ARITHMETIC COMBINATIONAL MODULES AND NETWORKS[1]

- 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

cont.

- 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:

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

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

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

CS M51A ADDITION OF POSITIVE INTEGERS

REPRESENTATION:

VALUE $x \in \{0, \ldots, 2^n - 1\}$

(n=6) BIT-VECTOR $\underline{x} = (x_5, x_4, x_3, x_2, x_1, x_0)$ $x_i \in \{0, 1\}$

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

RANGE: $\{\text{SMALLEST}, \ldots, \text{LARGEST}\} = \{0, \ldots, 2^n - 1\}$

(n=6) $x \in \{0, \ldots, 63\}$

EXAMPLES OF x AND $\underline{x}$

$x = 23 = 16 + 4 + 2 + 1$ $\underline{x} = (0\ 1\ 0\ 1\ 1\ 1)$

$\underline{x} = (\underbrace{1\ 1\ 0}_{6}\ \underbrace{0\ 1\ 1}_{3})$ $x = 1\times2^5 + 1\times2^4 + 1\times2^1 + 1\times2^0$

$= 6\times2^3 + 3 = 51$

— EASIER BY GROUPING INTO RADIX 2^b

ADDER MODULES FOR POSITIVE INTEGERS

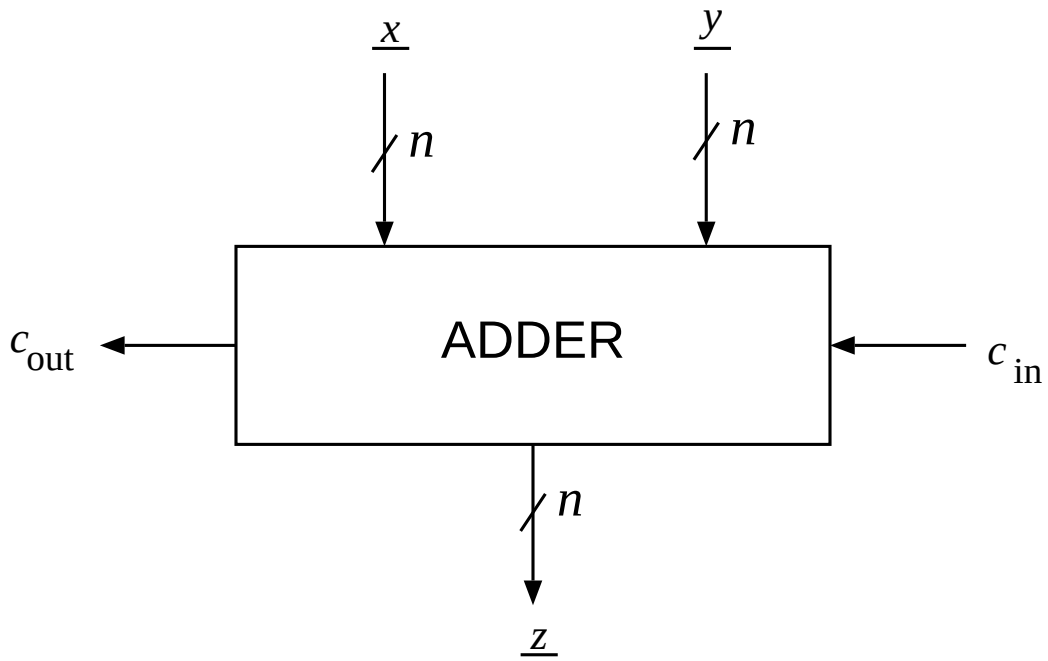


Figure 10.1: ADDER MODULE.

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

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

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

CS M51A ADDITION OF POSITIVE INTEGERS

$0 \le x, y \le 2^n - 1 \qquad c_{in}, c_{out} \in \{0, 1\}$

$x + y + c_{in} = 2^n c_{out} + z \qquad 0 \le z \le 2^n - 1$

HIGH LEVEL $\quad \overbrace{27}^{x} + \overbrace{51}^{y} + \overbrace{1}^{c_{in}} = \overbrace{2^6}^{c_{out}} \times 1 + \overbrace{15}^{z}$

$79 \qquad\qquad 64 + 15$

BINARY LEVEL

$x = 27 =$	0	1	1	0	1	1	
$y = 51 =$	1	1	0	0	1	1	
$c_{in} = 1$							1
w	1	0	0	1	1	1	1 $\quad = 79$
z		0	0	1	1	1	1
c_{out}	1						

$z = w \bmod 64 = 15$

$c_{out} = \lfloor w/64 \rfloor = 1$

RESULT $(1, 0\ 0\ 1\ 1\ 1\ 1) = 79$

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

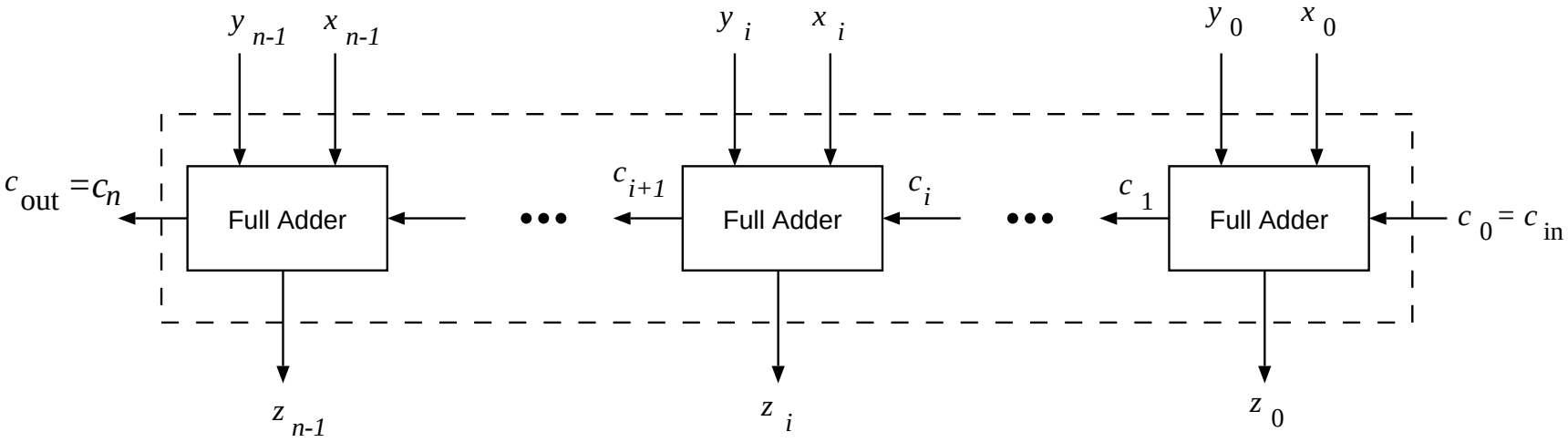


Figure 10.2: CARRY-RIPPLE ADDER MODULE.

- DELAY OF CARRY-RIPPLE ADDER

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

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

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

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

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

$$z_i = x_i y_i' c_i' + x_i' y_i c_i' + x_i' y_i' c_i + x_i y_i c_i$$

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

FULL ADDER TWO-LEVEL IMPLEMENTATION

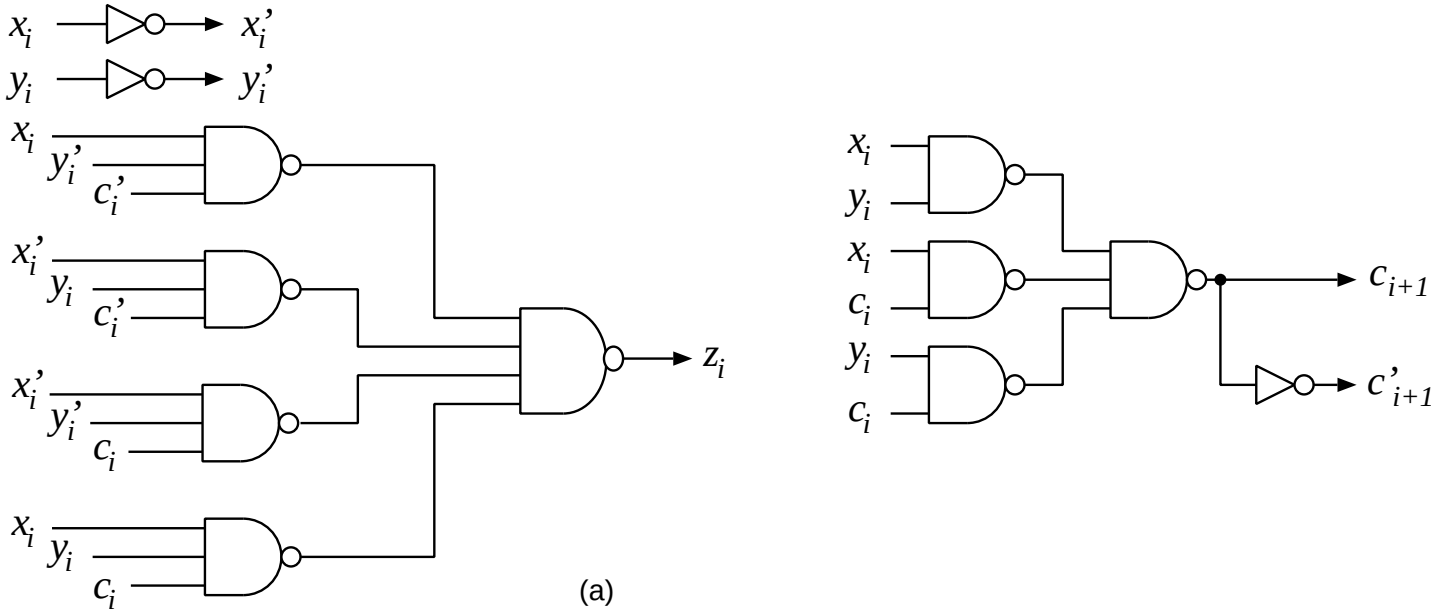


Figure 10.3: IMPLEMENTATIONS OF FULL-ADDER MODULE: a) TWO-LEVEL.

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 $c_i = 1)$:

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

cont.

- INTERMEDIATE VARIABLES

$$\text{PROPAGATE} \qquad p_i = x_i \oplus y_i$$
$$\text{GENERATE} \qquad g_i = x_i \cdot y_i$$

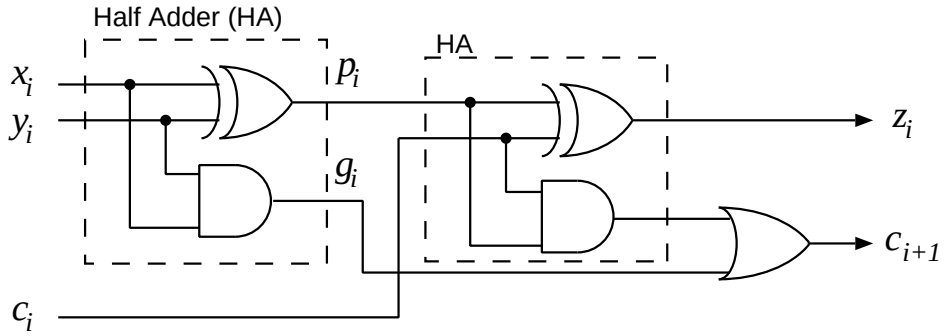
- HALF-ADDER

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

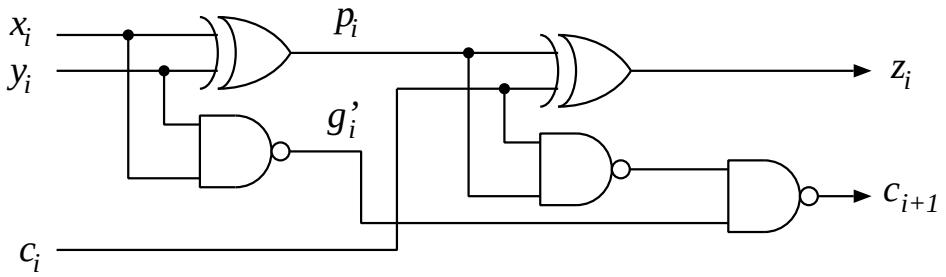
IMPLEMENTATION WITH HALF-ADDERS

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

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



(b)



(c)

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

CSM51A FA TWO-LEVEL IMPLEMENTATION

DELAY ANALYSIS

- THE SUM NETWORK IS NOT IN THE CRITICAL PATH: DO DELAY ANALYSIS $t(x_i, z_i)$, $t(c_i, z_i)$
- THE CARRY NETWORK IS IN THE CRITICAL PATH

LOAD OF c_i: $L=5$ ← (x_i, y_i NOT CRITICAL)

c_i': $L=2$

USE NAND GATES

$$t_{LH}(c_i, c_{i+1}) = t_{LH}(NAND2, L=1) + t_{LH}(NAND3, L=5)$$
$$= 0.088 + 0.26 = \boxed{0,348 ns}$$

$$t_{HL}(c_i, c_{i+1}) = t_{HL}(NAND2, L=1) + t_{HL}(NAND3, L=5)$$
$$= 0.107 + 0.285 = \boxed{0.392 ns}$$

