PCS 3225
Sistemas Digitais II

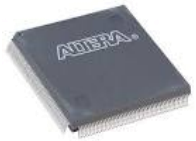
Dispositivos Lógicos Programáveis

Adaptado por Glauber (2018)

Lógica programável

- **Como implementar um sistema digital?**
  - Circuitos SSI ou MSI: programação via ligação entre os chips

    - Ex.: (de)mux, (de)codificador, somador, ...
  - μProcessador, μControlador: programação por linguagem de montagem (“software”)
  - ASICs (“Application Specific Integrated Circuit”): circuito personalizado para aplicação
  - **Dispositivos programáveis** : programação por fusíveis ou transistores especiais



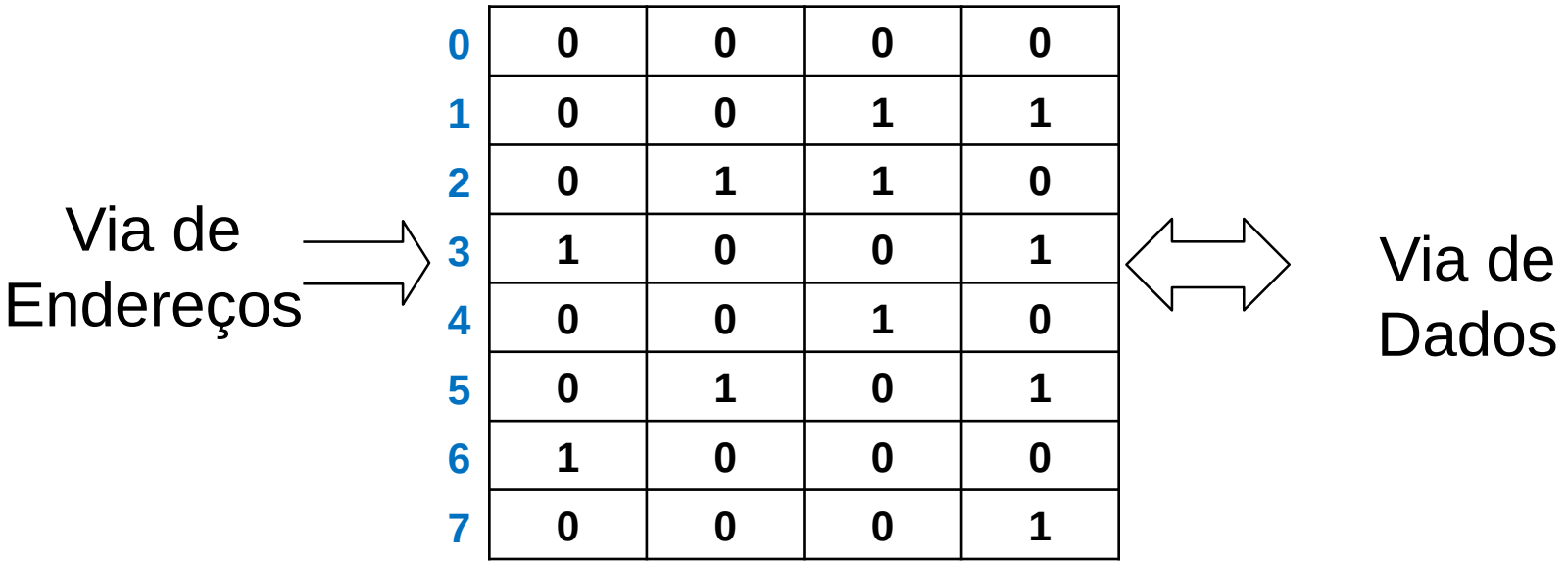
    - Ex.: Memórias, FPGA, CPLD

Memórias como PLDs

- **Desafio**: use uma memória para implementar a função f(x) = 3*x mod 10
  - Assuma que x tem 3 bits: 0 a 7

➔Quantos bits de endereço? E de dados?

➔Que valores precisam ser colocados na memória?



Memórias como PLDs

- **Objetivo**: use memória para construir um circuito que implemente a seguinte lógica

Entradas			Saídas	
A	**B**	**C**	**F1**	**F2**
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

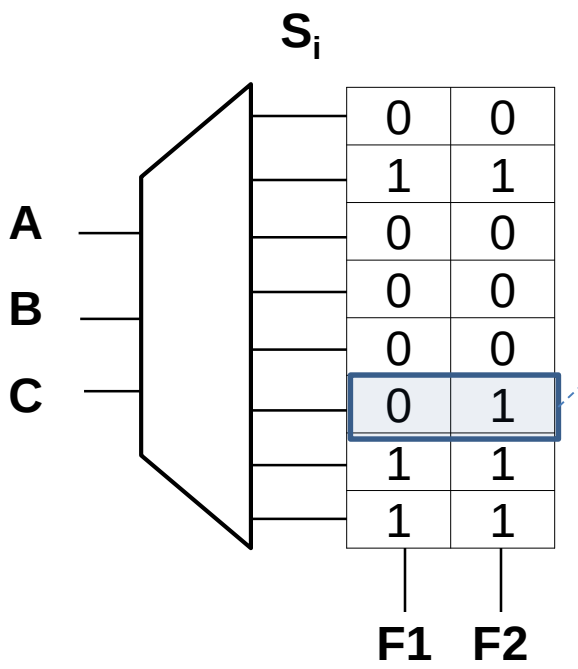
$$F1 = \overline{A}BC + A\overline{B}C + ABC$$

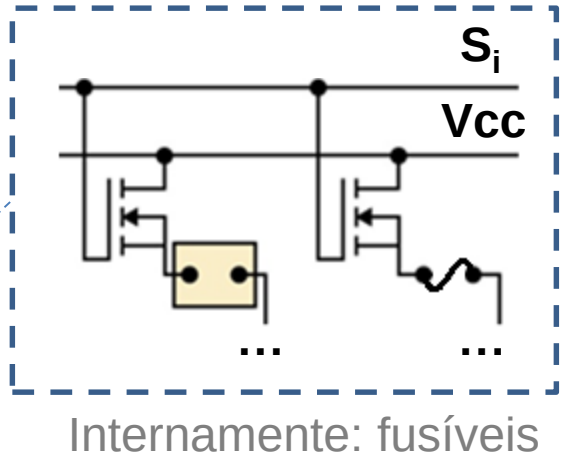
$$F2 = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

Memórias como PLDs

Memória PROM 8x2bits

Entradas			Saídas	
A	B	C	F1	F2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1





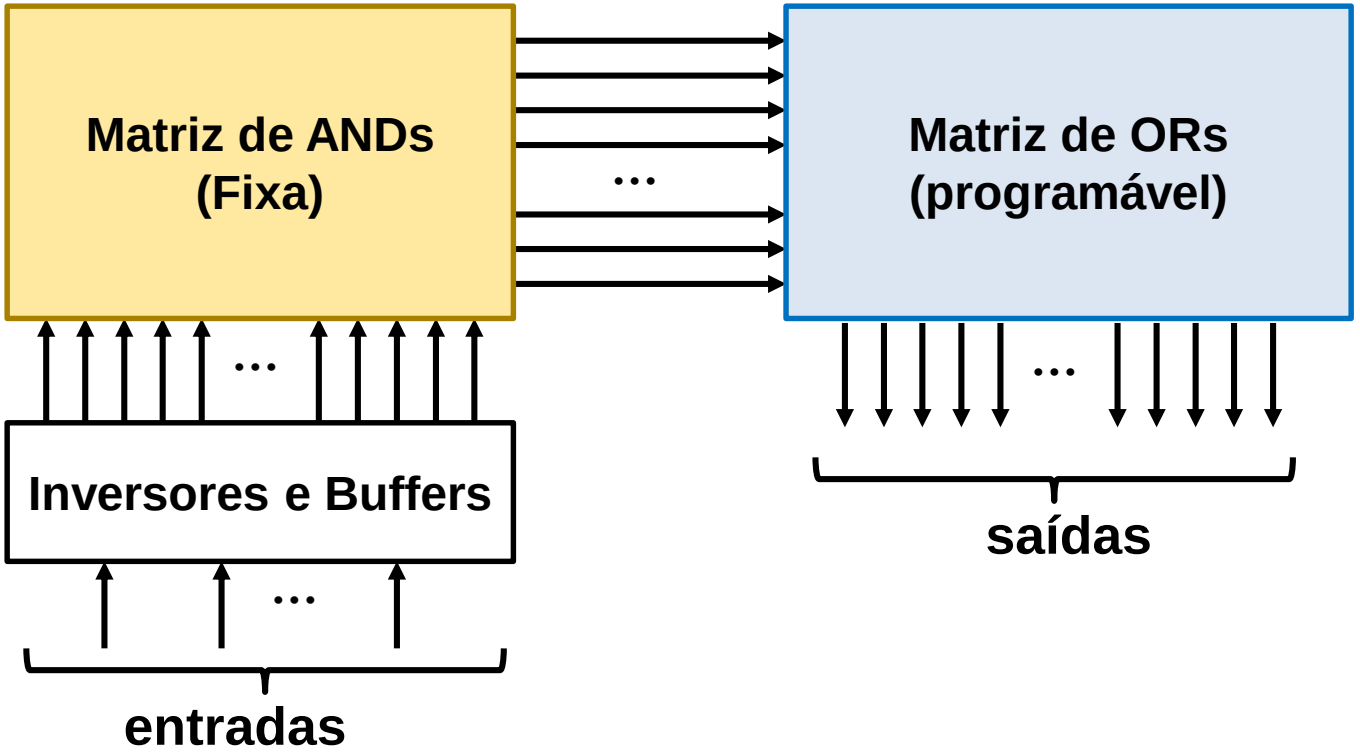
Internamente: fusíveis

$$F1 = \overline{A}BC + A\overline{B}C + ABC$$

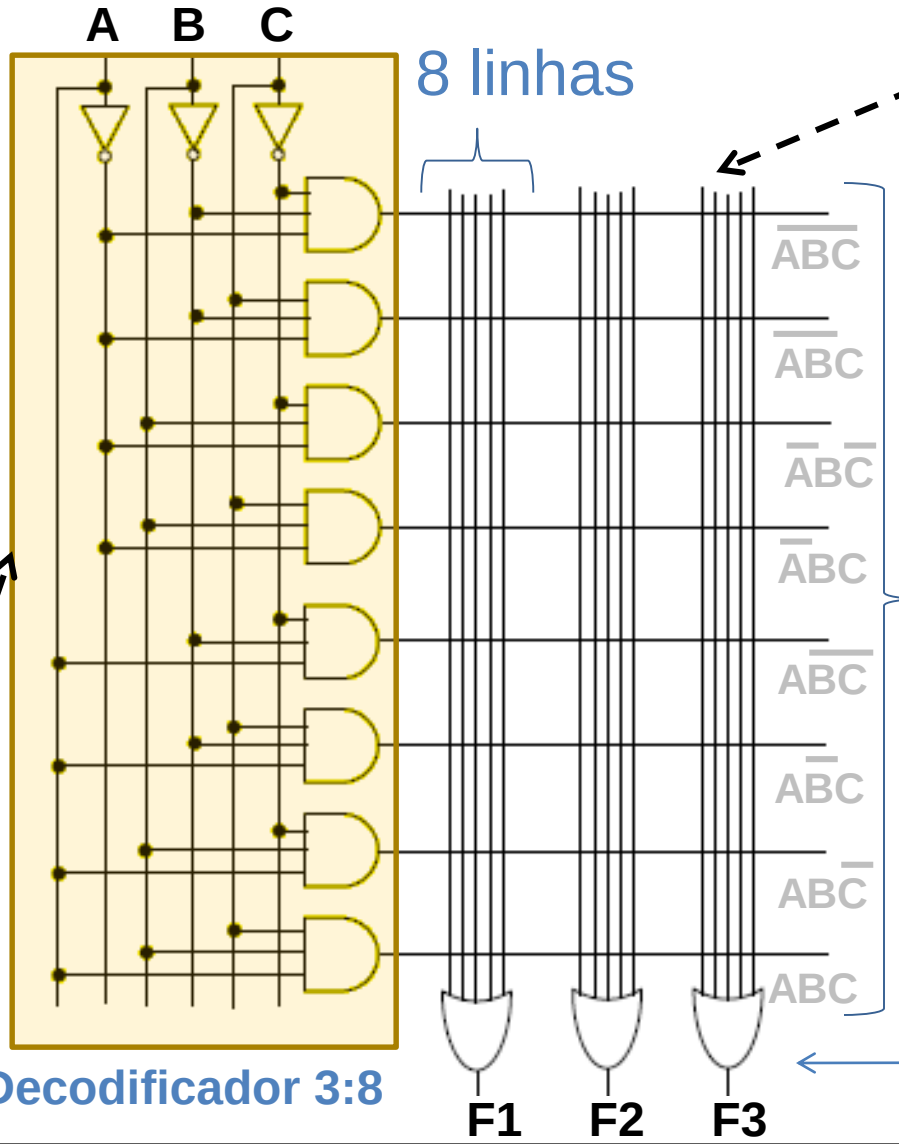
$$F2 = \overline{A}\overline{B}C + A\overline{B}C + AB\overline{C} + ABC$$

Memórias como PLDs

- Funcionalmente, *PROMs são formadas por uma matriz de portas AND fixas e uma matriz de portas OR programáveis.

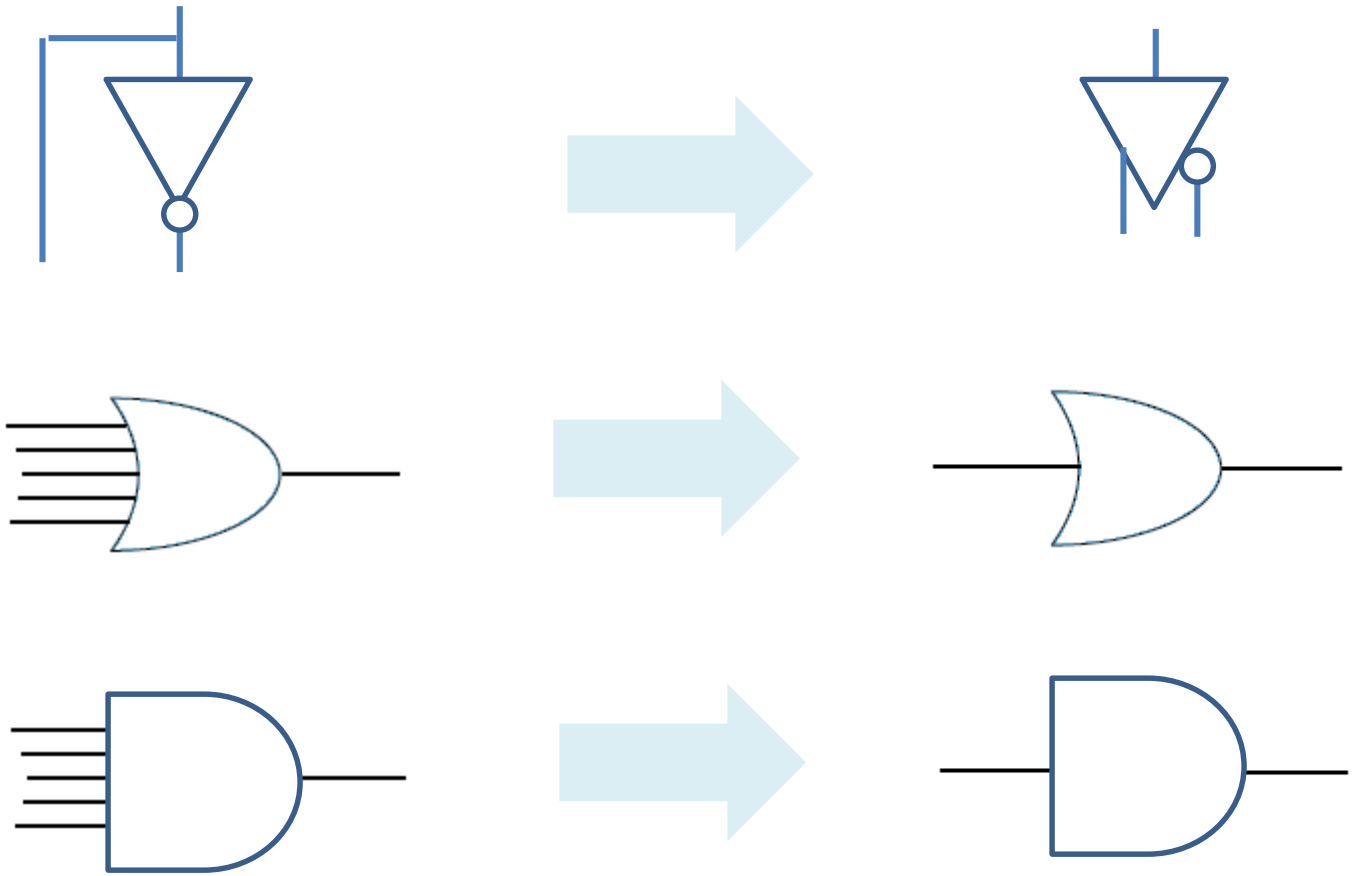


Memórias como PLDs



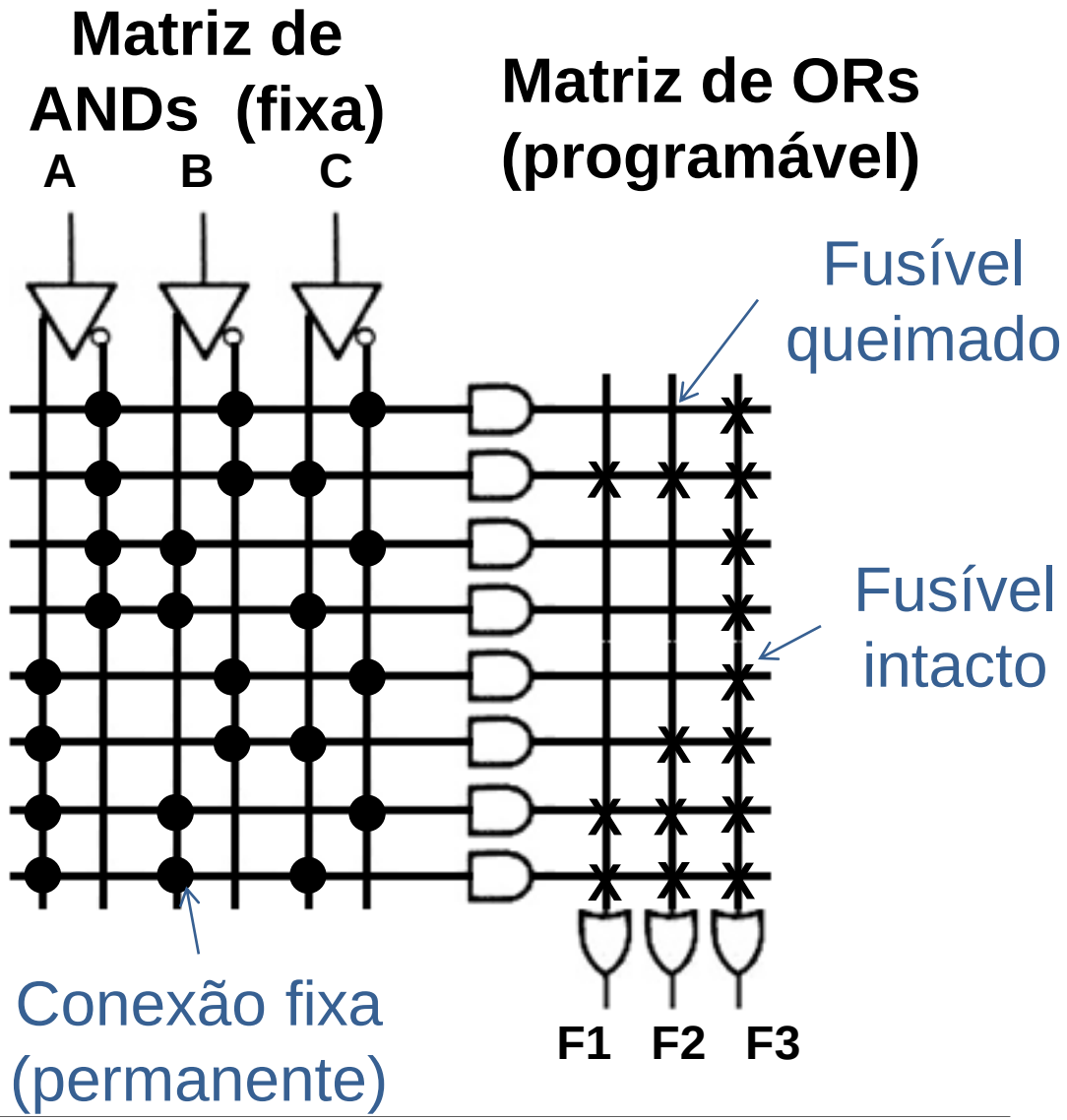
Memórias como PLDs

- Notação:



Memórias como PLDs

Entradas			Saídas		
A	B	C	F1	F2	F3
0	0	0	0	0	1
0	0	1	1	1	1
0	1	0	0	0	1
0	1	1	0	0	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	1	1	1
1	1	1	1	1	1



Memórias como PLDs: Exercício

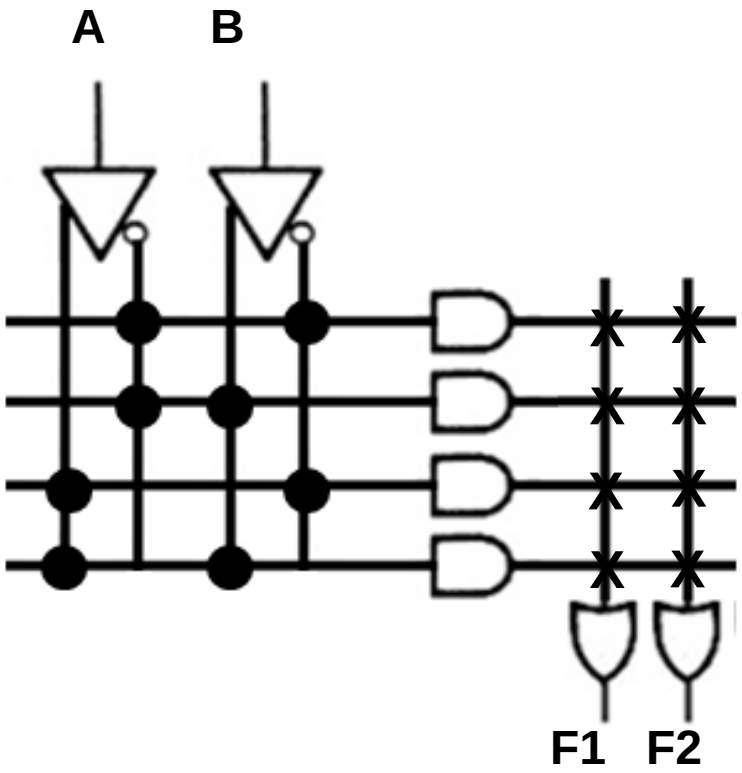
- Use uma ***PROM** para programar as funções F1 e F2 abaixo
- **Resposta**:

Entradas **Saídas**

A	B	F1	F2
0	0	0	0
0	1	1	1
1	0	0	1
1	1	1	0

$F1 = \overline{A}B + A\overline{B} = B$

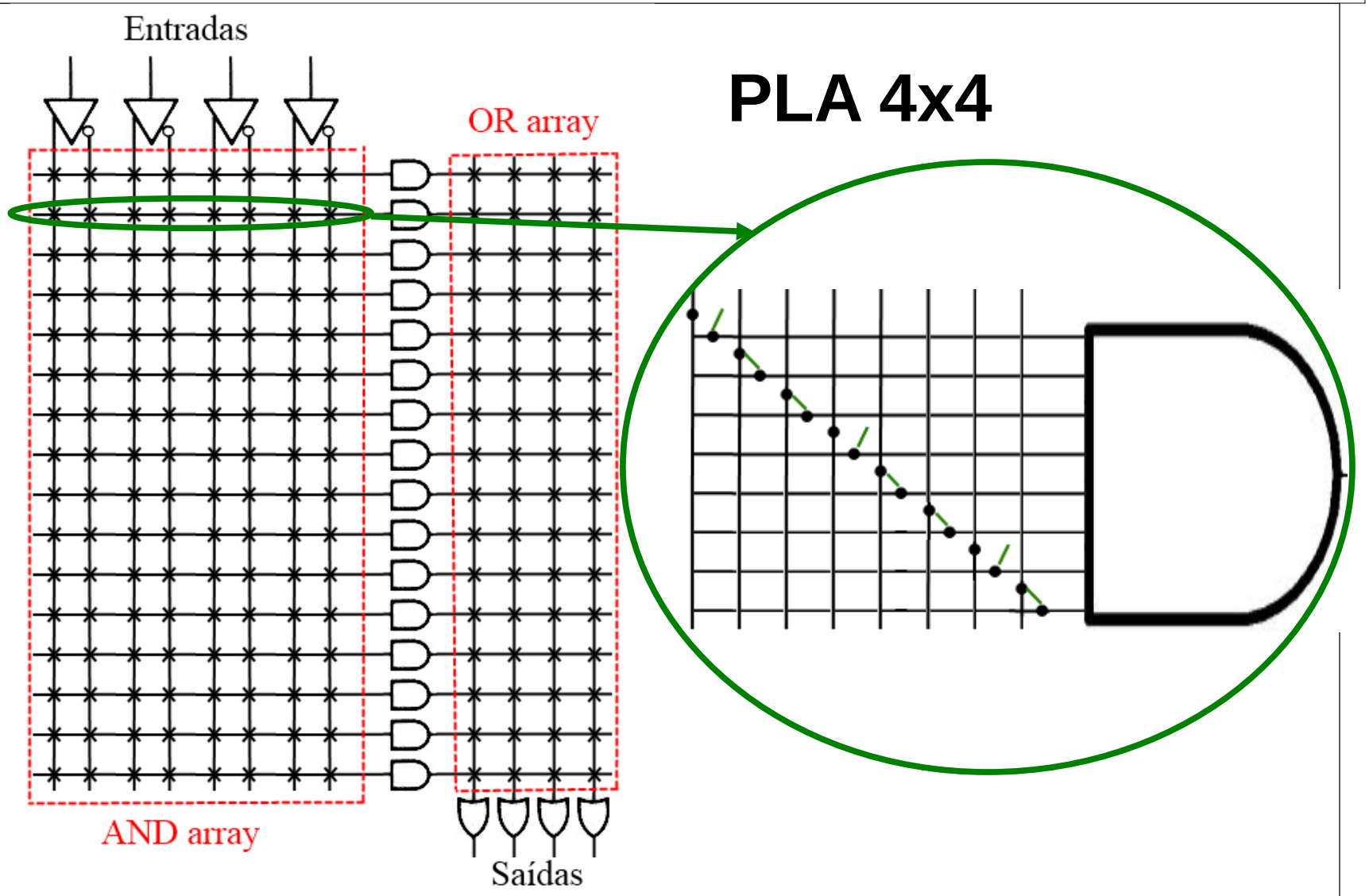
$F2 = A\overline{B} + \overline{A}B$



Arranjos Lógicos Programáveis: PLAs

- ***PROMs** como PLDs: Implementam **tabela verdade completa**, sem minimização. Paralelamente ao uso de memórias, dispositivos de lógica programável de fato foram criados.
- As **PLAs** (*Programmable Logic Arrays*) são matrizes de lógica programável nas quais tanto a **matriz de ANDs** (primeiro nível) como a **matriz de ORs** (segundo nível) são **programáveis**. Ou seja, PLAs sintetizam funções na forma de **soma de produtos**: Produtos compartilhados entre diversas saídas.

Arranjos Lógicos Programáveis: PLAs



Arranjos Lógicos Programáveis: Exercício 1

- Use uma **PLA** para programar as funções F1 e F2 abaixo
- **Resposta**:

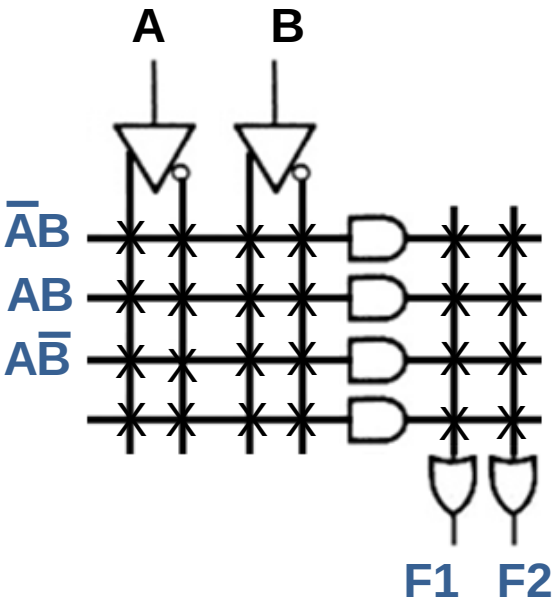
Entradas **Saídas**

A	B	F1	F2
0	0	0	0
0	1	1	1
1	0	0	1
1	1	1	0

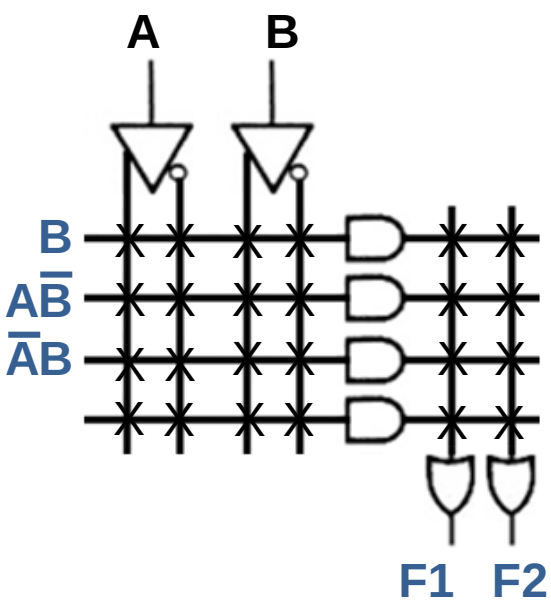
$F1 = \overline{A}B + A\overline{B} = B$

$F2 = A\overline{B} + \overline{A}B$

Solução 1



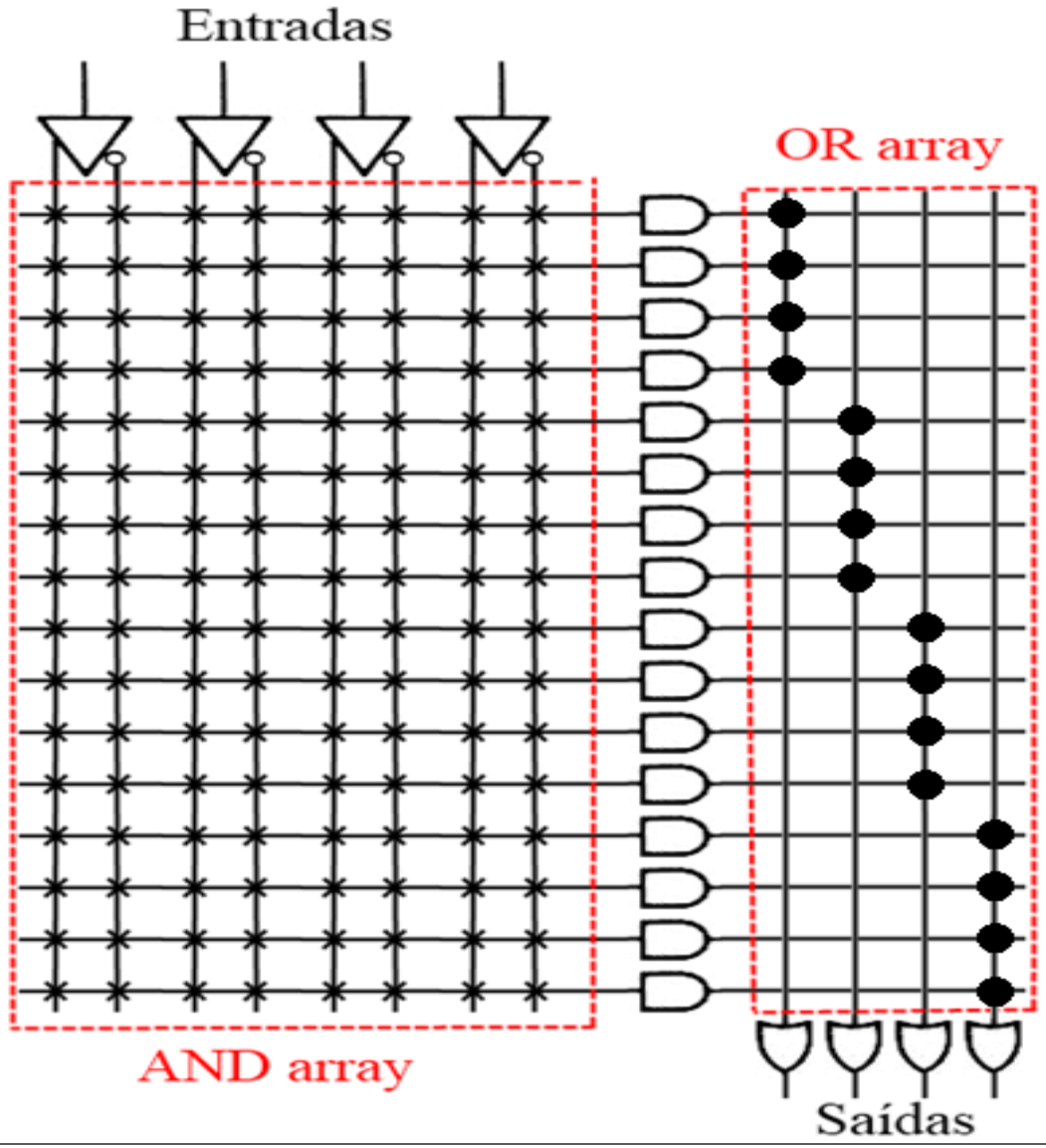
Solução 2



Arranjos Lógicos Programáveis: PALs

- PLAs eram de difícil utilização e caras.
- Como alternativa foram propostas as **PALs**, onde o **nível de ANDs é programável**, mas o **nível de ORs é fixo**.
  - Menor flexibilidade, porém custo também menor.
- Nesse caso, um número fixo de entradas para cada porta OR é usado.
  - Deve ser usada uma PAL adequada ao nível de complexidade desejado (número de termos).
- Tanto PALs como PLAs são não voláteis, ou seja, podem ser desligadas sem perda de informação.

