Lógica e Sistemas Digitais - 5

Códigos Numéricos, Circuitos Aritméticos e Iterativos

ISEL
Departamento de Engenharia de Electrónica
e Telecomunicações e de Computadores
Lisboa

Mário Araújo

2016-1

Sistema **POSICIONAL** ou ponderado: o número N é formado por uma sequência de dígitos ou algarismos, em que cada um possui um **peso** característico da posição que ocupa na sequência:

$$N = \sum_{i=-n}^{p-1} d_i \times b^i = \ldots + d_2 \times b^2 + d_1 \times b^1 + d_0 \times b^0 + d_{-1} \times b^{-1} + \ldots$$

Expressão de **SIGNIFICÂNCIA POSICIONAL** em que:

- d_i – dígitos (algarismos) da base b
- **b** – base de numeração
- **i** – índice posicional do algarismo
- **p** – nº de algarismos à esquerda da vírgula
- **n** – nº de algarismos à direita da vírgula

Numa base **b**, o algarismo de maior valor intrínseco é **b-1**. Na base dezasseis (hexadecimal) usam-se os símbolos A, B, C, D, E, F para significar os algarismos de valor intrínseco superior a nove:

Símbolos válidos nas seguintes bases:

Base 10 (Decimal):	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Base 16 (Hexadecimal):	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
Base 8 (Octal):	0, 1, 2, 3, 4, 5, 6, 7
Base 2 (Binária):	0, 1

MCDLXXIV

O sistema de numeração romano é não posicional (não há pesos associados aos dígitos)

EXPRESSÕES DE SIGNIFICÂNCIA POSICIONAL (EX. 5-1)

Exemplo 5-1

Usando o algoritmo de cálculo de um número:

$974.625_{10} = 9 \times 10^2 + 7 \times 10^1 + 4 \times 10^0 + 6 \times 10^{-1} + 2 \times 10^{-2} + 5 \times 10^{-3}$

$3CE.A_{16} = 3 \times 16^2 + C \times 16^1 + E \times 16^0 + A \times 16^{-1} = 974.625_{10}$

$1716.5_{8} = 1 \times 8^3 + 7 \times 8^2 + 1 \times 8^1 + 6 \times 8^0 + 5 \times 8^{-1} = 974.625_{10}$

$1111001110.101_{2} = 1 \times 2^9 + 1 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^{-1} + 1 \times 2^{-3} =$

$= 512 + 256 + 128 + 64 + 8 + 4 + 2 + 0,5 + 0,125 = 974.625_{10}$

- Um conjunto de 4 bits designa-se por **NIBBLE**
- Um conjunto de 8 bits designa-se por **BYTE**
- O bit mais significativo (com maior peso, mais à esquerda) designa-se **MSB** (MOST SIGNIFICANT DIGIT)
- O bit menos significativo (com menor peso, mais à direita) designa-se **LSB** (LEAST SIGNIFICANT DIGIT)
- Um número hexadecimal com dois dígitos representa um byte (exemplo 3F)
- Um número hexadecimal com quatro dígitos representa dois bytes (exemplo 3FEA).

Conversão entre Base 16 / Base 8 e Base 2 (Ex. 5-2)

Exemplo 5-2

$$1111001110.101_2 = 0011\ 1100\ 1110\ .\ 1010_2 =$$

$$= 3CE.A_{16} =$$

$$= 001\ 111\ 001\ 110\ .\ 101_2 =$$

$$= 1716.5_8$$

O processo de conversão da base 16 para a base 2 é o oposto do anterior: o número na base tem cada um dos seus dígitos decomposto em 4 bits.
Idem para a base 8, com grupos de 3 bits em vez de 4 bits:

$$3CE.A_{16} = \underbrace{0011}_{3}\ \underbrace{1100}_{C}\ \underbrace{1110}_{E}\ .\ \underbrace{1010}_{A}{}_2$$

$$1716.5_8 = \underbrace{001}_{1}\ \underbrace{111}_{7}\ \underbrace{001}_{1}\ \underbrace{110}_{6}\ .\ \underbrace{101}_{5}{}_2$$

As bases **16** e **8** são úteis para o sistema de numeração binário por serem potências inteiras de **2**.

Cada dígito **hexadecimal** é formado sempre por sequências de 4 dígitos binários, e cada dígito **octal** por sequências de 3 dígitos binários.

No processo de conversão da base 2 para a base 16 formam-se **grupos de 4 bits** (**nibbles**) a partir da vírgula do número binário, para a esquerda e para a direita, e escreve-se um dígito hexadecimal por cada grupo.

Se restarem grupos com menos do que 4 bits (nas posições mais significativas e menos significativas), preenchem-se esses agrupamentos com zeros não significativos até ficarem com 4 bits.

Idem para a base 8, com agrupamentos de 3 em vez de 4 bits.

CÓDIGO BCD (BINARY-CODED DECIMAL) (EX. 5-4)

Exemplo 5-4

- O código **BCD** codifica os dígitos decimais dígito a dígito.
- Para cada dígito de 0 a 9 é utilizada a sua representação binária sem sinal a 4 bits, entre 0000 e 1001 (as configurações entre 1010 e 1111 não são utilizadas).
- O código BCD corresponde à conversão de cada algarismo decimal na palavra de código binário a 4 bits correspondente, salvaguardando a ordem relativa dos dígitos a que correspondem.
- O código BCD é um código ponderado mas não é um sistema de numeração, dado os sucessivos pesos dos caracteres não serem todos potências inteiras de dois.
- Os pesos dos sucessivos caracteres BCD são 1, 2, 4, 8, 10, 20, 40, 80, 100, …
- As conversões entre as representações BCD e as representações decimais são imediatas, substituindo-se directamente cada grupo de 4 bits num dígito decimal e vice-versa. Exemplos:

$$745_{10} = \underbrace{0111}_{7}\ \underbrace{0100}_{4}\ \underbrace{0101}_{5}{}_{BCD} \qquad \underbrace{0110}_{6}\ \underbrace{0011}_{3}\ \underbrace{1001}_{9}{}_{BCD} = 639_{10}$$

n	2^n	
0	2^0 = 1	
1	2^1 = 2	
2	2^2 = 4	
3	2^3 = 8	
4	2^4 = 16	
5	2^5 = 32	
6	2^6 = 64	
7	2^7 = 128	
8	2^8 = 256	
9	2^9 = 512	
10	2^{10} = 1024	**K**ilo
11	2^{11} = 2048	
12	2^{12} = 4096	
20	2^{20} = 1M	**M**ega
30	2^{30} = 1G	**G**iga
40	2^{40} = 1T	**T**era
50	2^{50} = 1P	**P**eta

Potências inteiras sucessivas de 2.

Decimal	Binário (4 bits)	Hexa-decimal	Octal	BCD
0	0000	0	0	0000
1	0001	1	1	0001
2	0010	2	2	0010
3	0011	3	3	0011
4	0100	4	4	0100
5	0101	5	5	0101
6	0110	6	6	0110
7	0111	7	7	0111
8	1000	8	10	1000
9	1001	9	11	1001
10	1010	A	12	0001 0000
11	1011	B	13	0001 0001
12	1100	C	14	0001 0010
13	1101	D	15	0001 0011
14	1110	E	16	0001 0100
15	1111	F	17	0001 0101

A designação **1k** (“quilo”) dada à potência 2^{10} =$1024_{(10)}$ advém de ser essa a potência de 2 que mais se aproxima de $1000_{(10)}$.

CONVERSÃO ENTRE BASE 10 E BASE 2 (EX. 5-3)

Exemplo 5-3

A conversão da **parte inteira** é feita pelo método das **divisões sucessivas**, que consiste em reter os restos das sucessivas divisões por 2 do número decimal dado e dos quocientes entretanto obtidos, até obter um quociente nulo.

Exemplo para o NÚMERO DECIMAL **13**:

		Quociente Inteiro	Resto	Dígito
13/2	=	6	1	$d_0 = 1$
6/2	=	3	0	$d_1 = 0$
3/2	=	1	1	$d_2 = 1$
1/2	=	0	1	$d_3 = 1$

$$(\mathbf{13})_{10} = (d_3\, d_2\, d_1\, d_0)_2 = (1101)_2$$

Para a conversão da **parte decimal** utiliza-se o método das **multiplicações sucessivas**, que consiste em reter as partes inteiras das multiplicações por 2 da parte decimal dada e das sucessivas partes decimais entretanto obtidas, até que seja atingida a precisão pretendida para a parte fraccionária do número convertido.

Exemplo para o NÚMERO DECIMAL **0.625**:

		Parte Inteira		Parte Decimal	Dígito
0.625	x 2 =	1	+	0.25	$d_{-1} = 1$
0.250	x 2 =	0	+	0.50	$d_{-2} = 0$
0.500	x 2 =	1	+	0	$d_{-3} = 1$

$$(\mathbf{0.625})_{10} = (0\,.\,a_{-1}\, a_{-2}\, a_{-3})_2 = (0.101)_2$$

Código Gray a 4 bits – Conversão Gray-Binário Natural

Código Binário Natural	Nº Decimal	Código Binário Reflectido (Código Gray)
$B_3\ B_2\ B_1\ B_0$		$G_3\ G_2\ G_1\ G_0$
0 0 0 0	0	0 0 0 0
0 0 0 1	1	0 0 0 1
0 0 1 0	2	0 0 1 1
0 0 1 1	3	0 0 1 0
0 1 0 0	4	0 1 1 0
0 1 0 1	5	0 1 1 1
0 1 1 0	6	0 1 0 1
0 1 1 1	7	0 1 0 0
1 0 0 0	8	1 1 0 0
1 0 0 1	9	1 1 0 1
1 0 1 0	10	1 1 1 1
1 0 1 1	11	1 1 1 0
1 1 0 0	12	1 0 1 0
1 1 0 1	13	1 0 1 1
1 1 1 0	14	1 0 0 1
1 1 1 1	15	1 0 0 0

Tabela de verdade a 4 bits para os Códigos Binário Natural e Gray.

O Código Gray é **não ponderado** **contíguo** e **cíclico** - não podem ser atribuídos pesos às posições dos bits nas palavras (ou associar a cada palavra um equivalente decimal).

A sua característica essencial é que as figuras de código correspondentes a linhas consecutivas só diferem num único bit, o mesmo acontecendo às quantidades 0 e 2^n-1.

$$G_0 = B_0 \oplus B_1$$
$$G_1 = B_1 \oplus B_2$$
$$\ldots \quad \ldots \quad \ldots$$
$$G_k = B_k \oplus B_{k+1}$$
$$G_n = B_n$$

$$B_0 = G_0 \oplus G_1 \oplus \ldots \oplus G_n$$
$$B_1 = G_1 \oplus G_2 \oplus \ldots \oplus G_n$$
$$\ldots \quad \ldots \quad \ldots$$
$$B_k = G_k \oplus G_{k+1} \oplus \ldots \oplus G_n$$
$$B_n = G_n$$

Expressões de conversão extraídas directamente da tabela.

Circuitos Conversores Binário-Gray e Gray-Binário

De acordo com os registos históricos conhecidos o engenheiro francês Baudot usou códigos binários reflectidos em telegrafia em 1878.

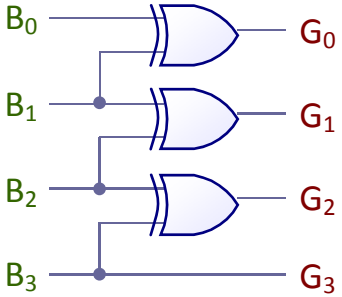
O engenheiro americano Frank Gray, dos laboratórios Bell, foi quem introduziu o termo **Código Binário Reflectido** (CBR) em 1947 num pedido de registo de patente (extracto ao lado), relativo a um método de conversão de sinais analógicos realizado com válvulas no domínio da TV a cores.

A patente foi concedida em 1953 e o CBR popularizou-se posteriormente como **Código Gray**.

The binary code with which the present invention deals may take various forms, all of which have the property that the symbol (or pulse group) representing each number (or signal amplitude) differs from the ones representing the next lower and the next higher number (or signal amplitude) in only one digit (or pulse position). Because this code in its primary form may be built up from the conventional binary code by a sort of reflection process and because other forms may in turn be built up from the primary form in similar fashion, the code in question, which has as yet no recognized name, is designated in this specification and in the claims as the "reflected binary code."

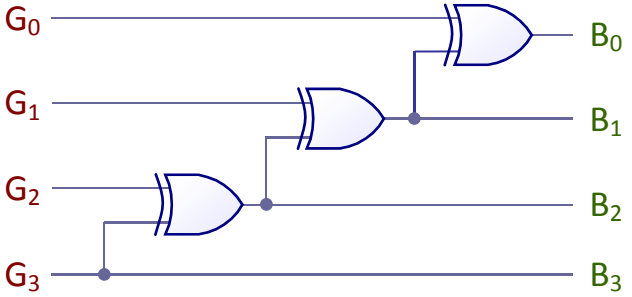
Frank Gray

1887-1969
(USA)



$G_k = B_k \oplus B_{k+1}$

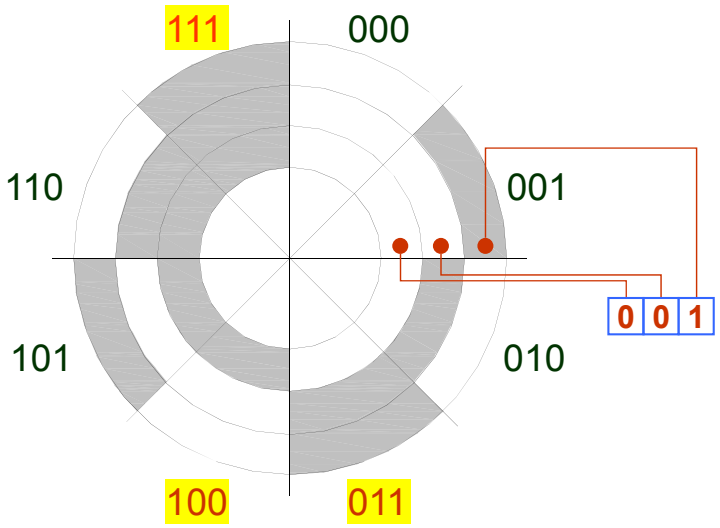
$G_n = B_n$



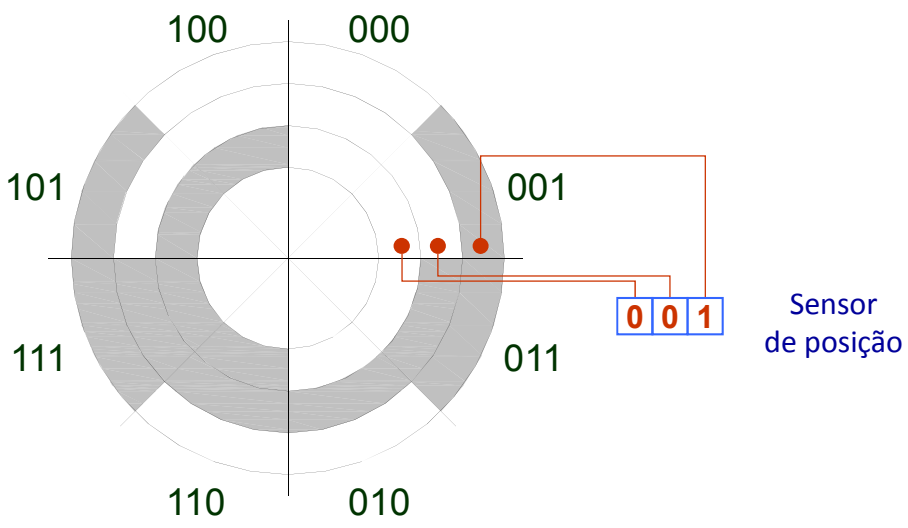
$B_k = G_k \oplus G_{k+1} \oplus \ldots \oplus G_n$

$B_n = G_n$

Utilização do Código Gray em Sistemas Electromecânicos



Disco codificado em
Código Binário Natural 3 bits.



Disco codificado em Código Gray a 3 bits.

Num disco com Codificação Binária Natural um ligeiro desalinhamento relativo do disco e do sensor de leitura na zona de transição dos sectores pode induzir um erro significativo - por exemplo, o código 011 pode ser lido como 111, o que significa um erro de 180º.

A codificação de discos em Código Gray diminui os erros de leitura e aumenta a fiabilidade dos sistemas de detecção de posição mecânica dos dispositivos rotativos.

CÓDIGOS ALFANUMÉRICOS

O código alfanumérico ASCII (AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE) original data de 1963 e possui 7 bits o que permite codificar um conjunto de 128 caracteres que correspondem a cada letra do alfabeto, a cada um dos algarismos, a diversos sinais de pontuação e a algumas funções especiais (caracteres não-imprimíveis como espaço e mudança de linha). Com o tempo, os 128 códigos mostraram-se insuficientes e aumentou-se o número de bits do código para 8 e o número de símbolos diferentes para 256 (ASCII estendido ou expandido).

Por exemplo o código para F é $46_H = 100\ 0110_2$ e a palavra VER pode ser representada como 1010110 1000101 1010010.

O código UNICODE foi criado entre 1988 e 1991. Utiliza 16 bits 65 536 códigos de caracteres possíveis para a normalização da codificação dos caracteres utilizados por todas as escritas existentes mundo.

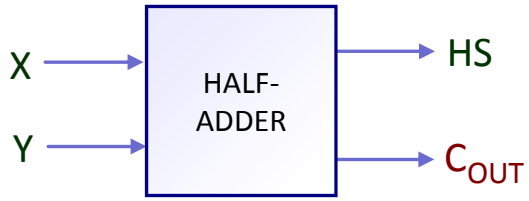
Palavra do código ASCII original a 7 bits.

$B_7\ B_6\ B_5$ \ $B_4\ B_3\ B_2\ B_1$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	“	#	$	%	&	‘	(	)	*	+	,	-	,	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Tabela do código ASCII original a 7 bits.

CIRCUITO SEMI-SOMADOR (HALF-ADDER)

$X + Y$ X e Y são operandos de um bit que produzem uma soma de dois bits.



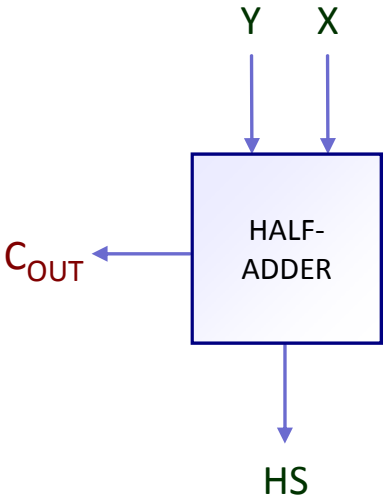
(HALF-SUM, MEIA SOMA)

(C_{OUT}, TRANSPORTE DE SAÍDA OU ARRASTO)

Símbolo lógico do módulo SEMI-SOMADOR.

Y	X	HS	C_{OUT}	
0	0	0	0	(0 + 0 = 0)
0	1	1	0	(0 + 1 = 1)
1	0	1	0	(1 + 0 = 1)
1	1	0	1	(1 + 1 = 10)

Tabela da adição binária.

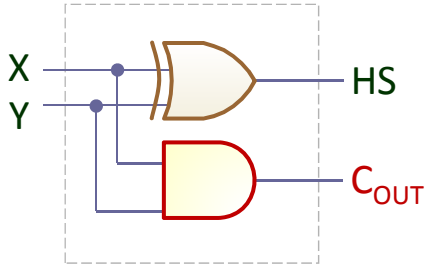


Símbolo lógico alternativo
do módulo SEMI-SOMADOR para iteração.

$$HS = X \oplus Y$$

$$C_{OUT} = X \cdot Y$$

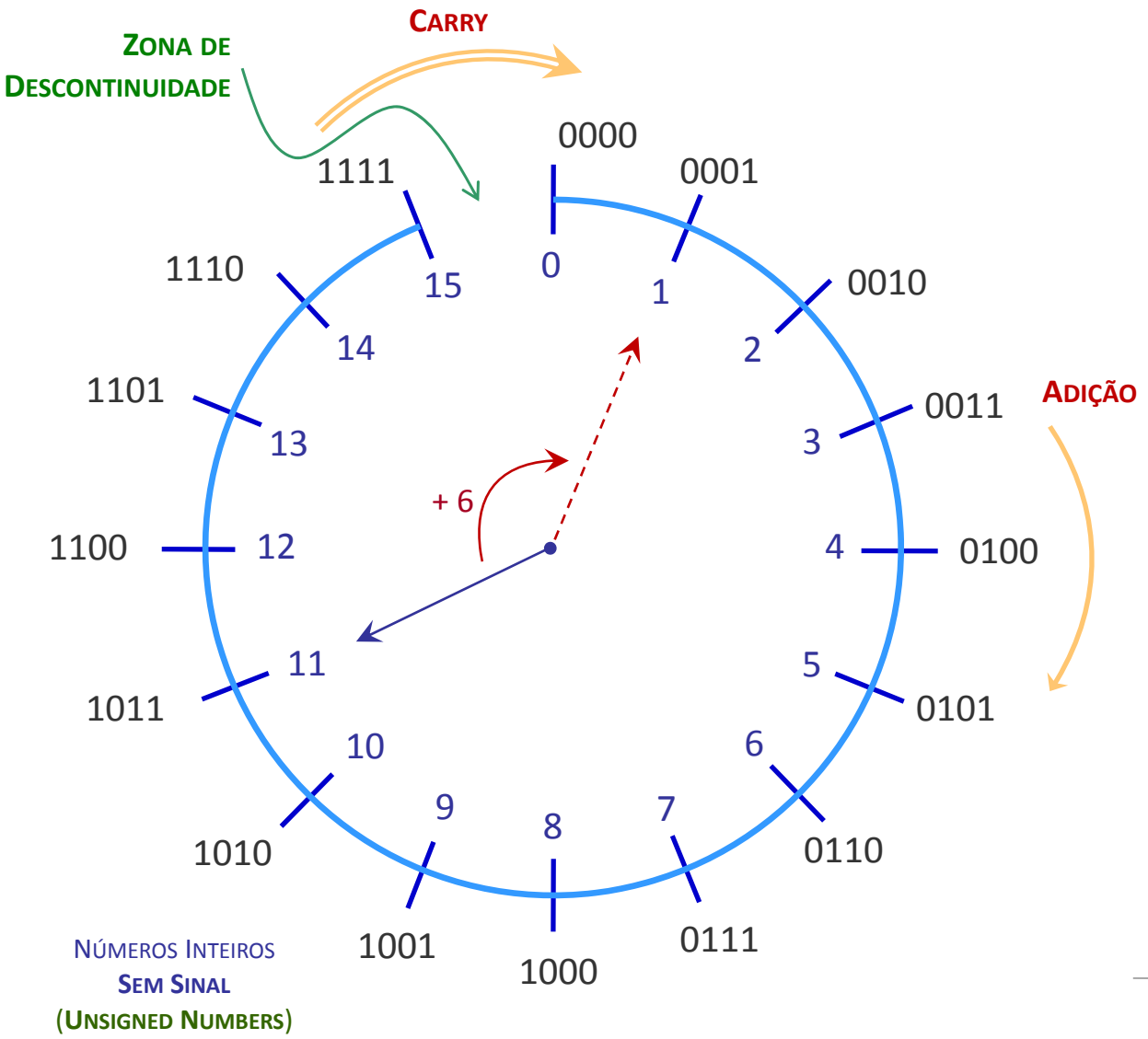
Expressões algébricas
para as funções HS e C_{OUT}.



Síntese de um módulo SEMI-SOMADOR com portas lógicas.

No SEMI-SOMADOR a soma inicia-se sem ARRASTO prévio (CARRY-IN).

Adição de Números Inteiros sem Sinal a 4 bits – representação gráfica



Representação gráfica da adição em Roda de Números.

Com n bits podem representar-se os números inteiros **i** no intervalo:

$$0 \le i \le 2^n - 1$$

- Um número é adicionado a outro movendo a seta no anel representado (Roda dos Números) um número de casas correspondente no sentido dos ponteiros do relógio.
- Se o resultado da operação Soma exceder a capacidade de representação, a seta mover-se-á através da zona de descontinuidade.
- Neste caso ocorre um Carry e terá de ser tomado em conta o bit de transporte.

Exemplo da adição dos números 11 e 6 com Carry :

		Peso	16	8	4	2	1
			1	1	1	0	
	11			1	0	1	1
+	6	+		0	1	1	0
	17		1	0	0	0	1
			Carry	Resultado a 4 bits, errado			

Adição Binária, Hexadecimal e Octal (Ex. 5-5)

Exemplo 5-5

Adição decimal		Peso	Adição Binária									Adição Hexadecimal			Adição Octal		
100	10	1		256	128	64	32	16	8	4	2	1	256	16	1	64	8	1
1	1	0	C_{IN}	1	0	1	1	1	1	1	1	0		1	0		1	0
1	9	1	X		1	0	1	1	1	1	1	1		B	F		2	7	7
+ 1	4	9	+ Y	+	1	0	0	1	0	1	0	1	+	9	5	+	2	2	5
3	4	0	X + Y	1	0	1	0	1	0	1	0	0	1	5	4		5	2	4

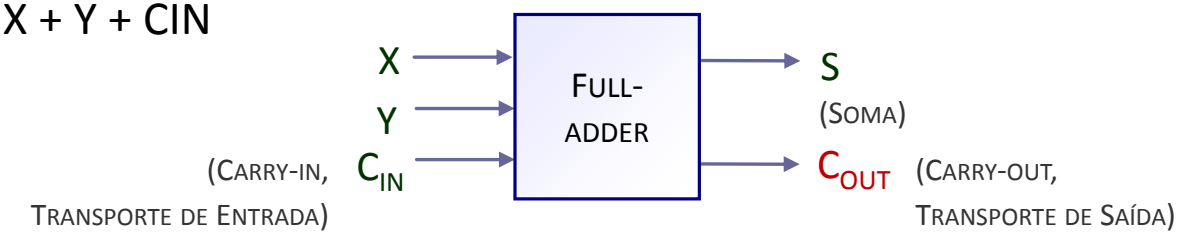
O algoritmo da adição Binária é formalmente idêntico ao da adição decimal excepto que se utiliza a base b=2 em vez de b=10. Idem para a adição Octal e Hexadecimal, e para qualquer sistema de numeração posicional.

Para executar uma adição começa-se pela coluna da direita, adicionando-se X + Y e colocando o resultado dessa adição ainda nessa coluna em baixo. Se houver Transporte ou Arrasto (Carry-out), o seu valor, indicado a vermelho na linha mais acima, é adicionado aos valores X e Y da coluna imediatamente à esquerda, funcionando como Carry-in dessa coluna. O algoritmo prossegue de forma idêntica para todas as colunas.

Para se implementar um circuito capaz de somar dois números de 32 bits ou 64 bits cada, e produzir um resultado de igual número de bits e ainda o Carry-out, como acontece nas Unidades Aritméticas dos processadores actuais, é necessário desenvolver uma aplicação iterativa baseada num módulo replicável, que, para além das entradas X e Y de cada operando, possua sinais de entrada e de saída para ligação em cascata e diálogo com os módulos vizinhos. Esse módulo designa-se por Somador Completo (Full-adder) e passa a somar 3 algarismos de 1 bit: X, Y e Carry-in, e a produzir 2 bits de resultado, Soma e Carry-out.