ALTERNATIVE IMPLEMENTATION

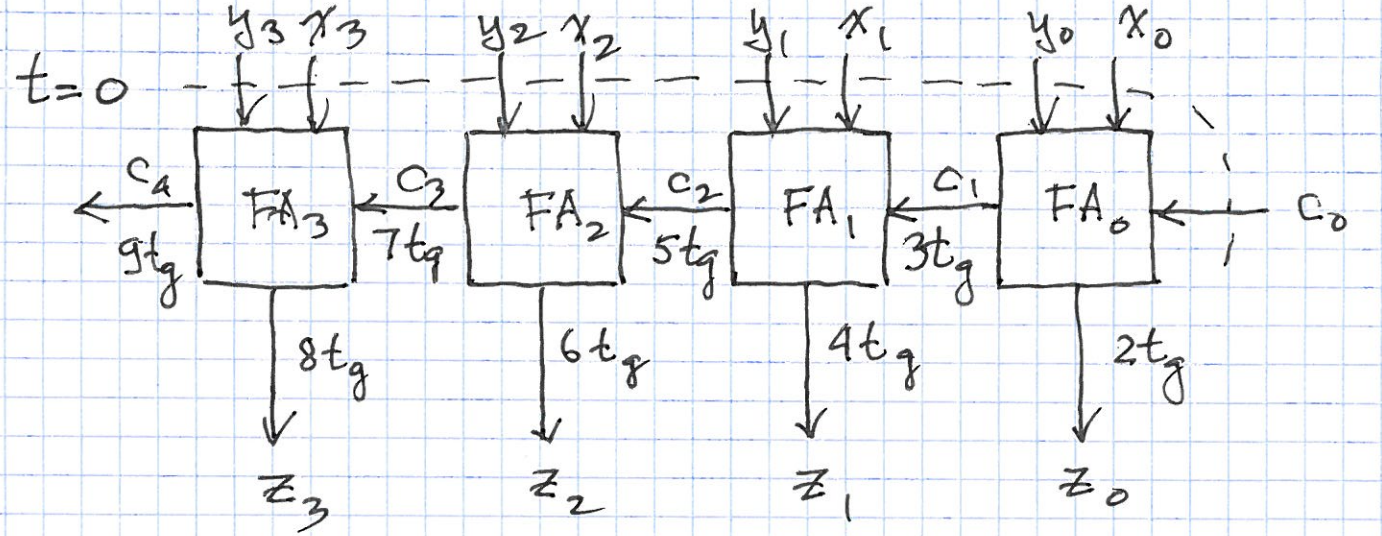
(HA-HA-OR)

$$t_{LH}(c_i, c_{i+1}) = t_{LH}(NAND2, 1) + t_{LH}(NAND2, 2.1)$$
$$= 0.088 + 0.1298 = \boxed{0,218 ns}$$ FASTER 37%

$$t_{HL}(c_i, c_{i+1}) = t_{HL}(NAND2, 1) + t_{HL}(NAND2, 2.1)$$
$$= 0,107 + 0.1367 = \boxed{0.244 ns}$$ 37%

CS M51A

4-BIT CARRY-RIPPLE ADDER (CRA)



ASSUME: $t_{NAND} = t_{XOR} = t_g$

$$T_{CRA-4} = 9t_g \Rightarrow O(n)$$

$$p_i = x_i \oplus y_i$$

$$g_i' = (x_i y_i)'$$

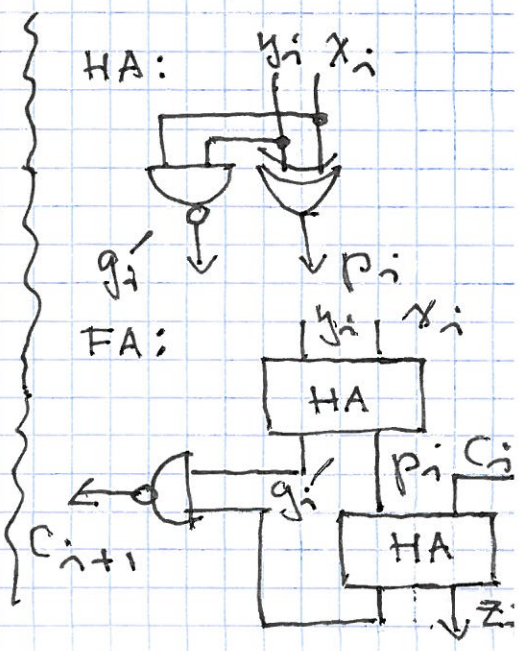
$$z_i = p_i \oplus c_i$$

$$c_{i+1} = g_i + p_i c_i = \left[g_i' \cdot (p_i c_i)' \right]'$$

$$t_{z_0} = 2t_{XOR} = 2t_g \qquad t_{z_i} = t_{XOR} = t_g$$

$$t_c = 2t_{NAND} = 2t_g$$

LINEAR



WORST-CASE DELAY

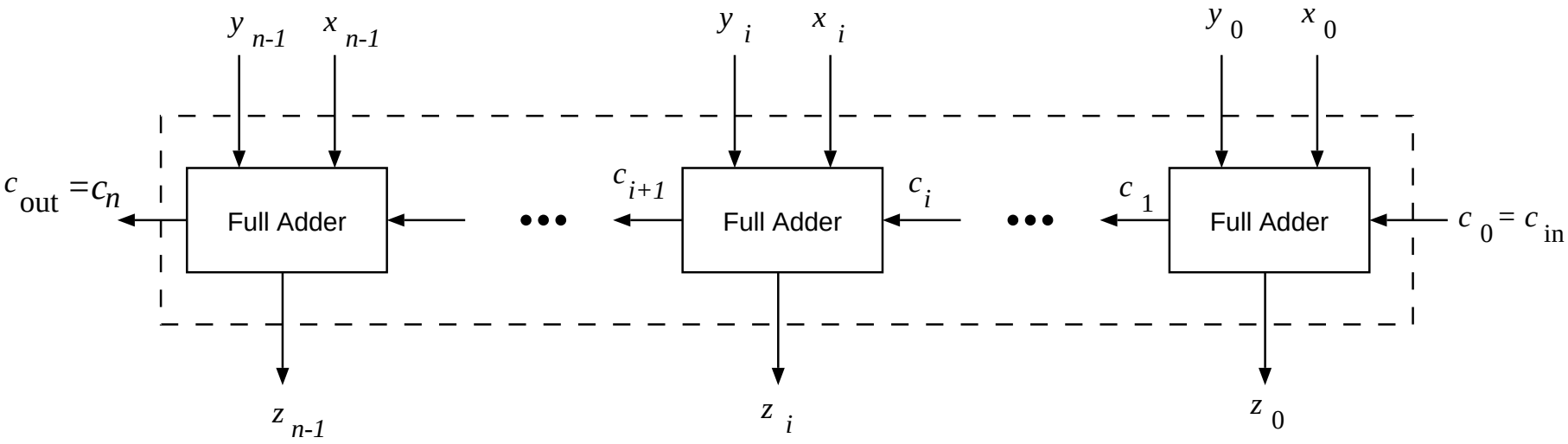


Figure 10.2: CARRY-RIPPLE ADDER MODULE.

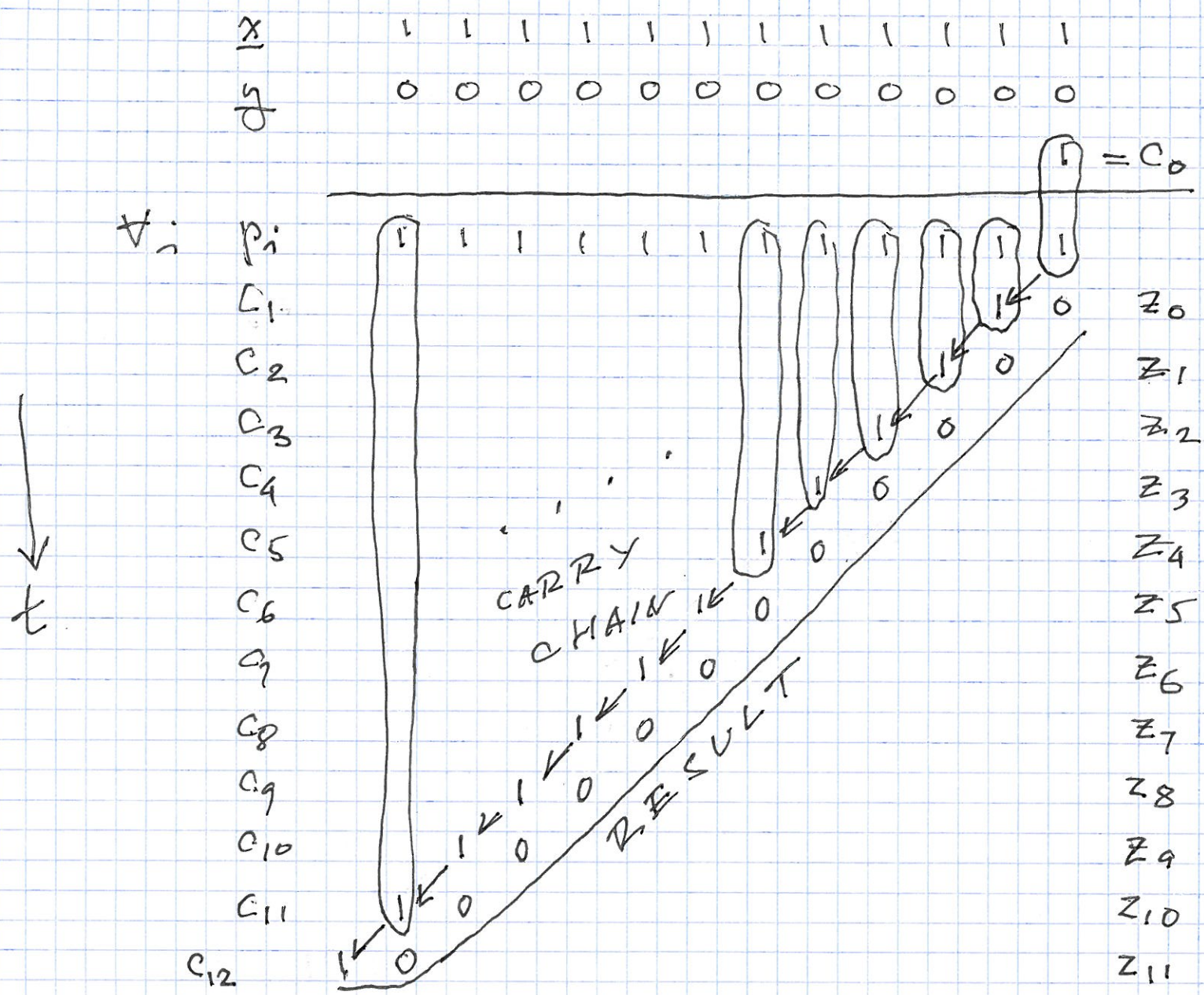
- WORST-CASE DELAY

$$t_p = t_{\text{XOR}} + 2(n-1)t_{\text{NAND}} + \max(2t_{\text{NAND}}, t_{\text{XOR}})$$

CS M51A

CARRY-RIPPLE ADDER

THE WORST CASE DELAY

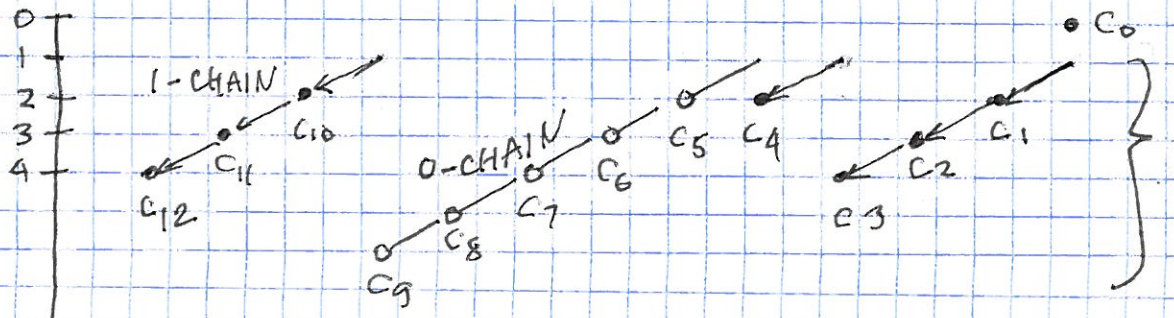


CSMSIA FIQ MDE

ABOUT CARRIES

i	11	10	9	8	7	6	5	4	3	2	1	0
x_i	1	0	1	0	1	1	0	0	1	1	0	1
y_i	0	1	1	0	0	0	1	0	1	0	1	0
p_i	1	1	0	0	1	1	1	0	0	1	1	1
g_i	0	0	1	0	0	0	0	0	1	0	0	0
c_i	1	1	1	0	0	0	0	0	1	1	1	1 (1) = c_0
z_i	1	0	0	0	0	1	1	1	1	0	0	0

CARRY CHAINS (PROPAGATE IN PARALLEL)



WHY CONSIDER 0-CARRY CHAINS?

