PCS 3115
Sistemas Digitais I

Memórias & FPGAs

Prof. Dr. Marcos A. Simplicio Jr.

Adaptado por Glauber (2018)

Memórias

- Dispositivos com grandes capacidade de **armazenamento** de dados
- **Finalidade**: Armazenar dados ou programas de maneira temporária ou definitiva.
- **Aplicações** dependem das características
  - Forma de **acesso**: sequencial ou aleatório?
  - **Velocidade**: quanto mais rápido o acesso, maior a eficiência do sistema como um todo
  - **Funcionalidade**: leitura e escrita ou apenas leitura?
  - **Volatilidade**: dados retidos mesmo se alimentação de energia cortada?

Memórias: Características

- **Tecnologia: Não Semicondutor**
  - **Magnético**
    - » Ferrite, Fita, Discos (Rígido, Flexível)
  - **Ópticos**
    - » CD, DVD, Blu-ray Disc (BD)

- **Tecnologia: Semicondutor**
  - CMOS: alta densidade, baixo consumo

Memórias: Características

- **Acesso a dados**

  - **Sequencial**: Fita magnética, CD/DVD
    - » Posições de memória em lugares sucessivos

    - » Ler ou escrever na posição n requer leitura sequencial de todas as posições anteriores (n-k, ..., n-1)
      - Tempo de acesso depende da posição de leitura atual (k) e da posição dos dados (n)

    - » Baixo custo, porém baixa velocidade
  - **Direto ou "Aleatório"**: Constituem a maior parte das memórias a semicondutores.

    - » Caracterizadas principalmente por possuírem o mesmo tempo de acesso para todas as posições

Memórias: Características

- **Acesso a dados (cont.)**
  - Possibilidade de **escrita** (no circuito):

» Memórias de **apenas leitura**: Gravação de conteúdo muito lenta ou fora do sistema;

» Memórias de **escrita e leitura**: Conteúdo pode ser lido e alterado durante seu uso no circuito.

Memórias: Características

- **Retenção dos dados (volatilidade)**
  - **Voláteis** – Perdem o conteúdo se ficam sem alimentação (ex.: RAM, registradores);

  - **Não Voláteis** – Não perdem o conteúdo mesmo, sem alimentação (ex.: EEPROM, Flash, disco).

Memórias: Observação

- **ROM**:
  - O termo “ROM” significa “*Read-Only-Memory*”, ou memória somente de leitura
  - Porém, ele é comumente usado para indicar **memórias não-voláteis**, incluindo memórias de leitura/escrita…

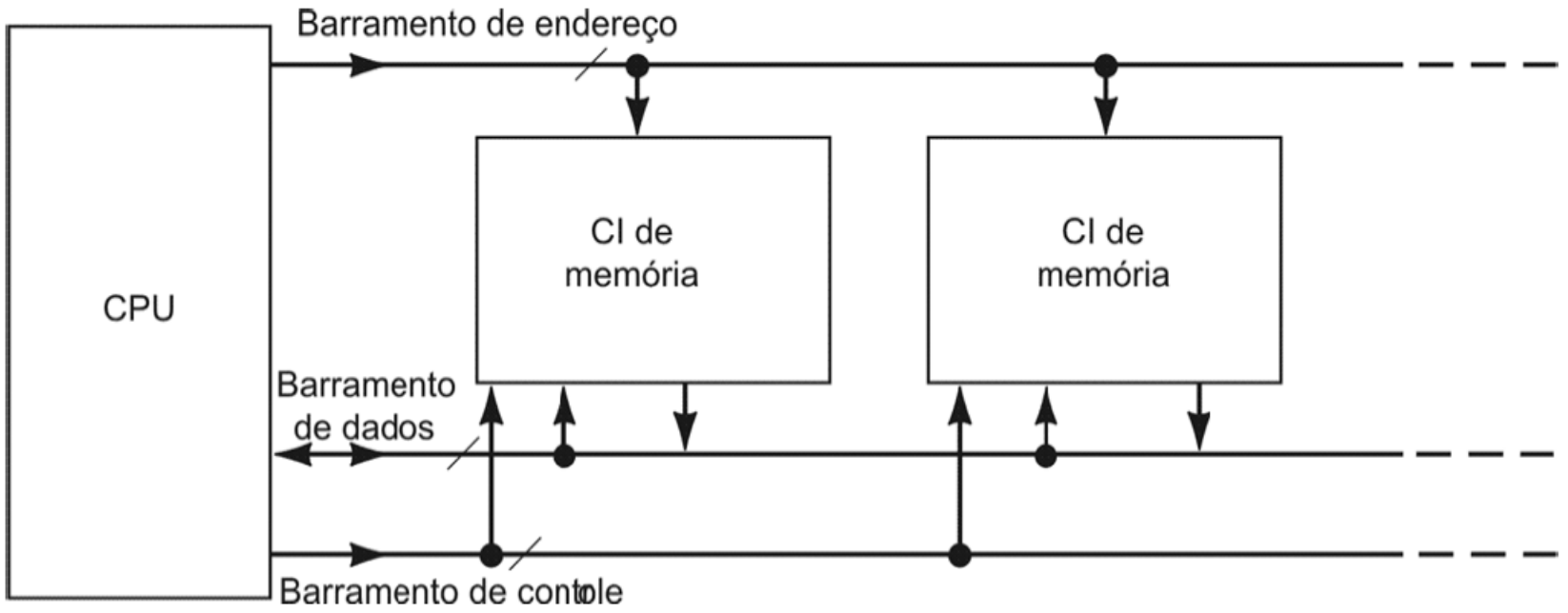
- **RAM**:
  - O termo “RAM” significa “*Random-Access-Memory*”, ou memória de acesso aleatório
  - Porém, ele é comumente usado para **memórias voláteis de acesso aleatório**

Memórias: Características (resumo)

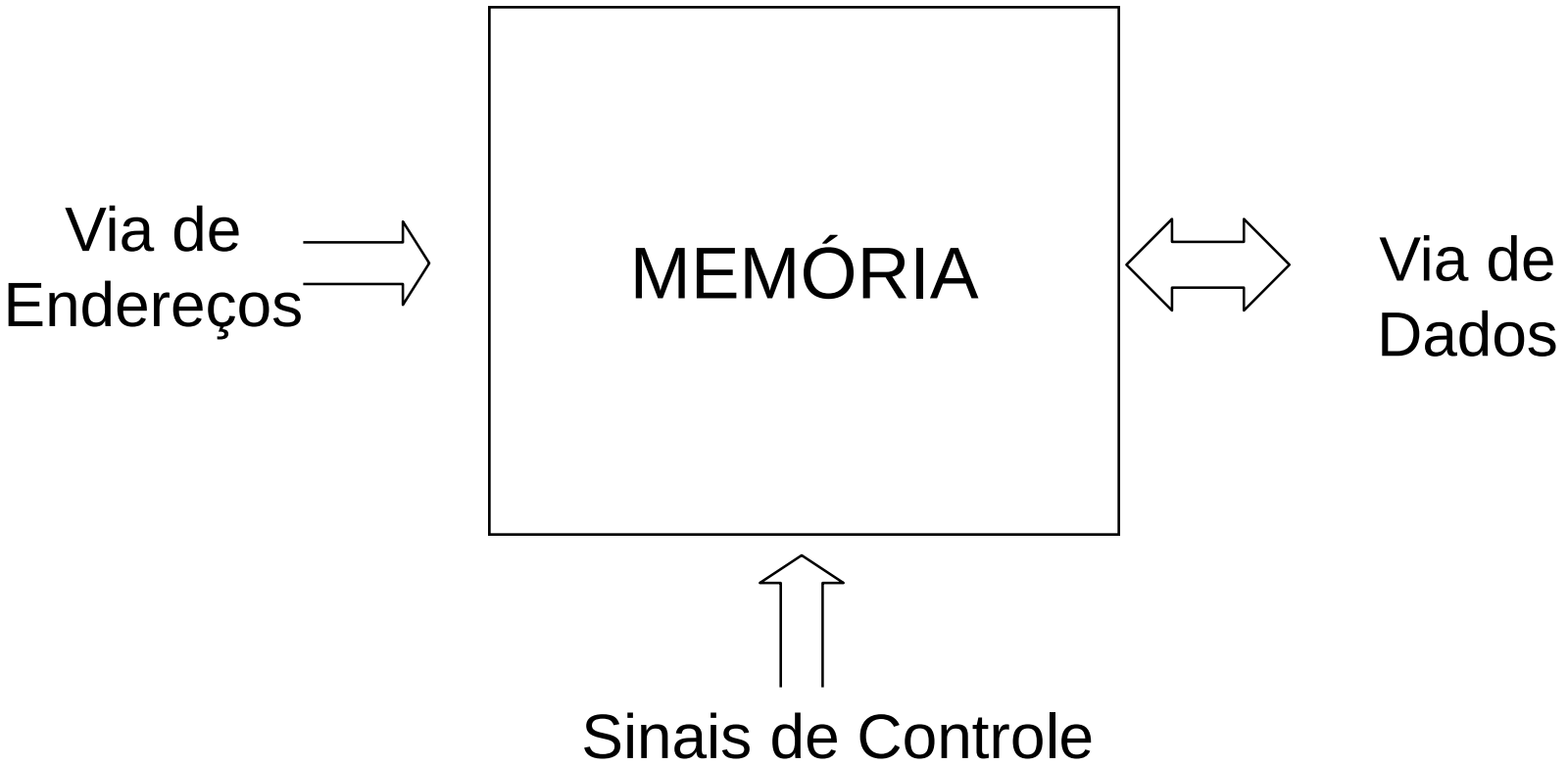
- "Tempo de acesso depende da **posição do dado**?"
  - Sim: acesso **sequencial**; Não: acesso **aleatório**.
- "É possível **ler e escrever**, ou apenas ler?"
  - Leitura e Escrita vs. apenas leitura
- "Os dados são perdidos quando **energia é removida**?"
  - Sim: **volátil**; Não: **não-volátil**

Memórias: Arquitetura

- Interconexão de memórias com CPU



Memórias: Arquitetura



Memórias: Arquitetura

seleciona a posição

Via de Endereços

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	1	0	0
4	1	0	0	0
5	0	0	1	1
6	0	1	1	0
7	1	1	0	0

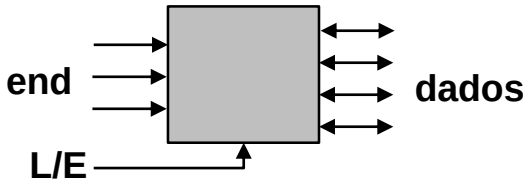
dados a serem lidos/escritos

Via de Dados

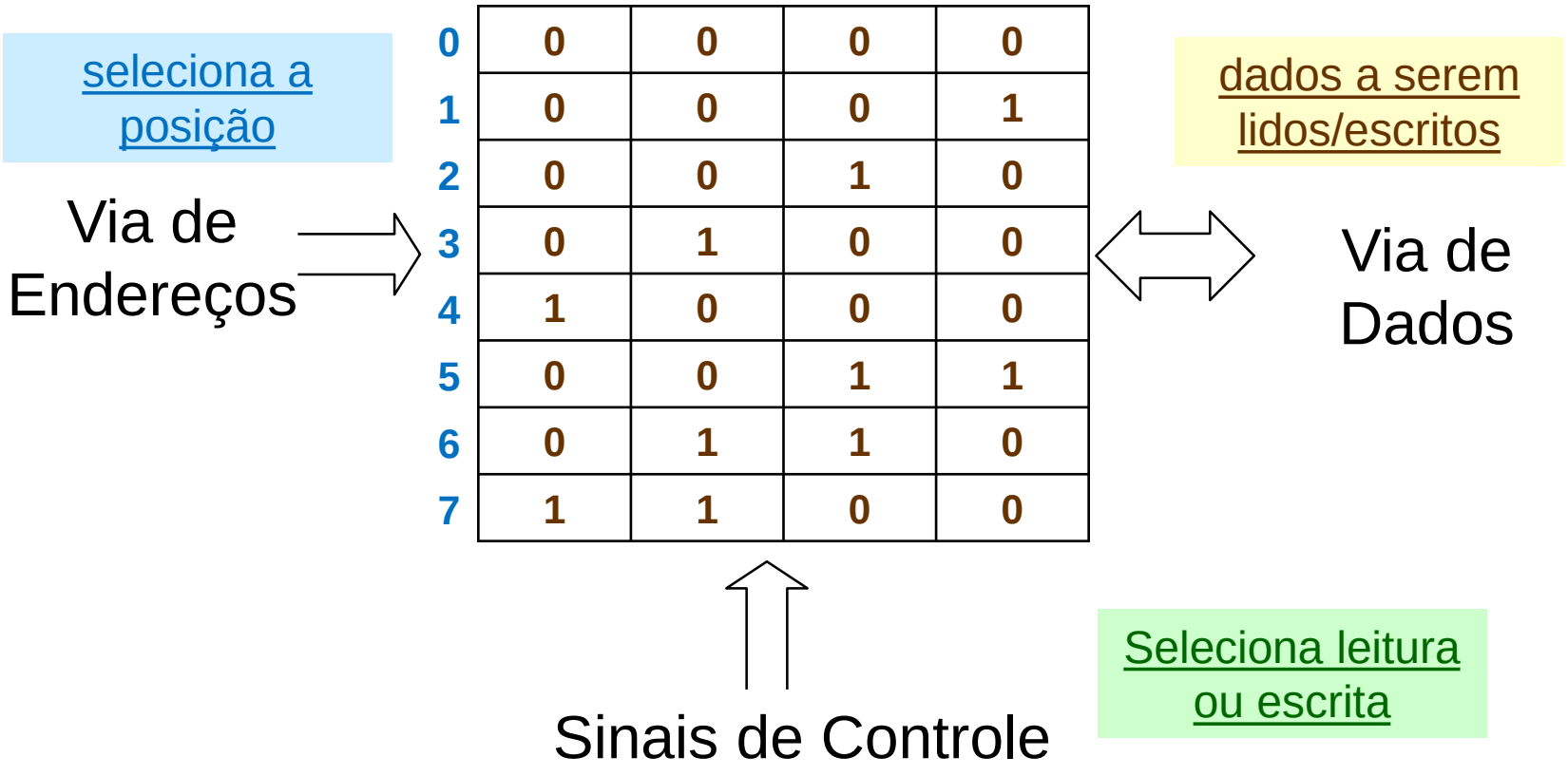
Sinais de Controle

Seleciona leitura ou escrita

➔Memória 8 x 4bits
(32 bits de capacidade)



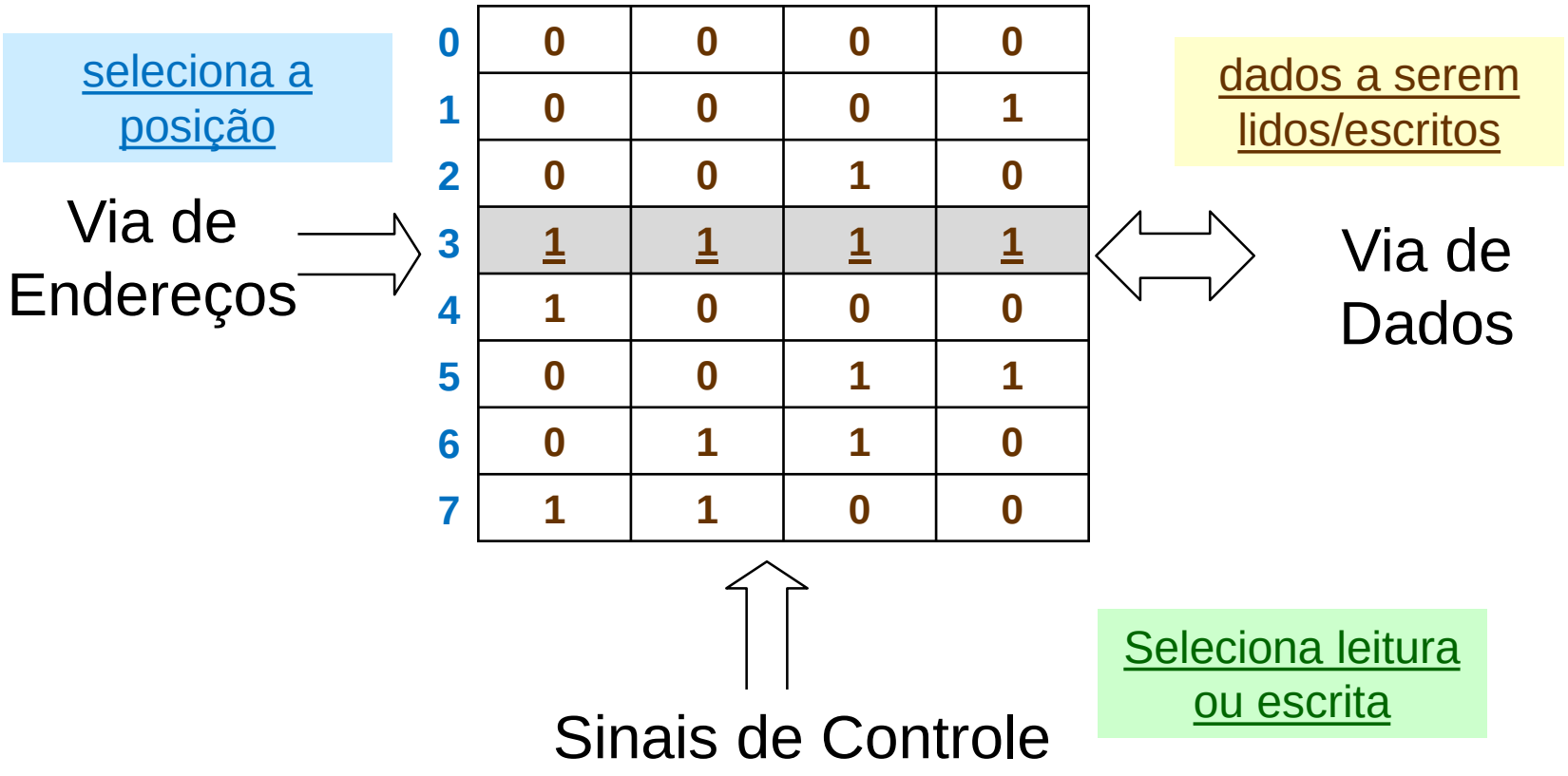
Memórias: Arquitetura



0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	1	0	0
4	1	0	0	0
5	0	0	1	1
6	0	1	1	0
7	1	1	0	0

Ex.: **via de endereços: 011**
via de dados: 1111
comando de escrita

Memórias: Arquitetura

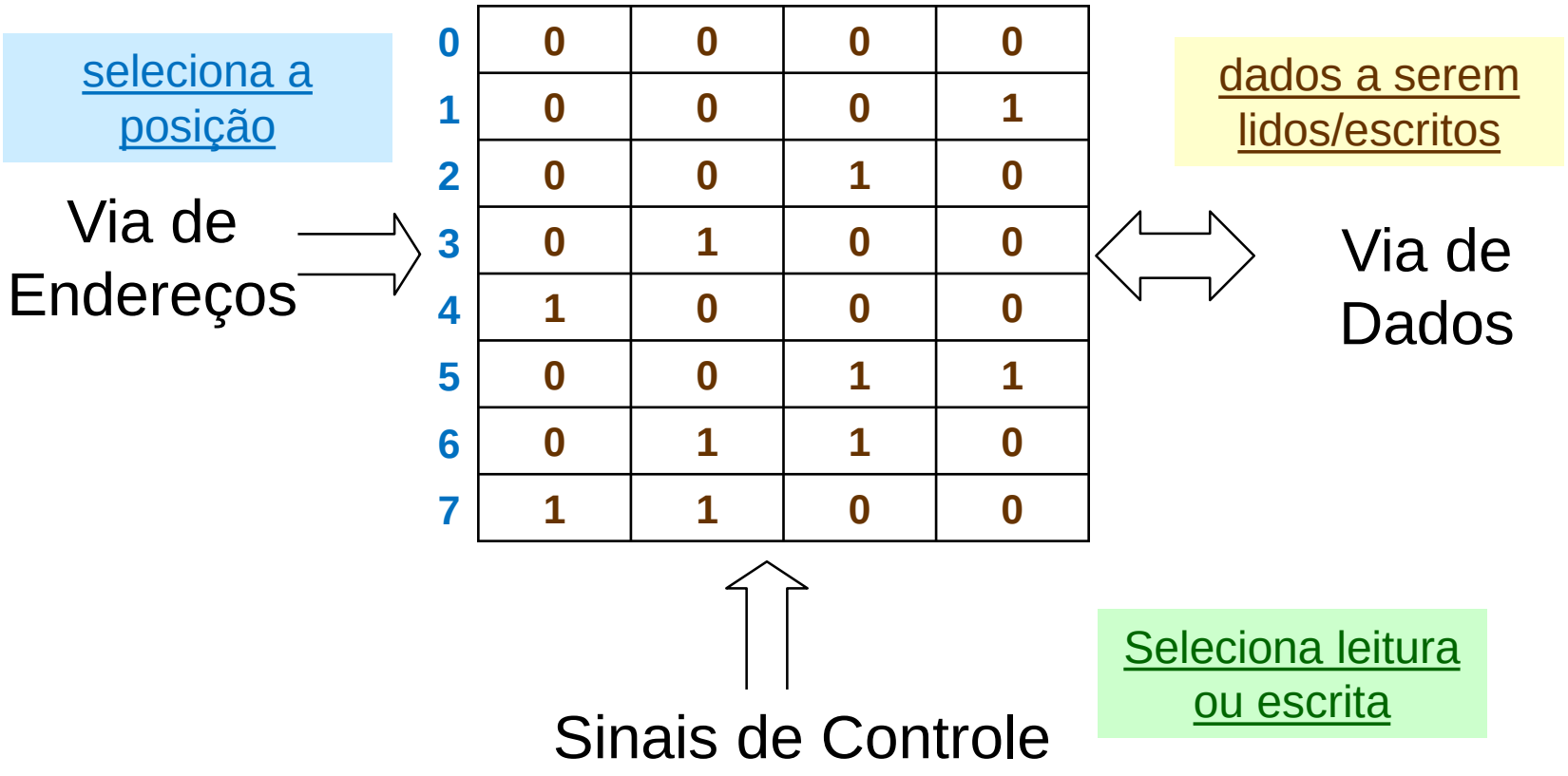


0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	1	1	1	1
4	1	0	0	0
5	0	0	1	1
6	0	1	1	0
7	1	1	0	0