Circuito Somador Completo (Full-adder)

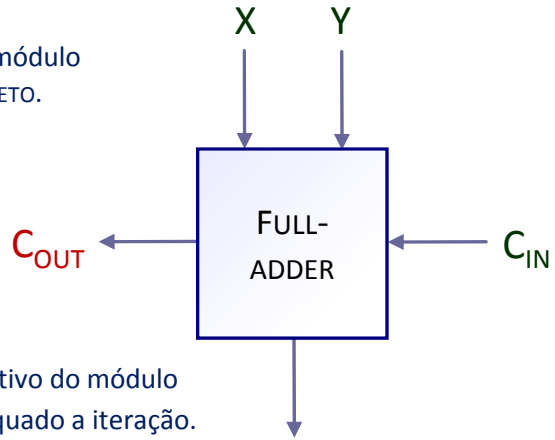
X + Y + CIN



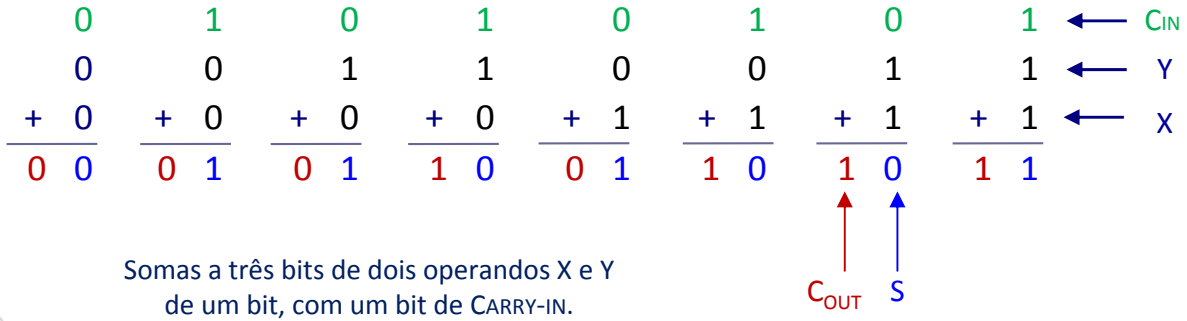
C_{IN}	Y	X	S	C_{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabela resultante da soma de 3 bits com a Soma (S) a ser um XOR e o Carry-out a função Maioria das três entradas..

Símbolo lógico do módulo Somador Completo.



Símbolo lógico alternativo do módulo Somador Completo adequado a iteração.



Somas a três bits de dois operandos X e Y de um bit, com um bit de Carry-in.

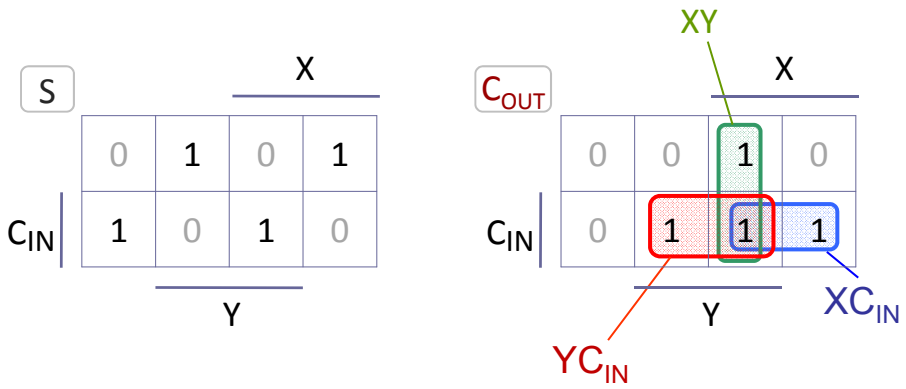
$$S = X \oplus Y \oplus C_{IN}$$

$$C_{OUT} = (X \oplus Y)\, C_{IN} + XY \quad \text{ou}$$

$$C_{OUT} = XC_{IN} + YC_{IN} + XY$$

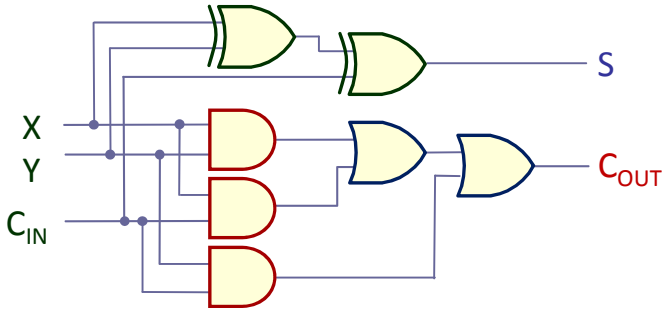
Expressões algébricas para as funções S e C_{OUT} .

SOMADOR COMPLETO – 3 IMPLEMENTAÇÕES POSSÍVEIS A PARTIR DE PORTAS DE 2 ENTRADAS



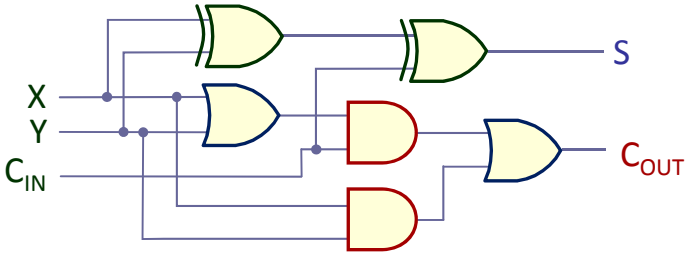
Quadros de Karnaugh das funções S (Soma) e C_{OUT} (Carry Out).

1 Implementação com 7 portas (de 2 entradas)



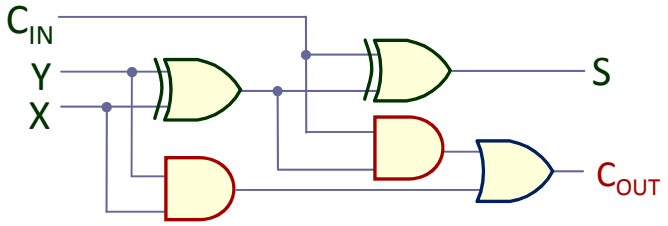
$$S = X \oplus Y \oplus C_{IN}$$
$$C_{OUT} = XC_{IN} + YC_{IN} + XY$$

2 Implementação com 6 portas



$$S = X \oplus Y \oplus C_{IN}$$
$$C_{OUT} = (X + Y)\, C_{IN} + XY$$

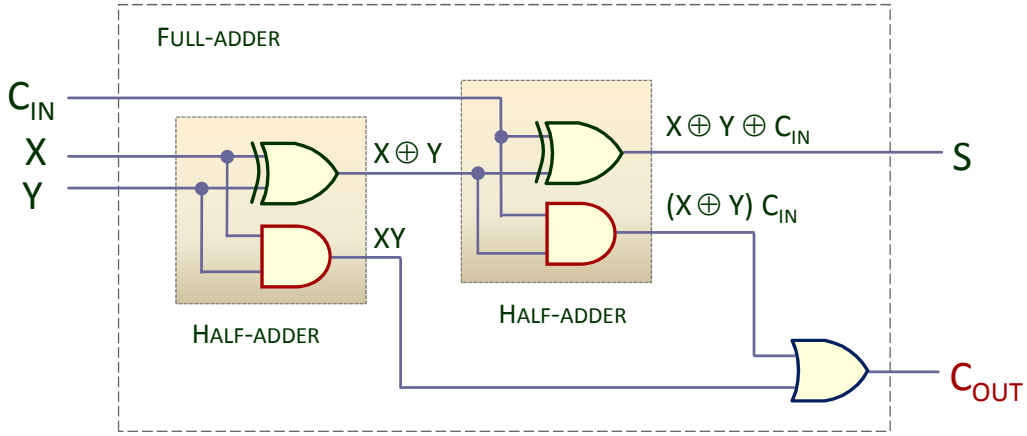
3 Implementação com 5 portas: versão **iterativa**



$$S = X \oplus Y \oplus C_{IN}$$
$$C_{OUT} = (X \oplus Y)\, C_{IN} + XY$$

As expressões de C_{OUT} que figuram em ② e ③ acima são equivalentes como ficou demonstrado no Capítulo-2 (slide 2-24) na referência à função MAIORIA M(x,y,z) – igual a 1 sempre que pelo menos dois dos seus três argumentos sejam iguais a 1.

SOMADOR COMPLETO– IMPLEMENTAÇÃO A PARTIR DE 2 SEMI-SOMADORES



Implementação com 5 portas: versão ITERATIVA estruturada por inferência modular a partir de dois SEMI-SOMADORES e de uma porta OR – um terceiro semi-somador nunca produziria arrasto e é substituído por uma porta OR ou XOR.

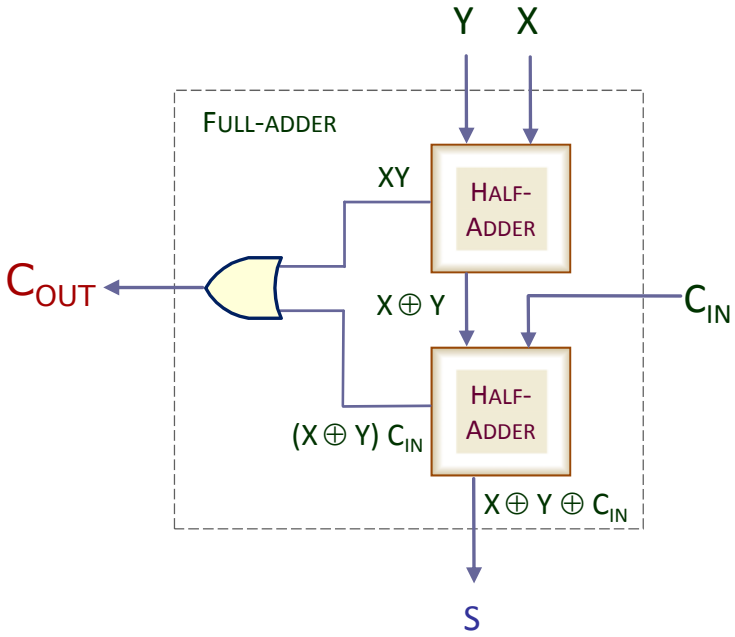
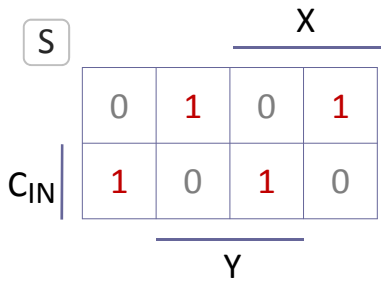


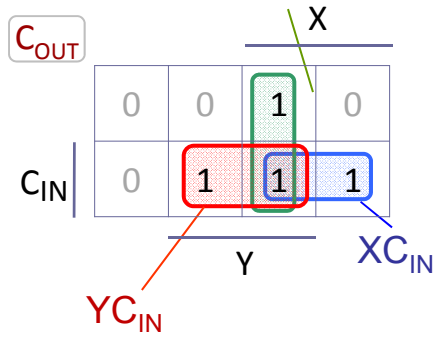
Diagrama de blocos do SOMADOR COMPLETO em simbologia iterativa.

$$S = X \oplus Y \oplus C_{IN}$$
$$C_{OUT} = (X \oplus Y)\, C_{IN} + XY$$

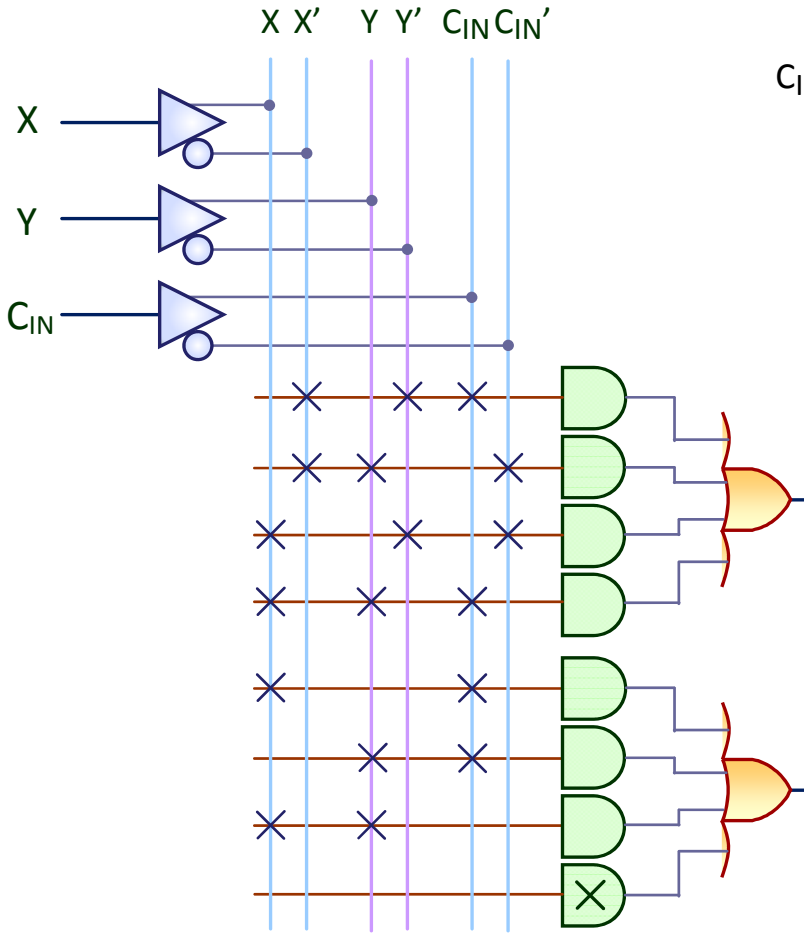
Expressões algébricas das funções S e C_{OUT}.

SOMADOR COMPLETO – IMPLEMENTAÇÃO AND-OR EM NOTAÇÃO PAL





Quadros de Karnaugh assinalando os mintermos das funções S (SOMA) e os termos produto da minimização AND-OR do C_{OUT} (CARRY OUT).



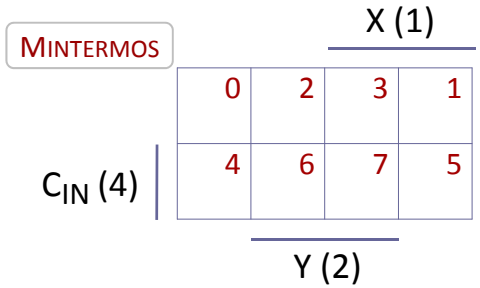
$$S = X \oplus Y \oplus C_{IN} =$$

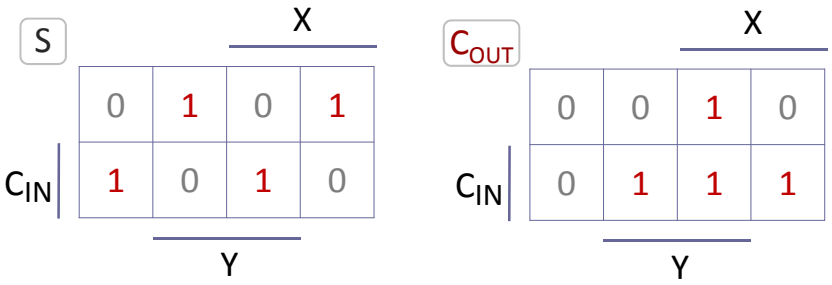
$$= X' Y' C_{IN} + X' Y C_{IN}' + X Y' C_{IN}' + X Y C_{IN}$$

$$C_{OUT} = X C_{IN} + Y C_{IN} + X Y$$

Diagrama lógico do circuito SOMADOR COMPLETO implementado em PAL em 2 saídas de 4 termos produto cada.

Somador Completo – Implementação a partir de Decoder e Portas Lógicas





Quadros de Karnaugh assinalando os mintermos das funções S (Soma) e C_{OUT} (Carry Out).

A implementação de circuitos com Decoders torna-se interessante quando há que gerar várias funções dependentes do mesmo conjunto de variáveis. O Somador Completo utiliza os termos produto gerados pelo descodificador.

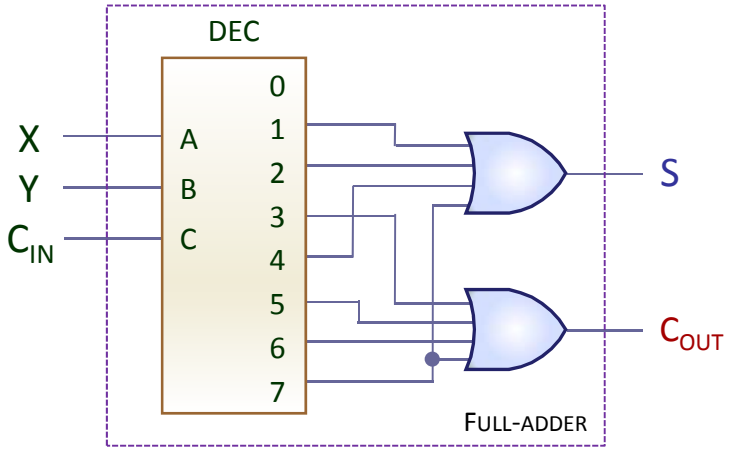


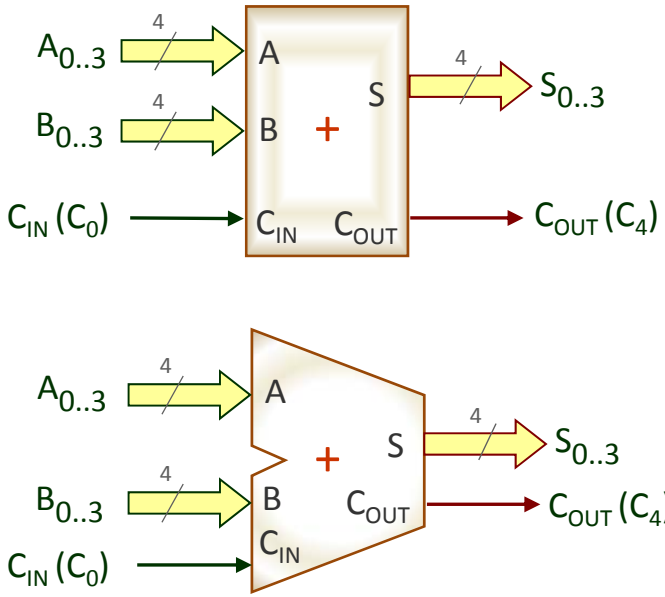
Diagrama lógico do circuito Somador Completo implementado a partir de um Decoder 3-to-8 e portas OR.

$$S = X \oplus Y \oplus C_{IN} = \sum m\,(1, 2, 4, 7)$$

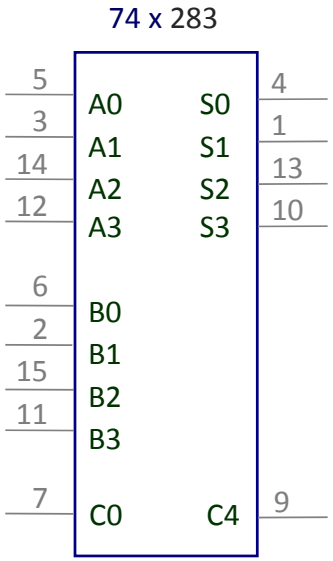
$$C_{OUT} = (X \oplus Y)\, C_{IN} + XY = \sum m\,(3, 5, 6, 7)$$

Expressões algébricas das funções S e C_{OUT}.

74 x 283 – 4 BIT BINARY FULL ADDER



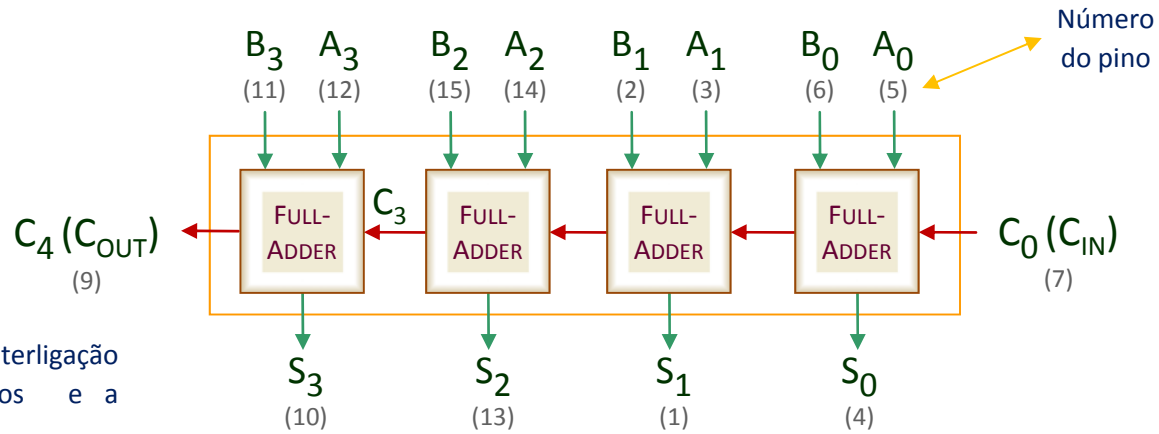
Representação em diagrama de blocos do SOMADOR de números de 4 bits: duas simbologias equivalentes.



Símbolo do circuito com descrição dos pinos.

Existem circuitos integrados contendo SOMADORES de números de vários algarismos, e com a possibilidade de concatenação. É exemplo o circuito 74 x 283, designado 4 BIT BINARY FULL ADDER WITH FAST CARRY.

$A_0 - A_3$	Entradas do operando A a 4 bits
$B_0 - B_3$	Entradas do operando B a 4 bits
$S_0 - S_3$	Saídas - resultado soma a 4 bits
C_0	CARRY-IN do bit 0, bit simples que representa o arrasto prévio
C_4	CARRY-OUT do bit 3 (peso 16).



Estrutura interna do SOMADOR mostrando a interligação dos módulos internos FULL-ADDER iterativos e a numeração dos pinos do encapsulamento.

SOMADOR DE NÚMEROS DE 8 BITS OBTIDO A PARTIR DE SOMADORES DE NÚMEROS DE 4 BITS

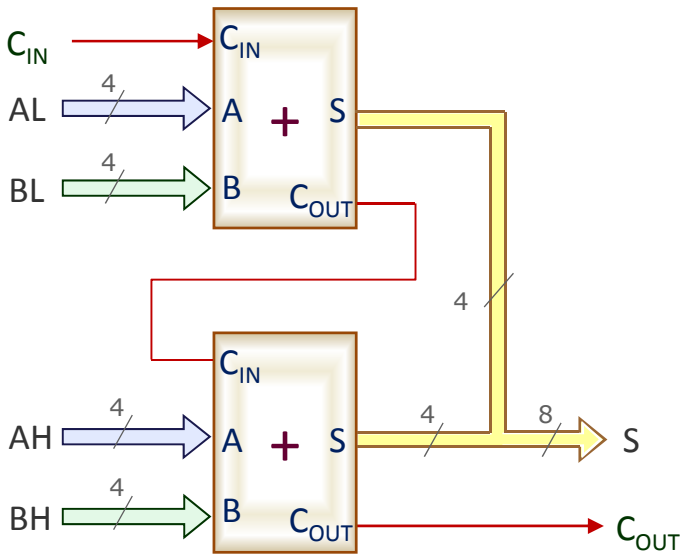


Diagrama de blocos de um SOMADOR DE NÚMEROS DE 8 BITS obtido pela concatenação de dois SOMADORES de NÚMEROS de 4 bits.

AL, BL – conjunto de 4 bits (NIBBLE) de menor peso de cada número .

AH, BH – conjunto de 4 bits de maior peso de cada número .

Os circuitos integrados prestam-se a uma concatenação modular.

A estrutura interna e os sinais de controlo que põem disponíveis, permitem ampliar o alcance funcional, por interligação iterativa, arborescente unidimensional ou matricial bidimensional. As Figs. mostram um SOMADOR de números 8 bits construído a partir de dois SOMADORES de números de 4 bits.

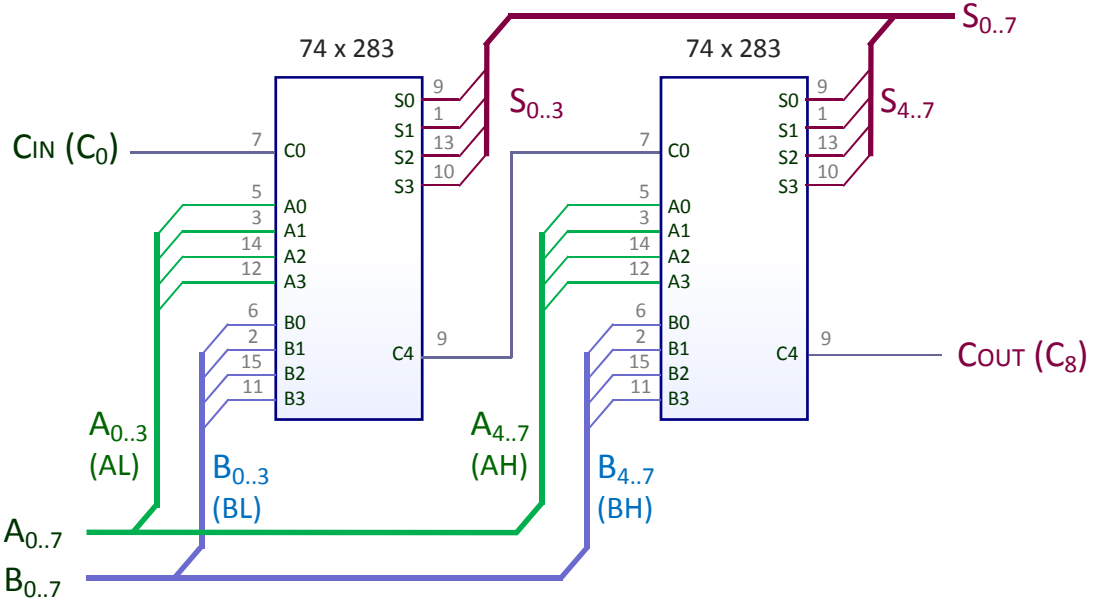


Diagrama lógico de um SOMADOR DE NÚMEROS DE 8 BITS obtido pela concatenação de dois SOMADORES de NÚMEROS de 4 bits.

Implementação de Funções Simétricas com Somadores (Ex. 5-6)

Exemplo 5-6

Os módulos somadores são estruturas adequadas à implementação de funções ditas Simétricas – por ser irrelevante quais das variáveis de entrada tomam o valor 1, interessando somente quantas tomam esse valor.

Considere-se a implementação de uma função de 7 variáveis, que se pretende que tome o valor lógico 1 quando o número de entradas activas for múltiplo de 3.

Considerando que cada entrada de A a G é um número de 1 algarismo, ao serem somadas todas as entradas, pode observar-se o resultado da soma a 3 bits dessas entradas S_2, S_1, S_0 , e verificar se esse valor é múltiplo de 3 (3 ou 6). Mostra-se a implementação da função, utilizando módulos Full Adder – somadores completos de 3 bits.