WORST-CASE CARRY-CHAIN LENGTH: $\sim 2nt_g$

AVERAGE MAX CARRY-CHAIN LENGTH: $\sim \log_2(n)\, t_g$

n	16	32	64
	32	64	128
	4	5	6

$\Rightarrow$ MUST DETECT WHEN ALL CARRY CHAINS ARE COMPLETED: SELF-TIMED ADDER

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

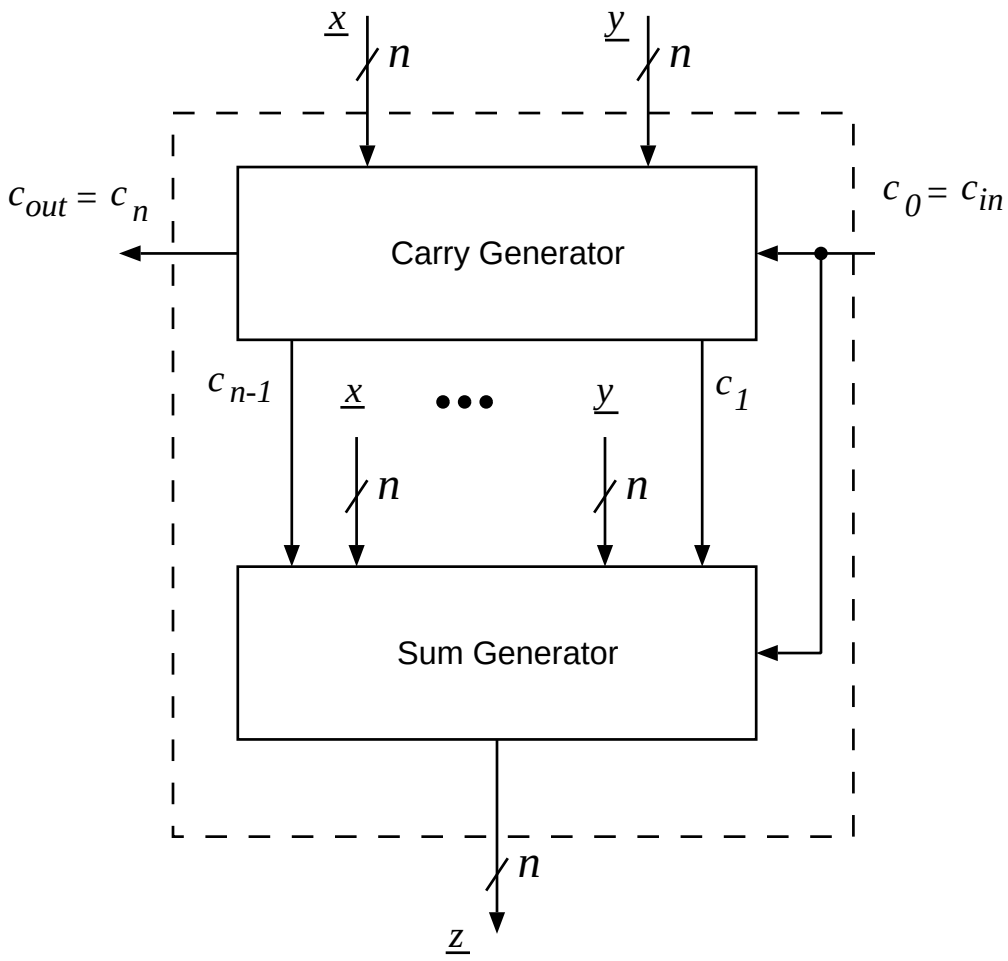


Figure 10.4: CARRY-LOOKAHEAD ADDER MODULE.

SWITCHING EXPRESSIONS

- INTERMEDIATE CARRIES:

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

BY SUBSTITUTION,

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

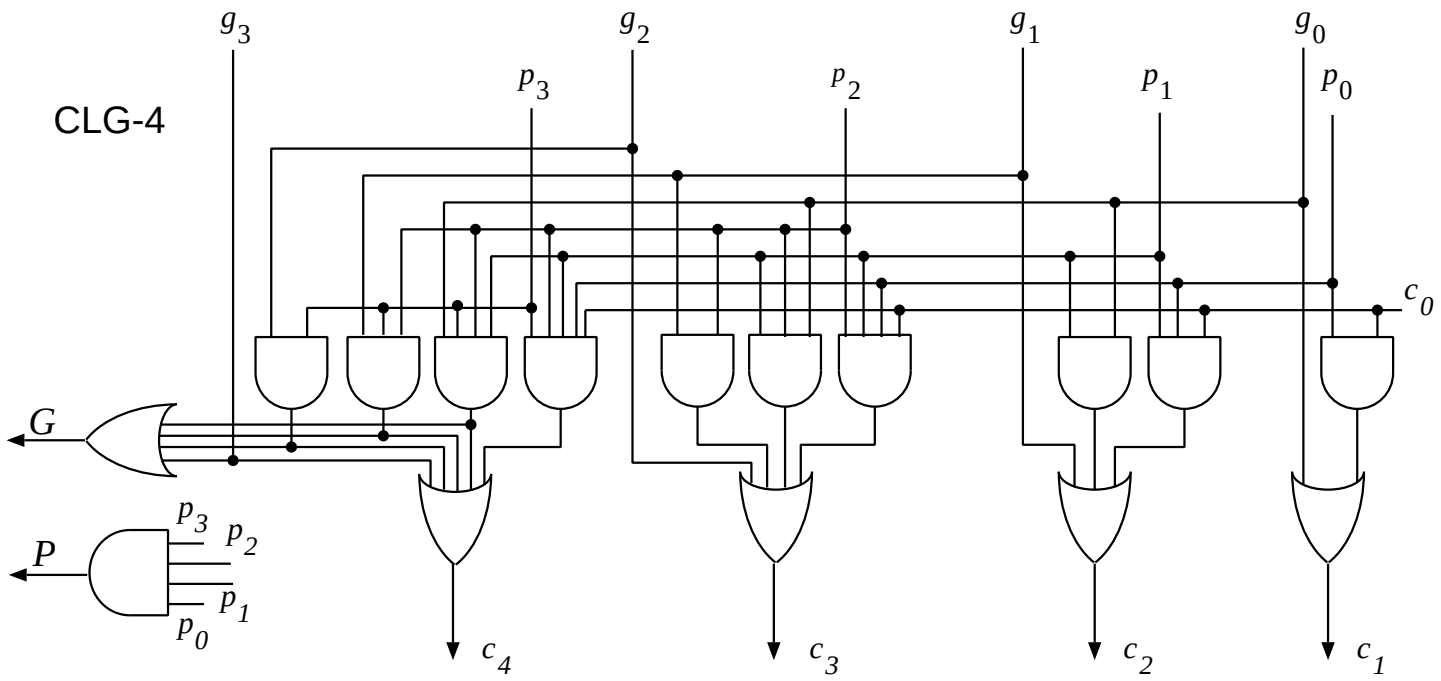
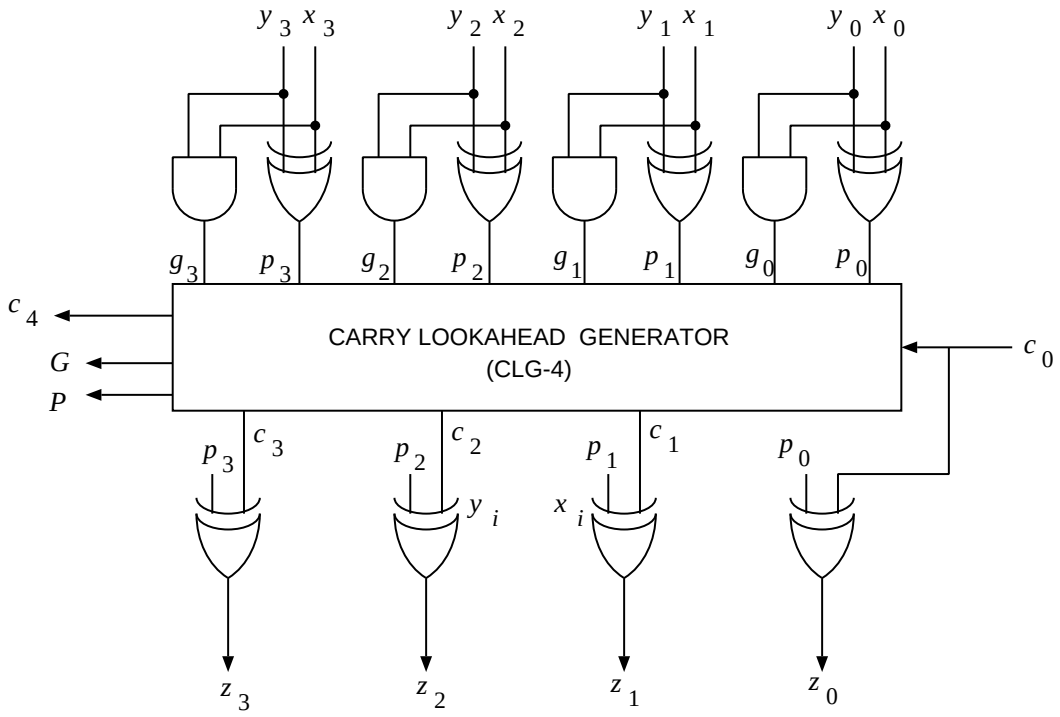


Figure 10.5: CARRY-LOOKAHEAD ADDER: a) 4-BIT CARRY-LOOKAHEAD GENERATOR WITH P and G OUTPUTS (CLG-4).



(b)

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

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

CS M51A

COMPARE DELAYS OF 4-BIT CRA AND 4-BIT CLA

- ASSUME GATE DELAY FOR ALL GATES IS t_g
  $\Rightarrow$ THIS GIVES A ROUGH ESTIMATE

- REPEAT ANALYSIS USING DELAY FORMULAS FOR EACH GATE; ALSO SIZE IN EQUIV. GATES

$$T_{CRA-4} = t_g + 4 \times 2t_g \sim 9t_g$$

— OBTAIN p_i, g_i SIGNALS
— PROPAGATE CARRIES AND OBTAIN z_i's

$$T_{CLA-4} = t_g + 2t_g + t_g \sim 4t_g$$

— OBTAIN p_i, g_i
— OBTAIN ALL CARRIES c_i
— OBTAIN z_i (SUM BITS)

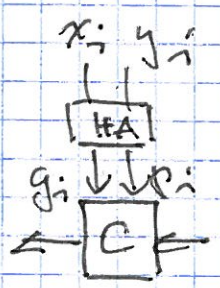
FACTOR ~ 2

$$T_{CRA-32} = t_g + 32 \times 2t_g \sim 65t_g$$

$$T_{CLA-32} = 8 \times T_{CLA-4} \sim 32t_g$$

CSMSIA F14 MDE

GENERALIZATION OF g_i, p_i AND c_i SIGNALS

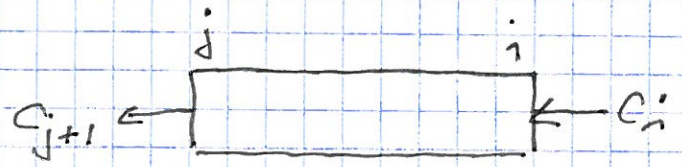


$g_i = x_i y_i$ $\quad g_i = 1$ IF $x_i + y_i = 2$ $g_i = 0$ OTHERWISE

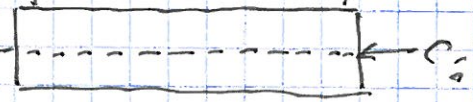
$p_i = x_i \oplus y_i$ $\quad p_i = 1$ IF $x_i + y_i = 1$ $p_i = 0$ OTHERWISE

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

CONSIDER A GROUP IN AN ADDER BETWEEN POSITIONS i AND j



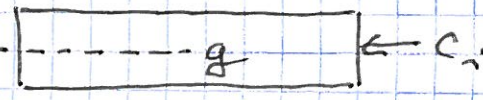
2 POSSIBLE CASES:

(a) 

GROUP PROPAGATES IN ALL POSITIONS

$P = p_j p_{j-1} \cdots p_{i+1} p_i = 1 \Rightarrow p_k = 1 \ \forall \ j \geq k \geq i$

$c_{j+1} = P c_i$

(b) 

GROUP GENERATES c_{j+1}, INDEPENDENT OF c_i

$G = g_j + p_j g_{j-1} + p_j p_{j-1} g_{j-2} \cdots + p_j p_{j-1} \cdots p_{i+1} g_i$

$c_{j+1} = G$

COMBINE (a) AND (b):

$c_{j+1} = G + P c_i$

— APPLIES TO ANY GROUP SIZE

(a) 1 1 0 0 0 1
0 0 1 1 1 0
p p p p p p ← ALL $p_k = 1$

(b) 0 1 1 1 0 0
1 0 0 1 1 1
← p p p g p p

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}

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

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

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

$$P = p_3 p_2 p_1 p_0$$

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

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

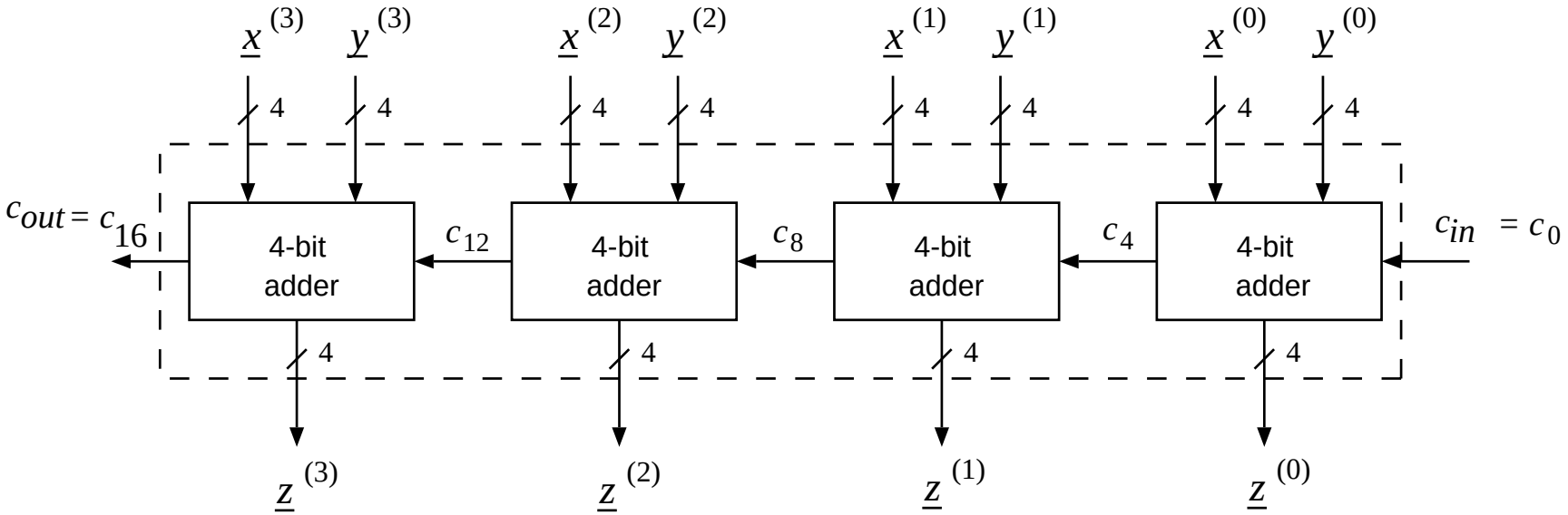
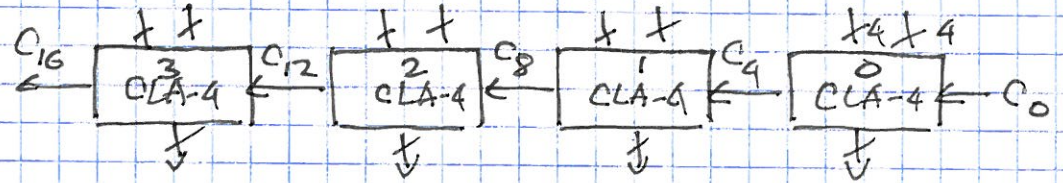


Figure 10.6: 16-BIT CARRY-RIPPLE ADDER NETWORK USING 4-BIT ADDER MODULES.

