PCS 3115 (PCS2215)

Sistemas Digitais I

Circuitos Combinatórios

Blocos Básicos

Prof. Dr. Marcos A. Simplicio Jr.

versão: 3.1 (Jan/2020)

Blocos básicos

- Codificadores e Decodificadores
  - Drivers de Display
  - Transcoders (BCD – 7 segmentos)
- (De) Multiplexadores e portas tri-state
- Exercícios

Contexto

- Circuitos SSI: *small scale integration*
  - Até ~20 transistores
  - Circuitos bastante simples, como portas lógicas NOT e AND, OR, XOR de 3 entradas
- Circuitos MSI: *medium scale integration*
  - Até ~200 transistores
  - São os circuitos que veremos neste módulo: codificadores, multiplexadores, somadores, etc.

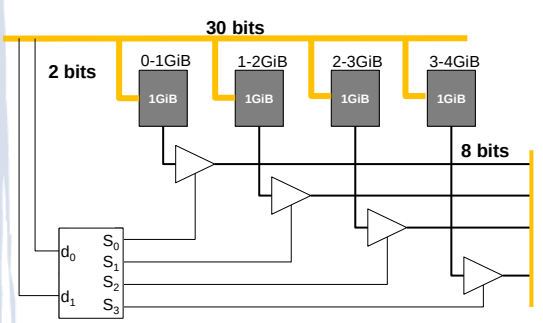
(DE)CODIFICADORES

Codificador

- **Converte** código de entrada em código de saída
- Nomenclatura comum (Wakerly):
  - Codificador/Encoder: #bits de entrada > #bits de saída
  - Decodificador/Decoder: #bits de entrada < #bits de saída
- Aqui: "Codificador" para ambos os tipos
  - E ver **problemas** interessantes que eles resolvem!

Codificador

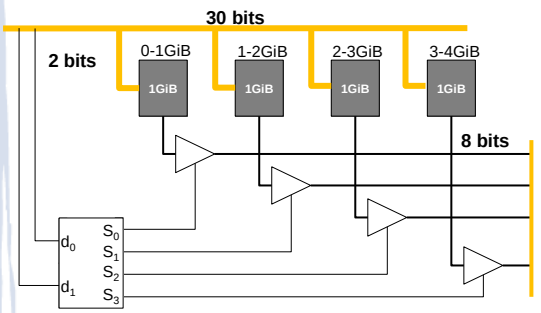
- **Problema**: leitura de dados em memória de 4 GiB
  - 4 Chips de 1 GiB: 30 bits de endereço
  - Entrada do Codificador: endereço de 2 bits
  - Saída: código de ativação do chip correspondente
    - Obs.: na prática buffer tri-state é interno a chip ("Chip-Select")



ENTRADA		SAÍDA			
d_1	d_0	S_3	S_2	S_1	S_0
?	?	?	?	?	?

Codificador

- **Problema**: leitura de dados em memória de 4 GiB
  - 4 Chips de 1 GiB: 30 bits de endereço
  - Entrada do Codificador: endereço de 2 bits
  - Saída: código de ativação do chip correspondente
    - Obs.: na prática buffer tri-state é interno a chip ("Chip-Select")



Codificador 1-de-m

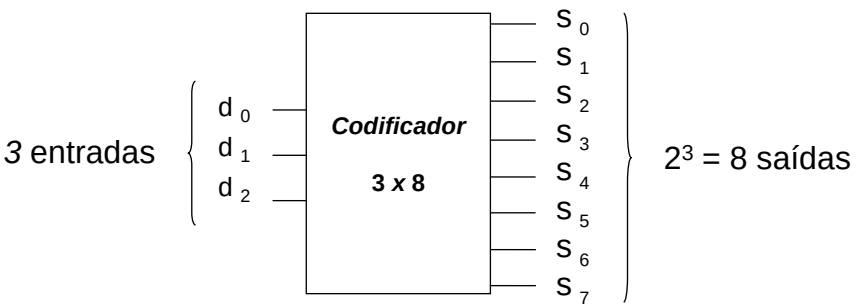
ENTRADA		SAÍDA			
d_1	d_0	S_3	S_2	S_1	S_0
0	0	0	0	0	**1**
0	1	0	0	**1**	0
1	0	0	**1**	0	0
1	1	**1**	0	0	0

Codificador: 1-de-m

- Codificador para código 1-de-m
  - Para cada combinação de valores das n entradas, **apenas uma saída é ativada**
- Funcionamento:
  - Entradas formam uma palavra binária de n bits
  - Palavra de entrada pode assumir os valores de 0 a $2^n - 1$
  - As 2^n saídas são numeradas de 0 a $2^n - 1$
  - Saída ativada é aquela cujo índice corresponde ao valor da palavra binária de entrada
- Exemplo (3 bits: "codificador 3x8" ou "1-de-8")
  - Entradas = 011 (3_{10}); ativa-se a saída S_3
  - Entradas = 101 (5_{10}); ativa-se a saída S_5

Codificador 1-de-m (3 bits)

- Símbolo funcional (convenção):
  - Entradas à esquerda
  - Saídas à direita
  - Índice menor indica bit menos significativo na palavra binária



Codificador de 3 bits

- Tabela verdade

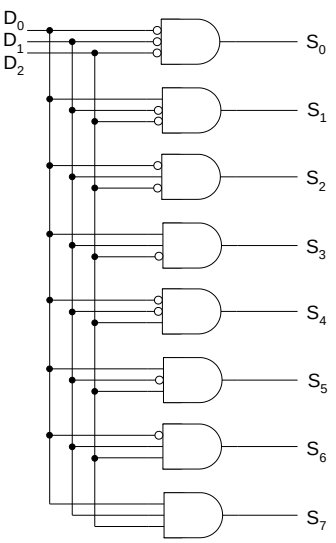
Saída S_2 é ativada...

2 em decimal

ENTRADAS			SAÍDAS							
d_2	d_1	d_0	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
0	0	0	0	0	0	0	0	0	0	**1**
0	0	1	0	0	0	0	0	0	**1**	0
0	1	0	0	0	0	0	0	**1**	0	0
0	1	1	0	0	0	0	**1**	0	0	0
1	0	0	0	0	0	**1**	0	0	0	0
1	0	1	0	0	**1**	0	0	0	0	0
1	1	0	0	**1**	0	0	0	0	0	0
1	1	1	**1**	0	0	0	0	0	0	0

Codificador de 3 bits

Circuito interno: mintermos!



$S_0 = D_2' .D_1' .D_0'$

$S_1 = D_2' .D_1' .D_0$

$S_2 = D_2' .D_1 .D_0'$

$S_3 = D_2' .D_1 .D_0$

$S_4 = D_2 .D_1' .D_0'$

$S_5 = D_2 .D_1' .D_0$

$S_6 = D_2 .D_1 .D_0'$

$S_7 = D_2 .D_1 .D_0$

$S_i = m_i$

Codificadores: VHDL

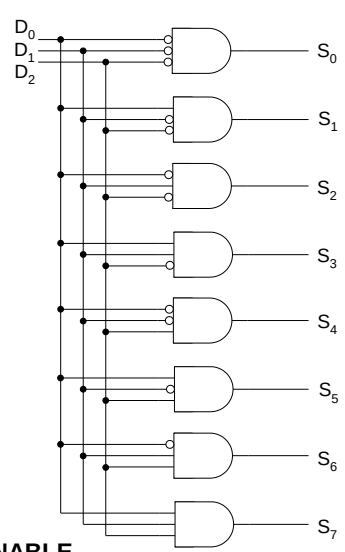
```
library IEEE;
use IEEE.std_logic_1164.all;

entity encoder is -- codificador 3x8
    port (d : in STD_LOGIC_VECTOR (2 downto 0);
          s : out STD_LOGIC_VECTOR (7 downto 0));
end encoder;

architecture encoder_behavior of encoder is
begin
    with d select
        s <=   "00000001" when "000",
               "00000010" when "001",
               "00000100" when "010",
               "00001000" when "011",
               "00010000" when "100",
               "00100000" when "101",
               "01000000" when "110",
               "10000000" when "111",
               "00000000" when others; -- catch all
end encoder_behavior;
```

