Universidade Federal de Santa Catarina
Departamento de Engenharia Elétrica

EEL 5310 - Sistemas Digitais

Prof. Carlos Maziero
Departamento de Automação e Sistemas

Versão 1.1

Florianópolis, agosto de 1997

Sumário

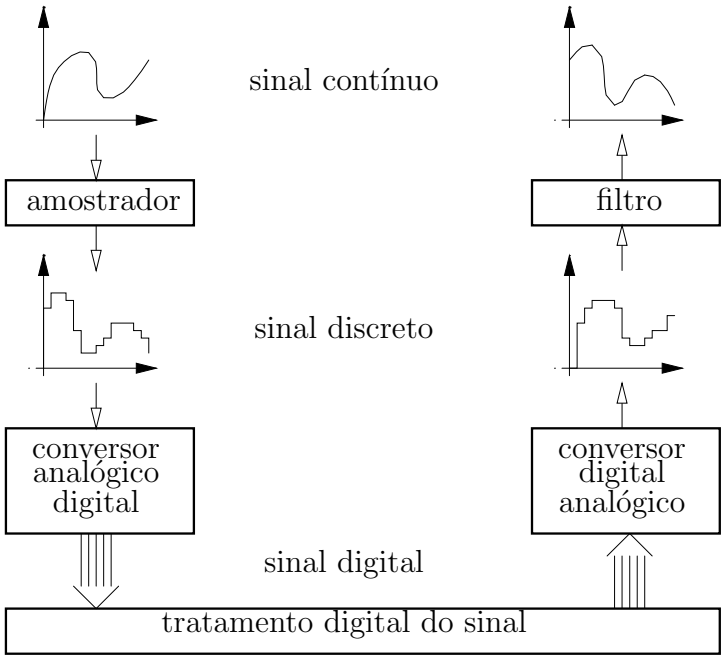
Introdução

A palavra "digital" vem do grego *digitus*, que significa *número*. Um sistema digital é portanto um sistema no qual a informação está codificada e circula sob a forma de números (ou seja, de valores discretos). No lado oposto, nos sistemas ditos *analógicos* a informação varia de modo contínuo. Exemplos comuns de sistemas digitais e analógicos são:

- *Sistemas digitais*: computador, comutador telefônico, semáforos, etc.
- *Sistemas analógicos*: amplificadores de som, TV tradicional, etc.

As vantagens de usar um sistema no qual a informação é tratada e armazenada sob uma forma digital são sobretudo a maior precisão no tratamento da informação e a maior imunidade a ruídos externos.

No mundo real a maioria das grandezas são analógicas (ou seja, variam de forma contínua). Para compatibilizá-las com o mundo digital é necessário converter os sinais analógicos em digitais e vice-versa. Isso é feito através de circuitos de amostragem filtros e conversores especiais chamados *conversores analógico-digitais*, como mostra a figura abaixo:



Um sistema digital é chamado *binário* quando considera somente dois valores possíveis para a codificação da informação a ser tratada ou armazenada. Podemos considerar esses valores como um par *aceso/apagado*, *ligado/desligado*, *verdadeiro/falso*, etc. Veremos mais tarde que esses

dois valores são suficientes para armazenar qualquer informação sob a forma digital, embora nem sempre sejam práticos. A grande vantagem da representação binária está na facilidade de implementação de circuitos eletrônicos para armazenar e realizar operações sobre informações binárias, permitindo a produção em larga escala de unidades que efetuam operações padronizadas e que podem ser associadas entre si para realizar operações mais complexas. Além disso, podem ser obtidas implementações bastante rápidas, operando em velocidades que ultrapassam as centenas de MHz.

A evolução dos sistemas digitais teve seu início no século 16, mas estes somente mostraram-se úteis neste século, e sua vulgarização se deu graças à recente evolução na microeletrônica. Eis um breve resumo, bastante incompleto, dessa evolução:

- *século 16*: Pascal e Leibniz propõe calculadoras baseadas em engrenagens.
- *século 19*: Charles Babbage constrói um computador programável mecânico.
- *década de 30*: computadores baseados em relés são usados para cálculos de balística.
- *1943*: construido o Eniac, com 18.000 válvulas.
- *1948*: invenção do transistor.
- *1951*: primeiro computador comercial, o Univac I.
- *anos 60*: apogeu dos computadores transistorizados.
- *anos 70*: circuitos integrados, invenção do micro-processador.
- *anos 80*: integração em larga escala (VLSI).
- *anos 90*: mais de 10^7 transitores em um chip.
- *futuro*: circuitos biológicos; circuitos usando luz; ...

Este curso visa apresentar as bases necessárias à compreensão, análise e projeto de circuitos envolvendo sinais digitais, e deve servir como base para um curso posterior sobre microprocessadores e microcontroladores. No **capítulo 1** veremos o sistema de numeração binário e como representar grandezas contínuas sob essa forma. Também veremos outros sistemas de numeração úteis, como o hexadecimal e o octal. O **capítulo 2** será dedicado ao estudo da álgebra booleana, que permite tratar valores codificados no sistema binário. No **capítulo 3** apresentaremos os circuitos ditos combinacionais, nos quais as saídas em um dado momento são função unica e exclusivamente das entradas naquele instante, ou seja, são circuítos sem memória. Os circuitos com memória, nos quais o estado das saídas em um instante pode depender de estados anteriores das entradas, são também chamados circuitos seqüenciais, e serão estudados no **capítulo 4**. No **capítulo 5** serão apresentados alguns circuitos complementares, que não se encaixam nas classificações anteriores, como os astáveis, mono-estáveis e o *schmitt-trigger*. O **capítulo 6** apresenta os princípios de funcionamento das memórias e seu emprego no projeto de circuitos. Finalmente o **capítulo 7** apresenta as diversas famílias de circuitos integrados digitais existentes no mercado, com especial atenção para as famílias TTL e CMOS.

Capítulo 1

Sistemas de numeração e codificação

1.1 Sistemas de numeração

1.1.1 O sistema decimal

O sistema decimal, também chamado sistema na base 10, é nosso sistema de numeração corrente. Ele opera com dez dígitos: 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9, e os números são formados por combinação destes dígitos. O incremento de uma unidade a um dígito faz avançar ao dígito seguinte, até chegar ao último (9). Ao incrementar este, ele retorna ao dígito inicial (0) e o dígito imediatamente à esquerda é incrementado seguindo a mesma regra:

$$\begin{aligned} 08 + 1 &= 09 \\ 09 + 1 &= 10 \\ 47 + 1 &= 48 \\ 99 + 1 &= 100 \end{aligned}$$

Assim, a posição dos dígitos em um número tem efeito multiplicador sobre a base. Desta forma, podemos decompor um número inteiro na base 10 da seguinte forma:

$$3754 = 3 \times 10^3 + 7 \times 10^2 + 5 \times 10^1 + 4 \times 10^0$$

e da mesma forma podemos decompor um número fracionário:

$$124.793 = 1 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 + 7 \times 10^{-1} + 9 \times 10^{-2} + 3 \times 10^{-3}$$

Dos exemplos acima podemos deduzir uma regra genérica para a decomposição de números na base 10, que nos servirá mais tarde para operações de mudança de base:

$$\cdots d_2 d_1 d_0 . d_{-1} d_{-2} \cdots = \cdots + d_2 \times 10^2 + d_1 \times 10^1 + d_0 \times 10^0 + d_{-1} \times 10^{-1} + d_{-2} \times 10^{-2} + \cdots$$

1.1.2 O sistema binário

O sistema binário, ou sistema na base 2, tem uma estrutura análoga à do sistema decimal, com a ressalva de operar somente com dois dígitos: 0 e 1. O incremento funciona da mesma forma que no sistema decimal:

$$\begin{aligned} 00 + 1 &= 01 \\ 01 + 1 &= 10 \\ 10 + 1 &= 11 \\ 11 + 1 &= 100 \end{aligned}$$

Podemos decompor um número binário da mesma forma que fizemos para um número decimal, como uma soma de potências da base:

$$100110_2 = 1 \times 10_2^5 + 0 \times 10_2^4 + 0 \times 10_2^3 + 1 \times 10_2^2 + 1 \times 10_2^1 + 0 \times 10_2^0$$

A notação 10_2 acima indica o número "10" na base 2 (que equivale ao número 2 na base 10, como veremos mais tarde), e que não deve ser confundido com o número 10_{10}. Por isso, todas as operações algébricas acima devem ser efetuadas na base 2. A decomposição também vale para os números fracionários.

Considerando os primeiros inteiros, podemos construir uma tabela de equivalência entre números decimais e binários (mais tarde veremos como converter números entre bases quaisquer através de métodos numéricos):

base 2	0	1	10	11	100	101	110	111	1000	1001	1010	...
base 10	0	1	2	3	4	5	6	7	8	9	10	...

Devido ao seu largo emprego em computadores, os números binários possuem uma nomenclatura própria. Assim, um dígito binário é chamado *bit*, enquanto um grupo de 8 bits (ou seja, um número binário inteiro positivo com 8 dígitos) é chamado *byte*. Em um número binário o bit mais significativo (o que tem maior peso) é chamado MSB (*Most Significant Bit*), e o bit menos significativo é chamado LSB (*Least Significant Bit*).

1.1.3 Os sistemas octal e hexadecimal

De forma forma análoga ao anterior, o sistema octal usa a base 8 e emprega os dígitos de 0 a 7 para a construção de números. A decomposição de números em potências da base funciona da mesma forma que nos casos anteriores. Entretanto, como $8 = 2^3$, a conversão entre números binários e octais é bastante facilitada, bastando agrupar os dígitos binários em grupos de 3 (começando pelo ponto decimal, tanto para a direita quanto para a esquerda) e obter os respectivos equivalentes octais, fazendo uso da seguinte tabela:

base 10	0	1	2	3	4	5	6	7	8	...
base 2	0	1	10	11	100	101	110	111	1000	...
base 8	0	1	2	3	4	5	6	7	10	...

Desta forma, o número 1011001100111_2 teria seus dígitos agrupados na forma

$$\underbrace{1}_{1}\ \underbrace{011}_{3}\ \underbrace{001}_{1}\ \underbrace{100}_{4}\ \underbrace{111}_{7}$$

e seu equivalente octal seria 13147_8.

O sistema hexadecimal usa a base 16 para seus números, e seus dígitos são $\{0, \ldots, 9, A, B, C, D, E, F\}$. Podemos construir a seguinte tabela de equivalência entre os primeiros inteiros nas bases decimal, binária, octal e hexadecimal:

base 10	base 2	base 8	base 16
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
...	...	...	...

Como $16 = 2^4$, a conversão entre binários e hexadecimais pode ser obtida agrupando os dígitos do número binário em grupos de 4, de forma similar à efetuada para os números octais. Assim, o número 1111001100111_2 teria seus dígitos agrupados na forma

$$\underbrace{1}_{1}\ \underbrace{1110}_{E}\ \underbrace{0110}_{6}\ \underbrace{0111}_{7}$$

e seu equivalente hexa seria $1E67_{16}$.

A representação em octal ou hexadecimal de números binários é bastante empregada em sistemas digitais, por oferecer números mais compactos (com menos dígitos) e mais fáceis de visualizar, e também porque as conversões da forma binária para hexa ou octal, e vice-versa, são bastante simples e rápidas.

1.2 Conversão entre bases

Podemos realizar as operações básicas de adição, subtração, multiplicação e divisão na própria base em que estamos trabalhando. No entanto, como nos é natural efetuar essas operações na base 10, frequentemente é mais simples converter os operandos para essa base, efetuar as operações e reconvertê-los novamente para a base de origem. Vamos estudar agora a conversão entre uma base qualquer e a base 10 de números inteiros e fracionários.

1.2.1 Números inteiros

Para converter um número inteiro na base 10 (n_{10}) para uma base b qualquer ($n_{10} \rightarrow n_b$), basta dividi-lo por essa base sucessivamente, usando operações de divisão inteira na base 10. Os restos r das divisões inteiras, tomados de trás para frente, nos fornecerão os dígitos do número n_b. O exemplo a seguir ilustra a conversão do número 87_{10} para a base 2 (q: quociente da divisão de n por b):

passo	1	2	3	4	5	6	7	8
n	87	43	21	10	5	2	1	0
q	43	21	10	5	2	1	0	-
r	1	1	1	0	1	0	1	-

Assim, 87_{10} equivale a 1010111_2. Essa operação pode ser empregada para converter qualquer número na base 10 para outra base qualquer.

A conversão de um número inteiro em uma base qualquer para a base 10 ($n_b \rightarrow n_{10}$) segue um processo diferente, que toma por base a decomposição do número em potências da base, como vimos no início deste capítulo. Para converter um número n_b de uma base qualquer b para a base 10 basta exprimi-lo como uma soma de potências da base, exprimir a base em seu equivalente na base 10 e em seguida efetuar as operações indicadas na base 10. O exemplo a seguir, no qual convertemos o número 1010111_2 para a base 10, ilustra esse processo:

$$\begin{aligned} 1010111_2 &= 1 \times 10_2^6 + 0 \times 10_2^5 + 1 \times 10_2^4 + \\ &\quad 0 \times 10_2^3 + 1 \times 10_2^2 + 1 \times 10_2^1 + 1 \times 10_2^0 \\ &= 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 64 + 16 + 4 + 2 + 1 \\ &= 87 \end{aligned}$$

A maneira mais simples de efetuar a conversão de um número de uma base a qualquer para outra base b qualquer ($n_a \rightarrow n_b$) é usar a base 10 como passo intermediário, efetuando $n_a \rightarrow n_{10} \rightarrow n_b$.

1.2.2 Números fracionários

Em uma mudança de base, a separação entre as partes inteira e fracionária de um número é mantida. Assim, a mudança de base de um número fracionário consiste da mudança de base de suas partes inteira e fracionária, separadamente. Como já vimos a mudança de base de números inteiros, resta agora estudar a troca de base da parte fracionária. Dado um número fracionário n_b, este pode ser expresso sob a forma $i_b.f_b$, onde i_b e f_b são respectivamente as partes inteira e fracionária de n_b. A parte f_b pode ser expressa da forma

$$f_b = d_{-1} \times b^{-1} + d_{-2} \times b^{-2} + d_{-3} \times b^{-3} + d_{-4} \times b^{-4} + \cdots$$

A conversão de um número fracionário de uma base b qualquer para a base 10 segue o mesmo procedimento da conversão de números inteiros, usando a decomposição acima. Por exemplo, a conversão do número $x_2 = 1101.001101_2$ para a base 10 efetua-se da seguinte forma:

$$\begin{aligned} x_{10} &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + \\ &\quad 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} + 0 \times 2^{-5} + 1 \times 2^{-6} \\ &= 2^3 + 2^2 + 2^0 + 2^{-3} + 2^{-4} + 2^{-6} \\ &= 8 + 4 + 1 + \frac{1}{8} + \frac{1}{16} + \frac{1}{64} \\ &= 13.203125 \end{aligned}$$

Para a conversão de um número fracionário da base 10 para outra base b qualquer ($n_{10} \rightarrow n_b$), convertemos a parte inteira usando o método apresentado anteriormente (divisão inteira) e convertemos a parte fracionária usando o método apresentado a seguir: vimos que podemos representar a parte fracionária f de um número em uma base b por:

$$f = d_{-1}b^{-1} + d_{-2}b^{-2} + d_{-3}b^{-3} + \cdots$$

onde $d_i \geq 0$ são os dígitos que compõe a parte fracionária. Por exemplo, $0.374_{10} = 3 \times 10^{-1} + 7 \times 10^{-2} + 4 \times 10^{-3}$. A conversão para uma base b qualquer consiste portanto em encontrar os valores d_i na base desejada b. Multiplicando f pela base b obtemos:

$$f \times b = d_{-1}b^{0} + d_{-2}b^{-1} + d_{-3}b^{-2} + \cdots$$

Assim d_{-1} pode ser retirado da parte inteira de $f \times b$. Aplicando sucessivamente essa multiplicação sobre a parte fracionária restante obteremos os demais dígitos de f. Como exemplo, vamos converter o valor decimal $X_{10} = 4.407$ para a base 2:

1. Parte inteira: $4_{10} = 100_2$
2. Parte fracionária: $f = 0.407$

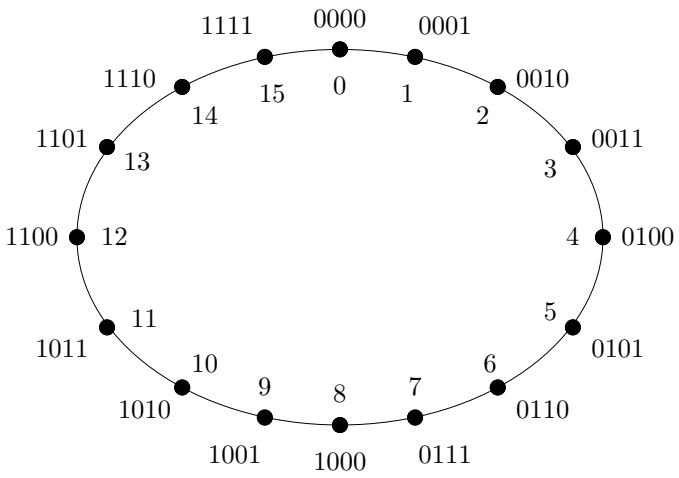
iteração	f_i	$f_i \times b$	d_{-i}
1	0.407	0.814	0
2	0.814	1.628	1
3	0.628	1.256	1
4	0.256	0.512	0
5	0.512	1.024	1
6	0.024	0.048	0
7	0.048	0.096	0
8	0.096	0.192	0
9	0.192	0.384	0
10	0.384	0.768	0
11	0.768	1.536	1
12	0.536	1.072	1
13	0.072	0.144	0
14	...	...	...

E assim $f_{10} = 0.407_{10} = 0.110100000110\ldots_2$ e finalmente $X_{10} = 4.407_{10} = 100.110100000110\ldots_2$.

Observe que um número com um número finito de dígitos em uma base pode tornar-se uma dízima periódica em outra base; isso ocorre com bastante freqüência.

1.3 Representação de números com sinal

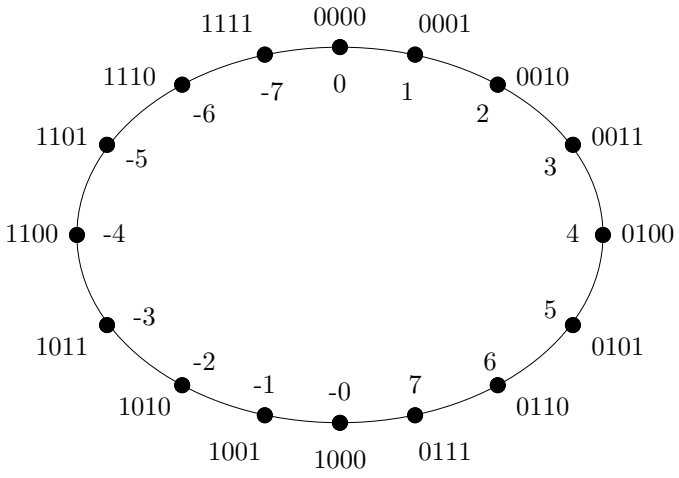
Usando números binários com d bits podemos representar até 2^d valores distintos. Por exemplo, em uma posição de memória de 8 bits podemos representar até 256 valores distintos. Em um registrador de 4 bits podemos ter os valores inteiros positivos de 0 a 15, como mostra a figura a seguir:



Caso desejarmos representar valores negativos, podemos empregar diversas abordagens, como veremos na seqüência.

1.3.1 Usando sinal e grandeza

Podemos empregar o bit mais significativo (MSB) para indicar o sinal ($0 = +$ e $1 = -$) e os demais bits para a grandeza do valor. Assim teriamos:



A faixa de valores representados vai de -7 a $+7$ ($\pm(2^{d-1} - 1)$). A grande desvantagem desta técnica é a dupla representação do zero ($+0$ e -0). Esta técnica também é chamada *sinal-magnitude*.

1.3.2 Usando números complementares

Os números binários manipulados por um processador tem um número n finito e normalmente constante de dígitos (8, 16, 32, ...). As operações aritméticas entre esses números são então efetuadas sempre em módulo $M = 2^n$. Para uma contagem usando 3 bits teríamos então:

$$000 \rightarrow 001 \rightarrow 010 \rightarrow 011 \rightarrow 100 \rightarrow 101 \rightarrow 110 \rightarrow 111 \rightarrow 000 \rightarrow 001 \rightarrow \cdots$$

Veremos agora que através dos números chamados *complementares* é possível implementar a subtração usando as operações de soma em módulo, o que permite simplificar bastante os circuitos necessários às operações aritméticas. Existem dois tipos de números complementares: o *complemento 2* (C2) e o *complemento 1* (C1), que estudaremos a seguir.

1.3.3 Complemento 2

Tendo sido especificado o número de dígitos d, e o módulo $M = 2^d$, o complemento 2 (C2) de um número n, indicado por $\overline{n}^2$, é definido por:

$$\overline{n}^2 = M - n \bmod M$$

Por exemplo, sendo um número de 4 bits $n = 5_{10} = 0101_2$, nosso módulo M vale $2^4 = 10000_2$ e o complemento 2 de n é dado por:

$$\overline{n}^2 = 2^4 - 5 \bmod 16 = 10000_2 - 0101_2 = 1011_2$$

Vejamos os complementos 2 dos primeiros números com 4 bits:

n_{10}	n_2	$2^4 - n$	$\overline{n}^2$
0	0000	10000	0000
1	0001	1111	1111
2	0010	1110	1110
3	0011	1101	1101
4	0100	1100	1100
5	0101	1011	1011

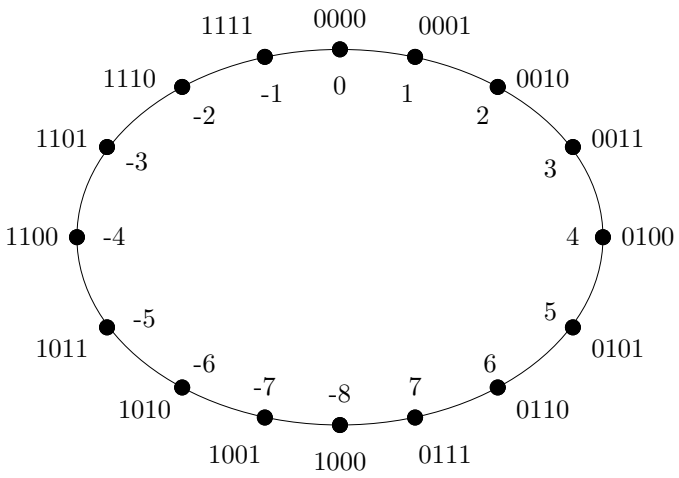
Ao invés de efetuarmos operações de subtração em binário, podemos calcular $\overline{n}^2$ através de uma simples regra prática:

1. O complemento 2 $\overline{n}^2$ tem o mesmo número de bits d que o número n.
2. Percorrendo n da direita para a esquerda (MSB $\leftarrow$ LSB), preservar todos os bits até o primeiro "1" (inclusive), e complementar os demais.

Vejamos alguns exemplos:

d	n	$\overline{n}^2$
4	1001	0111
7	1101100	0010100
3	101	011
1	1	1
8	11010110	00101010

Vamos agora representar números negativos usando esta técnica: o bit mais significativo continua representando o sinal, mas os números negativos são representados pelos complementos 2 dos números positivos correspondentes. Por exemplo, se $n_{10} = -2$, e $2_{10} = 0010_2$ então $n_2 = \overline{0010}^2 = 1110$. No sentido contrário, caso $n_2 = 1101$ então o número é negativo (MSB vale 1) e $\mid n_{10} \mid = \overline{n}_2^2 = \overline{1101}^2 = 0011 = 3_{10}$ e finalmente $n_{10} = -3$. O uso deste método em números de 4 bits está ilustrado na figura a seguir:



1.3.4 Complemento 1

O complemento 1 (C1) de um número binário n com d bits, indicado por $\overline{n}^1$, é definido por:

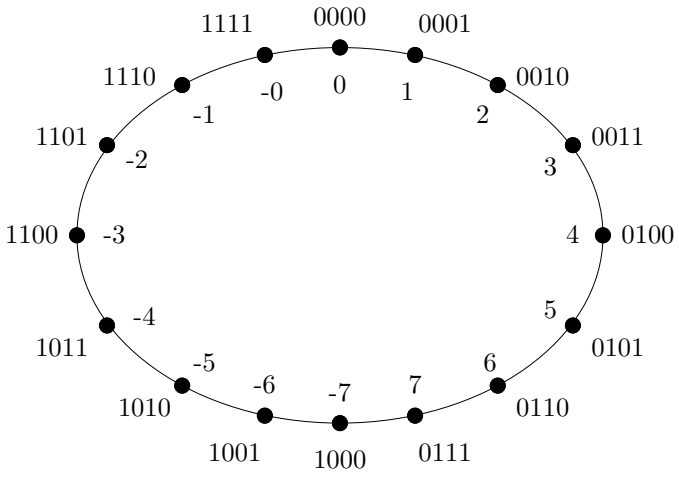
$$\overline{n}^1 = (2^d - 1) - n \bmod M$$

Como regra prática, basta complementar todos os bits do número n, como mostram os exemplos a seguir:

d	n	$\overline{n}^1$
4	1001	0110
7	1101100	0010011
3	101	010
1	1	0
8	11010110	00101001

A diferença entre os complementos 1 e 2 é de uma unidade: $\overline{n}^1 = \overline{n}^2 - 1$ ou $\overline{n}^2 = \overline{n}^1 + 1$.

Usando esta técnica, os números negativos podem ser representados pelos complementos 1 dos números positivos correspondentes. Assim temos:



Por exemplo, se $n_{10} = -3$ então $n_2 = \overline{0011}^1 = 1100$ (usando 4 bits temos $3_{10} = 0011_2$). De modo inverso se $n_2 = 1101$ então $n_{10} < 0$ e $\overline{n}_2^1 = \overline{1101}^1 = 0010 = 2_{10}$. Assim temos $n_{10} = -2$.

É importante observar que, em todas as técnicas apresentadas, quando $MSB(n_2) = 1$ então n_{10} é negativo.

1.4 Operações aritméticas

1.4.1 Aritmética binária

As operações aritméticas básicas em código binário seguem o mesmo mecanismo que as operações em base 10. Por exemplo, vejamos a operação de soma entre os números 11011011_2 e 10110000_2:

$$\begin{array}{cccccccccc} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & & & & & \\ & & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ + & & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}$$

Em muitos casos o resultado de uma operação deve ser mantido dentro de um determinado número de bits, e para isso os bits **mais significativos** em excesso devem ser descartados. Se quisermos manter o resultado da soma acima em 8 bits, o resultado a ser considerado deve ser 10001011_2.

Deve ser tomado muito cuidado em relação à codificação usada. Se estivermos considerando números com sinal (C1, C2, etc), a soma de dois números positivos pode ter seu bit mais significativo ativado e assim indicar um número negativo, como mostra o exemplo a seguir:

$$\begin{array}{cccccccccc} & & {\scriptstyle 1} & {\scriptstyle 1} & & & & & & \\ & & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ + & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ \hline & & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}$$

Da mesma maneira, a soma de dois números negativos pode provocar um excesso (que será descartado para manter o resultado em n bits) e assim gerar um resultado positivo.

A subtração entre dois números binários pode ser facilmente realizada através da soma do primeiro com o complemento (C1, C2) do segundo. O resultado deve ser considerado usando a mesma codificação de complemento. Vejamos por exemplo a operação $X = 37_{10} - 86_{10}$, que em binário fica $00100101_2 - 01010110_2$ (vamos considerar valores em 8 bits e usar uma codificação em C2):

$$\begin{aligned} X &= 00100101 - 01010110 \\ &= 00100101 + \overline{01010110}^2 \\ &= 00100101 + 10101010 \\ &= 11001111 \\ &= -(\overline{11001111}^2) \\ &= -00110001 \\ &= -49_{10} \end{aligned}$$

Assim como a soma, a subtração também pode ser afetada por eventuais excessos que devam ser descartados. Embora bem mais complexas, as operações de produto e divisão seguem os mecanismos conhecidos para a base decimal.

1.4.2 Aritmética hexadecimal

1.4.3 Aritmética em BCD

1.5 Outros códigos importantes

Embora os sistemas digitais sejam sempre binários, isto é, todos os sinais existentes no sistema só podem assumir dois valores, em alguns casos específicos é interessante a utilização de outras codificações binárias distintas da clássica, devido a certas vantagens oferecidas por estas.

1.5.1 Decimal Codificado em Binário (BCD)

Neste código cada dígito de um número decimal é codificado na forma de um número binário. Para representar os dez dígitos ($0 \cdots 9$) são necessários 4 bits. Assim, o número 347_{10} seria codificado como $\underbrace{0011}_{3}\underbrace{0100}_{4}\underbrace{0111}_{7}$. Observe que usando uma codificação binária clássica teríamos $347_{10} = 101011011_2$. Os números em BCD são mais longos que os binários normais, pois cada dígito decimal é representado separadamente. Um dos principais usos da codificação em BCD encontra-se na implementação de *displays* numéricos de calculadoras, relógios, etc.

1.5.2 Código Gray

Este código também é chamado de "código espelhado" e caracteriza-se pelo fato de que a representação de dois números consecutivos nunca difere em mais que um bit. A tabela a seguir indica a construção dos primeiros números neste código:

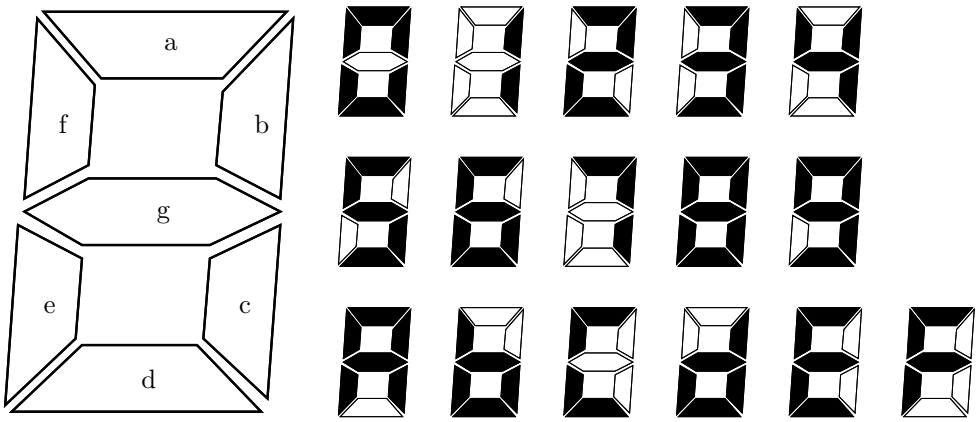
Decimal	Binário	Gray	Decimal	Binário	Gray
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

O código Gray pode ser importante em situações onde é necessário minimizar as transições de bits no sistema (por questões de velocidade e imunidade a ruídos). Por exemplo, para passar de 7 a 8 no sistema binário clássico são necessárias 4 transições de bits ($0111 \rightarrow 1000$) enquanto usando o código Gray apenas uma transição é necessária ($0100 \rightarrow 1100$).

1.5.3 Código 7 segmentos

Este código está relacionado com os mostradores de 7 segmentos usados normalmente em calculadoras e outros aparelhos simples, para a apresentação de resultados. Um mostrador desse

tipo é construído usando sete segmentos luminosos que podem ser combinados para representar os dígitos de 0 a 9, e as letras de "A" a "F", como mostra a figura a seguir:



Considerando um segmento iluminado como tendo o valor "1", temos a seguinte tabela para os dígitos hexadecimais n código de 7 segmentos:

nº	a	b	c	d	e	f	g	nº	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0	8	1	1	1	1	1	1	1
1	0	1	1	0	0	0	0	9	1	1	1	1	0	1	1
2	1	1	0	1	1	0	1	A	1	1	1	0	1	1	1
3	1	1	1	1	0	0	1	B	0	0	1	1	1	1	1
4	0	1	1	0	0	1	1	C	1	0	0	1	1	1	0
5	1	0	1	1	0	1	1	D	0	1	1	1	1	0	1
6	1	0	1	1	1	1	1	E	1	0	0	1	1	1	1
7	1	1	1	0	0	0	0	F	1	0	0	0	1	1	1

1.5.4 Código ASCII

1.6 Exercícios

1. Converta os seguintes números para as bases binária, octal, decimal e hexadecimal: 735_{10}, 10 0110 1001_2, 1997_{10}, 01 1001 0110_2, $2AC_{16}$, 0110 1101_2, $5BD_{16}$ e 7296.12_{10}.

2. Indique quais dos seguintes números são hexadecimais válidos: BED, CAB, $DEAD$, BAG e $F0CA$.

3. Quantos números inteiros positivos podem ser expressos usando k dígitos em uma base b?

4. Expresse os seguintes números decimais no sistema *sinal-magnitude*, com módulo de 8 bits: $+100$, -56, -127, -1, 0, $+1$ e $+127$.

5. Idem ao anterior, usando os sistemas *complemento 2* e *complemento 1*. Analise e compare a representação dos números em cada um desses sistemas. Quais são as vantagens e desvantagens de cada um deles?

6. Os seguintes números positivos e negativos correspondem a representações em diferentes bases, ajustados em 16 bits (ou seja, módulo 16). Encontre as representações decimais,

considerando que todos estão representados no sistema C2: $1101\ 0011\ 0110\ 1101_2$, $FA31_H$ e $0100\ 1111\ 0010\ 1011_2$.

7. Passando previamente os valores à base binária, calcular $X + Y$ e $X - Y$, considerando os números em C2. Indicar em que situação ocorre *overflow* ("estouro" da capacidade de um registro de n bits).

   a) em 8 bits: $X = B7_H$, $Y = 6C_H$ b) em 16 bits: $X = F7A_H$, $Y = 28513_{10}$

8. Resolver as seguintes operações:

   a) $S_1 = 10101_2 + 100100_2$
   b) $S_2 = 10000000_2 + 2A_H$
   c) $S_3 = 32_{10} + 1101000_2$
   d) $S_4 = 1100 \times 0011$

Capítulo 2

Álgebra de Boole

2.1 Introdução

Uma variável booleana pode assumir somente dois valores: 0 (*falso*) e 1 (*verdadeiro*). Portanto, se $x \neq 0$ então $x = 1$ e se $x \neq 1$ então $x = 0$. Chamamos de *função booleana* uma relação $\mathcal{F}$ entre duas ou mais variáveis booleanas $x_1, x_2, \ldots, x_n$, realizada com as operações de soma "+" e produto "$\cdot$" (muitas vezes $A \cdot B$ é representado na forma AB).

A álgebra de Boole apresenta uma série de propriedades que nos serão úteis na manipulação de funções lógicas, que estudaremos a seguir. Sendo A, B e C variáveis booleanas (também chamadas *variáveis lógicas*), temos:

1. Propriedade Comutativa:

$$\begin{aligned} A \cdot B &= B \cdot A \\ A + B &= B + A \end{aligned}$$

2. Propriedade associativa:

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

3. Propriedade distributiva:

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

Além das propriedades acima, uma série de *postulados* (ou leis fundamentais) rege a Álgebra de Boole. Esses postulados podem ser facilmente provados, mas isto foge ao objetivo deste curso. Os postulados básicos que utilizaremos são:

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

2. $1 + 1 = 1$
   $0 \cdot 0 = 0$

3. $0 + 0 = 0$
   $1 \cdot 1 = 1$

4. $0 + 1 = 1$
   $0 \cdot 1 = 0$

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

Além dos postulados, alguns *teoremas* envolvendo variáveis booleanas nos serão úteis:

1. $A + 0 = A$
   $A \cdot 1 = A$

2. $A + 1 = 1$
   $A \cdot 0 = 0$

3. $A + A = A$
   $A \cdot A = A$

4. $\overline{(\overline{A})} = A$

5. $A + \overline{A} = 1$
   $A \cdot \overline{A} = 0$

6. Teoremas de Morgan:
   $\overline{A + B + C + \cdots} = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \cdots$
   $\overline{A \cdot B \cdot C \cdot \cdots} = \overline{A} + \overline{B} + \overline{C} + \cdots$

7. $A(A + B) = A + AB = A$
   $A + AB = A(A + B) = A$

A verificação da validade dos teoremas é bastante simples, bastando construir as tabelas-verdade com todos os valores possíveis para as variáveis envolvidas.

Tanto os teoremas quanto os postulados se apresentam aos pares. Para cada um deles, seu par respectivo pode ser obtido trocando-se os "+" por "$\cdot$" e os "0" por "1", e vice-versa. Essa propriedade é chamada *dualidade*. Por exemplo, $A + 0 = A \Longleftrightarrow A \cdot 1 = A$.

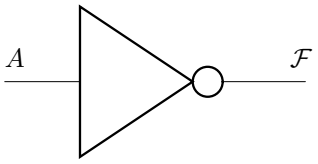
2.2 Funções Lógicas e Tabelas-Verdade

As funções lógicas ou booleanas básicas são descritas abaixo, com seus símbolos e suas tabelas-verdade. Uma tabela-verdade é simplesmente uma tabela enumerando todos os possíveis estados das entradas de uma função e seu respectivo valor de saída. As funções lógicas podem tambem ser chamadas *portas lógicas*, sobretudo quando implementadas em circuitos.