CSMSIA F14

SINGLE-LEVEL CARRY-LOOKAHEAD ADDER:



$T = 4 \times T_{CLA\text{-}4} \approx 4 \times 4t_g = 16t_g$

STILL DELAY $O(n)$. CAN WE DO BETTER?

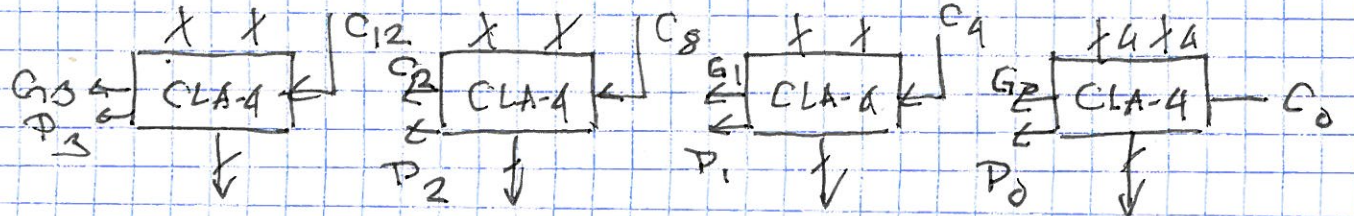
YES C_8?

$C_8 = G_1 + P_1 G_0 + P_1 P_0 C_0$ } G – GROUP GENERATE, P – " PROPAGATE

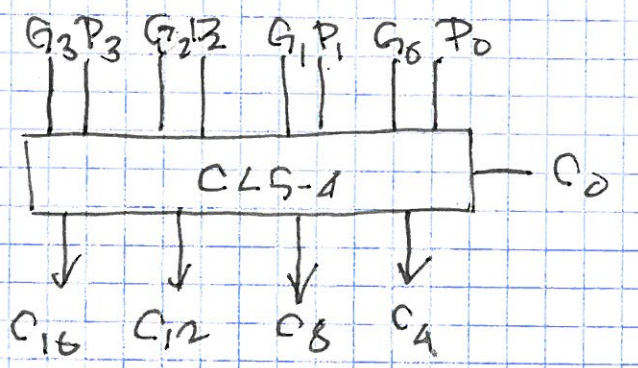
$C_{12} = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

$C_{16} = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$

— CORRESPOND TO CLG-4!



TWO-LEVEL CLA (CLA-2)



CARRY-LOOKAHEAD ADDER NETWORK

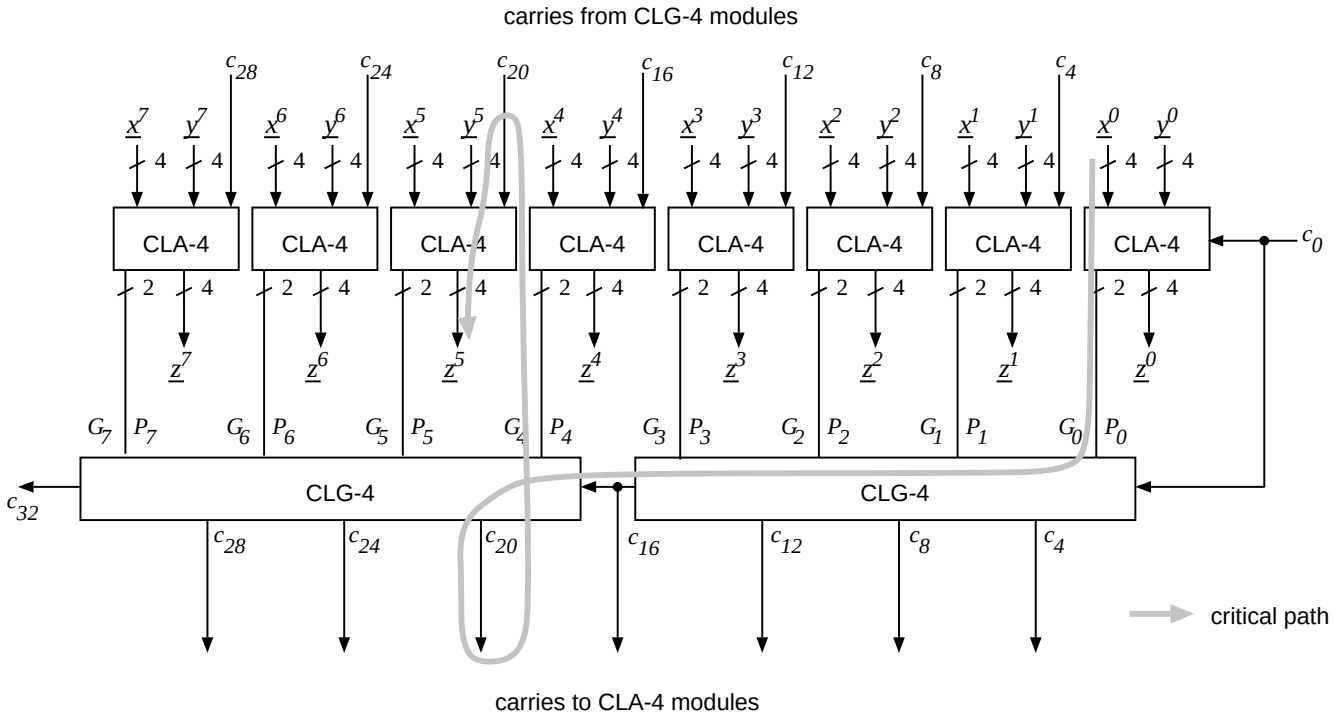


Figure 10.7: 32-BIT CARRY-LOOKAHEAD ADDER USING CLA-4 AND CLG-4 MODULES.

- PROPAGATION DELAY:

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

CLA ADDER (cont.)

$$
\begin{aligned}
c_4 &= G_0 + P_0 c_0 \\
c_8 &= G_1 + P_1 G_0 + P_1 P_0 c_0 \\
c_{12} &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0 \\
c_{16} &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0
\end{aligned}
$$

$$
\begin{aligned}
P_0 &= p_3 \cdot p_2 \cdot p_1 \cdot p_0 \\
G_0 &= g_3 + g_2 p_3 + g_1 p_3 p_2 + g_0 p_3 p_2 p_1
\end{aligned}
$$

CSM 51A F14 MDE

APPROXIMATE DELAY OF A 32-BIT 2-LEVEL CARRY-LOOKAHEAD LEVEL

$T_{CLA-32} = t_{pg}$ — COMPUTE $p'_{i+1}, g'_{i:0}, \forall i$

$+ t_{CLG-4}$ — " $Q_j, P_j \; j=0,1,\ldots,7$

" c_1, c_2, c_3

$+ t_{CLG-4}$ — " c_4, c_8, c_{12}, c_{16}

$+ t_{CLG-4}$ — " $c_{20}, c_{24}, c_{28}, c_{32}$

" c_5, c_6, c_7

c_9, c_{10}, c_{11}

c_{13}, c_{14}, c_{15}

$+ t_{CLG-4}$ — " c_{17}, c_{18}, c_{19}

c_{21}, c_{22}, c_{23}

c_{29}, c_{30}, c_{31}

$+ t_{XOR}$ — " $z_{31}, \ldots, z_0$

CS M51A DELAY OF CLA-2

$t_{pg} = 1t_g$

$t_{CLA\text{-}4} = 2t_g$

$t_{XOR} = 1t_g$

NOTE: Z_i COMPUTED AS SOON AS C_i IS KNOWN.

COULD YOU SHOW A TIMING DIAGRAM FOR $\{C_i, Z_i\}$

$T_{CLA\text{-}32} = 1 + 4 \times 2 + 1 = 10t_g$

$T_{CRA\text{-}32} = 32 \times 2 = 64t_g$

$\frac{T_{CRA\text{-}32}}{T_{CLA\text{-}32}} > 6$ TIMES FASTER

CALCULATE COST (CLA-32) AND COST (CRA-32) AND COMPARE

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)

CS MSIA

SM REPRESENTATION

$$\underline{x} = (x_s, \; x_m) \qquad -(2^{n-1}-1) \le x \le 2^{n-1}-1$$

1 bit: x_s — $n-1$ bits: x_m

SIGN MAGNITUDE

$n = 8 \qquad 0 \le x_m \le 2^7 - 1 \qquad -127 \le x \le 127$

$x = -69 \qquad \underline{x} = 1 \; \underbrace{1 \; 0 \; 0 \; 0 \; 1 \; 0 \; 1}$

$-1 \qquad\qquad 69$

POSITIVE ZERO: 0 0 0 0 0 0 0 0

NEGATIVE ZERO: 1 0 0 0 0 0 0 0

TEST FOR ZERO: CHECK BOTH ± 0

NOTE: x_s IS SEPARATED FROM x_m

$$x = (-1)^{x_s} \cdot \sum_{i=0}^{n-2} x_i 2^i$$

(x_s NOT PART OF WEIGHTED CODE FOR x_m)

CS MSIA

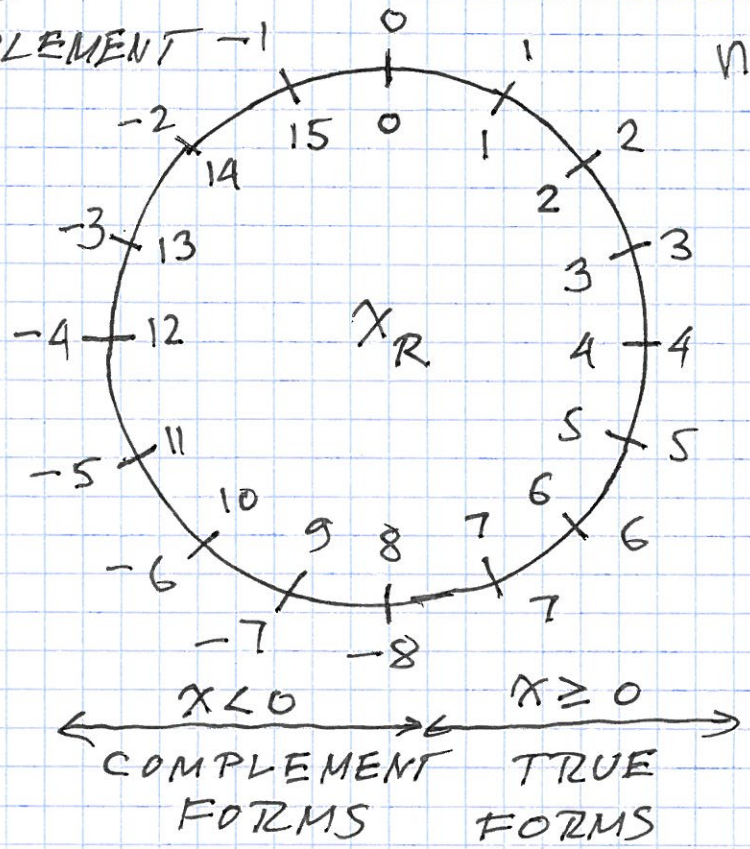
COMPLEMENT REPRESENTATION SYSTEMS

$x \in \{-8, -7, \ldots, -1, 0, 1, \ldots, 6, 7\}$

$x \in \{-7, -6 \ldots -1, -0, 0, 1, \ldots, 6, 7\}$

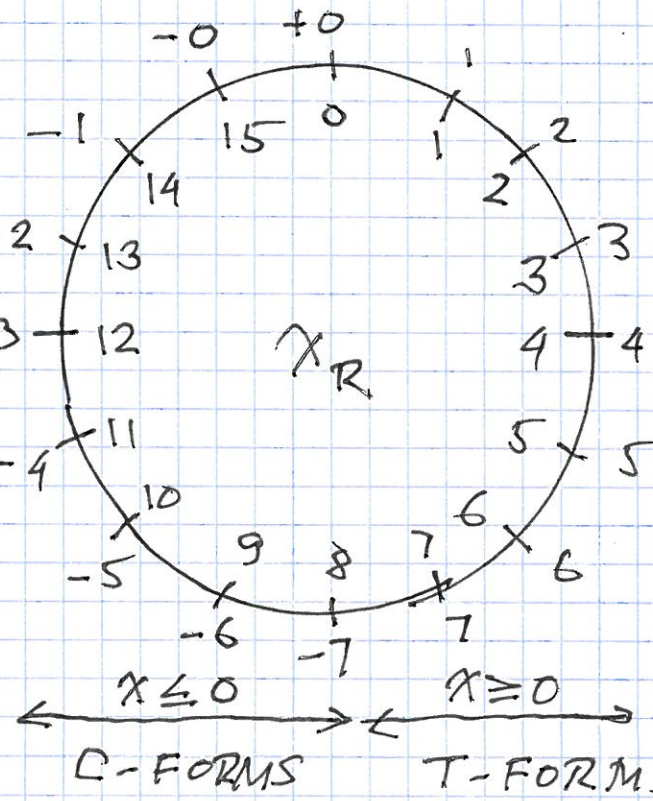
$x_R \in \{0, 1, 2, \ldots, 14, 15\}$

TWO'S COMPLEMENT



$x_R = 2^4 - |x|$ $\quad x_R = x$

1s' COMPL.