Arranjos Lógicos Programáveis: PALs



Arranjos Lógicos Programáveis: Exercício 2

- Use uma **PAL** para programar as funções F1 e F2 abaixo
- **Resposta**:

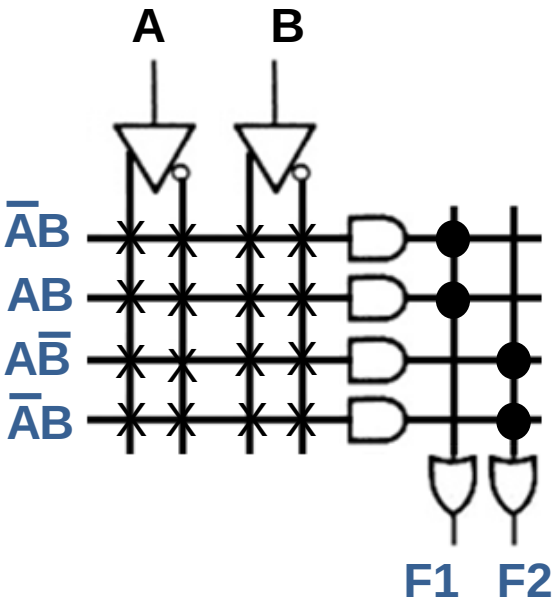
Entradas **Saídas**

A	B	F1	F2
0	0	0	0
0	1	1	1
1	0	0	1
1	1	1	0

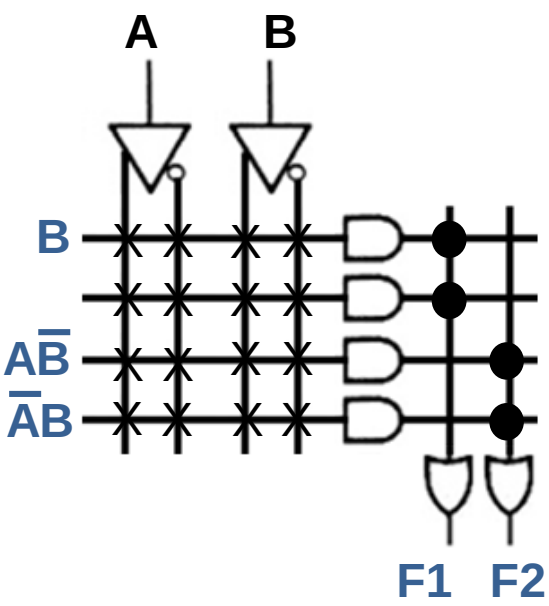
$F1 = \overline{A}B + A\overline{B} = B$

$F2 = A\overline{B} + \overline{A}B$

Solução 1



Solução 2



Arranjos Lógicos Programáveis: Complexidade

- Complexidade das funções que podem ser implementadas depende de:
  - Número total de **entradas/saídas**: ***e*** **/** ***s***
  - Núm. **total** de **produtos** nas **funções de saída**: ***p***
  - Número de **produtos distintos**: p_d
  - Núm. **máx.** de **produtos** nas **funções de saída**: m_p

	ANDs	**ORs**	**pinos**
***PROM**	2^e ($2e$ portas)	s (2^e portas)	$e + s$
PLA	p_d ($2e$ portas)	s ($\leq m_p$ portas)	$e + s$
PAL	p ($2e$ portas)	s ($\leq m_p$ portas)	$e + s$

Arranjos Lógicos Programáveis: Exemplo

- Vamos sintetizar as funções F1 e F2 seguintes:

Entradas			Saídas	
A	**B**	**C**	**F1**	**F2**
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

Arranjos Lógicos Programáveis: Exemplo

- Passo 1: Determinar os produtos

Entradas			Saídas	
A	**B**	**C**	**F1**	**F2**
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

F1 = A'B'C + ABC' + ABC
F2 = A'B'C + AB'C + ABC' + ABC

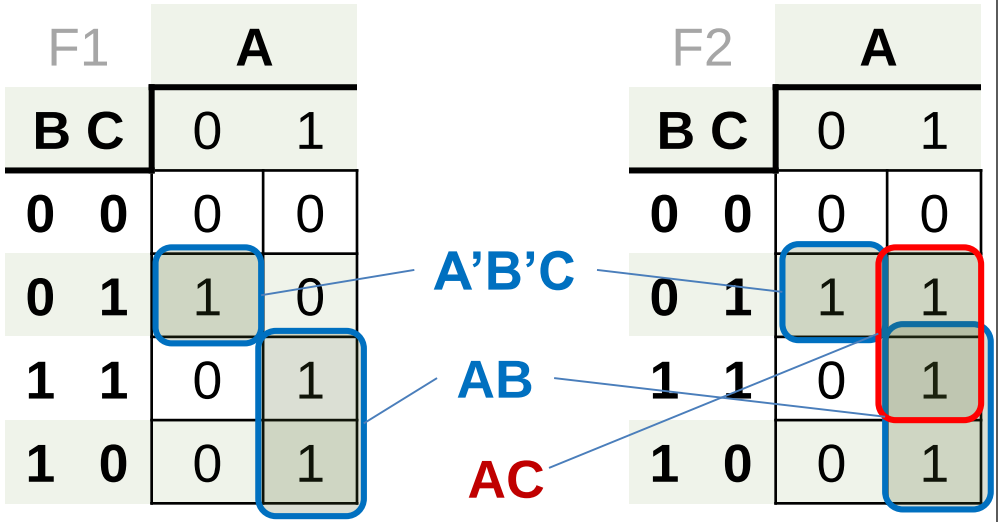
- Se usássemos uma *PROM, teríamos que implementar cada um dos mintermos

- Porém, com um PLA ou PAL, podemos minimizar as funções!

Arranjos Lógicos Programáveis: Exemplo

- Passo 1 (PLA): Determinar os produtos
  - Com PLA: maximizar produtos compartilhados

Entradas			Saídas	
A	B	C	F1	F2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1



F1 = A'B'C + AB
F2 = A'B'C + AB + AC

Arranjos Lógicos Programáveis: Exemplo

- Passo 2 (PLA): Matriz de Personalidade

F1 = A’B’C + AB
F2 = A’B’C + AB + AC

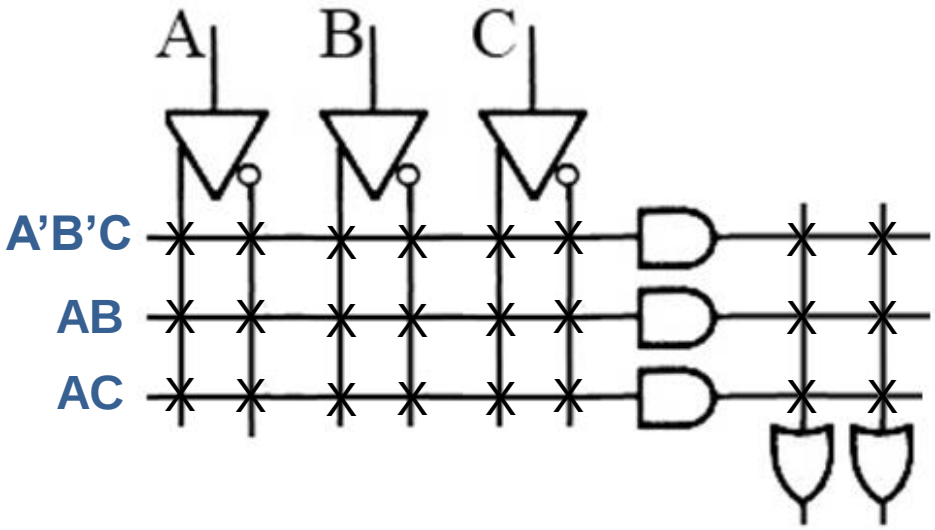
Produto	Entrada			Saídas	
	A	B	C	F1	F2
A'B'C	0	0	1	1	1
AB	1	1	-	1	1
AC	1	-	1	0	1

Compar-tilhados

Arranjos Lógicos Programáveis: Exemplo

- Passo 3 (PLA): Desenhar o circuito

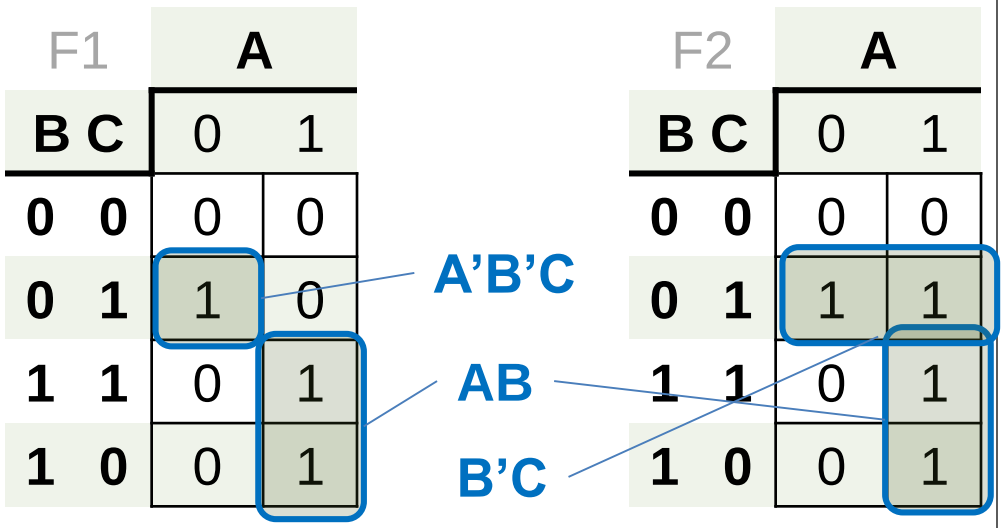
Produto	Entrada			Saídas	
	A	B	C	F1	F2
A'B'C	0	0	1	1	1
AB	1	1	-	1	1
AC	1	-	1	0	1



Arranjos Lógicos Programáveis: Exemplo

- Passo 1 (PAL): Determinar os produtos
  - Com PAL: minimizar número de produtos

Entradas			Saídas	
A	B	C	F1	F2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1



F1 = A'B'C + AB
F2 = B'C + AB

Arranjos Lógicos Programáveis: Exemplo

- Passo 2 (PAL): Matriz de Personalidade
  - Obs.: não é essencial para a PAL, mas ajuda.

F1 = A’B’C + AB
F2 = B’C + AB

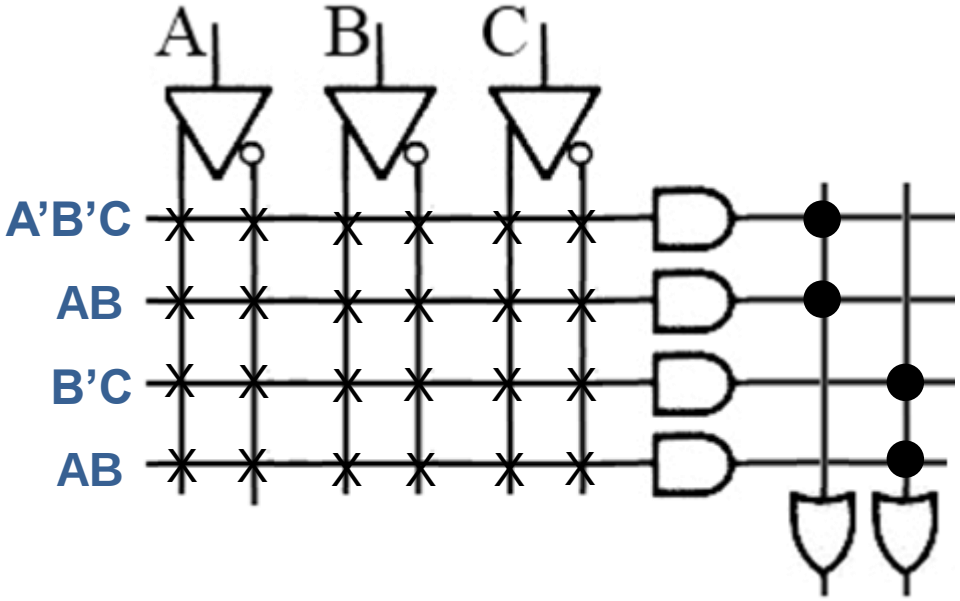
Produto	Entrada			Saídas	
	A	B	C	F1	F2
A'B'C	0	0	1	1	0
AB	1	1	-	1	0
B'C	-	0	1	0	1
AB	1	1	-	0	1

Repe-tidos

Arranjos Lógicos Programáveis: Exemplo

- Passo 3 (PAL): Desenhar o circuito

Produto	Entrada			Saídas	
	A	B	C	F1	F2
A'B'C	0	0	1	1	0
AB	1	1	-	1	0
B'C	-	0	1	0	1
AB	1	1	-	0	1



Arranjos Lógicos Programáveis: Exemplo

- Vamos sintetizar:
  - F0 = A + B'C'
  - F1 = AC' + AB
  - F2 = B'C' + AB
  - F3 = B'C + A

- Temos:
  - e = 3 entradas {A,B,C}
  - s = 4 saídas {F0,F1,F2,F3}
  - p_d = 5 produtos distintos {A, B'C', AC', AB, B'C}
  - m_p = 2 produtos por função de saída, no máximo

Arranjos Lógicos Programáveis: Exemplo

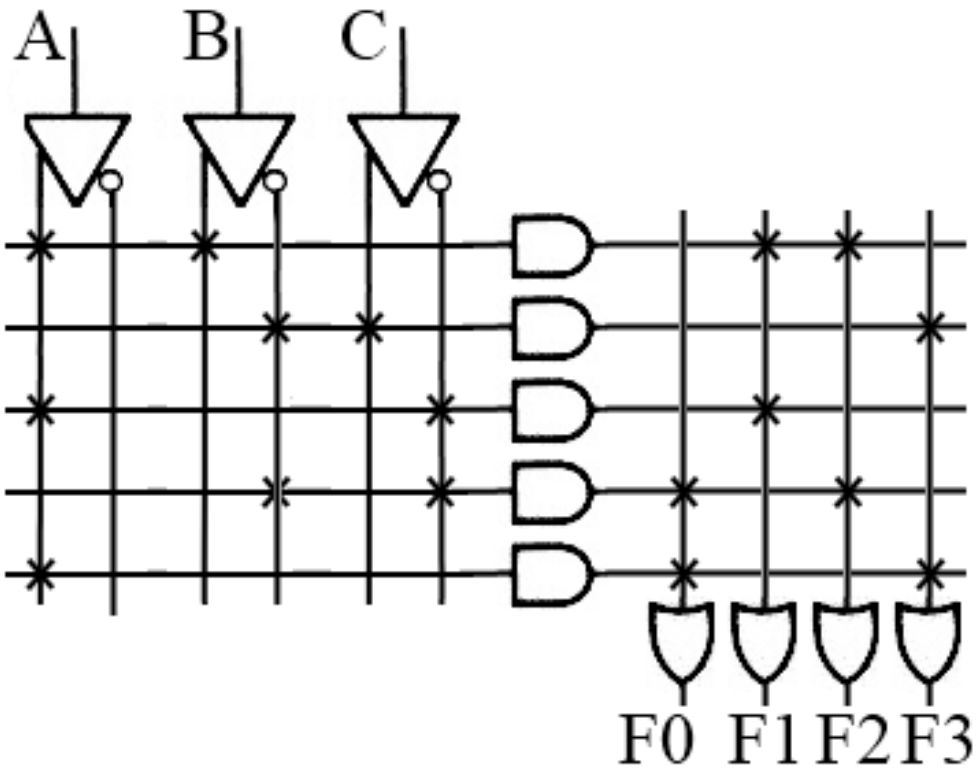
- Matriz das funções

Produto	Entrada			Saídas			
	A	B	C	F0	F1	F2	F3
AB	1	1	-	0	1	1	0
B'C	-	0	1	0	0	0	1
AC'	1	-	0	0	1	0	0
B'C'	-	0	0	1	0	1	0
A	1	-	-	1	0	0	1

Arranjos Lógicos Programáveis: Exemplo

- Implementação com PLA

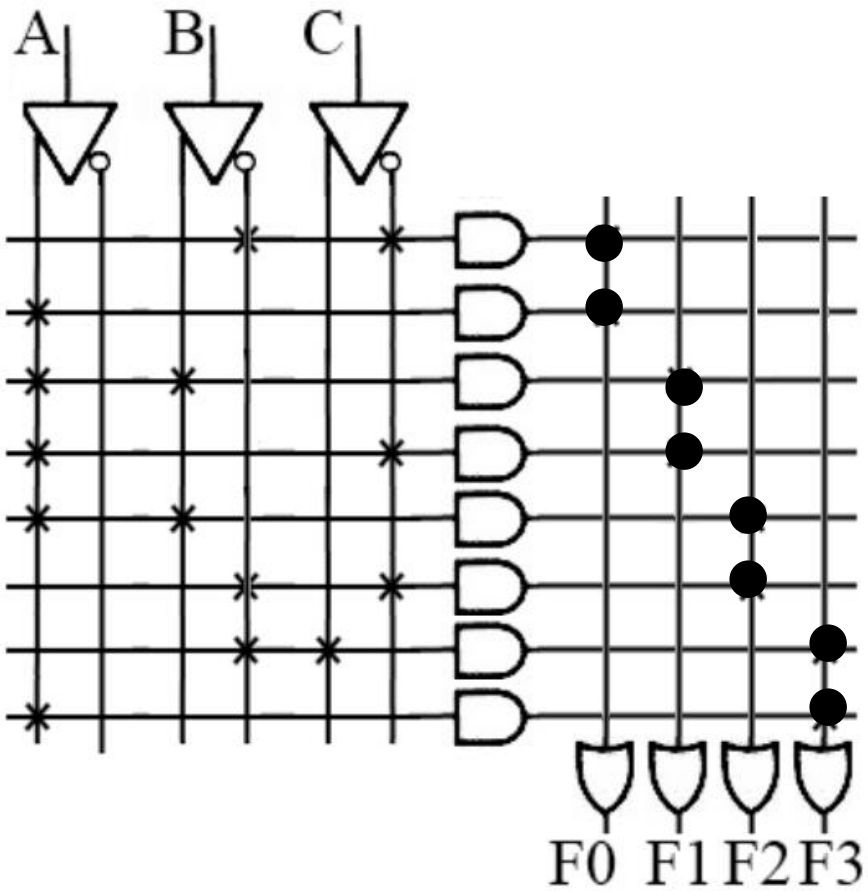
F0 = A + B'C'
F1 = AC' + AB
F2 = B'C' + AB
F3 = B'C + A



