FAST TWO-OPERAND ADDITION

A. Conventional number system.

Carry-propagate adders (CPA)

- Switched carry-ripple adder
- Carry-skip adder
- Carry-lookahead adder
- Prefix adder
- Carry-select adder and conditional-sum adder
- Variable-time adder

B. Redundant number system.

Totally-parallel adders (TPA); adders with limited carry propagation

- Carry-save adder
- Signed-digit adder

n-BIT ADDITION

$$x + y + c_{in} = 2^n c_{out} + s$$

The solution:

$$\begin{aligned} s &= (x + y + c_{in}) \bmod 2^n \\ c_{out} &= \begin{cases} 1 & \text{if } (x + y + c_{in}) \geq 2^n \\ 0 & \text{otherwise} \end{cases} \\ &= \lfloor (x + y + c_{in})/2^n \rfloor \end{aligned}$$

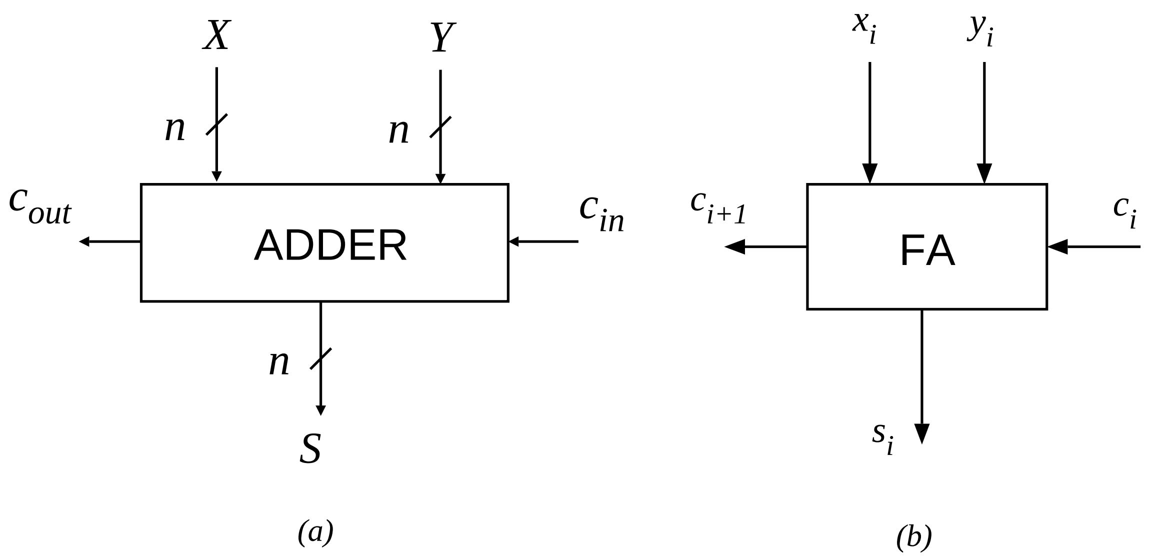


Figure 2.1: (a) An n-bit adder. (b) 1-bit adder (full adder module).

1-BIT ADDITION

- Primitive module *full adder* (FA)

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

with solution

$$s_i = (x_i + y_i + c_i) \bmod 2$$
$$c_{i+1} = \lfloor (x_i + y_i + c_i)/2 \rfloor$$

ADDITION: TWO-STEP PROCESS

1. Obtain carries (carry at i depends on $j \leq i$)
   – non-trivial to do fast
2. Compute sum bits (local function)

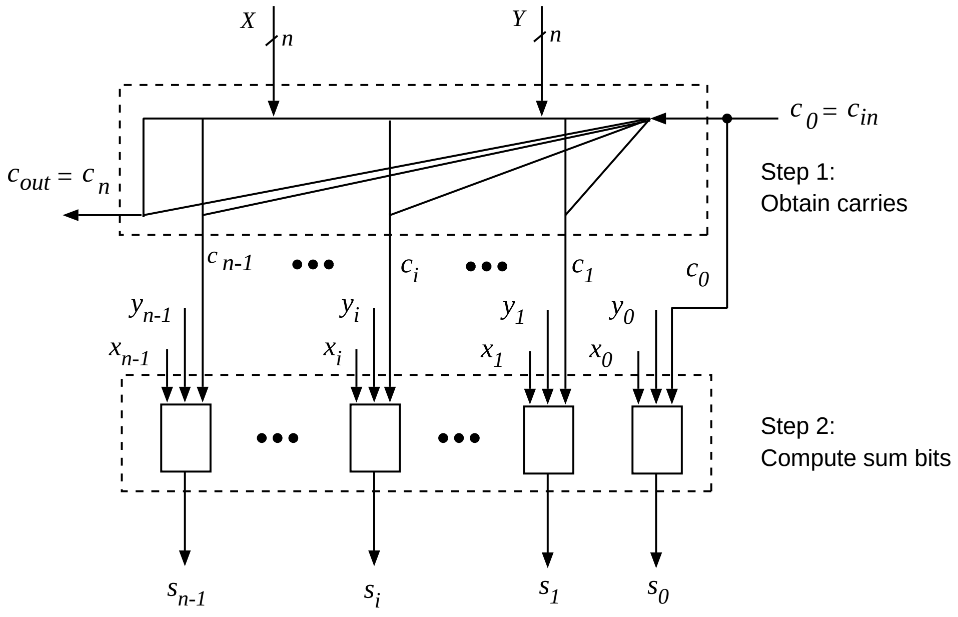


Figure 2.2: Steps in addition.

CARRY-OUT CASES

Case	x_i	y_i	$x_i + y_i$	c_{i+1}	Comment
1	0	0	0	0	kill (stop) carry-in
2	0	1	1	c_i	propagate carry-in
	1	0	1	c_i	propagate carry-in
3	1	1	2	1	generate carry-out

Case 1 (Kill): $k_i = x_i' y_i' = (x_i + y_i)'$
Case 2 (Propagate): $p_i = x_i \oplus y_i$
Case 3 (Generate): $g_i = x_i y_i$
Then

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

Alternative (simpler) expression:

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

Since $a_i = k_i'$ we call it "alive"

CARRY CHAINS

Two types:

1-carry chain consisting of carry=1
0-carry chain consisting of carry=0

i	9		8	7		6		5		4	3		2		1		0
x_i	1		0	1		0		1		1	1		1		0		0
y_i	0		0	0		1		0		1	0		0		1		0
	p		k	p		p		p		g	p		p		p		k
	a			a		a		a		a	a		a		a		
c_{i+1}	0	$\leftarrow$	0	1	$\leftarrow$	1	$\leftarrow$	1	$\leftarrow$	1	0	$\leftarrow$	0	$\leftarrow$	0	$\leftarrow$	0

Generalization to group of bits

$$c_{j+1} = g_{(j,i)} + p_{(j,i)} c_i = g_{(j,i)} + a_{(j,i)} c_i$$

or, for $i = 0$

$$c_{j+1} = g_{(j,0)} + p_{(j,0)} c_0 = g_{(j,0)} + a_{(j,0)} c_0$$

Recursive combining of subranges of variables:

$$\begin{aligned}
g_{(f,d)} &= g_{(f,e)} + p_{(f,e)} g_{(e-1,d)} = g_{(f,e)} + a_{(f,e)} g_{(e-1,d)} \\
a_{(f,d)} &= a_{(f,e)} a_{(e-1,d)} \\
p_{(f,d)} &= p_{(f,e)} p_{(e-1,d)}
\end{aligned}$$

Generalization (cont.)

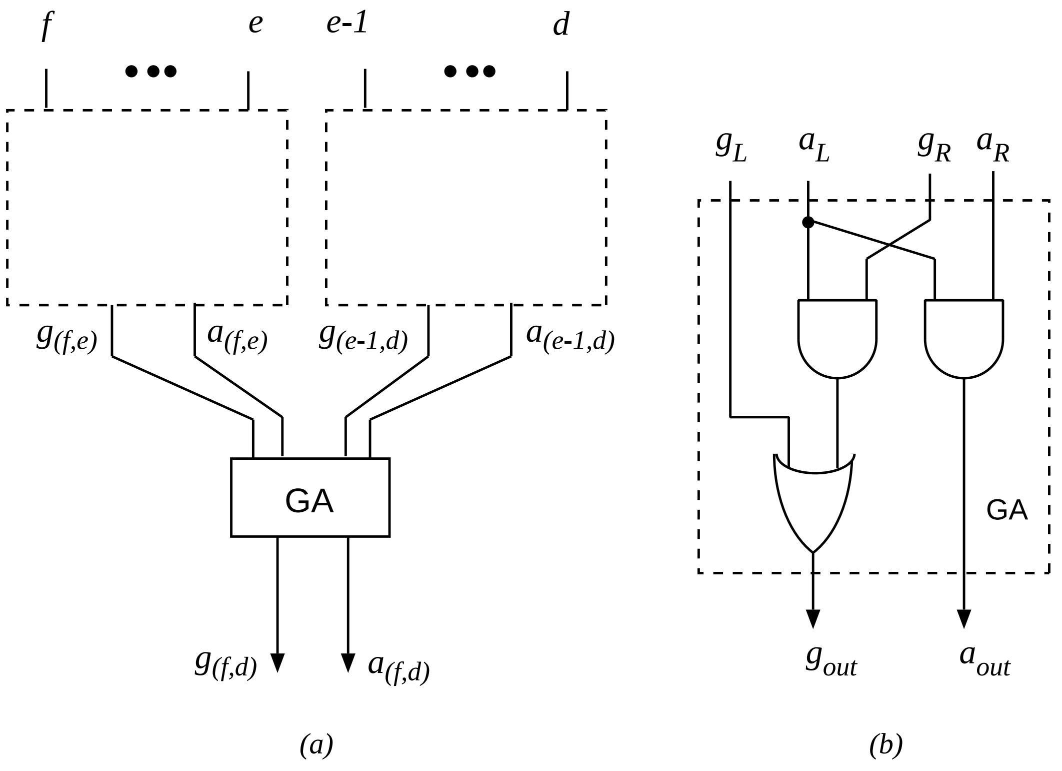


Figure 2.3: Computing $(g_{(f,d)}, a_{(f,d)})$.

BASIC CARRY-RIPPLE ADDER (CRA)

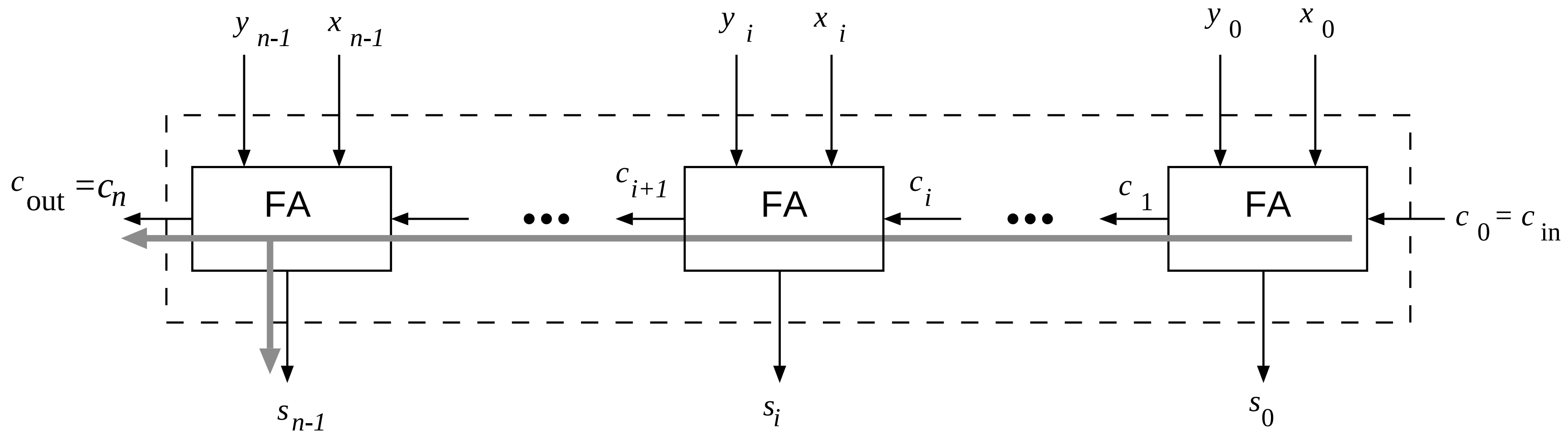


Figure 2.4: Carry-ripple adder.

$$T_{CRA} = (n-1)t_c + \max(t_c, t_s)$$

Implementations of full-adder

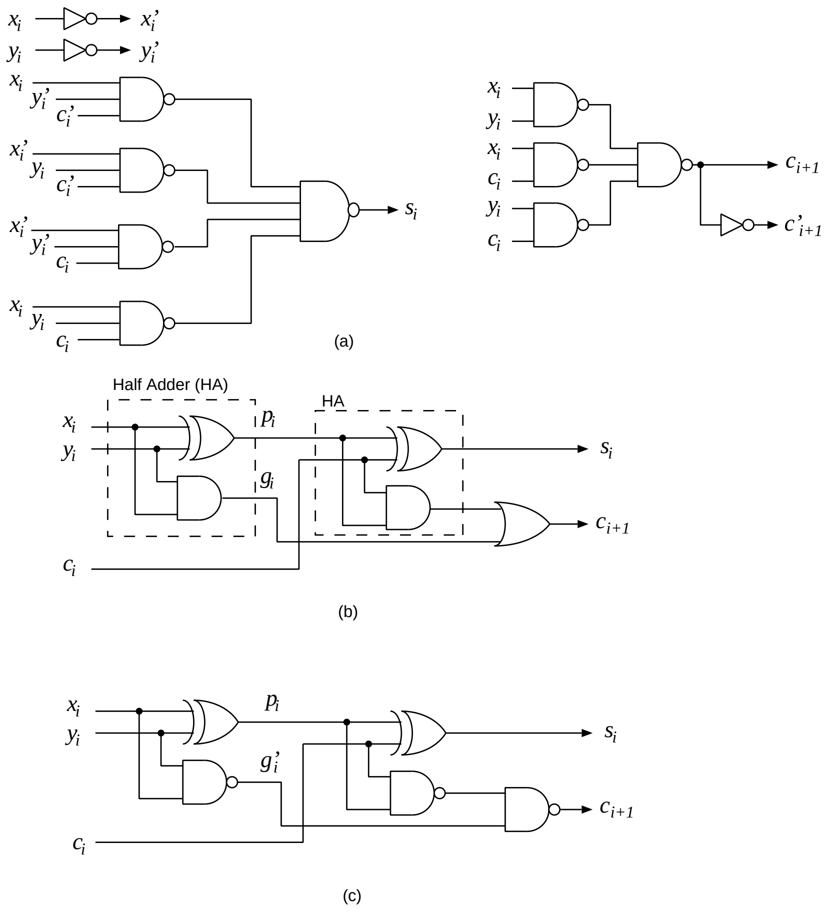


Figure 2.5: Implementation of full-adder. (a) Two-level network. (b) Multilevel network with XOR, AND and OR gates; (c) Multilevel implementation with XOR and NAND gates.

