Solutions for Appendix A Exercises

A.1

A	B	$\overline{A}$	$\overline{B}$	$\overline{A+B}$	$\overline{A}\cdot\overline{B}$	$\overline{A\cdot B}$	$\overline{A}+\overline{B}$
0	0	1	1	1	1	1	1
0	1	1	0	0	0	1	1
1	0	0	1	0	0	1	1
1	1	0	0	0	0	0	0

A.2 Here is the first equation:

$$E = ((A\cdot B)+(A\cdot C)+(B\cdot C))\cdot(\overline{A\cdot B\cdot C}).$$

Now use DeMorgan's theorems to rewrite the last factor:

$$E = ((A\cdot B)+(A\cdot C)+(B\cdot C))\cdot(\overline{A}+\overline{B}+\overline{C})$$

Now distribute the last factor:

$$E = ((A\cdot B)\cdot(\overline{A}+\overline{B}+\overline{C}))+((A\cdot C)\cdot(\overline{A}+\overline{B}+\overline{C}))+((B\cdot C)\cdot(\overline{A}+\overline{B}+\overline{C}))$$

Now distribute within each term; we show one example:

$$((A\cdot B)\cdot(\overline{A}+\overline{B}+\overline{C})) = (A\cdot B\cdot\overline{A})+(A\cdot B\cdot\overline{B})+(A\cdot B\cdot\overline{C}) = 0+0+(A\cdot B\cdot\overline{C})$$

(This is simply $A\cdot B\cdot\overline{C}$.) Thus, the equation above becomes

$$E = (A\cdot B\cdot\overline{C})+(A\cdot\overline{B}\cdot C)+(\overline{A}\cdot B\cdot C)$$, which is the desired result.

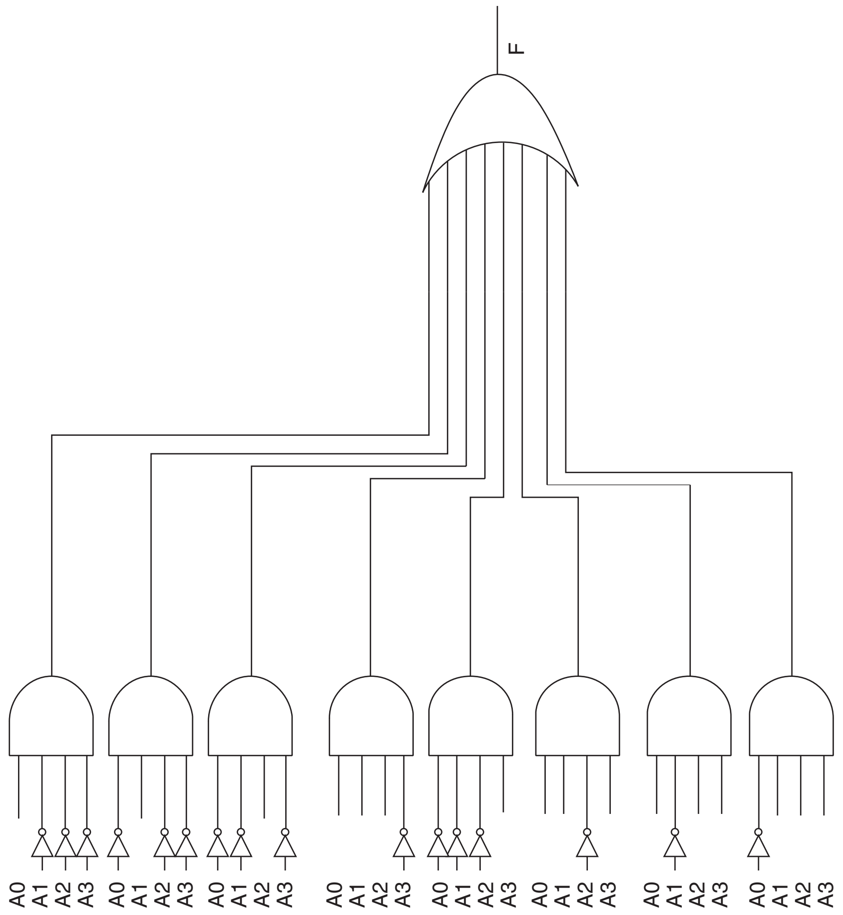
A.7 Four inputs A0–A3 & F (O/P) = 1 if an odd number of 1s exist in A.

