Circuitos combinatórios

João Canas Ferreira

Outubro de 2017

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

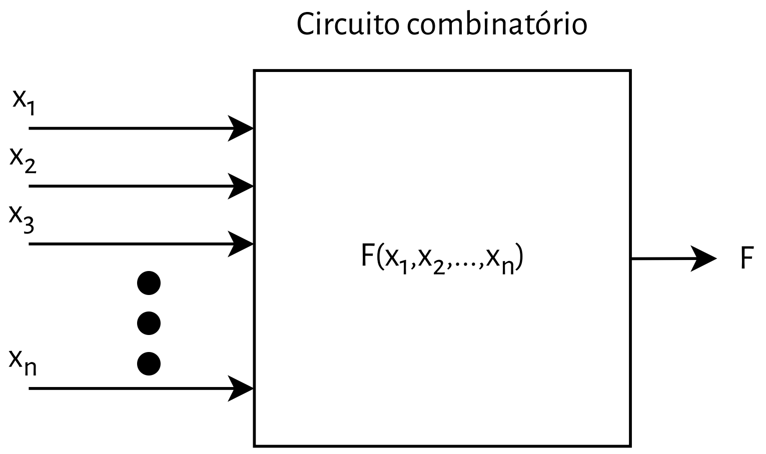
Tópicos

1. Álgebra de Boole
   - Representação abstrata do processamento binário
   - Especificação algébrica
   - Representações canónicas
2. Portas lógicas
   - Portas elementares
   - Descrição hierárquica de circuitos
3. Circuitos padrão
   - Multiplexadores
   - Descodificadores
   - Codificadores

1. Álgebra de Boole
   - Representação abstrata do processamento binário
   - Especificação algébrica
   - Representações canónicas
2. Portas lógicas
   - Portas elementares
   - Descrição hierárquica de circuitos
3. Circuitos padrão
   - Multiplexadores
   - Descodificadores
   - Codificadores

Tratamento de informação binária

➠ Como definir e representar o tratamento de informação binária?



➠ Modelo concetual mais simples: "caixa negra" que tem *n* entradas e 1 saída. O valor binário da saída depende da **combinação** de valores binários presentes nas entradas.

➠ Como definir a função *F* das *n* entradas binárias?

Definição exaustiva

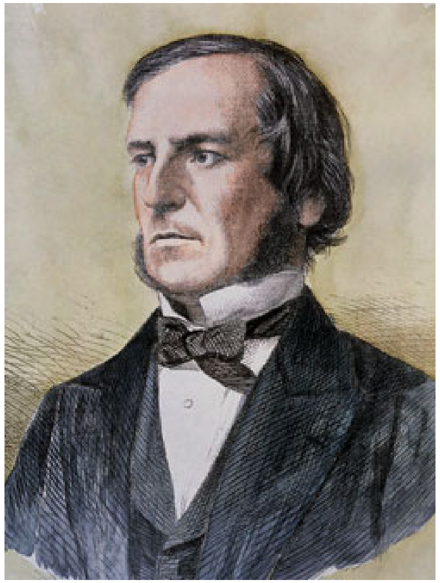
- Como as combinações de valores de entrada são finitas, podemos fazer uma lista (tabela) exaustiva do valor de saída correspondente a cada uma. Exemplo:

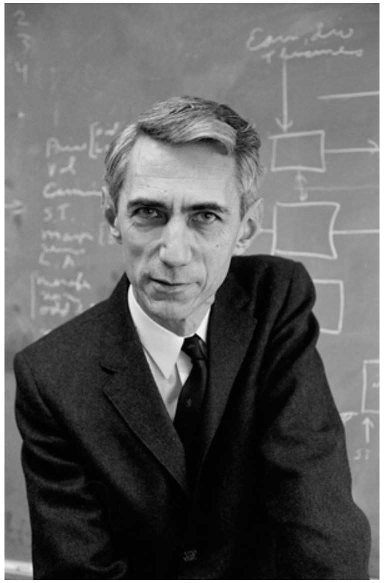
x_2	x_1	x_0	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

- Tabela de verdade (considerando 1 como verdadeiro e 0 como falso)
- Para uma função de *n* variáveis binárias, quantas linhas tem a tabela?
- Como calcular a **composição de funções**? (Quando um valor de entrada é, por sua vez, função de outros valores?)

Expressão algébrica

- A especificação e composição das funções de variáveis binárias pode ser simplificada com a introdução de expressões algébricas.
- Em 1938, Claude Shannon propôs a utilização de um método de cálculo inventado no século XIX (1854) pelo Reverendo George Boole para expressar as "leis do raciocínio".
  Nota: Esse método é equivalente ao cálculo da lógica proposicional.
- A formalização das operações associadas designa-se por **Álgebra de Boole** e pode ser feita sem requerer uma interpretação como "leis do raciocínio".


George Boole

Claude Shannon


Axiomas de Álgebra de Boole

- Uma **álgebra de Boole** é constituída por um conjunto A, dotado de duas operações binárias $+$ e $\bullet$, uma operação unária $\overline{}$ (complemento) e tendo (pelo menos) dois elementos distintos 0 e 1.

- Para quaisquer $x, y, z \in A$, valem os seguintes axiomas (Huntington, 1904)

$x + 0 = x$	$x \cdot 1 = x$	(identidade)
$x + y = y + x$	$x \cdot y = y \cdot x$	(comutatividade)
$x + (y \cdot z) = (x + y) \cdot (x + z)$	$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$	(distributividade)
$x + \overline{x} = 1$	$x \cdot \overline{x} = 0$	(complemento)

- Notações alternativas: $\vee$ para $+$ $\wedge$ para $\bullet$ $\neg x$ para $\overline{x}$.

- **Princípio da dualidade:** A uma igualdade verdadeira corresponde outra igualdade verdadeira obtida pelas trocas seguintes:

$$+ \leftrightarrow \bullet \qquad\qquad \mathbf{0} \leftrightarrow \mathbf{1}$$

(Porquê?)

- Para circuito digitais:
  álgebra de Boole com $A = \{0, 1\}$ (apenas dois elementos: 0 e 1)

Alguns teoremas úteis

- Precedência decrescente: negação, e-lógico (●), ou-lógico (+).

