Circuitos Lógicos e Organização de Computadores

Capítulo 5 – Representação Numérica e Circuitos Aritméticos

Ricardo Pannain

pannain@puc-campinas.edu.br

http://docentes.puc-campinas.edu.br/ceatec/pannain/

Conversão Decimal-Binária

Convert $(857)_{10}$

			Remainder	
$857 \div 2$	=	428	1	LSB
$428 \div 2$	=	214	0	
$214 \div 2$	=	107	0	
$107 \div 2$	=	53	1	
$53 \div 2$	=	26	1	
$26 \div 2$	=	13	0	
$13 \div 2$	=	6	1	
$6 \div 2$	=	3	0	
$3 \div 2$	=	1	1	
$1 \div 2$	=	0	1	MSB

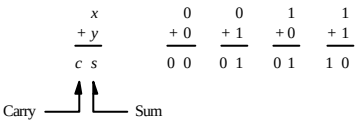
Result is $(1101011001)_2$

Números em diferentes Bases

Decimal	Binary	Octal	Hexadecimal
0_0	0000_0	0_0	0_0
0_1	0000_1	0_1	0_1
0_2	0001_0	0_2	0_2
0_3	0001_1	0_3	0_3
0_4	0010_0	0_4	0_4
0_5	0010_1	0_5	0_5
0_6	0011_0	0_6	0_6
0_7	0011_1	0_7	0_7
0_8	0100_0	1_0	0_8
0_9	0100_1	1_1	0_9
1_0	0101_0	1_2	0_A
1_1	0101_1	1_3	0_B
1_2	0110_0	1_4	0_C
1_3	0110_1	1_5	0_D
1_4	0111_0	1_6	0_E
1_5	0111_1	1_7	0_F
1_6	1000_0	2_0	1_0
1_7	1000_1	2_1	1_1
1_8	1001_0	2_2	1_2

3

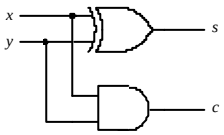
Meio somador (Half-adder)



(a) The four possible cases

x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(b) Truth table



(c) Circuit

(d) Graphical symbol

4

Adição Binária

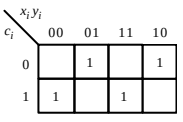
$X = x_4x_3x_2x_1x_0$	01111	?15 ??$_{10}$
$+ Y = y_4y_3y_2y_1y_0$	01010	?10 ??$_{10}$
	1110 ← Generated carries	
$S = s_4s_3s_2s_1s_0$	11001	?25 ??$_{10}$

5

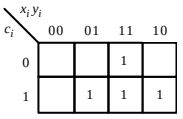
Somador Completo (Full-adder)

