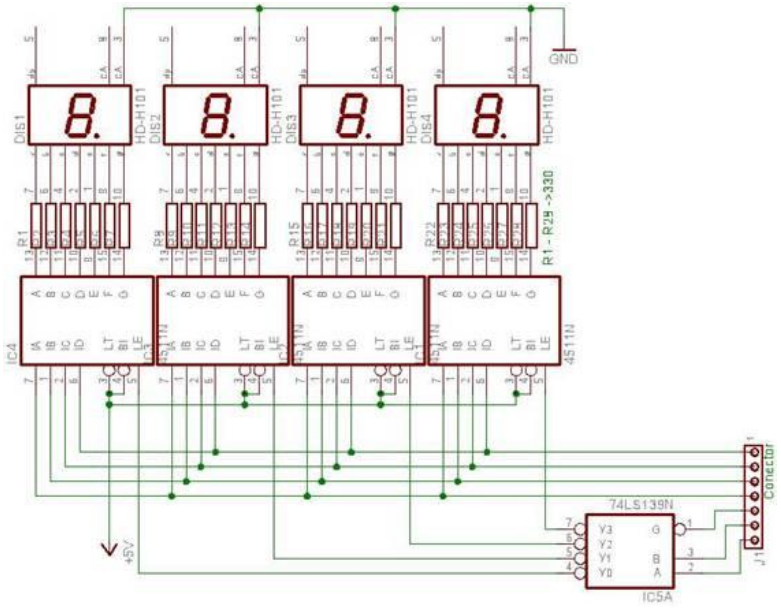
Eletrônica Digital



PET - Engenharia Elétrica UFC Março - 2014

Responsáveis

A apostila de **Eletrônica Digital** é de responsabilidade do **Programa de Educação Tutorial** do curso de **Engenharia Elétrica** da **Universidade Federal do Ceará**, tendo como principais responsáveis os bolsistas:

- Ícaro Silvestre Freitas Gomes
- José Antonio de Barros Filho
- Lucas Cordeiro Herculano
- Lucas Rebouças Maia
- Ricardo Antônio de Oliveira Sousa Júnior

SUMÁRIO

Introdução à Eletrônica Digital

- ### O que é representação digital?

Em muitas áreas do trabalho é necessário se trabalhar com quantidades. A medida dessas quantidades pode ser guardada, graduada, manipulada aritmeticamente de alguma maneira. Basicamente se tem duas maneiras de manipular ou representar essas quantidades a maneira **analógica** e a maneira **digital.**

o Analógica

Os valores analógicos estão relacionados aos valores contínuos. Esses valores podem variar para qualquer valor, numa faixa de valores. Ela nunca possui valor absoluto é uma posição aproximada em uma escala contínua.

Ex: Temperatura ambiente, corrente em um indutor, relógio analógico etc.

o Digital

Os valores digitais estão relacionados à valores discretos. Esses valores podem variar, de maneira fixa, numa faixa de valores. São sempre valores absolutos, quem variam com "saltos".

Ex: Relógio digital, a posição de uma chave de 10 posições, grãos de areia na praia etc.

Em contra partida dos dois conceitos apresentados acima, podemos então ter o caso rampa/escada, onde a rampa representaria o analógico e a escada representaria o digital.

- ### Vantagens e Desvantagens na utilização digital

o Vantagens

- Sistemas digitais são mais simples de serem projetados.
- Maior facilidade em manter precisão e exatidão.
- Operações podem ser programadas.
- Menos afetados por ruídos.
- CI's (Circuitos Integrados) podem ser fabricados com mais dispositivos.

o Desvantagens

- O mundo é quase todo analógico;
- Processar sinais digitais demanda tempo.

Logo, se for necessário trabalhar com, qualquer que seja o digital existe uma série de passos a seguir.

1. Converter a variável que se deseja em sinal elétrico (analógico);
2. Converter a variável analógica em variável digital;
3. Realizar o processamento da informação;
4. Converter o sinal Digital de volta para um sinal analógico.

▪ Sistemas de numeração

Existem infinitos sistemas de numeração diferentes com os quais se pode trabalhar. No dia a dia se está habituado a trabalhar com o sistema decimal (base 10), mas nos estudos de eletrônica digital será comum se encontrar outros sistemas como: Binário, Octal, Hexadecimal e BCD.

O polinômio geral para sistema de numeração em decimal é:

$$(n)_b = n_i.b^i + n_{i-1}.b^{i-1} + ... + n_1.b + n_0.b^0$$

○ Binário

O sistema binário utiliza dois dígitos (base 2) para representar qualquer quantidade. Assim utilizando o algoritmo acima o número binário 1101 pode ser representado da seguinte forma:

$1101 = 1 . 2^3 + 1 . 2^2 + 0 . 2^1 + 1 . 2^0$

$1101 = 8 + 4 + 0 + 1 = 13$

Note que os índices foram especificados em notação decimal, o que possibilita a conversão binária-decimal como descrito acima.

Através do exemplo anterior, podemos notar que a quantidade de dígitos necessária para representar um número qualquer, no sistema binário, é muito maior quando comparada ao sistema decimal.

A grande vantagem do sistema binário reside no fato de que, possuindo apenas dois dígitos, estes são facilmente representados por uma chave aberta e uma chave fechada ou, um relé ativado e um relé desativado, ou, um transistor saturado e um transistor cortado; o que torna simples a implementação de sistemas digitais mecânicos, eletromecânicos ou eletrônicos.

Obs1: Em sistemas eletrônicos, o dígito binário (0 ou 1) é chamado de BIT, enquanto que um conjunto de 8 bits é denominado *BYTE*.

Obs2: Um conjunto de 4 bits é denominado *Nibble*.

○ Octal

O sistema octal utiliza oito dígitos (base 8) para representar qualquer quantidade. Assim utilizando o algoritmo já citado o número octal 17 pode ser representado da seguinte forma:

$17(O) = 1. 8^1 + 7. 8^0 = 8 + 7 = 15 (D)$

Note que os índices foram especificados em notação decimal, o que possibilita a conversão octal-decimal como descrito acima.

Através do exemplo anterior, podemos notar que a quantidade de dígitos necessária para representar um número qualquer, no sistema octal, é maior quando comparada ao sistema decimal, e menor comparada ao

sistema binário. Sua vantagem é na sua fácil conversão com o sistema binário e com o sistema hexadecimal, que será citado adiante.

○ Hexadecimal

Enquanto que o sistema binário tem dígitos a menos que o sistema decimal, o sistema hexadecimal (16 dígitos) tem dígitos a mais. Sendo esse sistema composto por 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E e F com A = 10 e F = 15. Utilizando o algoritmo, o número 12B pode ser escrito, em decimal, como:

$1.16^2 + 2.16^1 + B.16^0 = 256+32+11 = 299.$

Novamente, os índices foram especificados em notação decimal, o que possibilita a conversão binária-decimal como descrito acima.

Com base no exemplo, nota-se a quantidade de dígitos necessária para representar um número qualquer, no sistema hexadecimal, é menor quando comparada ao sistema decimal.

As vantagens são óbvias. É mais fácil representar números decimais ou binários de grande ordem no sistema hexadecimal. Por isso é extremamente utilizado em aplicações computacionais e item indispensável para quem estuda linguagens de programação.

○ BCD (codificação binária decimal)

O código BCD é um código informal onde cada conjunto de quatro dígitos na base 2 representa um número da base 10 e é utilizado em eletrônica digital da forma que um nibble representa um número em decimal.

Ex: 1001 1001 este número em binário valeria $1*2^7 + 1*2^4 + 1*2^3 + 1*2^0 = 153$, enquanto que em BCD valeria 99.

▪ Conversão

Em alguns casos se torna trabalhoso, e até desconfortável trabalhar com valores em bases das quais não é de costume o trabalho, porém algumas vezes o inverso pode facilitar o trabalho. Deste modo em seguida será apresentada as seguintes conversões: Decimal-Binário, Binário-Decimal, Decimal-Octal, Octal - Decimal, Decimal-Hexadecimal, Hexadecimal-Decimal, Decimal-BCD, Hexadecimal-Binário e Binário-Hexadecimal

○ Decimal-Binário

Para se converter um número decimal em binário, divide-se sucessivamente o número decimal por 2 (base do sistema binário), até que o último quociente seja 1. Os restos obtidos das divisões e o último quociente compõem um número binário equivalente, lido de trás para frente, como mostra o exemplo a seguir.

Ex.:

a) 23 (D)

23/2 = 11 (1) → 11/2 = 5 (1) → 5/2 = 2 (1) → 2/2 = 1 (0) → 1/ 2 = 0 (1)

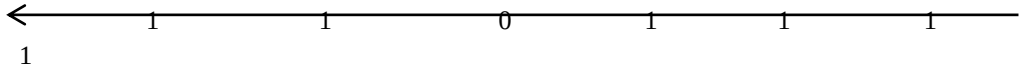
1 1 1 0 1

←

23 = 10111

b) 123(D)

123/2 = 61(1) → 61/2 = 30(1) → 30/2 = 15(0) → 15/2=7(1) → 7/2=3(1) → 3/2 = 1(1) → 1/ 2 = 0 (1)



123 = 1111011

- **Binário-Decimal**

Como já mostra, a mesma se dá pela soma das multiplicações de cada termo por sua potência de base 2 equivalente. Ex:

a) 1011(B)
$= 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 8+0+2+1 = 11$

- **Decimal-Octal**

Na conversão de decimal para octal, é feita a divisão sucessiva do número decimal por 8 (base do sistema octal), até que o último quociente seja menor que 8. Os restos obtidos das divisões e o último quociente compõem um número octal equivalente. A Será lido também de trás para frente. Ex:

a) 100 / 8 = 12 e resto 4 , pois 100 - 12 * 8 = 100 - 96 = 4
12 / 8 = 1 e resto 4
1/8 = 0 resto 1
logo, 100 (D) = 144(O)

- **Octal- Decimal**

Como já mostrado, tal conversão é feita pela soma das multiplicações de cada termo por sua potência de base 8 equivalente. Ex:

a) 245 (O)
$= 2*8^2 + 4*8^1 + 5*8^0 = 128+32+5 = 165$ (D)

- **Decimal-Hexadecimal**

Na conversão de decimal para hexadecimal, é feita a divisão sucessiva do número decimal por 16 (base do sistema hexadecimal), até que o último quociente seja menor que 16. Os restos obtidos das divisões e o último quociente compõem um número binário equivalente. A Será lido também de trás para frente. Ex:

b) 542 / 16 = 33 e resto 14 , pois 542 - 33 * 16 = 542 - 528 = 14
33 / 16 = 2 e resto 1
logo, 542 (D) = $2 * 16^2 + 1 * 16 + 14$ = 21E (H)

- **Hexadecimal-Decimal**

Como já mostrado, tal conversão é feita pela soma das multiplicações de cada termo por sua potência de base 16 equivalente. Ex:

b) 3B0A (H)
$= 3*16^3 + 11*16^2 + 0*16^1 + 10*16^0 = 12288+2816+0+10 = 15114$ (D)

○ Decimal-BCD

Para converter um número de Decimal para BCD, basta que se torne cada algarismo em um nibble corresponde ao mesmo. Ex:

a) 45(D) → 4 = 0100 (b) e 5 = 0101(b), logo 45(D) = 0100 0101 (BCD)
c) 81(D) → 8 = 1000(b) e 1 = 0001(b), logo 81(D) = 1000 0001 (BCD)

Obs 3.: Pode-se fazer uma analogia entre Decimal - BCD e Hexadecimal - Binário, ambas as conversões são feitas transformando algarismo em nibbles.

○ Hexadecimal-Binário, Binário-Hexadecimal

Na conversão hexa-binário usa-se de um simples algoritmo. Cada número em hexa equivale ao conjunto de um *nibble* em binário. Ex:

a) 2B = 0010 1011
2B= 2*16 + 11 = 43 (1)
0010 1011 = $1*2^5+2^3+2+1 = 32+8+2+1 = 43$ (2)
Como (1) = (2) → conversão correta.

b) 52A = 0101 0010 1010
52A = $5*16^2 + 2*16^1 + 10.16^0 = 1322$ (1)
= $0*2^{11} + 1*2^{10} + 0*2^9 + 1*2^8 + 0*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0$
+ 1024 + 0 + 256 + 0 + 0 + 32 + 0 + 8 + 0 + 2 + 0 = 1322 (2)
Como (1) = (2) → conversão correta.

Nota-se que a conversão binária-hexa é feita no sentido inverso, ou seja para cada nibble se terá um casa de algarismo em hexa. Ex:

a) 1001 1111 = 9F → 1001 = 9 e 1111 = F
b) 1110 1101 = DC → 1110 = D e 1101= C

A Tabela 1 contém um resumo com as devidas transformações de Binário-Octal-Hexadecimal-BCD-Decimal.

Binário	Octal	Hexadecimal	BCD	Decimal
0000	0	0	0000	0
0001	1	1	0001	1
0010	2	2	0010	2
0011	3	3	0011	3
0100	4	4	0100	4
0101	5	5	0101	5
0110	6	6	0110	6
0111	7	7	0111	7
1000	10	8	1000	8
1001	11	9	1001	9
1010	12	A	0001 0000	10
1011	13	B	0001 0001	11
1100	14	C	0001 0010	12
1101	15	D	0001 0011	13
1110	16	E	0001 0100	14

1111	17	F	0001 0101	15

Tabela 1 - Conversão entre Sistemas de Numeração

Circuitos Integrados e portas lógicas

Em lógica, existem somente dois resultados possíveis: falso ou verdadeiro. Por ter somente 2 resultados possíveis o sistema binário é ideal para ser trabalhado com problemas que envolvam lógica. Em 1854, um matemático britâncio chamado George Boole criou um método de expressar pensamentos de forma lógica e matemática que ficou conhecido como "Álgebra Booleana". Tal matemática é baseada em variáveis binárias, que podem assumir valor "0" ou "1", e operações próprias. Pode-se modelar um circuito elétrico lógico por meio de álgebra booleana e de componentes eletrônicos que mantem como relação de entrada/saída dos seus terminais uma operação booleana. Tais dispositivos são chamadas de portas lógicas.

Outra ferramenta de suma importância no estudo de Eletrônica Digital são as Tabelas-Verdade. São tabelas onde podemos escrever as saídas de um circuito em função de todas as possíveis entradas, como mostra a Fig. 1.

Entradas **Saída**

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

Figura 1 - Exemplo de tabela verdade.

Cada coluna se refere a uma entrada ou a uma saída. No tabela da Fig. 1 temos três entradas: A, B e C; e uma saída: S. Cada linha relaciona uma saída com uma combinação diferente das três entradas. Por

exemplo: na primeira linha, temos S=0. Logo, quando temos as entradas A=0, B=0 e C=0 a saída S é zero também. Podemos ter tabelas verdades com indefinidas entradas e indefinidas saídas. Mas, ao aumentar o número de variáveis (entradas ou saídas), o projetista torna o circuito mais complexo.

Existem três operações lógicas básicas: Or, And e Not. No entanto, outras operações também são bastante utilizadas: Nor, Nand, Xor e Xnor.

- **Operação OR**

A operação OR ("Ou") tem seguinte forma: O evento A *ou* o evento B (ou o evento C... ou indefinidos eventos) são verdadeiros?

Caso alguns deles seja verdadeiros (ou os dois) a resposta será verdadeira. A saída será 0 apenas se todas as entradas forem iguais a 0, ou seja, se pelo menos 1 entrada for igual a 1 a saída será 1.

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

Figura 2 - Tabela verdade da porta OR.

A expressão é do tipo: A + B = S, onde "+" siginifica a operação *Or*. Lê-se "A ou B é S". A representação simbólica e a representação em circuito estão mostradas nas Fig. 3 e 4, respectivamente:

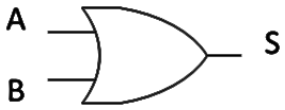


Figura 3 - Símbolo porta OR.

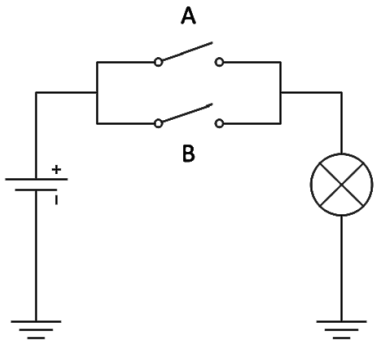


Figura 4 - Circuito representativo porta OR.

- **Operação AND**

A operação And ("E") tem seguinte forma: O evento A *e* o evento B (e o evento C... ou indefinidos eventos) são verdadeiros? Caso os dois sejam verdadeiros a resposta será verdadeira. Ou seja, a saída só será 1 se todas as entradas forem 1, se pelo menos uma entrada for 0 a saída será 0. Assim podemos montar a tabela verdade da Fig. 5.

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

Figura 5 - Tabela verdade da porta AND.

A expressão é do tipo: A . B = S, onde "." siginifica a operação And. Lê-se "A e B é S". A representação simbólica e a representação em circuito estão mostradas nas Fig.6 e 7, respectivamente:

Figura 6 - Símbolo porta AND.