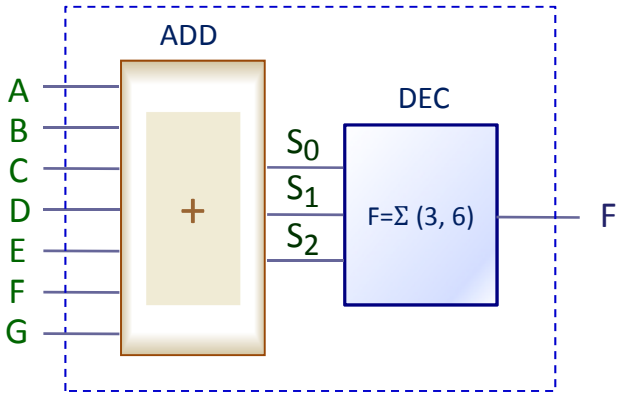
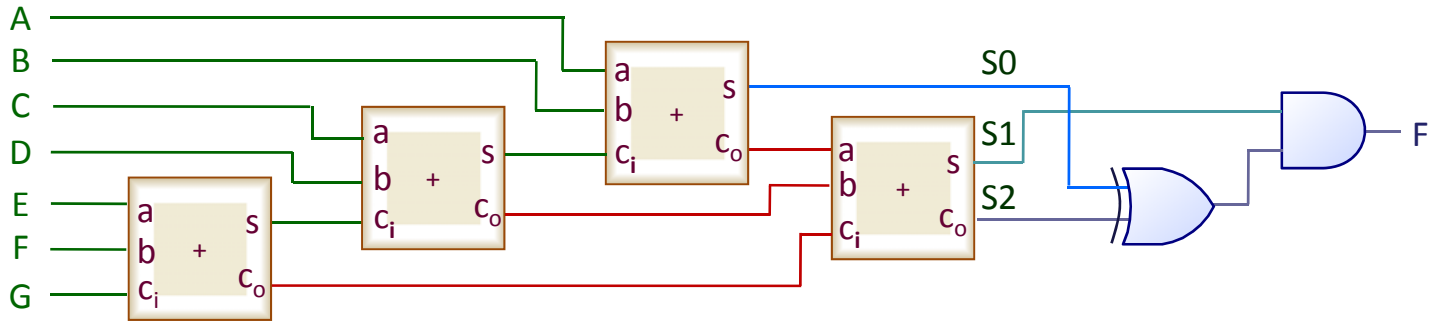


Diagrama de blocos da implementação de uma função F simétrica, de 7 variáveis A a G, que tem o valor lógico 1 quando o número de entradas activas for múltiplo de 3.

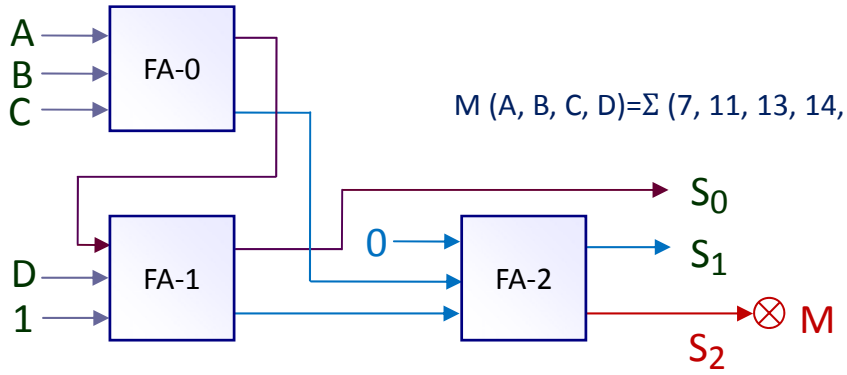


S_2	S_1	S_0	
0	1	1	3
1	1	0	6

Esquema lógico da implementação com módulos Full Adder isolados da função F a 7 variáveis 'número de entradas activas múltiplo de 3'.

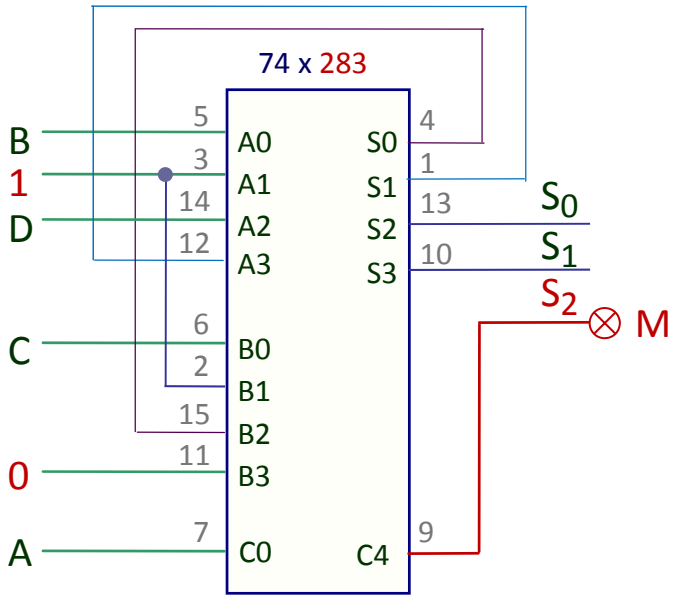
Implementação de Funções Simétricas com Somadores (Ex. 5-7)

Exemplo 5-7

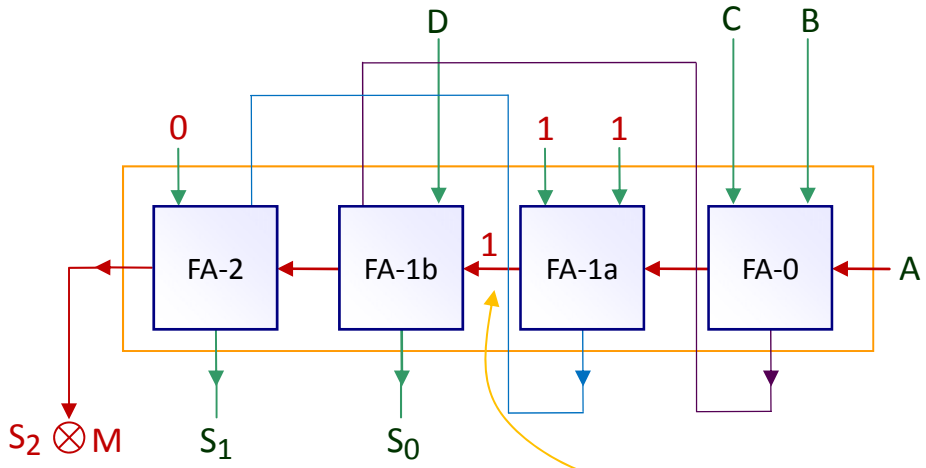


M (A, B, C, D)=Σ (7, 11, 13, 14, 15)

Esquema lógico da implementação da função M (Maioria) a 4 variáveis com 3 módulos Full Adder através da realização da soma A+B+C+D+1.



Esquema lógico da implementação de um Somador de Números de 4 bits da função M (Maioria) a 4 variáveis.



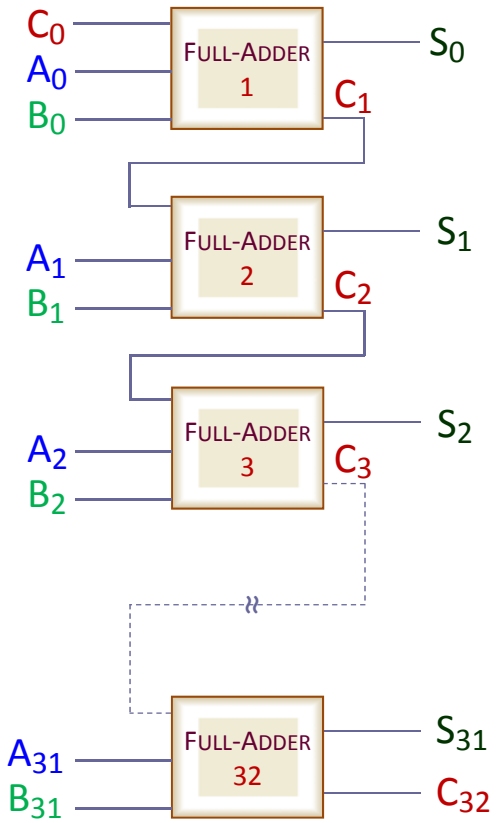
Esquema lógico da implementação da função M (Maioria) a 4 variáveis com 4 módulos Full Adder concatenados como no interior do circuito 74x283.

este 1 implica a soma A+B+C+D+1.

Pretende-se implementar a função Maioria a 4 variáveis, que toma o valor lógico 1 quando a maioria (3 ou 4) das entradas A, B, C e D tiver o valor 1.

Considerando que cada uma dessas entradas é um número de 1 algarismo, ao serem somadas todas as entradas, pode observar-se o resultado da soma a 3 bits dessas entradas S_2, S_1, S_0, e verificar se esse valor é 3 ou 4. Adicionando-se o valor fixo 1 ao resultado, obtém-se agora um resultado de 4 ou 5 o que simplifica a detecção pelo bit de maior peso (M=S_2). Mostra-se a implementação da função utilizando módulos Full Adder (à esquerda), e a sua realização com um único circuito Somador de Números de 4 bits (em cima).

Tempo de Propagação do Carry-out na Adição com Ripple-Carry Adder



Estrutura interna iterativa de um somador Ripple-carry de 32 bits evidenciando o caminho crítico ligado à propagação do Carry-out.

Na estrutura iterativa ao lado, a velocidade máxima de cálculo de uma soma de 2 números A e B de 32 algarismos é limitada pela necessidade de propagação do Carry-out entre os Full Adders (FA) desde o primeiro bit (C_1) até ao último (C_{33}).

Sendo FA_t o tempo de atraso imposto por cada módulo FA para a geração do Carry-out respectivo, o tempo de propagação do Carry-out será de 32 x FA_t o que se torna inaceitável em determinados contextos. Este tipo de somador é designado Ripple-Carry Adder (somador com propagação de Carry).

Exemplos:

Se **FA_t = 100 ps**, o atraso total é de **3200 ps**, e a frequência máxima do relógio limitada a

$1/(3200 \times 10^{-12})$ Hz = **312 MHz**.

O tempo de propagação (ou de atraso) através de uma porta típica tem diminuído linearmente com a evolução e o 'escalamento' tecnológico. Para uma tecnologia CMOS submicron de 0,13 micron o atraso típico τ por porta lógica é de **65 ps**.

Com um relógio de **1 GHz** (período de 1000 ps) virá então:

1000/65 ≈ 15 portas encadeadas no máximo por ciclo de relógio, o que é manifestamente insuficiente para a construção do somador de 32 bits aqui referido.

Existem várias técnicas para aceleração dos processos da adição, que conseguem diminuir o tempo total da realização de uma soma. Os slides seguintes referem a técnica denominada de **Lookahead** (utilizada na construção do circuito somador 74x283) que calcula o arrasto separadamente: aumenta-se a velocidade do circuito, mas também a sua complexidade. Mais à frente exemplifica-se sumariamente outra técnica, designada **Carry Select**.

ADIÇÃO COM RIPPLE-CARRY ADDER – BLOCO SOMADOR PARCIAL (1)

Revisita-se o somador completo, e introduz-se um bloco interno designado de somador parcial.

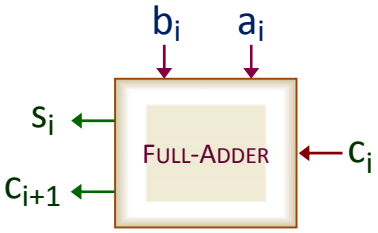
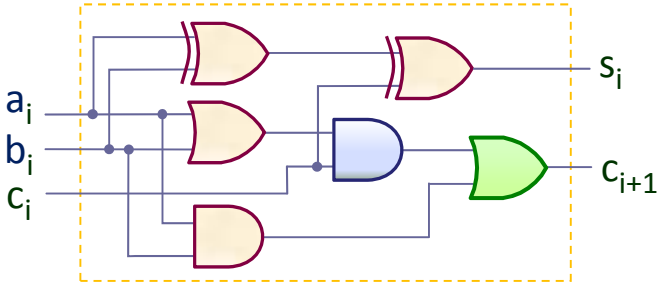


Diagrama de blocos de um andar SOMADOR COMPLETO em notação iterativa.



Estrutura interna de um andar SOMADOR COMPLETO (FULL-ADDER).

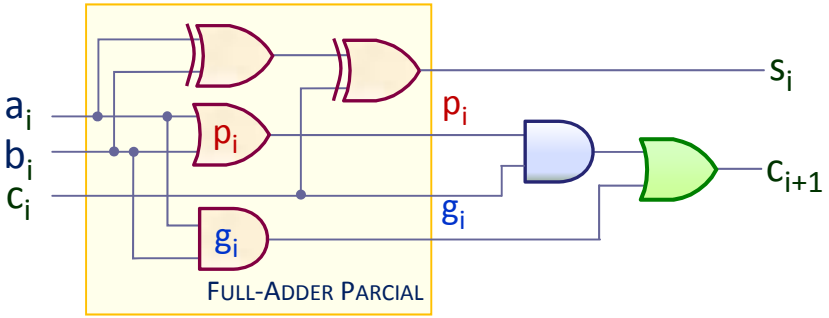
Propagador de Carry (pi) — Gerador de Carry (gi)

$$s_i = (a_i \oplus b_i) \oplus c_i$$

$$c_{i+1} = (a_i \oplus b_i)\, c_i + a_i\, b_i = (a_i + b_i)\, c_i + a_i\, b_i =$$

$$= p_i\, c_i + g_i$$

Equações da soma S_i e do Carry-out C_{i+1} evidenciando as funções GERADOR DE CARRY (g_i) e PROPAGADOR DE CARRY (p_i).



Estrutura interna de um andar SOMADOR COMPLETO com a demarcação do bloco designado de SOMADOR PARCIAL evidenciando as funções GERADOR DE CARRY (g_i) e PROPAGADOR DE CARRY (p_i).

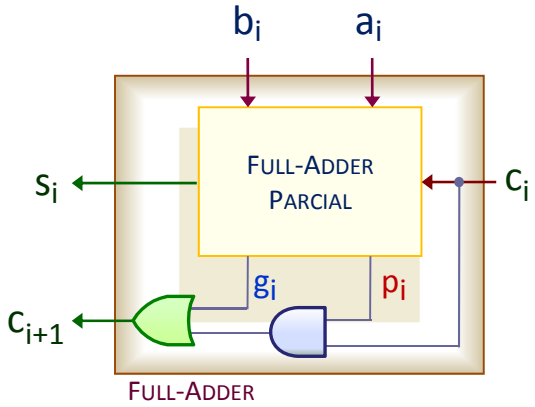


Diagrama de blocos de um andar SOMADOR COMPLETO construído a partir do módulo SOMADOR PARCIAL evidenciando as funções GERADOR DE CARRY (g_i) e PROPAGADOR DE CARRY (p_i) em notação iterativa.

ADIÇÃO COM RIPPLE-CARRY ADDER – BLOCO SOMADOR PARCIAL (2)

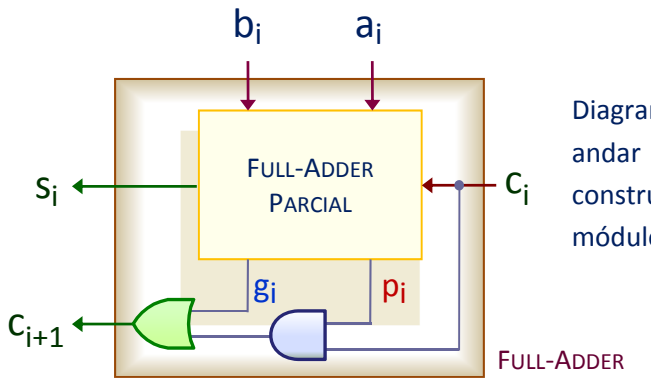


Diagrama de blocos de um andar SOMADOR COMPLETO construído a partir do módulo SOMADOR PARCIAL.

$$c_{i+1} = g_i + p_i\, c_i$$

$g_i = a_i\, b_i$

É designada função GERADOR DE CARRY e representa as condições em que o Carry-out de cada andar é gerado (posto a 1) independentemente do valor do Carry-in.

$p_i = a_i + b_i$

É designada função PROPAGADOR DE CARRY e representa as condições em que o Carry-out de cada andar é propagado através do somador a partir de um valor de Carry-in igual a 1. Pode ser calculado através do OR-inclusivo (a_i+b_i) entre a_i e b_i, ou alternativamente através do XOR entre a_i e b_i ($a_i \oplus b_i$), como visto no slide 2-24.

A chave para acelerar uma adição é determinar o Carry-in que se propaga para cada andar de ordem superior mais cedo.

Já foi dito que qualquer função pode ser implementada apenas em dois níveis de lógica. Como as únicas entradas externas de um somador são os operandos A e B e o Carry-in para o bit menos significativo, pode em teoria calcular-se cada Carry-out parcial de cada andar do somador apenas com dois níveis de lógica, função destas entradas A, B e Carry-in: começa-se por reescrever a equação do Carry-out como indicado no slide anterior e em cima: $c_{i+1} = g_i + p_i\, c_i$.

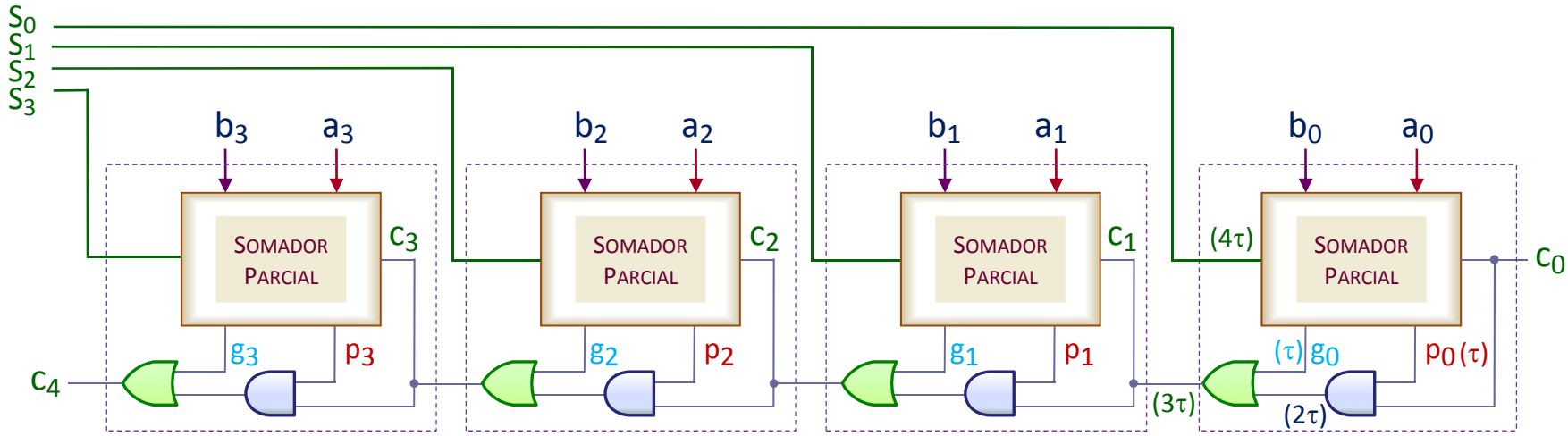
Esta equação significa pois que qualquer andar **i gera** incondicionalmente um Carry-out se ambas as parcelas a_i e b_i são 1 ($g_i = a_i\, b_i$), e **propaga** para o Carry-out um 1 se pelo menos uma dessas parcelas é 1 ($p_i = a_i + b_i$).

Isto significa que um andar produz um Carry-out sempre que esse andar gera um Carry, ou sempre que propaga um Carry se simultaneamente a entrada de Carry-in desse andar for 1.

Um somadorRIPPLE-CARRY ADDER (somador com propagação de Carry) de 4 bits pode então ser redesenhado como na Fig. do próximo slide.

Por razões de simplificação na modelação temporal considera-se para o que se segue que τ é o tempo de atraso imputável indistintamente a cada porta lógica simples AND ou OR, independentemente do seu número de entradas. Considera-se ainda que uma porta XOR contribuirá aproximadamente com um atraso de 2τ.

RIPPLE-CARRY ADDER A 4 BITS – ESTRUTURA INTERNA



Estrutura interna do somador de 4 bits do tipo clássico RIPPLE-CARRY ADDER evidenciando os SOMADORES PARCIAIS e as funções GERADOR DE CARRY (g_i) e PROPAGADOR DE CARRY (p_i).

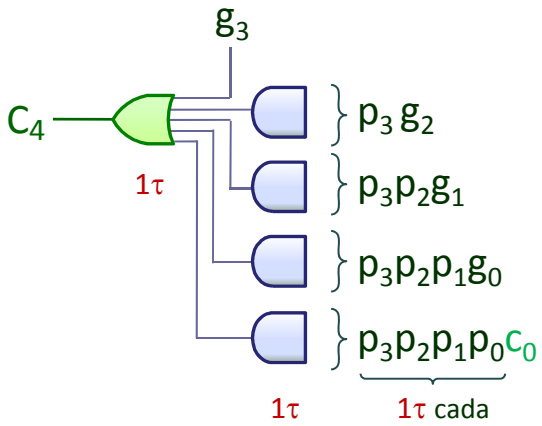
No SOMADOR COMPLETO clássico do tipo RIPPLE-CARRY que temos vindo a analisar, se a_i, b_i e c_0 estiverem disponíveis em t=0:

- Os sinais g_i e p_i são gerados simultaneamente com um atraso de τ em todos os andares (considera-se para este exemplo que uma porta simples do tipo AND, OR ou NOT contribui com um atraso τ, e uma porta XOR contribui aproximadamente com um atraso de 2τ).
- Cada sinal de saída S_i é gerado com um atraso típico de 4τ dentro do andar respectivo (correspondente a 2 portas XOR).
- Cada sinal de Carry-out c_i é gerado com um atraso típico 3τ dentro do andar respectivo (correspondente a 3 portas lógicas simples).
- Os atrasos inerentes à obtenção dos valores de S_i e c_{i+1} são proporcionais ao número de bits pois tem de se aguardar que sejam gerados todos os Carry-out de andares anteriores.
- O Carry-out C_4 será gerado com um atraso máximo de 9τ se não existir nenhuma contribuição de Geradores de Carry mas houver contribuição de todos os Propagadores de Carry; a saída S_3 será gerada também com um atraso de 9τ.
- Estes valores dizem apenas respeito a **CAMINHOS CRÍTICOS** que correspondem aos piores casos na propagação dos sinais – tipicamente o caminho crítico é o que atravessa mais portas lógicas.

Aceleração da Adição com Carry-Lookahead

Manipulando-se então as equações com o intuito de acelerar a adição do Ripple-carry adder representada no slide anterior, consegue-se que cada bit de transporte (Carry-out), parcial ou final, seja gerado directamente, sem que tenha de ser propagado de andar para andar. Na expressão finais de qualquer bit de Carry-out, parcial ou final, não aparecem os bits de Carry-out anteriores, só aparece o bit de carry-in c_0, e este fica imediatamente disponível, à custa de portas com maior número de entradas. O que é que significa, por exemplo, a equação de $c_2 = g_1 + p_1c_1 = g_1 + p_1g_0 + p_1p_0 c_0$? Que c_2 foi gerado diretamente no andar 1, ou que o andar 1 propagou um transporte c_1 gerado no andar 0, ou que os andares 1 e 0 propagaram o transporte Carry-in inicial c_0 (Carry-in). Assim, todos os transportes são gerados directamente na forma AND-OR sem a propagação inerente ao Somador Ripple-carry :

$$c_1 = g_0 + p_0 c_0$$
$$c_2 = g_1 + p_1 c_1 = g_1 + p_1 (g_0 + p_0c_0) = g_1 + p_1g_0 + p_1p_0 c_0$$
$$c_3 = g_2 + p_2 c_2 = g_2 + p_2 (g_1 + p_1g_0 + p_1p_0c_0) = g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0 c_0$$
$$c_4 = g_3 + p_3 c_3 = g_3 + p_3 (g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0) = g_3 + p_3 g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0 c_0$$



Geração de C4 com um atraso máximo total 3τ correspondente ao atraso de 3 portas lógicas standard, incluindo um AND de 5 entradas e também um OR de 5 entradas.

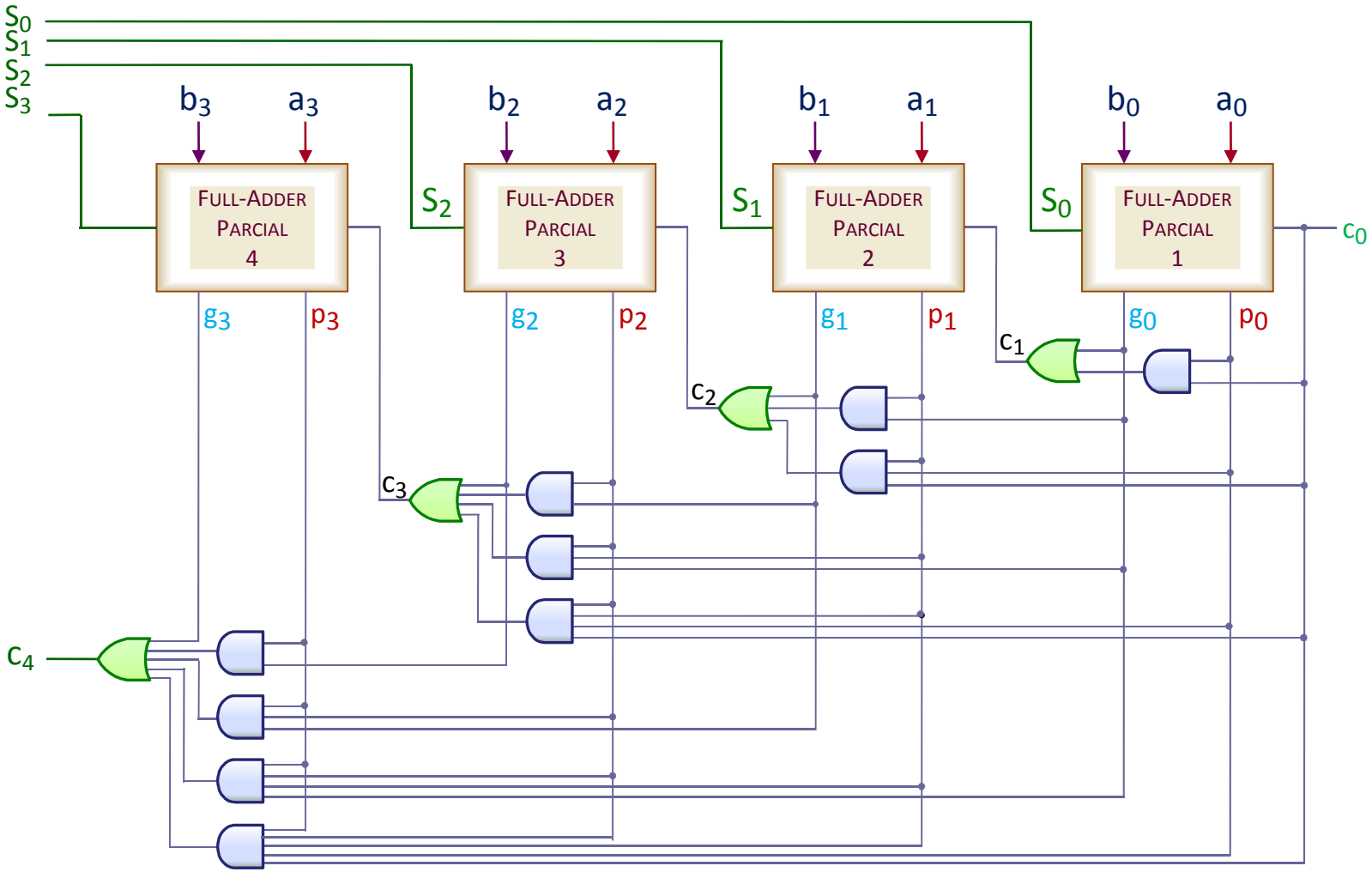
O Carry-out C_4 será agora gerado com um atraso total de apenas 3τ em vez do atraso de 9τ verificado no caso anterior do somador Ripple-carry. O somador implementado desta maneira é designado de Lookahead-carry e a sua estrutura interna está no slide seguinte.

Porque não prosseguir da mesma forma até aos 16 bits, ou até aos 64 bits?

Porque as equações dos Carry-out intermédios vêm muito complexas: para calcular os Carry-out de ordem elevada são necessárias portas lógicas com muitas entradas: C_4 ao lado corresponde a um OR de 5 termos AND respetivamente de 2, 3 , 4 e 5 entradas.