Codificador com Enable

- Entrada adicional: Enable
  - Permite habilitar/desabilitar o bloco todo

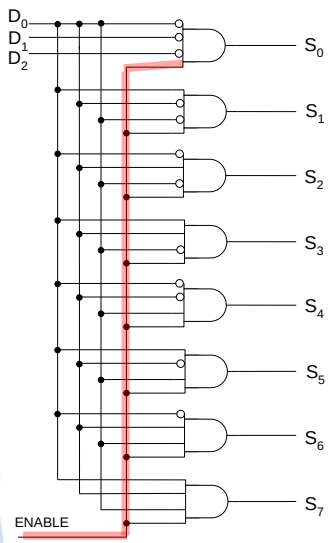
ENTRADAS				SAÍDAS							
EN	d_2	d_1	d_0	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
1	0	0	0	0	0	0	0	0	0	0	**1**
1	0	0	1	0	0	0	0	0	0	**1**	0
1	0	1	0	0	0	0	0	0	**1**	0	0
1	0	1	1	0	0	0	0	**1**	0	0	0
1	1	0	0	0	0	0	**1**	0	0	0	0
1	1	0	1	0	0	**1**	0	0	0	0	0
1	1	1	0	0	**1**	0	0	0	0	0	0
1	1	1	1	**1**	0	0	0	0	0	0	0
0	X	X	X	0	0	0	0	0	0	0	0

Codificador com Enable



Como adicionar o enable?

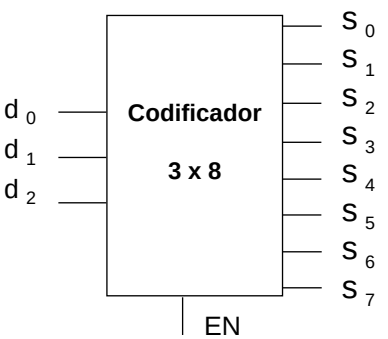
Codificador com Enable



$$S_0 = EN \cdot D_2' \cdot D_1' \cdot D_0'$$

...

$$S_7 = EN \cdot D_2 \cdot D_1 \cdot D_0$$

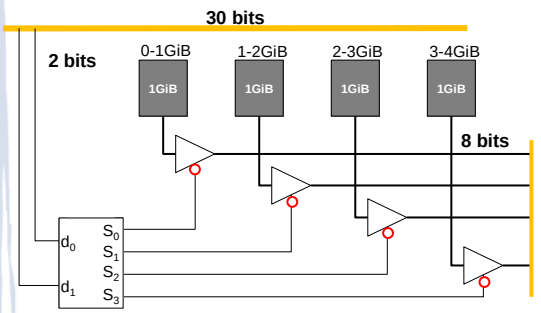


Codificadores: VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity encoderE is -- codificador 3x8 com enable
    port (d    : in STD_LOGIC_VECTOR (2 downto 0);
          en   : in STD_LOGIC;
          s    : out STD_LOGIC_VECTOR (7 downto 0));
end encoderE;

architecture encoderE_behavior of encoderE is
begin
    s <= "00000000" when (en = '0')   else   -- com enable
         "00000001" when (d = "000")  else
         "00000010" when (d = "001")  else
         "00000100" when (d = "010")  else
         "00001000" when (d = "011")  else
         "00010000" when (d = "100")  else
         "00100000" when (d = "101")  else
         "01000000" when (d = "110")  else
         "10000000" when (d = "111")  else
         "00000000"; -- catch all
end encoderE_behavior;
```

Codificador

- Problema: e se buffer tri-state for ativo em baixo?
  - 4 Chips de 1 GiB: 30 bits de endereço
  - Entrada do Codificador: endereço de 2 bits
  - Solução 1: usar inversores externos
  - Solução 2: usar um codificador ativo baixo

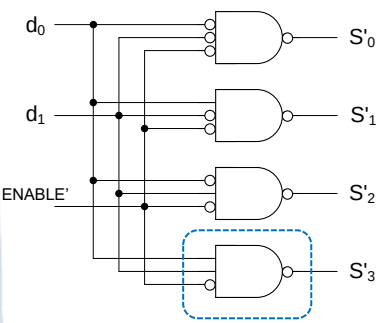


ENTRADA		SAÍDA			
d_1	d_0	S_3	S_2	S_1	S_0
0	0	1	1	1	**0**
0	1	1	1	**0**	1
1	0	1	**0**	1	1
1	1	**0**	1	1	1

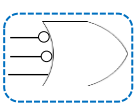
Codificador: active-low

- Codificadores podem ter as saídas invertidas (complementadas)
  - Diz-se que as saídas são *active-low* – ativas em ZERO
  - Logo, saídas inativas = 1; a única saída ativa = 0
- Implicações:
  - Na tabela verdade: inversão nas saídas
  - No circuito: uso de **NAND** no lugar de AND
  - Nomenclatura: S_i'
  - Cada saída é um **maxtermo** dos bits de entrada
- Também pode se aplicar a Enable
  - 0: habilita circuito (apenas 1 saída ativada);
  - 1: desabilita circuito (todas as saídas desativadas)

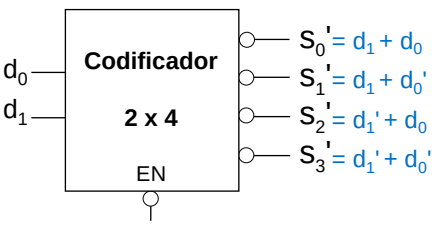
Codificador: active-low



=



$S_3' = d_1' + d_0'$



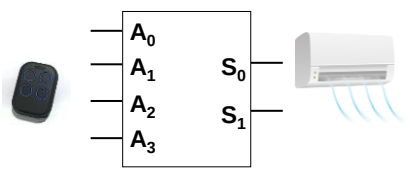
Entradas			Saídas			
EN'	d_1	d_0	S_3'	S_2'	S_1'	S_0'
0	0	0	1	1	1	**0**
0	0	1	1	1	**0**	1
0	1	0	1	**0**	1	1
0	1	1	**0**	1	1	1
1	X	X	1	1	1	1

Codificadores: VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity encoder_L is -- codificador 3x8 active low
    port (d    : in STD_LOGIC_VECTOR (2 downto 0);
          en_l : in STD_LOGIC;
          s_l  : out STD_LOGIC_VECTOR (7 downto 0));
end encoder_L;

architecture arch of encoder_L is
begin
    s_l <= "11111111" when (en_l = '1')  else   -- com enable
           "11111110" when (d = "000")   else
           "11111101" when (d = "001")   else
           "11111011" when (d = "010")   else
           "11110111" when (d = "011")   else
           "11101111" when (d = "100")   else
           "11011111" when (d = "101")   else
           "10111111" when (d = "110")   else
           "01111111" when (d = "111")   else
           "11111111"; -- catch all
end arch;
```

Codificadores

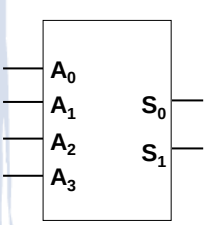
- **Problema**: economizar fios na transmissão de dados entre controle de 4 botões e ventilador controlado
  - Entrada: 4 botões, apenas 1 pressionado por vez
  - Saída: número mínimo de bits necessário para representar o botão pressionado.
    - **Pergunta**: quantos bits? **Resposta**: 2 bits ($2^2 = 4$ valores)
  - **Pergunta**: tabela verdade?



