Técnicas Digitais A

Departamento de Engenharia de Telecomunicações

Escola de Engenharia

Universidade Federal Fluminense

Prof. João Marcos Meirelles da Silva
jmarcos@vm.uff.br
Sala D425 – www.professores.uff.br/jmarcos

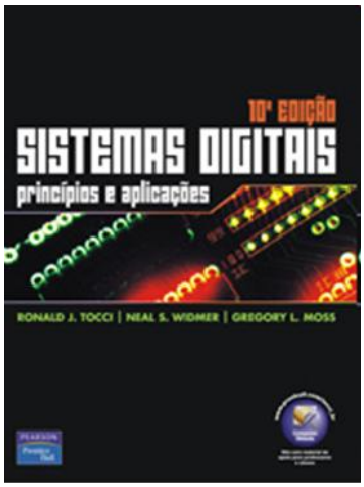
ÍNDICE

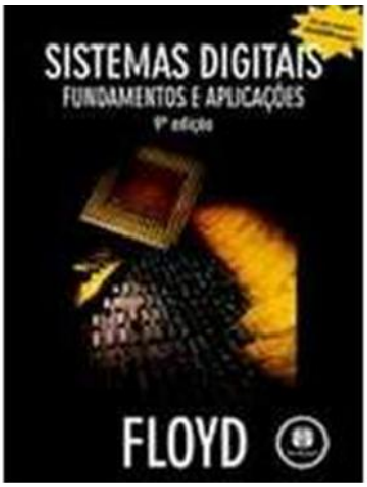
Introdução

Objetivo da Disciplina

Oferecer uma visão dos princípios básicos fundamentais dos sistemas digitais e tratar de modo abrangente os métodos tradicionais e modernos de análise, projeto e implementação.

Bibliografia





Avaliação

A avaliação do curso se dará da seguinte forma:

1. 2 provas (P1 e P2)
2. Média das notas de laboratório (L)

$$M = \frac{(P1 + P2 + L)}{3}$$

OBS: Não haverá prova de reposição. A prova de Verificação Suplementar fará o papel de prova de reposição.

Laboratório

Cada aula de laboratório deverá:

1. Ser precedida de um preparatório → O monitor dará um visto no preparatório antes da aula começar. Ao final da aula, o grupo deverá entregar o preparatório ao monitor (com exceção da 1a aula).
2. Ser concluída com um relatório → O grupo deverá preparar um relatório sobre a experiência, que por sua vez, deverá ser entregue ao monitor no início da próxima aula de laboratório.

OBS: *O preparatório corresponderá a 50% da nota do relatório.*

REGRAS:

- 12 aulas de laboratório;
- Grupos permanentes de até 4 alunos;
- Cada aula de laboratório perdida (falta) deverá ser reposta em dia e horário a ser definido junto com o monitor;
- Caso o aluno não faça a reposição, será lançada nota zero para o respectivo relatório;
- Será permitido até duas reposições por aluno/mês.
- Para a aula de reposição, tanto o preparatório quanto o relatório deverá ser individual.
- Os grupos deverão ser definidos já antes da primeira aula de laboratório e será formado por aqueles alunos que constarem na capa do primeiro relatório.
- Qualquer outra situação além das previstas aqui será decidida pelo professor.

$$L = \frac{\sum_{i=1}^{12} L_i}{12}$$

Onde Li = Nota do Preparatório (50%) + Nota do Relatório (50%)

Condições para aprovação

Após os cálculos das médias:

Se $M > 6$ → Aprovado
Se $4 \leq M < 6$ → VS
Se $M < 4$ → Reprovado

OBS:

- Os alunos que fizerem a VS deverão alcançar a nota mínima de 6,0 para serem aprovados;
- Não haverá reposição da P1 nem da P2. A VS fará o papel de prova substituta;
- Frequência mínima de 75% do total

Ementa

1. Sistemas de Numeração
   1.1. Logaritmos e base
   1.2. Codificação Posicional
   1.3. Representação de inteiros positivos
   1.4. Mudanças de base
   1.5. Números inteiros sinalizados
   1.6. Operações Aritméticas
   1.7. Números fracionários
2. Cálculo Proposicional
   2.1. Variáveis lógicas e funções lógicas
   2.2. Tabela Verdade
   2.3. Álgebra Booleana
   2.4. Postulados
   2.5. Identidades Booleanas. Teoremas de De Morgan
   2.6. Relação com Cálculo Proposicional
   2.7. Representação gráfica das funções
   2.8. Booleanas
   2.9. Formas Padrão
3. Simplificação de Funções Lógicas
   3.1. Simplificação algébrica
   3.2. Mapas de Veich-Karnaugh
   3.3. Método de Quine-McKluskey
4. Circuitos Combinacionais
   4.1. Abordagem hierárquica
   4.2. Somadores
   4.3. Blocos hierárquicos básicos
   4.4. Implementação com blocos básicos
5. Circuitos Sequenciais
   5.1. Latch
   5.2. Problemas de temporização
   5.3. Flip-flops JK, D e T

Por quê estudar técnicas digitais?

Sou da área de Engenharia de Telecomunicações. Por que devo estudar Técnicas Digitais?

- A tecnologia está migrando cada vez mais do "mundo analógico" para o "mundo digital", e isto não será reversível;
- As técnicas digitais clássicas e modernas são a base para a construção de microprocessadores, que por sua vez equipam celulares, roteadores, centrais telefônicas, etc...
- É a base para a cadeira de processamento digital de sinais, disciplina fundamental para trabalho com imagens, vídeo, som e texto atualmente;
- É a base para o entendimento da implementação via hardware de protocolos de comunicação e interfaces de redes.

Sistemas Analógicos x Sistemas Digitais

Sistemas Analógicos

Um sistema analógico contém dispositivos que manipulam quantidades físicas que são representadas na forma analógica. São exemplos de sistemas analógicos os microfones, alto-falantes, amplificadores de áudio, reguladores de luminosidade (dimmers), etc...

Sistemas Digitais

Um sistema digital é uma combinação de dispositivos projetados para manipular informações lógicas ou quantidades físicas que são representadas no formato digital, ou seja, as quantidades podem assumir valores discretos. São exemplos os computadores, calculadoras, telefonia, etc...

Vantagens

- São mais fáceis de projetar que os sistemas analógicos;
- O armazenamento de informações é mais fácil e eficiente;
- Mais fácil manter a precisão e exatidão em todo o sistema;
- As operações podem ser programadas;
- Maior imunidade a ruído;
- Capacidade de integração em um chip maior do que para circuitos analógicos (devido a capacitores e indutores);

Desvantagens

- Processar sinais digitais leva tempo → Sistemas digitais tendem a ser mais lentos que os analógicos;
- Corresponde a uma "aproximação" dos valores reais de grandezas → erros de "quantização";
- Requer uma "Banda Passante" maior do que a necessária para transmitir a mesma informação na forma analógica;
- Gera uma quantidade de *símbolos* muito grande.

Representações Numéricas

Existem basicamente duas formas diferentes de representação dos valores das quantidades:

Podem variar "continuamente" ao longo de uma faixa de valores.

Representação Analógica

Uma quantidade é representada por um indicador proporcional continuamente variável. Ex: velocímetro de um automóvel, termômetro de mercúrio, etc...

Podem variar somente em "saltos discretos" dentro de uma faixa de valores.

Representação Digital

As quantidades não são representadas por quantidades proporcionais, mas por símbolos denominados *dígitos*.

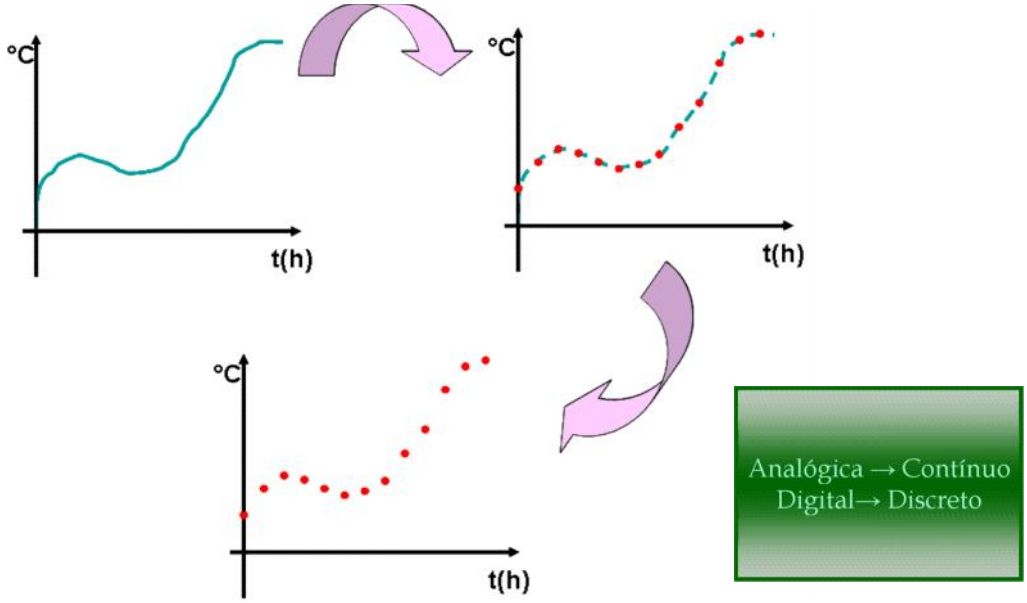


Figura 1: Um sinal analógico pode ser "discretizado".

Dentre as quantidades a seguir, quais são as que estão relacionadas a quantidades analógicas e quais as que estão relacionadas a quantidades digitais:

(a) Chave de dez posições
(b) A corrente que flui de uma tomada elétrica
(c) A temperatura de um ambiente
(d) Grãos de areia em uma praia
(e) O velocímetro de um automóvel

"...and then one day you'll find,
Ten years have got behind you
No one told you when to run,
You missed the starting gun."

Time, Pink Floyd
Album The Dark Side of the Moon

Sistemas de Numeração:

Os sistemas de numeração mais comuns no estudo de sistema digitais são:

- Decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 → 10 símbolos
- Octal: 0, 1, 2, 3, 4, 5, 6, 7 → 8 símbolos
- Hexadecimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F → 16 símbolos
- Binário: 0, 1 → 2 símbolos

O sistema de numeração decimal é o que estamos mais familiarizados em nosso dia-a-dia !

- Um computador que trabalhasse diretamente com o sistema de numeração hexadecimal seria muito mais eficiente que os computadores atuais, mas...
- Seria muito difícil de fabricá-lo e a probabilidade de interpretar um símbolo erroneamente seria alta. O circuito teria que interpretar 16 níveis de tensão corretamente, por exemplo: 0V, 1V, 2V, ..., 16V. Logo...

O sistema de numeração mais seguro deve ser aquele com o menor número de símbolos (dígitos) !

⇒ Sistema Binário

- Um possível problema no uso de máquinas binárias: o número binário precisa de mais dígitos para ser escrito do que o decimal.

$(2)_{10}$ número de animais representado em decimal

$(10)_2$ número de animais representado em binário

Vejamos um exemplo:

Quatro em decimal é representado como 4. Sua representação em binário é 100.

- **Consequência:** o computador binário seria mais preciso, porém muito lento porque a leitura da informação iria requerer mais tempo.

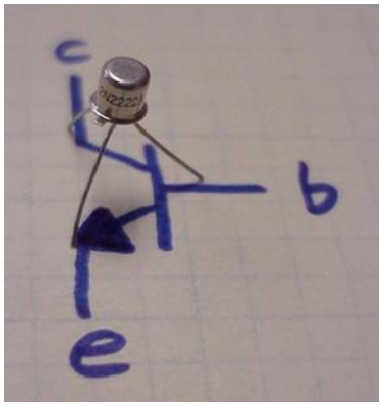
- **Uma solução:** o uso de dispositivos eletrônicos baseados na tecnologia dos semicondutores, como os transistores.

O **transistor:** é um dispositivo usado para controlar o fluxo de corrente. Ele tem duas características importantes:

1. É capaz de amplificar um sinal elétrico.
2. É capaz de chavear (comutar) entre ligado e desligado (ou fechado e aberto), deixando corrente passar através dele ou bloqueando-a. Essas condições são também denominadas "saturação" e "corte", respectivamente.

O **transistor** pode mudar da condição de saturação para o corte em velocidades acima de um milionésimo de segundo. Ele pode ser usado para caracterizar a presença (ou ausência) de um dígito binário (0 ou 1) e pode tomar decisões desse tipo a uma taxa superior a um milhão de decisões por segundo.

O primeiro Transistor



Um Transistor moderno

✓ Transistor: inventado nos Laboratórios da Bell Telephone em 12/1947 por John Bardeen, Walter Brattain e William Shockley – Prêmio Nobel de física de 1956. O transistor é capaz de comutar em um milionésimo de segundo entre o corte e a saturação.

Classificação

Os sistemas de numeração podem ser classificados da seguinte forma:

- Sistemas de numeração posicionais
- Sistemas de numeração não posicionais

Sistemas posicionais

- Nos sistemas de numeração posicional, o valor do dígito em um número depende da posição que ele ocupa neste mesmo número.

  1989 = 1000+900+80+9
  1989 = $1x10^3 + 9x10^2 + 8x10^1 + 9x10^0$

- Há um peso para cada posição ocupada pelo dígito. Os pesos crescem para esquerda na parte inteira e decrescem para a direita na parte fracionária

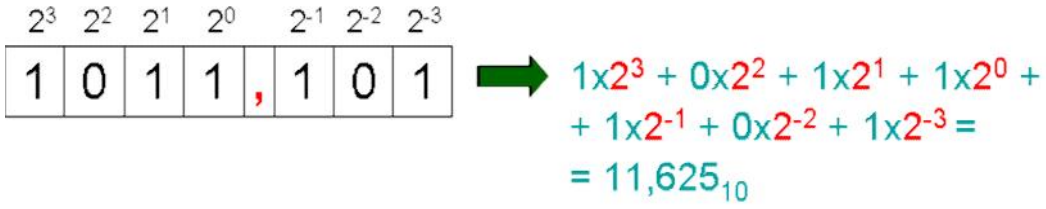
  1989,4= $1x10^3 + 9x10^2 + 8x10^1 + 9x10^0 + 4x10^{-1}$

Outro exemplo:

1. 27,35

Felizmente, os sistemas: binário, octal e hexadecimal também são sistemas posicionais. Logo, é possível aplicar os mesmos passos para estes sistemas. Vamos ver um exemplo para o sistema binário:

2. $(1011,101)_2$



Sistemas não posicionais

Nos sistemas de numeração não posicional, não há uma forma estruturada como a anterior para a indicação dos valores. Vamos tomar como exemplo o sistema de numeração Romano:

No número XX (vinte em decimal), o valor do dígito X à esquerda é o mesmo daquele à direita. Neste caso a representação é aditiva, com X representando a quantidade decimal 10, e com a combinação XX associada a 10+10=20. Por outro lado em IX (nove em decimal) a representação é subtrativa.

M = 1000
Como antes de M não tinha nenhuma letra, buscavam a segunda letra de maior valor.
D = 500
Depois tiravam de D o valor da letra que vem antes
D – C = 500 – 100 = 400
Somavam 400 ao valor de M porque CD está depois de M.
M – CD = 1000 + 400 = 1400
Sobrava apenas o V. Então:
MCDV = 1400 + 5= 1405

Geração de inteiros

Algoritmo de avanço de dígitos

Avançar um dígito de um alfabeto ordenado consiste em substituí-lo pelo próximo dígito na hierarquia. O dígito de maior valor do conjunto é sempre avançado para o aquele de menor valor na hierarquia.

$$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 0$$

Algoritmo de geração de inteiros

- O primeiro inteiro é o zero
- O próximo inteiro é obtido do precedente na lista avançando-se seu dígito mais à direita. No caso deste dígito avançar para zero, avança-se, então, o dígito adjacente à esquerda.

Exemplo: Gerar os 26 primeiros inteiros do sistema decimal.

$$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 11 \rightarrow 12 \rightarrow 13 \rightarrow 14 \rightarrow 15 \rightarrow 16 \rightarrow 17 \rightarrow 18 \rightarrow 19 \rightarrow 20 \rightarrow 21 \rightarrow 22 \rightarrow 23 \rightarrow 24 \rightarrow 25$$

- Observe que o nove avança para o zero, logo o dígito mais à esquerda (o zero, não mostrado explicitamente no número) é avançado para 1 gerando o próximo número na lista, o 10.

Contagem binária

Número binário de 4 bits				Número decimal equivalente
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

BIT = "Binary digIT"

Qual o maior número inteiro que possível representar com um número binário de N bits ?

$\Rightarrow 2^N$

Ex: $2^4 = (16)_{10}$
$2^8 = (256)_{10}$

Tabela 1: Contagem binária x decimal

Exercícios

Tocci – 10ª. Edição

1.1 e 1.2

Mudança de Bases

Para que possamos entender e projetar equipamentos digitais, é necessário que entendamos muito bem o sistema de numeração binário. Para isto, faz-se necessário aprendermos a converter números decimais em números **binários puros** (números binários sem sinal), e vice-versa.

Conversão de binário para decimal

$$(1101101)_2 \rightarrow ?_{10}$$

2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	1	0	1

$$(1\times2^6) + (1\times2^5) + (0\times2^4) + (1\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0) =$$

$$64 + 32 + 0 + 8 + 4 + 0 + 1 = 109_{10}$$

Converta o número abaixo para decimal

$(10110101)_2 \rightarrow$

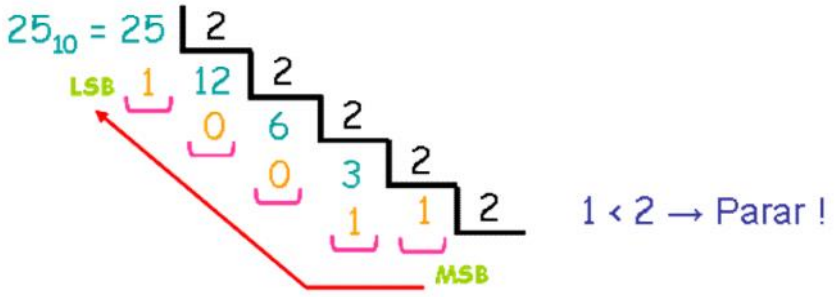
Conversão de decimal para binário

Existem basicamente dois métodos:

1. Inspeção

$$45_{10} = \begin{array}{cccccc} 32 & 16 & 8 & 4 & 2 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \end{array} \longrightarrow 101101_2$$

2. Divisão sucessiva



Converta o número abaixo para binário pelos dois métodos aprendidos

$(76)_{10}$

- Método da Inspeção

- Método das divisões sucessivas

Conversão de Números Binários Fracionários em Decimais

Até agora tratamos de números inteiros. E se no caso de números decimais fracionários, como convertê-los para binário?

Vamos analisar o exemplo abaixo:

$$101{,}101_2 = ?_{10}$$

Podemos escrever:

$$1x2^2 + 0x2^1 + 1x2^0 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3} =$$
$$= 4 + 1 + 0{,}5 + 0{,}125 = 5{,}625_{10}$$

Logo,

$$101{,}101_2 = 5{,}625_{10}$$

Converta o número binário abaixo para decimal:

$1010{,}1101_2 = ?_{10}$

Conversão de Números Decimais Fracionários em Binários

Agora vamos verificar o procedimento para converter números decimais fracionários em números binários

Vamos analisar o exemplo abaixo:

$$8{,}375_{10} = ?_2$$

Vamos transformar primeiro a parte inteira do número, depois transformaremos a parte fracionária:

$$8 \div 2 = 4 \rightarrow \text{resto} = 0$$
$$4 \div 2 = 2 \rightarrow \text{resto} = 0$$
$$2 \div 2 = 1 \rightarrow \text{resto} = 0$$
$$1 \div 2 \rightarrow \text{Não dá para dividir}$$

$$8_{10} = 1000_2$$

Agora vamos converter a parte fracionária. Para isto faremos multiplicações sucessivas das partes fracionárias resultantes pela base, até atingir zero.

$$0{,}375 \times 2 = 0{,}750 \rightarrow 1^{\circ}\text{ bit} = 0 \text{ (MSB)}$$
$$0{,}750 \times 2 = 1{,}500 \rightarrow 2^{\circ}\text{ bit} = 1$$

Quando atingirmos o número 1, e a parte fracionária não for nula, separamos esta última e reiniciamos o processo:

$$0{,}500 \times 2 = 1{,}000 \rightarrow 3^{\circ}\text{ bit} = 1 \text{ (LSB)}$$

O processo para aqui, pois obtivemos,000 na última multiplicação. Desta forma, temos:

$$0{,}375_{10} = 011_2$$

Para finalizarmos a conversão, basta juntarmos parte inteira da parte fracionária:

$$8{,}375_{10} = 1000{,}011_2$$

Converta o número decimal abaixo para binário:

$4{,}8_{10} = ?_2$

O que aconteceu neste exercício acima?

Este exercício nos mostra que nem todos os números decimais possuem uma representação em binário com um número finito de dígitos. Isto significa que os computadores acabam truncando números e gerando com isto um erro que, em alguns casos, podem ser cumulativos em cálculos complexos e/ou repetitivos.

Este é um dos motivos que levam os fabricantes de chips a aumentar o número de bits que um processador pode trabalhar: 8, 16, 32, 64 e 128 atualmente.

Sistema Octal

O sistema octal de numeração é um sistema de base 8 no qual existem 8 símbolos assim enumerados:

0,1,2,3,4,5,6 e 7

Atualmente, o sistema octal é pouco utilizado em eletrônica, mas concursos nas áreas de eletrônica, eletrotécnica, informática e telecomunicações sempre apresentam uma questão relacionada a este sistema.

DECIMAL	OCTAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	10
9	11
10	12
11	13
12	14
13	15
14	16
15	17
16	20

Tabela 2: Comparação entre o sistema decimal e octal.

Conversão do sistema octal para o sistema decimal

Para convertermos um número octal para decimal, utilizamos o conceito básico de sistema posicional, conforme já visto.

$$144_8 = ?_{10}$$

$$1 \times 8^2 + 4 \times 8^1 + 4 \times 8^0 = 64 + 32 + 4 = 100_{10}$$

Converta os números abaixo:

$77_8 = ?_{10}$

$100_8 = ?_{10}$

$476_8 = ?_{10}$

Conversão do sistema decimal para o sistema octal

O processo é análogo à conversão do sistema decimal para o sistema binário. Porém, a divisão é por 8:

Vejamos o seguinte exemplo:

$$92_{10} = ?_8$$

$$92 \div 8 = 11 \rightarrow \text{resto} = 4 \text{ (LSD)}$$
$$11 \div 8 = 1 \rightarrow \text{resto} = 3$$
$$1 \div 8 = \rightarrow \text{Não dá para dividir (MSD)}$$

Logo, temos:

$$92_{10} = 134_8$$

Converta os números abaixo:

$74_{10} = ?_8$

$512_{10} = ?_8$

$719_{10} = ?_8$

Conversão do sistema octal para o sistema binário

A conversão de um número octal para o número binário é bastante simples, pois basta transformar cada dígito em octal diretamente em seu equivalente binário, lembrando que cada dígito em octal corresponde sempre a 3 bits.

Seja o exemplo abaixo:

$$27_8 = ?_2$$

- 2 = 010
- 7 = 111

Logo, temos:

$$27_8 = 10111_2$$

Converta os números abaixo:

$34_8 = ?_2$

$536_8 = ?_2$

$44675_8 = ?_2$

Conversão do sistema binário para o sistema octal

Para efetuar esta conversão, basta aplicar o processo inverso ao utilizado na conversão de octal para binário.

Seja o exemplo abaixo:

$$110010_2 = ?_8$$

- 110 = 6
- 010 = 2

Logo, temos:

$$110010_2 = 62_8$$

Converta os números abaixo:

$11010101_2 = ?_8$

$1000110011_2 = ?_8$

Sistema Hexadecimal

Sistemas de Pesos

$$\cdots\ 16^4\ 16^3\ 16^2\ 16^1\ 16^0\ ,\ 16^{-1}\ 16^{-2}\ 16^{-3}\ 16^{-4}\cdots$$

DECIMAL	BINARIO	HEXADECIMAL
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

OBS: 1 dígito em hexadecimal corresponde a 4 dígitos em binário !

Contagem em hexadecimal

0	10	20	30	·	90	A0	·
1	11	21	31	·	91	A1	·
2	12	22	32	·	92	A2	·
3	13	23	33	·	93	A3	·
4	14	24	34	·	94	A4	·
5	15	25	35	·	95	A5	·
6	16	26	36	·	96	A6	·
7	17	27	37	·	97	A7	·
8	18	28	38	·	98	A8	·
9	19	29	39	·	99	A9	·
A	1A	2A	3A	·	9A	AA	·
B	1B	2B	3B	·	9B	AB	·
C	1C	2C	3C	·	9C	AC	·
D	1D	2D	3D	·	9D	AD	·
E	1E	2E	3E	·	9E	AE	·
F	1F	2F	3F	·	9F	AF	·

Conversão de hexadecimal para decimal

$$356_{16} = 3 \times 16^2 + 5 \times 16^1 + 6 \times 16^0 =$$
$$= 768 + 80 + 6 =$$
$$= 854_{10}$$

Conversão de decimal para hexadecimal

Obs: A conversão também pode ser feita pelo método da inspeção !

$423_{10} = 423 \mid 16$

$26 \mid 16$

LSD 7 10 1 MSD

$1 < 16 \rightarrow$ Parar !

$10_{10} \Rightarrow A_{16} \rightarrow 423_{10} = 1A7_{16}$

Conversão de hexadecimal para binário

$9F2_{16} =$ 9 F 2

1001_2 1111_2 0010_2

$9F2_{16} = 1001\ 1111\ 0010_2$

Conversão de binário para hexadecimal

$1110100110_2 = 0011 \mid 1010 \mid 0110$

3_{16} A_{16} 6_{16}

$1110100110_2 = 3A6_{16}$

Quantos números inteiros podemos representar com N dígitos na base hexadecimal?

$$16^N$$

Ex: N = 3 dígitos

$16^3 = 4096$ números inteiros $\Rightarrow$ 0..4095

Códigos

Quando números, letras ou palavras são representadas por um grupo especial de símbolos, dizemos que eles estão codificados, sendo o grupo de símbolos denominado *código*.

Ex: Código Morse

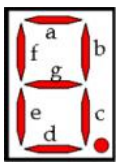
OBS: Quando um número decimal é representado pelo seu número binário equivalente, dizemos que é uma "codificação em binário puro"!

Ex: $1101_2 \rightarrow 13_{10}$

Letra	Código Internacional	Letra	Código Internacional
A	.-	N	-.
B	-...	O	---
C	-.-.	P	.--.
D	-..	Q	--.-
E	.	R	.-.
F	..-.	S	...
G	--.	T	-
H	....	U	..-
I	..	V	...-
J	.---	W	.--
K	-.-	X	-..-
L	.-..	Y	-.--
M	--	Z	--..

Código BCD

BCD → "Binary Coded Decimal" *(Decimal Codificado em Binário)*



- Cada dígito decimal deverá ser codificado em 4 bits em binário.
- Os números em código BCD variam de 0000 a 1001.

Os códigos BCD são muito utilizados em interfaces homem-máquina, principalmente através de *displays* (Exemplo: Display BCD-7 segmentos).

Código de Gray

Este código visa reduzir a "probabilidade" de ocorrência de erro nos equipamentos digitais devido à interpretação de vários bits modificando-se simultaneamente em certas situações;

No código de Gray, um número binário tem apenas 1 único bit modificado em relação ao número antecessor ou sucessor;

B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

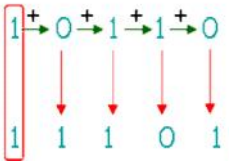
A linha sucessora ou antecessora só possui um bit de diferença !

Conversão de binário para Código de Gray

1. O MSB em ambos é sempre o mesmo;
2. Da esquerda para a direita, some cada par de bits adjacentes no código binário para obter o próximo bit do código Gray.

$0 + 0 = 0$ $\quad$ $0 + 1 = 1$
$1 + 0 = 1$ $\quad$ $1 + 1 = 0$

Ex: $10110_2 \rightarrow$? em código Gray



R: $10110_2 \rightarrow 11101$ (Gray)

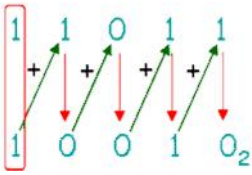
(em código Gray)

Conversão de Código de Gray para binário

1. O MSB em ambos é sempre o mesmo;
2. Some cada bit do código binário gerado ao bit do código Gray na próxima posição adjacente.

$0 + 0 = 0$ $\quad$ $0 + 1 = 1$
$1 + 0 = 1$ $\quad$ $1 + 1 = 0$

Ex: 11011 (em código Gray) → ? em código binário



R: 11011 (Gray) → 10010_2 (em código binário)

Aplicações do Código de Gray

O Código de Gray possui aplicações em:

1. Medida de posição de eixo mecânico
2. Mapas de Karnaugh (simplificação de expressões Booleanas)
3. Modulação digital em telecomunicações

Códigos Alfanuméricos

Os códigos alfanuméricos são utilizados em interfaces homem-máquina, uma vez que a interpretação de informações em binário, para nós seres humanos, é exaustiva e complicada. Desta forma, deixamos que as máquinas façam o trabalho de conversão de números e letras para códigos binários e vice-versa.

Em geral, um código alfanumérico possui no mínimo:

1. 26 letras minúsculas;
2. 26 letras maiúsculas;
3. 10 dígitos numéricos;
4. 7 sinais de pontuação;
5. 20 a 40 caracteres especiais como % $ & * \ / | # @

Código ASCII

- ✓ American Standard Code for Information Interchange;
- ✓ Código de 7 bits $\rightarrow 2^7 = 128$ símbolos;
- ✓ Foi muito utilizado em teclados para pequenos computadores;

EX:

$A = 65_{10}$

$a = 97_{10}$

$TAB = 9_{10}$

A Tabela a seguir apresenta o chamado Código ASCII.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

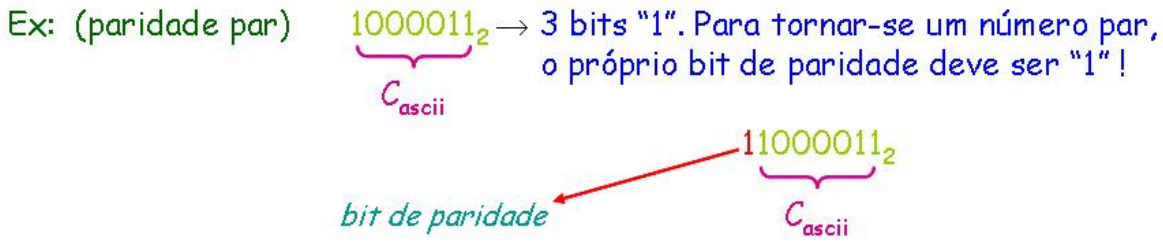
Outros Códigos Alfanuméricos

- ASCII Estendido → Mais 128 símbolos além dos originais. Utilizado em computadores pessoais, substituiu o ASCII inicial.
- EBCDIC → Foi muito utilizado em computadores de grande porte.
- UNICODE → Novo padrão que vem substituindo o ASCII entre outros

Códigos de Detecção de Erro

Método da Paridade

- Paridade Par: O número de 1s presentes no código binário, incluindo o próprio bit de paridade, deve ser par.
- Paridade ímpar: O número de 1s presentes no código binário, incluindo o próprio bit de paridade, de ser ímpar.

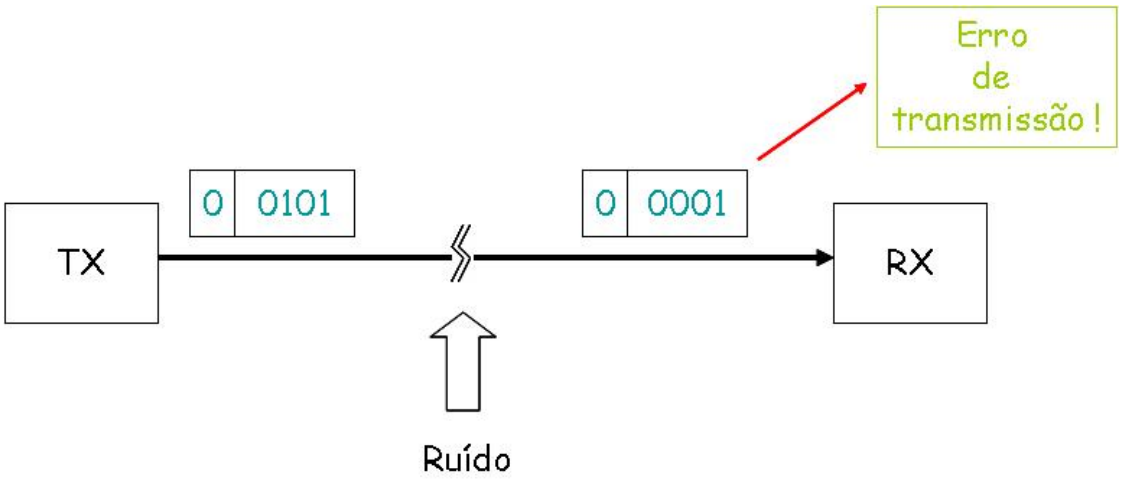


A Figura abaixo apresenta o bit de paridade para diversos exemplos, tanto para o caso de paridade par como para o caso de paridade ímpar.

Paridade PAR		Paridade ÍMPAR	
P	*BCD*	*P*	*BCD*
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	1000	0	1000
0	1001	1	1001

bit de paridade *bit de paridade*

Ex: Transmitir o BCD 0101 (paridade par)



Neste caso, o *nibble* 0101 deve ser transmitido através de um meio de comunicação (par metálico, rádio, satélite, fibra óptica), mas o ruído afeta a integridade da informação.

Para isto, o transmissor calcula o bit de paridade par para a informação a ser transmitida (o *nibble*) e acrescenta-o ao final, ficando 00101.

O receptor, por sua vez, recebe a informação 00001. Como ele faz para descobrir que houve erro de transmissão?

Ele calcula o bit de paridade para o *nibble* recebido (0001 → P = 1) e compara o bit de paridade calculado com o bit de paridade recebido (00001 → P = 0) e conclui que houve erro porque os dois bits de paridade, calculado e recebido são diferentes.

E se ocorre erro em mais de um bit durante a transmissão ?

Os bits abaixo foram recebidos. Verifique se houve erro durante a transmissão/recuperação, sabendo-se que a paridade utilizada é ímpar:

1. 10110101 →
2. 011011000110 →
3. 10001101101010110100 01 →

Códigos de Hamming

- Um único bit de paridade pode indicar que existe um erro num certo grupo de bits, mas não é possível identificar "onde" ocorreu o erro;
- O código de correção de erro de Hamming permite a localização e a correção de um único bit errado.
- O preço a pagar é a inserção de bits de paridade a mais.

São 4 passos:

1. Determinar o número de bits de paridade necessários
2. Inserção dos bits de paridade no código
3. Determinação dos valores dos bits de paridade
4. Determinar o código resultante

Passo 1: Determinar o número de bits de paridade necessários

Se o número de bits de dados projetado for d, então o número de bits de paridade, p, é determinado pela seguinte relação:

$$2^p \geq d + p + 1$$

Exemplo:

- 4 bits de dados (d)
  - Se p = 2 → 4 ≥ 4 + 2 + 1 → *incoerente*!
  - Se p = 3 → 8 ≥ 4 + 3 + 1 → *ok*

Logo, são necessários 3 bits de paridade!

Passo 2: Inserção dos bits de paridade

Agora que sabemos determinar o número de bits de paridade, temos que arranjar os bits adequadamente no código.

O bit mais à esquerda é designado como bit 1, o próximo bit é o 2 e assim por diante, conforme a seguir:

bit 1, bit 2, bit 3, bit 4, bit 5, bit 6, bit 7

Os bits de paridade estão localizados nas posições que são numeradas em correspondência às potências de dois ascendentes (1, 2, 4, 8, ...), conforme indicado:

P_1, P_2, D_1, P_3, D_2, D_3, D_4

O símbolo P_n designa um bit de paridade em particular e D_n designa um bit de dado em particular.

Passo 3: Determinação dos valores dos bits de paridade

Finalmente, temos que designar adequadamente o valor 0 ou 1 a cada bit de paridade.

1. Para determinar o valor do bit, primeiro numere cada posição de bit em binário;
2. Indique a localização dos bits de dados e de paridade, conforme a tabela abaixo;
3. Observe que o número da posição em binário do bit de paridade P_1 tem 1 no dígito mais à direita. *Esse bit de paridade verifica as posições de todos os bits, incluindo ele mesmo, que tem 1s na mesma posição nos números de posição em binário;*

Designação dos bits	P_1	P_2	D_1	P_3	D_2	D_3	D_4
Posição dos bits	1	2	3	4	5	6	7
Número da pos. bits	001	010	011	100	101	110	111
Bits de Dados (Dn)							
Bits de Paridade (Pn)							

→ O bit de paridade P_1 verifica as posições 1, 3, 5 e 7.

Designação dos bits	P_1	P_2	D_1	P_3	D_2	D_3	D_4
Posição dos bits	1	2	3	4	5	6	7
Número da pos. bits	001	010	011	100	101	110	111
Bits de Dados (Dn)							
Bits de Paridade (Pn)							

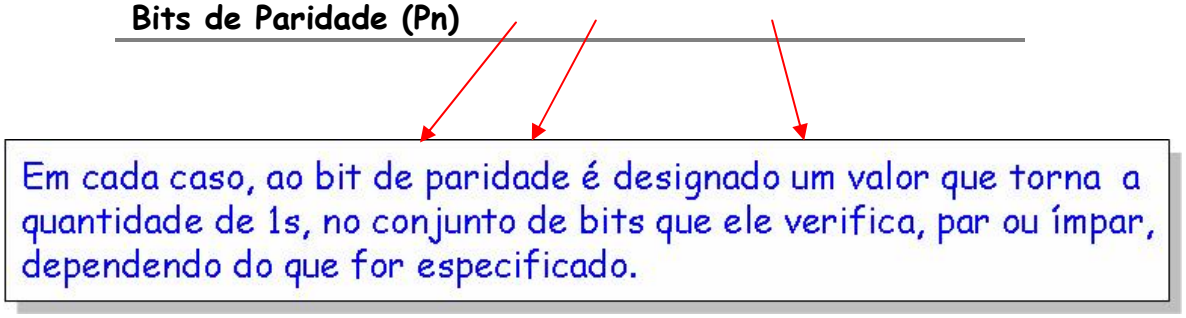
→ O bit de paridade P_2 verifica as posições 2, 3, 6 e 7.

Designação dos bits	P_1	P_2	D_1	P_3	D_2	D_3	D_4
Posição dos bits	1	2	3	4	5	6	7
Número da pos. bits	001	010	011	100	101	110	111
Bits de Dados (Dn)							
Bits de Paridade (Pn)							

→ O bit de paridade P_3 verifica as posições 4, 5, 6 e 7.

Passo 4: Determinar o código resultante

Designação dos bits	P_1	P_2	D_1	P_3	D_2	D_3	D_4
Posição dos bits	1	2	3	4	5	6	7
Número da pos. bits	001	010	011	100	101	110	111
Bits de Dados (Dn)							
Bits de Paridade (Pn)							



Exemplo: Determine o código de Hamming para o número BCD 1001 (bit de dados), usando paridade par:

- ✓ Passo 1: Determinar o número de bits de paridade. Suponha que p = 3.

$$2^p = 2^3 = 8$$
$$d + p + 1 = 4 + 3 + 1 = 8 \rightarrow Ok$$

Total de bits de código = 3 (paridade) + 4 (dados) = 7 bits

- ✓ Passo 2: Construa uma tabela de posição de bits e insira os bits de dados.

Designação dos bits	P_1	P_2	D_1	P_3	D_2	D_3	D_4
Posição dos bits	1	2	3	4	5	6	7
Número da pos. bits	001	010	011	100	101	110	111
Bits de Dados (Dn)			1		0	0	1
Bits de Paridade (Pn)							

- ✓ Passo 3: Determine os bits de paridade como a seguir:
  - ❑ O bit P_1 verifica os bits das posições 1, 3, 5 e 7 e tem que ser 0 para que o número de 1s (2) seja par nesse grupo;
  - ❑ O bit P_2 verifica os bits das posições 2, 3, 6 e 7 e tem que ser 0 para que o número de 1s (2) seja par nesse grupo;
  - ❑ O bit P_3 verifica os bits das posições 4, 5, 6 e 7 e tem que ser 1 para que o número de 1s (2) seja par nesse grupo;
- ✓ Passo 4: Determine o código resultante inserindo os bits de paridade em suas respectivas posições.

Designação dos bits	P_1	P_2	D_1	P_3	D_2	D_3	D_4
Posição dos bits	1	2	3	4	5	6	7
Número da pos. bits	001	010	011	100	101	110	111
Bits de Dados (Dn)			1		0	0	1
Bits de Paridade (Pn)	0	0		1			

Neste exemplo, o código resultante é 0011001.

Determine o código de Hamming para os bits de dados 10110, usando a paridade par.

Detecção e Correção de Erro

Agora que o método de Hamming para construção de um código de erro foi abordado, como o usamos para localizar e corrigir um erro?

- Passo 1: Comece com o grupo verificado por P_1;
- Passo 2: Verifique o grupo quanto a paridade correta. Um 0 representa uma verificação de paridade *correta* e um 1 representa uma verificação incorreta;
- Passo 3: Repita o passo 2 para cada grupo de paridade;
- Passo 4: O número binário formado pelo resultado de todas as verificações de paridade determina a posição do bit do código que está errado. Esse é o *código de posição de erro*. A primeira verificação de paridade gera o LSB. Se todas as verificações forem corretas, não há erro.

Exemplo: Considere que a palavra de código dada no penúltimo exemplo 0011001 seja transmitida e que a palavra de código 0010001 seja recebida. O receptor não "sabe" o que foi transmitido e tem que testar os bits de paridade para determinar se o código está correto. A única coisa que o receptor deve "saber" previamente é que a paridade utilizada é par.

Designação dos bits	P_1	P_2	D_1	P_3	D_2	D_3	D_4
Posição dos bits	1	2	3	4	5	6	7
Número da pos. bits	001	010	011	100	101	110	111
Dado recebido	0	0	1	0	0	0	1
Verificação de Paridade							

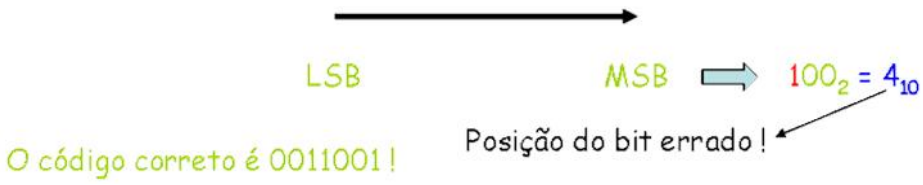
P_1 → 0, 1, 0 e 1 → Portanto, um número par de 1s. Logo P_1 deve ser zero! Ok, é zero! → Seu bit de Verificação de Paridade deve ser 0.

P_2 → 0, 1, 0 e 1 → Portanto, um número par de 1s. Logo P_2 deve ser zero! Ok, é zero! → Seu bit de Verificação de Paridade deve ser 0.

P_3 → 0, 0, 0 e 1 → Portanto, um número ímpar de 1s. Logo P_3 deveria ser um! Ih...não é zero! → Logo, seu bit de Verificação de Paridade deve ser 1.

A leitura da posição do bit errado deve ser feita através dos bits de Verificação de Paridade, sempre da esquerda para a direita.

Designação dos bits	P_1	P_2	D_1	P_3	D_2	D_3	D_4
Posição dos bits	1	2	3	4	5	6	7
Número da pos. bits	001	010	011	100	101	110	111
Dado recebido	0	0	1	0	0	0	1
Verificação de Paridade	0	0		1			



O código 101101010 é recebido, verifique se há erros e corrija-o se for o caso. A paridade utilizada é ímpar.

Exercícios

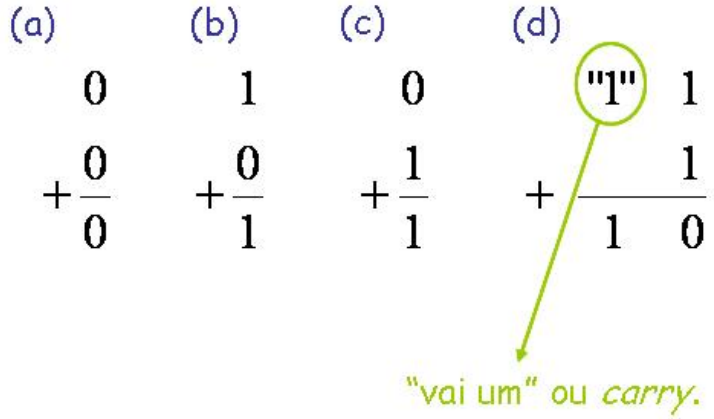
Tocci – 10ª edição

1.3 a 1.10
2.1 a 2.41

Aritmética Digital

Adição Binária

- Realizada da mesma forma que a adição dos números decimais.
- Quatro casos podem ocorrer na soma binária



Exemplos

$$\begin{array}{r} 011 \ (3) \\ +\ 110 \ (6) \\ \hline 1001 \ (9) \end{array} \qquad \begin{array}{r} 1001 \ (9) \\ +\ 1111 \ (15) \\ \hline 11000 \ (24) \end{array} \qquad \begin{array}{r} 11{,}011 \ (3{,}375) \\ +\ 10{,}110 \ (2{,}750) \\ \hline 110{,}001 \ (6{,}125) \end{array}$$

Não é necessário considerar a adição de mais de dois números binários simultaneamente porque em todos os sistemas digitais, o circuito que realiza a adição pode efetuar uma operação apenas com dois números de cada vez.

