DIGITAL ARITHMETIC
Miloš D. Ercegovac and Tomás Lang
with contributions by Elisardo Antelo and Fabrizio Lamberti
Morgan Kaufmann Publishers, an imprint of Elsevier, ©2004

Chapter 2: Solutions to Exercises

Exercise 2.1

Assuming that c_i is connected to the XOR input with load factor 1.1 (Fig. 2.5(c)), the average delay of the carry-out is

$$T_1 = t_{NAND}(1) + t_{NAND}(2.1) = 0.07 + 0.033 + 0.07 + 0.033 \times 2.1 = 0.242ns$$

Adding an inverter and changing the XOR into XNOR, we obtain for the carry delay:

$$T_2 = t_{NAND}(1) + t_{NAND}(2) = 0.239ns$$

This represents a 1.4% reduction in the carry delay. Note that the difference is very small because of the XOR input with load factor 1.1. A larger reduction would result if the XOR input load factors were symmetrical at 2.

Exercise 2.2

$T_{32} = max(t_{c-c}, t_{xy-c}) + 30t_{c-c} + max(t_{c-s}, t_{c-c})$ where, according to Table 2.2,

$t_{c-c} = 0.38 + 0.03(1.3) = 0.419ns$

where 1.3 is the load of the carry-out signal.

$t_{xy-c} = 0.72 + 0.03(1.3) = 0.759ns$

$t_{c-c_out} = 0.38$ and $t_{c-s_{31}} = 0.46$ ($L = 0$ for both outputs)

$T_{32} = 0.76 + 30 \times 0.419 + 0.46 \approx 13.8ns$

Exercise 2.3

A radix-4 full adder satisfies $x_i + y_i + c_i = 4c_{i+1} + z_i$ where $x_i,\ y_i,\ z_i \in \{0, 1, 2, 3\}$ and the carries are in {0,1}. The radix-4 digits are encoded on two binary variables as $x_i = (x_{i,1}, x_{i,0})$, etc. For simplicity the x, y, z binary variables are denoted as (x_1, x_0), etc.

The radix-4 full adder is defined by the following switching expressions.

$$\begin{aligned}
p_j &= x_j \oplus y_j;\ j = 0, 1\\
g_j &= x_j y_j;\ j = 0, 1\\
z_0 &= p_0 \oplus c_i\\
z_1 &= p_1 \oplus (g_0 + p_0 c_i) = p_0 \oplus g_0 \oplus p_0 c_i\\
c_{i+1} &= g_1 + p_1 g_0 + p_1 p_0 c_i
\end{aligned}$$

The radix-4 full adder and the radix-2 2-bit adder are compared with respect to delays in the carry-in – carry-out path, and with respect to circuit size. The following gates are used:

Type	Avg. delay	Eq. size
NOT	NA	1
NAND2	$0.07 + 0.033L$	1
NAND3	$0.08 + 0.039L$	2
XOR(XNOR)	NA	3

The carry path delays are:

$$\begin{aligned} T_4 &= t_{NAND3(L=1)} + t_{NAND3(L=3.1)} = 0.32ns \\ T_2 &= 2 \times (t_{NAND2(L=1)} + t_{NAND2(L=2.1)}) = 0.48ns \end{aligned}$$

where the loads are calculated as follows:

For $T4$: the second $NAND$ goes to c input of the next digit so $L = 1 + 1 + 1.1 = 3.1$

For $T2$: the output NAND goes to a NAND and to an XOR in the next bit-position so $L = 1 + 1.1 = 2.1$

The sizes in equivalent gates are:

$$\begin{aligned} S_4 &= 5 \times XOR(XNOR) + NOT + AND2 + 3 \times NAND2 + 2 \times NAND3 \\ &= 5 \times 3 + 1 + 2 + 3 \times 1 + 2 \times 2 = 25 \\ S_2 &= 2 \times (2 \times (NAND2 + XOR(XNOR)) + NAND2) \\ &= 2 \times (2 \times (1 + 3) + 1) = 18 \end{aligned}$$

Exercise 2.4

$T_{SRA} = t_{sw} + (n-1)t_p + (n/m)t_{buf} + t_s$ (Expression (2.27))

$t_{sw} = max(t_{gi}, t_{ki}, t_{pi}) + t_{NAND-2(L=2)} = t_{pi} + t_{NAND-2(L=2)} = 0.329 + 0.136 = 0.465ns$

where, assuming a switch has one standard load,

$$t_{gi} = t_{AND-2} = 0.16 + 0.027 \times 1 = 0.187ns$$

$$t_{ki} = t_{NOR-2} = 0.07 + 0.046 \times 1 = 0.116ns$$

$$t_{pi} = t_{XOR-2} = 0.30 + 0.029 \times 1 = 0.329ns$$

$t_p = t_{NAND-2} = 0.07 + 0.033 \times 2 = 0.136ns$ $(L = 2)$

$t_{buf} = 1.5 \times 0.136 = 0.204$

$t_s = 0.46 + 0.03 \times L = 0.46ns$ (Table 2.2, delay c_i to s_i with $L = 0$)

Therefore,

$T_{SRA} = 0.465 + 31 \times 0.136 + 8 \times 0.204 + 0.46 \approx 6.8ns$

From Exercise 2.2, $T_{CRA} = 13.8ns$ so the SRA aproximately halves the delay. Note that to reduce the load the network for computing the sum bits uses separately obtained p_i signals

Exercise 2.5

Figure E2.5 shows the carry chains for the given operands.

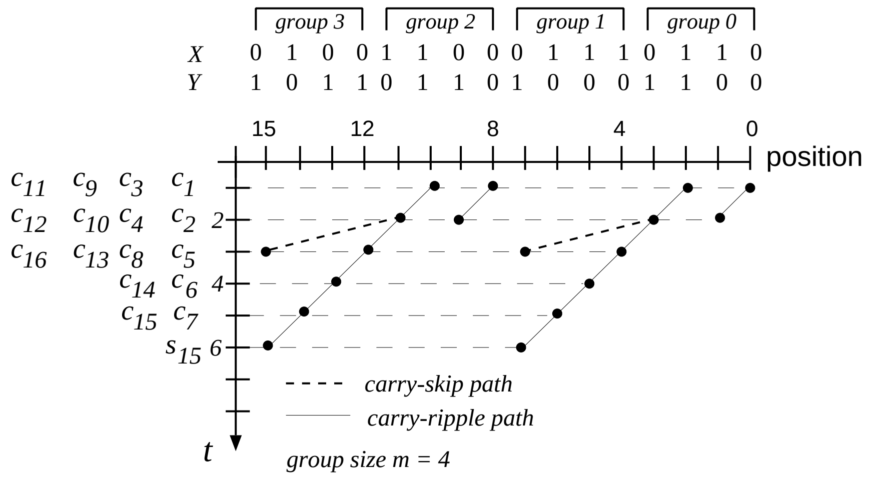


Figure E2.5: Carry chains in carry-skip adder (Exercise 2.5).

Exercise 2.6

We use the same assumption as in Exercise 2.2: $t_{pg} = t_c = t_{AND-OR} = 1$.
We assume $c_{in} = 0$

Operation 1	x	0000	0111	0000	1111
	y	1111	0000	1111	0001
	P	$P_3 = 1$	$P_2 = 0$	$P_1 = 1$	$P_0 = 0$
	c	00000	1111	1111	111-
Operation 2	x	0000	0111	0000	1111
	y	1111	0000	1111	0000
	P	$P_3 = 1$	$P_2 = 0$	$P_1 = 1$	$P_0 = 1$
	c	00000	0000	0000	000-

1. Although $P_0 = 1$ in second operation, it is necessary to wait for the propagation of the new carries in the first group ($t = 5$ to set $c_4 = 0$). Propagation through the OR adds 1 unit delay and, therefore, $t = 6$.

2. The same occurs in the second group. Although $P_1 = 1$, it is necessary to wait 4 units until $c_8 = 0$, so $t = 10$. Add 1 unit delay due to the OR gate and $t = 11$.

3. A correct carry is propagated to the most significant group. Since there are no changes in this group, there is no additional delay.

4. The carry chain propagates through the third group up to c_{11} ($t = 14$). There is one additional delay to produce s_{11} ($t = 15$).

Exercise 2.7

Expression (2.31) gives the worst-case delay of a carry-skip adder with fixed-size groups as

$$T_{CSK} = (2m-1)t_c + (n/m-1)t_{mux} + t_s$$

Differentiating this expression with respect to m and making the derivative equal to 0 we get

$$\frac{dT_{CSK}}{dm} = 2t_c - \frac{n}{m^2}t_{mux} = 0$$

resulting in

$$m_{opt} = (\frac{nt_{mux}}{2t_c})^{1/2}$$

and

$$T_{opt} \approx 2 \times (2nt_{mux}t_c)^{1/2} = (8nt_{mux}t_c)^{1/2}$$

Exercise 2.8

a) $m = 8$

$t_{CSKA} = (2m-1)t_c + (n/m-1)t_{MUX} + t_s$

From Table 2.2:

$t_c = 0.38 + 0.03 * L = 0.38 + 0.03(1.3) = 0.419ns$

$t_s = 0.46 + 0.03 * L = 0.46$ assuming $L = 0$

From Table 2.4:

$t_{MUX} = 0.21 + 0.050 * L = 0.21 + 0.050(1.8) = 0.3ns$ where $L = 0.5(MUX) + 1.3(c_{in}) = 1.8$

Therefore,

$t_{CSKA} = (15) * 0.419 + 7 * 0.3 + 0.46 = 6.285 + 2.1 + 0.46 = 8.85ns$

b) $m_{opt} = (\frac{t_{MUX}}{2t_c} \cdot n)^{1/2} = (\frac{0.3}{2 \cdot 0.419} \cdot 64)^{1/2} = 4.78$

To have a uniform block size which is a divisor of 64 and closest to m_{opt} we choose $m = 4$ (16 groups)

In this case, $t_{CSKA} = 7t_c + 15 * t_{MUX} + t_s = 7.9ns$.

