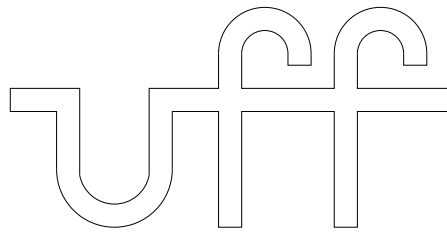


**Apostila
de
Teoria
para
Circuitos Digitais**

(Versão A2019M05D09)



Universidade Federal Fluminense

Alexandre Santos de la Vega

Departamento de Engenharia de Telecomunicações – TET

Escola de Engenharia – TCE

Universidade Federal Fluminense – UFF

Maio – 2019

621.3192	de la Vega, Alexandre Santos
(*)	
D278	Apostila de Teoria para Circuitos Digitais
(*)	/ Alexandre Santos de la Vega. – Niterói:
2019	UFF/TCE/TET, 2019.
	261 (sem romanos) ou 283 (com romanos) p. (*)
	Apostila de Teoria – Graduação, Engenharia de
	Telecomunicações, UFF/TCE/TET, 2019.
	1. Circuitos Digitais. 2. Técnicas Digitais.
	3. Telecomunicações. I. Título.

(*) OBTER INFO NA BIBLIOTECA E ATUALIZAR !!!

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 intitulado Apostila com Códigos de Programas Demonstrativos 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 bemvindas.

Rio de Janeiro, 08 de setembro de 2010.

Alexandre Santos de la Vega

UFF/TCE/TET

Agradecimentos

Aos alunos do Curso de Engenharia de Telecomunicações e aos professores do Departamento de Engenharia de Telecomunicações (TET), da Universidade Federal Fluminense (UFF), que colaboraram com críticas e sugestões bastante úteis à finalização deste trabalho. Em particular, à professora Carmen Maria Costa de Carvalho pela leitura meticulosa da versão original.

Aos funcionários e 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 tentar melhorar, em todos os sentidos.

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

Rio de Janeiro, 08 de setembro de 2010.

Alexandre Santos de la Vega

UFF/TCE/TET

Sumário

Prefácio	v
Agradecimentos	vii
Sumário	ix
Lista de Tabelas	xvii
Lista de Figuras	xix
I Apresentação	1
1 Descrição do documento	3
1.1 Introdução	3
1.2 Circuitos digitais combinacionais	3
1.3 Circuitos digitais sequenciais	4
1.4 Circuitos digitais programáveis	4
1.5 Tópicos adicionais	4
II Circuitos digitais combinacionais	5
2 Conceitos básicos	7
2.1 Telecomunicações	7
2.2 Sinais e sistemas	7
2.3 Sinais e sistemas digitais	8
2.4 Computação	8
2.5 Informação, quantidade e circuitos digitais	9
3 Funções lógicas	11
3.1 Introdução	11
3.2 Mecanismos básicos de raciocínio	11
3.2.1 Raciocínio por indução (ou por analogia)	12
3.2.2 Raciocínio por dedução	12
3.2.3 Estruturas axiomáticas	12
3.2.4 Classificação das lógicas dedutivas	13
3.3 Exemplo introdutório usando lógica clássica	14
3.4 Conceitos básicos	15
3.5 Formulação lógica clássica	15

3.6	Operadores lógicos clássicos	16
3.6.1	Funções de 1 variável	16
3.6.2	Funções de 2 variáveis	17
3.6.3	Funções de $N_v > 2$ variáveis	17
3.6.4	Operadores lógicos básicos	18
3.7	Relações de implicação	18
3.8	Tautologia e equivalência lógica	19
3.9	Conjunto funcionalmente completo de operadores	20
3.10	Teoremas de De Morgan	21
3.11	Decomposição em funções canônicas	21
3.12	Blocos funcionais fundamentais	23
3.13	Manipulação algébrica de blocos	23
3.14	Conectivos de ordem superior	26
3.15	Técnica de <i>bundling</i>	26
3.16	Exemplos de aplicação direta de portas lógicas	27
3.16.1	Uso de operador lógico como elemento de controle	27
3.16.2	Identificador de paridade e gerador de paridade	28
3.16.3	Identificador de igualdade entre padrões binários	28
3.17	Exercícios propostos	29
4	Álgebra de Boole	31
4.1	Introdução	31
4.2	Postulados de Huntington	32
4.3	Versão alternativa para os Postulados de Huntington	33
4.4	Dualidade	33
4.5	Lemas e teoremas fundamentais	34
4.6	Definição de uma estrutura algébrica particular	35
4.7	Exemplos de associação com a estrutura algébrica de Boole	35
4.8	Isomorfismo	36
4.9	Simplificação algébrica de expressões lógicas	36
4.10	Exemplo de manipulação algébrica não sistemática: Postulados, Lemas e Teoremas	36
4.11	Exemplo de manipulação algébrica por isomorfismo: Diagrama de Venn	39
4.12	Resumo das relações algébricas	40
4.13	Exercícios propostos	42
5	Formas padrões para representação de expressões booleanas	45
5.1	Introdução	45
5.2	Definições	46
5.3	Obtenção de formas SOP e POS padrões	47
5.3.1	Complementação da lista de termos canônicos	47
5.3.2	Manipulação algébrica	48
5.3.3	Utilização de tabela verdade	50
5.3.4	Negação das formas SOP e POS	52
5.4	Conjuntos de formas padrões	53
5.4.1	Definição	53
5.4.2	Obtenção	53
5.4.3	Utilização	53
5.5	Exercícios propostos	55

6	Simplificação algébrica sistemática de expressões booleanas	57
6.1	Definição do ponto final: forma mínima procurada	57
6.2	Processo de expansão \times processo de simplificação	58
6.2.1	Expansão sem redundância	58
6.2.2	Expansão com redundância	59
6.2.3	Análise das expansões	59
6.3	Definição do ponto inicial: forma máxima padrão	60
6.4	Simplificação sistemática de expressões booleanas a partir de SOP e POS padrões	60
6.4.1	Operações básicas: aglutinação e replicação	60
6.4.2	Uso da aglutinação	61
6.4.3	Uso da replicação	61
6.5	Eliminação sistemática de literais	62
6.6	Implicantes e implicados	65
6.6.1	Implicantes	65
6.6.2	Implicados	65
6.6.3	Implicantes, implicados e o processo de simplificação	66
6.7	Processo sistemático de simplificação	66
6.8	Subjetividade, complexidade e formas alternativas	67
6.9	Minimização com estruturas de ordem superior:fatoração	67
6.10	Exercícios propostos	69
7	Mapa de Karnaugh	71
7.1	Introdução	71
7.2	Construção do mapa-K	72
7.2.1	Funções de 1 variável	72
7.2.2	Funções de 2 variáveis	73
7.2.3	Funções de 3 variáveis	74
7.2.4	Funções de 4 variáveis	75
7.3	Preenchimento do mapa-K	76
7.4	Mapa-K como forma de expressão de função booleana	76
7.5	Mapa-K na simplificação de expressões booleanas	78
7.5.1	Adjacência lógica, aglutinação e replicação	78
7.5.2	Seleção sistemática de termos (implicantes ou implicados)	79
7.5.3	Mapa-K de funções com múltiplos mínimos e mapa cíclico	79
7.5.4	Indeterminações: <i>don't-care</i> e <i>can't-happen</i>	80
7.6	Exercícios propostos	82
8	Sistemas de numeração	85
8.1	Introdução	85
8.2	Sistema de numeração posicional convencional	87
8.2.1	Representação de números inteiros não negativos	87
8.2.2	Representação de números fracionários não negativos	88
8.2.3	Representação de números inteiros negativos	89
8.2.4	Representação de números fracionários negativos	99
8.2.5	Conversão entre bases	101
8.2.6	Bases mais comuns em circuitos digitais	103
8.3	Quantização	105
8.3.1	Conceitos básicos	105
8.3.2	SNPC: resolução, base e quantidade de dígitos	105

8.3.3	Classificações	107
8.4	Códigos numéricos	107
8.4.1	Introdução	107
8.4.2	Códigos numéricos comuns	107
8.4.3	Outros códigos numéricos	107
8.4.4	Códigos BCD	109
8.5	Aritmética binária	112
8.5.1	Tabelas de operações básicas entre dígitos	112
8.5.2	Escalamento por potência inteira da base	113
8.5.3	Adição e subtração em complemento-a-2	114
8.5.4	Funções envolvidas na adição de dígitos binários	116
8.6	Exercícios propostos	119
9	Circuitos combinacionais básicos	121
9.1	Introdução	121
9.2	Interpretações dos circuitos combinacionais	122
9.2.1	Exemplos de interpretações	122
9.3	Uso de portas lógicas como elementos de controle	123
9.4	Uso de elementos de controle para mascaramento	123
9.5	Gerador de funções lógicas	123
9.6	Conversor de códigos	123
9.7	Gerador e detector de paridade	124
9.8	Multiplexador e demultiplexador	124
9.9	Codificador e decodificador de endereço	124
9.10	Codificador de prioridade	125
9.11	Ordenador binário	125
9.12	Deslocadores (<i>shifters</i>)	125
9.13	Somadores em binário puro	126
9.13.1	<i>Half-adder</i> (HA)	126
9.13.2	<i>Full-adder</i> (FA)	126
9.13.3	<i>Ripple-carry adder</i> (RCA) ou <i>carry propagate adder</i> (CPA)	127
9.13.4	<i>Carry lookahead adder</i> (CLA)	127
9.14	Subtratores em binário puro	129
9.14.1	<i>Half-subtractor</i> (HS)	129
9.14.2	<i>Full-subtractor</i> (FS)	129
9.14.3	<i>Ripple-borrow subtractor</i> (RBS) ou <i>borrow propagate subtractor</i> (BPS)	129
9.15	Incrementador e decrementador em binário puro	129
9.16	Complementadores	130
9.16.1	Complementador-a-1 (<i>bitwise implementation</i>)	130
9.16.2	Complementador-a-2	130
9.17	Multiplicadores em binário puro	131
9.17.1	Multiplicador de 1 <i>bit</i>	131
9.17.2	Multiplicador de N <i>bits</i>	131
9.18	Comparadores da quantidade de dígitos	131
9.18.1	Comparador da quantidade de dígitos em um operando	131
9.18.2	Comparador da quantidade de dígitos em dois operandos	131
9.19	Comparadores numéricos de dois operandos	132
9.19.1	Identificador de igualdade	132
9.19.2	Projeto modular (<i>bit scanning</i>)	132

9.19.3 Projeto usando a técnica de complemento	133
--	-----

III Circuitos digitais seqüenciais 135

10 Circuitos seqüenciais: conceitos básicos	137
10.1 Introdução	137
10.2 Estados e variáveis de estado	137
10.3 Tipos de variáveis e suas interações	138
10.4 Modelo genérico para circuitos seqüenciais	139
10.5 Classificação de circuitos seqüenciais quanto à dependência do sinal de saída	140
10.6 Classificação de circuitos seqüenciais quanto ao tipo de controle da mudança de estado	141
10.6.1 Circuitos seqüenciais <i>clock-mode</i> ou <i>clocked</i>	141
10.6.2 Circuitos seqüenciais <i>pulsed</i>	142
10.6.3 Circuitos seqüenciais <i>level-mode</i>	142
11 Elementos básicos de armazenamento	145
11.1 Introdução	145
11.2 Classificação quanto à funcionalidade	146
11.3 Relacionamento entre os tipos básicos de <i>flip-flops</i>	147
11.4 Mapas de excitação dos <i>flip-flops</i>	149
11.5 Tipos de comportamento das saídas dos <i>flip-flops</i>	149
11.6 Excitação × comportamento	150
11.7 Funcionalidade × excitação × comportamento	150
11.8 Circuitos seqüenciais × tabelas dos <i>flip-flops</i>	151
11.9 Estruturas estáticas simétricas	153
11.10 Exemplos de <i>flip-flops</i>	154
11.10.1 <i>Flip-flops</i> do tipo <i>unclocked</i>	154
11.10.2 <i>Flip-flops</i> do tipo <i>clocked</i>	157
11.11 Variações de funcionalidade	162
11.12 Diferenças de nomenclatura	162
12 Circuitos seqüenciais <i>clock-mode</i>	163
12.1 Introdução	163
12.2 Controle de circuitos do tipo <i>clock-mode</i>	164
12.2.1 Características da estrutura <i>clock-mode</i>	164
12.2.2 Controle de circuitos do tipo Moore	164
12.2.3 Controle de circuitos do tipo Mealy	165
12.3 Representação dos estados	166
12.4 Estado inicial	166
12.5 Classificação quanto à capacidade de memorização	166
12.6 Análise de circuitos seqüenciais	168
12.6.1 Etapas de análise	168
12.6.2 Exemplos de análise	168
12.7 Projeto de circuitos seqüenciais	169
12.7.1 Opções de projeto e suas características	169
12.7.2 Etapas de projeto de circuitos seqüenciais	169
12.7.3 Exemplos de projeto de circuitos seqüenciais	170
12.8 Minimização de estados	172

12.8.1	Conceitos básicos	172
12.8.2	Eliminação de estados redundantes por simples inspeção	172
12.8.3	Método da partição em classes de estados indistinguíveis (método de Huffman-Mealy)	174
12.8.4	Método da tabela de implicação de estados (método de Paul-Unger)	177
12.9	Atribuição de estados	179
12.9.1	Considerações iniciais	179
12.9.2	Base teórica para as regras de atribuição de estados	180
12.9.3	Exemplo de regras simples (Armstrong-Humphrey)	186
12.9.4	Exemplo de regras mais refinadas	187
12.10	Efeitos causados por estados extras	187
12.10.1	Definição do problema	187
12.10.2	Possíveis soluções	188
13	Circuitos seqüenciais <i>pulsed</i>	189
13.1	Introdução	189
13.2	Restrições de operação	190
13.3	Classificação quanto aos pulsos de entrada	191
13.4	Circuitos <i>pulse-mode</i>	192
13.4.1	Motivação	192
13.4.2	Mudanças nas representações	192
13.4.3	Exemplos de projeto	194
13.5	Circuitos <i>ripple-clock</i>	195
13.5.1	Motivação	195
13.5.2	Operação	195
13.5.3	Desvantagens	195
13.5.4	Técnica de projeto	195
13.5.5	Exemplo	196
13.6	Circuitos <i>controlled-clock</i>	196
14	Circuitos seqüenciais <i>level-mode</i>	199
14.1	Introdução	199
14.2	Problemas comuns em circuitos <i>level-mode</i>	200
14.3	Exemplo de análise de circuito <i>level-mode</i>	201
14.4	Exemplo de projeto de circuito <i>level-mode</i>	201
14.5	Problemas causados pela realimentação contínua	202
14.5.1	Problemas causados pelo bloco de lógica combinacional	202
14.5.2	Problema natural dos circuitos <i>level-mode</i>	202
14.6	Solução para as corridas: atribuição de estados	203
14.6.1	Definição do problema	203
14.6.2	Possíveis soluções	205
14.7	Solução para os perigos	208
14.8	Valores das saídas em estados instáveis	209
IV	Circuitos digitais programáveis	211
15	Circuitos programáveis	213
15.1	Introdução	213
15.2	Circuitos configuráveis externamente	214

15.2.1	Configuração de portas lógicas	214
15.2.2	Configuração de funções lógicas com multiplexador	215
15.3	Circuitos configuráveis internamente: PLDs	218
V	Apêndices	219
A	Noções básicas sobre implementação de funções lógicas	221
A.1	Introdução	221
A.2	Conceitos básicos	221
A.3	Famílias lógicas	222
A.4	Modelo de chaves	222
A.4.1	Conceitos básicos	222
A.4.2	Arranjos série e paralelo de chaves	224
A.4.3	Modelo de chaves para a função NOT	225
A.4.4	Modelo de chaves para a função NAND	227
A.4.5	Modelo de chaves para a função NOR	229
A.4.6	Modelo de chaves para arranjos AOI e OAI	231
A.4.7	Modelo de chaves complementares genérico	234
A.4.8	Saída em <i>Tri-State</i> (3S)	235
B	Tópicos sobre divisão de números inteiros	237
B.1	Algoritmo de divisão inteira	237
B.2	Quociente	237
B.3	Resto ou resíduo	237
B.4	Congruência	237
B.5	Relações úteis	238
C	Minimização de tabela de estados	239
C.1	Introdução	239
C.2	Tabelas de estados completamente especificadas	240
C.2.1	Relações de equivalência	240
C.2.2	Estados e circuitos equivalentes	240
C.2.3	Determinação de classes de estados indistinguíveis	241
C.2.4	Circuito de classes de equivalência	241
C.3	Tabelas de estados não completamente especificadas	242
C.3.1	Introdução	242
C.3.2	Noções básicas de compatibilidade	242
C.3.3	Formalização dos conceitos de compatibilidade e de cobertura	243
C.3.4	Sistematização do processo de minimização	244
D	Introdução à linguagem VHDL	245
D.1	Introdução	245
D.1.1	Linguagens de descrição de <i>hardware</i>	245
D.1.2	Linguagem VHDL	247
D.2	Níveis de abstração	247
D.3	Abordagem hierárquica	248
D.4	VHDL como linguagem	248
D.4.1	Considerações gerais	248
D.4.2	Palavras reservadas	249

D.4.3	Identificadores definidos pelo usuário	250
D.4.4	Elementos sintáticos	250
D.5	Conceitos básicos sobre o código VHDL	251
D.5.1	Elementos básicos	251
D.5.2	Tipos de execução	251
D.5.3	Mecanismo genérico de simulação	252
D.6	Estrutura do código VHDL	253
D.6.1	Bibliotecas e pacotes	253
D.6.2	Entidade	254
D.6.3	Arquitetura	254
D.7	Algumas regras sintáticas de VHDL	255
D.7.1	Regras para biblioteca	255
D.7.2	Regras para pacote	255
D.7.3	Regras para entidade	255
D.7.4	Regras para arquitetura	256
D.7.5	Regras para processo	256
D.7.6	Regras para componente	257
D.8	Exemplos de declarações genéricas	257
D.8.1	Exemplos de biblioteca e de pacote	257
D.8.2	Exemplos de entidade	257
D.8.3	Exemplos de arquitetura	258
D.8.4	Exemplos de processo	259

Lista de Tabelas

3.1	Tabela de funções lógicas de uma variável.	16
3.2	Tabela de operadores lógicos de 1 variável.	16
3.3	Tabela de funções lógicas de duas variáveis.	17
3.4	Tabela de operadores lógicas de duas variáveis.	17
3.5	Exemplos de Tautologias ou Leis da Lógica.	19
3.6	Pares de proposições equivalentes, definindo os operadores OR, IMPLICA e XNOR, em função dos operadores AND e NOT.	19
3.7	Pares de proposições equivalentes, definindo os operadores AND, IMPLICA e XNOR, em função dos operadores OR e NOT.	19
3.8	Pares de proposições equivalentes, definindo os operadores AND, OR e XNOR, em função dos operadores IMPLICA e NOT.	19
3.9	Tabela de funções canônicas (mintermos e maxtermos) para duas variáveis.	21
3.10	Exemplo de decomposição em funções canônicas (mintermos e maxtermos).	22
3.11	Uso de operador lógico como elemento de controle.	27
4.1	Tabela de mapeamento: Cálculo Proposicional \times Álgebra de Boole.	35
4.2	Tabela de mapeamento: Teoria de Conjuntos \times Álgebra de Boole.	35
4.3	Resumo dos postulados de Huntington para a estrutura algébrica de Boole.	40
4.4	Resumo dos lemas para a estrutura algébrica de Boole.	40
4.5	Resumo dos teoremas para a estrutura algébrica de Boole.	41
4.6	Resumo da definição de uma estrutura algébrica de Boole particular.	41
4.7	Resumo das relações de isomorfismo.	41
5.1	Definição de mintermos para três variáveis (A,B,C).	47
5.2	Definição de maxtermos para três variáveis (A,B,C).	47
5.3	Exemplo de função e associação de mintermos.	50
5.4	Exemplo de função e definição de maxtermos.	51
5.5	Exemplo da obtenção do grupo AND-OR para a função XOR.	54
5.6	Exemplo da mudança de grupo para a função XOR.	54
5.7	Exemplo da obtenção do grupo OR-AND para a função XOR.	54
6.1	Tabela verdade genérica para funções de 3 variáveis.	62
7.1	Tabela verdade para funções de 1 variável.	72
7.2	Tabela verdade para funções de 2 variáveis.	73
7.3	Tabela verdade para funções de 3 variáveis.	74
7.4	Tabela verdade para funções de 4 variáveis.	75
7.5	Tabela verdade relativa à Equação (7.1).	77
7.6	Tabela verdade de função incompletamente especificada.	80
8.1	Tabela de sinal-e-magnitude, para número inteiros, $b = 2$ e $N = 4$	94
8.2	Tabela de complemento-a-1, para número inteiros, $b = 2$ e $N = 4$	96
8.3	Tabela de complemento-a-2, para número inteiros, $b = 2$ e $N = 4$	98
8.4	Tabela de sinal-e-magnitude, para números puramente fracionários, $b = 2$ e $N = 4$	99

8.5	Tabela de complemento-a-1, para números puramente fracionários, $b = 2$ e $N = 4$.	100
8.6	Tabela de complemento-a-2, para números puramente fracionários, $b = 2$ e $N = 4$.	100
8.7	Alguns valores de níveis em um SNPC, em função do número de dígitos empregados, para as bases	
8.8	Alguns valores de resolução numérica em um SNPC, em função do número de dígitos empregados,	
8.9	Efeito do pequeno aumento da resolução numérica em um SNPC, em função do número de dígitos	
8.10	Relação entre os códigos numéricos decimal, binário, octal e hexadecimal, para um código binário	
8.11	Relação entre os códigos numéricos decimal, binário, Gray, <i>One-hot</i> e Johnson, para um código binário	
8.12	Relação entre os dígitos decimais e os códigos BCD 8421, <i>Excess-3</i> e Mid-Gray.	109
8.13	Diferentes versões do código numérico BCD 631(-1).	109
8.14	Códigos numéricos BCD com mais de 4 bits: <i>2-out-of-5</i> e <i>Biquinary</i> .	110
8.15	Outros códigos numéricos BCD ponderados: 7421, 5421, 5311, 4221 e 2421.	111
8.16	Exemplos de códigos numéricos BCD não ponderados.	111
8.17	Forma 1 para representar a detecção de <i>overflow</i> na adição em complemento-a-2.	115
8.18	Forma 2 para representar a detecção de <i>overflow</i> na adição em complemento-a-2.	115
10.1	Tipos de interações entre sinais dos tipos nível e pulso.	138
11.1	Transformações envolvendo <i>flip-flops</i> dos tipos <i>JK</i> , <i>D</i> , T_1 e T_2 .	148
11.2	Definição dos tipos de comportamento apresentados pela saída de um <i>flip-flop</i> .	149
11.3	Tabela resumo de funcionalidade-excitação-comportamento para os <i>flip-flops</i> SR, JK, D e T_2 .	150
11.4	Tabela de mudanças de estado e de comportamento dos elementos de memória para um contador	
11.5	Operação das estruturas de armazenamento estáticas e simétricas controladas por meio de portas	
11.6	Diferentes nomenclaturas para <i>flip-flops</i> .	162
12.1	Número de atribuições de estados efetivamente diferentes.	179
14.1	Atribuição de estados universal, usando shared-row, para tabelas de 3 estados.	206
14.2	Atribuição de estados universal, usando multiple-row, para tabelas de 4 estados.	207
14.3	Atribuição de estados universal, usando shared-row, para tabelas de 5 a 8 estados.	207
14.4	Atribuição de estados universal, usando shared-row, para tabelas de 9 a 12 estados.	207
14.5	Atribuição de estados padrão, usando shared-row, para tabelas de 5 estados.	207
15.1	Classificação de circuitos digitais de acordo com a função realizada.	214
15.2	Comportamento funcional dos dispositivos NC7SZ57 e NC7SZ58.	215
15.3	Opções de configuração dos dispositivos NC7SZ57 e NC7SZ58.	215
A.1	Tipos básicos de associação entre valores de tensão e valores lógicos binários.	222
A.2	Comportamento elétrico do modelo de chaves NOT.	226
A.3	Comportamento lógico do modelo de chaves NOT.	226
A.4	Comportamento elétrico do modelo de chaves NAND.	227
A.5	Comportamento lógico do modelo de chaves NAND.	228
A.6	Comportamento elétrico do modelo de chaves NOR.	229
A.7	Comportamento lógico do modelo de chaves NOR.	230
A.8	Comparação do comportamento dos blocos de chaves N e P no modelo de chaves da função lógica	
A.9	Comparação do comportamento dos blocos de chaves N e P no modelo de chaves da função lógica	
A.10	Comparação do comportamento dos blocos de chaves N e P no modelo de chaves da função lógica	
D.1	Lista de fabricantes, produtos e funções, que lidam com HDL.	246
D.2	Operadores de VHDL.	251

Lista de Figuras

3.1	Blocos funcionais fundamentais, associados aos operadores lógicos básicos.	23
3.2	Manipulação algébrica de blocos: exemplo 1.	24
3.3	Manipulação algébrica de blocos: exemplo 2.	25
3.4	Uso de operador lógico como elemento de controle: simbologia genérica.	27
4.1	Mapeamento entre uma função genérica de duas variáveis e um Diagrama de Venn.	39
6.1	Eliminações de 1 literal em combinações de 2 mintermos.	63
6.2	Eliminações de 2 literais em combinações 4 de mintermos.	63
6.3	Eliminações de 1 literal em combinações de 2 maxtermos.	64
6.4	Eliminações de 2 literais em combinações 4 de maxtermos.	64
7.1	Exemplos de mapas de Karnaugh para funções de 1 variável.	72
7.2	Exemplos de mapas de Karnaugh para funções de 2 variáveis.	73
7.3	Exemplos de mapas de Karnaugh para funções de 3 variáveis.	74
7.4	Exemplos de mapas de Karnaugh para funções de 4 variáveis.	76
7.5	Mapa de Karnaugh relativo à Equação (7.1).	77
7.6	Mapa de Karnaugh relativo à Equação (7.6).	78
7.7	Mapa de Karnaugh relativo à Equação (7.7).	78
7.8	Mapa de Karnaugh com múltiplas formas mínimas.	80
7.9	Mapa de Karnaugh com ciclo.	80
7.10	Mapa de Karnaugh da Tabela 7.6.	81
7.11	Mapa de Karnaugh dos mintermos da Tabela 7.6.	81
7.12	Mapa de Karnaugh dos maxtermos da Tabela 7.6.	81
8.1	Representação de quantidades $q < b$, para $b = 3$	87
8.2	Representação de quantidades $q \geq b$, para $b = 3$, com ambigüidade.	87
8.3	Representação de quantidades $q \geq b$, para $b = 3$, com eliminação da ambigüidade através da junção.	87
8.4	Uso repetido da técnica de justaposição de dígitos para representação de quantidades $q \geq b$, para $b = 3$	87
8.5	Mapeamento decimal-binário para números não negativos e $N = 4$, sem sinal.	91
8.6	Mapeamento decimal-binário para números não negativos e $N = 4$, com sinal.	91
8.7	Mapeamento decimal-binário para números negativos e $N = 4$, com codificação Sinal-e-Magnitudinal.	91
8.8	Mapeamento decimal-binário para números negativos e $N = 4$, com codificação Complemento-1.	91
8.9	Mapeamento decimal-binário para números negativos e $N = 4$, com codificação Complemento-2.	91
8.10	Comparação dos mapeamentos decimal-binário para números negativos e $N = 4$	93
8.11	Tabelas de operações entre dígitos para $b = 2$: (a) adição e (b) multiplicação.	112
8.12	Tabelas de operações entre dígitos para $b = 3$: (a) adição e (b) multiplicação.	112
8.13	Tabelas de operações entre dígitos para $b = 4$: (a) adição e (b) multiplicação.	112
8.14	Tabelas que definem a adição entre dois dígitos binários (A e B): (a) adição completa, (b) “vai embora”.	112
8.15	Tabelas que definem a adição entre três dígitos binários (A , B e C_i): (a) adição completa, (b) “vai embora”.	112
8.16	Tabelas que definem três funções úteis na implementação de somadores binários com algoritmo de soma.	112

10.1	Modelo genérico para circuitos seqüenciais.	139
10.2	Exemplo de máquina de Mealy.	140
10.3	Exemplo de máquina de Moore.	141
10.4	Modelo genérico para circuitos seqüenciais <i>clock-mode</i>	142
10.5	Modelo genérico para circuitos seqüenciais <i>pulsed</i>	143
10.6	Modelo genérico para circuitos seqüenciais <i>level-mode</i>	143
11.1	Tabelas de operação básica para os <i>flip-flops</i> SR, JK, D e T_2	147
11.2	Mapas de excitação para os <i>flip-flops</i> SR, JK, D e T_2	149
11.3	Tipos de comportamento e respectivas excitações para os <i>flip-flops</i> SR, JK, D e T_2	150
11.4	Mapas-K de transição para os elementos de memória de um contador binário, crescente, de três <i>bits</i>	151
11.5	Mapas-K de excitação para os <i>flip-flops</i> JK de um contador binário, crescente, de três <i>bits</i>	152
11.6	Estrutura de armazenamento estática e simétrica, não controlável.	153
11.7	Estruturas de armazenamento estáticas e simétricas, controláveis por chaves.	154
11.8	Uso de portas lógicas NOR na implementação de controle em uma estrutura de armazenamento estática.	154
11.9	Uso de portas lógicas NAND na implementação de controle em uma estrutura de armazenamento estática.	154
11.10	Exemplo de implementação de <i>flip-flop</i> SR do tipo <i>clocked</i> elementar, usando portas lógicas NOR.	154
11.11	Exemplo de implementação de <i>flip-flop</i> SR do tipo <i>clocked</i> elementar, usando portas lógicas NAND.	154
11.12	Exemplo de implementação de <i>flip-flop</i> D do tipo <i>clocked</i> elementar, com base em um <i>flip-flop</i> SR.	154
11.13	Técnica de <i>pipelining</i> : (a) Bloco funcional original e (b) Bloco com <i>pipelining</i>	159
11.14	Exemplo de implementação de <i>flip-flop</i> D do tipo <i>master-slave</i> , com base em <i>flip-flops</i> SR.	159
11.15	Exemplo de implementação de <i>flip-flop</i> JK, a partir de <i>flip-flop</i> SR <i>unclocked</i> , com problema de oscilação.	160
11.16	Exemplo de implementação de <i>flip-flop</i> JK, a partir de <i>flip-flop</i> SR <i>clocked</i> , com problema de oscilação.	160
11.17	Exemplo de implementação de <i>flip-flop</i> JK, a partir de <i>flip-flop</i> SR <i>clocked</i> , sem problema de oscilação.	160
11.18	Exemplo 1 de implementação de <i>flip-flop</i> JK do tipo <i>master-slave</i>	161
11.19	Exemplo 2 de implementação de <i>flip-flop</i> JK do tipo <i>master-slave</i>	161
12.1	Modelo genérico para circuitos seqüenciais <i>clock-mode</i>	163
12.2	Modelo genérico para circuitos com memória finita.	167
12.3	Modelo genérico para circuitos com memória de entrada finita.	167
12.4	Modelo genérico para circuitos com memória de saída finita.	168
12.5	Fluxos de projeto para circuitos seqüenciais <i>clock-mode</i> : (a) Fluxo genérico, (b) Caso particular de projeto.	168
12.6	Eliminação de estados redundantes através da inspeção da tabela de estados.	173
12.7	Exemplo de minimização positiva em um passo.	175
12.8	Exemplo de minimização negativa em um passo.	175
12.9	Exemplo de minimização positiva em mais de um passo.	176
12.10	Tabela de implicação genérica do método de Paul-Unger.	178
12.11	Célula genérica da tabela do método de Paul-Unger.	178
12.12	Análise de minimização para as equações de excitação e de saída: mapa de Karnaugh simbólico.	178
12.13	Análise de minimização para as equações de excitação e de saída: tabela de atribuição de estados lógicos.	178
12.14	Análise de minimização para as equações de excitação: casos de estados atuais com mesmo próximo estado.	178
12.15	Análise de minimização para as equações de excitação: casos de estado atual com próximos estados.	178
12.16	Análise de minimização para as equações de saída.	185
12.17	Ilustração das regras de Armstrong-Humphrey.	186
13.1	Modelo genérico para circuitos seqüenciais <i>pulsed</i>	189
13.2	Equivalência de notações para mapa de Karnaugh utilizado na síntese de variáveis pulsadas.	193
13.3	Tabelas de estados para circuitos <i>pulse-mode</i> Mealy e Moore.	193
13.4	Mapas de Karnaugh para síntese de variáveis pulsadas, considerando-se duas entradas pulsadas: (a) Mapa de Karnaugh para a saída.	193
13.5	Mapas de Karnaugh para síntese de variáveis pulsadas, considerando-se três entradas pulsadas: (a) Mapa de Karnaugh para a saída.	193

13.6	Exemplo 1 de controle de sinal de <i>clock</i>	196
13.7	Exemplo 2 de controle de sinal de <i>clock</i>	196
13.8	Modelo genérico para circuitos seqüenciais <i>controlled-clock</i>	198
14.1	Modelo genérico para circuitos seqüenciais <i>level-mode</i>	199
14.2	Padrões de identificação de perigo essencial em tabelas de fluxo.	203
14.3	Quadro resumo das mudanças de estado nos circuitos seqüenciais <i>level-mode</i> , operando em mo	
15.1	Mapas de Karnaugh da função exemplo, para implementação usando MUX.	217
A.1	Chaves simples ou <i>Single-Pole Single-Throw</i> (SPST). (a) Chave-N. (b) Chave-P.	223
A.2	Modelo comportamental para a relação de complementariedade no acionamento, entre as chav	
A.3	Arranjo série de chaves simples. (a) Chave-N. (b) Chave-P.	224
A.4	Arranjo paralelo de chaves simples. (a) Chave-N. (b) Chave-P.	224
A.5	Modelo de chaves simples para uma implementação da função lógica NOT, onde: (a) chave N	
A.6	Modelo de chaves complementares para uma implementação da função lógica NOT.225	
A.7	Modelo de chaves simples para uma implementação da função lógica NAND, onde: (a) chave N	
A.8	Modelo de chaves complementares para uma implementação da função lógica NAND.228	
A.9	Modelo de chaves simples para uma implementação da função lógica NOR, onde: (a) chave N	
A.10	Modelo de chaves complementares para uma implementação da função lógica NOR.230	
A.11	Modelo de chaves complementares para uma implementação do arranjo AOI definido por $f(A,$	
A.12	Modelo de chaves complementares para uma implementação do arranjo OAI definido por $f(A,$	
A.13	Modelo de chaves complementares para uma implementação de função lógica NOT com saída	
D.1	Palavras reservadas de VHDL.	249
D.2	Símbolos especiais de VHDL.	250

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 sequenciais.

1.2 Circuitos digitais combinacionais

- Conceitos básicos: busca contextualizar a disciplina no âmbito do curso de graduação e apresentar os conceitos que serão necessários ao longo do texto.
- Funções lógicas: que define as bases para a representação de informações não numéricas em circuitos digitais tradicionais.
- Álgebra de Boole: que apresenta um formalismo matemático para a estrutura algébrica da lógica implementada em circuitos digitais tradicionais.
- Formas padrões para representação de expressões booleanas: que define formas de expressões booleanas adequadas a um processo sistemático de simplificação das mesmas.
- Simplificação algébrica de expressões booleanas: que ilustra um processo algébrico para a simplificação sistemática de expressões booleanas.
- Mapa de Karnaugh: que apresenta uma ferramenta tabular para a simplificação sistemática de expressões booleanas.
- Sistemas de numeração: que define as bases para a representação de quantidades numéricas em circuitos digitais.

- Circuitos combinacionais básicos: que apresenta os blocos funcionais combinacionais básicos utilizados em sistemas digitais.

1.3 Circuitos digitais sequenciais

- Conceitos básicos: busca contextualizar a disciplina no âmbito do curso de graduação e apresentar os conceitos que serão necessários ao longo do texto.
- Elementos básicos de armazenamento: apresenta os elementos de armazenamento utilizados nos circuitos seqüenciais abordados neste texto.
- 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.
- 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.
- 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.

1.4 Circuitos digitais programáveis

- Circuitos programáveis: apresenta uma breve descrição sobre circuitos digitais que podem ser configurados para cumprir diferentes funções, combinacionais e/ou seqüenciais.

1.5 Tópicos adicionais

- Noções básicas sobre implementação de funções lógicas: que lista as principais famílias lógicas utilizadas na implementação de circuitos digitais e apresenta a implementação de portas lógicas básicas empregando um arranjo complementar de chaves complementares.
- Tópicos sobre divisão de números inteiros: que aborda brevemente alguns itens sobre divisão de números inteiros, citados ao longo do texto.
- Minimização de tabelas de estados: define o problema e apresenta técnicas de minimização para tabelas de estados.
- Introdução à linguagem VHDL: trata de aspectos básicos da linguagem.

Parte II

Circuitos digitais combinacionais

Capítulo 2

Conceitos básicos

O conteúdo que é abordado no presente material didático tem sido utilizado em disciplinas ministradas nos seguintes cursos de graduação: Engenharia de Telecomunicações e Computação. Esse capítulo busca contextualizar tal conteúdo em tais cursos e tratar de alguns conceitos básicos utilizados ao longo do material.

2.1 Telecomunicações

- Telecomunicações: definição e exemplos.
- Sistemas de Telecomunicações: definição, anatomia e exemplos.