C_{n+1} seria gerado fazendo uso de algumas portas de n+1 entradas. Isto torna-se pouco prático de implementar. Ma abordagem mais realista recorre a soluções de compromisso fazendo uso de portas de menor número de entradas.

CARRY-LOOK AHEAD ADDER A 4 BITS – ESTRUTURA INTERNA



Estrutura de um somador de 4 bits do tipo CARRY-LOOKAHEAD (somador com antecipação de Carry). As funções $\mathbf{g_i}$ e $\mathbf{p_i}$ podem ser calculadas em paralelo (ao mesmo tempo em todos os andares) pois só dependem de a_i e b_i , o que origina que qualquer bit de transporte incluindo C_4 seja gerado com um atraso de 3τ (comparado com 9τ no somador RIPPLE-CARRY) pois só depende de C_0 e não do bit de transporte anterior; qualquer saída será gerada com um atraso de 5τ.

Carry-Lookahead Adder a 16 Bits – Super Funções de Grupo

Para somadores de maior número de bits o que se faz então é utilizar um somador de 4 bits como bloco construtor do somador maior, e concatenar os blocos obtidos na forma clássica de Ripple-carry. Isto origina a formação das Super Funções de Grupo, como se exemplifica ao lado para o grupo 0 (o Carry-out deste grupo é agora denominado C_1, correspondendo C_1 ao c_4 do somador Lookahead-carry de 4 bits).

- Gerador de Carry de Grupo $\mathbf{G_i}$ e
- Propagador de Carry de Grupo $\mathbf{P_i}$ em cada grupo.

Cada P_i é especificado num único nível de lógica AND usando p_i e cada G_i é especificado em dois níveis de lógica AND-OR usando p_i e g_i. Exemplificando para o primeiro grupo:

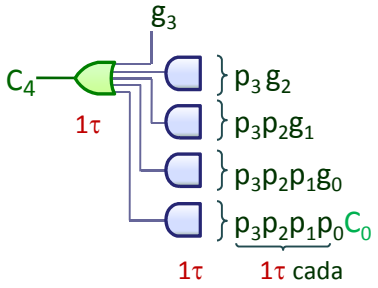
$$C_1 = c_4 = (g_3 + p_3 g_2 + p_3p_2g_1 + p_3p_2p_1g_0) + p_3p_2p_1p_0\, c_0 =$$

$$= G_{0\text{-}3} + P_{0\text{-}3}\, c_0 =$$

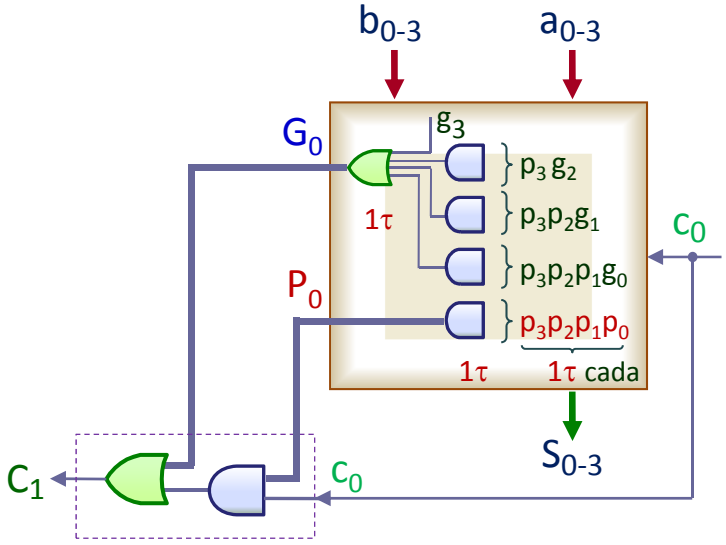
$$= G_0 + P_0\, c_0$$

P_0 é a função Propagador de Carry de grupo para o primeiro grupo. P_0 é 1 quando C_1 vem propagado através do primeiro grupo de 4 bits.

G_0 é a função Gerador de Carry de Grupo para o primeiro grupo. G_0 é 1 quando um Carry-out for gerado internamente no primeiro grupo, independentemente do Carry de entrada C_0.



Geração de C_4 no somador em Carry-Lookahead de 4 bits do slide anterior.



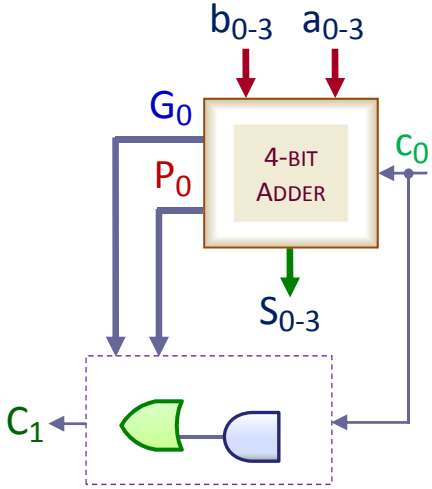


Diagrama de blocos do primeiro andar de 4 bits de um somador Carry-Lookahead de 16 bits evidenciando as Super Funções de Grupo G0, P0 e C1 – à esquerda em detalhe e à direita de forma simplificada.

CARRY-LOOKAHEAD ADDER A 16 BITS – ESTRUTURA INTERNA

Considera-se o somador CARRY-LOOKAHEAD de 4 bits como o bloco construtor do somador de 16 bits.

Podem ligar-se 4 somadores CARRY-LOOKAHEAD de 4 bits em modo RIPPLE-CARRY para formar o somador de 16 bits, mas para aumentar a velocidade pode repetir-se a mesma ligação em lógica CARRY-LOOKAHEAD usada no interior de cada bloco individual de 4 bits, mas desta vez usando os supergrupos P_i e G_i .

Para o Carry-out C_4 do andar correspondente aos bits a_{15} e b_{15} virá:

$$C_4 = c_{15} = G_3 + P_3C_3 =$$
$$= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0c_0$$

Cada P_i é especificado através de um nível de lógica AND usando p_i, cada G_i é especificado através de dois níveis de lógica AND-OR usando p_i e g_i – p_i e g_i eles próprios possuem apenas um nível de lógica em termos de a_i e b_i, o que conduz a um atraso de 5τ na obtenção do Carry-out C_4 à saída do andar dos bits a_{15} e b_{15} após a aplicação simultânea dos sinais $a_{0..15}$, $b_{0..15}$ e c_0 às entradas do somador.

Para o somador RIPPLE-CARRY de 16 bits do slide-24, para o qual se assume um atraso de 2τ para a geração de cada Carry-out parcial, o Carry-out do andar 15 será gerado fim de $16\times2\tau = 32\tau$, o que mostra que a adição pelo método CARRY-LOOKAHEAD é cerca de 6 vezes mais rápida que a adição pelo método RIPPLE-CARRY.

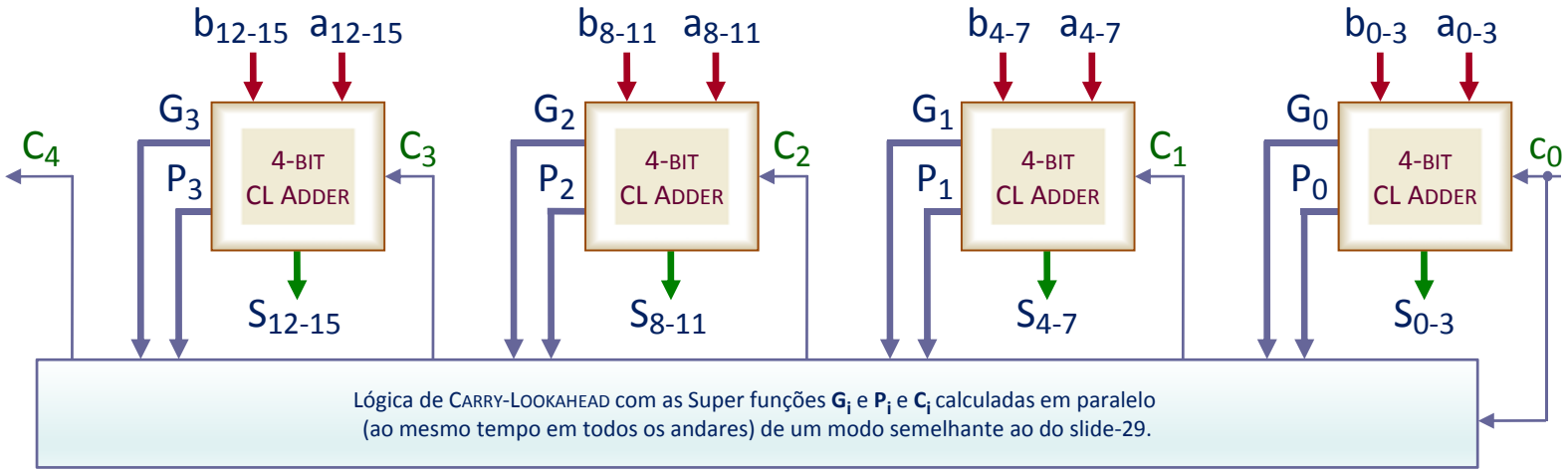
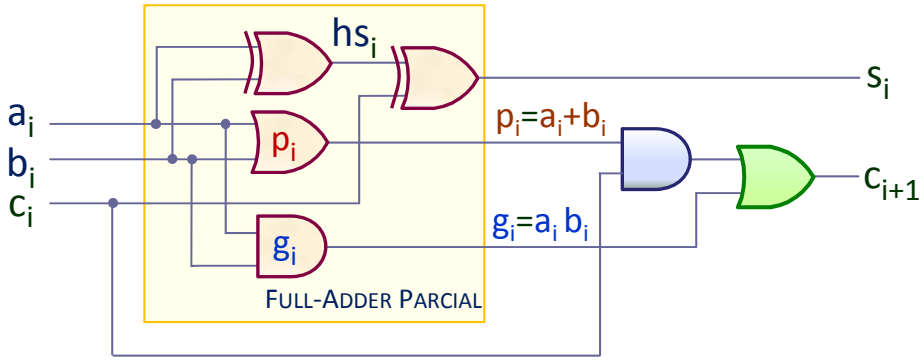


Diagrama de blocos de um somador CARRY-LOOKAHEAD de 16 bits formado por 4 somadores CARRY-LOOKAHEAD de 4 bits cada ligando os sinais de SUPER GRUPO PI E GI de cada um com lógica de antecipação de Carry semelhante à utilizada no interior de cada um dos 4.

74 x 283 – 4 bit binary Carry-Lookahead Adder (1)



Somador Completo evidenciando as funções g_i (Gerador de Carry) e p_i (Propagador de Carry).

$$hs_i = a_i \oplus b_i = a_i\, b_i' + a_i'\, b_i =$$
$$= a_i\, b_i' + a_i\, a_i' + a_i'\, b_i + b_i\, b_i' =$$
$$= (a_i + b_i)\,(a_i' + b_i') =$$
$$= p_i\, g_i'$$

$$s_i = hs_i \oplus c_i$$

$$c_{i+1} = g_i + p_i\, c_i = p_i\, g_i + p_i\, c_i = p_i\,(g_i + c_i)$$

Equações da Meia-soma hs_i, Soma S_i e do Carry-out C_{i+1} de um Somador Completo.

O somador 74x283 acelera a adição com a técnica Carry-Lookahead. A sua estrutura interna apresenta algumas alterações internas face ao circuito tradicional (lado esquerdo).

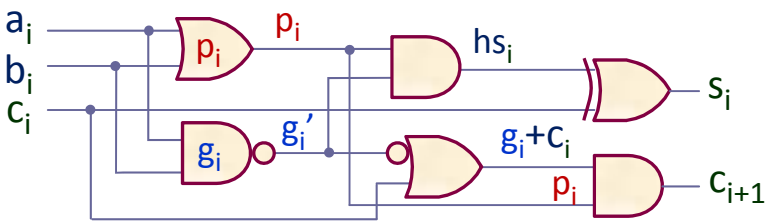
Tira-se por exemplo partido da manipulação algébrica (patente ao lado) da equação do XOR para a Meia-soma hsi (Half-sum) que leva a que:

$$hs_i = p_i\, g_i'$$

Para além destas diferenças o somador 74x283 tira partido do facto de ser:

$$c_{i+1} = g_i + p_i\, c_i = p_i\, g_i + p_i\, c_i = p_i\,(g_i + c_i)$$

A equação pode escrever-se deste modo porque p_i é sempre 1 quando g_i é 1.



Somador Completo evidenciando alterações da estrutura interna por manipulação das equações da Soma e do Carry-out.

74 x 283 – 4 bit binary Carry-Lookahead Adder (2)

Na arquitetura interna do somador 74x283 são utilizadas versões ACTIVE-LOW das funções gi e pi porque as portas inversoras são em geral mais rápidas.

Para a meia soma hsi utiliza-se em vez da tradicional porta XOR uma porta AND com entrada invertida como mostrado porque ocupa menos espaço e é mais rápida.

Tendo em conta o teorema da distributividade na forma dual: *a + bc = (a+b) (a+c)* (slide1-18) podem reescrever-se as equações do Carry-out utilizadas no circuito somador da forma OR-AND:

$$c_1 = p_0 (g_0+c_0)$$

$$c_2 = p_1 (g_1+c_1) = p_1 (g_1+ p_0 (g_0+c_0)) = p_1 (g_1+p_0)(g_1+g_0+c_0)$$

$$c_3 = p_2 (g_2+c_2) = p_2 (g_2+ p_1 (g_1+p_0) (g_1+g_0+c_0)) = p_2 (g_2+p_1)(g_2+g_1+p_0) (g_2+g_1+g_0+c_0)$$

$$c_4 = p_3 (g_3+c_3) = p_3 (g_3+ p_2 (g_2+p_1)(g_2+g_1+p_0) (g_2+g_1+g_0+c_0)) = p_3 (g_3+p_2)(g_3+g_2+p_1)(g_3+g_2+g_1+p_0) (g_3+g_2+g_1+g_0+c_0)$$

Equações do CARRY-OUT do CIRCUITO SOMADOR COMPLETO 74x283 utilizadas na sua estrutura interna.

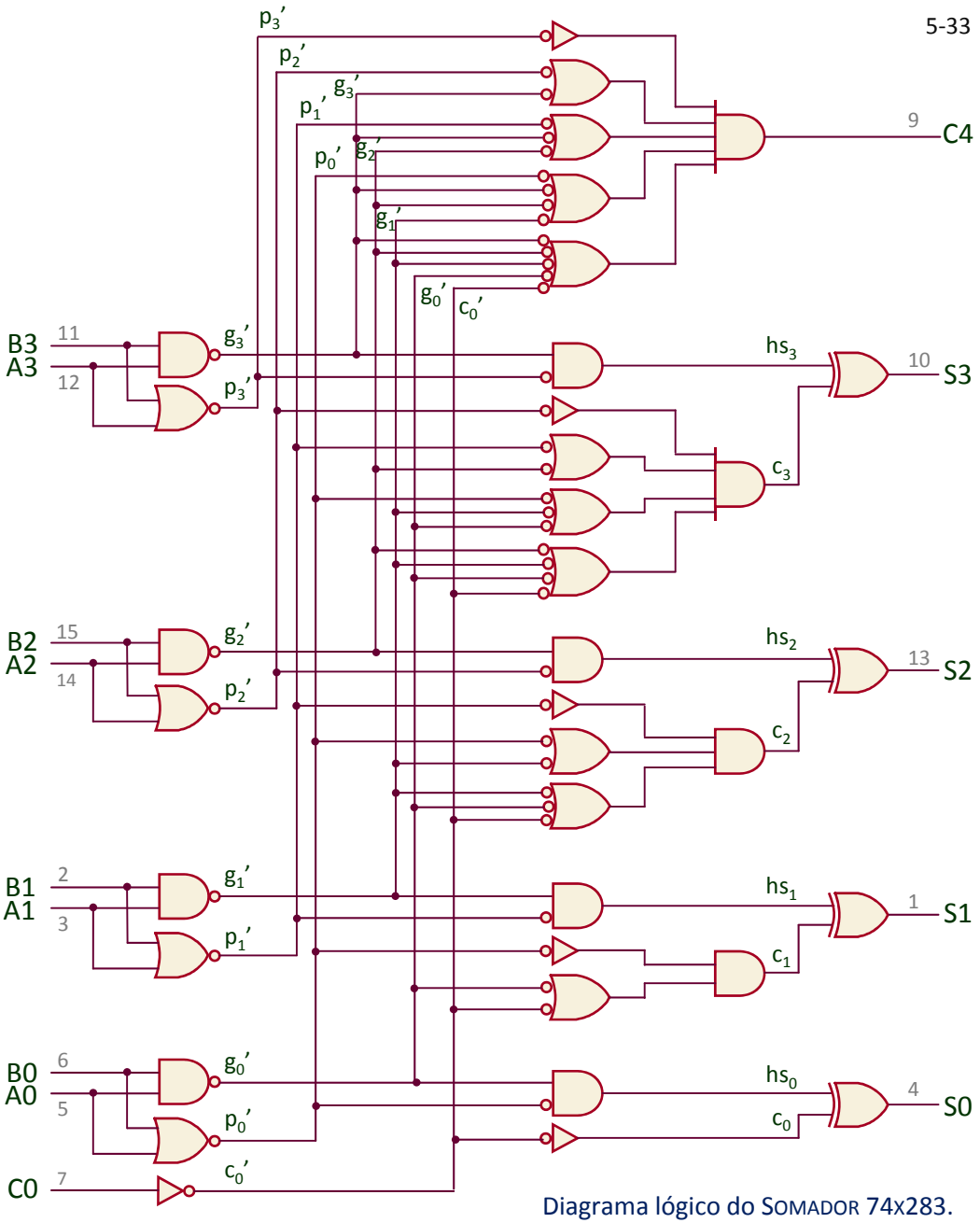
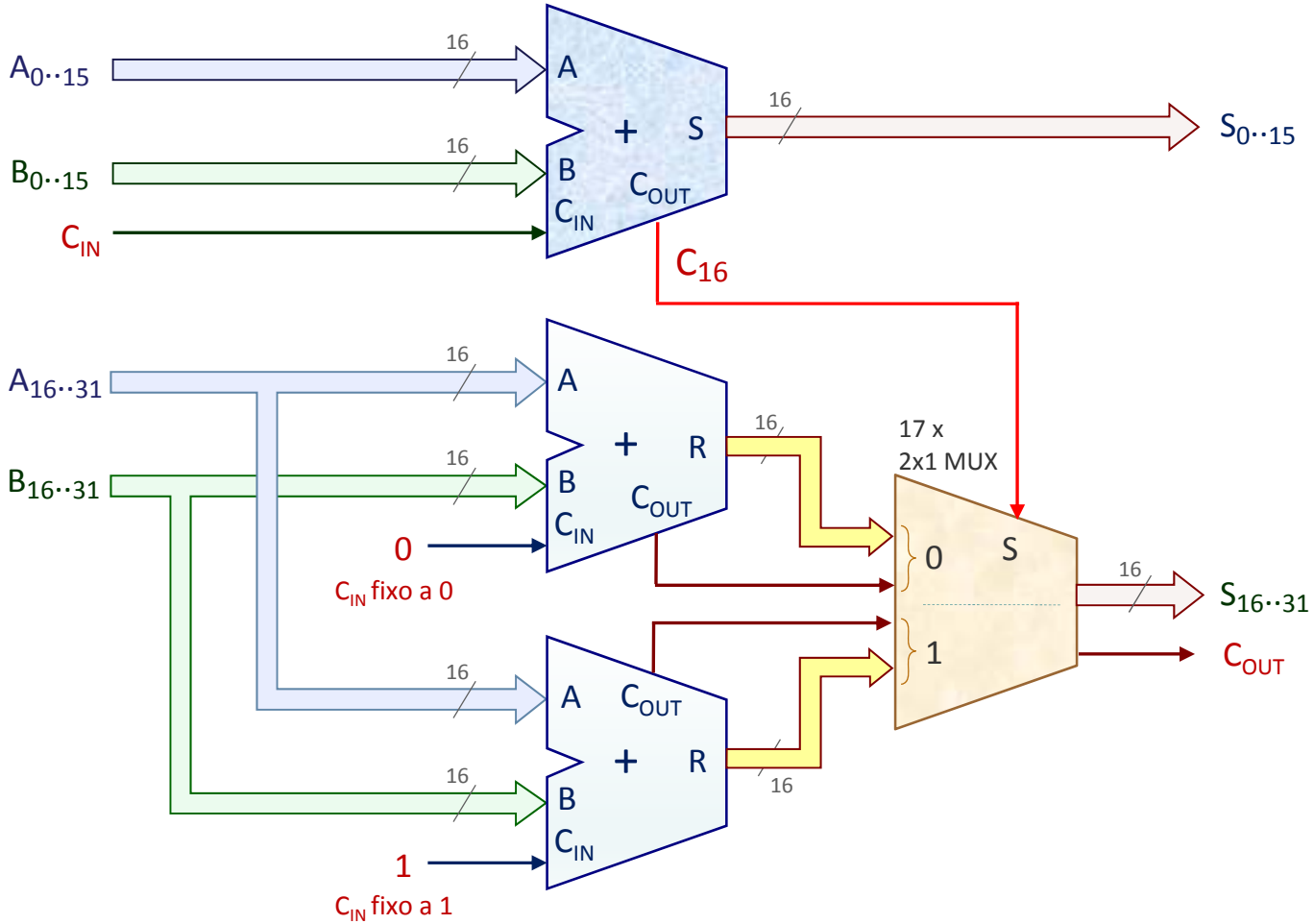


Diagrama lógico do SOMADOR 74x283.

Instituto Superior de Engenharia de Lisboa

ACELERAÇÃO DA ADIÇÃO COM CARRY-SELECT

Outra técnica de aceleração da adição é denominada CARRY-SELECT e utiliza hardware redundante. Consiste em gerar simultaneamente e em antecipação ambas as somas parciais dos bits de maior peso de A e B – neste exemplo da adição de 32 bits, a soma dos bits $A_{16..31}$ com os bits $B_{16..31}$ – para as hipóteses do Carry-out C_{16}, resultante da soma dos bits de menor peso $A_{0..15}$ com $B_{0..15}$, ser 0 ou 1.



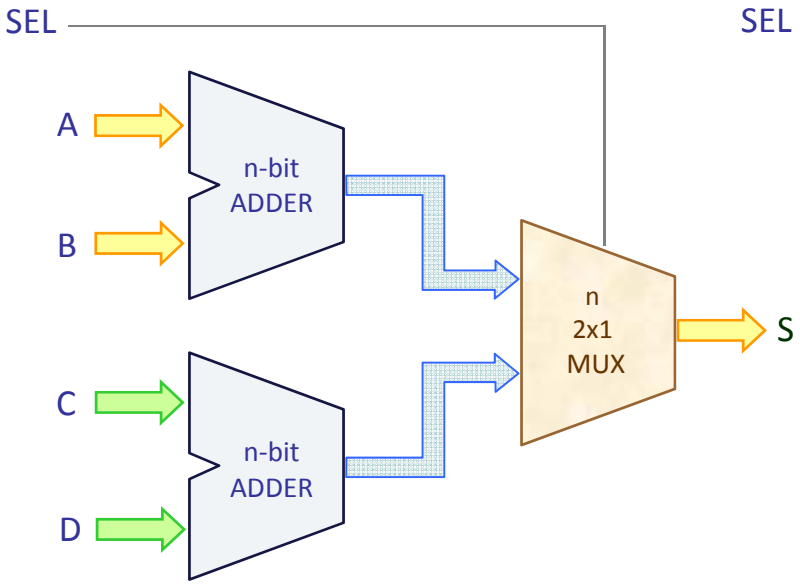
Estrutura de um somador de 32 bits do tipo CARRY-SELECT ADDER.

O bit C_{16} ao ser recebido fará a selecção da soma correcta através da ligação à entrada de controle do Multiplexer 17x1 de saída, que apresentará esse resultado nas saídas $S_{16..31}$.

No caso dos somadores serem do tipo RIPPLE-CARRY como no slide 5-24 o tempo de propagação do Carry-out será encurtado para um valor 16 x FA_t ao qual há que acrescer o tempo de propagação inerente ao Multiplexer de saída.

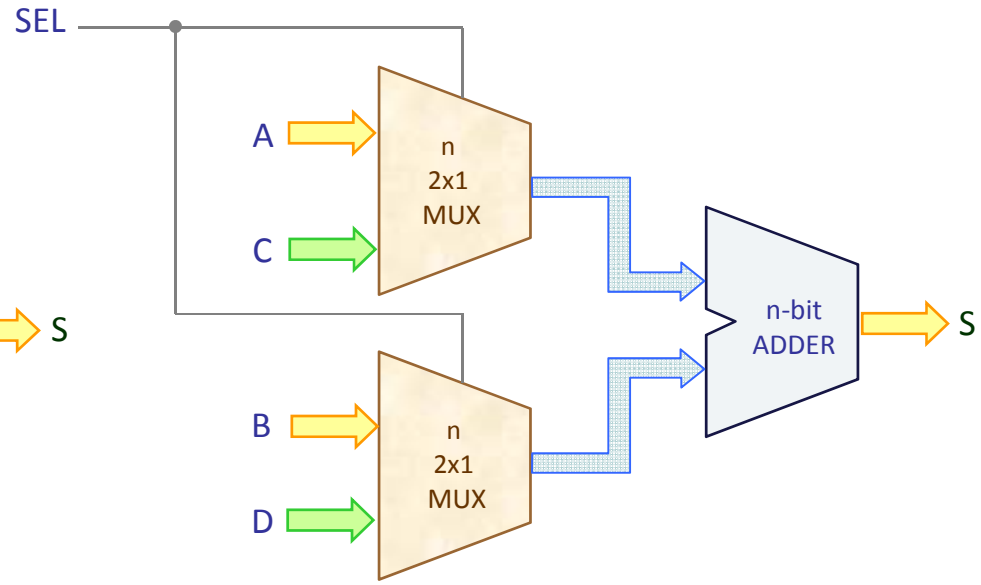
ADIÇÕES SELECTIVAS – CIRCUITOS ALTERNATIVOS

A, B, C e D são 4 operandos a n bits, e há que apresentar na saída S a soma do par (A,B), ou alternativamente do par (C,D), consoante o valor do bit de selecção (SEL) é 0 ou 1.



Circuito de selecção numa saída S de uma de duas somas (A+B ou C+D) utilizando dois SOMADORES (ADD) de números de n-bits e um MULTIPLEXER (MUX) de 2 entradas a n-bits.

O circuito da direita, sintetizado com um único SOMADOR e dois MULTIPLEXERS, corresponde a uma realização de maior simplicidade, pois um multiplexer de 2 entradas a n-bits requer um menor número de portas lógicas para a sua implementação do que um somador de números de n-bits.



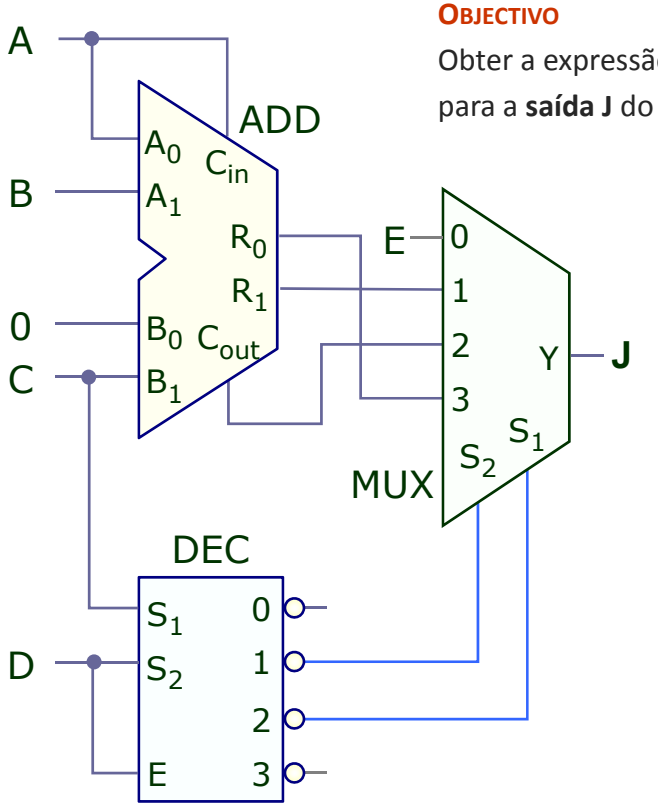
Circuito de selecção numa saída S de uma de duas somas (A+B ou C+D) utilizando apenas um SOMADOR (ADDER) de números de n-bits e dois MULTIPLEXER (MUX) de 2 entradas a n-bits que selecciona o par de OPERANDOS (A,B) ou (C,D).

