Notas de Aula

1. Introdução aos Sistemas Digitais

Um sistema digital pode ser definido como sendo todo o sistema em que os sinais a ele pertencentes possuem um número **finito** de valores **discretos**. Diferentemente de um sistema analógico em que os valores pertencem a um **conjunto contínuo,** ou seja, **infinito,** os valores de um sistema digital possuem limites muito bem definidos para identificar os valores dos sinais. Como exemplo podemos citar um velocímetro de automóvel digital, apresentado na Figura 1a. e um velocímetro analógico apresentado na Figura 1b.

a b

Figura 1. Exemplo de diferenças entre sistema digital (a) e analógico (b).

Ao longo de um certo período de tempo conhecido como sendo o período máxima de operação do sistema digital, idealmente, os valores de um sinal digital não apresentam modificações enquanto que em um sistema analógico, os valores dos sinais podem sofrer alterações no seu valor durante o tempo. Este comportamento pode ser observado na Figura 2.

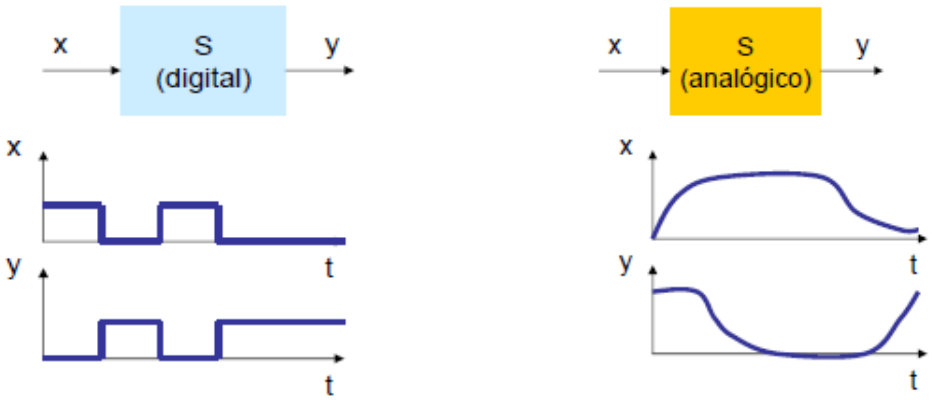


Figura 2. Valores de um sinal digital e um sinal analógico variando no tempo.

1.1 Vantagens e Desvantagens dos Sistemas Digitais

Dentre as vantagens dos sistemas digitais podemos destacar:

- Adequados tanto ao processamento numérico (0, 1, 2, 3, ...) quanto ao não-numérico (A, B, C, D, ...);
- Fáceis de projetar e armazenar a informação na forma de tensão elétrica;
- Possuem precisão e exatidão no processamento da informação, pois podem utilizar tantos dígitos quanto forem necessários para atingir às necessidades da aplicação;
- Os sinais digitais podem ser bastante insensíveis a variações ambientais como, por exemplo, temperatura de operação, pois os valores dos sinais podem sofrer pequenas alterações sem que o valor digital seja alterado;
- Os sistemas digitais podem ser programados através de linguagens de programação para facilitar o desenvolvimento de novos sistemas;
- Os circuitos digitais são extremamente baratos e fáceis de integrar em placas de circuitos;

As desvantagens dos sistemas digitais estão fortemente relacionadas às suas limitações. Como o mundo real é predominantemente analógico, os sistemas digitais estão limitados a precisão e poder computacional disponível. Por mais que a tecnologia evolua, os sinais digitais nunca irão atingir a exatidão da informação analógica e portanto, sempre serão limitados. Quanto maior a precisão e a capacidade de um sistema digital, maior é o seu custo.

A figura 3 ilustra o processo de digitalização da informação de um sinal sonoro, seu tratamento e processamento e finalmente, a sua transformação final em um novo sinal analógico.

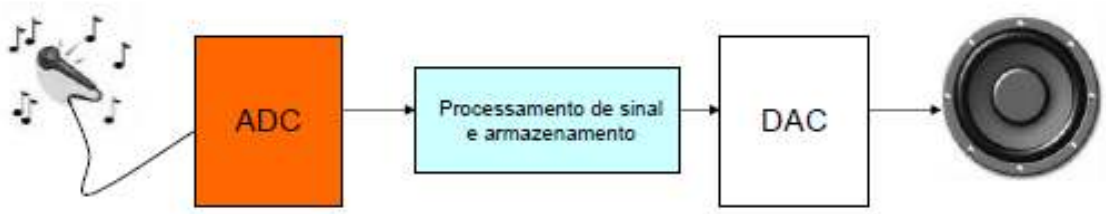


Figura 3. Etapas de um sinal digital.

1.2 Sistemas Digitais Combinacionais e Sequenciais

Os sistemas digitais podem ser divididos em sistemas combinacionais e sistemas seqüenciais. Sistema combinacional é um sistema em que as saídas em qualquer instante de tempo dependem apenas dos valores das entradas nesse instante de tempo. Já em um sistema seqüencial, as saídas do sistema em um dado instante de tempo dependem não só dos valores das entradas nesse instante de tempo, mas também dos valores em instantes anteriores: ou seja, sistemas seqüenciais necessariamente possuem elementos memória. Em um sistema seqüencial encontramos um sinal conhecido como sincronizador do sistema que é comumente

chamado de relógio do sistema (do inglês *clock*). Durante o nosso curso iremos estudar apenas os sistemas digitais combinacionais.

Exercícios:

1. Qual é a diferença entre um sistema digital e um sistema analógico?
2. Cite pelo menos 3 vantagens dos sistemas digitais.
3. Qual é a diferença entre um sistema combinacional e um sistema seqüencial?

2. Sistemas Numéricos

Uma característica comum a todos os sistemas numéricos utilizados pelo ser humano é que eles são posicionais, ou seja, cada dígito que forma o valor de um número possui um peso associado. Desta forma, o valor de um dado número corresponde a uma soma ponderada de seus dígitos. Para entender melhor este conceito, acompanhe o exemplo abaixo:

$$1234 = \mathbf{1} \times 1000 + \mathbf{2} \times 100 + \mathbf{3} \times 10 + \mathbf{4} \times 1$$

Podemos reescrever o valor que multiplica cada digito como sendo uma potência de 10.

$$1234 = \mathbf{1} \times 10^{\mathbf{3}} + \mathbf{2} \times 10^{\mathbf{2}} + \mathbf{3} \times 10^{\mathbf{1}} + \mathbf{4} \times 10^{\mathbf{0}}$$

Desta maneira, podemos escrever qualquer número decimal 'D' como sendo igual a soma do produto dos dígitos (d3, d2, d1, ...) por 10 elevado na potencia da posição do dígito menos um.

$$D = \mathbf{d_3} \times 10^{\mathbf{3}} + \mathbf{d_2} \times 10^{\mathbf{2}} + \mathbf{d_1} \times 10^{\mathbf{1}} + \mathbf{d_0} \times 10^{\mathbf{0}}$$

Podemos reescrever a expressão utilizando a fórmula de somatório e colocando as variáveis m e n como sendo os limites iniciais e finais do número:

$$D = \sum_{i=n}^{m} d_i \,.\, 10^i$$

Os seres humanos utilizam como sendo o seu sistema principal, o sistema **decimal**. Esta escolha decorre principalmente ao fato de que as ferramentas mais básicas do ser humano que são as suas mãos, possuem 10 dedos no total. No sistema decimal, utilizamos o número 10 como sendo a **base** do sistema do nosso sistema numérico. Em um sistema numérico, a base indica o número de dígitos que pertencem ao sistema e que, no caso da base decimal, são os números que vão de 0 a 9.

Com a evolução da microeletrônica e com o surgimento da tecnologia da informação surgiu o sistema **binário**. Este sistema diferentemente do sistema decimal, utiliza apenas dois números: '0' e '1' para formar os números na base binária, como pode ser observado no exemplo abaixo.

$$21_{10} = 10101_2$$

No exemplo acima, é apresentado o número binário correspondente ao número 21 em decimal. Da mesma forma que podemos representar um número decimal como sendo a soma dos dígitos multiplicados pela base numérica, elevada na potência da posição do dígito menos um, podemos também fazer a mesma representação para os números binários.

$$10101_2 = \mathbf{1} \times 2^{\mathbf{4}} + \mathbf{0} \times 2^{\mathbf{3}} + \mathbf{1} \times 2^{\mathbf{2}} + \mathbf{0} \times 2^{\mathbf{1}} + \mathbf{1} \times 2^{\mathbf{0}}$$

$$B = \sum_{i=n}^{m} b_i \,.\, 2^i$$

Em sistemas eletrônicos digitais, as informações binárias são representadas por valores de tensão (ou correntes) que estão presentes tanto nas entradas quanto nas saídas dos circuitos. Tipicamente, os números binários '0' e '1', são representados pelos níveis de tensão 0 Volts (V) para o valor '0' e 5 V para o valor '1'. Na realidade, o que encontramos nos circuitos digitais são faixas de valores de tensão utilizados para representar os valores binários '0' e '1'. Tipicamente encontramos o valor binário '0' com sendo uma tensão que pode ir de 0V a 0,8V e o valor binário '1' variando de 2V a 5V, dependendo do circuito utilizado. Circuitos mais voltados a redução do consumo de energia, utilizam valores de tensão menores para representar o '1' binário como por exemplo 3,3 V.

2.1 Base Octal e Hexadecimal

Além do sistema decimal e do sistema binário, dois outros sistemas são de grande importância por proverem representações convenientemente compactas de números grandes. Trata-se dos sistemas octal (base 8) e hexadecimal (base 16). No sistema octal, cada dígito representa um valor entre 0 e 7. Já no sistema hexadecimal, cada dígito representa um valor entre 0 e 15. Para representar os valores maiores do que 9 usando apenas um dígito, utilizam-se letras. Assim, o valor 10 é representado por A, o 11, por B e assim por diante, até 15 (que é representado por F). A Tabela 1 abaixo mostra os números décimal, binário, octal e hexadecimal dos números de 0 a 20.

Tabela 1. **Valores para os números de 0 a 20 nas bases binária, octal e hexadecimal.**

Decimal	Binário	Octal	Hexadecimal
0	00000	00	00
1	00001	01	01
2	00010	02	02
3	00011	03	03
4	00100	04	04
5	00101	05	05
6	00110	06	06
7	00111	07	07
8	01000	10	08
9	01001	11	09
10	01010	12	0A
11	01011	13	0B
12	01100	14	0C
13	01101	15	0D
14	01110	16	0E
15	01111	17	0F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14

2.2 Conversão entre bases numéricas

Geralmente, não se pode converter um número representado numa determinada base para outra base simplesmente substituindo-se dígitos da base original pelos seus equivalentes na outra. Isto funciona somente nos casos em que ambas as bases são potências de um mesmo número (como por exemplo, de binário para octal ou hexadecimal). Quando não é este o caso, é necessário utilizar-se operações aritméticas. A seguir, será mostrado como converter um número em qualquer base para a base 10, e viceversa, usando aritmética de base 10.

Como foi dito anteriormente, os números em qualquer base podem ser representados pela soma dos dígitos multiplicados pela base numérica, elevada na potencia da posição do dígito menos um. Vamos pegar o exemplo anterior de um número binário '10111' e expressá-lo na forma de soma de produtos pela base na potencia da posição:

$$10111_2 = \mathbf{1} \times 2^{\mathbf{4}} + \mathbf{0} \times 2^{\mathbf{3}} + \mathbf{1} \times 2^{\mathbf{2}} + \mathbf{1} \times 2^{\mathbf{1}} + \mathbf{1} \times 2^{\mathbf{0}}$$

Para obter o correspondente decimal, basta resolver a equação.

D = 1x16 + 0x8 + 1x4 + 1x2 + 1x1
D = 16 + 4 + 2 + 1 = 23

Logo o número 10111 na base 2 tem como equivalente o número 23 na base 10.

Para se fazer a transformação contrária, ou seja, obter o equivalente binário de um número decimal pode-se utilizar o método das divisões sucessivas. Este método consiste em obter o valor correspondente em binário, através de divisões sucessivas pelo valor da base, no caso 2 e ir anotando apenas o resto até não existir mais nada para dividir. Para exemplificar este método, vamos utilizar o mesmo número do exemplo anterior, ou seja, o número decimal 21 e obter o equivalente binário.

$\frac{23}{2} = 11$ e resta	1	↑ O
		R
$\frac{11}{2} = 5$ e resta	1	D
		E
$\frac{5}{2} = 2$ e resta	1	M
$\frac{2}{2} = 1$ e resta	0	D
		O
$\frac{1}{2} = 0$ e resta	1	
		N^0

$23_{10} = 10111_2$

Exercício exemplo:
Converta os números a seguir na base requerida.

a) $201_{10} = ?_2$
b) $123_8 = ?_2$

c) $2A_{16} = ?_2$
d) $65_7 = ?_4$

Outros Exercicios Propostos

1. Converta os seguintes números binários em decimal. *
   a) 10110 b) 100100001001 c) 11111111 d) 1111010111

2. Converta os seguintes valores decimais em binário. *
   a) 37 b) 189 c) 77 d) 205

3. Faça a conversão para a base requerida.
   a) $1417_{10} = ?_2$ b) $11010001_2 = ?_{10}$ c) $2497_{10} = ?_{16}$ d) $1600_{10} = ?_{16}$
   e) $3E1C_{16} = ?_2$ f) $525_8 = ?_2$ g) $123123_4 = ?_2$ h) $367_9 = ?_{16}$

3. Álgebra Booleana

Em 1854, George Boole introduziu o formalismo que até hoje se usa para o tratamento sistemático da lógica, que é a chamada Álgebra Booleana. Álgebra Booleana pode ser definida com um conjunto de operadores e um conjunto de axiomas, que são assumidos verdadeiros sem necessidade de prova. Foi somente em 1938 que C. E. Shannon fez a relação entre a álgebra booleana e circuitos elétricos. Ele demonstrou que as propriedades de circuitos elétricos de chaveamento podem ser representadas por uma álgebra Booleana com dois valores. Diferentemente da álgebra ordinária dos reais, onde as variáveis podem assumir valores no intervalo ($-\infty$; $+\infty$), as variáveis Booleanas só podem assumir um número finito de valores. Em particular, na álgebra Booleana de dois valores, cada variável pode assumir um dentre dois valores possíveis, os quais podem ser denotados por [F,V] (falso ou verdadeiro), [H,L] (*high and low*, em português alto e baixo) ou ainda [0,1]. Nesta disciplina, adotaremos a notação [0,1], a qual também é utilizada em eletrônica digital. Como o número de valores que cada variável pode assumir é finito (e pequeno), o número de estados que uma função Booleana pode assumir também será finito, o que significa que podemos descrever completamente as funções Booleanas utilizando tabelas. Devido a este fato, uma tabela que descreva uma função Booleana recebe o nome de tabela verdade, e nela são listadas todas as combinações de valores que as variáveis de entrada podem assumir e os correspondentes valores da função (saídas).

Na álgebra Booleana, existem três operações lógicas ou funções básicas. São elas, operação 'OU', operação 'E' e complementação ou inversão. Todas as funções Booleanas podem ser representadas em termos destas operações básicas.

3.1 Operação ou lógica OU

Uma definição para a operação **OU**, que também é denominada adição lógica, é:
"A operação **OU** resulta **1** se pelo menos uma das variáveis de entrada vale **1**". Como uma variável Booleana ou vale **1** ou vale **0**, e como o resultado de uma operação qualquer pode ser atribuído a uma variável Booleana, basta que definamos quando a operação vale **1**. Automaticamente, a operação resultará **0** nos demais casos. Assim, pode-se dizer que a operação **OU** resulta **0** somente quando todas as variáveis de entrada valem **0**. Um símbolo possível para representar a operação **OU** é "+", tal como o símbolo da adição algébrica (dos reais). Porém, como estamos trabalhando com variáveis Booleanas, sabemos que não se trata da adição algébrica, mas sim da adição lógica. Outro símbolo também encontrado na bibliografia é "$\vee$". Listando as possibilidades de combinações entre dois valores Booleanos e os respectivos resultados para a operação **OU**, tem-se:

$$0 + 0 = 0 \qquad 0 \vee 0 = 0$$
$$0 + 1 = 1 \qquad 0 \vee 1 = 1$$
$$1 + 0 = 1 \qquad 1 \vee 0 = 1$$
$$1 + 1 = 1 \qquad 1 \vee 1 = 1$$

Note que a operação **OU** só pode ser definida se houver, pelo menos, duas variáveis envolvidas. Ou seja, não é possível realizar a operação sobre somente uma variável. Devido a isso, o operador "+" ou "$\vee$" (lêem-se **OU**) são ditos **binários**.

Nas equações, não se costuma escrever todas as possibilidades de valores e portanto, adotamos uma letra (ou uma letra com um índice) para designar uma variável Booleana. Com

isso, já se sabe que aquela variável pode assumir ou o valor **0** ou o valor **1**. Para demonstrar o comportamento da lógica **OU** através da equação A+B ou A∨B (lê-se A ou B), utilizamos o que se chama de tabela verdade. Uma tabela verdade consiste basicamente de um conjunto de colunas, nas quais são listadas todas as combinações possíveis entre as variáveis de entrada (à esquerda) e o resultado da função (à direita). A Tabela 2 apresemta a tabela verdade do operador lógico **OU.** Também, podem-se criar colunas intermediárias, onde são avaliados os resultados de subexpressões contidas na expressão principal. Isto normalmente facilita a avaliação, principalmente no caso de equações muito complexas e/ou contendo muitas variáveis.

Tabela 2. **Tabela Verdade do operador binário OU**

A	B	A ∨ B
0	0	0
0	1	1
1	0	1
1	1	1

3.2 Operação ou lógica E

A operação **E**, ou **multiplicação** lógica, pode ser definida da seguinte forma:

"A operação **E** resulta '**0'** se pelo menos uma das variáveis de entrada vale '**0**. Pela definição dada, pode-se deduzir que o resultado da operação **E** será **1** se, e somente se, todas as entradas valerem **1**. O símbolo usualmente utilizado na operação **E** é " · " porém, outra notação possível é "∧". Podemos, também, listar as possibilidades de combinações entre dois valores Booleanos e os respectivos resultados, para a operação **E**:

$0 \cdot 0 = 0$
$0 \cdot 1 = 0$
$1 \cdot 0 = 0$
$1 \cdot 1 = 1$

$0 \wedge 0 = 0$
$0 \wedge 1 = 0$
$1 \wedge 0 = 0$
$1 \wedge 1 = 1$

Assim como a operação **OU**, a operação **E** só pode ser definida entre pelo menos duas variáveis. Ou seja, o operador " · " ou " ∧ " (lêem-se **E**) também são ditos **binários**. Para mostrar o comportamento da equação A ∧ B (lê-se A e B), a Tabela 3 apresenta a tabela verdade:

Tabela 3. **Tabela Verdade do operador binário E**

A	B	A ∧ B
0	0	0
0	1	0
1	0	0
1	1	1

3.3 Complementação (ou Negação, ou Inversão)

A operação **complementação** dispensa uma definição. É a operação cujo resultado é simplesmente o valor complementar ao que a variável apresenta. Também

devido ao fato de uma variável Booleana poder assumir um entre somente dois valores, o valor complementar será **1** se a variável vale **0** e será **0** se a variável vale **1**.
Os símbolos utilizados para representar a operação complementação sobre uma variável Booleana A são $\overline{A}$, ~A e A' (lê-se A negado). Nesta disciplina, adotaremos o primeiro símbolo. O resultado da operação complementação pode ser listado:

$$\overline{0} = 1$$
$$\overline{1} = 0$$

Diferentemente das operações **OU** e **E,** a **complementação** só é definida sobre uma variável, ou sobre o resultado de uma expressão. Ou seja, o operador **complementação** é dito **unário**. E a tabela verdade Tabela 4 para A é:

Tabela 4. **Tabela Verdade do operador Complementação**

A	$\overline{A}$
0	1
1	0

3.4 Lógicas OU e E com Multiplas Variáveis

Como ficará a lógica OU com mais de 2 variáveis?

Pode-se determinar o resultado da equação $A \vee B \vee C$ (lê-se A ou B ou C) utilizando diretamente a definição da operação **E**: o resultado será **1** se pelo menos uma das variáveis de entrada valer **1**

No exemplo a seguir, temos a Tabela 5 apresenta a tabela verdade do operador **OU** com três variáveis e abaixo a 0 apresenta a tabela com o operador **E**.

Tabela 5. **Tabela Verdade do operador binário OU com 3 variáveis.**

A	B	C	$A \vee B \vee C$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

De forma semelhante, pode-se determinar o resultado da equação $A \wedge B \wedge C$ (lê-se A e B e C) utilizando diretamente a definição da operação **E**: o resultado será **0** se pelo menos uma das variáveis de entrada valer **0**.

Tabela 6. **Tabela Verdade do operador binário E com 3 variáveis.**

A	B	C	$A \wedge B \wedge C$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

3.5 Propriedades da Lógica Booleana

Outra maneira de avaliar uma equação booleana é avaliando-a em pares. Por exemplo, pode-se primeiramente achar o resultado de $A \vee B$, para depois operar os valores resultantes com os respectivos valores de C. Esta propriedade é conhecida como **associatividade**. Também a ordem em que são avaliadas as variáveis A, B e C é irrelevante (propriedade **comutatividade**). Estas propriedades são ilustradas pela Tabela 7 a seguir. Nela, os parêntesis indicam subexpressões já avaliadas em coluna imediatamente à esquerda. Note que os valores das colunas referentes às expressões $A \vee B \vee C$, $(A \vee B) \vee C$ e $(B \vee C) \vee A$ são os mesmos (na mesma ordem).

Tabela 7. **Tabela Verdade demonstrando as propriedades de associatividade e comutatividade do operador lógico OU**

A	B	C	$A \vee B \vee C$	$A \vee B$	$(A \vee B) \vee C$	$B \vee C$	$(B \vee C) \vee A$
0	0	0	0	0	0	0	0
0	0	1	1	0	1	1	1
0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	1	1
1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1

Também para a operação **E** valem as propriedades de **associatividade** e **comutatividade**. Logo, da mesma forma que acontece para a lógica **OU**, a equação $A \wedge B \wedge C$ pode ainda ser avaliada tomando-se as variáveis aos pares, em qualquer ordem.

Abaixo a 0 apresenta a tabela verdade demonstrando as propriedades de associatividade e comutatividade.

Tabela 8. **Tabela Verdade demonstrando as propriedades de associatividade e comutatividade do operador lógico E**

A	B	C	$A \wedge B \wedge C$	$A \wedge B$	$(A \wedge B) \wedge C$	$B \wedge C$	$(B \wedge C) \wedge A$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	1	0	0	0	1	0
1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0
1	1	1	1	1	1	1	1

3.6 Avaliação de Expressões Booleanas

Dada a equação que descreve uma função Booleana qualquer, deseja-se saber detalhadamente como esta função se comporta para qualquer combinação das variáveis de entrada. O comportamento de uma função é descrito pela sua tabela verdade e este problema é conhecido como **avaliação** da função ou da expressão que descreve a função considerada. Em suma, deseja-se achar a tabela verdade para a função Booleana.

Como já foi dito anteriormente, uma tabela verdade consiste basicamente de um conjunto de colunas, nas quais são listadas todas as combinações possíveis entre as variáveis de entrada (à esquerda) e o resultado da função (à direita). Também, pode-se criar colunas intermediárias, onde são listados os resultados de subexpressões contidas na expressão principal. Isto normalmente facilita a avaliação, principalmente no caso de equações muito complexas e/ou contendo muitas variáveis.

Quando numa mesma equação Booleana aparecem operações **E** e **OU**, é necessário seguir a ordem de precedência. Tal como na álgebra dos reais, a multiplicação (lógica) tem precedência sobre a adição (lógica). Além disso, expressões entre parêntesis têm precedência sobre operadores **E** e **OU** que estejam no mesmo nível. Quanto à complementação, esta deve ser avaliada tão logo seja possível. Caso a complementação seja aplicada sobre uma subexpressão inteira, é necessário que se avalie primeiramente a subexpressão para, só após, inverter o seu resultado.

O número de combinações que as variáveis de entrada podem assumir pode ser calculado por 2^n, onde n é o número de variáveis de entrada. O procedimento para a criação da tabela verdade a partir de uma equação Booleana é:

1. Criar colunas para as variáveis de entrada e listar todas as combinações possíveis, utilizando a fórmula no de combinações = 2^n (onde n é o número de variáveis de entrada);
2. Criar uma coluna para cada variável de entrada que apareça complementada na equação e anotar os valores resultantes;
3. Avaliar a equação seguindo a ordem de precedência, a partir do nível de parêntesis mais internos:
    - 1^o multiplicação lógica
    - 2^o adição lógica

Tomemos como exemplo a expressão $W = X + Y \cdot \overline{Z}$, onde a variável W representa a função Booleana propriamente dita. Esta variável depende das variáveis que estão à direita do sinal '=', ou seja, depende de X, Y e Z o que nos dá um total de 3 variáveis de entrada. O total de combinações entre 3 variáveis será $2^3 = 8$. Então, a tabela verdade para W deverá ter 3 colunas à esquerda e 8 linhas conforme o número total de combinações possível das entradas. Seguindo o procedimento dado acima, cria-se uma coluna, na qual se listam os valores para $\overline{Z}$. Após, inicia-se a avaliação propriamente dita, a partir do nível mais interno de parêntesis. Como não há parêntesis na expressão, resolvem-se as subexpressões que envolvem a operação **E**. No caso em questão, há somente uma subexpressão, que é $Y \cdot Z$. Então, cria-se uma coluna para $Y \cdot \overline{Z}$, na qual anotam-se os resultados para este produto. Finalmente, utilizam-se os resultados de $Y \cdot \overline{Z}$, listados na coluna anterior, para operar o **OU** com a variável X. A 0 apresenta o resultado final da tabela verdade para a expressão de W.