c_i	x_i	y_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(a) Truth table

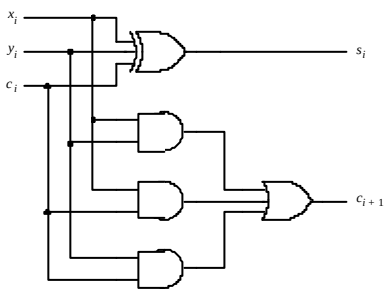


$s_i = x_i ? ?y_i ? ?c_i$



$c_{i+1} = x_iy_i + x_ic_i + y_ic_i$

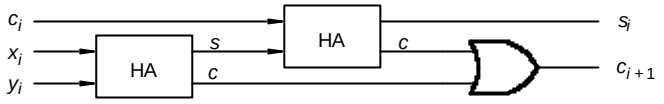
(b) Karnaugh maps



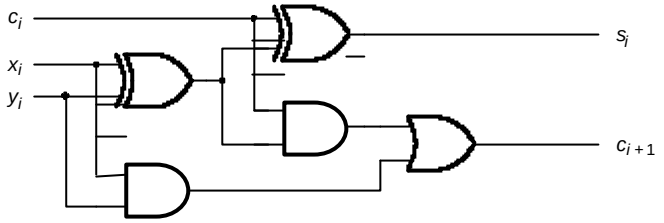
(c) Circuit

6

Implemantação de um Somador Completo usando decomposição



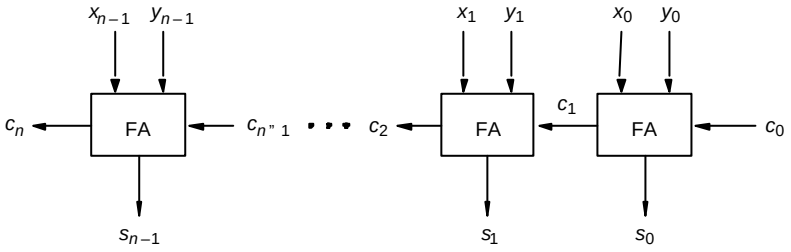
(a) Block diagram



(b) Detailed diagram

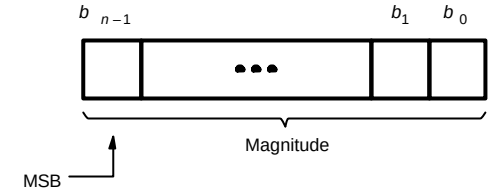
7

Somador de *n*-bits tipo ripple-carry

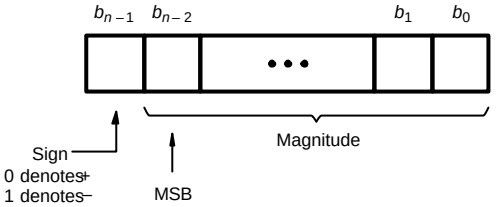


8

Formatos para representação de números inteiros



(a) Unsigned number



(b) Signed number

9

Números inteiros sinalizados com 4 bits

$b_3b_2b_1b_0$	Sign and magnitude	1 s complement	2 s complement
0111	+7	+7	+7
0110	+6	+6	+6
0101	+5	+5	+5
0100	+4	+4	+4
0011	+3	+3	+3
0010	+2	+2	+2
0001	+1	+1	+1
0000	+0	+0	+0
1000	0	7	8
1001	1	6	7
1010	2	5	6
1011	3	4	5
1100	4	3	4
1101	5	2	3
1110	6	1	2
1111	7	0	1

10

Exemplos de adição em complemento de 1

(+5)	0 1 0 1	?–5?	1 0 1 0
+(+2)	+ 0 0 1 0	+(+2)	+ 0 0 1 0
(+7)	0 1 1 1	?–3?	1 1 0 0

(+5)	0 1 0 1	?–5?	1 0 1 0
+?–2?	+ 1 1 0 1	+?–2?	+ 1 1 0 1
(+3)	1 0 0 1 0	?–7?	1 0 1 1 1
	↳ 1		↳ 1
	0 0 1 1		1 0 0 0

Exemplos de adição em complemento de 2

(+5)	0 1 0 1	?–5?	1 0 1 1
+ (+2)	+ 0 0 1 0	+ (+2)	+ 0 0 1 0
(+7)	0 1 1 1	?–3?	1 1 0 1

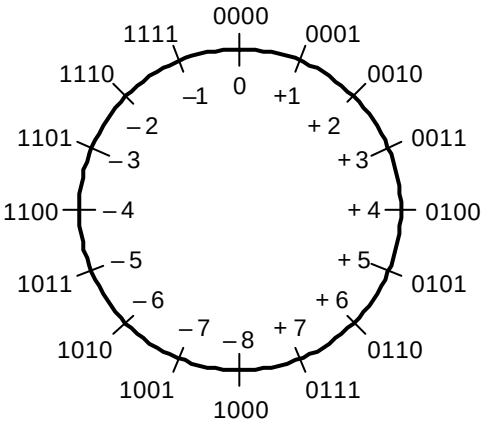
(+5)	0 1 0 1	?–5?	1 0 1 1
+ ?–2?	+ 1 1 1 0	+ ?–2?	+ 1 1 1 0
(+3)	1 0 0 1 1	?–7?	1 1 0 0 1
	↑		↑
	ignore		ignore

Exemplos de subtração em complemento de 1

(+5)	0 1 0 1		0 1 0 1
– (+2)	– 0 0 1 0	⟹	+ 1 1 1 0
(+3)			1 0 0 1 1
			↑ ignore
?–5?	1 0 1 1		1 0 1 1
– (+2)	– 0 0 1 0	⟹	+ 1 1 1 0
?–7?			1 1 0 0 1
			↑ ignore
(+5)	0 1 0 1		0 1 0 1
– ?–2?	– 1 1 1 0	⟹	+ 0 0 1 0
(+7)			0 1 1 1
?–5?	1 0 1 1		1 0 1 1
– ?–2?	– 1 1 1 0	⟹	+ 0 0 1 0
?–3?			1 1 0 1