2.2 Sinais e sistemas

- Sinais: definição.
- Classificação de sinais: analógico, amostrado, quantizado e digital.
- Sistemas: definição, anatomia (variáveis, elementos e topologia), análise \times projeto.
- Relação: circuito \times sistema.
- Classificação de sistemas:
 - parâmetros concentrados \times distribuídos.
 - parâmetros constantes no tempo, fixo ou invariante no tempo \times parâmetros variáveis no tempo, variável ou variante no tempo.
 - linear \times não linear.
 - instantâneo ou sem memória \times dinâmico ou com memória.
 - contínuo \times discreto (analógico/amostrado/quantizado/digital).
- Sistemas dinâmicos: estado e variáveis de estado.

2.3 Sinais e sistemas digitais

- Discretização: definição.
- Tipos de discretização: uniforme \times não uniforme.
- Discretização uniforme: resolução.
- Medição e armazenamento \rightarrow discretização de variáveis.
- Variável discretizada: amostragem \times quantização.
- Tipos de aproximação na quantização: truncamento, arredondamento e truncamento em magnitude.
- Anatomia de sistemas digitais:
 - Classes de sinais: sinais de dados, sinais de temporização, sinais de controle e alimentação.
 - Processamento digital dos sinais: digitais \times analógicos.
 - Processamento digital dos sinais analógicos: conversores A/D e D/A, filtro limitador de banda (*anti-aliasing filter*) e filtro de interpolação (*smoothing filter*).

2.4 Computação

- Hierarquia em *hardware* e em *software*:
 - Elementos básicos.
 - Blocos funcionais de complexidade básica, média e elevada.
 - Sistemas de complexidade básica, média e elevada.
- Codificação:
 - Código: sintaxe (símbolos) \times 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 (compressão) e do canal (redução de taxa de erro).
- Elementos codificados:
 - Informação (fatos, classificações) \times 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.

2.5 Informação, quantidade e circuitos digitais

- 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, já foi demonstrado 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 (estabilidade e robustez) de implementação, têm levado à escolha da representação através de mecanismos envolvendo 2 estados por dispositivo.
 - Assim, para informações envolvendo N estados, são necessários M dispositivos de 2 estados, tal que $2^M \geq N$.
- Representação de quantidade:
 - Representação numérica (quantizada) de uma informação analógica ou discreta.
 - Representação de informação analógica: amostragem e quantização.
 - Representação 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 de um conjunto, o qual 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, pode-se utilizar 100 símbolos fixos diferentes ou X símbolos variáveis, com Y valores cada.

- Circuitos digitais:
 - Um sistema de numeração significa, para a representação de quantidades numéricas, o mesmo que símbolos lógicos significam para a representação de informações não numéricas.
 - Utilizando-se uma simbologia que atenda a ambas as representações, pode-se implementar sistemas que manipulem quantidades numéricas e informações não numéricas sem distinção.
 - Tais sistemas, e seus circuitos, são denominados sistemas e circuitos digitais.

Capítulo 3

Funções lógicas

3.1 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.

3.2 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.

3.2.1 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.

3.2.2 Raciocínio por dedução

- O raciocínio por dedução envolve um conjunto de argumentos e um conjunto ou sistema de regras.
- Os argumentos 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.
- Os argumentos iniciais são denominados de premissas. Aplicando-se as regras sobre as premissas, procura-se demonstrar, ou deduzir, um argumento final, que é denominado 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. Caso contrário, ele é dito Inválido.

3.2.3 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.

3.2.4 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.

3.3 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: OR (\vee).
 - Exemplo: Banco do motorista OR banco do carona ($B_M \vee B_C$).
 - * Disjunção:
 - Função: AND (\wedge).
 - Exemplo: Ignição AND Marcha ($I \wedge M$).
 - * Equivalência:
 - Função: XNOR (\equiv).
 - Exemplo: Alarme XNOR “Modelo proposto” ($A \equiv MP$).
- Proposta de modelo: $A \equiv I \wedge \{M \wedge [(B_M \wedge \neg C_M) \vee (B_C \wedge \neg C_C)]\}$.

3.4 Conceitos básicos

- Elementos utilizados em uma formulação funcional: 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 por: tabelas (pequena quantidade de pontos), gráficos (grande quantidade de pontos) e equações (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.

3.5 Formulação lógica clássica

- Nesse texto, será abordado apenas um tipo de lógica: binária, bivalente ou clássica.
- Os argumentos serão representados por proposições.
- Uma proposição é uma sentença afirmativa declarativa (ou uma afirmação declarativa ou uma assertiva ou um *statement*), sobre a qual faz sentido se afirmar que a mesma é verdadeira ou falsa.
- Variáveis e valores fixos na lógica binária:
 - As variáveis representam assertivas (ou proposições ou argumentos).
 - 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 \rightarrow um sistema formado por um conjunto de assertivas (*statements*), associadas por meio de conectivos (operadores ou funções).
 - Conectivo \rightarrow elemento de conexão que é modelado por uma função lógica.
 - Assertiva \rightarrow afirmação declarativa (*statement*) sobre algum elemento do problema, a qual é representada por uma variável de asserção do sistema.
 - Variável de asserção \rightarrow variável do sistema associada a uma assertiva, à qual será atribuído um valor fixo lógico (*truth value*).
 - Valor fixo lógico \rightarrow 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 mutuamente excludentes, naturalmente surge a idéia de negação (da associação de assertivas, do conectivo, da assertiva, da variável de asserção, do valor ou do símbolo).

3.6 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 $V = f(V_1, V_2, \dots, V_{N_v})$.
- 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 assumir 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.

3.6.1 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 3.1. Os operadores lógicos associados às funções $X_i = f_i(A)$ são definidos na Tabela 3.2.

A	X_0	X_1	X_2	X_3
F	F	F	T	T
T	F	T	F	T

Tabela 3.1: Tabela de funções lógicas de uma variável.

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 (A) ou $(A)'$ ou $(A)^*$ ou $!(A)$	Negação lógica
X_3	(T)	(T)	Tautologia

Tabela 3.2: Tabela de operadores lógicos de 1 variável.

3.6.2 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 3.3. Os operadores lógicos associados às funções $X_i = f_i(A, B)$ são definidos na Tabela 3.4.

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 3.3: Tabela de funções lógicas de duas variáveis.

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)	($A \vee B$)
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)	$\neg(A \vee B)$ ou ($A \underline{\Delta} B$) 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 3.4: Tabela de operadores lógicas de duas variáveis.

3.6.3 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.

3.6.4 Operadores lógicos básicos

- Com base nas funções lógicas identificadas nas Tabelas 3.1 e 3.3, podem ser definidos os operadores lógicos encontrados nas Tabelas 3.2 e 3.4.
- Operadores unários:
 - Identidade lógica.
 - NOT (negação lógica).
- 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).
 - NOT-IMPLICA.

3.7 Relações de implicação

- Podem-se definir três relações de implicação de uma assertiva precedente A para uma assertiva conseqüente B :
 - Condição necessária:
 “SOMENTE SE $(A = T)$ ENTÃO $(B = T)$ ”
 ou
 “ $(B = T)$ SOMENTE SE $(A = T)$ ”.
 - Condição suficiente:
 “SE $(A = T)$ ENTÃO $(B = T)$ ”
 ou
 “ $(B = T)$ SE $(A = T)$ ”.
 - 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:
 - Condição necessária: $(A \leftarrow B)$.
 - Condição suficiente: $(A \rightarrow B)$.
 - Condição necessária e suficiente: $(A \rightarrow B) \wedge (A \leftarrow B) \equiv (A \leftrightarrow B) \equiv (A \equiv B)$.

3.8 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 3.5.

Lei	Nome
$p \vee \bar{p}$	Lei do meio excluído
$\overline{(p \wedge \bar{p})}$	Lei da contradição
$[(p \rightarrow q) \wedge (q \rightarrow r)] \rightarrow (p \rightarrow r)$	Lei do silogismo
$p \leftrightarrow \overline{(\bar{p})}$	Lei da dupla negação
$(p \rightarrow q) \leftrightarrow (\bar{q} \rightarrow \bar{p})$	Lei da contraposição

Tabela 3.5: Exemplos de Tautologias ou Leis da Lógica.

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 3.6, 3.7 e 3.8, 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 na próxima seção.

P_1	P_2
$p \vee q$	$\overline{(\bar{p} \wedge \bar{q})}$
$p \rightarrow q$	$\overline{(p \wedge \bar{q})}$
$p \leftrightarrow q$	$\overline{(p \wedge \bar{q}) \wedge (\bar{p} \wedge q)}$

Tabela 3.6: Pares de proposições equivalentes, definindo os operadores OR, IMPLICA e XNOR, em função dos operadores NOT e AND.

P_1	P_2
$p \wedge q$	$\overline{(\bar{p} \vee \bar{q})}$
$p \rightarrow q$	$\overline{(\bar{p} \vee q)}$
$p \leftrightarrow q$	$\overline{(\bar{p} \vee q) \vee (p \vee \bar{q})}$

Tabela 3.7: Pares de proposições equivalentes, definindo os operadores AND, IMPLICA e XNOR, em função dos operadores NOT e OR.

P_1	P_2
$p \wedge q$	$\overline{(p \rightarrow \overline{q})}$
$p \vee q$	$\overline{(\overline{p} \rightarrow q)}$
$p \leftrightarrow q$	$\overline{[(p \rightarrow q) \rightarrow \overline{(q \rightarrow p)}]}$

Tabela 3.8: Pares de proposições equivalentes, definindo os operadores AND, OR e XNOR, em função dos operadores NOT e IMPLICA.

3.9 Conjunto funcionalmente completo de operadores

- Deve-se notar, nas Tabelas 3.1 a 3.4, que metade das 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 3.6, 3.7 e 3.8.
- Portanto, as seguintes questões surgem naturalmente:
 - É possível descrever todos os demais operadores a partir de um determinado conjunto (conjunto completo) ?
 - Todos os operadores de um conjunto completo são absolutamente necessários (independentes)?
 - Existe um conjunto mínimo de operadores que forme um conjunto completo (conjunto completo mínimo) ?
- Respostas:
 - 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!
- Os operadores NOT, AND e OR são naturalmente utilizados nas expressões lógicas elaboradas pelo ser humano.
- Os operadores NOT, NAND e NOR são facilmente implementados por dispositivos eletro-eletrônicos.
- Assim sendo, é comum que se definam as expressões lógicas utilizando os operadores do conjunto {NOT, AND, OR} e, em seguida, que elas sejam convertidas em expressões equivalentes, empregando os operadores do conjunto {NOT, NAND, NOR}.

3.10 Teoremas de De Morgan

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

- $(A \text{ NAND } B) \equiv \text{NOT } (A \text{ AND } B) \equiv (\text{NOT } A) \text{ OR } (\text{NOT } B)$
ou
 $(A \uparrow B) \equiv \neg(A \wedge B) \equiv (\neg A) \vee (\neg B)$
- $(A \text{ NOR } B) \equiv \text{NOT } (A \text{ OR } B) \equiv (\text{NOT } A) \text{ AND } (\text{NOT } B)$
ou
 $(A \downarrow B) \equiv \neg(A \vee B) \equiv (\neg A) \wedge (\neg B)$

3.11 Decomposição em funções canônicas

- Sabe-se que uma função lógica genérica pode ser decomposta em uma combinação de funções NOT, OR e AND. Porém, demonstra-se que uma função lógica genérica também pode ser decomposta em 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.
- 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 mínimo de valores F, a função do tipo M , é denominada de maxtermo.
- Para funções de duas variáveis, os mintermos m_i são definidos por: $m_0(A, B) = (\neg A \wedge \neg B)$, $m_1(A, B) = (\neg A \wedge B)$, $m_2(A, B) = (A \wedge \neg B)$ e $m_3(A, B) = (A \wedge B)$.
- Para funções de duas variáveis, os maxtermos M_i são definidos por: $M_0(A, B) = (A \vee B)$, $M_1(A, B) = (A \vee \neg B)$, $M_2(A, B) = (\neg A \vee B)$ e $M_3(A, B) = (\neg A \vee \neg B)$.
- A Tabela 3.9 apresenta as funções canônicas (mintermos e maxtermos) para duas variáveis.

A	B	m_0	m_1	m_2	m_3	M_0	M_1	M_2	M_3
F	F	T	F	F	F	F	T	T	T
F	T	F	T	F	F	T	F	T	T
T	F	F	F	T	F	T	T	F	T
T	T	F	F	F	T	T	T	T	F

Tabela 3.9: 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 3.10 ilustra a decomposição da função $X(A, B) = (A \vee B)$ de duas formas, que são as seguintes:

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

A	B	$A \vee B$	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 3.10: Exemplo de decomposição em funções canônicas (mintermos e maxtermos).

- Uma vez 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 $X(\cdot)$, 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 notar ainda que $\neg(m_i) = M_i$.
- 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.
- Portanto, 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.

3.12 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 específica para os blocos funcionais a eles associados. Para a identidade lógica, o bloco é denominado de BUFFER. O bloco INVERSOR 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 3.1. 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, que apresentam um formato uniforme, foi estabelecido pelo *International Electrotechnical Commission* (Publicação IEC 117-15) e foi incluído no mesmo padrão.

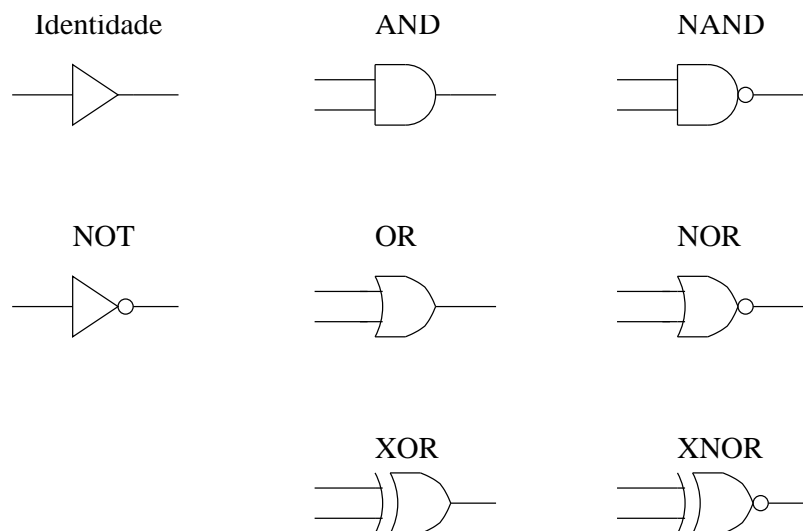


Figura 3.1: Blocos funcionais fundamentais, associados aos operadores lógicos básicos.

3.13 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 (3.1) e (3.2). Dois exemplos de manipulação algébrica de blocos, referentes às manipulações das Equações (3.1) e (3.2), são mostrados nas Figuras 3.2 e 3.3, respectivamente.

$$\begin{aligned}
 X(A, B) &= A \vee B \\
 &= (\neg A \wedge B) \vee (A \wedge \neg B) \\
 &= \neg(\neg((\neg A \wedge B) \vee (A \wedge \neg B))) \\
 &= \neg((\neg A \wedge B) \downarrow (A \wedge \neg B)) \\
 &= \neg((\neg(\neg A \wedge B)) \wedge (\neg(A \wedge \neg B))) \\
 &= (\neg A \uparrow B) \uparrow (A \uparrow \neg B) .
 \end{aligned}
 \tag{3.1}$$

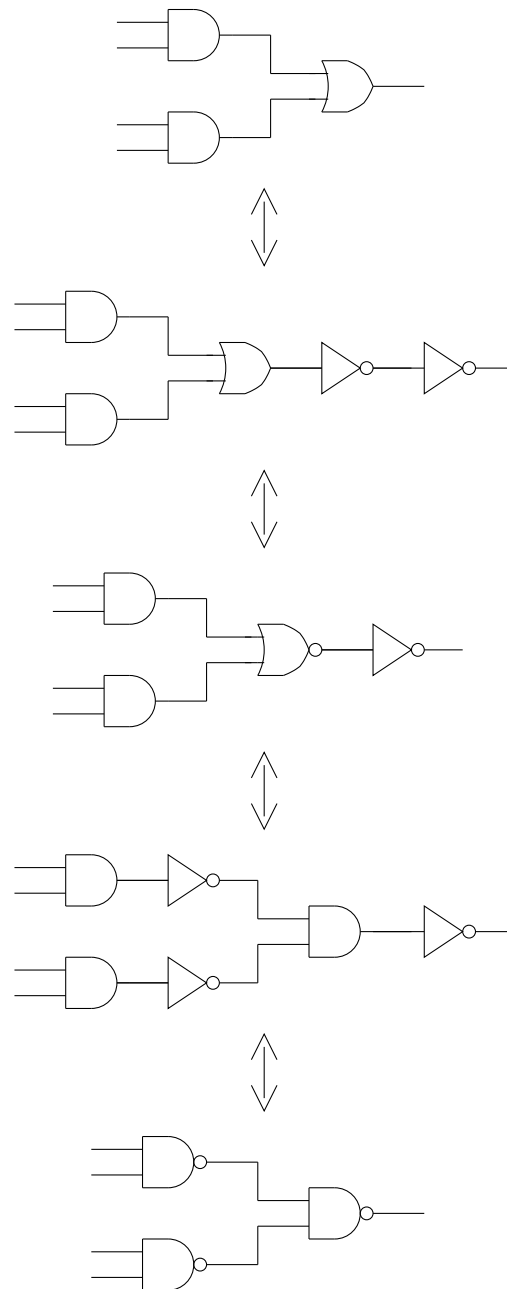


Figura 3.2: Manipulação algébrica de blocos: exemplo 1.

$$\begin{aligned}
 X(A, B) &= A \Downarrow B \\
 &= (A \vee B) \wedge (\neg A \vee \neg B) \\
 &= \neg(\neg((A \vee B) \wedge (\neg A \vee \neg B))) \\
 &= \neg((A \vee B) \uparrow (\neg A \vee \neg B)) \\
 &= \neg((\neg(A \vee B)) \vee (\neg(\neg A \vee \neg B))) \\
 &= (A \downarrow B) \downarrow (\neg A \downarrow \neg B) .
 \end{aligned}
 \tag{3.2}$$

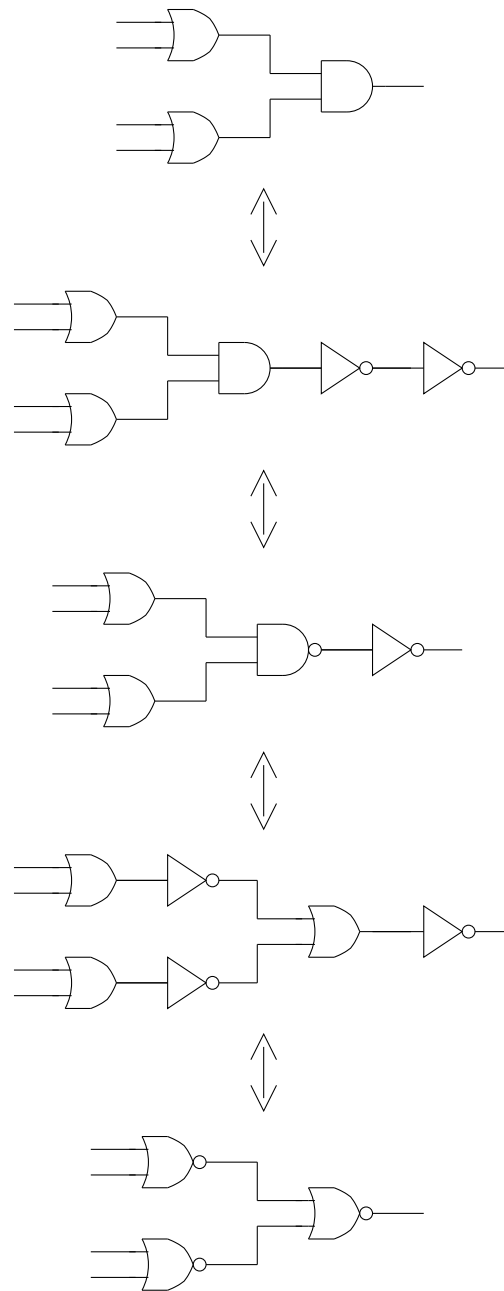


Figura 3.3: Manipulação algébrica de blocos: exemplo 2.

3.14 Conectivos de ordem superior

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

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

Possíveis implementações para o operador $AND(A,B,C,D)$, empregando-se apenas blocos fundamentais do tipo $AND(X,Y)$, são dadas por

$$AND(A, B, C, D) = AND(D, AND(C, AND(B, A))) \quad (3.3)$$

e por

$$AND(A, B, C, D) = AND(AND(D, C), AND(B, A)) . \quad (3.4)$$

3.15 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

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

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

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

3.16 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.

3.16.1 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 3.11.

Uma simbologia genérica para tais operações pode ser visualizada na Figura 3.4.

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

Tabela 3.11: Uso de operador lógico como elemento de controle.

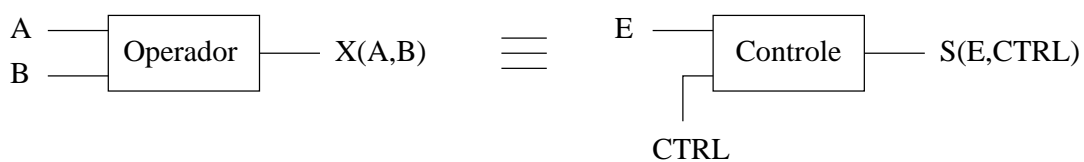


Figura 3.4: Uso de operador lógico como elemento de controle: simbologia genérica.

3.16.2 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 $\mathbf{V} = \{V_1, V_2, \dots, V_N\}$, onde $V_k \in \{F, T\}$. Nesse sentido, paridade ímpar e paridade par indicam que o conjunto de valores \mathbf{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 $\mathbf{V} = \{V_1, V_2, \dots, V_N\}$. Em outras, um novo valor V_{N+1} deve ser gerado e agregado a um dado conjunto \mathbf{V} , de tal forma que o novo conjunto aumentado $\mathbf{V}_P = \{\mathbf{V}, V_{N+1}\}$ 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 $\mathbf{V} = \{V_1, V_2\}$, respectivamente, um número ímpar de valores T e um número par de valores F.

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

3.16.3 Identificador de igualdade entre padrões binários

Dadas duas cadeias de valores binários, $\mathbf{V} = \{V_1, V_2, \dots, V_N\}$ e $\mathbf{W} = \{W_1, W_2, \dots, W_N\}$, 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 $\mathbf{V} = \{V_1\}$ e $\mathbf{W} = \{W_1\}$, os operadores XOR(A,B) e XNOR(A,B) podem ser usados como identificadores de diferença e de igualdade, respectivamente.

3.17 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 3.5.
5. Prove as equivalências das Tabelas 3.6, 3.7 e 3.8.
6. Prove os Teoremas de De Morgan.
7. Decomponha as funções de todos os conectivos binários como combinações das funções básicas m_i (minterms), associadas pelo conectivo OR.
8. Decomponha as funções de todos os conectivos binários como combinações das funções básicas M_i (maxterms), associadas pelo conectivo AND.

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

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

Equações lógicas:

- $F(A, B) = (A \vee B) \wedge (A \vee \neg B) \wedge (\neg A \vee B)$.
- $F(A, B) = (\neg A \vee B) \wedge (A \vee \neg B) \wedge (\neg A \vee \neg B)$.
- $F(A, B) = (A \wedge \neg B) \vee (\neg A \wedge B) \vee (A \wedge B)$.
- $F(A, B) = (\neg A \wedge \neg B) \vee (\neg A \wedge B) \vee (A \wedge \neg B)$.
- $F(A, B, C, D) = (A \vee B) \wedge (\neg(C \wedge D))$.

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

- Calcule o resultado da seguinte seqüência de operações:

$$(V_N Op (\dots (V_3 Op (V_2 Op V_1)) \dots)) .$$

(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.)

- Considere que $Op(V_1, V_2, \dots, V_N)$ é um operador de ordem N e verifique a validade da seguinte relação:

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

12. Verifique a validade das seguintes relações:

- $A XOR B = (\neg A AND B) OR (A AND \neg B) = (A OR B) AND (\neg A OR \neg B)$.
- $A XNOR B = (A AND B) OR (\neg A AND \neg B) = (\neg A OR B) AND (A OR \neg B)$.
- $A XOR B = \neg(A XNOR B) = \neg A XNOR B = A XNOR \neg B$.
- $A XNOR B = \neg(A XOR B) = \neg A XOR B = A XOR \neg B$.

Capítulo 4

Álgebra de Boole

4.1 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 Boole, a qual será tratada neste capítulo.

4.2 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 \mathbf{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 $b \in \mathbf{K}$, $(a + b) \in \mathbf{K}$. É definida uma operação, denotada por “.”, tal que, dados a e $b \in \mathbf{K}$, $(a \cdot b) \in \mathbf{K}$.
 3. Existe um elemento $0 \in \mathbf{K}$, tal que, para cada $a \in \mathbf{K}$, $(a + 0) = a$. Existe um elemento $1 \in \mathbf{K}$, tal que, para cada $a \in \mathbf{K}$, $(a \cdot 1) = a$.
 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 $a \in \mathbf{K}$ existe um elemento $\bar{a} \in \mathbf{K}$, tal que:

$$a + \bar{a} = 1$$

$$a \cdot \bar{a} = 0$$
 7. Deve haver, pelo menos, um total de dois elementos a e $b \in \mathbf{K}$, tal que $a \neq b$.

4.3 Versão alternativa para os Postulados de Huntington

- A Álgebra Booleana é um conjunto $B = \{a, b, c, \dots\}$ 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 $b \in \mathbf{B}$, 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 $a \in \mathbf{B}$ existe um elemento $\bar{a} \in \mathbf{B}$, tal que:

$$a \cup \bar{a} = u$$

$$a \cap \bar{a} = z$$
- 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.

4.4 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 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.

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

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

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c$$
 2. Para cada par a e $b \in \mathbf{K}$,

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

$$a \cdot (\bar{a} + b) = (a \cdot b)$$
 3. Para cada par a e $b \in \mathbf{K}$,

$$\overline{a + b} = \bar{a} \cdot \bar{b}$$

$$\overline{a \cdot b} = \bar{a} + \bar{b}$$
 4. Para quaisquer três elementos a, b e $c \in \mathbf{K}$, $(a \cdot b) + (\bar{a} \cdot c) + (b \cdot c) = (a \cdot b) + (\bar{a} \cdot c)$

4.6 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: $K = \{0, 1\}$.
- Complementos: $\begin{cases} 0 = \bar{1} \\ 1 = \bar{0} \end{cases}$.
- Identidades: $\begin{cases} 1 \cdot 1 = 1 + 1 = 1 + 0 = 0 + 1 = 1 \\ 0 + 0 = 0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0 \end{cases}$.

4.7 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 4.1.
- Teoria de conjuntos: a associação entre a teoria de conjuntos e a estrutura algébrica de Boole é apresentada na Tabela 4.2.
- Das Tabelas 4.1 e 4.2, 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
$\neg(S)$	\bar{S}

Tabela 4.1: Tabela de mapeamento: Cálculo Proposicional \times Álgebra de Boole.

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

Tabela 4.2: Tabela de mapeamento: Teoria de Conjuntos \times Álgebra de Boole.

4.8 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 isomórfico podem ser usadas para resolver problemas nos sistemas equivalentes, através do mapeamento adequado.

4.9 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 reduzi-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.

4.10 Exemplo de manipulação algébrica não sistemática: Postulados, Lemas e Teoremas

Inicialmente, usando a Tabela 4.1, 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 (4.1) a (4.3) 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.

$$\begin{aligned}
& (A + B) \cdot (A + \overline{B}) \cdot (\overline{A} + B) \\
& \quad \downarrow P5 \\
& ((A + B) \cdot (A)) + [(A + B) \cdot (\overline{B})] \cdot (\overline{A} + B) \\
& \quad \downarrow P5 \\
& ((A \cdot A) + (B \cdot A)) + [(A \cdot \overline{B}) + (B \cdot \overline{B})] \cdot (\overline{A} + B) \\
& \quad \downarrow L2/P6 \\
& ([A + (B \cdot A)] + [(A \cdot \overline{B}) + 0]) \cdot (\overline{A} + B) \\
& \quad \downarrow P4/P3 \\
& ([A + (A \cdot B)] + [(A \cdot \overline{B})]) \cdot (\overline{A} + B) \\
& \quad \downarrow L5 \\
& [A + (A \cdot \overline{B})] \cdot (\overline{A} + B) \\
& \quad \downarrow P5 \\
& [A \cdot (\overline{A} + B)] + [(A \cdot \overline{B}) \cdot (\overline{A} + B)] \\
& \quad \downarrow P5 \\
& (A \cdot \overline{A}) + (A \cdot B) + (A \cdot \overline{B} \cdot \overline{A}) + (A \cdot \overline{B} \cdot B) \\
& \quad \downarrow P6/P4 \\
& 0 + (A \cdot B) + (A \cdot \overline{A} \cdot \overline{B}) + (A \cdot B \cdot \overline{B}) \\
& \quad \downarrow P6 \\
& 0 + (A \cdot B) + (0 \cdot \overline{B}) + (A \cdot 0) \\
& \quad \downarrow P4/L3 \\
& 0 + (A \cdot B) + (\overline{B} \cdot 0) + 0 \\
& \quad \downarrow L3 \\
& 0 + (A \cdot B) + 0 + 0 \\
& \quad \downarrow P4 \\
& (A \cdot B) + 0 + 0 + 0 \\
& \quad \downarrow P3/P3/P3 \\
& (A \cdot B) \tag{4.1}
\end{aligned}$$

$$\begin{aligned}
& (A + B) \cdot (A + \overline{B}) \cdot (\overline{A} + B) \\
& \qquad \qquad \qquad \downarrow P5 \\
& [A + (B \cdot \overline{B})] \cdot (\overline{A} + B) \\
& \qquad \qquad \qquad \downarrow P6 \\
& (A + 0) \cdot (\overline{A} + B) \\
& \qquad \qquad \qquad \downarrow P3 \\
& A \cdot (\overline{A} + B) \\
& \qquad \qquad \qquad \downarrow P5 \\
& (A \cdot \overline{A}) + (A \cdot B) \\
& \qquad \qquad \qquad \downarrow P6 \\
& 0 + (A \cdot B) \\
& \qquad \qquad \qquad \downarrow P4 \\
& (A \cdot B) + 0 \\
& \qquad \qquad \qquad \downarrow P3 \\
& (A \cdot B)
\end{aligned} \tag{4.2}$$

$$\begin{aligned}
& (A + B) \cdot (A + \overline{B}) \cdot (\overline{A} + B) \\
& \qquad \qquad \qquad \downarrow L2 \\
& (A + B) \cdot (A + B) \cdot (A + \overline{B}) \cdot (\overline{A} + B) \\
& \qquad \qquad \qquad \downarrow P4 \\
& (A + B) \cdot (A + \overline{B}) \cdot (A + B) \cdot (\overline{A} + B) \\
& \qquad \qquad \qquad \downarrow P5 \\
& [A + (B \cdot \overline{B})] \cdot [(A \cdot \overline{A}) + B] \\
& \qquad \qquad \qquad \downarrow P6 \\
& [A + 0] \cdot [0 + B] \\
& \qquad \qquad \qquad \downarrow P3/P3 \\
& (A \cdot B)
\end{aligned} \tag{4.3}$$

4.11 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 4.1 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.

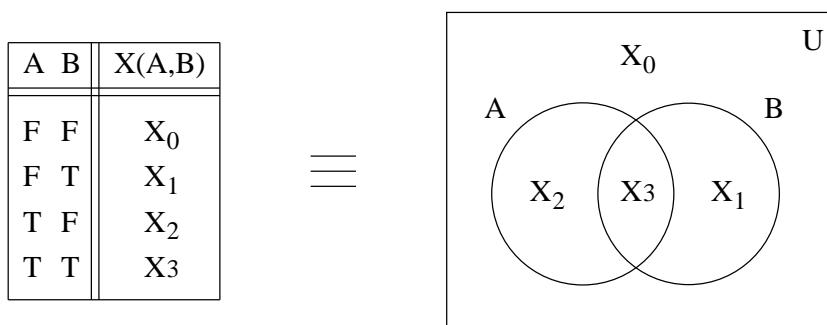


Figura 4.1: Mapeamento entre uma função genérica de duas variáveis e um Diagrama de Venn.

4.12 Resumo das relações algébricas

As Tabelas 4.3 a 4.7 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	$\begin{cases} a + 0 = a \\ a \cdot 1 = a \end{cases}$
P4	$\begin{cases} (a + b) = (b + a) \\ (a \cdot b) = (b \cdot a) \end{cases}$
P5	$\begin{cases} a + (b \cdot c) = (a + b) \cdot (a + c) \\ a \cdot (b + c) = (a \cdot b) + (a \cdot c) \end{cases}$
P6	$\begin{cases} a + \bar{a} = 1 \\ a \cdot \bar{a} = 0 \end{cases}$

Tabela 4.3: Resumo dos postulados de Huntington para a estrutura algébrica de Boole.

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

Tabela 4.4: 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	$\begin{cases} a + (\bar{a} \cdot b) = (a + b) \\ a \cdot (\bar{a} + b) = (a \cdot b) \end{cases}$
T3	$\begin{cases} \overline{(a + b)} = \bar{a} \cdot \bar{b} \\ \overline{(a \cdot b)} = \bar{a} + \bar{b} \end{cases}$
T4	$\begin{cases} (a \cdot b) + (\bar{a} \cdot c) + (b \cdot c) = (a \cdot b) + (\bar{a} \cdot c) \\ (a + b) \cdot (\bar{a} + c) \cdot (b + c) = (a + b) \cdot (\bar{a} + c) \end{cases}$

Tabela 4.5: Resumo dos teoremas para a estrutura algébrica de Boole.

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

Tabela 4.6: 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
$C(S)$	$\neg(S)$	\bar{S}

Tabela 4.7: Resumo das relações de isomorfismo.

4.13 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) $A \oplus 1 = \bar{A}$

(c) $A \oplus A = 0$

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

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

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

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

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

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

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

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

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

(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 $C = \bar{A}$, $D = B$, $E = A$ e $F = \bar{B}$.

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

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

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

Capítulo 5

Formas padrões para representação de expressões booleanas

5.1 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 min-terms e max-terms.
- Uma expansão em min-terms envolve realizar os min-terms necessários com operadores AND e, em seguida, combiná-los com operadores OR. Por essa razão, uma decomposição em min-terms é chamada de forma padrão AND-OR.
- Uma expansão em max-terms envolve realizar os max-terms necessários com operadores OR e, em seguida, combiná-los com operadores AND. Por essa razão, uma decomposição em max-terms 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 \rightarrow {AND-OR, NAND-NAND, OR-NAND, NOR-OR}.
 - Grupo OR-AND \rightarrow {AND-NOR, NAND-AND, OR-AND, NOR-NOR}.

5.2 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.
- Observações:
 - Uma vez que $(A \cdot A) = (A + A) = A$, $(A + \bar{A}) = 1$ e $(A \cdot \bar{A}) = 0$, 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, como será abordado a seguir, a forma normal expandida é o ponto de partida para o processo de simplificação adotado.
- 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.

- Exemplos da definição de mintermos e maxtermos, para três variáveis, são apresentados na Tabela 5.1 e na Tabela 5.2, respectivamente.

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

Tabela 5.1: Definição de mintermos para três variáveis (A,B,C).

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

Tabela 5.2: Definição de maxtermos para três variáveis (A,B,C).

5.3 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.

5.3.1 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 dos 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.

5.3.2 Manipulação algébrica

- Para se obter uma forma normal:
 - 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 se obter a forma normal expandida:
 - 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 (5.1), para $F = f(A, B, C, D)$.

$$\begin{aligned}
 F &= (A + B) \cdot \overline{(C \cdot D)} \\
 &= (A + B) \cdot (\overline{C} + \overline{D}) \\
 &= (A + B + 0) \cdot (0 + \overline{C} + \overline{D}) \\
 &= [A + B + (C \cdot \overline{C})] \cdot [(B \cdot \overline{B}) + \overline{C} + \overline{D}] \\
 &= (A + B + C) \cdot (A + B + \overline{C}) \cdot (B + \overline{C} + \overline{D}) \cdot (\overline{B} + \overline{C} + \overline{D}) \\
 &= (A + B + C + 0) \cdot (A + B + \overline{C} + 0) \cdot (0 + B + \overline{C} + \overline{D}) \cdot (0 + \overline{B} + \overline{C} + \overline{D}) \\
 &= [A + B + C + (D \cdot \overline{D})] \cdot [A + B + \overline{C} + (D \cdot \overline{D})] \cdot \\
 &\quad [(A \cdot \overline{A}) + B + \overline{C} + \overline{D}] \cdot [(A \cdot \overline{A}) + \overline{B} + \overline{C} + \overline{D}] \\
 &= (A + B + C + D) \cdot (A + B + C + \overline{D}) \cdot (A + B + \overline{C} + D) \cdot \\
 &\quad (A + B + \overline{C} + \overline{D}) \cdot (A + B + \overline{C} + \overline{D}) \cdot (\overline{A} + B + \overline{C} + \overline{D}) \cdot \\
 &\quad (A + \overline{B} + \overline{C} + \overline{D})(\overline{A} + \overline{B} + \overline{C} + \overline{D}) \\
 &= (A + B + C + D) \cdot (A + B + C + \overline{D}) \cdot (A + B + \overline{C} + D) \cdot \\
 &\quad (A + B + \overline{C} + \overline{D}) \cdot (\overline{A} + B + \overline{C} + \overline{D}) \cdot \\
 &\quad (A + \overline{B} + \overline{C} + \overline{D})(\overline{A} + \overline{B} + \overline{C} + \overline{D}) \\
 &= \prod M(0, 1, 2, 3, 11, 7, 15) \tag{5.1}
 \end{aligned}$$

- Um exemplo do procedimento para a obtenção de uma forma SOP padrão é apresentado na Equação (5.2), 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{D}) + (\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) \tag{5.2}
\end{aligned}$$

5.3.3 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 5.3, para uma função $X = f(A, B, C)$.

