UFSJ
UNIVERSIDADE FEDERAL
DE SÃO JOÃO DEL-REI

Arquitetura de Computadores

Parte 1 - Cap. 1 a 3

Prof. Erivelton Geraldo Nepomuceno

Departamento de Engenharia Elétrica
Universidade Federal de São João del-Rei

20 de fevereiro de 2018

Abstração Digital

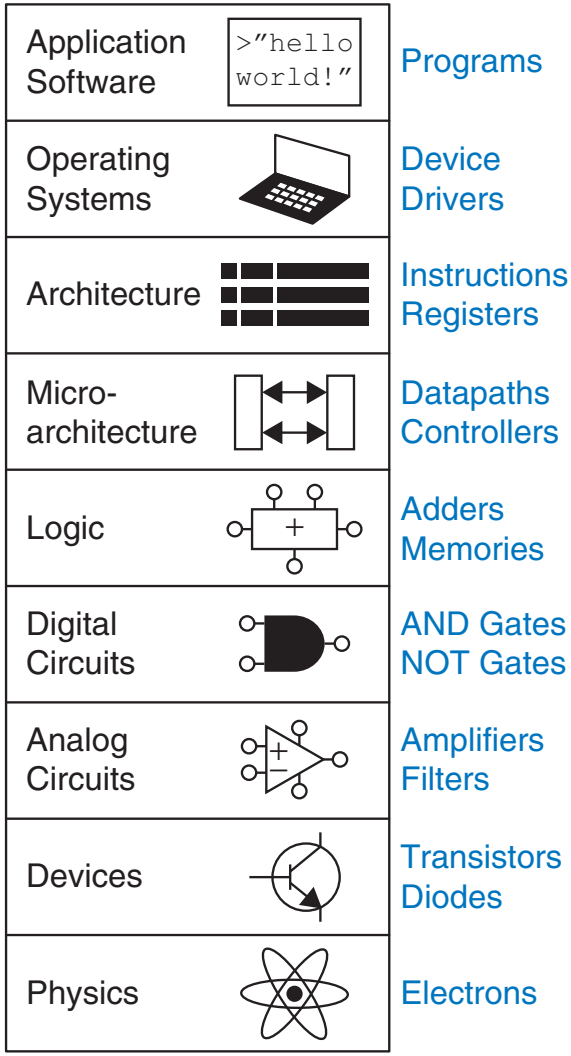


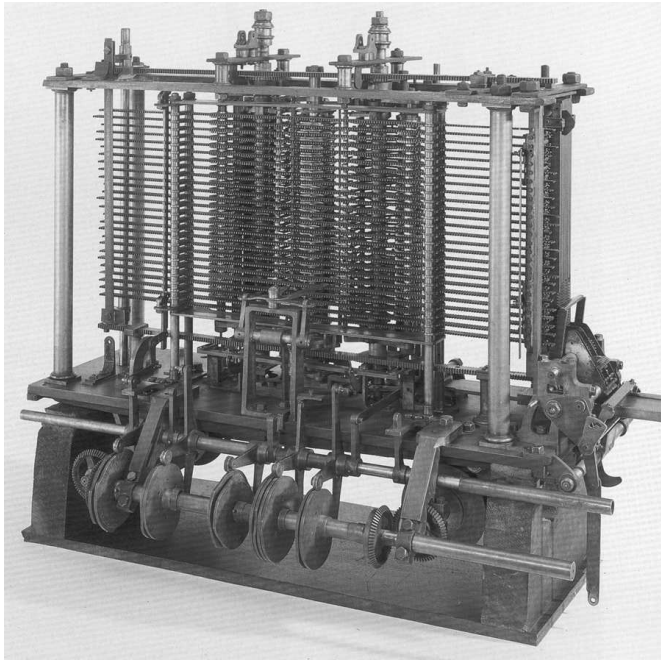
Figura 1: Níveis de abstração para um sistema computacional eletrônico.

Abstração Digital

- A maioria das variáveis físicas são contínuas:
  - A tensão de um fio;
  - A frequência de oscilação;
  - A posição de uma massa.
- A abstração digital considera um subconjunto de valores discretos.

Máquina Analítica de Babbage

- Desenvolvida por Charles Babbage;
- Considerada o primeiro computador digital;
- A Máquina Analítica utilizava engrenagens com dez posições marcadas de 0 a 9.



Máquina Analítica de Babbage (Imagem cortesia do Science Museum/Science and Society PictureLibrary).

Abstração Digital

- Quantidade de informação D numa variável discreta avaliada com n estados distintos.

$$D = \log_2 N \quad \textit{bits}. \tag{1}$$

Equivalência entre bases.

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

Sistema Numérico

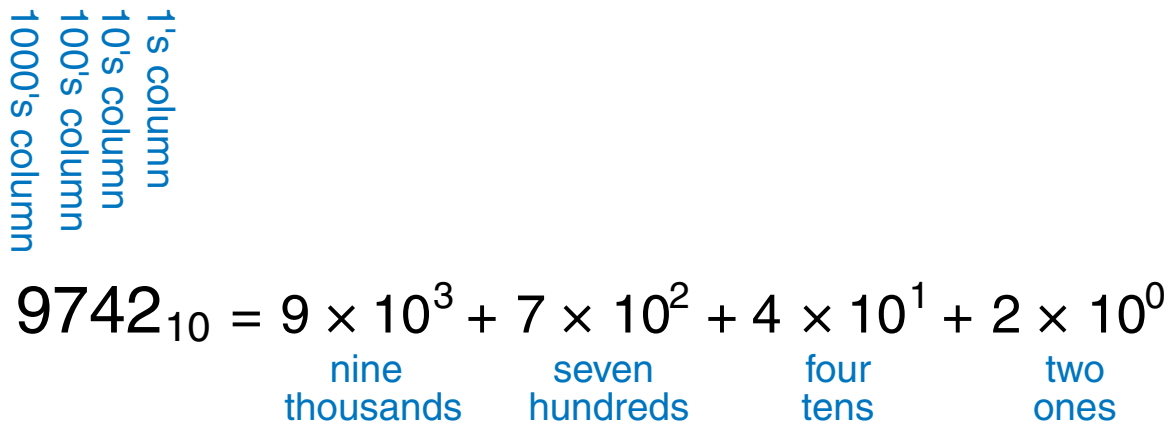


Figura 2: Base Decimal.

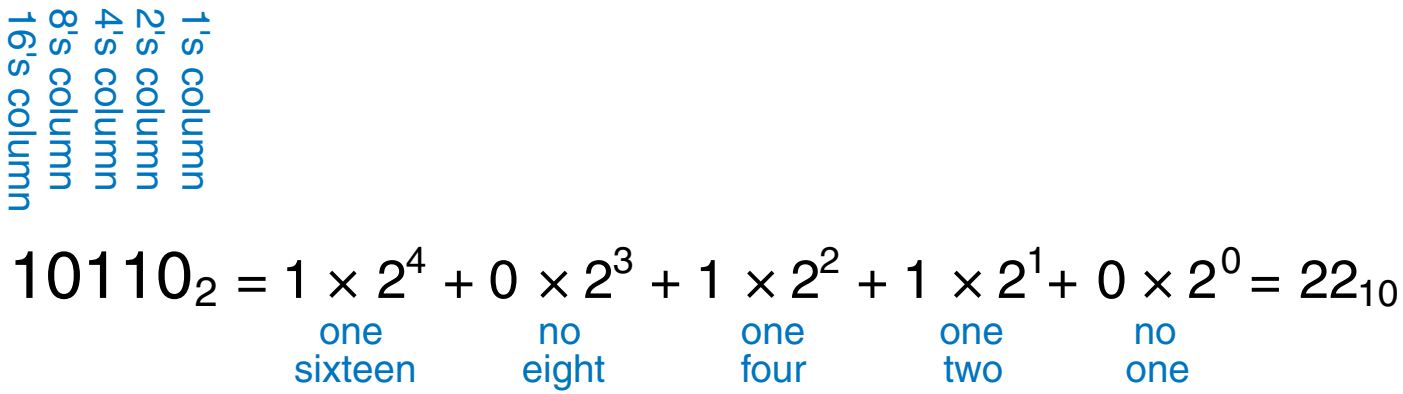


Figura 3: Base Binária.

Adição

Decimal

```
   11  ← carries
   3734
+  5168
------
   8902
```

Binário

```
   11  ← carries
   1011
+  0011
------
   1110
```

Adição Binária

Adicione os seguintes números binários de 4 bits

$$\begin{array}{r} \ 1 \\ \ 1001 \\ +\ 0101 \\ \hline \ 1110 \end{array}$$

Overflow!

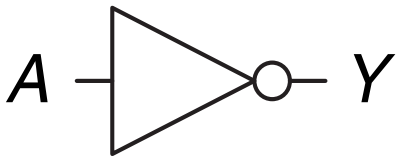
$$\begin{array}{r} \ 111 \\ \ 1011 \\ +\ 0110 \\ \hline 10001 \end{array}$$

Portas Lógicas

- Portas Lógicas: circuitos digitais simples que possuem uma ou mais entradas digitais e produzem uma saída binária.
- Equação Booleana: expressão matemática que usa variáveis binárias;
- Tabela verdade: lista as entradas à esquerda e a saída correspondente à direita, possuem uma linha para cada combinação possível de entradas.

Portas Lógicas

NOT

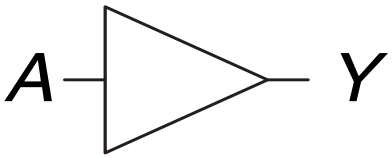


$Y = \overline{A}$

A	Y
0	1
1	0

(a) Porta NOT

BUF

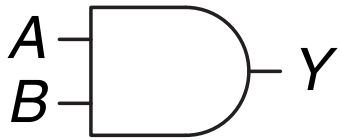


$Y = A$

A	Y
0	0
1	1

(b) *Buffer*

AND

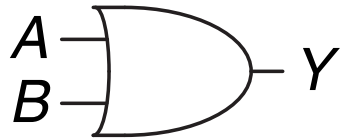


$Y = AB$

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

(c) AND

OR



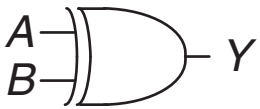
$Y = A + B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

(d) OR

Portas Lógicas

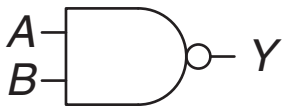
XOR



$Y = A \oplus B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

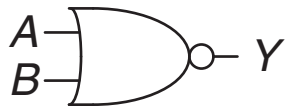
NAND



$Y = \overline{AB}$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR

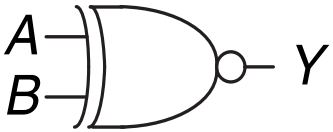


$Y = \overline{A + B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Figura 4: Mais portas de duas entradas.

XNOR



$$Y = \overline{A \oplus B}$$

A	B	Y
0	0	
0	1	
1	0	
1	1	

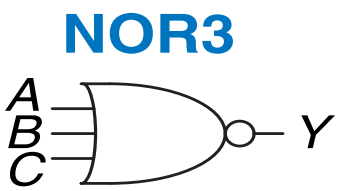
(a)

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

(b)

Figura 5: Porta XNOR.

Portas Lógicas



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

A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

(a)

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

(b)

Figura 6: Porta NOR três entradas.

Transistores CMOS

- Portas lógicas construídas a partir de transistores.
- Interruptor com controle de tensão de 3 portas.
  - Duas portas conectadas dependendo da tensão de uma terceira.
  - d e s estão conectados (ON) quando g é 1.

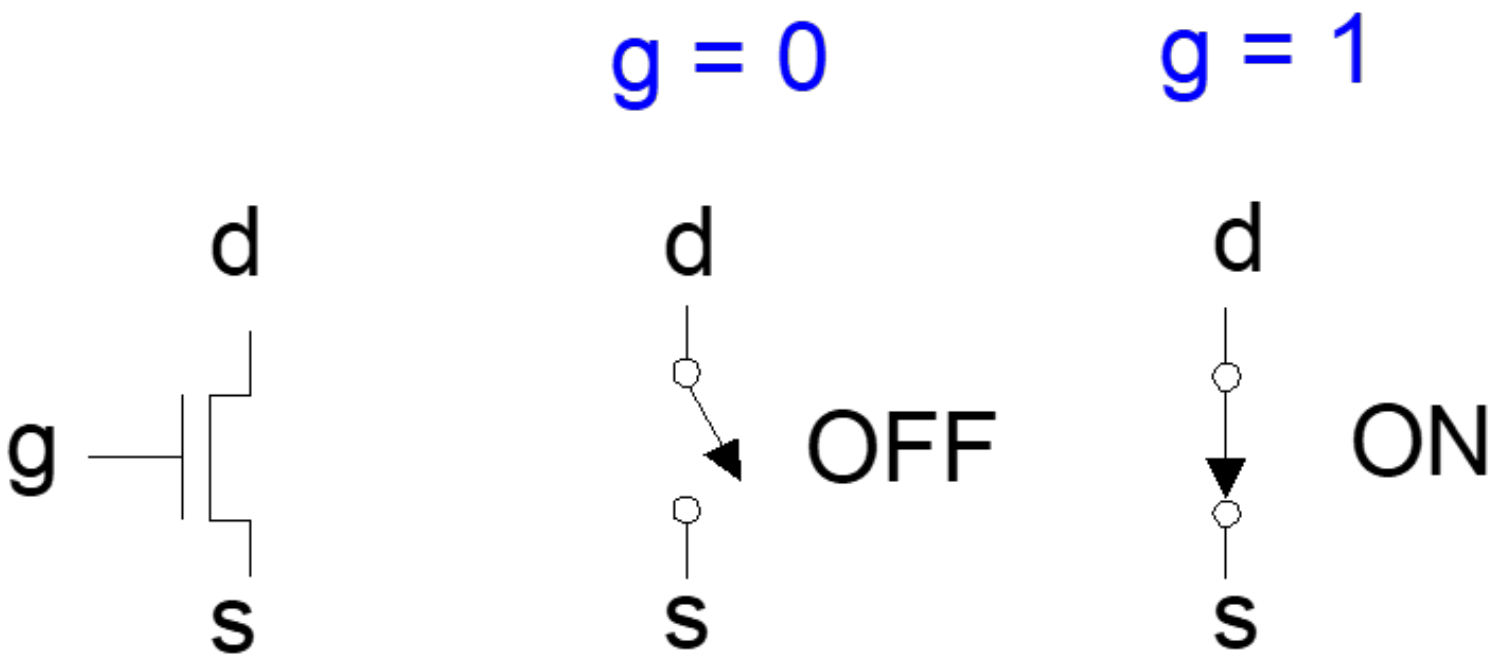


Figura 7: transistores.

Semicondutores

- Os transistores MOS são construídos a partir de silício.
- O silício é um mau condutor porque todos os elétrons estão amarrados em ligações covalentes.
- O silício puro é um condutor pobre.
- Torna-se um condutor melhor quando pequenas quantidades de impurezas, chamadas átomos dopantes, são cuidadosamente adicionadas.
  - Tipo-n: lacunas negativas livres (elétrons).
  - Tipo-p: lacunas positivas livres.

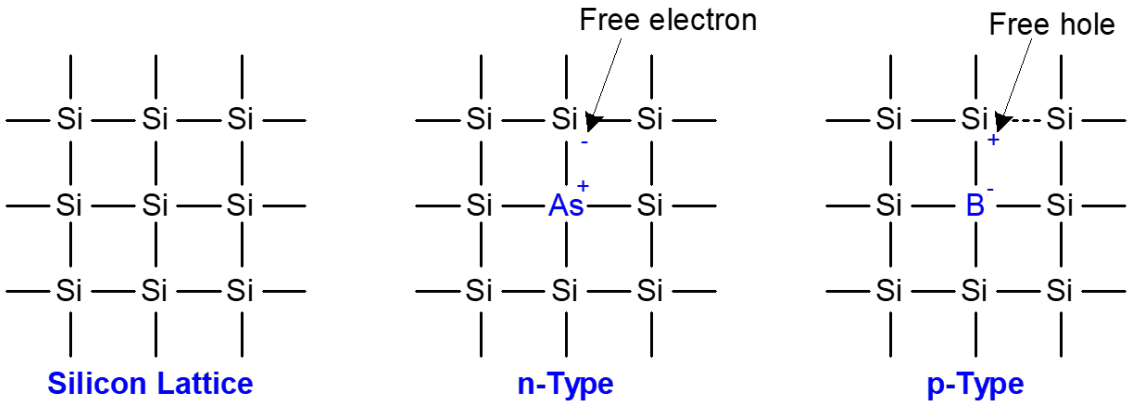


Figura 8: Estrutura cristalina do Silício de átomos dopantes.

Transistores MOS

- Transistores de óxido metálico de silício (MOS).

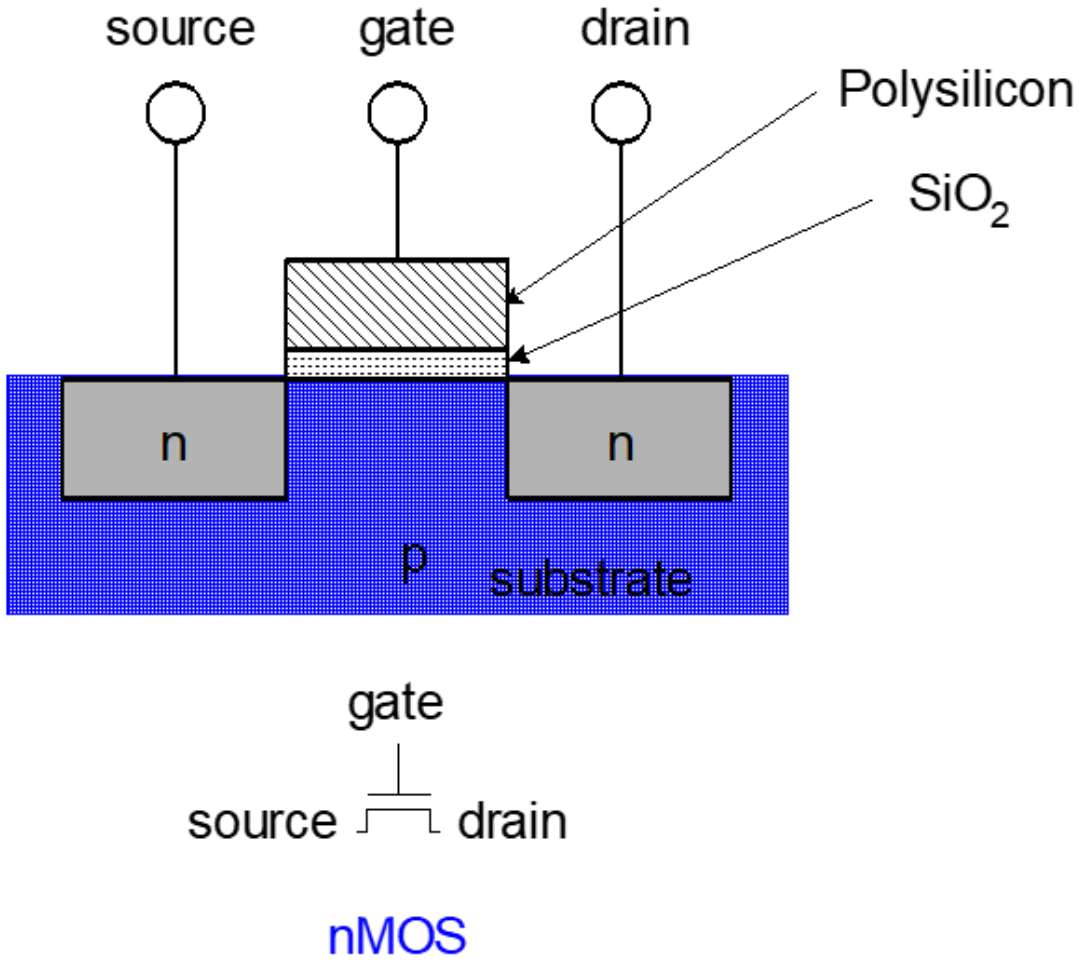


Figura 9: Transistores.

Transistores: nMOS

GATE = 0
OFF: sem conexão entre a fonte e o dreno.

GATE = 1
ON : canal entre fonte e dreno.

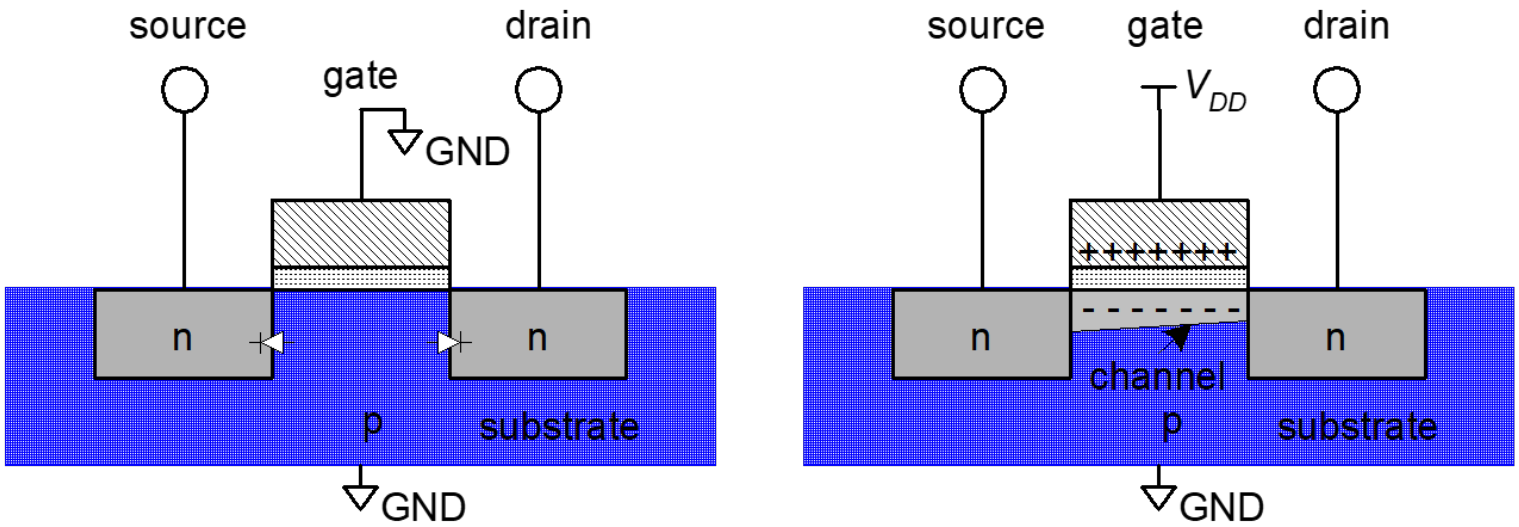


Figura 10: transistores nMOS.

Transistores: pMOS

- pMOS é o oposto do nMOS.
  - ON quando o Gate = 0.
  - OFF quando o Gate = 1.

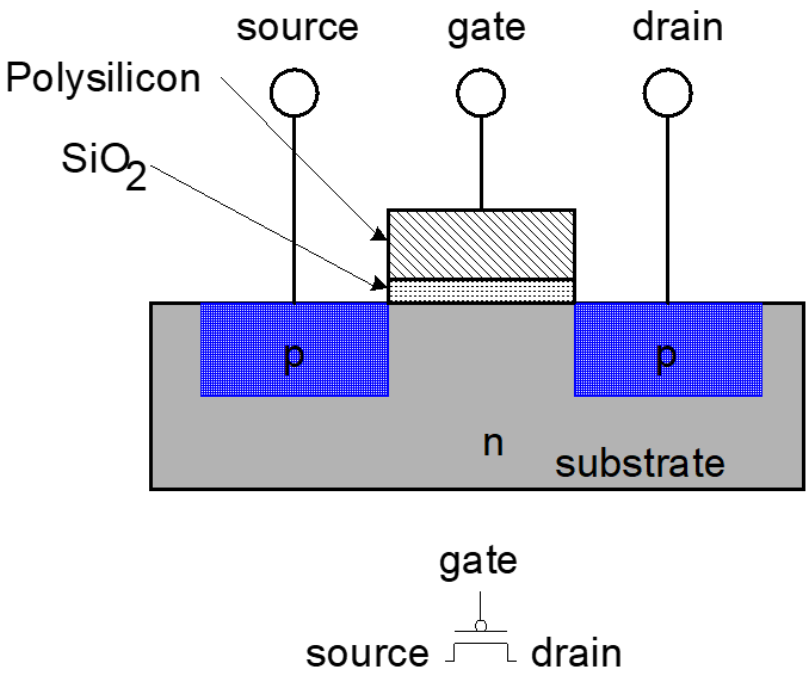


Figura 11: transistores pMOS.