1 variável	
$x + x = x$	$x \cdot x = x$
$x + 1 = 1$	$x \cdot 0 = 0$
$\overline{\overline{x}} = x$	
2 variáveis	
$x + x \cdot y = x$	$x \cdot (x + y) = x$
$x + \overline{x} \cdot y = x + y$	$x \cdot (\overline{x} + y) = x \cdot y$
$(x + y) + (\overline{x} \cdot \overline{y}) = 1$	$(x \cdot y) \cdot (\overline{x} + \overline{y}) = 0$
$(x + y) \cdot (\overline{x} \cdot \overline{y}) = 0$	$(x \cdot y) + (\overline{x} + \overline{y}) = 1$
$\overline{x + y} = \overline{x} \cdot \overline{y}$	$\overline{x \cdot y} = \overline{x} + \overline{y}$
3 variáveis	
$x \cdot (y \cdot z) = (x \cdot y) \cdot z$	$x + (y + z) = (x + y) + z$
$x \cdot y + \overline{x} \cdot z + y \cdot z = x \cdot y + \overline{x} \cdot z$	$(x + y) \cdot (\overline{x} + z) \cdot (y + z) = (x + y) \cdot (\overline{x} + z)$

leis de de Morgan (não é gralha!)

- Para a álgebra de Boole (de 2 elementos), os teoremas podem ser demonstrados construindo as tabelas de verdade das expressões de ambos os lados da igualdade e confirmando que as colunas dos resultados são iguais.

Simplificação de expressões

- Axiomas e teoremas podem ser usados na simplificação de expressões.

Exemplo (por convenção, pode omitir-se o operador •):

$$
\begin{aligned}
\overline{AB}\,(\overline{A}+B)(\overline{B}+B) &= \overline{AB}\,(\overline{A}+B) & \\
&= (\overline{A}+\overline{B})\,(\overline{A}+B) & \text{lei de de Morgan} \\
&= \overline{A}+\overline{B}B & \text{distributividade} \\
&= \overline{A} &
\end{aligned}
$$

- Alternativas:
  - mapas de Karnaugh (método gráfico; até 6 variáveis)
  - método Quine-McClusky (Karma3 em `http://bit.ly/boolmin`)
    expressões mínimas de dois níveis (pode demorar muito tempo)
  - métodos heurísticos (Logic Friday em `http://www.sontrak.com/`)
    programa original: Espresso
    (`http://bit.ly/espresso-sources`)

Teorema de expansão de Boole (Shannon)

- Para qualquer função booleana vale sempre:

$$F(x_1, x_2, \ldots, x_n) = x_1 \cdot F(1, x_2, \ldots, x_n) + \overline{x_1} \cdot F(0, x_2, \ldots, x_n)$$

- A aplicação repetida da expansão permite escrever qualquer função na **forma canónica disjuntiva** (soma de produtos).

Exemplo para duas variáveis:

$$\begin{aligned} F(x_1, x_2) &= x_1 \cdot F(1, x_2) + \overline{x_1} \cdot F(0, x_2) \\ &= x_1 x_2 \cdot F(1,1) + x_1 \overline{x_2} \cdot F(1,0) + \overline{x_1} x_2 \cdot F(0,1) + \overline{x_1}\,\overline{x_2} \cdot F(0,0) \end{aligned}$$

A expressão corresponde à tabela de verdade:

x_1	x_2	$F(x_1, x_2)$
0	0	$F(0,0)$
0	1	$F(0,1)$
1	0	$F(1,0)$
1	1	$F(1,1)$

Forma canónica disjuntiva

- Uma função booleana de *n* variáveis pode ser expressa por uma soma de produtos (termos), em que cada produto inclui **uma só vez cada uma das variáveis ou o seu complemento**.

x_2	x_1	x_0	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$$F(x_2, x_1, x_0) = (\overline{x_2}\,\overline{x_1}\,\overline{x_0}) + (\overline{x_2}x_1x_0) + (x_2\overline{x_1}\,\overline{x_0}) + (x_2x_1\overline{x_0})$$

- Cada um destes termos designa-se por **termo mínimo** ou *minterm*.
- Existe uma correspondência direta entre a forma canónica disjuntiva de uma função booleana e a sua tabela de verdade (considerando as variáveis pela mesma ordem).

Somas de produtos mínimas

- A forma canónica disjuntiva mostra que é possível representar qualquer função booleana com **expressões de dois níveis**.
- A forma canónica disjuntiva é uma soma de produtos (SOP), mas não é, geralmente, a expressão desse tipo com o menor número de termos ou os termos mais simples (com menos variáveis).
- Mapas de Karnaugh ou o método de Quine-McCluskey permitem obter SOPs mínimas de forma sistemática.
- Também se pode usar simplificação algébrica.

$$
\begin{aligned}
F(x_2, x_1, x_0) &= (\overline{x_2}\,\overline{x_1}\,\overline{x_0}) + (\overline{x_2}x_1x_0) + (x_2\overline{x_1}\,\overline{x_0}) + (x_2x_1\overline{x_0}) \\
&= (\overline{x_2}\,\overline{x_1}\,\overline{x_0}) + (\overline{x_2}x_1x_0) + x_2\overline{x_0}(\overline{x_1} + x_1) \\
&= (\overline{x_2}\,\overline{x_1}\,\overline{x_0}) + (\overline{x_2}x_1x_0) + x_2\overline{x_0}
\end{aligned}
$$

Forma canónica conjuntiva

- Devido ao princípio da dualidade, uma função booleana de *n* variáveis também pode ser expressa por um produto de somas (termos), em que cada soma inclui **uma só vez cada uma das variáveis ou o seu complemento**.

x_2	x_1	x_0	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

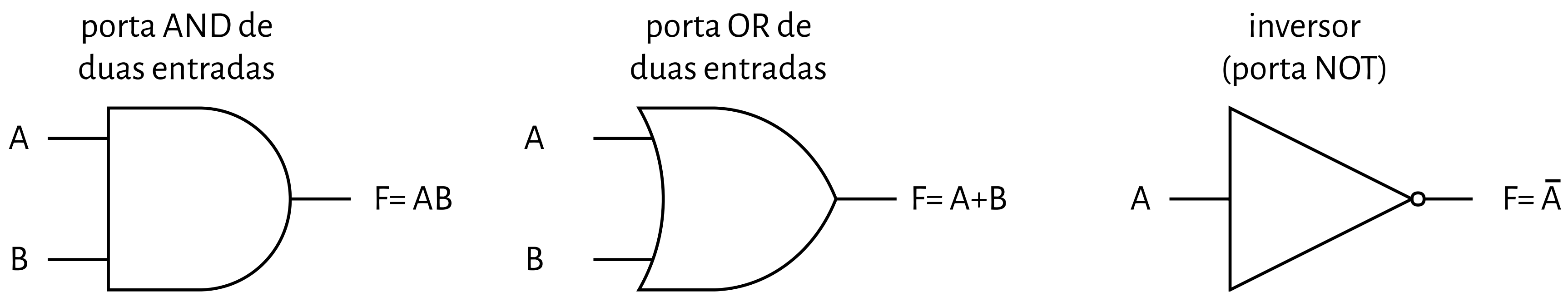
$$\begin{aligned} F(x_2, x_1, x_0) &= (x_2 + x_1 + \overline{x_0}) \cdot (x_2 + \overline{x_1} + x_0) \\ &\quad \cdot (\overline{x_2} + x_1 + \overline{x_0}) \cdot (\overline{x_2} + \overline{x_1} + \overline{x_0}) \end{aligned}$$

- Cada um destes termos designa-se por **termo máximo** ou *maxterm*.
- Fixada a ordem das variáveis, existe uma correspondência direta entre a forma canónica conjuntiva de uma função booleana e a sua tabela de verdade.
- A forma canónica conjuntiva é um **produto de somas** (POS), mas não é, geralmente, a expressão mais simples desse tipo.

