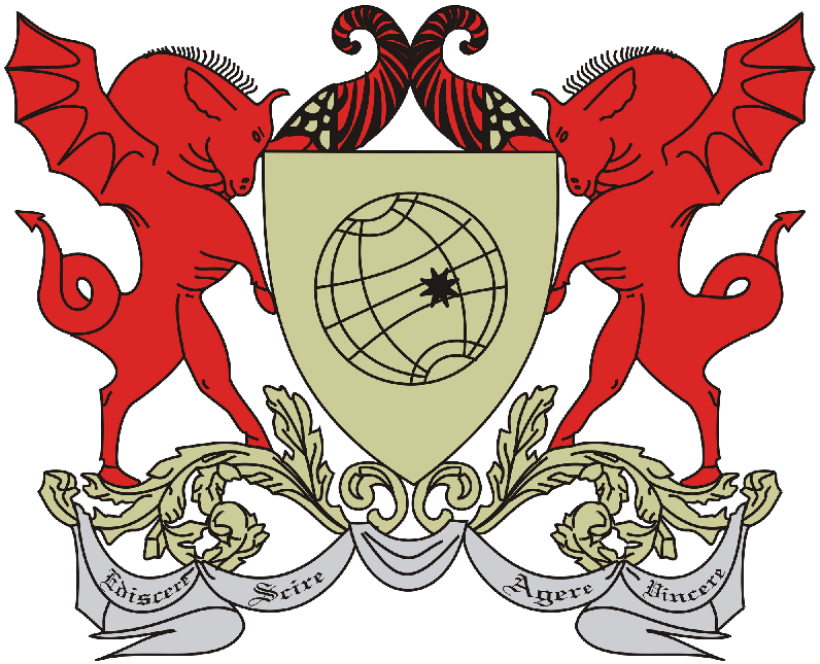


CCF 251 – Introdução aos Sistemas Lógicos

Aula 03 – Trabalhando com lógica combinacional

Prof. José Augusto Nacif – jnacif@ufv.br

Trabalhando com lógica combinacional

- Simplificação
  - Simplificação dois níveis
  - Explorando don’t cares (X)
  - Algoritmo para simplificação
- Implementação lógica
  - Lógica de dois níveis e forma canônica realizadas com NANDs e NORs
  - Lógica multi níveis, conversão entre ANDs e ORs
- Comportamento de tempo
- Linguagem de descrição de Hardware

Exemplo de projeto: Comparador dois bits

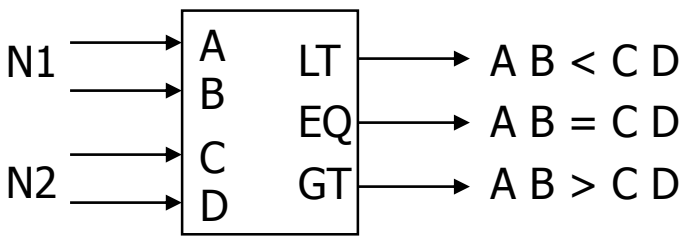
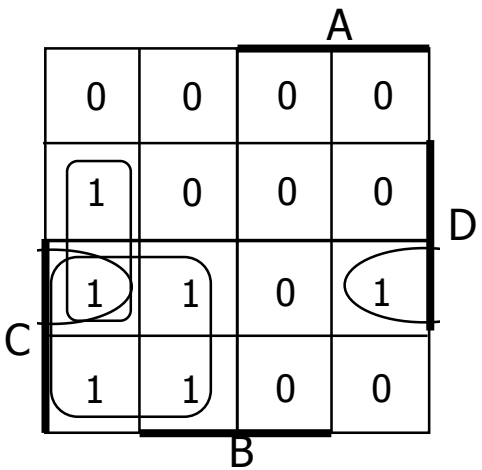


Diagrama de bloco
e
Tabela verdade

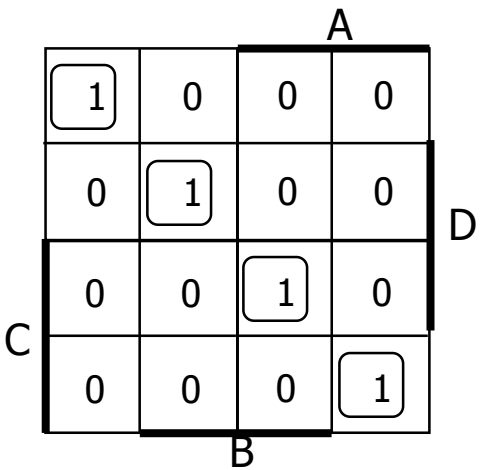
A	B	C	D	LT	EQ	GT
0	0	0	0	0	1	0
	0	1	1	0	0	
	1	0	1	0	0	
	1	1	1	0	0	
0	1	0	0	0	0	1
	0	1	0	1	0	
	1	0	1	0	0	
	1	1	1	0	0	
1	0	0	0	0	0	1
	0	1	0	0	1	
	1	0	0	1	0	
	1	1	1	0	0	
1	1	0	0	0	0	1
	0	1	0	0	1	
	1	0	0	0	1	
	1	1	0	1	0	

Nós precisaremos de um mapa de karnaugh com 4 variáveis para cada uma das 3 funções de saída

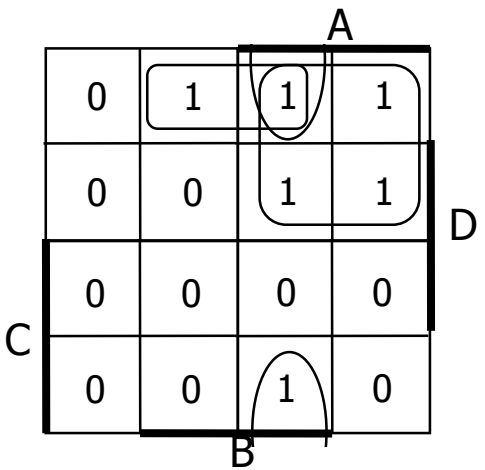
Exemplo de projeto: Comparador dois bits



K-map para LT



K-map para EQ



K-map para GT

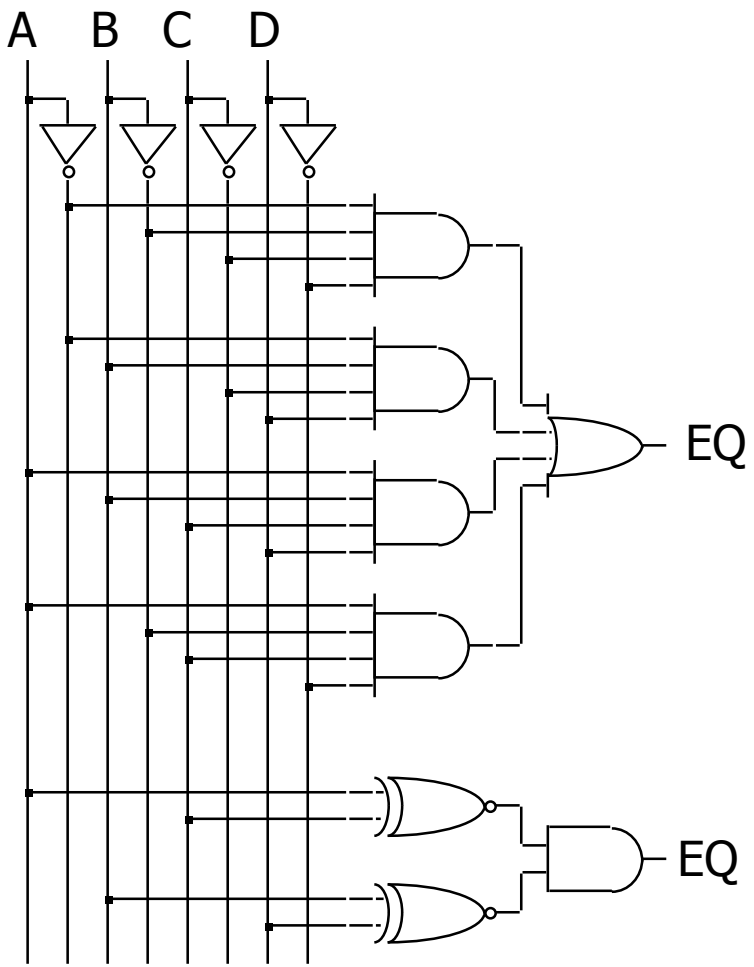
LT = A' B' D + A' C + B' C D

EQ = A' B' C' D' + A' B C' D + A B C D + A B' C D' = (A xnor C) • (B xnor D)

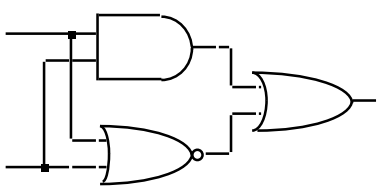
GT = B C' D' + A C' + A B D'

LT e GT são similares (flip A/C e B/D)

Exemplo de projeto: Comparador dois bits (cont'd)



Duas alternativas de implementação de EQ, com e sem XOR



XNOR é implementado com no mínimo 2 portas simples

Exemplo de projeto: Multiplicador 2x2 bits

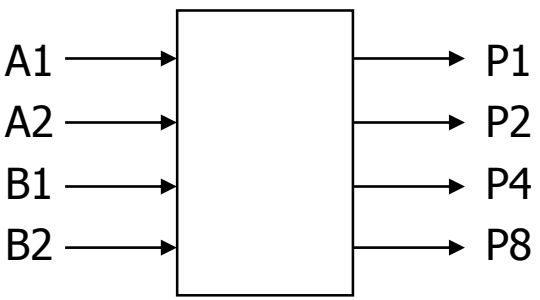
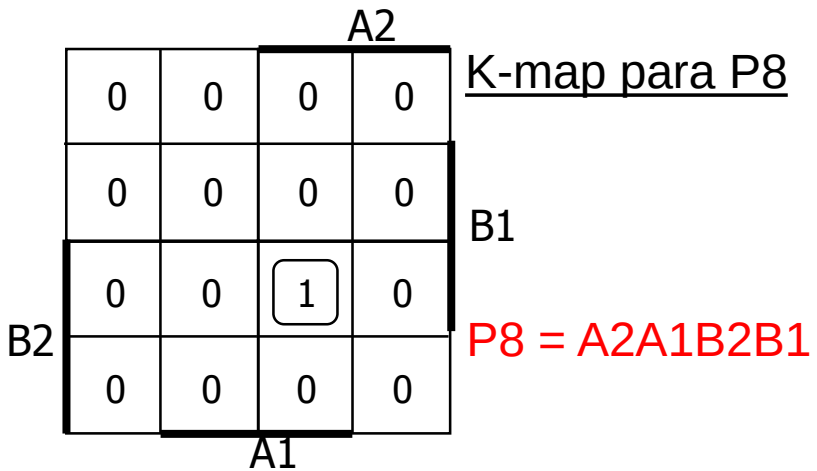