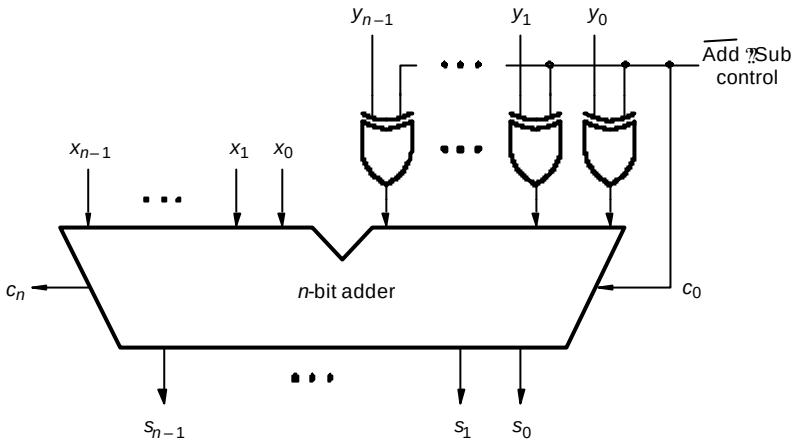
13

Interpretação Gráfica de números de 4 bits em complemento de 2



14

Somador / Subtrator



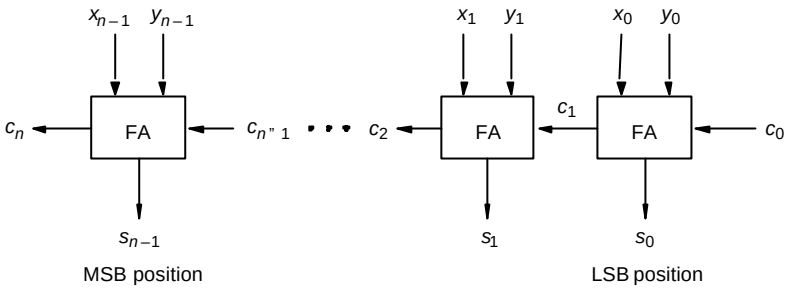
15

Exemplo de ocorrência de overflow

(+7)	0 1 1 1	?–7?	1 0 0 1
+ (+2)	+ 0 0 1 0	+ (+2)	+ 0 0 1 0
(+9)	1 0 0 1	?–5?	1 0 1 1
	$c_4 = 0$		$c_4 = 0$
	$c_3 = 1$		$c_3 = 0$
(+7)	0 1 1 1	?–7?	1 0 0 1
+ ?–2?	+ 1 1 1 0	+ ?–2?	+ 1 1 1 0
(+5)	1 0 1 0 1	?–9?	1 0 1 1 1
	$c_4 = 1$		$c_4 = 1$
	$c_3 = 1$		$c_3 = 0$

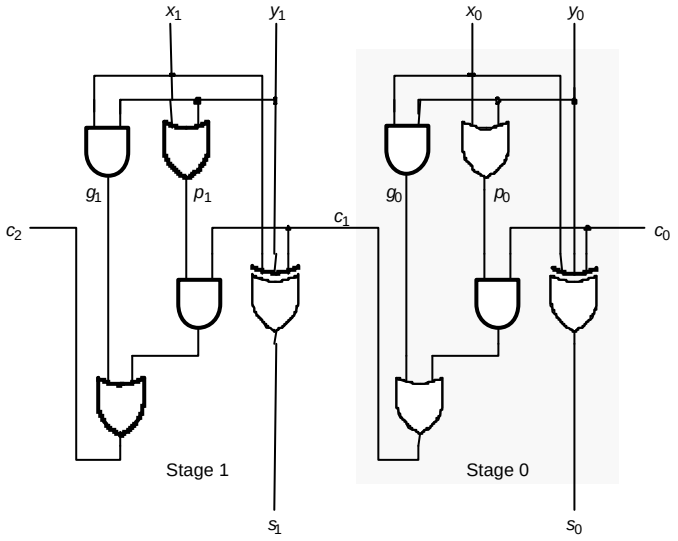
16

Somador de *n*-bits tipo ripple-carry



17

Somador ripple-carry com generação/propagação de sinais



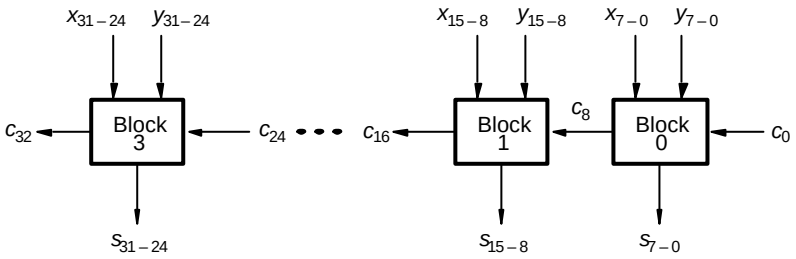
18

Somador carry-lookahead

Como fazer com que um somador trabalhe mais rápido ?