Arranjos Lógicos Programáveis: Exemplo

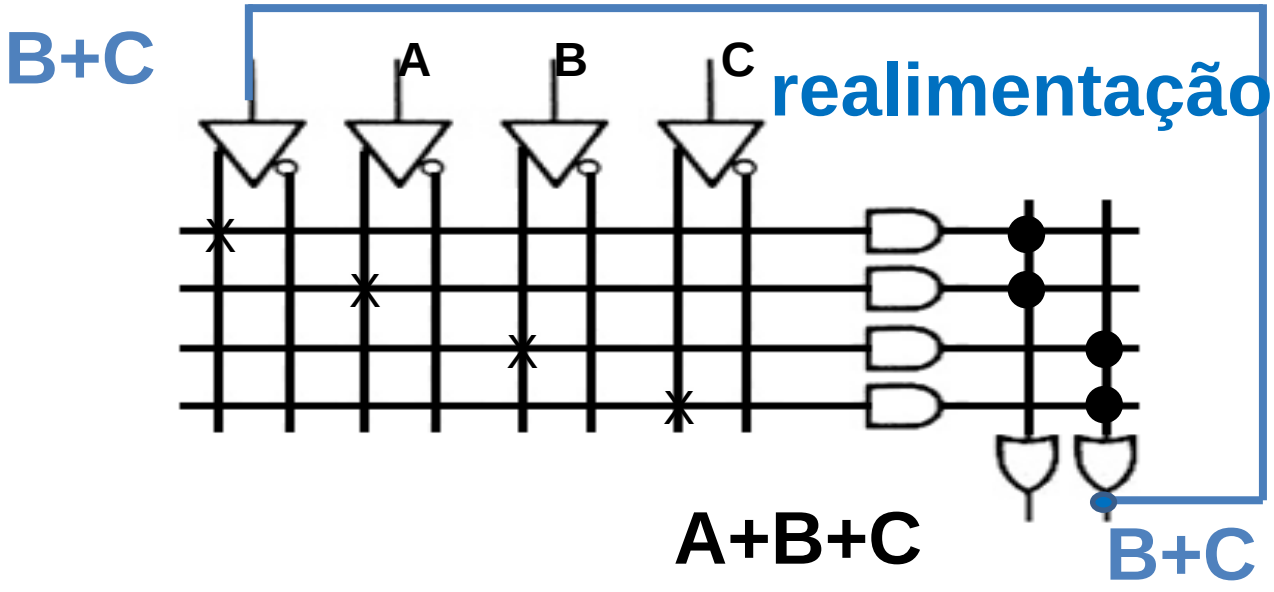
- Implementação com PAL

F0 = A + B'C'
F1 = AC' + AB
F2 = B'C' + AB
F3 = B'C + A



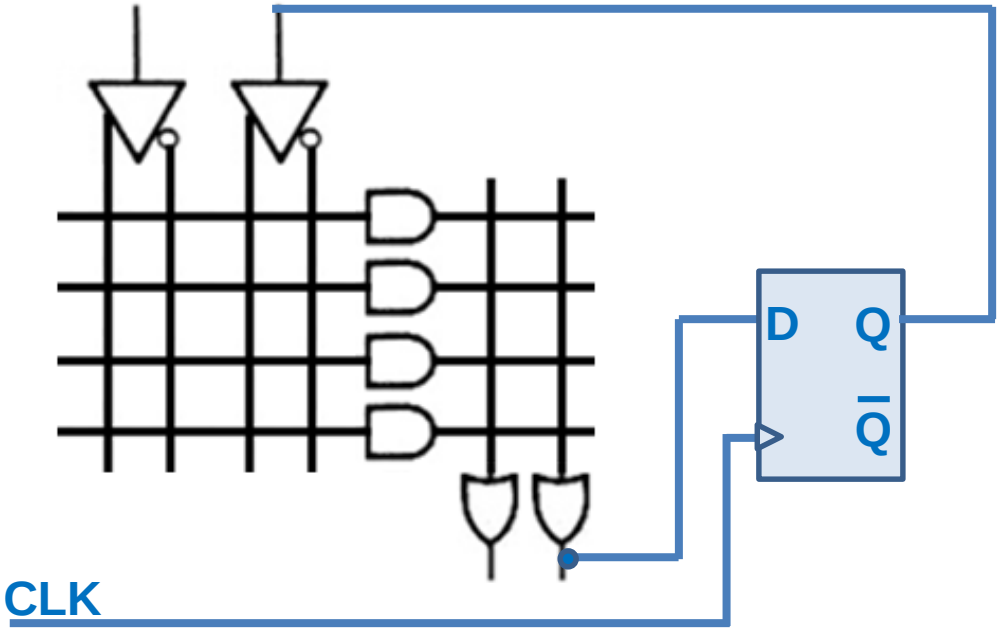
Arranjos Lógicos Programáveis: Extras

- Algumas PLAs e PALs comerciais apresentam funcionalidades adicionais, como:
  - Realimentação – Saída utilizada como entrada.



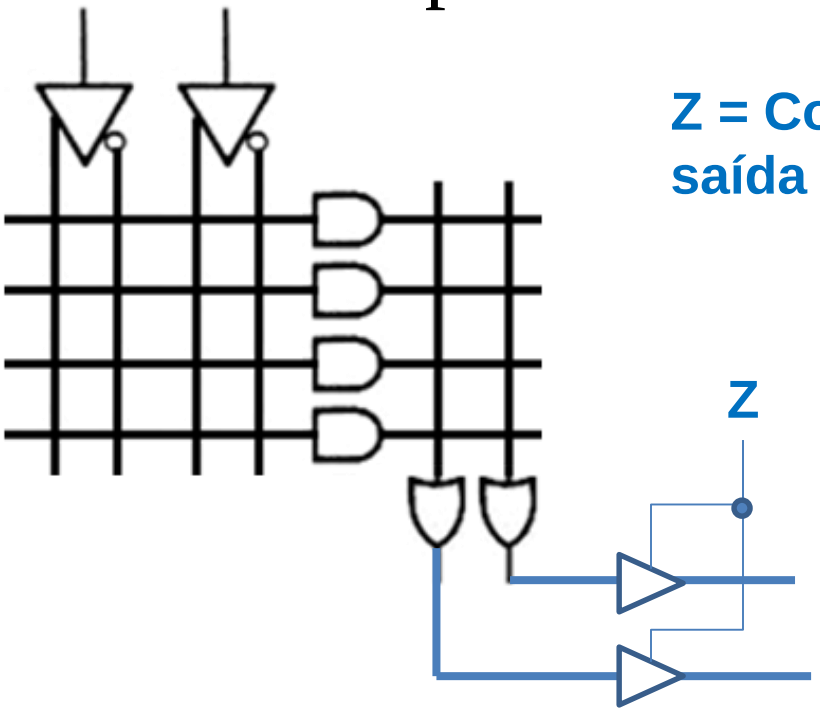
Arranjos Lógicos Programáveis: Extras

- Algumas PLAs e PALs comerciais apresentam funcionalidades adicionais, como:
  - Registradores na saída – Permitem sintetizar circuitos sequenciais.



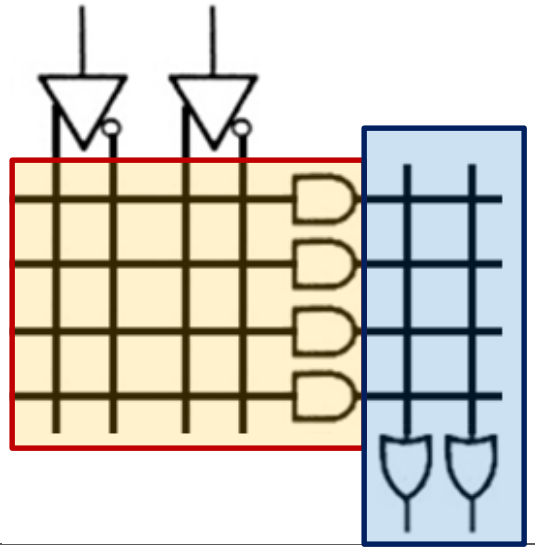
Arranjos Lógicos Programáveis: Extras

- Algumas PLAs e PALs comerciais apresentam funcionalidades adicionais, como:
  - Saídas *tri-state* – Permitem colocar a saída em estado de alta impedância.



Arranjos Lógicos Programáveis: Resumo

	Matriz AND	Matriz OR
PROM	fixa	programável
PLA	programável	programável
PAL	programável	fixa



Matriz de ANDs

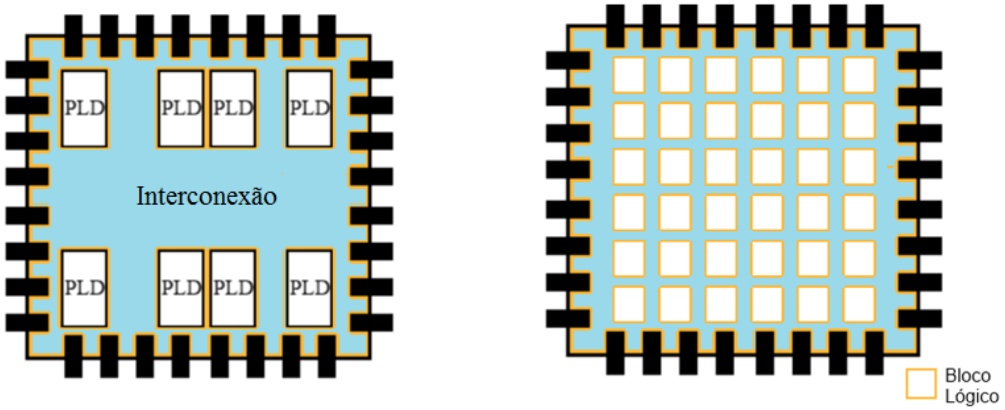
Matriz de ORs

PLDs modernos: CPLDs e FPGAs

- Com o avanço da microeletrônica, circuitos programáveis mais complexos foram sendo elaborados. Circuitos atuais se enquadram em duas categorias:
  - **CPLD – *Complex PLD*;**
  - **FPGA – *Field Programmable Gate Array*.**

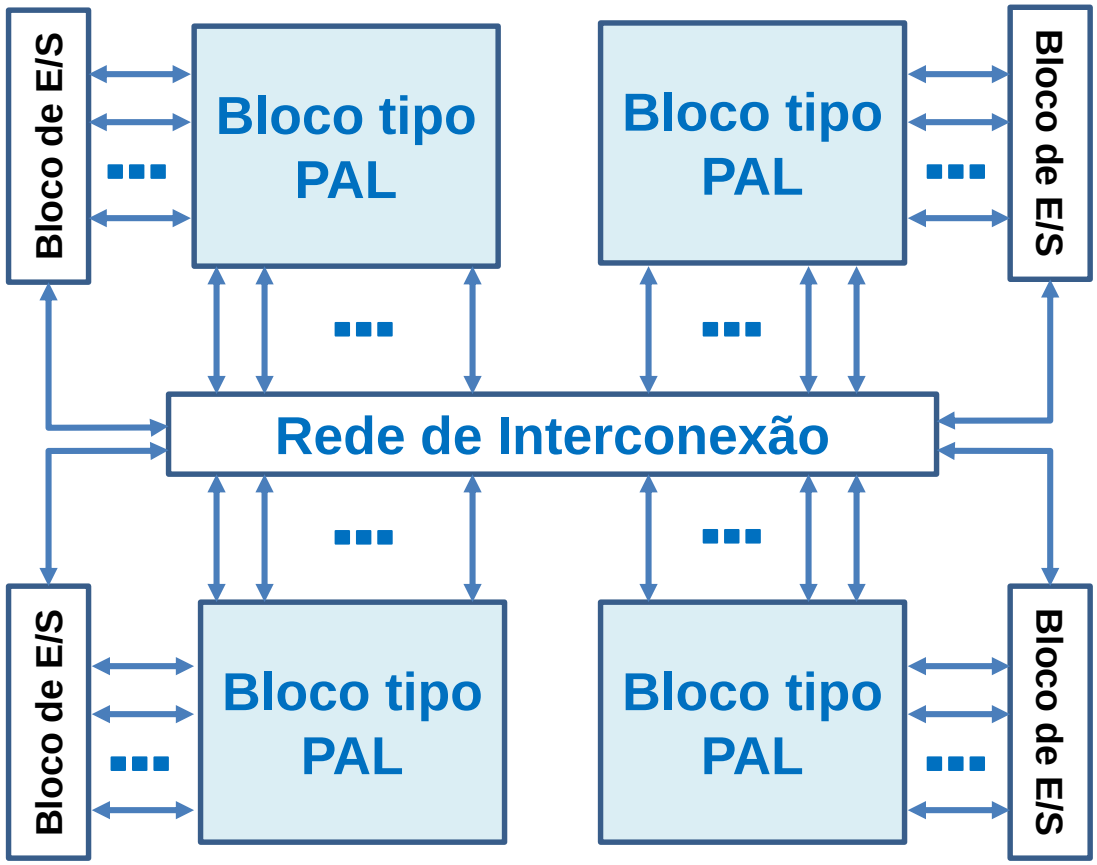
PLDs modernos: Características

- Grande quantidade de lógica programável – Flip-flops pré-implementados.
- Interconexões programáveis entre lógica programável, Flip-flops e entradas/saídas do dispositivo – É possível controlar o roteamento dos sinais dentro do circuito.



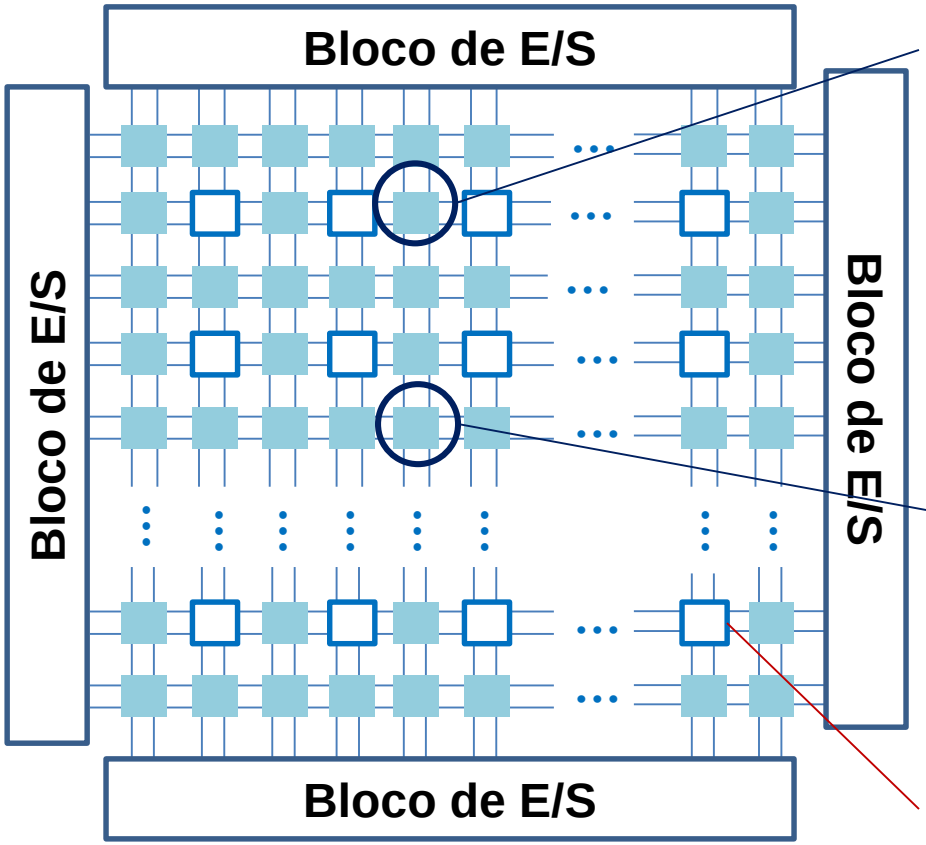
PLDs modernos: Estrutura

- CPLD – Estrutura contínua de conexão:
  - Em geral de 2 a 100 blocos tipo PAL.



PLDs modernos: Estrutura

- FPGA: estrutura segmentada de conexão



Chaves programáveis (ligações do bloco com linhas de interconexão)

Chaves programáveis (ligações entre linhas verticais e horizontais)

Bloco lógico

PLDs modernos: Comparação

- Complexidade:
  - PLAs e PALs são muito simples – Menos de 200 *gates* equivalentes;
  - CPLDs têm capacidade moderada – Até 100.000 *gates* equivalentes;
  - FPGAs atuais – Apresentam capacidade de mais de 1M de *gates* equivalentes.

PLDs modernos: Comparação

- Volatilidade:
  - CPLDs são dispositivos não voláteis, i.e., podem ser desligados sem perda de informação;
  - FPGAs são voláteis, i.e., quando desligadas perdem a informação;
  - Algumas FPGAs são associadas a ROMs para permitir que a informação seja recuperada ao ligar.
  - Atualmente FPGAs são mais utilizados.

PLDs modernos: Programação

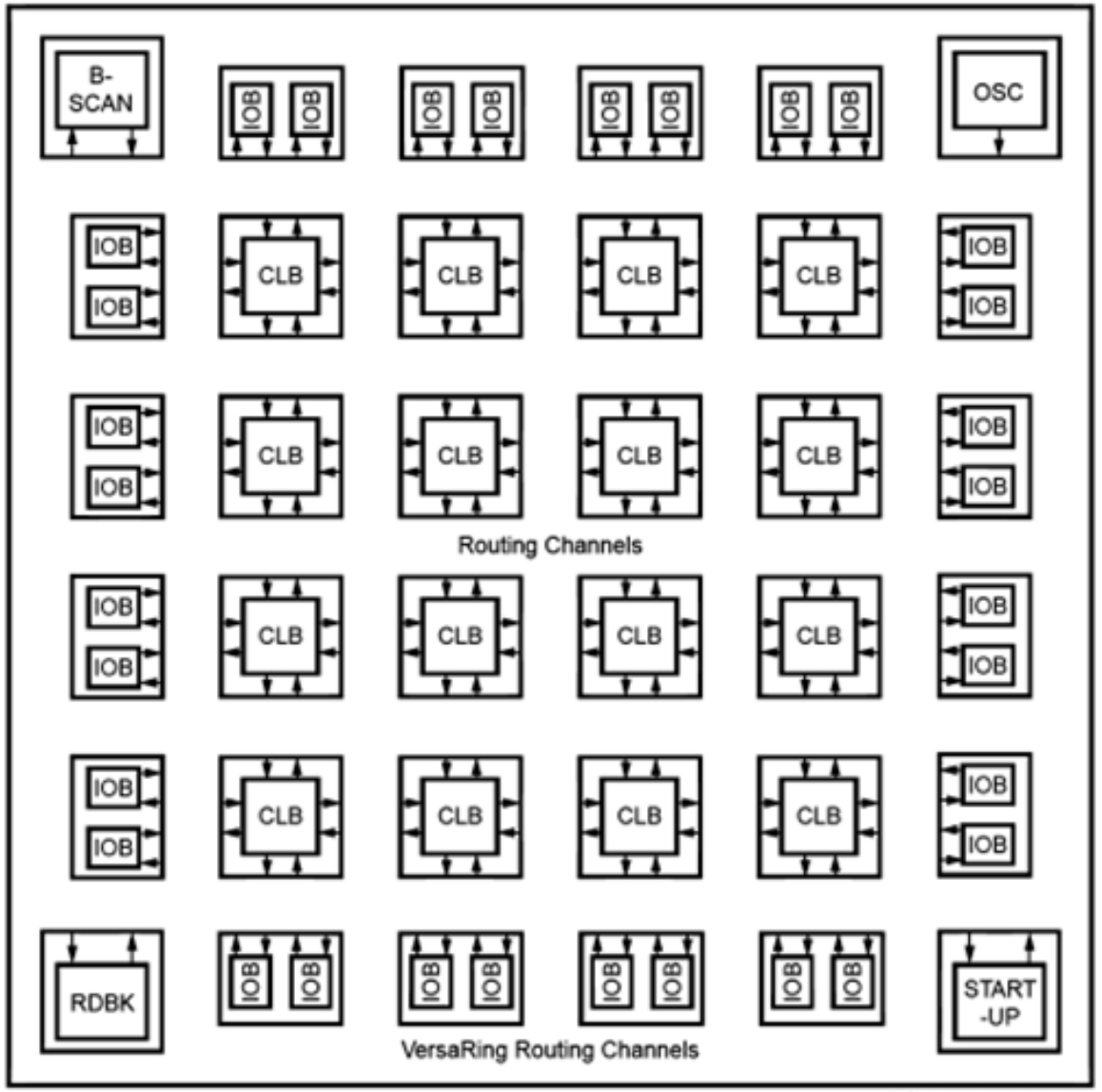
- Linguagens de programação próprias existentes – PALASM, CUPL, etc.
- Não são mais usadas devido às linguagens de descrição de Hardware de mais alto nível – VHDL, Verilog.
- As linguagens de alto nível são transcritas para fluxo de bits que é usado para programar o dispositivo.