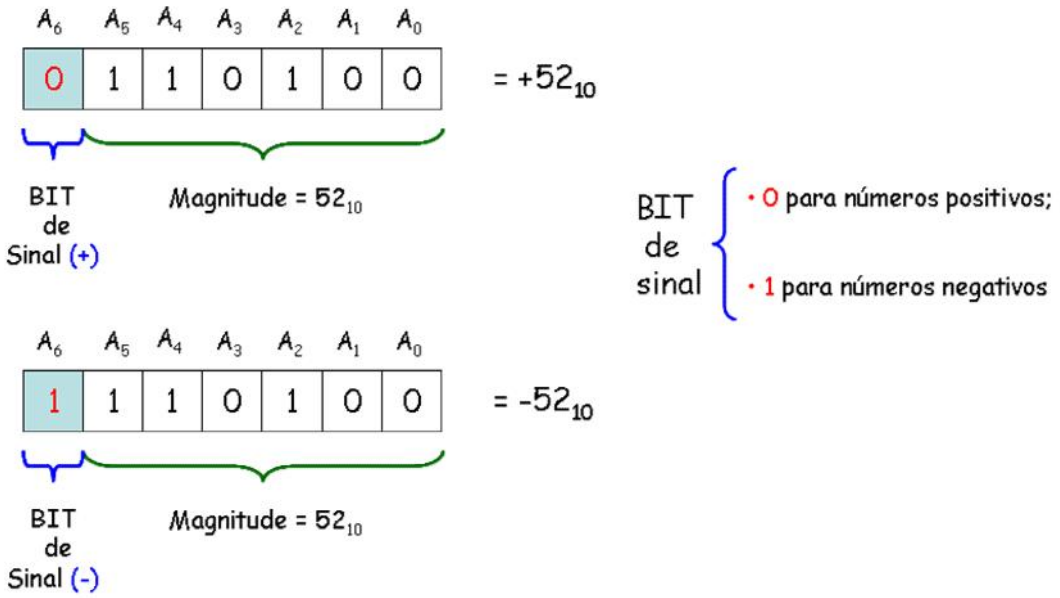
A adição é a operação aritmética mais importante nos sistemas digitais porque as operações de subtração, multiplicação e divisão, na realidade, são realizadas através apenas da adição.

Representação de números com sinal

- Necessário "arrumar" uma maneira de trabalhar com números positivos e negativos;
- Existem 3 formas de representarmos números positivos e negativos em binário:
-

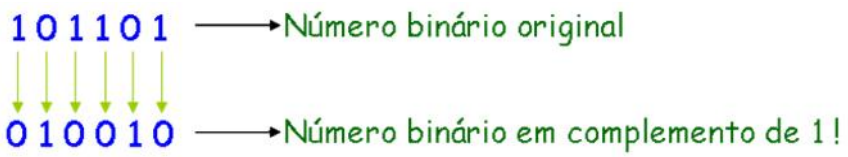
1. Sinal-Magnitude
2. Complemento a 1
3. Complemento a 2

Forma de Sinal Magnitude



Embora o sistema sinal-magnitude seja uma representação direta, os computadores e as calculadoras não utilizam porque a implementação do circuito é mais complexa do que em outros sistemas !

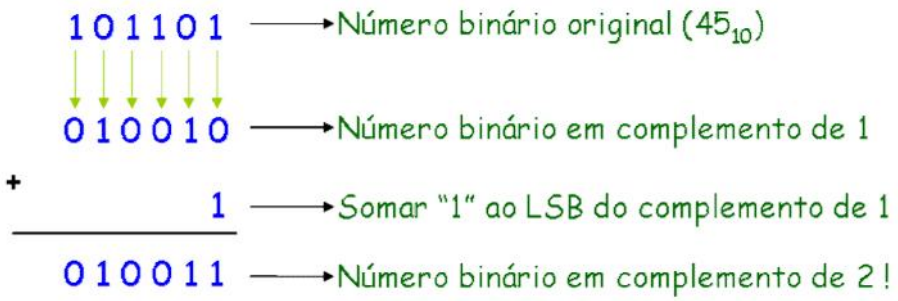
Forma de Complemento de 1



Assim, dizemos que o complemento de 1 de 101101 é 010010.

Forma de Complemento de 2

O complemento de 2 de um número binário é obtido tomando-se o complemento de 1 do número e somando "1" ao LSB.



Assim, dizemos que o complemento de 2 de 101101 é 010011.

Representação de números com sinal usando Complemento de 2

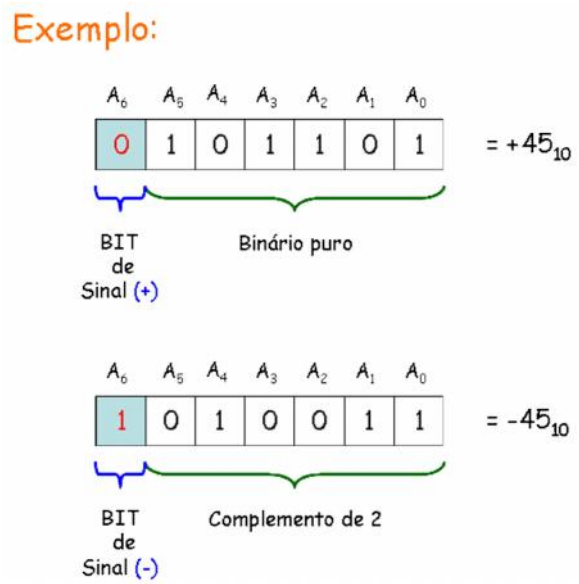
Duas Regras:

1. Se o número for positivo, a magnitude é representada na forma binária pura, e um bit de sinal "0" é colocado em frente ao MSB;

2. Se o número for negativo, a magnitude é representada na forma do complemento de 2, e um bit de sinal "1" é colocado na frente ao MSB.

O sistema de complemento de 2 é usado para representar números com sinal porque nos permite realizar a operação de subtração efetuando, na verdade, uma operação de adição. Isso é importante porque um computador digital pode usar o mesmo circuito tanto na adição quanto na subtração, desse modo poupando hardware.

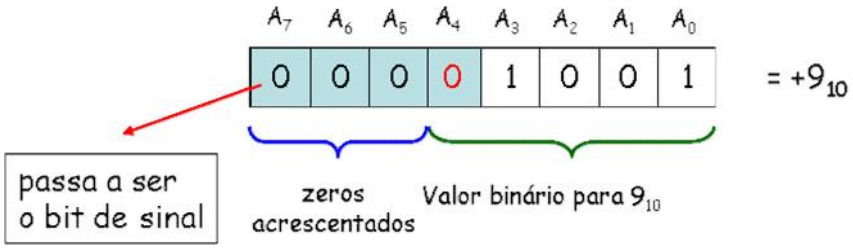
Exemplo:



Extensão de sinal para números positivos

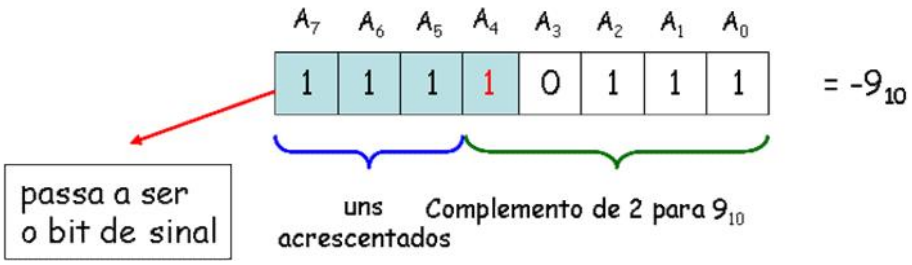
Como representar o número +9 através de 8 bits se o mesmo necessita de apenas 5 bits (já incluindo o bit de sinal) ?

R: Acrescentando 3 "zeros" à esquerda do sinal de bit.



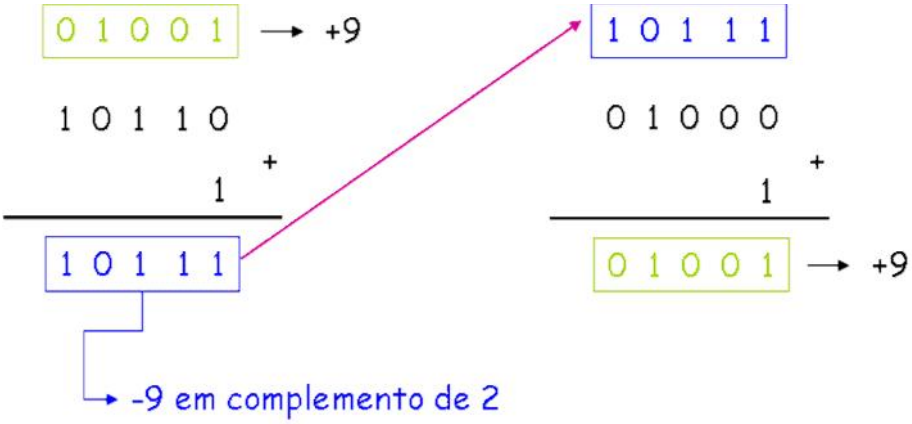
E como representar o número -9 através de 8 bits ?

R: Acrescentando 3 "uns" à esquerda do sinal de bit.



Negação

Negação é a operação de conversão de um número positivo em seu equivalente negativo ou a conversão de um número negativo em seu equivalente positivo.



Observações:

- Sempre que um número com sinal tiver 1 no bit de sinal e todos os bits de magnitude forem 0, seu equivalente decimal será -2N, em que N é o número de bits da magnitude:

$1000 = -2^3 = -8$
$10000 = -2^4 = -16$
$100000 = -2^5 = -32$

- Podemos dizer que a faixa completa de valores que pode ser representada no sistema de complemento de 2 com N bits é:

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

Adição no Sistema de Complemento de 2

- Caso I: Dois números positivos

```
+9 → 0 1 0 0 1
+4 → 0 0 1 0 0 +
-------------------
+13    0 1 1 0 1 → +13(10)
```

- Caso II: Um número positivo e um outro menor negativo

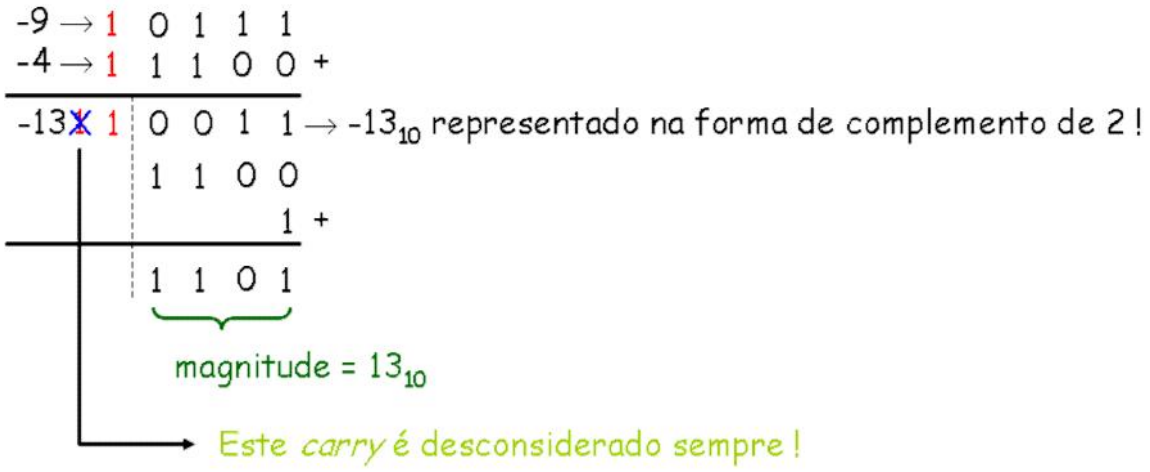
```
+9 → 0 1 0 0 1
-4 → 1 1 1 0 0 +
-------------------
+5  X 0 0 1 0 1 → +5(10)
```

Este *carry* é desconsiderado sempre !

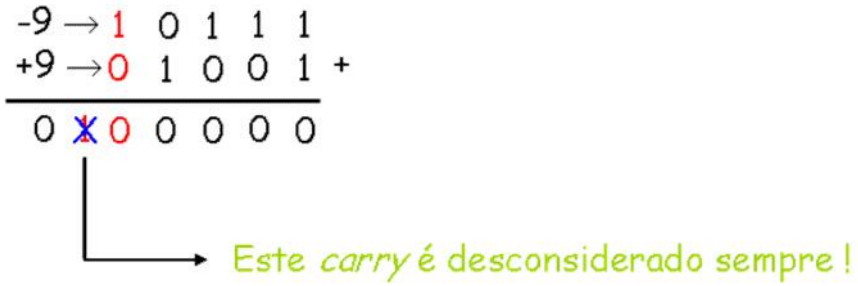
- Caso III: Um número positivo e um outro maior negativo

```
-9 → 1 0 1 1 1
+4 → 0 0 1 0 0 +
-------------------
-5   1 | 1 0 1 1 → -5(10) representado na forma de complemento de 2 !
       | 0 1 0 0
       |       1 +
-------------------
       | 0 1 0 1
       | magnitude = 5(10)
```

- Caso IV: Dois números negativos



- Caso V: Números iguais e de sinais opostos



Execute as seguintes operações em binário através do complemento de 2:

a) 6 + 4

b) 7 – 3

c) -6 – 5

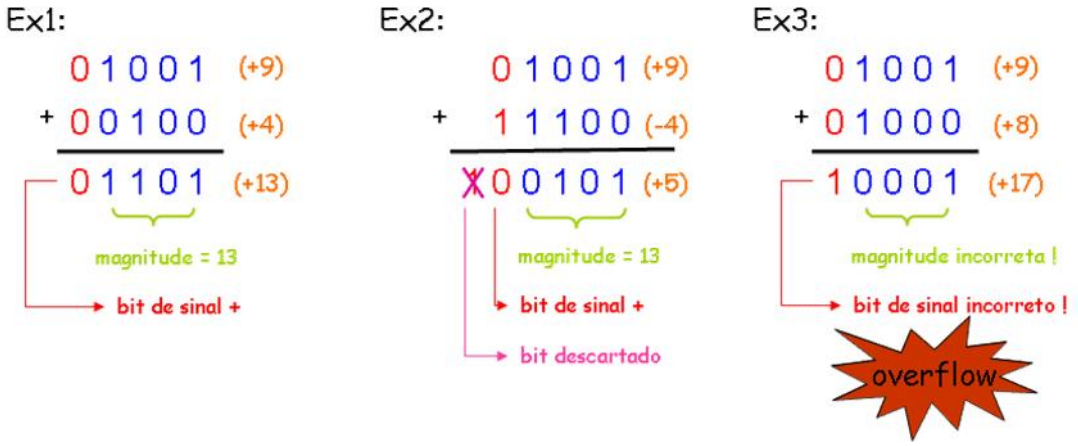
d) 4 - 8

Exercícios

Tocci – 10ª edição
6.1 a 6.7

Overflow Aritmético

Suponha que dois números binários sejam representados por 1 bit de sinal mais 4 bits de magnitude e que a soma destes números também deva ser representada por 1 bit de sinal e 4 bits de magnitude.



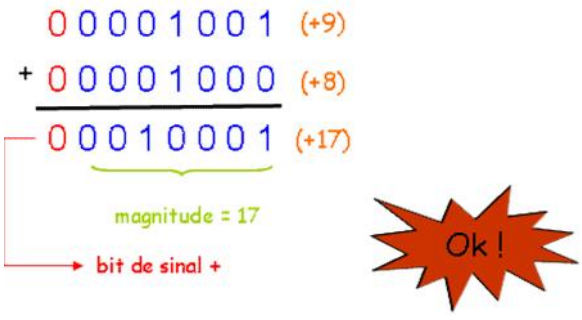
- A resposta deveria ser +17, mas a magnitude 17 requer mais que quatro bits para representá-la, logo, ocorre um overflow (transbordo) na posição do bit de sinal.

- O overflow só pode ocorrer quando dois números positivos ou dois números negativos são somados, e isto pode produzir resultados errôneos.

- O overflow pode ser detectado verificando se o bit de sinal do resultado tem o mesmo valor dos bits de sinal dos números que estão sendo somados.

- Um computador possui um circuito especial para detectar qualquer condição de overflow quando dois números são somados ou subtraídos.

Refaça a soma de +8 e +9 representando-os por um bit de sinal e sete bits de magnitude:



Arianne 5 Rocket Failure

On the morning of June 4, 1996, a French Arianne 5 rocket, carrying a European Space Agency (ESA) satellite, was scheduled for its first launch in French Guyana. About 37 seconds into its flight, the rocket veered off its flight path, broke up, and exploded. A board of inquiry was immediately appointed by the ESA and CNES (Centre National des Etudes Spatiales).

The investigation examined telemetry data received on ground through 42 seconds after liftoff, trajectory data from radar stations, observations from infrared cameras, and the inspection of recovered debris. The origin of the failure was soon narrowed to the flight control system. Fortunately, the two primary computer-controlled inertial reference subsystems were both recovered, and after further investigation the source of the failure was determined to be a software error within these units.

Particularly, the error was traced to software controlling the alignment of the rocket's strap-down inertial platform. An integer overflow error occurred when the program attempted to convert a 64-bit floating point number to a 16-bit integer. The floating point number, which measured a quantity related to the horizontal velocity of the platform, was simply too large to be represented as a 16-bit integer. The Arianne 5 software had been derived from the software for the previous generation Arianne 4 rocket. The Arianne 4 had a different initial trajectory that produced smaller horizontal velocity values. Hence, the larger values recorded during the Arianne 5 flight were out of the range that the software was designed to handle.

The overflow error caused the computer in the primary inertial reference system to shut down and attempt to switch to the backup (redundant) system. Unfortunately, the redundant system had experienced exactly the same fault and had already shut itself down when the primary computer attempted to transfer control to it.

Had this problem been identified in preflight software testing, it could have been corrected very easily. Such is the case for many software errors. The smallest of problems can cause a program to crash or shut down. This is one of the reasons why it is so much harder to engineer reliable software than to engineer reliable bridges and other complex physical structures. An analogy might be that the failure of one very small bolt in a bridge could cause the entire structure to collapse-possible perhaps, but highly unlikely. Yet the failure of "small bolts" often causes software systems to collapse. Our next example provides further demonstration of this fact.

Fonte: http://cs.furman.edu/digitaldomain/themes/risks/risks_numeric.htm

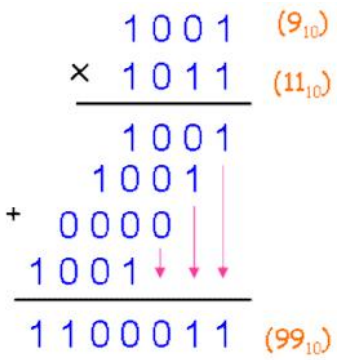
Acessar o video em http://www.youtube.com/watch?v=kYUrqdUyEpI

Multiplicação Binária

Em binário puro

Feito da mesma maneira que a multiplicação de números decimais, porém mais fácil!

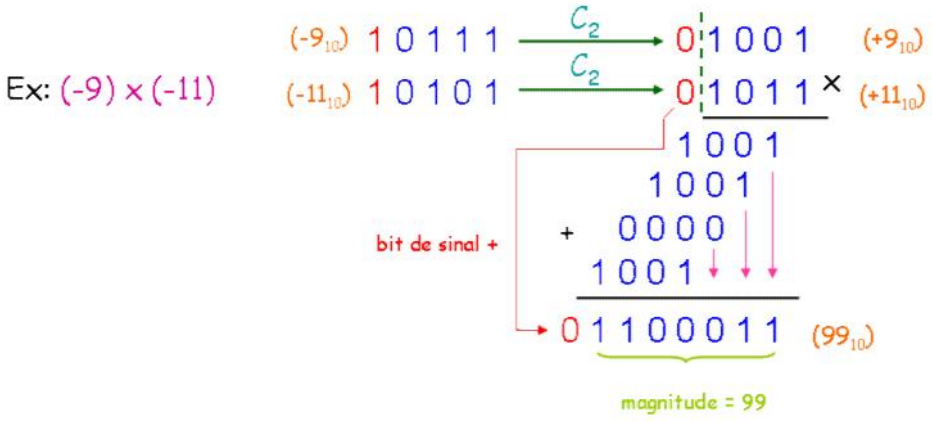
Exemplo:



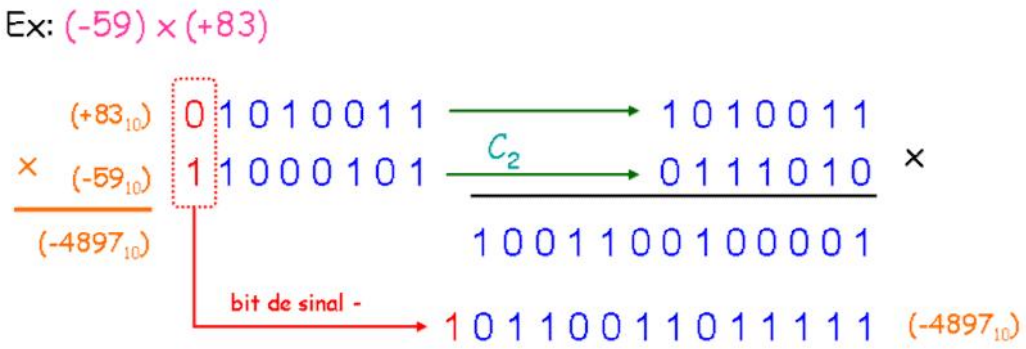
No sistema de complemento de 2

A. Números positivos → multiplicação binária vista anteriormente;

B. Números negativos → deverão estar na forma de complemento de 2. Aplicar a negação em ambos os números e multiplicar como em A;



C. Um números positivo e o outro negativo → O sinal do resultado da multiplicação será obrigatoriamente negativo. Aplicar a negação no número negativo (que deverá estar expresso em complemento de 2), proceder com a multiplicação de números binários normalmente, e aplicar o complemento de 2 no resultado. Juntar com o bit de sinal negativo e está pronto!



Divisão Binária

Os termos de uma divisão são o dividendo, o quociente e o divisor.

$$\frac{Dividendo}{Divisor} = quociente$$

A operação de divisão nos computadores é realizada utilizando-se subtrações. Como subtrações são feitas com um somador, a divisão também pode ser realizada com um somador.

O resultado de uma divisão é denominada quociente. O quociente é o número de vezes que o divisor "cabe dentro" do dividendo. Isto significa que o divisor pode ser subtraído a partir do dividendo um número de vezes igual ao quociente.

O sinal do quociente depende dos sinais do dividendo e do divisor de acordo com as seguintes regras:

- Se os sinais forem iguais, o quociente é positivo;
- Se os sinais forem diferentes, o quociente é negativo;

Quando dois números binários são divididos, os dois números tem que estar na forma verdadeira (não complementados).

Os passos básicos no processo de divisão são os seguintes:

1. Determine se os sinais do dividendo e do divisor são iguais ou diferentes. Isso determinará o sinal do quociente. Inicialize o quociente com zero;

2. Subtraia o divisor a partir do dividendo usando a adição do complemento de 2 para obter o primeiro resto parcial e somar "1" ao quociente. Se esse resto parcial for positivo, vá para o passo 3. Caso o resto parcial seja zero ou negativo, a divisão está completa.

3. Subtraia o divisor a partir do dividendo e some "1" ao quociente. Se o resultado for zero ou negativo, a divisão está completa.

4. Continue o divisor a partir do dividendo e os restos parciais até obter um resultado zero ou negativo. Conte o número de vezes que o divisor é subtraído e você terá o quociente.

Exemplo: Efetue a divisão de 01100100 por 00011001.

Passo 1: Os sinais dos dois números são positivos, de forma que o quociente será positivo. O quociente é inicializado em zero: 0000 0000.

Passo 2: Subtraia o divisor a partir do dividendo usando a adição do complemento de 2 (lembre-se que o carry final é descartado)

```
  0 1 1 0 0 1 0 0   dividendo
+ 1 1 1 0 0 1 1 1   complemento de 2 do divisor
  0 1 0 0 1 0 1 1   1º. resto parcial positivo
```

Some 1 ao quociente:

quociente = 0000 0001

Passo 3: Subtraia o divisor do 1º. resto parcial usando a adição do complemento de 2.

```
  0 1 0 0 1 0 1 1   1º. resto parcial
+ 1 1 1 0 0 1 1 1   complemento de 2 do divisor
  0 0 1 1 0 0 1 0   2º. resto parcial positivo
```

Some 1 ao quociente:

quociente = 0000 0010

Passo 4: Subtraia o divisor do 2º. resto parcial usando a adição do complemento de 2.

```
  0 0 1 1 0 0 1 0   2º. resto parcial
+ 1 1 1 0 0 1 1 1   complemento de 2 do divisor
-----------------
  0 0 0 1 1 0 0 1   3º. resto parcial positivo
```

Some 1 ao quociente:

quociente = 0000 0011

Passo 5: Subtraia o divisor do 3º. resto parcial usando a adição do complemento de 2.

Some 1 ao quociente:

```
  0 0 0 1 1 0 0 1   3º. resto parcial
+ 1 1 1 0 0 1 1 1   complemento de 2 do divisor
-----------------
  0 0 0 0 0 0 0 0   resto zero
```

quociente = 0000 0100

O processo está completo e o resultado da divisão é 0000 0100.

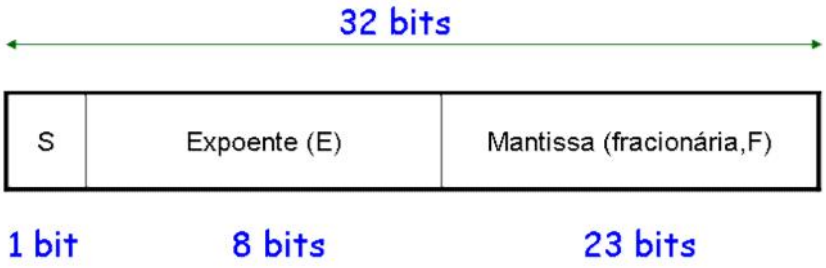
Números em Ponto Flutuante

A representação de números fracionários, grandes ou pequenos, pode ser realizada de forma mais eficiente através da notação científica, sem aumentar muito o número de bits.

Números Binários em P.F. (Norma 754-2008 da ANSI/IEEE)

- *Precisão simples (32 bits)*
- *Precisão dupla (64 bits)*
- *Precisão estendida (80 bits)*

Números binários em PF (Precisão Simples)



- Mantissa → entende-se que a vírgula binária está à esquerda dos 23 bits. Efetivamente, existem 24 bits na mantissa porque em qualquer número binário o MSB é 1. Portanto, este bit está lá, só não é representado.
- Expoente (polarizado) → É obtido acrescentando-se o valor 127 ao expoente. Sua finalidade é permitir números muito grandes sem a necessidade de um bit de sinal separado para os expoentes.

Exemplo: *Representar o número 1 0110 1001 0001 no formato de ponto flutuante.*

$$1011010010001 = 1{,}0110\ 1001\ 0001 \times 2^{12}$$

- S = 0 (admitindo-se que o número é positivo).
- E = 12 + 127 = 139 → 10001011
- F = 0110 1001 0001

0	10001011	01101001000100000000000

A abordagem geral para determinarmos o valor de um número em ponto flutuante é expressa por:

$$\text{Número} = (-1)^S(1+F)(^{2E-127})$$

Exemplo:

1	10010001	10001110001000000000000

$$\text{Número} = (-1)^1(1{,}10001110001)(2^{145-127})$$
$$= (-1)(1{,}10001110001)(2^{18})$$
$$= -1100011100010000000 = -407{,}688_{10}$$

- Um número no formato de ponto flutuante de 32 bits pode substituir um número binário inteiro de 129 bits.
- Como o expoente determina a posição da vírgula binária, números que contem partes inteira e fracionária podem ser representados.
- Exceções:

  número 0,0 → 0s em todas as posições

  número ∞ → E = 1s e F = 0s (*NaN*)

Exercícios

Tocci – 10ª edição
6.8 a 6.17

Álgebra Booleana

Análise formal para circuitos digitais:

- George Boole (1815-1865) publicou um trabalho intitulado "*Uma investigação das leis do pensamento, sobre as quais são fundadas as teorias matemáticas de lógica e probabilidades*".

- Em 1854, Boole inventou um sistema algébrico de dois valores, agora chamado de *Álgebra Booleana*.

- Usando esta álgebra, podemos formular proposições que são verdadeiras ou falsas, combiná-las para fazer novas proposições e determinar se as novas proposições são verdadeiras ou falsas.

- Claude Shannon foi o primeiro a aplicar o trabalho de Boole na análise e projeto de circuitos lógicos em 1938.

- Praticamente introduziu na área tecnológica o campo da eletrônica digital.

- A Eletrônica Digital emprega em seus sistemas um pequeno grupo de circuitos básicos padronizados conhecidos como "Portas Lógicas", com as quais é possível implementar qualquer expressão gerada pela Álgebra Booleana.

- Usa-se variáveis simbólicas (Ex: A,B,C,...) para representar a condição de um estado lógico, que pode assumir um dos dois valores possíveis: FALSO (0) ou VERDADEIRO (1).

- Álgebra baseada em teoremas e propriedades (axiomas).

Eis algumas definições utilizadas em Álgebra Booleana:

1. Variáveis: É um *símbolo (geralmente uma letra maiúscula em itálico) usado para representar o estado lógico de um sinal (verdadeiro ou falso);*
2. *Complemento: é o inverso de uma variável e é indicado por uma barra sobre a variável;*
3. *Literal: é a variável ou o complemento de uma variável.*

OBS:

- Seus elementos, em princípio, não tem significado numérico.
- Postulados: Se *X* é uma variável booleana, então:

$$\text{Se } X \neq F \Rightarrow X = V$$
$$\text{Se } X \neq F \Rightarrow X = V$$

Operadores da Álgebra Booleana

São definidas algumas operações elementares na álgebra booleana:

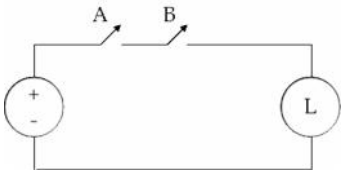
- ✓ Operação "Não" (NOT)
- ✓ Operação "E" (AND)
- ✓ Operação "Ou" (OR)
- ✓ Operação "Ou-Exclusivo" (Exclusive OR ou EXOR)

Operação "Não" (NOT)

- ✓ O operador é uma barra sobre a variável
- ✓ $\bar{0} = 1$
- ✓ $\bar{1} = 0$

Operação "E" (AND)

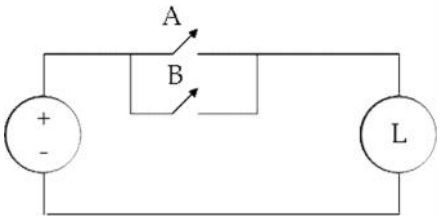
- ✓ O operador é um "ponto" (·)
- ✓ 0 . 0 = 0
- ✓ 0 . 1 = 0
- ✓ 1 . 0 = 0
- ✓ 1 . 1 = 1



Para entendermos melhor a idéia do operador "E", imagine o circuito elétrico na figura. Assuma que chave fechada = 1 e chave aberta = 0, lâmpada acesa = 1 e lâmpada apagada = 0. Em que situações a lâmpada acende?

Operação "Ou" (OR)

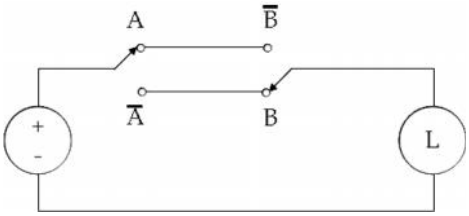
- ✓ O operador é um sinal de positivo (+)
- ✓ 0 + 0 = 0
- ✓ 0 + 1 = 1
- ✓ 1 + 0 = 1
- ✓ 1 + 1 = 1



A lâmpada acenderá quando a chave A OU B estiver fechada (1). Em linguagem lógica, basta que uma das condições seja verdadeira para que a lâmpada acenda (condição verdadeira).

Operação "Ou Exclusivo" (XOR ou EXOR)

- ✓ O operador é um sinal de positivo dentro de um círculo ($\oplus$)
- ✓ $0 \oplus 0 = 0$
- ✓ $0 \oplus 1 = 1$
- ✓ $1 \oplus 0 = 1$
- ✓ $1 \oplus 1 = 0$



Em quais condições a lâmpada Acenderá agora ?

Teoremas da Álgebra Booleana

A álgebra Booleana, assim como a Álgebra Euclidiana, também possui seus próprios teoremas.

Lei Comutativa

A lei comutativa da adição para duas variáveis é escrita da seguinte forma:

$$A + B = B + A$$

A lei comutativa da multiplicação para duas variáveis é a seguinte:

$$A.B = B.A$$

Lei Associativa

A lei associativa da adição escrita para três variáveis é mostrada a seguir:

$$A + (B + C) = (A + B) + C$$

A lei associativa da multiplicação escrita para três variáveis é mostrada a seguir:

$$A.(B.C) = (A.B).C$$

Lei Distributiva

A lei distributiva é mostrada a seguir:

$$A.(B + C) = A.B + A.C$$

$$(A + B).\ (A + C) = A + B.C$$

Lei de Absorção

$$A + A.B = A$$

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

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

Identidades Importantes

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

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

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

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

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

- NOT
  - $\overline{0} = 1$
  - $\overline{1} = 0$
  - $\overline{\overline{A}} = A$
- OR
  - $A + 1 = 1$
  - $A + 0 = A$
  - $A + A = A$
  - $\overline{A} + A = 1$
- AND
  - $A.1 = A$
  - $A.0 = 0$
  - $A.A = A$
  - $A.\overline{A} = 0$

Teoremas da Álgebra Booleana

1. $A + A.B = A$
2. $A.(A + B) = A$
3. $A.B + A.\overline{B} = A$
4. $(A + B).(A + \overline{B}) = A$
5. $A + \overline{A}.B = A + B$
6. $A.(\overline{A} + B) = A.B$
7. $A + B.C = (A + B).(A + C)$
8. $A.(B + C) = A.B + A.C$
9. $A.B + \overline{A}.C = (A + C).(\overline{A} + B)$
10. $(A + B).(\overline{A} + C) = A.C + A.B$
11. $A.B + \overline{A}.C + B.C = A.B + \overline{A}.C$
12. $(A + B).(\overline{A} + C).(B + C) = (A + B).(\overline{A} + C)$

Dualidade

Existe um princípio especial na Álgebra Booleana denominado "princípio da dualidade".

Para uma expressão Booelana qualquer, se trocarmos as operações "E" e as operações "OU" entre si, assim como os valores "0" e "1" entre si, obteremos uma expressão Booleana igualmente válida.

Exemplos:

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

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

$$A + A = A \Leftrightarrow A.A = A$$

$$A + \overline{A} = 1 \Leftrightarrow A.\overline{A} = 0$$

Teoremas de De Morgan

De Morgan, um matemático que conheceu Boole, propôs dois teoremas que representam uma parte importante da Álgebra Boolena.

Primeiro Teorema

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

Segundo Teorema

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

Estes dois teoremas permitem mudar o operador sem alterar a expressão Booleana !

Aplique os Teoremas de De Morgan nas expressões abaixo:

$\overline{A.B.C}$ =

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

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

$\overline{A.\overline{C.D}}$

Funções Booleanas

As funções Booleanas, às vezes também chamadas de funções lógicas, são funções que indicam se uma determinada expressão Booleana é *verdadeira* ou *falsa* para um determinado conjunto específico de valores (também verdadeiros ou falsos) atribuídos às variáveis que compõem esta expressão Booleana.

Exemplo:

f(A,B) = A.B $\rightarrow$ Esta função será verdadeira se, e somente se, A for verdadeira e B for verdadeira. Visto de outra maneira: Esta função será falsa para quaisquer combinações de A e B onde, ao menos uma das variáveis, seja falsa.

Uma função Booleana só pode ser falsa ou verdadeira.

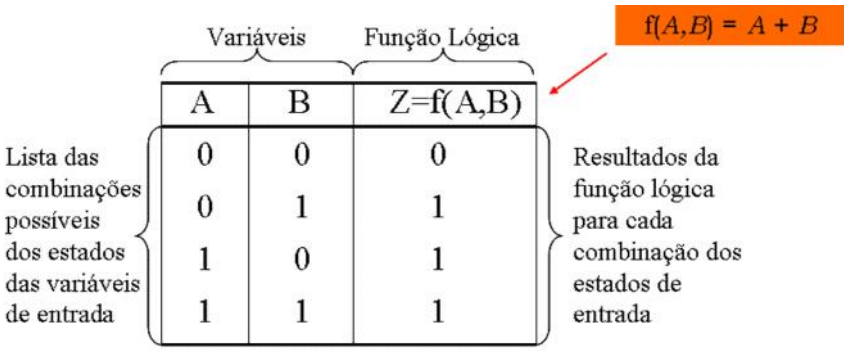
OBS: É muito comum também associarmos aos valores falso e verdadeiro, os valores (0) e (1) – também chamados de níveis lógicos - conforme já vimos, e também os valores ALTO (verdadeiro) e Baixo (falso). No entanto, a correspondência com falso e verdadeiro não é obrigatória. É apenas habitual. Porém, caso a correspondência habitual não seja a escolhida, é obrigatória a indicação da correspondência utilizada.

Em circuitos digitais, é muito comum a associação do valor VERDADEIRO ou nível lógico 1 a um valor de tensão como, por exemplo, +5V. Já para o valor FALSO ou nível lógico 0, é usual associarmos a um valor de tensão de 0V.

Tabela-Verdade

Os resultados de uma função Booleana podem ser expressos numa tabela relacionando todas as combinações possíveis dos valores que suas variáveis podem assumir e seus resultados correspondentes. Esta tabela chama-se de "Tabela-Verdade".

A Figura abaixo apresenta uma Tabela-Verdade para a função Booleana $f(A,B) = A + B$.



A Tabela-Verdade relaciona os resultados (saída) de uma função lógica para todas as combinações possíveis de suas variáveis (entradas)

Exemplo:

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

A	B	f(A,B)
0	0	0
0	1	0
1	0	1
1	1	1

Da Tabela-Verdade, podemos concluir que $f(A,B)$ independe dos valores assumidos por B, pois $f(A,B)$ só é verdadeira quando a variável A for verdadeira. Logo, $f(A,B) = A$, como visto na sub-seção "Identidades Importantes".

Avalie a função Booleana abaixo e complete a Tabela-Verdade:

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

A	B	f(A,B)
0	0	
0	1	
1	0	
1	1	

Portas Lógicas

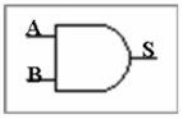
Se relacionarmos as variáveis A e B de uma função booleana f(A,B) como "entradas" e a própria função "f" como saída, podemos imaginar circuitos que implementem as operações básicas da Álgebra Booleana.

Tais circuitos eletrônicos, que não serão estudados neste momento, podem ser simbolizados através do conceito que chamamos de *portas*.

Uma porta representa um operador lógico onde suas entradas correspondem às variáveis lógicas e a sua saída corresponde à própria função.

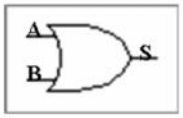
Vejamos as portas lógicas e suas respectivas Tabelas-Verdade.

AND



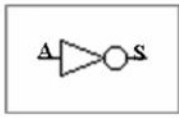
A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

OR



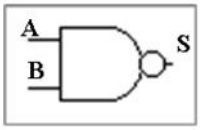
A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

NOT



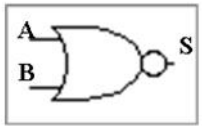
A	S
0	1
1	0

NAND



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

NOR



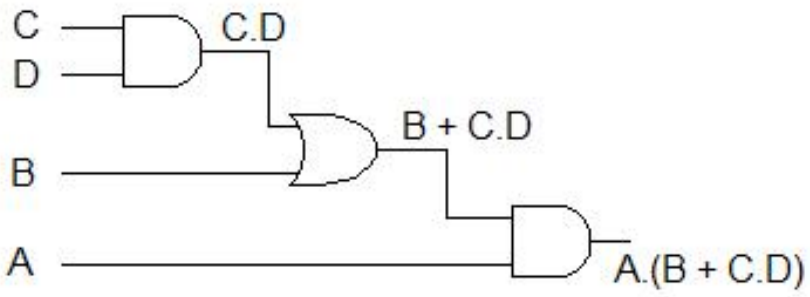
A	B	S
0	0	1
0	1	0
1	0	0
1	1	0

Implementação de Funções Booleanas

A síntese de circuitos digitais é, classicamente, realizada através do uso das portas lógicas a partir de uma Função Booleana.

Vamos tomar como exemplo a implementação da função Booelana abaixo:

$$f(A,B,C,D) = A.(B + C.D)$$



Todo circuito digital (do tipo combinacional) está associado a uma ou mais Funções Booleanas.

Princípio da Equivalência e Suficiência

Qualquer função Booleana pode ser expressa em termos dos operadores AND, OR e NOT

Simplificação de Expressões Booleanas

Ao utilizarmos a Álgebra Booleana, é sempre desejável reduzir uma determinada expressão para a sua forma mais simples (ou mínima), ou transformá-la em um formato mais conveniente a fim de implementar a expressão de maneira mais eficiente.

> Uma expressão booleana simplificada usa a menor quantidade de portas ou chips (circuitos integrados) para implementar uma dada expressão !

Exemplo: Usando as regras da Álgebra Booleana, simplifique a expressão Booleana A.B + A.(B + C) + B.(B + C)

$$AB+A(B+C)+B(B+C) = AB + AB + AC + BB + BC$$
$$= AB + AC + B + BC$$
$$= (A + 1 + C).B + AC$$
$$= B + AC$$

E como podemos fazer para verificar se a simplificação foi feita corretamente?

⇒ Utilize a Tabela-Verdade!

$AB+A(B+C)+B(B+C)$

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

$B + AC$

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

A simplificação de expressões Booleanas nos traz uma série de vantagens em relação à síntese de circuitos:

1. Permite a redução do custo total do circuito, pois requer um número menor de portas lógicas;
2. Um número menor de portas lógicas, por sua vez, nos permite:
    a. Reduzir o consumo de energia
    b. Reduzir a dissipação de calor
    c. Obter Circuitos mais rápidos
    d. Aumentar a confiabilidade do circuito
    e. Reduzir as dimensões físicas do circuito

OBS: Procure sempre minimizar uma expressão Booleana!

Simplifique a expressão $A.\overline{B} + A.\overline{(B + C)} + B.\overline{(B + C)}$ e verifique o seu resultado.

Simplifique as expressões abaixo:

1) $A.B + \overline{B.A.C} =$

2) $A.(\overline{A} + B)$

3) $\overline{A}.(A + B) + \overline{C} + C.B$

4) $\overline{A + A.\overline{B} + C.D}$

5) $\overline{A.[(B + C.(D + \overline{A})]}$

Formas de Expressões Booleanas

Todas as expressões booleanas, independente das suas formas, podem ser convertidas em qualquer uma das duas formas padrão:

- Soma-de-produtos
- Produto-de-somas

Exemplos de soma-de-produtos:

$$A.B + A.B.C$$

$$A.B.C + C.D.E + \overline{B}.\overline{C}.D$$

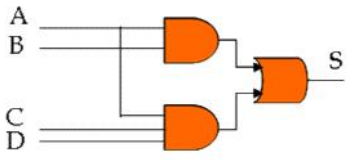
$$\overline{A}.B + \overline{A}.B.\overline{C} + A.C$$

$$A + \overline{A}.\overline{B}.C + B.C.D$$

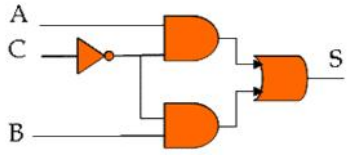
Numa expressão na forma de soma-de-produtos, uma barra não se estende por mais que uma variável. Entretanto, mais de uma variável num termo pode ter uma barra sobre ela !

Conversão de uma expressão geral para a forma de soma-de-produtos:

$$A.(B + C.D) = A.B + A.C.D$$



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



A chamada "forma padrão" de uma soma-de-produtos deve conter todas as variáveis em cada termo da expressão!

AB + ACD → não está na forma padrão

ABCD + ABCD → está na forma padrão

E qual a importância de uma expressão estar na forma padrão?

R: Mapas de Karnaugh mais adiante...

E quando uma expressão não estiver na forma padrão de uma soma-de-produtos, o que fazer ?

1. Multiplique cada termo-produto não padrão por um termo constituído de uma soma de uma variável que não aparece no termo com o seu complemento;
2. Repita o passo anterior até que todos os termos-produto resultantes contenham todas as variáveis do domínio na forma complementada ou não-complementada.