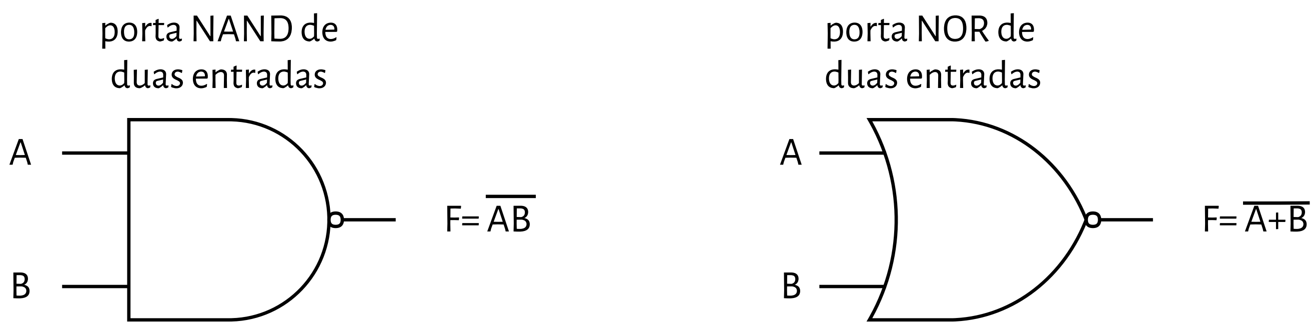
Elementos para processamento lógico de informação

➠ Para realizar fisicamente o processamento da informação, são usados circuitos eletrónicos que realizam as funções lógicas elementares: **portas lógicas**.

➠ Quando não interessam os detalhes de implementação, usam-se símbolos para representar cada porta lógica.



➠ Também existem portas lógicas que combinam a negação com outras operações lógicas.

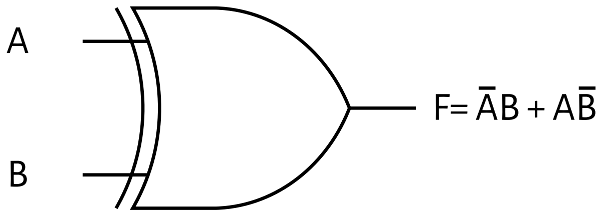


➠ Existem versões destas portas com mais entradas (3, 4, ...) [exceto inversor].

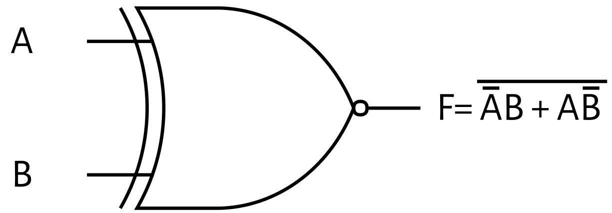
As portas lógicas XOR e XNOR

- OU-exclusivo: $F = A \oplus B = \overline{A}B + A\overline{B}$
  OU-exclusivo negado: $F = A \odot B = AB + \overline{A}\,\overline{B}$

porta XOR de
duas entradas



porta XNOR de
duas entradas



A	B	$A \oplus B$	$A \odot B$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

- OU-exclusivo é igual a 1 quando as entradas são diferentes.

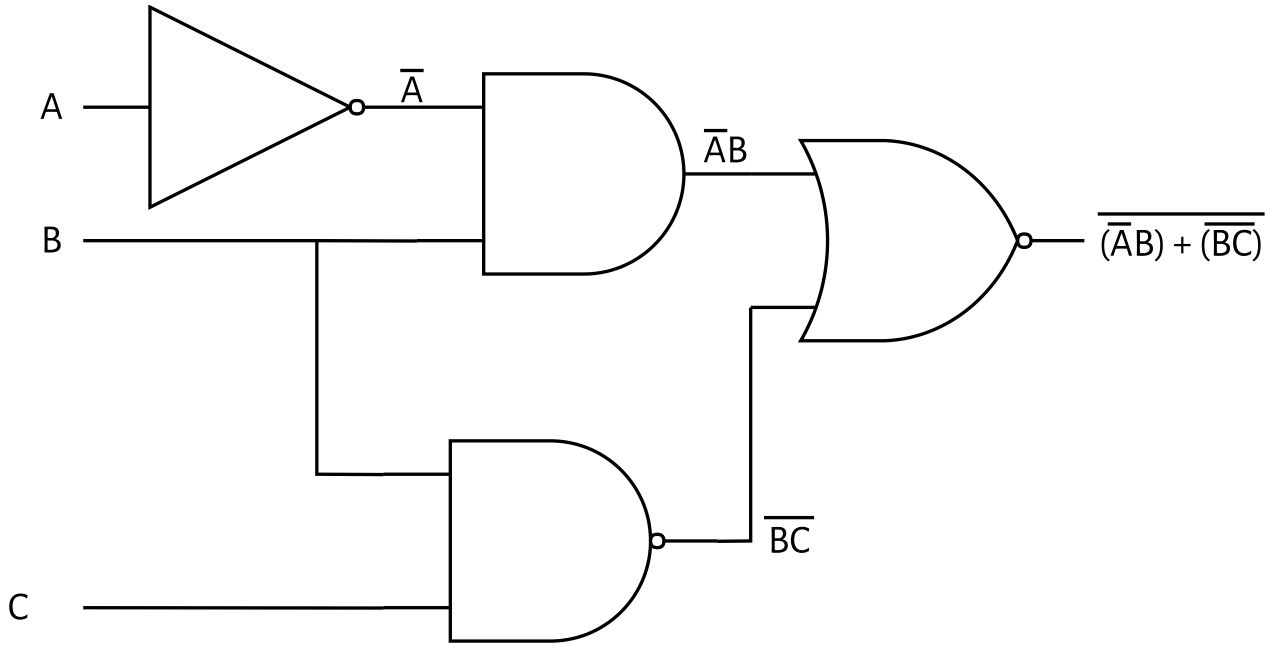
- OU-exclusivo negado é igual a 1 quando as entradas são iguais.