Função do Transistor

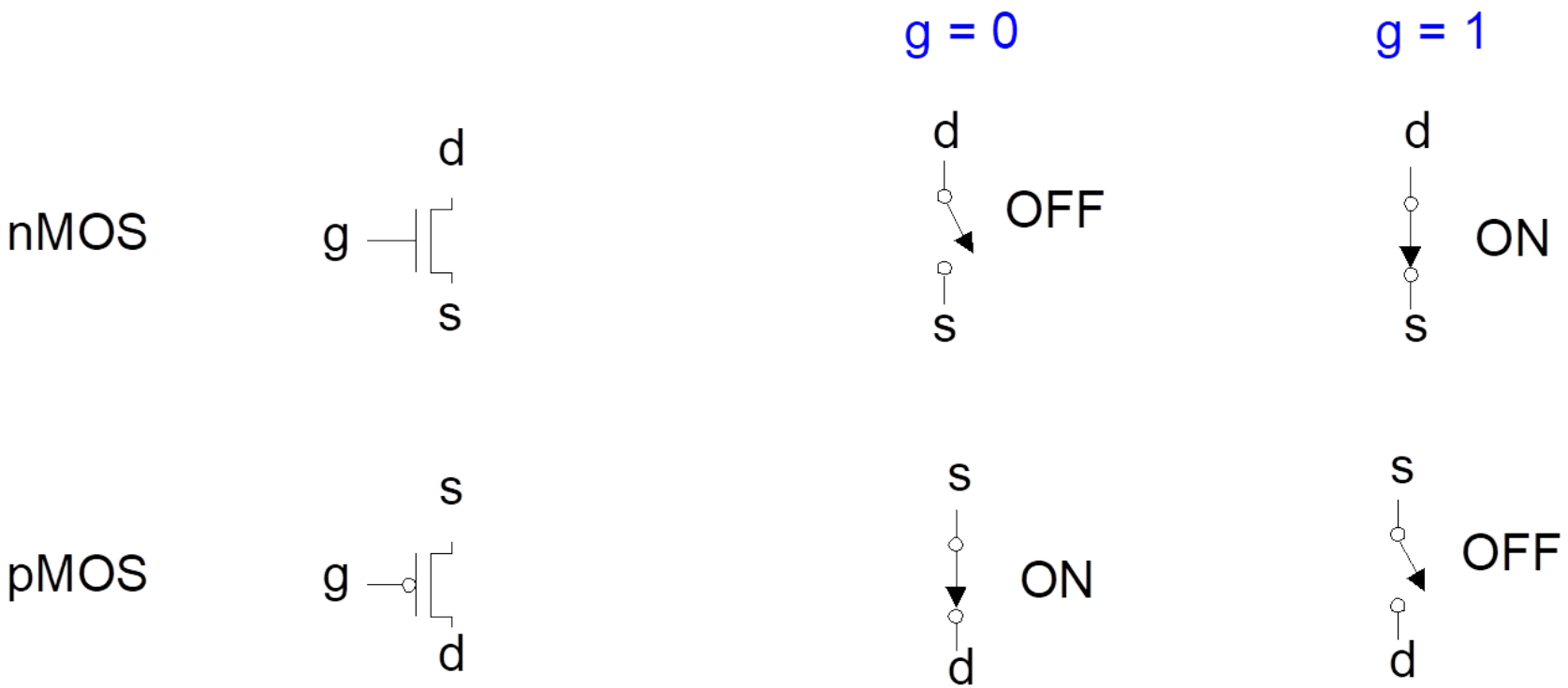
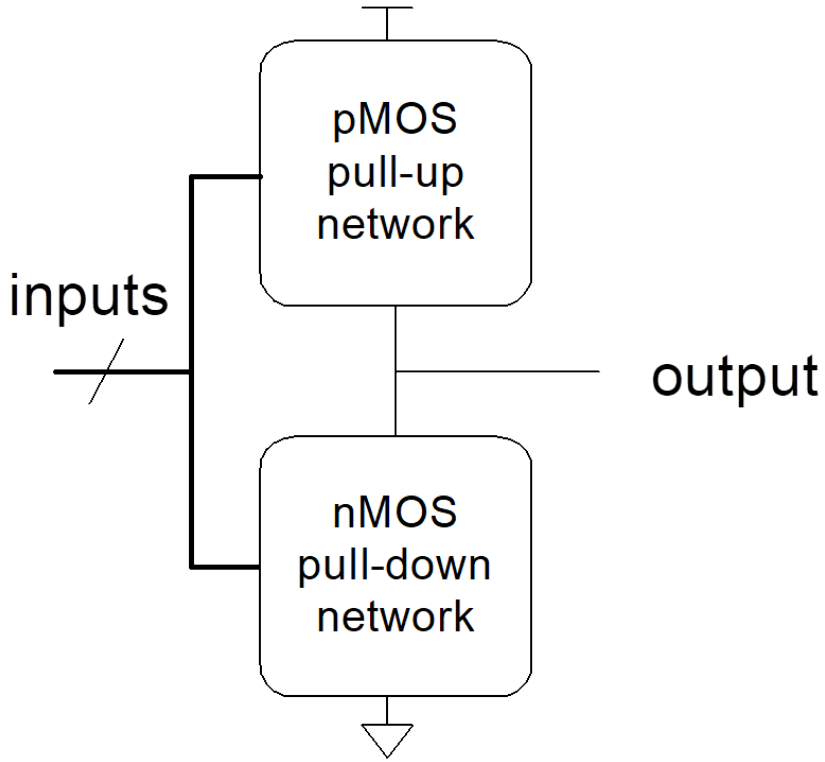


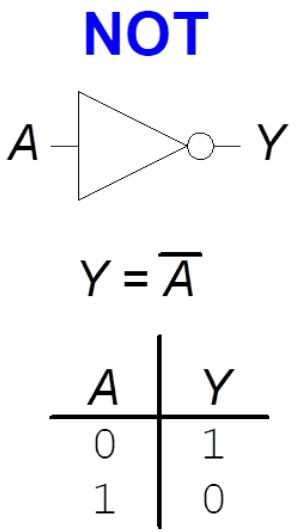
Figura 12: Modelo de interruptor dos MOSFET.

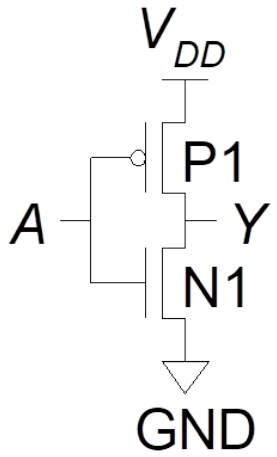
Função do transistor

- **nMOS** permitem bons 0, por isso uma rede *pull-down* de transistores nMOS é colocada entre a saída e GND para puxar a baixo a saída para 0;
- **pMOS** permitem bons 1, então uma rede de *pull-up* de transistores pMOS é colocada entre a saída e V_{DD} para puxar a saída até 1.



Operação de uma porta NOT

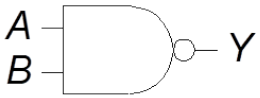




A	P1	N1	*Y*
0	ON	OFF	1
1	OFF	ON	0

Operação de uma porta NAND

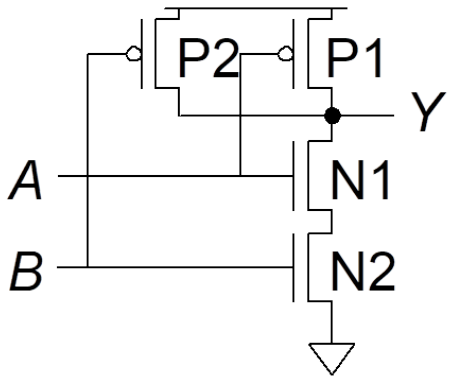
NAND



$Y = \overline{AB}$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

(c)



(d)

A	*B*	P1	P2	N1	N2	*Y*
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

Consumo de Energia

- Quantidade de energia utilizada por unidade de tempo;
- Eletricidade custa dinheiro;
- Sistema vai sobreaquecer se consumir muita energia;
- Sistemas digitais consumem energia dinâmica e estática;
- Fonte de alimentação para carregar um condensador;
- Os sistemas elétricos consomem corrente quando estão ociosos;
- Consumo de energia dinâmica:

$$P_{dinâmica} = \frac{1}{2} C (V_{DD})^2 f \tag{2}$$

Discussão

O rei recebe 64 moedas de ouro em impostos, mas tem razões para acreditar que uma é falsa. Ele convoca-o para identificar a moeda falsa. O leitor tem uma balança que pode conter moedas de cada lado. Quantas vezes o leitor precisa usar a balança para encontrar a moeda falsa mais leve?

Exercícios

1. Quantos números diferentes podem ser representados com 16 bits?
2. Converta os seguintes números binários sem sinal para decimal.
   a. 1010_2
   b. 110110_2
   c. 11110000_2
3. Converta os seguintes números hexadecimais para decimal.
   a. $A5_{16}$
   b. $3B_{16}$
   c. $FFFF_{16}$
   d. $D0000000_{16}$

Exercícios

4. Converta os seguintes números hexadecimais para binário.
   a. $A5_{16}$
   b. $3B_{16}$
   c. $FFFF_{16}$
   d. $D0000000_{16}$
5. A memória no microprocessador Pentium II é organizada como uma matriz retangular de bits com 2^8 linhas e 2^9 colunas.
6. Execute as seguintes adições de números binários não assinados. Indicar se a soma a soma transborda ou não num resultado de 8-bits.
   a. $10011001_2 + 01000100_2$
   b. $11010010_2 + 10110110_2$

Exercícios

7. Escreva um programa na sua linguagem favorita (por exemplo, C, Java, Perl) para converter números de binário para decimal. O utilizador deve digitar um número binário sem sinal. O programa deve imprimir o equivalente decimal.
8. Desenhe o símbolo, a equação booleana e tabela verdade para
   a. uma porta OR de três entradas.
   b. uma porta OR exclusivo (XOR) de três entradas portão.
   c. uma porta de quatro entradas XNOR.
9. A porta OR-AND invertida de três entradas (AOI), mostrada na Figura 13, produz uma saída FALSE se C é verdadeiro e A ou B são TRUE. Caso contrário, produz uma saída TRUE. Produza uma tabela verdade para esta porta.

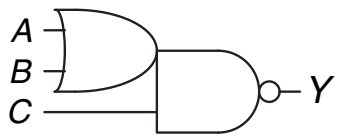


Figura 13: Porta AND-OR_NOT de 3 entradas.

Projeto de Lógica Combinatória

Um circuito lógico pode ser visto como uma caixa preta, e pode ser exemplificado como mostrado na Figura 14.

- Entradas (inputs);
- Saídas (outputs);
- Especificações da função (functional spec);
- Especificação de tempo (timing spec);

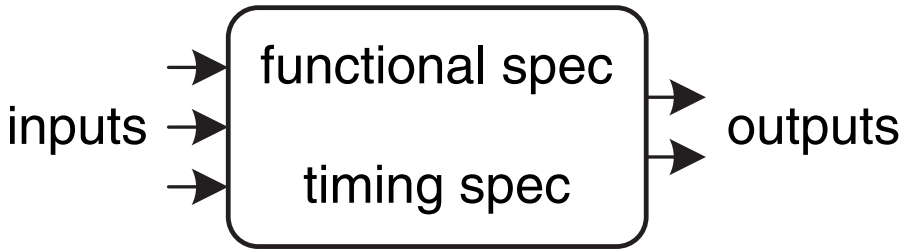


Figura 14: Circuito representado como uma caixa preta com entradas, saídas e especificações.

Projeto de Lógica Combinatória

Os circuitos digitais são classificados como combinatórios ou sequenciais.

- Circuito Combinatório
  - Não tem memória;
  - Saídas são determinadas pelos valores atuais das entradas.
- Circuito Sequencial
  - Tem memória;
  - Saídas são determinadas pelos valores atuais e anteriores das entradas.

- Cada elemento é combinatório;
- O circuito não contém caminhos cíclicos.

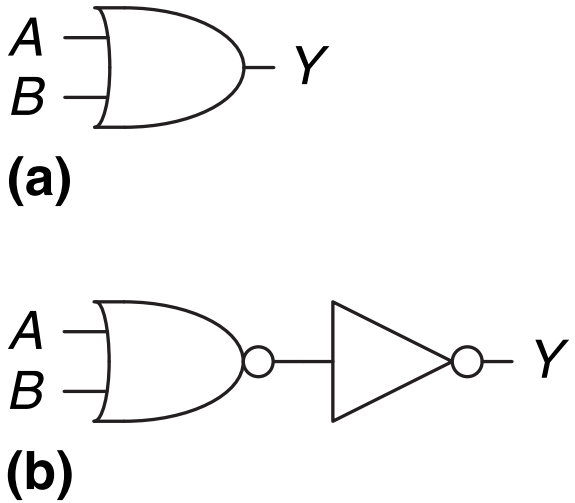


Figura 15: Duas combinações para porta OR.

Terminologia

- Lógica Digital: Variáveis que são TRUE ou FALSE;
- Complemento de uma variável A é seu inverso $\overline{A}$;
- Função lógica AND é chamada de produto;
- Função lógica OR é chamada de soma;
- Uma tabela verdade com N entradas contém 2^N linhas.

Forma de Soma de Produtos

- Todas as equações podem ser escritas em Soma de Produtos;
- O mintermo é um produto (AND) de variáveis;
- Cada mintermo é TRUE para somente uma linha;
- Soma (OR) de produtos (AND termos);

A	B	Y	minterm	minterm name
0	0	0	$\overline{A}\ \overline{B}$	m_0
0	1	1	$\overline{A}\ B$	m_1
1	0	0	$A\ \overline{B}$	m_2
1	1	1	$A\ B$	m_3

Figura 16: Tabela verdade com vários mintermos TRUE.

- $Y = F(A,B) = \overline{A}B + AB = \sum(1,3)$.

Forma de Produto de Somas

- Equações booleanas podem ser escritas em produtos de somas;
- Um maxtermo é uma soma (OR) de variáveis;
- Cada maxtermo é FALSE para essa linha (e apenas essa linha);
- Um produto (AND) de somas (OR) termos;

A	B	Y	maxterm	maxterm name
0	0	0	A + B	M_0
0	1	1	A + $\overline{\text{B}}$	M_1
1	0	0	$\overline{\text{A}}$ + B	M_2
1	1	1	$\overline{\text{A}}$ + $\overline{\text{B}}$	M_3

Figura 17: Tabela verdade com vários maxtermos FALSE.

- $Y = F(A,B) = (A+B)(A+\overline{B}) = \prod(0,2).$

Álgebra Booleana

- A álgebra Booleana utiliza axiomas e teoremas para simplificar as equações Booleanas;
- Semelhante à álgebra convencional, mais simples, já que as variáveis apresentam somente dois valores (1 ou 0).

Axiomas Booleanos

Tabela 1: Axiomas da álgebra Booleana

	Axioma		Dual	Nome
A1	$B = 0$ se $B \neq 1$	A1'	$B = 1$ se $B \neq 0$	Campo Binário
A2	$\overline{0} = 1$	A2'	$\overline{1} = 0$	NOT
A3	$0 \bullet 0 = 0$	A3'	$1 + 1 = 1$	AND/OR
A4	$1 \bullet 1 = 1$	A4'	$0 + 0 = 0$	AND/OR
A5	$0 \bullet 1 = 1 \bullet 0 = 0$	A5'	$1 + 0 = 0 + 1 = 1$	AND/OR

Teoremas Booleanos

Tabela 2: Teoremas Booleanos de uma variável.

	Teorema		Dual	Nome
T1	$B \bullet 1 = B$	T1'	$B + 0 = B$	Identidade
T2	$B \bullet 0 = 0$	T2'	$B + 1 = 1$	Elemento Nulo
T3	$B \bullet B = B$	T3'	$B + B = B$	Idempotência
T4		$\overline{\overline{B}} = B$		Involução
T5	$B \bullet \overline{B} = O$	T5'	$B + \overline{B} = 1$	Complemento

Teorema da Identidade

- $B \bullet 1 = B$;
- $B + 0 = B$.

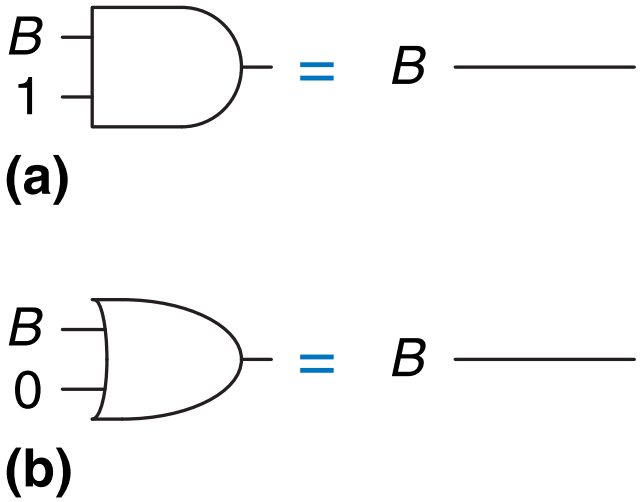


Figura 18: Teorema da identidade em hardware.

Teorema do Elemento Nulo

- $B \bullet 0 = 0$;
- $B + 1 = 1$.

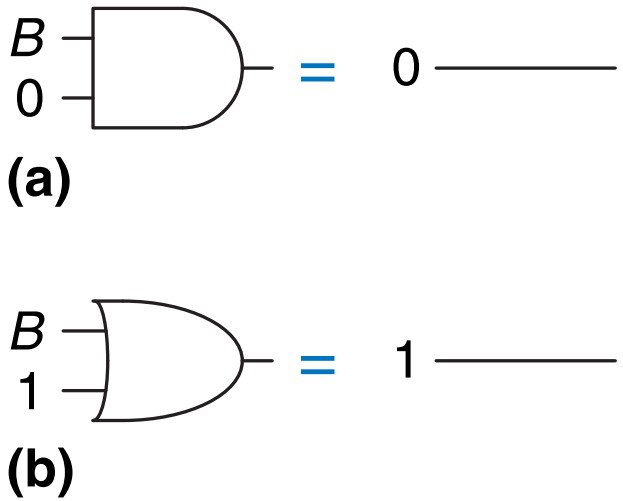


Figura 19: Teorema do elemento nulo em hardware.

Teorema da Idempotência

- $B \bullet B = B$;
- $B + B = B$.

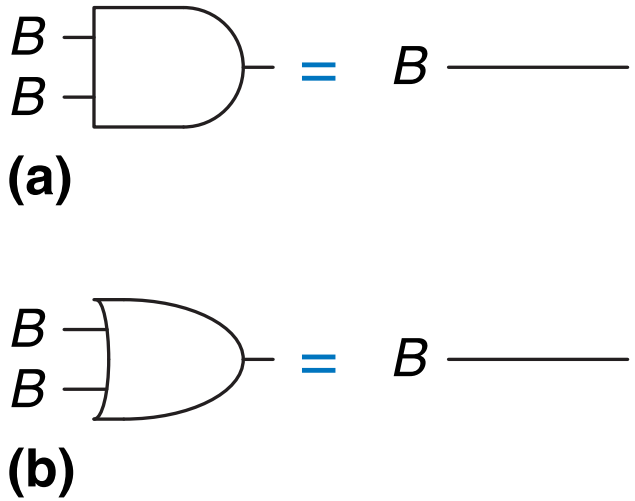


Figura 20: Teorema da idempotência em hardware.

Teorema da Involução

- $\overline{\overline{B}} = B$.

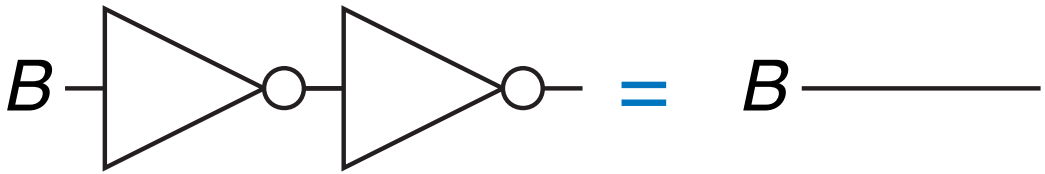


Figura 21: Teorema da involução.

Teorema do Complemento

- $B \bullet \overline{B} = 0$;
- $B + \overline{B} = 1$

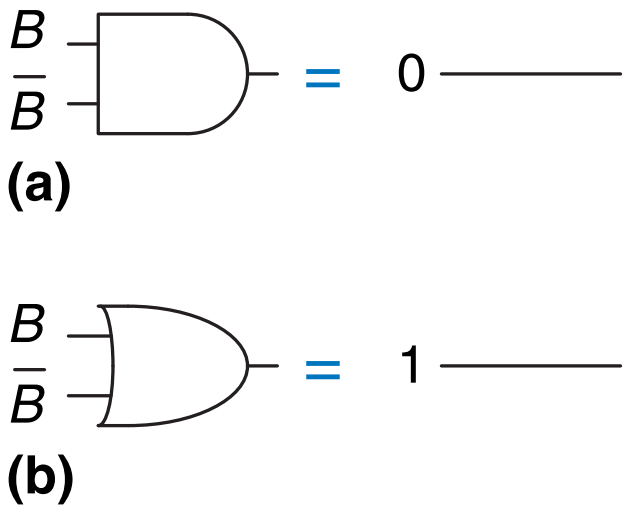


Figura 22: Teorema do complemento em hardware.

Teoremas Booleanos para muitas variáveis

Tabela 3: Teoremas Booleanos para muitas variáveis.