SWITCHED CARRY-RIPPLE (Manchester) ADDER

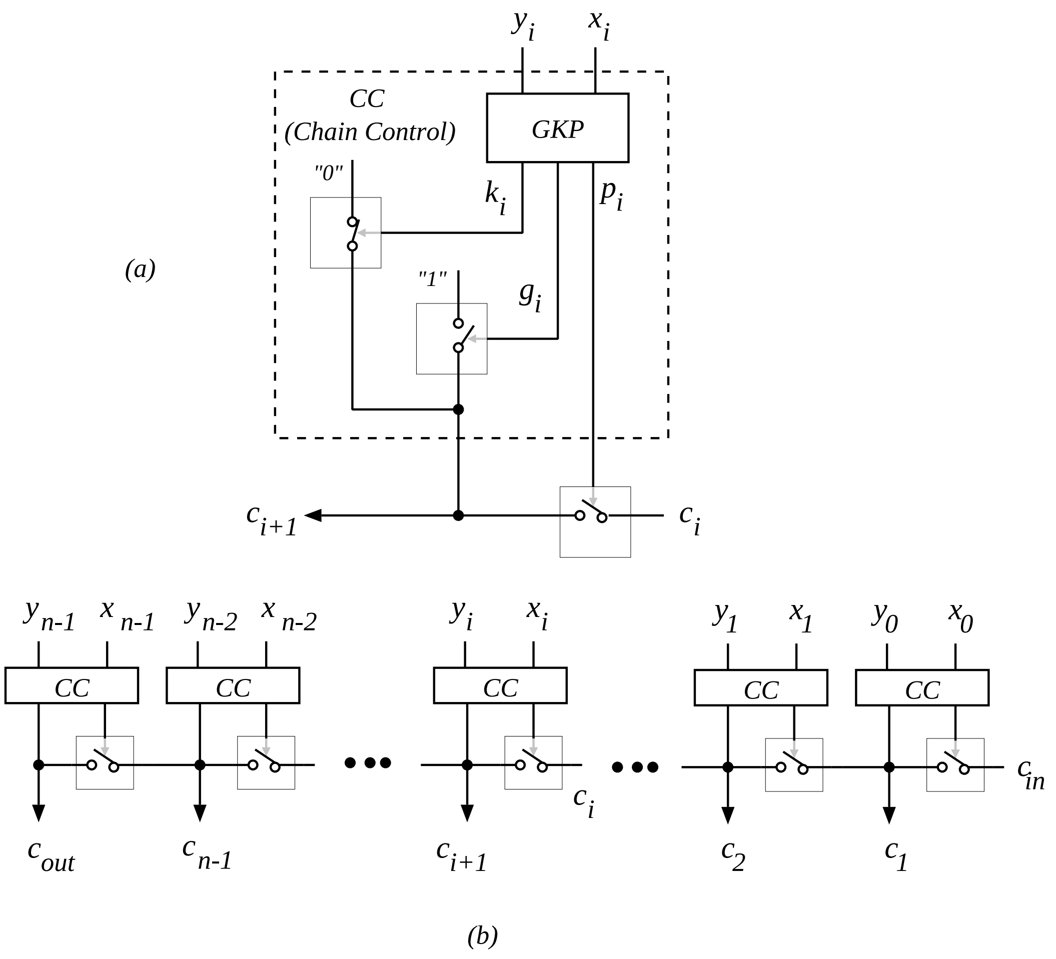
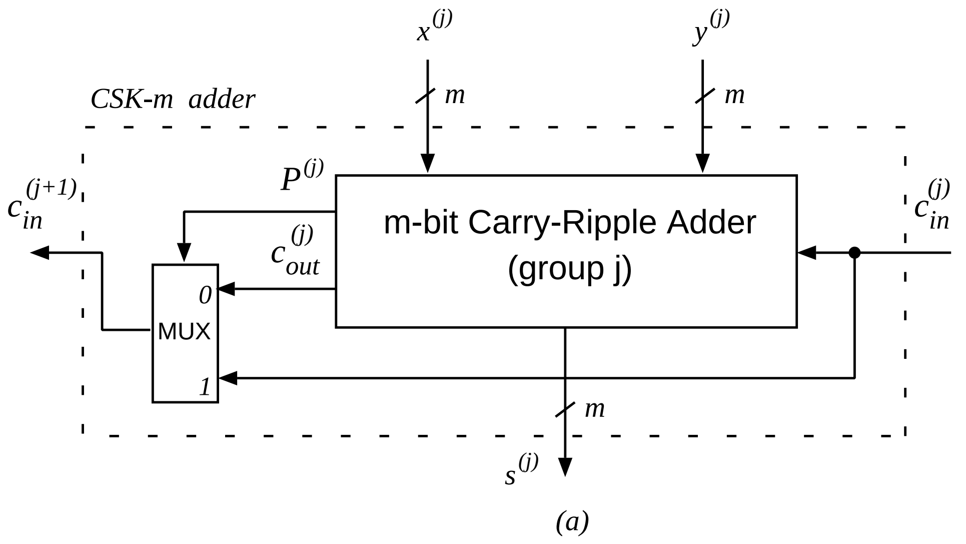


Figure 2.6: Switch carry-ripple network (Manchester circuit)

CARRY-SKIP ADDER



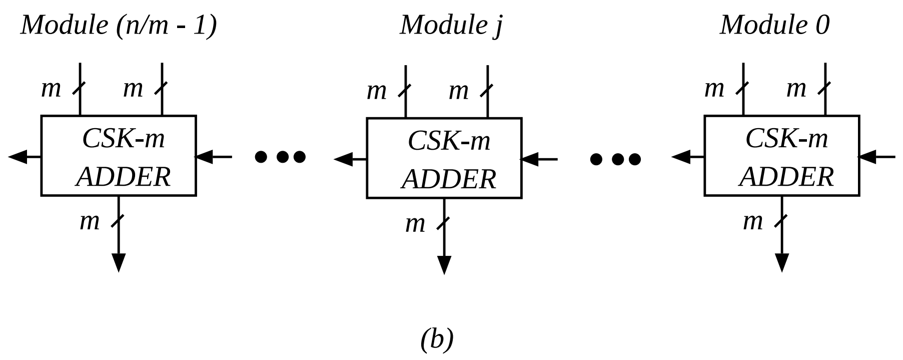


Figure 2.7: Carry-skip adder: (a) A group with carry bypass. (b) n-bit CSK adder.

CARRY CHAINS IN CARRY-SKIP ADDER

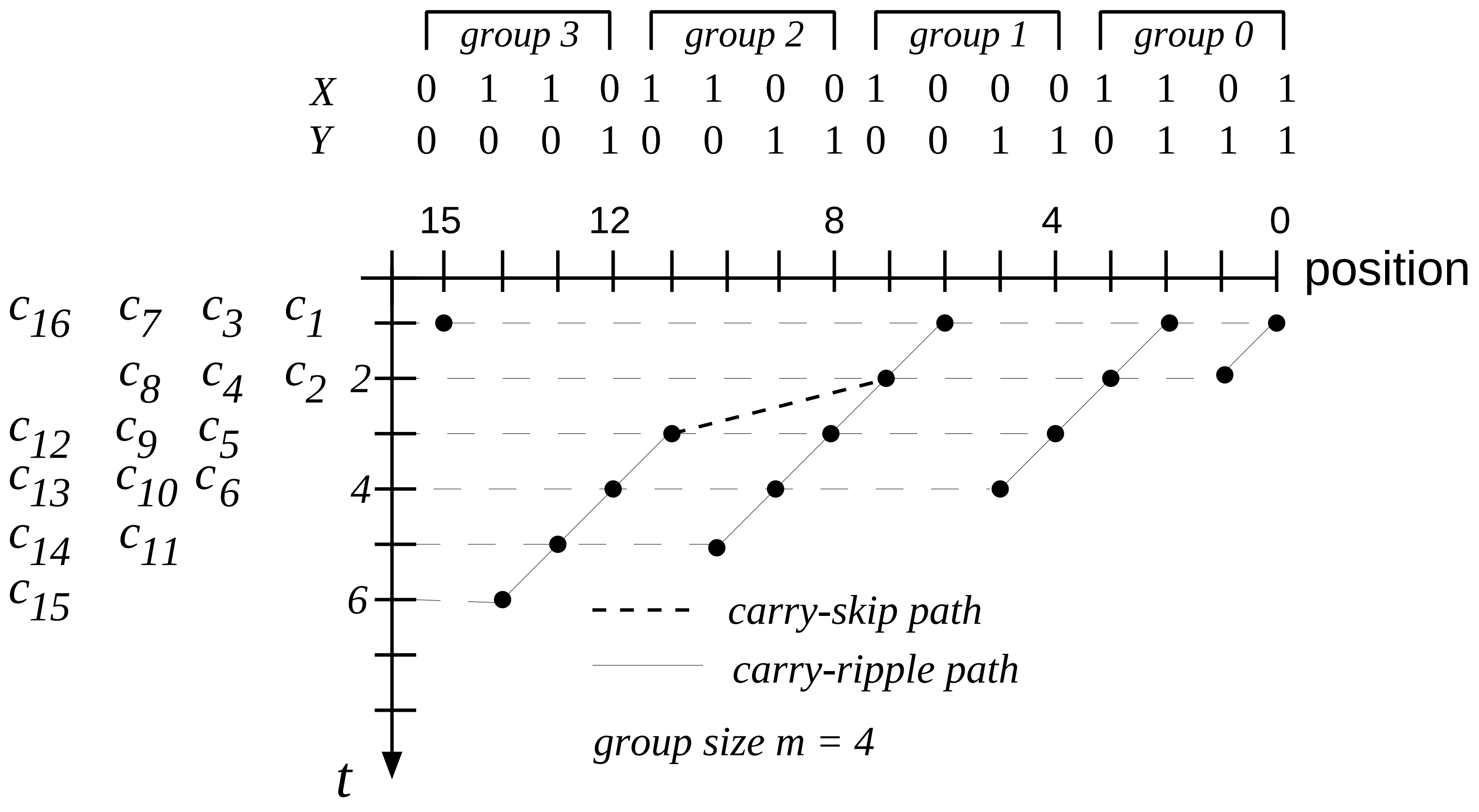
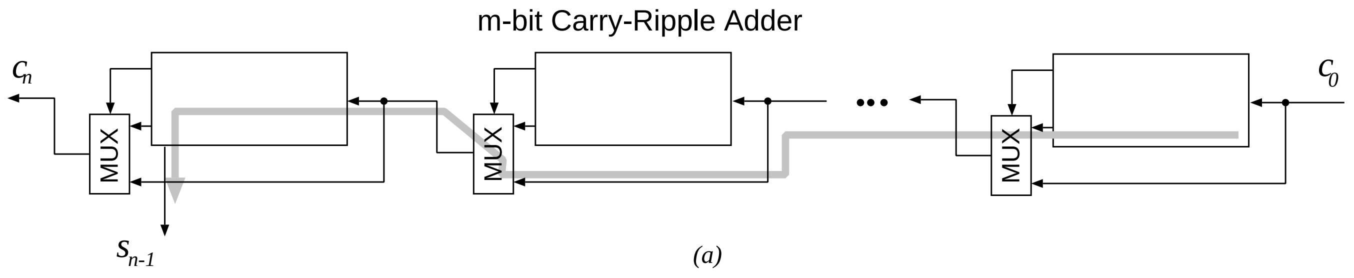


Figure 2.8: Carry chains in carry-skip adder: A case with several carry chains.



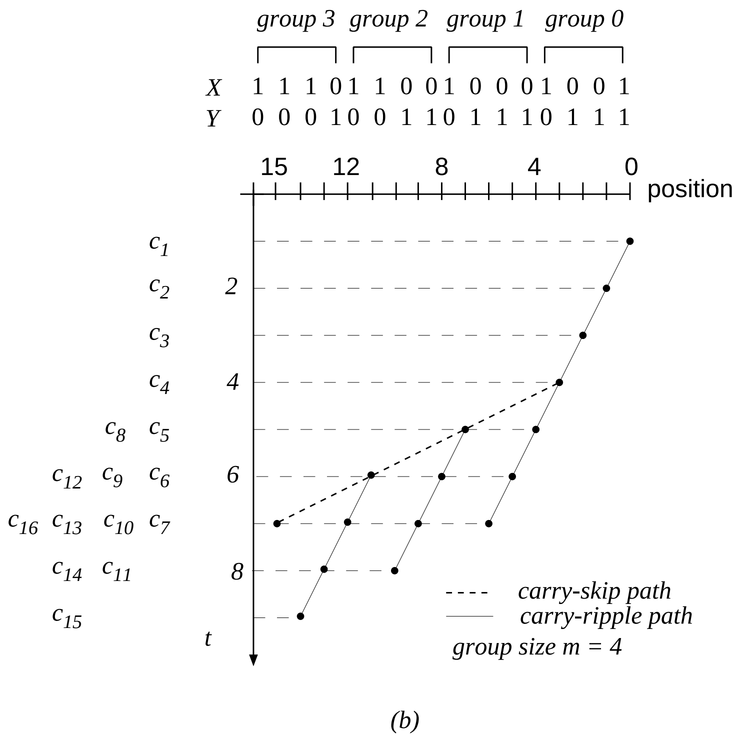

group 3 group 2 group 1 group 0
X 1 1 1 0 1 1 0 0 1 0 0 0 1 0 0 1
Y 0 0 0 1 0 0 1 1 0 1 1 1 0 1 1 1
15 12 8 4 0
position
t
2
4
6
8
c_1 c_2 c_3 c_4 c_5 c_6 c_7 c_8 c_9 c_{10} c_{11} c_{12} c_{13} c_{14} c_{15} c_{16}
carry-skip path
carry-ripple path
group size m = 4
(b)

Figure 2.9: (a) Critical path in carry-skip adder. (b) The worst-case situation for $n = 16$.

WORST-CASE DELAY

$$\begin{aligned} T_{CSK} &= mt_c + t_{mux} + (\frac{n}{m} - 2)t_{mux} + (m-1)t_c + t_s \\ &= (2m-1)t_c + (\frac{n}{m} - 1)t_{mux} + t_s \end{aligned}$$

PROBLEM WITH CLEARING OF CARRIES

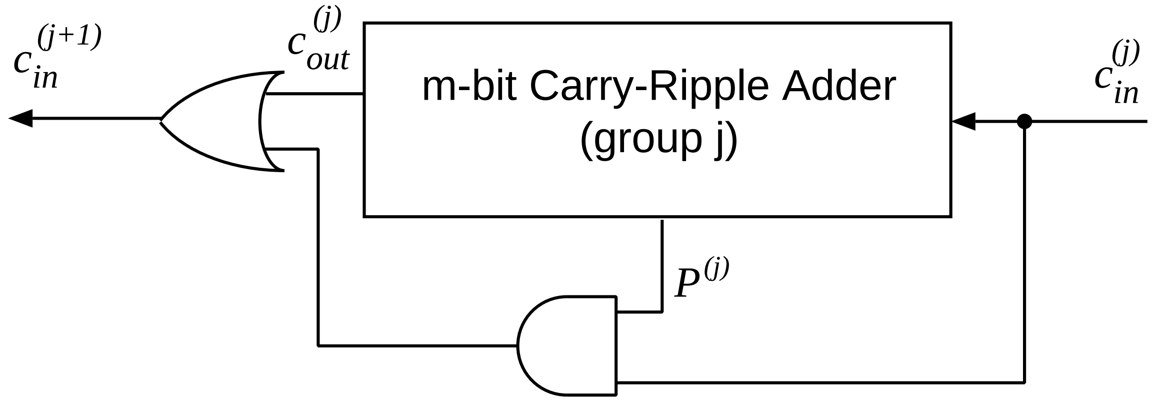


Figure 2.10: Carry-skip adder using AND-OR for bypass

GROUP SIZE IN CARRY-SKIP ADDERS

Fixed-size:

$$\begin{aligned} m_{opt} &= \left(\frac{t_{mux}}{2t_c} n\right)^{1/2} \quad \text{(minimum delay)} \\ T_{opt} &\approx (8 t_{mux} t_c n)^{1/2} \end{aligned}$$

Variable-size:

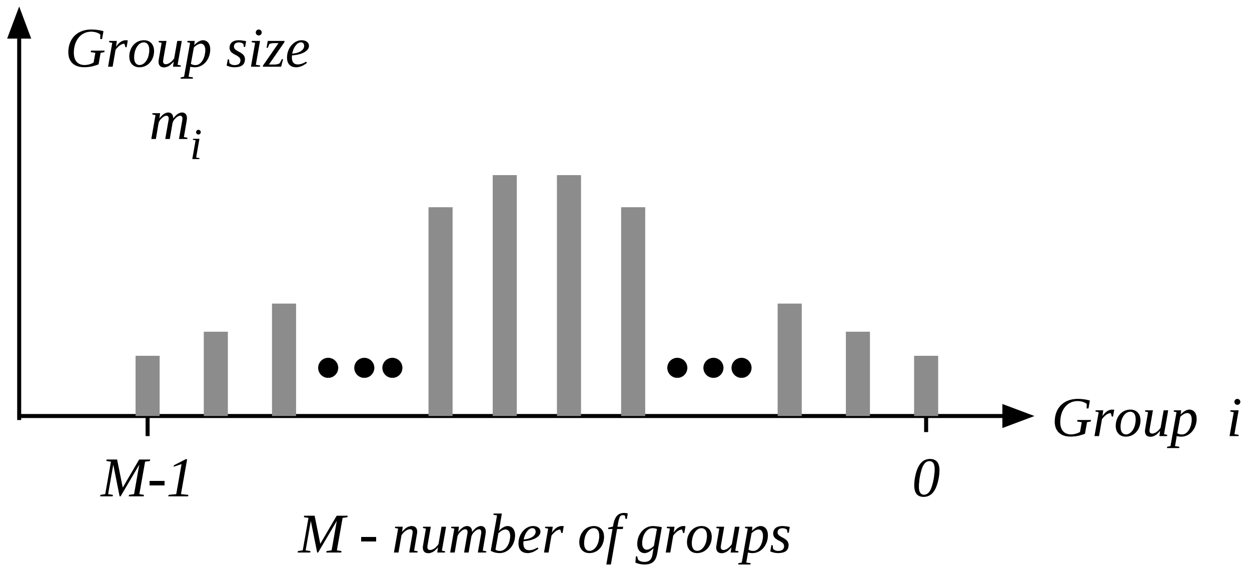


Figure 2.11: Optimal distribution of group sizes in carry-skip adder.