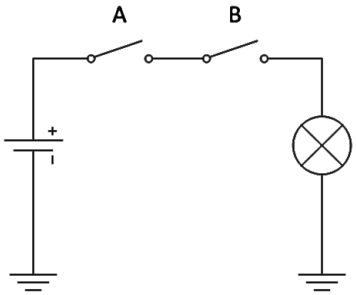


Figura 7 - Circuito representativo porta AND.

- **Operação NOT**

A operação Not ("Não") tem seguinte forma: O evento A é verdadeiro ou falso? Caso seja verdadeiro a resposta será falsa, caso falso, verdadeira. Seja a entrada A igual a 1 a saída S é 0 e seja a entrada A igual a 0 a saída S é 1. A operação Not é usada com uma única entrada. Assim podemos montar a tabela verdade da Fig. 8.

A	S
0	1
1	0

Figura 8 - Tabela verdade da porta NOT.

A expressão é do tipo: $\overline{A} = S$, onde "-" siginifica a operação Not. Lê-se "A barra é igual a S." A representação simbólica e a representação em circuito estão mostradas nas Fig. 9 e 10, respectivamente:

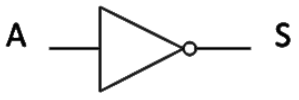


Figura 9 - Símbolo porta NOT.

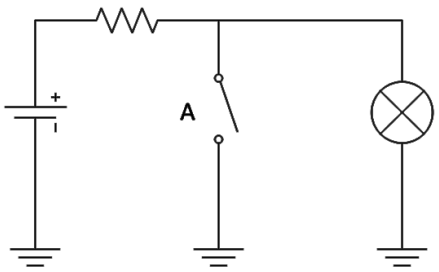


Figura 10 - Circuito representativo porta NOT.

Obs: Também pode-se representar por A' = S.

Exemplo: Seja o sinal de saída, S, igual a A+B+C, a forma de onda da saída de acordo com as formas de onda de A, B e C é:

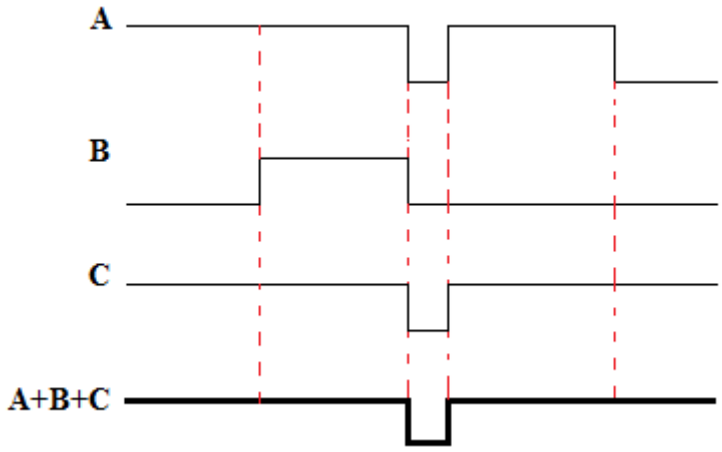


Figura 11 - Sinal de saída.

- **Operação NOR.**

A operação NOR ("não OR") tem seguinte forma: O evento A ou evento B são verdadeiros? Caso alguns deles seja verdadeiros (ou os dois) a resposta será falsa. Ou seja, se pelo menos uma entrada for igual 1 a saída será igual a 0. Assim podemos montar a seguinte tabela verdade.

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

Figura 12 - Tabela verdade da porta NOR.

A expressão é do tipo: $\overline{A + B} = S$, onde "-" siginifica a operação Not. Lê-se "A mais B barrados é igual a S." A representação simbólica está mostradas na Fig.13

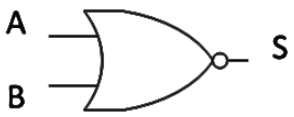


Figura 13 - Símbolo porta NOR

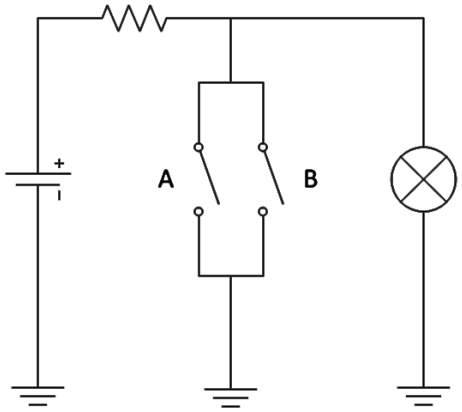


Figura 14 - Circuito representativo porta NOR.

- **Operação NAND.**

A operação NAND (“não E”) tem seguinte forma: O evento A e evento B são verdadeiros? Caso os dois sejam verdadeiros a resposta será falsa. Ou seja, a saída só será 0 se todas as entradas forem 1, se pelo menos uma entrada for 0 a saída será 1. Assim podemos montar a seguinte tabela verdade.

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

Figura 15 - Tabela verdade da porta NAND.

A expressão é do tipo: $\overline{A.B} = S$, onde “-” siginifica a operação Not. Lê-se “A multiplicado por B barrados é igual a S.” A representação simbólica está mostradas na Fig.13

Figura 16 - Símbolo porta NAND

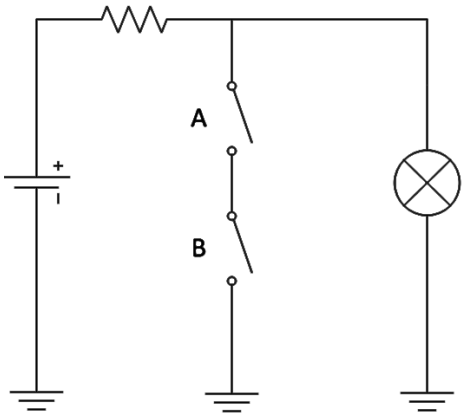


Figura 17 - Circuito representativo porta NAND.

- **Operação XOR.**

A operação XOR (“Ou exclusivo”) tem seguinte forma: O evento A *ou* evento B são diferentes? Caso ambos sejam diferentes é verdadeiro. Ou seja, caso a entrada A e a entrada B forem diferentes a saída S será igual a 1. Assim podemos montar a seguinte tabela verdade.

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

Figura 18 - Tabela verdade da porta XOR.

A expressão é do tipo: $\overline{A}.B + \overline{B}.A = S$, onde “-” siginifica a operação Not. Lê-se “A barrado multiplicado por B ou B barrado multiplicado por A é igual a S.” A representação simbólica está mostradas na Fig. 19 e 20.

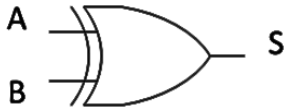


Figura 19 - Símbolo porta XOR.

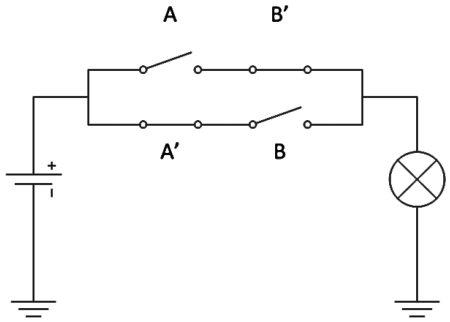


Figura 20 - Circuito representativo porta XOR.

- **Operação XNOR.**

A operação XNOR (“ Não Ou exclusivo”) tem seguinte forma: O evento A *ou* evento B são iguais? Caso ambos sejam diferentes é falso. Seja a entrada A e a entrada B diferentes a saída S será igual a 0. Assim podemos montar a seguinte tabela verdade.

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

Figura 21 - Tabela verdade da porta XOR.

A expressão é do tipo: $\overline{\overline{A}.B + \overline{B}.A} = S$, onde “-” siginifica a operação Not. Lê-se “A barrado multiplicado por B ou B barrado multiplicado por A barrados é igual a S.” A representação simbólica está mostrada nas Fig. 22 e 23.

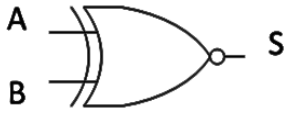


Figura 22 - Símbolo porta XOR.

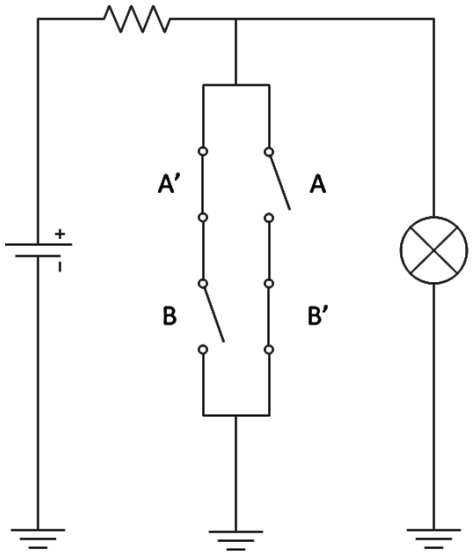


Figura 23 - Circuito representativo porta XOR.

Álgebra Booleana

A Álgebra Booleana está relacionada a George Boole, matemático britânico do século XIX que desenvolveu seu estudo a partir de duas circunstâncias: ou uma afirmativa é verdadeira (1) ou é falsa (0).

- **Operações na álgebra booleana.**

Basicamente, neste estudo existem três tipos de operações. São elas Or (ou), And (e), Not (não/inversor). As variáveis booleanas não implicam necessariamente valores em suas operações, mas sim, estados lógicos.

"A ou B" ↔ "B ou A" ↔ "A + B" ↔ "B + A"

"A e B" ↔ "B e A" ↔ "A . B" ↔ "B . A"

"NÃO A" ↔ " A' " ↔ " $\overline{A}$ "

Assim,

0 + 0 = 0	0 . 0 = 0	Não 1 = 1' = $\overline{1}$ = 0
1 + 1 = 1	1 . 1 = 1	Não 0 = 0' = $\overline{0}$ = 1
0 + 1 = 1 + 0 = 1	0 . 1 = 1 . 0 = 0	

Entende-se por "1" nível lógico Alto, e por "0" nível lógico Baixo.

- **Álgebra booleana na eletrônica digital**

As operações em eletrônica digital com portas lógicas serão vista na secção 5, porém alguns conceitos devem ser notados primeiramente, como as operações básicas com binários.

- **Carry**

O conceito de *carry* é parecido com o conceito de "Vai um" utilizado na operação de soma na matemática básica. Neste caso se terá uma nova tabela de somas.

0+0 = 0
0+1 = 1
1+0 = 1
1+1 = 0 (carry +1)

Exemplo: 1001 = 9
+ 0011 = 3
1100 = 12

- **Borrow.**

Semelhante ao conceito de *carry*, apresentado anteriormente, existe o conceito de *borrow* que na matemática básica era tido como o "Empresta um" utilizado na operação de subtração.

0 - 0 = 0
1 - 0 = 1
1 - 1 = 0
0 - 1 = 1 (borrow +1)

Exemplo: 1100 = 12
- 1001 = 9
0011 = 3

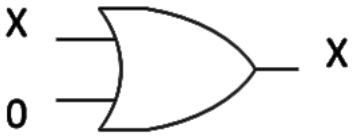
- **Teoremas na álgebra booleana.**

A álgebra booleana está repleta de condições que levam à consequências lógicas, logo será aqui dissertado sobre alguns teoremas conhecidos da álgebra booleana.

A Fig. 24 mostra os Teoremas 1-8 que são teoremas de uma única variável.

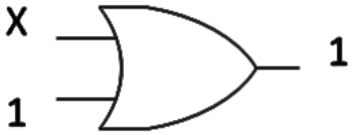
(1) X.0 = 0

(5) X+0 = X



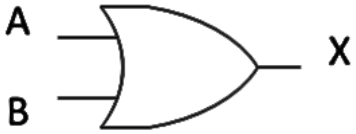
(2) X.1 = 1

(6) X+1 = 1

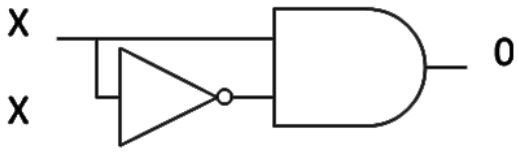


(3) X.X = X

(7) X+X = X



(4) X.X' = 0



(8) X+X' = 1

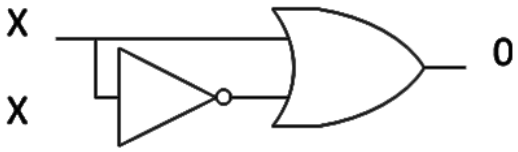


Figura 24 - Teoremas de uma única variável.

As propriedades associativas, comutativas e distributivas se aplicam aos teoremas de Boole como mostram os Teoremas 9-13:

(9) X+Y=Y+X

(10) X.Y = Y.X

(11) X+(Y+Z) = (X+Y) + Z

(12) X.(Y.Z) = (X.Y).Z

(13) X.(Y+Z) = X.Y+X.Z

A partir das leis associativas, comutativas e distributivas definidas podemos definir novas propriedades a partir delas, como seguem os Teoremas 14 e 15.

(14) X+Y.X = X

PROVA:

X + Y . X = X . (1 + Y) = X . 1 = X

(15) X+ X'.Y = X+Y

PROVA:

X . (1 + Y) + X' . Y = X + X . Y + X'. Y =

Y . (X + X') + X = Y . 1 + X = Y + X

Além das propriedades, existem duas das leis mais importantes de Boole que são os teoremas de DeMorgan. Estes teoremas foram um salto para eletrônica digital, provando que as portas "não e" e "não ou" podem representar qualquer elemento do circuito.

(17) $\overline{X + Y} = \overline{X}.\ \overline{Y}$

(18) $\overline{X.Y} = \overline{X} + \overline{Y}$

Introdução ao Proteus

O *software* Proteus 8 é uma eficiente ferramenta computacional bastante utilizada em eletrônica digital, pois possui vastas bibliotecas com portas lógicas e ferramentas extras para simulação. Tem interface dinâmica, muitas funções e é facil de utilizar. Para acessá-lo, utlize o seguinte caminho:

Menu Iniciar → Todos os Programas → Proteus 8 Professional

A tela inicial do Proteus é a seguinte:

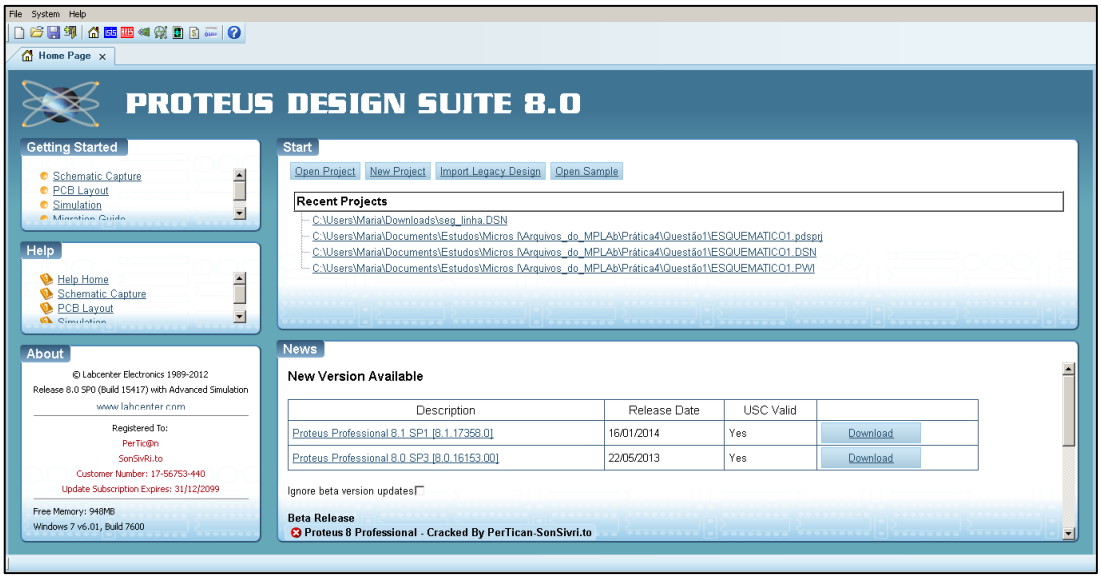


Figura 25 - Tela inicial do Proteus

Para criar um novo projeto, clique no menu *File*, na parte superior esquerda da tela, e, em seguida, *New Project*.

File → *New Project*

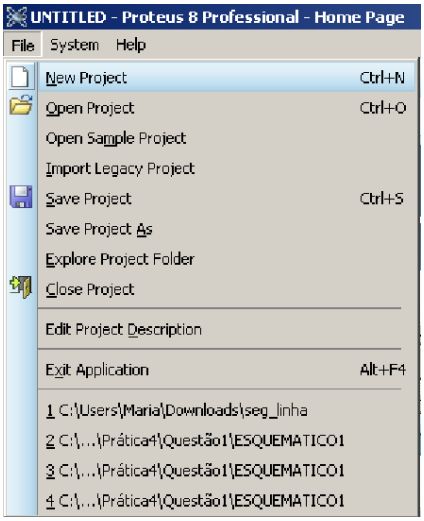


Figura 26 - Menu *File*

Em seguida será aberta uma janela semelhante à da Fig. 27.