Exemplo:

$$A\bar{B}C + \bar{A}\bar{B} = A\bar{B}C + \bar{A}\bar{B}(C + \bar{C})$$
$$= A\bar{B}C + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C}$$

Produto-de-somas:

$$(\bar{A} + B)(A + \bar{B} + C)$$
$$\bar{A}(A + \bar{B} + C)(\bar{C} + \bar{D} + E)$$

Numa expressão na forma de produto-de-somas, uma única barra sobreposta não pode se estender por mais que uma variável. Entretanto, mais de uma variável num termo pode conter uma barra sobreposta !

$(\bar{A} + B)(\overline{A + B + C})$ → não está na forma de produto-de-somas!

A chamada "forma padrão" de um produto-de-somas deve conter todas as variáveis em cada termo da expressão !

(A+B)(A+B+C) → não está na forma padrão

(A+B+C)(A+B+C) → está na forma padrão

E quando uma expressão não estiver na forma padrão de uma produto-de-somas, o quê fazer ?

1. Acrescente a cada termo-produto não padrão um termo constituído do produto da variável que não aparece pelo complemento dela. Isso resulta em dois termos-soma. Como sabemos, podemos somar 0 com qualquer coisa sem alterar o seu valor;
2. Aplique a regra (A+BC) = (A+B)(A+C);
3. Repita o passo 1 até que todos os termos-soma resultantes contenham todas as variáveis do domínio na forma complementada ou não-complementada.

Exemplo:

$$(A + \overline{B} + C).(\overline{B} + C + \overline{D}).(A + \overline{B} + \overline{C} + D)$$

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

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

Logo,

$$(A + \overline{B} + C + D).(A + \overline{B} + C + \overline{D}).(\overline{A} + \overline{B} + C + \overline{D}).(A + \overline{B} + \overline{C} + D)$$

Observações:

1. Uma expressão na forma de soma-de-produtos é igual a 1 apenas se um ou mais dos termos-produto na expressão for igual a 1 !

2. Uma expressão na forma de produto-de-somas é igual a 0 apenas se um ou mais termos-soma na expressão for igual a 0 !

Conversão de uma Soma-de-Produtos Padrão para um Produto-de-Somas Padrão

1. Determine os termos-produto ausentes na expressão Booleana;

2. Troque os operadores AND por OR em cada um dos termos ausentes e...

3. ...complemente cada uma das variáveis de cada termo, produzindo assim um termo-soma.

Exemplo:

Converta a seguinte expressão de soma-de-produtos padrão para uma expressão equivalente de produto-de-somas padrão:

$$\bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC$$

Como existem 3 variáveis no domínio dessa expressão, existe um total de 8 (2^3) combinações possíveis. Os termos-produto ausentes são:

$\bar{A}.\bar{B}.C \rightarrow A + B + \bar{C}$

$A.\bar{B}.\bar{C} \rightarrow \bar{A} + B + C$

$A.B.\bar{C} \rightarrow \bar{A} + \bar{B} + C$

$(A + B + \bar{C}).(\bar{A} + B + C).(\bar{A} + \bar{B} + C)$ → resultado

Converta a seguinte expressão de soma-de-produtos padrão para uma expressão equivalente de produto-de-somas padrão:

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

Observações

No exemplo abaixo:

$$X = ABC + \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + \bar{A}BC + A\bar{B}C$$

Cada termo produto pode ser chamado também de *mintermo*, e a representação alternativa da função é dada por:

$$X = 111 + 000 + 010 + 011 + 101$$

$$X = m_7 + m_0 + m_2 + m_3 + m_5 = \Sigma\, 0,2,3,5,7$$

No exemplo abaixo:

$$X = (A+\bar{B}+C).(\bar{A}+\bar{B}+\bar{C}).(A+B+\bar{C}).(\bar{A}+\bar{B}+C)$$

Cada termo produto pode ser chamado também de *maxtermo*, e a representação alternativa da função é dada por:

$$X = 010 \,.\, 111 \,.\, 001 \,.\, 110$$

$$X = m_2 \,.\, m_7 \,.\, m_1 \,.\, m_6 = \Pi\, 1,2,6,7$$

Os termos *mintermo* e *maxtermo* ainda são bastante encontrados na literatura sobre circuitos digitais.

Exercícios

Tocci – 10ª edição
3.2 a 3.40

Conversão de Expressões em Tabela-Verdade

Também podemos utilizar a Tabela-Verdade para fazermos conversões entre formas padrão.

Desenvolva uma Tabela-Verdade para a função de Soma-de-Produtos:

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

Existem 3 variáveis no domínio, assim existem oito combinações possíveis de valores binários para das variáveis. Os valores binários que tornam os termos-produto nas expressões iguais a 1 são:

001 + 100 + 111

Para cada um desses valores binários, coloque 1 na coluna de saída como mostrado na TV na outra página. Para cada uma das outras combinações binárias restantes, coloque 0 na coluna de saída da TV.

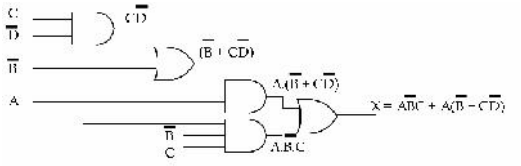
A	B	C	S	
0	0	0	0	
0	0	1	1	→ Termo-produto $\overline{A}\overline{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	1	→ Termo-produto $A\overline{B}\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	1	→ Termo-produto ABC

Determine um termo-soma para cada 0 da Tabela-Verdade

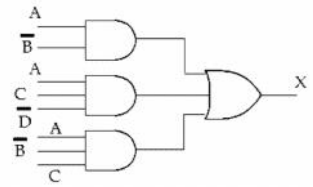
A	B	C	S	
0	0	0	0	→ Termo-soma (A + B + C)
0	0	1	1	
0	1	0	0	→ Termo-soma (A + $\overline{B}$ + C)
0	1	1	0	→ Termo-soma (A + $\overline{B}$ + $\overline{C}$)
1	0	0	1	
1	0	1	0	→ Termo-soma ($\overline{A}$ + B + $\overline{C}$)
1	1	0	0	→ Termo-soma ($\overline{A}$ + $\overline{B}$ + C)
1	1	1	1	

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

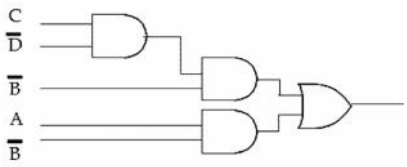
Determine quais dos circuitos lógicos mostrados abaixo são equivalentes:



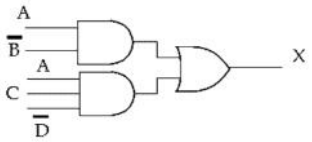
(a)



(b)



(c)



(d)

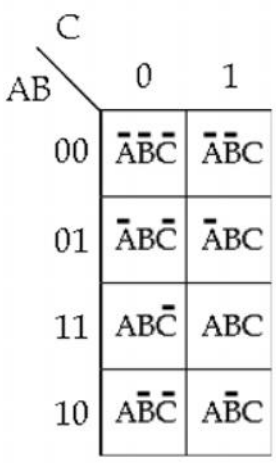
Mapas de Karnaugh

Um mapa de Karnaugh provê um método sistemático para simplificação de expressões Booleanas e, se usado adequadamente, produz uma expressão de soma-de-produtos ou produto-de-somas mínima.

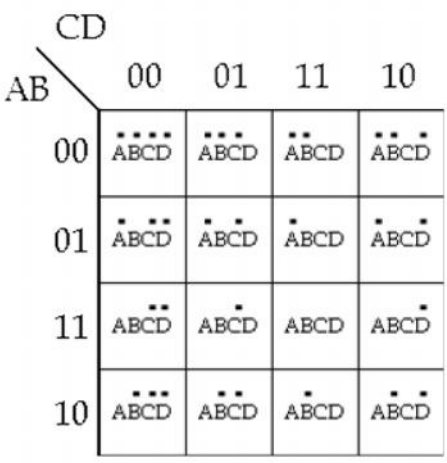
Uma mapa de Karnaugh é similar a uma Tabela-Verdade porque todos os valores possíveis das variáveis de entrada e a saída resultante para cada valor estão presentes no mapa.

O mapa de Karnaugh é um arranjo de "células" no qual cada célula representa um valor binário das variáveis de entrada e o mesmo pode ser utilizado para expressões com duas, três, quatro e cinco variáveis. Para um número maior de células, existe um outro método chamado de "Quine-McClusky"

O número de células numa mapa de Karnaugh é igual ao número de total de combinações possíveis das variáveis de entrada que é igual ao número de linhas da Tabela-Verdade.



Mapa de Karnaugh de 3 variáveis

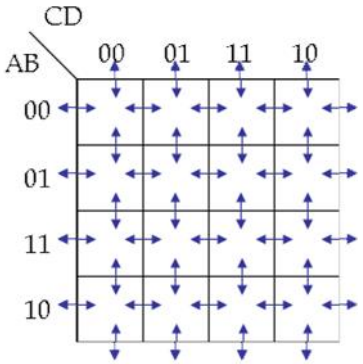


Mapa de Karnaugh de 4 variáveis

Células Adjacentes

As células num mapa de Karnaugh são arranjadas de forma que exista apenas uma mudança simples de variável entre células adjacentes. A adjacência é definida por uma mudança simples de variável.

Fisicamente, cada célula é adjacente a células que estão imediatamente próximas a ela por qualquer um dos seus quatro lados. Uma célula não é adjacente às células que tocam diagonalmente qualquer um dos seus vértices. Além disso, as linhas na linha superior são adjacentes às celulas correspondentes na linha inferior e as células na coluna mais à esquerda são adjacentes às celulas correspondentes na coluna mais a direita. Isso é denominado "**adjacência cilíndrica**".



Células Adjacentes num mapa de Karnaugh são aquelas que diferem uma da outra em apenas uma variável. As seta indicam as células adjacentes.

Mapeando uma expressão padrão de soma-de-produtos

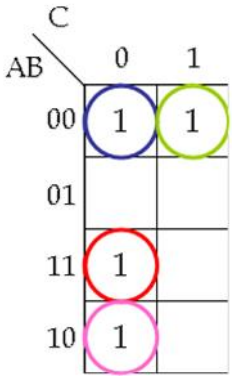
1. Determine o valor binário de cada termo-produto na expressão soma-de-produtos. Após adquirir alguma prática, podemos geralmente fazer a avaliação dos termos mentalmente;

2. À medida que cada termo-produto é avaliado, coloque 1 no mapa de Karnaugh na célula que tem o mesmo valor que o termo-produto.

Exemplo de inserção de uma expressão de soma-de-produtos no mapa:

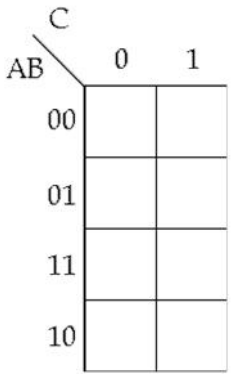
$$\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + AB\bar{C} + A\bar{B}\bar{C}$$

000 001 110 100

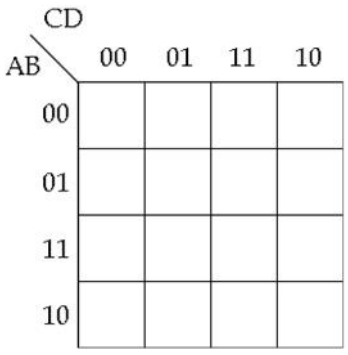


Insira no mapa de Karnaugh a seguinte expressão de soma-de-produtos:

$$\bar{A}\bar{B}C + \bar{A}B\bar{C} + AB\bar{C} + ABC$$



$$\bar{A}\bar{B}CD + \bar{A}B\bar{C}\bar{D} + AB\bar{C}D + ABCD + AB\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + A\bar{B}C\bar{D}$$



Mapeando uma expressão não padrão de Soma-De-Produtos:

Uma expressão booleana tem que estar primeiro na forma padrão antes de usarmos o mapa de Karnaugh. Se uma expressão não estiver na forma padrão, entaõ ela deve ser convertida para a forma padrão usando o procedimento abordado anteriormente ou através de expansão numérica.

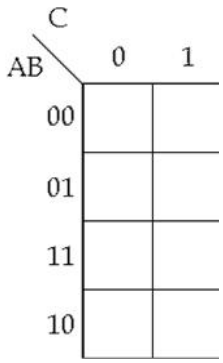
Considere que um dos termos de uma certa expressão booleana de soma-de-produtos de 3 variáveis (A,B,C) é dada apenas pela variável B. As variáveis A e C não fazem parte do termo-produto. Vamos expandí-lo numericamente:

ABC
010
011
110
111

Os 4 números binários resultantes são os valores dos termos $\overline{A}.B.\overline{C}$, $\overline{A}.B.C$, $A.B.\overline{C}$ e $A.B.C$ da soma-de-produtos padrão.

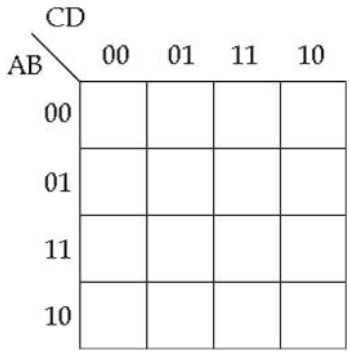
Insira no mapa de Karnaugh a seguinte expressão de soma-de-produtos:

$$\overline{A} + A.\overline{B} + A.B.\overline{C}$$



Insira no mapa de Karnaugh a seguinte expressão de soma-de-produtos:

$$\overline{B}.\overline{C} + A.\overline{B} + A.B.\overline{C} + A.\overline{B}.C.\overline{D} + \overline{A}.\overline{B}.\overline{C}.D + A.\overline{B}.C.D$$



Simplificação via Mapas de Karnaugh de Expressões de Soma-de-Produtos

O processo que resulta numa expressão que contém o menor número de termos possível é denominado ***minimização***.

Podemos fazer grupos de 1s no mapa de Karnaugh de acordo com as regras abaixo, enlaçando aquelas células adjacentes que contêm 1s. A meta é maximizar o tamanho dos grupos e minimizar o número de grupos:

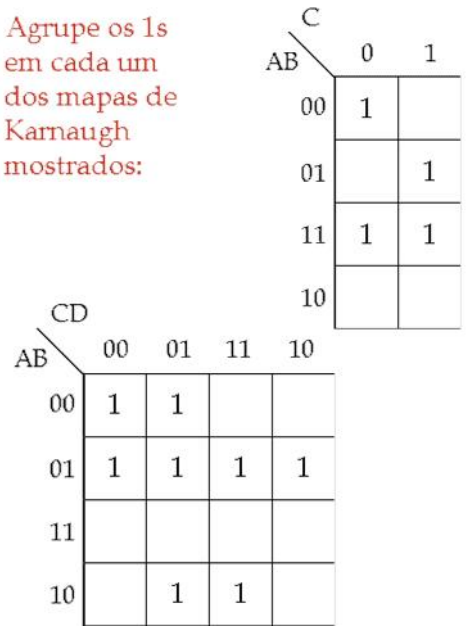
1. Um grupo tem que conter 1, 2, 4, 8 ou 16 células, cujos números são potências inteiras de 2;

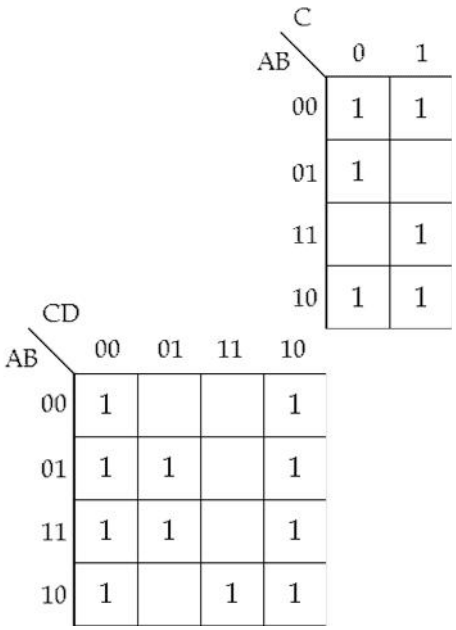
2. Cada célula num grupo tem que ser adjacente a uma ou mais células do mesmo grupo, porém todas as células não tem que ser adjacentes uma da outra;

3. Sempre inclua o maior número possível de 1s num grupo de acordo com a regra 1;

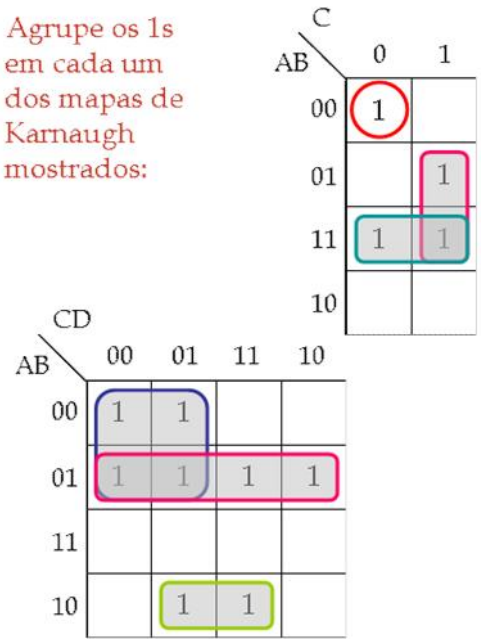
4. Cada 1 no mapa tem que ser incluído em pelo menos um grupo. Os 1s que já fazem parte de um grupo podem ser incluídos num outro grupo enquanto os grupos sobrepostos incluem 1s não comuns.

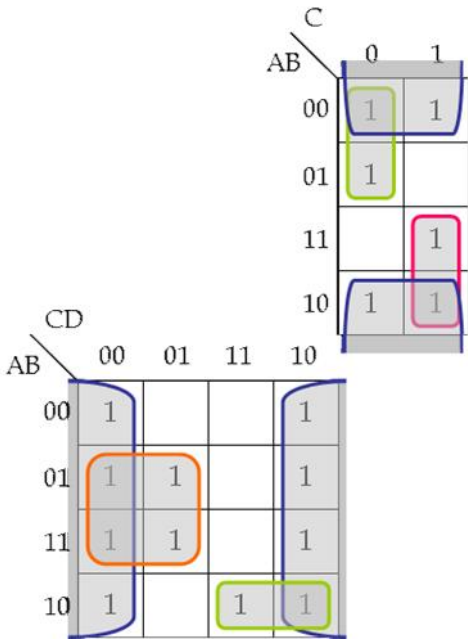
Agrupe os 1s em cada um dos mapas de Karnaugh mostrados:





Agrupe os 1s em cada um dos mapas de Karnaugh mostrados:

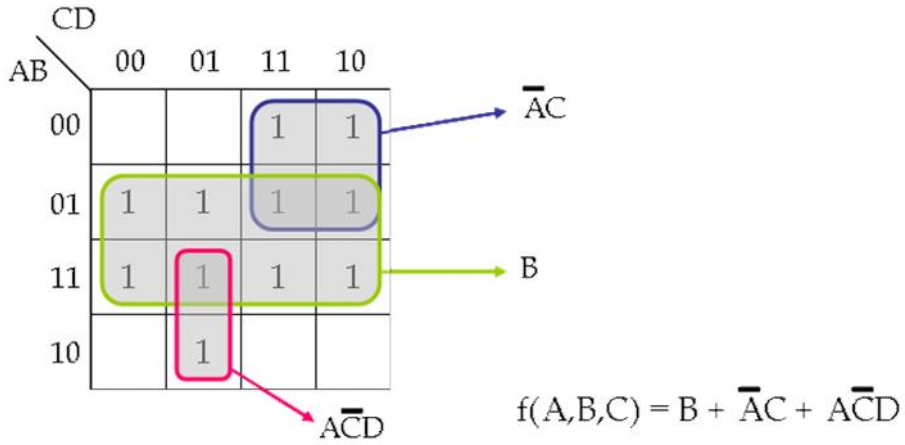




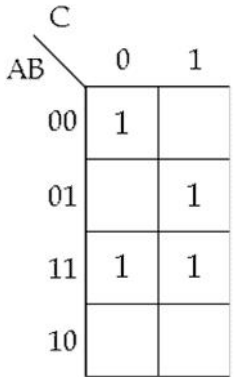
Determinação da expressão Soma-De-Produtos Mínima a partir do mapa

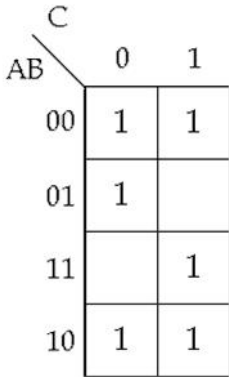
As regras a seguir são aplicadas para determinar os termos-produto mínimos e a expressão Soma-De-Produtos mínima:

1. *Agrupe as células que têm 1s conforme as regras já apresentadas. Cada grupo de células que contém 1s criam um termo-produto composto de todas as variáveis que ocorrem num formato apenas (não complementadas e complementadas) dentro do grupo. Variáveis que ocorrem tanto de forma complementada quanto não complementada dentro do grupo são eliminadas. Essas são denominadas variáveis contraditórias;*
2. *Determine o termo-produto mínimo para cada grupo;*
3. *Quando se obtém todos os termos-produto mínimos a partir do mapa de Karnaugh, eles são "somados" para formar a expressão de soma-de-produtos mínima.*



Encontre as funções booleanas mínimas a partir dos mapas de Karnaugh abaixo:





Encontre as funções booleanas mínimas a partir dos mapas de Karnaugh abaixo:

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

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

O Caso para 5 variáveis

Seja o mapa de Karnaugh dado abaixo. Como agrupar os 1s?

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

E = 0

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

E = 1

Os dois mapas devem ser superpostos para que haja adjacência entre as células deles:

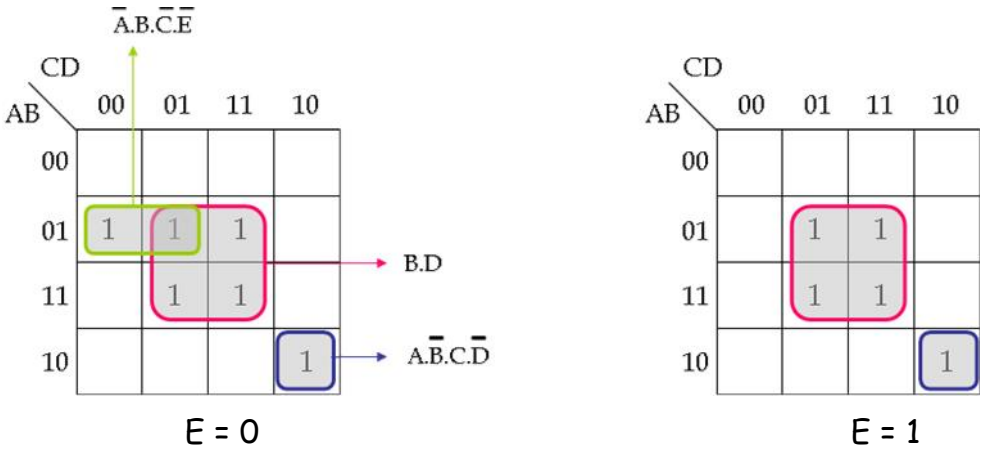
AB \ CD	00	01	11	10
00				
01	1	1	1	
11		1	1	
10				1

E = 0

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

E = 1

A simplificação agora considera os dois mapas simultaneamente.



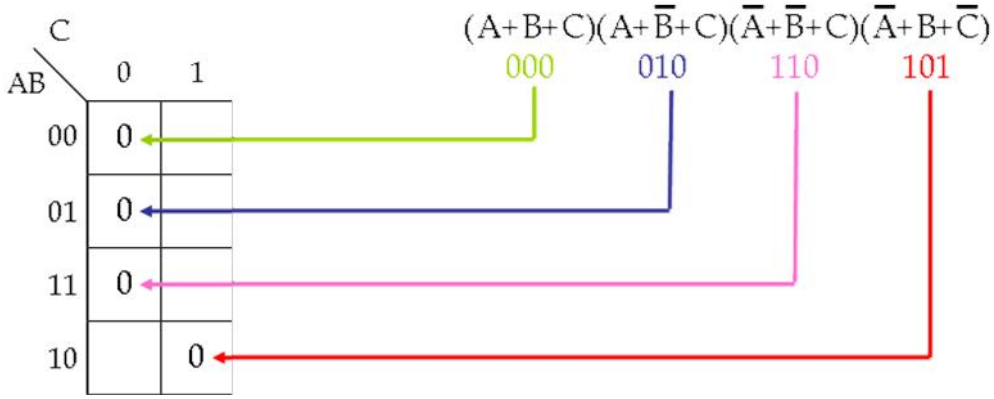
Minimização de Produto-De-Somas via Mapas de Karnaugh

A abordagem é a mesma que a da Soma-de-Produtos, exceto que, com expressões Produto-de-Somas, os "0" representam os termos-soma padrão que são colocados no mapa de Karnaugh em vez dos "1".

Para uma expressão de Produto-de-Somas na forma padrão, um "0" é colocado no mapa de Karnaugh para cada termo-soma na expressão. Cada "0" é colocado na célula correspondente ao valor de termo-soma

Ex: Para o termo ($A+\overline{B}+C$), um "0" é colocado na célula 010.

1. *Determine o valor binário de cada termo-soma na expressão de produto-de-somas padrão. Esse é um valor bináiro que torna o termo igual a 0;*
2. *À medida que cada termo-soma é avaliado, coloque um "0" no mapa de Karnaugh na célula correspondente;*



Preencha o mapa de Karnaugh com a seguinte expressão de Produto-De-Somas padrão:

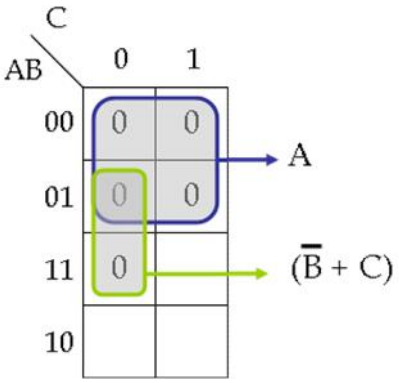
$$(\overline{A}+\overline{B}+C+D).(\overline{A}+B+\overline{C}+\overline{D}).(A+B+\overline{C}+D).(\overline{A}+\overline{B}+\overline{C}+\overline{D}).(A+B+\overline{C}+\overline{D})$$

Minimização de Produto-De-Somas via Mapas de Karnaugh

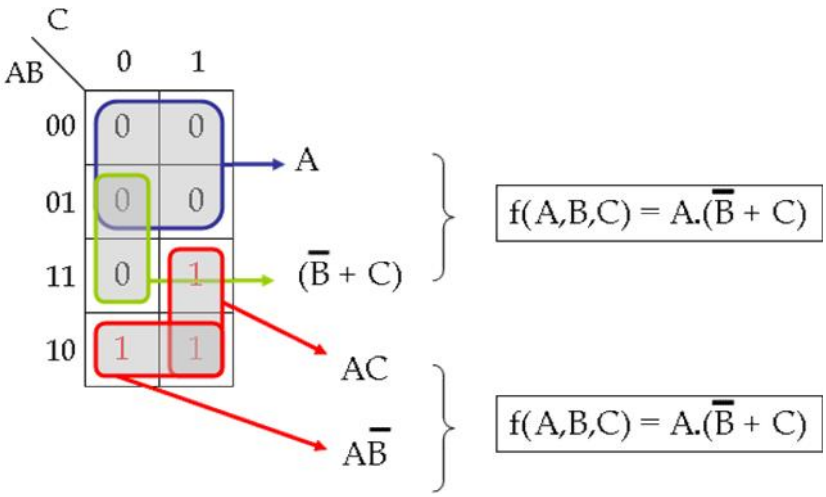
O processo para minimização de uma expressão de produto-de-somas é basicamente o mesmo que para uma expressão de soma-de-produtos, exceto que agrupamos os "0" para produzir termos-soma mínimo em vez de agruparmos os "1" e produzirmos termos-produto mínimos.

Use um Mapa de Karnaugh para minimizar a seguinte expressão de Produto-De-Somas:

$$(A + B + C).\ (A + B + \overline{C}).\ (A + \overline{B} + C).\ (A + \overline{B} + \overline{C})\ (\overline{A} + \overline{B} + C)$$



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



Os agrupamentos de "1" resultam numa expressão de Soma-De-Produtos que é igual ao dual da expressão obtida pelo agrupamento de "0".

Use um mapa de Karnaugh para minimizar a seguinte expressão de Produto-De-Somas padrão:

$$(B+C+D).(A+B+\overline{C}+D).(\overline{A}+B+C+\overline{D}).(A+\overline{B}+C+D).(\overline{A}+\overline{B}+C+D)$$

Conversão de Expressões Booleanas entre Formas Padrão

Quando uma expressão de produto-de-somas é inserida no mapa de Karnaugh, ela pode ser facilmente convertida para a forma de soma-de-produtos equivalente diretamente do mapa. E vice-versa.

Basta lembrar que todas as células que não contém "0" contém "1". A partir destes "1" podemos retirar a expressão na forma padrão de soma-de-produtos.

As células que não contém "1" contém "0". A partir destes "0" podemos retirar a expressão na forma padrão de Produto-De-Somas.

Usando um mapa de Karnaugh, converta a seguinte expressão booleana na forma de produto-de-somas numa expressão padrão de soma-de-produtos:

$(\overline{A}+\overline{B}+C+D)(A+\overline{B}+C+D)(A+B+C+\overline{D})(A+B+\overline{C}+\overline{D})(\overline{A}+B+C+\overline{D})(A+B+\overline{C}+D)$

1100 0100 0001 0011 1001 0010

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

Primeiro, mapeamos os "0" no mapa.

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

Segundo, preenchemos o restante do mapa com "1".

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

Terceiro, extraímos cada termo-produto do mapa e "somamos" os termos para finalmente obtermos a conversão desejada.

$f(A,B,C,D) = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}B\overline{C}\overline{D} + \ldots + ABCD$

Sejam as funções lógicas f1 e f2. Simplifique-as via mapa de Karnaugh e compare as expressões mínimas:

$f_1(A,B,C,D) = (A + B + \overline{C} + D).(A + B + \overline{C} + \overline{D}).(A + \overline{B} + C + D).(\overline{A} + \overline{B} + C + D).(A + B + C + \overline{D}).(\overline{A} + B + C + \overline{D})$

$f_2(A,B,C,D) = \overline{A}.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.\overline{D} + A.B.\overline{C}.\overline{D} + \overline{A}.B.C.D + A.B.C.D + A.\overline{B}.C.D + \overline{A}.B.C.\overline{D} + A.B.C.\overline{D} + A.\overline{B}.C.\overline{D}$

CD / AB	00	01	11	10
00		0	0	0
01	0			
11	0			
10		0		

$f_1(A,B,C,D)$

CD / AB	00	01	11	10
00	1			
01		1	1	1
11		1	1	1
10	1		1	1

$f_2(A,B,C,D)$

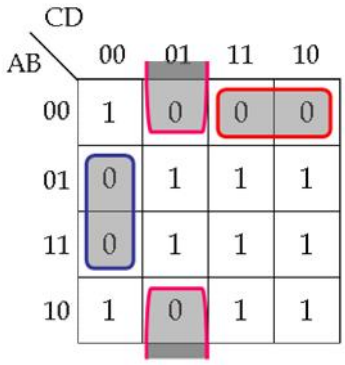
CD / AB	00	01	11	10
00	1	0	0	0
01	0	1	1	1
11	0	1	1	1
10	1	0	1	1

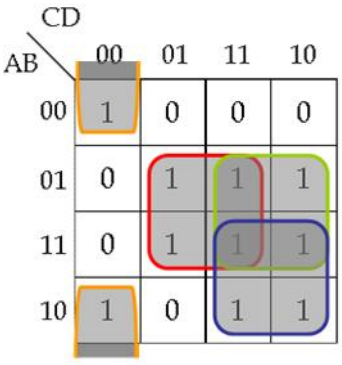
$f_1(A,B,C,D)$

CD / AB	00	01	11	10
00	1	0	0	0
01	0	1	1	1
11	0	1	1	1
10	1	0	1	1

$f_2(A,B,C,D)$

Vamos agora comparar as duas formas mínimas:





$f_1(A,B,C,D) = (A+B+\overline{C})(\overline{B}+C+D)(B+C+\overline{D})$

$f_2(A,B,C,D) = AC+BC+BD+\overline{B}\overline{C}\overline{D}$

As duas funções são duais entre si, pois apresentam a mesma Tabela-Verdade:

A	B	C	D	f_1
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

A	B	C	D	f_2
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Método de Quine-McCluskey

Como podemos observar, a complexidade dos mapas de Veicht-Karnaugh aumentam conforme o número de variáveis aumenta. A partir de 7 variáveis, a simplificação através do uso destes mapas torna-se impraticável.

Tendo em vista que, em muitos casos reais, o número de variáveis extrapola facilmente o número de 6, torna-se importante dispormos de um método mais prático, e de preferência que seja fácil de implementar via software.

Tais métodos existem e chamam-se de "Métodos Tabulares", sendo o método de Quine-McCluskey um dos mais conhecido dentre eles.

OBS: Atualmente, o método implementado em programas de síntese de circuitos digitais é o ESPRESSO (e seus variantes), criado por Robert Brayton na IBM/Berkeley.

O Processo de Agrupamento

Os *mintermos* são, inicialmente, separados de acordo com seus *índices*, onde o índice de um mintermo é o número de variáveis cujo estado seja verdadeiro.

Ex: $\overline{A}.B.\overline{C}.D \rightarrow 0101 \rightarrow 2 \text{ x } 1s \rightarrow$ index = 2

Os mintermos são agrupados conforme seus índices, e em ordem crescente da seguinte forma:

Exemplo 1: Listar os mintermos de quatro variáveis em grupos de mesmo índice.

Mintermo *(binário)*	Número do Mintermo	Índice
0000	0	0
0001, 0010, 0100, 1000	1, 2, 4, 8	1
0011, 0101, 0110, 1010, 1100	3, 5, 6, 9, 10, 12	2
0111, 1011, 1101, 1110	7, 11, 13, 14	3
1111	15	4

Dada a função Booleana expressa por

$$f(A,B,C,D,E,F) = \sum 0,3,6,9,11,14,15,17,21,25,29,30,31$$

Separe os *mintermos* em grupos de acordo com o seu *índice.*

Mintermo	No. Mintermo	Índice

Uma vez que os mintermos tenham sido separados em grupos de acordo com seus índices (mesmo índice), e listados em ordem crescente, a simplificação tabular segue os passos:

Passo 1

a) O número decimal representando cada mintermo em um grupo de mesmo índice é comparado com os números de todos os mintermos do grupo possuindo um índice mais alto;
b) Somente aqueles mintermos possuindo números maiores, e também índices maiores, precisam ser considerados para efeito de comparação. Por exemplo, m_9 (índice 2) não precisa ser comparado com o mintermo m_7 (índice 3), pois 7 é menor que 9 (ver Exemplo 1);
c) Dois mintermos irão combinar-se (como um grupo de 2) se suas representações decimais diferirem de 2^N. O grupo formará então um *implicante reduzido*[1];
d) Cada mintermo que é combinado em um grupo de dois, é então riscado da lista original;
e) Um dado mintermo pode combinar-se com mais de um grupo;
f) Os termos restantes que não formaram grupos são chamados de *implicantes essenciais*;

[1] Para compreender a origem dos termos empregados aqui, veja o livro *Introduction to Switching Theory and Logical Design* – Frederick J. Hill e Gerald R. Peterson – Wiley International, 1968.

Exemplo 2: Dada a função Booleana f(A,B,C,D) = ∑ 1,4,10,11,12,14,15, liste os mintermos em grupos de mesmo índice e aplique o passo 1:

Mintermo *(binário)*	Número do Mintermo	Índice
0001, 0100	1, 4	1
1010, 1100	10, 12	2
1011, 1110	11, 14	3
1111	15	4

Aplicando o Passo 1, temos:

Coluna 1
1
4
10
12
11
14
15

Passo 2:

a) Os implicantes reduzidos que foram produzidos no **Passo 1** são inseridos em uma segunda coluna na forma

$$I(2^N)$$

onde I é o menor número de mintermo do par formado, e 2^N é a potência de 2 pelo qual o mintermo difere;

b) Somente o mintermo de menor número precisa ser reescrito na coluna 2;

c) As entradas da segunda coluna devem ser separadas em grupos de implicantes reduzidos que resultaram dos vários cruzamentos do Passo 1;

Coluna 1	Coluna 2
1	4(8)
4 √	10(1)
10 √	10(4)
12 √	12(2)
11 √	11(4)
14 √	14(1)
15 √	

Passo 3

a) Os implicantes de cada grupo da coluna 2 são comparados com todos os membros do próximo grupo que possuem a mesma potência de 2 entre parênteses;
b) Se os números dos mintermos diferem por uma potência de 2, então os dois implicantes podem ser combinados. Isto corresponde a dois grupos de dois mintermos sendo combinados para formar um grupo de quatro;
c) O novo implicante reduzido formado deve ser reescrito em uma terceira coluna da seguinte forma:

$$I(2^N,2^M)$$

onde I o mintermo de menor número, e 2^N e 2^M são as potências de 2 utilizadas para formar o grupo;
d) Os implicantes utilizados para formar os grupos da coluna 3 devem ser riscados da coluna 2;
e) As entradas da coluna 3 devem ser comparadas de forma a verificar se outros grupos podem ser formados, levando à uma quarta coluna (grupos de 8 mintermos);
f) O processo continua até que nenhum novo grupo possa ser formado.

Coluna 1	Coluna 2	Coluna 3
1	4(8)	10(1,4)
4 √	10(1) √	
10 √	10(4) √	
12 √	12(2)	
11 √	11(4) √	
14 √	14(1) √	
15 √		

OBS: As entradas da última coluna e os implicantes que não foram riscados nas colunas anteriores formam o conjunto chamado de *implicantes prime*, que poderão ser utilizados para representar a função simplificada.

Para convertermos os *implicantes prime* em uma função Booleana reduzida, seguimos os seguintes passos:

a) Os mintermos cujo número aparecem à esquerda do implicante prime são convertidos para sua forma binária;
b) Os termos correspondentes às posições indicadas pelas potências de 2, entre parênteses, são eliminados.

No caso do exemplo 2, os implicantes prime são: 1, 4(8), 12(2) e 10(1,4), reveja as colunas:

Coluna 1	Coluna 2	Coluna 3
1	4(8)	10(1,4)
4 √	10(1) √	
10 √	10(4) √	
12 √	12(2)	
11 √	11(4) √	
14 √	14(1) √	
15 √		

Daí temos:

$1 = 0001 \rightarrow \overline{A}.\overline{B}.\overline{C}.D$

$4(8) = 0100 \rightarrow \overline{A}.B.\overline{C}.\overline{D} \rightarrow B.\overline{C}.\overline{D}$

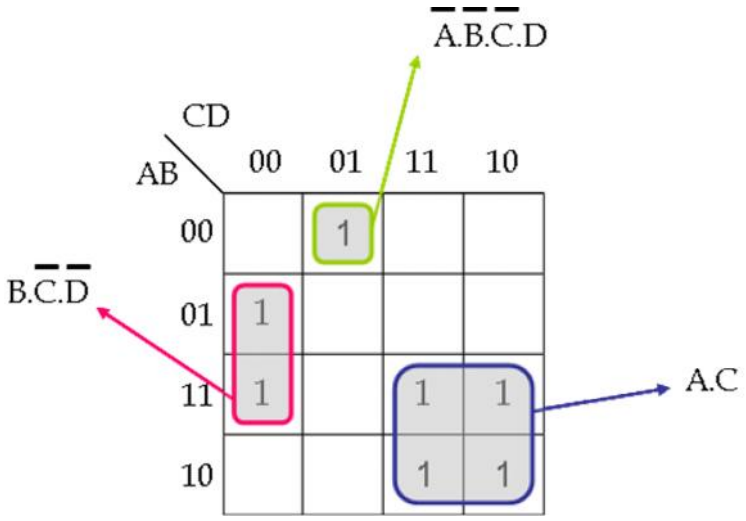
$12(2) = 1100 \rightarrow A.B.\overline{C}.\overline{D} \rightarrow A.B.\overline{D}$

$10(1,4) = 1010 \rightarrow A.\overline{B}.C.\overline{D} \rightarrow A.C$

Logo, a função simplificada toma a seguinte forma:

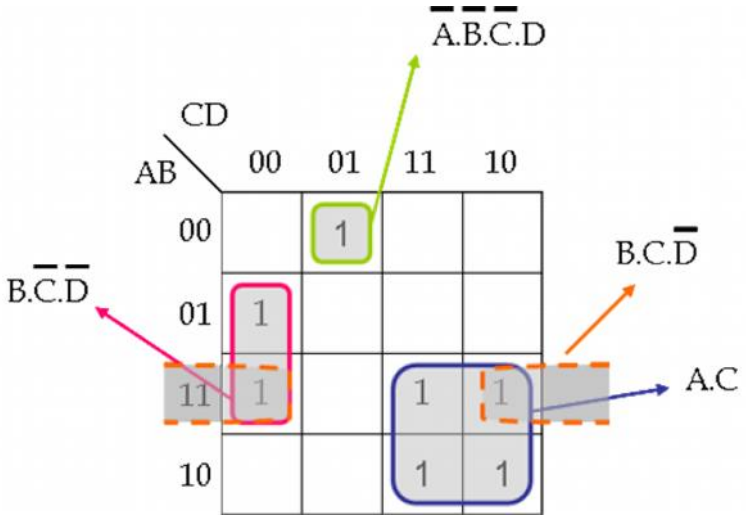
$$f(A,B,C,D) = \overline{A}.\overline{B}.\overline{C}.D + B.\overline{C}.\overline{D} + A.B.\overline{D} + A.C$$

No entanto, aparece um problema: A simplificação através de Mapa de Karnaugh indica que o termo $A.B.\overline{D}$ é desnecessário, conforme podemos observar:



Mas por que isto acontece?

O método de Quine-McCluskey indica todos os implicantes prime possíveis, sem considerar quais são realmente necessários para representar a função simplificada. Veja o Mapa de Karnaugh abaixo:



Nele podemos observar que há uma possibilidade de formação de mais um grupo, indicado pela cor laranja.

Para resolvermos este problema, faz-se necessário identificar os chamados *implicantes prime não essenciais*.

Seleção de Implicantes Prime

Para selecionarmos os melhores implicantes prime, é necessário montarmos uma tabela de implicantes prime.

No exemplo anterior. Os implicantes *prime* encontrados foram:

$$1 = \overline{A}.\overline{B}.\overline{C}.D = a$$

$$4(8) = B.\overline{C}.\overline{D} = b$$

$$12(2) = A.B.\overline{D} = c$$

$$10(1,4) = A.C = d$$

Os implicantes primes são rotulados apenas para facilitar a referência a eles.

	1	4	10	11	12	14	15
a	X						
b		X			X		
c					X	X	
d			X	X		X	X

O X na interseção da primeira linha com a primeira coluna mostra que o implicante *prime* a(m_1) cobre/representa ele mesmo. Já o implicante *prime* b cobre dois mintermos

Podemos notar que os implicantes a, b e d são *essenciais*, pois a cobre m_1, b cobre m_4 e m_{12}, e d cobre o restante dos mintermos, fazendo com que o implicante *prime* c seja desnecessário.

	1	4	10	11	12	14	15
a	Ă=						
b		Ă			X		
c					X	X	
d			Ă	X		X	X

Logo, a forma reduzida da função original é dada por[2]:

$$F = a + b + d = \overline{A}.\overline{B}.\overline{C}.D + B.\overline{C}.\overline{D} + A.C$$

[2] Para maiores detalhes, ver "Fundamentals of Digital Systems Design" de V. Thomas Rhyne, Prentice-Hall, 1973.

Implementando Circuitos Lógicos

Há três formas de implementação de circuitos lógicos atualmente:

1. Utilizando portas lógicas básicas com tecnologia TTL[3] ou CMOS[4];
2. Utilizando dispositivos de lógica programável – PLD[5];
3. Utilizando tecnologia ASIC[6]

Ainda existe uma quarta forma que seria através do emprego de um microprocessador, mas neste caso, não se considera uma implementação real de um circuito e sim uma emulação deste.

A forma mais tradicional, e ainda muito utilizada quando o circuito é muito simples, é a forma utilizando portas lógicas básicas (objeto de estudo da primeira aula de laboratório).

A forma mais moderna de implementação é através do uso de PLDs devido à sua praticidade, baixo custo, confiabilidade e capacidade de integração (reduz as dimensões físicas do circuito final). Tem sido a forma predominante atualmente para implementação de circuitos de média para alta complexidade quando a escala de produção é pequena.

Quando a escala de produção é alta e/ou há necessidade de máximo desempenho do circuito lógico, a técnica conhecida como ASIC entra em cena. Trata-se na verdade da produção de um chip que implementa de forma otimizada o circuito lógico desenhado.

Muitas vezes, um circuito lógico é desenvolvido primeiro (protótipo) utilizando PLDs e, após confirmados alguns parâmetros desejados, um ASIC daquele mesmo circuito é criado.