CARRY-LOOKAHEAD ADDER(CLA)

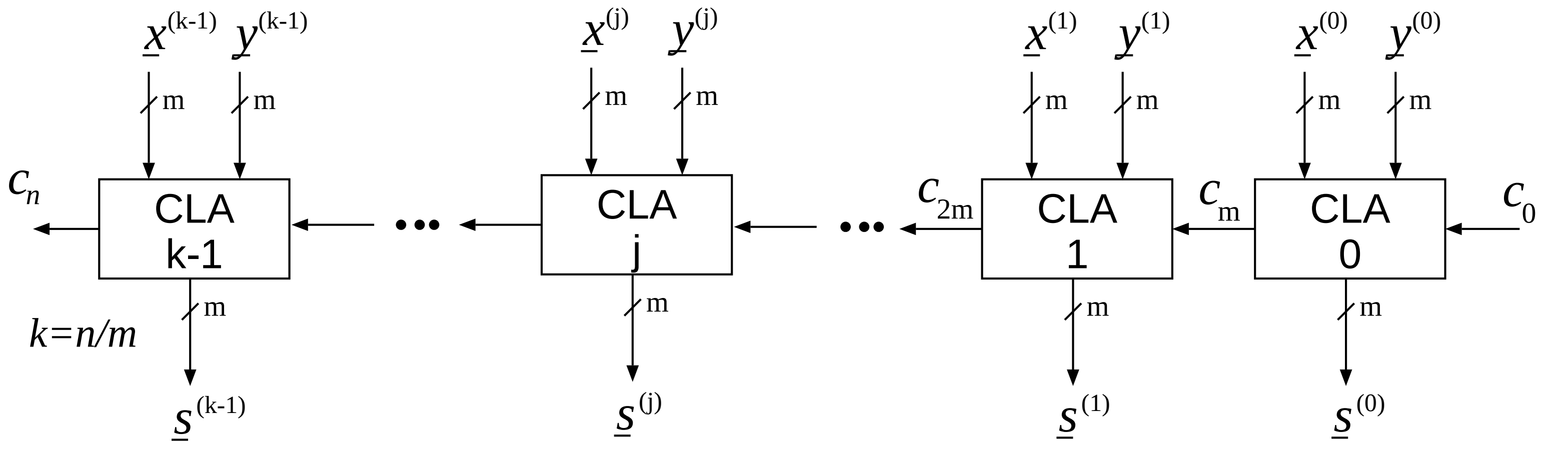


Figure 2.12: One-level carry-lookahead adder

CARRY-LOOKAHEAD MODULE

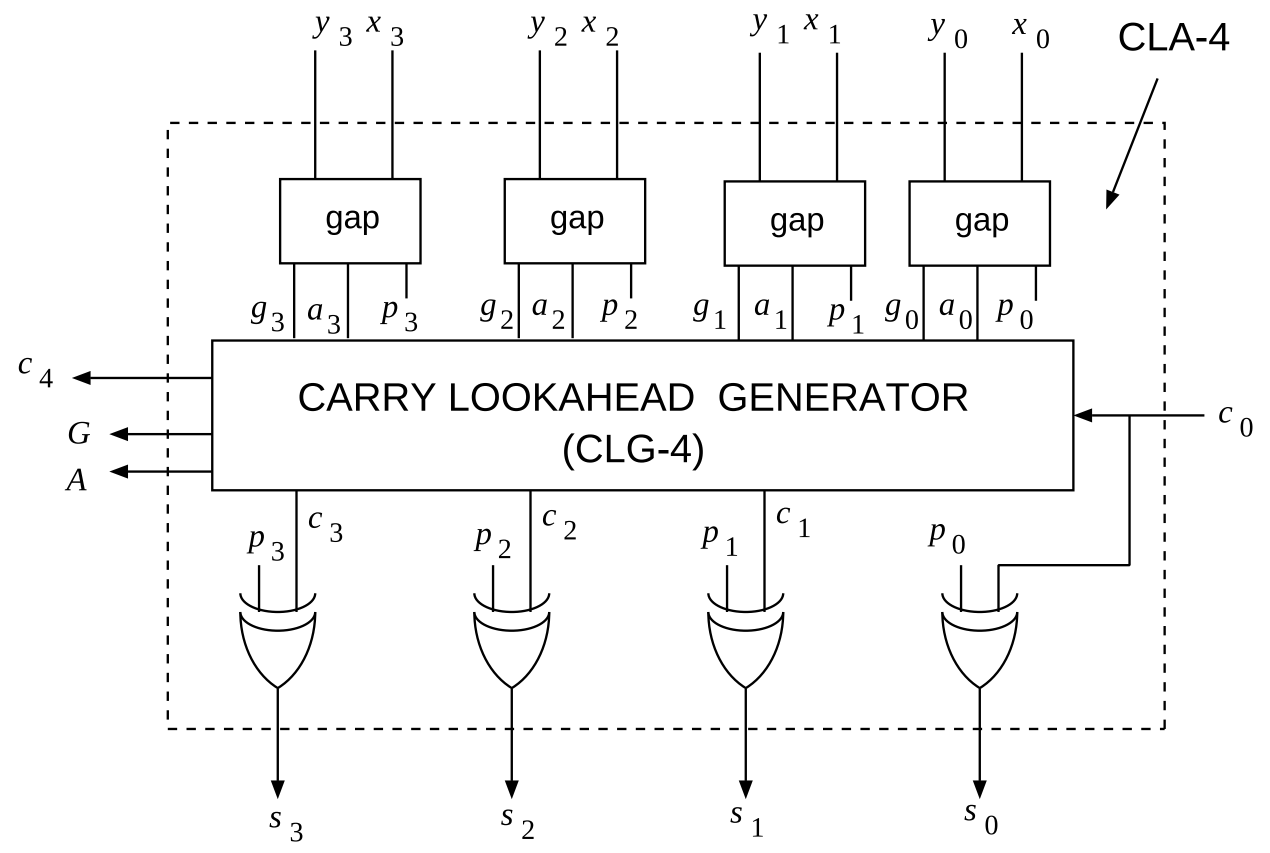

CLA-4
y_3 x_3 y_2 x_2 y_1 x_1 y_0 x_0
gap
g_3 a_3 p_3 g_2 a_2 p_2 g_1 a_1 p_1 g_0 a_0 p_0
CARRY LOOKAHEAD GENERATOR (CLG-4)
c_4
G
A
c_0
p_3 c_3 p_2 c_2 p_1 c_1 p_0
s_3 s_2 s_1 s_0

Figure 2.13: Carry-lookahead adder module ($m = 4$).

CARRY-LOOKAHEAD GENERATOR

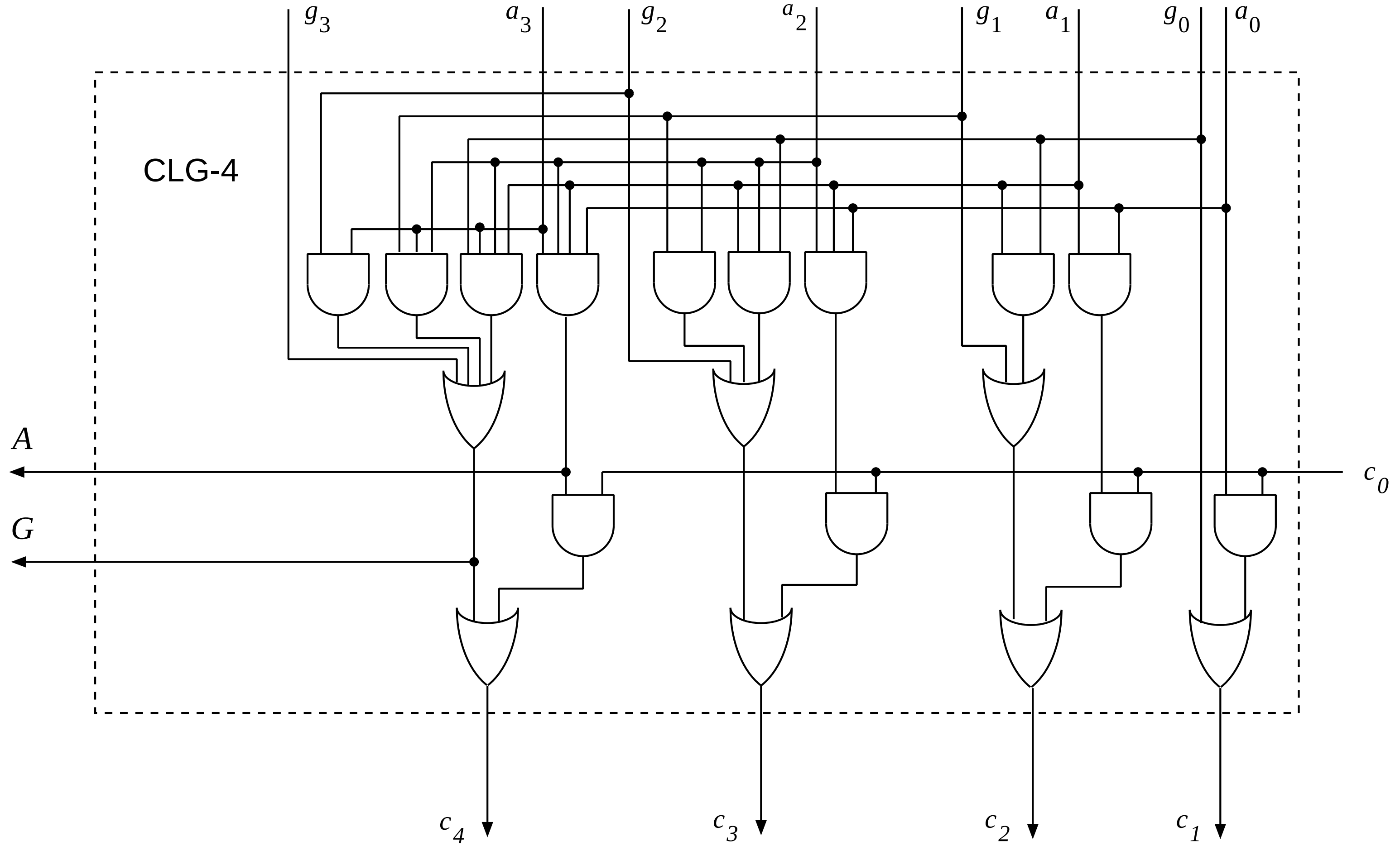


Figure 2.14: 4-bit carry-lookahead generator CLG-4.

TWO-LEVEL CARRY-LOOKAHEAD ADDER

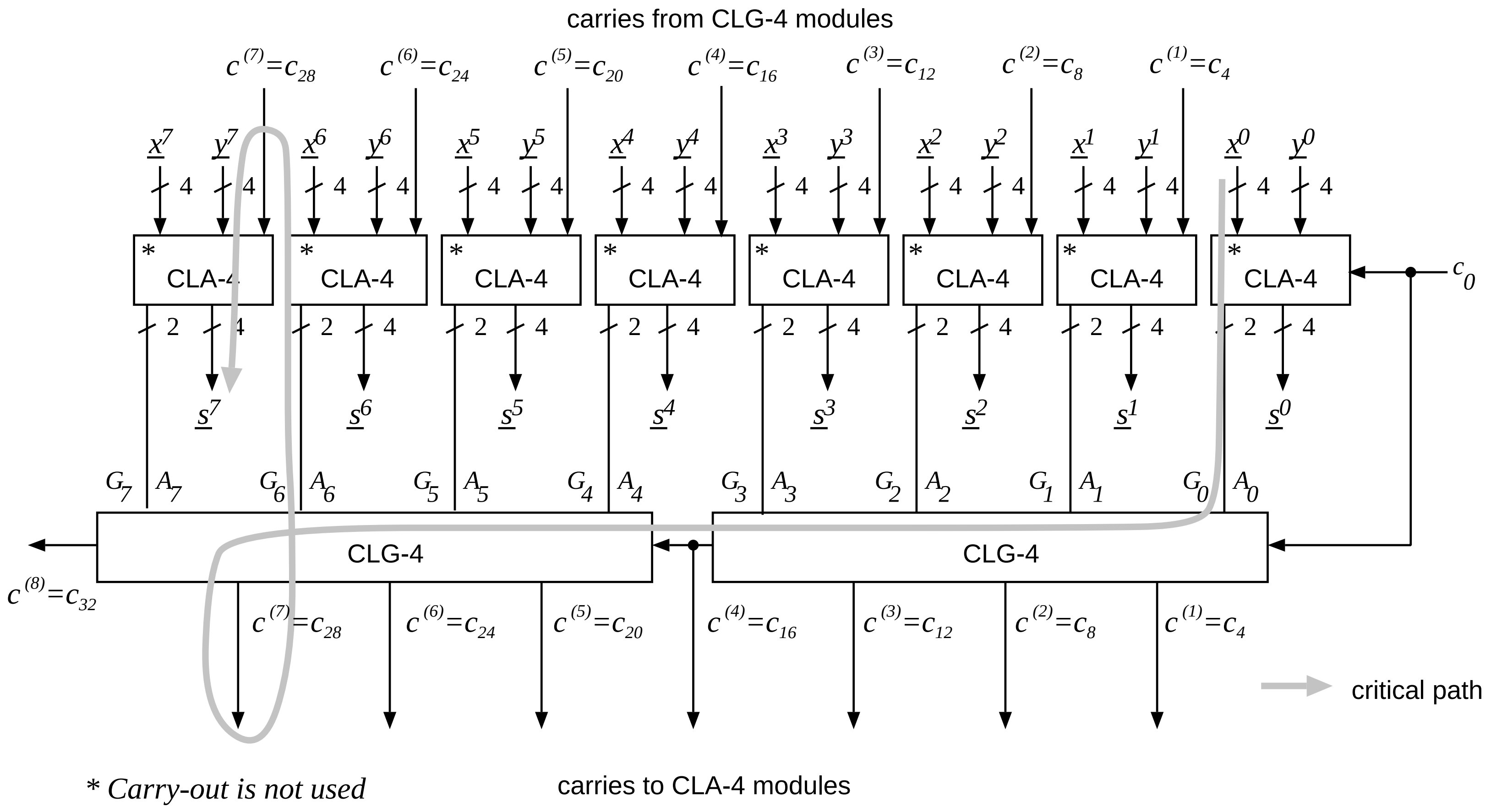

carries from CLG-4 modules
$c^{(7)}=c_{28}$
$c^{(6)}=c_{24}$
$c^{(5)}=c_{20}$
$c^{(4)}=c_{16}$
$c^{(3)}=c_{12}$
$c^{(2)}=c_{8}$
$c^{(1)}=c_{4}$
$\underline{x}^7$ $\underline{y}^7$ $\underline{x}^6$ $\underline{y}^6$ $\underline{x}^5$ $\underline{y}^5$ $\underline{x}^4$ $\underline{y}^4$ $\underline{x}^3$ $\underline{y}^3$ $\underline{x}^2$ $\underline{y}^2$ $\underline{x}^1$ $\underline{y}^1$ $\underline{x}^0$ $\underline{y}^0$
4
* CLA-4
c_0
2
$\underline{s}^7$ $\underline{s}^6$ $\underline{s}^5$ $\underline{s}^4$ $\underline{s}^3$ $\underline{s}^2$ $\underline{s}^1$ $\underline{s}^0$
G_7 A_7 G_6 A_6 G_5 A_5 G_4 A_4 G_3 A_3 G_2 A_2 G_1 A_1 G_0 A_0
CLG-4
CLG-4
$c^{(8)}=c_{32}$
$c^{(7)}=c_{28}$
$c^{(6)}=c_{24}$
$c^{(5)}=c_{20}$
$c^{(4)}=c_{16}$
$c^{(3)}=c_{12}$
$c^{(2)}=c_{8}$
$c^{(1)}=c_{4}$
critical path
* Carry-out is not used
carries to CLA-4 modules

Figure 2.15: Two-level carry-lookahead adder ($n = 32$)