A3	A2	A1	A0	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

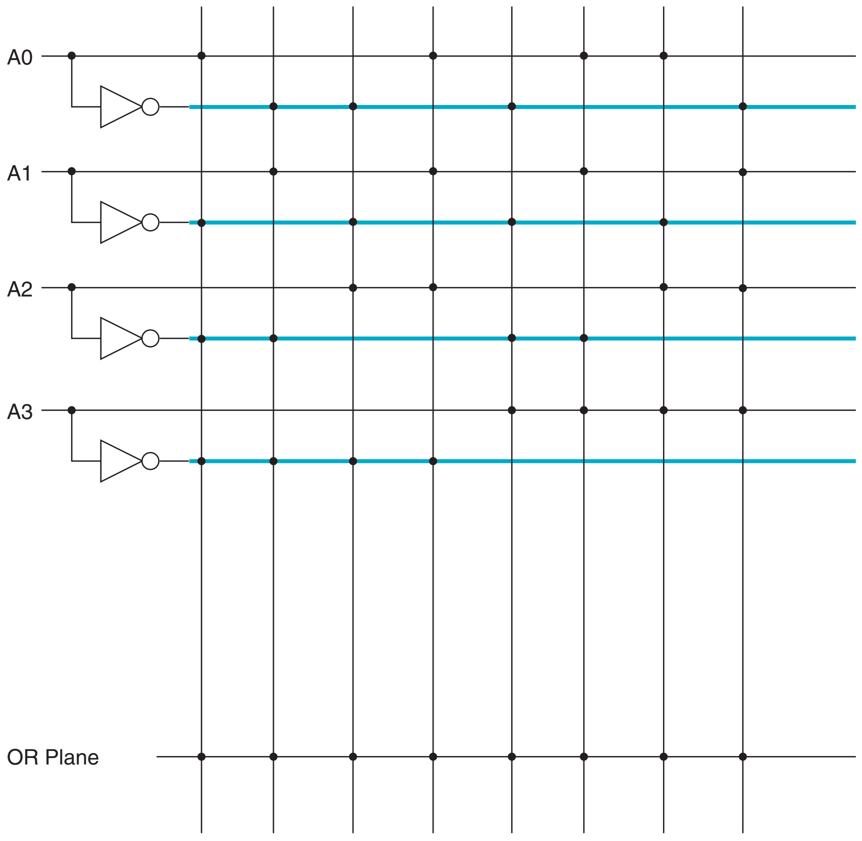
A.8 F = A3′A2′A1′A0 + A3′A2′A1 A0′ + A3′A2 A1′A0' + A3′A2 A1 A0 +

A3 A2′A1′A0′ + A3 A2′A1 A0 + A3′A2′A1 A0′ + A3 A2 A1 A0′

Note: F = A0 XOR A1 XOR A2 XOR A3. Another question can ask the students to prove that.


F
A0
A1
A2
A3
A0
A1
A2
A3
A0
A1
A2
A3
A0
A1
A2
A3
A0
A1
A2
A3
A0
A1
A2
A3
A0
A1
A2
A3
A0
A1
A2
A3

A.9



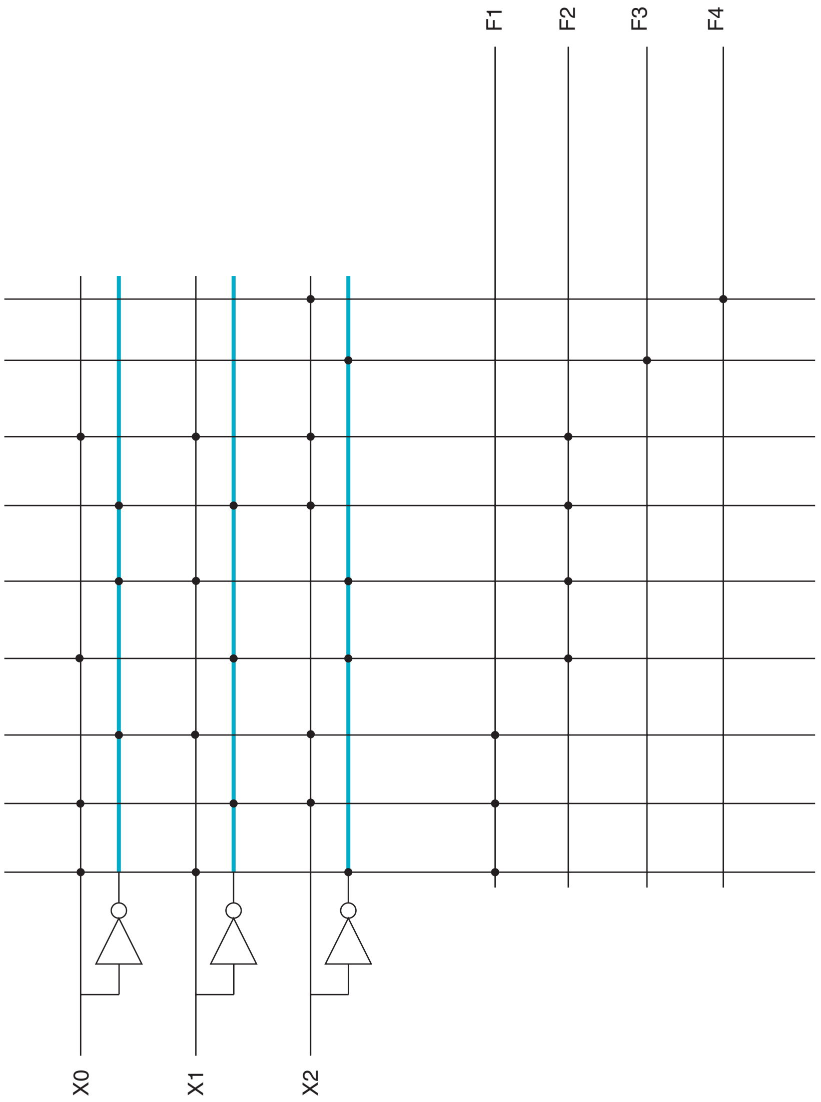
A.10 No solution provided.