FPGAs: Componentes

- Compostas por:
  - CLBs (*Configurable Logic Blocks*): blocos lógicos que executam funções lógicas
  - IOBs (*Input/Output Buffers*): interface com mundo externo
  - Interconexões programáveis: conecta CLBs e IOBs

FPGAs: Componentes

- Diagrama de blocos do Xilinx Spartan

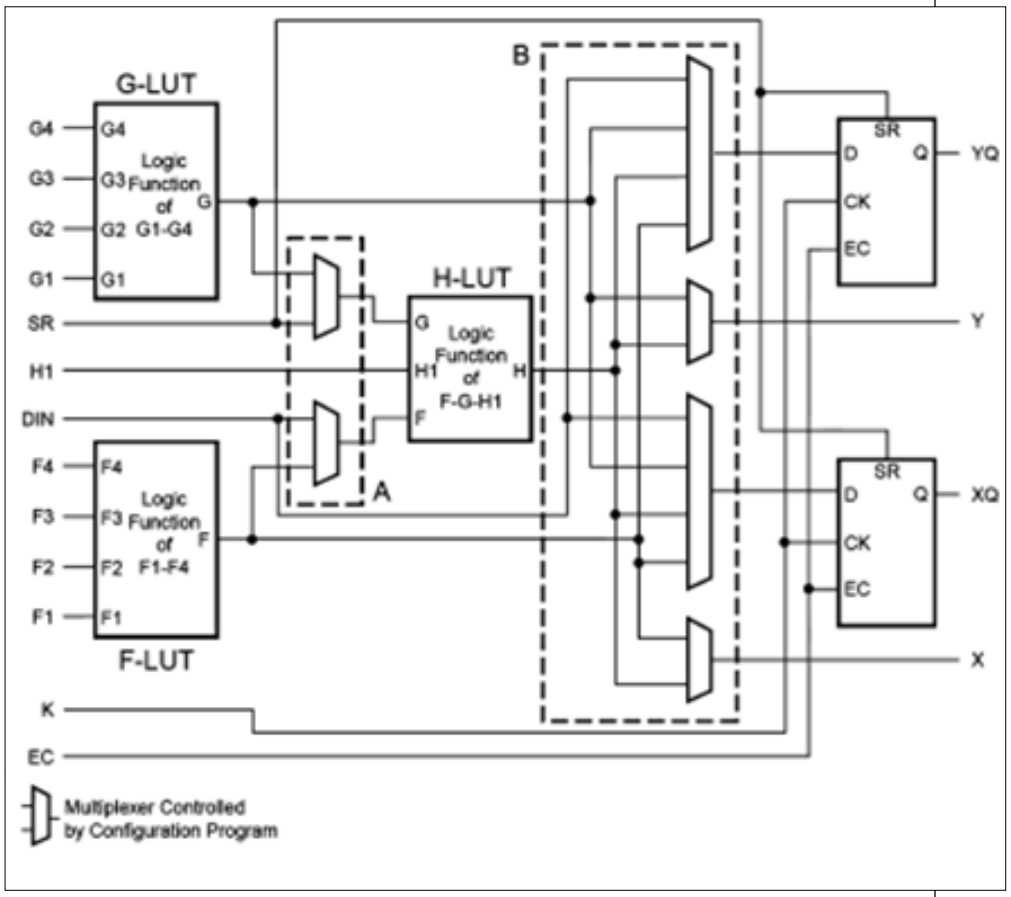


FPGAs: CLBs

- CLBs por sua vez são compostos de:
  - LUTs (*lookup tables*): implementam lógica combinatória
  - Flip-flops: implementam funções sequenciais
  - Multiplexadores: conectam LUTs e flip-flops

FPGAs: CLBs

- 3 LUTs:
  - LUT-F (16 x 1-bit)
  - LUT-G (16 x 1-bit)
  - LUT-H (8 x 1-bit)
- 2 saídas “registradas”
  - XQ e YQ
- 2 saídas combinatórias
  - X e Y



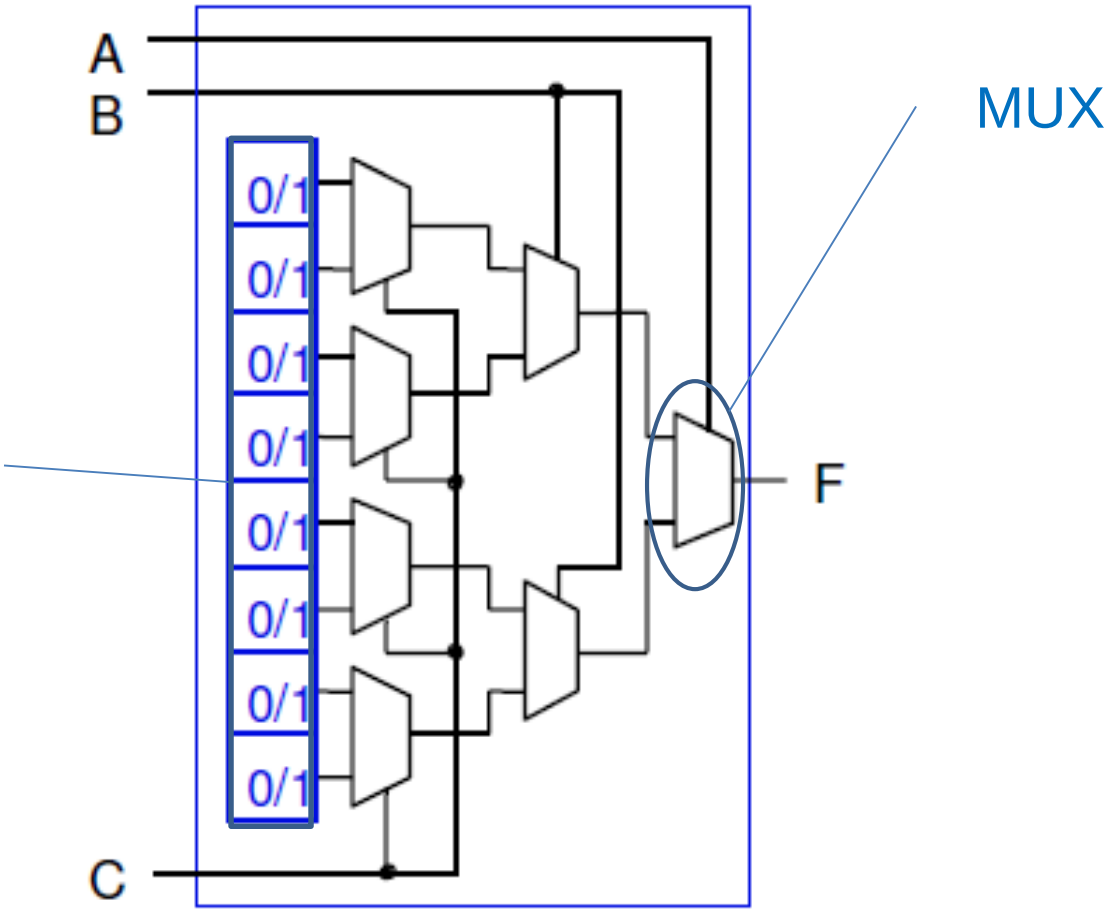
Xilinx Spartan

FPGAs: LUTs

- LUT-N: uma tabela-verdade de N entradas e 2^N posições de 1 bit.
  - Programar uma LUT significa preencher os valores da tabela-verdade.
  - LUT com 4 entradas ➔ 16 posições possíveis
    - Pode emular 2^{16} (~64 mil) funções booleanas distintas.
  - FPGAs comerciais podem conter LUTs de até 6 entradas (mais de 16 bilhões de funções emuláveis).
- Implementadas usando MUXes:

FPGAs: LUTs

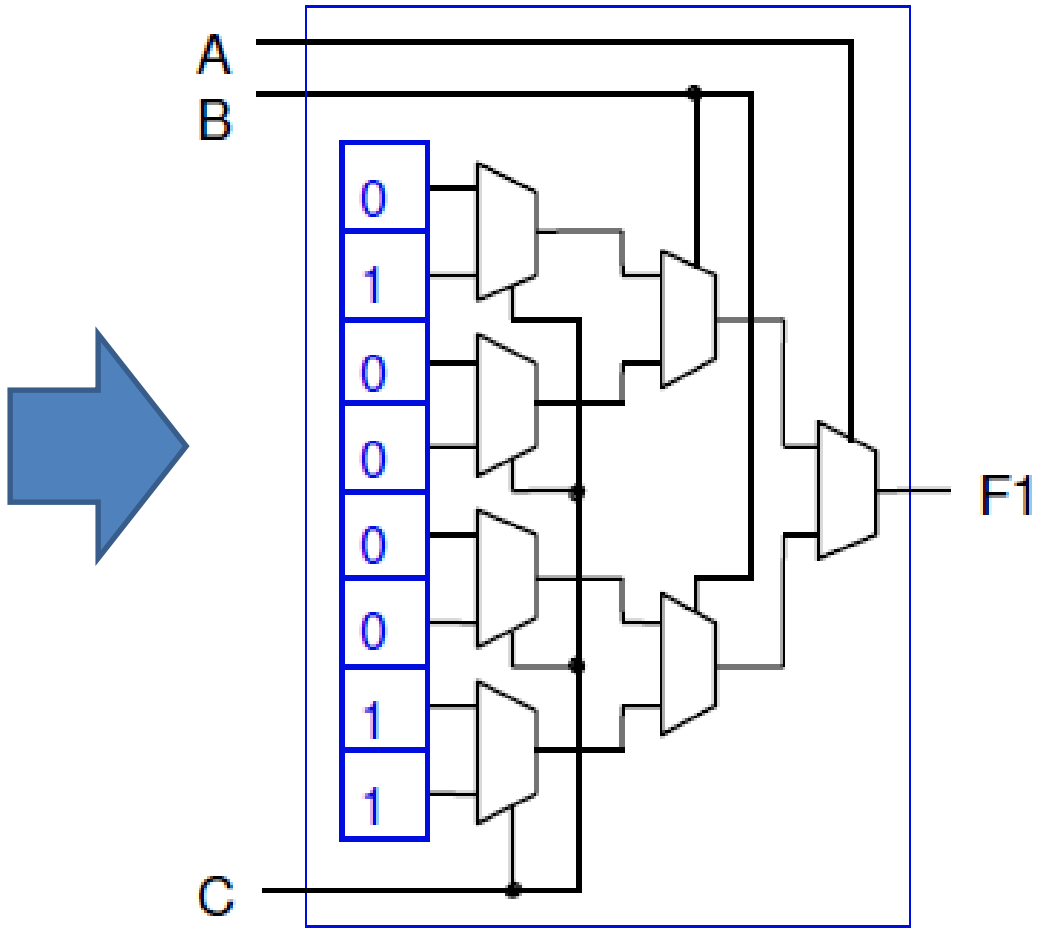
- Exemplo 1:



FPGAs: LUTs

- Exemplo 1: Programação de F1

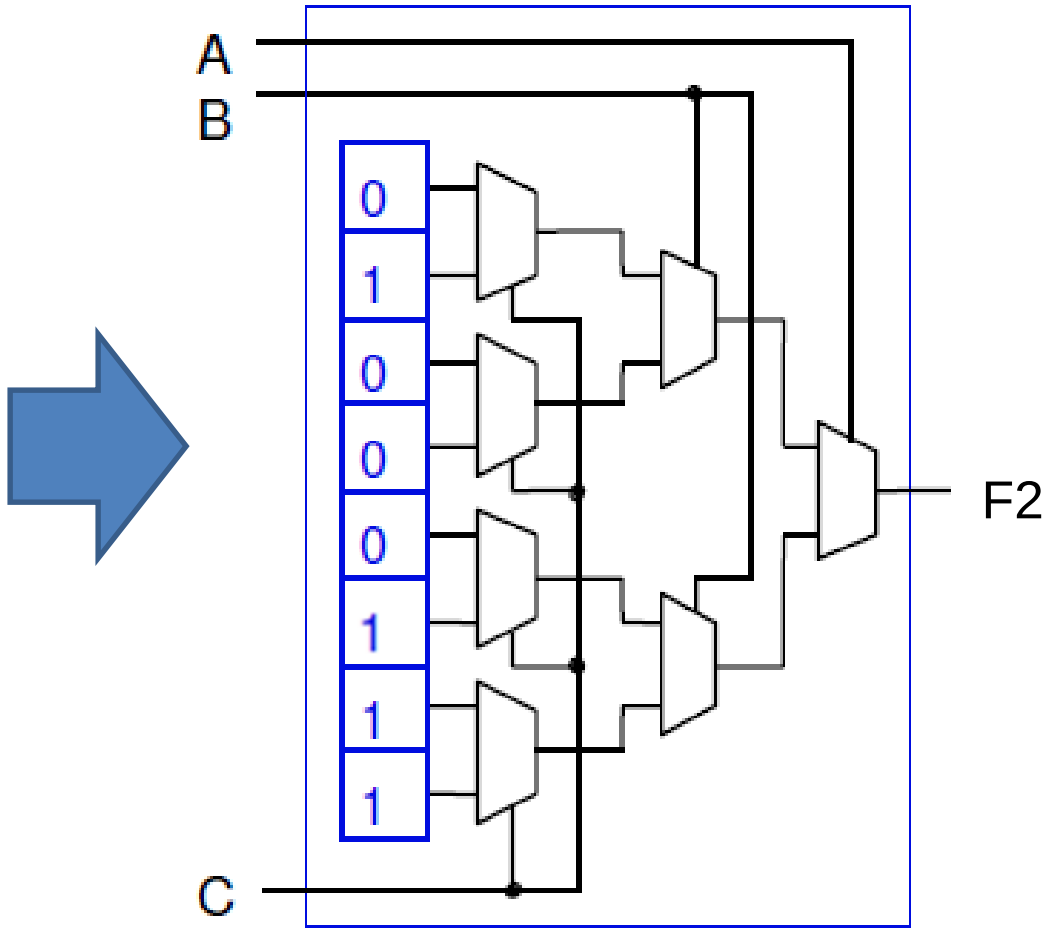
A	B	C	F1	F2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1



FPGAs: LUTs

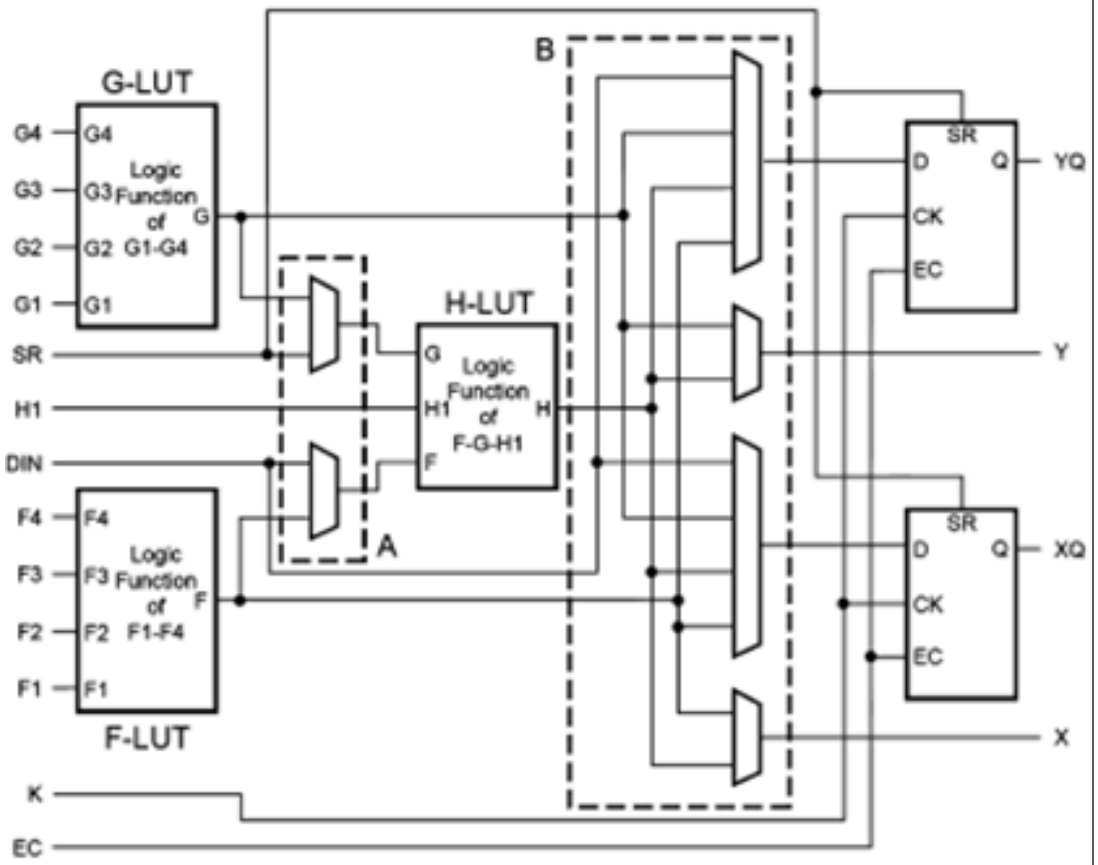
- Exemplo 1: Programação de F2

A	B	C	F1	F2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1



FPGAs: LUTs

- Exemplo 2: programação das funções X e Y na Xilinx Spartan:
  - X = A'B'C + ABC'
  - Y = AB'