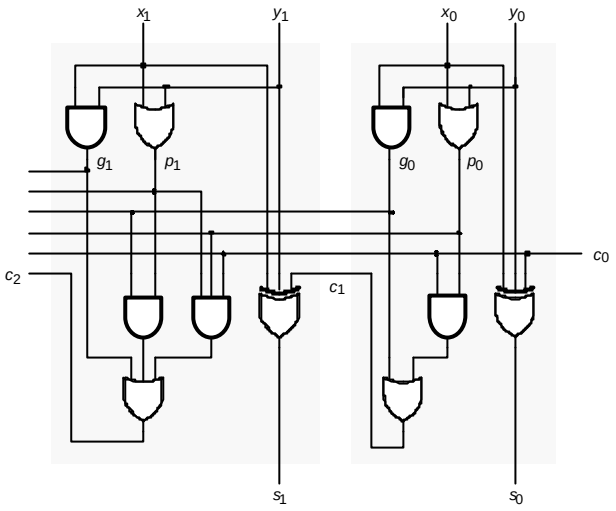
19

Somador carry-lookahead com ripple-carry entre blocos



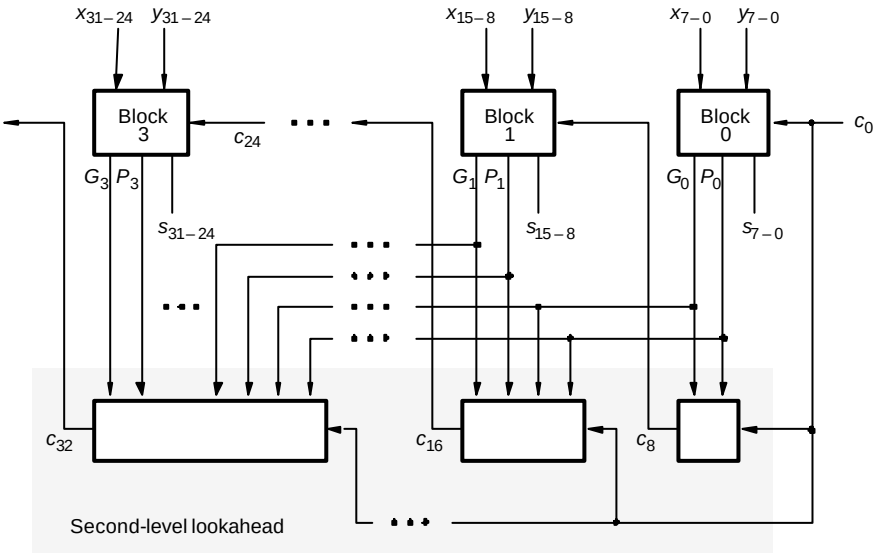
20

Dois estágios de um somador carry-lookahead



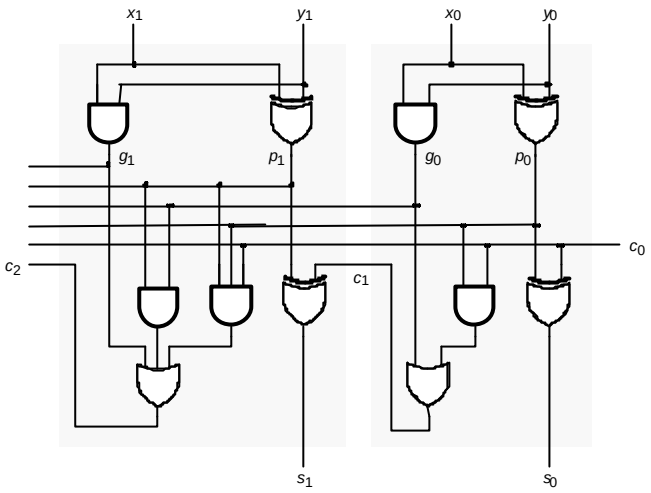
21

Um somador hierárquico carry-lookahead



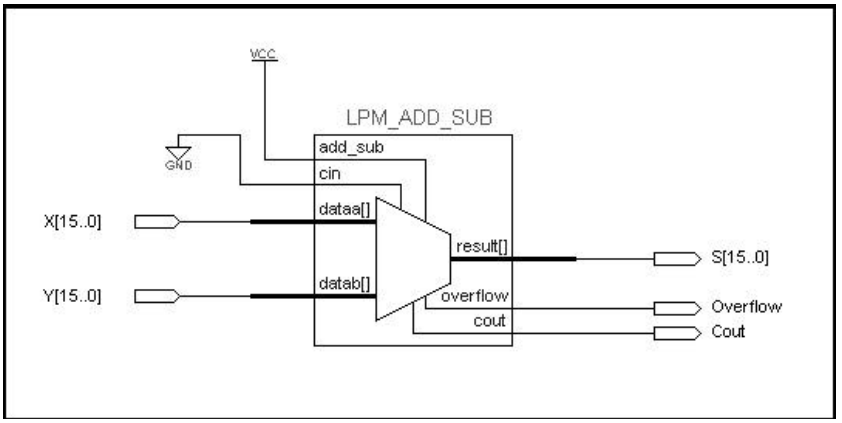
22

Projeto alternativo para um somador carry-lookahead



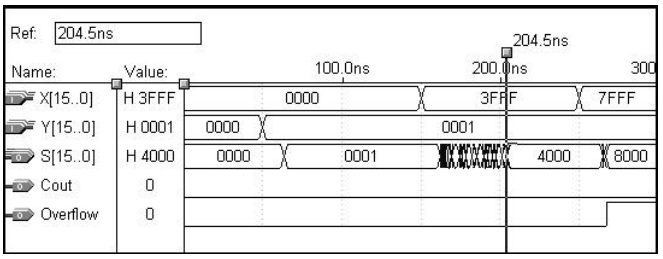
23

Esquemático usando um módulo LPM adder/subtractor

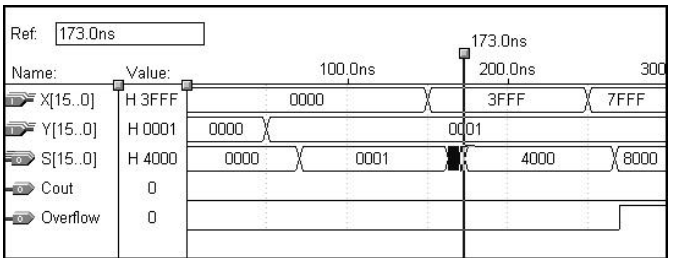


24

Resultado da simulação de um módulo LPM adder/subtrator



Optimized for cost



Optimized for speed

25

Código VHDL para um full-adder

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY fulladd IS
    PORT (  Cin, x, y   : IN      STD_LOGIC ;
            s, Cout     : OUT     STD_LOGIC ) ;
END fulladd ;

ARCHITECTURE LogicFunc OF fulladd IS
BEGIN
    s <= x XOR y XOR Cin ;
    Cout <= (x AND y) OR (Cin AND x) OR (Cin AND y) ;
END LogicFunc ;
```

26

Código VHDL para um somador de 4 bits

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY adder4 IS
    PORT (  Cin             : IN    STD_LOGIC ;
            x3, x2, x1, x0  : IN    STD_LOGIC ;
            y3, y2, y1, y0  : IN    STD_LOGIC ;
            s3, s2, s1, s0  : OUT   STD_LOGIC ;
            Cout            : OUT   STD_LOGIC ) ;
END adder4 ;

ARCHITECTURE Structure OF adder4 IS
    SIGNAL c1, c2, c3 : STD_LOGIC ;
    COMPONENT fulladd