Função NÃO : também chamada *inversor* ou *NOT*, e representada por $\mathcal{F}(A) = \overline{A}$. Sua tabela-verdade é dada por:

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

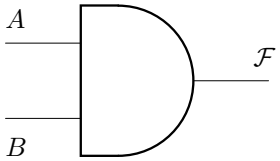
Em diagramas de circuitos digitais um inversor é representado pelo símbolo abaixo:



Função E : também denominada *AND*, assume valor verdadeiro se e somente se ambas as entradas são verdadeiras. É representada por $\mathcal{F}(A, B) = A \cdot B$ ou simplesmente AB (em lógica também pode ser encontrada a forma $A \wedge B$). Sua tabela-verdade é:

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

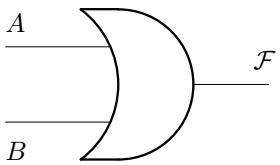
Seu símbolo em diagramas é:



Função OU : também denominada *OR*, esta função é verdadeira se alguma de suas entradas o for: $\mathcal{F}(A, B) = A + B$ (em lógica também pode ser encontrada a forma $A \vee B$). Sua tabela-verdade é:

A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1

Seu símbolo em diagramas é:

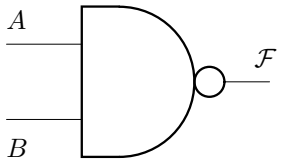


Além das funções básicas E, OU e NÃO, algumas outras funções simples que podem ser derivadas destas também nos serão úteis. São elas:

Função não-E : também chamada *NAND*, é a inversa da função E: $\mathcal{F}(A, B) = \overline{A \cdot B}$.

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

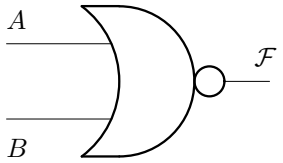
Seu símbolo em diagramas é:



Função não-OU : também chamada *NOR*, é a inversa da função OU : $\mathcal{F}(A, B) = \overline{A + B}$.

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

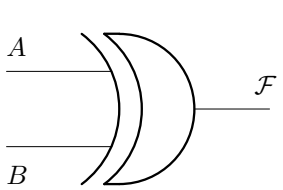
Seu símbolo em diagramas é:



Função OU-exclusivo : também chamada *XOR*, é indicada pelo operador "$\oplus$", e é verdadeira somente se suas duas entradas forem opostas: $\mathcal{F}(A, B) = A \oplus B = \overline{A}B + A\overline{B}$.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Seu símbolo em diagramas é:



As funções *Não-E* e *Não-OU* são conhecidas como *funções universais*, por serem freqüentemente empregadas na síntese das demais funções, devido à simplicidade de sua implementação interna.

2.3 Interpolação de Lagrange

Uma ferramenta muito útil na resolução de problema lógicos é a interpolação de Lagrange. Para uma função lógica $\mathcal{F}$ desconhecida com n variáveis, a partir de todas as combinações de suas variáveis de entrada e dos respectivos valores de saída, ela nos permite encontrar um polinômio representando a função $\mathcal{F}$. Por exemplo, para $n = 1$ temos a seguinte tabela-verdade:

x	$\mathcal{F}(x)$
0	$\mathcal{F}(0)$
1	$\mathcal{F}(1)$

O que nos dá o seguinte polinômio de Lagrange: $\mathcal{F}(x) = \mathcal{F}(0)\overline{x} + \mathcal{F}(1)x$. De modo similar, se $n = 2$ temos a seguinte tabela-verdade:

x	y	$\mathcal{F}(x,y)$
0	0	$\mathcal{F}(0,0)$
0	1	$\mathcal{F}(0,1)$
1	0	$\mathcal{F}(1,0)$
1	1	$\mathcal{F}(1,1)$

Que resulta no seguinte polinômio:

$$\mathcal{F}(x,y) = \mathcal{F}(0,0)\overline{x}\,\overline{y} + \mathcal{F}(0,1)\overline{x}\,y + \mathcal{F}(1,0)x\,\overline{y} + \mathcal{F}(1,1)x\,y$$

De maneira genérica, se tivermos valores da forma:

x_1	x_2	x_3	$\cdots$	x_n	$\mathcal{F}(x_1, x_2, x_3, \ldots, x_n)$
0	0	$\cdots$	0	0	$\mathcal{F}(0,0,\ldots,0,0)$
0	0	$\cdots$	0	1	$\mathcal{F}(0,0,\ldots,0,1)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
1	1	$\cdots$	1	0	$\mathcal{F}(1,1,\ldots,1,0)$
1	1	$\cdots$	1	1	$\mathcal{F}(1,1,\ldots,1,1)$

O que nos dá o seguinte polinômio genérico:

$$\begin{aligned}
\mathcal{P}(x_1, x_2, x_3, \ldots, x_n) &= \mathcal{F}(0,0,0,\ldots,0)\overline{x}_1\overline{x}_2\overline{x}_3\cdots\overline{x}_n \\
&+ \mathcal{F}(0,0,0,\ldots,1)\overline{x}_1\overline{x}_2\overline{x}_3\cdots x_n \\
&+ \cdots \\
&+ \mathcal{F}(1,1,1,\ldots,0)x_1x_2x_3\cdots\overline{x}_n \\
&+ \mathcal{F}(1,1,1,\ldots,1)x_1x_2x_3\cdots x_n
\end{aligned}$$

Vejamos agora um exemplo. Dada a seguinte tabela-verdade para uma função $\mathcal{F}(x, y)$, podemos determiná-la através do polinômio de Lagrange:

x	y	$\mathcal{F}(x,y)$
0	0	0
0	1	0
1	0	0
1	1	1

O polinômio resultante tem a seguinte forma:

$$\begin{aligned}\mathcal{F}(x,y) &= \mathcal{F}(0,0)\overline{x}\,\overline{y} + \mathcal{F}(0,1)\overline{x}\,y + \mathcal{F}(1,0)x\,\overline{y} + \mathcal{F}(1,1)x\,y \\ &= 0\cdot\overline{x}\,\overline{y} + 0\cdot\overline{x}\,y + 0\cdot x\,\overline{y} + 1\cdot x\,y \\ \mathcal{F}(x,y) &= x\,y\end{aligned}$$

2.4 Formas-padrão para expressões lógicas

Neste ítem estudaremos as formas-padrão nas quais as expressões lógicas podem ser escritas. Estas nos serão úteis mais tarde, para simplificar o estudo das funções lógicas. Existem basicamente duas formas: a *soma padrão de produtos* e o *produto padrão de somas*.

2.4.1 Produto-padrão de somas

Neste caso a função lógica é expressa como um produto de termos, cada termo sendo uma soma envolvendo todas as variáveis. Por exemplo:

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

Cada termo da expressão é chamado *maxitermo*, e a expressão em sua forma mínima é chamada *forma canônica de maxitermos*.

Esta forma apresenta a seguinte propriedade: $\mathcal{F} = 0$ se ao menos um dos maxitermos for zero, e um maxitermo será zero somente se todas as suas variáveis forem zero. Assim, pode-se concluir que cada maxitermo corresponde a uma linha da tabela-verdade na qual $\mathcal{F} = 0$. Veremos como empregar esta propriedade na seção 2.4.4.

2.4.2 Soma-padrão de produtos

Neste caso a função lógica é expressa como uma soma de termos, cada termo sendo um produto envolvendo todas as variáveis. Por exemplo:

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

Cada termo da expressão é chamado *minitermo*, e a expressão em sua forma mínima é chamada *forma canônica de minitermos*.

Esta forma apresenta a seguinte propriedade: $\mathcal{F} = 1$ se ao menos um dos minitermos for 1, e um minitermo será 1 somente se todas as suas variáveis forem 1. Assim, pode-se concluir que cada minitermo corresponde a uma linha da tabela-verdade na qual $\mathcal{F} = 1$. Veremos como empregar esta propriedade na seção 2.4.4.

2.4.3 Expansão às formas-padrão

Dada uma função lógica expressa sob a forma de uma soma de produtos, é possível expandi-la para a forma padrão aplicando os teoremas da lógica booleana, como mostra o exemplo abaixo:

$$\begin{aligned} Y &= AB + AC + \overline{B}C \\ &= AB \times 1 + AC \times 1 + \overline{B}C \times 1 \\ &= AB(C + \overline{C}) + AC(B + \overline{B}) + \overline{B}C(A + \overline{A}) \\ &= ABC + AB\overline{C} + ABC + A\overline{B}C + A\overline{B}C + \overline{A}\ \overline{B}C \\ &= ABC + A\overline{B}C + AB\overline{C} + \overline{A}\ \overline{B}C \end{aligned}$$

Da mesma forma, um produto de somas pode ser expandido para a forma padrão, como mostra o exemplo:

$$\begin{aligned} Y &= (P + \overline{Q})(Q + R)(P + \overline{R}) \\ &= (P + \overline{Q} + 0)(Q + R + 0)(P + \overline{R} + 0) \\ &= (P + \overline{Q} + R\overline{R})(P\overline{P} + Q + R)(P + Q\overline{Q} + \overline{R}) \\ &= (P + \overline{Q} + R)(P + \overline{Q} + \overline{R})(P + Q + R)(\overline{P} + Q + R)(P + Q + \overline{R})(P + \overline{Q} + \overline{R}) \\ &= (P + \overline{Q} + R)(P + \overline{Q} + \overline{R})(P + Q + R)(\overline{P} + Q + R)(P + Q + \overline{R}) \end{aligned}$$

2.4.4 Especificação de funções por maxitermos e minitermos

Veremos agora como utilizar as formas padrão de maxitermos e minitermos para representar funções lógicas. Dada uma função envolvendo n variáveis, teremos 2^n minitermos (indicados por m_i) e 2^n maxitermos (indicados por M_i) distintos. Veja o exemplo abaixo para três variáveis:

A	B	C	minitermo	maxitermo
0	0	0	$\overline{A}\ \overline{B}\ \overline{C} = m_0$	$A + B + C = M_0$
0	0	1	$\overline{A}\ \overline{B}\ C = m_1$	$A + B + \overline{C} = M_1$
0	1	0	$\overline{A}\ B\ \overline{C} = m_2$	$A + \overline{B} + C = M_2$
0	1	1	$\overline{A}\ B\ C = m_3$	$A + \overline{B} + \overline{C} = M_3$
1	0	0	$A\ \overline{B}\ \overline{C} = m_4$	$\overline{A} + B + C = M_4$
1	0	1	$A\ \overline{B}\ C = m_5$	$\overline{A} + B + \overline{C} = M_5$
1	1	0	$A\ B\ \overline{C} = m_6$	$\overline{A} + \overline{B} + C = M_6$
1	1	1	$A\ B\ C = m_7$	$\overline{A} + \overline{B} + \overline{C} = M_7$

Dada sua tabela-verdade, podemos expressar uma função sob a forma de maxitermos ou minitermos. Por exemplo, para a função lógica definida pela tabela-verdade abaixo:

A	B	C	$\mathcal{F}(A,B,C)$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Teremos sua expressão sob a forma de minitermos (nos quais $\mathcal{F} = 1$) dada por:

$$\begin{aligned}\mathcal{F}(A,B,C) &= \overline{A}\ \overline{B}\ \overline{C} + \overline{A}B\overline{C} + \overline{A}BC + ABC \\ &= m_0 + m_2 + m_3 + m_7 \\ &= \sum m(0,2,3,7)\end{aligned}$$

E a mesma função sob a forma de maxitermos (nos quais $\mathcal{F} = 0$) fica:

$$\begin{aligned}\mathcal{F}(A,B,C) &= (A+B+\overline{C})(\overline{A}+B+C)(\overline{A}+B+\overline{C})(\overline{A}+\overline{B}+C) \\ &= M_1 \cdot M_4 \cdot M_5 \cdot M_6 \\ &= \prod M(1,4-6)\end{aligned}$$

2.5 Mapas de Karnaugh

O mapa de Karnaugh de uma função lógica, também chamado mapa K, é uma representação gráfica da tabela-verdade da função (ou seja, de todas as combinações de todas as variáveis da função), expressa sob a forma de minitermos e/ou maxitermos. O ordenamento das células adjacentes em um mapa de Karnaugh segue o código Gray. A principal utilidade dos mapas de Karnaugh é a simplificação de expressões lógicas, como veremos na próxima seção.

O mapa de Karnaugh de uma função pode ser construído diretamente a partir de sua tabela verdade. Para uma função $\mathcal{F}$ com duas variáveis A e B teríamos a seguinte tabela-verdade e o respectivo mapa de Karnaugh:

A	B	$\mathcal{F}(A,B)$
0	0	$\mathcal{F}(0,0) = m_0$
0	1	$\mathcal{F}(0,1) = m_1$
1	0	$\mathcal{F}(1,0) = m_2$
1	1	$\mathcal{F}(1,1) = m_3$

B \ A	0	1
0	m_0	m_2
1	m_1	m_3

Para funções com três e quatro variáveis teríamos os seguintes mapas:

C \ AB	00	01	11	10
0	m_0	m_2	m_6	m_4
1	m_1	m_3	m_7	m_5

CD \ AB	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

Vejamos como exemplo a função abaixo, representada através de uma tabela-verdade e de seu respectivo mapa de Karnaugh:

A	B	$\mathcal{F}(A,B)$
0	0	1
0	1	0
1	0	0
1	1	1

B \ A	0	1
0	1	
1		1

Vejamos outro exemplo:

A	B	C	$\mathcal{F}(A,B,C)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

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

No caso acima podemos escrever $\mathcal{F} = \sum m(0,1,2,6,7) = \prod M(3,4,5)$. Deve-se observar que existe uma relação direta entre os minitermos/maxitermos e os quadros do mapa de Karnaugh. Além disso, em um mapa de Karnaugh representamos somente os minitermos (1) ou os maxitermos (0), por razões de clareza (para o processo de simplificação estaremos interessados em associar somente os minitermos ou somente os maxitermos).

Para construir um mapa de Karnaugh com mais de quatro variáveis construímos um mapa básico com quatro variáveis e o repetimos tantas vezes quantas forem as combinações das variáveis restantes, formando assim um mapa externo. O exemplo abaixo mostra a construção de um mapa de Karnaugh para uma função de seis variáveis:

B \ A	0					1				
	CD \ EF	00	01	11	10	CD \ EF	00	01	11	10
	00					00				
0	01					01				
	11					11				
	10					10				
	CD \ EF	00	01	11	10	CD \ EF	00	01	11	10
	00					00				
1	01					01				
	11					11				
	10					10				

2.6 Simplificação de funções lógicas

A minimização de funções lógicas é importante para minimizar os custos de implementação dos circuitos digitais e melhorar sua velocidade de trabalho (a cada operação lógica está associado

uma duração). Podemos encontrar uma forma mínima para uma função através dos mapas de Karnaugh ou através da aplicação dos postulados e teoremas da álgebra de Boole. Vejamos dois exemplos de simplificação usando a álgebra de Boole:

$$\begin{aligned} Y &= \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C} + ABC \\ &= \overline{A}B(\overline{C} + C) + AB(\overline{C} + C) \\ &= \overline{A}B + AB \\ &= B(\overline{A} + A) \\ &= B \end{aligned}$$

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

Para a simplificação usando os mapas de Karnaugh, nos baseamos em sua principal propriedade que é o fato de que dois quadros adjacentes (na horizontal ou na vertical) em um mapa de Karnaugh correspondem a dois minitermos (ou dois maxitermos) nos quais apenas uma variável difere, estando complementada em um deles e não no outro. Esses dois minitermos adjacentes podem ser combinados com a remoção da variável que difere entre ambos, como mostra o exemplo a seguir, para $\mathcal{F} = m_8 + m_{12}$:

CD \ AB	00	01	11	10
00			1	1
01				
11				
10				

Podemos escrever $\mathcal{F} = A\overline{B}\ \overline{C}\ \overline{D} + AB\overline{C}\ \overline{D} = A\overline{C}\ \overline{D}(B + \overline{B}) = A\overline{C}\ \overline{D}$. Assim, dois termos envolvendo quatro variáveis puderam ser substituídos por um só termo envolvendo três variáveis. Como regra geral, um conjunto regular de 2^n minitermos adjacentes (ou seja, um retângulo com 1, 2, 4, 8, ... minitermos) pode ser agrupado em um termo único do qual podem ser eliminadas n variáveis. Por exemplo, para o mapa abaixo:

CD \ AB	00	01	11	10
00				
01	1	1		
11	1	1		
10				

Podemos escrever $\mathcal{F} = m_1 + m_3 + m_5 + m_7 = \overline{A}D$. É importante observar que, devido ao uso do código Gray, as colunas e linhas externas são adjacentes entre si, como mostra o exemplo abaixo, no qual encontramos dois grupos com quatro termos cada:

CD \ AB	00	01	11	10
00		1	1	
01	1			1
11	1			1
10		1	1	

De acordo com o mapa acima podemos escrever: $\mathcal{F} = m_1 + m_3 + m_4 + m_6 + m_9 + m_{11} + m_{12} + m_{14} = \overline{B}D + B\overline{D} = B \oplus D$. Vejamos mais um exemplo:

CD \ AB	00	01	11	10
00	1			1
01				
11				
10	1			1

Neste caso $\mathcal{F} = m_0 + m_2 + m_8 + m_{10} = \overline{B}\ \overline{D}$. Ao construir a função mínima devemos associar entre si os termos de maneira a eliminar o maior número possível de variáveis, até cobrir todos os termos existentes no mapa. Às vezes um termo tem de ser usado em mais de uma associação, como mostra o exemplo abaixo, no qual temos $\mathcal{F} = BD + \overline{A}\ \overline{C}D$:

CD \ AB	00	01	11	10
00				
01	1	1	1	
11		1	1	
10				

Para levar a uma boa simplificação da função, devem ser seguidas as seguintes regras na construção dos grupos:

- Minimizar o número de grupos (para minimizar o número de termos na expressão final).
- Cada grupo deve englobar o maior número possível de casas (para eliminar o maior número possível de variáveis no termo correspondente ao grupo, na expressão final).

Por exemplo, no mapa abaixo as associações da esquerda levam a uma expressão mínima da função ($\mathcal{F} = \overline{A}\ \overline{C}D + AB\overline{C} + ACD + \overline{A}BC$), enquanto as associações da direita não ($\mathcal{F} = BD + \overline{A}\ \overline{B}\ \overline{C}D + AB\overline{C}D + A\overline{B}CD + \overline{A}BC\overline{D}$):

CD \ AB	00	01	11	10
00			1	
01	1	1	1	
11		1	1	1
10		1		

CD \ AB	00	01	11	10
00			1	
01	1	1	1	
11		1	1	1
10		1		

Caso sejam empregados maxitermos ao invés de minitermos, devemos tentar agrupar os zeros do mapa de Karnaugh, lembrando que a regra de complemento funciona ao contrário. Veja o exemplo:

CD \ AB	00	01	11	10
00	0	0		
01	0	0		
11			0	0
10				

Podemos escrever $\mathcal{F} = M_0 \cdot M_1 \cdot M_4 \cdot M_5 \cdot M_{11} \cdot M_{15} = (A + C) \cdot (\overline{A} + \overline{C} + \overline{D})$.

2.7 Funções incompletamente especificadas

Uma função lógica é dita incompletamente especificada quando, para uma dada combinação das variáveis de entrada, a saída é irrelevante, podendo assumir 0 ou 1. Isso ocorre quando:

- certas combinações das variáveis de entrada nunca ocorrem;
- as entradas existem em circunstâncias tais que não influenciam no comportamento global do sistema.

Nestas condições a função pode assumir 0 ou 1 na tabela-verdade ou no mapa de Karnaugh, sendo então indicada por um "X". Podemos considerar a função nessas condições como 0 ou 1, dependendo da conveniência (por exemplo, para auxiliar os agrupamentos no mapa de Karnaugh). Vejamos o exemplo da função dada por $\mathcal{F} = \sum m(1, 2, 5, 6, 9) + \sum X(10, 11, 12, 13, 14, 15)$ e seu respectivo mapa de Karnaugh:

CD \ AB	00	01	11	10
00				
01	1	1	X	1
11			X	X
10	1	1	X	X

Se interpretarmos como 0 os valores de m_{11} e m_{15} e como 1 os demais valores indeterminados, então podemos agrupar os termos em dois grupos e teremos a forma mínima $\mathcal{F} = \overline{C}D + C\overline{D} = C \oplus D$.

2.8 Exercícios

1. Utilize os postulados e teoremas da álgebra de Boole para simplificar as seguintes expressões lógicas:

   a) $S = W(X + Y(Z + \overline{W}))$
   b) $S = (W + X + Y)(W\overline{X} + Y)(\overline{Y} + Z)(W + Z)$
   c) $S = \overline{(X(Y + W)Z)}\ \overline{(X + Y)}$
   d) $S = X + (\overline{X}\ \overline{Y} + \overline{X}\ \overline{X})$

2. Indique quais das seguintes afirmações são válidas:

a) a operação NAND não é associativa
b) $AB \oplus AB = AB$
c) $\overline{AB}\ \overline{AB} = AB$
d) $A\overline{C} + \overline{A}C = B \iff A\overline{B} + \overline{A}B = C$

3. Expresse as funções abaixo como somas de produtos, indicando também sua formas canônicas (somas padrão de produtos):

a) $X = (\overline{A} + BC)(B + \overline{C}D)$ b) $X = (A + \overline{BC})(\overline{D + BC})$

4. Idem ao anterior, mas usando produtos de somas:

a) $X = A + BC$
b) $X = (\overline{A} + BC)(B + \overline{C}D)$
c) $X = (A + B)(\overline{A} + C)(B + C)$

5. Encontrar as expressões duais das expressões abaixo:

a) $F = XYZ + \overline{XY}Z$
b) $F = (X + \overline{Y})Z + \overline{X}Y\overline{Z}$
c) $F = \overline{X}(Y + \overline{Z}(X + \overline{Y}))$

6. Minimize as seguintes funções usando o mapa de Karnaugh:

a) $F_1 = \sum m(0 - 2, 4)$
b) $F_2 = \sum m(1, 3, 5 - 7)$
c) $F_3 = \sum m(0 - 5, 8, 10, 12 - 15)$
d) $F_4 = \prod M(0, 1, 3, 5, 11, 13, 14)$
e) $F_5 = \sum m(0, 3, 5, 11, 13, 14)$
f) $F_6 = \prod M(5, 10, 11) + X(1 - 4)$

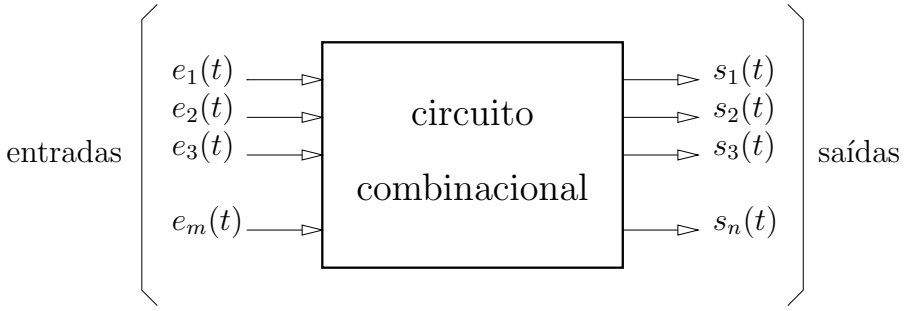
7. Uma função chamada *majoritária* produz uma saída lógica alta (1) quando a maioria de suas entradas está em nível alto (1). Considere e função para 4 entradas e encontre a expressão lógica correspondente através da tabela-verdade e do mapa de Karnaugh (escolha a forma mais conveniente: maxitermos ou minitermos).

Capítulo 3

Circuitos Combinacionais

3.1 Introdução

Este capítulo é dedicado ao estudo e projeto de sistemas lógicos combinacionais. Um circuito lógico é dito *combinacional* quando suas saídas em um determinado instante t são função unicamente de suas entradas naquele instante, ou seja, suas saídas são *combinações lógicas* de suas entradas naquele instante. Isso significa que qualquer modificação nas entradas será imediatamente considerada pelas saídas.



3.2 Síntese de circuitos combinacionais

O projeto de um circuito combinacional segue habitualmente os seguintes passos, que devem ser aplicados para cada uma das saídas do circuito:

1. Descrição do comportamento desejado para o sistema, através de uma tabela-verdade enumerando todos os estados possíveis para as entradas e os respectivos valores das saídas.
2. Construção do mapa de Karnaugh da função, a partir da tabela-verdade.
3. Obtenção da forma mínima para a função desejada, a partir dos agrupamentos de termos no mapa de Karnaugh.
4. Construção do circuito indicado pela expressão mínima da função, usando as portas lógicas básicas (*AND, OR, NOT*). Podem ser efetuadas manipulações sobre a expressão para permitir o uso de portas derivadas (*XOR*) ou universais (*NAND, NOR*).

No final deste capítulo veremos casos onde este roteiro não pode ser aplicado diretamente, devido ao grande número de entradas, que torna impraticável a definição de uma tabela-verdade

para o sistema inteiro (por exemplo, 8 entradas geram 256 estados possíveis). Entretanto os circuitos onde isso ocorre podem normalmente ser divididos em sub-sistemas, para os quais o roteiro acima pode ser empregado.

Vejamos o uso do roteiro acima em um exemplo: vamos projetar um detector de números primos de 4 bits, entre 0000_2 e 1111_2). Os números primos de 4 bits são: 1, 2, 3, 5, 7, 11 e 13, e nosso circuito dispõe de quatro entradas (os quatro dígitos do número binário) e uma saída (a indicação de que o número na entrada é primo), o que nos leva a seguinte tabela-verdade:

n	A	B	C	D	$\mathcal{F}$
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

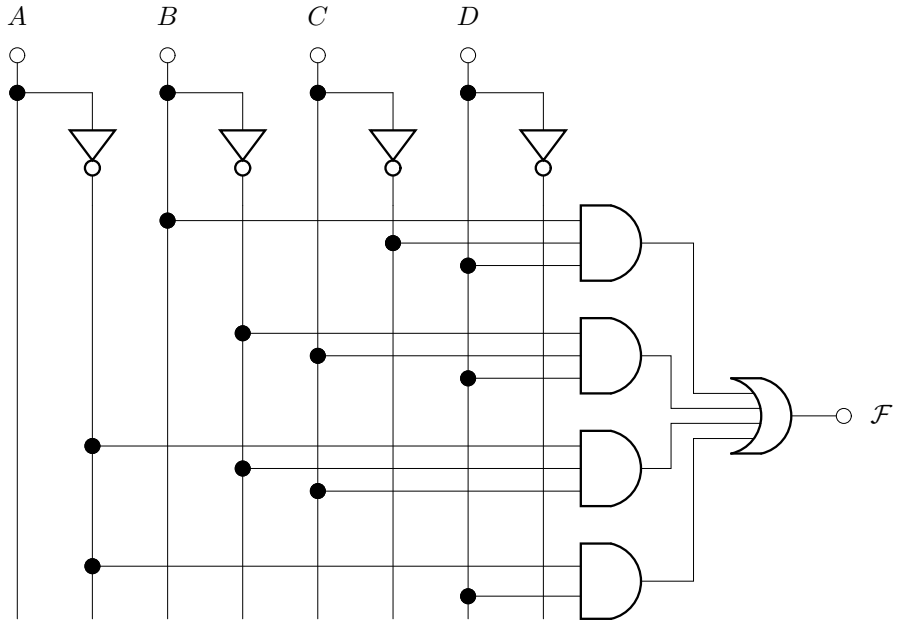
A partir da tabela-verdade acima podemos construir o seguinte mapa de Karnaugh para a função $\mathcal{F}$:

CD \ AB	00	01	11	10
00				
01	1	1	1	
11	1	1		1
10	1			

A expressão que pode ser obtida dos agrupamentos do mapa de Karnaugh tem a forma:

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

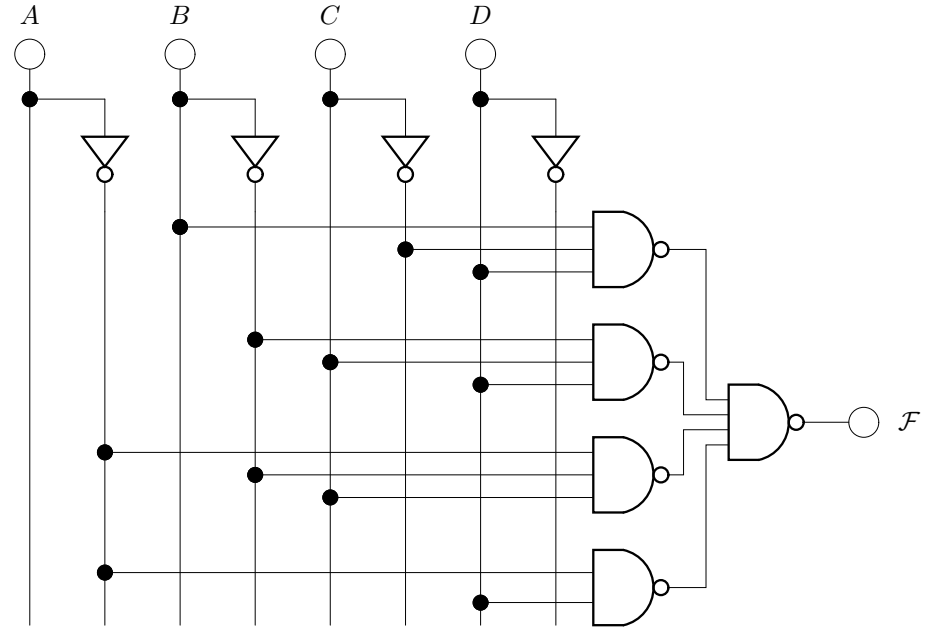
Podemos construir o circuito indicado pela expressão acima usando somente as portas lógicas básicas. Cada termo (produto) da expressão é implementado por uma porta *AND*, e a soma dos termos é implementada por uma porta *OR*, o que nos dá o seguinte circuito:



Podemos empregar os teoremas de Morgan ($\overline{A+B+C} = \overline{A}\,\overline{B}\,\overline{C}$) para transformar a função sob a forma de uma soma de produtos em um produto de complementos, eliminando as somas (e a porta *OR*):

$$\begin{aligned}
\mathcal{F} &= B\overline{C}D + \overline{B}CD + \overline{A}\,\overline{B}C + \overline{A}D \\
&= \overline{\overline{B\overline{C}D + \overline{B}CD + \overline{A}\,\overline{B}C + \overline{A}D}} \\
&= \overline{\overline{B\overline{C}D} \cdot \overline{\overline{B}CD} \cdot \overline{\overline{A}\,\overline{B}C} \cdot \overline{\overline{A}D}}
\end{aligned}$$

Desta forma a implementação do circuito passa a empregar somente portas *NAND*:



Embora implementado de outra forma, este circuito executa exatamente a mesma função que o anterior. Esta forma de implementação, empregando somente portas *NAND*, é chamada "lógica *NAND-NAND*".

3.3 Conversores de códigos

Os circuitos conversores de códigos se destinam à conversão entre diferentes sistemas de codificação de números binários, como o Gray, o BCD, o 7 segmentos, etc. A maioria destes circuitos se encontra disponível em circuitos integrados comerciais. Vejamos como exemplo a implementação de um conversor de números de 4 bits em código Gray para binário. A tabela-verdade da função de conversão possui quatro entradas (os 4 dígitos do número em código Gray) e quatro saídas (os quatro dígitos do código binário):

n	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

Com base na tabela-verdade podemos construir os mapas de Karnaugh para cada uma das saídas $B_3 \ldots B_0$. Começando com B_3 temos:

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

O que nos leva à expressão $B_3 = G_3$. Para B_2 teremos:

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

O que nos leva a $B_2 = \overline{G}_3 G_2 + G_3 \overline{G}_2 = G_3 \oplus G_2$. Para B_1 teremos:

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

$$\begin{aligned} B_1 &= \overline{G}_3\overline{G}_2 G_1 + \overline{G}_3 G_2 \overline{G}_1 + G_3\overline{G}_2\overline{G}_1 + G_3 G_2 G_1 \\ &= \overline{G}_3(\overline{G}_2 G_1 + G_2\overline{G}_1) + G_3(\overline{G}_2\overline{G}_1 + G_2 G_1) \\ &= \overline{G}_3(G_2 \oplus G_1) + G_3(\overline{G_2 \oplus G_1}) \\ &= G_3 \oplus (G_2 \oplus G_1) \\ B_1 &= G_3 \oplus G_2 \oplus G_1 \end{aligned}$$

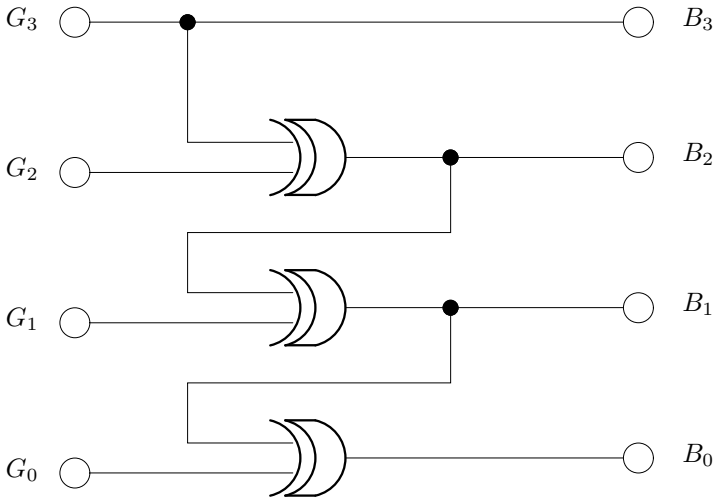
Finalmente para B_0 teremos:

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

O que nos levará a $B_0 = G_3 \oplus G_2 \oplus G_1 \oplus G_0$ (este resultado pode ser encontrado de maneira análoga ao obtido para B_1). Desta forma temos o seguinte conjunto de funções que devem ser implementadas:

$$\begin{aligned} B_3 &= G_3 \\ B_2 &= G_3 \oplus G_2 \\ B_1 &= G_3 \oplus G_2 \oplus G_1 = B_2 \oplus G_1 \\ B_0 &= G_3 \oplus G_2 \oplus G_1 \oplus G_0 = B_1 \oplus G_0 \end{aligned}$$

A implementação mais simples e direta destas expressões é dada pelo circuito abaixo:



Veremos agora em outro exemplo a conversão de binário para complemento 2, com palavras binárias de 4 bits ($B_3B_2B_1B_0$) e seus respectivos complementos ($C_3C_2C_1C_0$). A tabela-verdade assume a seguinte forma:

n	B_3	B_2	B_1	B_0	C_3	C_2	C_1	C_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	1	1	1
2	0	0	1	0	1	1	1	0
3	0	0	1	1	1	1	0	1
4	0	1	0	0	1	1	0	0
5	0	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	0
7	0	1	1	1	1	0	0	1
8	1	0	0	0	1	0	0	0
9	1	0	0	1	0	1	1	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	0	1	0	1
12	1	1	0	0	0	1	0	0
13	1	1	0	1	0	0	1	1
14	1	1	1	0	0	0	1	0
15	1	1	1	1	0	0	0	1

Com base na tabela-verdade podemos construir os mapas de Karnaugh e deduzir as expressões lógicas para cada uma das saídas $C_3 \ldots C_0$:

B_1B_0 \ B_3B_2	00	01	11	10
00		1		1
01	1	1		
11	1	1		
10	1	1		

$$C_3 = \overline{B}_3B_1 + \overline{B}_3B_0 + \overline{B}_3B_2 + B_3\overline{B}_2\overline{B}_1\overline{B}_0$$

B_1B_0 \ B_3B_2	00	01	11	10
00		1	1	
01	1			1
11	1			1
10	1			1

$$C_2 = \overline{B}_2B_1 + \overline{B}_2B_0 + B_2\overline{B}_1\overline{B}_0$$

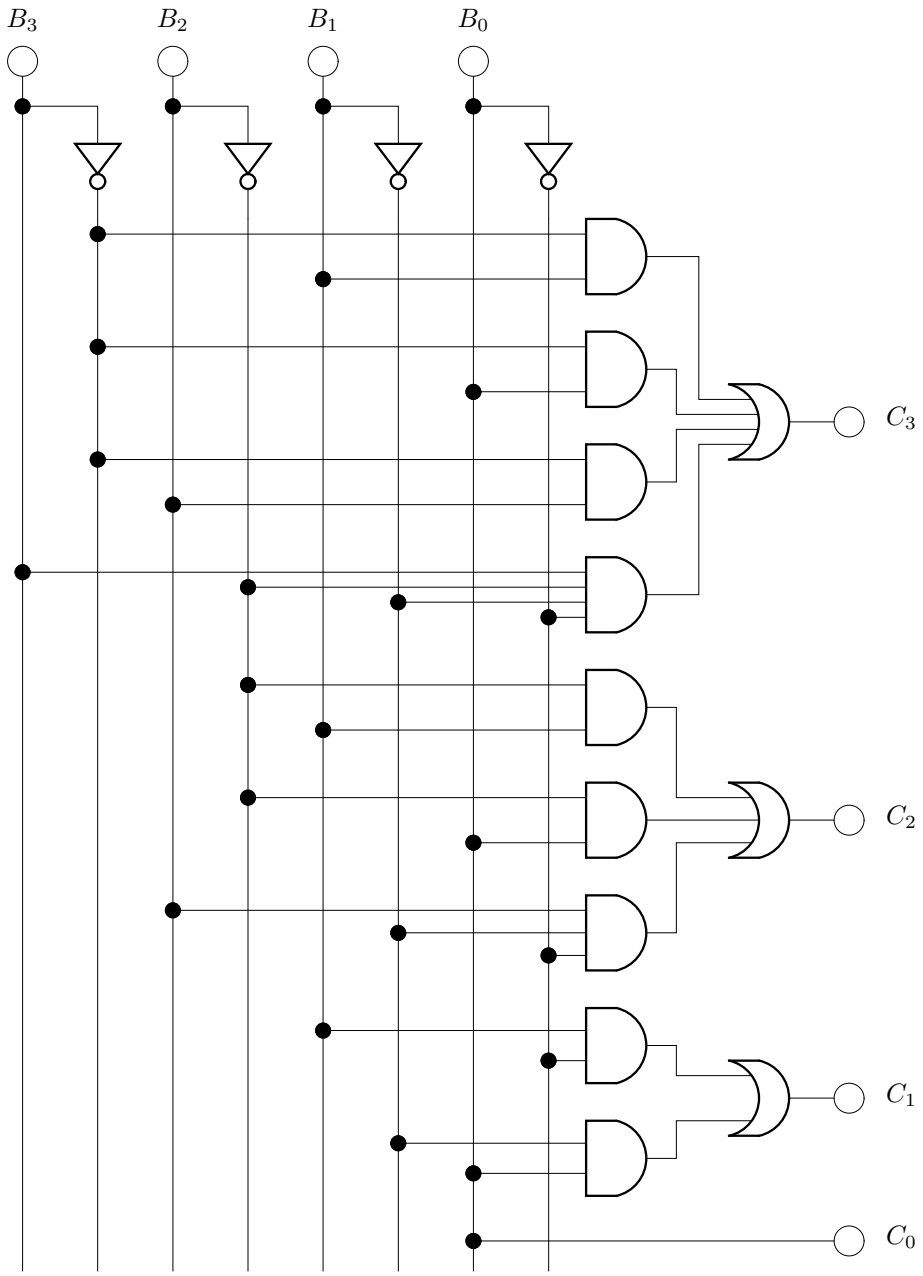
B_1B_0 \ B_3B_2	00	01	11	10
00				
01	1	1	1	1
11				
10	1	1	1	1

$$C_1 = \overline{B}_1B_0 + B_1\overline{B}_0$$

B_1B_0 \ B_3B_2	00	01	11	10
00				
01	1	1	1	1
11	1	1	1	1
10				

$$C_0 = B_0$$

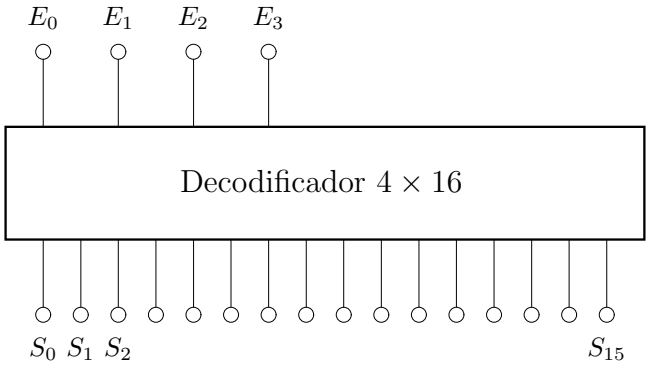
As expressões acima nos permitem implementar o circuito como apresentado na figura a seguir, usando as portas básicas. Obviamente outras implementações são possíveis, manipulando adequadamente as expressões acima.