$x_R = 2^4 - 1 - |x|$ $\quad x_R = x$

COMPLEMENTATION CONSTANT

$C = 2^4 = 16$

IN GENERAL $C = 2^n$

$C = 2^4 - 1 = 15$

$C = 2^n - 1$

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 $\bmod$, FOR $|x| < 2^n$: equivalent to

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

CS MSIA TWO'S COMPLEMENT SYSTEM

1. GIVEN $x \in \{-2^{n-1}, \ldots, -1, 0, 1, \ldots, 2^{n-1}-1\}$, FIND x_R

$$x_R = x \bmod 2^n = \begin{cases} x & \text{IF } x \geq 0 \\ 2^n - |x| & \text{IF } x < 0 \end{cases}$$

NOTE: Mod FUNCTION PRODUCES THE LEAST POSITIVE REMAINDER

E.G. $(-1) \bmod 16 = 15$ I.E. $(-1) = (-1)\cdot 16 + 15$

EXAMPLE: $x \in [-128, 127]$ $n = 8$ $C = 2^8 = 256$

x	x_R	x_R
81	81	0 1 0 1 0 0 0 1
-23	233	1 1 1 0 1 0 0 1

ALTERNATIVE WHEN $x < 0$ (BIT COMPLEMENT AND ADD 1)

$$x_R = 256 - |x| = 255 - |x| + 1$$

"255"	1 1 1 1 1 1 1 1
23	0 0 0 1 0 1 1 1
"255" - 23	1 1 1 0 1 0 0 0
+1	1
	1 1 1 0 1 0 0 1

BIT COMPLEMENT

$\leftrightarrow -23$

CS M51A TWO'S COMPLEMENT SYSTEM (CONT.)

2. GIVEN x_R, FIND x

$$x = \begin{cases} x_R & \text{IF } x_R < 2^{n-1} \\ x_R - 2^n & \text{IF } x_R \geq 2^{n-1} \end{cases}$$

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

EXAMPLE: $n = 6 \quad C = 2^6 = 64$

x_R

$43 \geq 32 \quad x = 43 - 64 = -21$

3. GIVEN $\underline{x_R}$, FIND x

i) IF $x_R < 2^{n-1}$, $x_{n-1} = 0$, $x \geq 0$

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

ii) IF $x_R \geq 2^{n-1}$, $x_{n-1} = 1$, $x < 0$

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

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

CS M51A CONVERSION $\underline{x}_R \rightarrow x$

$$\underline{x}_R = (\underbrace{x_{n-1}}_{\text{NEGATIVE WEIGHT}}, \underbrace{x_{n-2}, \ldots, x_0}_{\geq 0})$$

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

EXAMPLES

$n=4$

$$\underline{x}_R = 1011 = 11 \geq 2^3 \Rightarrow x = 11 - 2^4 = -5$$

$$x = -1 \cdot 2^3 + 3 = -5 \quad \checkmark$$

$n=8$

$$\underline{x}_R = 11011101 = 173 \geq 2^7 \Rightarrow x = 173 - 2^8 = -83$$

$$x = -1 \cdot 2^7 + 45 = -83 \quad \checkmark$$

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

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

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	

SM ADDITION

— OPERANDS HAVE DIFFERENT SIGNS

⇒ ACTUALLY WE PERFORM SUBTRACTION

AN ALGORITHM:

1. COMPARE x_m AND y_m
2. SUBTRACT SMALLER FROM LARGER
3. SIGN OF THE RESULT
   = SIGN OF THE OPERAND WITH LARGER MAGNITUDE.

$x = 29 \quad y = -45 \quad |y| > |x|$

$w = |y| - |x| = 16$

$sign(s) = sign(y)$

$s = -w = -16$

NEED TO KNOW RELATIVE MAGNITUDES

⇒ MORE COMPLEX IMPLEMENTATION THAN IN COMPLEMENT ADDITION

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: SHOW THAT

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

CORRESPONDS TO z_R

BY DEFINITION OF THE REPRESENTATION,

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

cont.

THEREFORE,

$$(x_R + y_R)\ mod\ 2^n = (x\ mod\ 2^n + y\ mod\ 2^n)\ mod\ 2^n$$

$$= (x + y)\ mod\ 2^n$$

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

BY DEFINITION OF REPRESENTATION $z\ mod\ 2^n = z_R$

AND, CONSEQUENTLY, $(x_R + y_R)\ mod\ 2^n = z_R$

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

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

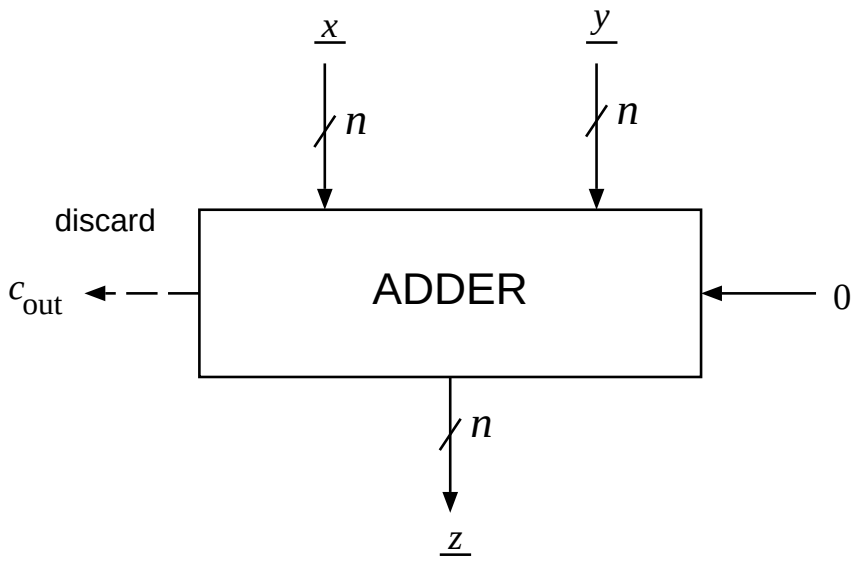


Figure 10.10: TWO'S-COMPLEMENT ADDER MODULE.

EXAMPLES OF 2'S COMPLEMENT ADDITION

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

CS M51A

CHANGE OF SIGN

OBTAIN Z_R FROM X_R SUCH THAT $Z = -X$

OPERATION IS: (COMPLEMENTATION)

$$Z_R = (2^n - X_R) \bmod 2^n$$

PROOF: IN THE TEXTBOOK — DO IT!

EXAMPLE: $X = 17 = X_R$ $\quad \underline{X_R} = 0\ 1\ 0\ 0\ 0\ 1$

$(n = 6)$

$$Z_R = (64 - 17) \bmod 64 = 47$$

$$\underline{Z_R} = 1\ 0\ 1\ 1\ 1\ 1$$

IMPLEMENTATION:

DO $2^n - X_R$ (SUBTRACTION) AS

COMPLEMENT $\longrightarrow (2^n - 1 - X_R) + 1 = \overline{X_R} + 1$

EXAMPLE: $n = 6$ $\quad X = 17 = X_R$

$$\begin{array}{rcccccc} & \underline{X_R} & 0 & 1 & 0 & 0 & 0 & 1 \\ & \overline{X_R} & 1 & 0 & 1 & 1 & 1 & 0 \\ + & & & & & & & 1 \\ \hline & & 1 & 0 & 1 & 1 & 1 & 1 \end{array}$$

BIT-WISE

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

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

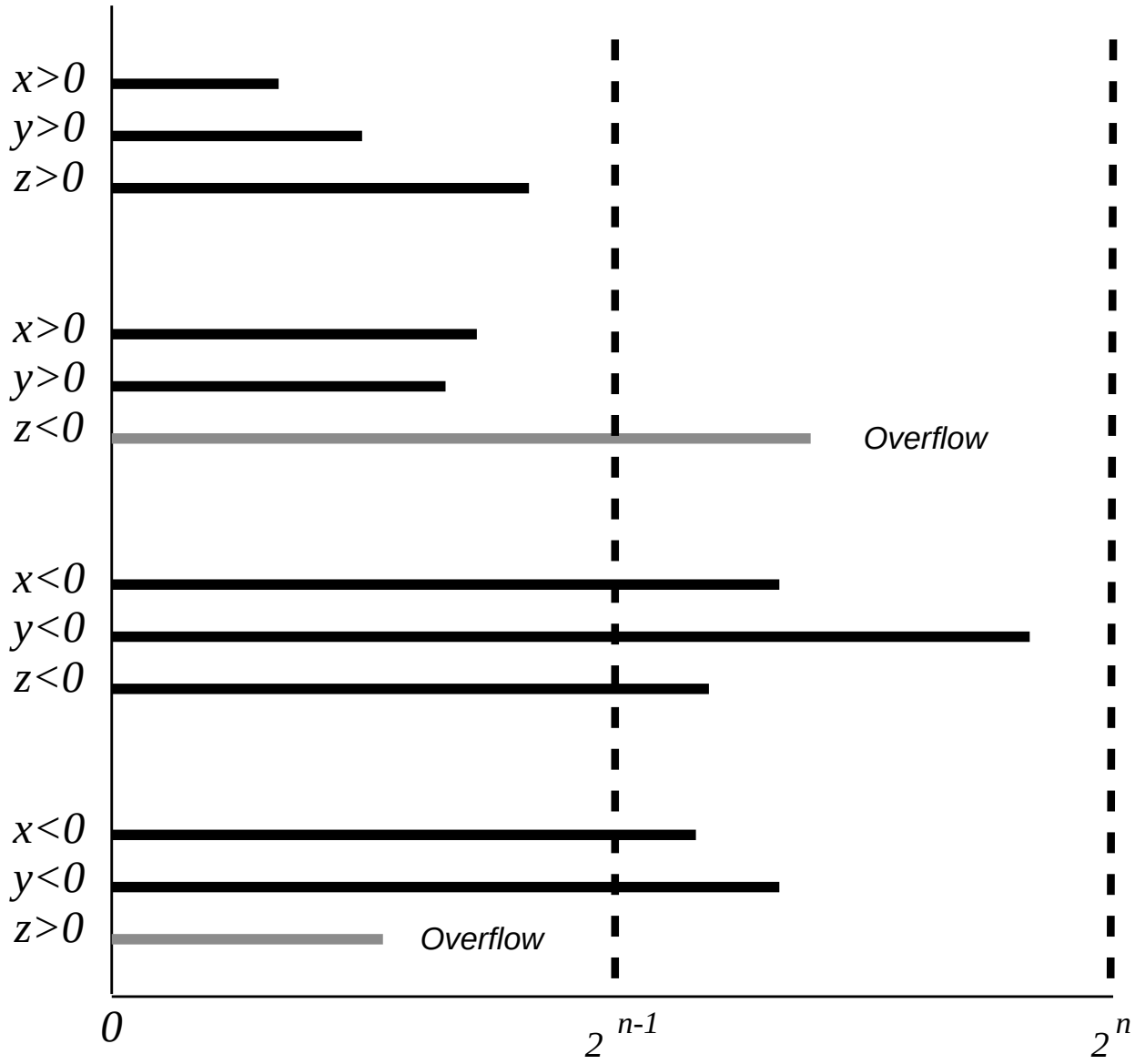


Figure 10.11: OVERFLOW IN TWO'S-COMPLEMENT SYSTEM.

CS M51A

OVERFLOW (ADDITION IN 2'S COMPLEMENT

$x \quad x_{n-1} \; x_{n-2} \; x_{n-3} \; \ldots \; x_0$

$y \quad y_{n-1} \; y_{n-2} \; y_{n-3} \; \ldots \; y_0$

$c_n \; c_{n-1} \; c_{n-2} \; c_{n-3} \; \ldots \; c_0$ CARRIES

$z \quad z_{n-1} \; z_{n-2} \; z_{n-3} \; \ldots \; z_0$

	x_{n-1}	y_{n-1}	c_{n-1}	c_n	z_{n-1}	
	0	0	0	0	0	
$x>0$ $y>0$	0	0	1	0	1	$z<0$ OVERFLOW
	0	1	0	0	1	
	0	1	1	1	0	
	1	0	0	0	1	
	1	0	1	1	0	
$x<0$ $y<0$	1	1	0	1	0	$z>0$ OVERFLOW
	1	1	1	1	1	

$\text{OVERFLOW} = c_n \oplus c_{n-1}$

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**

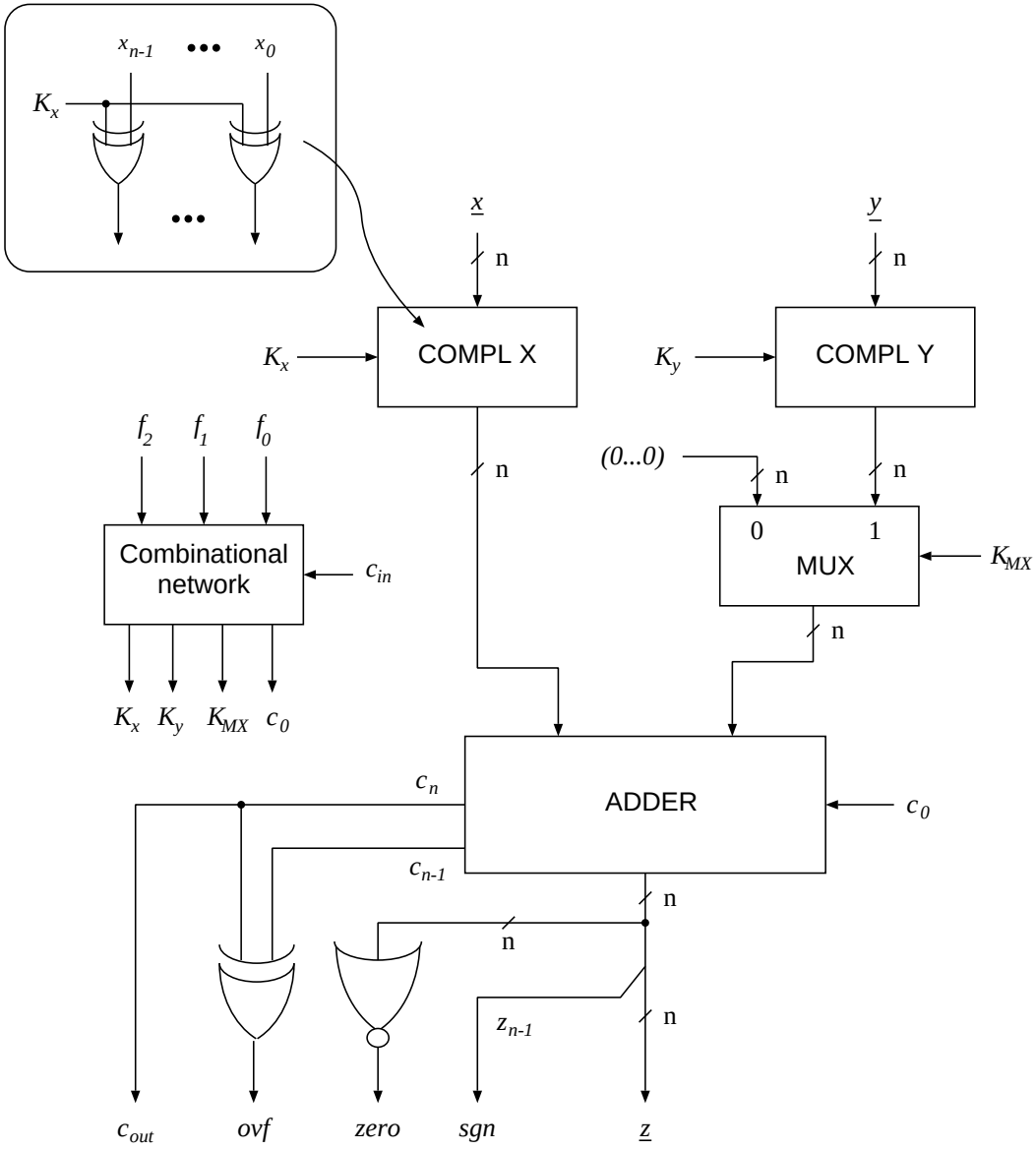


Figure 10.12: IMPLEMENTATION OF TWO'S-COMPLEMENT ARITHMETIC UNIT.

