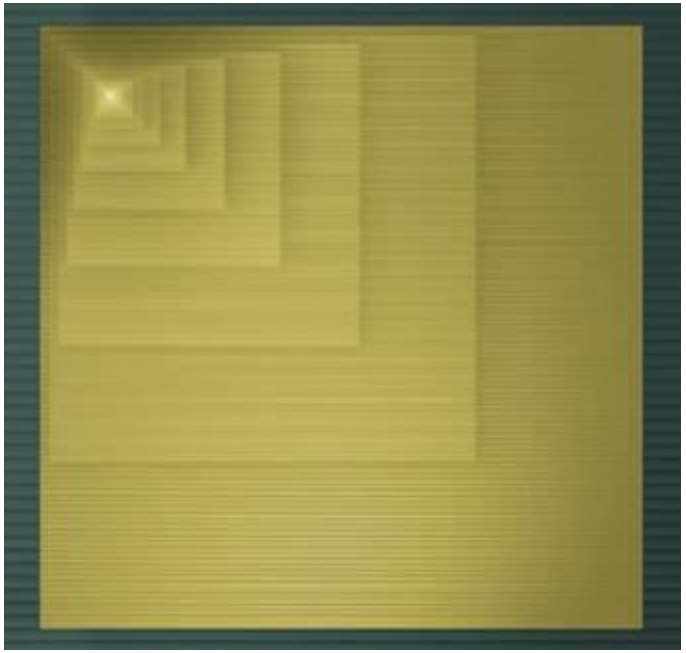


Projeto Lógico Digital Combinacional

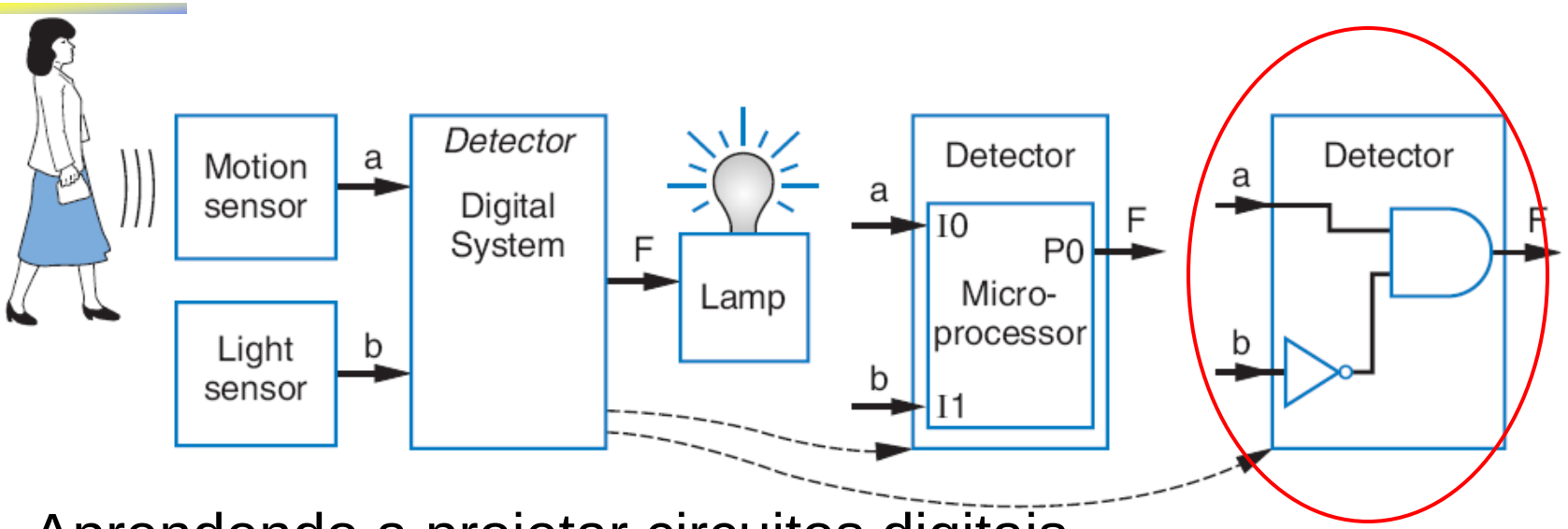
Traduzido, editado revisado e incrementado por Ney Calazans, com base em transparências relativas ao Capítulo 2 do livro de Frank Vahid

(Ver dados e texto de concessão de direitos de publicação abaixo)

Última Atualização – 17/10/2022

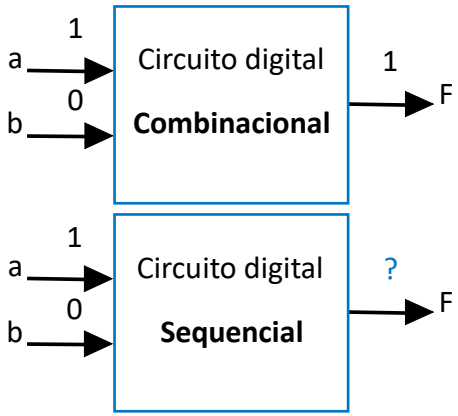
Baseado nos *slides* que acompanham o livro-texto *Digital Design*, 1a. Edição, de Frank Vahid, John Wiley and Sons Publishers, 2007

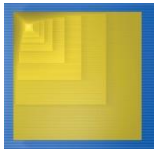
Introdução



Circuito digital

- Aprendendo a projetar circuitos digitais
- Aqui, apenas a forma mais simples de circuitos digitais
  - ***Circuitos Combinacionais***
    - Um circuito digital cujas saídas dependem única e exclusivamente da **combinação** instantânea dos valores das entradas do *circuito*





Circuitos Eletricos e Eletrônicos

Equações de Maxwell

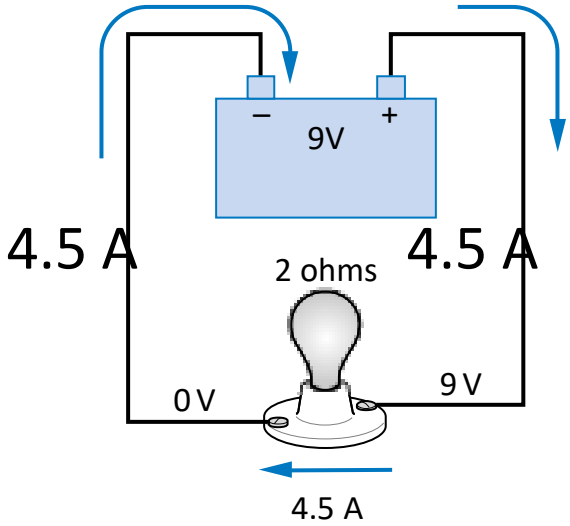
$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0}$$

$$\nabla \cdot \mathbf{B} = 0$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$$

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{j} + \frac{1}{c^2}\frac{\partial \mathbf{E}}{\partial t}$$

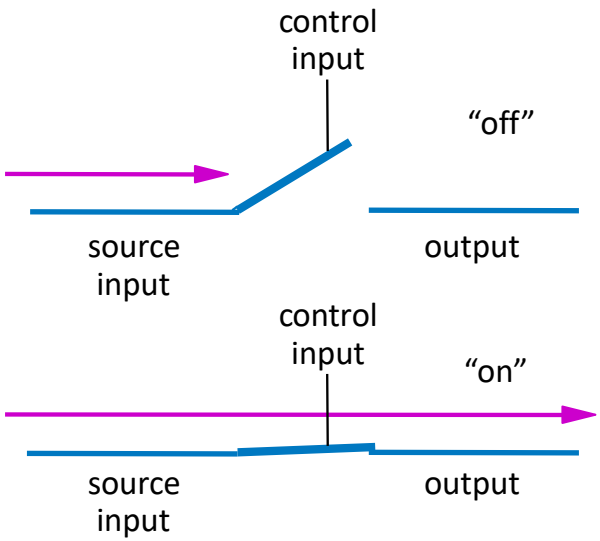
- Circuitos eletrônicos são a base de circuitos digitais
- Circuitos eletrônicos respeitam as leis gerais da Eletricidade
  - Terminologia elétrica → Grandezas principais
    - ***Voltagem (V)***: A diferença de potencial elétrico entre dois pontos ou a capacidade de fornecer energia a cargas elétricas (medida em Joules por Coulomb - J/C, 1J/C denomina-se **1 Volt - V**)
    - ***Corrente (I)***: Fluxo de partículas carregadas na unidades de tempo (medida em Coulombs por segundo - C/s, 1C/s denomina-se **1 Ampère - A**)
    - ***Resistência (R)***: Tendência de um componente (fio, pedaço de metal, madeira etc.) de resistir à passagem de corrente elétrica (medida em **Ohms - Ω**)
    - ***Potência Elétrica (P)***: Consumo instantâneo de energia (medida em Joules por segundo – J/s, 1 J/s denomina-se **Watt - W**)



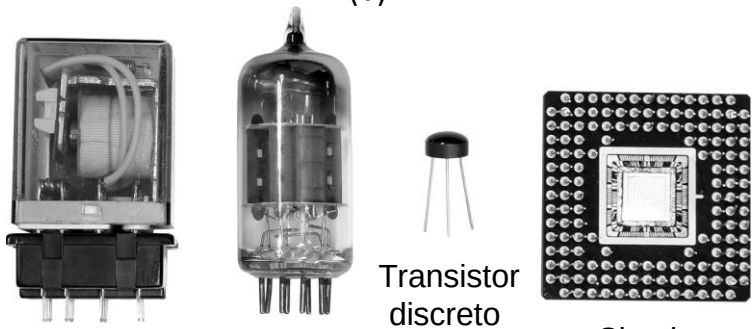
- Principais relações entre grandezas elétricas (muito mais será estudado na disciplina 98D07-06 - Física G. E. III, 3º sem. do curso)
  - V = R * I (Lei de Ohm), P = V * I
  - Lei das malhas (derivada da Lei de Ohm) – A soma das tensões aplicadas sobre os componentes de uma ***malha*** fechada é 0
  - Lei dos Nodos (derivada da Lei de Ohm) – A soma das correntes que entram e saem de um ***nodo*** de um circuito é 0
  - Note-se: A Lei de Ohm deriva de um conjunto de equações muito mais gerais do Eletromagnetismo, as chamadas Equações de Maxwell, equações diferenciais que relacionam campos elétricos e magnéticos na Natureza, sua propagação e mensuração, seja em materiais, seja no vácuo Nelas, tem-se: E e B – os campos elétrico e magnético, respectivamente; ρ e j – a função de carga e a corrente elétrica, respectivamente; ε_0 e μ_0 – as constantes de permitividade e permeabilidade do vácuo, respectivamente; ***c*** é a velocidade da luz no vácuo. Vejam mais detalhes em https://www.cantorsparadise.com/maxwells-equations-7484212839b1

Chaves Elétricas/Eletrônicas

- Uma chave possui três partes
  - Entrada fonte e saída
    - A corrente “quer” fluir da entrada fonte (*source input*) para a saída (*output*)
  - Entrada de controle (*control input*)
    - Voltagem que controla se a corrente flui ou não
- Lembrem-se: corrente só flui se houver um **circuito fechado**!!