	Teorema		Dual	Nome
T6	$B \bullet C = C \bullet B$	T6'	$B + C = C + B$	Comutatividade
T7	$(B \bullet C) \bullet D = B \bullet (C \bullet D)$	T7'	$(B + C) + D = B + (C + D)$	Associatividade
T8	$(B \bullet C) + (B \bullet D) = B \bullet (C \bullet D)$	T8'	$(B + C) \bullet (B + D) = B + (C \bullet D)$	Ditributividade
T9	$B \bullet (B + C) = B$	T9'	$B + (B \bullet C) = B$	
T10	$(B \bullet C) + (B \bullet \overline{C}) = B$	T10'	$(B + C) \bullet (B + \overline{C}) = B$	Complemento
T11	$(B \bullet C) + (\overline{B} \bullet D) + (C \bullet D) = B \bullet C + \overline{B} \bullet D$	T11'	$(B + C) \bullet (\overline{B} + D) \bullet (C + D) = (B + C) \bullet (\overline{B} + D)$	
T12	$\overline{B_0 \bullet B_1 \bullet B_2 \ldots} = (\overline{B_0} + \overline{B_1} + \overline{B_2} \ldots)$	T12'	$\overline{B_0 + B_1 + B_2 \ldots} = (\overline{B_0} \bullet \overline{B_1} \bullet \overline{B_2} \ldots)$	De Morgan

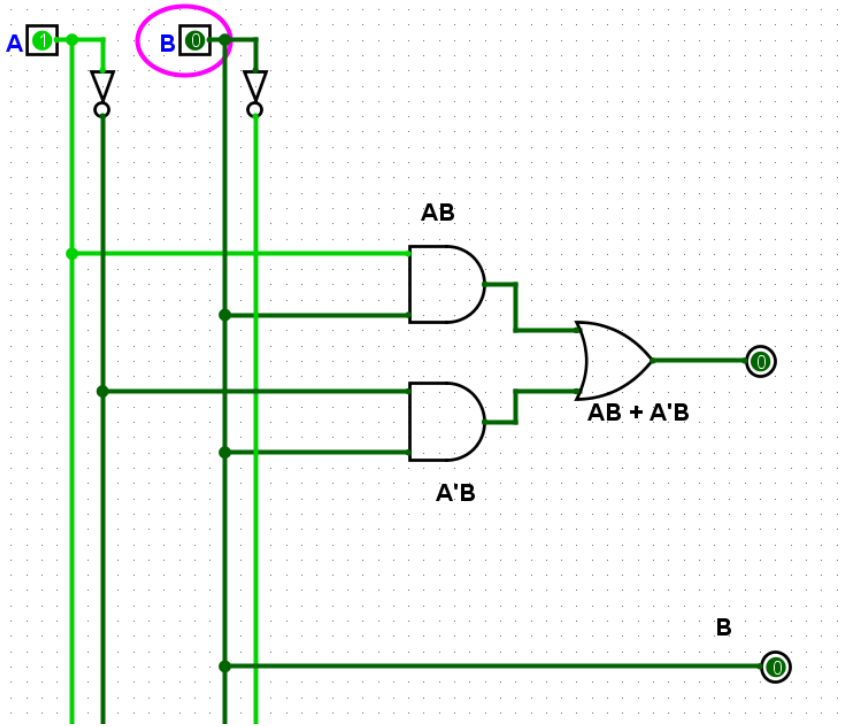
Simplificando Equações Boolenas

Exemplo 1:

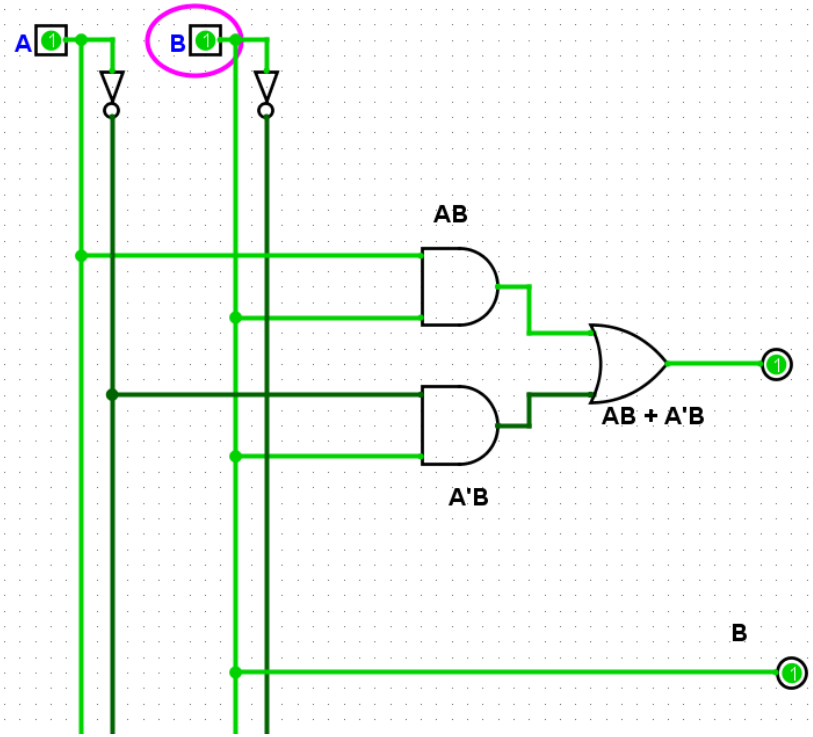
$$
\begin{aligned}
Y &= AB + \overline{A}B \\
&= B(A + \overline{A}) && T8 \\
&= B(1) && T5' \\
&= B && T1
\end{aligned}
$$

Simplificando Equações Boolenas

Exemplo 1:



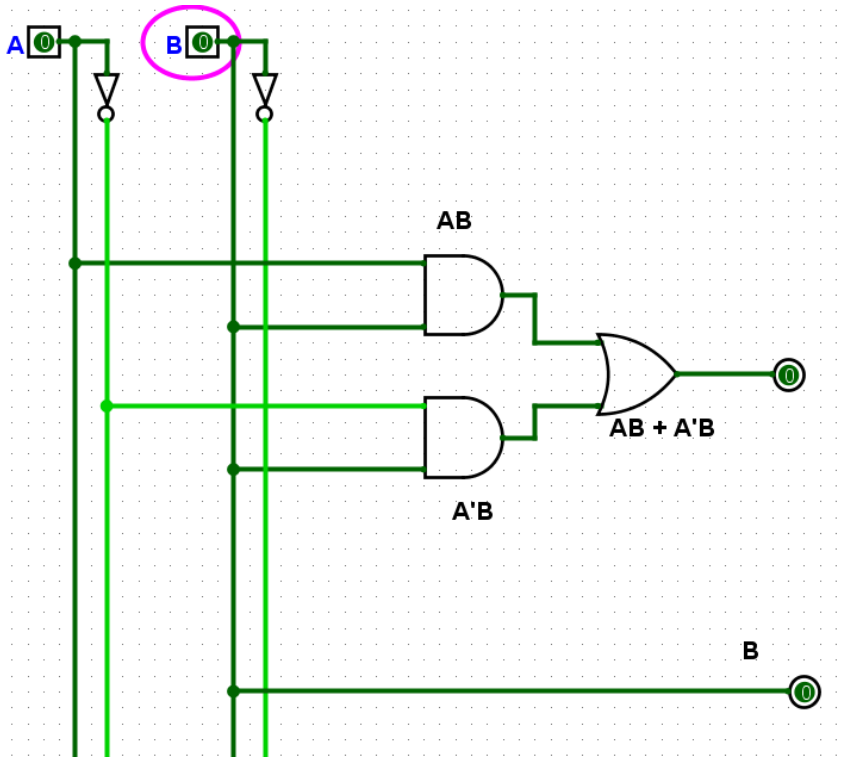
(a) A = 1 e B = 0



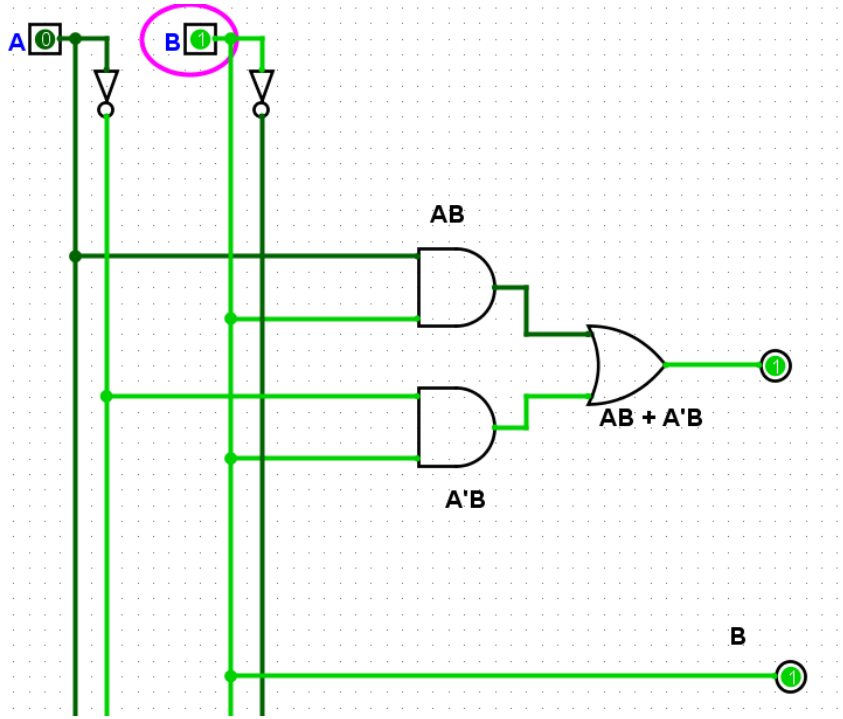
(b) A = 1 e B = 1

Simplificando Equações Boolenas

Exemplo 1:



(a) A = 0 e B = 0



(b) A = 0 e B = 1

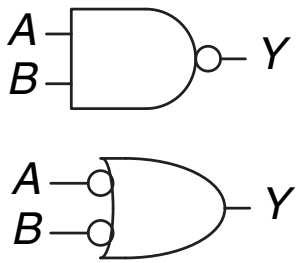
Simplificando Equações Boolenas

Exemplo 2:

$$
\begin{aligned}
Y &= A(AB + ABC) \\
&= A(AB(1 + C)) && T8 \\
&= A(AB(1)) && T2' \\
&= A(AB) && T1 \\
&= (AA)B && T7 \\
&= AB && T3
\end{aligned}
$$

Teorema de De Morgan

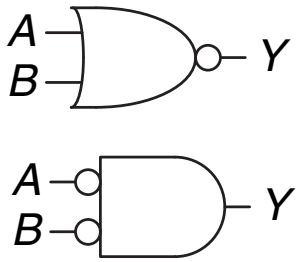
NAND



$$Y = \overline{AB} = \overline{A} + \overline{B}$$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR



$$Y = \overline{A + B} = \overline{A}\,\overline{B}$$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

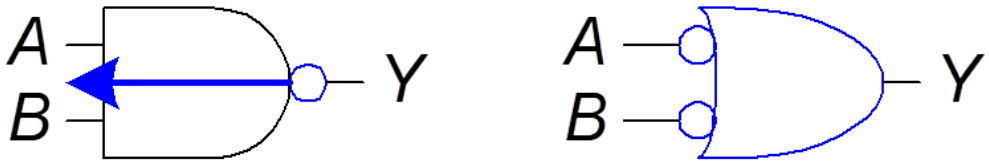
Figura 23: Portas equivalentes de De Morgan.

Lógica Combinatório Multi-Nível

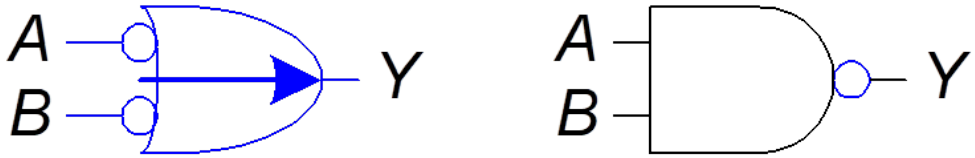
- Algumas funções lógicas necessitam de uma enorme quantidade de hardware quando construídas utilizando-se a lógica de dois níveis.

Empurrões de Bolha

- Backward: Adiciona a bolha para a entrada

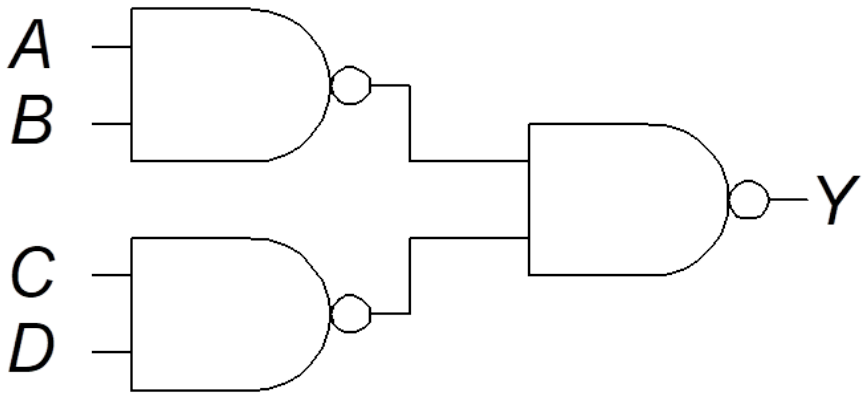


- Forward: Adiciona a bolha para a saída



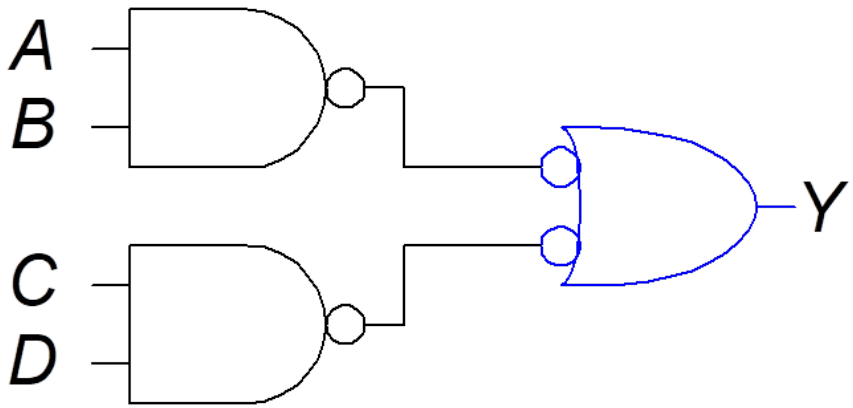
Empurrões de Bolha

- Qual é a expressão booleana para este circuito?



Empurrões de Bolha

- Qual é a expressão booleana para este circuito?



$$Y = AB + CD$$

Empurrões de Bolha

1. Comece nas saídas do circuito e trabalhe em direção às entradas;
2. Empurre qualquer bolha na saída de volta para a entrada;
3. Trabalhando no sentido contrário, desenhe cada porta de tal forma que as bolhas se possam cancelar.

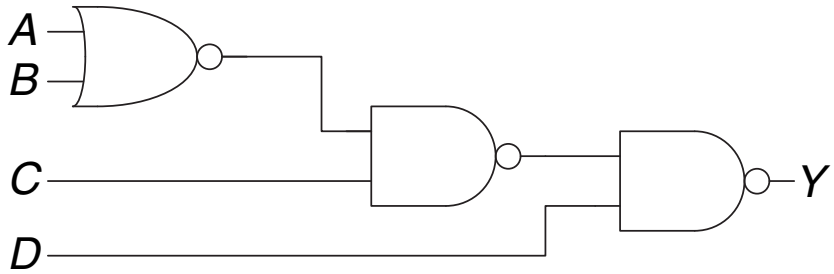


Figura 24: Circuito multi-nível utilizando NAND e NOR.

Empurrões de Bolha

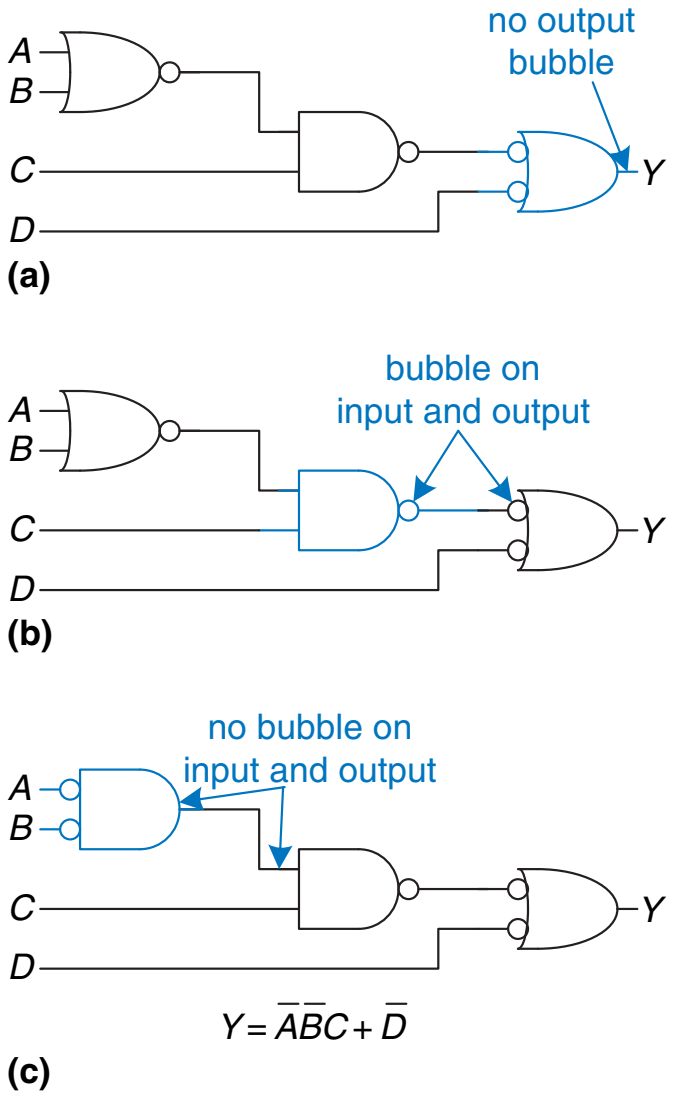


Figura 25: Circuito utilizando empurrões de bolha.

Mapas de Karnaugh

- As expressões podem ser minimizadas combinando termos;
- K-mapas minimizam as equações graficamente;
- $PA + PA = P$;

A	B	C	Y
0	0	0	**1**
0	0	1	**1**
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

(a)

Y \ AB / C	00	01	11	10
0	**1**	0	0	0
1	**1**	0	0	0

(b)

Y \ AB / C	00	01	11	10
0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}B\bar{C}$	$AB\bar{C}$	$A\bar{B}\bar{C}$
1	$\bar{A}\bar{B}C$	$\bar{A}BC$	ABC	$A\bar{B}C$

(c)

Figura 26: Função de três entradas: (a) tabela verdade, (b) K-map, (c) K-map mostrando os mintermos.

Mapas de Karnaugh

- Círculo de 1 em quadrados adjacentes;
- Na expressão booleana, inclua apenas literais cuja forma verdadeira e complementar não esteja no círculo;

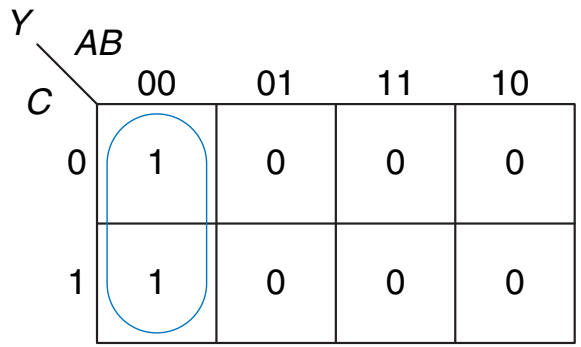


Figura 27: Minimização com mapa de Karnaugh.

3 entradas - Mapas de Karnaugh

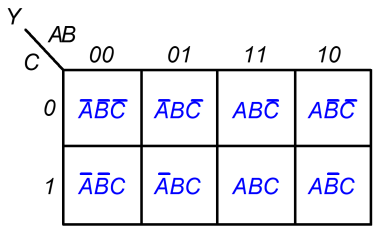


Figura 28: K-map para o exemplo.

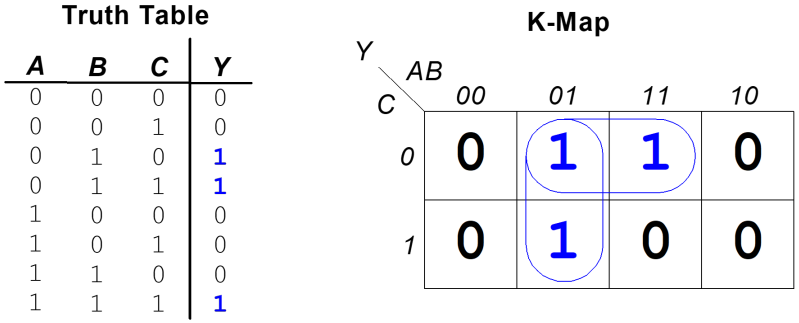


Figura 29: Minimização com mapa de Karnaugh.

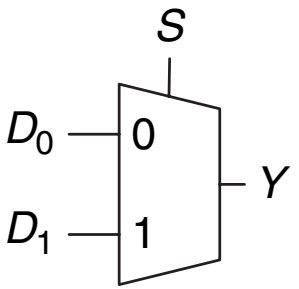
- $Y = \overline{A}B + B\overline{C}$

Blocos Combinatórios

- A lógica combinatória é frequentemente agrupada em blocos maiores a fim de se construírem sistemas mais complexos.
- Multiplexadores;
- Decodificadores.

Blocos Combinatórios

- Multiplexador: escolhe entre as N entradas de dados baseada na entrada de seleção:
- $log_2 N - bit$ seleciona a entrada;



S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Figura 30: Símbolo e tabela verdade de multiplexador 2:1.

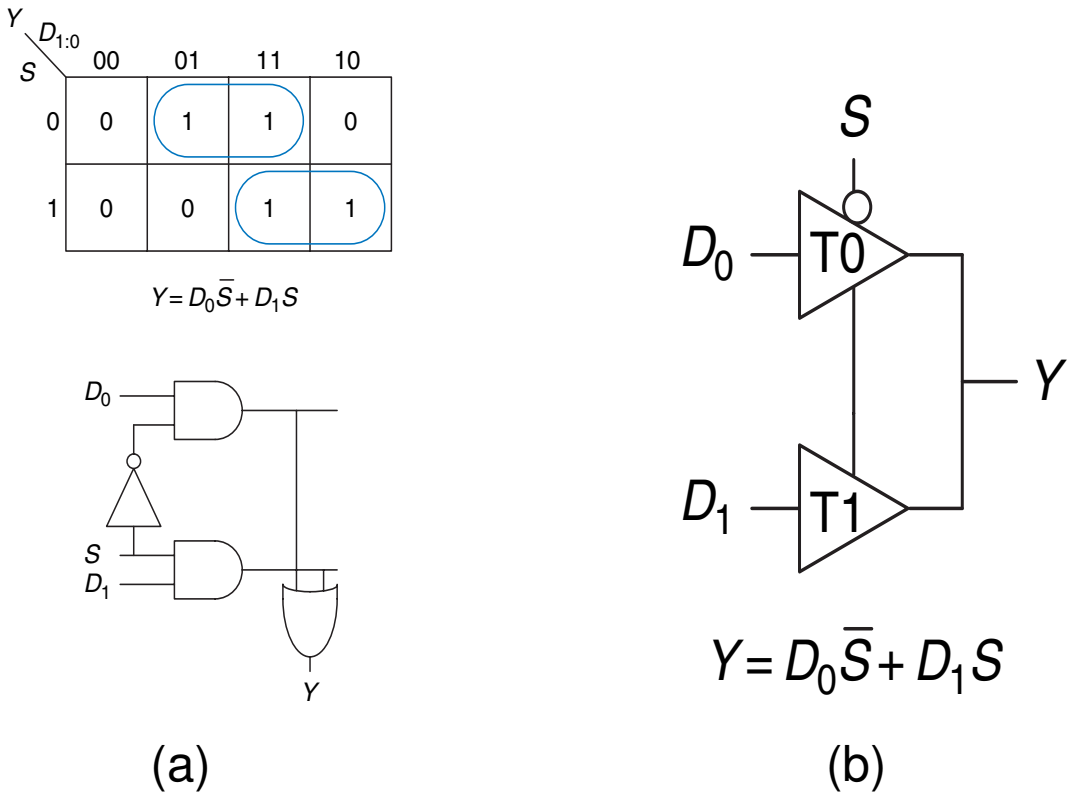


Figura 31: (a) Implementação de um multiplexador usando 2:1 usando lógica de níveis e (b) multiplexador usando *buffers* Tristate.

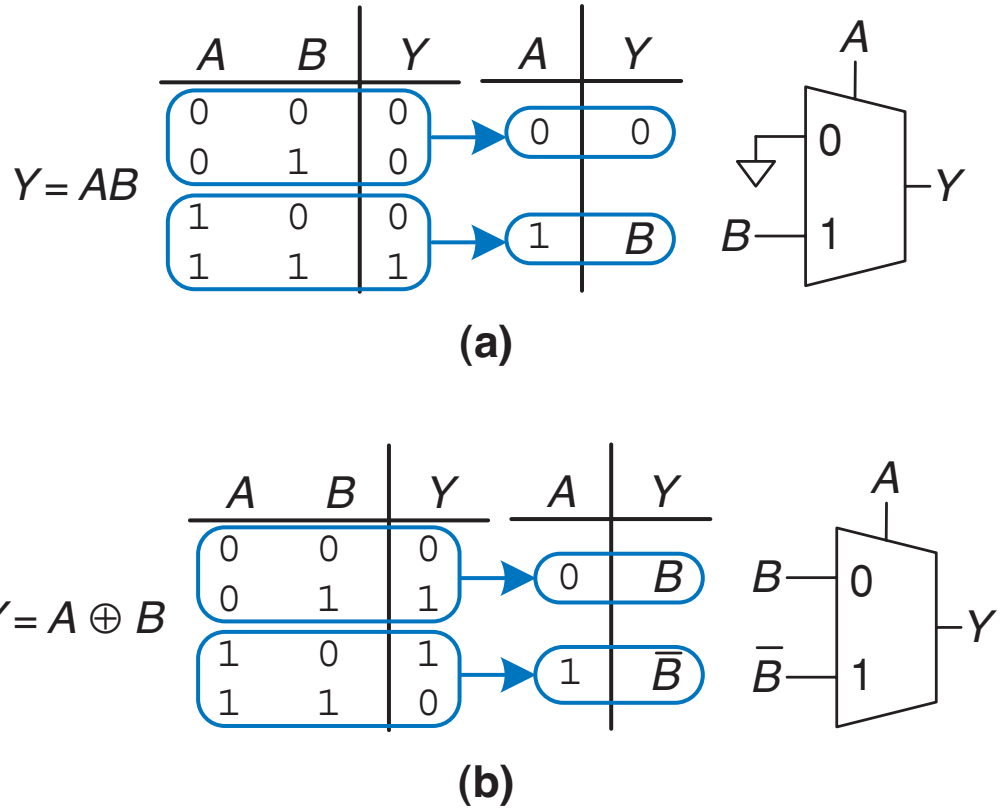
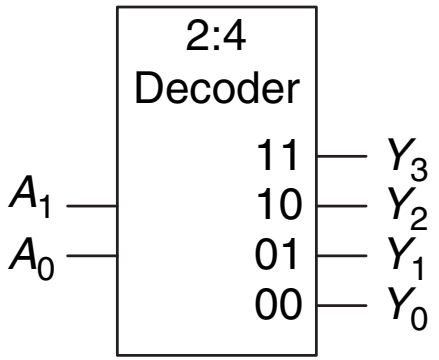


Figura 32: Lógica de multiplexador utilizando variáveis de entrada.