Além dos circuitos apresentados acima, podemos ter codificadores de binário para complemento 1 e vice-versa, de BCD para 7 segmentos, etc. que podem ser facilmente projetados pelo leitor usando a técnica apresentada acima.

3.4 Codificadores e decodificadores

Um decodificador é um dispositivo com n entradas e 2^n saídas. Para cada valor da entrada uma única saída é verdadeira e as demais são falsas. Normalmente representamos um decodificador através de um bloco, como por exemplo para o decodificador com 4 entradas e 16 saídas:



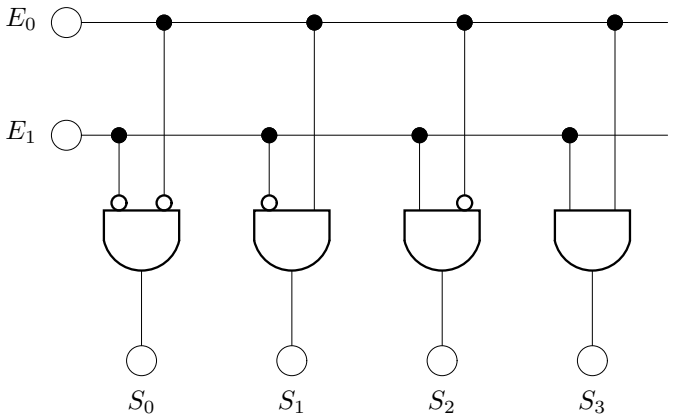
A tabela-verdade do decodificador 4×16 acima assume esta forma:

E_3	E_2	E_1	E_0	S_0	S_1	S_2	S_3	$\cdots$	S_{15}
0	0	0	0	1	0	0	0	$\cdots$	0
0	0	0	1	0	1	0	0	$\cdots$	0
0	0	1	0	0	0	1	0	$\cdots$	0
0	0	1	1	0	0	0	1	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
1	1	1	1	0	0	0	0	$\cdots$	1

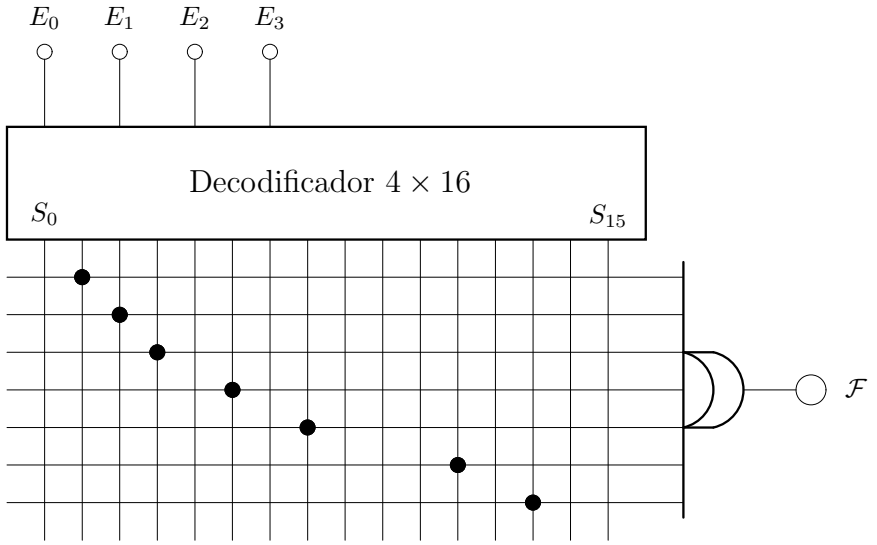
Por exemplo, para o decodificador de 2×4 linhas teremos a seguinte tabela-verdade:

E_1	E_0	S_0	S_1	S_2	S_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

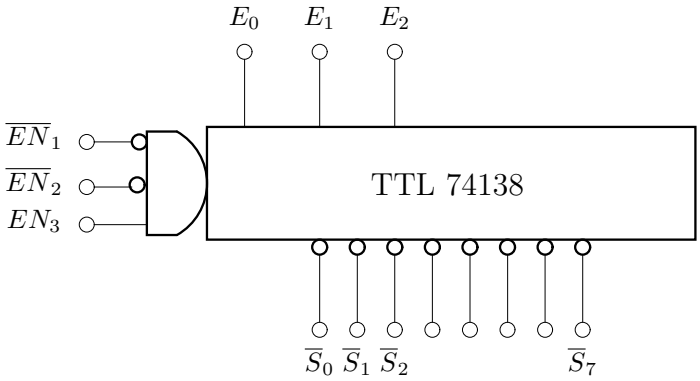
Que pode ser facilmente implementada através do circuito abaixo (note que usamos uma nova notação para as entradas complementadas):



Um decodificador pode ser útil na implementação de funções lógicas simples, como alguns dos circuitos vistos anteriormente. Por exemplo, eis uma implementação do detector de números primos de 4 bits usando um decodificador 4×16:



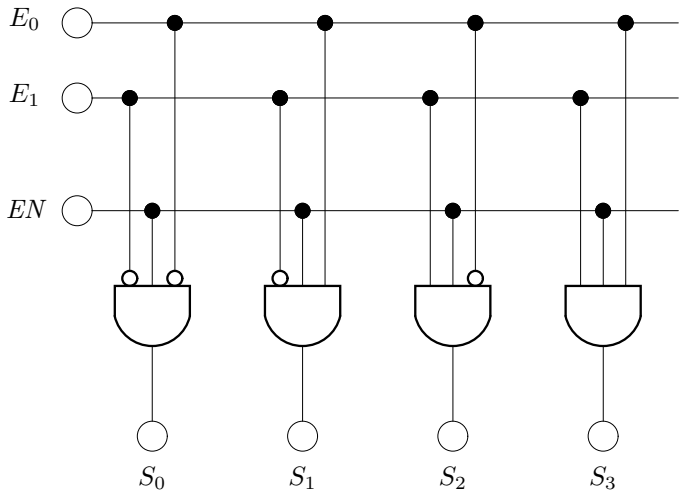
Devido às suas características construtivas, nas principais famílias lógicas uma saída no estado baixo (0 ou L(ow)) drena uma corrente significativamente maior que uma saída em estado alto (1 ou H(igh)). Para uma porta TTL convencional, a corrente em uma saída em estado baixo é da ordem de 1.6 mA, enquanto uma saída em estado alto drena apenas cerca de 200 μA, ou seja, uma corrente 8 vezes menor. O comportamento esperado para um um decodificador com 2^n saídas é ter a saída ativa em estado alto e as demais $2^n - 1$ saídas em estado baixo, sendo por isso chamado de *ativo alto*. Entretanto, para diminuir o consumo de energia e a conseqüente dissipação de calor nos circuitos integrados, a maioria dos decodificadores comerciais emprega uma lógica de *ativo baixo*, na qual a saída ativa se encontra em estado baixo (0) e as demais em estado alto (1). O decodificador comercial integrado TTL 74138, de 3×8 vias, é entao representado por:



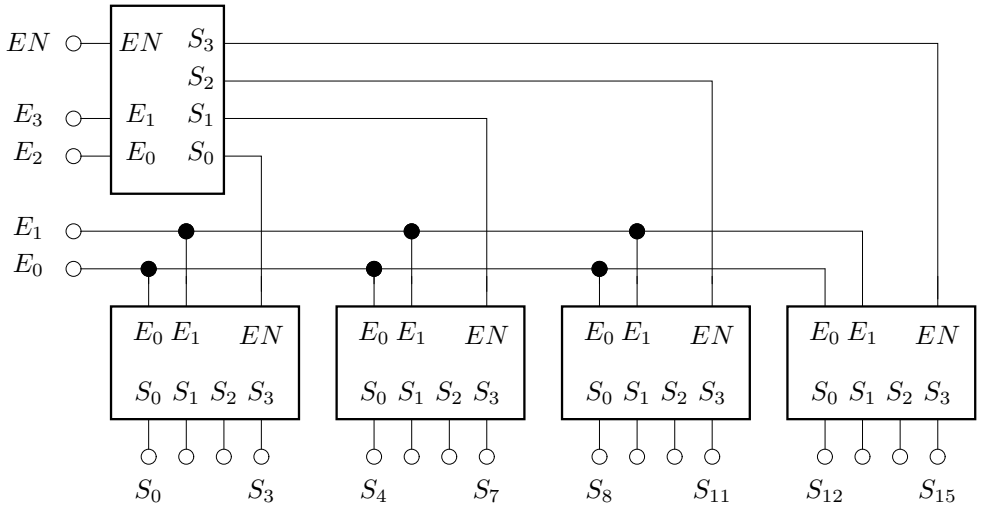
Além das entradas que definem a saída ativa, um decodificador pode ter uma ou mais entradas do tipo *strobe* ou *enable/disable*, como as entradas $\overline{EN}_1$, $\overline{EN}_2$ e EN_3 do decodificador 3×8 acima. A função dessas entradas é inibir a saída ativa do decodificador: na situação acima, enquanto $\overline{EN}_1\overline{EN}_2EN_3$ for falso, nenhuma saída será ativada, não importando o estado das entradas E_i. Considerando um decodificador 2×4 com uma entrada *enable* teremos a seguinte tabela-verdade:

EN	E_1	E_0	S_0	S_1	S_2	S_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Este comportamento pode ser facilmente implementado através do seguinte circuito:



As entradas do tipo *enable* podem ser usadas para conectar decodificadores em cascata, permitindo assim a construção de decodificadores maiores. O exemplo abaixo mostra a implementação de um decodificador 4×16 usando 5 decodificadores 2×4:



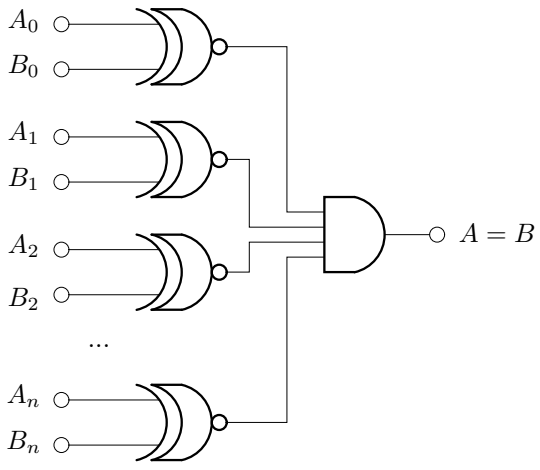
Ao contrário dos decodificadores, os codificadores convertem entre 2^n entradas e uma saída com n bits. O circuito integrado 74148 é um codificador de 3×8 vias, funcionando em ativo baixo (sua entrada ativa está em estado 0). Sua tabela-verdade é dada por:

$\overline{EN}$	$\overline{E}_0$	$\overline{E}_1$	$\overline{E}_2$	$\overline{E}_3$	$\overline{E}_4$	$\overline{E}_5$	$\overline{E}_6$	$\overline{E}_7$	$\overline{S}_0$	$\overline{S}_1$	$\overline{S}_2$	$\overline{EO}$	$\overline{GS}$
1	x	x	x	x	x	x	x	x	1	1	1	0	x
0	x	x	x	x	x	x	x	0	0	0	0	1	0
0	x	x	x	x	x	x	0	1	0	0	1	1	0
0	x	x	x	x	x	0	1	1	0	1	0	1	0
0	x	x	x	x	0	1	1	1	0	1	1	1	0
0	x	x	x	0	1	1	1	1	1	0	0	1	0
0	x	x	0	1	1	1	1	1	1	0	1	1	0
0	x	0	1	1	1	1	1	1	1	1	0	1	0
0	0	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	0	1

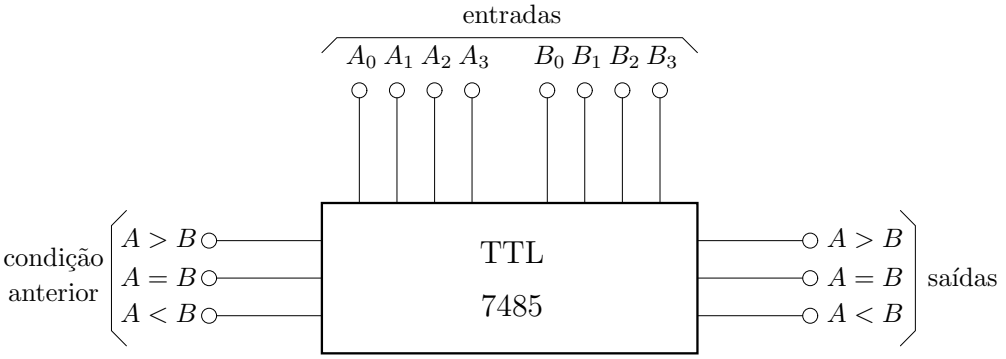
A entrada $\overline{EN}$ habilita a saída. As saídas complementadas $S_0 \ldots S_2$ indicam o dígito binário correspondente à entrada ativa de número mais elevado (trata-se então de um codificador com prioridade, onde as entradas de número mais elevado tem prioridade). A saída $\overline{EO}$ é ativada (0) quando não existir entrada ativa, ou caso as saídas estejam desabilitadas ($\overline{EN} = 1$). A saída $\overline{GS}$ (chamada *group service*) é ativada caso haja alguma entrada ativa, independente da habilitação das saídas.

3.5 Comparadores de palavras

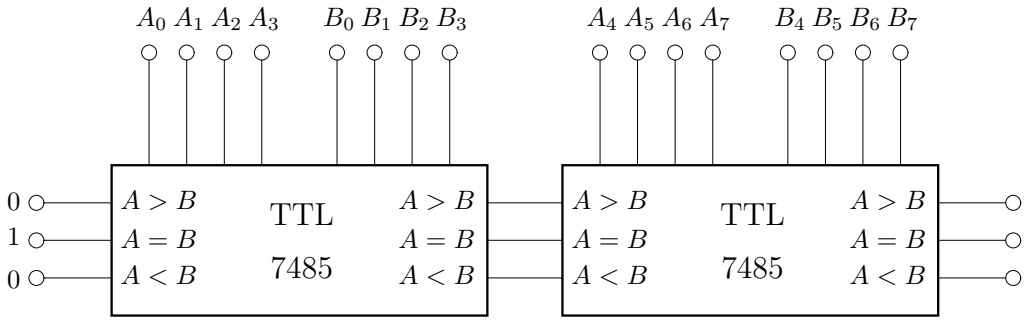
Um circuito comparador simples permite comparar duas palavras de n bits, indicando quando estas são iguais. Esse tipo de comparador pode ser facilmente implementado através de portas *não-ou-exclusivo* entre os respectivos bits das palavras a comparar ($\overline{A_i \oplus B_i} = A_i B_i + \overline{A}_i\ \overline{B}_i$):



Circuitos comparadores mais complexos podem ser construidos para indicar se $A = B$, $A > B$ ou $A < B$. O circuito integrado TTL 7485 é um comparador de palavras de 4 bits com essa possibilidade:

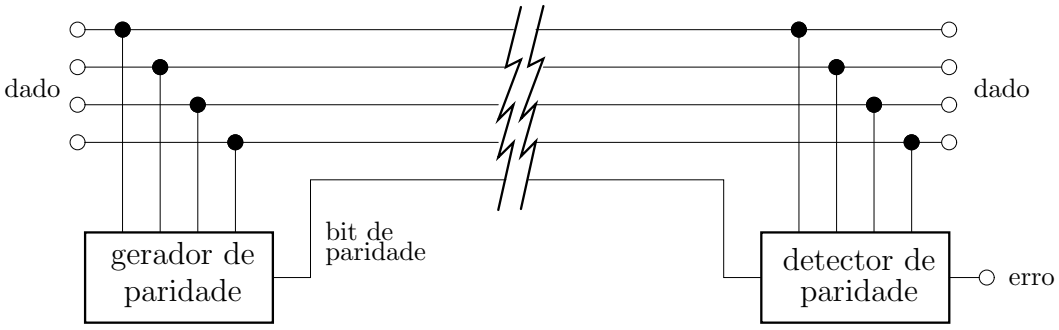


As entradas indicadas como *condição anterior* permitem a conexão de comparadores em cascata para efetuar a comparação de palavras maiores. No circuito do exemplo a seguir podem ser comparadas palavras de 8 bits:

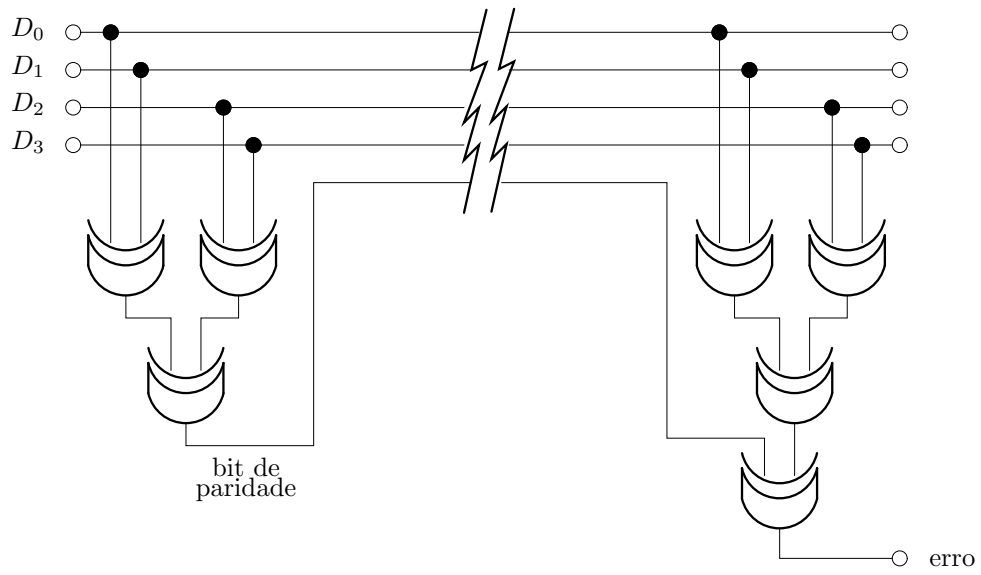


3.6 Geradores e detectores de paridade

Os geradores e detectores de paridade são muito úteis em comunicação de dados, onde permitem detectar a presença de erros de transmissão. A técnica básica consiste em associar um bit de paridade ao dado a ser transmitido, indicando se este tem um número par ou ímpar de bits ativos. Ao ser recebido, o dado é testado em relação ao bit de paridade, e eventuais erros podem ser detectados.



Um gerador simples de bit de paridade para palavras de 4 bits e seu respectivo detector são apresentados no diagrama a seguir.

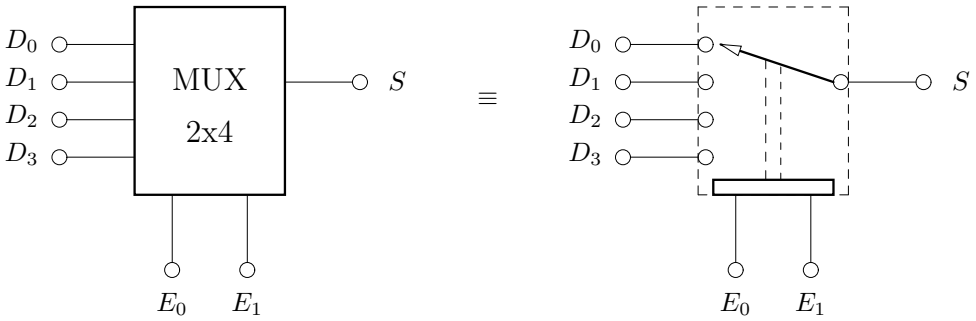


Um circuito é dito de *paridade par*, quando o número total de bits "1", incluindo o de paridade, é par. Caso contrário, o circuito é dito de *paridade ímpar*. O circuito acima apresentado é de paridade par, pois o bit de paridade vale 0 se o dado possuir um número par de bits ativos, e 1 senão; em conseqüência, o número total de bits ativos é sempre par.

No caso da detecção de erro em um dado, o sistema de recepção de dados pode solicitar que este seja retransmitido, ou tomar outra atitude. Os circuitos de paridade são capazes de detectar erros quando um número ímpar de bits é alterado durante a transmissão. Existem circuitos mais complexos de comunicação de dados capazes de detectar erros de modo mais abrangente e mesmo corrigir erros pequenos, sem necessidade de retransmissão. No entanto esses circuitos necessitam de mais bits de controle associados a cada dado.

3.7 Multiplexadores e demultiplexadores

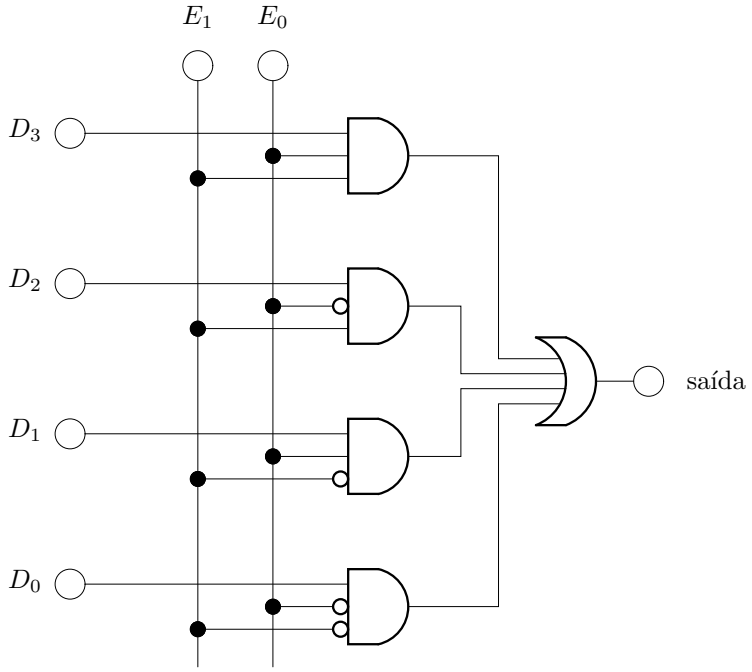
Um multiplexador é um dispositivo com 2^n entradas de dados, uma saída e n entradas de seleção que definem qual entrada de dados transferir para a saída. Um típico multiplexador de 4 vias é mostrado abaixo:



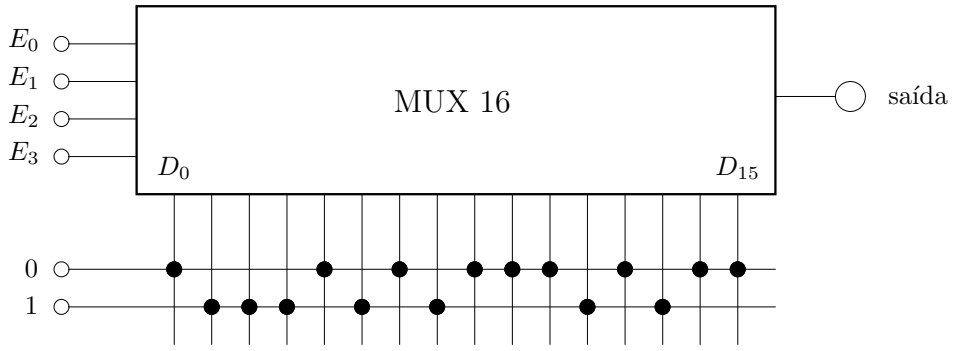
A tabela-verdade do multiplexador acima é dada por:

E_1	E_0	S
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

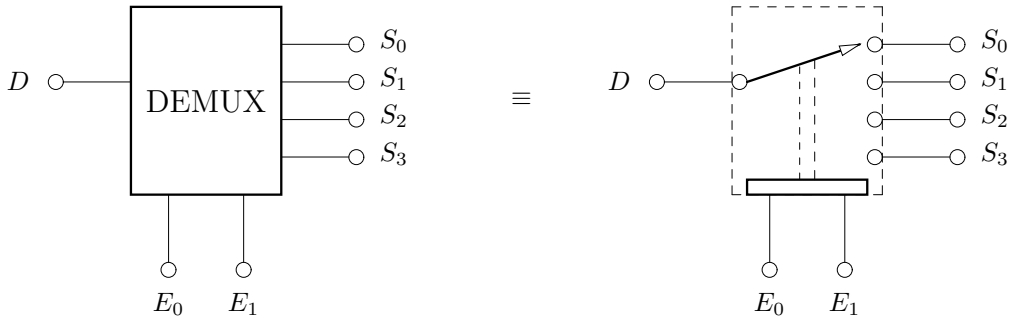
O que nos leva a $S = \overline{E}_1\overline{E}_0D_0 + \overline{E}_1E_0D_1 + E_1\overline{E}_0D_2 + E_1E_0D_3$. Esta função pode ser facilmente implementada pelo circuito abaixo:



Assim como os decodificadores, os multiplexadores podem ser usados na implementação de funções lógicas, como por exemplo a detecção de números primos:



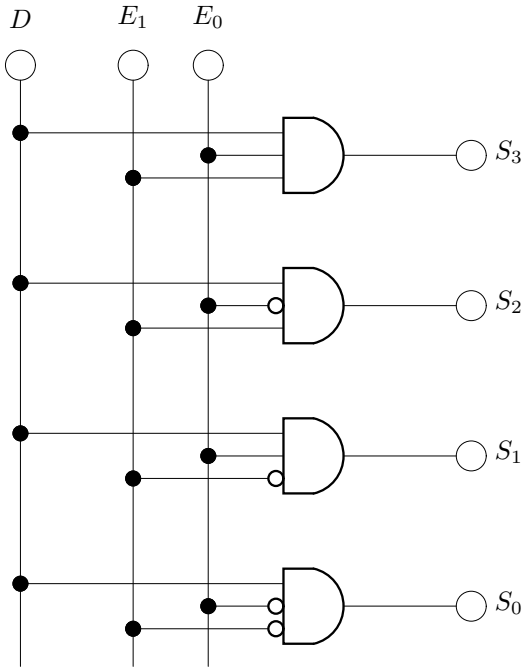
Os demultiplexadores permitem distribuir um dado em sua entrada para uma entre 2^n saídas, escolhida de acordo com as entradas de seleção:



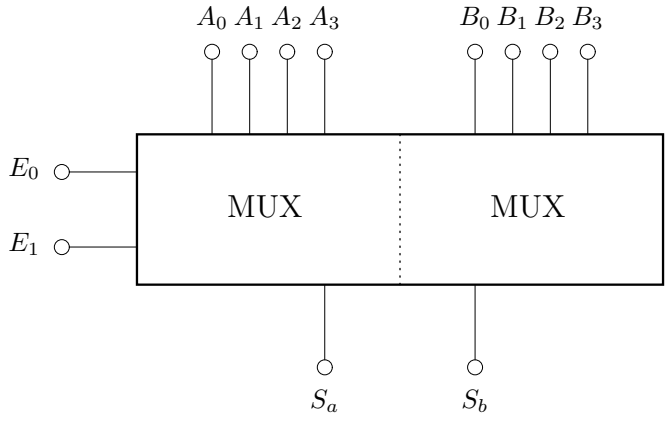
Para um demultiplexador de 4 vias teríamos a seguinte tabela-verdade:

E_1	E_0	S_0	S_1	S_2	S_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

Esta tabela pode ser implementada pelo seguinte circuito:



Os dispositivos mux/demux comerciais geralmente possuem uma ou mais entradas do tipo *enable*, complementadas ou não, que permitem conectar circuitos em cascata. As principais aplicações desses dispositivos são a seleção e encaminhamento de dados, a conversão série-paralelo de dados, a geração de formas de onda e a síntese de funções lógicas. Um exemplo de circuito integrado comercial é o TTL 74153, que possui dois multiplexadores de 4 vias controlados simultaneamente (as entradas de seleção são comuns a ambos):



3.8 Somadores

Os circuitos somadores permitem efetuar operações de soma entre duas palavras de n bits, levando em conta o "vai-um" (excesso ou *carry*). Para a soma de duas palavras A e B de 1 bit teremos a seguinte tabela-verdade, para a soma S e o excesso C:

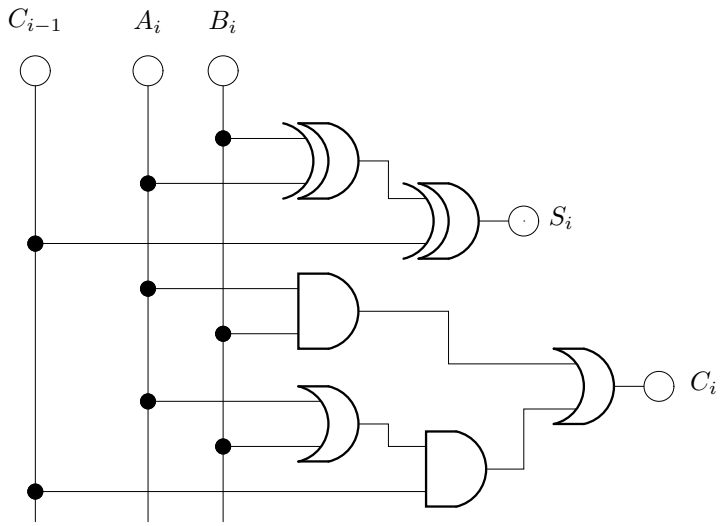
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Pode-se então concluir que $S = A \oplus B$ e $C = A \cdot B$. O circuito definido por essas funções é chamado "meio-somador" e sua implementação é trivial, usando apenas duas portas lógicas. Um circuito "somador completo" deve levar em conta o excesso ("vai-um") de um eventual antecessor, para poder ser associado em cascata a outros somadores e assim implementar somadores para palavras mais longas que 1 bit. A tabela-verdade para um somador completo de 1 bit é dada por:

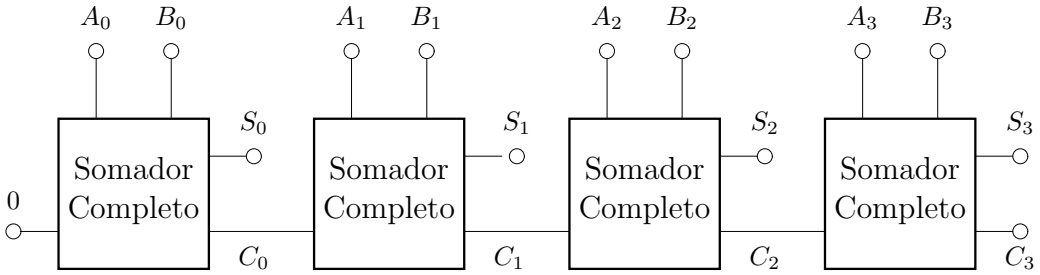
C_{i-1}	A_i	B_i	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Através dos mapas de Karnaugh obtemos as funções lógicas e sua implementação:

$$\begin{aligned} S_i &= A_i \oplus B_i \oplus C_{i-1} \\ C_i &= A_i B_i + C_{i-1}(A_i + B_i) \end{aligned}$$

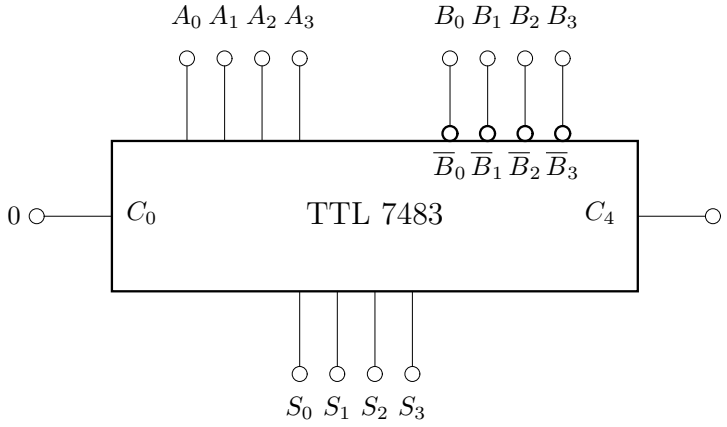


A partir desse módulo somador completo de 1 bit podemos construir somadores mais complexos, para palavras com n bits. Por exemplo, eis a implementação de um somador para palavras de 4 bits:



O circuito acima é encontrado comercialmente no integrado TTL 7483. Dois somadores deste tipo podem ser associados para constituir um somador de 8 bits.