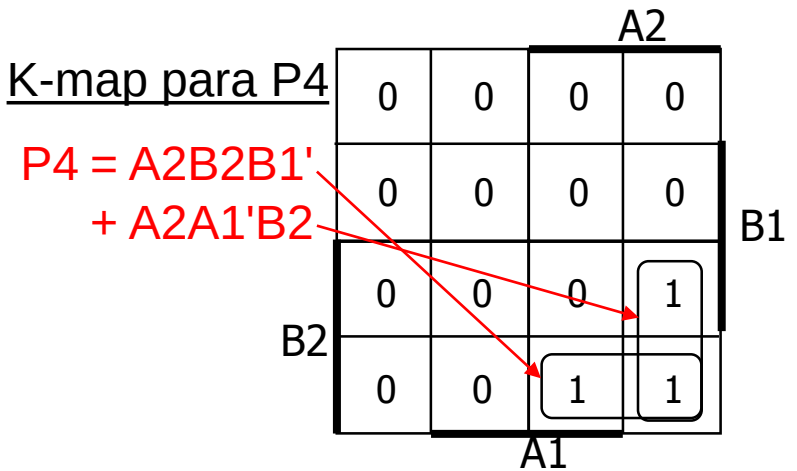
Diagrama de bloco
e
Tabela verdade

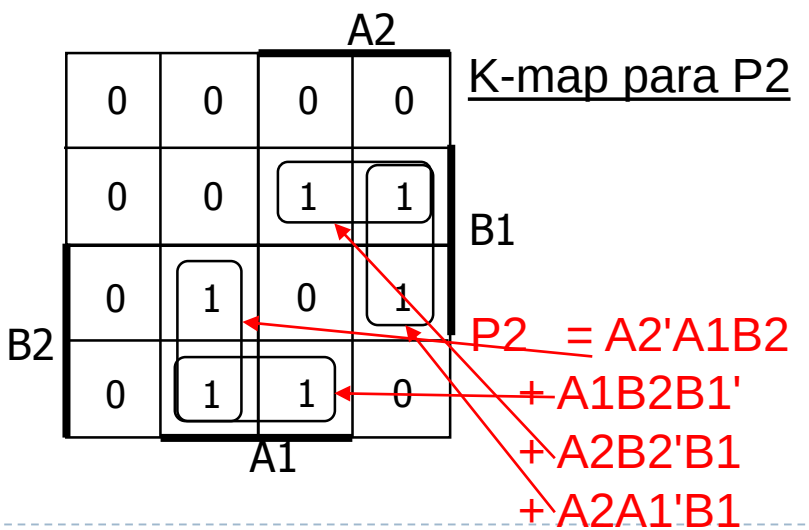
A2	A1	B2	B1	P8	P4	P2	P1
0	0	0	0	0	0	0	0
	0	1	0	0	0	0	
	1	0	0	0	0	0	
	1	1	0	0	0	0	
0	1	0	0	0	0	0	0
	0	1	0	0	0	1	
	1	0	0	0	1	0	
	1	1	0	0	1	1	
1	0	0	0	0	0	0	0
	0	1	0	0	1	0	
	1	0	0	1	0	0	
	1	1	0	1	1	0	
1	1	0	0	0	0	0	0
	0	1	0	0	1	1	
	1	0	0	1	1	0	
	1	1	1	0	0	1	

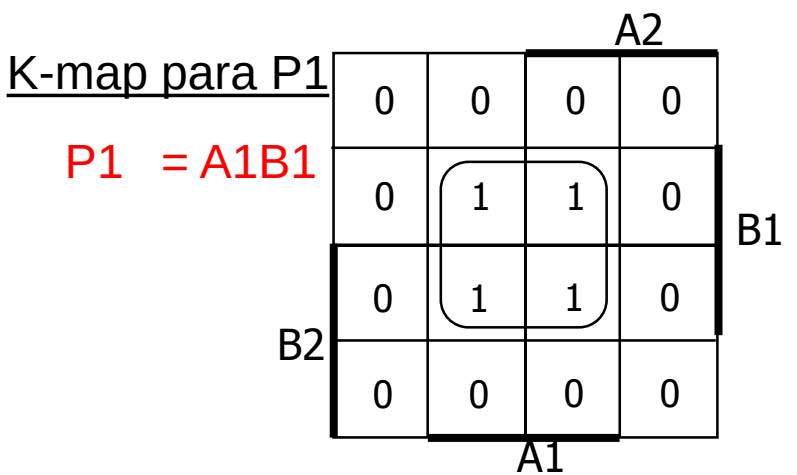
K-map com 4 variáveis
para cada uma das 4 funções de saída

Exemplo de projeto: Multiplicador 2x2 bits









Exemplo projeto: BCD incrementado por 1

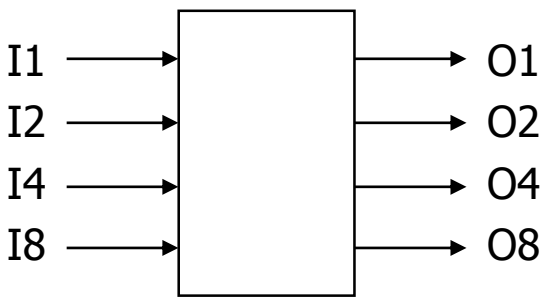
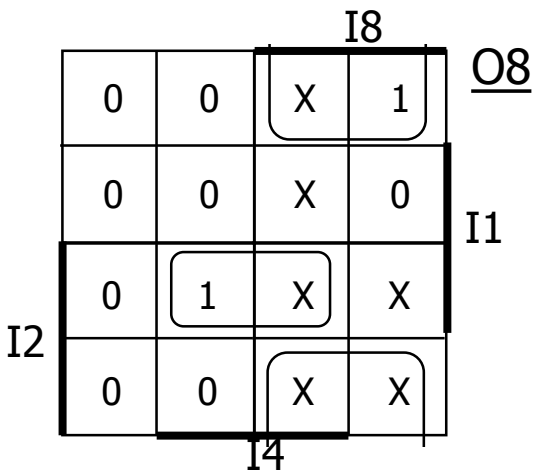


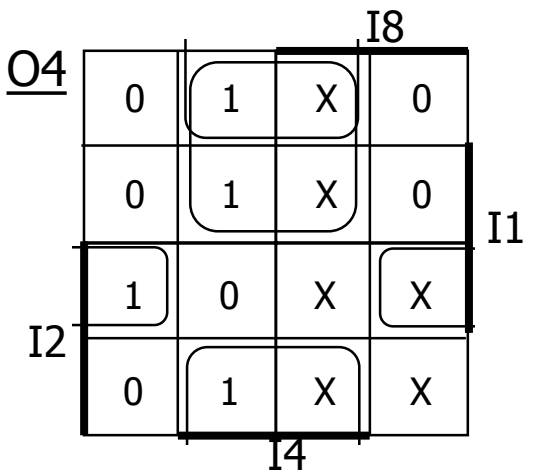
Diagrama de bloco
e
Tabela verdade

I8	I4	I2	I1	O8	O4	O2	O1
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

K-map com 4 variáveis
para cada uma das 4 funções de saída

Exemplo projeto: BCD incrementado por 1



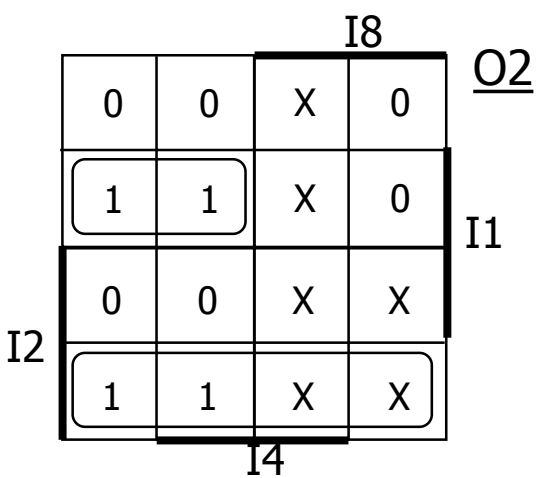


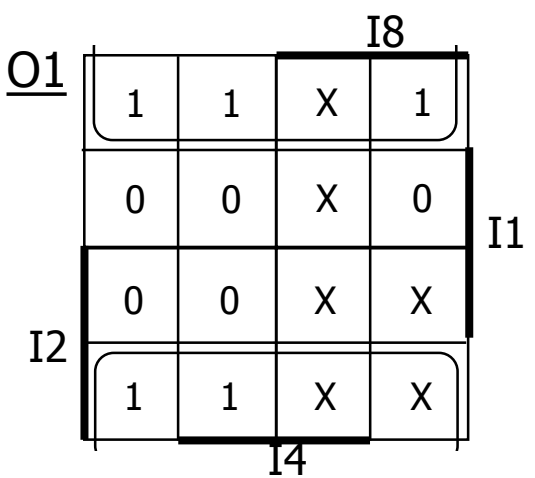
O8 = I4 I2 I1 + I8 I1'

O4 = I4 I2' + I4 I1' + I4' I2 I1

O2 = I8' I2' I1 + I2 I1'

O1 = I1'

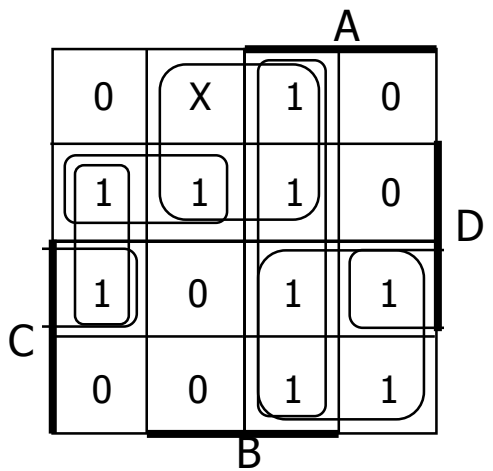




Definição de termos para simplicação dois níveis

- Implícito
  - único elemento de ON-set ou DC-set ou qualquer grupo destes elementos que possam ser combinados para formar um subcubo
- Implícito primordial
  - Implícito que não pode ser combinado com outro para formar um grande subcubo
- Implícito primordial essencial
  - Implícito primordial é essencial se sozinho abranger um elemento ON-set
  - Participará de todas coberturas possíveis de ON-set
  - DC-set é usado para formar implícitos primordiais, mas não para realizar implícitos essenciais
- Objetivo:
  - Aumentar implícitos dentro de implícitos primordiais (minimizar literais por termo)
  - Abranger os ON-set com menos implícitos primordiais possíveis (minimizar o número de termos de produtos)

Exemplos para ilustrar termos



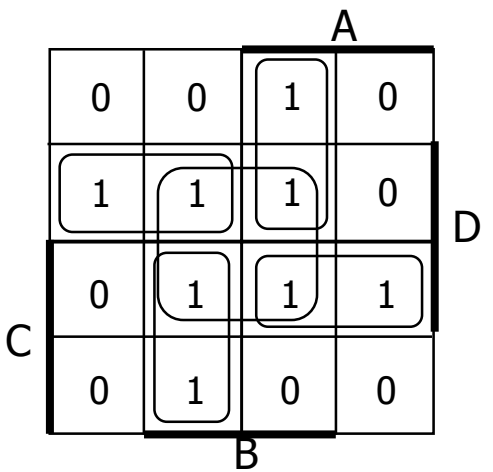
6 implícitos primordiais:
A'B'D, BC', AC, A'C'D, AB, B'CD

essencial

Cobertura mínima: AC + BC' + A'B'D

5 implícitos primordiais:
BD, ABC', ACD, A'BC, A'C'D

essencial

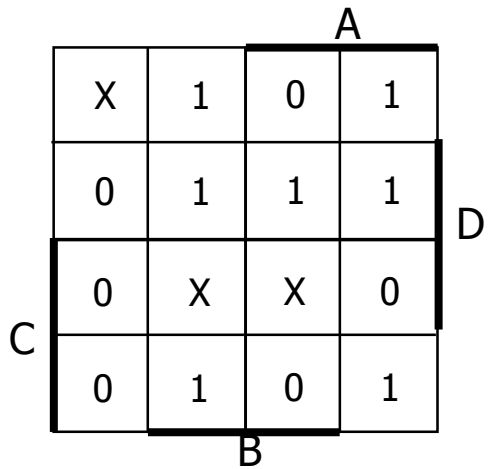


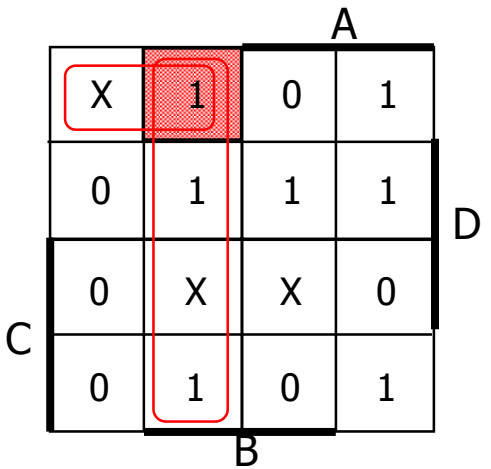
Cobertura mínima: 4 implícitos essenciais