If we allow a non-uniform group size and choose $m = 5$ which is the integer closest to m_{opt}, we have 12 groups of 5 bits and one of 4 bits:

$t_{CSKA} = 5t_c + t_{MUX} + 10 * t_{MUX} + 4t_c + t_s = 7.53ns$

c) Use, for instance, the following sequence: 445668866544. In this case the delay is:

$t_{CSKA} = 4t_c + t_{MUX} + 10 * t_{MUX} + 3t_c + t_s = 6.70ns$

Exercise 2.9

a) For a 9-bit adder we have two possible paths:

$b_1 \rightarrow b_2 \rightarrow S_1 \rightarrow b_3 \rightarrow S_1 \rightarrow b_6 \rightarrow b_7 \rightarrow b_8 = 8\delta$

$b_0 \rightarrow S_1 \rightarrow b_3 \rightarrow S_1 \rightarrow b_6 \rightarrow b_7 \rightarrow b_8 = 7\delta$

So, the worst-case delay is 8δ.

b) A 36-bit adder using blocks of 9-bit adders. The worst-case consists of the worst-case to obtain the carry-out of the least-significant group plus the maximum skip of intermediate groups plus the worst-case delay to obtain the sums in the most-significant group. That is,

```
[9-bit adder] <-- [9-bit adder] <-- [9-bit adder] <-- [9-bit adder]
worst-case sums     max-skip           max-skip        worst-case carry
```

Worst-case delay of carry-out of l.s. group: 4δ (several paths, i.e. b_4, b_5, MUX, MUX)

Delay of max-skip: $b_0 + MUX + MUX = 3\delta$

Worst-case sums (from c_{in} of group) $= 7\delta$.

Total delay: $7\delta + 3\delta + 3\delta + 4\delta = 17\delta$

Exercise 2.10

(a) $T = mt_c + (s-1)t_{mux} + (p-2)t_{mux} + (s-1)t_{mux} + (m-1)t_c + t_s$.

(b) Let $t_c = t_{mux}$ and $m = s$. $T = (4m - 3 + n/m^2)t_c + t_s$ and $m_{opt} = (n/2)^{1/3}$.

Exercise 2.11

a) Cost and delay of $CLG - 4$ $(m = 4)$

a) Since there are two passes through the module we design so as to reduce the critical path of both passes.

i) First pass:

The critical path goes from signals a,g to A,G. From Figure 2.14 we see that the critical output is G. We implement as follows:

$$G = g3 + a3g2 + a3a2g1 + a3a2a1g0 = [g3'.(a3g2)'.(a3a2g1)'.(a3a2a1a0)']'$$

Since G is the input to the next level module, we buffer the output of the NAND-4 with two NOT gates.

Moreover,

$$A = a3a2a1a0 = ((a3a2a1a0)')'$$

The other parts of the "upper" network are implemented in a similar way.

ii) For the second pass the critical path is c0 to c4. We implement is as

$$c4 = G + Ac0 = (G'.(Ac0)')'$$

The other carries are implemented in a similar way

So,

maximum fanin = 4

maximum fanout = 7 (this corresponds to output A, which goes to a1 of another module).

Equivalent gates:

NOT 1 (g3) + 1(G) + 1 (A) + 4(Gi, for ci) + 8 (ci) = 15 EQ=15

NAND-2 13 EQ=13

NAND-3 4 EQ=8

NAND-4 3 EQ=6

TOTAL EQ = 42

critical path (pass 1) $t_{a1,G}$ = NAND4(1) + NAND4(1) + NOT(1) + NOT(5) = 0.16+0.16+0.07+0.18 = 0.57

critical path (pass 2) $t_{c0,c4}$ = NAND2(1) + NAND2(1) + NOT(1) + NOT(4) = 0.10+ 0.10+ 0.07+0.15 = 0.42

b) Cost and delay of $CLG - 8$ $(m = 8)$. Following the same implementation approach of part a) we get

maximum fanin = 8

maximum fanout = 2x7 + 1 = 15

Equivalent gates

NOT = 1+1+1+8+16 = 27 EQ=27

NAND2 = 25 EQ=25

NAND3 = 8 EQ=16

NAND4 = 7 EQ=14

NAND5 = 6 EQ=24

NAND6 = 5 EQ=25

NAND7 = 4 EQ=24

NAND8 = 3 EQ=18

TOTAL= 173

critical path (pass 1) $t_{a1,G}$ = NAND8(1) + NAND8(1) + NOT(1) + NOT(9) = 0.36+0.36+0.07+0.29 = 1.08

critical path (pass 2) $t_{c0,c4}$ = NAND2(1) + NAND2(1) + NOT(1) + NOT(8) = 0.10+0.10+0.07+0.26 = 0.53

As an illustration consider a 64-bit adder using 4-bit modules and 8-bit modules. In the 4-bit module case, a 3-level CLA is used. The delay corresponds to

t(a,g) + 2t(G) + 3t(c) + ts = 0.32 + 1.14 + 1.26 + 0.15 = 2.87

where t(a,g) is the delay of NOR2(1)+NOT(6)= 0.12 + 0.21 = 0.32 and ts is delay of XOR(2) with L=0.

In the 8-bit module case, a 2-level CLA is used. The delay is

t(a,g) + t(G) + 2t(c) +ts = 0.39 + 1.08 + 1.06 + 0.15 = 2.68

where t(a,g) is NOR3(1)+NOT(5) = 0.21 + 0.18 = 0.39 (use 3 NOT gates in parallel for the load of 14)

Exercise 2.12

The 32-bit adder consists of 32 PG modules to produce p's and g's, 8 BCLA modules to produce carries $c_{4i+4},\ i = 0, \ldots, 7$, and 8 4-bit CRA-4 modules to produce the sum bits. Assuming the gates of Table 2.4, that a full-adder is implemented using the design of Figure 2.3(c), the delays of these modules are:

Module PG (XOR,AND): $t_{PG} = 0.30 + 0.029 \times 4 = 0.42ns$

Module CRA-4 (4 FAs): $t_{CPA-4} = 2t_{XOR} + 6t_{NAND} = 0.672 + 6 \times 0.107 = 1.31ns$

Module BCLA (NAND-NAND): $t_{BCLA} = 0.359 + 0.34 + 0.019 \times 2 = 0.74$

The worst case delay is

$$T_{sum} = 0.42 + 7 \times 0.74 + 1.31 = 6.91ns$$

$$T_{c_{out}} = 0.42 + 8 \times 0.74 = 6.34ns$$

A 32-bit single-level carry-skip adder, with group size 4, has the following worst-case delay: $T_{CSK} = 7 \times t_c + 7 \times t_{mux} + t_s = 7 \times 2 \times 0.107 + 7 \times 0.272 + 0.336 = 3.74ns$

This indicates that the BCLA 32-bit adder is slower 1.85 times than the carry-skip alternative. Moreover, the BCLA module uses more gates than the skip network in the 32-bit carry-skip adder with $m = 4$. In terms of equivalent gates, this cost ratio is about 3.

Exercise 2.13

The g_i and a_i signals are

x	0	1	0	1
y	1	0	0	1
g_i	0	0	0	1
a_i	1	1	0	1

The expressions and values for the CLG-4 carries are

$$\begin{aligned}
c_0 &= 1 \\
c_1 &= g_0 + a_0c_0 = 1 + 1 \cdot 1 = 1 \\
c_2 &= g_1 + a_1g_0 + a_1a_0c_0 = 0 + 0 \cdot 1 + 0 \cdot 1 \cdot 1 = 0 \\
c_3 &= g_2 + a_2g_1 + a_2a_1g_0 + a_2a_1a_0c_0 = 0 + 1 \cdot 0 + 1 \cdot 0 \cdot 1 + 1 \cdot 0 \cdot 1 \cdot 1 \cdot 1 = 0 \\
c_4 &= g_3 + a_3g_2 + a_3a_2g_1 + a_3a_2a_1g_0 + a_3a_2a_1a_0c_0 \\
&= 0 + 1 \cdot 0 + 1 \cdot 1 \cdot 0 + 1 \cdot 1 \cdot 0 \cdot 1 + 1 \cdot 1 \cdot 0 \cdot 1 \cdot 1 = 0
\end{aligned}$$

Exercise 2.14

X	0 1 1 0	1 1 1 0	1 0 1 1	1 0 0 1	
Y	1 0 0 1	0 0 0 0	1 1 1 0	0 1 0 1	
g_i	0 0 0 0	0 0 0 0	1 0 1 0	0 0 0 1	
a_i	1 1 1 1	1 1 1 0	1 1 1 1	1 1 0 1	
G	0	0	1	0	
A	1	0	1	0	
	c_{16}	c_{12}	c_8	c_4	c_0
	0	0	1	0	0
$c_i,\ i \neq 16, 12, 8, 4, 0$	0 0 0	0 0 0	1 1 0	0 0 1	

Exercise 2.15

A 64-bit, three-level carry-lookahead adder is shown in Figure E2.15.

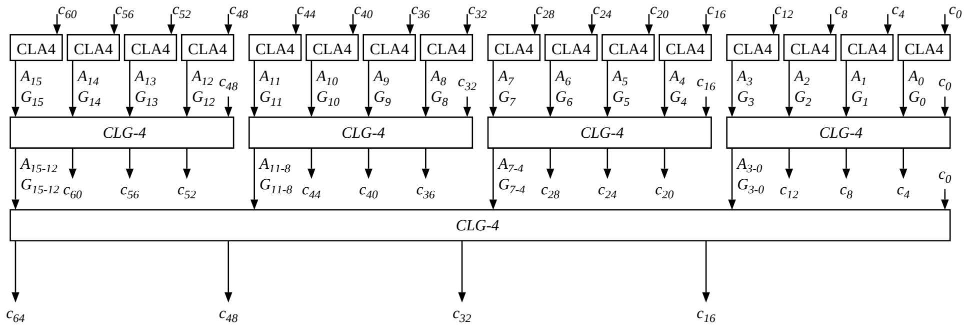

c_{60} c_{56} c_{52} c_{48} c_{44} c_{40} c_{36} c_{32} c_{28} c_{24} c_{20} c_{16} c_{12} c_8 c_4 c_0
CLA4
A_{15} G_{15} A_{14} G_{14} A_{13} G_{13} A_{12} G_{12} c_{48}
A_{11} G_{11} A_{10} G_{10} A_9 G_9 A_8 G_8 c_{32}
A_7 G_7 A_6 G_6 A_5 G_5 A_4 G_4 c_{16}
A_3 G_3 A_2 G_2 A_1 G_1 A_0 G_0 c_0
CLG-4
$A_{15\text{-}12}$ $G_{15\text{-}12}$ c_{60} c_{56} c_{52}
$A_{11\text{-}8}$ $G_{11\text{-}8}$ c_{44} c_{40} c_{36}
$A_{7\text{-}4}$ $G_{7\text{-}4}$ c_{28} c_{24} c_{20}
$A_{3\text{-}0}$ $G_{3\text{-}0}$ c_{12} c_8 c_4 c_0
CLG-4
c_{64} c_{48} c_{32} c_{16}

Figure E2.15: 64-bit three-level carry-lookahead adder.

