Circuitos comparadores

Um circuito comparador permite determinar se dois números binários são iguais, e não o sendo, qual deles é o maior.

Comparador de números de 2 bits

O circuito compara os números A (A_1, A_0) e B (B_1,B_0), indicando nas saídas se A>B(GT), A=B (EQ) ou A<B(LT), mutuamente exclusivas, isto é, apenas uma destas saídas pode estar activa.

Comparador de 2 bits

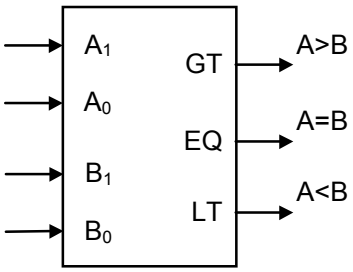


Tabela funcional

A_1	B_1	A_0	B_0	GT (A>B)	EQ (A=B)	LT (A<B)
$A_1>B_1$		X	X	1	0	0
$A_1<B_1$		X	X	0	0	1
$A_1=B_1$		$A_0>B_0$		1	0	0
$A_1=B_1$		$A_0<B_0$		0	0	1
$A_1=B_1$		$A_0=B_0$		0	1	0

Estrutura interna

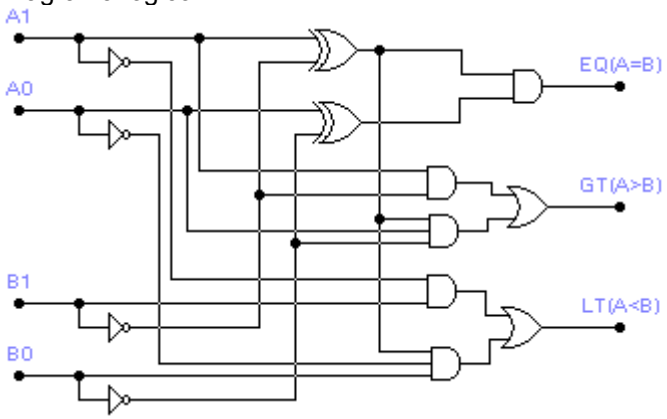
Da leitura da tabela funcional resulta que:

GT(A>B)=1 qd
$\Leftrightarrow(A_1>B_1)$ ou $(A_1=B_1$ e $A_0>B_0)$
$\Leftrightarrow(A_1B'_1)+((A_1\oplus B_1)' \cdot (A_0B'_0))$
$\Leftrightarrow(A_1B'_1)+((A_1\oplus B'_1) \cdot (A_0B'_0))$

EQ(A=B)=1 qd
$\Leftrightarrow(A_1=B_1)$ e $(A_0=B_0)$
$\Leftrightarrow(A_1\oplus B_1)' \cdot (A_0\oplus B_0)'$
$\Leftrightarrow(A_1\oplus B'_1) \cdot (A_0\oplus B'_0)$

LT(A<B)=1 qd
$\Leftrightarrow(A_1<B_1)$ ou $(A_1=B_1$ e $A_0<B_0)$
$\Leftrightarrow(A'_1B_1)+((A_1\oplus B_1)' \cdot (A'_0B_0))$
$\Leftrightarrow(A'_1B_1)+((A_1\oplus B'_1) \cdot (A'_0B_0))$

Diagrama lógico



Note-se que as saídas GT, EQ, LT são mutuamente exclusivas pelo que se poderia redefinir uma delas em função das restantes duas, como por exemplo, LT=GT' . EQ' ou EQ=LT' . GT', ou ainda, GT=EQ' . LT' .

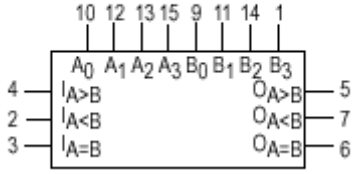
Comparador disponível sob a forma de CI

Exemplo:

- 54/74/XXX85 – 4 Bit Magnitude Comparator

  Principais funcionalidades:
  - comparador de 2 números de 4 bits ($A_3..A_0$; $B_3..B_0$);
  - entradas adicionais para permitir a cascata ($I_{A>B}$, $I_{A<B}$, $I_{A=B}$);
  - saídas activas a um ($O_{A>B}$, $O_{A<B}$, $O_{A=B}$);