Algoritmo para simplificação dois níveis

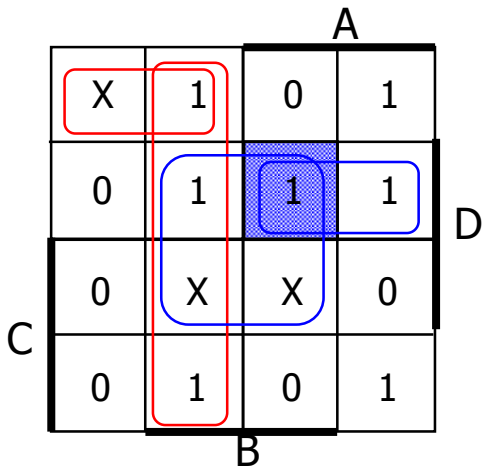
- Algoritmo: expressão mínima de soma de produtos a partir do mapa de karnaugh
  - Etapa 1: Escolher um elemento de ON-set
  - Etapa 2: Encontrar agrupamento “máximo” de 1s e Xs adjacente para o elemento
    - Considerar a linha acima/abaixo, coluna esquerda/direita, e cantos adjacentes
    - Isto forma implícitos primordiais (número de elementos sempre elevado a 2)
  - Repetir Etapa 1 e 2 para encontrar todos os implícitos primordiais
  - Etapa 3: visitar novamente os 1s no k-map
    - Se coberto por um implícito primordial único, é essencial, e participa na cobertura final
    - 1s cobertos por implícitos primordiais essenciais não precisam ser revisados
  - Etapa 4: Se existir 1s não cobertos por implícitos primordiais essenciais
    - Selecionar o menor número de implícitos primordiais que abrange a permanência de 1s

Algoritmo para simplificação dois níveis (exemplo)

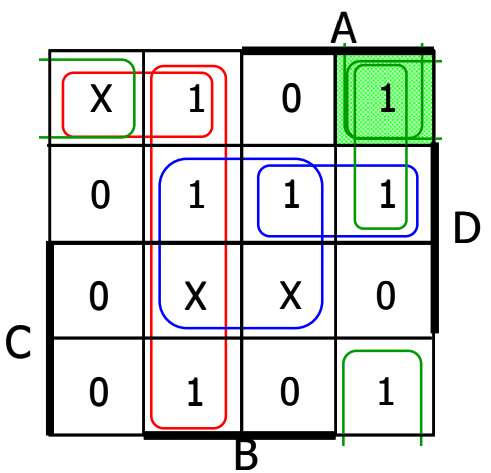




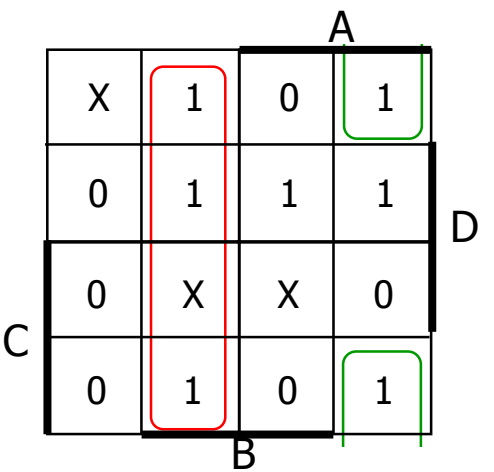
2 primordiais próximo de A'BC'D'



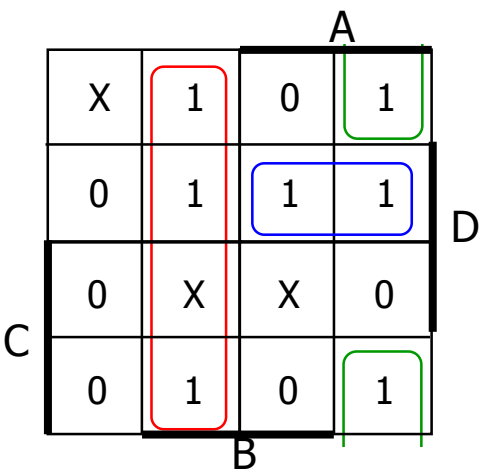
2 primordiais próximo de ABC'D



3 primordiais próximo de AB'C'D'



2 primordiais essenciais



cobertura mínima (3 primordiais)

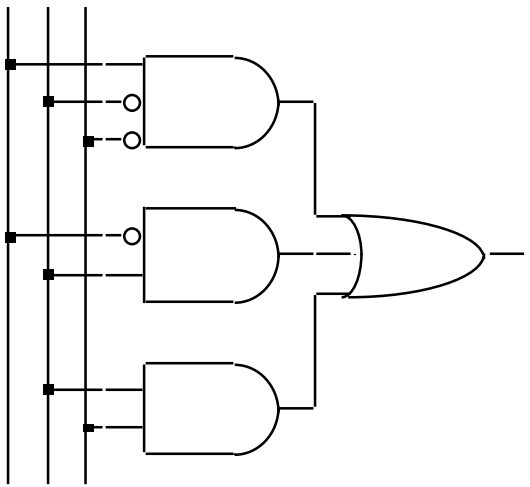
Atividade

- Liste todos os implícitos primordiais para o seguinte K-map:

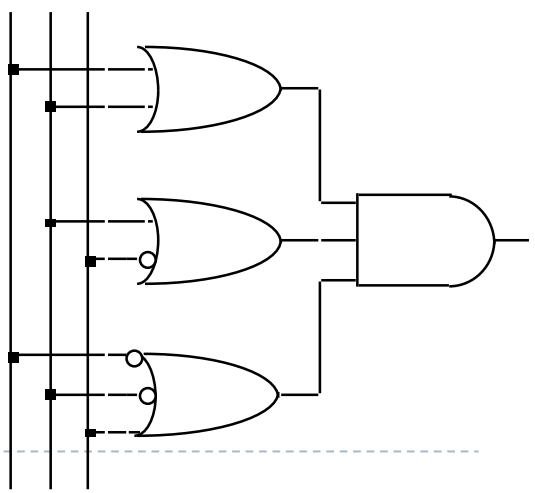
- Quais são implícitos primordiais essenciais?
- Qual é a cobertura mínima?

Implementações de lógica dois níveis

- Soma de produtos
  - Portas AND a partir dos termos do produto (mintermos)
  - Porta OR para formar a soma

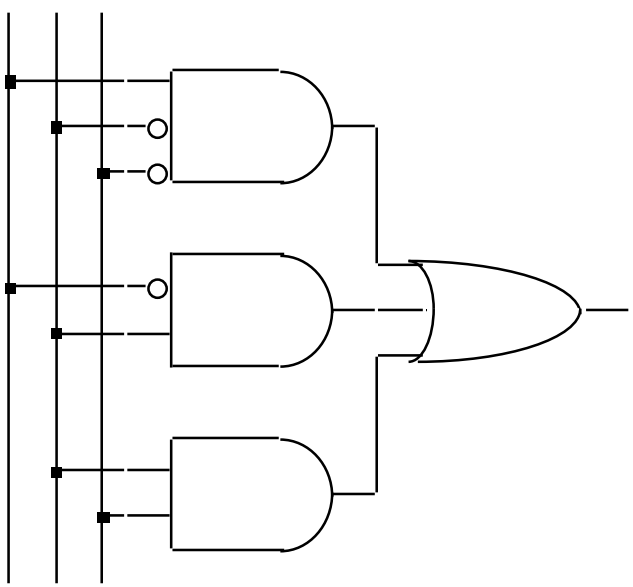


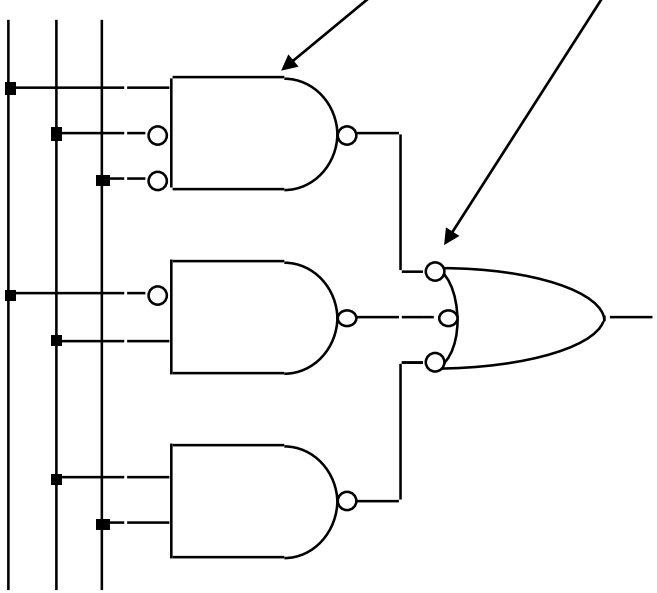
- Produto de somas
  - Porta OR para formar os termos da soma (maxtermos)
  - Porta AND para formar o produto



Lógica dois níveis usando portas NAND

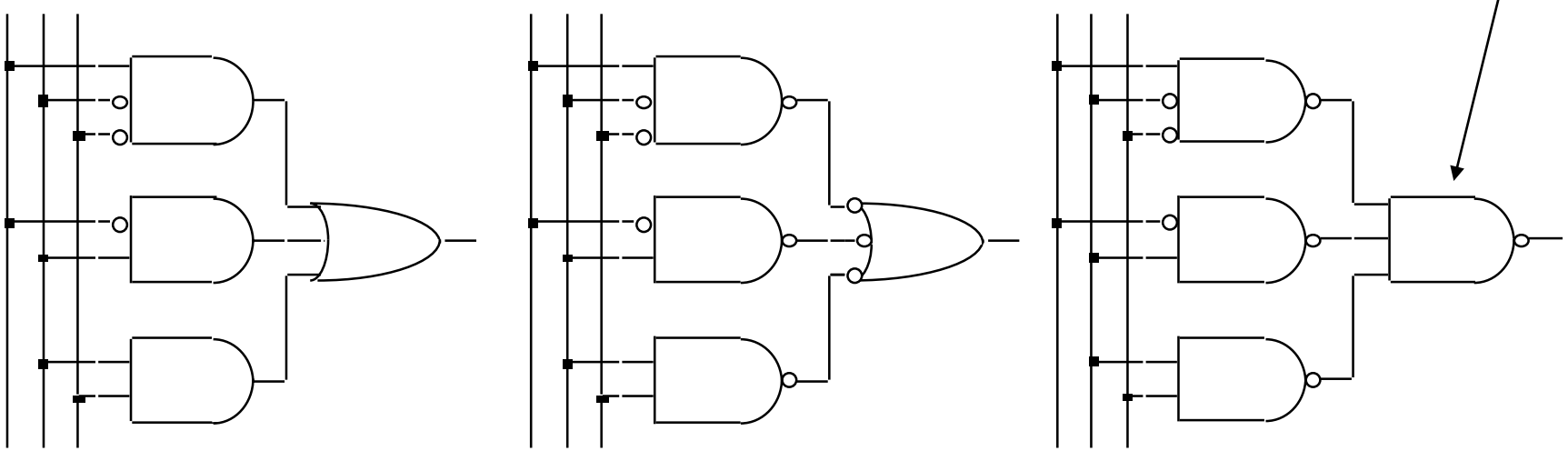
- Substituir portas AND mintermo por portas NAND
- Colocar inversão nas entradas da porta OR





