ARITHMETIC COMBINATIONAL MODULES AND NETWORKS

- SPECIFICATION OF ADDER MODULES FOR POSITIVE INTEGERS
- HALF-ADDER AND FULL-ADDER MODULES
- CARRY-RIPPLE AND CARRY-LOOKAHEAD ADDER MODULES
- NETWORKS OF ADDER MODULES
- REPRESENTATION OF SIGNED INTEGERS:
  1. sign-and-magnitude
  2. two's-complement
  3. ones'-complement
- ADDITION AND SUBTRACTION IN TWO'S COMPLEMENT
- ARITHMETIC-LOGIC UNITS (ALU)
- COMPARATOR MODULES AND NETWORKS
- MULTIPLICATION OF POSITIVE INTEGERS

ADDITION OF POSITIVE INTEGERS

- CONVENTIONAL RADIX-2 NUMBER SYSTEM:

$$\underline{x} = (x_{n-1}, \ldots, x_0) \;\Leftrightarrow\; x, \;\; \text{integer}$$

$$x = \sum_{i=0}^{n-1} x_i \times 2^i$$

- RANGE: 0 to $2^n - 1$

ADDER MODULES FOR POSITIVE INTEGERS

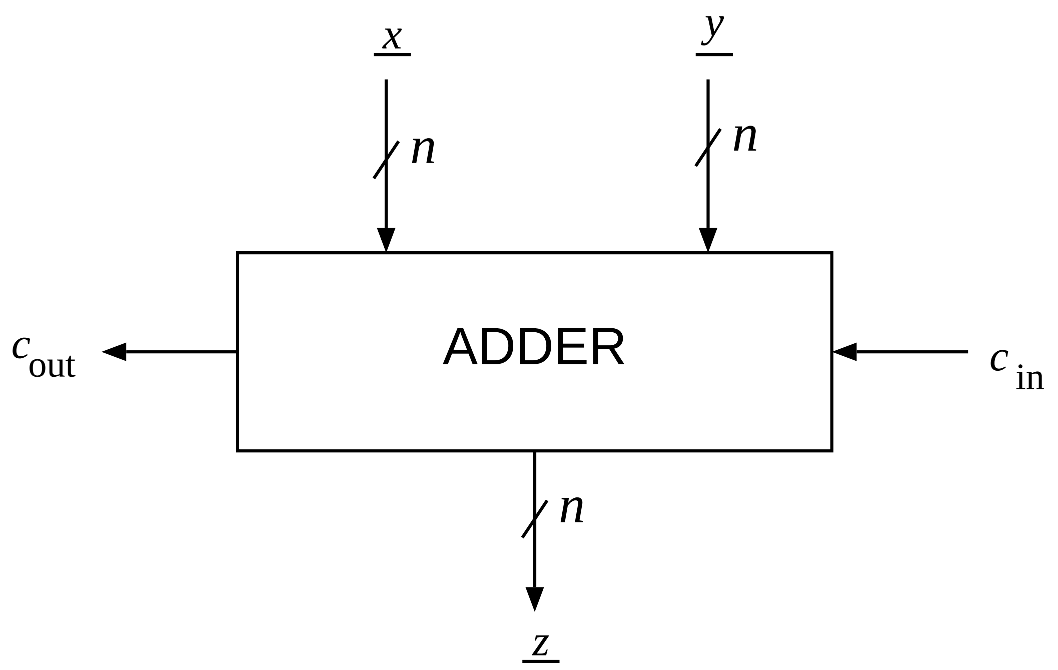


Figure 10.1: ADDER MODULE.

$$x + y + c_{\text{in}} = 2^n c_{\text{out}} + z$$

A HIGH-LEVEL SPECIFICATION OF ADDER MODULE

INPUTS: $\underline{x} = (x_{n-1}, \ldots, x_0), \quad x_j \in \{0,1\}$
$\underline{y} = (y_{n-1}, \ldots, y_0), \quad y_j \in \{0,1\}$
$c_{\text{in}} \in \{0,1\}$

OUTPUTS: $\underline{z} = (z_{n-1}, \ldots, z_0), \quad z_j \in \{0,1\}$
$c_{\text{out}} \in \{0,1\}$

FUNCTIONS: $z = (x + y + c_{\text{in}}) \bmod 2^n$

$$c_{\text{out}} = \begin{cases} 1 & \textbf{if} \quad (x + y + c_{\text{in}}) \geq 2^n \\ 0 & \textbf{otherwise} \end{cases}$$

EXAMPLE for n=5

x	y	c_{in}	z	c_{out}
12	14	1	$(12 + 14 + 1) \bmod 32 = 27$	0 because $(12 + 14 + 1) < 32$
19	14	1	$(19 + 14 + 1) \bmod 32 = 2$	1 because $(19 + 14 + 1) > 32$

CARRY-RIPPLE ADDER IMPLEMENTATION

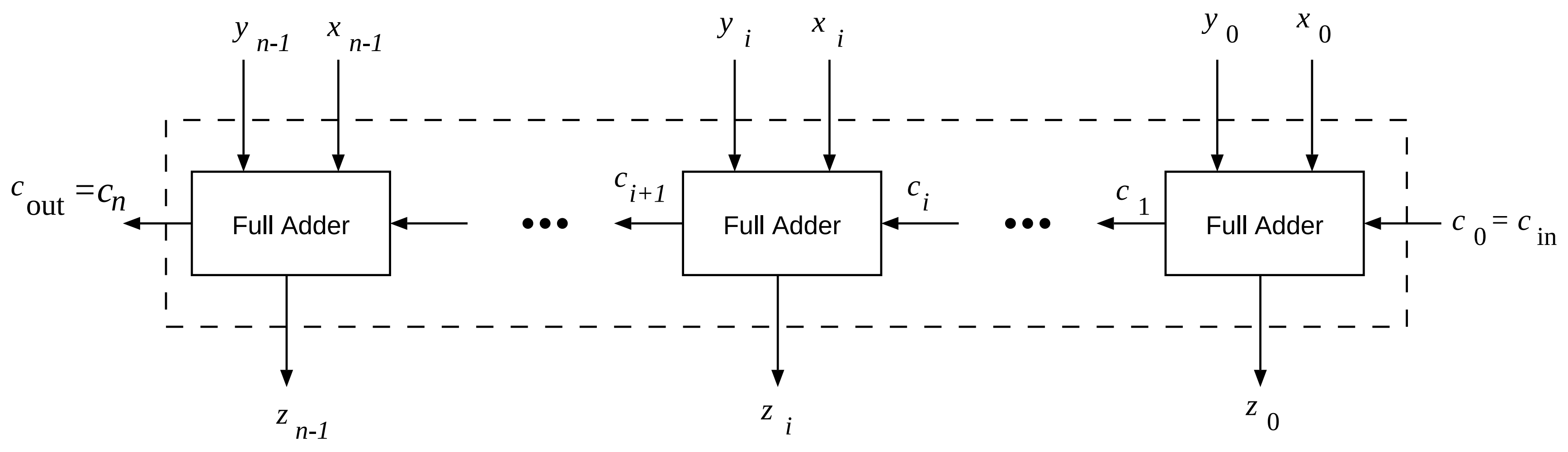


Figure 10.2: CARRY-RIPPLE ADDER MODULE.

- DELAY OF CARRY-RIPPLE ADDER

$$t_p(net) = (n-1)t_c + \max(t_z, t_c)$$

$$\begin{aligned} t_c &= \text{Delay}(c_i \rightarrow c_{i+1}) \\ t_z &= \text{Delay}(c_i \rightarrow z_i) \end{aligned}$$

HIGH-LEVEL SPECIFICATION OF FULL-ADDER

$$x_i + y_i + c_i = 2c_{i+1} + z_i$$

INPUTS: $x_i, y_i, c_i \in \{0,1\}$
OUTPUTS: $z_i, c_{i+1} \in \{0,1\}$

FUNCTION: $z_i = (x_i + y_i + c_i) \bmod 2$

$$c_{i+1} = \begin{cases} 1 & \textbf{if} \quad (x_i + y_i + c_i) \geq 2 \\ 0 & \textbf{otherwise} \end{cases}$$

FULL-ADDER IMPLEMENTATION

x_i	y_i	c_i	c_{i+1}	z_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

$$\begin{aligned} z_i &= x_i y_i' c_i' + x_i' y_i c_i' + x_i' y_i' c_i + x_i y_i c_i \\ c_{i+1} &= x_i y_i + x_i c_i + y_i c_i \end{aligned}$$

FULL ADDER TWO-LEVEL IMPLEMENTATION

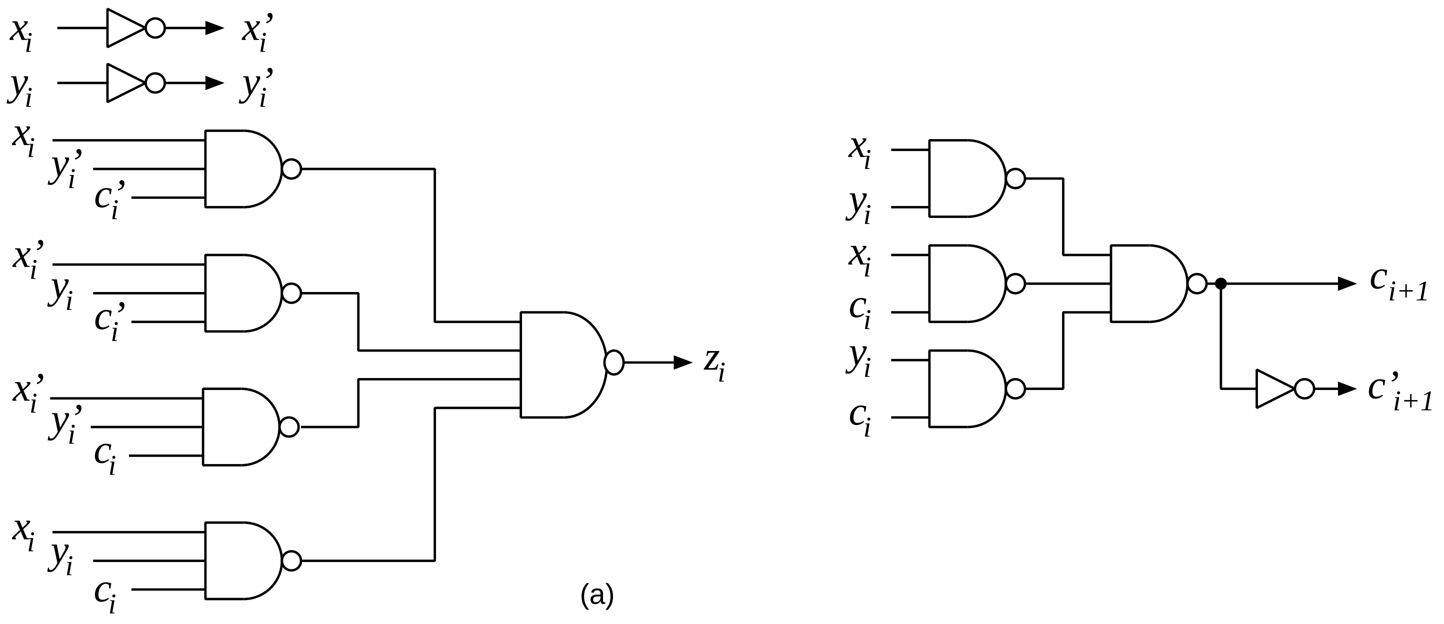


Figure 10.3: IMPLEMENTATIONS OF FULL-ADDER MODULE: a) TWO-LEVEL.