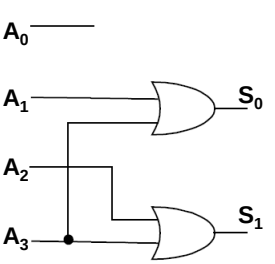
Entradas				Saídas	
A_3	A_2	A_1	A_0	S_1	S_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Codificadores: Binário "2^n : n"

- Codificador binário 4:2
- **Pergunta:** algum problema em potencial...?
  - Mais de uma entrada em 1 pode resultar em erro!
  - Ex.: 0110 → 11 (mesmo resultado obtido com 1000)
  - Como "consertar"?



Entradas				Saídas	
A_3	A_2	A_1	A_0	S_1	S_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	1	1	0	1	1



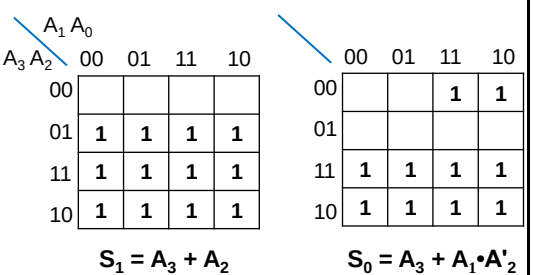
Codificadores: Binário "2^n : n"

- Codificador binário 4:2 **com prioridade**
  - Entradas têm níveis de prioridade: elimina o problema anterior (perceba os "don't care")
  - Resolução: construir o mapa de Karnaugh

Entradas | Saídas

A_3	A_2	A_1	A_0	S_1	S_0
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1
0	1	1	0	1	0

prioridade



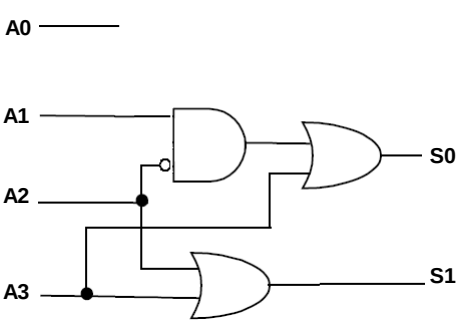
Codificadores: Binário "2^n : n"

- Codificador binário 4:2 **com prioridade**
  - Entradas têm níveis de prioridade: elimina o problema anterior (perceba os "don't care")
  - Usando o mapa de Karnaugh

Entradas | Saídas

A_3	A_2	A_1	A_0	S_1	S_0
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1
0	1	1	0	1	0

prioridade



Codificadores: VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity priority_encoder is -- Codificador 8:3 com prioridade
    port (A    : in STD_LOGIC_VECTOR (7 downto 0);
          en   : in STD_LOGIC;
          S    : out STD_LOGIC_VECTOR (2 downto 0));
end priority_encoder;

architecture behav of priority_encoder is
begin
    S <= "000" when (en   = '0')  else  -- com enable
         "111" when (A(7) = '1')  else  -- maior prioridade
         "110" when (A(6) = '1')  else
         "101" when (A(5) = '1')  else
         "100" when (A(4) = '1')  else
         "011" when (A(3) = '1')  else
         "010" when (A(2) = '1')  else
         "001" when (A(1) = '1')  else
         "000" when (A(0) = '1')  else -- menor prioridade
         "000"; -- "catch all"
end behav;
```

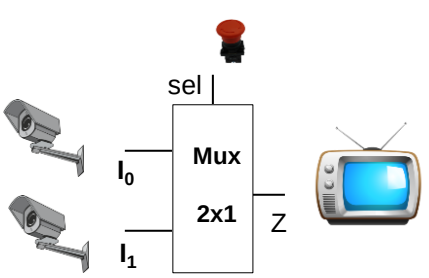

Codificadores: outros exemplos

- Reduzem o número de fios e aplicações com diversas entradas
  - Ex.: codificador de teclado (em vez de ter um fio por tecla, valor de cada tecla pode ser antes codificado)
- Regular entradas com diferentes prioridades
  - Ex.: controle de interrupções em computadores
- Conversão entre sistemas numéricos
  - Ex.: o codificador BCD (Binary Coded Decimal – Decimal Codificado em Binário) apresenta 10 entradas, resultando em 4 bits de saída em binário: A5 → 0101, A8 → 1000, etc.
- Exemplos extras nos **exercícios** ao final da aula

(DE)MULTIPLEXADORES

Multiplexadores

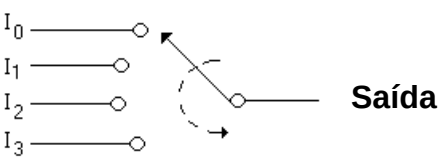
- **Problema**: sistema de vigilância com 1 TV e 2 câmeras
  - Operador escolhe qual câmera deseja ver com botão seletor
  - Transmissão serial: 1 bit por vez em cada entrada I
- **Pergunta**: qual a tabela verdade?



Entradas			Saída
Sel	I_0	I_1	Z
0	**0**	0	**0**
0	**0**	1	**0**
0	**1**	0	**1**
0	**1**	1	**1**
1	0	**0**	**0**
1	0	**1**	**1**
1	1	**0**	**0**
1	1	**1**	**1**

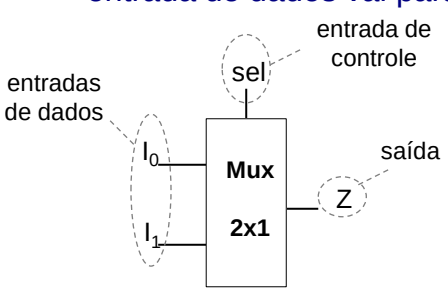
Multiplexador: Conceito

- Multiplexação: seleção da entrada que deve ir para a saída
  - Operação de uma chave seletora



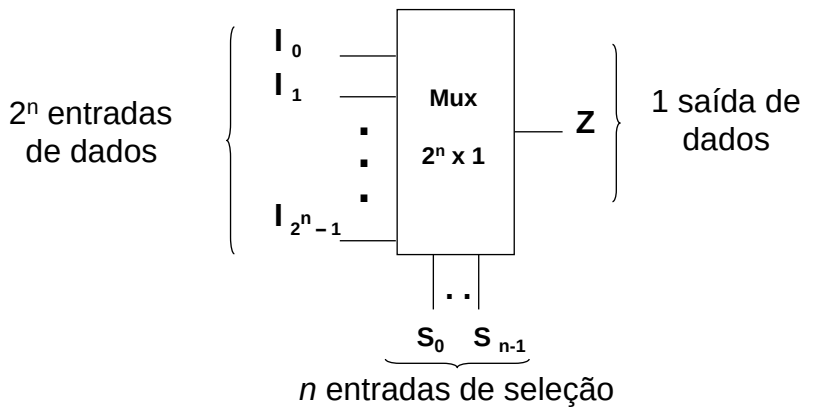
Multiplexador: Conceito

- Multiplexador ou Multiplexer ou Mux
  - Possui 2^n entradas de dados e uma saída: só uma das 2^n entradas é "conectada" à saída
  - Um total de **n entradas de seleção** indicam qual entrada de dados vai para a saída

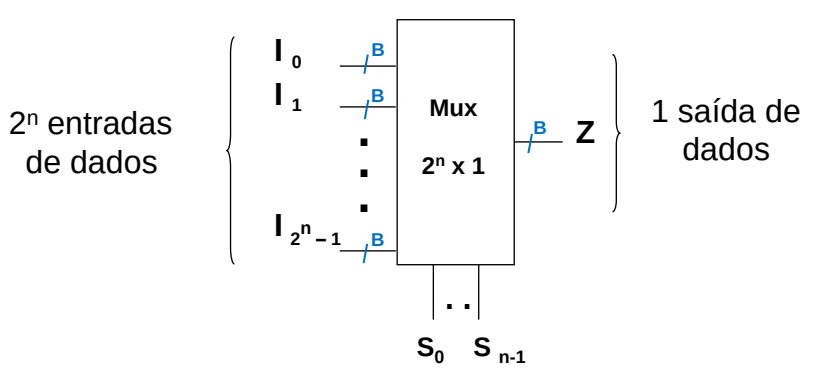


sel	I_0	I_1	Z
0	**0**	0	**0**
0	**0**	1	**0**
0	**1**	0	**1**
0	**1**	1	**1**
1	0	**0**	**0**
1	0	**1**	**1**
1	1	**0**	**0**
1	1	**1**	**1**

Multiplexador: visão geral



Multiplexador: visão geral



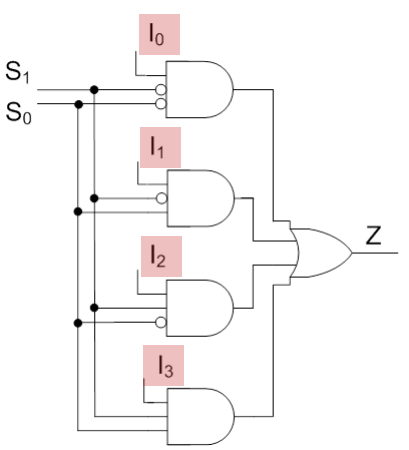
Nota: cada entrada/saída pode ser na realidade um barramento com B bits ➔ todos são selecionados ao mesmo tempo. NÃO vamos explorar este conceito nos slides a seguir

Multiplexador: visão geral

- Qual entrada de dados vai para a saída ?
  - As **entradas de seleção** formam uma palavra binária de *n* bits
    - A palavra pode assumir os valores de 0 a $2^n - 1$
    - Menor índice = bit menos significativo na palavra binária
  - As 2^n entradas de dados são numeradas de 0 a $2^n - 1$
    - A saída recebe a entrada de dados cujo índice corresponde ao valor da palavra binária das entradas de seleção
- Exemplo: Mux de 3 bits
  - Entradas de seleção $S_2S_1S_0 = 011\ (3_{10})$; Z recebe I_3
  - Entradas de seleção $S_2S_1S_0 = 110\ (6_{10})$; Z recebe I_6

Multiplexador de 2 bits: circuito interno

Entradas de Seleção		Saída
S_1	S_0	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



$$Z = \underbrace{S_1' \cdot S_0'}\cdot I_0 + \underbrace{S_1' \cdot S_0}\cdot I_1 + \underbrace{S_1 \cdot S_0'}\cdot I_2 + \underbrace{S_1 \cdot S_0}\cdot I_3$$

mintermos

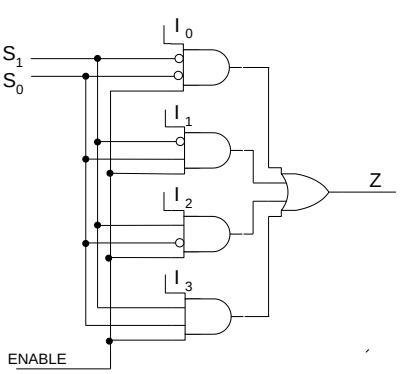
Multiplexador de 2 bits: VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux4_1 is
    port (sel          : in  STD_LOGIC_VECTOR (1 downto 0);
          I0,I1,I2,I3  : in  STD_LOGIC;
          Y            : out STD_LOGIC);
end mux4_1;

architecture arch_mux of mux4_1 is
begin
    with sel select -- Comumente traduzido para MUX!
       Y <= I0  when "00",
            I1  when "01",
            I2  when "10",
            I3  when "11",
            '0' when others; -- "catch all"
end arch_mux;
```