A maior parte dos compiladores de linguagens do tipo HARDWARE DESCRIPTION LANGUAGE (**HDL**) utiliza a solução 'inteligente' do lado direito.

Na implementação de ALUs (mais à frente), esta problemática ganha relevância.

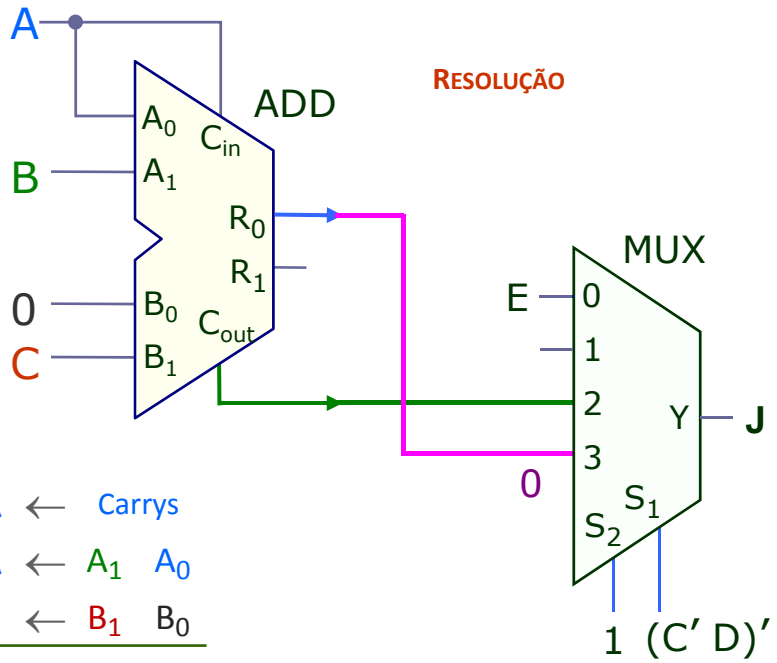
CIRCUITOS CONSTITUÍDOS POR MÓDULOS MSI COMBINATÓRIOS (EX. 5-8)

Exemplo 5-8



OBJECTIVO

Obter a expressão simplificada para a **saída J** do circuito.



RESOLUÇÃO

	A	A	←	Carrys
	B	A	← A_1	A_0
+	C	0	← B_1	B_0
C_{OUT}	R_1	0	←	Resultado

Operação SOMA efectuada pelo Somador (ADD).

$$J = C' D C_{OUT} + 0 = C' D \underbrace{(AB + AC + BC)}_{C_{OUT}} =$$

$$= A B C' D$$

Expressão simplificada de J.

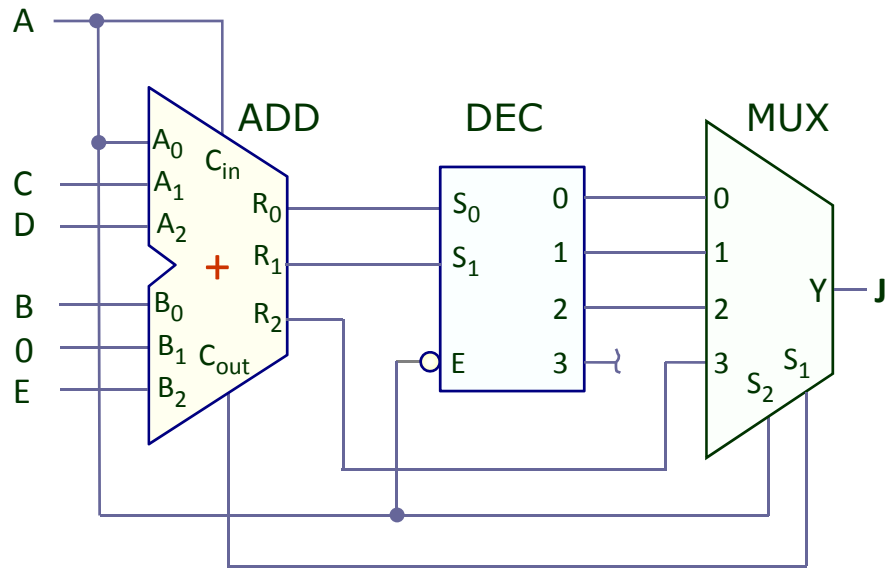
Com D=0 o DEC estará DISABLED e será selecionada a entrada 3 do MUX que levará $J=R_0=0$. Logo terá de ser D=1 que com C=0 fará com que $J=C_{OUT}$, sendo C_{OUT} a função Maioria de A, B e C.

		Saídas DEC				Entrada MUX
D	C	0_L	1_L	2_L	3_L	seleccionada
0	0	1	1	1	1	$3 - R_0$
0	1	1	1	1	1	$3 - R_0$
1	0	1	1	0	1	$2 - C_{out}$
1	1	1	1	1	0	$3 - R_0$

Tabela funcional do Decoder (DEC) assinalando os valores das saídas 1_L e 2_L do tipo ACTIVE-LOW.

CIRCUITOS CONSTITUÍDOS POR MÓDULOS MSI COMBINATÓRIOS (EX. 5-9-1)

Exemplo 5-9



OBJECTIVO

Obter a expressão simplificada para a **saída J** do circuito constituído por um SOMADOR de números de 3 bits (ADD), um DECODER 2x4 DEC), e um MULTIPLEXER 4x1 (MUX).

RESOLUÇÃO

	AC	A	A	←	Carrys		
	D	C	A	←	A_2	A_1	A_0
+	E	0	B	←	B_2	B_1	B_0
C_{OUT}	R_2	R_1	R_0	←	Resultado		

Operação Soma efectuada pelo Somador (ADD).

Distinguem-se as situações de A=0 (Decoder **enabled** ou activo), e de A=1 (Decoder **disabled** ou inactivo).

Com A=1 para se obter J=1 (entrada 3 do MUX A 1) terá que ser R_2=1 e C_{OUT}=1, o que implica C=D=E=1.

Com A=0 para se obter J=1 (entradas 0 ou 1 do MUX a 1) terá que ser R_1=0 o que implica C=0.

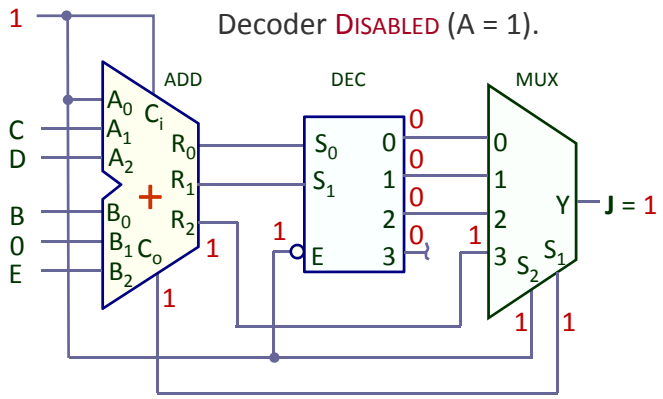
Se B=0 terá de ser C_{OUT}=0 e D e E não podem ser ambos 1.

Se B=1 terá de ser C_{OUT}=1 e D e E terão ambos de ser 1.

CIRCUITOS CONSTITUÍDOS POR MÓDULOS MSI COMBINATÓRIOS (EX. 5-9-2)

Exemplo 5-9

Decoder DISABLED (A = 1).

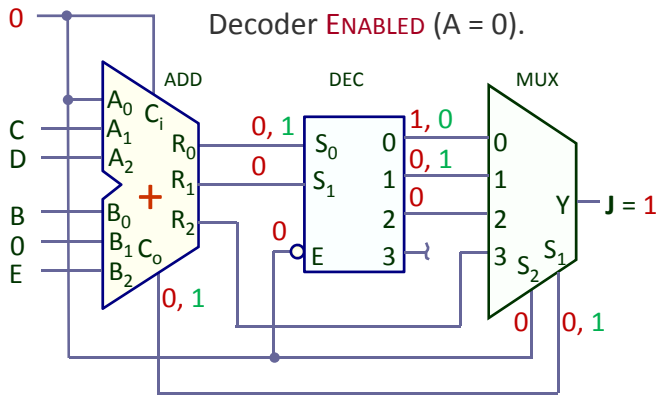


A	C_{OUT}	MUX Entrada seleccionada	J
1	0	2	0
1	1	3	R_2

C 1 1 ← Carrys
D C 1 ← A_2 A_1 A_0
\+ E 0 B ← B_2 B_1 B_0
1 1 C' B ← Resultado

⇒ ACDE

Decoder ENABLED (A = 0).



A	C_{OUT}	MUX Entrada seleccionada	J
0	0	0	$R_0'R_1'$
0	1	1	$R_0 R_1'$

0 0 0 ← Carrys
D C 0 ← A_2 A_1 A_0
\+ E 0 B ← B_2 B_1 B_0
0 D⊕E 0 0 ← Resultado

⇒ A'B'C'(DE)'

0 0 0 ← Carrys
D C 0 ← A_2 A_1 A_0
\+ E 0 B ← B_2 B_1 B_0
1 0 0 1 ← Resultado

⇒ A'BC'DE

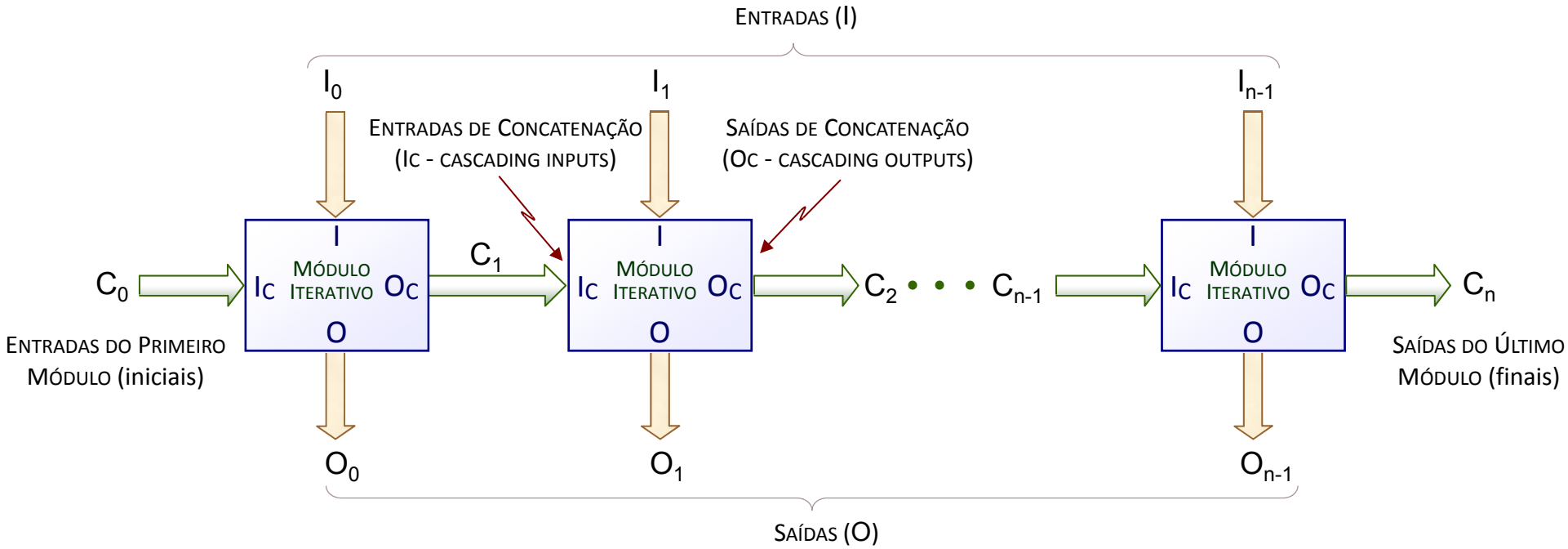
RESOLUÇÃO (CONTINUAÇÃO)

As configurações de variáveis que em cada caso conduzem ao valor 1 na saída J do Multiplexer estão explicitadas ao lado, e permitem obter a expressão final de J.

$$J = ACDE + A'B'C'(DE)' + A'BC'DE = ACDE + A'C' [(B'(DE)' + BDE] = ACDE + A'C'(B \oplus DE)'$$

Equação lógica da saída J.

Circuito Iterativo Combinatório Unidimensional - Iteração no Espaço



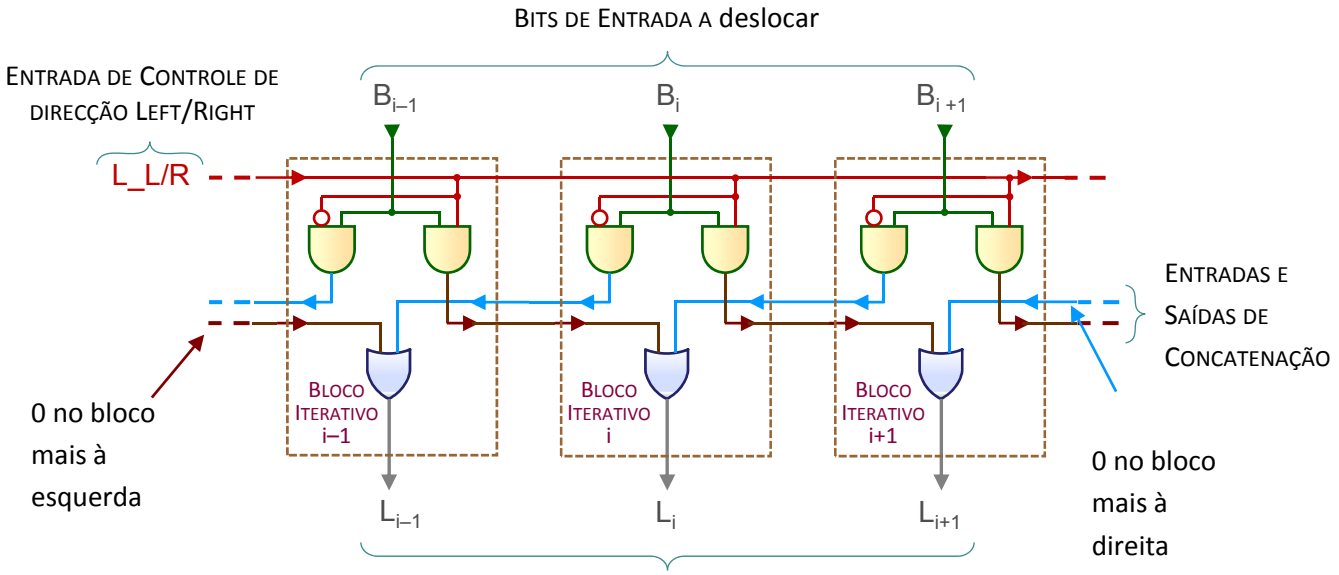
Circuito iterativo espacial combinatório composto por n módulos idênticos, cada um com entradas e saídas principais e ainda com entradas e saídas para ligação em cascata (Cascading ou Boundary inputs) - as entradas dos extremos estão normalmente ligadas a valores fixos.

A forma mais simples de um circuito iterativo espacial é a de uma ligação em cascata (iteração horizontal) de n-módulos (células) idênticos. Existem 1 ou mais sinais de entrada I_i (ditos primários), 1 ou mais sinais de saída principais (ditos primários), e 1 ou mais sinais para ligação em cascata e diálogo com os módulos adjacentes (ditos de concatenação). Os sinais de concatenação viajam numa direção e transportam a informação sobre o 'estado' de uma célula para a seguinte.

Uma solução iterativa torna-se interessante quando o número de entradas é elevado e o procedimento associado a cada entrada é recorrente. Exemplos típicos de estruturas iterativas são o somador (já analisado) e o comparador (analisado nos slides seguintes – o comparador, ao contrário do somador, não possui saídas primárias). A estrutura regular de um circuito iterativo torna-o mais simples de fabricar do que um circuito não iterativo.

Circuito Iterativo Combinatório de Deslocamento de 1 bit (Ex. 5-10-1)

Exemplo 5-10



Circuito Iterativo combinatório de deslocamento de 1 bit.

Bits de Saída deslocados de uma posição à direita (se L_L/R=1), ou à esquerda (se L_L/R=0)

Quando L_L/R=1 a porta AND do lado direito (em cima) do bloco iterativo i passa o bit B_i para a entrada da porta OR do bloco i+1 à sua direita.

Essa porta OR tem nesse momento a sua outra entrada a zero, o que permite realizar o deslocamento do bit B_i de uma posição para a direita: $L_{i+1} = B_i$.

Idem para o caso inverso. Considera-se que para cada deslocamento é feita a entrada de um 0 pelos módulos dos extremos.

Objectivo

Desenhar a estrutura interna e a expressão simplificada para a **saída Li** de cada módulo de um circuito combinatório iterativo que executa o deslocamento de um bit de entrada **Bi**.

O deslocamento será de uma posição para a esquerda ou para a direita consoante o valor lógico de uma entrada de controle L_L/R como indicado:

L_L/R =0 deslocamento à esquerda.
L_L/R =1 deslocamento à direita.

Resolução

Usando a convenção i+1 está à direita de i, vem:

$$L_i = L\_L/R' \cdot B_{i+1} + L\_L/R \cdot B_{i-1}$$

Equação algébrica das saídas.

Implementação a 5 Bits do Circuito Iterativo para Deslocamento de 1 bit (Ex. 5-10-2)

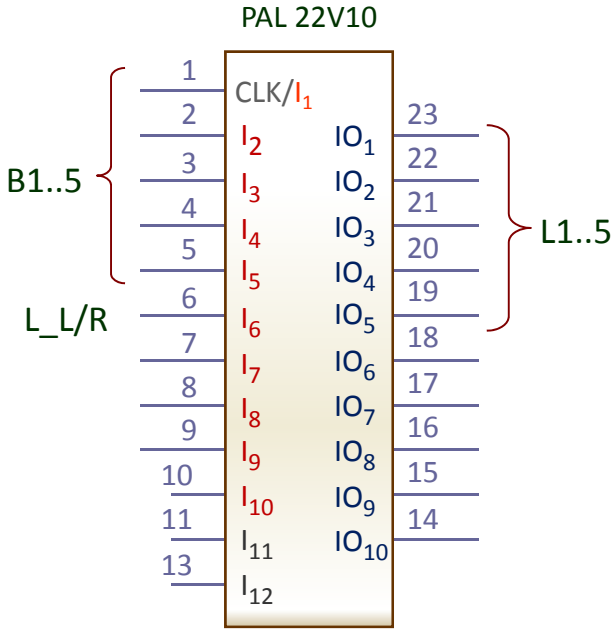
Exemplo 5-10

Objectivo

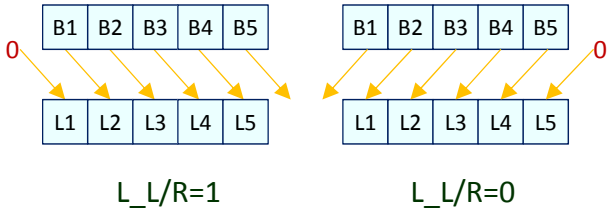
Implementar em PAL 22V10 o circuito iterativo do slide anterior para 5 pares de bits entrada-saída considerando que por cada deslocamento (à esquerda ou à direita) é feita a entrada de um 0.

```
Device   p22v10  ;
/* ***** Input Pins ****** **** */
PIN [1..5] = [B1..5 ] ;/*Bits a deslocar*/
PIN  6     = LLR       ;/*Controle de direcção*/

/* ***** Output Pins ********** */
PIN [23..19] = [L1..5] ;/*Bits deslocados*/

/* Equação de iteração e valores dos extremos */
$REPEAT i = [1.. 5]
L{i} =  !LLR & B{i+1} # LLR & B{i-1};
$REPEND
B0 = 'b'0 ;
B6 = 'b'0 ;
```

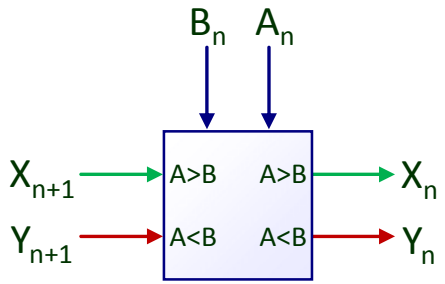
Troço do **Código CUPL** com a atribuição dos pinos de entrada e saída e a equação de iteração geradora do deslocamento.



Símbolo lógico da PAL (em cima) assinalando os pinos de entrada correspondentes aos bits Bi a deslocar, ao controle de direção L_L/R, e aos bits de saída Li já deslocados de uma posição para a esquerda ou direita em relação aos bits de entrada (em baixo).



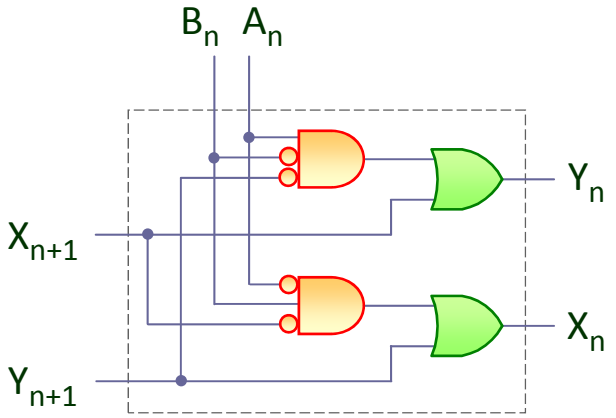
COMPARADOR COMO CIRCUITO ITERATIVO



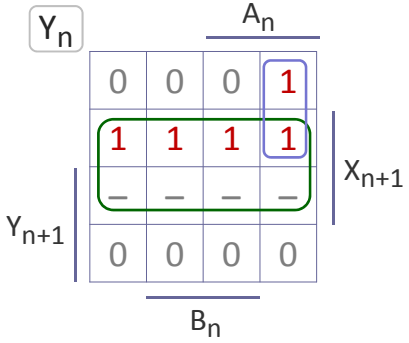
Módulo COMPARADOR de dois bits A_n e B_n.

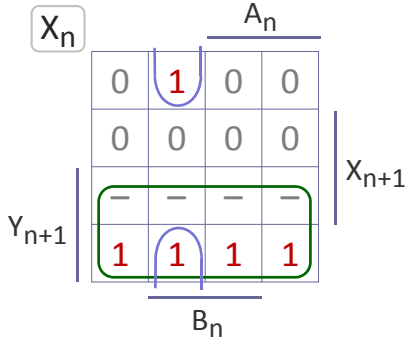
X_{n+1}	Y_{n+1}	A_n	B_n	X_n	Y_n
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	–	–	0	1
1	0	–	–	1	0
1	1	–	–	–	–

Tabela de verdade simplificada da comparação de dois bits A_n e B_n.



Realização do módulo iterativo da comparação com portas lógicas AND, OR e NOT.





Mapas de Karnaugh e equações lógicas das saídas Xn e Yn.

$$Y_n = X_{n+1} + A_n \cdot B_n' \cdot Y_{n+1}' \qquad (A_n < B_n)$$

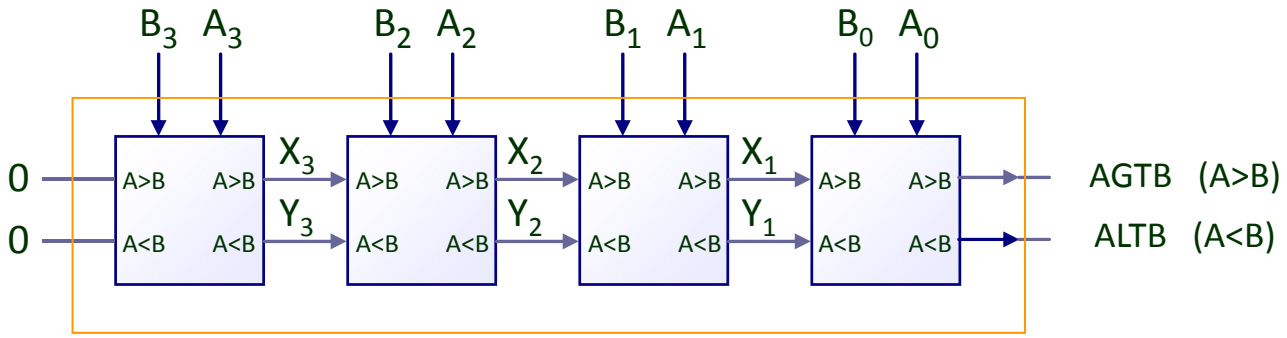
$$X_n = Y_{n+1} + A_n' \cdot B_n \cdot X_{n+1}' \qquad (A_n > B_n)$$

Comparadores de Números de 4 Bits por Iteração de Módulos Simples

A relação de grandeza entre dois números pode ser obtida:

- Por subtracção (tratada mais adiante).
- Por comparação dígito a dígito no sentido dos algarismos de maior peso para os de menor peso. Só quando os algarismos de maior peso forem idênticos ficará a decisão pendente da comparação de algarismos de menor peso.

Um comparador de 2 números de 4 bits é uma estrutura iterativa formada pela concatenação de módulos comparadores de 2 bits do mesmo peso (slide anterior), com entradas resultantes de comparação de bits de maior peso e saídas indicadoras do resultado.

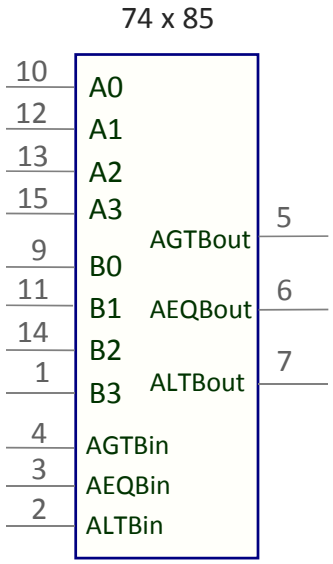


Estrutura interna de um comparador de dois números A e B de 4 bits cada, sintetizado por iteração de módulos comparadores de um par de bits analisados no slide anterior.

AGTB: A Greater Than B
ALTB: A Less Than B
AEQB: A Equal B

Saída AEQB redundante, obtida a partir das anteriores através da adição de uma porta NOR.

74x85 – 4 Bit Magnitude Comparator



Símbolo lógico e configuração dos pinos de um circuito 74x85 4-bit Magnitude Comparator (comparador de números de 4 bits).

O circuito 74x85 comparador de números de 4 bits (símbolo lógico à esquerda) tem 3 entradas e 3 saídas para permitir a ligação em cascata e a expansão iterativa. Uma das entradas e uma das saídas são redundantes, dado serem exclusivas as 3 condições a que correspondem.