ALTERNATIVE IMPLEMENTATION

- ADDITION mod 2 $\rightarrow$ SUM IS 1 WHEN NUMBER OF 1'S IN INPUTS (including the carry-in) IS ODD:

$$z_i = x_i \oplus y_i \oplus c_i$$

- CARRY-OUT IS 1 WHEN $(x_i + y_i = 2)$ or $(x_i + y_i = 1$ and $c_i = 1)$:

$$c_{i+1} = x_i y_i + (x_i \oplus y_i) c_i$$

- INTERMEDIATE VARIABLES

$$\begin{aligned} \text{PROPAGATE} \quad p_i &= x_i \oplus y_i \\ \text{GENERATE} \quad g_i &= x_i \cdot y_i \end{aligned}$$

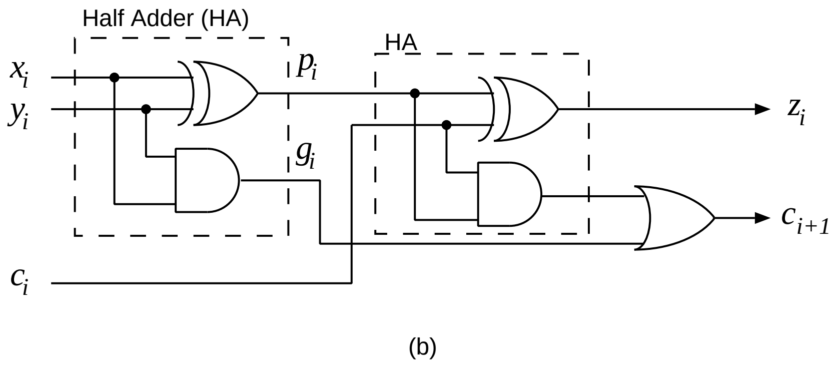
- HALF-ADDER

x_i	y_i	g_i	p_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

IMPLEMENTATION WITH HALF-ADDERS

- FA EXPRESSIONS IN TERMS OF $p_i's$, $g_i's$ and $c_i's$

$$\begin{aligned} z_i &= p_i \oplus c_i \\ c_{i+1} &= g_i + p_i \cdot c_i \end{aligned}$$



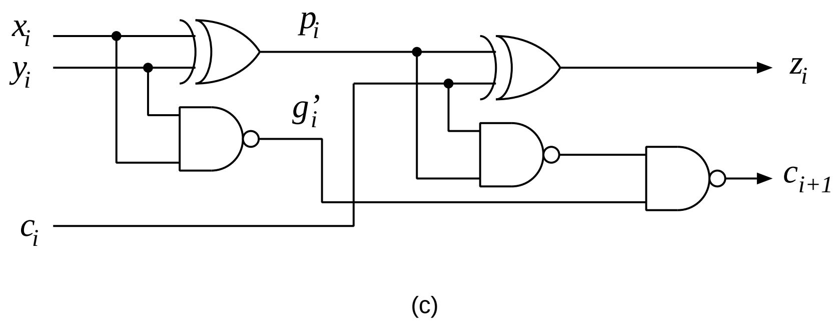


Figure 10.3: IMPLEMENTATIONS OF FULL-ADDER MODULE: b) MULTILEVEL GATE NETWORK WITH XORs, ANDs and OR; c) WITH XORs and NANDs.

WORST-CASE DELAY

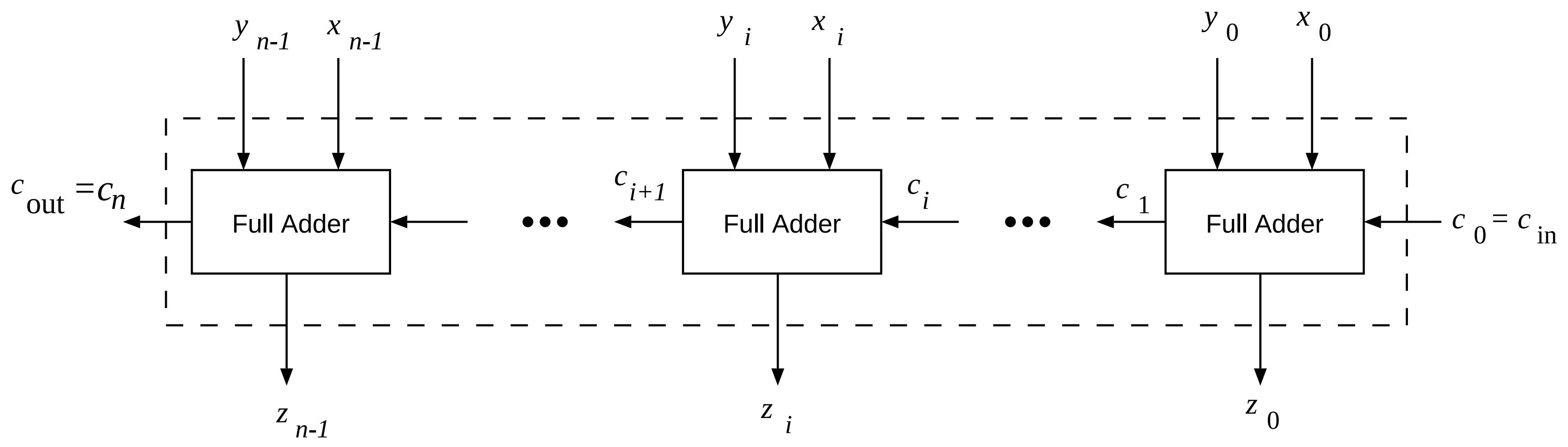


Figure 10.2: CARRY-RIPPLE ADDER MODULE.

- WORST-CASE DELAY $t_p = t_{\mathrm{XOR}} + 2(n-1)t_{\mathrm{NAND}} + \max(2t_{\mathrm{NAND}}, t_{\mathrm{XOR}})$

CHARACTERISTICS OF FULL-ADDER IN CMOS FAMILY

Input	[standard loads]
c_i	1.3
x_i	1.1
y_i	1.3

Size: 7 [equivalent gates]

From	To	Propagation delays	
		t_{pLH} [ns]	t_{pHL} [ns]
c_i	z_i	$0.43 + 0.03L$	$0.49 + 0.02L$
x_i	z_i	$0.68 + 0.04L$	$0.74 + 0.02L$
y_i	z_i	$0.68 + 0.04L$	$0.74 + 0.02L$
c_i	c_{i+1}	$0.36 + 0.04L$	$0.40 + 0.02L$
x_i	c_{i+1}	$0.73 + 0.04L$	$0.71 + 0.02L$
y_i	c_{i+1}	$0.37 + 0.04L$	$0.64 + 0.02L$

L: load on the gate output

CARRY-LOOKAHEAD ADDER IMPLEMENTATION

- FASTER IMPLEMENTATION
- ADDITION AS A TWO-STEP PROCESS:
  1. DETERMINE THE VALUES OF ALL THE CARRIES
  2. SIMULTANEOUSLY COMPUTE ALL THE RESULT BITS

CARRY-LOOKAHEAD ADDER MODULE

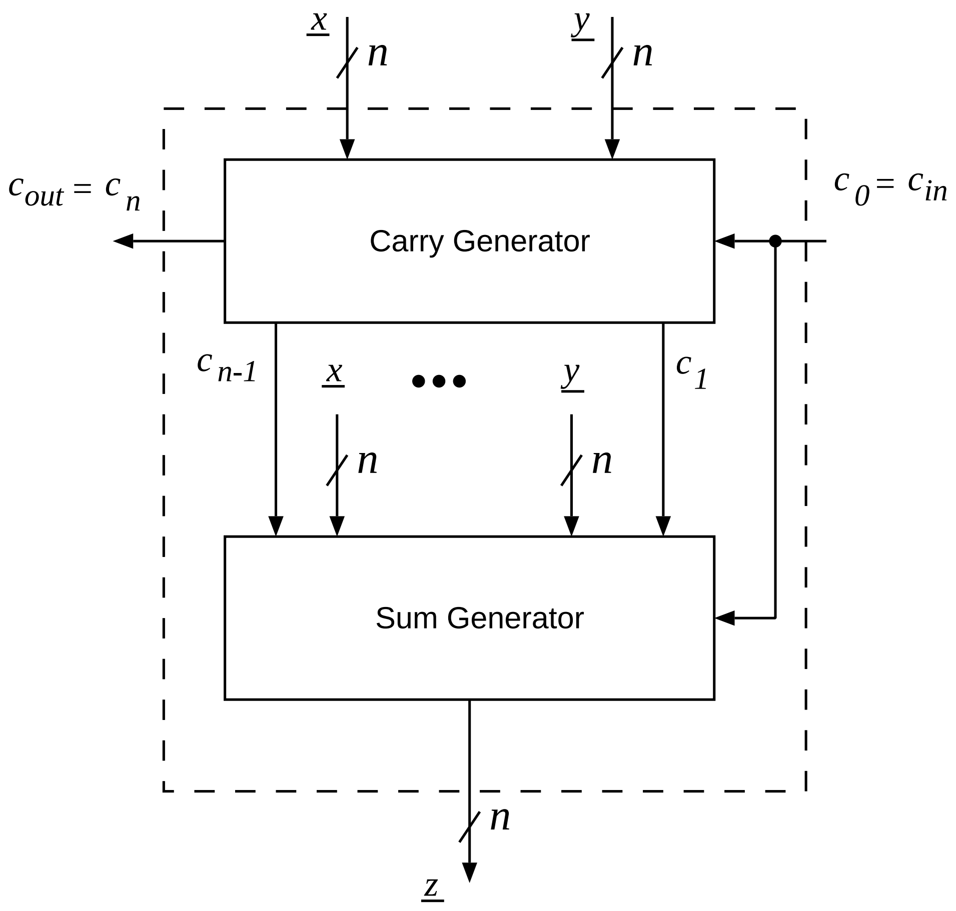


Figure 10.4: CARRY-LOOKAHEAD ADDER MODULE.

SWITCHING EXPRESSIONS

- INTERMEDIATE CARRIES:

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

BY SUBSTITUTION,

$$\begin{aligned}
c_1 &= g_0 + p_0 c_0 \\
c_2 &= g_1 + p_1 c_1 \\
&= g_1 + p_1 g_0 + p_1 p_0 c_0 \\
c_3 &= g_2 + p_2 c_2 \\
&= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \\
c_4 &= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0
\end{aligned}$$

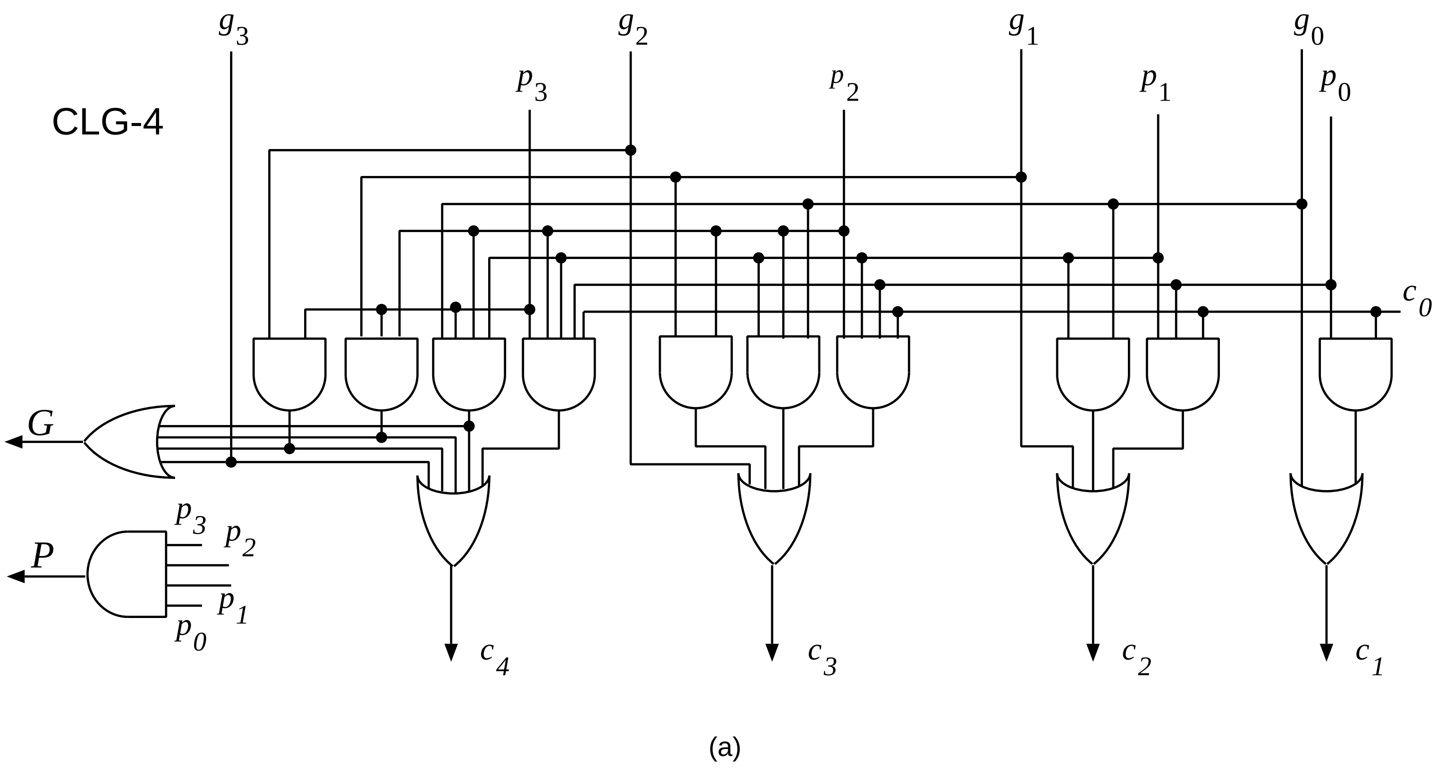


Figure 10.5: CARRY-LOOKAHEAD ADDER: a) 4-BIT CARRY-LOOKAHEAD GENERATOR WITH P and G OUTPUTS (CLG-4).