A construção de subtratores pode ser feita associando somadores a conversores de complemento 1 ou 2. Por exemplo, o circuito TTL 7483 pode ser empregado para implementar um subtrator de palavras positivas de 4 bits, usando o complemento de 1. Para isso é necessário complementar o número B ($A - B = A + (-B)$) e atribuir 1 ao excesso inicial ($C_0 = 1$). O bit S_3 indicará o sinal da saída, e os números negativos serão apresentados em complemento 1. O circuito abaixo indica a forma de implementação:

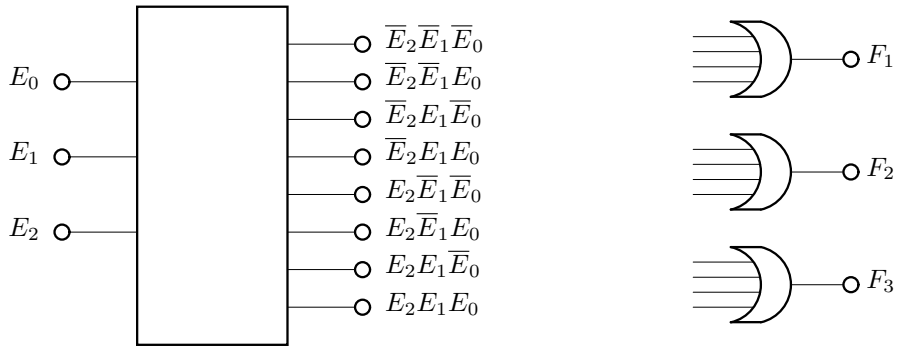


3.9 Matrizes de funções lógicas

As matrizes de funções lógicas permitem sintetizar conjuntos de funções lógicas que dependem das mesmas variáveis de entrada. A síntese das funções pode ser decomposta em duas etapas:

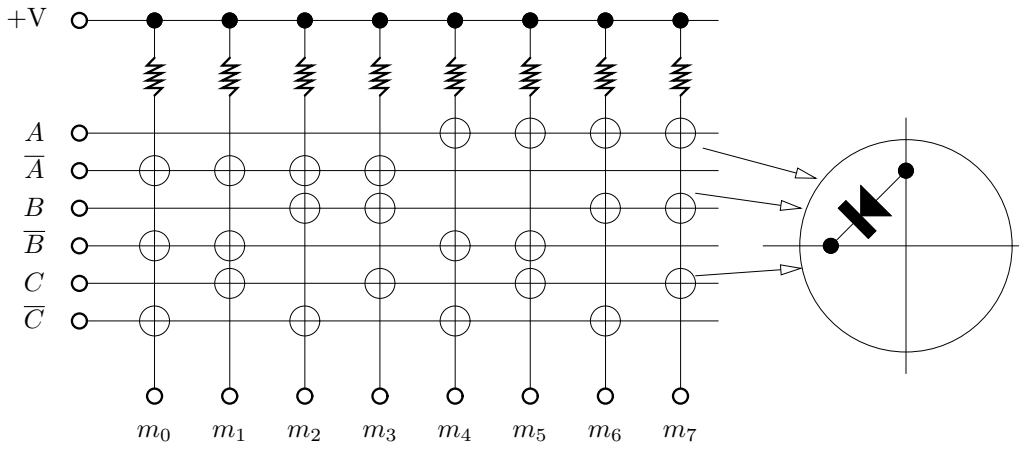
- geração dos produtos de variáveis booleanas (minitermos);
- geração das somas de produtos que vão formar as funções desejadas (somas dos minitermos).

As etapas acima são chamadas respectivamente "blocos E" e "blocos OU", devido à natureza das operações que são executadas.

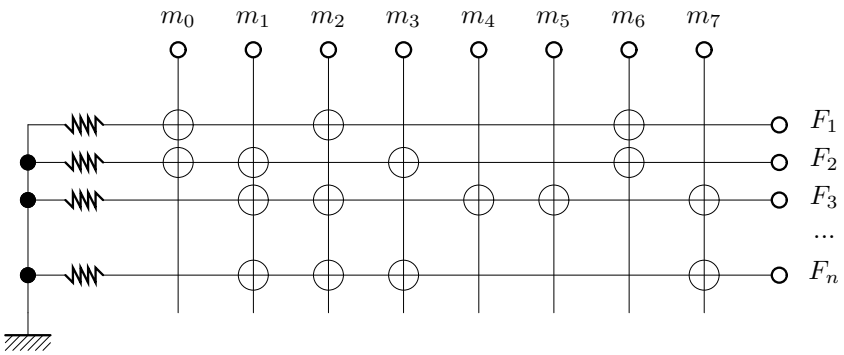


bloco E bloco OU

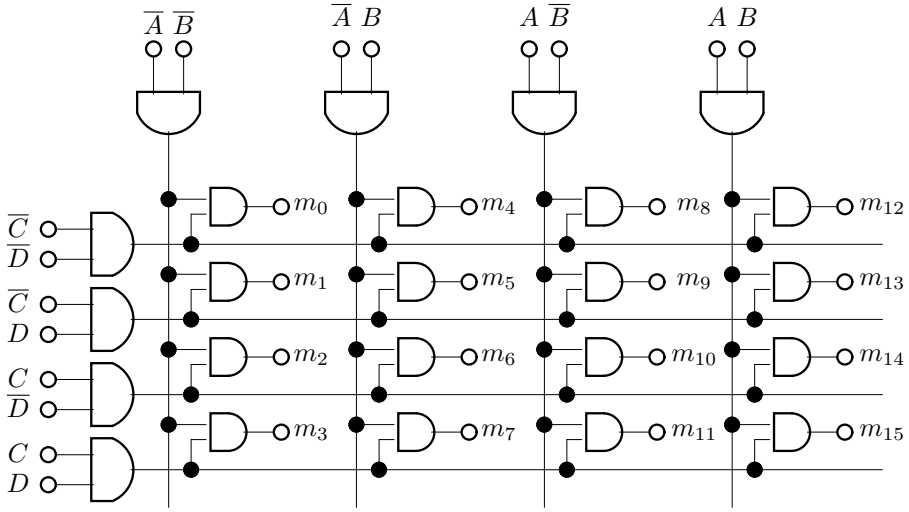
A geração dos minitermos (bloco E) pode ser efetuada através de uma estrutura eletrônica bastante simples e facilmente integrável, composta por uma matriz de diodos e resistores, conforme mostra o diagrama abaixo:



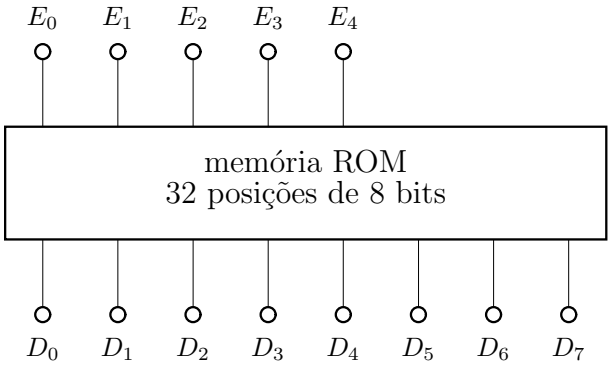
Da mesma forma, a geração das funções (bloco OU) também pode ser implementada através de uma matriz de diodos:



Outra estrutura bastante empregada para a geração dos minitermos, por sua facilidade de integração, é a chamada *matriz de duplo encadeamento*, mostrada a seguir:



Um circuito combinacional bastante conhecido e normalmente implementado através de matrizes lógicas é o das memórias ROM (*Read-Only Memories*). Neste tipo de memória os dados armazenados são acessados através de um "endereço" fornecido na entrada do circuito. Por exemplo, uma memória ROM de 8 bits com 32 posições pode ser representada sob a forma do bloco abaixo:

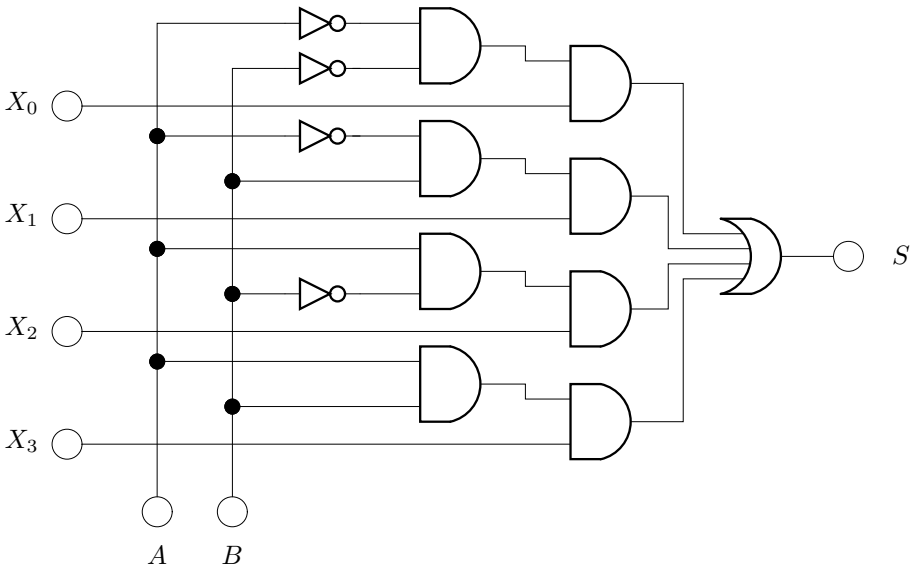


Ao selecionar um determinado endereço na entrada (por exemplo, 01001_2), o bloco de memória fornece em suas saídas o valor binário armazenado naquela posição (por exemplo 11001010_2). As memórias ROM podem ser facilmente implementadas através dos circuitos

vistos até agora, mas outras tecnologias podem ser empregadas para obter memórias ROM, como veremos no capítulo 6.

3.10 Exercícios

1. Obtenha a expressão lógica do bloco abaixo, sua respectiva tabela-verdade e explique que função este realiza:



2. Projete um circuito conversor de código Gray para o código decimal BCD, com 4 bits.

3. Implemente a função lógica $\mathcal{F}(A, B, C) = \sum m(0, 5, 7) + X(1, 2)$, usando a) multiplexador, b) decodificador e c) portas lógicas básicas.

4. Projete um somador binário completo para palavras de 3 bits.

5. Projete um circuito decodificador de binário (4 bits) para 7 segmentos, considerando todos os dígitos hexadecimais, conforme indicado na seção 1.5.3.

6. Um sensor de rotação de um eixo gera um sinal de 4 bits que indica a posição do eixo em passos de 30º, utilizando código Gray. Supõe-se que as combinações acima de 360º, não utilizadas, nunca ocorrem. Projete um circuito que indique quando o eixo estiver no 2º quadrante, utilizando:

   - mapa de Karnaugh, em minitermos;
   - mapa de Karnaugh, em maxitermos;
   - multiplexador;
   - decodificador.

7. Projete um circuito gerador de paridade ímpar e um circuito detector de paridade, responsável pela verificação dos dados recebidos, considerando palavras de 3 bits.

8. Um circuito incrementador/decrementador tem como entrada um valor X e uma entrada de controle C. Se $C = 1$ a saída S do circuito apresenta o valor de X incrementado ($S = X + 1$) e se $C = 0$ a saída S do circuito apresenta o valor de X decrementado ($S = X - 1$). A entrada X pode ter valores entre 1 e 6. Obtenha a tabela-verdade, a expressão mínima e uma implementação usando decodificador.

9. O sistema de controle de uma válvula deve abrí-la ou fechá-la em função de duas variáveis de entrada: a *Classe*, que pode assumir os valores 0, 1, 2 ou 3, e o *Grupo*, que pode assumir os valores A, B ou C. A válvula deve abrir somente quando $Grupo = A$ e $Classe \geq 2$; caso contrário ela deve permanecer fechada. Projete o circuito de controle, apresentando sua tabela-verdade, mapa de Karnaugh e função mínima (sugestão: codifique as entradas).

10. Implementar um decodificador de binário para código Gray, com 4 bits, utilizando: a) somente portas NAND; b) somente portas NOR; c) decodificador 4x16.

11. Implementar e testar no simulador um somador completo (um único bit) utilizando: a) CI 74LS138; b) CI 74LS153.

12. Deseja-se implementar um sistema de controle simplificado para controlar o sinal num cruzamento de trânsito. As ruas serão chamadas de A, B, C e D. As saídas dos sinais serão *verdes* (nível 1) ou *vermelhas* (nível 0) e receberão os nomes de S_a, S_b, S_c e S_d, correspondentes às respectivas ruas. Deseja-se ter um controle que dê preferência aos carros das seguintes ruas:

    - Durante os dias úteis: $A > B > C > D$, isto é, os carros da rua A têm preferência de passagem sobre os demais.
    - Durante os finais de semana: $D > C > B > A$.

    Sabe-se que não circulam carros nas seguintes ruas e dias:

    - C, durante os dias úteis.
    - B, durante os finais de semana.

    Implemente o sistema de controle utilizando: a) DEMUX 16 canais; b) Portas NAND.

13. Implementar e testar no simulador um circuito que gere os complementos $C2$ e $C1$ do número colocado em sua entrada, de acordo com uma entrada de seleção: $S = 1 \rightarrow C2$, $S = 0 \rightarrow C1$.

14. Obter um MUX 16x1 a partir da associação de: a) MUX 4x1; b) MUX 2x1.

15. Obter um DEMUX de 9 canais, utilizando: a) DEMUX 2 canais; b) DEMUX 4 canais.

16. Implementar um circuito que detecte os números primos de 0 a 15, utilizando: a) MUX 16x1; b) Um único MUX 8x1.

17. Implementar e testar no simulador um circuito somador/subtrator de dois números de quatro bits cada, a partir de um somador e portas lógicas, disponíveis comercialmente (sugestão: portas XOR e somador 74283, 7483, 7486, etc).

18. Implementar e testar no simulador um circuito que realize a divisão de dois números de dois bits cada, e que apresente as seguintes informações de saida:

- Resultado de dois bits, representando a parte inteira da operação;
- Um bit indicando overflow;
- Um bit indicando que o resultado é composto por parte fracionária diferente de zero.

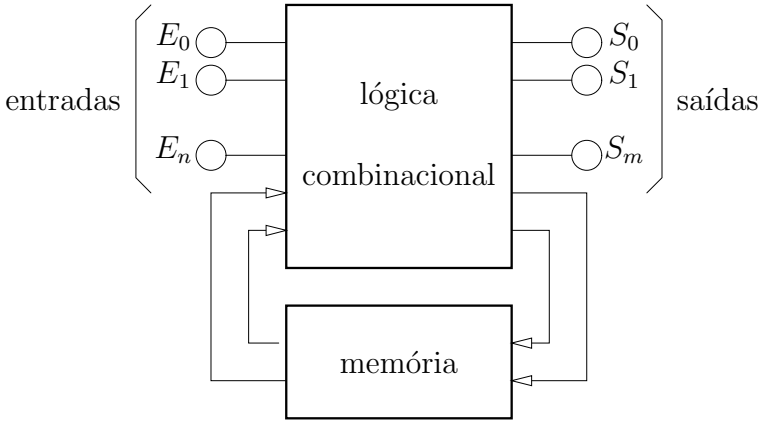
Capítulo 4

Lógica Seqüencial

4.1 Introdução

Nos circuitos combinacionais, as saídas em um instante t dependiam única e exclusivamente das entradas do circuito naquele mesmo instante, ou seja, do estado atual de suas entradas. Nos circuitos que estudaremos neste capítulo, as saídas não dependem mais somente do estado atual das entradas, mas também de estados anteriores do circuito.

Podemos representar um circuito seqüencial como sendo composto de um circuito combinacional associado a uma unidade de memória, capaz de armazenar estados anteriores do sistema:

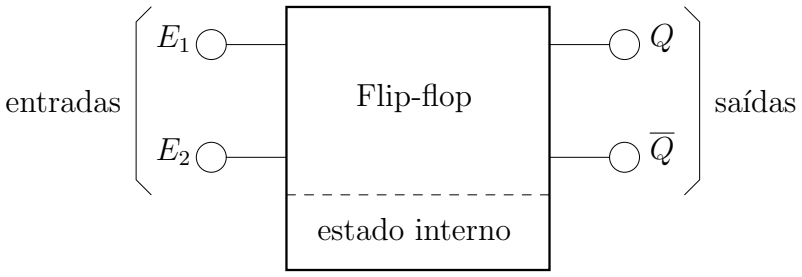


Neste texto estudaremos somente os circuitos seqüenciais *síncronos*. Neste tipo de circuito, todas as mudanças de estado internas são cadenciadas por um sinal externo de sincronização chamado *clock*, que geralmente é uma onda quadrada de freqüência constante. O sinal de *clock* deve ser comum e único para todas as partes do circuito. Desta forma, consideramos que as mudanças de estado em um circuito seqüencial são provocadas pelos pulsos do sinal de *clock*, em instantes de tempo discretos: t, $t+1$, $t+2$, $t+3$, ...

A saída de um circuito seqüencial é função de um número limitado de estados anteriores, que devem portanto ser armazenados em algum lugar do circuito seqüencial. As unidades básicas de memória empregadas em circuitos seqüenciais são os *flip-flops*, que estudaremos a seguir.

4.2 Flip-flops

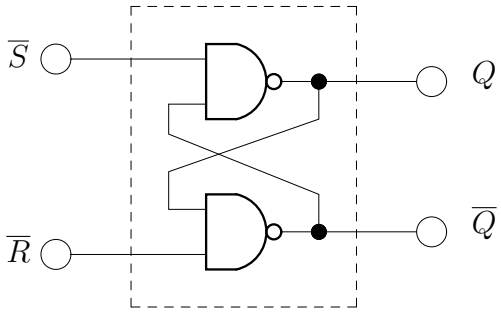
Um *flip-flop* (gangorra, em inglês) é um dispositivo que possui dois estados internos estáveis e complementares, sendo por isso também chamado de *biestável*. O estado interno de um flip-flop vale "0" ou "1", e pode permanecer indefinidamente em um deles, em função de suas entradas. Desta forma, um flip-flop pode armazenar 1 bit de informação digital, e por isso constitui a célula básica de memória. Um flip-flop normalmente apresenta um conjunto de entradas capaz de alterar seu estado interno, e duas saídas apresentando respectivamente seu estado interno e o complemento deste.



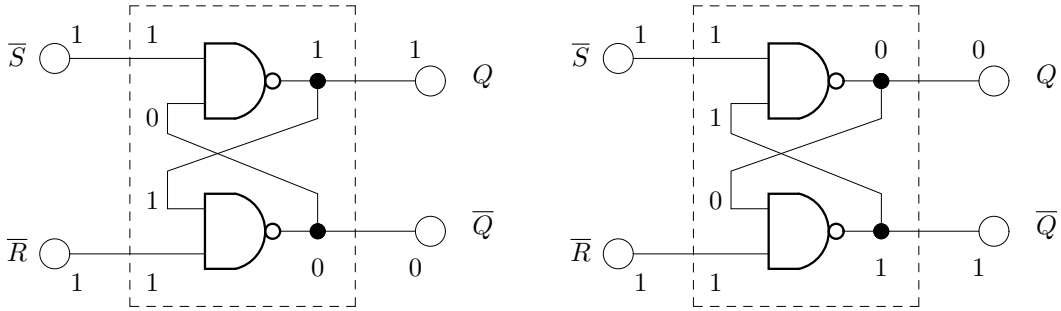
Existem diversos tipos de flip-flops com comportamentos distintos, mas que podem ser transformados entre si. A seguir veremos os tipos mais conhecidos:

4.2.1 Flip-flop RS (*Reset-Set)*

O flip-flop de tipo RS (*Reset-Set*) é o mais simples desses dispositivos, e pode ser facilmente construído a partir de duas portas *não-E*, como mostra a figura abaixo:



Mantendo as entradas $\overline{R}$ e $\overline{S}$ inativas (1), temos dois estados possíveis de equilíbrio para essa estrutura, que podemos observar na figura abaixo:



A determinação do estado das saídas de um flip-flop RS em um instante futuro (Q_{t+1} e $\overline{Q}_{t+1}$) deve levar em conta suas entradas atuais $\overline{R}_t$ e $\overline{S}_t$ e o estado interno atual Q_t do mesmo:

	$\overline{S}_t$	$\overline{R}_t$	Q_t	Q_{t+1}	$\overline{Q}_{t+1}$
1	0	0	0	1	1
2	0	0	1	1	1
3	0	1	0	1	0
4	0	1	1	1	0
5	1	0	0	0	1
6	1	0	1	0	1
7	1	1	0	0	1
8	1	1	1	1	0

Vamos analisar as situações possíveis para as entradas do flip-flop:

- Nas linhas 1 e 2, temos $\overline{S}_t = \overline{R}_t = 0$, e por conseqüência $Q_{t+1} = \overline{Q}_{t+1} = 1$, o que corresponde a uma situação "proibida" no funcionamento normal do flip-flop (as duas saídas sempre devem ser complementares).
- As linhas 3 e 4, nas quais $\overline{S}_t = 0$ e $\overline{R}_t = 1$ correspondem a uma situação de *Set*, ou seja, impomos o valor "1" ao estado seguinte Q_{t+1} do flip-flop, pouco importando seu estado atual Q_t.
- As linhas 5 e 6, nas quais $\overline{S}_t = 1$ e $\overline{R}_t = 0$ correspondem a uma situação de *Reset*, ou seja, impomos o valor "0" ao estado seguinte Q_{t+1} do flip-flop, pouco importando seu estado atual Q_t.
- Na situação das linhas 7 e 8 temos $\overline{S}_t = \overline{R}_t = 1$, e o flip-flop armazena seu estado anterior: $Q_{t+1} = Q_t$.

Após estas constatações, a tabela anterior pode ser resumida assim:

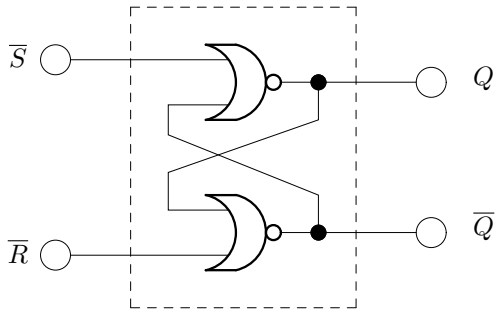
$\overline{S}$	$\overline{R}$	Q_{t+1}
0	0	proibido
0	1	1 (*set*)
1	0	0 (*reset*)
1	1	Q_t (estado anterior)

A partir da tabela acima podemos construir o mapa de Karnaugh e obter a função lógica que representa o comportamento do flip-flop RS. Como os estados onde $R_t = S_t = 0$ são proibidos, vamos considerar a função nestes pontos como sendo irrelevante (X). Obtemos o seguinte mapa, e a função lógica associada:

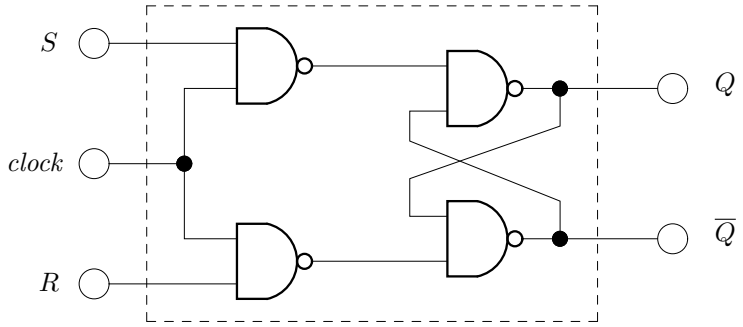
Q_t \ R_tS_t	00	01	11	10
0	X	1		
1	X	1	1	

$$\longrightarrow \quad Q_{t+1} = \overline{R}_t + S_t Q_t$$

O flip-flop RS também pode ser construído a partir de portas *não-OU*, apresentando um comportamento similar:



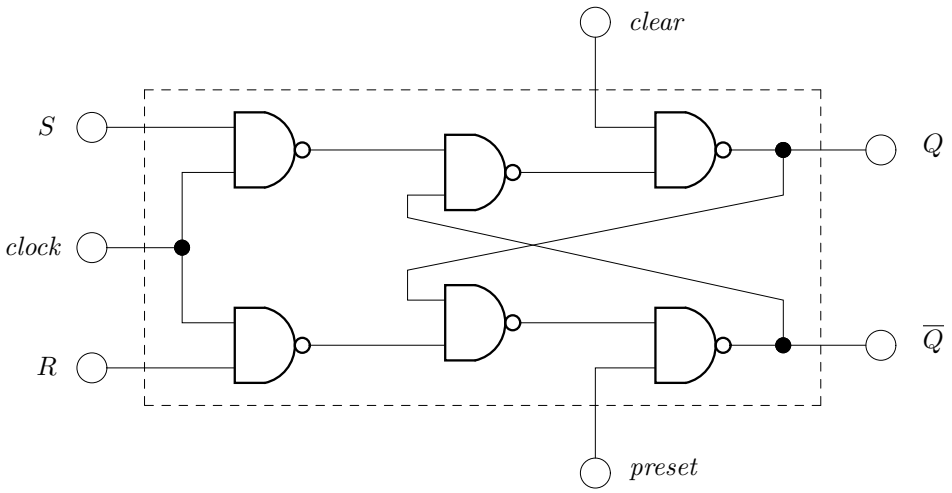
Podemos modificar o flip-flop RS para incluir uma entrada de controle, que pode servir por exemplo para sincronizá-lo em relação a um sinal de relógio externo (*clock*):



Neste novo circuito as entradas R e S são ditas *síncronas* em relação ao sinal de *clock* Ck. Temos então a seguinte tabela-verdade:

Ck	S	R	Q_{t+1}
0	X	X	Q_t
1	0	0	Q_t
1	0	1	0 (*reset*)
1	1	0	1 (*set*)
1	1	1	proibido

Podemos ainda acrescentar à estrutura do flip-flop RS duas entradas do tipo *clear* e *preset*, para alterar imediatamente o estado do flip-flop sem necessidade das entradas R e S (ou seja, de forma assíncrona em relação ao sinal de *clock*):



Pela tabela verdade podemos observar que as entradas *preset* e *clear* não podem operar simultaneamente:

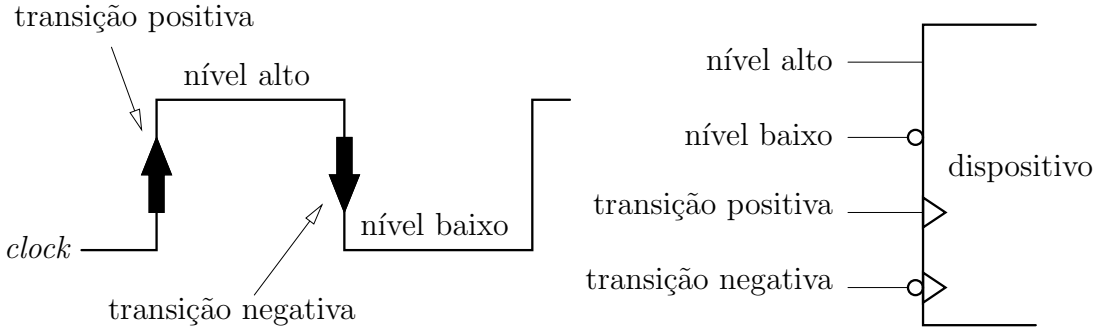
$\overline{PS}$	$\overline{CLR}$	Q_{t+1}
0	0	proibido
0	1	1
1	0	0
1	1	operação normal

4.2.2 Níveis e transições

Os circuitos vistos até o momento têm entradas *sensíveis a nível*, cujo funcionamento é baseado em níveis lógicos constantes e bem definidos (0 ou 1). Este comportamento, desejável em circuitos combinacionais, pode provocar problemas em circuitos seqüenciais síncronos. Por exemplo, o circuito do flip-flop RS com entrada de sincronização (*clock*) vai estar ativo durante toda a duração de cada pulso de *clock*. Com isto, ele pode mudar de estado diversas vezes em cada pulso de clock, caso as entradas mudem durante esse intervalo. Para contornar este problema foram criadas as entradas *sensíveis a transição*, que são consideradas ativas somente durante as transições de nível lógico do sinal aplicado. Com isso uma entrada pode ser sensível a quatro diferentes tipos de excitação:

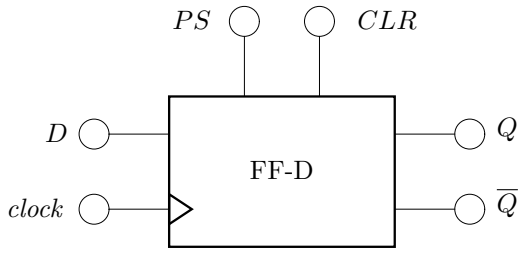
- **Nível lógico alto:** o sinal aplicado vale 1.
- **Nível lógico baixo:** o sinal aplicado vale 0.
- **Transição positiva:** o sinal aplicado passa de 0 a 1.
- **Transição negativa:** o sinal aplicado passa de 1 a 0.

O sinal de *clock* é normalmente aplicado a entradas sensíveis a transição, positiva ou negativa. Transições positivas e negativas são representadas em tabelas-verdade respectivamente pelos símbolos ↑ e ↓. A figura abaixo indica os pontos de sensibilidade de uma entrada em relação ao sinal de clock, e indica os símbolos usados para representar cada tipo de entrada:



4.2.3 Flip-flop D (*Data)*

Neste tipo de flip-flop a saída Q assume o valor de uma entrada de dados D sempre que for habilitado pelo sinal de *clock*. Este tipo de dispositivo é bastante empregado na construção de registradores de deslocamento e no armazenamento de dados (*buffer*). Sua representação e sua tabela-verdade são as seguintes:

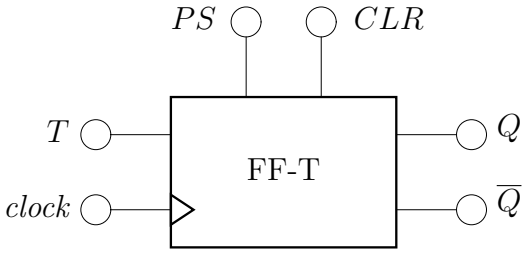


Ck	D_t	Q_{t+1}
0	X	Q_t
$\uparrow$	D_t	D_t
1	X	Q_t

A partir da tabela podemos escrever que na transição $Q_{t+1} = D_t$.

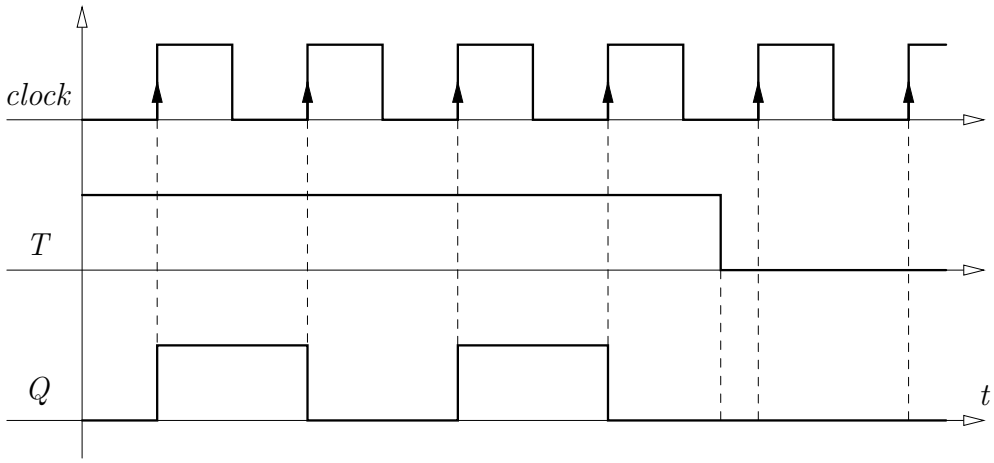
4.2.4 Flip-flop T (*Toggle)*

Neste tipo de flip-flop a saída Q é invertida (*toggled*) quando a entrada T está ativa e o sinal de *clock* a habilitar. Este tipo de dispositivo é bastante empregado em contadores e divisores de freqüência, pois seu comportamento permite a divisão por dois da freqüência do sinal de entrada. Sua representação e sua tabela-verdade são as seguintes:



Ck	T_t	Q_{t+1}
0	X	Q_t
$\uparrow$	0	Q_t
$\uparrow$	1	$\overline{Q}_t$

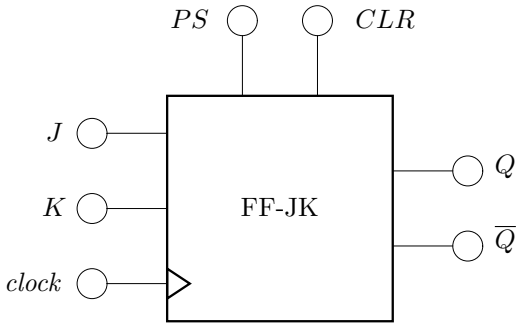
A partir da tabela podemos escrever que na transição $Q_{t+1} = \overline{T}_t Q_t + T_t \overline{Q}_t = T_t \oplus Q_t$. Para compreender melhor o funcionamento deste tipo de flip-flop, vamos observar seu comportamento temporal:



Durante o período em que $T = 1$, a cada transição positiva do *clock* o nível da saída Q é invertido. Com isso, o sinal da saída tem a metade da freqüência do sinal de *clock*.

4.2.5 Flip-flop JK

O flip-flop de tipo JK apresenta um comportamento misto entre os flip-flops RS e T. Ele possui duas entradas J e K (que equivalem respectivamente às entradas S e R dos flip-flops RS) e uma entrada de *clock*. Para os estados normais, o comportamento é o mesmo do flip-flop RS, mas quando $J = K = 1$ (definido como estado proibido no flip-flop RS), a saída é complementada, como ocorre no flip-flop T. Com isso, podemos definir a seguinte representação e tabela-verdade para o flip-flop JK:



Ck	J_t	K_t	Q_{t+1}
0	X	X	Q_t
↑	0	0	Q_t
↑	0	1	0
↑	1	0	1
↑	1	1	$\overline{Q}_t$

O flip-flop JK é bastante popular, pois a partir dele podemos obter os demais flip-flops sem necessidade de circuitos adicionais, como veremos na seção 4.2.7.

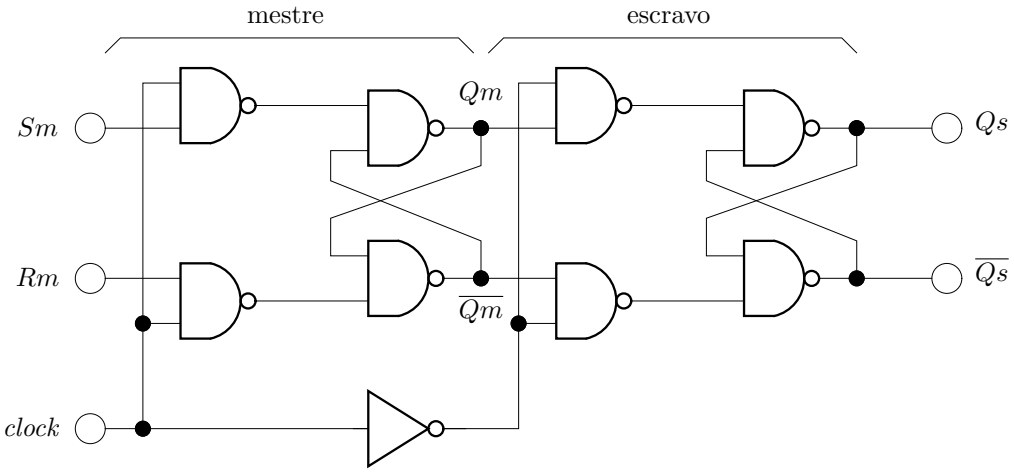
A equação de estado do flip-flop JK pode ser obtida a partir de seu mapa de Karnaugh:

Q_t \ J_tK_t	00	01	11	10
0			1	1
1	1			1

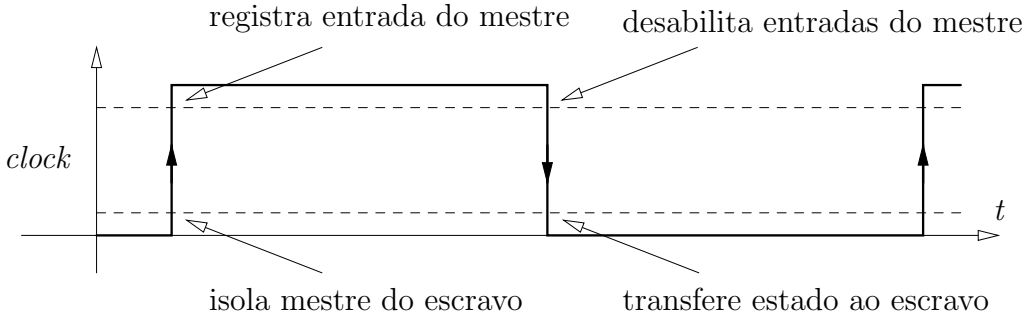
$$\longrightarrow \quad Q_{t+1} = J_t\overline{Q}_t + \overline{K}_tQ_t$$

4.2.6 Flip-flop mestre-escravo

Antes da disponibilidade de flip-flops com entradas sensíveis a transição, uma estrutura especial denominada flip-flop *mestre-escravo* era usada para isolar as entradas instáveis e sincronizar o sistema em relação ao *clock*. O circuito normalmente usado para implementar flip-flops mestre-escravo é composto por dois flip-flops RS em cascata, e pode ser visto na figura a seguir:



Vamos analisar o que ocorre em um flip-flop mestre-escravo em um pulso completo de *clock*. No início, com a descida do sinal de *clock* no escravo, este é desconectado do mestre. Ao final da subida, as entradas do mestre são habilitadas e este começa a atualizar seu estado interno em função as entradas. O mestre permanece assim até o início da descida do pulso de *clock*, quando suas entradas são desabilitadas, e ao final da descida seu estado é transferido ao escravo. Com isso, a saída do flip-flop escravo é completamente isolada de eventuais oscilações nas entradas $\overline{Sm}$ e $\overline{Rm}$. A figura a seguir ilustra esse comportamento.