- $A \oplus (B \oplus C) = (A \oplus B) \oplus C \qquad A \odot (B \odot C) = (A \odot B) \odot C$
- $A \oplus 0 = A \qquad A \oplus 1 = \overline{A} \qquad A \oplus A = 0 \qquad A \oplus \overline{A} = 1$
- $A \oplus \overline{B} = \overline{A} \oplus B = \overline{(A \oplus B)} = A \odot B$

Circuitos com portas lógicas

- Existe uma correspondência direta entre uma função lógica e o circuito de portas lógicas elementares que a realiza.

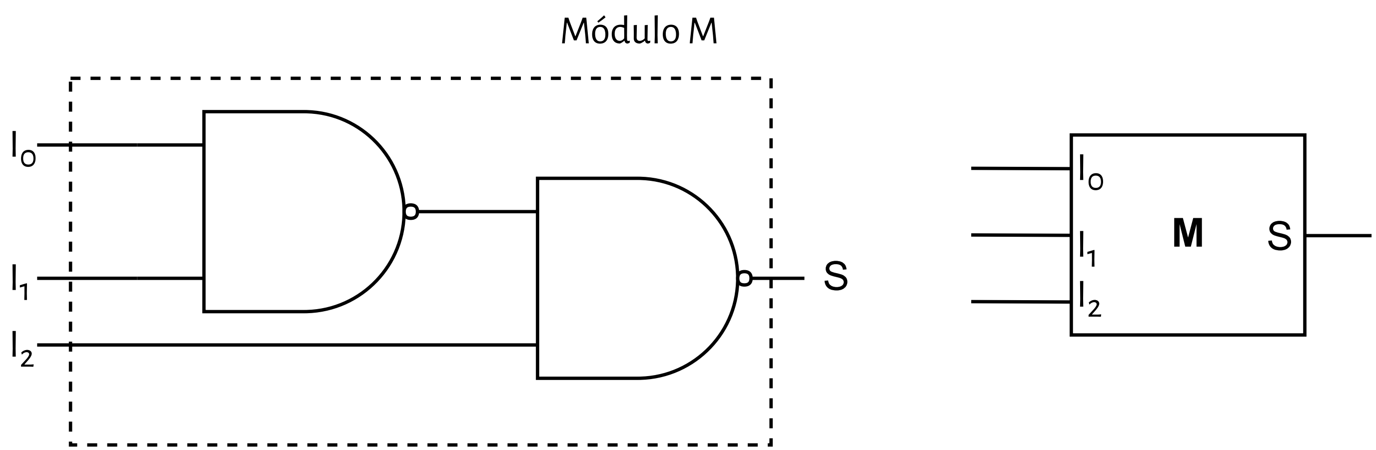
Exemplo:



- Quanto mais simples for a expressão, mais pequeno é o circuito.
- Circuitos diferentes podem realizar a mesma função lógica (correspondem a expressões equivalentes).

Complexidade e modularidade

- Circuitos lógicos podem ser muito complexos $\Rightarrow$ como projetá-los?
- Dividir e conquistar: usar uma abordagem modular e hieráquica:
  - Portas lógicas são usadas para descrever funções lógicas mais complexas, implementadas por “módulos”.
  - Os módulos pode ser usados na definição de circuitos lógicos mais complexos.

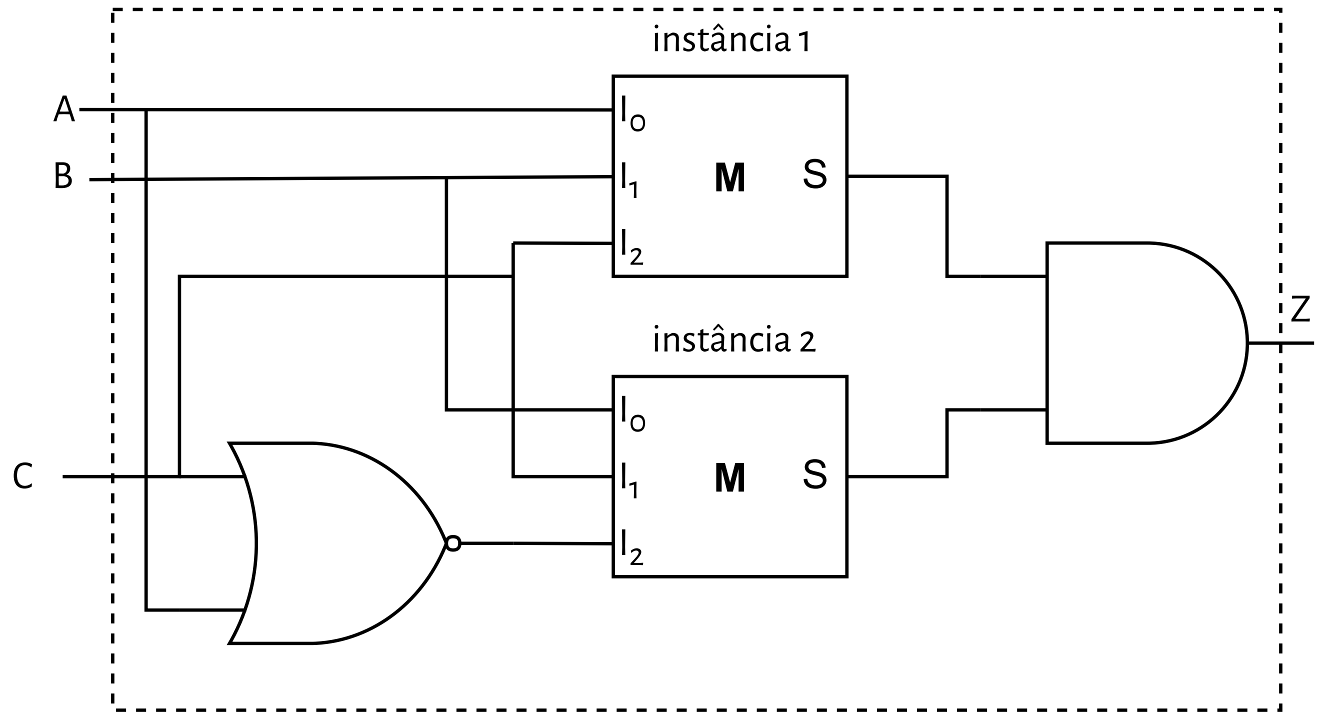


- Função lógica $M(I_2, I_1, I_0) = \overline{\overline{I_0 I_1}\, I_2}$

Descrição hierárquica

- Módulos podem ser combinados com outros módulos e portas lógicas.