As expressões em baixo entre ( ) não são expressões lógicas, representando antes uma comparação aritmética entre as entradas A3-A0 e B3-B0.

$$AGTBout = (A>B) + (A=B) \cdot AGTBin$$
$$AEQBout = (A=B) \cdot AEQBin$$
$$ALTBout = (A<B) + (A=B) \cdot ALTBin$$

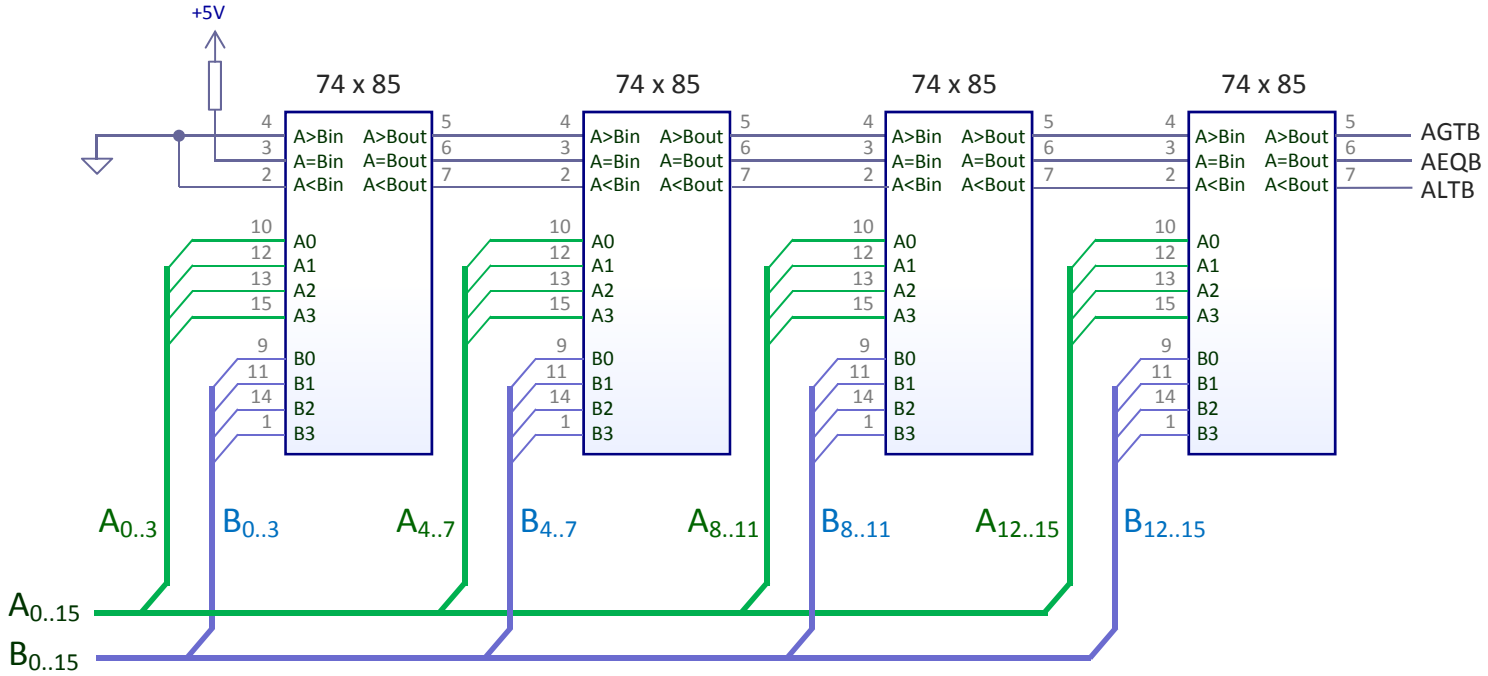
Equações pseudo-lógicas das saídas de ligação em cascata.

Uma expressão lógica normal para (A>B) seria:

$$A>B = A_3\,B_3' + (A_3 \oplus B_3)'\,A_2\,B_2' + (A_3 \oplus B_3)'(A_2 \oplus B_2)'\,A_1\,B_1' +$$
$$(A_3 \oplus B_3)'(A_2 \oplus B_2)'(A_1 \oplus B_1)'\,A_0\,B_0'$$

Portanto AGTBout ficará activa se A>B, ou se A=B mas AGTBin estiver activa, ou seja, se os bits de ordem inferior forem idênticos.

COMPARADORES – LIGAÇÃO DE MÓDULOS EM CASCATA



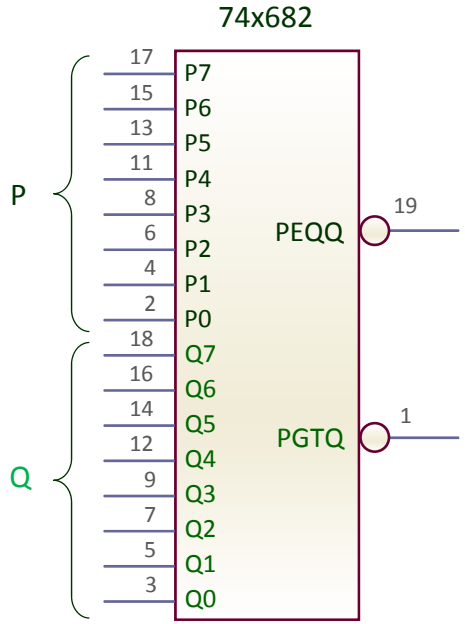
Ligação em cascata de circuitos comparadores de 2 números binários de 4 bits para realização de comparação entre dois números A e B de 16 bits cada.

ENTRADAS: AGTBin ou A>Bin (A Greater Than B)
ALTBin ou A<Bin (A Less Than B)
AEQBin ou A=Bin (A Equal B)

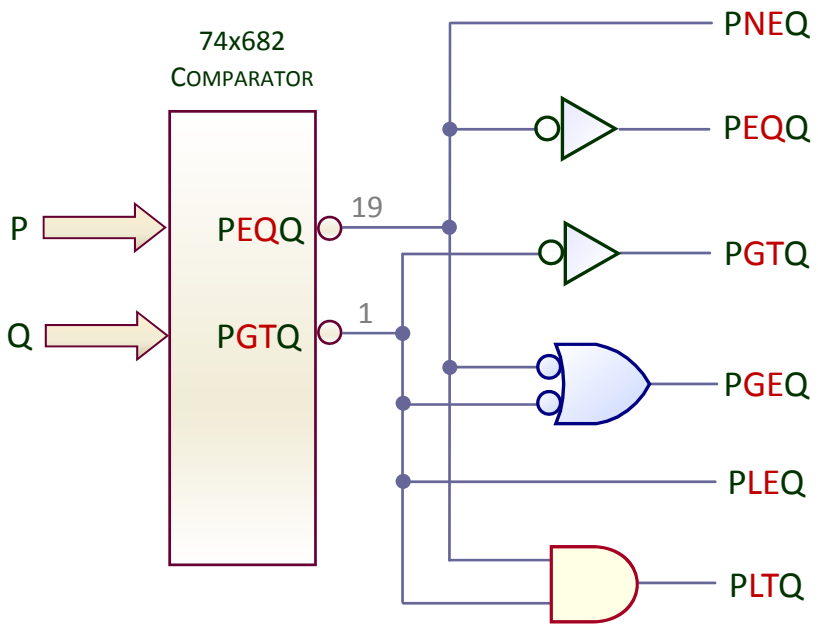
SAÍDAS: AGTBOUT ou A>Bout (A Greater Than B)
ALTBout ou A<Bout (A Less Than B)
AEQBout ou A=Bout (A Equal B)

As entradas e saídas listadas em cima, dedicadas à ligação em cascata dos circuitos 74x85, são definidas para que as saídas de um módulo que esteja a comparar um conjunto de bits menos significativos dos operandos sejam ligadas às entradas de outro módulo idêntico que esteja dedicado à comparação do conjunto de bits imediatamente mais significativos , como indicado, e de acordo com o slide anterior.

74x682 – 8 Bit Magnitude Comparator e Indicadores Relacionais



Símbolo lógico tradicional e configuração dos pinos de um circuito 74x682 8-bit Magnitude Comparator.



Ligação de lógica auxiliar ao Comparador de 8 bits 74x682 para indicação de saídas relacionais adicionais (de natureza Active-high).

NE – Not Equal
EQ – Equal
GT – Greater Than
GE – Greater or Equal
LE – Less or Equal
LT – Less Than

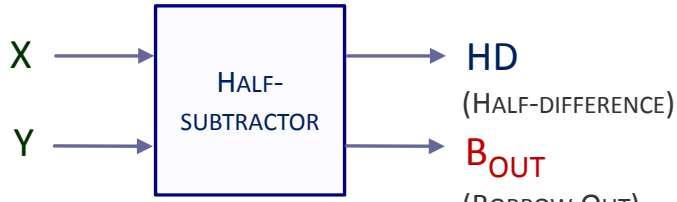
Existe uma classe de circuitos integrados MSI designada por Magnitude Comparators, de que fazem parte o 74x85 mas também os circuitos 74x682, 74x684 e 74x688. Interpretam os números de entrada e produzem indicadores (Flags) relacionais indicativos de uma relação aritmética (GT – Greater Than – ou LT – Less Than) entre eles.

Este três últimos circuitos comparam dois números P e Q de 8 bits cada um, em código binário ou BCD. Todos possuem uma saída P=Q, e os 682 e 684 possuem ainda uma saída P>Q. O 74x682 dispõe ainda de amplificação da corrente nas saídas que é superior à típica da família TTL. As entradas exibem histerese (são do tipo Schmitt Trigger). Não possuem contudo, ao contrário do 74x85, entradas para ligação em cascata, nem a saída LT – Less Than. No entanto, qualquer dos indicadores relacionais pode ser obtido a partir das 2 saídas existentes, como mostrado na Fig. da direita em cima.

CIRCUITO SEMI-SUBTRACTOR (HALF-SUBTRACTOR)

X - Y

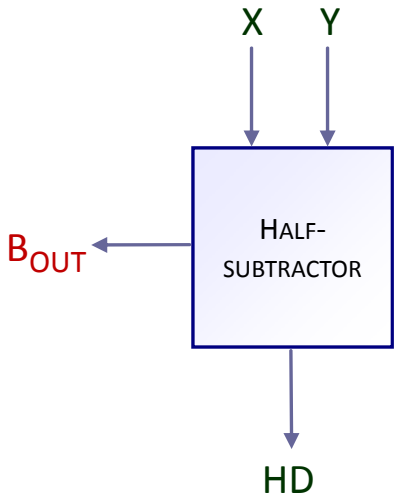
X e Y são operandos de um bit (X é o **aditivo** e Y o **subtractivo**) que produzem uma diferença de dois bits.



Símbolo lógico.

X	Y	HD	B_{OUT}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Tabela da subtracção binária.



Símbolo lógico
alternativo para iteração.

$$HD = X \oplus Y$$

$$B_{OUT} = X' Y$$

Equações algébricas
das funções HD e B_{OUT}.

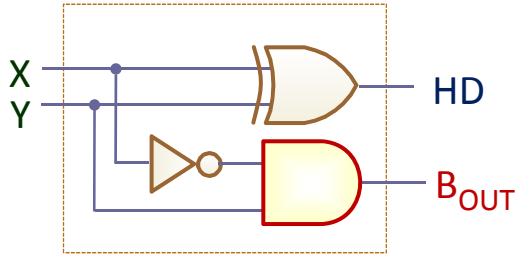
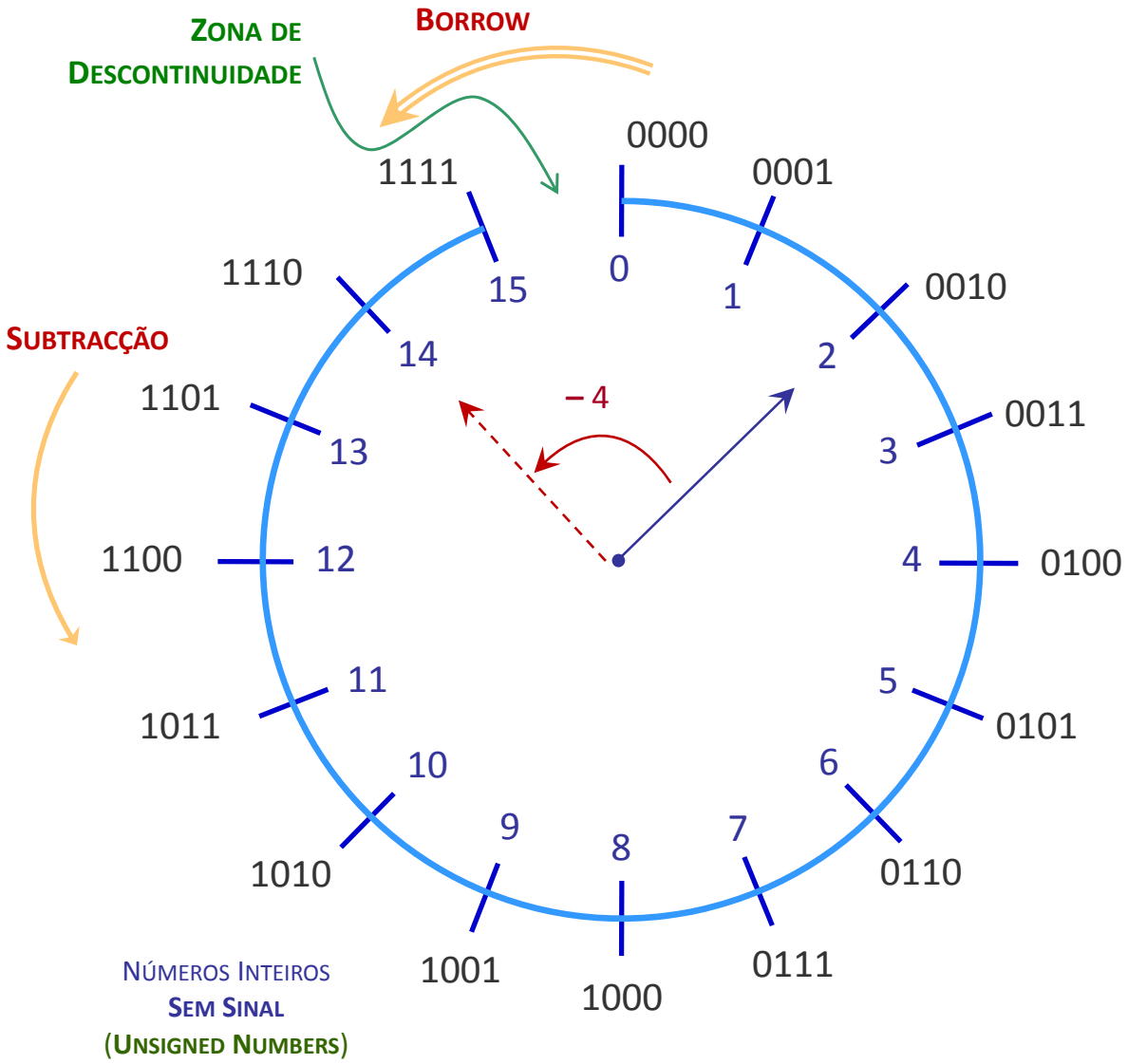


Diagrama Lógico.

SUBTRACÇÃO DE NÚMEROS INTEIROS SEM SINAL A 4 BITS – REPRESENTAÇÃO GRÁFICA



Representação gráfica em 'Roda de Números'.

- Um número é subtraído de outro movendo a seta no anel um número correspondente de posições (no sentido contrário ao dos ponteiros do relógio).
- Se o resultado da operação subtracção exceder a capacidade de representação a seta mover-se-á através da zona de descontinuidade.
- Neste caso ocorre um BORROW e terá de ser tomado em conta o bit de transporte.

Exemplo da subtracção dos nºs 2 e 4 com BORROW:

	PESO	16	8	4	2	1	
			1	0	0	0	← B_{IN}
	2		0	0	1	0	
–	4	–	0	1	0	0	
		1	1	1	1	0	

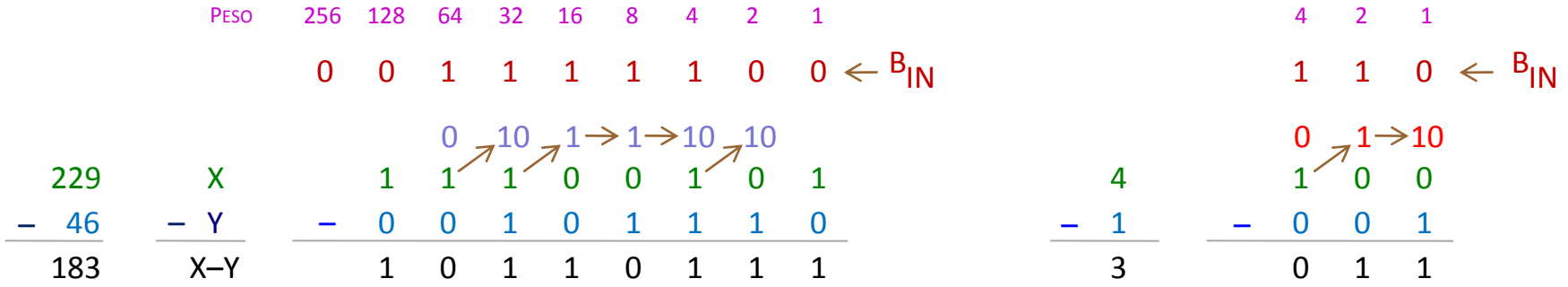
BORROW — RESULTADO a 4 bits, errado

EXEMPLOS DA SUBTRACÇÃO BINÁRIA (EX. 5-11)

Exemplo 5-11

X	ADITIVO (SUBTRAENDO OU DIMINUENDO) MINUEND
– Y	SUBTRACTIVO (SUBTRACTOR OU DIMINUIDOR) SUBTRAHEND
X – Y	DIFERENÇA DIFFERENCE

O algoritmo da subtracção é formalmente idêntico ao da adição.



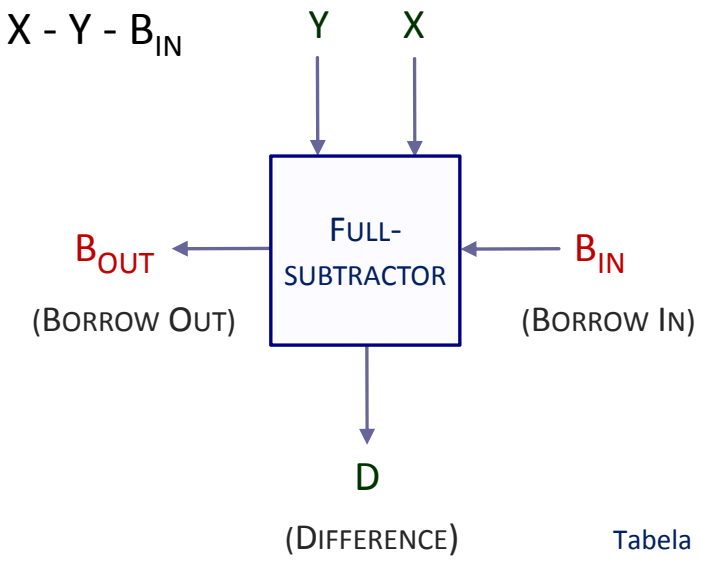
Se o aditivo é maior ou igual ao subtractivo, faz-se a subtracção sem 'empréstimo' em cada coluna.

Se o aditivo é menor do que o subtractivo, para que a subtracção se realize, faz-se o 'empréstimo' proveniente da coluna à esquerda, e só depois se faz a subtracção, como representado acima pelas setas e pelos bits associados .

Neste caso diz-se que se gerou um BORROW (pedido de empréstimo) para a coluna seguinte, porque esta quantidade foi pedida aos algarismos de maior peso do número.

O algoritmo mais comum utilizado dia a dia quando se realizam subtracções é o de somar o arrasto ao subtractivo por ser mais simples, tendo em conta que produz os mesmos efeitos.

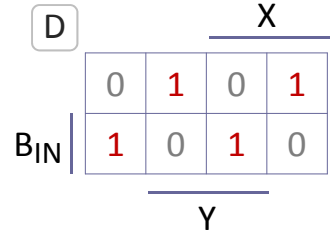
CIRCUITO SUBTRACTOR-COMPLETO (FULL-SUBTRACTOR)

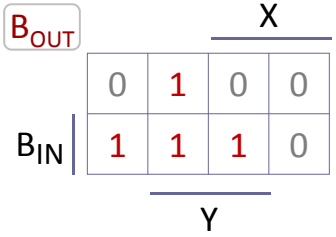


Símbolo lógico adequado para iteração.

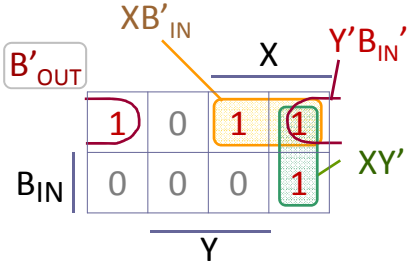
B_{IN}	X	Y	D	B_{OUT}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

Tabela de verdade (para cada par de valores nas entradas X e Y adiciona-se B_{IN} a Y subtraindo-se seguidamente de X para a obtenção da diferença D e do transporte B_{OUT}).





$$B_{OUT} = X'Y + X'B_{IN} + YB_{IN}$$



$$D = X \oplus Y \oplus B_{IN} = X \oplus Y' \oplus (B_{IN})'$$

$$(B_{OUT})' = XY' + X\,(B_{IN})' + Y'(B_{IN})'$$

Mapas de Karnaugh e equações algébricas das funções D e $(B_{OUT})'$.

	0	0	0	0	1	1	1	1 ← B_{IN}
	0	0	1	1	0	0	1	1 ← X
−	0	− 1	− 0	− 1	− 0	− 1	− 0	− 1 ← Y
	0 0	1 1	0 1	0 0	1 1	1 0	0 0	1 1
							B_{OUT}	D

Subtracções a três bits de dois operandos X e Y de um bit, com um bit de Borrow In (B_{IN}).

SUBTRACTOR-COMPLETO OBTIDO A PARTIR DE UM SOMADOR-COMPLETO

$$S = X \oplus Y \oplus C_{IN}$$

$$C_{OUT} = X\,Y + X\,C_{IN} + Y\,C_{IN}$$

Equações obtidas para o circuito somador completo.

$$D = X \oplus Y \oplus B_{IN} = X \oplus Y' \oplus (B_{IN})'$$

$$(B_{OUT})' = XY' + X\,(B_{IN})' + Y'(B_{IN})'$$

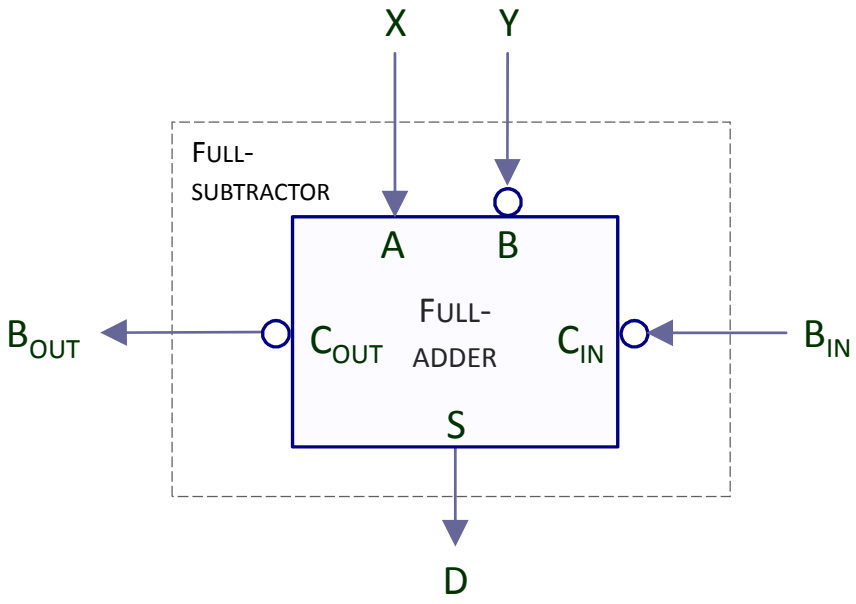
Equações para o circuito Subtractor completo.

$$Y \Rightarrow Y'$$

$$C_{IN} \Rightarrow (B_{IN})'$$

$$C_{OUT} \Rightarrow (B_{OUT})'$$

Transformação de variáveis necessária para obtenção do circuito subtractor a partir de um circuito somador.

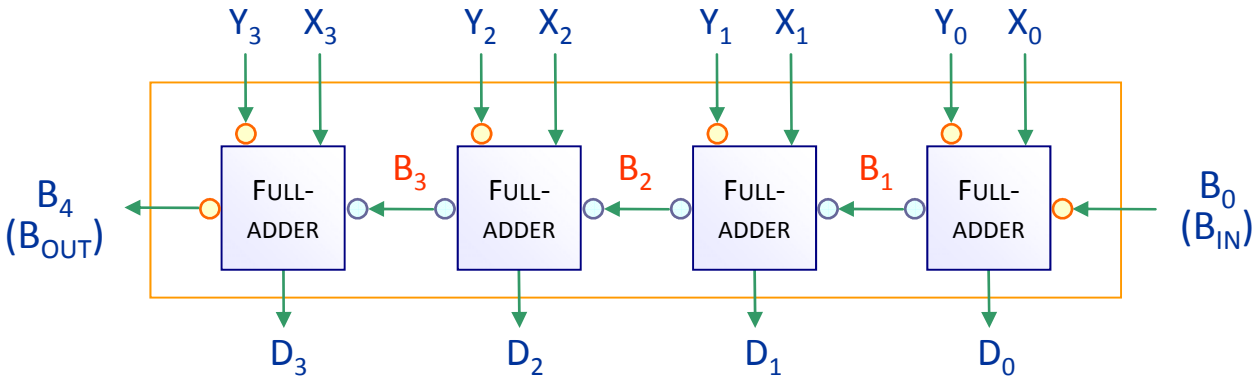


Símbolo lógico do SUBTRACTOR obtido a partir de um somador.

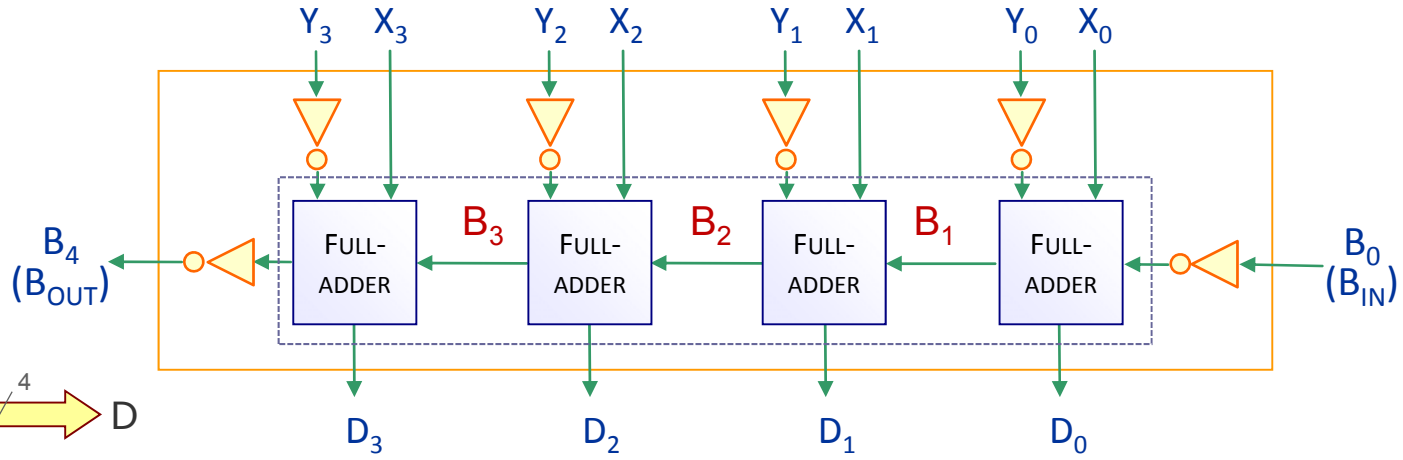
A ADIÇÃO BINÁRIA é a operação aritmética mais executada pelos sistemas digitais.