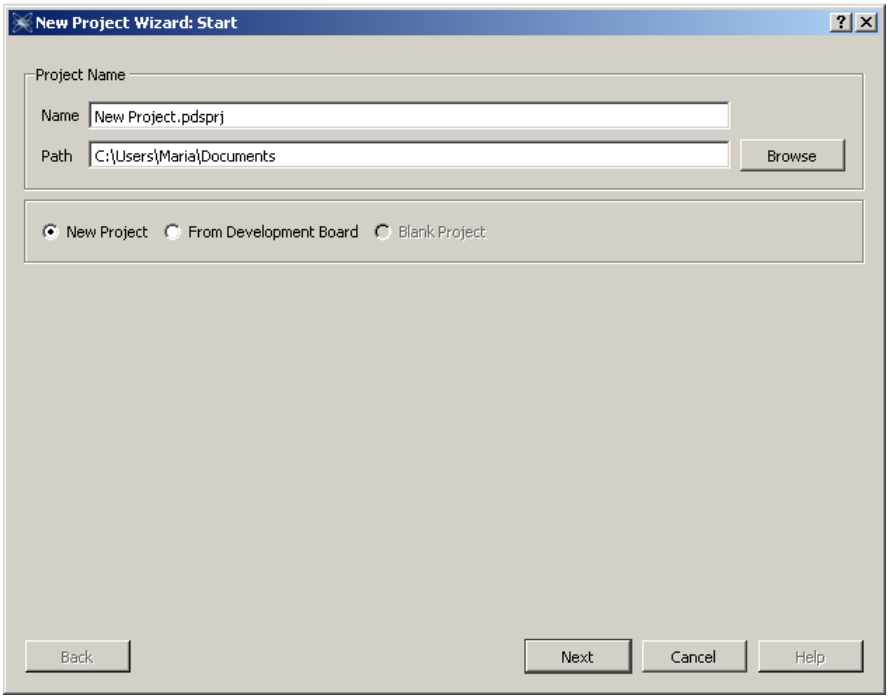


Figura 27 - *New Project Wizard*: *Start*

No campo *Name*, será colocado o nome do projeto, e em *Path*, o local de instalação. Em seguida clique em *Next*. Aparecerão na tela outras janelas para inserir arquivos no projeto. Como não é o caso clique em *Next* nas próximas janelas. Por fim, aparece uma tela com as caracterísitcas do projeto e clique em *Finish*.

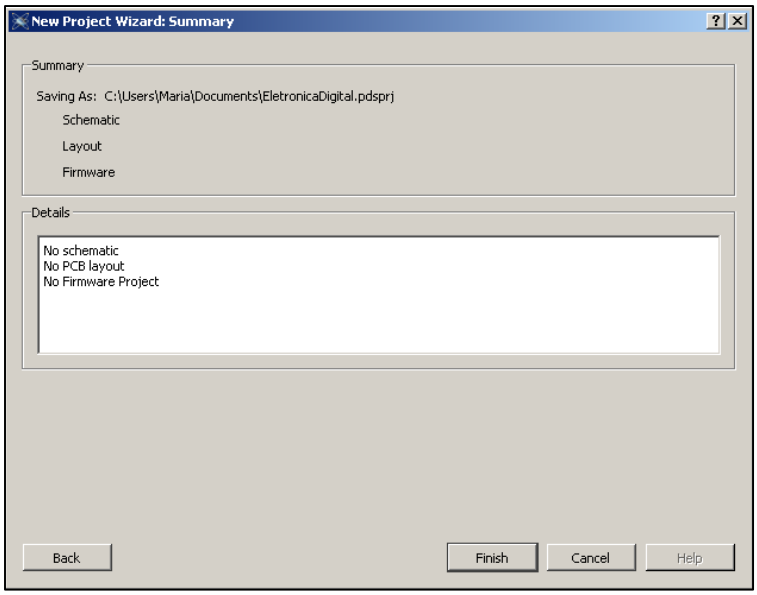


Figura 28 - *New Project Wizard*: *Summary*

Aparecerá uma janela semelhante a da Fig. 29.

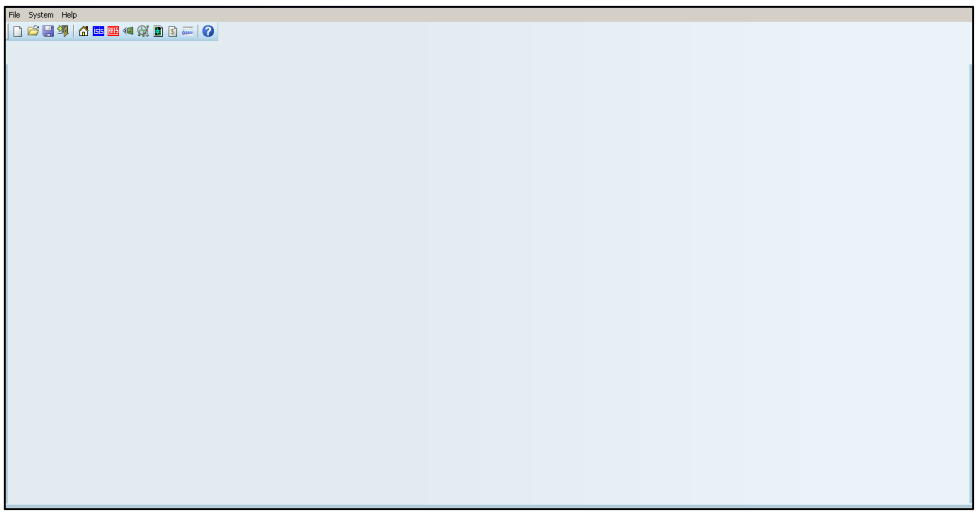


Figura 29 - Tela inicial do Proteus

Existem duas ferramentas extremamente úteis no *Proteus* 8 *Professional*. A primeira é o *Ísis* que é usado para desenhar circuitos elétricos em geral inclusive esquemáticos de circuitos digitais. A segunda, *Ares*, é utilizada pra rotear placas de circuito impresso (PCB). Clique em *Isis* para aparecer uma área de trabalho onde poderemos editar nossos circuitos.

Figura 30 - Atalhos para acessar as ferramentas ISIS ou ARES do Proteus

Será aberta uma figura semelhante a da Fig. 31. Haverá uma *toolbox* onde pode-se utilizar todos os componentes e dispositivos utilizados em eletrônica digital.

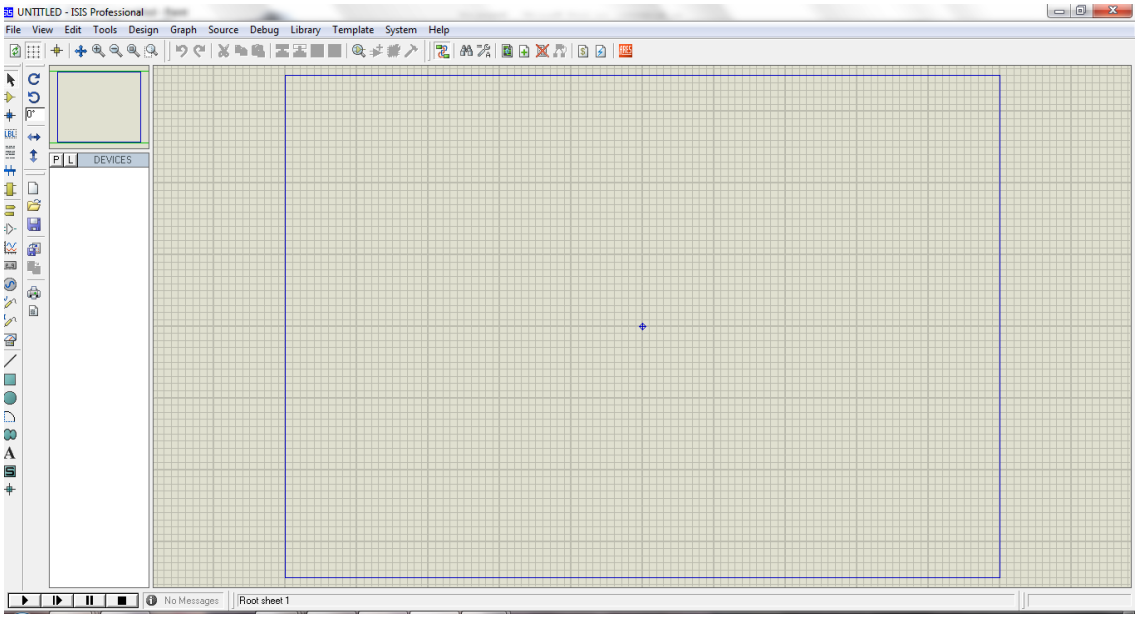


Figura 31 - Area de trabalho do Proteus

Antes de começar a simular os circuitos digitais, é preciso conhecer um pouco sobre os recursos que o programa oferece (ao lado esquerdo da área de trabalho).

Na Fig. 32, escolhemos na opção *Component Mode*, seta de número 1 na Fig. 32, todos os componentes que serão utilizados no projeto após clicar em "P" ao lado de *devices*. Dessa forma é possível escolher quais portas lógicas que serão utilizadas, como por exemplo: OR, XOR, AND, NOT, NOR, NAND, etc. Vale ressaltar que nessa opção é possível adicionar outros componentes como resistores, LED's, indutores, capacitores, microprocessadores, além de outros dispositivos analógicos e digitais.

Após apertar na opção *Component Mode*, aperta-se na opção *Pick Devices*, ícone P indicado pela seta de número 2 na Fig. 32, e abrirá a tela dos dispositivos. Após esse passo, basta digitar no espaço *Keyword*, seta de número 3 na Fig. 32, o nome do dispositivo ou da porta lógica que se deseja utilizar, como por exemplo: AND_2. O dispositivo selecionado aparecerá na lista dos dispositivos e poderá ser utilizado a qualquer momento, bastando clicar duas vezes sobre o nome dele que aparecer.

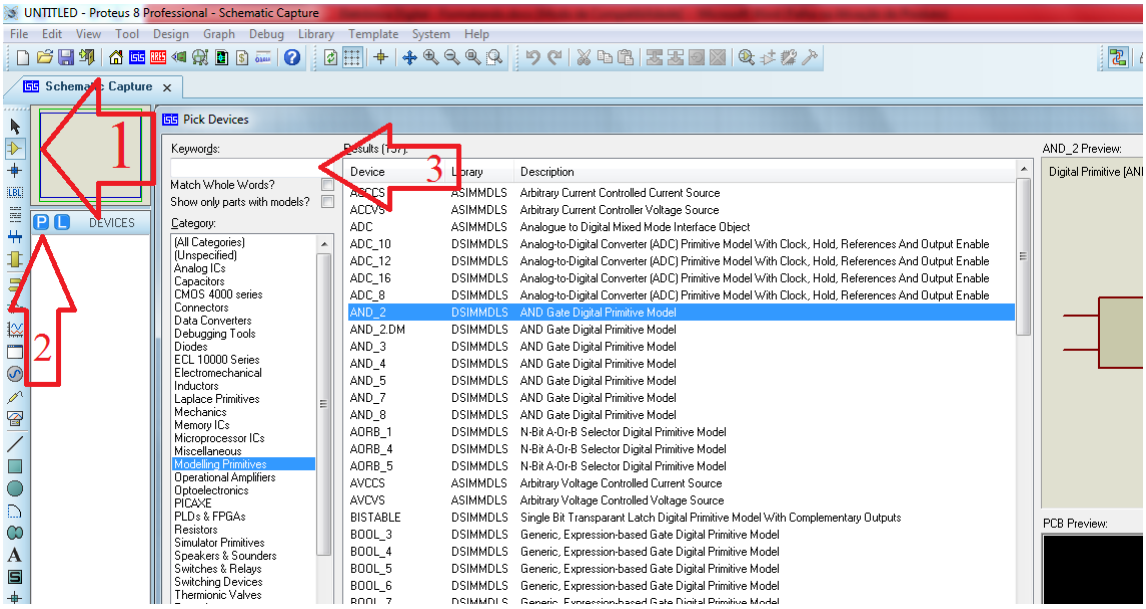


Figura 32 - Tela Pick Devices

Em eletrônica digital é preciso definir também as entradas e saídas do circuito lógico, neste caso o Proteus possui ferramentas para colocar entradas que variam de acordo com a frequência configurada ou entradas que permanecem fixas em 0 ou 1. Para definir entradas fixas através do espaço *Keyword*, seta de número 3 na Fig. 32, é necessário digitar *logictoogle* ou *logicstate* e colocar no circuito. Já a saída é preciso digitar *logicprobe (big)*. Na Fig. 33 tem-se o esquemático do componente.

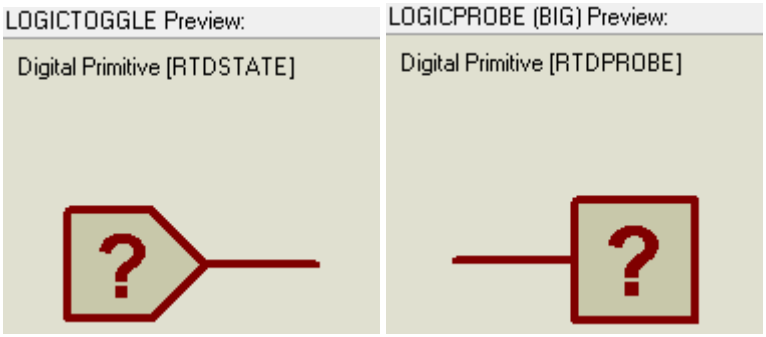


Figura 33 - *Logictoggle* e *logicprobe* (*big*)

Para colocar uma entrada variável de acordo com a frequência, basta apertar em *Generation Mode*, indicada pela seta na Fig. 34, e selecionar a opção *DCLOCK*.

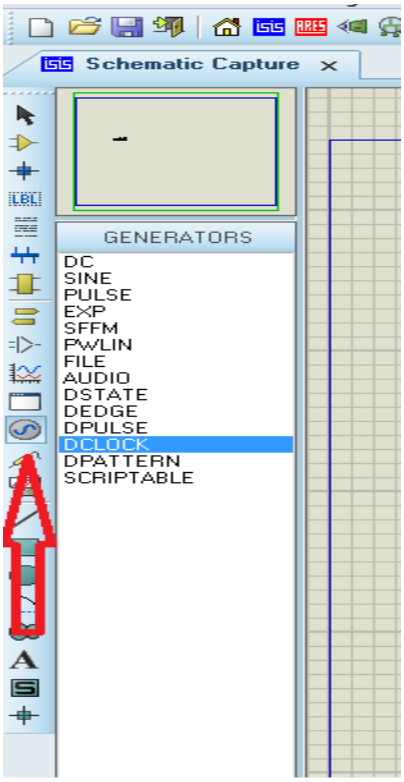


Figura 34 - Colocando o DCLOCK

Dessa forma, ao ligar o dispositivo como entrada no circuito, pode-se alterar a frequência apertando duas vezes sobre ele e modificando o valor no espaço *Frequency*, Fig. 35.

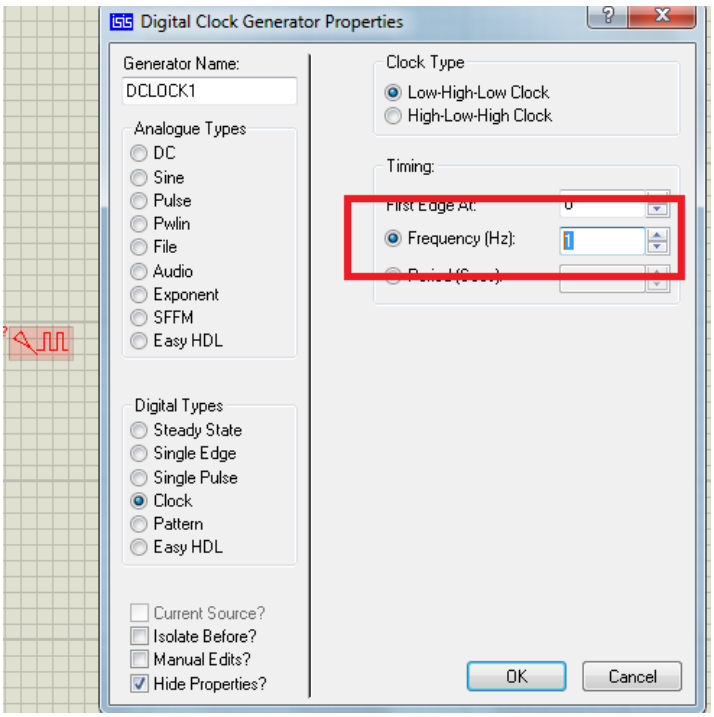


Figura 35 - Alterando a frequência do DCLOCK

Para dar início à simulação é necessário realizar as ligações entre os componentes, com a ferramenta *2D Graphic Line Mode >> Component*, Fig. 36.