Ex.: **via de endereços: 011**
via de dados: 1111
comando de escrita

➔ **Linha 3 sobrescrita**

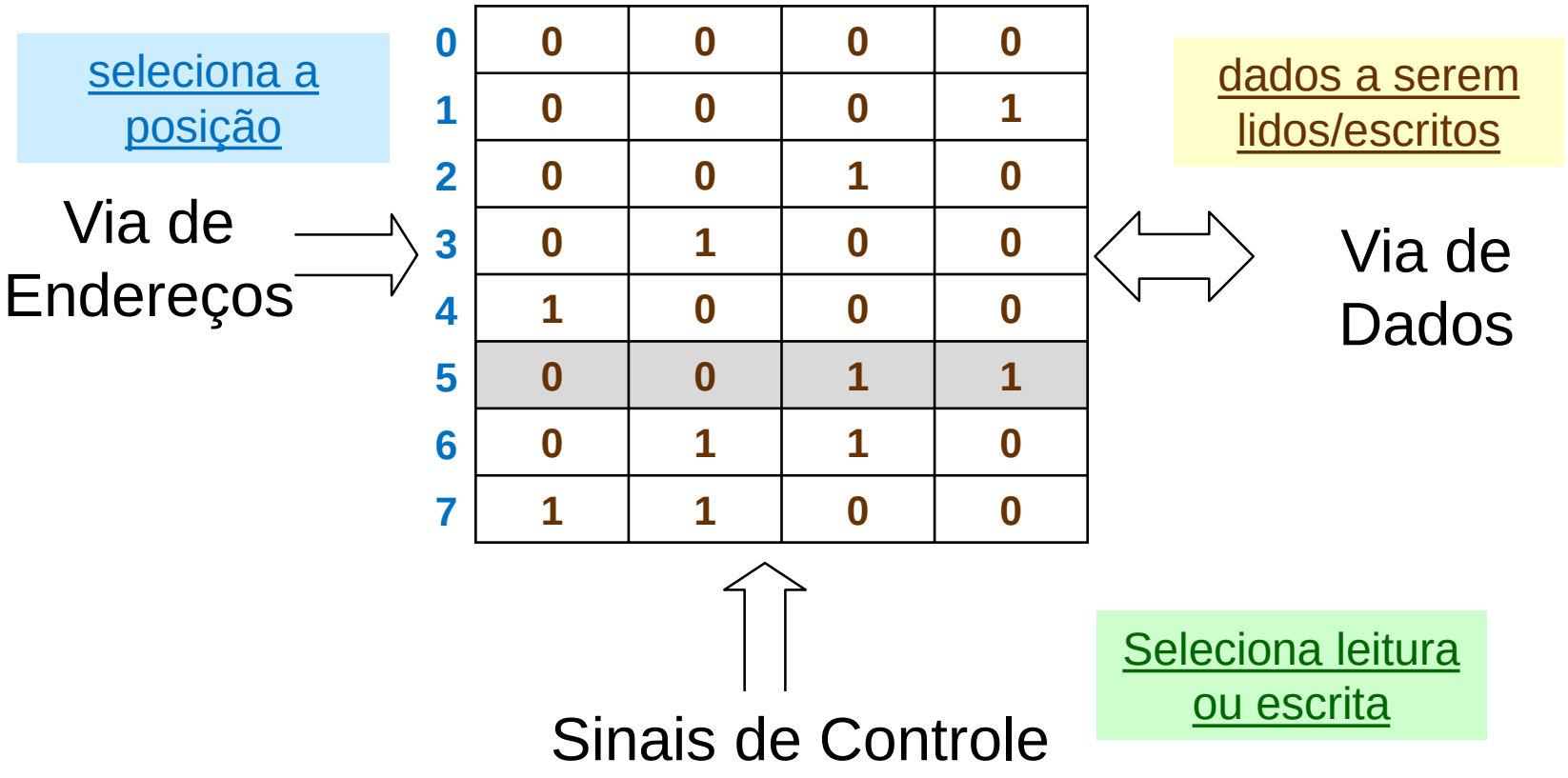
Memórias: Arquitetura



0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	1	0	0
4	1	0	0	0
5	0	0	1	1
6	0	1	1	0
7	1	1	0	0

Ex.: **via de endereços: 101**
comando de leitura

Memórias: Arquitetura



0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	1	0	0
4	1	0	0	0
5	0	0	1	1
6	0	1	1	0
7	1	1	0	0

Ex.: **via de endereços: 101**
comando de leitura

➔ **Via de dados: 0011**

Memórias: Capacidade

- **Palavra:**
  - » Nº de bits tratados simultaneamente (escrita/leitura): Típico: 1, 4, 8, 16, 32, 64, 128 ...
- **Nº de Posições:**
  - » Nº de palavras que a memória pode armazenar
  - » Cada posição está associada a um ENDEREÇO.
- **Capacidade:**
  - » Quantidade de bits de uma memória, expressa em:
    - – Nº total de bits (ou bytes) ou
    - – Nº de posições x Tamanho de palavra em bits ou bytes
    - – **Obs.: 1 byte = 8 bits**

Memórias: Capacidade

– **Relação entre nº de posições e nº de linhas na Via de Endereços**

» **Cada posição => um endereço (acesso direto)**

$$
\begin{array}{ll}
2 \text{ posições} & \Rightarrow 1 \text{ linha de endereçamento} \\
4 & \Rightarrow 2 \\
\vdots & \vdots \\
2^n & \Rightarrow n
\end{array}
$$

Ex: 1024 posições **10 linhas na via de endereço: A0-A9**

Ex: 2^{16} = 65536 posições **16 linhas : A0-A15**

Memórias: Capacidade

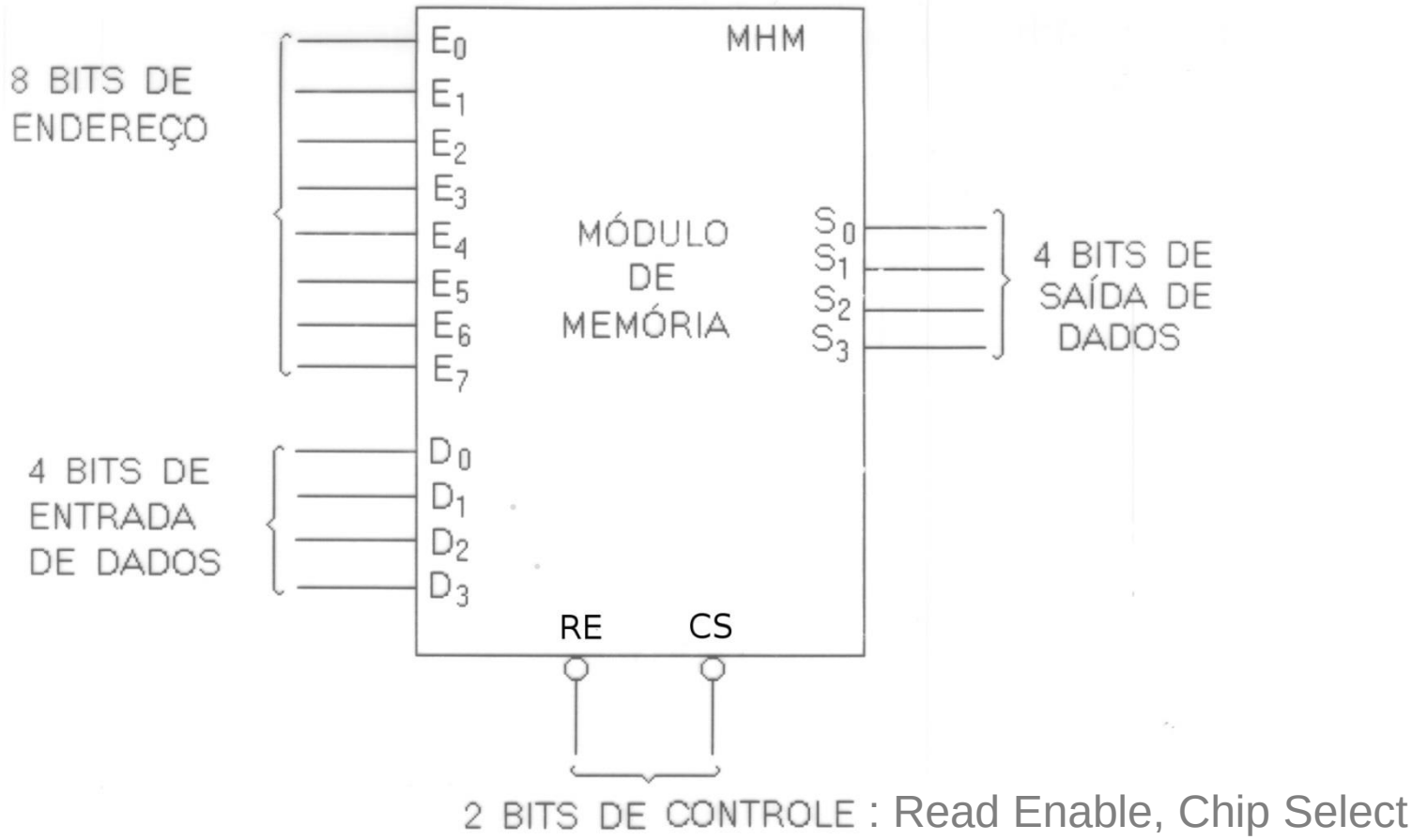
- **Ex. Capacidade:**
  - » **Ex.: Memória de** { 1024 posições; palavra de 8 bits

Capacidade = 8192 bits ou 1024 x 8
Convenção: 1024 = 1ki (ISO 8000)
1 bit = b ; 1 byte = B
➔ 8 kib, 1ki x 8 bits, 1ki bytes, 1 kiB
Obs.: referências antigas adotam 1k = 1024

recomendável

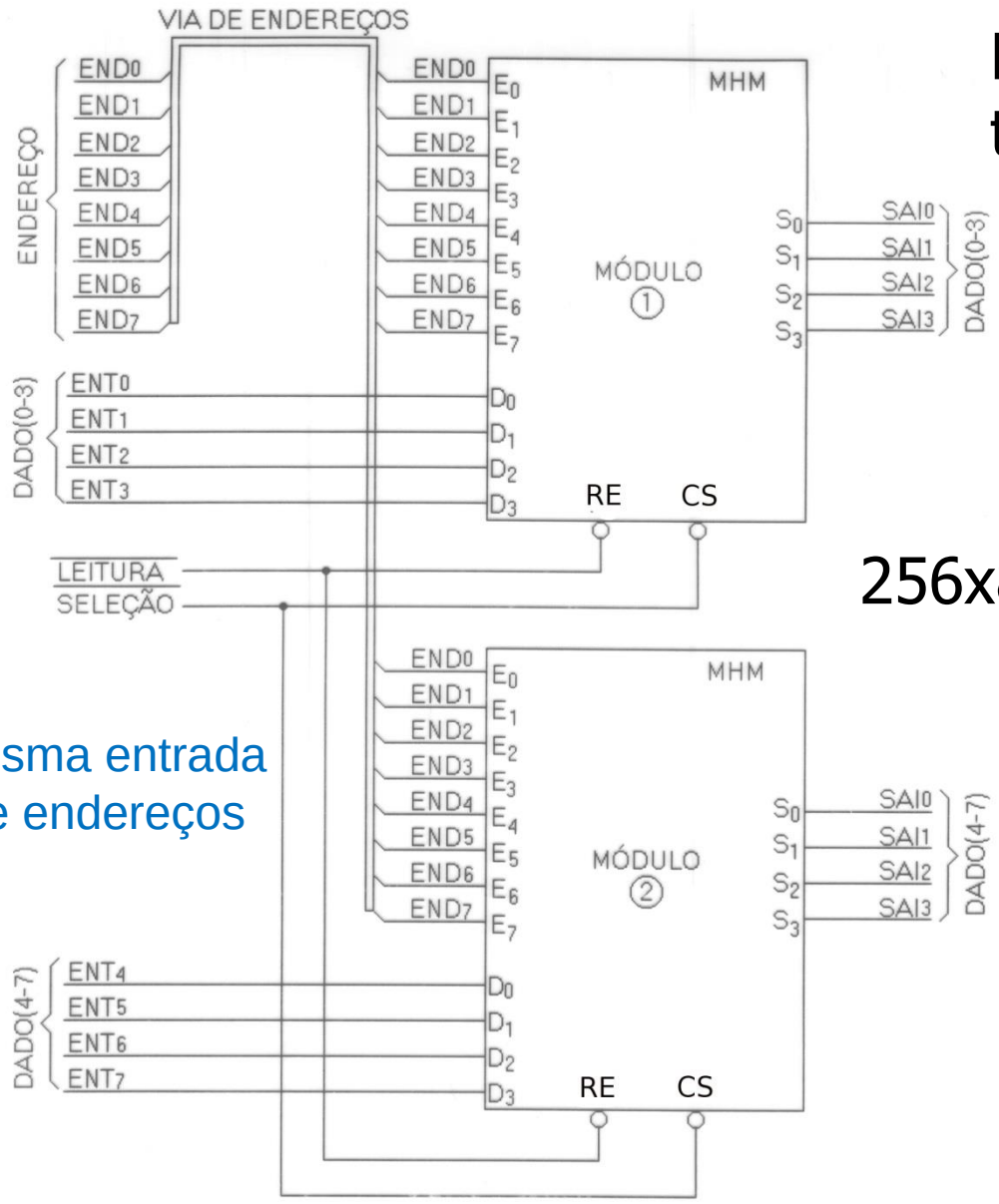
  - » **Ex.: Memória de 64 kiB =** { 64 ki posições; palavra de 1 byte

Associação de Memórias



- **Pergunta**: usando o mesmo tipo de chip, expandir **tamanho das palavras** para 8 bits

Associação de Memórias



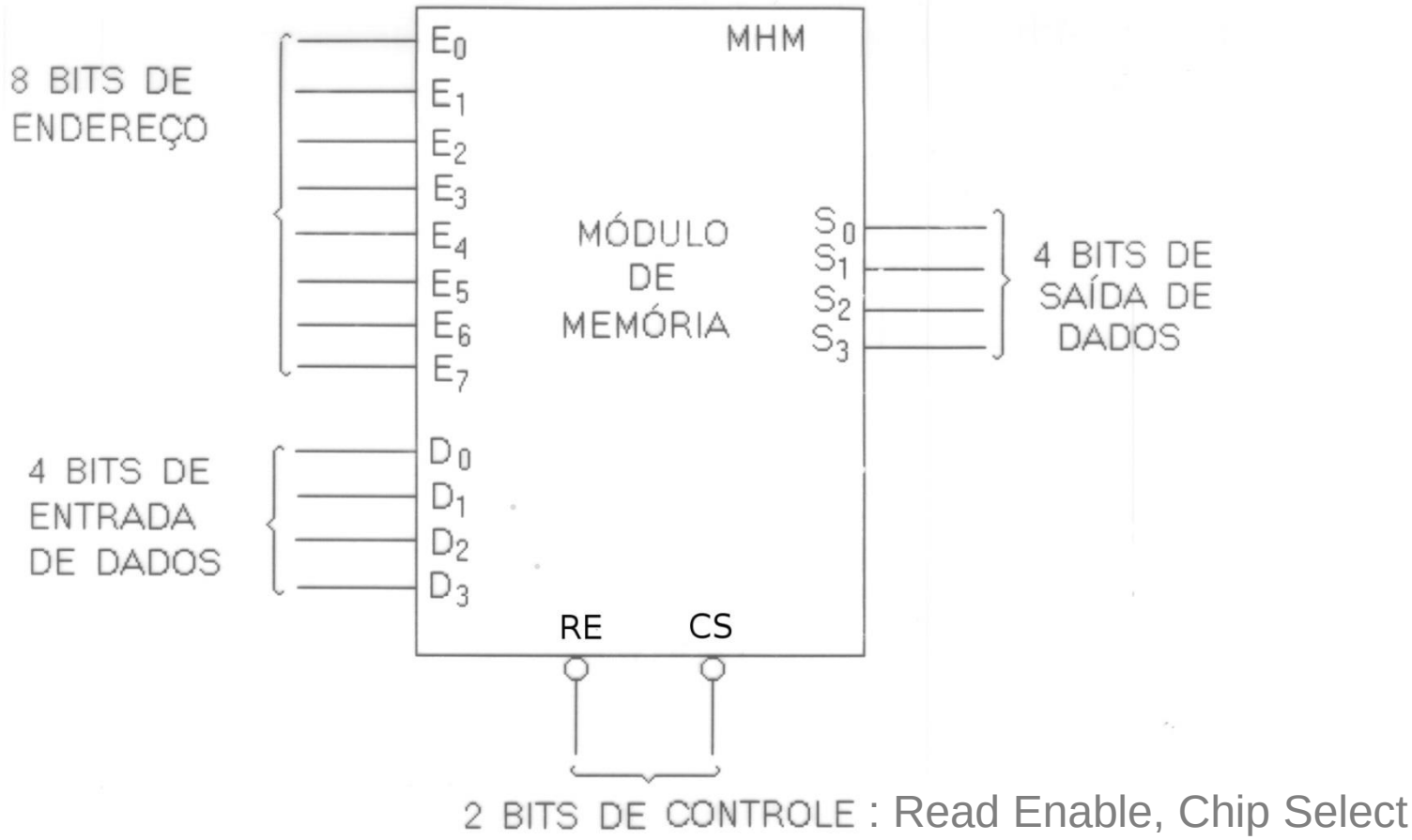
Ex.: expansão de tamanho da palavra.