LOGIC SYMBOL

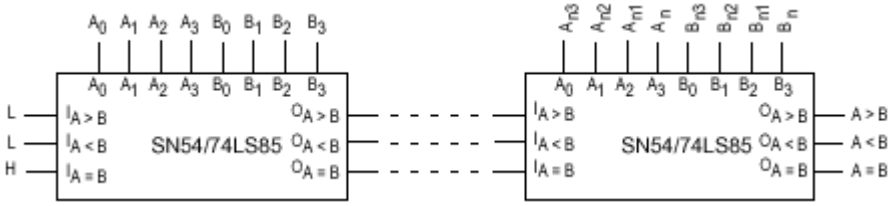


TRUTH TABLE

COMPARING INPUTS				CASCADING INPUTS			OUTPUTS		
A_3,B_3	A_2,B_2	A_1,B_1	A_0,B_0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	$O_{A>B}$	$O_{A<B}$	$O_{A=B}$
$A_3>B_3$	X	X	X	X	X	X	H	L	L
$A_3<B_3$	X	X	X	X	X	X	L	H	L
$A_3=B_3$	$A_2>B_2$	X	X	X	X	X	H	L	L
$A_3=B_3$	$A_2<B_2$	X	X	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1>B_1$	X	X	X	X	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1<B_1$	X	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	A_1=B1	$A_0>B_0$	X	X	X	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0<B_0$	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	H	L	L	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	H	L	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	X	X	H	L	L	H
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	H	H	L	L	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	L	L	H	H	L

Comparador de números de 4xN bits

Implementa-se uma cascata de N CI's 7485.



Circuitos aritméticos

Existem vários circuitos aritméticos sob a forma de CI, tais como somadores (ou adicionadores), multiplicadores, existindo também unidades combinadas que permitem seleccionar qual a operação a efectuar. Alguns destes circuitos requerem lógica sequencial, pelo que ainda não poderão ser analisados neste contexto.

Somadores (adicionadores)

Um circuito somador é um circuito que produz a soma de dois números que lhe são fornecidos num determinado código binário. São vários os tipos de circuitos somadores em função do número de bits e do tipo de código binário utilizado nas parcelas e na respectiva soma. Considera-se, a título exemplificativo, apenas os circuitos somadores de números expressos em código binário natural.

Exemplo:
A= $11_{(10)}=1011_{(2)}$
B= $14_{(10)}= 1110_{(2)}$

	←1←	←1←	←1←	←0←	←Transporte(“Carry”)
	1	0	1	1	(11)
+	1	1	1	0	(14)
1	1	0	0	1	(25)

Meio adicionador (“half-adder”)

Um circuito meio adicionador realiza a soma de dois números de um bit (A_i, B_i), produzindo, tal como acontece na soma com números decimais, um eventual transporte a ser considerado na posição seguinte (C_{i+1}).

Tabela funcional

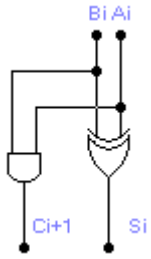
A_i	B_i	S_i	C_{i+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Da tabela funcional, obtém-se:

$S_i=A_i \oplus B_i$

$C_{i+1}=A_i \ . \ B_i$

Resultando no diagrama lógico:



Na realização de uma soma de números de N bits em código binário natural, o circuito meio adicionador apenas pode ser utilizado na soma dos dois bits menos significativos, pois este tipo de circuito não considera um eventual transporte de posições anteriores.

Adicionador completo (“full-adder”)

Um circuito adicionador completo realiza a soma de dois números de um bit (A_i, B_i), considerando um transporte (C_i) de posições anteriores e produzindo um transporte a ser considerado na posição seguinte (C_{i+1}).

Tabela funcional

C_i	A_i	B_i	S_i	C_{i+1}
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

Da tabela funcional, obtém-se:

$S_i = A_i \oplus B_i \oplus C_i$

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

Diagrama de blocos

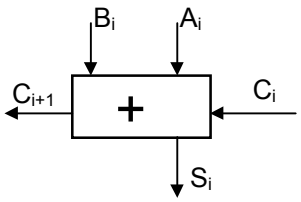
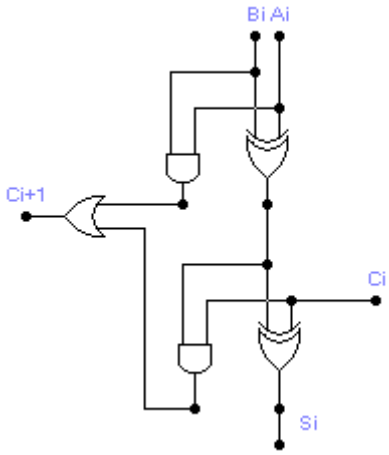