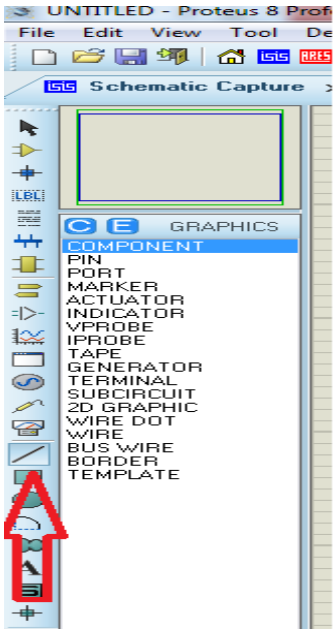


Figura 36 - Interligando os componentes

Agora, com todos os componentes interligados, no caso de eletrônica digital as portas lógicas, é possível realizar a simulação apertando no símbolo *play*, Fig. 37.

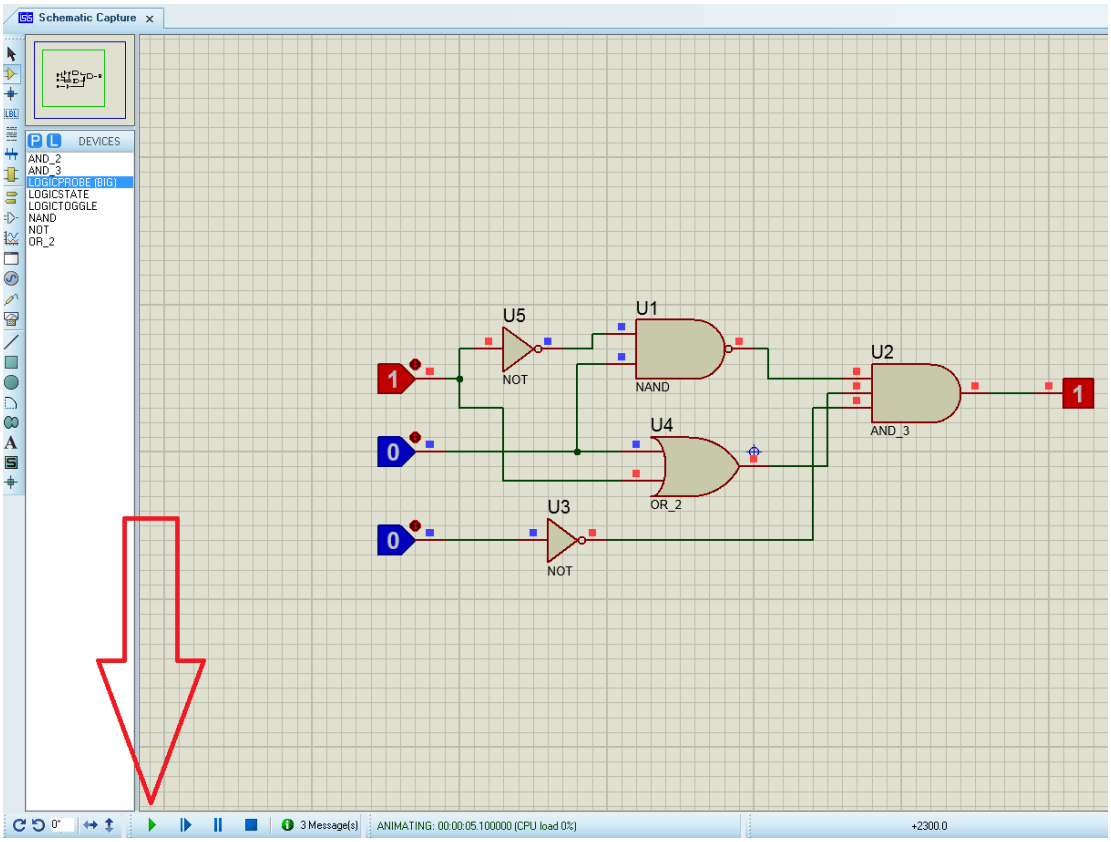


Figura 37 - Simulando no Proteus.

Operações com portas lógicas

Em lógica, existem apenas dois estados para uma entrada ou saída: verdadeiro ou falso. Por meio desse conceito, é possível criar circuitos que resultam em operações lógicas, inteligentes e coerentes. Escrever essas operações pode se tornar cansativo, por isso, existe a preferência por descrever as variáveis por letras (A, B, x, y,...) e aplicar técnicas de simplificação ao circuito.

É importante o estudo da análise da sequência de operações com portas lógicas para se ter uma real ideia da operação do circuito como todo. Esse estudo não é simples e requer uma série de exercícios para se fixar bem o estudo.

- **Descrevendo circuitos digitais por meio de variáveis;**

Qualquer circuito, independente da complexidade, como já foi explicado anteriormente, pode ser descritos a partir das três operações booleanas básicas: OR, AND, NOT. Nesta parte desse material é feito a descrição de circuitos lógicos a partir dessas operações.

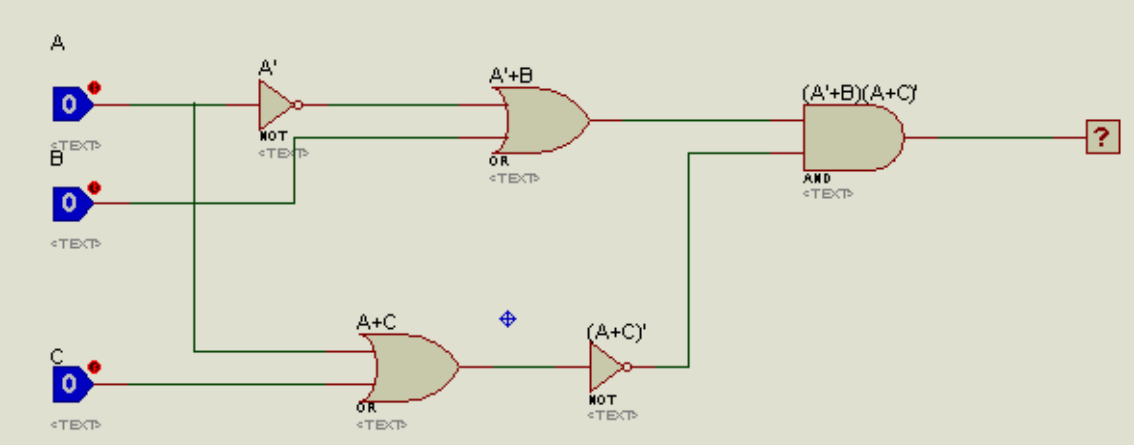


Figura 38 - Análise da sequência de portas lógicas

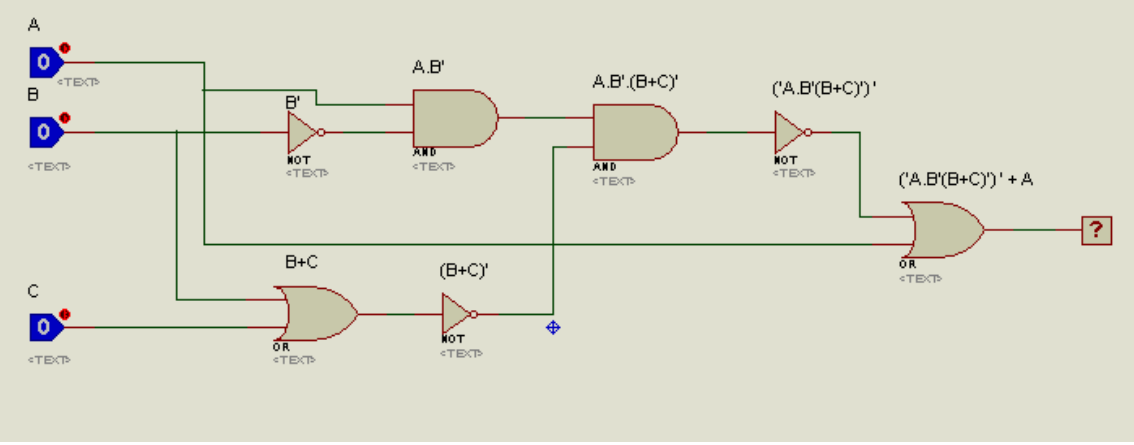


Figura 39 - Outro exemplo de equacionamento de circuito por meio de variáveis

▪ Avaliando as saídas dos Circuitos Lógicos

A partir da expressão booleana para a saída de um circuito, podemos realizar a análise da saída para todo o conjunto de níveis lógicos de entrada. Esse estudo é descrito em uma tabela-verdade, a qual a partir das variáveis de entrada, tem-se a saída do circuito. Para facilitar esse esboço, tem que ser seguidas as seguintes regras:

1. Realizar as operações NOT;
2. Realizar as operações dentro dos parênteses;
3. Realizar as operações AND antes das operações OR;
4. Se uma expressão possuir uma barra sobre, realize a operação descrita pela expressão, em seguida, inverta o sinal.

Como exemplo, a tabela-verdade da Fig. 38 é construída:

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

Fig. 40 - Tabela Verdade do circuito $(\bar{\mathbf{A}} + \mathbf{B})\overline{(\mathbf{A} + \mathbf{C})}$

Simulando o circuito no *Proteus*, aparece o gráfico da Fig. 41, o qual ratifica os valores encontrados na Fig. 40.

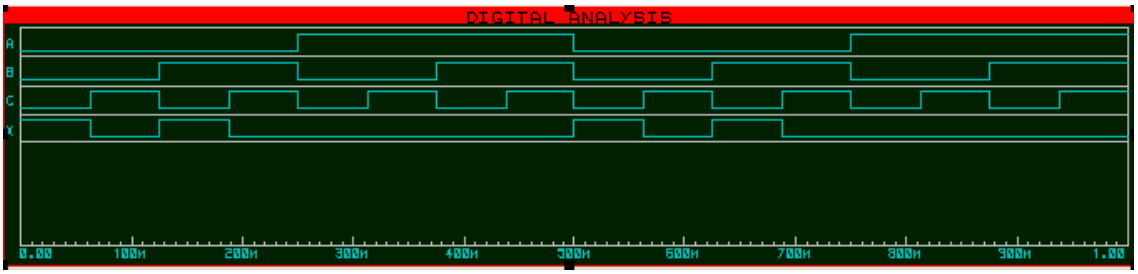


Figura 41 - Tabela-verdade com a simulação no *Proteus*.

▪ Teoremas Booleanos e DeMorgan em circuitos lógicos

Para que sejam otimizados, alguns circuitos podem ser reduzidos a fim de simplifica-lo e, assim, diminuir interferências e ruídos. Para isso, existem os teoremas booleanos já explicados anteriormente.

Exemplo:

i. Simplifique a expressão do circuito da Fig. 42:

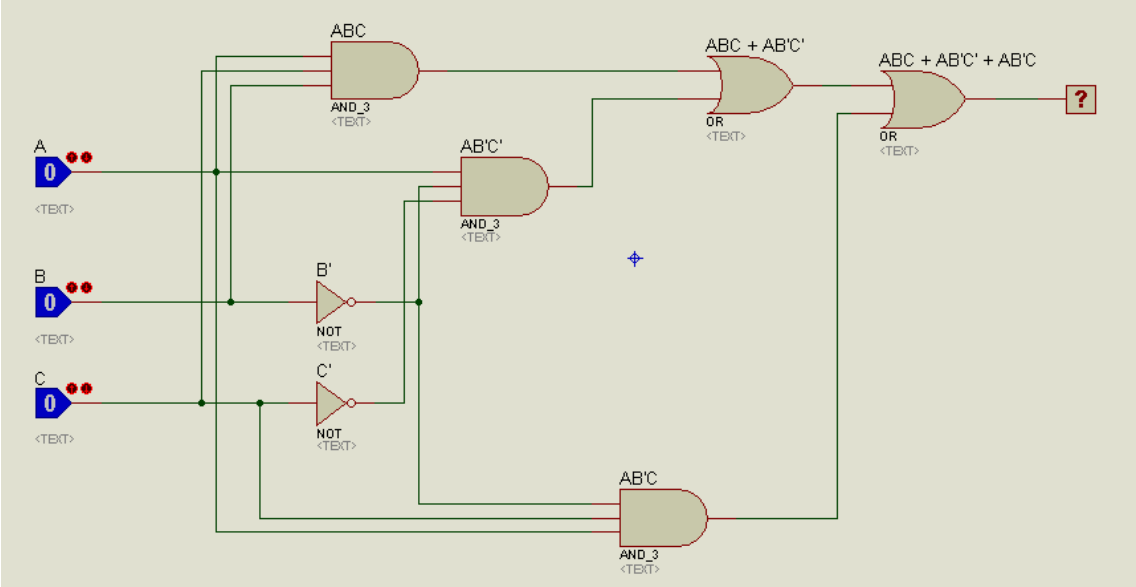


Fig. 42 - Circuito de Exemplo "i"

$z = A\bar{B}\bar{C} + A\bar{B}C + ABC$:

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

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

$= A\bar{B}(1) + ABC$

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

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

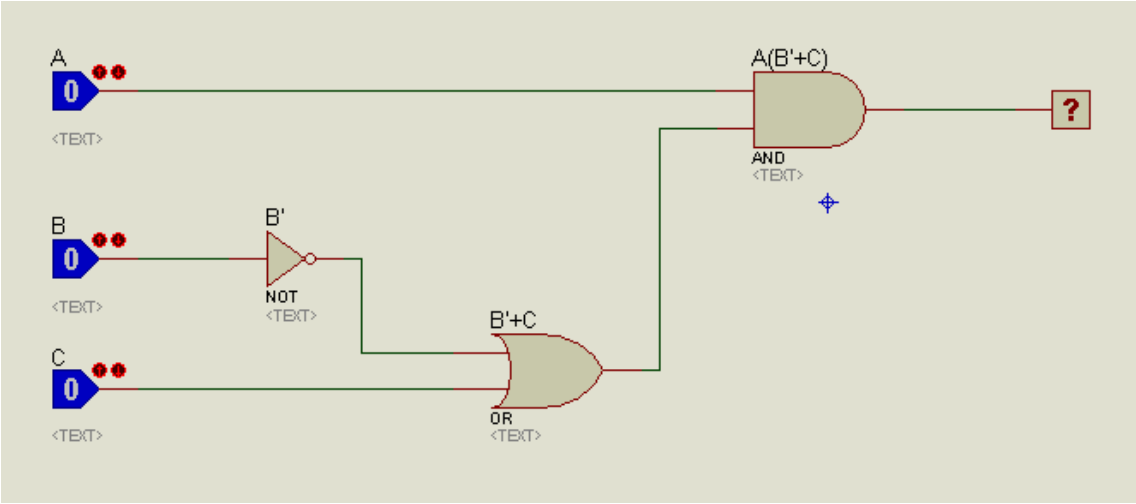


Fig. 43 - Circuito simplificado do Exemplo "i"

ii. Simplifique a expressão do circuito da Fig. 44:

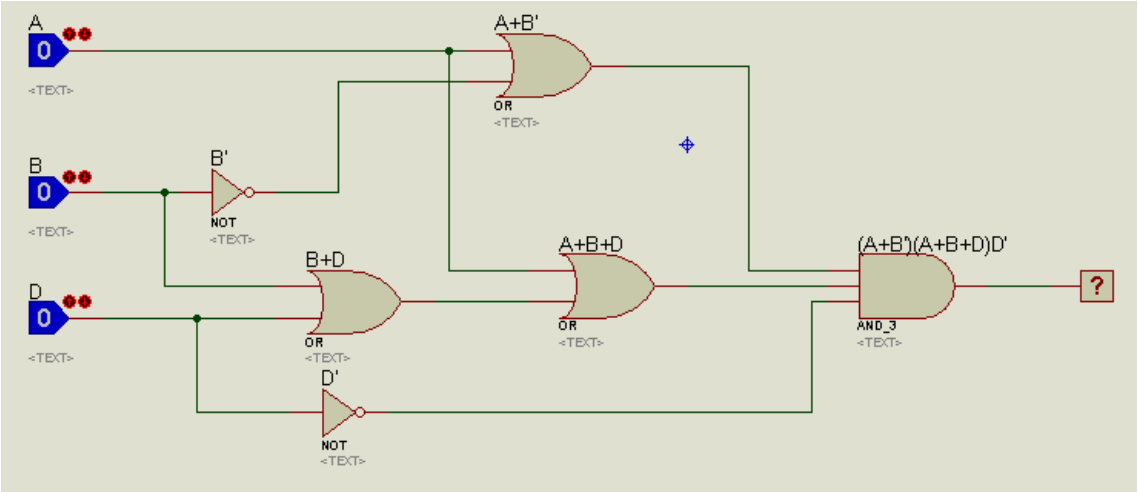


Fig. 44 - Circuito de Exemplo "ii"

$z = (\overline{A} + B)(A + B + D)\overline{D}$:

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

$= \overline{A}A\overline{D} + \overline{A}B\overline{D} + \overline{A}D\overline{D} + BA\overline{D} + BB\overline{D} + BD\overline{D}$

$= \cancel{\overline{A}A\overline{D}} + \overline{A}B\overline{D} + \cancel{\overline{A}D\overline{D}} + BA\overline{D} + BB\overline{D} + \cancel{BD\overline{D}}$

$= \overline{A}B\overline{D} + BA\overline{D} + B\overline{D}$

$= B\overline{D}(\overline{A} + A + 1)$

$z = B\overline{D}$

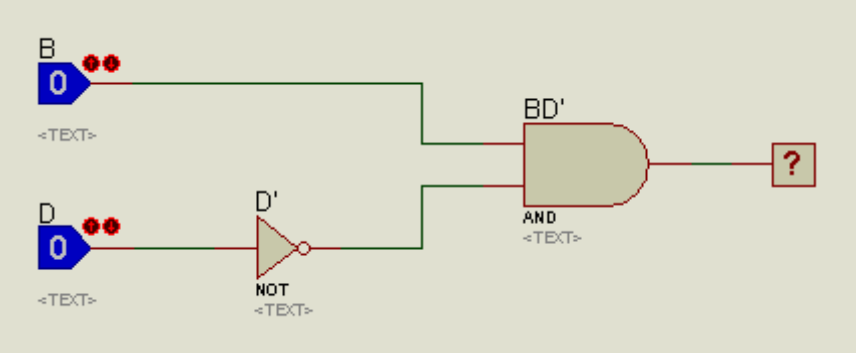


Fig. 45 - Circuito simplificado do Exemplo "ii"

Circuitos Lógicos Combinacionais

O estudo da álgebra booleana e das portas lógicas dão espaço à criação e implementação de circuitos elétricos cada vez mais complexos e com crescente capacidade de "tomar decisões lógicas". A Análise de circuitos dividiu os projetos em dois grandes blocos. Circuitos Combinacionais e Circuitos Sequenciais. Basicamente circuitos combinacionais dependem somente das variáveis de entrada e circuitos sequenciais dependem também da saída anterior. Circuitos sequenciais são o objeto de estudo do próximo capítulo.