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

Tabela 5.3: Exemplo de função e associação de mintermos.

- Da Tabela 5.3, pode-se escrever que

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

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, $y = f(x_1, x_2, \dots, x_N)$, 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 5.4, para uma função $X = f(A, B, C)$.

A	B	C	X	Z_1	Z_2	Z_3	Z_4	Z_5	Linha	Lógica	Maxtermo
0	0	0	0	0	1	1	1	1	0	$(A + B + C)$	M_0 (ou M_7)
0	0	1	1	1	1	1	1	1	1	$(A + B + \overline{C})$	M_1 (ou M_6)
0	1	0	0	1	0	1	1	1	2	$(A + \overline{B} + C)$	M_2 (ou M_5)
0	1	1	0	1	1	0	1	1	3	$(A + \overline{B} + \overline{C})$	M_3 (ou M_4)
1	0	0	1	1	1	1	1	1	4	$(\overline{A} + B + C)$	M_4 (ou M_3)
1	0	1	0	1	1	1	0	1	5	$(\overline{A} + B + \overline{C})$	M_5 (ou M_2)
1	1	0	1	1	1	1	1	1	6	$(\overline{A} + \overline{B} + C)$	M_6 (ou M_1)
1	1	1	0	1	1	1	1	0	7	$(\overline{A} + \overline{B} + \overline{C})$	M_7 (ou M_0)

Tabela 5.4: Exemplo de função e definição de maxtermos.

- Da Tabela 5.4, pode-se escrever que

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

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, $y = f(x_1, x_2, \dots, x_N)$, pode ser expressa por uma forma POS padrão.

5.3.4 Negação das formas SOP e POS

Dadas uma função genérica $F(A, B, C, \dots) = F(\cdot)$ e a sua negação $\overline{F}(A, B, C, \dots) = \overline{F}(\cdot)$, 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

$$\overline{(X \cdot Y \cdot Z \cdot \dots)} = (\overline{X} + \overline{Y} + \overline{Z} + \dots)$$

e

$$\overline{(X + Y + Z + \dots)} = (\overline{X} \cdot \overline{Y} \cdot \overline{Z} \cdot \dots) .$$

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

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

e

$$F_{POS}(\cdot) = (M_l \cdot M_m \cdot M_n \cdot \dots) ,$$

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

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

e

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

obtêm-se as seguintes relações:

$$\begin{aligned} \overline{F}_{POS}(\cdot) &= (M_i \cdot M_j \cdot M_k) \\ &= (\overline{m_i} \cdot \overline{m_j} \cdot \overline{m_k}) \\ &= \overline{(m_i + m_j + m_k)} \\ &= \overline{[F_{SOP}(\cdot)]} \end{aligned}$$

e

$$\begin{aligned} \overline{F}_{SOP}(\cdot) &= (m_l + m_m + m_n + \dots) \\ &= (\overline{M_l} + \overline{M_m} + \overline{M_n} + \dots) \\ &= \overline{(M_l \cdot M_m \cdot M_n \cdot \dots)} \\ &= \overline{[F_{POS}(\cdot)]} . \end{aligned}$$

5.4 Conjuntos de formas padrões

5.4.1 Definição

- 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: {AND-OR, NAND-NAND, OR-NAND, NOR-OR}.
- A partir da forma OR-AND, pode-se obter o seguinte grupo de formas padrões: {OR-AND, NOR-NOR, AND-NOR, NAND-AND}.

5.4.2 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 5.5 a 5.7, exemplificam, para a função XOR, a obtenção do grupo AND-OR, a mudança de grupo e a obtenção do grupo OR-AND, respectivamente.
- Partindo-se da tabela verdade, pode-se obter diretamente algumas formas:
 - A soma de mintermos da função fornece a forma AND-OR.
 - O produto de maxtermos da função, fornece a forma OR-AND.
 - A soma de mintermos da função complementar, fornece a forma AND-NOR (grupo OR-AND).
 - O produto de maxtermos da função complementar, fornece 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.

5.4.3 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, elas 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
$F(A, B) = A \oplus B$	
$= (\overline{A} \cdot B) + (A \cdot \overline{B})$	AND-OR
$= \overline{(\overline{A} \cdot B) + (A \cdot \overline{B})}$	
$= \overline{(\overline{A} \cdot B)} \cdot \overline{(A \cdot \overline{B})} = (\overline{A} \uparrow B) \uparrow (A \uparrow \overline{B})$	NAND-NAND
$= \overline{(A + \overline{B})} \cdot \overline{(\overline{A} + B)} = (A + \overline{B}) \uparrow (\overline{A} + B)$	OR-NAND
$= \overline{(A + \overline{B})} + \overline{(\overline{A} + B)} = (A \downarrow \overline{B}) + (\overline{A} \downarrow B)$	NOR-OR
$= (\overline{A} \cdot B) + (A \cdot \overline{B})$	AND-OR

Tabela 5.5: Exemplo da obtenção do grupo AND-OR para a função XOR.

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

Tabela 5.6: Exemplo da mudança de grupo para a função XOR.

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

Tabela 5.7: Exemplo da obtenção do grupo OR-AND para a função XOR.

5.5 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 $F_{SOP}(\cdot)$.
 - Equacionando-se cada valor 0 de $F(\cdot)$ isoladamente, obtém-se uma POS padrão $F_{POS}(\cdot)$.
 - Dado que $0 = \bar{1}$ e $1 = \bar{0}$:
 - Pode-se obter $F_{POS}(\cdot)$ pela SOP da negação de $F(\cdot)$: $F_{POS}(\cdot) = \overline{F_{SOP}(\cdot)}$.
 - Pode-se obter $F_{SOP}(\cdot)$ pela POS da negação de $F(\cdot)$: $F_{SOP}(\cdot) = \overline{F_{POS}(\cdot)}$.

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 booleanas:

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

$$\text{ii. } F(A, B, C) = \overline{\left\{ \left[(\bar{A} + B) + C \right] \cdot \left[\bar{A} + \overline{(B + \bar{C})} \right] \right\}}.$$

Capítulo 6

Simplificação algébrica sistemática de expressões booleanas

6.1 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: i) menor quantidade de circuito necessário à implementação, ii) circuito com o menor tempo de resposta e iii) 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 também são conhecidas por: estruturas em dois níveis e estruturas de segunda ordem.

Uma expressão com dois planos de operações lógicas será considerada mínima se: i) não existir outra expressão desse tipo com um número menor de termos e ii) 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

$$F(A, B, C, D) = (A + B) \cdot [(C \cdot \bar{D}) + (\bar{C} \cdot D)] \quad (6.1)$$

e

$$F(A, B, C, D) = (A \cdot \bar{C} \cdot D) + (A \cdot C \cdot \bar{D}) + (B \cdot \bar{C} \cdot D) + (B \cdot C \cdot \bar{D}) . \quad (6.2)$$

Na Equação (6.1), 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 (6.2), a expressão booleana encontra-se na forma SOP. Embora a expressão da Equação (6.1) 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 (6.2) 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 uma determinada função lógica pode ter diversas expressões mínimas (locais) equivalentes.

Além disso, não se pode garantir que, para uma dada função, a expressão mínima 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.

6.2 Processo de expansão \times 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.

6.2.1 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:

$$f(A, B) = (B) = (B) \cdot (1) = (B) \cdot (A + \bar{A}) = (B \cdot A) + (B \cdot \bar{A}) = (A \cdot B) + (A \cdot \bar{B}), \quad (6.3)$$

$$f(A, B) = (B) = (B) + (0) = (B) + (A \cdot \bar{A}) = (B + A) \cdot (B + \bar{A}) = (A + B) \cdot (A + \bar{B}), \quad (6.4)$$

$$f(A, B, C) = (A \cdot B) = (A \cdot B) \cdot (1) = (A \cdot B) \cdot (C + \bar{C}) = (A \cdot B \cdot C) + (A \cdot B \cdot \bar{C}), \quad (6.5)$$

$$f(A, B, C) = (A + B) = (A + B) + (0) = (A + B) + (C \cdot \bar{C}) = (A + B + C) \cdot (A + B + \bar{B}), \quad (6.6)$$

$$\begin{aligned} f(A, B, C) &= (B) \\ &= (B) \cdot (1) \\ &= (B) \cdot (A + \bar{A}) \\ &= (B \cdot A) + (B \cdot \bar{A}) \\ &= (A \cdot B) + (A \cdot \bar{B}) \\ &= [(A \cdot B) \cdot (1)] + [(A \cdot \bar{B}) \cdot (1)] \\ &= [(A \cdot B) \cdot (C + \bar{C})] + [(A \cdot \bar{B}) \cdot (C + \bar{C})] \\ &= (A \cdot B \cdot C) + (A \cdot B \cdot \bar{C}) + (A \cdot \bar{B} \cdot C) + (A \cdot \bar{B} \cdot \bar{C}) \end{aligned} \quad (6.7)$$

e

$$\begin{aligned} f(A, B, C) &= (B) \\ &= (B) + (0) \\ &= (B) + (A \cdot \bar{A}) \\ &= (B + A) \cdot (B + \bar{A}) \\ &= (A + B) \cdot (A + \bar{B}) \\ &= [(A + B) + (0)] \cdot [(A + \bar{B}) + (0)] \\ &= [(A + B) + (C \cdot \bar{C})] \cdot [(A + \bar{B}) + (C \cdot \bar{C})] \\ &= (A + B + C) \cdot (A + B + \bar{C}) \cdot (A + \bar{B} + C) \cdot (A + \bar{B} + \bar{C}). \end{aligned} \quad (6.8)$$

6.2.2 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:

$$\begin{aligned}
 f(A, B, C) &= (A \cdot B) + (A \cdot C) \\
 &= [(A \cdot B) \cdot (1)] + [(A \cdot C) \cdot (1)] \\
 &= [(A \cdot B) \cdot (C + \overline{C})] + [(A \cdot C) \cdot (B + \overline{B})] \\
 &= [(A \cdot B \cdot C) + (A \cdot B \cdot \overline{C})] + [(A \cdot C \cdot B) + (A \cdot C \cdot \overline{B})] \\
 &= [(A \cdot B \cdot C) + (A \cdot B \cdot \overline{C})] + [(A \cdot B \cdot C) + (A \cdot \overline{B} \cdot C)] \\
 &= (A \cdot B \cdot C) + (A \cdot B \cdot \overline{C}) + (A \cdot \overline{B} \cdot C)
 \end{aligned} \tag{6.9}$$

e

$$\begin{aligned}
 f(A, B, C) &= (A + B) \cdot (A + C) \\
 &= [(A + B) + (0)] \cdot [(A + C) + (0)] \\
 &= [(A + B) + (C \cdot \overline{C})] \cdot [(A + C) + (B \cdot \overline{B})] \\
 &= [(A + B + C) \cdot (A + B + \overline{C})] \cdot [(A + C + B) \cdot (A + C + \overline{B})] \\
 &= [(A + B + C) \cdot (A + B + \overline{C})] \cdot [(A + B + C) \cdot (A + \overline{B} + C)] \\
 &= (A + B + C) \cdot (A + B + \overline{C}) \cdot (A + \overline{B} + C) .
 \end{aligned} \tag{6.10}$$

6.2.3 Análise das expansões

As Equações (6.3) a (6.10) 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 gera 2 termos (somas ou produtos) padrões de 2 literais.
- A expansão de 1 literal gera 4 termos (somas ou produtos) padrões de 3 literais.
- Pode-se mostrar que a expansão de 1 literal gera 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 por: P3 \rightarrow P6 \rightarrow P5.
- A expansão com redundâncias de um termo qualquer segue o procedimento geral dado por: P3 \rightarrow P6 \rightarrow P5 \rightarrow L2.
- Eventualmente, durante a expansão, pode ser necessário aplicar o Postulado P4.

6.3 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.

6.4 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.

6.4.1 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 (6.11) e (6.12), respectivamente. Da Álgebra de Boole, a aglutinação utiliza os Postulados P5, P6 e P3, enquanto a replicação emprega o Lema L2.

$$\text{Aglutinação} \begin{cases} (A \cdot B) + (A \cdot \overline{B}) = A \cdot (B + \overline{B}) = A \cdot 1 = A \\ (A + B) \cdot (A + \overline{B}) = A + (B \cdot \overline{B}) = A + 0 = A \end{cases} \quad (6.11)$$

$$\text{Replicação} \begin{cases} A = A + A + A + \dots \\ A = A \cdot A \cdot A \cdot \dots \end{cases} \quad (6.12)$$

6.4.2 Uso da aglutinação

A propriedade de distributividade mostra que, se dois termos diferem de apenas um literal (A e \bar{A}), 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 (6.11), (6.13) e (6.14).

$$\begin{aligned}
 F(A, B, C) &= \sum m(4, 6) \\
 &= m_4 + m_6 \\
 &= (A \cdot \bar{B} \cdot \bar{C}) + (A \cdot B \cdot \bar{C}) \\
 &= (A \cdot \bar{C} \cdot \bar{B}) + (A \cdot \bar{C} \cdot B) \\
 &= [(A \cdot \bar{C}) \cdot \bar{B}] + [(A \cdot \bar{C}) \cdot B] \\
 &= (A \cdot \bar{C}) \cdot (\bar{B} + B) \\
 &= (\bar{A} \cdot \bar{C}) \cdot (1) \\
 &= (A \cdot \bar{C})
 \end{aligned} \tag{6.13}$$

$$\begin{aligned}
 F(A, B, C) &= \prod (0, 4) \\
 &= M_0 \cdot M_4 \\
 &= (A + B + C) \cdot (\bar{A} + B + C) \\
 &= [A + (B + C)] \cdot [\bar{A} + (B + C)] \\
 &= (A \cdot \bar{A}) + (B + C) \\
 &= (0) + (B + C) \\
 &= (B + C)
 \end{aligned} \tag{6.14}$$

6.4.3 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 (6.15).

$$\begin{aligned}
 F(A, B, C) &= \sum m(0, 1, 2) \\
 &= m_0 + m_1 + m_2 \\
 &= (\bar{A} \cdot \bar{B} \cdot \bar{C}) + (\bar{A} \cdot \bar{B} \cdot C) + (\bar{A} \cdot B \cdot \bar{C}) \\
 &= m_0 + m_1 + m_2 + m_0 \\
 &= (\bar{A} \cdot \bar{B} \cdot \bar{C}) + (\bar{A} \cdot \bar{B} \cdot C) + (\bar{A} \cdot B \cdot \bar{C}) + (\bar{A} \cdot \bar{B} \cdot \bar{C}) \\
 &= [(\bar{A} \cdot \bar{B} \cdot \bar{C}) + (\bar{A} \cdot \bar{B} \cdot C)] + [(\bar{A} \cdot B \cdot \bar{C}) + (\bar{A} \cdot \bar{B} \cdot \bar{C})] \\
 &= [(\bar{A} \cdot \bar{B}) \cdot (\bar{C} + C)] + [(\bar{A} \cdot \bar{C}) \cdot (\bar{B} + B)] \\
 &= (\bar{A} \cdot \bar{B}) + (\bar{A} \cdot \bar{C}) \\
 &= A \cdot (\bar{B} + \bar{C})
 \end{aligned} \tag{6.15}$$

6.5 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 6.1 apresenta uma tabela verdade genérica para funções de 3 variáveis. Para tais funções, as Figuras 6.1 e 6.2 ilustram as possibilidades de eliminação de 1 e 2 literais em combinações de 2 e 4 mintermos, respectivamente. Por sua vez, as Figuras 6.3 e 6.4 ilustram as possibilidades de eliminação de 1 e 2 literais em combinações de 2 e 4 maxtermos, respectivamente.

<i>Linha</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>F(A, B, C)</i>
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 6.1: Tabela verdade genérica para funções de 3 variáveis.

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

Figura 6.1: 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 6.2: 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 6.3: 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 6.4: Eliminações de 2 literais em combinações 4 de maxtermos.

6.6 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.

6.6.1 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 (6.16) 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.

$$\begin{aligned}
 F(A, B, C) &= \sum m(0, 1, 7) \\
 &= (\bar{A} \cdot \bar{B} \cdot \bar{C}) + (\bar{A} \cdot \bar{B} \cdot C) + (A \cdot B \cdot C) \\
 &= (\bar{A} \cdot B) + (A \cdot B \cdot C)
 \end{aligned} \tag{6.16}$$

6.6.2 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 (6.17) 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.

$$\begin{aligned}
 F(A, B, C) &= \prod M(2, 3, 5) \\
 &= (A + \bar{B} + C) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + \bar{C}) \\
 &= (A + \bar{B}) \cdot (\bar{A} + B + \bar{C})
 \end{aligned} \tag{6.17}$$

6.6.3 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.

6.7 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 expressões. 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.

6.8 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.

6.9 Minimização com estruturas de ordem superior: fatoração

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.

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 $(L \cdot \bar{L}) = 0$ e $(L + \bar{L}) = 1$, que acabam por suprimir o plano extra criado pela fatoração. Por exemplo, em uma fatoração com supressão de termo, tem-se que

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

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

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

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 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. Por outro lado, quando o tempo de resposta não é decisivo, pode-se tentar encontrar uma fatoração favorável que leve à estrutura absolutamente mínimo. Nesse caso, a dificuldade é encontrar um modo simples e prático de se realizar tais fatorações em uma expressão booleana qualquer.

6.10 Exercícios propostos

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

(a) $F(A, B, C) = (\bar{A} \cdot \bar{B} \cdot \bar{C}) + (\bar{A} \cdot B \cdot \bar{C}) + (A \cdot \bar{B} \cdot C) + (A \cdot B \cdot C).$

(b) $F(A, B, C) = (\bar{A} \cdot \bar{B} \cdot C) + (\bar{A} \cdot B \cdot C) + (A \cdot \bar{B} \cdot \bar{C}) + (A \cdot B \cdot \bar{C}).$

(c) $F(A, B, C) = (\bar{A} \cdot \bar{B} \cdot \bar{C}) + (\bar{A} \cdot \bar{B} \cdot C) + (A \cdot B \cdot \bar{C}) + (A \cdot B \cdot C).$

(d) $F(A, B, C) = (\bar{A} \cdot B \cdot \bar{C}) + (\bar{A} \cdot B \cdot C) + (A \cdot \bar{B} \cdot \bar{C}) + (A \cdot \bar{B} \cdot C).$

(e) $F(A, B, C) = (\bar{A} \cdot \bar{B} \cdot \bar{C}) + (\bar{A} \cdot B \cdot C) + (A \cdot \bar{B} \cdot \bar{C}) + (A \cdot B \cdot C).$

(f) $F(A, B, C) = (\bar{A} \cdot \bar{B} \cdot C) + (\bar{A} \cdot B \cdot \bar{C}) + (A \cdot \bar{B} \cdot C) + (A \cdot B \cdot \bar{C}).$

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

(a) $F(A, B, C) = (A + B + \bar{C}) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + C) \cdot (\bar{A} + \bar{B} + C).$

(b) $F(A, B, C) = (A + B + C) \cdot (A + \bar{B} + C) \cdot (\bar{A} + B + \bar{C}) \cdot (\bar{A} + \bar{B} + \bar{C}).$

(c) $F(A, B, C) = (A + \bar{B} + C) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + C) \cdot (\bar{A} + B + \bar{C}).$

(d) $F(A, B, C) = (A + B + C) \cdot (A + B + \bar{C}) \cdot (\bar{A} + \bar{B} + C) \cdot (\bar{A} + \bar{B} + \bar{C}).$

(e) $F(A, B, C) = (A + B + \bar{C}) \cdot (A + \bar{B} + C) \cdot (\bar{A} + B + \bar{C}) \cdot (\bar{A} + \bar{B} + C).$

(f) $F(A, B, C) = (A + B + C) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + C) \cdot (\bar{A} + \bar{B} + \bar{C}).$

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 booleanas:

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

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

Capítulo 7

Mapa de Karnaugh

7.1 Introdução

- O mapa de Karnaugh (mapa-K) é mais uma das possíveis expressões de uma função lógica, além de 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 para a minimização da equação que a define.
- Ele pode ser interpretado como uma tabela verdade rearranjada ou como uma representação análoga ao Diagrama de Venn.
- Para cada linha da tabela verdade de uma função lógica booleana é associada uma posição no mapa.
- Uma vez que cada linha da tabela verdade é associada a um mintermo ou a um maxtermo, a cada um deles também é associada uma posição do mapa.
- A fim de que o mapa seja empregado no processo de simplificação de funções lógicas booleanas, ele deve ser arranjado da seguinte forma:
 - Deve existir uma localização única no mapa para cada combinação das variáveis das quais a função lógica é dependente.
 - As localizações devem ser arranjadas de tal forma que grupos de mintermos/maxtermos possam ser facilmente combinados em formas reduzidas.
- 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.

7.2 Construção do mapa-K

7.2.1 Funções de 1 variável

<i>Linha</i>	<i>A</i>	<i>F(A)</i>
0	0	F_0
1	1	F_1

Tabela 7.1: Tabela verdade para funções de 1 variável.

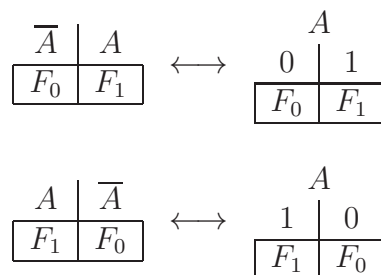


Figura 7.1: Exemplos de mapas de Karnaugh para funções de 1 variável.

7.2.2 Funções de 2 variáveis

<i>Linha</i>	<i>A</i>	<i>B</i>	$F(A, B)$
0	0	0	F_0
1	0	1	F_1
2	1	0	F_2
3	1	1	F_3

Tabela 7.2: Tabela verdade para funções de 2 variáveis.

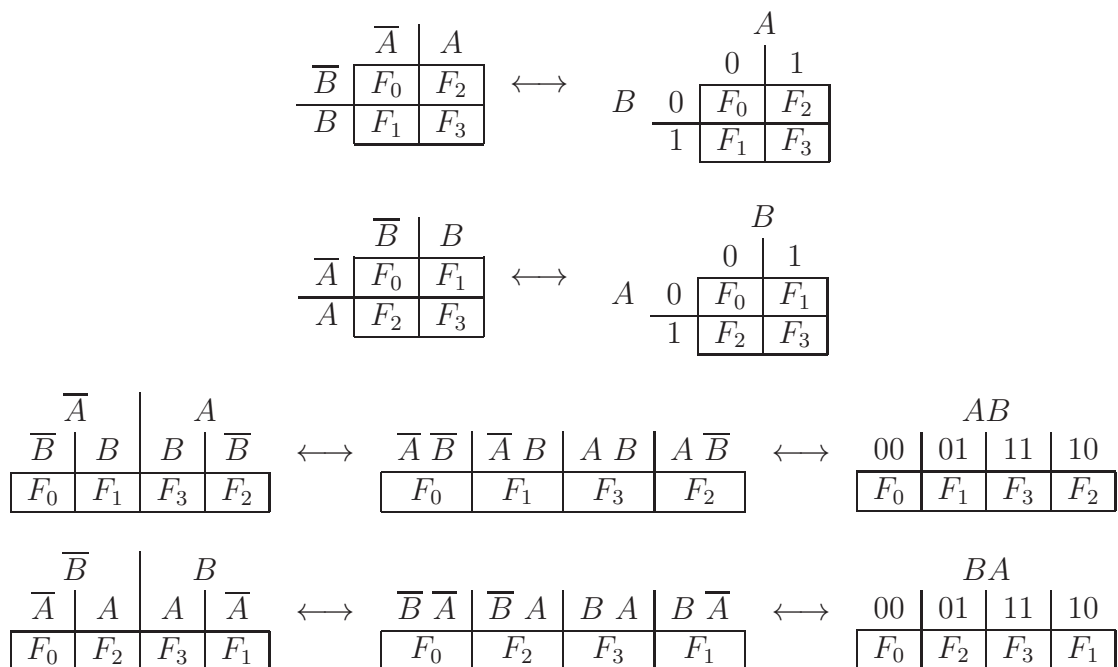


Figura 7.2: Exemplos de mapas de Karnaugh para funções de 2 variáveis.

7.2.3 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 7.3: Tabela verdade para funções de 3 variáveis.

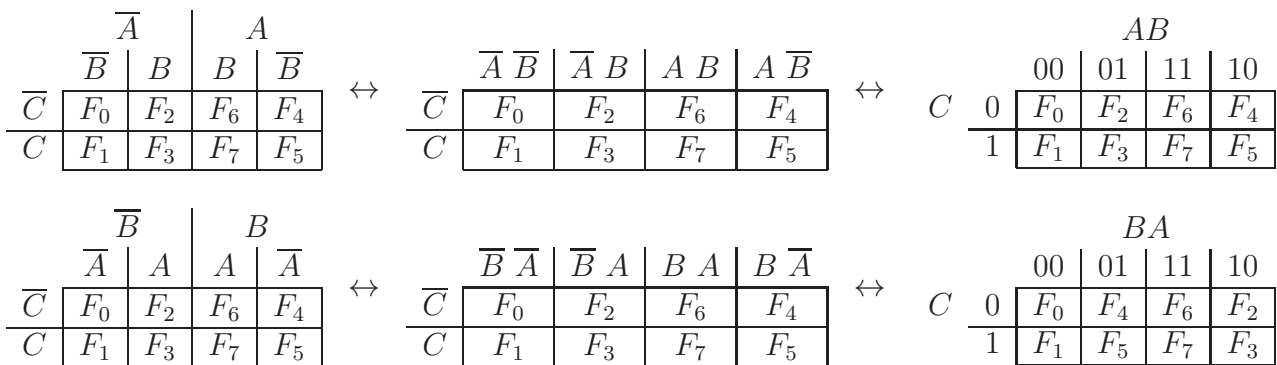


Figura 7.3: Exemplos de mapas de Karnaugh para funções de 3 variáveis.

7.2.4 Funções de 4 variáveis

<i>Linha</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	$F(A, B, C, D)$
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 7.4: Tabela verdade para funções de 4 variáveis.

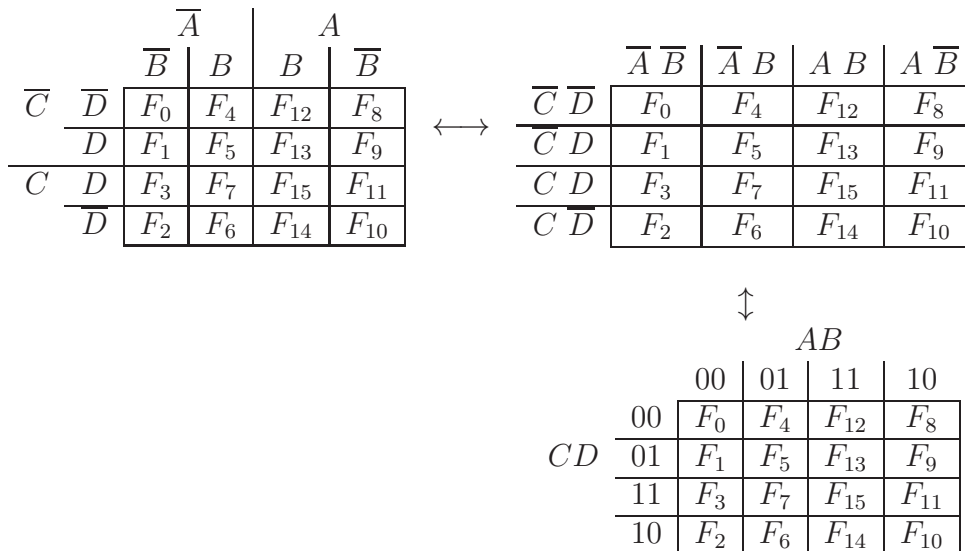


Figura 7.4: Exemplos de mapas de Karnaugh para funções de 4 variáveis.

7.3 Preenchimento do mapa-K

- 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 é preenchida com o respectivo valor lógico da função (0 ou 1).
- Para montar e simplificar uma forma SOP, deve-se manter os valores lógicos 1 (mintermos) no mapa e ignorar os valores lógicos 0 (maxtermos).
- Para montar e simplificar uma forma POS, deve-se manter os valores lógicos 0 (maxtermos) no mapa e ignorar os valores lógicos 1 (mintermos).

7.4 Mapa-K como forma de expressão de função booleana

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

$$F(A, B, C) = \overline{\overline{A + (B + C)}}. \quad (7.1)$$

Após alguma manipulação algébrica, a Equação (7.1) pode gerar a forma POS

$$F(A, B, C) = (A) \cdot (B + C) \quad (7.2)$$

e a forma SOP

$$F(A, B, C) = (A \cdot B) + (A \cdot C). \quad (7.3)$$

Expandindo-se os termos das Equações (7.2) e (7.3), obtém-se, respectivamente, a forma padrão POS

$$\begin{aligned} F(A, B, C) &= (A + B + C) \cdot (A + B + \overline{C}) \cdot (A + \overline{B} + C) \cdot (A + \overline{B} + \overline{C}) \cdot (\overline{A} + B + C) \\ &= \prod M(0, 1, 2, 3, 4) \end{aligned} \quad (7.4)$$

e a forma padrão SOP

$$\begin{aligned} F(A, B, C) &= (A \cdot \overline{B} \cdot C) + (A \cdot B \cdot \overline{C}) + (A \cdot B \cdot C) \\ &= \sum m(5, 6, 7). \end{aligned} \quad (7.5)$$

Por sua vez, a tabela verdade referente à Equação (7.1) é apresentada na Tabela 7.5.

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 7.5: Tabela verdade relativa à Equação (7.1).

Finalmente, o mapa-K da função é mostrado na Figura 7.5.

		<i>AB</i>			
		00	01	11	10
<i>C</i>	0	0	0	1	0
	1	0	0	1	1

Figura 7.5: Mapa de Karnaugh relativo à Equação (7.1).

7.5 Mapa-K na simplificação de expressões booleanas

7.5.1 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 \bar{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

$$\begin{aligned} F(A, B, C) &= (\bar{A} \cdot \bar{B} \cdot \bar{C}) + (A \cdot \bar{B} \cdot \bar{C}) + (\bar{A} \cdot B \cdot C) + (A \cdot B \cdot C) \\ &= (\bar{B} \cdot \bar{C}) + (B \cdot C). \end{aligned} \quad (7.6)$$

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

$$\begin{aligned} F(A, B, C) &= (\bar{A} \cdot \bar{B} \cdot \bar{C}) + (\bar{A} \cdot B \cdot \bar{C}) + (A \cdot \bar{B} \cdot \bar{C}) \\ &= (\bar{A} \cdot \bar{B} \cdot \bar{C}) + (\bar{A} \cdot B \cdot \bar{C}) + (A \cdot \bar{B} \cdot \bar{C}) + (\bar{A} \cdot \bar{B} \cdot \bar{C}) \\ &= (\bar{A} \cdot \bar{C}) + (\bar{B} \cdot \bar{C}). \end{aligned} \quad (7.7)$$

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 7.6 exemplifica a Equação (7.6), onde são realizadas as combinações $m_0 + m_4$ e $m_3 + m_7$.

		AB			
		00	01	11	10
C	0	1	0	0	1
	1	0	1	1	0

Figura 7.6: Mapa de Karnaugh relativo à Equação (7.6).

No mapa, a replicação é interpretada como a combinação de um termo com os demais geometricamente adjacentes. O mapa da Figura 7.7 exemplifica a Equação (7.7), onde o mintermo m_0 é replicado para as combinações $m_0 + m_2$ e $m_0 + m_4$.

		AB			
		00	01	11	10
C	0	1	1	0	1
	1	0	0	0	0

Figura 7.7: Mapa de Karnaugh relativo à Equação (7.7).

7.5.2 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.

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.

7.5.3 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 7.8 apresenta o mapa de uma função com múltiplas formas mínimas, envolvendo o termo m_2 , que possui duas soluções de mesmo custo: i) $(m_0 + m_4), (m_3 + m_7), (m_0 + m_2)$ e ii) $(m_0 + m_4), (m_3 + m_7), (m_3 + m_2)$.

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 7.9 apresenta um mapa com ciclo, que possui duas soluções de mesmo custo: i) $(m_0 + m_2), (m_3 + m_7), (m_4 + m_5)$ e ii) $(m_0 + m_4), (m_2 + m_3), (m_5 + m_7)$.

		AB			
		00	01	11	10
C	0	1	1	0	1
	1	0	1	1	0

Figura 7.8: Mapa de Karnaugh com múltiplas formas mínimas.

		AB			
		00	01	11	10
C	0	1	1	0	1
	1	0	1	1	1

Figura 7.9: Mapa de Karnaugh com ciclo.

7.5.4 Indeterminações: *don't-care* e *can't-happen*

Em alguns problemas, as funções booleanas podem não ser 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 (*don't-care*). Por outro lado, pode acontecer que uma determinada combinação de literais nunca ocorra (*can't-happen*). Em ambas as situações, pode-se especificar livremente qualquer um dos valores lógicos para a função. Para caracterizar o aspecto indeterminado da sua especificação, costuma-se atribuir um valor lógico indeterminado X .

Os valores indeterminados podem ser utilizados no processo de simplificação de formas padrões contendo mintermos ou maxtermos.

A Tabela 7.6 exemplifica uma função incompletamente especificada, a qual também pode ser expressa por

$$F(A, B, C) = \sum m(0, 3, 4) + \sum d(2, 7) = \prod M(1, 5, 6) \cdot \prod d(2, 7). \quad (7.8)$$

As Figuras 7.10–7.12 ilustram os mapas de Karnaugh da função, de seus mintermos e de seus maxtermos, respectivamente.

<i>Linha</i>	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 7.6: Tabela verdade de função incompletamente especificada.

		<i>AB</i>			
		00	01	11	10
<i>C</i>	0	1	X	0	1
	1	0	1	X	0

Figura 7.10: Mapa de Karnaugh da Tabela 7.6.

		<i>AB</i>			
		00	01	11	10
<i>C</i>	0	1	X		1
	1		1	X	

Figura 7.11: Mapa de Karnaugh dos mintermos da Tabela 7.6.

		<i>AB</i>			
		00	01	11	10
<i>C</i>	0		X	0	
	1	0		X	0

Figura 7.12: Mapa de Karnaugh dos maxtermos da Tabela 7.6.

Da configuração de mintermos apresentada no mapa da Figura 7.11, pode-se escrever que

$$\begin{aligned}
 F(A, B, C) &= \sum m(0, 3, 4) + \sum d(2, 7) \\
 &= (m_0 + m_4) + (m_3 + d_2) = (\overline{B} \cdot \overline{C}) + (\overline{A} \cdot B) \\
 &= (m_0 + m_4) + (m_3 + d_7) = (\overline{B} \cdot \overline{C}) + (B \cdot C). \quad (7.9)
 \end{aligned}$$

Da configuração de maxtermos apresentada no mapa da Figura 7.12, pode-se escrever que

$$\begin{aligned}
 F(A, B, C) &= \prod M(1, 5, 6) \cdot \prod d(2, 7) \\
 &= (M_1 \cdot M_5) \cdot (M_6 \cdot d_2) = (B + \overline{C}) \cdot (\overline{B} + C) \\
 &= (M_1 \cdot M_5) \cdot (M_6 \cdot d_7) = (B + \overline{C}) \cdot (\overline{A} + \overline{B}). \quad (7.10)
 \end{aligned}$$

As Equações 7.9 e 7.10 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.