CONTROL OF TWO'S-COMPLEMENT ARITHMETIC OPERATIONS

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

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

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

cont.

- 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	$\underline{z} = \underline{x} + \underline{y}$
XOR	$\underline{z} = \underline{x} \oplus \underline{y}$
ONE	$\underline{z} = 1111$

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

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

4-bit ALU

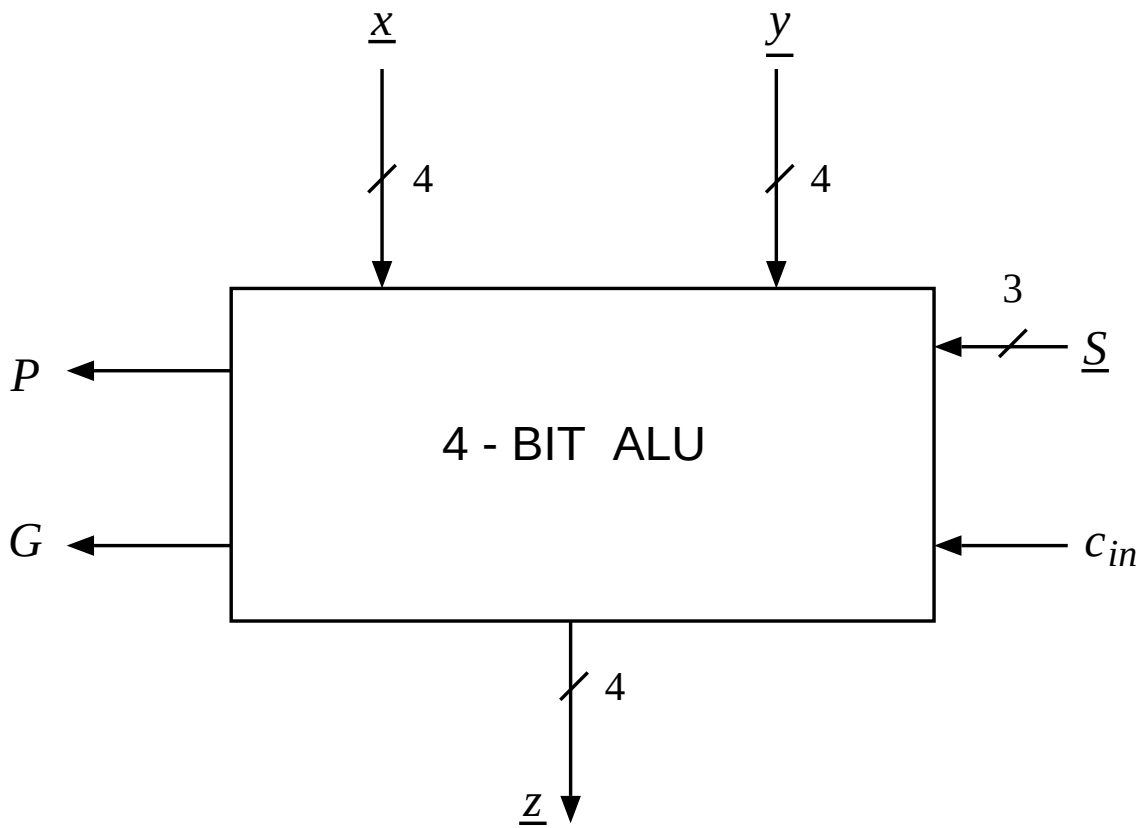


Figure 10.13: 4-bit ALU.

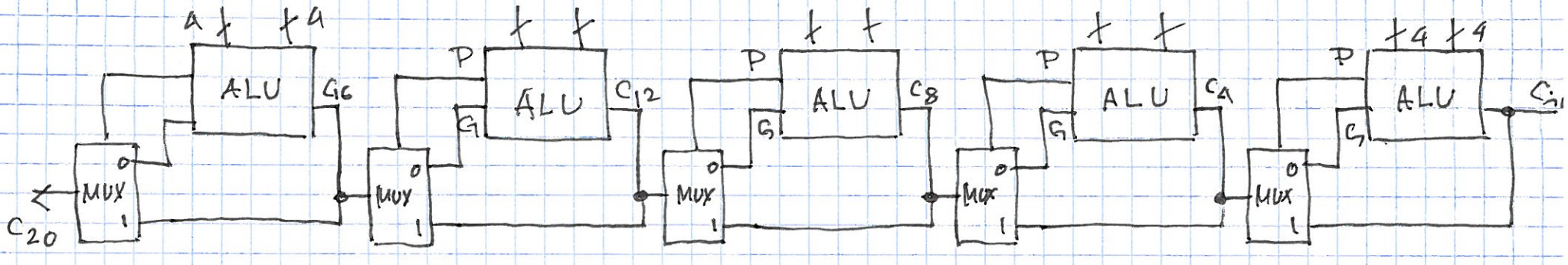
NETWORKS OF ALU MODULES

- 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*

CS MS1A CARRY-SKIP ADDER



ALU USES 4-BIT CARRY-RIPPLE ADDER

DETERMINE THE WORST CASE DELAY

16-bit ALU

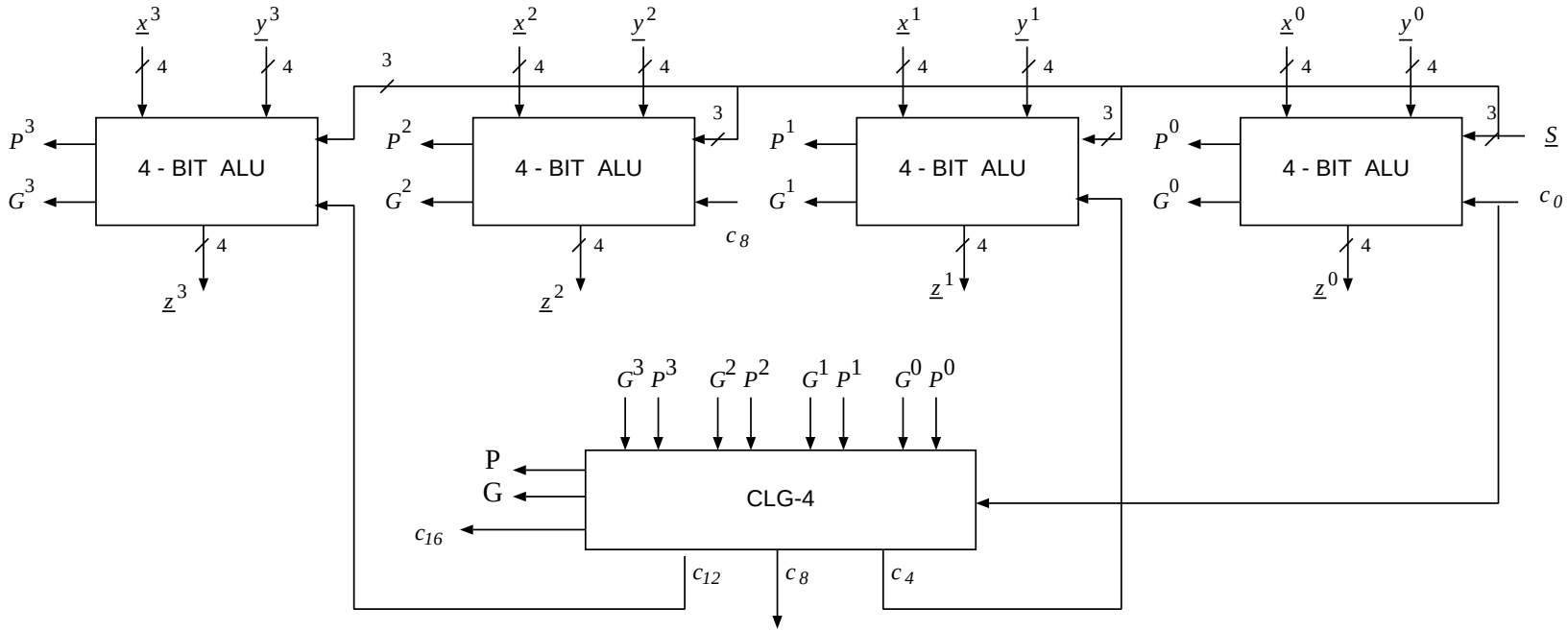


Figure 10.14: 16-bit ALU.

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 $\underline{y}$

cont.

- IMPLEMENTATION OF 4-bit COMPARATOR MODULE

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

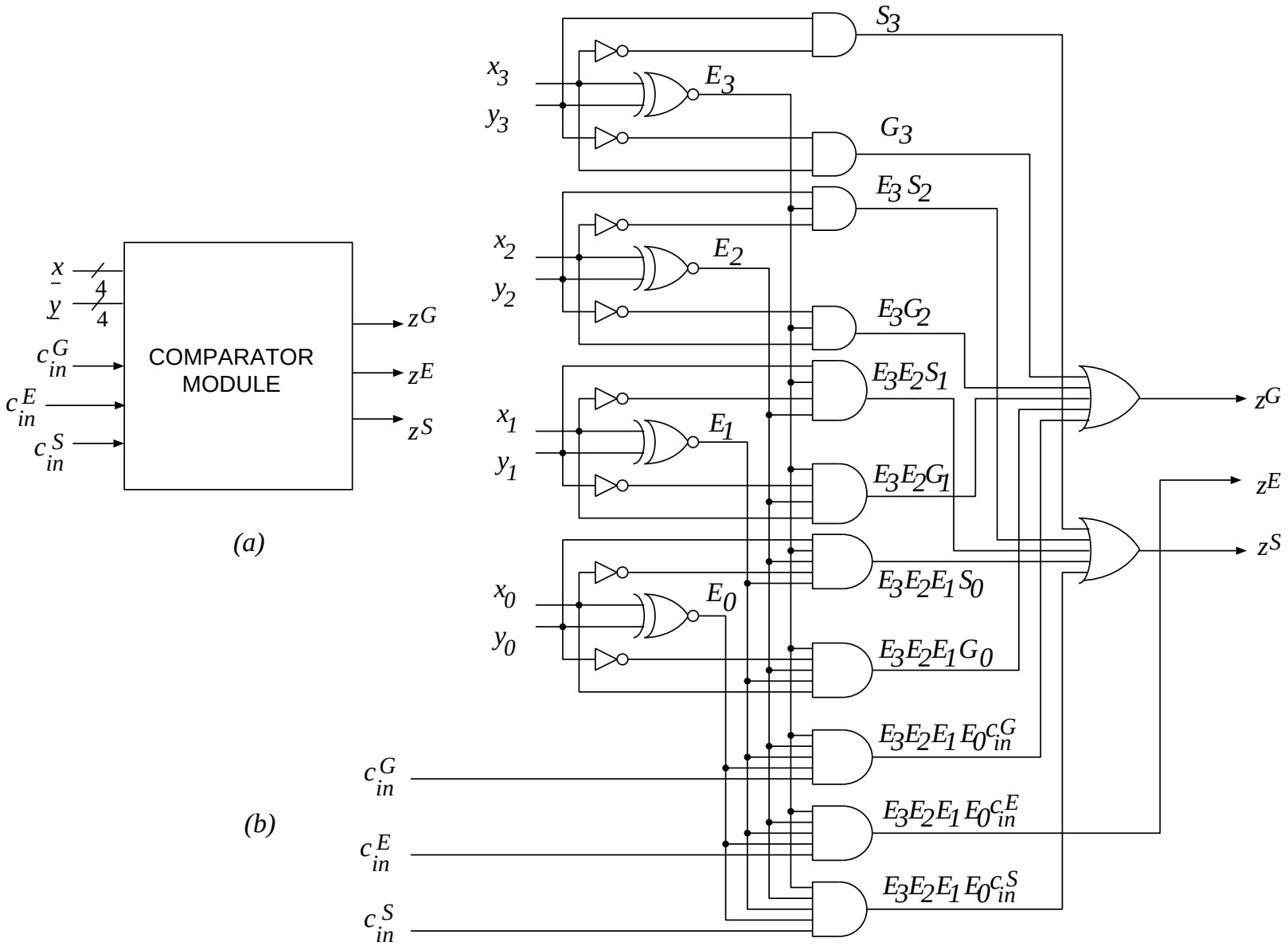


Figure 10.15: 4-BIT COMPARATOR MODULE: a) block diagram; b) gate-network implementation.