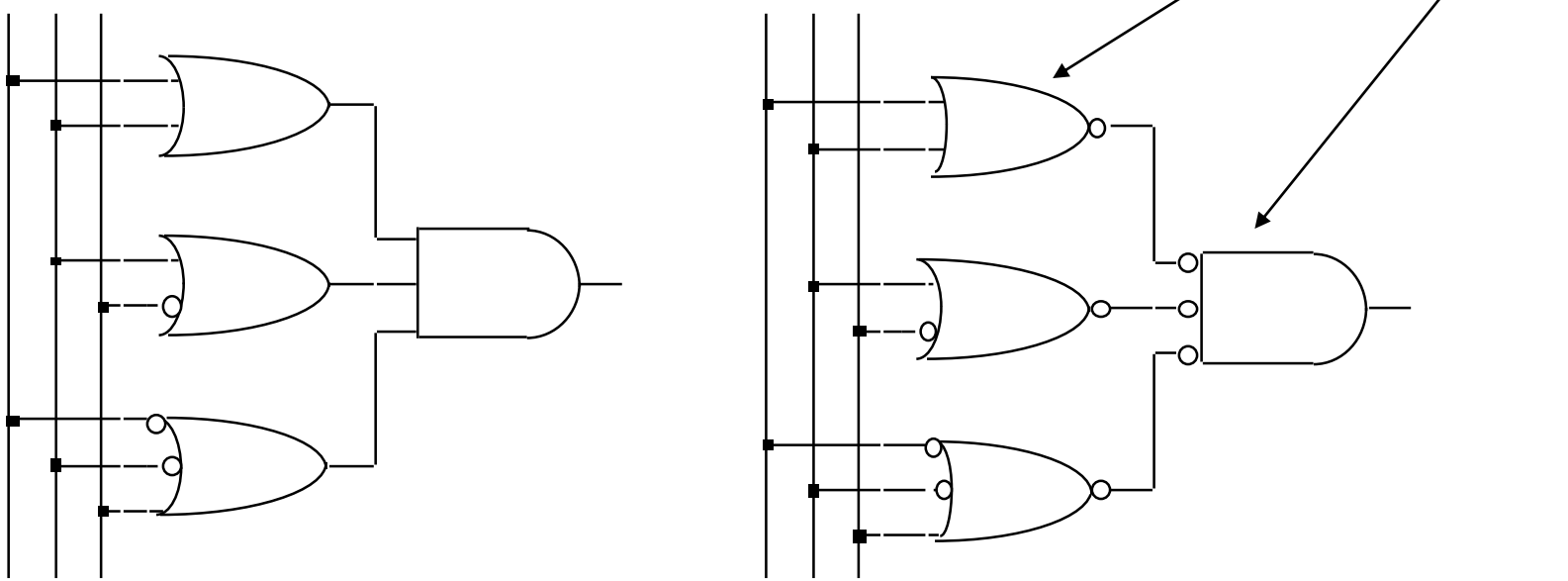
Lógica dois níveis usando portas NAND

- Porta OR com entrada invertida é uma porta NAND
  - de Morgan's: A' + B' = (A • B)'
- Rede NAND-NAND dois níveis
  - Entradas invertidas não são contadas
  - Em um circuito típico, inversão é feito uma única vez e com sinal distribuído



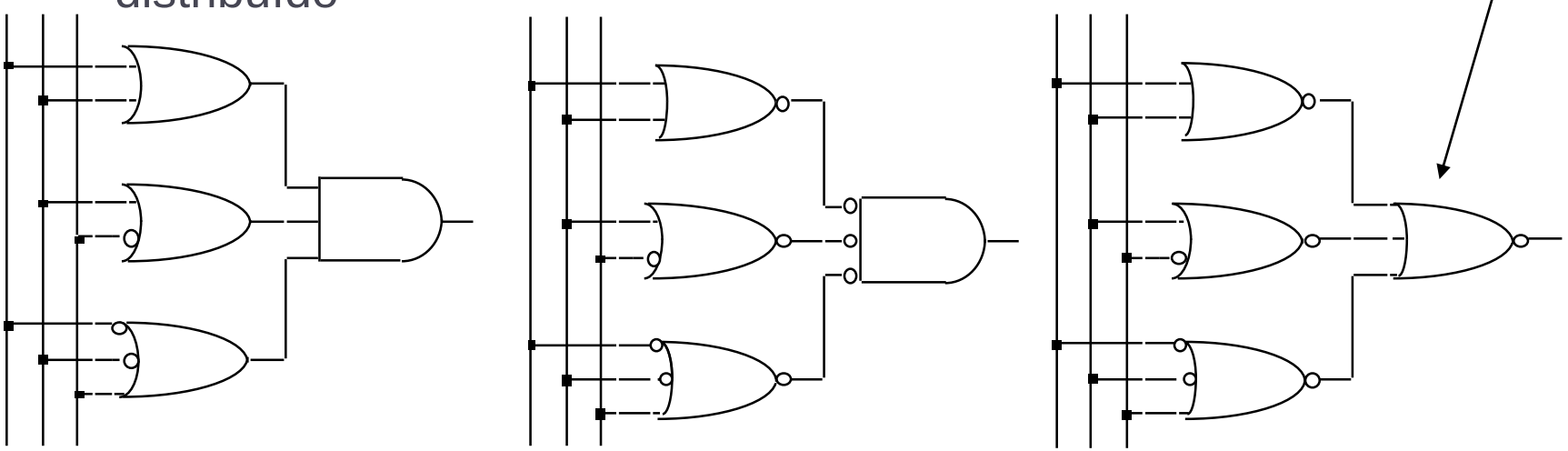
Lógica dois níveis usando portas NOR

- Substituir portas OR maxtermo por portas NOR
- Colocar inversão nas entradas da porta AND



Lógica dois níveis usando portas NOR

- Porta AND com entradas invertidas é uma porta NOR
  - de Morgan's: A' • B' = (A + B)'
- Rede NOR-NOR dois níveis
  - Entradas invertidas não são contadas
  - Em um circuito típico, inversão é feito uma única vez e com sinal distribuído



Lógica dois níveis usando portas NAND e NOR

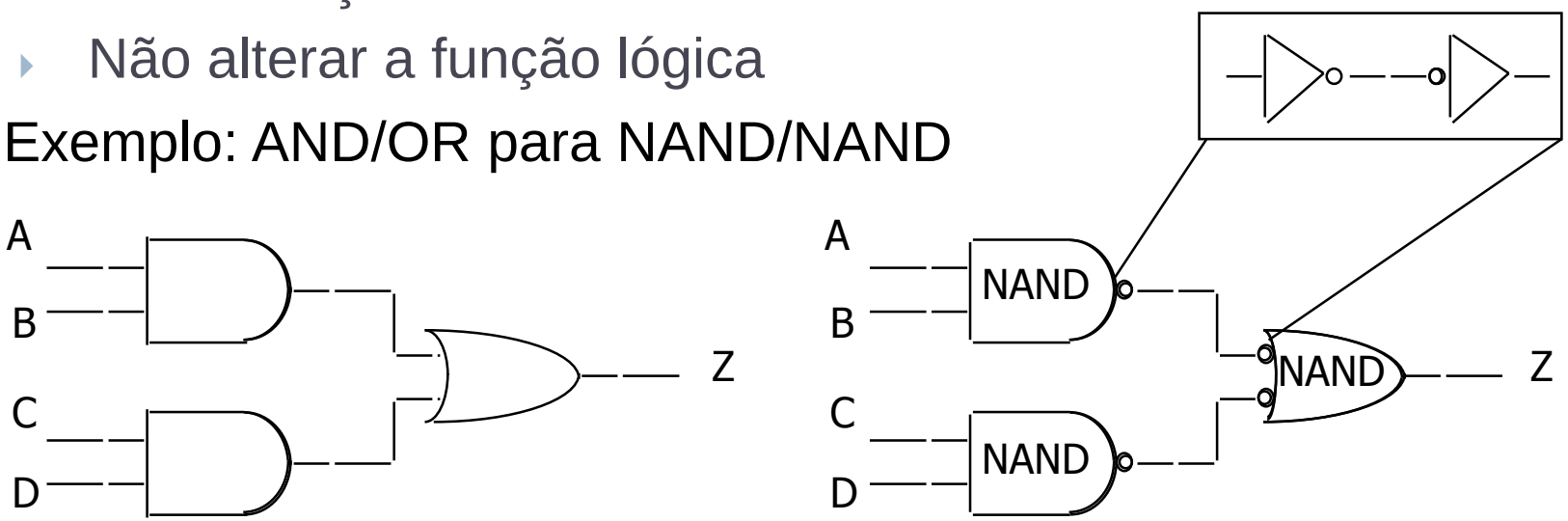
- Redes NAND-NAND e NOR-NOR
  - Lei de deMorgan's : $(A + B)' = A' \cdot B'$ $(A \cdot B)' = A' + B'$
  - Escrito de maneira diferente:

$A + B = (A' \cdot B')'$ $(A \cdot B) = (A' + B')'$

- Em outras palavras —
  - OR é a mesma que NAND com entradas complementadas
  - AND é a mesma que NOR com entradas complementadas
  - NAND é a mesma que OR com entradas complementadas
  - NOR é a mesma que AND com entradas complementadas

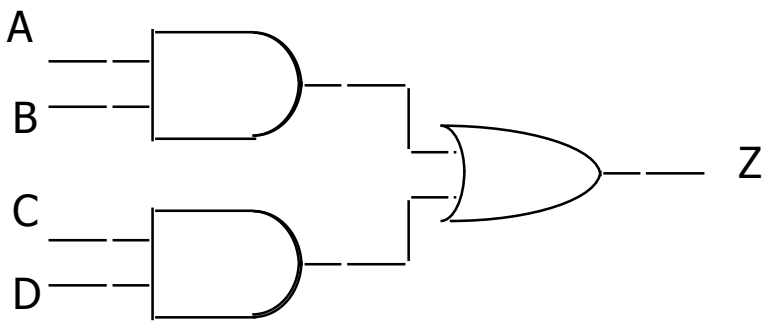
Conversão entre formas

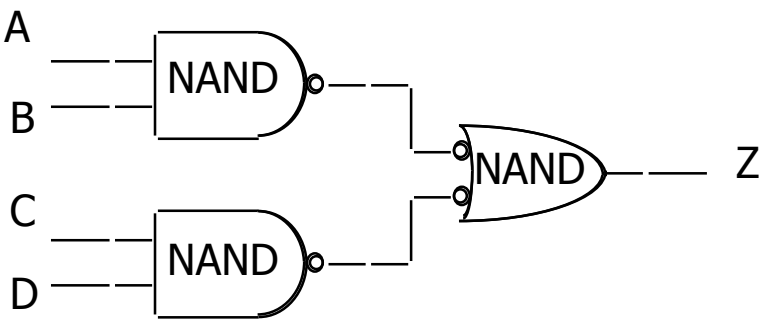
- Converter a partir de ANDs e ORs para redes de NANDs e NORs
  - Introduzir inversões apropriadas (“Bolhas”)
- Cada “bolha” introduzida precisa coincider com uma “bolha” correspondente.
  - Conservação de inversões
  - Não alterar a função lógica
- Exemplo: AND/OR para NAND/NAND