        PORT (  Cin, x, y   : IN    STD_LOGIC ;
                s, Cout     : OUT   STD_LOGIC ) ;
    END COMPONENT ;
BEGIN
    stage0: fulladd PORT MAP ( Cin, x0, y0, s0, c1 ) ;
    stage1: fulladd PORT MAP ( c1, x1, y1, s1, c2 ) ;
    stage2: fulladd PORT MAP ( c2, x2, y2, s2, c3 ) ;
    stage3: fulladd PORT MAP (
        Cin => c3, Cout => Cout, x => x3, y => y3, s => s3  ) ;
END Structure ;
```

27

Declaração de um Package

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

PACKAGE fulladd_package IS
    COMPONENT fulladd
        PORT (  Cin, x, y  : IN    STD_LOGIC ;
                s, Cout    : OUT   STD_LOGIC ) ;
    END COMPONENT ;
END fulladd_package ;
```

28

Usando um package para somador de 4 bits

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE work.fulladd_package.all ;

ENTITY adder4 IS
    PORT (  Cin             : IN    STD_LOGIC ;
            x3, x2, x1, x0  : IN    STD_LOGIC ;
            y3, y2, y1, y0  : IN    STD_LOGIC ;
            s3, s2, s1, s0  : OUT   STD_LOGIC ;
            Cout            : OUT   STD_LOGIC ) ;
END adder4 ;

ARCHITECTURE Structure OF adder4 IS
    SIGNAL c1, c2, c3 : STD_LOGIC ;
BEGIN
    stage0: fulladd PORT MAP ( Cin, x0, y0, s0, c1 ) ;
    stage1: fulladd PORT MAP ( c1, x1, y1, s1, c2 ) ;
    stage2: fulladd PORT MAP ( c2, x2, y2, s2, c3 ) ;
    stage3: fulladd PORT MAP (
            Cin => c3, Cout => Cout, x => x3, y => y3, s
=> s3  ) ;
END Structure ;
```