The critical path is

$$(x_0, y_0) \to (A_0, G_0) \to (A_{3-0}, G_{3-0}) \to c_{48} \to c_{60} \to (A_{15}, G_{15}) \to (A_{15-12}, G_{15-12}) \to c_{64}$$

Exercise 2.16

The number of CLGs is divided by m from one level to the next. Therefore, at level i there are n/m^i CLGs. The total number is obtained by adding from $i = 1$ to L.

Exercise 2.17

$n = 128$, $m = 4$, $t_{clg} = t_{AG} = 6t_{ag} = 3t_s$

$T_{1-CLA} = t_{ag} + (n/m)t_{clg} + t_s = 1 + 32 \times 6 + 2 = 195t_{ag}$

$T_{2-CLA} = t_{ag} + t_{AG} + (n/m^2)t_{clg} + t_{clg} + t_s = 1 + 6 + 8 \times 6 + 6 + 2 = 63t_{ag}$

$T_{3-CLA} = t_{ag} + 2t_{AG} + (n/m^3)t_{clg} + 2t_{clg} + t_s = 1 + 12 + 12 + 12 + 2 = 39t_{ag}$

For the 4-level CLA we use another level with a group size of 2. Because of the smaller size of this group the delay of this level is smaller, we assume it to be $t_{clg2} = 2t_{a,g}$.

$$T_{4-CLA} = t_{ag} + 2t_{AG4} + t_{clg2} + 3t_{clg} + t_s = 1 + 12 + 2 + 18 + 2 = 35t_{ag}$$

Exercise 2.18

a) From the definition $h_i = c_i + c_{i+1} = g_i + p_i c_i + c_i = g_i + c_i$. Moreover, $s_i = t_i \oplus h_i + g_i t_{i-1} h_{i-1}$.

1st term: $t_i \oplus h_i = t_i' h_i + t_i h_i' = (x_i + y_i)'(x_i y_i + c_i) + (x_i + y_i)(x_i y_i + c_i)' = x_i' y_i' c_i + (x_i \oplus y_i) c_i'$

2nd term: $g_i t_{i-1} h_{i-1} = g_i(x_{i-1} + y_{i-1})(x_{i-1} y_{i-1} + c_{i-1}) = g_i(g_{i-1} + p_{i-1} c_{i-1}) = g_i c_i = x_i y_i c_i$

Therefore,

$$\begin{aligned} s_i &= x_i' y_i' c_i + (x_i' y_i + x_i y_i') c_i' + x_i y_i c_i \\ &= x_i' y_i' c_i + x_i' y_i c_i' + x_i y_i' c_i' + x_i y_i c_i \\ &= c_i'(x_i \oplus y_i) + c_i(x_i \oplus y_i)' = c_i \oplus p_i \end{aligned}$$

b) Note that $t_i g_i = (x_i + y_i)(x_i y_i) = x_i y_i = g_i$. The switching expressions for the carries in a 4-bit Ling adder we get

$$\begin{aligned} h_0 &= g_0 + c_{in} \\ h_1 &= g_1 + g_0 + t_0 c_{in} \\ h_2 &= g_2 + g_1 + t_1 g_0 + t_1 t_0 c_{in} \\ h_3 &= (g_3 + g_2) + t_2 g_1 + t_2 t_1 g_0 + t_2 t_1 t_0 c_{in} \end{aligned}$$

Assuming a two-level implementation these expressions require the following gates: 2 AND-2, 2 AND-3, 1 AND-4, 2 OR-2, 1 OR-3, 2 OR-4. The largest fan-in is 4.

The corresponding expressions for a conventional CLA described in the text require more gates and a larger fan-in: 4 AND-2, 3 AND-3, 2 AND-4, 1 AND-5, 1 OR-2, 1 OR-3, 1 OR-4, and 1 OR-5. The largest fan-in is 5.

Exercise 2.19

The 8-bit prefix adder without a carry-in is shown in Figure E2.19.

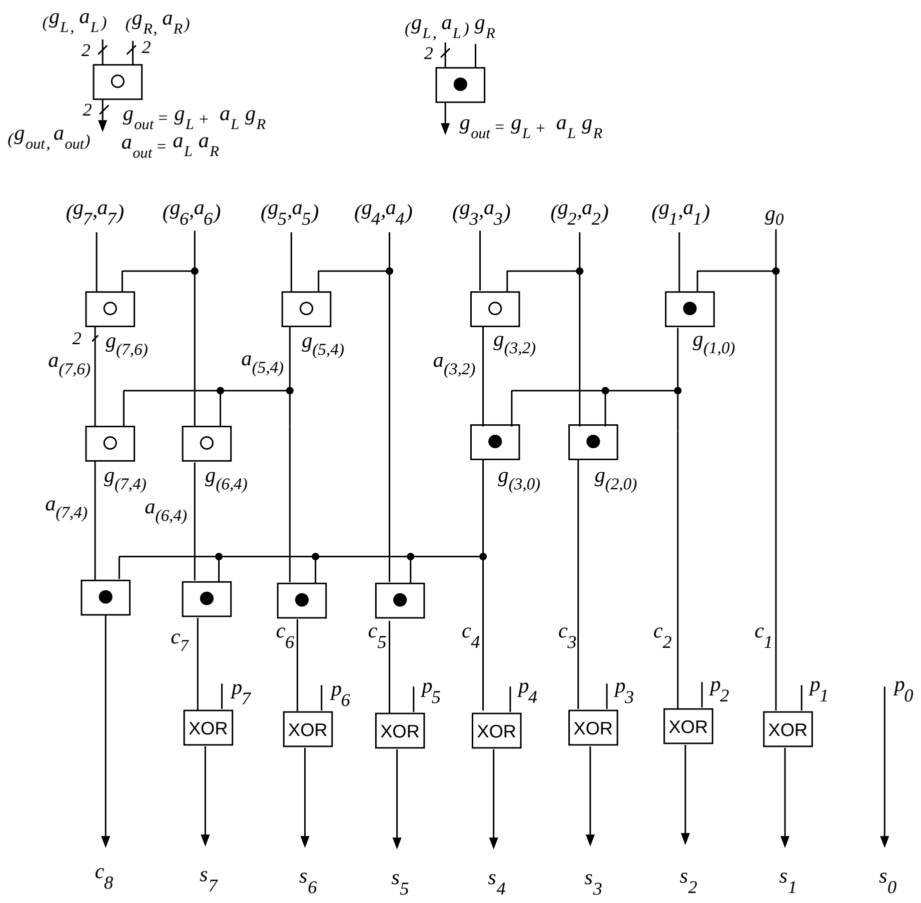

(g_L, a_L) (g_R, a_R)
$g_{out} = g_L + a_L g_R$
$a_{out} = a_L a_R$
(g_{out}, a_{out})
(g_L, a_L) g_R
$g_{out} = g_L + a_L g_R$
(g_7,a_7) (g_6,a_6) (g_5,a_5) (g_4,a_4) (g_3,a_3) (g_2,a_2) (g_1,a_1) g_0
$g_{(7,6)}$ $a_{(7,6)}$ $g_{(5,4)}$ $a_{(5,4)}$ $g_{(3,2)}$ $a_{(3,2)}$ $g_{(1,0)}$
$g_{(7,4)}$ $a_{(7,4)}$ $g_{(6,4)}$ $a_{(6,4)}$ $g_{(3,0)}$ $g_{(2,0)}$
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
c_8 s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

Figure E2.19: Prefix adder for Exercise 2.19.

Exercise 2.20

i	8	7	6	5	4	3	2	1	0
x_i		0	1	0	1	0	1	1	1
y_i		1	1	1	0	0	1	1	1
g_i		0	1	0	0	0	1	1	1
a_i		1	1	1	1	0	1	1	1
p_i		1	0	1	1	0	0	0	0

Level 1 outputs:

$$\begin{aligned}
g_{(0,-1)} &= 1 = c_1 \\
g_{(1,0)} &= g_1 + a_1 g_0 = 1, \quad a_{(1,0)} = a_1 a_0 = 1 \\
g_{(2,1)} &= g_2 + a_2 g_1 = 1, \quad a_{(2,1)} = a_2 a_1 = 1 \\
g_{(3,2)} &= g_3 + a_3 g_2 = 0, \quad a_{(3,2)} = a_3 a_2 = 0 \\
g_{(4,3)} &= g_4 + a_4 g_3 = 0, \quad a_{(4,3)} = a_4 a_3 = 0 \\
g_{(5,4)} &= g_5 + a_5 g_4 = 0, \quad a_{(5,4)} = a_5 a_4 = 1 \\
g_{(6,5)} &= g_6 + a_6 g_5 = 1, \quad a_{(6,5)} = a_6 a_5 = 1 \\
g_{(7,6)} &= g_7 + a_7 g_6 = 1, \quad a_{(7,6)} = a_7 a_6 = 1
\end{aligned}$$