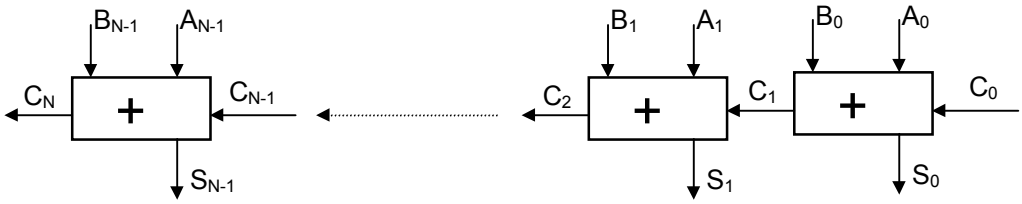
Diagrama lógico



O circuito adicionador completo é construído com base em dois circuitos meios adicionadores e uma porta OR alimentada pelos dois transportes resultantes.

Adicionador de números de N bits

Com base nos adicionadores completos, é possível construir um circuito adicionador de N bits, colocando-os em cascata ("ripple adder"), tal como se representa na figura.

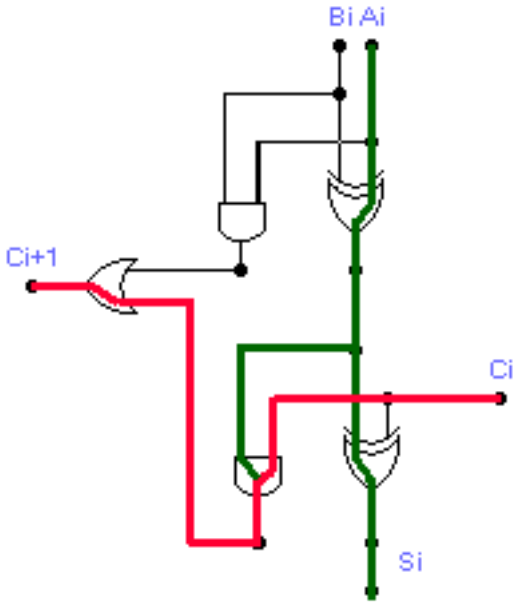


Qualquer alteração nas entradas B_0, A_0 ou C_0 provoca a propagação dos transportes $C_1..C_{N-1}$ ao longo da cascata. Esta solução caracteriza-se por apresentar um tempo de propagação elevado e variável em função do número de bits.

Considerando a estrutura interna dum adicionador completo, verifica-se que, relativamente à alteração na entrada A_i:

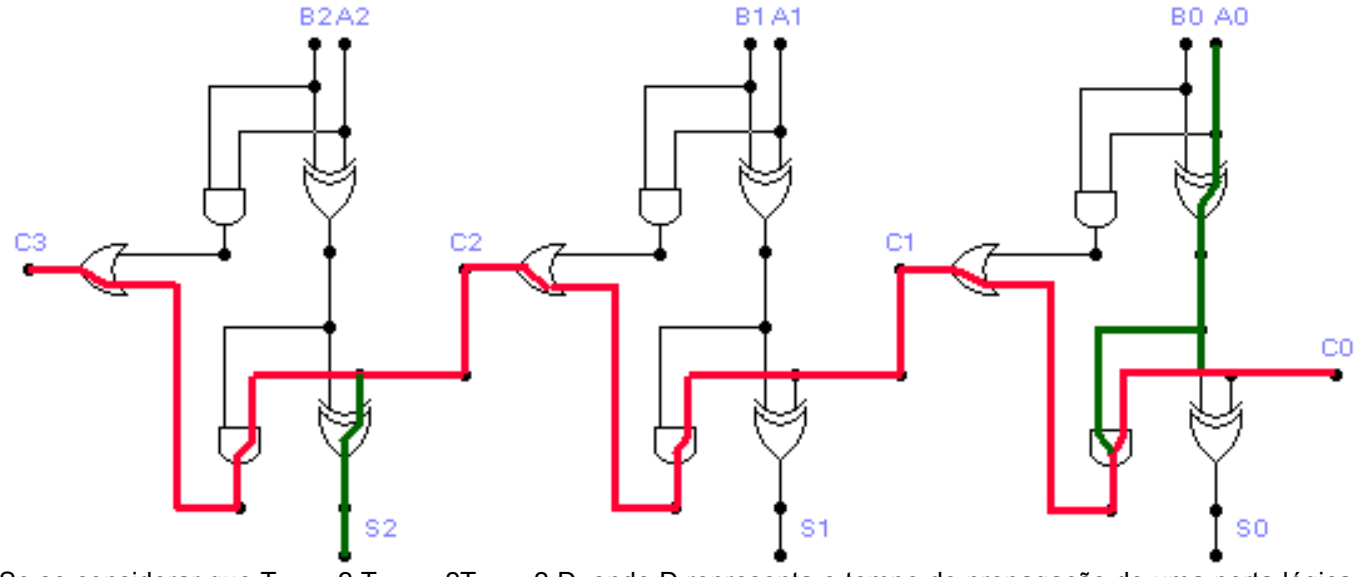
sendo,
T_{XOR} = tempo de propagação de uma porta XOR,
T_{AND} = tempo de propagação de uma porta AND,
T_{OR} = tempo de propagação de uma porta OR,
$T_{XOR}>T_{AND}$,

