificar a representação, duas bases são muito utilizadas: octal e hexadecimal. A base octal emprega $b = 8$ e dígitos $d_i \in \boldsymbol{S} = \{0, 1, 2, \cdots, 7\}$. Por sua vez, a base hexadecimal emprega $b = 16$ e dígitos $d_i \in \boldsymbol{S} = \{0, 1, 2, \cdots, 9, A, B, ..., F\}$. Supondo-se números não negativos e inteiros, as Equações (9.24) a (9.26) ilustram as notações nas três bases.

$$(q_I)_2 = (d_J \times 2^J) + \cdots + (d_2 \times 2^2) + (d_1 \times 2^1) + (d_0 \times 2^0) \ . \tag{9.24}$$

$$(q_I)_8 = (d'_K \times 8^K) + \cdots + (d'_2 \times 8^2) + (d'_1 \times 8^1) + (d'_0 \times 8^0) \ . \tag{9.25}$$

$$(q_I)_{16} = (d''_L \times 16^L) + \cdots + (d''_2 \times 16^2) + (d''_1 \times 16^1) + (d''_0 \times 16^0) \ . \tag{9.26}$$

As bases binária, octal e hexadecimal são comumente utilizadas em conjunto, devido à facilidade de conversão entre as três bases. As Equações (9.27) a (9.31) ilustram a relação entre as bases binária e octal.

$$\begin{aligned}
(q_I)_2 &= (d_J \times 2^J) + (d_{J-1} \times 2^{J-1}) + (d_{J-2} \times 2^{J-2}) + \cdots + \\
&\quad (d_5 \times 2^5) + (d_4 \times 2^4) + (d_3 \times 2^3) + \\
&\quad (d_2 \times 2^2) + (d_1 \times 2^1) + (d_0 \times 2^0) \\
&= \Big[(d_J \times 2^2) + (d_{J-1} \times 2^1) + (d_{J-2} \times 2^0)\Big] \times 2^{J-2} + \cdots + \\
&\quad \Big[(d_5 \times 2^2) + (d_4 \times 2^1) + (d_3 \times 2^0)\Big] \times 2^3 + \\
&\quad \Big[(d_2 \times 2^2) + (d_1 \times 2^1) + (d_0 \times 2^0)\Big] \times 2^0 \\
&= (d'_K \times 8^K) + \cdots + (d'_1 \times 8^1) + (d'_0 \times 8^0) \\
&= (q_I)_8 \ .
\end{aligned} \tag{9.27}$$

$$J - 2 = 3K \ . \tag{9.28}$$

$$[d_2 d_1 d_0]_2 = [d'_0]_8 \ . \tag{9.29}$$

$$[d_5 d_4 d_3]_2 = [d'_1]_8 \ . \tag{9.30}$$

$$[d_J d_{J-1} d_{J-2}]_2 = [d'_K]_8 \ . \tag{9.31}$$

As Equações (9.32) a (9.36) ilustram a relação entre as bases binária e hexadecimal.

$$\begin{aligned}
(q_I)_2 &= (d_J \times 2^J) + (d_{J-1} \times 2^{J-1}) + (d_{J-2} \times 2^{J-2}) + (d_{J-3} \times 2^{J-3}) + \cdots + \\
&\quad (d_7 \times 2^7) + (d_6 \times 2^6) + (d_5 \times 2^5) + (d_4 \times 2^4) + \\
&\quad (d_3 \times 2^3) + (d_2 \times 2^2) + (d_1 \times 2^1) + (d_0 \times 2^0) \\
&= \Big[(d_J \times 2^3) + (d_{J-1} \times 2^2) + (d_{J-2} \times 2^1) + (d_{J-3} \times 2^0)\Big] \times 2^{J-3} + \cdots + \\
&\quad \Big[(d_7 \times 2^3) + (d_6 \times 2^2) + (d_5 \times 2^1) + (d_4 \times 2^0)\Big] \times 2^4 + \\
&\quad \Big[(d_3 \times 2^3) + (d_2 \times 2^2) + (d_1 \times 2^1) + (d_0 \times 2^0)\Big] \times 2^0 \\
&= (d''_L \times 16^L) + \cdots + (d''_1 \times 16^1) + (d''_0 \times 16^0) \\
&= (q_I)_{16} \ . \qquad (9.32)
\end{aligned}$$

$$J - 3 = 4L \ . \qquad (9.33)$$

$$[d_3 d_2 d_1 d_0]_2 = [d''_0]_{16} \ . \qquad (9.34)$$

$$[d_7 d_6 d_5 d_4]_2 = [d''_1]_{16} \ . \qquad (9.35)$$

$$[d_J d_{J-1} d_{J-2} d_{J-3}]_2 = [d''_L]_{16} \ . \qquad (9.36)$$

As Equações (9.37) a (9.41) ilustram a relação entre as bases octal e hexadecimal.

$$\begin{aligned}
(q_I)_8 &= (d'_K \times 8^K) + (d'_{K-1} \times 8^{K-1}) + \cdots + \\
&\quad (d'_3 \times 8^3) + (d'_2 \times 8^2) + \\
&\quad (d'_1 \times 8^1) + (d'_0 \times 8^0) \\
&= \Big[(d'_K \times 8^1) + (d'_{K-1} \times 8^0)\Big] \times 8^{K-1} + \cdots + \\
&\quad \Big[(d'_3 \times 8^1) + (d'_2 \times 8^0) +\Big] \times 8^2 + \\
&\quad \Big[(d'_1 \times 8^1) + (d'_0 \times 8^0)\Big] \times 8^0 \\
&= (d''_L \times 16^L) + \cdots + (d''_1 \times 16^1) + (d''_0 \times 16^0) \\
&= (q_I)_{16} \ . \qquad (9.37)
\end{aligned}$$

$$K - 1 = 2L \ . \qquad (9.38)$$

$$[d'_1 d'_0]_8 = [d''_0]_{16} \ . \qquad (9.39)$$

$$[d'_3 d'_2]_8 = [d''_1]_{16} \ . \qquad (9.40)$$

$$[d'_K d'_{K-1}]_2 = [d''_L]_{16} \ . \qquad (9.41)$$

Embora todas as equações tenham sido definidas para números não negativos e inteiros, não é difícil mostrar que as relações se mantêm para números não negativos e fracionários.

9.3 Quantização

9.3.1 Conceitos básicos

Quantizar significa representar, através de uma aproximação, uma faixa contínua de valores originais por uma faixa discreta de valores correspondentes. Na representação discreta de valores contínuos, o intervalo mínimo de representação é denominado de resolução.

Todo sistema físico de medição possui um intervalo mínimo de medida (resolução da medida). Por outro lado, todo sistema de numeração possui um intervalo mínimo de representação das quantidades numéricas (resolução da representação). Logo, toda medida, bem como a sua respectiva representação, possuem um grau intrínseco de aproximação.

Conseqüentemente, toda representação discreta de valores contínuos apresenta um erro, intrinsicamente ligado à sua medida e/ou à sua representação, denominado erro de quantização.

9.3.2 SNPC: resolução, base e quantidade de dígitos

A resolução de um SNPC está diretamente ligada com a sua base e com a quantidade de dígitos utilizados na representação numérica.

Uma medida da resolução pode ser obtida por meio da quantidade de níveis representáveis. Pensando-se em uma representação de valores inteiros, percebe-se que, a cada dígito adicionado na representação, a quantidade de níveis é multiplicada pelo valor da base. A Tabela 9.7 apresenta alguns valores de níveis em um SNPC, em função do número de dígitos empregados, para as bases $b = 2$ e $b = 10$.

Número de dígitos	Padrão inteiro	Níveis (b = 2)	Níveis (b = 10)
1	d_0	$2^1 = 2$	$10^1 = 10$
2	d_1d_0	$2^2 = 4$	$10^2 = 100$
3	$d_2d_1d_0$	$2^3 = 8$	$10^3 = 1000$
4	$d_3d_2d_1d_0$	$2^4 = 16$	$10^4 = 10000$
5	$d_4d_3d_2d_1d_0$	$2^5 = 32$	$10^5 = 100000$
6	$d_5d_4d_3d_2d_1d_0$	$2^6 = 64$	$10^6 = 1000000$

Tabela 9.7: Alguns valores de níveis em um SNPC, em função do número de dígitos empregados, para as bases $b = 2$ e $b = 10$.

Por sua vez, a resolução numérica também pode ser empregada como forma de medida da resolução. Pensando-se em uma representação de valores fracionários, percebe-se que, a cada dígito adicionado na representação, a resolução numérica é multiplicada pelo valor da base. A Tabela 9.8 apresenta alguns valores de resolução numérica em um SNPC, em função do número de dígitos empregados, para as bases $b = 2$ e $b = 10$.

A partir dessas duas abordagens, pode-se concluir que, quanto maior for o valor da base, maior será o aumento da resolução, para cada dígito adicionado na representação.

Número de dígitos	Padrão fracionário	Resolução (b = 2)	Resolução (b = 10)
1	$.d_{-1}$	$2^{-1} = .5$	$10^{-1} = .1$
2	$.0d_{-2}$	$2^{-2} = .25$	$10^{-2} = .01$
3	$.00d_{-3}$	$2^{-3} = .125$	$10^{-3} = .001$
4	$.000d_{-4}$	$2^{-4} = .0625$	$10^{-4} = .0001$
5	$.0000d_{-5}$	$2^{-5} = .03125$	$10^{-5} = .00001$
6	$.00000d_{-6}$	$2^{-6} = .015625$	$10^{-6} = .000001$

Tabela 9.8: Alguns valores de resolução numérica em um SNPC, em função do número de dígitos empregados, para as bases $b = 2$ e $b = 10$.

Para uma determinada base, a única forma de se aumentar a resolução é aumentar o número de dígitos utilizados na representação. Porém, como já observado acima, para bases de pequeno valor o aumento da resolução com o aumento de dígitos também é pequeno. Com isso, pode ser necessária uma quantidade de dígitos indesejada ou ainda inviável. A Tabela 9.9 mostra o efeito do pequeno aumento da resolução numérica em um SNPC, em função do número de dígitos empregados, para a base $b = 2$.

Número de dígitos	Padrão fracionário	Resolução (b = 2)
1	$.d_{-1}$	$2^{-1} = .5$
2	$.0d_{-2}$	$2^{-2} = .25$
3	$.00d_{-3}$	$2^{-3} = .125$
4	$.000d_{-4}$	$2^{-4} = .0625$
5	$.0000d_{-5}$	$2^{-5} = .03125$
6	$.00000d_{-6}$	$2^{-6} = .015625$
7	$.000000d_{-7}$	$2^{-7} = .0078125$
8	$.0000000d_{-8}$	$2^{-8} = .00390625$
9	$.00000000d_{-9}$	$2^{-9} = .001953125$
10	$.000000000d_{-10}$	$2^{-10} = .0009765625$

Tabela 9.9: Efeito do pequeno aumento da resolução numérica em um SNPC, em função do número de dígitos empregados, para a base $b = 2$.

A título de exemplo, os números $(.93750)_{10} = (.11110)_2$ e $(.96875)_{10} = (.11111)_2$ podem ser representados com 5 *bits*. Porém, para que se represente o número $(.94200)_{10}$ são necessários 53 *bits*. Alternativamente, pode-se representá-lo das seguintes formas:

- Com 5 *bits*, onde $(.94200)_{10} \approx (.11110)_2$, gerando um erro relativo de $e_r = .48$ %.
- Com 8 *bits*, onde $(.94200)_{10} \approx (.11110001)_2$, gerando um erro relativo de $e_r = .06$ %.
- Com 11 *bits*, onde $(.94200)_{10} \approx (.11110001001)_2$, gerando um erro relativo de $e_r = .01$ %.

Para aumentar a resolução sem alterar a base e sem aumentar excessivamente a quantidade de dígitos empregados na representação, é comum que se abandone a codificação em ponto fixo. A alternativa mais utilizada é a codificação denominada de ponto flutuante (*floating-point*), definido pelo padrão IEEE 754.

9.3.3 Classificações

Dependendo dos parâmetros considerados, a quantização pode assumir diversas clasificações.

Quanto à regularidade da discretização efetuada, a quantização pode ser classificada como: uniforme e não uniforme. Na quantização uniforme é utilizado um intervalo de discretização único. Na quantização não uniforme são empregados diversos intervalos de discretização diferentes.

Quanto à aproximação adotada para o valor numérico, podem-se destacar três tipos de quantização: truncamento, arredondamento e truncamento em magnitude. O truncamento assume o simples abandono dos dígitos menos significativos. Assim sendo, não se pode garantir que o valor final seja mais próximo do valor original. Além disso, dependendo do código utilizado para representar a quantidade numérica, o módulo do valor original pode diminuir ou aumentar. No arredondamento, é realizada uma análise dos dígitos menos significativos, de forma que o valor final seja mais próximo do valor original. Em alguns sistemas digitais, é desejado que o módulo dos valores quantizados nunca seja aumentado. Dessa forma, realiza-se o denominado truncamento em magnitude. Para alguns códigos, isso significa o simples truncamento do valor original. Para outros, deve-se efetuar uma análise do valor original, de forma a garantir que não ocorra um aumento no seu módulo.

9.4 Códigos numéricos

9.4.1 Introdução

O uso de variáveis e valores booleanos para lidar (representar, armazenar, processar ou transmitir) com itens multivalorados é comumente chamado de codificação (*coding* ou *encoding*). A cada um dos diferentes valores envolvidos é atribuída uma combinação particular de valores booleanos. Os padrões booleanos utilizados são denominados de palavras do código (*code words*) [[Rhy73]]. Alguns códigos numéricos são apresentados a seguir.

9.4.2 Códigos numéricos comuns

Para representar os números naturais, comumente são empregados os seguintes códigos numéricos: decimal, binário, octal e hexadecimal. Em essência, todos esses códigos são representações dos números naturais em um SNPC, com uma determinada base. A relação entre tais códigos é ilustrada na Tabela 9.10, para um código binário com 4 *bits*.

O código binário também pode receber as seguintes denominações: binário posicional, binário seqüencial, binário convencional, binário comum, binário simples e binário puro.

9.4.3 Outros códigos numéricos

Alguns outros códigos numéricos para números naturais são os seguintes:

- Gray.
- *One-hot*.
- Johnson.

A relação de tais códigos com os códigos decimal e binário é ilustrada na Tabela 9.11, para um código binário com 4 *bits*.

Decimal	Binário	Octal	Hexadecimal
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Tabela 9.10: Relação entre os códigos numéricos decimal, binário, octal e hexadecimal, para um código binário com 4 *bits*.

Decimal	Binário	Gray	*One-hot*	Johnson
00	0000	0000	0000000000000001	00000000
01	0001	0001	0000000000000010	10000000
02	0010	0011	0000000000000100	11000000
03	0011	0010	0000000000001000	11100000
04	0100	0110	0000000000010000	11110000
05	0101	0111	0000000000100000	11111000
06	0110	0101	0000000001000000	11111100
07	0111	0100	0000000010000000	11111110
08	1000	1100	0000000100000000	11111111
09	1001	1101	0000001000000000	01111111
10	1010	1111	0000010000000000	00111111
11	1011	1110	0000100000000000	00011111
12	1100	1010	0001000000000000	00001111
13	1101	1011	0010000000000000	00000111
14	1110	1001	0100000000000000	00000011
15	1111	1000	1000000000000000	00000001

Tabela 9.11: Relação entre os códigos numéricos decimal, binário, Gray, *One-hot* e Johnson, para um código binário com 4 *bits*.

9.4.4 Códigos BCD

Na codificação de números decimais, é comum que se utilizem padrões binários para representar separadamente cada dígito decimal. Inúmeras representações podem ser propostas e o conjunto de todas elas é globalmente denominado de BCD (*Binary-Coded Decimal* ou *Boolean-Coded Decimal*).

Alguns exemplos de códigos BCD muito empregados na prática são os seguintes:

- 8421.
- *Excess-3* (XS3).
- Mid-Gray-4 (parte central do código Gray de 4 *bits*).

Os códigos 8421 e XS3 são muito usados em aritmética decimal. O código Mid-Gray-4 é largamente utilizado em sensores de posição. A relação de tais códigos com os dígitos decimais é ilustrada na Tabela 9.12.

Decimal	8421	*Excess-3*	Mid-Gray-4
0	0000	0011	0010
1	0001	0100	0110
2	0010	0101	0111
3	0011	0110	0101
4	0100	0111	0100
5	0101	1000	1100
6	0110	1001	1101
7	0111	1010	1111
8	1000	1011	1110
9	1001	1100	1010

Tabela 9.12: Relação entre os dígitos decimais e os códigos BCD 8421, *Excess-3* e Mid-Gray-4.

Os códigos BCD não possuem, necessariamente, uma representação única. A Tabela 9.13 apresenta diferentes versões do código numérico BCD 631(-1).

Decimal	631(-1)	631(-1)
0	0000	0000
1	0010	0010
2	0101	0101
3	0100	0100
4	0110	0110
5	1001	1001
6	1000	1011
7	1010	1010
8	1101	1101
9	1100	1111

Tabela 9.13: Diferentes versões do código numérico BCD 631(-1).

Apesar de representarem apenas 10 valores diferentes, os códigos BCD não são compostos necessariamente por 4 *bits*. A Tabela 9.14 apresenta os códigos *2-out-of-5* e *Biquinary*, que são compostos por 5 e 7 *bits*, respectivamente.

Decimal	*2-out-of-5*	*2-out-of-5*	*Biquinary* (50 43210)	*Biquinary* (05 01234)
0	00011	00011	01 00001	10 10000
1	00101	00101	01 00010	10 01000
2	00110	01001	01 00100	10 00100
3	01001	10001	01 01000	10 00010
4	01010	00110	01 10000	10 00001
5	01100	01010	10 00001	01 10000
6	10001	10010	10 00010	01 01000
7	10010	01100	10 00100	01 00100
8	10100	10100	10 01000	01 00010
9	11000	11000	10 10000	01 00001

Tabela 9.14: Códigos numéricos BCD com mais de 4 *bits*: *2-out-of-5* e *Biquinary*.

De acordo com algumas características apresentadas pelos códigos BCD, eles podem ser classificados em:

- Código ponderado (*weighted code*): nessa classe de códigos, cada *bit* B_k é associado a um peso numérico w_k. No caso geral, o valor numérico é calculado por

$$q = C_B + \sum_k w_k \; B_k \; , \tag{9.42}$$

  onde C_B é a constante de polarização (*bias*) do código.

- Código auto-complementado (*self-complementing code*): nesse caso, a simples inversão dos *bits* (complemento a 1) conduz à complementação a 9 do valor numérico decimal relativo ao padrão binário.

- Código refletido (*reflected code*): para esses códigos, os padrões binários relativos aos valores decimais 0 a 4 são refletidos em relação aos dos valores 5 a 9, com exceção de uma das colunas de *bits*. Tais códigos podem ser complementados a 9 apenas pela inversão do *bit* pertencente à coluna não refletida.

- Código de conta exata (*exact count code*): em tais códigos, cada padrão binário apresenta a mesma quantidade de *bits* com valor booleano igual a "1".

- Código de distância unitária (*unit distance code*): em tal classe, os padrões binários numericamente adjacentes diferem apenas em uma das colunas de *bits*.

Quando os códigos BCD são empregados nos cálculos da aritmética decimal, a codificação mais utilizada é o complemento a 10. Porém, dado que estão sendo realizados cálculos numéricos, o processo mais natural de complementação a 10 é por meio da complementação a 9, seguida da adição de uma unidade ao resultado. Por isso, é interessante que um código BCD apresente simplicidade na sua complementação a 9.

Além daqueles já apresentados, alguns outros códigos BCD ponderados são reunidos na Tabela 9.15. Por sua vez, exemplos de códigos BCD não ponderados são apresentados na Tabela 9.16.

Decimal	7421	5421	5311	4221	2421	2421
0	0000	0000	0000	0000	0000	0000
1	0001	0001	0001	0001	0001	0001
2	0010	0010	0011	0010	0010	0010
3	0011	0011	0100	0011	0011	0011
4	0100	0100	0101	1000	0100	0100
5	0101	0101	1000	0111	0101	1011
6	0110	0110	1001	1100	0110	1100
7	1000	0111	1011	1101	0111	1101
8	1001	1011	1100	1110	1110	1110
9	1010	1100	1101	1111	1111	1111

Tabela 9.15: Outros códigos numéricos BCD ponderados: 7421, 5421, 5311, 4221 e 2421.

Decimal	I	II	III
0	0000	1000	1001
1	0001	0011	1000
2	0010	0010	0101
3	0011	1011	0000
4	0100	0000	1100
5	1100	0100	0011
6	1011	1111	1111
7	1010	0110	1010
8	1001	0111	0111
9	1000	1100	0110

Tabela 9.16: Exemplos de códigos numéricos BCD não ponderados.

9.5 Representação em ponto flutuante

A seguir, é apresentada uma representação denominada de ponto flutuante, normatizada pelo padrão IEEE 754. Em seguida, é realizada uma breve comparação entre a representação em ponto flutuante e a representação em ponto fixo.

9.5.1 O padrão de ponto flutuante IEEE 754

O padrão IEEE 754 define uma representação numérica denominada de ponto flutuante, contendo os seguintes elementos:

- S : *bit* de sinal, onde S=0 para um valor positivo e S=1 para um valor negativo.
- E : expoente relativo ou deslocado, que é representado por meio de um número inteiro, com X *bits*. O expoente efetivo e é calculado por $e = (E - V)$, onde $V = [2^{(X-1)} - 1]$.
- F : valor puramente fracionário, que é representado com R *bits*.
- q : a quantidade a ser representada, calculada a partir de S, F e E.

A quantidade q é representada em notação científica normalizada, o que significa dizer que, para todos os valores possíveis, sempre haverá exatamente um dígito inteiro não nulo antes do ponto fracionário. Uma vez que, na representação, é utilizada uma base binária, onde o conjunto de símbolos é $C_S = \{0, 1\}$, o dígito inteiro deve ser sempre igual a "1" e, portanto, não é necessário armazená-lo. Para a representação de números negativos é adotada uma codificação em Sinal-e-Magnitude. Assim, a quantidade q é calculada por

$$\begin{aligned} q &= (-1)^S\ (1+F)\ 2^e \\ &= (-1)^S\ (1+F)\ 2^{(E-V)} \\ &= (-1)^S\ (1+F)\ 2^{\{E-[2^{(X-1)}-1]\}}\ . \end{aligned} \tag{9.43}$$

A quantidade positiva $(1 + F)$ é denominada de significando ou mantissa. Uma vez que F é armazenado com R *bits*, a quantidade q é representada efetivamente com $(R + 1)$ *bits*.

O expoente armazenado E é um número inteiro e não negativo. Isso garante a representação de números grandes. Porém, para a representação de números pequenos, é necessário que o expoente seja negativo. Por essa razão, o valor de *offset* V é subtraído do expoente relativo E, gerando o expoente efetivo e, que passa a ser inteiro.

Alguns valores de q são reservados para representar determinados casos particulares. A Tabela 9.17 apresenta o conjunto de representações possíveis com o padrão IEEE 754.

Sinal (S)	Expoente (E)	Fração (F)	Quantidade (q)
0 / 1	0	0	$+0$ / -0
0 / 1	max	0	$+\infty$ / $-\infty$
0 / 1	max	$\neq$ 0	NaN
0 / 1	0	$\neq$ 0	não normalizado
0 / 1	1 a (max-1)	qualquer	normalizado

Tabela 9.17: Conjunto de representações possíveis com o padrão de ponto flutuante IEEE 754, onde: max $= (2^X - 1)$, X é o total de *bits* de E e NaN (*Not a Number*) indica que q não é um número.

O padrão IEEE 754 define duas representações básicas, com as seguintes denominações: precisão simples e precisão dupla.

No caso da precisão simples, é empregado um total de 32 *bits*, sendo 1 *bit* para S, 8 *bits* para E e 23 *bits* para F. Os valores representáveis são:

$$q = -\infty \ ; \ -2^{128} < q \leq -2^{-126} \ ; \ q = \pm 0 \ ; \ +2^{-126} \leq q < +2^{128} \ ; \ q = +\infty \ .$$

No caso da precisão dupla, é empregado um total de 64 *bits*, sendo 1 *bit* para S, 11 *bits* para E e 52 *bits* para F. Os valores representáveis são:

$$q = -\infty \ ; \ -2^{1024} < q \leq -2^{-1022} \ ; \ q = \pm 0 \ ; \ +2^{-1022} \leq q < +2^{1024} \ ; \ q = +\infty \ .$$

9.5.2 Ponto flutuante × ponto fixo

Supondo uma representação em ponto fixo com o mesmo tamanho de palavra que uma representação em ponto flutuante, W = B *bits*, ambas irão representar o mesmo total de valores numéricos $T = 2^B$.

A diferença entre as duas representações irá se manifestar em dois itens: na resolução da representação e na faixa de valores representáveis. Dito de outra forma, esses dois itens definem quais valores serão efetivamente representados.

No tocante à resolução, a representação em ponto fixo apresenta um resolução fixa, enquanto que, na representação em ponto flutuante, a resolução é variável.

Para alterar a faixa de valores representáveis, na representação de ponto fixo, é necessário aplicar um fator de escala fixo M_{fix}, externo à representação. Por outro lado, a notação científica, empregada na representação em ponto flutuante, já aplica, de forma implícita, um fator de escala variável M_{var}, aumentando naturalmente a sua faixa de valores.

O efeito causado pelas representações é uma distribuição uniforme de valores na faixa representável, para ponto fixo, e uma distribuição com uma compressão de resolução em um dos extremos da faixa e uma expansão de resolução no outro extremo, para ponto flutuante.

Esses aspectos são brevemente discutidos a seguir, de forma matemática.

Ponto fixo

Supondo-se uma representação em ponto fixo, codificada em Sinal-e-Magnitude, onde S é o *bit* de sinal, com tamanho de palavra W = B *bits*, sendo $B = (1 + X + R)$, então uma quantidade inteira q_I pode ser definida por

$$q_I = (-1)^S \ \left(\sum_{k=0}^{(B-1)-1} d_k \ 2^k \right) \ ,$$

com uma resolução

$$r_{q_I} = 1 \ .$$

Por sua vez, uma quantidade q_M, escalada por um fator de escala M_{fix}, é definida por

$$q_M = q_I \ M_{fix} = (-1)^S \ \left(\sum_{k=0}^{(B-1)-1} d_k \ 2^k \right) \ M_{fix} \ ,$$

com uma resolução

$$r_{q_M} = r_{q_I} \ M_{fix} = M_{fix} \ .$$

Assim, pode-se dizer que q_I é um caso particular de q_M, de tal forma que

$$q_I = q_M|_{M_{fix}=1}$$

e

$$r_{q_I} = r_{q_M}|_{M_{fix}=1} \ .$$

A Tabela 9.18 apresenta alguns exemplos para uma representação numérica em ponto fixo, codificado em Sinal-e-Magnitude, para alguns fatores de escala M_{fix} diferentes.

Padrão binário ($B = 1+3+2$)	q_I ($r_{q_I} = 1$)	q_M ($r_{q_M} = M_{fix} = 2^{-5}$)	q_M ($r_{q_M} = M_{fix} = 2^5$)
0/1-000-00	±0	±0.00000	± 0
0/1-000-01	±1	±0.03125	± 32
0/1-000-10	±2	±0.06250	± 64
0/1-000-11	±3	±0.09375	± 96
0/1-001-00	±4	±0.12500	±128
0/1-001-01	±5	±0.15625	±160
0/1-001-10	±6	±0.18750	±192
0/1-001-11	±7	±0.21875	±224
⋮	⋮	⋮	⋮
0/1-110-00	±24	±0.75000	±768
0/1-110-01	±25	±0.78125	±800
0/1-110-10	±26	±0.81250	±832
0/1-110-11	±27	±0.84375	±864
0/1-111-00	±28	±0.87500	±896
0/1-111-01	±29	±0.90625	±928
0/1-111-10	±30	±0.93750	±960
0/1-111-11	±31	±0.96875	±992

Tabela 9.18: Exemplos para uma representação numérica em ponto fixo, codificado em Sinal-e-Magnitude, para alguns fatores de escala M_{fix} diferentes.

Ponto flutuante

Supondo-se uma representação em ponto flutuante, onde S é o *bit* de sinal, com tamanho de palavra W = B *bits*, sendo $B = (1 + X + R)$, onde o expoente relativo E possui X *bits* e o valor fracionário F possui R *bits*, então uma quantidade q pode ser definida por

$$\begin{aligned} q &= (-1)^S \ (1+F) \ 2^e \\ &= (-1)^S \ (1+F) \ 2^{(E-V)} \\ &= (-1)^S \ q_s \ M_{var} \ , \end{aligned} \tag{9.44}$$

onde: $q_s = (1 + F)$ é o significando, V é um valor de *offset* e $M_{var} = 2^{(E-V)}$ é um fator de escala, variável com o valor de E.

A resolução r_s do significando q_s é fixa e dada por

$$r_s = 2^{-R} \ .$$

Por sua vez, a resolução r_q de q é variável e dada por

$$r_q = r_s \ M_{var} = 2^{-R} \ 2^{(E-V)} = 2^{(-R+E-V)}$$

e, para $0 \leq E \leq (2^X - 1)$, ocupa a faixa $2^{(-R-V)} \leq r_q \leq 2^{\{-R+(2^X-1)-V\}}$.

A Tabela 9.19 apresenta um exemplo para uma representação numérica em ponto flutuante, sem valores reservados, com X=3, R=2 e V=1.

Padrão binário (S-E-F)	q_s (1+F)	E	M_{var} $2^{(E-V)}$	q $[(-1)^S\ q_s\ M_{var}]$	r_q $2^{(-R+E-V)}$
0/1-000-00	1.00	0	2^{-1}	±0.500	0.125
0/1-000-01	1.25			±0.625	
0/1-000-10	1.50			±0.750	
0/1-000-11	1.75			±0.875	
0/1-001-00	1.00	1	2^{0}	±1.000	0.250
0/1-001-01	1.25			±1.250	
0/1-001-10	1.50			±1.500	
0/1-001-11	1.75			±1.750	
⋮	⋮	⋮	⋮	⋮	⋮
0/1-110-00	1.00	6	2^{5}	±32	8
0/1-110-01	1.25			±40	
0/1-110-10	1.50			±48	
0/1-110-11	1.75			±56	
0/1-111-00	1.00	7	2^{6}	±64	16
0/1-111-01	1.25			±80	
0/1-111-10	1.50			±96	
0/1-111-11	1.75			±112	

Tabela 9.19: Exemplo para uma representação numérica em ponto flutuante, sem valores reservados, com X=3, R=2 e V=1.

9.6 Aritmética binária

9.6.1 Tabelas de operações básicas entre dígitos

- Para uma determinada base, as operações de adição e multiplicação entre dígitos podem ser facilmente definidas por meio de tabelas.

- As Figuras 9.11 a 9.13 apresentam as tabelas para as bases $b = 2$, $b = 3$ e $b = 4$, respectivamente.

- A partir de tais tabelas, definidas para dígitos, podem ser definidos algoritmos e implementações para uma operação envolvendo quantidades genéricas, expressas na base em questão.

+	0	1
0	0	1
1	1	10

(a)

×	0	1
0	0	0
1	0	1

(b)

Figura 9.11: Tabelas de operações entre dígitos para $b = 2$: (a) adição e (b) multiplicação.

+	0	1	2
0	0	1	2
1	1	2	10
2	2	10	11

(a)

×	0	1	2
0	0	0	0
1	0	1	2
2	0	2	11

(b)

Figura 9.12: Tabelas de operações entre dígitos para $b = 3$: (a) adição e (b) multiplicação.

+	0	1	2	3
0	0	1	2	3
1	1	2	3	10
2	2	3	10	11
3	3	10	11	12

(a)

×	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	10	12
3	0	3	12	101

(b)

Figura 9.13: Tabelas de operações entre dígitos para $b = 4$: (a) adição e (b) multiplicação.

9.6.2 Escalamento por potência inteira da base

Um multiplicador é um circuito com relativa complexidade de implementação. Por isso, possui relevantes medidas de custo (espaço ocupado, energia consumida e tempo de operação).

Por outro lado, o escalamento por potência inteira da base é uma operação simples, com baixa complexidade de implementação. O escalamento pode ser de dois tipos, dependendo do valor da potência inteira da base: multiplicação (valor positivo) ou divisão (valor negativo).

Na Equação (9.45), é apresentada uma quantidade genérica q, representada na base b. A multiplicação de q pela base b é definida nas Equações (9.46) e (9.47). A divisão de q pela base b é definida nas Equações (9.48) e (9.49). De acordo com as Equações (9.45) e (9.49), a implementação do escalamento pode ser obtida através do simples deslocamento dos dígitos da representação.

$$(q)_b = (q_I)_b + (q_F)_b = \sum_{k=N_I}^{-N_F} d_k b^k = [d_{N_I} \cdots d_2 d_1 d_0 \cdot d_{-1} d_{-2} \cdots d_{-N_F}]_b \ . \tag{9.45}$$

$$\begin{aligned}
(q\prime)_b &= (q)_b \times b \\
&= \left(\sum_{k=N_I}^{-N_F} d_k b^k \right) \times b = \sum_{k=N_I}^{-N_F} d_k b^{k+1} = \sum_{k=N_I+1}^{-N_F+1} d_{k-1} b^k = \sum_{k=N_I+1}^{-N_F+1} d'_k b^k \\
&= [d_{N_I} \cdots d_2 d_1 d_0 d_{-1} \cdot d_{-2} \cdots d_{-N_F}]_b \\
&= [d'_{N_I+1} \cdots d'_3 d'_2 d'_1 d'_0 \cdot d'_{-1} d'_{-2} \cdots d'_{-N_F+1}]_b \ .
\end{aligned} \tag{9.46}$$

$$d'_k = d_{k-1} \ . \tag{9.47}$$

$$\begin{aligned}
(q\prime)_b &= (q)_b \times b^{-1} \\
&= \left(\sum_{k=N_I}^{-N_F} d_k b^k \right) \times b^{-1} = \sum_{k=N_I}^{-N_F} d_k b^{k-1} = \sum_{k=N_I-1}^{-N_F-1} d_{k+1} b^k = \sum_{k=N_I-1}^{-N_F-1} d'_k b^k \\
&= [d_{N_I} \cdots d_2 d_1 \cdot d_0 d_{-1} d_{-2} \cdots d_{-N_F}]_b \\
&= [d'_{N_I-1} \cdots d'_3 d'_2 d'_1 d'_0 \cdot d'_{-1} d'_{-2} \cdots d'_{-N_F-1}]_b \ .
\end{aligned} \tag{9.48}$$

$$d'_k = d_{k+1} \ . \tag{9.49}$$

9.6.3 Adição e subtração em complemento-a-2

A codificação em complemento-a-2 apresenta, entre outras, a grande vantagem de conseguir transformar o processo de subtração em pura adição: $x_1 - x_2 = x_1 + (-x_2) = x_1 + (x_2)_{C2}$. Assim, um único bloco somador, projetado para trabalhar com operandos representados em binário puro, sem codificação adicional, pode ser usado para realizar as operações de adição e subtração de números codificados em complemento-a-2.

A adição de dois números puramente fracionários pode produzir um número com parte inteira. Na representação de números puramente fracionários, com operação em ponto fixo, não são utilizados dígitos para valores inteiros. Portanto, um resultado contendo parte inteira, positivo ou negativo, é considerado uma situação de *overflow*. Para que o resultado da adição em complemento-a-2 possa ser considerado correto, a ocorrência de *overflow* deve ser detectada e devidamente tratada.

Análise de *overflow* na adição em complemento-a-2

Dado um bloco somador, projetado para trabalhar com operandos representados em binário puro, sem codificação adicional, operando em ponto fixo, com dados interpretados como puramente fracionários, codificados em complemento-a-2, o sinal de saída *carry out* representa uma parte inteira de valor $v_I = 2$.

Em tal tipo de procedimento, considerando-se d_s como o dígito de sinal de um número e C_o como o *carry out* do somador, podem-se identificar os seguintes casos:

- Caso 1: adição de números positivos $x_A = x_1 + x_2$.
  Tem-se que: $0 \leq x_1 < 1$, $0 \leq x_2 < 1$ e $x_R = x_1 + x_2$.
  Logo: $0 \leq x_R < 2$.
  Se $0 \leq x_R < 1$: resultado sem *overflow* $\rightarrow C_o = 0$ e $d_{sR} = 0$.
  Se $1 \leq x_R < 2$: resultado com *overflow* $\rightarrow C_o = 0$ e $d_{sR} = 1$.
  No caso do resultado sem *overflow*: $x_A = x_R$.

- Caso 2: subtração de números positivos $x_A = x_1 - |x_2|$
  ou adição de um número positivo com um número negativo $x_A = x_1 + (-|x_2|)$.
  Tem-se que: $0 \leq x_1 < 1$, $-1 < x_2 < 0$, $(-|x_2|)_{C2} = 2 - |x_2|$ e
  $x_R = x_1 + x_2 = x_1 + (-|x_2|) = x_1 + (x_2)_{C2} = x_1 + (2 - |x_2|) = 2 + (x_1 - |x_2|)$.
  Logo: $-1 < x_R < 1$.
  Portanto, nesse caso, não há ocorrência de *overflow*.
  Se $x_1 \geq |x_2|$: resultado não negativo $\rightarrow C_o = 1$ e $d_{sR} = 0$.
  Se $x_1 < |x_2|$: resultado negativo $\rightarrow C_o = 0$ e $d_{sR} = 1$.
  Para o resultado não negativo, basta ignorar o sinal de *carry out*: $x_A = (x_R - 2)$.
  Para o resultado negativo, ele já está corretamente codificado: $x_A = (x_R)_{C2} = (2 - |x_R|)$.

- Caso 3: adição de números negativos $x_A = (-|x_1|) + (-|x_2|)$.
  Tem-se que: $-1 < x_2 < 0$, $-1 < x_2 < 0$, $(-|x_1|)_{C2} = 2 - |x_1|$, $(-|x_2|)_{C2} = 2 - |x_2|$ e
  $x_R = x_1 + x_2 = (-|x_1|)_{C2} + (-|x_2|)_{C2} = (2 - |x_1|) + (2 - |x_2|) = 2 + [2 - (|x_1| + |x_2|)]$.
  Logo: $0 < |x_1| + |x_2| < 2 \rightarrow 2 < x_R < 4$.
  Se $2 < x_R \leq 3$: adição com *overflow* $\rightarrow C_o = 1$ e $d_{sR} = 0$.
  Se $3 < x_R < 4$: adição sem *overflow* $\rightarrow C_o = 1$ e $d_{sR} = 1$.
  No caso do resultado sem *overflow*, ele já está corretamente codificado, bastando ignorar o sinal de *carry out*: $x_A = (x_R - 2)$.

Exemplos para os casos de adição em complemento-a-2

- Caso 1: adição de números positivos $x_A = x_1 + x_2$.

```
      0|1011            11              0.6875
    + 0|0011          +  3            +  0.1875
    --------          -------         ------------
[0]   0|1110            14              0.8750

      0|1011            11              0.6875
    + 0|0111          +  7            +  0.4375
    --------          -------         ------------
[0]   1|0010            18              1.1250
```

- Caso 2: subtração de números positivos $x_A = x_1 - |x_2|$
  ou adição de um número positivo com um número negativo $x_A = x_1 + (-|x_2|)$.

```
      0|1011            11              0.6875
    + 1|1101          + (- 3)         + (- 0.1875)
    --------          -------         ------------
[1]   0|1000             8              0.5000

      0|1011            11              0.6875
    + 1|0101          + (-11)         + (- 0.6875)
    --------          -------         ------------
[1]   0|0000             0              0.0000

      0|0011             3              0.1875
    + 1|0101          + (-11)         + (- 0.6875)
    --------          -------         ------------
[0]   1|1000            (- 8)           (- 0.5000)
```

- Caso 3: adição de números negativos $x_A = (-|x_1|) + (-|x_2|)$.

```
      1|0101            (-11)           (- 0.6875)
    + 1|1101          + (- 3)         + (- 0.1875)
    --------          -------         ------------
[1]   1|0010            (-14)           (- 0.8750)

      1|0101            (-11)           (- 0.6875)
    + 1|1001          + (- 7)         + (- 0.4375)
    --------          -------         ------------
[1]   0|1110            (-18)         + (- 1.1250)
```

Detecção de *overflow* na adição em complemento-a-2

A partir dos resultados da análise de *overflow* na adição em complemento-a-2, não é difícil encontrar um mecanismo que indique sua ocorrência. Utilizando sinais externos ao bloco de adição, a detecção de *overflow* pode ser feita através da análise dos *bits* de sinal dos operandos (d_{s1} e d_{s2}) e do resultado (d_{sA}), bem como do sinal de *carry out* (C_o) do bloco somador. As Tabelas 9.20 e 9.21 apresentam duas formas para representar a detecção de *overflow* na adição em complemento-a-2, onde $OF = 0$ e $OF = 1$ indicam a ausência e a presença de *overflow*, respectivamente. A diferença entre as duas formas é que, na Tabela 9.20, são levados em consideração os casos que não podem acontecer (*can't happen*), o que pode levar a simplificações da função final. Pode-se verificar que, em ambos os casos, a equação simplificada é dada por

$$OF = \left(\overline{d_{s1}} \cdot \overline{d_{s2}} \cdot d_{sA}\right) + \left(d_{s1} \cdot d_{s2} \cdot \overline{d_{sA}}\right) \ . \tag{9.50}$$

Tratamento de *overflow*

O tratamento de *overflow* mais comumente empregado é a saturação do resultado no valor máximo representável (positivo ou negativo).

Em alguns casos, onde é exigido que os dados sejam interpretados como sendo estritamente menores que a unidade ($|x| < 1$), a solução é a saturação do resultado no valor máximo representável (positivo ou negativo) imediatamente menor que a unidade.

Caso	d_{s1}	d_{s2}	d_{sA}	C_o	OF
Adição de positivos	0	0	0	0	0
				1	X
			1	0	1
				1	X
Subtração de positivos	0	1	0	0	X
				1	0
			1	0	0
				1	X
	1	0	0	0	X
				1	0
			1	0	0
				1	X
Adição de negativos	1	1	0	0	X
				1	1
			1	0	X
				1	0

X = *can't happen*

$OF = 0 \rightarrow$ sem *overflow*
$OF = 1 \rightarrow$ com *overflow*

Tabela 9.20: Forma 1 para representar a detecção de *overflow* na adição em complemento-a-2.

Caso	d_{s1}	d_{s2}	d_{sA}	C_o	OF
Adição de positivos	0	0	0	X	0
	0	0	1	X	1
Subtração de positivos	0	1	0	X	0
	1	0	0	X	0
Adição de negativos	1	1	0	X	1
	1	1	1	X	0

$X = \textit{don't care}$

$OF = 0 \rightarrow$ sem *overflow*
$OF = 1 \rightarrow$ com *overflow*

Tabela 9.21: Forma 2 para representar a detecção de *overflow* na adição em complemento-a-2.

9.6.4 Funções envolvidas na adição de dígitos binários

A adição de dígitos binários pode ser facilmente definida por meio de tabelas. A Figura 9.14 apresenta as tabelas que definem a adição entre dois dígitos binários (A e B). A Figura 9.14.(a) define a operação completa, com todos os resultados descritos por dois dígitos. A Figura 9.14.(b) define os dígitos da esquerda do resultado, que representam a parte excedente da adição, denominada de "vai-um" (*carry out* - C_o). A Figura 9.14.(c) define os dígitos da direita, que representam o resultado básico da adição, denominado de soma (S).

+	0	1
0	00	01
1	01	10

(a)

C_o	0	1
0	0	0
1	0	1

(b)

S	0	1
0	0	1
1	1	0

(c)

Figura 9.14: Tabelas que definem a adição entre dois dígitos binários (A e B): (a) adição completa, (b) "vai-um" (*carry out* - C_o) e (c) soma (S).

Apesar dos resultados das tabelas de C_o e de S serem numéricos, os mesmos podem ser interpretados como valores booleanos. Assim, supondo-se os dígitos binários A e B, pode-se dizer que

$$C_o = f_{C2}(A, B) = (A \cdot B)$$

e que

$$S = f_{S2}(A, B) = \left(\overline{A} \cdot B\right) + \left(A \cdot \overline{B}\right) = (A \oplus B) \ .$$

Na adição entre dois números com diversos dígitos binários, naturalmente acontece uma propagação de excessos de soma, denominados de C_o (*carry out*) para os blocos que os geram e de C_i (*carry in*) para os blocos que os recebem. Logo, torna-se necessário definir a adição entre três dígitos binários (A, B e C_i), o que é feito nas tabelas da Figura 9.15.

+	00	01	11	10
0	00	01	10	01
1	01	10	11	10

(a)

C_o	00	01	11	10
0	0	0	1	0
1	0	1	1	1

(b)

S	00	01	11	10
0	0	1	0	1
1	1	0	1	0

(c)

Figura 9.15: Tabelas que definem a adição entre três dígitos binários (A, B e C_i): (a) adição completa, (b) "vai-um" (*carry out* - C_o) e (c) soma (S).

Nesse caso, supondo-se os dígitos A e B na parte superior das tabelas e o dígito C_i à esquerda, pode-se dizer que

$$\begin{aligned} S &= f_{S3}(A,B,C_i) = \left(\overline{A}\cdot B\cdot \overline{C_i}\right) + \left(A\cdot \overline{B}\cdot \overline{C_i}\right) + \left(\overline{A}\cdot \overline{B}\cdot C_i\right) + (A\cdot B\cdot C_i) \\ &= \left[(A\oplus B)\cdot \overline{C_i}\right] + \left[\overline{(A\oplus B)}\cdot C_i\right] \\ &= (A\oplus B)\oplus C_i \end{aligned}$$

e que

$$C_o = f_{C3}(A,B,C_i) = \left(A\cdot B\cdot \overline{C_i}\right) + (A\cdot B\cdot C_i) + \left(\overline{A}\cdot B\cdot C_i\right) + \left(A\cdot \overline{B}\cdot C_i\right) \ ,$$

que pode ser simplificada como

$$\begin{aligned} C_o &= f_{C3}(A,B,C_i) \\ &= (A\cdot B) + (A\cdot C_i) + (B\cdot C_i) \end{aligned}$$

e ainda como

$$\begin{aligned} C_o &= f_{C3}(A,B,C_i) \\ &= (A\cdot B) + [(A+B)\cdot C_i] \ , \end{aligned}$$

ou como

$$\begin{aligned} C_o &= f_{C3}(A,B,C_i) \\ &= (A\cdot B) + \left[\left(\overline{A}\cdot B\right) + \left(A\cdot \overline{B}\right)\right]\cdot C_i \\ &= (A\cdot B) + [(A\oplus B)\cdot C_i] \ . \end{aligned}$$

Procurando-se otimizar algumas implementações, pode-se mostrar ainda que

$$\overline{C_o} = f_{C3}\left(\overline{A},\overline{B},\overline{C_i}\right)$$

e que

$$\overline{S} = f_{S3}\left(\overline{A},\overline{B},\overline{C_i}\right) \ .$$

As funções definidas acima são suficientes para implementar um somador de dois números com N dígitos binários através do algoritmo mais elementar. Porém, três funções mostram-se bastante úteis na implementação de somadores com algoritmos mais complexos: *Generate* (G), *Propagate* (P) e *Kill* (K). A Figura 9.16 apresenta as tabelas que definem as funções G, P e K, supondo-se os dígitos A e B na parte superior das tabelas e o dígito C_i à esquerda. Para melhor entender a sua aplicabilidade, é importante ressaltar a principal característica das funções G, P e K, que é não depender de C_i.

G	00	01	11	10
0	0	0	1	0
1	0	0	1	0

(a)

P	00	01	11	10
0	0	1	0	1
1	0	1	0	1

(b)

K	00	01	11	10
0	1	0	0	0
1	1	0	0	0

(c)

Figura 9.16: Tabelas que definem três funções úteis na implementação de somadores binários com algoritmos complexos: (a) *Generate* (G), (b) *Propagate* (P) e (c) *Kill* (K).

A função *Generate* (G) assume o valor booleano "1" quando $C_o = 1$ independentemente do valor de C_i. Logo, deve-se ter $A = B = 1$, definindo-se

$$G = f_G(A, B, C_i) = (A \cdot B) \ .$$

A função *Propagate* (P) assume o valor booleano "1" quando $C_o = 1$ por propagação exclusiva do valor $C_i = 1$. Logo, deve-se ter $A = 0$ e $B = 1$ ou $A = 1$ e $B = 0$, definindo-se

$$P = f_P(A, B, C_i) = (A \oplus B) \ .$$

A função *Kill* (K) assume o valor booleano "1" quando é impossível ter $C_o = 1$ independentemente do valor de C_i. Logo, deve-se ter $A = B = 0$, definindo-se

$$K = f_K(A, B, C_i) = (\overline{A} \cdot \overline{B}) \ .$$

As funções S e C_o podem ser escritas em função de G e P, de tal forma que

$$\begin{aligned} S &= f_{S3}(A, B, C_i) \\ &= (A \oplus B) \oplus C_i \ = \ (P \oplus C_i) \end{aligned}$$

e

$$\begin{aligned} C_o &= f_{C3}(A, B, C_i) \\ &= (A \cdot B) + [(A \oplus B) \cdot C_i] \ = \ G + (P \cdot C_i) \\ &= (A \cdot B) + [(A + B) \cdot C_i] \ = \ G + (P^+ \cdot C_i) \ , \end{aligned}$$

onde

$$P^+ = (A + B)$$

pode ser usada para diminuir o tempo de propagação envolvido na geração do sinal C_o.

9.7 Exercícios propostos

1. Considerando o SNPC, para cada uma das bases listadas abaixo, obter as respectivas representações para as quantidades apresentadas em seguida.

   (a) Base $b = 2$.

   (b) Base $b = 3$.

   (c) Base $b = 16$.

   Quantidades numéricas:

   i. $q = (17)_{10}$.
   ii. $q = (24)_{10}$.
   iii. $q = (32)_{10}$.
   iv. $q = (48)_{10}$.
   v. $q = (80)_{10}$.
   vi. $q = (144)_{10}$.
   vii. $q = (272)_{10}$.
   viii. $q = (528)_{10}$.

2. Considerando o SNPC, com base $b = 2$, para cada uma das codificações listadas abaixo, obter as respectivas representações para as quantidades apresentadas em seguida.

   (a) Sinal-e-magnitude.

   (b) Complemento-a-1.

   (c) Complemento-a-2.

   Quantidades numéricas:

   i. $q = (-17)_{10}$.
   ii. $q = (-24)_{10}$.
   iii. $q = (-32)_{10}$.
   iv. $q = (-48)_{10}$.
   v. $q = (-80)_{10}$.
   vi. $q = (-144)_{10}$.
   vii. $q = (-272)_{10}$.
   viii. $q = (-528)_{10}$.

3. Considerando o SNPC, com base $b = 2$, com codificação em complemento-a-2, analise o resultado das seguintes operações:

   (a) $(00100) + (01001)$.

   (b) $(01100) + (01101)$.

   (c) $(00100) + (10111)$.

   (d) $(10100) + (01101)$.

   (e) $(11100) + (10111)$.

   (f) $(10100) + (10011)$.

4. Considerando o SNPC, com base $b = 2$, com codificação em complemento-a-2, com um total 5 dígitos, para cada uma das quantizações listadas abaixo, obter as respectivas representações para as quantidades apresentadas em seguida.

   (a) Truncamento.

   (b) Arredondamento.

   (c) Truncamento em magnitude.

   Quantidades numéricas:

   i. (0010000).
   ii. (0010001).
   iii. (0010010).
   iv. (0010011).
   v. (0010100).
   vi. (0010101).
   vii. (0010110).
   viii. (0010111).
   ix. (1110000).
   x. (1101111).
   xi. (1101110).
   xii. (1101101).
   xiii. (1101100).
   xiv. (1101011).
   xv. (1101010).
   xvi. (1101001).

5. Dada a Tabela 9.22, projete um circuito combinacional que receba o Código 1 e produza o Código 2.

Código 1	Código 2
000	1111
001	0011
010	0101
011	1011
100	0000
101	1101
110	1000
111	0111

Tabela 9.22: Tabela de conversão do Código 1 para o Código 2.

Capítulo 10

Circuitos combinacionais básicos

10.1 Introdução

- Esse capítulo trata do projeto de alguns exemplos de circuitos combinacionais simples, básicos e tanto necessários quanto comuns a diversas aplicações.
- Técnicas de projetos
  - Não existe uma técnica de projeto única que atenda a todos os tipos de problemas.
  - Para problemas com baixa complexidade e poucas variáveis:
    - Projeto formal: equacionamento lógico direto + minimização das equações + implementação do circuito.
    - Tentativa-e-erro (*cut-and-try*): sugestão de uma solução, sem a aplicação de uma técnica formal, seguida de verificação da funcionalidade.
  - Para problemas com alta complexidade e/ou muitas variáveis:
    - Divisão do sistema original em subsistemas (*divide-to-conquer*), a fim de diminuir a complexidade do sistema a ser projetado.
    - Cada subsistema pode ser subdividido, acarretando um projeto hierárquico.
    - Para cada subsistema:
      - Projeto formal.
      - Uso de blocos já projetados.
      - Tipos de blocos: idênticos (projeto modular) ou diferentes.
- Arquiteturas básicas
  - Paralela.
  - Serial.

10.2 Interpretações dos circuitos combinacionais

- Internamente, um circuito combinacional é apenas um conjunto de portas lógicas interligadas, sem realimentações, que realizam operações lógicas com nenhum significado extra.
- Externamente, baseado nas relações entre as variáveis de entrada e de saída, os circuitos combinacionais podem ser interpretados de diferentes formas, de acordo com suas aplicações.

10.2.1 Exemplos de interpretações

- Gerador de funções lógicas
  - Entrada: variáveis ou parâmetros de entrada para funções lógicas.
  - Saída: resultados provenientes da avaliação das funções lógicas implementadas.
- Interpretador de comandos
  - Padrões binários apresentados na entrada dos circuitos combinacionais podem ser interpretados como palavras de comando ou instruções a serem interpretadas e executadas pelo circuito.
  - O conjunto de padrões possíveis de serem apresentados, interpretados e executados pelo circuito, representa o denominado conjunto de instruções (*instruction set*) que o circuito compreende.
  - Por exemplo, um circuito combinacional que implemente a função lógica AND, com duas entradas, pode ser interpretado como um circuito que compreende quatro comandos e produz uma saída que pode assumir quatro valores possíveis.
  - Entrada: palavra de comando ou instrução, podendo conter dados dentro do código da instrução.
  - Saída: sinais de controle que irão controlar a execução do comando e, possivelmente, dados.
- Conversor de códigos
  - Entrada: código original.
  - Saída: novo código.
- Sistema digital instantâneo ou sem memória
  - Entrada: seqüência ou sinal de entrada.
  - Saída: seqüência ou sinal de saída.
- Controle de fluxo de dados
  - Entrada: dados de entrada + sinais de controle de entrada.
  - Saída: dados de saída + sinais de controle de saída.
- Operador
  - Entrada: Operandos.
  - Saída: Resultados das operações realizadas pelo circuito sobre os operandos apresentados.
- Operador programável
  - Entrada: Operandos + Sinais de controle da operação.
  - Saída: Resultados das operações escolhidas para serem realizadas pelo circuito sobre os operandos apresentados.

10.3 Uso de portas lógicas como elementos de controle

- Uma porta lógica com N entradas pode ser interpretada como um bloco funcional com 1 saída, 1 entrada e $(N-1)$ sinais de controle.

10.4 Uso de elementos de controle para mascaramento

- Por vezes, é necessário interromper o fluxo de um sinal, baseado em determinadas condições. Isso é denominado de mascaramento do sinal, onde a máscara do processo é formada pelo conjunto de condições envolvidas.

- Um elemento de controle pode ser empregado para implementar o mascaramento, aplicando-se o sinal na sua entrada e a máscara no seu controle.

10.5 Gerador de funções lógicas

- Um circuito combinacional com N entradas e M saídas pode ser interpretado como M funções lógicas de N variáveis.

- Cada uma das M funções pode ser definida independentemente das demais.

- As funções podem ser implementadas de forma independente uma das outras, facilitando o projeto, os testes e a manutenção, ou compartilhando partes do circuito, a fim de reduzir custos de implementação.

10.6 Conversor de códigos

- Um conversor de códigos é um circuito combinacional com N entradas e M saídas, onde, para cada padrão de valores de entrada definido, existe um padrão de valores de saída correspondente.

10.6.1 Exemplo de projeto para conversor de códigos

A seguir, é apresentado um exemplo de projeto para um conversor numérico, de binário para BCD 8421.

Conversor numérico: binário para BCD 8421

A Tabela 10.1 apresenta a relação entre os códigos numéricos decimal, binário e BCD 8421, para um código binário com 4 *bits*. Para implementar um conversor numérico, de binário para BCD 8421, basta considerar cada coluna do código BCD 8421 como uma tabela verdade de uma função lógica dependente das quatro variáveis do código binário.

Decimal	Binário	BCD 8421
	$(B_3B_2B_1B_0)$	$(D_0\ U_3U_2U_1U_0)$
00	0000	0 0000
01	0001	0 0001
02	0010	0 0010
03	0011	0 0011
04	0100	0 0100
05	0101	0 0101
06	0110	0 0110
07	0111	0 0111
08	1000	0 1000
09	1001	0 1001
10	1010	1 0000
11	1011	1 0001
12	1100	1 0010
13	1101	1 0011
14	1110	1 0100
15	1111	1 0101

Tabela 10.1: Relação entre os códigos numéricos decimal, binário e BCD 8421, para um código binário com 4 *bits*.