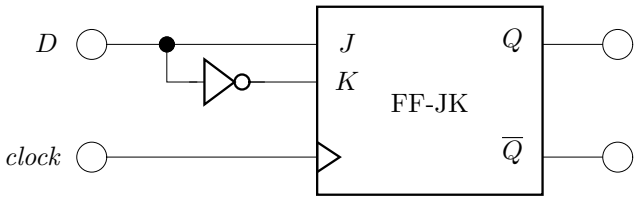


O flip-flop de tipo mestre-escravo caiu em desuso com o surgimento de entradas sensíveis a transição, sendo raramente usado hoje em dia.

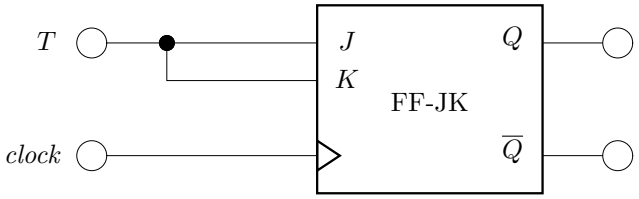
4.2.7 Conversão entre flip-flops

Os flip-flops apresentados têm comportamentos similares, e podem ser facilmente convertidos entre si, através de conexões simples e do uso de algumas portas adicionais. Vejamos alguns exemplos:

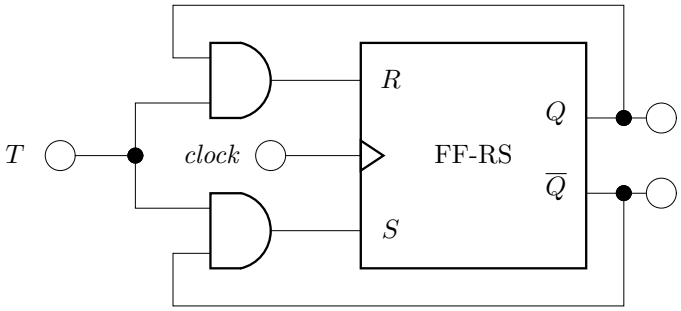
- Conversão de flip-flop JK em flip-flop D:



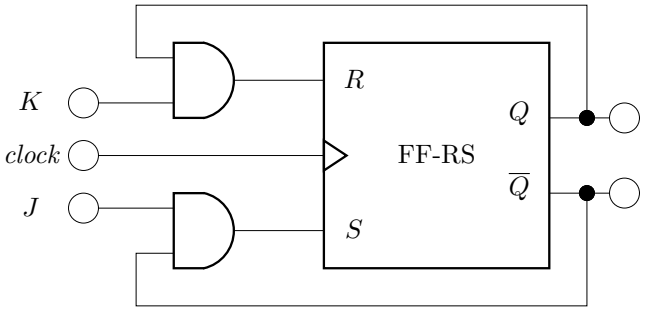
- Conversão de flip-flop JK em flip-flop T:



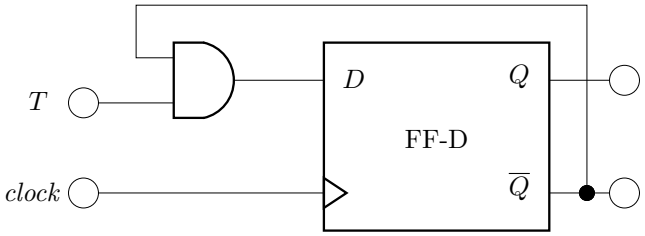
- Conversão de flip-flop RS em flip-flop T:



- Conversão de flip-flop RS em flip-flop JK:



- Conversão de flip-flop D em flip-flop T:



4.2.8 Parâmetros operacionais

Para a operação correta dos flip-flops alguns parâmetros devem ser respeitados, sobretudo no que diz respeito às características temporais dos sinais de entrada. Os parâmetros mais importantes são:

Freqüência máxima f_{max}: é a máxima freqüência admitida para o sinal de clock, ou seja, a máxima freqüência de operação do dispositivo.

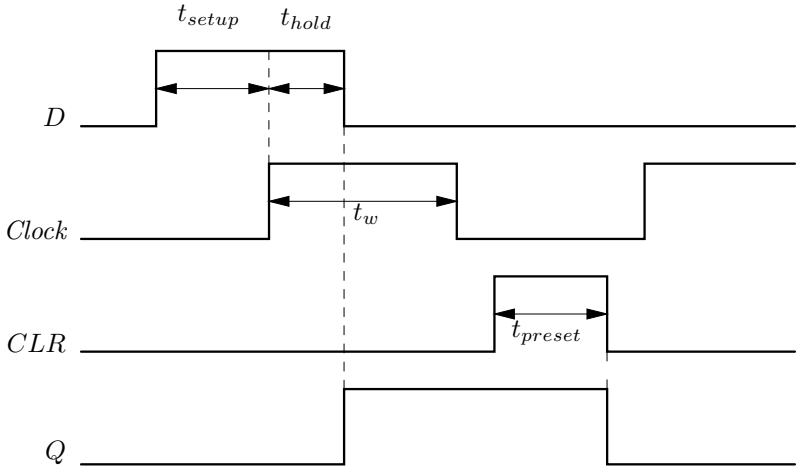
Tempo de *setup* t_{setup}: é o tempo mínimo de presença do sinal em uma entrada de dados **antes** da ocorrência do pulso de *clock*.

Tempo de permanência t_{hold}: é o tempo mínimo que o sinal deve permanecer em uma entrada de dados **após** a transição do *clock*.

Tempo de *preset* t_{preset}: é o tempo mínimo que uma entrada do tipo *preset* ou *clear* precisa estar ativa para efetuar sua função.

Largura de pulso t_w: é o tempo mínimo que o *clock* precisa permanecer em um nível alto (caso a porta seja sensível a ↑) ou baixo (caso a porta seja sensível a ↓) para que possa ser confiável.

A figura abaixo ilustra esses parâmetros, que são fortemente dependentes da tecnologia empregada para a construção do dispositivo. Alguns destes valores serão revistos no capítulo 7, mas os manuais de dados técnicos dos dispositivos (*Data Sheets* dos fabricantes) os apresentam em detalhe.



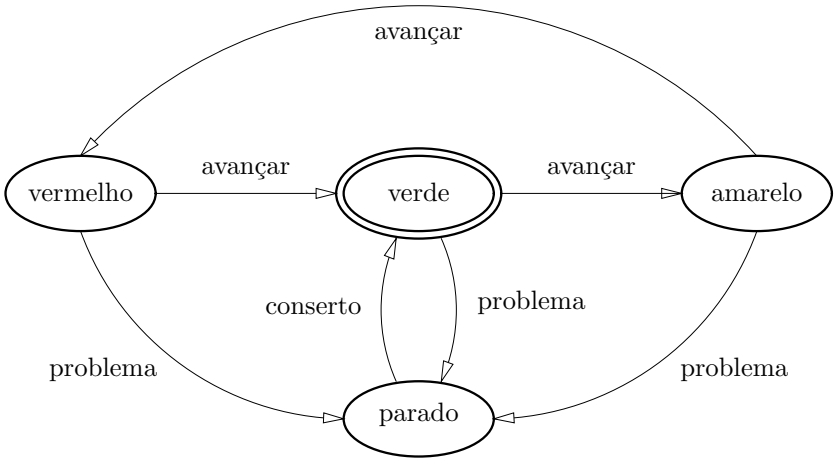
4.3 Diagramas de estado

Podemos representar o comportamento de um circuito seqüencial através de equações lógicas ou de tabelas-verdade. Essa forma de representação somente é viável para circuitos simples, contendo apenas um flip-flop. Quando a complexidade do circuito aumenta, torna-se necessário empregar outras formas de representação para definir seu comportamento de maneira fiel e sem inconsistências ou ambigüidades. Uma ferramenta muito útil para a representação do comportamento de circuitos seqüenciais complexos é o *diagrama de estados*, também chamado *autômato finito*, que veremos nesta seção.

4.3.1 Estrutura básica

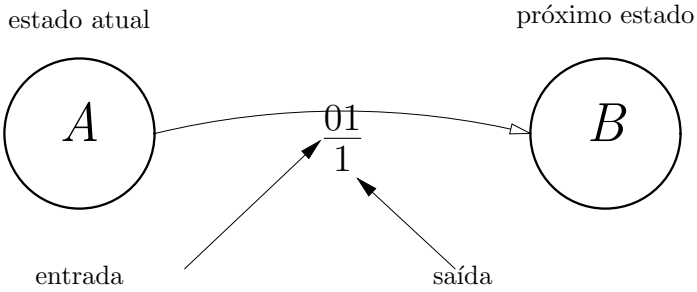
Um diagrama de estados é uma construção gráfica composta por um conjunto de estados (indicados por círculos) e de transições (indicados por arcos com setas). Os estados representam todas as situações possíveis para o sistema, e as transições indicam as mudanças de estado possíveis, e em que condições elas são provocadas (os valores das entradas que as provocam). O estado inicial do sistema é indicado por um círculo duplo.

Vejamos por exemplo o diagrama de estados de um semáforo, indicado na figura a seguir. Esse diagrama possui os estados *verde* (estado inicial), *amarelo*, *vermelho* e *parado*. As transições entre estados estão indicadas pelas setas, com os nomes dos eventos que as provocam.



No caso específico dos circuitos seqüenciais síncronos, o diagrama de estados possui algumas características importantes que devem ser levadas em conta em sua interpretação. A primeira é o seu aspecto síncrono: o diagrama deve indicar as mudanças possíveis no sistema no próximo **próximo pulso de clock**, em função do valor das entradas do circuito naquele momento.

Outra característica diz respeito à nomeação dos arcos que representam as transições: um determinado arco leva de um estado atual ao seu próximo estado, e seu rótulo indica a combinação de entrada que ativa aquela transição e o valor de saída que ela irá provocar. A figura abaixo ilustra esse funcionamento:

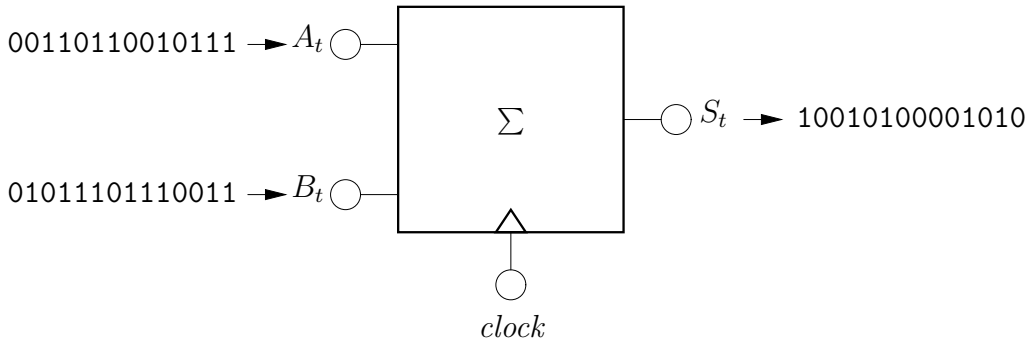


A interpretação da transição acima é a seguinte: no próximo pulso de *clock*, se o sistema estiver no estado interno A e ocorrer a entrada 01, então o sistema passará ao estado B e a saída passará a valer 1.

4.3.2 Um exemplo: o somador serial

Vamos usar o diagrama de estados para representar o comportamento de um circuito um pouco mais complexo: um somador completo serial. Este circuito tem duas entradas A_t e B_t que

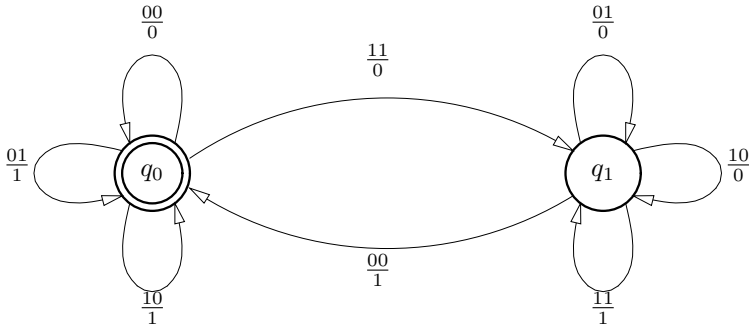
recebem os dois dígitos binários a somar, e uma saída S_t, que apresenta a soma obtida. Além disso, o circuito armazena o excesso C_{t-1} (*carry*) da soma anterior e considera esse valor na soma atual. Desta forma, podemos somar números binários longos, processando um bit por pulso de clock.



A partir das entradas A_t e B_t e do excesso da última soma C_{t-1} podemos construir a seguinte tabela-verdade para a soma atual S_t e seu excesso C_t:

C_{t-1}	A_t	B_t	S_t	C_t
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

O valor a ser memorizado é o último excesso, que constitui portanto o estado interno do sistema. Com isso podemos deduzir que o sistema possui dois estados internos: um quando $C_{t-1} = 0$, que chamaremos q_0, e outro quando $C_{t-1} = 1$, que chamaremos q_1. Podemos então construir o diagrama de estados que representa seu comportamento, considerando como entradas o par A_tB_t e como saída a soma S_t:



O diagrama acima permite representar de forma sucinta e sem ambigüidades o comportamento esperado para o somador serial. Esses diagramas serão de suma importância para a análise e projeto de circuitos seqüenciais síncronos, por isso sua estrutura deve ser perfeitamente compreendida.

4.3.3 Tabelas de estados

Uma forma alternativa de representação do comportamento de um circuito seqüencial é sob a forma de uma tabela, que indica para cada estado e para cada combinação das entradas, o próximo estado e o valor da saída (sob a forma de fração). Para o somador serial podemos construir a seguinte tabela de estados (que pode ser obtida da tabela-verdade ou do diagrama de estados):

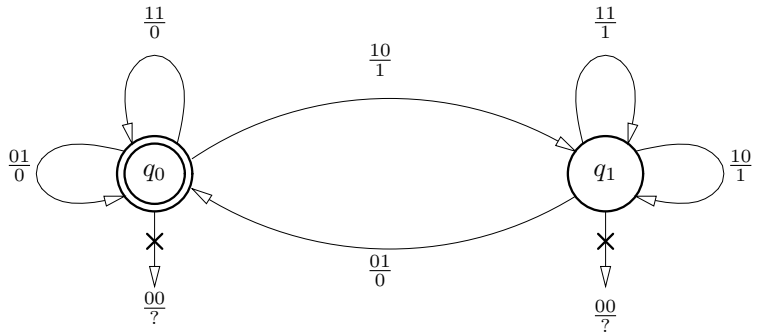
q_i \ A_tB_t	00	01	10	11
q_0	$\frac{q_0}{0}$	$\frac{q_0}{1}$	$\frac{q_0}{1}$	$\frac{q_1}{0}$
q_1	$\frac{q_0}{1}$	$\frac{q_1}{0}$	$\frac{q_1}{0}$	$\frac{q_1}{1}$

O diagrama de estados e a tabela de estados contém exatamente a mesma informação e por isso são equivalentes.

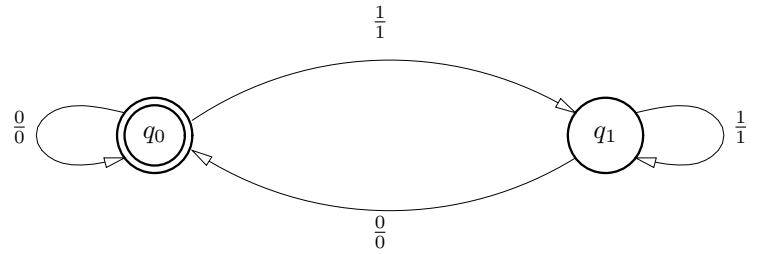
4.3.4 Diagramas de estado dos flip-flops

Podemos construir os diagramas de estado dos flip-flops a partir das tabelas-verdade e das equações lógicas vistas até o momento para os mesmos. O diagrama de estado de um flip-flop é bastante simples, porque a saída se confunde com o próximo estado interno do sistema (Q é ao mesmo tempo saída e estado). Vejamos como ficam os diagramas de estado dos principais flip-flops:

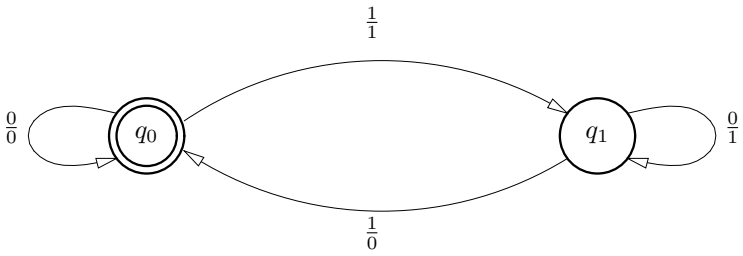
Flip-flop RS: Considerando como entrada o par $\overline{R}\ \overline{S}$ e como saída Q:



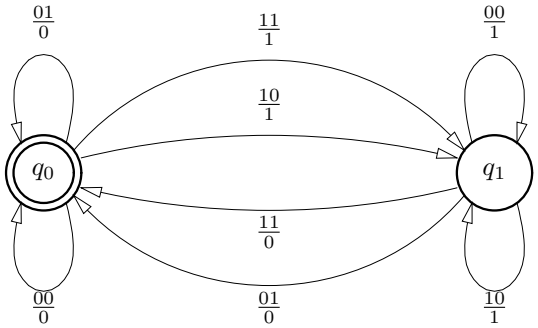
Flip-flop D: Considerando como entrada D e como saída Q:



Flip-flop T: Considerando como entrada T e como saída Q:

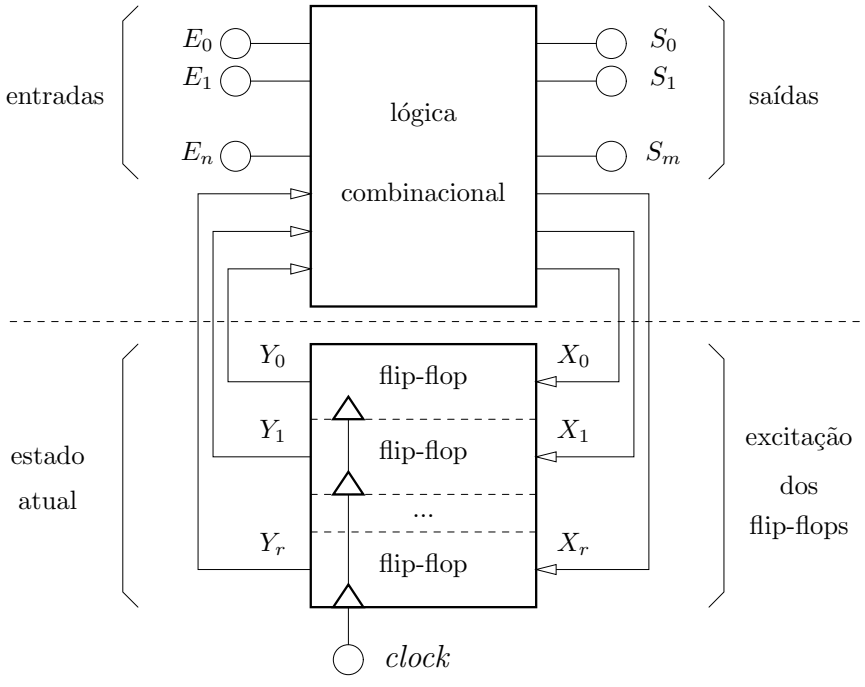


Flip-flop JK: considerando como entrada o par JK e como saída Q:



4.4 Análise de circuitos seqüenciais síncronos

O comportamento de um circuito seqüencial é definido por uma seqüência de estados que evolui em função de seu estado atual e das entradas do circuito; essa evolução é cadenciada por um sinal de *clock*. A partir das definições apresentadas no início deste capítulo, podemos propor a seguinte estrutura genérica para um circuito seqüencial síncrono:



Na estrutura acima, a parte combinacional fornece as saídas S do circuito e os sinais de controle X que irão excitar os flip-flops, ambos em função de suas entradas E e de seu estado

atual Y. Os flip-flops armazenam o estado Y do sistema, cuja evolução é cadenciada pelo sinal de *clock*.

4.4.1 Objetivo

As relações entre entradas, estado atual, saídas e próximo estado podem ser completamente especificadas através de um diagrama de estados, como vimos na seção 4.3. O objetivo principal da análise de um circuito seqüencial síncrono é a obtenção de um diagrama de estados indicando seu comportamento, e o estudo deste para a compreensão do funcionamento do circuito. Para a análise de um circuito devem ser efetuados os seguintes passos:

1. Identificar as variáveis (sinais) de entrada, de saída, de controle dos flip-flops (excitação) e de estado.
2. Obter as equações relativas à parte combinacional do circuito, através das técnicas já estudadas no capítulo 3:

$$\begin{aligned} S_i(t) &= \mathcal{F}(E_0(t), \ldots, E_n(t), Y_o(t), \ldots, Y_r(t)) \\ X_k(t) &= \mathcal{G}(E_0(t), \ldots, E_n(t), Y_o(t), \ldots, Y_r(t)) \end{aligned}$$

3. Escrever as equações de próximo estado de cada flip-flop, em função do tipo de flip-flop, de seu estado atual e das entradas de controle X_i:

$$Y_i(t+1) = \mathcal{H}(Y_i(t), X_i(t))$$

4. Montar uma tabela-verdade relacionando cada uma das entradas $E(t)$, saídas $S(t)$, estados $X(t)$ e próximos estados $X(t+1)$, com a seguinte forma:

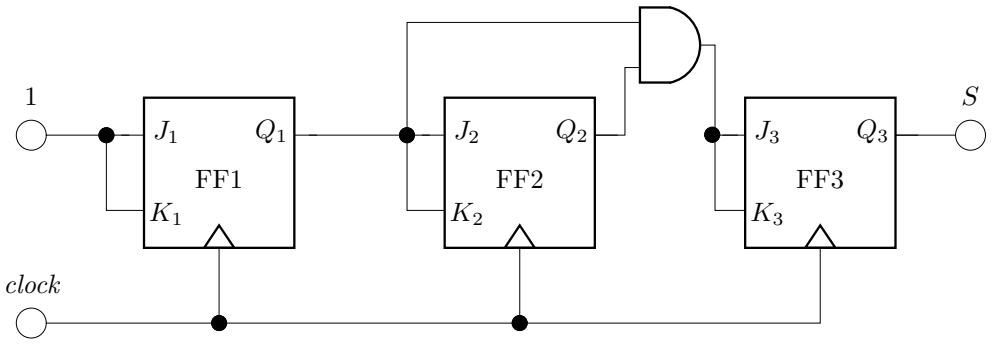
$E_0(t)$	...	$E_n(t)$	$X_0(t)$	...	$X_r(t)$	$S_0(t)$	...	$S_m(t)$	$X_0(t+1)$	...	$X_r(t+1)$
0	...	0	0	...	0	0	...	0	0	...	0
...	...	...	...	...	...	...	...	...	...	...	...

5. A partir da tabela, construir o diagrama de estados do circuito.

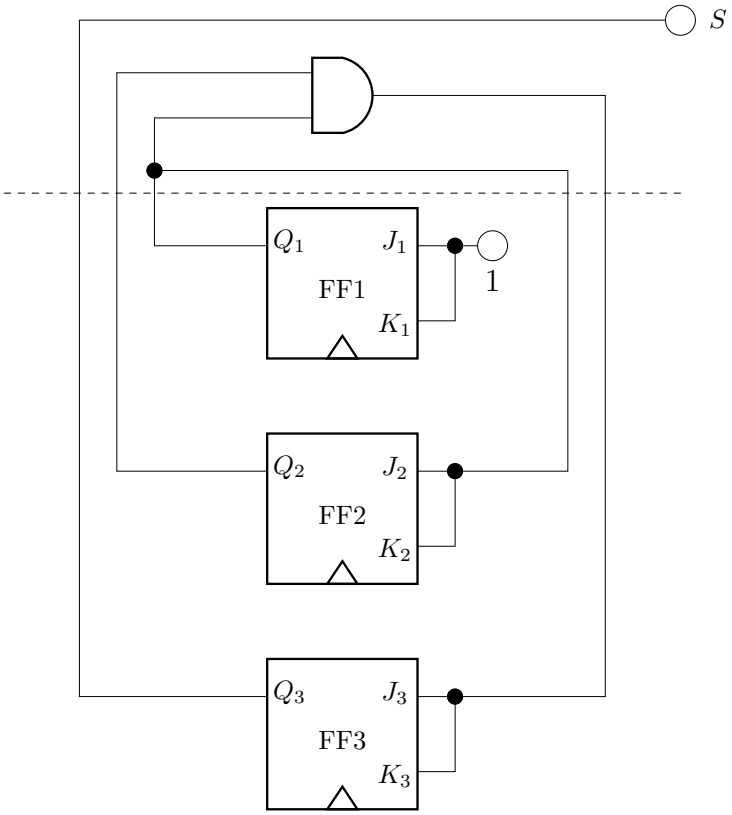
A seguir veremos como a técnica de análise é aplicada, através do estudo de dois exemplos.

4.4.2 Um exemplo

Vamos analisar o circuito da figura a seguir, para obter seu diagrama de estados:



O circuito pode ser redesenhado para colocar em evidência a separação existente entre suas partes combinacional e seqüencial (o sinal de *clock* dos flip-flops fica subentendido):



Nosso primeiro passo na análise consiste em identificar todas as variáveis envolvidas:

- Variáveis de entrada: nenhuma
- Variáveis de saída: S
- Variáveis de controle: J_1 K_1 J_2 K_2 J_3 K_3
- Variáveis de estado: Q_1 Q_2 Q_3

A seguir vamos identificar e refinar as equações:

- Saída:

$$S(t) = Q_3(t)$$

- Controle:

$$\begin{aligned} J_1(t) &= K_1(t) = 1 \\ J_2(t) &= K_2(t) = Q_1(t) \\ J_3(t) &= K_3(t) = Q_1(t)Q_2(t) \end{aligned}$$

- Próximo estado:

$$\begin{aligned} Q_1(t+1) &= J_1(t)\overline{Q}_1(t) + \overline{K}_1(t)Q_1(t) \\ &= 1\overline{Q}_1(t) + 0Q_1(t) \\ &= \overline{Q}_1(t) \end{aligned}$$

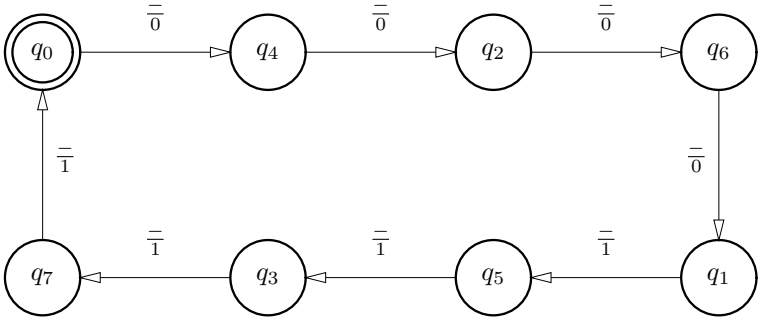
$$\begin{aligned} Q_2(t+1) &= J_2(t)\overline{Q}_2(t) + \overline{K}_2(t)Q_2(t) \\ &= Q_1(t)\overline{Q}_2(t) + \overline{Q}_1(t)Q_2(t) \\ &= Q_1(t) \oplus Q_2(t) \end{aligned}$$

$$\begin{aligned} Q_3(t+1) &= J_3(t)\overline{Q}_3(t) + \overline{K}_3(t)Q_3(t) \\ &= Q_1(t)Q_2(t)\overline{Q}_3(t) + \overline{(Q_1(t)Q_2(t))}Q_3(t) \\ &= (Q_1(t)Q_2(t)) \oplus Q_3(t) \end{aligned}$$

A partir desses dados podemos construir a tabela-verdade para esse circuito. Para sua construção, devem ser enumeradas todas as combinações possíveis para as entradas $E(t)$ e estados dos flip-flops $Q(t)$. A partir desses dados e das equações obtidas, podem ser determinadas as saídas do sistema $S(t)$ e os estados futuros dos flip-flops $Q(t+1)$. Deve-se observar que cada combinação de estados dos flip-flops corresponde a um estado interno q_i diferente para o sistema.

Entradas	Estado atual				Saídas	Próximo estado			
—	nº	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$	$S(t)$	$Q_1(t+1)$	$Q_2(t+1)$	$Q_3(t+1)$	nº
—	q_0	0	0	0	0	1	0	0	q_4
—	q_1	0	0	1	1	1	0	1	q_5
—	q_2	0	1	0	0	1	1	0	q_6
—	q_3	0	1	1	1	1	1	1	q_7
—	q_4	1	0	0	0	0	1	0	q_2
—	q_5	1	0	1	1	0	1	1	q_3
—	q_6	1	1	0	0	0	0	1	q_1
—	q_7	1	1	1	1	0	0	0	q_0

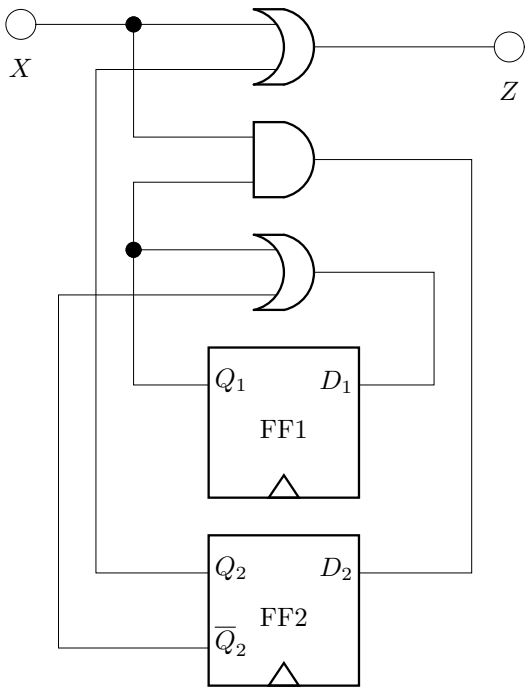
A partir da tabela-verdade obtida podemos montar o diagrama de estados do circuito, que representa seu comportamento temporal:



O diagrama de estados do circuito permite observar facilmente que a seqüência de valores da saída do circuito será composta alternadamente por quatro "0" e quatro "1": 000011110000111100001111...

4.4.3 Outro exemplo

Vamos analisar o circuito da figura a seguir, para obter seu diagrama de estados:



Primeiramente vamos identificar todas as variáveis envolvidas:

- Variáveis de entrada: X
- Variáveis de saída: Z
- Variáveis de controle: D_1 D_2
- Variáveis de estado: Q_1 Q_2

A seguir vamos identificar e refinar as equações:

- Saída:

$$Z(t) = X(t) + Q_2(t)$$

- Controle:

$$\begin{aligned} D_1(t) &= Q_1(t) + \overline{Q}_2(t) \\ D_2(t) &= X(t)Q_1(t) \end{aligned}$$

- Próximo estado:

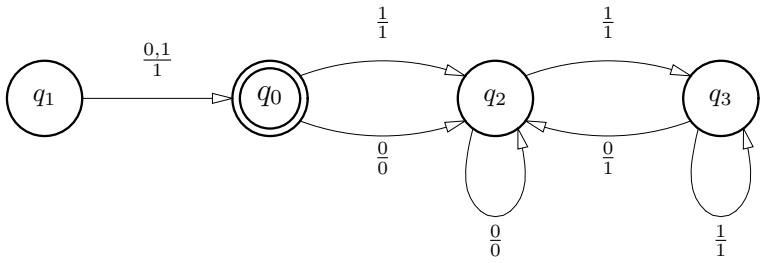
$$\begin{aligned} Q_1(t+1) &= D_1(t) \\ &= Q_1(t) + \overline{Q}_2(t) \end{aligned}$$

$$\begin{aligned} Q_2(t+1) &= D_2(t) \\ &= X(t)Q_1(t) \end{aligned}$$

A partir desses dados podemos construir a tabela-verdade para esse circuito:

Entradas	Estado atual			Saídas	Próximo estado		
$X(t)$	nº	$Q_1(t)$	$Q_2(t)$	$Z(t)$	$Q_1(t+1)$	$Q_2(t+1)$	nº
0	q_0	0	0	0	1	0	q_2
0	q_1	0	1	1	0	0	q_0
0	q_2	1	0	0	1	0	q_2
0	q_3	1	1	1	1	0	q_2
1	q_0	0	0	1	1	0	q_2
1	q_1	0	1	1	0	0	q_0
1	q_2	1	0	1	1	1	q_3
1	q_3	1	1	1	1	1	q_3

Finalmente podemos montar o diagrama de estados do circuito:



Como vimos anteriormente, podemos também representar o diagrama de estados na forma de uma tabela de estados:

estado atual	entradas	
	$X(t)=0$	$X(t)=1$
q_0	$\frac{q_2}{0}$	$\frac{q_2}{1}$
q_1	$\frac{q_0}{1}$	$\frac{q_0}{1}$
q_2	$\frac{q_2}{0}$	$\frac{q_3}{1}$
q_3	$\frac{q_2}{1}$	$\frac{q_3}{1}$

4.5 Projeto de circuitos seqüenciais síncronos

O projeto de circuitos seqüenciais síncronos segue uma abordagem análoga à utilizada no processo de análise, mas em sentido inverso. Desta forma, o projeto pode ser decomposto nos seguintes passos:

1. Descrição completa da operação desejada para o circuito, envolvendo:
   - identificação das entradas e saídas;
   - identificação dos estados internos (n flip-flops para 2^n estados);
   - definição do comportamento desejado, através de um diagrama de estados.
2. Determinação da tabela-verdade para o circuito, com base nos valores possíveis para as entradas e estados internos do circuito. Esta tabela também deverá conter colunas para os sinais de excitação dos flip-flops, em função do tipo de flip-flop escolhido.
3. Minimização das funções correspondentes à parte combinacional do circuito, ou seja, saídas e controles dos flip-flops.
4. Construção do circuito final.

Para a determinação da tabela-verdade teremos de empregar "tabelas de excitação" para os flip-flops, que permitem determinar os valores das entradas de controle em função da transição de estado desejada. Essas tabelas podem ser facilmente deduzidas a partir das tabelas-verdade dos flip-flops. Por exemplo, para o flip-flop JK temos:

Q_t	J_t	K_t	Q_{t+1}
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	1
1	1	1	0

$\longrightarrow$

Q_t	Q_{t+1}	J_t	K_t
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

De maneira similar podemos obter a tabela de excitação para o flip-flop D:

Q_t	D_t	Q_{t+1}
0	0	0
0	1	1
1	0	0
1	1	1

$\longrightarrow$

Q_t	Q_{t+1}	D_t
0	0	0
0	1	1
1	0	0
1	1	1

4.5.1 Um exemplo

Neste exemplo vamos projetar um circuito para implementar o somador serial cujo comportamento foi apresentado na seção 4.3.2. Nosso primeiro passo é determinar a operação desejada para o circuito, e para isso vamos determinar os seguintes dados:

- Entradas: os bits a somar: $A(t)$ e $B(t)$
- Saída: a soma acumulada: $S(t)$
- Estados internos: dois estados, indicando o excesso da última operação efetuada: *sem carry* (q_0) e *com carry* (q_1).

- Comportamento: indicado no diagrama de estados apresentado na seção 4.3.2.

Como temos dois estados internos, precisaremos de apenas uma variável de estado $Q(t)$ e por conseqüência apenas um flip-flop.