256x8 bits

Mesma entrada de endereços

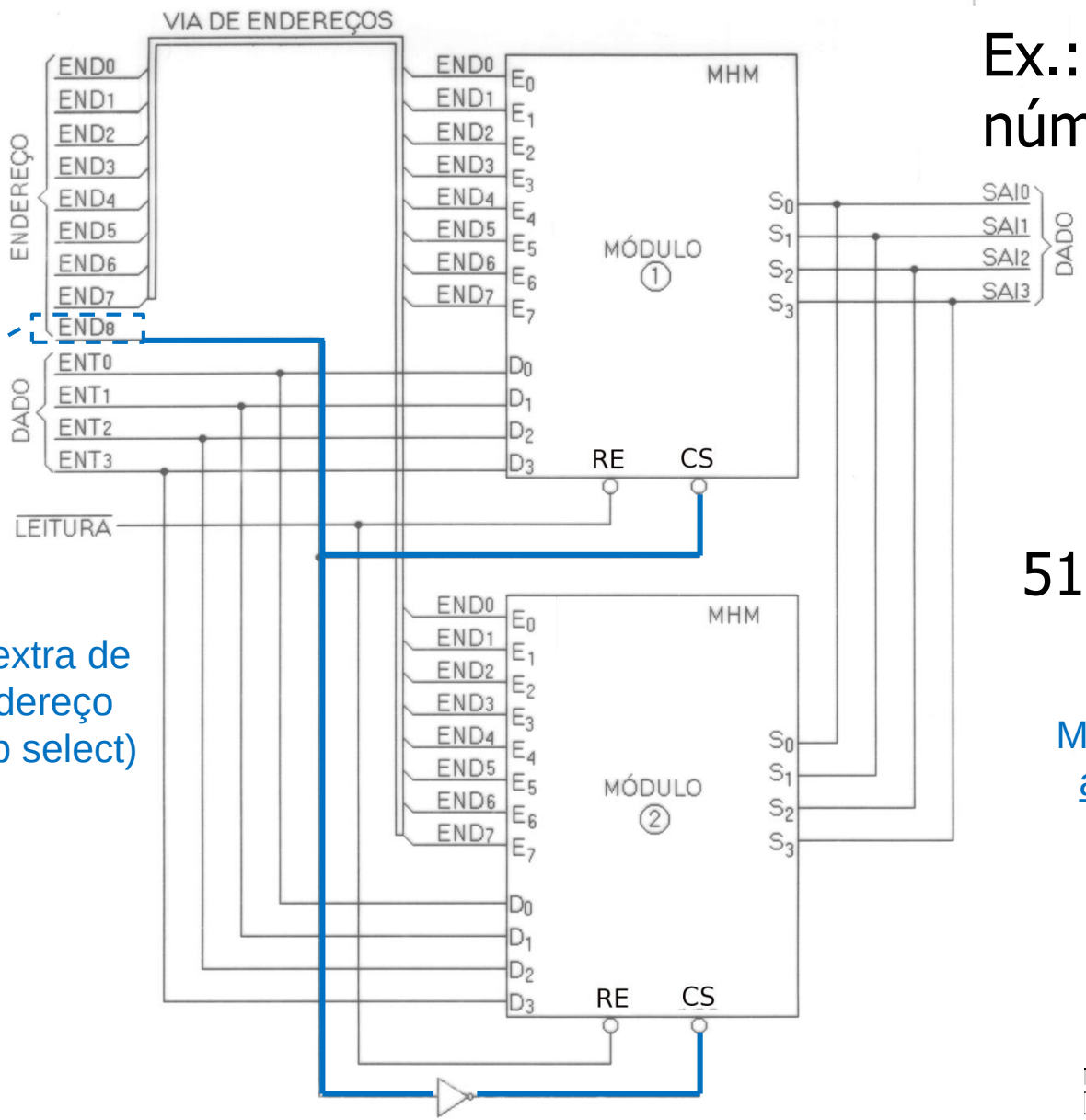
Combinação dos bits de dados

Associação de Memórias



- **Pergunta**: usando o mesmo tipo de chip, expandir **<u>número de palavras</u>** para 512

Associação de Memórias



Ex.: expansão do número de palavras

512x4 bits

Bit extra de endereço (chip select)

Mesma via de dados: alta impedância se CS desativado

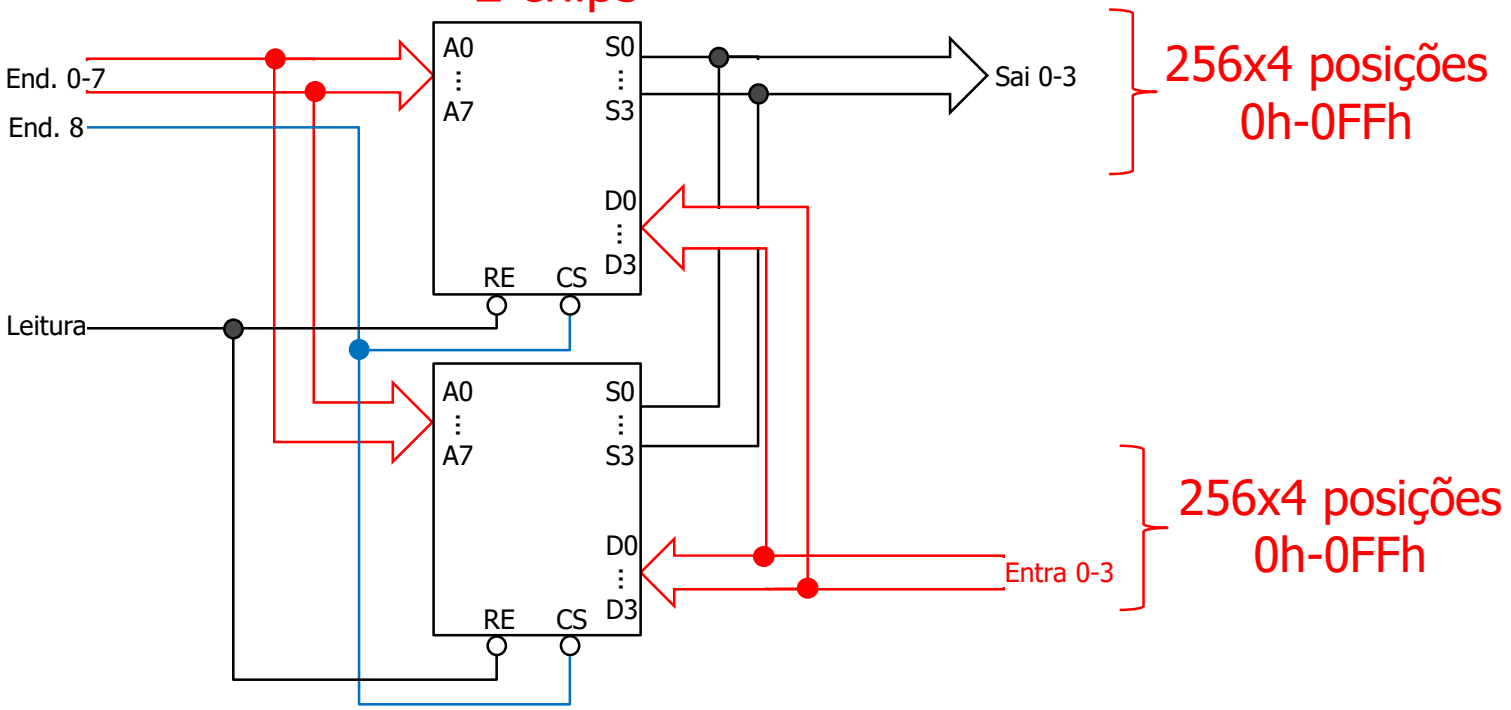
Associação de Memórias

Aumento do número de posições

Ex.: A partir do Módulo de 256x4 => obter 512x4

512 posições => 2^9 => Via de endereços 9 bits

=> 2 chips



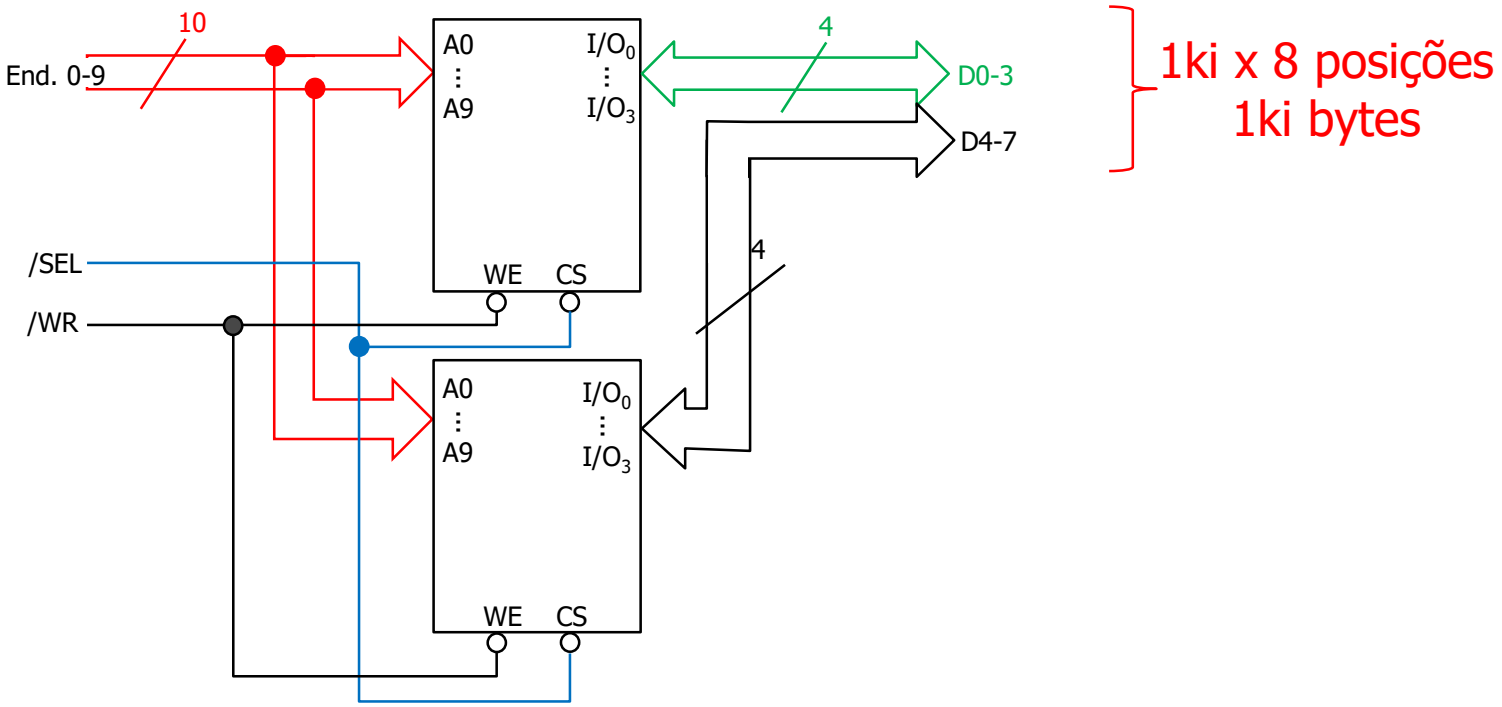
-Acesso não simultâneo a cada CI
-Necessidade de saídas tri-state => barramento de dados (data bus)
-Controle da seleção dos CIs: bits de endereço que excedem a capacidade de endereçamento interno.
-Endereçamento dentro dos CI: bits menos significativos.

Associação de Memórias

Aumento do tamanho da palavra

Ex.: Dada a RAM 1ki x 4bits, obter 1ki x 1 byte.

Tamanho: 1 byte = 8 bits => 2 chips de 4 bits

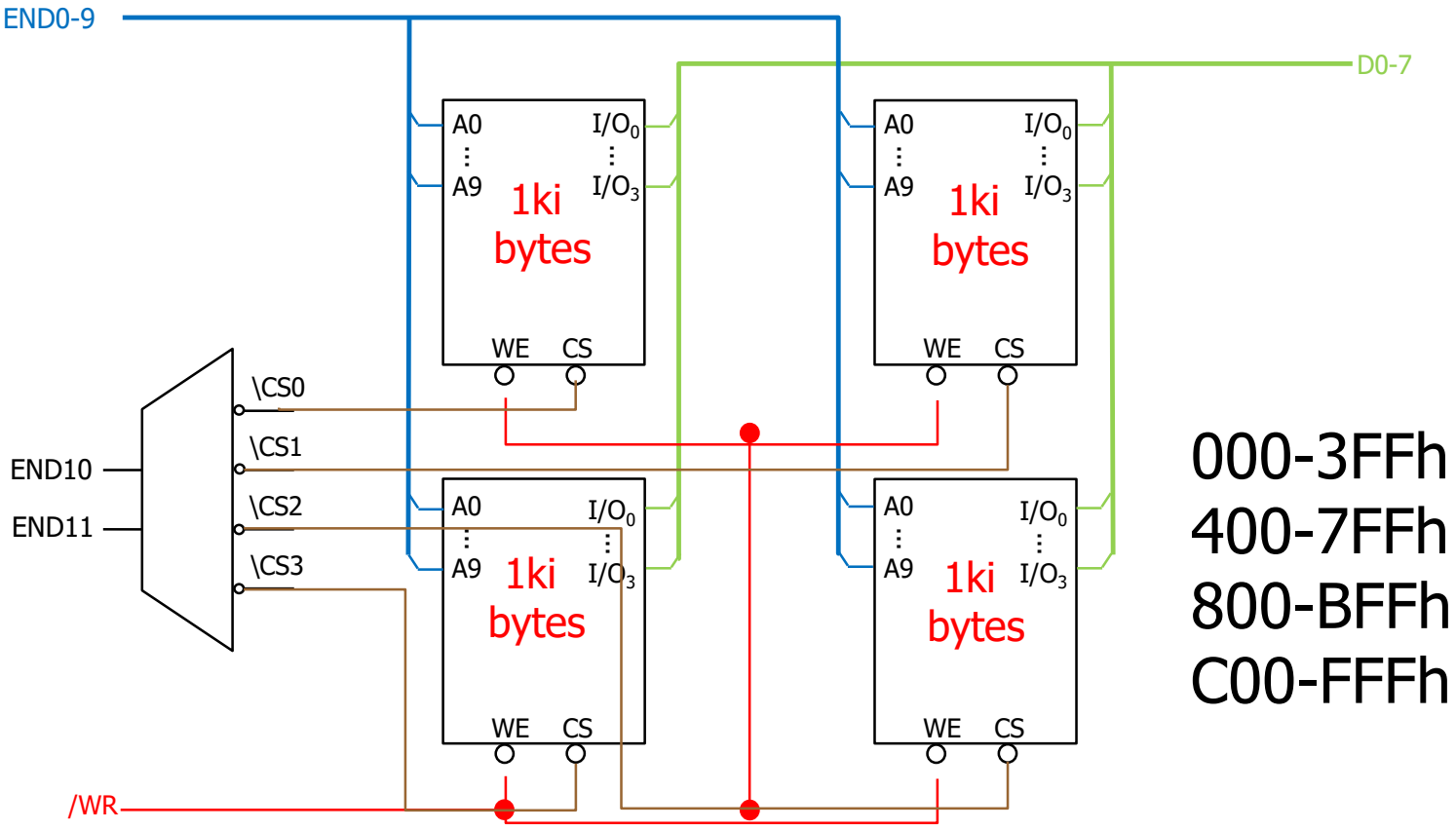


Associação de Memórias

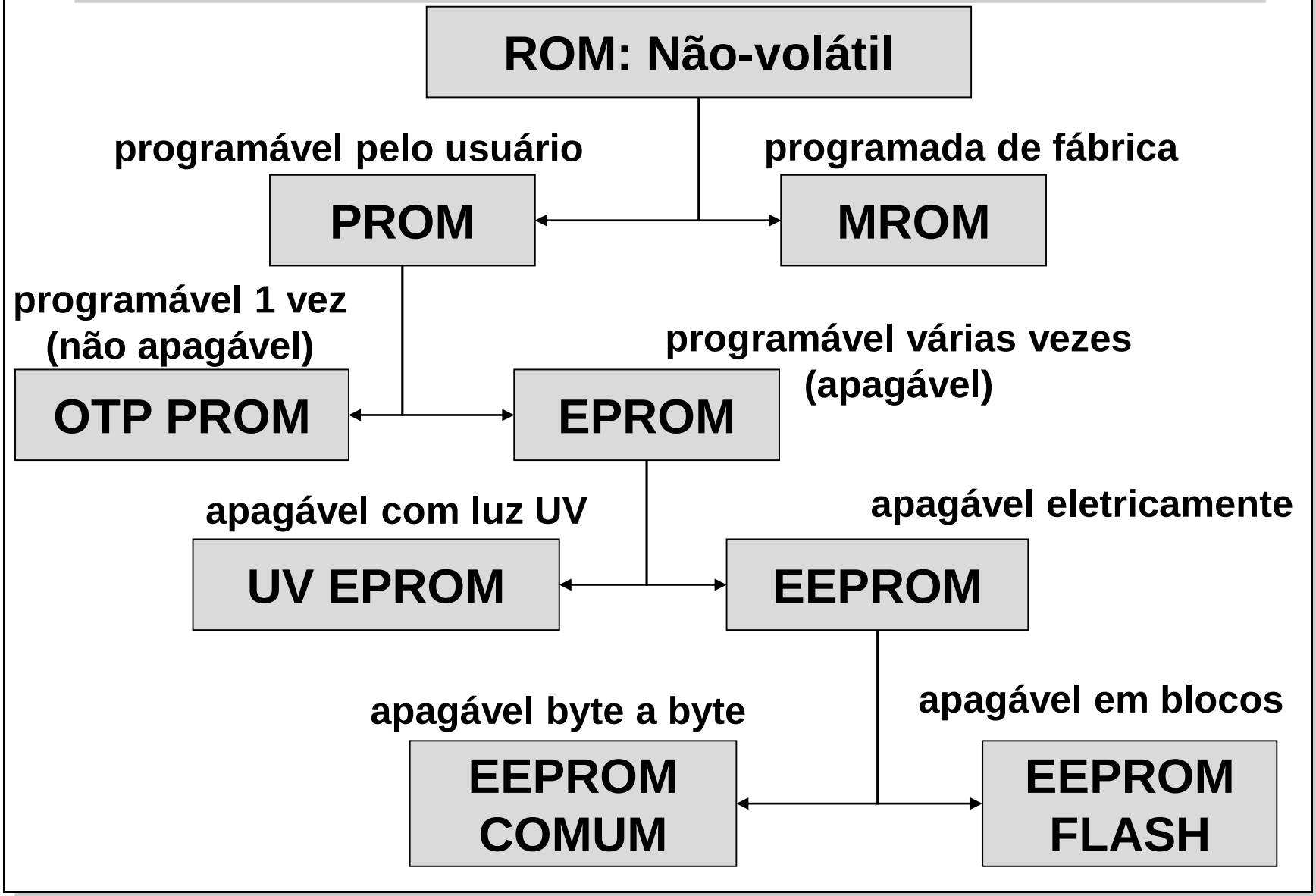
Aumento do número de posições

Ex.: Dada a RAM 1ki x 8bits, obter 4ki x 1 byte.

Endereços: 4ki posições => 2^{12} => Via de endereços 12 bits => 4 chips



Memórias ROM: Tipos

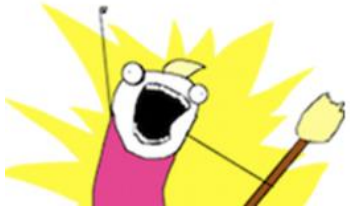


Memórias RAM

- **RAM:** Memórias de leitura e escrita, voláteis, mais rápidas que ROM, porém mais caras
- Principal tecnologia: **CMOS**
- Dois principais tipos: estática (**SRAM**) e dinâmica (**DRAM**)
- **DDR** DRAM: double-data-rate, lê escreve no dobro da velocidade.

Memórias: Hierarquia

– O que queremos?



» Memórias rápidas **E** baratas!

– O que temos?

» Memórias rápidas **OU** baratas.

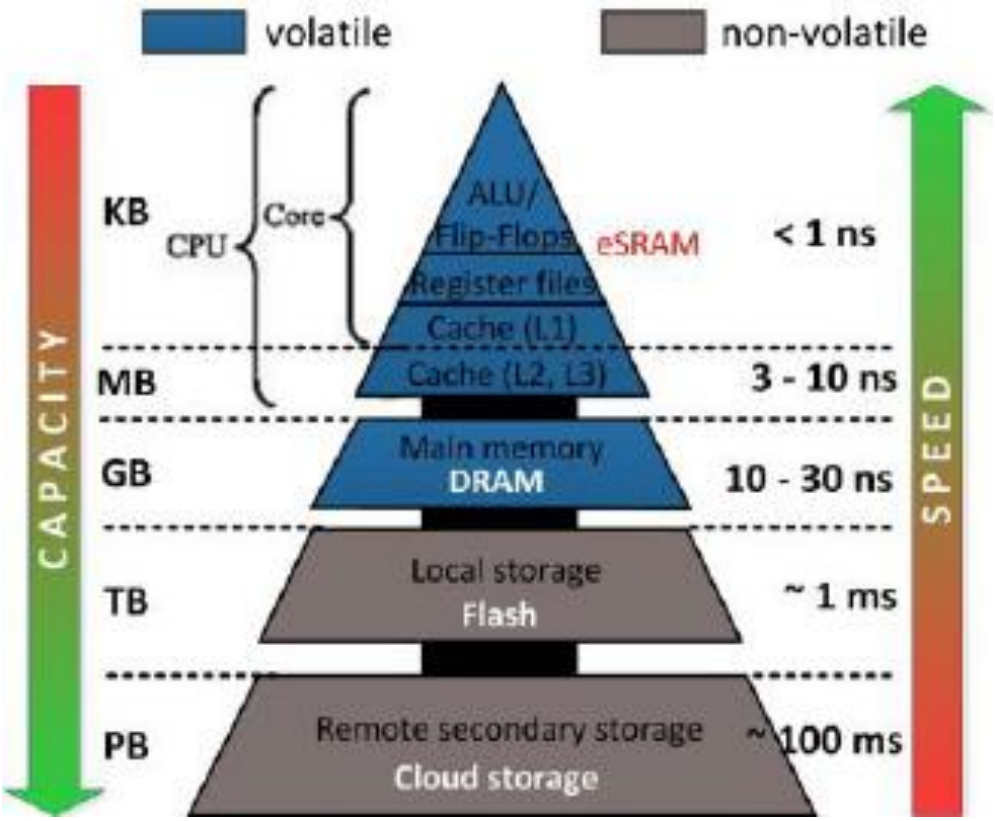
– SRAM: 0.01 centavo por bit, acesso em 1 ns

– HD: 10^{-9} centavos/bit, acesso em 10 ms

– **Solução**: Hierarquia de memórias.