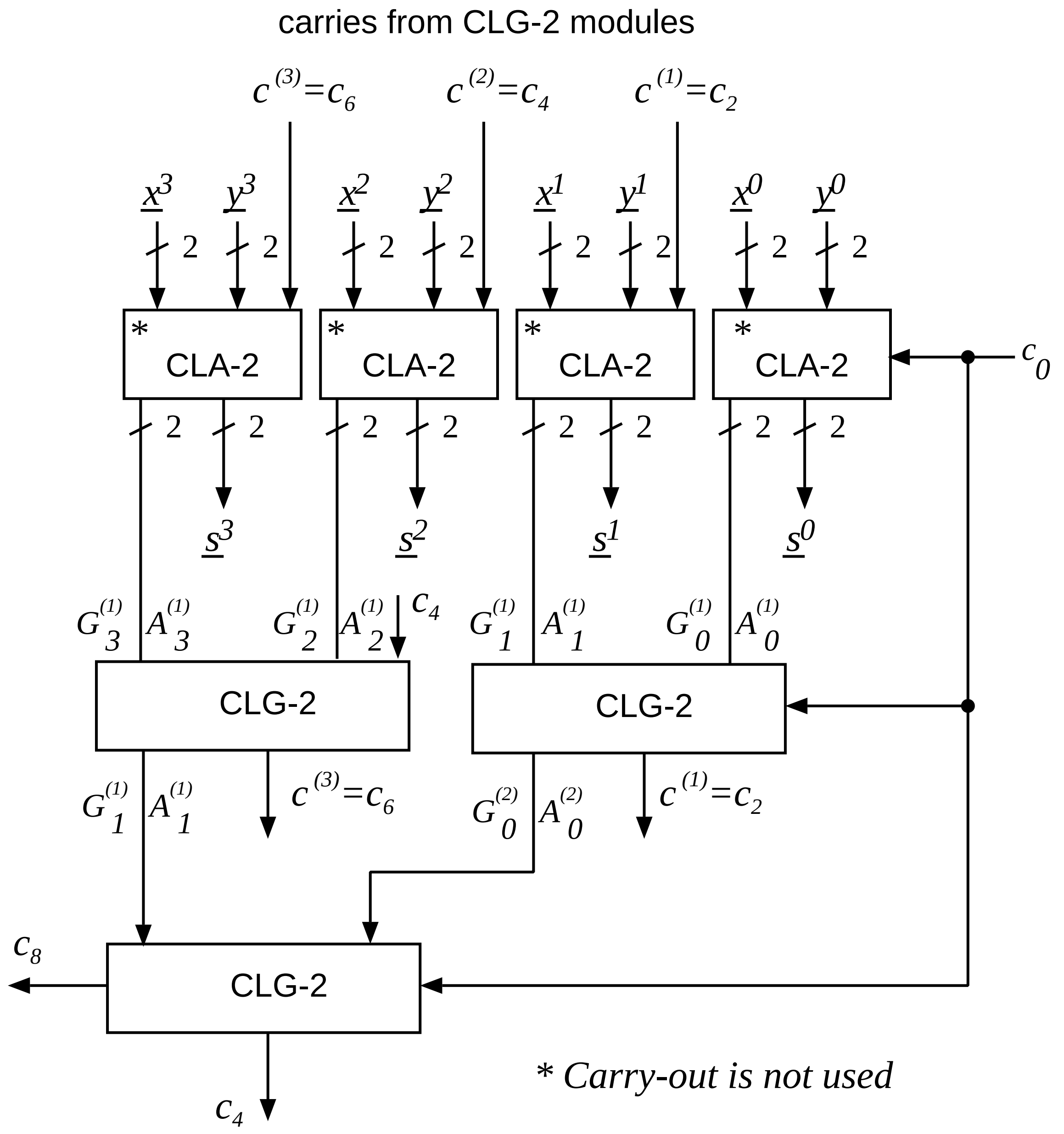

carries from CLG-2 modules
$c^{(3)}=c_6$
$c^{(2)}=c_4$
$c^{(1)}=c_2$
$\underline{x}^3$ $\underline{y}^3$ $\underline{x}^2$ $\underline{y}^2$ $\underline{x}^1$ $\underline{y}^1$ $\underline{x}^0$ $\underline{y}^0$
2
* CLA-2
c_0
$\underline{s}^3$ $\underline{s}^2$ $\underline{s}^1$ $\underline{s}^0$
$G_3^{(1)}$ $A_3^{(1)}$ $G_2^{(1)}$ $A_2^{(1)}$ c_4 $G_1^{(1)}$ $A_1^{(1)}$ $G_0^{(1)}$ $A_0^{(1)}$
CLG-2
$G_1^{(1)}$ $A_1^{(1)}$ $c^{(3)}=c_6$ $G_0^{(2)}$ $A_0^{(2)}$ $c^{(1)}=c_2$
c_8
c_4
* Carry-out is not used

Figure 2.16: Three-level carry-lookahead adder ($n = 8$, $m = 2$).

Prefix adders

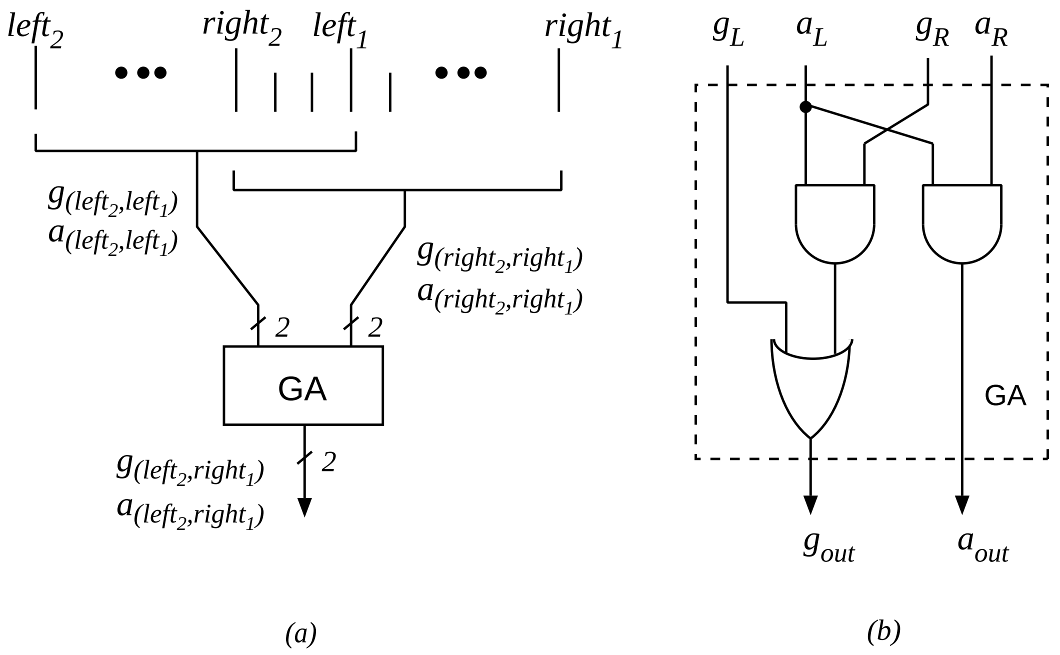
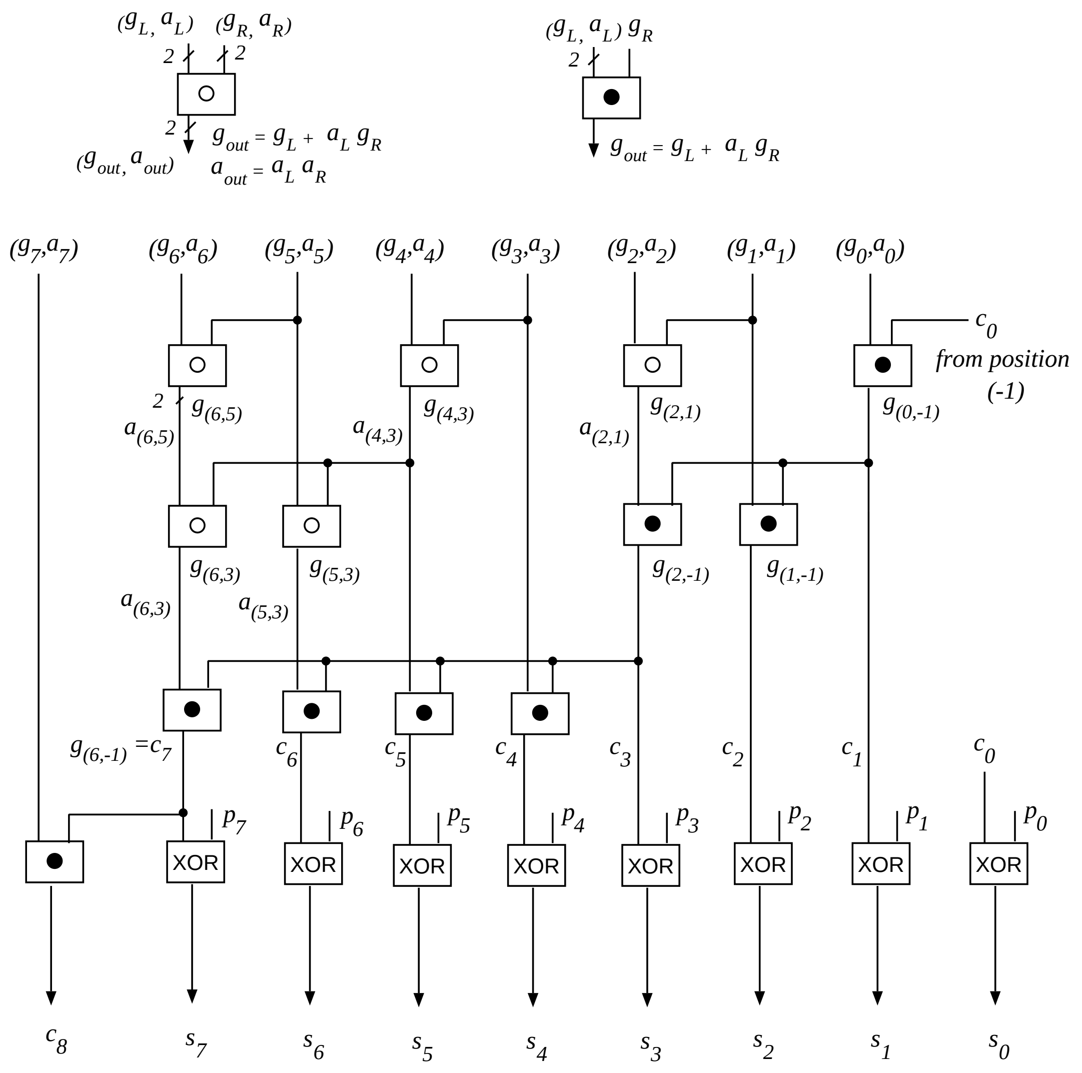
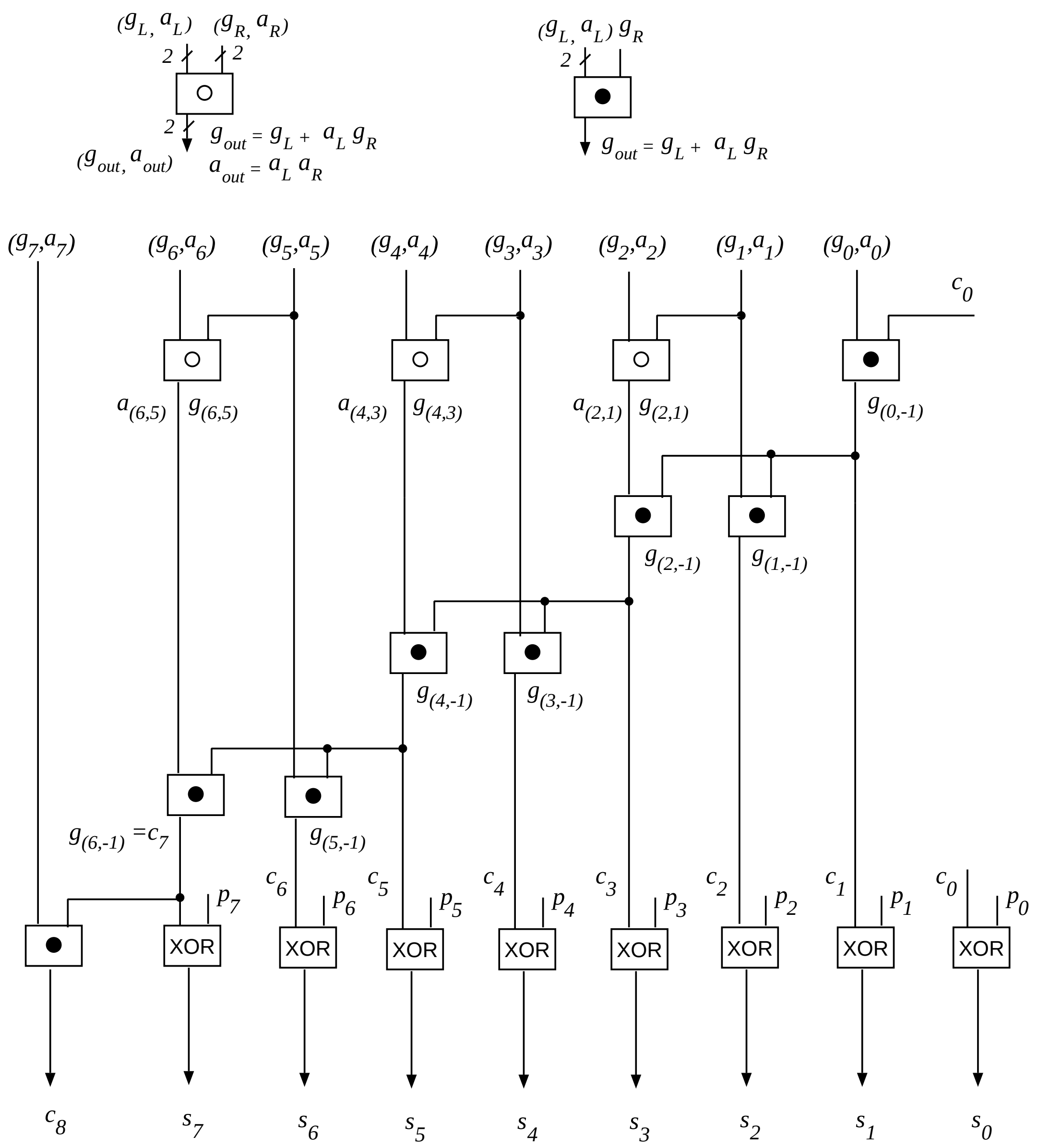


Figure 2.17: Composition of spans in computing (g, a) signals.