Decodificadores

- Reduzindo o tamanho do multiplexador.
- Possui N entradas e 2^N saídas
- Saídas únicas: apenas uma saída ALTA de uma só vez.



A_1	A_0	Y_3	Y_2	Y_1	Y_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

Figura 33: Decodificador 2:4.-

Lógica utilizando decodificador

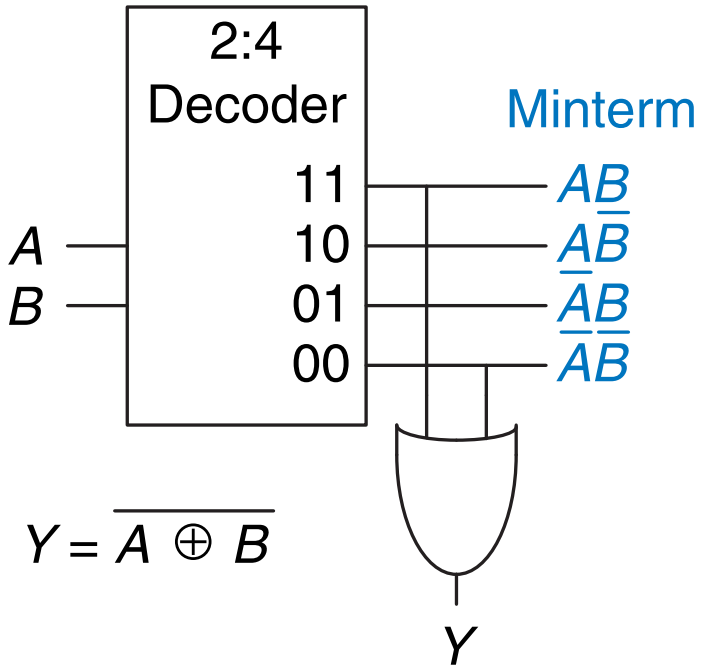


Figura 34: Função lógica utilizando decodificador.

Temporização

- Atraso entre a mudança da entrada e da saída.

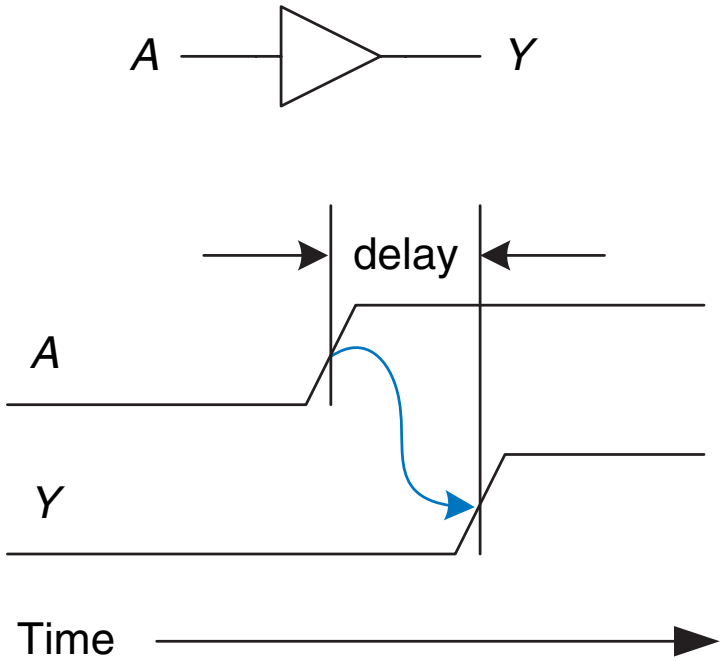


Figura 35: Atraso do Circuito.

Atraso de Propagação e de Contaminação

A lógica combinatória é caracterizada por seu atraso de propagação e de contaminação.

- O atraso de propagação t_{pd} é o tempo máximo a partir do qual a entrada muda de valor, até quando a saída ou as saídas atingem seu valor final.
- O atraso de contaminação t_{cd} é o tempo mínimo a partir do qual a entrada muda de nível, até quando qualquer uma das saídas comece a mudar de valor.

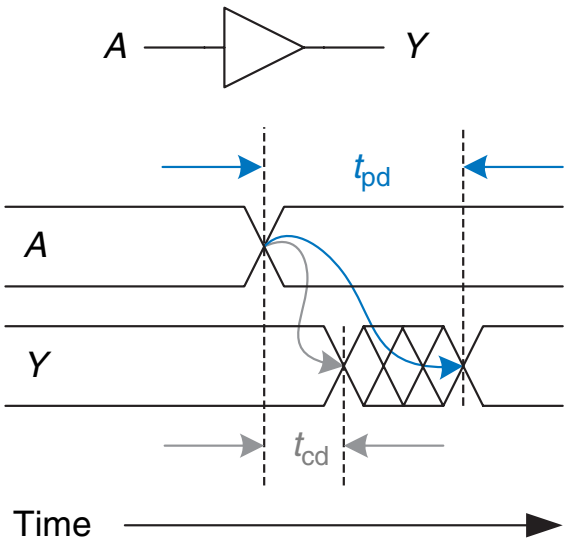


Figura 36: Atraso de Propagação e de Contaminação de um Buffer.

Atraso de Propagação e de Contaminação

O atraso é causado por:

- Tempo necessário para carregar as capacitâncias em um circuito;
- Limitação da velocidade da luz.

t_{pd} e t_{cd} podem ser diferentes devido a muitas razões, incluindo:

- Diferentes atrasos de subida e descida;
- Múltiplas entradas e saídas, algumas das quais são mais rápidas que outras;
- Circuitos diminuem a velocidade quando estão quentes e aumentam quando estão frios.

Atraso de Propagação e de Contaminação

Os atrasos de propagação e de contaminação são também determinados através do caminho que um sinal faz da entrada para a saída.

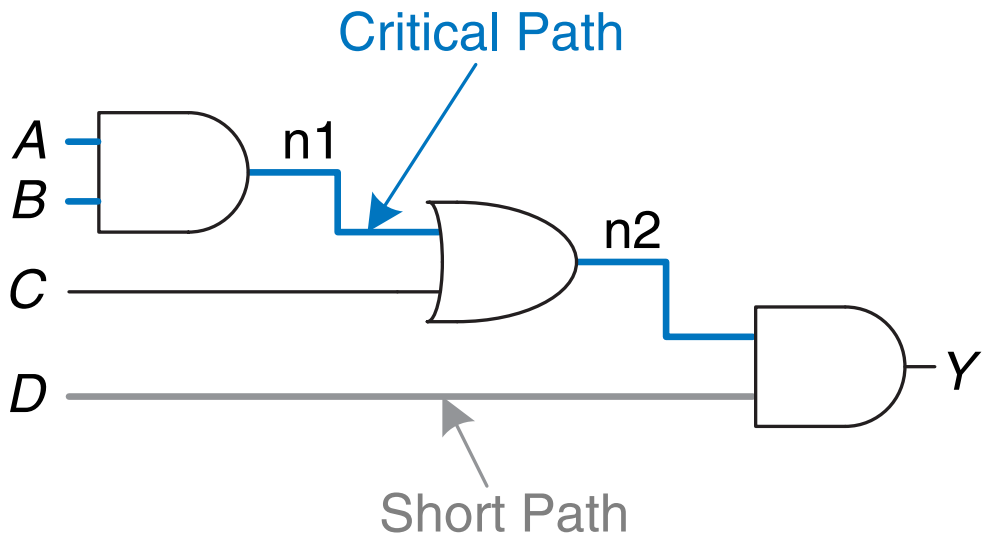


Figura 37: Caminho curto e caminho crítico.

Atraso de Propagação e de Contaminação

O atraso de propagação de um circuito combinatório é igual à soma dos atrasos de propagação através de cada elemento no caminho crítico. E o atraso de contaminação é a soma através de cada elemento no caminho curto.
Para o exemplo apresentado na Figura 37, esses atrasos são representados por:

$$t_{pd} = 2t_{pd_{AND}} + t_{pd_{OR}}$$

$$t_{cd} = t_{cd_{AND}}$$

Atraso de Propagação e de Contaminação

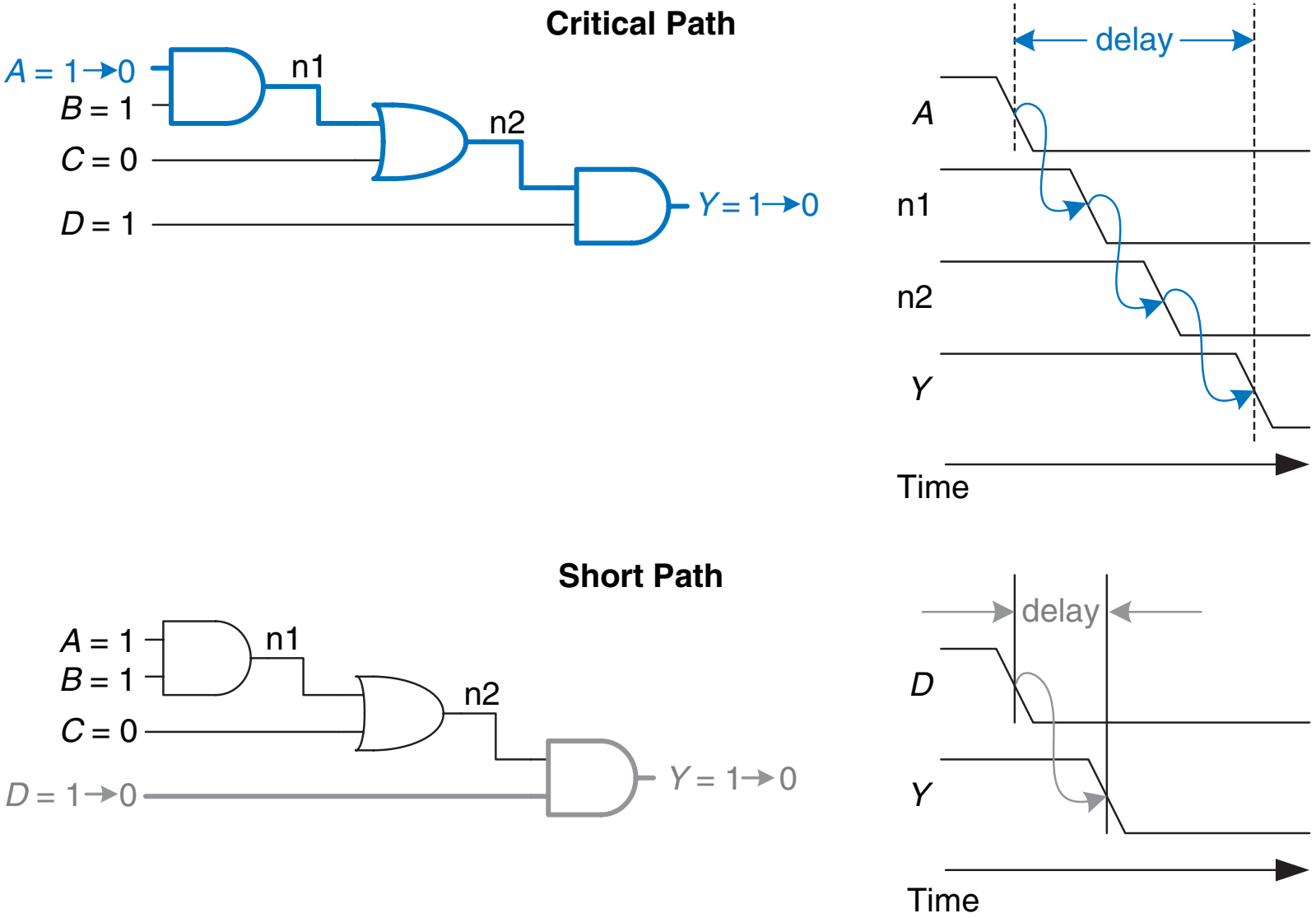
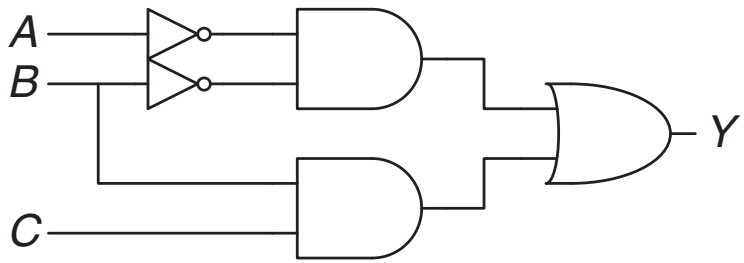


Figura 38: Formas de onda do caminho curto e do caminho crítico.

Glitches

O que acontece quando $A = 0$, $C = 1$ e B transita de 1 para 0.



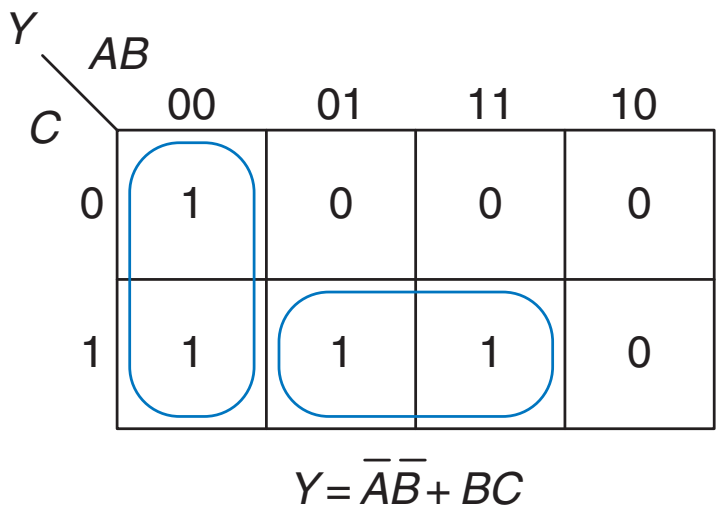


Figura 39: Circuito com glitch.

Glitches

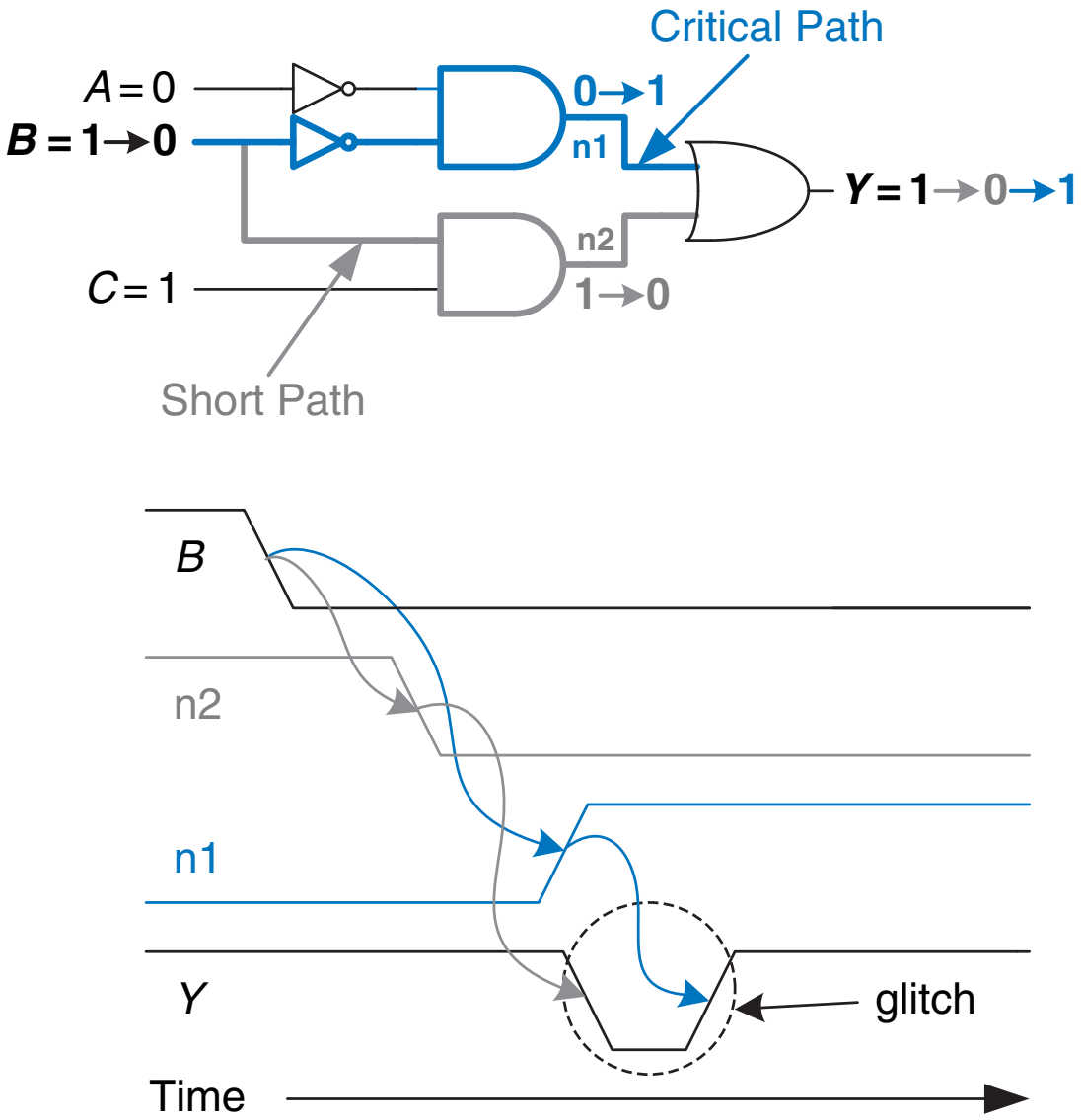
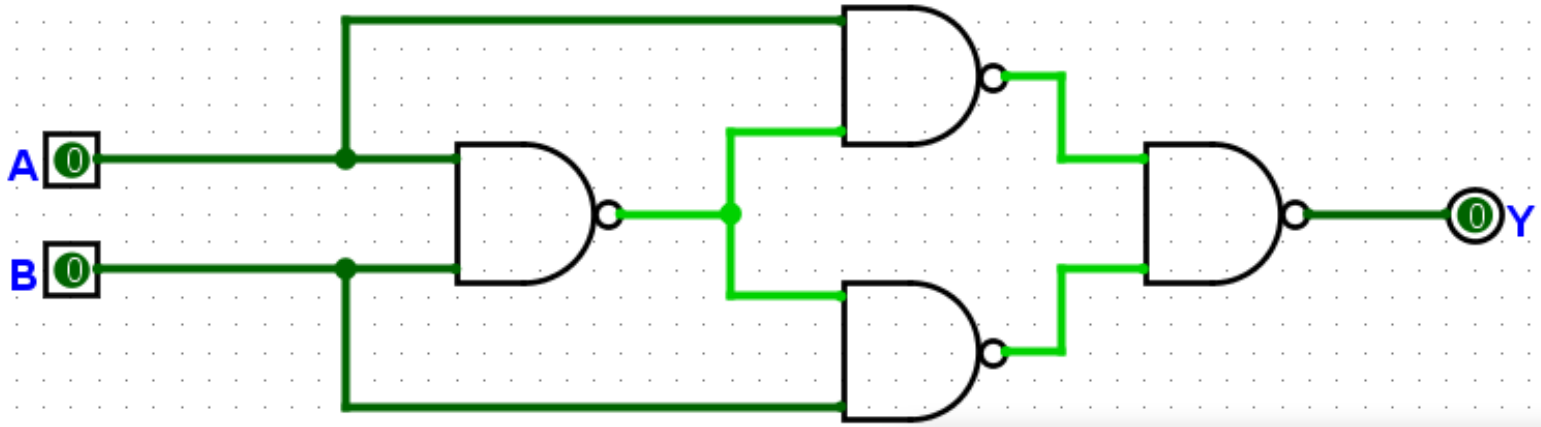


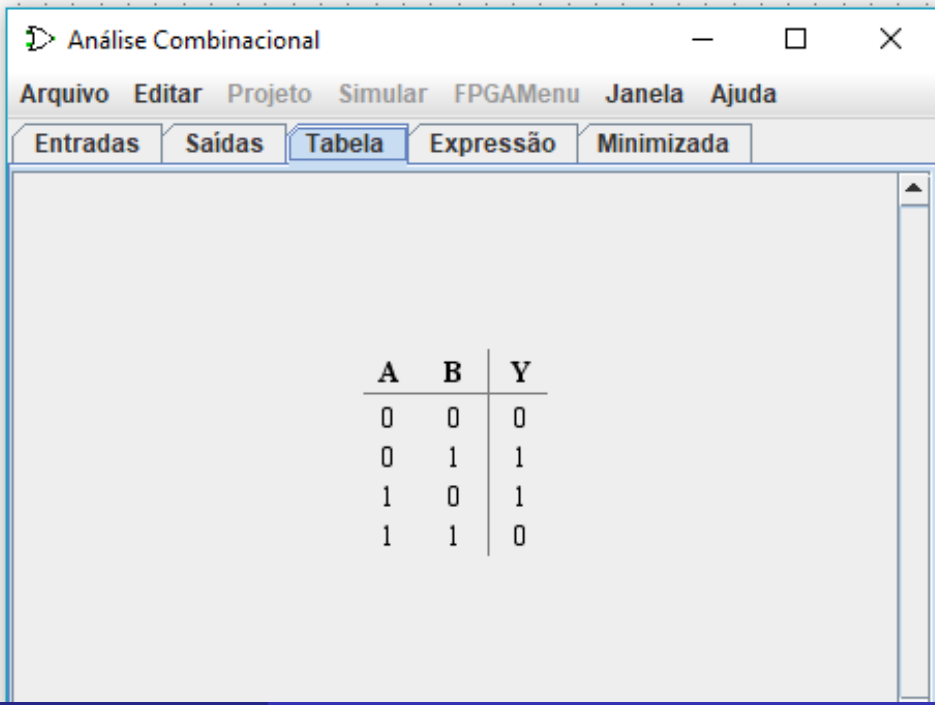
Figura 40: Temporização de um glitch.

Discussão

Esboce um esquema para a função XOR de duas entradas usando apenas portas NAND.

Discussão





A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Exercícios

1. Escreva a equação Booleana na forma canônica de soma de produtos para cada uma das tabelas verdade na Figura 41.

(a)

A	B	Y
0	0	1
0	1	0
1	0	1
1	1	1

(b)

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(c)

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

(d)

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

(e)

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Figura 41: Tabelas verdade.

Exercícios

2. Simplifique as seguintes equações Booleanas utilizando os teoremas Booleanos. Verifique se as respostas estão corretas utilizando uma tabela verdade ou um K-map.
   a. $Y = A\,C + \overline{A}\,\overline{B}\,C$
   b. $Y = \overline{A}\,\overline{B} + \overline{A}\,B\,\overline{C} + \overline{\overline{(A + \overline{C})}}$
   c. $Y = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + A\,\overline{B}\,\overline{C} + A\,\overline{B}\,C\,\overline{D} + A\,B\,D + \overline{A}\,\overline{B}\,C\,\overline{D} + B\,\overline{C}\,D + \overline{A}$

3. Simplifique cada uma das seguintes equações Booleanas. Esboce um circuito combinatório relativamente simples implementando a equação simplificada.
   a. $Y = \overline{B\,C + \overline{A}\,\overline{B}\,\overline{C} + B\,\overline{C}}$
   b. $Y = \overline{A + \overline{A}\,B + \overline{A}\,\overline{B} + A + \overline{B}}$
   c. $Y = A\,B\,C + A\,B\,D + A\,B\,E + A\,C\,D + A\,C\,E + \overline{(A + B + E)} + \overline{B}\,\overline{C}\,D + \overline{B}\,\overline{C}\,E + \overline{B}\,\overline{D}\,\overline{E} + \overline{C}\,\overline{D}\,\overline{E}$

Exercícios

4. Alyssa P. Hacker diz que qualquer função Booleana pode ser escrita na forma mínima de soma de produtos como uma soma de todos os implicantes primos da função. Ben Bitdiddle diz que existem algumas funções para as quais as suas equações mínimas não envolvem todos os implicantes primos. Explique por que Alyssa está certa ou dê um contra-exemplo demostrando o ponto de Ben.
5. Utilizando portas equivalentes de De Morgan e métodos de empurrão de bolha, redesenhe o circuito da Figura 42 a fim de que o leitor possa encontrar a equação Booleana por inspeção. Escreva a equação Booleana.