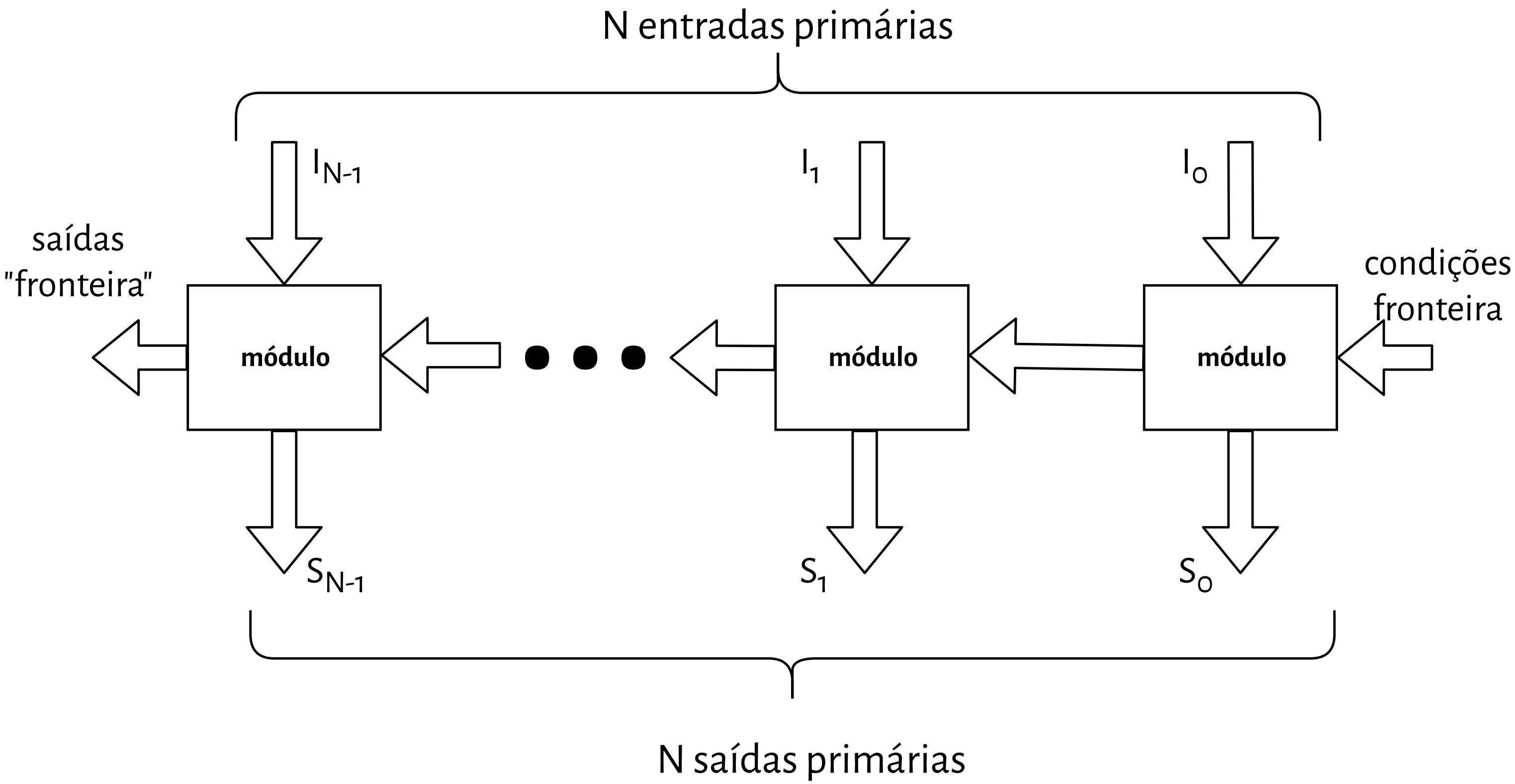
Exemplo de módulo hierárquico $Z(A, B, C) = ?$:



- Este processo pode ser repetido um número arbitrário de vezes.

Circuitos iterativos

- Circuitos **iterativos** são constituídos por repetições de um mesmo módulo.