Tabela 9. **Tabela verdade que apresenta a solução para a equação de W**

X	Y	Z	$\overline{Z}$	$Y \cdot \overline{Z}$	W
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	1	1
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	0	0	1
1	1	0	1	1	1
1	1	1	0	0	1

3.7 Leis Fundamentais e Propriedades da Álgebra Booleana

As leis da álgebra Booleana dizem respeito ao espaço Booleano (isto é., valores que uma variável pode assumir) e operações elementares deste espaço. Já as propriedades podem ser deduzidas a partir das definições das operações.

Sejam A e B duas variáveis Booleanas. Então, o espaço Booleano é definido:

se $A \neq 0$, então $A=1$;
se $A \neq 1$, então $A=0$.

As operações elementares deste espaço são operações **OU**, operações **E** e **complementação**, cujas definições foram dadas nas seções 3.1, 3.2 e 3.3, respectivamente.

As propriedades da álgebra Booleana são as seguintes.

Da adição lógica:

$$A + 0 = A \quad (1)$$
$$A + 1 = 1 \quad (2)$$
$$A + A = A \quad (3)$$
$$A + \overline{A} = 1 \quad (4)$$

Da multiplicação lógica:

$$A \cdot 0 = 0 \quad (5)$$
$$A \cdot 1 = A \quad (6)$$
$$A \cdot A = A \quad (7)$$
$$A \cdot \overline{A} = 0 \quad (8)$$

Da complementação:

$$\overline{\overline{A}} = A \quad (9)$$

Comutatividade:

$$A + B = B + A \quad (10)$$
$$A \cdot B = B \cdot A \quad (11)$$

Associatividade:

$$A + (B + C) = (A + B) + C = (A + C) + B \quad (12)$$
$$A \cdot (B \cdot C) = (A \cdot B) \cdot C = (A \cdot C) \cdot B \quad (13)$$

Distributiva (da multiplicação em relação à adição):

$$A \cdot (B + C) = A \cdot B + A \cdot C \quad (14)$$

3.8 Teoremas de De Morgan

O primeiro teorema de De Morgan diz que a complementação de um produto lógico equivale à soma lógica das negações de cada variável do referido produto. A equação abaixo generaliza este teorema:

$$\overline{A \cdot B \cdot C \cdot \ldots} = \overline{A} + \overline{B} + \overline{C} + \ldots$$

O segundo teorema é o contraponto do primeiro, ou seja, a complementação de uma soma lógica equivale ao produto lógico das negações individuais das variáveis:

$$\overline{A + B + C + \ldots} = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \ldots$$

3.9 Propriedade Especial

$$A + \bar{A}.B = A + B$$

3.1 Complete cada expressão

a) A + 1 = 1
b) A . A = A
c) B . $\overline{B}$ = 0
d) C + C = C
e) x . 0 = 0
f) D . 1 = D
g) D + 0 = D

h) $C + \overline{C} = 1$
i) $G + GF = G \cdot (1 + F) = G \cdot (1) = G$
j) $y + \overline{w}y = y \cdot (1 + /w) = y \cdot (1) = y$

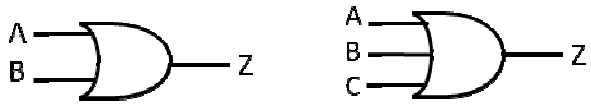
4. Portas Lógicas

Vimos que uma função Booleana pode ser representada por uma equação ou expressada pela sua tabela verdade. Veremos nesta seção, que uma função Booleana também pode ser representada de forma gráfica, onde cada operador lógico é associado a um símbolo específico, permitindo um reconhecimento visual da operação de forma mais direta. Tais símbolos são conhecidos por **portas lógicas**.

Na realidade, mais do que símbolos de operadores lógicos ou as portas lógicas representam recursos físicos, isto é, circuitos eletrônicos capazes de realizar as operações lógicas. Na eletrônica que trabalha com somente dois níveis, a qual é denominada eletrônica digital, o nível lógico 0 normalmente está associado à ausência de tensão (0 volt) enquanto o nível lógico 1, à presença de tensão (a qual geralmente é 5 volts). Nesta disciplina, nos limitaremos ao mundo da álgebra Booleana, admitindo que as portas lógicas representam circuitos eletrônicos que, de alguma maneira, realizam as funções Booleanas. Então, ao conjunto de portas lógicas e respectivas conexões que simbolizam uma **equação Booleana**, chamaremos de **circuito lógico**.

4.1 Porta OU

O símbolo da **porta OU** pode ser visto na Figura 4. Como pode ser visto na figura as entradas são colocadas à esquerda da porta e a saída, à direita. Deve haver no mínimo duas entradas, mas há somente uma saída. O funcionamento da porta **OU** segue a definição da operação **OU**, dada na seção 3.1.



(a) (b)

Figura 4. Símbolo da porta lógica **OU** com 2 entradas (a) e com 3 entradas (b).

4.2 Porta E

O símbolo da **porta E** é mostrado na Figura 5. À esquerda estão dispostas as entradas (no mínimo duas, obviamente) e à direita, a saída (única). As linhas que conduzem as variáveis de entrada e saída podem ser interpretadas como fios que transportam os sinais elétricos associados às variáveis. O comportamento da porta **E** segue extritamente a definição (e tabela verdade) dadas na seção 3.2.

(a) (b)

Figura 5. Símbolo da porta lógica **E** com 2 entradas (a) e com 3 entradas (b).

4.3 Inversor (ou Porta Inversora, ou Negador)

A porta que simboliza a operação **complementação** é conhecida como **inversor** (ou porta inversora, ou negador). Como a operação complementação só pode ser realizada sobre uma variável por vez (ou sobre o resultado de uma subexpressão), o inversor só possui uma entrada e, obviamente, uma saída. Caso se queira complementar uma expressão, é necessário obter-se primeiramente o seu resultado, para só então aplicar a complementação. O símbolo do inversor é mostrado na Figura 6.

Figura 6. Símbolo do **inversor** (também conhecido como negador ou porta inversora).

4.4 Exemplo de Circuito Lógico

Dada uma equação Booleana qualquer, é possível desenhar-se o circuito lógico que a implementa. O circuito lógico é composto das portas lógicas relacionadas às operações que são realizadas sobre as variáveis de entrada. Os resultados das operações são conduzidos por fios, os quais, no desenho, são representados por linhas simples.

Os passos a serem seguidos para se realizar o desenho do circuito lógico a partir de uma equação são praticamente os mesmos usados na avaliação da expressão. Tomemos como exemplo a equação $W = X + Y \cdot \overline{Z}$, avaliada na seção 3.6. Inicialmente, identificamos as variáveis independentes, que no caso são X, Y e Z. Para cada uma destas, traçamos uma linha (da esquerda para a direita), representando os fios que conduzem os valores. Feito isto, deve-se seguir desenhando as portas necessárias para representar cada uma das subexpressões, na mesma ordem tomada para a avaliação, ou seja:

1^o parêntesis (dos mais internos para os mais externos);
2^o operações E;
3^o operações OU.

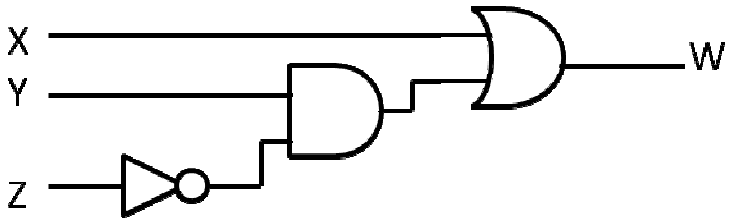
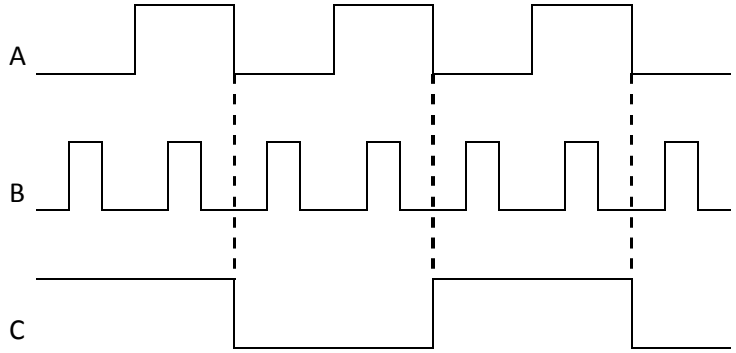


Figura 7. Circuito lógico para a equação $W = X + Y \cdot \overline{Z}$.

Exercícios propostos

4.1 Desenhe a forma de onda de saída para a porta OR da figura 2.1 abaixo



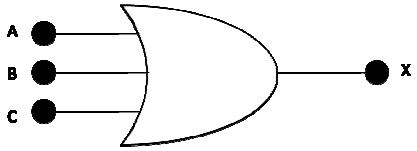


Figura 2.1

4.2 Suponha que a entrada A da figura 2.1 esteja ligada curto-circuitada com a linha de alimentação +5 V ou seja, A=1. Redesenhe a forma de onda da saída resultante.

4.3 Troque a porta OR da Figura 2.1 por uma porta AND e desenhe a saída x para esta nova situação.

4.4 Para cada uma das expressões a seguir, desenhe o circuito lógico correspondente usando portas AND, OR e INVERSORES.

a) $x = \overline{AB(C + D)}$

b) $z = \overline{(A + B + \bar{C}D\bar{E})} + \bar{B}C\bar{D}$

4.5 Escreva a expressão lógica que representa o circuito abaixo e utilize-a para escrever a tabela verdade da função que descreve o circuito. Em seguida, aplique as formas de ondas da figura 2.2.

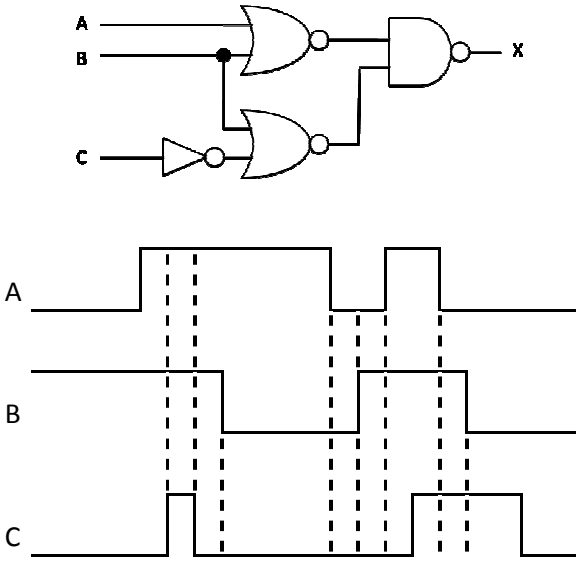
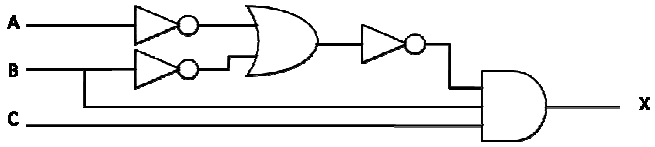


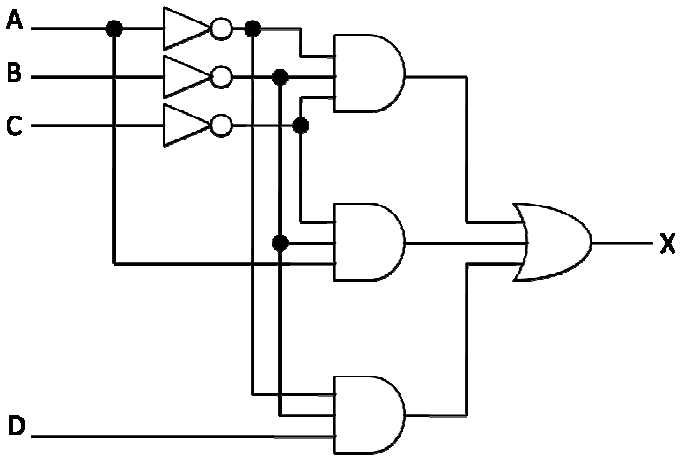
Figura 2.2

4.6 Escreva a expressão booleana para os circuitos abaixo e determine o valor de x para todas as condições possíveis de entrada, relacionando-os em uma tabela verdade.

a)



b)



5. Derivação de Expressões Booleanas

Dada uma função Booleana, descrita por sua tabela verdade, derivar uma expressão Booleana para esta função é encontrar uma equação que a descreva. Logo, a derivação de expressões Booleanas é o problema inverso da avaliação de uma expressão Booleana, descrito na seção 3.6.

Há basicamente duas maneiras de se definir (ou descrever) uma função Booleana: descrevendo-se todas as situações das variáveis de entrada para as quais a função vale **1** ou, alternativamente, todas as situações em que a função vale **0**. O primeiro método é conhecido por **soma de produtos** (SdP), enquanto que o segundo é chamado **produto de somas** (PdS).

Qualquer função Booleana pode ser descrita por meio de soma de produtos ou por meio de produto de somas. Como as funções Booleanas só podem assumir um dentre dois valores (0 ou 1), basta usar-se um dos dois métodos para se encontrar uma equação para uma função. A seguir, são detalhados os métodos de derivação de expressões Booleanas.

5.1 Derivação de Expressões usando Soma de Produtos (SdP)

Dada uma função Booleana de n variáveis (ou seja, n entradas), haverá 2^n combinações possíveis de valores. Dizemos que esse conjunto de valores que as variáveis podem assumir, juntamente com os respectivos valores da função, constituem o espaço da função. A cada combinação de entradas podemos associar um **termo produto,** no qual todas as variáveis da função estão presentes, e que é construído da seguinte forma: se a variável correspondente vale '**0'**, ela deve aparecer negada; se a variável vale '**1'**, ela deve aparecer não negada. Por exemplo, a Tabela 10 a seguir lista os termos produto associados a cada combinação de entradas para uma função Booleana de três variáveis A, B e C.

Tabela 10. **Tabela com os mintermos para 3 variáveis**

A	B	C	mintermo
0	0	0	$\overline{A} \cdot \overline{B} \cdot \overline{C}$
0	0	1	$\overline{A} \cdot \overline{B} \cdot C$
0	1	0	$\overline{A} \cdot B \cdot \overline{C}$
0	1	1	$\overline{A} \cdot B \cdot C$
1	0	0	$A \cdot \overline{B} \cdot \overline{C}$
1	0	1	$A \cdot \overline{B} \cdot C$
1	1	0	$A \cdot B \cdot \overline{C}$
1	1	1	$A \cdot B \cdot C$

Cada termo produto construído conforme a regra anteriormente descrita é denominado **mintermo** (ou minitermo). Note que, para um dado mintermo, se substituirmos os valores das variáveis associadas, obteremos 1. Porém, se substituirmos nesse mesmo mintermo quaisquer outras combinações de valores, obteremos 0. Dessa forma, se quisermos encontrar a equação para uma função a partir de sua tabela verdade, basta montarmos um **OU** entre os mintermos associados aos **1**s da função (também chamados **mintermos 1**).

Exemplo 5.1: encontrar a equação em soma de produtos (SdP) para a função F, descrita pela tabela verdade apresentada na Tabela 11:

Tabela 11. **Tabela verdade da função F**

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

Na tabela acima, F é função das variáveis A, B e C e os valores de (A,B,C) para os quais F=1 são (0,1,0), (0,1,1), (1,0,0) e (1,1,1). Logo, a equação em soma de produtos para F será o **OU** entre estes produtos, conforme segue:

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

A fim de simplificar a notação, o símbolo da operação **E** pode ser omitido. Desta forma, a equação anterior pode ser reescrita de maneira mais concisa:

$$F = \overline{A}\, B\, \overline{C} + \overline{A}\, B\, C + A\, \overline{B}\, \overline{C} + A\, B\, C$$

5.2 Derivação de Expressões usando Produto de Somas (PdS)

O método de derivação usando produto de somas é o **dual** (isto é, o oposto) do método de derivação em soma de produtos. A cada combinação das variáveis de entrada de uma função podemos associar um **termo soma,** no qual todas as variáveis da função estão presentes, e que é construído da seguinte forma: se a variável correspondente vale '**1'**, ela deve aparecer negada; se a variável vale '**0'**, ela deve aparecer não negada. A Tabela 12 a seguir lista os termos soma associados a cada combinação de entradas para uma função Booleana de três variáveis (A, B e C, por exemplo).

Tabela 12. **Tabela com os maxtermos para 3 variáveis**

A	B	C	Maxtermos
0	0	0	$A + B + C$
0	0	1	$A + B + \overline{C}$
0	1	0	$A + \overline{B} + C$
0	1	1	$A + \overline{B} + \overline{C}$
1	0	0	$\overline{A} + B + C$
1	0	1	$\overline{A} + B + \overline{C}$
1	1	0	$\overline{A} + \overline{B} + C$
1	1	1	$\overline{A} + \overline{B} + \overline{C}$

Cada termo soma construído conforme a regra anteriormente descrita é denominado **maxtermo** (ou maxitermo). Note que, para um dado maxtermo, se substituirmos os valores das variáveis associadas, obteremos 0. Porém, se substituirmos nesse mesmo maxtermo quaisquer outras combinações de valores, obteremos 1. Dessa forma, se quisermos encontrar a equação para uma função a partir de sua tabela verdade, basta montarmos um **E** entre os maxtermos associados aos **0**s da função (também chamados **maxtermos 0**).

Exemplo 5.2: encontrar a equação em produto de somas (PdS) para a função F, descrita pela tabela verdade abaixo (Tabela 13):

Tabela 13. **Tabela verdade da função F exemplo**

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

Foi escolhida a mesma função do exemplo anterior, para que se possa estabelecer comparações entre os dois métodos de derivação. Os valores das variáveis de entrada A,B e C para os quais F = 0 são (0,0,0), (0,0,1), (1,0,1) e (1,1,0). Logo, a equação em produto de somas para F será o **E** entre estas somas:

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

Note que a ordem de precedência de uma expressão em produto de somas é "primeiro cada soma deve ser avaliada, para só então avaliar-se o produto". Isto significa que os parêntesis em torno de cada termo soma **são obrigatórios**! Repare também que os símbolos referentes à operação **E** (entre os termos soma) podem ser omitidos.

$$F = (A + B + C)\ (A + B + \overline{C})\ (\overline{A} + B + \overline{C})\ (\overline{A} + \overline{B} + C)$$

5.3 Formas Canônicas, Padrão e Não-Padrão

As representações em soma de produtos e em produto de somas são denominadas formas **padrão**. A soma de produtos e o produto de somas descritos nas duas seções anteriores apresentam ainda uma característica bastante particular: em cada termo soma e em cada termo produto todas as variáveis da função estão presentes. Devido a essa característica, essas formas são chamadas **canônicas**. Neste curso adotaremos a forma de representação de mintermos como sendo a forma canônica adotada para representar as expressões.

Apesar da praticidade das representações canônicas, elas são pouco úteis para a implementação de circuitos digitais. O número de elementos (portas lógicas e conexões) de um circuito lógico depende diretamente do número de operações Booleanas (inversão, E e OU) contidas na expressão associada. Desta forma, é normal que se deseje reduzir o número de operações contidas numa função, de modo a poder-se implementá-la com circuitos lógicos mais simples, e portanto, de menor custo. A redução do número de operações é obtida mediante a eliminação de literais da expressão, aplicando-se as propriedades da álgebra Booleana descritas na seção 3.5. Um literal é uma variável negada ou uma variável não negada. O processo de redução de literais (ou de redução de operações, equivalentemente) é denominado **simplificação**.

Para exemplificar os passos básicos para a simplificação algébrica (literal) de expressões Booleanas, tomemos a expressão canônica, em soma de produtos, para a função F:

$$F = \overline{A}\,B\,\overline{C} + \overline{A}\,B\,C + A\,\overline{B}\,C + A\,B\,\overline{C} \quad (e1)$$

O primeiro passo é identificar pares de mintermos que se diferenciam por apenas um literal, a fim de aplicar a propriedade (14 apresentada na seção 3.5). Os mintermos ABC e AB$\overline{C}$, por exemplo, possuem os mesmos literais, exceto pela variável C: no primeiro, o literal é C, enquanto no segundo, o literal é $\overline{C}$. Então, com o uso da propriedade (14), pode-se fatorar esses dois mintermos, obtendo-se:

$$F = \overline{A}\,B\,(\overline{C} + C) + A\,\overline{B}\,C + A\,B\,\overline{C} \quad (e2)$$

Pela propriedade (4), tem-se que $C + \overline{C} = 1$. Então, obtém-se:

$$F = \overline{A}\,B \cdot (1) + A\,\overline{B}\,C + A\,B\,\overline{C} \quad (e3)$$

E pela propriedade (6), $\overline{A}\,B \cdot 1 = \overline{A}\,B$. Então, obtém-se:

$$F = \overline{A}\,B + A\,\overline{B}\,C + A\,B\,\overline{C} \quad (e4)$$

Assim, pela manipulação algébrica, obtivemos uma expressão em soma de produtos que é simplificada em relação a sua expressão em soma de produtos na forma canônica, pois o número de operações e também de literais foram reduzidos.

Entretanto, na equação (e1), o mintermo $\overline{A}B\overline{C}$ também poderia ter sido agrupado com o mintermo $AB\overline{C}$, pois ambos possuem os mesmos literais, exceto pela variável A ($\overline{A}$ no primeiro e A no segundo). Naturalmente, os passos a serem seguidos seriam os mesmos descritos anteriormente. E a equação resultante seria um pouco diferente, mas com o mesmo número de operações sendo, portanto, de mesma complexidade. Na verdade, o melhor seria se pudéssemos agrupar o mintermo $\overline{A}B\overline{C}$ com o mintermo $\overline{A}BC$ e ao mesmo tempo com o mintermo $AB\overline{C}$. Felizmente, a propriedade (3) da álgebra Booleana diz que o **OU** entre duas ou mais variáveis Booleanas iguais é igual a própria variável Booleana em questão.

Estendendo esta propriedade, pode-se dizer que o **OU** entre duas ou mais funções (inclusive produtos) Booleanas iguais equivale à própria função Booleana em questão. Desta forma, pode-se expandir o mintermo $\overline{A}B\overline{C}$ para:

$$\overline{A}B\overline{C} = \overline{A}B\overline{C} + \overline{A}B\overline{C} \quad \text{(e5)}$$

Retomando a equação (e1) e utilizando (e5) , segue que:

$$F = \overline{A}\,B\,\overline{C} + \overline{A}\,B\,C + A\,\overline{B}\,C + A\,B\,\overline{C} + \overline{A}\,B\,\overline{C} \quad \text{(e6)}$$

Então, a propriedade (3) garante que as expressões (e1) e (e6) são equivalentes, embora o mintermo $\overline{A}B\overline{C}$ apareça duplicado. E pelo fato de aparecer duas vezes, pode-se usar uma cópia de $\overline{A}B\overline{C}$ para simplificar com $\overline{A}BC$ e outra para simplificar com $AB\overline{C}$. Os passos da simplificação são os mesmos já descritos: pela propriedade (14), segue:

$$F = \overline{A}\,B \cdot (\overline{C} + C) + A\,\overline{B}\,C + (\overline{A} + A) \cdot B\,\overline{C} \quad \text{(e7)}$$

E pela propriedade (6), vem:

$$F = \overline{A}\,B \cdot (1) + A\,\overline{B}\,C + (1) \cdot B\,\overline{C} \quad \text{(e8)}$$

Finalmente, pela propriedade (4), tem-se:

$$F = \overline{A}\,B + A\,\overline{B}\,C + B\,\overline{C} \quad \text{(e9)}$$

Repare que o mintermo $A\overline{B}C$ não pôde ser agrupado com nenhum outro mintermo. Note também que foram feitas todas as simplificações possíveis, uma vez que foram agrupados e simplificados todos os pares de mintermos que se diferenciam de somente uma variável. Logo, a expressão (e9) representa a máxima simplificação possível sob a forma de soma de produtos. E por esse motivo, ela é dita equação **mínima** em **soma de produtos** da função F. Quanto a expressão (e4), diz-se ser uma equação em **soma de produtos simplificada** (porém, não-mínima). Logo, toda equação mínima é simplificada, porém, nem toda equação que foi simplificada é necessariamente mínima.

Embora a equação mínima em soma de produtos apresente menor número de operações Booleanas que a representação na forma canônica, as vezes pode ser possível reduzir-se ainda mais o número de operações, fatorando-se literais. Por exemplo, na expressão (e9) pode-se fatorar o primeiro e o terceiro mintermos como segue:

$$F = B \cdot (A + \overline{C}) + A\,\overline{B}\,C \quad \text{(e10)}$$

A expressão (e10), obtida pela fatoração de (e9), não é nem do tipo soma de produtos, nem produto de somas, pois há um termo que não é nem produto, nem soma. Diz-se que a expressão está na **forma fatorada**. No caso de (e10), a fatoração não resultou em redução do número de operações.