Da Tabela 10.1, podem-se obter as listas de mintermos e de maxtermos, bem como as formas SOP padrão e POS padrão, para cada função geradora do código BCD 8421. A partir delas, podem-se alcançar as formas SOP mínima e POS mínima, como é apresentado a seguir.

$$
\begin{aligned}
D_0 &= \sum m(10,11,12,13,14,15) \\
&= (m_{10}+m_{11}+m_{14}+m_{15})+(m_{12}+m_{13}+m_{14}+m_{15}) \\
&= \Big[(B_3\cdot\overline{B_2}\cdot B_1\cdot\overline{B_0})+(B_3\cdot\overline{B_2}\cdot B_1\cdot B_0)+(B_3\cdot B_2\cdot B_1\cdot\overline{B_0})+(B_3\cdot B_2\cdot B_1\cdot B_0)\Big]+ \\
&\quad \Big[(B_3\cdot B_2\cdot\overline{B_1}\cdot\overline{B_0})+(B_3\cdot B_2\cdot\overline{B_1}\cdot B_0)+(B_3\cdot B_2\cdot B_1\cdot\overline{B_0})+(B_3\cdot B_2\cdot B_1\cdot B_0)\Big] \\
&= (B_3\cdot B_1)+(B_3\cdot B_2) \qquad (10.1)
\end{aligned}
$$

$$
\begin{aligned}
&= \prod M(0,1,2,3,4,5,6,7,8,9) \\
&= (M_0\cdot M_1\cdot M_2\cdot M_3\cdot M_4\cdot M_5\cdot M_6\cdot M_7)\cdot(M_0\cdot M_1\cdot M_8\cdot M_9) \\
&= \Big[(B_3+B_2+B_1+B_0)\cdot(B_3+B_2+B_1+\overline{B_0})\cdot(B_3+B_2+\overline{B_1}+B_0)\cdot(B_3+B_2+\overline{B_1}+\overline{B_0})\cdot \\
&\quad (B_3+\overline{B_2}+B_1+B_0)\cdot(B_3+\overline{B_2}+B_1+\overline{B_0})\cdot(B_3+\overline{B_2}+\overline{B_1}+B_0)\cdot(B_3+\overline{B_2}+\overline{B_1}+\overline{B_0})\Big]\cdot \\
&\quad \Big[(B_3+B_2+B_1+B_0)\cdot(B_3+B_2+B_1+\overline{B_0})\cdot(\overline{B_3}+B_2+B_1+B_0)\cdot(\overline{B_3}+B_2+B_1+\overline{B_0})\Big] \\
&= (B_3)\cdot(B_2+B_1) \qquad (10.2)
\end{aligned}
$$

$$
\begin{aligned}
U_3 &= \sum m(8,9) \\
&= (m_8 + m_9) \\
&= (B_3 \cdot \overline{B_2} \cdot \overline{B_1} \cdot \overline{B_0}) + (B_3 \cdot \overline{B_2} \cdot \overline{B_1} \cdot B_0) \\
&= (B_3 \cdot \overline{B_2} \cdot \overline{B_1}) \qquad (10.3)
\end{aligned}
$$

$$
\begin{aligned}
&= \prod M(0,1,2,3,4,5,6,7,10,11,12,13,14,15) \\
&= (M_0 \cdot M_1 \cdot M_2 \cdot M_3 \cdot M_4 \cdot M_5 \cdot M_6 \cdot M_7) \cdot \\
&\quad (M_4 \cdot M_5 \cdot M_6 \cdot M_7 \cdot M_{12} \cdot M_{13} \cdot M_{14} \cdot M_{15}) \cdot \\
&\quad (M_2 \cdot M_3 \cdot M_6 \cdot M_7 \cdot M_{10} \cdot M_{11} \cdot M_{14} \cdot M_{15}) \\
&= \Big[(B_3 + B_2 + B_1 + B_0) \cdot (B_3 + B_2 + B_1 + \overline{B_0}) \cdot (B_3 + B_2 + \overline{B_1} + B_0) \cdot (B_3 + B_2 + \overline{B_1} + \overline{B_0}) \cdot \\
&\quad (B_3 + \overline{B_2} + B_1 + B_0) \cdot (B_3 + \overline{B_2} + B_1 + \overline{B_0}) \cdot (B_3 + \overline{B_2} + \overline{B_1} + B_0) \cdot (B_3 + \overline{B_2} + \overline{B_1} + \overline{B_0})\Big] \cdot \\
&\quad \Big[(B_3 + \overline{B_2} + B_1 + B_0) \cdot (B_3 + \overline{B_2} + B_1 + \overline{B_0}) \cdot (B_3 + \overline{B_2} + \overline{B_1} + B_0) \cdot (B_3 + \overline{B_2} + \overline{B_1} + \overline{B_0}) \cdot \\
&\quad (\overline{B_3} + \overline{B_2} + B_1 + B_0) \cdot (\overline{B_3} + \overline{B_2} + B_1 + \overline{B_0}) \cdot (\overline{B_3} + \overline{B_2} + \overline{B_1} + B_0) \cdot (\overline{B_3} + \overline{B_2} + \overline{B_1} + \overline{B_0})\Big] \cdot \\
&\quad \Big[(B_3 + B_2 + \overline{B_1} + B_0) \cdot (B_3 + B_2 + \overline{B_1} + \overline{B_0}) \cdot (B_3 + \overline{B_2} + \overline{B_1} + B_0) \cdot (B_3 + \overline{B_2} + \overline{B_1} + \overline{B_0}) \cdot \\
&\quad (\overline{B_3} + B_2 + \overline{B_1} + B_0) \cdot (\overline{B_3} + B_2 + \overline{B_1} + \overline{B_0}) \cdot (\overline{B_3} + \overline{B_2} + \overline{B_1} + B_0) \cdot (\overline{B_3} + \overline{B_2} + \overline{B_1} + \overline{B_0})\Big] \\
&= (B_3) \cdot (\overline{B_2}) \cdot (\overline{B_1}) \qquad (10.4)
\end{aligned}
$$

$$
\begin{aligned}
U_2 &= \sum m(4,5,6,7,14,15) \\
&= (m_4 + m_5 + m_6 + m_7) + (m_6 + m_7 + m_{14} + m_{15}) \\
&= \Big[(\overline{B_3} \cdot B_2 \cdot \overline{B_1} \cdot \overline{B_0}) + (\overline{B_3} \cdot B_2 \cdot \overline{B_1} \cdot B_0) + (\overline{B_3} \cdot B_2 \cdot B_1 \cdot \overline{B_0}) + (\overline{B_3} \cdot B_2 \cdot B_1 \cdot B_0)\Big] + \\
&\quad \Big[(\overline{B_3} \cdot B_2 \cdot B_1 \cdot \overline{B_0}) + (\overline{B_3} \cdot B_2 \cdot B_1 \cdot B_0) + (B_3 \cdot B_2 \cdot B_1 \cdot \overline{B_0}) + (B_3 \cdot B_2 \cdot B_1 \cdot B_0)\Big] \\
&= (\overline{B_3} \cdot B_2) + (B_2 \cdot B_1) \qquad (10.5)
\end{aligned}
$$

$$
\begin{aligned}
&= \prod M(0,1,2,3,8,9,10,11,12,13) \\
&= (M_0 \cdot M_1 \cdot M_2 \cdot M_3 \cdot M_8 \cdot M_9 \cdot M_{10} \cdot M_{11}) \cdot (M_8 \cdot M_9 \cdot M_{12} \cdot M_{13}) \\
&= \Big[(B_3 + B_2 + B_1 + B_0) \cdot (B_3 + B_2 + B_1 + \overline{B_0}) \cdot (B_3 + B_2 + \overline{B_1} + B_0) \cdot (B_3 + B_2 + \overline{B_1} + \overline{B_0}) \cdot \\
&\quad (\overline{B_3} + B_2 + B_1 + B_0) \cdot (\overline{B_3} + B_2 + B_1 + \overline{B_0}) \cdot (\overline{B_3} + B_2 + \overline{B_1} + B_0) \cdot (\overline{B_3} + B_2 + \overline{B_1} + \overline{B_0})\Big] \cdot \\
&\quad \Big[(\overline{B_3} + B_2 + B_1 + B_0) \cdot (\overline{B_3} + B_2 + B_1 + \overline{B_0}) \cdot (\overline{B_3} + \overline{B_2} + B_1 + B_0) \cdot (\overline{B_3} + \overline{B_2} + B_1 + \overline{B_0})\Big] \\
&= (B_2) \cdot (\overline{B_3} + B_1) \qquad (10.6)
\end{aligned}
$$

$$
\begin{aligned}
U_1 &= \sum m(2,3,6,7,12,13) \\
&= (m_2 + m_3 + m_6 + m_7) + (m_{12} + m_{13}) \\
&= \Big[(\overline{B_3} \cdot \overline{B_2} \cdot B_1 \cdot \overline{B_0}) + (\overline{B_3} \cdot \overline{B_2} \cdot B_1 \cdot B_0) + (\overline{B_3} \cdot B_2 \cdot B_1 \cdot \overline{B_0}) + (\overline{B_3} \cdot B_2 \cdot B_1 \cdot B_0)\Big] + \\
&\quad \Big[(B_3 \cdot B_2 \cdot \overline{B_1} \cdot \overline{B_0}) + (B_3 \cdot B_2 \cdot \overline{B_1} \cdot B_0)\Big] \\
&= (\overline{B_3} \cdot B_1) + (B_3 \cdot B_2 \cdot \overline{B_1}) \qquad (10.7)
\end{aligned}
$$

$$
\begin{aligned}
&= \prod M(0,1,4,5,8,9,10,11,14,15) \\
&= (M_0 \cdot M_1 \cdot M_4 \cdot M_5) \cdot (M_{10} \cdot M_{11} \cdot M_{14} \cdot M_{15}) \cdot (M_0 \cdot M_1 \cdot M_8 \cdot M_9) \\
&\quad \Big[(B_3 + B_2 + B_1 + B_0) \cdot (B_3 + B_2 + B_1 + \overline{B_0}) \cdot (B_3 + \overline{B_2} + B_1 + B_0) \cdot (B_3 + \overline{B_2} + B_1 + \overline{B_0})\Big] \cdot \\
&\quad \Big[(\overline{B_3} + B_2 + \overline{B_1} + B_0) \cdot (\overline{B_3} + B_2 + \overline{B_1} + \overline{B_0}) \cdot (\overline{B_3} + \overline{B_2} + \overline{B_1} + B_0) \cdot (\overline{B_3} + \overline{B_2} + \overline{B_1} + \overline{B_0})\Big] \cdot \\
&\quad \Big[(B_3 + B_2 + B_1 + B_0) \cdot (B_3 + B_2 + B_1 + \overline{B_0}) \cdot (\overline{B_3} + B_2 + B_1 + B_0) \cdot (\overline{B_3} + B_2 + B_1 + \overline{B_0})\Big] \\
&= (B_3 + B_1) \cdot (\overline{B_3} + \overline{B_1}) \cdot (B_2 + B_1)
\end{aligned}
$$

$$
\begin{aligned}
&= (M_0 \cdot M_1 \cdot M_4 \cdot M_5) \cdot (M_{10} \cdot M_{11} \cdot M_{14} \cdot M_{15}) \cdot (M_8 \cdot M_9 \cdot M_{10} \cdot M_{11}) \\
&\quad \Big[(B_3 + B_2 + B_1 + B_0) \cdot (B_3 + B_2 + B_1 + \overline{B_0}) \cdot (B_3 + \overline{B_2} + B_1 + B_0) \cdot (B_3 + \overline{B_2} + B_1 + \overline{B_0})\Big] \cdot \\
&\quad \Big[(\overline{B_3} + B_2 + \overline{B_1} + B_0) \cdot (\overline{B_3} + B_2 + \overline{B_1} + \overline{B_0}) \cdot (\overline{B_3} + \overline{B_2} + \overline{B_1} + B_0) \cdot (\overline{B_3} + \overline{B_2} + \overline{B_1} + \overline{B_0})\Big] \cdot \\
&\quad \Big[(\overline{B_3} + B_2 + B_1 + B_0) \cdot (\overline{B_3} + B_2 + B_1 + \overline{B_0}) \cdot (\overline{B_3} + B_2 + \overline{B_1} + B_0) \cdot (\overline{B_3} + B_2 + \overline{B_1} + \overline{B_0})\Big] \\
&= (B_3 + B_1) \cdot (\overline{B_3} + \overline{B_1}) \cdot (\overline{B_3} + B_2) \qquad (10.8)
\end{aligned}
$$

$$
\begin{aligned}
U_0 &= \sum m(1,3,5,7,9,11,13,15) \\
&= (m_1 + m_3 + m_5 + m_7 + m_9 + m_{11} + m_{13} + m_{15}) \\
&= \Big[(\overline{B_3} \cdot \overline{B_2} \cdot \overline{B_1} \cdot B_0) + (\overline{B_3} \cdot \overline{B_2} \cdot B_1 \cdot B_0) + (\overline{B_3} \cdot B_2 \cdot \overline{B_1} \cdot B_0) + (\overline{B_3} \cdot B_2 \cdot B_1 \cdot B_0) + \\
&\quad (B_3 \cdot \overline{B_2} \cdot \overline{B_1} \cdot B_0) + (B_3 \cdot \overline{B_2} \cdot B_1 \cdot B_0) + (B_3 \cdot B_2 \cdot \overline{B_1} \cdot B_0) + (B_3 \cdot B_2 \cdot B_1 \cdot B_0)\Big] \\
&= (B_0) \qquad (10.9)
\end{aligned}
$$

$$
\begin{aligned}
&= \prod M(0,2,4,6,8,10,12,14) \\
&= (M_0 \cdot M_2 \cdot M_4 \cdot M_6 \cdot M_8 \cdot M_{10} \cdot M_{12} \cdot M_{14}) \\
&= \Big[(B_3 + B_2 + B_1 + B_0) \cdot (B_3 + B_2 + \overline{B_1} + B_0) \cdot (B_3 + \overline{B_2} + B_1 + B_0) \cdot (B_3 + \overline{B_2} + \overline{B_1} + B_0) \cdot \\
&\quad (\overline{B_3} + B_2 + B_1 + B_0) \cdot (\overline{B_3} + B_2 + \overline{B_1} + B_0) \cdot (\overline{B_3} + \overline{B_2} + B_1 + B_0) \cdot (\overline{B_3} + \overline{B_2} + \overline{B_1} + B_0)\Big] \\
&= (B_0) \qquad (10.10)
\end{aligned}
$$

10.7 Gerador e detector de paridade

- Dado um operando de $(N-1)$ *bits*, o circuito gera um enésimo *bits*, de tal forma que o total de valores "1" seja par ou ímpar, conforme definido.

- Dado um operando de N *bits*, o circuito indica se o número de valores "1" é par ou ímpar, conforme definido.

10.8 Multiplexador e demultiplexador

- Um multiplexador é um circuito combinacional com N_1 entradas e 1 saída, controlado por N_2 sinais de controle. De acordo com o padrão de valores aplicados nos sinais de controle, uma das entradas é copiada para a saída.

- Um demultiplexador é um circuito combinacional com 1 entrada e N_1 saídas, controlado por N_2 sinais de controle. De acordo com o padrão de valores aplicados nos sinais de controle, a entrada é copiada para uma das saídas, enquanto as demais assumem um valor lógico/booleano fixo.

- A fim de se utilizar toda a funcionalidade do circuito implementado, normalmente é empregada a relação $N_1 = 2^{N_2}$.

10.8.1 Exemplos de projeto de multiplexador

A seguir, são apresentados os exemplos de projeto de dois multiplexadores, denominados de MUX 2x1 e MUX 4x1.

MUX 2x1

Um multiplexador com 2 entradas binárias (E_0 e E_1), 1 sinal de controle (C_0) e 1 saída (S), também denominado de MUX 2x1, é definido por

$$S = \begin{cases} E_0 & , \quad C_0 = 0 \\ E_1 & , \quad C_0 = 1 \end{cases} ,$$

cuja tabela verdade é apresentada na Tabela 10.2.

C_0	E_1	E_0	S
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Tabela 10.2: Tabela verdade do MUX 2x1.

Equacionando-se a saída por uma composição de mintermos de 3 variáveis (C_0, E_1 e E_0), obtém-se

$$\begin{aligned} S &= m_1 + m_3 + m_6 + m_7 \\ &= (\overline{C_0} \cdot \overline{E_1} \cdot E_0) + (\overline{C_0} \cdot E_1 \cdot E_0) + (C_0 \cdot E_1 \cdot \overline{E_0}) + (C_0 \cdot E_1 \cdot E_0) \\ &= \left[(\overline{C_0} \cdot E_0) \cdot (\overline{E_1} + E_1)\right] + \left[(C_0 \cdot E_1) \cdot (\overline{E_0} + E_0)\right] \\ &= (\overline{C_0} \cdot E_0) + (C_0 \cdot E_1) \ . \end{aligned} \tag{10.11}$$

Equacionando-se a saída por uma composição de maxtermos de 3 variáveis (C_0, E_1 e E_0), obtém-se

$$\begin{aligned} S &= M_0 + M_2 + M_4 + M_5 \\ &= (C_0 + E_1 + E_0) \cdot (C_0 + \overline{E_1} + E_0) \cdot (\overline{C_0} + E_1 + E_0) \cdot (\overline{C_0} + E_1 + \overline{E_0}) \\ &= \left[(C_0 + E_0) + (E_1 \cdot \overline{E_1})\right] \cdot \left[(\overline{C_0} + E_1) + (E_0 \cdot \overline{E_0})\right] \\ &= (C_0 + E_0) \cdot (\overline{C_0} + E_1) \ . \end{aligned} \tag{10.12}$$

As formas simplificadas das Equações (10.11) e (10.12) geram a tabela verdade simplificada que é apresentada na Tabela 10.3.

C_0	E_1	E_0	S
0	X	0	0
0	X	1	1
1	0	X	0
1	1	X	1

Tabela 10.3: Tabela verdade simplificada do MUX 2x1.

Podem-se ainda reescrever as formas simplificadas das Equações (10.11) e (10.12) por meio de mintermos e maxtermos de 1 variável (C_0), o que resulta em

$$\begin{aligned} S &= (m_0 \cdot E_0) + (m_1 \cdot E_1) \\ &= (M_0 + E_0) \cdot (M_1 + E_1) \ . \end{aligned} \tag{10.13}$$

MUX 4x1

Um multiplexador com 4 entradas binárias (E_0, E_1, E_2 e E_3), 2 sinais de controle (C_0 e C_1) e 1 saída (S), também denominado de MUX 4x1, é definido por

$$S = \begin{cases} E_0 & , \ C_1C_0 = 00 \\ E_1 & , \ C_1C_0 = 01 \\ E_2 & , \ C_1C_0 = 10 \\ E_3 & , \ C_1C_0 = 11 \end{cases} ,$$

cuja tabela verdade simplificada é apresentada na Tabela 10.4.

Equacionando-se a saída a partir da tabela verdade simplificada, obtém-se

$$\begin{aligned} S &= (\overline{C_1} \cdot \overline{C_0} \cdot E_0) + (\overline{C_1} \cdot C_0 \cdot E_1) + (C_1 \cdot \overline{C_0} \cdot E_2) + (C_1 \cdot C_0 \cdot E_3) \\ &= (C_1 + C_0 + E_0) \cdot (C_1 + \overline{C_0} + E_1) \cdot (\overline{C_1} + C_0 + E_2) \cdot (\overline{C_1} + \overline{C_0} + E_3) \ . \end{aligned} \tag{10.14}$$

Podem-se ainda reescrever as formas simplificadas da Equação (10.14) por meio de mintermos e maxtermos de 2 variáveis (C_1 e C_0), o que resulta em

$$\begin{aligned} S &= (m_0 \cdot E_0) + (m_1 \cdot E_1) + (m_2 \cdot E_2) + (m_3 \cdot E_3) \\ &= (M_0 + E_0) \cdot (M_1 + E_1) \cdot (M_2 + E_2) \cdot (M_3 + E_3) \ . \end{aligned} \tag{10.15}$$

C_1	C_0	E_3	E_2	E_1	E_0	S
0	0	X	X	X	0	0
0	0	X	X	X	1	1
0	1	X	X	0	X	0
0	1	X	X	1	X	1
1	0	X	0	X	X	0
1	0	X	1	X	X	1
1	1	0	X	X	X	0
1	1	1	X	X	X	1

Tabela 10.4: Tabela verdade simplificada do MUX 4x1.

10.8.2 Exemplos de projeto de demultiplexador

A seguir, são apresentados os exemplos de projeto de dois demultiplexadores, denominados de DEMUX 2x1 e DEMUX 4x1.

DEMUX 2x1

Um demultiplexador com 1 entrada binária (E), 1 sinal de controle (C_0) e 2 saídas (S_0 e S_1), também denominado de DEMUX 2x1, é definido por

$$S_0 = \begin{cases} E & , \quad C_0 = 0 \\ 0 & , \quad C_0 = 1 \end{cases}$$

e

$$S_1 = \begin{cases} 0 & , \quad C_0 = 0 \\ E & , \quad C_0 = 1 \end{cases} ,$$

cuja tabela verdade é apresentada na Tabela 10.5.

C_0	E	S_0	S_1
0	0	0	0
0	1	1	0
1	0	0	0
1	1	0	1

Tabela 10.5: Tabela verdade do DEMUX 2x1.

Equacionando-se as saídas por uma composição de mintermos de 2 variáveis (C_0 e E), obtém-se

$$S_0 = m_1 = (\overline{C_0} \cdot E) \tag{10.16}$$

e

$$S_1 = m_3 = (C_0 \cdot E) \ . \tag{10.17}$$

Equacionando-se as saídas por uma composição de maxtermos de 2 variáveis (C_0 e E), obtém-se

$$\begin{aligned} S_0 &= M_0 + M_2 + M_3 \\ &= (C_0 + E) \cdot (\overline{C_0} + E) \cdot (\overline{C_0} + \overline{E}) \\ &= (C_0 + E) \cdot (\overline{C_0} + E) \cdot (\overline{C_0} + E) \cdot (\overline{C_0} + \overline{E}) \\ &= \left[(C_0 \cdot \overline{C_0}) + (E)\right] \cdot \left[(\overline{C_0}) + (E \cdot \overline{E})\right] \\ &= (E) \cdot (\overline{C_0}) \ . \end{aligned} \tag{10.18}$$

e

$$\begin{aligned} S_1 &= M_0 + M_1 + M_2 \\ &= (C_0 + E) \cdot (C_0 + \overline{E}) \cdot (\overline{C_0} + E) \\ &= (C_0 + E) \cdot (C_0 + \overline{E}) \cdot (\overline{C_0} + E) \cdot (C_0 + E) \\ &= \left[(C_0) + (E \cdot \overline{E})\right] \cdot \left[(\overline{C_0} \cdot C_0) + (E)\right] \\ &= (C_0) \cdot (E) \ . \end{aligned} \tag{10.19}$$

Podem-se ainda reescrever as Equações (10.16) a (10.19) por meio de mintermos de 1 variável (C_0), o que resulta em

$$S_0 = (m_0 \cdot E) \tag{10.20}$$

e

$$S_1 = (m_1 \cdot E) \ . \tag{10.21}$$

DEMUX 4x1

Um demultiplexador com 1 entrada binária (E), 2 sinais de controle (C_0 e C_1) e 4 saídas (S_0, S_1, S_2 e S_3), também denominado de DEMUX 4x1, é definido por

$$S_0 = \begin{cases} E & , \quad C_1C_0 = 00 \\ 0 & , \quad \text{caso contrário} \end{cases} ,$$

$$S_1 = \begin{cases} E & , \quad C_1C_0 = 01 \\ 0 & , \quad \text{caso contrário} \end{cases} ,$$

$$S_2 = \begin{cases} E & , \quad C_1C_0 = 10 \\ 0 & , \quad \text{caso contrário} \end{cases}$$

e

$$S_3 = \begin{cases} E & , \quad C_1C_0 = 11 \\ 0 & , \quad \text{caso contrário} \end{cases} ,$$

cuja tabela verdade é apresentada na Tabela 10.6.

C_1	C_0	E	S_0	S_1	S_2	S_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

Tabela 10.6: Tabela verdade do DEMUX 4x1.

Equacionando-se as saídas por uma composição de mintermos de 3 variáveis (C_1, C_0 e E), obtém-se

$$S_0 = m_1 = (\overline{C_1} \cdot \overline{C_0} \cdot E) \ , \tag{10.22}$$

$$S_1 = m_3 = (\overline{C_1} \cdot C_0 \cdot E) \ , \tag{10.23}$$

$$S_2 = m_5 = (C_1 \cdot \overline{C_0} \cdot E) \tag{10.24}$$

e

$$S_3 = m_7 = (C_1 \cdot C_0 \cdot E) \ . \tag{10.25}$$

Podem-se ainda reescrever as Equações (10.22) a (10.25) por meio de mintermos de 2 variáveis (C_1 e C_0), o que resulta em

$$S_0 = (m_0 \cdot E) \ , \tag{10.26}$$

$$S_1 = (m_1 \cdot E) \ , \tag{10.27}$$

$$S_2 = (m_2 \cdot E) \tag{10.28}$$

e

$$S_3 = (m_3 \cdot E) \ . \tag{10.29}$$

10.9 Codificador e decodificador de endereço

- Um codificador/decodificador de endereço (*address coder/decoder*) também é conhecido como um codificador/decodificador de linha (*line coder/decoder*).

- Um codificador de endereço é um circuito combinacional com N_1 entradas e N_2 saídas. Apenas uma das entradas assumirá um valor lógico/booleano, enquanto todas as demais assumirão o valor lógico/booleano complementar. O padrão de valores gerado na saída (interpretado como um endereço), será referente à entrada com valor lógico/booleano diferente das demais entradas.

- Um decodificador de endereço é um circuito combinacional com N_2 entradas e N_1 saídas. De acordo com o padrão de valores aplicados na entrada (interpretado como um endereço), uma das saídas assumirá um valor lógico/booleano, enquanto todas as demais assumirão o valor lógico/booleano complementar.

- A fim de se utilizar toda a funcionalidade do circuito implementado, normalmente é empregada a relação $N_1 = 2^{N_2}$.

- De uma forma geral, ambos os blocos funcionais podem ser interpretados como conversores de códigos. Um codificador de endereço pode ser visto como um conversor de código *one-hot* para código binário. Por sua vez, um decodificador de endereço pode ser dito um conversor de código binário para código *one-hot*.

- Um decodificador de endereço pode também ser interpretado como um demultiplexador cuja entrada recebe o valor lógico/booleano desejado para a saída a ser selecionada. Logo, pode-se projetá-lo a partir de um demultiplexador, com as possíveis simplificações.

- Dependendo dos valores lógicos/booleanos adotados para as saídas, um decodificador de endereço ainda pode ser interpretado como um gerador de mintermos ou de maxtermos.

10.10 Codificador de prioridade

- Um codificador de prioridades é um circuito combinacional com R entradas, numeradas de E_0 a E_{R-1}, onde cada uma delas é associada a uma requisição.
- Uma requisição é representada por um valor lógico/booleano.
- As requisições são independentes entre si, podendo ocorrer um total de 0 a R requisições simultâneas.
- As prioridades das requisições são organizadas na ordem crescente ou decrescente dos números das entradas.
- Em uma primeira versão, o circuito apresenta R saídas. Nesse caso, a saída S_k, para $0 \leq k \leq (R-1)$, deve assumir um valor lógico/booleano somente quando houver uma requisição na entrada E_k e ela for a de mais alta prioridade no momento, enquanto todas as demais saídas assumirão o valor lógico/booleano complementar.
- Em uma outra versão, o circuito apresenta N saídas, onde $R = 2^N$. Aqui, o padrão de valores gerado na saída (interpretado como um endereço), será referente à entrada onde ocorre a requisição de mais alta prioridade.

10.10.1 Exemplos de projeto de codificador de prioridade com um número de saídas igual ao número de entradas

A seguir, são apresentados exemplos de projeto de codificador de prioridade com um número de saídas igual ao número de entradas.

Codificador de prioridade com 2 entradas e 2 saídas

A Tabela 10.7 apresenta a definição de um codificador de prioridade com 2 entradas e 2 saídas, com prioridade crescente.

E_1	E_0	S_1	S_0
0	0	0	0
0	1	0	1
1	X	1	0

Tabela 10.7: Tabela verdade simplificada do codificador de prioridade com 2 entradas e 2 saídas.

A partir da definição do codificador, pode-se propor o seguinte equacionamento:

$$S_1 = 1 \cdot E_1 = \overline{(0)} \cdot E_1$$

e

$$S_0 = \overline{E_1} \cdot E_0 = \overline{(E_1)} \cdot E_0 \ .$$

Codificador de prioridade com 3 entradas e 3 saídas

A Tabela 10.8 apresenta a definição de um codificador de prioridade com 3 entradas e 3 saídas, com prioridade crescente.

A partir da definição do codificador, pode-se propor o seguinte equacionamento:

$$S_2 = 1 \cdot E_2 = \overline{(0)} \cdot E_2 \ ,$$

$$S_1 = \overline{E_2} \cdot E_1 = \overline{(E_2)} \cdot E_1$$

e

$$S_0 = \overline{E_2} \cdot \overline{E_1} \cdot E_0 = \overline{(E_2 + E_1)} \cdot E_0 \ .$$

E_2	E_1	E_0	S_2	S_1	S_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	X	0	1	0
1	X	X	1	0	0

Tabela 10.8: Tabela verdade simplificada do codificador de prioridade com 3 entradas e 3 saídas.

Codificador de prioridade com R entradas e R saídas

A Tabela 10.9 apresenta a definição de um codificador de prioridade com R entradas e R saídas, com prioridade crescente.

E_{R-1}	$\cdots$	E_2	E_1	E_0	S_{R-1}	$\cdots$	S_2	S_1	S_0
0	$\cdots$	0	0	0	0	$\cdots$	0	0	0
0	$\cdots$	0	0	1	0	$\cdots$	0	0	1
0	$\cdots$	0	1	X	0	$\cdots$	0	1	0
0	$\cdots$	1	X	X	0	$\cdots$	1	0	0
$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
1	$\cdots$	X	X	X	1	$\cdots$	0	0	0

Tabela 10.9: Tabela verdade simplificada do codificador de prioridade com R entradas e R saídas.

A partir da definição do codificador, pode-se propor o seguinte equacionamento:

$$S_{R-1} = 1 \cdot E_{R-1} = \overline{(0)} \cdot E_{R-1} \ ,$$

$$S_{R-2} = \overline{E_{R-1}} \cdot E_{R-2} = \overline{(E_{R-1})} \cdot E_{R-2} \ ,$$

$$S_{R-3} = \overline{E_{R-1}} \cdot \overline{E_{R-2}} \cdot E_{R-3} = \overline{(E_{R-1} + E_{R-2})} \cdot E_{R-3}$$

$$\vdots$$

$$S_2 = \overline{E_{R-1}} \cdot \ \cdots \ \cdot \overline{E_3} \cdot E_2 = \overline{(E_{R-1} + \cdots + E_3)} \cdot E_2 \ ,$$

$$S_1 = \overline{E_{R-1}} \cdot \ \cdots \ \cdot \overline{E_2} \cdot E_1 = \overline{(E_{R-1} + \cdots + E_2)} \cdot E_1$$

e

$$S_0 = \overline{E_{R-1}} \cdot \ \cdots \ \cdot \overline{E_1} \cdot E_0 = \overline{(E_{R-1} + \cdots + E_1)} \cdot E_0 \ .$$

Codificador de prioridade com R entradas e R saídas: solução modular

O codificador de prioridade com R entradas e R saídas aceita uma solução modular. Pode-se propor um bloco funcional básico, a ser utilizado para cada entrada, com a função de bloqueio da requisição. Um sinal para a indicação de bloqueio (B) é propagado entre cada par de blocos consecutivos.

Supondo-se um codificador com prioridade crescente, as entradas do bloco básico são: a requisição E_k e o sinal de bloqueio B_k, onde $0 \leq k \leq (R-1)$ e

$$B_k = \begin{cases} 0 & , \text{ não bloqueia a requisição k.} \\ 1 & , \text{ bloqueia a requisição k.} \end{cases} .$$

As saídas do bloco básico são: a requisição efetiva S_k e o sinal de bloqueio B_{k-1}.

A Tabela 10.10 apresenta a definição do bloco básico para o codificador de prioridade com R entradas e R saídas, modular, com prioridade crescente. Da tabela, conclui-se que

$$S_k = \overline{B_k} \cdot E_k$$

e

$$B_{k-1} = (B_k + E_k) \ .$$

B_k	E_k	S_k	B_{k-1}
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	1

Tabela 10.10: Tabela verdade do bloco básico para o codificador de prioridade com R entradas e R saídas modular, com prioridade crescente.

10.11 Ordenador, separador ou agrupador binário

- O circuito possui N *bits* na entrada e N *bits* na saída.
- Os *bits* da entrada devem ser separados, de tal forma que, na saída, todos os *bits* de valor 1 sejam agrupados próximos ao MSB (*Most Significant Bit*) e todos os *bits* de valor 0 sejam agrupados próximos ao LSB (*Least Significant Bit*), ou vice-versa.
- Estruturas modulares de N *bits* podem ser obtidas a partir de uma célula básica de 2 *bits*. A seguir, são comentadas a célula básica e duas dessas estruturas modulares.

10.11.1 Célula básica para um separador binário

A célula básica para um separador binário de N bits é um separador de $N = 2$ bits. Assim, um dos dois *bits* da saída deve receber prioritariamente um dos valores 1 da entrada, enquanto o outro *bit* deve receber prioritariamente um dos valores 0 da entrada. Dada uma entrada $E = \{X, Y\}$ e uma saída $S = \{S_1, S_2\}$, onde S_1 deverá assumir o valor 1 caso haja pelo menos um valor 1 na entrada e S_2 só deverá assumir o valor 1 se houver dois valores 1 na entrada, podem-se definir os seguintes mapeamentos:

$$S_1 = (X + Y)$$

e

$$S_2 = (X \cdot Y) \ .$$

10.11.2 Estrutura modular com isolamento de um *bit* por vez

Nessa estrutura, os N *bits* da entrada são aplicados a um primeiro conjunto de $(N-1)$ células básicas (CB's), ligadas em cascata, com o propósito de separar o primeiro dos valores desejados. Por sua vez, os $(N-1)$ *bits* provenientes do primeiro conjunto de CB's são ligados a um segundo conjunto de $(N-2)$ CB's, ligadas em cascata, com o propósito de separar o segundo dos valores desejados. Conjuntos similares, com funções similares, contendo $(N-k)$ CB's, completam a estrutura. O último conjunto, onde $k = (N-1)$, é formado por uma única CB e separa os dois *bits* finais. Com essa organização, é construída uma estrutura modular, com o formato de uma matriz triangular, contendo um total de $(N-1)N/2$ CB's. Pode-se dizer que a estrutura final é uma ligação em cascata de conjuntos que são formados, cada um deles, por uma ligação em cascata de CB's.

10.11.3 Estrutura modular com sucessivas trocas em paralelo

Essa estrutura é formada por uma ligação em cascata de N camadas C_k, $1 \leq k \leq N$, que são formadas, cada uma delas, por um conjunto de CB's que operam em paralelo. Para N par, as camadas com valores ímpares de k são formadas por $(N/2)$ CB's, enquanto as camadas com valores pares de k são formadas por $[(N/2)-1]$ CB's. Para N ímpar, todas as camadas são formadas por $(N-1)/2$ CB's. Portanto, nesse caso também é empregado um total de $(N-1)N/2$ CB's. A estrutura obtida nessa construção apresenta a capacidade potencial de testar e de comutar a posição de todos os *bits* da entrada, dois a dois. Assim, um *bit* que ocupe uma posição extrema na entrada pode ser permutado várias vezes até atingir o extremo oposto na saída. Da mesma forma, os *bits* em posições medianas podem ser adequadamente testados e, se necessário, permutados.

10.12 Deslocadores (*shifters*)

- O circuito possui N *bits* na entrada e N *bits* na saída.
- O padrão de saída é uma versão deslocada do padrão de entrada.
- O deslocamento pode ser de qualquer quantidade e para qualquer um dos dois sentidos.
- Três tipos de deslocamento são comumente implementados:
  - Deslocamento lógico: as posições vazias são ocupadas pelo valor booleano "0".
  - Deslocamento aritmético: as posições vazias são ocupadas pelo valor booleano da extremidade mais próxima.
  - Deslocamento circular ou rotação: cada posição vazia é ocupada pelo valor removido.
- O termo *barrel shifter* é usado tanto para circuitos que implementam apenas a rotação como também para aqueles que realizam os demais deslocamentos.

10.13 Somadores em binário puro

A seguir, são apresentados somadores de 2 operandos, codificados em binário puro.

10.13.1 *Half-adder* (HA)

- Um *half-adder* é um circuito combinacional com 2 entradas e 2 saídas, todas de 1 *bit*, que se comporta como um somador de 2 operandos, onde uma das saídas é o resultado da soma e a outra é o sinal de "vai-um" de saída (*carry out*).
- Equações básicas de um *half-adder*:

$$C_o = (A \cdot B) \ .$$
$$S = \left(\overline{A} \cdot B\right) + \left(A \cdot \overline{B}\right) = (A \oplus B) \ .$$

10.13.2 *Full-adder* (FA)

- Um *full-adder* é um circuito combinacional com 3 entradas e 2 saídas, todas de 1 *bit*, que se comporta como um somador de 2 operandos, onde uma das entradas é sinal de "vai-um" de entrada (*carry in*), enquanto uma das saídas é o resultado da soma e a outra é o sinal de "vai-um" de saída (*carry out*).
- Empregando-se diferentes elementos constituintes, diversas implementações podem ser encontradas para um *full-adder*. Alguns exemplos de tais elementos são os seguintes:
  - SOP mínima:

$$C_o = (A \cdot B) + (A \cdot C_i) + (B \cdot C_i) \ .$$
$$S = \left(\overline{A} \cdot B \cdot \overline{C_i}\right) + \left(A \cdot \overline{B} \cdot \overline{C_i}\right) + \left(\overline{A} \cdot \overline{B} \cdot C_i\right) + (A \cdot B \cdot C_i) = (A \oplus B) \oplus C_i \ .$$

  - Bloco HA:

$$C_{o_1} = (A \cdot B) = G \ .$$
$$S_1 = (A \oplus B) = P \ .$$
$$C_{o_2} = (S_1 \cdot C_i) = (P \cdot C_i) \ .$$
$$S_2 = (S_1 \oplus C_i) = (P \oplus C_i) \ .$$
$$C_o = C_{o_1} + C_{o_2} = G + (P \cdot C_i) \ .$$
$$S = S_2 = (P \oplus C_i) \ .$$

  - Bloco AOI (com diferentes composições):

$$\overline{C_o} = \overline{(A \cdot B) + (A \cdot C_i) + (B \cdot C_i)} = \overline{(A \cdot B) + [(A + B) \cdot C_i]} \ .$$
$$\overline{S} = \overline{(A \cdot B \cdot C_i) + \left[(A + B + C_i) \cdot \overline{C_o}\right]} = \overline{[(A \cdot B) \cdot C_i] + \left[(A + B + C_i) \cdot \overline{C_o}\right]} \ .$$

  - Multiplexador:

$$MUX_1 = \left(\overline{C_i} \cdot \overline{A}\right) + (C_i \cdot A) \ .$$
$$MUX_2 = \left(B \cdot \overline{MUX_1}\right) + (A \cdot MUX_1) \ .$$
$$MUX_3 = \left(\overline{MUX_2} \cdot \overline{MUX_1}\right) + (B \cdot MUX_1) \ .$$
$$\overline{C_o} = \overline{MUX_2} \ .$$
$$C_o = MUX_2 \ .$$
$$S = MUX_3 \ .$$

10.13.3 *Ripple-carry adder* (RCA) ou *carry propagate adder* (CPA)

- Um *ripple-carry adder* ou um *carry propagate adder* é um circuito combinacional com 1 entrada de 1 *bit*, 2 entradas de N *bits*, 1 saída de N *bits* e 1 saída de 1 *bit*. que se comporta como um somador de 2 operandos de N *bits*, que recebe um sinal de "vai-um" de entrada (*carry in*) e gera um sinal de "vai-um" de saída (*carry out*), além do resultado de N *bits*.

- Na sua forma original, o circuito é modular, sendo formado por uma seqüência de N blocos do tipo *full-adder*, interligados pelos sinais "vai-um" de entrada (*carry in*) e "vai-um" de saída (*carry out*) de cada par de blocos, o que justifica o seu nome.

- A facilidade de projeto é contrabalanceada pelo tempo de estabilização do resultado, que é lento, uma vez que deve-se esperar pelo tempo total de N propagações do sinal de "vai-um" (*carry*), acrescido do tempo de operação do último *full-adder*.

- Algumas implementações do bloco FA necessitam de um inversor tanto na saída C_o quanto na saída S. A remoção desses inversores produz um bloco $\overline{FA}$ que apresenta uma redução de tempo na geração do *carry*. Utilizando-se esse FA modificado para implementar o somador RCA, pode-se reduzir o tempo de caminho crítico do somador, que é o tempo total de propagação de *carry*. Porém, para o correto funcionamento do somador RCA, inversores extras devem ser anexados. Considerando-se que os operandos são formados pelos N dígitos $\boldsymbol{Op} = [\ d_{N-1}\ d_{N-2}\ \cdots\ d_1\ d_0\ ]$, deve-se adicionar um inversor na saída dos blocos $\overline{FA}$ com índice zero e par. Por sua vez, deve-se adicionar um inversor em cada uma das entradas dos blocos $\overline{FA}$ com índice ímpar. Se N for ímpar, deve-se acrescentar um inversor na saída *carry out* do somador RCA.

10.13.4 *Carry lookahead adder* (CLA)

- Um *carry lookahead adder* é um circuito combinacional com 1 entrada de 1 *bit*, 2 entradas de N *bits*, 1 saída de N *bits* e 1 saída de 1 *bit*. que se comporta como um somador de 2 operandos de N *bits*, que recebe um sinal de "vai-um" de entrada (*carry in*) e gera um sinal de "vai-um" de saída (*carry out*), além do resultado de N *bits*.

- Esse é um dos diversos somadores que busca uma melhoria de eficiência através da diminuição do tempo de caminho crítico, associado à propagação interna do *carry*.

- O circuito também é formado por uma seqüência de N blocos do tipo *full-adder*. Porém, a característica básica desse somador é que a geração do sinal C_o em cada estágio é feita localmente, ao invés de esperar a propagação de *carry* pelos estágios anteriores.

- Supondo-se que o sinal C_k é o *carry* de ligação entre os estágios $k-1$ e k, as equações originais do somador, para $N = 4$, são:

$$\begin{aligned} C_0 &= C_{in} \\ C_1 &= G_0 + (P_0 \cdot C_0) \\ C_2 &= G_1 + (P_1 \cdot C_1) \\ C_3 &= G_2 + (P_2 \cdot C_2) \\ C_4 &= G_3 + (P_3 \cdot C_3) \ = \ C_{out}\ . \end{aligned}$$

- Por sua vez, supondo-se que o sinal S_k é a saída do estágio k, as equações originais do somador, para $N = 4$, são:

$$\begin{aligned} S_0 &= (P_0 \oplus C_0) \\ S_1 &= (P_1 \oplus C_1) \\ S_2 &= (P_2 \oplus C_2) \\ S_3 &= (P_3 \oplus C_3) \ . \end{aligned}$$

- Para obter a citada aceleração do tempo de caminho crítico, são definidas as funções *generate* de grupo $G_{i:j}$ e *propagate* de grupo $P_{i:j}$, definidas a seguir, de tal forma que os sinais C_k são reescritos como

$$\begin{aligned} C_1 &= G_0 + (P_0 \cdot C_0) \\ &= G_{0:0} + (P_{0:0} \cdot C_0) \end{aligned}$$

$$\begin{aligned} C_2 &= G_1 + (P_1 \cdot C_1) \\ &= G_1 + (P_1 \cdot (G_0 + (P_0 \cdot C_0))) \\ &= (G_1 + P_1 \cdot G_0) + (P_1 \cdot P_0) \cdot C_0 \\ &= G_{1:0} + (P_{1:0} \cdot C_0) \end{aligned}$$

$$\begin{aligned} C_3 &= G_2 + (P_2 \cdot C_2) \\ &= G_2 + (P_2 \cdot (G_1 + (P_1 \cdot (G_0 + (P_0 \cdot C_0))))) \\ &= (G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0) + (P_2 \cdot P_1 \cdot P_0) \cdot C_0 \\ &= G_{2:0} + (P_{2:0} \cdot C_0) \end{aligned}$$

$$\begin{aligned} C_4 &= G_3 + (P_3 \cdot C_3) \\ &= G_3 + (P_3 \cdot (G_2 + (P_2 \cdot (G_1 + (P_1 \cdot (G_0 + (P_0 \cdot C_0))))))) \\ &= (G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0) + (P_3 \cdot P_2 \cdot P_1 \cdot P_0) \cdot C_0 \\ &= G_{3:0} + (P_{3:0} \cdot C_0) \ , \end{aligned}$$

onde

$$G_k = (A_k \cdot B_k)$$

e

$$P_k = (A_k \oplus B_k) \ ,$$

lembrando-se ainda que, para o cálculo dos sinais C_k, pode-se utilizar

$$P_k^+ = (A_k + B_k) \ .$$

10.14 Subtratores em binário puro

- A seguir, são apresentados subtratores de 2 operandos, codificados em binário puro.

10.14.1 *Half-subtractor* (HS)

- Um *half-subtractor* é um circuito combinacional com 2 entradas e 2 saídas, todas de 1 *bit*, que se comporta como um subtrator de 2 operandos, onde uma das saídas é o resultado da subtração e a outra é o sinal de "veio-um" de saída (*borrow out*).

10.14.2 *Full-subtractor* (FS)

- Um *full-subtractor* é um circuito combinacional com 3 entradas e 2 saídas, todas de 1 *bit*, que se comporta como um subtrator de 2 operandos, onde uma das entradas é sinal de "veio-um" de entrada (*borrow in*), enquanto uma das saídas é o resultado da subtração e a outra é o sinal de "veio-um" de saída (*borrow out*).

10.14.3 *Ripple-borrow subtractor* (RBS) ou *borrow propagate subtractor* (BPS)

- Um *ripple-borrow subtractor* ou um *borrow propagate subtractor* é um circuito combinacional com 1 entrada de 1 *bit*, 2 entradas de N *bits*, 1 saída de N *bits* e 1 saída de 1 *bit*. que se comporta como um subtrator de 2 operandos de N *bits*, que recebe um sinal de "veio-um" de entrada (*borrow in*) e gera um sinal de "veio-um" de saída (*borrow out*), além do resultado de N *bits*.

- Na sua forma original, o circuito é modular, sendo formado por uma seqüência de N blocos do tipo *full-subtractor*, interligados pelos sinais "veio-um" de entrada (*borrow in*) e "veio-um" de saída (*borrow out*) de cada par de blocos, o que justifica o seu nome.

- A facilidade de projeto é contrabalanceada pelo tempo de estabilização do resultado, que é lento, uma vez que deve-se esperar pelo tempo total de N propagações do sinal de "veio-um" (*borrow*), acrescido do tempo de operação do último *full-subtractor*.

10.15 Somadores/subtratores configuráveis (binário puro)

- A seguir, são apresentados somadores/subtratores configuráveis de 2 operandos, codificados em binário puro.

10.15.1 *Full-adder-subtractor*

- Um *full-adder-subtractor* é um circuito combinacional com 3 entradas e 2 saídas, todas de 1 *bit*, que se comporta como um somador/subtrator de 2 operandos, de acordo com uma entrada de configuração, onde uma das entradas é sinal de "vai-um / veio-um" de entrada (*carry in /borrow in*), enquanto uma das saídas é o resultado da soma/subtração e a outra é o sinal de "vai-um / veio-um" de saída (*carry in / borrow out*).

- Empregando-se SOP mínima, obtém-se:

$$\begin{aligned} V_o(Op, V_i, A, B) &= \sum m(3,5,6,7,9,12,13,15) \\ &= \left(\overline{Op}\cdot A\cdot B\right) + \left(Op\cdot\overline{A}\cdot B\right) + \left(\overline{Op}\cdot A\cdot V_i\right) + \left(Op\cdot\overline{A}\cdot V_i\right) + (B\cdot V_i) \\ &= ((Op\oplus A)\cdot B) + ((Op\oplus A)\cdot V_i) + (B\cdot V_i)\ . \end{aligned}$$

$$\begin{aligned} S(Op, V_i, A, B) &= \sum m(1,2,4,7,9,10,12,15) \\ &= \left(\overline{A}\cdot B\cdot\overline{V_i}\right) + \left(A\cdot\overline{B}\cdot\overline{V_i}\right) + \left(\overline{A}\cdot\overline{B}\cdot V_i\right) + (A\cdot B\cdot V_i) \\ &= (A\oplus B)\oplus V_i\ . \end{aligned}$$

- Equações genéricas alternativas (com portas lógicas):

$$\begin{aligned} V_o &= \left(\left(\left(\overline{Op}\cdot A\right)\cdot B\right)\cdot\overline{V_i}\right) + \left(\left(\left(\overline{Op}\cdot A\right)+B\right)\cdot V_i\right) + \\ &\quad \left(\left(\left(Op\cdot\overline{A}\right)\cdot B\right)\cdot\overline{V_i}\right) + \left(\left(\left(Op\cdot\overline{A}\right)+B\right)\cdot V_i\right) \\ &= \left(((Op\oplus A)\cdot B)\cdot\overline{V_i}\right) + (((Op\oplus A)+B)\cdot V_i)\ . \end{aligned}$$

- Equações genéricas alternativas (com MUX):

$$MUX_1 = \left(\overline{Op}\cdot A\right) + \left(Op\cdot\overline{A}\right) = (Op\oplus A)\ .$$

$$MUX_2 = \left((MUX_1\cdot B)\cdot\overline{V_i}\right) + ((MUX_1+B)\cdot V_i)\ .$$

$$V_0 = MUX_2\ .$$

10.16 Incrementador e decrementador em binário puro

- Os circuitos incrementador e decrementador são versões simplificadas dos circuitos somador e subtrator, respectivamente.
- Eles adicionam ou subtraem, respectivamente, uma unidade ao *bit* menos significativo (LSB) do operando a ser incrementado ou decrementado.
- Assim sendo, pode-se adotar um projeto por uso de blocos pré-existentes.
- Por exemplo:
  - Um incrementador pode ser obtido de um *ripple-carry adder* baseado em *full-adder*, aplicando-se $\boldsymbol{Op_2} = [\ 0\ 0\ \cdots\ 0\ 0\ ] = 0$ e $C_{in} = 1$, além de realizar as conseqüentes simplificações no circuito.
  - Um decrementador pode ser obtido de um *ripple-borrow subtractor* baseado em *full-subtractor*, aplicando-se $\boldsymbol{Op_2} = [\ 0\ 0\ \cdots\ 0\ 0\ ] = 0$ e $B_{in} = 1$, além de realizar as conseqüentes simplificações no circuito.
  - Um decrementador pode ser obtido de um *ripple-carry adder* baseado em *full-adder*, aplicando-se $\boldsymbol{Op_2} = [\ 1\ 1\ \cdots\ 1\ 1\ ] = -1$ e $C_{in} = 0$, além de realizar as conseqüentes simplificações no circuito.

10.17 Complementadores

- A seguir, são apresentados dois exemplos de conversores de códigos numéricos.

10.17.1 Complementador-a-1 (*bitwise implementation*)

- Um complementador-a-1 é um circuito combinacional com N entradas e N saídas, que realiza a conversão entre os códigos numéricos binário puro e complemento-a-1.
- Uma vez que, nessa conversão, as operações sobre os *bits* são independentes, um projeto modular elementar pode ser adotado.
- Para um complementador incondicional, são necessários apenas inversores.
- Para um complementador condicional, utiliza-se a porta lógica XOR como bloco básico.

10.17.2 Complementador-a-2

- Um complementador-a-2 é um circuito combinacional com N entradas e N saídas, que realiza a conversão entre os códigos numéricos binário puro e complemento-a-2.
- Uma vez que, nessa conversão, as operações sobre os *bits* não são independentes, diferentes técnicas podem ser adotadas, as quais são abordadas a seguir.

Complementador-a-1 + somador em binário puro

- Nesse caso, um complementador-a-2 é implementado usando um circuito complementador-a-1 em conjunto com um circuito somador em binário puro. Por sua vez, o circuito somador pode ser formado por somadores básicos de dois operandos ou pode ser apenas um incrementador.

Decrementador com saídas invertidas

- Para um complementador-a-2 incondicional, pode-se adotar um decrementador com as saídas invertidas. Isso é matematicamente comprovado por

$$(q)_{C2} = 2^N - |q| = 2^N - 1 + 1 - |q| = 2^N - 1 - (|q| - 1) = (|q| - 1)_{C1} \ .$$

Complementador-a-2 puro (*bit-scanning implementation*)

- Nesse caso, um complementador-a-2 é implementado usando um projeto modular.
- É realizada uma varredura do *bit* menos significativo (LSB) para o mais significativo (MSB). Enquanto não for encontrado o primeiro *bit* com valor "1", os *bits* com valor "0" são mantidos. Ao encontrar-se o primeiro *bit* com valor "1", este também é mantido. A partir daí, todos os demais *bits* têm os seus valores invertidos.
- No caso de um complementador incondicional, o módulo k recebe o *bit* B_k e um sinal F_{k-1} que indica se o primeiro valor "1" já foi encontrado, gerando o *bit* complementado C_k e o novo sinal F_k.
- No caso de um complementador condicional, pode-se adotar duas soluções básicas. Na primeira delas, é utilizado o complementador incondicional, com uma porta lógica AND externamente adicionada a cada módulo, para controlar a propagação do sinal de varredura F. A outra incorpora o sinal de controle diretamente no projeto do módulo.

10.18 Multiplicadores em binário puro

- A seguir, é apresentada uma possível implementação modular para um multiplicador de 2 operandos, em binário puro.

10.18.1 Multiplicador de 1 *bit*

- É facilmente demonstrável que um multiplicador de 2 operandos de 1 *bit* pode ser implementado por uma porta lógica AND.

10.18.2 Multiplicador de N *bits*

- Utilizando-se um projeto modular, pode-se mostrar que um multiplicador de 2 operandos de N *bits* pode ser implementado usando apenas multiplicadores de 1 *bit* e *full-adders*.

10.19 Detector de *overflow* em adição de valores codificados em complemento-a-2

Em implementações com aritmética de ponto fixo e tamanho de palavra finita, o resultado da adição de valores codificados em complemento-a-2 pode não ser representável, caracterizando uma situação de *overflow*.

A ocorrência de *overflow* pode ser detectada analisando-se alguns dos sinais externos ao bloco somador.

Conforme discutido em 9.6.3, a Equação (9.50) define uma possível forma de detecção, que é dada por

$$OF = \left(\overline{d_{s1}} \cdot \overline{d_{s2}} \cdot d_{sA}\right) + \left(d_{s1} \cdot d_{s2} \cdot \overline{d_{sA}}\right) \ ,$$

onde OF, d_{s1}, d_{s2} e d_{sA} representam o sinal indicativo de *overflow* e os dígitos de sinal do operando 1, do operando 2 e do resultado da adição, respectivamente.

10.20 Saturador (valor codificado em complemento-a-2)

Em implementações com aritmética de ponto fixo e tamanho de palavra finita, é comum a ocorrência de *overflow*.

O tratamento mais comumente empregado é a saturação do valor em *overflow* no valor máximo representável, positivo (0 1 1 1 $\cdots$ 1 1 1) ou negativo (1 0 0 0 $\cdots$ 0 0 0).

Em alguns casos, onde é exigido que os dados sejam interpretados como sendo estritamente menores que a unidade ($|x| < 1$), a solução é a saturação do valor em *overflow* no valor máximo representável imediatamente menor que a unidade, positivo (0 1 1 1 $\cdots$ 1 1 1) ou negativo (1 0 0 0 $\cdots$ 0 0 1).

Considerando-se uma representação dada por [B_{N-1} B_{N-2} B_{N-3} $\cdots$ B_2 B_1 B_0], e uma variável para seleção do tipo de saturação t_sat, de tal forma que $t\_sat = 0$ significa saturação em valores máximos e $t\_sat = 1$ significa saturação em valores máximos menores que a unidade, pode-se mostrar que um conjunto de equações que implementam tal solução é dado por

$$S_{N-1} = \left(\overline{OF} \cdot E_{N-1}\right) + \left(OF \cdot \overline{E_{N-1}}\right) \ ,$$

$$S_k = \left(\overline{OF} \cdot E_k\right) + \left(OF \cdot E_{N-1}\right)$$

e

$$S_0 = \left(\overline{OF} \cdot E_0\right) + (OF \cdot E_{N-1}) + (OF \cdot t\_sat) \ ,$$

onde OF, E e S, significam, respectivamente, um sinal de indicação de *overflow*, a entrada e a saída, bem como $(N-2) \leq k \leq 1$.

10.21 Comparadores da quantidade de dígitos

- A seguir, são apresentados exemplos de circuitos combinacionais utilizados para a comparação da quantidade de dígitos em operandos.

10.21.1 Comparador da quantidade de dígitos em um operando

- Dado um operando, com N dígitos, o circuito fornece 3 saídas, que indicam se o número de valores "0" é menor, igual ou maior que o número de valores "1".