(b)

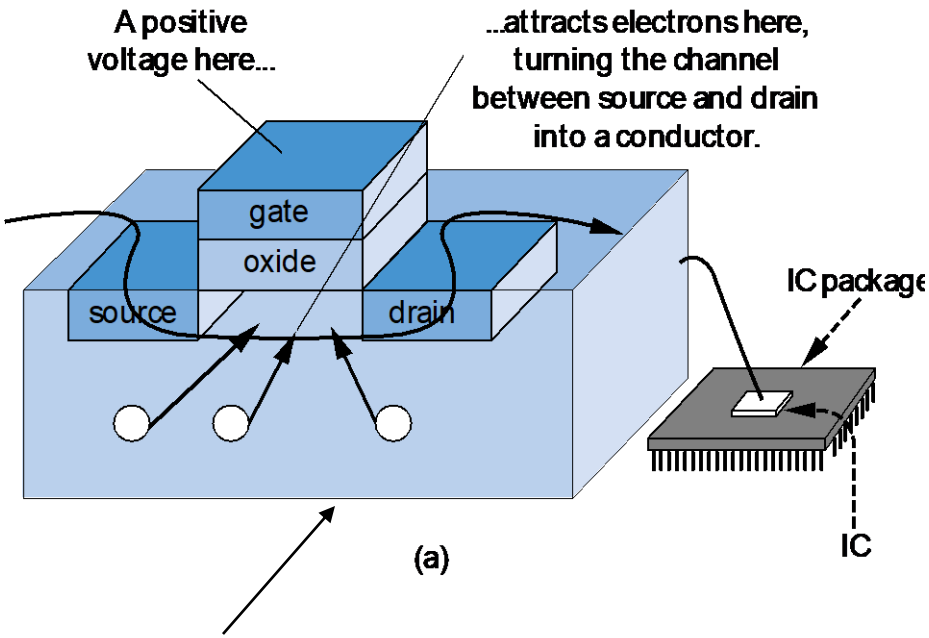


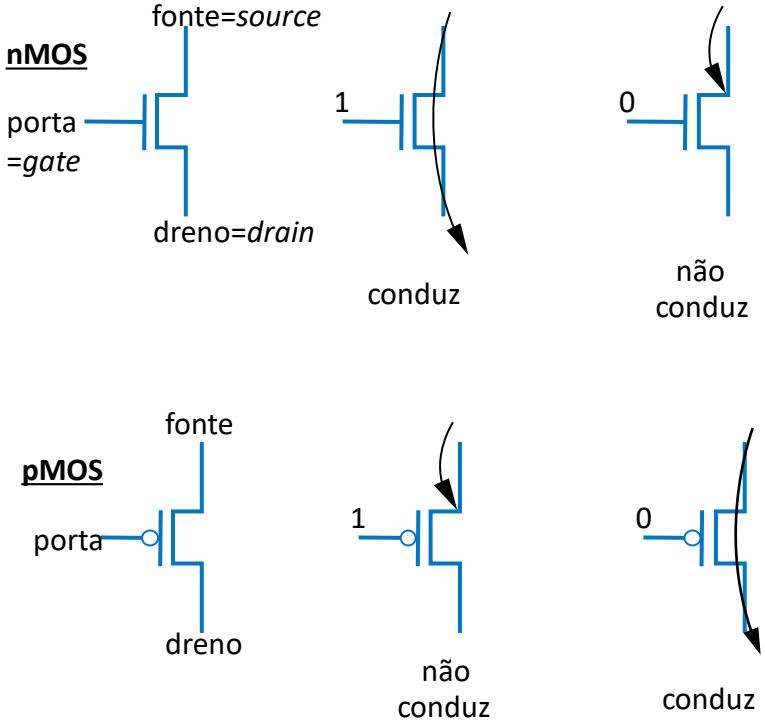
Relé | Válvula eletrônica | Transistor discreto | Circuito Integrado (CI)

Moeda de US$0.25 (em escala)

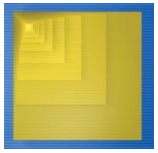
Os Transistores CMOS

- Transistores MOS
  - Componentes fundamentais em circuitos integrados modernos
  - Há dois tipos fundamentais: o transistor nMOS e o transistor pMOS





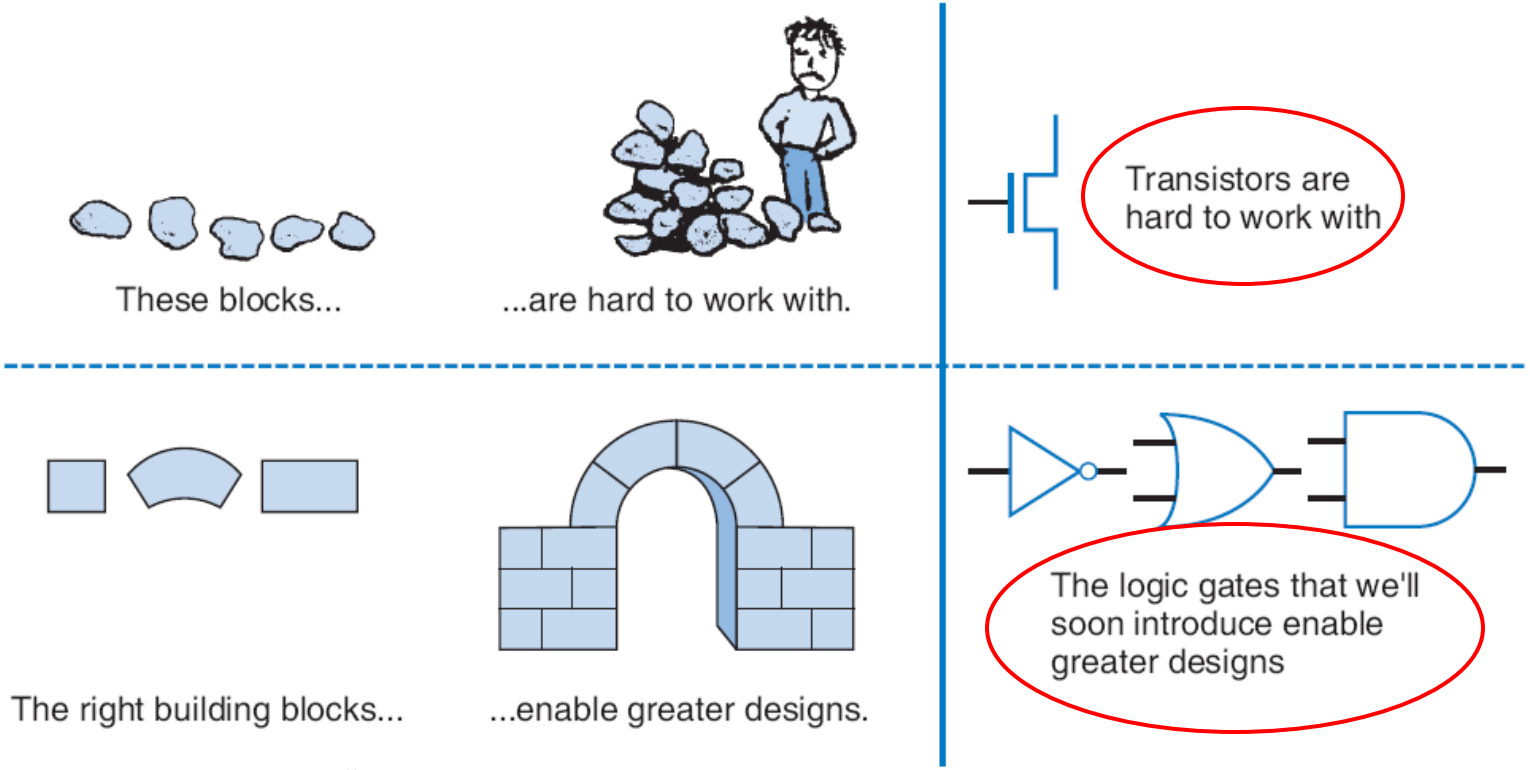
Base (ou substrato) de silício
Não é nem plenamente condutor nem totalmente isolante, é
Semicondutor



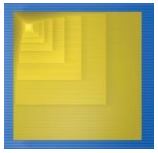
Portas Lógicas Booleanas

Os Blocos Construtivos de Circuitos Digitais

(Pois Chaves Eletrônicas são difíceis de se trabalhar)



- "Portas Lógicas" são blocos construtivos melhores para circuitos digitais que chaves (transistores)
  - Porque?
    (1) São agregados de transistores (blocos de mais alto nível)
    (2) seguem as regras simples da álgebra de chaveamento