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

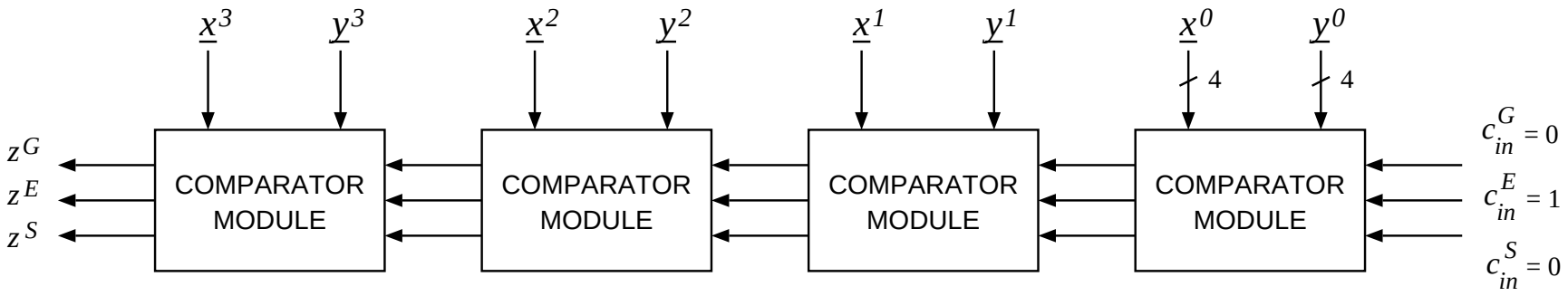
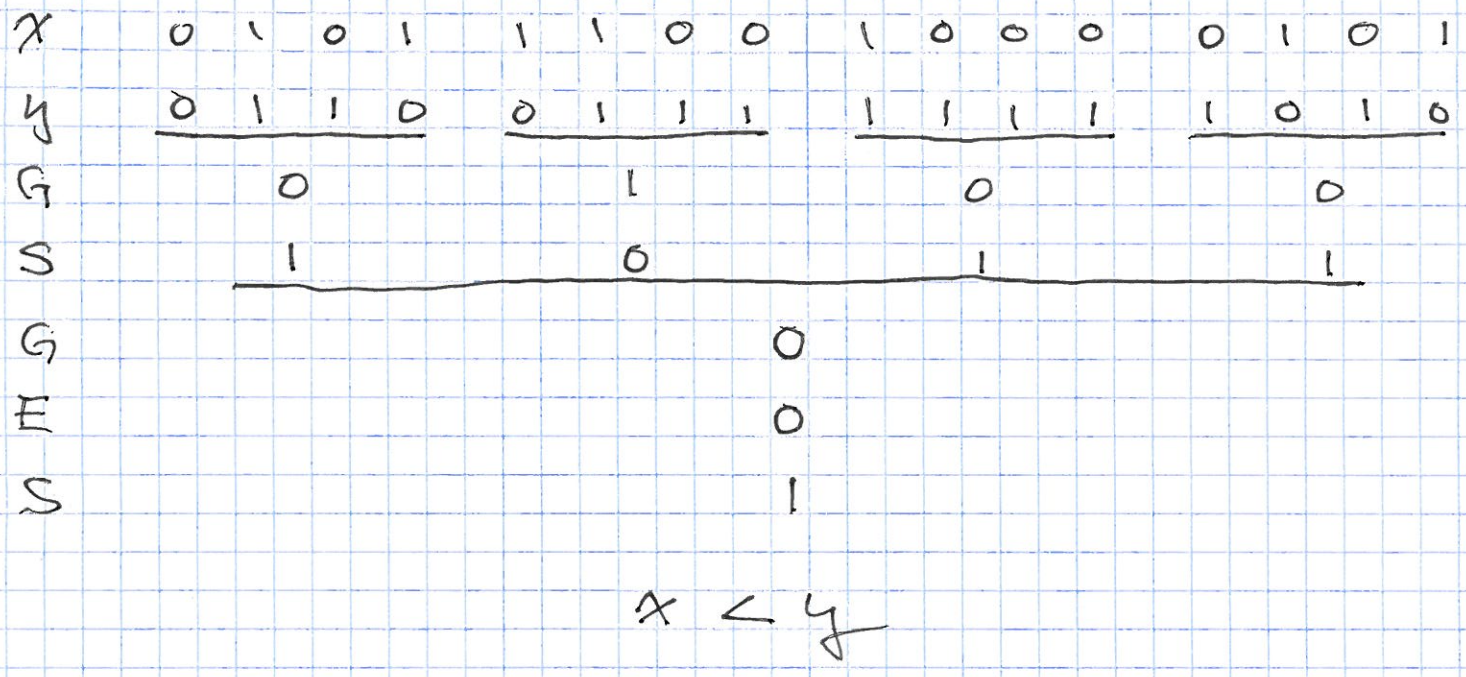


Figure 10.16: 16-BIT ITERATIVE COMPARATOR NETWORK.

EXAMPLE: COMPARATOR TREE



$X < Y$

2 LEVELS OF COMPARATORS INSTEAD OF 4 LEVELS

5 COMPARATORS VS. 4 COMPARATORS

TREE COMPARATOR NETWORK

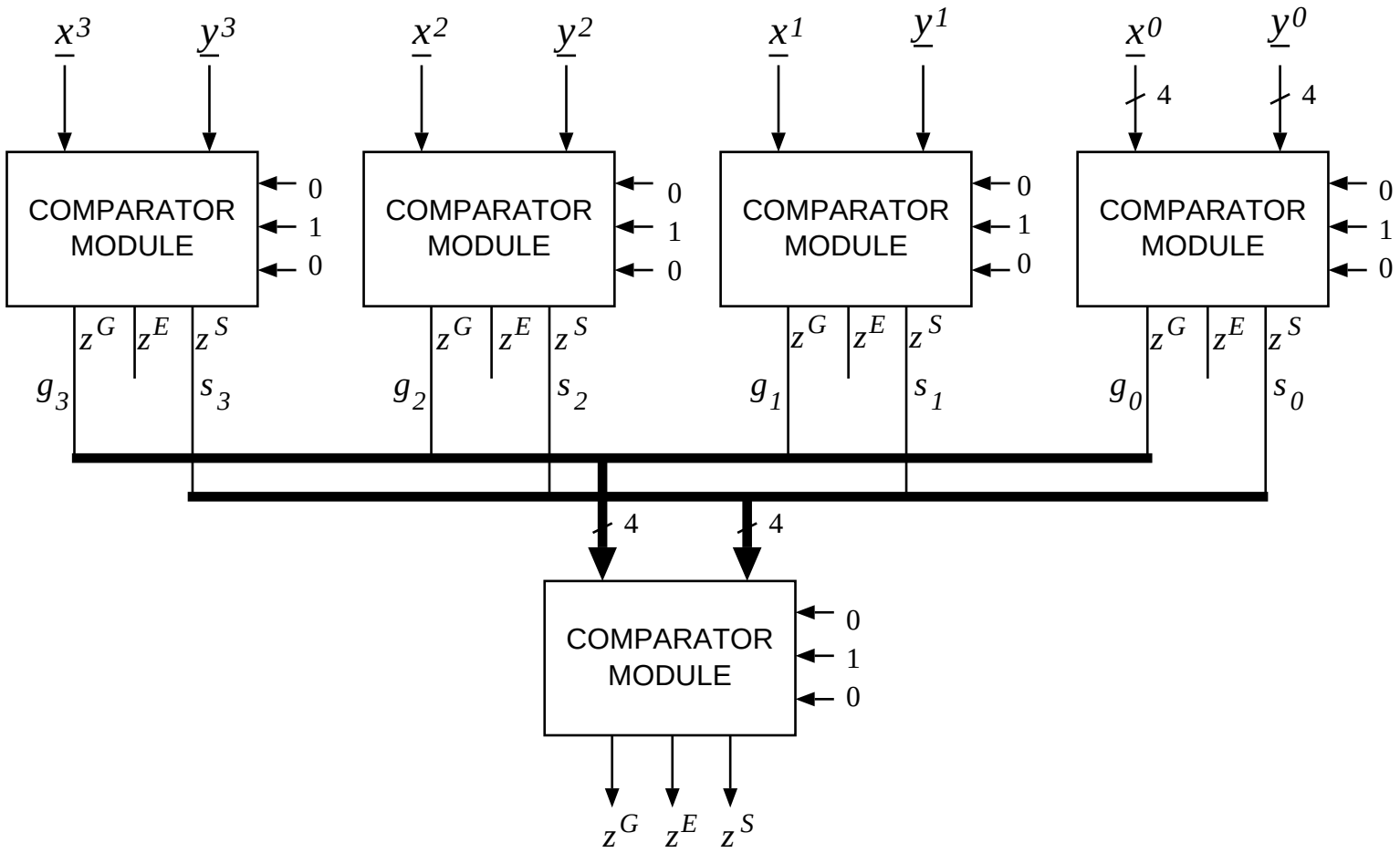


Figure 10.17: 16-BIT TREE COMPARATOR NETWORK.

TREE COMPARATOR (cont.)

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

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

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

- 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

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

CS M51A F19 MDE

MULTIPLIER DESIGN: AN EXAMPLE —

4-BIT × 3-BIT MULTIPLICATION

$p = x \cdot y \qquad x \in \{0, \ldots, 15\} \quad y \in \{0, \ldots 7\}$

$p \in \{0, \ldots, 105\}$

$x = (x_3, x_2, x_1, x_0) \qquad y = (y_2, y_1, y_0)$

$p = (p_6, p_5, \ldots, p_0)$

$$p = x \cdot y = x \sum (y_2 2^2 + y_1 2^1 + y_0 2^0)$$
$$= \sum (x \cdot y_2 \cdot 2^2 + x y_1 2^1 + x y_0 2^0)$$

$x = 1001 = 9 \qquad y = 110 = 6$

$$(1\ 0\ 0\ 1) \times 0 \cdot 2^0 \qquad \text{PARTIAL PRODUCTS } P_0$$
$$+ (1\ 0\ 0\ 1) \times 1 \cdot 2^1 \qquad P_1$$
$$+ (1\ 0\ 0\ 1) \times 1 \cdot 2^2 \qquad P_2$$

$\Downarrow$

$$\begin{array}{r} 0\ 0\ 0\ 0 \\ 1\ 0\ 0\ 1 \\ +\ 1\ 0\ 0\ 1 \\ \hline 1\ 1\ 0\ 1\ 1\ 0 \end{array} = 54$$

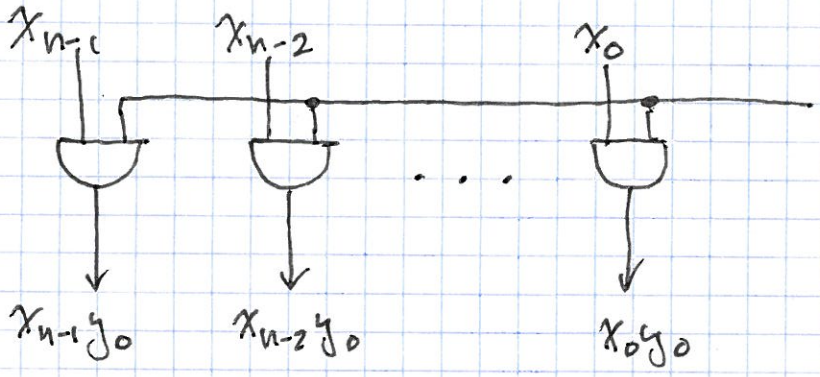
} ALIGNED BIT-MATRIX OF PARTIAL PRODUCTS

CS MSIA IMPLEMENTATION OF MULTIPLIERS

2 STAGES:

1. OBTAIN BIT-MATRIX

- SIMPLE IN RADIX-2 CASE



y_i: MULTIPLIER DIGIT

MULTIPLE (PARTIAL PRODUCT)

$n \times n$-BIT MULTIPLIER:

n^2 AND GATES

CS M51A

2, ADD ROWS TO OBTAIN THE PRODUCT

- TWO BASIC APPROACHES

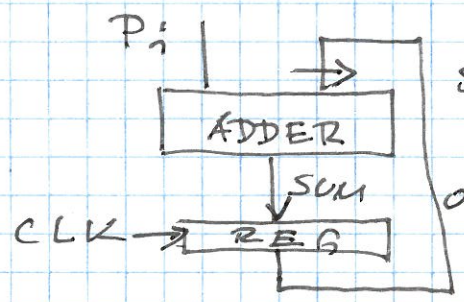
a) – SEQUENTIAL

b) – COMBINATIONAL

(*) LINEAR ARRAY OF ADDERS

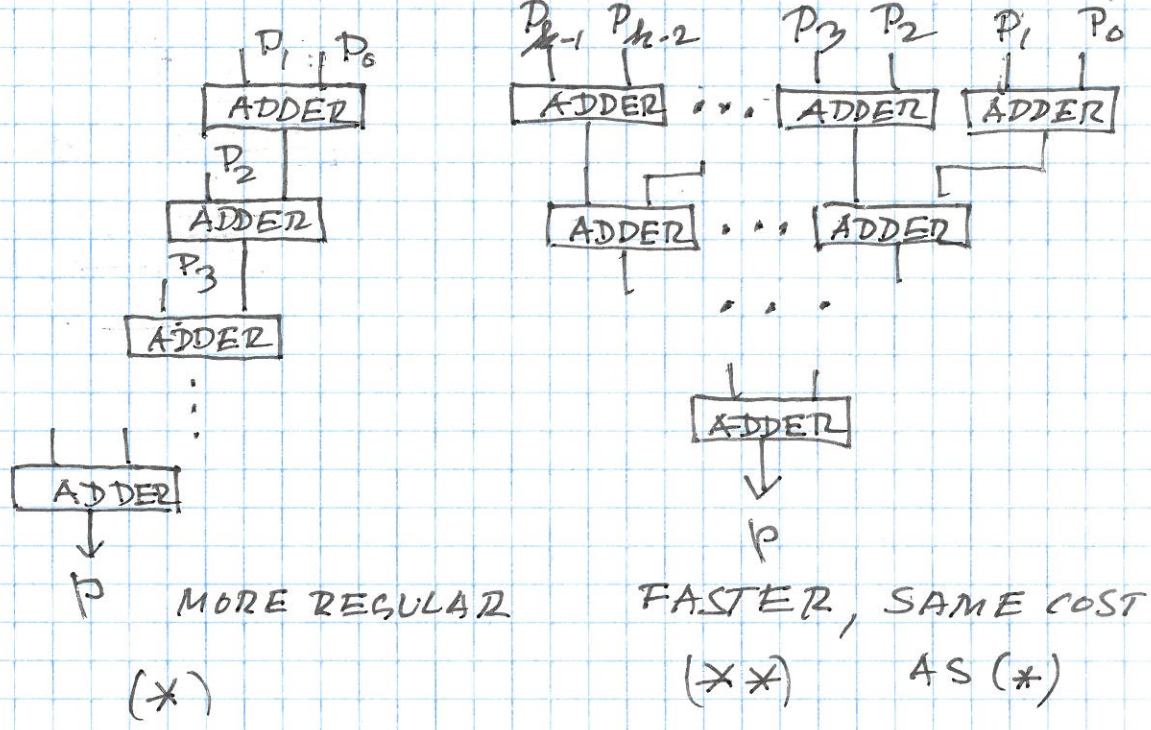
(**) TREE OF ADDERS

a)