No que se refere a terminologia, as formas soma de produtos e produto de somas são ditas **formas padrão** (formas *standard*). A forma fatorada é dita **não-padrão**. As formas **canônicas** são, pois, casos especiais de formas padrão, nas quais os termos são mintermos ou maxtermos. A fim de diferenciar somas de produtos canônicas de somas de produtos simplificadas, usaremos a expressão "**soma de mintermos"**. De maneira similar, usaremos a expressão "**produto de maxtermos"** para diferenciar produtos de somas canônicos de produtos de somas simplificados.

5.4 Circuitos Lógicos para Formas Padrão e Não-Padrão

As regras gerais para se realizar o desenho de circuitos lógicos já foram apresentadas na seção 4.4. No caso de equações na forma soma de produtos (canônica ou simplificada), há um primeiro nível (desconsiderando-se possíveis inversores), constituído somente por portas **E**, onde cada porta **E** implementa um dos produtos da equação. Há ainda um segundo nível, constituído por uma porta **OU**, responsável pela "soma" lógica dos produtos. A figura abaixo (Figura 8) mostra um possível circuito lógico para a equação (e1).

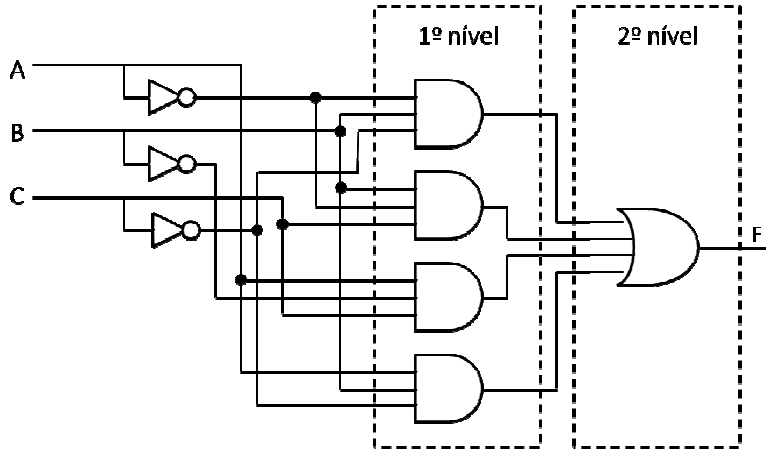


Figura 8. Possível circuito lógico que implementa equação (e1).

A fim de facilitar a compreensão do circuito acima, as seguintes regras devem ser observadas:

- as variáveis de entrada devem ser identificadas preferencialmente à esquerda, junto aos respectivos fios;
- inversores devem ser providos para as variáveis que aparecem negadas na equação;
- as portas que implementam as operações Booleanas que aparecem na equação normalmente são posicionadas da esquerda para a direita, seguindo a ordem de avaliação dos operadores.

Repare que em todas as interseções de fios em que há conexão física, **deve haver um ponto** (suficientemente grande), como se fora uma "solda". Logo, quando não há o referido ponto na interseção de fios, significa que tais fios estão "eletricamente isolados".

O circuito da equação (e1) ainda pode ser desenhado utilizando-se uma notação simplificada para os inversores das entradas. Ao invés de se desenhar um inversor para cada variável que aparece negada na equação, coloca-se um círculo junto a cada entrada de cada porta na qual há uma variável negada. A aplicação desse procedimento para o circuito da equação (e1) resulta no circuito apresentado na Figura 9:

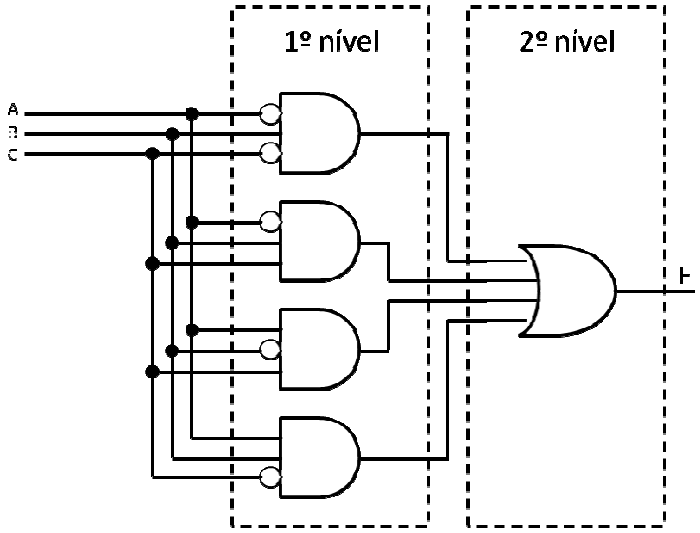


Figura 9. Um circuito lógico para soma de produtos.

No caso de equações na forma produto de somas (canônica ou simplificada), o primeiro nível é constituído por portas **OU**, sendo cada uma responsável por uma das "somas" lógicas da equação. O segundo nível, por sua vez, é constituído por uma porta **E**, que realiza o produto lógico das parcelas. A Figura 10 mostra um possível circuito lógico para a equação (e1).

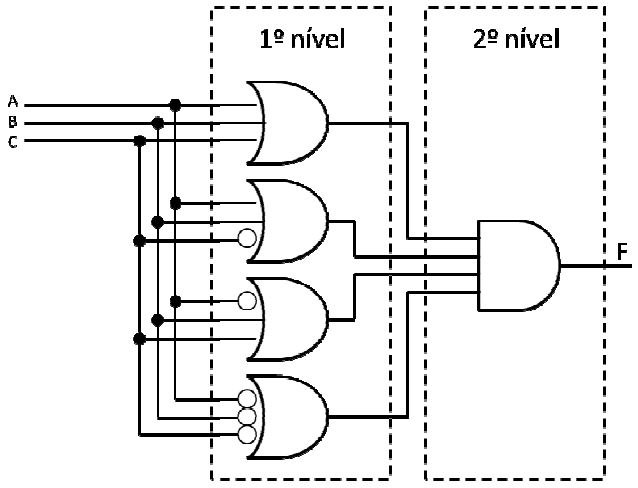


Figura 10. Um circuito lógico para produto de somas.

Pelo fato de apresentarem apenas dois níveis de portas (dois níveis lógicos), circuitos para equações representadas nas formas padrão, canônicas ou simplificadas, são ditos **circuitos em dois níveis** (ou **lógica a dois níveis**).

A Figura 11 mostra o circuito lógico para a equação (e9), que é a **forma mínima** para a função da equação (e1). Note que este circuito é de menor complexidade que o circuito da Figura 9 e Figura 10.

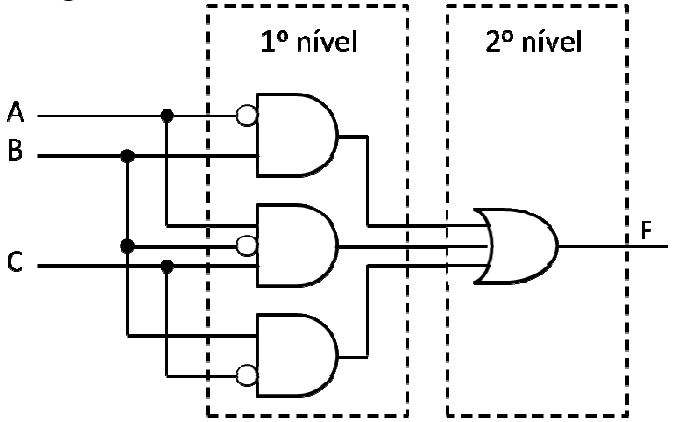


Figura 11. Circuito lógico para a equação (e9).

Dada uma equação canônica de uma função qualquer, o circuito para uma equação simplificada a partir da canônica possui menos portas e/ou portas de menor complexidade. (A complexidade relativa de uma porta lógica pode ser medida pelo número de entradas que ela possui). A complexidade relativa de um circuito lógico pode ser calculada somando-se o número de entradas das portas do circuito. No circuito da Figura 9 e da Figura 10 há 4 portas de 3 entradas e 1 porta de 4 entradas. Então, a complexidade relativa será 4x3+1x4=16. No circuito da figura 2.8 há 2 portas de 2 entradas e 2 portas de 3 entradas. Sua complexidade relativa será 2x2+2x3=10. Claramente, o circuito da Figura 11 é de menor complexidade que os circuitos da Figura 9 e da Figura 10. Estes dois são de mesma complexidade relativa. No cálculo da complexidade relativa, as inversões normalmente não são levadas em conta. Circuitos para formas fatoradas podem ser vistos como o caso mais genérico. Em geral, as formas fatoradas conduzem a circuitos cujo número de níveis lógicos é maior do que dois. Por isso, circuitos lógicos para formas fatoradas são denominados **circuitos multinível** (**lógica multinível**). Como dito na seção 5.4, as vezes uma forma fatorada pode apresentar menor número de operações do que a respectiva forma padrão. Quando isso ocorre, o circuito associado à forma fatorada também será de menor complexidade relativa. Entretanto, se não ocorrer redução no número de operações, mesmo assim é possível que o circuito para a forma fatorada seja de menor complexidade relativa, pois o conceito de complexidade relativa também inclui o número de entradas de cada porta. Então, a maneira mais segura de saber se o circuito associado à forma fatorada é de menor complexidade ou não é desenhá-lo e somar o número de entradas. A Figura 12 mostra o circuito para a equação 2.21, obtida a partir da equação (e9) fatorando-se o literal B.

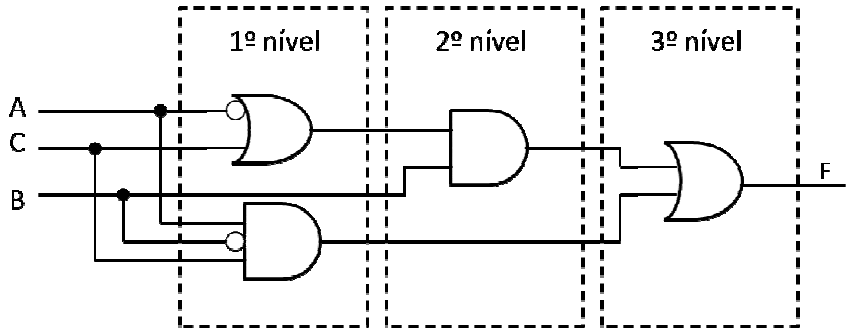


Figura 12. Circuito lógico multinível, associado à equação (e10), a qual está na forma fatorada.

Note que o número de operações Booleanas destas equações é o mesmo: 4. No entanto, a complexidade do circuito da forma fatorada é 3x2+1x3=9, portanto menor do que a complexidade do circuito da Figura 11.

5.1 Simplifique a seguinte expressão

$X = (M + N)(\overline{M} + P)(\overline{N} + \overline{P})$

5.2 Simplifique as seguintes expressões utilizando álgebra booleana.

a) $x = ABC + \overline{A}C$
b) $y = (Q + R)(\overline{Q} + \overline{R})$
c) $w = ABC + A\overline{B}C + \overline{A}$
d) $q = \overline{RST}\ \overline{(R + S + T)}$
e) $x = \overline{A}\overline{B}\overline{C} + \overline{A}BC + ABC + A\overline{B}\overline{C} + A\overline{B}C$
f) $z = (B + \overline{C})(\overline{B} + C) + \overline{A} + B + \overline{C}$
g) $y = \overline{(C + D)} + \overline{A}C\overline{D} + A\overline{B}\overline{C} + \overline{A}\overline{B}CD + AC\overline{D}$
h) $x = AB(CD) + ABD + BCD$

6. Simplificação de Funções Booleanas usando Mapas de Karnaugh

O método de simplificação apresentado na seção 3.7 é de aplicabilidade limitada, uma vez que é bastante difícil certificar-se que todos os pares de mintermos que podiam ser simplificados foram determinados. Alternativamente àquele método, há outro método de simplificação baseado na identificação visual de grupos de mintermos passíveis de serem simplificados. No entanto, para que se possa identificar tais grupos, é necessário que os mintermos sejam dispostos de maneira mais conveniente, ou seja, de forma que os mintermos vizinhos possuam apenas 1 termo que se modifica. A esta estrutura de organização dos mintermos vizinhos damos o nome de Mapa de Karnaugh (criados por Edward Veitch (1952) e aperfeiçoados pelo engenheiro de telecomunicações Maurice Karnaugh). Nas figuras abaixo são apresentadas as reorganizações dos mintermos para 2, 3 e 4 variáveis (Figura 13 Figura 14 e Figura 15 respectivamente) de forma que cada mintermo difere apenas em 1 termo de seus vizinhos.

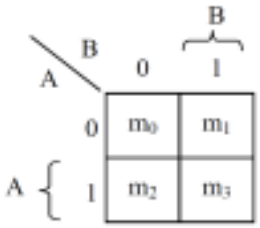


Figura 13. Mapa de Karnaugh de 2 variáveis.

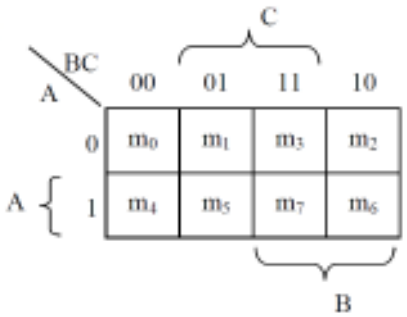


Figura 14. Mapa de Karnaugh de 3 variáveis.

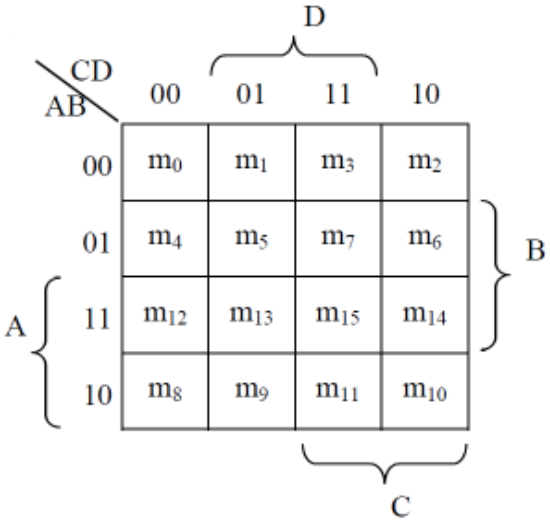


Figura 15. Mapa de Karnaugh de 4 variáveis.

A partir do mapa de Karnaugh de uma função, é possível realizar a simplificação da função de forma mais fácil e direta do que se fossemos utilizar as leis e simplificações da álgebra booleana. O exemplo abaixo ilustra a simplificação de uma função de 3 variáveis utilizando o mapa de Karnaugh.

Exemplo 6.1: Dada a tabela verdade com os mintermos da função F abaixo (Tabela 14), monte o mapa de Karnaugh e encontre a equação simplificada.

Tabela 14. **Tabela Verdade exemplo**

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

O primeiro passo para simplificar-se uma função usando mapa de Karnaugh é escolher o mapa conforme o número de variáveis da função e preencher os valores dos mintermos conforme a tabela verdade fornecida, ou conforme a equação fornecida. O segundo passo é identificar grupos de mintermos adjacentes que formem grupos de 2^m elementos adjacentes entre si, com 'm' podendo valer no máximo o número de variáveis da função. Estes grupos são denominados **subcubos**. No caso de se querer encontrar uma expressão em soma de produtos, estaremos interessados nos subcubos de mintermos-1. Então, cada subcubo contendo mintermos-1 irá originar um produto, no qual uma ou mais variáveis poderão estar ausentes devido à simplificação que é obtida. Os produtos associados aos subcubos de mintermos-1, simplificados ou

não, são denominados **implicantes**. É importante ressaltar que quanto maior o número de elementos do subcubo, maior será a simplificação obtida.

Ao mapearmos os mintermos da tabela verdade acima (Tabela 14) e agruparmos os termos vizinhos, obtemos o mapa de Karnaugh de 3 variáveis da Figura 16.

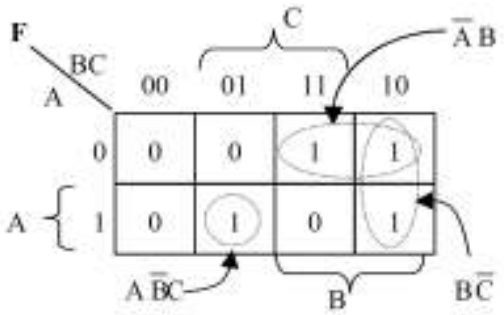


Figura 16. Tabela de adjacências para uma função de 3 variáveis.

Repare que nessa nova tabela, quaisquer dois mintermos adjacentes (na horizontal ou na vertical) são passíveis de serem simplificados, pois só se diferenciam de uma variável. É importante ressaltar que esse conceito de adjacência **não está restrito aos limites da tabela**, uma vez que os elementos extremos de uma mesma linha (ou de uma mesma coluna) também são simplificáveis. Isto implica que a tabela de adjacências de mintermos da Figura 16 pode e deve ser encarada como uma figura geométrica tridimensional do tipo "toróide" (ou uma "rosquinha").

Após, deve-se identificar todos os grupos de mintermos-1 adjacentes entre si. Cada grupo de mintermos-1 originará um produto, conforme indicado na Figura 16. A equação em soma de produtos simplificada será o **OU** entre os produtos encontrados

$$F = \overline{A}\,B + A\,\overline{B}\,C + B\,\overline{C}$$

É importante ressaltar que o conceito de adjacência é aplicável na horizontal e na vertical, **mas nunca na diagonal**.

No caso de se querer encontrar uma expressão em produtos de somas, estaremos interessados nos subcubos de mintermos-0. Então, cada subcubo contendo mintermos-0 irá originar uma soma, no qual uma ou mais variáveis poderão estar ausentes devido à simplificação que é obtida. As somas associados aos subcubos de mintermos-0, simplificadas ou não, são denominadas **implicados**. Também neste caso, quanto maior o número de elementos do subcubo, maior será a simplificação obtida.

6.1 Cobertura dos Mapas de Karnaugh

Normalmente, é possível identificar-se numa mesma função Booleana mais de um implicante (ou mais de um implicado). Neste caso, é necessário determinar o conjunto de implicantes (ou implicados) que melhor "cobre" a função, onde a melhor

cobertura significa necessariamente a expressão mais simplificada possível, a qual é denominada expressão mínima.

O procedimento básico para se determinar a melhor cobertura (também chamada cobertura mínima) para uma expressão em soma de produtos é o seguinte:

1. Identificar os subcubos de mintermos-1 com maior número de elementos possível, iniciando do tamanho 2^n, onde n é o número de variáveis da função. Caso algum mintermo-1 fique isolado (isto é, não há nenhum outro mintermo-1 adjacente a ele), então ele constituirá um subcubo de um elemento;

2. Identificar o menor conjunto de subcubos de modo que cada mintermo-1 pertença apelo menos um subcubo (seja coberto pelo menos uma vez).

Observações:

a. Cada mintermo-1 pode ser coberto por mais de um subcubo, caso isso resulte numa simplificação maior;

b. Um último teste para verificar se a expressão obtida é realmente a mínima consiste em verificar se algum subcubo pode ser removido, sem deixar algum mintermo-1 descoberto. Um subcubo que poder ser removido sem descobrir mintermos é dito subcubo **não-essencial**. Logo, todo o subcubo que não pode ser removido é dito **essencial**;

c. Pode haver mais de uma expressão mínima para uma mesma função Booleana;

d. A expressão mínima é aquela de menor complexidade. E a complexidade será medida pelo número de literais de uma função.

Conforme já mencionado anteriormente, também é possível obter-se uma expressão mínima em **produto de somas** a partir do mapa de Karnaugh da função Booleana. Para tanto, deve-se identificar os subcubos de mintermos-0, ao invés de subcubos de mintermos-1. Cada subcubo de mintermo-0 irá originar um termo soma, possivelmente já simplificado. Os passos para a obtenção de uma cobertura mínima são os mesmos já descritos para a obtenção da expressão em soma de produtos.

Exercicios Propostos

6.1 Implemente o mapa k das expressões do exercício 5.2 do capítulo anterior e confira se a sua resposta ficou igual ao encontrado na simplificação booleana.

7. Meio Somador e Somador Completo

A operação aritmética binária mais simples é a adição de dois dígitos binários (bits), a qual pode ser vista como a adição de dois números binários de um bit cada. Considerando-se todas as 4 combinações de valores que podem ocorrer, os resultados possíveis dessa adição são:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10$$

Repare que no último caso acima, o resultado da adição é o valor 2, que em binário necessita de dois dígitos para ser representado (10). Um circuito aritmético para realizar a operação de adição, deve operar corretamente para qualquer combinação de valores de entrada.

Isso significa que o circuito para a adição de dois bits deve possuir duas entradas e duas saídas, para poder representar o máximo valor possível de ser obtido em uma soma de 2 dígitos de 1 bit. A este componente que tem a capacidade de somar 2 bits, damos o nome de meio somador. A Figura 17 abaixo apresenta o desenho representativo de um meio somador.

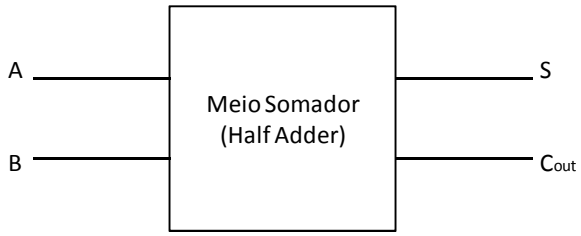


Figura 17. Meio Somador

Denomina-se **meia-soma** a operação de adição de dois bits. O circuito mostrado na Figura 17 é denominado **meio somador** (half adder, em inglês). As duas entradas, A e B, representam os dois bits a serem adicionados. A saída S representa o dígito menos significativo do resultado, enquanto que a saída Cout representa o dígito mais significativo do resultado, o qual também é conhecido por transporte de saída (carry out, em inglês), uma vez que ele assume valor 1 somente quando o resultado da soma de A e B não pode ser representado num único dígito.

A fim de se projetar o circuito do meio somador, devemos montar uma tabela verdade (Tabela 15) para as saídas S e Cout utilizando-se os valores que resultam da adição de dois dígitos binários, como segue:

Tabela 15. **Tabela Verdade de um Meio Somador**

A	B	Cout	S
0	0	0	0
0	1	0	1

1	0	0	1
1	1	1	0

Felizmente para facilitar a implementação do circuito que faz a soma de 2 entradas binárias, existe uma porta lógica que implementa esta função de soma chamada de Lógica OU EXCLUSIVA que em inglês é chamada de forma simplificada de XOR e tem como '⊕' o símbolo de operador lógico. A Tabela verdade da porta XOR é igual a saída S da Tabela 15. Portanto, a saída S nada mais é do que o XOR entre A e B ($S = \overline{A}.B + A.\overline{B}$ = A ⊕ B). Já a saída Cout é o E entre A e B (Cout = A ⋅ B). Então, um circuito para o meio somador usa apenas uma porta XOR de duas entradas e uma porta E de duas entradas, conforme mostrado na Figura 18.

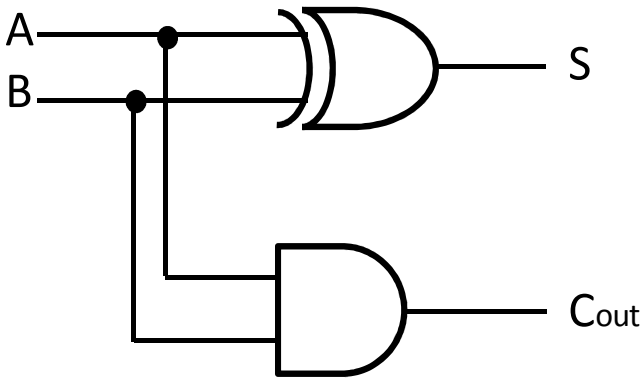


Figura 18. Circuito que implementa o meio somador (*half adder* ou HAD).

Entretanto, ao somarmos dois números binários que possuem mais de um dígito cada ocorrer transporte diferente de zero para a soma de um par de dígitos intermediários, a soma do par seguinte deverá considerar esse transporte proveniente do par anterior, conforme ilustra o exemplo a seguir (Figura 19).

```
transporte    1101#
A               1101
B             + 0101
              ------
resultado     10010
```

Figura 19. Exemplo de uma operação de soma de 4 bits.

O exemplo mostrado na Figura 19 ilustra bem o fato de, para cada posição exceto a menos significativa, o resultado é obtido mediante a adição de três bits: um pertencente ao número A, um pertencente ao número B e um terceiro que é o transporte proveniente do resultado da adição entre os bits da posição anterior. Chamamos este terceiro bit, ou seja, o bit de transporte de bit de "carry", o que quer dizer transporte em inglês, e o abreviamos como Cin.

O circuito capaz de realizar a soma de três bits (A, B e Cin), gerando o resultado em dois bits (S e Cout) é denominado somador completo (full adder, em inglês).

Apesar da entrada Cin normalmente receber o transporte proveniente da soma imediantamente anterior (carry in, em inglês), a rigor as três entradas são absolutamente equivalentes sob o ponto de vista funcional, ou seja, em um circuito somador completo, tanto faz se a entrada A for ligada a entrada do Cin e vice-versa. A tabela verdade para a soma completa é mostrada na 0, juntamente com o mapa de Karnaugh e as equações mínimas resultantes para S e Cout. A figura 3.9 mostra um circuito para o somador completo.