Álgebra Booleana e sua Relação com Circuitos Digitais

- ***Álgebra Booleana – Álgebra de Chaveamento***
  - **Variáveis** podem assumir apenas os valores 0 e 1
  - **Operadores** retornam apenas 0 ou 1
  - Operadores fundamentais
    - AND: *a AND b* retorna 1 somente quando ambos a=1 e b=1
    - OR: *a OR b* retorna 1 se uma das duas situações ocorrer (ou ambas): a=1 ou b=1
    - NOT: *NOT a* retorna o inverso de a (1 se a=0, 0 se a=1)

a	b	a AND b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a OR b
0	0	0
0	1	1
1	0	1
1	1	1

a	NOT a
0	1
1	0

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

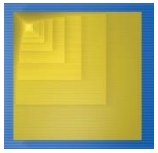
a	b	a NAND b
0	0	1
0	1	1
1	0	1
1	1	0

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0

a	b	NOT b
0	0	1
0	1	0
1	0	1
1	1	0

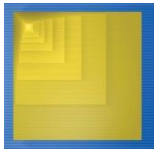
a	b	a NXOR b
0	0	1
0	1	0
1	0	0
1	1	1

Função Degenerada de 2 variáveis

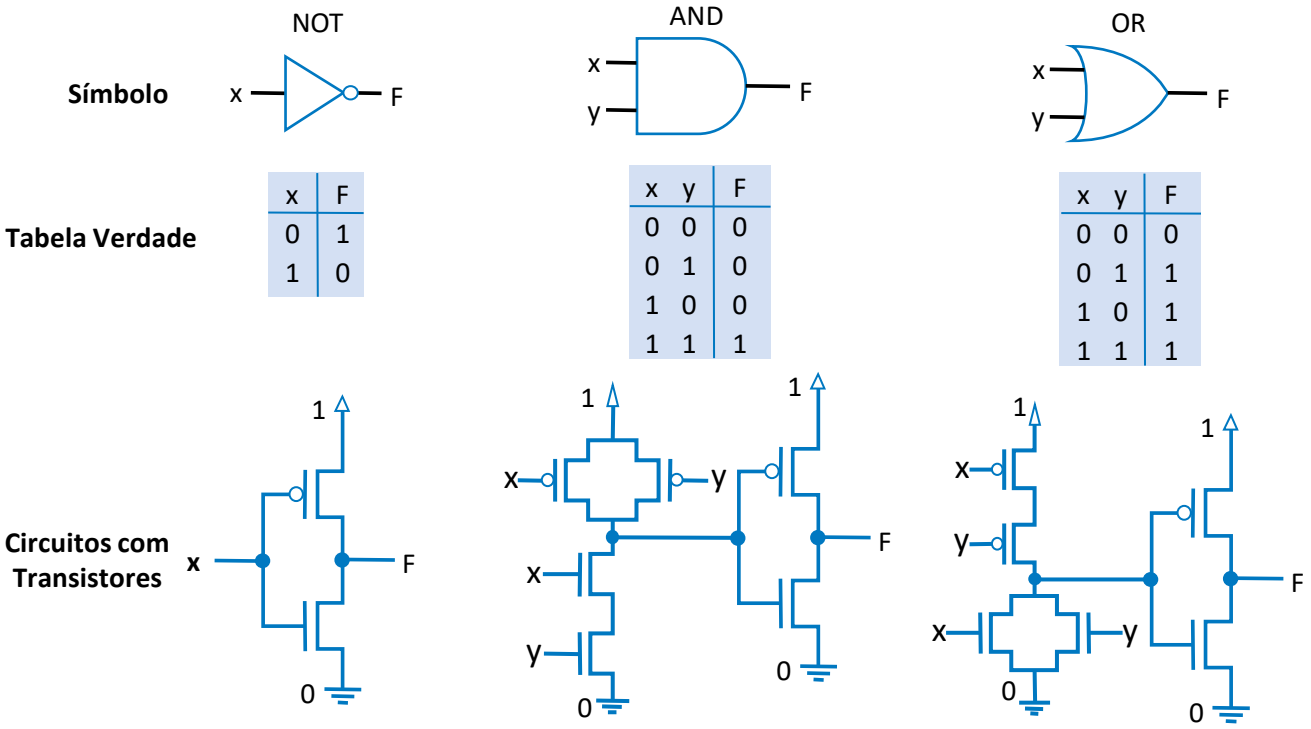


Convertendo Língua Natural para Equações Booleanas

- Converta as seguintes frases para uma equação Booleana
  - Q1. a é 1 e b é 1.
    - Resposta: F = a AND b → Também se pode escrever **a.b** ou **ab**, se não houver risco de ambiguidade
  - Q2. um dos dois, a ou b, é 1.
    - Resposta: F = a OR b → Também se pode escrever **a+b**, se não houver risco de ambiguidade
  - Q3. ambos, a e b não são 0.
    - Respostas
      - (a) Opção 1: F = NOT(a) AND NOT(b)
      - (b) Opção 2 (mesma interpretação que item anterior): F = a NOR b
      - (c) Opção (3) (interpretação de alternativa ambígua): F = a OR b
  - Observação: a muito importante função NOT, pode ser notada por **a'** ou $\bar{a}$

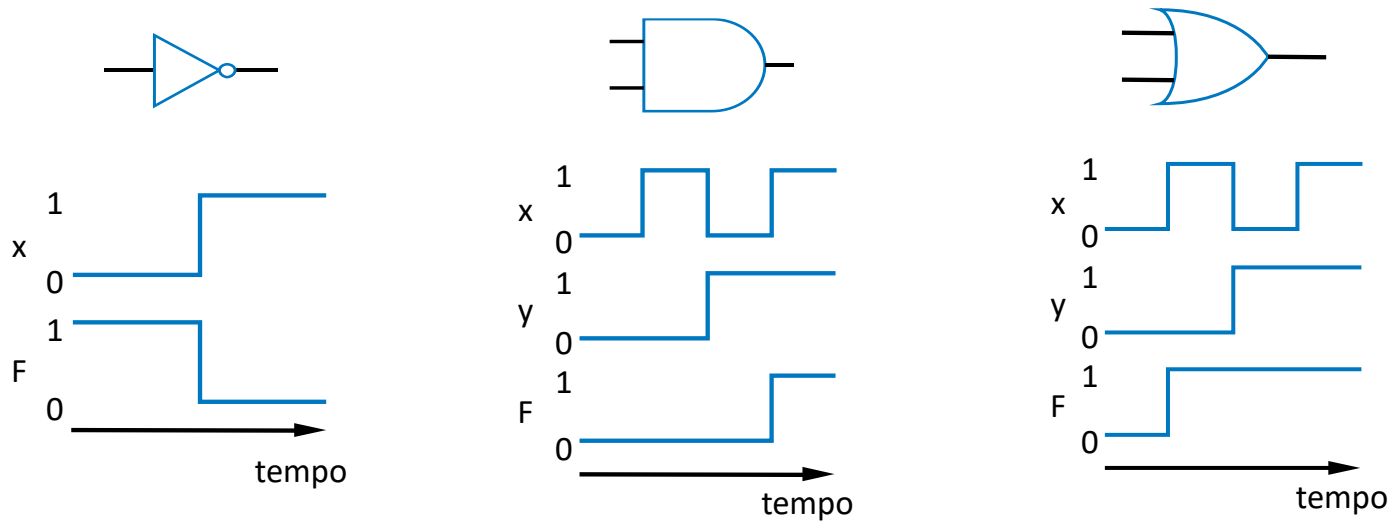


Relacionando Álgebra Booleana com Projeto Digital



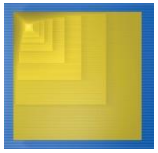
- Implementam operadores Booleanos usando transistores
  - Estas implementações são denominadas portas lógicas (em inglês, ***logic gates***)

Diagramas de Tempo para Portas Lógicas (NOT/OR/AND)



Observação sobre representação de funções Booleanas com equações

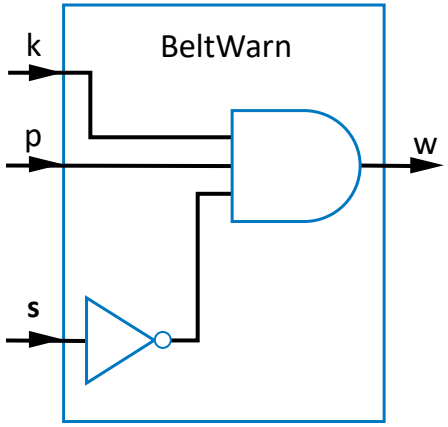
- Existe uma precedência padrão entre operadores NOT, AND e OR
- NOT possui maior precedência, seguido por AND, terminando com OR, que tem a menor precedência.
- Assim a'.b + b.c equivale a ((a').b) + (b.c) → parênteses sempre podem ser usados para explicitar uma precedência diferente do padrão



Exemplo: Sistema de Luz de Aviso de Cinto de Segurança

- Projetando o circuito que controla a luz de alarme do cinto de segurança
- Sensores
  - s=1: cinto afivelado
  - k=1: chave na ignição, virada
  - p=1: pessoa sentada no assento
- Capturando a equação Booleana
  - pessoa sentada, **e** cinto **não** afivelado **e** chave na ignição → luz acende, caso contrário, luz apaga
- Convertendo a equação em um circuito
- Exercício → Assuma agora que o carro tem 5 assentos e que a luz deve acender se qualquer um dos assentos está com pessoa sentada sem afivelar o cinto. Gere o novo circuito de controle que controle o sinal w, que diz se a luz de aviso deve acender ou não

w = p AND NOT(s) AND k



Terminologia da Álgebra Booleana (de Chaveamento)