Memórias: Hierarquia

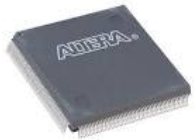
Memórias mais rápidas para o que mais usamos



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



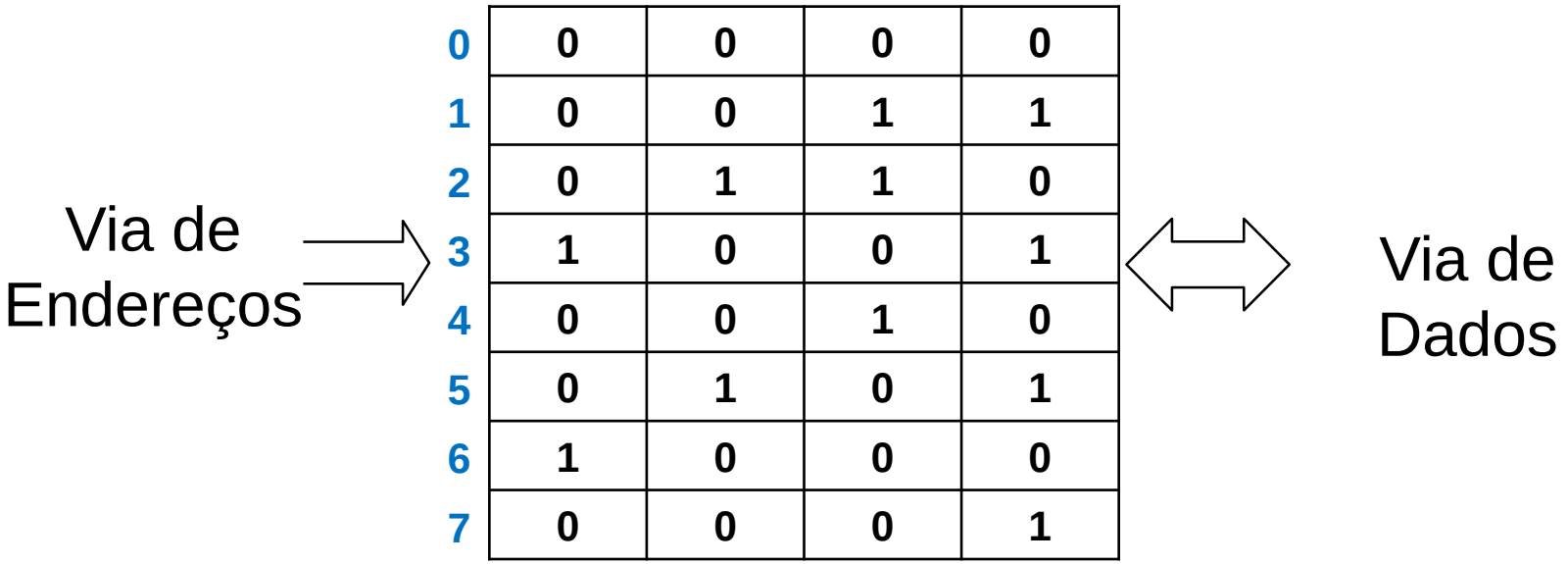
    - Ex.: Memórias, FPGA, CPLD

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

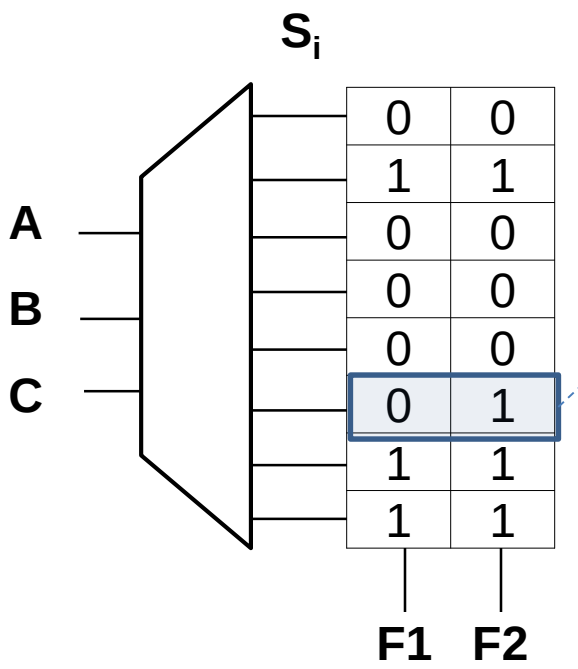
$$F1 = \overline{A}BC + A\overline{B}C + ABC$$

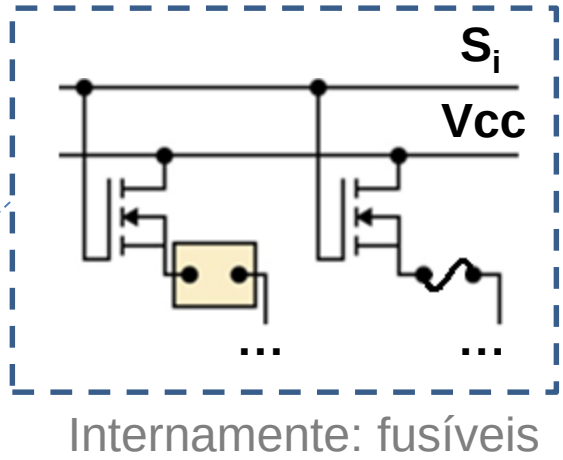
$$F2 = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

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





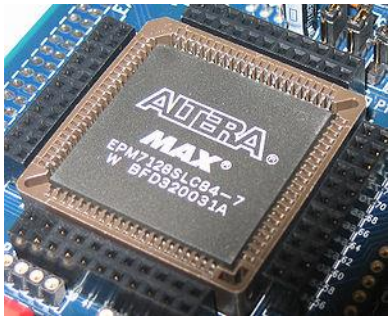
Internamente: fusíveis

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

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

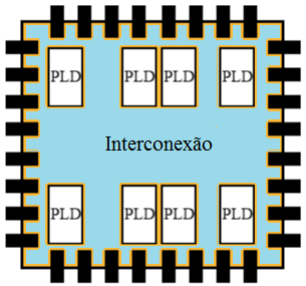
PLDs modernos: CPLDs e FPGAs

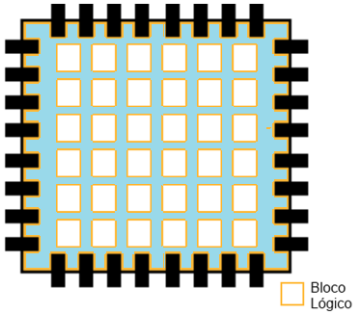
- Com o avanço da microeletrônica, circuitos programáveis mais complexos foram sendo elaborados.
- Circuitos atuais se enquadram em duas categorias:
  - CPLD – “Complex PLD”
  - **FPGA** – “Field Programmable Gate Array”



PLDs modernos: Características

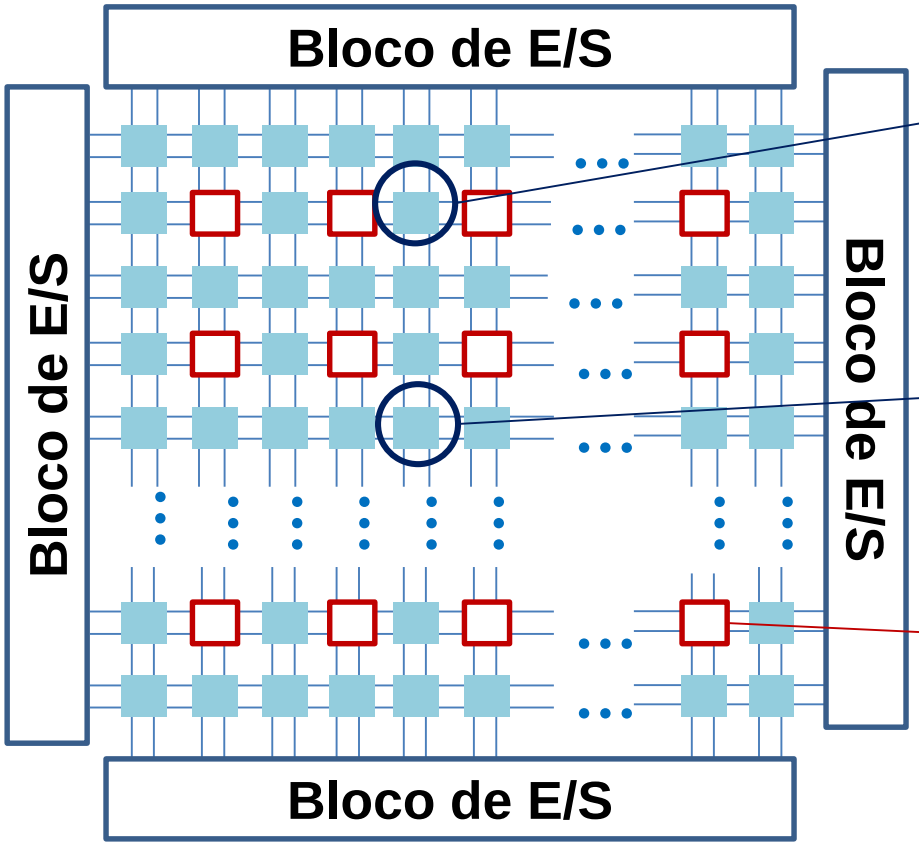
- Grande quantidade de lógica programável;
- Registradores pré-implementados para implementar circuitos sequenciais
- Interconexões programáveis entre lógica programável, registradores e entradas e saídas do dispositivo.
  - É possível controlar o roteamento dos sinais dentro do circuito





PLDs modernos: FPGAs

- 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: FPGAs

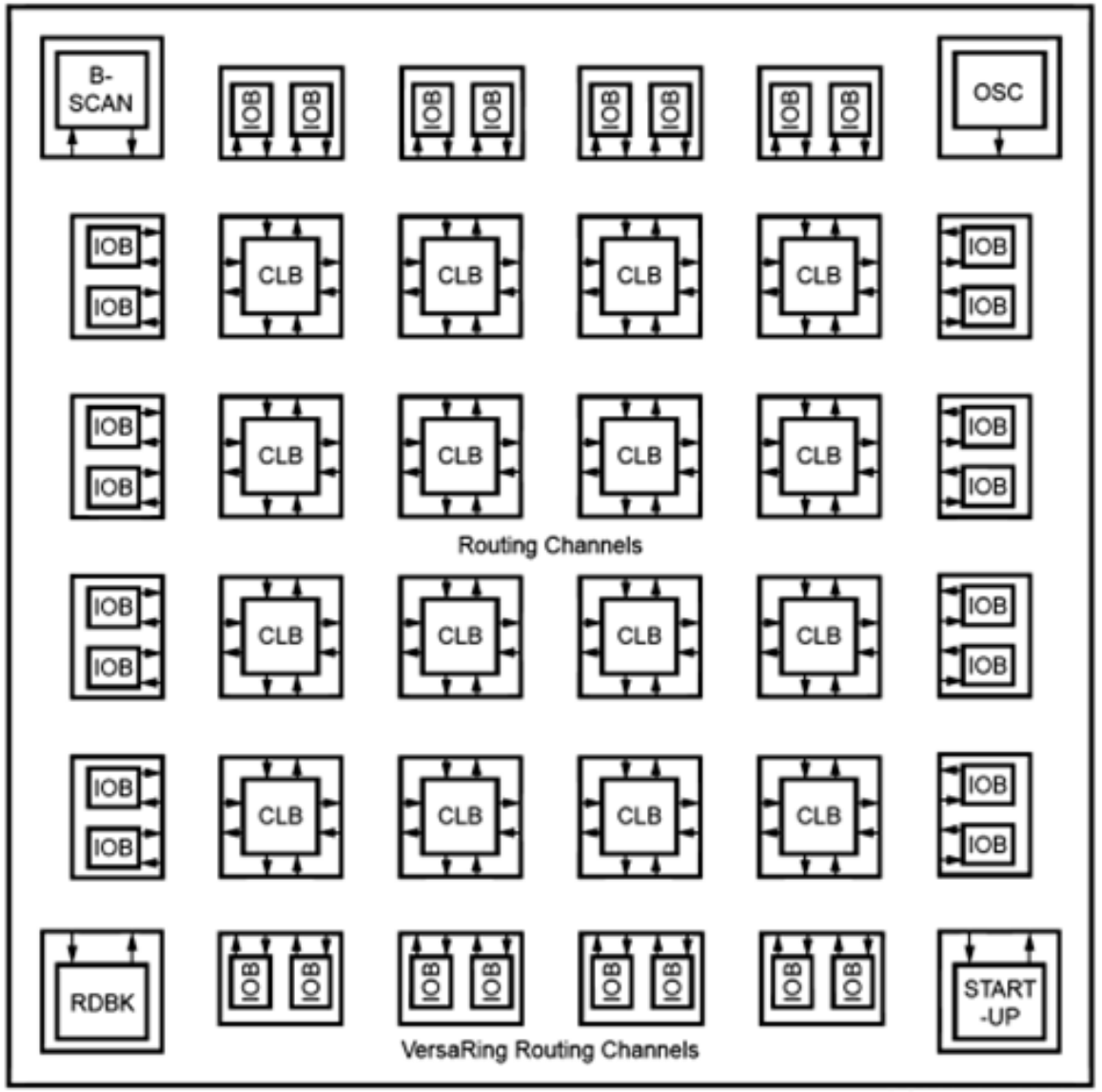
- **Complexidade**:
  - Podem emular algumas centenas de milhares de portas lógicas.
- **Volatilidade**
  - FPGAs: voláteis (perdem a informação se desligadas).
    - Várias FPGAs são associadas a *PROMs para permitir que informação seja recuperada ao ligar.
- **Programação**: linguagens de descrição de hardware
  - VHDL, Verilog: traduzidas por sintetizador para o conjunto de bits que efetivamente programa o circuito

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

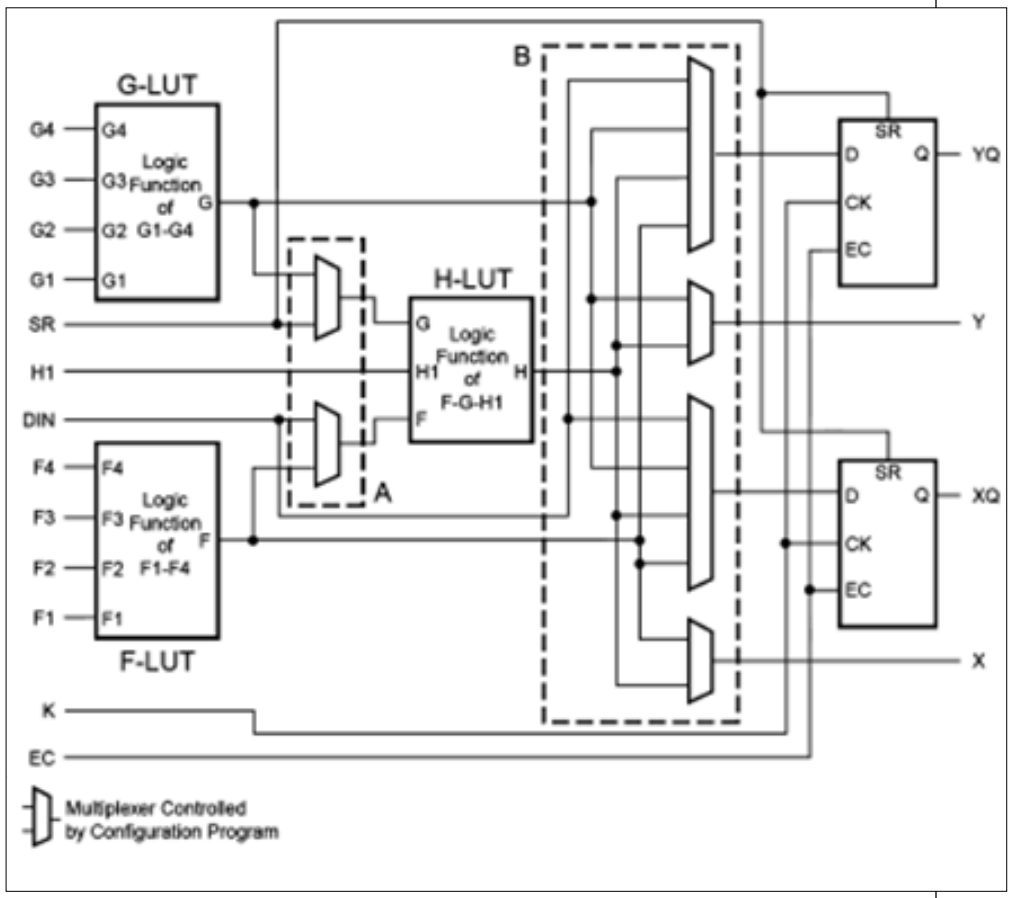


FPGAs: CLBs

- 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



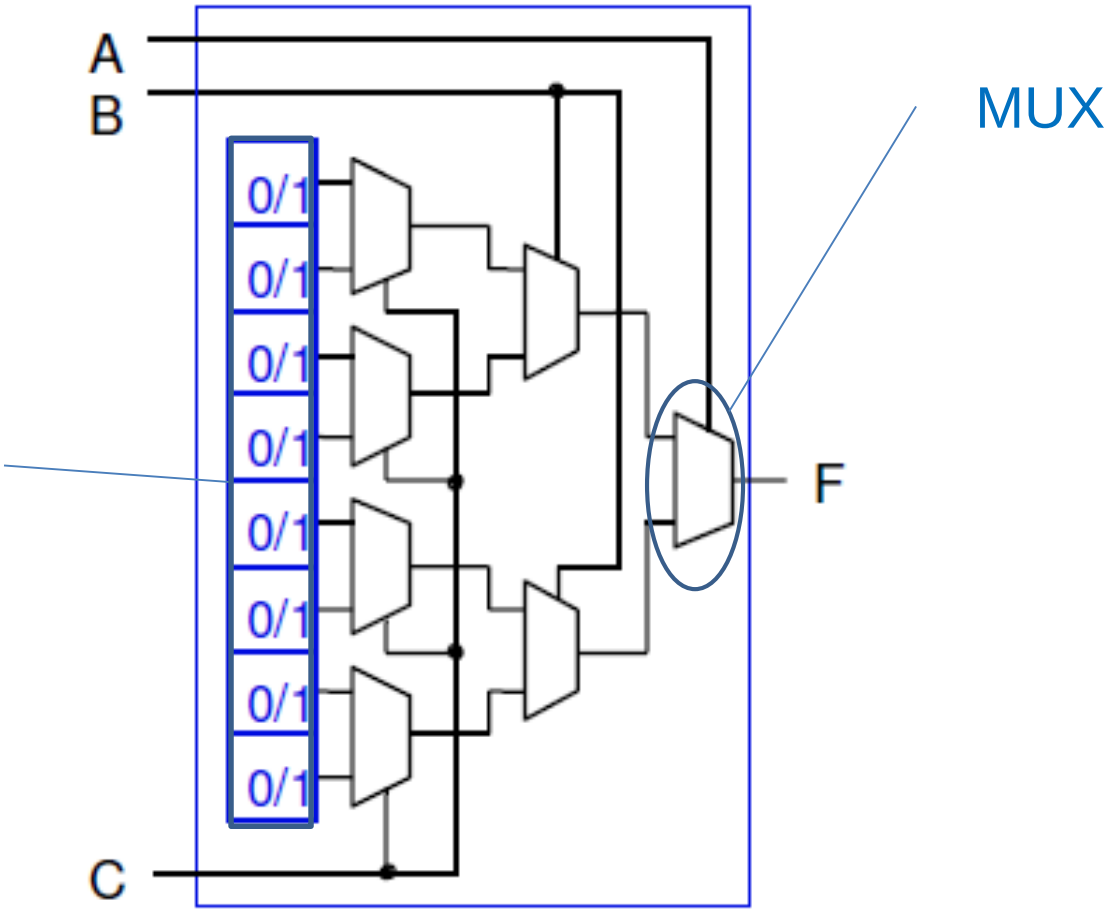
Xilinx Spartan

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

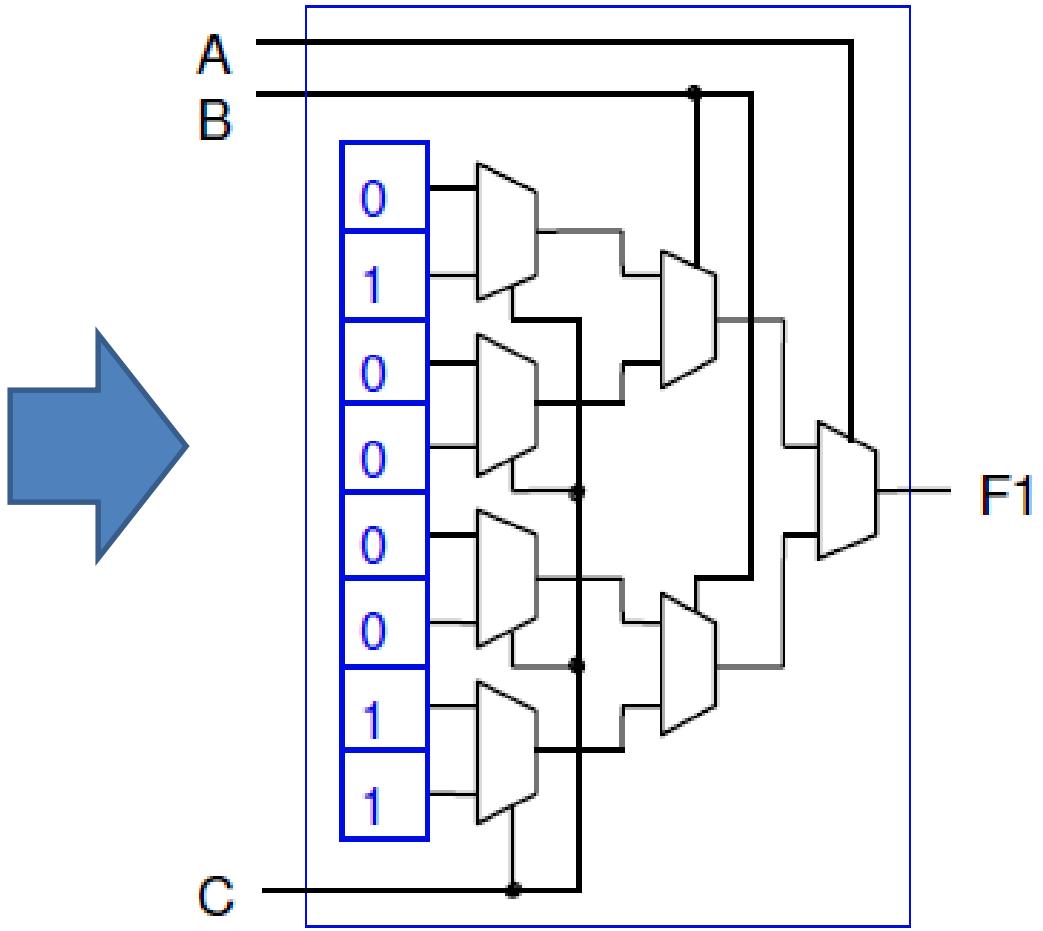
- Exemplo 1:



FPGAs: LUTs

- Exemplo 1: Programação de F1

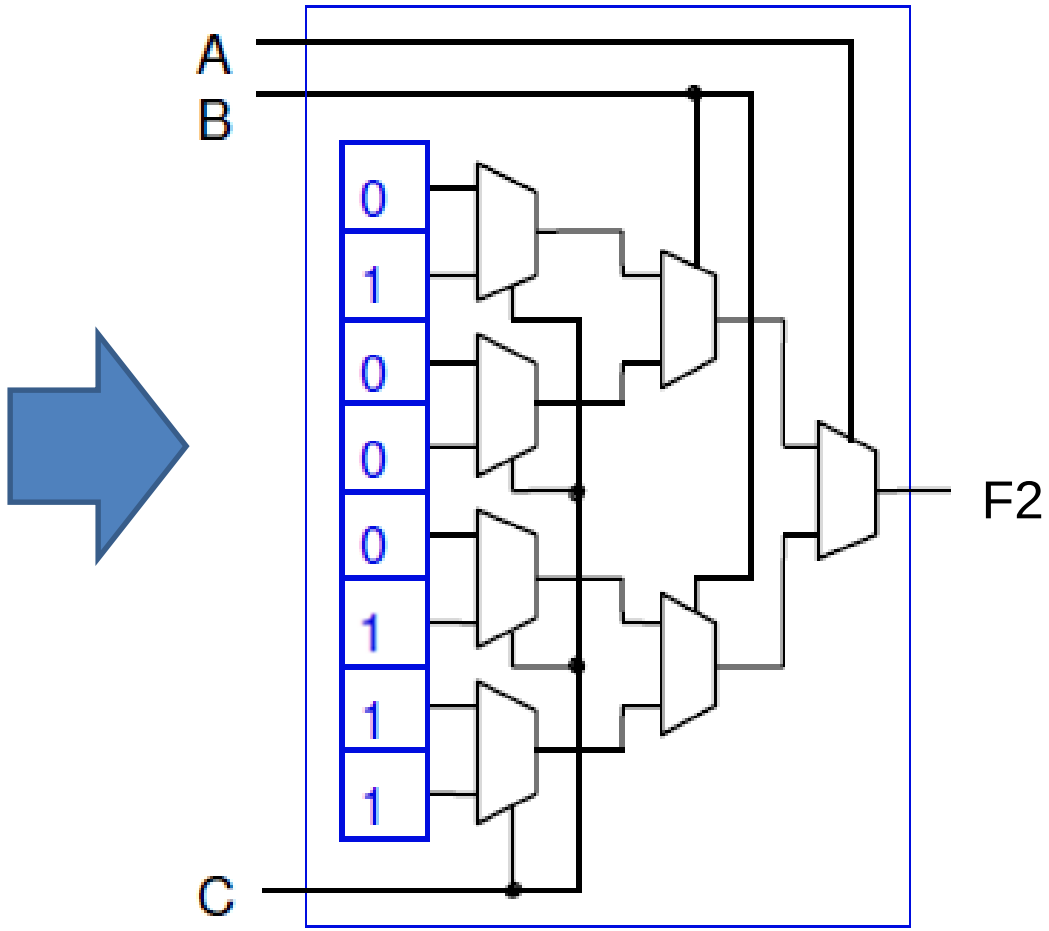
A	B	C	F1	F2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1



FPGAs: LUTs

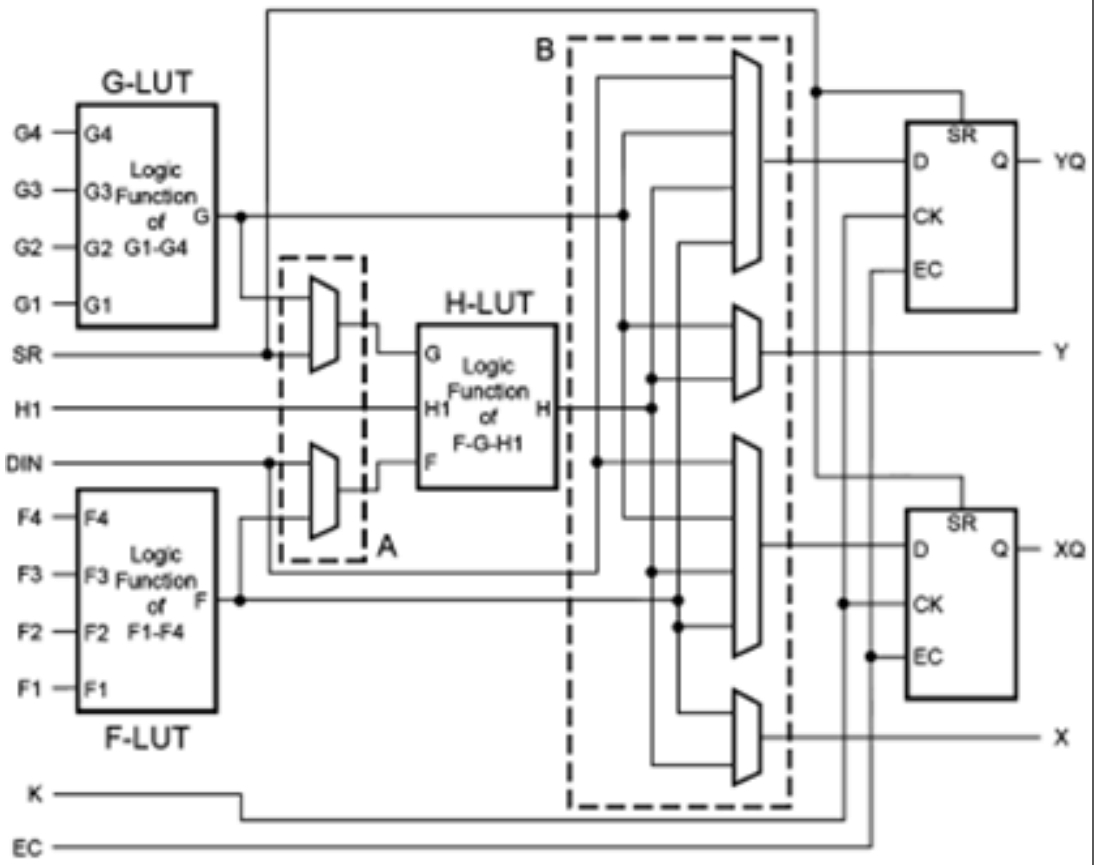
- Exemplo 1: Programação de F2

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

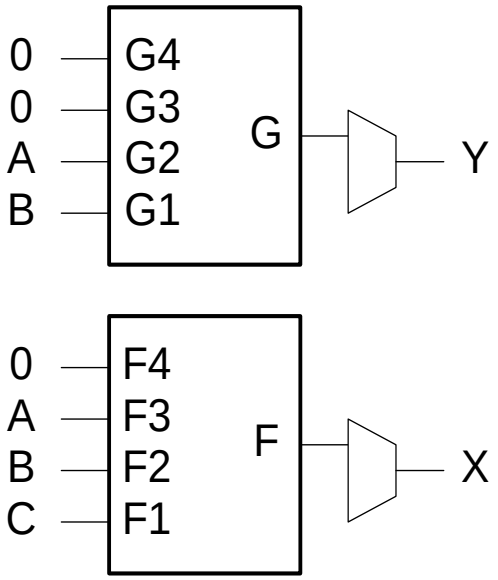
- Exemplo 2: programação das funções X e Y na Xilinx Spartan:
  - X = A'B'C + ABC'
  - Y = AB'

	(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

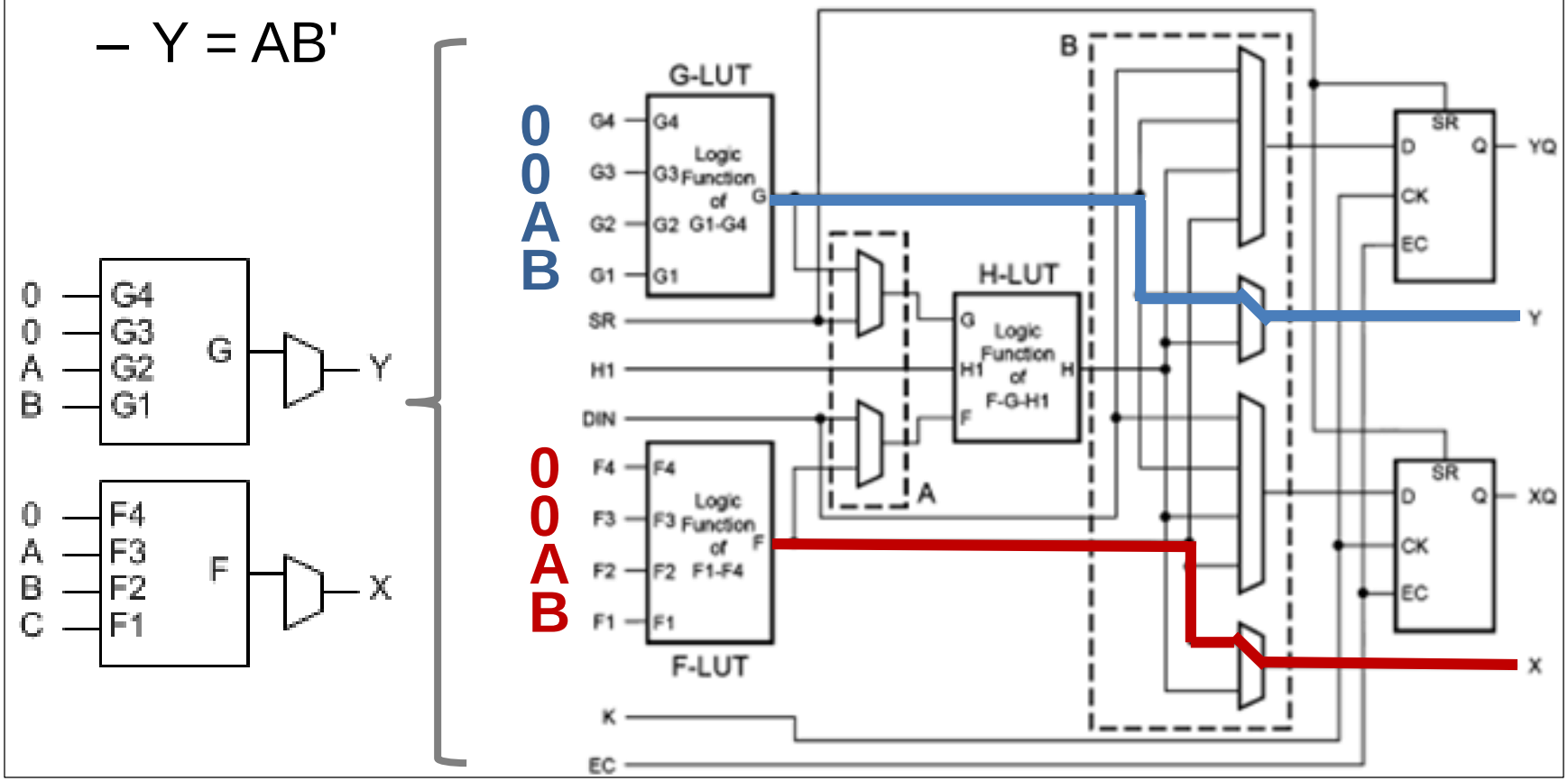
		(A)	(B)	(Y)
G4	G3	G2	G1	G
X	X	0	0	0
X	X	0	1	0
X	X	1	0	1
X	X	1	1	0

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

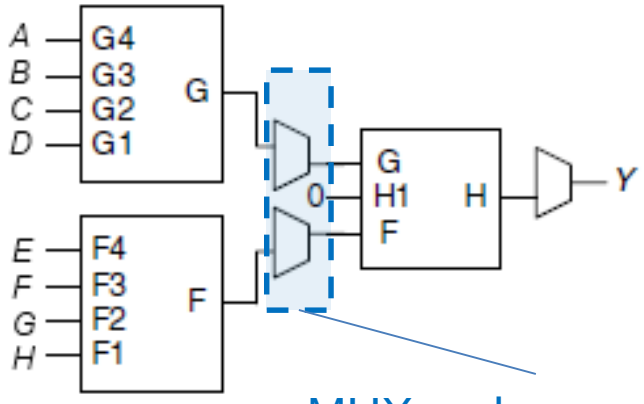
FPGAs: LUTs

- 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

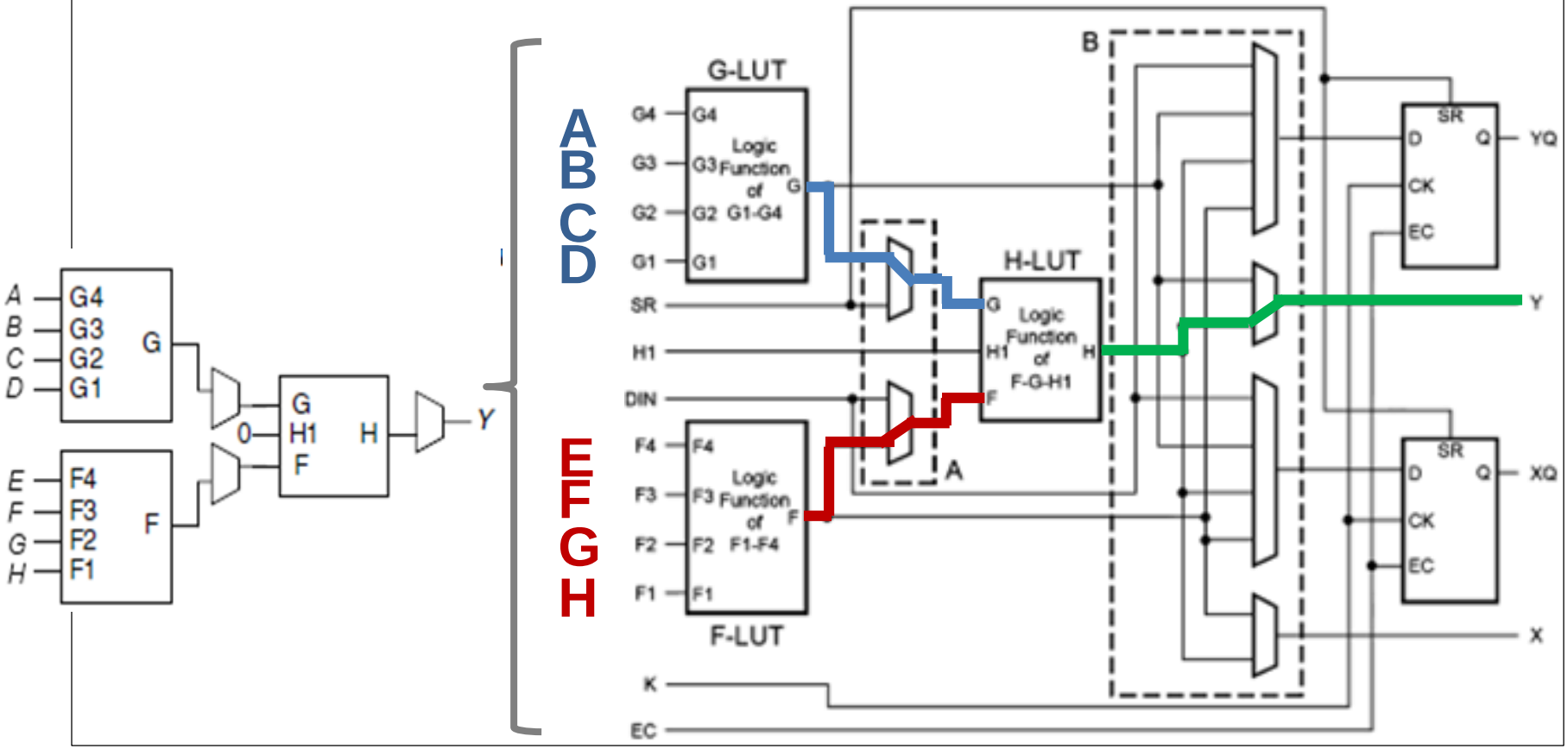
H1	F	G	H
x	0	0	0
x	0	1	0
x	1	0	0
x	1	1	1



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

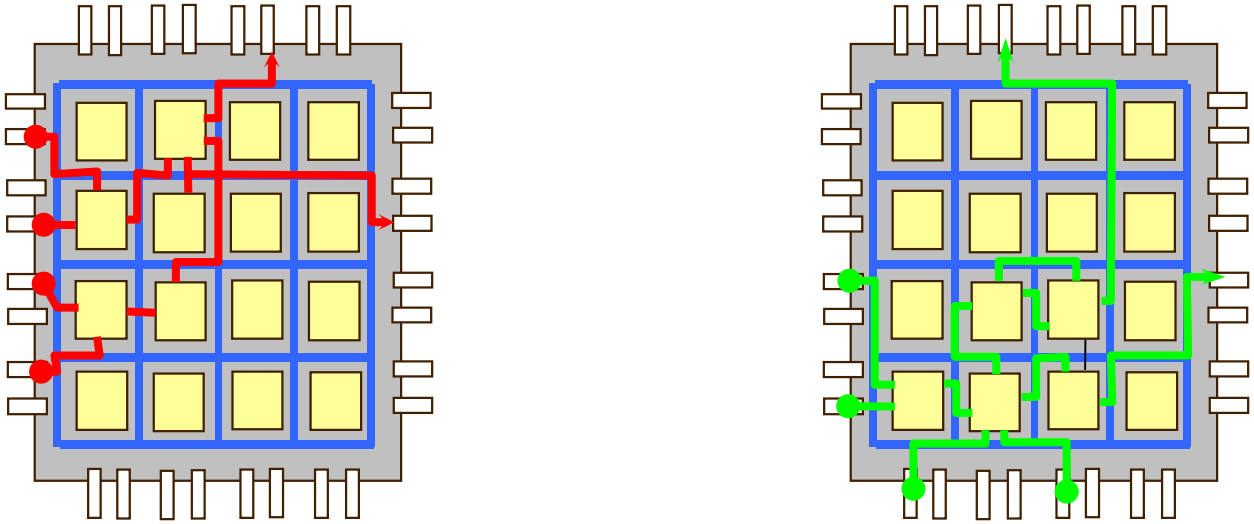


FPGAs: LUTs

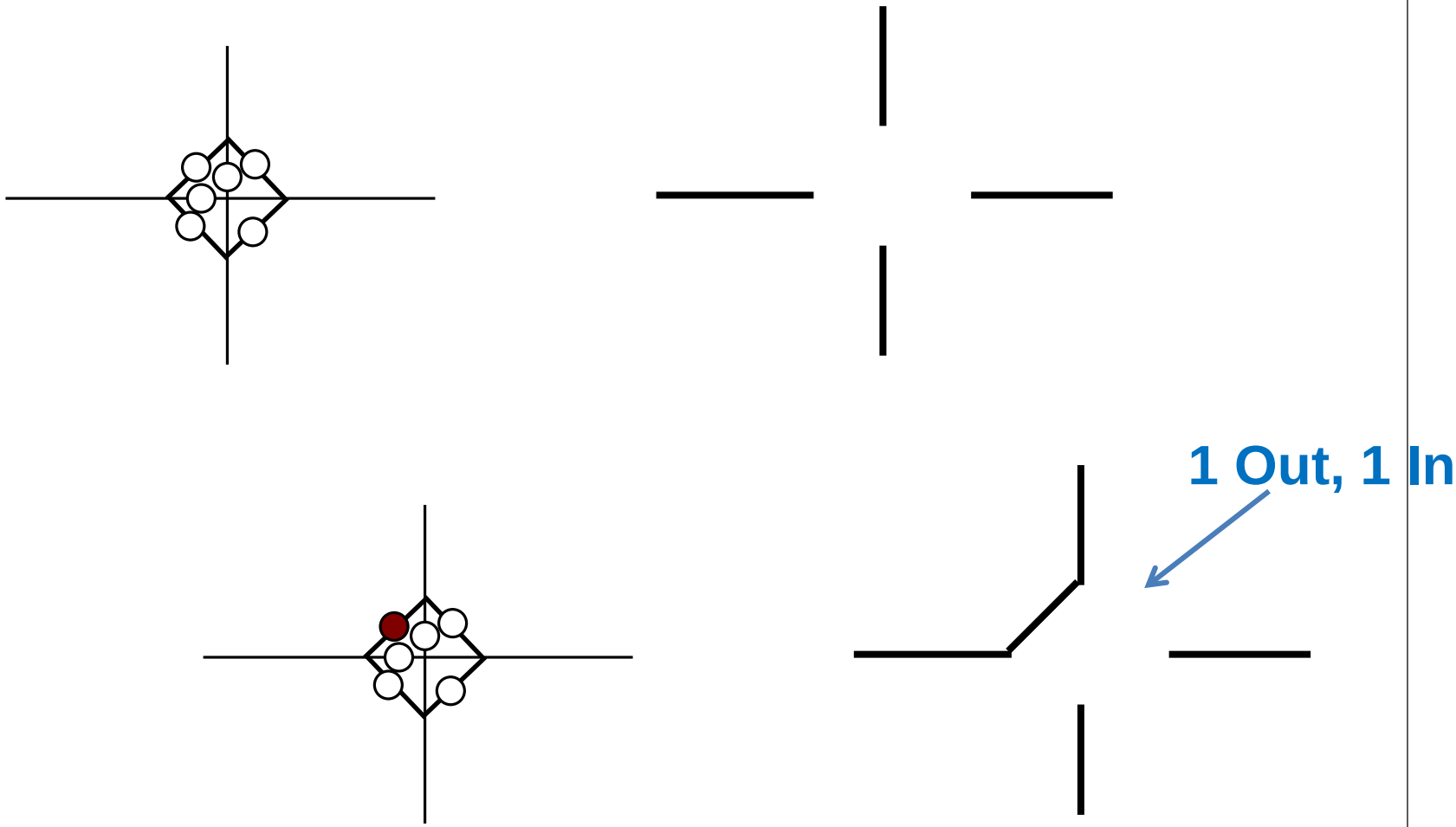
- Usando associações, cada CLB Spartan pode calcular funções de até 9 variáveis.
- E para calcular funções de mais variáveis?
  - **Associam-se alguns CLBs**, por meio das saídas combinatórias.
- Nota: ao se associar diversos CLBs, existe uma **redução na frequência máxima**
  - Afinal, sinal deverá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