Conversão entre formas

- Exemplo: verificar equivalência das duas formas

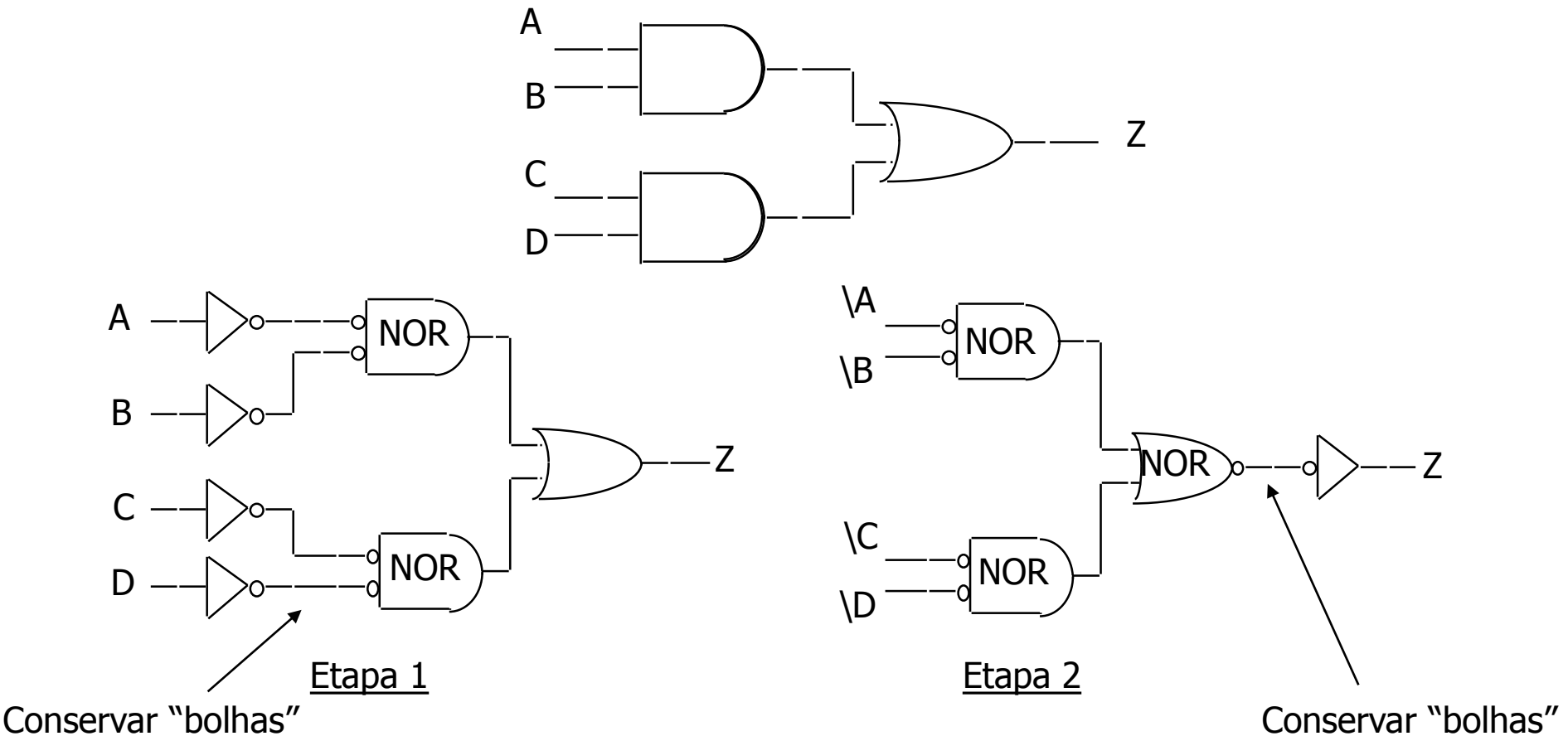




$$
\begin{aligned}
Z &= [\,(A \bullet B)' \bullet (C \bullet D)'\,]' \\
&= [\,(A' + B') \bullet (C' + D')\,]' \\
&= [\,(A' + B')' + (C' + D')'\,] \\
&= (A \bullet B) + (C \bullet D) \;\checkmark
\end{aligned}
$$

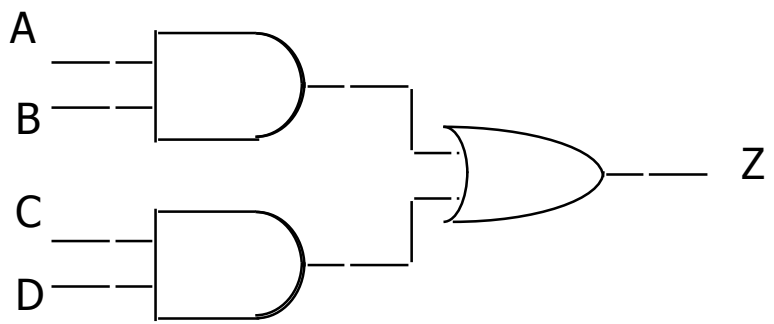
Conversão entre formas

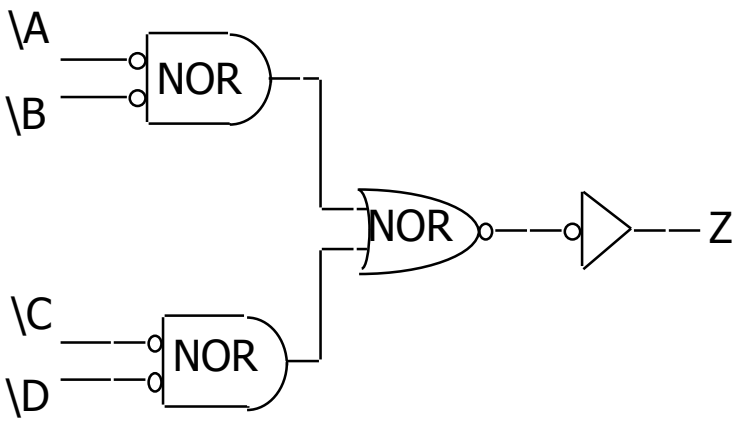
- Exemplo: Mapa rede AND/OR para uma rede NOR/NOR



Conversão entre formas

- Exemplo: verificar equivalência das duas formas





$$Z = \{\ [\ (A' + B')' + (C' + D')'\ ]'\ \}'$$
$$= \{\ \ (A' + B') \bullet (C' + D')\ \ \}'$$
$$= \quad (A' + B')' + (C' + D')'$$
$$= \quad (A \bullet B) + (C \bullet D) \quad \checkmark$$

Síntese multinível

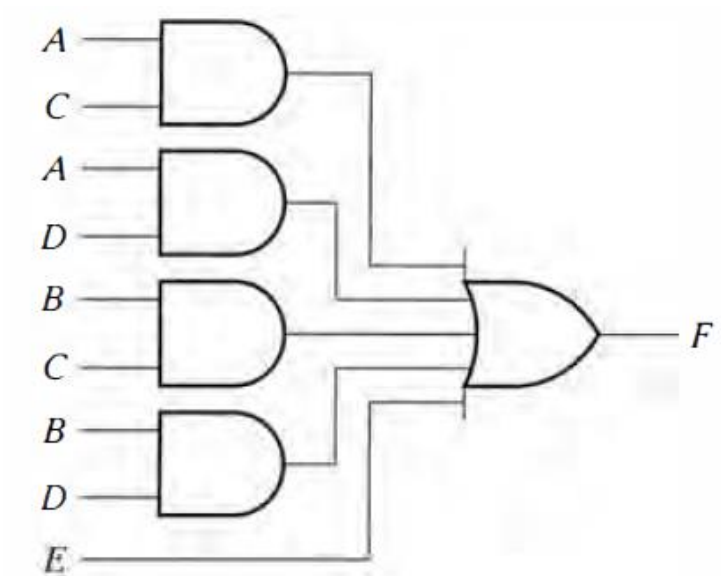
- Otimização de dois níveis
  - Mapas de Karnaugh, Quine-McCluskey, Espresso...
  - Rápido, pois sinal só deve propagar por duas portas
  - Desvantagem: Fan-out muito grande
    - Portas lógicas com mais de 4 entradas são muito raras na maioria das tecnologias
    - Diminuição do fan-out substituindo por portas com menos entradas
      - Aumenta tempo de propagação -> Multinível
- Síntese multinível
  - Compromisso entre área e velocidade
  - Fase 1: Independente de tecnologia
  - Fase 2: Dependente de tecnologia: depende das portas lógicas disponíveis.

Síntese multinível

- Fatoração
  - Expressão de dois níveis é transformada em multinível sem introduzir funções intermediárias. Exemplo:

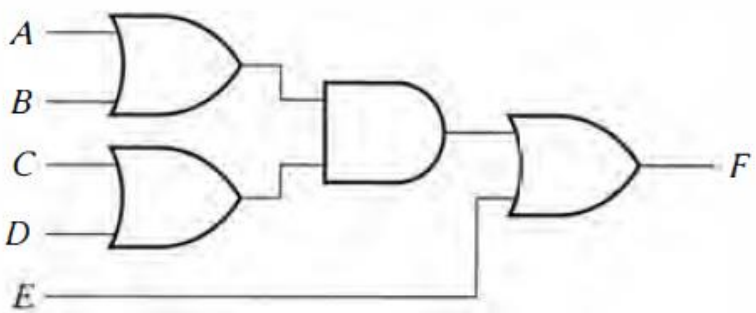
Inicial

$$F = AC + AD + BC + BD + E$$



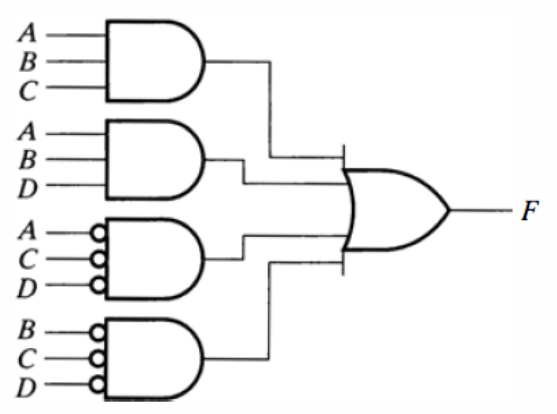
Fatorada

$$F = (A + B)(C + D) + E$$

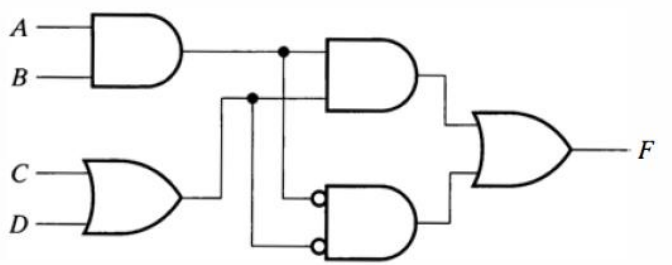


Síntese multinível

- Decomposição
  - Aplicado em expressões booleanas já fatoradas. Exemplo:
    - Inicial
      - $F = \mathrm{ABC} + \mathrm{ABD} + \overline{A}\,\overline{C}\,\overline{D} + \overline{B}\,\overline{C}\,\overline{D}$