- Equação Exemplo: **F(a,b,c) = a'.b.c + a.b.c' + a.b + c**
- ***Variáveis***
  - Representa um valor que a cada instante é 0 ou 1
  - Três variáveis: a, b, e c
- ***Literal***
  - Consiste em uma forma alternativa de considerar uma variável, seja na forma afirmada ou na forma negada (complementada)
    - Um literal é de fato uma forma de representar funções simples, que são usadas na composição de outras funções mais complexas. Um literal na forma afirmada é a função identidade, enquanto que um literal complementado é a função inversor.
  - Há nove literais na expressão acima: a', b, c, a, b, c', a, b, c
- ***Termo Produto***
  - É um produto (Booleano) de literais (literais combinados via operador AND)
  - Há quatro termos produto na expressão acima da **F(a,b,c)**: a'.b.c, a.b.c', a.b, c
- ***Soma-de-produtos***
  - Denomina-se assim uma equação escrita como um OR de termos produto
  - A equação acima está escrita nesta forma. "G(a,b,c,d) = (a+b).c + d" não está

Propriedades da Álgebra Booleana (de Chaveamento)

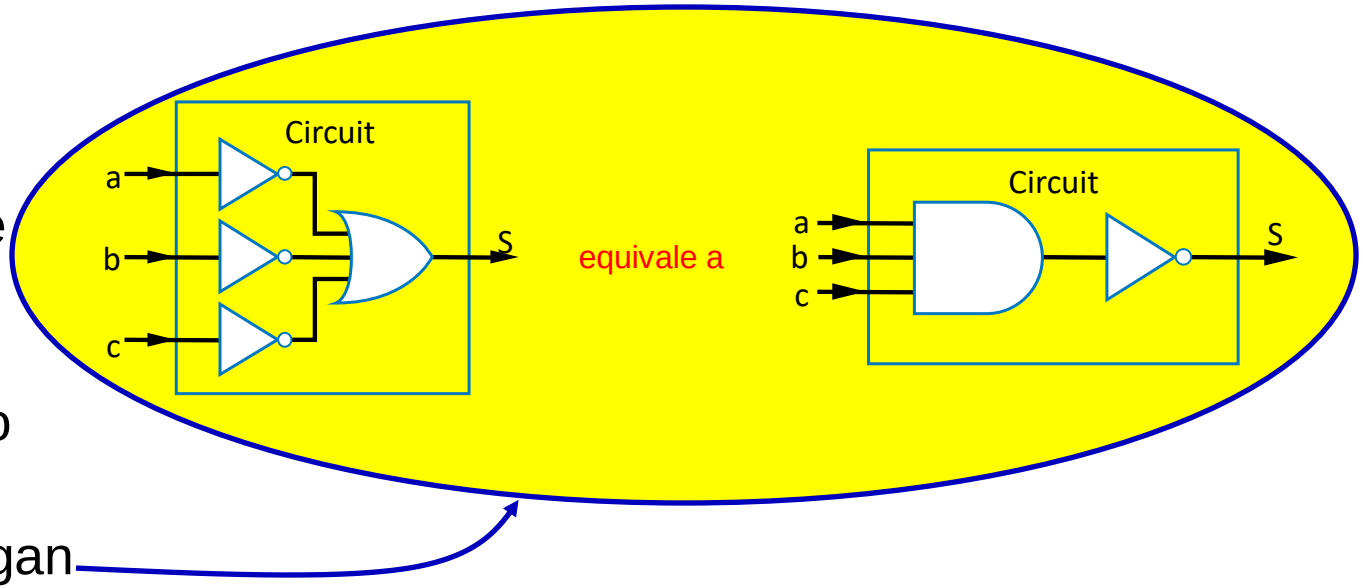
- Comutativa
  - a + b = b + a
  - a . b = b . a
- Distributiva
  - a . (b + c) = (a . b) + (a . c)
  - a + (b . c) = (a + b) . (a + c)
    - (parece antinatural, mas vale!)
- Associativa
  - (a + b) + c = a + (b + c)
  - (a . b) . c = a . (b . c)
- Identidade
  - 0 + a = a + 0 = a
  - 1 . a = a . 1 = a
- Complemento
  - a + a' = 1
  - a . a' = 0
- Para provar, basta testar exaustivamente todas as possibilidades

Exemplos de uso das propriedades

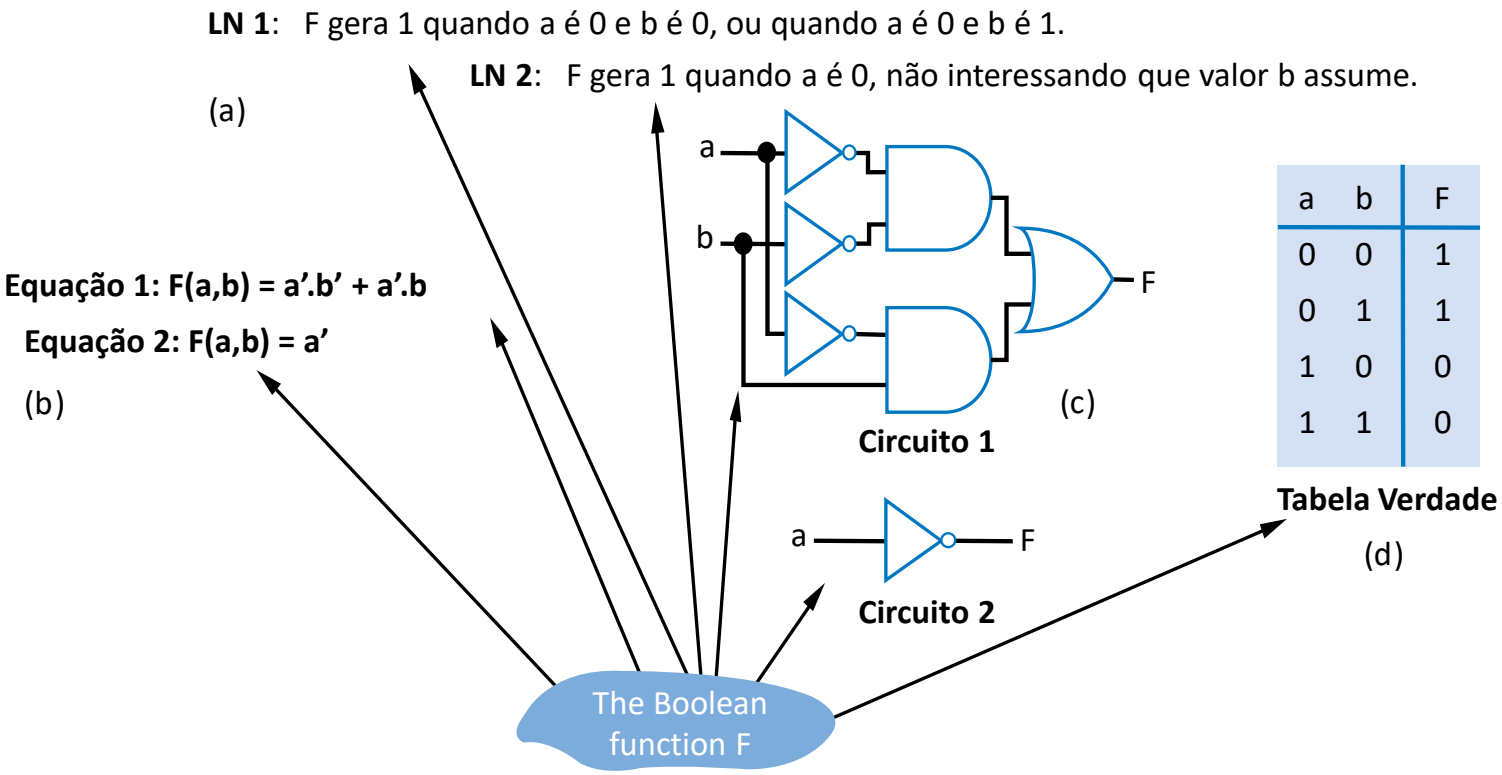
- Mostre que abc + abc' = ab
  - Primeiro, use a propriedade distributiva, obtendo
    - abc + abc' = ab(c+c')
  - A seguir, use a propriedade Complemento e obtenha
    - c+c' é 1: ab(c+c') = ab(1)
  - Finalmente, use a propriedade Identidade e obtenha o que se quer
    - ab(1) = ab.1 = ab

Álgebra de Chaveamento: Propriedades Adicionais

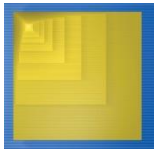
- Elemento Nulo
  - a + 1 = 1
  - a . 0 = 0
- Lei Idempotente
  - a + a = a
  - a . a = a
- Lei da Involução
  - (a')' = a
- Leis de De Morgan
  - (a + b)' = a'.b'
  - (a.b)' = a' + b'
  - Muito útil e importante!
- Para provar, de novo, uma forma é testar, **exaustivamente**, usando tabelas verdade



Representações de Funções de Chaveamento



- Uma função de chaveamento pode ser representada de diferentes formas
  - Acima mostram-se 7 representações de uma função (*degenerada*) F(a,b) usando 4 métodos diferentes: Linguagem Natural, Equações, Circuitos, e Tabelas Verdade



A Representação Tabular de Funções Booleanas

- Define-se o valor da F para cada valor possível de combinação de valores de entradas
  - Função de 2 entradas : 4 linhas
  - Função de 3 entradas : 8 linhas
  - Função de 4 entradas : 16 linhas
  - etc.
- Exercício: Use uma tabela verdade para definir a função F(a,b,c) que vale 1 quando abc, convertido de binário para decimal, é 5 ou um número maior ou igual a 5

a	b	F
0	0	
0	1	
1	0	
1	1	

(a)

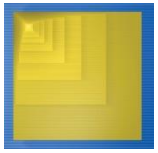
a	b	c	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

(b)

a	b	c	d	F
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

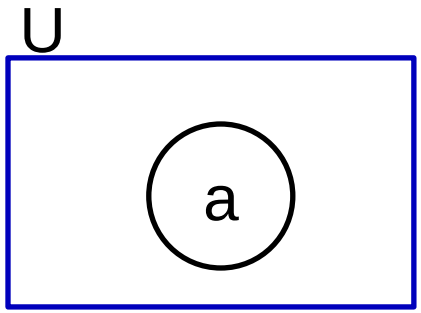
(c)

a	b	c	F
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**0**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