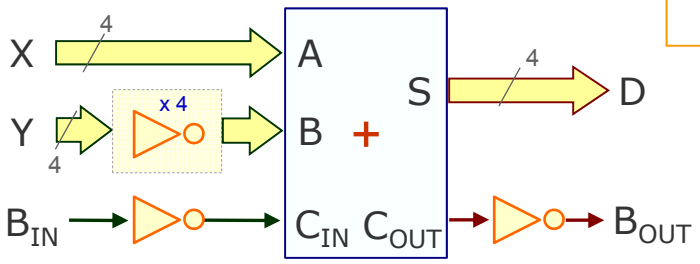
Os mesmos circuitos somadores são utilizados para a adição em N_0 (números sem sinal) e em Z (números com sinal) como se verá adiante.

SUBTRACTOR DE NÚMEROS DE 4 BITS



Realizar a SUBTRACÇÃO $X-Y-B_{IN}$ é o mesmo que realizar a SOMA $X + Y' + B'_{IN}$ tomando em atenção que $B_{OUT} = C'_{OUT}$.

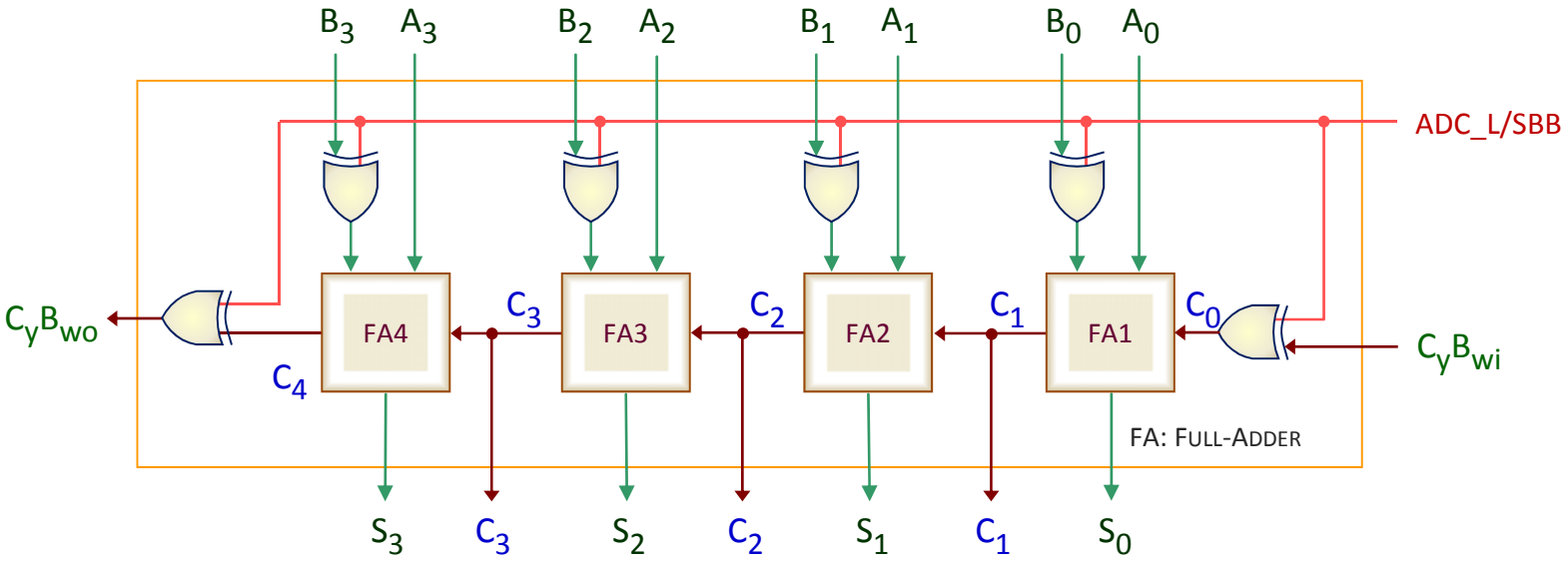




Representação em diagrama de blocos de um SUBTRACTOR de números de 4 bits obtido a partir de um somador.

Estrutura do SUBTRACTOR mostrando a sua síntese a partir de um circuito somador pela interligação dos módulos iterativos FULL ADDER e dos inversores nas linhas de Y, C_{IN} e C_{OUT}.

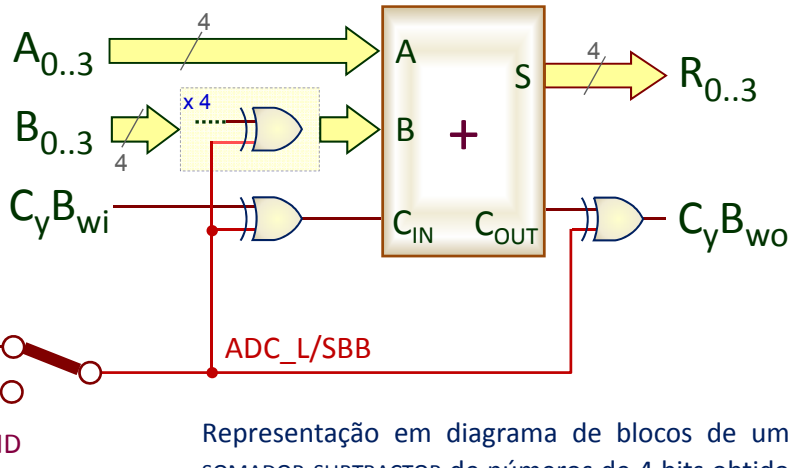
SOMADOR-SUBTRACTOR DE NÚMEROS DE 4 BITS



Estrutura do SOMADOR-SUBTRACTOR mostrando a sua síntese a partir de um circuito SOMADOR pela interligação dos módulos iterativos FULL ADDER e dos inversores controlados (portas XOR) nas linhas de B, C_yB_{wi} e C_yB_{wo}.

C_yB_{wi} – a mesma linha tem o significado de CARRY-IN ou de BORROW-IN consoante se trate de uma ADIÇÃO ou de uma SUBTRACÇÃO.

C_yB_{wo} – a mesma linha tem o significado de CARRY-OUT ou de BORROW-OUT consoante se trate de uma ADIÇÃO ou de uma SUBTRACÇÃO.



SUBTRACT WITH BORROW	$R = A - B - C_yB_{wi}$	**SBB**	1
ADD WITH CARRY	$R = A + B + C_yB_{wi}$	**ADC_L**	0

Representação em diagrama de blocos de um SOMADOR-SUBTRACTOR de números de 4 bits obtido a partir de um somador e de portas XORs.

Exemplos da Subtracção Binária implementada com Somador (Ex. 5-12 com B_{WO}= 0)

Exemplo 5-12

9 – 7 – 0

	0		1	1	0	0
	9		1	0	0	1
–	7	–	0	1	1	1
	2		0	0	0	1	0

B_{WO}= 0 Resultado a 4 bits

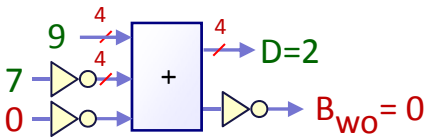
Subtracção X-Y-B_{wi} a 'Papel e Lápis'

		0	0	1	1	← 0
		1	0	0	1
+		1	0	0	0
	1	0	0	1	0

B_{WO}= 0 Resultado a 4 bits

Subtracção X-Y-B_{wi} mecanizada através da Adição X + Y'+ $(B_{wi})'$

B_{wi} = 0



9 – 7 – 1

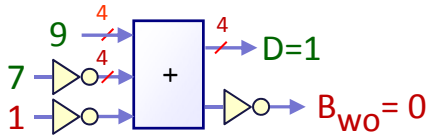
	1		1	1	1	1
	9		1	0	0	1
–	7	–	0	1	1	1
	1		0	0	0	1

B_{WO}= 0 Resultado a 4 bits

		0	0	0	0	← 1
		1	0	0	1
+		1	0	0	0
	1	0	0	0	1

B_{WO}= 0 Resultado a 4 bits

B_{wi} = 1



Realizar a Subtracção:

$X - Y - B_{wi}$

é o mesmo que realizar a Soma:

$X + Y' + (B_{wi})'$

Exemplos da Subtracção Binária Implementada com Somador (Ex. 5-13 com B_{WO}= 1)

Exemplo 5-13

7 – 9 – 0

	0	0 0 0 0
	7	0 1 1 1
–	9	– 1 0 0 1
	Erro	1 1 1 1 0

B_{WO}= 1 — Resultado a 4 bits

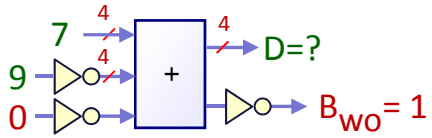
Subtracção X-Y-B_{wi} a 'Papel e Lápis'

		1 1 1 1 ← 0
		0 1 1 1
		+ 0 1 1 0
		0 1 1 1 0

B_{WO}= 1 — Resultado a 4 bits

Subtracção X-Y-B_{wi} mecanizada através da Adição X + Y'+ $(B_{wi})'$

B_{wi} = 0



Realizar a Subtracção:

$X - Y - B_{wi}$

é o mesmo que realizar a Soma:

$X + Y' + (B_{wi})'$

7 – 9 – 1

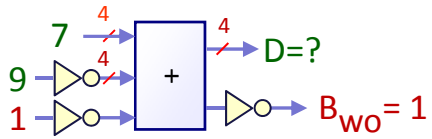
	1	0 0 1 1
	7	0 1 1 1
–	9	– 1 0 0 1
	Erro	1 1 0 1

B_{WO}= 1 — Resultado a 4 bits

		1 1 0 0 ← 1
		0 1 1 1
		+ 0 1 1 0
		0 1 1 0 1

B_{WO}= 1 — Resultado a 4 bits

B_{wi} = 1

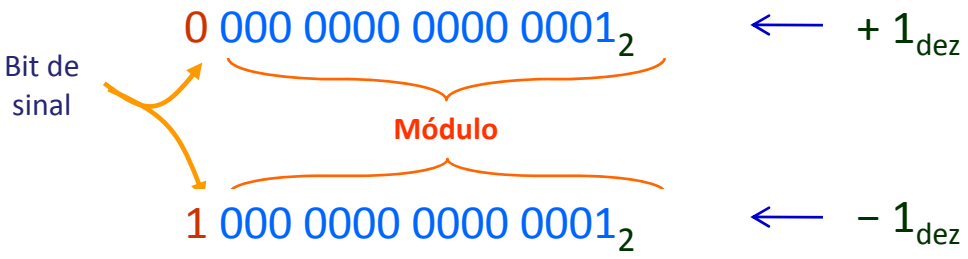


A representação de números inteiros tem de ter em conta que os números podem ser POSITIVOS, NEGATIVOS ou o número 0.

Uma solução para representar números inteiros **com sinal** (positivos e negativos) é a notação por **SINAL E MÓDULO** (**SIGN AND MAGNITUDE**), em que:

- se reserva o bit mais significativo (à esquerda) para o sinal, e se define que se esse bit for **0** o número é **positivo**, e se for **1** é **negativo**;
- os bits restantes representarão o valor absoluto (módulo) do número.

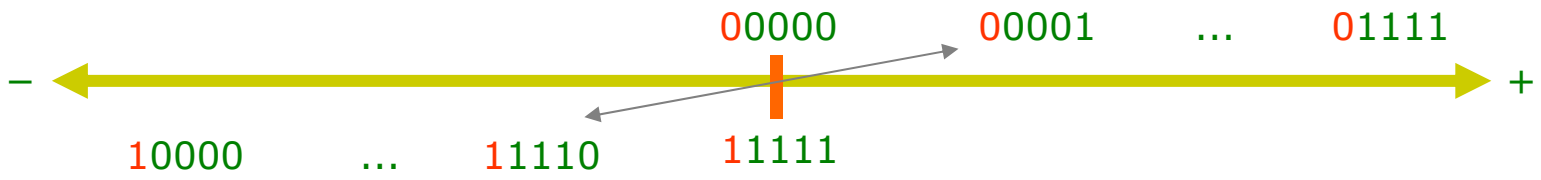
Os números $+1_{dez}$ e -1_{dez} numa notação de Sinal e Módulo a 16 bits escrevem-se como indicado:



Esta notação tem alguns inconvenientes:

- existem duas representações para o número 0: 1000 = **– 0**, 0000 = **+ 0** (a 4 bits);
- para realizar operações é necessário processar de forma diferente o **módulo** e o **sinal**;
- isto complica significativamente os circuitos aritméticos que realizam as operações de soma e de subtracção, pelo que esta notação é pouco utilizada.

REPRESENTAÇÃO DE NÚMEROS INTEIROS COM SINAL – COMPLEMENTO PARA 1



Outra representação é designada de **COMPLEMENTO PARA 1** (1'S COMPLEMENT), e consiste em:

- Inverter bit a bit todos os bits de um número para determinar o seu complemento para 1.
- Para um número X a n bits o complemento para 1 (ou complemento restringido) é $(2^n-1) – X$.
- Quando o bit mais à esquerda é **0** o número é **positivo**, quando é **1** o número é **negativo.**

- O número $+1_{dez}$ numa representação **complemento para 1** de números inteiros a 16 bits será:

0 000 0000 0000 0001

- O número simétrico -1_{dez} nesta representação será a negação bit a bit:

1 111 1111 1111 1110

- Apesar das boas propriedades para a adição e dos circuitos aritméticos advirem mais simples, existem ainda duas representações para o número 0:

  1111 = **– 0**, 0000 = **+ 0** (a 4 bits).

Representação de Números Inteiros com Sinal – Complemento para 2

- O Complemento para 2 de um número X com n bits, ou o Complemento para 2^n , é definido como $2^n - X$ para X≠0, e 0 para X=0. Obtém-se, desta forma, o Simétrico de X.
- Exemplo: o complemento para 2^4 de 0011 (+3) é, a 4 bits, $\mathbf{2^4 - 0011}$, ou seja, o número original 0011 invertido bit a bit adicionado de 1, que é o número 1101 (-3):

$$\begin{array}{r} 1\ 0\ 0\ 0\ 0 \\ -\ 0\ 0\ 1\ 1 \\ \hline 1\ 1\ 0\ 1 \end{array}$$

Inversão bit a bit de +3 adicionada de 1

$$\begin{array}{r} 1\ 1\ 1\ 1 \\ -\ 0\ 0\ 1\ 1 \\ \hline 1\ 1\ 0\ 0 \end{array}$$

Inversão bit a bit de +3

$$(1101)_2 = 1 \times (-2^3) + 1 \times (2^2) + 1 \times (2^0) = -8 + 5 = -3$$

$$-3 = (2^4) - 3 = (1\,0000)_2 - 3 = ((1111)_2 + 1) - 3 = \underbrace{((1111)_2 - 3)}_{\text{Inversão bit a bit de +3}} + 1 = 13$$

- Se um número X tem n bits, o complemento para 2 é representável de igual modo por n bits.
- O complemento para 2 do complemento para 2 de um número X é X pois que $\mathbf{2^n - (2^n - X) = X}$
- O intervalo de representação em complemento para 2 é $[-2^{n-1}, +2^{n-1}-1]$ (por exemplo [–8, +7], [–4, +3]), ou seja, existe mais um número negativo do que o maior dos positivos, devendo-se esta assimetria ao facto de o número 0 ter uma só representação nesta notação.

Código de Complemento para 2 a 4 Bits

Bit de Sinal →

MSB
Most Significant Bit

	Pesos				Número representado
	-2^3	2^2	2^1	2^0	
0	0	0	0	0	+0
1	0	0	0	1	+1
2	0	0	1	0	+2
3	0	0	1	1	+3
4	0	1	0	0	+4
5	0	1	0	1	+5
6	0	1	1	0	+6
7	0	1	1	1	+7
8	1	0	0	0	−8
9	1	0	0	1	−7
10	1	0	1	0	−6
11	1	0	1	1	−5
12	1	1	0	0	−4
13	1	1	0	1	−3
14	1	1	1	0	−2
15	1	1	1	1	−1

Só há uma representação para o Zero (zero é considerado Positivo porque o bit de sinal é zero)

Tabela com a representação de números a 4 bits em Complemento para 2.

O Complemento para 2 de um número consiste em complementar (negar) todos os bits do número e somar-lhe 1:

Exemplo:

- Representação de -7 (partindo de $+7$, a verde):

  $0111 \Rightarrow 1000 + 1 \Rightarrow 1001$

- Representação de $+7$ (partindo de -7, a verde):

  $1001 \Rightarrow 0110 + 1 \Rightarrow 0111$

O Complemento para 2 de um número pode ser formado percorrendo a representação do número a complementar, desde o bit menos significativo até ao mais significativo: até ao encontro do primeiro 1, incluindo-o, mantêm-se todos os 0s, e, a partir daí, invertem-se os restantes bits.

Na representação em Complemento para 2:

- Um **número positivo** é representado pelo seu Módulo (como na representação de sinal e módulo).
- Um **número negativo** é representado pelo Complemento para 2 do seu Módulo.

A tabela ao lado exemplifica este procedimento.

Representação de Números Inteiros com Sinal nas três Notações Consideradas

Decimal	Módulo e Sinal	Complemento para 1	Complemento para 2
-8	- - - -	- - - -	1 0 0 0
-7	1 1 1 1	1 0 0 0	1 0 0 1
-6	1 1 1 0	1 0 0 1	1 0 1 0
-5	1 1 0 1	1 0 1 0	1 0 1 1
-4	1 1 0 0	1 0 1 1	1 1 0 0
-3	1 0 1 1	1 1 0 0	1 1 0 1
-2	1 0 1 0	1 1 0 1	1 1 1 0
-1	1 0 0 1	1 1 1 0	1 1 1 1
+0	1000 ou 0000	1111 ou 0000	0 0 0 0
+1	0 0 0 1	0 0 0 1	0 0 0 1
+2	0 0 1 0	0 0 1 0	0 0 1 0
+3	0 0 1 1	0 0 1 1	0 0 1 1
+4	0 1 0 0	0 1 0 0	0 1 0 0
+5	0 1 0 1	0 1 0 1	0 1 0 1
+6	0 1 1 0	0 1 1 0	0 1 1 0
+7	0 1 1 1	0 1 1 1	0 1 1 1

Tabela comparativa das notações utilizadas para a representação de números inteiros com sinal a 4 bits.

Comparação de Números sem Sinal e de Números com Sinal (Ex. 5-14)

Exemplo 5-14

Qual dos números é maior ?

$$X = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{2}$$

$$Y = 0011\ 1011\ 1001\ 1010\ 1100\ 1010\ 0000\ 0000_{2}$$

É $X > Y$?

Conversão para decimal para verificação:

- Sem Sinal (Unsigned): $X = 4\ 294\ 967\ 292_{10} > Y = 1\ 000\ 000\ 000_{10}$ logo X > Y
- Com Sinal (Signed): $X = -4_{10} < Y = 1\ 000\ 000\ 000_{10}$ logo X < Y

Em linguagem C:

int

Declara um número Com Sinal

Usa a notação de complemento para dois.

unsigned int

Declara um número **Sem Sinal**

Trata os números de 32 bits como inteiros sem sinal.

Somas e Subtracções de Números com Sinal (Ex. 5-15)

Exemplo 5-15

Bit de Sinal

	+4		0	1	0	0
+	+3	+	0	0	1	1
	+7		0	1	1	1

	+4		0	1	0	0
+	−3	+	1	1	0	1
	+1	1	0	0	0	1

Ignora-se o **carry out** do bit de sinal que sai dos 4 bits usados na representação .

	−4		1	1	0	0
+	+3	+	0	0	1	1
	−1		1	1	1	1

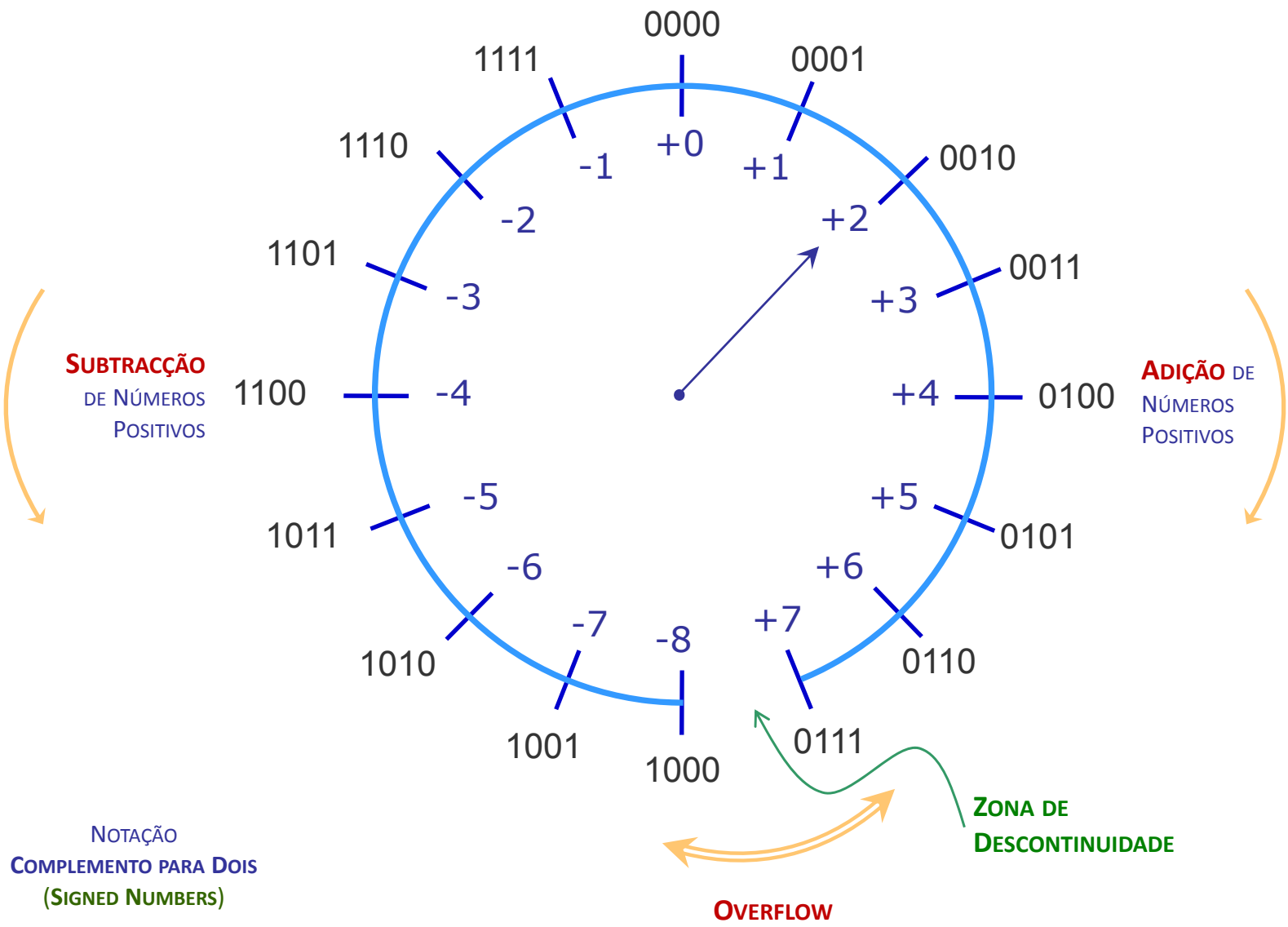
	−4		1	1	0	0
+	−3	+	1	1	0	1
	−7	1	1	0	0	1

Os números em complemento para 2 podem ser somados através da adição binária normal, mas ignorando qualquer **carry** do **bit de sinal** (MSB).

Quando a operação envolve dois números com o mesmo sinal, é possível que o resultado não possa ser representado com o número de bits disponível. Diz-se então que ocorre **Overflow**.

O overflow nunca ocorre em operações de adição entre números com sinal contrário – o resultado será neste caso sempre correcto desde que o intervalo do sistema de numeração seja respeitado.

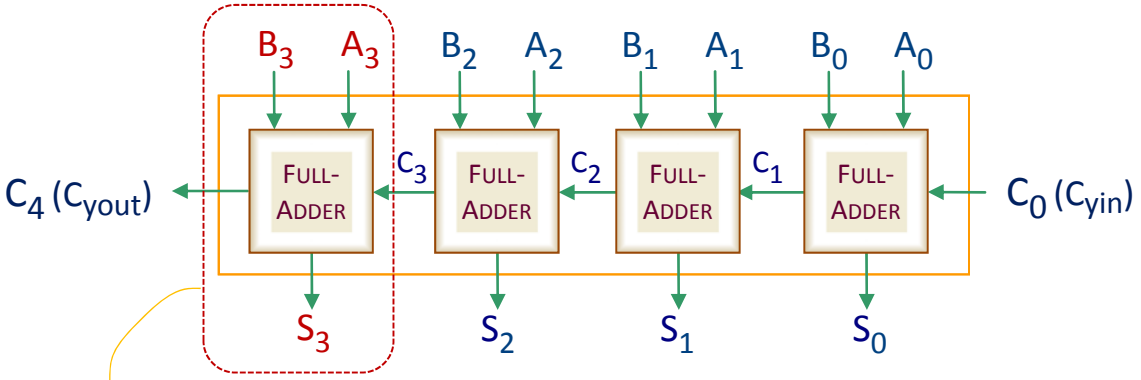
Adição e Subtracção de Números Inteiros com Sinal a 4 bits – Representação Gráfica



Overflow – Circuito Detector a partir dos Bits de Sinal

C_3	B_3	A_3	S_3	C_4	O_v
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	**1**	**1**	**0**	1	**1**
1	**0**	**0**	**1**	0	**1**
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	1	1	0

Tabela de verdade mostrando o Overflow (O_v) detectado através dos bits de sinal dos operandos A_3 e B_3 e do resultado soma S_3.



Estrutura interna de um somador de 4 bits assinalando as posições relevantes dos bits de sinal dos operandos A_3 e B_3 e da soma S_3 para a detecção de Overflow (Ov).

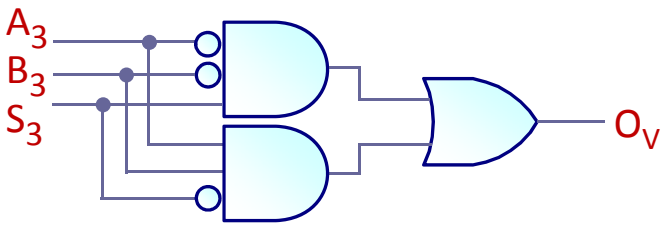
Existe Overflow quando os operandos são ambos do mesmo sinal e o resultado de sinal contrário.

		1	0	0	← Carry-out parcial →		0	0	1
	+6	0	1	1	0	−7	1	0	0	1
+	+4	+ 0	1	0	0	+ −3	+ 1	1	0	1
	−6	~~0~~	1	0	1	0	+6	~~1~~	0	1	1	0

A soma de dois números **positivos** (+6 e +4) origina um número **negativo** (−6) em vez de +10. Ignora-se o Carry-out do bit de sinal (cruzado na Fig.).

A soma de dois números **negativos** (−7 e −3) origina um número **positivo** (+6) em vez de −10.

Exemplos de ocorrência e detecção de Overflow (Ov).