Por tratar-se de uma técnica otimizada e totalmente específica para aquele circuito lógico, a tecnologia ASIC é cara e só é justificada, muitas vezes, quando o volume de produção é alto. Devido à isto, tem sido muito comum encontrar equipamentos comerciais no mercado utilizando PLDs.

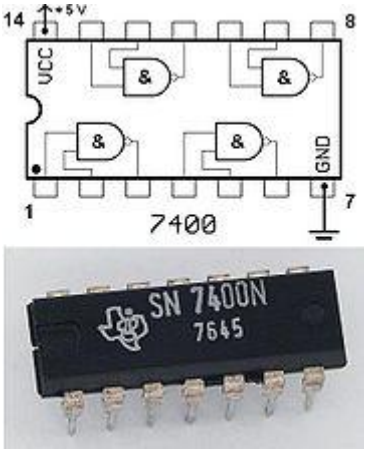
---

[3] Tecnologia mais antiga de fabricação das portas conhecida como *Transistor-Transistor Logic*. Será estudada mais adiante.
[4] Tecnologia mais recente que a TTL (porém, não a mais atual) conhecida como *Complementary Metal-Oxide Semiconductor*. Também será estudada mais adiante.
[5] *Programmable Logic Device*.
[6] *Application-Specific Integrated Circuit*.

Implementação com Portas Lógicas Básicas

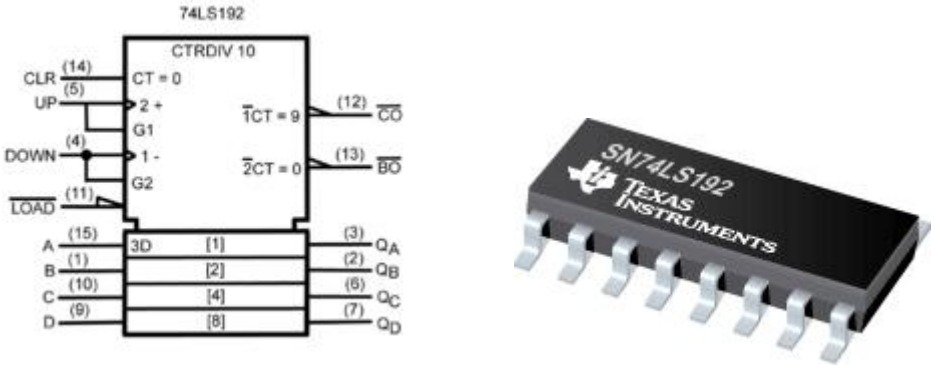
A implementação de circuitos lógicos simples pode ser realizada empregando-se chips contendo as portas lógicas básicas, como o chip 7400 da *Texas Instruments*. Estes chips normalmente dispõem de poucas unidades de portas lógicas, o que faz com que diversos chips sejam necessários para um circuito lógico de complexidade razoável.

Devido ao número reduzido de portas em um único chip, estes são conhecidos com **chips SSI** – *Small Scale of Integration*.



O chip TTL 7400, contendo 4 portas NAND.

Para circuitos básicos como contadores, comparadores, etc..., que normalmente fazem parte de outros circuitos de maior complexidade, o uso de chips SSI tornaria a montagem impraticável. Por sorte, os fabricantes disponibilizam alguns circuitos já prontos em chips conhecidos como **chips MSI** – *Médium Scale of Integration*.



Chip um pouco mais complexo, já contendo um circuito completo.

Implementação em PLDs

O uso de dispositivos de lógica programável acelera o desenvolvimento de circuitos digitais devido à sua grande praticidade, custo e tamanho reduzido.

Os PLDs como, por exemplo, os CPLDs e FPGAs, são dispositivos contendo elementos de circuitos[7]. Cada elemento deste pode assumir, de forma grosseiramente falando, a função de qualquer porta lógica básica através de programação.

Exemplo de CPLD da Altera.

Tais dispositivos são programados através de uma linguagem de descrição de hardware, tais como a Verilog, AHDL e a VHDL, dentre outras.

Existem hoje dois grandes fabricantes de CPLDs e FPGAs no mercado: A Altera e a Xilinx.

Utilizaremos os chips da Altera e o seu programa de desenvolvimento: O Max+Plus II.

No Max+Plus II, há três formas de descrevermos os circuitos. Utilizaremos somente as duas primeiras:

1. Diagrama de circuitos
2. Linguagem de descrição de hardware[8]
3. Diagramas de Tempo

[7] Serão explicados em maiores detalhes no decorrer do curso. Há diferenças consideráveis entre CPLDs e FPGAs.
[8] Usaremos apenas a linguagem VHDL.

A Linguagem VHDL

A linguagem VHDL envolve dois elementos fundamentais:

1. A definição das entradas/saídas do circuito;
2. A definição de como as saídas respondem às entradas (operação).

Vejamos um exemplo de circuito lógico, no caso uma porta AND, descrito em VHDL:

```
ENTITY and_gate IS
PORT( a, b : IN BIT;
          y    : OUT BIT);
END and_gate;

ARCHITECTURE ckt OF and_gate IS
BEGIN
     y <= a AND b;
END ckt;
```

A declaração ENTITY pode ser encarada como uma descrição de bloco, pois o circuito que estamos descrevendo deve estar contido dentro de "algo". O circuito se chama "and_gate".

Em VHDL, a palavra-chave PORT diz ao compilador que estamos definindo entradas e saídas para o circuito. No caso, a e b são entradas (*IN*) do tipo *BIT* (só pode assumir 0 ou 1). A linha contendo END and_gate termina a descrição do bloco.

A declaração ARCHITECTURE é usada para descrever a operação de tudo que está dentro do bloco. O projetista inventa um nome para essa descrição da arquitetura do funcionamento interno do bloco ENTITY. No caso, o nome escolhido foi *ckt*. Aqui, BEGIN e END abrem e fecham a descrição desse bloco, respectivamente.

A saída é composta pelo sinal de *atribuição concorrente* (<=) e pela expressão a AND b. A atribuição concorrente significa que todas as declarações (há somente uma neste exemplo) entre BEGIN e END são avaliadas constante e concorrentemente[9].

[9] Essa é uma importante diferença entre uma linguagem de programação comum e uma linguagem de descrição de hardware.

Em muitos projetos, é preciso definir pontos de sinal "dentro" do bloco de circuito. Esses pontos não são nem entradas nem saídas. São apenas pontos de referência e que, como tal podem ser utilizados para conectar um lugar a outro lugar no circuito, distribuindo o sinal presente naquele ponto a diversos outros locais onde esse sinal seja necessário. Esses pontos são chamados de *nós internos* ou *sinais locais*. Também é interessante adicionar comentários no código em VHDL.

Veja o exemplo abaixo:

```
-- Variáveis intermediárias em VHDL.

ENTITY xyz IS
PORT( a, b, c : IN BIT;   -- define entradas no bloco.
          y    : OUT BIT);  -- define a saída do bloco.
END xyz;

ARCHITECTURE ckt OF xyz IS
  SIGNAL m : BIT;  -- nomeia um sinal intermediário.
BEGIN
     M <= a AND b;  -- gera um termo de produto (interno).
     y <= m OR c;     -- gera soma na saída.
END ckt;
```

Implementação em ASIC

Os chips ASIC também utilizam linguagens de descrição para serem produzidos.

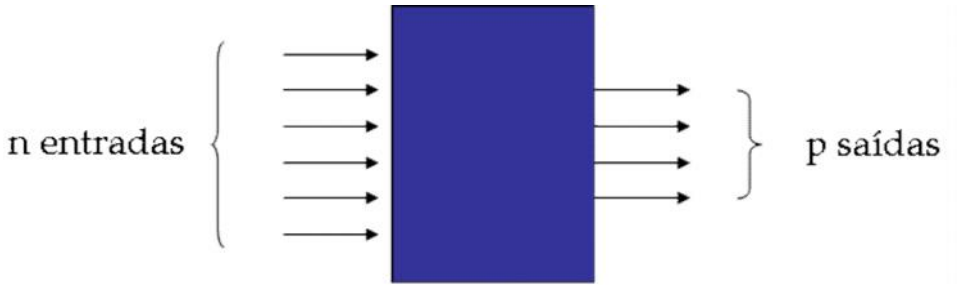
Cartela contendo chips implementados via ASIC.

No entanto, ASICs são circuitos integrados não programáveis e totalmente customizados para uma determinada aplicação.

Implementações modernas de chips via ASIC normalmente incluem processadores de 32-bits (ex: processadores ARM), blocos de memória (ROM, RAM, EEPROM, FLASH) e muitos outros blocos de circuitos. Tais chips são normalmente chamados de SoC (*System-on-Chip*)

Circuitos Combinacionais

Os circuitos combinacionais são aqueles cuja saída(s) só depende(m), em qualquer instante de tempo, das entradas



Projeto de Circuitos Combinacionais

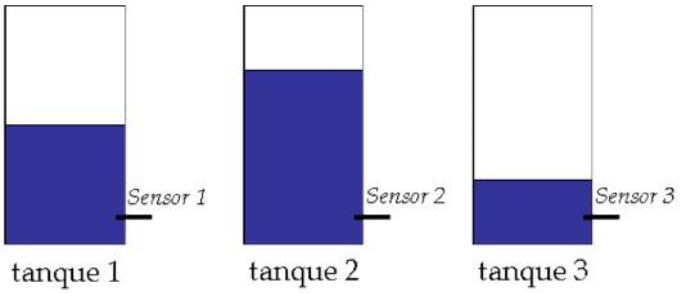
Vamos verificar, através de um exemplo, como se dá o passo-a-passo para o projeto de circuito digital a partir da descrição de um problema prático.

Situação

Em uma certa planta de um processo químico, uma substância na forma líquida é usada num processo industrial. O líquido é armazenado em três tanques diferentes. Um sensor de nível em cada tanque produz uma tensão de nível alto quando o nível do líquido no tanque cai abaixo de um ponto mínimo.

Problema

Projetar um circuito combinacional que monitore o nível do líquido em cada tanque e indique quando o nível em dois tanques quaisquer cair abaixo do ponto especificado.



Projeto

O primeiro passo no problema é identificar as entradas e as saídas.

Através da leitura do problema, é possível dizer que os sensores 1, 2 e 3 são as entradas e a saída é a indicação do nível em dois tanques abaixo do ponto especificado (esta indicação pode ser luminosa, sonora ou até mesmo a ligação automática de uma bomba para encher os tanques novamente).

Entradas: sensor 1, sensor 2, sensor 3
Saída: Indicação de nível baixo

Em seguida, vamos atribuir mnemônicos às entradas e saída para simplificar o trabalho com a Álgebra Booleana:

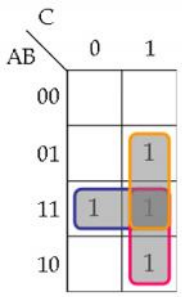
Sensor 1 → A
Sensor 2 → B
Sensor 3 → C
Saída → S

Agora vamos definir o estado lógico das variáveis de entrada e da saída. O problema já nos informa que o sensor nos fornece um nível lógico alto quando o nível do líquido está baixo, mas não fala sobre o nível lógico da saída. Logo, vamos definir:

Nível do líquido baixo → Nível lógico 1
Saída (dois tanques em nível baixo) → Nível lógico 1

Agora que os "significados" dos níveis lógicos foram definidos, podemos criar uma Tabela-Verdade que visa cobrir todas as combinações possíveis de entrada/saída:

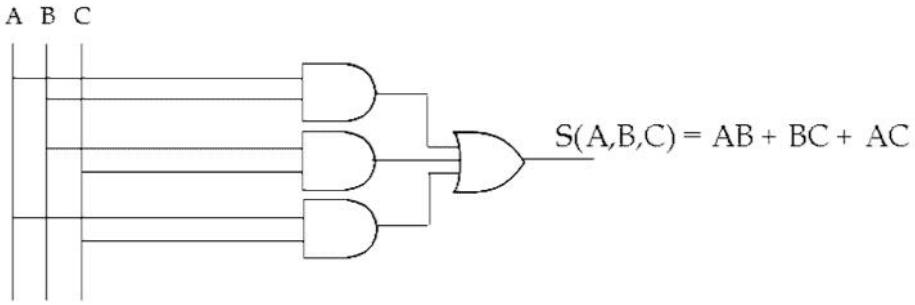
A	B	C	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



$S(A,B,C) = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$

$S(A,B,C) = AB + BC + AC$

Uma vez obtida a expressão mínima, podemos pensar na implementação através das portas lógicas:



Um fazendeiro precisa ir à cidade. No entanto há alguns problemas: O fazendeiro possui algumas galinhas que esperam por uma oportunidade de encontrar a porta do silo aberta e comer todo o seu estoque de grãos. Ele também possui um cão que espera por uma oportunidade de encontrar uma galinha nas imediações dele, e variar seu cardápio "jantando" uma. O fazendeiro também possui um empregado que vive esquecendo de trancar a porta do silo, trancar o canil e trancar o galinheiro. Como um fazendeiro teme que algum problema possa ocorrer durante a sua ausência, ele resolveu contratar você para projetar um alarme que permita alertar o empregado, através de uma sirene, de um perigo iminente. Faça o projeto do alarme. Considere o canil, o galinheiro e o silo, próximos entre si.

Condições " Don't Care"

Algumas vezes surge uma situação na qual uma combinação das variáveis de entrada não é permitida, ou não nos interessa. Como esses "estados" não permitidos nunca ocorrerão, ou já que eles não nos interessam, eles podem ser tratados como termos "don't care" (não importam) em relação aos seus efeitos na saída. Ou seja, para esses termos "don't care" podemos associar 0 ou 1 à saída, já que eles nunca irão ocorrer.

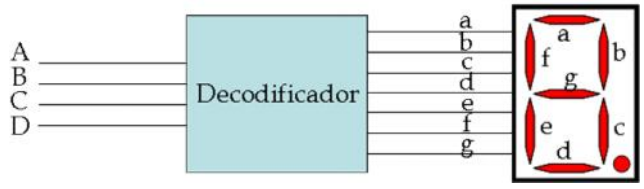
No entanto, podemos obter vantagens no Mapa de Karnaugh quando para cada termo "don't care" inserimos um "X" na célula correspondente e o tratamos como se fosse "1" !

Exemplo: Decodificador Binário-Decimal (Código BCD)

A	B	C	D	S
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Código BCD

Estados "Don't Care"



Para exemplificar o uso dos estados "don't care" no Mapa de Karnaugh. Vamos imaginar um circuito decodificador BCD.

Para simplificar, vamos realizar o circuito parcialmente, uma vez que ele possui sete saídas. Uma para cada segmento do display.

Vamos realizar um circuito nos preocupando apenas para o segmento "a" do display. Os mesmos passos podem ser aplicados para o outros segmentos e, assim, finalizar o circuito completo do decodificador.

Tabela-Verdade

A	B	C	D	a
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Saída para o segmento "a" do display

Estados "Don't Care"

O segmento "a" do display deve acender apenas para os números 0, 2, 3, 5, 6, 7, 8 e 9. Logo, para as entradas correspondentes a estes números, a saída da TV deve ser "1" !

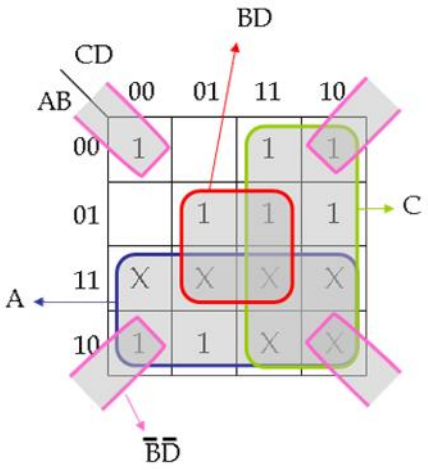
Tabela-Verdade

A	B	C	D	a
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Saída para o segmento "a" do display

Estados "Don't Care"

Mapeando agora a TV no Mapa de Karnaugh, temos:



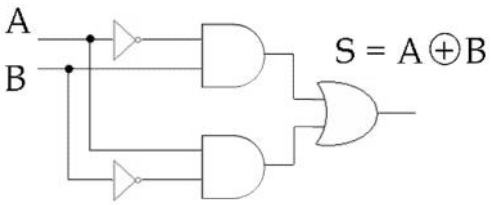
$a(A,B,C,D) = A + C + BD + \overline{B}\overline{D}$

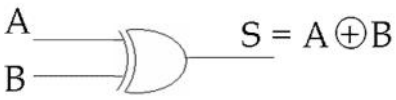
Exemplos de Aplicações de Circuitos Combinacionais

A seguir, vamos analisar alguns circuitos combinacionais básicos (e clássicos) que nos irão nos servir de "blocos de construção" para circuitos ainda mais complexos.

Porta XOR ou EXOR ou OU EXCLUSIVO

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





O "OU EXCLUSIVO" aparece com tanta freqüência na Álgebra Booleana que recebe um status de operador desta, apesar de ser um operador derivado da combinação dos demais operadores (AND, OR e NOT).

Como tal, recebe também um símbolo especial ($\oplus$).

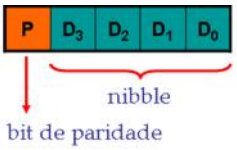
A porta XOR funciona como um detector de diferenças, comparando uma entrada (A) com a outra (B) e indicando se elas são diferentes.

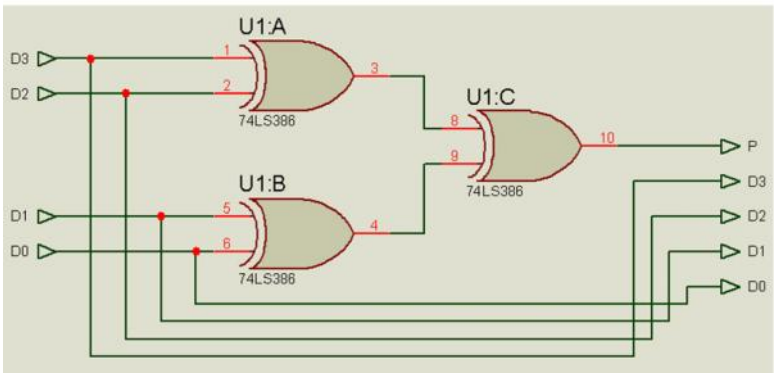
Outra possibilidade bastante interessante é vermos a porta XOR como uma porta que realiza a soma binária de dois bits (A e B). É verdade que, neste caso, ainda temos o problema do famoso "vai-um" (mais conhecido como *carry*) em uma soma que não é gerado, mas veremos a solução para este problema mais a frente.

Gerador de Paridade (par)

Lembrar que:

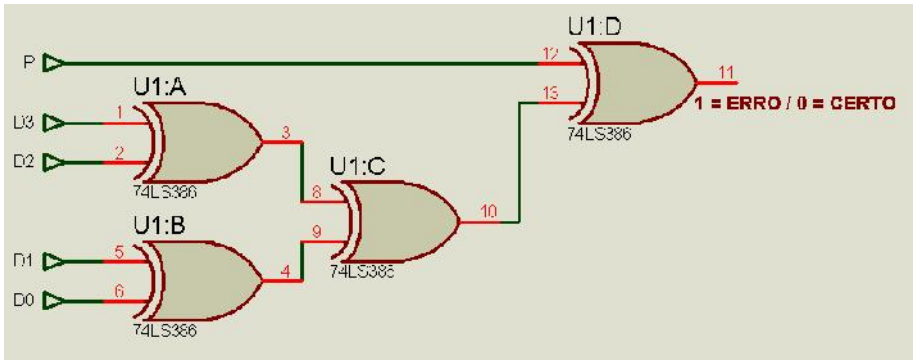
- No esquema de paridade par, deve haver um número par de "1s", incluindo o próprio bit de paridade;
- O bit de paridade, normalmente, antecede o MSB.





Circuito gerador de paridade par.

Detector de Paridade (par)

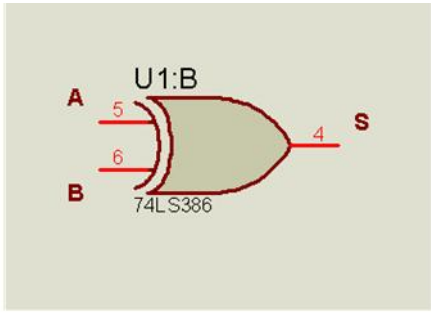


Circuito verificador de paridade par.

Note que, no circuito verificador de paridade, também há uma geração de bit de paridade (circuito formado pelas portas U1:A, U1:B e U1:C), que será utilizado para comparar com o bit de paridade recebido (através da porta U1:D).

Somadores

A porta EXOR pode ser vista como um somador básico. Verifique que, somando-se os valores das colunas A e B, a coluna S apresenta o que seria o resultado da soma binária entre A e B.



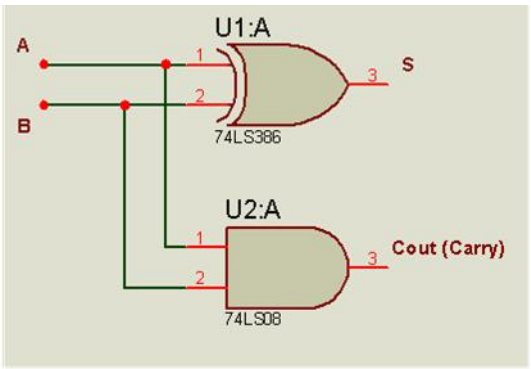
entradas *saída*

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

→ e o vai um? (*carry*)

Aqui, dizemos "seria" devido à um único detalhe: A última linha (1 + 1 = 0) deveria ter também a geração do famoso "vai-um" (também conhecido como *carry* ou *carry-out*).

Como podemos resolver isto? Ora, adicionando-se um circuito lógico que gere este *bit de carry-out* quando as entradas A e B forem "1" simultaneamente. Veja o circuito refeito abaixo:



entradas *saídas*

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

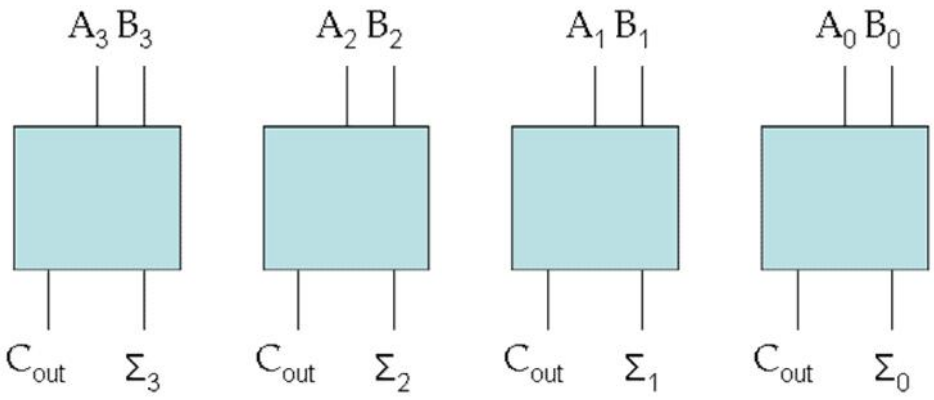
O meio-somador aceita dois dígitos binários em suas entradas e produz dois dígitos binários em suas saídas. Um bit de soma e um bit de carry.

O circuito agora é conhecido como "meio-somador" (*half-adder*).

Vamos dar continuidade na análise do nosso circuito somador e tentar responder à seguinte pergunta:

"Como fazer para somar dois números de 4 bits cada, por exemplo, utilizando nosso meio-somador?"

A figura abaixo mostra 4 meio-somadores, como interligá-los?

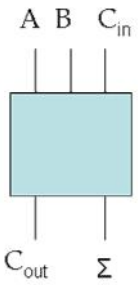


O que fazer com os C_{out} de cada meio-somador se eles são necessários para realizar a soma?

Claramente, nosso circuito ainda não está totalmente apto para resolver o problema apresentado. Então, vamos reprojetá-lo mais uma vez:

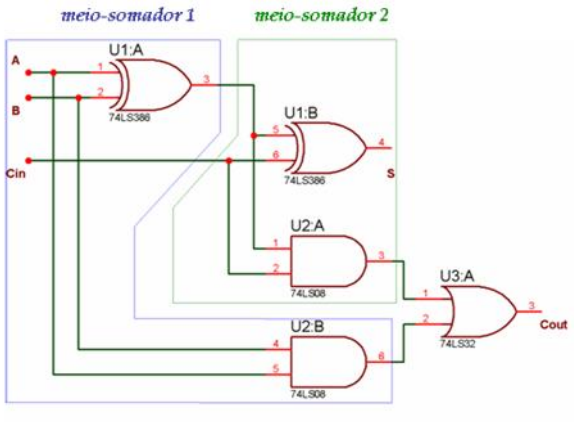
entradas *saídas*

A	B	C_{in}	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



O somador completo aceita dois bits de entradas e um carry de entrada, e gera uma saída de soma e um carry de saída.

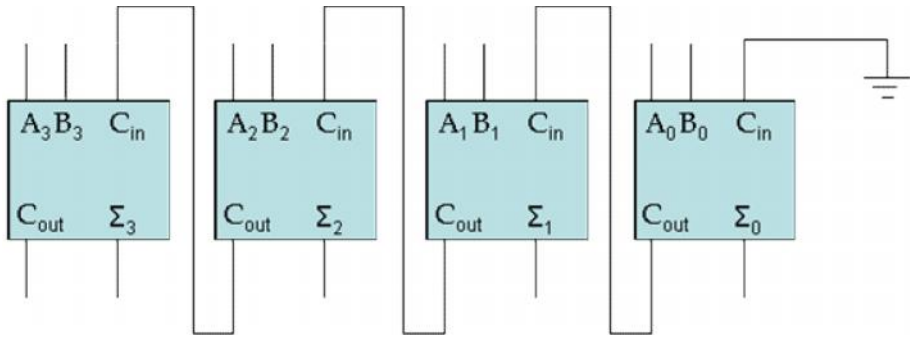
entradas			saídas	
A	B	C_{in}	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Agora nosso circuito passa a contar com uma entrada conhecida como *carry-in*.

Este circuito agora é conhecido como "somador completo" (*full adder*)

Voltando à pergunta de como fazer para somar dois números de 4 bits cada, temos o circuito completo agora.



O circuito acima é conhecido como *Somador Paralelo com Carry Ondulante* e, ainda apresenta mais um problema a ser resolvido: O do tempo necessário para realizar a soma devido à propagação do *carry out*.

A solução definitiva encontra-se no chamado *Somador Paralelo com Carry Antecipado*.

Vamos ver a solução.

A velocidade com a qual uma adição pode ser realizada é limitada pelo tempo necessário para os *carries* se propagarem (ondulação) através de todos os estágios do somador paralelo.

O somador paralelo com carry antecipado antecipa o carry de saída de cada estágio e, com base apenas nas entradas, produz o carry de saída através da geração ou da propagação de carry.

A Geração de Carry

Ocorre quando um carry de saída é produzido (gerado) internamente pelo somador-completo. Um carry é gerado apenas quando os dois bits de entrada são "1s". O carry gerado, Cg, é expresso como uma função AND do dois bits de entrada, A e B.

$$Cg = A.B$$

A Propagação de Carry

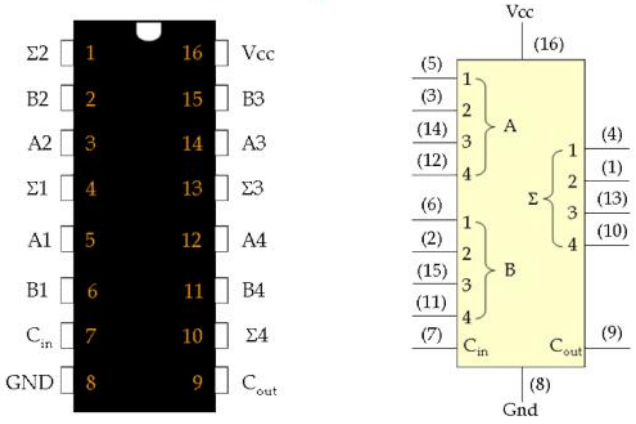
Ocorre quando um carry de entrada "passa" por dentro de um somador completo (ondulação) até se tornar um carry de saída. Um carry de entrada pode ser propagado pelo somador-completo quando um ou os dois bits de entrada forem "1s". O carry propagado, Cp, é expresso por uma função OR entre os bits de entrada.

$$Cp = A + B$$

Logo, o carry antencipado pode ser expresso pela seguinte expressão Booleana:

$$Cout = Cg + Cp.Cin$$

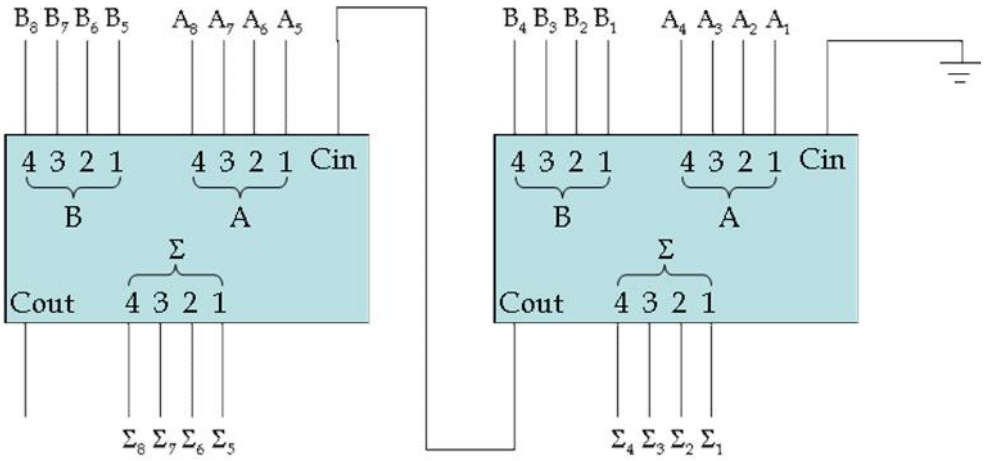
74LS283 (*carry antecipado*)



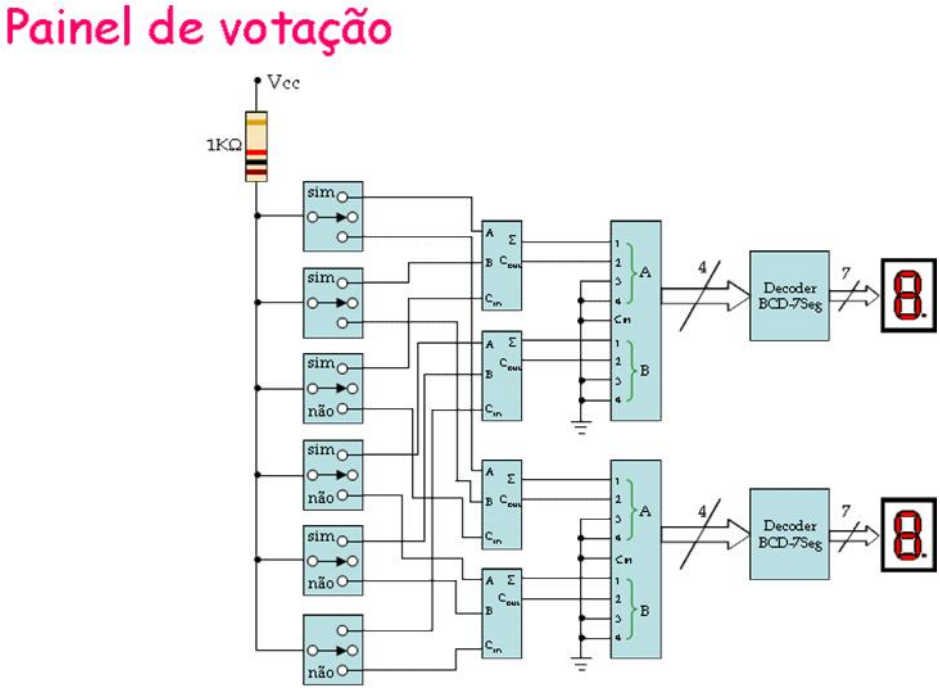
Expansão de um Somador

É muito comum nos depararmos com a seguinte necessidade: Desejamos somar números com mais de 4 bits, mas só dispomos de somadores de 4 bits. Como resolver?

A solução está na expansão de somadores, mostrada na figura abaixo:

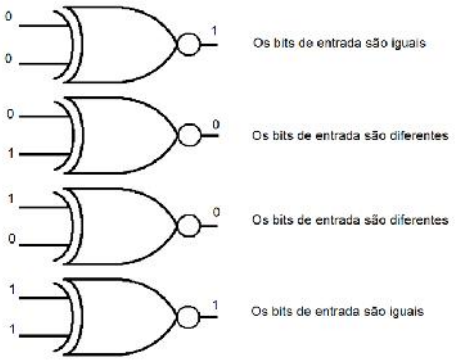


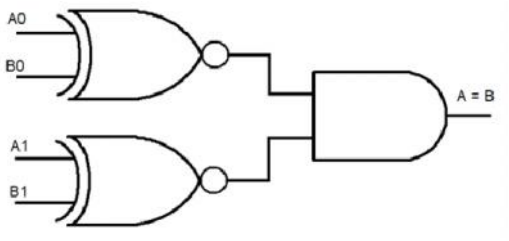
Como exemplo de aplicação de somadores, temos o painel de votação mostrado abaixo:



Comparadores

A Porta EX-NOR como comparador básico



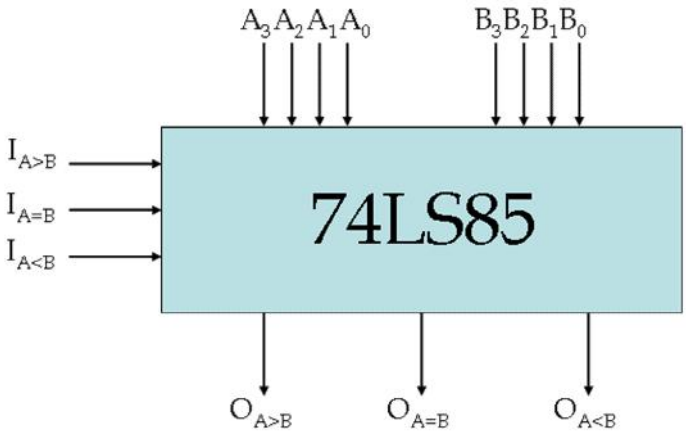


Comparador básico de 2 bits:

- Se saída = 1, A e B são iguais
- Se saída = 0, A e B são diferentes

Comparador de Magnitude

O CI 7485 é um comparador de magnitude que permite a comparação entre dois números de 4 bits. Graças às suas entradas de cascateamento, é possível expandí-lo para números com mais de 4 bits.

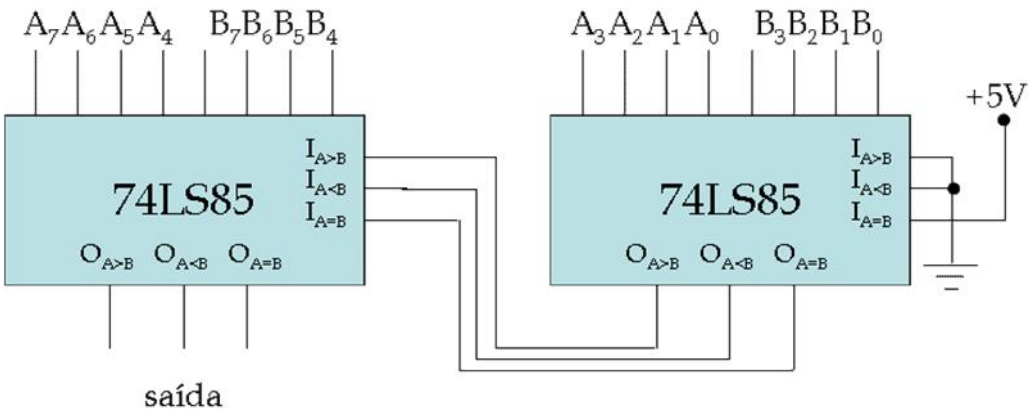


Comparando Entradas				Entradas de Cascateamento			Saídas		
A_3B_3	A_2B_2	A_1B_1	A_0B_0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	$O_{A>B}$	$O_{A<B}$	$O_{A=B}$
$A_3>B_3$	X	X	X	X	X	X	H	L	L
$A_3<B_3$	X	X	X	X	X	X	L	H	L
$A_3=B_3$	$A_2>B_2$	X	X	X	X	X	H	L	L
$A_3=B_3$	$A_2<B_2$	X	X	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1>B_1$	X	X	X	X	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1<B_1$	X	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0>B_0$	X	X	X	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0<B_0$	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	H	L	L	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	H	L	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	X	X	H	L	L	H
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	L	L	H	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	H	H	L	L	L	L

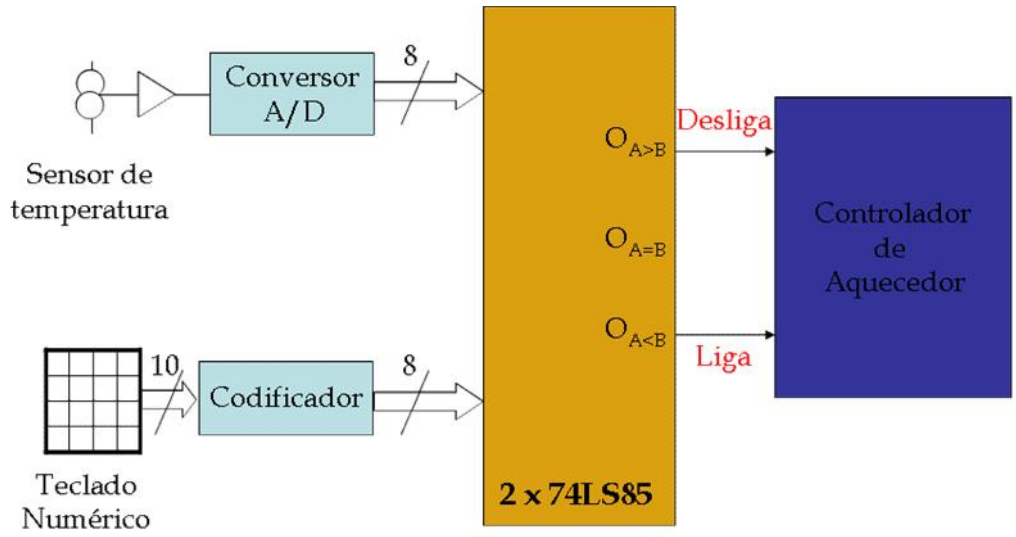
Tabela 3: Funcionamento do Comparador de Magnitude 74LS85.

Expansão do Comparador de Magnitude

As entradas de cascateamento fornecem um meio de expandir a operação de comparação por mais de quatro bits, cascateando dois ou mais comparadores.

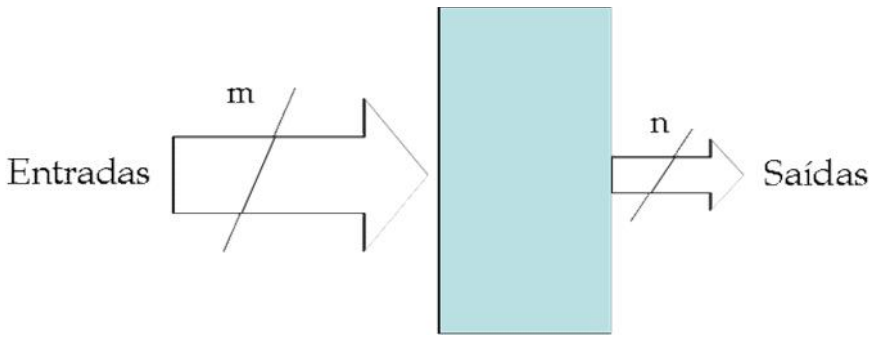


Aplicação: Controlador de temparatura



Codificador Básico

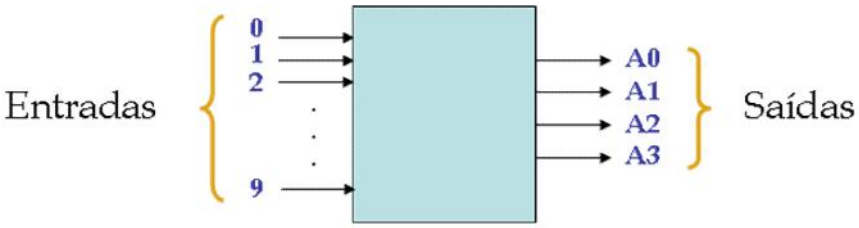
Um codificador é um circuito lógico combinacional que possui ***m*** entradas e que somente uma delas é ativada por vez, produzindo um código de saída de ***n*** bits. O código da saída depende de qual entrada foi acionada.



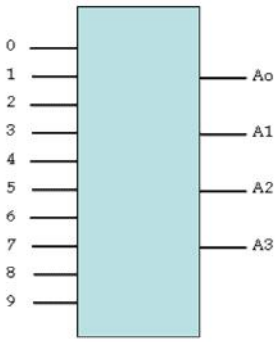
Codificador de Decimal para BCD

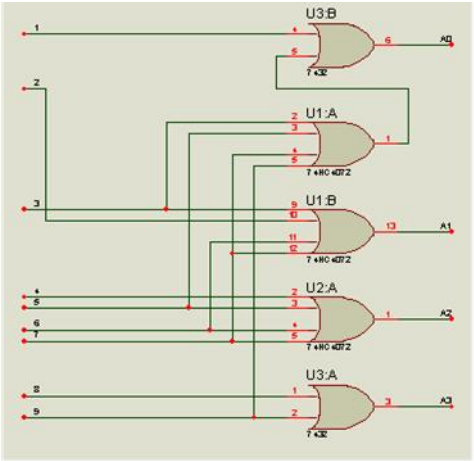
Um codificador que encontra muitas aplicações é o chamado Codificador Decimal BCD.

Este codificador é muito usado nas chamadas interfaces homem-máquina (IHM), como por exemplo, teclados de telefones, calculadoras, computadores, etc...



Dígito Decimal	Código BCD			
	A3	A2	A1	A0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1





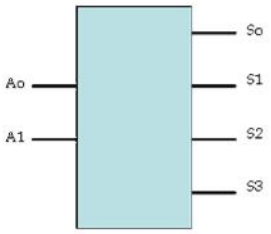
Um "codificador de teclado" é basicamente um circuito lógico que realiza a conversão automática de um decimal (0 a 9) em seu correspondente binário (0000 a 1001).

Decodificadores

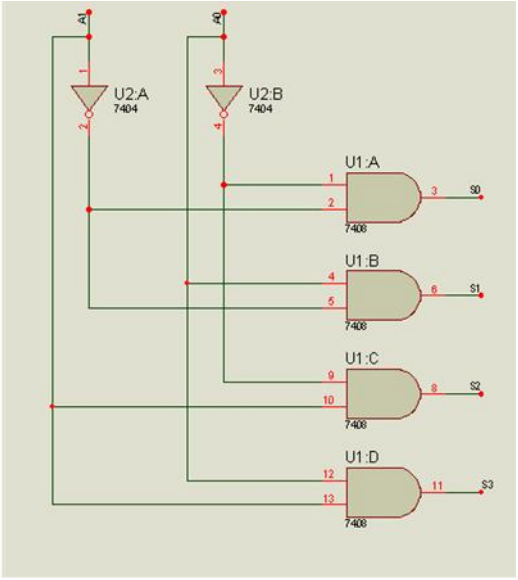
Um decodificador faz o contrário do codificiador. Consiste em um circuito lógico combinacional com ***n*** entradas e ***m*** saídas.

Para cada código binário apresentado à sua entrada, apenas uma das saídas é ativada.

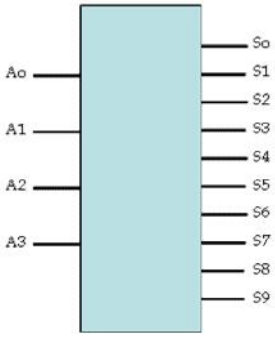
Decodificador de 2 bits



A1	A0	S0	S1	S2	S3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



Decodificador BCD para Decimal

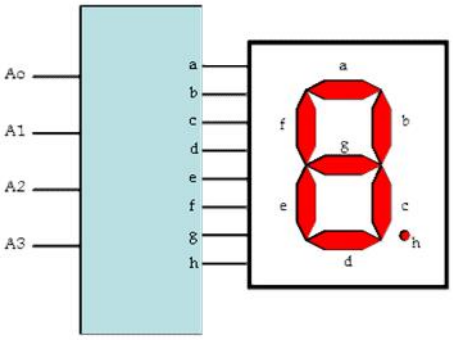


Segue o mesmo princípio que o decodificador de 2 bits, porém, neste caso, para 9 saídas !

Decodificador BCD 7 Segmentos

Um decodificador muito utilizado é o chamado decodificador BCD 7 Segmentos, utilizado para transformar um número binário puro de 4 bits no seu equivalente decimal de 0 a 9.

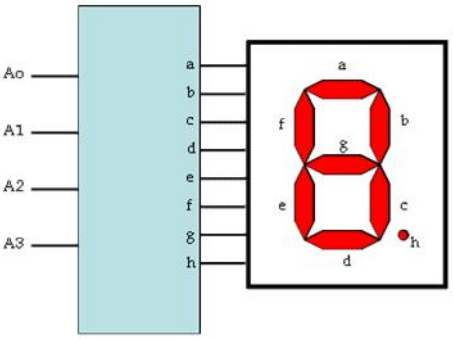
Decodificador BCD para 7 segmentos:



Deverá existir uma expressão booleana para cada um dos sete segmentos !