FPGAs: LUTs

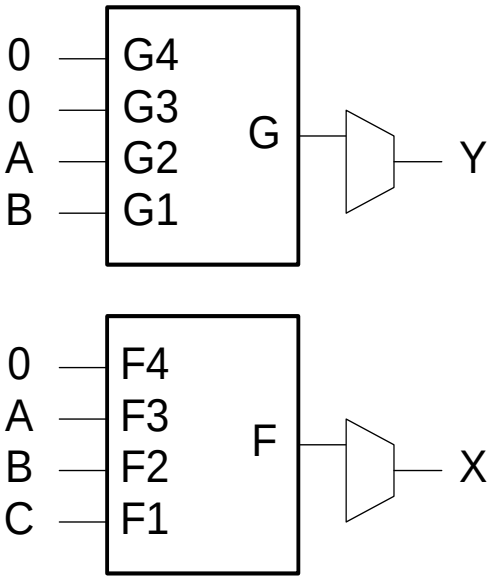
- Exemplo 2: programação das funções X e Y na Xilinx Spartan:
  - X = A'B'C + ABC'
  - Y = AB'

	(A)	(B)	(C)	(X)
F4	F3	F2	F1	F
X	0	0	0	0
X	0	0	1	1
X	0	1	0	0
X	0	1	1	0
X	1	0	0	0
X	1	0	1	0
X	1	1	0	1
x	1	1	1	0

Programa na LUT-F

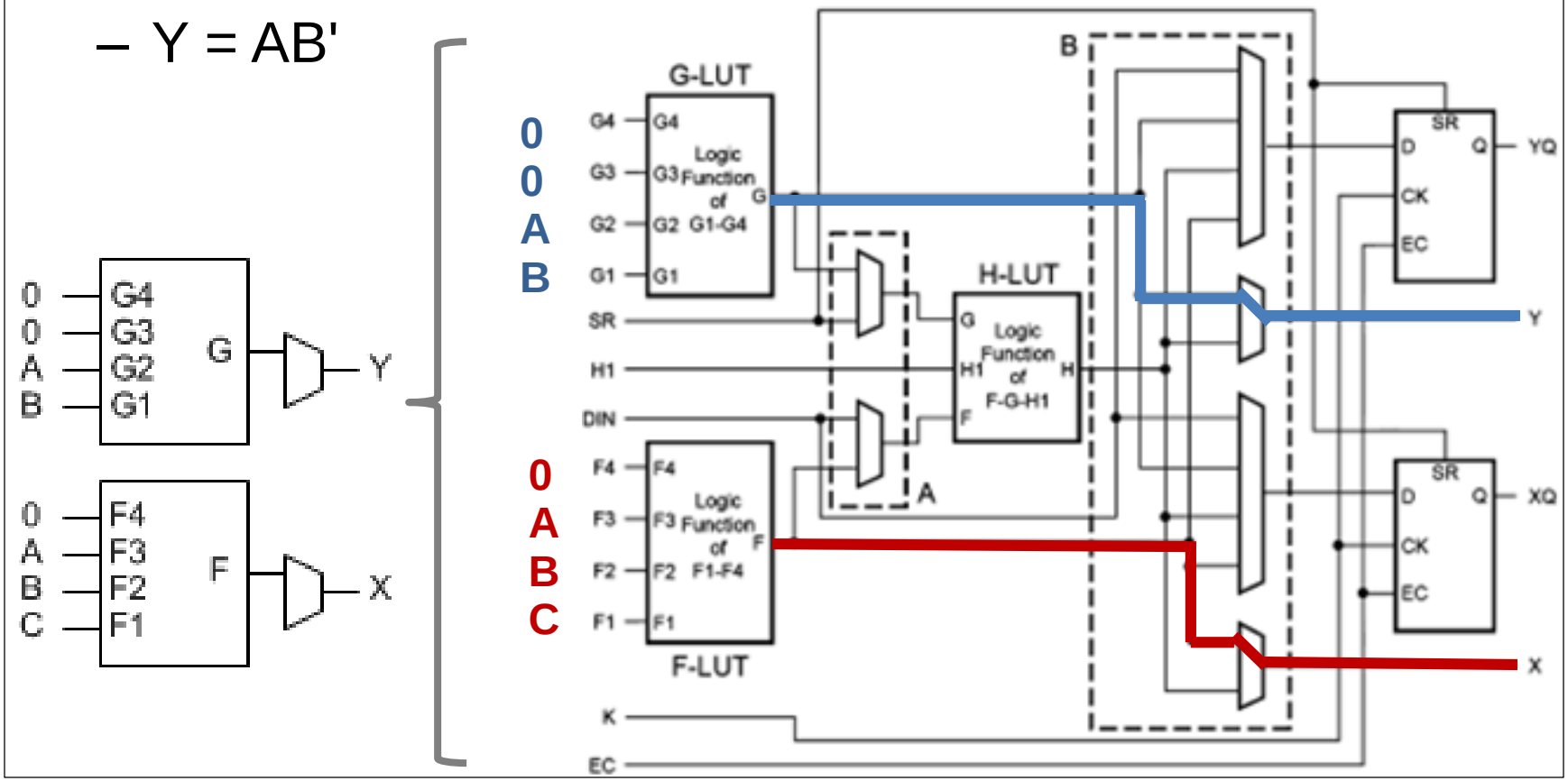
		(A)	(B)	(Y)
G4	G3	G2	G1	G
X	X	0	0	0
X	X	0	1	0
X	X	1	0	1
X	X	1	1	0

Programa na LUT-G



FPGAs: LUTs

- Exemplo 2: programação das funções X e Y na Xilinx Spartan:
  - X = A'B'C + ABC'
  - Y = AB'



FPGAs: LUTs

- LUTs de 4 entradas: como calcular funções com maior número de variáveis?
  - Resposta: associando 2 ou mais LUTs.
- Por exemplo, vamos calcular a função:
  - $X = A \bullet B \bullet C \bullet D \bullet E \bullet F \bullet G \bullet H$
- Para isso precisamos inicialmente dividir a função em subfunções:
  - $X_1 = A \bullet B \bullet C \bullet D$
  - $X_2 = E \bullet F \bullet G \bullet H$
  - $X = X_1 \bullet X_2$

3 LUTs

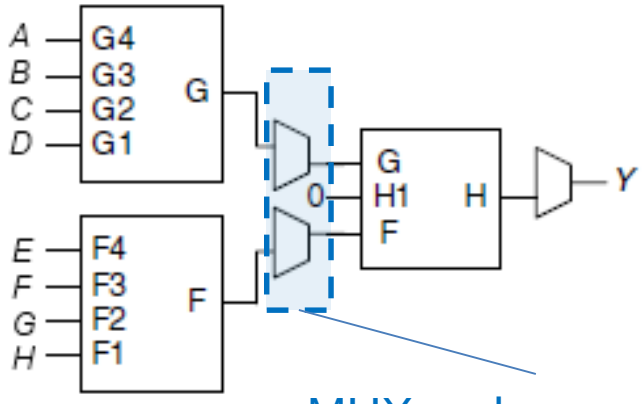
FPGAs: LUTs

- Vamos configurar:
  - LUT-G para calcular $X_1 = A \bullet B \bullet C \bullet D$
  - LUT-F para calcular $X_2 = E \bullet F \bullet G \bullet H$
  - LUT-H para calcular $X = X_1 \bullet X_2$

F4	F3	F2	F1	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

G4	G3	G2	G1	G
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

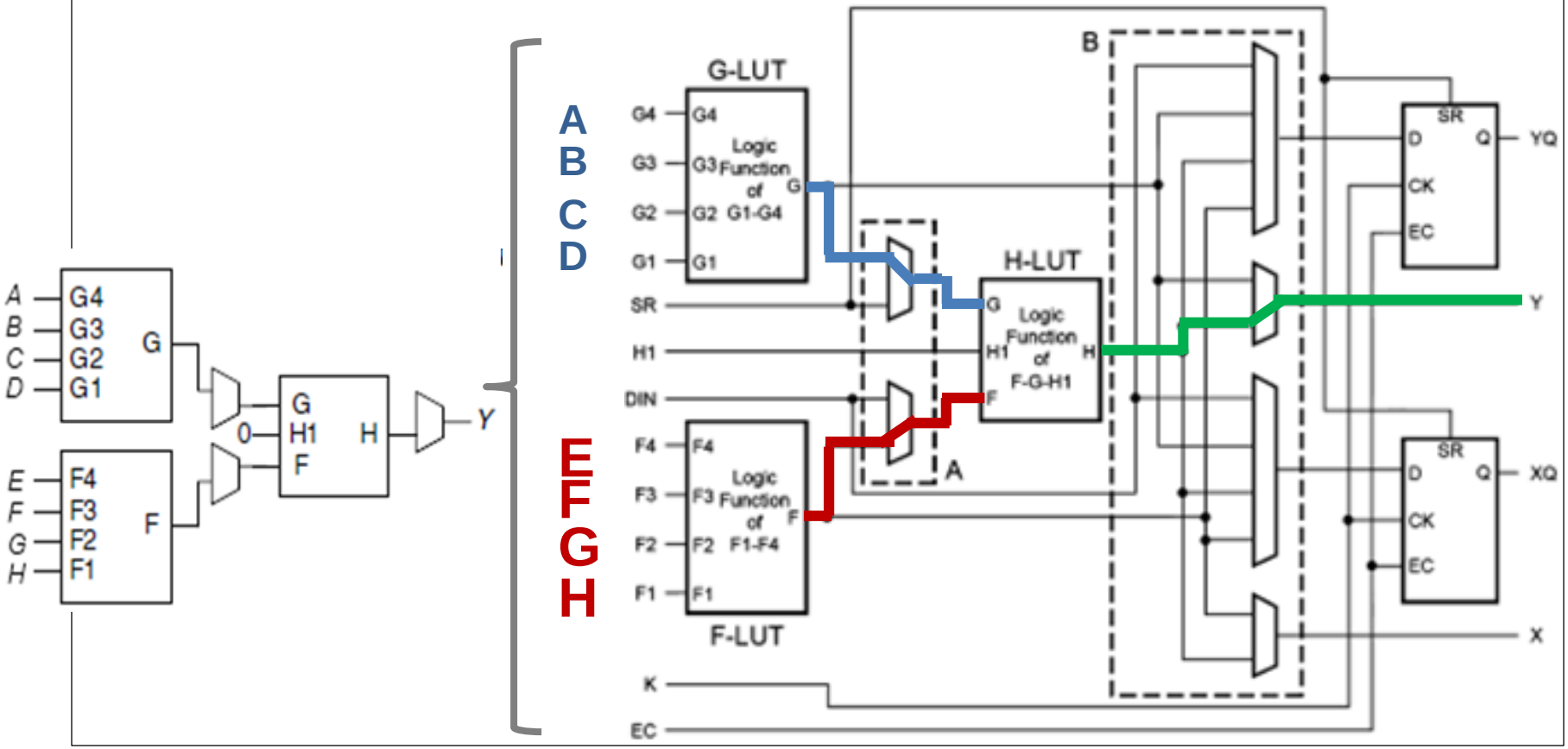
H1	F	G	H
x	0	0	0
x	0	1	0
x	1	0	0
x	1	1	1



MUXes de passagem devem ser configurados.

FPGAs: LUTs

- LUT-G: A•B•C•D
- LUT-F: E•F•G•H
- LUT-H: A•B•C•D•E•F•G•H

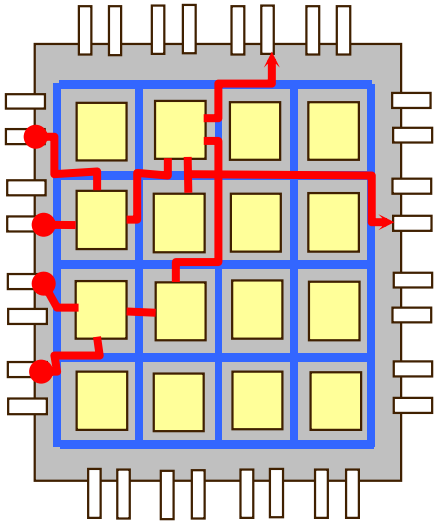


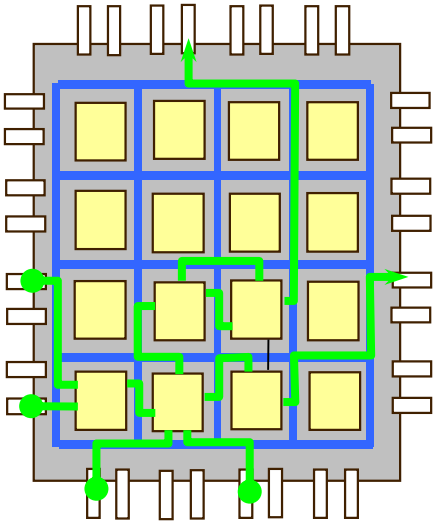
FPGAs: LUTs

- Usando associações, cada CLB Spartan pode calcular funções de até 9 variáveis.
- E para calcular funções de mais variáveis?
  - **Associam-se alguns CLBs**, por meio das saídas combinatórias.
- Nota: ao se associar diversos CLBs, existe uma **redução na frequência máxima**
  - Afinal, sinal deverá atravessar um número maior de LUTs...