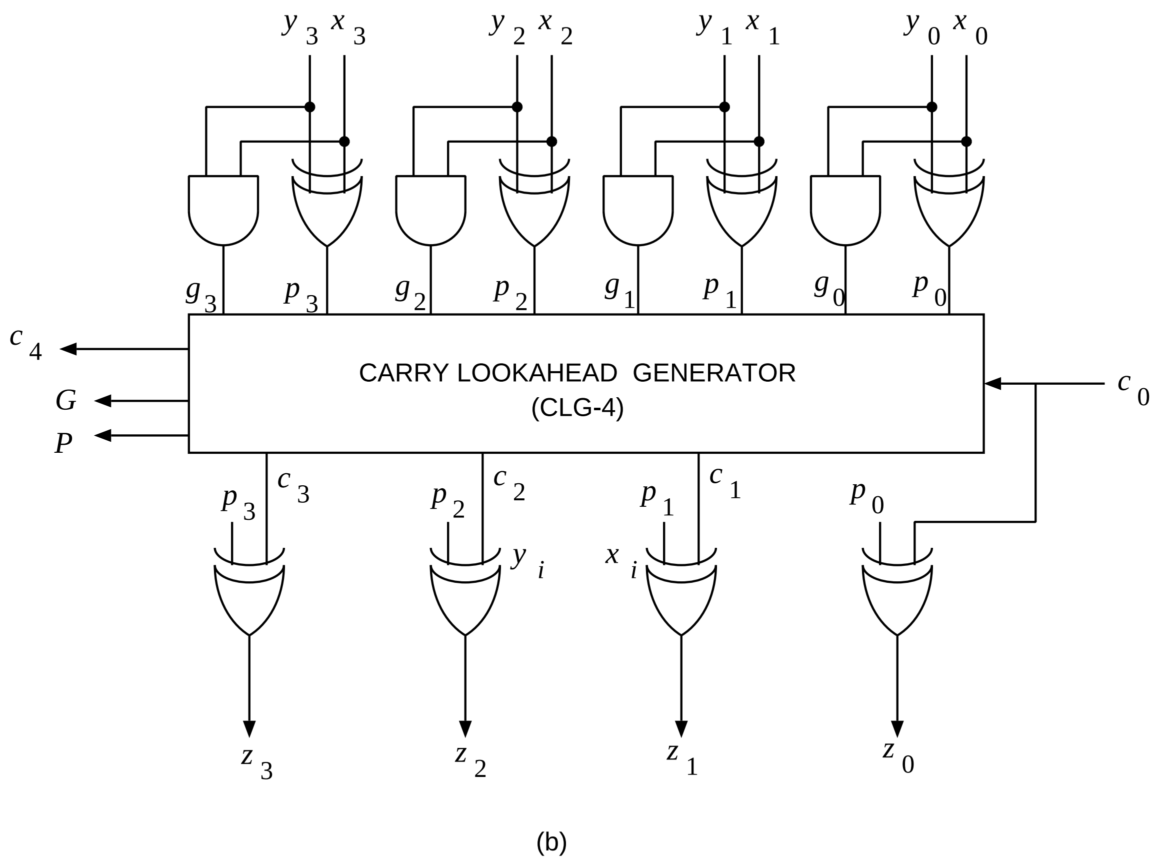


Figure 10.5: CARRY-LOOKAHEAD ADDER: b) 4-BIT MODULE (CLA-4); (CLG-4).

$$
\begin{aligned}
t_p(x_0 \rightarrow c_4) &= t_{pg} + t_{CLG-4} \\
t_p(c_0 \rightarrow c_4) &= t_{CLG-4} \\
t_p(x_0 \rightarrow P, G) &= t_{pg} + t_{CLG-4} \\
t_p(x_0 \rightarrow z_3) &= t_{pg} + t_{CLG-4} + t_{XOR}
\end{aligned}
$$

MODULE PROPAGATE AND GENERATE SIGNALS

$P = 1$: c_{in} PROPAGATED BY THE MODULE

$G = 1$: $c_{\text{out}} = 1$ GENERATED BY THE MODULE, IRRESPECTIVE OF c_{in}

$$P = \begin{cases} 1 & \textbf{if} \quad x + y = 2^4 - 1 \\ 0 & \textbf{otherwise} \end{cases}$$

$$G = \begin{cases} 1 & \textbf{if} \quad x + y \geq 2^4 \\ 0 & \textbf{otherwise} \end{cases}$$

$$c_{out} = G + P \cdot c_{in}$$

$$P = p_3 p_2 p_1 p_0$$

$$G = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$$

NETWORKS OF ADDER MODULES

ITERATIVE (CARRY-RIPPLE) ADDER NETWORK

$$\begin{aligned}
\underline{x} &= (\underline{x}^{(3)}, \underline{x}^{(2)}, \underline{x}^{(1)}, \underline{x}^{(0)}) \\
\underline{x}^{(3)} &= (x_{15}, x_{14}, x_{13}, x_{12}) \\
\underline{x}^{(2)} &= (x_{11}, x_{10}, x_9, x_8) \\
\underline{x}^{(1)} &= (x_7, x_6, x_5, x_4) \\
\underline{x}^{(0)} &= (x_3, x_2, x_1, x_0)
\end{aligned}$$

where

$$x = 2^{12}x^{(3)} + 2^8x^{(2)} + 2^4x^{(1)} + x^{(0)}$$

16-BIT CARRY-RIPPLE ADDER

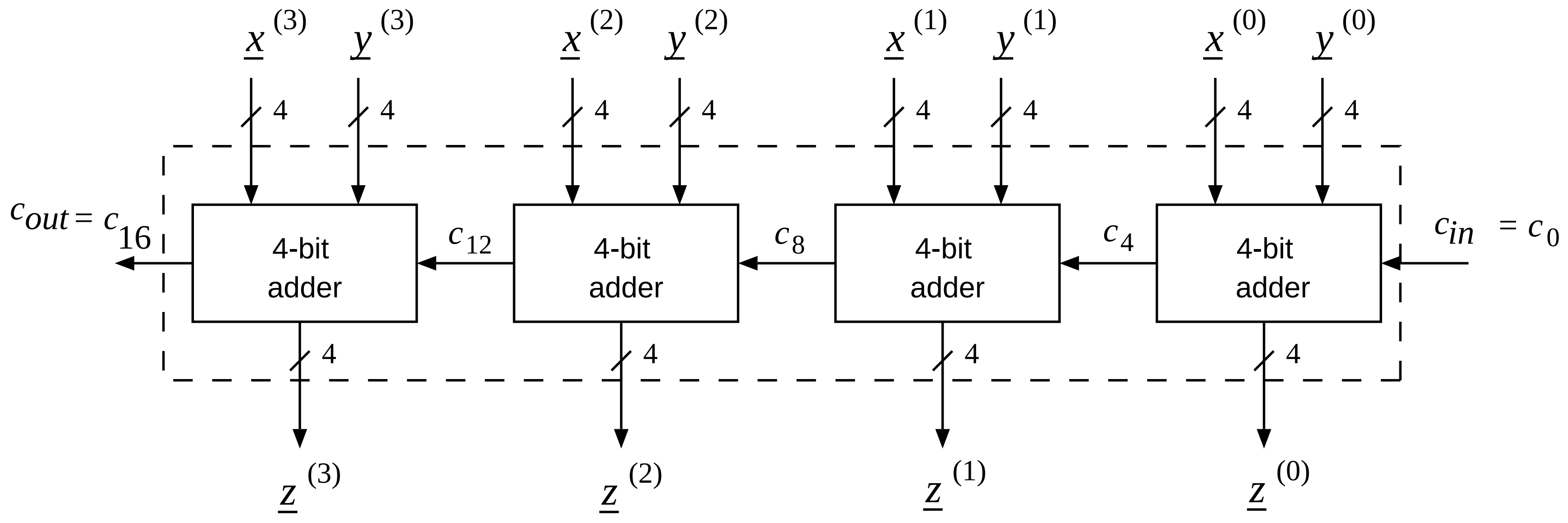


Figure 10.6: 16-BIT CARRY-RIPPLE ADDER NETWORK USING 4-BIT ADDER MODULES.

CARRY-LOOKAHEAD ADDER NETWORK

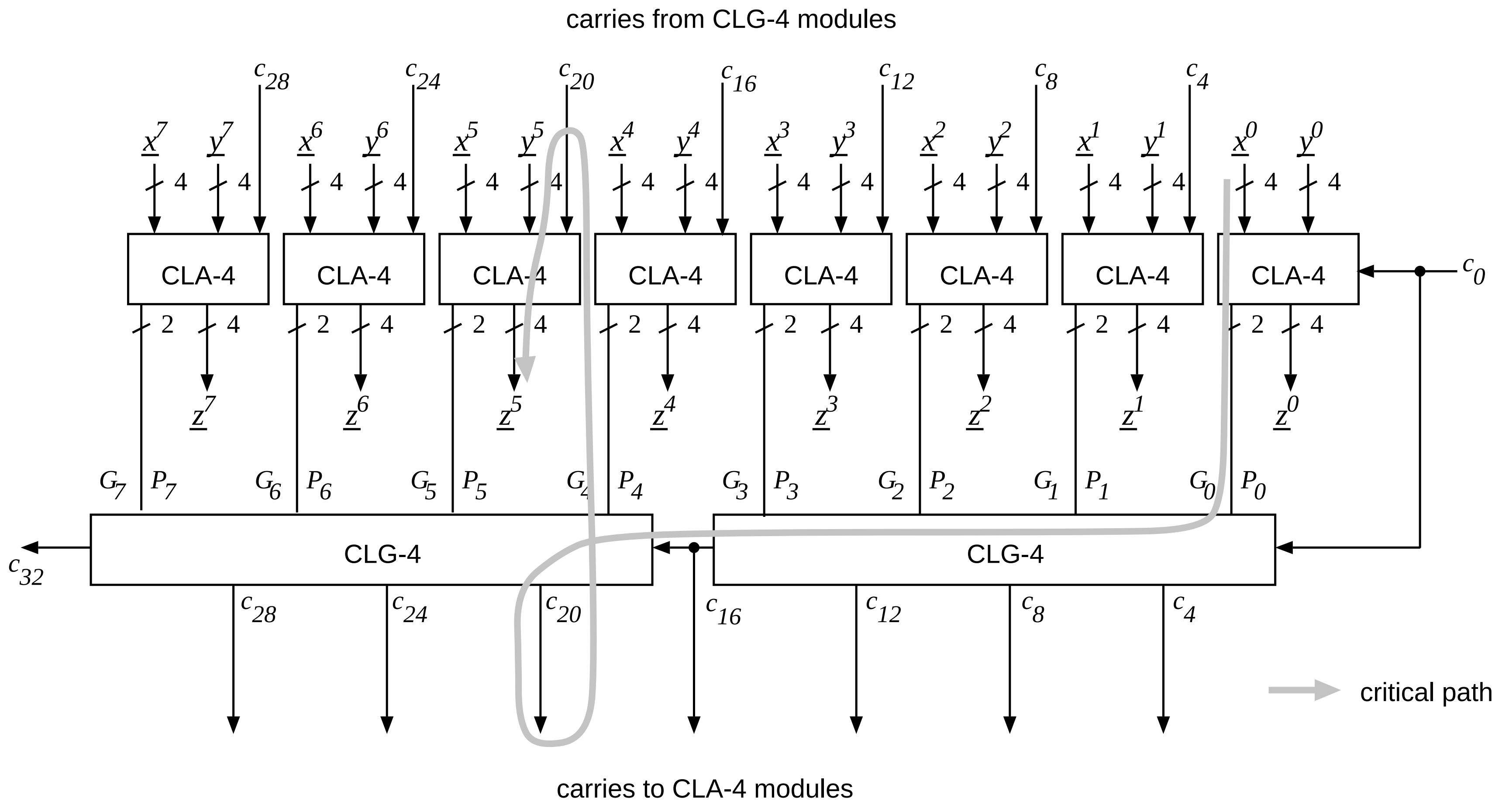

carries from CLG-4 modules
c28 c24 c20 c16 c12 c8 c4
x7 y7 x6 y6 x5 y5 x4 y4 x3 y3 x2 y2 x1 y1 x0 y0
CLA-4
c0
z7 z6 z5 z4 z3 z2 z1 z0
G7 P7 G6 P6 G5 P5 G4 P4 G3 P3 G2 P2 G1 P1 G0 P0
CLG-4
c32
c28 c24 c20 c16 c12 c8 c4
critical path
carries to CLA-4 modules

Figure 10.7: 32-BIT CARRY-LOOKAHEAD ADDER USING CLA-4 AND CLG-4 MODULES.