	Entrada				Segmentos						
	A3	A2	A1	A0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
-	1	0	1	0	x	x	x	x	x	x	x
-	1	0	1	1	x	x	x	x	x	x	x
-	1	1	0	0	x	x	x	x	x	x	x
-	1	1	0	1	x	x	x	x	x	x	x
-	1	1	1	0	x	x	x	x	x	x	x
-	1	1	1	1	x	x	x	x	x	x	x

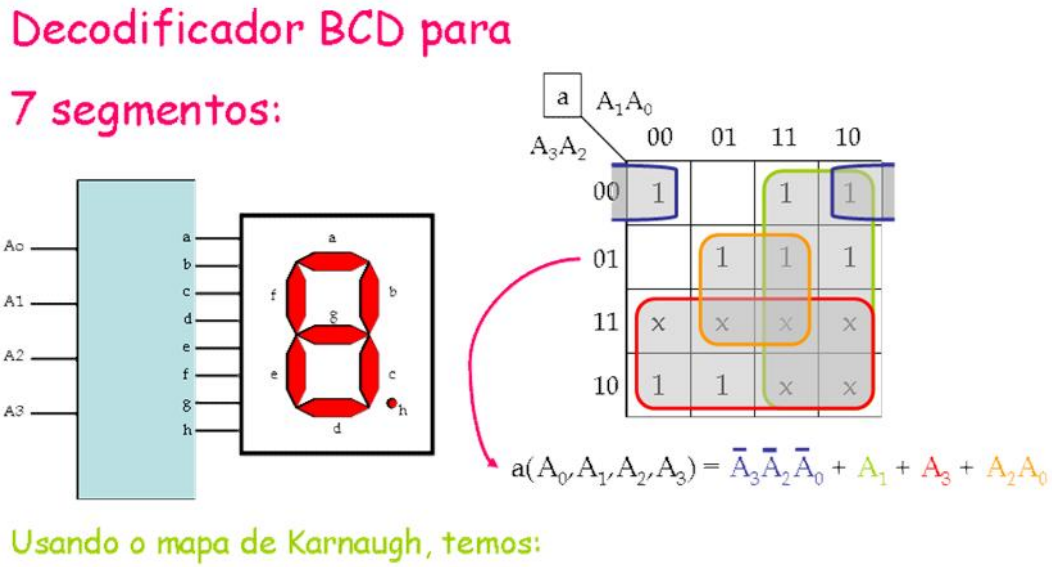
Decodificador BCD para 7 segmentos:



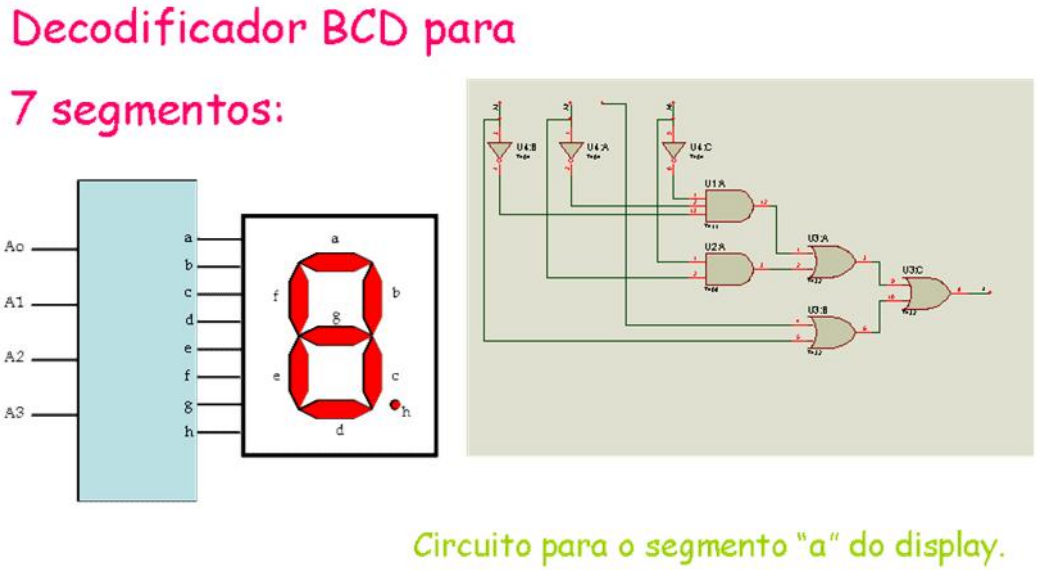
Para simplificar, vamos verificar o projeto do decodificador BCD-7 Segmentos apenas para o segmento "a" do display. Para os demais segmentos, o processo é o mesmo !

	Entrada				Segmentos						
	A3	A2	A1	A0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
-	1	0	1	0	x	x	x	x	x	x	x
-	1	0	1	1	x	x	x	x	x	x	x
-	1	1	0	0	x	x	x	x	x	x	x
-	1	1	0	1	x	x	x	x	x	x	x
-	1	1	1	0	x	x	x	x	x	x	x
-	1	1	1	1	x	x	x	x	x	x	x

Preenchendo o mapa de Karnaugh e simplificando, temos os seguinte *mintermos*:



A implementação do circuito responsável apenas pelo funcionamento do segmento a do display pode ser visto na Figura abaixo:



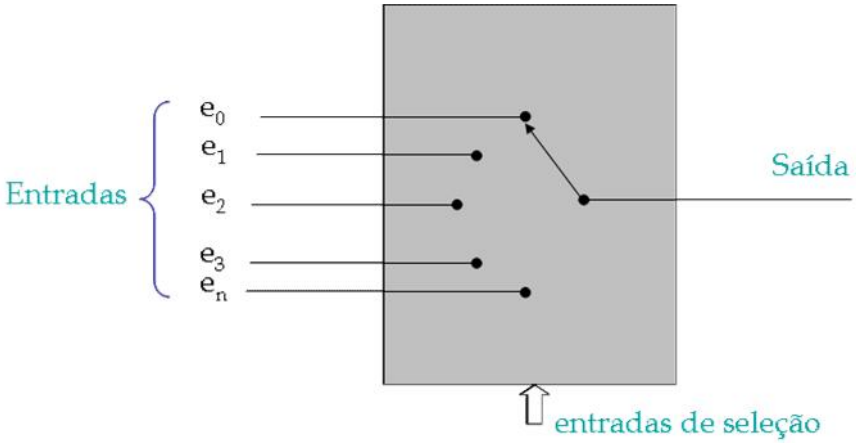
Para o decodificador BCD 7 Segmentos completo, haverá um circuito parecido como o da Figura acima para cada um dos segmentos restantes.

Felizmente, existem circuitos integrados prontos para serem utilizados como, por exemplo, o chip 7448 (tecnologia TTL) ou o 4511 (tecnologia CMOS), dentre outros.

Multiplexadores

Os multiplexadores são circuitos lógico combinacionais que funcionam como seletores de entrada. Também é chamado de MUX ou Multiplex (no jargão de telefonia, onde é muito utilizado).

Um sistema de som moderno pode ter uma chave que seleciona música de uma das quatro fontes: CD, rádio, USB ou entrada auxiliar. Essa chave seleciona um dos sinais eletrônicos dessas quatro fontes e o envia para o amplificador de potência e alto-falantes.

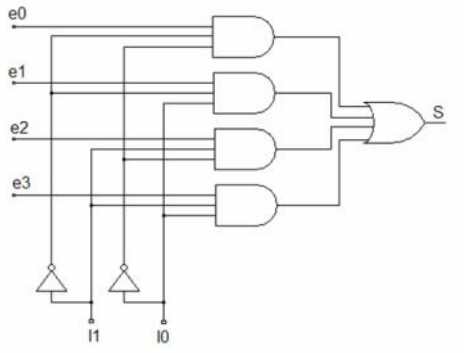


Também podem ser vistos como seletores de dados.

seleção | saída

I_1	I_0	S
0	0	e_0
0	1	e_1
1	0	e_2
1	1	e_3

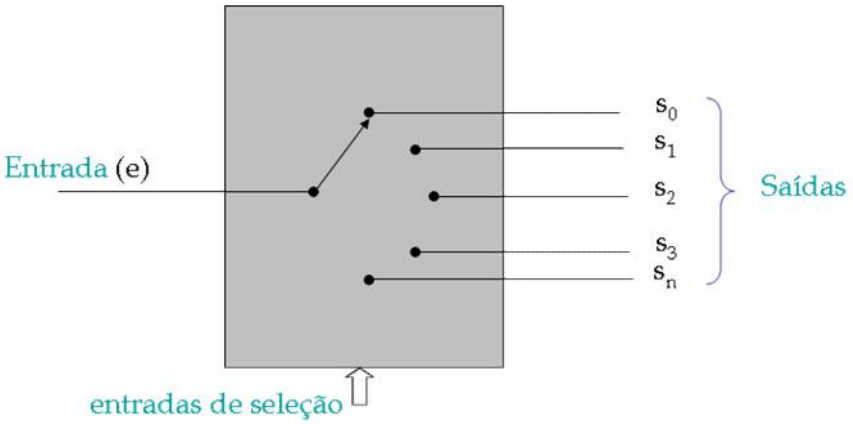
$$S = e_0.\bar{I}_1.\bar{I}_0 + e_1.\bar{I}_1.I_0 + e_2.I_1.\bar{I}_0 + e_3.I_1.I_0$$



Demultiplexadores

Um demultiplexador é um circuito lógico combinacional que faz exatamente o oposto do multiplexador. Também é conhecido com DEMUX ou demultiplex.

Um sistema de som moderno pode ter uma chave que seleciona música de uma das quatro fontes: CD, rádio, USB ou entrada auxiliar. Essa chave seleciona um dos sinais eletrônicos dessas quatro fontes e o envia para o amplificador de potência e alto-falantes.



Também pode ser visto como um distribuidor de dados.

seleção saídas

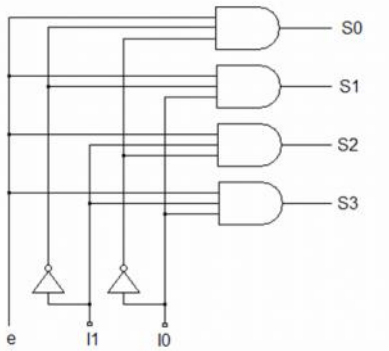
I_1	I_0	S_3	S_2	S_1	S_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

$S_0 = e.\overline{I}_1.\overline{I}_0$

$S_1 = e.\overline{I}_1.I_0$

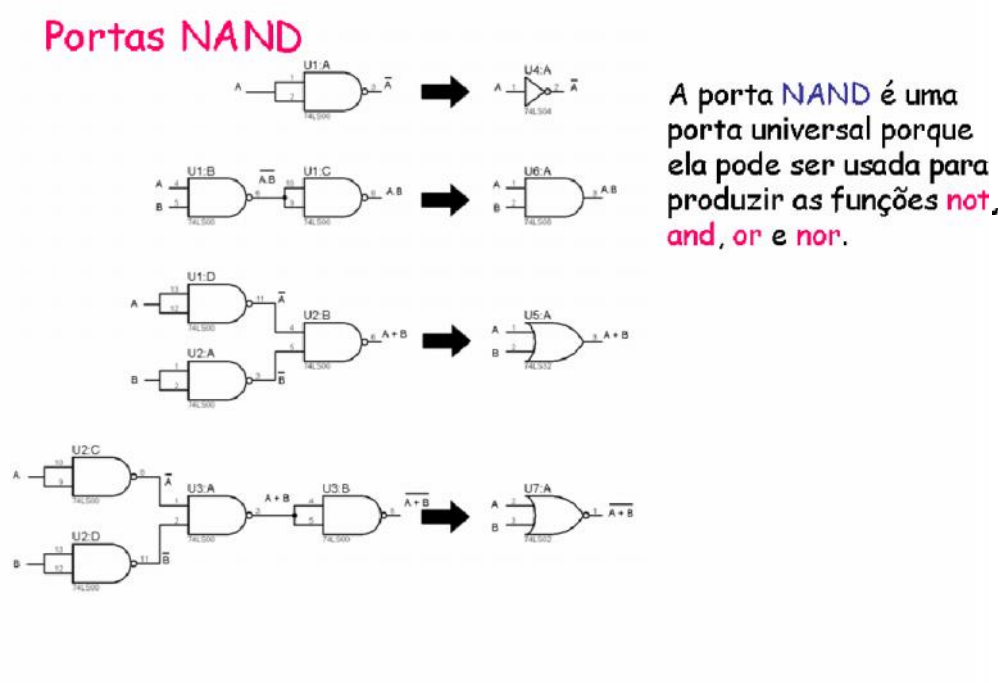
$S_2 = e.I_1.\overline{I}_0$

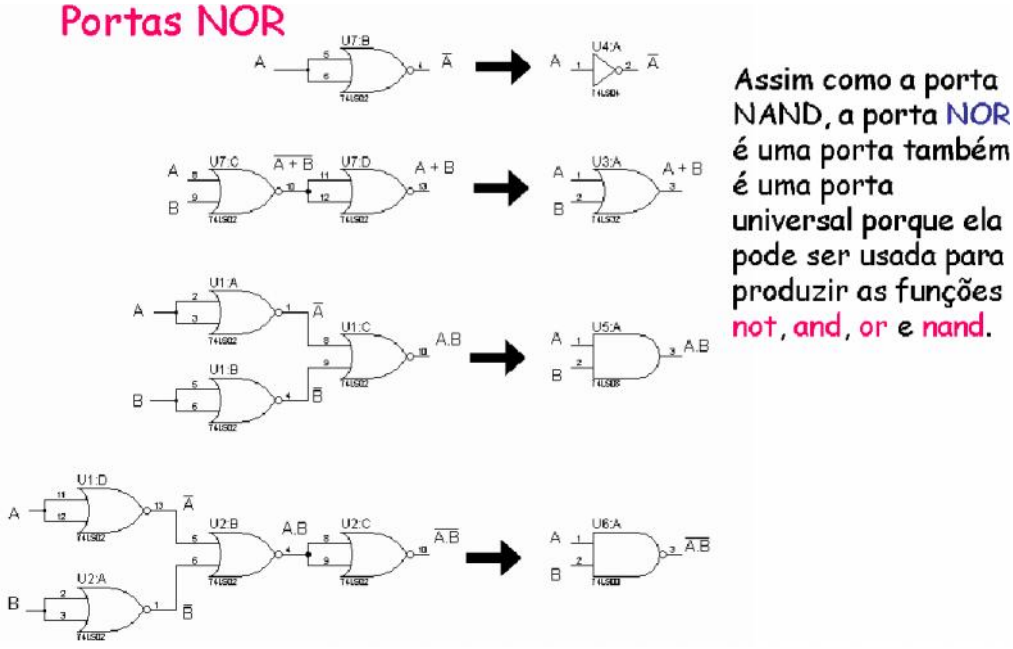
$S_3 = e.I_1.I_0$



Em telefonia, e também em comunicação de dados, é muito comum o emprego do par mux + demux.

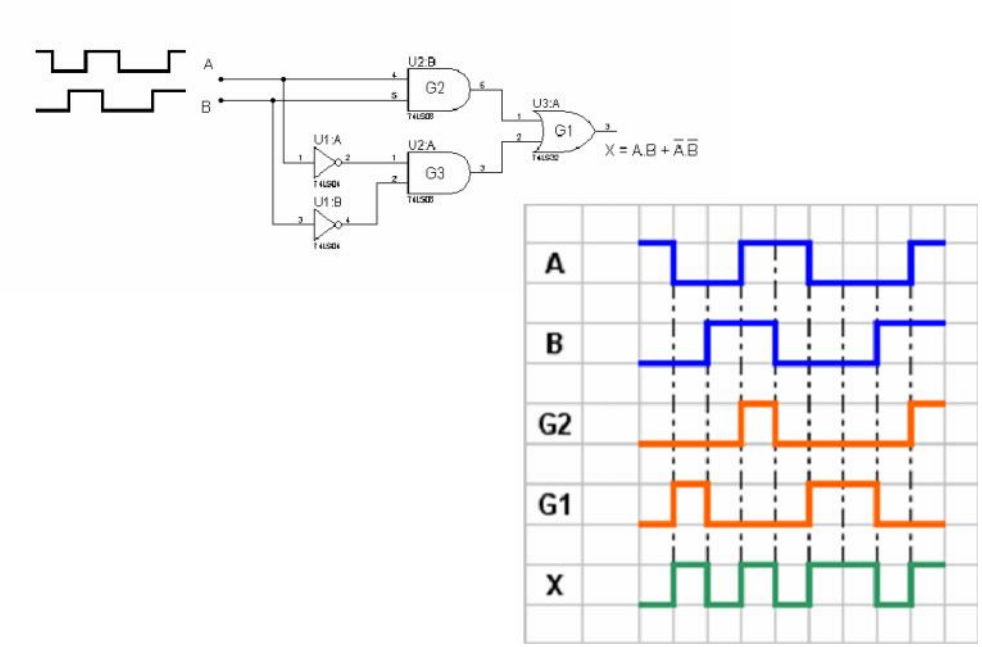
Equivalência de Portas





Formas de Onda Digitais na Entrada

É muito comum analisarmos o funcionamento de um circuito lógico através da aplicação de formas de onda quadrada em suas entradas e verificar a(s) saída(s).



Se as saídas não forem condizentes com as entradas aplicadas e a função Booleana do circuito, então há uma indicação de funcionamento anormal deste.

Características Básicas de Circuitos Integrados Digitais

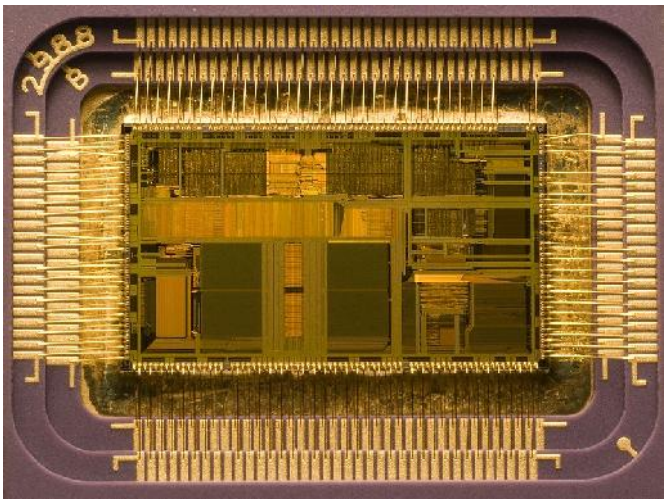
Os circuitos integrados, também chamados de CIs ou Chips, podem ser classificados de diferentes formas, sendo as três principais:

1. Quanto à sua escala de integração, ou seja, a ordem de grandeza do número de portas lógicas equivalentes que se consegue fabricar no CI;
2. Quanto à sua família;
3. E mais modernamente, quanto à espessura de fabricação de suas camadas (gerações).

Quanto à sua escala de integração

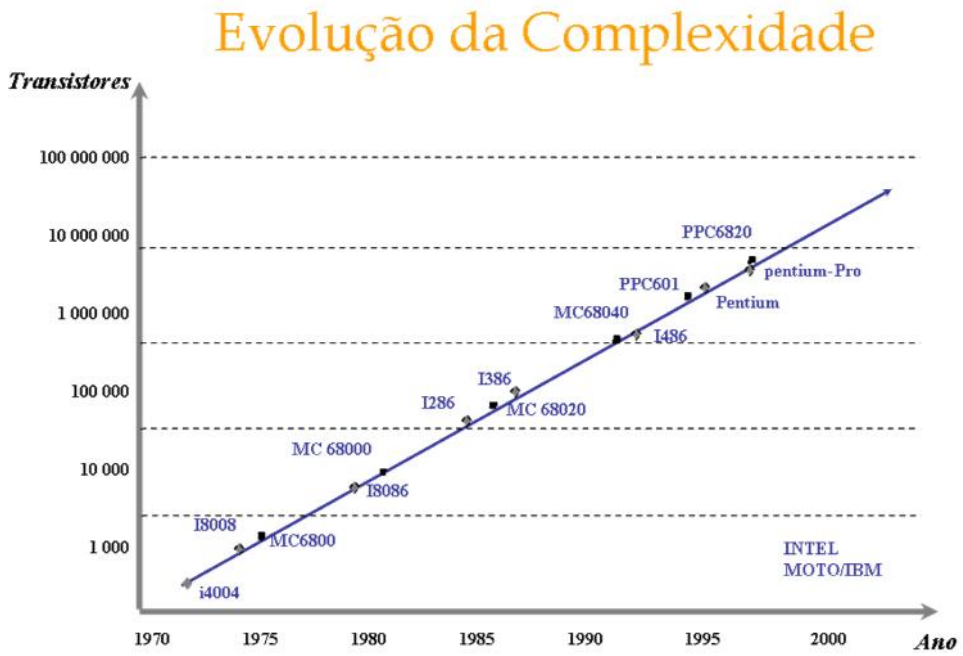
COMPLEXIDADE	SIGLA	PORTAS PO CI
Integração em pequena escala	SSI	< 12
Integração em média escala	MSI	12 a 99
Integração em grande escala	LSI	100 a 9.999
Integração em escala muito grande	VLSI	10.000 a 99.999
Integração em escala ultragrande	ULSI	100.000 a 999.999
Integração em escala GIGA	GSI	> 1.000.000

A figura abaixo mostra um exemplo de CI VLSI. No caso, um microprocessador 486DX2 da Intel Corporation.



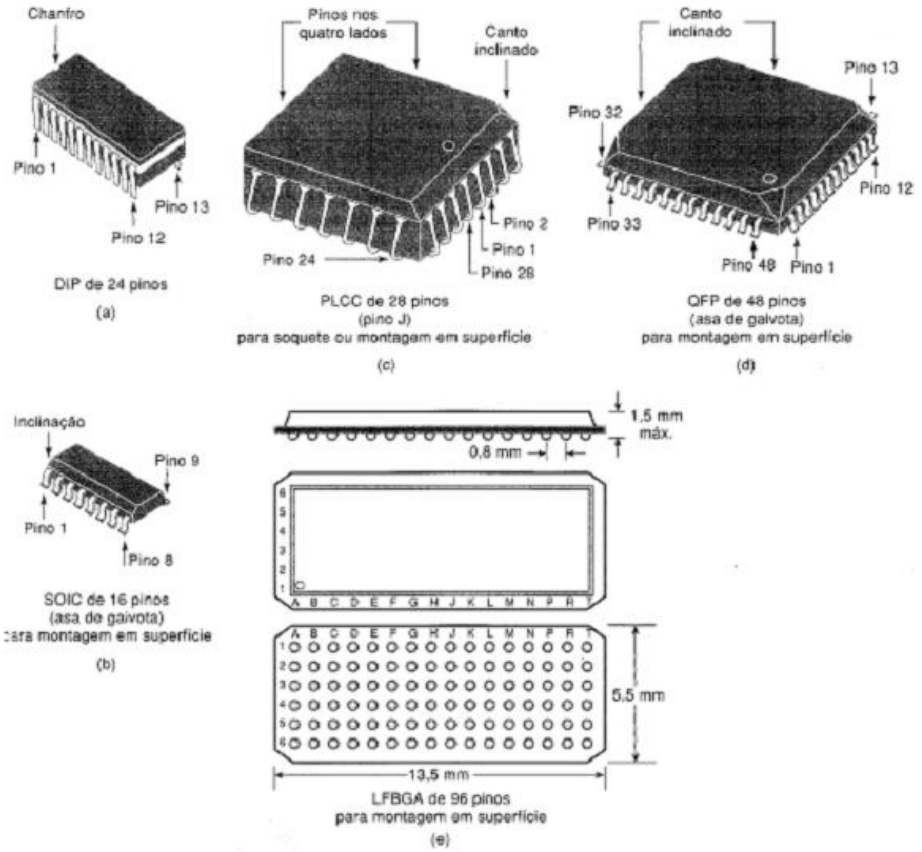
Die de um microprocessador 486DX2 (já com conexões).

A figura abaixo mostra a evolução da complexidade dos CIs ao longo dos anos:



Evolução da complexidade ao longo do tempo.

Quanto ao Encapsulamento



SIGLA	Nome do Encapsulamento	Altura	Dist. entre pinos
DIP	Dual-in-Line Package	5,1mm	2,54mm
SOIC	Small Outline Integrated Circuit	2,65mm	1,27mm
SSOP	Shrink Small Outline Package	2.0mm	0,65mm
TSSOP	Thin Shrink Small Outline Package	1,1mm	0,65mm
TVSOP	Thin Very Small Outline Package	1,2mm	0,4mm
PLCC	Plastic Leaded Chip Carrier	4,5mm	1,27mm
QFP	Quad Flat Pack	4,5mm	0,635mm
TQFP	Thin Quad Flat Pack	1,6mm	0,5mm
LFBGA	Low-Profile Fine-Pitch Ball Grid Array	1,5mm	0,8mm

Quanto à sua família

Entende-se por família de circuitos lógicos, os tipos de estruturas internas que nos permitem a confecção destes blocos em circuitos integrados. Cada família lógica utiliza determinados componentes em seus blocos e, de acordo com estes, a família possuirá determinadas características relacionadas ao seu funcionamento e desempenho prático.

As famílias utilizadas atualmente dentro da área de Eletrônica Digital são a TTL (*Transistor-Transistor Logic*) e a CMOS (Complementary Metal-Oxide Semiconductor), porém estas derivam se uma série de famílias lógicas que hoje estão completamente obsoletas.

1. DCTL (*Direct-Coupled Transistor Logic*)
2. RTL (*Resistor-Transistor Logic*)
3. RCTL (*Resistor-Capacitor Logic*)
4. DTL (*Diode-Transistor Logic*)
5. HTL (*High-Threshold Logic*)

Terminologia de CIs Digitais

Parâmetros de Corrente e Tensão

Embora existam muitos fabricantes de CIs, a maior parte da nomenclatura e da terminologia é padronizada. Os termos mais úteis são:

- **V_{IH} (mín):** Tensão de entrada em nível alto (*high-level input voltage*). O nível de tensão mínimo em uma entrada para que a porta entenda como sendo nível lógico 1. Qualquer tensão abaixo desse nível não será aceita como nível ALTO pelo circuito lógico.
- **V_{IL} (máx):** Tensão de entrada em nível baixo (*low-level input voltage*). O nível de tensão máximo em uma entrada para que a porta entenda como sendo nível lógico 0. Qualquer tensão acima desse nível não será aceita como nível BAIXO pelo circuito lógico.
- V_{OH} (mín): Tensão de saída em nível alto (*high-level output voltage*). O nível de tensão mínimo na saída de um circuito lógico, no estado lógico 1, sob determinadas condições de carga.
- V_{OL} (máx): Tensão de saída em nível baixo (*low-level output voltage*). O nível máximo de tensão na saída de um circuito lógico, no estado lógico 0, sob determinadas condições de carga.
- I_{IH}: Corrente de entrada em nível alto (*high-level input current*). A corrente que flui para uma entrada quando uma tensão de nível ALTO especificada é aplicada nessa entrada.
- I_{IL}: Corrente de entrada em nível baixo (*low-level input current*). A corrente que flui para uma entrada quando uma tensão de nível baixo especificada é aplicada nessa entrada.
- I_{OH}: Corrente de saída em nível alto (*high-level output current*). A corrente que flui de uma saída, no estado lógico 1, sob determinadas condições de carga.
- I_{OL}: Corrente de saída em nível baixo (*low-level output current*). A corrente que flui de uma saída, no estado lógico 0, sob determinadas condições de carga.

Fan-Out

Também chamado de *Fator de Acionamento de Carga.*

É o número máximo de entradas lógicas que uma saída pode acionar com segurança !

(para uma mesma família de CIs)

Atrasos de Propagação

Um sinal lógico sempre sofre atraso ao atravessar um circuito. Os dois tempos de atrasos de propagação existentes em uma porta lógica digital podem ser definidos como:

- t_{pLH}: Tempo de atraso da porta quando a saída comuta de nível BAIXO para nível ALTO.
- t_{pHL}: Tempo de atraso da porta quando a saída comuta de nível alto para nível BAIXO.

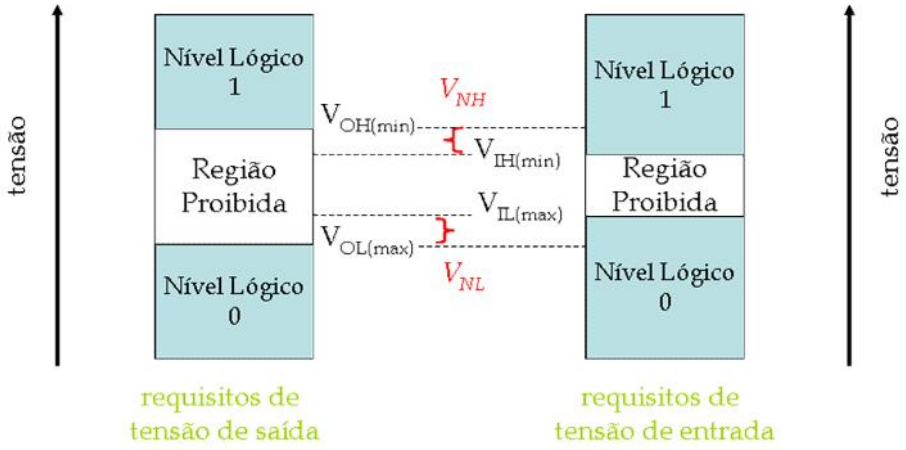
Os valores de tempo de propagação são utilizados como medida de velocidade relativa dos circuitos lógicos. Estas velocidades podem mudar em função da carga.

Imunidade ao Ruído

Campos elétricos e magnéticos podem induzir tensões nos fios de conexão entre os circuitos lógicos. Esses sinais espúrios e indesejáveis são chamados de *ruído* e podem algumas vezes fazer a tensão de entrada de um circuito lógico cair de V_{IH} (min) ou aumentar além de V_{IL} (máx), o que poderia produzir uma operação imprevisível.

A *imunidade ao ruído* de um circuito lógico refere-se à capacidade do circuito de tolerar ruídos sem provocar alterações espúrias na tensão de saída.

Uma medida quantitativa de imunidade ao ruído é denominada ***margem de ruído*** e é ilustrada na Figura abaixo:



- V_{NH}: Margem de ruído para o estado alto
- V_{NL}: Margem de ruído para o estado baixo

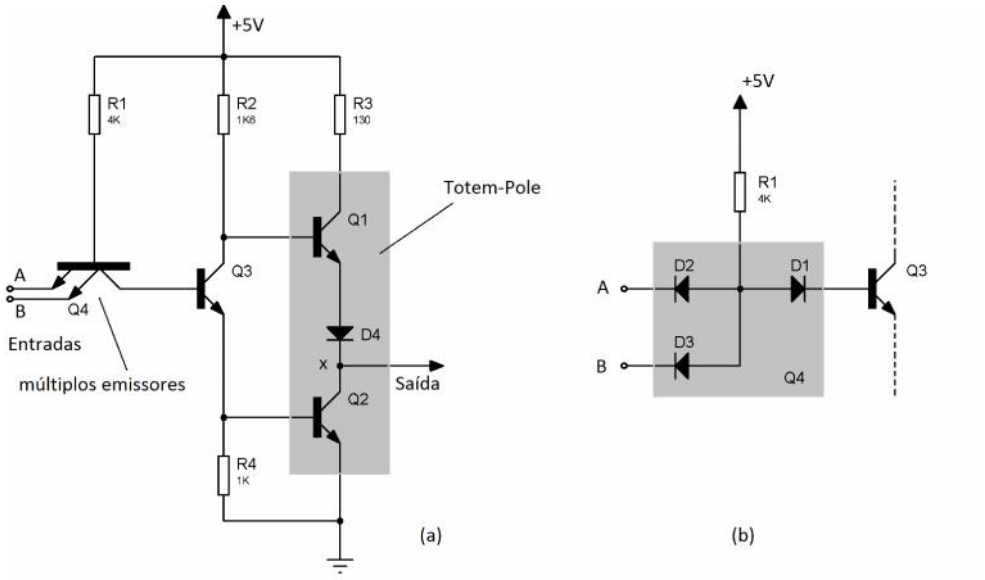
Para operar adequadamente, os níveis de tensão de entrada de um circuito lógico devem ser mantidos fora da região proibida.

Para a família TTL, por exemplo, a região proibida situa-se entre, aproximadamente, 0,8V e 2V.

Em geral, essas regiões não devem nos preocupar muito pois os CIs foram projetados para trabalhar fora dessa região, mas problemas com a fonte de alimentação ou excesso de carga na saída de uma porta lógica, podem levá-la a operar na região proibida.

A Família Lógica TTL

A tecnologia de projeto TTL (*Transistor-Transistor Logic*) existe há mais de 45 anos e encontra-se na fase de declínio. No entanto, ela inspirou toda uma nova série de tecnologias de projeto mais modernas.



(a) Porta NAND TTL básica (b) Equivalente a diodo para Q4

Observe que a saída do circuito através dos transistores Q1 e Q2 estão em uma configuração chamada de *totem-pole*, onde os transistores operam como chave. A função de Q1 é conectar Vcc à saída, produzindo nível lógico ALTO. A função de Q2 é conectar a saída à GND, produzindo nível lógico BAIXO.

A figura cima apresenta a porta NAND TTL operando com sua saída em nível baixo. Embora esse circuito pareça complexo, podemos simplificar a análise utilizando o equivalente ao diodo do transistor de múltiplos emissores Q4, conforme mostrado na própria figura. Os diodos D2 e D3 representam as duas junções base-emissor (BE) de Q4, e D1 representa a junção base-coletor (BC).

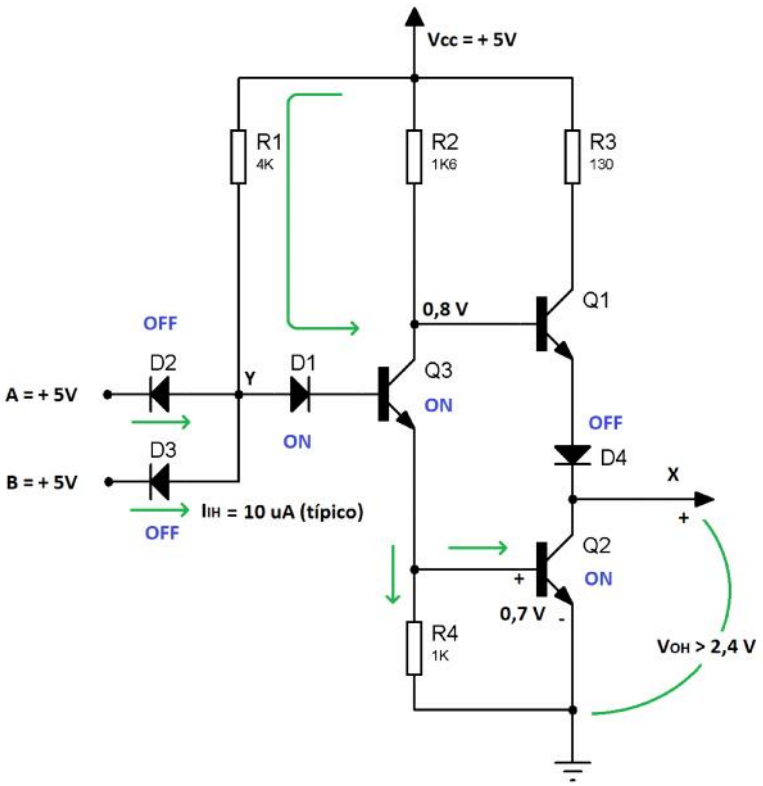
Quando as entradas A e B estão em nível ALTO, a tensão de + 5V nos catodos de D2 e D3 os deixam cortados, e eles praticamente não conduzem corrente. A fonte Vcc fornecerá corrente por R1 e D1 para a base de Q3, que começará a conduzir. A corrente do emissor de Q3 fluirá para a base de Q2 e o fará conduzir.

Ao mesmo tempo, o fluxo de corrente no coletor de Q3 produz queda de tensão sobre R2, que reduz a tensão no coletor de Q3 para um valor insuficiente para Q1 conduzir.

A tensão no coletor de Q3 é de aproximadamente 0,8 V. Isso porque o emissor de Q3 está a 0,7 V em relação à GND, devido à tensão direta entre BE de Q2, e o coletor de Q3 está a 0,1 V em relação a seu emissor devido a Vce (sat). Esse valor de 0,8 V na base de Q1 não é suficiente para polarizar diretamente a junção de BE de Q1 e o diodo D1. Na verdade, D1 é necessário para manter Q3 cortado nessa situação.

Com Q2 conduzindo, o terminal de saída X estará com tensão muito baixa, visto que a resistência de Q2, quando conduz, é baixa (1 a 25 Ω). Na verdade, a tensão de saída V_{OL} depende de quanta corrente de coletor Q2 conduz. Com Q1 cortado, não existe corrente vindo do terminal da fonte Vcc por R4. A corrente do coletor de Q2 virá das entradas TTL às quais o terminal de saída X estiver conectado (transistor Q4 multiemissor).

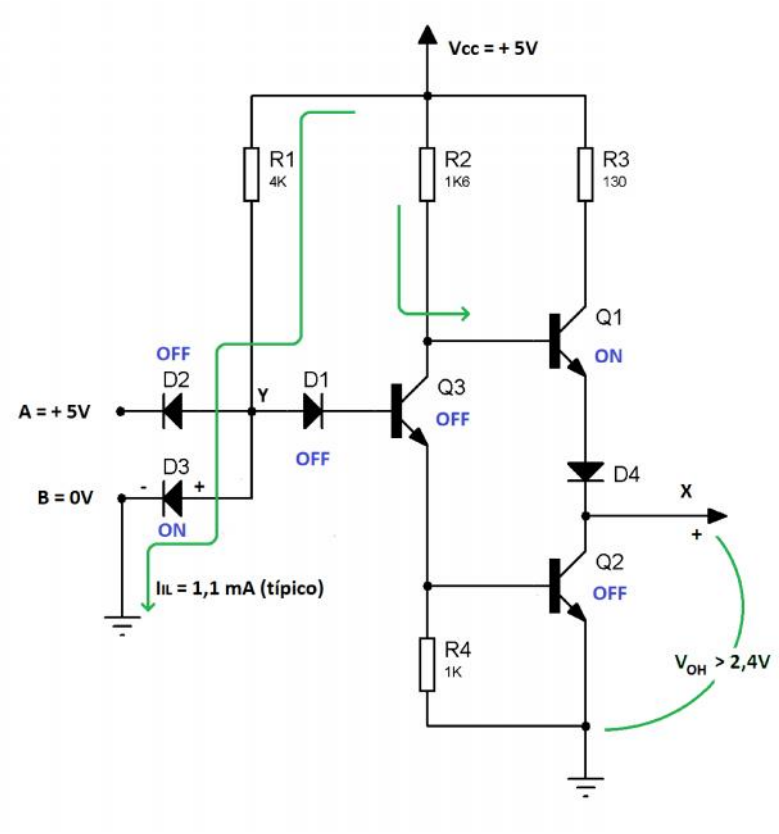
Em nível ALTO, as entradas A e B devem fornecer apenas a pequena corrente de fuga dos diodos D2 e D3, que é tipicamente de 10 μA à temperatura ambiente.



Porta NAND TTL com saída em nível lógico BAIXO.

A próxima figura mostra a situação em que a saída da porta está em nível lógico ALTO. Nesse caso, a entrada B está em nível lógico BAIXO. Isso irá polarizar D3 diretamente e a tensão no ponto Y será de 0,7 V. Essa tenão não é suficiente para polarizar D1 e a junção BE de Q3 para condução.

Com Q3 em corte, não existe corrente de base para Q2 e ele também corta. Como não existe corrente de coletor em Q2, a tensão na base de Q1 será grande o suficiente para polarizar diretamente Q1 e D4, de modo que Q1 conduza.



Porta NAND TTL com saída em nível lógico ALTO.

Séries TTL

Os fabricantes normalmente marcam seus chips com um prefixo próprio. Por exemplo, a Texas Instruments utiliza o prefixo SN, a National Semiconductor utiliza o prefixo DM, a Signetics utiliza o prefixo S. Alguns exemplos podem ser: SN7400, DM7404, S7485,...

Existem diversas séries dentro da família TTL como 54, 74, 74LS, 74S, 74 ALS, etc...

- S – Schottky
- L – Low Power
- A – Advanced
- F - Fast

A família 54ALS é a mais cara dentre todas, pois por trabalhar em condições de extremas temperaturas, tem grande emprego em aplicações militares e espaciais (-55° C a +125° C contra 0° a +70° C das outras).

	74	74S	74LS	74AS	74ALS	74F
Atraso de propagação (ns)	9	3	9,5	1,7	4	3
Dissipação de potência (mW)	10	20	2	8	1,2	6
Taxa de clock máxima (MHz)	35	125	45	200	70	100
Fan-out	10	20	20	40	20	33
V_{OH} (min) (V)	2,4	2,7	2,7	2,5	2,5	2,5
V_{OH} (máx) (V)	0,4	0,5	0,5	0,5	0,5	0,5
V_{IH} (min) (V)	2,0	2,0	2,0	2,0	2,0	2,0
V_{OH} (máx) (V)	0,8	0,8	0,8	0,8	0,8	0,8

OBS: Qualquer entrada para um circuito TTL que é deixada desconectada (aberta ou flutuando) é interpretada como sendo sempre nível lógico 1. No entanto, não é recomendado devido à ruído.

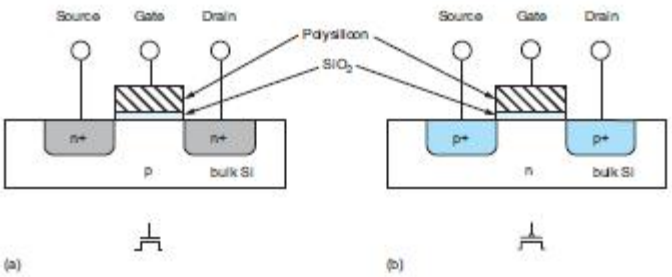
A Família Lógica CMOS

O termo tecnologia MOS (*Metal Oxide Semiconductor*) é derivado da estrutura MOS básica, que consiste em um eletrodo de metal sobre um óxido isolante, que por sua vez está sobre um substrato de semicondutor. Os transistores implementados com essa tecnologia são transistores de efeito de campo chamados de MOSFET (ou simplesmente MOS).

As principais vantagens dos transistores MOSFET são: simplicidade de construção, baixo custo de fabricação, dimensões muito reduzidas e baixíssimo consumo de energia.

A maior desvantagem em um dispositivo MOS é o risco de ser danificado por eletricidade estática.

A figura abaixo mostra a estrutura de um transistor nMOS e de um pMOS.



(a) Transistor nMOS (b) Transistor pMOS.

Os CIs CMOS (união de nMOS com pMOS) fornecem as mesmas funções lógicas disponíveis na família TTL, mas também várias funções especiais não disponíveis nesta.

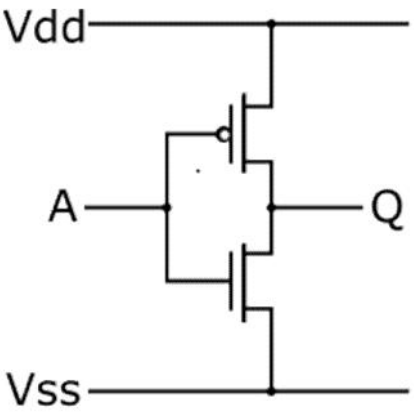
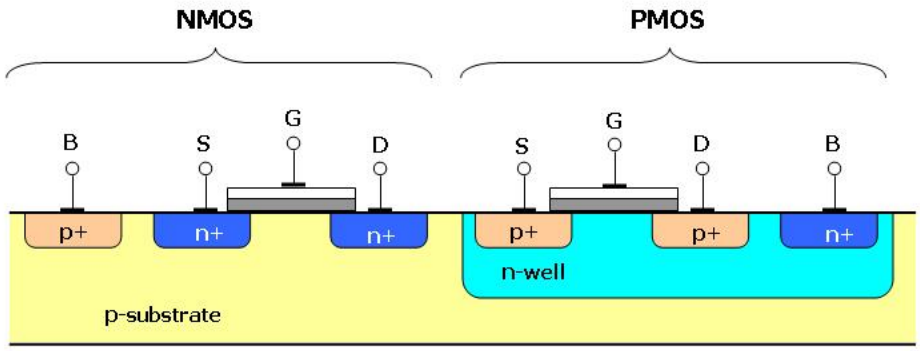


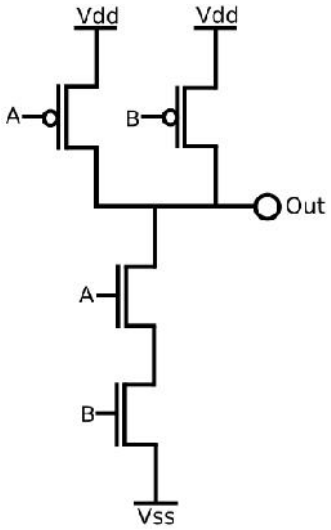
Diagrama esquemático de uma porta inversora CMOS.