Level 2 outputs:

$$\begin{aligned}
g_{(1,-1)} &= g_{(1,0)} + a_{(1,0)} c_0 = 1 = c_2 \\
g_{(2,-1)} &= g_{(2,1)} + a_{(2,1)} g_{(0,-1)} = 1 = c_3 \\
g_{(3,0)} &= g_{(3,2)} + a_{(3,2)} g_{(1,0)} = 0, \quad a_{(3,0)} = a_{(3,2)} a_{(1,0)} = 0 \\
g_{(4,1)} &= g_{(4,3)} + a_{(4,3)} g_{(2,1)} = 0, \quad a_{(4,1)} = a_{(4,3)} a_{(2,1)} = 0 \\
g_{(5,2)} &= g_{(5,4)} + a_{(5,4)} g_{(3,2)} = 0, \quad a_{(5,2)} = a_{(5,4)} a_{(3,1)} = 0 \\
g_{(6,3)} &= g_{(6,5)} + a_{(6,5)} g_{(4,3)} = 1, \quad a_{(6,3)} = a_{(6,5)} a_{(4,3)} = 0 \\
g_{(7,4)} &= g_{(7,6)} + a_{(7,6)} g_{(5,4)} = 1, \quad a_{(7,4)} = a_{(7,6)} a_{(5,4)} = 1
\end{aligned}$$

Level 3 outputs:

$$\begin{aligned}
c_4 &= g_{(3,0)} + a_{(3,0)} c_0 = 0 \\
c_5 &= g_{(4,1)} + a_{(4,1)} g_{(0,-1)} = 0 \\
c_6 &= g_{(5,2)} + a_{(5,2)} g_{(1,-1)} = 0 \\
c_7 &= g_{(6,3)} + a_{(6,3)} g_{(2,0)} = 1
\end{aligned}$$

Level 4 outputs:

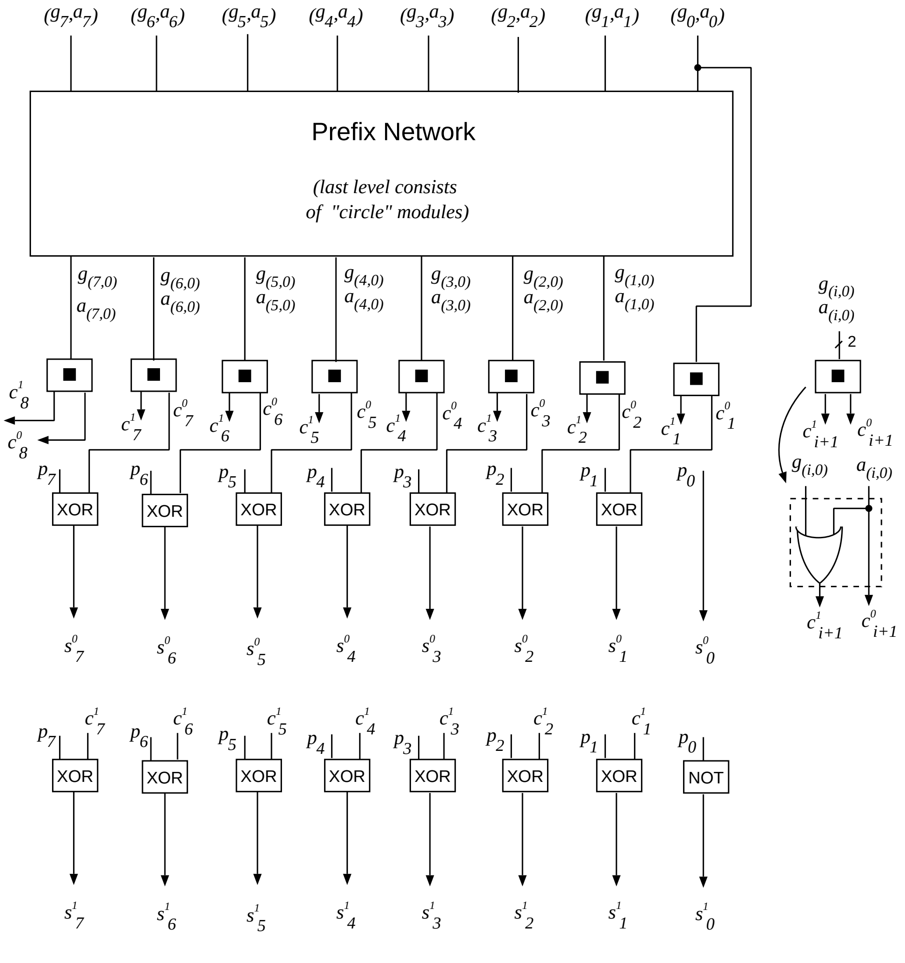
$$\begin{aligned}
s_0 &= p_0 \oplus c_0 = 1 \\
s_1 &= p_1 \oplus c_1 = 1 \\
s_2 &= p_2 \oplus c_2 = 1 \\
s_3 &= p_3 \oplus c_3 = 1 \\
s_4 &= p_4 \oplus c_4 = 1 \\
s_5 &= p_5 \oplus c_5 = 1 \\
s_6 &= p_6 \oplus c_6 = 0 \\
s_7 &= p_7 \oplus c_7 = 0 \\
c_8 &= g_{(7,0)} + a_{(7,0)} c_0 = 1
\end{aligned}$$

Exercise 2.21

The prefix adder shown in Figure E2.21 produces $x + y$ and $x + y + 1$.

As before, for $x + y$ the carry is $c_i^0 = g_{(i-1,0)}$. On the other hand, for the $x+y+1$ case, the carry c_i is 1 if a carry is generated in bits 0 to i-1 ($g_{(i-1,0)} = 1$) and if a carry is propagated from bit 0 to i-1 ($p_{(i-1,0)} = 1$). Consequently,

$$c_i^1 = g_{(i-1,0)} + p_{(i-1,0)} = g_{(i-1,0)} + a_{(i-1,0)}$$



Prefix Network
(last level consists of "circle" modules)
XOR
NOT
outputs with superscript 0 correspond to x+y
outputs with superscript 1 correspond to x+y+1

Figure E2.21: Prefix adder for Exercise 2.21.

Exercise 2.22

A diagram of a 16-bit prefix adder is shown in Figure E2.22.

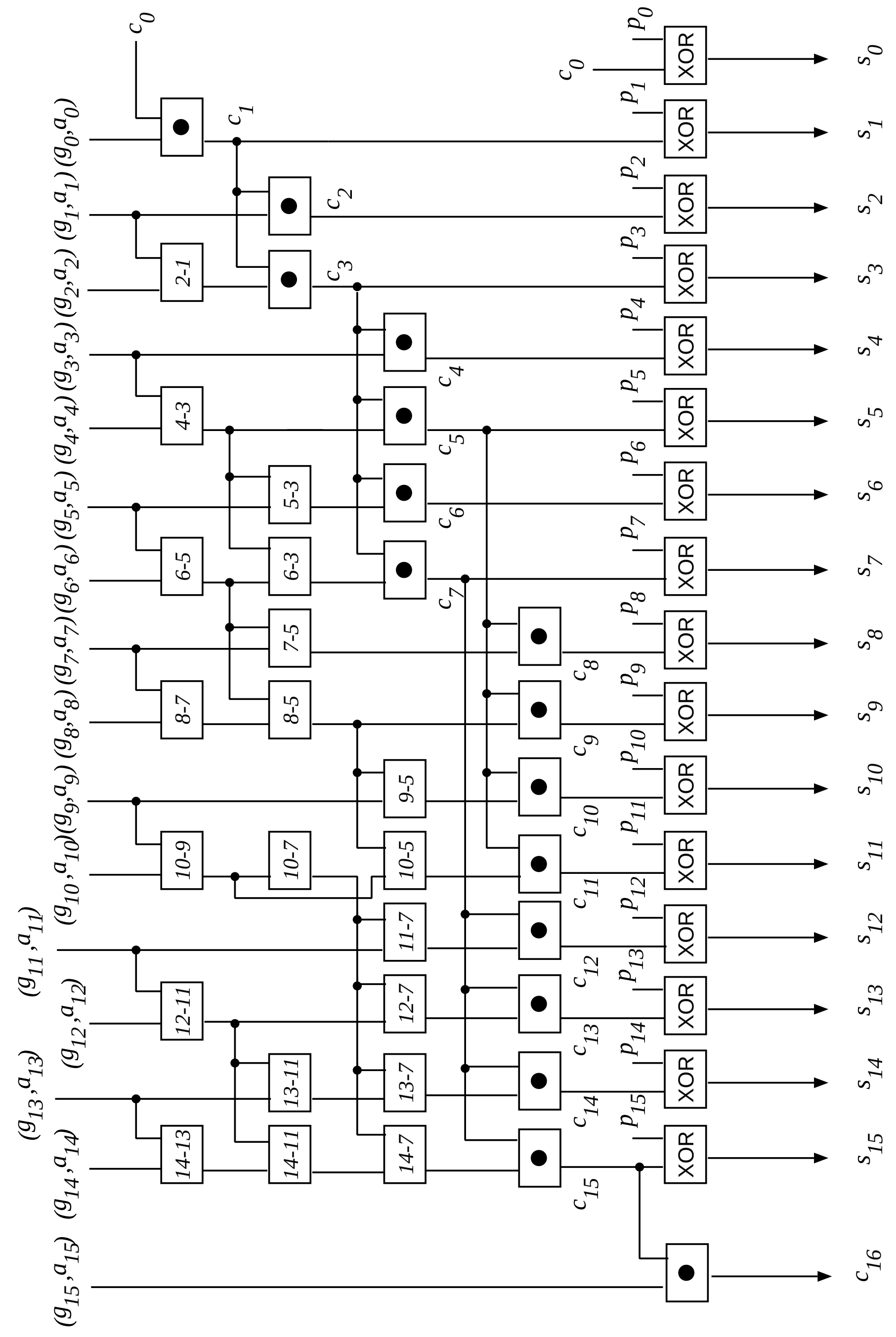


Figure E2.22: Prefix adder for Exercise 2.22.

Exercise 2.23

A diagram of a 4-bit conditional-adder module is shown in Figure E2.23.