7.6 Exercícios propostos

1. Obtenha, utilizando o Mapa de Karnaugh, as formas SOP mínima e POS mínima para as seguintes equações booleanas:

(a) $F(A, B, C) = (\bar{A} \cdot \bar{B} \cdot \bar{C}) + (\bar{A} \cdot B \cdot \bar{C}) + (A \cdot \bar{B} \cdot C) + (A \cdot B \cdot C).$

(b) $F(A, B, C) = (\bar{A} \cdot \bar{B} \cdot C) + (\bar{A} \cdot B \cdot C) + (A \cdot \bar{B} \cdot \bar{C}) + (A \cdot B \cdot \bar{C}).$

(c) $F(A, B, C) = (\bar{A} \cdot \bar{B} \cdot \bar{C}) + (\bar{A} \cdot \bar{B} \cdot C) + (A \cdot B \cdot \bar{C}) + (A \cdot B \cdot C).$

(d) $F(A, B, C) = (\bar{A} \cdot B \cdot \bar{C}) + (\bar{A} \cdot B \cdot C) + (A \cdot \bar{B} \cdot \bar{C}) + (A \cdot \bar{B} \cdot C).$

(e) $F(A, B, C) = (\bar{A} \cdot \bar{B} \cdot \bar{C}) + (\bar{A} \cdot B \cdot C) + (A \cdot \bar{B} \cdot \bar{C}) + (A \cdot B \cdot C).$

(f) $F(A, B, C) = (\bar{A} \cdot \bar{B} \cdot C) + (\bar{A} \cdot B \cdot \bar{C}) + (A \cdot \bar{B} \cdot C) + (A \cdot B \cdot \bar{C}).$

2. Obtenha, utilizando o Mapa de Karnaugh, as formas SOP mínima e POS mínima para as seguintes equações booleanas:

(a) $F(A, B, C) = (A + B + \bar{C}) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + C) \cdot (\bar{A} + \bar{B} + C).$

(b) $F(A, B, C) = (A + B + C) \cdot (A + \bar{B} + C) \cdot (\bar{A} + B + \bar{C}) \cdot (\bar{A} + \bar{B} + \bar{C}).$

(c) $F(A, B, C) = (A + \bar{B} + C) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + C) \cdot (\bar{A} + B + \bar{C}).$

(d) $F(A, B, C) = (A + B + C) \cdot (A + B + \bar{C}) \cdot (\bar{A} + \bar{B} + C) \cdot (\bar{A} + \bar{B} + \bar{C}).$

(e) $F(A, B, C) = (A + B + \bar{C}) \cdot (A + \bar{B} + C) \cdot (\bar{A} + B + \bar{C}) \cdot (\bar{A} + \bar{B} + C).$

(f) $F(A, B, C) = (A + B + C) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + C) \cdot (\bar{A} + \bar{B} + \bar{C}).$

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

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

4. 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. $F(A, B, C) = \sum m(0, 2, 3, 5, 7)$

- ii. $F(A, B, C) = \sum m(0, 1, 3, 4, 5)$
- iii. $F(A, B, C) = \sum m(0, 1, 2, 4, 6, 7)$
- iv. $F(A, B, C, D) = \sum m(4, 5, 11, 13, 15)$
- v. $F(A, B, C, D) = \sum m(0, 1, 5, 6, 7, 14)$
- vi. $F(A, B, C, D) = \sum m(0, 1, 2, 6, 7, 8, 9, 10, 14)$
- vii. $F(A, B, C, D) = \sum m(0, 1, 2, 5, 6, 7, 8, 9, 10, 14)$
- viii. $F(A, B, C, D) = \sum m(0, 1, 2, 6, 7, 8, 9, 10, 14, 15)$
- ix. $F(A, B, C, D) = \sum m(0, 1, 2, 5, 6, 7, 8, 9, 10, 14, 15)$
- x. $F(A, B, C, D) = \sum m(0, 1, 2, 5, 6, 7, 8, 9, 10, 13, 14, 15)$
- xi. $F(A, B, C, D) = \sum m(0, 2, 6, 7, 8, 9, 10, 12, 13)$
- xii. $F(A, B, C, D) = \sum m(0, 2, 6, 7, 8, 9, 10, 12, 13, 15)$
- xiii. $F(A, B, C, D) = \sum m(0, 1, 2, 3, 5, 7, 15)$
- xiv. $F(A, B, C, D) = \sum m(0, 1, 2, 3, 5, 7, 14, 15)$
- xv. $F(A, B, C, D) = \sum m(0, 1, 2, 3, 5, 7, 11, 15)$
- xvi. $F(A, B, C, D) = \sum m(0, 1, 2, 3, 5, 7, 13, 15)$
- xvii. $F(A, B, C, D) = \sum m(0, 2, 5, 6, 8, 10, 13)$
- xviii. $F(A, B, C, D) = \sum m(0, 2, 5, 7, 8, 10, 11, 13, 15)$
- xix. $F(A, B, C, D) = \sum m(1, 5, 6, 7, 11, 12, 13, 15)$
- xx. $F(A, B, C, D) = \sum m(2, 3, 4, 5, 6, 7, 10, 12, 13, 15)$
- xxi. $F(A, B, C, D) = \sum m(1, 2, 4, 6, 7, 9, 11, 12, 13, 14, 15)$

Capítulo 8

Sistemas de numeração

8.1 Introdução

- Sistema numérico (*number system*) \times 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 $\mathbf{S} = \{s_1, s_2, \dots, s_M\}$.
 - Os números x são representados por grupos de dígitos (vetores) pertencentes a \mathbf{S} : $\mathbf{x} = [d_N, \dots, d_2, d_1]_S$, 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: $\mathbf{w} = [w_N, \dots, w_2, w_1]$.
 - 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: $\mathbf{m} = [m_1, m_2, \dots, m_N]$.
 - 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: $\mathbf{x} = [r_1, r_2, \dots, r_N]_m$.
 - 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 $\mathbf{L}_\mu = \{x \mid |x| = \mu^i, i \in \mathbb{Z}\} \cup \{0\}$.
 - É objetivada uma melhoria de precisão na representação dos números, conseguida através de arredondamento geométrico.

8.2 Sistema de numeração posicional convencional

Nos itens que se seguem, são abordados diversos 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.

8.2.1 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) $d_i \in \mathbf{S} = \{s_1, s_2, \dots, s_M\} = \{0, 1, 2, \dots, (b-1)\}$, juntamente com uma técnica de ponderação ou escala. O número de elementos de \mathbf{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 $w_i \in \mathbf{W} = \{w_1, w_2, w_3, \dots\} = \{b^0, b^1, b^2, \dots\}$.

Uma visão geométrica modular do processo de representação pode ser encontrada nas Figuras 8.1 – 8.4, para $b = 3$. Para representar cada uma das quantidades $q < b$, é utilizado apenas um dos elementos de \mathbf{S} , como na Figura 8.1. Para as quantidades $q \geq b$, como não existem outros símbolos disponíveis, repetem-se os elementos de \mathbf{S} , em módulos de comprimento b , como exemplificado na Figura 8.2. Porém, isso gera ambigüidade na representação, a qual é resolvida através da combinação de símbolos, como ilustrado na Figura 8.3. Agora, a cada módulo de b símbolos, no nível básico de representação $L = 0$, são justapostos os elementos de \mathbf{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 8.4.

Em cada nível da representação existe um módulo formado pelos símbolos de \mathbf{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 $\mathbf{W} = \{b^0, b^1, b^2, \dots\}$. As mudanças de símbolos, dentro de cada nível, são reguladas pelo fator de escala do nível. Dessa forma, dentro de cada nível $L = 0, 1, 2, \dots, (N-1)$, ocorre uma mudança de símbolos a cada b^L unidades da quantidade representada.

0	1	2
---	---	---

Figura 8.1: Representação de quantidades $q < b$, para $b = 3$.

0	1	2	0	1	2	0	1	2
---	---	---	---	---	---	---	---	---

Figura 8.2: Representação de quantidades $q \geq b$, para $b = 3$, com ambigüidade.

0	1	2	0	1	2	0	1	2
0			1			2		

Figura 8.3: 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														

Figura 8.4: 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 $d_k \in \mathbf{S}$, ponderado por um fator $w_k \in \mathbf{W}$, conforme a Equação (8.1). A notação pode ser simplificada através da justaposição dos dígitos, acompanhada da especificação da base, como ilustrado na Equação (8.2). Nos casos onde se opera sempre com a mesma base, a sua indicação pode ser omitida, como na Equação (8.3).

$$(q_I)_b = (d_{N-1} \times b^{N-1}) + \cdots + (d_2 \times b^2) + (d_1 \times b^1) + (d_0 \times b^0) = \sum_{k=0}^{N-1} d_k b^k. \quad (8.1)$$

$$(q_I)_b = \sum_{k=0}^{N-1} d_k b^k = [d_{N-1} \cdots d_2 d_1 d_0]_b. \quad (8.2)$$

$$q_I = \sum_{k=0}^{N-1} d_k b^k = [d_{N-1} \cdots d_2 d_1 d_0]. \quad (8.3)$$

8.2.2 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 (8.1) a (8.3), pode-se dizer que

$$(q_F)_b = b^{-N} \cdot (q_I)_b = (d_{N-1} \times b^{-1}) + \cdots + (d_1 \times b^{-N+1}) + (d_0 \times b^{-N}) = \sum_{k=0}^{N-1} d_k b^{-N+k} \quad (8.4)$$

ou, utilizando-se uma notação mais genérica, que

$$(q_F)_b = (d_{-1} \times b^{-1}) + (d_{-2} \times b^{-2}) + \cdots + (d_{-N} \times b^{-N}) = \sum_{k=-N}^{-1} d_k b^k = [d_{-1} d_{-2} \cdots d_{-N}]_b. \quad (8.5)$$

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 (8.6), para números puramente inteiros, na Equação (8.7), para números puramente fracionários, e na Equação (8.8), para números com partes inteira e fracionária.

$$(q_I)_b = \sum_{k=0}^{N_I-1} d_k b^k = [d_{N_I-1} \cdots d_1 d_0 \cdot]_b = [d_{N_I-1} \cdots d_1 d_0 \cdot 0]_b . \quad (8.6)$$

$$(q_F)_b = \sum_{k=-N_F}^{-1} d_k b^k = [\cdot d_{-1} d_{-2} \cdots d_{-N_F}]_b = [0 \cdot d_{-1} d_{-2} \cdots d_{-N_F}]_b . \quad (8.7)$$

$$(q)_b = (q_I)_b + (q_F)_b = \sum_{k=-N_F}^{N_I-1} d_k b^k = [d_{N_I-1} \cdots d_2 d_1 d_0 \cdot d_{-1} d_{-2} \cdots d_{-N_F}]_b . \quad (8.8)$$

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.

8.2.3 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.
- Representação numérica
 - Sistema: SNPC com base b .
 - Dígitos: $d_i \in S = \{0, 1, 2, \dots, (b-1)\}$.
 - 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 positivos
 - Dígito mais significativo: $d_{N-1} = s_{N-1} = 0$.
 - Representação: $(q_{I+})_b = [s_{N-1}d_{N-2} \cdots d_2d_1d_0]_b = [0 d_{N-2} \cdots d_2d_1d_0]_b$.
 - Codificação: sinal-e-magnitude.
- Números negativos
 - Dígito mais significativo: $d'_{N-1} = s'_{N-1} = (b - 1)$.
 - Representação: $(q_{I-})_b = [s'_{N-1}d'_{N-2} \cdots d'_2d'_1d'_0]_b = [(b - 1) d'_{N-2} \cdots d'_2d'_1d'_0]_b$.
 - Codificações:
 - * Sinal-e-magnitude.
 - * Sinal-e-complemento:
 - 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 8.5 a 8.10 apresentam uma visão geométrica das codificações.
- Na representação de números inteiros não negativos, considerando-se $b = 2$, é utilizada a justaposição de N dígitos para representar 2^N valores consecutivos, na faixa $[0; (2^N - 1)]$. Por exemplo, para $N = 4$, o valor $V = (6)_{10}$ é representado por $V = (0110)_2$. Isso é ilustrado na Figura 8.5, para $N = 4$.
- Para possibilitar a representação de números inteiros negativos, o dígito mais significativo é utilizado para simbolizar os sinais “+” e “-”. Normalmente, são adotados “0” e “1”, respectivamente. Portanto, uma metade dos 2^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 $N = 4$, o valor $V = (+6)_{10}$ é representado por $V_{SM} = (0110)_2$. Isso é ilustrado na Figura 8.6, para $N = 4$.
- Por outro lado, para os valores negativos, três outros tipos de associação são comumente empregados.
- 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 “1”, para representar o sinal “-”. Por exemplo, para $N = 4$, o valor $V = (-6)_{10}$ é representado por $V_{SM} = (1110)_2$. Isso é ilustrado na Figura 8.7, para $N = 4$.
- Nas codificações denominadas de complementares, um valor negativo ($V = -|V|$) é representado por seu valor complementar (V_C) em relação a um determinado valor de referência (V_R), de tal forma que $V_C = V_R - |V|$.

- Na codificação denominada de Complemento-a-1, o valor de referência é igual ao maior valor representável ($V_R = b^N - 1 = 2^N - 1$). Por exemplo, para $N = 4$, o valor $V = (-6)_{10}$ é representado pelo valor complementar $V_{C1} = V_R - |V| = (15 - |6|)_{10} = (9)_{10} = (1001)_2$. Isso é ilustrado na Figura 8.8, para $N = 4$.
- Na codificação denominada de Complemento-a-2, o valor de referência é o valor seguinte ao maior valor representável ($V_R = b^N = 2^N$). Por exemplo, para $N = 4$, o valor $V = (-6)_{10}$ é representado pelo valor complementar $V_{C1} = V_R - |V| = (16 - |6|)_{10} = (10)_{10} = (1010)_2$. Isso é ilustrado na Figura 8.9, para $N = 4$.
- Para melhor comparação, todas as codificações são reunidas na Figura 8.10, para $N = 4$.

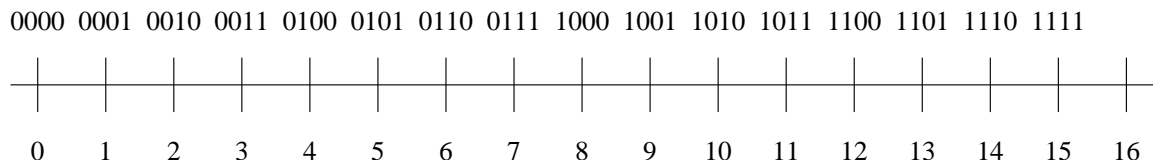


Figura 8.5: Mapeamento decimal-binário para números não negativos e $N = 4$, sem sinal.

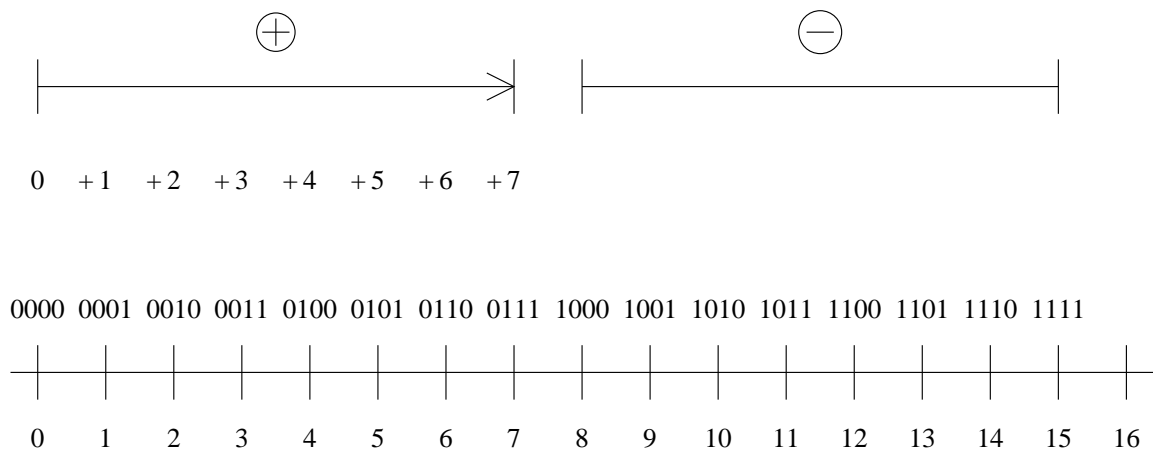


Figura 8.6: Mapeamento decimal-binário para números não negativos e $N = 4$, com sinal.

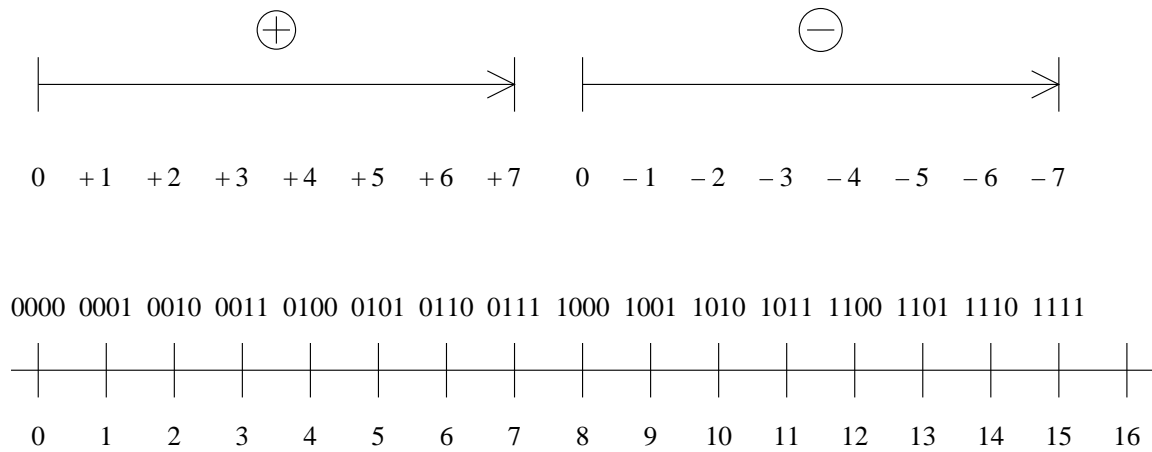


Figura 8.7: Mapeamento decimal-binário para números negativos e $N = 4$, com codificação Sinal-e-Magnitude.

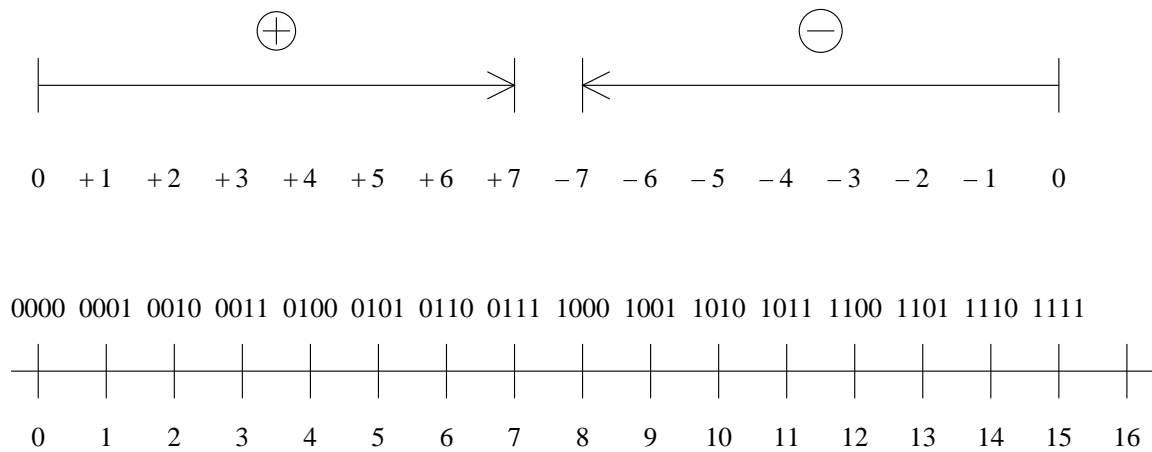


Figura 8.8: Mapeamento decimal-binário para números negativos e $N = 4$, com codificação Complemento-a-1.

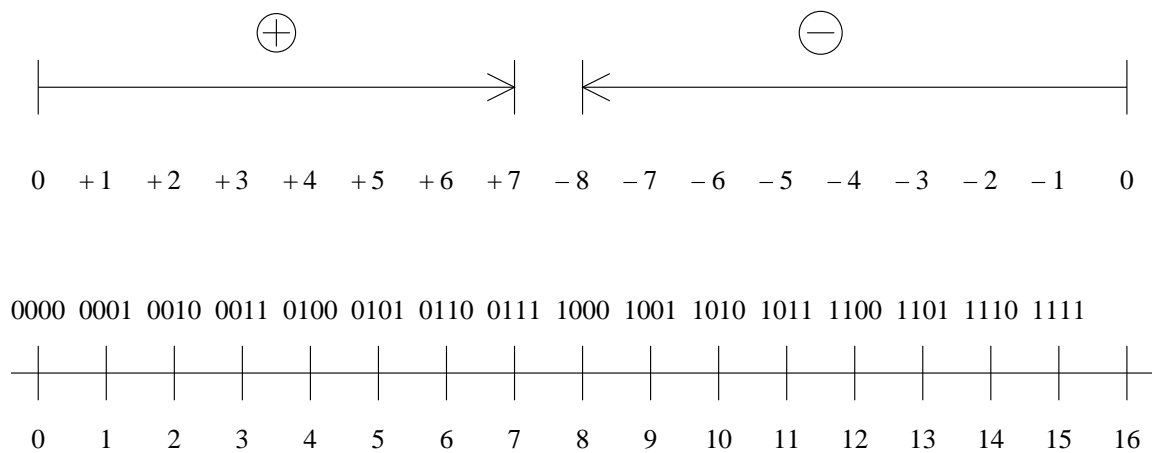


Figura 8.9: Mapeamento decimal-binário para números negativos e $N = 4$, com codificação Complemento-a-2.

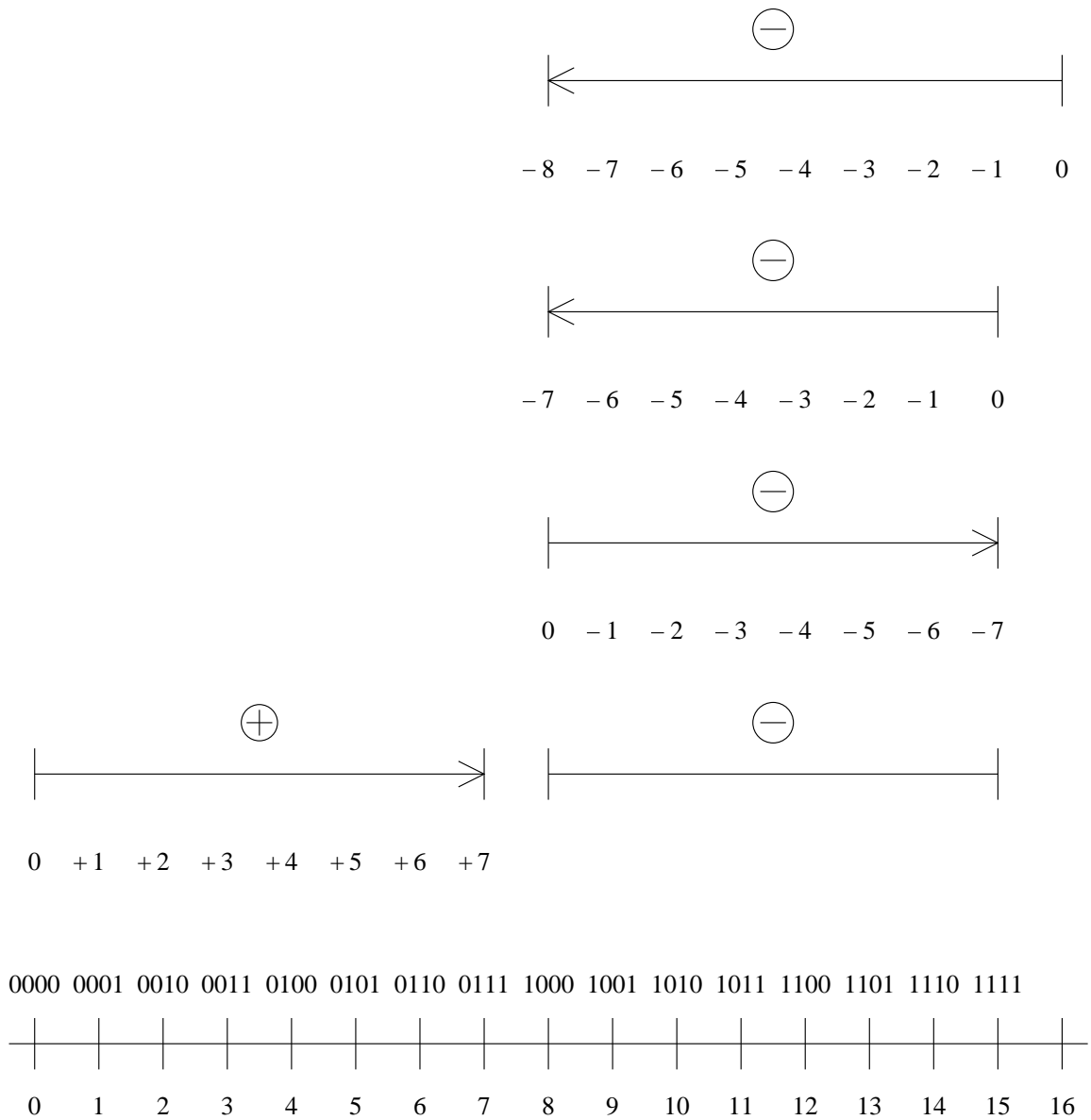


Figura 8.10: Comparação dos mapeamentos decimal-binário para números negativos e $N = 4$.

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 (8.9) apresenta uma interpretação numérica da representação, para $b = 2$.
- Um exemplo é apresentado na Tabela 8.1, para $b = 2$ e $N = 4$.

$$\begin{aligned}
 (q_I)_2 &= [0/1 d_{N-2} \cdots d_2 d_1 d_0]_2 = [s_{N-1} d_{N-2} \cdots d_2 d_1 d_0]_2 \\
 &= (-1)^{s_{N-1}} \times [(d_{N-2} \times 2^{N-2}) + \cdots + (d_2 \times 2^2) + (d_1 \times 2^1) + (d_0 \times 2^0)] \\
 &= (-1)^{s_{N-1}} \times \left(\sum_{k=0}^{N-2} d_k 2^k \right)
 \end{aligned} \tag{8.9}$$

	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 8.1: Tabela de sinal-e-magnitude, para número inteiros, $b = 2$ e $N = 4$.

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: $-|x| \leftrightarrow x_C = (b^N - 1) - |x|$.
 - As Equações (8.10) e (8.11) apresentam uma interpretação numérica da representação, para $b = 2$.

$$(q_{I-})_2 = -|x| = - \left(\sum_{k=0}^{N-1} d_k 2^k \right) = - [d_{N-1} d_{N-2} \cdots d_1 d_0]_2 = - [0 d_{N-2} \cdots d_1 d_0]_2 \quad (8.10)$$

$$\begin{aligned} x_{C1} &= (2^N - 1) - |x| = (2^N - 1) - \left(\sum_{k=0}^{N-1} d_k 2^k \right) \\ &= [1 1 \cdots 1 1]_2 - [0 d_{N-2} \cdots d_1 d_0]_2 = [1 d'_{N-2} \cdots d'_1 d'_0]_2 \end{aligned} \quad (8.11)$$

- 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 (8.12) e (8.13) apresentam uma interpretação numérica da representação, para $b = 2$.

$$\begin{aligned} (q_{I+})_2 &= [0 d_{N-2} \cdots d_2 d_1 d_0]_2 \\ &= [s_{N-1} d_{N-2} \cdots d_2 d_1 d_0]_2 \\ &= s_{N-1} \cdot [-(2^{N-1} - 1)] + [(d_{N-2} \times 2^{N-2}) + \cdots + (d_2 \times 2^2) + (d_1 \times 2^1) + (d_0 \times 2^0)] \\ &= s_{N-1} \cdot [-(2^{N-1} - 1)] + \left(\sum_{k=0}^{N-2} d_k 2^k \right) \end{aligned} \quad (8.12)$$

$$\begin{aligned} (q_{I-})_2 &= [1 d'_{N-2} \cdots d'_2 d'_1 d'_0]_2 \\ &= [s'_{N-1} d'_{N-2} \cdots d'_2 d'_1 d'_0]_2 \\ &= s'_{N-1} \cdot [-(2^{N-1} - 1)] + [(d'_{N-2} \times 2^{N-2}) + \cdots + (d'_2 \times 2^2) + (d'_1 \times 2^1) + (d'_0 \times 2^0)] \\ &= s'_{N-1} \cdot [-(2^{N-1} - 1)] + \left(\sum_{k=0}^{N-2} d'_k 2^k \right) \end{aligned} \quad (8.13)$$

- Um exemplo é apresentado na Tabela 8.2, 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 8.2: Tabela de complemento-a-1, para número inteiros, $b = 2$ e $N = 4$.

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 (8.14) e (8.15) apresentam uma interpretação numérica da representação, para $b = 2$.

$$(q_{I-})_2 = -|x| = - \left(\sum_{k=0}^{N-1} d_k 2^k \right) = - [d_{N-1} d_{N-2} \cdots d_1 d_0]_2 = - [0 d_{N-2} \cdots d_1 d_0]_2 \quad (8.14)$$

$$x_{C2} = 2^N - |x| = 2^N - \left(\sum_{k=0}^{N-1} d_k 2^k \right) = [1 0 0 \cdots 0 0]_2 - [0 d_{N-2} \cdots d_1 d_0]_2 = [1 d'_{N-2} \cdots d'_1 d'_0]_2 \quad (8.15)$$

- 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 (8.16) e (8.17) apresentam uma interpretação numérica da representação, para $b = 2$.

$$\begin{aligned} (q_{I+})_2 &= [0 d_{N-2} \cdots d_2 d_1 d_0]_2 \\ &= [s_{N-1} d_{N-2} \cdots d_2 d_1 d_0]_2 \\ &= s_{N-1} \cdot [-(2^{N-1})] + [(d_{N-2} \times 2^{N-2}) + \cdots + (d_2 \times 2^2) + (d_1 \times 2^1) + (d_0 \times 2^0)] \\ &= s_{N-1} \cdot [-(2^{N-1})] + \left(\sum_{k=0}^{N-2} d_k 2^k \right) \end{aligned} \quad (8.16)$$

$$\begin{aligned} (q_{I-})_2 &= [1 d'_{N-2} \cdots d'_2 d'_1 d'_0]_2 \\ &= [s'_{N-1} d'_{N-2} \cdots d'_2 d'_1 d'_0]_2 \\ &= s'_{N-1} \cdot [-(2^{N-1})] + [(d'_{N-2} \times 2^{N-2}) + \cdots + (d'_2 \times 2^2) + (d'_1 \times 2^1) + (d'_0 \times 2^0)] \\ &= s'_{N-1} \cdot [-(2^{N-1})] + \left(\sum_{k=0}^{N-2} d'_k 2^k \right) \end{aligned} \quad (8.17)$$

- Um exemplo é apresentado na Tabela 8.3, 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 se obter sua representação complementar basta: i) que se troque os numerais 0 por 1 e que se troque 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 se obter sua representação complementar deve-se realizar uma busca a partir do dígito menos significativo (LSB). Durante a busca, os dígitos não serão modificados até que seja encontrado o primeiro numeral 1, que também não será modificado. A partir deste ponto, basta que se troque os numerais 0 por 1 e que se troque os numerais 1 por 0.
 - Uma vez que 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 8.3: Tabela de complemento-a-2, para número inteiros, $b = 2$ e $N = 4$.

8.2.4 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 (8.14) e (8.15), não é difícil demonstrar a seguinte equivalência: $-|x_F| \leftrightarrow (x_F)_{C2} = 2 - |x_F|$. Finalmente, partindo-se das Equações (8.16) e (8.17), 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 8.1 a 8.3, que representam números inteiros, são transformadas nas Tabelas 8.4 a 8.6, para números puramente fracionários, através do fator de escala $FE = 2^{-(N-1)} = 2^{-3} = 8^{-1}$.
- Comparando-se os conteúdos das Tabelas 8.1 a 8.6, 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 8.4: 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 8.5: 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 8.6: Tabela de complemento-a-2, para números puramente fracionários, $b = 2$ e $N = 4$.

8.2.5 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 (8.18), deseja-se encontrar os dígitos d_i da Equação (8.19). Considerando-se todas as quantidades expressas na base s , podem-se definir as relações expressas na Equação (8.20). 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.

$$(q)_s = [d'_J \cdots d'_1 d'_0]_s = (N_0)_s . \quad (8.18)$$

$$(q)_t = [d_K \cdots d_1 d_0]_t . \quad (8.19)$$

$$\begin{aligned} N_0 &= (d_K \times t^K + \cdots + d_2 \times t^2 + d_1 \times t^1 + d_0 \times t^0) \\ &= (d_K \times t^{K-1} + \cdots + d_2 \times t^1 + d_1 \times t^0) \times t + (d_0 \times t^0) \\ &= N_1 \times t + d_0 \end{aligned}$$

$$\begin{aligned} N_1 &= (d_K \times t^{K-1} + \cdots + d_2 \times t^1 + d_1 \times t^0) \\ &= (d_K \times t^{K-2} + \cdots + d_2 \times t^0) \times t + (d_1 \times t^0) \\ &= N_2 \times t + d_1 \end{aligned}$$

⋮

$$\begin{aligned} N_{K-1} &= (d_K \times t^1 + d_{K-1} \times t^0) \\ &= (d_K) \times t + (d_{K-1} \times t^0) \\ &= N_K \times t + d_{K-1} \end{aligned}$$

$$N_K = d_K . \quad (8.20)$$

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 (8.21), deseja-se encontrar os dígitos d_i da Equação (8.22). Considerando-se todas as quantidades expressas na base s , podem-se definir as relações expressas na Equação (8.23). 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.

$$(q)_s = [d'_{-1}d'_{-2} \cdots d'_{-j}]_s = (N_{-1})_s . \quad (8.21)$$

$$(q)_t = [d_{-1}d_{-2} \cdots d_{-K}]_t . \quad (8.22)$$

$$\begin{aligned} N_{-1} \times t &= (d_{-1} \times t^{-1} + d_{-2} \times t^{-2} + d_{-3} \times t^{-3} + \cdots + d_{-K} \times t^{-K}) \times t \\ &= (d_{-1} \times t^0) + (d_{-2} \times t^{-1} + d_{-3} \times t^{-2} + \cdots + d_{-K} \times t^{-K+1}) \\ &= d_{-1} + N_{-2} \\ \\ N_{-2} \times t &= (d_{-2} \times t^{-1} + d_{-3} \times t^{-2} + \cdots + d_{-K} \times t^{-K+1}) \times t \\ &= (d_{-2} \times t^0) + (d_{-3} \times t^{-1} + \cdots + d_{-K} \times t^{-K+2}) \\ &= d_{-2} + N_{-3} \\ \\ &\vdots \\ \\ N_{-K+1} \times t &= (d_{-K+1} \times t^{-1} + d_{-K} \times t^{-2}) \times t \\ &= (d_{-K+1} \times t^0) + (d_{-K} \times t^{-1}) \\ &= d_{-K+1} + N_{-K} \\ \\ N_{-K} \times t &= (d_{-K} \times t^{-1}) \times t \\ &= (d_{-K} \times t^0) \\ &= d_{-K} \end{aligned} \quad (8.23)$$

8.2.6 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 $d_i \in \mathbf{S} = \{0, 1, 2, \dots, 7\}$. Por sua vez, a base hexadecimal emprega $b = 16$ e dígitos $d_i \in \mathbf{S} = \{0, 1, 2, \dots, 9, A, B, \dots, F\}$. Supondo-se números não negativos e inteiros, as Equações (8.24) – (8.26) ilustram as notações nas três bases.

$$(q_I)_2 = (d_J \times 2^J) + \dots + (d_2 \times 2^2) + (d_1 \times 2^1) + (d_0 \times 2^0) . \quad (8.24)$$

$$(q_I)_8 = (d'_K \times 8^K) + \dots + (d'_2 \times 8^2) + (d'_1 \times 8^1) + (d'_0 \times 8^0) . \quad (8.25)$$

$$(q_I)_{16} = (d''_L \times 16^L) + \dots + (d''_2 \times 16^2) + (d''_1 \times 16^1) + (d''_0 \times 16^0) . \quad (8.26)$$

As bases binária, octal e hexadecimal são comumente utilizadas em conjunto, devido à facilidade de conversão entre as três bases. As Equações (8.27) – (8.31) ilustram a relação entre as bases binária e octal.

$$\begin{aligned} (q_I)_2 &= (d_J \times 2^J) + (d_{J-1} \times 2^{J-1}) + (d_{J-2} \times 2^{J-2}) + \dots + \\ &\quad (d_5 \times 2^5) + (d_4 \times 2^4) + (d_3 \times 2^3) + \\ &\quad (d_2 \times 2^2) + (d_1 \times 2^1) + (d_0 \times 2^0) \\ &= [(d_J \times 2^2) + (d_{J-1} \times 2^1) + (d_{J-2} \times 2^0)] \times 2^{J-2} + \dots + \\ &\quad [(d_5 \times 2^2) + (d_4 \times 2^1) + (d_3 \times 2^0)] \times 2^3 + \\ &\quad [(d_2 \times 2^2) + (d_1 \times 2^1) + (d_0 \times 2^0)] \times 2^0 \\ &= (d'_K \times 8^K) + \dots + (d'_1 \times 8^1) + (d'_0 \times 8^0) \\ &= (q_I)_8 . \end{aligned} \quad (8.27)$$

$$J - 2 = 3K . \quad (8.28)$$

$$[d_2 d_1 d_0]_2 = [d'_0]_8 . \quad (8.29)$$

$$[d_5 d_4 d_3]_2 = [d'_1]_8 . \quad (8.30)$$

$$[d_J d_{J-1} d_{J-2}]_2 = [d'_K]_8 . \quad (8.31)$$

As Equações (8.32) – (8.36) ilustram a relação entre as bases binária e hexadecimal.

$$\begin{aligned}
(q_I)_2 &= (d_J \times 2^J) + (d_{J-1} \times 2^{J-1}) + (d_{J-2} \times 2^{J-2}) + (d_{J-3} \times 2^{J-3}) + \dots + \\
&\quad (d_7 \times 2^7) + (d_6 \times 2^6) + (d_5 \times 2^5) + (d_4 \times 2^4) + \\
&\quad (d_3 \times 2^3) + (d_2 \times 2^2) + (d_1 \times 2^1) + (d_0 \times 2^0) \\
&= [(d_J \times 2^3) + (d_{J-1} \times 2^2) + (d_{J-2} \times 2^1) + (d_{J-3} \times 2^0)] \times 2^{J-3} + \dots + \\
&\quad [(d_7 \times 2^3) + (d_6 \times 2^2) + (d_5 \times 2^1) + (d_4 \times 2^0)] \times 2^4 + \\
&\quad [(d_3 \times 2^3) + (d_2 \times 2^2) + (d_1 \times 2^1) + (d_0 \times 2^0)] \times 2^0 \\
&= (d''_L \times 16^L) + \dots + (d''_1 \times 16^1) + (d''_0 \times 16^0) \\
&= (q_I)_{16} .
\end{aligned} \tag{8.32}$$

$$J - 3 = 4L . \tag{8.33}$$

$$[d_3 d_2 d_1 d_0]_2 = [d''_0]_{16} . \tag{8.34}$$

$$[d_7 d_6 d_5 d_4]_2 = [d''_1]_{16} . \tag{8.35}$$

$$[d_J d_{J-1} d_{J-2} d_{J-3}]_2 = [d''_L]_{16} . \tag{8.36}$$

As Equações (8.37) – (8.41) ilustram a relação entre as bases octal e hexadecimal.

$$\begin{aligned}
(q_I)_8 &= (d'_K \times 8^K) + (d'_{K-1} \times 8^{K-1}) + \dots + \\
&\quad (d'_3 \times 8^3) + (d'_2 \times 8^2) + \\
&\quad (d'_1 \times 8^1) + (d'_0 \times 8^0) \\
&= [(d'_K \times 8^1) + (d'_{K-1} \times 8^0)] \times 8^{K-1} + \dots + \\
&\quad [(d'_3 \times 8^1) + (d'_2 \times 8^0)] \times 8^2 + \\
&\quad [(d'_1 \times 8^1) + (d'_0 \times 8^0)] \times 8^0 \\
&= (d''_L \times 16^L) + \dots + (d''_1 \times 16^1) + (d''_0 \times 16^0) \\
&= (q_I)_{16} .
\end{aligned} \tag{8.37}$$

$$K - 1 = 2L . \tag{8.38}$$

$$[d'_1 d'_0]_8 = [d''_0]_{16} . \tag{8.39}$$

$$[d'_3 d'_2]_8 = [d''_1]_{16} . \tag{8.40}$$

$$[d'_K d'_{K-1}]_2 = [d''_L]_{16} . \tag{8.41}$$

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.