A figura anterior apresenta o diagrama esquemático de uma porta inversora utilizando dois transistores MOS, um PMOS e o outro NMOS (daí o nome CMOS), enquanto a próxima apresenta a estrutura da porta em um substrato de silício.

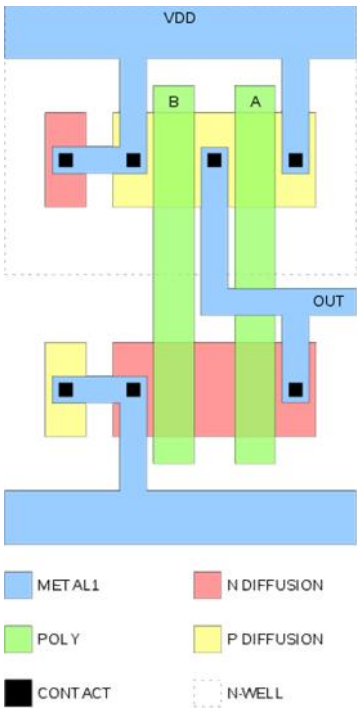


Apresentação de uma porta inversora (sem as demais conexões) CMOS em um chip de silício.

Como outro exemplo, veja as duas figuras a seguir que apresentam o diagrama esquemático de uma porta NAND CMOS.



Esquemático de uma porta NAND com transistores MOS.



Layout de uma porta NAND CMOS de duas entradas.

A figura acima mostra o layout da porta NAND CMOS. Em geral, cada estrutura colorida dará origem a uma máscara para fabricação do chip.

Séries CMOS

Existem diversas séries dentro da família CMOS como:

- 4000 (RCA)/1400 (Motorola) – mais antigas e funcionalmente equivalentes. Possuem baixo consumo e operam entre (3V e 15 V). São lentos quando comparados com a família TTL ou a outras séries CMOS;
- 74HC/74HCT – São cerca de dez vezes mais rápidos quando comparados à família 74LS e capacidade de corrente maior. São compatíveis pino a pino e funcionalmente equivalentes a TTL de mesma numeração;
- 74AC/74ACT – Melhor imunidade a ruídos, funcionalmente equivalentes a TTL, mas não são equivalentes pino a pino;
- 74AHC/AHCT – CMOS avançado de alta velocidade, baixo consumo e baixa capacidade de acionamento;
- 74BCT/74ABT – Alguns fabricantes uniram as melhores características da lógica bipolar (BJTs) e do CMOs, criando assim a tecnologia BiCMOS, que oferecem:
    - Baixo consumo

- Alta velocidade
- Redução de 75% no consumo em relação à família 74F, porém com mesma velocidade e capacidade de acionamento

OBS: Família 4000 e 14000 – 3 a 15V, família 74HC/HCT, 74AC/ACT e 74AHC/AHCT – 2 a 6V.

	CMOS							TTL			
Parâmetro	4000B	74HC	74HCT	74AC	74ACT	74AHC	74AHCT	74	74LS	74AS	74ALS
V_{IH}(min)	3,5	3,5	2,0	3,5	2,0	3,85	2,0	2,0	2,0	2,0	2,0
V_{IL}(máx)	1,5	1,0	0,8	1,5	0,8	1,65	0,8	0,8	0,8	0,8	0,8
V_{OH}(min)	4,95	4,9	4,9	4,9	4,9	4,4	3,15	2,4	2,7	2,7	2,5
V_{OL}(máx)	0,05	0,1	0,1	0,1	0,1	0,44	0,1	0,4	0,5	0,4	0,5
V_{NH}	1,45	1,4	2,9	1,4	2,9	0,55	1,15	0,4	0,7	0,7	0,7
V_{NL}	1,45	0,9	0,7	1,4	0,7	1,21	0,7	0,4	0,3	0,3	0,4

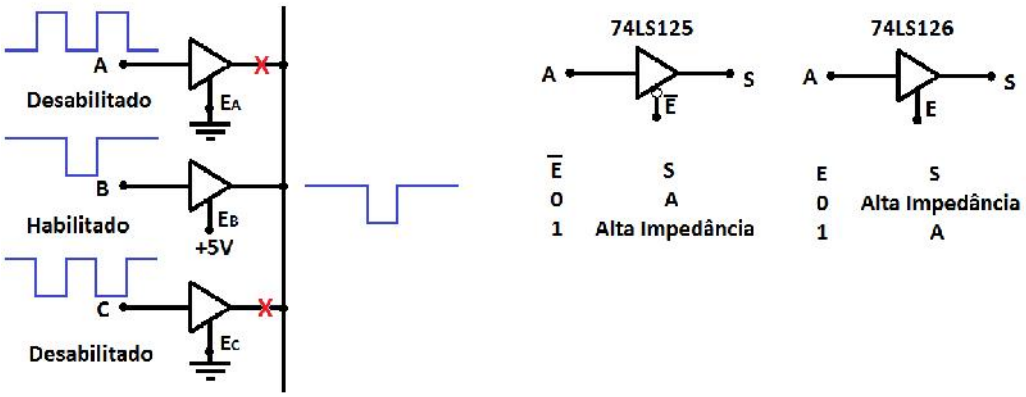
- Imunidade à Ruído: CMOS é melhor que TTL
- Dissipação de potência: CMOS é melhor que TTL em baixas frequências, porém, a partir de 2 a 3 MHz, começa a perder;
- Fan-out: Depende do atraso de propagação e da frequência de operação, basicamente, empata;
- Velocidade de Comutação: CMOS melhor que TTL, principalmente a família 74AHC;

Buffers Tristate (Três Estados)

Um *buffer tristate* é um circuito que controla a passagem de um sinal lógico da entrada para a saída (alguns invertem o sinal).

Basicamente, são empregados em circuitos digitais em que diversos sinais são conectados a linhas em comum (como um barramento, por exemplo).

São exemplos de buffers tristate os CIs 74LS125, 74LS126 e 74AHC126.



Exemplo de uso de buffers tristate.

Circuitos Seqüenciais

Latch

O circuito mostrado abaixo consiste de um par de inversores acoplados, e é a estrutura básica de um circuito lógico e de grande importância chamado de *latch estático*.

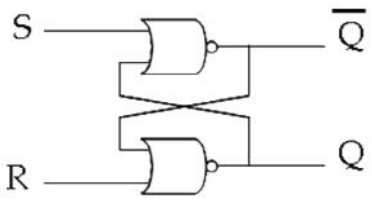
O elemento novo e essencial que o latch introduz às estruturas lógicas até agora consideradas é que o latch pode ser usado para estabelecer e manter um nível lógico sem qualquer intervenção externa.

Em todas as estruturas lógicas que examinamos anteriormente, a saída de uma porta dependia das entradas da porta que eram estabelecidas por uma fonte externa. Na ausência de entradas não é possível conceber saídas de portas sem incorrer em ambigüidades.

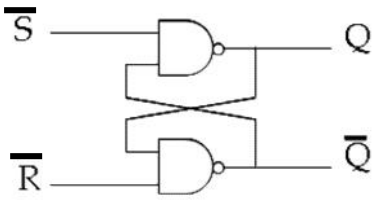
Devido a esta independência do latch das entradas externas, ele pode ser usado para armazenar, isto é, registrar ou lembrar um bit lógico.

O Latch SR

Um "Latch" é um tipo de dispositivo lógico biestável ou multivibrador. Um latch S-R (Set-Reset) com entrada ativa em nível alto é formado por duas portas NOR tendo acoplamento cruzado. Um latch S-R com entrada ativa em nível baixo é formado por duas portas NAND.



Entrada ativa em nível alto



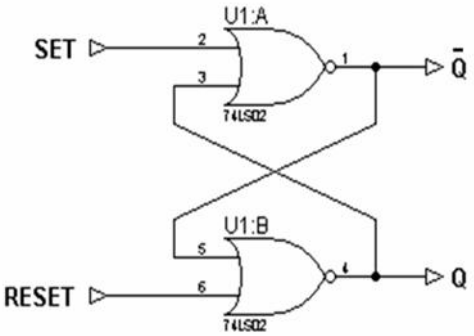
Entrada ativa em nível baixo

Latch com Portas NOR

A tabela abaixo é a Tabela de Estados do Latch SR com portas NOR.

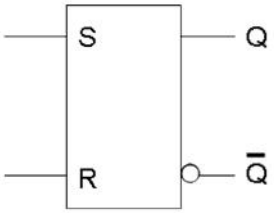
ENTRADA		SAÍDAS				
S	R	Q_{t-1}	Q_t	$\overline{Q}_{t-1}$	$\overline{Q}_t$	
0	0	0	0	1	1	Estado de repouso
0	0	1	1	0	0	
0	1	0	0	1	1	Reset
0	1	1	0	0	1	
1	0	0	1	1	0	Set
1	0	1	1	0	0	
1	1	0	0	1	0	Estado Inválido
1	1	1	0	0	0	

Tabela de Estados do Latch SR com portas NOR.



SET	RESET	Saída Q
0	0	Não muda
0	1	Q = 0
1	0	Q = 1
1	1	Inválida*

* Produz $Q = \overline{Q} = 0$



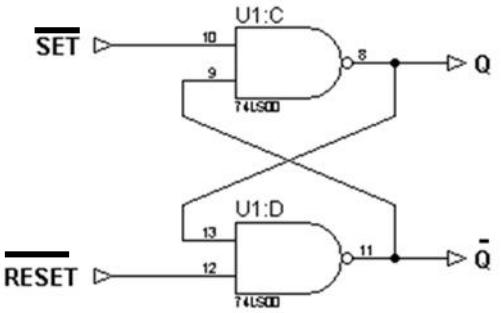
Símbolo do latch SR com entras ativas em nível alto.

Latch com Portas NAND

A tabela abaixo é a Tabela de Estados do Latch SR com portas NOR.

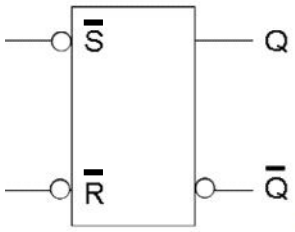
ENTRADA		SAÍDAS				
S	R	Q_{t-1}	Q_t	$\overline{Q}_{t-1}$	$\overline{Q}_t$	
0	0	0 →	1	1 →	1	Estado Inválido
0	0	1 →	1	0 →	1	
0	1	0 →	1	1 →	0	Set
0	1	1 →	1	0 →	0	
1	0	0 →	0	1 →	1	Reset
1	0	1 →	0	0 →	1	
1	1	0 →	0	1 →	1	Estado de repouso
1	1	1 →	1	0 →	0	

Tabela de Estados do latch com portas NAND.



SET	RESET	Saída Q
0	0	Inválida*
0	1	Q = 1
1	0	Q = 0
1	1	Não muda

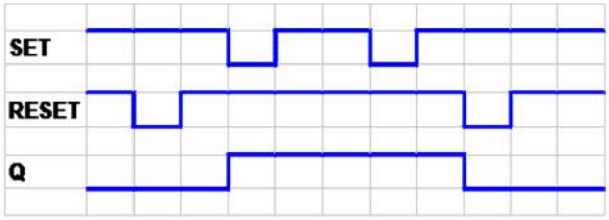
* Produz $Q = \overline{Q} = 0$

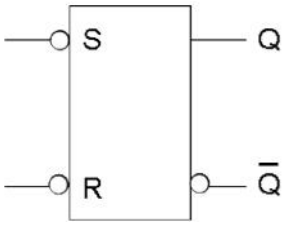


Símbolo do latch SR com entras ativas em nível baixo.

Exemplo:

As formas de onda mostras abaixo são aplicadas no latch SR. Considerando que o estado inicial do latch é dado por Q = 0, determine a forma de onda na saída Q ao longo do tempo.





Estado do latch quando energizado

Quando um flip-flop é energizado, não é possível prever o estado inicial da saída do flip-flop se as entradas SET e RESET estiverem inativas (S=R=1 para um latch NAND, S=R=0 para um latch NOR). Isto dependerá de vários fatores.

Se um flip-flop tiver de iniciar em um estado particular para garantir uma operação adequada de um circuito, ele terá de ser colocado no estado desejado, ativando momentaneamente a entrada SET ou RESET no início da operação do circuito.

Isso é obtido aplicando-se um pulso na entrada apropriada !

O Latch SR Controlado

Os sistemas digitais podem operar tanto em modo síncrono como assíncrono.

Nos sistemas assíncronos, as saídas dos circuitos lógicos podem mudar de estado a qualquer momento em que uma ou mais entradas mudarem de estado.

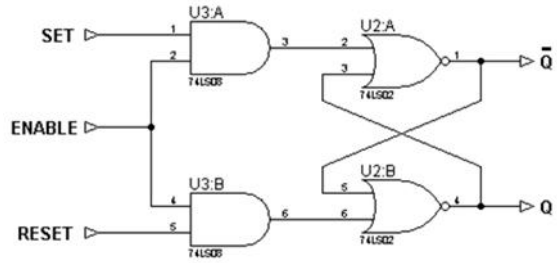
Tanto o projeto quanto a análise de defeitos são bem mais difíceis em um sistema assíncrono.

Em sistemas síncronos, os momentos exatos em que uma saída qualquer pode mudar de estado, são determinados por um sinal normalmente denominado clock (um sinal de habilitação para o flip-flop).

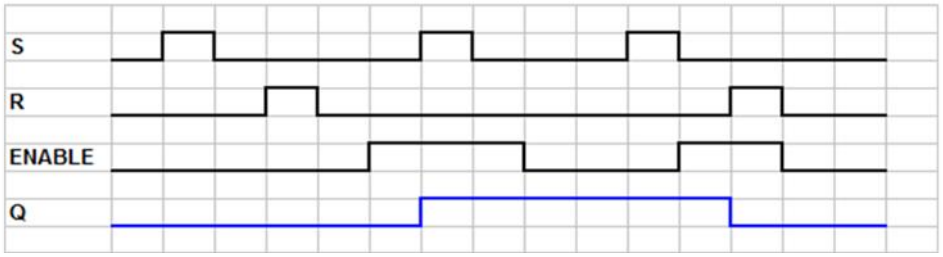
- O sinal de clock é geralmente um trem de pulsos retangulares ou uma onda quadrada.

- O clock é distribuído para todas as partes do sistema, sendo que todas as saídas dos flip-flops (ou a maioria) muda de estado apenas durante o intervalo onde o clock encontra-se em nível lógico "1".

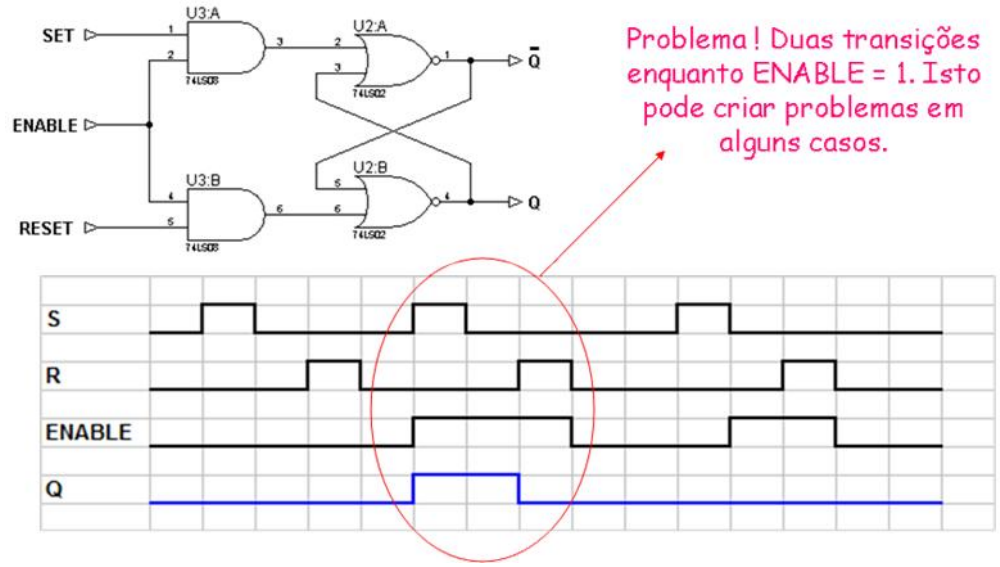
Exemplo:



Agora, somente durante os instantes onde ENABLE = 1 é que o latch responde às mudanças nas entradas!



Mas ainda há um problema:



- Durante o intervalo de tempo em que o sinal de ENABLE (clock) está em nível lógico 1, mudanças no estado do flip-flop podem ocorrer devido à presença de ruído, o que é indesejável !

- Uma solução é estreitar ao máximo o pulso de clock, reduzindo assim o intervalo onde ele permanece em nível lógico "1".

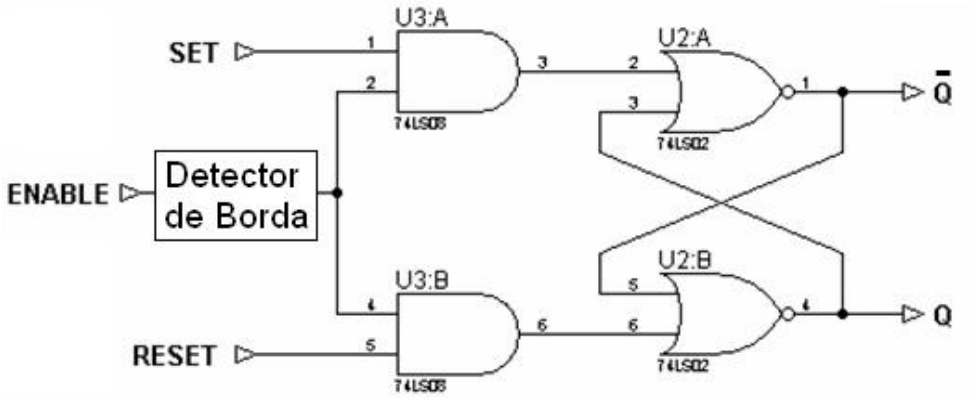
- No limite, poderemos detectar a transição do nível lógico do pulso de clock e utilizá-lo a nosso favor.

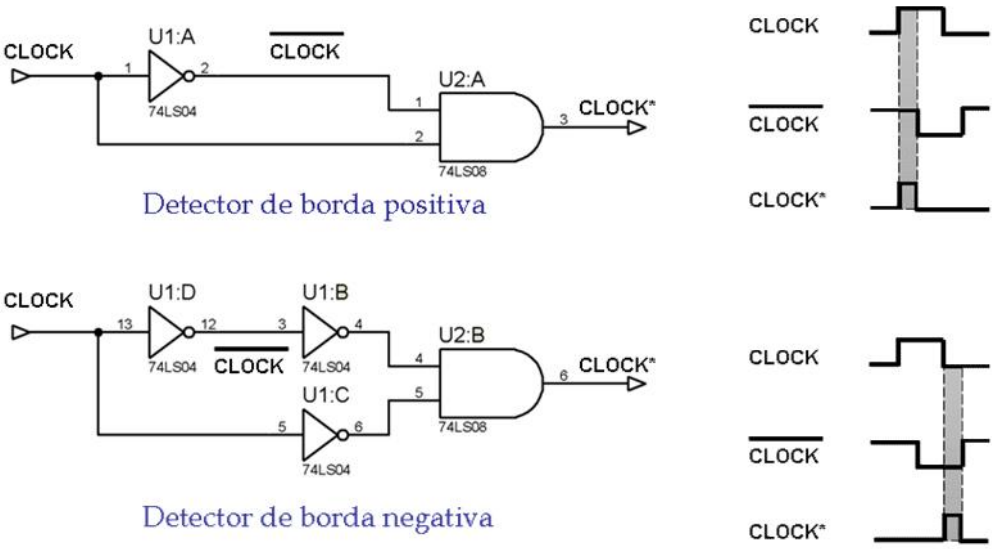
- Quando o clock faz uma transição de 0 para 1, denomina-se transição de subida (ou borda de subida).

- Quando o clock faz uma transição de 1 para 0, denomina-se transição de descida (ou borda de descida).

O Detector de Borda (ou transição)

Como dito anteriormente, uma solução para o problema é a inserção de um detector de borda ou transição:

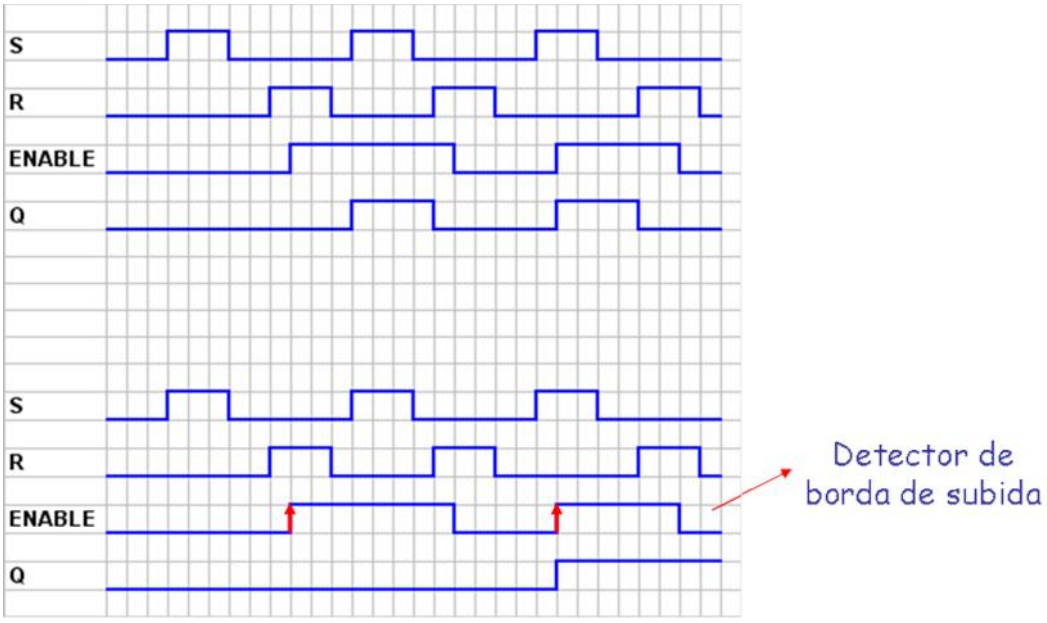




A duração dos pulsos CLOCK é normalmente de 2 a 5 ns.*

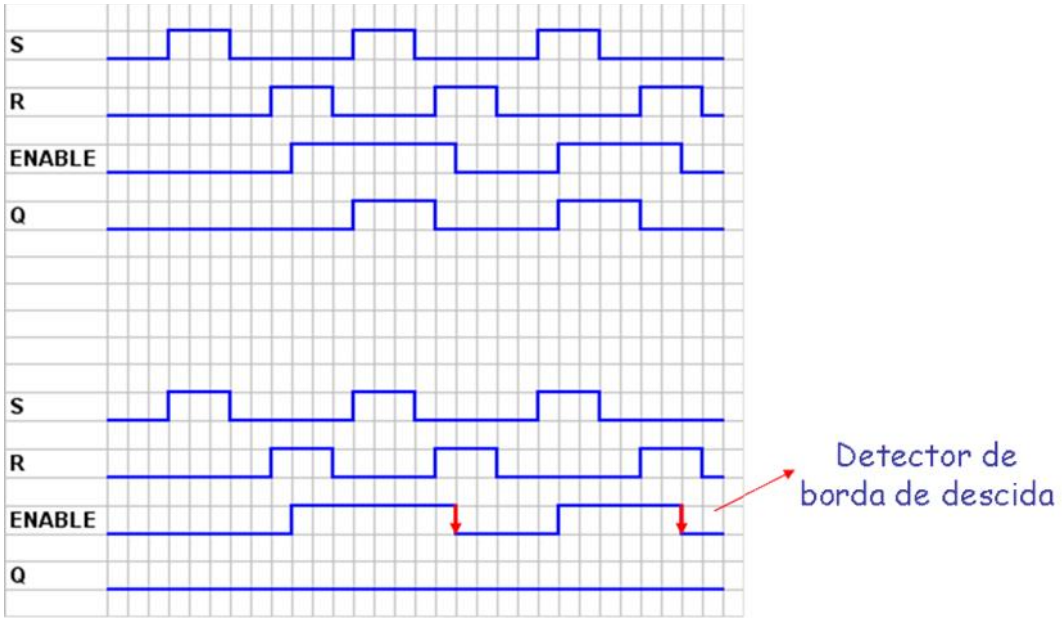
O Latch Controlado com Detector de Borda

A figura a seguir apresenta um exemplo de funcionamento do latch controlado com detector de borda de subida.



Resposta do latch controlado com detector de borda de subida.

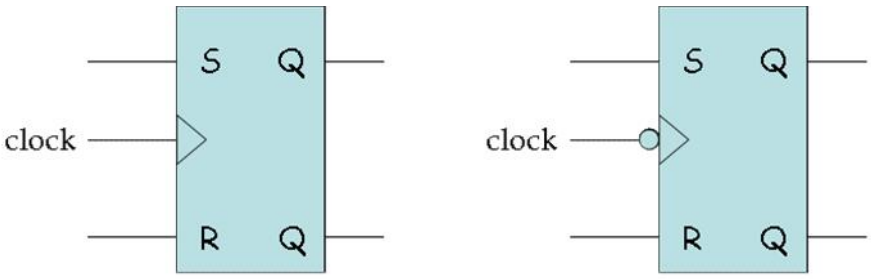
A figura a seguir apresenta um exemplo de funcionamento do latch controlado com detector de borda de descida.



Resposta do latch controlado com detector de borda de descida.

O Latch SR com Clock

O latch controlado com detector de borda passa agora a receber um novo nome: FLIP-FLOP.



ENTRADAS			SAÍDA
S	R	CLK	Q
0	0	↑	Q_0
0	1	↑	0
1	0	↑	1
1	1	↑	Inválido

ENTRADAS			SAÍDA
S	R	CLK	Q
0	0	↓	Q_0
0	1	↓	0
1	0	↓	1
1	1	↓	Inválido

Exercícios

Tocci: 5.1 a 5.3, 5.9, 5.13 a 5.15 e 5.20.

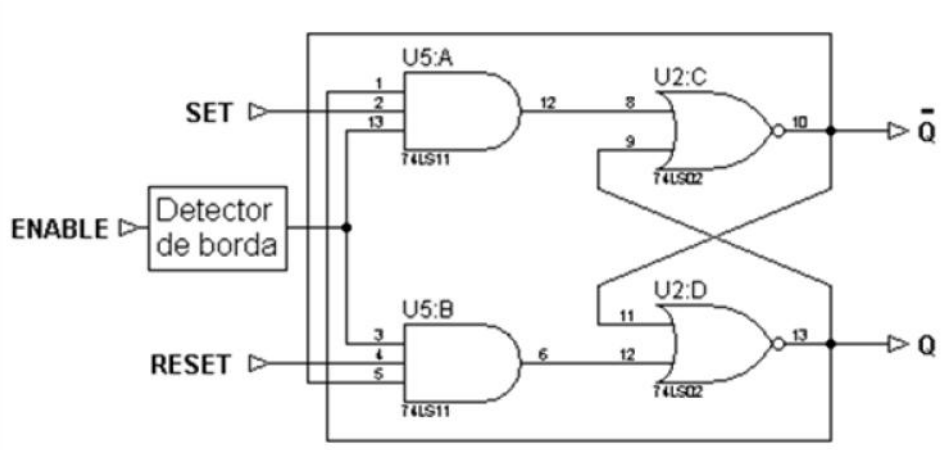
Flip-Flops

O Flip-Flop JK disparado por borda

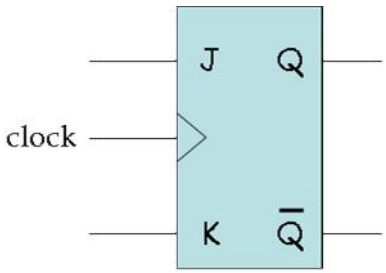
Vamos retornar ao problema do latch, que possuía um estado chamado de estado PROIBIDO.

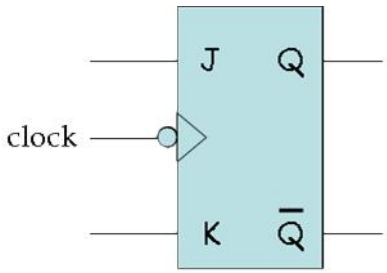
O problema do estado proibido é que o projetista deve preocupar-se com este estado durante o projeto, de forma a garantir que as entradas proibidas jamais ocorram.

Um possível solução é alterar o projeto do flip-flop de forma a eliminarmos o estado proibido, substituindo-o por algo mais útil. Esta é a proposta do flip-flop JK.



ENTRADAS			SAÍDA	
J	K	CLK	Q	
0	0	↑	Q_0	repouso
0	1	↑	0	reset
1	0	↑	1	set
1	1	↑	$\bar{Q}_0$	toggle

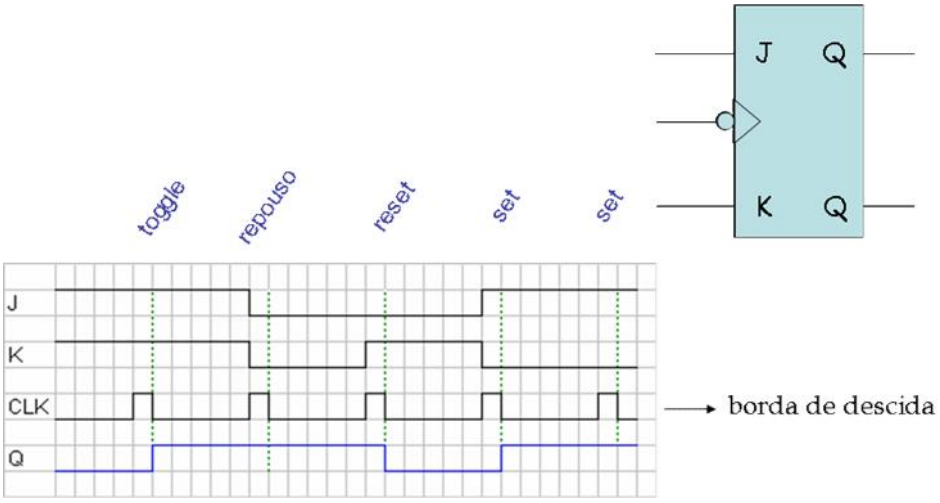




ENTRADAS			SAÍDA
J	K	CLK	Q
0	0	↑	Q_0
0	1	↑	0
1	0	↑	1
1	1	↑	$\overline{Q}_0$

ENTRADAS			SAÍDA
J	K	CLK	Q
0	0	↓	Q_0
0	1	↓	0
1	0	↓	1
1	1	↓	$\overline{Q}_0$

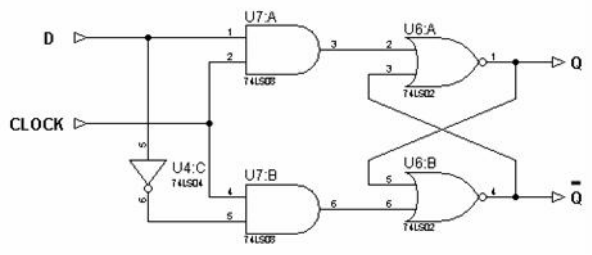
Exemplo de funcionamento:

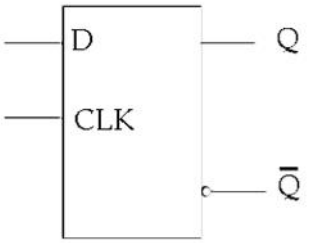


O Flip-Flop do tipo T

O Flip-Flop do tipo T é um flip-flop que é mencionado na literatura do circuitos digitais, mas que não é encontrado comercialmente. O flip-flop tipo T é tão somente um flip-flop do tipo JK com as entradas J e K interligadas. Com isto, o flip-flop do tipo T só pode estar em *repouso* ou em *toggle*.

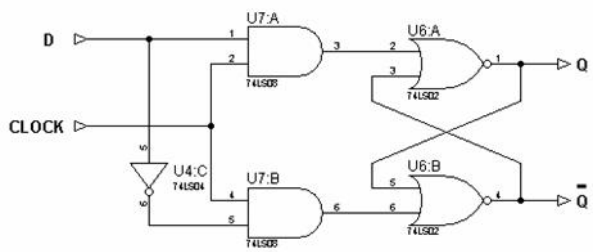
Flip-Flop do tipo D

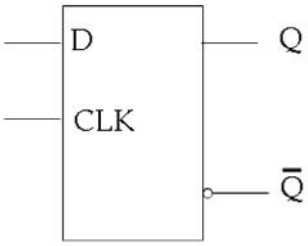


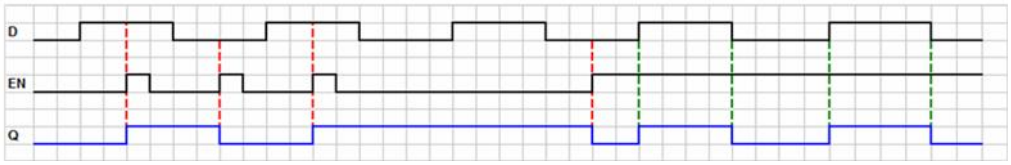


D	CLK	Q	$\bar{Q}$	Comentário
0	⎍	0	1	RESET
1	⎍	1	0	SET

Exemplo de funcionamento:





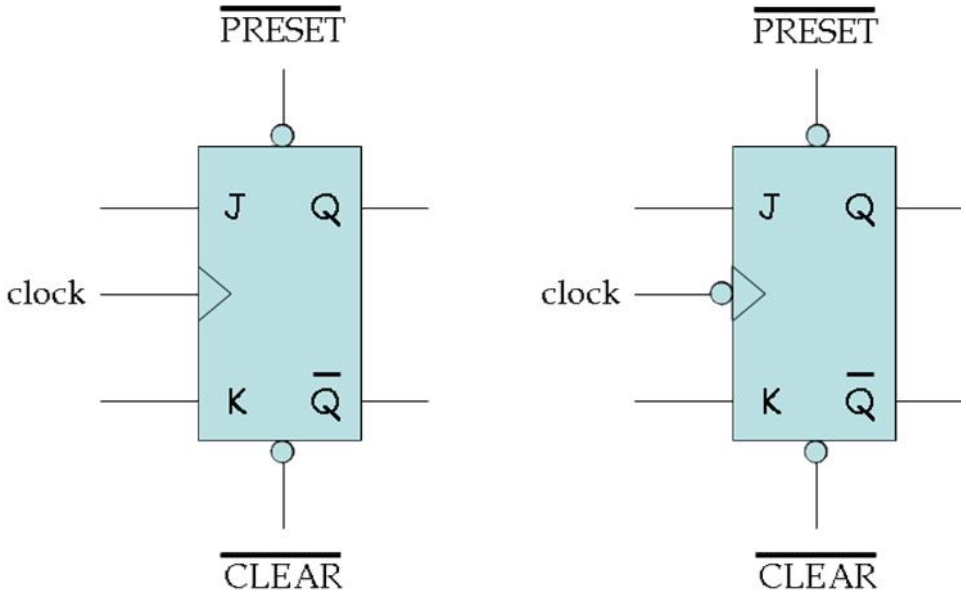


Entradas Assíncronas

Até agora, as entradas J, K, S, R e D tem sido chamadas de entradas de controle. Estas entradas também são conhecidas como entradas síncronas, porque seu efeito na saída do FF é sincronizado com a borda de subida (ou descida) do clock.

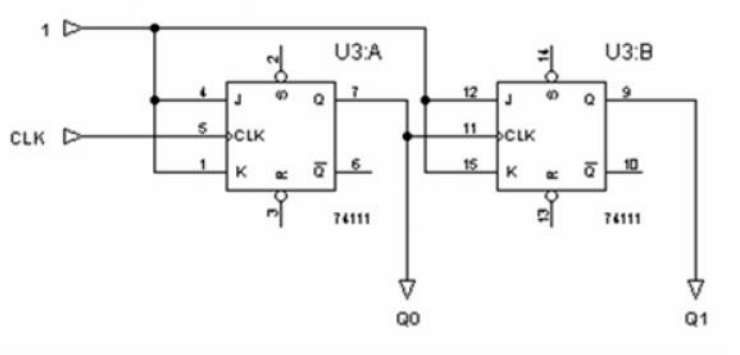
A maioria dos FFs possuem uma ou mais entradas assíncronas que operam de forma independente das entradas síncronas e da entrada de clock. Essas entradas podem ser usadas para colocar o FF no estado 1 ou 0 em *qualquer instante de tempo*, independentemente das condições das outras entradas.

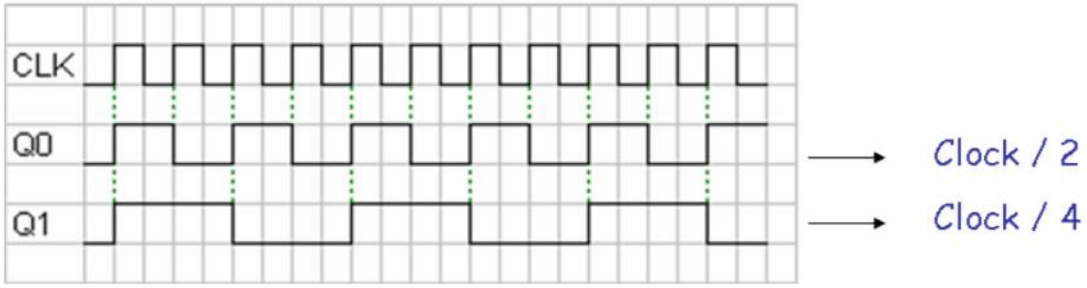
Em geral, essas entradas são chamadas de $\overline{\text{PRESET}}$ (ou $\overline{\text{SET}}$) e $\overline{\text{CLEAR}}$ (ou $\overline{\text{RESET}}$) e são ativas em nível lógico 0.



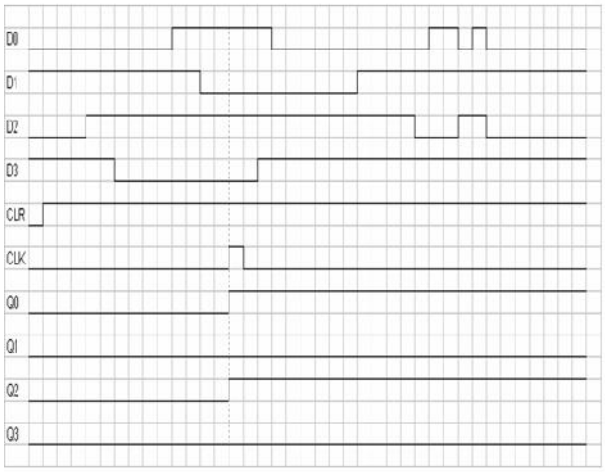
Aplicações

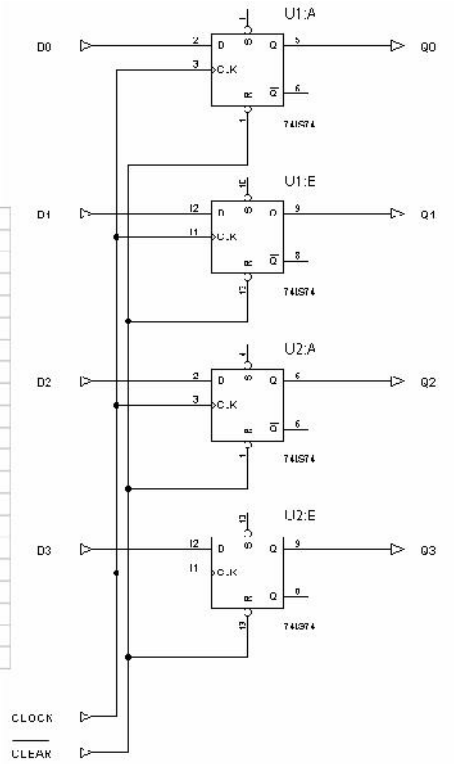
Divisor de Freqüência



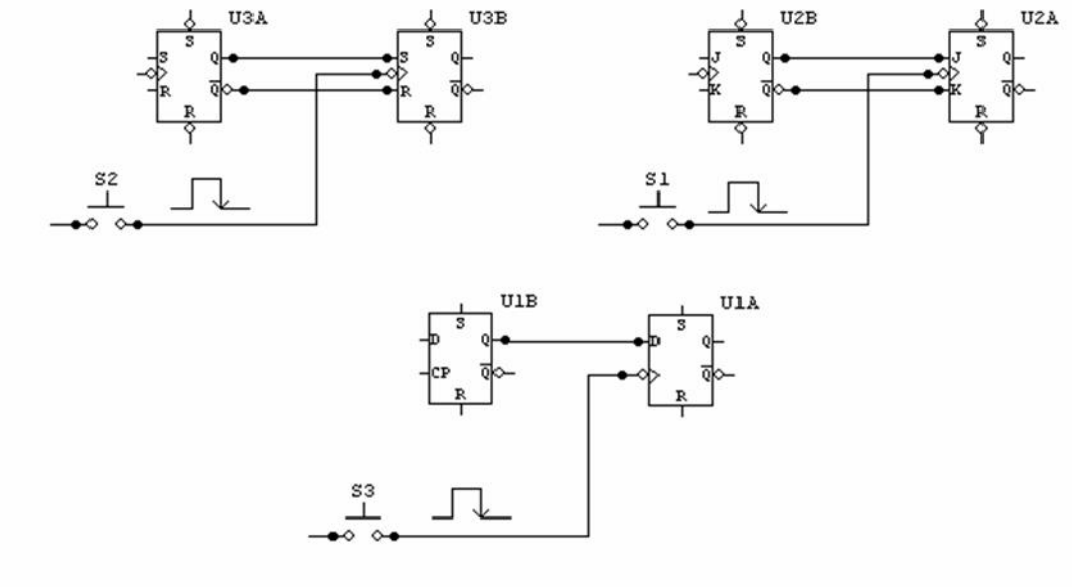


Armazenamento de Dados em Paralelo

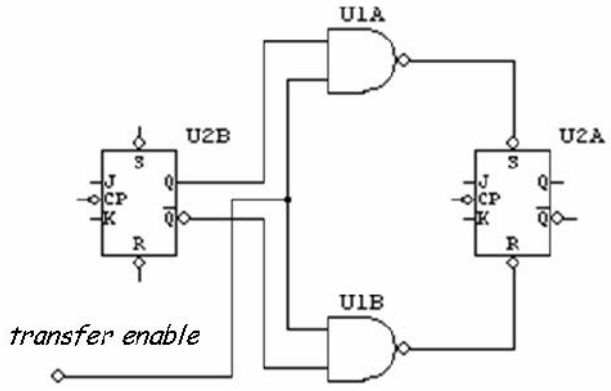




Transferência de Dados



Transferência síncrona de dados entre FFs de diferentes tipos



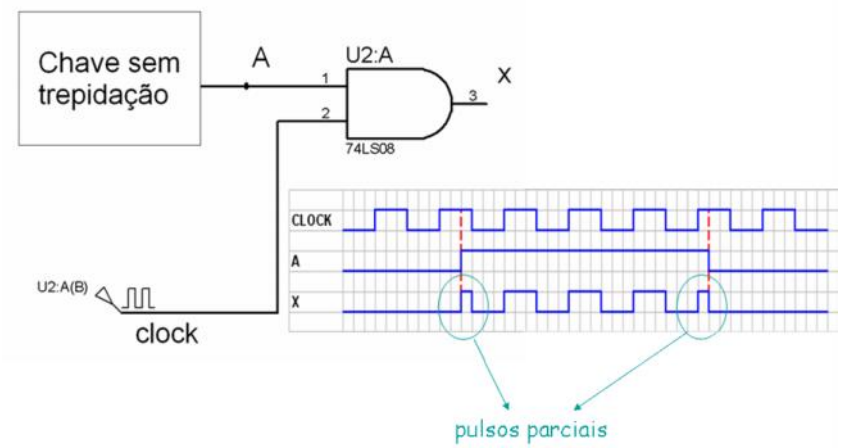
Transferência assíncrona de dados entre FFs de diferentes tipos

Sincronização

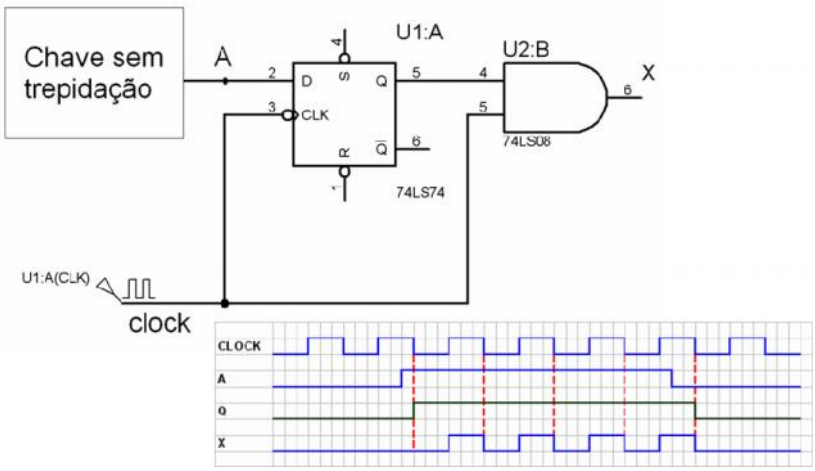
A maioria dos sistemas digitais opera de forma essencialmente síncrona, e a maioria dos sinais muda de estado em sincronismo com as transições de clock.