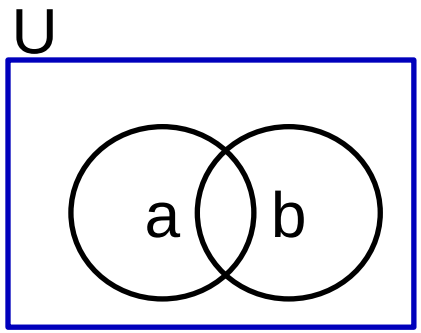


Diagramas de Venn

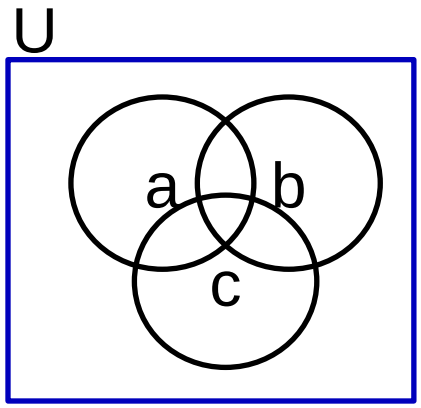
- Diagramas de Venn: (mais) uma forma gráfica bem simples de visualizar funções de chaveamento
  - Útil para funções de até 3-4 variáveis
  - Ideia básica → variável é um círculo, que tem lado de dentro e lado de fora. No lado de dentro, a variável vale 1, no lado de fora ela vale 0
  - Círculos de diferentes variáveis intersectam
  - O conjunto Universo U é um retângulo envolvendo todos os círculos
  - Regiões hachuradas podem representar funções



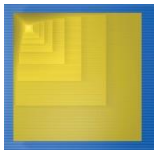
Venn de 1 variável



Venn de 2 variáveis



Venn de 3 variáveis



Representação Padronizada: Tabela Verdade

- Como determinar se 2 funções de chaveamento são a mesma?
  - Usar métodos algébricos
  - Mas se falharmos, isto prova que as funções *não são* iguais?
    - **Não!**
- Solução: Converter funções para tabelas verdade
  - Só existe UMA tabela verdade para uma dada função
    - Representação ***Padronizada*** – se para uma dada função, só existe uma forma padronizada

Q: Determine se F=ab+a' é a mesma função que G=a'b'+a'b+ab, convertendo cada uma delas para uma tabela verdade

F = ab + a'

a	b	F
0	0	1
0	1	1
1	0	0
1	1	1

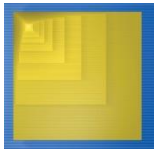
G = a'b' + a'b + ab

a	b	G
0	0	1
0	1	1
1	0	0
1	1	1

Mesma

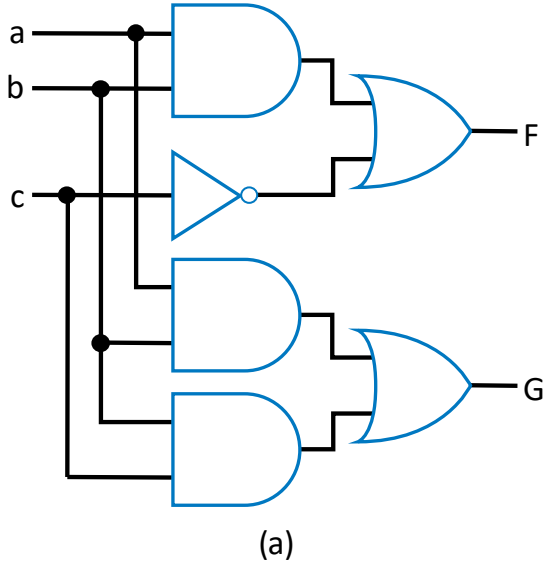
Forma Canônica - Soma de Mintermos

- Tabelas verdade (TVs) podem ser muito grandes para funções de muitas entradas (e.g. 10 ou 20 variáveis)
- Pode-se usar uma forma padronizada de equação, ao invés de TVs
  - Existem formas conhecidas como ***formas canônicas***
  - Em álgebra de chaveamento: cria-se uma soma de mintermos
    - ***Mintermo***: um termo produto com cada literal da função aparecendo exatamente uma vez, na forma afirmada ou complementada
    - Simplesmente gere equação que contenha a soma de termos produto
    - Em seguida, expandir cada termo até que todos sejam mintermos

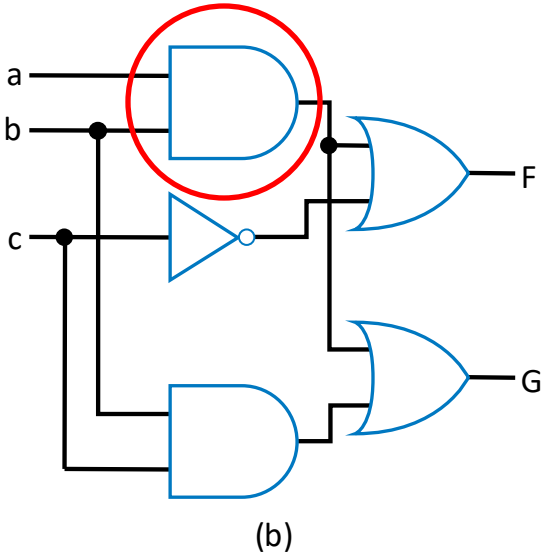


Circuitos com Múltiplas Saídas

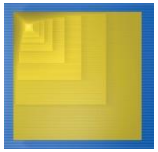
- Muitos circuitos de chaveamento têm mais de uma saída
- Pode-se computar cada saída separadamente, mas também se pode compartilhar portas lógicas (funções parciais)
- Exemplo: F = a.b + c', G = a.b + b.c



Opção 1: Com circuitos separados



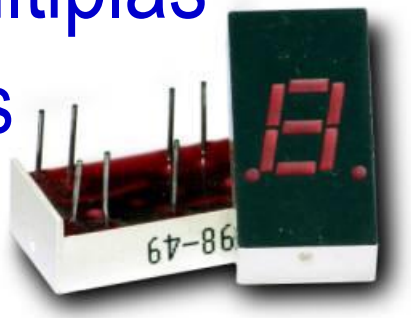
Opção 2: Com portas compartilhadas

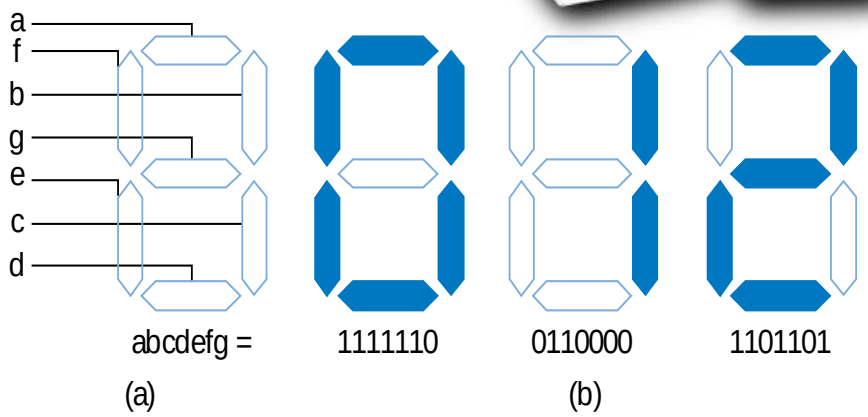


Exemplo de Circuitos com Saídas Múltiplas
Conversor BCD para 7 Segmentos

TABLE 2-4 4-bit binary number to seven-segment display truth table

w	x	y	z	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

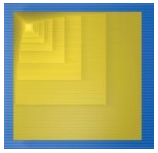




a = w'x'y'z' + w'x'yz' + w'x'yz + w'xy'z + w'xyz' + w'xyz + wx'y'z' + wx'y'z

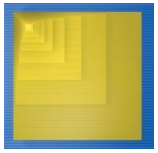
b = w'x'y'z' + w'x'y'z + w'x'yz' + w'x'yz + w'xy'z' + w'xyz + wx'y'z' + wx'y'z

Exercício: Crie as funções para as outras 6 saídas, as funções **c** a **g**



Processo de Projeto para Lógica Combinacional

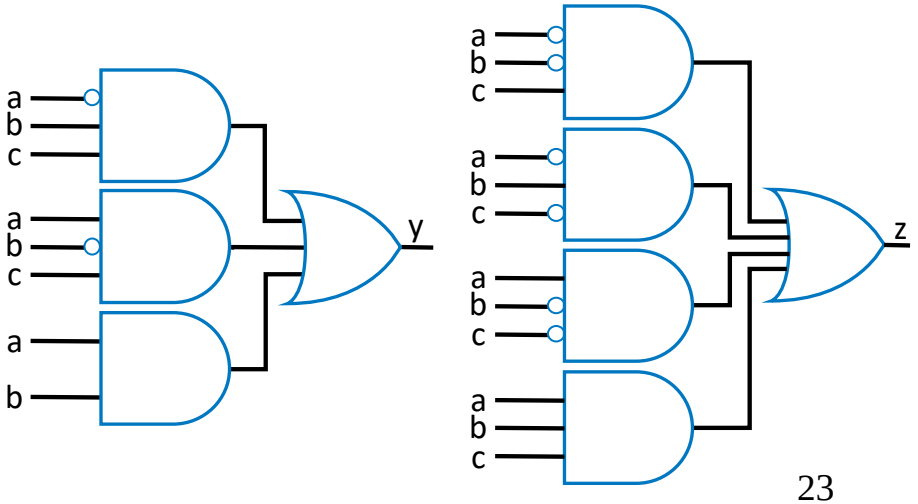
	Passo	Descrição
Passo 1	**Capture** a função	Crie uma tabela verdade ou equações, ***usando a forma mais natural para o problema dado***, descrevendo o comportamento desejado da lógica combinacional
Passo 2	**Converta** para equações	Este passo só é necessário se você capturou a função usando uma tabela verdade e não equações. Crie uma equação para cada saída, ***OReando*** todos os mintermos para aquela saída. Simplifique as equações se desejar
Passo 3	**Implemente** como um circuito baseado em portas	Para cada saída, crie um circuito que corresponde às equações da saída. (Compartilhar portas entre saídas múltiplas está ok, mas é opcional)