10.21.2 Comparador da quantidade de dígitos em dois operandos

- Dados 2 operandos, com N dígitos cada, o circuito fornece 3 saídas, que indicam se o número de valores "0", ou "1", no operando Op_1 é menor, igual ou maior que no operando Op_2.

10.22 Comparadores numéricos de dois operandos

Supondo-se dois números, x e $y \in \mathbb{N}$, codificados em binário puro, com N dígitos, os mesmos podem assumir valores na faixa $[0; (2^N - 1)]$. Deseja-se compará-los e gerar três sinais binários que indiquem as seguintes condições: $x < y$, $x = y$ e $x > y$.

A seguir, são discutidas uma técnica para identificação de igualdade e três técnicas que podem ser usadas no projeto de um comparador genérico.

10.22.1 Identificador de igualdade

Para que duas cadeias de valores binários sejam iguais, elas devem ser iguais dígito a dígito. A igualdade entre dois dígitos pode ser testada com o operador XNOR $(x_k,\ y_k)$. Portanto, a igualdade entre duas cadeias de N dígitos pode ser testada por meio da seguinte relação:

$$\begin{aligned} \boldsymbol{I(x,y)} &= I(x_{N-1}, y_{N-1})\ AND\ \cdots\ I(x_1, y_1)\ AND\ I(x_0, y_0) \\ &= (x_{N-1}\ XNOR\ y_{N-1})\ AND\ \cdots\ (x_1\ XNOR\ y_1)\ AND\ (x_0\ XNOR\ y_0)\ . \end{aligned} \tag{10.30}$$

A Equação 10.30 pode ser implementada de forma modular (*bit scanning*), considerando-se que os módulos sejam definidos por

$$I(x_k, y_k) = (x_k\ XNOR\ y_k)\ AND\ I(x_{k+1}, y_{k+1})\ ,$$

para $(N-1) \leq k \leq 0$ e $I(x_N, y_N) = 1$.

10.22.2 Projeto modular (*bit scanning*)

Nesse caso, os dígitos d_k de x e de y são comparados um a um, do mais significativo (d_{N-1}) até o menos significativo (d_0), empregando-se N blocos comparadores de dígitos.

Além dos dígitos x_k e y_k, o bloco comparador dos dígitos d_k deve receber também outros dois sinais ($S_{1_{k+1}}$ e $S_{2_{k+1}}$), provenientes da comparação realizada no bloco de posição $k+1$. Como resultado, ele deve gerar dois novos sinais equivalentes (S_{1_k} e S_{2_k}), relativos à sua comparação, que deverão ser fornecidos ao bloco de posição $k-1$.

Os dois sinais de entrada do bloco de posição $k = (N-1)$ representam os dois sinais de entrada do comparador, devendo ser escolhidos adequadamente.

Os dois sinais de saída do bloco de posição $k = 0$ representam os dois sinais de saída do comparador, de tal forma que $S_1 = S_{1_0}$ e $S_2 = S_{2_0}$.

O terceiro sinal de saída do comparador (S_3) pode ser obtido pela combinação adequada dos outros dois.

Os sinais S_1 e S_2 podem ser escolhidos dentre as seguintes opções:

1. $S_1 = (x < y)$ e $S_2 = (x = y)$, onde $S_{1_N} = 0$ e $S_{2_N} = 1$.
2. $S_1 = (x < y)$ e $S_2 = (x > y)$, onde $S_{1_N} = 0$ e $S_{2_N} = 0$.
3. $S_1 = (x = y)$ e $S_2 = (x > y)$, onde $S_{1_N} = 1$ e $S_{2_N} = 0$.

Em seguida, o sinal S_3 pode gerado por

$$S_3 = (NOT\ S_1)\ AND\ (NOT\ S_2) = S_1\ NOR\ S_2\ .$$

Opção 2: $L = S_1 = (x < y)\ ;\ G = S_2 = (x > y)$

A Tabela 10.11 apresenta a Tabela Verdade do módulo básico de comparação.
A Tabela 10.12 apresenta o Mapa de Karnaugh para o sinal G_k.
A Tabela 10.13 apresenta o Mapa de Karnaugh para o sinal L_k.
Dos mapas de Karnaugh, podem-se obter as seguintes formas mínimas:

$$G_k = G_{k+1} + \left(x_k \cdot \overline{y_k} \cdot \overline{L_{k+1}}\right)$$

e

$$L_k = L_{k+1} + \left(\overline{x_k} \cdot y_k \cdot \overline{G_{k+1}}\right)\ .$$

Opção 3: $E = S_1 = (x = y)\ ;\ G = S_2 = (x > y)$

A Tabela 10.14 apresenta a Tabela Verdade do módulo básico de comparação.
A Tabela 10.15 apresenta o Mapa de Karnaugh para o sinal G_k.
A Tabela 10.16 apresenta o Mapa de Karnaugh para o sinal E_k.
Dos mapas de Karnaugh, podem-se obter as seguintes formas mínimas:

$$G_k = G_{k+1} + (x_k \cdot \overline{y_k} \cdot E_{k+1})$$

e

$$E_k = (\overline{x_k} \cdot \overline{y_k} \cdot E_{k+1}) + (x_k \cdot y_k \cdot E_{k+1})\ .$$

G_{k+1}	L_{k+1}	x_k	y_k	G_k	L_k
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	X	X
1	1	0	1	X	X
1	1	1	0	X	X
1	1	1	1	X	X

Tabela 10.11: Tabela Verdade do módulo básico de comparação. Sinais maior (G) e menor (L).

		G_{k+1} L_{k+1}			
		00	01	11	10
x_k y_k	00	0	0	X	1
	01	0	0	X	1
	11	0	0	X	1
	10	1	0	X	1

Tabela 10.12: Mapa de Karnaugh para o sinal G_k do módulo básico de comparação.

		G_{k+1} L_{k+1}			
		00	01	11	10
x_k y_k	00	0	1	X	0
	01	1	1	X	0
	11	0	1	X	0
	10	0	1	X	0

Tabela 10.13: Mapa de Karnaugh para o sinal L_k do módulo básico de comparação.

G_{k+1}	E_{k+1}	x_k	y_k	G_k	E_k
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	0	0
0	1	1	0	1	0
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	X	X
1	1	0	1	X	X
1	1	1	0	X	X
1	1	1	1	X	X

Tabela 10.14: Tabela Verdade do módulo básico de comparação. Sinais igual (E) e maior (G).

		G_{k+1} E_{k+1}			
		00	01	11	10
x_k y_k	00	0	0	X	1
	01	0	0	X	1
	11	0	0	X	1
	10	0	1	X	1

Tabela 10.15: Mapa de Karnaugh para o sinal G_k do módulo básico de comparação.

		G_{k+1} E_{k+1}			
		00	01	11	10
x_k y_k	00	0	1	X	0
	01	0	0	X	0
	11	0	1	X	0
	10	0	0	X	0

Tabela 10.16: Mapa de Karnaugh para o sinal E_k do módulo básico de comparação.

10.22.3 Projeto usando a técnica de complemento

A comparação numérica aqui abordada é realizada com padrões binários interpretados como números não negativos, sem codificação adicional. Porém, a técnica de codificação por complemento, usada na representação de números negativos, pode ser empregada na implementação de um comparador. Isso é discutido a seguir.

Uso de complemento a 1

Na codificação por complemento a 1, o valor do módulo de codificação para um número binário com N dígitos vale $V_{mod_{C1}} = (2^N - 1)$, que é o maior valor representável V_{max}. Dados os números x e y, o seu complemento a 1 pode ser definido por $x_{C1} = (V_{mod_{C1}} - x)$ e por $y_{C1} = (V_{mod_{C1}} - y)$, respectivamente.

Realizando-se a operação $r_{yx_{C1}} = y + x_{C1} = y + (V_{mod_{C1}} - x) = V_{mod_{C1}} + (y - x) = V_{max} + (y - x)$, pode-se observar que, se $y \leq x$ então $r_{yx_{C1}} \leq V_{max}$, e que, se $y > x$ então $r_{yx_{C1}} > V_{max}$. Portanto, se o sinal de *carry out* do somador for igual a 0 ou a 1, tem-se que $y \leq x$ ou $y > x$, respectivamente.

Da mesma forma pode-se realizar a operação $r_{xy_{C1}} = x + y_{C1} = x + (V_{mod_{C1}} - y) = V_{mod_{C1}} + (x - y) = V_{max} + (x - y)$, verificando-se as condições $x \leq y$ ou $x > y$.

Finalmente, pode-se verificar se $x = y$, utilizando-se os dois resultados anteriores.

Uso de complemento a 2

Na codificação por complemento a 2, o valor do módulo de codificação para um número binário com N dígitos vale $V_{mod_{C2}} = 2^N = V_{max} + 1$. Dados os números x e y, o seu complemento a 2 pode ser definido por $x_{C2} = (V_{mod_{C2}} - x)$ e por $y_{C2} = (V_{mod_{C2}} - y)$, respectivamente.

Realizando-se a operação $r_{yx_{C2}} = y + x_{C2} = y + (V_{mod_{C2}} - x) = V_{mod_{C2}} + (y - x) = V_{max} + 1 + (y - x)$, pode-se observar que, se $y < x$ então $r_{yx_{C2}} \leq V_{max}$, e que, se $y \geq x$ então $r_{yx_{C2}} > V_{max}$. Portanto, se o sinal de *carry out* do somador for igual a 0 ou a 1, tem-se que $y < x$ ou $y \geq x$, respectivamente.

Da mesma forma pode-se realizar a operação $r_{xy_{C2}} = x + y_{C2} = x + (V_{mod_{C2}} - y) = V_{mod_{C2}} + (x - y) = V_{max} + 1 + (x - y)$, verificando-se as condições $x < y$ ou $x \geq y$.

Finalmente, pode-se verificar se $x = y$, utilizando-se os dois resultados anteriores.

Parte IV

Circuitos digitais seqüenciais

Capítulo 11

Circuitos seqüenciais: conceitos básicos

11.1 Introdução

- Circuitos combinacionais × circuitos seqüenciais.
- Circuitos combinacionais são sistemas instantâneos ou sem memória.
- Circuitos seqüenciais são sistemas dinâmicos ou com memória.
- Por serem sistemas instantâneos, os circuitos combinacionais respondem sempre da mesma forma, em qualquer momento, para os mesmos valores das variáveis de entrada.
- Por sua vez, por serem sistemas dinâmicos, dependendo da informação que se encontre armazenada, os circuitos seqüenciais podem responder de formas diferentes, em diferentes momentos, para os mesmos valores das variáveis de entrada.
- Circuitos seqüenciais também podem ser denominados de máquinas de estados ou de autômatos.

11.2 Estados e variáveis de estado

- Uma vez que eles são capazes de armazenar energia, os sistemas dinâmicos podem apresentar diversas configurações energéticas diferentes, denominadas **estados**.
- Uma medida do estado de um sistema, em um instante de tempo $t = t_n$, são os valores assumidos por todas as variáveis do sistema, em $t = t_n$.
- Interpretando-se o conjunto de todas as variáveis de um sistema como um espaço vetorial, pode-se selecionar um conjunto mínimo de variáveis para formar uma base para esse espaço. Uma vez que, a partir da base, podem ser obtidas todas as demais variáveis e, portanto, pode-se caracterizar o estado do sistema, as variáveis da base são denominadas **variáveis de estado** do sistema.
- Dessa forma, uma definição clássica para estado e variáveis de estado é: "O estado de um sistema, em qualquer instante de tempo $t = t_n$, é o menor conjunto de variáveis (denominadas variáveis de estado), calculadas em $t = t_n$, suficiente para determinar o comportamento do sistema para qualquer instante de tempo $t \geq t_n$, quando a entrada do sistema é conhecida para $t \geq t_n$".

11.3 Tipos de variáveis e sua interações

- Será considerado que todas as variáveis do circuito são booleanas.
- Assim sendo, os valores das variáveis podem ser interpretados como:
  - Nível: a informação é representada pelos níveis lógicos das variáveis boolenas (0 e 1). Cada nível representa um evento.
  - Borda: a informação é associada à seqüência de níveis 0 e 1 (borda positiva) ou à seqüência de níveis 1 e 0 (borda negativa). Cada borda representa um evento.
  - Transição: a informação é associada à troca de níveis 0 para 1 (transição positiva) ou à troca de níveis 1 para 0 (transição negativa). Cada transição representa um evento.
  - Pulso: a informação é associada à seqüência de níveis 0 e 1 e 0 (pulso positivo) ou à seqüência de níveis 1 e 0 e 1 (pulso negativo). A duração do valor intermediário da seqüência é denominada de largura do pulso (*pulsewidth*) e deve ser pequena em relação aos tempos envolvidos. Cada pulso representa um evento.
- Para alguns tipos de circuitos, as interações entre sinais dos tipos nível e pulso são de particular interesse. A Tabela 11.1 resume as possíveis interações, considerando-se as operações lógicas AND e OR. Os resultados indicam que, para tais operações, alguns tipos de interações produzem resultados indeterminados. Portanto, no projeto de sistemas com sinais pulsados, tais resultados devem ser levados em consideração.

A	B	$A \cdot B$	$A + B$
Nível	Nível	Nível	Nível
Nível	Pulso Positivo	Pulso Positivo	Indeterminado
Pulso Positivo	Pulso Positivo	Indeterminado	Pulso Positivo
Nível	Pulso Negativo	Indeterminado	Pulso Negativo
Pulso Negativo	Pulso Negativo	Pulso Negativo	Indeterminado
Pulso Positivo	Pulso Negativo	Indeterminado	Indeterminado

Tabela 11.1: Tipos de interações entre sinais dos tipos nível e pulso.

11.4 Modelo genérico para circuitos seqüenciais

Os circuitos seqüenciais podem ser classificados de diversas formas. A Figura 11.1 apresenta um modelo genérico para circuitos seqüenciais, onde:

- $x_i \in \boldsymbol{x}$, $i = 1, 2, \cdots, L$, são as variáveis de entrada ou variáveis de entrada principais.
- $z_i \in \boldsymbol{z}$, $i = 1, 2, \cdots, M$, são as variáveis de saída ou variáveis de saída principais.
- $Y_i \in \boldsymbol{Y}$, $i = 1, 2, \cdots, P$, são as variáveis de excitação ou variáveis de saída secundárias.
- $y_i \in \boldsymbol{y}$, $i = 1, 2, \cdots, R$, são as variáveis de estado ou variáveis de entrada secundárias.

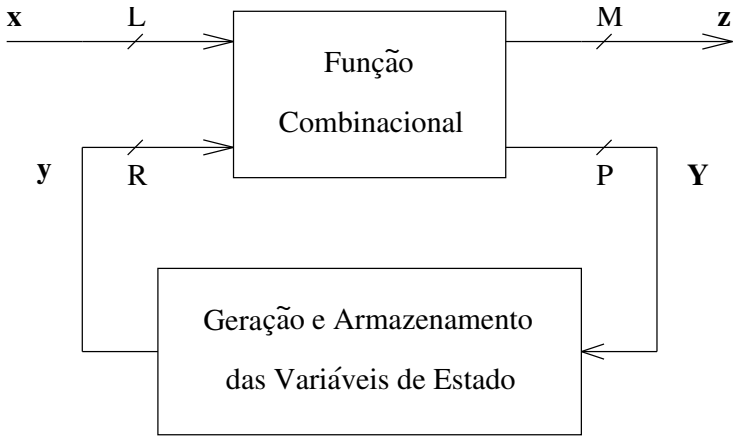


Figura 11.1: Modelo genérico para circuitos seqüenciais.

Considerando-se que a variável tempo faz parte da definição do fluxo de operação dos circuitos seqüenciais, pode-se dizer que:

- Tratando-se de um tempo discretizado (sinal amostrado), cada intervalo de tempo consecutivo Δt é considerado um único "instante de tempo" n.
- (t_n) ou (n) é o instante atual, (t_{n-1}) ou $(n-1)$ é o instante anterior e (t_{n+1}) ou $(n+1)$ é o próximo instante.
- v^n é o valor atual de v, $v^{(n-1)}$ é o valor anterior de v e $v^{(n+1)}$ é o próximo valor de v.
- O conjunto das variáveis y_i^n é denominado estado atual.
- Por sua vez, o conjunto das variáveis x_i^n e y_i^n é dito estado atual total.

Da Figura 11.1, podem-se definir as seguintes relações combinacionais:

$$z_i^n = f_i(x_1^n, \cdots, x_L^n, y_1^n, \cdots, y_R^n),\ i = 1, 2, \cdots, M \tag{11.1}$$

e

$$Y_j^n = f_j(x_1^n, \cdots, x_L^n, y_1^n, \cdots, y_R^n),\ j = 1, 2, \cdots, P\ , \tag{11.2}$$

bem como a seguinte relação seqüencial:

$$y_k^{n+1} = f_k(Y_1^n, \cdots, Y_P^n) = g_k(x_1^n, \cdots, x_L^n, y_1^n, \cdots, y_R^n),\ k = 1, 2, \cdots, R\ . \tag{11.3}$$

O bloco denominado **Função Combinacional** é um circuito combinacional que, de acordo com a viabilidade de custo, pode ser implementado através de portas lógicas individuais, memórias ROM (*Read-Only Memory*) ou circuitos PLA (*Programmable Logic Array*).

O bloco de memória denominado **Geração e Armazenamento das Variáveis de Estado** representa um dispositivo genérico de memória (*flip-flop*, banco de memória, atrasos de propagação).

A função do bloco de memória não é simplesmente armazenar Y_i^n na forma de y_i^{n+1}. Pelo contrário, a sua função é mais complexa. De uma forma geral, a partir dos P valores Y_j^n devem ser gerados os R valores y_k^{n+1}, os quais, então, serão retidos (*latched*) ou armazenados (*stored*).

As relações que definem o fluxo de operação dos circuitos seqüenciais, Equações (11.1) a (11.3), podem ser tabeladas, como ilustram as Tabelas 11.2 a 11.5. As Tabelas 11.4 e 11.5 representam apenas as formas compactas das Tabelas 11.2 e 11.3, respectivamente. As tabelas na forma expandida são mais utilizadas em ferramentas de *software*, enquanto as versões compactas são mais encontradas nos textos relativos ao assunto.

estado atual	entrada atual	excitação atual	saída atual
$(y_R \cdots y_1)^n$	$(x_L \cdots x_1)^n$	$(Y_P \cdots Y_1)^n$	$(z_M \cdots z_1)^n$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Tabela 11.2: Excitação e saída em função de estado e entrada: forma expandida.

estado atual	entrada atual	próximo estado	saída atual
$(y_R \cdots y_1)^n$	$(x_L \cdots x_1)^n$	$(y_R \cdots y_1)^{n+1}$	$(z_M \cdots z_1)^n$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Tabela 11.3: Estado e saída em função de estado e entrada: forma expandida.

estado atual	excitação atual			saída atual			
$(y_R \cdots y_1)^n$	$(Y_P \cdots Y_1)^n$			$(z_M \cdots z_1)^n$			entrada atual
	$0 \cdots 0$	$\cdots$	$1 \cdots 1$	$0 \cdots 0$	$\cdots$	$1 \cdots 1$	$\leftarrow (x_L \cdots x_1)^n$
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	

Tabela 11.4: Excitação e saída em função de estado e entrada: forma compacta.

estado atual	próximo estado			saída atual			
$(y_R \cdots y_1)^n$	$(y_R \cdots y_1)^{n+1}$			$(z_M \cdots z_1)^n$			entrada atual
	$0 \cdots 0$	$\cdots$	$1 \cdots 1$	$0 \cdots 0$	$\cdots$	$1 \cdots 1$	$\leftarrow (x_L \cdots x_1)^n$
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	

Tabela 11.5: Estado e saída em função de estado e entrada: forma compacta.

11.5 Classificação quanto à dependência do sinal de saída

- De uma forma geral, nos circuitos seqüenciais, as variáveis de saída z_i^n dependem diretamente das variáveis de estado y_k^n e das variáveis de entrada x_j^n.

- Porém, em alguns casos, as variáveis de saída z_i^n dependem diretamente das variáveis de estado y_k^n, mas indiretamente das variáveis de entrada x_j^n.

- Máquinas (circuitos) de Mealy e de Moore.

- Máquinas de Mealy: $z_i^n = f_i(x_1^n, \cdots, x_L^n, y_1^n, \cdots, y_R^n)$, $i = 1, 2, \cdots, M$.

- Máquinas de Moore: $z_i^n = f_i(y_1^n, \cdots, y_R^n)$, $i = 1, 2, \cdots, M$.

- As Figuras 11.2 e 11.3 apresentam, respectivamente, um exemplo de máquina de Mealy e um exemplo de máquina de Moore.

- Geralmente, as máquinas de Mealy são implementadas por circuitos mais simples do que as máquinas de Moore.

- Por outro lado, nas máquinas de Moore, em conseqüência de sua definição, os valores dos sinais de saída permanecem constantes entre dois estados consecutivos. Portanto, torna-se mais simples controlar a interação entre diversos blocos de circuitos desse tipo. Pela mesma razão, é mais fácil acompanhar a evolução dos estados do circuito, o que simplifica a depuração de erros.

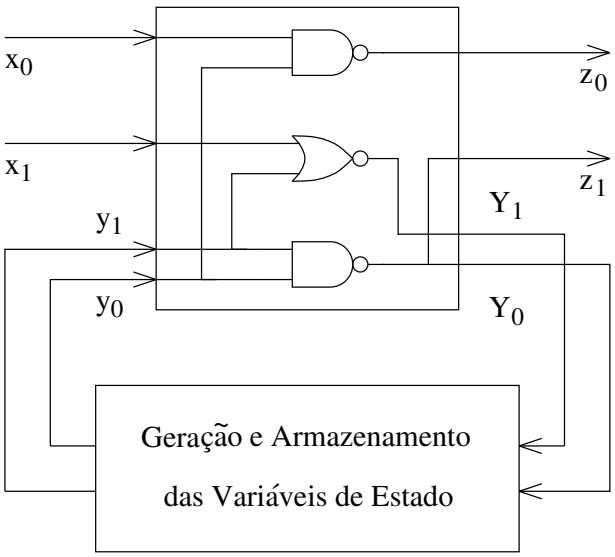


Figura 11.2: Exemplo de máquina de Mealy.

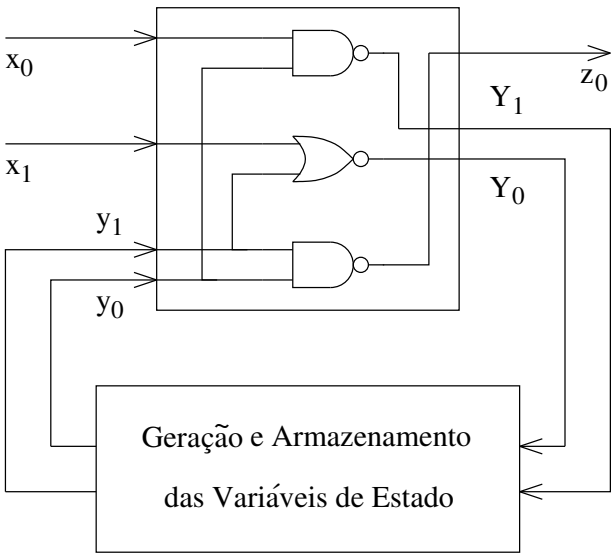


Figura 11.3: Exemplo de máquina de Moore.

11.6 Classificação quanto ao tipo de controle da mudança de estado

Na literatura, são encontradas várias denominações diferentes para designar os diversos tipos de circuitos seqüenciais existentes. A nomenclatura aqui utilizada será a seguinte: *Clock-mode* ou *clocked*, *Pulsed*, *Level-mode*. Tais classes são brevemente descritas a seguir.

11.6.1 Circuitos seqüenciais *clock-mode* ou *clocked*

- A Figura 11.4 ilustra um modelo genérico para circuitos seqüenciais *clock-mode*.
- Todas as variáveis carregam informação nos níveis.
- As variáveis de estado são modificadas apenas pela ação de um sinal pulsante, com função de temporização ou de controle, comumente denominado de relógio (*clock*).
- Apesar de ser um sinal pulsante, não é necessário que o *clock* seja periódico.
- O sinal de *clock* não carrega qualquer tipo de informação. Ele só determina quando haverá mudança de estado.
- As variáveis de excitação, em conjunto com os elementos de armazenamento, determinam qual será a mudança de estado.
- As variáveis de entrada devem estar estáveis quando da atuação do *clock*.
- Um *clock* atuando em t_n, com $\boldsymbol{x}^n$, $\boldsymbol{z}^n$, e $\boldsymbol{Y}^n$ estáveis, provoca uma mudança de estado de $\boldsymbol{y}^n$ para $\boldsymbol{y}^{n+1}$.
- O circuito deve estar estável entre dois pulsos de *clock*. Assim, o que limita a freqüência máxima de operação do circuito é, basicamente, a soma do tempo de estabilização da memória com o tempo de propagação máximo do circuito combinacional.

- De certa forma, um circuito seqüencial *clock-mode* pode ser interpretado como um caso particular de circuitos seqüenciais *pulsed*.

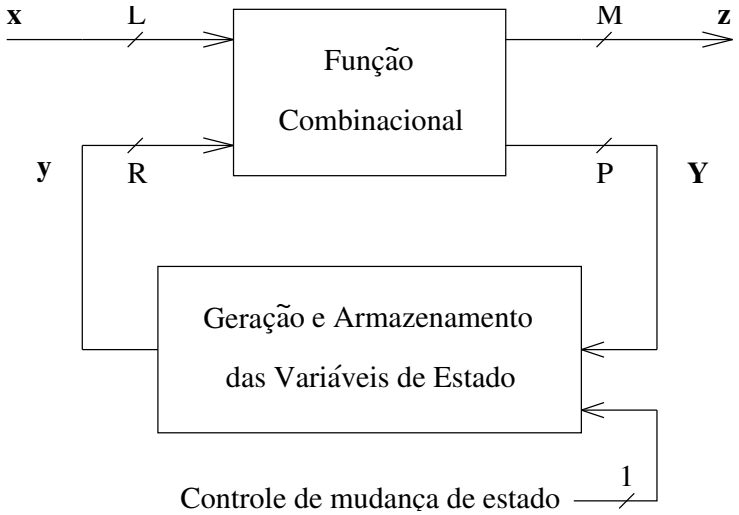


Figura 11.4: Modelo genérico para circuitos seqüenciais *clock-mode*.

11.6.2 Circuitos seqüenciais *pulsed*

- A Figura 11.5 apresenta um modelo genérico para circuitos seqüenciais *pulsed*.
- Não há um sinal pulsante de *clock* separado, sem informação.
- A mudança de estado ocorre pela atuação de um pulso em um sinal de entrada.

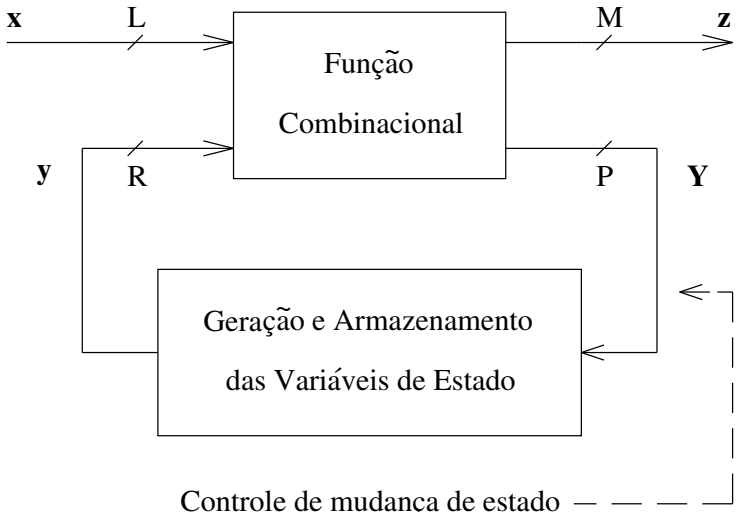


Figura 11.5: Modelo genérico para circuitos seqüenciais *pulsed*.

11.6.3 Circuitos seqüenciais *level-mode*

- Na Figura 11.6 pode ser visto um modelo genérico para circuitos seqüenciais *level-mode*.
- A realimentação das variáveis de excitação Y_i, gerando as variáveis de estado y_j, é realizada de forma contínua, ao contrário das demais classes, onde a mesma é controlada.
- A mudança de estado ocorre pela atuação de níveis dos sinais de entrada.
- Caso particular: operação em modo fundamental, onde uma mudança de nível só pode ocorrer após a mudança de nível anterior ter levado a máquina a um estado estável.
- Assim como nos demais classes: $y_k^{n+1} = f_k(Y_1^n, \cdots, Y_P^n)$, $k = 1, 2, \cdots, R$.
- Mais especificamente, neste caso: $P = R$ e $y_k(t + \Delta t_k) = Y_k(t)$, $k = 1, 2, \cdots, P$.
- Os atrasos Δt_k que implementam o bloco de memória não são blocos de retardo isolados. Eles representam a concentração de atrasos de propagação existentes no circuito combinacional.

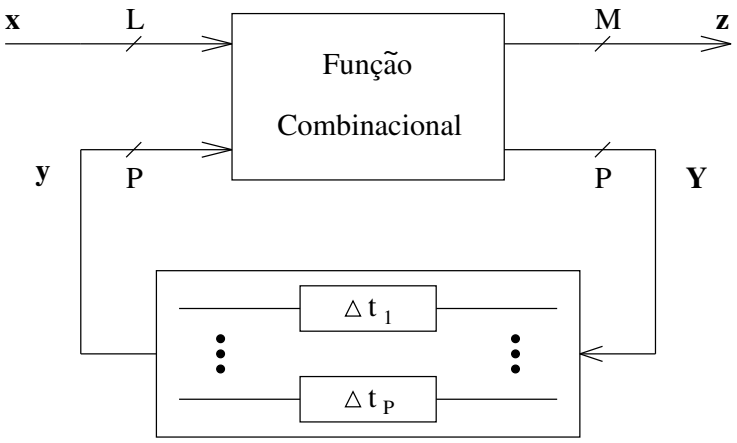


Figura 11.6: Modelo genérico para circuitos seqüenciais *level-mode*.

Capítulo 12

Elementos básicos de armazenamento

12.1 Introdução

- Se toda a informação presente em um circuito seqüencial for expressa por meio de valores binários, os elementos básicos de armazenamento deverão ser dispositivos capazes de armazenar variáveis booleanas.

- Assim, os requisitos básicos para tais dispositivos são:

  - Capacidade de representar os valores lógicos "0" e "1".
  - Possibilidade de representar apenas os valores lógicos "0" e "1".
  - Capacidade de travar (*latch*) os valores lógicos "0" e "1" por tempo indeterminado.
  - Capacidade de decidir sobre o valor lógico a ser armazenado, a partir de sinais de acionamento.

- Os requisitos acima definem um dispositivo com dois estados, estáveis, cuja mudança de estados é disparada (*triggered*) por sinais de ativação específicos.

- Tecnicamente, tal dispositivo é denominado de multivibrador biestável.

- Popularmente, embora não haja um consenso sobre a classificação dos dispositivos, são empregadas as denominações *latch* e *flip-flop*.

- Um dispositivo multivibrador biestável pode ser implementado através de circuitos analógicos, utilizando-se transistores, resistores e capacitores.

- Por outro lado, é possível obter uma implementação dita digital, utilizando-se apenas portas lógicas como elementos primitivos.

- Do ponto de vista de integração do sistema (lógica combinacional + lógica seqüencial), a implementação digital pode ser interpretada como a mais adequada para o projeto de sistemas digitais, uma vez que utiliza portas lógicas como elementos primitivos.

- Deve ser ressaltado que, pela sua própria caracterização, os elementos básicos de armazenamento, implementados de forma digital, são circuitos seqüenciais elementares, do tipo *level-mode*.

12.2 Classificação quanto à funcionalidade

- No tocante à funcionalidade, existem quatro tipos básicos de *flip-flops*: SR, JK, D e T.

- Dependendo do tipo de implementação do dispositivo e dos sinais de ativação existentes, diversas variações desses quatro tipos básicos podem ser definidas e implementadas.

- Independentemente das possíveis variações, a funcionalidade básica de cada um dos quatro tipos citados pode ser representada pelas seguintes equações, onde X^n representa o valor da variável X no instante t_n e X^{n+1} representa o valor da variável X no instante seguinte t_{n+1}:

  - *Flip-flop* SR:

$$\begin{cases} Q^{n+1} = (S^n) + \left(\overline{R^n} \cdot Q^n\right) & , \text{ para } (S^n \cdot R^n) = 0 \\ Indeterminado & , \text{ para } S^n = R^n = 1 \end{cases} . \tag{12.1}$$

  - *Flip-flop* JK:

$$Q^{n+1} = \left(J^n \cdot \overline{Q^n}\right) + \left(\overline{K^n} \cdot Q^n\right) . \tag{12.2}$$

  - *Flip-flop* D:

$$Q^{n+1} = D^n . \tag{12.3}$$

  - *Flip-flop* T_1:

$$Q^{n+1} = \overline{Q^n} \tag{12.4}$$

  - *Flip-flop* T_2:

$$Q^{n+1} = \left(T^n \cdot \overline{Q^n}\right) + \left(\overline{T^n} \cdot Q^n\right) . \tag{12.5}$$

- As operações básicas, associadas às Equações (12.1), (12.2), (12.3) e (12.5), podem ser mais facilmente identificadas através de suas respectivas tabelas, apresentadas na Figura 12.1.

- As variáveis S, R, J, K, D e T representam os sinais de entrada, enquanto a variável Q representa o sinal de saída dos respectivos *flip-flops*.

- Das equações apresentadas, e de suas respectivas tabelas, torna-se natural o significado da nomenclatura dos sinais: Q (*Quiescent*), SR (*Set-Reset*), D (*unit Delay*) e T (*Toggle*).

- A nomenclatura JK surgiu historicamente, sem qualquer relação com a sua funcionalidade.

S^n	R^n	Q^{n+1}
0	0	Q^n
0	1	0
1	0	1
1	1	proibido

J^n	K^n	Q^{n+1}
0	0	Q^n
0	1	0
1	0	1
1	1	$\overline{Q^n}$

D^n	Q^{n+1}
0	0
1	1

T^n	Q^{n+1}
0	Q^n
1	$\overline{Q^n}$

Figura 12.1: Tabelas de operação básica para os *flip-flops* SR, JK, D e T_2.

12.3 Relacionamento entre os tipos básicos de *flip-flops*

- Observando-se as equações dos tipos básicos de *flip-flops*, e suas respectivas tabelas, pode-se notar um estreito relacionamento entre eles.
- Alguns desses relacionamentos podem ser estabelecidos sem o emprego de realimentação, o que acontece nos casos de um *flip-flop* com mais funcionalidade para um *flip-flop* com menos funcionalidade.
- Os casos contrários requerem que o *flip-flop* seja realimentado.
- Inicialmente, pode-se estabelecer as seguintes relações entre os *flip-flops* SR, JK, D e T:
  - Para as combinações de entrada “00”, “01” e “10”, os *flip-flops* SR e JK possuem o mesmo comportamento.
  - O *flip-flop* JK amplia a operação do *flip-flop* SR, implementando uma funcionalidade para a combinação de entrada “11”.
  - O *flip-flop* JK, com as entradas $J = K = 1$ ou $J = K = T$, é equivalente ao *flip-flop* T, de acordo com as Equações (12.4) e (12.5), respectivamente.
  - Por sua vez, um *flip-flop* D pode ser implementado a partir de *flip-flops* SR ou JK, se $S = D$ e $R = \overline{S}$ ou se $J = D$ e $K = \overline{J}$, respectivamente.
  - Um *flip-flop* T_1 pode ser implementado a partir de um *flip-flop* T_2, fazendo-se $T = 1$.
  - A partir de um *flip-flop* D pode-se implementar um *flip-flop* T, adotando-se $D = \overline{Q}$ ou $D = \left(T \cdot \overline{Q}\right) + \left(\overline{T} \cdot Q\right)$, conforme as Equações (12.4) e (12.5), respectivamente.
- A Tabela 12.1 apresenta um resumo de transformações envolvendo *flip-flops* dos tipos SR, JK, D, e T, utilizando suas entradas e saídas como variáveis de projeto. As opções marcadas com (*) indicam a impossibilidade desse tipo de projeto, uma vez que o *flip-flop* do tipo T_1 não possui entrada de dados. Existe apenas um sinal de sincronismo ($CTRL$ ou CK) que controla a sua operação. Sendo assim, uma solução diferente deve ser proposta, a qual atue sobre tal sinal de controle.

Transformação desejada	Tipo de arquitetura	
	Sem realimentação	Com realimentação
$JK \longrightarrow SR$	Não aplicar: $J = K = 1$	—
$JK \longrightarrow D$	$J = D$; $K = \overline{J}$	—
$JK \longrightarrow T_1$	$J = K = 1$	—
$JK \longrightarrow T_2$	$J = K = T$	—
$SR \longrightarrow JK$	—	$S = (J \cdot \overline{Q})$; $R = (K \cdot Q)$
$D \longrightarrow JK$	—	$D = (J \cdot \overline{Q}) + (\overline{K} \cdot Q)$
$T_1 \longrightarrow JK$	(*)	(*)
$T_2 \longrightarrow JK$	—	$T = (J \cdot \overline{Q}) + (K \cdot Q)$
$SR \longrightarrow D$	$S = D$; $R = \overline{S}$	—
$SR \longrightarrow T_1$	—	$S = \overline{Q}$; $R = Q$
$SR \longrightarrow T_2$	—	$S = (T \cdot \overline{Q})$; $R = (T \cdot Q)$
$D \longrightarrow SR$	—	$D = (S) + (\overline{R} \cdot Q)$
$T_1 \longrightarrow SR$	(*)	(*)
$T_2 \longrightarrow SR$	—	$T = (S \cdot \overline{Q}) + (R \cdot Q)$
$D \longrightarrow T_1$	—	$D = \overline{Q}$
$D \longrightarrow T_2$	—	$D = (T \cdot \overline{Q}) + (\overline{T} \cdot Q)$
$T_1 \longrightarrow D$	(*)	(*)
$T_2 \longrightarrow D$	—	$T = (D \cdot \overline{Q}) + (\overline{D} \cdot Q)$
$T_1 \longrightarrow T_2$	(*)	(*)
$T_2 \longrightarrow T_1$	$T = 1$	—

Tabela 12.1: Transformações envolvendo *flip-flops* dos tipos JK, D, T_1 e T_2.

12.4 Mapas de excitação dos *flip-flops*

- Uma outra forma de descrever a operação de um *flip-flop* é através do tipo de excitação que deve ser aplicado nas suas entradas a fim de provocar uma determinada variação na sua saída. Tal forma de descrição é denominada mapa de excitação.

- A Figura 12.2 apresenta os mapas de excitação para os *flip-flops* SR, JK, D e T_2.

Q^n	$\rightarrow$	Q^{n+1}	S^n	R^n
0	$\rightarrow$	0	0	X
0	$\rightarrow$	1	1	0
1	$\rightarrow$	0	0	1
1	$\rightarrow$	1	X	0
0	$\rightarrow$	X	X	X
1	$\rightarrow$	X	X	X

Q^n	$\rightarrow$	Q^{n+1}	J^n	K^n
0	$\rightarrow$	0	0	X
0	$\rightarrow$	1	1	X
1	$\rightarrow$	0	X	1
1	$\rightarrow$	1	X	0
0	$\rightarrow$	X	X	X
1	$\rightarrow$	X	X	X

Q^n	$\rightarrow$	Q^{n+1}	D^n
0	$\rightarrow$	0	0
0	$\rightarrow$	1	1
1	$\rightarrow$	0	0
1	$\rightarrow$	1	1
0	$\rightarrow$	X	X
1	$\rightarrow$	X	X

Q^n	$\rightarrow$	Q^{n+1}	T^n
0	$\rightarrow$	0	0
0	$\rightarrow$	1	1
1	$\rightarrow$	0	1
1	$\rightarrow$	1	0
0	$\rightarrow$	X	X
1	$\rightarrow$	X	X

Figura 12.2: Mapas de excitação para os *flip-flops* SR, JK, D e T_2.

12.5 Tipos de comportamento das saídas dos *flip-flops*

- Os tipos de comportamento que a saída de um *flip-flop* pode apresentar, de um instante de tempo (t_n) para o instante de tempo seguinte (t_{n+1}), são definidos na Tabela 12.2.

Q^n	$\rightarrow$	Q^{n+1}	Símbolo	Tipo de Comportamento
0	$\rightarrow$	0	0	Estático
0	$\rightarrow$	1	α	Dinâmico
1	$\rightarrow$	0	β	Dinâmico
1	$\rightarrow$	1	1	Estático
0	$\rightarrow$	X	X	Indeterminado
1	$\rightarrow$	X	X	Indeterminado

Tabela 12.2: Definição dos tipos de comportamento apresentados pela saída de um *flip-flop*.

12.6 Excitação × comportamento

- As tabelas da Figura 12.3 associam os tipos de comportamento da saída às respectivas excitações que as entradas devem sofrer, para os *flip-flops* SR, JK, D e T_2.

Q^n	→	Q^{n+1}	Variação	S^n	R^n
0	→	0	0	0	X
0	→	1	α	1	0
1	→	0	β	0	1
1	→	1	1	X	0
0	→	X	X	X	X
1	→	X	X	X	X

Q^n	→	Q^{n+1}	Variação	J^n	K^n
0	→	0	0	0	X
0	→	1	α	1	X
1	→	0	β	X	1
1	→	1	1	X	0
0	→	X	X	X	X
1	→	X	X	X	X

Q^n	→	Q^{n+1}	Variação	D^n
0	→	0	0	0
0	→	1	α	1
1	→	0	β	0
1	→	1	1	1
0	→	X	X	X
1	→	X	X	X

Q^n	→	Q^{n+1}	Variação	T^n
0	→	0	0	0
0	→	1	α	1
1	→	0	β	1
1	→	1	1	0
0	→	X	X	X
1	→	X	X	X

Figura 12.3: Tipos de comportamento e respectivas excitações para os *flip-flops* SR, JK, D e T_2.

12.7 Funcionalidade × excitação × comportamento

- A Tabela 12.3 apresenta um resumo geral de funcionalidade-excitação-comportamento, relacionando os valores de excitação a serem aplicados nas entradas, a partir de cada tipo de comportamento da saída, para cada tipo de *flip-flop*.

Entrada	Entrada = "1"	Entrada = "0"	Entrada = "X"
S	α	0 , β	1 , X
R	β	1 , α	0 , X
J	α	0	1 , β , X
K	β	1	0 , α , X
D	1 , α	0 , β	X
T	α , β	0 , 1	X

Tabela 12.3: Tabela resumo de funcionalidade-excitação-comportamento para os *flip-flops* SR, JK, D e T_2.

12.8 Circuitos seqüenciais × tabelas dos *flip-flops*

- Uma vez que os *flip-flops* podem usados como elementos básicos de armazenamento nos circuitos seqüenciais, as tabelas que os definem apresentam-se como ferramentas de análise e síntese para tais circuitos.
- As aplicações e os termos citados a seguir serão definidos nos próximos capítulos.
- No processo de análise de um circuito seqüencial, as tabelas de operação dos *flip-flops* são utilizadas para montar a tabela de mudança de estados.
- No processo de síntese, as tabelas de excitação e de comportamento são necessárias para montar os mapas-K de excitação e de transição, respectivamente.
- Os mapas-K de excitação apresentam os valores que as variáveis de excitação do circuito seqüencial, que são as variáveis de entrada dos elementos de memória, devem assumir, em função das suas variáveis de estado e das variáveis de entrada. É utilizado um mapa-K específico para cada entrada de cada *flip-flop*.
- Os mapas-K de transição descrevem o comportamento dos elementos de memória do circuito seqüencial, em função das suas variáveis de estado e das variáveis de entrada. É necessário apenas um único mapa-K para todos os tipos de *flip-flops*, para cada elemento de memória.
- Portanto, as funções lógicas que geram as variáveis de excitação, que são as variáveis de entrada dos elementos de memória, podem ser obtidas: i) do mapa-K de excitação de cada entrada, de cada *flip-flop* ou ii) do mapa-K de transição de cada elemento de memória, em conjunto com a tabela resumo 12.3.
- Como exemplo, a Tabela 12.4 descreve as mudanças de estado e os tipos de comportamento dos elementos de memória para um contador binário, crescente, de três *bits*. Por sua vez, os mapas-K de transição dos elementos de memória e os mapas-K de escitação para *flip-flops* JK são apresentados na Figuras 12.4 e 12.5, respectivamente. Deve-se notar que este contador não possui variáveis de entrada. Das tabelas das Figuras 12.4 e 12.5, pode-se obter

$$J_2 = K_2 = (Q_1 \cdot Q_0) \ , \tag{12.6}$$

$$J_1 = K_1 = Q_0 \tag{12.7}$$

e

$$J_0 = K_0 = 1 \ . \tag{12.8}$$

Q_2^n	Q_1^n	Q_0^n	Q_2^{n+1}	Q_1^{n+1}	Q_0^{n+1}	Q_2	Q_1	Q_0
0	0	0	0	0	1	0	0	α
0	0	1	0	1	0	0	α	β
0	1	0	0	1	1	0	1	α
0	1	1	1	0	0	α	β	β
1	0	0	1	0	1	1	0	α
1	0	1	1	1	0	1	α	β
1	1	0	1	1	1	1	1	α
1	1	1	0	0	0	β	β	β

Tabela 12.4: Tabela de mudanças de estado e de comportamento dos elementos de memória para um contador binário, crescente, de três *bits*.

$FF2$

Q_2 \ Q_1Q_0	00	01	11	10
0	0	0	α	0
1	1	1	β	1

$FF1$

Q_2 \ Q_1Q_0	00	01	11	10
0	0	α	β	1
1	0	α	β	1

$FF0$

Q_2 \ Q_1Q_0	00	01	11	10
0	α	β	β	α
1	α	β	β	α

Figura 12.4: Mapas-K de transição para os elementos de memória de um contador binário, crescente, de três *bits*.

J_2

Q_2 \ Q_1Q_0	00	01	11	10
0	0	0	1	0
1	X	X	X	X

K_2

Q_2 \ Q_1Q_0	00	01	11	10
0	X	X	X	X
1	0	0	1	0

J_1

Q_2 \ Q_1Q_0	00	01	11	10
0	0	1	X	X
1	0	1	X	X

K_1

Q_2 \ Q_1Q_0	00	01	11	10
0	X	X	1	0
1	X	X	1	0

J_0

Q_2 \ Q_1Q_0	00	01	11	10
0	1	X	X	1
1	1	X	X	1

K_0

Q_2 \ Q_1Q_0	00	01	11	10
0	X	1	1	X
1	X	1	1	X

Figura 12.5: Mapas-K de excitação para os *flip-flops* JK de um contador binário, crescente, de três *bits*.

12.9 Estruturas estáticas simétricas

- Os elementos básicos de armazenamento (*flip-flops*) podem ser implementados de diversas formas diferentes.

- Duas características são de grande interesse para o projeto de circuitos seqüenciais: i) que os *flip-flops* possuam saídas complementares e ii) que a temporização das mudanças dos valores de tais saídas possua o maior sincronismo possível.

- Tais características podem ser obtidas através de estruturas simétricas.

- A Figura 12.6 apresenta uma estrutura simétrica de armazenamento, implementada por dois inversores autorealimentados.

- A autorealimentação confere uma característica de armazenamento estático à estrutura, pois suas saídas Q e $\overline{Q}$ estarão estáveis (quiescentes) enquanto os inversores estiverem energizados.

- A estrutura da Figura 12.6 apresenta uma grande desvantagem: não é controlável.

- Algumas propostas para tornar o circuito da Figura 12.6 controlável são ilustradas na Figura 12.7.

  – No primeiro caso, Figura 12.7.a, utiliza-se um inversor com capacidade de corrente alta (inversor forte), um inversor com capacidade de corrente baixa (inversor fraco) e uma única chave responsável pela escrita do dado binário.

  – No segundo caso, Figura 12.7.b, são utilizados dois inversores idênticos, enquanto uma chave de duas posições controla a escrita e a manutenção do dado binário.

  – No terceiro caso, Figura 12.7.c, são utilizados dois inversores idênticos e a chave de duas posições é implementada através de duas chaves com controles independentes para escrita e armazenamento.

  – No último caso, Figura 12.7.d, são utilizados dois inversores idênticos e a chave de duas posições é implementada através de duas chaves com acionamentos complementares para escrita e armazenamento.

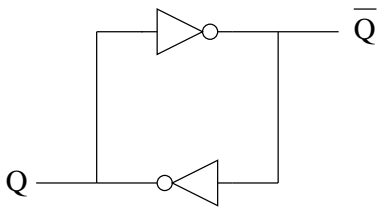


Figura 12.6: Estrutura de armazenamento estática e simétrica, não controlável.

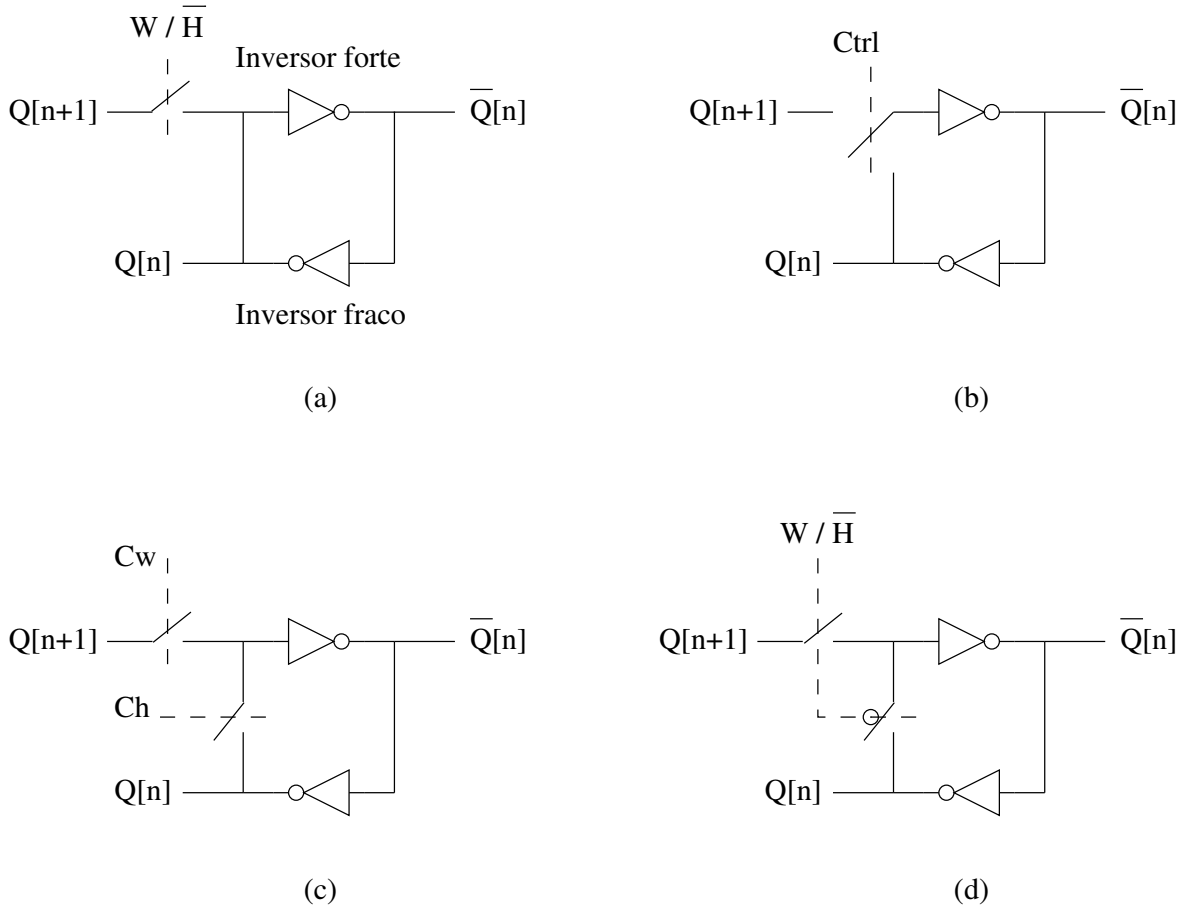


Figura 12.7: Estruturas de armazenamento estáticas e simétricas, controláveis por chaves.

12.10 Exemplos de *flip-flops*

- Uma vez que *flip-flops* são circuitos seqüenciais do tipo *level-mode*, os mesmos devem ser projetados adequadamente, por meio das técnicas existentes para tais tipos de sistemas.
- Porém, ainda que não se conheça a forma como foram projetados, não é difícil analisar o funcionamento de um determinado *flip-flop*.
- A seguir são apresentadas algumas implementações de *flip-flops*.
- Embora não haja um consenso na classificação dos *flip-flops*, os mesmos serão divididos em: *unclocked* (sem sinal de controle de sincronismo) e *clocked* (com sinal de controle de sincronismo).

12.10.1 *Flip-flops* do tipo *unclocked*

- Os *flip-flops* do tipo *unclocked* são também denominados de *latches*.
- O circuito de armazenamento estático da Figura 12.6 pode ser controlado usando apenas portas lógicas. O primeiro passo nesse sentido é substituir os inversores por portas lógicas NOR ou NAND. Em seguida, um terminal de entrada de cada porta deve ser desconectado, a fim de ser utilizado como terminal de controle (S e R). O processo é ilustrado nas Figuras 12.8 e 12.9.

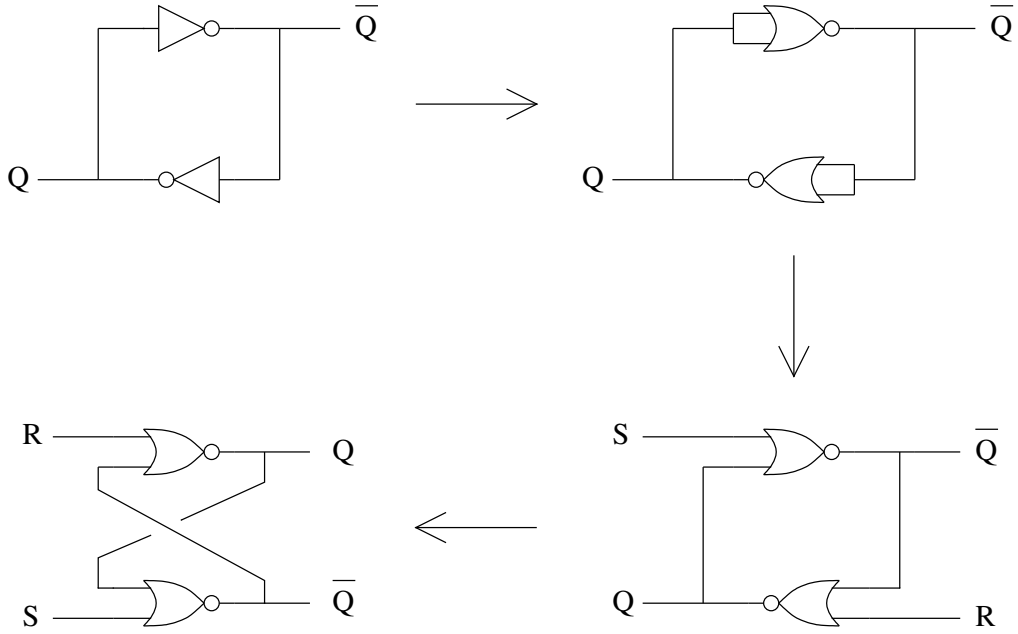


Figura 12.8: Uso de portas lógicas NOR na implementação de controle em uma estrutura de armazenamento estática e simétrica.

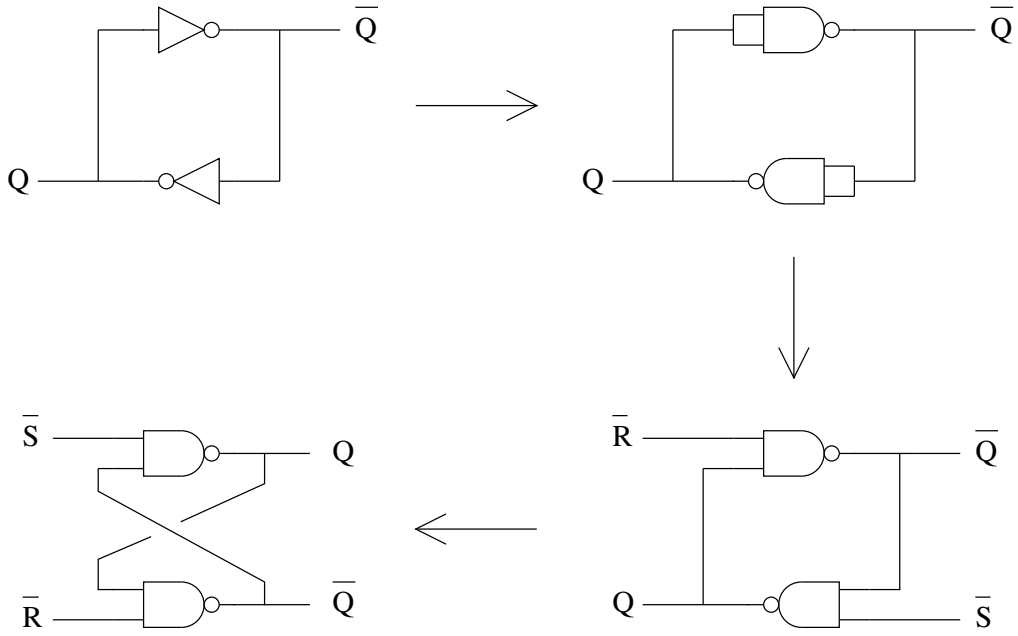


Figura 12.9: Uso de portas lógicas NAND na implementação de controle em uma estrutura de armazenamento estática e simétrica.