Tabela 16. **Tabela Verdade de um somador Completo**

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

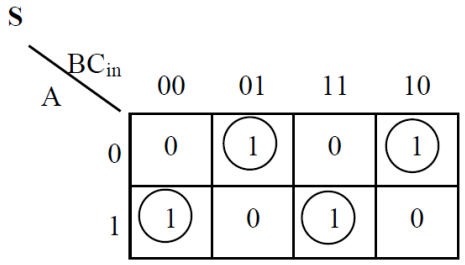


Figura 20. Mapa de Karnaugh da saída soma

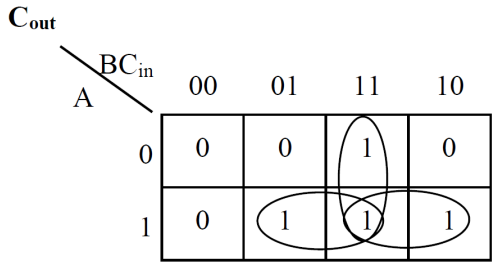


Figura 21. Mapa de Karnaugh da saída Carry out

Conforme pode-se ver pelo mapa de Karnaugh acima (Figura 20), a expressão mínima em soma de produtos para S contém todos os mintermos da função:

$$S = \overline{A} \cdot \overline{B} \cdot Cin + \overline{A} \cdot B \cdot \overline{Cin} + A \cdot \overline{B} \cdot \overline{Cin} + A \cdot B \cdot Cin$$

O circuito que implementa a saída S do somador completo pode ser derivado a partir da equação em soma de produtos acima. No entanto, pode-se ainda manipular tal equação conforme segue:

$$S = \bar{A} \cdot (\bar{B}.Cin + B.\overline{Cin}) + A \cdot (\bar{B}.\overline{Cin} + B.Cin)$$
$$S = \bar{A} \cdot (B \oplus Cin) + A \cdot \left(\overline{B \oplus Cin}\right)$$
$$S = (A \oplus B \oplus Cin)$$

Logo, o circuito para a saída S do somador completo pode também ser representado com duas portas XOR, conforme mostra a Figura 22.

A saída Cout após ser simplificada com o mapa de karnaugh da Figura 21, tem como expressão mínima em soma de produtos a equação apresentada abaixo e ilustrava na figura Figura 22:

$$C_{out} = A \cdot B + A \cdot C_{in} + B \cdot C_{in}$$

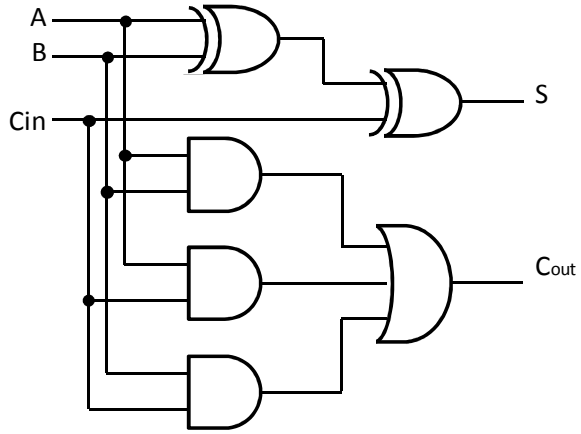


Figura 22. Circuito para o somador completo (*full adder* ou FAD).

O somador tipo propagador de carry (*ripple carry)*

Utilizando-se n somadores completos como o da Figura 22, pode-se realizar um somador capaz de operar dois números binários de n bits. Particularmente, o dígito de ordem i do resultado, Si, será obtido pela adição de Ai, Bi e Ci, onde Ci é o transporte proveniente do dígito anterior. O somador de índice i recebe como entradas Ai, Bi e Ci, gerando a soma Si e o valor de transporte Ci+1, o qual será entrada para o somador completo do dígito seguinte (i+1). A Figura 23 mostra um circuito somador paralelo para números binários com 4 bits.

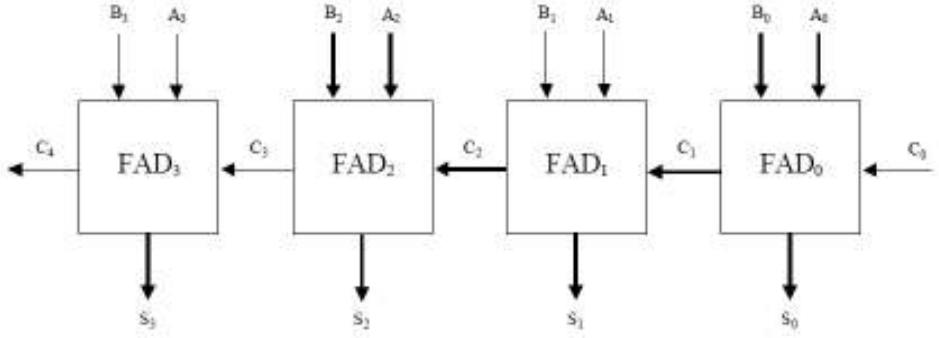


Figura 23. Somador do tipo Ripple Carry

Descrição do Trabalho Prático – Montagem de um somador tipo *ripple carry* de 4 bits

Implemente através da montagem em proto-board o circuito de um somador tipo ripple carry de 4 bits como o circuito apresentado na Figura 23. Use como entrada chaves do tipo Dipswitch e como saída leds e circuitos apenas com portas AND, OR, XOR e suas respectivas versões com saídas invertidas, NAND, NOR e XNOR. Apresente um relatório documentando todos os passos realizados para implementar o circuito. Apresente neste relatório todos os dados relativos aos componentes utilizados na montagem do circuito. Apresente também figuras que ilustrem todo o projeto realizado e também que ilustrem a montagem feita em proto-board. Para fazer estas figuras, você pode utilizar os desenhos dos CIs que aparecem nos respectivos Datasheets. Faça a figura que mostra o planejamento da montagem das ligações entre os CIs.

Data de entrega:

8. Codificadores e Decodificadores

Neste capítulo estudaremos alguns circuitos lógicos combinacionais destinados a uma função específica, que é a conversão entre os diversos códigos existentes.

São vários os códigos ou formas de codificar a informação digital dentro do campo da eletrônica. A criação de várias formas diferentes de codificação surgiu pois existem situações em que a utilização de um determinado código é mais vantajosa em relação a outro. Neste capítulo vamos descrever os códigos mais conhecidos de uma maneira rápida e nos deteremos especificamente em dois dos considerados os mais importantes. Após estudaremos seus respectivos circuitos codificadores e decodificadores finalizando com a representação decimal em display de 7 segmentos.

8.1 Códigos Numéricos

Códigos numéricos são códigos que trabalham unicamente com valores numéricos em sua composição.

Código 9876543210: é um código binário que converte cada dígito decimal em um conjunto de 10 bits, onde o valor 1 assume a posição correspondente ao número decimal, e o restante é completado com o valor 0. Foi bastante utilizado quando os sistemas mostradores de algarismos eram válvulas eletrônicas. Notamos no código, que em 10 saidas somente uma vale 1 em cada caso, acendendo assim o algarismo correspondente. A formação deste código é vista na Tabela 17 abaixo.

Tabela 17. **Tabela que relaciona números decimais de 1 dígito com o correspondente no código 9876543210**

Decimal	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	0	1	0	0
3	0	0	0	0	0	0	1	0	0	0
4	0	0	0	0	0	1	0	0	0	0
5	0	0	0	0	1	0	0	0	0	0
6	0	0	0	1	0	0	0	0	0	0
7	0	0	1	0	0	0	0	0	0	0
8	0	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0

Código BCD 8421: (de *Binary-coded decimal* 8421) é um sistema de codificação de números decimais em binários de quatro bits. Os valores 8421 são respectivamente os valores de 2 elevado ao valor de sua posição (3,2,1,0). Este código assume apenas 10 dígitos, variando de 0 a 9.

Tabela 18. **Tabela que relaciona números decimais de 1 dígito com o correspondente no código BCD 8421**

Decimal	2^3 (8)	2^2 (4)	2^1 (2)	2^0 (1)
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Código Excesso-3 :(XS-3) é um código binário decimal, chamado também de código de Excesso-N, que segue o mesmo príncipio de conversão do número decimal para binário, incrementando 3 unidades ao resultado binário.

Tabela 19. **Tabela que relaciona números decimais de 1 dígito com o correspondente no código Excesso-3**

Decimal	Binário	Decimal	Binário
0	0011	**9**	1100
1	0100	**8**	1011
2	0101	**7**	1010
3	0110	**6**	1001
4	0111	**5**	1000

Códigos BCD de 4 bits: existem diversos códigos BCDs que assumem valores diferentes de acordo com alguma variação em seu cálculo. Entre eles podemos destacar o BCD 7421, BCD 2421 e o BCD 5211.

Tabela 20. **Tabela que relaciona números decimais de 1 dígito com os correspondentes nos códigos BCD 8421, BCD 2421, BCD 5211**

Decimal	BCD 8 4 2 1	Excesso-3 ou Código de Stibitz	BCD 2 4 2 1 Código Aiken	BCD 5 2 1 1
0	0000	0011	0000	0000
1	0001	0100	0001	0001
2	0010	0101	0010	0011
3	0011	0110	0011	0101
4	0100	0111	0100	0111
5	0101	1000	1011	1000
6	0110	1001	1100	1010
7	0111	1010	1101	1100
8	1000	1011	1110	1110
9	1001	1100	1111	1111

Código Gray: é um sistema de código binário onde de um número para outro apenas um bit varia. Este sistema de codificação surgiu quando os circuitos lógicos digitais se realizavam com válvulas termoiônicas e dispositivos eletromecânicos. Os contadores necessitavam de potências muito elevadas e geravam ruído quando vários bits modificavam-se simultâneamente. O uso do código Gray garantiu que qualquer mudança variaria apenas um bit.

Atualmente o código Gray é utilizado em sistemas sequênciais mediante o uso dos Mapas de Karnaugh, já que o príncipio do desenho de buscar transições mais simples e rápidas segue vigente, apesar de que os problemas de ruído e potência tenham sido reduzidos.

Tabela 21. **Tabela que relaciona números decimais de 1 dígito com o correspondente no código GRAY**

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

8.2 Códigos de 5 bits:

Código 2 entre 5: possui sempre dois bits iguais a 1 dentro de seus bits.

Tabela 22. **Correspondência entre código decimal de 1 dígito e o código 2entre5**
Código 2 entre 5

Dígito decimal	Código 2entre5	Dígito decimal	Código 2entre5
0	0 0 0 1 1	5	0 1 1 0 0
1	0 0 1 0 1	6	1 0 0 0 1
2	0 0 1 1 0	7	1 0 0 1 0
3	0 1 0 0 1	8	1 0 1 0 0
4	0 1 0 1 0	9	1 1 0 0 0

Código Johnson: (Johnson-Mobius) é um código especial utilizado na construção do Contador de Johnson. Este código constitui-se em um código binário e cíclico (como o código Gray) cuja capacidade de codificação é dada por 2n, sendo *n* o número de bits. Para codificar os dígitos decimais são necessários 5 bits:

Tabela 23. **Correspondência entre código decimal de 1 dígito e o código Johnson**
Código Johnson

Dígito	Código	Dígito decimal	Código Johnson

decimal	**Johnson**		
0	00000	5	11111
1	00001	6	11110
2	00011	7	11100
3	00111	8	11000
4	01111	9	10000

Este código permite a simplicidade de criação de contadores, e por isto é utilizado em sistemas digitais de alta velocidade. Proporciona uma maior proteção contra erros mas é menos eficiente em memória do que o código binário decimal.

8.3 Códigos Alfanuméricos

Códigos alfanuméricos são códigos que representam todo o tipo de caracteres imprimíveis, numeros e letras além de caracteres de controle na sua composição.

Código ASCII: o "American Standard Code for Information Interchange" comumente referido como ASCII – também chamado ASCII completo, ou ASCII estendido –, é uma forma especial de código binário que é largamente utilizado em microprocessadores e equipamentos de comunicação de dados.

Um novo nome para este código que está se tornando popular é "American National Standard Code for Information Interchange" (ANSCII). Entretanto, utilizaremos o termo consagrado, ASCII. É um código binário que usado em transferência de dados entre microprocessadores e seus dispositivos periféricos, e em comunicação de dados por rádio e telefone. Com 7 bits pode-se representar um total de 27 = 128 caracteres diferentes. Estes caracteres compreendem números decimais de 0 até 9, letras maiúsculas e minúsculas do alfabeto, mais alguns outros caracteres especiais usados para pontuação e controle de dados.

Tabela 24. **Tabela que relaciona números decimais de 1 dígito com o correspondente no código ASCII**

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

8.4 Circuitos Decodificadores

8.4.1 Decodificadores BCD para Decimal

O código BCD expressa cada dígito de um número decimal por uma palavra binária de 4 bits (Nibble) no formato b3 b2 b1 b0 através da relação:

NúmeroDecimal = $b3 \cdot 2^3 + b2 \cdot 2^2 + b1 \cdot 2^1 + b0 \cdot 2^0$. A Tabela 25 mostra o resultado desta relação.

Tabela 25. **Tabela que relaciona números decimais de 1 dígito com o correspondente no código BCD 8421 ou simplesmente BCD**

Nibble				Número Decimal
b_3	b_2	b_1	b_0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

Por exemplo, o número decimal 8963 codificado em BCD resulta em:

8	9	6	3
↓	↓	↓	↓
1000	1001	0110	0011

A Figura 24 abaixo representa o diagrama interno de um Decodificador BCD-para-Decimal. Este decodificador é conhecido como decodificador 1-de-10, porque para cada Nibble ABCD na entrada do decodificador, somente uma das 10 saídas está em nível lógico 1. Por exemplo, para ABCD = 0011 temos para as saídas:

Y3 = 1 e todas as demais saídas Yk = 0 , com k ≠ 3 . Note que o subscrito da saída cujo nível lógico é 1 corresponde ao valor decimal do Nibble codificado em BCD nas entradas ABCD .

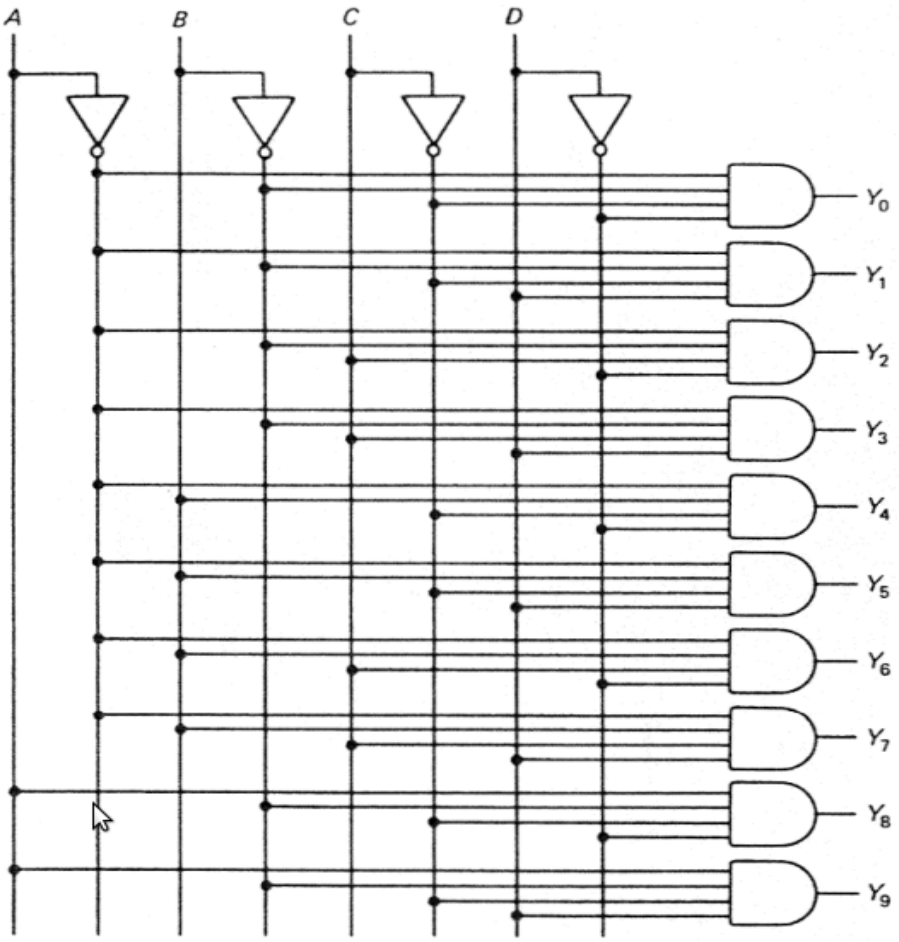


Figura 24. Decodificador BCD para Decimal

8.4.2 Codificadores Decimal para BCD

A Figura 25 abaixo representa o diagrama interno de um Codificador Decimal-para-BCD. As chaves são do tipo pushbutton (como no teclado de um computador). Por exemplo, quando o pushbutton 3 é pressionado as portas OR cujas saídas são C e D têm entradas cujo nível lógico é 1, resultando ABCD = 0011 .

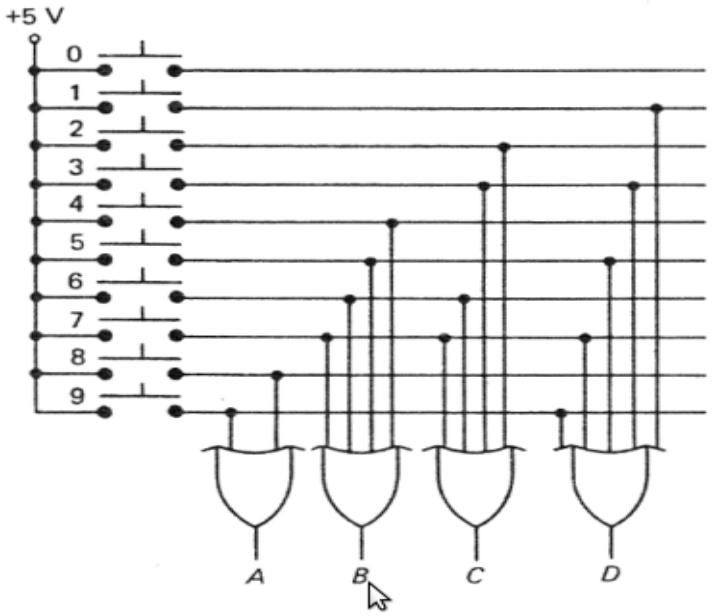


Figura 25. Codificador Decimal para BCD

8.4.3 Decodificador Gray para Binário

Como já foi visto o código Gray é um código digital com a propriedade de que duas palavras-código consecutivas diferem apenas de um bit.

O Código Gray é um código que se enquadra na classe de “Códigos Refletidos”, enquadramento devido ao algoritmo de construção do mesmo. Por exemplo, a tabela abaixo mostra a construção por quantificação-reflexão do Código

Exemplo de construção do Gray para 4 bits:

Representando o nibble do Código Gray por G3 G2 G1 G0 e o nibble do Código Binário por B3 B2 B1 B0 temos:

Quantificação	Reflexão	Quantificação	Reflexão	Quantificação	Reflexão	Quantificação
0	0	00	00	000	000	0000
1	1	01	01	001	001	0001
	1	11	11	011	011	0011
	0	10	10	010	010	0010
			10	110	110	0110
			11	111	111	0111
			01	101	101	0101
			00	100	100	0100
					100	1100
					101	1101
					111	1111
					110	1110
					010	1010
					011	1011
					001	1001
					000	1000

Figura 26. Exemplo de construção do código GRAY com 4 bits.

Tabela 26. **Tabela que relaciona o código Gray de 4 bits com o respectivo em binário**

	Gray				Binário			
	G_3	G_2	G_1	G_0	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	1	0	0	1	0
3	0	0	1	0	0	0	1	1
4	0	1	1	0	0	1	0	0
5	0	1	1	1	0	1	0	1
6	0	1	0	1	0	1	1	0
7	0	1	0	0	0	1	1	1
8	1	1	0	0	1	0	0	0
9	1	1	0	1	1	0	0	1
10	1	1	1	1	1	0	1	0
11	1	1	1	0	1	0	1	1
12	1	0	1	0	1	1	0	0
13	1	0	1	1	1	1	0	1
14	1	0	0	1	1	1	1	0
15	1	0	0	0	1	1	1	1

A partir da tabela verdade acima, devemos simplificar as saídas binárias através da técnica de mapa de karnaugh. Esta etapa é apresentada a seguir:

Mapa de Karnaugh para B_0

$$B_0 = \bar{G}_2\bar{G}_3(G_0\bar{G}_1 + \bar{G}_0G_1) + G_2\bar{G}_3(G_0G_1 + \bar{G}_0\bar{G}_1) + G_2G_3(G_0\bar{G}_1 + \bar{G}_0G_1) + \bar{G}_2G_3(G_0G_1 + \bar{G}_0\bar{G}_1)$$
$$= (G_0\bar{G}_1 + \bar{G}_0G_1)(G_2G_3 + \bar{G}_2\bar{G}_3) + (G_0G_1 + \bar{G}_0\bar{G}_1)(G_2\bar{G}_3 + \bar{G}_2G_3)$$
$$= (G_0 \oplus G_1)\overline{(G_2 \oplus G_3)} + \overline{(G_0 \oplus G_1)}(G_2 \oplus G_3)$$
$$B_0 = G_0 \oplus G_1 \oplus G_2 \oplus G_3$$

G_3G_2 \ G_1G_0	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

B_0

Mapa de Karnaugh para B_1

$$B_1 = G_1\bar{G}_2\bar{G}_3 + \bar{G}_1G_2\bar{G}_3 + G_1G_2G_3 + \bar{G}_1\bar{G}_2G_3$$
$$= G_1(G_2G_3 + \bar{G}_2\bar{G}_3) + \bar{G}_1(G_2\bar{G}_3 + \bar{G}_2G_3)$$
$$= G_1\overline{(G_2 \oplus G_3)} + \bar{G}_1(G_2 \oplus G_3)$$
$$B_1 = G_1 \oplus G_2 \oplus G_3$$

G_3G_2 \ G_1G_0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	0	0	1	1
10	1	1	0	0

B_1

Mapa de Karnaugh para B_2

$$B_2 = G_2\bar{G}_3 + \bar{G}_2G_3$$
$$B_2 = G_2 \oplus G_3$$

G_3G_2 \ G_1G_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

B_2

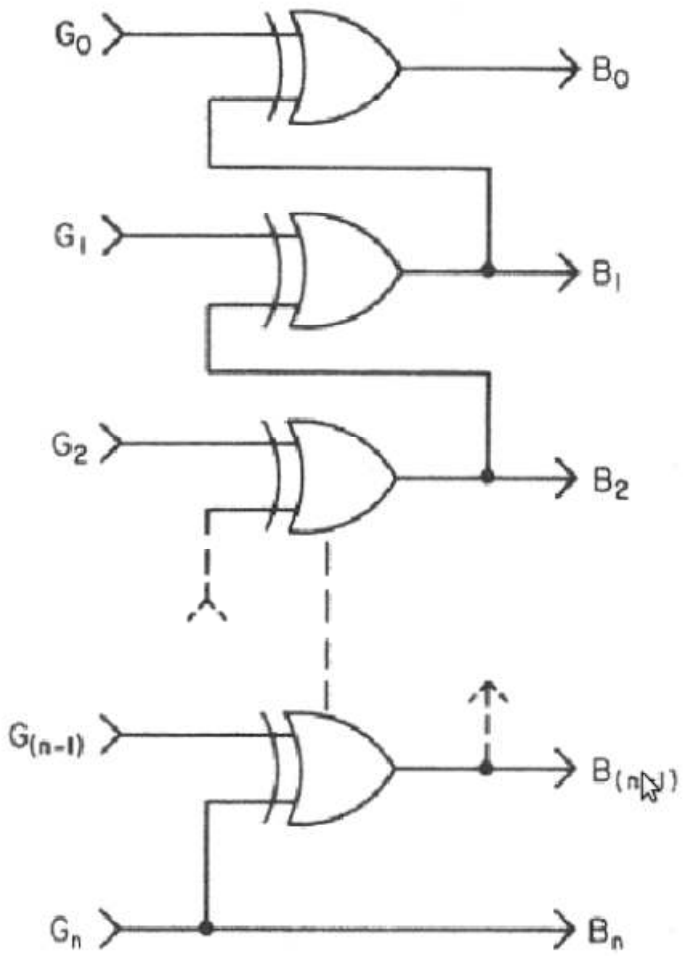
Mapa de Karnaugh para B_3

$$B_3 = G_3$$

G_3G_2 \ G_1G_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

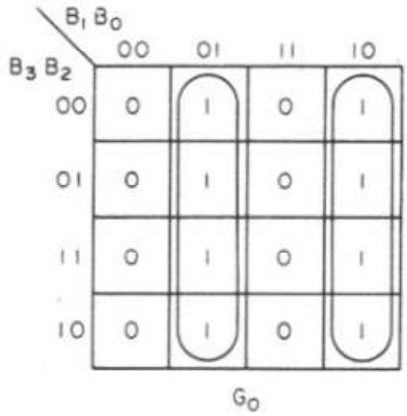
B_3

Generalizando para um Código Gray de N bits, podemos escrever que $Bn = Gn \oplus G(n+1) \oplus G(n+2) \oplus \ldots \oplus G(n-1)$, o que sugere o circuito lógico mostrado na figura abaixo:



8.4.4 Decodificador Binário para Gray

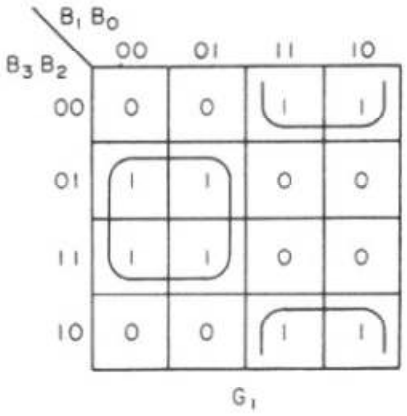
A seguir, são apresentados os mapas K para as funções lógicas que expressam G0, G1 , G2 e G3 em função de B3 B2 B1 B0 , tendo como ponto de partida a Tabela 26.