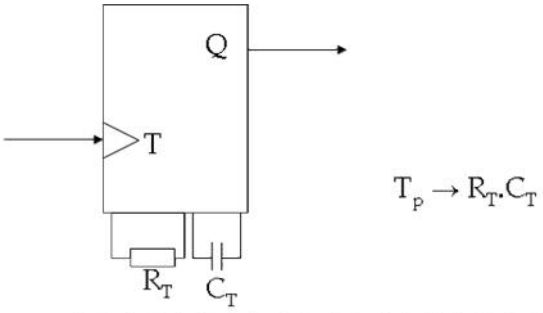
Entretanto, em alguns outros casos, haverá um sinal externo que não estará sincronizado com o clock (devido à sua natureza imprevisível, por exemplo). Em outras palavras, o sinal será assíncrono.

Exemplo: Pulsos parciais

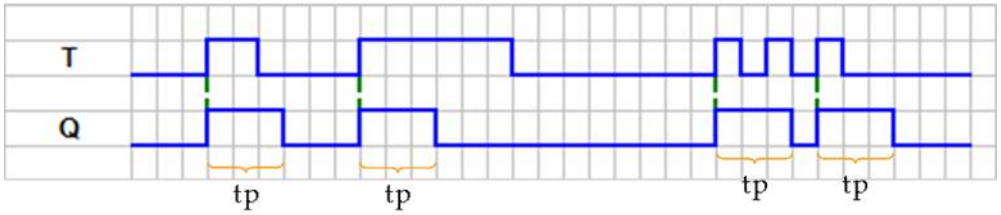


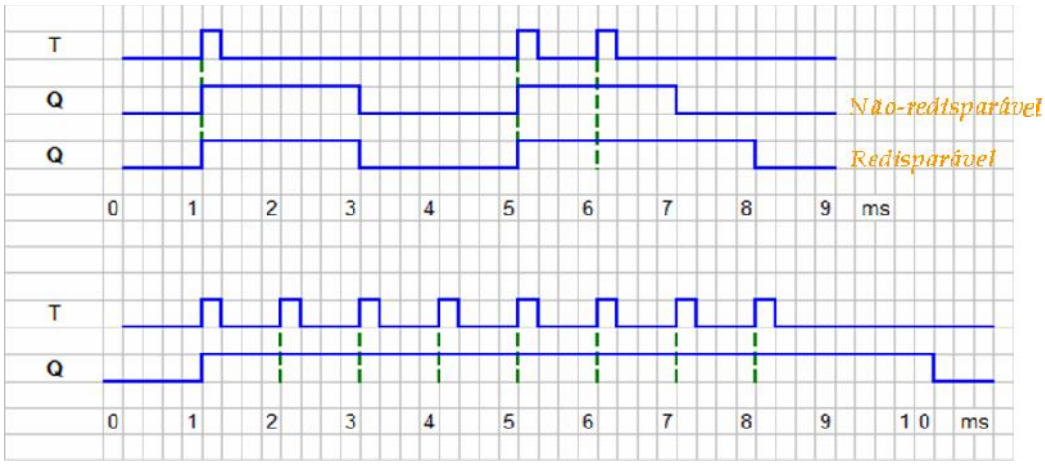
A solução para esse problema pode ser obtida através do uso de um flip-flop do tipo D:





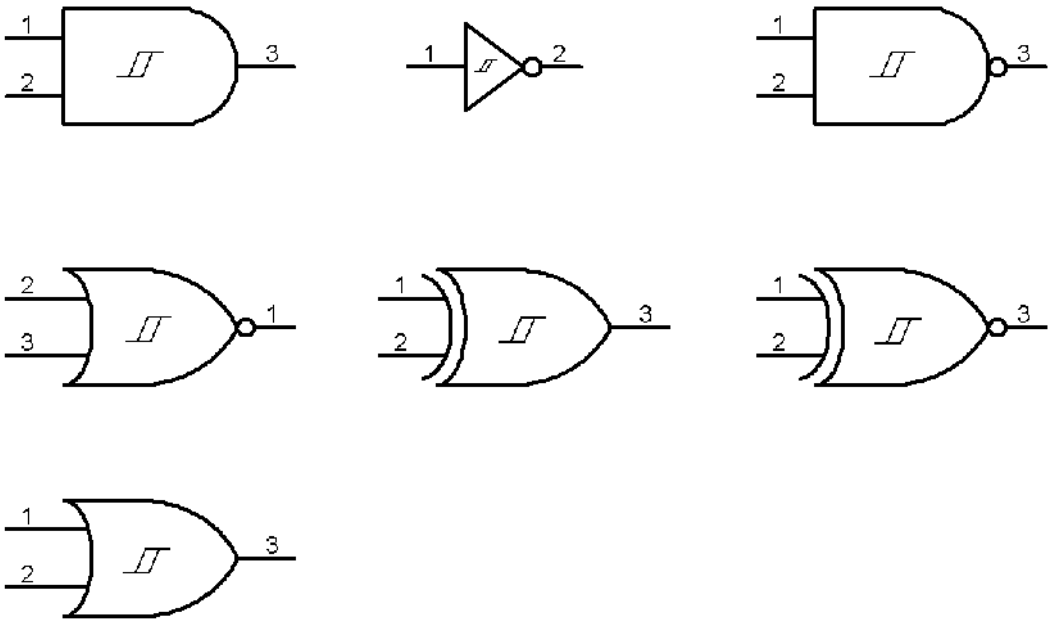
$$T_p \rightarrow R_T.C_T$$





Dispositivos Schmitt-Trigger

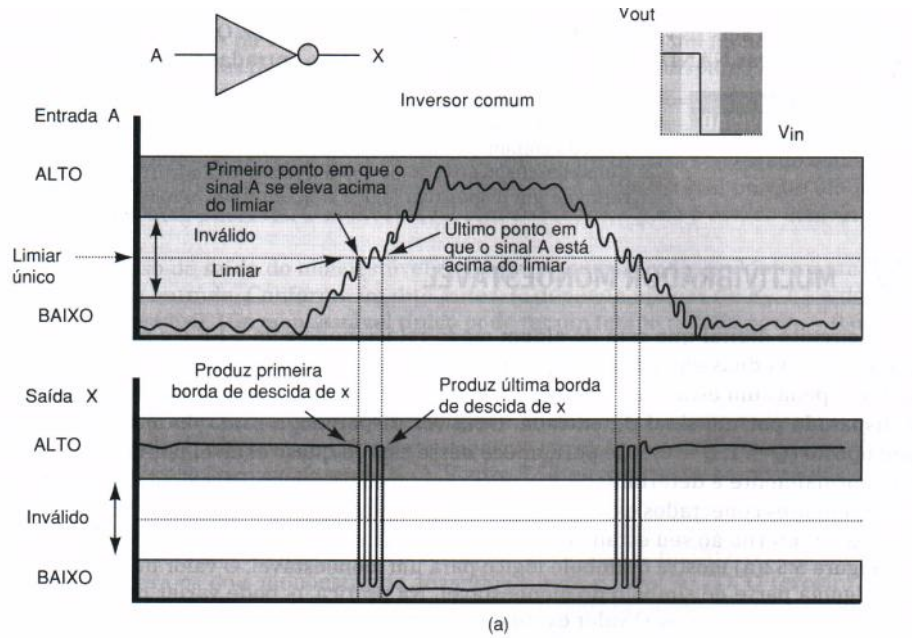
- Não é um Flip-Flop
- Exibe características de memória (efeito de histerese)
- Útil em determinadas situações:
  - Transições lentas
  - Sinal lógico ruidoso
- Todas as portas lógicas que possuem a característica de schmitt-trigger possuem um símbolo especial.



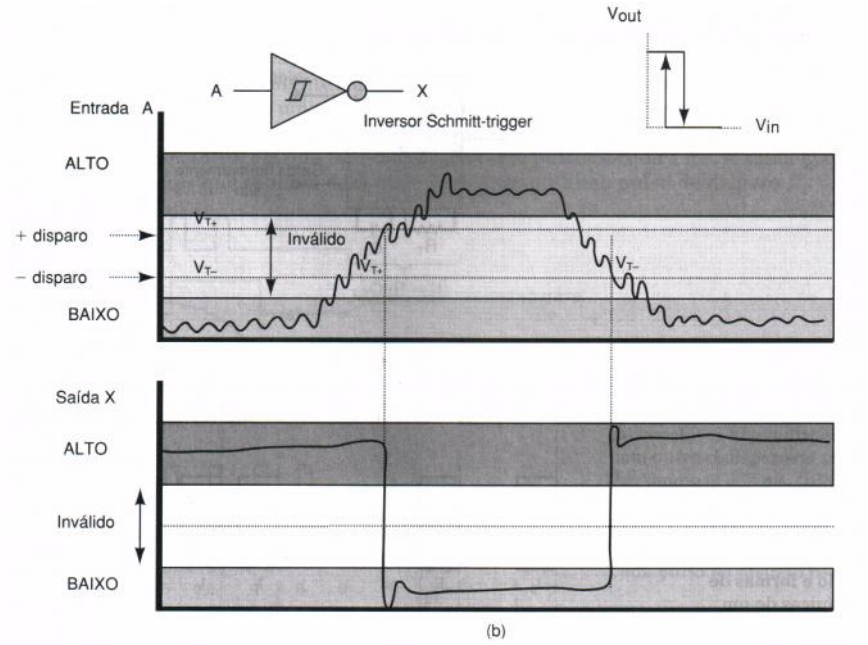
Portas lógicas tipo Schmitt-Trigger

A figura acima apresenta as portas lógicas do tipo Schmitt-Trigger.

A figura a seguir apresenta o processo de uma transição positiva e negativa de nível lógico em uma porta lógica comum. A saída responde ao ruído existente quando a transição ocorre em torno do limiar único.



Transições de nível lógico em uma porta comum.



Transições de nível lógico em uma porta Schmitt-Trigger.

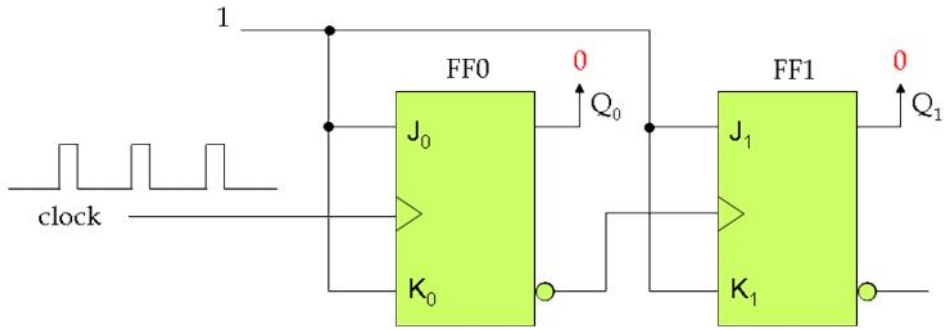
A figura acima apresenta o processo de uma transição positiva e negativa de nível lógico em uma porta lógica do tipo Schmitt-Trigger. A saída não responde ao ruído existente porque agora 'há dois limiares diferentes: Um alto para a transição de subida (+ disparo) e um baixo para a transição de descida (- disparo).

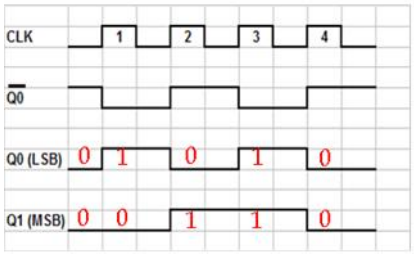
Contadores Assíncronos

O termo assíncrono se refere aos eventos que não tem uma relação temporal fixa entre si e, geralmente, não ocorrem simultaneamente.

Um contador assíncrono é aquele no qual os flip-flops (FF) que constituem o contador não mudam de estado exatamente ao mesmo tempo porque eles não têm um pulso de clock em comum.

Contador Binário Assíncrono



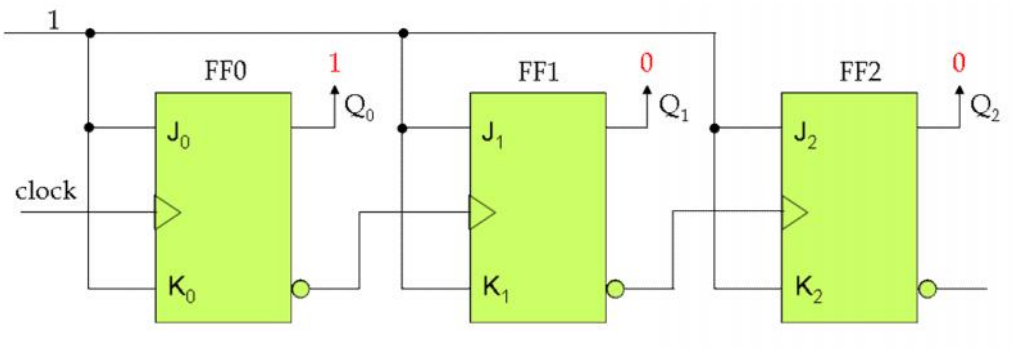


Pulso de Clock	Q_1	Q_0
Condição inicial	0	0
1	0	1
2	1	0
3	1	1
4	0	0

Contador assíncrono de 2 bits.

O FF0 está sempre na condição de *toggle* e, consequentemente, muda de estado a cada pulso de *clock*. O FF1 tmabém está sempre na condição de *toggle*, mas seu *clock* é comandado pela saída Q0 complementar de FF0.

Com 2 FFs, a contagem vai de 0 a $2^2 - 1$.



Pulso de Clock	Q_2	Q_1	Q_0	Pulso de Clock	Q_2	Q_1	Q_0
Condição inicial	0	0	0	5	1	0	1
1	0	0	1	6	1	1	0
2	0	1	0	7	1	1	1
3	0	1	1	8	0	0	0
4	1	0	0	9	0	0	1

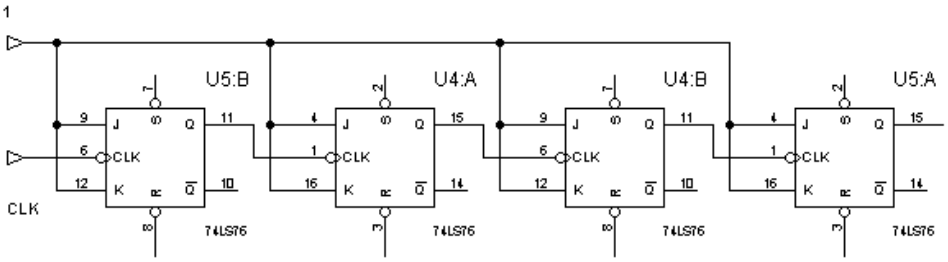
Contador assíncrono de 3 bits.

Adicionando um FF a mais, a contagem passar a ser de 0 a $2^3 - 1$. A análise do circuito continua sendo feita da mesma forma que a anterior.

Os contadores assíncronos são normalmente chamados de contadores ondulantes ou *ripple counters* pela seguinte razão: O efeito do pulso de clock na entrada é "sentido" primeiro pelo FF0. Esse efeito não chega ao FF1 imediatamente devido ao atraso de propagação através de FF0 (da ordem de 5ns a 10ns). Então existe um atraso de propagação através de FF1 até que FF2 possa ser "disparado". Portanto, o efeito do pulso de clock na entrada "ondula" através do contador por todos os flip-flops.

Este atraso, que é cumulativo, em um contador assíncrono é a principal desvantagem em muitas aplicações porque limita a freqüência do clock e causa problemas de sincronismo.

Qual a freqüência máxima de *clock* para o contador mostrado na figura abaixo, sabendo-se que cada flip-flop possui um atraso de propagação de 10ns?



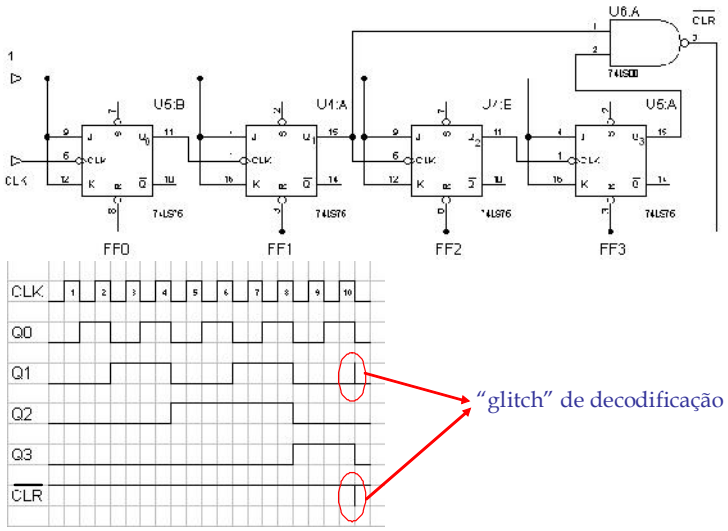
Contador assíncrono de 4 bits.

Atraso total = 4 x 10 = 40 ns

fmax do clock = 1/40ns = 25 MHz

Contador de década assíncrono

O módulo de um contador é o número de estados únicos pelos quais o contador estabelece uma seqüência. O número máximo de estados possíveis (módulo máximo) de um contador é 2^n, onde n é o número de flip-flops do contador. Os contadores podem ser projetados para ter um número de estados em sua seqüência que é menor que o valor máximo de 2^n. Esse tipo de seqüência é denominada de *seqüência truncada*.

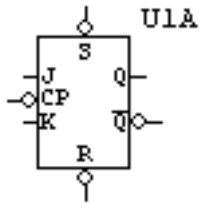


Contador de década assíncrono (truncado).

Na figura anterior temos o contador de década. Este contador conta de 0 a 9 (apesar de haver 4 FFs). A porta AND decodifica o valor 10 (em binário) e reinicia todos os FFs obrigando-se a irem para zero através de suas entradas assíncronas *RESET*.

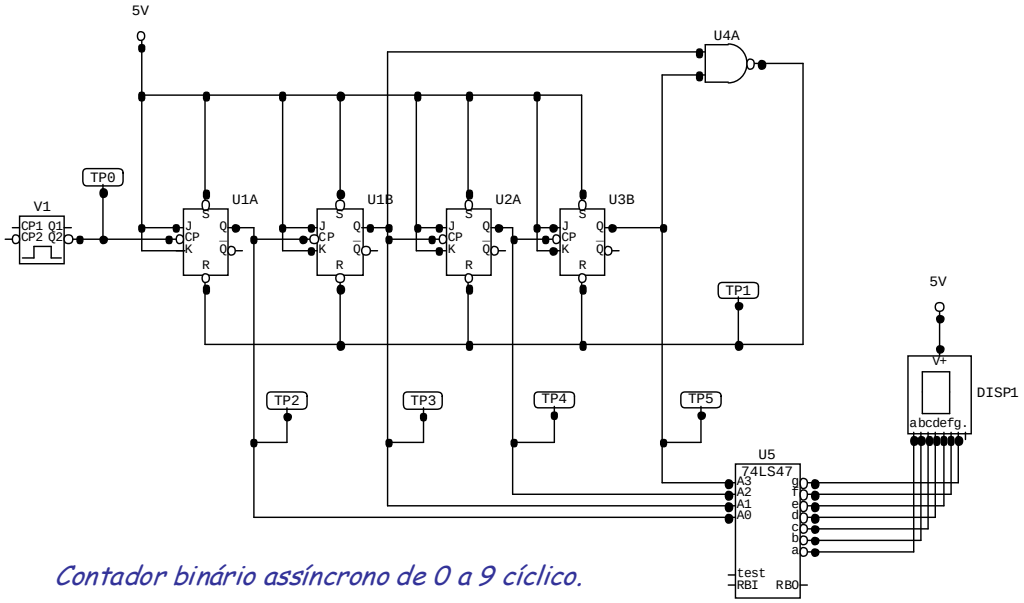
Logo após o contador chegar a contagem 1010_2 (10_{10}), a porta lógica AND vai a nível lógico ZERO e reinicia todos os FFs. Isto gera os chamados *glitchs de decodificação*. No caso, em Q1 e CLEAR.

Desenhe o esquema de um contador binário assíncrono, utilizando FFs JK (mostrado abaixo), que conte de 0 a 9 indefinidamente:



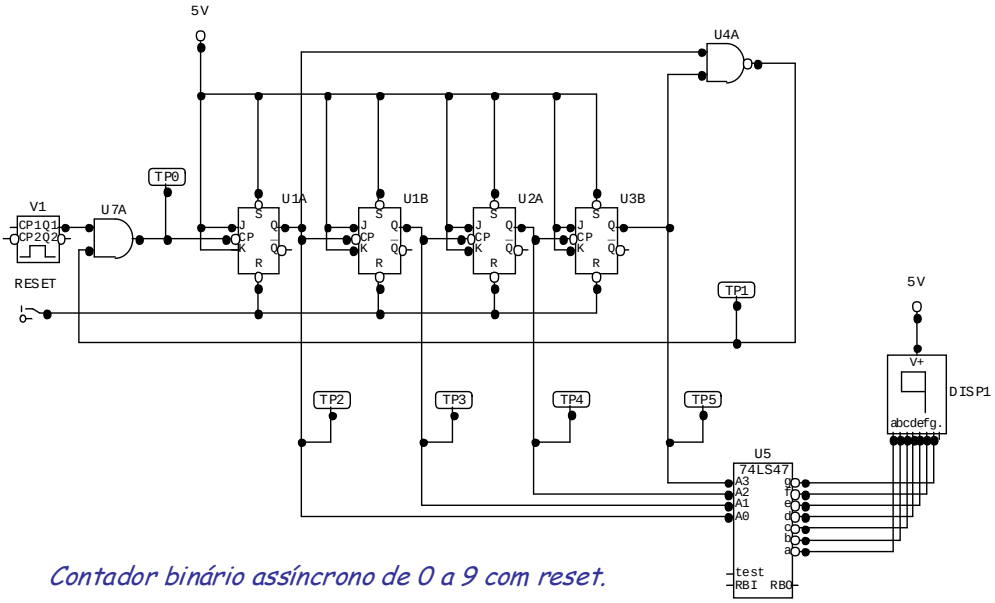
Modifique o circuito anterior para que ele pare de contar quando chegar ao número 9. Incluir uma entrada assíncrona que permita reiniciar a contagem a qualquer momento.

Resposta do exercício anterior:



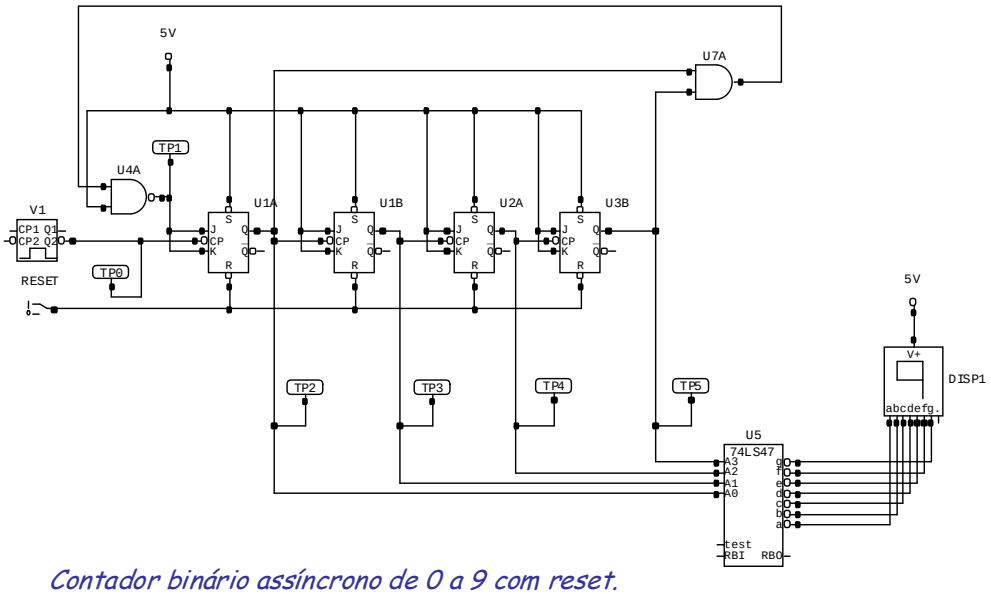
Contador binário assíncrono de 0 a 9 cíclico.

Executar o arquivo Contador_0a9_ciclico.ckt no CircuitMaker !



Contador binário assíncrono de 0 a 9 com reset.

Executar o arquivo Contador_0a9b_ciclico.ckt no CircuitMaker !



Contador binário assíncrono de 0 a 9 com reset.

(Outra solução possível !)

Executar o arquivo Contador_0a9c_ciclico.ckt no CircuitMaker !

Exercícios

Tocci: 7.2, 7.3, 7.5 a 7.11.

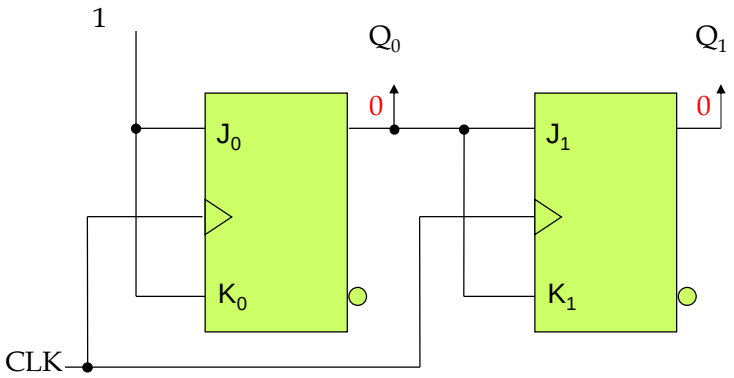
Contadores Síncronos

O termo síncrono se refere aos eventos que tem uma relação de tempo fixa entre si e, geralmente, ocorrem simultaneamente.

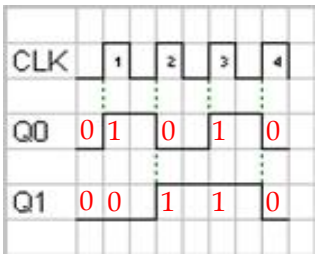
Um contador síncrono é aquele no qual os flip-flops (FFs) que constituem o contador mudam de estado exatamente ao mesmo tempo, porque eles tem uma entrada de clock em comum.

Contador Binário Síncrono de 2 bits

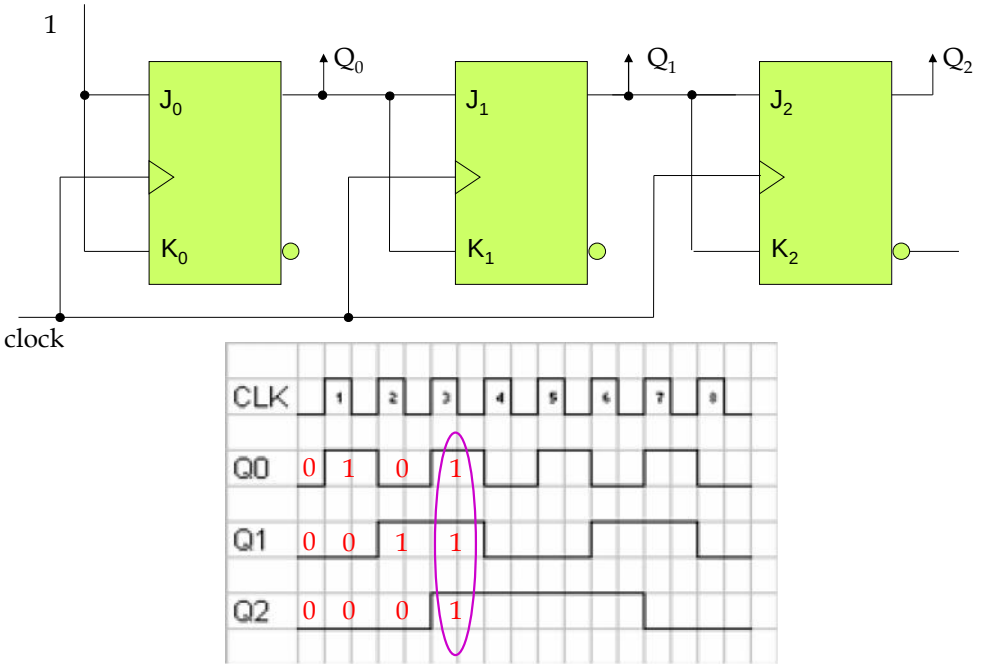
O contador a seguir (com dois flip-flops) é síncrono e conta de 0 a 3.



Pulso de Clock	Q_1	Q_0
estado inicial	0	0
1	0	1
2	1	0
3	1	1
4	0	0

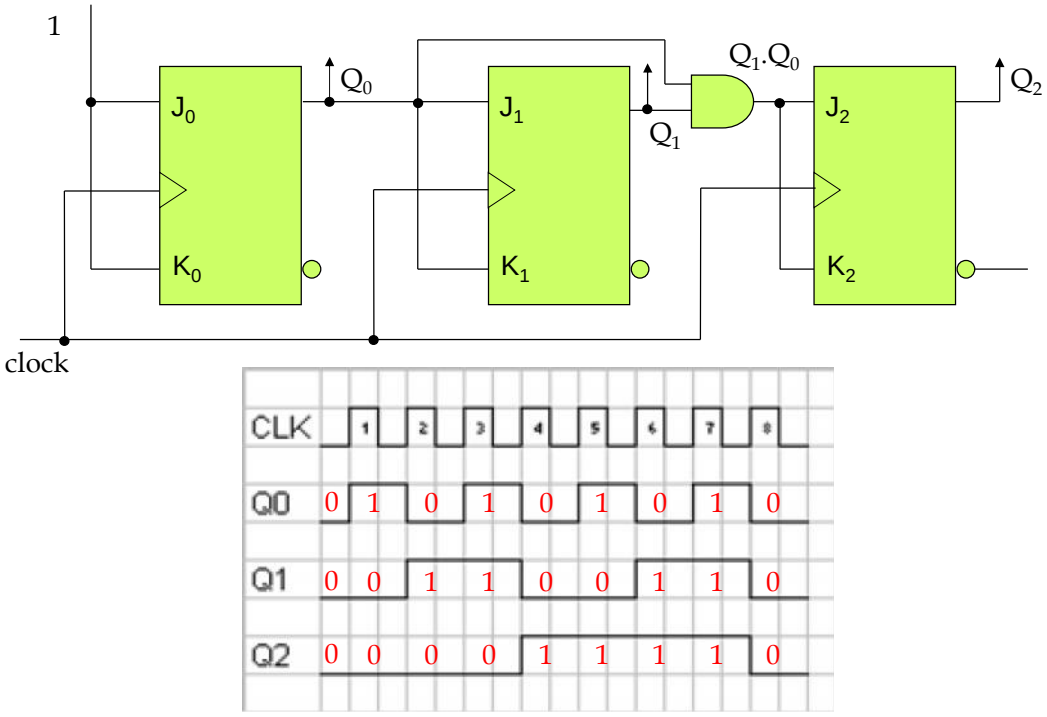


Como no caso do contador assíncrono, vamos aumentar o circuito em mais um flip-flop e verificar o que acontece:



Desta vez, a contagem sofre um erro após o terceiro pulso de clock. Como resolver?

A inserção de um porta lógica AND entre Q1 e Q0 com sua saída conectada em J2 resolve o problema.



Mais à frente estudaremos projetos de circuitos síncronos e veremos que a porta AND não assume aquela posição por tentativa e erro, e sim por consequência do projeto mesmo.

Projeto de Contadores Síncronos

Em geral, os circuitos sequenciais podem ser classificados em dois tipos quanto à sua saída

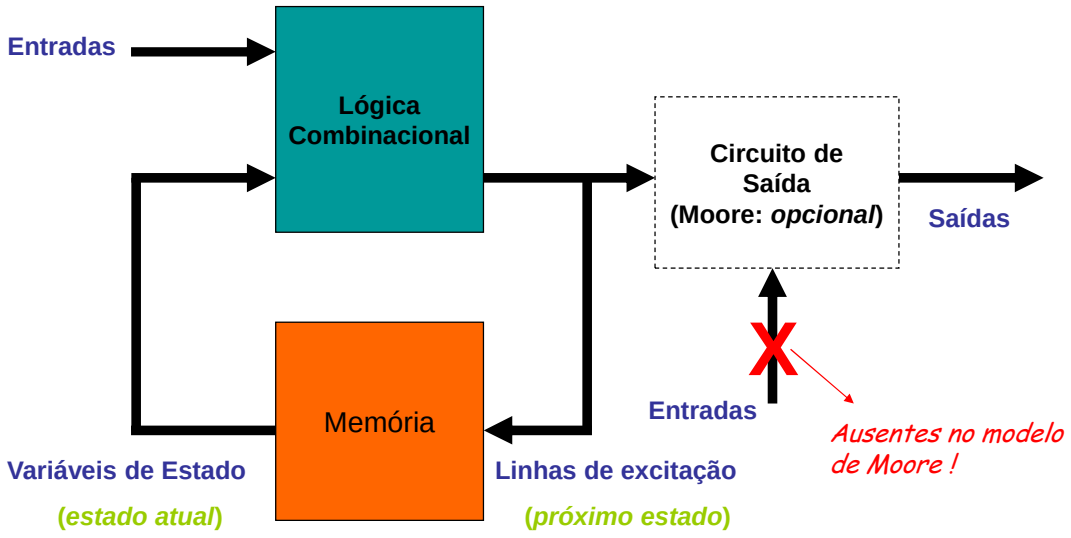
Máquina de Moore

São os circuitos sequenciais nos quais a(s) saída(s) depende(m) apenas do estado atual do circuito (valor na saída Q dos flip-flops).

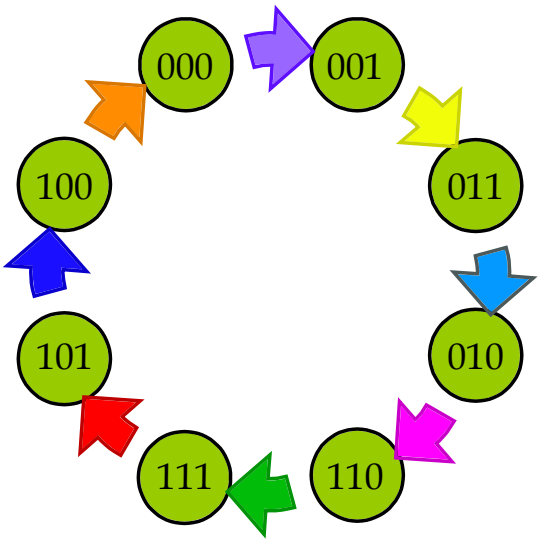
Máquinas de Mealy

São os circuitos sequenciais nos quais a(s) saída(s) depende(m) do estado atual do circuito e da(s) entrada(s).

Modelo Geral de um Circuito Seqüencial



Passo 1: Diagrama de Estados



Um *diagrama de estados* mostra a progressão de estados através dos quais o contador avança quando recebe um pulso de clock. Este exemplo, em particular, não tem outras entradas além do clock nem outras saídas além das saída obtidas de cada FF do contador. (contador Gray de 3 bits)

Passo 2: Tabela do Próximo Estado

ESTADO ATUAL			PRÓXIMO ESTADO		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	1	0	1	0
0	1	0	1	1	0
1	1	0	1	1	1
1	1	1	1	0	1
1	0	1	1	0	0
1	0	0	0	0	0

Tabela de Próximo Estado para o contador Gray

O *próximo estado* é o estado para o qual o contador passa a partir do estado atual com a aplicação de um pulso de clock.

Passo 3: Tabela de Transição de Flip-Flop

Transições de Saída		Entradas do Flip-Flop	
Q_N	Q_{N+1}	J	K
0 →	0	0	X
0 →	1	1	X
1 →	0	X	1
1 →	1	X	0

Q_N : Estado Atual

Q_{N+1} : Próximo Estado

X : Estado "don't care"

Tabela de transição para um FF JK

J	K	CLK	Q
0	0	↑	Q_0
0	1	↑	0
1	0	↑	1
1	1	↑	Q_0

Tabela Verdade de um FF JK

Passo 4: Mapas de Karnaugh

Estado Atual			Próximo Estado		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	1	0	1	0
0	1	0	1	1	0
1	1	0	1	1	1
1	1	1	1	0	1
1	0	1	1	0	0
1	0	0	0	0	0

Q_2Q_1 \ Q_0	0	1
00	1	
01		
11		
10		X

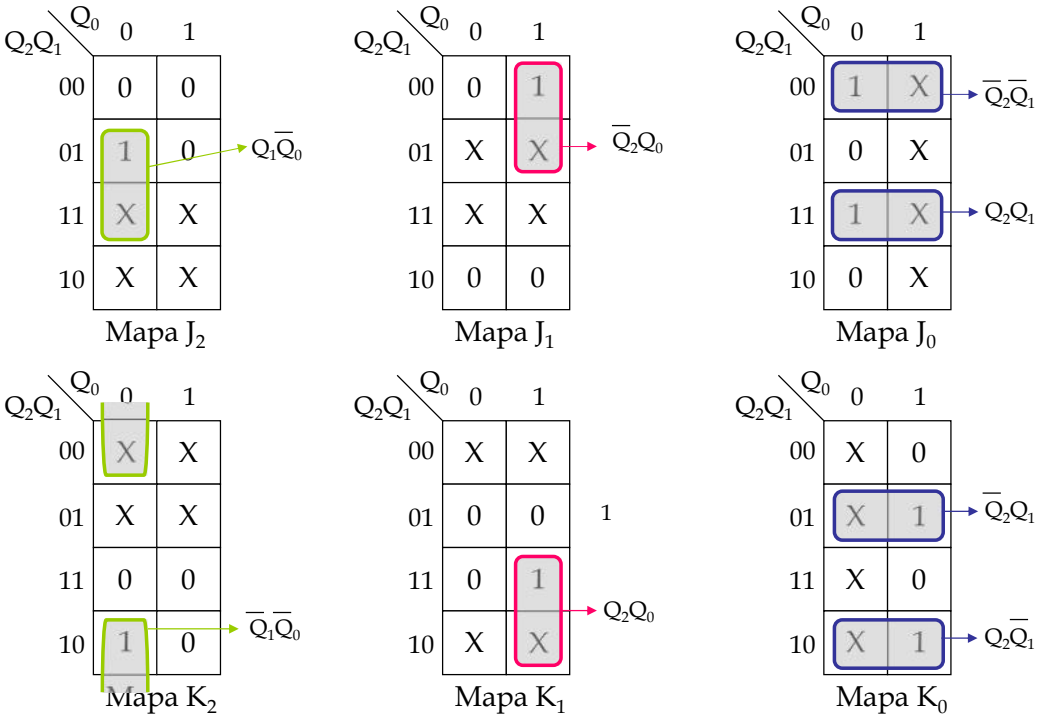
Mapa J_0

Q_2Q_1 \ Q_0	0	1
00	X	
01		
11		
10		1

Mapa K_0

Transições de Saída		Entradas do FF	
Q_N	Q_{N+1}	J	K
0 →	0	0	X
0 →	1	1	X
1 →	0	X	1
1 →	1	X	0

Completando os mapas de Karnaugh e simplificando-os:



Passo 5: Expressões Lógicas para as Entradas dos Flip-Flops

$$J_0 = Q_2.Q_1 + \overline{Q}_2.\overline{Q}_1 = \overline{Q_2 \oplus Q_1}$$

$$K_0 = Q_2.\overline{Q}_1 + \overline{Q}_2.Q_1 = Q_2 \oplus Q_1$$

$$J_1 = \overline{Q}_2.Q_0$$

$$K_1 = Q_2.Q_0$$

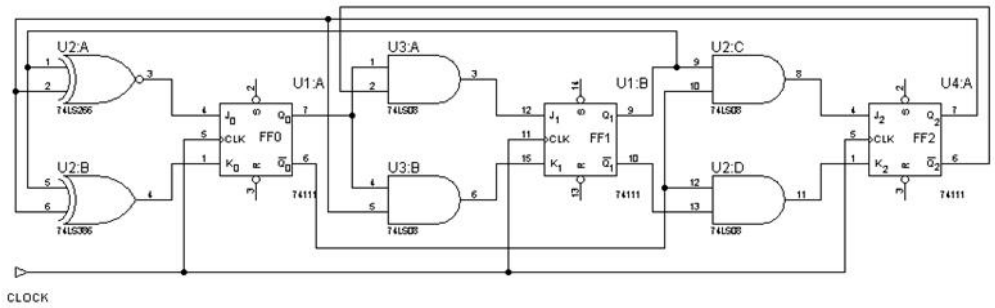
$$J_2 = Q_1.\overline{Q}_0$$

$$K_2 = \overline{Q}_1.\overline{Q}_0$$

No passo 5, temos as equações de excitação dos flip-flops.

O passo 6 é o passo de implementação do circuito projetado.

Passo 6: Implementação do Contador



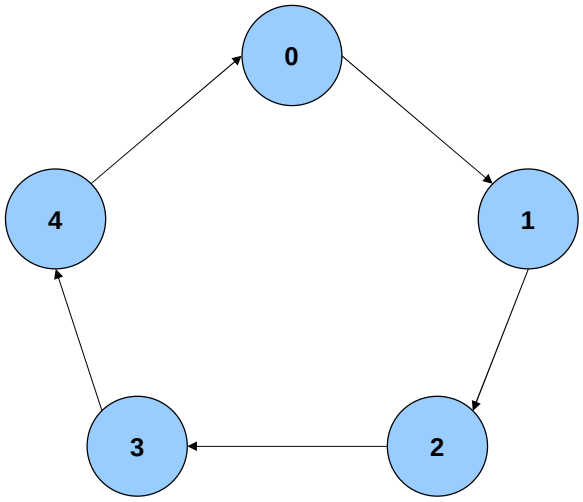
Contador Gray de 3 bits

Exercícios

Tocci: 7.43 a 7.46.

Exemplo de Projeto

Projete um contador para a sequência dada abaixo:



Como necessitamos de 3 FFs para incrementar os estados de 0 a 4, o número total de estados será de 2^3 = 8 estados.

O contador é cíclico, ou seja, após ele chegar ao último estado ele retorna ao estado inicial.