- Deve ser notado que, enquanto $S = R = 0$, os valores de Q e $\overline{Q}$ são mantidos estáveis.
- Alterando-se os valores dos sinais de controle para $S = 1$ e $R = 0$, obtém-se:
  - $\overline{Q}_{NOR} = 0$ e, em seguida, $Q_{NOR} = 1$.
  - $Q_{NAND} = 1$ e, em seguida, $\overline{Q}_{NAND} = 0$.
- Retornando-se à condição $S = R = 0$, os valores de Q e $\overline{Q}$ são mantidos estáveis.
- Alterando-se os valores dos sinais de controle para $S = 0$ e $R = 1$, obtém-se:
  - $Q_{NOR} = 0$ e, em seguida, $\overline{Q}_{NOR} = 1$.
  - $\overline{Q}_{NAND} = 1$ e, em seguida, $Q_{NAND} = 0$.
- Se forem atribuídos os valores $S = R = 1$, o resultado é indeterminado e não complementar. No caso da implementação com NOR, $Q_{NOR} = \overline{Q}_{NOR} = 0$. No caso da implementação com NAND, $Q_{NAND} = \overline{Q}_{NAND} = 1$. Por essa razão, tal configuração é dita proibida.
- A Tabela 12.5 resume a análise acima, de onde pode-se observar que ambos os circuitos implementam um *flip-flop* do tipo *unclocked* SR.
- Quanto aos demais tipos de *flip-flop*:
  - Acrescentando-se uma porta lógica inversora aos circuitos, de forma que $R = \overline{S}$, eles podem implementar um *flip-flop* do tipo *unclocked* D. Porém, tal construção não tem utilidade prática, uma vez que o circuito final passa a se comportar como um mero propagador do sinal de entrada, sem controle de retenção.
  - Devido a problemas de instabilidade, não é possível implementar *flip-flops* dos tipos *unclocked* JK e *unclocked* T.
- Finalmente, cabe observar que, embora o *flip-flop* do tipo *unclocked* SR possua várias limitações, o mesmo é usado como núcleo básico para a implementação dos *flip-flops* do tipo *clocked*, conforme será ilustrado a seguir.

S^n	R^n	Q^{n+1}
0	0	Q^n
0	1	0
1	0	1
1	1	proibido

Tabela 12.5: Operação das estruturas de armazenamento estáticas e simétricas controladas por meio de portas lógicas NOR e NAND.

12.10.2 *Flip-flops* do tipo *clocked*

- Dependendo da arquitetura utilizada, podem ser destacadas três classes de *flip-flops* do tipo *clocked*: elementar, *master-slave* e *edge-triggered*.

Flip-flops do tipo *clocked* elementar

- Em relação aos *flip-flops* do tipo *clocked* elementar, pode-se dizer que um SR é um *latch* com controle de sincronismo, conforme exemplificado nas Figuras 12.10 e 12.11. Por sua vez, um *flip-flop* D pode ser implementado a partir de um SR, conforme ilustrado na Figura 12.12.

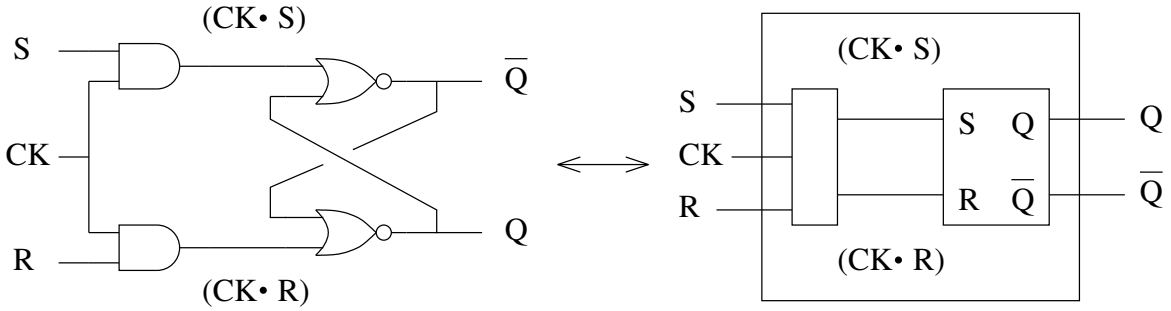


Figura 12.10: Exemplo de implementação de *flip-flop* SR do tipo *clocked* elementar, usando portas lógicas NOR.

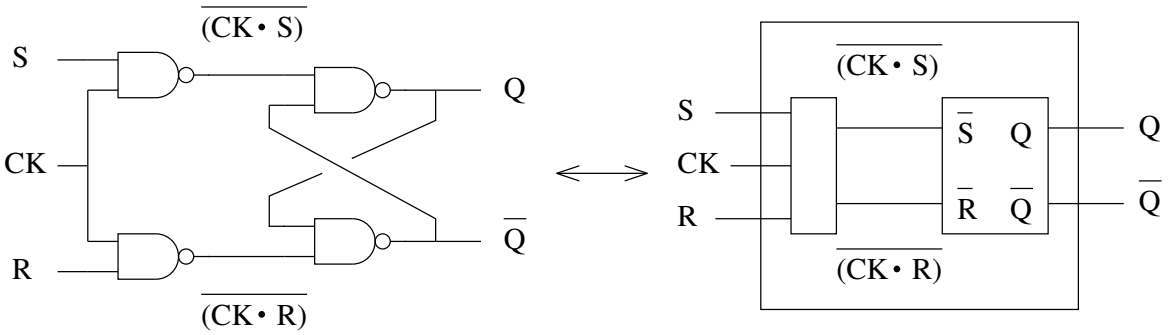


Figura 12.11: Exemplo de implementação de *flip-flop* SR do tipo *clocked* elementar, usando portas lógicas NAND.

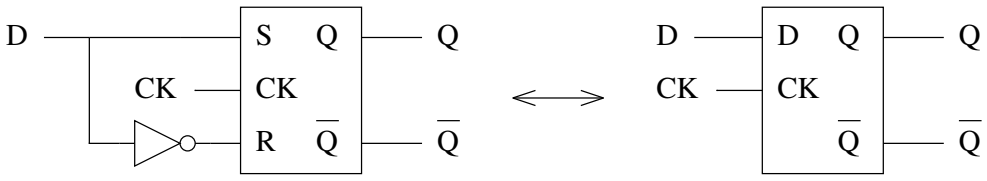


Figura 12.12: Exemplo de implementação de *flip-flop* D do tipo *clocked* elementar, com base em um *flip-flop* SR.

Flip-flops* dos tipos *clocked master-slave* e *clocked edge-triggered

- O tipo elementar pode ser usado como bloco básico de construção para outras estruturas funcionais. Os tipos *master-slave* e *edge-triggered* são soluções propostas para problemas que podem surgir em tais implementações.

- O tipo *master-slave* emprega o conceito de *pipelining*. A idéia por trás dessa técnica é que, a cada unidade funcional de uma cadeia de processamento, sejam adicionados elementos de memória de entrada (*master*) e de saída (*slave*), com sinais de controle de carregamento alternados. Dessa forma, todas as unidades da cadeia trabalham em paralelo, aumentando o fluxo de processamento (*throughput*). A técnica é ilustrada na Figura 12.13. No caso do *flip-flop master-slave*, a unidade funcional é apenas uma transmissão, conectando os elementos de memória de entrada e de saída.

- Embora uma estrutura *master-slave* empregue o dobro do circuito necessário ao armazenamento, ela permite um maior controle de fluxo entre a entrada e a saída do *flip-flop*. Uma vez que os sinais de entrada só provocam modificações na saída após uma alternância de sinais de controle, tais *flip-flops* podem ser interpretados como sensíveis a bordas (de subida ou de descida) ou a pulsos (positivo ou negativo).

- O tipo *edge-triggered* é uma solução proposta para um problema de operação apresentado pelo tipo *master-slave*. Nessa estrutura, além da célula básica de armazenamento, circuitos realimentados garantem que, logo após ocorra uma transição do sinal de controle, o *flip-flop* fique insensível a qualquer variação dos sinais de entrada, até que ocorra uma outra transição do mesmo tipo. Assim, desprezando-se o tempo necessário à insensibilização da estrutura, pode-se dizer que a mesma é sensível a transições (positiva ou negativa).

- Um exemplo de implementação para um *flip-flop* D do tipo *clocked*, com estrutura *master-slave*, pode ser encontrado na Figura 12.14, onde é empregado um *flip-flop* SR como célula básica.

- Não é difícil mostrar que um *flip-flop* SR pode ser usado para implementar um *flip-flop* JK, desde que $S = (\overline{Q} \cdot J)$ e $R = (Q \cdot K)$. Implementações utilizando *flip-flops* SR *unclocked* e *clocked* são mostradas nas Figuras 12.15 e 12.16, respectivamente. Uma vez que a realimentação das saídas (Q e $\overline{Q}$) para as entradas (J e K) é realizada de forma contínua, ambas apresentam o mesmo problema: oscilam quando $J = K = 1$. Para solucionar esse problema, exemplos de implementação para um *flip-flop* JK do tipo *clocked*, com estrutura *master-slave*, são apresentados nas Figuras 12.17 – 12.19.

- Devido a problemas de temporização, o *flip-flop* D da Figura 12.14 pode apresentar mau funcionamento e até mesmo oscilações. Uma implementação mais robusta é alcançada utilizando-se o *flip-flop* JK *master-slave*, com $D = J$ e $K = \overline{J}$.

- Por sua vez, um *flip-flop* T pode ser implementado com $J = K = 1$ ou $J = K = T$.

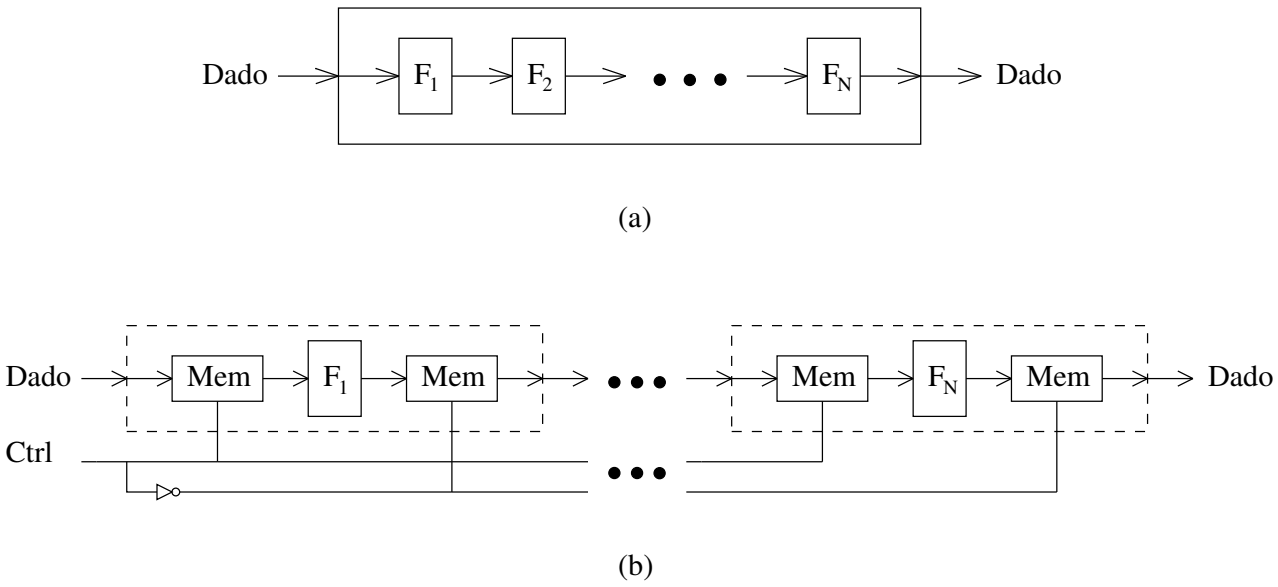


Figura 12.13: Técnica de *pipelining*: (a) Bloco funcional original e (b) Bloco com *pipelining*.

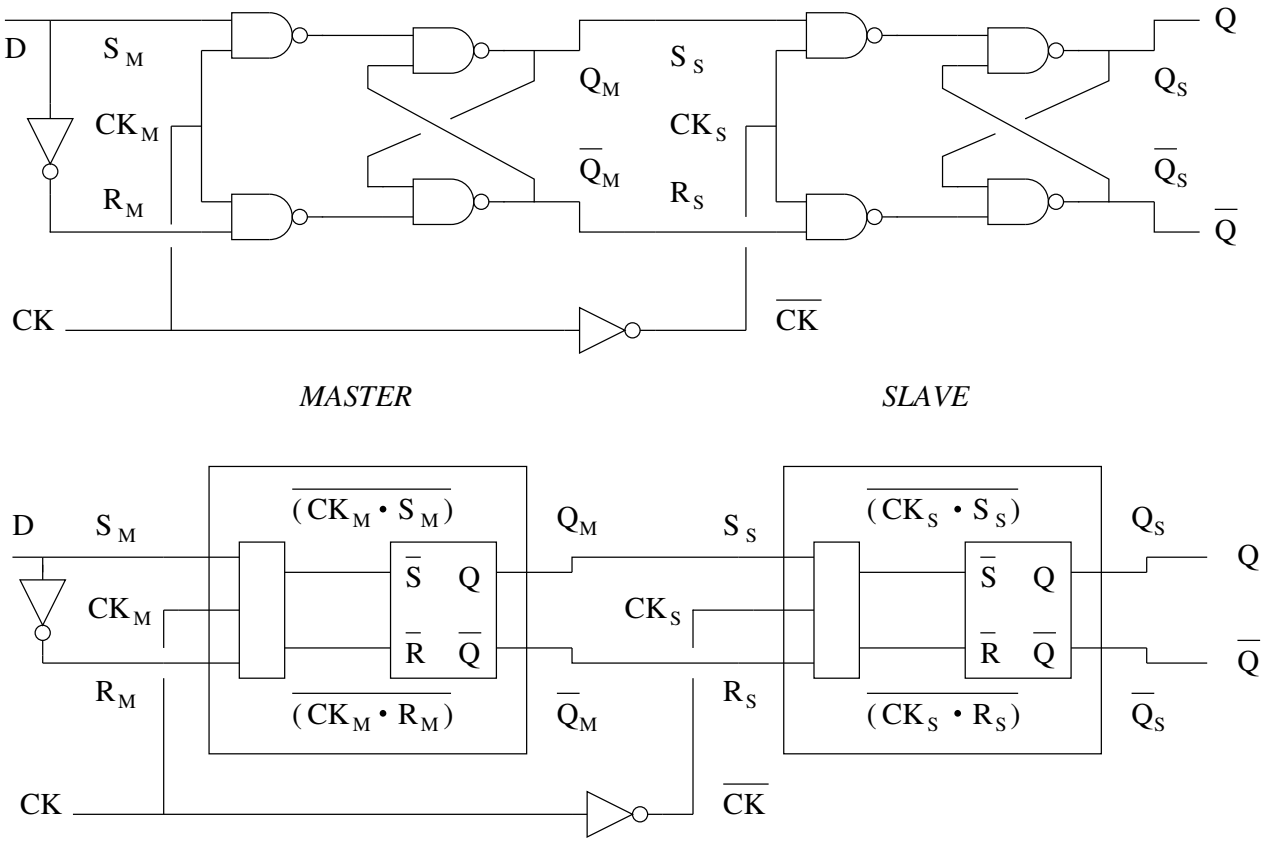


Figura 12.14: Exemplo de implementação de *flip-flop* D do tipo *master-slave*, com base em *flip-flops* SR.

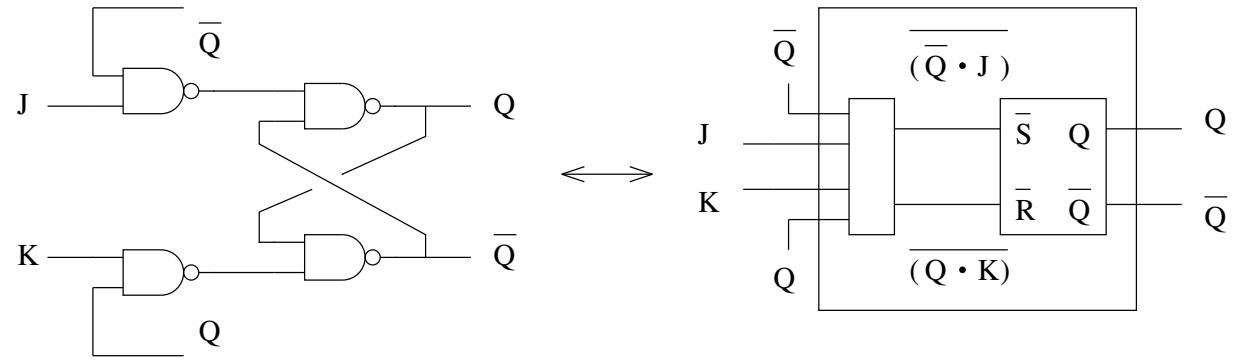


Figura 12.15: Exemplo de implementação de *flip-flop* JK, a partir de *flip-flop* SR *unclocked*, com problema de oscilação.

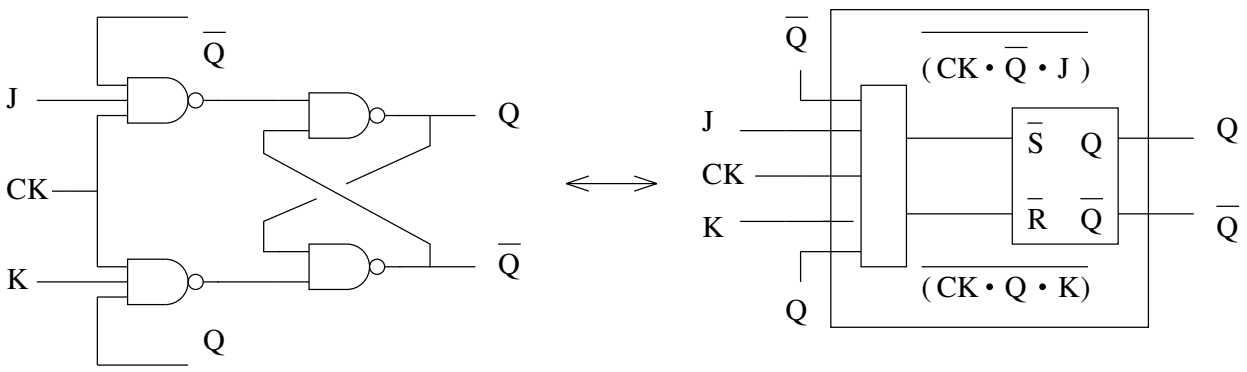


Figura 12.16: Exemplo de implementação de *flip-flop* JK, a partir de *flip-flop* SR *clocked*, com problema de oscilação.

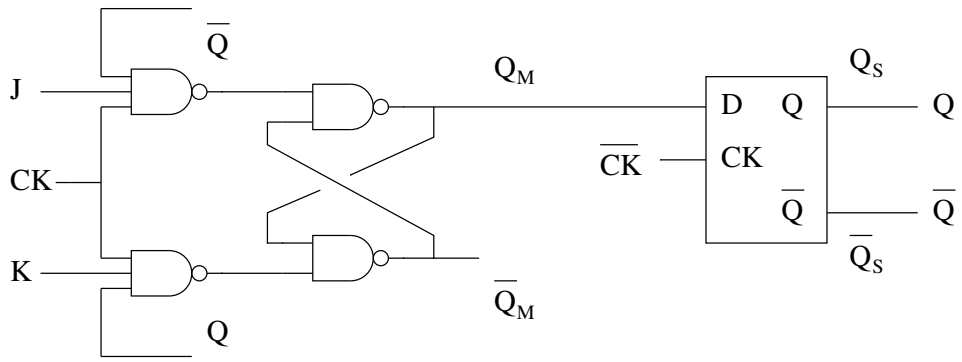


Figura 12.17: Exemplo de implementação de *flip-flop* JK, a partir de *flip-flop* SR *clocked*, sem problema de oscilação, devido ao uso de estrutura *master-slave*.

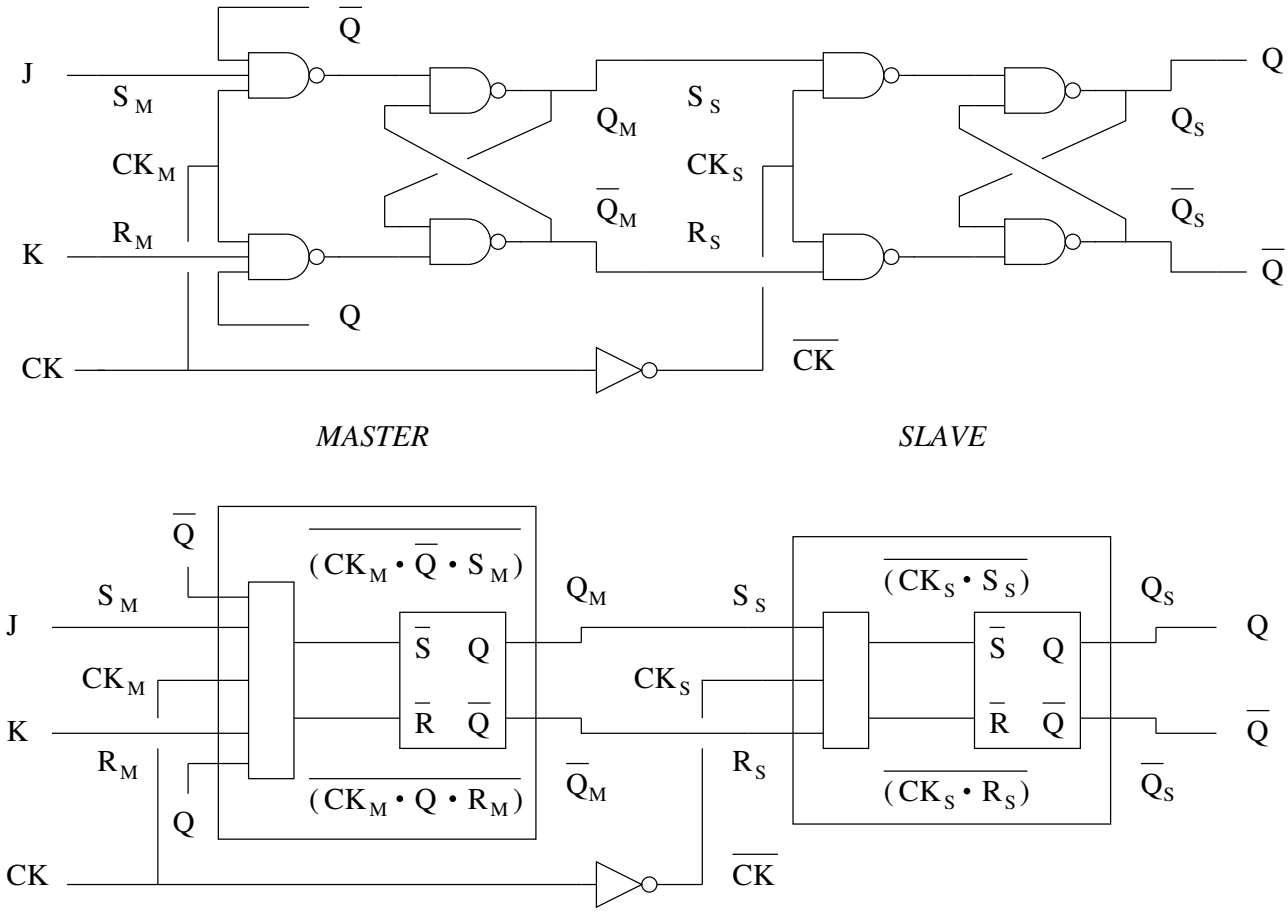


Figura 12.18: Exemplo 1 de implementação de *flip-flop* JK do tipo *master-slave*.

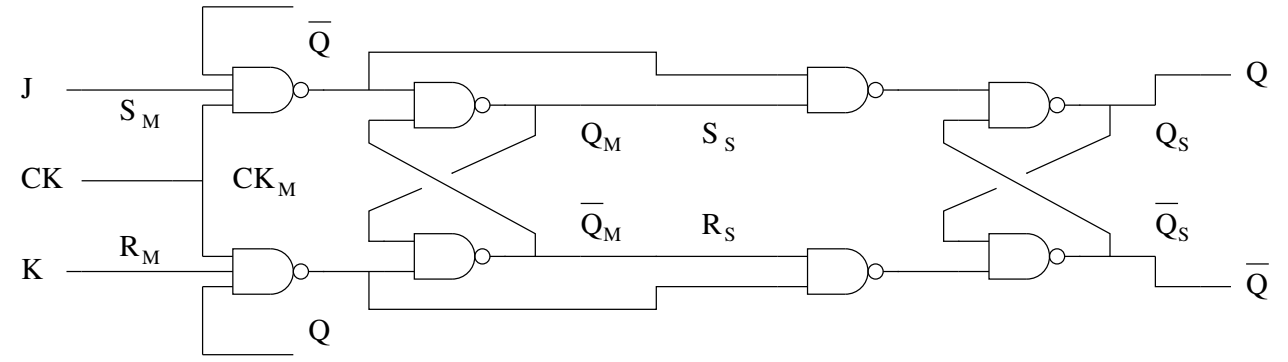


Figura 12.19: Exemplo 2 de implementação de *flip-flop* JK do tipo *master-slave*.

12.11 Variações de funcionalidade

- De acordo com o circuito implementado, um *flip-flop* pode apresentar algumas variações nas suas características funcionais.
- Saídas disponíveis: simples (Q) ou dupla e complementar (Q e $\overline{Q}$).
- Entradas para inicialização da saída: *CLEAR* ($Q = 0$) e *PRESET* ($Q = 1$).
- Tipo de ativação dos sinais de entrada: nível baixo (nível lógico "0") ou nível alto (nível lógico "1").
- Tipo de ativação dos sinais de controle: nível (baixo ou alto), borda (descida ou subida), transição (subida ou descida) ou pulso (negativo ou positivo).

12.12 Diferenças de nomenclatura

- Diversas nomenclaturas diferentes podem ser encontradas na literatura técnica.
- Utilizando como referência os tipos aqui definidos, as nomenclaturas mais comumente encontradas são apresentados na Tabela 12.6.

Nomenclatura	Nomes	Tipos aqui definidos
N1	*Flip-flop*	Todos (os tipos *unclocked* e *clocked* elementar são considerados *flip-flops* elementares)
N2	*Latch*	*unclocked* e *clocked* elementar
	Flip-flop	*clocked master-slave* e *clocked edge-triggered*
N3	*Latch*	*unclocked* e *clocked* elementar
	Latch master-slave	*clocked master-slave*
	Flip-flop	*clocked edge-triggered*
N4	*Latch*	*unclocked*
	Controlled/Clocked-latch	*clocked* elementar
N5	*Positive/Negative-edge flip-flop*	*clocked master-slave*

Tabela 12.6: Diferentes nomenclaturas para *flip-flops*.

Capítulo 13

Circuitos seqüenciais *clock-mode*

13.1 Introdução

- A Figura 13.1 ilustra um modelo genérico para circuitos seqüenciais *clock-mode*.

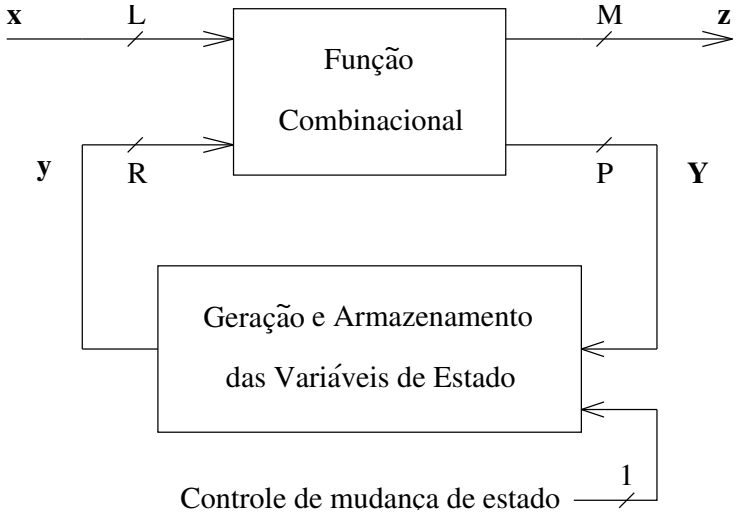


Figura 13.1: Modelo genérico para circuitos seqüenciais *clock-mode*.

- As variáveis de estado são modificadas apenas pela ação de um sinal pulsante, com função de temporização ou de controle, comumente denominado de relógio (*clock*). Apesar de ser um sinal pulsante, não é necessário que o *clock* seja periódico.
- O sinal de *clock* não carrega qualquer tipo de informação. Ele só determina quando haverá mudança de estado.
- As variáveis de excitação, em conjunto com os elementos de armazenamento, determinam qual será a mudança de estado.
- Um *clock* atuando em t_n, com $\boldsymbol{x}^n$, $\boldsymbol{z}^n$, e $\boldsymbol{Y}^n$ estáveis, provoca uma mudança de estado de $\boldsymbol{y}^n$ para $\boldsymbol{y}^{n+1}$.
- O circuito deve estar estável entre dois pulsos de *clock*. Logo, cada circuito possuirá uma freqüência máxima de operação. Tal freqüência será limitada por: i) acionamento dos sinais de entrada, ii) tempos de retardo no bloco "Função Combinacional" e iii) tempos de retardo no bloco "Geração e Armazenamento das Variáveis de Estado".

13.2 Controle de circuitos do tipo *clock-mode*

13.2.1 Características da estrutura *clock-mode*

- $\boldsymbol{z}^n = \boldsymbol{f_1}(\boldsymbol{y}^n, \boldsymbol{x}^n)$, para circuitos do tipo Mealy.
- $\boldsymbol{z}^n = \boldsymbol{f_2}(\boldsymbol{y}^n)$, para circuitos do tipo Moore.
- $\boldsymbol{Y}^n = \boldsymbol{f_3}(\boldsymbol{y}^n, \boldsymbol{x}^n)$.
- $\boldsymbol{y}^{n+1} = \boldsymbol{f_4}(\boldsymbol{Y}^n)$.
- Tempos de propagação:
  - Estabilização da entrada $\boldsymbol{x}$: $\boldsymbol{\Delta t_x}$.
  - Entrada $\boldsymbol{x}$ para saída $\boldsymbol{z}$: $\boldsymbol{\Delta t_{zx}}$.
  - Entrada $\boldsymbol{x}$ para excitação $\boldsymbol{Y}$: $\boldsymbol{\Delta t_{Yx}}$.
  - Excitação $\boldsymbol{Y}$ para estado $\boldsymbol{y}$: $\boldsymbol{\Delta t_{yY}}$.
  - Estado $\boldsymbol{y}$ para saída $\boldsymbol{z}$: $\boldsymbol{\Delta t_{zy}}$.
  - Estado $\boldsymbol{y}$ para excitação $\boldsymbol{Y}$: $\boldsymbol{\Delta t_{Yy}}$.
  - Tempo máximo de propagação: $\Delta t_{max} = max\{\boldsymbol{\Delta t}\} = max\{\Delta t_1, \Delta t_2, \cdots, \Delta t_k\}$.
- Condições de correta operação:
  - Para uma leitura correta dos sinais de saída $\boldsymbol{z}$, os mesmos devem estar estáveis no momento da leitura.
  - Para uma operação previsível do bloco Geração e Armazenamento das Variáveis de Estado (G&A), as variáveis de excitação $\boldsymbol{Y}$ devem estar estáveis no momento do acionamento do bloco.

13.2.2 Controle de circuitos do tipo Moore

- Será assumindo como $\Delta t_x \geq max\{\boldsymbol{\Delta t_x}\}$ o intervalo de tempo entre o acionamento do bloco G&A e a estabilização dos sinais de entrada $\boldsymbol{x}$.
- Assumindo que as variáveis de estado $\boldsymbol{y}$ estejam estáveis, as variáveis de excitação $\boldsymbol{Y}$ estarão estáveis após um tempo $\Delta t_{Yx} \geq max\{\boldsymbol{\Delta t_{Yx}}\}$, a partir da estabilização dos sinais de entrada $\boldsymbol{x}$.
- Assumindo que todos os sinais estejam estáveis, as variáveis de estado $\boldsymbol{y}$ estarão estáveis após um tempo $\Delta t_{yY} \geq max\{\boldsymbol{\Delta t_{yY}}\}$, a partir do acionamento do bloco G&A.
- As variáveis de saída $\boldsymbol{z}$ estarão estáveis após um tempo $\Delta t_{zy} \geq max\{\boldsymbol{\Delta t_{zy}}\}$, a partir da estabilização dos sinais de estado $\boldsymbol{y}$.
- Assumindo que os sinais de entrada $\boldsymbol{x}$ estejam estáveis, as variáveis de excitação $\boldsymbol{Y}$ estarão estáveis após um tempo $\Delta t_{Yy} \geq max\{\boldsymbol{\Delta t_{Yy}}\}$, a partir da estabilização das variáveis de estado $\boldsymbol{y}$.

- Uma vez que, nos circuitos do tipo Moore, a saída depende apenas das variáveis de estado, só é possível ler um valor de saída diferente a cada estado. Assim, ainda que a entrada varie durante o período de tempo de um estado, haverá interesse apenas no seu valor estável final, antes do próximo acionamento que causará uma mudança de estado. Portanto, é necessário considerar apenas o tempo total de estabilização dos sinais de entrada $\boldsymbol{x}$.
- Logo, para cumprir as condições de correta operação, o período de acionamento do bloco G&A deve ser

$$T_{CTRL} = T_{CK} \geq max\{(\Delta t_x + \Delta t_{Yx}), (\Delta t_{yY} + \Delta t_{Yy}), (\Delta t_{yY} + \Delta t_{zy})\} \ . \tag{13.1}$$

- É recomendável que se utilize

$$\Delta t_{yY} < (\Delta t_x + \Delta t_{Yx}) < max\{(\Delta t_{yY} + \Delta t_{Yy}), (\Delta t_{yY} + \Delta t_{zy})\} \ . \tag{13.2}$$

13.2.3 Controle de circuitos do tipo Mealy

- Será assumindo como $\Delta t_x \geq max\{\boldsymbol{\Delta t_x}\}$ o intervalo de tempo entre o acionamento do bloco G&A e a estabilização dos sinais de entrada $\boldsymbol{x}$.
- Assumindo que as variáveis de estado $\boldsymbol{y}$ estejam estáveis, as variáveis de saída $\boldsymbol{z}$ estarão estáveis após um tempo $\Delta t_{zx} \geq max\{\boldsymbol{\Delta t_{zx}}\}$, a partir da estabilização dos sinais de entrada $\boldsymbol{x}$.
- Assumindo que as variáveis de estado $\boldsymbol{y}$ estejam estáveis, as variáveis de excitação $\boldsymbol{Y}$ estarão estáveis após um tempo $\Delta t_{Yx} \geq max\{\boldsymbol{\Delta t_{Yx}}\}$, a partir da estabilização dos sinais de entrada $\boldsymbol{x}$.
- Assumindo que todos os sinais estejam estáveis, as variáveis de estado $\boldsymbol{y}$ estarão estáveis após um tempo $\Delta t_{yY} \geq max\{\boldsymbol{\Delta t_{yY}}\}$, a partir do acionamento do bloco G&A.
- Assumindo que os sinais de entrada $\boldsymbol{x}$ estejam estáveis, as variáveis de saída $\boldsymbol{z}$ estarão estáveis após um tempo $\Delta t_{zy} \geq max\{\boldsymbol{\Delta t_{zy}}\}$, a partir da estabilização dos sinais de estado $\boldsymbol{y}$.
- Assumindo que os sinais de entrada $\boldsymbol{x}$ estejam estáveis, as variáveis de excitação $\boldsymbol{Y}$ estarão estáveis após um tempo $\Delta t_{Yy} \geq max\{\boldsymbol{\Delta t_{Yy}}\}$, a partir da estabilização das variáveis de estado $\boldsymbol{y}$.
- Logo, para cumprir as condições de correta operação, supondo uma única mudança nos sinais de entrada a cada estado, o período de acionamento do bloco G&A deve ser

$$T_{CTRL} = T_{CK} \geq max\{(\Delta t_x + \Delta t_{zx}), (\Delta t_x + \Delta t_{Yx}), (\Delta t_{yY} + \Delta t_{zy}), (\Delta t_{yY} + \Delta t_{Yy})\} \ . \tag{13.3}$$

- Nesse caso, é recomendável que se utilize

$$\Delta t_{yY} < (\Delta t_x + \Delta t_{Yx}) < max\{(\Delta t_{yY} + \Delta t_{zy}), (\Delta t_{yY} + \Delta t_{Yy})\} \tag{13.4}$$

e

$$(\Delta t_x + \Delta t_{zx}) < max\{(\Delta t_{yY} + \Delta t_{zy}), (\Delta t_{yY} + \Delta t_{Yy})\} \ . \tag{13.5}$$

13.3 Representação dos estados

- Recursos comuns: texto, equações, tabelas, diagramas gráficos, diagramas temporais.
- Equações: equações de definição dos elementos de memória, equações de próximo estado.
- Tabelas: tabela de transição (de estados), tabela de atribuição de estados e tabela (de transição) de estados.
- Diagramas gráficos: diagrama de fluxo (fluxograma) e diagrama de estados.

13.4 Estado inicial

- Os circuitos seqüenciais, dependendo de sua classe, devem ou podem apresentar um estado explícito de inicialização (*reset state*).
- O estado inicial pode ser um estado extra ou apenas um dos estados já pertencentes à operação normal do circuito.
- Associada ao estado de inicialização, deve haver uma seqüência de inicialização (*reset sequence* ou *synchronizing sequence*).
- Normalmente, a seqüência de inicialização é fornecida por um único e particular sinal de entrada, denominado sinal ou linha de inicialização (*reset line*).
- O sinal de inicialização pode atuar sobre os elementos de memória através das variáveis de excitação ou através de entradas de controle específicas para inicialização (*CLEAR* e *PRESET*), caso existam.

13.5 Classificação quanto à capacidade de memorização

- Circuito com memória não finita
  - Apresenta um estado inicial ou de inicialização (*reset state*).
  - Apresenta um estado final ou um ciclo de estados final.
  - Possui uma seqüência de inicialização (*reset sequence* ou *synchronizing sequence*).
  - Caso particular:
    - Circuito de Moore onde o número de estados distintos é igual ao número de valores distintos de saída, de forma que se possa estabelecer uma correspondência biunívoca entre valores de estados e de saídas ($z_i = y_i, i = 1, 2, \cdots, K$).

- Circuito com memória finita
  - A Figura 13.2 apresenta um circuito com memória finita.
  - Os blocos de retardo unitário $\boldsymbol{D}$ são conjuntos de *flip-flops* do tipo D.
  - Os vetores $\boldsymbol{x}^{n-r}$, $r = 0, 1, \cdots, R$, e $\boldsymbol{z}^{n-s}$, $s = 0, 1, \cdots, S$, representam os sinais de entrada x_i^{n-r}, $i = 1, 2, \cdots, L$, e de saída z_j^{n-s}, $j = 1, 2, \cdots, M$, respectivamente.
  - Neste tipo de circuito: $\boldsymbol{z}^n = f(\boldsymbol{x}^n, \boldsymbol{x}^{n-1}, \cdots, \boldsymbol{x}^{n-R}, \boldsymbol{z}^{n-1}, \cdots, \boldsymbol{z}^{n-S})$.
  - O valor $P = \max\{R, S\}$ é definido como comprimento ou profundidade da memória.
  - Dependendo do projeto, pode haver um estado de inicialização explícito, com uma seqüência de inicialização associada.
  - Um circuito com memória finita pode ser empregado como passo inicial para uma solução com memória não finita.
  - As Figuras 13.3 e 13.4 destacam, respectivamente, dois casos particulares:
    - Circuitos com memória de entrada finita: $\boldsymbol{z}^n = f(\boldsymbol{x}^n, \boldsymbol{x}^{n-1}, \cdots, \boldsymbol{x}^{n-R})$.
    - Circuitos com memória de saída finita: $\boldsymbol{z}^n = g(\boldsymbol{z}^{n-1}, \cdots, \boldsymbol{z}^{n-S})$.

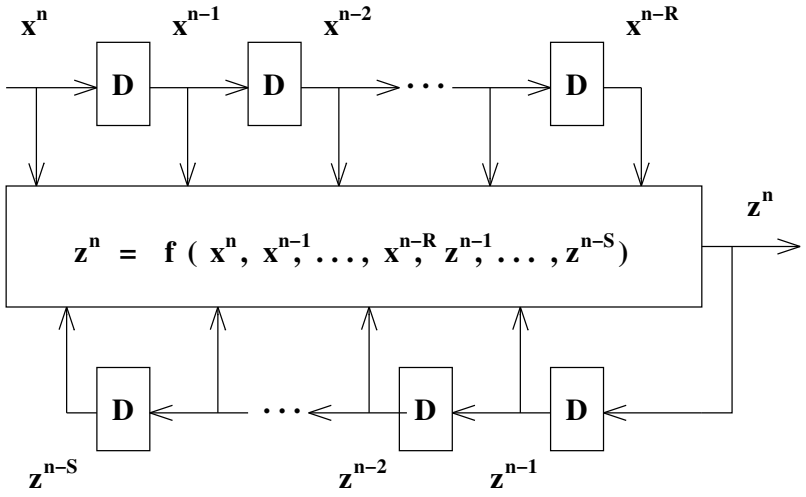


Figura 13.2: Modelo genérico para circuitos com memória finita.

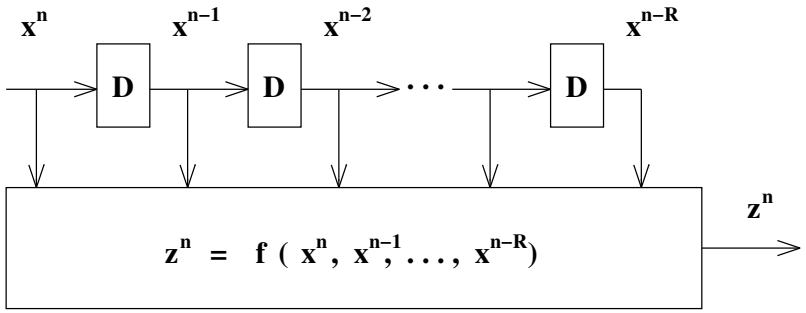


Figura 13.3: Modelo genérico para circuitos com memória finita de entrada.

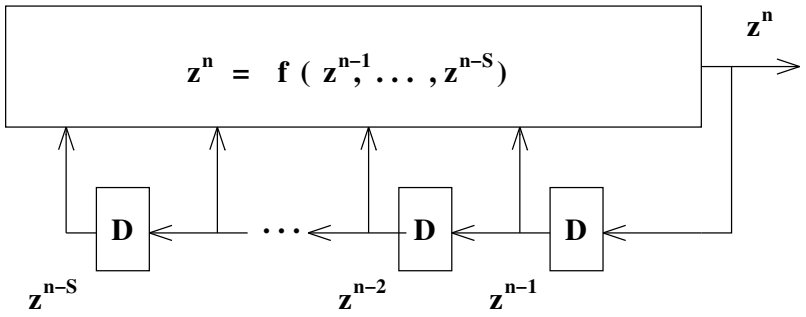


Figura 13.4: Modelo genérico para circuitos com memória finita de saída.

13.6 Análise de circuitos seqüenciais

Dado um circuito digital seqüencial, existem algumas etapas genéricas para a análise do seu comportamento. A seguir, tais etapas são abordadas e alguns exemplos são apresentados.

13.6.1 Etapas genéricas para análise

Dado um circuito com memória não finita genérico, os passos principais para a sua análise são os seguintes:

A1 - Circuito a ser analisado.

A2 - Equações das variáveis de saída, baseadas nas ligações do circuito.

A3 - Equações das variáveis de excitação, baseadas nas ligações do circuito.

A4 - Equações de próximo estado, baseadas na operação dos elementos de memória.

A5 - Tabela de transição de estados (*transition table*), contendo os valores das variáveis de estado.

A6 - Tabela de atribuição de estados, associando nomes aos valores das variáveis de estado.

A7 - Tabela de transição de estados (*state table*), contendo os nomes atribuídos aos estados.

A8 - Diagrama de estados.

13.6.2 Exemplos de análise

Podem-se considerar os seguintes exemplos de análise:

- Circuito com memória finita: circuito com memória finita de entrada, circuito com memória finita de saída e circuito com memória finita de entrada e de saída.
- Caso particular de circuito de Moore onde o número de estados distintos é igual ao número de valores distintos de saída, de forma que se possa estabelecer uma correspondência biunívoca entre valores de estados e de saídas ($z_i = y_i, i = 1, 2, \cdots, K$).
- Circuito com memória não finita genérico.

Circuito com memória não finita genérico

Exemplo 1: Máquina de Mealy.

Supondo-se o circuito apresentado na Figura 11.2, implementado com *flip-flops* do tipo D e com estado inicial dado por $(y_1 y_0)^{n=0} = 10$, serão seguidos os passos de análise definidos anteriormente.

As equações das suas variáveis de saída são dadas por

$$z_0^n = (x_0^n \text{ NAND } y_0^n)$$

e

$$z_1^n = (y_1^n \text{ NAND } y_0^n) \ .$$

As equações das suas variáveis de excitação são dadas por

$$Y_0^n = z_1^n = (y_1^n \text{ NAND } y_0^n)$$

e

$$Y_1^n = (x_1^n \text{ NOR } y_1^n) \ .$$

Supondo-se o uso de *flip-flops* do tipo D, as equações das suas variáveis de estado, no próximo instante, são dadas por

$$y_0^{(n+1)} = Y_0^n$$

e

$$y_1^{(n+1)} = Y_1^n \ .$$

Baseado nas equações anteriores, a sua *transition table* é apresentada na Tabela 13.1.

$(y_1y_0)^n$	$(y_1y_0)^{n+1}$				$(z_1z_0)^n$				
	00	01	11	10	00	01	11	10	$\leftarrow (x_1x_0)^n$
00	11	11	01	01	11	11	11	11	
01	11	11	01	01	11	10	10	11	
11	00	00	00	00	01	00	00	01	
10	01	01	01	01	11	11	11	11	

Tabela 13.1: *Transition table* do exemplo de análise de uma Máquina de Mealy.

Associando-se nomes aos padrões binários das variáveis de estado, uma possível Tabela de Atribuição de Estados é apresentada na Tabela 13.2.

(y_1y_0)	q
00	q_0
01	q_1
11	q_2
10	q_3

Tabela 13.2: Possível Tabela de Atribuição de Estados para o exemplo de análise de uma Máquina de Mealy.

q^n	q^{n+1}				z^n				
	00	01	11	10	00	01	11	10	$\leftarrow (x_1x_0)^n$
q_0	q_2	q_2	q_1	q_1	3	3	3	3	
q_1	q_2	q_2	q_1	q_1	3	2	2	3	
q_2	q_0	q_0	q_0	q_0	0	0	0	1	
q_3	q_1	q_1	q_1	q_1	3	3	3	3	

Tabela 13.3: *State table* do exemplo de análise de uma Máquina de Mealy.

Com o emprego da Tabela de Atribuição de Estados e associando-se números decimais aos padrões binários das variáveis de saída, a sua *transition table* é transformada na sua *state table*, apresentada na Tabela 13.3.

Exemplo 2: Máquina de Moore.

Supondo-se o circuito apresentado na Figura 11.3, implementado com *flip-flops* do tipo D e com estado inicial dado por $(y_1y_0)^{n=0} = 01$, serão seguidos os passos de análise definidos anteriormente.

A equação da sua variável de saída é dada por

$$z_0^n = (y_1^n \text{ NAND } y_0^n) \ .$$

As equações das suas variáveis de excitação são dadas por

$$Y_0^n = (x_1^n \text{ NOR } y_1^n)$$

e

$$Y_1^n = (x_0^n \text{ NAND } y_0^n) \ .$$

Supondo-se o uso de *flip-flops* do tipo D, as equações das suas variáveis de estado, no próximo instante, são dadas por

$$y_0^{(n+1)} = Y_0^n$$

e

$$y_1^{(n+1)} = Y_1^n \ .$$

Baseado nas equações anteriores, a sua *transition table* é apresentada na Tabela 13.4.

$(y_1y_0)^n$	$(y_1y_0)^{n+1}$				$(z_0)^n$	
	00	01	11	10		$\leftarrow (x_1x_0)^n$
00	11	11	10	10	1	
01	11	01	00	10	1	
11	10	00	00	10	0	
10	10	10	10	10	1	

Tabela 13.4: *Transition table* do exemplo de análise de uma Máquina de Moore.

Associando-se nomes aos padrões binários das variáveis de estado, uma possível Tabela de Atribuição de Estados é apresentada na Tabela 13.5.

Com o emprego da Tabela de Atribuição de Estados e associando-se números decimais aos padrões binários das variáveis de saída, a sua *transition table* é transformada na sua *state table*, apresentada na Tabela 13.6.

$(y_1 y_0)$	q
00	q_0
01	q_1
11	q_2
10	q_3

Tabela 13.5: Possível Tabela de Atribuição de Estados para o exemplo de análise de uma Máquina de Moore.

q^n	q^{n+1}				z^n	
	00	01	11	10		$\leftarrow (x_1 x_0)^n$
q_0	q_2	q_2	q_3	q_3	1	
q_1	q_2	q_1	q_0	q_3	1	
q_2	q_3	q_0	q_0	q_3	0	
q_3	q_3	q_3	q_3	q_3	1	

Tabela 13.6: *State table* do exemplo de análise de uma Máquina de Moore.

13.7 Projeto de circuitos seqüenciais

- Uma vez que a síntese é o processo reverso em relação à análise, as etapas de projeto podem ser obtidas, a princípio, revertendo-se a ordem das etapas de análise.
- Porém, existe uma profunda diferença entre os dois processos. Na análise, há um único circuito, uma única entrada e um único estado inicial. Portanto, uma saída única é obtida no processo. Por outro lado, no processo de síntese há uma entrada e uma saída, únicas, e se procura por um circuito que realize o mapeamento entrada-saída. A solução nesse caso raramente é única, pois, em cada passo do processo de síntese, decisões podem ser fazer necessárias, gerando uma árvore de opções.
- A seguir, são comentadas as características de projeto para cada um dos tipos de circuito *clock-mode* acima definidos, bem como são especificadas as etapas de projeto para tais circuitos e são apresentados alguns exemplos.

13.7.1 Opções de projeto e suas características

Podem ser identificadas três classes de projetos:

- Circuito com memória finita: ausência de lógica combinacional (ligação por meio de fios) na geração das variáveis de excitação.
- Caso particular de circuito de Moore onde o número de estados distintos é igual ao número de valores distintos de saída, de forma que se possa estabelecer uma correspondência biunívoca entre valores de estados e de saídas ($z_i = y_i, i = 1, 2, \cdots, K$): ausência de lógica combinacional (ligação por meio de fios) na geração das variáveis de saída.
- Circuito com memória não finita genérico: possível existência de lógica combinacional na geração das variáveis de excitação e de saída, a qual pode ser minimizada.

A Figura 13.5 apresenta os fluxos de projeto para cada uma das três opções.

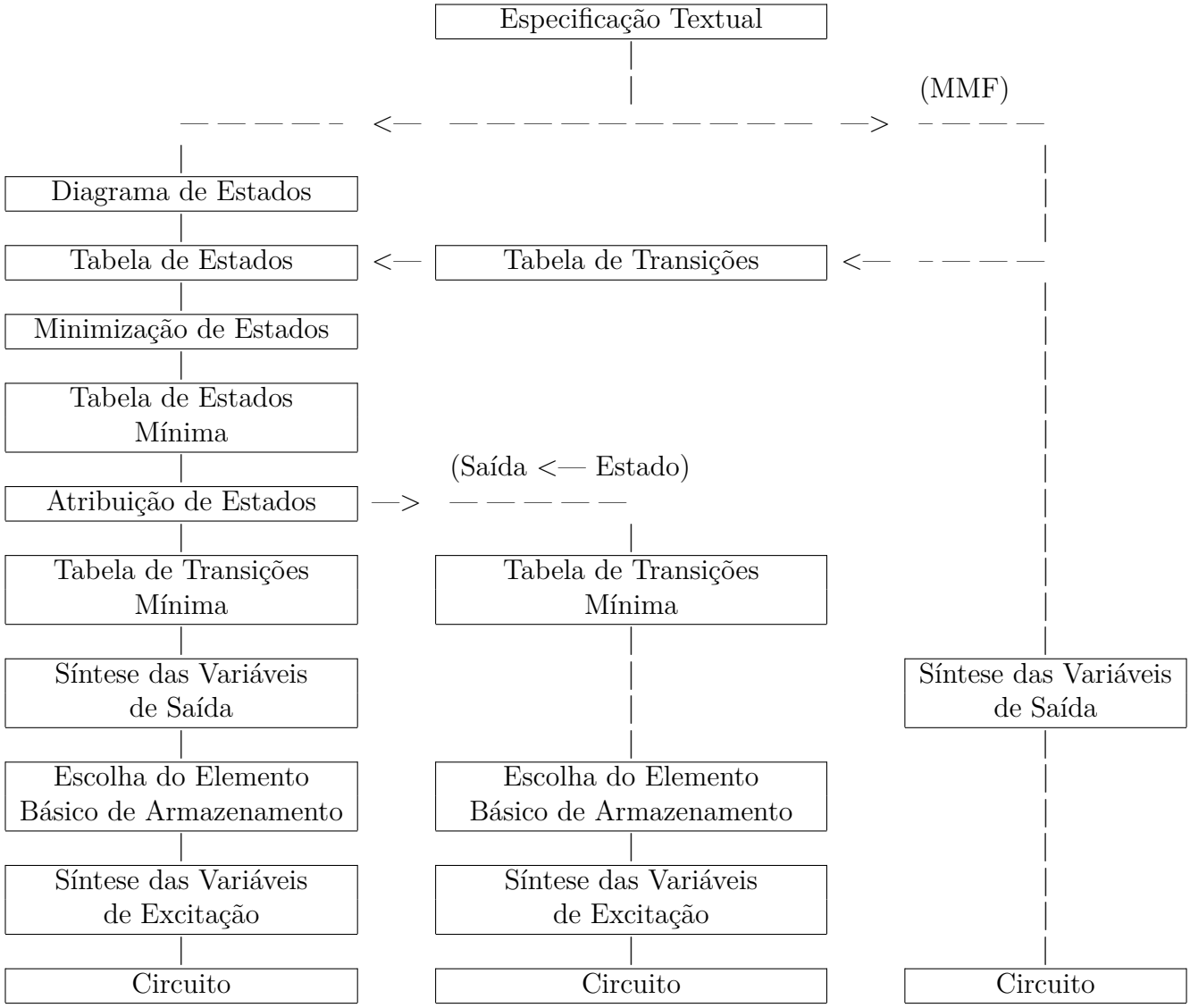


Figura 13.5: Fluxos de projeto para circuitos seqüenciais *clock-mode*: (a) Fluxo genérico, (b) Caso particular de Máquina de Moore e (c) Máquina de Memória Finita (MMF).

13.7.2 Etapas genéricas para projeto

Os três tipos de projeto definidos na Figura 13.5 possuem etapas que são particulares para cada caso. Porém, pode-se definir um fluxo geral de projeto, que atenda a todos os três tipos. Assim, dependendo do tipo de projeto, pode-se utilizar apenas as etapas necessárias a cada caso.

As etapas genéricas para projeto são as seguintes:

P1 - Definição do problema a ser resolvido.

P2 - Descrição funcional do problema (descrição textual).

P3 - Descrição diagramática, baseada na descrição textual:

- Diagrama de fluxo (fluxograma).
- Diagrama de estados.

P4 - Tabela de transição de estados (*state table*):

- Diretamente obtida da descrição funcional (circuito com memória finita).
- Baseada na descrição diagramática (circuito com memória não finita).

P5 - Tentativa de minimização da *state table*. Raramente é feita uma minimização global, que envolva o circuito combinacional e a memória ao mesmo tempo. Ao invés disso, o mais comum é que se realize o processo em duas etapas:

P5.1 - Memória: tentativa de redução do número de estados e, possivelmente, do número de variáveis de estado, baseada em técnicas de minimização de estados, gerando-se um tabela de transição de estados reduzida (*minimal-state table*),

P5.2 - Combinacional: dependente da classe do circuito a ser projetado. No caso de circuito com memória finita e no caso particular de circuito de Moore, a minimização combinacional já é uma característica intrínseca da estrutura. No caso de circuito com memória não finita, tal minimização é realizada no passo **P6**.

P6 - Tabela de atribuição de estados, baseada em regras genéricas para escolha da melhor atribuição, buscando minimizar a parte combinacional do circuito.

P7 - Tabela de transição de estados (*transition table*):

- Diretamente da especificação do problema (circuito com memória finita).
- Diretamente da especificação das variáveis de saída (caso particular de circuito de Moore com memória não finita).
- Baseada na atribuição de estados (circuito com memória não finita genérico).

P8 - Equações das variáveis de saída, obtidas diretamente da tabela de transição de estados (*transition table*).

P9 - Escolha dos elementos de memória (*flip-flops*).