8.3 Quantização

8.3.1 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, intrinsecamente ligado à sua medida e/ou à sua representação, denominado erro de quantização.

8.3.2 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 8.7 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	$d_1 d_0$	$2^2 = 4$	$10^2 = 100$
3	$d_2 d_1 d_0$	$2^3 = 8$	$10^3 = 1000$
4	$d_3 d_2 d_1 d_0$	$2^4 = 16$	$10^4 = 10000$
5	$d_4 d_3 d_2 d_1 d_0$	$2^5 = 32$	$10^5 = 100000$
6	$d_5 d_4 d_3 d_2 d_1 d_0$	$2^6 = 64$	$10^6 = 1000000$

Tabela 8.7: 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 8.8 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 8.8: 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 8.9 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	$.000000000d_{-10}$	$2^{-9} = .0009765625$

Tabela 8.9: 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.

8.3.3 Classificações

Dependendo dos parâmetros considerados, a quantização pode assumir diversas classificações.

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.

8.4 Códigos numéricos

8.4.1 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.

8.4.2 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 8.10, 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.

8.4.3 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 8.11, 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 8.10: Relação entre os códigos numéricos decimal, binário, octal e hexadecimal, para um código binário com 4 *bits*.

Decimal	Binário	Gray	<i>One-hot</i>	Johnson
00	0000	0000	0000000000000001	00000000
01	0001	0001	0000000000000010	10000000
02	0010	0011	0000000000000100	11000000
03	0011	0010	0000000000001000	11100000
04	0100	0110	0000000000100000	11110000
05	0101	0111	0000000001000000	11111000
06	0110	0101	0000000010000000	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 8.11: Relação entre os códigos numéricos decimal, binário, Gray, *One-hot* e Johnson, para um código binário com 4 *bits*.

8.4.4 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 8.12.

Decimal	8421	<i>Excess-3</i>	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 8.12: 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 8.13 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 8.13: 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 8.14 apresenta os códigos *2-out-of-5* e *Biquinary*, que são compostos por 5 e 7 *bits*, respectivamente.

Decimal	<i>2-out-of-5</i>	<i>2-out-of-5</i>	<i>Biquinary</i> (50 43210)	<i>Biquinary</i> (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 8.14: 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

$$q = C_B + \sum_k w_k B_k, \quad (8.42)$$

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 8.15. Por sua vez, exemplos de códigos BCD não ponderados são apresentados na Tabela 8.16.

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 8.15: 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 8.16: Exemplos de códigos numéricos BCD não ponderados.

8.5 Aritmética binária

8.5.1 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 8.11 - 8.13 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 8.11: 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 8.12: 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 8.13: Tabelas de operações entre dígitos para $b = 4$: (a) adição e (b) multiplicação.

8.5.2 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 (8.43), é apresentada uma quantidade genérica q , representada na base b . A multiplicação de q pela base b é definida nas Equações (8.44) e (8.45). A divisão de q pela base b é definida nas Equações (8.46) e (8.47). De acordo com as Equações (8.43) e (8.47), a implementação do escalamento pode ser obtida através do simples deslocamento dos dígitos da representação.

$$(q)_b = (q_I)_b + (q_F)_b = \sum_{k=N_I}^{-N_F} d_k b^k = [d_{N_I} \cdots d_2 d_1 d_0 \cdot d_{-1} d_{-2} \cdots d_{-N_F}]_b . \quad (8.43)$$

$$\begin{aligned} (q')_b &= (q)_b \times b \\ &= \left(\sum_{k=N_I}^{-N_F} d_k b^k \right) \times b = \sum_{k=N_I}^{-N_F} d_k b^{k+1} = \sum_{k=N_I+1}^{-N_F+1} d_{k-1} b^k = \sum_{k=N_I+1}^{-N_F+1} d'_k b^k \\ &= [d_{N_I} \cdots d_2 d_1 d_0 d_{-1} \cdot d_{-2} \cdots d_{-N_F}]_b \\ &= [d'_{N_I+1} \cdots d'_3 d'_2 d'_1 d'_0 \cdot d'_{-1} d'_{-2} \cdots d'_{-N_F+1}]_b . \end{aligned} \quad (8.44)$$

$$d'_k = d_{k-1} . \quad (8.45)$$

$$\begin{aligned} (q')_b &= (q)_b \times b^{-1} \\ &= \left(\sum_{k=N_I}^{-N_F} d_k b^k \right) \times b^{-1} = \sum_{k=N_I}^{-N_F} d_k b^{k-1} = \sum_{k=N_I-1}^{-N_F-1} d_{k+1} b^k = \sum_{k=N_I-1}^{-N_F-1} d'_k b^k \\ &= [d_{N_I} \cdots d_2 d_1 \cdot d_0 d_{-1} d_{-2} \cdots d_{-N_F}]_b \\ &= [d'_{N_I-1} \cdots d'_3 d'_2 d'_1 d'_0 \cdot d'_{-1} d'_{-2} \cdots d'_{-N_F-1}]_b . \end{aligned} \quad (8.46)$$

$$d'_k = d_{k+1} . \quad (8.47)$$

8.5.3 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 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 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, operando com dados puramente fracionários, codificados em complemento-a-2, o sinal de saída *carry-out* representa uma parte inteira de valor $v_I = 2$.

- Caso 1: adição de números positivos.
 $0 \leq x_1 < 1, 0 \leq x_2 < 1$ e $x_A = x_1 + x_2$.
 Logo: $0 \leq x_A < 2$.
 Se $0 \leq x_A < 1$: adição sem *overflow*.
 Se $1 \leq x_A < 2$: adição com *overflow*.
- Caso 2: subtração de números positivos.
 $0 \leq x_1 < 1, -1 < x_2 < 0, (-|x_2|)_{C2} = 2 - |x_2|$ e
 $x_A = (x_1 + x_2) = x_1 - |x_2| = x_1 + (x_2)_{C2} = 2 + (x_1 - |x_2|)$.
 Logo: $-1 < x_A < 1$.
 Portanto, nesse caso, não haverá ocorrência de *overflow*.
 Se $x_1 \geq |x_2|$: resultado positivo, bastando ignorar o sinal de *carry-out* ($x_P = x_A - 2$).
 Se $x_1 < |x_2|$: resultado negativo já codificado ($x_N = x_A$).
- Caso 3: adição de números negativos.
 $-1 < x_1 < 0, -1 < x_2 < 0, (-|x_1|)_{C2} = 2 - |x_1|, (-|x_2|)_{C2} = 2 - |x_2|$ e
 $(x_{C2})_A = x_1 + x_2 = (-|x_1|)_{C2} + (-|x_2|)_{C2} = (2 - |x_1|) + (2 - |x_2|) = 2 + [2 - (|x_1| + |x_2|)]$.
 Logo: $0 < |x_1| + |x_2| < 2 \rightarrow 2 < (x_{C2})_A < 4$.
 Se $2 < (x_{C2})_A \leq 3$: adição com *overflow*.
 Se $3 < (x_{C2})_A < 4$: adição sem *overflow*.
 Se o resultado for sem *overflow*, o mesmo já estará codificado, bastando ignorar o sinal de *carry-out* ($(x_A)_{C2} = (x_{C2})_A - 2$).

Detecção e tratamento de *overflow* na adição em complemento-a-2

Pelos resultados da análise de *overflow* para adição em complemento-a-2, não é difícil encontrar um mecanismo que indique sua ocorrência. 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 8.17 e 8.18 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 8.17, são levados em consideração os casos que não podem acontecer (*can't happen*), o que pode levar a simplificações da função final. O tratamento de *overflow* mais comumente empregado é a saturação do resultado no valor máximo representável (positivo ou negativo).

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

$X = \text{can't happen}$

$OF = 0 \rightarrow \text{sem overflow}$

$OF = 1 \rightarrow \text{com overflow}$

Tabela 8.17: 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
			1	X	1
Subtração de positivos	0	1	0	X	0
			1	0	0
Adição de negativos	1	1	0	X	1
			1	1	0

$X = \text{don't care}$

$OF = 0 \rightarrow \text{sem overflow}$

$OF = 1 \rightarrow \text{com overflow}$

Tabela 8.18: Forma 2 para representar a detecção de *overflow* na adição em complemento-a-2.

8.5.4 Funções envolvidas na adição de dígitos binários

A adição de dígitos binários pode ser facilmente definida por meio de tabelas. A Figura 8.14 apresenta as tabelas que definem a adição entre dois dígitos binários (A e B). A Figura 8.14.(a) define a operação completa, com todos os resultados descritos por dois dígitos. A Figura 8.14.(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 8.14.(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 8.14: 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

$$C_o = f_{C_2}(A, B) = (A \cdot B)$$

e que

$$S = f_{S_2}(A, B) = (\bar{A} \cdot B) + (A \cdot \bar{B}) = (A \oplus B) .$$

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 8.15.

+	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 8.15: 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

$$\begin{aligned} S &= f_{S_3}(A, B, C_i) = (\bar{A} \cdot B \cdot \bar{C}_i) + (A \cdot \bar{B} \cdot \bar{C}_i) + (\bar{A} \cdot \bar{B} \cdot C_i) + (A \cdot B \cdot C_i) \\ &= [(A \oplus B) \cdot \bar{C}_i] + [(\overline{A \oplus B}) \cdot C_i] \\ &= (A \oplus B) \oplus C_i \end{aligned}$$

e que

$$C_o = f_{C3}(A, B, C_i) = (A \cdot B \cdot \overline{C_i}) + (A \cdot B \cdot C_i) + (\overline{A} \cdot B \cdot C_i) + (A \cdot \overline{B} \cdot C_i) ,$$

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

$$\begin{aligned} C_o &= f_{C3}(A, B, C_i) \\ &= (A \cdot B) + [(\overline{A} \cdot B) + (A \cdot \overline{B})] \cdot C_i \\ &= (A \cdot B) + [(A \oplus B) \cdot C_i] . \end{aligned}$$

Procurando-se otimizar algumas implementações, pode-se mostrar ainda que

$$\overline{C_o} = f_{C3}(\overline{A}, \overline{B}, \overline{C_i})$$

e que

$$\overline{S} = f_{S3}(\overline{A}, \overline{B}, \overline{C_i}) .$$

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 8.16 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 .

<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><th>G</th><th>00</th><th>01</th><th>11</th><th>10</th></tr> <tr><th>0</th><td>0</td><td>0</td><td>1</td><td>0</td></tr> <tr><th>1</th><td>0</td><td>0</td><td>1</td><td>0</td></tr> </table>	G	00	01	11	10	0	0	0	1	0	1	0	0	1	0	<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><th>P</th><th>00</th><th>01</th><th>11</th><th>10</th></tr> <tr><th>0</th><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><th>1</th><td>0</td><td>1</td><td>0</td><td>1</td></tr> </table>	P	00	01	11	10	0	0	1	0	1	1	0	1	0	1	<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><th>K</th><th>00</th><th>01</th><th>11</th><th>10</th></tr> <tr><th>0</th><td>1</td><td>0</td><td>0</td><td>0</td></tr> <tr><th>1</th><td>1</td><td>0</td><td>0</td><td>0</td></tr> </table>	K	00	01	11	10	0	1	0	0	0	1	1	0	0	0
G	00	01	11	10																																											
0	0	0	1	0																																											
1	0	0	1	0																																											
P	00	01	11	10																																											
0	0	1	0	1																																											
1	0	1	0	1																																											
K	00	01	11	10																																											
0	1	0	0	0																																											
1	1	0	0	0																																											
(a)	(b)	(c)																																													

Figura 8.16: 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

$$K = f_K(A, B, C_i) = (\bar{A} \cdot \bar{B}) .$$

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 .

8.6 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) Arredondento.
- (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).

Capítulo 9

Circuitos combinacionais básicos

9.1 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.
- Arquitetura da solução
 - Paralela.
 - Serial.

9.2 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.

9.2.1 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.

9.3 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.

9.4 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.

9.5 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.

9.6 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.

9.7 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.

9.8 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 $N_1 = 2^{N_2}$.

9.9 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 $N_1 = 2^{N_2}$.

9.10 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 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.

9.11 Ordenador binário

- O circuito possui N entradas e N saídas.
- A saída apresenta todas as ocorrências de um valor lógico/booleano na entrada, seguidas de todas as ocorrências do valor lógico/booleano complementar.

9.12 Deslocadores (*shifters*)

- O circuito possui N entradas e N saídas.
- 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.

9.13 Somadores em binário puro

- A seguir, são apresentados somadores de 2 operandos, codificados em binário puro.

9.13.1 *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*:

$$C_o = (A \cdot B) .$$

$$S = (\bar{A} \cdot B) + (A \cdot \bar{B}) = (A \oplus B) .$$

9.13.2 *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 são:

- SOP mínima:

$$C_o = (A \cdot B) + (A \cdot C_i) + (B \cdot C_i) .$$

$$S = (\bar{A} \cdot B \cdot \bar{C}_i) + (A \cdot \bar{B} \cdot \bar{C}_i) + (\bar{A} \cdot \bar{B} \cdot C_i) + (A \cdot B \cdot C_i) = (A \oplus B) \oplus C_i .$$

- 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):

$$\bar{C}_o = \overline{(A \cdot B) + (A \cdot C_i) + (B \cdot C_i)} = \overline{(A \cdot B) + [(A + B) \cdot C_i]} .$$

$$\bar{S} = \overline{(A \cdot B \cdot C_i) + [(A + B + C_i) \cdot \bar{C}_o]} = \overline{[(A \cdot B) \cdot C_i] + [(A + B + C_i) \cdot \bar{C}_o]} .$$

- Multiplexador:

$$MUX_1 = (\bar{C}_i \cdot \bar{A}) + (C_i \cdot A) .$$

$$MUX_2 = (B \cdot \overline{MUX_1}) + (A \cdot MUX_1) .$$

$$MUX_3 = (\overline{MUX_2} \cdot \overline{MUX_1}) + (B \cdot MUX_1) .$$

$$\bar{C}_o = \overline{MUX_2} .$$

$$C_o = MUX_2 .$$

$$S = MUX_3 .$$

9.13.3 *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 $\mathbf{Op} = [d_{N-1} d_{N-2} \cdots d_1 d_0]$, 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.

9.13.4 *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

$$P_k^+ = (A_k + B_k) .$$

9.14 Subtratores em binário puro

- A seguir, são apresentados subtratores de 2 operandos, codificados em binário puro.

9.14.1 *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*).

9.14.2 *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*).

9.14.3 *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*.

9.15 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 $\mathbf{Op}_2 = [0\ 0\ \dots\ 0\ 0] = 0$ 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 $\mathbf{Op}_2 = [0\ 0\ \dots\ 0\ 0] = 0$ 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 $\mathbf{Op}_2 = [1\ 1\ \dots\ 1\ 1] = -1$ e $C_{in} = 0$, além de realizar as conseqüentes simplificações no circuito.

9.16 Complementadores

- A seguir, são apresentados dois exemplos de conversores de códigos numéricos.

9.16.1 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.

9.16.2 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

$$(q)_{C_2} = 2^N - |q| = 2^N - 1 + 1 - |q| = 2^N - 1 - (|q| - 1) = (|q| - 1)_{C_1} .$$

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.

9.17 Multiplicadores em binário puro

- A seguir, é apresentada uma possível implementação modular para um multiplicador de 2 operandos, em binário puro.

9.17.1 Multiplicador de 1 *bit*

- É facilmente demonstrável que um multiplicador de 2 operandos de 1 *bit* pode ser implementado por uma porta lógica AND.

9.17.2 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*.

9.18 Comparadores da quantidade de dígitos

- A seguir, são apresentados exemplos de circuitos combinacionais utilizados para a comparação da quantidade de dígitos em operandos.

9.18.1 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”.

9.18.2 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 .

9.19 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.

9.19.1 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:

$$\begin{aligned} I(\mathbf{x}, \mathbf{y}) &= I(x_{N-1}, y_{N-1}) \text{ AND } \cdots I(x_1, y_1) \text{ AND } I(x_0, y_0) \\ &= (x_{N-1} \text{ XNOR } y_{N-1}) \text{ AND } \cdots (x_1 \text{ XNOR } y_1) \text{ AND } (x_0 \text{ XNOR } y_0) . \end{aligned} \quad (9.1)$$

A Equação 9.1 pode ser implementada de forma modular (*bit scanning*), considerando-se que os módulos sejam definidos por

$$I(x_k, y_k) = (x_k \text{ XNOR } y_k) \text{ AND } I(x_{k+1}, y_{k+1}) ,$$

para $(N - 1) \leq k \leq 0$ e $I(x_N, y_N) = 1$.

9.19.2 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, de tal forma que $S_{1_N} = 0$ e $S_{2_N} = 0$.

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:

- $S_1 = (x < y)$ e $S_2 = (x = y)$.
- $S_1 = (x < y)$ e $S_2 = (x > y)$.
- $S_1 = (x = y)$ e $S_2 = (x > y)$.

Em seguida, o sinal S_3 pode gerado por

$$S_3 = (\text{NOT } S_1) \text{ AND } (\text{NOT } S_2) = S_1 \text{ NOR } S_2 .$$

9.19.3 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 III

Circuitos digitais sequenciais

Capítulo 10

Circuitos seqüenciais: conceitos básicos

10.1 Introdução

- Circuitos combinacionais \times 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.

10.2 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 $t \geq t_n$ ”.

10.3 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 booleanas (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 10.1 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 10.1: Tipos de interações entre sinais dos tipos nível e pulso.

10.4 Modelo genérico para circuitos seqüenciais

- Na Figura 10.1 é apresentado um modelo genérico para circuitos seqüenciais, onde:
 - $x_i \in \mathbf{x}$, $i = 1, 2, \dots, L$, são as variáveis de entrada ou variáveis de entrada principais.
 - $z_i \in \mathbf{z}$, $i = 1, 2, \dots, M$, são as variáveis de saída ou variáveis de saída principais.
 - $Y_i \in \mathbf{Y}$, $i = 1, 2, \dots, P$, são as variáveis de excitação ou variáveis de saída secundárias.
 - $y_i \in \mathbf{y}$, $i = 1, 2, \dots, R$, são as variáveis de estado ou variáveis de entrada secundárias.
 - t_n é o instante atual, t_{n-1} é o instante anterior e t_{n+1} é o próximo instante.
 - $z_i^n = f_i(x_1^n, \dots, x_L^n, y_1^n, \dots, y_R^n)$, $i = 1, 2, \dots, M$.
 - $Y_j^n = f_j(x_1^n, \dots, x_L^n, y_1^n, \dots, y_R^n)$, $j = 1, 2, \dots, P$.
 - $y_k^{n+1} = f_k(Y_1^n, \dots, Y_P^n)$, $k = 1, 2, \dots, R$.
- O conjunto das variáveis y_i é denominado estado atual.
- Por sua vez, conjunto das variáveis x_i e y_i é dito estado atual total.
- 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: a partir de alguns Y_i^n deve ser gerado y_j^{n+1} , o qual, então, será retido (armazenado).

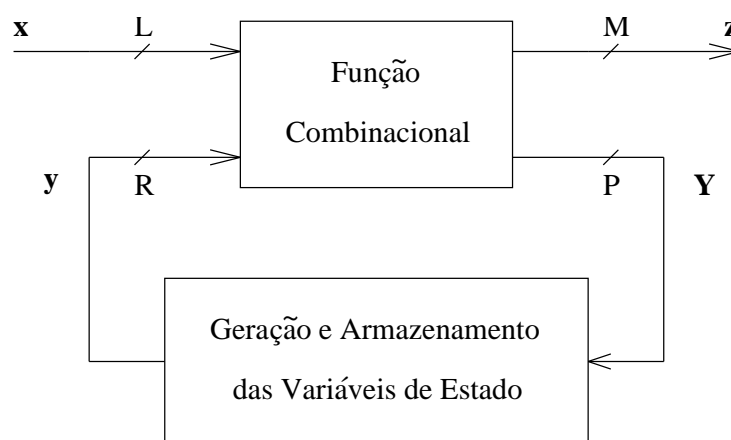


Figura 10.1: Modelo genérico para circuitos seqüenciais.

10.5 Classificação de circuitos seqüenciais quanto à dependência do sinal de saída

- Máquinas (circuitos) de Mealy e de Moore.
- Máquinas de Mealy: $z_i^n = f_i(x_1^n, \dots, x_L^n, y_1^n, \dots, y_R^n)$, $i = 1, 2, \dots, M$.
- Máquinas de Moore: $z_i^n = f_i(y_1^n, \dots, y_R^n)$, $i = 1, 2, \dots, M$.
- As Figuras 10.2 e 10.3 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.

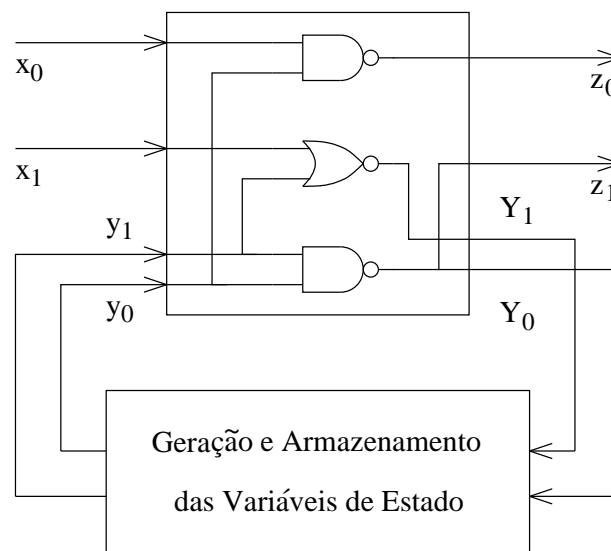


Figura 10.2: Exemplo de máquina de Mealy.

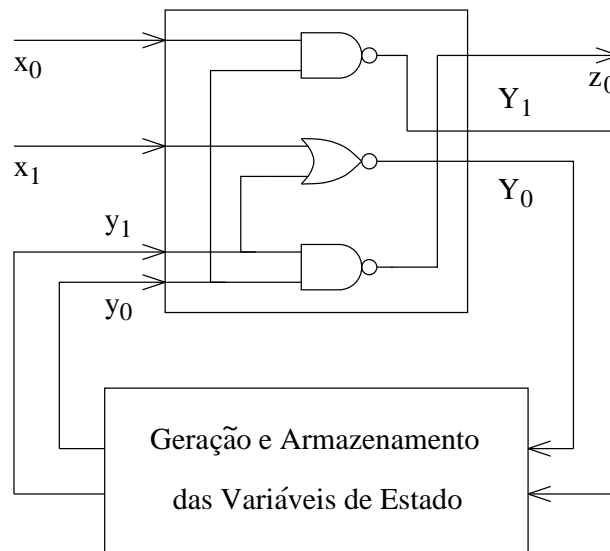


Figura 10.3: Exemplo de máquina de Moore.

10.6 Classificação de circuitos seqüenciais 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*.

10.6.1 Circuitos seqüenciais *clock-mode* ou *clocked*

- A Figura 10.4 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 \mathbf{x}^n , \mathbf{z}^n , e \mathbf{Y}^n estáveis, provoca uma mudança de estado de \mathbf{y}^n para \mathbf{y}^{n+1} .
- O circuito deve estar estável entre dois pulsos de *clock*. Assim, o que limita a frequência 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*.

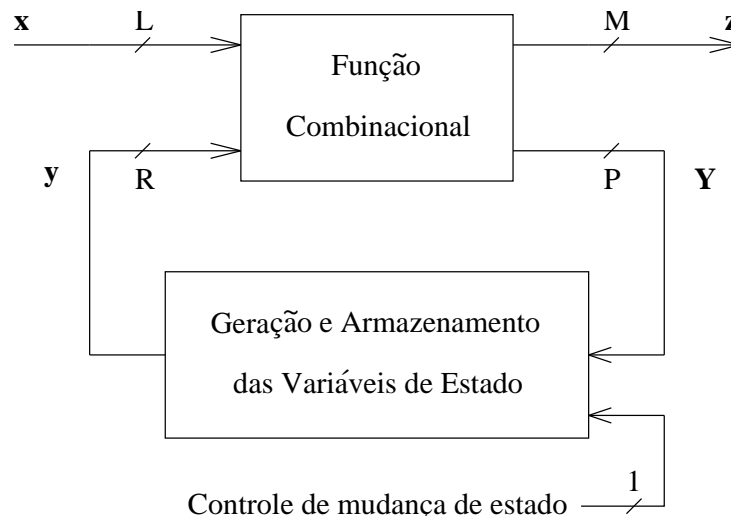


Figura 10.4: Modelo genérico para circuitos seqüenciais *clock-mode*.

10.6.2 Circuitos seqüenciais *pulsed*

- A Figura 10.5 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.

10.6.3 Circuitos seqüenciais *level-mode*

- Na Figura 10.6 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: $y_k^{n+1} = f_k(Y_1^n, \dots, Y_P^n)$, $k = 1, 2, \dots, R$.
- Mais especificamente, neste caso: $P = R$ e $y_k(t + \Delta t_k) = Y_k(t)$, $k = 1, 2, \dots, P$.

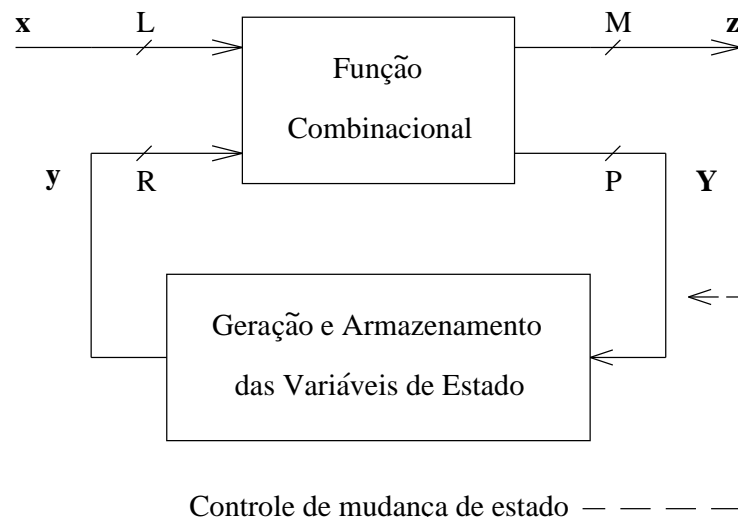


Figura 10.5: Modelo genérico para circuitos seqüenciais *pulsed*.

- 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.

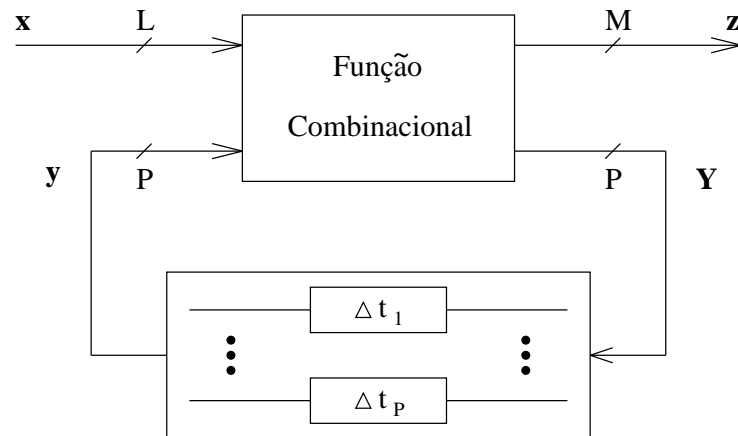


Figura 10.6: Modelo genérico para circuitos seqüenciais *level-mode*.

Capítulo 11

Elementos básicos de armazenamento

11.1 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*.

11.2 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:

$$\begin{cases} Q^{n+1} = (S^n) + (\overline{R^n} \cdot Q^n) & , \text{ para } (S^n \cdot R^n) = 0 \\ \text{Indeterminado} & , \text{ para } S^n = R^n = 1 \end{cases} \quad (11.1)$$

– *Flip-flop* JK:

$$Q^{n+1} = (J^n \cdot \overline{Q^n}) + (\overline{K^n} \cdot Q^n) \quad (11.2)$$

– *Flip-flop* D:

$$Q^{n+1} = D^n \quad (11.3)$$

– *Flip-flop* T_1 :

$$Q^{n+1} = \overline{Q^n} \quad (11.4)$$

– *Flip-flop* T_2 :

$$Q^{n+1} = (T^n \cdot \overline{Q^n}) + (\overline{T^n} \cdot Q^n) \quad (11.5)$$

- As operações básicas, associadas às Equações (11.1), (11.2), (11.3) e (11.5), podem ser mais facilmente identificadas através de suas respectivas tabelas, apresentadas na Figura 11.1.
- 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	$\overline{1}$
1	1	$\overline{Q^n}$

D^n	Q^{n+1}
0	0
1	1

T^n	Q^{n+1}
0	$\overline{Q^n}$
1	$\overline{Q^n}$

Figura 11.1: Tabelas de operação básica para os *flip-flops* SR, JK, D e T_2 .

11.3 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 (11.4) e (11.5), 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 $D = (T \cdot \overline{Q}) + (\overline{T} \cdot Q)$, conforme as Equações (11.4) e (11.5), respectivamente.
- A Tabela 11.1 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
$JK \rightarrow SR$	Não aplicar: $J = K = 1$	—
$JK \rightarrow D$	$J = D ; K = \overline{J}$	—
$JK \rightarrow T_1$	$J = K = 1$	—
$JK \rightarrow T_2$	$J = K = T$	—
$SR \rightarrow JK$	—	$S = (J \cdot \overline{Q}) ; R = (K \cdot Q)$
$D \rightarrow JK$	—	$D = (J \cdot \overline{Q}) + (\overline{K} \cdot Q)$
$T_1 \rightarrow JK$	(*)	(*)
$T_2 \rightarrow JK$	—	$T = (J \cdot \overline{Q}) + (K \cdot Q)$
$SR \rightarrow D$	$S = D ; R = \overline{S}$	—
$SR \rightarrow T_1$	—	$S = \overline{Q} ; R = Q$
$SR \rightarrow T_2$	—	$S = (T \cdot \overline{Q}) ; R = (T \cdot Q)$
$D \rightarrow SR$	—	$D = (S) + (\overline{R} \cdot Q)$
$T_1 \rightarrow SR$	(*)	(*)
$T_2 \rightarrow SR$	—	$T = (S \cdot \overline{Q}) + (R \cdot Q)$
$D \rightarrow T_1$	—	$D = \overline{Q}$
$D \rightarrow T_2$	—	$D = (T \cdot \overline{Q}) + (\overline{T} \cdot Q)$
$T_1 \rightarrow D$	(*)	(*)
$T_2 \rightarrow D$	—	$T = (D \cdot \overline{Q}) + (\overline{D} \cdot Q)$
$T_1 \rightarrow T_2$	(*)	(*)
$T_2 \rightarrow T_1$	$T = 1$	—

Tabela 11.1: Transformações envolvendo *flip-flops* dos tipos JK , D , T_1 e T_2 .

11.4 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 11.2 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 11.2: Mapas de excitação para os *flip-flops* SR, JK, D e T_2 .

11.5 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 11.2.

$Q^n \rightarrow Q^{n+1}$	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 11.2: Definição dos tipos de comportamento apresentados pela saída de um *flip-flop*.

11.6 Excitação \times comportamento

- As tabelas da Figura 11.3 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 .

$Q^n \rightarrow Q^{n+1}$	Variação	S^n	R^n
0 \rightarrow 0	0	0	X
0 \rightarrow 1	α	1	0
1 \rightarrow 0	β	0	1
1 \rightarrow 1	1	X	0
0 \rightarrow X	X	X	X
1 \rightarrow X	X	X	X

$Q^n \rightarrow Q^{n+1}$	Variação	J^n	K^n
0 \rightarrow 0	0	0	X
0 \rightarrow 1	α	1	X
1 \rightarrow 0	β	X	1
1 \rightarrow 1	1	X	0
0 \rightarrow X	X	X	X
1 \rightarrow X	X	X	X

$Q^n \rightarrow Q^{n+1}$	Variação	D^n
0 \rightarrow 0	0	0
0 \rightarrow 1	α	1
1 \rightarrow 0	β	0
1 \rightarrow 1	1	1
0 \rightarrow X	X	X
1 \rightarrow X	X	X

$Q^n \rightarrow Q^{n+1}$	Variação	T^n
0 \rightarrow 0	0	0
0 \rightarrow 1	α	1
1 \rightarrow 0	β	1
1 \rightarrow 1	1	0
0 \rightarrow X	X	X
1 \rightarrow X	X	X

Figura 11.3: Tipos de comportamento e respectivas excitações para os *flip-flops* SR, JK, D e T_2 .

11.7 Funcionalidade \times excitação \times comportamento

- A Tabela 11.3 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 11.3: Tabela resumo de funcionalidade-excitação-comportamento para os *flip-flops* SR, JK, D e T_2 .

11.8 Circuitos seqüenciais \times 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 11.3.
- Como exemplo, a Tabela 11.4 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 excitação para *flip-flops* JK são apresentados na Figuras 11.4 e 11.5, respectivamente. Deve-se notar que este contador não possui variáveis de entrada. Das tabelas das Figuras 11.4 e 11.5, pode-se obter

$$J_2 = K_2 = (Q_1 \cdot Q_0) , \quad (11.6)$$

$$J_1 = K_1 = Q_0 \quad (11.7)$$

e

$$J_0 = K_0 = 1 . \quad (11.8)$$

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 11.4: Tabela de mudanças de estado e de comportamento dos elementos de memória para um contador binário, crescente, de três *bits*.

$FF2$		Q_1Q_0					$FF1$		Q_1Q_0				
		00	01	11	10			00	01	11	10		
Q_2	0	0	0	α	0	Q_2	0	0	α	β	1		
	1	1	1	β	1		1	0	α	β	1		

$FF0$		Q_1Q_0				
		00	01	11	10	
Q_2	0	α	β	β	α	
	1	α	β	β	α	

Figura 11.4: Mapas-K de transição para os elementos de memória de um contador binário, crescente, de três *bits*.

J_2		Q_1Q_0					K_2		Q_1Q_0				
		00	01	11	10			00	01	11	10		
Q_2	0	0	0	1	0	Q_2	0	X	X	X	X		
	1	X	X	X	X		1	0	0	1	0		

J_1		Q_1Q_0					K_1		Q_1Q_0				
		00	01	11	10			00	01	11	10		
Q_2	0	0	1	X	X	Q_2	0	X	X	1	0		
	1	0	1	X	X		1	X	X	1	0		

J_0		Q_1Q_0					K_0		Q_1Q_0				
		00	01	11	10			00	01	11	10		
Q_2	0	1	X	X	1	Q_2	0	X	1	1	X		
	1	1	X	X	1		1	X	1	1	X		

Figura 11.5: Mapas-K de excitação para os *flip-flops* JK de um contador binário, crescente, de três *bits*.

11.9 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 11.6 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 \bar{Q} estarão estáveis (quiescentes) enquanto os inversores estiverem energizados.
- A estrutura da Figura 11.6 apresenta uma grande desvantagem: não é controlável.
- Algumas propostas para tornar o circuito da Figura 11.6 controlável são ilustradas na Figura 11.7.
 - No primeiro caso, Figura 11.7.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 11.7.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 11.7.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 11.7.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.

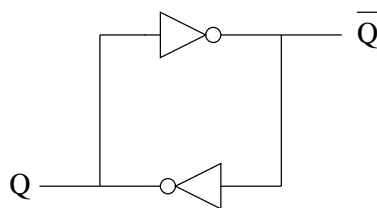


Figura 11.6: Estrutura de armazenamento estática e simétrica, não controlável.