29

Somador de 4 bits usando sinais multibit

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE work.fulladd_package.all ;

ENTITY adder4 IS
    PORT ( Cin      : IN    STD_LOGIC ;
           X, Y     : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           S        : OUT   STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           Cout     : OUT   STD_LOGIC ) ;
END adder4 ;

ARCHITECTURE Structure OF adder4 IS
    SIGNAL C : STD_LOGIC_VECTOR(1 TO 3) ;
BEGIN
    stage0: fulladd PORT MAP ( Cin, X(0), Y(0), S(0), C(1) ) ;
    stage1: fulladd PORT MAP ( C(1), X(1), Y(1), S(1), C(2) ) ;
    stage2: fulladd PORT MAP ( C(2), X(2), Y(2), S(2), C(3) ) ;
    stage3: fulladd PORT MAP ( C(3), X(3), Y(3), S(3), Cout ) ;
END Structure ;
```

30

Código VHDL code para um somador de 16-bit

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;

ENTITY adder16 IS
      PORT (   X, Y      : IN       STD_LOGIC_VECTOR(15 DOWNTO 0) ;
               S         : OUT      STD_LOGIC_VECTOR(15 DOWNTO 0) ) ;
END adder16 ;

ARCHITECTURE Behavior OF adder16 IS
BEGIN
      S <= X + Y ;
END Behavior ;
```

31

Somador de 16-bit com carry e overflow

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;

ENTITY adder16 IS
      PORT (   Cin              : IN       STD_LOGIC ;
               X, Y             : IN       STD_LOGIC_VECTOR(15 DOWNTO 0) ;
               S                : OUT      STD_LOGIC_VECTOR(15 DOWNTO 0) ;
               Cout, Overflow   : OUT      STD_LOGIC ) ;
END adder16 ;

ARCHITECTURE Behavior OF adder16 IS
      SIGNAL Sum : STD_LOGIC_VECTOR(16 DOWNTO 0) ;
BEGIN
      Sum <= ('0' & X) + Y + Cin ;
      S <= Sum(15 DOWNTO 0) ;
      Cout <= Sum(16) ;
      Overflow <= Sum(16) XOR X(15) XOR Y(15) XOR Sum(15) ;
END Behavior ;
```

32

Use of the arithmetic Uso de package com circuito aritmético

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_arith.all ;

ENTITY adder16 IS
      PORT (  Cin              : IN     STD_LOGIC ;
              X, Y             : IN     SIGNED(15 DOWNTO 0) ;
              S                : OUT    SIGNED(15 DOWNTO 0) ;
              Cout, Overflow   : OUT    STD_LOGIC ) ;
END adder16 ;

ARCHITECTURE Behavior OF adder16 IS
      SIGNAL Sum : SIGNED(16 DOWNTO 0) ;
BEGIN
      Sum <= ('0' & X) + Y + Cin ;
      S <= Sum(15 DOWNTO 0) ;
      Cout <= Sum(16) ;
      Overflow <= Sum(16) XOR X(15) XOR Y(15) XOR Sum(15) ;
END Behavior ;
```

33

Um somador de 16-bit adder usando sinais INTEGER

```
ENTITY adder16 IS
     PORT (   X, Y     : IN      INTEGER RANGE -32768 TO 32767 ;
              S        : OUT     INTEGER RANGE -32768 TO 32767 ) ;
END adder16 ;

ARCHITECTURE Behavior OF adder16 IS
BEGIN
     S <= X + Y ;
END Behavior ;
```

34

Circuito multiplicador 4 X 4

Multiplicand M	(14)	1 1 1 0
Multiplier Q	(11)	? 1 0 1 1
		1 1 1 0
		1 1 1 0
		0 0 0 0
		1 1 1 0
Product P	(154)	1 0 0 1 1 0 1 0