O circuitos combinacionais são, em geral, mais básicos. Dependem somente da combinação feita com as variáveis de entrada de tal forma que pode-se entender perfeitamente seu funcionamento com o úso de uma tabela verdade simples de *entrada versus saídas*. Ao se mudar uma entrada, a saída altera seu estado imediatamente e a saída anterior é perdida, ou seja, o projeto não tem um local onde se pode salvar informação, não tem memória. A Fig. 46 mostra o típico exemplo de um circuito combinacional.

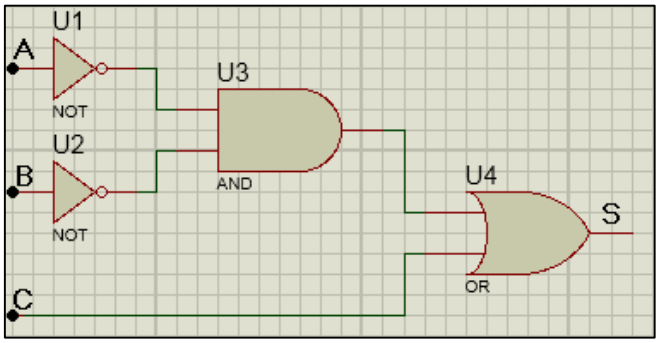


Figura 46 - Exemplo de circuito lógico no Proteus.

- ### Projetando Circuitos Lógicos combinacionais

Com o design de um circuito em mãos, pode-se facilmente definir as saídas em função de todas as entradas e a partir desses dados coletados montar a tabela verdade do circuito. No entanto, um projeto real de engenharia cria a situação contrária: determinado problema existe e o engenheiro é solicitado para criar um projeto que resolva o problema. Ele terá de ver o problema, interpretá-lo, imaginar uma possível solução, criar um projeto e , por fim, implementá-lo. Na interpretação do problema era irá modelar a situação em variáveis de entrada e, em função da ondem em que essas variavéis são dispostas , e possíveis saídas: um problema lógico! Primeiramente,os métodos de simplifcação e projeto de um circuito lógico requerem que as expressões estejam em forma padronizada, as quais podem ser Soma-de-Produtos ou Produto-de-Somas.

Vamos definir primeiramente **Soma-de-Produtos**:

Exemplos:

- $A.C + \bar{C}$
- $A + \bar{A}.\bar{C} + \bar{C}$
- $\bar{A} + C.\bar{D} + E + E.G.K + H.\bar{L}$

Podemos observar que cada uma dessas expressões consiste em uma ou mais variáveis unidas por operações AND e conectadas entre si por meio de portas lógicas OR, as quais realizam a "soma" entre os termos agrupados. Vale lembrar que esses termos podem ser complementados (barrados) ou não complementados.

Agora podemos definir **Produto-de-Somas**:

- $(A + \bar{B} + C).(A + C)$
- $(A + \bar{B}).(C + E)F$
- $(A).(B + C)$

Essa forma é bem menos utilizada para simplificação de expressões algébricas, visto que outra técnica que será vista mais a frente, chamada Mapa-K, fornece diretamente a expressão completa em Soma-de-Produtos. Esse Produto-de-Somas consiste em dois ou mais termos OR conectados por operações lógicas AND. Vale ressaltar que o **Produto-de-Somas** consiste nas mesmas operações lógicas (OR e AND) que na **Soma-de-Produtos,** porém, a ordem que as operações são realizadas é invertida. No **Produto-de-Somas** é realizado primeiramente operações AND e depois OR, contudo, na **Soma-de-Produtos** é realizada primeiramente operações OR e logo depois operações AND.

De posse de uma tabela verdade, podemos agora, por meio dos Produtos-de-Soma ou da Soma-de-Produtos, obter a expressão lógica do circuito. Por exemplo:

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	0
1	1	1	1

Figura 47 - Tabela verdade

Para a Fig. 47, podemos expressar a expressão lógica das duas maneiras citadas anteriormente. Se observamos as saídas 1, poderemos associar as entradas como soma-de-produtos, porém se quisermos utilizar as saídas barradas utilizaremos produto-de-somas. Assim, observando as saídas 1, teremos:

$$S = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC \qquad \textbf{Soma de Produtos} \text{ (Mais utilizada)}$$

ou podemos escrever observando as saídas nulas, tendo o cuidado que o barrado agora é igual a 1.

$$S = (A + B + C)(A + B + \bar{C})(\bar{A} + B + C)(\bar{A} + \bar{B} + C) \qquad \textbf{Produto-de-Soma}$$

Agora, que já sabemos obter a expressão de uma tabela verdade, podemos estudar mais profundamente os métodos de simplificação como a Algébrica e o Mapa-K.

- **Simplificação Algébrica**

Devemos utilizar os teoremas da Álgebra booleana para simplificar as expressões lógicas. Existem casos que a dificuldade de simplificação é grande, porém basta praticar bastante e adquirir mais experiência para resolver esses tipos de problemas tranquilamente.

É aconselhável seguir dois passos:

1. Realiza-se operações de algebra booleana simples para transformar as expressões de produto-de-soma em soma-de-produtos, pode-se aplicar também DeMorgan para simplificar ainda mais.
2. Quando estiver na forma de soma-de-produtos, ou se ela já estiver, verifica-se se os termos possuem fatores em comum. Dessa forma, com algumas manipulações algébricas, por em evidência alguma variável ou somar a zero, é possível simplificar a expressão.

Agora que foi vista a teoria de simplificação das expressões, analisaremos primeiramente as expressões advindas da tabela verdade.

Simplificando a expressão da tabela verdade da seguinte forma:

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

Figura 48 - Tabela verdade do circuito exemplo

Pela Tabela-Verdade da Fig. 48, teremos a seguinte soma de produtos:

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

Observe o exemplo da Fig. 49 a qual contém o circuito original e o circuito simplificado na Fig. 50 após a utilização do método de simplificação algébrica.

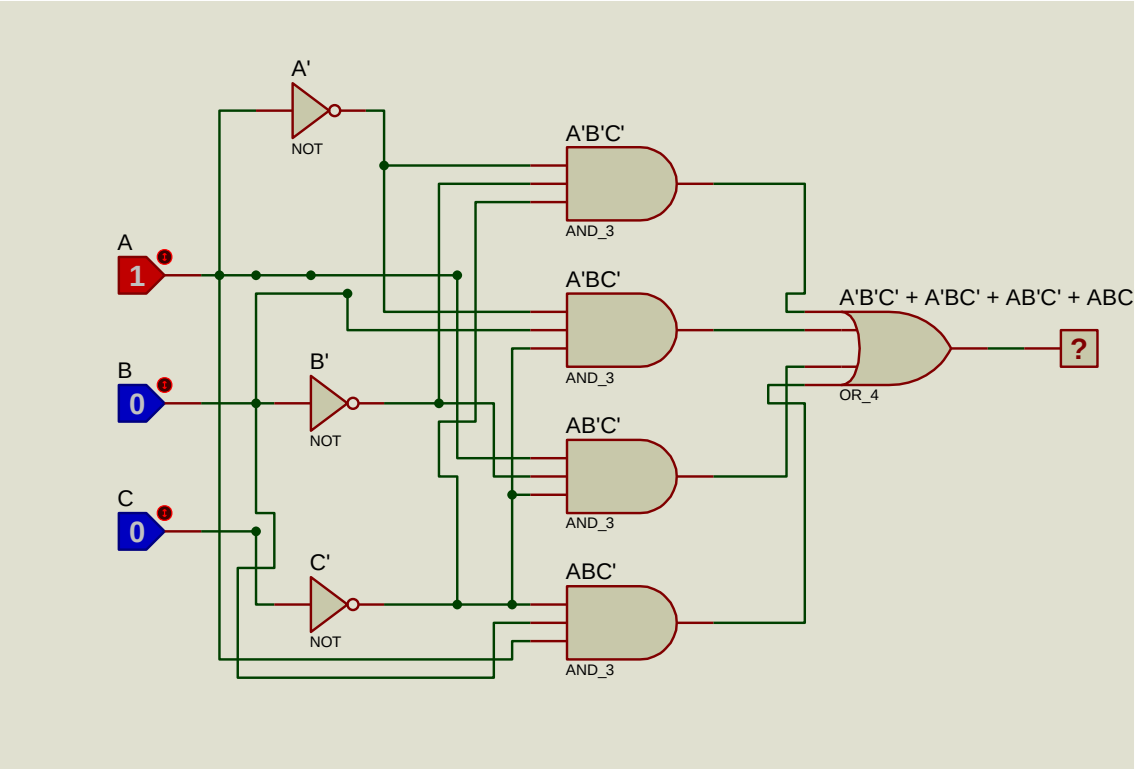


Figura 49 - Circuito lógico referente a tabela

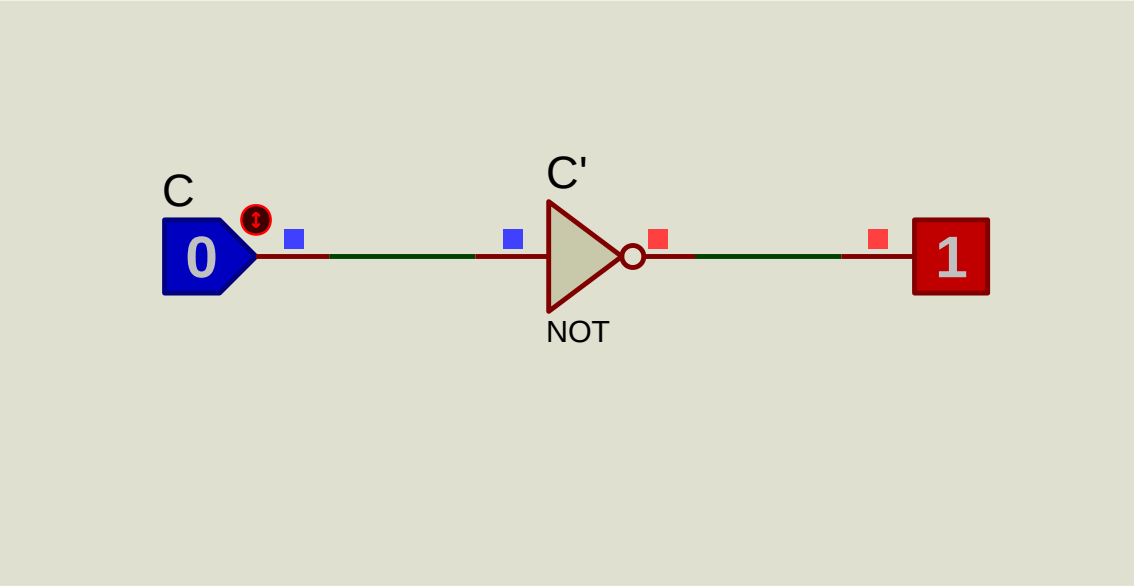


Figura 50 - Circuito lógico simplificado

▪ Mapa de Karnaugh (Mapa-K)

Agora que já sabemos como realizar a simplificação algébrica de maneira correta, podemos conhecer outro método mais autônomo que simplifica graficamente a expressão do circuito em soma-de-produtos a partir da tabela verdade. Essa técnica abordada agora é conhecida como Mapa de Karnaugh.

O Mapa-K é, basicamenete, um método grafico utilizado para simplificar uma expressão lógica ou para converter uma tabela-verdade no seu circuito lógico correspondente mais simplificado na forma de soma de produtos com algoritmos preestabelicidos de fácil aplicação. Nesta apostila será visto Mapa-K com até 4 variáveis, pois o estudo de mais de quatro variáveis no mapa é bem mais complexo, fugindo ao escopo desse curso. O mapa mostra a relação entre entradas e saídas desejada. Alguns fatos sobre o mapa devem ser notados:

- O Mapa-K terá 2^n espaços a serem preenchidos, onde n é o número de entradas. A tabela-verdade fornece valor da saída X para as combinações de entradas, por outro lado no Mapa-K cada linha da tabela corresponde a um espaço (quadrado) a ser preenchido.
- Os espaços (quadrados) são nomeados de forma que os quadrados adjacentes horizontalmente ou verticamente, diferem em apenas uma variável (Por exemplo: 00, 01, 11, 10 em código gray. Traduzindo para as entradas teremos: $\bar{A}\bar{B}, \bar{A}B, AB, A\bar{B}$). Vale ressaltar também que 00 é adjancente a 10 pois diferem em apenas uma variável, no mapa será confuso visualizar esse fato incialmente, mas após algumas vizualizações ficará mais intuitivo.
- A saída da expressão será dada pela simplificação dos agrupamentos de números 1s. Após realizar cada agrupamento, é necessário realizar uma operação OR entre eles para ter a saída final.

Temos os seguintes exemplos de Mapa-K:

Mapa-K → Duas Variáveis.

Tabela Verdade:

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Expressão Lógica:

$X = \bar{A}\bar{B}$

Mapa de Karnaugh:

	$\bar{B}$	B
$\bar{A}$	1	0
A	0	0

Mapa-K → Três Variáveis.

Tabela Verdade:

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

Expressão Lógica:

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

Mapa de Karnaugh:

	$\bar{C}$	C
$\bar{A}\bar{B}$	1	0
$\bar{A}B$	0	1
AB	0	1
$A\bar{B}$	1	0

Mapa-K → Quatro Variáveis.

Tabela Verdade:

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

Expressão Lógica:

$$X = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D} + \bar{A}BCD + AB\bar{C}\bar{D} + ABC\bar{D} + ABCD + AB\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D}$$

Mapa de Karnaugh:

	$\bar{C}\bar{D}$	$C\bar{D}$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	0	0	1
$\bar{A}B$	0	1	1	0
AB	1	1	1	1
$A\bar{B}$	1	0	0	1

Agora que já sabe-se como montar o Mapa-K, poderemos simplificar agora as expressões de saída com os agrupamentos de um par, um quarteto ou um octeto de 1s.

Agrupando um par:

Expressão Lógica:

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

(Observamos que uma dupla, retira uma variável da expressão)

Mapa de Karnaugh:

	$\bar{C}$	C
$\bar{A}\bar{B}$	1	1
$\bar{A}B$	0	0
AB	0	0
$A\bar{B}$	0	0

Agrupando um quarteto:

Expressão Lógica:

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

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

$X = \bar{A}\bar{C} + A\bar{C} = \bar{C}$

(Um quarteto elimina duas variáveis)

Mapa de Karnaugh:

	$\bar{C}$	C
$\bar{A}\bar{B}$	1	0
$\bar{A}B$	1	0
AB	1	0
$A\bar{B}$	1	0

Agrupando um octeto:

Expressão Lógica:

$X = \bar{D}$

(Um octeto elimina três variáveis)

Mapa de Karnaugh:

	$\bar{C}\bar{D}$	$C\bar{D}$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	0	0	1
$\bar{A}B$	1	0	0	1
AB	1	0	0	1
$A\bar{B}$	1	0	0	1

Podemos concluir que quando uma variável aparece nas formas complementadas e não complementada em um agrupamento, essa variável é eliminada, já as variáveis que não se alteram para todos os agrupamentos permanecem na expressão final. Um simples passo-a-passo pode facilitar a simplificação de um Mapa-K:

1. Construa o Mapa-K com os respectivos 1s.
2. Agrupe primeiramente os 1s que não possuem outros 1s adjacentes, ou seja, são os 1s isolados.
3. Procure os 1s que são adjacentes a somente outros 1s e agrupe todos os pares que o contenham
4. Agrupe agora os octetos mesmo que contenha algum 1 já agrupado anteriormente.
5. Agrupe os quartetos, que contenha um ou mais 1s que ainda não tenha sido agrupado.
6. Agrupa agora os pares para incluir os outros 1s que não foram agrupados.
7. Some agora na forma OR, de soma de produtos.