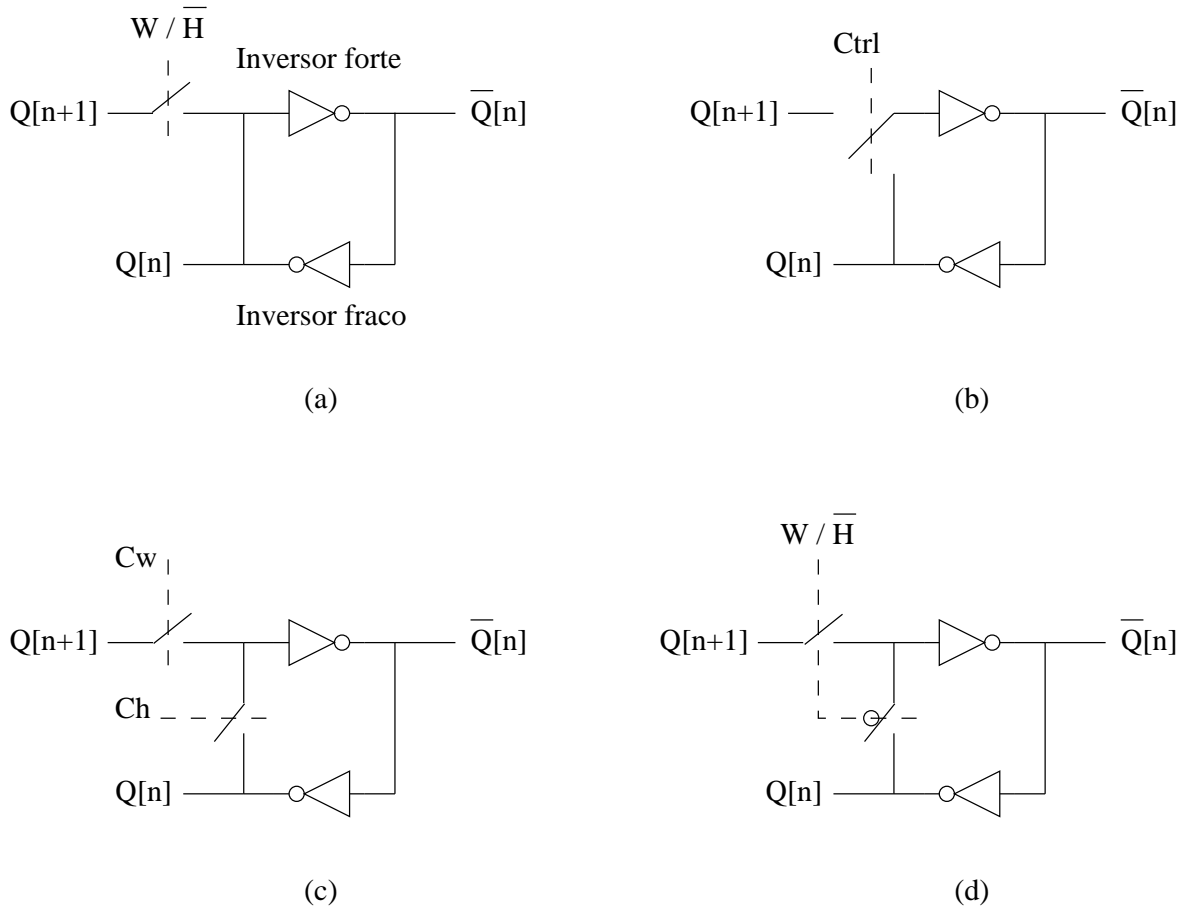


Figura 11.7: Estruturas de armazenamento estáticas e simétricas, controláveis por chaves.

11.10 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).

11.10.1 *Flip-flops* do tipo *unclocked*

- Os *flip-flops* do tipo *unclocked* são também denominados de *latches*.
- O circuito de armazenamento estático da Figura 11.6 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 11.8 e 11.9.

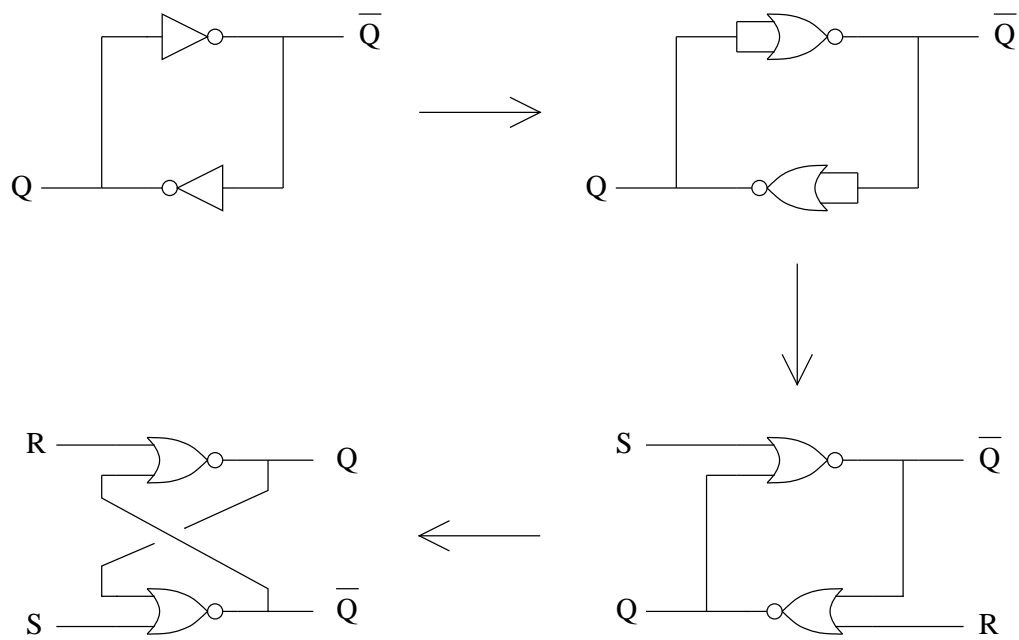


Figura 11.8: Uso de portas lógicas NOR na implementação de controle em uma estrutura de armazenamento estática e simétrica.

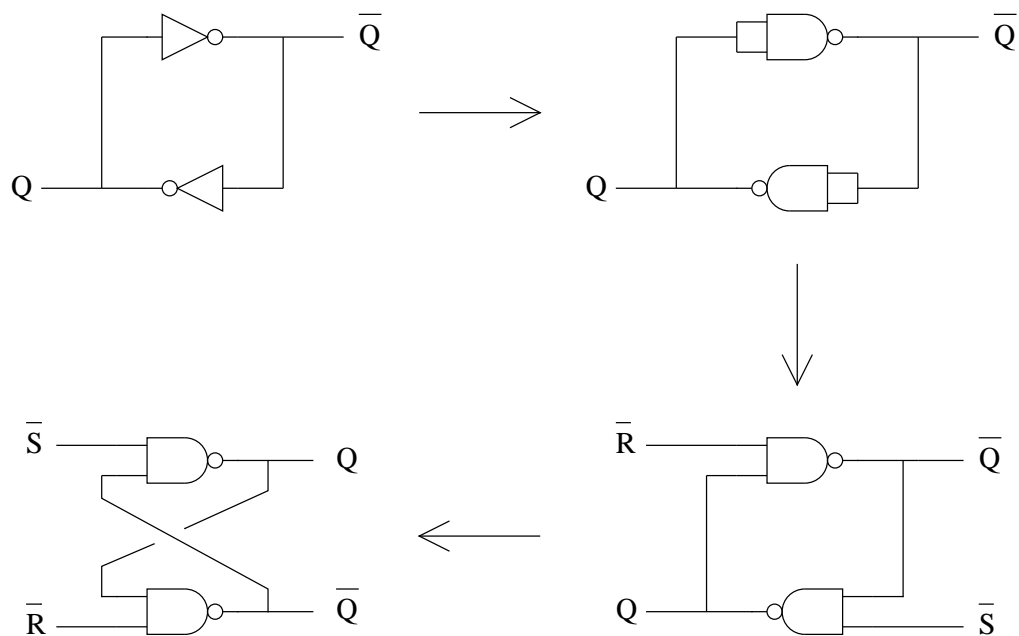


Figura 11.9: Uso de portas lógicas NAND na implementação de controle em uma estrutura de armazenamento estática e simétrica.

- 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 11.5 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 11.5: Operação das estruturas de armazenamento estáticas e simétricas controladas por meio de portas lógicas NOR e NAND.

11.10.2 *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 11.10 e 11.11. Por sua vez, um *flip-flop* D pode ser implementado a partir de um SR, conforme ilustrado na Figura 11.12.

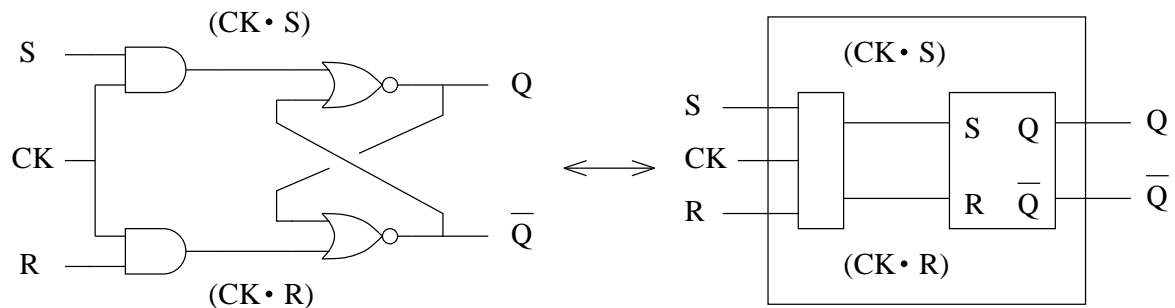


Figura 11.10: Exemplo de implementação de *flip-flop* SR do tipo *clocked* elementar, usando portas lógicas NOR.

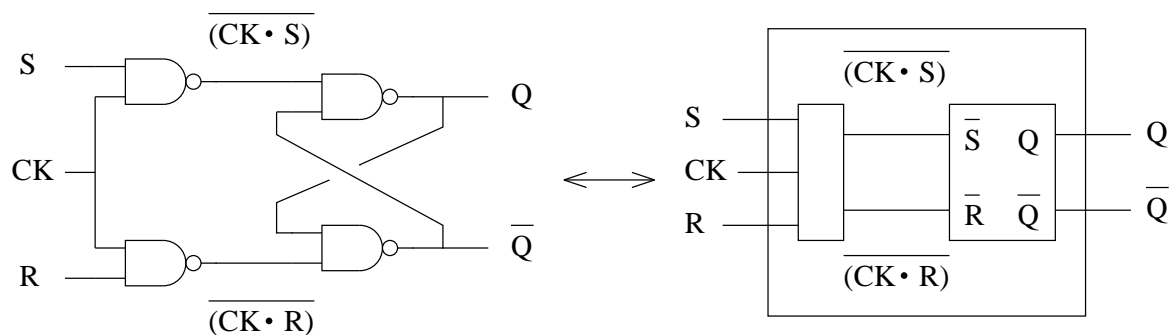


Figura 11.11: Exemplo de implementação de *flip-flop* SR do tipo *clocked* elementar, usando portas lógicas NAND.

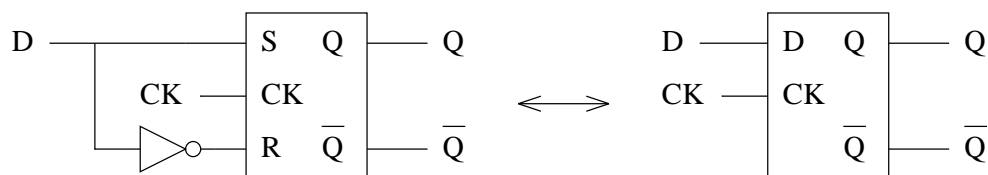


Figura 11.12: Exemplo de implementação de *flip-flop* D do tipo *clocked* elementar, com base em um *flip-flop* SR.

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 11.13. 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 11.14, 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 11.15 e 11.16, 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 11.17 – 11.19.
- Devido a problemas de temporização, o *flip-flop* D da Figura 11.14 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 $J = K = T$.

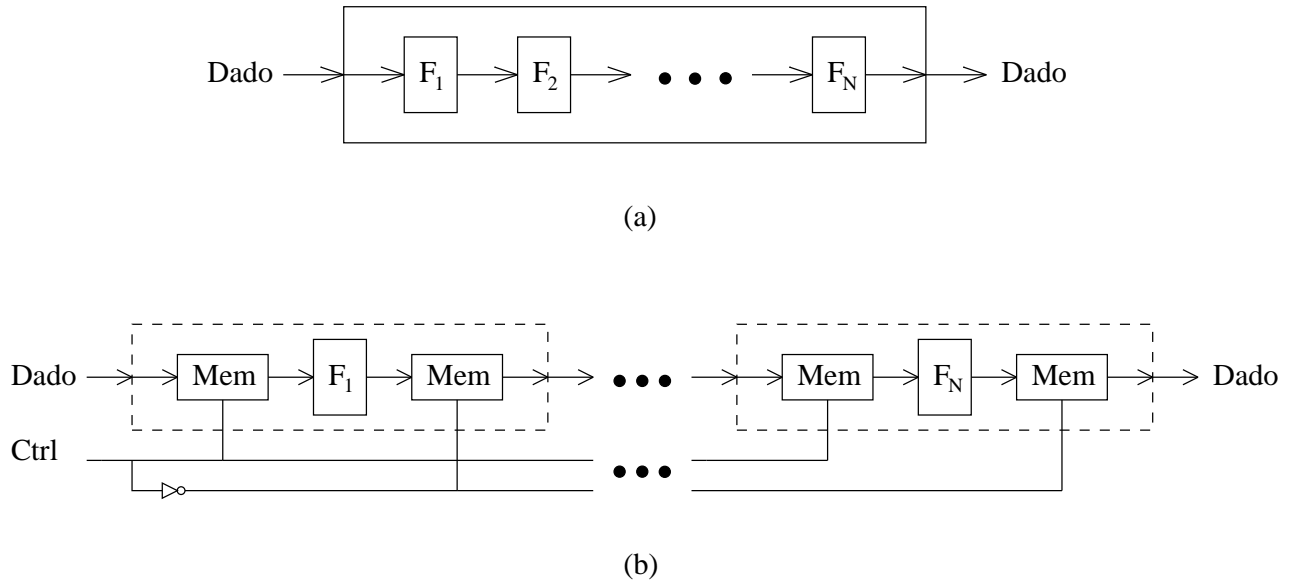


Figura 11.13: Técnica de *pipelining*: (a) Bloco funcional original e (b) Bloco com *pipelining*.

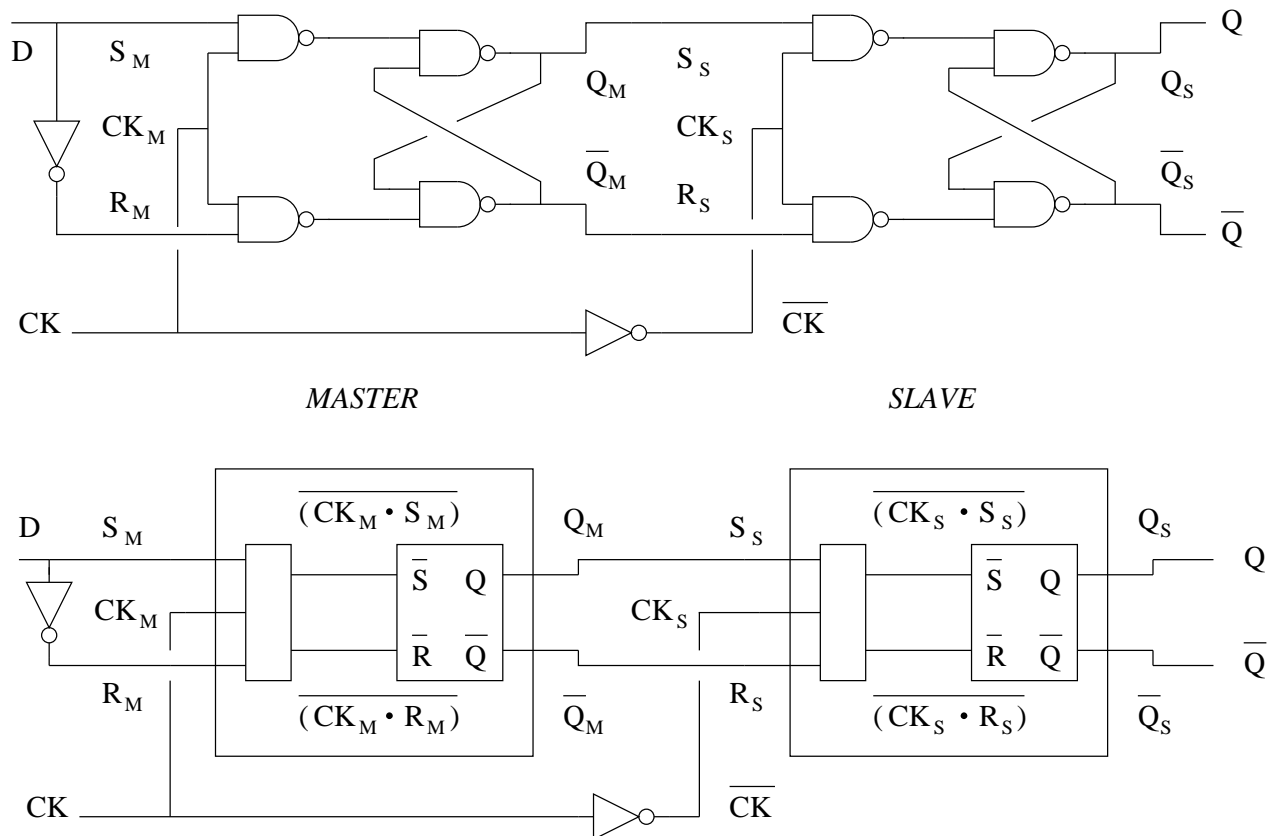


Figura 11.14: Exemplo de implementação de *flip-flop* D do tipo *master-slave*, com base em *flip-flops* SR.

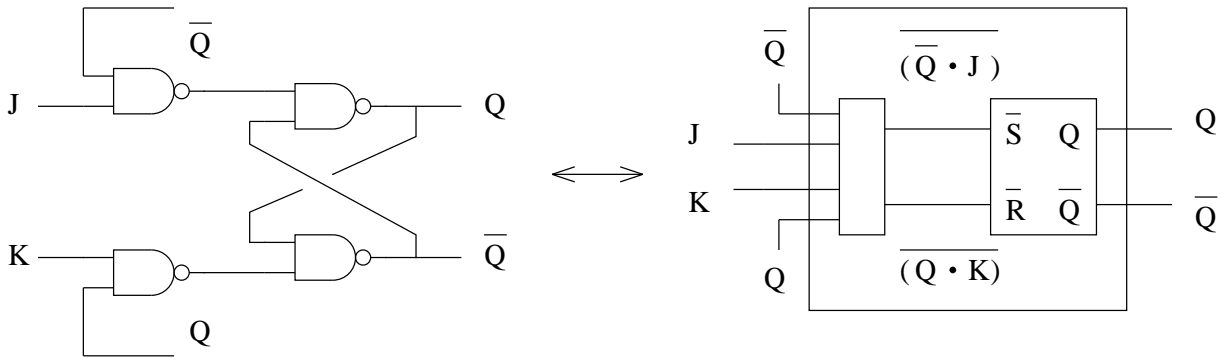


Figura 11.15: Exemplo de implementação de *flip-flop* JK, a partir de *flip-flop* SR *unclocked*, com problema de oscilação.

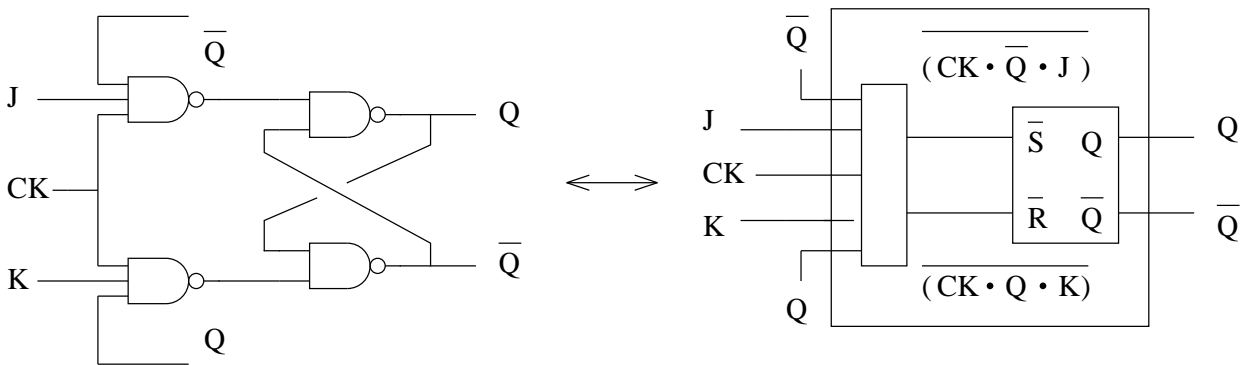


Figura 11.16: Exemplo de implementação de *flip-flop* JK, a partir de *flip-flop* SR *clocked*, com problema de oscilação.

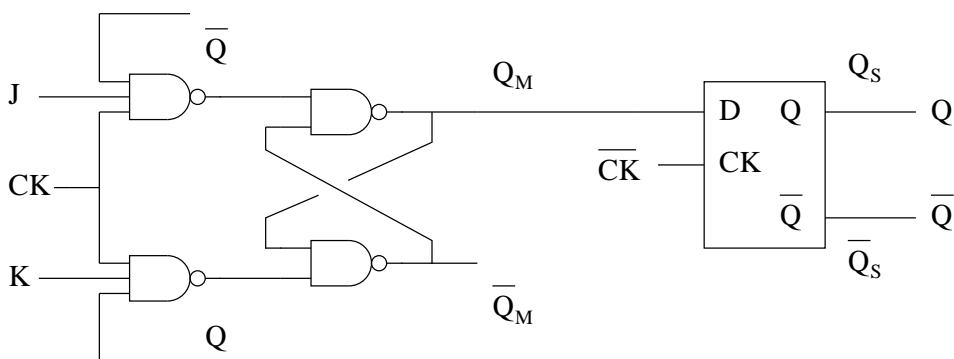


Figura 11.17: Exemplo de implementação de *flip-flop* JK, a partir de *flip-flop* SR *clocked*, sem problema de oscilação, devido ao uso de estrutura *master-slave*.

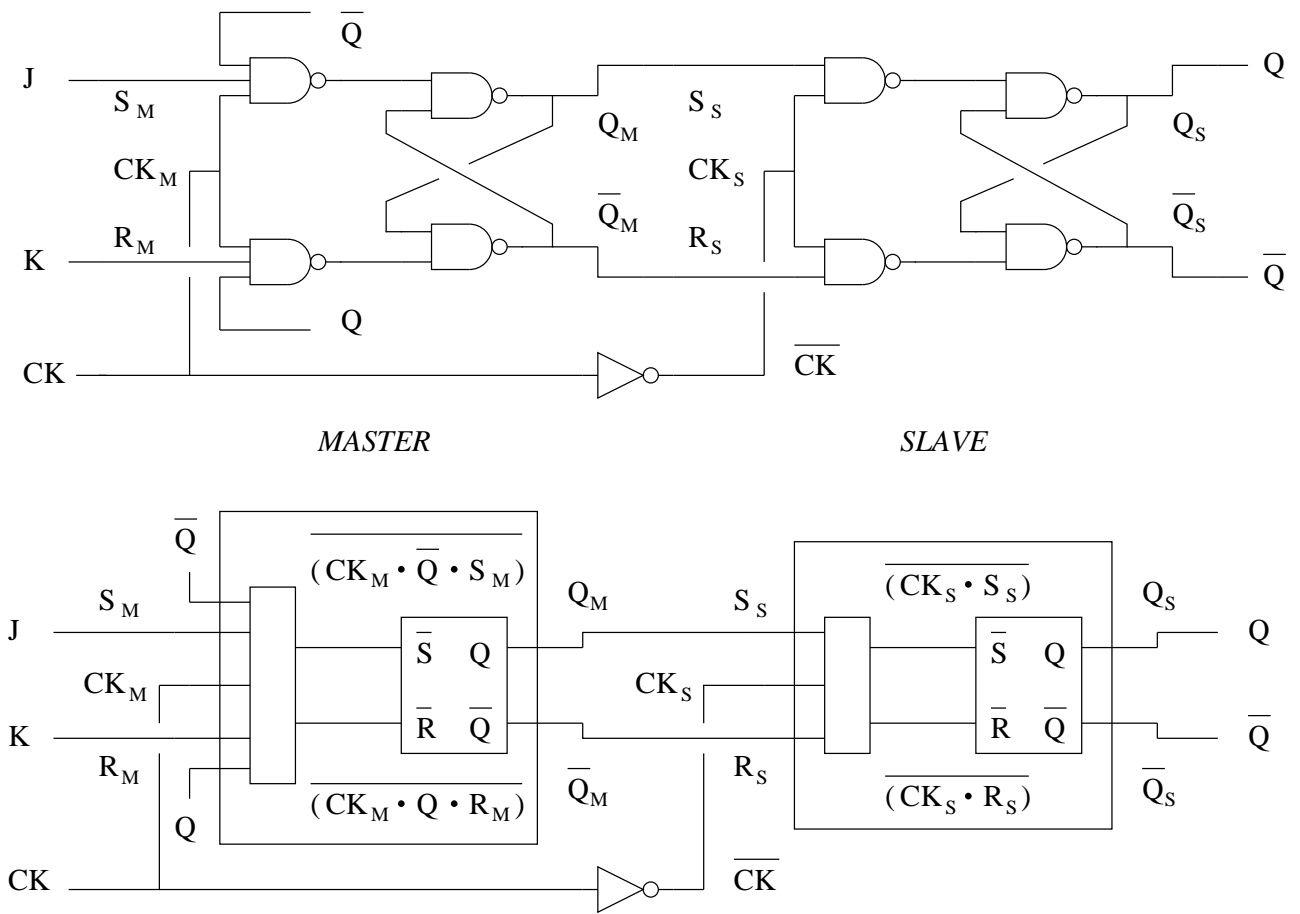


Figura 11.18: Exemplo 1 de implementação de *flip-flop* JK do tipo *master-slave*.

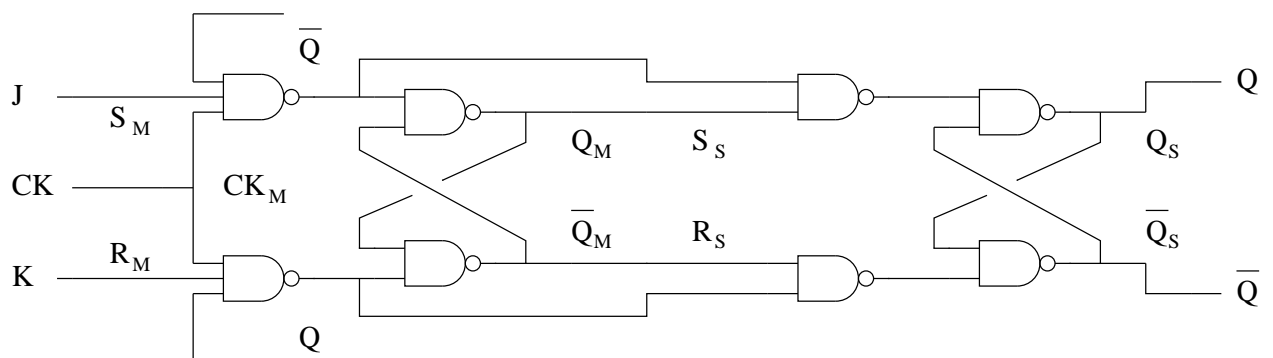


Figura 11.19: Exemplo 2 de implementação de *flip-flop* JK do tipo *master-slave*.

11.11 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).

11.12 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 11.6.

Nomenclatura	Nomes	Tipos aqui definidos
N1	<i>Flip-flop</i>	Todos (os tipos <i>unclocked</i> e <i>clocked</i> elementar são considerados <i>flip-flops</i> elementares)
N2	<i>Latch</i>	<i>unclocked</i> e <i>clocked</i> elementar
	<i>Flip-flop</i>	<i>clocked master-slave</i> e <i>clocked edge-triggered</i>
N3	<i>Latch</i>	<i>unclocked</i> e <i>clocked</i> elementar
	<i>Latch master-slave</i>	<i>clocked master-slave</i>
	<i>Flip-flop</i>	<i>clocked edge-triggered</i>
N4	<i>Latch</i>	<i>unclocked</i>
	<i>Controlled/Clocked-latch</i>	<i>clocked</i> elementar
N5	<i>Positive/Negative-edge flip-flop</i>	<i>clocked master-slave</i>

Tabela 11.6: Diferentes nomenclaturas para *flip-flops*.

Capítulo 12

Circuitos seqüenciais *clock-mode*

12.1 Introdução

- A Figura 12.1 ilustra um modelo genérico para circuitos seqüenciais *clock-mode*.

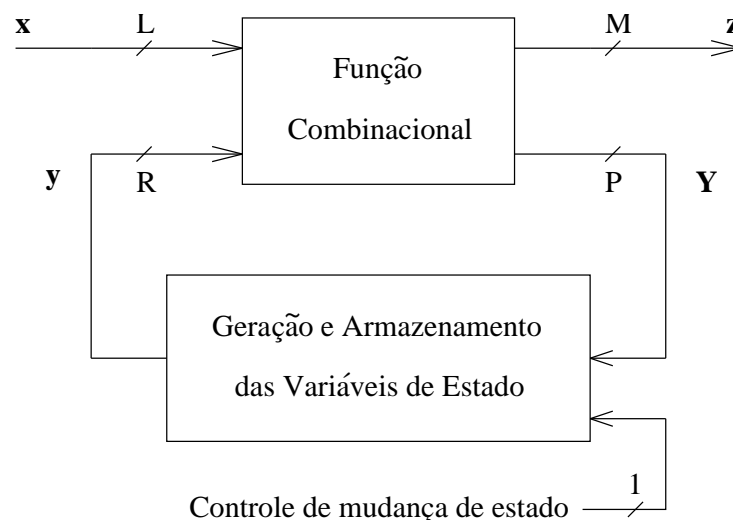


Figura 12.1: Modelo genérico para circuitos seqüenciais *clock-mode*.

- 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 \mathbf{x}^n , \mathbf{z}^n , e \mathbf{Y}^n estáveis, provoca uma mudança de estado de \mathbf{y}^n para \mathbf{y}^{n+1} .
- O circuito deve estar estável entre dois pulsos de *clock*. Logo, cada circuito possuirá uma frequência máxima de operação. Tal frequência 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”.

12.2 Controle de circuitos do tipo *clock-mode*

12.2.1 Características da estrutura *clock-mode*

- $z^n = f_1(\mathbf{y}^n, \mathbf{x}^n)$, para circuitos do tipo Mealy.
- $z^n = f_2(\mathbf{y}^n)$, para circuitos do tipo Moore.
- $\mathbf{Y}^n = f_3(\mathbf{y}^n, \mathbf{x}^n)$.
- $\mathbf{y}^{n+1} = f_4(\mathbf{Y}^n)$.
- Tempos de propagação:
 - Estabilização da entrada \mathbf{x} : Δt_x .
 - Entrada \mathbf{x} para saída \mathbf{z} : Δt_{zx} .
 - Entrada \mathbf{x} para excitação \mathbf{Y} : Δt_{Yx} .
 - Excitação \mathbf{Y} para estado \mathbf{y} : Δt_{yY} .
 - Estado \mathbf{y} para saída \mathbf{z} : Δt_{zy} .
 - Estado \mathbf{y} para excitação \mathbf{Y} : Δt_{Yy} .
 - Tempo máximo de propagação: $\Delta t_{max} = \max\{\Delta \mathbf{t}\} = \max\{\Delta t_1, \Delta t_2, \dots, \Delta t_k\}$.
- Condições de correta operação:
 - Para uma leitura correta dos sinais de saída \mathbf{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 \mathbf{Y} devem estar estáveis no momento do acionamento do bloco.

12.2.2 Controle de circuitos do tipo Moore

- Será assumindo como $\Delta t_x \geq \max\{\Delta \mathbf{t}_x\}$ o intervalo de tempo entre o acionamento do bloco G&A e a estabilização dos sinais de entrada \mathbf{x} .
- Assumindo que as variáveis de estado \mathbf{y} estejam estáveis, as variáveis de excitação \mathbf{Y} estarão estáveis após um tempo $\Delta t_{Yx} \geq \max\{\Delta \mathbf{t}_{Yx}\}$, a partir da estabilização dos sinais de entrada \mathbf{x} .
- Assumindo que todos os sinais estejam estáveis, as variáveis de estado \mathbf{y} estarão estáveis após um tempo $\Delta t_{yY} \geq \max\{\Delta \mathbf{t}_{yY}\}$, a partir do acionamento do bloco G&A.
- As variáveis de saída \mathbf{z} estarão estáveis após um tempo $\Delta t_{zy} \geq \max\{\Delta \mathbf{t}_{zy}\}$, a partir da estabilização dos sinais de estado \mathbf{y} .
- Assumindo que os sinais de entrada \mathbf{x} estejam estáveis, as variáveis de excitação \mathbf{Y} estarão estáveis após um tempo $\Delta t_{Yy} \geq \max\{\Delta \mathbf{t}_{Yy}\}$, a partir da estabilização das variáveis de estado \mathbf{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 \mathbf{x} .
- Logo, para cumprir as condições de correta operação, o período de acionamento do bloco G&A deve ser

$$T_{CTRL} = T_{CK} \geq \max\{(\Delta t_x + \Delta t_{Yx}), (\Delta t_{yY} + \Delta t_{Yy}), (\Delta t_{yY} + \Delta t_{zy})\}. \quad (12.1)$$

- É recomendável que se utilize

$$\Delta t_{yY} < (\Delta t_x + \Delta t_{Yx}) < \max\{(\Delta t_{yY} + \Delta t_{Yy}), (\Delta t_{yY} + \Delta t_{zy})\}. \quad (12.2)$$

12.2.3 Controle de circuitos do tipo Mealy

- Será assumindo como $\Delta t_x \geq \max\{\Delta t_x\}$ o intervalo de tempo entre o acionamento do bloco G&A e a estabilização dos sinais de entrada \mathbf{x} .
- Assumindo que as variáveis de estado \mathbf{y} estejam estáveis, as variáveis de saída \mathbf{z} estarão estáveis após um tempo $\Delta t_{zx} \geq \max\{\Delta t_{zx}\}$, a partir da estabilização dos sinais de entrada \mathbf{x} .
- Assumindo que as variáveis de estado \mathbf{y} estejam estáveis, as variáveis de excitação \mathbf{Y} estarão estáveis após um tempo $\Delta t_{Yx} \geq \max\{\Delta t_{Yx}\}$, a partir da estabilização dos sinais de entrada \mathbf{x} .
- Assumindo que todos os sinais estejam estáveis, as variáveis de estado \mathbf{y} estarão estáveis após um tempo $\Delta t_{yY} \geq \max\{\Delta t_{yY}\}$, a partir do acionamento do bloco G&A.
- Assumindo que os sinais de entrada \mathbf{x} estejam estáveis, as variáveis de saída \mathbf{z} estarão estáveis após um tempo $\Delta t_{zy} \geq \max\{\Delta t_{zy}\}$, a partir da estabilização dos sinais de estado \mathbf{y} .
- Assumindo que os sinais de entrada \mathbf{x} estejam estáveis, as variáveis de excitação \mathbf{Y} estarão estáveis após um tempo $\Delta t_{Yy} \geq \max\{\Delta t_{Yy}\}$, a partir da estabilização das variáveis de estado \mathbf{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

$$T_{CTRL} = T_{CK} \geq \max\{(\Delta t_x + \Delta t_{zx}), (\Delta t_x + \Delta t_{Yx}), (\Delta t_{yY} + \Delta t_{zy}), (\Delta t_{yY} + \Delta t_{Yy})\}. \quad (12.3)$$

- Nesse caso, é recomendável que se utilize

$$\Delta t_{yY} < (\Delta t_x + \Delta t_{Yx}) < \max\{(\Delta t_{yY} + \Delta t_{zy}), (\Delta t_{yY} + \Delta t_{Yy})\} \quad (12.4)$$

e

$$(\Delta t_x + \Delta t_{zx}) < \max\{(\Delta t_{yY} + \Delta t_{zy}), (\Delta t_{yY} + \Delta t_{Yy})\}. \quad (12.5)$$

12.3 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.

12.4 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.

12.5 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 ($z_i = y_i, i = 1, 2, \dots, K$).

- Circuito com memória finita
 - A Figura 12.2 apresenta um circuito com memória finita.
 - Os blocos de retardo unitário \mathbf{D} são conjuntos de *flip-flops* do tipo D .
 - Os vetores \mathbf{x}^{n-r} , $r = 0, 1, \dots, R$, e \mathbf{z}^{n-s} , $s = 0, 1, \dots, S$, representam os sinais de entrada x_i^{n-r} , $i = 1, 2, \dots, L$, e de saída z_j^{n-s} , $j = 1, 2, \dots, M$, respectivamente.
 - Neste tipo de circuito: $\mathbf{z}^n = f(\mathbf{x}^n, \mathbf{x}^{n-1}, \dots, \mathbf{x}^{n-R}, \mathbf{z}^{n-1}, \dots, \mathbf{z}^{n-S})$.
 - 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 12.3 e 12.4 destacam, respectivamente, dois casos particulares:
 - * Circuitos com memória de entrada finita: $\mathbf{z}^n = f(\mathbf{x}^n, \mathbf{x}^{n-1}, \dots, \mathbf{x}^{n-R})$.
 - * Circuitos com memória de saída finita: $\mathbf{z}^n = g(\mathbf{z}^{n-1}, \dots, \mathbf{z}^{n-S})$.