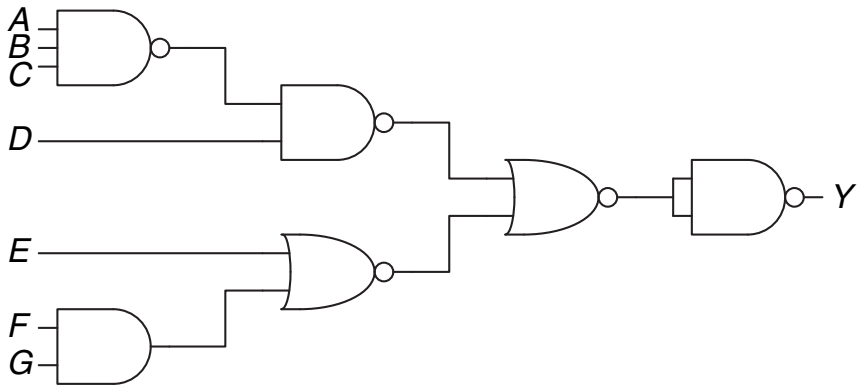


Figura 42: Circuito esquemático.

Exercícios

6. Encontre a equação Booleana mínima para a função na Figura 43.

A	*B*	*C*	*D*	*Y*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	X
0	0	1	1	X
0	1	0	0	0
0	1	0	1	X
0	1	1	0	X
0	1	1	1	X
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	X
1	1	1	1	1

Figura 43: Tabela verdade.

Exercícios

7. Um circuito possui quatro entradas e duas saídas. As entradas $A_{3:0}$ representa um número de 0 a 15. A saída P deve ser TRUE se o número for primo (0 e 1 não são primos, mas 2,3,5 e assim por diante, são). A saída D deve ser TRUE se o número for divisível por 3. Dê equações Booleanas simplificadas para cada saída e esboce o circuito.
8. Escreva a equação Booleana minimizada para a função realizada pelo circuito na Figura 44.

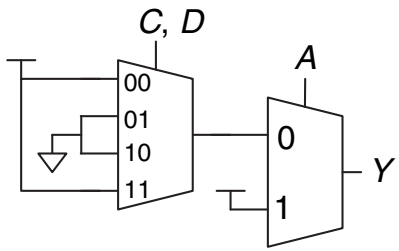
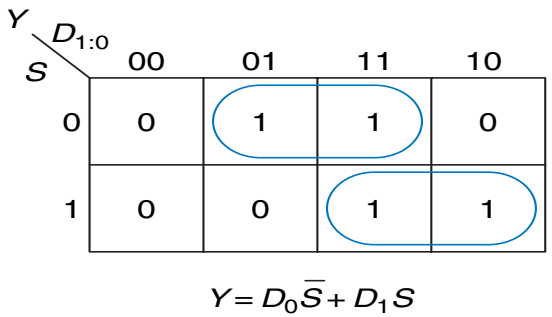


Figura 44: Circuito multiplexador.

Exercícios

9. Determine o atraso de propagação e o atraso de contaminação do circuito na Figura 45. Use os seguintes atrasos de porta.



(a)

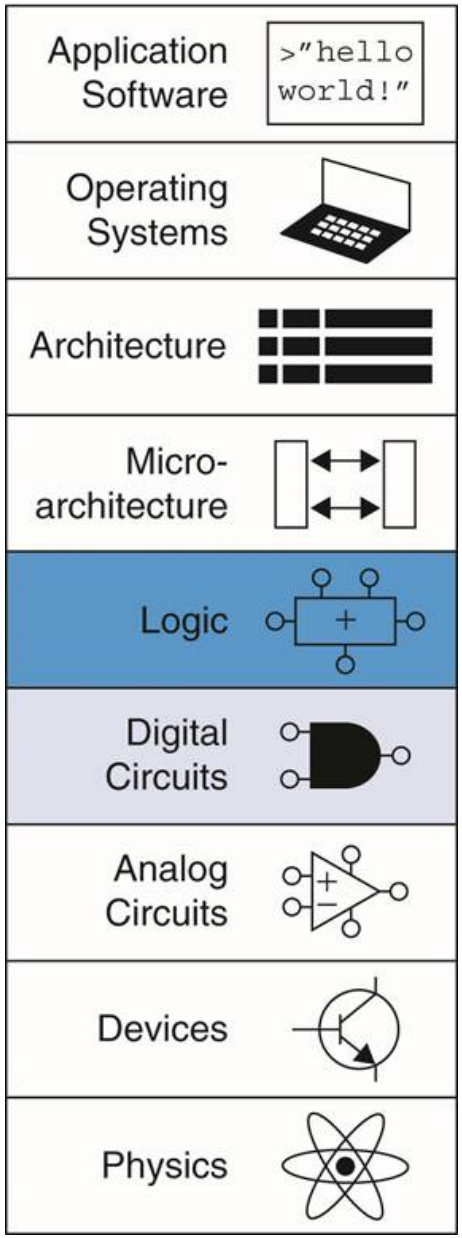
Gate	t_{pd} (ps)	t_{cd} (ps)
NOT	15	10
2-input NAND	20	15
3-input NAND	30	25
2-input NOR	30	25
3-input NOR	45	35
2-input AND	30	25
3-input AND	40	30
2-input OR	40	30
3-input OR	55	45
2-input XOR	60	40

(b)

Figura 45: (a) Circuito esquemático e (b) atrasos de porta.

Capítulo 3: Tópicos

- Introdução;
- Básculas e Flip-Flops;
- Temporização da Lógica Sequencial;
- Paralelismo.



Introdução

- As saídas da lógica sequencial dependem tanto dos atuais valores da entrada como dos anteriores. Assim, a lógica sequencial tem memória.
- Estado: todas as informações sobre o passado necessárias para explicar o comportamento futuro do circuito;
- Básculas e *flip-flops* são circuitos sequenciais simples que armazenam o estado de um bit.
- Circuitos sequenciais síncronos são formados por lógica combinatória e bancos de *flip-flops* que contêm o estado do circuito.

Circuito Biestável

- O bloco de construção fundamental de memória é um elemento biestável, um elemento com dois estados estáveis;
- Duas saídas: Q e $\overline{Q}$;
- Não tem entradas;

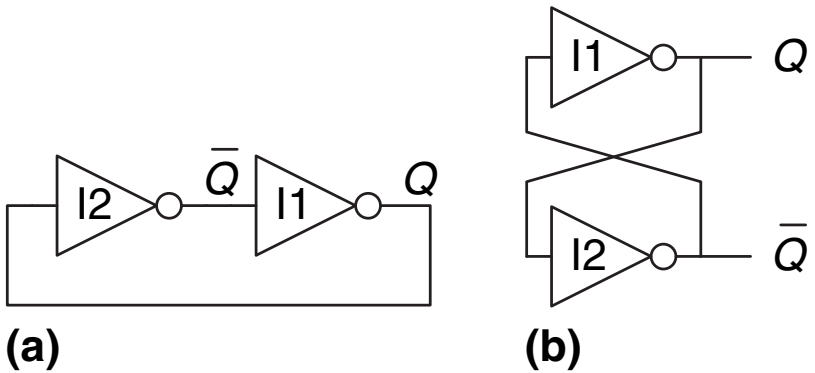
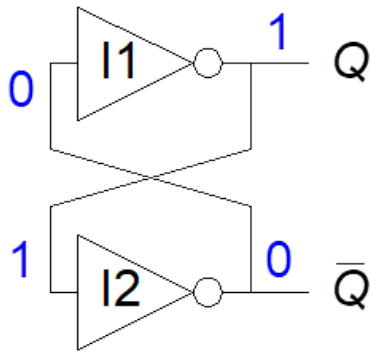


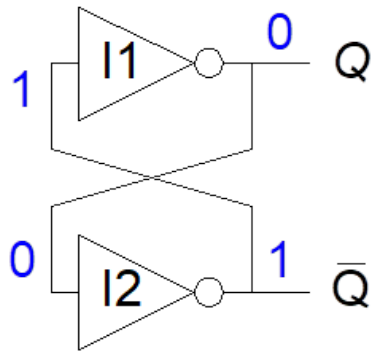
Figura 46: Par de inversores *cross-coupled*.

Análise do Circuito Biestável

- Considere os dois possíveis casos:

1) $Q = 0$
então $\overline{Q} = 1$, $Q = 0$ (estável)
2) $Q = 1$
então $\overline{Q} = 0$, $Q = 1$ (estável)





- Armazena 1 bit de estado na variável de estado, Q (ou $\overline{Q}$);
- Mas não há entradas para controlar o estado.

Báscula SR

- Báscula SR

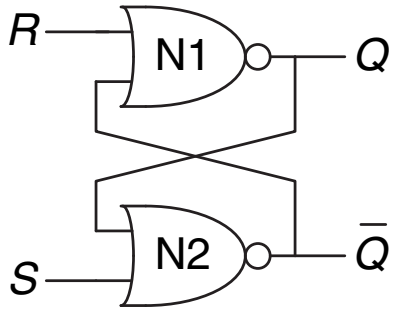
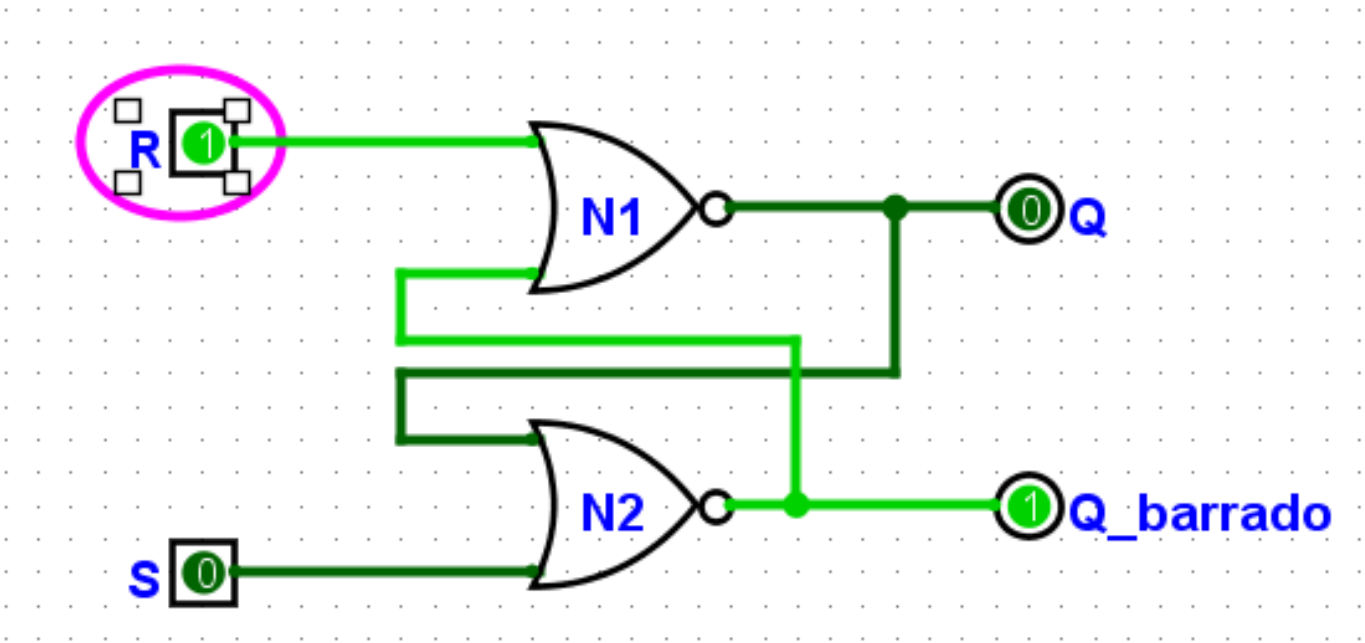


Figura 47: Esquema de uma báscula RS.

- Considere os quatro possíveis casos
  - R = 1, S = 0
  - S = 1, R = 0
  - S = 1, R = 1
  - S = 0, R = 0

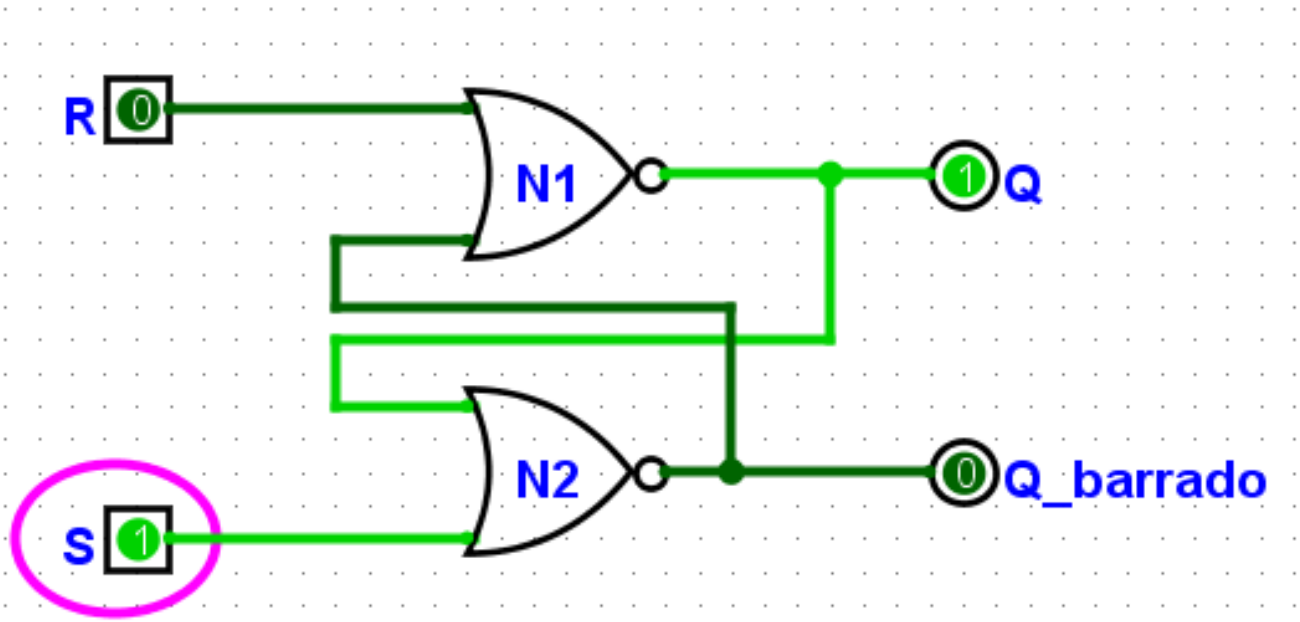
Báscula SR

R = 1, S = 0:
N1 vê uma entrada TRUE, R, de modo que produz uma saída FALSE em Q. N2 vê ambos Q e S FALSE, por isso produz uma saída TRUE em $\overline{Q}$;



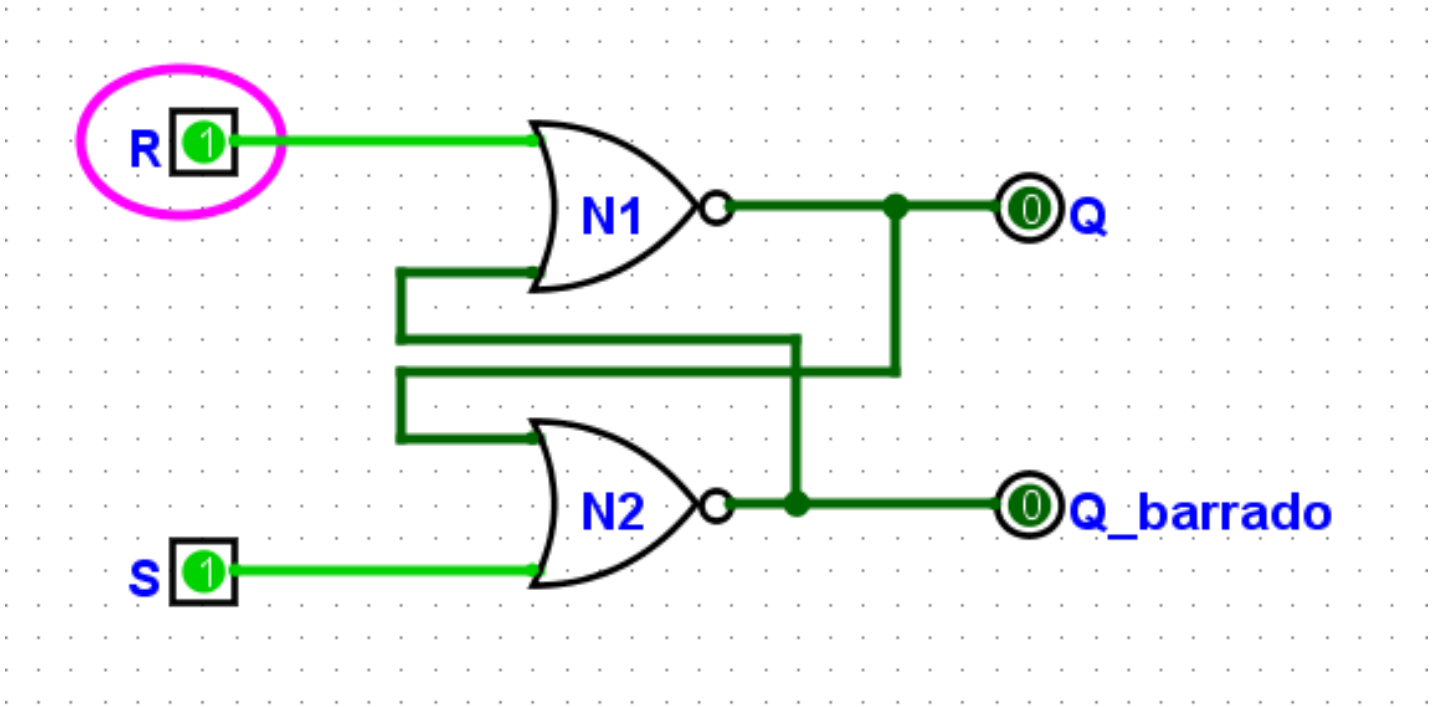
Báscula SR

R = 0, S = 1:
N1 recebe entradas de 0 e $\overline{Q}$. N2 recebe finalmente uma entrada TRUE, S, por isso produz uma saída FALSE em $\overline{Q}$: Agora podemos re-visitar N1, sabendo que ambas as entradas são FALSE, então a saída Q é TRUE;



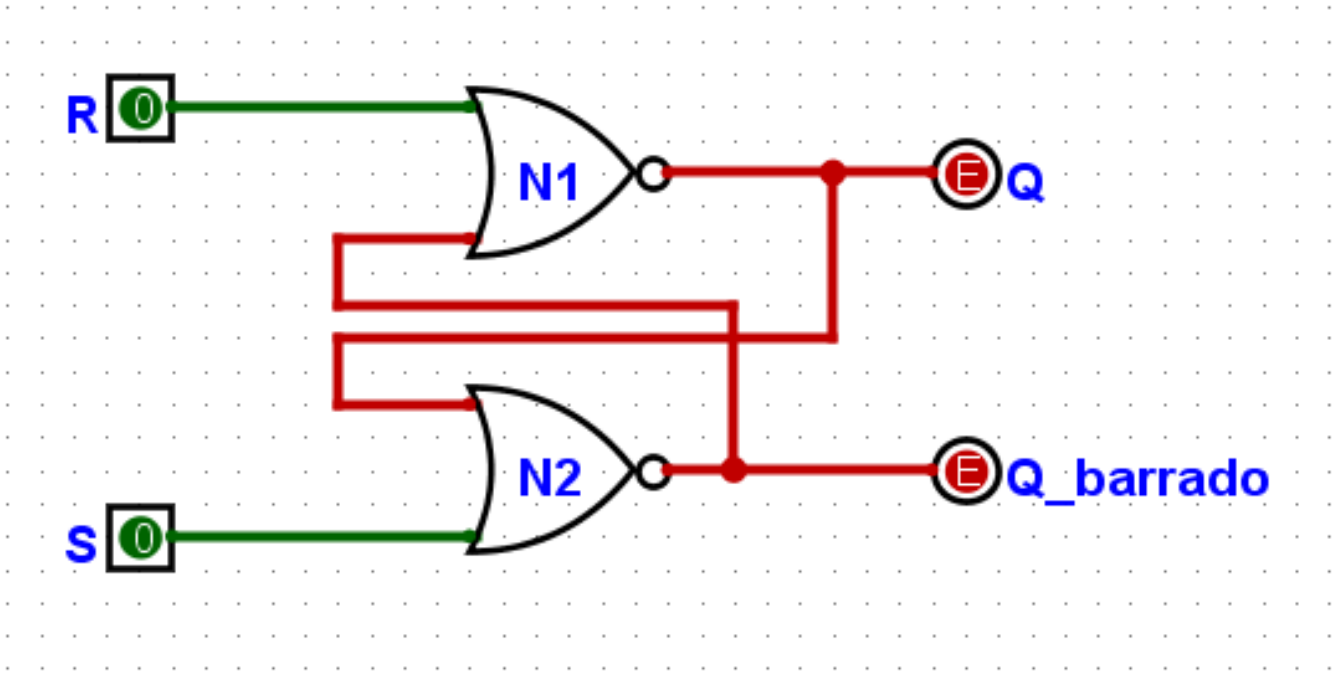
Báscula SR

R = 1, S = 1:
N1 e N2 veem uma entrada TRUE (R ou S), de modo que cada uma produz uma saída FALSE. Daí Q e $\overline{Q}$ serem ambas FALSE;



Báscula SR

R = 0, S = 0:
N1 recebe as entradas de 0 e $\overline{Q}$. N2 recebe entradas de 0 e Q. Agora estamos presos. Esta é uma reminiscência dos inversores cross-coupled. Entretanto que Q deve ser 0 ou 1.



Análise Báscula SR

- Caso IVa: Q = 0
  então $Q = 0, \overline{Q} = 1$
- Caso IVb: Q = 1
  então $Q = 1, \overline{Q} = 0$

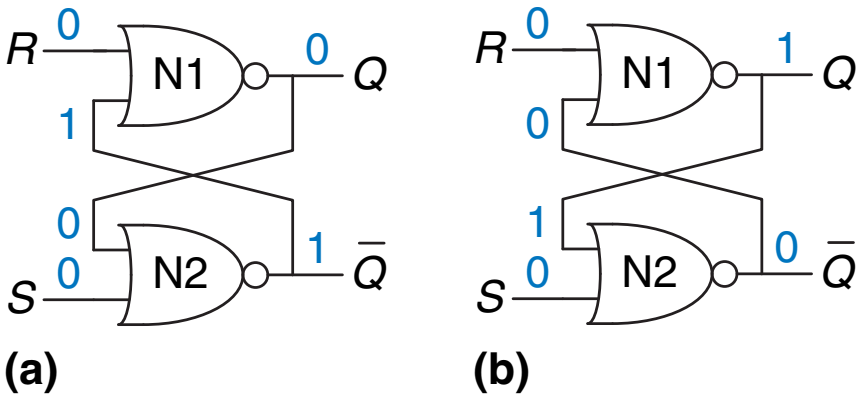
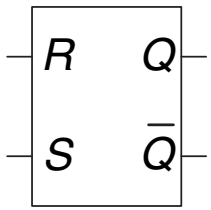


Figura 48: Estados biestáveis de uma báscula SR

Análise Báscula SR

Case	S	R	Q	$\overline{Q}$
IV	0	0	Q_{prev}	$\overline{Q}_{prev}$
I	0	1	0	1
II	1	0	1	0
III	1	1	0	0

(a)



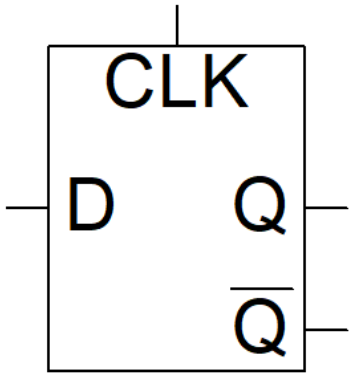
(b)

Figura 49: (a) Tabela verdade de uma báscula SR. (b) Símbolo de uma báscula SR.

- Se Q tem algum valor antes conhecido, será chamado de Q_{prev}.

Báscula D

- A entrada de dados, D, controla qual deve ser o próximo estado.
- A entrada de clock, CLK, controla quando o estado deve mudar



Função:

Quando CLK = 1:

D passa para Q (báscula está transparente).

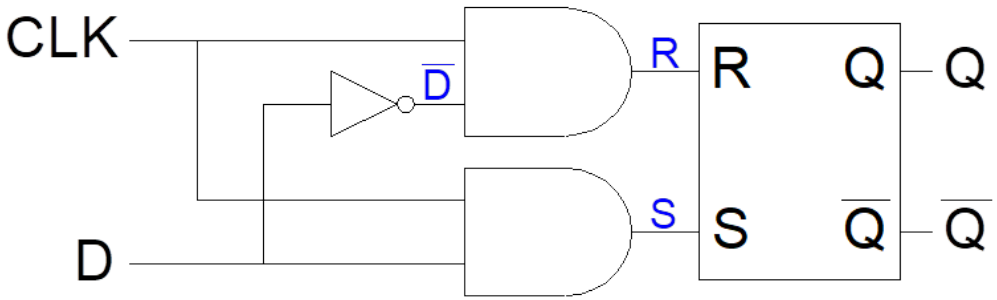
Quando CLK = 0:

Q mantém o seu valor anterior (báscula está opaca).