Mapas de Karnaugh

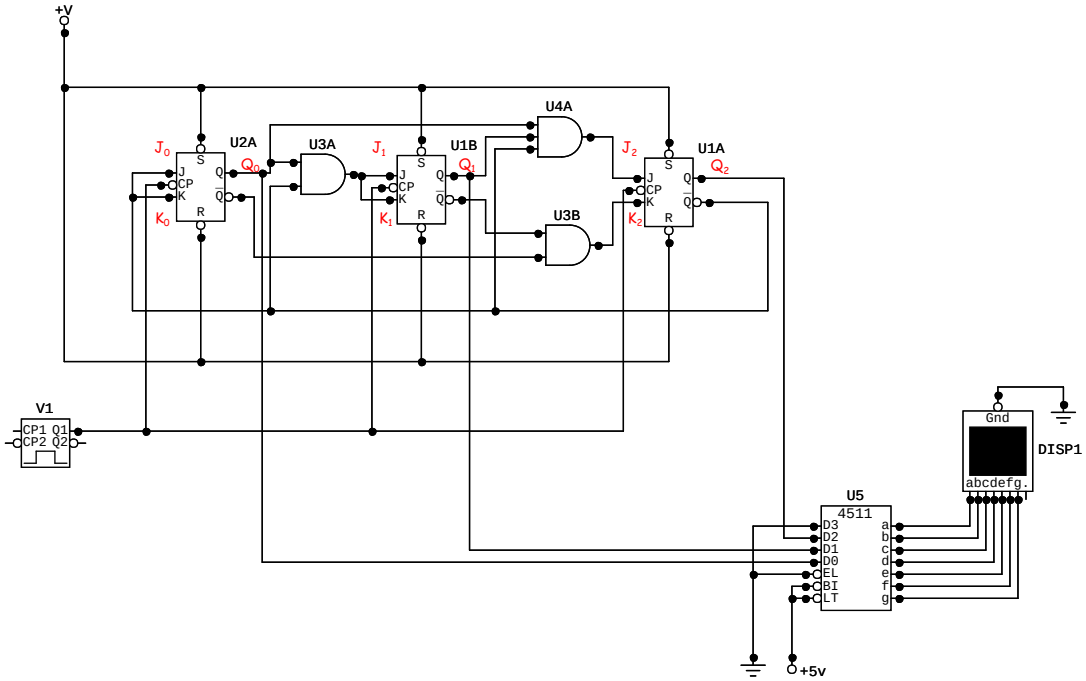
Estado Atual			Próximo Estado			J_2		K_2		J_1		K_1		J_0		K_0	
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	$\overline{Q}_0$	Q_0	$\overline{Q}_0$	Q_0	$\overline{Q}_0$	Q_0	$\overline{Q}_0$	Q_0	$\overline{Q}_0$	Q_0	$\overline{Q}_0$	Q_0
0	0	0	0	0	1	0	0	X	X	0	1	X	X	1	X	X	1
0	0	1	0	1	0	0	1	X	X	X	X	0	1	1	X	X	1
0	1	0	0	1	1												
0	1	1	1	0	0	X		1		0		X		0		X	
1	0	0	0	0	0	*Mapas de Karnaugh*											

$\overline{Q}_2\overline{Q}_1$
$\overline{Q}_2Q_1$
Q_2Q_1
$Q_2\overline{Q}_1$

$J_0 = \overline{Q}_2$
$K_0 = \overline{Q}_2$
$J_1 = Q_0.\overline{Q}_2$
$K_1 = Q_0.\overline{Q}_2$
$J_2 = Q_0.Q_1.\overline{Q}_2$
$K_2 = \overline{Q}_0.\overline{Q}_1$

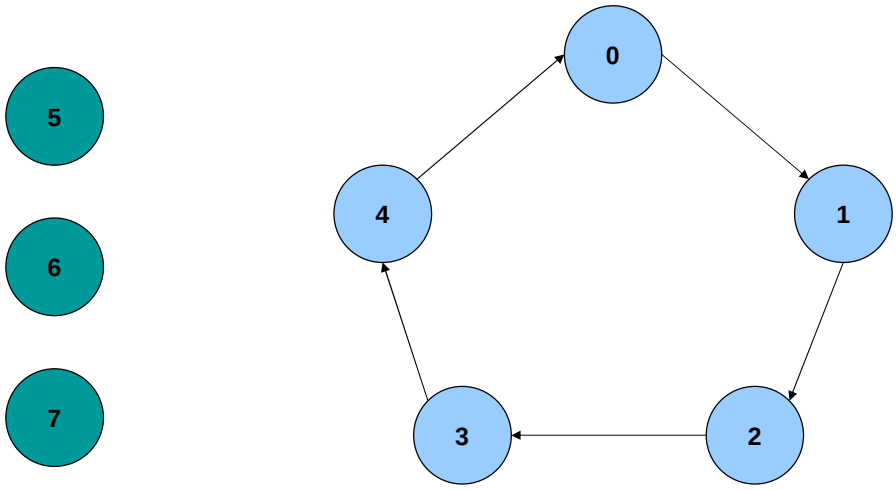
Transições de Saída		Entradas do FF	
Q_N	Q_{N+1}	J	K
0	0 →	0	X
0	1 →	1	X
1	0 →	X	1
1	1 →	X	0

Implementando o circuito, temos:



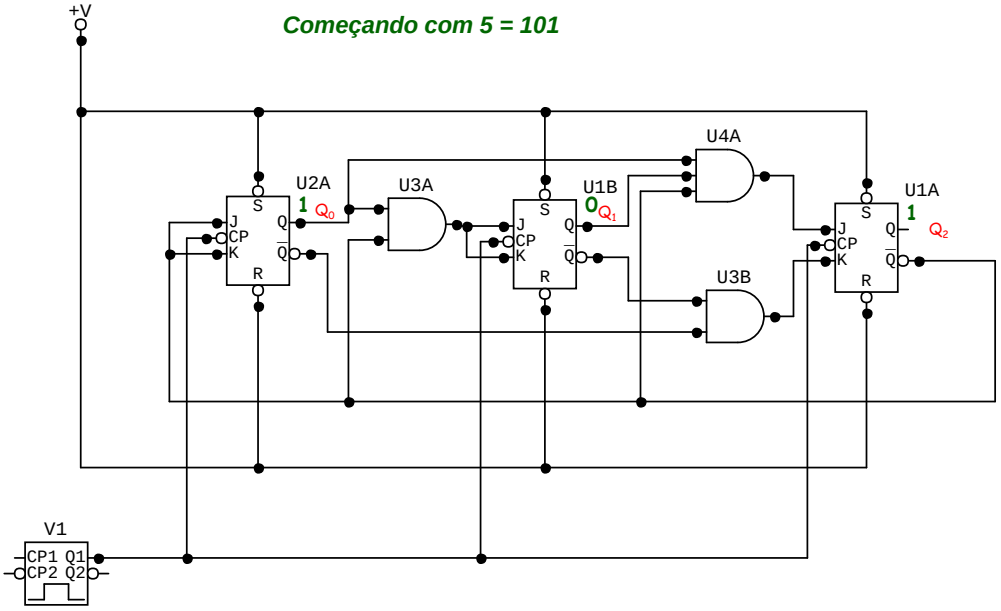
Contador 0 a 4 – (Arquivo contador02.ckt)

O que poderá acontecer se o contador iniciar sua contagem ou, por algum motivo, passar para um dos estados que não foram tratados no projeto?



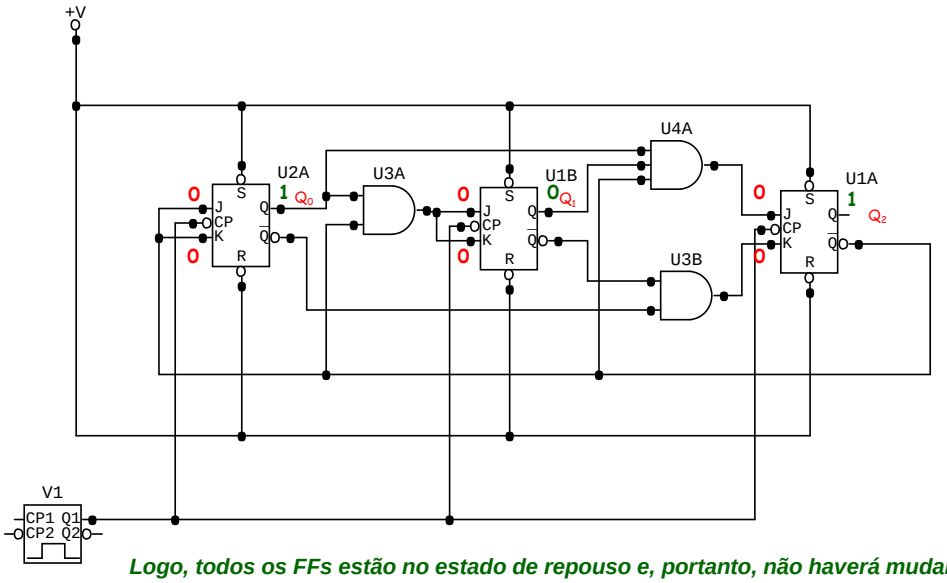
Damos como condição inicial do circuito um dos estados que não fizeram parte do projeto, como por exemplo o estado de número 5.

Para responder a esta pergunta, vamos analisar como o circuito se comporta quando em um destes estados.



Para esse estado, verificamos quais são as entradas presentes em cada um dos flip-flops. Isto nos permite verificar se o flip-flop está em *repouso, set, reset* ou *toggle*, o que, por sua vez, nos permite prever para que estado cada um dos flipflops irá na chegada do próximo pulso de clock.

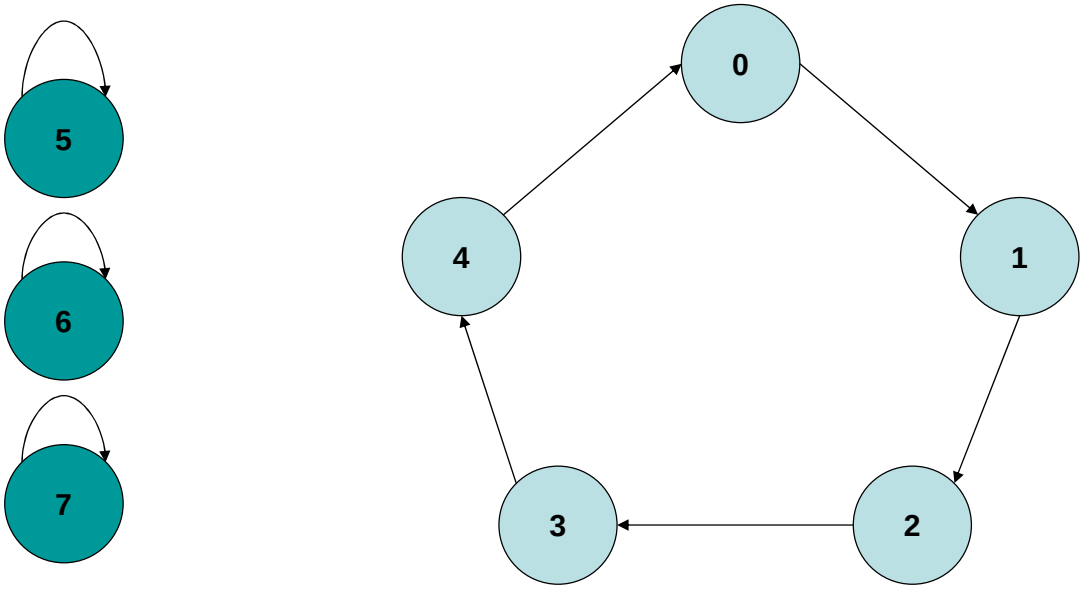
Para este estado inicial, as entradas dos flip-flops serão:



Logo, todos os FFs estão no estado de repouso e, portanto, não haverá mudanças.

Como todos os flip-flops estão em *repouso*, podemos concluir que não haverá nenhuma mudança de estados no futuro. Ou seja, o circuito permanecerá no estado 5 para sempre. Em palavras menos técnicas, podemos dizer que o contador "travou". Apenas um reset ou o corte de energia do circuito poderá tirá-lo dessa situação.

Fazendo a mesma análise para os demais estados que ficaram de fora do projeto, temos que a mesma situação se repete.

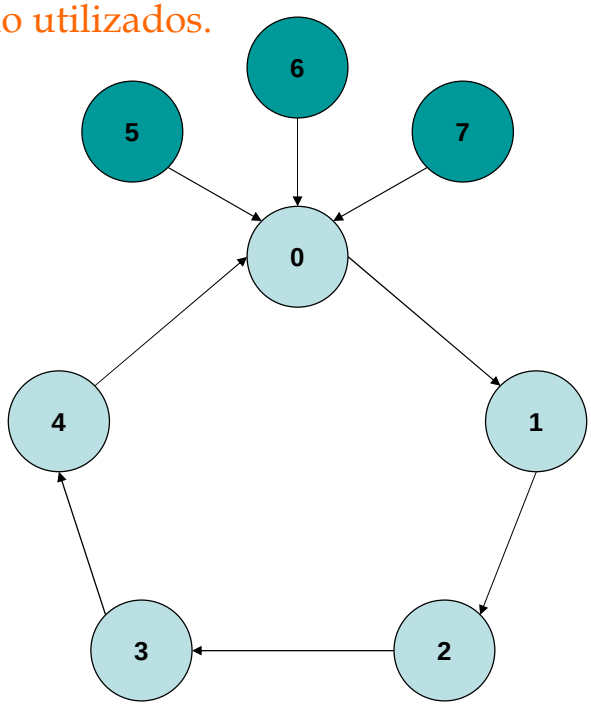


Se o contador iniciar ou passar para um dos estados: 5, 6 e 7, ele irá "travar" o seu funcionamento.

Veja o arquivo "contador03.ckt" – Realize um SET no FF0 quando o contador estiver em 4 e veja o que acontece.

Na prática, nunca é bom deixar os estados restantes (aqueles que não serão utilizados) sem uma definição de "próximo estado".

Refaça o projeto do contador anterior, desta vez "amarrando" o estados não utilizados.



Projeto de contadores com FFs do tipo D

Projete um contador síncrono para a sequência mostrada pelo diagrama de estados abaixo. Utilize Flip-Flops do tipo D.

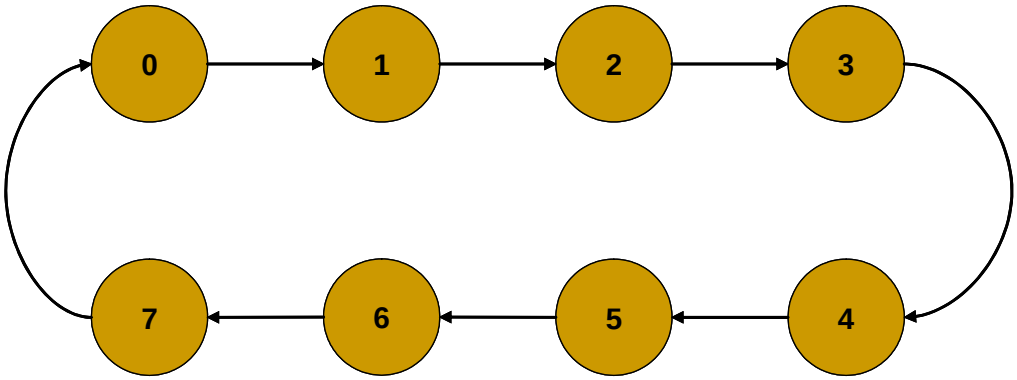


Tabela de Sequência de Estados

ESTADO ATUAL			PRÓXIMO ESTADO		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Tabela de Transição do FF D

Transições de Saída		Entrada do FF
Q_N	Q_{N+1}	D
0	0 →	0
0	1 →	1
1	0 →	0
1	1 →	1

ESTADO ATUAL			PRÓXIMO ESTADO		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Transições de Saída		Entrada do FF
Q_N	Q_{N+1}	D
0	0 →	0
0	1 →	1
1	0 →	0
1	1 →	1

	D_2		D_1		D_0	
$Q_2Q_1Q_0$	0	1	0	1	0	1
00	0	0	0	1	1	0
01	0	1	1	0	1	0
11	1	0	1	0	1	0
10	1	1	0	1	1	0

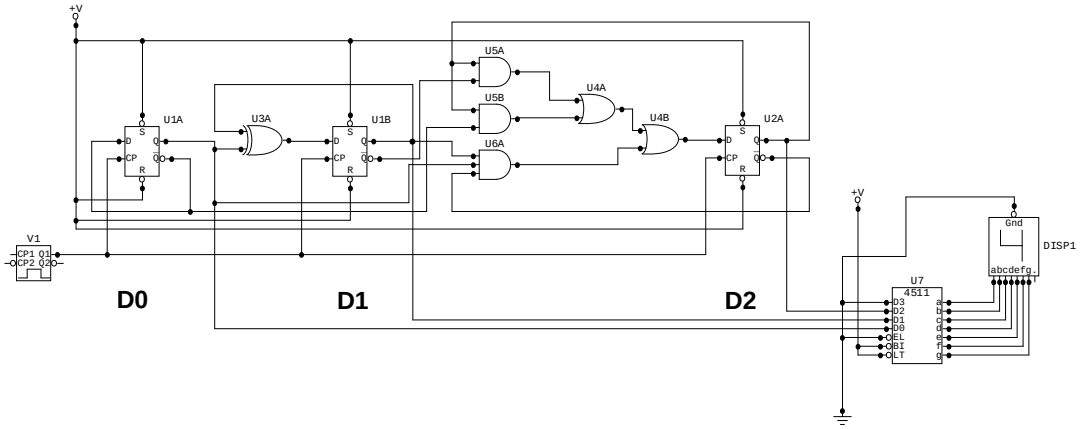
$D_2 = Q2.\overline{Q1} + Q2.\overline{Q0} + \overline{Q2}.Q1.Q0$

$D_1 = Q1.\overline{Q0} + \overline{Q1}.Q0 = Q1 \oplus Q0$

$D_0 = \overline{Q0}$

Mapas de Karnaugh

Circuito



Veja a simulação do circuito no CircuitMaker através do arquivo "ContadorFFD.ckt"

Máquinas de Estado

Até agora, vimos projetos de contadores (um tipo particular de máquina de estados) cujas saídas são derivadas diretamente das saídas dos flip-flops. Melhor dizendo, as saídas são os próprios estados do sistema

Vamos tratar agora de máquinas de estado cuja(s) saída(s) é/são realizada(s) indiretamente (por uma lógica combinacional) dos estados do sistema.

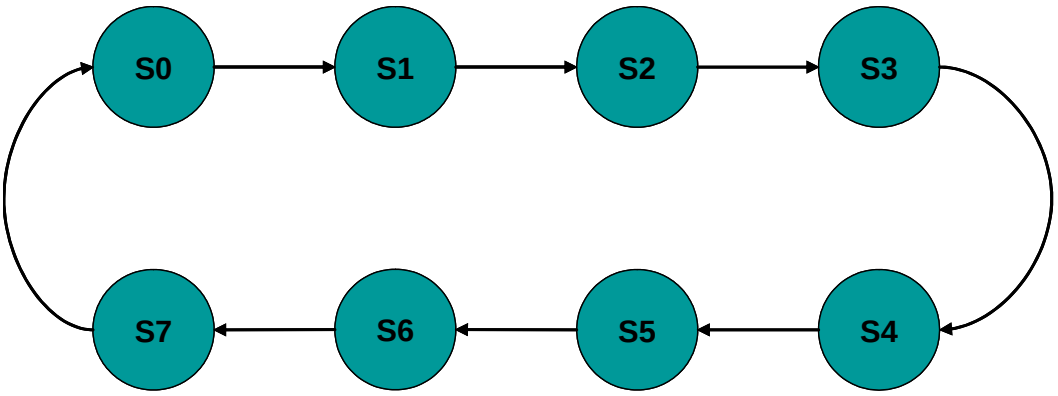


Figura x: Diagrama de estados cuja(s) saída(s) geralmente é/são o próprio estado do sistema.

Note que, neste exemplo, os valores binários do estado não são importantes. Então deixamos a definição dos valores de S0 a S7 em aberto por enquanto.

Como já foi visto anteriormente, neste diagrama de estados não há entradas e as saídas (se houverem) são os próprios estados do sistema (saídas dos flip-flops).

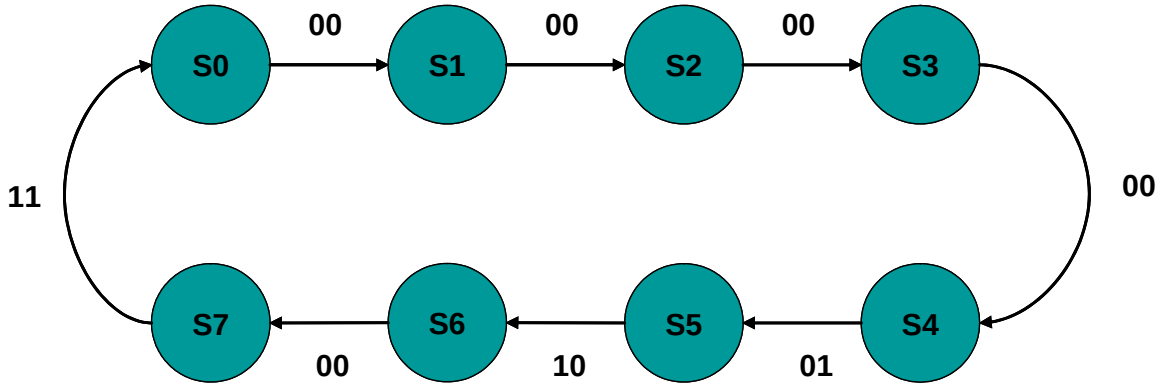


Figura x: Diagrama de estados com entradas (E1E0).

Neste exemplo, temos uma notação que mostra os valores de entrada necessários para o sistema passar de um estado para o outro. Estes valores de entradas necessários são geralmente indicados nas "transições entre estados".

No exemplo, temos duas entradas em cada transição (E_1E_0). Para saírmos de S3 até S4, temos que (E_1E_0 = 00). Para saírmos de S4 para S5, temos que (E_1E_0 = 01).

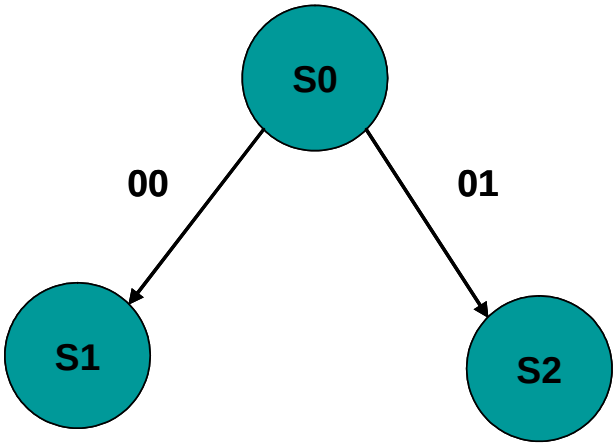


Figura x: Diagrama de estados com múltiplos percursos.

Veja este exemplo. Se o estado atual for S0, então teremos duas possibilidades de Estado Futuro(Próximo Estado). Tudo dependerá dos valores das entradas (E_1E_0).

Se (E_1E_0 = 01), então o Estado Futuro será S2. Se (E_1E_0 = 00), então o Estado Futuro será S1.

E as demais possibilidades de Entradas (E_1E_0 = ?) Não irão provocar nenhuma transição e o próprio Estado Atual será o Estado Futuro.

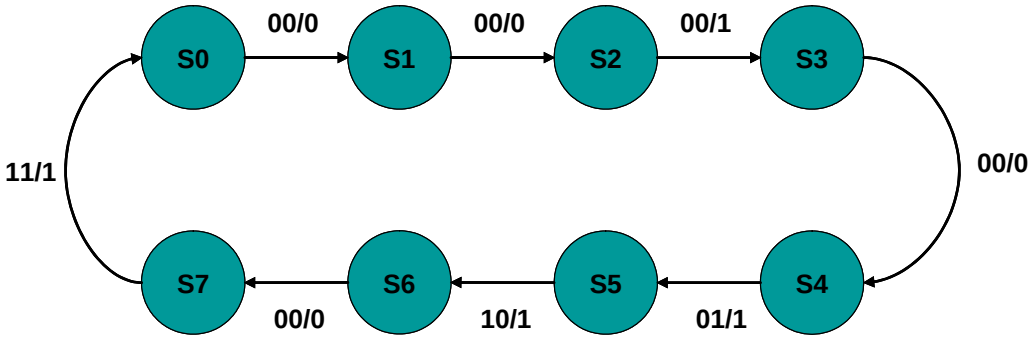


Figura x: Notação para máquinas de Mealy.

Olhe atentamente este exemplo agora. As transições incluem tanto os valores de entrada necessários para que ela ocorra, como um valor de "saída" do sistema. Esta notação é utilizada para as chamadas Máquinas de Mealy.

Notação para Máquinas de Mealy:

Entrada(s) / Saída(s)

Vejamos agora a notação utilizada para as máquinas de Moore:

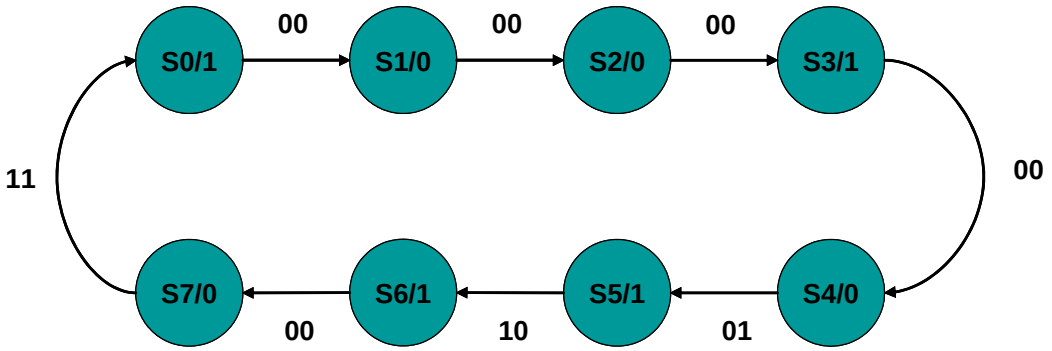


Figura x: Notação para máquinas de Moore.

Temos agora a seguinte notação nos *nodes* ("bolinhas")

Notação para Máquinas de Moore:

Estado Atual / Saída(s)

E a seguinte notação nas Transições ("setinhas")

Notação:

Entrada(s)

Compare as duas notações atentamente !

Exemplo de projeto: Detector de sequências

Considere um circuito digital que aceita uma sequência arbitrária de 0s e 1s como entrada. A cada passo, o circuito gera uma saída da seguinte forma:

- Se 3 ou mais 1s consecutivos forem detectados na entrada, então o circuito gera uma saída S = 1;
- Em qualquer outro caso, o circuito gera uma saída S = 0.

Projete uma máquina de estado (Máquina de Moore) para este circuito.

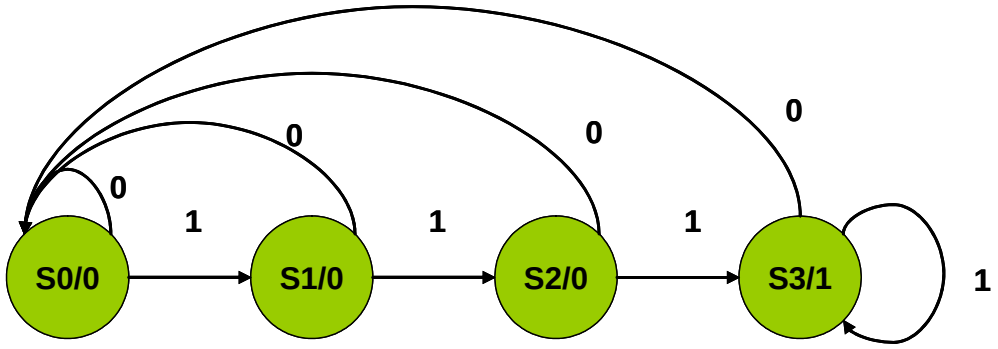


Diagrama de Sequência de Estados

Figura x: Diagrama de estados para o detector de seqüências.

Vamos começar definindo valores para S0, S1, S2 e S3. Como temos 4 estados, necessitaremos de 2 flip-flops. Vamos utilizar FFs do tipo D para o projeto.

ESTADO	Q_1Q_0
S0	00
S1	01
S2	10
S3	11

Agora, vamos criar Tabela de Estado Atual/Futuro:

ESTADO ATUAL Q_1Q_0E	ESTADO FUTURO Q_1Q_0	SAÍDA S
000	00	0
001	01	0
010	00	0
011	10	0
100	00	0
101	11	0
110	00	1
111	11	1

ESTADO ATUAL Q_1Q_0E	ESTADO FUTURO Q_1Q_0	SAÍDA S
000	00	0
001	01	0
010	00	0
011	10	0
100	00	0
101	11	0
110	00	1
111	11	1

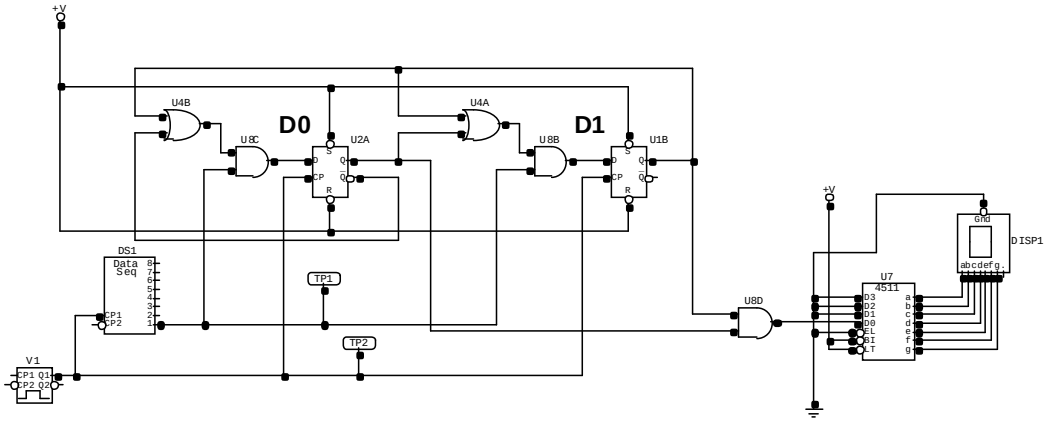
$D_0 = E.Q_1 + E.\overline{Q_0} = E.(\overline{Q_0} + Q_1)$

$D_1 = E.Q_0 + E.Q_1 = E.(Q_0 + Q_1)$

$S = Q_1.Q_0$

	D_1		D_0		S	
Q_1Q_0 \ E	0	1	0	1	0	1
00	0	0	0	1	0	0
01	0	1	0	0	0	0
11	0	1	0	1	1	1
10	0	1	0	1	0	0

A implementação do circuito então, fica:



Contadores Binários em VHDL

Existem diversas abordagens para criarmos contadores binários em VHDL:

1. Podemos codificar as equações Booleanas de excitação do contador diretamente como sinais concorrentes;
2. Podemos descrever o comportamento do contador;
3. Podemos utilizar uma instrução CASE para implementar o diagrama de estados do contador, ou;
4. Podemos utilizar um contador pré-definido na biblioteca LPM (*Library of Parameterized Modules).*

Vejamos o exemplo a seguir de um contador intitulado "ct_simp.vhd":

```
ENTITY ct_simp IS
PORT(
        clk : IN BIT;
        clear : IN BIT;
        q : OUT INTEGER RANGE 0 TO 255);
END ct_simp;

ARCHITECTURE a OF ct_simp IS
BEGIN
        PROCESS (clk, clear)
                VARIABLE count : INTEGER RANGE 0 TO 255;
        BEGIN
                IF (clear = '0') THEN
                        count := 0;
                ELSE
                IF (clk'EVENT AND clk = '1') THEN
                        count := count + 1;
                END IF;
        END IF;
        q <= count;
        END PROCESS;
END a;
```

Controladores

Transferências entre Registradores

As operações elementares, lógicas e aritméticas, que podem ser efetuadas sobre palavras lógicas são muito simples.

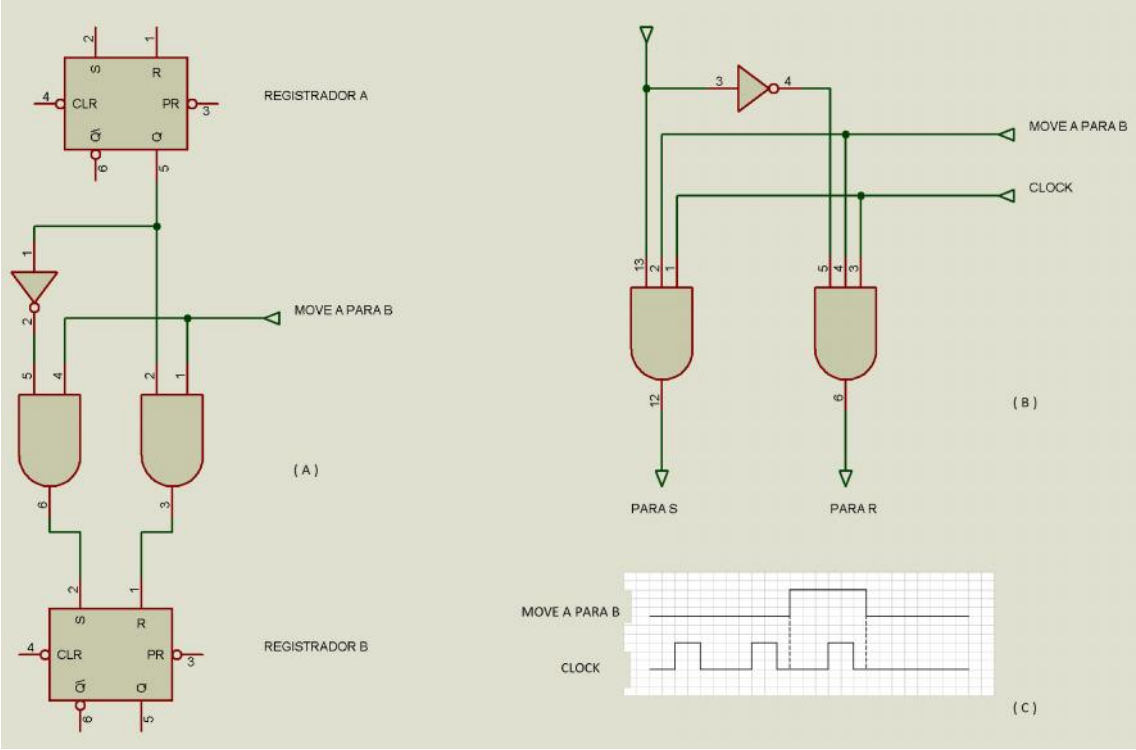


Figura x: (a) Elevação do terminal de controle “Move A para B” para lógica 1 brevemente movimenta o conteúdo do registrador A para o registrador B. (b) Um relógio é acrescentado para operação síncrona. (c) Temporização da operação de movimentar.

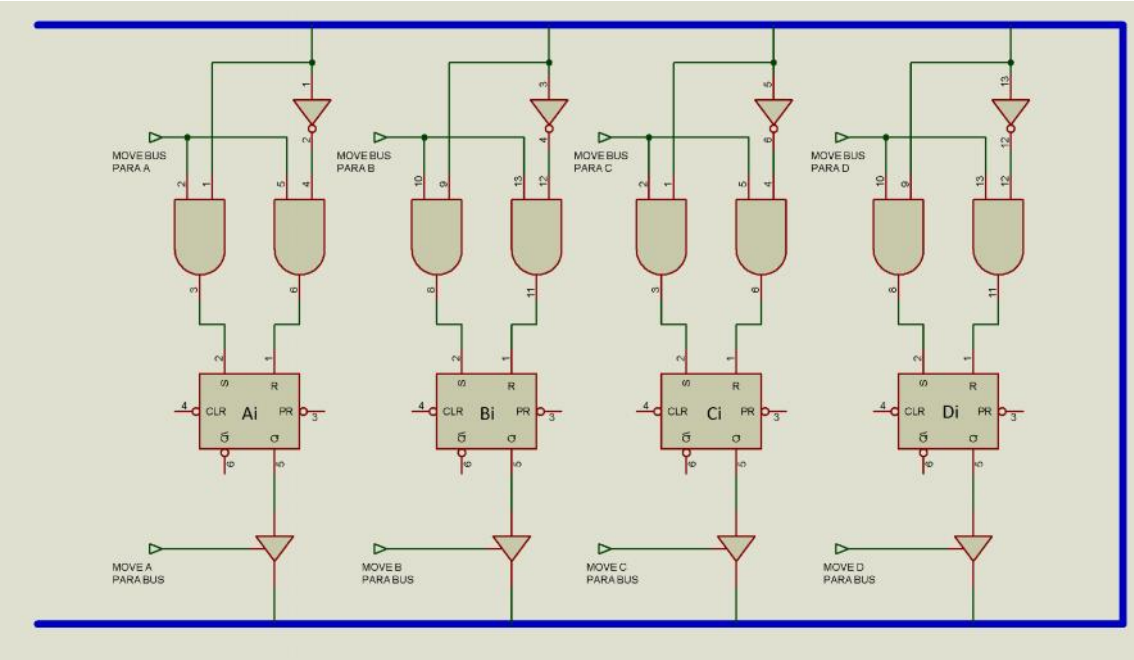


Figura x: Inúmeros registradores compartilham um bus comum. Elevando-se para 1 lógico um controle de entrada e um controle de saída de cada vez, as transferências entre registradores são realizadas.

Outras Operações

Complementação

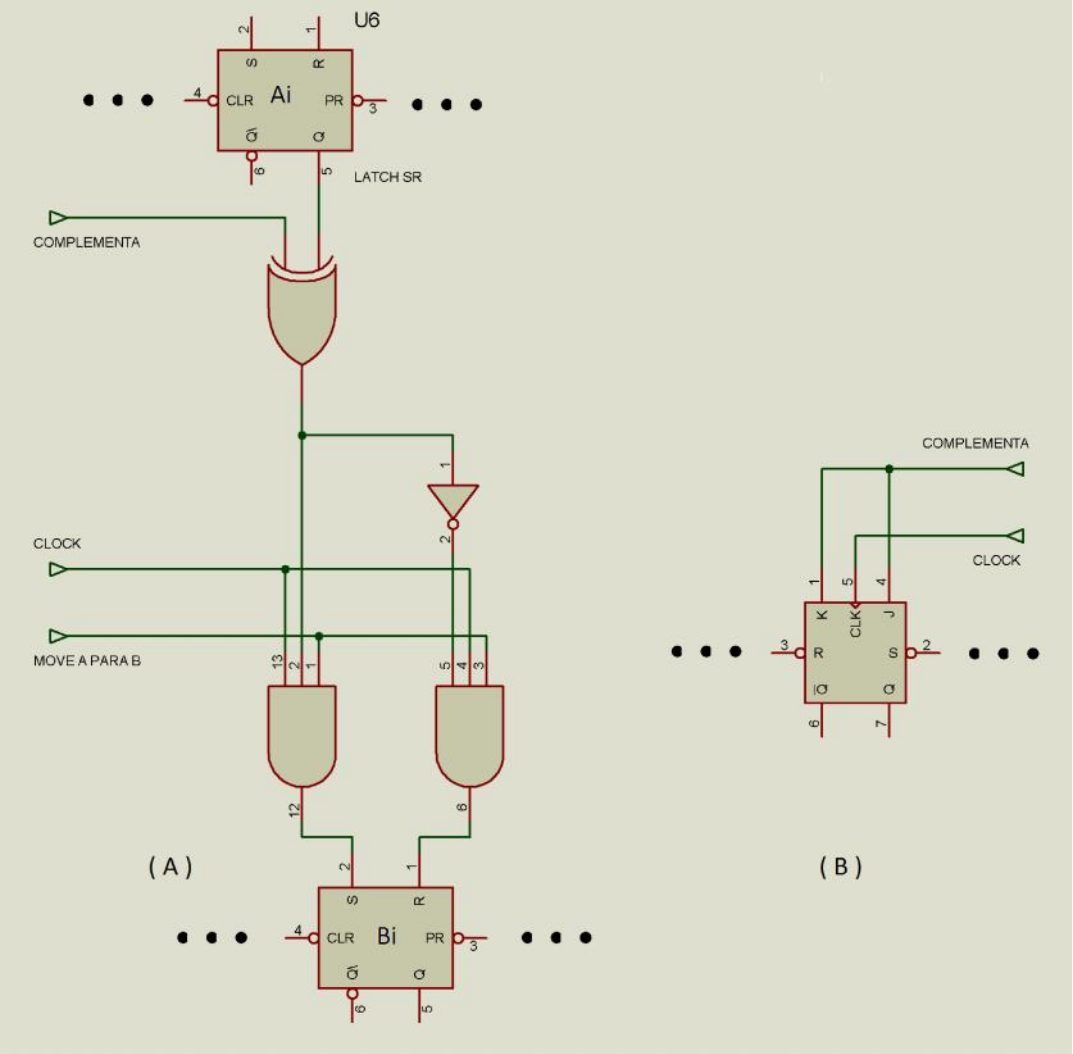


Figura x: (a) Um arranjo que permite uma transferência do registrador A para o registrador B do conteúdo de A (Complementa = 0) ou o complemento de A, bit por bit, (Complemento = 1). (b) Um arranjo que possibilita a complementação sem transferência.

Registrador Sensível a Comandos Múltiplos

Comando	Símbolo
1. Escrever a palavra do *bus* no registrador (*write*)	W
2. Ler a palavra do registrador no *bus* (*read*)	R
3. Incrementar o registrador	I

4. Complementar o registrador	*C*
5. Limpar o registrador de modo que todos os Qs sejam zero (*reset*)	Z

Um Controlador Simples

Implementação do Controlador

O Controlador do Registrador de Deslocamentos

Resposta Condicional de Controladores

Sequência para Subtração

Um Computador Simples

Operação do Computador

Projeto do Controlador do Computador

Interrupção

Confirmação de Conexão (Handshaking)

Circuitos Monoestáveis, biestáveis e astáveis

Existem alguns circuitos que podem ser classificados como um dentre as 3 classes principais de nosso interesse em eletrônica digital, segundo os pontos de equilíbrio:

1. Circuitos Monoestáveis
2. Circuitos Biestáveis
3. Circuitos Astáveis

Circuitos Monoestáveis

São circuitos eletrônicos que possuem um único ponto de equilíbrio ou estado de equilíbrio. Qualquer situação ou condição inicial sobre o circuito, ele tende a voltar ao seu estado de equilíbrio normal.

Circuitos Biestáveis

São circuitos que apresentam dois pontos de equilíbrio ou estados de equilíbrio. Um exemplo de circuito biestável é o FLIP-FLOP. Pode ficar indefinidamente no estado 0 (baixo) ou indefinidamente no estado 1 (alto).

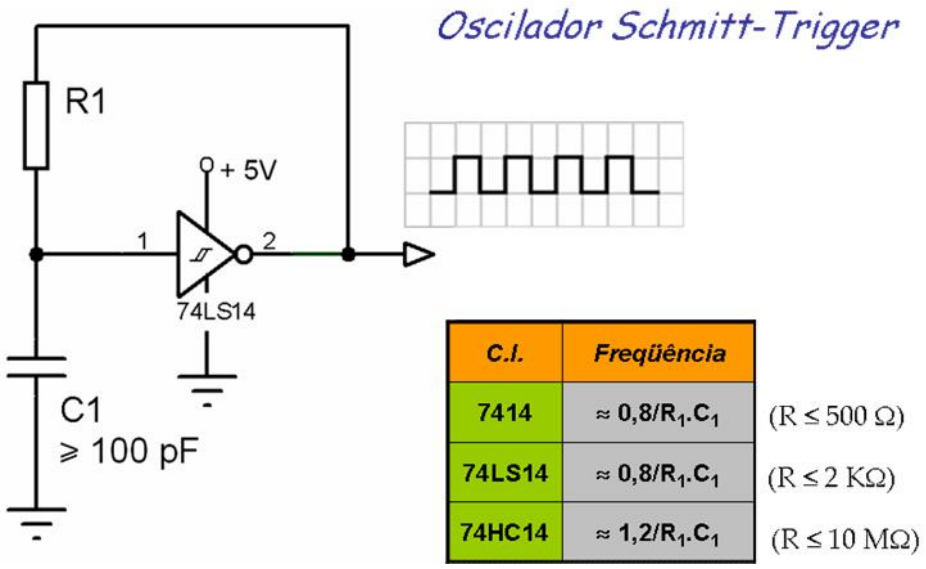
Circuitos Astáveis

São circuitos que não apresentam nenhum ponto de equilíbrio e que, em geral, mudam constantemente de estado em busca de um ponto de equilíbrio que não existe. Tais circuitos exibem características oscilatórias que são a base dos circuitos osciladores.

Circuitos osciladores que podem gerar um trem de pulsos retangulares e/ou quadrados são normalmente empregados em circuitos eletrônicos digitais como geradores de clock.

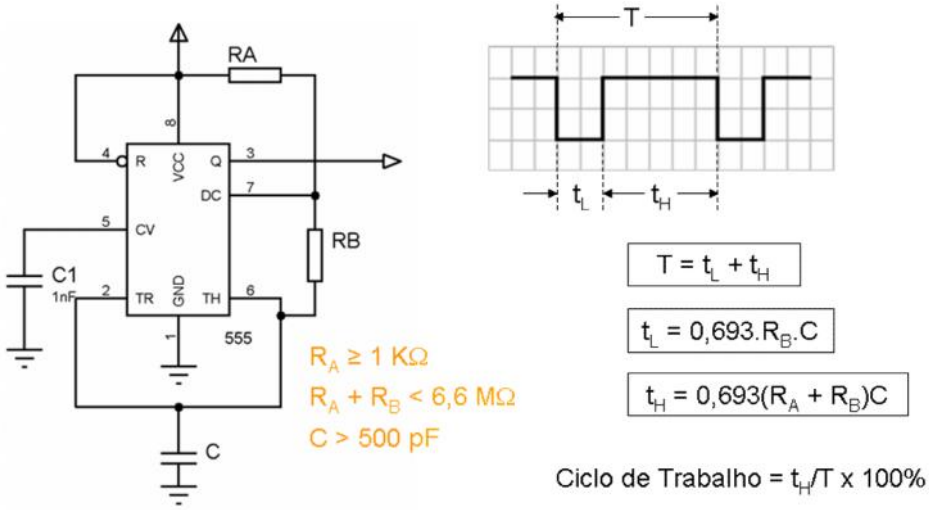
Há inúmeros circuitos que podem ser utilizados para gerar um sinal de clock. Dentre eles, podemos destacar:

Oscilador com Schmitt-Trigger



C.I.	Freqüência	
7414	$\approx 0{,}8/R_1.C_1$	($R \leq 500\ \Omega$)
74LS14	$\approx 0{,}8/R_1.C_1$	($R \leq 2\ K\Omega$)
74HC14	$\approx 1{,}2/R_1.C_1$	($R \leq 10\ M\Omega$)

Oscilador com 555

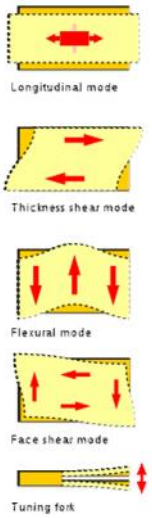


Temporizador 555 como multivibrador astável

Oscilador a Cristal

- As freqüências de saída dos osciladores visto anteriormente dependem dos valores dos resistores e capacitores, que variam com o tempo, temperatura e até umidade → Instabilidade na freqüência de clock.

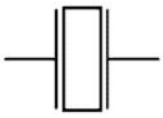
Solução → Cristal de quartzo (efeito *piezoelétrico*) !
Normalmente encontrados com valores entre 10 KHz a 80 MHz.
Ajudam a gerar intervalos precisos de tempo.
Empregado praticamente junto a todos os microprocessadores e microcontroladores modernos.



Modos de vibração

Crescimento Natural

Crescimento Artificial



Símbolo eletrônico