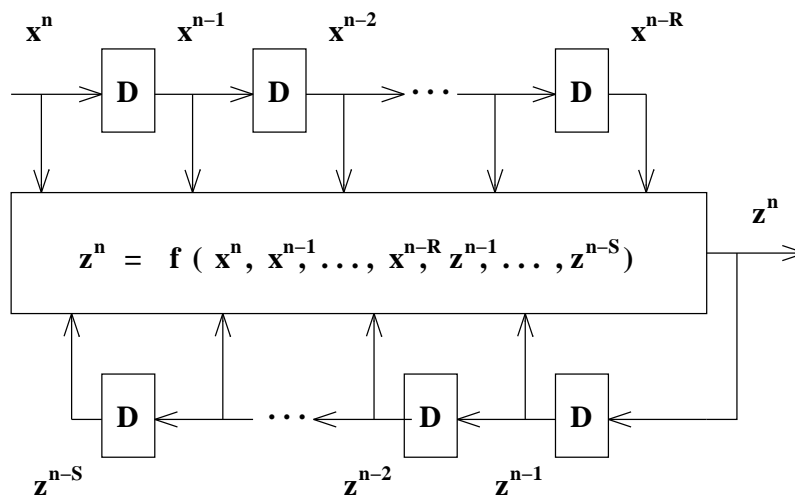


Figura 12.2: Modelo genérico para circuitos com memória finita.

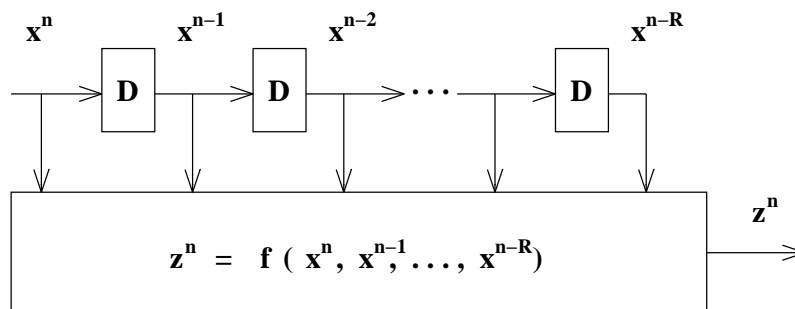


Figura 12.3: Modelo genérico para circuitos com memória de entrada finita.

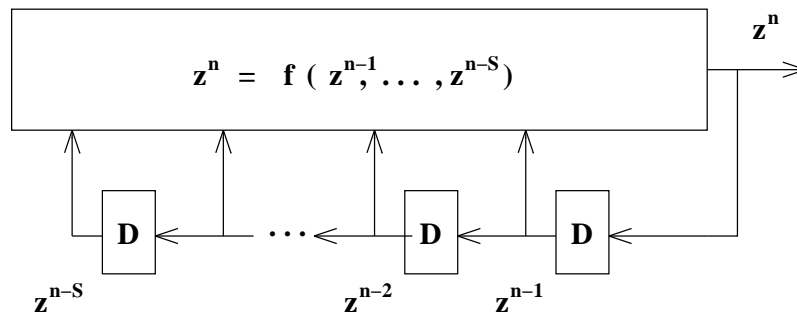


Figura 12.4: Modelo genérico para circuitos com memória de saída finita.

12.6 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.

12.6.1 Etapas de análise

- Passos principais:

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.

12.6.2 Exemplos de análise

- Circuito com memória finita.
- 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 ($z_i = y_i, i = 1, 2, \dots, K$).
- Circuito com memória não finita genérico.

12.7 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.

12.7.1 Opções de projeto e suas características

- 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 ($z_i = y_i, i = 1, 2, \dots, K$): 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 12.5 apresenta os fluxos de projeto para cada uma das três opções.

12.7.2 Etapas de projeto de circuitos seqüenciais

- Os três tipos de projeto abordados 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.
- Etapas gerais de projeto:
 - P1** - Problema a ser resolvido.
 - P2** - Descrição funcional do problema (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, onde 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: Tabela de transição de estados reduzida (*minimal-state table*), baseada em técnicas de minimização de estados.

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 é uma característica 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 de atribuição.

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 - Escolha dos elementos de memória.

P9 - Equações de entrada dos elementos de memória (variáveis de excitação), baseadas na tabela de transição de estados (*transition table*) e nas tabelas de excitação dos elementos de memória (*excitation table/map* ou *transition list/table/map*).

P10 - Circuito proposto.

P11 - Análise do circuito para verificação de comportamento dos estados não utilizados e não especificados, caso existam.

12.7.3 Exemplos de projeto de circuitos seqüenciais

- Circuito com memória finita de entrada.
- Circuito com memória finita de saída.
- 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 ($z_i = y_i, i = 1, 2, \dots, K$).
- Circuito com memória não finita genérico.
- Relacionamento dos três tipos de projeto.

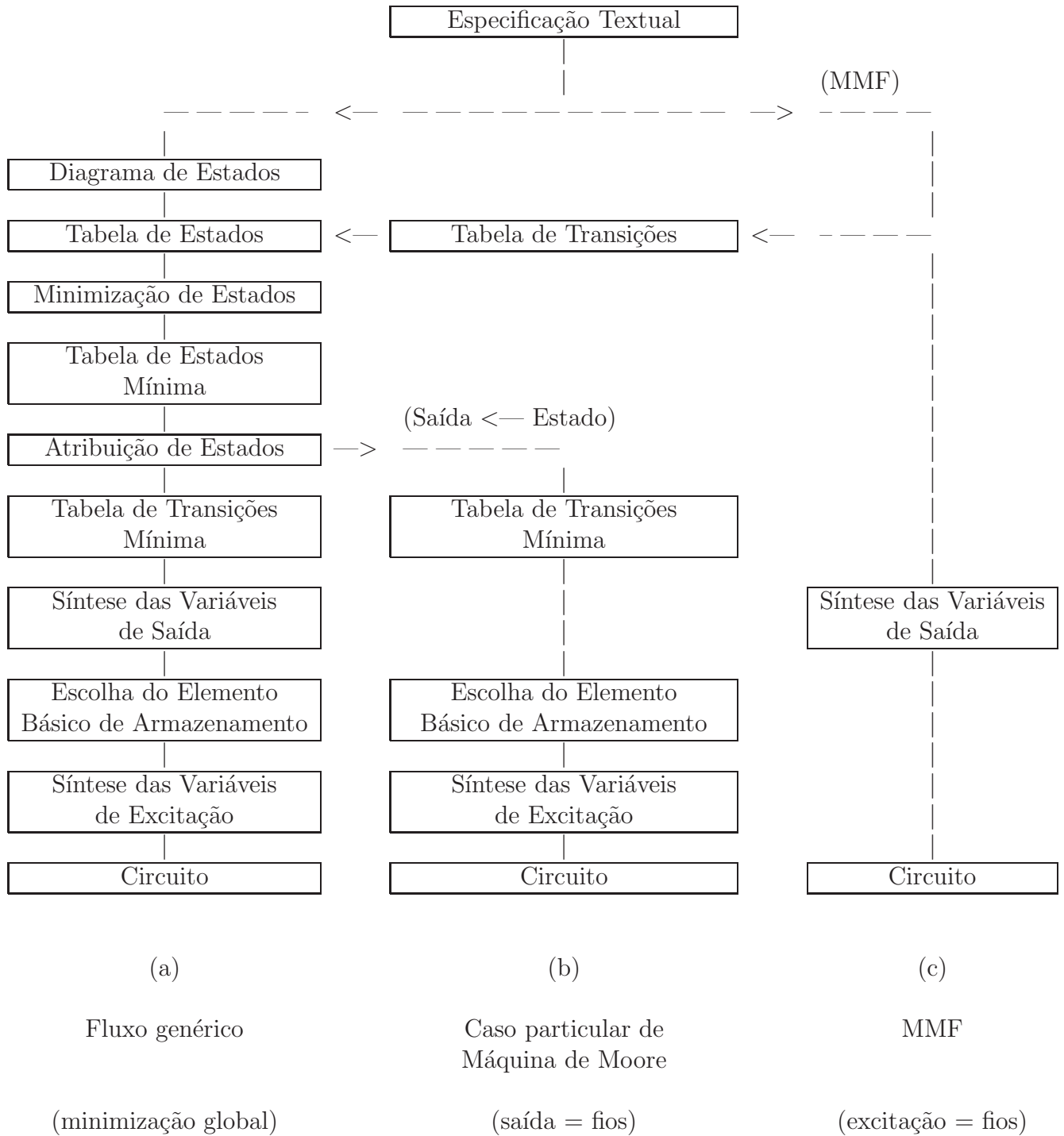


Figura 12.5: Fluxos de projeto para circuitos seqüenciais *clock-mode*: (a) Fluxo genérico, (b) Caso particular de Máquina de Moore e (c) Máquina de Memória Finita (MMF).

12.8 Minimização de estados

12.8.1 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. Conseqüentemente, a tabela de transição de estados (*state table*) é simplificada e, possivelmente, o circuito lógico minimizado.
- Uma formalismo teórico é apresentado no Apêndice C.

12.8.2 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.
- Condição de redundância: 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.
- Algoritmo de eliminação de estados redundantes por simples inspeção:

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 12.6 apresenta um exemplo de eliminação de estados redundantes por simples inspeção.

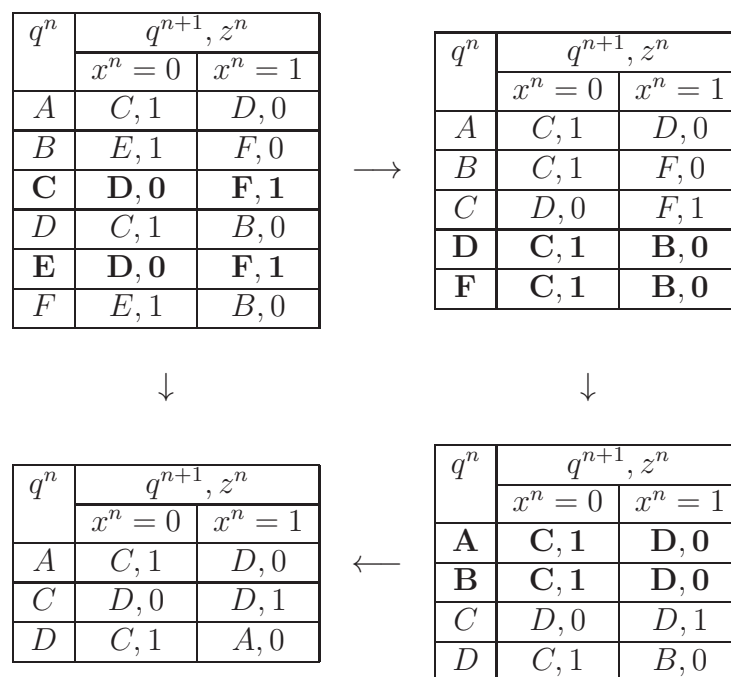


Figura 12.6: Eliminação de estados redundantes através da inspeção da tabela de estados.

12.8.3 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 C 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 C, 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 12.7, 12.8 e 12.9 ilustram o processo para diferentes tabelas de estado.

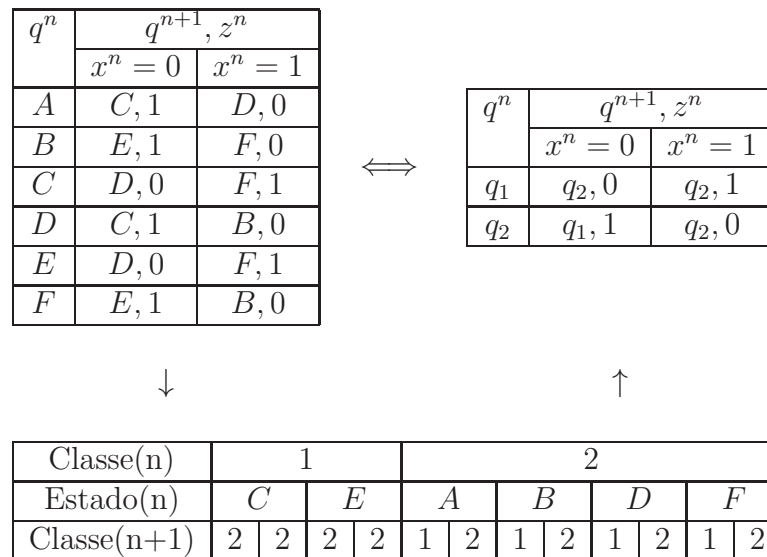


Figura 12.7: Exemplo de minimização positiva em um passo.

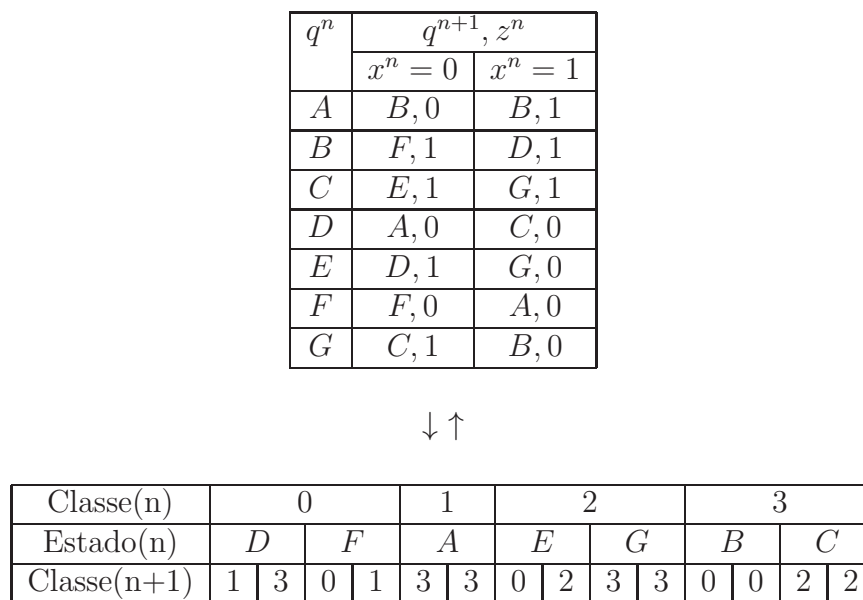


Figura 12.8: Exemplo de minimização negativa em um passo.

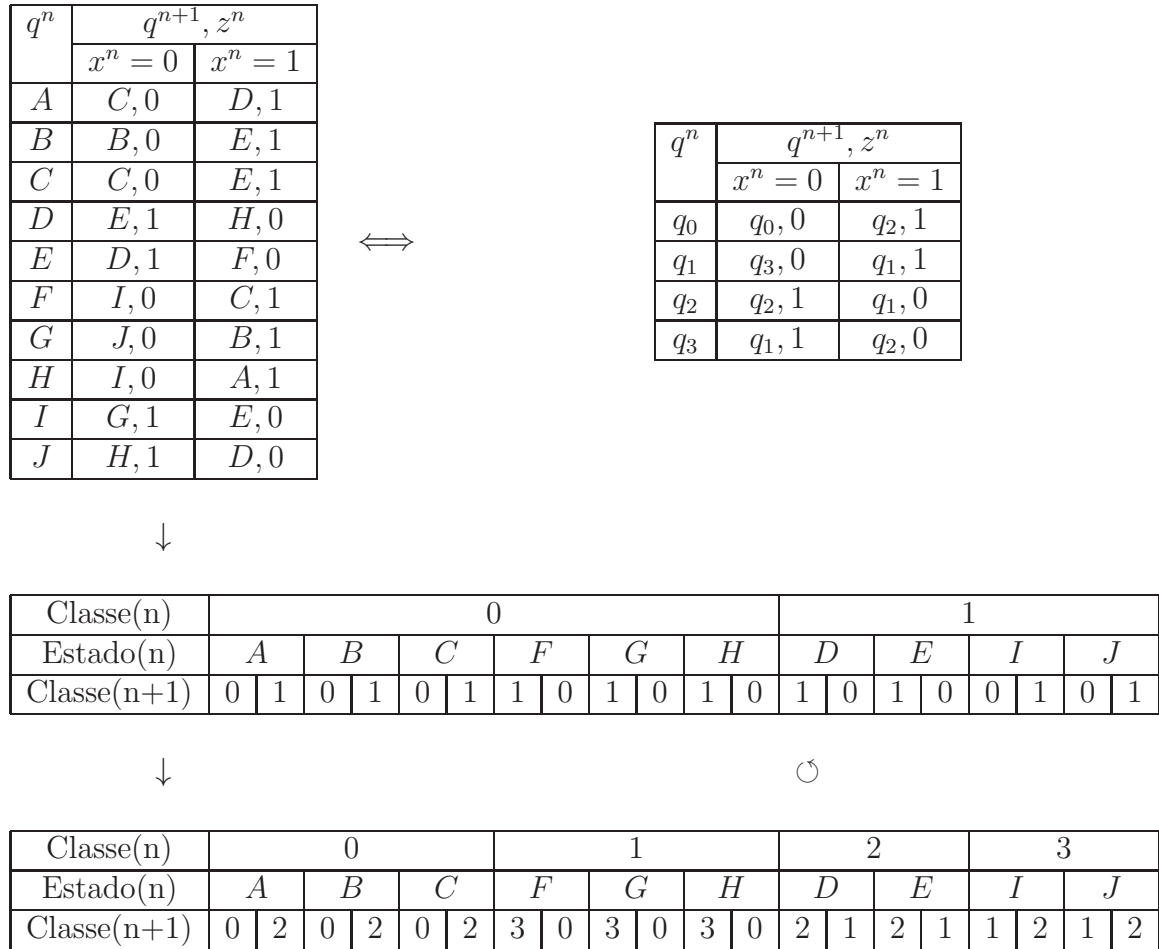


Figura 12.9: Exemplo de minimização positiva em mais de um passo.

12.8.4 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, todas 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 12.10 e 12.11.

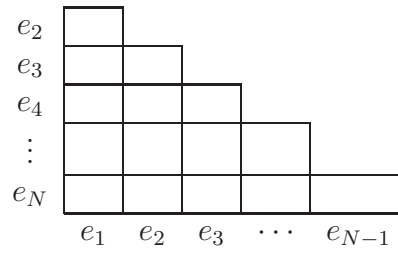


Figura 12.10: Tabela de implicação genérica do método de Paul-Unger.

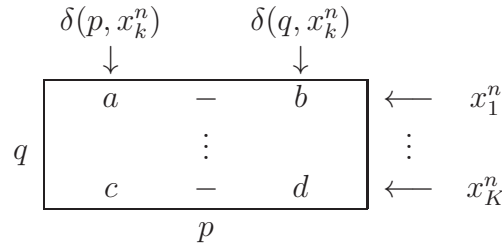


Figura 12.11: Célula genérica da tabela do método de Paul-Unger.

- 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 $(h_i^{n+1} - v_j^{n+1})_k$, 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 $(h_i^{n+1} - v_j^{n+1})_k$ 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.

12.9 Atribuição de estados

12.9.1 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 $A_{dif} = \frac{(2^V-1)!}{(2^V-S)! \cdot V!}$.
- A Tabela 12.1 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 12.1: 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.

12.9.2 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 12.12, 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.

		y_1y_0			
		00	01	11	10
x_1x_0	00	v_r	v_r		
	01				
	11			v_c	
	10			v_c	

Figura 12.12: 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 12.12, 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 12.13 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 12.14 e 12.15 ilustram a análise de minimização para as variáveis de excitação.
- A Figura 12.14 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 12.15 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 deve ser prioritária em relação à segunda.

Estados	Variáveis de Estado
q	$y_1 y_0$
<i>a</i>	00
<i>b</i>	01
<i>d</i>	11
<i>c</i>	10

Figura 12.13: 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}		$y_1^{n+1} y_0^{n+1}$	
		$x^n = 0$	$x^n = 1$	$x^n = 0$	$x^n = 1$
...
b	01	a	...	00	...
c	10	a	...	00	...
...

Tabela de transição de estados

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0		E_{00}		E_{10}
	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_1 Dinâmica da variável de estado y_0

a) Caso sem simplificação.

q^n	$y_1^n y_0^n$	q^{n+1}		$y_1^{n+1} y_0^{n+1}$	
		$x^n = 0$	$x^n = 1$	$x^n = 0$	$x^n = 1$
...
b	01	a	...	00	...
d	11	a	...	00	...
...

Tabela de transição de estados

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0		E_{00}	E_{10}	
	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_1 Dinâmica da variável de estado y_0

b) Caso com simplificação.

Figura 12.14: 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}		$y_1^{n+1} y_0^{n+1}$	
		$x^n = 0$	$x^n = 1$	$x^n = 0$	$x^n = 1$
...
a	00	b	c	01	10
...
...

Tabela de transição de estados

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0	E_{00}			
	1	E_{01}			

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0	E_{01}			
	1	E_{00}			

Dinâmica da variável de estado y_1 Dinâmica da variável de estado y_0

a) Caso sem simplificação.

q^n	$y_1^n y_0^n$	q^{n+1}		$y_1^{n+1} y_0^{n+1}$	
		$x^n = 0$	$x^n = 1$	$x^n = 0$	$x^n = 1$
...
a	00	b	d	01	11
...
...

Tabela de transição de estados

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0	E_{00}			
	1	E_{01}			

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0	E_{01}			
	1	E_{01}			

Dinâmica da variável de estado y_1 Dinâmica da variável de estado y_0

b) Caso com simplificação.

Figura 12.15: 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 12.12, destaca-se uma situação estática, não envolvendo mudanças de estado.
- Com base na atribuição de estados apresentada na Figura 12.13, a Figura 12.16 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	
		$x^n = 0$	$x^n = 1$
...
b	01	1	0
c	10	1	0
...

Tabela de transição de estados

		$y_1^n y_0^n$			
		00	01	11	10
x^n	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	
		$x^n = 0$	$x^n = 1$
...
b	01	1	0
d	11	1	0
...

Tabela de transição de estados

		$y_1^n y_0^n$			
		00	01	11	10
x^n	0		1	1	
	1		0	0	

z_i

Mapa-K da saída z_i

b) Caso com simplificação.

Figura 12.16: Análise de minimização para as equações de saída.

12.9.3 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 12.17 ilustra as regras descritas acima.

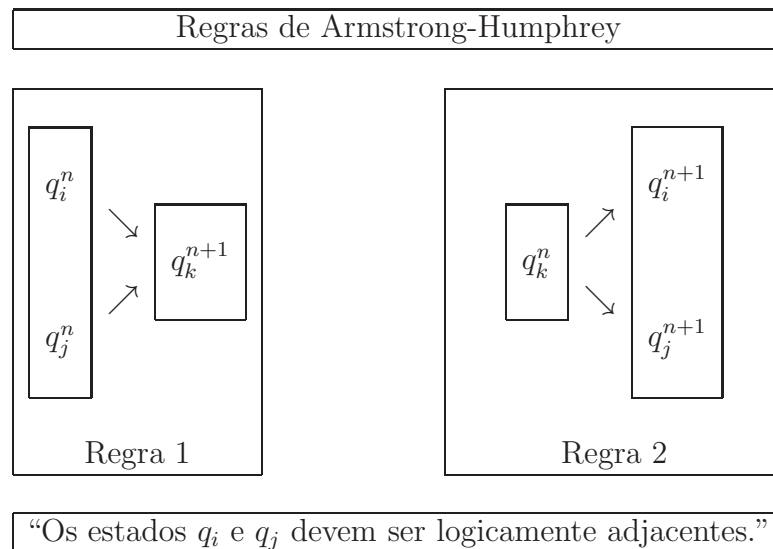


Figura 12.17: Ilustração das regras de Armstrong-Humphrey.

12.9.4 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 dos 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.

12.10 Efeitos causados por estados extras

12.10.1 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*).

12.10.2 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 13

Circuitos seqüenciais *pulsed*

13.1 Introdução

- A Figura 13.1 apresenta um modelo genérico para circuitos seqüenciais *pulsed*.

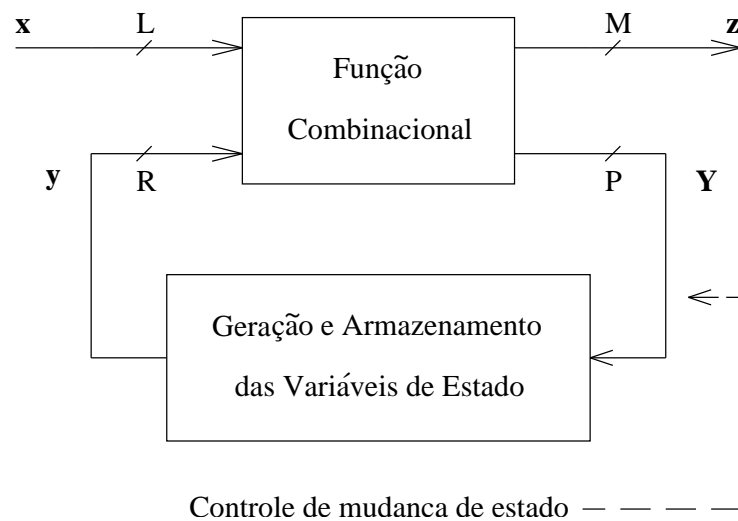


Figura 13.1: Modelo genérico para circuitos seqüenciais *pulsed*.

- 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 \bar{x}_p representam, respectivamente, sinais dos tipos nível, pulso positivo e pulso negativo, as seguintes combinações podem ser empregadas:

$$* \left\{ \begin{array}{l} x_l \cdot x_l = x_l \quad , \quad x_l + x_l = x_l \quad , \quad x_l \cdot x_p = x_p \quad , \quad x_p + x_p = x_p \\ \text{ou} \\ x_l \cdot x_l = x_l \quad , \quad x_l + x_l = x_l \quad , \quad x_l + \bar{x}_p = \bar{x}_p \quad , \quad \bar{x}_p \cdot \bar{x}_p = \bar{x}_p \end{array} \right.$$

13.2 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.

13.3 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 uma tentativa de otimização que pode levar à redução da quantidade de *hardware* em detrimento da frequência 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.

13.4 Circuitos *pulse-mode*

13.4.1 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.

13.4.2 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 utilizadas 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 \bar{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 \bar{z}_p (pulso negativo).
- Na tabela de estados, os valores $z_p = \bar{z}_p = 0$ e $z_p = \bar{z}_p = 1$, de saídas pulsantes z_p (pulso positivo) e \bar{z}_p (pulso negativo), representam, respectivamente, a ausência e a presença de um pulso em z_p e \bar{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}, \dots) , 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 consequência 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 13.2.
- 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 13.3 apresenta exemplos de tabelas de estados para circuitos *pulse-mode* Mealy e Moore. A tabela da Figura 13.3.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 13.3.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.

		$x_{p1}x_{p2}$			
		00	01	11	10
y_1y_2	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

		$x_{p1}x_{p2}$			
		00	01	11	10
y_1y_2	00	0	0/ Π /X	—	0/ Π /X
	01	0	0/ Π /X	—	0/ Π /X
	11	0	0/ Π /X	—	0/ Π /X
	10	0	0/ Π /X	—	0/ Π /X

Figura 13.2: 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

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

a) Circuito do tipo Mealy. b) Circuito do tipo Moore.

Figura 13.3: Tabelas de estados para circuitos *pulse-mode* Mealy e Moore.

13.4.3 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”.

		$x_{p1}x_{p2}$			
		00	01	11	10
y_1y_2	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)

		x_{p1}	x_{p2}
		y_1y_2	00
01	0/Π		0/Π
11	0/Π		0/Π
10	0/Π		0/Π

(b)

Figura 13.4: Mapas de Karnaugh para síntese de variáveis pulsadas, considerando-se duas entradas pulsadas: (a) Mapa completo e (b) Mapa simplificado.

		$x_{p1}x_{p2}x_{p3}$							
		000	001	011	010	100	101	111	110
y_1y_2	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)

		x_{p1}	x_{p2}	x_{p3}
		y_1y_2	00	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 13.5: Mapas de Karnaugh para síntese de variáveis pulsadas, considerando-se três entradas pulsadas: (a) Mapa completo e (b) Mapa simplificado.

13.5 Circuitos *ripple-clock*

13.5.1 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.

13.5.2 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.

13.5.3 Desvantagens

- As desvantagens deste tipo de acionamento são: i) o aumento do tempo de estabilização nas mudanças de estado, o que é equivalente à redução da frequência 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.

13.5.4 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 encontra 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 das 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.

13.5.5 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.

13.6 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 13.6 e 13.7 apresentam exemplos de controle de sinal de *clock*.

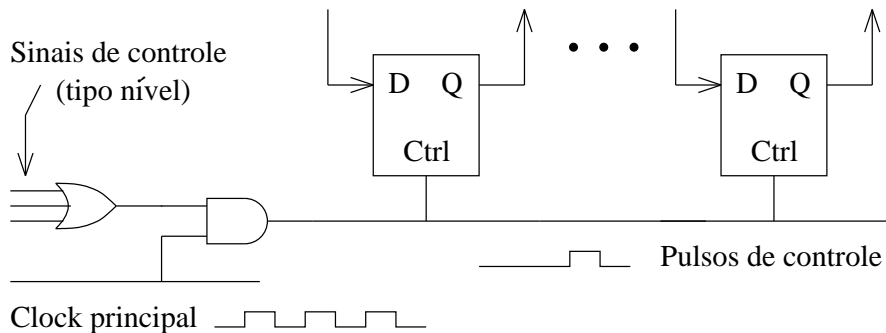


Figura 13.6: Exemplo 1 de controle de sinal de *clock*.

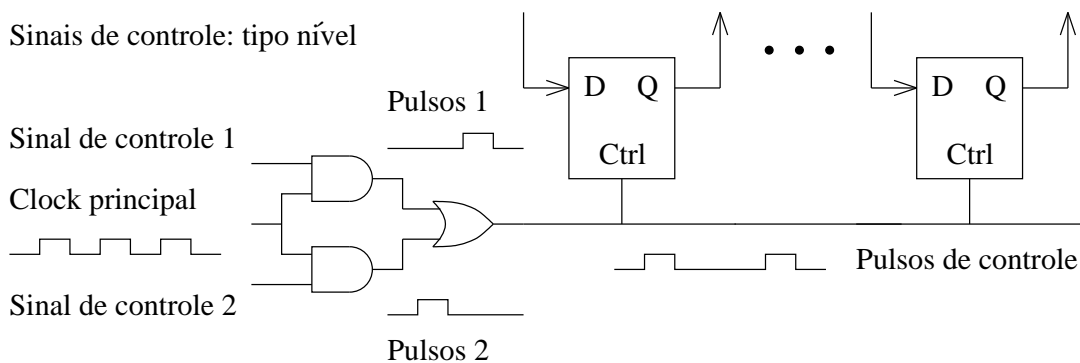


Figura 13.7: Exemplo 2 de controle de sinal de *clock*.

- 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 13.8, 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.

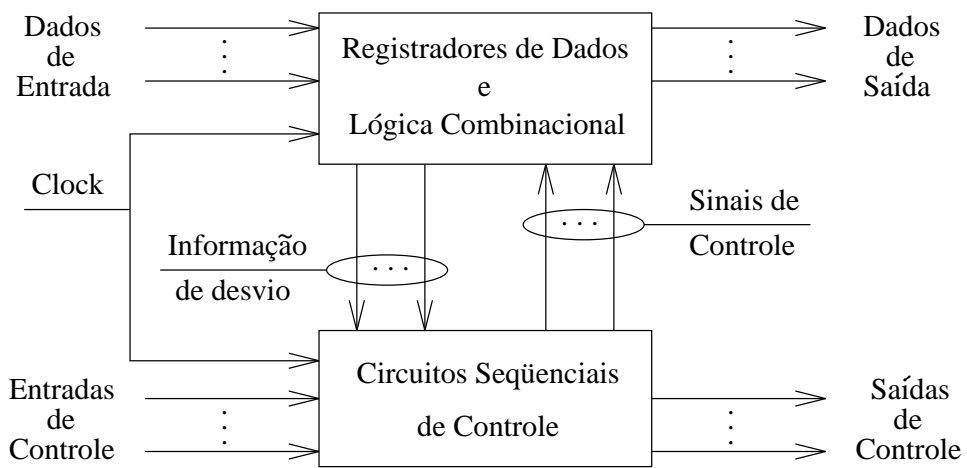


Figura 13.8: Modelo genérico para circuitos seqüenciais *controlled-clock*.

Capítulo 14

Circuitos seqüenciais *level-mode*

14.1 Introdução

- A Figura 14.1 apresenta um modelo genérico para circuitos seqüenciais *level-mode*.

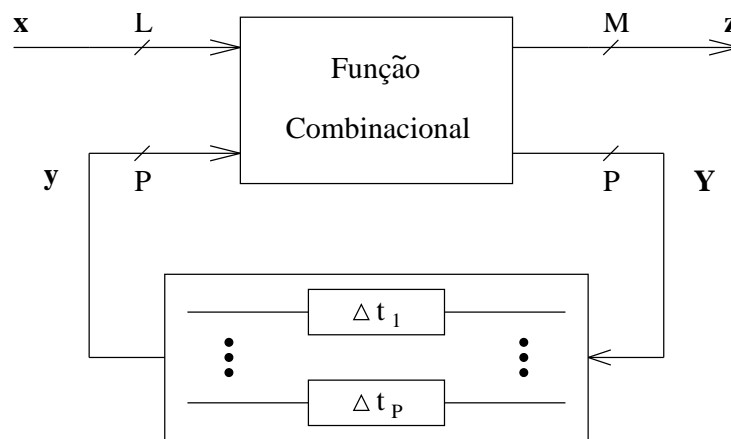


Figura 14.1: Modelo genérico para circuitos seqüenciais *level-mode*.

- 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: $y_k^{n+1} = f_k(Y_1^n, \dots, Y_P^n)$, $k = 1, 2, \dots, R$.
- Mais especificamente, neste caso: $P = R$ e $y_k(t + \Delta t_k) = Y_k(t)$, $k = 1, 2, \dots, P$.
- 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 14.1 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.

14.2 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 exhibe 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 exhibe 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.

14.3 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*).

14.4 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.

14.5 Problemas causados pela realimentação contínua

14.5.1 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 complementado. 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.

14.5.2 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 ocorrer se quaisquer duas colunas adjacentes da tabela de fluxo exibirem um dos dois padrões apresentados na Figura 14.2.

(1)	2
3	(2)
(3)	(3)

(a)

(1)	2
4	(2)
	(3)
(4)	3

(b)

Figura 14.2: Padrões de identificação de perigo essencial em tabelas de fluxo.

14.6 Solução para as corridas: atribuição de estados

14.6.1 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 14.3 apresenta um quadro resumo das mudanças de estado nos circuitos seqüenciais *level-mode*, operando em modo fundamental.

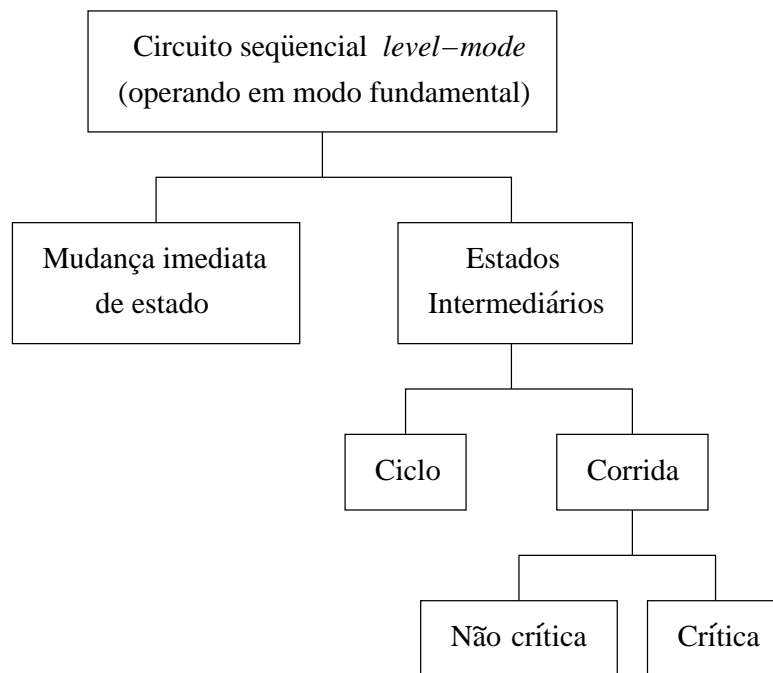


Figura 14.3: Quadro resumo das mudanças de estado nos circuitos seqüenciais *level-mode*, operando em modo fundamental.

14.6.2 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 14.1 – 14.4, 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 14.5.

		y_1	
		0	1
y_0	0	+	+
	1	+	

Tabela 14.1: 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 14.2: 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 14.3: 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 14.4: 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 14.5: 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].

14.7 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.

14.8 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 IV

Circuitos digitais programáveis

Capítulo 15

Circuitos programáveis

15.1 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 15.1 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 abordados os diversos 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 15.1: Classificação de circuitos digitais de acordo com a função realizada.

15.2 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.

15.2.1 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 trajetória do sinal, evitando eventuais desvios de trilhas. Como consequência, 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 Semiconductors¹ 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 15.2 descreve o comportamento funcional dos dispositivos NC7SZ57 e NC7SZ58. Por sua vez, a Tabela 15.3 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: $A \text{ XOR } B \equiv \overline{A} \text{ XNOR } B \equiv A \text{ XNOR } \overline{B}$ e $A \text{ XNOR } B \equiv \overline{A} \text{ XOR } B \equiv A \text{ XOR } \overline{B}$.