então,
o tempo de propagação na saída C_{i+1} = $T_{XOR}+T_{AND}+T_{OR}$,
o tempo de propagação na saída Si=2 T_{XOR}.



Para um adicionador em cascata de N bits obtém-se:

- o tempo de propagação na saída C_N = $T_{XOR}+N(T_{AND}+T_{OR})$;
- o tempo de propagação na saída S_{N-1}=2 T_{XOR} + (N-1)($T_{AND}+T_{OR}$)



Se se considerar que T_{XOR}= 2 T_{AND} = 2T_{OR} = 2 D, onde D representa o tempo de propagação de uma porta lógica, então, o tempo máximo necessário para calcular o resultado é de D(2N+2).

Adicionadores com transporte antecipado ("Carry look-ahead"/"Fast carry")

Conforme foi referido anteriormente, nos adicionadores em cascata, o tempo de propagação é proporcional ao número de bits do adicionador. Para eliminar esta desvantagem, são construídos circuitos adicionadores com transporte antecipado.

Para cada somador completo:

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

Ao definir-se:

$G_i = A_i \ . \ B_i$ $\Rightarrow$ transporte gerado

$P_i = A_i \oplus B_i$ $\Rightarrow$ transporte propagado

Obtém-se:

$C_{i+1} = G_i + C_i \ P_i$

Escrevendo as expressões de S_0, C_1, S_1, C_2, etc. , em função de P_i e G_i, obtém-se:

$S_0 = A_0 \oplus B_0 \oplus C_0 = P_0 \oplus C_0$

$C_1 = G_0 + P_0 \ C_0$

$S_1 = A_1 \oplus B_1 \oplus C_1 = A_1 \oplus B_1 \oplus (G_0 + P_0 \ C_0) = P_1 \oplus (G_0 + P_0 \ C_0)$

$C_2 = G_1 + P_1 \ C_1 = G_1 + P_1 \ (G_0 + P_0 \ C_0) = G_1 + P_1 \ G_0 + P_1 \ P_0 \ C_0$

$S_2 = A_2 \oplus B_2 \oplus C_2 = A_2 \oplus B_2 \oplus (G_1 + P_1 \ G_0 + P_1 \ P_0 \ C_0)$

$C_3 = G_2 + P_2 \ C_2 = G_2 + P_2 \ (G_1 + P_1 \ G_0 + P_1 \ P_0 \ C_0) = G_2 + P_2 \ G_1 + P_2 \ P_1 \ G_0 + P_2 \ P_1 \ P_0 \ C_0$

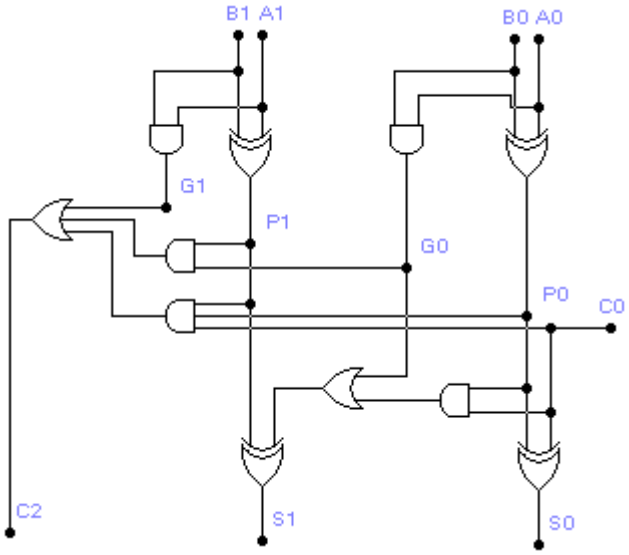
.....