- PROPAGATION DELAY:

$$t_p(net) = t_{PG} + 2t_{\text{CLG}-4} + t_{\text{ADD}}$$

CLA ADDER (cont.)

$$\begin{aligned}
c_4 &= G_0 + P_0c_0 \\
c_8 &= G_1 + P_1G_0 + P_1P_0c_0 \\
c_{12} &= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0c_0 \\
c_{16} &= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0c_0
\end{aligned}$$

$$\begin{aligned}
P_0 &= p_3 \cdot p_2 \cdot p_1 \cdot p_0 \\
G_0 &= g_3 + g_2p_3 + g_1p_3p_2 + g_0p_3p_2p_1
\end{aligned}$$

REPRESENTATION OF SIGNED INTEGERS AND BASIC OPERATIONS

- TWO COMMON REPRESENTATIONS:
  - SIGN-AND-MAGNITUDE (SM)
  - TRUE-AND-COMPLEMENT (TC)

SIGN-AND-MAGNITUDE (SM) SYSTEM

- x REPRESENTED BY PAIR (x_s, x_m)

  sign:

$$x_s = \begin{cases} 0 \ \textbf{if} \ \ x \geq 0 \\ 1 \ \textbf{if} \ \ x \leq 0 \end{cases}$$

  magnitude:

$$x_m$$

- RANGE OF SIGNED INTEGERS

total number of bits:	n
sign:	1
magnitude:	$n-1$

$$-(2^{n-1}-1) \leq x \leq 2^{n-1}-1$$

- TWO REPRESENTATIONS OF ZERO:

  $x_s = 0, x_m = 0$ (positive zero)

  $x_s = 1, x_m = 0$ (negative zero)

TWO'S-COMPLEMENT SYSTEM

- NO SEPARATION BETWEEN THE REPRESENTATION OF SIGN AND REPRESENTATION OF MAGNITUDE
- SIGNED INTEGER x REPRESENTED BY $POSITIVE$ INTEGER x_R
- MAP 2: BINARY REPRESENTATION OF x_R

$$x_R = \sum_{i=0}^{n-1} x_i 2^i \quad , \quad 0 \leq x_R \leq 2^n - 1$$

- MAP 1: TWO'S COMPLEMENT

$$x_R = x \bmod 2^n$$

BY DEFINITION OF mod, FOR $|x| < 2^n$: equivalent to

$$x_R = \begin{cases} x & \textbf{if} \quad x \geq 0 \\ 2^n - |x| & \textbf{if} \quad x < 0 \end{cases}$$

FOR UNAMBIGUOUS SYMMETRICAL REPRESENTATION

$$|x|_{max} \leq 2^{n-1} - 1$$

x	-4	-3	-2	-1	0	1	2	3
x_R	4	5	6	7	0	1	2	3

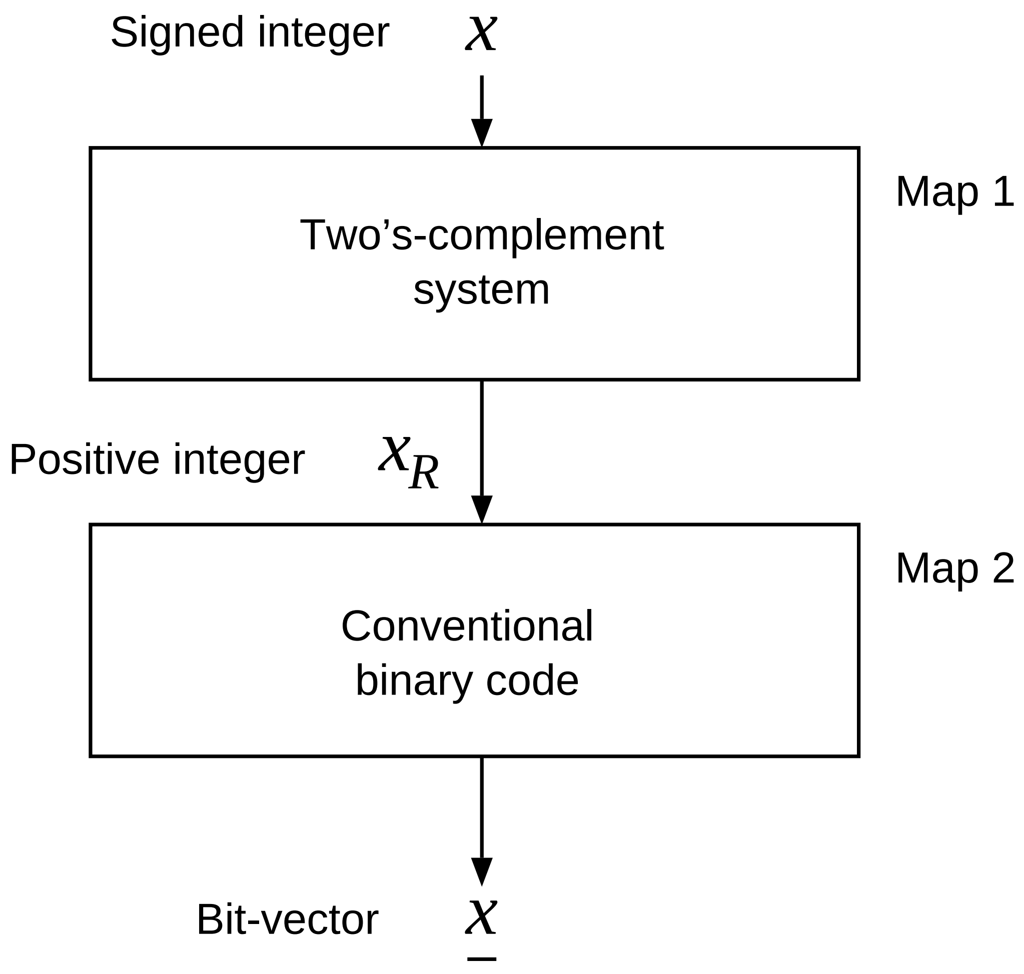


Figure 10.8: SIGNED INTEGER REPRESENTED BY POSITIVE INTEGER.

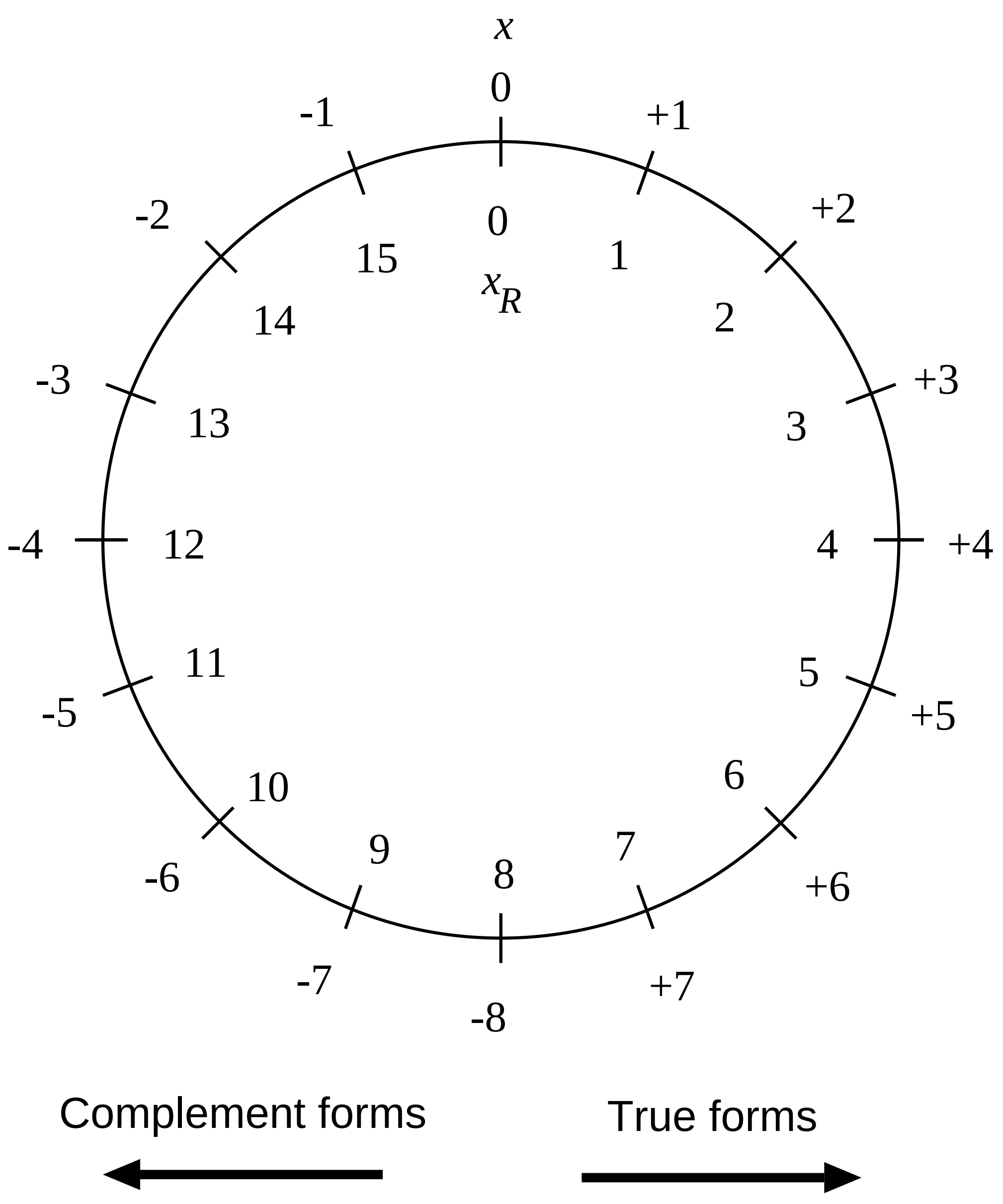


Figure 10.9: TWO'S COMPLEMENT REPRESENTATION FOR $n = 4$.

MAPPING IN TWO'S-COMPLEMENT SYSTEM

x	x_R	$\underline{x}$	
0	0	00...000	
1	1	00...001	
2	2	00...010	
-	-	-	True forms
-	-	-	(positive)
-	-	-	$x_R = x$
$2^{n-1} - 1$	$2^{n-1} - 1$	01...111	
-2^{n-1}	2^{n-1}	10...000	
$-(2^{n-1} - 1)$	$2^{n-1} + 1$	10...001	
-	-	-	
-	-	-	Complement forms
-	-	-	(negative)
-2	$2^n - 2$	11...110	$x_R = 2^n - \lvert x \rvert$
-1	$2^n - 1$	11...111	

EXAMPLE 10.2: MAPPINGS FOR $-4 \leq x \leq 3$

x	x_R	$\underline{x}$
3	3	011
2	2	010
1	1	001
0	0	000
-1	7	111
-2	6	110
-3	5	101
-4	4	100

CONVERSE MAPPING

$$x = \begin{cases} x_R & \textbf{if} \;\; x_R \leq 2^{n-1} - 1 \;\; (x \geq 0) \\ x_R - 2^n & \textbf{if} \;\; x_R \geq 2^{n-1} \;\; (x < 0) \end{cases}$$

IN TERMS OF BIT VECTOR $(x_{n-1}, x_{n-2}, ..., x_1, x_0)$

i) For $x_R < 2^{n-1}$, bit x_{n-1} is 0 and $x \geq 0$.

$$x = x_R = 0 \times 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i$$

ii) For $x_R \geq 2^{n-1}$ bit x_{n-1} is 1 and $x < 0$.

$$x = x_R - 2^n = (1 \times 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i) - 2^n = -1 \times 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i$$

COMBINING BOTH CASES

$$x = -x_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i$$

$$x = -x_{n-1}2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i$$

8-BIT EXAMPLES:

$\underline{x}$	x	
01000101	0 + 69	= 69
11000101	$-128 + 69$	$= -58$

SIGN DETECTION:

$$\begin{array}{lll} x \geq 0 & \textbf{if} & x_{n-1} = 0 \\ x < 0 & \textbf{if} & x_{n-1} = 1 \end{array}$$