P10 - Equações das variáveis de excitação, que são as equações de entrada dos elementos de memória. No caso de *flip-flops* do tipo D, são obtidas diretamente da tabela de transição de estados (*transition table*). No caso de *flip-flops* do tipo JK, são obtidas da tabela de transição de estados (*transition table*) em conjunto com as tabelas de excitação do elemento de memória (*excitation table/map* ou *transition list/table/map*).

P11 - Circuito final proposto, com a ligação dos sinais de controle e de inicialização.

P12 - Análise do circuito final, para verificação de comportamento dos estados não utilizados e não especificados, caso existam.

13.7.3 Exemplos de projeto

Podem-se considerar os seguintes exemplos de projeto:

- Circuito com memória finita: circuito com memória finita de entrada, circuito com memória finita de saída e circuito com memória finita de entrada e de saída.
- Caso particular de circuito de Moore onde o número de estados distintos é igual ao número de valores distintos de saída, de forma que se possa estabelecer uma correspondência biunívoca entre valores de estados e de saídas ($z_i = y_i, i = 1, 2, \cdots, K$).
- Circuito com memória não finita genérico.

Circuito com memória não finita genérico

Exemplo 1: Máquina de Moore - Contador.

As especificações básicas são:

- É suposta uma máquina de Moore, cuja saída deve gerar a seguinte contagem modular: $\boldsymbol{z}^n = \{\cdots, 0, 5, 9, 15, 0, 5, 9, 15, \cdots\}$.
- A entrada deve informar o fluxo da contagem. Se $x^n = 0$, a contagem deve ser crescente. Se $x^n = 1$, a contagem deve ser decrescente.
- Como estado inicial, é assumido o estado onde $z^{n=0} = 0$.
- No projeto inicial, serão empregados *flip-flops* do tipo JK.
- Em seguida, será feito um projeto alternativo, empregando *flip-flops* do tipo D.
- Para facilitar a inicialização da máquina, os *flip-flops* deverão ter controles assíncronos de *clear* e *preset*.

A partir das especificações fornecidas, pode-se montar o diagrama de estados apresentado na Figura 13.6.

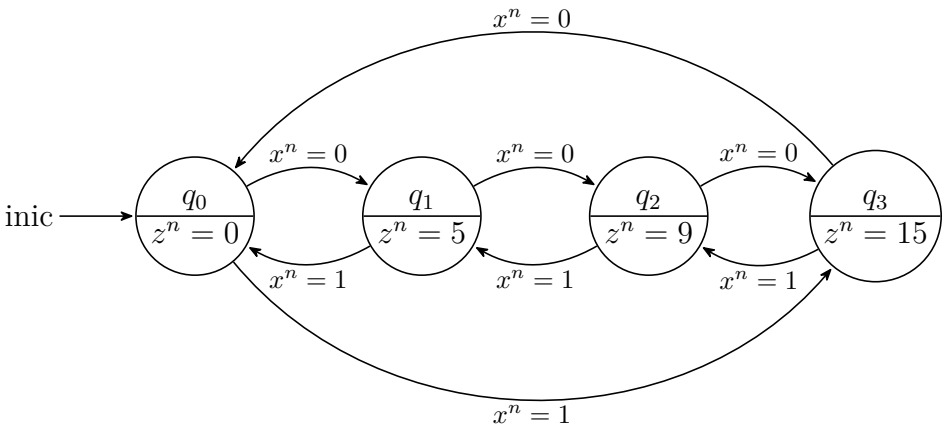


Figura 13.6: Diagrama de estados do exemplo de projeto de uma Máquina de Moore - Contador.

A partir do diagrama de estados proposto, pode-se montar a tabela de estados apresentada na Tabela 13.7.

q^n	q^{n+1}		z^n
	$x^n = 0$	$x^n = 1$	
q_0	q_1	q_3	0
q_1	q_2	q_0	5
q_2	q_3	q_1	9
q_3	q_0	q_2	15

Tabela 13.7: *State table* do exemplo de projeto de uma Máquina de Moore - Contador.

Assumindo-se uma atribuição de estados trivial, pode-se montar a tabela de atribuição de estados apresentada na Tabela 13.8. Uma vez que o estado inicial $q^{n=0}$ (onde $z^{n=0} = 0$) foi denominado de q_0 e a este foi atribuído o padrão $y_1y_0 = 00$, a inicialização do contador pode ser feita atribuindo-se aos controles assíncronos dos *flip-flops* os seguintes sinais: $clear = inic$ e $preset = 0$.

(y_1y_0)	q
00	q_0
01	q_1
10	q_2
11	q_3

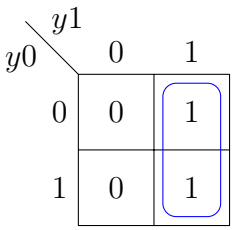
Tabela 13.8: Possível Tabela de Atribuição de Estados para o exemplo de projeto de uma Máquina de Moore - Contador.

A partir da atribuição de estados adotada, a tabela de estados pode ser transformada na tabela de transição apresentada na Tabela 13.9.

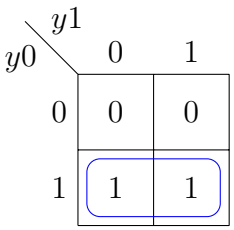
$(y_1y_0)^n$	$(y_1y_0)^{n+1}$		$(z_3z_2z_1z_0)^n$
	$x^n = 0$	$x^n = 1$	
00	01	11	0000
01	10	00	0101
10	11	01	1001
11	00	10	1111

Tabela 13.9: *Transition table* do exemplo de projeto de uma Máquina de Moore - Contador.

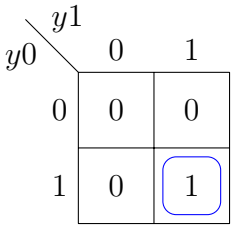
A partir da tabela de transição, pode-se realizar diretamente a síntese das funções que geram as variáveis de saída. Isso é ilustrado nos mapas de Karnaugh da Figura 13.7, onde encontram-se marcadas as formas SOP mínimas. Destes mapas, obtêm-se as seguintes funções: $z_3^n = y_1^n$, $z_2^n = y_0^n$, $z_1^n = (y_1^n \cdot y_0^n)$ e $z_0^n = (y_1^n + y_0^n)$.



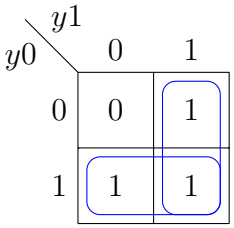
(a) Minimização da função z_3.



(b) Minimização da função z_2.



(c) Minimização da função z_1.



(d) Minimização da função z_0.

Figura 13.7: Mapas de Karnaugh das variáveis de saída para o exemplo de projeto de uma Máquina de Moore - Contador.

Na síntese das funções que geram as variáveis de excitação, deve-se levar em consideração que elas são as variáveis de entrada dos *flip-flops*, os quais, quando disparados, irão promover a mudança de estado. Portanto, para esta síntese, deve-se usar a tabela de transição juntamente com a tabela de excitação dos *flip-flops* escolhidos.

Para um primeiro projeto, serão utilizados *flip-flops* do tipo JK, cuja tabela de excitação é apresentada na Tabela 13.10.

$Q^n \rightarrow Q^{n+1}$	J^n	K^n
$0 \rightarrow 0$	0	X
$0 \rightarrow 1$	1	X
$1 \rightarrow 0$	X	1
$1 \rightarrow 1$	X	0

Tabela 13.10: Tabela de excitação para o *flip-flop* JK.

Dessa forma, empregando-se a tabela de transição e a tabela de excitação do *flip-flop* JK, pode-se montar a tabela verdade de cada variável de excitação, como ilustrado na Tabela 13.11.

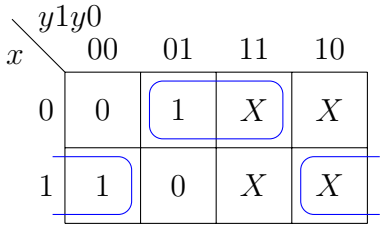
A partir da tabela verdade de cada variável de excitação, pode-se realizar diretamente a síntese das funções que geram tais variáveis. Isso é ilustrado nos mapas de Karnaugh da Figura 13.8, onde encontram-se marcadas as formas SOP mínimas. Destes mapas, obtêm-se as seguintes funções: $Y_3^n = J_1^n = (y_0^n \cdot \overline{x^n}) + (\overline{y_0^n} \cdot x^n)$, $Y_2^n = K_1^n = (y_0^n \cdot \overline{x^n}) + (\overline{y_0^n} \cdot x^n)$, $Y_1^n = J_0^n = 1$ e $Y_0^n = K_0^n = 1$.

Por fim, observando-se que as entradas J e K são iguais, podem-se trocar estes *flip-flops* JK por *flip-flops* T, com entradas $T_1^n = J_1^n = K_1^n = (y_0^n \cdot \overline{x^n}) + (\overline{y_0^n} \cdot x^n)$ e $T_0^n = J_0^n = K_0^n = 1$.

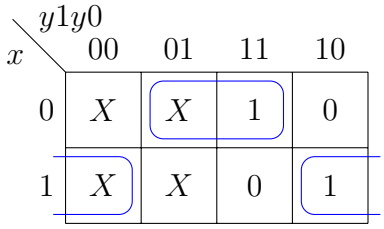
Além disso, no caso do *flip-flop* T onde a entrada é igual a "1", pode-se trocá-lo por um *flip-flop* T sem a entrada "T".

x^n	y_1	y_0	$Y_3^n = J_1^n$	$Y_2^n = K_1^n$	$Y_1^n = J_0^n$	$Y_0^n = K_0^n$
0	0	0	0	X	1	X
0	0	1	1	X	X	1
0	1	0	X	0	1	X
0	1	1	X	1	X	1
1	0	0	1	X	1	X
1	0	1	0	X	X	1
1	1	0	X	1	1	X
1	1	1	X	0	X	1

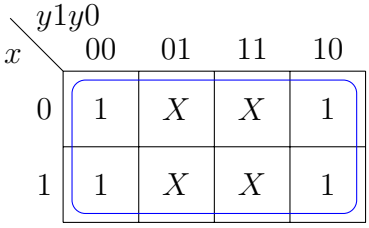
Tabela 13.11: Tabela verdade de cada variável de excitação, para o exemplo de projeto com *flip-flop* JK, de uma Máquina de Moore - Contador.



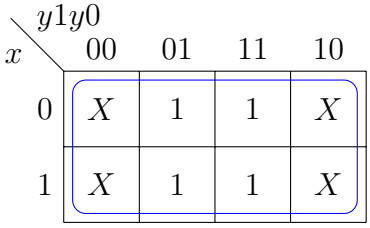
(a) Minimização da função $Y_3^n = J_1^n$.



(b) Minimização da função $Y_2^n = K_1^n$.



(c) Minimização da função $Y_1^n = J_0^n$.



(d) Minimização da função $Y_0^n = K_0^n$.

Figura 13.8: Mapas de Karnaugh das variáveis de excitação, para o exemplo de projeto com *flip-flop* JK, de uma Máquina de Moore - Contador.

Como projeto alternativo, podem ser utilizados *flip-flops* do tipo D, cuja tabela de excitação é apresentada na Tabela 13.12.

$Q^n \rightarrow Q^{n+1}$	D^n
$0 \rightarrow 0$	0
$0 \rightarrow 1$	1
$1 \rightarrow 0$	0
$1 \rightarrow 1$	1

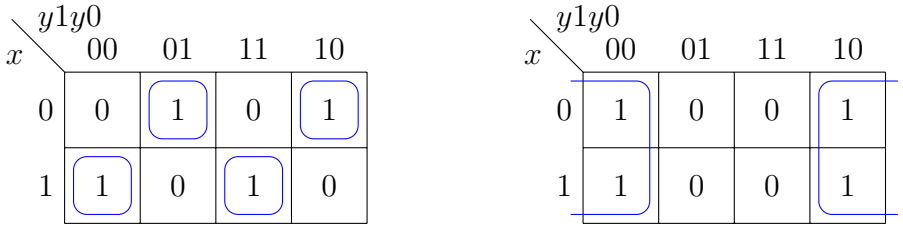
Tabela 13.12: Tabela de excitação para o *flip-flop* D.

Dessa forma, empregando-se a tabela de transição e a tabela de excitação do *flip-flop* D, pode-se montar a tabela verdade de cada variável de excitação, como ilustrado na Tabela 13.13.

x^n	y_1	y_0	$Y_1^n = D_1^n$	$Y_0^n = D_0^n$
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	1
1	1	1	1	0

Tabela 13.13: Tabela verdade de cada variável de excitação, para o exemplo de projeto, com *flip-flop* D, de uma Máquina de Moore - Contador.

A partir da tabela verdade de cada variável de excitação, pode-se realizar diretamente a síntese das funções que geram tais variáveis. Isso é ilustrado nos mapas de Karnaugh da Figura 13.9, onde encontram-se marcadas as formas SOP mínimas. Destes mapas, obtêm-se as seguintes funções: $Y_0^n = D_0^n = \overline{y_0^n}$ e $Y_1^n = D_1^n = (\overline{y_1^n} \cdot \overline{y_0^n} \cdot x^n) + (\overline{y_1^n} \cdot y_0^n \cdot \overline{x^n}) + (y_1^n \cdot y_0^n \cdot x^n) + (y_1^n \cdot \overline{y_0^n} \cdot \overline{x^n}) = \overline{y_1^n} \cdot [(\overline{y_0^n} \cdot x^n) + (y_0^n \cdot \overline{x^n})] + y_1 \cdot [(y_0^n \cdot x^n) + (\overline{y_0^n} \cdot \overline{x^n})] = \overline{y_1^n} \cdot [(\overline{y_0^n} \cdot x^n) + (y_0^n \cdot \overline{x^n})] + y_1 \cdot \overline{[(\overline{y_0^n} \cdot x^n) + (y_0^n \cdot \overline{x^n})]}$.



(a) Minimização da função $Y_1^n = D_1^n$. (b) Minimização da função $Y_0^n = D_0^n$.

Figura 13.9: Mapas de Karnaugh das variáveis de excitação, para o exemplo de projeto com *flip-flop* D, de uma Máquina de Moore - Contador.

Por fim, observando-se as funções encontradas para as variáveis de excitação, podem-se trocar os *flip-flops* D por *flip-flops* T, com entradas $T_1^n = (y_0^n \cdot \overline{x^n}) + (\overline{y_0^n} \cdot x^n)$ e $T_0^n = 1$.

Além disso, no caso do *flip-flop* T onde a entrada é igual a "1", pode-se trocá-lo por um *flip-flop* T sem a entrada "T".

Exemplo 2: Máquina de Mealy - Contador.

As especificações básicas são:

- É suposta uma máquina de Mealy, cujos estados devem evoluir de forma modular, com a seguinte ordenação: $\boldsymbol{q}^n = \{\cdots, q_2, q_3, q_0, q_1, q_2, q_3, q_0, q_1, \cdots\}$.
- Estando no estado $q^n = q_k$, a máquina deve exibir uma saída $z^n = (4k)$, para $x^n = 0$, ou uma saída $z^n = (4k + 3)$, para $x^n = 1$.
- Como estado inicial, é assumido o estado onde $z^{n=0} = 0$, para $x^{n=0} = 0$.
- No projeto inicial, serão empregados *flip-flops* do tipo JK.
- Em seguida, será feito um projeto alternativo, empregando *flip-flops* do tipo D.
- Para facilitar a inicialização da máquina, os *flip-flops* deverão ter controles assíncronos de *clear* e *preset*.

A partir das especificações fornecidas, pode-se montar o diagrama de estados apresentado na Figura 13.10.

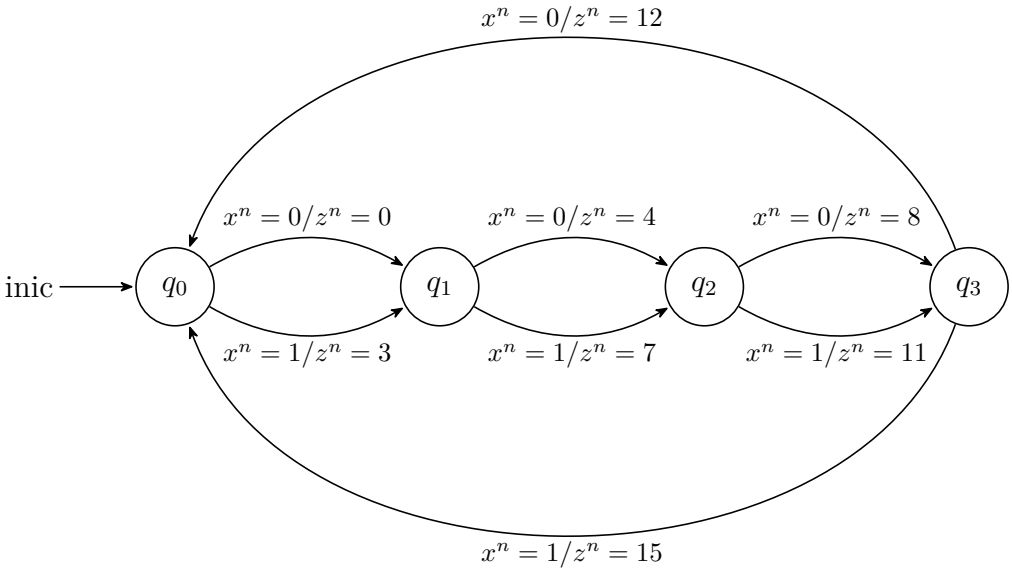


Figura 13.10: Diagrama de estados do exemplo de projeto de uma Máquina de Mealy - Contador.

A partir do diagrama de estados proposto, pode-se montar a tabela de estados apresentada na Tabela 13.14.

q^n	q^{n+1}		z^n	
	$x^n = 0$	$x^n = 1$	$x^n = 0$	$x^n = 1$
q_0	q_1	q_1	0	3
q_1	q_2	q_2	4	7
q_2	q_3	q_3	8	11
q_3	q_0	q_0	12	15

Tabela 13.14: *State table* do exemplo de projeto de uma Máquina de Mealy - Contador.

Assumindo-se uma atribuição de estados trivial, pode-se montar a tabela de atribuição de estados apresentada na Tabela 13.15. Uma vez que o estado inicial $q^{n=0}$ (onde $z^{n=0} = 0$, para $x^{n=0} = 0$) foi denominado de q_0 e a este foi atribuído o padrão $y_1y_0 = 00$, a inicialização do contador pode ser feita atribuindo-se aos controles assíncronos dos *flip-flops* os seguintes sinais: $clear = inic$ e $preset = 0$.

(y_1y_0)	q
00	q_0
01	q_1
10	q_2
11	q_3

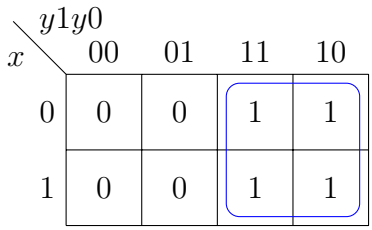
Tabela 13.15: Possível Tabela de Atribuição de Estados para o exemplo de projeto de uma Máquina de Mealy - Contador.

A partir da atribuição de estados adotada, a tabela de estados pode ser transformada na tabela de transição apresentada na Tabela 13.16.

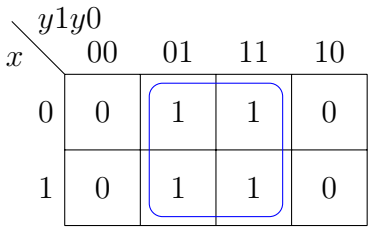
$(y_1y_0)^n$	$(y_1y_0)^{n+1}$		$(z_3z_2z_1z_0)^n$	
	$x^n = 0$	$x^n = 1$	$x^n = 0$	$x^n = 1$
00	01	01	0000	0011
01	10	10	0100	0111
10	11	11	1000	1011
11	00	00	1100	1111

Tabela 13.16: *Transition table* do exemplo de projeto de uma Máquina de Mealy - Contador.

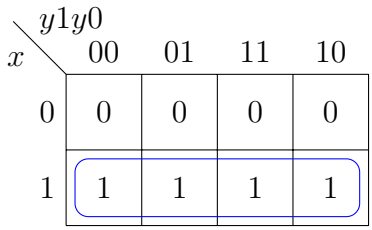
A partir da tabela de transição, pode-se realizar diretamente a síntese das funções que geram as variáveis de saída. Isso é ilustrado nos mapas de Karnaugh da Figura 13.11, onde encontram-se marcadas as formas SOP mínimas. Destes mapas, obtêm-se as seguintes funções: $z_3^n = y_1^n$, $z_2^n = y_0^n$, $z_1^n = x^n$ e $z_0^n = x^n$.



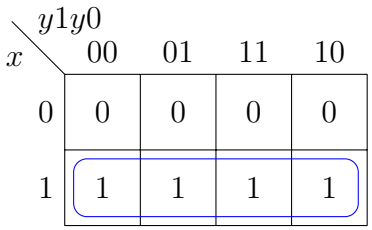
(a) Minimização da função z_3.



(b) Minimização da função z_2.



(c) Minimização da função z_1.



(d) Minimização da função z_0.

Figura 13.11: Mapas de Karnaugh das variáveis de saída para o exemplo de projeto de uma Máquina de Moore - Contador.

Na síntese das funções que geram as variáveis de excitação, deve-se levar em consideração que elas são as variáveis de entrada dos *flip-flops*, os quais, quando disparados, irão promover a mudança de estado. Portanto, para esta síntese, deve-se usar a tabela de transição juntamente com a tabela de excitação dos *flip-flops* escolhidos.

Para um primeiro projeto, serão utilizados *flip-flops* do tipo JK, cuja tabela de excitação é apresentada na Tabela 13.17.

$Q^n \rightarrow Q^{n+1}$	J^n	K^n
$0 \rightarrow 0$	0	X
$0 \rightarrow 1$	1	X
$1 \rightarrow 0$	X	1
$1 \rightarrow 1$	X	0

Tabela 13.17: Tabela de excitação para o *flip-flop* JK.

Dessa forma, empregando-se a tabela de transição e a tabela de excitação do *flip-flop* JK, pode-se montar a tabela verdade de cada variável de excitação, como ilustrado na Tabela 13.18.

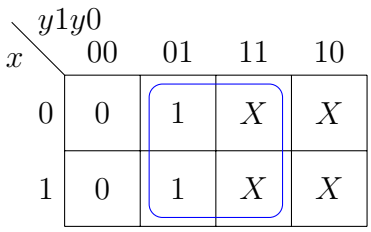
A partir da tabela verdade de cada variável de excitação, pode-se realizar diretamente a síntese das funções que geram tais variáveis. Isso é ilustrado nos mapas de Karnaugh da Figura 13.12, onde encontram-se marcadas as formas SOP mínimas. Destes mapas, obtêm-se as seguintes funções: $Y_3^n = J_1^n = y_0^n$, $Y_2^n = K_1^n = y_0^n$, $Y_1^n = J_0^n = 1$ e $Y_0^n = K_0^n = 1$.

Por fim, observando-se que as entradas J e K são iguais, podem-se trocar estes *flip-flops* JK por *flip-flops* T, com entradas $T_1^n = J_1^n = K_1^n = y_0$ e $T_0^n = J_0^n = K_0^n = 1$.

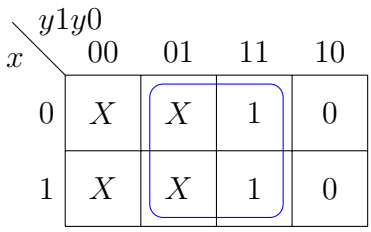
Além disso, no caso do *flip-flop* T onde a entrada é igual a "1", pode-se trocá-lo por um *flip-flop* T sem a entrada "T".

x^n	y_1	y_0	$Y_3^n = J_1^n$	$Y_2^n = K_1^n$	$Y_1^n = J_0^n$	$Y_0^n = K_0^n$
0	0	0	0	X	1	X
0	0	1	1	X	X	1
0	1	0	X	0	1	X
0	1	1	X	1	X	1
1	0	0	0	X	1	X
1	0	1	1	X	X	1
1	1	0	X	0	1	X
1	1	1	X	1	X	1

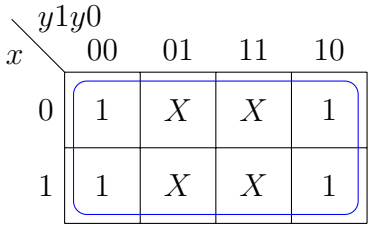
Tabela 13.18: Tabela verdade de cada variável de excitação, para o exemplo de projeto com *flip-flop* JK, de uma Máquina de Mealy - Contador.



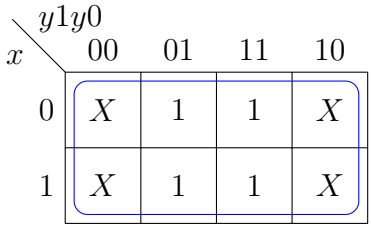
(a) Minimização da função $Y_3^n = J_1^n$.



(b) Minimização da função $Y_2^n = K_1^n$.



(c) Minimização da função $Y_1^n = J_0^n$.



(d) Minimização da função $Y_0^n = K_0^n$.

Figura 13.12: Mapas de Karnaugh das variáveis de excitação, para o exemplo de projeto com *flip-flop* JK, de uma Máquina de Mealy - Contador.

Como projeto alternativo, podem ser utilizados *flip-flops* do tipo D, cuja tabela de excitação é apresentada na Tabela 13.19.

$Q^n \rightarrow Q^{n+1}$	D^n
$0 \rightarrow 0$	0
$0 \rightarrow 1$	1
$1 \rightarrow 0$	0
$1 \rightarrow 1$	1

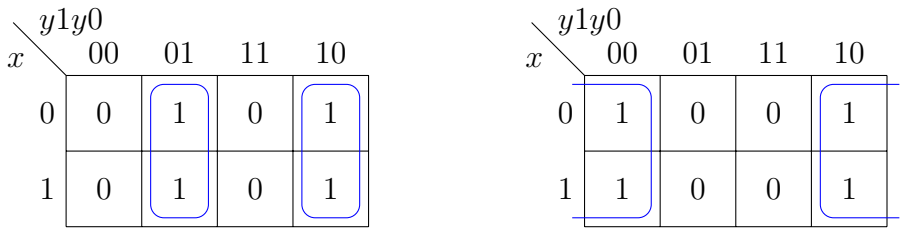
Tabela 13.19: Tabela de excitação para o *flip-flop* D.

Dessa forma, empregando-se a tabela de transição e a tabela de excitação do *flip-flop* D, pode-se montar a tabela verdade de cada variável de excitação, como ilustrado na Tabela 13.20.

x^n	y_1	y_0	$Y_1^n = D_1^n$	$Y_0^n = D_0^n$
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

Tabela 13.20: Tabela verdade de cada variável de excitação, para o exemplo de projeto, com *flip-flop* D, de uma Máquina de Mealy - Contador.

A partir da tabela verdade de cada variável de excitação, pode-se realizar diretamente a síntese das funções que geram tais variáveis. Isso é ilustrado nos mapas de Karnaugh da Figura 13.13, onde encontram-se marcadas as formas SOP mínimas. Destes mapas, obtêm-se as seguintes funções: $Y_1^n = D_1^n = (\overline{y_1^n} \cdot y_0^n) + (y_1^n \cdot \overline{y_0^n})$ e $Y_0^n = D_0^n = \overline{y_0^n}$.



(a) Minimização da função $Y_1^n = D_1^n$. (b) Minimização da função $Y_0^n = D_0^n$.

Figura 13.13: Mapas de Karnaugh das variáveis de excitação, para o exemplo de projeto com *flip-flop* D, de uma Máquina de Mealy - Contador.

Por fim, observando-se as funções encontradas para as variáveis de excitação, podem-se trocar os *flip-flops* D por *flip-flops* T, com entradas $T_1^n = y_0^n$ e $T_0^n = 1$.

Além disso, no caso do *flip-flop* T onde a entrada é igual a "1", pode-se trocá-lo por um *flip-flop* T sem a entrada "T".

13.8 Minimização de estados

13.8.1 Conceitos básicos

A minimização do número de estados de um circuito seqüencial pode conduzir à redução da quantidade de circuitos lógicos necessários para implementar os estados (bloco Geração e Armazenamento) e as saídas (bloco Função Combinacional).

Dada uma tabela de transição de estados (*state table*), pode-se constatar que diferentes estados podem realizar a mesma função. Do ponto de vista externo ao circuito, pode-se dizer que não é possível distinguir entre tais estados, uma vez que eles apresentam o mesmo resultado. Nesse caso, tal conjunto de estados pode ser representado por um único estado. Como resultado, a tabela de transição de estados (*state table*) é simplificada e, possivelmente, o circuito lógico minimizado.

Uma formalismo teórico para tais conceitos é apresentado no Apêndice D.

13.8.2 Eliminação de estados redundantes por simples inspeção

A simples inspeção da tabela de transição de estados (*state table*) pode revelar estados redundantes, os quais podem ser imediatamente unificados em um estado equivalente. Em geral, esse método não conduz a um conjunto mínimo de estados, funcionando apenas como um pré-processamento para os demais métodos de minimização. Nesse caso, a condição de redundância é dada por: estados (q^n) que, para cada entrada simples (x^n), conduzem aos mesmos próximos estados e às mesmas saídas (q^{n+1}, z^n), representam um único estado equivalente.

O algoritmo de eliminação de estados redundantes por simples inspeção é dado por:

EI1 - Verificar a existência de redundância.

EI2 - Se não houver redundância, ir ao passo EI6.

EI3 - Se houver redundância, escolher um dos estados redundantes como estado equivalente, mantendo-o na tabela e eliminando todos os demais estados redundantes.

EI4 - Atualizar a tabela, trocando a designação dos estados eliminados por aquela do estado escolhido como equivalente.

EI5 - Voltar ao passo EI1.

EI6 - Fim.

A Figura 13.14 apresenta um exemplo de eliminação de estados redundantes por simples inspeção.

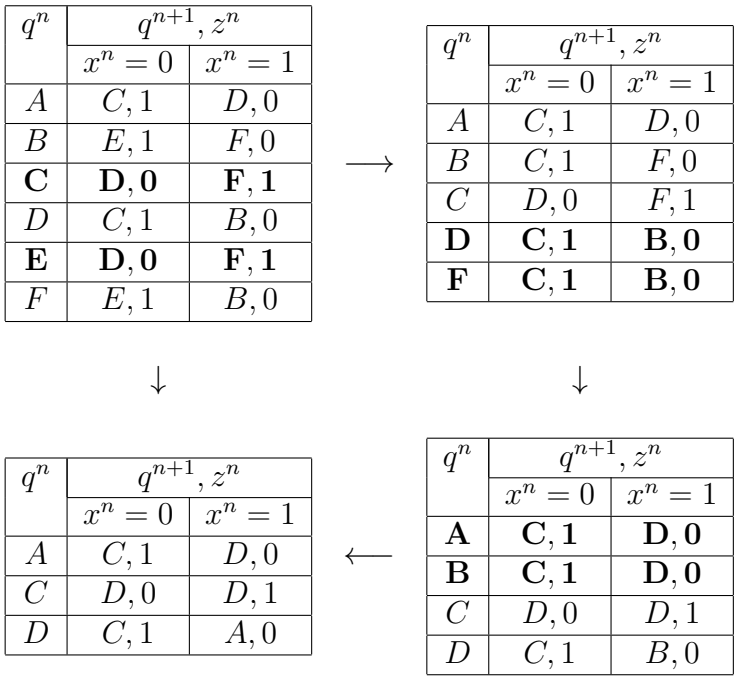


Figura 13.14: Eliminação de estados redundantes através da inspeção da tabela de estados.

13.8.3 Método da partição em classes de estados indistinguíveis (método de Huffman-Mealy)

- O processo é simples, mas não pode ser aplicado para os casos de tabelas de estados não completamente especificadas.
- Ele é baseado no **Teorema 1**, discutido no Apêndice D e apresentado a seguir.
- **Teorema 1**: Suponha-se que os estados de um circuito seqüencial foram particionados em classes disjuntas, onde $p \triangleq q$ denota que os estados p e q pertencem à mesma classe. A partição é composta por classes de equivalência de estados indistinguíveis se e somente se as duas condições seguintes forem satisfeitas por cada par de estados p e q da mesma classe, para cada entrada simples x^n:
  1. $\lambda(p^n, x^n) = \lambda(q^n, x^n)$.
  2. $\delta(p^n, x^n) \triangleq \delta(q^n, x^n)$.
- Conforme definido no Apêndice D, as funções $\lambda(q^n, x^n) = z^n$ e $\delta(q^n, x^n) = q^{n+1}$, representam, respectivamente, a saída atual e o próximo estado.
- Basicamente, o método pode ser dividido em duas partes:
  - Aplicação da condição (1) do **Teorema 1**.
  - Aplicações sucessivas da condição (2) do **Teorema 1**.
- Algoritmo de minimização por partição em classes de estados indistinguíveis:

  HM0 - Tentar eliminar estados redundantes por simples inspeção da tabela de estados original. Se houver alguma eliminação, a tabela de estados reduzida passa a representar a tabela de estados original para o restante do algoritmo. Este passo não é necessário, mas diminui o espaço de busca do algoritmo.

  HM1 - A partir da tabela de estados original, separar, em classes distintas ($C_{z_i} \in C_z$), os estados (e_j) que possuem os mesmos conjuntos de saídas (z_{i_k}), para cada valor da entrada (x_k).

  HM2 - Se houver apenas um estado por classe, ir para o passo HM7.

  HM3 - Se houver pelo menos uma classe atual com mais de um estado, descobrir as classes referentes aos próximos estados de cada estado atual, as quais serão denominadas de próximas classes.

  HM4 - Para cada classe com mais de um estado, verificar as próximas classes, para cada valor da entrada (x).

  HM5 - Se, dentro de uma mesma classe, houver estados com próximas classes diferentes dos demais, separá-los em uma nova classe e retornar para o passo HM2.

  HM6 - Se, dentro de cada classe, não houver estado com próximas classes diferentes dos demais, ir para o passo HM7.

  HM7 - Fim.

- As Figuras 13.15, 13.16 e 13.17 ilustram o processo para diferentes tabelas de estado.

q^n	q^{n+1}, z^n	
	$x^n = 0$	$x^n = 1$
A	$C, 1$	$D, 0$
B	$E, 1$	$F, 0$
C	$D, 0$	$F, 1$
D	$C, 1$	$B, 0$
E	$D, 0$	$F, 1$
F	$E, 1$	$B, 0$

$\Longleftrightarrow$

q^n	q^{n+1}, z^n	
	$x^n = 0$	$x^n = 1$
q_1	$q_2, 0$	$q_2, 1$
q_2	$q_1, 1$	$q_2, 0$

$\downarrow$ $\uparrow$

Classe(n)	1				2							
Estado(n)	C		E		A		B		D		F	
Classe(n+1)	2	2	2	2	1	2	1	2	1	2	1	2

Figura 13.15: Exemplo de minimização positiva em um passo.

q^n	q^{n+1}, z^n	
	$x^n = 0$	$x^n = 1$
A	$B, 0$	$B, 1$
B	$F, 1$	$D, 1$
C	$E, 1$	$G, 1$
D	$A, 0$	$C, 0$
E	$D, 1$	$G, 0$
F	$F, 0$	$A, 0$
G	$C, 1$	$B, 0$

$\downarrow\uparrow$

Classe(n)	0				1		2				3			
Estado(n)	D		F		A		E		G		B		C	
Classe(n+1)	1	3	0	1	3	3	0	2	3	3	0	0	2	2

Figura 13.16: Exemplo de minimização negativa em um passo.

q^n	q^{n+1}, z^n	
	$x^n = 0$	$x^n = 1$
A	$C, 0$	$D, 1$
B	$B, 0$	$E, 1$
C	$C, 0$	$E, 1$
D	$E, 1$	$H, 0$
E	$D, 1$	$F, 0$
F	$I, 0$	$C, 1$
G	$J, 0$	$B, 1$
H	$I, 0$	$A, 1$
I	$G, 1$	$E, 0$
J	$H, 1$	$D, 0$

$\Longleftrightarrow$

q^n	q^{n+1}, z^n	
	$x^n = 0$	$x^n = 1$
q_0	$q_0, 0$	$q_2, 1$
q_1	$q_3, 0$	$q_1, 1$
q_2	$q_2, 1$	$q_1, 0$
q_3	$q_1, 1$	$q_2, 0$

$\downarrow$

Classe(n)	0												1							
Estado(n)	A		B		C		F		G		H		D		E		I		J	
Classe(n+1)	0	1	0	1	0	1	1	0	1	0	1	0	1	0	1	0	0	1	0	1

$\downarrow$ $\circlearrowleft$

Classe(n)	0						1						2				3			
Estado(n)	A		B		C		F		G		H		D		E		I		J	
Classe(n+1)	0	2	0	2	0	2	3	0	3	0	3	0	2	1	2	1	1	2	1	2

Figura 13.17: Exemplo de minimização positiva em mais de um passo.

13.8.4 Método da tabela de implicação de estados (método de Paul-Unger)

- Processo mais complexo do que o apresentado pelo método da partição em classes.
- Porém, ele é mais genérico, podendo ser aplicado para os casos de tabelas de estados não completamente especificadas.
- **Definição 1**: Um conjunto de estados P é implicado por um conjunto de estados R se, para alguma entrada específica x_k, P é o conjunto de todos os próximos estados $p_i^{n+1} = \delta(r_j^n, x_k^n)$, para todos os estados atuais $r_j \in R$.
- A partir do **Teorema 1** e da **Definição 1**, pode-se dizer que os estados de um conjunto R são equivalentes apenas se todos os estados de um conjunto P, implicado por R, também são equivalentes.
- Para que os estados de um conjunto R sejam equivalentes, todos os pares $(r_i, r_j) \in R$ devem ser equivalentes.
- Logo, para verificar a equivalência dos estados de um conjunto, basta testar a implicação para cada par de estados do conjunto.
- Uma forma de realizar esse teste é montar uma árvore de implicação.
- A partir de um determinado par $(r_i, r_j) \in R$, são determinados os estados implicados para cada entrada. Partindo de cada novo conjunto implicado, a operação é repetida. Se algum conjunto implicado da árvore de (r_i, r_j) não for equivalente, o par inicial (r_i, r_j) não pode ser equivalente.
- Tal processo de investigação, que caracteriza uma prova por absurdo ou contradição, possui uma complexidade muito elevada.
- Uma forma mais eficiente de verificar a equivalência de estados é através de uma prova por negação.
- Nesse caso, os estados são organizados em uma tabela de implicação, onde todas as combinações de pares de estados encontram-se representadas. Para cada par, são determinados os estados implicados, para cada entrada. Em seguida, todos as implicações proibidas são eliminadas da tabela. O processo de proibição é repetido até que nenhuma proibição seja encontrada. Por fim, são listadas as classes de equivalência.
- As proibições iniciais são provenientes de pares de estados que apresentam saídas diferentes para as mesmas entradas.
- Uma tabela de implicação e uma de suas células são apresentadas, respectivamente, nas Figuras 13.18 e 13.19.

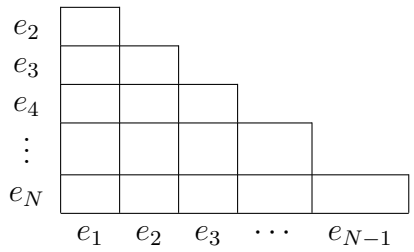


Figura 13.18: Tabela de implicação genérica do método de Paul-Unger.

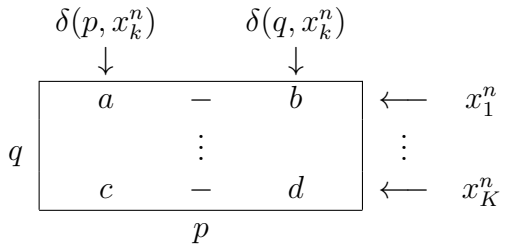


Figura 13.19: Célula genérica da tabela do método de Paul-Unger.

- Algoritmo de minimização por tabela de implicação de estados:

  PU0 - Tentar eliminar estados redundantes por simples inspeção da tabela de estados original. Se houver alguma eliminação, a tabela de estados reduzida passa a representar a tabela de estados original para o restante do algoritmo. Este passo não é necessário, mas diminui o espaço de busca do algoritmo.

  PU1 - A partir da tabela de estados original, separar, em classes distintas ($C_{z_i} \in C_z$), os estados (e_j) que possuem os mesmos conjuntos de saídas (z_{i_k}), para cada valor da entrada (x_k).

  PU2 - Se houver apenas um estado por classe, ir para o passo PU8.

  PU3 - Montar uma matriz triangular inferior, contendo índices horizontais $h_i \equiv e_i \in \{C_z - e_N\}$ e índices verticais $v_j \equiv e_j \in \{C_z - e_1\}$.

  PU4 - Anular todas as posições da matriz, referentes às combinações $h_i \times v_j$, onde $C_z(e_i) \neq C_z(e_j)$.

  PU5 - Preencher todas as posições da matriz, referentes às combinações $h_i^n \times v_j^n$, onde $C_z(e_i^n) = C_z(e_j^n)$, com os pares $(h_i^{n+1} \text{ — } v_j^{n+1})_k$, se $h_i^{n+1} \neq v_j^{n+1}$, para cada valor da entrada (x_k).

  PU6 - Repetir, até que não haja mais anulações, para cada posição não anulada $h_i^n \times v_j^n$ da matriz:

  PU61 - Verificar se os pares $(h_i^{n+1} \text{ — } v_j^{n+1})_k$ foram anulados.

  PU62 - Se, pelo menos um dos pares tiver sido anulado, anular a posição corrente $h_i^n \times v_j^n$ da matriz e notificar a ocorrência de anulação.

  PU7 - Organizar em classes de equivalência os estados cujas combinações $h_i^n \times v_j^n$ não foram anuladas e em classes individuais os demais estados.

  PU8 - Fim.

13.9 Atribuição de estados

13.9.1 Considerações iniciais

- No projeto de um circuito digital seqüencial, a atribuição de estados tem influência direta na síntese da lógica combinacional que gera as variáveis de excitação e as variáveis de saída.
- A prática demonstra que atribuições de estados diferentes podem produzir lógicas combinacionais diferentes.
- Portanto, a fim de se obter o circuito combinacional de menor custo, deve-se procurar a atribuição de estados que favoreça a sua síntese.
- Uma vez que os estados são representados por um conjunto de V variáveis booleanas, duas situações podem ocorrer. Na primeira, o número de estados (S) que se deseja representar é igual ao número de estados representáveis, de forma que $S = 2^V$. Caso contrário, o número de estados a serem representados encontra-se na seguinte faixa: $2^{V-1} < S < 2^V$.
- Quando $S = 2^V$, o problema de atribuição de estados se resume a estabelecer uma relação de equivalência dos estados desejados com as configurações existentes para as variáveis de estado.
- No caso de $2^{V-1} < S < 2^V$, além da equivalência, é necessário também escolher S configurações a serem utilizadas dentre as 2^V existentes.
- Para um número de estados na faixa $2^{V-1} < S \leq 2^V$, pode-se demonstrar que o número total de atribuições (A_{tot}) pode ser calculado por $A_{tot} = \frac{2^V!}{(2^V-S)!}$.
- Porém, muitas dessas atribuições são redundantes, pois representam apenas trocas e/ou complementações lógicas das variáveis de estado.
- Assim, pode-se demonstrar que o número de atribuições efetivamente diferentes (A_{dif}) pode ser calculado por $A_{dif} = \frac{(2^V-1)!}{(2^V-S)!\ V!}$.
- A Tabela 13.21 ilustra algumas possibilidades.

Estados (S)	Variáveis de Estado (V)	Atribuições (A_{dif})
2	1	1
3	2	3
4	2	3
5	3	140
6	3	420
7	3	840
8	3	840
9	4	10.810.800

Tabela 13.21: Número de atribuições de estados efetivamente diferentes.

- Assim sendo, para $S = 3$ ou 4, podem ser realizados 3 projetos, a partir das 3 atribuições possíveis, escolhendo-se o de menor custo.

- Para $S \geq 5$, pode-se visualizar duas soluções:
  - Aplicar um algoritmo que encontre a atribuição de menor custo.
  - Aplicar regras que indiquem um conjunto reduzido de atribuições de menor custo, projetar cada uma delas e realizar a escolha.
- Na literatura relativa ao assunto, podem ser encontradas várias propostas de técnicas a serem aplicadas no processo de atribuição de estados.
- Infelizmente, nenhuma delas apresenta um algoritmo de busca da melhor atribuição.
- Na realidade, são apresentadas regras genéricas, cujo emprego conduz a um conjunto reduzido de atribuições de menor custo.
- Portanto, enfatizando, a função das regras propostas é a de reduzir o número total de atribuições para uma quantidade mínima de atribuições que mereçam ser analisadas.
- De posse de um conjunto reduzido de candidatas a uma atribuição de menor custo, o projetista pode testar as alternativas e realizar a escolha.
- Vale ressaltar, ainda, que a aplicação das regras não garante que a melhor atribuição seja encontrada.
- Dependendo da especificação do circuito seqüencial e do tipo de elemento de memória utilizado, as regras podem apontar para uma solução que não é a de menor custo, porém é bem próxima.

13.9.2 Base teórica para as regras de atribuição de estados

- A atribuição de estados de menor custo é aquela que sintetiza as variáveis de excitação e as variáveis de saída através da menor quantidade de circuito combinacional.
- A redução da quantidade de circuito combinacional empregada é associada à simplificação da equação lógica que o representa.
- Por sua vez, a minimização de uma equação lógica é conseguida através da combinação de mintermos ou maxtermos que possuam adjacência lógica.
- Por adjacência lógica entende-se a situação onde dois mintermos (ou maxtermos) diferem pelo valor de apenas um de seus *bit*.
- No mapa de Karnaugh simbólico da Figura 13.20, os conjuntos de variáveis $\{x_1, x_0\}$ e $\{y_1, y_0\}$ representam, respectivamente, as variáveis de entrada e as variáveis de estado.
- Utilizando-se o mapa na síntese das variáveis de excitação (Y) e das variáveis de saída (z), destacam-se três situações distintas.
- Supondo-se que o mapa se refere à síntese de variáveis de excitação, ocorrerem dois casos que envolvem uma dinâmica de mudança de estados. O primeiro deles é relacionado com a possibilidade de simplificação de valores em linha (v_r). Ele trata da mudança de dois estados atuais para dois próximos estados, para um mesmo valor de entrada. O outro caso é relacionado com a possibilidade de simplificação de valores em coluna (v_c). Ele trata da mudança de um estado atual para dois próximos estados, para dois valores de entrada diferentes.

- Por outro lado, se o mapa se refere à síntese das variáveis de saída, ocorre a terceira alternativa, estática. Nesse caso, os valores atuais ("0" ou "1") da saída podem promover simplificações em linha (v_r) e/ou em colunas (v_c).

- Com base na análise de cada situação, pode-se definir um conjunto de regras básicas que indique uma atribuição de estados adequada.

- Ainda que tais regras não conduzam à maior simplificação possível, elas ajudam a escolher uma solução próxima da ótima.

x_1x_0 \ y_1y_0	00	01	11	10
00	v_r	v_r		
01				
11			v_c	
10			v_c	

Figura 13.20: Análise de minimização para as equações de excitação e de saída: mapa de Karnaugh simbólico.

Análise para a síntese de variáveis de excitação

- Do mapa de Karnaugh da Figura 13.20, destacam-se duas situações dinâmicas distintas.

- Uma simplificação de linha (v_r) envolve a dinâmica de dois estados atuais para dois próximos estados, considerando uma mesma entrada.

- Por sua vez, uma simplificação de coluna (v_c) envolve a dinâmica de um estado atual para dois próximos estados, considerando duas entradas diferentes.

- Na simplificação de linha (v_r), podem ser identificados alguns subcasos, de acordo com os próximos estados: i) todos iguais para as mesmas entradas, ii) todos iguais para entradas diferentes, iii) alguns iguais para as mesmas entradas, iv) alguns iguais para entradas diferentes, e v) todos diferentes. Tais subcasos não serão analisados, sendo deixados como proposta de exercício.

- A Figura 13.21 apresenta uma tabela de atribuição de estados hipotética. Nesse caso, os estados logicamente adjacentes são: (a, b), (a, c), (b, d) e (c, d).

- As Figuras 13.22 e 13.23 ilustram a análise de minimização para as variáveis de excitação.

- A Figura 13.22 mostra que, se dois estados atuais possuem o mesmo próximo estado e não são logicamente adjacentes, as suas excitações para os elementos de memória (E_{ij}) não poderão ser combinadas. Caso contrário, elas se combinarão com certeza, minimizando a expressão lógica, a menos de uma das variáveis de estado, para a qual não há garantia.

- Assim, desconsiderando-se os subcasos, a recomendação é: "Dois estados que possuam o mesmo próximo estado devem ser logicamente adjacentes!".

- A Figura 13.23 mostra que, se um estado atual possui dois próximos estados que não possuem adjacência lógica, nada garante que as excitações dos elementos de memória (E_{ij} e E_{kl}) serão as mesmas e, portanto, nada garante que elas serão agrupadas para minimizar a expressão lógica. Caso contrário, a minimização é possível com certeza, a menos de uma das variáveis de estado, para a qual não há garantia.
- Nesse caso, a recomendação é: "Dois estados que sejam próximos estados de um mesmo estado devem ser logicamente adjacentes!".
- Uma vez que as duas recomendações envolvem, respectivamente, um impedimento e uma possibilidade, a primeira delas delas deve ser prioritária em relação à segunda.

Estados q	Variáveis de Estado y_1y_0
a	00
b	01
d	11
c	10

Figura 13.21: Análise de minimização para as equações de excitação e de saída: tabela de atribuição de estados hipotética.

q^n	$y_1^n y_0^n$	q^{n+1}		$y_1^{n+1} y_0^{n+1}$	
		$x^n = 0$	$x^n = 1$	$x^n = 0$	$x^n = 1$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
b	01	a	$\cdots$	00	$\cdots$
c	10	a	$\cdots$	00	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Tabela de transição de estados

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0		E_{00}		E_{10}
	1				

Dinâmica da variável de estado y_1

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0		E_{10}		E_{00}
	1				

Dinâmica da variável de estado y_0

a) Caso sem simplificação.

q^n	$y_1^n y_0^n$	q^{n+1}		$y_1^{n+1} y_0^{n+1}$	
		$x^n = 0$	$x^n = 1$	$x^n = 0$	$x^n = 1$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
b	01	a	$\cdots$	00	$\cdots$
d	11	a	$\cdots$	00	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Tabela de transição de estados

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0		E_{00}	E_{10}	
	1				

Dinâmica da variável de estado y_1

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0		E_{10}	E_{10}	
	1				

Dinâmica da variável de estado y_0

b) Caso com simplificação.

Figura 13.22: Análise de minimização para as equações de excitação: casos de estados atuais com mesmo próximo estado.

q^n	$y_1^n y_0^n$	q^{n+1}		$y_1^{n+1} y_0^{n+1}$	
		$x^n = 0$	$x^n = 1$	$x^n = 0$	$x^n = 1$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
a	00	b	c	01	10
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Tabela de transição de estados

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0	E_{00}			
	1	E_{01}			

Dinâmica da variável de estado y_1

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0	E_{01}			
	1	E_{00}			

Dinâmica da variável de estado y_0

a) Caso sem simplificação.

q^n	$y_1^n y_0^n$	q^{n+1}		$y_1^{n+1} y_0^{n+1}$	
		$x^n = 0$	$x^n = 1$	$x^n = 0$	$x^n = 1$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
a	00	b	d	01	11
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Tabela de transição de estados

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0	E_{00}			
	1	E_{01}			

Dinâmica da variável de estado y_1

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0	E_{01}			
	1	E_{01}			

Dinâmica da variável de estado y_0

b) Caso com simplificação.

Figura 13.23: Análise de minimização para as equações de excitação: casos de estado atual com próximos estados diferentes.

Análise para a síntese de variáveis de saída

- Do mapa de Karnaugh da Figura 13.20, destaca-se uma situação estática, não envolvendo mudanças de estado.
- Com base na atribuição de estados apresentada na Figura 13.21, a Figura 13.24 ilustra a análise de minimização para as variáveis de saída. Se dois estados atuais possuem a mesma saída, para a mesma entrada, e não são logicamente adjacentes, os valores de saída não poderão ser combinados. Caso contrário, os valores de saída serão combinados com certeza, minimizando a expressão lógica.
- Portanto, a recomendação é: "Dois estados atuais que possuam a mesma saída, para a mesma entrada, devem ser logicamente adjacentes!".
- Normalmente, o número de variáveis de saída é menor que o número de variáveis de excitação. Assim sendo, tal recomendação terá a menor prioridade.

q^n	$y_1^n y_0^n$	z_i^n	
		$x^n = 0$	$x^n = 1$
$\cdots$	$\cdots$	$\cdots$	$\cdots$
b	01	1	0
c	10	1	0
$\cdots$	$\cdots$	$\cdots$	$\cdots$

Tabela de transição de estados

x^n \ $y_1^n y_0^n$	00	01	11	10
0		1		1
1		0		0

z_i

Mapa-K da saída z_i

a) Caso sem simplificação.

q^n	$y_1^n y_0^n$	z_i^n	
		$x^n = 0$	$x^n = 1$
$\cdots$	$\cdots$	$\cdots$	$\cdots$
b	01	1	0
d	11	1	0
$\cdots$	$\cdots$	$\cdots$	$\cdots$

Tabela de transição de estados

x^n \ $y_1^n y_0^n$	00	01	11	10
0		1	1	
1		0	0	

z_i

Mapa-K da saída z_i

b) Caso com simplificação.

Figura 13.24: Análise de minimização para as equações de saída.

13.9.3 Exemplo de regras simples (Armstrong-Humphrey)

- No projeto de circuitos seqüenciais que possuam um número pequeno de estados, podem ser utilizadas duas regras básicas no processo de atribuição de estados [Arm62], [Hum58].
- Tais regras são originadas na tentativa de minimização da lógica responsável pela geração das variáveis de excitação.
- A principal motivação para o emprego destas regras é que elas são de curta descrição, de fácil compreensão, de simples aplicação e conduzem a bons resultados.
- Regras:
  - **Regra 1**: Dois ou mais estados que possuam o mesmo próximo estado devem ser logicamente adjacentes.
  - **Regra 2**: Dois ou mais estados que sejam próximos estados de um mesmo estado devem ser logicamente adjacentes.
- É importante ressaltar que as regras são listadas em ordem decrescente de prioridade.
- A Figura 13.25 ilustra as regras descritas acima.

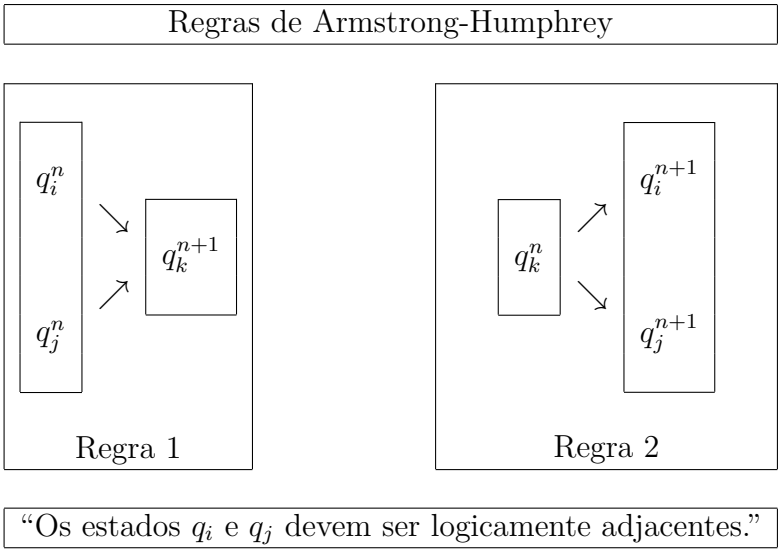


Figura 13.25: Ilustração das regras de Armstrong-Humphrey.

13.9.4 Exemplo de regras mais refinadas

- Um conjunto de regras mais completo pode ser obtido: i) ao se detalhar a Regra 1, anteriormente apresentada, e ii) ao se incorporar a tentativa de minimização da lógica responsável pela geração das variáveis de saída.
- É importante ressaltar que as regras são listadas em ordem decrescente de prioridade.
- Regras:
  - **Regras 1**:
    - **Regra 1a**: Os estados que possuam todos os próximos estados iguais, coluna a coluna, devem ser logicamente adjacentes. Se possível, os próximos estados também devem ser logicamente adjacentes, de acordo com a **Regra 2**.
    - **Regra 1b**: Os estados que possuam todos os próximos estados iguais, mas em colunas diferentes, devem ser logicamente adjacentes se os próximos estados também puderem ser logicamente adjacentes.
    - **Regra 1c**: Os estados que possuam alguns do próximos estados iguais devem ser logicamente adjacentes. A prioridade de adjacência será maior para os estados que apresentarem um maior número de próximos estados iguais.
  - **Regra 2**: Os próximos estados provenientes de um mesmo estado atual devem ser logicamente adjacentes.
  - **Regra 3**: As atribuições devem ser feitas de forma a simplificar os mapas das variáveis de saída. Assim sendo, os estados que possuam as mesmas saídas, para as mesmas entradas, devem ser logicamente adjacentes.