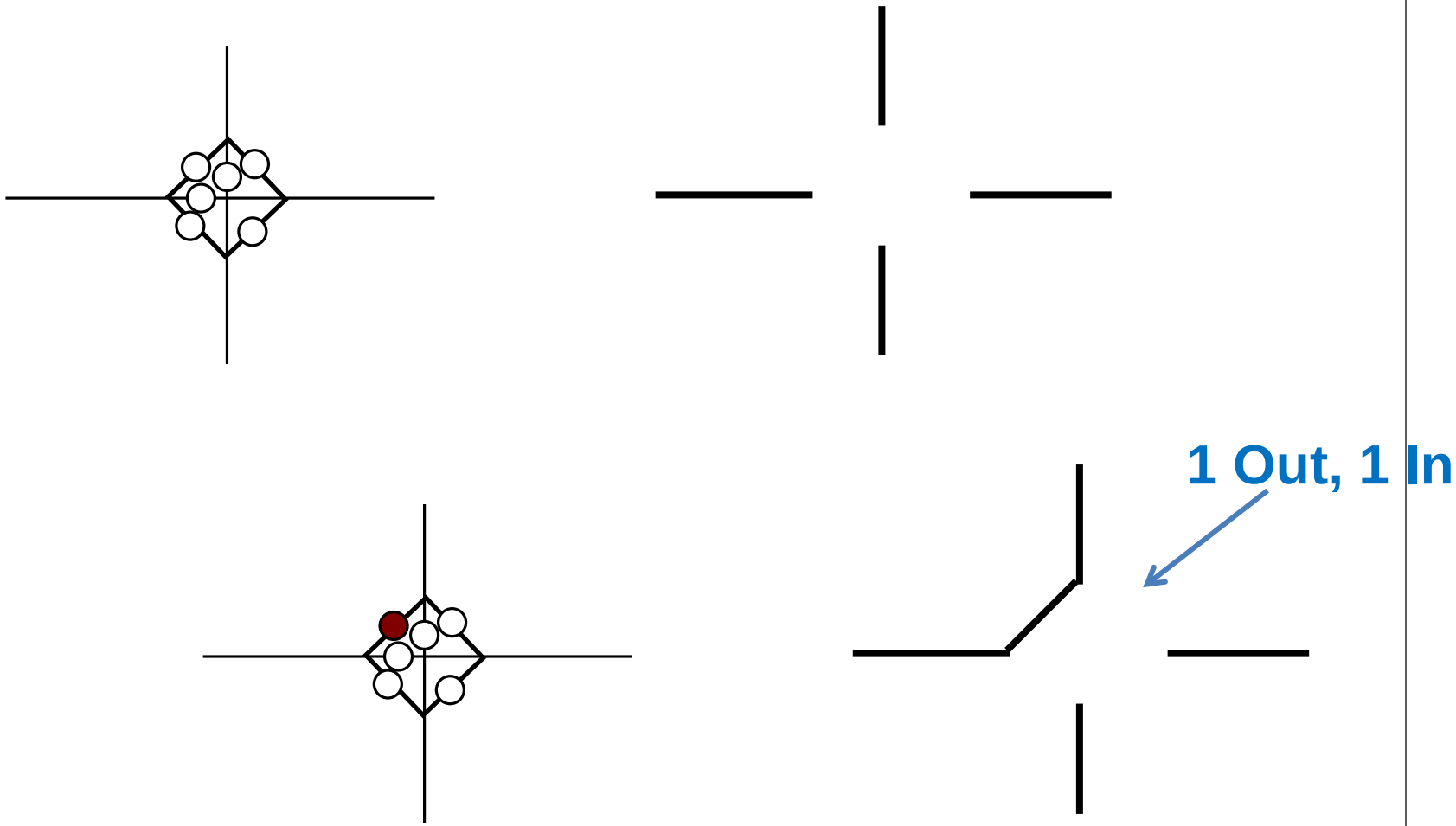
FPGAs: Interconexões Programáveis

- Permitem conexão entre os CLBs
- Configuráveis de acordo com necessidade

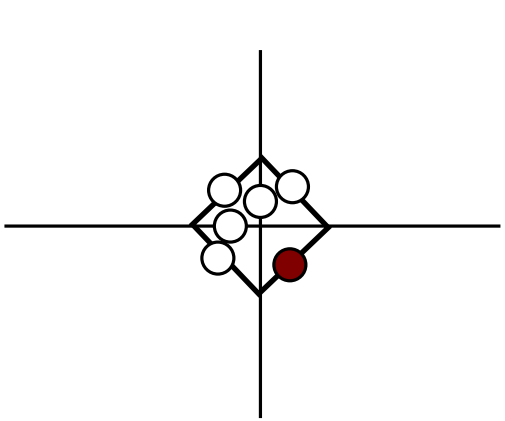


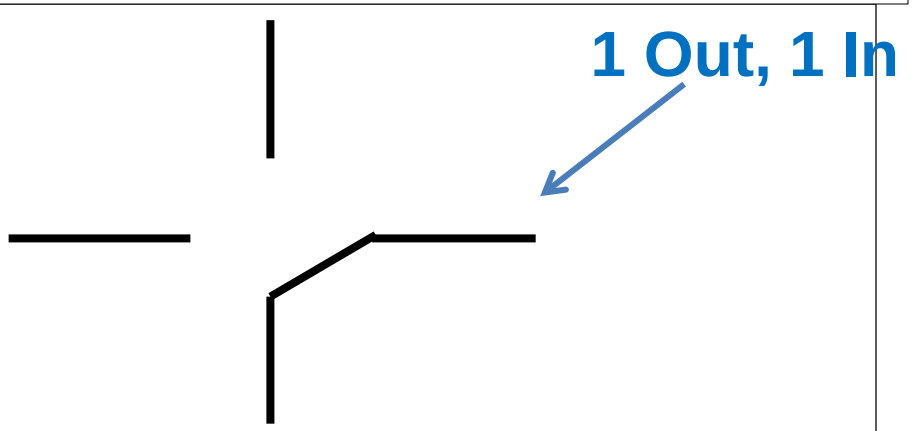


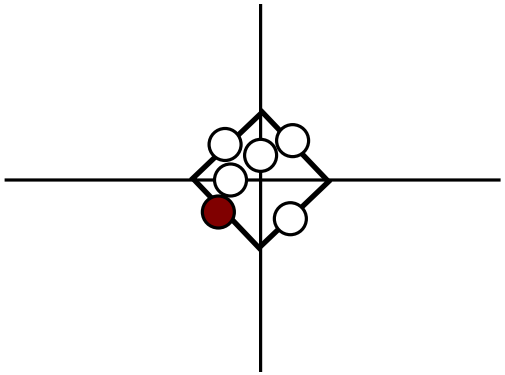
FPGAs: Interconexões Programáveis

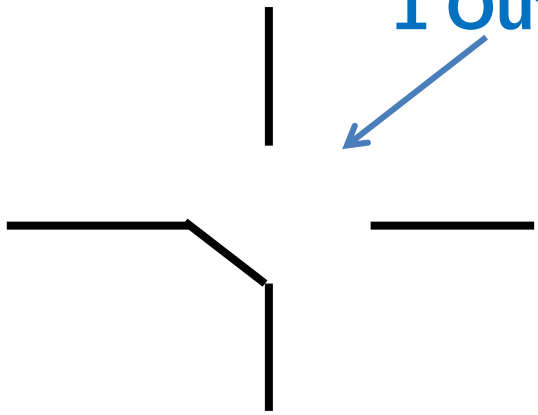


FPGAs: Interconexões Programáveis







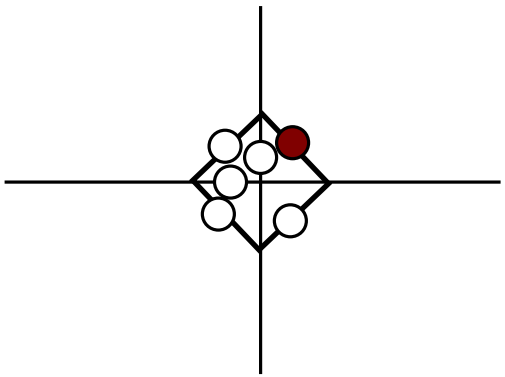


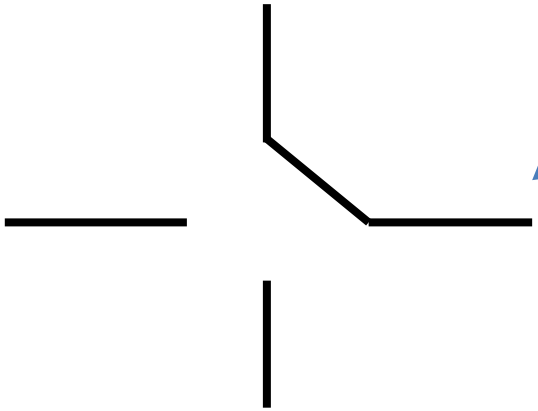
● **On**
○ **Off**

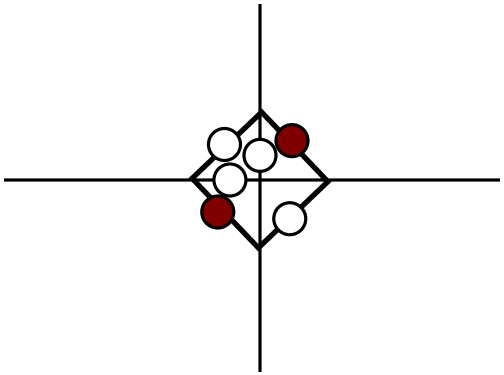
} Transmissor MOS

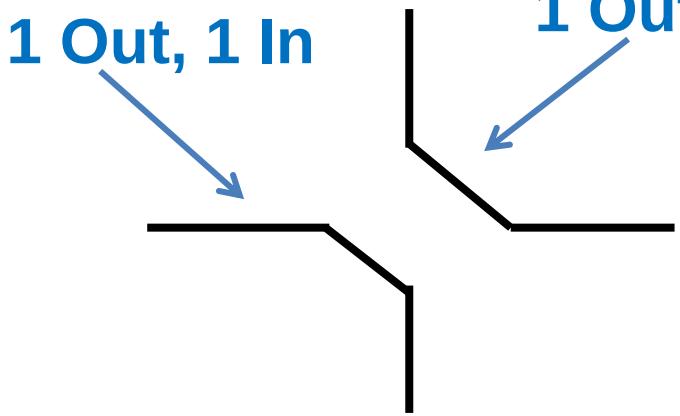
* Ligar duas saídas pode gerar curtos!

FPGAs: Interconexões Programáveis







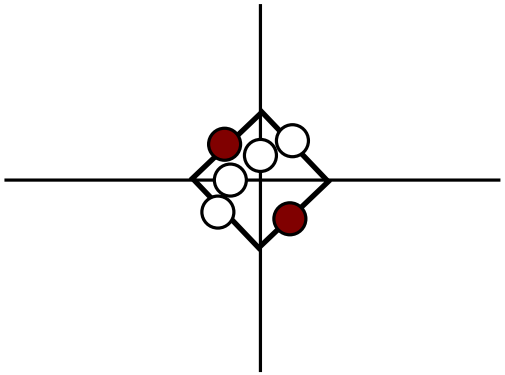


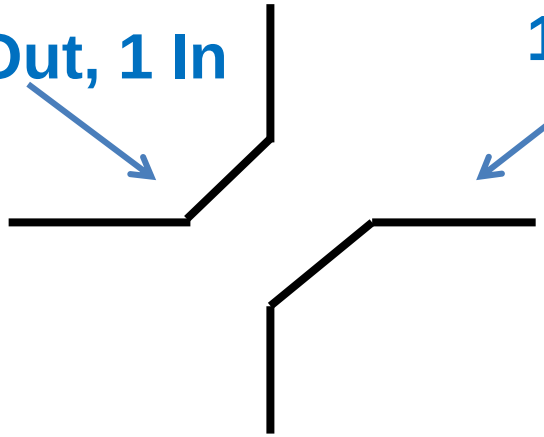
● **On**
○ **Off**

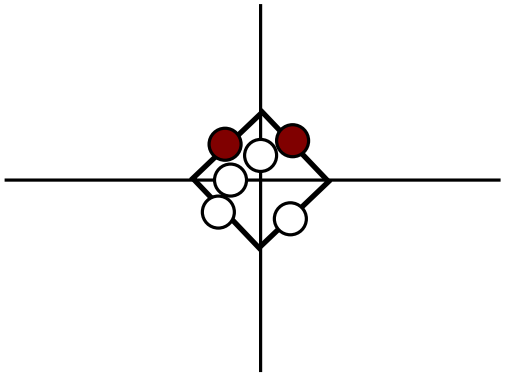
Transmissor MOS

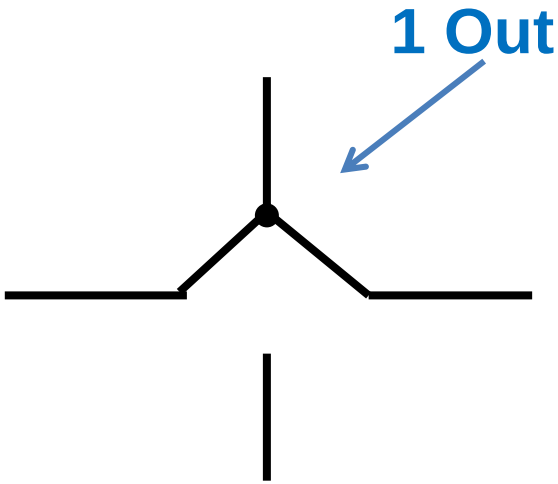
* Ligar duas saídas pode gerar curtos!

FPGAs: Interconexões Programáveis







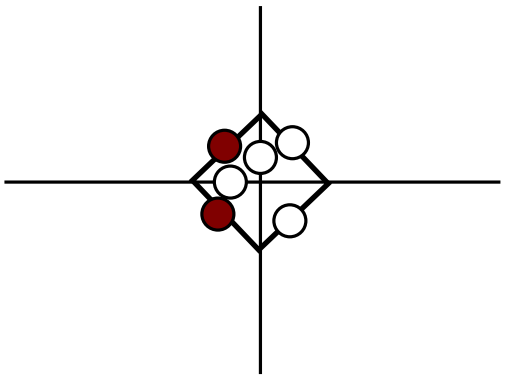


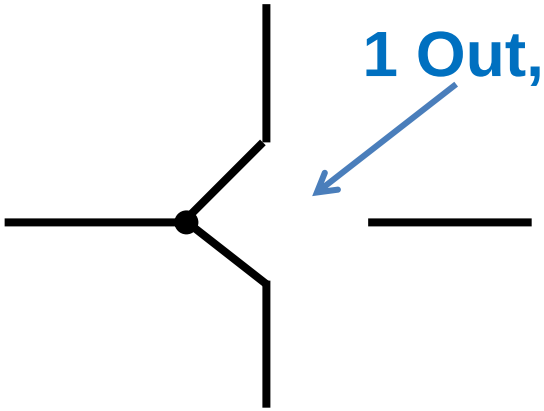
● **On**
○ **Off**

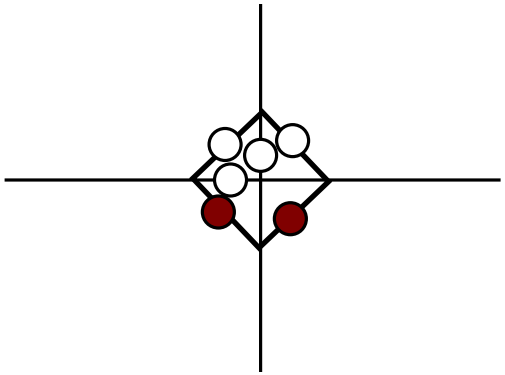
Transmissor MOS

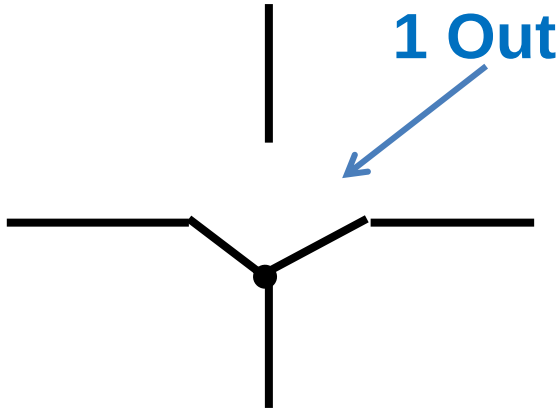
* Ligar duas saídas pode gerar curtos!

FPGAs: Interconexões Programáveis







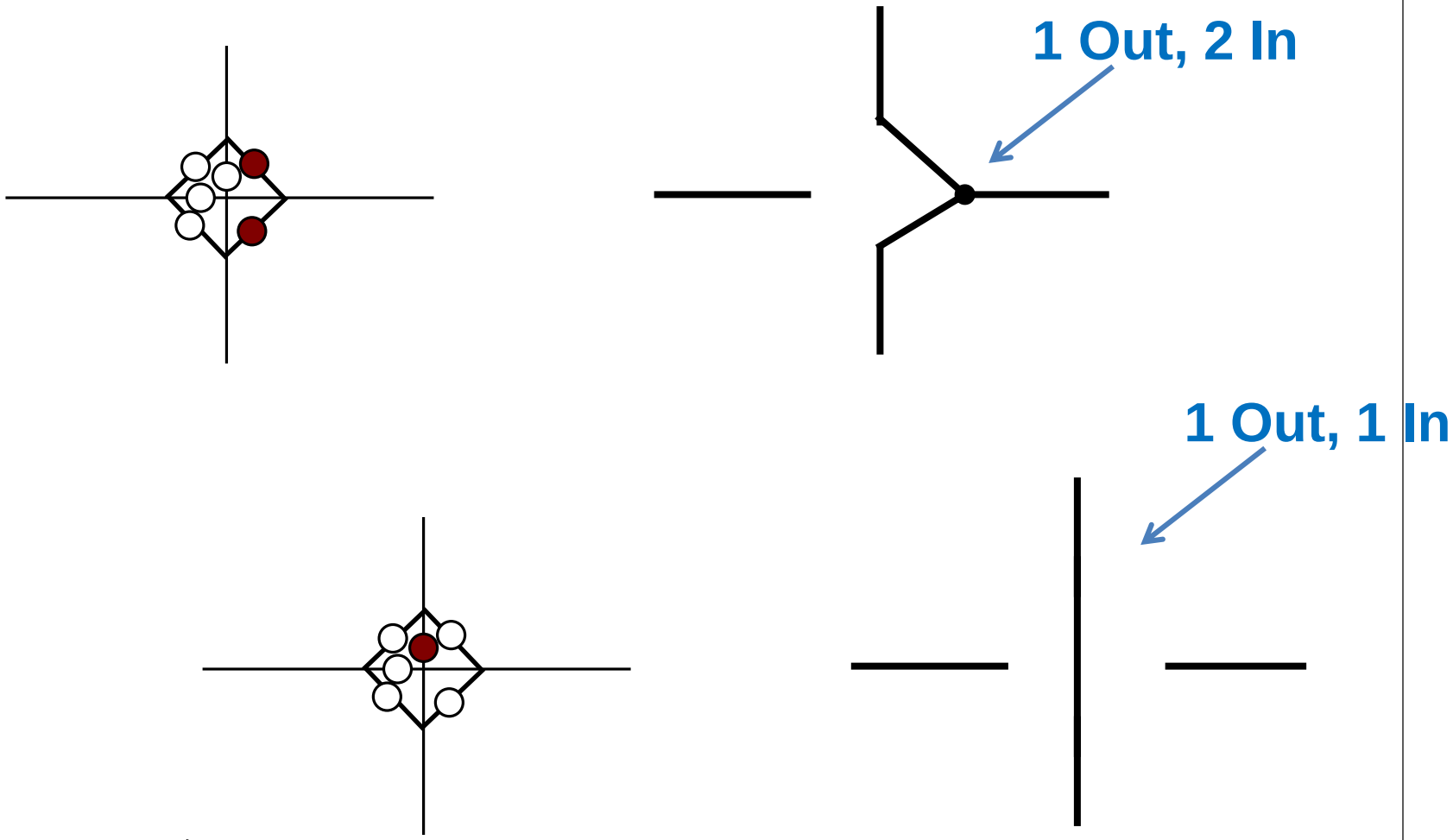


● **On**
○ **Off**

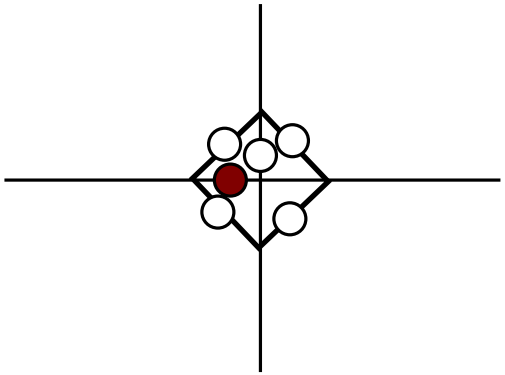
Transmissor MOS

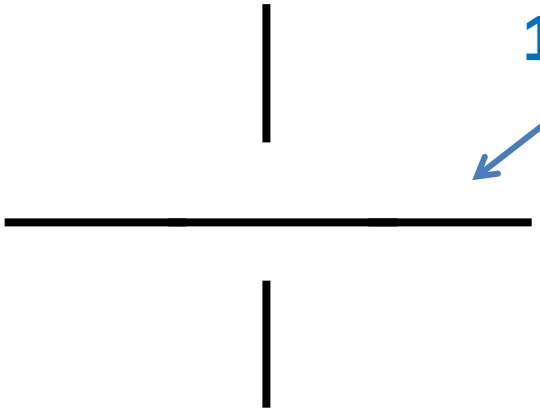
* Ligar duas saídas pode gerar curtos!

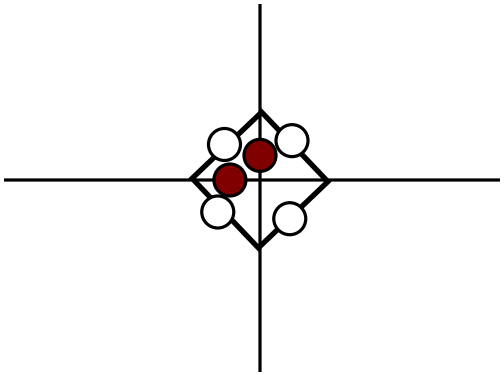
FPGAs: Interconexões Programáveis

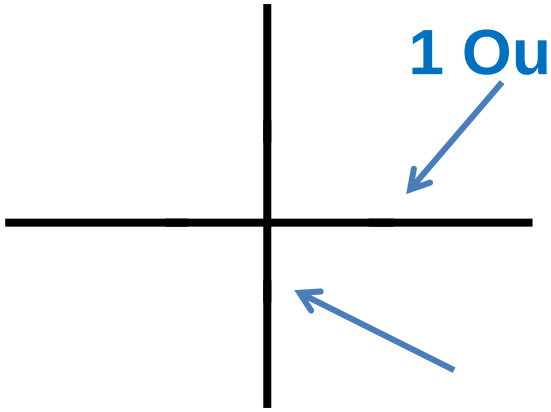


FPGAs: Interconexões Programáveis







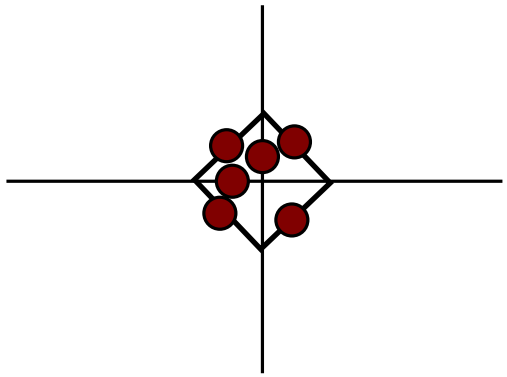


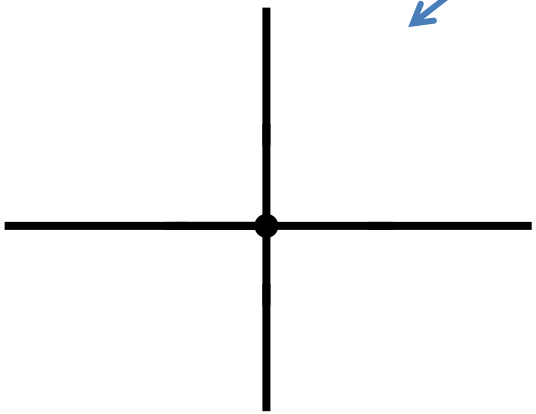
● On
○ Off

} Transmissor MOS

* Ligar duas saídas pode gerar curtos!