Neste ponto podemos montar a tabela-verdade do circuito. Já conhecemos A_t, B_t, S_t, Q_t e Q_{t+1}, apresentados no diagrama de estados. Devemos então escolher um tipo de flip-flop para a implementação e obter as colunas correspondentes à sua excitação. Essas colunas serão determinadas a partir de cada par "estado atual → estado futuro" para cada flip-flop, usando a tabela de excitação do flip-flop escolhido. Escolhendo um flip-flop JK ou D, nossa tabela de estados assume a seguinte forma:

entradas		estado atual	saídas	próximo estado	excitação dos flip-flops		
A_t	B_t	Q_t	S_t	Q_{t+1}	J_t	K_t	D_t
0	0	0	0	0	0	X	0
0	1	0	1	0	0	X	0
1	0	0	1	0	0	X	0
1	1	0	0	1	1	X	1
0	0	1	1	0	X	1	0
0	1	1	0	1	X	0	1
1	0	1	0	1	X	0	1
1	1	1	1	1	X	0	1

Com a tabela construida, podemos passar à determinação das funções combinacionais necessárias à geração da saída e das excitações do flip-flop. Para S_t podemos construir o seguinte mapa de Karnaugh:

Q_t \ A_tB_t	00	01	11	10
0		1		1
1	1		1	

$$\longrightarrow \quad S_t = A_t \oplus B_t \oplus Q_t$$

Para J_t temos:

Q_t \ A_tB_t	00	01	11	10
0			1	
1	X	X	X	X

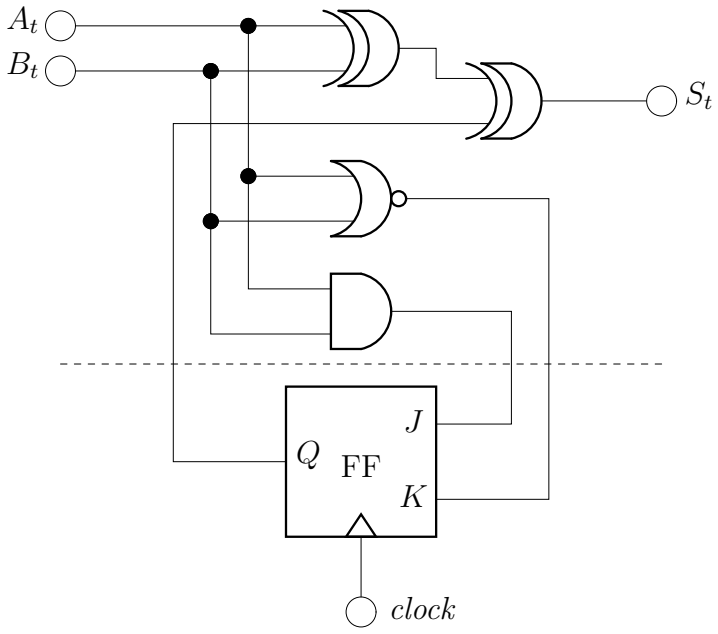
$$\longrightarrow \quad J_t = A_t B_t$$

Para K_t temos:

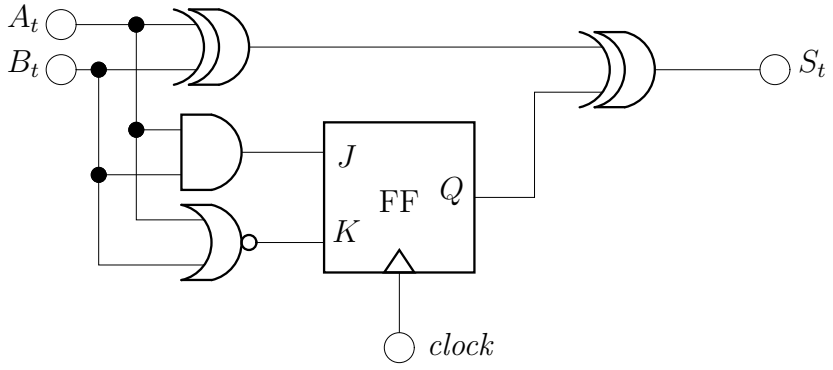
Q_t \ A_tB_t	00	01	11	10
0	X	X	X	X
1	1			

$$\longrightarrow \quad K_t = \overline{A}_t\overline{B}_t = \overline{A_t + B_t}$$

A partir das funções mínimas obtidas podemos sintetizar a parte combinacional do circuito, que acoplada ao flip-flop nos dará o circuito final:

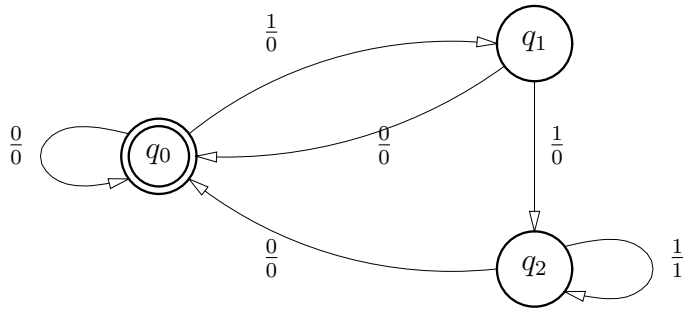


O circuito acima pode ser apresentado de uma forma mais agradável e intuitiva:



4.5.2 Outro exemplo

Vamos construir um circuito para implementar o comportamento descrito através do seguinte diagrama de estados:



Nosso primeiro passo é determinar a operação desejada para o circuito, e para tal vamos determinar os seguintes dados:

- Entrada: uma entrada, que chamaremos E_t
- Saída: uma saída, que chamaremos S_t
- Estados internos: o sistema possui três estados (q_0, q_1 e q_2), o que nos leva a um mínimo de dois flip-flops e portanto duas variáveis de estado $Q1_t$ e $Q2_t$, cuja combinação nos permitirá a indicação do estado do sistema:

$Q1_t$	$Q2_t$	estado
0	0	q_0
0	1	q_1
1	0	q_2
1	1	sem uso

- Comportamento: indicado no diagrama de estados apresentado acima.

Neste ponto podemos montar a tabela-verdade do circuito. Já conhecemos E_t, S_t, $Q1_t$, $Q2_t$, $Q1_{t+1}$ e $Q2_{t+1}$, apresentados no diagrama de estados. Escolhendo um flip-flop JK, devemos então determinar $J1_t$, $K1_t$, $J2_t$ e $K2_t$:

entrada	estado atual			saída	próximo estado			excitação dos flip-flops			
E_t	nº	$Q1_t$	$Q2_t$	S_t	nº	$Q1_{t+1}$	$Q2_{t+1}$	$J1_t$	$K1_t$	$J2_t$	$K2_t$
0	q_0	0	0	0	q_0	0	0	0	X	0	X
0	q_1	0	1	0	q_0	0	0	0	X	X	1
0	q_2	1	0	0	q_0	0	0	X	1	0	X
0	—	1	1	X	—	X	X	X	X	X	X
1	q_0	0	0	0	q_1	0	1	0	X	1	X
1	q_1	0	1	0	q_2	1	0	1	X	X	1
1	q_2	1	0	1	q_2	1	0	X	0	0	X
1	—	1	1	X	—	X	X	X	X	X	X

Com a tabela assim construida, podemos passar à determinação das funções combinacionais necessárias à geração da saída e das excitações dos flip-flops, em função da entrada E_t e dos estados $Q1_t$ e $Q2_t$. Para S_t podemos construir o seguinte mapa de Karnaugh:

E_t \ $Q1_tQ2_t$	00	01	11	10
0			X	
1			X	1

$\longrightarrow \quad S_t = E_t Q1_t$

Para as excitações $J1_t$, $K1_t$, $J2_t$ e $K2_t$ temos:

E_t \ $Q1_tQ2_t$	00	01	11	10
0			X	X
1		1	X	X

$\longrightarrow \quad J1_t = E_t Q2_t$

E_t \ $Q1_tQ2_t$	00	01	11	10
0	X	X	X	1
1	X	X	X	

$\longrightarrow \quad K1_t = \overline{E}_t$

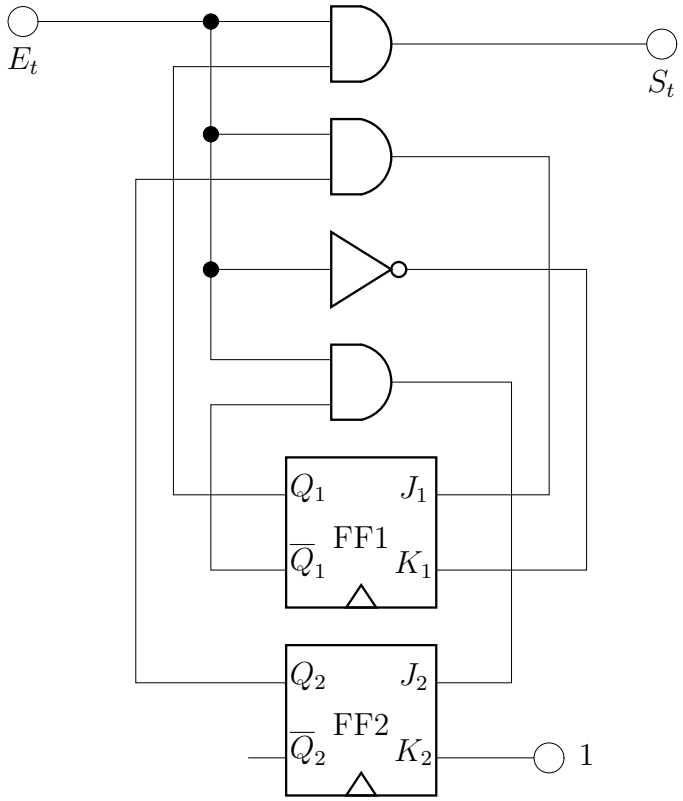
E_t ^{Q1_tQ2_t}	00	01	11	10
0		X	X	
1	1	X	X	

$\longrightarrow \quad J2_t = E_t\overline{Q1}_t$

E_t ^{Q1_tQ2_t}	00	01	11	10
0	X	1	X	X
1	X	1	X	X

$\longrightarrow \quad K2_t = 1$

A partir das funções mínimas obtidas podemos sintetizar a parte combinacional do circuito, que acoplada aos flip-flops nos dará o circuito final:



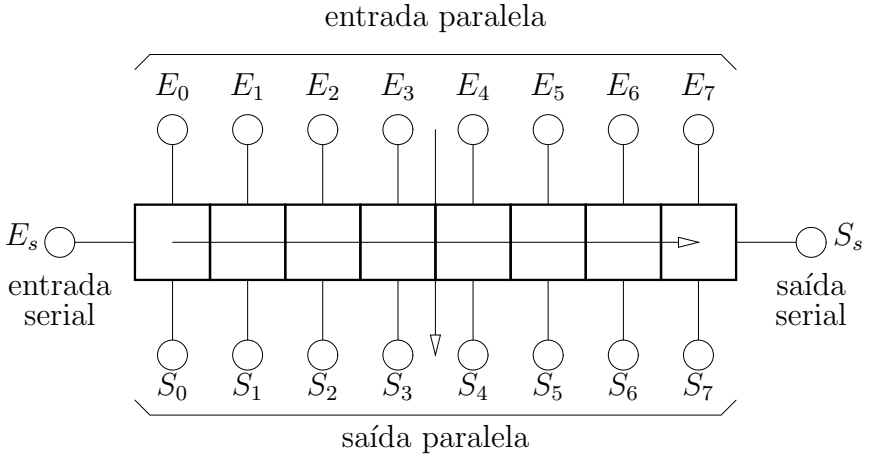
4.6 Principais circuitos seqüenciais síncronos

Nesta seção veremos alguns circuitos seqüenciais síncronos de uso bastante freqüente, e portanto facilmente encontrados na forma de chips completos. Esses circuitos podem também ser sintetizados através da técnica de projeto vista na seção anterior. Abordaremos aqui os registradores de deslocamento e os contadores.

4.6.1 Registradores de deslocamento

Um registrador de deslocamento é um arranjo linear de n flip-flops capaz de armazenar n bits de informação. A cada pulso de *clock* os dados podem ser deslocados uma posição para a direita ou para a esquerda, de acordo com a implementação do registrador, o que justifica o nome desse dispositivo. Os dados podem ser carregados no registrador de forma paralela (todos simultaneamente, através de entradas especiais) ou seqüencial, através de um dos extremos do

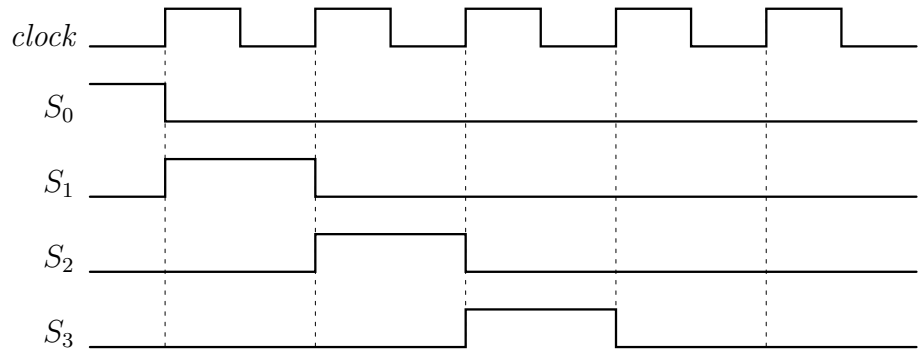
registrador e fazendo uso do mecanismo de deslocamento de bits. A figura a seguir mostra o diagrama simplificado de um registrador de deslocamento para a direita com 8 bits (foram omitidos o sinal de *clock* e as entradas de controle):



As duas setas indicam os sentidos possíveis do fluxo de dados no interior do registrador de deslocamento:

- fluxo seqüencial: os dados fluem para a direita na cadência do *clock*. O primeiro flip-flop assume o valor da entrada serial E_s, e o dado do outro extremo, que já atravessou o registrador, é apresentado na saída serial S_s. Portanto, a cada pulso de *clock* o dado mais antigo (à direita) é descartado e um novo dado entra no registrador (à esquerda). Desta forma, um dado demora n pulsos de *clock* para atravessar um registrador de deslocamento de n bits.
- fluxo paralelo: o conteúdo do registrador pode ser acessado através das saídas paralelas, e pode ser totalmente alterado fazendo-se uso das entradas paralelas. Essas operações normalmente podem ser efetuadas de maneira assíncrona.

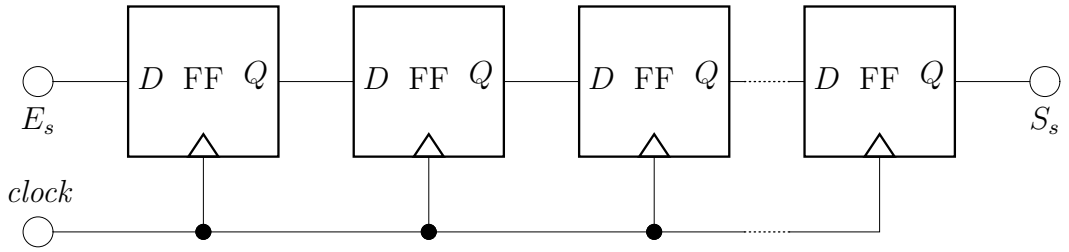
Podemos compreender melhor o funcionamento de um registrador de deslocamento analisando seu comportamento temporal. O diagrama de tempo a seguir mostra o comportamento de um registrador de deslocamento de 4 bits com o conteúdo inicial 1000 (apenas um bit ativo no primeiro registro), e com $E_s = 0$:



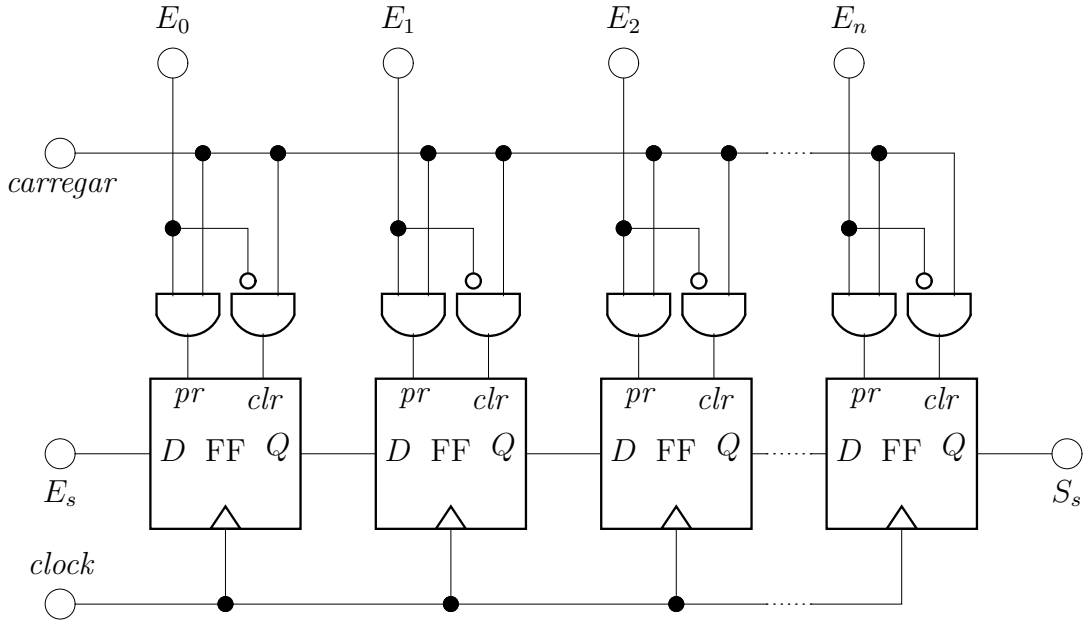
Segundo a maneira como a informação flui no interior de um registrador de deslocamento, podemos classificá-lo em quatro grupos:

- Série-série: como cada bit demora n pulsos de *clock* para atravessar o registrador, este dispositivo é usado em linhas de retardo digitais, para criar atrasos no sinal digital (por exemplo, em câmaras de eco digitais podem ser usados registradores com milhares de bits).
- Série-paralelo: os dados são carregados em série e retirados através da saída paralela. Esta estrutura é normalmente usada para em comunicação de dados, para converter sinais seriais (na linha telefônica) em sinais paralelos (no interior do computador).
- Paralelo-série: exerce a função inversa do anterior, sendo por isso também empregado em comunicação de dados.
- Paralelo-paralelo: a carga e descarga do registrador é feita através das portas paralelas. Pode ser usado para deslocar (*shift*) valores binários, ou para armazená-los temporariamente (buffer).

Veremos a seguir algumas estruturas simples para a implementação dos registradores de deslocamento. Empregamos flip-flops de tipo D por serem os que melhor se enquadram neste tipo de aplicação, e por sua simplicidade. Para construir um registrador de deslocamento série-série, basta acoplarmos flip-flops tipo D em série:

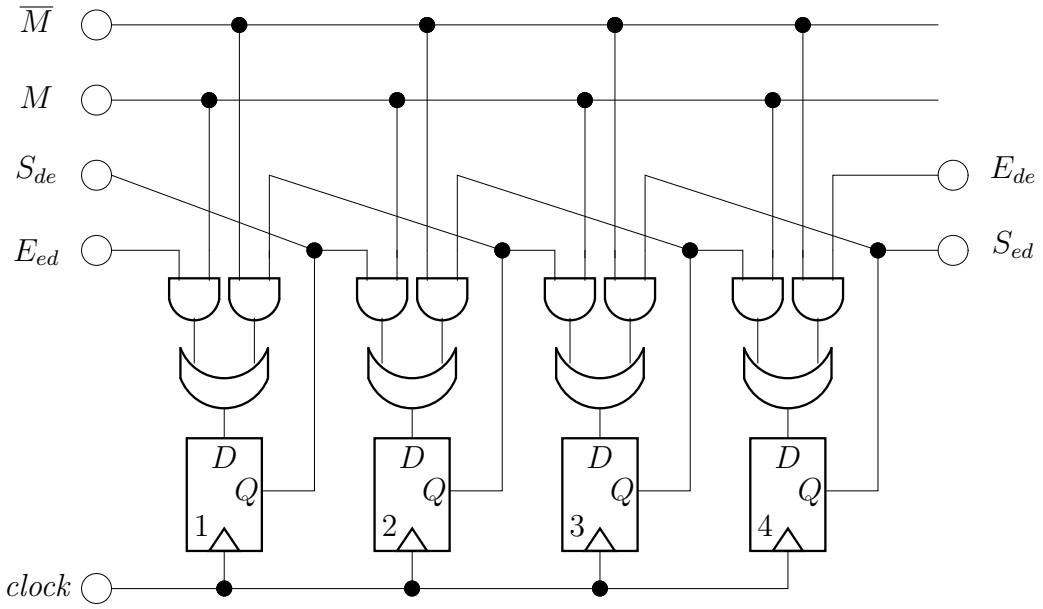


O registrador série-paralelo tem uma implementação similar, bastando extrair a saída individual de cada flip-flop para compor a saída paralela. Para construir os registradores paralelo-série e paralelo-paralelo, usamos as entradas *preset* e *clear* de cada flip-flop para carregar o registrador com os valores presentes na entrada paralela:



Podemos também construir um registrador bidirecional, cujo sentido de deslocamento (da esquerda para a direita ou vice-versa) é estabelecido através de uma porta de controle M:

- $M = 1$: esquerda → direita: $E_{ed} \rightarrow FF_1 \rightarrow FF_2 \rightarrow FF_3 \rightarrow FF_4 \rightarrow S_{ed}$
- $M = 0$: direita → esquerda: $E_{de} \rightarrow FF_4 \rightarrow FF_3 \rightarrow FF_2 \rightarrow FF_1 \rightarrow S_{de}$



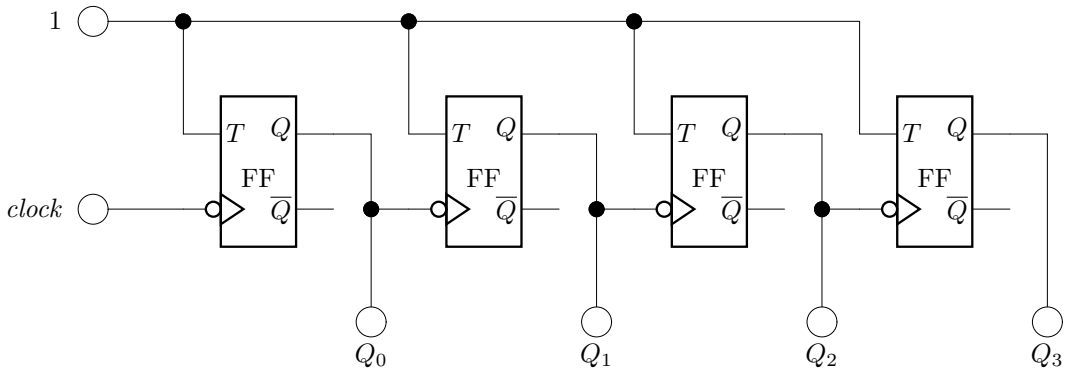
4.6.2 Contadores

Contadores são circuitos seqüenciais que permitem contar pulsos de uma entrada, apresentando a contagem sob a forma de um número binário, em uma saída com n bits. Os contadores têm muitas aplicações, dentre as quais a contagem de eventos, a divisão de freqüência, o sequenciamento de operações, etc.

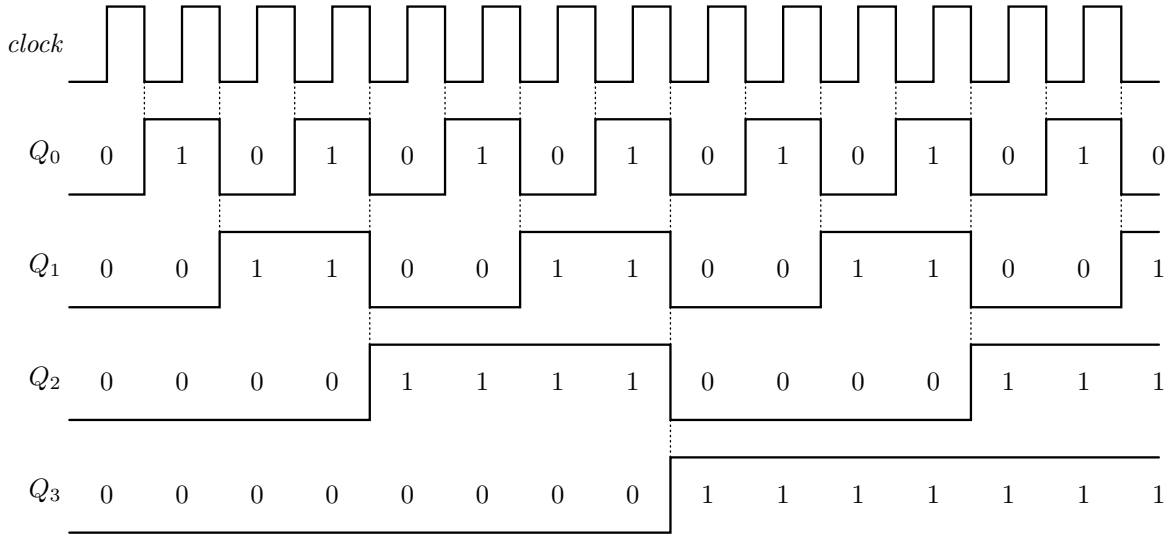
Podemos classificar os contadores segundo diversos parâmetros:

- Sincronismo: um contador pode ser *síncrono*, quando todos os seus flip-flops estão sob o comando de um mesmo *clock*, ou *assíncronos*, quando os flip-flops podem ser excitados por *clocks distintos*.
- Sentido: um contador pode contar de forma *ascendente* ou *descendente*, ou ambos.
- Programação: um contador pode efetuar uma contagem entre dois extremos fixos, ou podemos estabelecer os valores inicial e final para a contagem.

O contador de implementação mais simples é o assíncrono ascendente, que pode ser facilmente obtido através da associação de flip-flops tipo T em cascata:

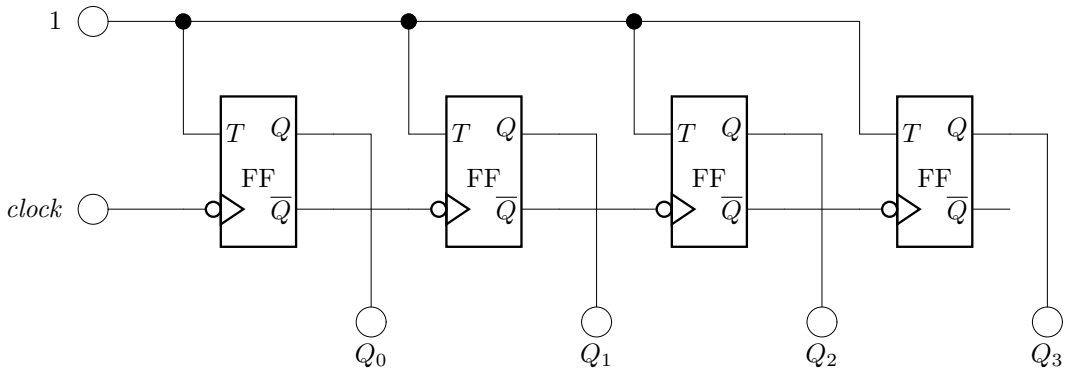


O circuito acima permite contar 16 passos, de 0000_2 a 1111_2, em sentido ascendente, como mostra o diagrama temporal a seguir:

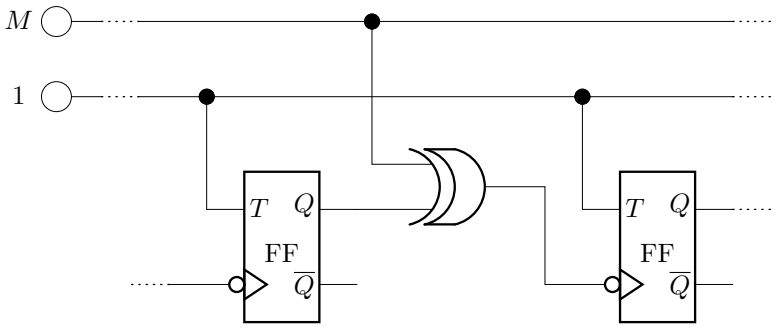


Caso seja necessária uma contagem com saída seqüencial, podemos associar à saida do contador um decodificador 4×16, e assim teremos na saída do decodificador uma seqüência $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \cdots \rightarrow S_{15}$ que evolui a cada pulso do *clock*. Também podemos ver o circuito anterior como um divisor de freqüência: para um sinal de *clock* de freqüência f_c, a saída Q_0 tem freqüência $f_c/2$, a saída Q_1 tem freqüência $f_c/4$ e assim por diante.

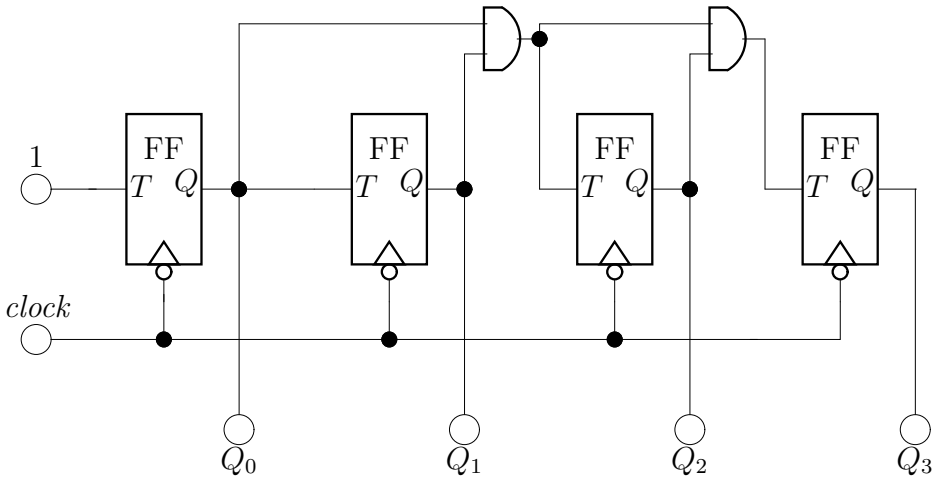
Para a construção de um contador assíncrono descendente basta usar as saídas $\overline{Q}$ para associar os flip-flops. O contador da figura a seguir efetua o ciclo $0000 \rightarrow 1111 \rightarrow 1110 \rightarrow 1101 \rightarrow 1100 \rightarrow \cdots \rightarrow 0000$:

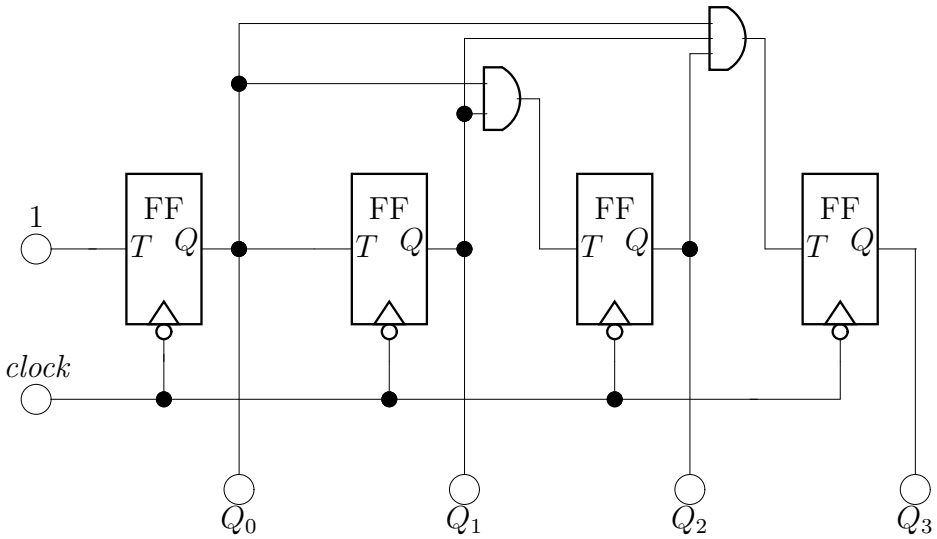


Podemos associar os dois circuitos e obter um contador ascendente-descendente, sob o comando de uma porta de controle M: caso $M = 1$ teremos uma contagem ascendente, e caso $M = 0$ ela será descendente:



O projeto de contadores síncronos pode ser feito facilmente através da técnica de projeto descrita neste capítulo. Duas estruturas de contadores síncronos são bastante conhecidas: o contador síncrono *com transporte série* ou com *transporte paralelo*. Estes nomes indicam a forma como a evolução dos bits menos significativos do contador são consideradas na evolução dos bits mais significativos. Ambas as estruturas são apresentadas a seguir (para 4 bits):



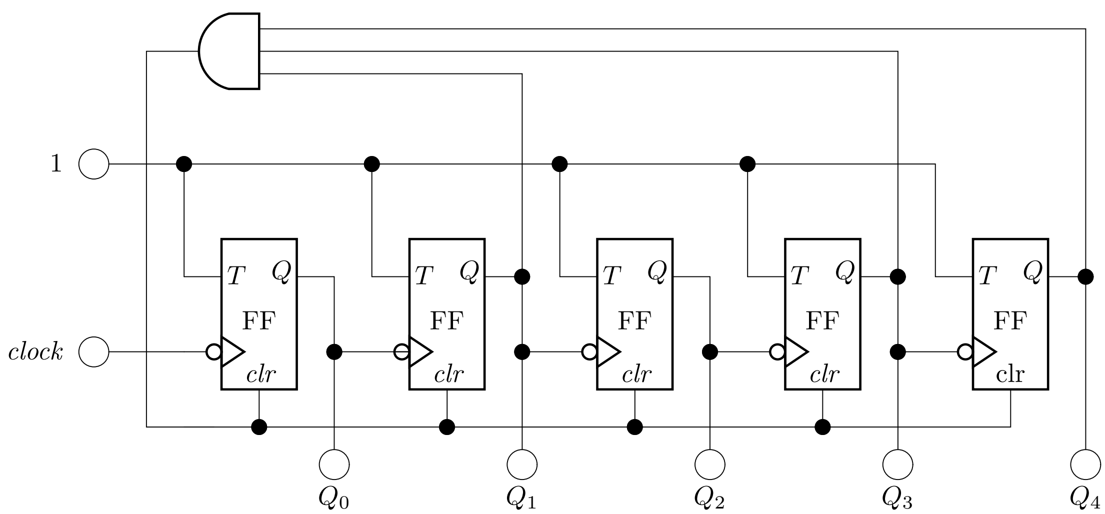


Para a construção de contadores síncronos decrescentes ou bidirecionais podem ser seguidos os mesmos procedimentos apresentados para os contadores assíncronos, ou seja, usar $\overline{Q}$ ao invés de Q ou associar uma lógica de controle para selecionar entre Q e $\overline{Q}$.

Os contadores vistos até o momento permitem contar $M = 2^n$ passos, onde n é o número de flip-flops usados. Podemos no entanto construir contadores para operar com um valor M qualquer, que chamaremos *contadores em módulo M*. Por exemplo, um contador em módulo 5 permite contar 5 passos, percorrendo o ciclo $000 \rightarrow 001 \rightarrow 010 \rightarrow 011 \rightarrow 100 \rightarrow 000 \rightarrow \cdots$.

O primeiro passo para a construção de um contador em módulo M é a determinação do número de flip-flops necessários para a contagem. Isso pode ser feito com base no número de dígitos binários necessários para representar os estados do contador. Por exemplo, um contador em módulo 6 ($6_{10} = 110_2$) irá precisar de 3 flip-flops, enquanto um contador em módulo 26 irá necessitar de 5 flip-flops (pois $26_{10} = 11010_2$).

A seguir deve-se construir um contador assíncrono usando os flip-flops necessários. Para obter a contagem até M, deve-se conectar todas as saídas ativas no estado M a uma porta AND que irá ativar as entradas $\overline{clr}$ de todos os flip-flops. Assim, quando a contagem atingir M, a porta AND será ativada e os flip-flops voltarão a zero, reiniciando a contagem. O circuito a seguir usa essa técnica para implementar um contador em módulo 26 (ou seja, que vai de 0 a 25):



4.7 Exercícios

1. Projete um registrador em anel de 3 bits, com deslocamento para a direita. Qualquer que seja o valor inicial, o valor de cada bit é deslocado para a direita, e o último à direita passa a ser o primeiro à esquerda. O bit da esquerda é o mais significativo. Use flip-flops de tipo D.

2. Projete um contador de 3 bits com *clear*, usando flip-flops de tipo T.

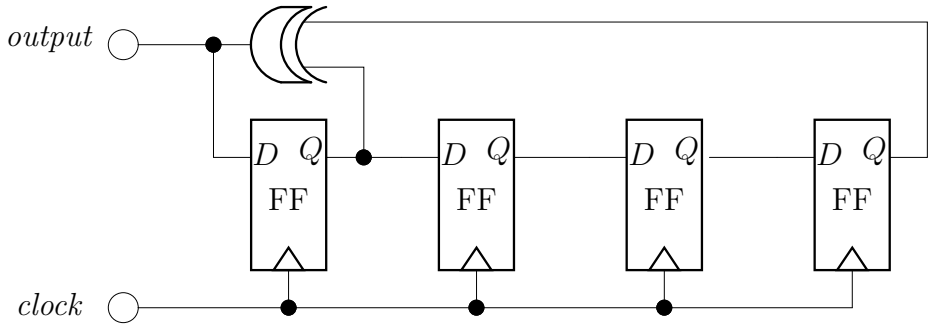
3. Projete um contador crescente-decrescente de 2 bits em código Gray. Se a entrada $U = 1$ o contador é crescente, e senão é decrescente. Esse tipo de contador é usado para controlar motores de passo bidirecionais.