13.10 Efeitos causados por estados extras

13.10.1 Definição do problema

- No projeto de circuitos seqüenciais, é comum ocorrer a situação onde o número total de estados que pode ser implementado pelo circuito é maior do que o número total de estados que constam na sua especificação.
- Uma vez que, teoricamente, não haverá transições dos estados principais para os estados extras, os valores de próximo estado e de saída para os estados extras podem ser assumidos como não especificados (*don't care* ou com valor lógico "X").
- Tal decisão de projeto acarreta duas conseqüências imediatas. Por um lado, evita-se empregar uma quantidade extra de circuito lógico combinacional, responsável pelo correto funcionamento a partir dos estados extras. Além disso, os valores lógicos "X" podem acarretar simplificações no projeto do circuito lógico principal, durante a síntese das variáveis de excitação.
- Na prática, porém, algum mal funcionamento do circuito pode colocá-lo em um dos estados extras.
- Por essa razão, deve-se realizar uma análise do circuito projetado, de modo a verificar o comportamento de tais estados.

- As seguintes situações podem ocorrer nos circuitos cujo projeto contém estados não especificados:
  - No caso particular do uso de *flip-flops* do tipo SR, pode acontecer alguma indeterminação no circuito seqüencial devido a indeterminações nos *flip-flops* ($S = R = 1$).
  - As saídas do circuito podem apresentar valores não esperados e/ou não especificados.
  - Podem surgir estados extras isolados (*dead states*) ou ciclos isolados de estados extras (*dead cycles*), totalmente independentes dos estados relativos à operação normal do circuito seqüencial projetado.
  - Todos os estados extras podem formar seqüências de estados que convergem para os estados relativos à operação normal do circuito seqüencial projetado. O diagrama de estado de tais circuitos é chamado de arbusto (*bush*), sendo o conjunto dos estados normais de operação denominado de tronco (*trunk*) e as seqüências de estados extras de ramos (*branches*). Nesses casos, o circuito seqüencial é dito auto-corretivo (*self-correcting*).

13.10.2 Possíveis soluções

- As soluções para o retorno do circuito aos seus estados principais, a partir de algum estado extra, podem envolver dois tipos de ações.
- Adotando-se uma correção ativa, pode-se empregar circuitos lógicos adicionais, com a função de auxiliar na detecção dos estados extras e na atuação sobre o circuito.
- Em um tipo de correção passiva, pode-se projetar o circuito de tal forma que seu Diagrama de Estados final seja um arbusto.
- Ações ativas (após o projeto):
  - Detecção de erro: que exige um circuito adicional para identificação de um estado extra.
  - Sinalização de erro: que pode ser implementada através de um sinal extra de saída (*flag* de erro) ou de um valor de saída não especificado.
  - Interrupção do sinal de *clock*: que necessita de um circuito extra para mascaramento do sinal de *clock* original.
  - Correção ativa: que executa o retorno a um dos estados principais através de um sinal de RESET.
- Ações passivas (durante o projeto):
  - Verificar os mapas-K de excitação, para evitar que ocorram indeterminações (valores $S = R = 1$) em *flip-flops* do tipo SR.
  - Verificar os mapas-K de excitação, para evitar que ocorram *dead states* e/ou *dead cycles*.
  - Verificar os mapas-K de saída, para garantir consistência nos valores das mesmas.
- Deve ser ressaltado que, em projetos onde a operação correta é fundamental, ambos os tipos de ações devem ser empregados.

Capítulo 14

Circuitos seqüenciais *pulsed*

14.1 Introdução

- A Figura 14.1 apresenta um modelo genérico para circuitos seqüenciais *pulsed*.

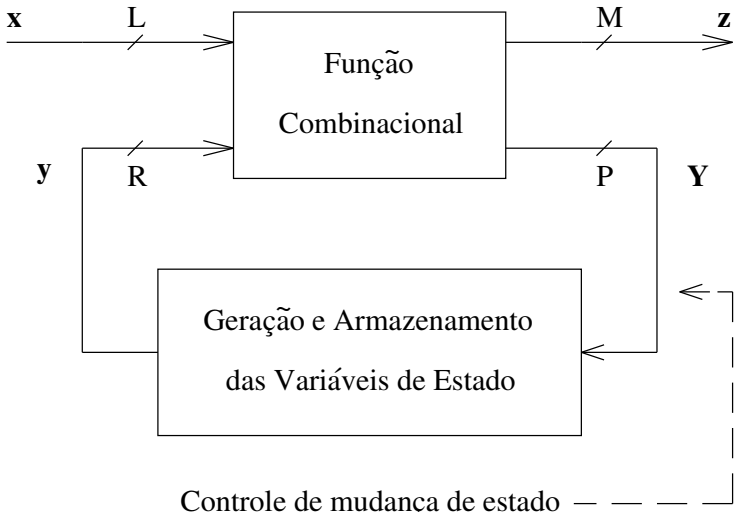


Figura 14.1: Modelo genérico para circuitos seqüenciais *pulsed*.

- O modelo destaca a ausência de um sinal especial de relógio ou *clock*, que atue diretamente sobre o circuito de memória, destinado puramente ao sincronismo.

- Uma mudança de estado é provocada pela ocorrência de um pulso em um dos sinais de entrada.

- Qualquer um dos sinais de entrada pode ser do tipo pulso.

- Os sinais de entrada x_i podem ser tanto do tipo nível quanto do tipo pulso. Porém, é obrigatório que pelo menos um deles seja do tipo pulso.

- No caso de circuitos seqüenciais do tipo Mealy, por definição, as saídas poderão ser do tipo nível e/ou do tipo pulso, uma vez que poderão ser provenientes das combinações dos níveis das variáveis de estado com os níveis e os pulsos dos sinais de entrada. Porém, o mais comum é que as saídas sejam todas do tipo pulso.

- Para os circuitos seqüenciais do tipo Moore, as saídas deverão ser do tipo nível e deverão permanecer estáveis durante o intervalo de tempo entre dois pulsos de entrada consecutivos.

- Os elementos de memória podem ser dos tipos *unclocked* ou *clocked*.

- Em relação às combinações de sinais dos tipos nível e pulso, vale destacar os seguintes aspectos:

  - Uma vez que combinações lógicas AND e OR entre pulsos positivos e negativos produzem resultados indeterminados, apenas um tipo de pulso (positivo ou negativo) deve ser usado.

  - Após a escolha do tipo de pulso a ser utilizado (positivo ou negativo), ainda deve ser lembrado que algumas interações entre sinais dos tipos nível e pulso geram resultados indeterminados para as operações lógicas AND e OR.

  - Portanto, considerando-se que os sinais x_l, x_p e $\overline{x}_p$ representam, respectivamente, sinais dos tipos nível, pulso positivo e pulso negativo, as seguintes combinações podem ser empregadas:

$$* \begin{cases} x_l \cdot x_l = x_l \quad , \quad x_l + x_l = x_l \quad , \quad x_l \cdot x_p = x_p \quad , \quad x_p + x_p = x_p \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{ou} \\ x_l \cdot x_l = x_l \quad , \quad x_l + x_l = x_l \quad , \quad x_l + \overline{x}_p = \overline{x}_p \quad , \quad \overline{x}_p \cdot \overline{x}_p = \overline{x}_p \; . \end{cases}$$

14.2 Restrições de operação

- Os circuitos seqüenciais *pulsed* apresentam as seguintes restrições para o seu correto funcionamento:

  - Deve ser garantido que os elementos de memória operem de tal forma que ocorra apenas uma mudança de estado para cada pulso de entrada.

  - Todos os pulsos de entrada devem apresentar uma duração (largura de pulso) suficiente para o correto acionamento dos elementos de memória.

  - As bordas de disparo dos pulsos de entrada consecutivos, em um mesmo sinal de entrada ou em sinais de entrada diferentes, devem ser espaçadas de um intervalo maior que o tempo de mudança de estado dos elementos de memória.

    * Como conseqüência desta restrição, é vetada a ocorrência de pulsos simultâneos em sinais de entrada diferentes.

  - As entradas do tipo nível devem estar estáveis quando ocorrer um pulso em qualquer das entradas do tipo pulso.

14.3 Classificação quanto aos pulsos de entrada

- Três classes de circuitos seqüenciais do tipo *pulsed* podem ser destacadas: *controlled-clock*, *pulse-mode* e *ripple-clock*.
- Circuitos do tipo *controlled-clock* são casos particulares, sujeitos a mais restrições. Ainda assim, tal abordagem permite o projeto de sistemas digitais mais complexos do que aqueles pertencentes à classe de circuitos *clock-mode*.
- Circuitos do tipo *pulse-mode* representam uma classe mais geral dentro dos circuitos do tipo *pulsed*. Eles podem ser empregados nos casos onde as restrições de sincronismo dos circuitos *clock-mode* e *controlled-clock* não possam ser cumpridas.
- Circuitos do tipo *ripple-clock* resultam de um tentativa de otimização que pode levar à redução da quantidade de *hardware* em detrimento da freqüência máxima de operação.
- Circuitos do tipo *controlled-clock*:
  - Os elementos de memória são do tipo *clocked*.
  - Existe somente uma entrada pulsada, sendo esta periódica e denominada de *clock*.
  - O sinal de *clock* não é aplicado diretamente nas entradas de controle dos elementos de memória. Ele é combinado com os outros sinais de entrada e/ou com as variáveis de estado para gerar fontes secundárias de sinais pulsados, sincronizados com o sinal de *clock*.
- Circuitos do tipo *pulse-mode*:
  - Os elementos de memória podem ser dos tipos *unclocked* ou *clocked*.
  - Normalmente, existe mais de uma entrada pulsada.
  - Podem ser destacados dois casos: i) coexistência de entradas dos tipos nível e pulso e ii) existência apenas de entradas do tipo pulso.
  - Geralmente, os diversos sinais de entrada pulsantes são aperiódicos e temporalmente descorrelacionados.
- Circuitos do tipo *ripple-clock*:
  - Existe, pelo menos, uma entrada pulsada.
  - Existe, pelo menos, um elemento de memória ativado pelos pulsos de entrada. Em seguida, as saídas desse elemento servem de sinal de ativação para outros elementos de memória, e assim consecutivamente, até que todos os elementos de memória tenham sido ativados.
  - As entradas pulsadas podem ser periódicas ou não.
  - O intervalo de tempo entre pulsos de entrada consecutivos deve levar em conta o tempo de propagação de disparos sucessivos dos elementos de memória. Isso pode ser controlado através do pior caso ou através de sinais de término de disparos.
  - Geralmente, o circuito apresenta estados intermediários não estáveis (transitórios). Se necessário for, as saídas devem ser controladas pelos pulsos de entrada, a fim de que apresentem apenas os resultados estáveis.

14.4 Circuitos *pulse-mode*

14.4.1 Motivação

- Existem situações onde as restrições de sincronismo para os circuitos *clock-mode* e *controlled-clock* não podem ser atendidas.
- Uma situação típica é a interface entre subsistemas projetados independentemente uns dos outros.
- Outra situação típica é a interconexão de subsistemas implementados com famílias lógicas diferentes, onde a diferença de taxa de operação é significativa.
- Por exemplo, um sinal de saída do tipo nível em um subsistema com taxas elevadas de chaveamento pode ser interpretado com um pulso de entrada em um subsistema mais lento.
- Utilizando a técnica de projeto *pulse-mode*, o projetista ganha liberdade para designar quais sinais serão interpretados como sendo do tipo nível ou do tipo pulso.

14.4.2 Mudanças nas representações

- O diagrama de estados, a tabela de transição de estados (*state table*) e o mapa-K usados na síntese de circuitos *pulse-mode* apresentam algumas mudanças em relação àqueles que são empregados em circuitos *clock-mode*.
- Tanto a sintaxe quanto a semântica de tais representações sofrem modificações.
- Diversas sintaxes, bem como seus significados, podem ser propostas.
- A sintaxe e a semântica utlizadas no presente texto são detalhadas a seguir.
- Assim como nos circuitos *clock-mode*, os valores $x_l = 0$ e $x_l = 1$, de uma entrada do tipo nível, e os valores $z_l = 0$ e $z_l = 1$, de uma saída do tipo nível, representam os níveis lógicos que tais sinais podem assumir.
- No diagrama de estados, a ausência ou a presença de um pulso (positivo ou negativo) em um sinal de entrada pulsante x_p é representada, respectivamente, pela ausência ou pela presença da variável x_p (pulso positivo) ou de sua negação lógica $\overline{x}_p$ (pulso negativo).
- No diagrama de estados, a ausência ou a presença de um pulso (positivo ou negativo) em um sinal de saída pulsante z_p é representada, respectivamente, pelo valor lógico "0" ou pela presença da variável z_p (pulso positivo) ou de sua negação lógica $\overline{z}_p$ (pulso negativo).
- Na tabela de estados, os valores $z_p = \overline{z}_p = 0$ e $z_p = \overline{z}_p = 1$, de saídas pulsantes z_p (pulso positivo) e $\overline{z}_p$ (pulso negativo), representam, respectivamente, a ausência e a presença de um pulso em z_p e $\overline{z}_p$.
- Entradas não especificadas nas transições do diagrama de estados, bem como as saídas nesses casos, são representadas na tabela de estados como *don't care* ("X").
- No diagrama de estados, a especificação conjunta de duas ou mais variáveis de entrada do tipo pulso, $(x_{p1},x_{p2},\cdots)$, indica apenas que a ocorrência de um pulso em qualquer dos sinais x_{pi} acarretará uma mudança de estado. Afinal, deve ser lembrado que, devido às restrições de operação, é proibida a ocorrência de pulsos simultâneos.

- Como conseqüência das possibilidades de combinação entre sinais do tipo nível e sinais do tipo pulso, as variáveis de saída e as variáveis de excitação devem ser geradas por SOP envolvendo pulsos positivos ou por POS envolvendo pulsos negativos.

- A sintaxe e a semântica do mapa-K, usado na síntese das funções combinacionais, vão depender do tipo de elemento de memória utilizado.

- Na síntese das variáveis pulsadas (excitação ou saída), é comum que se utilize os valores "0" e "1" para representar, respectivamente, a ausência ou a presença de pulsos. Esse tipo de representação é mais adequado para um tratamento por computador. Para uso humano, pode ser de grande auxílio utilizar um sinal indicativo de pulso ("Π"), conforme ilustrado na Figura 14.2.

- Vale a pena ressaltar que, por vezes, o funcionamento desejado do circuito produz um diagrama e uma tabela de estados não completamente especificados. Nesses casos, cabe ao projetista decidir como proceder em relação aos itens não especificados durante a realização do projeto.

- A Figura 14.3 apresenta exemplos de tabelas de estados para circuitos *pulse-mode* Mealy e Moore. A tabela da Figura 14.3.a especifica que deverá ocorrer um pulso na saída z_p quando o circuito estiver no estado $q = B$ e ocorrer um pulso na entrada x_{p2} ou quando o circuito estiver no estado $q = C$ e ocorrer um pulso na entrada x_{p1}. Por sua vez, a tabela da Figura 14.3.b determina que a saída deverá assumir o nível $z_l = 1$ enquanto o circuito estiver no estado $q = D$ e não ocorrer um pulso em qualquer das entradas.

y_1y_2 \ $x_{p1}x_{p2}$	00	01	11	10
00	0	$0/1/X$	—	$0/1/X$
01	0	$0/1/X$	—	$0/1/X$
11	0	$0/1/X$	—	$0/1/X$
10	0	$0/1/X$	—	$0/1/X$

$\longleftrightarrow$

y_1y_2 \ $x_{p1}x_{p2}$	00	01	11	10
00	0	$0/\Pi/X$	—	$0/\Pi/X$
01	0	$0/\Pi/X$	—	$0/\Pi/X$
11	0	$0/\Pi/X$	—	$0/\Pi/X$
10	0	$0/\Pi/X$	—	$0/\Pi/X$

Figura 14.2: Equivalência de notações para mapa de Karnaugh utilizado na síntese de variáveis pulsadas.

q^n	q^{n+1}, z_p	
	x_{p1}	x_{p2}
A	$A, 0$	$B, 0$
B	$-$, $-$	$C, 1$
C	$A, 1$	$D, 0$
D	$A, 0$	$A, 0$

a) Circuito do tipo Mealy.

q^n	q^{n+1}		z_l^n
	x_{p1}	x_{p2}	
A	B	–	0
B	D	C	0
C	A	A	0
D	C	A	1

b) Circuito do tipo Moore.

Figura 14.3: Tabelas de estados para circuitos *pulse-mode* Mealy e Moore.

14.4.3 Exemplos de projeto

- Exemplo utilizando *flip-flop* JK, *master-slave*, ativado por pulso nas entradas J e K, enquanto a entrada de controle de sincronismo C é mantida em nível lógico "1".

y_1y_2 \ $x_{p1}x_{p2}$	00	01	11	10
00	0	0/Π	—	0/Π
01	0	0/Π	—	0/Π
11	0	0/Π	—	0/Π
10	0	0/Π	—	0/Π

(*a*)

y_1y_2	x_{p1}	x_{p2}
00	0/Π	0/Π
01	0/Π	0/Π
11	0/Π	0/Π
10	0/Π	0/Π

(*b*)

Figura 14.4: Mapas de Karnaugh para síntese de variáveis pulsadas, considerando-se duas entradas pulsadas: (a) Mapa completo e (b) Mapa simplificado.

y_1y_2 \ $x_{p1}x_{p2}x_{p3}$	000	001	011	010	100	101	111	110
00	0	0/Π	—	0/Π	0/Π	—	—	—
01	0	0/Π	—	0/Π	0/Π	—	—	—
11	0	0/Π	—	0/Π	0/Π	—	—	—
10	0	0/Π	—	0/Π	0/Π	—	—	—

(*a*)

y_1y_2	x_{p1}	x_{p2}	x_{p3}
00	0/Π	0/Π	0/Π
01	0/Π	0/Π	0/Π
11	0/Π	0/Π	0/Π
10	0/Π	0/Π	0/Π

(*b*)

Figura 14.5: Mapas de Karnaugh para síntese de variáveis pulsadas, considerando-se três entradas pulsadas: (a) Mapa completo e (b) Mapa simplificado.

14.5 Circuitos *ripple-clock*

14.5.1 Motivação

- A classe de circuitos *ripple-clock* surge como uma tentativa de otimização no acionamento dos elementos de memória do circuito seqüencial.
- A mudança na forma de acionamento dos elementos de memória pode levar a uma simplificação da lógica combinacional do circuito seqüencial.
- Tal simplificação acarreta uma redução da quantidade de *hardware* do circuito combinacional.

14.5.2 Operação

- Nos circuitos do tipo *clock-mode*, os elementos de memória são acionados simultaneamente pelo sinal de sincronismo (*clock*).
- De forma semelhante, nos circuitos do tipo *pulse-mode*, os elementos de memória são potencialmente acionados em paralelo. A diferença, neste caso, é que, dependendo dos sinais de entrada, alguns elementos de memória podem não ser acionados em uma determinada mudança de estado. Ainda assim, a forma de acionamento é estruturalmente paralela.
- Nos circuitos *ripple-clock*, o acionamento é realizado por uma seqüência de eventos. Um sinal de entrada provoca o acionamento de um ou mais elementos de memória. Por sua vez, as modificações nas saídas destes elementos acionam outros elementos de memória. Este mecanismo se repete até que um último conjunto de elementos de memória seja ativado, completando a mudança de estado do circuito seqüencial.

14.5.3 Desvantagens

- As desvantagens deste tipo de acionamento são: i) o aumento do tempo de estabilização nas mudancas de estado, o que é equivalente à redução da freqüência máxima de operação do circuito seqüencial e ii) o surgimento de estados e de conjunto de saídas intermediários (instáveis) durante uma mudança de estados estáveis.
- No cálculo do período mínimo para o sinal de acionamento inicial, deve-se levar em conta o pior caso, que é quando ocorrem todos os níveis de acionamento intermediários.

14.5.4 Técnica de projeto

- Na síntese da lógica combinacional para os circuitos *clock-mode* (ou *pulse-mode*), torna-se necessário que os valores das variáveis de excitação que preenchem os mapas-K sejam rigidamente controlados, pois os elementos de memória serão constantemente (ou potencialmente) acionados, independentemente do estado em que se encontre o circuito.
- No caso dos circuitos *ripple-clock*, os elementos de memória poderão ser acionados apenas quando necessário. Portanto, para os estados onde não ocorrerá acionamento, os valores da variáveis de excitação podem ser considerados *don't care* ("X"), o que pode conduzir a simplificações na lógica combinacional.
- O desafio, portanto, é obter um arranjo de acionamentos que reduza ao máximo a lógica combinacional necessária.

14.5.5 Exemplo

- O exemplo mais clássico é a obtenção do circuito *ripple-clock* para um contador binário, a partir de um projeto de circuito *clock-mode* que utiliza um *flip-flop* JK sensível a transição.

14.6 Circuitos *controlled-clock*

- Os elementos de memória são do tipo *clocked*.
- Assim como nos circuitos seqüenciais *clock-mode*, existe somente uma entrada pulsada, sendo esta periódica e denominada de *clock*.
- Porém, o sinal de *clock* não é aplicado diretamente nas entradas de controle dos elementos de memória.
- Como o próprio nome indica, o sinal de *clock* principal (*master clock*) é combinado com sinais de controle do tipo nível (sinais de entrada e/ou variáveis de estado) para gerar fontes secundárias de sinais pulsados, sincronizados com o sinal de *clock*.
- Tais sinais pulsados secundários são aplicados nas entradas de controle dos elementos de memória ou ainda enviados para circuitos do tipo *pulse-mode*.
- As Figuras 14.6 e 14.7 apresentam exemplos de controle de sinal de *clock*.

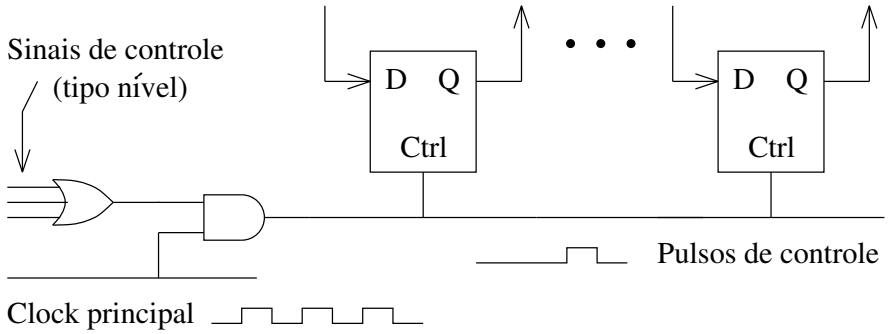


Figura 14.6: Exemplo 1 de controle de sinal de *clock*.

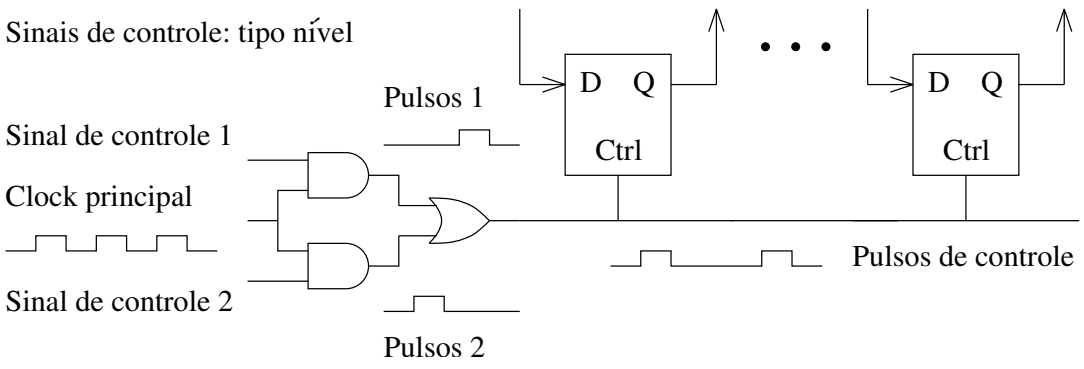


Figura 14.7: Exemplo 2 de controle de sinal de *clock*.

- Em uma grande variedade de aplicações de sistemas digitais, o conteúdo dos elementos de memória ou não é modificado ou é condicionalmente carregado com o resultado da aplicação de alguma função sobre um conjunto de dados.
- Portanto, para tais sistemas, o *flip-flop* do tipo D é o mais utilizado, pois realiza a função de armazenamento com um custo menor do que o *flip-flop* do tipo JK.
- Sinais de controle de CLEAR e PRESET, independentes do sinal de ativação do *flip-flop*, são comumente utilizados.
- Porém, a fim de evitar mudanças impróprias, provocadas pela aplicação de tais sinais ao mesmo tempo em que o *flip-flop* é ativado, tais entradas de controle são normalmente utilizadas apenas para a inicialização (*reset*) do circuito.
- Uma arquitetura do tipo *controlled-clock* comumente encontrada na prática é a denominada Lógica de Transferência entre Registradores (*Register-Transfer Logic* ou RTL).
- Nos circuitos que possuem tal arquitetura, os dados são condicionalmente armazenados em registradores.
- De acordo com o processamento a ser realizado, os dados são transferidos entre registradores específicos.
- Eventualmente, podem ser inseridos circuitos combinacionais no caminho de ligação entre dois registradores, os quais serão responsáveis pela implementação de funções lógicas e/ou aritméticas, necessárias ao processamento dos dados armazenados.
- As transferências são controladas por meio de sinais pulsantes secundários, sincronizados com o sinal de *clock* principal.
- Normalmente, todos os sinais de um sistema são organizados em conjuntos de ligações, denominados de barras ou barramentos: barra de dados (*data bus*), barra de controle (*control bus*) e barra de alimentação (*power bus*).
- A transferência entre dois registradores é realizada por meio de uma barra de dados (*data bus*).
- A Figura 14.8, apresentada em [HP81], ilustra um modelo genérico para circuitos seqüenciais *controlled-clock*.
- O modelo separa o sistema em duas partes: um bloco de processamento de dados e um bloco de controle.
- O bloco de processamento de dados incorpora os registradores que armazenam os dados a serem processados e a lógica combinacional necessária à realização das funções de processamento.
- O bloco de controle representa os circuitos seqüenciais responsáveis por gerar os sinais de controle (níveis e pulsos) que realizam as transferências apropriadas, sincronizadas com o sinal de *clock* principal.
- Normalmente, o número de linhas de entradas de controle e o número de linhas de sinais de controle são pequenos em comparação tanto ao número de linhas de dados de entrada e de saída quanto ao número de linhas de interconexão de dados, internas ao bloco de processamento de dados.

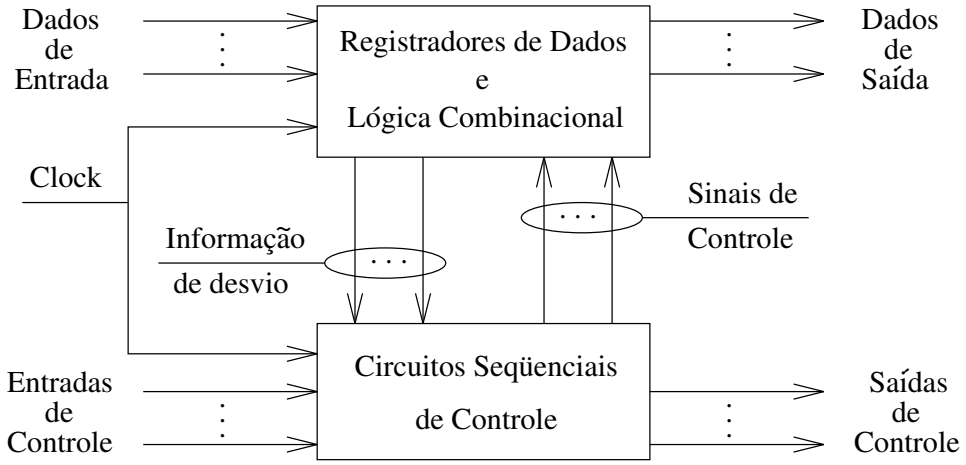


Figura 14.8: Modelo genérico para circuitos seqüenciais *controlled-clock*.

Capítulo 15

Circuitos seqüenciais *level-mode*

15.1 Introdução

- A Figura 15.1 apresenta um modelo genérico para circuitos seqüenciais *level-mode*.

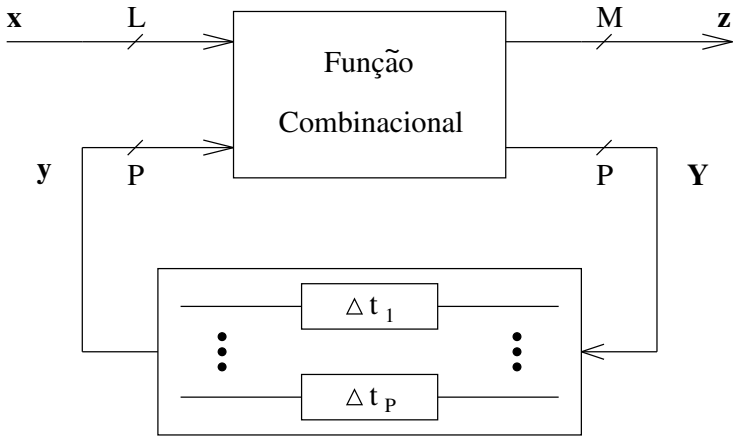


Figura 15.1: Modelo genérico para circuitos seqüenciais *level-mode*.

- O modelo destaca a ausência de elementos de memória permanente.
- Ao invés disso, tal estrutura se utiliza de elementos de memória temporária, implementados através de atrasos.
- Por sua vez, os atrasos que implementam o bloco de memória não são blocos de retardo isolados. Eles representam a concentração de atrasos de propagação existentes no circuito combinacional. Conseqüentemente, os valores de tais atrasos podem variar ao longo do tempo, uma vez que eles serão dependentes dos diversos fluxos que os sinais podem percorrer através do circuito combinacional.
- Assim como nos demais classes: $y_k^{n+1} = f_k(Y_1^n, \cdots, Y_P^n)$, $k = 1, 2, \cdots, R$.
- Mais especificamente, neste caso: $P = R$ e $y_k(t + \Delta t_k) = Y_k(t)$, $k = 1, 2, \cdots, P$.
- Todos os sinais presentes no circuito são do tipo nível.
- Uma mudança de estado é provocada por uma mudança de nível nos sinais de entrada.

- Em resumo, a Figura 15.1 indica que um circuito seqüencial *level-mode* genérico é simplesmente um circuito combinacional realimentado, com entradas do tipo nível.
- Porém, deve ser lembrado que, por definição, todo circuito seqüencial deve ser realimentado, mas a simples realimentação de um circuito combinacional não garante que ele passe a se comportar como um circuito seqüencial.
- Em circuitos seqüenciais pulsados (*pulsed* e *clock-mode*), é natural que as saídas dos elementos de armazenamento sejam escolhidas como variáveis de estado, uma vez que eles são também os elementos de sincronismo do sistema.
- Nos circuitos *level-mode* a realimentação é continuamente aplicada. Assim, qualquer ponto dela pode ser identificado como uma variável de estado sem causar prejuízo à análise ou ao projeto do circuito.
- Diz-se que um circuito opera em modo fundamental se, e somente se, não forem permitidas mudanças nos valores de suas variáveis de entrada até que o circuito atinja um estado estável.
- Deve-se observar que o modo fundamental é uma restrição quanto à forma como o circuito é operado e não quanto ao tipo de projeto executado.
- O modo fundamental pode ser implementado permitindo-se que apenas uma das variáveis de entrada seja modificada por vez e garantindo-se que modificações sucessivas em tais variáveis só ocorram após a estabilização do circuito.

15.2 Problemas comuns em circuitos *level-mode*

- Nos circuitos seqüenciais controlados por pulsos (*pulsed* e *clock-mode*) a realimentação é interrompida pelo bloco de memória e é ativada segundo um certo sincronismo.
- Por outro lado, nos circuitos seqüenciais *level-mode* a realimentação encontra-se ativa durante todo o tempo.
- Conseqüentemente, podem ocorrer instabilidades e incertezas.
- Alguns problemas mais comuns são:
  - As condições de entrada ou de saída de um circuito podem ser indeterminadas.
  - A condição da saída de um circuito pode ser instável, a qual pode apresentar mudanças ainda que as entradas não sejam modificadas.
  - A condição da saída de um circuito, mesmo que estável, pode não ser preditível a partir das condições da entrada.
- As soluções mais empregadas para tais problemas são:
  - Evitar instabilidades crônicas (oscilações): se o circuito exibe oscilações para alguns valores de entrada e é estável para outros, então as condições que imprimem oscilações devem ser evitadas.
  - Evitar incertezas: se o circuito exibe comportamento indeterminado para alguns valores de entrada e determinismo para outros, então as condições que provocam indeterminismo devem ser evitadas.

  - Operar em modo fundamental.
  - Operar em modo pulsado (*pulsed* e *clock-mode*).

- Alguns pontos devem ser ressaltados:

  - Circuitos que exibem oscilação sob certas condições não podem ser utilizados em aplicações de armazenamento ou processamento de dados. Porém, tal comportamento é essencial quando a intenção é gerar sinais de sequenciamento ou temporização.
  - Nem sempre é possível garantir a operação em modo fundamental, uma vez que sinais provenientes de diversas fontes diferentes podem variar aleatoriamente. Nesses casos, uma solução é empregar circuitos sincronizadores extras para garantir a operação em modo fundamental.

15.3 Exemplo de análise de circuito *level-mode*

- Análise de dois circuitos seqüenciais que implementam um *flip-flop* SR.
- Tabela de transição (de estados).
- Tabela de fluxo de estados (*flow table*).
- Tabela de fluxo de estados primitiva (*primitive flow table*).

15.4 Exemplo de projeto de circuito *level-mode*

- Diversas opções de projeto para circuitos seqüenciais que implementem um *flip-flop* SR.
- Definição de uma especificação para um *flip-flop* SR.
- Exemplo de diagrama de tempo.
- Tabela de fluxo de estados primitiva (*primitive flow table*).
- Tabela de fluxo de estados (*flow table*) minimizada ou reduzida.
- Tabela de atribuição de estados.
- Tabela de transição (de estados).
- Síntese das variáveis de excitação e de saída.
- Circuito final.

15.5 Problemas causados pela realimentação contínua

15.5.1 Problemas causados pelo bloco de lógica combinacional

- Existem dois efeitos comuns em circuitos combinacionais: corrida (*race*) e perigo (*hazard*).
- No primeiro caso, após uma mudança nos sinais binários de entrada, espera-se alterar mais de um dos sinais binários de saída. Devido a atrasos internos, os sinais de saída, partindo de um valor inicial (estável), podem assumir configurações intermediárias (instáveis) antes de atingir o seu valor final (estável).
- No segundo caso, após uma mudança nos sinais binários de entrada, duas situações podem ocorrer. Na primeira delas, espera-se que o valor de um determinado sinal binário de saída não seja modificado. Porém, devido a atrasos internos, ainda que o valores inicial e final sejam o mesmo, surgem variações intermediárias. Isso é denominado perigo estático (*static hazard*). Na segunda situação, espera-se que o valor de um determinado sinal binário de saída seja coplementado. Porém, devido a atrasos internos, ainda que o valor final seja o complemento do valor inicial, surgem variações intermediárias. Isso é denominado perigo dinâmico (*dynamic hazard*).
- Para os circuitos combinacionais, embora a ocorrência de valores intermediários não previstos seja inoportuna, uma solução simples é aguardar a estabilização do resultado final.
- Em circuitos seqüenciais pulsados (*pulsed* e *clock-mode*), a ocorrência de configurações intermediárias nas variáveis de excitação também não representa sério problema, uma vez que a realimentação é interrompida pelo bloco que gera e armazena as variáveis de estado. Novamente, uma solução simples é aguardar a estabilização do resultado final.
- Porém, nos circuitos seqüenciais *level-mode*, a realimentação acontece de forma contínua.
- Nesse caso, os valores intermediários não previstos, causados por corridas e/ou perigos no bloco combinacional, geram estados intermediários não previstos, os quais podem provocar mudanças de estado não desejadas, comprometendo o funcionamento do circuito seqüencial.

15.5.2 Problema natural dos circuitos *level-mode*

- Devido à realimentação contínua, os circuitos *level-mode* apresentam um problema envolvendo duas ou mais variáveis de excitação/estado.
- Supondo operação em modo fundamental, após uma variação nos sinais de entrada, uma variável de excitação Y_1 pode sofrer modificação, ser realimentada e atuar sobre uma outra variável de excitação Y_2, antes que a variação da entrada exerça influência sobre Y_2.
- Nesse caso, Y_2 pode assumir um valor não esperado, comprometendo o funcionamento do circuito seqüencial.
- Esse comportamento é denominado de perigo essencial (*essential hazard*).
- Uma vez que o problema é associado ao tipo de estrutura e à sua especificação, ele pode ser detectado diretamente na tabela de fluxo.

- Supondo um sinal de entrada binário, que sofra três variações consecutivas. Caso a primeira e a terceira variações conduzam o circuito aos estados q_1 e q_3, tal que $q_1 \neq q_3$, então existe perigo essencial na tabela de fluxo do circuito [Ung59].

- O comportamento em questão irá ocorrrer se quaisquer duas colunas adjacentes da tabela de fluxo exibirem um dos dois padrões apresentados na Figura 15.2.

(1)	2
3	(2)
(3)	(3)

(a)

(1)	2
4	(2)
	(3)
(4)	3

(b)

Figura 15.2: Padrões de identificação de perigo essencial em tabelas de fluxo.

15.6 Solução para as corridas: atribuição de estados

15.6.1 Definição do problema

Objetivo da atribuição de estados

- Em circuitos seqüenciais pulsados (*pulsed* e *clock-mode*) a escolha da atribuição de estados visa a minimização do bloco de lógica combinacional.

- Em circuitos seqüenciais *level-mode*, operando em modo fundamental, o problema de estados intermediários, causados por corridas no bloco combinacional, pode ser resolvido através de uma atribuição de estados adequada.

- Dependendo da tabela de fluxo em questão, para que se encontre uma atribuição de estados adequada, pode ser necessário aumentar o número de estados do circuito.

Tipos de mudança de estado

- Duas situações podem ocorrer durante uma mudança de estado: i) alteração imediata de estado ou ii) surgimento de estados intermediários (instáveis) não previstos.

- No primeiro caso, as variáveis de estado modificam-se de tal forma que o circuito passa diretamente do estado inicial ao final, sem estados intermediários. Na prática, isso ocorre porque apenas uma das variáveis de estado necessita trocar de valor.

- No segundo caso, duas situações podem ocorrer: ciclo (*cycle*) ou corrida (*race*).

- Um ciclo é definido por uma seqüência única de estados intermediários, instáveis, entre dois estados estáveis (o inicial e o final).

- Os únicos problemas causados pelo ciclo são o prolongamento e a não uniformidade do tempo de estabilização da mudança de estado.

- A corrida caracteriza-se pela existência de diferentes ciclos para um mesmo estado inicial estável. Nesse caso, não é possível prever por qual ciclo o circuito irá fluir.
- Dois tipos de corrida podem ser definidos: não crítica (*non-critical*) e crítica (*critical*).
- Nas corridas não críticas, o estado final estável é sempre o mesmo, independentemente da seqüência de troca das variáveis de estado e, portanto, dos ciclos percorridos. Nesses casos, os problemas são os mesmos dos ciclos.
- Nas corridas críticas, os diferentes ciclos podem levar a diferentes estados finais estáveis. Portanto, corridas críticas representam comportamento não desejado.
- A Figura 15.3 apresenta um quadro resumo das mudanças de estado nos circuitos seqüenciais *level-mode*, operando em modo fundamental.

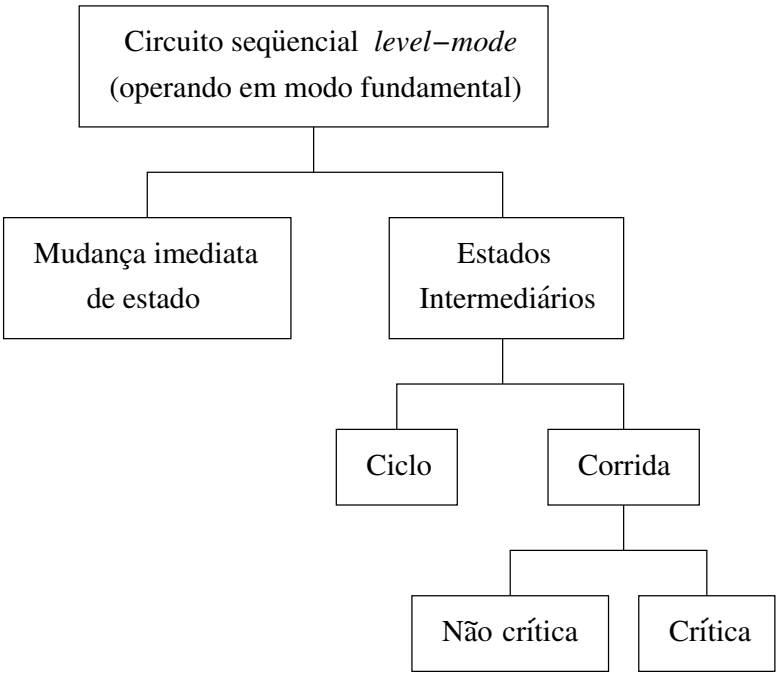


Figura 15.3: Quadro resumo das mudanças de estado nos circuitos seqüenciais *level-mode*, operando em modo fundamental.

15.6.2 Possíveis soluções

Origem do problema

- Em função do que foi exposto, pode-se concluir que: i) uma alteração imediata de estado pode ser interpretada como um caso particular de ciclo e ii) as situações de corrida (ciclos múltiplos) podem acarretar resultados indesejáveis (corrida crítica).
- Portanto, uma solução para o problema de funcionamento indesejado é adotar uma atribuição de estados que realize mudanças de estado apenas por ciclos.
- Para que uma mudança de estado seja executada em ciclo, cada par de estados, do inicial ao final, deve possuir adjacência lógica.
- Dessa forma, para cada mudança de estado, apenas uma variável deverá trocar de valor, evitando a ocorrência de corrida entre as variáveis.

Identificação do problema

- A análise dos tipos de mudanças de estado (ciclos e/ou corridas) que ocorrem em uma dada tabela de fluxo de estados pode ser feita através de um hipercubo booleano.
- Inicialmente, um hipercubo é montado, contendo um número de nós suficiente para conter a quantidade de estados estáveis da tabela de fluxo.
- Em seguida, percorrendo a tabela de fluxo, os estados são associados aos nós do hipercubo.
- A presença de ciclos e/ou corridas é verificada pelas transições presentes no hipercubo.
- Transições realizadas pelas arestas do hipercubo representam ciclos.
- Transições que ocorrem por diagonais significam corridas.
- A classificação das corridas deve ser realizada com o auxílio da tabela de fluxo.
- Supondo-se uma tabela de fluxo organizada de forma que as combinações de entrada definam as colunas, uma transição para uma coluna que contenha apenas um estado estável é associada a uma corrida não crítica. Por outro lado, se a coluna possuir dois ou mais estados estáveis diferentes, a transição representa uma corrida crítica.

Estados reservas (*spare states*)

- A atribuição de estados deve ser feita de tal forma que sejam respeitadas todas as adjacências lógicas em todas as mudanças de estado.
- Dada uma determinada tabela e um determinado número de estados, pode-se não conseguir uma atribuição de estados adequada.
- Neste caso, devem-se empregar estados reservas (*spare states*).
- Para números de estados que não sejam potências de dois, podem-se usar os estados extras como estados reservas.
- Porém, quando o número de estados é uma potência de dois ou não se consegue uma atribuição adequada com os estados extras já existentes, deve-se gerar estados reservas acrescentando-se novas variáveis de estado.

Técnicas de atribuição

- Existem duas técnicas básicas para usar os estados reservas: atribuição por múltiplas linhas (*multiple-row assignment*) e atribuição por linhas compartilhadas (*shared-row assignment*).
- Na técnica de atribuição por múltiplas linhas, aproveitando-se o fato de que o número de estados é dobrado para cada nova variável de estado acrescentada, cada estado original passa a ser representado por duas ou mais linhas na tabela de transição. Esta multiplicidade de representação para cada estado permite que se implemente adjacência lógica para cada par de estados. Conseqüentemente, em qualquer tabela de fluxo, com qualquer número de estados originais, todas as corridas podem ser transformadas em ciclos.
- A técnica de múltiplas linhas necessita que o número de linhas da tabela de transição seja igual a, pelo menos, o dobro do número de estados. Assim, caso o número de estados não seja uma potência de dois, é recomendável que se tente aplicar a técnica de linhas compartilhadas.
- Na técnica de linhas compartilhadas, as combinações reservas de variáveis de estado (linhas da tabela de transição) não são atribuídas a estados individuais. Como o próprio nome já diz, cada linha é compartilhada por diferentes configurações de entrada (colunas da tabela de transição), a fim de transformar corridas em ciclos.

Atribuições tabeladas

- Dois tipos de atribuições de estado podem ser empregadas: universal e padrão [Sau67].
- Atribuições universais são apresentadas em [Sau67], as quais se utilizam de 2 variáveis para 3 estados, 3 variáveis para 4 estados, 4 variáveis para até 8 estados e 5 variáveis para até 12 estados.
- Tais atribuições, ilustradas nas Tabelas 15.1 – 15.4, realizam quaisquer tabelas de fluxo, com os referidos números de estados, sem corridas críticas.
- Dado um determinado número de estados, as atribuições padrões procuram utilizar um número menor de variáveis de estado para representá-los. Porém, elas não são capazes de realizar todas as tabelas com tal número de estados.
- Um exemplo de atribuição padrão para tabelas com 5 estados é apresentado na Tabela 15.5.

		y_1	
		0	1
y_0	0	+	+
	1	+	

Tabela 15.1: Atribuição de estados universal, usando shared-row, para tabelas de 3 estados.

		y_2y_1			
		00	01	11	10
y_0	0	0	2	1	3
	1	1	3	0	2

Tabela 15.2: Atribuição de estados universal, usando multiple-row, para tabelas de 4 estados.

		y_3y_2			
		00	01	11	10
y_1y_0	00	+	+		
	01			+	+
	11	+	+		
	10			+	+

Tabela 15.3: Atribuição de estados universal, usando shared-row, para tabelas de 5 a 8 estados.

		$y_4y_3y_2$							
		000	001	011	010	110	111	101	100
y_1y_0	00	+	+			+	+		
	01			+	+				
	11	+	+						
	10			+	+			+	+

Tabela 15.4: Atribuição de estados universal, usando shared-row, para tabelas de 9 a 12 estados.

		y_2y_1			
		00	01	11	10
y_0	0	+		+	
	1	+	+	+	

Tabela 15.5: Atribuição de estados padrão, usando shared-row, para tabelas de 5 estados.

Conjunto de destinação (*destination set*)

- Conjunto de destinação (*destination set*) é um conceito que se pode utilizar na tentativa de atender a uma determinada tabela de fluxo com uma atribuição que utilize apenas estados reservas já existentes, sem acrescentar uma variável de estado extra.

- Dada uma tabela de fluxo, formam-se conjuntos de destinação para cada configuração das variáveis de entrada (coluna da tabela).

- Para cada coluna, tais conjuntos são formados por um estado estável da coluna com um estado (linha da tabela) que faça transição para o estado estável.

- A fim de que não haja corridas críticas, os membros de cada conjunto de destinação devem ser logicamente adjacentes ou devem ser alocados, em relação aos estados reservas, de forma que as transições cíclicas formadas para todos os conjuntos sejam atendidas sem interferência mútua (cruzamento de ciclos).

Comparações

- Uma comparação entre as duas técnicas pode ser feita com base na complexidade e no tempo de operação do circuito final.

- A técnica de linhas compartilhadas requer um número menor de variáveis de estado. Portanto, o seu uso gera circuitos mais simples.

- A técnica de múltiplas linhas gera transições imediatas. Portanto, o seu emprego produz circuitos com menor tempo de operação.

- Outras técnicas, que reduzem o tempo de operação do circuito, embora demandem maior tempo de projeto e aumento da complexidade do circuito, podem ser encontradas em [Ung69].

15.7 Solução para os perigos

- Dado o perigo estático para o valor binário "1", ele ocorre porque o circuito desativa o mintermo inicial antes de ativar o mintermo final. Dessa forma, acontece a transição $1|_{min\_inicial} \rightarrow 0 \rightarrow 1|_{min\_final}$.

- Portanto, para solucionar o problema, basta acrescentar um mintermo redundante, que permanecerá ativo durante a troca dos mintermos inicial e final. Assim, será realizada a transição $1|_{min\_inicial} \rightarrow 1|_{min\_redundante} \rightarrow 1|_{min\_final}$.

- Adicionalmente, é apresentado em [McC65] o seguinte teorema:
"Um circuito combinacional implementado na forma padrão SOP de segunda ordem que for livre de todos os perigos estáticos para o valor binário "1", será livre de todos os perigos estáticos e dinâmicos.".

- Finalmente, uma forma comum de evitar o perigo essencial é acrescentar atrasos de propagação (inversores em número par) ao circuito.

15.8 Valores das saídas em estados instáveis

- Em mudanças de estados que se fazem por meio de ciclos, deve-se tomar cuidado com os valores atribuídos para as saídas durante os estados instáveis, a fim de se evitar a geração de pulsos espúrios.
- Se, tanto no estado inicial quanto no estado final, o valor especificado para a saída for o mesmo, ele deverá permanecer constante durante o ciclo.
- Se, do estado inicial para o estado final, os valores especificados para a saída forem diferentes, deverá ocorrer apenas uma mudança durante o ciclo. Conseqüentemente, o valor da saída só poderá ser especificado como *don´t care* ('X' ou '—') para um dos estados do ciclo.

Parte V

Circuitos digitais programáveis

Capítulo 16

Circuitos programáveis

16.1 Introdução

De acordo com a função realizada por um circuito digital, ele pode ser classificado em: circuito fixo (ou invariante no tempo) e circuito variável (ou variante no tempo). Como os próprios nomes já indicam, um circuito fixo realiza uma única função, enquanto um circuito variável pode permitir a realização de um conjunto de funções diferentes.

Costuma-se dizer que circuitos variáveis são programáveis. Embora não esteja errado, o termo programável induz ao pensamento de que o circuito é capaz de executar o que hoje é denominado de um programa de computador. Logo, talvez seja mais indicado dizer que os circuitos variáveis são configuráveis, deixando o termo programável para indicar um tipo específico de circuito configurável. A partir dessa denominação, podem-se definir diversos tipos de configurações e, portanto, diversos tipos de circuitos variáveis.

Pode-se pensar em dividir a configuração dos circuitos variáveis em: externa e interna.

Na configuração dita externa, podem-se agrupar os circuitos que possuem uma estrutura fixa, mas que mudam de função de acordo com as combinações de valores aplicados em algumas de suas entradas. Esse tipo de configuração pode ser usado em circuitos digitais combinacionais e/ou seqüenciais que ocupam uma extensa faixa de complexidade: portas lógicas, funções lógicas, circuitos digitais básicos, blocos funcionais e até mesmo sistemas digitais mais complexos.

Na configuração denominada de interna, podem-se reunir os circuitos que possuem uma estrutura com elementos constituintes fixos, mas que podem sofrer modificações na sua interconexão (roteamento), no conteúdo de informação armazenada (dado) ou em ambos. Esse tipo de configuração é preferencialmente utilizado em circuitos digitais complexos, que podem implementar circuitos combinacionais e/ou seqüenciais, tais como: Dispositivo Lógico Programável (ou *Programmable Logic Device* ou PLD) e processador (microprocessador, microcontrolador e processador de sinal digital).

Naturalmente, sistemas computacionais de complexidade elevada utilizam todos os tipos de circuitos (fixos e variáveis).

A Tabela 16.1 resume a classificação de circuitos digitais de acordo com a função realizada, definida acima.

A seguir, é apresentada uma introdução sobre o assunto. Deve ser ressaltado que, devido ao fato de apresentarem um conteúdo de alta complexidade e de alta especificidade, não serão abordades os diveros tipos de processadores nem os incontáveis sistemas computacionais de complexidade elevada.

Circuito digital	→ Fixo		
	→ Variável	→ Configuração externa	
		→ Configuração interna	→ Troca de roteamento
			→ Troca de dados
			→ Troca de ambos

Tabela 16.1: Classificação de circuitos digitais de acordo com a função realizada.

16.2 Circuitos configuráveis externamente

A seguir, são apresentados alguns exemplos de circuitos configuráveis externamente. São abordadas as configurações de portas lógicas, de funções lógicas, de circuitos digitais básicos, de blocos funcionais e de sistemas digitais mais complexos.

16.2.1 Configuração de portas lógicas

Nesse caso, a configuração externa do dispositivo possibilita a implementação uma porta lógica específica, dentro de um conjunto fixo e reduzido de opções.

Em algumas aplicações, é interessante que o dispositivo implemente apenas uma porta lógica. Isso facilita o roteamento na placa de circuito impresso, uma vez que a porta lógica é localizada diretamente na trajeto do sinal, evitando eventuais desvios de trilhas. Como conseqüência, pode-se ter trilhas diretas e mais curtas, reduzindo resistências e acoplamentos eletromagnéticos das trilhas, o que se traduz em menores tempos de resposta. Além disso, se for necessário realizar alguma modificação simples no circuito, torna-se mais fácil redesenhar a placa de circuito impresso.

Por exemplo, no início dos anos 2000, a Fairchild Semiconductors[1] fabricava uma família de dispositivos denominada de TinyLogic. Os dispositivos dessa família possuíam características adequadas ao uso em aparelhos portáteis, tais como: baixo custo, dimensão reduzida, baixo consumo e reduzido tempo de resposta. Existiam as seguintes séries: HS (*High Speed*), HT (*High speed Ttl*), UHS (*Ultra-High Speed*) e ULP (*Ultra-Low Power*). Um exemplo de dispositivos da série UHS são o NC7SZ57 e o NC7SZ58. Algumas das características apresentadas por esses dispositivos eram:

- Podiam ser configuradas cinco portas lógicas diferentes, com possíveis negações lógicas nas entradas e/ou na saída. As portas eram: AND, NAND, OR, NOR e XOR/XNOR.
- A saída de um deles era logicamente complementar a do outro.
- No caso da alimentação assumir o valor de tensão nulo, tanto as entradas quanto a saída assumiam um estado de alta impedância.
- A corrente de saída era suficiente para acionar diretamente alguns dispositivos, tais como: LED, acoplador óptico e relé de baixa corrente.
- Todas as entradas possuíam histerese.
- O encapsulamento (SC70) era do tipo SMD (*Surface-Mounted Device*), com dimensões reduzidas (2,00 mm × 1,25 mm) e organização DIL (*Dual In Line*) de seis pinos, sendo: dois pinos para alimentação (V_{CC} e GND), três pinos de entrada e um pino de saída.

[1] http://www.fairchildsemi.com

A Tabela 16.2 descreve o comportamento funcional dos dispositivos NC7SZ57 e NC7SZ58. Por sua vez, a Tabela 16.3 descreve as opções de configuração dos dispositivos. Em ambas as tabelas, são feitas as seguintes associações: "1" = $V_{HIGH} = V_{CC}$ e "0" = $V_{LOW} = GND$.

Cabe lembrar que se pode obter um segundo conjunto de portas diferentes, porém logicamente equivalentes ao primeiro, por meio da aplicação dos Teoremas de De Morgan e das seguintes relações: A XOR $B \equiv \overline{A}$ XNOR $B \equiv A$ XNOR $\overline{B}$ e A XNOR $B \equiv \overline{A}$ XOR $B \equiv A$ XOR $\overline{B}$.

Entradas			Saída	
E_2	E_1	E_0	NC7SZ57	NC7SZ58
0	0	0	1	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	1	0

Tabela 16.2: Comportamento funcional dos dispositivos NC7SZ57 e NC7SZ58.

Entradas	Saída	
	NC7SZ57	NC7SZ58
$E_0 = 0$	$\overline{E_1}$ NAND E_2	$\overline{E_1}$ AND E_2
$E_0 = 1$	E_1 AND E_2	E_1 NAND E_2
$E_1 = 0$	E_0 NOR E_2	E_0 OR E_2
$E_1 = 1$	E_0 NAND $\overline{E_2}$	E_0 AND $\overline{E_2}$
$E_2 = 0$	$\overline{E_0}$	E_0
$E_2 = 1$	E_1	$\overline{E_1}$
$E_0 = E_1$	E_0 XNOR E_2	E_0 XOR E_2
$E_0 = E_2$	E_0 NAND $\overline{E_1}$	E_0 AND $\overline{E_1}$
$E_1 = E_2$	$\overline{E_1}$ NAND E_0	$\overline{E_1}$ AND E_0

Tabela 16.3: Opções de configuração dos dispositivos NC7SZ57 e NC7SZ58.

16.2.2 Configuração de funções lógicas com multiplexador

Um multiplexador (MUX) digital é um dispositivo com N sinais de entrada E_n, M sinais de controle C_m e 1 sinal de saída S, onde: cada sinal assume apenas valores binários, $N \leq 2^M$ e $M \in \mathbb{N}^+$.

Um MUX tem a função de um seletor. A saída S deverá assumir o mesmo valor da entrada E_n, quando o padrão binário presente em $\boldsymbol{C} = [\ C_{M-1}\ C_{M-2}\ \cdots\ C_1\ C_0\ ]$ apresentar o valor $(\boldsymbol{C})_2 = n = [\boldsymbol{C}]_n$. Logo, os padrões binários de $\boldsymbol{C}$ podem ser interpretados como endereços que serão empregados na seleção do sinal de entrada que será copiado para a saída do MUX.