.....

Os sinais de saída, S_i e C_{i+1}, dependem, agora, directamente dos sinais A_i, B_i, P_i, G_i e C_0, onde P_i e G_i dependem directamente de A_i e B_i. Assim sendo, o tempo de propagação do adicionador é constante e independente do número de bits do adicionador. Claro que isto tem um custo, o significativo aumento da complexidade do circuito (número de portas lógicas necessárias para determinar os transportes antecipados) à medida que o número de bits do adicionador aumenta.

Estrutura interna de um adicionador de 2 bits com transporte antecipado

Considerando as expressões obtidas anteriormente para S_0, S_1, C_2, resulta o diagrama lógico da figura.



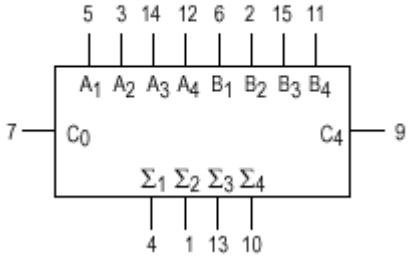
Adicionador disponível sob a forma de CI

Exemplo:

- 54/74/XXX283 – 4 Bit Full Adder With Fast Carry

  Principais funcionalidades:
  - adicionador de 2 números de 4 bits ($A_4..A_1$; $B_4..B_1$) com transporte antecipado;
  - saídas (Σ_1, Σ_2, Σ_3, Σ_4);
  - entrada C_0 e saída C_4 para permitir a cascata;

LOGIC SYMBOL



FUNCTIONAL TRUTH TABLE

C (n−1)	A_n	B_n	Σ_n	C_n
L	L	L	L	L
L	L	H	H	L
L	H	L	H	L
L	H	H	L	H
H	L	L	H	L
H	L	H	L	H
H	H	L	L	H
H	H	H	H	H

$C_1 – C_3$ are generated internally
C_0 is an external input
C_4 is an output generated internally

Pode operar com entradas e saídas activas a um ou a zero, ou seja, considerando, lógica positiva ou lógica negativa.

Exemplo:

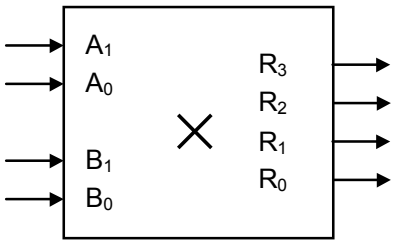
	C_0	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4	Σ_1	Σ_2	Σ_3	Σ_4	C_4	
logic levels	L	L	H	L	H	H	L	L	H	H	H	L	L	H	
Active HIGH	0	0	1	0	1	1	0	0	1	1	1	0	0	1	(10+9=19)
Active LOW	1	1	0	1	0	0	1	1	0	0	0	1	1	0	(carry+5+6=12)

Multiplicadores

Existem várias soluções para a construção de circuitos multiplicadores, nomeadamente no que se refere ao tipo de lógica utilizada, combinacional ou sequencial. Apenas se analisa um circuito multiplicador que tem por base circuitos adicionadores e alguma lógica combinacional.

Multiplicador de números de 2 bits

O circuito multiplicador da figura realiza o produto de dois números de dois bits (A_1 A_0, B_1 B_0), produzindo um resultado de 4 bits (R_3..R_0). Ambos os operandos e o resultado são definidos em código binário natural.



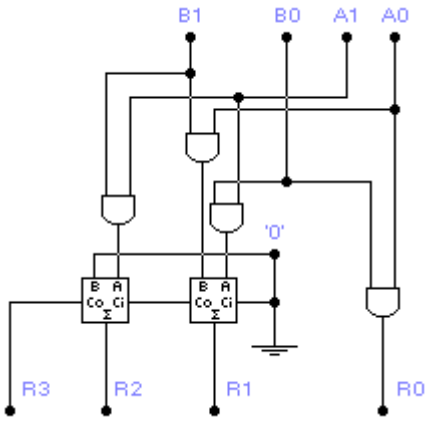
Na implementação deste circuito consideram-se as operações lógicas descritas na figura seguinte, utilizando circuitos adicionadores completos para as operações assinaladas a cinzento.