Multiplexador com Enable

- Entrada adicional: Enable
  - Permite habilitar/desabilitar o bloco todo

Entradas			Saída
EN	S_1	S_0	Z
1	0	0	I_0
1	0	1	I_1
1	1	0	I_2
1	1	1	I_3
0	X	X	0



$$Z = EN \cdot S_1' \cdot S_0' \cdot I_0 + EN \cdot S_1' \cdot S_0 \cdot I_1 + EN \cdot S_1 \cdot S_0' \cdot I_2 + EN \cdot S_1 \cdot S_0 \cdot I_3$$

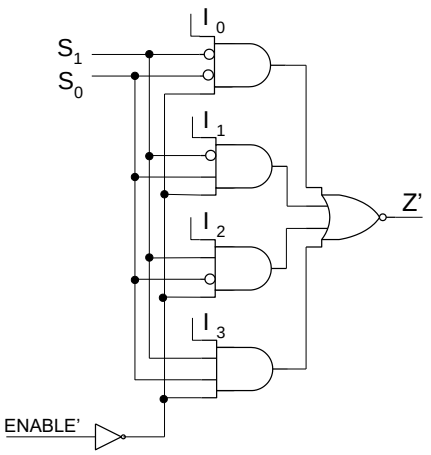
Multiplexador de 2 bits: VHDL

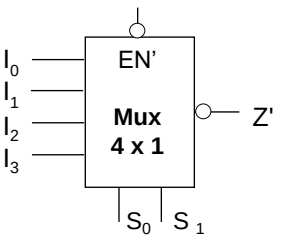
```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux4_1_en is
    port (sel          : in  STD_LOGIC_VECTOR (1 downto 0);
          I0,I1,I2,I3  : in  STD_LOGIC;
          en           : in  STD_LOGIC;
          Y            : out STD_LOGIC);
end mux4_1_en;

architecture arch_mux of mux4_1_en is
begin
    Y <= '0' when (en  = '0' ) else  -- com enable
         I0  when (sel = "00") else
         I1  when (sel = "01") else
         I2  when (sel = "10") else
         I3  when (sel = "11") else
         '0'; -- "catch all"
end arch_mux;
```


Multiplexador: *active-low*

- Multiplexadores podem ter a saída invertida (complementada)
  - Diz-se que as saídas são *active-low* – ativas em ZERO
- Implicações:
  - Na tabela verdade: inversão nas saídas (Z' = I_i')
  - No circuito: uso de **NOR** no lugar de OR
  - Nomenclatura: Z‘
- Também pode se aplicar a Enable
  - 0: habilita circuito (1 saída passada para entrada);
  - 1: desabilita circuito (todas as saídas desativadas)

Multiplexador: *active-low*

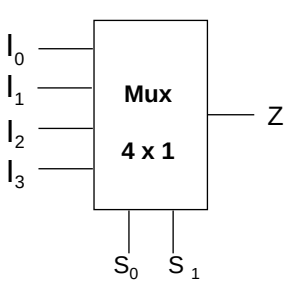


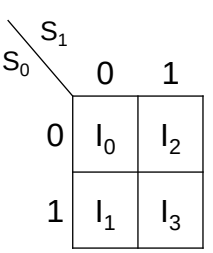


Entradas			Saída
EN'	S_1	S_0	Z'
0	0	0	I_0'
0	0	1	I_1'
0	1	0	I_2'
0	1	1	I_3'
1	X	X	1

Multiplexadores: Uso

- Síntese de funções
  - Qualquer função de chaveamento de *n* variáveis pode ser implementada com um único MUX 2^n **:** 1
  - Ex.: Função de 2 variáveis e Mux 4x1