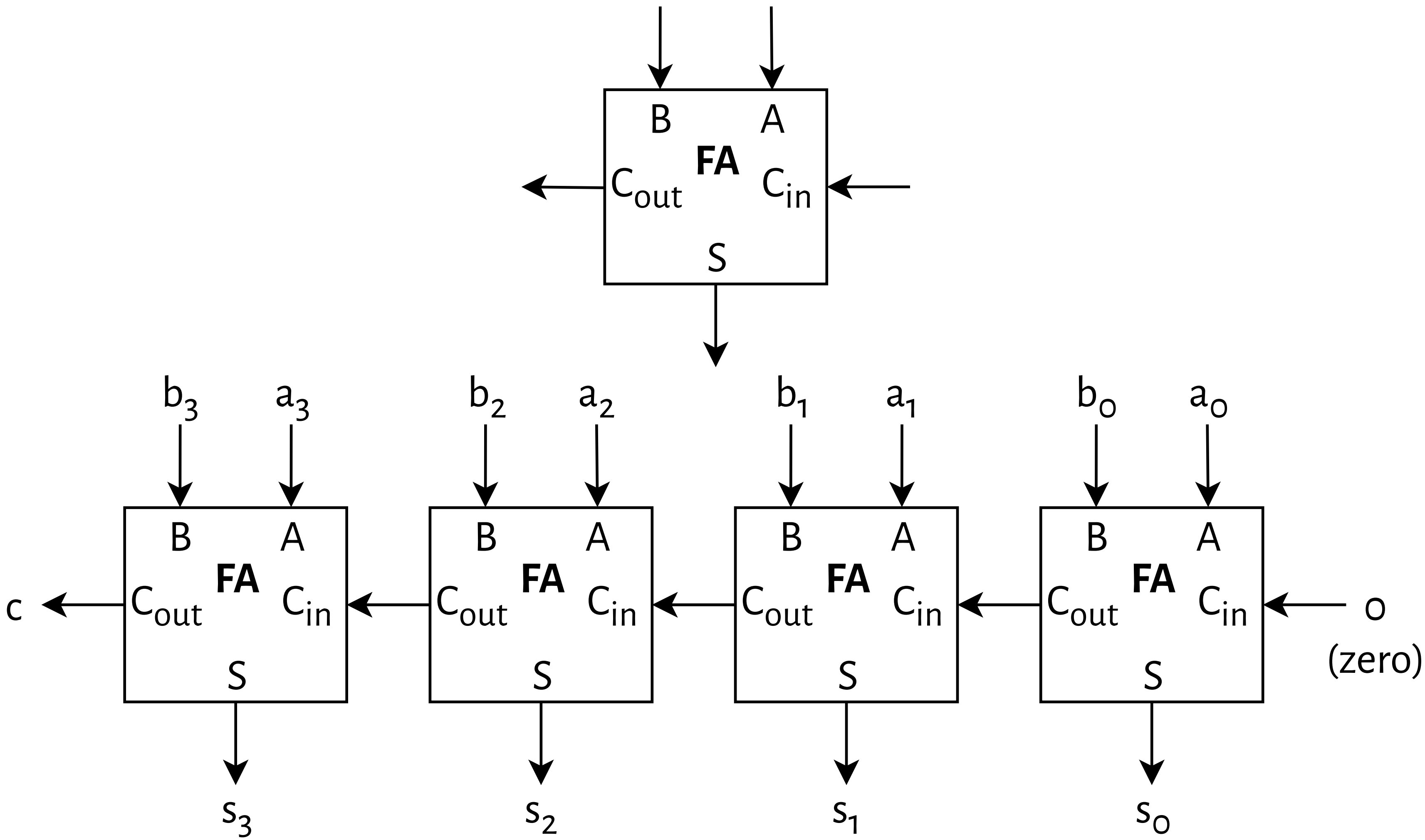


1. Identificar e projetar o módulo de base;
2. Interligar uniformemente instâncias do módulo base (eventualmente com portas lógicas).

- Este tipo de circuito pode ser facilmente expandido.

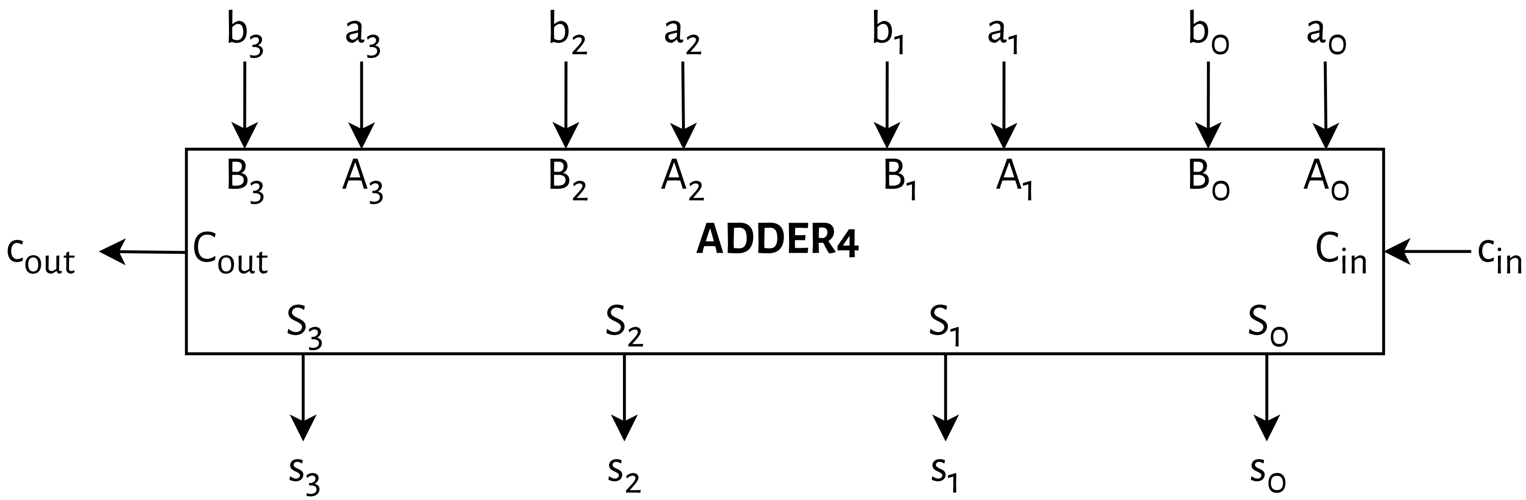
Somador do tipo ripple-carry

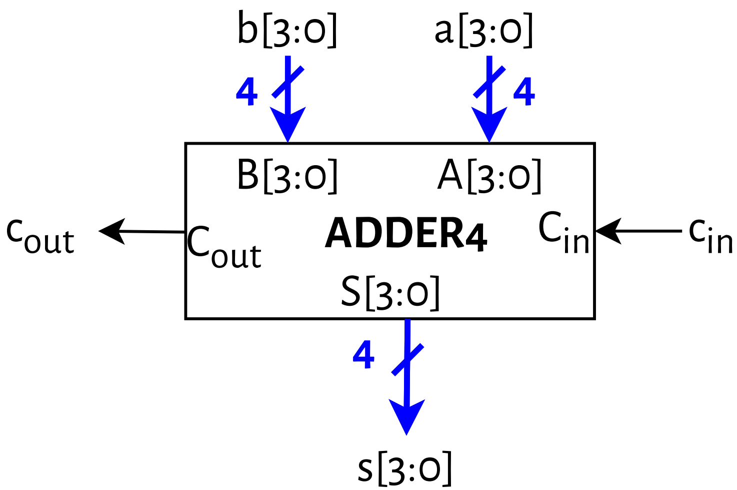
- O somador do tipo *ripple-carry* é um bom exemplo de um circuito iterativo.
- Exemplo: somar dois números de 4 bits $\mathbf{a_3a_2a_1a_0}$ e $\mathbf{b_3b_2b_1b_0}$ usando um módulo que calcula a soma de 2 bits (FA: *full adder*). Resultado: $\mathbf{s_3s_2s_1s_0}$ e **c**



Simplificar diagramas usando barramentos

➠ Um "barramento" é um grupo de sinais que interessa tratar como uma unidade. O seu uso simplifica muito os diagramas (comparar as duas figuras).





1. Álgebra de Boole
   - Representação abstrata do processamento binário
   - Especificação algébrica
   - Representações canónicas

2. Portas lógicas
   - Portas elementares
   - Descrição hierárquica de circuitos

3. **Circuitos padrão**
   - **Multiplexadores**
   - **Descodificadores**
   - **Codificadores**

Funções lógicas comuns

➠ A experiência mostrou que existe um conjunto de funções lógicas que encontram utilização em muitos sistemas digitais.

➠ Essas funções são realizadas por **circuitos padrão** de “média complexidade” (i.e., mais complexos que simples portas lógicas).

➠ A sua utilização facilita o projeto de sistemas digitais

O circuito *full adder* estudado anteriormente pode ser considerado uma dessas funções.