(a) Multiplication by hand

35

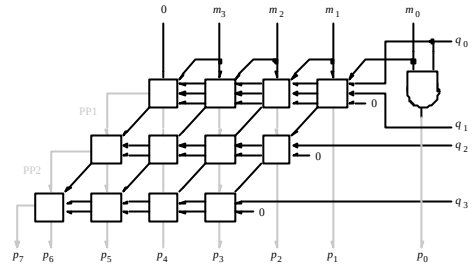
Circuito multiplicador 4 X 4

Multiplicand M	(11)	1 1 1 0
Multiplier Q	(14)	? 1 0 1 1
Partial product 0		1 1 1 0
		+ 1 1 1 0
Partial product 1		1 0 1 0 1
		+ 0 0 0 0
Partial product 2		0 1 0 1 0
		+ 1 1 1 0
Product P	(154)	1 0 0 1 1 0 1 0

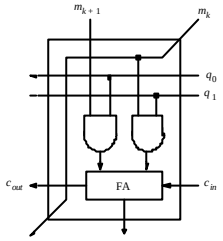
(b) Implemantação da multiplicação em hardware

36

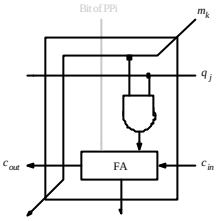
Circuito multiplicador 4 X 4



(a) Structure of the circuit



(b) A block in the top row



(c) A block in the bottom two rows

37

Multiplicação de números sinalizados

Multiplicand M	(+14)	0 1 1 1 0
Multiplier Q	(+11)	x 0 1 0 1 1
Partial product 0		0 00 1 1 1 0
		+ 0 0 1 1 1 0
Partial product 1		0 0 1 0 1 0 1
		+ 0 0 0 0 0 0
Partial product 2		0 0 0 1 0 1 0
		+ 0 0 1 1 1 0
Partial product 3		0 0 1 0 0 1 1
		+ 0 0 0 0 0 0
Product P	(+154)	0 0 1 0 0 1 1 0 1 0

(a) Positive multiplicand

38

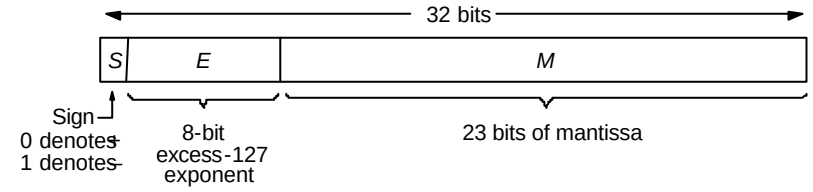
Multiplicação de números sinalizados

Multiplicand M	(–14)	1 0 0 1 0
Multiplier Q	(+11)	??0 1 0 1 1
Partial product 0		1 11 0 0 1 0
		+ 1 1 0 0 1 0
Partial product 1		1 1 0 1 0 1 1
		+ 0 0 0 0 0 0
Partial product 2		1 1 1 0 1 0 1
		+ 1 1 0 0 1 0
Partial product 3		1 1 0 1 1 0 0
		+ 0 0 0 0 0 0
Product P	(–154)	1 1 0 1 1 0 0 1 1 0