ONES'-COMPLEMENT SYSTEM

$$x_R = x \bmod C$$

ONES'-COMPLEMENT SYSTEM: $C = 2^n - 1$

- the ones'-complement system symmetrical, with the range $-(2^n - 1) \leq x \leq 2^n - 1$;
- two representations for zero, namely $x_R = 0$ and $x_R = 2^n - 1$;
- the sign also detected by the most-significant bit

$$x \geq 0 \quad \textbf{if} \quad (x_{n-1} = 0) \textbf{ or } (x_R = 2^n - 1)$$

MAPPING IN ONES'-COMPLEMENT SYSTEM

x	x_R	$\underline{x}$	
0	0	00...000	
1	1	00...001	
2	2	00...010	
-	-	-	True forms
-	-	-	(positive)
-	-	-	$x_R = x$
$2^{n-1} - 1$	$2^{n-1} - 1$	01...111	
$-(2^{n-1} - 1)$	2^{n-1}	10...000	
-	-	-	
-	-	-	Complement forms
-2	$2^n - 3$	11...101	(negative)
-1	$2^n - 2$	11...110	$x_R = 2^n - 1 - \|x\|$
0	$2^n - 1$	11...111	

ADDITION IN TWO'S COMPLEMENT SYSTEM

- TO GET

$$z = x + y$$

COMPUTE

$$z_R = (x_R + y_R) \; mod \; 2^n$$

CORRECT IF $-2^{n-1} \leq (x + y) \leq 2^{n-1} - 1$

- PROOF: CONSIDER

$$(x_R + y_R) \; mod \; 2^n$$

AND SHOW THAT IT CORRESPONDS TO z_R

BY DEFINITION OF THE REPRESENTATION,

$$x_R = x \; mod \; 2^n \quad and \quad y_R = y \; mod \; 2^n$$

THEREFORE,

$$\begin{aligned} (x_R + y_R) \; mod \; 2^n &= (x \; mod \; 2^n + y \; mod \; 2^n) \; mod \; 2^n \\ &= (x + y) \; mod \; 2^n = z \; mod \; 2^n \end{aligned}$$

BY DEFINITION OF REPRESENTATION

$$z \; mod \; 2^n = z_R$$

2's COMPLEMENT ADDITION: A SUMMARY

1. ADD x_R AND y_R (use adder for positive operands)
2. PERFORM THE mod OPERATION

- DOES NOT DEPEND ON THE RELATIVE MAGNITUDES OF THE OPERANDS AND ON THEIR SIGNS (simpler than in S+M)

EXAMPLES OF ADDITION FOR C=64 and $-32 \leq x, y, z \leq 31$

Signed operands		Representation		Two's-complement addition	Signed result
x	y	x_R	y_R	$(x_R + y_R) \bmod 64 = z_R$	z
13	9	13	9	$22 \bmod 64 = 22$	22
13	-9	13	55	$68 \bmod 64 = 4$	4
-13	9	51	9	$60 \bmod 64 = 60$	-4
-13	-9	51	55	$106 \bmod 64 = 42$	-22

THE *mod* OPERATION

- Let $w_R = x_R + y_R$. Then

$$x_R, y_R < 2^n \quad \Rightarrow \quad w_R < 2 \times 2^n$$

$$z_R = w_R \; mod \; 2^n = \begin{cases} w_R & \textbf{if} \;\; w_R < 2^n \\ w_R - 2^n & \textbf{if} \;\; 2^n \leq w_R < 2 \times 2^n \end{cases}$$

- Since $w_R < 2 \times 2^n$

$$\underline{w} = (w_n, w_{n-1}, ..., w_0)$$

$$w_R = \begin{cases} < 2^n & \textbf{if} \;\; w_n = 0 \\ \geq 2^n & \textbf{if} \;\; w_n = 1 \end{cases}$$

Case 1. $w_R < 2^n$. Then $w_R \; mod \; 2^n = w_R \Leftrightarrow (w_{n-1}, ..., w_0)$.

Case 2. $w_R \geq 2^n$

$$\begin{aligned} w_R \; mod \; 2^n = w_R - 2^n &\Leftrightarrow (1, w_{n-1}, \ldots, w_0) - (1, 0, \ldots, 0) \\ &= (w_{n-1}, \ldots, w_0) \end{aligned}$$

- CONCLUSION: $w_R \; mod \; 2^n = (w_{n-1}, ..., w_0)$

- mod OPERATION PERFORMED BY DISCARDING MOST-SIGNIFICANT BIT OF $\underline{w}$
- 2'S COMPLEMENT ADDITION:

  RESULT CORRESPONDS TO OUTPUT OF ADDER, DISCARDING THE CARRY-OUT

$$\underline{z} = ADD(\underline{x}, \underline{y}, 0)$$

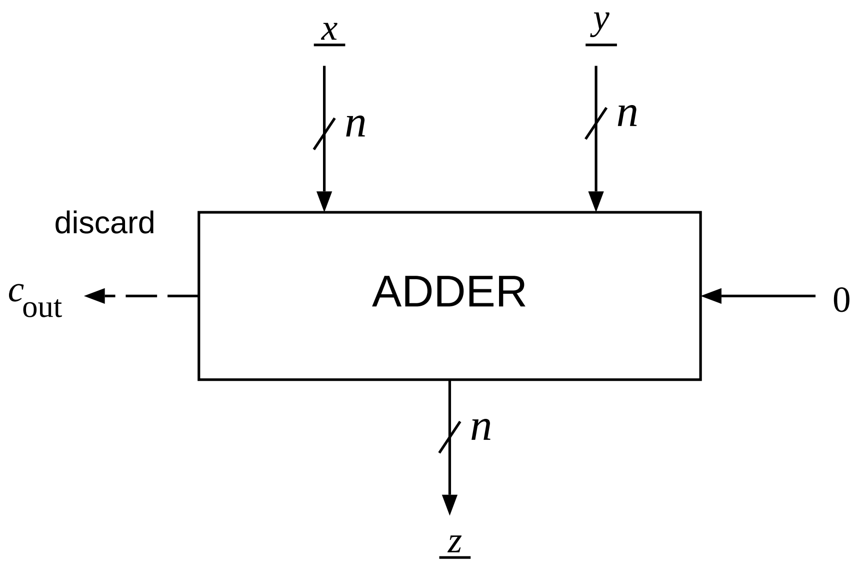


Figure 10.10: TWO'S-COMPLEMENT ADDER MODULE.

EXAMPLES OF 2'S COMPLEMENT ADDITION

	Bit-level computation	Positive representation	Signed values
$n = 4$	$\underline{x} = 1011$	$x_R = 11$	$x = -5$
	$\underline{y} = 0101$	$y_R = 5$	$y = 5$
	$\underline{w} = 10000$	$w_R = 16$	
	$\underline{z} = 0000$	$z_R = 0$	$z = 0$
$n = 8$	$\underline{x} = 11011010$	$x_R = 218$	$x = -38$
	$\underline{y} = 11110001$	$y_R = 241$	$y = -15$
	$\underline{w} = 111001011$	$w_R = 459$	
	$\underline{z} = 11001011$	$z_R = 203$	$z = -53$

CHANGE OF SIGN IN TWO'S COMPLEMENT SYSTEM

- $z = -x$

$$z_R = (2^n - x_R)\ mod\ 2^n$$

$x = 0$: since $z = -x = 0$ we have $z_R = x_R = 0$. Moreover,

$$z_R = (2^n - 0)\ mod\ 2^n = 0$$

$x > 0$: since $z = -x$ is negative,

$$z_R = 2^n - |z| = 2^n - |x|$$

Moreover, x is positive so that

$$x_R = x$$

Substitute: $z_R = 2^n - x_R$.

$x < 0$: since $z = -x$ is positive,

$$z_R = z = -x$$

Moreover, x is negative so that

$$x_R = 2^n - |x| = 2^n + x$$

Substitute: $z_R = 2^n - x_R$.

CHANGE OF SIGN (cont.)

- DIRECT SUBTRACTION $2^n - x_R$ COMPLEX

EXAMPLE:

$$\begin{array}{rr} 2^8 & 100000000 \\ x_R & 01011110 \\ \hline & 10100010 \end{array}$$

- INSTEAD, USE $2^n = (2^n - 1) + 1$

$$z_R = (2^n - 1 - x_R) + 1$$

- THE COMPLEMENT WITH RESPECT TO $2^n - 1$: COMPLEMENT EACH BIT OF $\underline{x}$

$$\begin{array}{lr} x_R = 17 & 010001 \\ 63 - x_R & 111111 - 010001 \\ & 101110 \end{array}$$

CHANGE-OF-SIGN OPERATION

TWO-STEP OPERATION:

1. COMPLEMENT EACH BIT OF $\underline{x}$ denoted $\underline{x'}$.
2. ADD 1 (set carry-in $c_0 = 1$)

- DESCRIPTION:

$$\underline{z} = ADD(\underline{x'}, \underline{0}, 1)$$

EXAMPLE FOR $n = 4$, $x = -3$:

$\underline{x}$	1101	$x = -3$
$\underline{x'}$	0010	
0	0000	
c_0	1	
$\underline{z}$	0011	$z = 3$

SUBTRACTION IN TWO'S COMPLEMENT SYSTEM

- $z = x - y = x + (-y)$

$$z_R = (x_R + (2^n - 1 - y_R) + 1) \bmod 2^n$$

- THE CORRESPONDING DESCRIPTION

$$\underline{z} = ADD_R(\underline{x}, \underline{y}', 1)$$

EXAMPLE:

$\underline{x}$			01100000
$\underline{y}$	00110001	$\underline{y}'$	11001110
			1
$\underline{z}$			00101111

SUMMARY OF 2'S COMPLEMENT OPERATIONS

OPERATION	2's COMPLEMENT SYSTEM
$z = x + y$	$\underline{z} = ADD(\underline{x}, \underline{y}, 0)$
$z = -x$	$\underline{z} = ADD(\underline{x}', 0, 1)$
$z = x - y$	$\underline{z} = ADD(\underline{x}, \underline{y}', 1)$

OVERFLOW DETECTION IN ADDITION

- OVERFLOW – result exceeds most positive or negative representable integer

$$-2^{n-1} \leq z \leq 2^{n-1} - 1$$

- BOTH OPERANDS SAME SIGN, RESULT OPPOSITE SIGN

$$v = x'_{n-1}y'_{n-1}z_{n-1} + x_{n-1}y_{n-1}z'_{n-1}$$

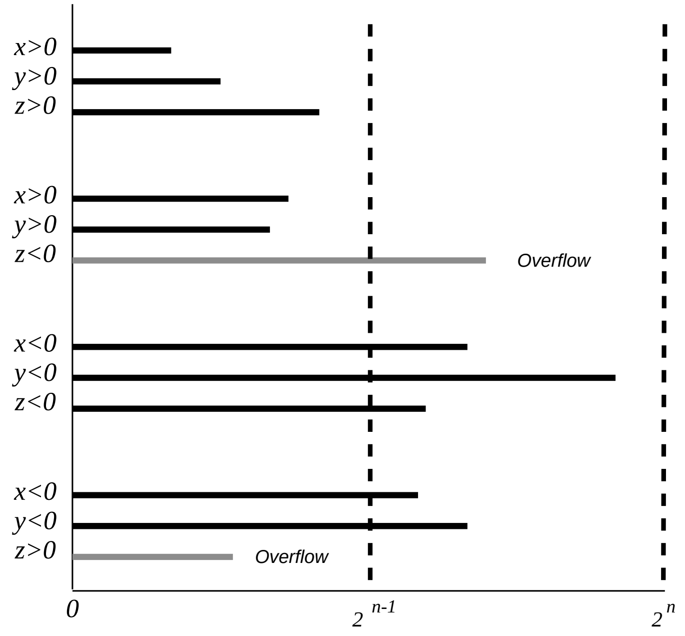


Figure 10.11: OVERFLOW IN TWO'S-COMPLEMENT SYSTEM.