OBS$_1$: A todo o momento foi utilizada a palavra adjacentes, pois como dito anteriormente 00 e 10 são adjacentes, mas não vizinhos no Mapa-K

OBS$_2$: Certifique-se de utilizar o menor número de agrupamentos.

Agora, iremos colocar alguns exemplos resolvidos para melhor entendimento:

Exemplo 1:

Tabela Verdade:

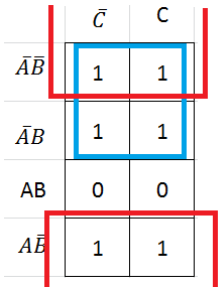
A	B	C	X
0	0	0	1
0	0	1	1
0	1	0	1

Mapa -K:

Expressão Simplificada:

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

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

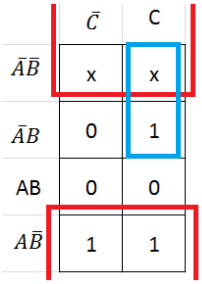


Existem também as condições de "*don't care*" que ocorre quando existem certas condições de entrada para as quais não existem níveis de saída especificada, normalmente essas condições nunca ocorrerão. Para estas condições a saída não é nem 0 nem 1, é marcada por um "x" que indica "*don't care*". Como não há uma saída especificada, é livre a escolha para fazer a saída ser zero ou um de forma mais conveniente para obtermos a expressão mais simples, logo, basta manipular esse "x" para obter o melhor agrupamento no Mapa-K. Por exemplo:

Tabela Verdade:

A	B	C	X
0	0	0	x
0	0	1	X
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Mapa -K:



Expressão Simplificada:

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

○ *Display* de Sete Segmentos.

Uma aplicação bastante didática de Eletrônica Digital é o *Display* de Sete Segmentos. Um *display* de sete segmentos é composto de sete elementos que podem ser ligados ou desligados individualmente. Para formar cada número (ou letras do alfabeto no caso de números Hexadecimal), devem ser ligados os elementos necessários para que, quando acessos, formem o número (ou a letra) em questão. Abaixo são indicados quais elementos devem ser acessados para formar cada número.

Número	
0 (zero)	a, b, c, d, e
1 (um)	b, c
2 (dois)	a, b, d, e, g
3 (três)	a, b, c, d, g
4 (quatro)	b, c, f, g
5 (cinco)	a, c, d, f, g
6 (seis)	a, c, d, e, f, g
7 (sete)	a, b, c
8 (oito)	a, b, c, d, e, f, g
9 (nove)	a, b, c, f, g

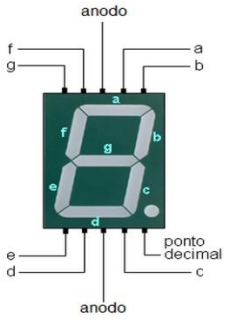
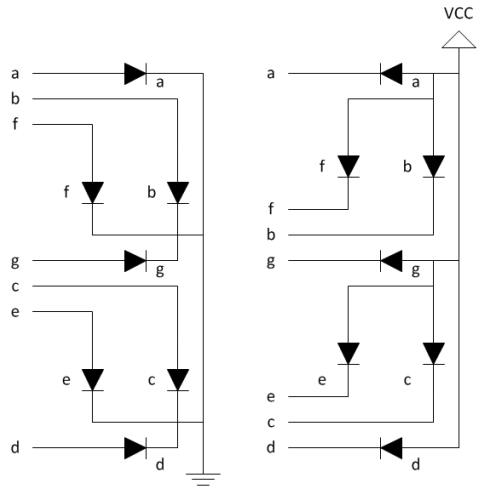


Figura 51 - circuito interno de um display de 7 segmentos

Existem duas configurações para os *displays* de sete segmentos: Ânodo Comum e Catodo Comum.

- **Anodo Comum:** Os *leds* do *display* estão ligados ao VCC (5V) logo, para acender cada *led*, é necessário aplicar uma entrada em nível baixo (zero) no pino correspondente ao *led* que deverá ser acendido.
- **Catodo Comum:** Os *leds* do *display* estão ligados ao GND (0V) logo, para acender cada *led*, é necessário aplicar uma entrada em nível alto (um) no pino correspondente ao *led* que deverá ser acendido.

Em alguns casos, é necessário utilizar Resistores de *Pull-Up* ou *Pull-Down* para evitar a flutuação dos pinos e garantir a entrada no nível lógico desejado. Os resistores de *Pull-Up* vão do pino do condutor a um VCC (+5V) garantindo que se esta entrada for desconectada, ela estará sempre em nível lógico alto. Já os resistores de *Pull-Down* vão do pino do condutor a um GND (0V) garantindo que se esta entrada for desconectada, ela estará sempre em nível lógico baixo. Os *Pull-Down* podem ser utilizados por exemplo em CI's da família CMOS, os quais possuem entradas controlados por tensão. Os pinos não podem flutuar, sendo necessário colocar um resistor de *pull-down* nos mesmos. Atualmente, o *display* mais comercializado é o do tipo ânodo comum.



Cátodo Comum Ânodo Comum

Figura 48 - circuito interno de um display de 7 segmentos

Agora que já foi visto todas as características dos Mapas-K é possível facilmente projetar e montar um *display* de sete segmentos. Primeiramente, considerando somente as entradas X0 e X1 de um conversor BCD para 7 segmentos, e mostrar os valores 0, 1, 2 e 3 (pois somente esses valores são possíveis com 2 bits) de BCD para 7 segmentos. Deve-se montar a tabela-verdade levando em consideração qual posição no *display* nós queremos que acenda e colocaremos um 1 onde for acender na tabela (pois o *display* é catodo comum). Assim para a montagem do Mapa-K as entradas serão X1 e X0 e as saídas cada letra do *display* individualmente.

Teremos a tabela formada:

BCD	X1	X0	a	b	c	d	e	f	g
0	0	0	1	1	1	1	1	1	0
1	0	1	0	1	1	0	0	0	0
2	1	0	1	1	0	1	1	0	1
3	1	1	1	1	1	1	0	0	1

E os seguintes Mapas-K formados:

$A = X1 + \overline{X0}$

	$\overline{X0}$	$X0$
$\overline{X1}$	1	0
$X1$	1	1

$B = 1$

	$\overline{X0}$	$X0$
$\overline{X1}$	1	1
$X1$	1	1

$C = \overline{X1} + X0$

	$\overline{X0}$	$X0$
$\overline{X1}$	1	1
$X1$	0	1

$D = X1 + \overline{X0}$

	$\overline{X0}$	$X0$
$\overline{X1}$	1	0
$X1$	1	1

$E = \overline{X0}$

	$\overline{X0}$	$X0$
$\overline{X1}$	1	0
$X1$	1	0

$F = \overline{X1}.\overline{X0}$

	$\overline{X0}$	$X0$
$\overline{X1}$	1	0
$X1$	0	0

$G = X1$

	$\overline{X0}$	$X0$
$\overline{X1}$	0	0
$X1$	1	1

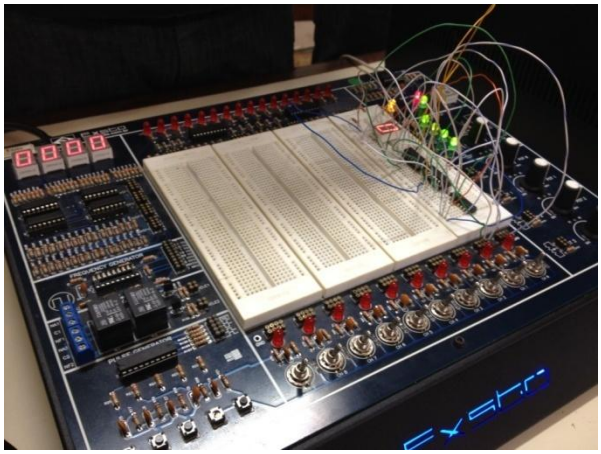


Figura 49 - Display de Sete Segmentos BCD (0-3)

A seguir, será feito o mapa de um circuito que converta todos os possíveis valores de BCD (ou seja, de 0 a 9) para um *display* de 7 segmentos. Segue o mapa de *Karnaugh* das entradas de A a G, porém as condições X de "*don't care*" foram utilizadas (pois com 4 *bits* podemos ir além de 9) para se obter o menor número de soma de produtos possíveis.

	$\overline{X1}.\overline{X0}$	$\overline{X1}.X0$	$X1.X0$	$X1.\overline{X0}$
$\overline{X3}.\overline{X2}$	1	0	1	1
$\overline{X3}.X2$	0	1	1	1
$X3.X2$	X	X	X	X
$X3.\overline{X2}$	1	1	X	X

→

	$\overline{X1}.\overline{X0}$	$\overline{X1}.X0$	$X1.X0$	$X1.\overline{X0}$
$\overline{X3}.\overline{X2}$	1	0	1	1
$\overline{X3}.X2$	0	1	1	1
$X3.X2$	1	1	1	1
$X3.\overline{X2}$	1	1	1	1

$A = X3 + X1 + X2.X0 + \overline{X2}.\overline{X0}$

	$\overline{X1}.\overline{X0}$	$\overline{X1}.X0$	$X1.X0$	$X1.\overline{X0}$
$\overline{X3}.\overline{X2}$	1	1	1	1
$\overline{X3}.X2$	1	0	1	0
$X3.X2$	X	X	X	X
$X3.\overline{X2}$	1	1	X	X

→

	$\overline{X1}.\overline{X0}$	$\overline{X1}.X0$	$X1.X0$	$X1.\overline{X0}$
$\overline{X3}.\overline{X2}$	1	1	1	1
$\overline{X3}.X2$	1	0	1	0
$X3.X2$	1	1	1	1
$X3.\overline{X2}$	1	1	1	1

$B = X3 + \overline{X2} + X1.X0 + \overline{X1}.\overline{X0}$

	$\overline{X1}.\overline{X0}$	$\overline{X1}.X0$	$X1.X0$	$X1.\overline{X0}$
$\overline{X3}.\overline{X2}$	1	1	1	0

	$\overline{X1}.\overline{X0}$	$\overline{X1}.X0$	$X1.X0$	$X1.\overline{X0}$
$\overline{X3}.\overline{X2}$	1	1	1	0

$\overline{X3}.X2$	1	1	1	1	→	$\overline{X3}.X2$	1	1	1	1
$X3.X2$	X	X	X	X		$X3.X2$	1	1	1	1
$X3.\overline{X2}$	1	1	X	X		$X3.\overline{X2}$	1	1	1	1

$C = X3 + X2 + \overline{X1} + X0$

	$\overline{X1}.\overline{X0}$	$\overline{X1}.X0$	$X1.X0$	$X1.\overline{X0}$			$\overline{X1}.\overline{X0}$	$\overline{X1}.X0$	$X1.X0$	$X1.\overline{X0}$
$\overline{X3}.\overline{X2}$	1	0	1	1		$\overline{X3}.\overline{X2}$	1	0	1	1
$\overline{X3}.X2$	0	1	0	1	→	$\overline{X3}.X2$	0	1	0	1
$X3.X2$	X	X	X	X		$X3.X2$	1	1	1	1
$X3.\overline{X2}$	1	1	X	X		$X3.\overline{X2}$	1	1	1	1

$D = X3 + \overline{X2}.X1 + X1.\overline{X0} + X2.\overline{X1}.X0 + \overline{X2}.\overline{X0}$

	$\overline{X1}.\overline{X0}$	$\overline{X1}.X0$	$X1.X0$	$X1.\overline{X0}$			$\overline{X1}.\overline{X0}$	$\overline{X1}.X0$	$X1.X0$	$X1.\overline{X0}$
$\overline{X3}.\overline{X2}$	1	0	0	1		$\overline{X3}.\overline{X2}$	1	0	0	1
$\overline{X3}.X2$	0	0	0	1	→	$\overline{X3}.X2$	0	0	0	1
$X3.X2$	X	X	X	X		$X3.X2$	0	0	0	1
$X3.\overline{X2}$	1	0	X	X		$X3.\overline{X2}$	1	0	0	1

$E = \overline{X2}.\overline{X0} + X1.\overline{X0}$

	$\overline{X1}.\overline{X0}$	$\overline{X1}.X0$	$X1.X0$	$X1.\overline{X0}$			$\overline{X1}.\overline{X0}$	$\overline{X1}.X0$	$X1.X0$	$X1.\overline{X0}$
$\overline{X3}.\overline{X2}$	1	0	0	0		$\overline{X3}.\overline{X2}$	1	0	0	0
$\overline{X3}.X2$	1	1	0	1	→	$\overline{X3}.X2$	1	1	0	1
$X3.X2$	X	X	X	X		$X3.X2$	1	1	1	1
$X3.\overline{X2}$	1	1	X	X		$X3.\overline{X2}$	1	1	1	1

$F = X3 + X2.\overline{X1} + X2.\overline{X0} + \overline{X1}.\overline{X0}$

	$\overline{X1}.\overline{X0}$	$\overline{X1}.X0$	$X1.X0$	$X1.\overline{X0}$			$\overline{X1}.\overline{X0}$	$\overline{X1}.X0$	$X1.X0$	$X1.\overline{X0}$
$\overline{X3}.\overline{X2}$	0	0	1	1		$\overline{X3}.\overline{X2}$	0	0	1	1
$\overline{X3}.X2$	1	1	0	1	→	$\overline{X3}.X2$	1	1	0	1
$X3.X2$	X	X	X	X		$X3.X2$	1	1	1	1
$X3.\overline{X2}$	1	1	X	X		$X3.\overline{X2}$	1	1	1	1

$G = X3 + X2.\overline{X1} + \overline{X2}.X1 + X1.\overline{X0}$

Circuito *Display* de Sete Segmentos BCD 0-9 (PROTEUS):

X3 X2 X1 X0

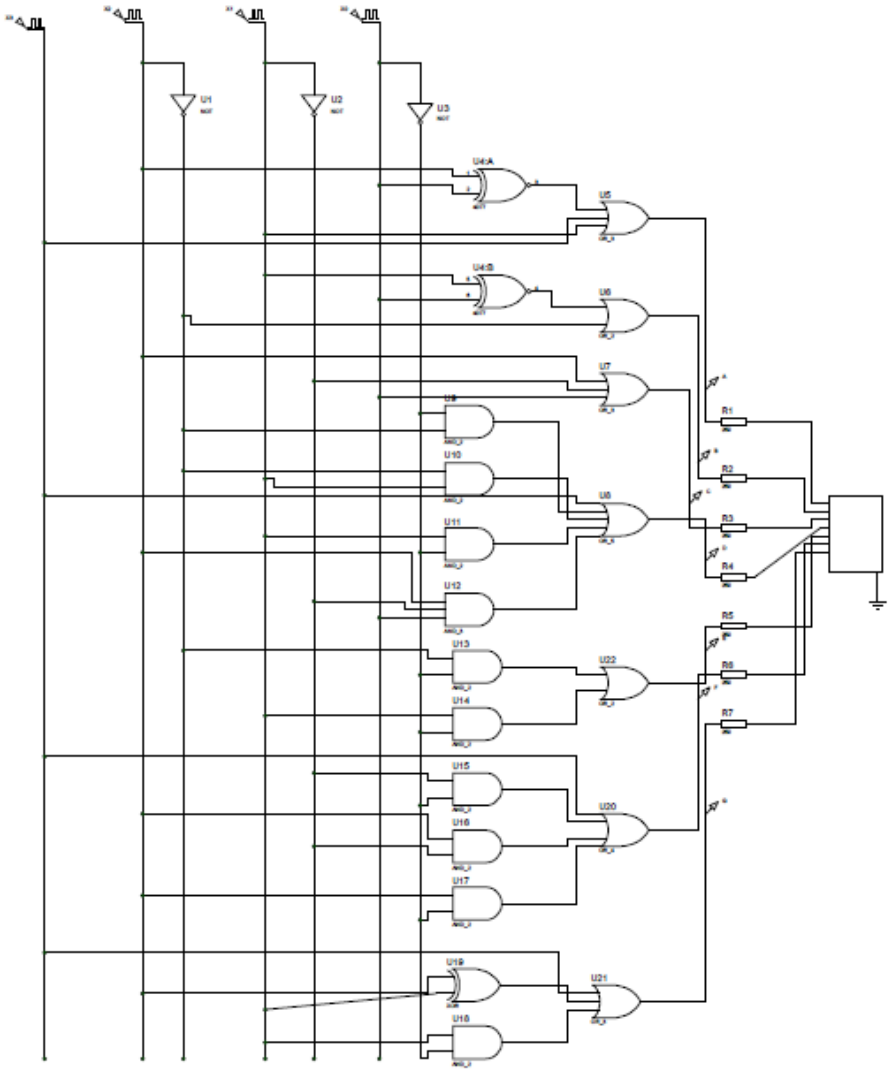


Figura 50 - Esquemático do Proteus

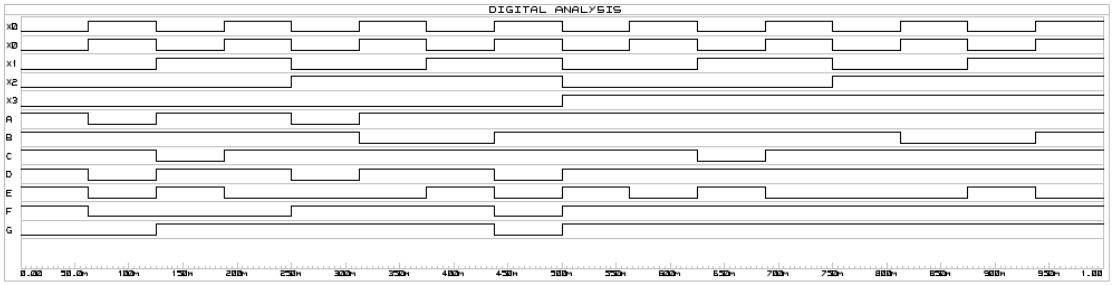


Figura 51 - Gráfico da Simulação (Proteus)

Projetar um simples *display* de sete segmentos BCD 0-9 exige bastante trabalho e esforço, porém existem outros CI's que foram feitos para facilitar esta aplicação, como por exemplo, os circuitos integrados 7447 e 7448 que são decodificadores para *display* de sete segmentos. Esses CI's recebem um código BCD e decodificam para o *display* de 7 segmentos, ou seja, eles já realizam todo o trabalho por meio de um circuito lógico próprio interno de transformar as entradas BCD para o *display* de sete segmentos. Dessa forma ele decodifica 4 entradas em 7 saídas.