Mapa de Karnaugh para G_0

$G_0 = B_0\bar{B}_1 + \bar{B}_0 B_1$

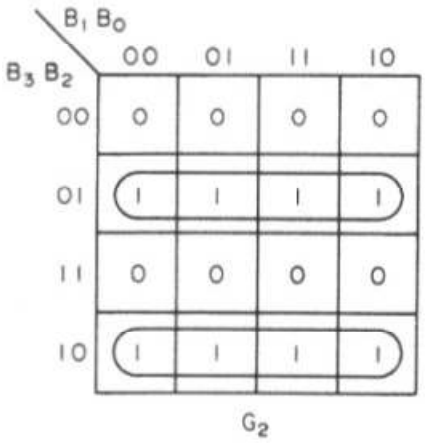
$G_0 = B_0 \oplus B_1$



Mapa de Karnaugh para G_1

$G_1 = B_1\bar{B}_2 + \bar{B}_1 B_2$

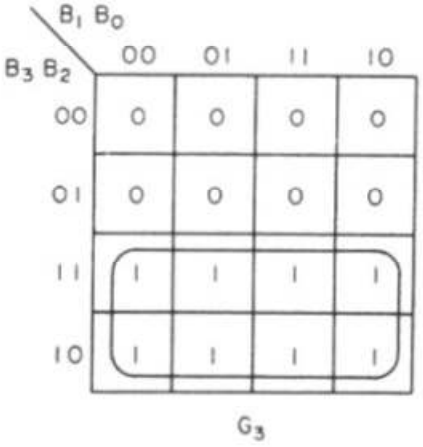
$G_1 = B_1 \oplus B_2$



Mapa de Karnaugh para G_2

$G_2 = B_2\bar{B}_3 + \bar{B}_2 B_3$

$G_2 = B_2 \oplus B_3$



Mapa de Karnaugh para G_3

$G_3 = B_3$

Generalizando para um Código Gray de N bits, podemos escrever que $Gn = Bn \oplus B(n+1)$, sendo n + 1 ≤ N − 1 . Isto sugere o circuito lógico mostrado na Figura 27 abaixo:

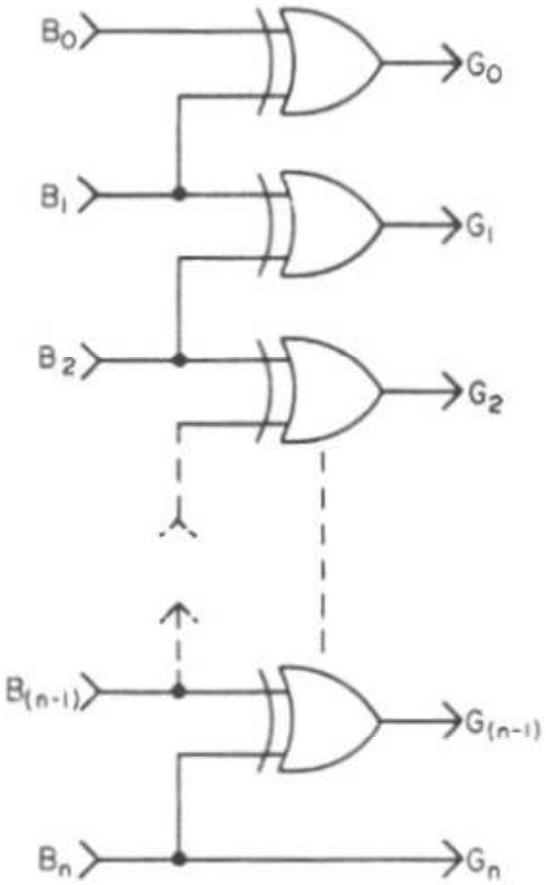


Figura 27. Circuito do decodificar de binário para Gray de n bits

8.5 Decodificadores para Display de 7 Segmentos

Um display de 7 segmentos mostra ao usuário de um sistema digital um algarismo decimal de 0 a 9, conforme mostram as Figura 28 abaixo.

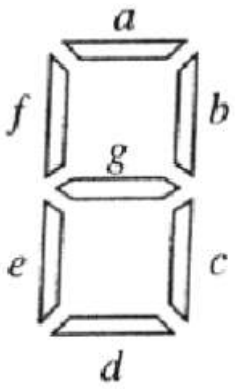


Figura 28. Display de 7 segmentos

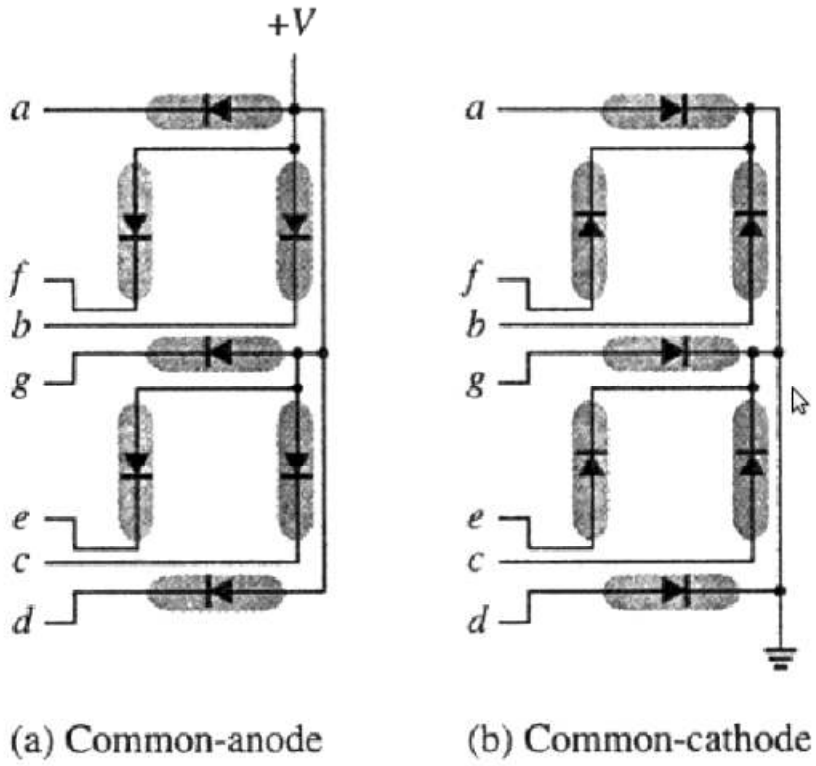


Figura 29. Dois tipos de displays de 7 segmentos, Anodo Comum (ativo com o nível '0') (a) e Catodo Comum (ativo com o nível '1').

Um Decodificador para Display de 7 Segmentos é um circuito digital formado por portas lógicas que, ao receber uma palavra binária de 4 bits representativa do algarismo decimal a ser mostrado, aciona os segmentos correspondente no display, conforme mostram a Figura 30 e a Tabela 27 e Tabela 28 abaixo.

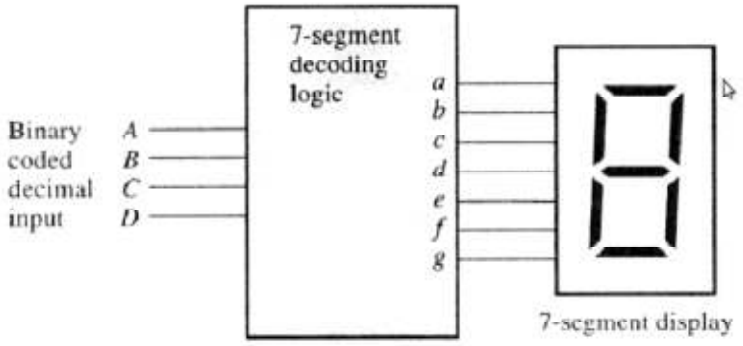


Figura 30. Circuito do decodificados de BCD para display de 7 segmentos.

Tabela 27. **Relação dos dígitos a serem ativados para acender o número correspondente em decimal**

Digit	Segments Activated
0	*a*, *b*, *c*, *d*, *e*, *f*
1	*b*, *c*
2	*a*, *b*, *d*, *e*, *g*
3	*a*, *b*, *c*, *d*, *g*
4	*b*, *c*, *f*, *g*
5	*a*, *c*, *d*, *f*, *g*
6	*a*, *c*, *d*, *e*, *f*, *g*
7	*a*, *b*, *c*
8	*a*, *b*, *c*, *d*, *e*, *f*, *g*
9	*a*, *b*, *c*, *d*, *f*, *g*

Tabela 28. **Tabela Verdade de um Decodificador para Display de 7 Segmentos**

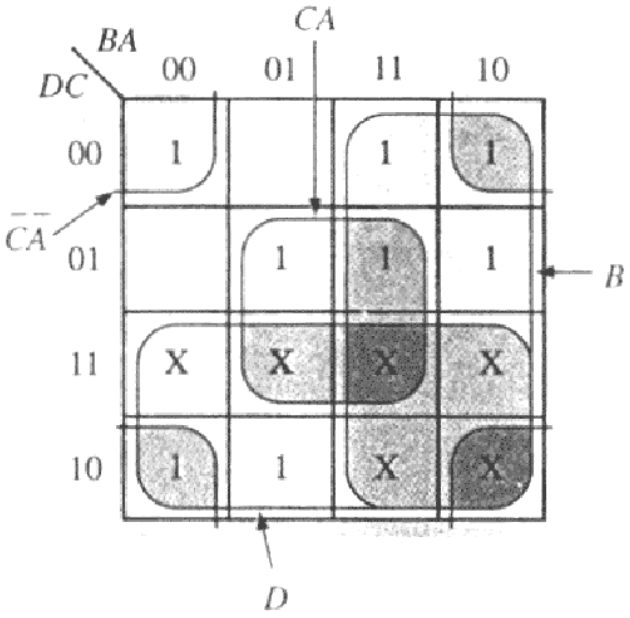
Decimal Digit	Inputs				Segment Outputs						
	D	*C*	*B*	*A*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	1	0	1	0	X	X	X	X	X	X	X
11	1	0	1	1	X	X	X	X	X	X	X
12	1	1	0	0	X	X	X	X	X	X	X
13	1	1	0	1	X	X	X	X	X	X	X
14	1	1	1	0	X	X	X	X	X	X	X
15	1	1	1	1	X	X	X	X	X	X	X

Um Decodificador para Display de 7 Segmentos é um Circuito Integrado que contém as combinações de portas lógicas necessárias e otimizadas para a implementação do conjunto de Expressões Booleanas definidas pela Tabela 28 acima.

• Por exemplo, verificamos que a Expressão Booleana para o segmento a é:

$$a = \overline{D}\,\overline{C}\,\overline{B}\,\overline{A} + \overline{D}\,\overline{C}\,B\,\overline{A} + \overline{D}\,\overline{C}\,B\,A + \overline{D}\,C\,\overline{B}\,A +$$
$$+ \overline{D}\,C\,B\,\overline{A} + \overline{D}\,C\,B\,A + D\,\overline{C}\,\overline{B}\,\overline{A} + D\,\overline{C}\,\overline{B}\,A$$

Cujo Mapa K é:



A Expressão Booleana minimizada resulta em :

$$a = D + B + C\,A + \overline{C}\,\overline{A}$$

E o circuito :

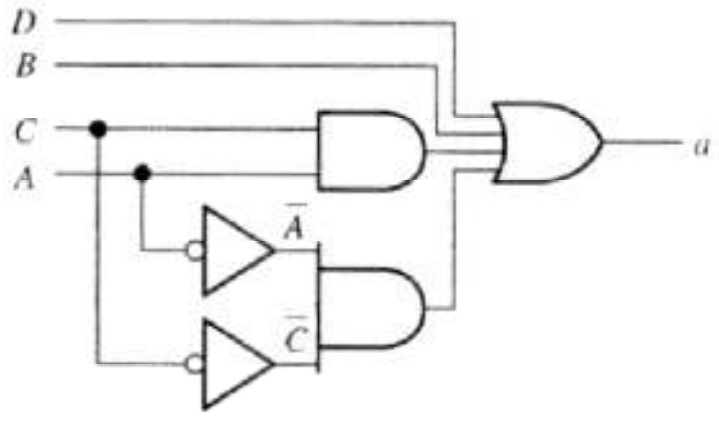


Figura 31. Circuito Resultante do Decodificador BCD para o segmento 'a'.

Exercício Proposto:

Determine o circuito lógico completo para o acionamento dos segmentos b, c, d, e, f, g. Caso, após a minimização individual das expressões booleanas para cada segmento, as funções lógicas resultantes para o acionamento de dois ou mais segmentos compartilharem termos comuns, faça a minimização adicional aproveitando o compartilhamento entre os termos.

9. Circuitos de Comparação, Multiplexadores e Demultiplexadores

O circuito de comparação mais utilizado e facilmente implementado é o circuito Exclusive OR, ou XOR. Em uma porta XOR, sempre que as entradas estiverem em níveis opostos o circuito lógico do exclusive-OR (XOR) irá indicar um nível alto em sua saída. Portanto, sempre que as entradas forem diferentes, a saída estará em nível lógico '1'.

9.1 Circuito de comparação com porta XOR

A porta XOR, além de ser utilizada como somadora binária, pode ser utilizada na comparação de bits. A seguir é apresentada a porta XOR e a sua tabela verdade, seguidas pelo mapa K.

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Figura 32. Porta XOR como comparador e Tabela verdade

A figura abaixo apresenta o correspondente Mapa de Karnaugh para a porta XOR.

	$\overline{\mathbf{B}}$	**B**
$\overline{\mathbf{A}}$	0	1
A	1	0

Figura 33. Mapa de Karnaugh da lógica XOR

Como já vimos anteriormente, sempre que tivermos a lógica $\bar{A}B + A\bar{B}$ podemos substituir pela lógica **A** ⊕ **B**. O mesmo vale para uma lógica com mais de um bit em A e B.

	B1 B0	**B1 B0**	**B1 B0**	**B1 B0**
$\overline{\mathbf{A1}}$ $\overline{\mathbf{A0}}$	0	1	1	1
$\overline{\mathbf{A1}}$ **A0**	1	0	1	1
A1 A0	1	1	0	1
A1 $\overline{\mathbf{A0}}$	1	1	1	0

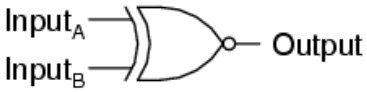
Figura 34. Mapa de Karnaugh. Resultado das combinações em que os valores de 2bits de A e B diferem.

$$X = A1 \oplus B1 + A0 \oplus B0$$

9.2 Circuito de comparação com porta Exclusive-NOR (XNOR)

Sempre que as entradas forem iguais o circuito lógico exlusive-NOR (XNOR) irá indicar um nível alto em sua saída.

Temos a seguinte tabela verdade.



A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

Equivalent gate circuit

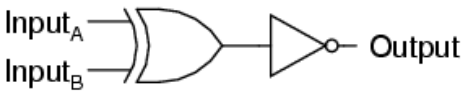


Figura 35. Porta XOR como comparador, Tabela Verdade e circuito equivalente com porta XOR e inversora

A figura abaixo apresenta o correspondente Mapa de Karnaugh para a porta XNOR.

	$\overline{\mathbf{B}}$	**B**
$\overline{}$		

A	1	0
A	0	1

Figura 36. Mapa de Karnaugh da lógica XNOR

Portanto sempre que houver uma lógica $\bar{A}.\bar{B} + A.B$, podemos substituir a mesma, por uma lógica **A** ⊙ **B**. O mesmo valerá para lógicas com A e B com mais de 1 bit

	$\overline{B1}\ \overline{B0}$	$\overline{B1}\ B0$	$B1\ B0$	$B1\ \overline{B0}$
$\overline{A1}\ \overline{A0}$	1	0	0	0
$\overline{A1}\ A0$	0	1	0	0
$A1\ A0$	0	0	1	0
$A1\ \overline{A0}$	0	0	0	1

$$X = A1 \odot B1 \,.\, A0 \odot B0$$

9.3 Circuito Comparador

Realiza a comparação entre 2 números binários, onde esta pode ser retornada como igualdade, informando se os números comparados são iguais ou não, ou ainda de magnitude, informando qual é maior.

Comparando-se dois números de 1 bit, temos a seguinte tabela verdade:

A	B	A = B	A > B	A < B
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

Figura 37. Tabela verdade para o comparador de 1 bit

Circuito que implementa o comparador

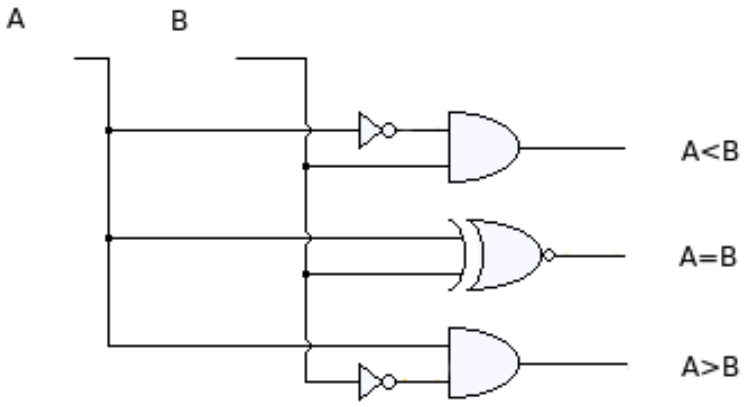


Figura 38. Circuito que realiza o comparador de 1 bit.

Para compararmos dois números de 2 bits utilizaremos o mapa de Karnaugh.

A seguir são apresentados os mapas de karnaugh para os circuitos A=B, A>B e A<B

Para calcularmos a expressão do circuito de comparação A = B.

	$\overline{A1}\ \overline{A0}$	$\overline{A1}\ A0$	$A1\ A0$	$A1\ \overline{A0}$
$\overline{B1}\ \overline{B0}$	1	0	0	0
$\overline{B1}\ B0$	0	1	0	0
$B1\ B0$	0	0	1	0
$B1\ \overline{B0}$	0	0	0	1

Figura 39. Mapa de Karnaugh que apresenta o resultado da expressão A=B.

(A = B) temos $\overline{A1}.\overline{A0}.\overline{B1}.\overline{B0} + \overline{A1}.A0.\overline{B1}.B0 + A1.\overline{A0}.A1.\overline{B0} + A1.A0.A1.B0$

(A = B) temos A1 ⊙ B1 . A0 ⊙ B0

Para calcularmos a expressão do circuito de comparação A > B

	$\overline{A1}\ \overline{A0}$	$\overline{A1}\ A0$	$A1\ A0$	$A1\ \overline{A0}$
$\overline{B1}\ \overline{B0}$	0	1	1	1
$\overline{B1}\ B0$	0	0	1	1
$B1\ B0$	0	0	0	0
$B1\ \overline{B0}$	0	0	1	0

Figura 40. Mapa de Karnaugh que apresenta o resultado da expressão A>B.

(A > B) temos $A1.\overline{B1}.\overline{B0} + \overline{A1}.A0.\overline{B0} + A1.\overline{B0}$

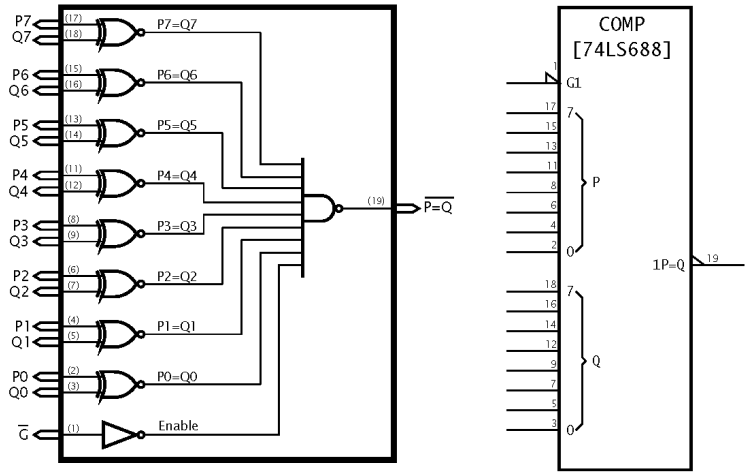
Para calcularmos a expressão do circuito de comparação A < B

	$\overline{A1}\ \overline{A0}$	$\overline{A1}\ A0$	$A1\ A0$	$A1\ \overline{A0}$
$\overline{B1}\ \overline{B0}$	0	0	0	0
$\overline{B1}\ B0$	1	0	0	0
$B1\ B0$	1	1	0	1
$B1\ \overline{B0}$	1	1	0	0

Figura 41. Mapa de Karnaugh que apresenta o resultado da expressão A<B.

(A < B) temos $\overline{A1}.\overline{A0}.B0 + \overline{A0}.B1.B0 + \overline{A1}.B1$

Temos comparadores já integrados em CIs como comparadores de igualdade 74LS688 e o de magnitude 74LS85. Estes circuitos integrados são apresentados na Figura 42 abaixo.



(a) Logic function *(b) ANSI/IEC logic symbol*

Figura 42. Circuito integrado que indica a igualdade de 2 entradas de 7 bits

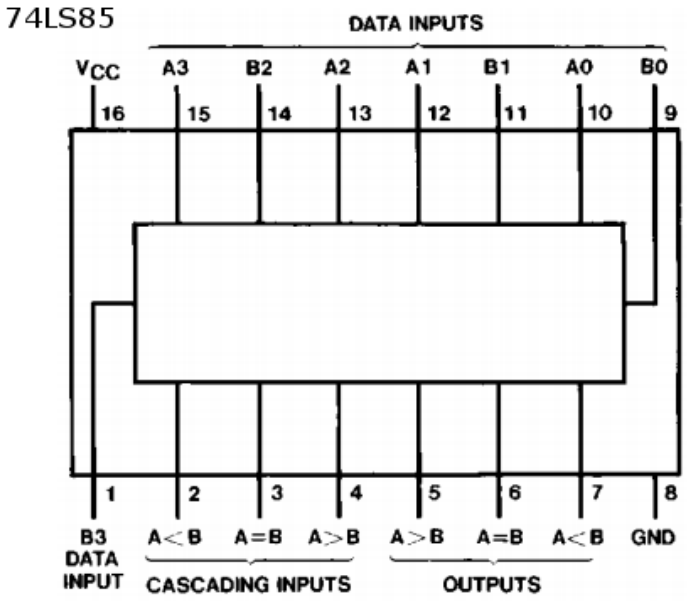


Figura 43. Circuito que compara a magnitude de 2 entradas de 7 bits.

9.4 Multiplexadores

Um circuito multiplexador é aquele que permite, através de uma chave de controle, que a saída copie uma dentre um grupo de entradas. Este circuito é apresentado na sua forma mais simples, ou seja, no seu tamanho mínimo na Figura 44.

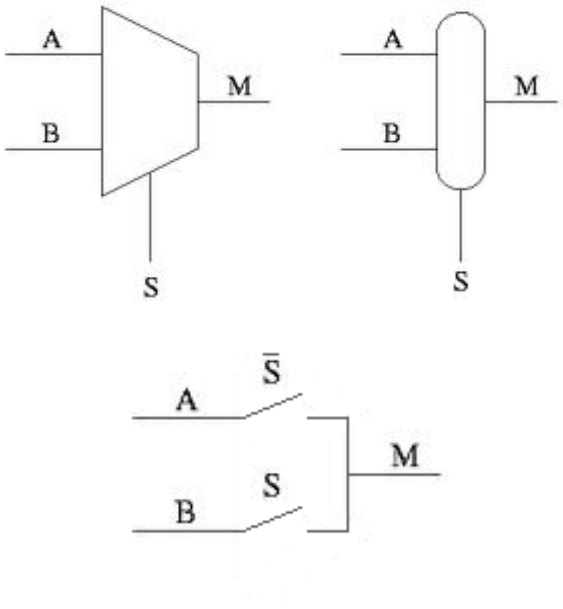


Figura 44. Circuito Multiplexador de 2 entradas e 1 saída.

O circuito apresentado na é comumente conhecido como MUX 2:1 por ser composto por 2 entradas e 1 saída.

Como a seleção das entradas não depende do nível lógico das mesmas a tabela-verdade que representa o funcionamento deste multiplexador deve ter na mesma coluna da saída, ao invés de níveis lógicos, o nome das variáveis de entrada:

S	**M**
0	A
1	B

Figura 45. Tabela Verdade do MUX 2:1

Exemplo: Seja um multiplexador de 4 entradas (d0, d1, d2 e d3) para uma saída q, com o controle sendo feito por select0 e select1.