TWO'S COMPLEMENT ARITHMETIC UNIT

INPUTS: $\underline{x} = (x_{n-1}, \ldots, x_0), \quad x_j \in \{0,1\}$
$\underline{y} = (y_{n-1}, \ldots, y_0), \quad y_j \in \{0,1\}$
$c_{\text{in}} \in \{0,1\}$
$F = (f_2, f_1, f_0)$

OUTPUTS: $\underline{z} = (z_{n-1}, \ldots, z_0), \quad z_j \in \{0,1\}$
$c_{\text{out}}, sgn, zero, ovf \in \{0,1\}$

FUNCTIONS:

F	Operation		
001	ADD	add	$z = x + y$
011	SUB	subtract	$z = x - y$
101	ADDC	add with carry	$z = x + y + c_{in}$
110	CS	change sign	$z = -x$
010	INC	increment	$z = x + 1$

$sgn = 1$ **if** $z < 0$, 0 **otherwise** (the sign)
$zero = 1$ **if** $z = 0$, 0 **otherwise**
$ovf = 1$ **if** z overflows, 0 **otherwise**

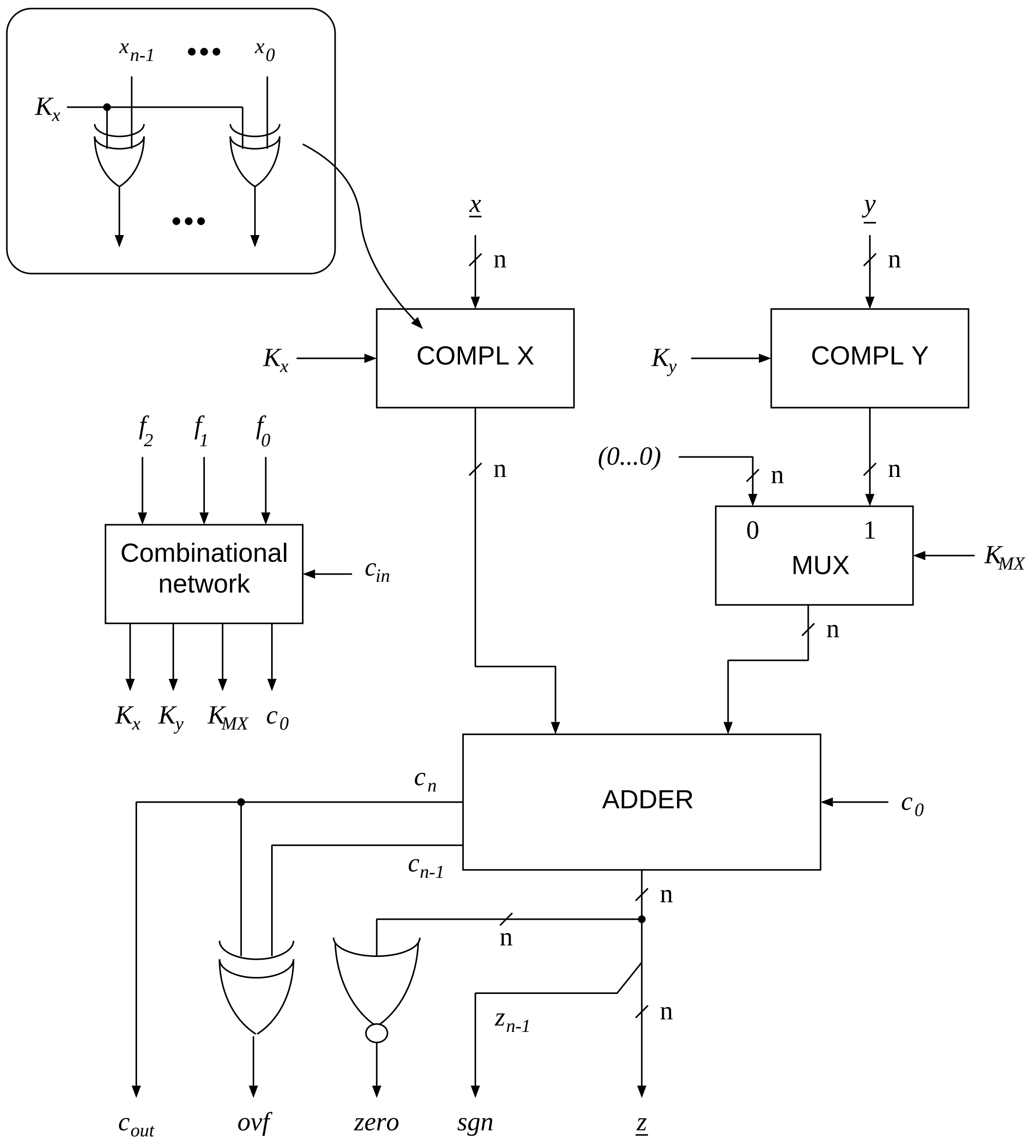


Figure 10.12: IMPLEMENTATION OF TWO'S-COMPLEMENT ARITHMETIC UNIT.

CONTROL OF TWO'S-COMPLEMENT ARITHMETIC OPERATIONS

- OPERATION IDENTIFIED BY BIT-VECTOR $F = (f_2, f_1, f_0)$
- COMPLEMENT OPERATION $\underline{a} = \textsc{compl}(\underline{b}, K)$:

$$a_i = \begin{cases} b_i & \textbf{if} \quad K = 0 \\ b_i' & \textbf{if} \quad K = 1 \end{cases}$$

Operation	Op-code		Control Signals		
	$f_2 f_1 f_0$	$\underline{z}$	K_x	K_y	K_{MX}
ADD	001	$\textsc{add}(\underline{x}, \underline{y}, 0)$	0	0	1
SUB	011	$\textsc{add}(\underline{x}, \underline{y'}, 1)$	0	1	1
ADDC	101	$\textsc{add}(\underline{x}, \underline{y}, c_{\text{in}})$	0	0	1
CS	110	$\textsc{add}(\underline{x'}, \underline{0}, 1)$	1	d.c.	0
INC	010	$\textsc{add}(\underline{x}, \underline{0}, 1)$	0	d.c.	0

- CONTROL SIGNALS:

$$\begin{aligned} K_x &= f_2 f_1 \\ K_y &= f_1 \\ K_{MX} &= f_0 \\ c_0 &= f_1 + f_2 f_0 c_{\text{in}} \end{aligned}$$

ALU MODULES AND NETWORKS

- *ARITHMETIC-LOGIC UNIT*

  module realizing set of arithmetic and logic functions
- Why build ALUs?
  1. Use in many different applications
  2. ALU modules used in processors: function selected by control unit

TYPICAL EXAMPLE OF ALU

Control (S)	Function
ZERO	$z = 0$
ADD	$z = (x + y + c_{\text{in}}) \bmod 16$
SUB	$z = (x + y' + c_{\text{in}}) \bmod 16$
EXSUB	$z = (x' + y + c_{\text{in}}) \bmod 16$
AND	$\underline{z} = \underline{x} \cdot \underline{y}$
OR	$\underline{z} = \underline{x} \mathbf{+} \underline{y}$
XOR	$\underline{z} = \underline{x} \oplus \underline{y}$
ONE	$\underline{z} = 1111$

a' denotes the integer represented by vector $\underline{a}'$

$\cdot$, $\mathbf{+}$, and $\oplus$ are applied to the corresponding bits

4-bit ALU

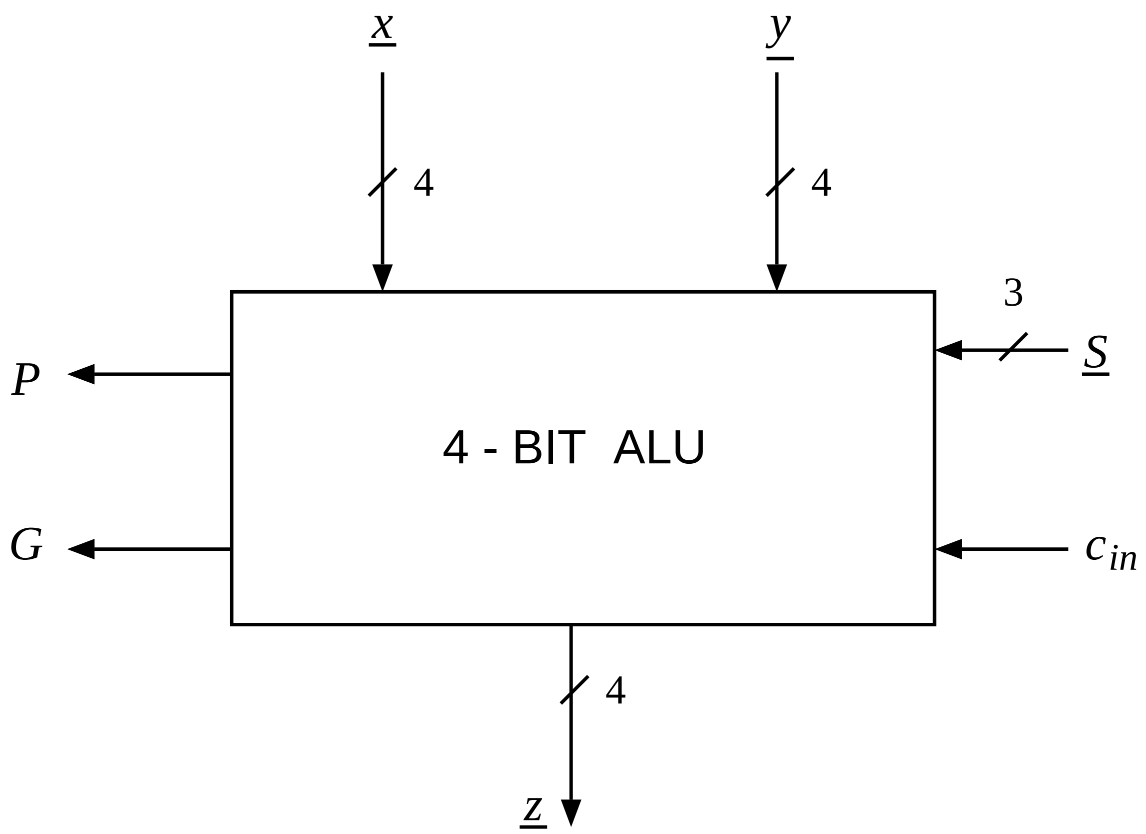


Figure 10.13: 4-bit ALU.

NETWORKS OF ALU MODULES

- MODULE HAS NO CARRY-OUT SIGNAL
  - cannot be used directly in an iterative (carry-ripple) network
  - carry-out signal implemented as

$$c_{\text{out}} = G + P \cdot c_{\text{in}}$$

  - *carry-skip network*

16-bit ALU

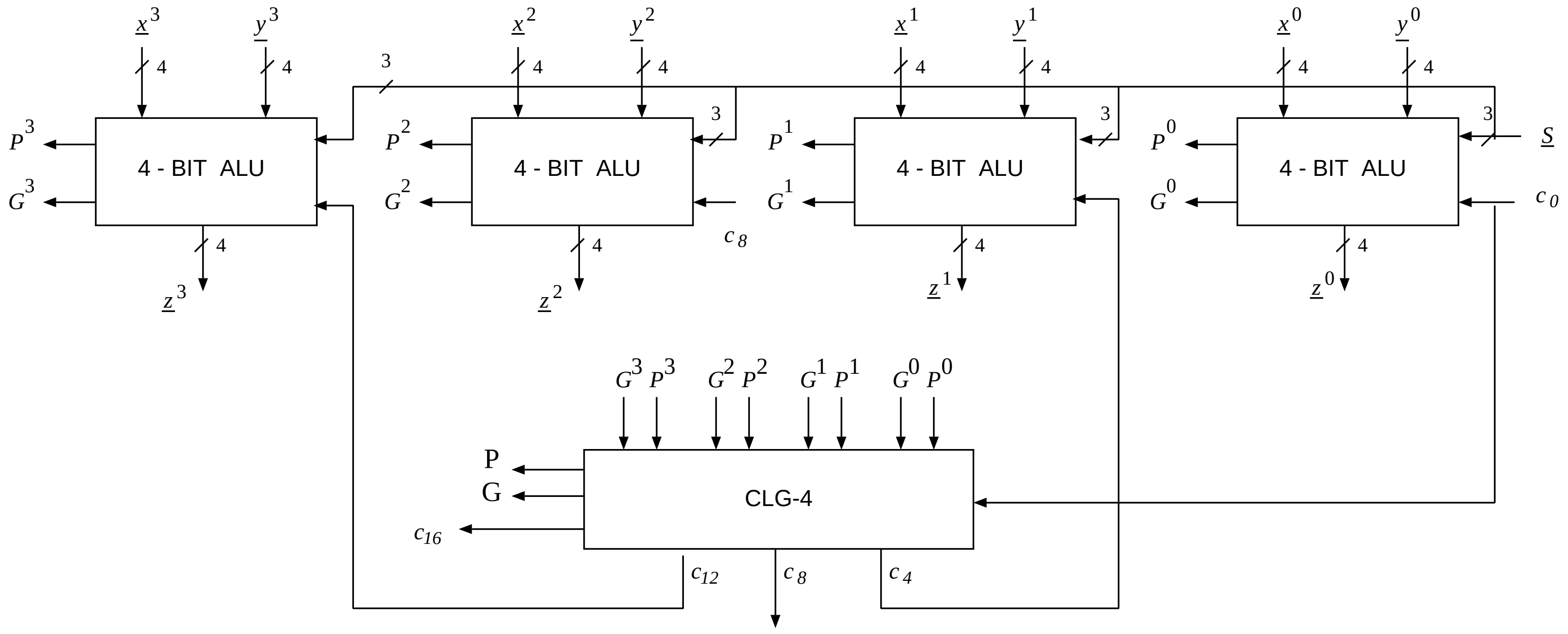


Figure 10.14: 16-bit ALU.