Os dois CI's se diferenciam pelo fato do CI 7447 ser ânodo comum, enquanto que o CI 7448 é do tipo cátodo comum.

Circuitos Lógicos Sequenciais

Agora estudaremos uma nova categoria de circuitos lógicos. São os circuitos lógicos sequenciais. Eles diferem dos circuitos lógicos combinacionais pelo fato de terem memória, ou seja, a saída do sistema depende não só do estado atual da entrada, mas de estados anteriores. O principal elemento de memória utilizado é o Flip-Flop. Ele é constituído por portas lógicas que, sozinhas, não têm capacidade de armazenamento, mas conectadas entre si transformam o circuito num sistema com memória.

Esse conjunto de portas lógicas que formam o Flip-Flop pode ser representado pela Fig. 52:

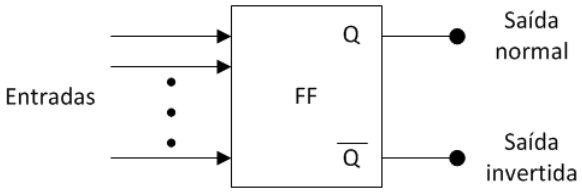


Figura 52 - Representação de um Flip-Flop genérico

As entradas de um Flip-Flop servem para controlar os estados de saída. Porém, as entradas precisam ser alteradas apenas momentaneamente para que as saídas mudem e permaneçam no novo estado. O que mostra a propriedade de memória desse dispositivo.

- ***Latchs* com portas NAND e portas NOR**

O primeiro exemplo de Flip-Flop que mostraremos é chamado *latch*, e é formado apenas por portas NAND, com duas entradas e as saídas Q e $\overline{Q}$, como mostrado na Fig. 53:

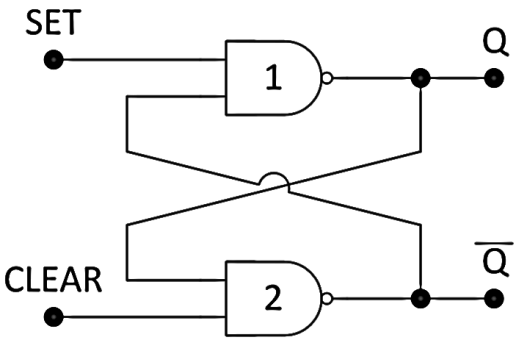


Figura 53 - Representação de um *latch* com portas NAND

Analisemos agora as quatro possibilidades de combinação de entradas *SET* e *CLEAR* (*RESET*). Para *SET* = *RESET* = 1, vemos que tanto podemos ter Q=0 como Q=1. Esse é chamado "estado de repouso" do latch. Com um pulso negativo na entrada *SET*, temos a combinação 01 para a entrada. Com isso, fazemos com que Q=1, "setando" o *latch*, independente de como estava a saída anteriormente. Ao retornarmos para a combinação 11 na entrada, com o fim do pulso, a saída não se altera.

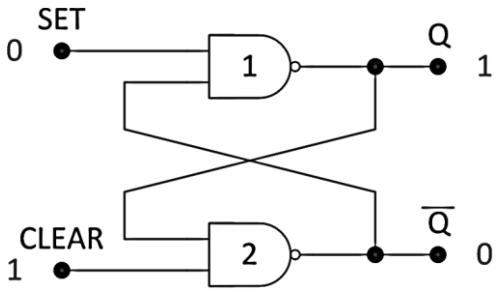


Figura 54 - Setando o *latch*

Com um pulso negativo na entrada *RESET*, temos a combinação 10 na entrada, o que faz com que Q=0, "resetando" o *latch*, não dependendo da entrada anterior. A saída permanece a mesma quando o pulso acaba e a entrada retorna para 11.

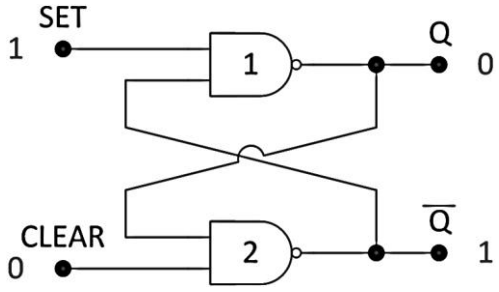


Figura 55 - Resetando o *latch*

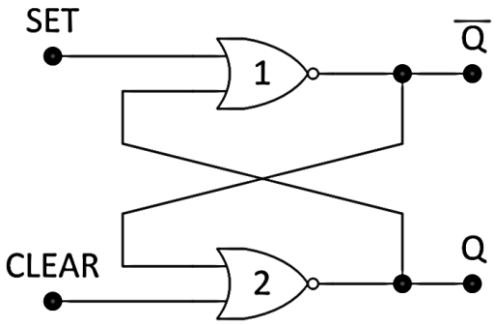
A combinação 00 na entrada é considerada indesejada, pois faz com que $Q = \overline{Q} = 1$, o que contradiz a característica do *Flip-Flop* de ter saídas complementares. Essa combinação, portanto, não é utilizada. Encontramos um resumo da discussão acima na tabela verdade da Fig. 56:

Set	Reset / Clear	Saída
1	1	Não muda
0	1	Q = 1
1	0	Q = 0
0	0	Inválida*

* Produz Q = $\overline{Q}$ = 1

Figura 56 - Tabela-verdade do *latch* com portas NAND

Outro tipo de *latch* pode ser formado com duas portas NOR, com a mesma estrutura do *latch* com portas NAND. Na Fig. 57 é mostrado o circuito e a tabela-verdade que o representa, a qual pode ser verificada pelo leitor como exercício:



Set	Reset / Clear	Saída
0	0	Não muda
1	0	Q = 1
0	1	Q = 0
1	1	Inválida*

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

Figura 57 - *Latch* com portas NOR e tabela-verdade

Podemos observar que as entradas do *latch* com portas NAND são ativas em nível baixo e as do *latch* com portas NOR são ativas em nível alto. Nas Fig. 58 e 59, as representações dos dois *latchs* com destaque para as entradas:

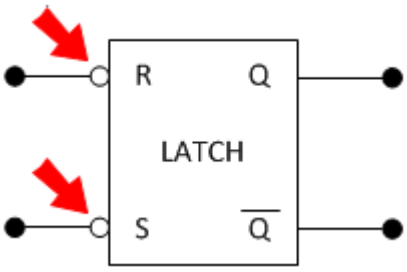


Figura 58 - Representação de *latch* com portas NAND

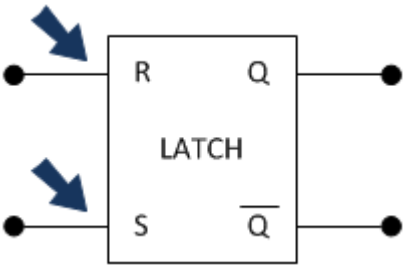


Figura 59 - Representação de *latch* com portas NOR

- **Sistemas assíncronos x Sistemas síncronos**

Podemos dizer que os *latchs* são caracterizados por serem *Flip-Flops* assíncronos, mas o que significa esse conceito? Significa que a saída desses circuitos lógicos pode ser alterada a qualquer momento em que uma ou mais entradas mudem de estado.

A grande maioria dos *Flip-Flops* é caracterizada pela sincronicidade, ou seja, a saída só muda em momentos exatos, definidos por um sinal externo chamado sinal de *clock*. Sistemas síncronos são muito mais fáceis de projetar e analisar.

Circuitos Lógicos Sequenciais Síncronos têm como base os *Flip-Flops* com *clock*. Estes dispositivos tem uma entrada denominada CLK, CK ou CP, e geralmente ela é disparada por borda (o que é representado por um triângulo na entrada). Na Fig. 60 temos um *Flip-Flop* com *clock* ativado por borda de subida e outro ativado por borda de descida:

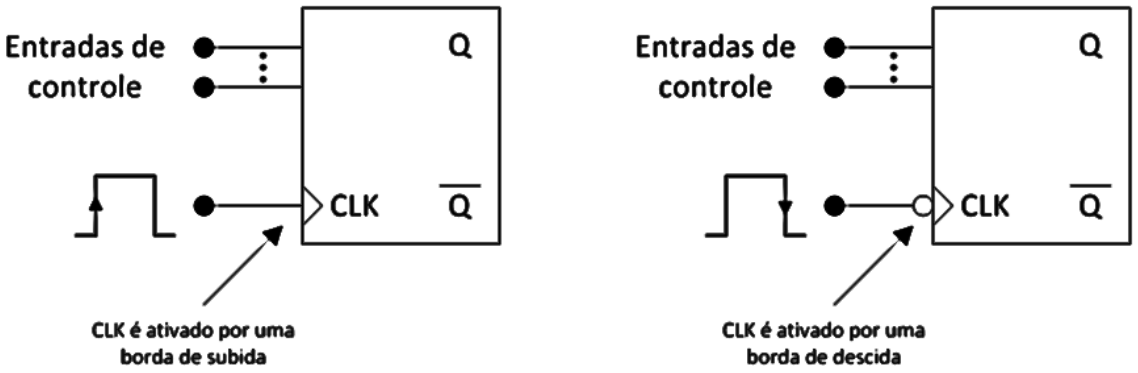
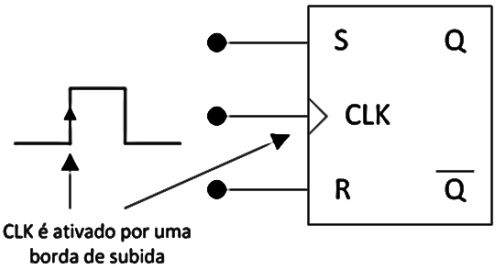


Figura 60 - Representação de Flip-Flops síncronos

Podemos perceber que ambos têm uma ou mais entradas de controle, as quais variam em número e função dependendo do tipo de Flip-Flop que utilizarmos. No entanto, as entradas de controle só terão efeito na saída numa transição ativa do *clock* (borda de subida ou descida dependendo do tipo), ou seja, as entradas deixam as saídas do FF (*Flip-Flop*) prontas para mudarem de estado, mas elas só mudarão na transição ativa.

- ***Flip-Flop* S-R com *clock***

O primeiro tipo de *Flip-Flop* síncrono que estudaremos é uma extensão da definição de *latch*, com a diferença de ser síncrono. Na Fig. 61 estão a tabela-verdade e a representação gráfica de um FF S-R com *clock*, entradas ativas em nível alto e disparado por borda de subida:



Entradas			Saída
S	R	CLK	Q
0	0	↑	Q_0 (Não muda)
1	0	↑	1
0	1	↑	0
1	1	↑	Ambíguo

Q_0 é o nível de saída anterior a ↑ de CLK.
↓ de CLK não produz mudança em Q.

Figuras 61 - Flip-Flop S-R e sua tabela-verdade

A seta para cima na tabela indica a necessidade da borda de subida do cliock para que a saída mude. Q0 representa o estado em que o Flip-Flop estava antes da transição ativa do clock.

Para entender melhor como funciona um FF síncrono, observe o exemplo abaixo de formas de onda de entrada e saída de um FF S-R com *clock*:

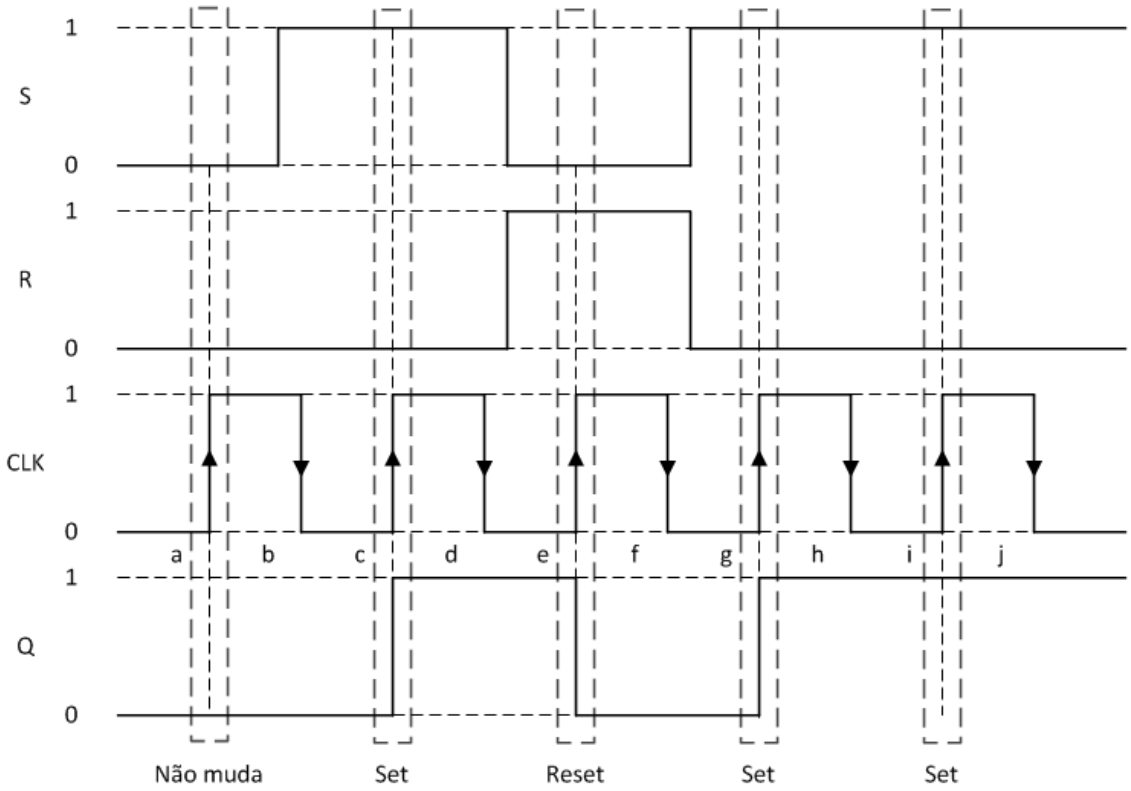
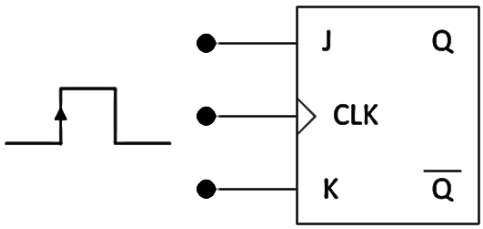


Figura 62 - Formas de onda de entrada e saída de um FF S-R com *clock*

- ***Flip-Flop* J-K com *clock***

Com o objetivo de dar uma função para a combinação de entrada 11 no FF S-R, foi criado o FF J-K. Como podemos observar na tabela-verdade abaixo, a combinação que antes era indesejada agora tem a função de inverter o estado do FF. Ou seja, se Q0 é o estado do FF antes da transição ativa do *clock* e as duas entradas são ativadas, a nova saída será o inverso, $\overline{Q_0}$. A Fig. 63 mostra a tabela-verdade e a representação desse novo tipo de FF.