FPGAs: Interconexões Programáveis

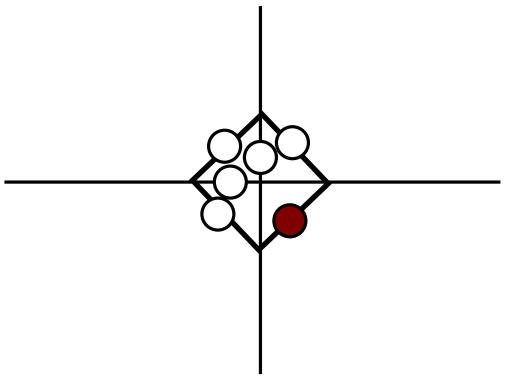
- Permitem conexão entre os CLBs
- Configuráveis de acordo com necessidade

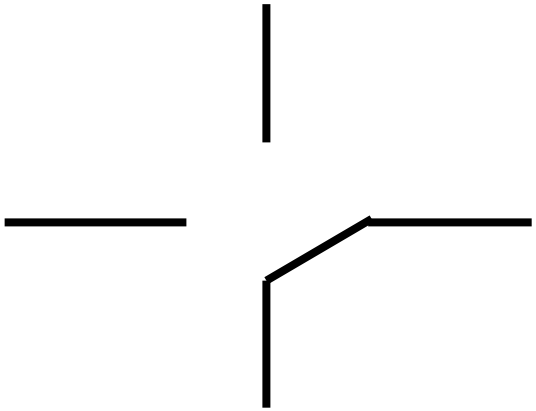


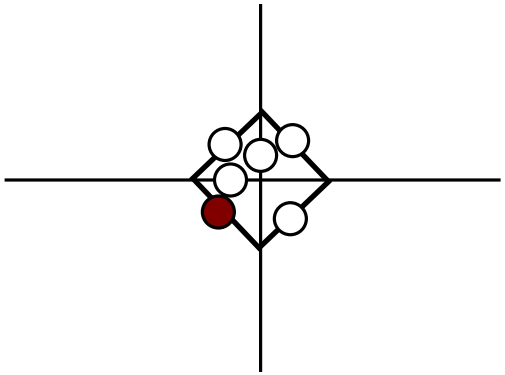
FPGAs: Interconexões Programáveis

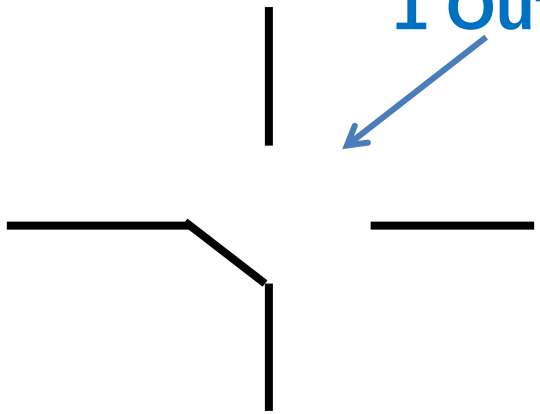


FPGAs: Interconexões Programáveis







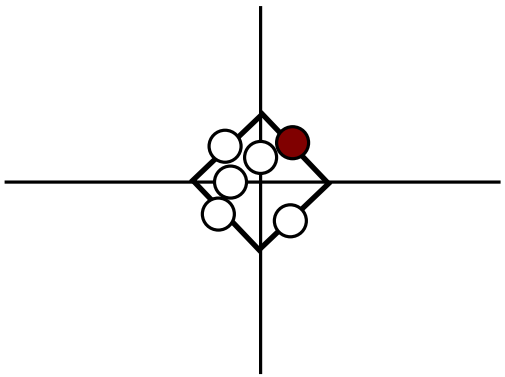


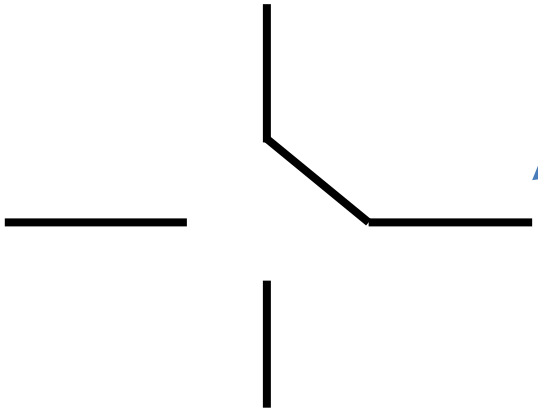
● **On**
○ **Off**

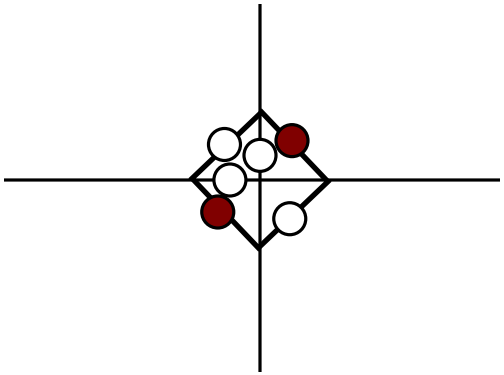
Transmissor MOS

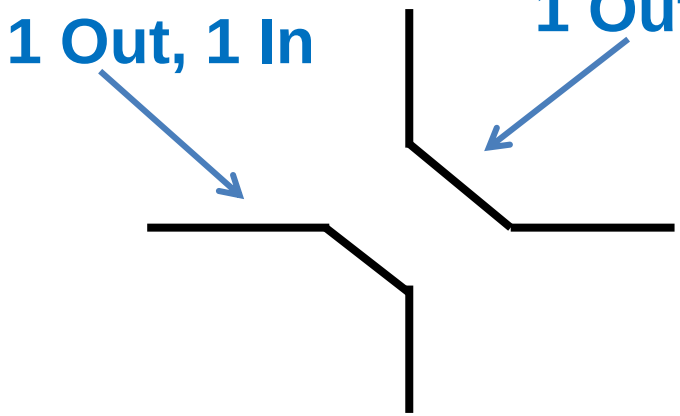
* Ligar duas saídas pode gerar curtos!

FPGAs: Interconexões Programáveis







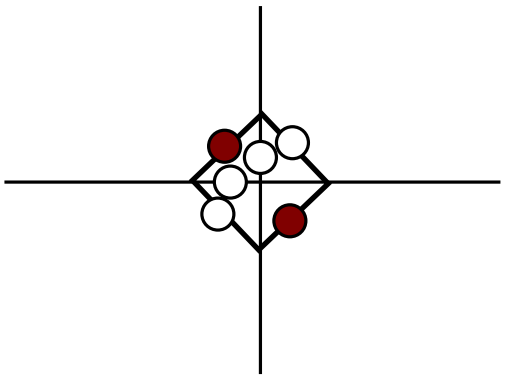


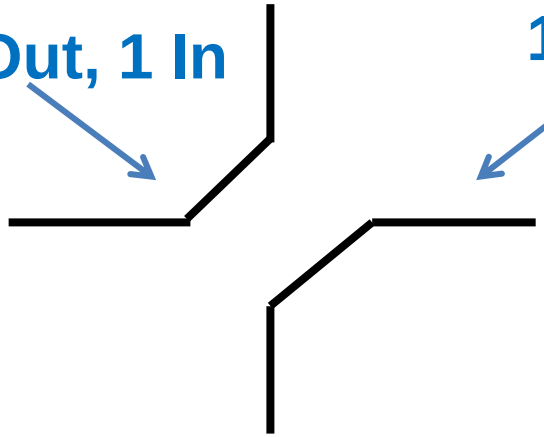
● **On**
○ **Off**

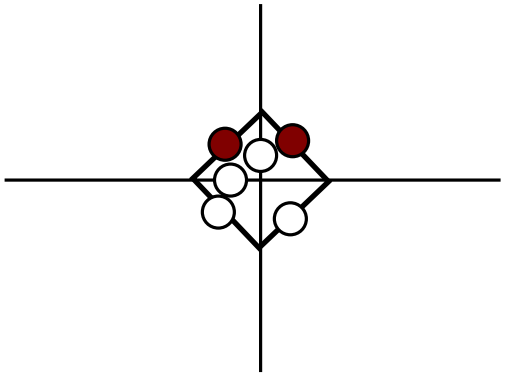
} Transmissor MOS

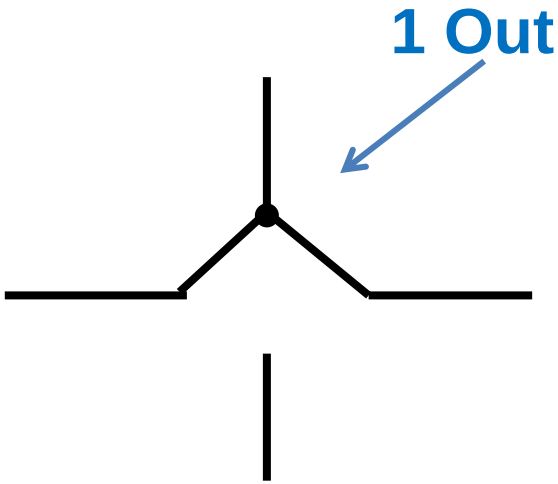
* Ligar duas saídas pode gerar curtos!

FPGAs: Interconexões Programáveis







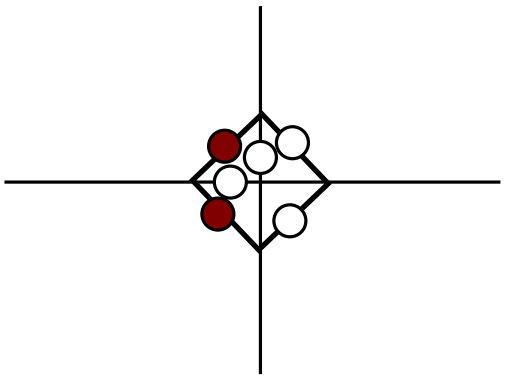


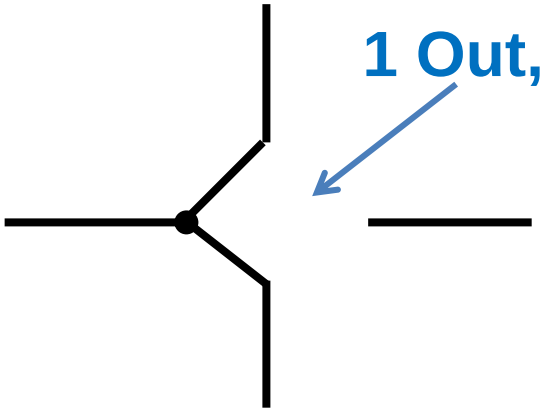
● **On**
○ **Off**

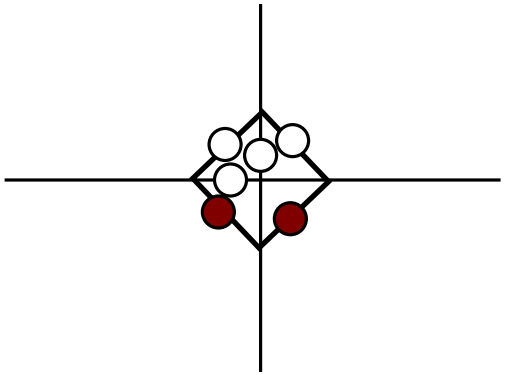
Transmissor MOS

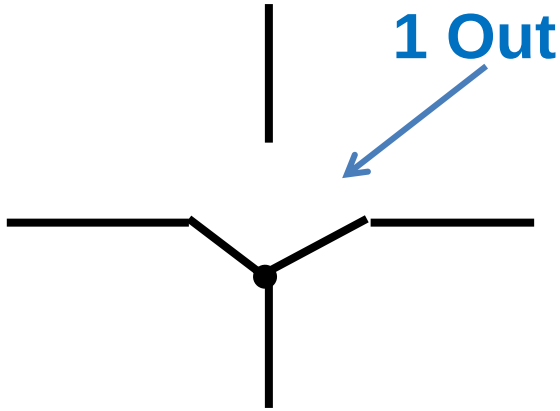
* Ligar duas saídas pode gerar curtos!

FPGAs: Interconexões Programáveis







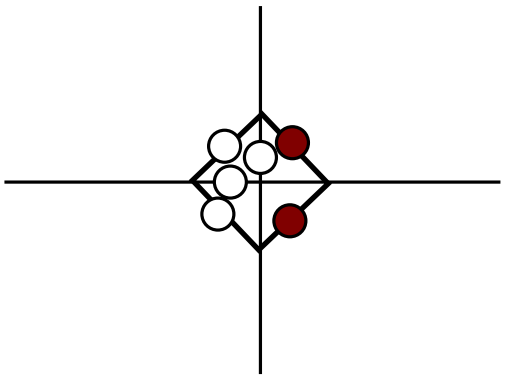


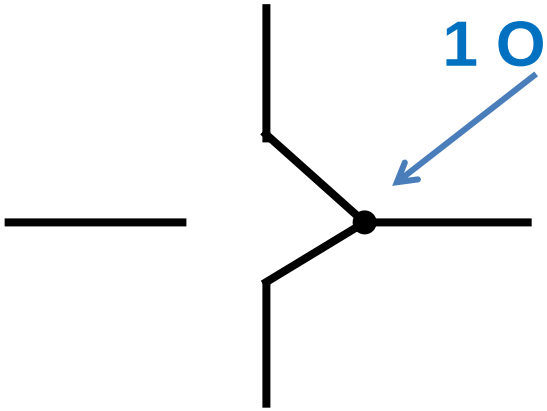
● **On**
○ **Off**

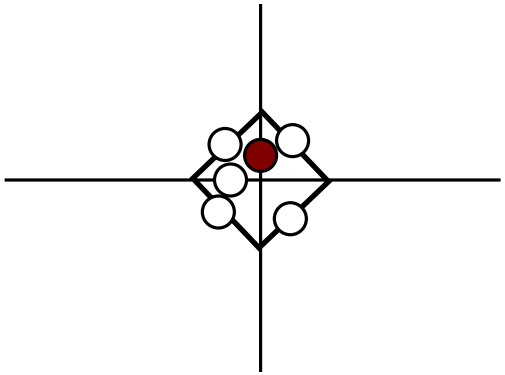
} Transmissor MOS

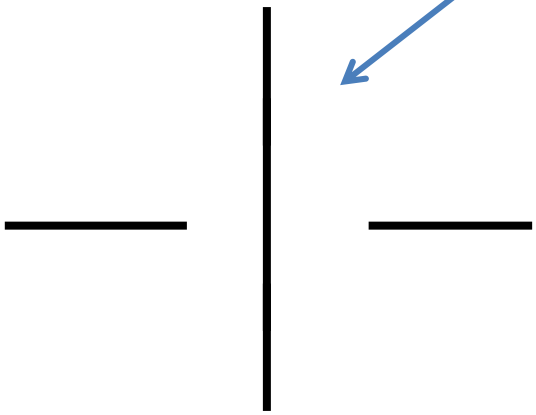
* Ligar duas saídas pode gerar curtos!

FPGAs: Interconexões Programáveis







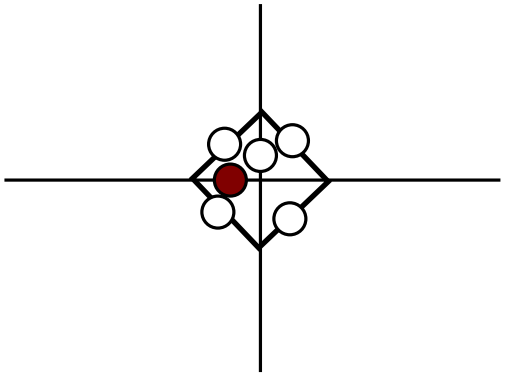


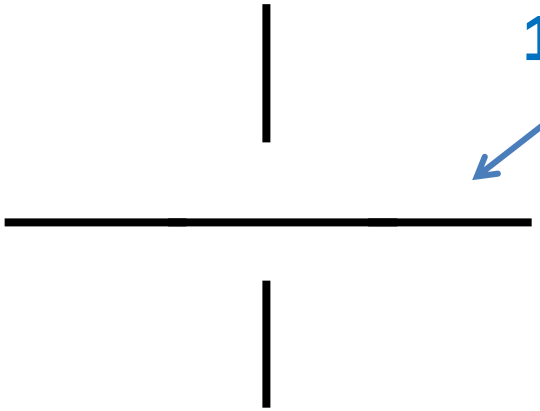
● **On**
○ **Off**

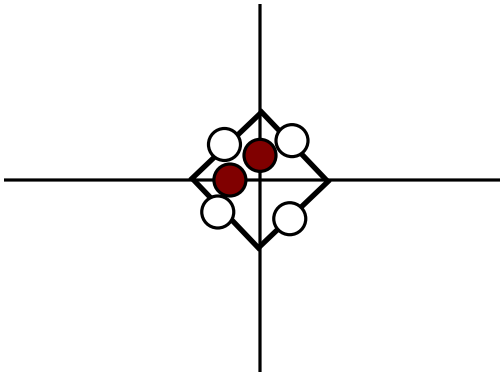
Transmissor MOS

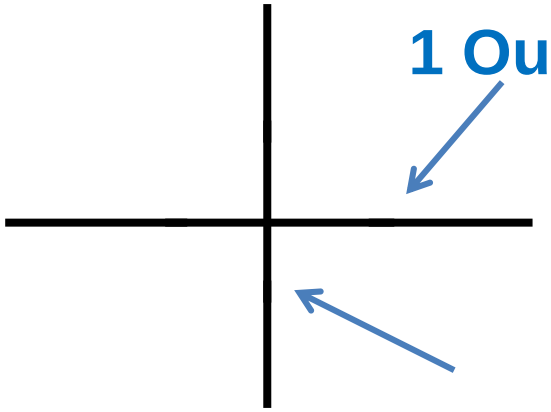
* Ligar duas saídas pode gerar curtos!

FPGAs: Interconexões Programáveis







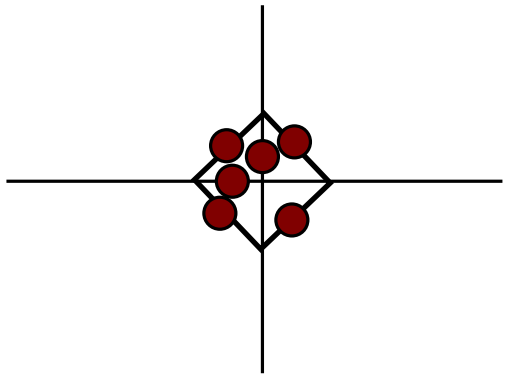


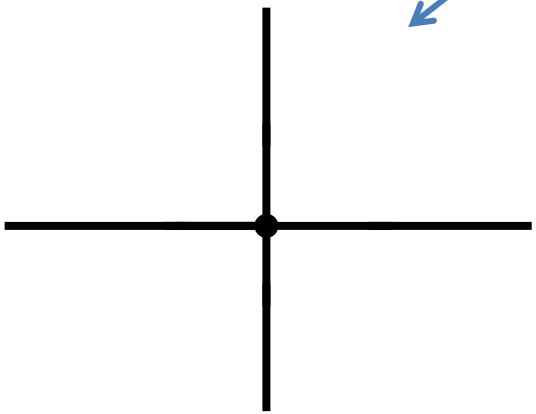
● **On**
○ **Off**

} Transmissor MOS

* Ligar duas saídas pode gerar curtos!