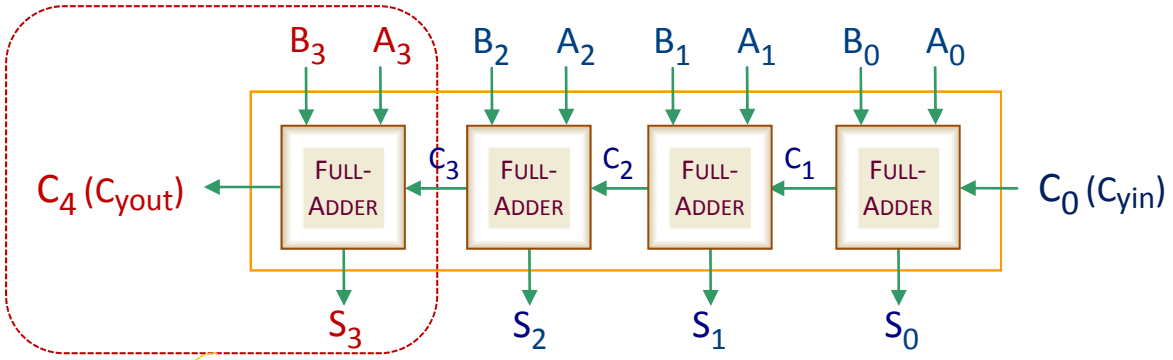
$$O_V = A_3' B_3' S_3 + A_3 B_3 S_3'$$

Circuito de detecção na forma AND-OR.

Overflow – Circuito Detector Alternativo evocando o Carry-out do Somador

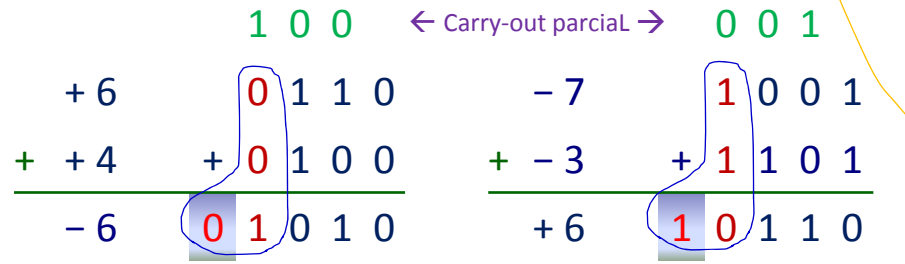
C_3	B_3	A_3	S_3	C_4	O_v
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	**1**	**1**	**0**	**1**	**1**
1	**0**	**0**	**1**	**0**	**1**
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	1	1	0

Tabela de verdade mostrando O_v resultando do XOR entre os 3 bits de sinal A_3, B_3, S_3 e o Carry-out C_4.



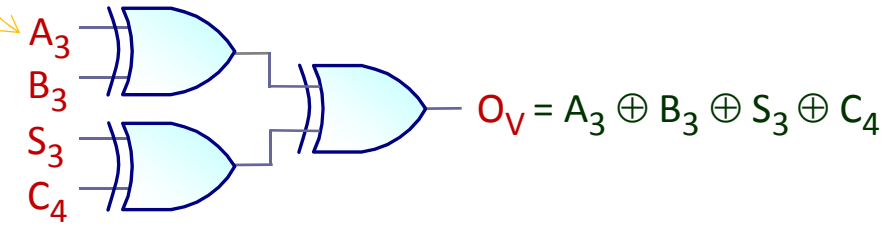
Estrutura interna de um somador de 4 bits assinalando as posições relevantes dos bits de sinal e do Carry-out C_4 para a detecção de Overflow.

Evoca-se o Carry-out (C_4) do somador (Carry-out do bit de sinal, ignorado e sem validade no resultado), que é função das variáveis existentes A3, B3 e S3. Obtém-se para o Overflow uma expressão mais simplificada implementável por uma porta XOR de 4 entradas:



$O_V = 0 \oplus 0 \oplus 1 \oplus 0 = 1$ $\quad$ $O_V = 1 \oplus 1 \oplus 0 \oplus 1 = 1$

Exemplos de ocorrência e detecção de Overflow (Ov) a partir do XOR entre os 3 bits de sinal A_3, B_3, S_3 e o Carry-out C_4.

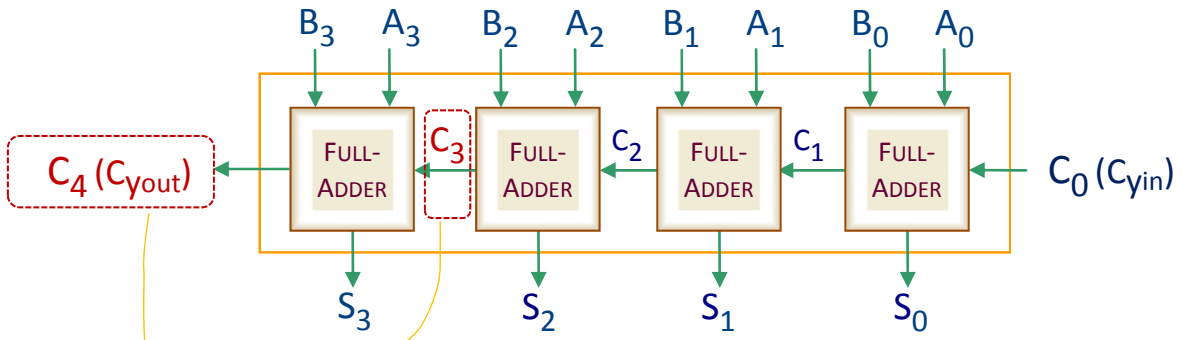


Circuito de detecção simplificado com 3 portas XOR de duas entradas.

Overflow – Circuito Detector Alternativo evocando o Carry-in e o Carry-out do Bit de Sinal

C_3	B_3	A_3	S_3	C_4	O_v
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	0	**1**	**1**
1	0	0	1	**0**	**1**
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	1	1	0

Tabela de verdade mostrando
a dependência de O_v de C_3 e de C_4.



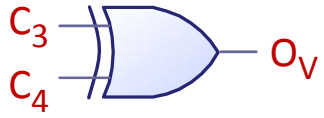
Estrutura interna do somador 74x283 de 4 bits
assinalando as posições de C_3 e C_4 e do bit de sinal S_3.

O Overflow (O_v) ocorre sempre que o Carry-in para o interior da posição de bit de sinal C_3 é diferente do Carry-out para o exterior do somador C_4.

			1	0	0	← Carry-out parcial →	0	0	1	
	+6		0	1	1	0	−7		1	0	0	1
+	+4	+	0	1	0	0	+	−3	+	1	1	0	1
	−6		0	1	0	1	0	+6		1	0	1	1	0

$O_V = C_3 \oplus C_4 = 1 \oplus 0 = 1$ $O_V = C_3 \oplus C_4 = 0 \oplus 1 = 1$

Exemplos de ocorrência e detecção de Overflow (Ov) a partir do
Carry-in C3 e Carry-out C4 da posição de bit de sinal.

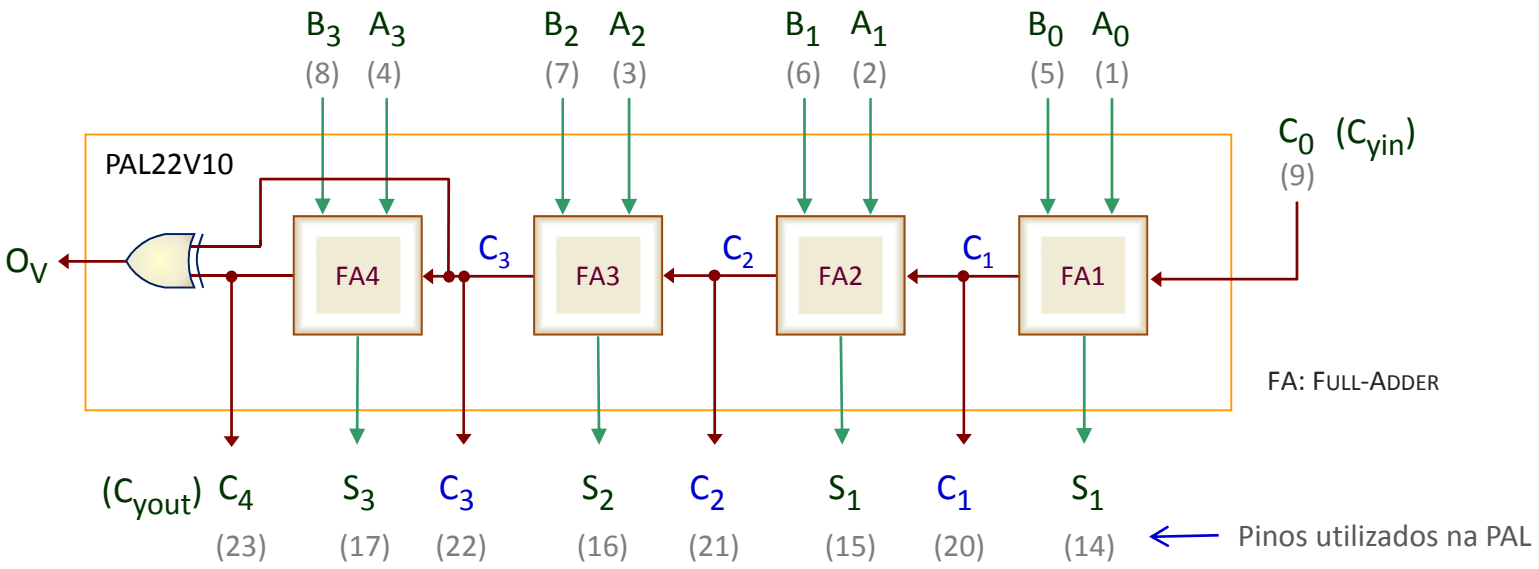


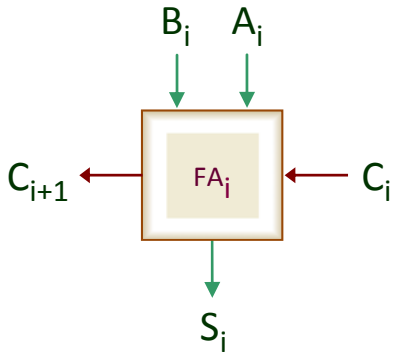
$O_V = C_3 \oplus C_4$

Circuito de detecção
simplificado com porta XOR.

SOMADOR DE NÚMEROS DE 4 BITS COM DETECÇÃO DE OVERFLOW: IMPLEMENTAÇÃO EM PAL22V10

Funcionalidade interna da PAL correspondente ao SOMADOR iterativo simples de números de 4 bits com deteção de OVERFLOW (O_V).





Bloco iterativo genérico correspondente a cada módulo-i do FULL ADDER.

$$S_i = (A_i \oplus B_i) \oplus C_i$$

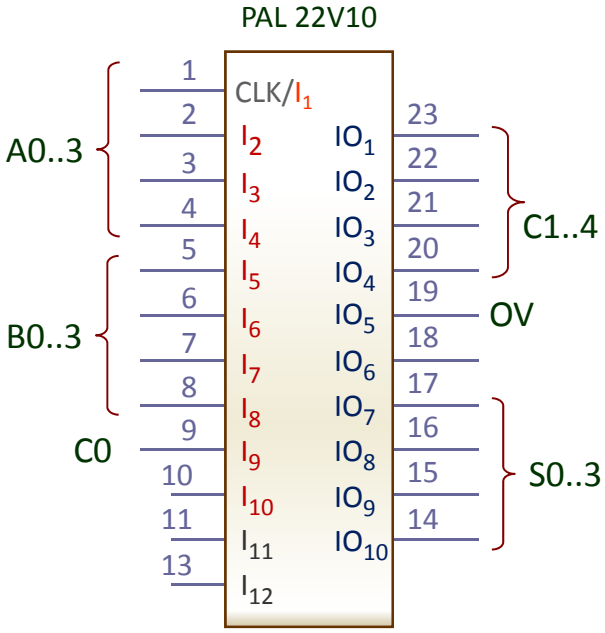
$$C_{i+1} = (A_i \oplus B_i)\, C_i + A_i\, B_i$$

$$O_V = C_3 \oplus C_4$$

Equações das saídas do bloco iterativo a implementar na PAL.

```
Device   p22v10       ;

/* ***** Input Pins ****** **** */
PIN [ 1..4 ]  = [ A0..3 ]    ; /*Operando A*/
PIN [ 5..8 ]  = [ B1..3 ]    ; /*Operando B*/
PIN  9        = C0           ; /*Carry-in*/

/* ***** Output Pins ********** */
PIN [ 14..17 ] = [ S0..3 ]    ; /*Resultado Soma*/
PIN [ 20..23 ] = [ C1..4 ]    ; /*Carry-out 1-4*/
PIN  19        = OV           ; /*Flag Overflow*/

/* ********* Descrição **********/
$REPEAT i = [ 0..3 ]
S{i}   =  A{i} $ B{i} $ C{i};
C{i+1} = (A{i} $ B{i}) & C{i} # (A{i} & B{i});
$REPEND
OV = C3 $ C4 ;
```

Troço do **Código CUPL** com a atribuição dos pinos de entrada e saída e corpo do programa.



Símbolo lógico da PAL assinalando os pinos:

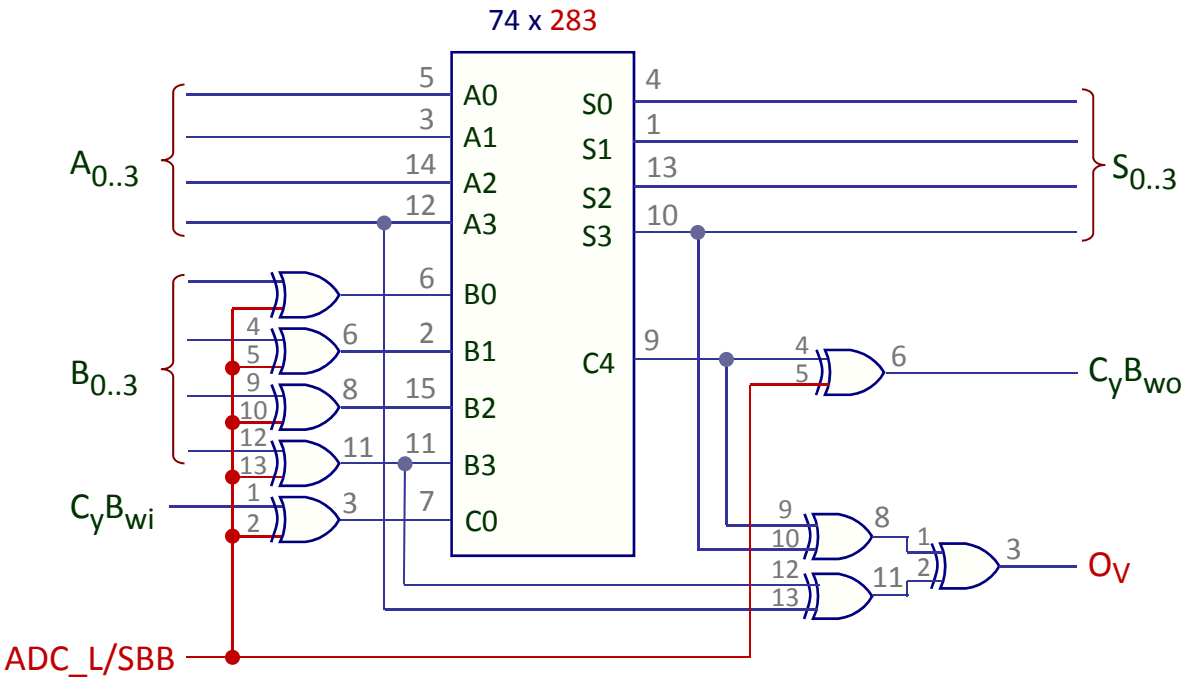
1. de **entrada**:

   A_i e B_i dos operandos A e B, e Carry-in (C_0);

2. de **saída**:

   S_i do resultado soma S, e C_{i+1} correspondentes a todos os Carrys parciais e ao Carry-out final (C_4).

SOMADOR-SUBTRACTOR DE NÚMEROS DE 4 BITS COM DETEÇÃO DE OVERFLOW EM LÓGICA DISCRETA

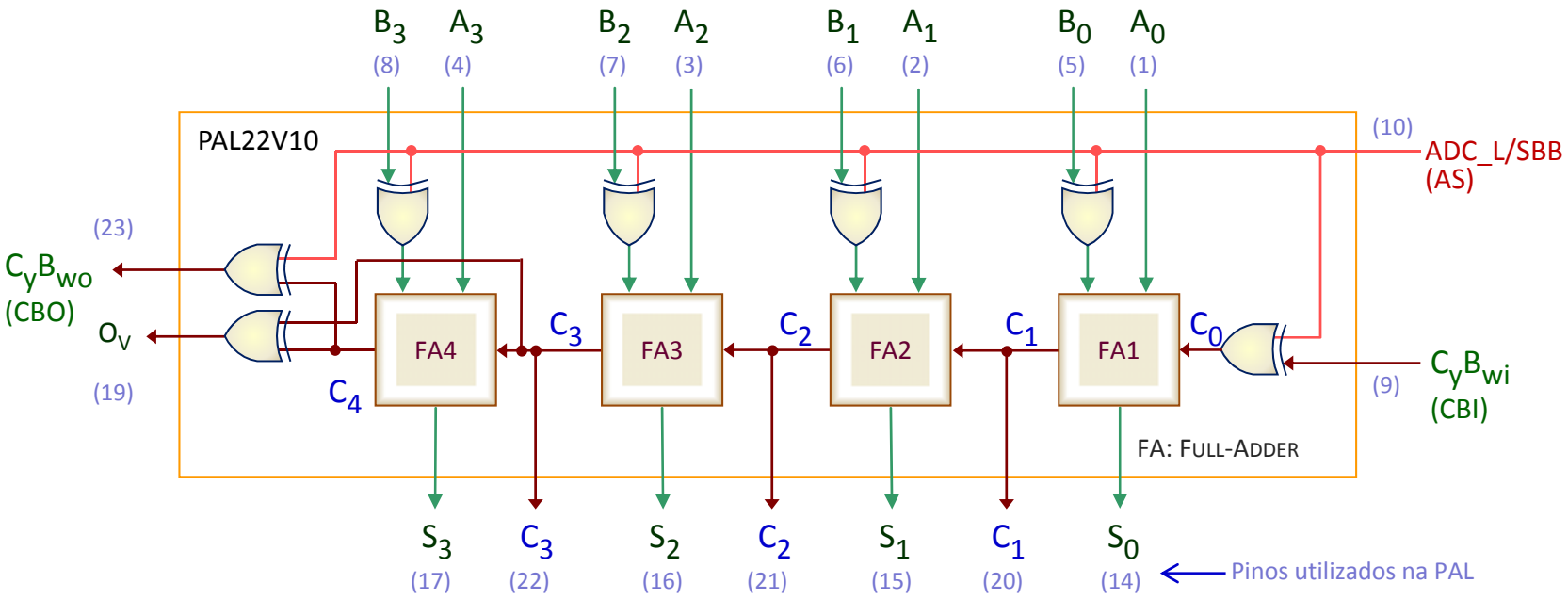


Representação em diagrama lógico de um SOMADOR-SUBTRACTOR de números de 4 bits obtido a partir de um somador 74x283 e de portas XORs.

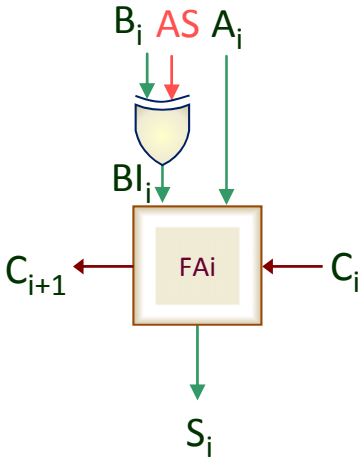
O SOMADOR-SUBTRACTOR é baseado no circuito somador de números de 4 bits 74x283 e programado por acção de um comutador ADC_L/SBB (ADD WITH CARRY de natureza ACTIVE-LOW e SUBTRACT WITH BORROW de natureza ACTIVE-HIGH).

$A_{0..3}$	–	Bits do Operando A
$B_{0..3}$	–	Bits do Operando B
C_yB_{wi}	–	Carry-in/Borrow-in do SOMADOR-SUBTRACTOR
C_yB_{wo}	–	Carry-out/Borrow-out do SOMADOR-SUBTRACTOR
$S_{0..3}$	–	Bits do resultado SOMA-SUBTRACÇÃO
ADC_L/SBB	–	Comutador SOMA-SUBTRACÇÃO

SOMADOR-SUBTRACTOR DE NÚMEROS DE 4 BITS COM DETECÇÃO DE OVERFLOW EM PAL22V10



Funcionalidade interna da PAL correspondente ao SOMADOR-SUBTRACTOR iterativo simples de números de 4 bits com deteção de OVERFLOW (O_V).



FULL-ADDER (FA) e lógica associada ao módulo-i do SOMADOR-SUBTRACTOR iterativo da PAL.

$$BI_i = B_i \oplus AS$$

$$S_i = A_i \oplus BI_i \oplus C_i$$

$$C_{i+1} = (A_i \oplus BI_i)\, C_i + A_i\, BI_i$$

$$O_V = C_3 \oplus C_4$$

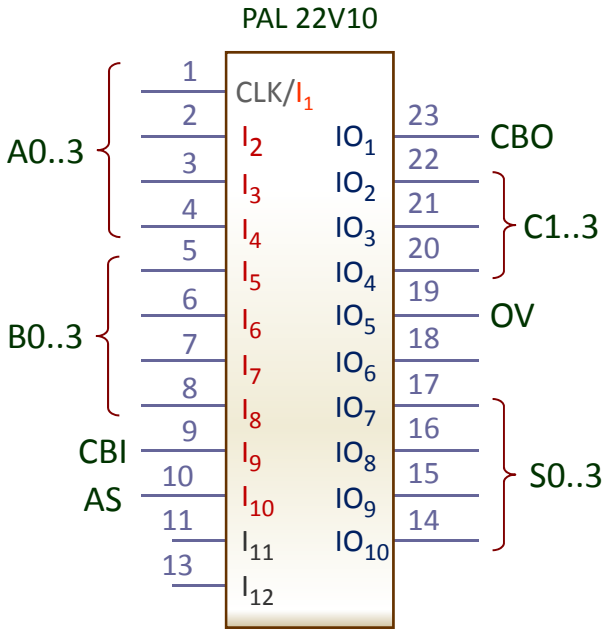
Equações lógicas dos bits intervenientes em cada módulo-**i** do SOMADOR-SUBTRACTOR iterativo a 4 bits com detecção de OVERFLOW.

SOMADOR-SUBTRACTOR – TROÇO DO CÓDIGO CUPL E CONFIGURAÇÃO DA PAL (1)

```
Device   p22v10       ;
/* ***** Input Pins ******** */
PIN [ 1..4 ]  = [ A0.. 3 ] ; /*Operando A*/
PIN [ 5..8 ]  = [ B0.. 3 ] ; /*Operando B*/
PIN  9        =  CBI       ; /*CyBwi*/
PIN  10       =  AS        ; /*Add-subtract*/
/* ***** Output Pins ******* */
PIN [ 17..14 ] = [ S0..3 ] ; /*Resultado*/
PIN [ 22..20 ] = [ C1..3 ] ; /*Carry-out 1-3*/
PIN  23        = CBO       ; /*CyBwo*/
PIN  19        = OV        ; /*Flag Overflow*/
/* ********* Body ********** */
C0 = CBI $ AS
$REPEAT i = [ 0.. 3 ]
BI {i} = B{i} $ AS ;
S{i} =  A{i} $ BI{i} $ C{i};
C{i+1} = (A{i} $ BI{i}) & C{i} # (A{i} & BI{i});
$REPEND
OV  = C3 $ C4 ;
CBO = AS $ C4 ;
```

Troço do código CUPL com a atribuição dos pinos de entrada e saída e corpo do programa.

- A_i – Bit i do Operando A
- B_i – Bit i do Operando B
- BI_i – Bit i do Operando B invertido
- C_{i+1} – Carry-out do módulo-i do Full-adder
- S_i – Bit i do Resultado Soma-subtração



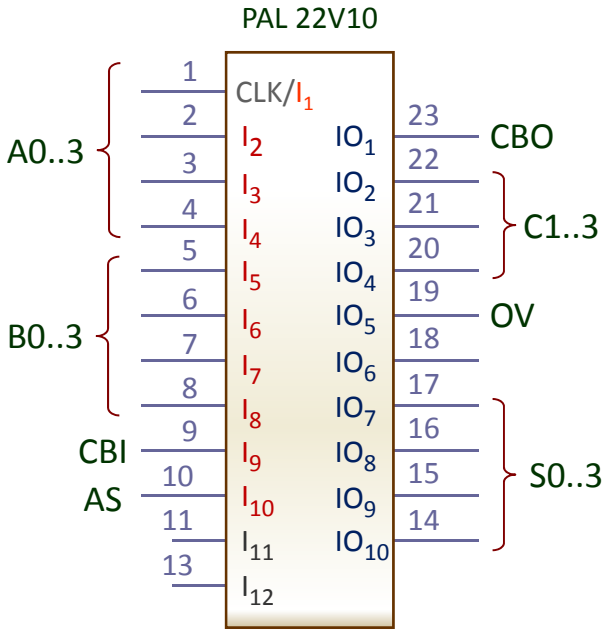
Símbolo lógico da PAL assinalando os 10 pinos de entrada e os 9 pinos de saída do SOMADOR-SUBTRATOR.

- A0..3 – Bits do Operando A
- B0..3 – Bits do Operando B
- C1..3 – Bits de Carry-out parciais
- CBI – Carry-in/Borrow-in do SOMADOR-SUBTRACTOR
- CBO – Carry-out/Borrow-out do SOMADOR-SUBTRACTOR
- S0..3 – Bits do resultado SOMA-SUBTRACÇÃO
- AS – Comutador SOMA-SUBTRACÇÃO

SOMADOR-SUBTRACTOR – TROÇO DO CÓDIGO CUPL E CONFIGURAÇÃO DA PAL (2)

```
Device   p22v10       ;
/* ***** Input Pins ******** */
PIN [ 1..4 ]  = [ A0.. 3 ] ; /*Operando A*/
PIN [ 5..8 ]  = [ B0.. 3 ] ; /*Operando B*/
PIN  9        =  CBI       ; /* CyBwi */
PIN  10       =  AS        ; /*Add-subtract*/
/* ***** Output Pins ******* */
PIN [ 17..14 ] = [ S0..3 ] ; /*Resultado Soma*/
PIN [ 22..20 ] = [ C1..3 ] ; /*Carry-out 1-3*/
PIN  23        = CBO       ; /* CyBwo */
PIN  19        = OV        ; /*Flag Overflow*/
/* ********* Body ********** */
C0 = CBI $ AS
FUNCTION ADC(A, BI, CBin, CBout) {
   CBout = (A $ BI) & CBin # (A & BI);
   ADC = A $ BI$ CBin;
}
$REPEAT i = [0..3]
BI{i} = B{i} $ AS;
S{i} = ADC(A{i}, BI{i}, C{i}, C{i+1});
$REPEND

OV  = C3 $ C4 ;
CBO = AS $ C4 ;
```

Troço de uma outra variante do código CUPL em tudo idêntica à anterior mas encapsulando a lógica do SOMADOR dentro da função ADC evocada com a sintaxe da palavra FUNCTION.



Símbolo lógico da PAL assinalando os 10 pinos de entrada e os 9 pinos de saída do SOMADOR-SUBTRATOR.

- A0..3 – Bits do Operando A
- B0..3 – Bits do Operando B
- C1..3 – Bits parciais de Carry-out
- CBI – Carry-in/Borrow-in do SOMADOR-SUBTRACTOR
- CBO – Carry-out/Borrow-out do SOMADOR-SUBTRACTOR
- S0..3 – Bits do resultado SOMA-SUBTRACÇÃO
- AS – Comutador SOMA-SUBTRACÇÃO

LSD – 5 ÍNDICE 1

LSD – 5 ÍNDICE 3