Exemplo: Contador do Número de 1s

- Problema: Gerar uma saída em dois bits (y e z) que dê a contagem do número de 1s sobre 3 entradas
  - 010 → 01 101 → 10 000 → 00
  - **Passo 1: Capture** a função
    - Tabela verdade ou equação?
      - Tabela verdade é direto
  - **Passo 2: Converta** para equações
    - y = a'bc + ab'c + abc' + abc
    - z = a'b'c + a'bc' + ab'c' + abc
  - **Passo 3: Implemente** na forma de um circuito com portas lógicas

Inputs			(# of 1s)	Outputs	
a	b	c		y	z
0	0	0	(0)	0	0
0	0	1	(1)	0	1
0	1	0	(1)	0	1
0	1	1	(2)	1	0
1	0	0	(1)	0	1
1	0	1	(2)	1	0
1	1	0	(2)	1	0
1	1	1	(3)	1	1



Um Conjunto Básico de Portas Lógicas (1)

Símbolo

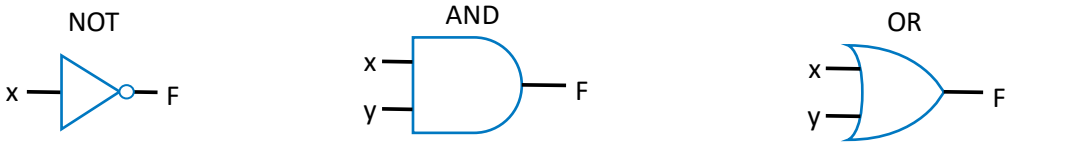


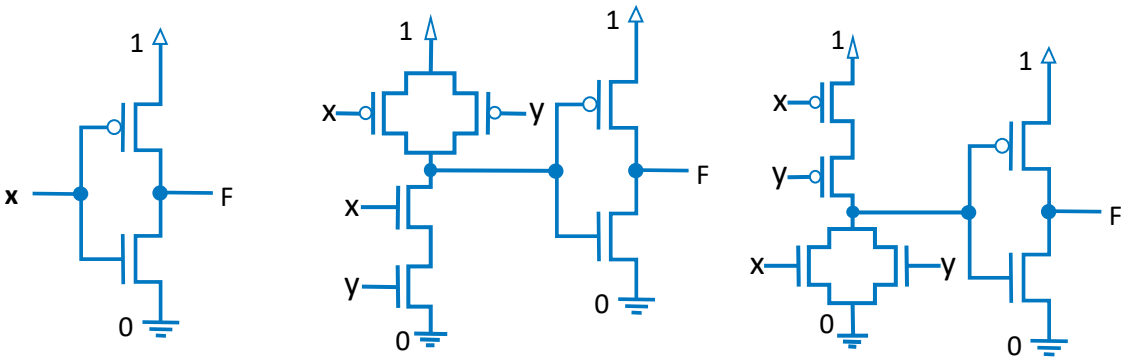
Tabela Verdade

x	F
0	1
1	0

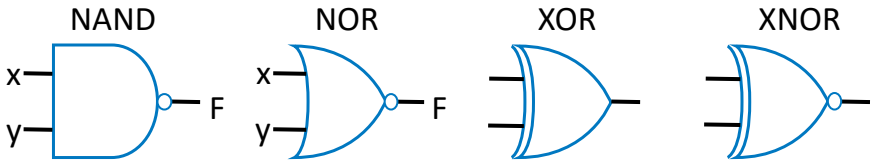
x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

Circuitos com Transistores



Um Conjunto Básico de Portas Lógicas (2)

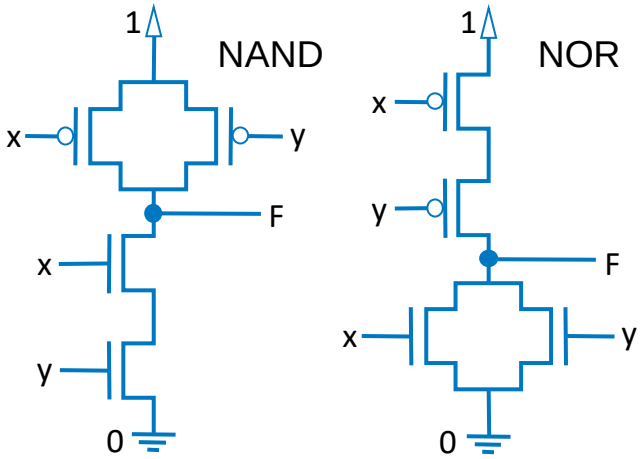


x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

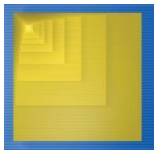
x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

x	y	F
0	0	1
0	1	0
1	0	0
1	1	1



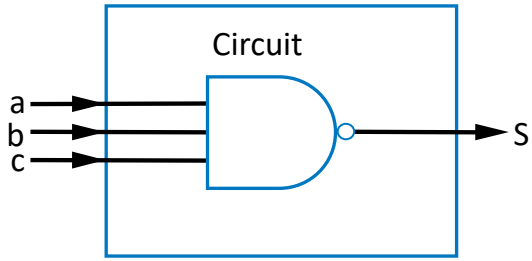
- NAND: Inverso do AND (“NOT AND”)
- NOR: Inverso do OR (“NOT OR”)
- XOR: Se número de 1s na entrada é ímpar, a saída é 1, senão é 0, ou seja XOR é um calculador de paridade ímpar
- XNOR: Inverso do XOR (“NOT XOR”) , ou seja XNOR é um calculador de paridade ímpar. Também chamado de NXOR

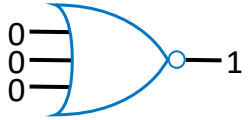
- Uma porta AND poderia ser criada usando uma NAND com pontos de alimentação invertidos, mas não é...
- Porque? nMOS conduz 0s bem, mas não 1s, e vice-versa para pMOS (motivos serão vistos na disciplina 4456C-04 Microeletrônica, 7º sem. do curso)
- Assim:
  - AND em CMOS: Uma NAND seguida de NOT
  - OR em CMOS: NOR seguida de NOT
- NANDs/NORs e lógica negada são mais baratas que ANDs/Ors, logo mais comuns...

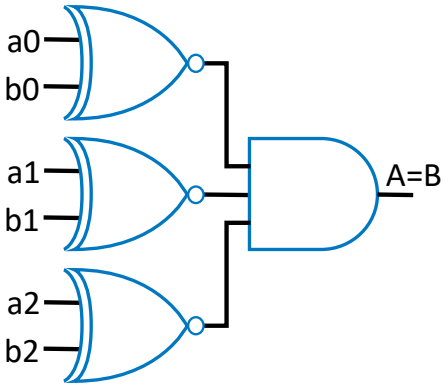


Portas Lógicas: Exemplos de Utilidade

- S = (abc)’
- Detector de “tudo em 0”
  - Use NOR
- Detector de igualdade (opera bit a bit)
  - Use XNOR
- Detector de número ímpar de 1s
  - Use XOR
- XORs/XNORs Úteis para gerar “paridades” (detectar erros, etc.)

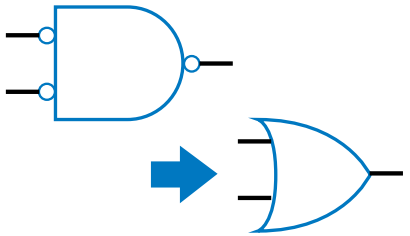






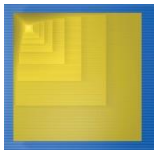
Suficiência de NAND e de NOR

- Qualquer função Booleana pode ser implementada *usando apenas portas NAND.* Porque?
  - Preciso de AND, OR e NOT
    - NOT: NAND com 2 entradas ligadas entre si
    - AND: NAND seguida de NOT
    - OR: NAND precedida por NOTs



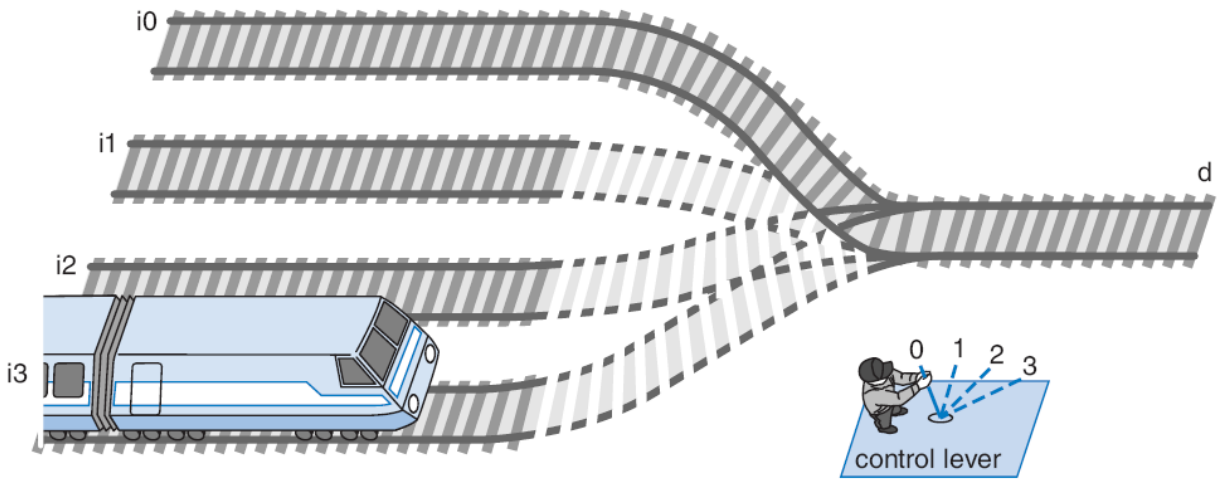
- A mesma afirmativa **vale** para a porta NOR