COMPARATOR MODULES

- HIGH-LEVEL DESCRIPTION OF AN n-BIT COMPARATOR:

INPUTS: $\underline{x} = (x_{n-1}, \ldots, x_0), \quad x_j \in \{0,1\}$

$\underline{y} = (y_{n-1}, \ldots, y_0), \quad y_j \in \{0,1\}$

$c_{\text{in}} \in \{\text{G,E,S}\}$

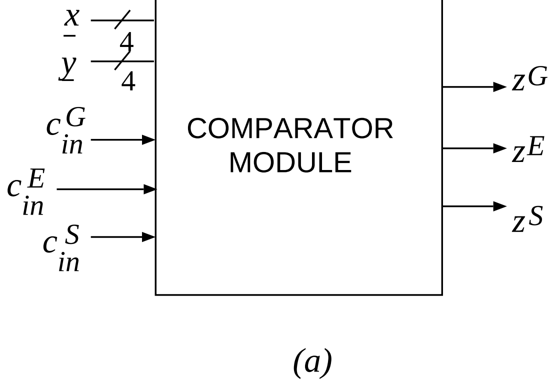
OUTPUT: $z \in \{\text{G,E,S}\}$

FUNCTION: $$z = \begin{cases} \text{G} & \textbf{if} \quad (x > y) \textbf{ or } (x = y \textbf{ and } c_{\text{in}} = \text{G}) \\ \text{E} & \textbf{if} \quad (x = y) \textbf{ and } (c_{\text{in}} = \text{E}) \\ \text{S} & \textbf{if} \quad (x < y) \textbf{ or } (x = y \textbf{ and } c_{\text{in}} = \text{S}) \end{cases}$$

x and y – the integers represented $\underline{x}$ and $\underline{y}$

- IMPLEMENTATION OF 4-bit COMPARATOR MODULE

$$\begin{aligned} \underline{c_{\text{in}}} &= (c_{\text{in}}^G, c_{\text{in}}^E, c_{\text{in}}^S) \quad , \quad c_{\text{in}}^G, c_{\text{in}}^E, c_{\text{in}}^S \in \{0,1\} \\ \underline{z} &= (z^G, z^E, z^S) \quad , \quad z^G, z^E, z^S \in \{0,1\} \end{aligned}$$



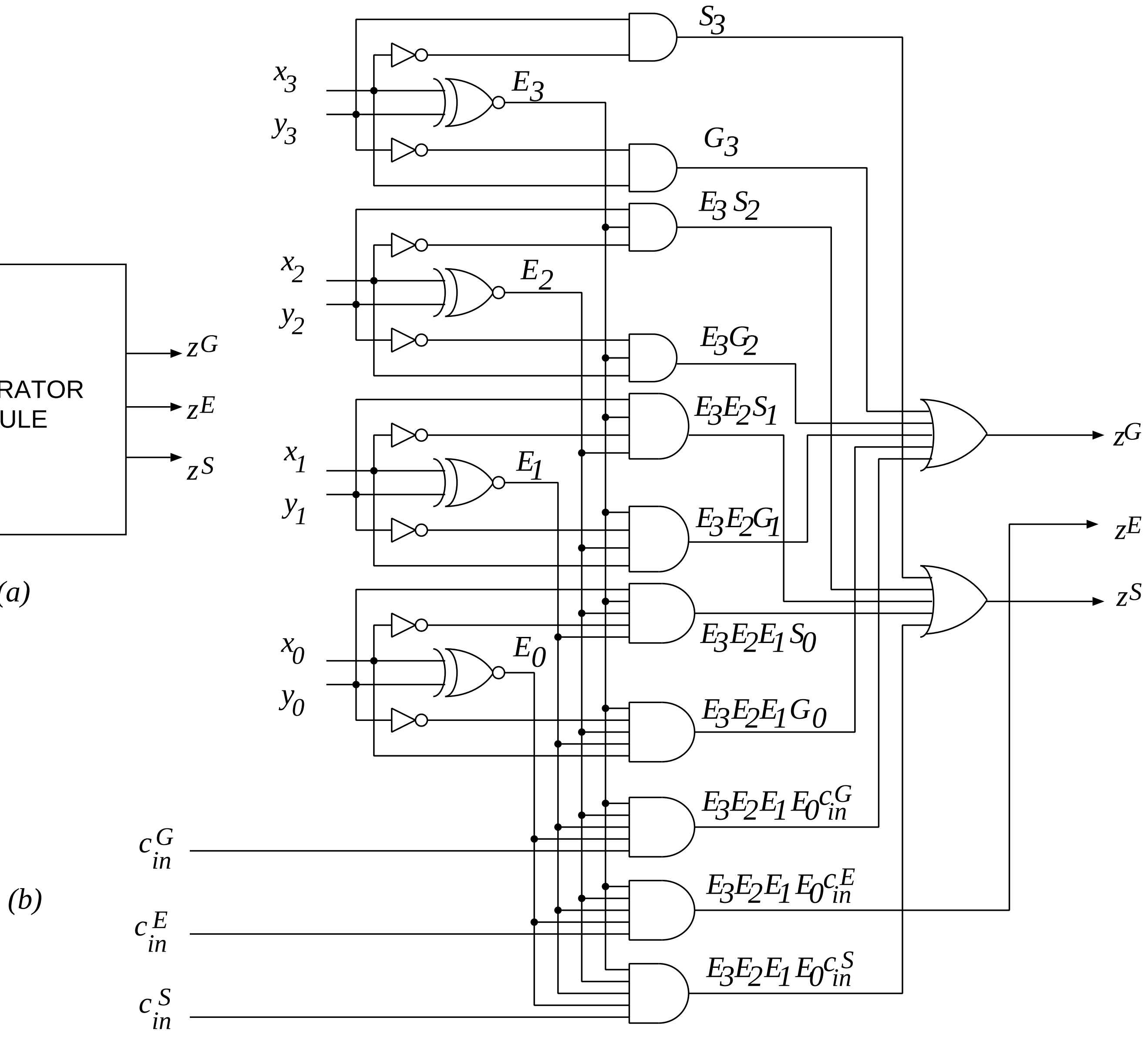


Figure 10.15: 4-BIT COMPARATOR MODULE: a) block diagram; b) gate-network implementation.

BINARY-LEVEL DESCRIPTION

$$
\begin{aligned}
S_i &= x_i' y_i \\
E_i &= (x_i \oplus y_i)', \quad i = 0, \ldots, 3 \\
G_i &= x_i y_i'
\end{aligned}
$$

$$
\begin{aligned}
z^G &= G_3 + E_3 G_2 + E_3 E_2 G_1 + E_3 E_2 E_1 G_0 + E_3 E_2 E_1 E_0 c_{in}^G \\
z^E &= E_3 E_2 E_1 E_0 c_{in}^E \\
z^S &= S_3 + E_3 S_2 + E_3 E_2 S_1 + E_3 E_2 E_1 S_0 + E_3 E_2 E_1 E_0 c_{in}^S
\end{aligned}
$$

ITERATIVE COMPARATOR NETWORK

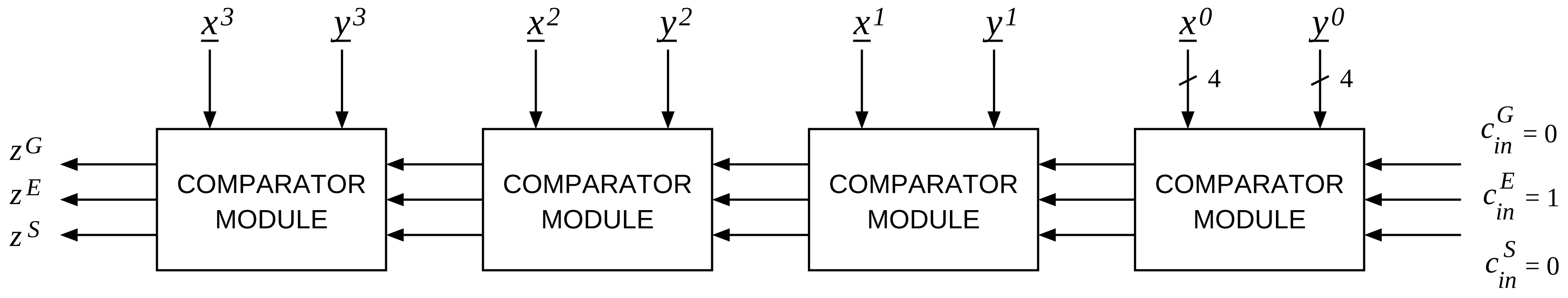


Figure 10.16: 16-BIT ITERATIVE COMPARATOR NETWORK.

TREE COMPARATOR NETWORK

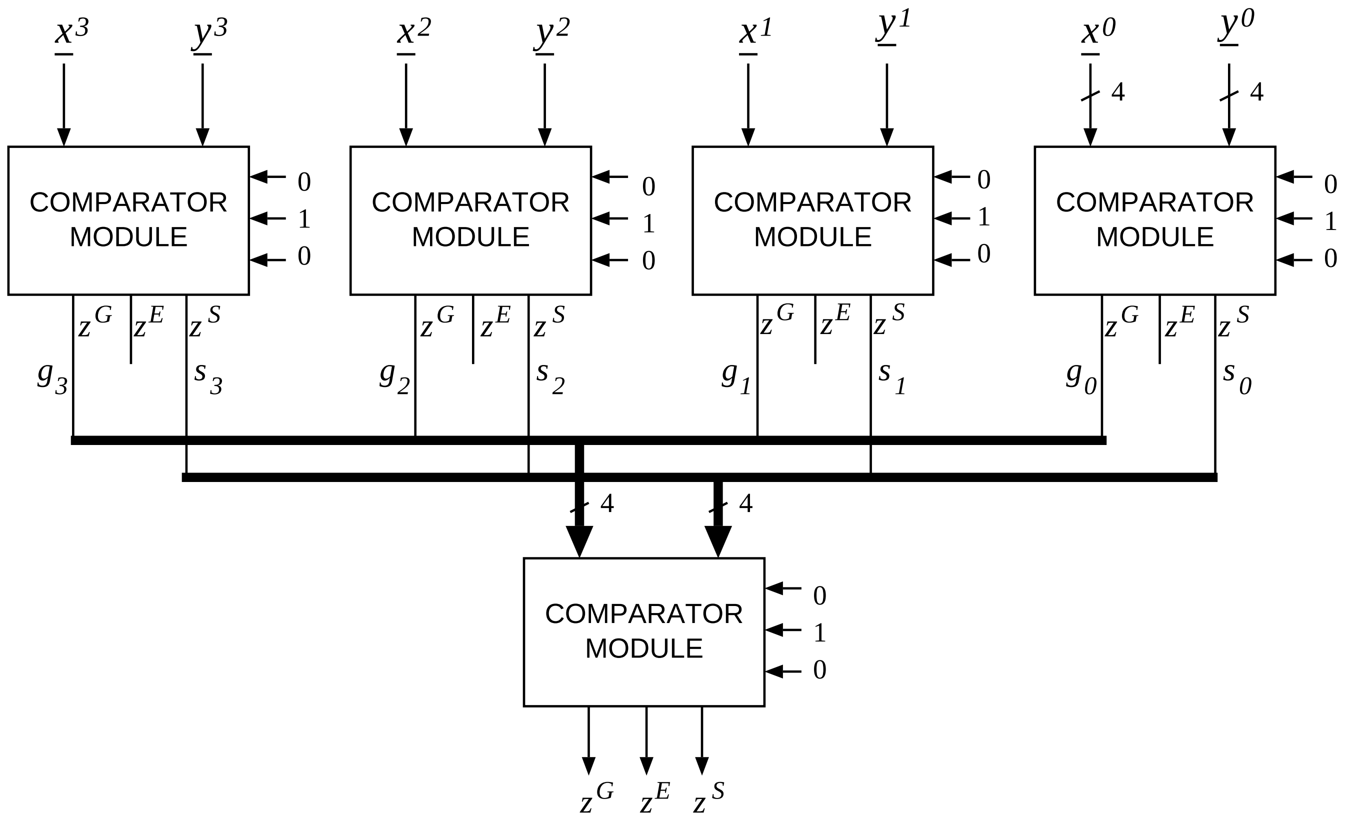


Figure 10.17: 16-BIT TREE COMPARATOR NETWORK.