Evita casos inválidos quando:

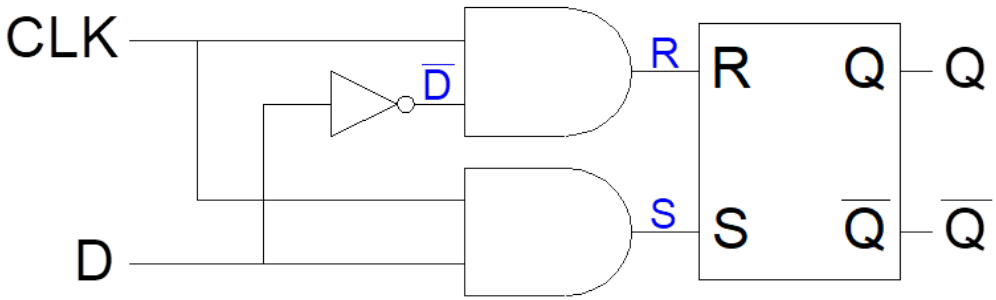
$Q \neq NOT \quad \overline{Q}$

Circuito Interno Báscula D



CLK	D	$\overline{D}$	S	R	Q	$\overline{Q}$
0	X					
1	0					
1	1					

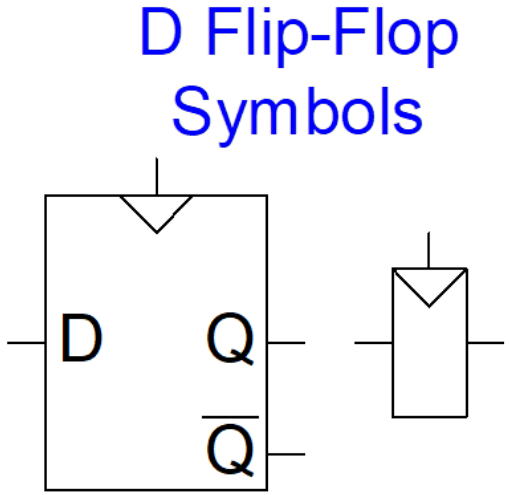
Circuito Interno Báscula D



CLK	D	$\overline{D}$	S	R	Q	$\overline{Q}$
0	X	$\overline{X}$	0	0	Q_{prev}	$\overline{Q_{prev}}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

Flip-Flop D

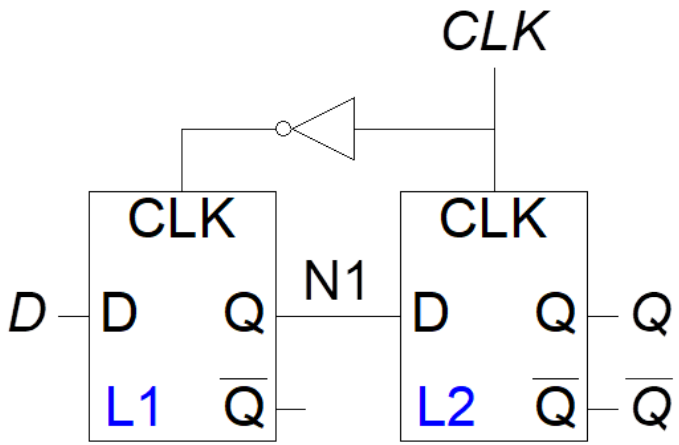
- **Entradas**: CLK, D;
- Um flip-flop D pode ser construído a partir de duas básculas D controladas por relógios complementares;
- A primeira báscula, L1, é chamada de mestre. A segunda báscula, L2, é chamada de escravo. O nó entre elas é chamado de N1.



Flip-flop D: Símbolo e símbolo condensado.

Circuito Interno Flip-Flop D

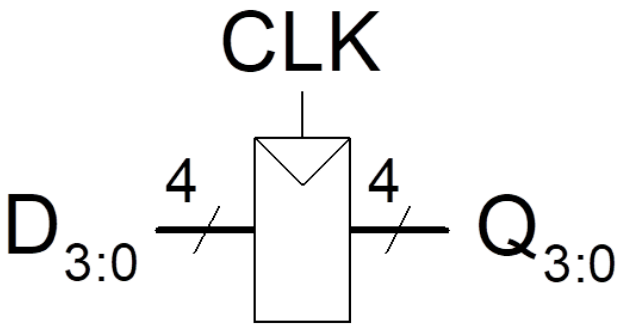
- Quando CLK = 0
  - L1 é transparente;
  - L2 é opaco;
  - O valor em D é propagado para $N1$.
- Quando CLK = 1
  - L1 é opaco;
  - L2 é transparente;
  - O valor em $N1$ é propagado para Q.
- Assim, na borda do relógio (quando CLK sobe de 0 $\rightarrow$ 1)
  - O valor que estava em D é copiado para Q.



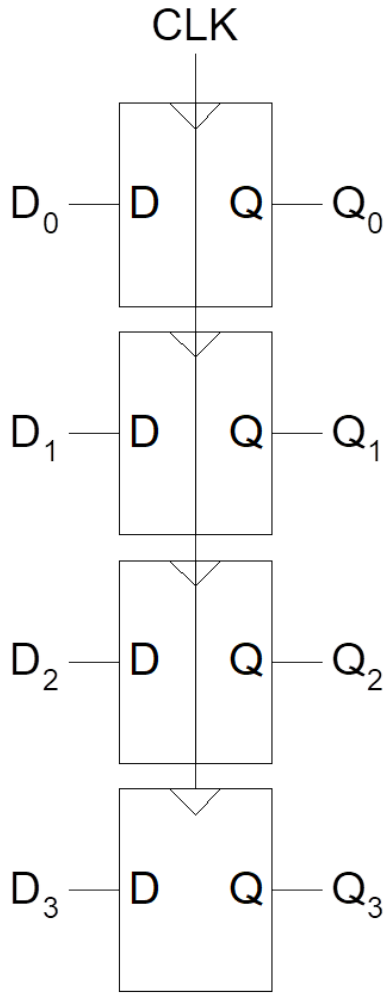
Flip-flop D: Esquemático.

Registro

- Um registro de N-bits é um banco de N *flip-flops* que compartilham uma entrada CLK comum, de modo que todos os bits do registros são atualizados em simultâneo.



Registro de 4-bits: Símbolo.



Registro de 4-bits: Esquemático.

Enabled Flip-Flops

- **Entradas**: CLK, D, EN.
- A entrada (EN) controla quando novos dados (D) é armazenado;
- **Função**:
  - $EN = 1$: D passa para Q;
  - $EN = 0$: o flip-flop mantém seu estado anterior.

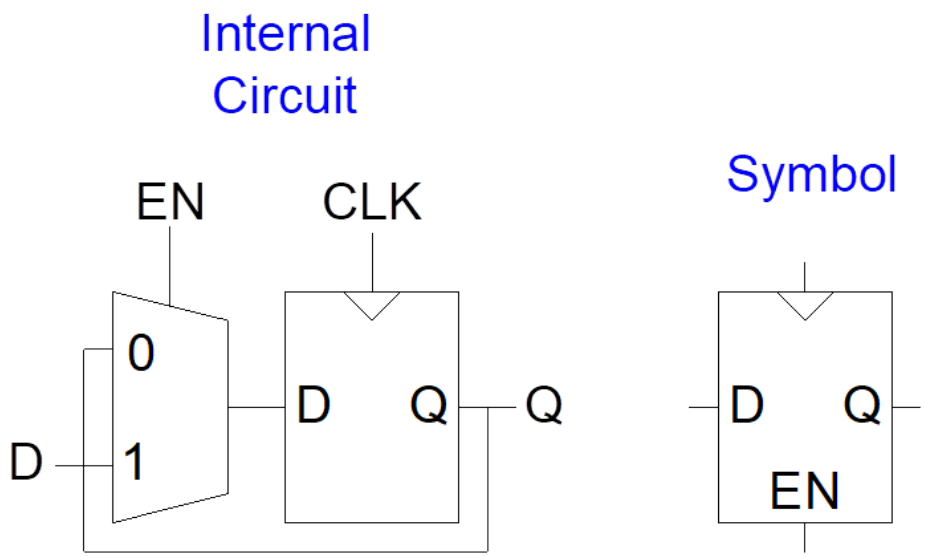


Figura 50: *Enabled flip-flop*: Esquemático e símbolo.

Resettable Flip-Flops

- **Entradas**: $CLK, D, Reset$.
- **Função**:
  - $Reset = 1$: Q é forçado para 0;
  - $Reset = 0$: comporta-se como um $flip-flop D$ comum.

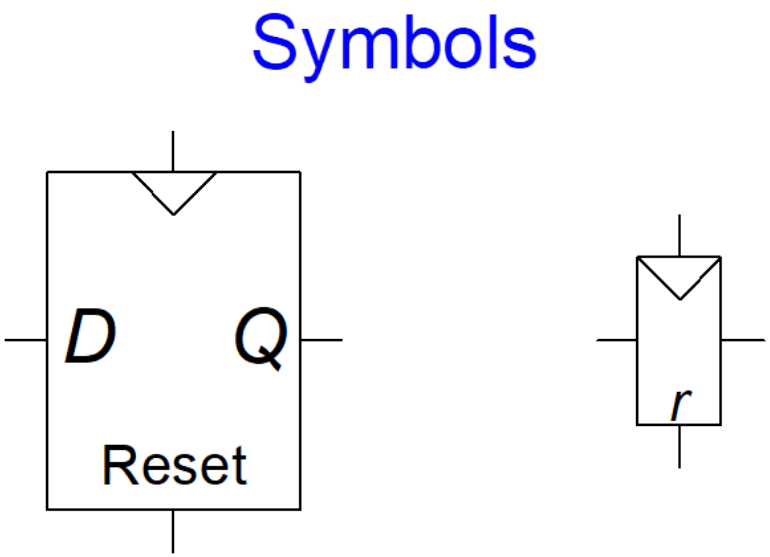


Figura 51: *Resettable flip-flop*: Esquemático e símbolo

Resettable Flip-Flops

- Dois tipos:
  - Síncrono: fazem *reset* a eles próprios apenas no flanco ascendente do relógio *CLK*;
  - Assíncrono: fazem *reset* a eles próprios, logo que *RESET* se torne *TRUE*, independente do *CLK*.

Settable Flip-Flops

- **Entradas**: CLK, D, Set.
- **Função**:
  - $Set = 1$: Q é definido como 1;
  - $Set = 0$: comporta-se como um $flip-flop D$ comum.

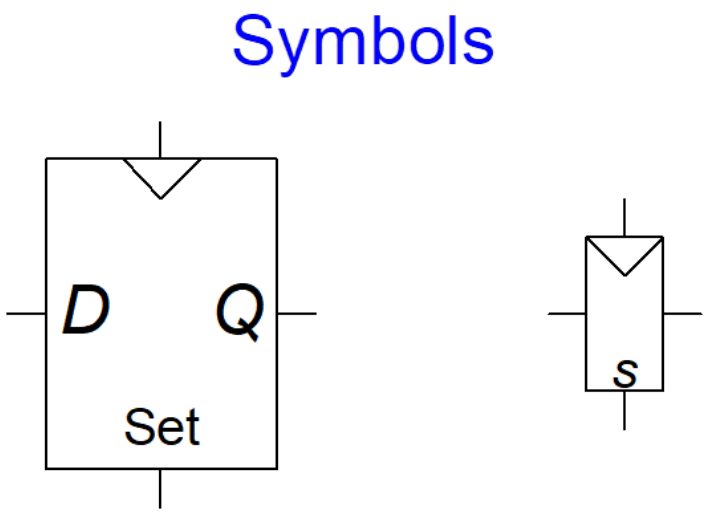


Figura 52: *Settable flip-flop*: Esquemático e símbolo

Temporização da Lógica Sequencial

- Um flip-flop copia a entrada D para a saída Q no flanco ascendente do relógio. Este processo é chamado de amostragem (sampling) de D na borda ascendente do relógio.
- D deve ser estável quando amostrado.
- Um elemento sequencial tem um tempo de abertura em torno do flanco do relógio, durante o qual a entrada deve ser estável para que o flip-flop possa produzir uma saída bem definida.

Restrições de Tempo de Entrada

- Tempo de preparação (Setup time): t_{setup} = intervalo de tempo antes dos dados da borda do relógio em que a entrada deve ser estável.
- Tempo de espera (Hold time): t_{hold} = intervalo de tempo após os dados da borda do relógio em que a entrada deve ser estável.
- Tempo de abertura (Aperture time): t_a = o tempo em torno dos dados da borda do relógio em que a entrada deve ser estável ($t_a = t_{setup} + t_{hold}$).

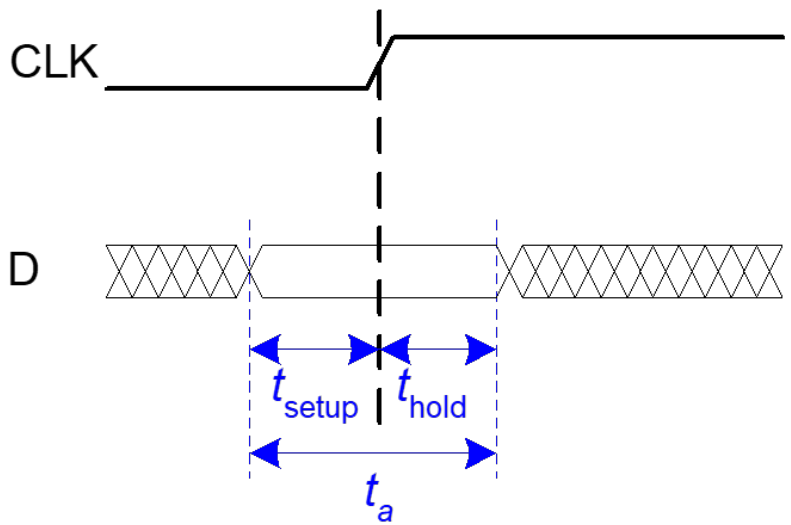


Figura 53: Restrições de Tempo de Entrada.

Restrições de Tempo de Saída

- Atraso de propagação (Propagation delay): t_{pcq} = tempo após a borda do relógio que garante que Q seja estável.
- Atraso de contaminação (Contamination delay): t_{ccq} = tempo após a borda do relógio em que Q pode ser instável.

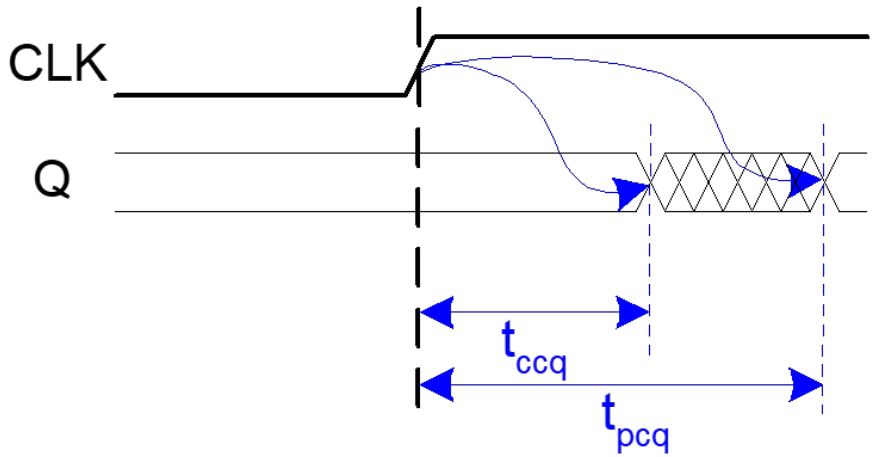


Figura 54: Restrições de Tempo de Saída.

Disciplina Dinâmica

- As entradas do circuito sequencial síncrono devem ser estáveis durante o tempo de abertura (setup e hold) em torno da borda do relógio.
- As entradas devem ser estáveis:
  - Pelo menos o t_{setup} antes da borda do relógio.
  - Pelo menos até depois do t_{hold} na borda do relógio.

Disciplina Dinâmica

O atraso entre registros apresentam um atraso mínimo e máximo, dependendo dos atrasos dos elementos do circuito.

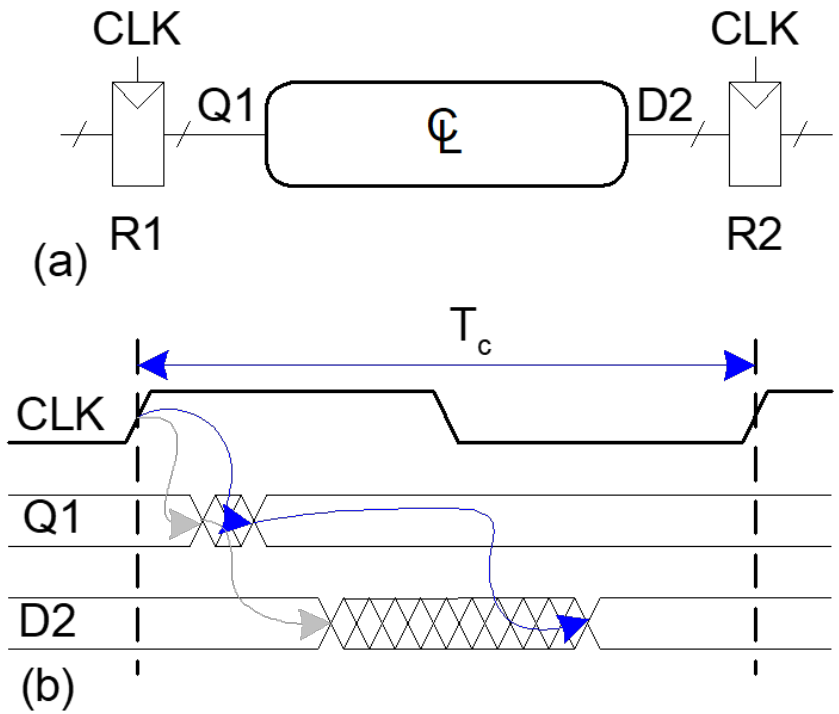


Figura 55: Caminho entre registros e diagrama temporal.

Restrição do Setup Time

- Depende do atraso máximo do registro R1 através da lógica combinacional para R2.
- A entrada para o registrar R2 deve ser estável, pelo menos, para t_{setup} antes da borda do relógio.

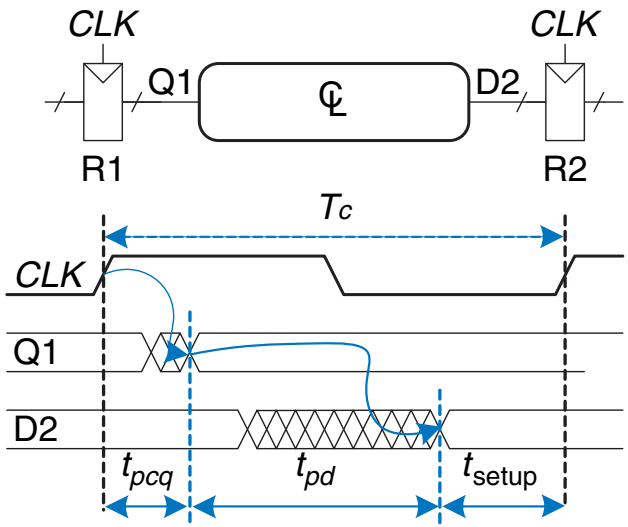


Figura 56: Máximo atraso para a restrição de setup time.

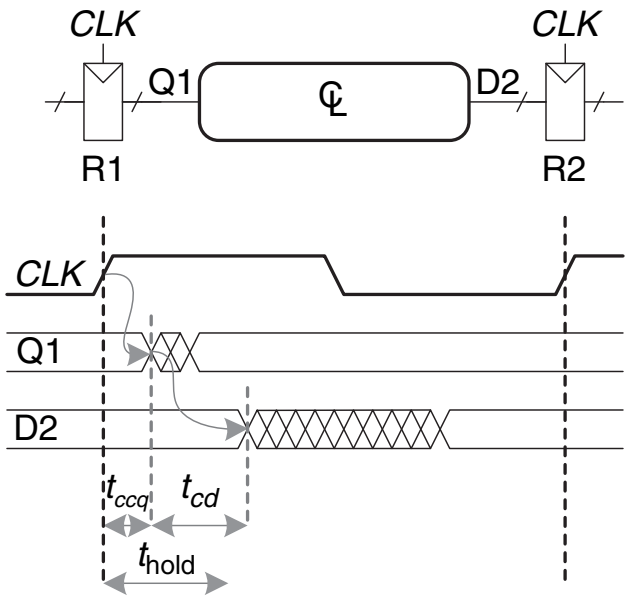
$$T_c \geq t_{pcq} + t_{pd} + t_{setup}$$

$$t_{pd} \leq T_c - (t_{pcq} + t_{setup})$$

$(t_{cpq} + t_{setup})$: sobrecarga de sequenciamento (sequencing overhead).

Restrição do Hold Time

- Depende do atraso mínimo do registro R1 através da lógica combinacional para R2.
- A entrada para registrar R2 deve ser estável pelo menos t_{hold} depois da borda do relógio.



$$t_{hold} < t_{ccq} + t_{cd}$$

$$t_{cd} > t_{hold} - t_{ccq}$$

Figura 57: Mínimo atraso para a restrição de hold time.

Skew do Relógio

- O relógio não chega a todos os registros ao mesmo tempo.
- Skew: diferença entre duas bordas do relógio.

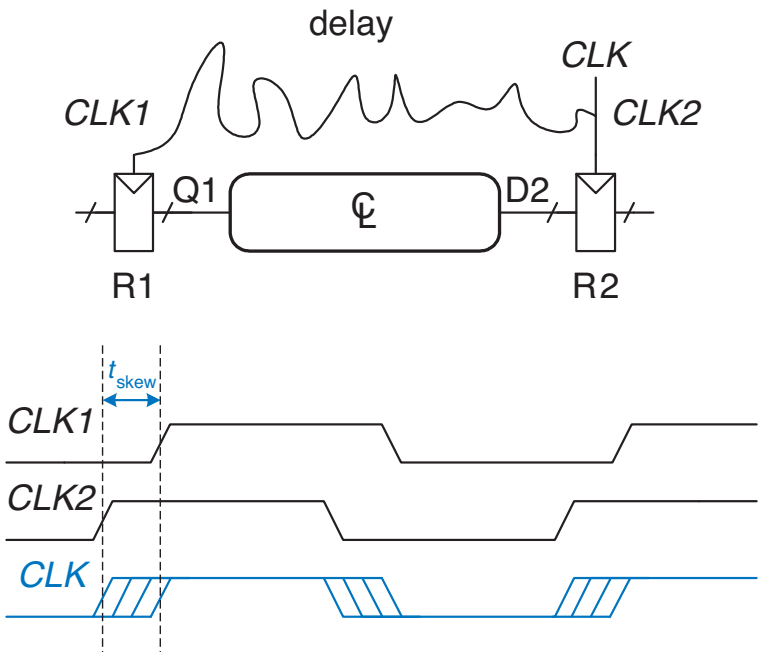


Figura 58: Skew do relógio provocado pelo atraso no condutor.

Restrição do setup time com skew do relógio

No pior dos casos, R1 recebe o relógio em atraso e R2 recebe o relógio em avanço, deixando pouco tempo para que os dados se possam propagar entre os registros.

$$T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$$

$$t_{pd} \leq T_c - (t_{pcq} + t_{setup} + t_{skew})$$

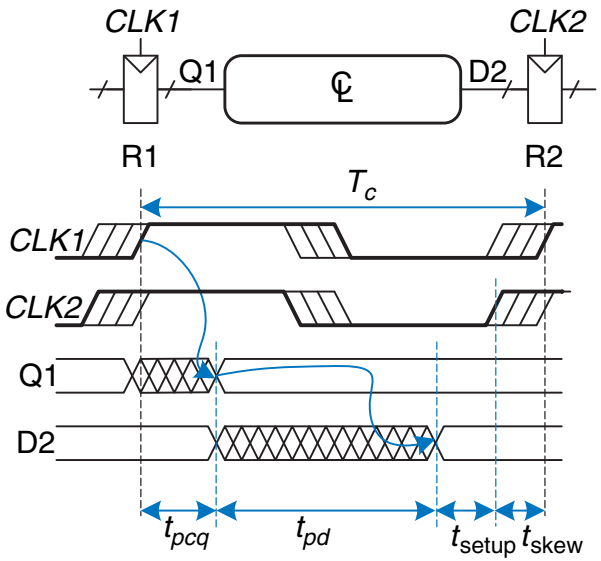


Figura 59: Restrição do setup time com skew do relógio.