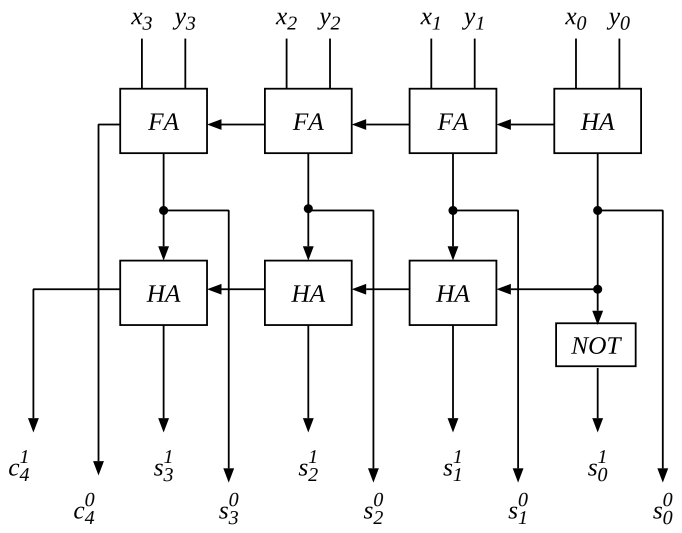


Figure E2.23: 4-bit conditional adder for Exercise 2.23.

Exercise 2.24

X	0111	1000	1010	1010
Y	1010	1011	1011	0010
(c^1, S^1)	10010	10100	10110	01101
(c^0, S^0)	10001	10011	10101	01100
(c, S)	10010	0100	0101	1100

Exercise 2.25

a) Assume delay of a k-bit CRA to be $k\delta$. The optimum size for each adder is obtained by requiring that the carry out is complete just at the time the decision signal arrives. That is, this condition is

$$tk(i) = k(i)\delta = tk(0) + (i-1)t_{MUX} \tag{1}$$

where $tk(i)$ is the delay of adder i.

Using m adders, the delay of the n-bit adder is

$$T_a = tk(m-1) + t_{MUX} = tk(0) + (m-1)t_{MUX} = k(0)\delta + (m-1)t_{MUX}$$

The total length of adders must satisfy

$$\sum_{i=0}^{m-1} k(i) = n \tag{2}$$

Substituting (1) in (2) we get

$$\begin{aligned} n &= \textstyle\sum_{i=0}^{m-1} k(i) = \frac{1}{\delta}\sum_{i=0}^{m-1}(k(0)\delta + (i-1)t_{MUX}) \\ &= \textstyle\sum_{i=0}^{m-1}(k(0) + (i-1)\frac{t_{MUX}}{\delta}) \\ &= mk(0) + \frac{(m-1)(m-2)}{2}(\frac{t_{MUX}}{\delta}) \end{aligned}$$

From this expression we obtain $k(0)$ in terms of m:

$$k(0) = \frac{n - \frac{(m-1)(m-2)}{2}(\frac{t_{MUX}}{\delta})}{m}$$

The delay of the n-bit adder in terms of m is

$$T_a(m) = k(0)\delta + (m-1)t_{MUX} = \frac{n\delta - \frac{(m-1)(m-2)t_{MUX}}{2}}{m} + (m-1)t_{MUX}$$

From $dT_a(m)/dm = 0$ we obtain the following equation

$$(m-1)^2 - (m-1) - \frac{2}{3}(n\frac{\delta}{t_{MUX}} + 2) = 0$$

Solving for m we get

$$m = \frac{2}{3} + (\frac{1}{4} + \frac{2}{3}(n\frac{\delta}{t_{MUX}} + 2)^{1/2}$$

For large n and $t_{MUX} = \delta$ we have

$$m \approx \lceil(\frac{2}{3}(n+2))^{1/2} + \frac{3}{2}\rceil$$

After obtaining m we calculate $k(0)$ and then $k(i)$'s. For example, for $n = 64$ we get $m = 9$ and

$$k(0) = 4, \quad k(i) = k(0) + i - 1, \quad i = 1, \ldots, 8$$

The delay is $T_a(64) = 4\delta + (9-1)t_{MUX} = 12\delta$.

b) Since in this case all carries are computed in log time, the size of the group is not very significant in the delay of the carry. Consequently, to simplify the derivation we assume that the output carries of each block are produced in constant time $c\delta$ and that the block size is constant (k) so that for a n-bit adder there are $m = n/k$ blocks. The delay for the sum bits of each block is $k\delta$.

The minimum delay is achieved when the delay of the sum of the last block is equal to the delay of the select signal obtained from previous blocks:

$$c\delta + (m-2)t_{MUX} = k\delta$$

For $t + MUX = \delta$ we get $c + m - 2 = k$. Since $n = mk$

$$m = \frac{2 - c + \sqrt{4n + (c-2)^2}}{2}$$

The total delay is $T = c\delta + (m-1)t_{MUX} = (c + m - 1)\delta$. For example, for $n = 64$ and $c = 2$, we get $m = 8$ and $k = 8$. The delay in this case is $T = (2 + 7)\delta = 9\delta$.

Exercise 2.26

X	01	01	01	11
Y	10	10	11	11
S^0	11	11	00	10
c^0	0	0	1	1
S^1	00	00	01	11
c^1	1	1	1	1
S^0	11	11	01	10
c^0	0		1	
S^1	00	00	01	11
c^1	1		1	
S^0	00	00	01	10
c^0	1			
S^1	00	00	01	11
c^1	1			

The result is (c^0, S^0) because $c_{in} = 0$.

Exercise 2.27

Scheme A: $T_A = t_{CRA}(n/4) + 2t_{MUX} = (n/4) \times 2\delta + 2 \times \delta = (2^{p-1} + 2)\delta$

Scheme B: $T_B = t_{CA} + 4t_{MUX} = (2\delta + (p-3)\delta) + 4\delta = (p+3)\delta$

- $T_A < T_B$ if $2^{p-1} + 2 < p + 3$ which holds for $p \leq 2$.

Exercise 2.28

(a) The initial conditional carries are obtained by CC-1 modules: $c^0_{i+1} = x_i y_i$, $c^1_{i+1} = x_i + y_i$ (Figure E2.28a). The stages are composed of modules organized as shown in Figure E2.28b and c. An Mk module consists of $2k$ 2-input MUXes.

A 16-bit conditional-carry adder, shown in Figure E2.28c, consists of the following stages:

Stage 0: obtain (c^1_i, c^0_i) using 16 (AND,OR) modules with delay t_g and $p'_i s$ with 16 XORs (t_{XOR} with delay $2t_g$.)

Stage 1: obtain conditional-carry bit-vectors of size 2: $7 \times 2 + 1 = 15$ MUXes ($2t_g$)

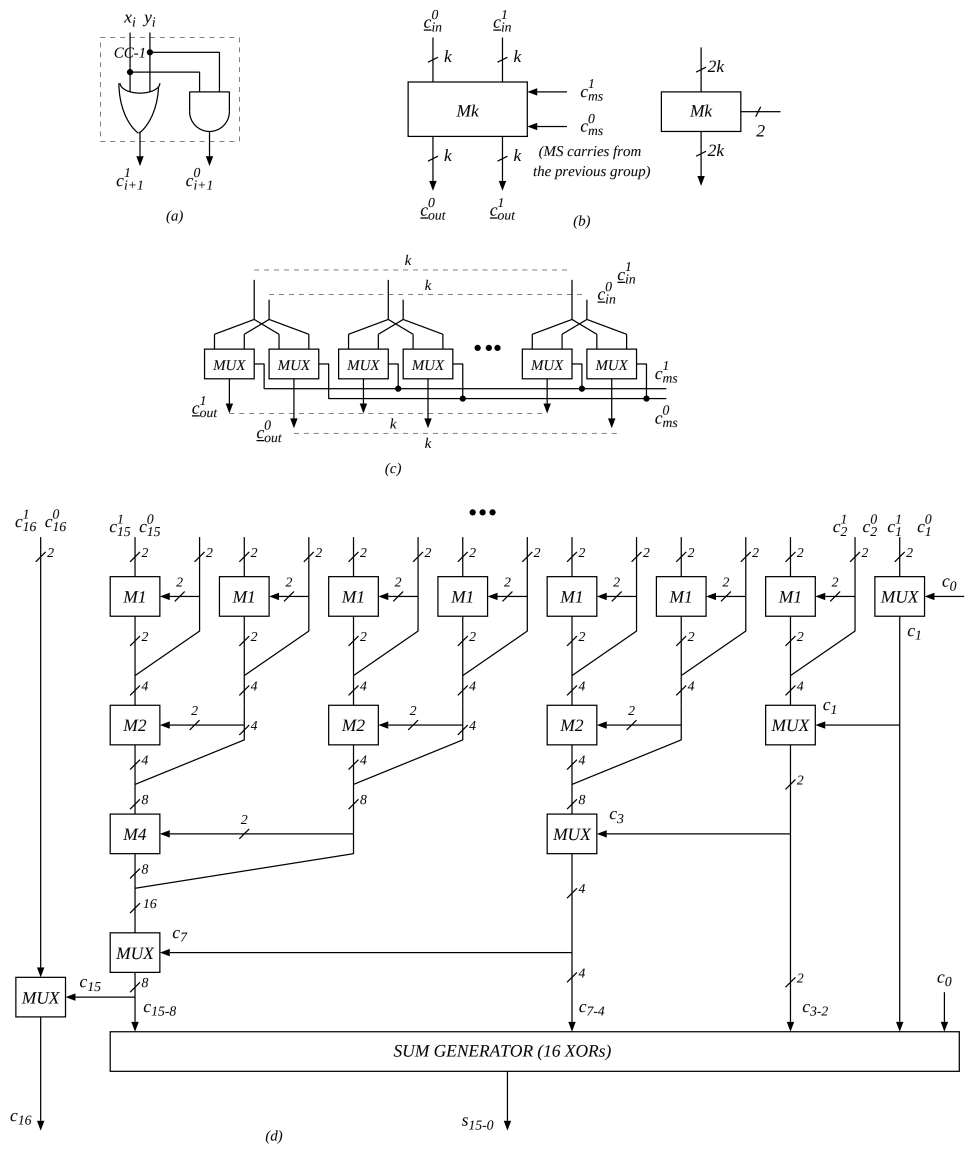


Figure E2.28: (a) Generation of conditional carries. (b+c) Symbols and implementation of MUX-based modules for conditional carries. (d) 16-bit conditional-carry adder. (Exercise 2.28).