SHIFT RIGHT TO ALIGN P_i AND THE SUM OF PARTIAL PRODUCTS

b)



MORE REGULAR

(*)

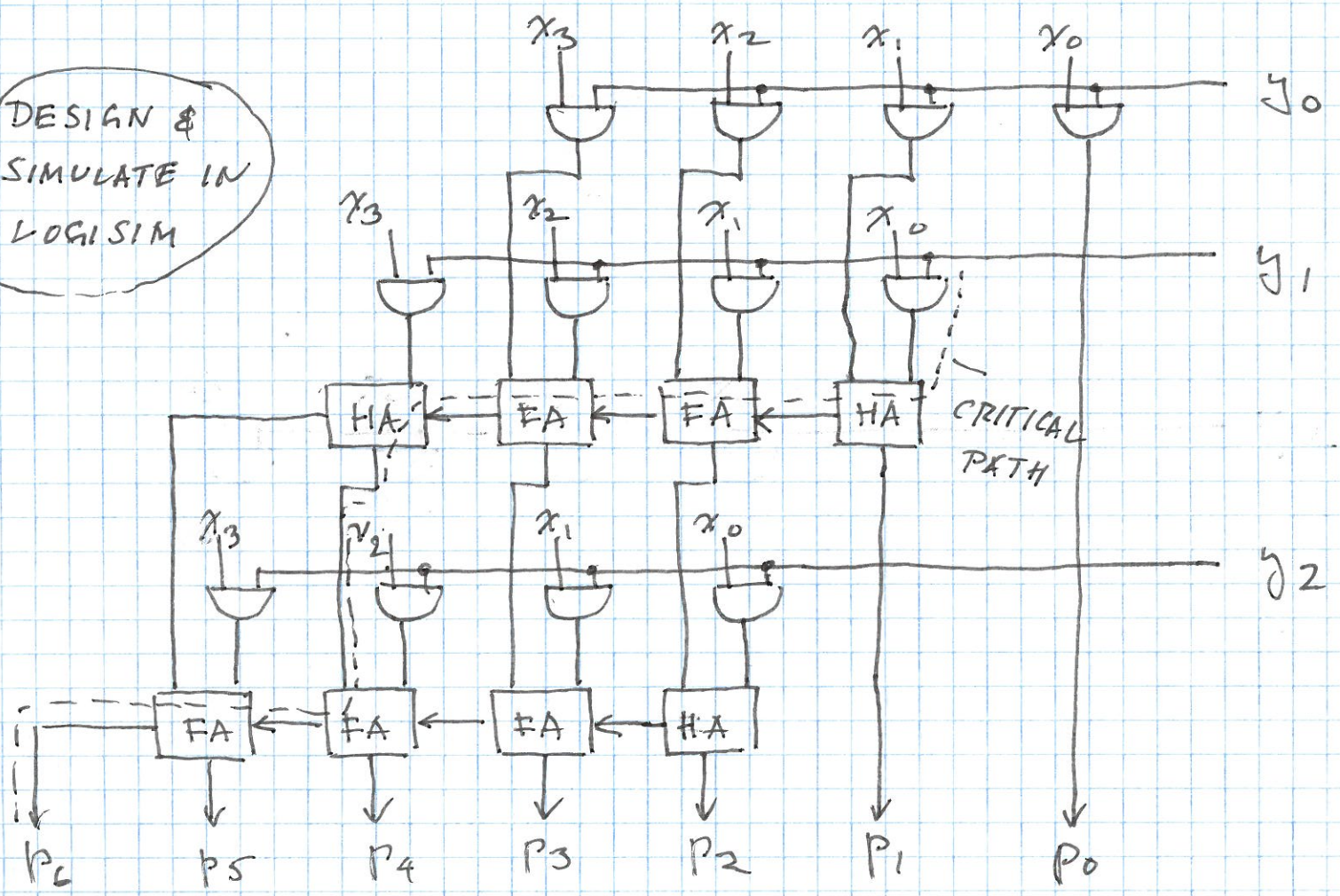
FASTER, SAME COST AS (*)

(**)

CS M51A

4 × 3 ARRAY MULTIPLIER

DESIGN & SIMULATE IN LOGISIM



FOR $n = m$

$T \rightarrow 3n.2t_g$

DELAY $T = t_{AND} + 3 \cdot t_c + t_s + 2t_c = 13 t_g \quad t_s = t_c = 2t_g$

IN GENERAL FOR $n \times m$ ARRAY MULTIPLIER

$$T = t_{AND} + (n-1)t_c + (t_s + t_c)(m-2)$$

$$\approx (n + 2m - 4)\, 2t_g$$

MULTIPLICATION BIT MATRIX

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

Multiplier implementation:

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

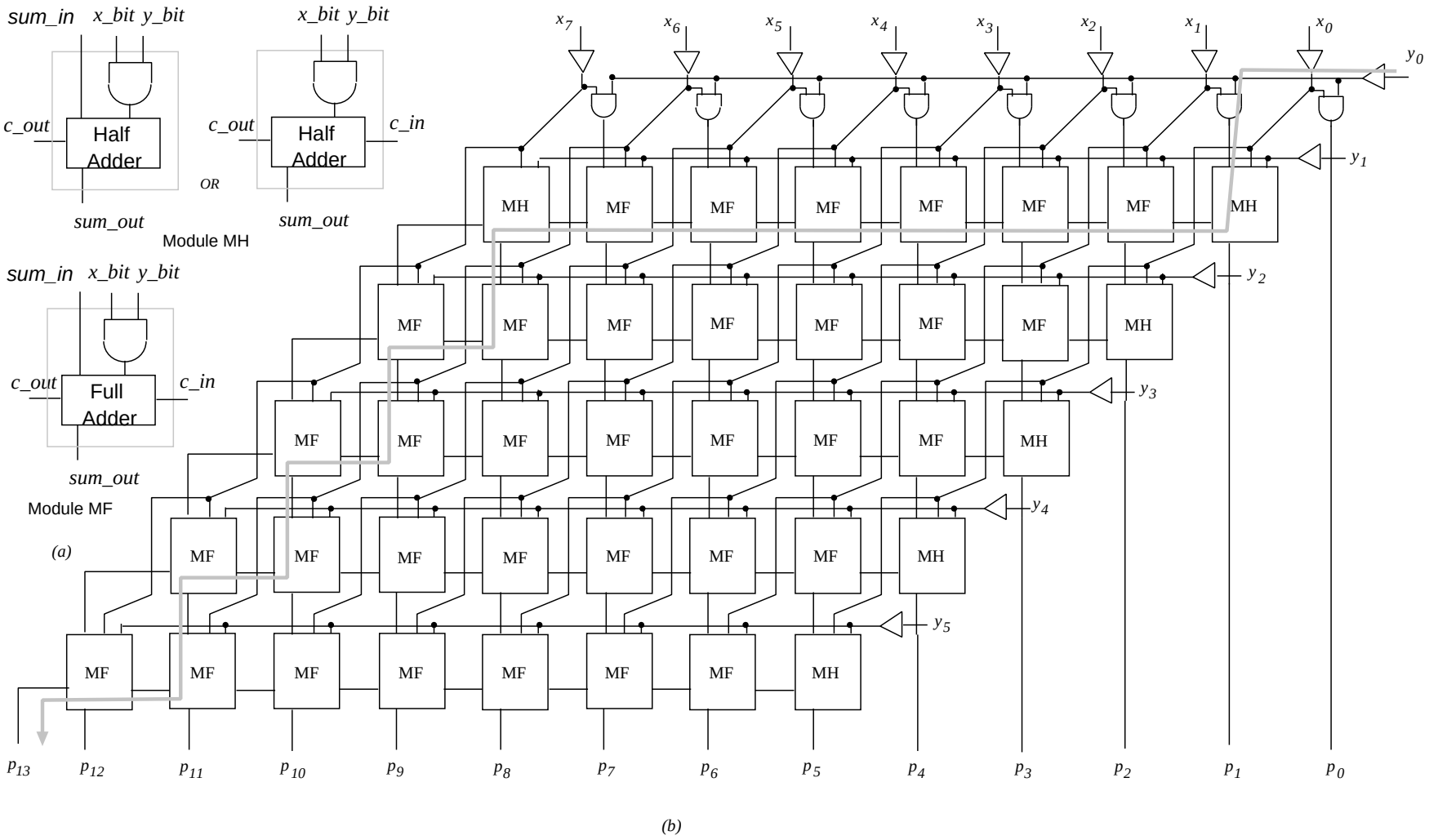


Figure 10.18: IMPLEMENTATION OF AN 8×6 MULTIPLIER: a) PRIMITIVE MODULES; b) NETWORK.

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:

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

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

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

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.

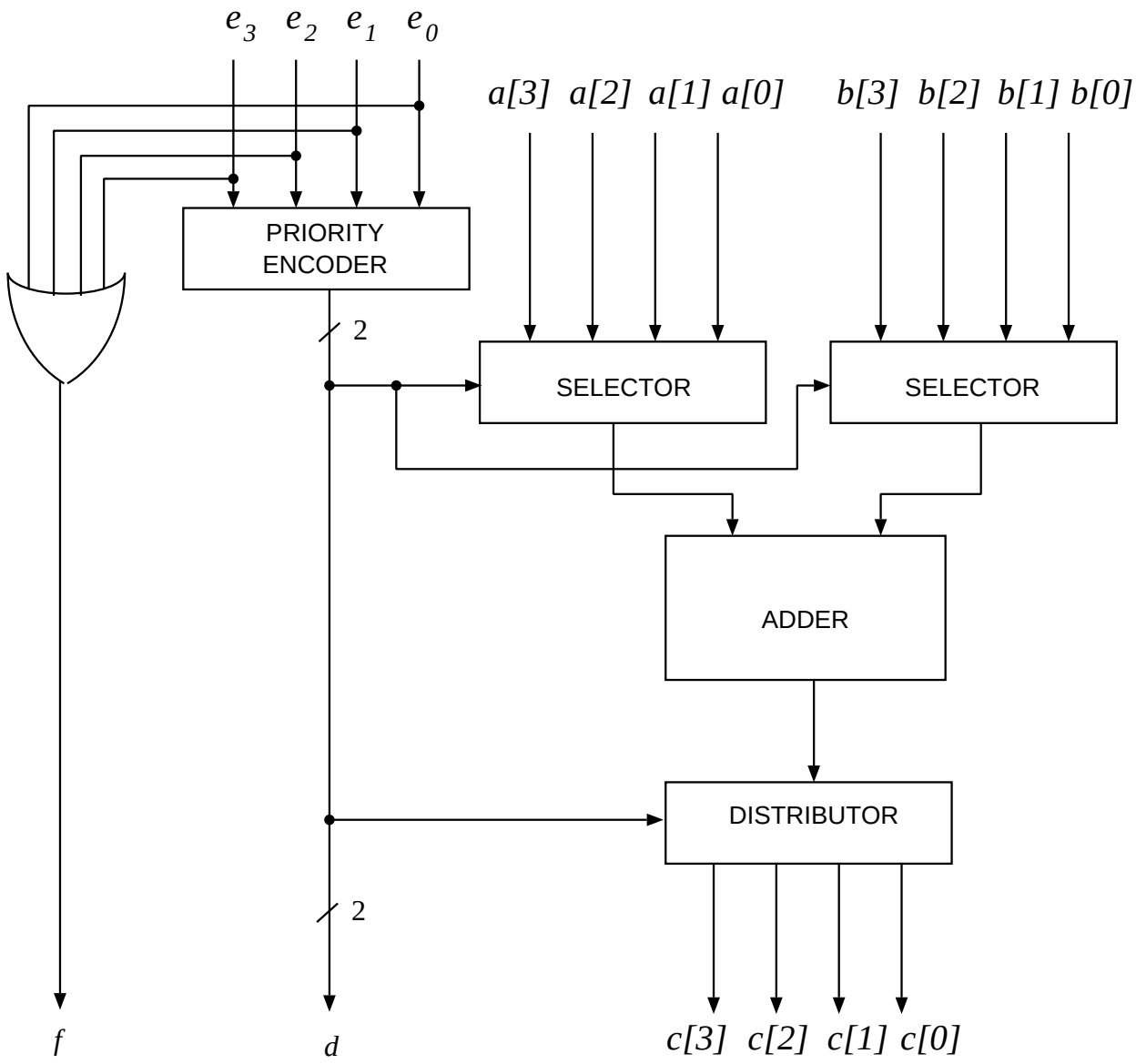


Figure 10.19: NETWORK IN EXAMPLE 10.5