FPGAs: Interconexões Programáveis





1 Out, 3 In

● **On**
○ **Off**

} Transmissor MOS

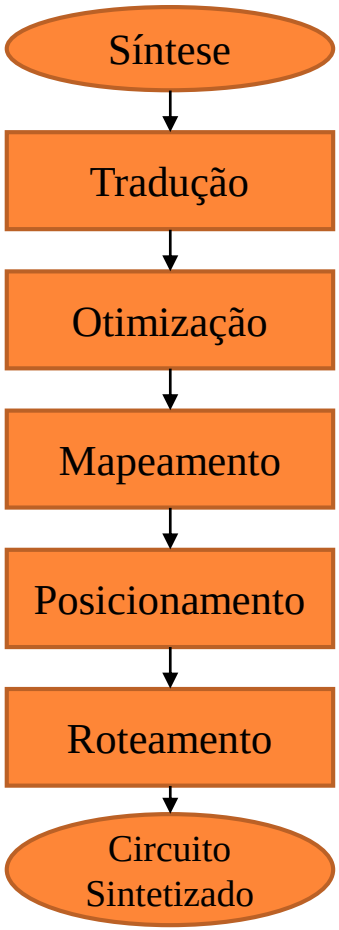
* Ligar duas saídas pode gerar curtos!

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:



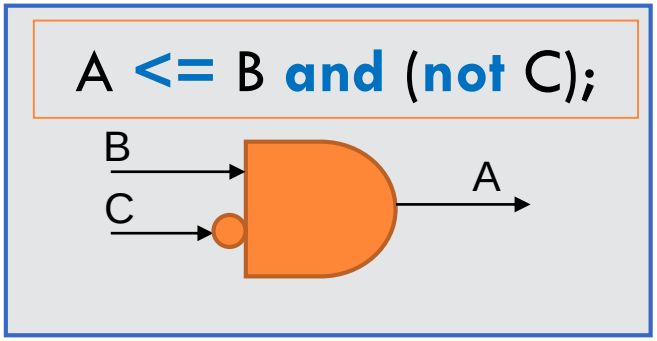
- **Tradução**: comandos VHDL são traduzidos para blocos de circuito lógico seguindo padrões pré-definidos pelo programa de síntese.
- **Otimização**: blocos padronizados são analisados com o intuito de otimizar o circuito sintetizado segundo critérios estabelecidos pelo projetista.
- **Mapeamento**: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

Projeto Lógico com FPGAs: VHDL

- A tradução de comandos VHDL para implementação em FPGA segue as seguintes regras:
  - **Atribuições**: pode implicar na utilização de um registrador mesmo quando não desejado.
  - **Case**: usam multiplexadores para compor os casos especificados. Se nem todas as alternativas forem especificadas, a síntese usa também um registrador.
  - **If**: usam multiplexadores.
  - **Comparações**: usam componentes aritméticos e comparadores.
  - **Operações funcionais**: usam componentes lógicos e aritméticos, registradores e comparadores, dependendo da operação.

Projeto Lógico com FPGAs: VHDL

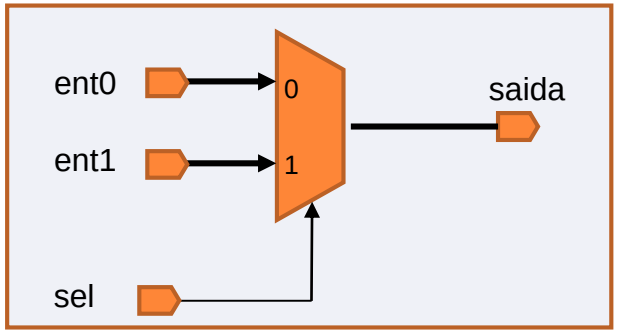
- Alguns comandos VHDL e suas traduções em FPGA
  - Lembrando: portas lógicas são implementadas com LUTs
- Em casos mais complexos o circuito sintetizado pode variar dependendo de parâmetros e requisitos de otimização (desempenho ou área).



Projeto Lógico com FPGAs: VHDL

- Síntese de with-select ou case
  - Em princípio, usam MUXes

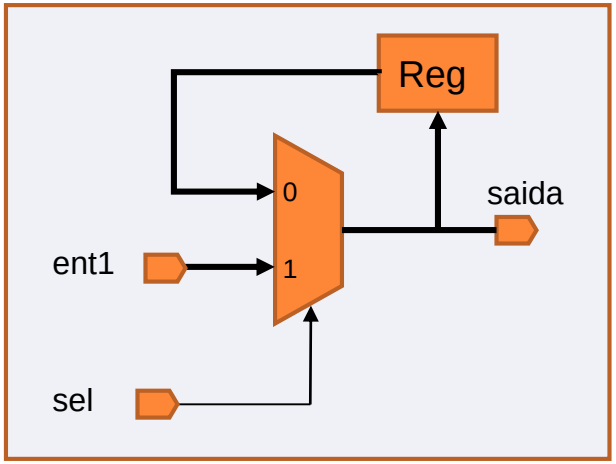
```
signal sel : std_logic
with sel select
  saida <= ent0 when '0',
           ent1 when '1',
           "0000" when others;
```



Projeto Lógico com FPGAs: VHDL

- Síntese de with-select ou case
  - Em princípio, usam MUXes.
  - Podem usar registradores para manter valor anterior se nem todos os casos forem especificados
    - Esses circuitos “com memória” (denominados “sequenciais”) são temas da disciplina de Sistemas Digitais II

```
signal sel : std_logic
with sel select
  saida <= ent1 when '1';
```

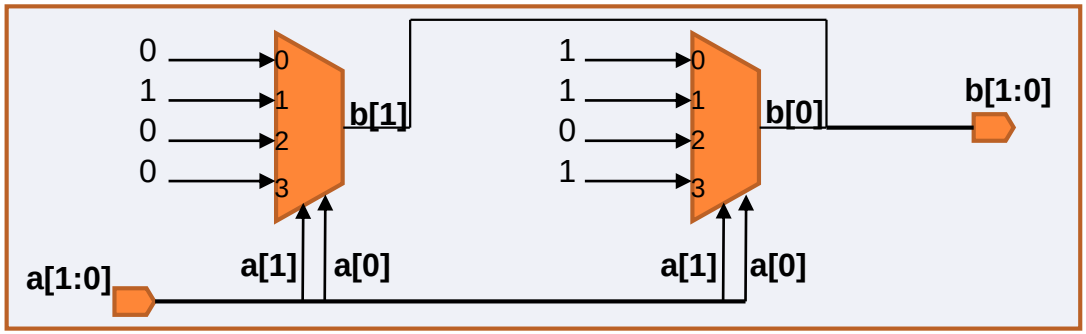


Projeto Lógico com FPGAs: VHDL

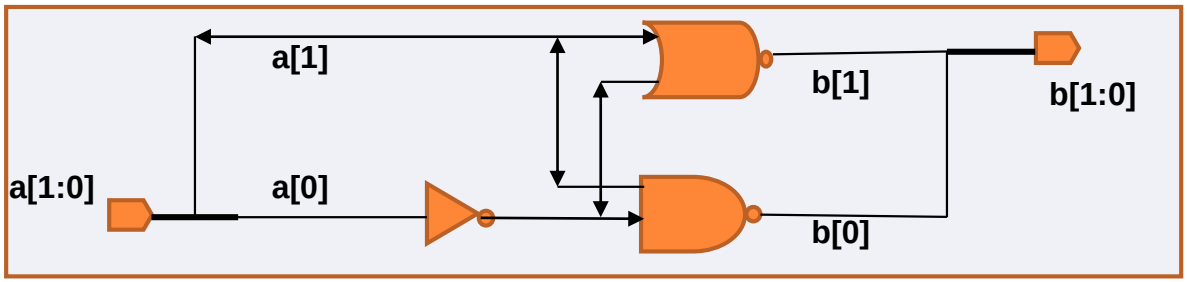
- Síntese de with-select ou case :

```
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:



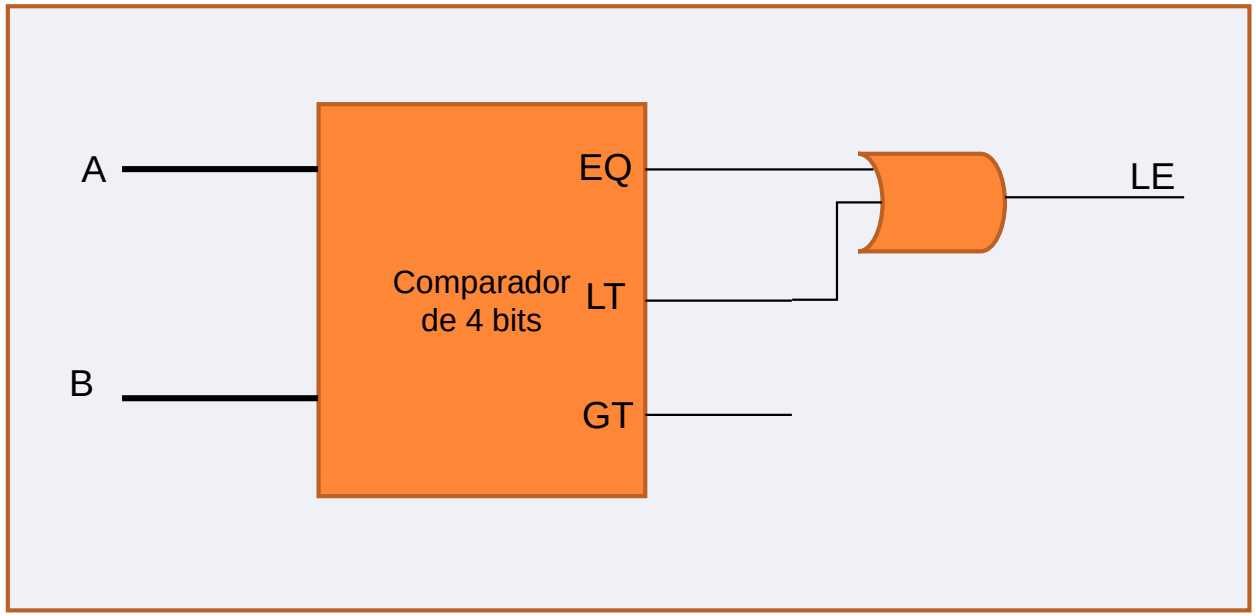
Com otimização:



Projeto Lógico com FPGAs: VHDL

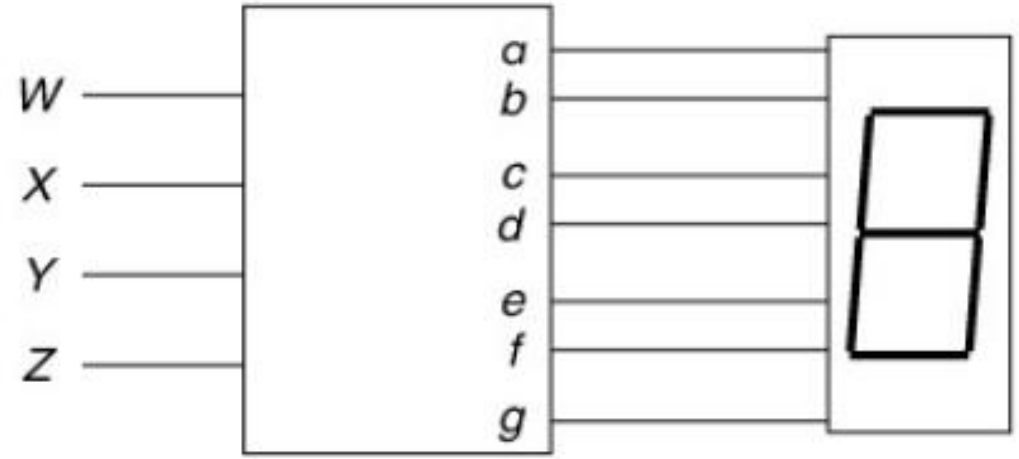
- Comparações devem ser usadas com cuidado.
  - **Comparações de inteiros** usam blocos grandes de circuito, pois não existe forma trivial de fazê-las:

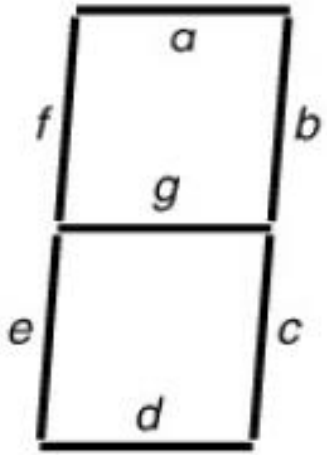
```
if (A <= B) then
```

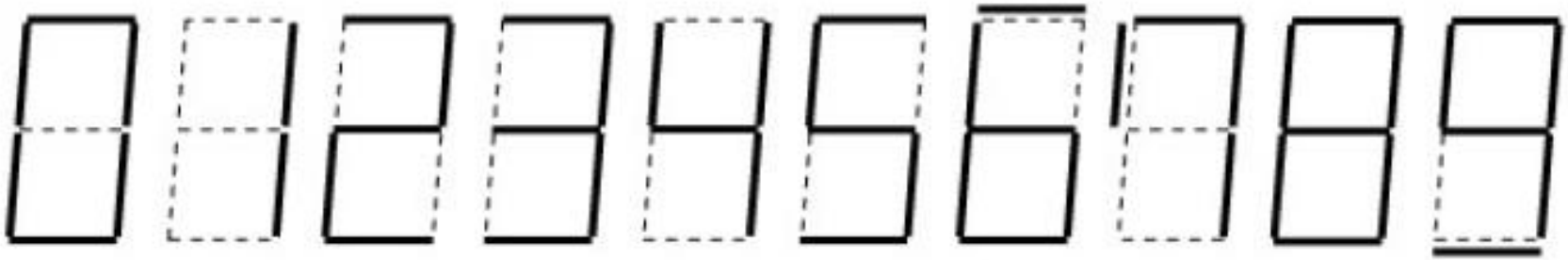


Exercício 1

- Implementação de um display de 7 segmentos usando: (1) memória e (2) FPGA







Exercício 2a

- Implemente a arquitetura do verificador de faixas de magnitude de 4 bits abaixo, usando **`with-select`**

entrada	saida
0000 a 0011	100
0100 a 1001	010
1010 a 1111	001
valor desconhecido	000

```
architecture arch of checkMag is
begin
   with entrada select
      saida <=    "100"   when "0000"|"0001"|"0010"|"0011",
                  "010"   when "0100"|"0101"|"0110"|"0111"|"1000"|"1001",
                  "001"   when "1010"|"1011"|"1100"|"1101"|"1110"|"1111",
                  "000"   when others; -- "catch all"
end arch;
```

Exercício 2b

- Reimplemente a arquitetura do verificador de faixas de magnitude usando **comandos de comparação**
  - O circuito criado é mais ou menos eficiente do que o do item 2a?

entrada	saida
0000 a 0011	100
0100 a 1001	010
1010 a 1111	001
valor desconhecido	000