- O mesmo **não vale** para AND ou OR, ou NOT…

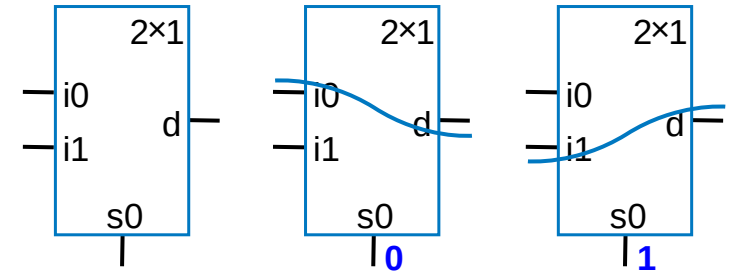


Multiplexadores (Muxes) ou Seletores

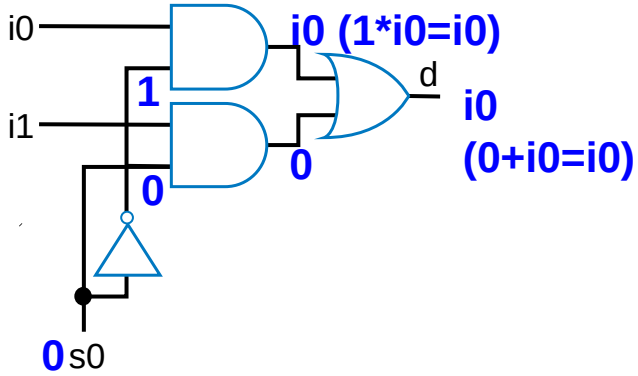
- Mux: Outro bloco combinacional bastante usado
  - Uma definição → Roteia uma de suas N **entradas de dados** para a sua (única) saída, baseado no valor binário nas suas **entradas de seleção**
    - Mux de 4 entradas ou Mux 4:1 → precisa de 2 entradas de seleção para indicar qual entrada rotear para a saída
    - Mux de 8 entradas ou Mux 8:1 → 3 entradas de seleção
    - Mux de N entradas → $\log_2(N)$ entradas de seleção
  - Funciona como chave para selecionar trilhos em uma estrada de ferro

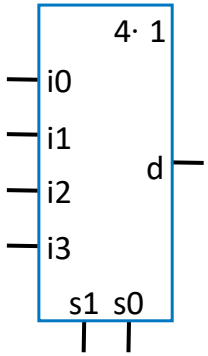


Projeto Interno de Muxes

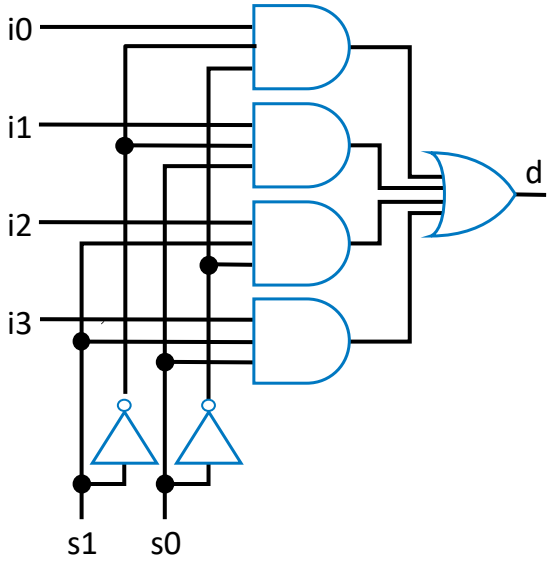


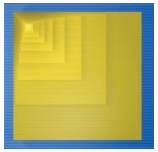
Mux 2x1



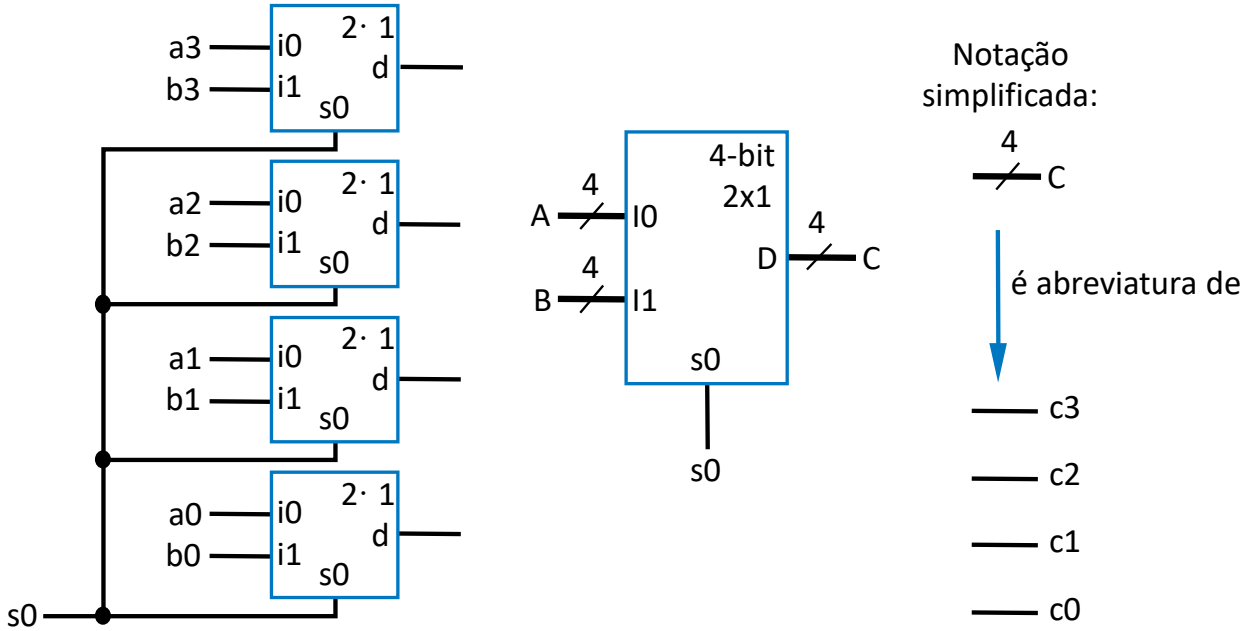


Mux 4x1



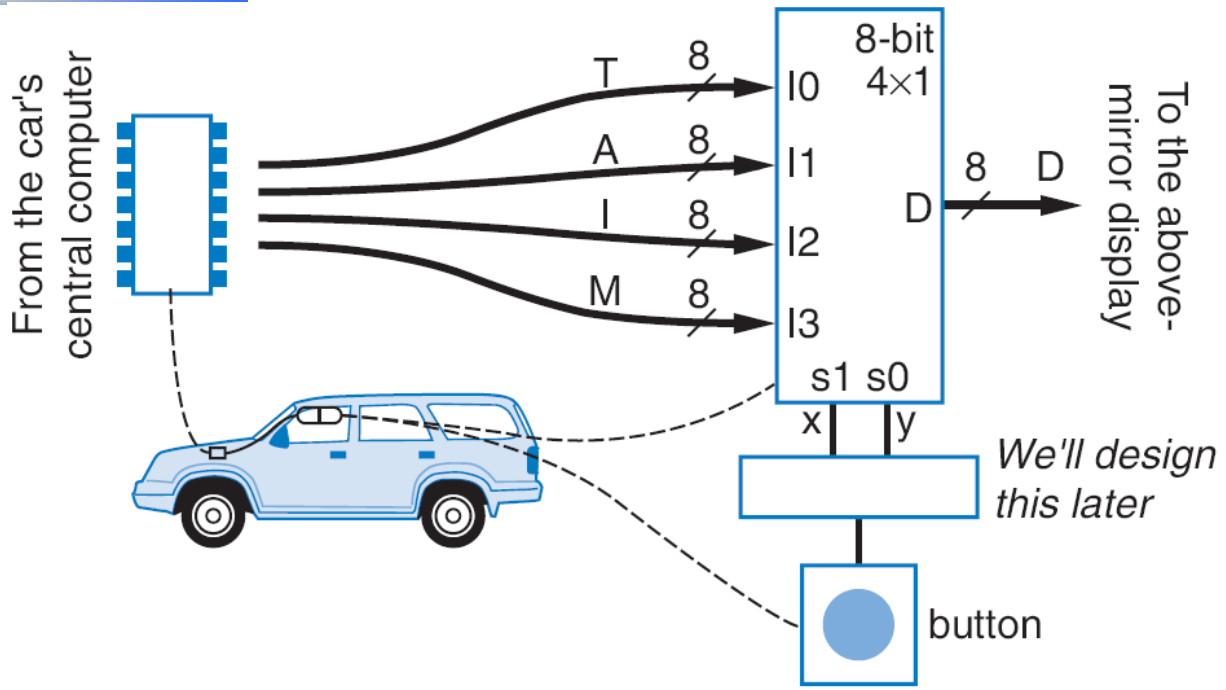


Combinando Muxes → Saídas de N bits

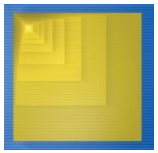


- Ex: Duas entradas de 4 bits, A (`a3 a2 a1 a0`), e B (`b3 b2 b1 b0`)
  - Um mux 2:1 de 4 bits (são só 4 muxes 2:1 que compartilham a entrada de seleção) pode selecionar entre A ou B para colocar na saída