Stage 2: obtain conditional-carry bit-vectors of size 4: $3 \times 4 + 2 = 14$

Stage 3: obtain conditional-carry bit-vectors of size 8: $1 \times 8 + 4 = 12$

Stage 4: obtain conditional-carry bit-vectors of size 8: $8 \times 1 = 8$

Stage 5: obtain the sum: $16\times$ XOR: total 16; $t_{XOR} = 2t_g$

The delay: $T_{CC-16} = t_g + 4 \times 2t_g + 2t_g = 11t_g$
The cost: 16 AND-2 + 16 OR-2 + 2×16 XOR + 49 MUXes + 2 AND-3 + OR-2.
From Table 2.4:
Total: $32 \times 2 + 32 \times 3 + 49 \times 2 + 3 \times 2 = 264$ eqv. gates

(b) Using the conditional-sum adder of Fig.2.24. The COND-ADD-4 is made of modified CLA-4 module: in the CLG-4 we duplicate the AND-OR subnetwork at the bottom to generate conditional carries; we also in the CLA-4 we duplicate XORs to get conditional sums.
The cost of the COND-ADD-4: 4 AND-2 + 4 OR-2 + 3×4 XOR + CLG-4 + 4 AND-2 + 4 OR-2 = $8 + 8 + 12 \times 3 + 44 + 8 + 8 = 112$ eqv. gates.
The delay: $T_{CSUM-16} = t_{COND-ADD-4} + 2 \times 2t_g = 5t_g + 4t_g = 9t_g$
The cost: 3 COND-ADD-4 + CLA-4 + $(3 \times 5 + 9)$ MUXes. Total: $3 \times 112 + 84 + 24 \times 3 = 492$ eqv.gates
The conditional-carry adder has a similar delay and a significantly lower cost than a conditional-sum adder with 4-bit conditional adders.

Exercise 2.29

a) Type 1 adder:

x	1000	100	111
y	0111	000	110
c_i^0	11111	110	011
c_i^1	00000	001	100
c_i	00000	001	100
s_i	01111	101	101

The actual delay, assuming critical path in producing F, is

$$T_{Type1} = t_{XOR} + t_{OR-2} + 10 \times t_c + t_{OR-2}$$

where t_c is the delay of producing a carry:

$$t_c = t_{AND-2} + t_{OR-2}$$

Given that t_c has the same expression for the carry-ripple adder and that the actual delay of t_c is 15% smaller than its worst-case delay and assuming the same variation for t_{XOR} and t_{OR-2}, we get:

$$T_{Type1} \approx 0.85 T_{CRA}$$

b) Type 2 adder:

x	1000100111
y	0111000110
chains	jihgfedcba
timing	6543211111

In this example, the longest chain is zero-carry chain efghij of 6 positions.

The actual delay is

$$T_{Type2} = t_{XOR} + t_{max} + t_{OR-2} + t_{AND-10}$$

where $t_{max} = 6t_c$.

Consequently, including the delay of AND-10, for this input pattern the addition delay is rougly 70% of that of the adder of type I.

Exercise 2.30

a) Figure E2.30 shows an implementation of a 32-bit carry-select adder using self-timed modules: 8-bit Type-1 adders and bit-vector multiplexers.

The delay of this carry-select self-timed adder is

$$T_{CSST-32} = t_{XOR} + t_{OR} + \sum_{i=0}^{7} tc_i + \sum_{i=1}^{3}(t_{AND} + t_{OR}) + t_{OR} \approx 12tc$$

where tc is the average carry delay and $t_{XOR} + t_{OR} + t_{OR} \approx 2tc$.

b) The delay of the 32-bit self-timed adder based on a carry-ripple adder (Fig. 2.27) is

$$T_{CRST-32} = t_{XOR} + t_{OR} + \sum_{i=0}^{31} tc_i + t_{OR} \approx 33tc$$

Consequently, we estimate the variable-time adder based on the carry-select scheme to be about 2.75 times faster than the one based on the carry-ripple adder.

Comparing Figure E2.30 with Figure 2.27 shows that the number of adder modules is multiplied by 1.75. Including the multiplexers, we estimate the cost to be 2.5 times higher.

Exercise 2.31

The critical path in Fig. 2.20 is

$$g_0 \rightarrow g_{(1,0)} \rightarrow g_{(3,0)} \rightarrow g_{(7,0)} \rightarrow g_{(7,-1)} \rightarrow g_{(0,-1)} \rightarrow g_{(2,-1)} \rightarrow c_7 \rightarrow s_7$$

Assuming delay t per (g,a) module, the delay is

$$T_{Fig.2.20} = 8t + t_{XOR} \approx 9t$$

The critical path in Fig.2.29 is

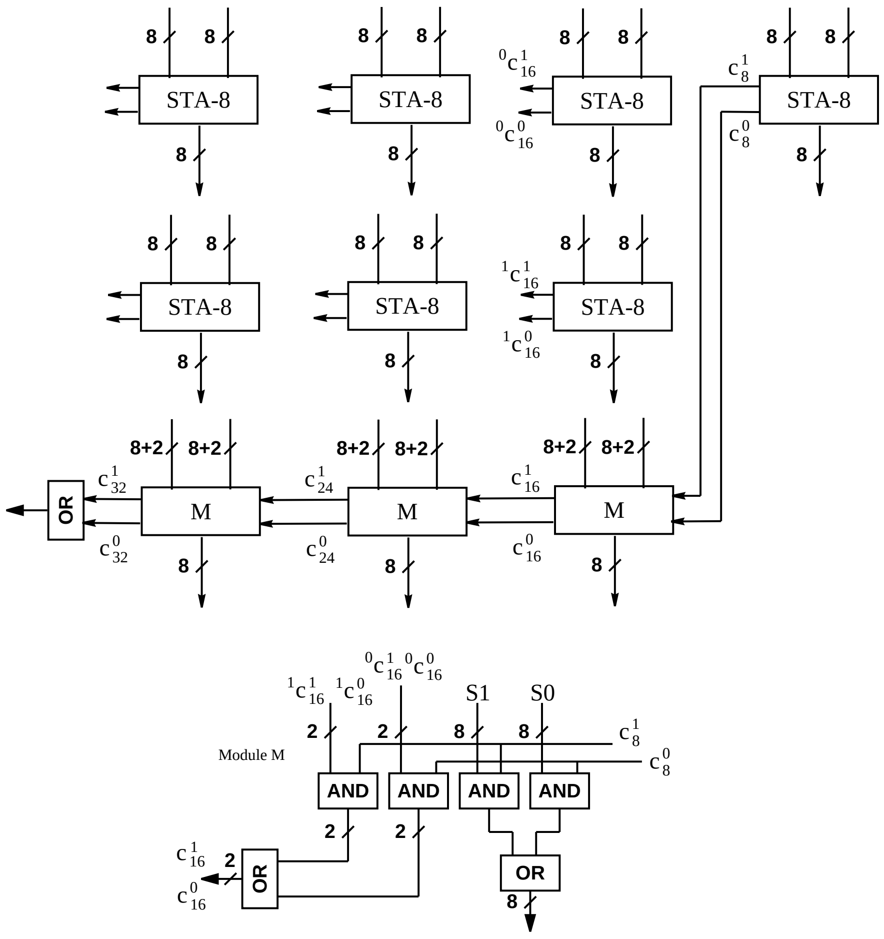


Figure E2.30: Carry-select adder with self-timed modules (Exercise 2.30).

$$g_0 \rightarrow g_{(1,0)} \rightarrow g_{(3,0)} \rightarrow g_{(7,0)} \rightarrow c_7 \rightarrow s_7$$

Assuming delay t per (g,a) module, the delay is

$$T_{Fig.2.29} = 3t + t_{buff} + t + t_{XOR} \approx 6t$$

Exrecise 2.32

a)

X		0	1	1	1	1	1	0	0	0	0	1	1	0	0	1	1
Y		1	1	1	0	0	0	1	0	0	1	1	0	1	1	0	1
W		1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
$S*$		0	0	1	1	0	1	0	0	1	1	1	1	0	1	0	0
$C*$	1	1	1	0	1	0	1	0	0	0	1	0	1	0	1	1	0a
Z		1	0	1	0	1	1	1	1	0	1	1	1	0	1	1	1
S	1	0	1	0	0	1	1	1	1	1	1	0	1	0	1	0	1
C	1	0	1	1	0	1	0	0	0	1	1	1	0	1	1	0	

a carry in

b)

X		0	1	1	1	1	1	0	0	0	0	1	1	0	0	1	1
Y		1	1	1	0	0	0	1	0	0	1	1	0	1	1	0	1
W		1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
Z		1	0	1	0	1	1	1	1	0	1	1	1	0	1	1	1
T	1	1	1	0	1	0	1	0	0	0	1	0	1	0	1	1	0a
P		1	0	0	1	1	0	1	1	1	0	0	0	0	0	1	1
S	1	0	1	0	0	1	1	1	1	1	1	0	1	0	1	0	1
C	1	0	1	1	0	1	0	0	0	1	1	1	0	1	1	0	

a carry in; P output of Odd-parity module.

Exercise 2.33

a) Consider a [4:2] adder made of full adders, arranged in two levels as shown in Figure 2.32 with one bit-slice repeated in Figure E2.33.