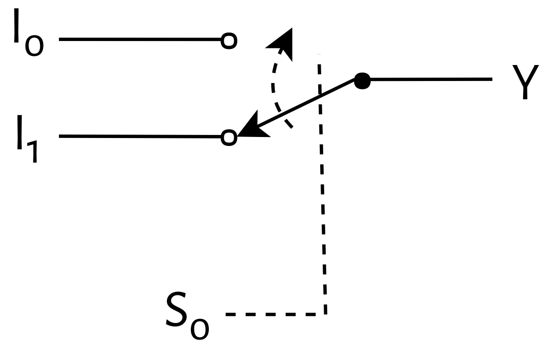
➠ Outras funções incluem comparadores, des/codificadores de vários tipos, de/multiplexadores.

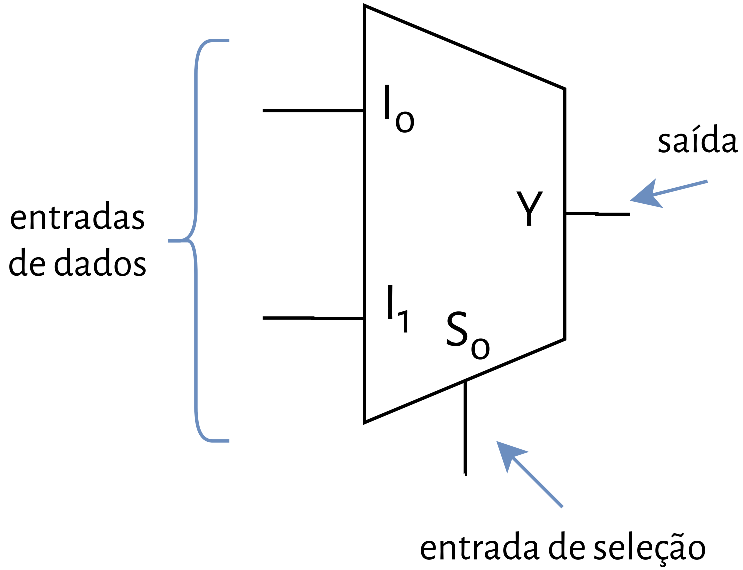
➠ Circuitos padrão estão disponíveis no mercado

➠ Existem normas para alguns símbolos “padrão” (ex.: IEEE Graphic Symbols for Logic Functions IEEE-91)
Mais simples: usar um retângulo com entradas (à esquerda) e saídas (à direita).

Multiplexador de 2 entradas

➠ Um multiplexador *(multiplexer. mux)* 2:1 é um circuito que permite selecionar uma de duas entradas de dados.





S_0	I_1	I_0	$\text{mux}(I_0, I_1)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

➠ Existem multiplexadores 4:1, 8:1, ..., 2^N:1

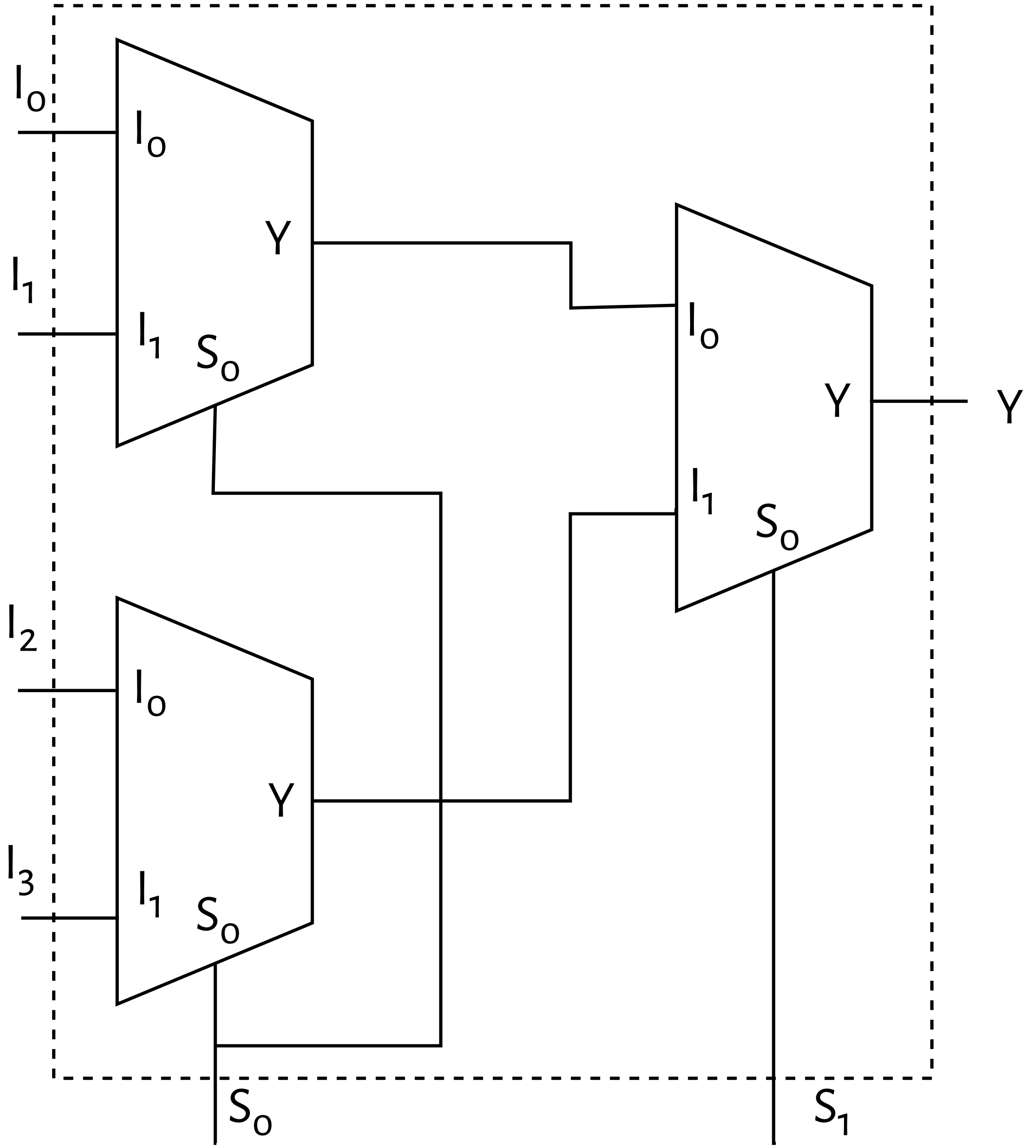
➠ Um multiplexador 2^N:1 tem N entradas de controlo $S_{N-1}, \ldots, S_1, S_0$

➠ Todas as expressões lógicas podem ser implementadas com multiplexadores 2:1. (Porquê?)

➠ Expressões de n variáveis podem ser realizadas com um multiplexador de n entradas de dados.