Tem-se então a seguinte tabela verdade

Select0	**Select1**	**Q**
0	0	D0
0	1	D1

1	1	D2
1	0	D3

Figura 46. Tabela Verdade do MUX 4:1

A Figura 47 apresenta o circuito que realiza um multiplexador de 4 entradas e 1 saída, utilizando para controle da saída 2 bits de seleção.

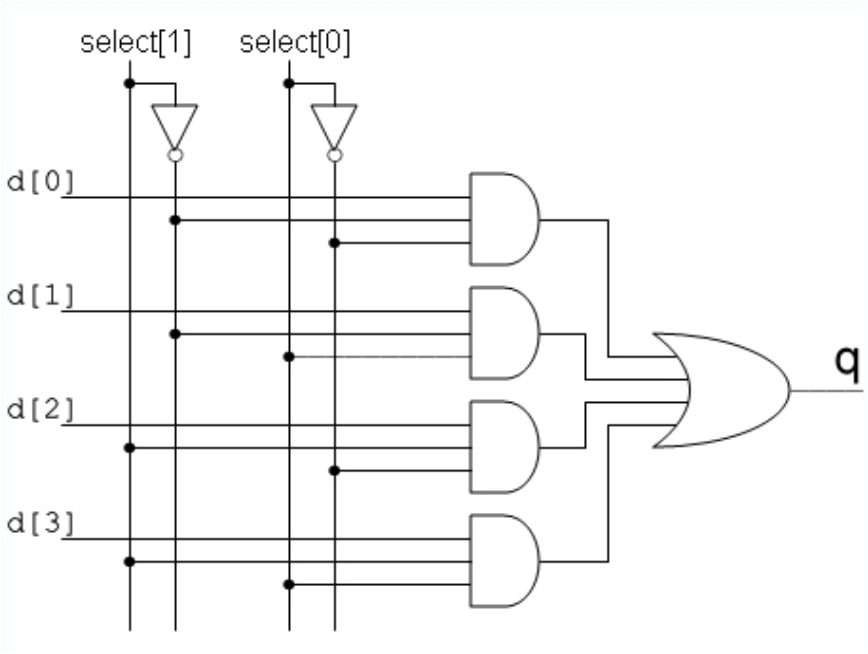
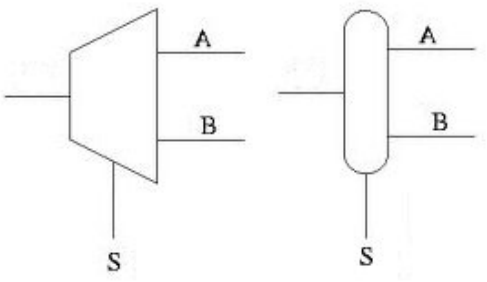


Figura 47. Circuito que realiza a lógica de um MUX 4:1.

9.5 Demultiplexadores

Um circuito demultiplexador pode ser entendido como um multiplexador invertido.



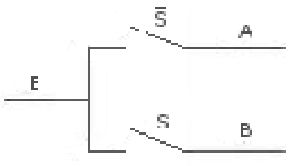


Figura 48. Circuito Demultiplexador de 1 entrada e 2 saídas.

Como a seleção das saídas não depende do nível lógico de entrada, a tabela verdade do demultiplexador deve ter nas saídas um nível lógico fixo quando a saída não é a selecionada (normalmente "0") e o nome da variável de entrada caso seja a saída selecionada:

S	**A**	**B**
0	E	0
1	0	E

Figura 49. Tabela verdade de um Demultiplexador 1:2

Exemplo: Seja um demultiplexador de 4 bits sendo suas saídas S0, S1, S2, S3, com o controle sendo feito por A e B, temos:

A	B	S_0	S_1	S_2	S_3
0	0	E	0	0	0
0	1	0	E	0	0
1	0	0	0	E	0
1	1	0	0	0	E

Figura 50. Tabela Verdade de um DEMUX 1:4

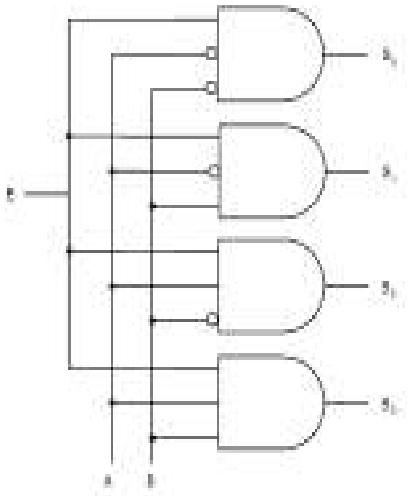


Figura 51. Circuito que realiza um DEMUX 1:4.

10.Unidade Lógica e Aritmética

A Unidade Lógica e Aritmética (ULA) é um circuito combinatório responsável pela execução de somas, subtrações e funções lógicas, em um sistema digital. Na Figura 52 é mostrado um esquema simplificado de uma ULA.

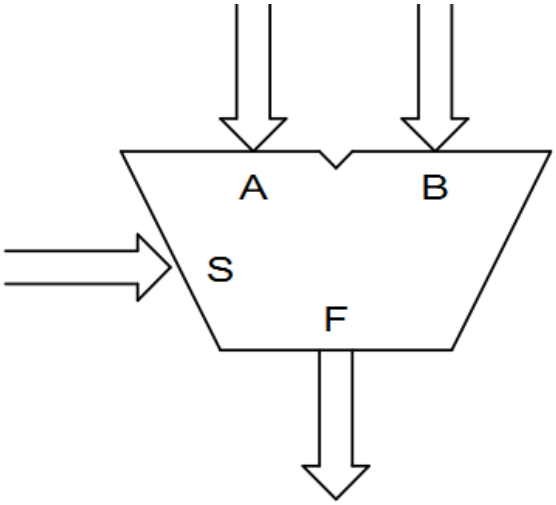


Figura 52. Esquema Simplificado de uma ULA.

A operação que deve ser executada com os dados de entrada (A e B) é determinada pelos sinais de controle (S) e o resultado é obtido na saída (F). A complexidade da ULA é proporcional à complexidade do sistema em que será utilizada; assim, sistemas simples permitem o uso de ULAs simples e sistemas sofisticados exigem ULAs sofisticadas.

10.1 Operações Aritméticas com Binários

10.1.1 Adição Binária

A adição decimal é mostrada a seguir:

$$\begin{array}{r} 376 \\ +461 \\ \hline 837 \end{array}$$

A adição é feita a partir do algarismo menos significativo. Quando a adição resulta em um valor maior que 9 ocorre um carry (vai um) para a próxima posição.

Como já vimos em um capítulo anterior, na adição binária podemos encontrar apenas quatro possibilidades:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10$$

(0 + carry 1 para a próxima posição)

Por exemplo, para dois números de 3 bits temos:

$$\begin{array}{r} 011 \\ +\ 110 \\ \hline 1001 \end{array}$$

10.1.2 Representação de Números com Sinal

Em sistemas digitais, os números binários são armazenados e manipulados em conjuntos de flip-flops, os registradores. Um registrador com 6 flip-flops pode armazenar números binários de 000000 a 111111 (0 a 6310), representando a magnitude do número.

Como os computadores e calculadoras podem operar com números positivos e negativos, uma maneira de representar números positivos e negativos é mostrada na Figura 53.

A6	A5	A4	A3	A2	A1	A0	
0	1	1	0	1	0	0	$= +52_{10}$
Bit de Sinal (+)	Magnitude = 52_{10}						

A6	A5	A4	A3	A2	A1	A0	
1	1	1	0	1	0	0	$= -52_{10}$
Bit de Sinal (-)	Magnitude = 52_{10}						

Figura 53. Representação de números positivos e negativos

O bit A6 é chamado bit de sinal, ou seja, ele determina qual o sinal do número. Esse número possui magnitude de 6 bits mais um bit de sinal. Embora esse sistema

seja direto, computadores e calculadoras não o utilizam normalmente porque a implementação do circuito é mais complexa. O sistema de representação de números binários com sinal mais utilizado é o sistema de complemento de 2.

Forma do Complemento de 1:

O complemento a 1 de um número binário é obtido substituindo-se cada 0 por 1 e cada 1 por 0. Isso pode ser visto a seguir:

1 0 1 1 0 1
0 1 0 0 1 0

Forma do Complemento a 2 :

O complemento a 2 de um número binário é obtido tomando-se o complemento a 1 do número e adicionando-se 1 na posição do bit menos significativo. Por exemplo:

	1 0 1 1 0 1	Equivalente binário de 4510
	0 1 0 0 1 0	Complemento a 1
+	1	Adiciona-se 1
	0 1 0 0 1 1	Complemento a 2 do número binário original

<u>Representação de Números com Sinal Usando Complemento de 2</u>

O sistema de complemento de 2 para representar números com sinal funciona do seguinte modo:

- Se o número é positivo, a magnitude é mostrada na sua forma binária direta e um bit de sinal 0 é colocado na frente do bit mais significativo (MSB).

- Se o número é negativo, a magnitude é representada na sua forma de complemento a 2 e um bit de sinal 1 é colocado na frente do bit mais significativo (MSB). Podemos observar na Figura 54 um exemplo que ilustra esta relação para o número 45.

A6	A5	A4	A3	A2	A1	A0	
0	1	0	1	1	0	1	$= +45_{10}$
Bit de Sinal (+)	Binário direto						

A6	A5	A4	A3	A2	A1	A0	
1	0	1	0	0	1	1	$= -45_{10}$
Bit de Sinal (-)	Complemento a 2						

Figura 54. Representação de números positivos e negativos usando complemento a 2

10.1.3 Negação

A negação é a operação que converte um número positivo no seu negativo equivalente ou um número negativo no seu positivo equivalente. Por exemplo:

```
0  1  0  0  1  = +9 (número binário original)
1  0  1  1  1  = -9  (complemento a 2, negar)
0  1  0  0  1  = +9 (negar novamente)
```

Faixa de Representação do Complemento a 2:

A faixa completa de valores que pode ser representada no sistema de complemento de 2 que tem N bits de magnitude é:

$$-2^N \text{ a } +(2^N-1)$$

Por exemplo, com N = 3 bits, a faixa de números sinalizados é mostrada na tabela abaixo

Valor Decimal	Complemento a 2
$+7 = 2^3 - 1$	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
$-8 = -2^3$	1000

Figura 55. Exemplo para ilustrar a faixa de representação de números em complemento de 2

10.2 Adição no Sistema de Complemento de 2

Vamos analisar vários casos de adição:

I) Dois números positivos: A adição de dois números positivos é direta.

+9	0	1001
+4	0	0100
+13	0	1101

II) Um número positivo e um outro menor e negativo: O número negativo deve estar na forma de complemento a 2. A soma é feita sobre todos os bits, inclusive os bits de sinal. O carry (vai um) gerado na última posição (MSB) é sempre descartado.

+9		0	1001
-4		1	1100
+5	1	0	0101

III) Um número positivo e um outro maior e negativo:

-9	1	0111
+4	0	0100
-5	1	1011

IV) Dois números negativos:

-9		1	0111
-4		1	1100
-13	1	1	0011

V) Dois números iguais em magnitude mas de sinais contrários:

+9		0	1001
-9		1	0111
0	1	0	0000

10.2.1 Subtração no Sistema de Complemento de 2

A operação de subtração no sistema de complemento a 2, na verdade, envolve uma operação de adição. Quando subtraímos um número binário (o subtraendo) de outro número binário (minuendo),

usamos o seguinte procedimento:

- Negar o subtraendo.

- Adicionar o número obtido ao minuendo.

Por exemplo, +9 – (+4) = +5

+9		0	1001
+4		0	0100
+9		0	1001
-4		1	1100
+5	1	0	0101

10.2.2 Overflow Aritmético

O overflow aritmético ocorre quando temos, por exemplo, a adição de +9 e +8:

+9	0	1001
+8	0	1000
	1	0001

O resultado esperado seria +17 mas a resposta tem um sinal negativo e uma magnitude incorreta. A representação do 17 precisa de mais de quatro bits,

ocasionando um erro de overflow. O overflow pode ocorrer sempre que dois números positivos ou dois números negativos estão

sendo somados.

10.3 Projeto de uma Unidade Lógica e Aritmética Simples

As operações aritméticas tipicamente realizadas por uma ULA são adição, subtração, incremento e decremento. Dentre as operações lógicas citam-se o E, o OU, identidade e complemento. As primeiras duas operações lógicas são definidas entre os pares de bits, cada um pertencendo a um dos operandos. Já as duas últimas operações lógicas são definidas para os bits de (somente) um dos operandos.

Como todas as operações aritméticas se baseiam na adição, pode-se projetar uma ULA acrescentando-se alguns circuitos lógicos especiais às entradas de um somador (ripple-carry). Os circuitos lógicos usados na realização das operações lógicas recebem o nome de extensores lógicos (EL) enquanto que aqueles usados na realização das operações aritméticas são chamados extensores aritméticos (EA).

A Figura 56 mostra um somador para dois números de 4 bits (A e B) com os extensores lógicos e aritméticos acoplados às suas entradas.

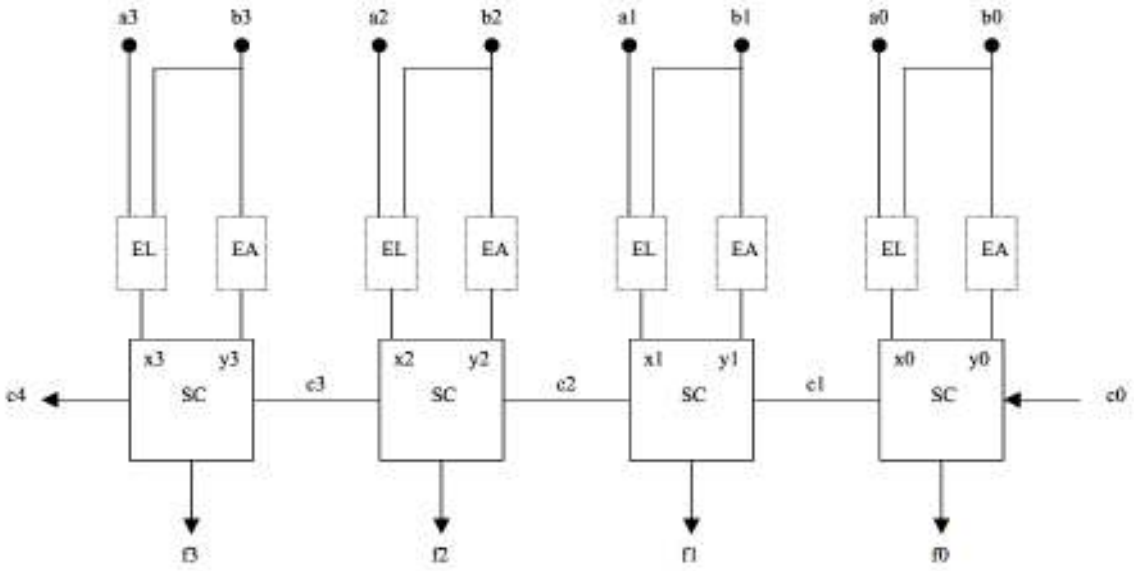


Figura 56. Somador para dois números de 4 bits com extensores lógicos (EL) e aritméticos(EA).

Para uma ULA capaz de realizar 4 operações lógicas e 4 operação aritméticas precisamos dos seguintes sinais de controle: M, S0 e S1.

- Se M=0, as variáveis S1 e S0 selecionam uma dentre as 4 operações lógicas disponíveis.

- Se M=1, S1 e S0 selecionam uma dentre as 4 operações aritméticas.

A Tabela 29 abaixo mostra as operações possíveis.

Tabela 29. **Operações possíveis de serem realizadas pela ULA**

M	S1	S0	nome da função	F	X	Y	C0
0	0	0	complementa	A'	A'	0	0
0	0	1	E	A E B	A E B	0	0
0	1	0	identidade	A	A	0	0
0	1	1	OU	A OU B	A OU B	0	0
1	0	0	decrementa	A-1	A	todos 1s	0
1	0	1	soma	A+B	A	B	0
1	1	0	subtrai	A+B'+1	A	B'	1
1	1	1	incrementa	A+1	A	todos 0s	1

10.3.1 Extensores Aritméticos

Os extensores aritméticos devem realizar as operações aritméticas, sendo utilizados ao se fazer M=1, ou seja, quando uma operação é do tipo aritmética. Examinando a tabela anterior observamos que para as situações em que M=1 a entrada X do somador deve receber o valor A (sempre), enquanto que a entrada Y recebe 1s, B, B'ou 0s, conforme for a combinação de S1 e S0. Portanto, cada extensor aritmético irá operar sobre um bit de B, de modo a gerar o resultado conforme mostrado na coluna Y.

Além disso, quando a operação solicitada corresponder a uma operação lógica (M=0), Y deve valer 0 (ou seja, todos os bits de Y devem ser iguais a zero).

A seguir são apresentadas a tabela-verdade e o mapa de Karnaugh para o extensor aritmético da ULA projetada.

S1	S0	bi	yi
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

S1S0 / bi	00	01	11	10
0	**1**	0	0	**1**
1	**1**	**1**	0	0

Figura 57. Tabela verdade e mapa de karnaugh para o extensor aritmético da ULA.

A equação minimizada, obtida a partir do mapa de Karnaugh será:

$$Yi = \overline{S1}.bi + \overline{S0}.\overline{bi}$$

A Figura 58 apresenta o circuito final do extensor aritmético da ULA projetada

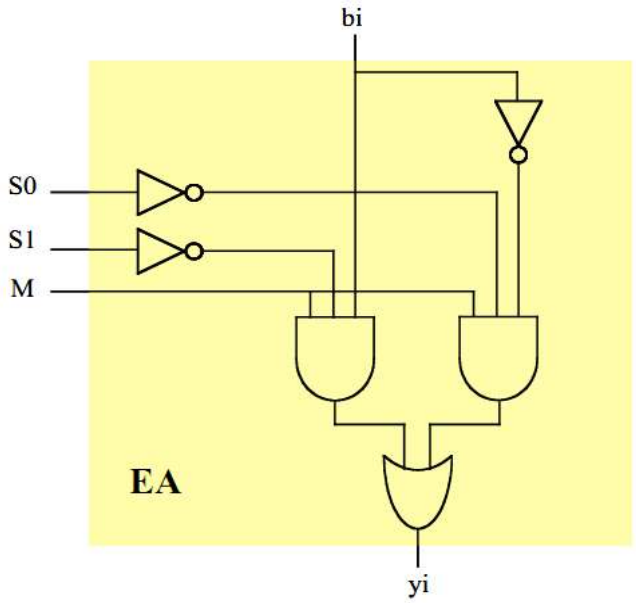


Figura 58. Circuito de um extensor aritmético (EA).

10.3.2 Extensores Lógicos

Analisando-se a tabela de operações, podemos observar que Y=0 e c0=0 quando M=0. Estas condições são conseqüência do fato de se ter definido que todas as operações lógicas são executadas exclusivamente pelos extensores lógicos, de modo que o somador não é utilizado quando M=0 por esta operação se tratar de uma operação aritmética. Porém, como X (que corresponde às saídas dos extensores lógicos) entra no somador, será necessário fazer Y=0 e c0=0 de modo a permitir que X

passe pelo somador sem ser alterado ou seja, quando selecionamos uma operação lógica X deve ser igual a saída do extensor lógico .

A seguir são apresentadas a tabela-verdade e o mapa de Karnaugh para o extensor lógico da ULA projetada.

Obs.: Considerar o símbolo (‘) como sendo o sinal de negação ou inversão lógica.

M	S1	S0	xi
0	0	0	ai’
0	0	1	ai.bi
0	1	0	ai
0	1	1	ai+bi
1	?	?	ai

	M=0				M=1			
S1S0 / ai bi	00	01	11	10	00	01	11	10
00	1							
01	1		1					
11		1	1	1	1	1	1	1
10			1	1	1	1	1	1

Figura 59. Tabela verdade e Mapa de Karnaugh do Extensor Lógico

A partir do mapa de Karnaugh chega-se a seguinte expressão:

$$Xi = \overline{M}.\overline{S1}.\overline{S0}.\overline{ai} + \overline{M}.S1.S0.bi + S0.ai.bi + S1.ai + M.ai$$

A Figura 60 apresenta o circuito final do extensor aritmético da ULA projetada

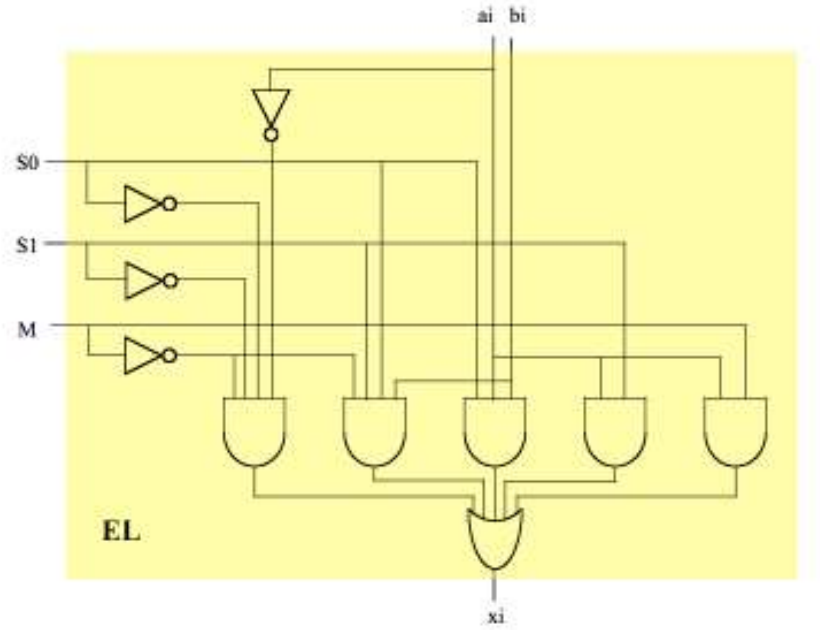


Figura 60. Circuito de um extensor logico (EL).

Observando-se a tabela de operações podemos deduzir a equação de c0 = M.S1, ou seja, o E entre M e S1.

Para finalizar o projeto da ULA, como partimos de um somador que pode ser configurado para realizar subtrações, o sinal de overflow deve ser formado da mesma maneira que em um somador/subtrator, ou seja, mediante a colocação de uma porta OU-exclusivo (XOR) entre o último e o penúltimo transportes, conforme mostra a Figura 61 abaixo.

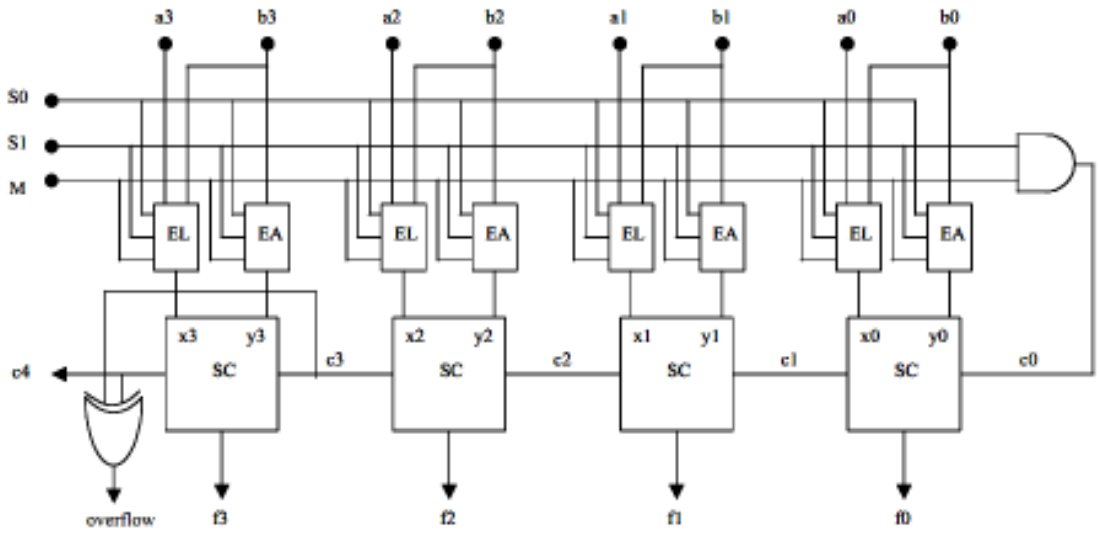


Figura 61. Circuito completo da ULA projetada.

10.3.3 Circuito Integrado 74181 – uma ULA de 4 bits

O circuito integrado MSI 74181 é uma ULA de 4 bits que tem possibilidade de executar 16 operações aritméticas binárias e 16 operações lógicas. A Figura 62 apresenta um diagrama simplificado deste circuito integrado.

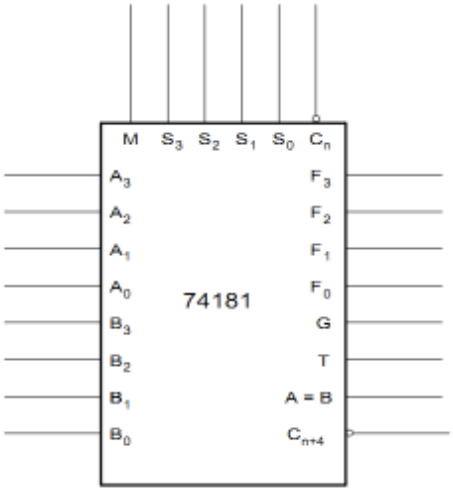


Figura 62. Esquema simplificado do CI 74181

As tabelas a seguir apresentam a descrição dos pinos e as operações da ULA, respectivamente.