			A_1	A_0
		X	B_1	B_0
		0	$A_1 . B_0$	$A_0.B_0$
+		$A_1 . B_1$	$A_0 . B_1$	
	R_3	R_2	R_1	R_0

Tendo por base o seguinte adicionador completo:

C_i=Transporte anterior (“Carry in”)
C_o=Transporte seguinte(“Carry out”)

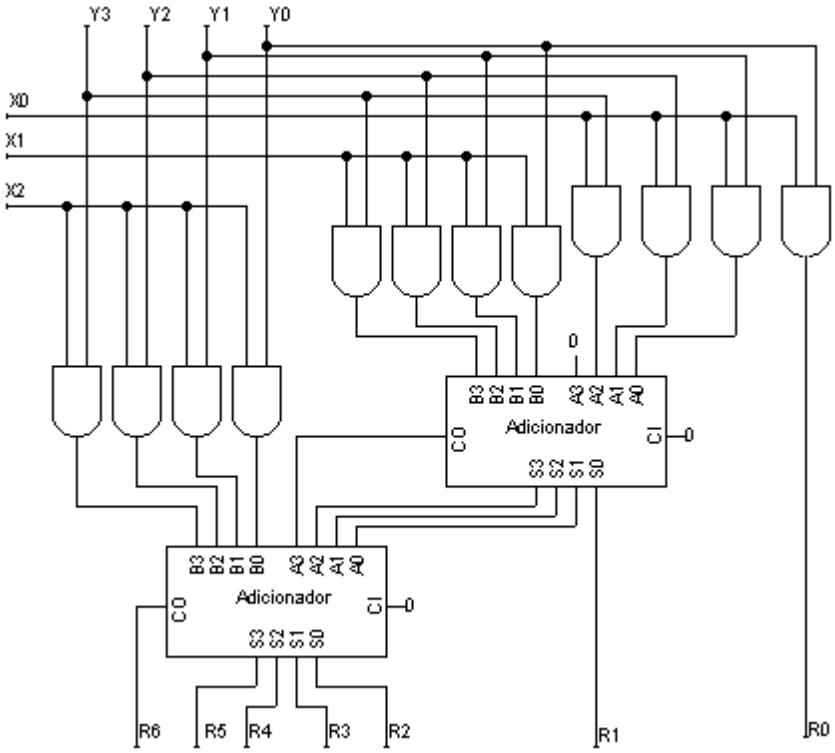
Obtém-se o diagrama lógico:



Multiplicador de 4 por 3 bits

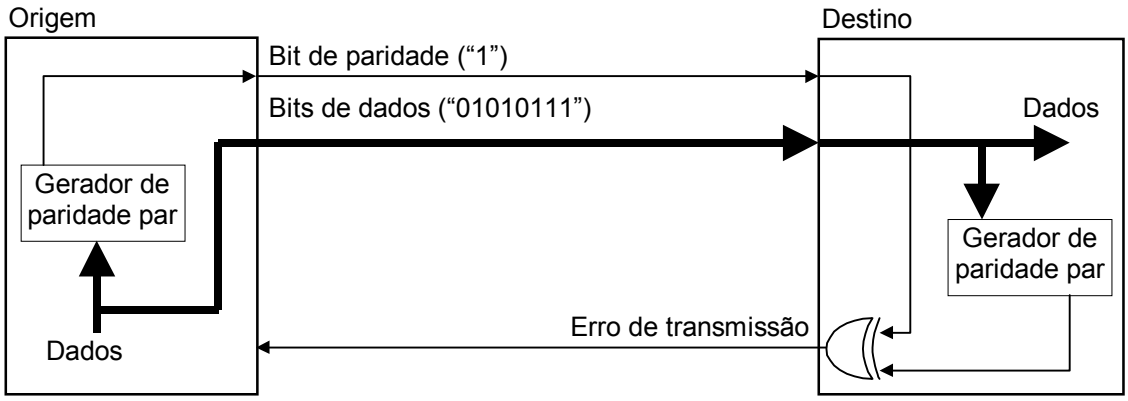
O circuito da figura multiplica um número de 4 bits ($Y_3..Y_0$) por um número de 3 bits ($X_2..X_0$) colocando o resultado em $R_6..R_0$.