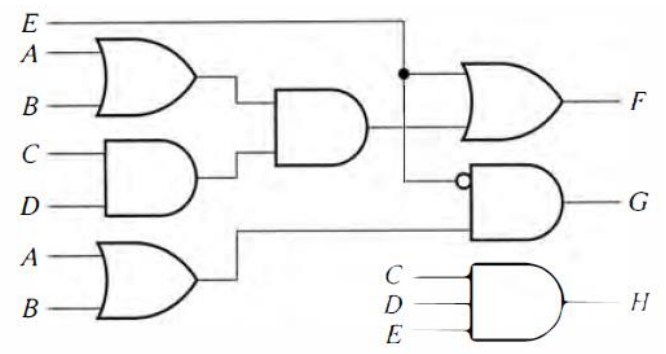


    - Fatoração
      - $F = (\mathrm{AB})\,(\mathrm{C} + \mathrm{D}) + (\overline{A} + \overline{B})(\overline{C}\,\overline{D})$
      - $F = (\mathrm{AB})\,(\mathrm{C} + \mathrm{D}) + (\overline{AB})(\overline{C + D})$
    - Decomposição
      - $X = AB$
      - $Y = C + D$
      - $F = XY + \overline{X}\,\overline{Y}$

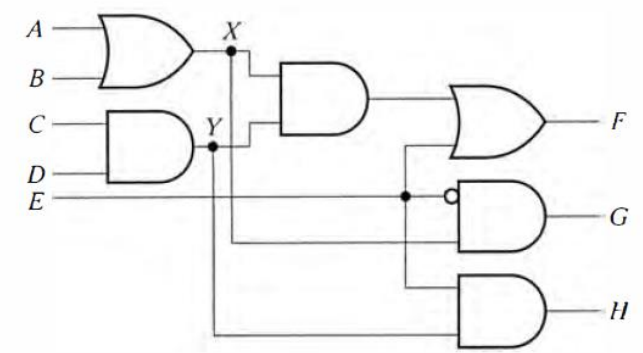


Síntese multinível

- Extração
  - Aplicado em conjuntos de expressões booleanas. Exemplo:
    - Inicial
      - $F = (\mathrm{A} + \mathrm{B})\mathrm{CD} + \mathrm{E}$
      - $\mathrm{G} = (\mathrm{A} + \mathrm{B})\,\overline{E}$
      - $\mathrm{H} = \mathrm{CDE}$



    - Extração
      - $\mathrm{X} = \mathrm{A} + \mathrm{B}$
      - $\mathrm{Y} = CD$
      - $F = XY + E$
      - $G = X\overline{E}$
      - $H = YE$

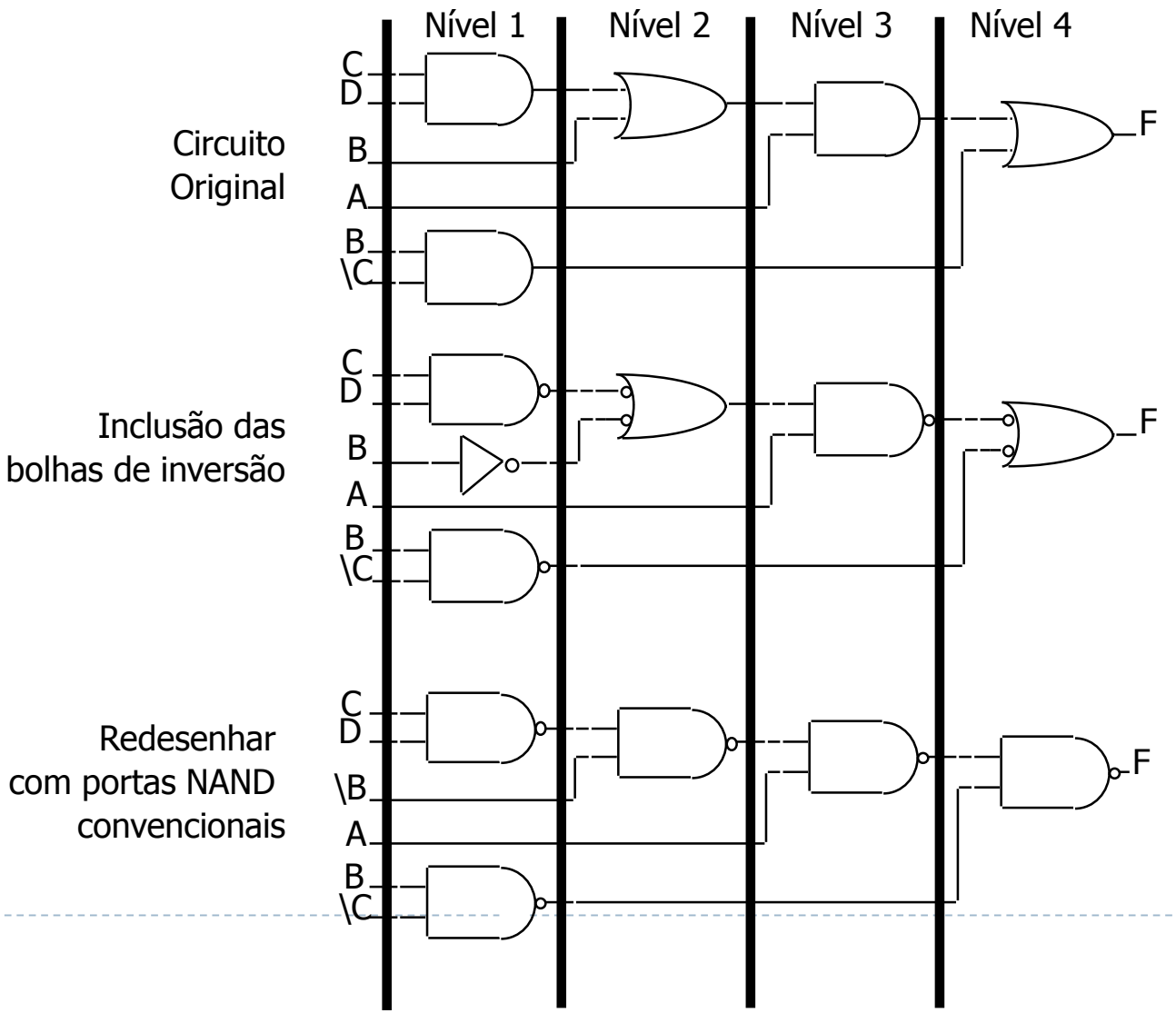


Síntese multinível

- Substituição
  - Utilizada para re-expressar funções como formas fatoradas de subexpressões. Exemplo:
    - Inicial: $F = \mathrm{A} + \mathrm{BCD},\ \mathrm{G} = \mathrm{A} + \mathrm{BC}$
    - Substituição: $\mathrm{F} = \mathrm{A} + \mathrm{BCD} = \mathrm{G}(\mathrm{A} + \mathrm{D})$
- Colapso
  - Contrário de substituição. Utilizado para reduzir o número de níveis de lógica e melhorar a temporização. Exemplo:
    - $F = \mathrm{G}(\mathrm{A} + \mathrm{D})$
    - $F = (\mathrm{A} + \mathrm{BCD})(\mathrm{A} + \mathrm{D})$
    - $F = \mathrm{AA} + \mathrm{AD} + \mathrm{ABC} + \mathrm{BCD}$
    - $F = \mathrm{A} + \mathrm{BCD}$

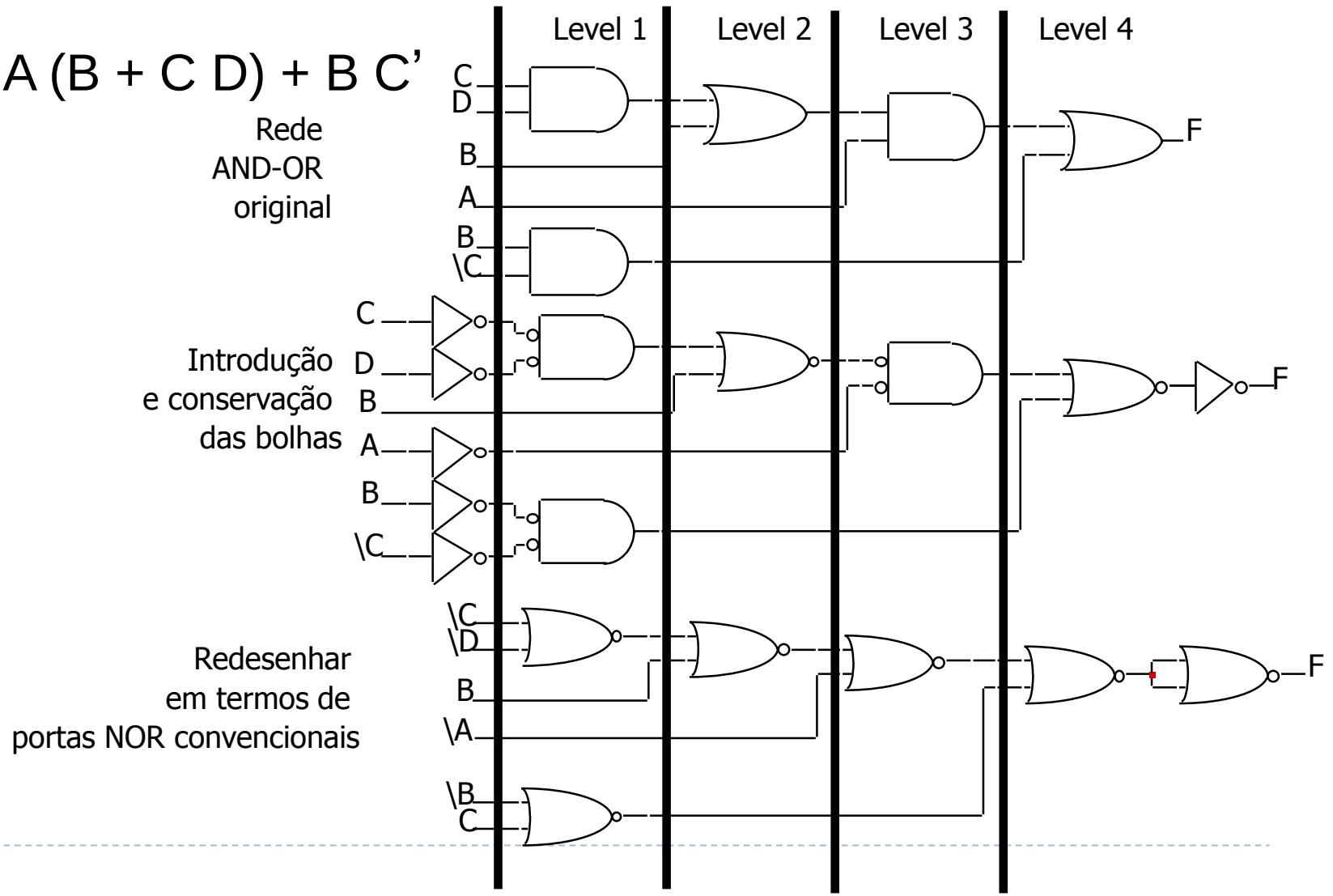
Conversão lógica multi níveis para portas NAND

- $F = \mathrm{A}(\mathrm{B} + \mathrm{CD}) + \mathrm{B}\overline{C}$



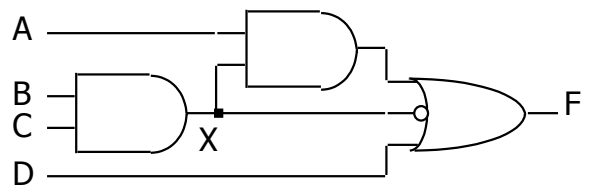
Conversão lógica multi níveis para portas NAND

- F = A (B + C D) + B C’