J	K	CLK	Q
0	0	↑	Q_O (Não muda)
1	0	↑	1
0	1	↑	0
1	1	↑	$\overline{Q_o}$ (Comuta)

Figura 63 - Flip-Flop J-K e sua tabela-verdade

Apresentamos aqui também um exemplo de forma a ilustrar o funcionamento do FF J-K com *clock*:

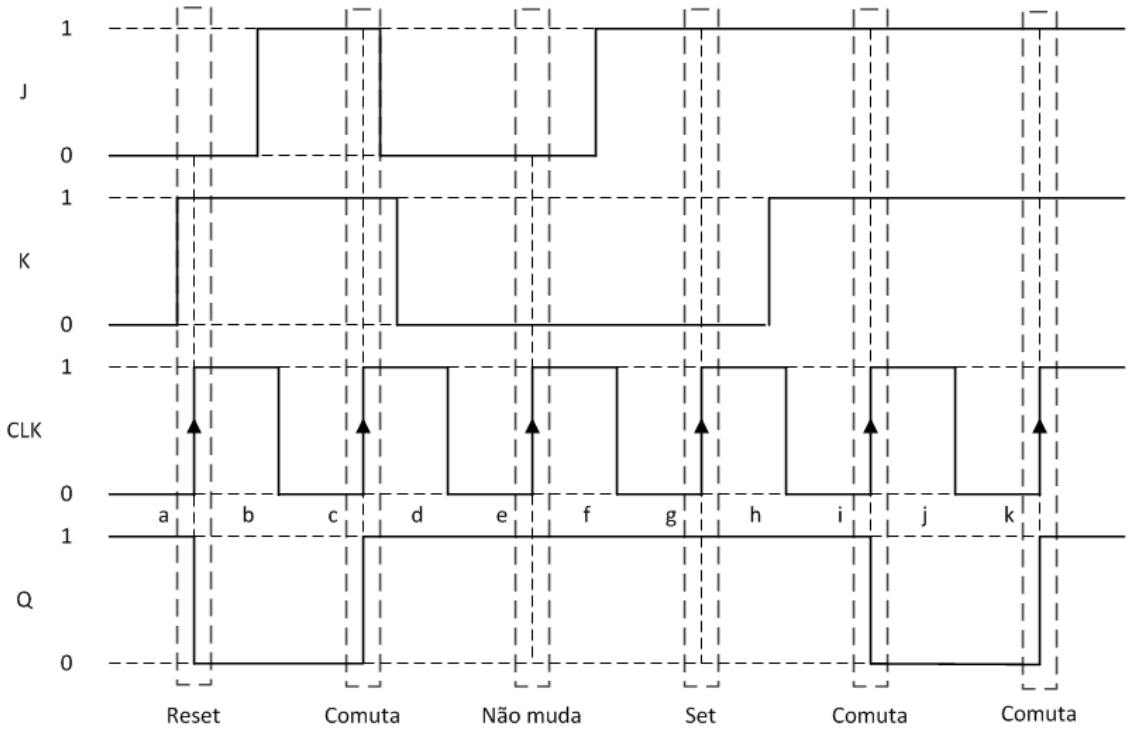
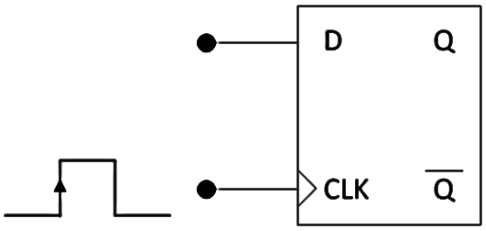


Figura 64 - Formas de onda de entrada e saída de um FF J-K com *clock*

- ***Flip-Flop* D com *clock***

O terceiro tipo de FF que apresentaremos aqui tem apenas uma entrada, e o dispositivo funciona como um armazenador de dados. A representação e a tabela-verdade são bem simples:



D	CLK	Q
0	↑	0
1	↑	1

Figura 65 - Flip-Flop D e sua tabela verdade

O FF tipo D armazena a informação que está na entrada no instante em que ocorre a transição ativa do *clock*, modificando a saída Q de forma que Q = D. Tendo uma forma de onda de entrada assíncrona, podemos usar o *Flip-Flop* D para sincronizar a entrada com o sinal de *clock*, como no exemplo abaixo:

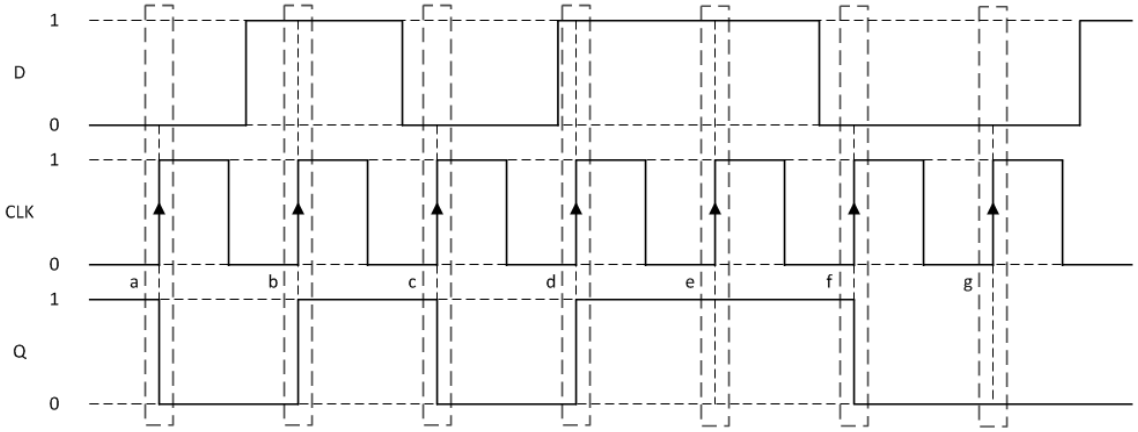
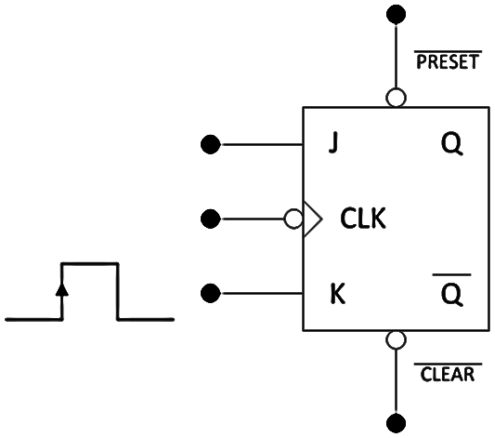


Figura 66 - Formas de onda de entrada e saída de um FF D com *clock*

- **Entradas assíncronas**

Mesmo que o funcionamento síncrono dos FFs estudados até agora facilitem o projeto e a análise dos dispositivos, surge a necessidade de entradas que sejam independentes do sinal de *clock*. Essas entradas são chamadas de entradas de sobreposição, pois anulam o efeito das entradas de controle quando necessário. Temos a representação dessas entradas num FF J-K, assim como a tabela-verdade, na figura abaixo:



$\overline{PRESET}$	$\overline{CLEAR}$	Resposta do FF
1	1	Operação com clock*
0	1	Q = 1 (Independente do CLK)
1	0	Q = 0 (Independente do CLK)
0	0	Não usada

*Q responderá a JK e CLK

Figura 67 - entradas assíncronas e tabela-verdade

Vemos que, para as combinações 01 e 10, a operação do FF independe do *clock* e das entradas síncronas ou de controle. Para a combinação 11 o FF opera de acordo com o *clock* e as entradas de controle, seguindo as tabelas-verdade mostradas anteriormente. A combinação 00 não é utilizada.

Como exemplo de aplicação dessas entradas, podemos ter um circuito lógico sequencial formado por vários FFs J-K cujas entradas *CLEAR* estejam todas conectadas. Como será visto mais a frente, podemos projetar contadores com circuitos desse tipo, como um contador de 0 a 3. Ao acabar a contagem, podemos mandar um pulso negativo na entrada *CLEAR* de todos os FFs e recomeçar a contagem do zero.

Máquinas de Estado

O Flip-Flop pode ser utilizado de maneira a criar dispositivos mais robustos com capacidades de guardar informações, dividir frequências, entre outras aplicações. Esses dispositivos são chamados de Máquinas de Estados Finitos ou somente de Máquinas de Estado.

Existem basicamente dois tipos de máquinas de Estado: As Máquinas de Moore e as Máquinas de Mealy. A diferença entre as duas está nas entradas do circuito. Uma máquina de Moore depende somente do estado atual enquanto que uma máquina de Mealy depende do estado atual e da entrada recebida. A Fig. 68 mostra o funcionamento de uma máquina de Moore e a Fig. 69, de uma máquina de Mealy. Ambas podem ser representadas por meio de Diagrama de Estados.

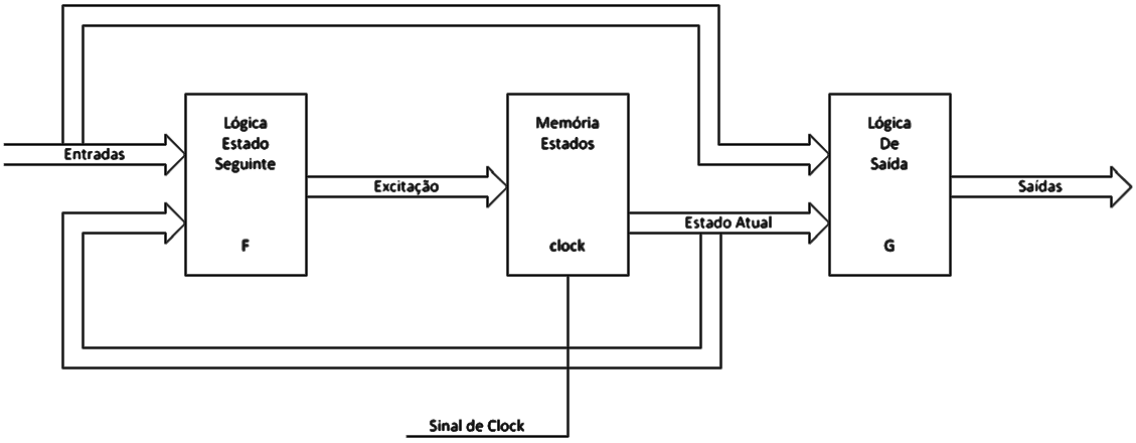


Figura 68 - Funcionamento da Máquina de Moore

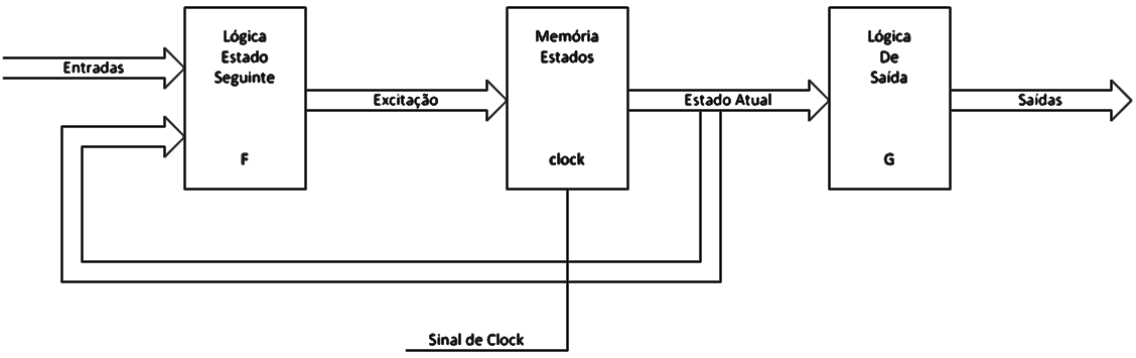


Figura 69 - Funcionamento da Máquina de Mealy

- ### Diagramas de estado

É uma forma prática e intuitiva de representar as máquinas de estado. As máquinas de Moore podem ser representadas com círculos que indicam o estado atual e a sua saída correspondente. Entre os círculos têm-se setas que mostram o estado anterior e os próximos. Cada flecha é uma transição de estado. Ver Fig. 70.

Os diagramas de estados das máquinas de Mealy são similares. O que diverge é que a saída está na seta, ou seja, enquanto a saída de uma máquina de Moore muda somente ao chegar no estado seguinte, na máquina de Mealy é alterada antes da transição (a partir do *clock*). Ver Fig. 71.

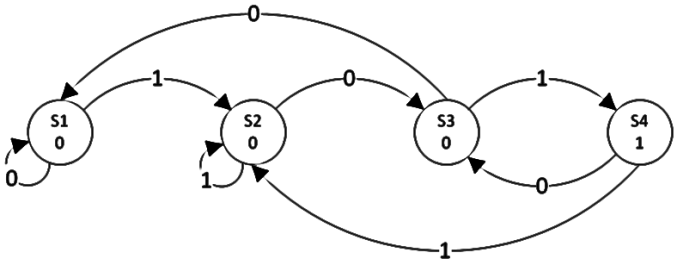


Figura 70 - Máquina de Moore.

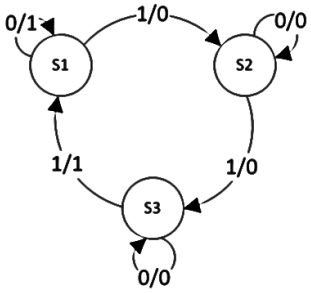


Figura 71 - Máquina de Moore.

- ## Projeto de Máquina de Estado

Etapas:

1. Desenhar o diagrama de estados de acordo com as especificações do projeto.
2. Atribuir códigos binários a cada estado do diagrama.
3. Preencher a tabela de estados.
4. Escolher o Flip-Flop a ser utilizado na implementação do circuito.
5. Obter as equações de entrada.
6. Obter as equações das saída.
7. Fazer o esquemático do circuito.

Exemplo:

Deseja-se projetar uma máquina de estado cuja saída seja 0 até que as últimas três entradas sejam **110**. Após isso a máquina deverá entrar em *loop* com saída igual a 1.

1. Desenhar o diagrama de estados de acordo com as especificações do projeto.

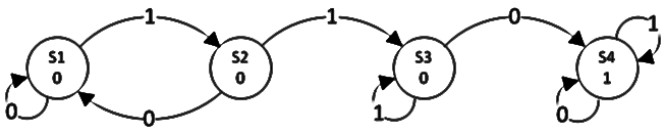


Figura 72 - Diagrama de estados do circuito.

2. Atribuir códigos binários a cada estado do diagrama.

Estado	Codificação
S0	00
S1	01
S2	10
S3	11

Figura 73 - codificação dos estados.

3. Preencher a tabela de estados.

Estado Atual		Entradas	Próximo Estado	
S_1	S_0	X	S_1'	S_0'
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

Figura 74 - Tabela de Estados.

Estado Atual		Saída
S_1	S_0	Y
0	0	0
0	1	0
1	0	0
1	1	1

Figura 75 - Tabela de Saídas.

4. Escolher o Flip-Flop a ser utilizado na implementação do circuito.

Utilizaremos o Flip-Flop tipo D.

5. Obter as equações de entrada.

$$D_1 = S_1 + S_0.X$$

$$D_2 = S_1.S_0 + S_1.X + S_1.\overline{S_0}.X$$

6. Obter as equações de saída.

$$Y = S_1.S_0$$

7. Fazer o esquemático do circuito.

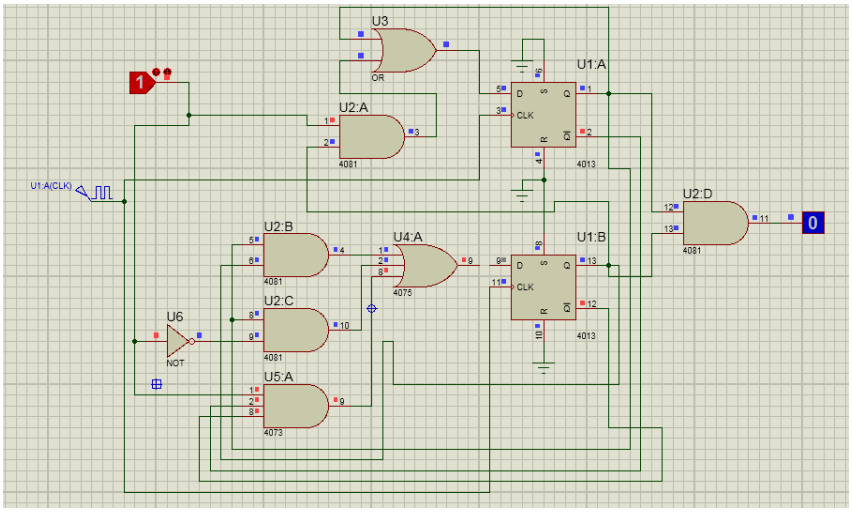


Figura 76 - Esquemático do Projeto

Acho que dava pra fazer um exemplo com Máquina de Mealy e usando Flip-FlopJK, pra poder saber quando usar um e quando usar outro. Já que nesse exemplo ensina a fazer com Moore e Flip-Flop D. Ah, e tipo, explicar, tipo um resumo, quando que se usa Moore e quando que se usa Mealy. E quando se usa Flip-Flop tipo JK ou tipo D.