			Y_3	Y_2	Y_1	Y_0
		x		X_2	X_1	X_0
		0	$X_0 . Y_3$	$X_0 . Y_2$	$X_0 . Y_1$	$X_0.Y_0$
	+	$X_1 . Y_3$	$X_1 . Y_2$	$X_1 . Y_1$	$X_1.Y_0$	
+	$X_2 . Y_3$	$X_2 . Y_2$	$X_2 . Y_1$	$X_0.Y_0$		
R_6	R_5	R_4	R_3	R_2	R_1	R_0



Geradores/Detectores de paridade

Este tipo de circuito tem particular aplicabilidade na detecção de alguns tipos de erros em sistemas de transmissão de dados binários.
Considere-se o sistema de transmissão descrito no diagrama de blocos da figura.



Ao longo da transmissão, em série ou em paralelo, de um conjunto de bits da origem para o destino, um dos bits de dados pode ser afectado por, por exemplo, ruído eléctrico, e na origem ter o valor 1 e ser detectado no destino como tendo o valor 0. Ao conjunto de bits de dados é adicionado um bit de paridade que sendo também ele transmitido para o destino é utilizado na detecção de eventuais erros. No protocolo de transmissão de dados, a origem e o destino utilizam o mesmo tipo de paridade:

- paridade par (“even parity”) – número par de uns, incluindo o próprio bit de paridade;
- paridade ímpar (“odd parity”) – número impar de uns, incluindo o próprio bit de paridade;

Gerador/Detector de paridade de 4 bits

O gerador/detector para 4 bits produz a saída (I) que indica se o número de 1’s nos 4 bits é par. O complemento desta saída (P) indica o caso contrário, ou seja, se o número de 1’s nos 4 bits é ímpar.

Tabela de verdade

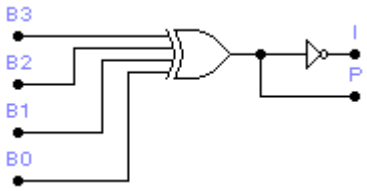
B_3	B_2	B_1	B_0	I	P
0	0	0	0	1	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	0	1
1	0	0	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	0	1
1	1	1	0	0	1
1	1	1	1	1	0

Mapa de Karnaugh

B_3 \ B_2 \ B_1 \ B_0			
1	0	1	0
0	1	0	1
1	0	1	0
0	1	0	1

$I(B3,B2,B1,B0)= FMSP = \sum m(0,3,5,6,9,10,12,15)$
$=(B_3 \oplus B_2 \oplus B_1 \oplus B_0)'$
$P(B3,B2,B1,B0)= FMSP = \sum m(1,2,4,7,8,11,13,14)$
$=(B_3 \oplus B_2 \oplus B_1 \oplus B_0)$

Estrutura interna



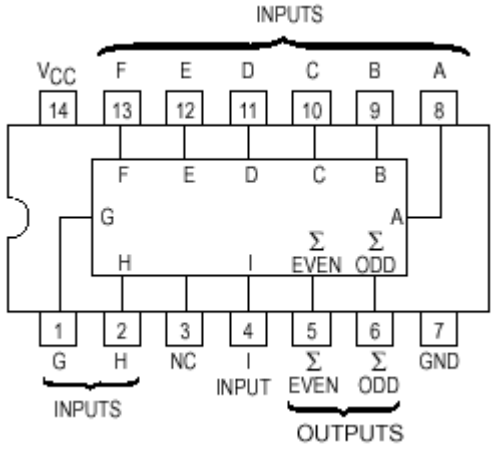
As saídas P e I podem ser utilizadas para gerar paridade par e ímpar respectivamente.

Gerador/Detector de paridade disponível sob a forma de CI

Exemplo:

- 54/74/XXX280 – 9-Bit Odd/Even Parity Generators/Checkers

  Principais funcionalidades:
  - gerador/detector de paridade de 9 bits (A .. I).
  - saída ∑EVEN indica se existe um número par de 1's nas entradas A .. I.
  - saída ∑ODD indica se existe um número ímpar de 1's nas entradas A .. I.



FUNCTION TABLE

NUMBER OF INPUTS A THRU 1 THAT ARE HIGH	OUTPUTS	
	∑EVEN	∑ODD
0, 2, 4, 6, 8	H	L
1, 3, 5, 7, 9	L	H