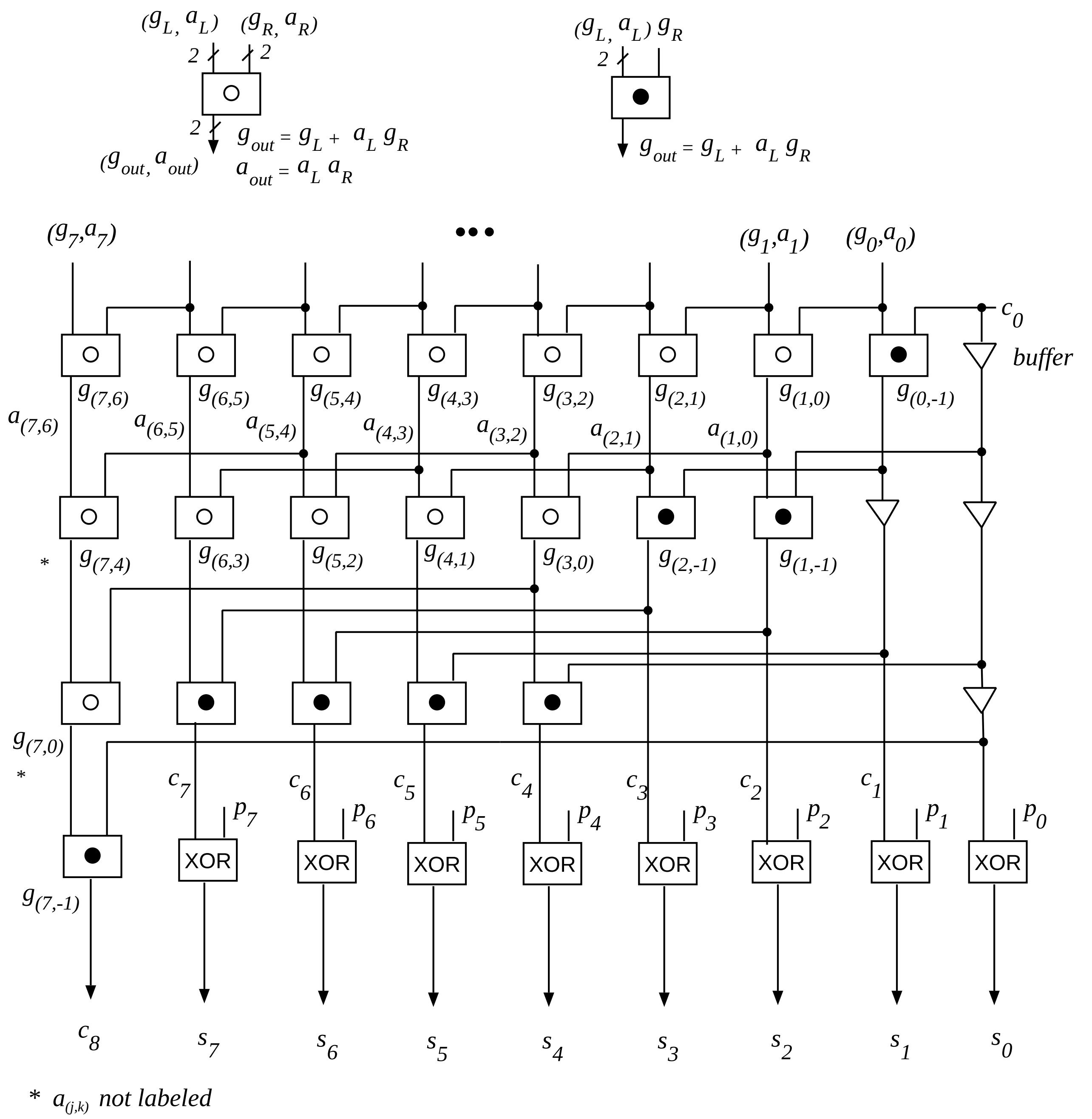
(g_L, a_L) (g_R, a_R)
2
(g_{out}, a_{out})
$g_{out} = g_L + a_L g_R$
$a_{out} = a_L a_R$
(g_L, a_L) g_R
$g_{out} = g_L + a_L g_R$
(g_7,a_7) (g_6,a_6) (g_5,a_5) (g_4,a_4) (g_3,a_3) (g_2,a_2) (g_1,a_1) (g_0,a_0)
c_0 from position (-1)
$g_{(6,5)}$ $a_{(6,5)}$ $g_{(4,3)}$ $a_{(4,3)}$ $g_{(2,1)}$ $a_{(2,1)}$ $g_{(0,-1)}$
$g_{(6,3)}$ $a_{(6,3)}$ $g_{(5,3)}$ $a_{(5,3)}$ $g_{(2,-1)}$ $g_{(1,-1)}$
$g_{(6,-1)} = c_7$
c_6 c_5 c_4 c_3 c_2 c_1 c_0
p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0
XOR
c_8 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

Figure 2.18: 8-bit prefix adder. (Modules to obtain p_i, g_i, and a_i signals not shown.)



(g_L, a_L) (g_R, a_R)
$g_{out} = g_L + a_L g_R$
$a_{out} = a_L a_R$
(g_{out}, a_{out})
(g_L, a_L) g_R
$g_{out} = g_L + a_L g_R$
(g_7,a_7) (g_6,a_6) (g_5,a_5) (g_4,a_4) (g_3,a_3) (g_2,a_2) (g_1,a_1) (g_0,a_0)
c_0
$a_{(6,5)}$ $g_{(6,5)}$ $a_{(4,3)}$ $g_{(4,3)}$ $a_{(2,1)}$ $g_{(2,1)}$ $g_{(0,-1)}$
$g_{(2,-1)}$ $g_{(1,-1)}$
$g_{(4,-1)}$ $g_{(3,-1)}$
$g_{(6,-1)} = c_7$ $g_{(5,-1)}$
c_6 c_5 c_4 c_3 c_2 c_1 c_0
p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0
XOR
c_8 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

Figure 2.19: 8-bit prefix adder with maximum fanout of three and five levels. (Modules to obtain p_i, g_i, and a_i signals not shown.)



(g_L, a_L) (g_R, a_R)
2
2
2
(g_{out}, a_{out})
$g_{out} = g_L + a_L g_R$
$a_{out} = a_L a_R$
(g_L, a_L) g_R
2
$g_{out} = g_L + a_L g_R$
(g_7, a_7)
(g_1, a_1)
(g_0, a_0)
c_0
buffer
$g_{(7,6)}$ $g_{(6,5)}$ $g_{(5,4)}$ $g_{(4,3)}$ $g_{(3,2)}$ $g_{(2,1)}$ $g_{(1,0)}$ $g_{(0,-1)}$
$a_{(7,6)}$ $a_{(6,5)}$ $a_{(5,4)}$ $a_{(4,3)}$ $a_{(3,2)}$ $a_{(2,1)}$ $a_{(1,0)}$
$g_{(7,4)}$ $g_{(6,3)}$ $g_{(5,2)}$ $g_{(4,1)}$ $g_{(3,0)}$ $g_{(2,-1)}$ $g_{(1,-1)}$
$g_{(7,0)}$
c_7 c_6 c_5 c_4 c_3 c_2 c_1
p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0
XOR
$g_{(7,-1)}$
c_8 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0
* $a_{(j,k)}$ not labeled

Figure 2.20: 8-bit prefix adder with minimum number of levels and fanout of two.(Modules to obtain p_i, g_i, and a_i signals not shown.)

CONDITIONAL ADDER (COND ADDER)

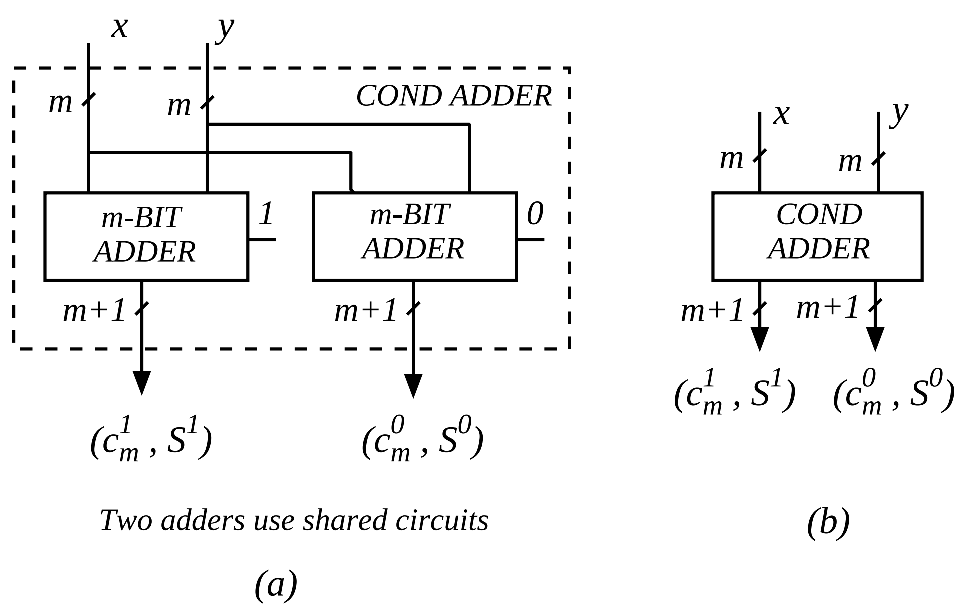


Figure 2.21: (a) Obtaining conditional outputs. (b) Combined conditi onal adder.

CARRY-SELECT ADDER

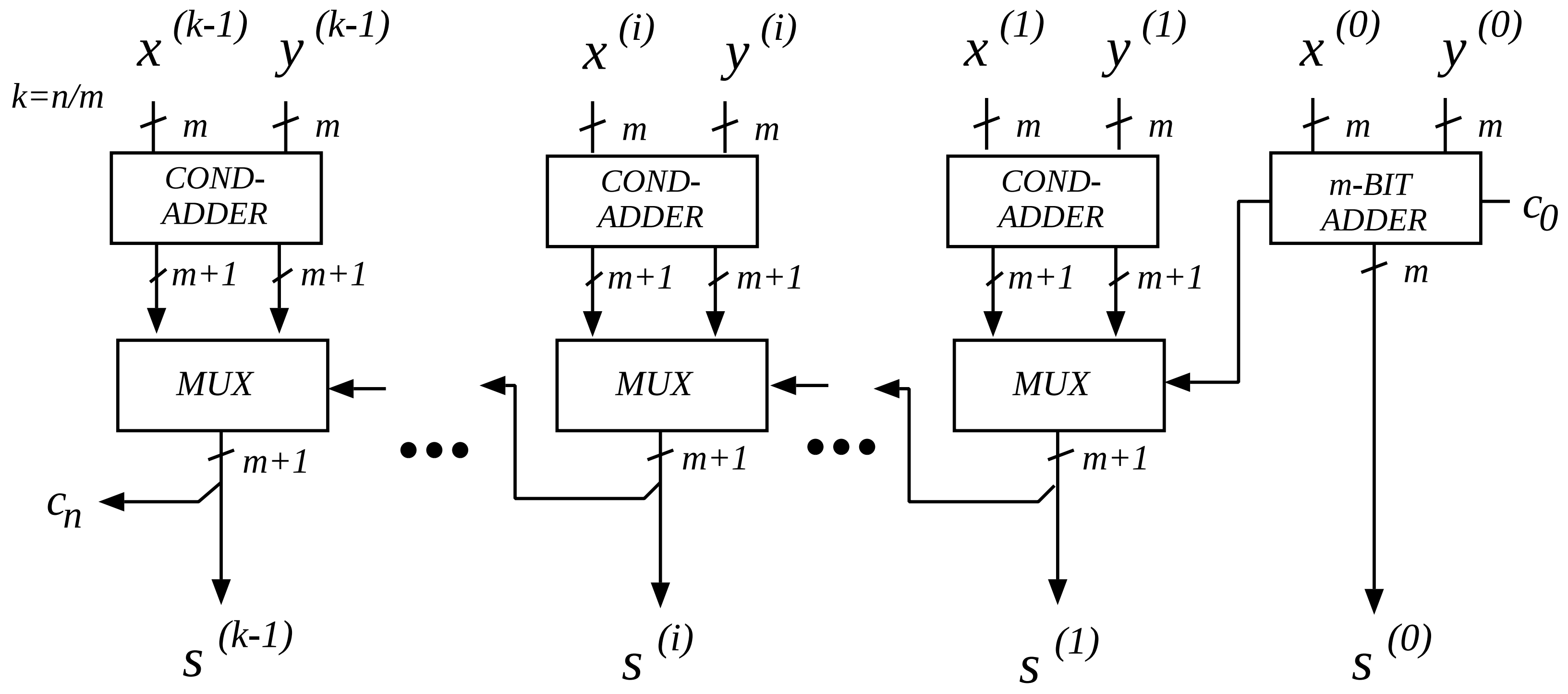


Figure 2.22: Carry-select adder.

CONDITIONAL-SUM ADDER

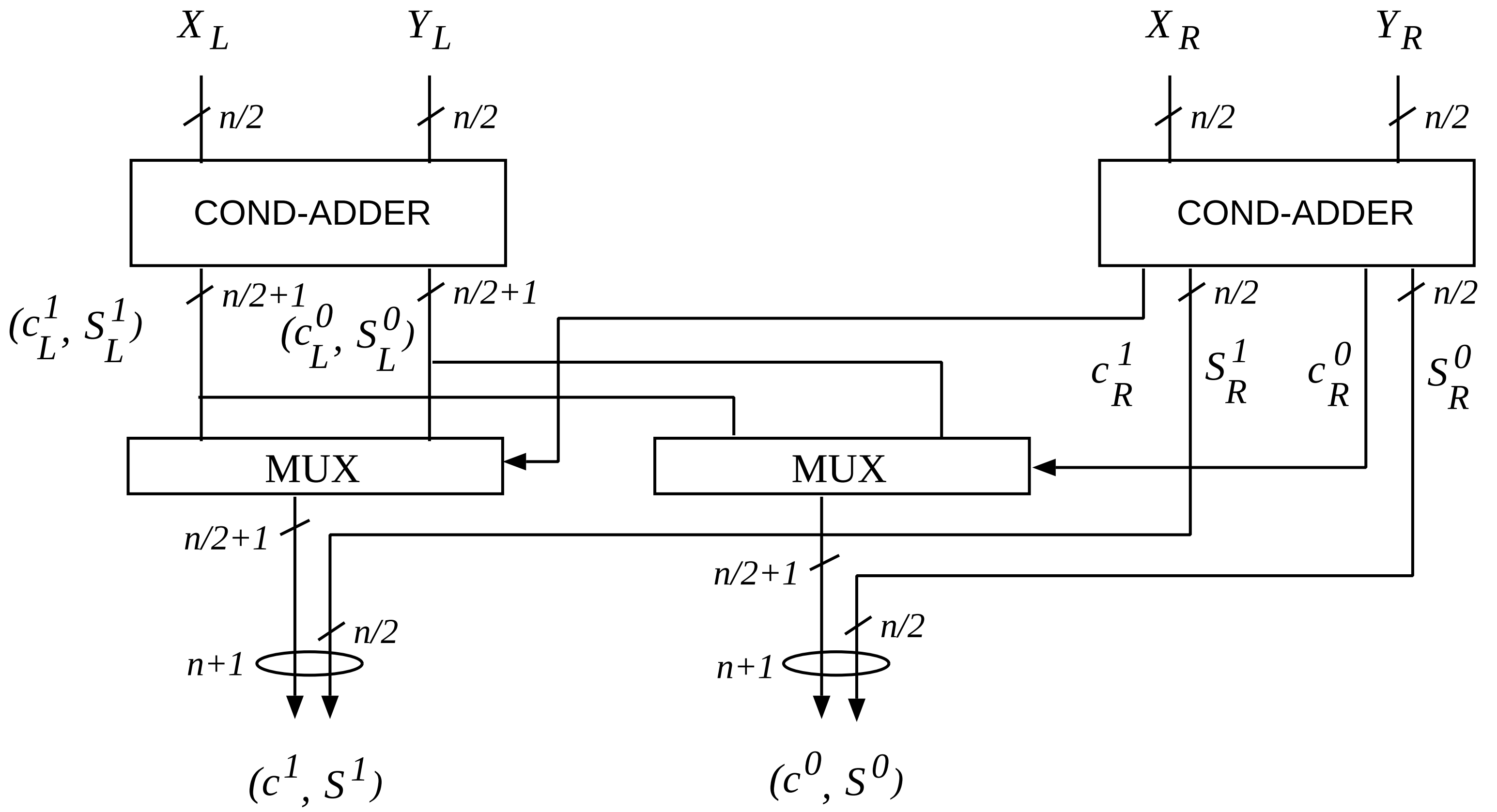


Figure 2.23: Doubling the number of bits of the conditional sum.

16-bit CONDITIONAL-SUM ADDER

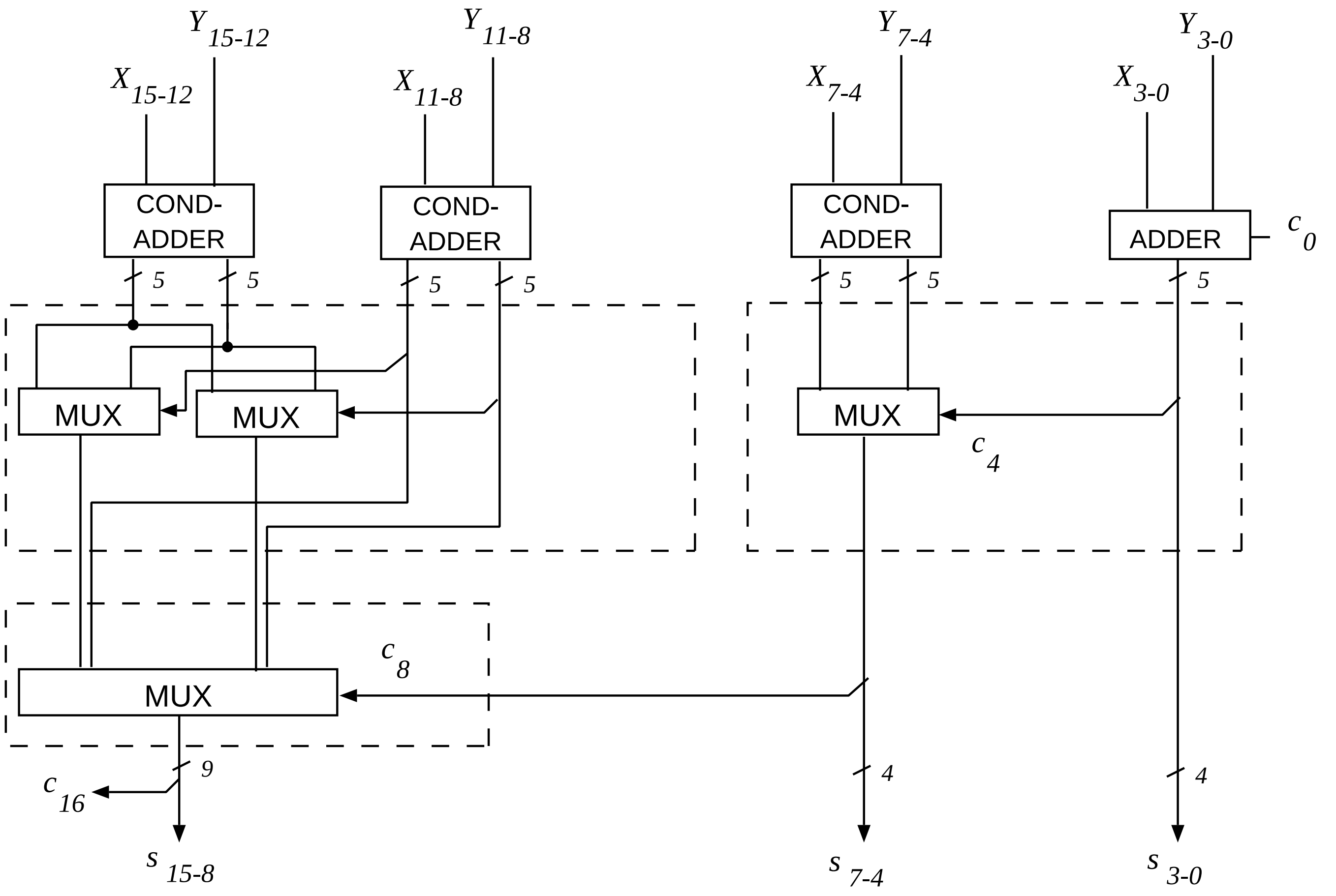


Figure 2.24: 16-bit conditional-sum adder ($m = 4$).