Restrição do hold time com skew do relógio

No pior dos casos, R1 recebe um relógio em avanço, CLK1, e R2 recebe um relógio em atraso, CLK2. Os dados percorrem os registros e a lógica combinatória, mas não devem chegar até um hold time após o último relógio.

$$t_{ccq} + t_{cd} > t_{hold} + t_{skew}$$

$$t_{cd} > t_{hold} + t_{skew} - t_{ccq}$$

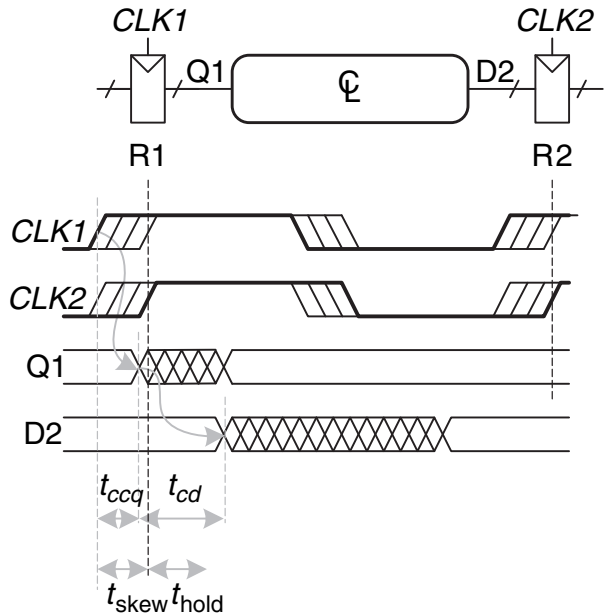


Figura 60: Restrição do hold time com skew do relógio.

Metaestabilidade

Quando um flip-flop amostra uma entrada que está em mudança durante a sua abertura, a saída Q pode assumir momentaneamente uma tensão entre 0 e VDD situada na zona proibida. Isso é chamado um estado metaestável.

- Dispositivos Biestáveis: apresenta tem um estado metaestável entre os dois estados estáveis.
- Flip-Flop: dois estados estáveis (1 e 0) e um estado metaestável.

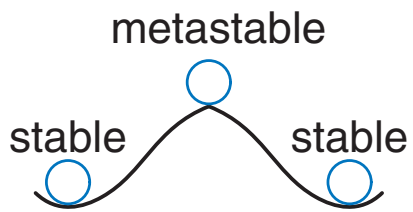


Figura 61: Estados estáveis e metaestáveis.

Sincronizadores

- Entradas assíncronas são inevitáveis.
- Objetivo do sincronizador: tornar a probabilidade de falha baixa.
- O sincronizador não consegue fazer a probabilidade de falha 0.

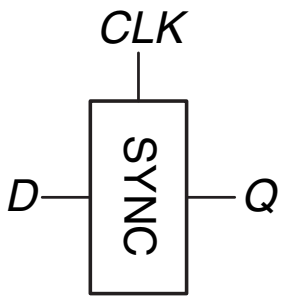


Figura 62: Símbolo de um sincronizador.

Sincronizador Interno

- Sincronizador: construído com dois flip-flops.
- Suponha que D esteja em transição quando amostrado pela F1.
- O sinal interno D2 tem $(T_c - t_{setup})$ tempo para resolver 1 ou 0.

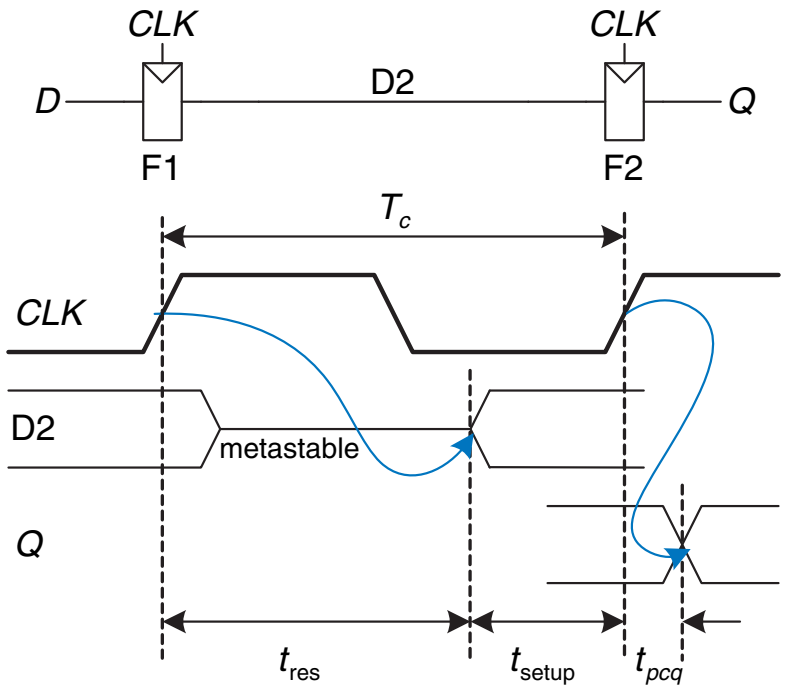


Figura 63: Sincronizador simples.

Probabilidade de Falha do Sincronizador

Para cada amostra, a probabilidade de falha é:

$$P(falha) = \frac{T_0}{T_c} e^{-(T_c - t_{setup})/\tau}$$

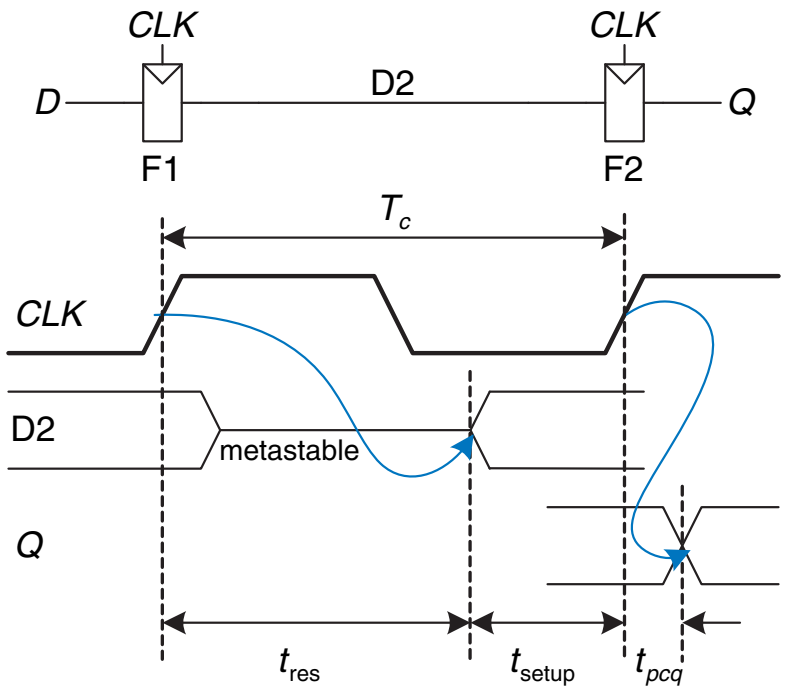


Figura 64: Sincronizador simples.

Tempo médio entre falhas do sincronizador

- Se a entrada assíncrona muda uma vez por segundo, a probabilidade de falha por segundo é P.
- Se a entrada muda N vezes por segundo, a probabilidade de falha por segundo é:

$$P(falha)/seg = \frac{NT_0}{T_c} e^{-(T_c - t_{setup})/\tau}$$

- A fiabilidade do sistema geralmente é medida em tempo médio entre falhas (MTBF - mean time between failures). O MTBF é a quantidade média de tempo entre as falhas do sistema.

$$MTBF = 1/(P(falha)/seg) \frac{T_c}{NT_0} e^{-(T_c - t_{setup})/\tau}$$

Exemplo

Considere:

- $T_c = 1/1500\,MHz = 2\,ns$.
- $T_0 = 150\,ps$.
- $N = 10$ eventos por segundo.
- $\tau = 200\,ps$.
- $t_{setup} = 100\,ps$.

Qual a probabilidade de falta? MTBF?

Paralelismo

- Paralelismo espacial: múltiplas cópias do hardware são fornecidas de modo que múltiplas tarefas podem ser realizadas ao mesmo tempo.
- Paralelismo temporal: uma tarefa é dividida em etapas.

Definições de paralelismo:

- Token: grupo de entradas que são processadas para produzir um grupo de saídas.
- Latência: tempo requerido para que um token passe através do sistema desde o início até ao fim.
- Taxa de transferência: é determinada pelo número de tokens que podem ser processados por unidade de tempo.

Exercícios

1. Dadas as formas de onda mostradas na Figura abaixo, esboçar a saída, Q, de uma báscula SR.

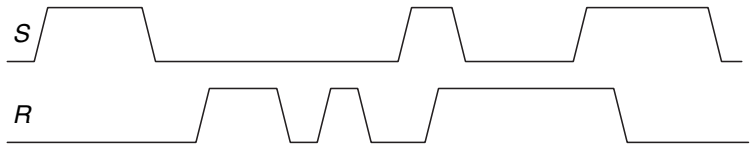


Figura 65: Formas de onda de entrada da báscula SR.

2. Tendo em conta as formas de onda mostradas na Figura abaixo, esboçar a saída, Q, de um flip-flop D.

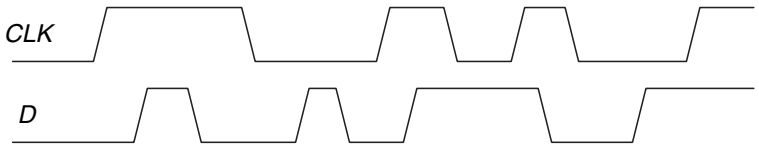


Figura 66: Formas de onda de entrada da báscula D ou flip-flop.

Exercícios

3. O circuito na Figura abaixo é de lógica combinatória ou de lógica sequencial? Explicar de uma forma simples, qual é a relação entre as entradas e saídas. Como o leitor chamaria a este circuito?

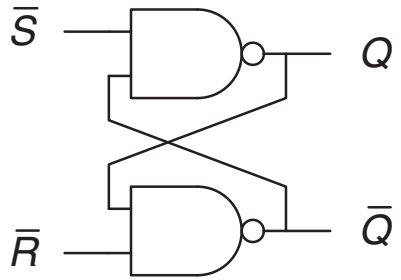


Figura 67: Circuito mistério.

4. O leitor está a projetar um controlador para um elevador de um edifício de 25 andares. O controlador possui duas entradas: para cima e para baixo. Ele produz uma saída que indica o andar que o elevador está ligado. Não há piso 13. Qual é o número mínimo de bits de estado do controlador?

Exercícios

5. O leitor gostaria de construir um sincronizador que pode receber entradas assíncronas com um MTBF de 50 anos. O sistema opera a 1 GHz, e são utilizados flip-flops com amostragem $\tau = 100ps$, $T_0 = 110ps$, e $t_setup = 70ps$. O sincronizador recebe uma nova entrada assíncrona em média $0{,}5$ vezes por segundo (isto é, uma vez a cada 2 segundos). Qual é a probabilidade de falha requerida no cumprimento desta MTBF? Quantos ciclos de relógio seriam necessários esperar antesde ler o sinal de entrada amostrado para obter essa probabilidade de erro?

6. O leitor construiu um sincronizador usando flip-flops com $T_0 = 20ps$ e $\tau = 30ps$. O seu chefe diz-lhe que vai ser preciso aumentar o MTBF por um fator de 10. De quanto será preciso incrementar o período de relógio?

Exercícios

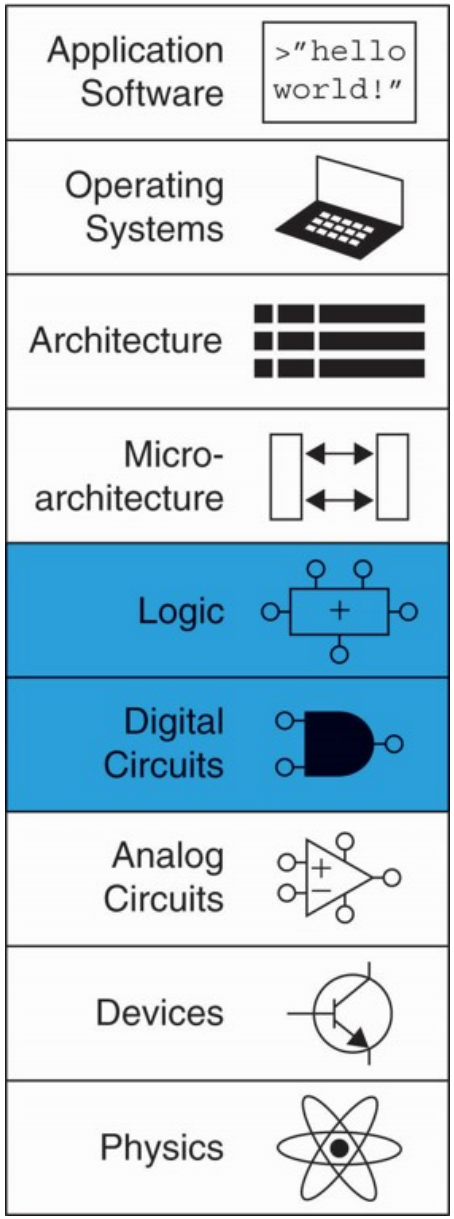
7. Qual é a diferença entre uma báscula e um flip-flop? Em que circunstâncias é preferível cada um?

8. Descreva o que significa para um flip-flop ter um hold time negativo.

9. Considere um bloco de lógica entre dois registros. Explicar as restrições de tempo. Se for adicionado um buffer na entrada do relógio do receptor (o segundo flip-flop), a restrição de setup time ficar melhor ou pior?

Linguagem de Descrição de Hardware

- Introdução;
- Lógica combinatória em SystemVerilog;
- Modelo estrutural;
- Lógica sequencial.



Introdução

- Linguagem de descrição de hardware (HDL)
  - Especifica apenas a função lógica.
  - A ferramenta de projeto auxiliado por computador(CAD-computer-aided design) produz as portas otimizadas.
- A maioria dos projetos comerciais utilizam HDLs.
- As duas principais linguagens de descrição de hardware são a SystemVerilog e a VHDL.
  - SystemVerilog
    - Desenvolvida em 1984 pela Gateway Design Automation.
    - Tornou-se um padrão IEEE em 1995.
    - A linguagem foi estendida em 1995 para simplificar idiossincrasias e para melhor suportar a modelagem e a verificação de sistemas.
  - VHDL 2008
    - Desenvolvida em 1981 pelo Departamento de Defesa.
    - Padronização do IEEE 1987.
    - Atualização do IEEE em 2008.

Simulação e Síntese

- Simulação
  - Entradas aplicadas no circuito.
  - As saídas são examinadas para verificar se o módulo opera corretamente.
- Síntese
  - A síntese lógica transforma o código HDL numa netlist descrevendo o hardware (isto é, as portas lógicas e os fios que as conectam).

SystemVerilog:

```
module example(input logic a, b, c,
               output logic y);
  assign y = ~a & ~b & ~c |a & ~b & ~c | a & ~b & c;
endmodule
```

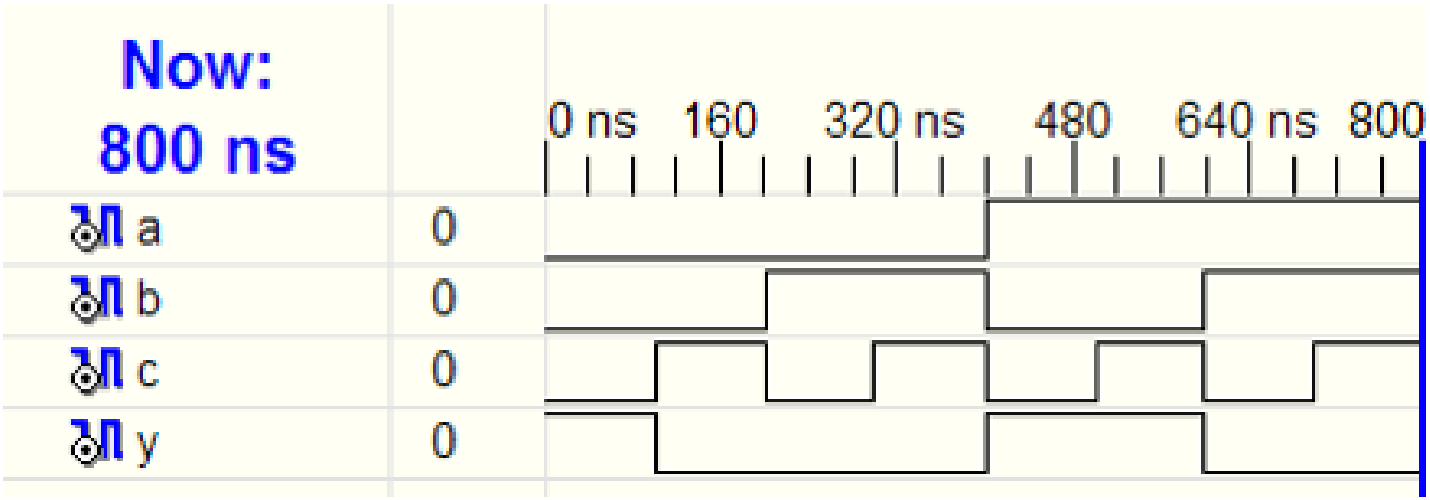


Figura 68: Formas de onda de simulação.

```
module example(input logic a, b, c,
               output logic y);
  assign assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b &  c;

endmodule
```

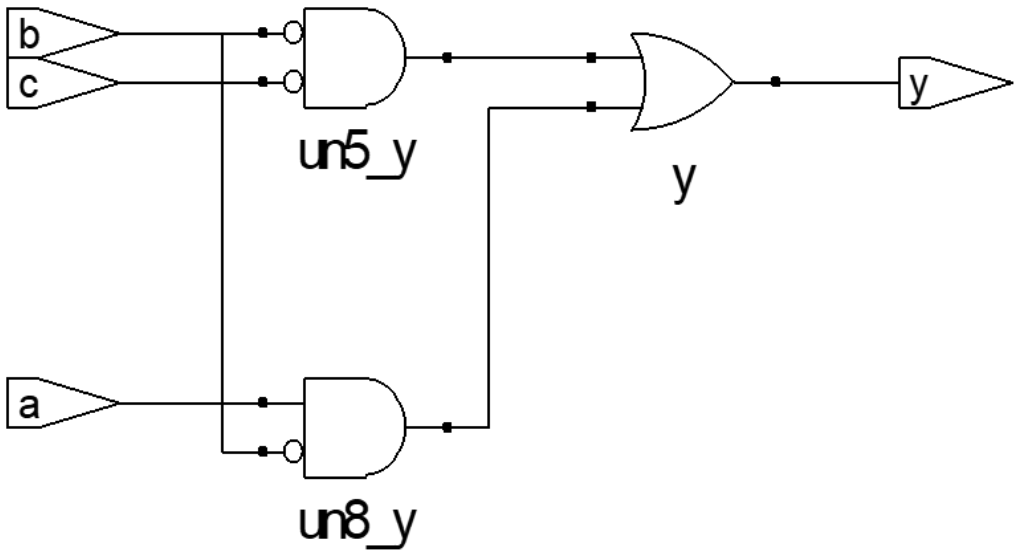


Figura 69: Circuito Sintetizado.

Operadores Bit a Bit

```
module gates(input  logic [3:0]  a, b,
             output logic [3:0] y1, y2, y3, y4, y5);
  /* Five different two-input logic
      gates acting on 4 bit busses */
  assign y1 = a & b;    // AND
  assign y2 = a | b;    // OR
  assign y3 = a ^ b;    // XOR
  assign y4 = ~(a & b); // NAND
  assign y5 = ~(a | b); // NOR
endmodule
```

Operadores de redução

```
module and8(input  logic [7:0] a,
            output logic       y);
  assign y = &a;
  // &a is much easier to write than
  // assign y = a[7] & a[6] & a[5] & a[4] &
  //            a[3] & a[2] & a[1] & a[0];
endmodule
```

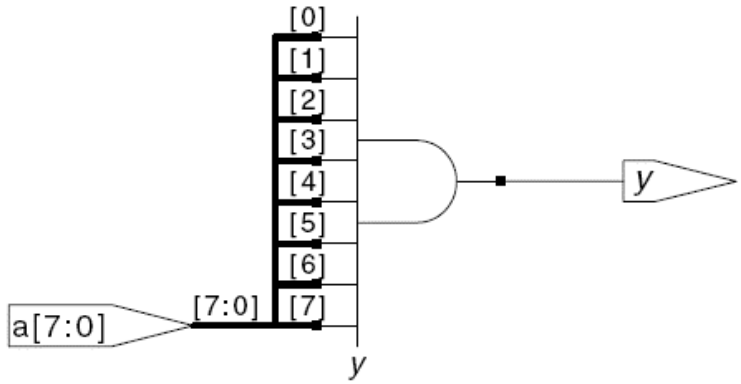


Figura 70: Circuito and8 sintetizado.

Atribuição condicional

```
module mux2(input  logic [3:0] d0, d1,
            input  logic       s,
            output logic [3:0] y);
  assign y = s ? d1 : d0;
endmodule
```

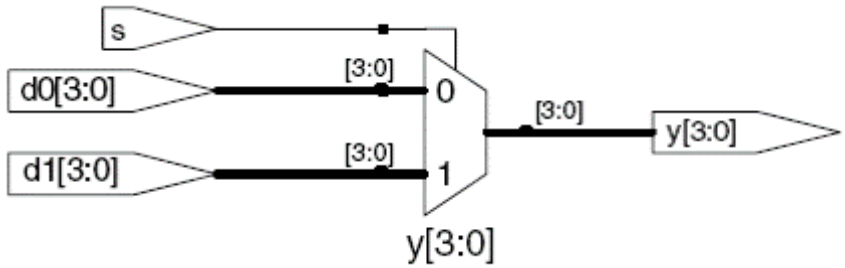


Figura 71: Circuito mux2 sintetizado.

? : é chamado de operador ternário porque opera em 3 entradas: s, d1 e d0.

Variáveis Internas

```
module fulladder(input  logic a, b, cin,
                 output logic s, cout);
  logic p, g;   // internal nodes
  assign p = a ^ b;
  assign g = a & b;
  assign s = p ^ cin;
  assign cout = g | (p & cin);
endmodule
```

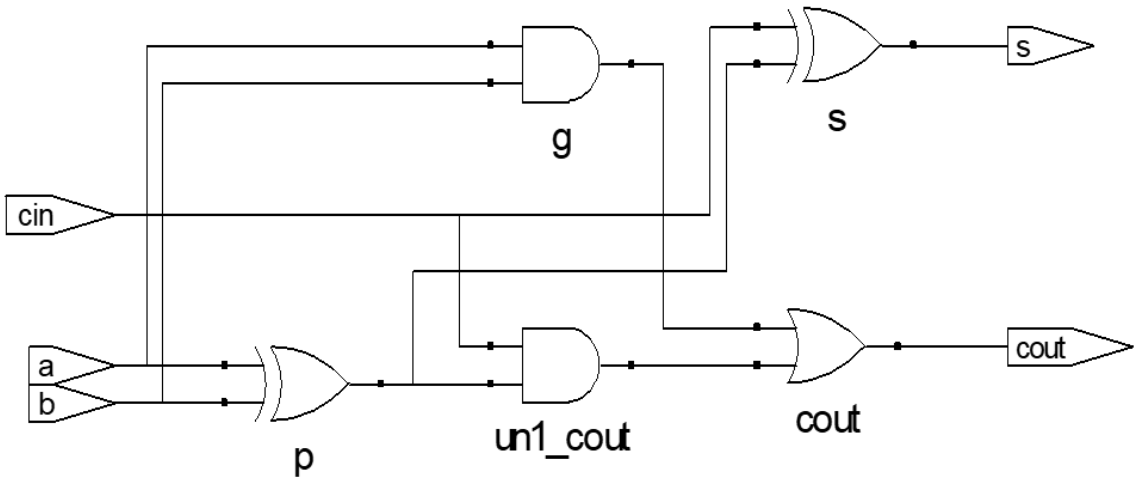


Figura 72: Circuito full adder sintetizado.