4. Projete um contador programável obedecendo as características da tabela a seguir, usando flip-flops de tipo JK:

entradas		modo de
x_1	x_2	operação
0	0	não muda
0	1	módulo 3
1	0	módulo 5
1	1	módulo 7

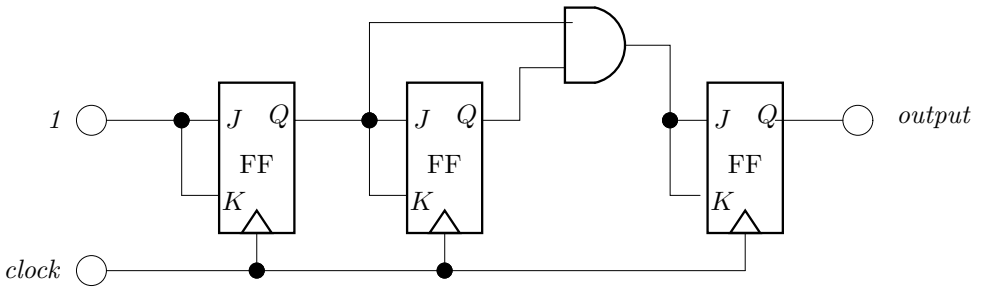
5. Considere um somador paralelo de 4 bits, no qual X é o sinal de controle, e A e B são os registradores de 4 bits. Se $X = 0$ nenhuma operação é efetuada, mas se $X = 1$ é realizada a soma dos registradores A e B, e o resultado é depositado no registrador A. Projete o circuito do somador, considerando que o mesmo pode ser decomposto em células somadoras de 1 bit (o excesso ou *carry* de cada estágio deve servir como entrada para o estágio seguinte).

6. Construa o diagrama de estados do gerador pseudo-aleatório de ruído cujo circuito é apresentado a seguir:

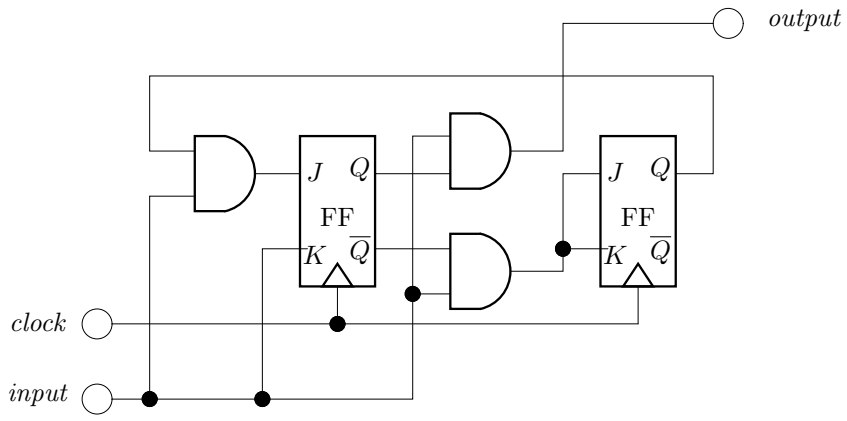


7. Projete um circuito detector de paridade par serial para palavras de 8 bits, usando flip-flops de tipo D, e responda: a) quantos pulsos de relógio são necessários para detectar a paridade de uma palavra; b) compare o circuito obtido com o detector de paridade paralelo apresentado na seção 3.6.

8. Obtenha o diagrama de estados do circuito a seguir:



9. Obtenha o diagrama de estados do circuito a seguir:



10. Projete um divisor de freqüência por 10.

Capítulo 5

Circuitos Complementares

Denominamos *circuitos complementares* aqueles que permitem gerar ou modificar um determinado sinal no tempo. Como exemplos de tais circuitos temos os temporizadores, osciladores (geração de sinais de *clock*), modificadores de largura de pulsos, limitadores, detectores de pulsos, etc. Os circuitos complementares estudados neste capítulo são muito utilizados nos projetos de sistemas digitais. Basicamente, neste texto, nos limitaremos a estudar duas classes de circuitos complementares: os circuitos multivibradores e o *Schmitt-Trigger*, por serem de uso mais freqüente na prática.

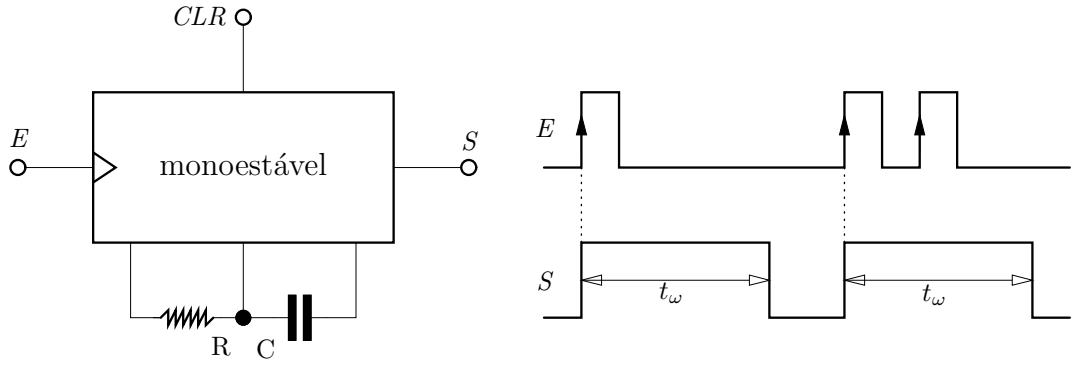
5.1 Circuitos multivibradores

Circuito multivibrador é o nome genérico para circuitos cuja saída pode variar entre dois estados distintos. São circuitos utilizados como osciladores ou circuitos modificadores de sinais. Existem três classes diferentes de circuitos multivibradores:

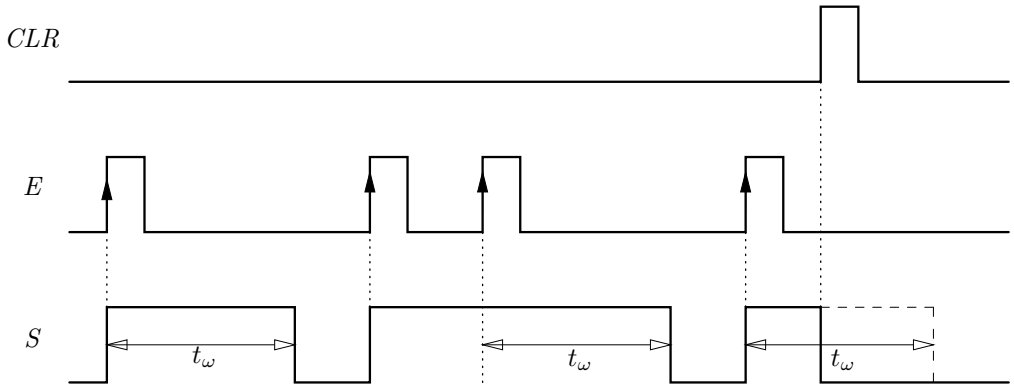
- **Biestáveis**: possuem dois estados estáveis. Como exemplo mais corrente podemos citar os diversos tipos de *flip-flops*.
- **Monoestáveis**: possuem um estado estável, no qual podem permanecer indefinidamente e um estado "quase-estável", no qual pode permanecer durante um intervalo de tempo pré-determinado e geralmente constante, geralmente denominado *largura de pulso* (t_ω).
- **Astáveis**: não possuem estados estáveis, e por isso sua saída oscila indefinidamente entre os dois estados possíveis, com uma freqüência pré-determinada.

5.1.1 Circuitos mono-estáveis

Como o próprio nome indica, os monoestáveis são circuitos que possuem um único estado estável, no qual podem permanecer indefinidamente e um estado "quase-estável" no qual podem permanecer durante um intervalo de tempo t_ω, normalmente determinado pela constante de tempo de um circuito RC acoplado ao dispositivo. Desta forma, a saída do monoestável é um pulso de onda quadrada, com duração t_ω. A transição do estado estável para o "quase-estável" só se realiza se o circuito for disparado (*triggered*) externamente, enquanto o retorno ao estado estável se dá automaticamente após o tempo t_ω, como mostra a figura:

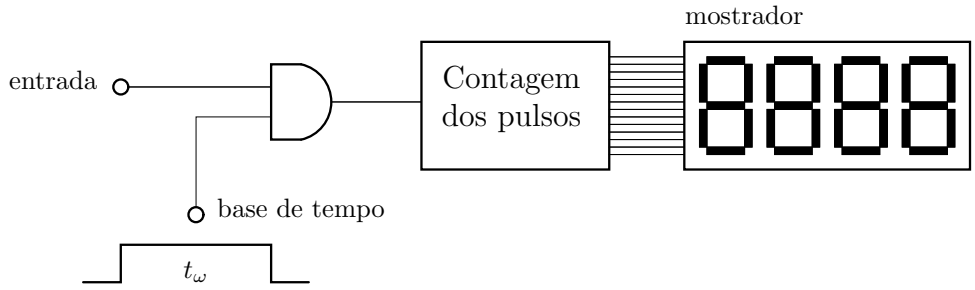


A duração t_ω do pulso é normalmente proporcional à constante de tempo do circuito RC acoplado ao monoestável ($t_\omega \propto RC$), mas pode ser alterada em algumas circunstâncias. Alguns monoestáveis aceitam ser "redisparados" (*retriggered*) durante o ciclo ativo, para aumentar a largura do pulso de saída. Além disso, a entrada CLR pode ser empregada para abreviar o ciclo ativo. A figura a seguir ilustra essas duas situações:

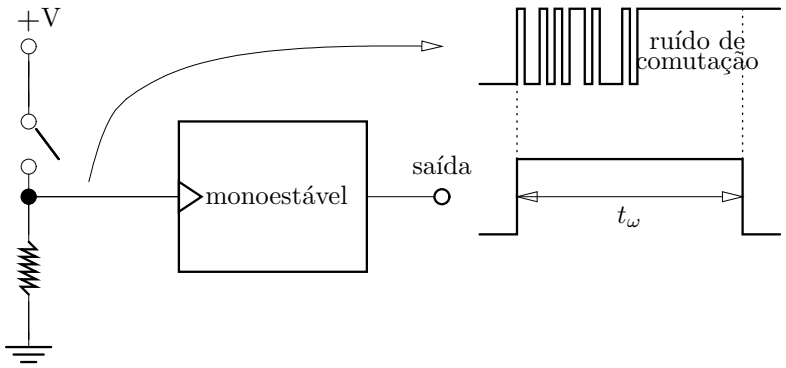


Entre as principais aplicações dos monoestáveis podemos ressaltar:

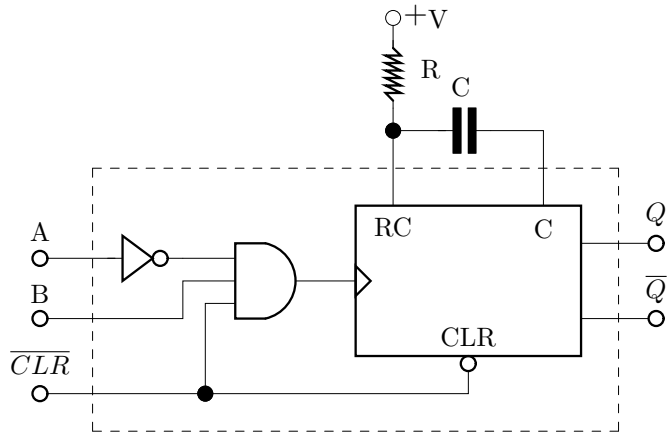
- Aumentar ou diminuir a largura de um pulso. Aplicando-se este na entrada do monoestável, a saída irá fornecer um pulso de largura constante.
- Para gerar pulsos de largura fixa conhecida. Por exemplo, na construção de um freqüencímetro digital devemos contar os pulsos de entrada durante um determinado intervalo de tempo ($1s$, $1\mu s$, etc.):



- Para filtrar sinais de entrada com ruído. Por exemplo, podemos usar um monoestável para retirar o ruído de comutação do sinal gerado por uma chave de seleção:

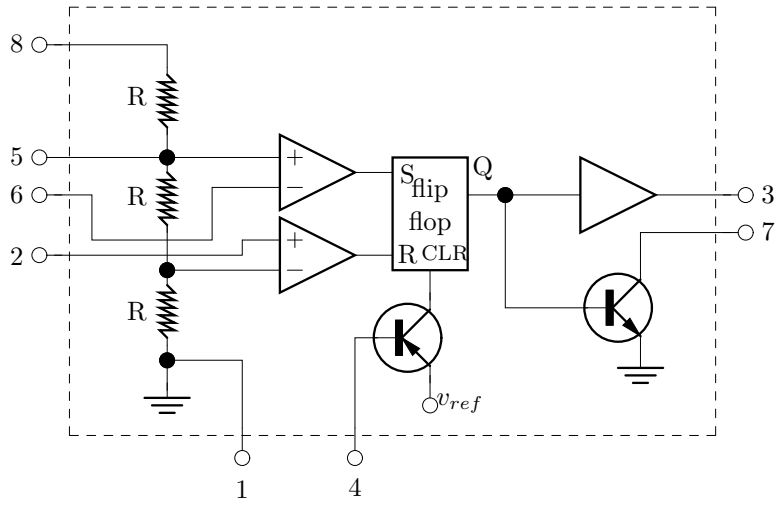


Existem diversos monoestáveis comerciais, como por exemplo o circuito TTL 74123, cujo funcionamento é apresentado a seguir (t_ω =?):



$\overline{CLR}$	A	B	Q	$\overline{Q}$
0	X	X	0	1
X	1	X	0	1
X	X	0	0	1
1	0	$\uparrow$	$\sqcap$	$\sqcup$
1	$\downarrow$	1	$\sqcap$	$\sqcup$
$\uparrow$	0	1	$\sqcap$	$\sqcup$

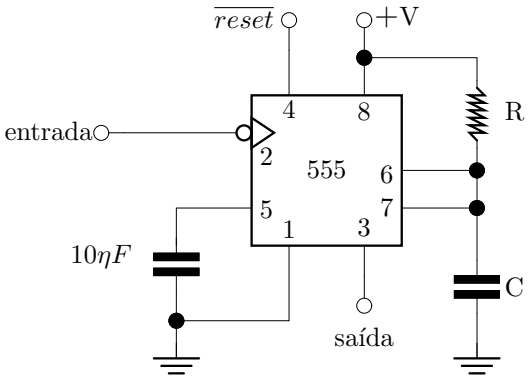
Podemos também implementar um monoestável utilizando o 555, um circuito integrado linear de uso bastante freqüente em sistemas digitais. A estrutura interna do 555, indicada na figura a seguir, é composta por um flip-flop, dois comparadores de tensão, um divisor de tensão resistivo e dois transistores:



Pinagem:

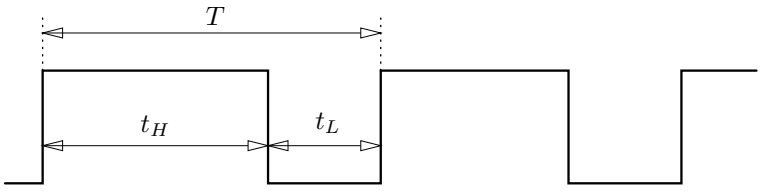
1: terra
2: *trigger*
3: saída
4: *reset*
5: tensão de controle
6: *threshold*
7: descarga
8: V_{cc}

Podemos empregar o circuito 555 para construir um monoestável, como mostra a figura a seguir. A duração de pulso será dada por $t_\omega \approx 1.1RC$ (respeitar $R \geq 1K\Omega$).



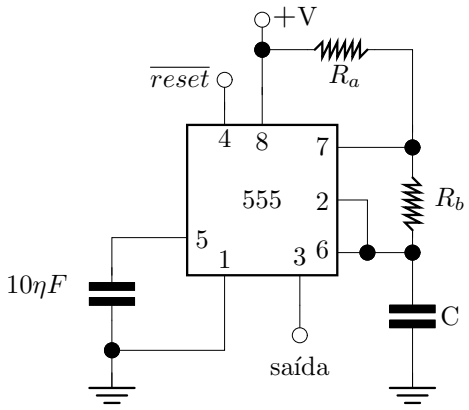
5.1.2 Circuitos astáveis

Os circuitos astáveis não possuem estados estáveis. Sua saída oscila entre dois estados, gerando assim uma onda quadrada que pode ser usada como sinal de *clock* para circuitos seqüenciais. A forma de onda na saída de um astável é definida por dois parâmetros: seu período T, que define também sua *freqüência* $F = T^{-1}$, e seu *ciclo de trabalho* (*Duty Cycle*), que define a parcela do período que o sinal está ativo:



$$D = \frac{t_H}{t_H + t_L} = \frac{t_H}{T}$$

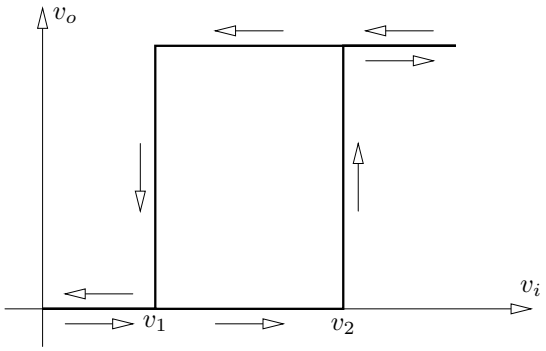
Geralmente a frequência e o ciclo de trabalho de um astável podem ser ajustados através de resistores e capacitores externos. Podemos empregar o CI 555 para construir um astável, como mostra a figura:



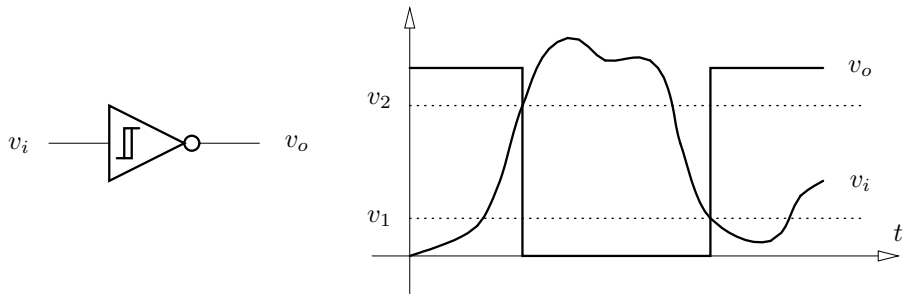
$$f = T^{-1} = \frac{1}{t_H + t_L} \approx \frac{1.44}{(R_a + 2R_b)C} \quad \text{e} \quad D = \frac{t_L}{t_H + t_L} \approx \frac{R_b}{R_a + 2R_b} \quad \text{com} \quad \begin{cases} R_a \geq 1K\Omega \\ C \geq 500pF \end{cases}$$

5.2 Schmitt-Trigger

O circuito *Schmitt Trigger* (disparador de Schmitt) é um dispositivo que apresenta como característica de transferência (relação entre as tensões de entrada e de saída) um ciclo de histerese, como indica a figura:



Em geral essa caracterítica é incorporada a circuitos que implementam funções lógicas básicas, resultando em inversores Schmitt-Trigger, portas NAND e NOR Schmitt-Trigger, etc. Vejamos o comportamento de um inversor ST alimentado por um sinal de entrada variável:

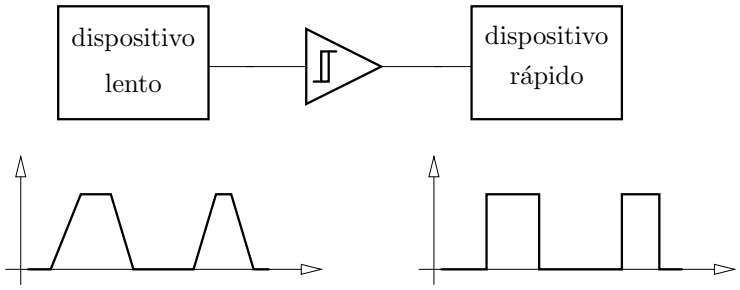


As principais aplicações dos circuitos Schmitt-Triggers são:

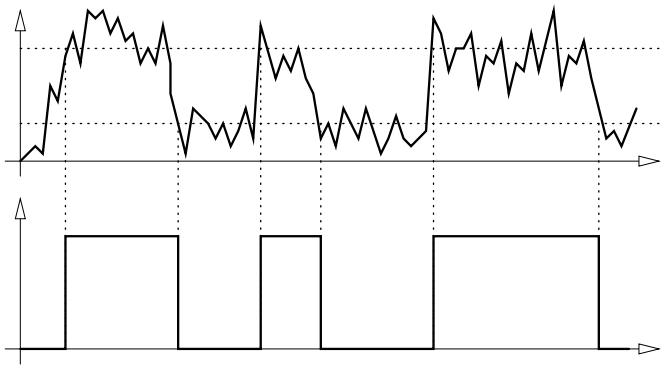
- acoplamento entre dispositivos lentos e rápidos. Quando o sinal de entrada em uma porta lógica varia muito lentamente, podem ocorrer os seguintes problemas:

- Dispositivos sensíveis a tempo de subida podem não operar satisfatoriamente.
- Portas lógicas podem ficar polarizadas na região ativa por muito tempo, gerando instabilidade, consumo excessivo de corrente e aquecimento (as portas lógicas são implementadas por transistores normalmente polarizados nas regiões de corte e saturação).
- Os retardos de propagação (característica física das portas lógicas) tornam-se de difícil previsão.

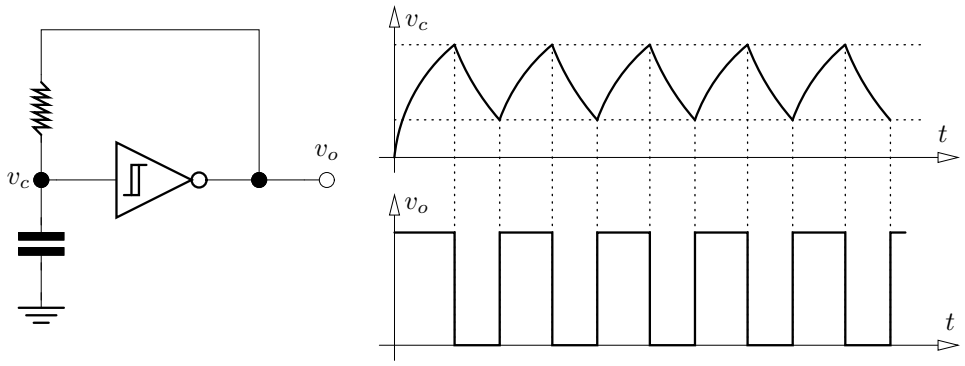
Nessas situações pode ser empregado um circuito Schmitt Trigger para tornar as variações do sinal lento mais abruptas e definidas, e assim acoplar o dispositivo com sinal de saída lento ao dispositivo rápido. Essa estratégia é normalmente empregada para efetuar a conexão entre circuitos da família CMOS (mais lentos) e circuitos da família TTL (mais rápidos), como veremos no capítulo 7.



- Conformador ou limitador de pulsos. Como o Schmitt Trigger suporta sinais de entrada de amplitude variável, ele pode ser usado para "limpar" sinais digitais com ruído excessivo:



- Devido à sua curva de histerese e também por aceitar tensões de entrada variáveis, um inversor Schmitt Trigger pode ser usado para implementar um circuito astável (oscilador de onda quadrada):



CI	f (Hz)
7414	$\approx \frac{0.8}{RC}$, $R \leq 500\Omega$
74LS14	$\approx \frac{0.8}{RC}$, $R \leq 2K\Omega$
74HC14	$\approx \frac{1.2}{RC}$, $R \leq 10M\Omega$

5.3 Exercícios

1. Projetar um circuito astável usando o CI 555, para operar com frequência de 10 KHz e ciclo de trabalho de 30%.

2. Projetar um circuito monoestável com o CI 555, para operar com $t_\omega = 50ms$.

3. Determine a frequência da onda gerada por um oscilador usando inversor Schmitt-trigger implementado usando o CI 74LS14, com $R = 1K\Omega$ e $C = 10\eta F$.

Capítulo 6

Memórias

6.1 Introdução

Normalmente denominamos "memória" todo dispositivo capaz de armazenar informação. Todavia, no contexto desta disciplina, consideraremos apenas os dispositivos semicondutores capazes de armazenar dados, de forma temporária ou permanente.

As memórias semicondutoras são normalmente empregadas na construção de computadores, e por esta razão vamos inicialmente estudar a estrutura básica dessas máquinas.

6.2 Estrutura do computador

O computador é um sistema seqüencial complexo que pode realizar diferentes operações indicadas por meio de instruções. A estrutura básica dos computadores atuais segue o modelo proposto por Von Neumann, composto pelos seguintes elementos:

- **Unidade de Controle** (UC): Seqüencia e supervisiona as tarefas que estão sendo executadas pelos demais elementos do computador. Por exemplo, durante a execução de um programa, ela controla o fluxo de dados entre os dispositivos de entrada/saída, a memória e a unidade lógica e aritmética.
- **Unidade lógica e aritmética** (ULA): Executa operações lógicas e aritméticas com os dados presentes na memória. Em conjunto com a unidade de controle constitui o processador (*Central Processing Unit* - CPU).
- **Memória**: Armazena os programas (instruções) e dados em forma binária. Cada "palavra" (dado ou instrução) está contida em um endereço (posição) distinto na memória.
- **Dispositivos de entrada e saída**: Permitem a ligação do computador com o usuário. teclado, vídeo, *mouse*, etc. Nesta categoria também estão incluídos os dispositivos de armazenamento de massa, como discos rígidos, disquetes, CD-ROMs, etc.

Esses diferentes elementos se comunicam entre si através de três barramentos distintos, aos quais todos estão conectados:

- **Barramento de dados**: permite transferir informações (dados) entre processador, memória e dispositivos de E/S. A largura (número de vias) desse barramento depende do processador usado. As larguras de barramento de dados mais comuns são:

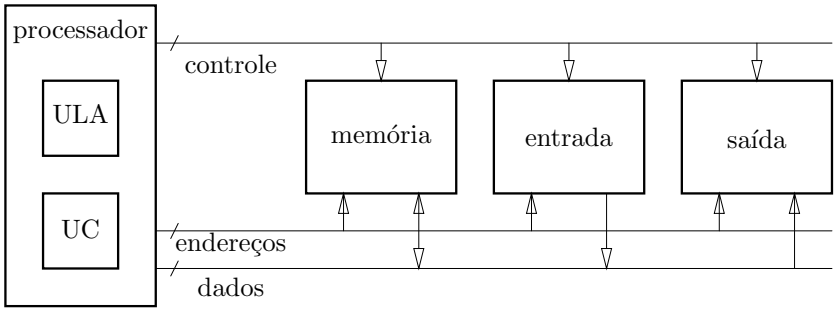
  - 8 bits: 8086, 8088
  - 16 bits: 386-SX, 486-SX
  - 32 bits: 386-DX, 486-DX, Pentium
  - 64 bits: Ultra, Alpha, P6

- **Barramento de endereços**: permite ao processador selecionar o endereço (posição) de memória ou de E/S que deseja acessar. As larguras mais comuns para esse barramento são:

  - 20 bits (2^{20} = 1 Mbyte): 8086, 8088
  - 24 bits (2^{24} = 16 Mbytes): 286
  - 32 bits (2^{32} = 4 Gbytes): 386, 486, Pentium

- **Barramento de controle**: veicula sinais de controle entre o processador e os demais dispositivos, dentre os quais podemos ressaltar:

  - Operação de escrita ou leitura ($R/\overline{W}$): indica se na operação atual o processador deseja ler ou escrever o dado presente do barramento de dados no endereço indicado pelo barramento de endereços.
  - Operação em memória ou dispositivo de E/S ($IO/\overline{M}$): indica se na operação atual o processador deseja acessar um endereço (indicado pelo barramento de endereços) em memória ou em um dispositivo de entrada/saída.
  - Sinalização de interrupções de hardware.



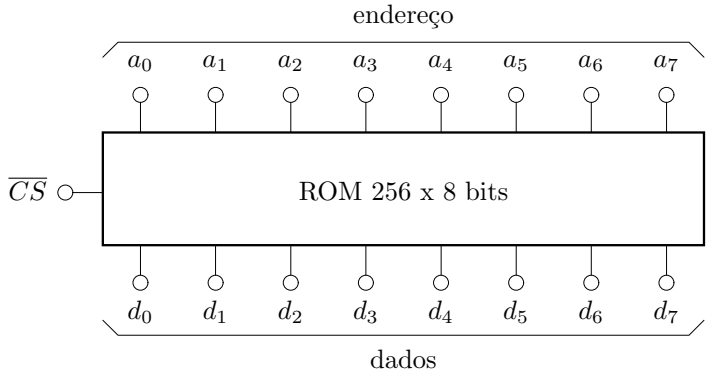
6.3 Memórias ROM

As memórias ROM (*Read-Only Memory*) são dispositivos de armazenamento de dados cujo conteúdo é fixo (não pode ser mudado facilmente) e não-volátil (é mantido mesmo quando o dispositivo é desligado). As memórias ROM são normalmente empregadas para armazenar informações permanentes ou de longa duração, como programas de inicialização (*boot*) de sistemas ou tabelas de dados fixos.

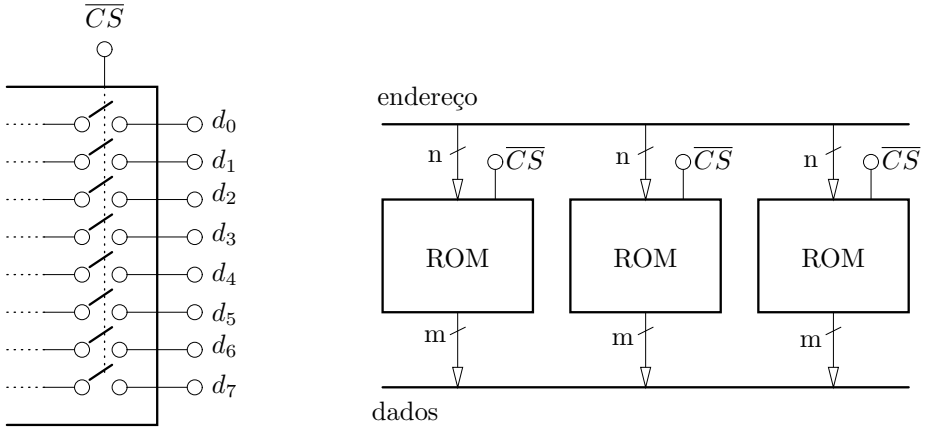
6.3.1 Estrutura básica

Um chip de memória ROM pode ser visto como um circuito combinacional, como os estudados no capítulo 3. Sua entrada é o endereço interno a acessar e sua saída é o valor (fixo) armazenado naquele endereço. A entrada de endereços possui n vias para 2^n endereços distintos (*address*

$a_0 \cdots a_{n-1}$). A saída de um chip de memória pode apresentar m bits. O exemplo abaixo indica uma memória ROM com 256 posições de 8 bits cada, perfazendo um total de 2048 bits armazenados:



No exemplo acima percebemos a existência de uma entrada ainda não apresentada. Trata-se da entrada *Chip Select* ($\overline{CS}$), que controla o estado das saídas do chip. Caso esta entrada seja ativada, os dados do endereço selecionado são colocados na saída da memória. Caso contrário, a saída é desconectada da estrutura interna da memória e mantida em um estado de "alta impedância". Esse mecanismo é necessário porque todas as saídas de memórias e dispositivos de E/S são conectadas juntas ao barramento de dados do computador. As entradas *chip select* das memórias do sistema devem ser adequadamente controladas para garantir que em um determinado instante apenas o conteúdo de uma posição de memória será colocado no barramento de dados, sem a possibilidade de conflitos. A figura a seguir ilustra esse mecanismo:



6.3.2 Tecnologias

Existem atualmente diversas tecnologias para a construção de memórias ROM, dentre as quais podemos ressaltar:

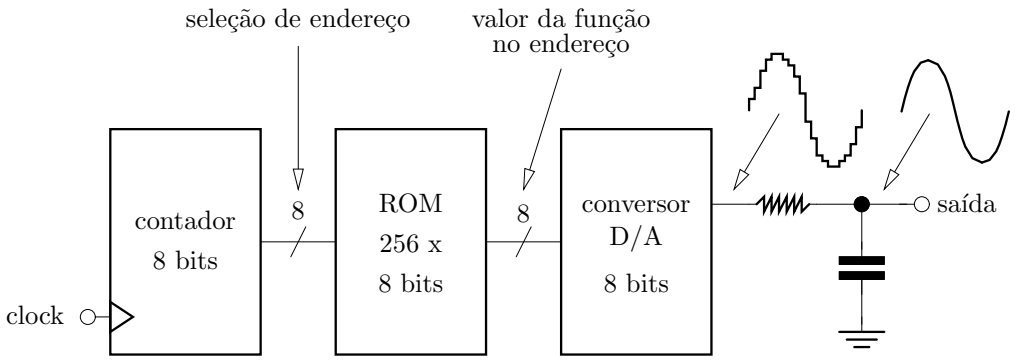
- **Matriz de diodos**: os dados são armazenados durante a fabricação do circuito, e não podem mais ser modificados. É uma tecnologia rápida e barata, usada na produção de sistemas em escala industrial.
- **PROM** (*Programable ROM*): pode ser programada uma única vez pelo usuário, através da queima de micro-fusíveis no interior do chip.

- **EPROM** (*Erasable PROM*): muito usada em prototipagem, pode ser programada diversas vezes através da aplicação de tensão elevada em pontos especiais; seu apagamento (completo) se dá através de um banho prolongado de raios ultra-violeta.
- **EEPROM** (*Electrically Erasable PROM*): pode ser escrita e apagada (lentamente) através de sinais elétricos. É bastante usada para armazenar configurações de hardware em computadores (BIOS dos IBM-PC).
- **FLASH**: pode ser gravada e apagada por sinais elétricos. É mais rápida e permite níveis de integração bem maiores que as EEPROM. Está pouco a pouco tomando presença no mercado, devido ao seu preço.

6.3.3 Aplicações

O principal campo de aplicação das memórias ROM encontra-se no armazenamento permanente de valores binários. Nesse campo podemos ressaltar as seguintes aplicações:

- **Firmware**: guardar programas residentes em sistemas usando micro-processadores, como equipamentos de vídeo, eletrodomésticos, brinquedos, controladores industriais, etc.
- **Bootstrap**: memória de inicialização de computadores, contendo as instruções iniciais a executar para lançar o equipamento (testes de hardware, mensagens, carga do sistema operacional a partir de um disco, etc.).
- **Tabelas de dados**: armazenamento de dados fixos como tabelas de senos, cossenos, etc. Com isso pode-se ganhar um tempo considerável evitando cálculos complexos e freqüentes.
- **Conversão de dados**: podemos usar as memórias ROM para converter dados entre diferentes formatos binários, considerando o endereço como dado de entrada e o conteúdo da memória naquele endereço como dado de saída. Por exemplo, podemos armazenar em uma ROM de 256×8 bits o valor do complemento 2 para cada endereço. Selecionando um determinado endereço teremos na saída o respectivo valor complementado.
- **Geração de funções**: podemos produzir formas de onda específicas armazenando os valores correspondentes a um período da função em uma memória ROM. Esta pode então ser percorrida seqüencialmente para reconstituir a forma de onda, de acordo com o circuito a seguir:



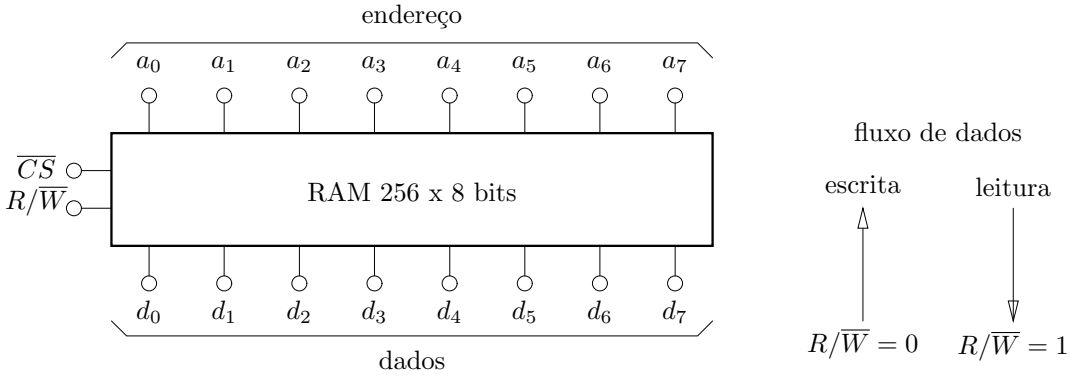
- **Armazenamento de configurações**: como usado para armazenar as configurações de BIOS dos computadores compatíveis IBM-PC (EEPROM).

6.4 Memórias RAM

As memórias RAM (*Random Access Memories*) permitem armazenar dados e modificá-los continuamente, servindo assim como área de trabalho para a execução de programas (variáveis, dados temporários, pilha de execução, etc). O conteúdo de uma RAM é normalmente volátil, ou seja, desaparece se o circuito for desligado.