TEMPLATE

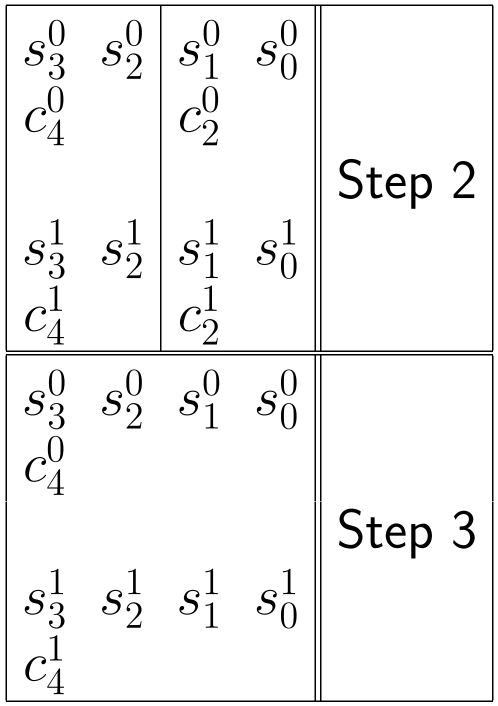

s_3^0 s_2^0 s_1^0 s_0^0
c_4^0 c_2^0
s_3^1 s_2^1 s_1^1 s_0^1
c_4^1 c_2^1
Step 2
s_3^0 s_2^0 s_1^0 s_0^0
c_4^0
s_3^1 s_2^1 s_1^1 s_0^1
c_4^1
Step 3

Figure 2.25: Conditional-sum addition for eight bits with $m = 1$: (a) Template. (b) E xample.

	7	6	5	4	3	2	1	0	
x	0	1	0	1	1	0	1	1	$c_0 = 0$
y	0	1	0	1	0	1	0	1	
s^0	0	0	0	0	1	1	1	0	
c^0	0	**1**	0	**1**	0	**0**	0	**1**	
									Step 1
s^1	1	1	1	1	0	0	0		
c^1	0	**1**	0	**1**	1	**1**	1		
s^0	1	0	1	0	1	1	0	0	
c^0	0		**0**		0		**1**		
									Step 2
s^1	1	1	1	1	0	0			
c^1	0		**0**		1				
s^0	1	0	1	0	0	0	0	0	
c^0	0				**1**				
									Step 3
s^1	1	0	1	1					
c^1	0								
s	1	0	1	1	0	0	0	0	

PIPELINED ADDERS

Increase throughput

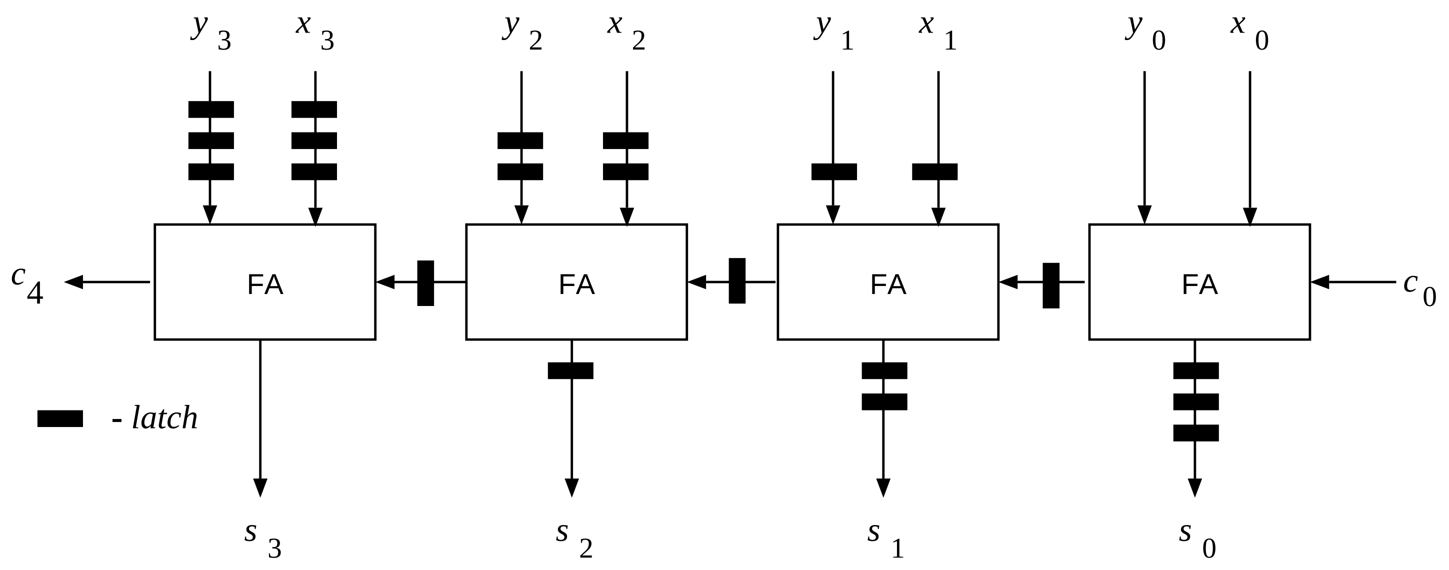


Figure 2.26: Pipelined carry-ripple adder (for group size of 1 and $n = 4$)

VARIABLE-TIME ADDER: Type 1

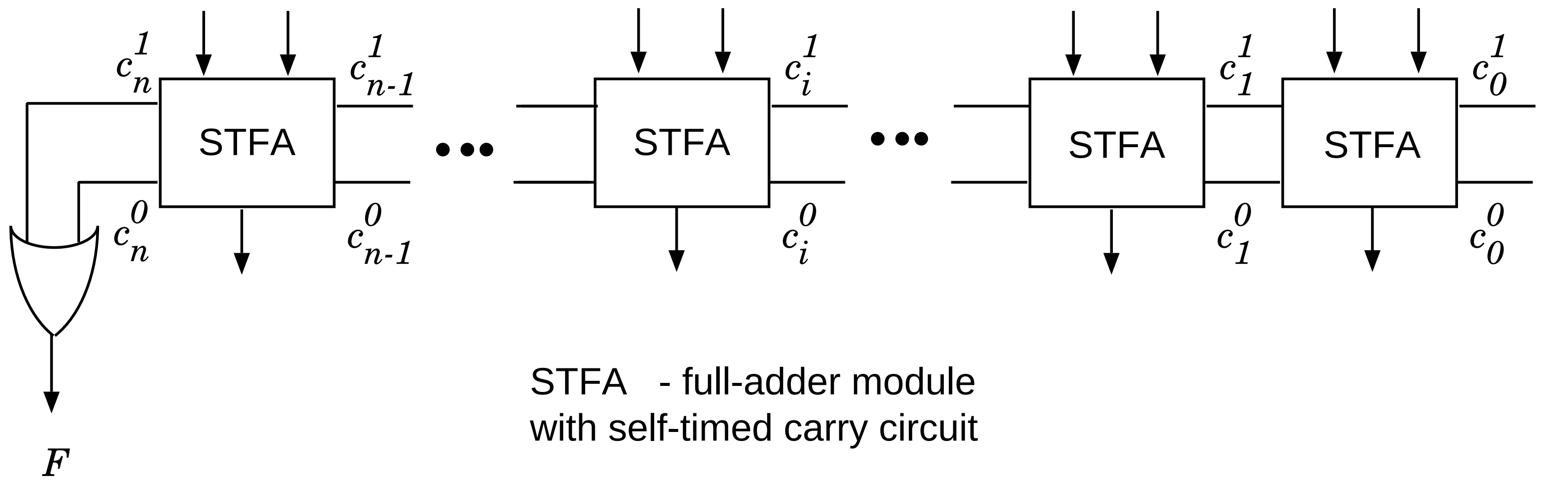


Figure 2.27: Variable-time adder: Type 1.

Two carry signals:

$$c_i^0 \quad zero\ carry \qquad c_i^1 \quad one\ carry$$

with coding:

c_i^0	c_i^1	c_i
0	0	not determined (yet)
0	1	1
1	0	0
1	1	does not occur

VARIABLE-TIME ADDER: Type 1 cont.

STFA module expressions:

$$\begin{aligned} c^0_{i+1} &= k_i(c^0_i + c^1_i) + p_i c^0_i = k_i c^1_i + (p_i + k_i) c^0_i \\ c^1_{i+1} &= g_i(c^0_i + c^1_i) + p_i c^1_i = g_i c^0_i + (p_i + g_i) c^1_i \end{aligned}$$

$$s_i = p_i \oplus c^1_i$$

$$k_i = x'_i y'_i, \quad g_i = x_i y_i, \quad p_i = x_i \oplus y_i$$

Addition time: based on $actual$ delays, not worst-case

$$T_{var-1} = \sum_{i=0}^{n-1} t_{c,i}$$

VARIABLE-TIME ADDER: Type 2

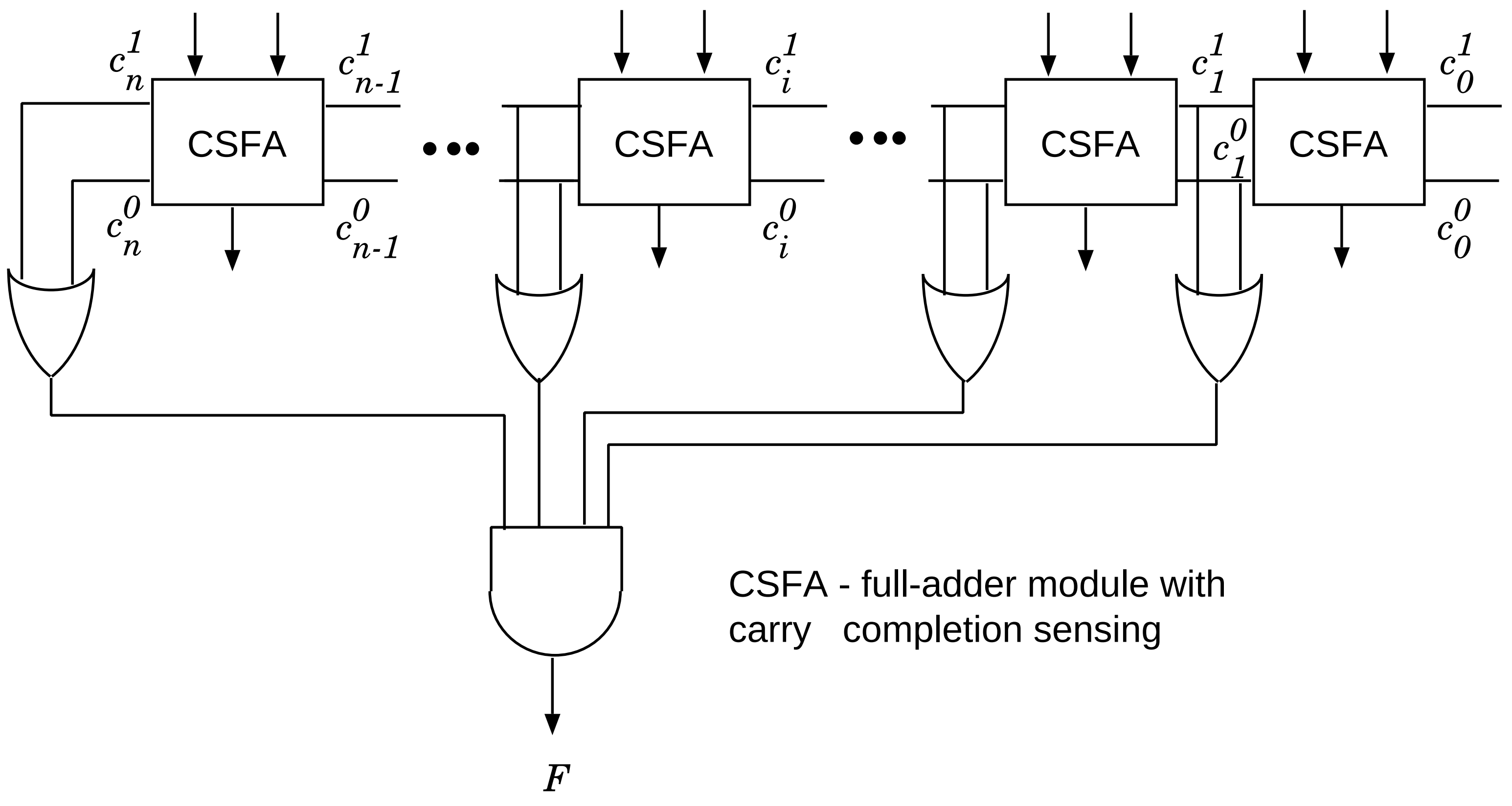


Figure 2.28: Variable-time adder: Type 2.