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 15.2: Comportamento funcional dos dispositivos NC7SZ57 e NC7SZ58.

Entradas	Saída	
	NC7SZ57	NC7SZ58
$E_0 = 0$	$\overline{E_1} \text{ NAND } E_2$	$\overline{E_1} \text{ AND } E_2$
$E_0 = 1$	$E_1 \text{ AND } E_2$	$E_1 \text{ NAND } E_2$
$E_1 = 0$	$E_0 \text{ NOR } E_2$	$E_0 \text{ OR } E_2$
$E_1 = 1$	$E_0 \text{ NAND } \overline{E_2}$	$E_0 \text{ AND } E_2$
$E_2 = 0$	$\overline{E_0}$	E_0
$E_2 = 1$	$\overline{E_1}$	E_1
$E_0 = E_1$	$E_0 \text{ XNOR } \overline{E_2}$	$E_0 \text{ XOR } E_2$
$E_0 = E_2$	$E_0 \text{ NAND } \overline{E_1}$	$E_0 \text{ AND } E_1$
$E_1 = E_2$	$\overline{E_1} \text{ NAND } E_0$	$\overline{E_1} \text{ AND } E_0$

Tabela 15.3: Opções de configuração dos dispositivos NC7SZ57 e NC7SZ58.

15.2.2 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 $\mathbf{C} = [C_{M-1} C_{M-2} \cdots C_1 C_0]$ apresentar o valor $(\mathbf{C})_2 = n = [\mathbf{C}]_n$. Logo, os padrões binários de \mathbf{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

$$AND(E_n, [C]_n) = AND(E_n, [C_{M-1} C_{M-2} \cdots C_1 C_0]_n) = AND(E_n, m(n)) ,$$

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

$$\begin{aligned} f(A, B, C, D) &= \sum m(1, 3, 5, 7, 10, 11, 12, 13) = \prod M(0, 2, 4, 6, 8, 9, 14, 15) \\ &= (\overline{A} \cdot D) + (A \cdot \overline{B} \cdot C) + (A \cdot B \cdot \overline{C}) \\ &= (A + D) \cdot (\overline{A} + B + C) \cdot (\overline{A} + \overline{B} + \overline{C}) . \end{aligned} \quad (15.1)$$

A Figura 15.1 ilustra algumas formas de organização do Mapa de Karnaugh da função exemplo, para implementação usando MUX.

		<i>CD</i>			
		00	01	11	10
<i>AB</i>	00	0	1	1	0
	01	0	1	1	0
	11	1	1	0	0
	10	0	0	1	1

(a)

		<i>BD</i>			
		00	01	11	10
<i>AC</i>	00	0	1	1	0
	01	0	1	1	0
	11	1	1	0	0
	10	0	0	1	1

(b)

		<i>BC</i>			
		00	01	11	10
<i>AD</i>	00	0	0	0	0
	01	1	1	1	1
	11	0	1	0	1
	10	0	1	0	1

(c)

Figura 15.1: 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:

$$\begin{cases} [C_2 C_1 C_0] = [B C D] \\ [E_7 E_6 \cdots E_1 E_0] = [\bar{A} 0 1 A 1 A \bar{A} 0] \end{cases}$$

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:

$$\begin{cases} [C_1 C_0] = [A B] \\ [E_3 E_2 E_1 E_0] = [\bar{C} C D D] \end{cases} ,$$

$$\begin{cases} [C_1 C_0] = [A C] \\ [E_3 E_2 E_1 E_0] = [\bar{B} B D D] \end{cases} ,$$

$$\begin{cases} [C_1 C_0] = [A D] \\ [E_3 E_2 E_1 E_0] = [(B \oplus C) (B \oplus C) 1 0] \end{cases} ,$$

$$\left\{ \begin{array}{l} [C_1 C_0] = [C D] \\ [E_3 E_2 E_1 E_0] = [(\bar{A} + \bar{B}) (A \cdot \bar{B}) (\bar{A} + B) (A \cdot B)] \end{array} \right. ,$$

$$\left\{ \begin{array}{l} [C_1 C_0] = [B D] \\ [E_3 E_2 E_1 E_0] = [(\bar{A} + \bar{C}) (A \cdot \bar{C}) (\bar{A} + C) (A \cdot C)] \end{array} \right.$$

e

$$\left\{ \begin{array}{l} [C_1 C_0] = [B C] \\ [E_3 E_2 E_1 E_0] = [(\bar{A} \cdot D) (A + D) (A + D) (\bar{A} \cdot D)] \end{array} \right. .$$

15.3 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/programmable device*).
- SPLD (*Simple Programmable Logic Device*).
- CPLD (*Complex Programmable Logic Device*).
- FPGA (*Field-Programmable Gate Array*).

Parte V
Apêndices

Apêndice A

Noções básicas sobre implementação de funções lógicas

A.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.

A.2 Conceitos básicos

Circuito integrado monolítico é um termo empregado para designar a construção de um circuito eletrônico em uma única porção de material semiconductor.

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 A.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 A.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.

A.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.

A.4 Modelo de chaves

A.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. As 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.

A.4.2 Arranjos série e paralelo de chaves

A Figura A.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.

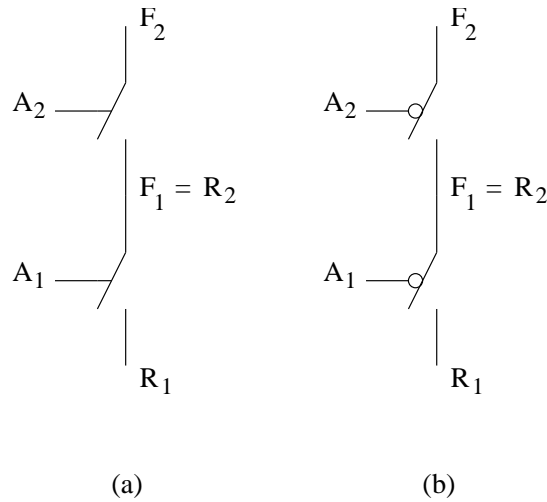


Figura A.3: Arranjo série de chaves simples. (a) Chave-N. (b) Chave-P.

A Figura A.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.

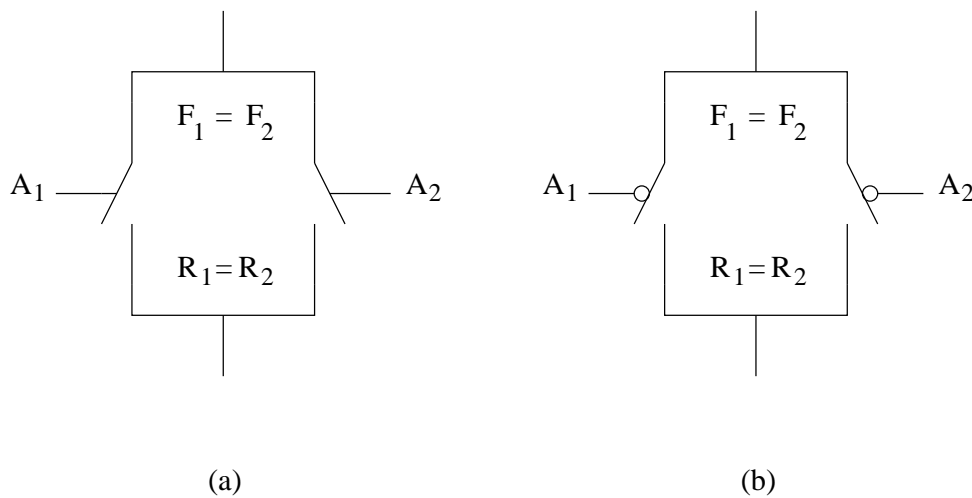


Figura A.4: Arranjo paralelo de chaves simples. (a) Chave-N. (b) Chave-P.

A.4.3 Modelo de chaves para a função NOT

Nas Figuras A.5 e A.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 A.2. Estabelecendo-se a associação $+|V| = T$ e $-|V| = F$, pode-se identificar o comportamento lógico apresentado na Tabela A.3. Esses tipos de arranjo modelam uma implementação da função lógica NOT.

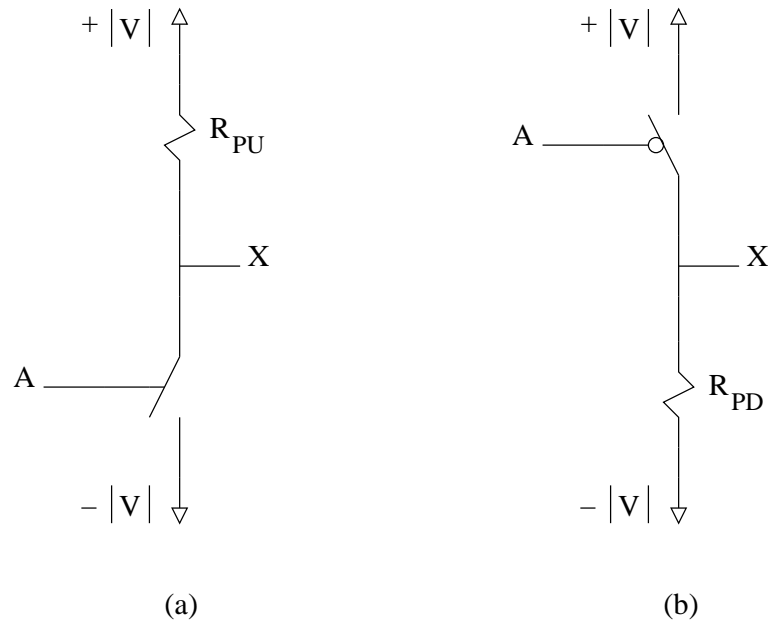


Figura A.5: Modelo de chaves simples para uma implementação da função lógica NOT, onde: (a) chave N e (b) chave P.

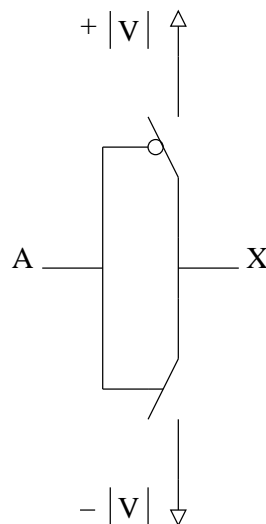


Figura A.6: Modelo de chaves complementares para uma implementação da função lógica NOT.

V_A	$V_X = f(V_A)$
$- V $	$+ V $
$+ V $	$- V $

Tabela A.2: Comportamento elétrico do modelo de chaves NOT.

A	$X = f(A)$
F	T
T	F

Tabela A.3: Comportamento lógico do modelo de chaves NOT.

A.4.4 Modelo de chaves para a função NAND

Nas Figuras A.7 e A.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 A.4. Estabelecendo-se a associação $+|V| = T$ e $-|V| = F$, pode-se identificar o comportamento lógico apresentado na Tabela A.5. Esses tipos de arranjo modelam uma implementação da função lógica NAND.

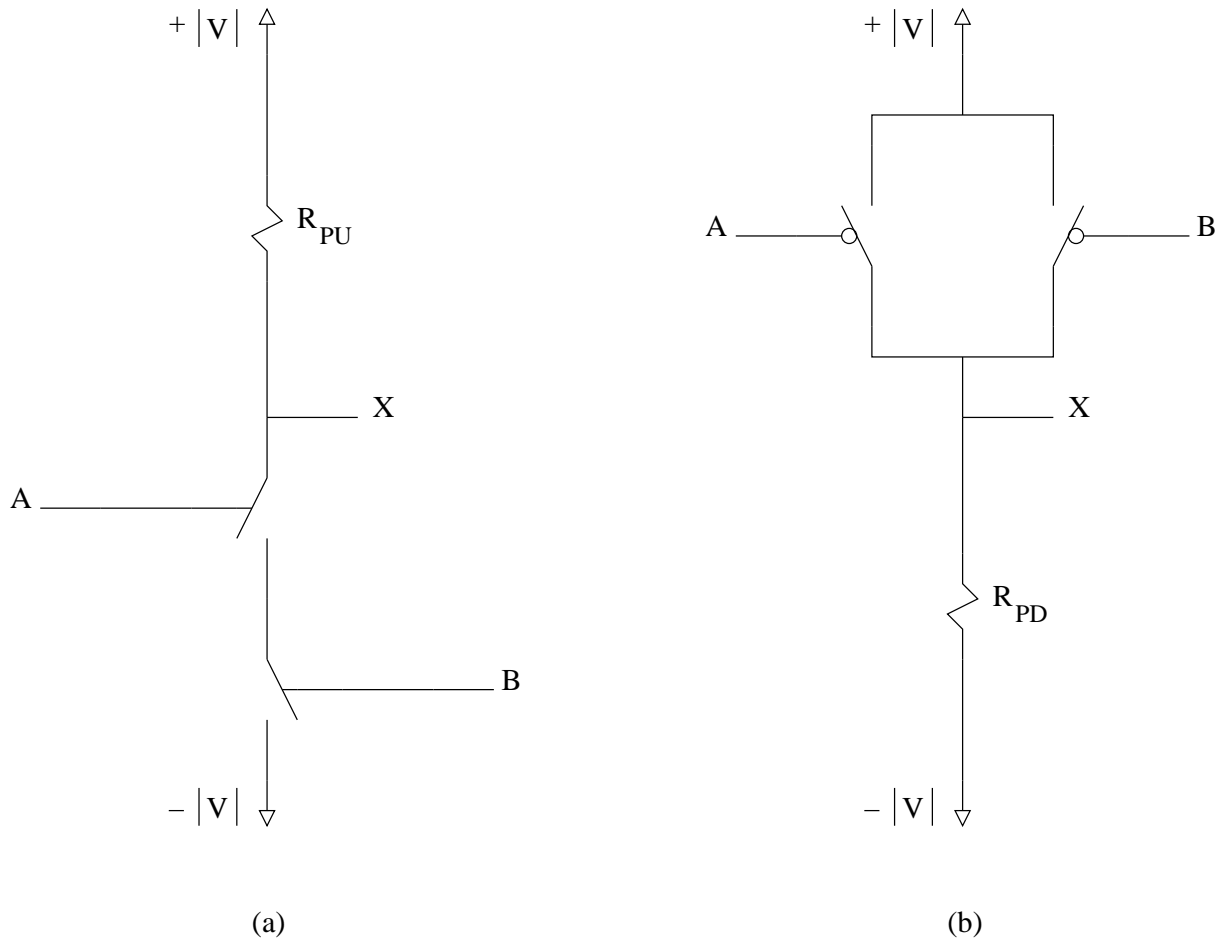


Figura A.7: Modelo de chaves simples para uma implementação da função lógica NAND, onde: (a) chave N e (b) chave P.

V_A	V_B	$V_X = f(V_A, V_B)$
$- V $	$- V $	$+ V $
$- V $	$+ V $	$+ V $
$+ V $	$- V $	$+ V $
$+ V $	$+ V $	$- V $

Tabela A.4: Comportamento elétrico do modelo de chaves NAND.

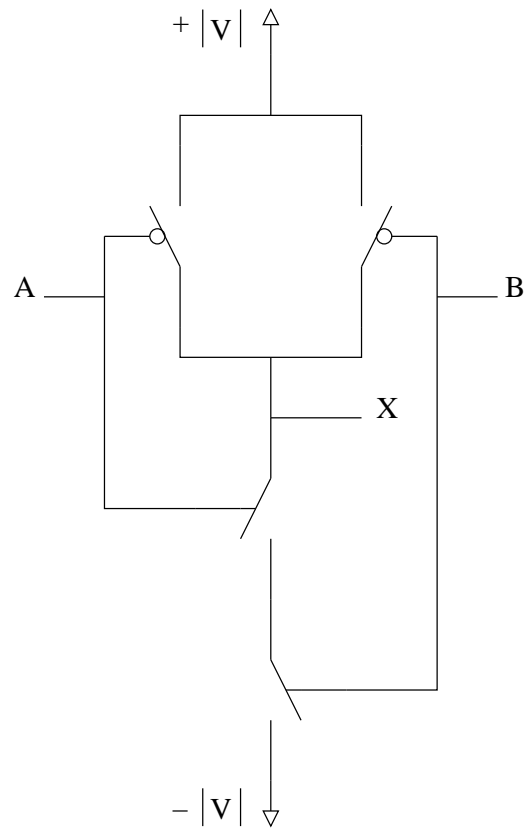


Figura A.8: Modelo de chaves complementares para uma implementação da função lógica NAND.

A	B	$X = f(A, B)$
F	F	T
F	T	T
T	F	T
T	T	F

Tabela A.5: Comportamento lógico do modelo de chaves NAND.

A.4.5 Modelo de chaves para a função NOR

Nas Figuras A.9 e A.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 A.6. Estabelecendo-se a associação $+|V| = T$ e $-|V| = F$, pode-se identificar o comportamento lógico apresentado na Tabela A.7. Esses tipos de arranjo modelam uma implementação da função lógica NAND.

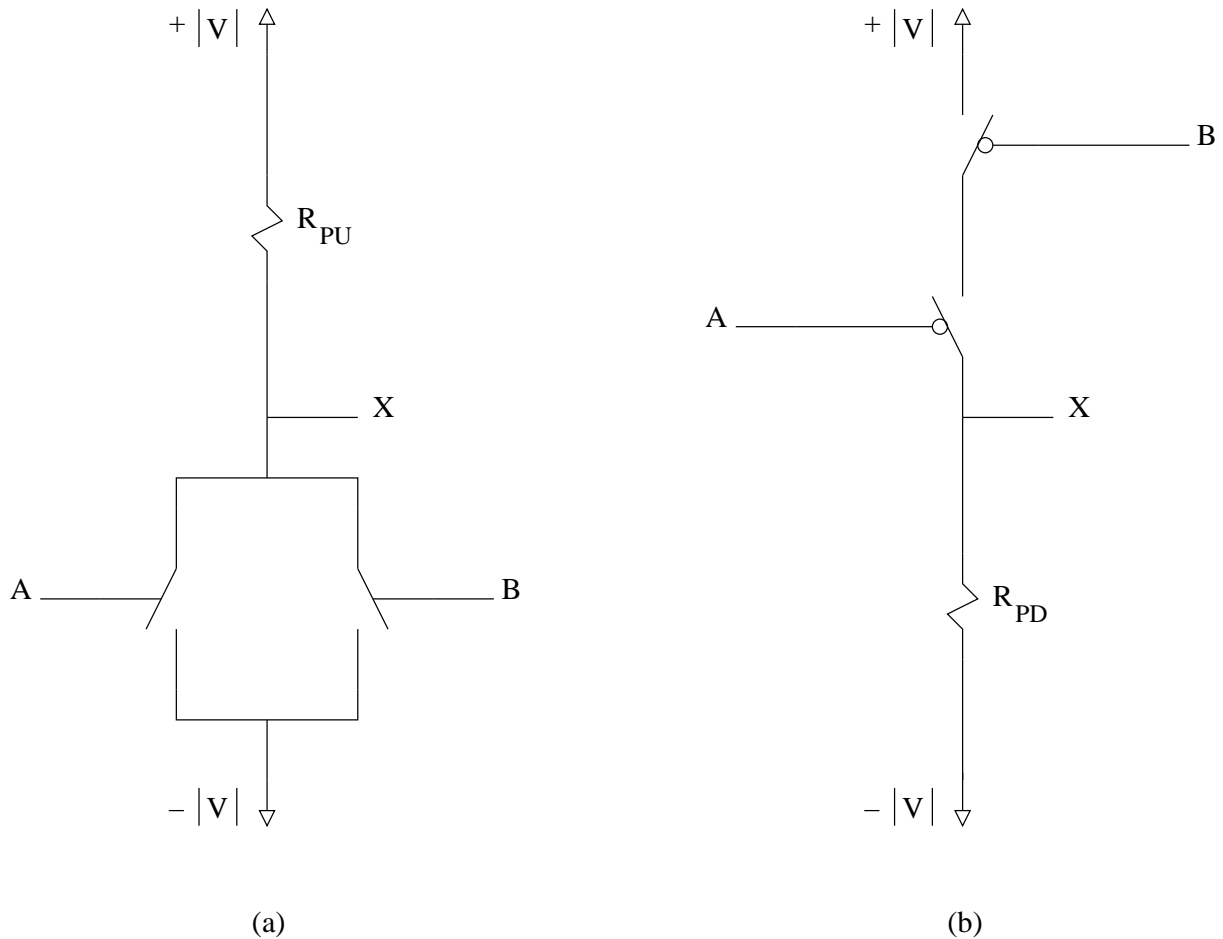


Figura A.9: Modelo de chaves simples para uma implementação da função lógica NOR, onde: (a) chave N e (b) chave P.

V_A	V_B	$V_X = f(V_A, V_B)$
$- V $	$- V $	$+ V $
$- V $	$+ V $	$- V $
$+ V $	$- V $	$- V $
$+ V $	$+ V $	$- V $

Tabela A.6: Comportamento elétrico do modelo de chaves NOR.

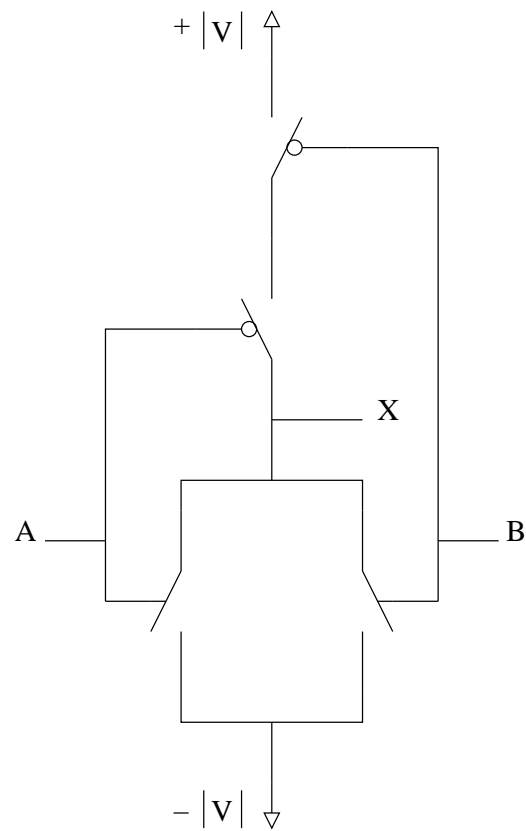


Figura A.10: Modelo de chaves complementares para uma implementação da função lógica NOR.

A	B	$X = f(A, B)$
F	F	T
F	T	F
T	F	F
T	T	F

Tabela A.7: Comportamento lógico do modelo de chaves NOR.

A.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

$$f(A, B) = \neg((A \wedge \neg B) \vee (\neg A \wedge B)),$$

$$f(A, B, C) = \neg((\neg A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge \neg C) \vee (A \wedge \neg B \wedge \neg C) \vee (A \wedge B \wedge C)),$$

e

$$f(A, B, C, D) = \neg((\neg A \wedge B \wedge \neg C \wedge D) \vee (\neg A \wedge B \wedge C \wedge \neg D) \vee (A \wedge \neg B \wedge C \wedge \neg D) \vee (A \wedge \neg B \wedge C \wedge D)).$$

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 A.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

$$f(A, B) = \neg((A \vee \neg B) \wedge (\neg A \vee B)),$$

$$f(A, B, C) = \neg((\neg A \vee \neg B \vee C) \wedge (\neg A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C) \wedge (A \vee B \vee C)),$$

e

$$f(A, B, C, D) = \neg((\neg A \vee B \vee \neg C \vee D) \wedge (\neg A \vee B \vee C \vee \neg D) \wedge (A \vee \neg B \vee C \vee \neg D) \wedge (A \vee \neg B \vee C \vee D)).$$

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 A.12.

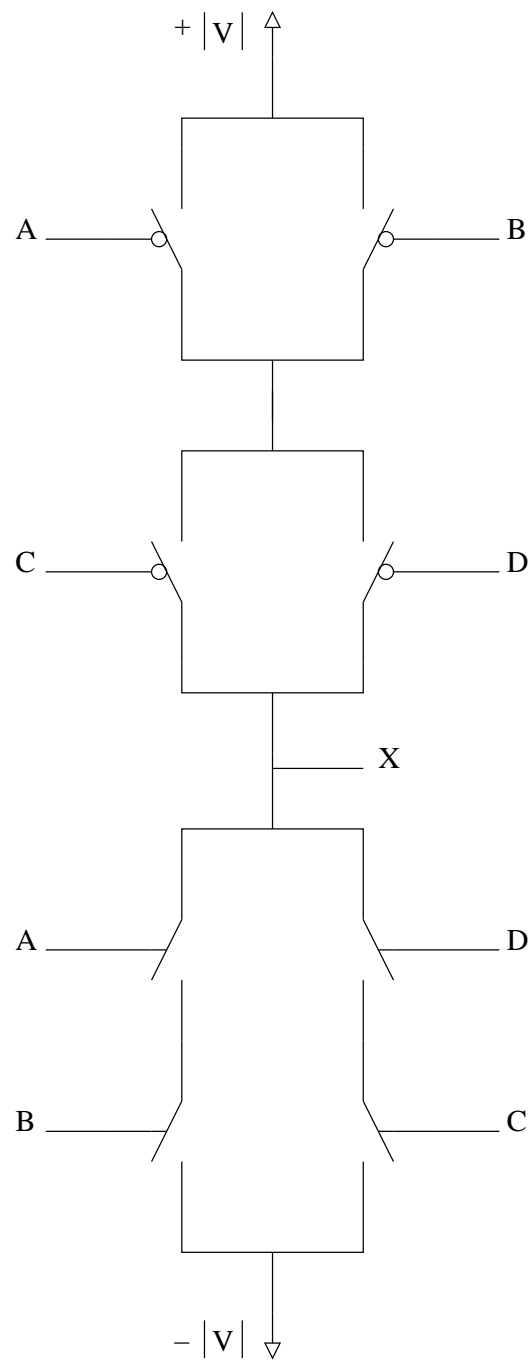


Figura A.11: Modelo de chaves complementares para uma implementação do arranjo AOI definido por $f(A, B, C, D) = \neg((A \wedge B) \vee (C \wedge D))$.

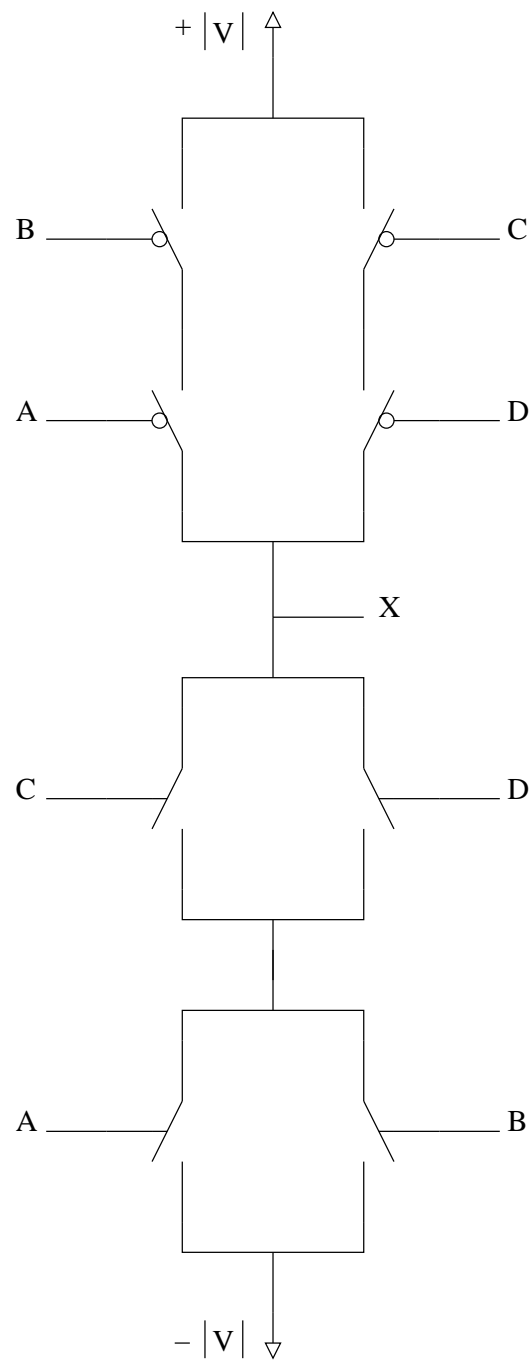


Figura A.12: Modelo de chaves complementares para uma implementação do arranjo OAI definido por $f(A, B, C, D) = \neg((A \vee B) \wedge (C \vee D))$.

A.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 herdaram 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 A.8, A.9 e A.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 A.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 A.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.

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 A.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.

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 $\neg X(A, B, C, \dots)$ o bloco de chaves-N realiza o seu complemento: $X(A, B, C, \dots)$.

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. 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 $\neg X(A, B, C, \dots)$.
- Sintetizar, através de um arranjo de chaves-N, a função $X(A, B, C, \dots)$.
- Aplicar o Teorema de De Morgan sobre $\neg X(A, B, C, \dots)$. Desconsiderar a negação das variáveis de entrada. Sintetizar a equação final através de um arranjo de chaves-P.

A.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 A.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).

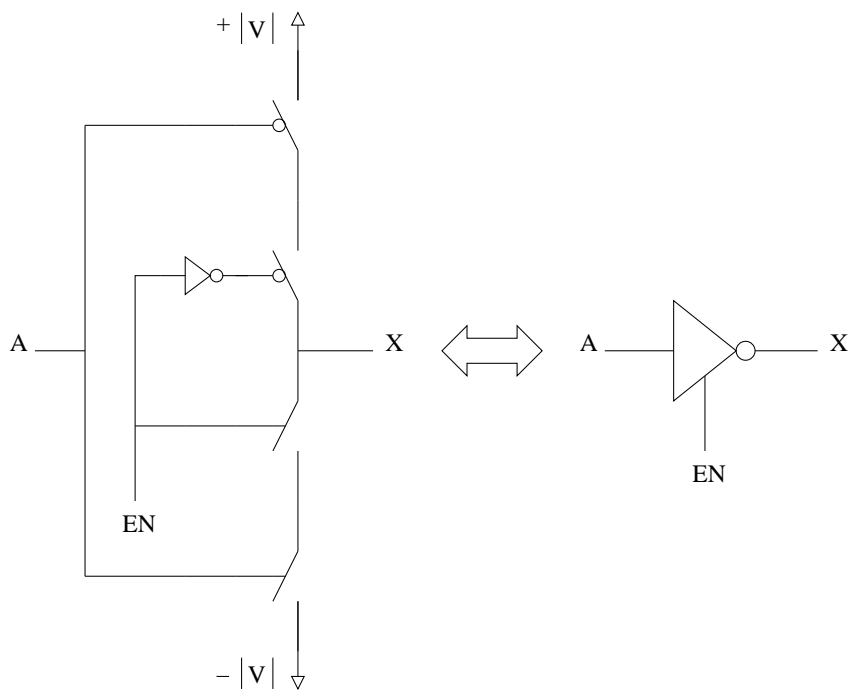


Figura A.13: Modelo de chaves complementares para uma implementação de função lógica NOT com saída em *Tri-State* (3S).

Apêndice B

Tópicos sobre divisão de números inteiros

B.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 $0 \leq r < d$.

B.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) .

B.3 Resto ou resíduo

O resto da divisão de c por d pode ser descrito por

$$r = R_d[c] = ((c)) = c \pmod{d} ,$$

podendo ainda ser denominado de resíduo de c , módulo d .

B.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

$$c_1 \equiv c_2 \pmod{d} ,$$

onde \equiv denota uma relação de equivalência.

B.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 C

Minimização de tabela de estados

C.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].

C.2 Tabelas de estados completamente especificadas

C.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 $(xRy \text{ e } yRz) \rightarrow xRz$, então R é transitiva.
 - Equivalência: se R é reflexiva, simétrica e transitiva, então R é uma relação de equivalência.

C.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 $X = x^n x^{n+1} \dots x^{n+R}$, tem-se: $\delta(q_i^n, x^n x^{n+1} \dots x^{n+R}) = q_j^{n+(R+1)}$ e $\lambda(q_i^n, x^n x^{n+1} \dots x^{n+R}) = z^n z^{n+1} \dots z^{n+R}$.
- 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 $(x^n x^{n+1} \dots x^{n+R})$ 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 $\lambda_T(p^n, x^n x^{n+1} \dots x^{n+R}) = \lambda_S(q^n, x^n x^{n+1} \dots x^{n+R})$ 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 $q \equiv p$.

C.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. $\delta(p^n, x^n) \triangleq \delta(q^n, x^n)$.

C.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 .

C.3 Tabelas de estados não completamente especificadas

C.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 mudanças 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.

C.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.

C.3.3 Formalização dos conceitos de compatibilidade e de cobertura

- **Definição 1:** Seja uma seqüência de valores de entrada $x = \{x^n x^{n+1} \dots x^{n+(K+1)}\}$, 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

$$\lambda_S (\delta(p, x^n x^{n+1} \dots x^{n+K}), x^{n+(K+1)}) = \lambda_S (\delta(q, x^n x^{n+1} \dots x^{n+K}), x^{n+(K+1)}) ,$$

sempre que ambas as saídas forem especificadas, para cada seqüência x aplicável a ambos os estados, onde $x = \{x^n x^{n+1} \dots x^{n+(K+1)}\}$.

- **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 $\{q_1, q_2, \dots, q_m\}$ da coleção e para toda entrada simples x , todos os próximos estados especificados, $\delta(q_1^n, x^n), \delta(q_2^n, x^n), \dots, \delta(q_m^n, x^n)$, 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, \dots, 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.

C.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 D

Introdução à linguagem VHDL

D.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:
 - 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.
 - Adotar uma visão hierárquica na elaboração do sistema, de forma que, em cada nível de representação, toda a complexidade dos níveis inferiores seja ocultada.

D.1.1 Linguagens de descrição de *hardware*

- Na área de projeto de circuitos integrados, diversas Linguagens de Descrição de *Hardware* (*Hardware Description Language* ou HDL) têm sido propostas, 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 associada 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 mapeamento 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 VHDL e gera um código adequado para tal implementação. Sínteses típicas são: a geração de código para dispositivos lógicos programáveis e a geração de máscaras (*layout*) para fabricação de circuitos integrados.
- Exemplos de HDL
 - Independentes de tecnologia e de fabricante: VHDL, Verilog.
 - Dependentes de fabricante: AHDL (Altera).
- A Tabela D.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
Synopsys/Synplicity	Design Compiler Ultra	Síntese
	Synplify Pro/Premier	
	VCS	Simulação
Cadence	NC-Sim	Simulação

Tabela D.1: Lista de fabricantes, produtos e funções, que lidam com HDL.

D.1.2 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 Multivalued 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.
- Algumas das características que têm levado VHDL a ser largamente empregada são as seguintes:
 - Apresenta padrões bem estabelecidos e bem documentados.
 - Apresenta características encontradas em outras HDLs.
 - Possui vasta literatura disponível.
 - Existem vários compiladores para VHDL, e de diversos tipos, tais como: (a) domínio público ou comercial e (b) implementado isoladamente ou incluído em ambiente de desenvolvimento integrado (*Integrated Development Environment* ou IDE).
- 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).

D.2 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.
- Comportamental: que apresenta um nível de representação mais abstrato e mais distante do sistema físico. Encontra aplicação 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.

D.3 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.

D.4 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.

D.4.1 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 hífen 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.

D.4.2 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 D.1 apresenta as palavras reservadas de VHDL.

abs	disconnect	label	package	then
access	downto	library	port	to
after		linkage	postponed	transport
alias	else	literal	procedure	type
all	elsif	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 D.1: Palavras reservadas de VHDL.

D.4.3 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.

D.4.4 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 D.2 apresenta símbolos especiais de VHDL.

Símbolo	Significado	Símbolo	Significado
--	Comentário		OR condicional
;	Terminador	=>	Símbolo de THEN em CASE
(Parêntese da esquerda	+	Adição ou identidade unária
)	Parêntese da direita	-	Subtração ou negação unária
:	Separação entre elemento e tipo	*	Multiplicação
<>	Declaração de faixa indefinida (<i>box</i>)	/	Divisão (com truncamento)
#	Notação <i>base#número#</i>	**	Exponenciação
.	Notação de ponto	=	Igual a
'	Aspas simples ou marca de <i>tick</i>	/=	Diferente de
”	Aspas duplas	<	Menor do que
&	Concatenador	>	Maior do que
<=	Atribuição a sinal	<=	Menor do que ou igual a
:=	Atribuição a variável/constante	>=	Maior do que ou igual a
=>	Atribuição a elemento de conjunto		

Figura D.2: Símbolos especiais de VHDL.

D.5 Conceitos básicos sobre o código VHDL

D.5.1 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 D.2 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 D.2: Operadores de VHDL.

D.5.2 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.

D.5.3 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 agendadas para o tempo corrente são executadas.
 - As execuções das transações podem causar a ocorrência de eventos sobre sinais.
 - Todos os processos sensíveis aos sinais sobre 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 agendada, a simulação atinge seu fim.

D.6 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 saídas.
- 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.

D.6.1 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.

D.6.2 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 saídas. As entradas e saídas são associadas aos pontos ou pinos de acesso do circuito físico e são conjuntamente denominadas de portas.

D.6.3 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.

D.7 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, algumas regras sintáticas de VHDL são apresentadas a seguir.

D.7.1 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.

D.7.2 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 ] ;
```

D.7.3 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
```

D.7.4 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 ;
```

D.7.5 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   ] ;
```

D.7.6 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 ) { , ... } ;

```

D.8 Exemplos de declarações genéricas

D.8.1 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;
-----

```

D.8.2 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.

D.8.3 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;
--

```

D.8.4 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.