O modelo básico para um MUX, baseado em portas lógicas, é um arranjo padrão AND-OR, composto de um plano com N portas AND de $(M+1)$ entradas, seguido de um plano com 1 porta OR de N entradas. Nesse modelo, cada porta AND realiza a operação

$$AND\ (E_n, [\boldsymbol{C}]_n) = AND\ (E_n, [\ C_{M-1}\ C_{M-2}\ \cdots\ C_1\ C_0\ ]_n) = AND\ (E_n, m(n))\ ,$$

onde $m(n)$ representa o mintermo n. Portanto, enquanto a saída da porta AND endereçada pelo mintermo n apresenta uma cópia da entrada E_n do MUX, as demais portas AND terão saída igual a "0". Por sua vez, a porta OR completa a cópia da entrada selecionada para a sua saída, que é a saída do MUX.

Uma vez que a função básica de um MUX é a de um seletor, pode-se pensar em utilizá-lo para implementar uma função lógica genérica, por meio da configuração adequada das suas entradas (E_n e C_m).

Em uma primeira tentativa, basta interpretar a tabela verdade (TV) de uma função lógica de V variáveis como um processo de seleção de valores "0" e "1", onde os endereços da seleção são os padrões apresentados pelo conjunto de variáveis. Portanto, uma dada função lógica de V variáveis pode ser implementada por um MUX com $M = V$ e $N = 2^M$, conectando-se as M variáveis aos sinais C_m do MUX e fornecendo-se os valores "0" e "1" às entradas E_n do MUX, de acordo com a TV da função alvo.

Com o intuito de diminuir a quantidade de circuito necessária para implementar uma mesma função lógica, pode-se pensar em utilizar um MUX menor. Para isso, deve-se observar que, nas funções lógicas de V variáveis, haverá 2 valores na TV para cada combinação de $(V-1)$ variáveis. Esses valores poderão ser iguais a "00", "01", "10" ou "11", bem como poderão ser associados a "0", "1", V_k e $\overline{V_k}$, onde V_k é a variável desconsiderada no endereçamento. Portanto, pode-se implementar uma dada função lógica de V variáveis por um MUX com $M = (V-1)$ e $N = 2^M$, conectando-se as M variáveis escolhidas aos sinais C_m do MUX e fornecendo-se os valores "0", "1", V_k ou $\overline{V_k}$, às entradas E_n do MUX, de acordo com a TV da função alvo. A economia é significativa, pois o MUX é reduzido à metade e, mesmo no caso onde o complemento de uma das variáveis não esteja à disposição, é necessário que se acrescente apenas um inversor.

Pode-se obter uma redução ainda maior no tamanho do MUX, em troca da inclusão de um circuito lógico adicional. Para isso, deve-se observar que, nas funções de $V = (K+L)$ variáveis, haverá 2^L valores na TV para cada combinação de K variáveis. Esses valores poderão ser associados a "0", "1" e aos valores provenientes de combinações lógicas das L variáveis desconsideradas no endereçamento. Portanto, pode-se implementar uma dada função lógica de $V = (K+L)$ variáveis por um MUX com $M = K$ e $N = 2^M$, conectando-se as M variáveis escolhidas aos sinais C_m do MUX e fornecendo-se os valores "0", "1" e de combinações lógicas das L variáveis não utilizadas no endereçamento, às entradas E_n do MUX, de acordo com a TV da função alvo. Nesse caso, deve-se fazer um balanceamento entre a redução da quantidade de circuito do MUX e o aumento da quantidade de circuito adicional.

Deve ser ressaltado que, em todos os casos, diversas soluções podem ser propostas, dependendo de quais variáveis serão escolhidas para o endereçamento e da sua ordenação. Dentro desse universo de soluções, algumas delas poderão requerer menos circuitos do que outras.

A título de exemplo, será considerada a função lógica definida por

$$\begin{aligned} f(A,B,C,D) &= \sum\ m(1,3,5,7,10,11,12,13) \ = \prod\ M(0,2,4,6,8.9,14,15) \\ &= \left(\overline{A}\cdot D\right) + \left(A\cdot\overline{B}\cdot C\right) + \left(A\cdot B\cdot\overline{C}\right) \\ &= (A+D)\cdot\left(\overline{A}+B+C\right)\cdot\left(\overline{A}+\overline{B}+\overline{C}\right)\ . \end{aligned} \tag{16.1}$$

A Figura 16.1 ilustra algumas formas de organização do Mapa de Karnaugh da função exemplo, para implementação usando MUX.

AB \ CD	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	1	0	0
10	0	0	1	1

(a)

AC \ BD	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	1	0	0
10	0	0	1	1

(b)

AD \ BC	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	1	0	1
10	0	1	0	1

(c)

Figura 16.1: Mapas de Karnaugh da função exemplo, para implementação usando MUX.

Para um MUX com $M = 4$ e $N = 16$, pode ser adotada a seguinte configuração:

$$\begin{cases} [\ C_3\ C_2\ C_1\ C_0\ ] \ = \ [\ A\ B\ C\ D\ ] \\ [\ E_{15}\ E_{14}\ \cdots\ E_1\ E_0\ ] \ = \ [\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ ] \end{cases} .$$

Para um MUX com $M = 3$ e $N = 8$, podem ser adotadas as seguintes configurações:

$$\begin{cases} [\ C_2\ C_1\ C_0\ ] \ = \ [\ B\ C\ D\ ] \\ [\ E_7\ E_6\ \cdots\ E_1\ E_0\ ] \ = \ [\ \overline{A}\ 0\ 1\ A\ 1\ A\ \overline{A}\ 0\ ] \end{cases}$$

e

$$\begin{cases} [\ C_2\ C_1\ C_0\ ] \ = \ [\ A\ B\ C\ ] \\ [\ E_7\ E_6\ \cdots\ E_1\ E_0\ ] \ = \ [\ 0\ 1\ 1\ 0\ D\ D\ D\ D\ ] \end{cases} .$$

Para um MUX com $M = 2$ e $N = 4$, podem ser adotadas as seguintes configurações:

$$\begin{cases} [\ C_1\ C_0\ ] \ = \ [\ A\ B\ ] \\ [\ E_3\ E_2\ E_1\ E_0\ ] \ = \ [\ \overline{C}\ C\ D\ D\ ] \end{cases} ,$$

$$\begin{cases} [\ C_1\ C_0\ ] \ = \ [\ A\ C\ ] \\ [\ E_3\ E_2\ E_1\ E_0\ ] \ = \ [\ \overline{B}\ B\ D\ D\ ] \end{cases} ,$$

$$\begin{cases} [\ C_1\ C_0\ ] \ = \ [\ A\ D\ ] \\ [\ E_3\ E_2\ E_1\ E_0\ ] \ = \ [\ (B \oplus C)\ (B \oplus C)\ 1\ 0\ ] \end{cases} ,$$

$$\begin{cases} [\ C_1\ C_0\ ] \ = \ [\ C\ D\ ] \\ [\ E_3\ E_2\ E_1\ E_0\ ] \ = \ [\ (\overline{A}+\overline{B})\ (A\cdot\overline{B})\ (\overline{A}+B)\ (A\cdot B)\ ] \end{cases},$$

$$\begin{cases} [\ C_1\ C_0\ ] \ = \ [\ B\ D\ ] \\ [\ E_3\ E_2\ E_1\ E_0\ ] \ = \ [\ (\overline{A}+\overline{C})\ (A\cdot\overline{C})\ (\overline{A}+C)\ (A\cdot C)\ ] \end{cases}$$

e

$$\begin{cases} [\ C_1\ C_0\ ] \ = \ [\ B\ C\ ] \\ [\ E_3\ E_2\ E_1\ E_0\ ] \ = \ [\ (\overline{A}\cdot D)\ (A+D)\ (A+D)\ (\overline{A}\cdot D)\ ] \end{cases}.$$

16.3 Circuitos configuráveis internamente: PLDs

Os PLDs comerciais passaram por uma linha evolutiva de complexidade. Em ordem crescente de complexidade, os PLDs receberam as seguintes denominações:

- ROM (*Read-Only Memory*).
- PAL (*Programmable Array Logic*).
- PLA (*Programmable Logic Array*).
- PAL/PLA com um *flip-flop* do tipo D (DFF) em cada saída.
- GAL (*Generic Array Logic*).
- PALCE (*PAL CMOS Electrically erasable/programable device*).
- SPLD (*Simple Programmable Logic Device*).
- CPLD (*Complex Programmable Logic Device*).
- FPGA (*Field-Programmable Gate Array*).

16.4 Exercícios propostos

1. A implementação de funções lógicas usando multiplexadores é muito utilizada na prática devido à simplicidade do seu projeto.

   Suponha a função booleana que é expressa por

   $$F(A,B,C) = \left(\overline{A}\cdot\overline{B}\cdot C\right) + \left(\overline{A}\cdot B\cdot C\right) + \left(A\cdot\overline{B}\cdot\overline{C}\right) + \left(A\cdot B\cdot C\right) \ .$$

   Suponha ainda que estão à disposição um multiplexador (MUX) e as ligações para acesso aos seguintes sinais: as variáveis e as suas negações lógicas; uma fonte de alimentação positiva V_+ ("1" booleano); e uma fonte de alimentação negativa V_- ("0" booleano).

   Atenda aos seguintes itens:

   (a) Implemente a função booleana dada, usando **APENAS** um multiplexador "8 por 1" (MUX 8x1).

   (b) Implemente a função booleana dada, usando **APENAS** um multiplexador "4 por 1" (MUX 4x1).

   (c) Suponha que sinais auxiliares podem ser gerados, a partir das variáveis e das suas negações lógicas, usando portas lógicas AND, OR e NOT.
   Implemente a função booleana dada, usando **APENAS** um multiplexador "2 por 1" (MUX 2x1).

Parte VI

Apêndices

Apêndice A

Exemplos de expressões equivalentes, empregando NAND e NOR

A.1 Introdução

Neste apêndice, são apresentados alguns exemplos de expressões equivalentes, empregando os operadores binários NAND e NOR. Será assumido que ambos os operadores possuem um tempo de propagação típico de t_p s. O tempo de propagação é o intervalo de tempo decorrido entre a estabilização dos sinais de entrada e a estabilização do sinal de saída. Onde for mais relevante, serão indicados a quantidade de operadores utilizados e o tempo total de propagação.

A.2 Relações com o operador NAND

A seguir, são apresentados alguns exemplos de relações baseadas no operador NAND.

A.2.1 Operador NOT

$$\neg(A) = \neg(A \wedge A) = (A \uparrow A) \ .$$

$$A = \neg\left(\neg(A)\right) = \left(\neg(A) \uparrow \neg(A)\right) = (A \uparrow A) \uparrow (A \uparrow A) \ .$$

$$A = \neg\left(\neg(A)\right) = \neg\left(A \uparrow A\right) = (A \uparrow A) \uparrow (A \uparrow A) \ .$$

A.2.2 Operador AND

$$(A \wedge B) = \neg\left(\neg(A \wedge B)\right) = \neg\left(A \uparrow B\right) = (A \uparrow B) \uparrow (A \uparrow B) \ .$$

$$(\neg(A) \wedge B) = ((A \uparrow A) \uparrow B) \uparrow ((A \uparrow A) \uparrow B) \ .$$

$$(A \wedge \neg(B)) = (A \uparrow (B \uparrow B)) \uparrow (A \uparrow (B \uparrow B)) \ .$$

$$(\neg(A) \wedge \neg(B)) = ((A \uparrow A) \uparrow (B \uparrow B)) \uparrow ((A \uparrow A) \uparrow (B \uparrow B)) \ .$$

A.2.3 Operador OR

$$(A \vee B) = \neg\left(\neg(A \vee B)\right) = \neg\left(\neg(A) \wedge \neg(B)\right) = \left(\neg(A) \uparrow \neg(B)\right) = (A \uparrow A) \uparrow (B \uparrow B) \ .$$

$$\begin{aligned}(\neg(A) \vee B) &= ((A \uparrow A) \uparrow (A \uparrow A)) \uparrow (B \uparrow B) \\ &= \neg\left(\neg\left(\neg(A) \vee B\right)\right) = \neg\left(A \wedge \neg(B)\right) = \left(A \uparrow \neg(B)\right) = \left(A \uparrow (B \uparrow B)\right) \ .\end{aligned}$$

$$\begin{aligned}(A \vee \neg(B)) &= (A \uparrow A) \uparrow ((B \uparrow B) \uparrow (B \uparrow B)) \\ &= \neg\left(\neg\left(A \vee \neg(B)\right)\right) = \neg\left(\neg(A) \wedge B\right) = \left(\neg(A) \uparrow B\right) = ((A \uparrow A) \uparrow B) \ .\end{aligned}$$

$$\begin{aligned}(\neg(A) \vee \neg(B)) &= ((A \uparrow A) \uparrow (A \uparrow A)) \uparrow (B \uparrow B) \uparrow (B \uparrow B) \\ &= \neg\left(\neg\left(\neg(A) \vee \neg(B)\right)\right) = \neg\left(A \wedge B\right) = \left(A \uparrow B\right) \ .\end{aligned}$$

A.2.4 Operador XOR

$$\begin{aligned}(A \veebar B) &= (\neg(A) \wedge B) \vee (A \wedge \neg(B)) \\ &= \neg\left(\neg\left((\neg(A) \wedge B) \vee (A \wedge \neg(B))\right)\right) \\ &= \neg\left(\neg\left(\neg(A) \wedge B\right) \wedge \neg\left(A \wedge \neg(B)\right)\right) \\ &= \left(\neg\left(\neg(A) \wedge B\right) \uparrow \neg\left(A \wedge \neg(B)\right)\right) \\ &= \left(\neg(A) \uparrow B\right) \uparrow \left(A \uparrow \neg(B)\right) \\ &= ((A \uparrow A) \uparrow B) \uparrow (A \uparrow (B \uparrow B)) \ . \\ &\quad \boxed{\text{Expressão 1: 5 NANDs e 3 } t_p}\end{aligned}$$

$$\begin{aligned}&= \left(\neg\left((\neg(A) \wedge B) \vee (0)\right) \uparrow \neg\left((A \wedge \neg(B)) \vee (0)\right)\right) \\ &= \left(\neg\left((\neg(A) \wedge B) \vee (\neg(B) \wedge B)\right) \uparrow \neg\left((A \wedge \neg(B)) \vee (A \wedge \neg(A))\right)\right) \\ &= \left(\neg\left((\neg(A) \vee \neg(B)) \wedge (B)\right) \uparrow \neg\left((A) \wedge (\neg(A) \vee \neg(B))\right)\right) \\ &= \left(\left((\neg(A) \vee \neg(B)) \uparrow (B)\right) \uparrow \left((A) \uparrow (\neg(A) \vee \neg(B))\right)\right) \\ &= (((A \uparrow B) \uparrow B) \uparrow (A \uparrow (A \uparrow B))) \ . \\ &\quad \boxed{\text{Expressão 2: 4 NANDs e 3 } t_p}\end{aligned}$$

$$\begin{aligned}&= (A \vee \neg(B)) \uparrow (\neg(A) \vee B) \\ &= ((A \vee \neg(B)) \wedge (1)) \uparrow ((\neg(A) \vee B) \wedge (1)) \\ &= ((A \vee \neg(B)) \wedge (B \vee \neg(B))) \uparrow ((\neg(A) \vee B) \wedge (\neg(A) \vee A)) \\ &= ((A \wedge B) \vee (\neg(B))) \uparrow ((\neg(A)) \vee (A \wedge B)) \\ &= \neg\left(\neg\left(A \wedge B\right) \wedge (B)\right) \uparrow \neg\left((A) \wedge \neg\left(A \wedge B\right)\right) \\ &= (((A \uparrow B) \uparrow B) \uparrow (A \uparrow (A \uparrow B))) \ . \\ &\quad \boxed{\text{Expressão 2: 4 NANDs e 3 } t_p}\end{aligned}$$

$$
\begin{aligned}
(A \veebar B) &= (A \vee B) \wedge (\neg(A) \vee \neg(B)) \\
&= ((A \uparrow A) \uparrow (B \uparrow B)) \wedge (A \uparrow B) \\
&= \neg\left(\neg\left(((A \uparrow A) \uparrow (B \uparrow B)) \wedge (A \uparrow B)\right)\right) \\
&= \neg\left(((A \uparrow A) \uparrow (B \uparrow B)) \uparrow (A \uparrow B)\right) \\
&= (((A \uparrow A) \uparrow (B \uparrow B)) \uparrow (A \uparrow B)) \uparrow (((A \uparrow A) \uparrow (B \uparrow B)) \uparrow (A \uparrow B)) \ .
\end{aligned}
$$

Expressão 3: 6 NANDs e 4 t_p

A.3 Relações com o operador NOR

A seguir, são apresentados alguns exemplos de relações baseadas no operador NOR.

A.3.1 Operador NOT

$$\neg(A) = \neg(A \vee A) = (A \downarrow A) \ .$$

$$A = \neg\left(\neg(A)\right) = (\neg(A) \downarrow \neg(A)) = (A \downarrow A) \downarrow (A \downarrow A) \ .$$

$$A = \neg\left(\neg(A)\right) = \neg\left(A \downarrow A\right) = (A \downarrow A) \downarrow (A \downarrow A) \ .$$

A.3.2 Operador AND

$$(A \wedge B) = \neg\left(\neg(A \wedge B)\right) = \neg\left(\neg(A) \vee \neg(B)\right) = (\neg(A) \downarrow \neg(B)) = (A \downarrow A) \downarrow (B \downarrow B) \ .$$

$$
\begin{aligned}
(\neg(A) \wedge B) &= ((A \downarrow A) \downarrow (A \downarrow A)) \downarrow (B \downarrow B) \\
&= \neg\left(\neg\left(\neg(A) \wedge B\right)\right) = \neg\left(A \vee \neg(B)\right) = (A \downarrow \neg(B)) = (A \downarrow (B \downarrow B)) \ .
\end{aligned}
$$

$$
\begin{aligned}
(A \wedge \neg(B)) &= (A \downarrow A) \downarrow ((B \downarrow B) \downarrow (B \downarrow B)) \\
&= \neg\left(\neg\left(A \wedge \neg(B)\right)\right) = \neg\left(\neg(A) \vee B\right) = (\neg(A) \downarrow B) = ((A \downarrow A) \downarrow B) \ .
\end{aligned}
$$

$$
\begin{aligned}
(\neg(A) \wedge \neg(B)) &= ((A \downarrow A) \downarrow (A \downarrow A)) \downarrow (B \downarrow B) \downarrow (B \downarrow B) \\
&= \neg\left(\neg\left(\neg(A) \wedge \neg(B)\right)\right) = \neg\left(A \vee B\right) = (A \downarrow B) \ .
\end{aligned}
$$

A.3.3 Operador OR

$$(A \vee B) = \neg\left(\neg(A \vee B)\right) = \neg\left(A \downarrow B\right) = (A \downarrow B) \downarrow (A \downarrow B) \ .$$
$$(\neg(A) \vee B) = ((A \downarrow A) \downarrow B) \downarrow ((A \downarrow A) \downarrow B) \ .$$
$$(A \vee \neg(B)) = (A \downarrow (B \downarrow B)) \uparrow (A \downarrow (B \downarrow B)) \ .$$
$$(\neg(A) \vee \neg(B)) = ((A \downarrow A) \downarrow (B \downarrow B)) \downarrow ((A \downarrow A) \downarrow (B \downarrow B)) \ .$$

A.3.4 Operador XOR

$$\begin{aligned}
(A \veebar B) &= (\neg(A) \wedge B) \vee (A \wedge \neg(B)) \\
&= (A \downarrow (B \downarrow B)) \vee ((A \downarrow A) \downarrow B) \\
&= \neg\left(\neg\left((A \downarrow (B \downarrow B)) \vee ((A \downarrow A) \downarrow B)\right)\right) \\
&= \neg\left(((A \downarrow (B \downarrow B)) \downarrow ((A \downarrow A) \downarrow B))\right) \\
&= (((A \downarrow (B \downarrow B)) \downarrow ((A \downarrow A) \downarrow B))) \downarrow (((A \downarrow (B \downarrow B)) \downarrow ((A \downarrow A) \downarrow B))) \ . \\
&\quad \boxed{\text{Expressão 4: 6 NORs e 4 } t_p}
\end{aligned}$$

$$\begin{aligned}
&= \neg\left(\neg\left(\neg(A) \wedge B\right)\right) \vee \neg\left(\neg\left(A \wedge \neg(B)\right)\right) \\
&= \neg\left(\neg\left(\neg(A) \wedge B\right) \vee (0)\right) \vee \neg\left(\neg\left(A \wedge \neg(B)\right) \vee (0)\right) \\
&= \neg\left(\neg\left(\neg(A) \wedge B\right) \vee (A \wedge \neg(A))\right) \vee \neg\left(\neg\left(A \wedge \neg(B)\right) \vee (B \wedge \neg(B))\right) \\
&= \neg\left(\neg\left(\neg(A) \wedge (A \vee B)\right)\right) \vee \neg\left(\neg\left(\neg(B) \wedge (A \vee B)\right)\right) \\
&= \neg\left(A \vee \neg\left(A \vee B\right)\right) \vee \neg\left(B \vee \neg\left(A \vee B\right)\right) \\
&= (A \downarrow (A \downarrow B)) \vee (B \downarrow (A \downarrow B)) \\
&= \neg\left(\neg\left((A \downarrow (A \downarrow B)) \vee (B \downarrow (A \downarrow B))\right)\right) \\
&= \neg\left((A \downarrow (A \downarrow B)) \downarrow (B \downarrow (A \downarrow B))\right) \\
&= ((A \downarrow (A \downarrow B)) \downarrow (B \downarrow (A \downarrow B))) \downarrow ((A \downarrow (A \downarrow B)) \downarrow (B \downarrow (A \downarrow B))) \ . \\
&\quad \boxed{\text{Expressão 5: 5 NORs e 4 } t_p}
\end{aligned}$$

$$\begin{aligned}
&= \neg\left(\neg\left(\neg(A) \wedge B\right)\right) \vee \neg\left(\neg\left(A \wedge \neg(B)\right)\right) \\
&= \neg\left(A \vee \neg(B)\right) \vee \neg\left(\neg(A) \vee B\right) \\
&= \neg\left(\left(A \vee \neg(B)\right) \wedge (1)\right) \vee \neg\left(\left(\neg(A) \vee B\right) \wedge (1)\right) \\
&= \neg\left(\left(A \vee \neg(B)\right) \wedge \left(A \vee \neg(A)\right)\right) \vee \neg\left(\left(\neg(A) \vee B\right) \wedge \left(B \vee \neg(B)\right)\right) \\
&= \neg\left(A \vee \left(\neg(A) \wedge \neg(B)\right)\right) \vee \neg\left(B \vee \left(\neg(A) \wedge \neg(B)\right)\right) \\
&= \neg\left(A \vee \neg\left(A \vee B\right)\right) \vee \neg\left(B \vee \neg\left(A \vee B\right)\right) \\
&= (A \downarrow (A \downarrow B)) \vee (B \downarrow (A \downarrow B)) \\
&= \neg\left(\neg\left((A \downarrow (A \downarrow B)) \vee (B \downarrow (A \downarrow B))\right)\right) \\
&= \neg\left((A \downarrow (A \downarrow B)) \downarrow (B \downarrow (A \downarrow B))\right) \\
&= ((A \downarrow (A \downarrow B)) \downarrow (B \downarrow (A \downarrow B))) \downarrow ((A \downarrow (A \downarrow B)) \downarrow (B \downarrow (A \downarrow B))) \ . \\
&\quad \boxed{\text{Expressão 5: 5 NORs e 4 } t_p}
\end{aligned}$$

$$
\begin{aligned}
(A \veebar B) &= (A \vee B) \wedge (\neg(A) \vee \neg(B)) \\
&= \neg\left(\neg\left((A \vee B) \wedge (\neg(A) \vee \neg(B))\right)\right) \\
&= \neg\left(\neg\left(A \vee B\right) \vee \neg\left(\neg(A) \vee \neg(B)\right)\right) \\
&= \left(\neg\left(A \vee B\right) \downarrow \neg\left(\neg(A) \vee \neg(B)\right)\right) \\
&= (A \downarrow B) \downarrow (\neg(A) \downarrow \neg(B)) \\
&= (A \downarrow B) \downarrow ((A \downarrow A) \downarrow (B \downarrow B)) \ .
\end{aligned}
$$

Expressão 6: 5 NORs e 3 t_p

Apêndice B

Noções básicas sobre implementação de funções lógicas

B.1 Introdução

A implementação de funções lógicas é um assunto que possui grande extensão e é rico em detalhes. Ainda que tal conteúdo não seja o objetivo do presente documento, é importante que algumas noções básicas sejam abordadas. Este capítulo tem por objetivo apresentar, de forma breve e superficial, alguns itens comumente encontrados em implementações típicas. Inicialmente, são abordados alguns conceitos básicos, presentes na implementação de circuitos lógicos em circuitos integrados. Em seguida, as famílias lógicas são comentadas. Finalmente, é discutido um modelo de chaves para a implementação de funções da lógica binária, com processamento de tensão.

B.2 Conceitos básicos

Circuito integrado monolítico é um termo empregado para designar a construção de um circuito eletrônico em uma única porção de material semicondutor.

Os elementos básicos usados em sistemas eletro-eletrônicos que implementam circuitos lógicos baseados em circuitos integrados são: elementos externos ao circuito integrado (fontes de alimentação e geradores de sinais) e elementos internos ao circuito integrado (transistores).

As fontes de alimentação são elementos de transdução, que transformam grandezas de outros sistemas físicos em uma grandeza elétrica de tensão ou de corrente, de valor fixo. Elas podem ser também elementos de transformação de grandezas elétricas de tensão ou de corrente de valor variável em uma grandeza elétrica de tensão ou de corrente de valor fixo.

Na implementação de um determinado sistema lógico matemático por um sistema elétrico físico, os valores lógicos fixos devem ser associados a valores elétricos fixos, fornecidos pelas fontes (na alimentação ou nos sinais de excitação) e pelas saídas dos circuitos integrados. No caso da lógica binária, devem ser implementados os valores lógicos fixos F (*False*) e T (*True*). Considerando-se um processamento de tensão, podem ser associados a eles os seguintes valores elétricos fixos: $+|V|$ e 0, 0 e $-|V|$ ou $+|V|$ e $-|V|$. Os dois tipos básicos de associação entre valores de tensão e valores lógicos binários são apresentados na Tabela B.1.

Os transistores são dispositivos eletrônicos de três terminais. Um dos terminais é usado para realizar um acionamento que produzirá um efeito sobre os outros dois terminais. Observam-se três modos de operação, considerando-se o efeito causado nos dois terminais controlados: circuito aberto, curto-circuito e fonte de corrente controlada (por tensão ou por corrente).

Tensões			Associação 1	Associação 2
$+\|V\|$	0	$+\|V\|$	T	F
0	$-\|V\|$	$-\|V\|$	F	T

Tabela B.1: Tipos básicos de associação entre valores de tensão e valores lógicos binários.

As implementações clássicas de funções lógicas binárias, tem, como modelo, o uso de dispositivos que possuem dois estados. Assim, em cada tipo de implementação, normalmente, são utilizados apenas dois dos três modos de operação dos transistores.

B.3 Famílias lógicas

Na implementação dos circuitos digitais, podem ser utilizados diferentes dispositivos físicos, associados a diversas técnicas de composição e a diversos modos de operação.

Se determinados dispositivos físicos são conectados de uma forma específica, são operados de um modo particular e apresentam parâmetros lógicos e físicos que obedecem a padrões estabelecidos, então diz-se que o circuito final pertence a uma Família Lógica.

Historicamente, várias famílias lógicas foram propostas, empregando-se dispositivos eletrônicos e processamento de tensão. Usando transistor bipolar de junção (*Bipolar Junction Transistor* ou BJT), podem ser citadas: *Resistor-Transistor Logic* (RTL), *Direct-Coupled Transistor Logic* (DCTL), *Diode-Transistor Logic* (DTL), *Transistor-Transistor Logic* (TTL), *Emitter-Coupled Logic* (ECL) e *Integrated Injection Logic* (IIL ou I2L). Usando transistor de efeito de campo (*Field Effect Transistor* ou FET) do tipo *Metal-Oxide-Semiconductor* (MOS), denominado de MOSFET, podem ser citadas: lógica com transistor do tipo N (NMOS), lógica com transistor do tipo P (PMOS) e lógica com arranjo complementar de ambos os tipos de transistor (CMOS).

As famílias lógicas mais comumente utilizadas são a TTL e a CMOS.

Devido ao seu baixo consumo de energia e à sua baixa ocupação de espaço, os circuitos lógicos CMOS são largamente utilizados em implementações de circuitos integrados de alta densidade.

B.4 Modelo de chaves

B.4.1 Conceitos básicos

Nas implementações que utilizam processamento de tensão, os transistores são usados, basicamente, como chaves de passagem, operando nos modos de circuito aberto e curto-circuito entre os dois terminais controlados. A fontes de alimentação são utilizadas, ao mesmo tempo, como sinal de acionamento e como resultado da avaliação da função lógica. Neste caso, os sinais de entrada do circuito, que representam os parâmetros dos quais depende a função lógica, são usados apenas para acionar transistores operando como chaves, cujo acionamento pode ainda acionar outros transistores. Uma vez selecionado um determinado caminho formado por chaves fechadas, uma das duas fontes de alimentação é conectada com a saída, o que irá representar o valor da função lógica implementada para um dado conjunto de valores dos seus parâmetros.

Deve-se notar que os circuitos em si não implementam funções lógicas. Internamente, eles apenas realizam operações elétricas. Porém, através de associações de significados pertinentes, pode-se, externamente, interpretar o seu funcionamento como a avaliação de uma função lógica.

Pode-se propor um modelo de chaves para a implementação de funções da lógica binária, com processamento de tensão, utilizando-se as chaves apresentadas na Figura B.1, denominadas de chave-N e chave-P. O terminal A é o ponto de acionamento, o terminal R é o ponto de referência e o terminal F é o ponto para onde irá fluir o valor da referência, quando a chave for fechada. Enquanto a chave estiver aberta, o valor do terminal F permanece indefinido.

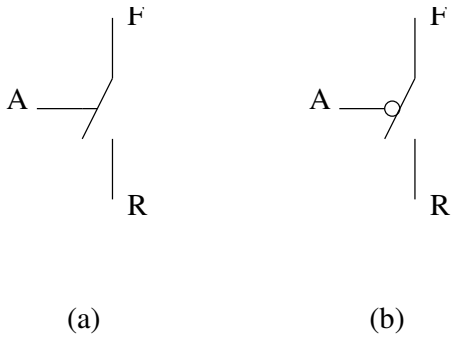


Figura B.1: Chaves simples ou *Single-Pole Single-Throw* (SPST). (a) Chave-N. (b) Chave-P.

Será adotado o seguinte padrão de acionamento para as chaves N e P. Supondo-se que $V_{AR} = V_A - V_R$ é a diferença de potencial entre os terminais A e R, a chave-N será fechada quando $V_{AR} > 0$ ou $V_A > V_R$ e será aberta quando $V_{AR} \leq 0$ ou $V_A \leq V_R$. De forma contrária, a chave-P será fechada quando $V_{AR} < 0$ ou $V_A < V_R$ e será aberta quando $V_{AR} \geq 0$ ou $V_A \geq V_R$.

Supondo-se que $V_R = -|V_L|$ na chave-N e que $V_R = +|V_H|$ na chave-P, pode-se dizer, do ponto de vista elétrico, que, quando $A = +|V_H|$, a chave-N será fechada e a chave-P será aberta. De forma contrária, quando $A = -|V_L|$, a chave-N será aberta e a chave-P será fechada.

Definindo-se que $F = -|V_L|$ e que $T = +|V_H|$, pode-se dizer, do ponto de vista lógico, que, quando $A = T$, a chave-N será fechada e a chave-P será aberta. De forma contrária, quando $A = F$, a chave-N será aberta e a chave-P será fechada.

A partir do padrão de acionamento definido, conclui-se que as chaves N e P são complementares em relação ao seu acionamento. Dito de outra forma, pode-se interpretar o comportamento de uma chave-P acionada por uma variável A como o de uma chave-N acionada por uma variável ¬A. Isso é ilustrado na Figura B.2, onde o comportamento de uma chave-P é interpretada como a associação de um bloco inversor lógico (NOT) com uma chave-N. Por essa razão, a chave-P é representada como a chave-N acrescida de um círculo de negação.

Quanto ao terminal F, quando a chave está fechada, tem-se $V_F = V_R$. Por outro lado, quando a chave está aberta, o valor de V_F é indefinido e costuma-se dizer que o ponto F está *flutuando*.

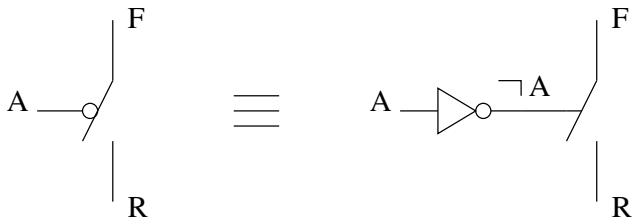


Figura B.2: Modelo comportamental para a relação de complementariedade no acionamento, entre as chaves N e P.

B.4.2 Arranjos série e paralelo de chaves

A Figura B.3 mostra um arranjo série de chaves. Para as chaves-N, se $V_{A_1} > V_{R_1}$ e $V_{A_2} > V_{R_1}$, então ambas as chaves estarão fechadas e $V_{F_2} = V_{R_2} = V_{F_1} = V_{R_1}$. Caso contrário, uma das chaves estará aberta, ou ambas, e o valor de V_F é indefinido. Para as chaves-P, se $V_{A_1} < V_{R_1}$ e $V_{A_2} < V_{R_1}$, então ambas as chaves estarão fechadas e $V_{F_2} = V_{R_2} = V_{F_1} = V_{R_1}$. Caso contrário, uma das chaves estará aberta, ou ambas, e o valor de V_F é indefinido. Esse tipo de arranjo é a base para modelar uma implementação da função lógica AND.

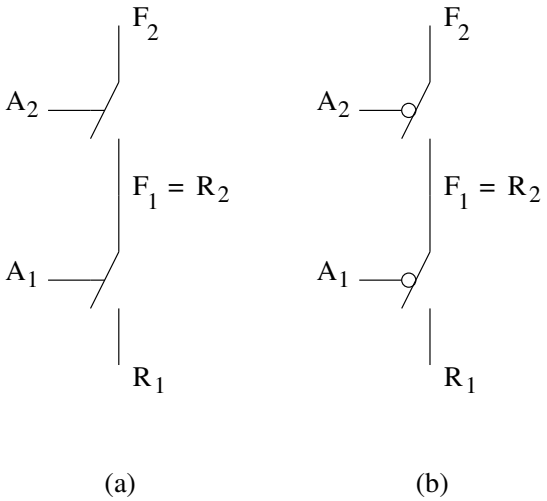


Figura B.3: Arranjo série de chaves simples. (a) Chave-N. (b) Chave-P.

A Figura B.4 mostra um arranjo série de chaves. Para as chaves-N, se $V_{A_1} > V_{R_1}$ ou $V_{A_2} > V_{R_2}$, ou ambos, então uma das chaves estará fechada, ou ambas, e $V_{F_2} = V_{R_2} = V_{F_1} = V_{R_1}$. Caso contrário, ambas as chaves estarão abertas, e o valor de V_F é indefinido. Para as chaves-P, se $V_{A_1} < V_{R_1}$ ou $V_{A_2} < V_{R_2}$, ou ambos, então uma das chaves estará fechada, ou ambas, e $V_{F_2} = V_{R_2} = V_{F_1} = V_{R_1}$. Caso contrário, ambas as chaves estarão abertas, e o valor de V_F é indefinido. Esse tipo de arranjo é a base para modelar uma implementação da função lógica OR.

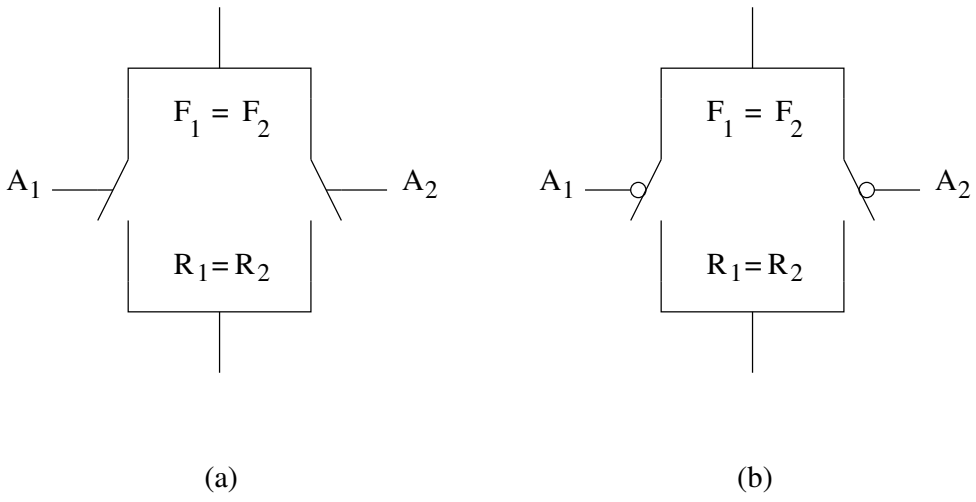


Figura B.4: Arranjo paralelo de chaves simples. (a) Chave-N. (b) Chave-P.

B.4.3 Modelo de chaves para a função NOT

Nas Figuras B.5 e B.6, são apresentados arranjos de chaves N e P, com fontes de alimentação. Baseado na operação das chaves, pode-se comprovar o comportamento elétrico apresentado na Tabela B.2. Estabelecendo-se a associação $+|V| = T$ e $-|V| = F$, pode-se identificar o comportamento lógico apresentado na Tabela B.3. Esses tipos de arranjo modelam uma implementação da função lógica NOT.

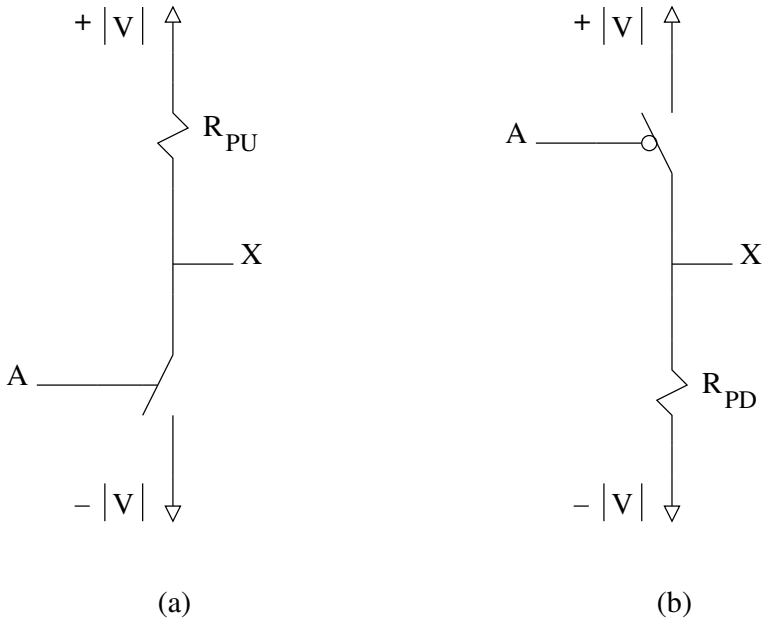


Figura B.5: Modelo de chaves simples para uma implementação da função lógica NOT, onde: (a) chave N e (b) chave P.

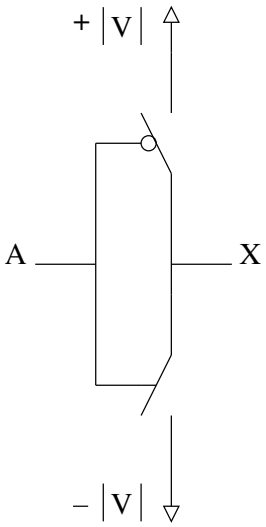


Figura B.6: Modelo de chaves complementares para uma implementação da função lógica NOT.

V_A	$V_X = \mathrm{f}(V_A)$
$-\lvert V\rvert$	$+\lvert V\rvert$
$+\lvert V\rvert$	$-\lvert V\rvert$

Tabela B.2: Comportamento elétrico do modelo de chaves NOT.

A	X = f(A)
F	T
T	F

Tabela B.3: Comportamento lógico do modelo de chaves NOT.

B.4.4 Modelo de chaves para a função NAND

Nas Figuras B.7 e B.8, são apresentados arranjos de chaves N e P, com fontes de alimentação. Baseado na operação das chaves, pode-se comprovar o comportamento elétrico apresentado na Tabela B.4. Estabelecendo-se a associação $+|V| = T$ e $-|V| = F$, pode-se identificar o comportamento lógico apresentado na Tabela B.5. Esses tipos de arranjo modelam uma implementação da função lógica NAND.

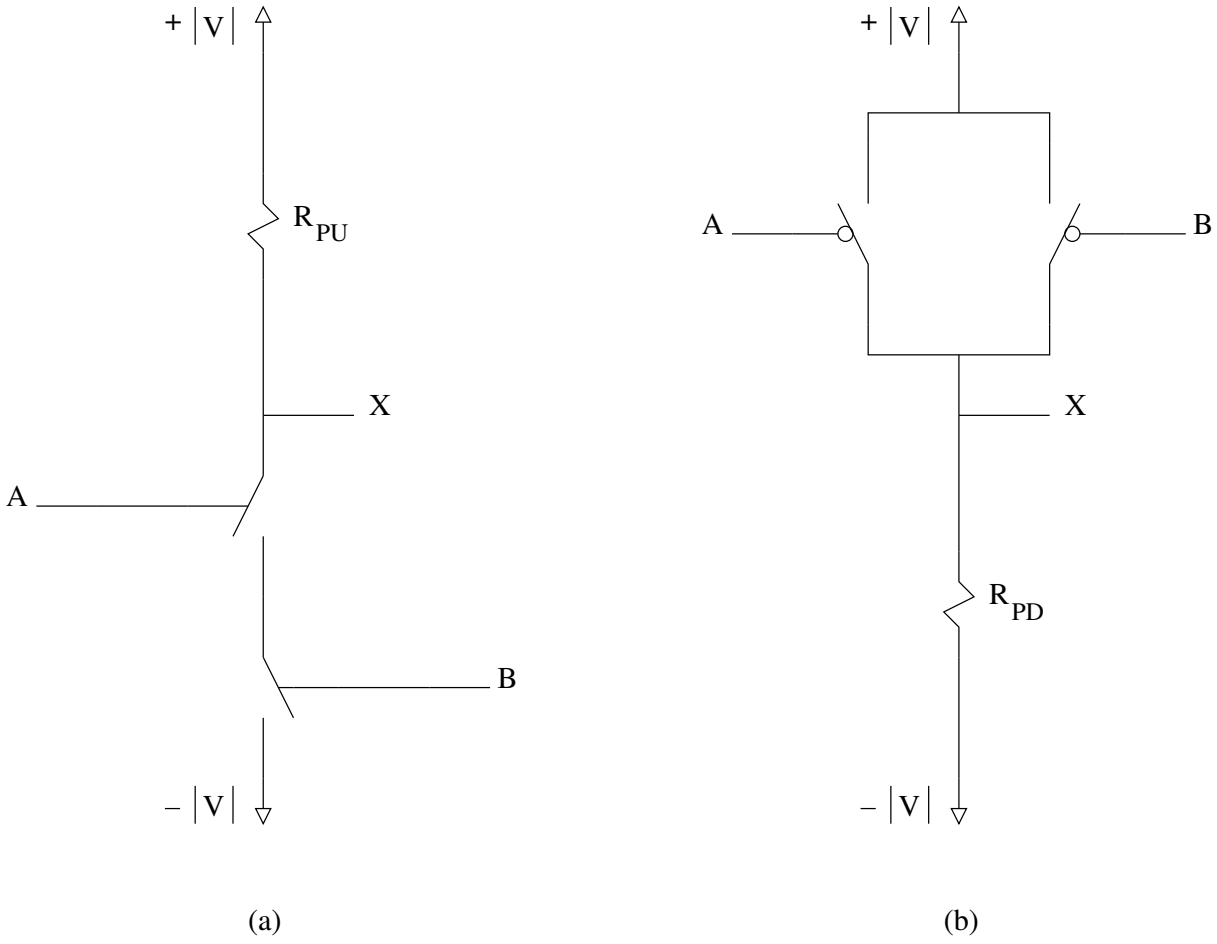


Figura B.7: Modelo de chaves simples para uma implementação da função lógica NAND, onde: (a) chave N e (b) chave P.

V_A	V_B	$V_X = \mathrm{f}(V_A, V_B)$
$-\|V\|$	$-\|V\|$	$+\|V\|$
$-\|V\|$	$+\|V\|$	$+\|V\|$
$+\|V\|$	$-\|V\|$	$+\|V\|$
$+\|V\|$	$+\|V\|$	$-\|V\|$

Tabela B.4: Comportamento elétrico do modelo de chaves NAND.

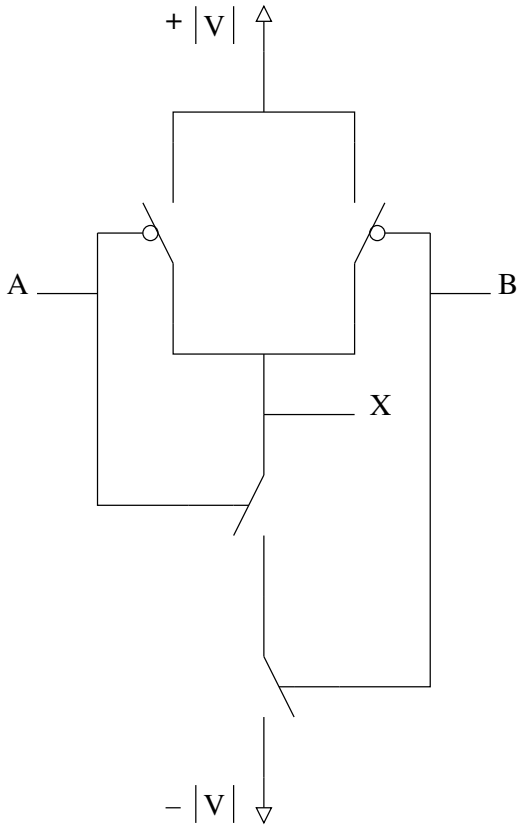


Figura B.8: Modelo de chaves complementares para uma implementação da função lógica NAND.

A	B	X = f(A, B)
F	F	T
F	T	T
T	F	T
T	T	F

Tabela B.5: Comportamento lógico do modelo de chaves NAND.

B.4.5 Modelo de chaves para a função NOR

Nas Figuras B.9 e B.10, são apresentados arranjos de chaves N e P, com fontes de alimentação. Baseado na operação das chaves, pode-se comprovar o comportamento elétrico apresentado na Tabela B.6. Estabelecendo-se a associação $+|V| = T$ e $-|V| = F$, pode-se identificar o comportamento lógico apresentado na Tabela B.7. Esses tipos de arranjo modelam uma implementação da função lógica NAND.

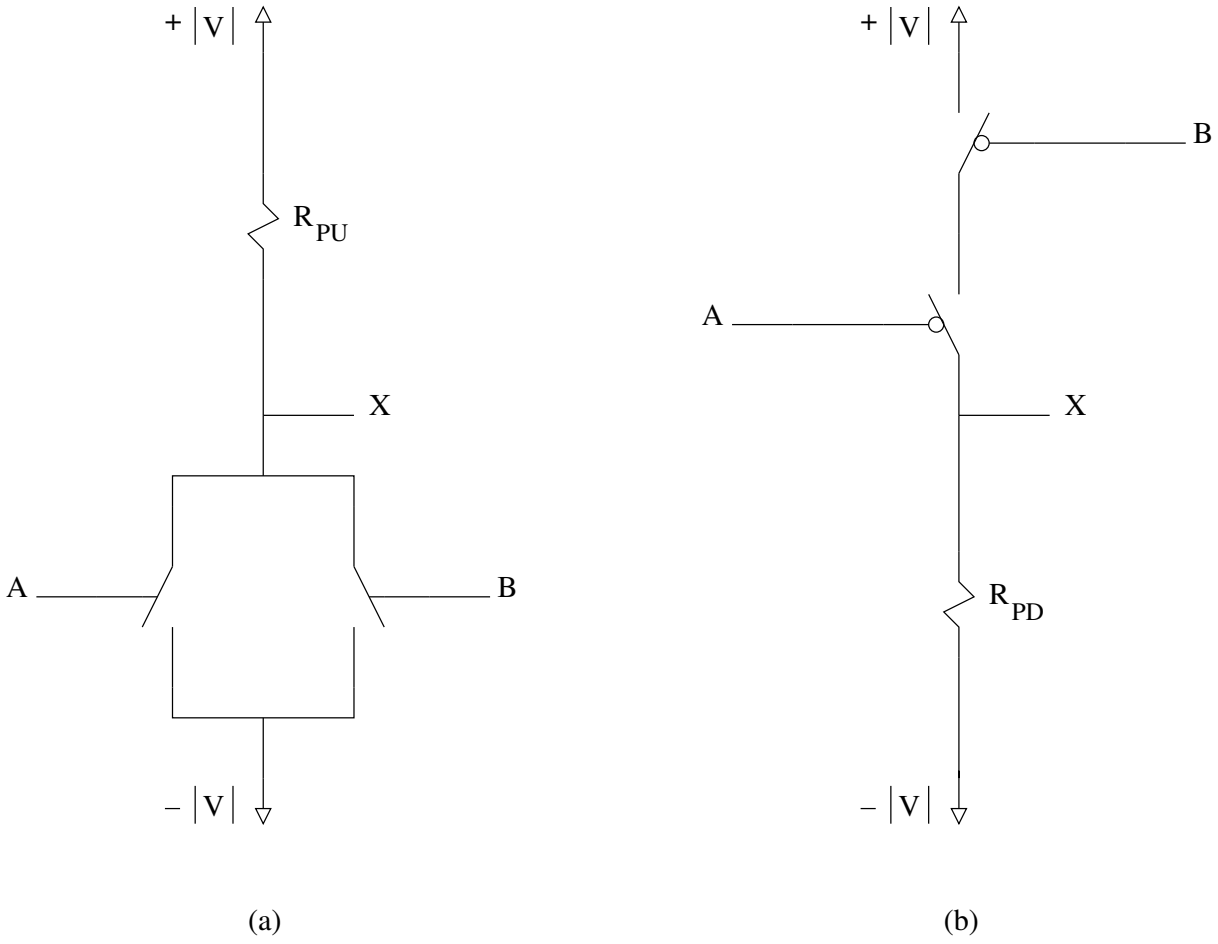


Figura B.9: Modelo de chaves simples para uma implementação da função lógica NOR, onde: (a) chave N e (b) chave P.

V_A	V_B	$V_X = \mathrm{f}(V_A, V_B)$
$-\|V\|$	$-\|V\|$	$+\|V\|$
$-\|V\|$	$+\|V\|$	$-\|V\|$
$+\|V\|$	$-\|V\|$	$-\|V\|$
$+\|V\|$	$+\|V\|$	$-\|V\|$

Tabela B.6: Comportamento elétrico do modelo de chaves NOR.

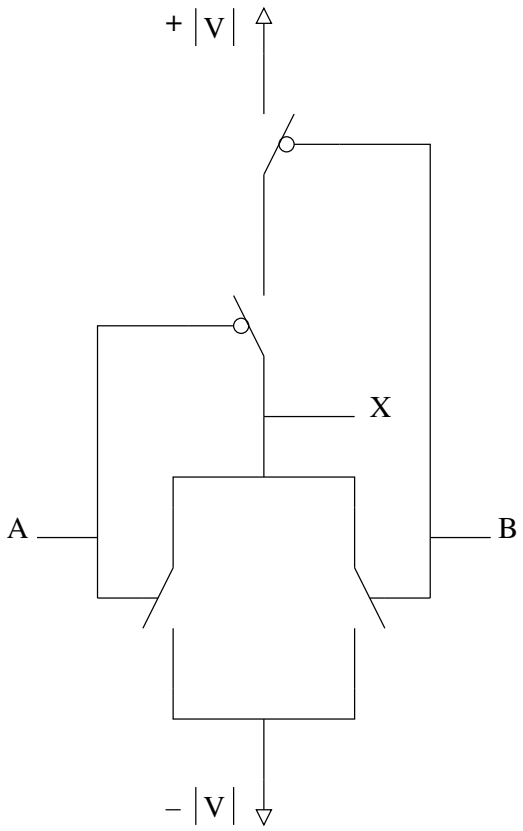


Figura B.10: Modelo de chaves complementares para uma implementação da função lógica NOR.

A	B	X = f(A, B)
F	F	T
F	T	F
T	F	F
T	T	F

Tabela B.7: Comportamento lógico do modelo de chaves NOR.

B.4.6 Modelo de chaves para arranjos AOI e OAI

Além dos conectivos lógicos básicos (NOT, NAND e NOR), dois tipos de arranjos são largamente utilizados na implementação de funções lógicas, os quais são denominados de AO (AND-OR) e de OA (OR-AND). Como foi visto acima, o modelo de chaves complementares apresenta uma inversão intrínseca. Assim, são naturalmente implementados os arranjos AOI (AND-OR-INVERTER) e OAI (OR-AND-INVERTER), descritos a seguir.

Arranjos AOI

Os arranjos AOI (AND-OR-INVERTER) são funções lógicas onde as variáveis (e suas negações) são inicialmente combinadas por conectivos AND. Em seguida, tais termos são combinados por conectivos OR. Finalmente, toda a função sofre uma inversão através da operação NOT.

Exemplos de arranjos AOI são

$$f(A,B) = \neg(\ (A\ \wedge\ \neg B)\ \vee\ (\neg A\ \wedge\ B)\ )\ ,$$

$$\begin{aligned} f(A,B,C)\ \ =\ \ & \neg(\ (\neg A\ \wedge\ \neg B\ \wedge\ C)\ \vee\ (\neg A\ \wedge\ B\ \wedge\ \neg C)\ \vee \\ & (A\ \wedge\ \neg B\ \wedge\ \neg C)\ \vee\ (A\ \wedge\ B\ \wedge\ C)\ \vee\ )\ , \end{aligned}$$

e

$$\begin{aligned} f(A,B,C,D)\ \ =\ \ & \neg(\ (\neg A\ \wedge\ B\ \wedge\ \neg C\ \wedge\ D)\ \vee\ (\neg A\ \wedge\ B\ \wedge\ C\ \wedge\ \neg D)\ \vee \\ & (A\ \wedge\ \neg B\ \wedge\ C\ \wedge\ \neg D)\ \vee\ (A\ \wedge\ \neg B\ \wedge\ C\ \wedge\ D)\ )\ . \end{aligned}$$

O arranjo AOI definido por $f(A,B,C,D) = \neg(\ (A \wedge B) \vee (C \wedge D)\ )$ pode ser implementado pelo modelo de chaves complementares mostrado na Figura B.11.

Arranjos OAI

Os arranjos OAI (OR-AND-INVERTER) são funções lógicas onde as variáveis (e suas negações) são inicialmente combinadas por conectivos OR. Em seguida, tais termos são combinados por conectivos AND. Finalmente, toda a função sofre uma inversão através da operação NOT.

Exemplos de arranjos OAI são

$$f(A,B) = \neg(\ (A\ \vee\ \neg B)\ \wedge\ (\neg A\ \vee\ B)\ )\ ,$$

$$\begin{aligned} f(A,B,C)\ \ =\ \ & \neg(\ (\neg A\ \vee\ \neg B\ \vee\ C)\ \wedge\ (\neg A\ \vee\ B\ \vee\ \neg C)\ \wedge \\ & (A\ \vee\ \neg B\ \vee\ \neg C)\ \wedge\ (A\ \vee\ B\ \vee\ C)\ \wedge\ )\ , \end{aligned}$$

e

$$\begin{aligned} f(A,B,C,D)\ \ =\ \ & \neg(\ (\neg A\ \vee\ B\ \vee\ \neg C\ \vee\ D)\ \wedge\ (\neg A\ \vee\ B\ \vee\ C\ \vee\ \neg D)\ \wedge \\ & (A\ \vee\ \neg B\ \vee\ C\ \vee\ \neg D)\ \wedge\ (A\ \vee\ \neg B\ \vee\ C\ \vee\ D)\ )\ . \end{aligned}$$

O arranjo OAI definido por $f(A,B,C,D) = \neg(\ (A \vee B) \wedge (C \vee D)\ )$ pode ser implementado pelo modelo de chaves complementares mostrado na Figura B.12.