Carry chains initiated simultaneously
CSFA module expressions:

$$c_{i+1}^0 = k_i + p_i c_i^0, \quad c_{i+1}^1 = g_i + p_i c_i^1$$

Example

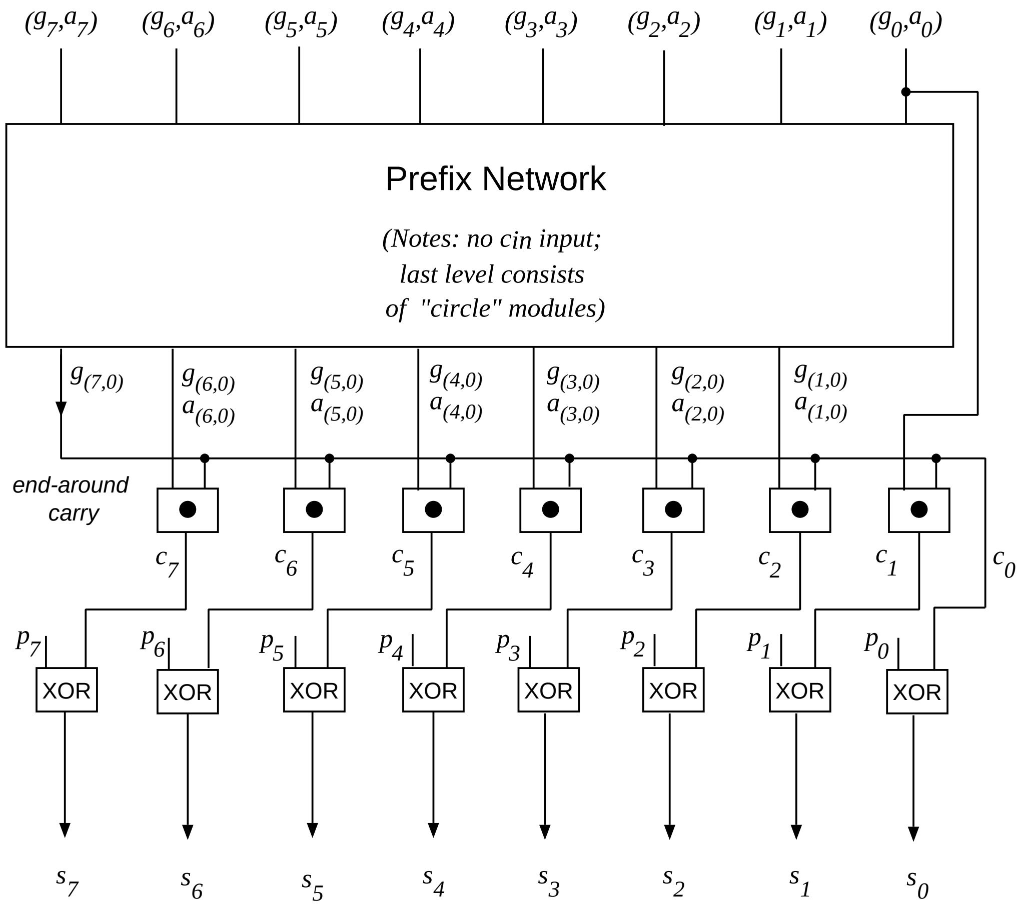
X	0	1	1	0	0	0	1	1	1	0	0	1	1	0	1	0	
Y	1	0	1	0	1	1	0	0	1	1	1	0	0	1	1	0	
+	a	a	a	b	c	c	c	c	c	d	d	d	d	d	d	e	Prop.chains

Completion signal:

$$F = \prod_{i=0}^{n-1} (c_i^0 + c_i^1)$$

Addition time: proportional to $log_2(n)$

2's COMPLEMENT AND 1s' COMPLEMENT ADDERS



(g_7,a_7) (g_6,a_6) (g_5,a_5) (g_4,a_4) (g_3,a_3) (g_2,a_2) (g_1,a_1) (g_0,a_0)
Prefix Network
(Notes: no cin input; last level consists of "circle" modules)
$g_{(7,0)}$
$g_{(6,0)}$ $a_{(6,0)}$
$g_{(5,0)}$ $a_{(5,0)}$
$g_{(4,0)}$ $a_{(4,0)}$
$g_{(3,0)}$ $a_{(3,0)}$
$g_{(2,0)}$ $a_{(2,0)}$
$g_{(1,0)}$ $a_{(1,0)}$
end-around carry
c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0
p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0
XOR
s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

Figure 2.29: Implementing ones' complement adder with prefix network. (Modules to obtain p_i, g_i, and a_i signals not shown.)

ADDERS WITH REDUNDANT DIGIT-SET

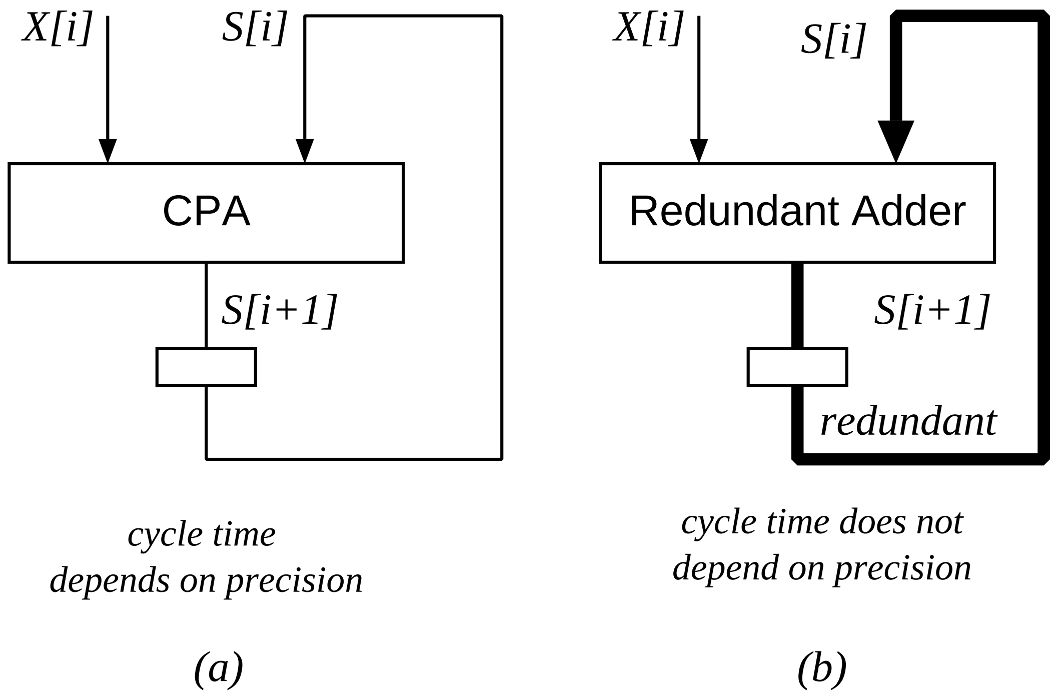


Figure 2.30: Accumulation with (a) non-redundant, and (b) redundant representation of sum.

CARRY-SAVE ADDER

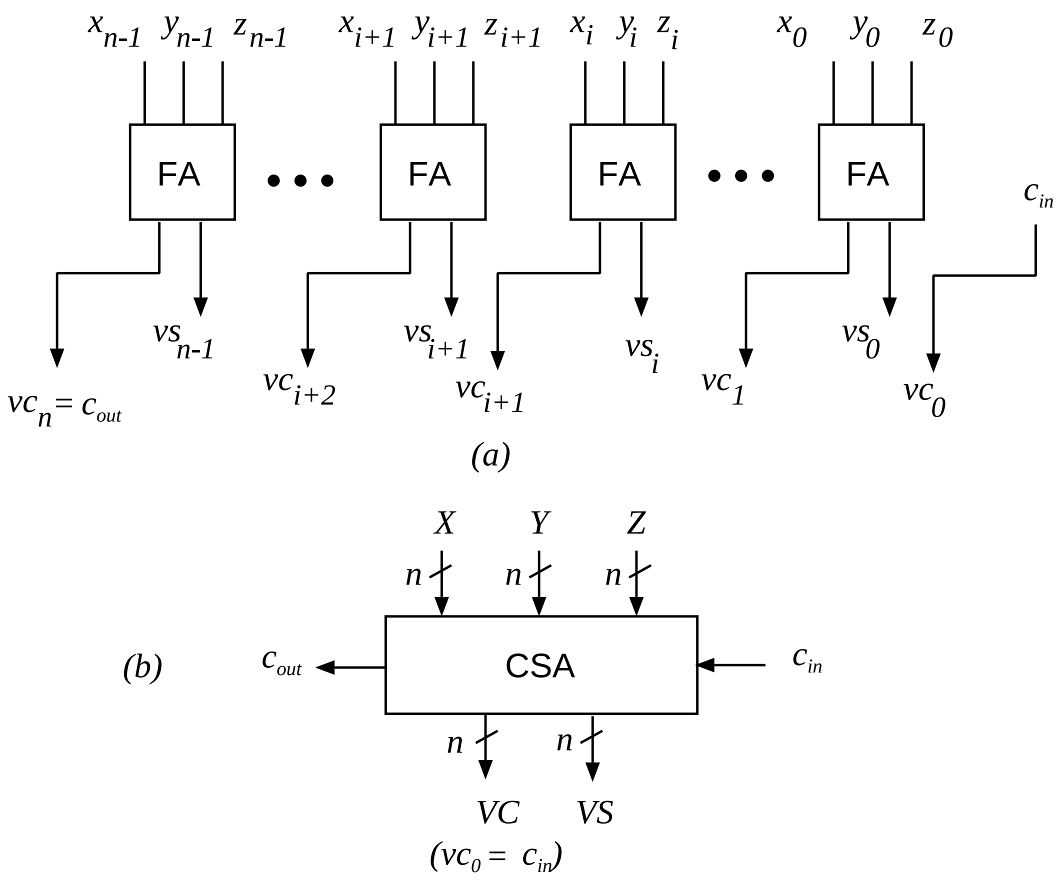


Figure 2.31: Carry-save adder: (a) Bit level. (b) Bit-vector level.

EXAMPLE OF CARRY-SAVE OPERATION

X		0	1	1	1	0	1	0	0
Y		0	0	1	1	1	0	1	1
Z		1	0	1	0	1	0	1	0
VS		1	1	1	0	0	1	0	1
(c_{out}, VC)	0	0	1	1	1	0	1	0	1
digit value	0	1	2	2	1	0	2	0	2

[4:2] ADDER

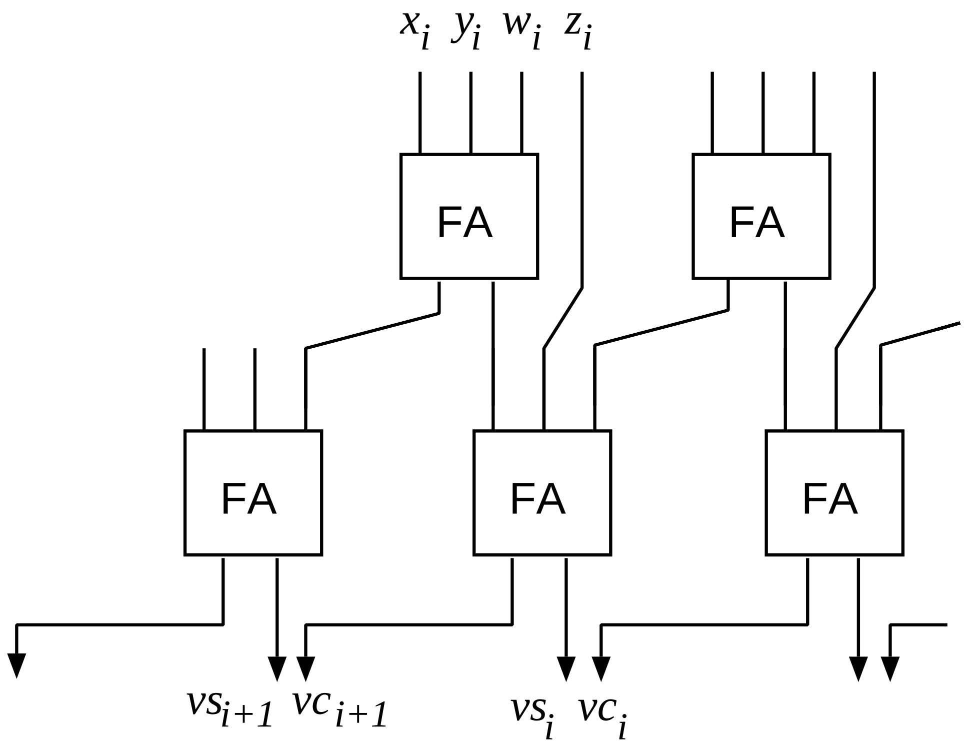


Figure 2.32: [4:2] adder.

HIGH RADIX CARRY-SAVE REPRESENTATION

XS		1	0	1	1	0	1	1	0	0
XC				1			1			0
Y		0	1	0	0	0	1	1	1	1
VS		0	0	0	1	1	1	0	1	1
$(c_{out}$,VC)	1			0			1			0

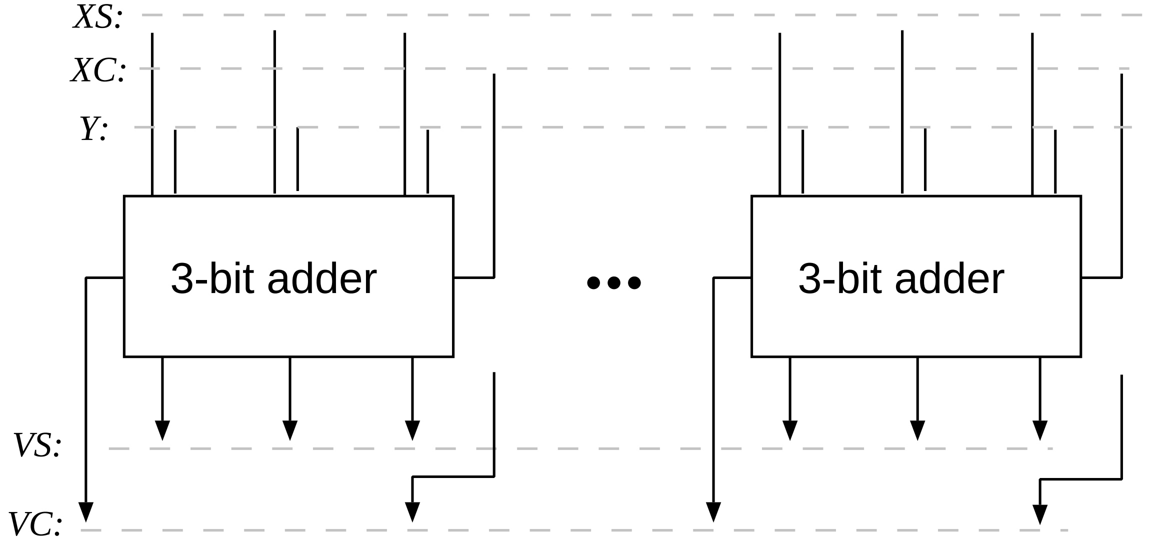


Figure 2.33: Radix-8 carry-save adder.

SIGNED-DIGIT ADDITION

- Uses signed-digit representation (redundant)

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

with digit set

$$D = \{-a, \ldots, -1, 0, 1, \ldots, a\}$$

- Limits carry propagation to next position
- Addition algorithm:

$$\begin{aligned} \text{Step 1:}\quad & x + y = w + t \\ & x_i + y_i = w_i + rt_{i+1} \\ \\ \text{Step 2:}\quad & s = w + t \\ & s_i = w_i + t_i \end{aligned}$$

- No carry produced in Step 2

SD ADDER

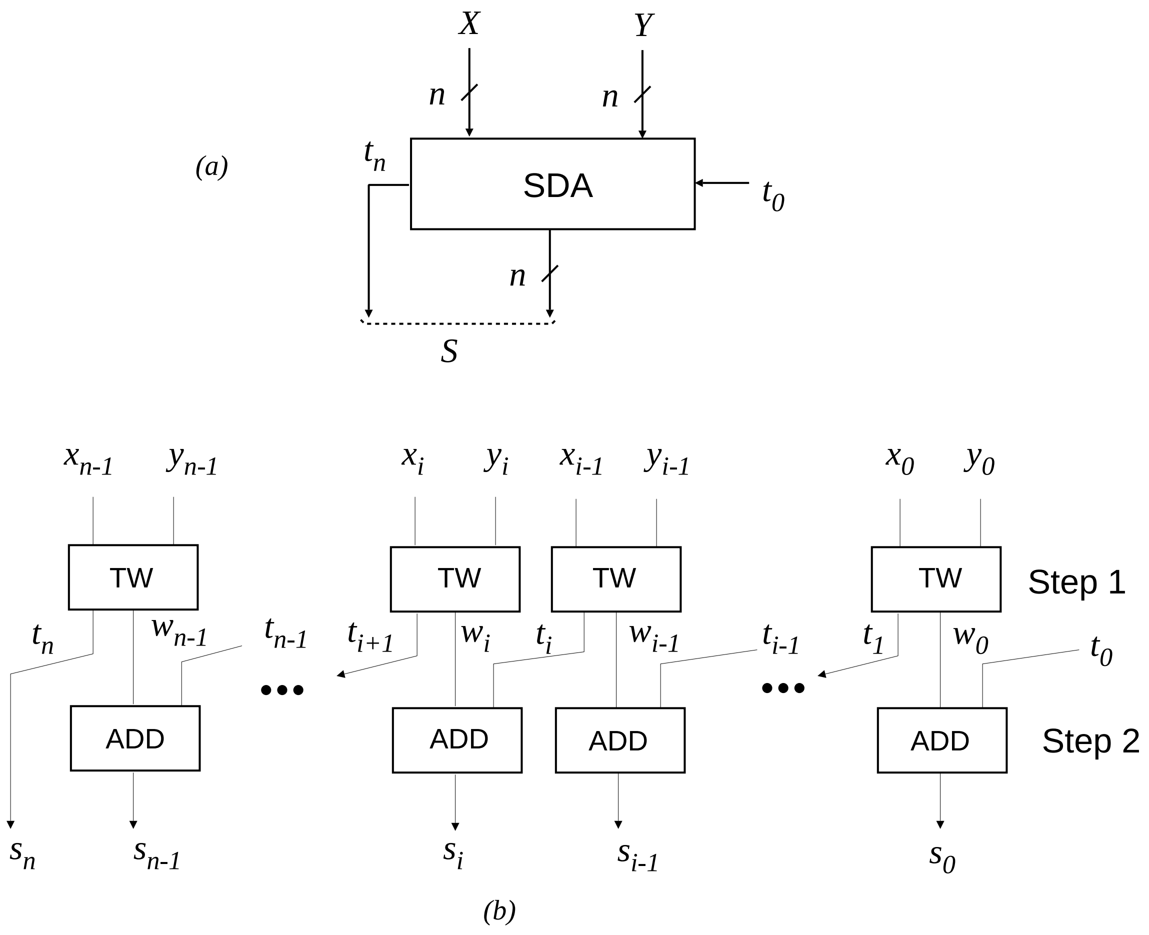


Figure 2.34: Signed-digit addition.