```
architecture arch of checkMag is
-- sinal extra para enxergar entrada como "unsigned"
signal u_entrada : unsigned(3 downto 0);
begin
   u_entrada <= unsigned(entrada);
   saida <= "100" when u_entrada <= "0011" else
            "010" when u_entrada >= "0100" and u_entrada <= "1001" else
            "001" when u_entrada >= "1010" else
            "000"; -- "catch all": omissão pode levar sintetizador a
                   -- inserir registrador para guardar valor da saída
end arch;
```

Eficiência:
1 MUX 16:1 ➔ 4 comparadores...

Lição de Casa

- Leitura Obrigatória:
  - Capítulo 9 do Livro Texto.
- Exercícios Obrigatórios:
  - Capítulo 9 do Livro Texto.

Memórias “por dentro”: exemplos

APÊNDICE

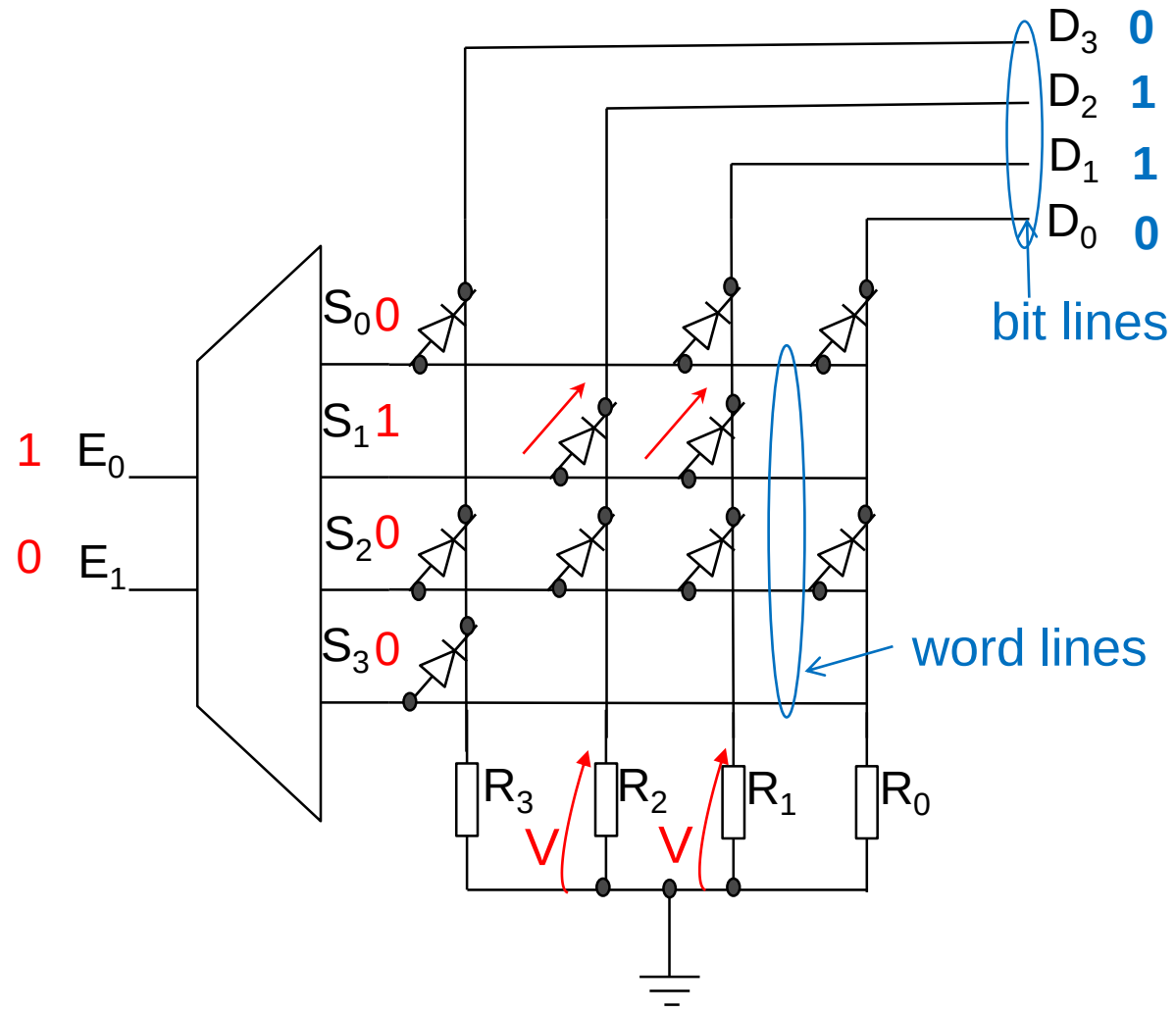
4. Tipos de Memórias a Semicondutor: ROM

Ex.: ROM de 4 palavras de 4 bits

E_1	E_0		D_3	D_2	D_1	D_0		S_3	S_2	S_1	S_0		D_3	D_2	D_1	D_0
0	0		1	0	1	1		0	0	0	1		1	0	1	1
0	1		0	1	1	0	=>	0	0	1	0		0	1	1	0
1	0		1	1	1	1		0	1	0	0		1	1	1	1
1	1		1	0	0	0		1	0	0	0		1	0	0	0
End.			Dados					Seleção Interna					Dados			

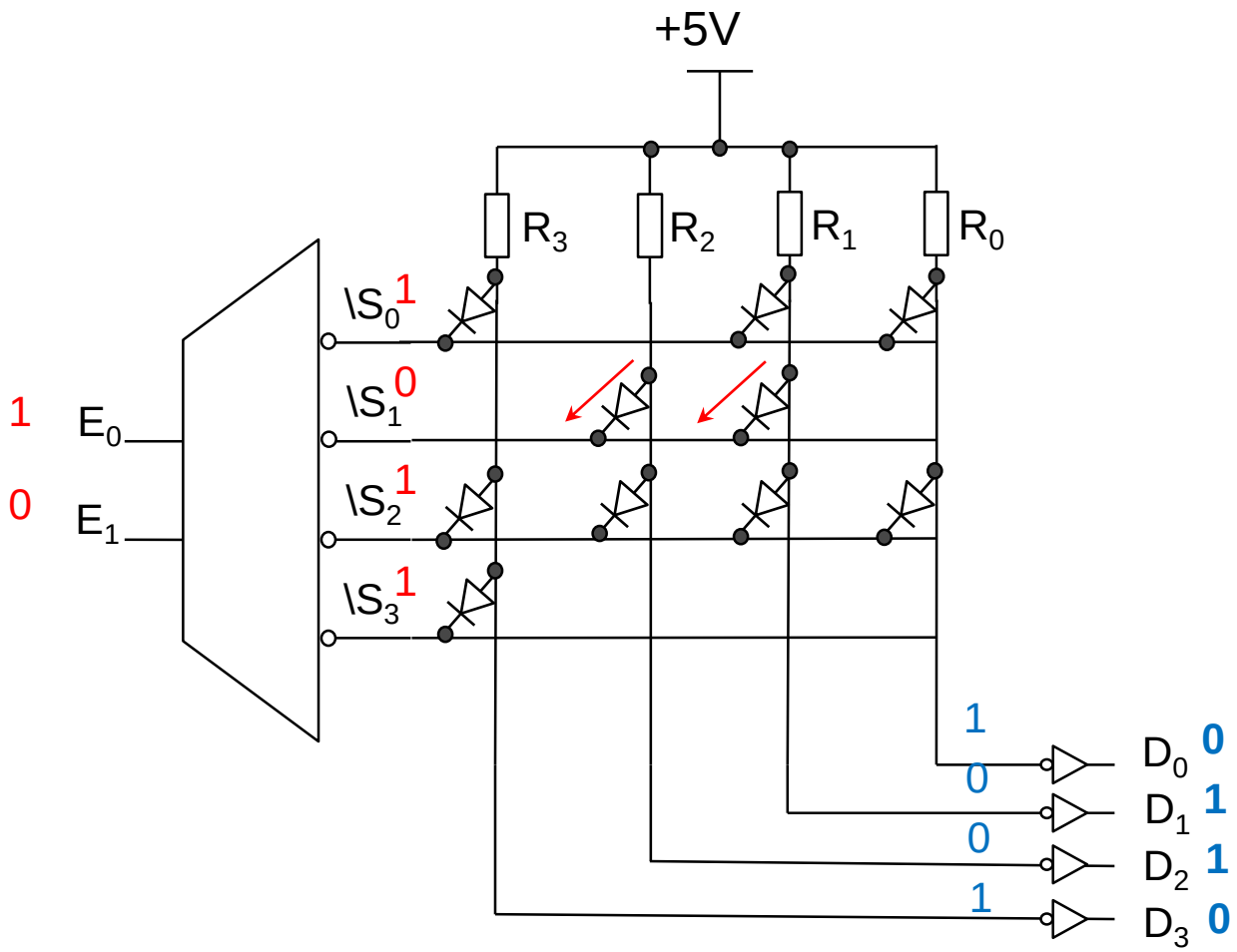
4. Tipos de Memórias a Semicondutor: ROM

Ex.: ROM de 4 palavras de 4 bits



4. Tipos de Memórias a Semicondutor: ROM

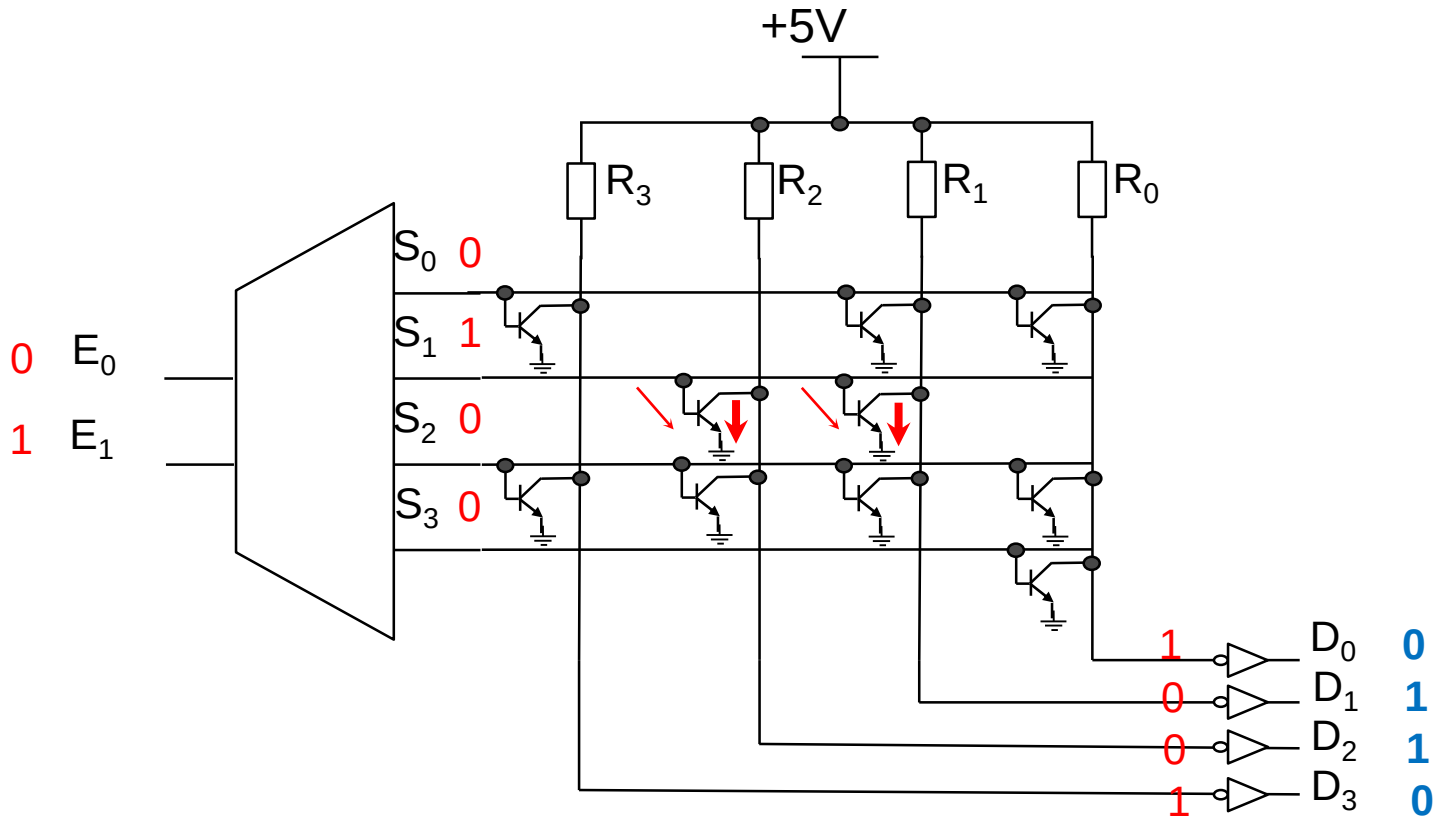
Ex.: ROM de 4 palavras de 4 bits



4. Tipos de Memórias a Semicondutor: ROM

Inconveniente: S_i **deve ter alto *fan-out*.**

Solução: Transistor ➔ I_c drenada da fonte, reduzindo corrente $I_b = (I_c/\beta)$ drenada de S_i



4. Tipos de Memórias a Semicondutor: MROM

MROM – *Mask-programmed ROM*

– **Características**:

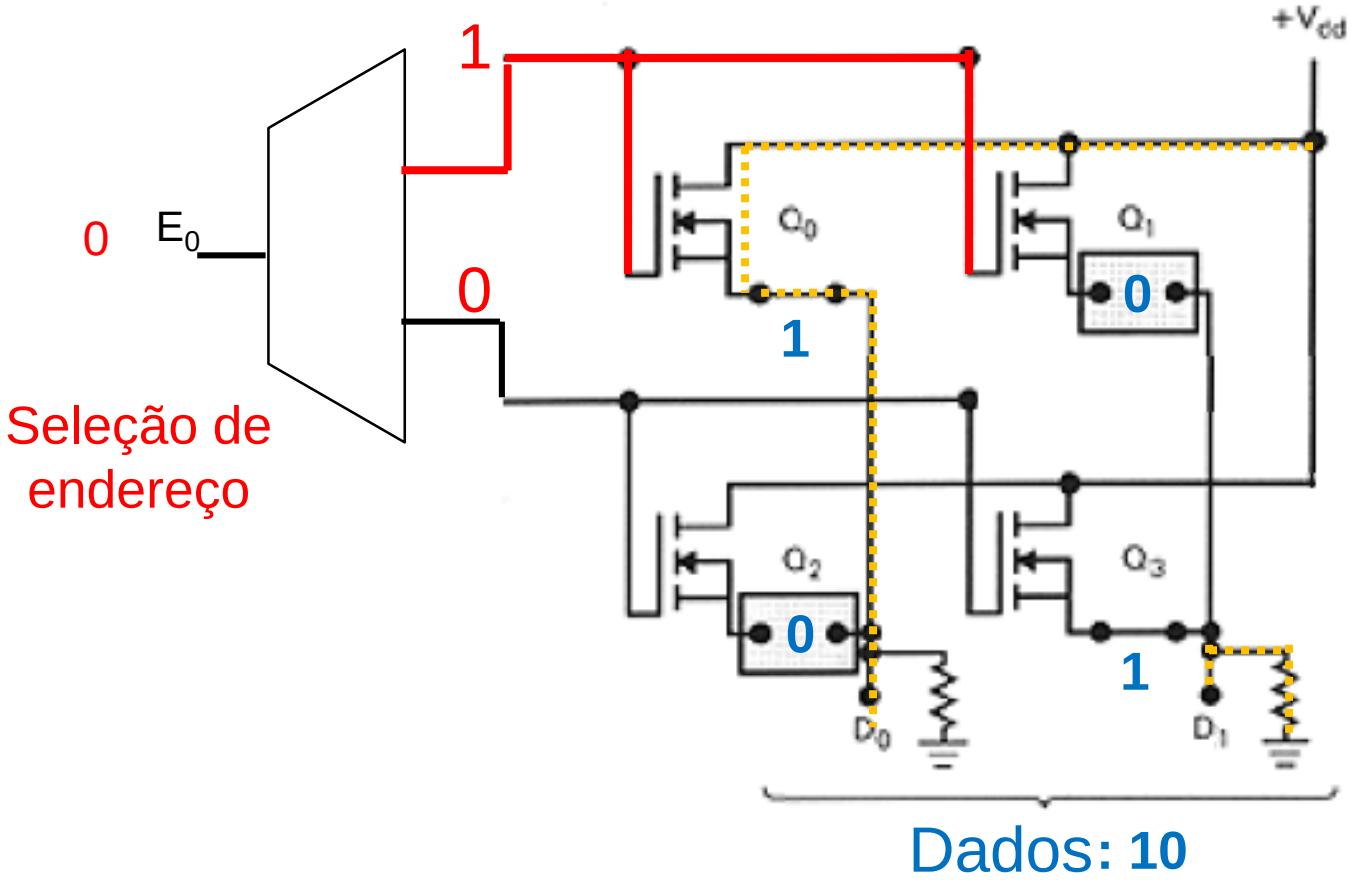
- » Programada na fabricação do circuito integrado
- » As informações são armazenadas conectando ou desconectando o *Gate* fonte de um transistor MOSFET à coluna de saída (MROM MOS).
- » Programação na etapa de metalização do circuito: uma **máscara** para depositar metais sobre o silício.
- » Alto custo total / baixo custo unitário

– **Utilização:**

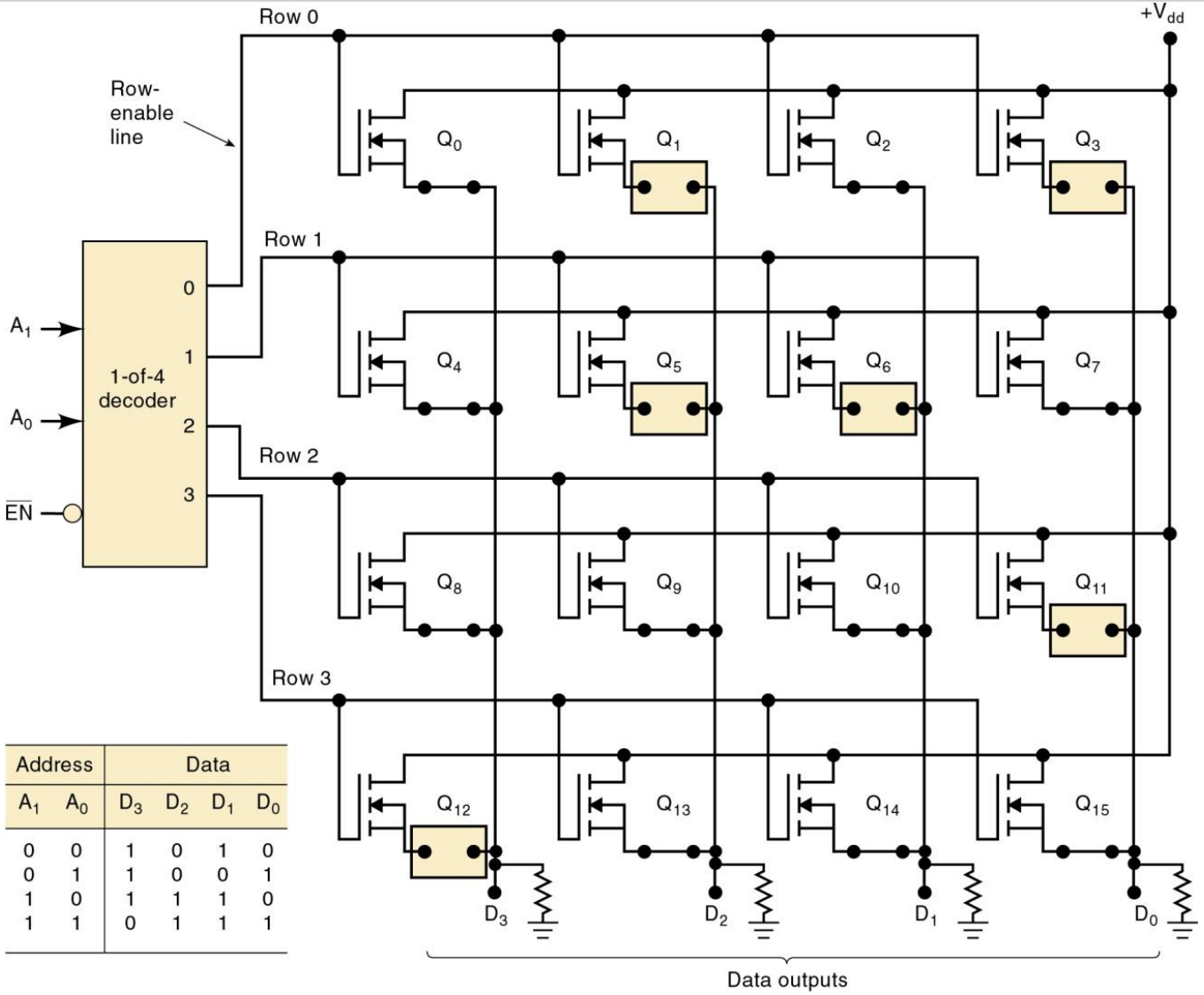
- » Armazenamento de tabelas de funções matemáticas, códigos: uso geral.
- » Programas de dispositivos produzidos em larga escala

4. Tipos de Memórias a Semicondutor: MROM

- Exemplo: MROM 2x2
  - Transistor: conectado = 1, desconectado = 0



4. Tipos de Memórias a Semicondutor: MROM



Address		Data			
A_1	A_0	D_3	D_2	D_1	D_0
0	0	1	0	1	0
0	1	1	0	0	1
1	0	1	1	1	0
1	1	0	1	1	1

Fonte: Digital Systems: Principles and Applications, 11ed. Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss

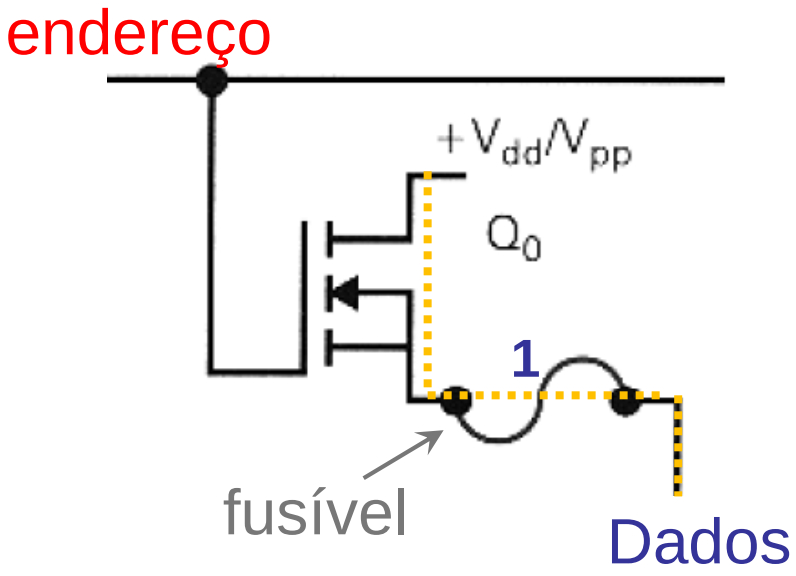
4. Tipos de Memórias a Semicondutor: PROM

PROMs -Programmable ROM

–Programação feita pelo usuário e fixa (uma só vez).

–Programador (queimador) específico para cada tipo de PROM.

–Tempo de programação: ~ 2min.



Inicialmente: Valor lógico 1 armazenado em todos os bits.

4. Tipos de Memórias a Semicondutor: PROM

PROMs -Programmable ROM

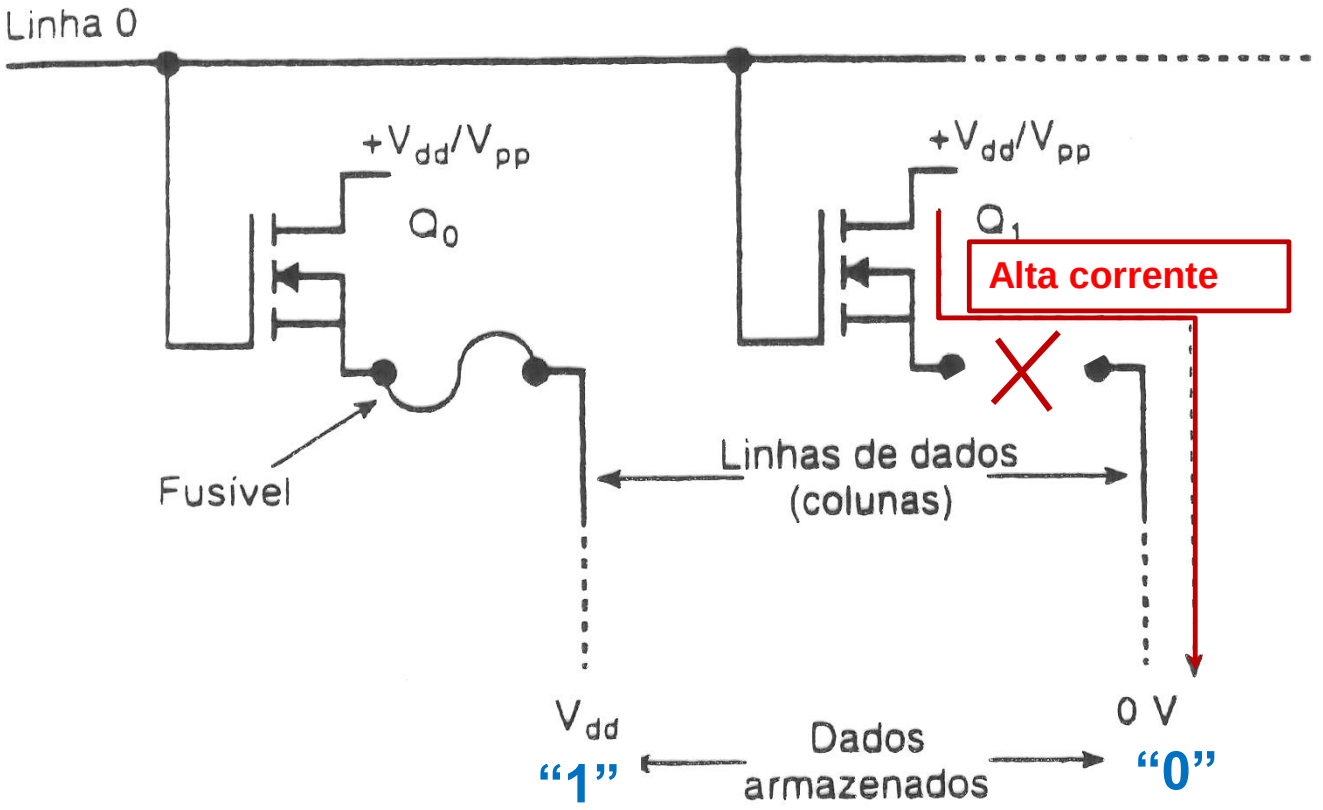
–**Programação**: queimar fusíveis onde deve haver <u>zero</u>

» Aplicação de 10 a 30V em um pino especial na pastilha

» Correntes/Tensões elevadas no nó (valor e tempo de aplicação depende da PROM)

–**Utilização**: produtos de escala menor do que ROM.

4. Tipos de Memórias a Semicondutor: PROM



As PROMs utilizam fusíveis que podem ser abertos seletivamente pelo usuário para programar o nível lógico 0 na célula.