Mapa do Mux 4*x*1

$$Z = S_1'.S_0'.I_0 + S_1'.S_0.I_1 + S_1.S_0'.I_2 + S_1.S_0.I_3$$

Multiplexadores: Síntese

Mapa do Mux 4*x*1

S_0 \ S_1	0	1
0	I_0	I_2
1	I_1	I_3

Mapa da Função F(b,a)

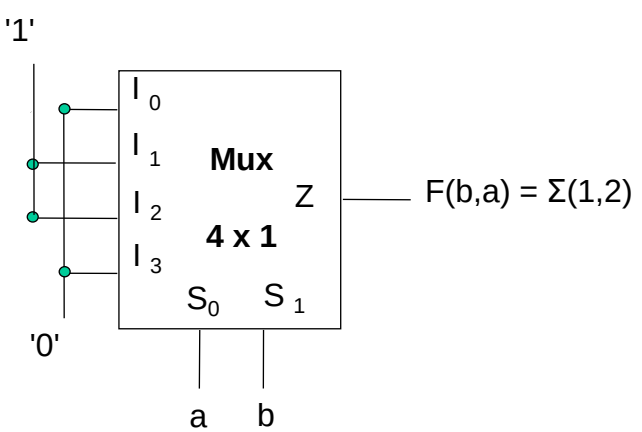
a \ b	0	1
0	0	1
1	1	0

F(b,a) = Σ(1,2)

- Da comparação dos dois mapas obtém-se:

  $I_0 = 0$, $I_1 = 1$, $I_2 = 1$, $I_3 = 0$

Multiplexadores: Síntese

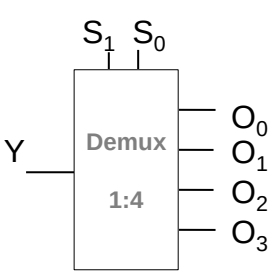


Multiplexadores: Síntese

- FAQ: "Mas isso é usado na prática?!"
- Resposta: Sim... E muito!
  - Esta é a base para a construção de circuitos de lógica programável (programmable logic device -- PLD)
  - Os conceitos envolvidos são semelhantes aos que aparecem em dispositivos modernos programados usando HDLs, como FPGAs
  - Logo, em projetos com HDL você provavelmente não verá os "mintermos e MUXes", mas esse é o tipo de tarefa que será realizada para você pelo sintetizador...

Demultiplexadores

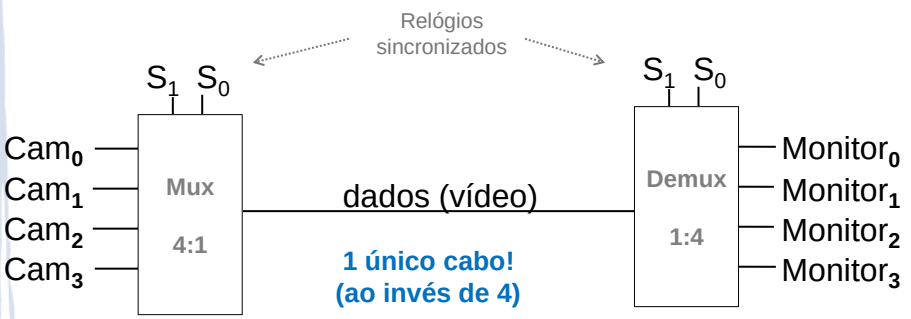
- Função inversa ao multiplexador:
  - Demux 1:2^n direciona a entrada para uma dentre 2^n saídas de dados



S_1	S_0	O_1	O_0	O_2	O_3
0	0	**Y**	0	0	0
0	1	0	**Y**	0	0
1	0	0	0	**Y**	0
1	1	0	0	0	**Y**

(De)multiplexadores: Uso

- Pode ser usado para compartilhamento de canal entre diversos fluxos (multiplexação de vídeo)
  - Ex.: envio serial de vídeo de diferentes câmeras para uma central de vigilância



Demultiplexador: VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity demux4_1 is
    port (S            : in STD_LOGIC_VECTOR (2 downto 0)
          Y            : in STD_LOGIC_VECTOR (7 downto 0);
          O0,O1,O2,O3  : out STD_LOGIC_VECTOR (7 downto 0));
end demux4_1;

architecture arch_demuxZ of demux4_1 is -- com alta impedância
begin
    O0 <= Y when S = "00" else "ZZZZZZZZ"; -- código alternativo:
    O1 <= Y when S = "01" else "ZZZZZZZZ"; -- else (others => 'Z');
    O2 <= Y when S = "10" else "ZZZZZZZZ";
    O3 <= Y when S = "11" else "ZZZZZZZZ";
end arch_demuxZ;

architecture arch_demux0 of demux4_1 is -- desabilitado com 0s
begin
    O0 <= Y when S = "00" else "00000000"; -- código alternativo:
    O1 <= Y when S = "01" else "00000000"; -- else (others => '0');
    O2 <= Y when S = "10" else "00000000";
    O3 <= Y when S = "11" else "00000000";
end arch_demux0;
```

EXERCÍCIOS

Codificador decimal

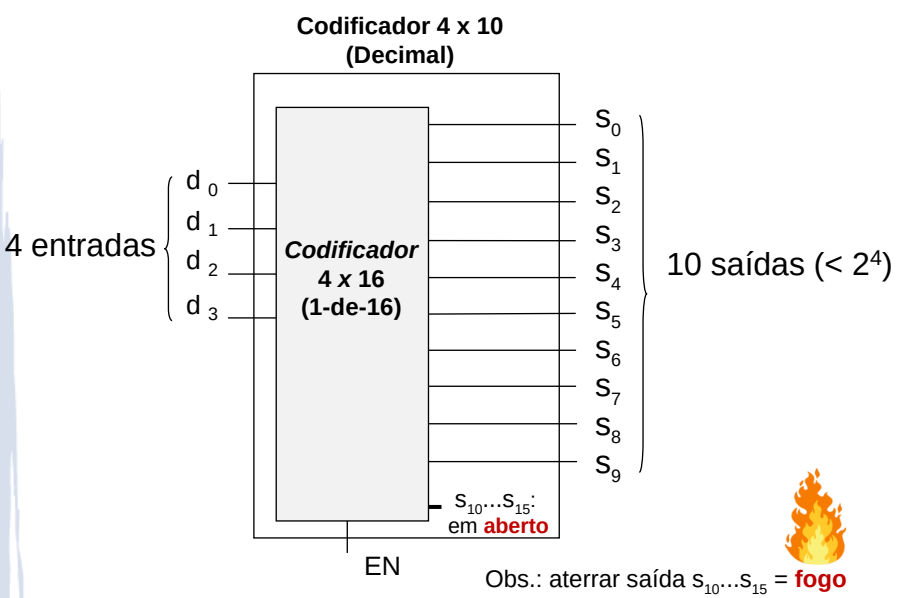
- Projete um codificador BCD para decimal usando codificador 1-de-m de tamanho adequado
  - Entrada: 1 dígito BCD (4 bits)
  - Saída: 10 bits ($<2^4$)
    - Saída ativa: aquele correspondente à entrada, ou nenhuma se entrada inválida (> 1001) ou se Enable=0

Dígito decimal	BCD	Dígito decimal	BCD
0	0000	8	1000
1	0001	9	1001
2	0010	inválido	1010
3	0011		1011
4	0100		1100
5	0101		1101
6	0110		1110
7	0111		1111

Codificador decimal: resposta

EN	d_3	d_2	d_1	d_0	S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
1	0	0	0	1	0	0	0	0	0	0	0	0	**1**	0
1	0	0	1	0	0	0	0	0	0	0	0	**1**	0	0
1	0	0	1	1	0	0	0	0	0	0	**1**	0	0	0
1	0	1	0	0	0	0	0	0	0	**1**	0	0	0	0
1	0	1	0	1	0	0	0	0	**1**	0	0	0	0	0
1	0	1	1	0	0	0	0	**1**	0	0	0	0	0	0
1	0	1	1	1	0	0	**1**	0	0	0	0	0	0	0
1	1	0	0	0	0	**1**	0	0	0	0	0	0	0	0
1	1	0	0	1	**1**	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0
	.	.	.	.										
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
0	X	X	X	X	0	0	0	0	0	0	0	0	0	0

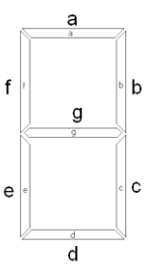
Codificador decimal : resposta

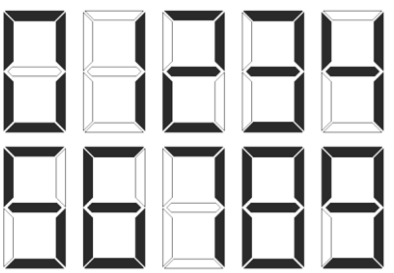


Obs.: aterrar saída $s_{10}...s_{15}$ = **fogo**

Codificador BCD p/ 7 segmentos

- Projetar codificador que aciona *display* de 7 segmentos a partir de código BCD
  - Entrada: 4 dígitos BCD (0000 a 1001)
  - Saídas: 7 bits, código 7 segmentos
- Forneça: (1) tabela verdade e (2) código VHDL





Codificador BCD p/ 7 segmentos

Tabela verdade

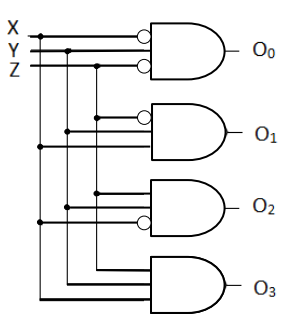
Decimal	EN	d_3	d_2	d_1	d_0	a	b	c	d	e	f	g
0	1	0	0	0	0	1	1	1	1	1	1	
1	1	0	0	0	1		1	1				
2	1	0	0	1	0	1	1		1	1		1
3	1	0	0	1	1	1	1	1	1			1
4	1	0	1	0	0		1	1			1	1
5	1	0	1	0	1	1		1	1		1	1
6	1	0	1	1	0	1		1	1	1	1	1
7	1	0	1	1	1	1	1	1				
8	1	1	0	0	0	1	1	1	1	1	1	1
9	1	1	0	0	1	1	1	1	1		1	1
10	1	1	0	1	0	0	0	0	0	0	0	0
		.	.	.	.							
15	1	1	1	1	1	0	0	0	0	0	0	0
	0	X	X	X	X	0	0	0	0	0	0	0

Codificador BCD p/ 7 segmentos