Tabela 30. **Descrição dos pinos do CI 74181**

Pinos	Tipo	Descrição
A_0 a A_3 e B_0 a B_3	Entrada	Dados de entrada
C_n	Entrada	Bit de "vem um"
S_0 a S_3	Entrada	Seleção da operação
M	Entrada	Modo de operação: M=0 - para operações aritméticas M=1 - para operações lógicas
F_0 a F_3	Saída	Dados de saída (resultado)
C_{n+4}	Saída	Bit de "vai-um"
G e T	Saídas	Utilizadas para expansão utilizando *carry look-ahead*
A=B	Saída	Indica igualdade das duas entradas

Tabela 31. **Descrição das operações aritméticas realizadas pelo CI 74181**

Seleção				Funções Lógicas (M = 1)	Funções Aritméticas (M = 0)	
S_3	S_2	S_1	S_0		C_n = 1 (sem *carry*)	C_n = 0 (com *carry*)
0	0	0	0	$F = \overline{A}$	F = A	F = A + 1
0	0	0	1	$F = \overline{(A \text{ OR } B)}$	F = A OR B	F = (A OR B) + 1
0	0	1	0	$F = \overline{A} \cdot B$	$F = A \text{ OR } \overline{B}$	$F = A \text{ OR } \overline{B} + 1$
0	0	1	1	F = 0	F = -1 (*)	F = 0
0	1	0	0	$F = \overline{A} \cdot \overline{B}$	$F = A + A \cdot \overline{B}$	$F = A + A \cdot \overline{B} + 1$
0	1	0	1	$F = \overline{B}$	$F = (A \text{ OR } B) + A \cdot \overline{B}$	$F = (A \text{ OR } B) + A \cdot \overline{B} + 1$
0	1	1	0	$F = A \oplus B$	F = A - B - 1	F = A - B
0	1	1	1	$F = A \cdot \overline{B}$	$F = A \cdot \overline{B} - 1$	$F = A \cdot \overline{B}$
1	0	0	0	$F = \overline{A} \text{ OR } B$	F = A + A.B	F = A + A.B + 1
1	0	0	1	$F = \overline{(A \oplus B)}$	F = A + B	F = A + B + 1
1	0	1	0	F = B	$F = A \text{ OR } \overline{B} + A \text{ . } B$	$F = A \text{ OR } \overline{B} + A \text{ . } B + 1$
1	0	1	1	F = A . B	F = A . B - 1	F = A . B
1	1	0	0	F = 1	F = A + A	F = A + A + 1
1	1	0	1	$F = A \text{ OR } \overline{B}$	F = (A OR B) + A	F = (A OR B) + A + 1
1	1	1	0	F = A OR B	$F = A \text{ OR } \overline{B} + A$	$F = A \text{ OR } \overline{B} + A + 1$
1	1	1	1	F = A	F = A - 1	F = A

11.Método da Paridade para Detecção de Erros

Quando uma informação é transmitida de um dispositivo (transmissor) para outro (receptor), há a possibilidade de ocorrência de erros quando o receptor não recebe uma informação idêntica àquela que foi enviada pelo transmissor.

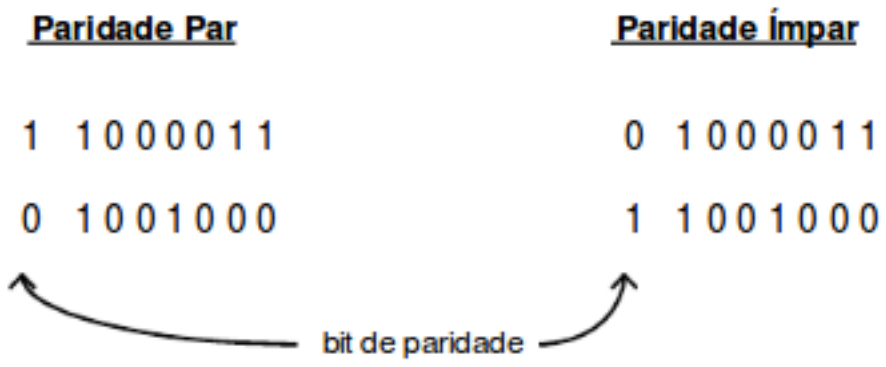
A movimentação de dados binários e de códigos de uma posição de armazenamento para outra é uma operação que acontece frequentemente em sistemas digitais. Durante a transmissão entre um dispositivo (o transmissor) e outro (o receptor), existe a possibilidade da ocorrência de erros, principalmente devido à presença de ruídos elétricos.

Esse ruído elétrico consiste em flutuações espúrias na tensão ou corrente que estão presentes em praticamente todos os sistemas eletrônicos. Por isso, muitos sistemas digitais utilizam algum método de detecção de erros.

Uma das técnicas mais simples para detecção de erros é o “Método de Paridade”.Um bit de paridade consiste em um bit extra anexado ao conjunto de bits a ser transferido. O bit de paridade pode ser 0 ou 1, dependendo do número de 1s contido no conjunto de bits. Dois métodos diferentes são usados. No método que usa “paridade par”, o valor do bit de paridade é determinado para que o número total de 1s no conjunto de bits (incluindo o bit de paridade) seja um número par.

Por exemplo, suponha que o conjunto de bits seja 1000011. Esse conjunto de bits tem três 1s; portanto, anexamos um bit de paridade par igual a 1 para tornar par o número total de 1s. O novo conjunto de bits, incluindo o bit de paridade, passa a ser: 11000011. Se o grupo de bits já contiver um número par de 1s, o bit de paridade terá valor 0.

O método de “paridade ímpar” é usado da mesma maneira, exceto que o bit de paridade é determinado para que o número total de 1s, incluindo o bit de paridade, seja ímpar



O bit de paridade é gerado para detectar erros de apenas um bit que ocorram durante a transmissão. Por exemplo, suponha que o conjunto de bits 1000001 seja transmitido com paridade ímpar. O código transmitido seria: 11000001. O receptor verifica se a informação transmitida contém um número ímpar de 1s (incluindo o bit de paridade). Em caso afirmativo, o receptor considera que o código foi recebido corretamente.

Agora, suponha que, devido a algum ruído, seja recebido o seguinte código: 11000000. O receptor identificará que o código tem um número par de 1s. Isso significa que há algum erro no código, devendo ser descartado.

É evidente que o método de paridade não funcionará se ocorrer erro em dois bits, porque dois bits errados não geram alteração na paridade do código. Na prática, o método de paridade é usado em situações em que a probabilidade de erro de um único bit é baixa e a probabilidade de erro em dois bits seja zero. O circuito mostrado na figura seguinte é usado para “geração de paridade” e “verificação de paridade”.

Esse exemplo usa quatro bits de dados fazendo uso da paridade par. Esse circuito pode ser facilmente adaptado para usar paridade ímpar e um número qualquer de bits. Os dados a serem transmitidos são aplicados ao circuito gerador de paridade que produz um bit de paridade par em sua saída, totalizando cinco bits para transmissão. Esses cinco bits entram no circuito verificador de paridade do receptor, o qual gera uma saída de erro (E), que indica se ocorreu ou não um erro em um único bit.

O circuito emprega portas OU-Exclusivo, pois ela opera de tal forma que gera 1 se o número de 1s nas entradas for ímpar e gera uma saída 0 se o número de 1s nas entradas for par. A Figura 63 apresenta o circuito gerador de paridade do tipo PAR e a apresenta o circuito verificador de paridade do tipo PAR.

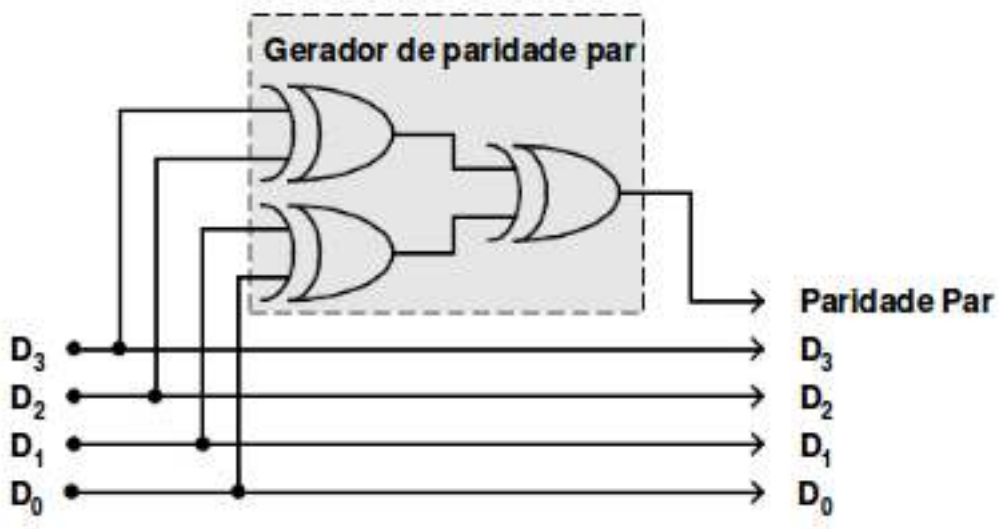


Figura 63. Circuito Gerador de Paridade PAR

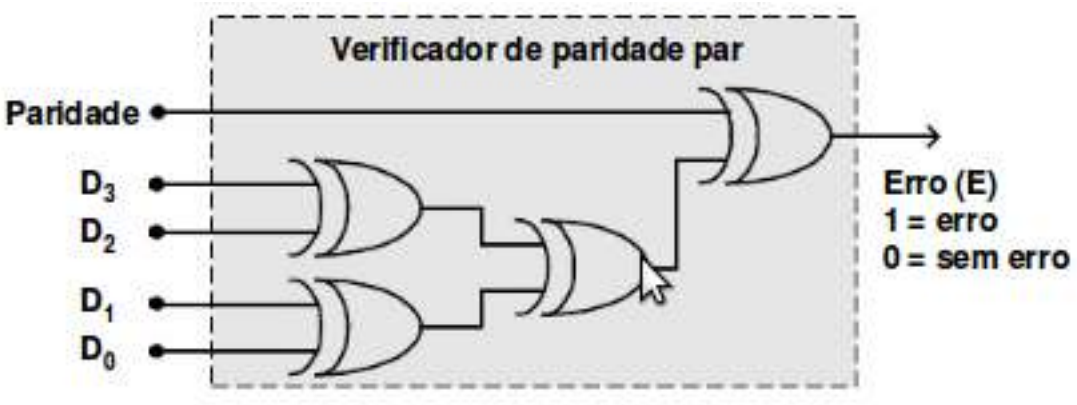


Figura 64. Circuito Verificador de Paridade do tipom PAR

Exercício Exemplo: A letra "C" em código ASCII é representada por 1000011.

Acrescentando o bit de paridade tanto par quanto ímpar temos:

1 1000011

bit de paridade par

0 1000011

bit de paridade impar

Exercícios:

Determine o bit de paridade par dos números binários abaixo.

a) 100101

b) 01011011

c) 1110111

Os dados abaixo foram recebidos por um circuito verificador de paridade ímpar de 7 bits, sendo o MSB o bit de paridade. Determine quais conjuntos de dados tiveram um bit errado na transmissão.

a) 10010100

b) 01001011

c) 11001011

Porque o método de paridade não consegue detectar um erro duplo de bit em um dado transmitido?

Determine a saída do gerador de paridade mostrado na figura acima para cada um dos seguintes conjuntos de dados de entrada D3 D2 D1 D0:

a) 0111

b) 1001

c) 0000

d) 0100

Determine a saída do verificador de paridade da página anterior para cada um dos conjuntos de dados enviados pelo transmissor:

a) 01010

b) 11110

c) 11111

d) 10000

12.Circuitos Sequenciais

Circuitos Combinacionais: para uma dada entrada, ou conjunto de entradas, apenas estas, em qualquer instante de tempo, interessam para a(s) resposta(s) em sua(s) saída(s). Quaisquer condições de entradas prévias não possuem efeito sobre as saídas atuais. Em outras palavras, circuitos combinacionais são desprovidos de memória, ou seja, da capacidade de armazenar uma informação.

Circuitos Seqüenciais: para uma dada entrada, ou conjunto de entradas, agora não apenas as entradas atuais, interessam para a(s) resposta(s) do circuito, mas também a(s) entrada(s) em instantes prévios ao atual. Ou seja, circuitos seqüenciais possuem memória.

Elemento de Memória: possui a capacidade de armazenar uma informação ocorrida no instante atual (t) para disponibilizá-la no instante futuro adjacente (t+1).

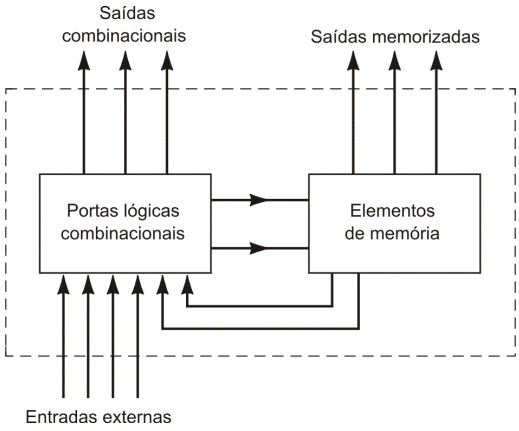


Figura 65. Sistema Digital (lógica combinacional e seqüencial).

12.1 Flip-Flops x Latches

Em eletrônica digital são os elementos básicos de memória. Isto, pois têm a capacidade de armazenar a menor porção de informação possível, o *bit*. São formadas por um arranjo de portas lógicas, que dependendo da maneira que este se dá, formam ou implementam os diferentes tipos de *flip-flops* e *latches*. Utilizam os princípios de atraso de propagação (tempo de porta) que serão posteriormente estuados.

Os *latches* dependem somente da combinação de suas portas de entrada para produzirem seu efeito de memória, já os *flip-flops* introduzem um novo conceito, até aqui, para realizarem o efeito memória. Este conceito é o conceito de borda, ou comutação (transição de nível), pois as entradas dos *flip-flops* somente serão avaliadas no momento em que o sinal de controle transicionar de nível lógico. Esta transição é sempre dada em um único sentido, subida ou descida. Borda de subida (quando o sinal vai do nível lógico ‘0’ para o nível lógico ‘1’) e borda de descida (quando ocorre o contrário).

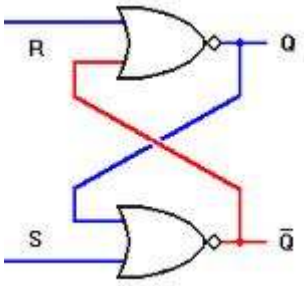
12.2 Latch

Elemento de memória cuja saída é alterada conforme o **nível** dos sinais de entrada, normalmente utilizado para registrar e manter um sinal (informação) sob dada condição.

12.2.1 Latch SR

O *latch* fundamental mais simples é este (o *latch SR*). Ele é implementado pela combinação de duas portas lógicas (normalmente portas lógicas *NAND ou NOR*) em um arranjo no qual cada porta lógica é ralimentada com a saída da outra. Assim com o atraso de propagação destes sinais é dado o efeito memória dada a combinação das entradas.

Set	Clear	Saída



0	0	Mantém
0	1	0
1	0	1
1	1	Inválida

Figura 66. Circuito básico do *Latch SR* (*NOR*) e a respectiva tabela verdade

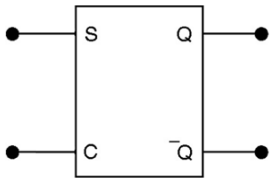
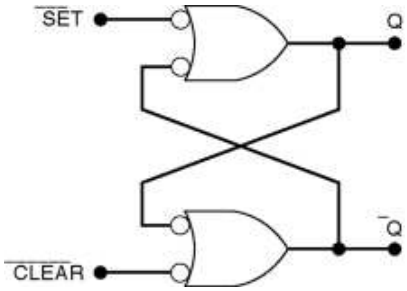


Figura 67. Bloco lógico do *Latch SR*.

Outra maneira de implementar um *latch SR* é utilizar o mesmo arranjo anteriormente mostrado, porém trocando as portas *NAND* por portas *OR* com as entradas invertidas. Este tipo de *latch* é conhecido como *Latch SR* com entradas invertidas. A Figura 68 mostra esta implementação.



$\overline{Set}$	$\overline{Clear}$	Saída
0	0	Inválida
0	1	1
1	0	0
1	1	Mantém

Figura 68. Circuito básico do *Latch SR* (*NAND*) e a respectiva tabela verdade

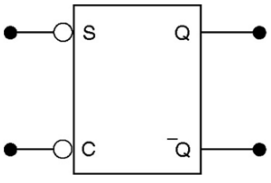


Figura 69. Bloco lógico do *Latch SR com entradas negadas (lógica negada)*.

Comportamento temporal :

O *Latch SR* com entradas invertidas tem o seguinte comportamento temporal, de acordo com a sua tabela verdade Figura 68. Observe que entre o instante de tempo T_1 e T_2 é satisfeita a condição de *set* já entre T_2 e T_3 a condição de manter o estado da saída. A condição de *clear* (*reset*)aparece entre os instantes de tempo T_3 e T_4, sendo que entre T_4 e T_5 mantém-se a saída. Já no instante T_5 existe uma condição de *clear* que se mantém até T_6 e após segue a condição de manter o sinal de saída.

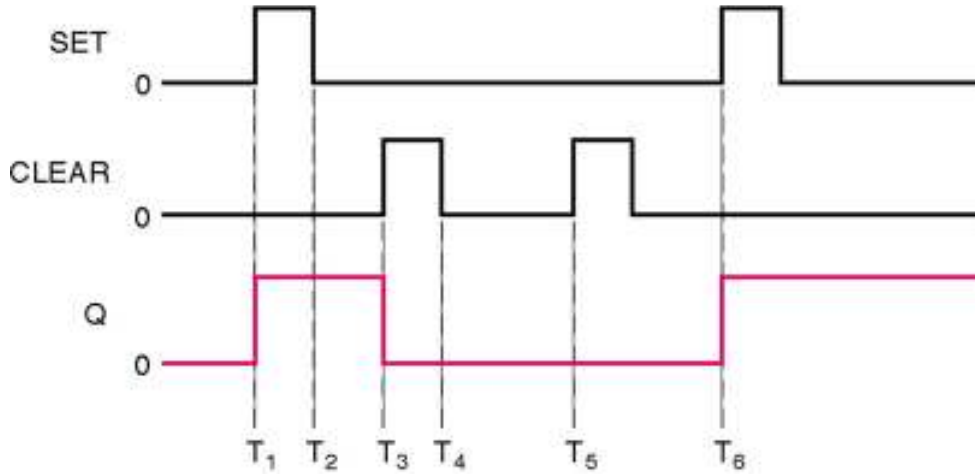


Figura 70. Comportamento temproral Latch SR.

Já quando este possui as entradas invertidas, se comportamento temporal é dado conforme a tabela verdade apresentada na Figura 68. De maneira análoga ao mostrado na Figura 6, o comportamento apresentado abaixo é marcado por instantes temporais. No instante T_1 é dada uma condição de *clear*, mantendo-se assim até o instante T_2 onde existe a condição de *set*, entre os instantes T_3 e T_4 há a condição de manter o sinal de saída, que é dado assim até o instante T_5 então é realizada a ação de *clear* que é mantida pela a partir do instante T_6:

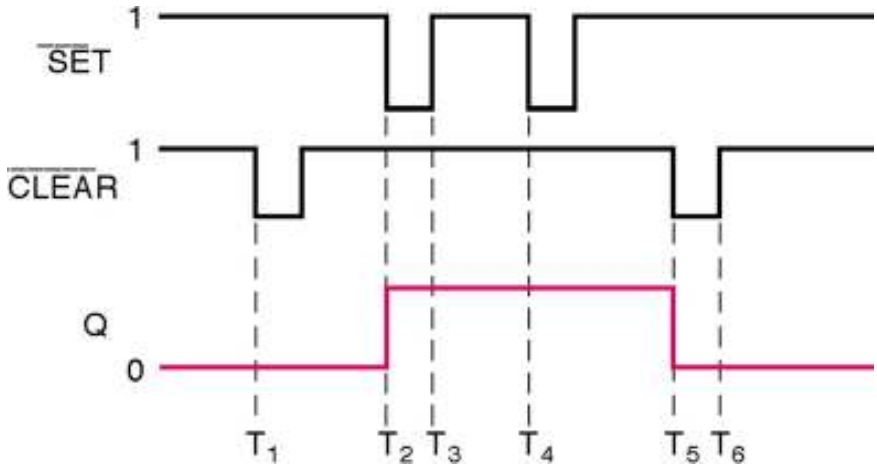


Figura 71. Comportamento temporal Latch SR com entradas invertidas.

12.2.2 Latch D – Latch Transparente:

Chamado de *latch* transparente, pois devido a sua construção permite ou não o registro (memorização) do dado na entrada *D*. Esta designação é feita por um sinal de controle *EN* (*enable*) que habilita ou não o registro do dado de entrada e o disponibiliza em sua saída. Sua construção é dada pela associação de duas portas *NAND* com um *latch SR* (*NAND*), conforme mostra a figura abaixo:

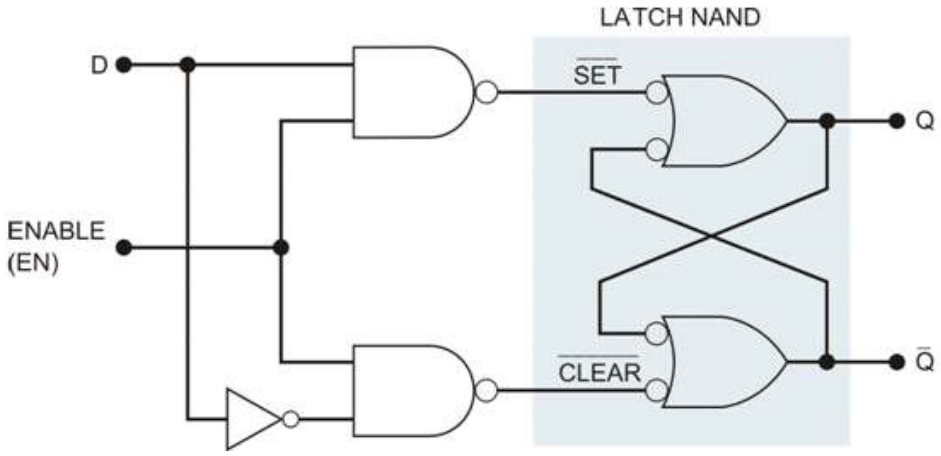
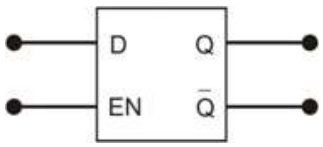


Figura 72. Construção do Latch D.

A tabela verdade, o bloco lógico e o comportamento temporal do *Latch D* são apresentados Figura 73 e Figura 74 respectivamente.



EN	D	Saída
0	X	Mantém
1	0	0
1	1	1

Figura 73. Bloco lógico e e tabela verdade do Latch tipo D.

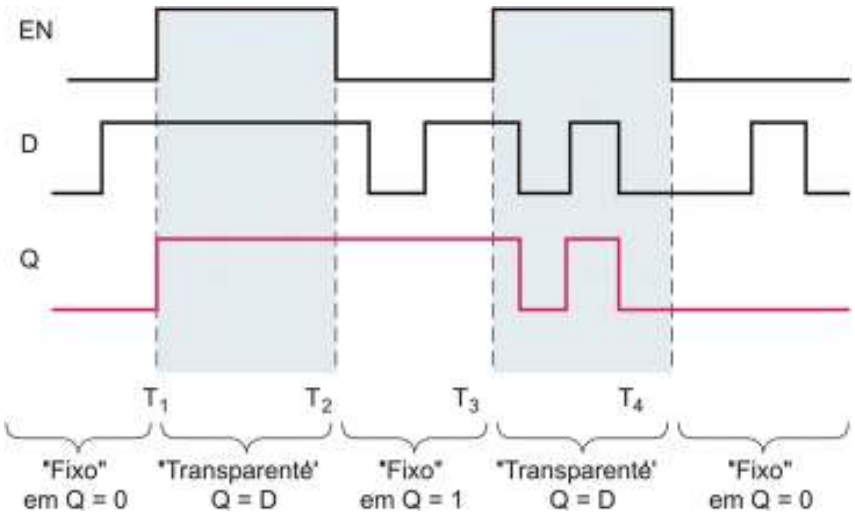


Figura 74. Comportamento temporal do Latch D.

12.3 Flip-Flop

Elemento de memória cuja saída é alterada conforme a **transição** do sinal de **controle** (normalmente chamado de *clock*). Aqui entra um conceito importante, que é a detecção de borda. Uma borda é uma transição de nível lógico de sinal, sendo borda de subida a transição do nível lógico '0' para nível lógico '1' Figura 75 (esquerda), e borda de descida o oposto Figura 75 (direita).

Figura 75. Clock funcionando com borda de subida (a esquerda) e com borda de descida (a direita).

Para detectar estes eventos de ocorrência de borda, tanto de descida ou de subida, utiliza-se um circuito bastante simples que faz esta detecção. O princípio utilizado para tal funcionalidade é o mesmo utilizado para a implementação dos *Latches*, que é o atraso de propagação. A seguir (Figura 76 e Figura 77) são apresentados os circuitos de detecção de borda de subida e descida e seus respectivos comportamentos temporais.