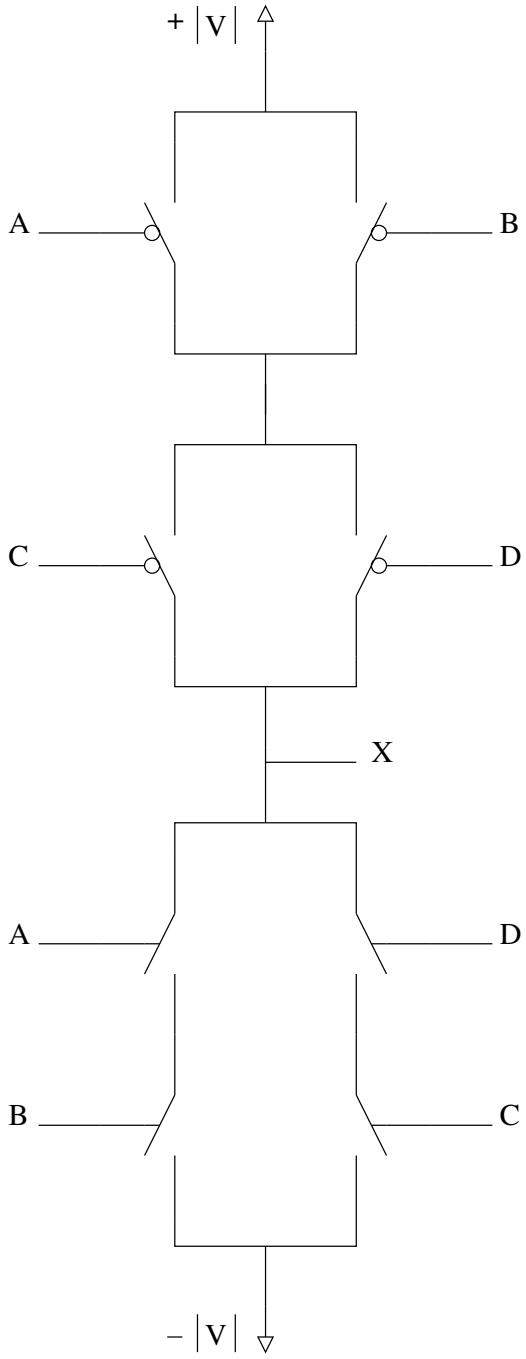


Figura B.11: Modelo de chaves complementares para uma implementação do arranjo AOI definido por $f(A, B, C, D) = \neg(\ (A\ \wedge\ B)\ \vee\ (C\ \wedge\ D)\ )$.

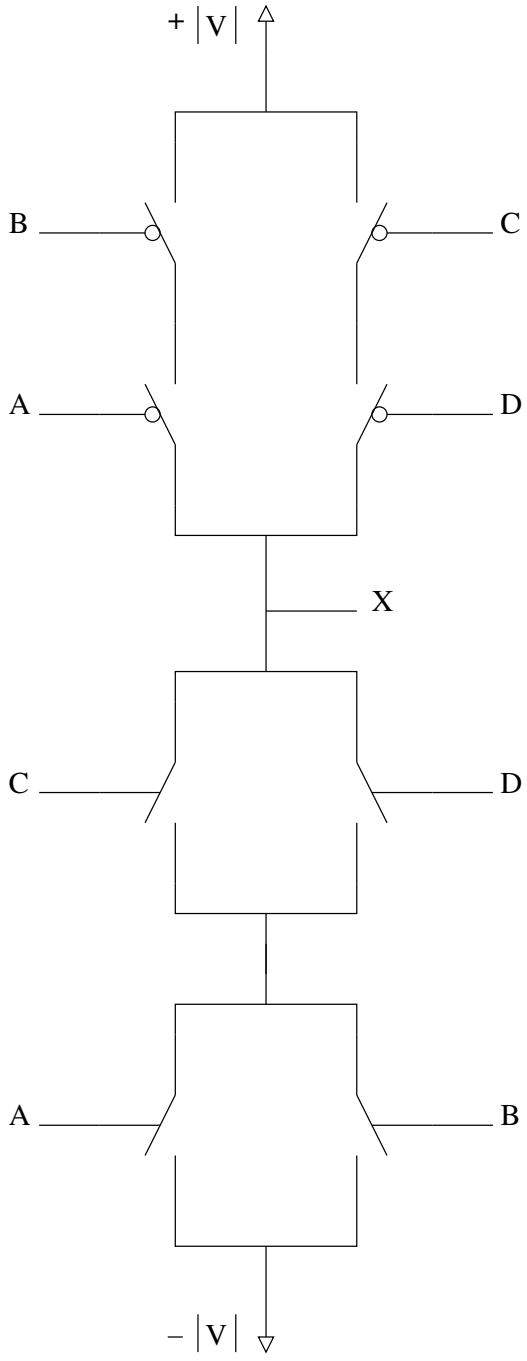


Figura B.12: Modelo de chaves complementares para uma implementação do arranjo OAI definido por $f(A, B, C, D) = \neg(\ (A\ \vee\ B)\ \wedge\ (C\ \vee\ D)\ )$.

B.4.7 Modelo de chaves complementares genérico

Do ponto de vista funcional, os conectivos lógicos anteriormente modelados (NOT, NAND e NOR) são suficientes para gerar qualquer função lógica binária desejada. Mesmo assim, cabe investigar tanto a possibilidade de síntese de uma função genérica quanto a existência de uma lei de formação para tal.

Observando-se os modelos apresentados para as funções NOT, NAND e NOR, percebe-se que os mesmos compartilham certas características.

Uma vez que as chaves N e P sofrem acionamento complementar, os blocos de chaves N e P herdam o mesmo comportamento. Esse comportamento complementar garante a consistência da operação, pois quando um bloco proporciona um caminho de condução ao longo de sua estrutura e força um valor lógico na saída, o outro bloco gera uma obstrução e provoca uma indeterminação. O comportamento complementar dos blocos de chaves N e P nos modelos NOT, NAND e NOR, pode ser observado, respectivamente, nas Tabelas B.8, B.9 e B.10, onde o valor lógico I significa uma indeterminação.

A	X(A)
F	I
T	F

(a)

A	X(A)
F	T
T	I

(b)

A	X(A)
F	T
T	F

(c)

Tabela B.8: Comparação do comportamento dos blocos de chaves N e P no modelo de chaves da função lógica NOT. (a) Chave-N. (b) Chave-P. (c) Arranjo complementar.

A	B	X(A,B)
F	F	I
F	T	I
T	F	I
T	T	F

(a)

A	B	X(A,B)
F	F	T
F	T	T
T	F	T
T	T	I

(b)

A	B	X(A,B)
F	F	T
F	T	T
T	F	T
T	T	F

(c)

Tabela B.9: Comparação do comportamento dos blocos de chaves N e P no modelo de chaves da função lógica NAND. (a) Chave-N. (b) Chave-P. (c) Arranjo complementar.

Pode-se constatar que o bloco de chaves-N possui uma ligação direta com a função desejada. Porém, ele apresenta uma inversão intrínseca à estrutura. No caso da função NOT, quando a variável de entrada vale T, o bloco conduz, realizando a passagem de um valor F para a saída. No caso da função NAND, quando as variáveis de entrada valem ambas T, o bloco conduz, realizando uma operação AND, mas passando um valor F para a saída. No caso da função NOR, quando uma das variáveis de entrada vale T, ou ambas, o bloco conduz, realizando uma operação OR, mas passando um valor F para a saída. Logo, pode-se dizer que, definindo-se a função desejada como $\neg X(A, B, C, \cdots)$ o bloco de chaves-N realiza o seu complemento: $X(A, B, C, \cdots)$.

Por sua vez, o bloco de chaves-P deve colaborar na síntese da função desejada de uma forma funcionalmente complementar ao bloco de chaves-N, ao mesmo tempo que seu acionamento deve ser provocado pelo complemento das variáveis de entrada que acionam o bloco de chaves-N.

A	B	X(A,B)
F	F	I
F	T	F
T	F	F
T	T	F

(a)

A	B	X(A,B)
F	F	T
F	T	I
T	F	I
T	T	I

(b)

A	B	X(A,B)
F	F	T
F	T	F
T	F	F
T	T	F

(c)

Tabela B.10: Comparação do comportamento dos blocos de chaves N e P no modelo de chaves da função lógica NOR. (a) Chave-N. (b) Chave-P. (c) Arranjo complementar.

Isso é equivalente à aplicação do Teorema de De Morgan ao complemento da funcionalidade do bloco de chaves-N. Porém, nessa estrutura, a função do bloco de chaves-N já é o complemento da função desejada. Logo, a síntese do bloco de chaves-P é equivalente à aplicação do Teorema de De Morgan sobre a função desejada. Além disso, o acionamento das chaves-P já é baseado no complemento das variáveis que acionam o bloco de chaves-N. Portanto, após aplicar o Teorema de De Morgan sobre a função desejada, deve-se desprezar a negação das variáveis de entrada na equação final. Isso pode ser facilmente verificado nos modelos de chaves para as funções NOT, NAND e NOR.

Resumindo, a regra geral para a síntese de uma função lógica binária qualquer, usando chaves complementares, através de um arranjo complementar, pode ser definida da seguinte forma:

- Definir a função desejada como $\neg X(A, B, C, \cdots)$.
- Sintetizar, através de um arranjo de chaves-N, a função $X(A, B, C, \cdots)$.
- Aplicar o Teorema de De Morgan sobre $\neg X(A, B, C, \cdots)$. Desconsiderar a negação das variáveis de entrada. Sintetizar a equação final através de um arranjo de chaves-P.

B.4.8 Saída em *Tri-State* (3S)

Em algumas aplicações, pode ser necessário que o estágio de saída de um circuito digital isole a parte interna da parte externa do circuito. Eletricamente, essa desconexão é interpretada como um circuito aberto ou uma ligação de alta impedância, denominada de *High-Z*. Do ponto de vista lógico, quando a conexão é estabelecida, a saída assume os estados F ou T. Porém, quando a conexão é interrompida, e a saída entra em modo *High-Z*, define-se um terceiro estado. Por essa razão, diz-se que o circuito possui uma saída do tipo *Tri-State* ou 3S. A Figura B.13 apresenta um modelo de chaves complementares para uma implementação de função lógica NOT com saída em *Tri-State* (3S). Conforme indicado, o controle de conexão da saída é realizado por um sinal de habilitação (*enable* ou EN).

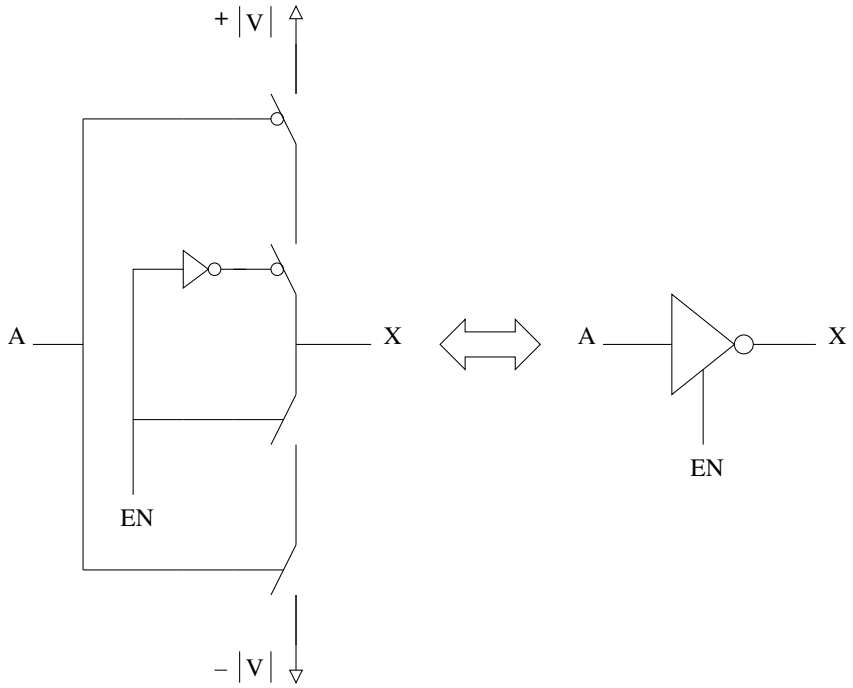


Figura B.13: Modelo de chaves complementares para uma implementação de função lógica NOT com saída em *Tri-State* (3S).

Apêndice C

Tópicos sobre divisão de números inteiros

C.1 Algoritmo de divisão inteira

Teorema (Divisão com resto): Para cada inteiro c (dividendo) e cada inteiro positivo d (divisor), existe um único par de inteiros Q (quociente) e r (resto), tal que $c = d \cdot Q + r$, onde $0 \leq r < d$.

C.2 Quociente

O quociente pode ser descrito por

$$Q = \left\lfloor \frac{c}{d} \right\rfloor \ ,$$

onde $\lfloor(\cdot)\rfloor$ representa o maior inteiro menor que $(\cdot)$.

C.3 Resto ou resíduo

O resto da divisão de c por d pode ser descrito por

$$r = R_d[c] = ((c)) = c \qquad (\text{mod} \quad d) \ ,$$

podendo ainda ser denominado de resíduo de c, módulo d.

C.4 Congruência

Dois números inteiros c_1 e c_2 que, divididos por um terceiro inteiro positivo d, apresentam o mesmo resto (ou resíduo) r são ditos congruentes, módulo d, e são representados por

$$c_1 \equiv c_2 \qquad (\text{mod} \quad d) \ ,$$

onde $\equiv$ denota uma relação de equivalência.

C.5 Relações úteis

Teorema: Para um mesmo número inteiro positivo d,

(i) $R_d[a+b] = R_d[R_d[a] + R_d[b]]$

(ii) $R_d[a \cdot b] = R_d[R_d[a] \cdot R_d[b]]$

onde $+$ e $\cdot$ denotam, respectivamente, as operações de adição e multiplicação entre números inteiros.

Apêndice D

Minimização de tabela de estados

D.1 Introdução

- A minimização do número de estados de um circuito seqüencial pode conduzir à redução da quantidade de circuitos lógicos necessários para implementar os estados (bloco Geração e Armazenamento) e as saídas (bloco Função Combinacional).
- Dada uma tabela de transição de estados (*state table*), pode-se constatar que diferentes estados podem realizar a mesma função.
- Do ponto de vista externo ao circuito, pode-se dizer que não é possível distinguir entre tais estados, uma vez que eles apresentam o mesmo resultado.
- Nesse caso, tal conjunto de estados pode ser representado por um único estado.
- Conseqüentemente, a tabela de transição de estados (*state table*) é simplificada e, possivelmente, o circuito lógico minimizado.
- Na minimização do número de estados de uma máquina seqüencial, a idéia básica é organizar os estados de uma máquina M1 em classes que possuam uma determinada propriedade e, em seguida, definir uma máquina M2, de tal forma que cada estado em M2 cumpra a função de uma das classes em M1.
- Podem-se destacar dois grupos de descrição de máquinas: i) descrição completamente especificada e ii) descrição não completamente especificada.
- Nas máquinas seqüenciais com descrição completamente especificada, utiliza-se o critério de equivalência entre máquinas.
- No caso das máquinas com descrição não completamente especificada, utilizam-se os critérios de compatibilidade e cobertura.
- Pode-se dizer que a equivalência é um caso particular de cobertura, que, por sua vez, é um caso particular de compatibilidade.
- Nas máquinas seqüenciais com descrição completamente especificada, a solução é única e, portanto, o processo é mais simples e direto. Nesses casos, utiliza-se o critério de equivalência para garantir o cumprimento da mesma função por duas máquinas, M1 e M2. Empregando-se as condições de exclusão definidas para equivalência, os estados de uma máquina M1 são organizados em classes disjuntas de equivalência. Para cada classe de M1 é definido um estado equivalente em M2.

- Nas máquinas seqüenciais com descrição não completamente especificada, normalmente deve-se avaliar diferentes soluções possíveis, o que torna o processo mais complexo e menos objetivo. Nesses casos, utiliza-se o critério de cobertura para garantir o cumprimento da mesma função por duas máquinas, M1 e M2. Empregando-se as condições de exclusão definidas para compatibilidade, os estados de uma máquina M1 são organizados em classes conjuntas de compatibilidade máxima. Em seguida, deve-se determinar uma coleção de cobertura (*cover collection*) mínima, que é uma coleção fechada (*closed collection*) mínima que contém cada estado de M1 em, pelo menos, uma classe de compatibilidade. Para cada classe de compatibilidade da coleção de cobertura de M1 é definido um estado de cobertura em M2.

- O formalismo (Definições, Teoremas e Corolários) apresentado nesse capítulo foi retirado integralmente de [HP81].

D.2 Tabelas de estados completamente especificadas

D.2.1 Relações de equivalência

- Quando um par ordenado de elementos (x, y) possui uma propriedade R que os relaciona, pode-se dizer que "x é R-relacionado com y", o que é simbolizado por xRy.

- A relação R é definida como o conjunto de todos os pares ordenados que possuem a propriedade em questão.

- Pode-se assumir que R é uma relação definida sobre um conjunto de elementos, de tal forma que x ou y possam representar qualquer elemento do conjunto.

- Classificação das relações:

  - Reflexão: se xRx é válida para qualquer x, então R é reflexiva.
  - Simetria: se $yRx \leftrightarrow xRy$, então R é simétrica.
  - Transitividade: se (xRy e yRz) $\rightarrow xRz$, então R é transitiva.
  - Equivalência: se R é reflexiva, simétrica e transitiva, então R é uma relação de equivalência.

D.2.2 Estados e circuitos equivalentes

- As tabelas de transição de estados (*state tables*) representam duas funções: a função de próximo estado $\delta(\cdot)$ e a função de saída $\lambda(\cdot)$.

- Pode-se definir a função de próximo estado por: $\delta(q_i^n, x^n) = q_j^{n+1}$.

- Pode-se definir a função de saída por: $\lambda(q_i^n, x^n) = z^n$.

- Para uma seqüência de sinais de entrada dada por $X = x^n x^{n+1} \cdots x^{n+R}$, tem-se: $\delta(q_i^n, x^n x^{n+1} \cdots x^{n+R}) = q_j^{n+(R+1)}$ e $\lambda(q_i^n, x^n x^{n+1} \cdots x^{n+R}) = z^n z^{n+1} \cdots z^{n+R}$.

- Em última análise, dado um estado inicial e uma seqüência de valores de entrada, a função de um circuito seqüencial é produzir uma seqüência de valores de saída apropriada.

- Dessa forma, podem-se estabelecer relações de equivalência entre estados e entre circuitos seqüenciais.

- **Definição 1**: Sejam S e T dois circuitos seqüenciais completamente especificados, sujeitos a seqüências de entrada possíveis e idênticas. Seja $(x^n x^{n+1} \cdots x^{n+R})$ uma seqüência de possíveis valores de entrada, de comprimento arbitrário. Os estados $p \in T$ e $q \in S$ são ditos indistinguíveis (equivalentes), definido por $p \equiv q$, se e somente se $\lambda_T(p^n, x^n x^{n+1} \cdots x^{n+R}) = \lambda_S(q^n, x^n x^{n+1} \cdots x^{n+R})$ para cada possível seqüência de entrada.

- **Definição 2**: Os circuitos seqüenciais S e T são ditos equivalentes, definido por $S \equiv T$, se e somente se para cada estado p em T existe um estado q em S tal que $p \equiv q$, e, inversamente, para cada estado q em S existe um estado p em T tal que $q \equiv p$.

D.2.3 Determinação de classes de estados indistinguíveis

- Uma proposta para se obter a tabela de transição de estados (*state table*) mínima é particioná-la no menor número possível de classes de equivalência de estados indistinguíveis.

- Em seguida, pode-se obter um circuito seqüencial equivalente, onde cada estado corresponda a uma classe do circuito original.

- Uma vez que nem toda partição é uma classe de equivalência, deve-se ter uma forma de se definir corretamente as partições.

- **Teorema 1**: Suponha que os estados de um circuito seqüencial foram particionados em classes disjuntas, onde $p \triangleq q$ denota que os estados p e q pertencem à mesma classe. A partição é composta por classes de equivalência de estados indistinguíveis se e somente se as duas condições seguintes forem satisfeitas por cada par de estados p e q da mesma classe, para cada entrada simples x^n:

  1. $\lambda(p^n, x^n) = \lambda(q^n, x^n)$.
  2. $\delta(p^n, x^n) \triangleq \delta(q^n, x^n)$.

D.2.4 Circuito de classes de equivalência

- Com a tabela de transição de estados (*state table*) particionada em classes de equivalência, pode-se obter um circuito seqüencial equivalente ao original, com o número de estados minimizado.

- **Teorema 2**: Suponha que seja formado um circuito seqüencial T, que corresponda a um circuito completamente especificado S, de forma que para cada estado $p_j \in T$ corresponda uma classe de equivalência $C_j \in S$. O circuito T assim construído, denominado circuito de classes de equivalência, é equivalente a S. Além disso, nenhum outro circuito equivalente a S possuirá um número menor de estados do que T e qualquer circuito equivalente a S que possua o mesmo número de estados de T deve ser T.

D.3 Tabelas de estados não completamente especificadas

D.3.1 Introdução

- Na representação de um circuito digital, a falta de especificação de valores pode surgir por diversos fatores.
- Em circuitos combinacionais, determinadas entradas e/ou saídas dos blocos funcionais podem não ocorrer (*can't happen*) ou podem não importar (*don't care*). Genericamente, ambos os casos são empregados como *don't care*, durante o processo de minimização dos circuitos.
- Em circuitos seqüenciais, as indeterminações podem apresentar várias origens:
  - Nas máquinas completamente especificadas que possuem um número de estados cujo valor não é uma potência de dois, os estados extras da tabela de atribuição podem ser assumidos como *don't cares*. Nesses casos, é comum que eles recebam a denominação de *don't cares* acidentais (*incidental don't cares*).
  - Determinadas seqüências de entrada podem nunca acontecer, gerando indeterminações na tabela de estados (próximas entradas e saídas), as quais podem ser especificadas como *don't cares*.
  - Em máquinas onde as saídas são amostradas em intervalos de tempo maiores do que aqueles das mudancas de estado, podem-se atribuir valores indeterminados às saídas intermediárias, as quais também podem ser assumidas como *don't cares*.
- Em circuitos combinacionais, o valor *don't care* ('X') pode ser substituído apenas por valores booleanos ('0' ou '1'). Nesses casos, a substituição de valores é um processo simples de ser executado e sempre auxilia na minimização das expressões.
- Em circuitos seqüenciais, situações diferentes podem ocorrer:
  - Um valor *don't care* ('X') de estado/saída pode ser substituído por N/M valores de estados/saídas.
  - A substituição indiscriminada de valores de estados/saídas pode: i) proporcionar a minimização do número de estados, ii) conduzir a um número próximo do mínimo ou iii) impedir a minimização.
- Assim, deve-se adotar um método sistemático na tentativa de minimização de estados de máquinas não completamente especificadas.

D.3.2 Noções básicas de compatibilidade

- Dada uma tabela de estados, com próximos estados e/ou saídas não completamente especificados, é possível que se realize combinações de estados, reduzindo o número total estados da tabela.
- Porém, não se pode falar, genericamente, de equivalência de estados em máquinas seqüenciais não completamente especificadas.
- A equivalência entre estados exige que tanto suas saídas quanto seus próximos estados sejam definidos para todos os valores das entradas.

- Tal exigência não é cumprida por máquinas não completamente especificadas.
- Estados não completamente especificados que podem ser combinados em um único estado final são ditos compatíveis entre si.
- Obviamente, estados que são idênticos em seus valores especificados podem ser transformados em estados equivalentes através da atribuição adequada de seus valores não especificados.
- Conseqüentemente, tais estados são compatíveis e podem ser combinados em um único estado.
- Porém, tal condição é suficiente, mas não necessária.
- Estados não idênticos também podem ser combinados, em algumas condições.
- Em qualquer associação de estados, é aplicado o conceito segundo o qual estados que possuem a mesma função dentro do circuito devem produzir os mesmos valores de saída, para os mesmos valores de entrada.
- No caso das máquinas não completamente especificadas, a compatibilidade é associada apenas aos valores especificados de entrada, de próximo estado e de saída.

D.3.3 Formalização dos conceitos de compatibilidade e de cobertura

- **Definição 1**: Seja uma seqüência de valores de entrada $x = \{x^n x^{n+1} \cdots x^{n+(K+1)}\}$, aplicada a um circuito S, cuja descrição é não completamente especificada e que se encontra em um estado inicial q^n. A seqüência x é dita aplicável a q se todos os valores de próximo estado forem especificados, exceto, possivelmente, aquele produzido pela última entrada da seqüência.
- **Definição 2**: Dois estados, p e q, de um circuito S, são ditos compatíveis se e somente se

$$\lambda_S\left(\delta(p, x^n x^{n+1} \cdots x^{n+K}), x^{n+(K+1)}\right) = \lambda_S\left(\delta(q, x^n x^{n+1} \cdots x^{n+K}), x^{n+(K+1)}\right) \ ,$$

  sempre que ambas as saídas forem especificadas, para cada seqüência x aplicável a ambos os estados, onde $x = \{x^n x^{n+1} \cdots x^{n+(K+1)}\}$.
- **Teorema 1**: Se dois estados, p e q, de um circuito S, são compatíveis, então as seguintes condições devem ser satisfeitas para toda entrada simples x:
  1. $\lambda(p^n, x^n) = \lambda(q^n, x^n)$, sempre que ambas forem especificadas.
  2. $\delta(p^n, x^n)$ e $\delta(q^n, x^n)$ são compatíveis, sempre que ambos forem especificados.
- **Definição 3**: Um conjunto de estados S_i, de um circuito S, é denominado uma classe de compatibilidade se cada par de estados em S_i for compatível.
- **Definição 4**: Uma classe de compatibilidade máxima é uma classe de compatibilidade que deixará de sê-la, se um estado que não lhe for pertencente for a ela adicionado. Um estado isolado, que não é compatível com qualquer outro estado, é definido como uma classe de compatibilidade máxima.

- **Definição 5**: Diz-se que um estado p, de uma tabela de estados T, cobre um estado q, de uma tabela de estados S, o que é definido por $p \geq q$, se, para qualquer seqüência de entradas aplicável a q e aplicada a ambas as tabelas, inicialmente nos estados p^n e q^n, respectivamente, as duas seqüências de saídas forem idênticas, sempre que a saída de S for especificada.

- **Definição 6**: Diz-se que uma tabela de estados T cobre uma tabela de estados S se, para estado q em S, existe um estado p em T que cobre q.

- **Teorema 2**: Se um estado p em T cobre ambos os estados q_i e q_j em S, então os estados q_i e q_j devem ser compatíveis.

- **Corolário 2.1**: Se um estado p em T cobre um conjunto de estados S_i de S, então tais estados devem formar uma classe de compatibilidade.

- **Definição 7**: Uma coleção de classes de compatibilidade é dita fechada se, para qualquer classe $\{q_1,\ q_2,\ \cdots,\ q_m\}$ da coleção e para toda entrada simples x, todos os próximos estados especificados, $\delta(q_1^n, x^n)$, $\delta(q_2^n, x^n)$, $\cdots$, $\delta(q_m^n, x^n)$, pertencem a uma única classe da coleção.

- **Teorema 3**: Suponha que, a partir dos n estados de um circuito seqüencial não completamente especificado S, seja formada uma coleção de m classes de compatibilidade, de modo que cada um dos n estados seja membro de, pelo menos, uma das m classes. O circuito S poderá ser coberto por um circuito T, que possua exatamente m estados, $(p_1,\ p_2,\ \cdots,\ p_m)$, de forma que cada classe de compatibilidade de S seja coberta por um dos estados de T, se e somente se a coleção de m classes de compatibilidade de S for fechada.

D.3.4 Sistematização do processo de minimização

- A **Definição 1**, a **Definição 2**, e o **Teorema 1** apresentam as condições de exclusão que podem ser usadas na organização dos estados em classes de compatibilidade.

- A **Definição 3** e a **Definição 4**, fornecem as diretrizes para a geração das classes de compatibilidade.

- A **Definição 5**, a **Definição 6**, o **Teorema 2** e o **Corolário 2.1** tratam da propriedade de cobertura e de sua relação com a propriedade de compatibilidade.

- A **Definição 7**, e o **Teorema 3** indicam as condições de cobertura entre máquinas.

Apêndice E

Linguagens de descrição de *hardware*

E.1 Introdução

- Desde a implementação do primeiro dispositivo eletrônico em circuito integrado, os avanços tecnológicos têm possibilitado um rápido aumento na quantidade de elementos que podem ser combinados em um único circuito nesse tipo de implementação.
- Naturalmente, com a oferta de uma maior densidade de componentes, a complexidade dos circuitos projetados cresce na mesma taxa.
- Porém, a capacidade de um ser humano em lidar com a idealização, o projeto, a documentação e a manutenção de sistemas com um grande número de componentes é extremamente limitada.
- Dessa forma, torna-se necessário o uso de ferramentas de apoio, adequadas a tal tipo de problema.
- Existem duas técnicas de projeto largamente utilizadas na abordagem de problemas de elevada complexidade:
  - Adotar uma visão hierárquica na elaboração do sistema, de forma que, em cada nível de representação, toda a complexidade dos níveis inferiores seja ocultada.
  - Aumentar o nível de abstração na descrição do sistema, de forma que o foco esteja mais na função desempenhada e menos na implementação propriamente dita.

E.2 Abordagem hierárquica

- Na definição de um sistema de baixa complexidade, pode-se descrever a sua operação de uma forma simples e direta.
- Por outro lado, na definição de sistemas com complexidade elevada, pode-se utilizar o conceito organizacional de hierarquia.
- Em uma abordagem hierárquica, um sistema de complexidade genérica é recursivamente dividido em módulos ou unidades mais simples. O ponto de parada da recursividade é subjetivo e costuma ser escolhido como a descrição comportamental mais simples possível e/ou desejada.

- Nesse sentido, o sistema completo pode ser interpretado como o módulo mais complexo ou mais externo da hierarquia.

- A abordagem hierárquica facilita a descrição, a análise e o projeto dos circuitos, uma vez que cada módulo pode ser tratado como um circuito único, isolado dos demais.

- A descrição hierárquica no sentido do todo para as partes mais simples é chamada de *top-down*.

- Por outro lado, a descrição hierárquica no sentido das partes mais simples para o todo é chamada de *bottom-up*.

- Normalmente, aplica-se um processo *top-down* para a especificação e um processo *bottom-up* para a implementação de sistemas.

E.3 Níveis de abstração

A descrição, a análise e a síntese de circuitos podem ser realizadas em diversos níveis de abstração.

Do ponto de vista do comportamento modelado, os seguintes níveis podem ser considerados:

- Físico-matemático: que adota equações matemáticas para descrever um modelo físico de comportamento. Obviamente, é o modelo mais próximo do comportamento físico do circuito. É tipicamente utilizado na descrição funcional de circuitos analógicos.

- Lógico: que emprega equações lógicas na sua descrição. É naturalmente utilizado na descrição funcional de circuitos digitais.

- Estrutural: que utiliza, intrinsicamente, uma descrição hierárquica do circuito modelado. Inicialmente, são definidos, testados e validados, blocos de baixa complexidade. Em seguida, realizando-se instanciações desses blocos, são definidos blocos de complexidade mais elevada. Tal processo é repetido até que o sistema desejado seja adequadamente definido. Portanto, esse é um modelo que preserva a visão da estrutura física dos circuitos.

- Comportamental: que apresenta um nível de representação mais abstrato e mais distante do sistema físico. Encontra aplicação direta em testes de funcionalidade dos circuitos.

Do ponto de vista da complexidade dos sistemas, os seguintes níveis podem ser considerados:

- Componentes: que representam os elementos básicos de circuitos.

- Células básicas: que são circuitos de baixa complexidade.

- Blocos funcionais: que são circuitos de média complexidade.

- Sistemas: que são circuitos de alta complexidade.

E.4 Linguagens de descrição de *hardware*

- Na área de projeto de circuitos integrados, o uso de uma Linguagem de Descrição de *Hardware* (*Hardware Description Language* ou HDL) tem sido proposto, a fim de permitir uma descrição mais abstrata dos seus elementos constituintes e de possibilitar que estes sejam organizados de forma hierárquica.
- Uma HDL é uma linguagem de modelagem, utilizada para descrever tanto a estrutura quanto a operação de um *hardware* digital.
- Em linguagens de programação comuns, os processos de interpretação e de compilação podem ser modelados como a tradução de uma linguagem entendida por uma máquina virtual para uma outra linguagem asssociada a uma outra máquina virtual.
- No caso de uma HDL, o modelo é um pouco diferente. A partir da descrição apresentada pelo código elaborado, o compilador deve inferir um *hardware* digital equivalente.
- Portanto, por meio de uma HDL, além de um mapeamento lingüístico e/ou matemático, é realizado um mapemento físico.
- De acordo com o seu comportamento funcional, um *hardware* digital pode ser classificado da seguinte forma:
  - Combinacional: sistema instantâneo (ou sem memória), com operação concorrente de eventos.
  - Seqüencial: sistema dinâmico (ou com memória), com operação seqüencial de eventos.
- Logo, uma HDL deve ser capaz de descrever ambos os comportamentos: o concorrente e o seqüencial.
- As aplicações típicas para uma HDL são as seguintes:
  - Documentação de circuitos digitais.
  - Análises de circuitos digitais, tais como: simulações e checagens diversas.
  - Síntese (projeto) de circuitos digitais. Uma vez definida uma implementação alvo, o compilador infere um circuito equivalente à descrição HDL e gera um código adequado para tal implementação. Sínteses típicas são as seguintes: a geração de código para configurar dispositivos lógicos programáveis e a geração de máscaras (*layout*) para fabricação de circuitos integrados.
- Algumas características que levam uma HDL a ser largamente empregada são as seguintes:
  - Apresentar padrões bem estabelecidos e bem documentados.
  - Apresentar características encontradas em outras HDLs.
  - Possuir vasta literatura disponível.
  - A existência de várias ferramentas computacionais para a HDL em questão, tais como: editores, compiladores, simuladores, checadores diversos.
  - A existência de ferramentas computacionais de diversos tipos, tais como:
    * Domínio público ou comercial.

    - Implementada isoladamente ou incluída em ambiente de desenvolvimento integrado (*Integrated Development Environment* ou IDE).
    - Implementada em diversas plataformas computacionais.

- Exemplos de HDL
  - Independentes de tecnologia e de fabricante: VHDL, Verilog, SystemVerilog.
  - Dependentes de fabricante: AHDL (Altera HDL).
- A Tabela E.1 apresenta uma lista de fabricantes, produtos e funções, que lidam com HDL.

Fabricante	Produto	Função
Altera	Quartus II	Síntese e simulação
Xilinx	ISE	Síntese e simulação
Menthor Graphics	Precision RTL	Síntese
	ModelSim	Simulação
Synopys/Synplicity	Design Compiler Ultra	Síntese
	Synplify Pro/Premier	
	VCS	Simulação
Cadence	NC-Sim	Simulação

Tabela E.1: Lista de fabricantes, produtos e funções, que lidam com HDL.

Apêndice F

Introdução à linguagem VHDL

A linguagem VHDL é apresentada a seguir, de forma introdutória. Para que se adquira um conhecimento mais aprofundado sobre a linguagem, é recomendado consultar uma literatura específica (manuais e livros especializados).

F.1 Histórico da linguagem VHDL

- Durante o desenvolvimento do programa *Very High Speed Integrated Circuits* (VHSIC), iniciado em 1980 pelo Departamento de Defesa (DoD) dos Estados Unidos da América, surgiu a necessidade de uma HDL específica para lidar com os tipos de circuitos integrados envolvidos no programa. Em função disso, foi proposta a primeira versão da linguagem VHDL (*VHSIC Hardware Description Language*).

- A linguagem VHDL continuou a ser desenvolvida pelo IEEE (*Institute of Electrical and Electronics Engineers*) e foi a primeira HDL padronizada, por meio dos padrões IEEE Standard 1076 (*Standard VHDL Language Reference Manual* - 1987) e IEEE Standard 1164 (*Standard Multivalue Logic System for VHDL Model Interoperability* - 1993).

- Os padrões IEEE são revisados, pelo menos, a cada cinco anos. Portanto, já foram gerados os padrões VHDL-1987, VHDL-1993, VHDL-2002 e VHDL-2008.

F.2 VHDL como linguagem

Como qualquer linguagem escrita, VHDL utiliza um conjunto específico de símbolos e de regras que definem aspectos de sintaxe e de semântica.

Embora seja uma linguagem específica para a descrição de circuitos eletrônicos digitais, aplicada na documentação, simulação e síntese automática de tais circuitos, VHDL ainda pode ser interpretada como uma linguagem de programação.

Vista como uma linguagem de programação, VHDL apresenta elementos comuns a diversas linguagens de programação modernas, alguns dos quais são discutidos a seguir.

F.2.1 Considerações gerais

- Arquivos contendo código VHDL são formatados em tipo TEXTO.

- O nome do arquivo que contém o código VHDL (`nome.vhd`) deve ser o mesmo nome da entidade mais externa na hierarquia de circuitos descrita pelo arquivo em questão.

- Comentários são iniciados com dois hífens consecutivos (- -).
- Um sinal físico, com valores binários, é definido pelo tipo BIT. Por sua vez, um conjunto de tais sinais é definido pelo tipo BIT_VECTOR.
- O valor de um sinal do tipo BIT é indicado com aspas simples (p.ex.: '0' e '1'), enquanto o valor de um sinal do tipo BIT_VECTOR é delimitado por aspas duplas (p.ex.: "0000", "0101" e "1111").
- Duas operações básicas no uso de VHDL são a compilação e a simulação. A partir de uma descrição VHDL do circuito digital, armazenada em arquivo do tipo texto, o compilador VHDL infere um circuito equivalente na implementação alvo e armazena tal informação em um outro arquivo do tipo texto. A partir do arquivo que contém informação sobre o circuito compilado e de uma descrição de sinais de teste, armazenada em arquivo do tipo texto, o simulador VHDL calcula os sinais gerados pelas saídas do circuito.

F.2.2 Palavras reservadas

As palavras reservadas (*reserved words* ou *keywords*) são identificadores que possuem um significado especial dentro da linguagem. Assim, seu uso é restrito à sua definição original e, uma vez que não podem ser redefinidas, elas não podem ser empregadas para nenhum outro propósito.

A Figura F.1 apresenta as palavras reservadas de VHDL.

F.2.3 Identificadores definidos pelo usuário

Algumas regras básicas para a construção de identificadores são as seguintes:

- Todo identificador é formado por uma seqüência de caracteres (*string*) única, de qualquer comprimento.
- Identificadores podem ser formados apenas com letras minúsculas e/ou maiúsculas (a até z e A até Z), com números de 0 a 9 e com o símbolo "_" (sublinhado ou *underscore*).
- Todo identificador deve começar com uma letra.
- O símbolo de *underscore* não pode ser usado como primeiro nem como último caractere do identificador. Também não é permitido usar dois símbolos de *underscore* consecutivos.
- VHDL não é *case sensitive*. Logo: identificador $\equiv$ IdEnTiFiCaDoR $\equiv$ IDENTIFICADOR.

Os padrões mais recentes permitem o uso de um conjunto expandido de caracteres, incluindo a utilização de hífen e de acentos. Porém, as regras acima são suficientes para garantir a compatibilidade entre os diversos padrões.

F.2.4 Elementos sintáticos

Além das palavras reservadas e dos identificadores definidos pelo usuário, podem-se utilizar símbolos especiais para escrever o código VHDL. Assim como as palavras reservadas, seu uso é restrito à sua definição original.

A Figura F.2 apresenta símbolos especiais de VHDL.

abs	disconnect	label	package	then
access	downto	library	port	to
after		linkage	postponed	transport
alias	else	literal	procedure	type
all	elseif	loop	process	
and	end		protected	unaffected
architecture	entity	map	pure	units
array	exit	mod		until
assert			range	use
attribute	file	nand	record	
	for	new	register	variable
begin	function	next	reject	
block		nor	rem	wait
body	generate	not	report	when
buffer	generic	null	return	while
bus	group		rol	with
	guarded	of	ror	
case		on		xnor
component	if	open	select	xor
configuration	impure	or	severity	
constant	in	others	shared	
	inertial	out	signal	
	inout		sla	
	is		sll	
			sra	
			srl	
			subtype	

Figura F.1: Palavras reservadas de VHDL.

Símbolo	Significado
−−	Comentário
;	Terminador
(	Parêntese da esquerda
)	Parêntese da direita
:	Separação entre elemento e tipo
<>	Declaração de faixa indefinida (*box*)
#	Notação `base#número#`
.	Notação de ponto
’	Aspas simples ou marca de *tick*
”	Aspas duplas
&	Concatenador
<=	Atribuição a sinal
:=	Atribuição a variável/constante
=>	Atribuição a elemento de conjunto

Símbolo	Significado
|	OR condicional
=>	Símbolo de THEN em CASE
+	Adição ou identidade unária
−	Subtração ou negação unária
*	Multiplicação
/	Divisão (com truncamento)
**	Exponenciação
=	Igual a
/=	Diferente de
<	Menor do que
>	Maior do que
<=	Menor do que ou igual a
>=	Maior do que ou igual a

Figura F.2: Símbolos especiais de VHDL.

F.3 Conceitos básicos sobre o código VHDL

F.3.1 Elementos básicos

- Alguns elementos básicos de um código VHDL são: constante, variável, sinal e operador.
- Constantes são geralmente empregadas para flexibilização e/ou otimização do código.
- Variável é um elemento abstrato para armazenamento de informação matemática.
- Sinal é o elemento da linguagem associado com elementos físicos de conexão (pino e fio).
- Operadores representam as relações funcionais básicas. Eles são definidos com as palavras reservadas e com os elementos sintáticos, sendo organizados em seis classes: atribuição, lógica, aritmética, comparação, deslocamento e concatenação. A Tabela F.1 apresenta os operadores de VHDL.

Classe	Operadores
Atribuição	<=, :=, =>.
Lógica	NOT, AND, OR, XOR, NAND, NOR, XNOR.
Aritmética	+, -, *, /, **, ABS, MOD(1), REM(2).
Comparação	=, /=. <, >, <=, >=.
Deslocamento	SLL, SRL, SLA, SRA, ROL, ROR.
Concatenação	& (“,” e OTHERS).

(1) *Module*: resto de a/b, com sinal de b.
(2) *Remainder*: resto de a/b, com sinal de a.

Tabela F.1: Operadores de VHDL.

F.3.2 Tipos de execução

- De acordo com o tipo de execução, os códigos VHDL são divididos em: concorrente e seqüencial.
- Códigos concorrentes são empregados para descrever circuitos digitais combinacionais. Por sua vez, os códigos seqüenciais são utilizados para descrever tanto circuitos combinacionais quanto circuitos seqüenciais.
- As instruções VHDL são naturalmente executadas de forma concorrente. Assim, embora o código seja organizado em linhas, todas as linhas têm igual precedência.
- Para que um código seja considerado seqüencial, isso deve ser forçado. O mecanismo mais comum para forçar um código a ser seqüencial é denominado de processo, definido pela instrução PROCESS. Um processo é concorrente com qualquer outro comando e com qualquer outro processo. Outras opções para gerar código seqüencial são os subprogramas (FUNCTION e PROCEDURE).

- As constantes podem ser declaradas e usadas em ambos os tipos de código.
- Variáveis só podem ser declaradas e usadas dentro de um código seqüencial.
- Os sinais só podem ser declarados dentro de código concorrente, mas podem ser utilizados em ambos os tipos de código.
- Os operadores podem ser usados para construir ambos os tipos de código.
- As seguintes instruções de controle de fluxo podem ser empregadas apenas em código concorrente: WHEN, SELECT e GENERATE.
- Por outro lado, as seguintes instruções de controle de fluxo podem ser empregadas apenas em código seqüencial, ou seja, dentro de PROCESS, FUNCTION ou PROCEDURE: IF, CASE, LOOP e WAIT.

F.3.3 Mecanismo genérico de simulação

- A simulação de um modelo em VHDL é baseada em eventos.
- A passagem do tempo é simulada em passos discretos, associados à ocorrência de eventos.
- Quando um novo valor é agendado para ser atribuído a um dado sinal, em um tempo futuro, diz-se que ocorreu o agendamento de uma transação sobre tal sinal.
- Ao ocorrer uma atribuição a um sinal, se o novo valor for diferente do valor anterior, diz-se que ocorreu um evento sobre tal sinal.
- Uma simulação é dividida em duas partes: a fase de inicialização e os ciclos de simulação.
- A fase de inicialização é composta pelas seguintes etapas:
  - O tempo da simulação é ajustado para o valor inicial $t = 0$ s.
  - A cada sinal declarado, é atribuído um valor inicial.
  - Para cada um dos processos declarados, é ativada uma instância e seus comandos seqüenciais são executados.
  - Usualmente, um processo contém atribuições a sinais, que agendarão transações sobre eles, em valores futuros de tempo.
  - Cada processo executa até que seja alcançado o comando WAIT, o que causa a sua suspensão.
  - Após a suspensão de todos os processos ativados, a fase de inicialização é terminada, passando-se para a execução dos ciclos de simulação.
- Os ciclos de simulação são formados pelas seguintes etapas:
  - O tempo de simulação é avançado até o próximo valor para o qual foi agendada uma transação sobre um sinal.
  - Todas as transações agendadadas para o tempo corrente são executadas.
  - As execuções das transações podem causar a ocorrência de eventos sobre sinais.
  - Todos os processos sensíveis aos sinais sobres os quais ocorreram eventos são reativados.

  - Os processos reativados executam seus comandos seqüenciais.
  - Possivelmente, os processos reativados agendarão novas transações sobre sinais
  - Cada processo executa até que seja alcançado o comando WAIT, o que causa a sua suspensão.
  - Após a suspensão de todos os processos reativados, o ciclo é repetido.
- Quando não houver mais qualquer transação agendadada, a simulação atinge seu fim.

F.4 Estrutura do código VHDL

Um código VHDL genérico, que contém a descrição de um determinado circuito, apresenta as seguintes partes:

- Declaração de bibliotecas e pacotes: que é um conjunto de declarações sobre as bibliotecas a serem consideradas e sobre os pacotes pertencentes a tais bibliotecas que deverão ser empregados.
- Entidade: que descreve a interface de acesso ao circuito, definindo a sua identificação, as suas entradas e as suas saidas.
- Arquitetura: que descreve a operação do circuito, definindo as relações entre as suas saídas e as suas entradas.

Cada uma dessas partes é discutida a seguir.

F.4.1 Bibliotecas e pacotes

- Em aplicativos de desenvolvimento, é comum que diversos elementos sejam previamente definidos, tais como: identificadores, valores constantes, nomes de variáveis e de estruturas, inicialização de variáveis e de estruturas, macros, funções e objetos.
- O objetivo em se definir previamente tais elementos é facilitar o trabalho do projetista.
- Uma vez definidos, testados e validados, tais elementos podem ser utilizados em quaisquer projetos, sem que seja necessária a sua definição a cada projeto.
- Cada aplicativo possui seus próprios padrões para a organização dos elementos previamente definidos.
- Uma organização simples e bastante utilizada são os arquivos de configuração.
- Por outro lado, uma forma mais estruturada de organização é obtida através do agrupamento de informações em um pacote (*package*) e de pacotes em uma biblioteca (*library*).
- Os compiladores VHDL normalmente consideram a inclusão automática das seguintes bibliotecas: *std* e *work*.
- A biblioteca *std* contém definições sobre os tipos básicos de dados e os correspondentes operadores. Por sua vez, a biblioteca *work* indica o diretório onde estão armazenados os arquivos do projeto.
- Nos circuitos digitais descritos em VHDL, são largamente utilizados a biblioteca padrão *ieee* e os seus pacotes *standard* e *IEEE 1164*, ambos definidos pelo IEEE.

F.4.2 Entidade

Entidade é o termo associado a um módulo de circuito em VHDL. A declaração de uma entidade descreve a interface de acesso ao circuito, definindo a sua identificação, as suas entradas e as suas saidas. As entradas e saídas são associadas aos pontos ou pinos de acesso do circuito físico e são conjuntamente denominadas de portas.

F.4.3 Arquitetura

Aspectos gerais

Arquitetura é o mecanismo utilizado em VHDL para descrever a operação de um circuito. Uma declaração de arquitetura é sempre associada a uma determinada entidade. Na arquitetura, são definidas as relações entre as saídas e as entradas da entidade a ela associada.

Um mesmo circuito digital pode ser descrito por diversas maneiras equivalentes, tais como: tabela verdade, equações genéricas diferentes, equações relacionadas a diferentes decomposições específicas, diferentes decomposições hierárquicas. Assim, dependendo do nível de abstração utilizado, uma mesma entidade pode ter seu funcionamento descrito por diversas arquiteturas diferentes.

Tipos de descrição

De uma forma geral, a descrição das operações de um circuito dentro de uma arquitetura pode assumir duas formas: comportamental ou estrutural. Em uma descrição comportamental, é feita uma descrição explícita das relações entre as saídas e as entradas. Para tal, são usados os operadores e/ou as instruções de controle de fluxo. Por sua vez, em uma descrição estrutural, é utilizado o conceito de hierarquia. Nesse caso, são utilizados módulos mais simples, previamente descritos, bem como são definidas as conexões que os interligam. Dado que uma descrição estrutural é um modelo hierárquico, os módulos que compõem o nível mais baixo e básico da hierarquia devem ser descritos de uma forma comportamental.

Instanciação de módulos

Deve-se notar que a existência de um mecanismo que possibilita a definição de um circuito em forma hierárquica permite a construção de bibliotecas de módulos e a instanciação de módulos a partir de uma determinada biblioteca.

A técnica de instanciação de módulos a partir de uma biblioteca otimiza o trabalho de descrição de circuitos complexos, bem como permite a reusabilidade de código.

Em VHDL, um módulo instanciável é denominado de componente (COMPONENT). Pode-se instanciar um módulo componente de duas formas básicas:

- O componente é declarado em um pacote, que é localizado em uma biblioteca, e é instanciado no código principal.

- O componente é declarado e instanciado no código principal.

F.5 Algumas regras sintáticas de VHDL

Comumente, as regras sintáticas das linguagens de programação são apresentadas com as notações denominadas de BNF (*Backus-Naur Form*) e EBNF (*Extended Backus-Naur Form*). Seguindo esse padrão, alguma regras sintáticas de VHDL são apresentadas a seguir.

F.5.1 Regras para biblioteca

```
library_clause <=
    LIBRARY identifier { , ... } ;

use_clause <=
    USE selected_name { , ... } ;

selected_name <=
    identifier . identifier . ( identifier | ALL )
```

Em `selected_name`, o `identifier` mais à esquerda refere-se à biblioteca, o `identifier` do meio indica o pacote e o `identifier` mais à direita aponta o item a ser utilizado.

F.5.2 Regras para pacote

```
package_declaration <=
    PACKAGE identifier IS
        { package_declarative_item}
    END [ PACKAGE ] [ identifier ] ;

package_body <=
    PACKAGE BODY identifier IS
        { package_body_declarative_item}
    END [ PACKAGE BODY ] [ identifier ] ;
```

F.5.3 Regras para entidade

```
entity_declaration <=
    ENTITY identifier IS
        [ GENERIC ( generic_interface_list ) ; ]
        [ PORT    (    port_interface_list ) ; ]
    END [ ENTITY ] [ identifier ] ;

generic_interface_list <=
    ( identifier { , ... } : subtype_indication [ := expression ] )
    { , ... }

port_interface_list <=
    ( identifier { , ... } : [ mode ] subtype_indication ) { , ... }
```

```
modet <=
    IN | OUT | INOUT
```

F.5.4 Regras para arquitetura

```
architecture_body <=
    ARCHITECTURE identifier OF entity_name IS
        { block_declarative_item }
    BEGIN
        { concurrent_statement }
    END [ ARCHITECTURE ] [ identifier ] ;

signal_declaration <=
    SIGNAL identifier { , ... } : subtype_indication [ := expression ] ;

signal_assignment_statement <=
    name <= ( value_expression [ AFTER time_expression ] ) { , ... } ;

conditional_signal_assignment <=
    name <= { waveform WHEN boolean_expression ELSE }
              waveform [ WHEN boolean_expression ] ;

selected_signal_assignment <=
    WITH expression SELECT
        name <= { waveform WHEN choices , }
                  waveform WHEN choices ;
```

F.5.5 Regras para processo

```
process_statement <=
    process_label:
    PROCESS [ ( signal_name { , ... } ) ] [ IS ]
        { process_declarative_item }
    BEGIN
        { sequential_statement }
    END PROCESS ;

wait_statement <=
    WAIT [ ON    signal_name { , ... } ]
         [ UNTIL boolean_expression    ]
         [ FOR   time_expression       ];
```

F.5.6 Regras para componente

```
component_instantiation_statement <=
    instantiation_label:
    ENTITY entity_name ( architecture_identifier )
        [ GENERIC MAP ( generic_association_list ) ]
        PORT MAP ( port_association_list ) ;

generic_association_list <=
    ( [ generic_name => ] ( expression | OPEN ) ) { , ... } ;

port_association_list <=
    ( [ port_name => ] signal_name ) { , ... } ;
```

F.6 Exemplos de declarações genéricas

F.6.1 Exemplos de biblioteca e de pacote

Um código tipicamente encontrado em arquivos VHDL, para a declaração de bibliotecas e para o uso de pacotes, é o seguinte:

```
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

F.6.2 Exemplos de entidade

Uma declaração genérica de entidade é a seguinte:

```
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY __entity_name IS
  --
  GENERIC(__parameter_name : __type := __default_value;
          __parameter_name : __type := __default_value);
  --
  PORT(__input_name, __input_name   : IN    BIT;
       __input_vector_name          : IN    BIT_VECTOR(__high downto __low);
       __bidir_name,  __bidir_name  : INOUT BIT;
       __output_name, __output_name : OUT   BIT);
  --
END ENTITY __entity_name;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

Deve ser ressaltado que o campo GENERIC é um conteúdo opcional.

F.6.3 Exemplos de arquitetura

Uma declaração genérica de arquitetura é a seguinte:

```
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE __architecture_name OF __entity_name IS
  --  Declarative section
    --  TYPE
    --  CONSTANT
    --  SIGNAL
    --  COMPONENT
    --  VARIABLE
    --  FUNCTION
  --
BEGIN
  --  Code section
    --  Process Statement
    --  Concurrent Procedure Call
    --  Concurrent Signal Assignment
    --  Conditional Signal Assignment
    --  Selected Signal Assignment
    --  Component Instantiation Statement
    --  Generate Statement
  --
END ARCHITECTURE __architecture_name;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

Um exemplo de declaração de arquitetura que emprega componentes (declarados no próprio código) é o seguinte:

```
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE __architecture_name OF __entity_name IS
  --
  COMPONENT C_1 IS
    PORT(x, y : IN  BIT;
         z    : OUT BIT);
  END COMPONENT C_1;
  --
  COMPONENT C_2 IS
    PORT(x, y, z : IN  BIT;
         w       : OUT BIT);
  END COMPONENT C_2;
  --
  SIGNAL s1, s2 : BIT;
  --
BEGIN
  --
  Inst_1 : C_1 PORT MAP(x => a, y => b, z => s1)  --> Syntax #1
  --
```

```
  Inst_2 : C_1 PORT MAP(c,d,s2)  --> Syntax #2
  --
  Inst_3 : C_2 PORT MAP(e,s1,s2,f)
  --
END ARCHITECTURE __architecture_name;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

F.6.4 Exemplos de processo

Uma declaração genérica de processo, comumente usada no caso de circuitos combinacionais, é a seguinte:

```
--
__process_label : PROCESS (__signal_name, __signal_name) IS
  VARIABLE __variable_name : __type;
  VARIABLE __variable_name : __type;
BEGIN
  --  Signal Assignment Statement
  --  Variable Assignment Statement
  --  Procedure Call Statement
  --  If Statement
  --  Case Statement
  --  Loop Statement
END PROCESS __process_label;
--
```

Uma declaração genérica de processo, comumente usada no caso de circuitos seqüenciais, é a seguinte:

```
--
__process_label : PROCESS IS
  VARIABLE __variable_name : __type;
  VARIABLE __variable_name : __type;
BEGIN
  WAIT UNTIL __clk_signal = __valor;
  --  Signal Assignment Statement
  --  Variable Assignment Statement
  --  Procedure Call Statement
  --  If Statement
  --  Case Statement
  --  Loop Statement
END PROCESS __process_label;
--
```

Referências bibliográficas

[Arm62] D. B. Armstrong. A Programmed Algorithm for Assigning Internal Codes to Sequential Machines. *IRE Transactions on Electronic Computers*, EC 11(4):466–472, August 1962.

[HP81] F. J. Hill and G. R. Peterson. *Introduction to Switching Theory and Logical Design*. John Wiley, New York, NY, 3rd edition, 1981.

[Hum58] W. S. Humphrey. *Switching Circuits with Computer Applications*. McGraw-Hill, New York, NY, 1958.

[IC08] I. V. Idoeta and F. G. Capuano. *Elementos de Eletrônica Digital*. Editora Érica, 40.ª edição edition, 2008.

[McC65] E. J. McCluskey. *Introduction to the Theory of Switching Circuits*. McGraw-Hill, New York, NY, 1965.

[Rhy73] V. T. Rhyne. *Fundamentals of Digital Systems Design*. Prentice-Hall, Englewood Cliffs, NJ, 1973.

[Ric56] R. K. Richards. *Arithmetic Operations in Digital Computers*. Van Nostrand Reinhold, New York, NY, 1956.

[Sau67] G. A. Saucier. Encoding of Asynchronous Sequential Networks. *IEEE Transactions on Computers*, EC 16(3), 1967.

[Tau82] H. Taub. *Digital Circuits and Microprocessors*. McGraw-Hill, New York, NY, 1982. Em português: McGraw-Hill, Rio de Janeiro, 1984.

[TWM07] R. J. Tocci, N. S. Widmer, and G. L. Moss. *Sistemas Digitais: Princípios e Aplicações*. Prentice Hall, Pearson Education, 10.ª edição edition, 2007.

[Ung59] S. H. Unger. Hazards and Delays in Asynchronous Sequential Switching Circuits. *IRE Transactions on Circuit Theory*, CT-6(12), 1959.

[Ung69] S. H. Unger. *Asynchronous Sequential Switching Circuits*. John Wiley, New York, NY, 1969.

[Uye02] J. P. Uyemura. *Sistemas Digitais: Uma abordagem integrada*. Thomson Pioneira, São Paulo, SP, 2002.

[Wik] Wikipedia. Sistema Internacional de Unidades. Disponível em: https://pt.wikipedia.org/wiki/Sistema_Internacional_de_Unidades. Acesso em: 21/07/2023.