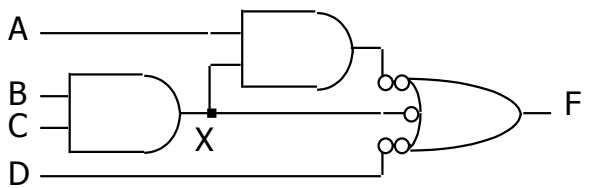


Conversão entre formas

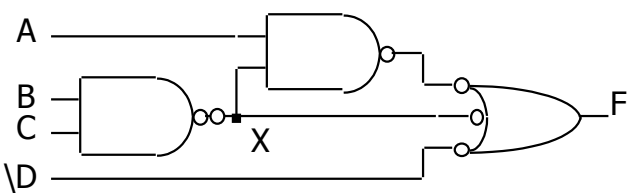
- Exemplo



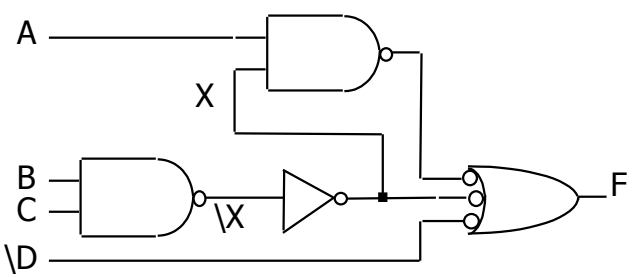
Circuito original



Adicionar duas bolhas para
inverter todas as entradas da porta OR



Adicionar duas bolhas para
inverter as saídas das portas AND

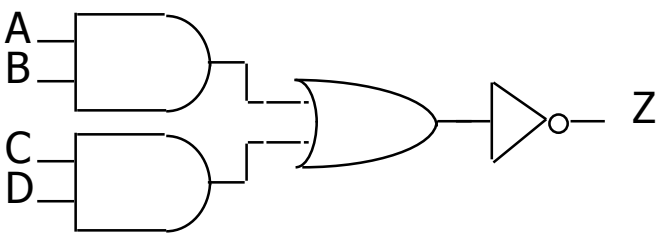


Inserir inversores para eliminar
as duas bolhas no fio

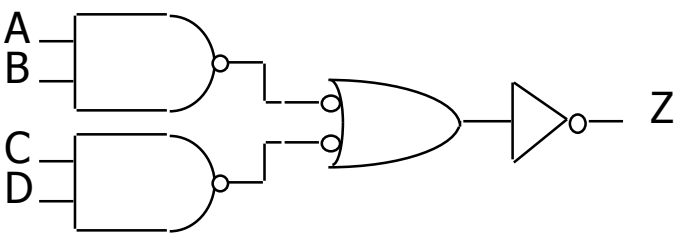
Portas AND-OR invertidas

- Função AOI: três estágios de lógica — AND, OR, inversor
  - Múltiplas portas “empacotadas” como um único bloco de circuito

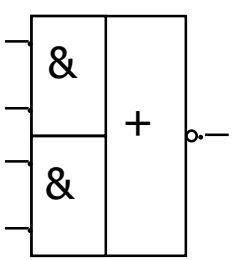
Lógica conceitual



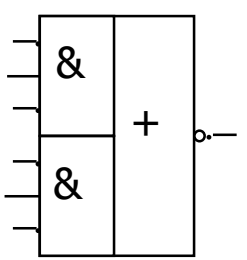
Implementação possível



Símbolo porta AOI 2x2

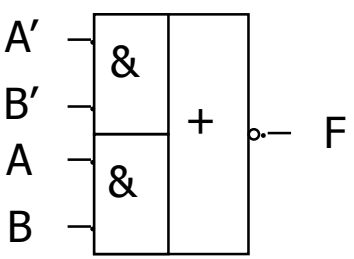


Símbolo porta AOI 3x2



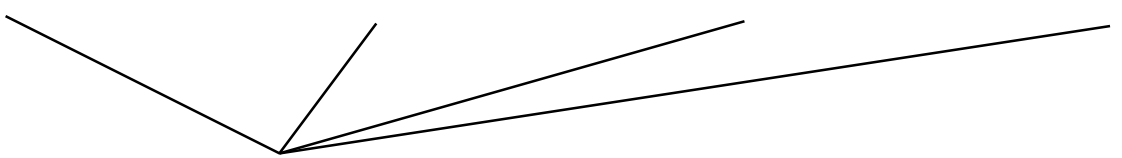
Conversão para forma AOI

- Procedimentos gerais para colocar na forma AOI
  - Calcula o complemento da função na forma soma de produtos
  - Agrupando os 0s no mapa de Karnaugh
- Exemplo: Implementação XOR
  - A xor B = A' B + A B'
  - Forma AOI:
    - F = (A' B' + A B)'



Exemplos usando portas AOI

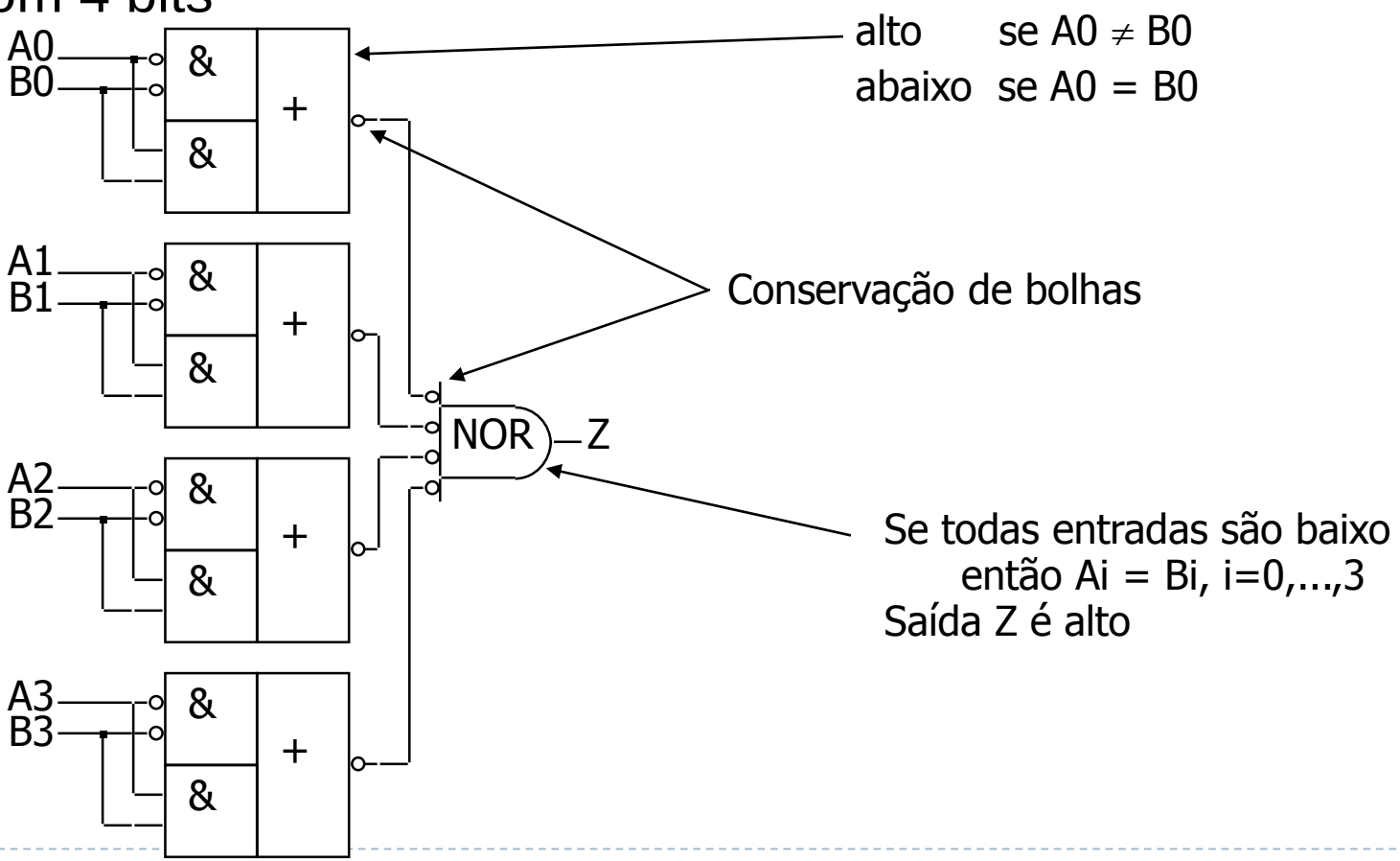
- Exemplo:
  - F = A B + A C' + B C'
  - F = (A' B' + A' C + B' C)'
  - Implementado por 2-entradas 3 pilhas de porta AOI
  - F = (A + B) (A + C') (B + C')
  - F = [(A' + B') (A' + C) (B' + C)]'
  - Implementado por 2 entradas 3 pilhas de porta OAI
- Exemplo: função igualdade 4 bits
  - Z = (A0 B0 + A0' B0')(A1 B1 + A1' B1')(A2 B2 + A2' B2')(A3 B3 + A3' B3')



Cada implementação é uma porta AOI 2x2 simples

Exemplos usando portas AOI

- Exemplo: Implementação AOI de uma função de igualdade com 4 bits



Resumo multi níveis

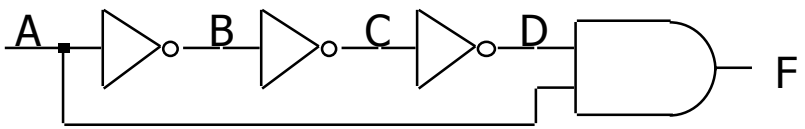
- Vantagens
  - Circuitos podem ser menores
  - Portas tem menor fan-in
  - Circuitos podem ser rápidos
- Desvantagens
  - Mais dificuldade para projetar
  - Ferramentas para otimizar não são tão boas quanto para dois níveis
  - Análises são mais complexas

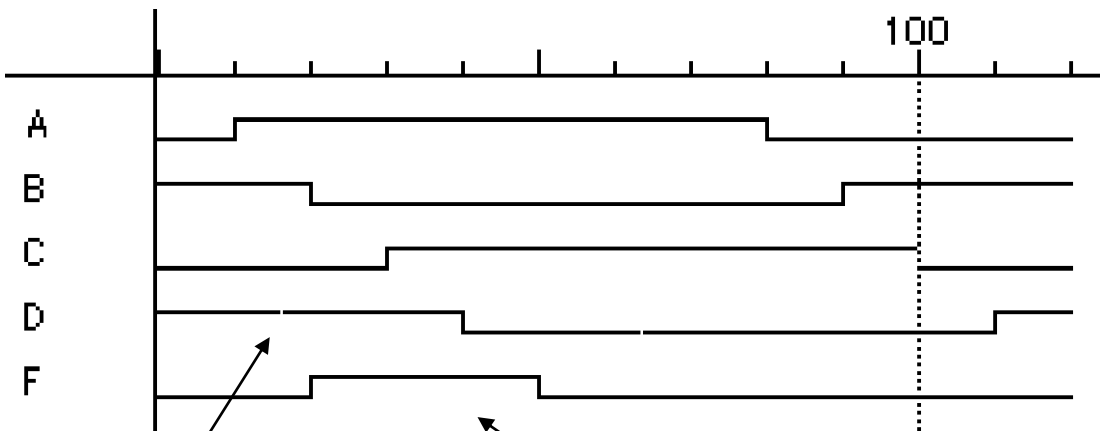
Comportamento de tempo em redes combinacionais