```
library IEEE;
use IEEE.std_logic_1164.all;

entity enc_BCD_7seg is -- codificador BCD p/ 7 segmentos
    port (d     : in STD_LOGIC_VECTOR (3 downto 0);
          en    : in STD_LOGIC;
          s     : out STD_LOGIC_VECTOR (6 downto 0)); -- abcdefg
end enc_BCD_7seg;

architecture arch of enc_BCD_7seg is
begin
   s <= "0000000" when (en = '0') else
        "1111110" when (d = "0000")   else
        "0110000" when (d = "0001")   else
        "1101101" when (d = "0010")   else
        "1111001" when (d = "0011")   else
        "0110011" when (d = "0100")   else
        "1011011" when (d = "0101")   else
        "1011111" when (d = "0110")   else
        "1110000" when (d = "0111")   else
        "1111111" when (d = "1000")   else
        "1111011" when (d = "1001")   else
        "0000000"; -- catch all
end arch;
```


P2-2018a

- O circuito abaixo é um codicador binário 2x4.
  - Uma das 3 entradas (X, Y, Z), é a habilitação (En).
  - As outras (I_0, I_1), determinam o índice k = (I_1I_0) da única saída ativa O_k quando o circuito está habilitado.



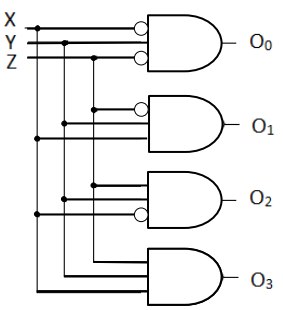
Preencher:

X	Y	Z	O_0	O_1	O_2	O_3
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

Identificar En, I_1 e I_0:

P2-2018a

- O circuito abaixo é um codicador binário 2x4.
  - Uma das 3 entradas (X, Y, Z), é a habilitação (En).
  - As outras (I_0, I_1), determinam o índice k = (I_1I_0) da única saída ativa O_k quando o circuito está habilitado.



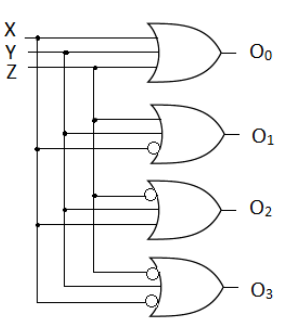
Preencher:

X	Y	Z	O_0	O_1	O_2	O_3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	1	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	1	0	0
1	1	1	0	0	0	1

Identificar En, I_1 e I_0: I_0 En I_1

P2-2018b

- O circuito abaixo é um decodicador binário 2x4.
  - Uma das 3 entradas (X, Y, Z), é a habilitação.
  - As outras (I_0, I_1), determinam o índice k = (I_1I_0) da única saída ativa O_k quando o circuito está habilitado.

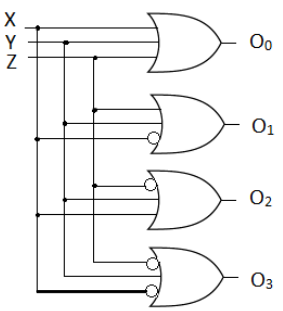


Preencher:

X	Y	Z	O_0	O_1	O_2	O_3
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

P2-2018b

- O circuito abaixo é um decodicador binário 2x4.
  - Uma das 3 entradas (X, Y, Z), é a habilitação.
  - As outras (I_0, I_1), determinam o índice k = ($I_1 I_0$) da única saída ativa O_k quando o circuito está habilitado.



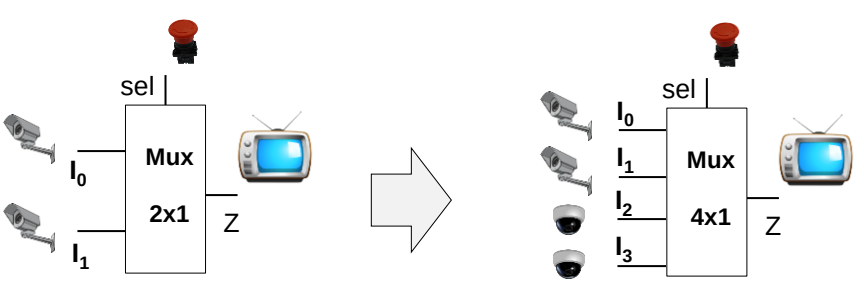
Preencher:

X	Y	Z	O_0	O_1	O_2	O_3
0	0	0	0	1	1	1
0	0	1	1	1	0	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	0	1	1
1	0	1	1	1	1	0
1	1	0	1	1	1	1
1	1	1	1	1	1	1

Identificar En, I_1 e I_0: I_0 En I_1

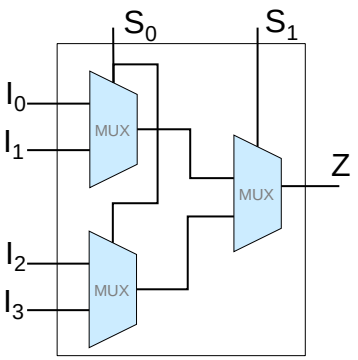
Multiplexadores - Cascateamento

- **Problema**: expansão do sistema de vigilância com 1 TV e 2 câmeras, para suportar 4 câmeras
  - Porém, só há MUXes 2:1 disponíveis (peças de reposição)...
- **Pede-se**: construir um MUX 4:1 usando MUXes 2:1 na quantidade que julgar necessária



Multiplexadores - Cascateamento

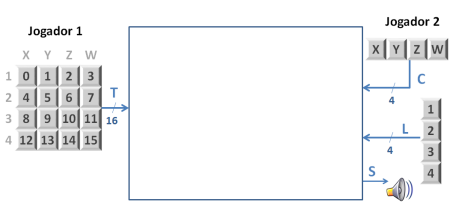
- **Problema**: expansão do sistema de vigilância com 1 TV e 2 câmeras, para suportar 4 câmeras
  - Porém, só há MUXes 2:1 disponíveis (peças de reposição)...
- **Pede-se**: construir um MUX 4:1 usando MUXes 2:1



S_1	S_0	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

PREC 2018

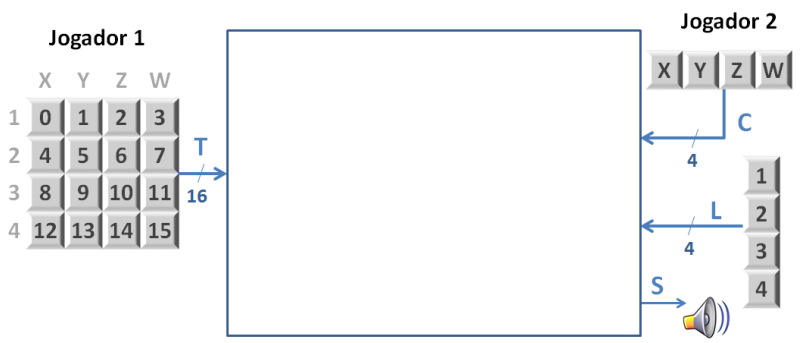
- Cenário: jogo de batalha naval