6.4.1 Estrutura básica

O modelo básico de memória RAM segue o definido para as memórias ROM, com a inclusão de vias para a entrada de dados (escrita na memória) e de uma entrada para indicar se desejamos efetuar uma leitura ou uma escrita no endereço indicado (entrada $R/\overline{W}$). Para economizar pinos nos chips, as entradas e saídas de dados são feitas através dos mesmos pinos, cuja função é então definida pelo valor da entrada $R/\overline{W}$:



6.4.2 Tecnologias

A forma como os dados binários são armazenados no interior de uma RAM permite classificá-las em dois grupos:

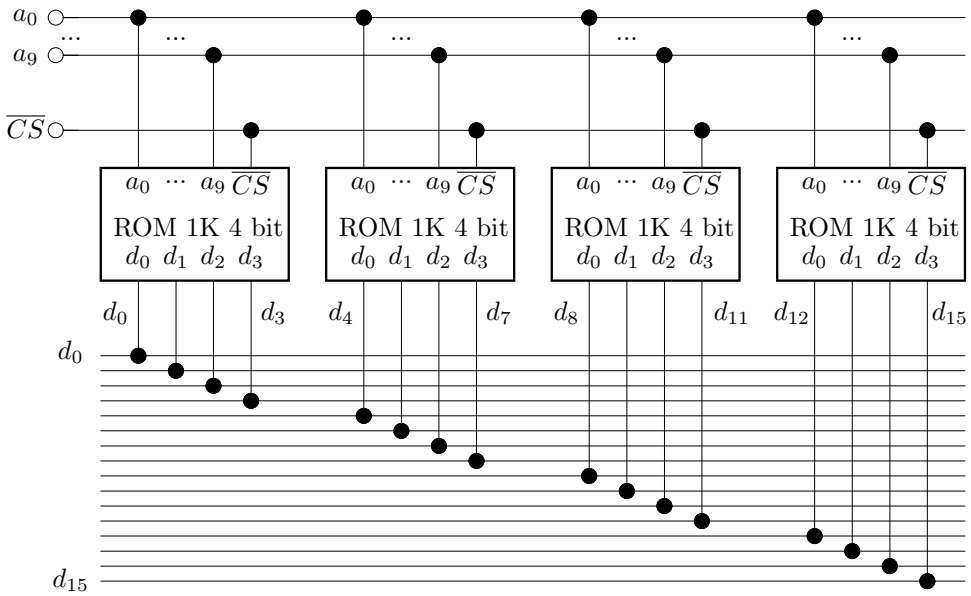
- **RAM estáticas** (SRAM): a célula básica de armazenamento da informação binária é um flip-flop, que armazena 1 bit; ele permanece nesse estado até ser explicitamente alterado.
- **RAM dinâmicas** (DRAM): a célula básica de armazenamento da informação binária é um capacitor que armazena 1 bit; ele precisa ser periodicamente recarregado (*refresh*) para permanecer em 1, caso armazene um bit ativo.

As RAM estáticas são mais rápidas e simples de construir que as dinâmicas, pois não precisam de circuitos auxiliares para percorrer toda a memória e efetuar o *refresh* dos capacitores carregados (bits ativos). Todavia as RAM dinâmicas são bem mais baratas e muito mais compactas, além de consumirem menos energia. Desta forma, as memórias RAM estáticas são empregadas em sistemas precisando de pouca memória mas com alta velocidade, como micro-controladores, processamento de sinais em tempo-real, memórias de cache e de vídeo, etc. As RAM dinâmicas são normalmente empregadas quando prioriza-se o volume de memória e o baixo consumo, como no caso da memória principal dos computadores pessoais.

6.5 Bancos de memória

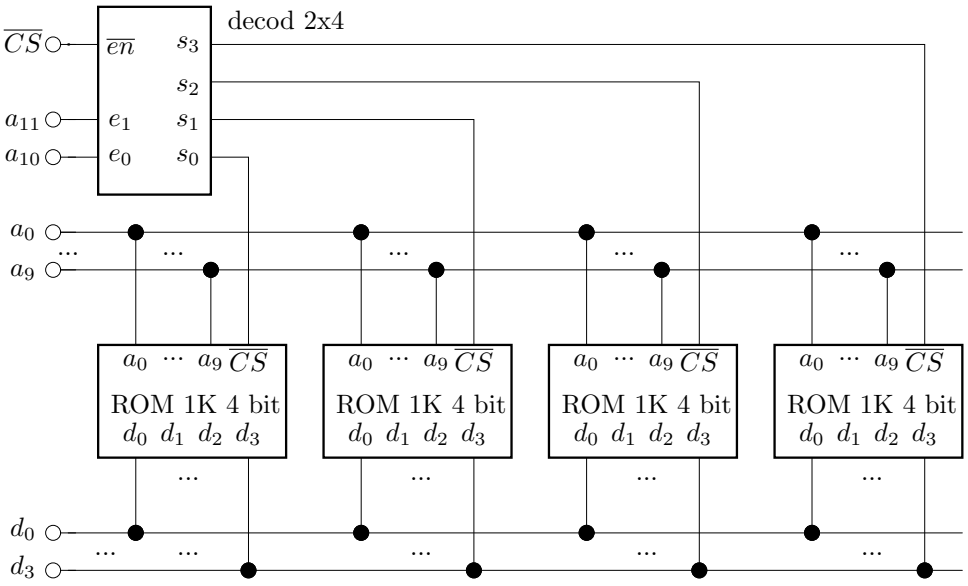
Um sistema de memória de computador, também denominado *banco de memória*, normalmente usa um barramento de dados com largura de 8, 16, 32 ou 64 bits (também chamados de *tamanho da palavra de memória*). Por exemplo, o barramento de dados de um processador Pentium tem largura de 32 bits. Como os chips de memória atuais possuem uma capacidade que vai de 1K (2^{10}) a 1M (2^{20}) palavras de 1, 4 ou 8 bits, nem sempre dispomos de chips de memória com a largura de palavra necessária para um determinado barramento. Além disso, muitas vezes a exigência em termos de espaço de endereçamento (capacidade de armazenamento) também são maiores que a capacidade de um só chip.

Para construir bancos de memória com a largura de palavra e o espaço de endereçamento necessários devemos então associar chips entre si. Para obtermos palavras de memória com a largura desejada, devemos associar chips de memória em paralelo. Por exemplo, se dispusermos de chips de 1K posições de 4 bits e desejamos construir um banco de memória com palavras de largura de dados de 16 bits, devemos associar quatro chips em paralelo, como mostra a figura:



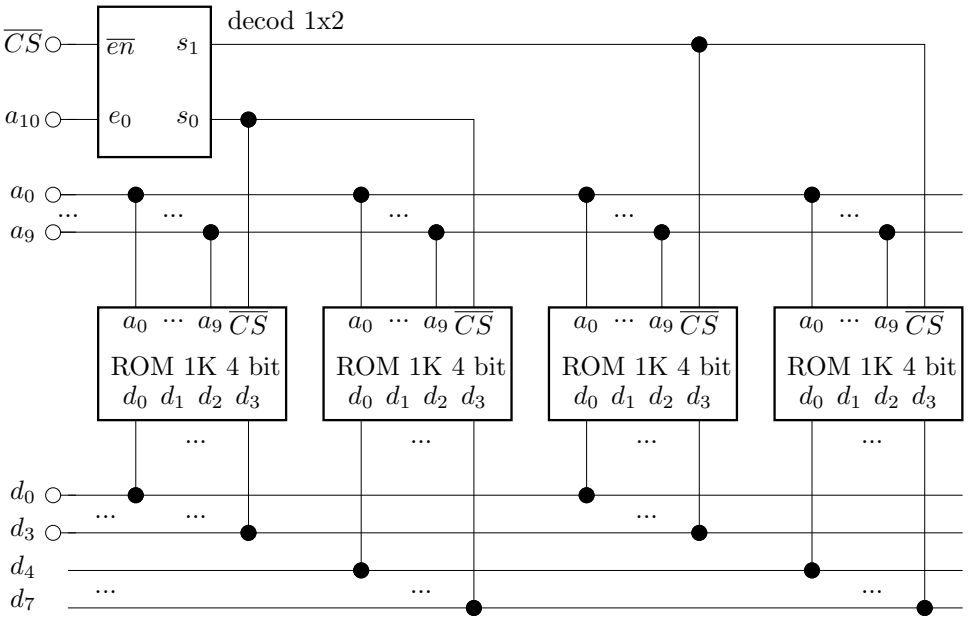
Com indicado na figura, cada chip de memória fica responsável pela armazenamento de uma parcela (4 bits) de cada palavra de 16 bits. As 10 vias do barramento de endereços (2^{10} posições em cada chip) são conectadas em paralelo, e também o sinal *chip select*. A capacidade total do banco de memória acima resulta então em 1K posições de 16 bits.

Para construir bancos de memória com espaço de endereçamento maior que a capacidade de cada chip, devemos associar os chips de forma diferente: cada chip será responsável por armazenar uma parcela do espaço de endereçamento. Assim, na construção de um banco de memória de 4K posições de 4 bits usando chips de 1K posições de 4 bits, devemos empregar 4 chips, sendo cada um deles responsável por uma parcela de 1K x 4 bits. Devemos usar as vias de endereçamento mais elevadas para controlar as entradas *chip select* de cada chip e assim selecionar qual deles será ativado em um determinado endereço. Normalmente empregamos um decodificador para essa função, como mostra a figura a seguir:



Como mostra o exemplo acima, normalmente as vias menos significativas (LSB) do barramento de endereços são ligadas diretamente aos chips de memória, para selecionar as células de memória dentro dos chips. As vias de endereços mais elevadas (MSB) são empregadas para selecionar qual chip estará ativo em um determinado endereço, através de um decodificador. Para selecionar um endereço no conjunto de chips o decodificador deve estar habilitado (*enabled*), o que é feito através de sua entrada $\overline{en}$.

Podemos combinar ambas as técnicas para construir bancos de memória com a largura de dados e a capacidade de armazenamento desejadas. O exemplo a seguir mostra a construção de um banco de 2K x 8 bits a partir de chips de 1K x 4 bits:



No caso de bancos de memórias RAM, as técnicas empregadas são exatamente as mesmas. Além disso, as entradas $R/\overline{W}$ de todos os chips de memória devem ser conectadas à via $R/\overline{W}$ do barramento de controle. Estas técnicas também podem ser usadas para a construção de bancos compostos de RAM e ROM.

6.6 Exercícios

1. Apresente a estrutura básica de um computador, descrevendo as características de seus principais componentes e barramentos.

2. Determine o número de linhas de endereços e de dados necessárias para acessar as seguintes memórias (expressas no formato *posições* × *bits*):

512 × 8	1K × 1	512 × 4
4K × 1	256 × 4	8M × 32

3. Comente as diferenças entre as memórias de tipo RAM, ROM, PROM, EPROM e EEPROM.

4. Comente as diferenças entre memórias RAM estáticas e dinâmicas.

5. Para que serve a entrada "Chip Select" ($\overline{CS}$) em um chip de memória ? Qual a sua influência sobre as portas de saída do chip ?

6. Projete um banco de memória ROM de 4K × 8 bis a partir de memórias comerciais ROM de 4K × 1 bit.

7. Projete um banco de memória ROM de 4K × 8 bis a partir de memórias comerciais ROM de 1K × 8 bits.

8. Projete um banco de memória RAM de 8K células de 8 bits, usando chips comerciais de 4K×4 bits. Não esqueça de indicar os sinais do barramento de controle.

9. Por que, ao construir um banco de memórias de 4K × 8 bits usando chips de 2K × 8 bits, podemos curto-circuitar as linhas de dados sem causar problemas ?

10. Um telefone celular possui um micro-controlador de 8 bits que coordena seu funcionamento. Para funcionar, este precisa de 8 Kb de ROM para seus programas residentes e dados fixos, e 8Kb de RAM como área temporária. Projete o banco de memória necessário, usando chips comerciais ROM de 4k x 8 bits e RAM de 8k x 4 bits. A memória ROM deve estar no início do espaço de endereçamento.

11. Uma agenda eletrônica possui um micro-controlador de 8 bits que coordena seu funcionamento. Para operar, este precisa de 48 Kbytes de ROM para seus programas residentes e 16 Kbytes de RAM como área de trabalho. Projete o banco de memória necessário, usando chips comerciais ROM de 16k x 8 bits e RAM de 16k x 2 bits. A memória ROM deve estar no início do espaço de endereçamento. Não esqueça de indicar os sinais $\overline{CS}$ e $R/\overline{W}$.

12. Projete as áreas de memória RAM, ROM e de I/O (área reservada para dispositivos de entrada/saída, mapeada em memória) de um pequeno computador experimental. A CPU possui 14 linhas de endereços e 8 linhas de dados (8 bits = 1 byte). A memória ROM deve ter 2 Kbytes e deve ser mapeada no final da memória. A memória RAM deve ter 8 Kbytes e deve estar no início do mapa de memória. A área reservada para I/O deve ter 2 Kbytes e situar-se a partir do endereço 2400_H. Devem ser usados chips ROM de 1K × 4 bits e RAM de 2 K × 8 bits. Indique os endereços inicial e final de cada bloco (ROM, RAM, I/O) no mapa de memória e construa o diagrama do circuito.

Capítulo 7

Famílias Lógicas

7.1 Introdução

Os dispositivos digitais podem ser classificados por sua complexidade, de acordo com o número de portas lógicas básicas necessárias para sua implementação. A classificação vigente usa cinco níveis:

- SSI: até 12 portas por chip
- MSI: até 100 portas por chip
- LSI: até 10.000 portas por chip
- VLSI: até 100.000 portas por chip
- ULSI: acima de 100.000 portas em um chip

Os circuitos integrados digitais também se diferenciam entre si pelo tipo de dispositivo semicondutor empregado em sua fabricação. Assim temos, entre outros, CIs fabricados com tecnologia MOS – *Metal-Oxide Semiconductor* (famílias NMOS, CMOS, etc) ou com tecnologia bipolar (famílias I2L, RTL, TTL, ECL). Dentro de uma mesma família, podem existir várias séries, que se diferenciam por parâmetros como velocidade de operação e consumo de energia, devido a diferenças em suas estruturas internas e nas tecnologias empregadas para sua fabricação.

Por serem as duas famílias mais populares, concentraremos nossa atenção neste capítulo às famílias CMOS e TTL.

7.2 Tecnologias de Fabricação

A tecnologia bipolar baseia-se na associação de transistores convencionais de junção PNP e NPN. Resultam desta dispositivos rápidos mas com um consumo de corrente significativo. Como cada transistor bipolar ocupa uma área significativa na pastilha semicondutora, o uso desta tecnologia limita-se a dispositivos SSI e MSI, ou seja, na construção de sistemas lógicos simples ou na interligação de componentes mais complexos. As famílias TTL e ECL são os resultados mais difundidos desta tecnologia.

Os dispositivos de tecnologia MOS (*Metal-Oxide Semiconductor*) são constituídos quase que exclusivamente de transistores de efeito de campo, que contribuem para um consumo de corrente extremamente baixo. Entretanto esses dispositivos são normalmente mais lentos que seus

equivalentes bipolares TTL e ECL. A maior vantagem da tecnologia MOS é a alta capacidade de integração: dispositivos MOS podem ocupar menos de 5% da área de seus equivalentes bipolares. Isto faz desta tecnologia a mais usada na implementação de circuitos VLSI como processadores, grandes memórias, etc, através das famílias NMOS e PMOS.

7.3 Parâmetros de Circuitos Integrados

Tensão de alimentação : é a tensão que precisamos aplicar para energizar o circuito. Certas tecnologias podem requerer uma tensão fixa, enquanto outras podem operar em uma faixa larga de tensões. Por exemplo, os CIs TTL da série 74XX devem operar entre 4.75 e 5.25 V, enquanto algumas séries CMOS podem receber uma tensão de alimentação entre 3 e 15V.

Consumo : define a potência requerida por uma porta para funcionar em um regime de trabalho médio (metade do tempo em nível alto e metade em nível baixo). A potência consumida é geralmente da ordem de mW, sendo convertida em calor no interior de circuito. Uma elevada dissipação de potência pode implicar em maiores custos para a fabricação e operação dos circuitos, devido às maiores necessidades em termos de fontes de alimentação, ventilação, vida útil, etc. Além disso, a necessidade de dissipar o calor gerado pode comprometer o grau de integração possível de obter com uma determinada tecnologia.

Velocidade : Um circuito requer um determinado tempo para mudar seu estado lógico. Este tempo, da ordem de ηS, é conhecido como *atraso de propagação* e define a velocidade máxima de operação do circuito. Existe uma relação direta entre a frequência de operação e a potência dissipada por porta.

Tensões de nível lógico : São os níveis de tensão correspondentes aos valores lógicos 0 e 1, e que devem ser respeitados para o funcionamento correto do circuito. São normalmente definidos quatro níveis de tensão:

V_{IH} : tensão de entrada em nível alto (1).

V_{IL} : tensão de entrada em nível baixo (0).

V_{OH} : tensão de saída em nível alto (1).

V_{OL} : tensão de saída em nível alto (0).

Vejamos os valores de tensão definidos para as famílias TTL 74XX e CMOS 4XXX, considerando uma tensão de alimentação de +5V:

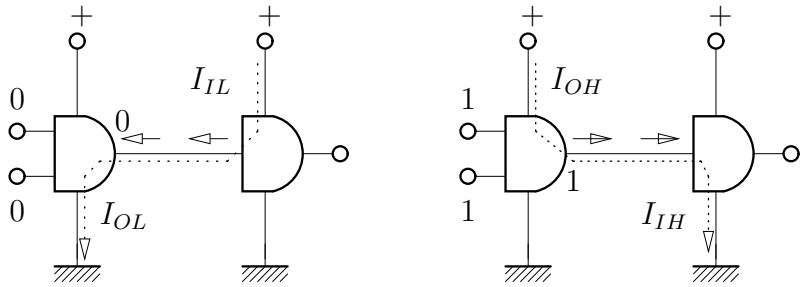
Família	tensão	mínimo	máximo
TTL	V_{IL}	–	0.8
	V_{IH}	2.0	–
	V_{OL}	–	0.4
	V_{OH}	2.4	–
CMOS	V_{IL}	–	1.5
	V_{IH}	3.5	–
	V_{OL}	–	0.1
	V_{OH}	4.9	–

Margem de ruído : A margem de ruído define a variação possível entre o sinal de saída de uma porta e o sinal de entrada de outra de mesma tecnologia. Podemos definir duas margens de ruído, para os níveis alto e baixo, definidas por:

$$\begin{aligned} V_{NH} &= V_{OH_{min}} - V_{IH_{min}} \\ V_{NL} &= V_{IL_{max}} - V_{OL_{max}} \end{aligned}$$

Assim, para a série TTL 74XX teremos $V_{NH} = V_{NL} = 0.4V$, e o funcionamento do circuito é então garantido para um nível de ruído induzido que não exceda 0.4V.

Fator de carga : o fator de carga, também chamado *fan-out*, indica a capacidade de carga suportada por uma saída, sem violação das tensões de nível lógico. Por exemplo, uma porta com *fan-out* de 10 é capaz de alimentar até 10 entradas de portas de mesma tecnologia. O valor de *fan-out* está diretamente relacionado às correntes fornecidas ou drenadas pelas entradas e saídas das portas lógicas. Em uma conexão lógica podemos definir quatro correntes máximas, que nos permitirão determinar o fator de carga:



I_{IL} : corrente máxima de entrada baixa.

I_{IH} : corrente máxima de entrada alta.

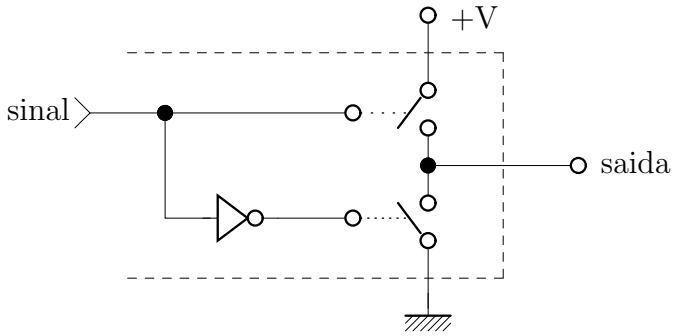
I_{OL} : corrente máxima de saída baixa.

I_{OH} : corrente máxima de saída alta.

A série TTL 74XX possui $I_{IL} = 1.6mA$, $I_{OL} = 16mA$, $I_{IH} = 40\mu A$, $I_{OH} = 400\mu A$, o que nos dá um *fan-out* de 10 em ambos os níveis lógicos. Como as portas operam nos dois níveis, devemos sempre considerar o menor valor de *fan-out* como limite.

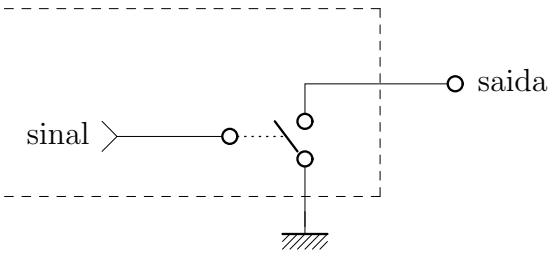
7.4 Estrutura das saídas

As portas de saída de circuitos digitais devem normalmente apresentar um nível lógico bem definido. Entretanto em algumas situações é interessante construir saídas com comportamentos especiais, como vimos no caso das saídas em 3º estado nos circuitos de acesso a barramentos de computadores. A estrutura de saída mais usual é a que apresenta sempre um nível lógico definido na saída, e que pode ser grosseiramente representada pelo diagrama abaixo (no qual representamos os transistores por chaves):

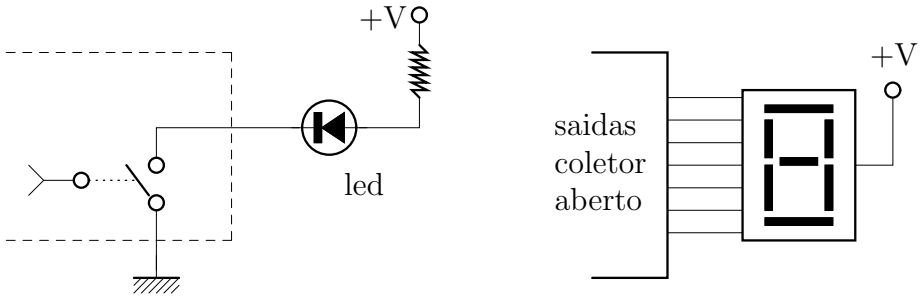


Nos dispositivos TTL essa saída é chamada *totem-pole*; esse tipo de saída também é bastante usual em dispositivos CMOS. Devido aos dois transistores atuarem de forma complementar, saídas deste tipo nunca podem ser interligadas, sob o risco de curto-circuitos.

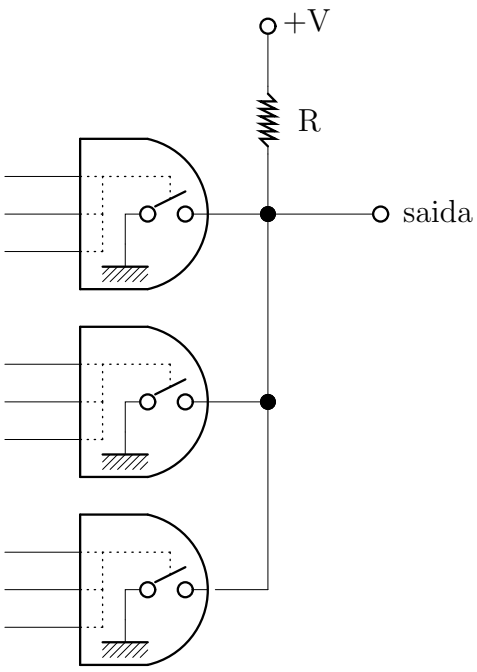
Outra estrutura de saída bastante conhecida é a estrutura em *coletor aberto* (ou *dreno aberto*, para os dispositivos CMOS). Nesta estrutura um dos transistores é eliminado:



Este tipo de saída é bastante útil para o acionamento de dispositivos de saída como *leds* ou *displays*, como mostra o diagrama abaixo:

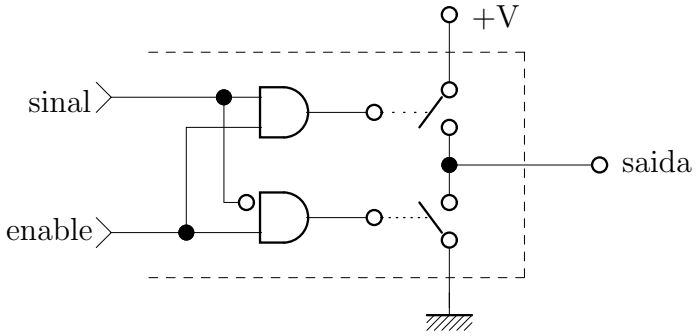


Outra aplicação importante das saídas em coletor aberto é a construção de funções lógicas usando uma estrutura chamada *Wired AND* ("E por fio"). Nesta estrutura, construimos artificialmente uma porta AND interligando as saídas em coletor aberto; um resistor é usado para gerar o nível lógico alto quando todas as saídas estiverem desligadas. Caso uma saída em coletor aberto seja acionada, ela aterra a conexão e o sinal de saída da estrutura *wired AND* desce a zero:



Este tipo de conexão permite uma redução significativa de componentes e do tempo de propagação do circuito. O resistor elevador de nível R deve ser calculado em função das tensões de nível lógico e das correntes de saída das portas.

Um terceiro tipo de estrutura de saída é a de terceiro estado, bastante empregada em circuitos de memória e acesso a barramentos. Nesta estrutura, a saída pode ser completamente desligada dos níveis lógicos, ficando então "flutuante". O diagrama abaixo ilustra seu funcionamento:



Saídas deste tipo podem ser interligadas entre si, mas apenas uma pode estar habilitada em um determinado instante, para evitar curto-circuitos.

7.5 A Família TTL

A família TTL (*Transistor-Transistor Logic*) é bastante popular e ainda muito usada nas funções lógicas simples, permitindo interligar componentes lógicos mais complexos. Os circuitos TTL são construídos usando a tecnologia bipolar, através do acoplamento direto entre transistores, o que permite atingir altas velocidades de comutação, embora o consumo de corrente seja um pouco acentuado. De acordo com suas características de tensão de alimentação e faixa de temperaturas de trabalho, esta família está dividida em dois grupos: a linha 74, para uso geral, e a linha 54, para situações críticas, sobretudo em aplicações militares e espaciais:

Grupo	alimentação	temperatura
54XX	4.5 a 5.5 V	-55 a $+125^o$ C
74XX	4.75 a 5.25 V	0 a $+70^o$ C

Os circuitos TTL standard são raramente empregados hoje em dia, pois novas tecnologias permitiram a construção de dispositivos TTL com características melhoradas, sobretudo no que se refere a potência dissipada e velocidade de comutação. As principais séries TTL são:

Série L (Low Power) : foi desenvolvida buscando componentes de baixo consumo ($\sim 1mW$), às custas de maiores tempos de propagação ($\sim 35\eta S$).

Série H (High Speed) : esta série propõe maior velocidade ($\sim 6\eta S$), impondo para tal um maior consumo de potência ($\sim 25mW$).

Série S (Schottky) : substituiu a série H, pois permite menores tempos de propagação ($\sim 3\eta S$), com um consumo de potência semelhante.

Série LS (Low Power Schottky) : é uma versão de baixa potência da série S, que tornou-se quase universal, por possuir os mesmos tempos de propagação da série standard ($\sim 9\eta S$) com um consumo bastante inferior ($\sim 2mW$). Ainda é muito usada, mas está perdendo terreno para suas equivalentes CMOS 74HC e 74HCT.

Séries AS e ALS (Advanced S/LS) : séries recentes, ainda de alto custo, mas com significativas melhoras em velocidade, consumo e *fan-out* em relação às suas equivalentes S e LS.

Série F (Fast) : recente, emprega uma técnica de fabricação que busca minimizar as capacitâncias internas entre dispositivos, para melhorar a velocidade de comutação. Ainda pouco empregada devido ao seu custo.

Quadro comparativo entre as principais séries TTL:

característica	74	74S	74LS	74AS	74ALS	74F
Tempo de propagação (ηS)	9	3	9.5	1.7	4	3
Frequência máxima (MHz)	35	125	45	200	70	100
Consumo de potência (mW)	10	20	2	8	1.2	6
Fan-out	10	20	20	40	20	33

7.6 A Família ECL

A família ECL (*Emitter-Coupled Logic*) também é construída usando a tecnologia bipolar, mas sua estrutura procura resolver diversos problemas inerentes à estrutura TTL, entre eles a saturação dos transistores, o que resulta em uma velocidade de comutação bastante melhorada. Suas principais características são:

- Alta velocidade de comutação, permitindo trabalhar em frequências de até 600 MHz.
- Baixa margem de ruído (250 mV), o que limita sua aplicação em aplicações industriais.
- *Fan-out* médio, por volta de 25.

- Alta dissipação de energia (25 mW).
- Pouca diferença de corrente entre os níveis, gerando pouco ruído interno de comutação.

O uso desta família é restrito a aplicações específicas, onde são necessárias altas velocidades de comutação, como sistemas de tramento digital de sinais, etc.

7.7 A Família CMOS

A família CMOS (*Complementary MOS*) é a que apresenta o menor consumo e a maior velocidade de operação entre as famílias de tecnologia MOS. Entretanto sua complexidade interna impede seu uso em LSI e VLSI, limitando sua aplicação a dispositivos de baixa complexidade, competindo assim com a família bipolar TTL. As principais séries da família CMOS são:

4XXX/14XXX : é a série mais antiga; são dispositivos de baixíssimo consumo e podem operar com uma larga gama de tensões (3 a 15V), mas são muito lentos e suas saídas tem baixa capacidade de corrente. Estão sendo gradualmente substituídos pelas novas séries.

74C : tem as mesmas características elétricas das da série 4XXX. Sua vantagem é a equivalência funcional e de pinagem com a família TTL. Assim, um chip 74C07 possui a mesma função lógica e a mesma pinagem que um 7407 (embora suas características elétricas possam ser incompatíveis).

74HC/HCT (High Speed CMOS) : são as séries CMOS mais populares, possuem velocidade comparável à série TTL 74LS. Além de possuir a mesma funcionalidade e pinagem das séries TTL, a série HCT é eletricamente compatível com a TTL, podendo substituí-la.

74AC/ACT (Advanced CMOS) : série recente, com diversas vantagens mas custo elevado. A pinagem dos chips foi distribuída de modo a minimizar a influência do ruído de comutação, por isso eles não são compatíveis em pinagem com as demais séries.

BiCMOS : busca combinar o melhor das tecnologias CMOS e bipolar, produzindo chips de altíssima velocidade e baixo consumo. Por seu custo, são empregados somente em aplicações específicas, como interfaceamento de computadores.

Eis um quadro comparativo entre as principais séries CMOS, incluindo a série TTL 74LS para comparação:

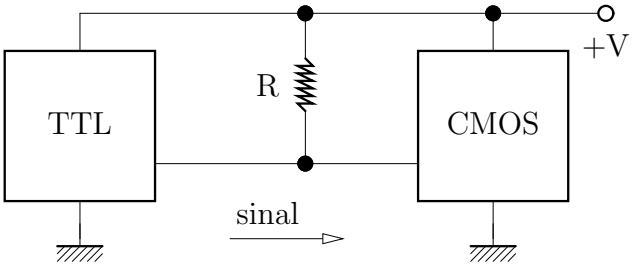
característica	74HC/HCT	74AC/ACT	4000B	74LS
Tempo de propagação (ηS)	8	4.7	50	9.5
Frequência máxima (MHz)	40	100	12	45
Consumo de potência (mW)	0.17	0.08	0.1	2
Margem de ruído (V)	0.9	0.7	1.5	0.4

É importante observar que as entradas de dispositivos CMOS possuem um dreno de corrente extremamente baixo ($\sim 1\eta A$), e por isso são muito sensíveis: a eletricidade estática de um toque de dedo pode acionar uma porta, ou mesmo queimá-la. Por isso, entradas sem uso devem *sempre* ser conectadas a um nível lógico.

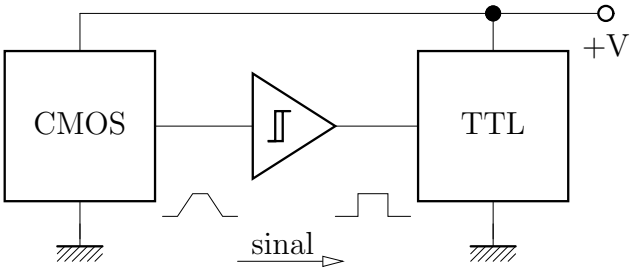
7.8 Compatibilidade entre TTL e CMOS

O acoplamento entre dispositivos CMOS e TTL padrão não pode ser feito diretamente, devido às diferenças entre as tensões de nível lógico e as correntes de entrada e de saída das duas famílias, mesmo sendo providas as mesmas tensões de alimentação. Além disso, a diferença de velocidades de comutação pode causar problemas.

Para conectar uma saída TTL a uma entrada CMOS precisamos elevar a tensão de nível lógico alto da saída, que é muito baixa no padrão TTL (3.5 V). Isto pode ser obtido através de um resistor de *pull-up*, como indica o diagrama:



Para acoplar uma saída CMOS a uma entrada TTL precisamos aumentar a capacidade de corrente da saída, o que pode ser feito usando um *buffer*. Além disso, no caso de dispositivos CMOS lentos, é necessário aumentar a velocidade de comutação do sinal da saída, através de um *schmitt-trigger*:



As novas séries de dispositivos TTL e CMOS permitem uma maior flexibilidade de interconexão, devido à melhoria nas características elétricas das duas famílias, sobretudo na família CMOS (maior velocidade de comutação e maior capacidade de corrente nas saídas). As regras acima continuam no entanto válidas como caso geral: antes de interligar circuitos digitais de famílias distintas, é necessário certificar-se de que as tensões de nível lógico, a velocidade de operação e as correntes envolvidas nas entradas e saídas são compatíveis entre si.

7.9 Exercícios

1. Quais as principais famílias de circuitos integrados usados em sistemas digitais ? Compare-as em termos de velocidade de operação e consumo.

2. Descreva as características dos três principais tipos de saídas empregados em dispositivos digitais.

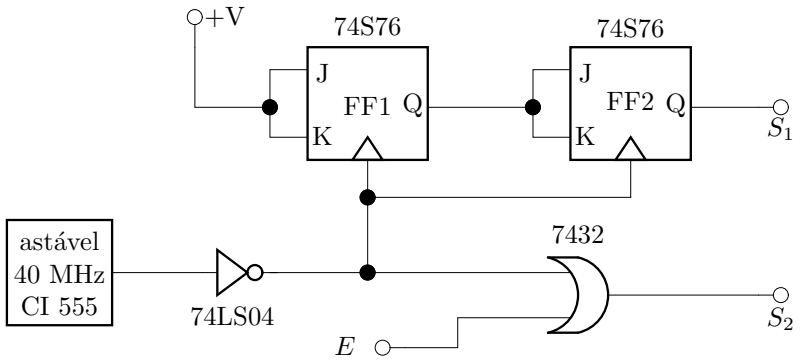
3. Quais os principais problemas que podem ocorrer no uso conjunto de dispositivos TTL e CMOS, e quais as técnicas para contorná-los ?

4. Duas famílias lógicas diferentes têm os seguintes valores para as tensões limites de entrada e saída nos dois níveis lógicos:

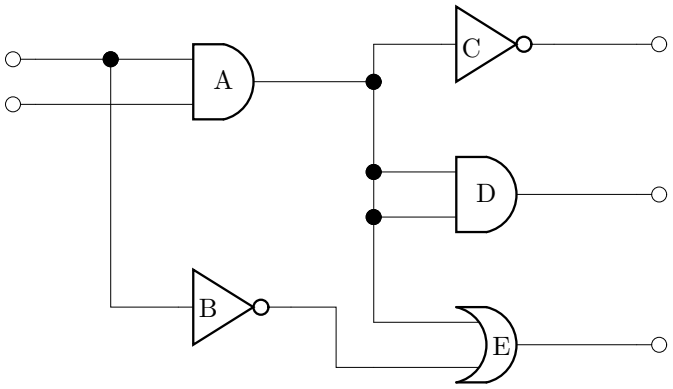
Tensões	Família A	Família B
V_{IL}	0.9	0.7
V_{IH}	1.6	1.8
V_{OL}	0.4	0.3
V_{OH}	2.2	2.5

   Indique qual das duas famílias tem a maior margem de ruído, justificando. Como o ruído induzido pode perturbar o funcionamento de circuitos digitais ?

5. Analise o circuito da figura abaixo, em relação às freqüências máximas de operação e às capacidades de carga das portas de entrada e de saída.



6. Idem, considerando: a) A=L, B=LS, C=D=E=padrão; b) A=LS, B=padrão, C=H, D=E=S.



7. Como um circuito integrado da família TTL se comporta em relação a entradas desconectadas (flutuantes) ? E um circuito integrado CMOS ?

Referências Bibliográficas