- Formas de onda
  - Visualização de valores transportados no sinal do fio sobre o tempo
  - Útil para explicar sequências de eventos (alteração nos valores)
- Ferramentas de simulação são usadas para criar estas formas de onda
  - Entrada para o simulador inclui portas e suas conexões
  - Estímulos de entrada, que são, sinais de entrada das formas de onda
- Alguns termos
  - Atraso porta — Tempo para alterar a entrada causando a modificação da saída
    - Atraso min – atraso típico/nominal – Atraso max
    - Projetores cuidadosos no projeto para o pior caso
  - Tempo de subida — tempo para a saída transitar a partir da voltagem baixa para alta
  - Tempo de descida — tempo para saída transitar a partir da voltagem alta para baixa
  - Largura do pulso — tempo que uma saída fica alta ou baixa entre as trocas

Alterações momentâneas nas saídas

- Pode ser útil — circuitos modelados por pulso
- Pode ser um problema — operação circuito incorreto (glitches/hazards)
- Exemplo: circuito modelado por pulso
  - A' • A = 0
  - Atrasos importantes



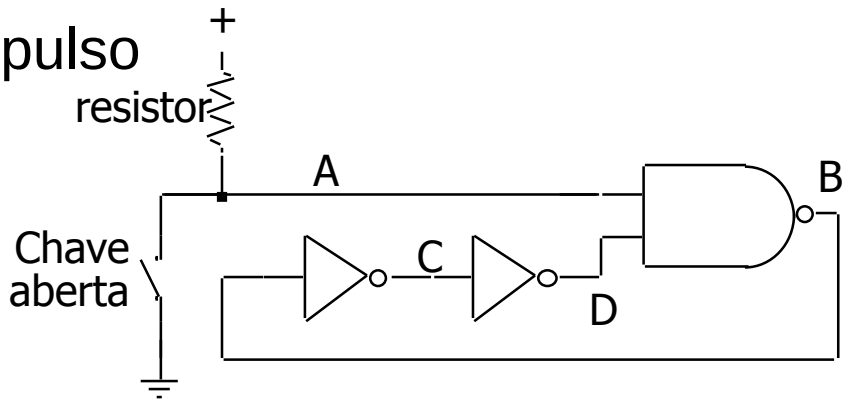


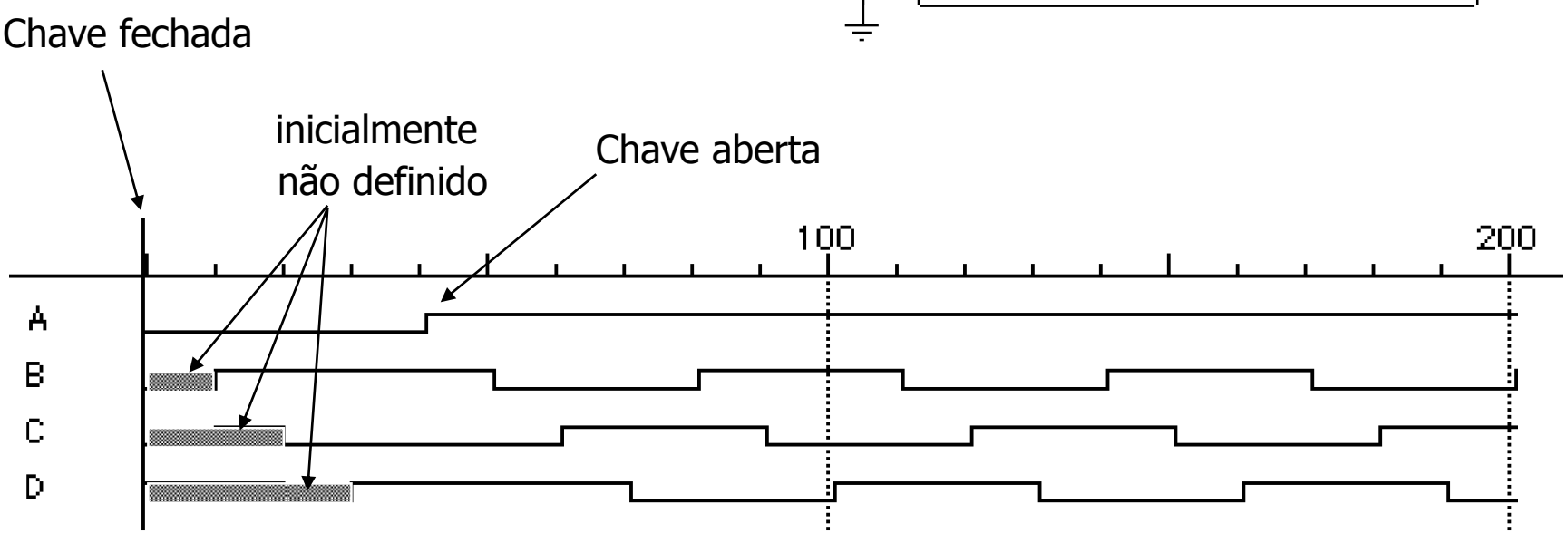
D permace alto para o atraso das três portas, depois A altera de baixo para alto

F não é sempre 0
Largura do pulso do atraso das 3 portas

Comportamento Oscilatório

- Outro circuito modelado por pulso





Linguagem de descrição de Hardware

- Descreve o hardware em níveis de variação de abstração
- Descrição estrutural
  - Substituição textual para esquemático
  - Composição hierárquica de módulos a partir das primitivas
- Descrição comportamental/funcional
  - Descrição do que o módulo faz, não como
  - Síntese do circuito gerado pelo módulo
- Simulação semântica

HDLs

- Abel (cerca de 1983) – deselvolvido por Data-I/O
  - Alvo para dispositivos lógicos programáveis
  - Não é bom para algo além de máquina de estados
- ISP (cerca de 1977) – projeto de pesquisa da CMU
  - Simulação, mas não síntese
- Verilog (cerca de 1985) – desenvolvido por Gateway (absorvido por Cadence)
  - Similar ao Pascal e C
  - Atrasos são somente interações com o simulador
  - Razoavelmente eficiente e fácil para escrever
  - Padrão IEEE
- VHDL (cerca de 1987) – Padrão DoD patrocinado
  - Similar ao Ada (ênface na reutilização e na facilidade de manutenção)
  - Simulação semântica visível
  - Muito geral mas verboso
  - Padrão IEEE

Verilog

- Suporte a descrição estrutural e comportamental
- Estrutural
  - Estrutura explícita do circuito
  - e.g., cada porta lógica instanciada e conectada a outras
- Comportamental
  - Programa descreve o comportamento da entrada/saída do circuito
  - Muitas implementações estruturais podem ter o mesmo comportamento
  - e.g., diferente implementação de uma função Boolean
- Maioritariamente se utiliza Verilog comportamental no Aldec ActiveHDL
  - Confiar nos esquemas quando queremos descrições estruturais

Modelo estrutural

```
module xor_gate (out, a, b);
  input     a, b;
  output    out;
  wire      abar, bbar, t1, t2;

  inverter invA (abar, a);
  inverter invB (bbar, b);
  and_gate and1 (t1, a, bbar);
  and_gate and2 (t2, b, abar);
  or_gate  or1 (out, t1, t2);

endmodule
```

Modelo comportamental simples

- Atribuição contínua

```
module xor_gate (out, a, b);
  input          a, b;
  output         out;
  reg            out;

  assign #6 out = a ^ b;

endmodule
```

Simulação registrador – Mantêm o valor do sinal

Atraso a partir da alteração da entrada para modificar a saída

Modelo comportamental simples

- Bloco always

```
module xor_gate (out, a, b);
  input          a, b;
  output         out;
  reg            out;

  always @(a or b) begin
     #6 out = a ^ b;
  end

endmodule
```

Especifica quando o bloco será executado
Ex: acionado quando a ou b forem alterados

Manipulando uma simulação através do “testbench”

```
module testbench (x, y);
  output          x, y;
  reg [1:0]       cnt;

  initial begin
    cnt = 0;
    repeat (4) begin
      #10 cnt = cnt + 1;
      $display ("@ time=%d, x=%b, y=%b, cnt=%b",
        $time, x, y, cnt); end
    #10 $finish;
  end

  assign x = cnt[1];
  assign y = cnt[0];
endmodule
```

Vetor 2 bits

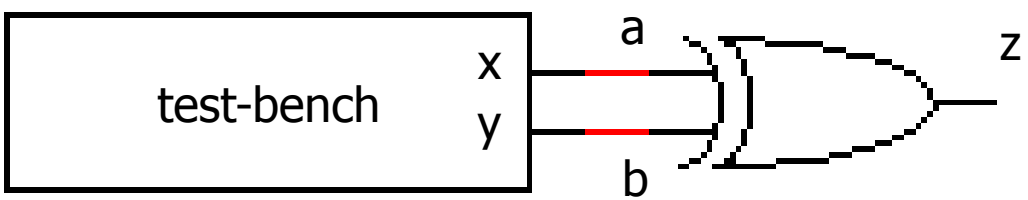
Bloco initial executado somente quando a simulação começa

print no console

Diretiva para parar a simulação

Simulação completa

- Componente de estímulos instantâneos e dispositivo para testar um esquemático



Exemplo comparador

```
module Compare1 (Equal, Alarger, Blarger, A, B);
  input     A, B;
  output    Equal, Alarger, Blarger;

  assign #5 Equal = (A & B) | (~A & ~B);
  assign #3 Alarger = (A & ~B);
  assign #3 Blarger = (~A & B);
endmodule
```

Mais um modelo comportamental complexo

```
module life (n0, n1, n2, n3, n4, n5, n6, n7, self, out);
  input     n0, n1, n2, n3, n4, n5, n6, n7, self;
  output    out;
  reg       out;
  reg [7:0] neighbors;
  reg [3:0] count;
  reg [3:0] i;

  assign neighbors = {n7, n6, n5, n4, n3, n2, n1, n0};

  always @(neighbors or self) begin
    count = 0;
    for (i = 0; i < 8; i = i+1) count = count + neighbors[i];
    out = (count == 3);
    out = out | ((self == 1) & (count == 2));
  end

endmodule
```

Linguagem de descrição de Hardware vs. Linguagem de programação

- Estrutura programa
  - Instanciação de múltiplos componentes do mesmo tipo
  - Interconexões especificadas entre módulos via esquemático
  - Hierarquia de módulos (apenas leaves podem ser HDL em Aldec ActiveHDL)
- Atribuição
  - Atribuição contínua (lógica sempre calculável)
  - Propagação do atraso (computação leva tempo)
  - Temporização dos sinais são importantes (quando a computação tem efeito)
- Estrutura de dados
  - Tamanho explicitamente expresso – nenhuma estrutura dinâmica
  - Nenhum ponteiro
- Paralelismo
  - hardware é naturalmente paralelo (precisa do suporte de múltiplas threads)
  - Atribuições podem ocorrer em paralelo (não somente sequencial)

Linguagem de descrição de Hardware vs. Linguagem de programação

- Módulos - especificação das entradas, saída, bidirecional, sinais internos
- Atribuição contínua – uma saída da porta é uma função de todos os tempos das entradas da porta (não precisa esperar ser “chamada”)
- Propagação do atraso – conceito de tempo e atraso nas entradas afetam a saída da porta
- Composição – módulos conectados junto com fios
- Hierarquia – módulos encapsulados em blocos funcionais

Resumo - Trabalhando com lógica combinacional

- Problemas projetos
  - Preenchendo tabelas verdades
  - Funções incompletamente especificadas
  - Simplificação lógica dois níveis
- Realização lógica de dois níveis
  - Redes NAND e NOR
  - Redes de funções boolean e seus comportamentos de tempo
- Comportamento de tempo
- Linguagem de descrição de Hardware
- Depois
  - Tecnologias de lógica combinacional
  - Mais estudos de casos de projetos