TREE COMPARATOR (cont.)

$$z^G = \begin{cases} 1 & \textbf{if} \quad g > s \\ 0 & \textbf{otherwise} \end{cases}$$

$$z^E = \begin{cases} 1 & \textbf{if} \quad g = s \\ 0 & \textbf{otherwise} \end{cases}$$

$$z^S = \begin{cases} 1 & \textbf{if} \quad g < s \\ 0 & \textbf{otherwise} \end{cases}$$

- g and s are the integers represented by the vectors $\underline{g}$ and $\underline{s}$, respectively.

MULTIPLIERS

- $n \times m$ bits multiplier:
  $0 \leq x \leq 2^n - 1$ (the multiplicand)
  $0 \leq y \leq 2^m - 1$ (the multiplier),
  $0 \leq z \leq (2^n - 1)(2^m - 1)$ (the product).
- The high-level function:

$$z = x \times y$$

$$z = x(\sum_{i=0}^{m-1} y_i 2^i) = \sum_{i=0}^{m-1} x y_i 2^i$$

Since y_i is either 0 or 1, we get

$$xy_i = \begin{cases} 0 & \text{if } y_i = 0 \\ x & \text{if } y_i = 1 \end{cases}$$

MULTIPLICATION BIT MATRIX

$$\begin{array}{ccccccccccccccc}
 & & & & & & & x_7y_0 & x_6y_0 & x_5y_0 & x_4y_0 & x_3y_0 & x_2y_0 & x_1y_0 & x_0y_0 \\
 & & & & & & x_7y_1 & x_6y_1 & x_5y_1 & x_4y_1 & x_3y_1 & x_2y_1 & x_1y_1 & x_0y_1 & \\
 & & & & & x_7y_2 & x_6y_2 & x_5y_2 & x_4y_2 & x_3y_2 & x_2y_2 & x_1y_2 & x_0y_2 & & \\
 & & & & x_7y_3 & x_6y_3 & x_5y_3 & x_4y_3 & x_3y_3 & x_2y_3 & x_1y_3 & x_0y_3 & & & \\
 & & & x_7y_4 & x_6y_4 & x_5y_4 & x_4y_4 & x_3y_4 & x_2y_4 & x_1y_4 & x_0y_4 & & & & \\
 & & x_7y_5 & x_6y_5 & x_5y_5 & x_4y_5 & x_3y_5 & x_2y_5 & x_1y_5 & x_0y_5 & & & & &
\end{array}$$

Multiplier implementation:

- m arrays of n AND gates
- $m-1$ n-bit adders

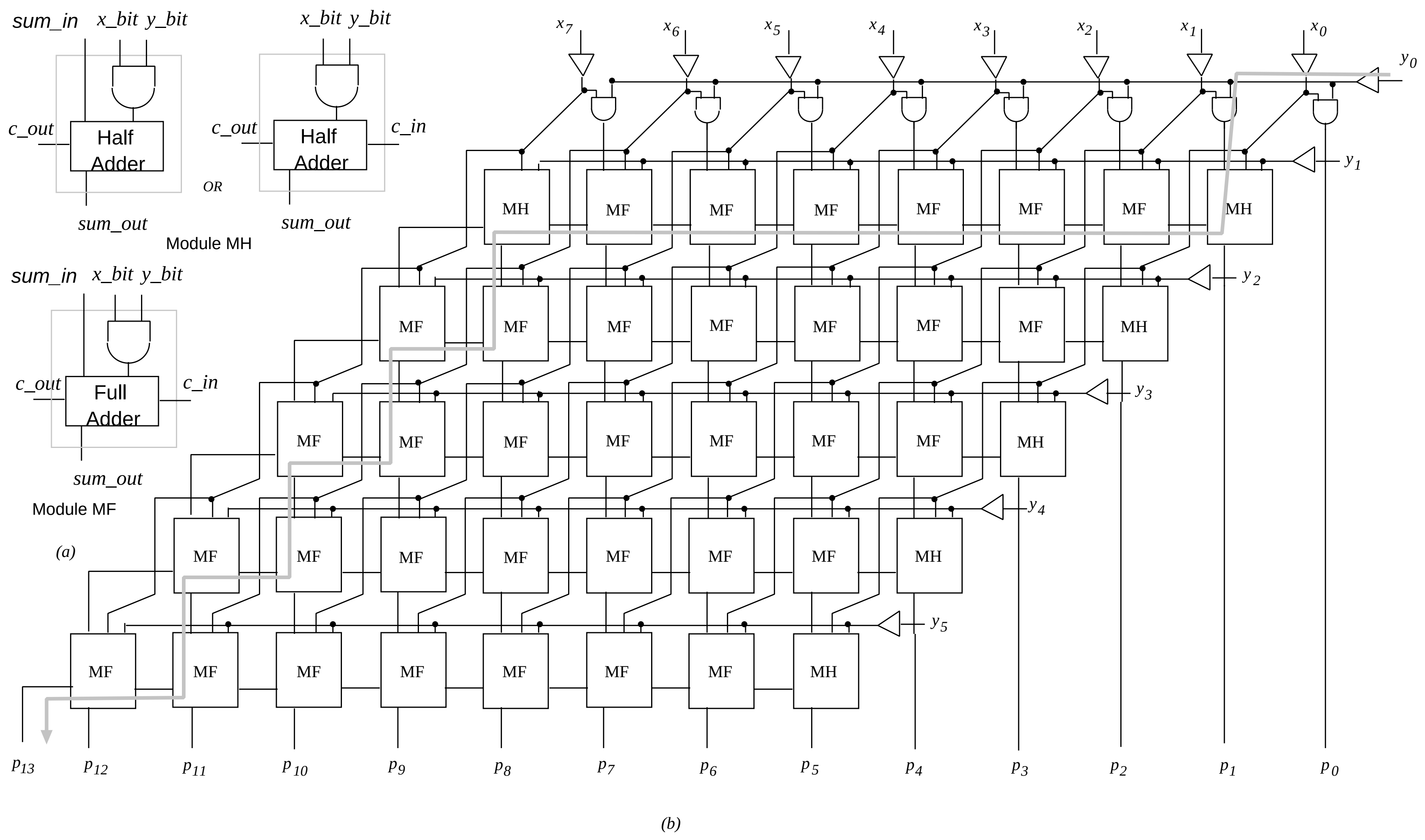

sum_in
x_bit
y_bit
c_out
Half Adder
sum_out
OR
c_out
Half Adder
c_in
sum_out
Module MH
sum_in
x_bit
y_bit
c_out
Full Adder
c_in
sum_out
Module MF
(a)
x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
y_0 y_1 y_2 y_3 y_4 y_5
MH MF
p_{13} p_{12} p_{11} p_{10} p_9 p_8 p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0
(b)

Figure 10.18: IMPLEMENTATION OF AN 8×6 MULTIPLIER: a) PRIMITIVE MODULES; b) NETWORK.

MULTIPLIER DELAY

- delay of the buffer which connects signal y_0 to the n AND gates
- delay of the AND gate
- delay of the adders

$$t_{\text{adders}} = t_c(n-1) + t_s + (t_c + t_s)(m-2)$$

If $t_s = t_c$, we get

$$t_{\text{adders}} = (n + 2(m-2))t_s = (n + 2m - 4)t_s$$

FOR THE 8×6 CASE: $t_{\text{adders}} = (8 + 12 - 4)t_s = 16t_s$

EXAMPLE OF NETWORKS WITH STANDARD ARITHMETIC MODULES

Inputs: $a[3], a[2], a[1], a[0], b[3], b[2], b[1], b[0] \in \{0, ..., 2^{16} - 1\}$

$\underline{e} = (e_3, e_2, e_1, e_0) \quad , \quad e_i \in \{0, 1\}$

Outputs: $c[3], c[2], c[1], c[0] \in \{0, ..., 2^{17} - 1\}$

$d \in \{0, 1, 2, 3\}$

$f \in \{0, 1\}$

Function:

$$f = \begin{cases} 1 \ \textbf{if} \quad \text{at least one } e_j = 1 \\ 0 \ \textbf{otherwise} \end{cases} \quad , \quad j = 0, 1, 2, 3$$

$$d = \begin{cases} i \ \textbf{if} \ \ e_i \text{ is the highest priority event} \\ 0 \ \textbf{if} \ \ \text{no event occurred} \end{cases}$$

$$c[i] = \begin{cases} a[i] + b[i] & \textbf{if} \ \ e_i \text{ is the highest priority event} \\ 0 & \textbf{otherwise} \end{cases}$$

MODULAR IMPLEMENTATION

CONSISTS OF

- a PRIORITY ENCODER to determine the highest-priority event;
- an ADDER;
- two SELECTORS (multiplexers) to select the corresponding inputs to the adder;
- a DISTRIBUTOR (demultiplexer) to send the output of the adder to the corresponding system output; and
- an OR gate to determine whether at least one event has occurred.

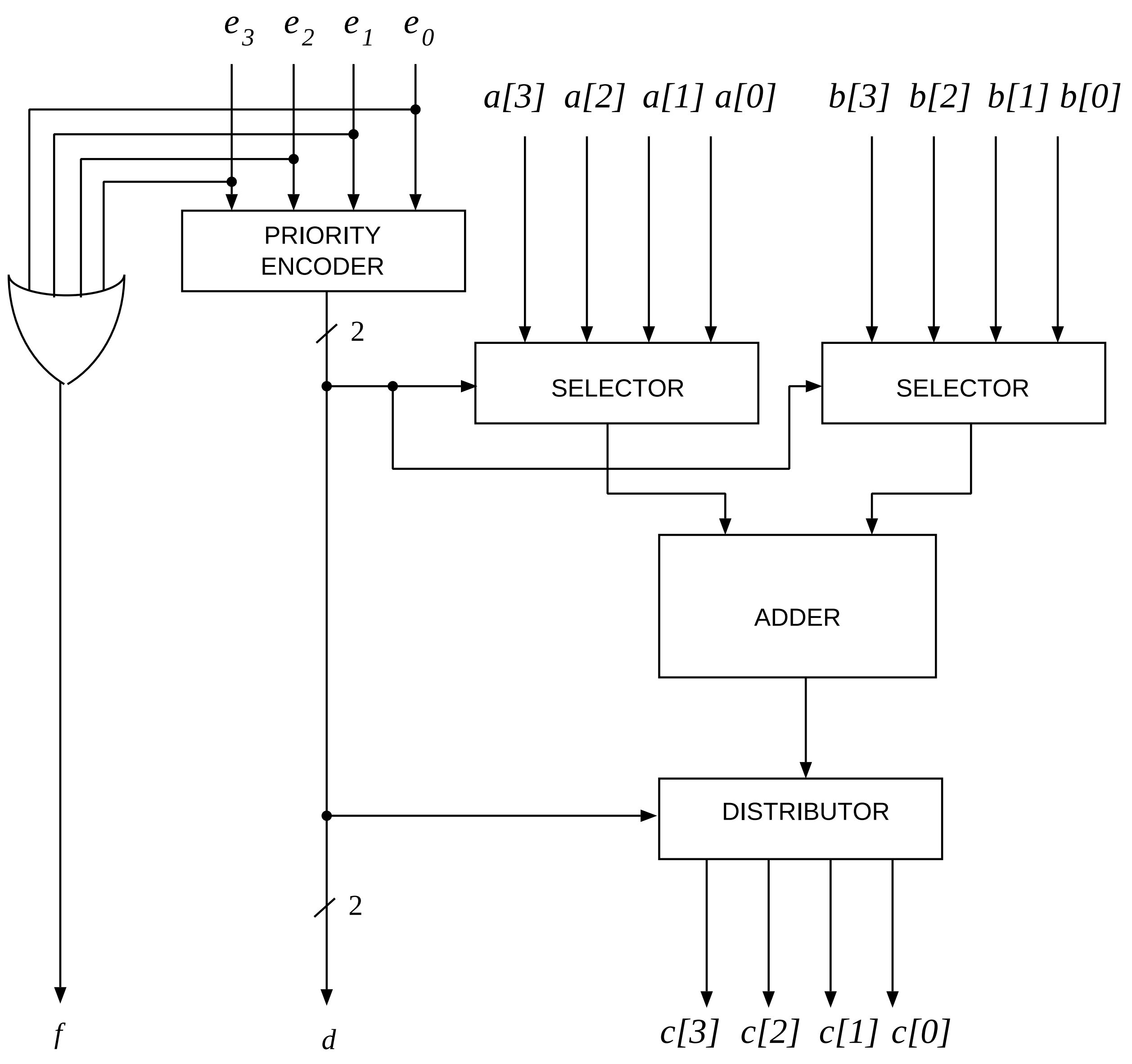


Figure 10.19: NETWORK IN EXAMPLE 10.5