Precedência

~	NOT
*, /, %	mult, div, mod
+, -	add,sub
<<, >>	shift
<<<, >>>	arithmetic shift
<, <=, >, >=	comparison
==, !=	equal, not equal
&, ~&	AND, NAND
^, ~^	XOR, XNOR
|, ~|	OR, NOR
?:	ternary operator

Figura 73: Ordem das operações.

Números

Formato: N'Bvalue

N = número de bits, B = base

Number	# Bits	Base	Decimal Equivalent	Stored
3'b101	3	binary	5	101
'b11	unsized	binary	3	00…0011
8'b11	8	binary	3	00000011
8'b1010_1011	8	binary	171	10101011
3'd6	3	decimal	6	110
6'o42	6	octal	34	100010
8'hAB	8	hexadecimal	171	10101011
42	Unsized	decimal	42	00…0101010

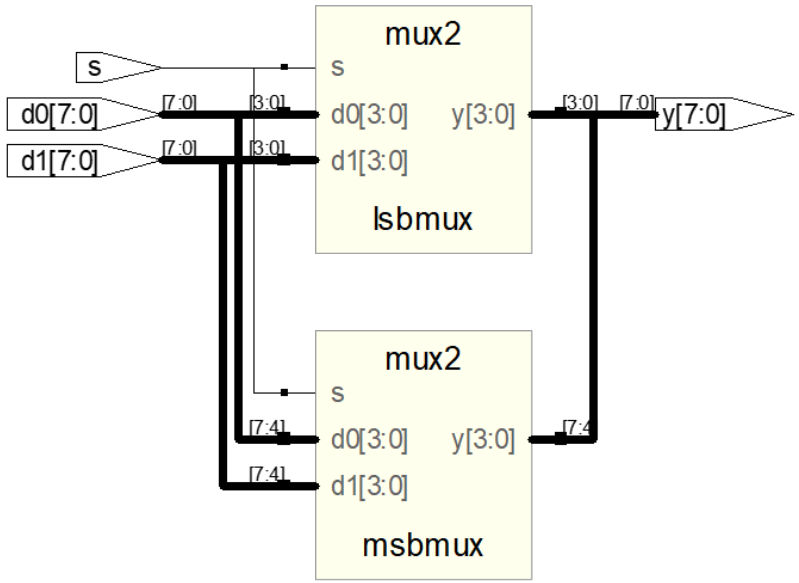
Figura 74: Números em SystemVerilog.

Manipulação de bits - exemplo

SystemVerilog:

```
module mux2_8(input  logic [7:0] d0, d1,
              input  logic       s,
              output logic [7:0] y);

    mux2 lsbmux(d0[3:0], d1[3:0], s, y[3:0]);
    mux2 msbmux(d0[7:4], d1[7:4], s, y[7:4]);
endmodule
```

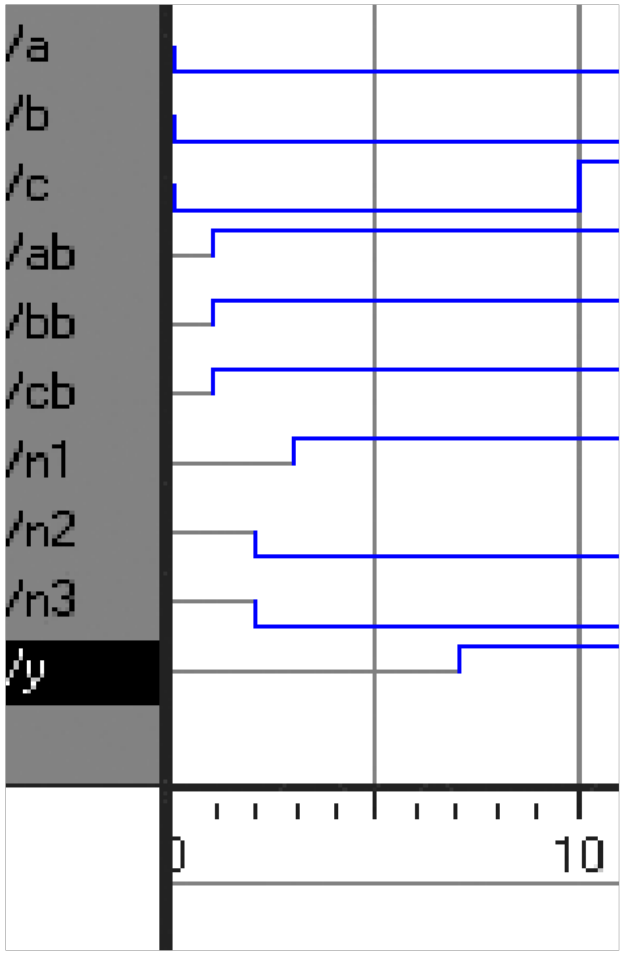


Atrasos

$y = \bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c} + a\bar{b}c;$

Atrasos: Inversores - $1ns$, porta AND (3 entradas) - $2ns$, porta OR (3 entradas) - $4ns$;

```
module example(input  logic a,b,c,
               output logic y);
  logic ab, bb, cb, n1, n2, n3;
  assign #1 {ab, bb, cb} =
                   ~{a, b, c};
  assign #2 n1 = ab & bb & cb;
  assign #2 n2 = a & bb & cb;
  assign #2 n3 = a & bb & c;
  assign #4 y = n1 | n2 | n3;
endmodule
```



Atrasos

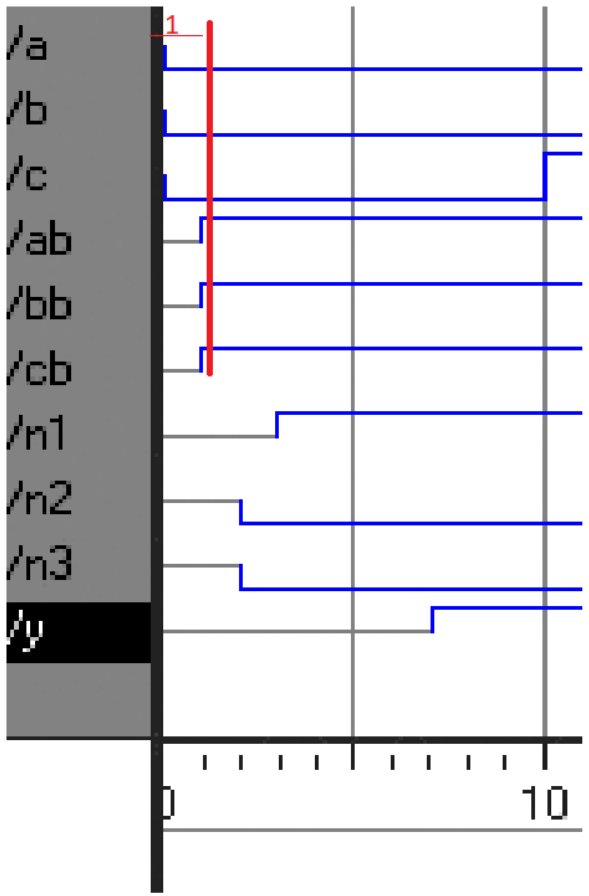
$y = \bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c} + a\bar{b}c$;

Atrasos: Inversores - $1ns$, porta AND (3 entradas) - $2ns$, porta OR (3 entradas) - $4ns$;

```
module example(input  logic a,b,c,
               output logic y);
  logic ab, bb, cb, n1, n2, n3;


 assign # 1 {ab,bb,cb}=
~{a,b,c};

  assign #2 n1 = ab & bb & cb;
  assign #2 n2 = a & bb & cb;
  assign #2 n3 = a & bb & c;
  assign #4 y = n1 | n2 | n3;
endmodule
```



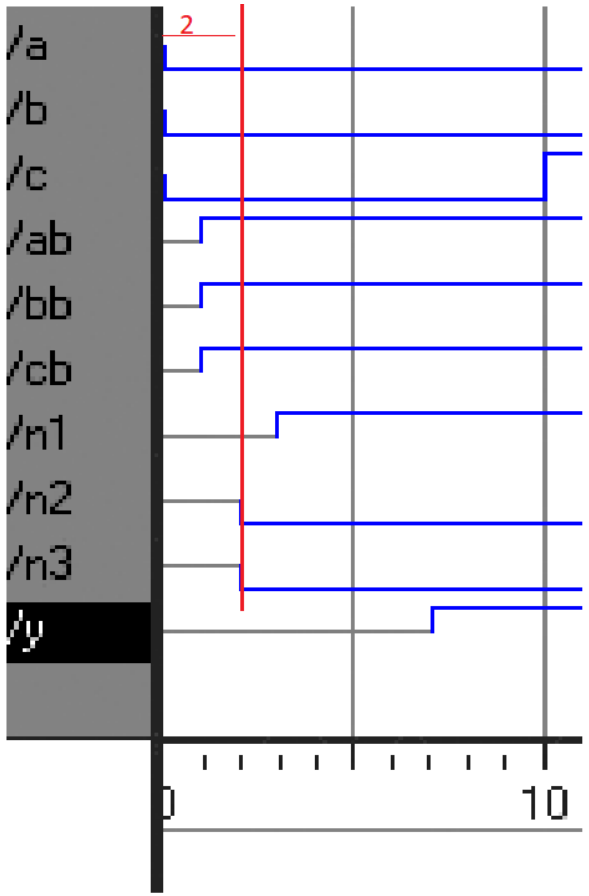
Atrasos

$y = \bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c} + a\bar{b}c;$

```
module example(input  logic a,b,c,
               output logic y);
  logic ab, bb, cb, n1, n2, n3;
  assign #1 {ab, bb, cb} =
                    ~{a, b, c};
  assign #2 n1 = ab & bb & cb;
```

```
 assign #2 n2 = a & bb & cb;
assign #2 n3 = a & bb & c;
```

```
  assign #4 y = n1 | n2 | n3;
endmodule
```



Atrasos

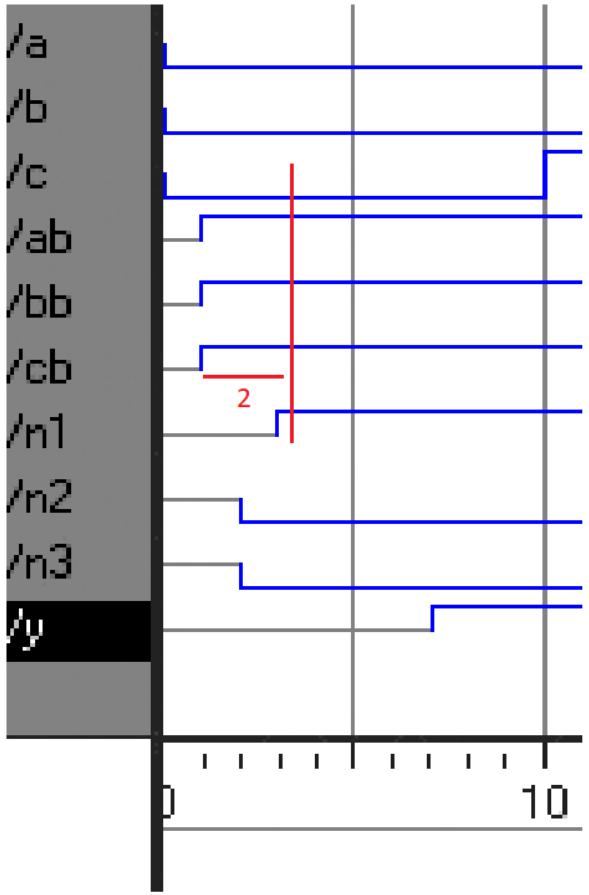
$y = \bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c} + a\bar{b}c;$

Atrasos: Inversores - $1ns$, porta AND (3 entradas) - $2ns$, porta OR (3 entradas) - $4ns$;

```
module example(input  logic a,b,c,
               output logic y);
  logic ab, bb, cb, n1, n2, n3;
  assign #1 {ab, bb, cb} =
                  ~{a, b, c};

 assign #2 n1 =ab & bb & cb;

  assign #2 n2 = a & bb & cb;
  assign #2 n3 = a & bb & c;
  assign #4 y = n1 | n2 | n3;
endmodule
```



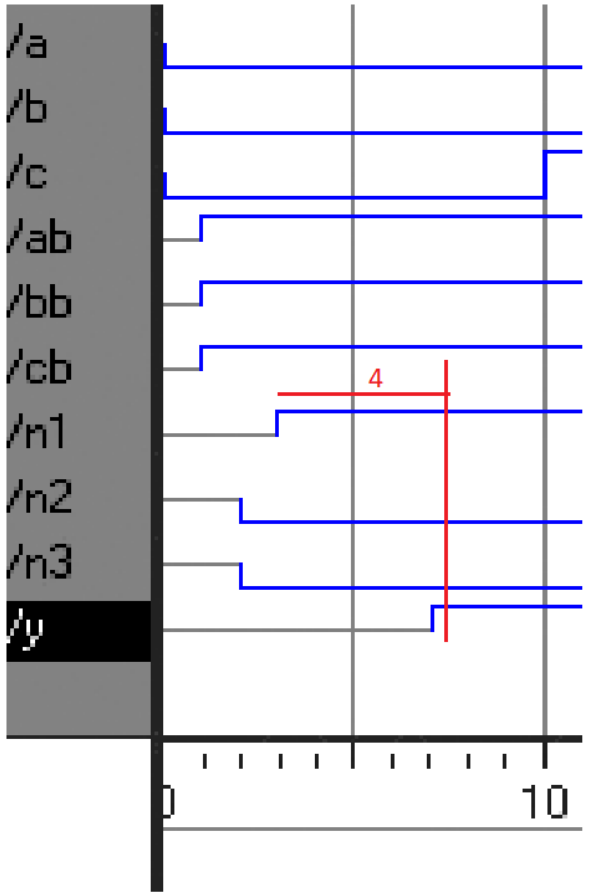
Atrasos

$y = \bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c} + a\bar{b}c;$

```
module example(input  logic a,b,c,
               output logic y);
  logic ab, bb, cb, n1, n2, n3;
  assign #1 {ab, bb, cb} =
                   ~{a, b, c};
  assign #2 n1 = ab & bb & cb;
  assign #2 n2 = a & bb & cb;
  assign #2 n3 = a & bb & c;


 assign # 4 y = n1 | n2 | n3;

endmodule
```



Lógica Sequencial

- *SystemVerilog* usa expressões idiomáticas para descrever registradores e básculas;

- Outros estilos de código podem simular corretamente os circuitos, porém com erros flagrantes ou sutil.

Declaração Always

Estrutura Geral:

```
always @(sensitivity list)
  statement;
```

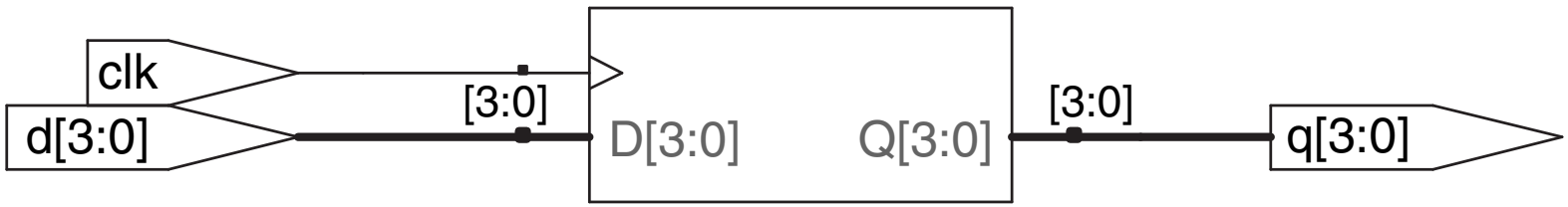
Sempre que ocorrer o evento na `sensitivity list`, o `statement` é executada;

Flip-Flop D

```
module flop(input  logic        clk,
            input  logic [3:0] d,
            output logic [3:0] q);

  always_ff @(posedge clk)
    q <= d;

endmodule
```



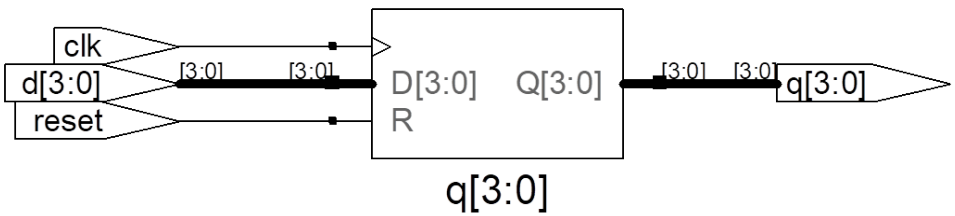
Flip-Flop D Resettable

```
module flopr(input  logic        clk,
             input  logic        reset,
             input  logic [3:0]  d,
             output logic [3:0]  q);

  // synchronous reset

  always_ff @(posedge clk)
    if (reset) q <= 4'b0;
    else       q <= d;

endmodule
```

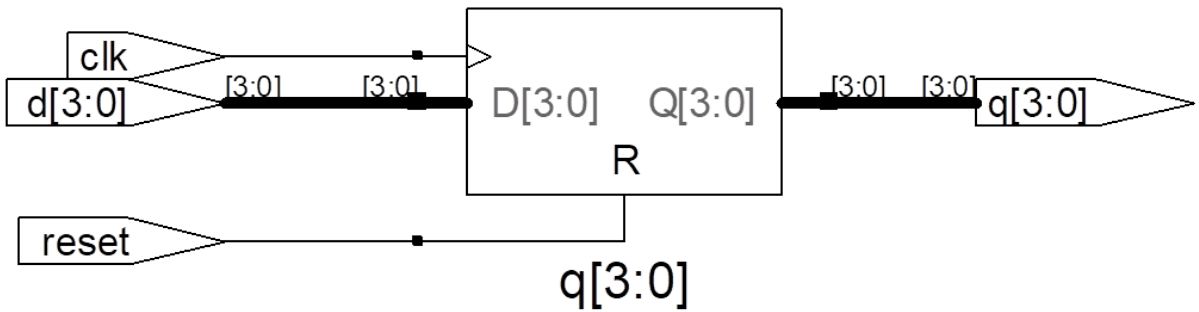


Flip-Flop D Resettable

```
module flopr(input  logic        clk,
             input  logic        reset,
             input  logic [3:0]  d,
             output logic [3:0]  q);

  // asynchronous reset
  always_ff @(posedge clk, posedge reset)
    if (reset) q <= 4'b0;
    else       q <= d;

endmodule
```

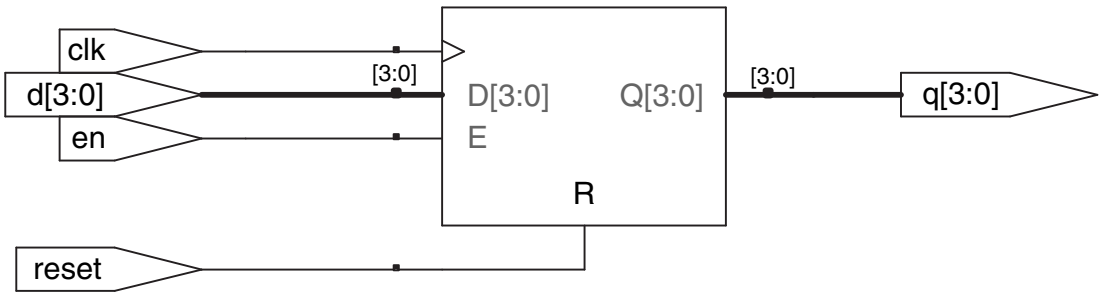


Registradores Enabled

```
module flopren(input  logic       clk,
               input  logic       reset,
               input  logic       en,
               input  logic [3:0] d,
               output logic [3:0] q);

 // asynchronous reset and enable
  always_ff @(posedge clk, posedge reset)
    if      (reset) q <= 4'b0;
    else if (en)    q <= d;

endmodule
```



Básculas

```
module latch(input  logic       clk,
             input  logic [3:0] d,
             output logic [3:0] q);

  always_latch
    if (clk) q <= d;

endmodule
```

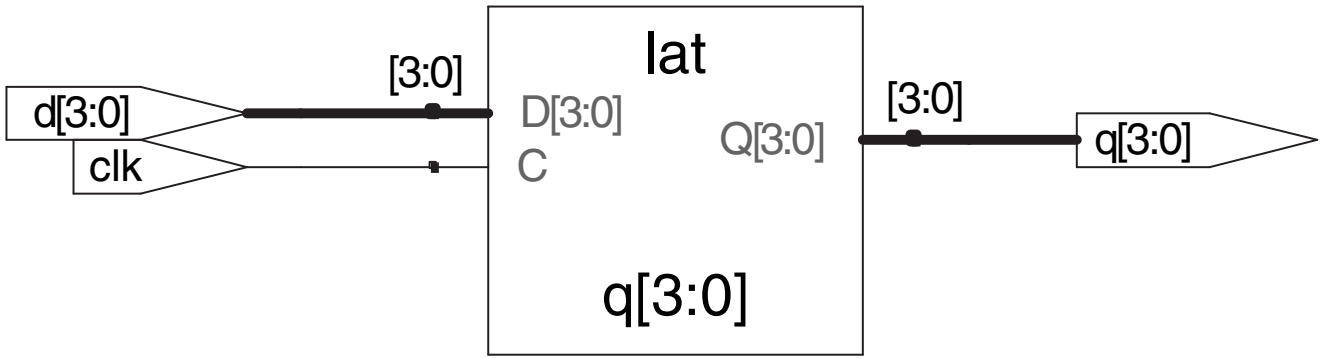


Figura 76: Circuito báscula sintetizado

Outras Declarações

Declarações que devem estar dentro de `always statements`:

- `if`/`else`
- `case`, `casez`

Lógica Combinatória usando `always`

```
    // combinational logic using an always statement
module gates(input  logic [3:0] a, b,
             output logic [3:0] y1, y2, y3, y4, y5);

always_comb          // need begin/end because there is
    begin              // more than one statement in always
      y1 = a & b;      // AND
      y2 = a | b;      // OR
      y3 = a ^ b;      // XOR
      y4 = ~(a & b); // NAND
      y5 = ~(a | b); // NOR
    end
endmodule
```

Lógica Combinatória usando `case`

```
  module sevenseg(input  logic [3:0] data,
                  output logic [6:0] segments);
  always_comb
    case (data)
      //                      abc_defg
      0: segments =       7'b111_1110;
      1: segments =       7'b011_0000;
      2: segments =       7'b110_1101;
      3: segments =       7'b111_1001;
      4: segments =       7'b011_0011;
      5: segments =       7'b101_1011;
      6: segments =       7'b101_1111;
      7: segments =       7'b111_0000;
      8: segments =       7'b111_1111;
      9: segments =       7'b111_0011;
      default: segments = 7'b000_0000; // required
  endcase
endmodule
```

Lógica Combinatória usando `casez`

```
module priority_casez(input  logic [3:0] a,
                      output logic [3:0] y);
  always_comb
    casez(a)
      4'b1???: y = 4'b1000;  // ? = don't care
      4'b01??: y = 4'b0100;
      4'b001?: y = 4'b0010;
      4'b0001: y = 4'b0001;
      default: y = 4'b0000;
  endcase
endmodule
```

Exercícios

1. Esboce o esquemático do circuito descrito pelo seguinte código HDL. Simplifique o esquemático para que ele tenha um número mínimo de portas lógicas.

SystemVerilog

```
module exercise1(input logic a, b, c,
                 output logic y, z);
  assign y = a & b & c | a & b & ~c | a & ~b &
c;
  assign z = a & b | ~a & ~b;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity exercise1 is
  port(a, b, c: in STD_LOGIC;
       y, z: out STD_LOGIC);
end;

architecture synth of exercise1 is
begin
  y <= (a and b and c) or (a and b and not c)
or (a and not b and c);
  z <= (a and b) or (not a and not b);
end;
```

2. Escreva um módulo HDL chamado minority. Ele recebe três entradas, a, b e c, e produz uma saída, y, que é TRUE se ao menos duas das entradas são FALSE.

Exercícios

3. Escreva um módulo HDL para um decodificador de display de sete segmentos hexadecimal. O decodificador deve lidar com os dígitos A, B, C, D, E e F, assim como de 0 a 9.

4. Escreva um módulo multiplexador 8:1 chamado mux8 com entradas $S_{2:0}$,$d0$,$d1$,$d3$,$d4$,$d5$,$d6$,$d7$ e a saída y.

5. Escreva um módulo estrutural para computar a função lógica $y = a\overline{b} + \overline{b}\overline{c} + \overline{a}bc$, utilizando lógica de multiplexadores. Utilize o multiplexador do exercício anterior.

Exercícios

6. Repita o exercício anterior utilizando um multiplexador 4:1 e quantas portas NOT forem necessárias.

7. Escreva um módulo HDL para uma báscula SR.

8. Qual o significado para um sinal declarado *tri* em SystemVerilog?