Exemplo de Uso de um Mux de N bits

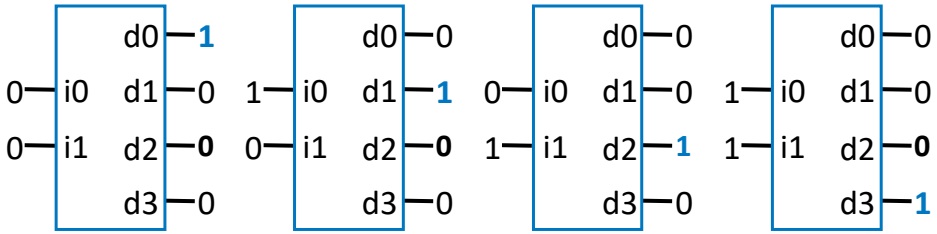


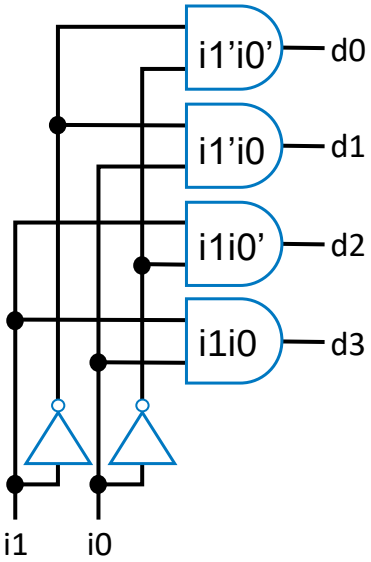
- Imagine que se pode mostrar quatro coisas diferentes na mesma posição do painel acima do retrovisor de um automóvel
  - Temperatura (T), Média de Consumo, em km/l (A), Consumo Instantâneo (I), e Km restantes com o tanque atual (M) – cada dado é representado em 8 bits
  - Escolha o que mostrar no painel usando os fios de controle ***x*** e ***y***
  - Use um mux 4:1 de 8 bits

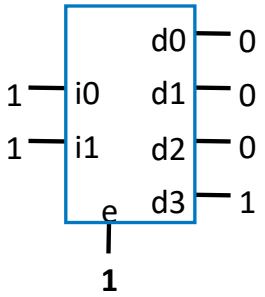


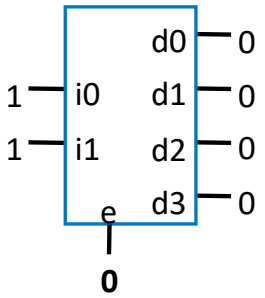
Decodificadores (Decods)

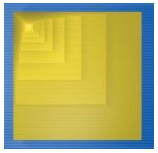
- **Decodificador**: Bloco lógico muito usado → pode ser montado com portas lógicas
  - Uma definição: Converte um número binário para uma saída única em 1
- Decodificador de 2 entradas, ou decodificador 2 para 4: há 4 números binários possíveis na sua entrada
  - Possui 4 saídas, 1 para cada número possível na entrada
- Projeto interno
  - Uma porta AND para cada saída, que detecta uma combinação de entrada específica
- Decodificador com entrada de habilitação (*enable e*)
  - Todas as saídas em 0 se ***e***=0
  - Comportamento descrito, se ***e***=1
- Decodificador de n entradas ou decodificador n para 2^n: tem 2^n saídas



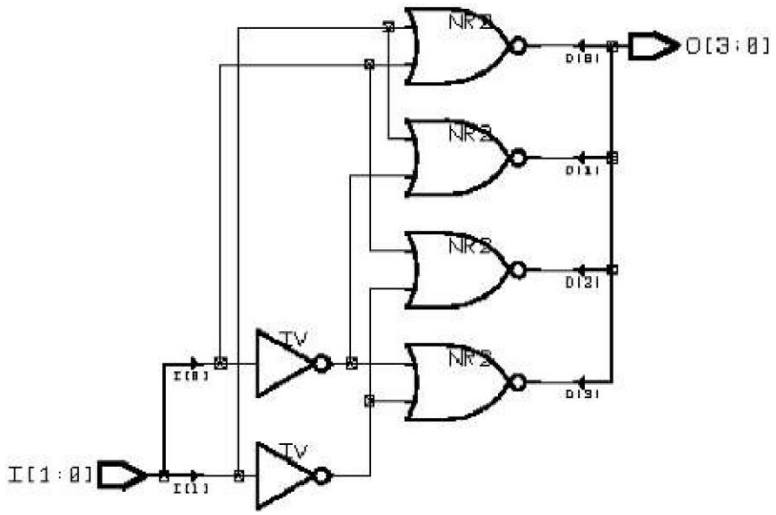


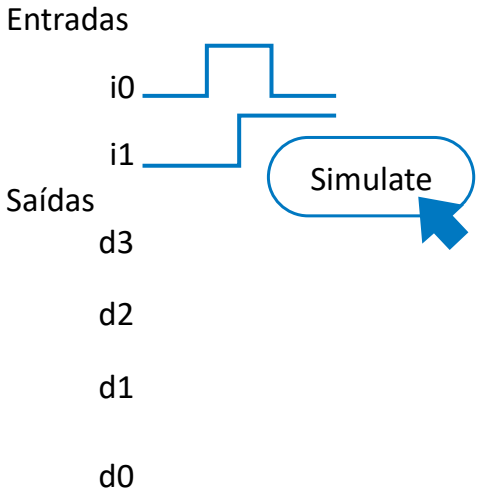


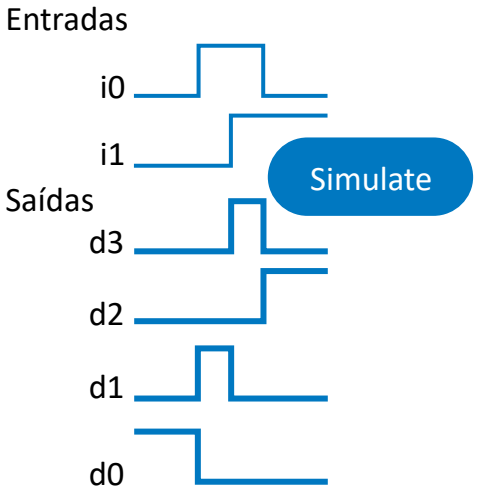




Considerações Adicionais:
Captura de Esquemáticos e Simulação



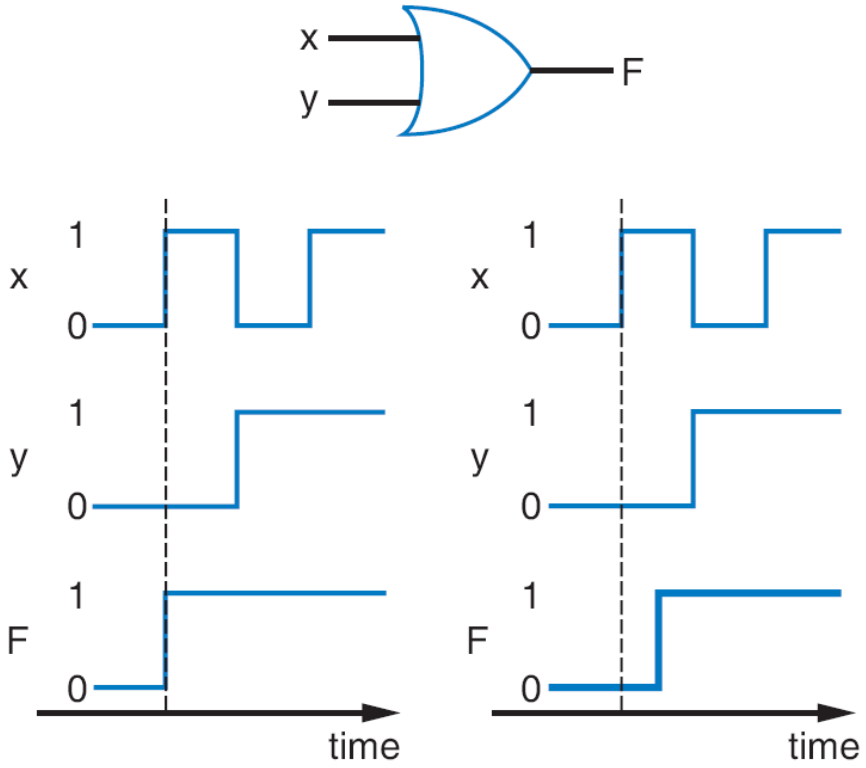




- **Captura de Esquemáticos**
  - Ferramenta de computador que permite ao usuário (projetista de hardware) capturar graficamente um circuito lógico
- **Simulador**
  - Ferramenta de computador para mostrar que saídas um circuito projetado fornece para entradas dadas
    - Saídas são normalmente mostradas como uma forma de onda temporal (em inglês, **waveform**)

Considerações Adicionais:

Atrasos → comportamento não-Ideal de Portas Lógicas



- Portas lógicas reais sempre possuem atraso (Leis da termodinâmica mandam)
  - Saídas não mudam imediatamente após as entradas mudarem

Sumário

- Circuitos Combinacionais
  - Circuitos cujas saídas são função única e exclusivamente do valor instantâneo das entradas
    - Cuidado, circuitos combinacionais não existem!!! Hein??? Veja slide anterior
- Chaves: componentes básicos de qualquer circuito digital
- Portas lógicas Booleanas: AND, OR, NOT → blocos construtivos melhores (mais fáceis de usar) que chaves eletrônicas
  - Mas lembrem-se, portas são montadas com chaves eletrônicas (chaves eletrônicas=transistores)
  - Habilitam o uso da álgebra Booleana para projetar circuitos
- Álgebra de Chaveamento: usa variáveis/operadores que lidam com 0/1 ou verdadeiro/falso, etc.
- Representações de funções de chaveamento
  - Existem várias e se pode usar o que se quiser, dependendo da conveniência
  - Se pode converter entre diferentes representações de uma mesma função (portas, TVs, diagramas de Venn etc.)
- O processo de projeto combinacional: Traduz de equações (ou TVs) para circuitos, usando passos bem definidos
- Mais portas (além de AND, OR, NOT): NAND, NOR, XOR, XNOR também são úteis
- Multiplexadores e decodificadores: São blocos construtivos adicionais, úteis em geral