CASES CONSIDERED

Case A : two SD operands; result SD

Step 1:

$$(t_{i+1},\ w_i) = \begin{cases} (0,\ x_i + y_i) & \textbf{if } -a+1 \le x_i + y_i \le a-1 \\ (1,\ x_i + y_i - r) & \textbf{if } x_i + y_i \ge a \\ (-1,\ x_i + y_i + r) & \textbf{if } x_i + y_i \le -a \end{cases}$$

- algorithm modified for $r = 2$

Case B : two conventional operands; result SD

Case C : one conventional, one SD; result SD

SIGNED-BINARY ADDITION: METHOD 1 (DOUBLE RECODING)

RECODING 1:

$$\begin{aligned} x_i + y_i = 2h_{i+1} + z_i \quad &\in \{-2, -1, 0, 1, 2\} \\ h_i \in \{0, 1\}, z_i &\in \{-2, -1, 0\} \\ q_i = z_i + h_i \quad &\in \{-2, -1, 0, 1\} \end{aligned}$$

RECODING 2:

$$\begin{aligned} q_i = z_i + h_i = 2t_{i+1} + w_i \quad &\in \{-2, -1, 0, 1\} \\ t_i \in \{-1, 0\}, \quad w_i &\in \{0, 1\} \end{aligned}$$

THE RESULT: $s_i = w_i + t_i \quad \in \{-1, 0, 1\}$

METHOD 1 SD ADDER

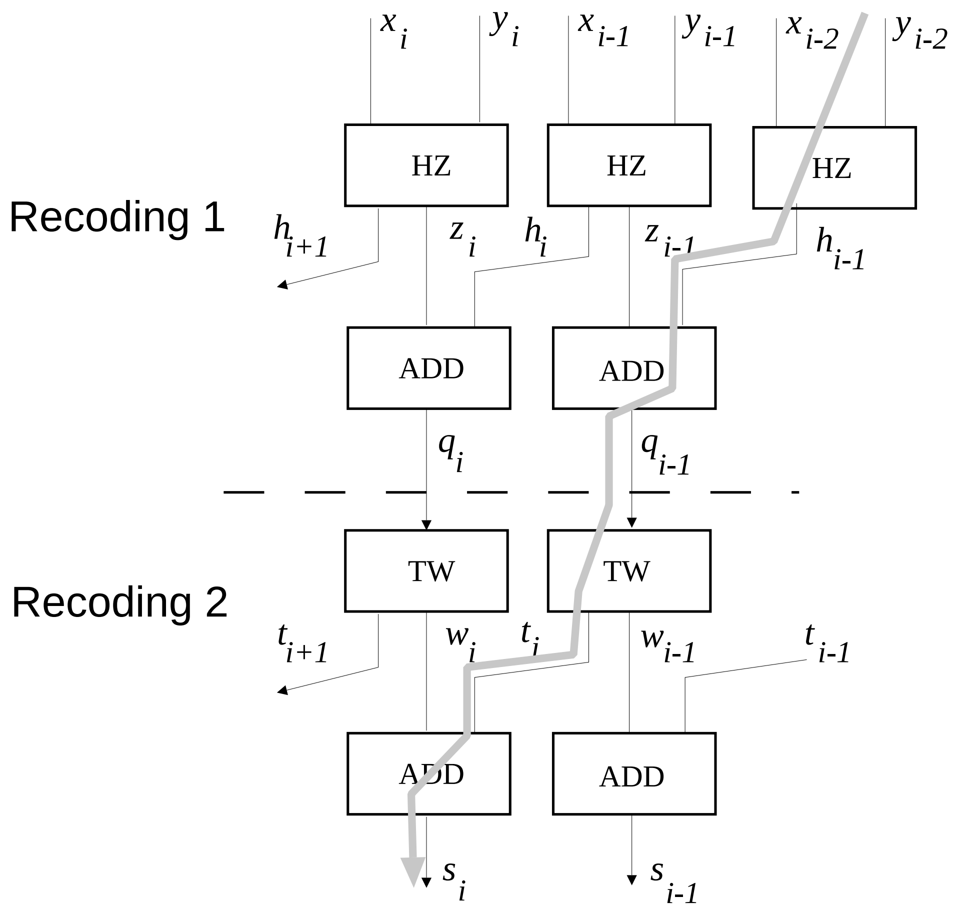


Figure 2.35: Double recoding method for signed-bit addition

SIGNED BINARY ADDITION: METHOD 2 (Using Previous Digit)

$$P_i = \begin{cases} 0 & \text{if } (x_i, y_i) \text{ both nonnegative} \\ & \text{(which implies } t_{i+1} \geq 0) \\ 1 & \text{otherwise } (t_{i+1} \leq 0) \end{cases}$$

$x_i + y_i$	P_{i-1}	t_{i+1}	w_i
2	-	1	0
1	$0(t_i \geq 0)$	1	-1
1	$1(t_i \leq 0)$	0	1
0	-	0	0
-1	$0(t_i \geq 0)$	0	-1
-1	$1(t_i \leq 0)$	-1	1
-2	-	-1	0

METHOD 2 SD ADDER

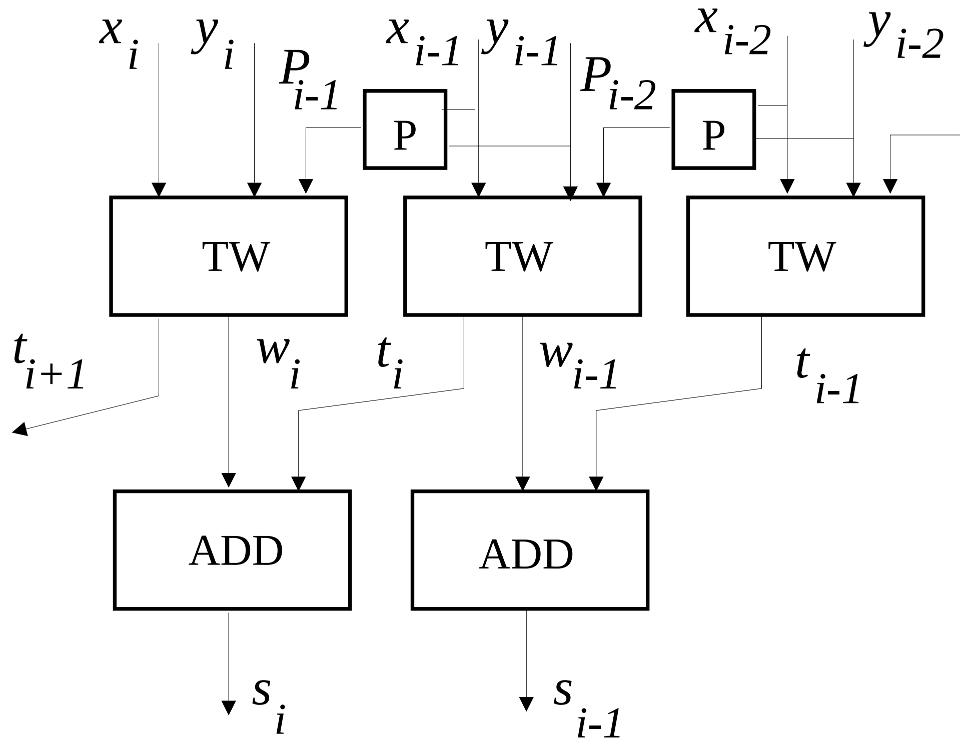


Figure 2.36: Signed-bit addition using the information from previous digit

Example

$$\begin{array}{lcccccccccc}
X & & 0 & 1 & 1 & 1 & \bar{1} & 1 & 0 & \bar{1} & 1 \\
Y & & 0 & 1 & 1 & 0 & \bar{1} & 0 & 1 & 0 & 1 \\
 & & & & & & & & & & \\
P & & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
 & & & & & & & & & & \\
W & & 0 & 0 & 0 & 1 & 0 & \bar{1} & 1 & \bar{1} & 0 \\
T & 0 & 1 & 1 & 0 & \bar{1} & 1 & 0 & 0 & 1 & 0 \\
 & & & & & & & & & & \\
S & & 1 & 1 & 0 & 0 & 1 & \bar{1} & 1 & 0 & 0
\end{array}$$

BIT-LEVEL IMPLEMENTATION OF RADIX-2 ALGORITHMS

- Case C: $x_i \in \{0,1\}, \quad y_i, s_i \in \{-1,0,1\}$
- Code: borrow-save $y_i = y_i^+ - y_i^-$, $y_i^+, y_i^- \in \{0,1\}$, sim. for s_i
- $x_i + y_i \in \{-1,0,1,2\}$: recode to (t_{i+1}, w_i), $t_{i+1} \in \{0,1\}$, $w_i \in \{-1,0\}$

$$x_i + y_i^+ - y_i^- = 2t_{i+1} + w_i$$

$x_i + y_i$	-1	0	1	2
w_i	-1	0	-1	0
t_{i+1}	0	0	1	1

SWITCHING FUNCTIONS FOR t_{i+1} AND w_i

x_i	y_i^+	y_i^-	$x_i + y_i$	t_{i+1}	$-w_i$
0	0	0	0	0	0
0	0	1	-1	0	1
0	1	0	1	1	1
0	1	1	0	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	2	1	0
1	1	1	1	1	1

$$\begin{aligned} w_i &= (x_i \oplus y_i^+ \oplus (y_i^-)')' \\ t_{i+1} &= x_i y_i^+ + x_i (y_i^-)' + y_i^+ (y_i^-)' \end{aligned}$$

$\Longrightarrow$ implemented using a full-adder and inverters (for variables subtracted)

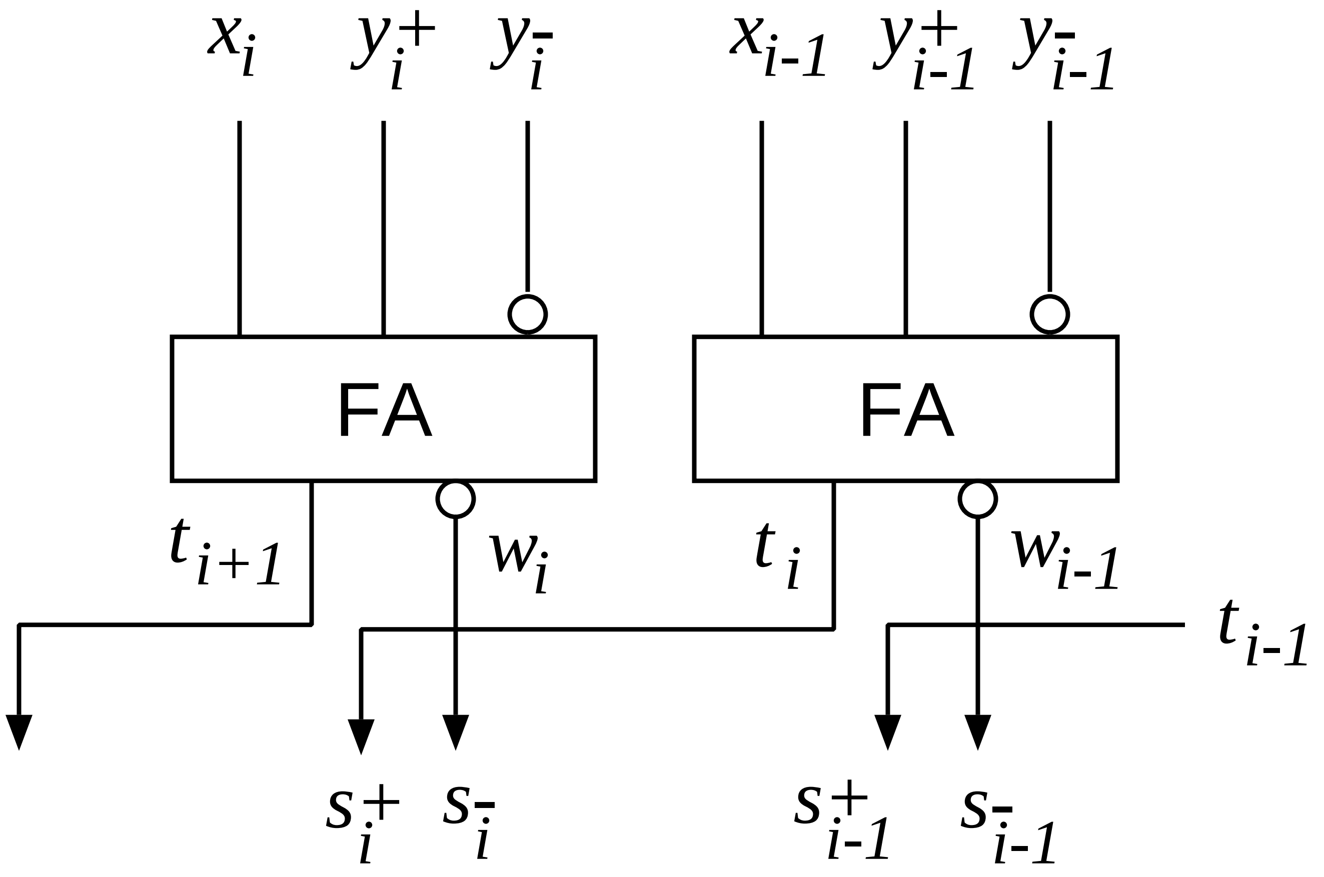


Figure 2.37: Redundant adder: one operand conventional, one operand redundant, result redundant.

BOTH OPERANDS REDUNDANT

- Apply double recoding

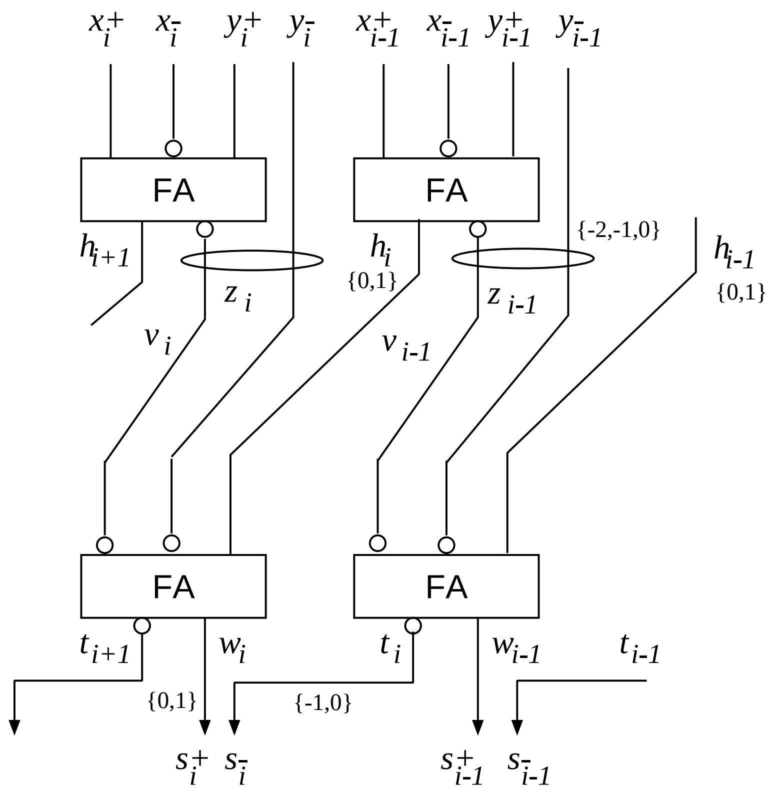


Figure 2.38: Redundant adder: operands and result redundant

SUMMARY

Scheme	Delay proportional to	Area proportional to
Linear structures:		
Carry ripple	n	n
Carry lookahead (one level)	n/m	$(k_m m)(n/m) = k_m n$
Carry select (one level)	n/m	$(k_m m)(n/m) = k_m n$
Carry skip (one level)	$\sqrt{n}$	n
Logarithmic structures:		
Carry lookahead (max. levels)	$2\log_m n$	$(k_m m)(n/m) = k_m n$
Prefix	$\log_m n$	$((k_m m)\log_m n)n$
Conditional sum	$\log_2(n/m)$	$(k_m + \log_2(n/m))n$
Completion signal (avg. delay)	$(\log_2 n)/m$	$k_m m(n/m) = k_m n$
Redundant	$const.$	n