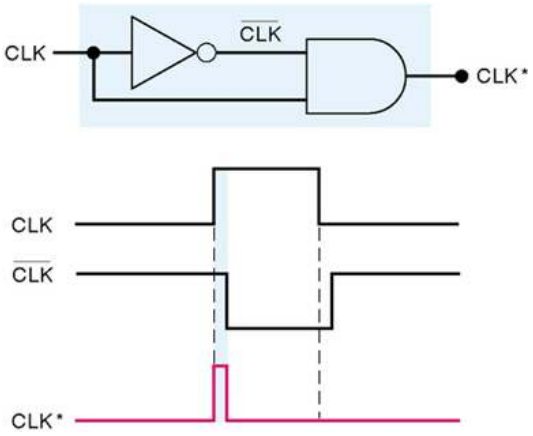


Figura 76. Circuito que realiza o detector de borda de subida para o clock e forma de onda deste circuito.

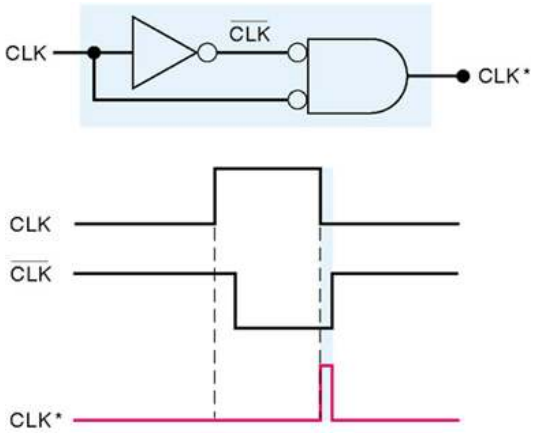


Figura 77. Circuito que realiza o detector de borda de descida para o clock e forma de onda deste circuito.

Com base neste detector são implementados todos os *Flip-Flops*, onde este circuito fica encarregado que habilitar e desabilitar o registro por parte do *latch* interno. A seguir serão apresentados os tipos de *Flip-Flops* típicos, ou mais comumente encontrados.

12.3.1 Flip-Flop SR

Este *Flip-Flop* nada mais é, que um *Latch SR* com a adição de um detector de borda em sua construção. Como podemos ver na Figura 76 e Figura 77, o detector de borda recebe um sinal de controle (normalmente chamado de *clock*) e entrega seu sinal de saída às entradas das portas lógicas que perfazem o circuito de *latch*. Como o detector está normalmente em nível lógico '0', o efeito memória do *latch* fica sempre em condição de manter o último dado. Quando da ocorrência de uma borda por um breve instante (tempo do pulso de detecção) o *lacth* é habilitado a fazer a memorização do estado atual de seus comandos de entrada.

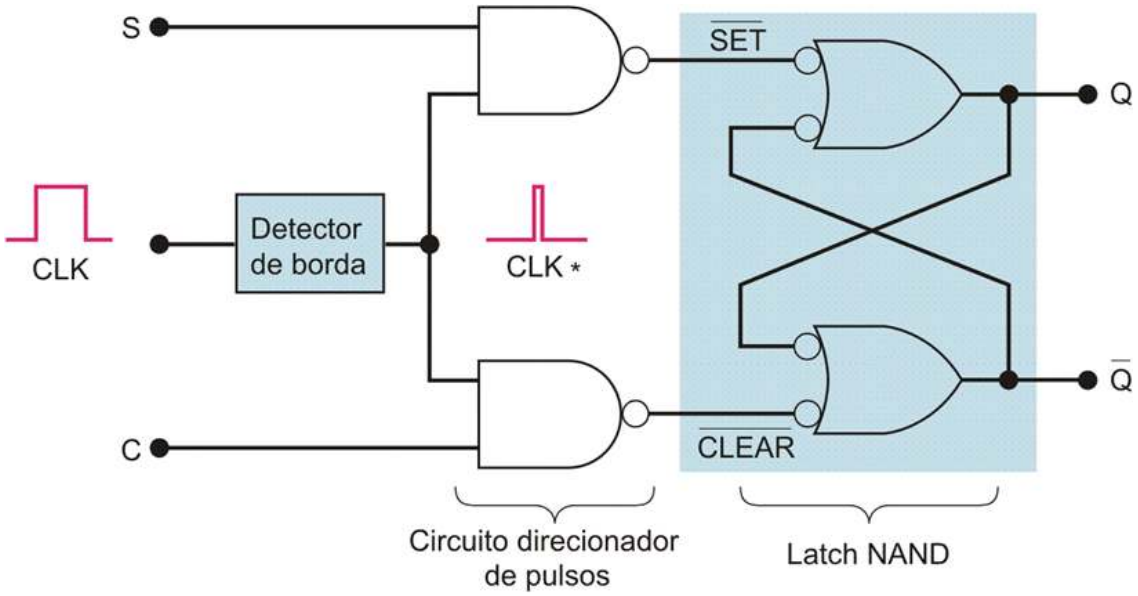


Figura 78. Estrutura de um Flip-Flop SR.

O comportamento temporal, o bloco lógico e a tabela verdade do *Flip-Flop SR* são apresentados respectivamente a seguir nas Figura 79 e Figura 80.

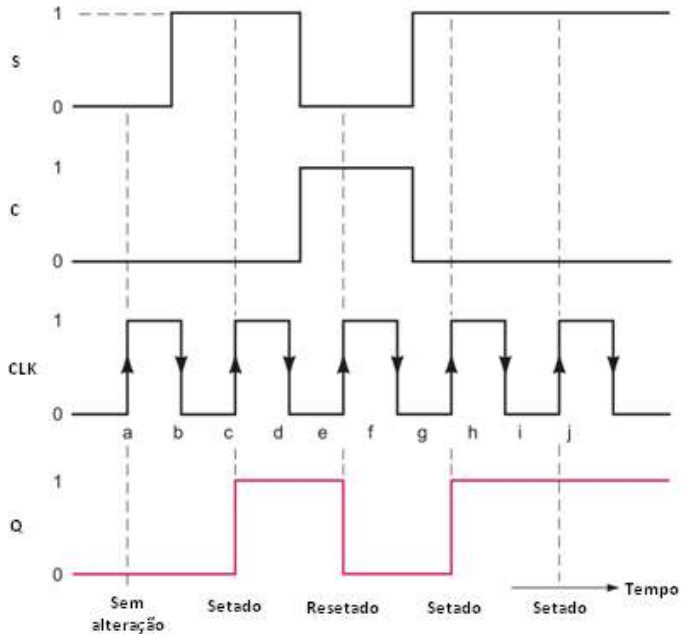
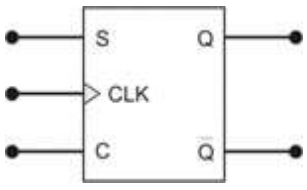


Figura 79. **Comportamento temporal**



S	C	CLK	Saída
0	0	↑	Mantém Q_0
1	0	↑	1
0	1	↑	0
1	1	↑	Inválida

Figura 80. Bloco lógico do Flip-Flop RS e sua tabela verdade.

12.3.2 Flip-Flop JK

Para evitar a sitação de condição inválida do *Flip-Flop SR*, foi criado o *Flip-Flop JK*. Este problema foi contornado adicionando a construção do *Flip-Flop SR* uma realimentação cruzada das saídas do *latch* para as entradas do circuido de habilitação via detecção de borda. A interpretação dada ao comando anteriormente inválido é de comutar o estado da saída atual. Comutar significa inverter o nível lógico.

Desta forma todas as combinações de entrada são passíveis de uma saída respectiva e válida. Uma curiosidade é a origem do nome *Flip-Flop JK*. Este *flip-flop* tem este nome devido a uma homenagem ao cientista que o concebeu *Jack Kilby*. Ele trabalhava na *Texas Instruments* e estava em um projeto contratado por uma empresa chamada *Huges Aircraft*, o qual eram desenvolvidas soluções para aviônica, quando (1960) concebeu o conceito do *flip-flop* que receberia suas inicias. Alguns anos mais tarde, mais precisamente em 2000, *Kilby* receberia o prêmio *Nobel* em física em reconhecimento a sua contribuição para as novas tecnologias de informação através da invenção e concepção do Circuito Integrado, o CI, em 1958.

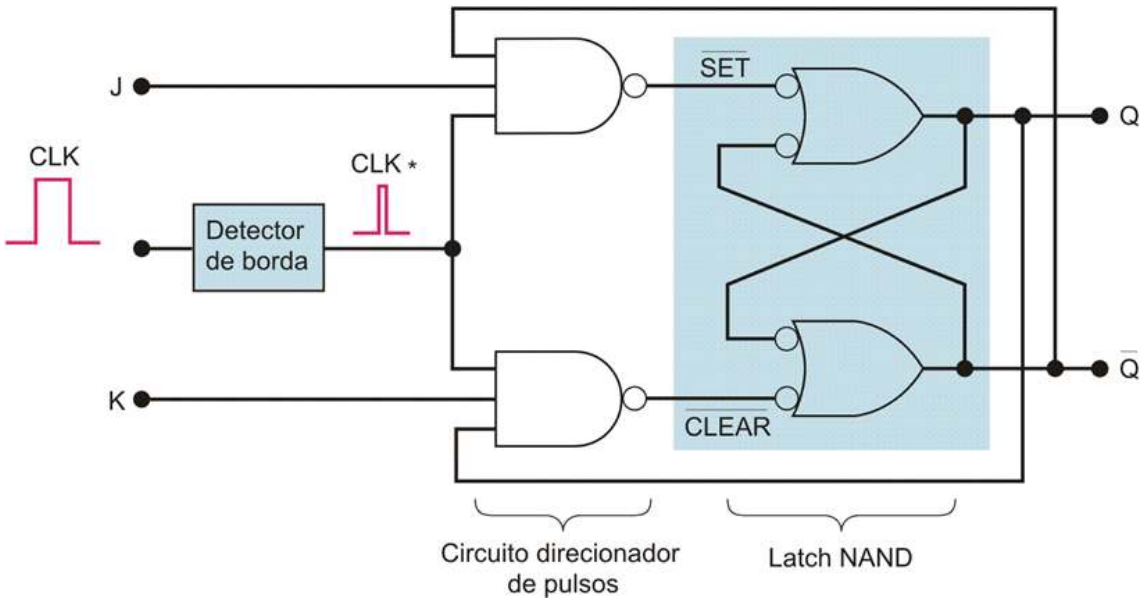


Figura 81. **Estrutura interna do *Flip-Flop JK***

O comportamento temporal, bloco lógico e a tabela verdade do *Flip-Flop JK* são apresentados respectivamente através da Figura 82 e Figura 83.

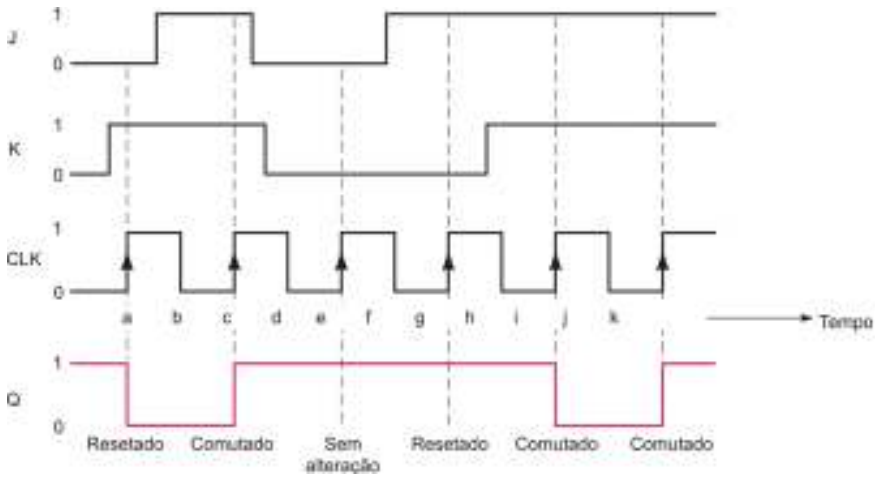
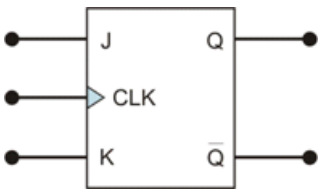


Figura 82. **Comportamento temporal do Flip-Flop JK**



J	K	CLK	Saída
0	0	↓	Mantém Q_0
1	0	↓	1
0	1	↓	0
1	1	↓	Comuta

Figura 83. **Bloco lógico e tabela verdade do Flip Flop JK.**

Com a estrutura do *Flip-FLop JK* pode-se per formar outros dois tipos de *flip-flops* batantes comuns e de grande utilidade em projetos de sistemas digitais. São estes o *Flip-Flop T* (*Toggle*) e o *Flip-Flip D* (*transparente*).

12.3.3 Flip-Flop T

Implementa a função de *toggle*, que nada mais é que é dado a informação de entrada (na entrada *T*) o sinal de saída do *flip-flop* é comutado ou mantido. Como já mencionado anteriormente, este *flip-flop* é construído a partir da estrutura do *Flip-Flop JK*. Abaixo, na Figura 84 segue o arranjo realizado com o *Flip-Flop JK* que possibilita o seu funcionamento como um *Flip-Flop T*.

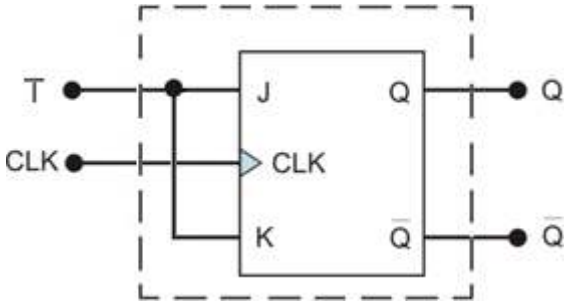


Figura 84. **Flip-Flop T a partir de um Flip-Flop JK.**

O comportamento temporal do *Flip-Flop T*, é mostrado na Figura 85, assim como seu bloco lógico (Figura 86) e sua tabela verdade (Figura 86), onde é mostrado seu comportamento frente entrada e saídas.

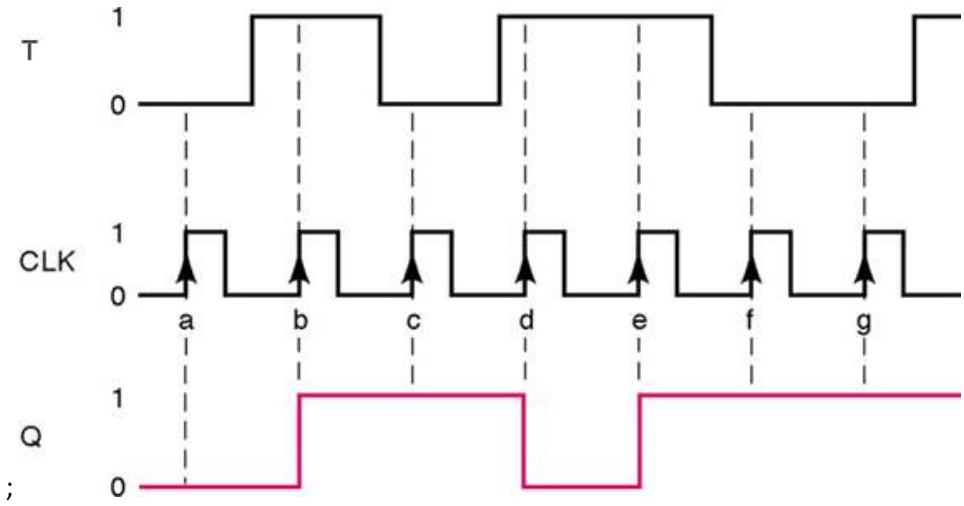
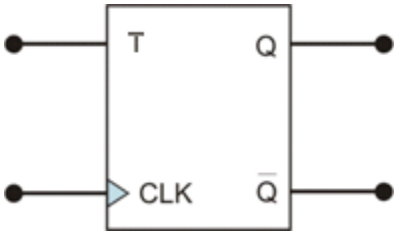


Figura 85. Comportamento temporal do Flip Flop tipo T



T	CLK	Saída
0	↓	Mantém Q_0
1	↓	Comuta

Figura 86. Bloco lógico e tabela verdade do Flip Flop tipo T

12.3.4 Flip-Flop D

O outro *flip-flop* que utiliza a estrutura do *Flip-Flop JK*, e com apenas uma simples lógica de ligação passa a garantir seu funcionamento é o *Flip-Flop D*. Assim como o *Latch D* é conhecido como transparente, pois ele transfere para a saída exatamente a informação que ele possui na entrada dado o sinal de controle (*clock*). Neste tipo de *flip-flop* as entradas *J* e *K* do *Flip-Flop JK* são ligadas por meio de uma porta inversora no sentido de *J* para *K*. Com isso consegue-se o efeito de transparência no *Flip-Flop JK* que é o comportamento do *Flip-Flop D* (Figura 87).

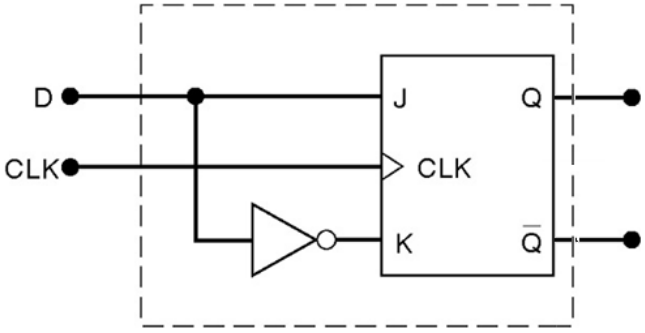


Figura 87. Flip-Flop D a partir de um Flip-Flop JK.

O comportamento temporal do *Flip-Flop D*, seu bloco lógico, bem como a sua tabela verdade, onde é mostrado seu comportamento frente entrada e saídas, estão apresentados na Figura 88 e Figura 89, respectivamente.

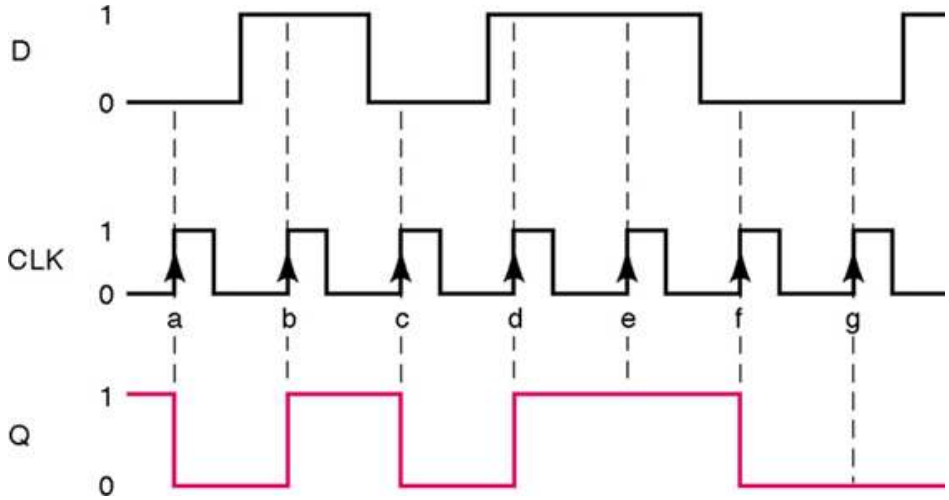
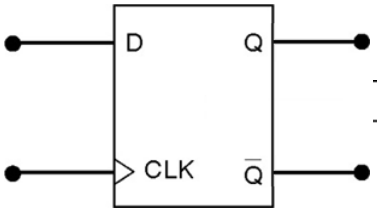


Figura 88. Comportamento temporal do Flip Flop tipo D.



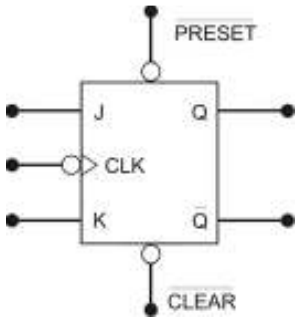
D	CLK	Saída
0	↓	0
1	↓	1

Figura 89. Bloco lógico e tabela verdade do Flip Flop tipo D.

12.3.5 Sinais de Preset e Reset Assíncronos

Um recurso importante a ser explorado em *flip-flops* são as portas de controle assíncronas disponíveis em alguns *flip-flops*. Com isto pode-se controlar as saídas independentemente dos eventos do sinal de controle síncrono (*clock*) *setando* (sinal em nível lógico '1') ou *resetando* (sinal em nível lógico '0') as mesmas. É importante ter em mente que estes sinais têm prioridade em relação a quaisquer controles síncronos. Caso acionado o *preset* o sinal de saída é *setado* durante todo o período de tempo em que este é acionado, caso o controle assíncrono acionado seja o *reset* o sinal de saída será *resetado*. O impasse se dá quando acionados simultaneamente os sinais de *preset* e *reset*. Quando isso ocorre o *flip-flop* fica com suas saídas em um estado inválido (sem definição).

A seguir é apresentado o exemplo de um *Flip-Flop JK* com *preset* e *reset* assíncronos. Este exemplo serve para ilustrar todos os demais tipos de *flip-flops* dotados desta característica. O bloco lógico que representa este *flip-flop* e sua tabela verdade (sinais de controle assíncronos) são apresentados na Figura 90 respectivamente.



$\overline{PRESET}$	$\overline{CLEAR}$	Saída
1	1	FF_JK
0	1	Q = 1
1	0	Q = 0
0	0	Inválida

Figura 90. Bloco lógico e tabela verdade do FF JK com PRESET e CLEAR assíncronos.

Um exemplo do comportamento temporal de um *Flip-Flop JK* é dado a partir da configuração do (utilizando um *Flip-Flop JK* com sinais de controle assíncronos) apresentada na Figura 91. Nesta configuração suas entradas *J* e *K* são fixadas em nível lógico '1', e são variadas as entradas de controle assíncronas. Desta forma pode-se observar mais isoladamente o comportamento das saídas de acordo com o comando dos controles assíncronos.

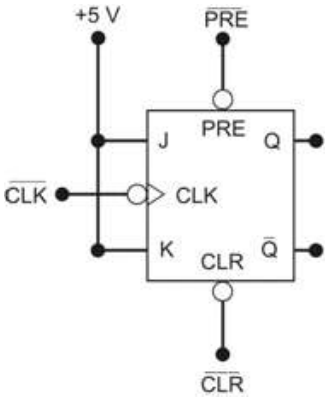


Figura 91. Configuração de exemplo com Flip-Flop JK com preset e clear assíncronos.

O comportamento temporal propriamente dito é mostrado na Figura 92, se onde pode observar os efeitos dos sinais de controle assíncrono (*preset* e *clear*), inclusive sua prevalência de prioridade frente ao sinal de controle síncrono (*clock*). Já a Figura 93 traz a análise detalhada instante a instante de cada ação dos sinais de controle, tanto síncrono como assíncronos.

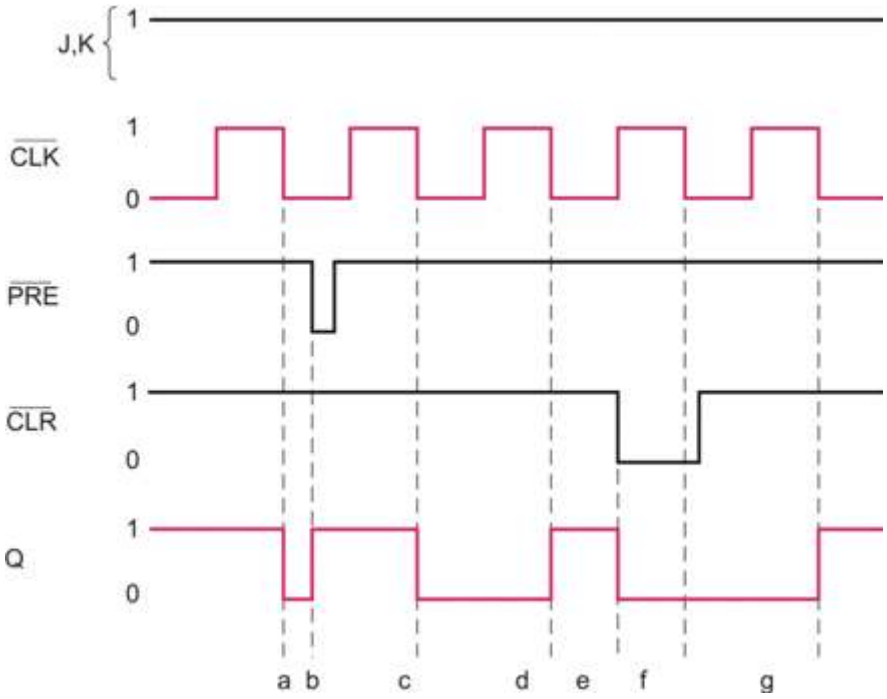


Figura 92. Comportamento temporal.

Instante	Operação
a	Comutação síncrona
b	$\overline{PRE}$ assíncrono
c	Comutação síncrona
d	Comutação síncrona
e	$\overline{CLR}$ assíncrono

f	$\overline{CLR}$ com maior prioridade que borda de $\overline{CLK}$
g	Comutação síncrona

Figura 93. Análise do comportamento temporal.