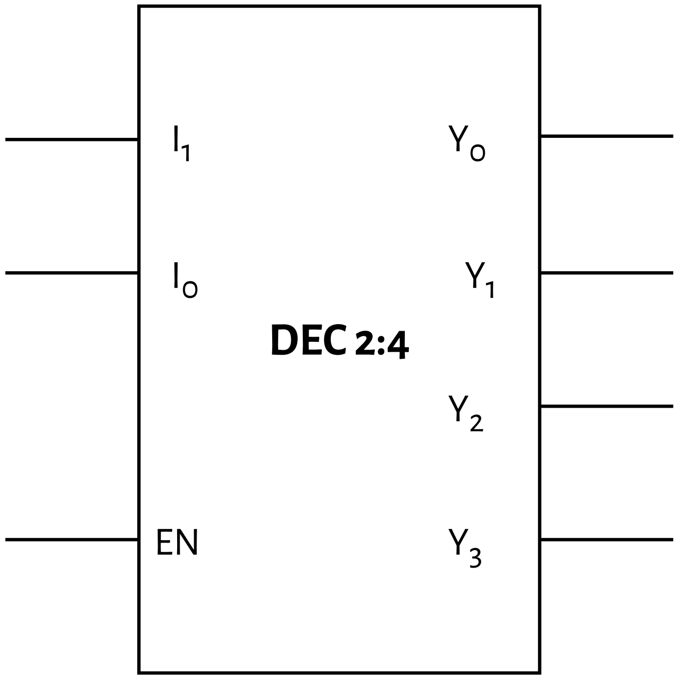
Combinar multiplexadores

- Como construir um multiplexador de 4:1?



Descodificador binário

- Descodificador *(decoder)* de N-para-M (geralmente N < M) transforma um código noutro com mais bits.
- Descodificador binário de N-para-2^N



- Descodificador binário de 2:4

EN	I1	I0	Y3	Y2	Y1	Y0
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

- Como fazer um descodificador binário 3:8 ?

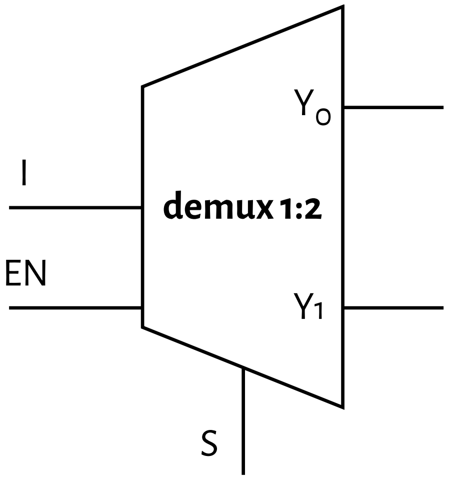
- A entrada EN designa-se por **entrada de habilitação** *(enable)*.
- Descodificador binário seguido de porta lógica OU permite realizar todas as funções de N variáveis. (Como?)

Desmultiplexador

➟ Um desmultiplexador *(demultipler, demux)* de 1:2^N tem 1 entrada de dados, N entradas de controlo (endereço) e 2^N saídas.

➟ O valor da saída selecionada é igual ao da entrada de dados (entrada I).

Exemplo: desmultiplexador 1:2 (N=1)



EN	S_0	I	Y_1	Y_0
0	X	X	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

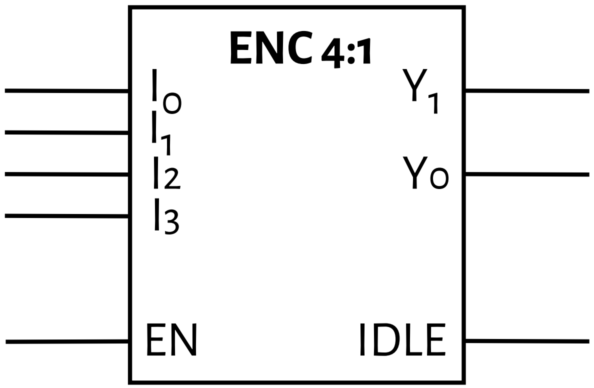
➟ Um desmultiplexador pode ser considerado como um descodificador binário com uma entrada adicional que define o valor da saída selecionada.

➟ Como construir um desmultiplexador 1:4 usando circuitos padrão?

Codificador binário

➠ Um codificador *(encoder)* transforma um código de X bits num código de Y bits, com X > Y.

➠ O codificador binário tem 2^N entradas e N saídas (e sinais de controlo).

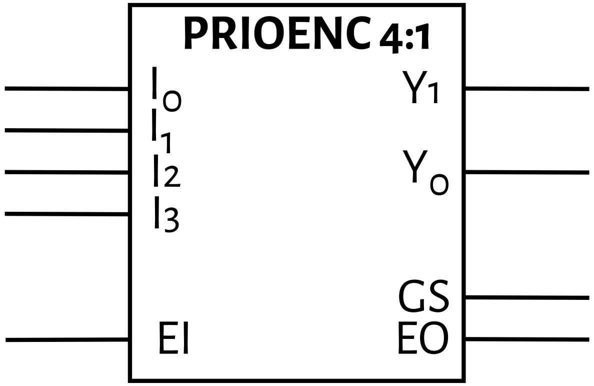


EN	I_3	I_2	I_1	I_0	Y_1	Y_0	IDLE
0	X	X	X	X	0	0	1
1	0	0	0	0	0	0	1
1	0	0	0	1	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	1	0	0	0	1	1	0

➠ Para as restantes combinações de valores de entrada, as saídas não estão definidas!

Codificador de prioridade

➠ A saída de um codificador de prioridade é definida pela entrada de maior prioridade que estiver a "1".
Exemplo: codificador de prioridade 4:2 (I_3 tem a maior prioridade; I_0 a menor)



EI	I_3	I_2	I_1	I_0	Y_1	Y_0	GS	EO
0	X	X	X	X	0	0	0	0
1	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	0
1	0	0	1	X	0	1	1	0
1	0	1	X	X	1	0	1	0
1	1	X	X	X	1	1	1	0

EI: *(enable input)* circuito habilitado;
EO: *(enable output)* para habilitar circuito de menor prioridade;
GS: *(got something)* está a "1"se EI=1 e alguma entrada de dados está a "1"

☞ Como fazer um codificador de prioridade 8:3 com dois codificadores 4:2 e portas lógicas?

Referências

COD4 D. A. Patterson & J. L. Hennessey, Computer Organization and Design, 4 ed.

COD3 D. A. Patterson & J. L. Hennessey, Computer Organization and Design, 3 ed.

Alguns dos tópicos tratados nesta apresentação são descritos nas seguintes secções de [COD4]:

- apêndice C, secções C.1–C.3

Também são tratados nas seguintes secções de [COD3]:

- apêndice B, secções B.1–B.3