Since, according to Table 2.2, in the second FA the delays $t_{c,s}$ and $t_{c,c}$ are smaller than the other delays, the overall delay is reduced if we connect to pin c the output of the first FA with largest delay. However, for this FA implementation the delays $t_{x,s}$ and $t_{x,c}$ are almost the same. So, assume that we connect output s to input c (same result would be obtained by connecting output c to input c). We get the critical path

$$T_{[4:2]} = t_{x,c} + t_{x,s} = (0.72 + 0.04) + (0.71 + 0.03) = 1.50$$

This delay is roughly two times the delay of one full adder. This is due to the implementation of the adder of Table 2.2 (an implementation based on two half adders). A reduction in delay would be obtained if the full adder is implemented with a separate network for the carry out signal (a two-level gate network). For

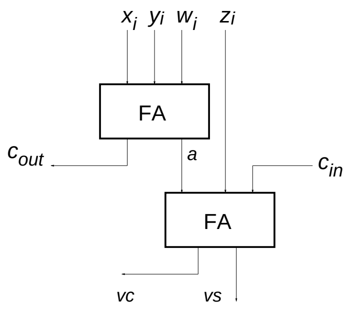


Figure E2.33: [4:2] adder implemented with FAs.

instance, if the delay of this signal would be 0.38 + 0.03, we would get a total delay of $(0.72 + 0.04) + (0.38 + 0.04) = 1.18$, roughly 1.5 the delay of a FA.

b) To show that the network in Fig.2.41 implements a [4:2] carry-save adder module, consider the following arithmetic function:

$$vs = (x_i + y_i + w_i + z_i + t_i) mod 2$$

$$(t_{i+1} + c_{i+1}) = (x_i + y_i + w_i + z_i + t_i)/2$$

Clearly, the ODD PARITY module followed by the XOR produces the correct vs (odd parity of x, y, w, z, t).

Now for t_{i+1}, c_{i+1}, we see that the implementation is symmetric with respect to x, y, w so we can consider the sum $(x + y + w)$. We have

If $x + y + w = 2$ or 3 then $t = maj(x, y, w) = 1$

Moreover,

$c = 1$ if

i) $(x + y + w = 1)$ and $(z = 1$ or $t = 1)$

In this case $x + y + w + z + t = 2$ or 3 and we get $(t = 0, c = 1)$.

ii) $(x + y + w = 2)$ and $z = 1$ and $t = 1$

In this case $x + y + w + z + t = 4$ and we get $(t = 1, c = 1)$

iii) $(x + y + w = 3)$ and $(z = 1$ or $t = 1)$

In this case $x + y + w + z + t = 4$ or 5 and we get $(t = 1, c = 1)$

In summary, if

$x + y + w = 1$ and $z = 1$ or $t = 1$ then SUM =2 or 3 and $t = 0$ and $c = 1$

$x + y + w = 2$ and $z + t < 2$ then SUM=2 or 3 and $t = 1$ and $c = 0$

$x + y + w = 2$ and $z + t = 2$ then SUM=4 and $t = 1$ and $c = 1$

$x + y + w = 3$ and $z = 1$ or $t = 1$ then SUM=4 or 5 and $t = 1$ and $c = 1$.

We now compute the delays of the [4:2] adder shown in Figure 2.41.

- Delay of s_i:

  The module ODD PARITY consists of a two-level tree of XOR gates: XOR2(XOR1,XOR1). The output s_i is produced by a level 3 XOR. Assuming a unit load on s_i , the delay is

$$\begin{aligned} ts_i &= t_{XOR1} + t_{XOR2} + t_{XOR3} \\ t_{XOR1} &= 0.3 + 0.029 \times 1.1 = 0.33ns \\ t_{XOR2} &= 0.3 + 0.029 \times (2 + 0.5) = 0.38ns \\ t_{XOR3} &= 0.15 + 0.028 \times 1 = 0.188ns \\ s_i &= 0.89ns \end{aligned}$$

- Delay of t_{i+1}: $t_t = t_{NAND1} + t_{NAND2}$:

  t_{i+1} is produced by a majority network implemented by a two-level NAND network as $NAND2(NAND1, NAND1, NAND1)$.

$$\begin{aligned} t_{NAND1} &= 0.07 + 0.033 \times 1.1 = 0.10ns \\ t_{NAND2} &= 0.08 + 0.039 \times (1.1 + 0.5) = 0.14ns \\ t_t &= 0.24ns \end{aligned}$$

- Delay of c_{i+1}: $t_c = max(t_t, t_{MUX-SEL}) + t_{MUX})$

$$\begin{aligned} t_c &= t_{MUX-SEL} + t_{MUX} \\ &= t_{XOR1} + t_{XOR2} + t_{MUX} \\ &= 0.33 + 0.38 + 0.21 + 0.05 \times 1 = 0.97ns \end{aligned}$$

We conclude that the [4:2] adder in (b) is faster than the [4:2] adder in (a).

Exercise 2.34

A radix-8 carry-save cell and a 12-bit (4-octal digit) carry-save adder are shown in Figure E2.34.

Exercise 2.35

	101	110	110	011
	1	1	0	1
	011	100	111	011
	001	011	101	111
1	1	1	0	0

Exercise 2.36

The addition of two radix-8 carry-save operands is performed in two stages:

1. A row of half-adders and full-adders as shown in Figure E2.36 reduces two radix-8 carry-save operands to an intermediate result consisting of one radix-8 carry-save and one conventional bit-vector.
2. The intermediate result is reduced to one radix-8 carry-save result using CSA8 cells defined in Exercise 2.34.

Exercise 2.37

Radix-4 signed digit addition:

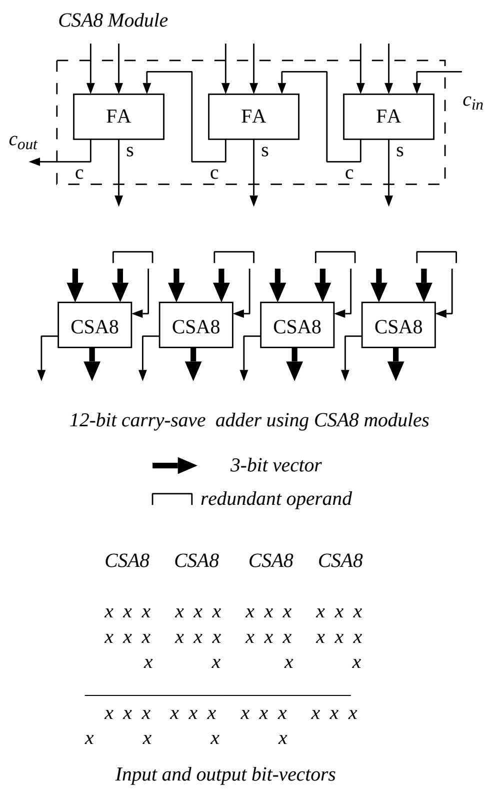


Figure E2.34: Radix-8 carry-save adder (Exercise 2.34).

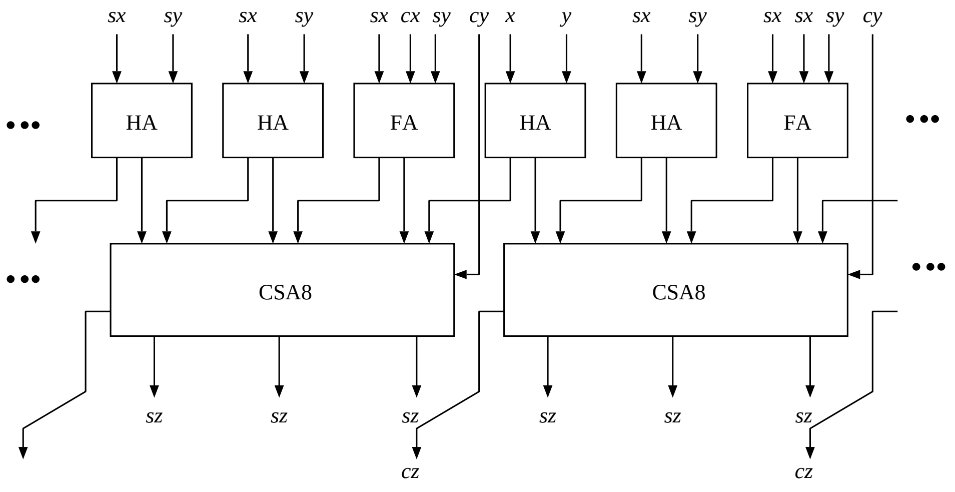


Figure E2.36: Adder implementing addition of two radix-8 carry-save operands and producing a radix-8 carry-save result. (Exercise 2.36)

X		0	$\bar{2}$	1	$\bar{3}$	1	0	1	$\bar{1}$
Y		1	0	$\bar{1}$	$\bar{3}$	$\bar{1}$	1	1	0
W		1	$\bar{2}$	0	$\bar{2}$	0	1	2	$\bar{1}$
T	0	0	0	$\bar{1}$	0	0	0	0	
S	0	1	$\bar{2}$	$\bar{1}$	$\bar{2}$	0	1	2	$\bar{1}$

Exercise 2.38

The arithmetic expressions for a radix-4 signed-digit addition are

$$\begin{aligned} a_i + b_i &= 4t_{i+1} + w_i \\ s_i &= w_i + t_i \end{aligned}$$

where the intermediate variables are $t_i \in \{-1, 0, 1\}$ and $w_i \in \{-2, -1, 0, 1, 2\}$. The two-step algorithm is not possible since the condition $a \geq (r+1)/2$ is not satisfied for $a = 2$ and $r = 4$. Therefore, a modified signed-digit addition algorithm has to be used. We chose Method 2. Let

$$P_i = \begin{cases} 1 & \textbf{if} \quad a_{i-1} \geq 0 \quad \textbf{and} \quad b_{i-1} \geq 0 \\ 0 & \textbf{otherwise} \end{cases}$$

resulting in the following table:

$a_i + b_i$	P_i	t_{i+1}	w_i
4	-	1	0
3	-	1	-1
2	0	0	2
	1	1	-2
1	-	0	1
0	-	0	0
-1	-	0	-1
-2	0	-1	2
	1	0	-2
-3	-	-1	-1
-4	-	-1	0

The block diagram is shown in Figure E2.38. The delay in the critical path is $t_P + t_{TW} + t_s$.

On the other hand, a radix-4 signed-digit adder with $a = 3$ requires only TW and S stages with the corresponding delays t^*_{TW} and t_s. The delay t^*_{TW} is larger than t_{TW} since the digit sets are larger with $a = 3$ with $a = 3$. The difference between t_P and $(t^*_{TW} - t_{TW})$ determines which adder is faster.