  - Jogador 1: coloca submarinos no tabuleiro (T), composto por 16 botões pressionáveis do tipo toggle (ao pressionar botão, ele muda de 0 para 1 e vice-versa): posições pressionadas recebem o valor 1, indicando a presença de um submarino.
    - Ex.: posições 0, 1, 2 e 3 ➔ T = 1111000000000000;
      posições 3, 7, 11 e 15 ➔ T = 0001000100010001
  - Jogador 2: bombardeia usando teclas de linha (L) e coluna (C).
    - Ex.: para posição 0, pressionar 1 e X ➔ L = 1000, C = 1000
      para posição 13, pressionar 4 e Y ➔ L = 0001, C = 0100



PREC 2018

Circuito a ser projetado: entradas T (16 bits), L (4 bits) e C (4 bits), e saída S (alto-falante): S=0 ➔ “chuá” (erro); S=1 ➔ “bum” (acerto)

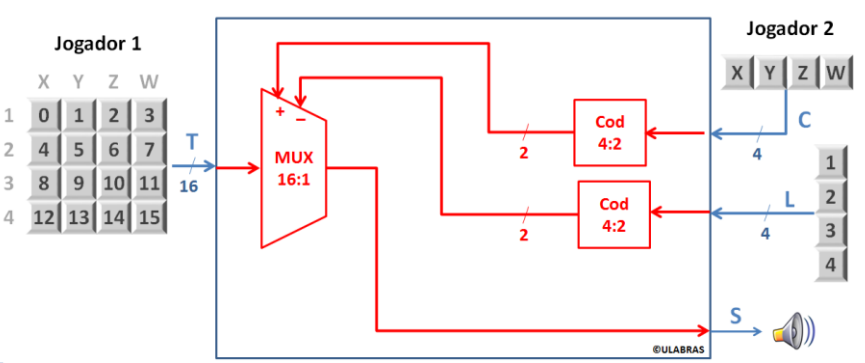
a) Projete o circuito usando apenas MUXes, DeMUXes e Codificadores/ Decodificadores binários de 2 a 4 bits cada. Assuma que L e C têm sempre um único bit 1 na entrada



PREC 2018 - RESPOSTA

Circuito a ser projetado: entradas T (16 bits), L (4 bits) e C (4 bits), e saída S (alto-falante): S=0 ➔ “chuá” (erro); S=1 ➔ “bum” (acerto)

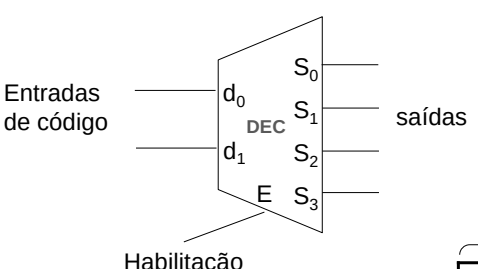
a) Projete o circuito usando apenas MUXes, DeMUXes e Codificadores/ Decodificadores binários de 2 a 4 bits cada. Assuma que L e C têm sempre um único bit 1 na entrada



APÊNDICE

Decodificadores: detalhes extras

Codificadores: Símbolo alternativo



Entradas			Saídas			
E	**d_1**	**d_0**	**S_3**	**S_2**	**S_1**	**S_0**
0	X	X	0	0	0	0
1	0	0	0	0	0	**1**
1	0	1	0	0	**1**	0
1	1	0	0	**1**	0	0
1	1	1	**1**	0	0	0

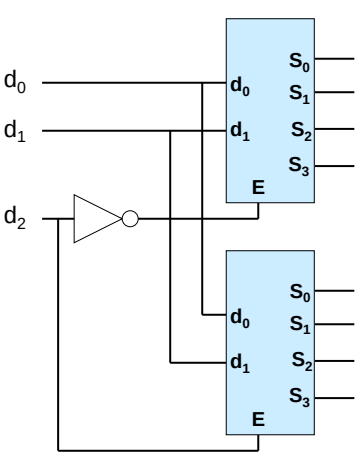
Tabela verdade

Codificadores: cascateamento

- Objetivo: expansão de capacidade:
  - Decodificadores 2x4 ➔ 3x8 ➔ 4x16 ➔ 5x32 ➔ n x 2^n
- Estratégia: combinar decodificadores menores, usando entrada de habilitação
- **Exercício**: montar decodificador 3x8 sem entrada de habilitação a partir de dois decodificadores 2x4 com entrada de habilitação

Codificadores: cascateamento

- Codificador 3x8, sem entrada de habilitação



Exercício: como adicionar uma entrada de habilitação neste circuito?

➔ Adicionar um AND na entrada de habilitação de cada decodificador, fazendo $E \cdot d_2'$ no de cima e $E \cdot d_2$ no de baixo

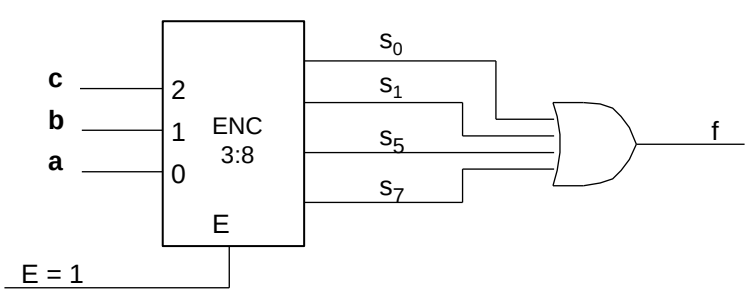
Codificadores: uso (síntese)

- Síntese de funções de chaveamento
  - Saídas dos codificadores são os mintermos para as variáveis de entrada
  - 1ª forma canônica: Função = $\Sigma_{mintermos}$
- **Exemplo**:
  - F(d,c,b,a) = Σ(0,1,5,9,12)

Codificadores: uso (síntese)

- **Exemplo: ativo-alto ➔ soma de mintermos**

  F(c,b,a) = Σ(0,1,5,7)

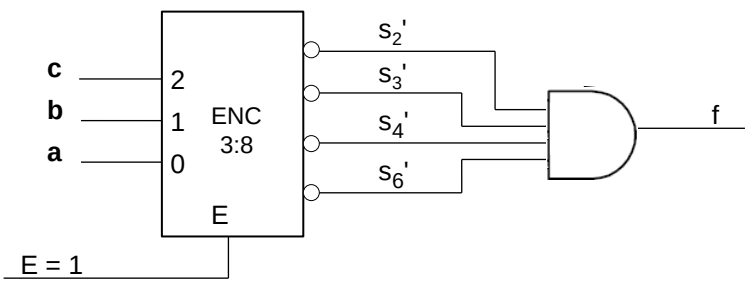


Codificadores: uso (síntese)

- **Exemplo: ativo-baixo ➔ produto de maxtermos**

  F(c,b,a) = Σ(0,1,5,7)

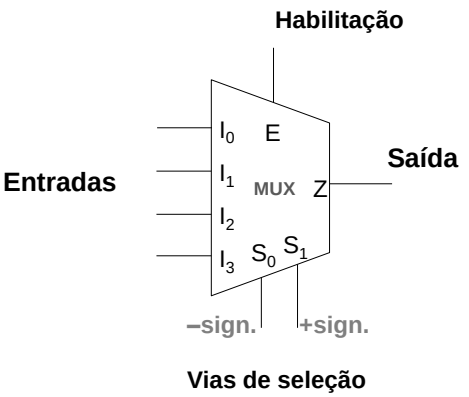
  = ∏(2,3,4,6)



APÊNDICE

Multiplexadores: detalhes extras

Multiplexador: Símbolo alternativo



Multiplexadores: VHDL (Wakerly)

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux4in8b is
    port (
        S: in STD_LOGIC_VECTOR (1 downto 0);      -- Select inputs, 0-3 ==> A-D
        A, B, C, D: in STD_LOGIC_VECTOR (1 to 8); -- Data bus input
        Y: out STD_LOGIC_VECTOR (1 to 8)          -- Data bus output
    );
end mux4in8b;

architecture mux4in8b of mux4in8b is
begin
    with S select Y <=
      A when "00",
      B when "01",
      C when "10",
      D when "11",
      (others => 'U') when others; -- this creates an 8-bit vector of 'U'
end mux4in8b;
```