FPGAs: Interconexões Programáveis





1 Out, 3 In

● **On**
○ **Off**

} Transmissor MOS

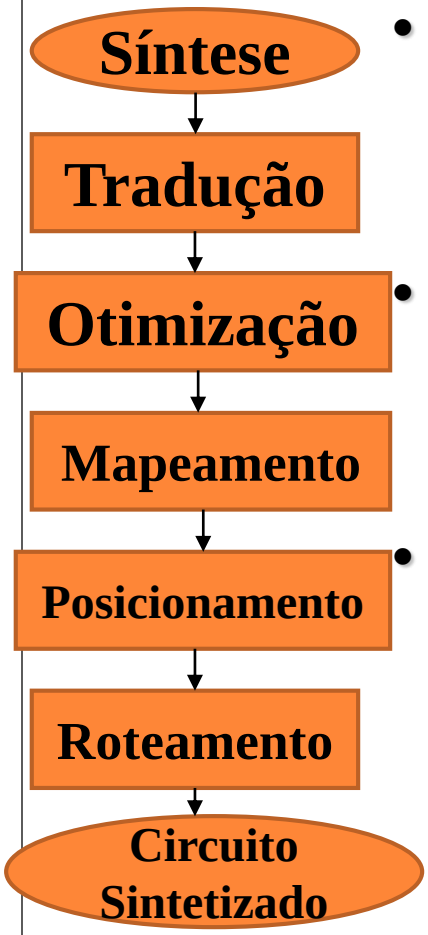
* Ligar duas saídas pode gerar curtos!

Projeto Lógico com FPGAs

- O projeto lógico com FPGA em geral envolve o seguinte fluxo de atividades:
  - Uso de ferramenta CAD para projeto e implementação de sistema digital, com entrada através de desenho esquemático e/ou HDL.
  - Usuário simula o projeto.
  - Ferramenta de síntese converte código para hardware e mapeia na FPGA.
  - Ferramenta faz download da configuração na FPGA.
    - Isso programa CLBs e conexões entre CLBs e IOBs.

Projeto Lógico com FPGAs

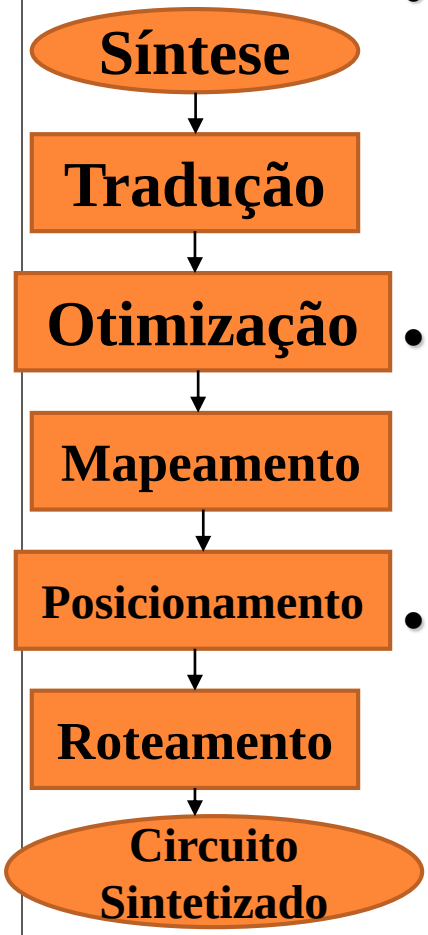
- A síntese de código HDL é feita seguindo os passos **[1/2]**:



- **Tradução** – Comandos VHDL são traduzidos para blocos de circuito lógico seguindo padrões pré-definidos pelo programa de síntese.
- **Otimização** – Blocos padronizados são analisados com o intuito de otimizar o circuito sintetizado segundo critérios estabelecidos pelo projetista.
- **Mapeamento** – Circuito lógico é mapeado nos componentes básicos disponíveis na tecnologia escolhida. Os blocos lógicos são mapeados nos blocos típicos da FPGA do componente alvo.

Projeto Lógico com FPGAs

- A síntese de código HDL é feita seguindo os passos **[2/2]**:



- **Posicionamento** – Blocos lógicos da FPGA identificados na etapa anterior são posicionados dentro daqueles disponíveis na FPGA alvo.
- **Roteamento** – Interligações entre os blocos lógicos previamente posicionados criam o circuito final.
- **Circuito Sintetizado**: Obtenção do circuito propriamente dito (não é um dos passos da síntese).

Projeto Lógico com FPGAs: VHDL

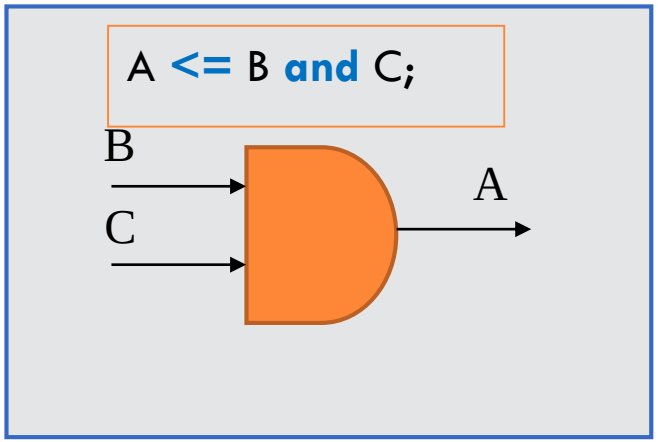
- A tradução de comandos VHDL para implementação em FPGA segue as seguintes regras **[1/2]**:
  - **Atribuições** – Pode implicar na utilização de um “FF-Latch” mesmo quando não desejado.
  - **Case** – Usa multiplexadores para compor os casos especificados. Se nem todas as alternativas forem especificadas, a síntese usa também um FF-Latch.

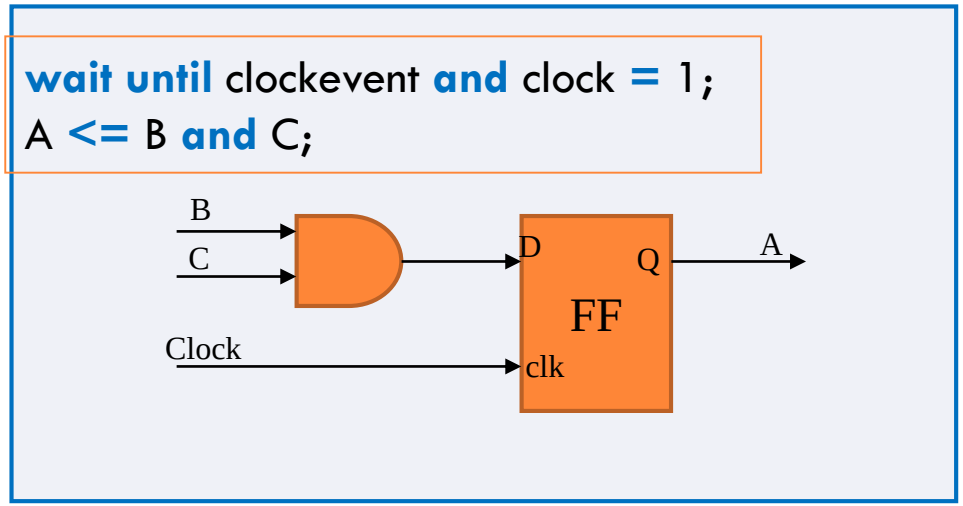
Projeto Lógico com FPGAs: VHDL

- A tradução de comandos VHDL para implementação em FPGA segue as seguintes regras **[2/2]**:
  - **If** – Utiliza multiplexadores.
  - **Comparações** – Utiliza componentes aritméticos e comparadores.
  - **Operações funcionais** – utiliza componentes lógicos e aritméticos, registradores e comparadores, dependendo da operação.

Projeto Lógico com FPGAs: VHDL

- Alguns comandos VHDL e suas traduções em FPGA (lembrando que as portas lógicas são implementadas com LUTs).
- Em casos mais complexos o circuito sintetizado pode variar dependendo de parâmetros e requisitos de otimização (desempenho ou área).



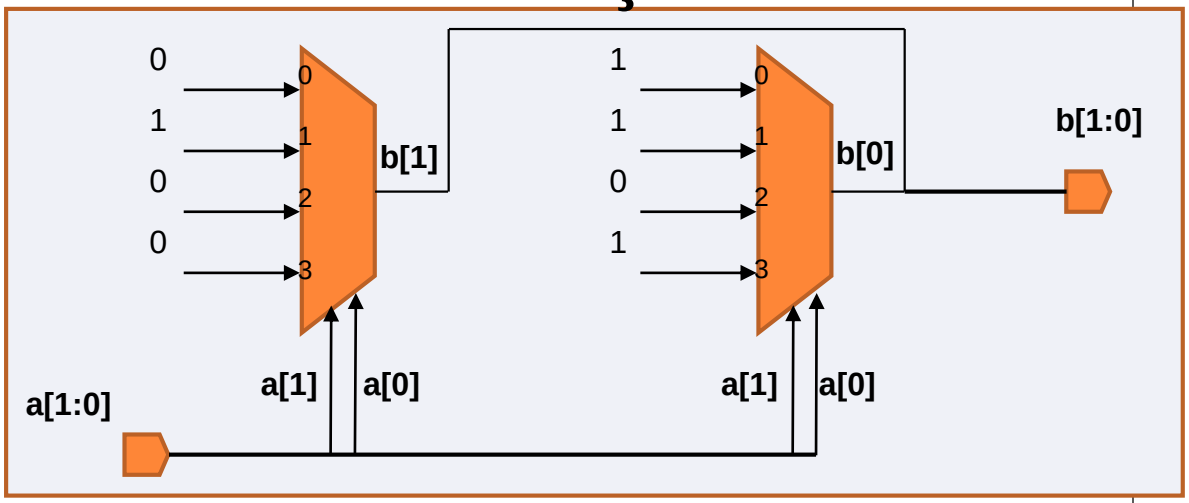


Projeto Lógico com FPGAs: VHDL

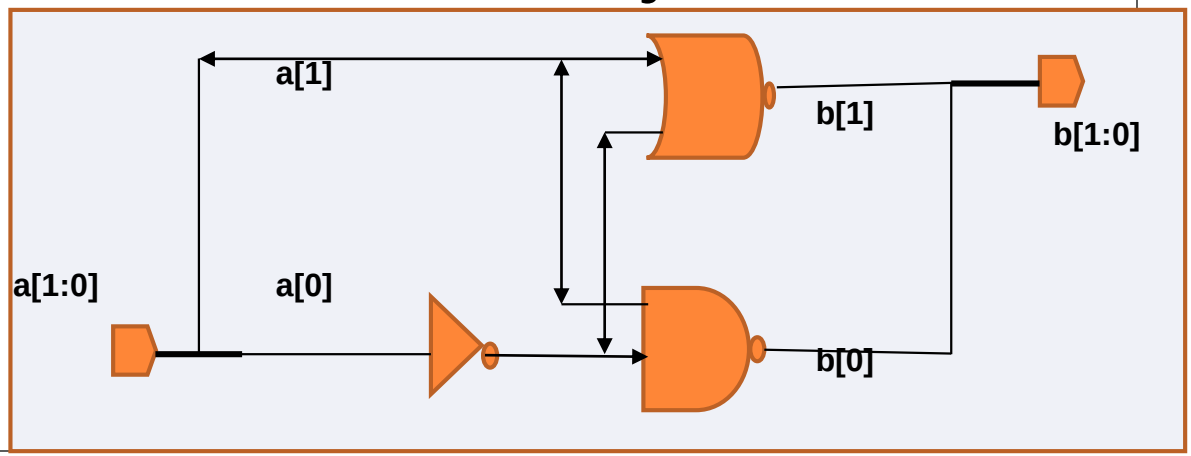
- **Síntese de case:**

```
process (a)
 begin
   case a is
     when 0  =>  b <= 1 ;
     when 1  =>  b <= 3 ;
     when 2  =>  b <= 0 ;
     when 3  =>  b <= 1 ;
   end case;
 end process;
```

Sem otimização:



Com otimização:



Projeto Lógico com FPGAs: VHDL

- Síntese de if – Também sintetizados com MUX.
- É preciso cuidado especial com comandos if onde nem todas as condições são cobertas.
  - Considere:

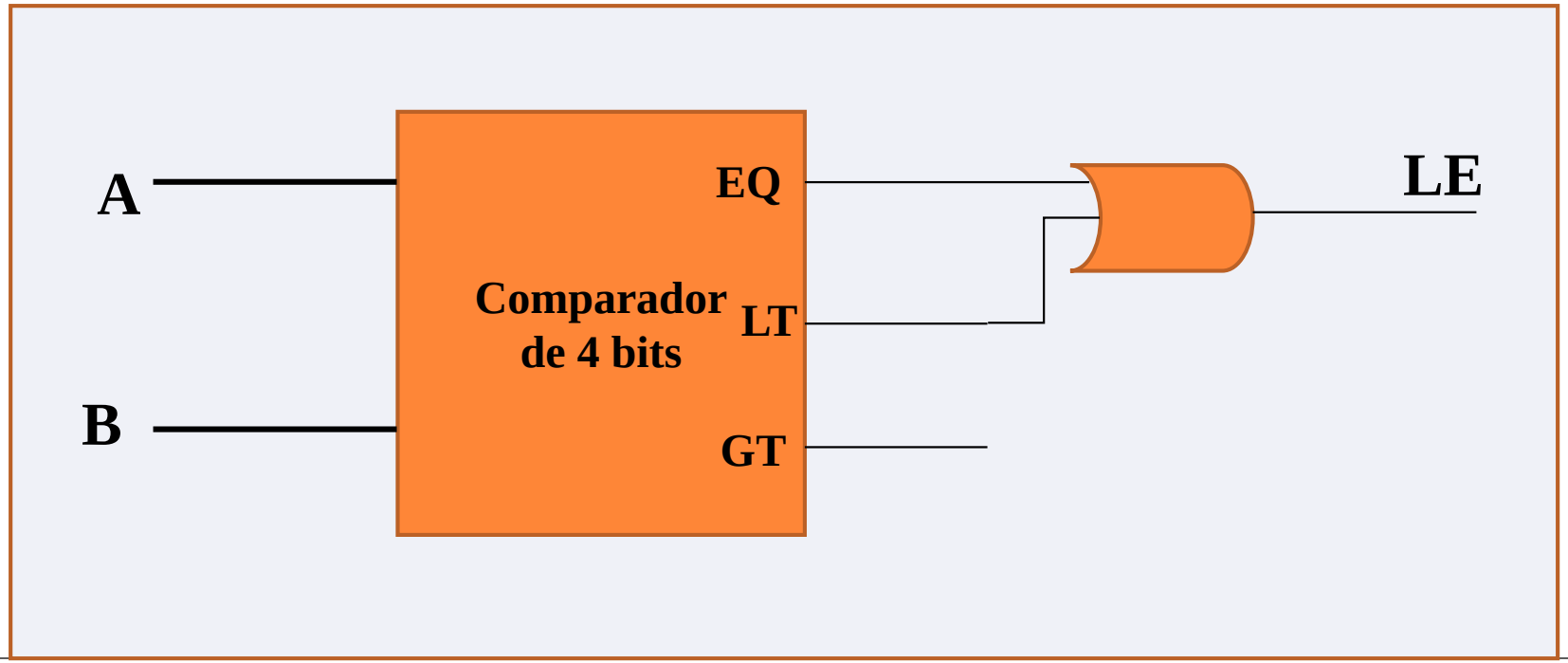
```
if A = 1 then Nextstate <= 3; Z <= 1;
end if;
```

  - A condição else não foi especificada – Possivelmente, deseja-se manter o valor anterior de "Nextstate".
    - Mas o programa de síntese não tendo certeza disso ... pode atribuir o valor X (*unknown*) a este sinal, algo que pode ser indesejado ...

Projeto Lógico com FPGAs: VHDL

- Comparações devem ser usadas com cuidado.
  - Comparações de inteiros usam blocos grandes de circuito, pois não existe forma trivial de fazê-las:

if (A <= B) then



Livro Texto

- Wakerly, J.F.; *Digital Design – Principles & Practices;* Fourth Edition, ISBN: 0-13-186389-4, Pearson & Prentice-Hall, Upper Saddle, River, New Jersey, 07458, 2006.

Lição de Casa

- Leitura Obrigatória:
  - Capítulo 9 do Livro Texto, itens 9.5 e 9.6.
- Exercícios Obrigatórios:
  - Exercícios 9.28 a 9.36 do Capítulo 9 do Livro Texto.

Bibliografia Adicional Deste Assunto

- “Digital Design and Computer Architecture”. Harris, D.M. & Harris, S.L. Morgan Kaufmann, 2007.
- “Digital Systems Design Using VHDL”. Charles H. Roth Jr and Lizy Kurian John. 2nd Edition, 2008. ISBN-10: 0534384625 ISBN-13: 9780534384623
- “The Ten Commandments of Excellent Design—VHDL Code Examples”. Peter Chambers. VLSI Technology.
- Chapter 3 - The Art of VHDL Synthesis. LeonardoSpectrum HDL Synthesis Manual.