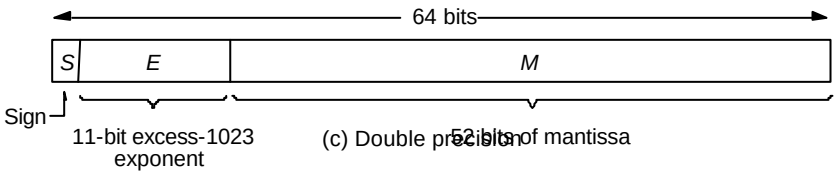
(b) Negative multiplicand

39

Padrão IEEE 754 standard número de ponto flutuante



(a) Single precision



(c) Double precision

40

Código BCD – Binary Code Decimal

Decimal digit	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

41

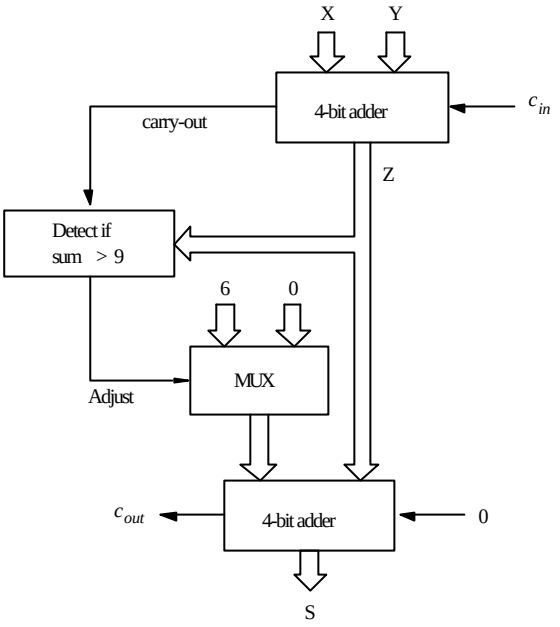
Adição em BCD

X	0 1 1 1	7
+ Y	+ 0 1 0 1	+ 5
Z	1 1 0 0	12
	+ 0 1 1 0	
carry →	1 0 0 1 0	
	S = 2	

X	1 0 0 0	8
+ Y	+ 1 0 0 1	+ 9
Z	1 0 0 0 1	17
	+ 0 1 1 0	
carry →	1 0 1 1 1	
	S = 7	

42

Somador BCD de um dígito – diagrama de blocos



43

Somador BCD de um dígito – código VHDL

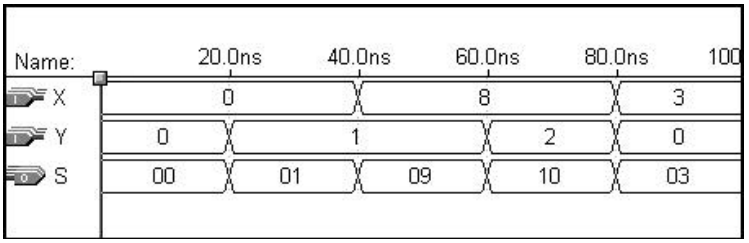
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY BCD IS
    PORT (  X, Y      : IN      STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            S         : OUT     STD_LOGIC_VECTOR(4 DOWNTO 0) ) ;
END BCD ;

ARCHITECTURE Behavior OF BCD IS
    SIGNAL Z : STD_LOGIC_VECTOR(4 DOWNTO 0) ;
    SIGNAL Adjust : STD_LOGIC ;
BEGIN
    Z <= ('0' & X) + Y ;
    Adjust <= '1' WHEN Z > 9 ELSE '0' ;
    S <= Z WHEN (Adjust = '0') ELSE Z + 6 ;
END Behavior ;
```

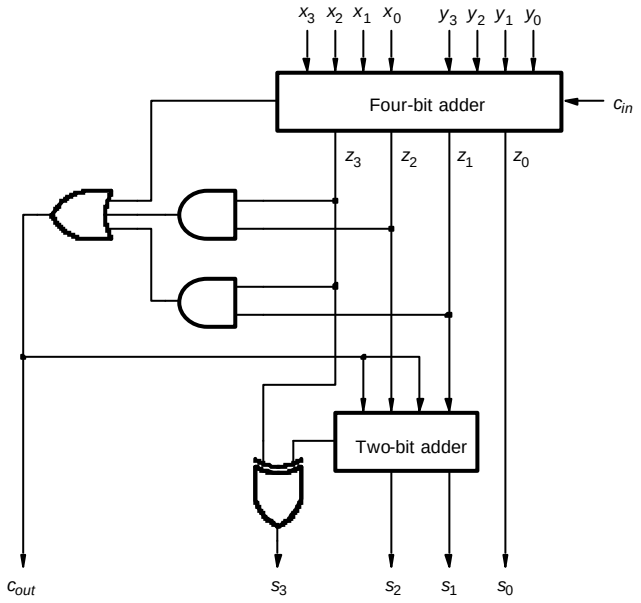
44

Simulação de um somador BCD de um dígito



45

Circuito para um somador BCD de um dígito



46

Bit positions	Bit positions 654							
3210	000	001	010	011	100	101	110	111
0000	NUL	DLE	SPACE	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS		=	M	]	m	}
1110	SO	RS	.	>	N	^	n	
1111	SI	US	/	?	O		o	DEL

NUL	Null/Idle	SI	Shift in
SOH	Start of header	DLE	Data link escape
STX	Start of text	DC1 DC4	Device control
ETX	End of text	NAK	Negative acknowledgement
EOT	End of transmitted	SYN	Synchronous idle
ENQ	Enquiry	ETB	End of transmitted block
ACQ	Acknowledgement	CAN	Cancel (error in data)
BEL	Audible signal	EM	End of medium
BS	Back space	SUB	Special sequence
HT	Horizontal tab	ESC	Escape
LF	Line feed	FS	File separator
VT	Vertical tab	GS	Group separator
FF	Form feed	RS	Record separator
CR	Carriage return	US	Unit separator
SO	Shift out	DEL	Delete/Idle

Bit positions of code format = 6 5 4 3 2 1 0

47