Obs.: No código, entradas/saídas são barramentos de 8 bits

Multiplexadores: Síntese

- Síntese de funções com Mux de **menor ordem**
  - Implementar função de chaveamento de *n* variáveis com um único MUX $2^{n-1} : 1$
  - Exemplo:
    - Função de 4 variáveis
    - Mux 8x1 (3 variáveis de seleção)

Multiplexadores: Síntese

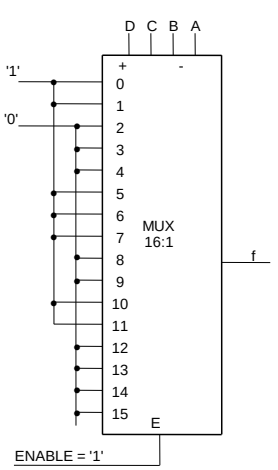
- Solução simples: com Mux de 16x1
  - Capacidade é suficiente!

BA \ DC	00	01	11	10
00	I_0	I_4	I_{12}	I_8
01	I_1	I_5	I_{13}	I_9
11	I_3	I_7	I_{15}	I_{11}
10	I_2	I_6	I_{14}	I_{10}

MAPA DE MUX 16:1

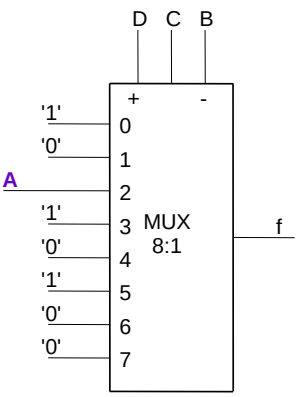
BA \ DC	00	01	11	10
00	1			
01	1	1		
11		1		1
10		1		1

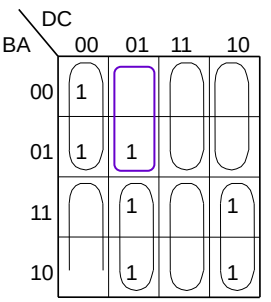
$F = \Sigma(0,1,5,6,7,10,11)$



Multiplexadores: Síntese

- Solução 1: Mux de 8x1 e **A na entrada**





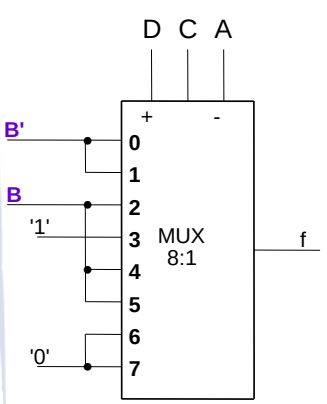
1. Cubos que podem conter A (i.e., A não tem valor fixo no cubo)

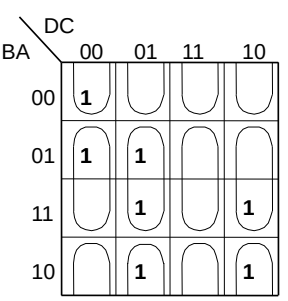
2. Função f: D'C'B' + D'CB'**A** + D'CB + DC'B

0 2 3 5

Multiplexadores: Síntese

- Solução 2: Mux de 8x1 e **B na entrada**





1. Cubos que podem conter B

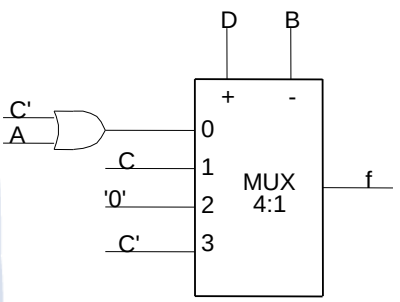
0 1 2

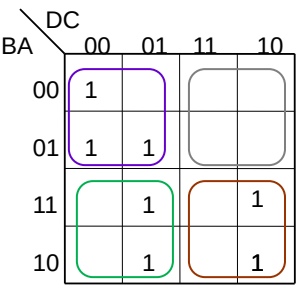
2. Função f = D'C'A'**B'** + D'C'A**B'** + D'CA'**B** + D'CA + DC'A'**B** + DC'A**B**

3 4 5

Multiplexadores: Síntese

- Solução 3: Mux de 4x1 e **A e C na entrada**





1. Cubos que podem conter A e C

2. Função f = **D'B'**(A+C') + **D'B**C + **DB**C'

Exercícios

- 6.1. Sintetize a função de chaveamento abaixo utilizando

  $F(d,c,b,a) = \Sigma(0,1,3,8,10)$

  a. Um Mux de 16x1
  b. Um Mux de 8x1
  c. Um Mux de 4x1

- 6.2. Sintetize a função de chaveamento dada pela tabela verdade ao lado utilizando um decodificador binário 3:8.

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