Exercise 2.39

Method 1:

X			0	1	$\bar{1}$	1	$\bar{1}$	0	1	$\bar{1}$
Y			1	0	1	0	1	$\bar{1}$	$\bar{1}$	1
H		1	1	0	1	0	0	0	0	
Z			$\bar{1}$	$\bar{1}$	0	$\bar{1}$	0	$\bar{1}$	0	0
Q		1	0	$\bar{1}$	1	$\bar{1}$	0	$\bar{1}$	0	0
T	0	0	$\bar{1}$	0	$\bar{1}$	0	$\bar{1}$	0	0	
W		1	0	1	1	1	0	1	0	0
S		1	$\bar{1}$	1	0	1	$\bar{1}$	1	0	0

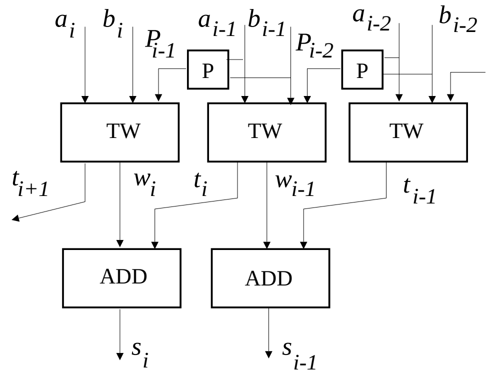


Figure E2.38: Radix-4 signed digit adder scheme with $a = 2$. (Exercise 2.38)

Method 2:

X		0	1	$\bar{1}$	1	$\bar{1}$	0	1	$\bar{1}$
Y		1	0	1	0	1	$\bar{1}$	$\bar{1}$	1
P		0	0	1	0	1	1	1	1
T	1	0	0	0	0	$\bar{1}$	0	0	
W		$\bar{1}$	1	0	1	0	1	0	0
S	1	$\bar{1}$	1	0	1	$\bar{1}$	1	0	0

Exercise 2.40

- A high-level description of the double recoding approach resulting in the redundant adder shown in Figure 2.38:

  Recoding 1:

  $x_i + y_i = 2h_{i+1} + z_i \in \{-2, -1, 0, 1, 2\}$ such that $h_i \in \{0, 1\}$ and $z_i \in \{-2, -1, 0\}$

  Recoding 2:

  $h_i + z_i = 2t_{i+1} + w_i \in \{-2, -1, 0, 1\}$ such that $t_i \in \{-1, 0\}$ and $w_i \in \{0, 1\}$

  The output $s_i = t_i + w_i \in \{-1, 0, 1\}$

- A binary-level description:

  The inputs and the output are coded as follows:

  $$x_i = x_i^+ - x_i^-, \quad y_i = y_i^+ - y_i^-, \quad s_i = s_i^+ - s_i^-$$

  The non-binary intermediate variable $z_i \in \{-2, -1, 0\}$ is coded as follows:

  $$z_i = -za_i - zb_i$$

  All coding variables are binary. In terms of these binary variables we express the high-level expressions as:

$$x_i^+ - x_i^- + y_i^+ - y_i^- = 2h_{i+1} - za_i - zb_i$$

which is decomposed as

$$x_i^+ - x_i^- + y_i^+ = 2h_{i+1} - za_i, \quad y_i^- = zb_i$$

The expression on the left is implemented by a full-adder in which signals with a negative sign are inverted. This corresponds to FAs at the top level of Figure 2.38. In the figure za_i is denoted as v_i.

Recoding 2 is desribed at the binary level by

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

which is implemented by a full-adder in which signals with a negative sign are inverted. This corresponds to the bottom level of full-adders in Figure 2.38. Finaly, the output $s_i = s_i^+ - s_i^-$ is obtained as $s_i^+ = w_i$ and $s_i^- = t_i$.

Exercise 2.41

Since $z_i, s_i \in \{-2, -1, 0, 1, 2\}$, the radix 4 should be used also for x and y operands so that $x_i, y_i \in \{0, 1, 2, 3\}$ and $x_{n-1}, y_{n-1} \in \{-2, -1, 0, 1\}$. Consequently,

$$-2 \leq x_i + y_i + z_i \leq 6$$

for $0 \leq i \leq n-2$. Moreover,

$$-6 \leq x_{n-1} + y_{n-1} + z_{n-1} \leq 4$$

The operation for digits 0 to $n-2$ consists of the following three steps:
STEP 1 $x_i + y_i + z_i = 4h_{i+1} + v_i$ where $h_i \in \{-1, 0, 1, 2\}$ and $v_i \in \{-1, 0, 1, 2\}$ so that $h_i + v_i \in \{-2, -1, 0, 1, 2, 3, 4\}$

$x_i + y_i + z_i$	-2	-1	0	1	2	3	4	5	6	7	8
h_{i+1}	-1	0	0	0	0	1	1	1	1	2	2
v_i	2	-1	0	1	2	-1	0	1	2	-1	0

STEP 2 $h_i + v_i = 4t_{i+1} + u_i$ where $t_i \in \{0, 1\}$ and $u_i \in \{-2, -1, 0, 1\}$

$h_i + v_i$	-2	-1	0	1	2	3	4
t_{i+1}	0	0	0	0	1	1	1
u_i	-2	-1	0	1	-2	-1	0

STEP 3 $t_i + u_i = s_i \in \{-2, -1, 0, 1, 2\}$

For the most-significant digit ($i = n-1$) the algorithm is modified as follows:
STEP 1 : $x_{n-1} + y_{n-1} + z_{n-1} = 4h_n + v_{n-1}$

Since $h_{n-1} \in \{-1, 0, 1, 2\}$ (from algorithm for $i \leq n-2$) we make

$$v_{n-1} \in \{-2, -1, 0, 1\}, \qquad h_n \in \{-1, 0, 1\}$$

STEP 2 : $v_{n-1} + h_{n-1} \in \{-3, ..., 3\}$

Since $t_{n-1} \in \{0, 1\}$ (from algorithm for $0 \leq i \leq n-2$) we make

$$u_{n-1} \in \{-2, -1, 0, 1\}, \qquad t_n \in \{-1, 0, 1\}$$

STEP 3 : $s_{n-1} = u_{n-1} + t_{n-1}$

Finally, $s_n = t_n + h_n$ with $s_n \in \{-2, ..., 2\}$.

The critical path includes HV, TW, and S modules.

Exercise 2.42

The input operands x and y are in radix 4 with the digit set {0,1,2,3}. The output s is in radix-4 with the signed-digit set {-2,-1,0,1,2}.

1. Method 1 - addition consists of two recodings and a carry-free addition.

   Recoding 1: $x_i + y_i = 4h_{i+1} + z_i$. The sum of input digits is in the range [0,6]. The intermediate variables are $h_I \in \{0, 1\}$ and $z_i \in \{-1, 0, 1, 2\}$.

$x_i + y_i$	0	1	2	3	4	5	6
h_{i+1}	0	0	0	1	1	1	1
z_i	0	1	2	-1	0	1	2

   Recoding 2: $h_i + z_i = 4t_{i+1} + w_i$. The input to this recoding is in the range [-1,3] and the intermediate variables are $t_i \in \{0, 1\}$ and $w_i \in \{-2, -1, 0, 1\}$.

$h_i + z_i$	-1	0	1	2	3
t_{i+1}	0	0	0	1	1
w_i	-1	0	1	-2	-1

   The sum is obtained by a carry-free addition: $s_i = w_i + t_i$, $s_i \in \{-2, -1, 0, 1, 2\}$

2. Method 2 - addition uses information from previous digit.

   Consider the following recoding:

$x_i + y_i$	0	1	2	3	4	5	6
t_{i+1}	0	0	1	1	1	1	2
w_i	0	1	-2	-1	0	1	-2

   This recoding produces an out-of-range output digit $s_i = 3$ if $t_i = 2$ and $w_i = 1$. This can be avoided by using alternate recoding when $x_i + y_i$ is 1 or 5, marked by (*). The choice depends on the value of $x_{i-1} + y_{i-1}$.

   $$d_i = \begin{cases} 1 & \textbf{if} \;\; x_{i-1} + y_{i-1} \geq 5 \\ 0 & \textbf{otherwise} \end{cases}$$

   If $d_i = 1$, choose the recoding in ().

$x_i + y_i$	0	1(*)	2	3	4	5(*)	6
t_{i+1}	0	0 (1)	1	1	1	1 (2)	2
w_i	0	1 (-3)	-2	-1	0	1 (-3)	-2

   This recoding guarantees that if $t_i = 2$, w_i cannot be 1. Consequently, $s_i = t_i + w_i \in \{-2, 1, 0, 1, 2\}$. To simplify implementation, it is possible to relax the condition when $d_i = 1$ to $x_{i-1} + y_{i-1} \geq 2$.

Exercise 2.43

Radix-2 signed digit addition of one conventional and one signed-digit operand:

X		0	1	1	1	0	1	1	0
Y^+		1	0	1	0	0	0	1	1
Y^-		0	1	0	0	0	1	0	0
W		1	0	0	1	0	0	0	1
T	1	0	1	1	0	0	1	1	
S^+	1	0	1	1	0	0	1	1	0
S^-		1	0	0	1	0	0	0	1
S	1	$\bar{1}$	1	1	$\bar{1}$	0	1	1	$\bar{1}$

Exercise 2.44

Radix-2 signed digit addition of two signed-digit operands:

X^+		0	1	0	0	0	1	0	0
X^-		0	0	1	1	1	0	0	0
Y^+		1	1	1	0	0	1	1	1
Z		1	0	0	1	1	0	1	1
H	1	1	0	0	0	1	1	1	0
Y^-		0	0	0	1	1	0	0	0
S^+	1	0	0	0	0	1	1	0	0
S^-	0	0	0	1	1	0	0	1	0
S	1	0	0	$\bar{1}$	$\bar{1}$	1	1	$\bar{1}$	1

Exercise 2.45

The adder output s_i is a switching function of $2i$ inputs (ignoring the carry-in). It is a well known result that using modules of m binary inputs, the implementation requires at least $\lceil log_m(2i)\rceil$ levels. The proof of this is based on building a tree of modules: the first level has $\lceil 2i/m\rceil$ modules and outputs. For the second level, group m outputs from the first level per module, and so on until the last level has one module.