A.11

x2	x1	x0	F1	F2	F3	F4
0	0	0	0	0	1	0
0	0	1	0	1	1	0
0	1	0	0	1	1	0
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	1	1	0	0	1
1	1	0	1	0	0	1
1	1	1	0	1	0	1

$F1 = X2'X1\ X0 + X2\ X1'X0 + X2\ X1\ X0'$
$F2 = X2'X1'X0 + X2'X1\ X0' + X2\ X1'X0' + X2\ X1\ X0 = (A \text{ XOR } B \text{ XOR } C)$
$F3 = X2'$
$F4 = X2\ (= F3')$

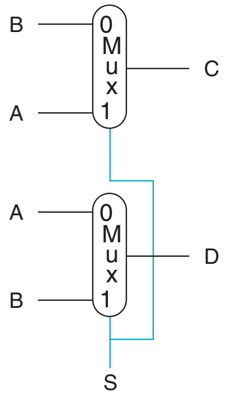
A.12



A.13

- $\overline{x2}y2 + x2y2\overline{x1}y1 + x2y2x1y1\overline{x0}y0 + \overline{x2y2x1}y1 + \overline{x2y2x1y1x0}y0 + \overline{x2y2}x1y1\overline{x0}y0 + x2y2\overline{x1y1x0}y0$
  $+ x2y2x1y1\overline{x0}y0$
- $x2\overline{y2} + x2y2\overline{x1}y1 + x2y2x1y1\overline{x0}y0 + \overline{x2y2x1}y1 + x2y2\overline{x1y1x0}y0 + \overline{x2y2x1y1x0}y0 + \overline{x2y2}x1y1\overline{x0}y0$
- $(x2y2 + \overline{x2}\,\overline{y2})(x1y1 + \overline{x1}\,\overline{y1})(x0y0 + \overline{x0}\,\overline{y0})$

A.14



A.15 Generalizing DeMorgan's theorems for this exercise, if $\overline{A+B} = \overline{A} \cdot \overline{B}$, then $\overline{A+B+C} = \overline{A+(B+C)} = \overline{A} \cdot \overline{(B+C)} = \overline{A} \cdot (\overline{B} \cdot \overline{C}) = \overline{A} \cdot \overline{B} \cdot \overline{C}$.

Similarly,

$\overline{A \cdot B \cdot C} = \overline{A \cdot (B \cdot C)} = \overline{A} + \overline{B \cdot C} = \overline{A} \cdot (\overline{B} + \overline{C}) = \overline{A} + \overline{B} + \overline{C}$.

Intuitively, DeMorgan's theorems say that (1) the negation of a sum-of-products form equals the product of the negated sums, and (2) the negation of a product-of-sums form equals the sum of the negated products. So,

$E = \overline{\overline{E}}$

$= \overline{\overline{(A \cdot B \cdot \overline{C}) + (A \cdot C \cdot \overline{B}) + (B \cdot C \cdot \overline{A})}}$

$= \overline{\overline{(A \cdot B \cdot \overline{C})} \cdot \overline{(A \cdot C \cdot \overline{B})} \cdot \overline{(B \cdot C \cdot \overline{A})}}$; first application of DeMorgan's theorem

$= \overline{(\overline{A} + \overline{B} + C) \cdot (\overline{A} + \overline{C} + B) \cdot (\overline{B} + \overline{C} + A)}$; second application of DeMorgan's theorem and product-of-sums form

A.16 No solution provided.

A.18 2-1 multiplexor and 8 bit up/down counter.

A.19

```
module LATCH(clock,D,Q,Qbar)
input clock,D;
reg Q;
wire Qbar;
assign Qbar = ~Q;
always @(D,clock)  //senstivity list watches clock and data
begin
   if(clock)
     Q = D;
end
endmodule
```

A.20

```
module  decoder (in, out, enable);
input [1:0] in;
input enable
output [3:0] out;
reg [3:0] out;

always @ (enable, in)
  if (enable) begin
  out = 0;
  case (in)
         2'h0 : out = 4'h1;
         2'h1 : out = 4'h2;
         2'h2 : out = 4'h4;
         2'h3 : out = 4'h8;
  endcase
end
endmodule
```

A.21

```
module ACC(Clk, Rst, Load, IN, LOAD, OUT);

input Clk, Rst, Load;
input [3:0] IN;
input [15:0] LOAD
output [15:0] OUT;

wire [15:0] W;
reg [15:0] Register;

initial begin
Register = 0;
end
assign W = IN + OUT;

always @ (Rst,Load)
begin
if Rst begin
         Register = 0;
end

if Load begin
         Register = LOAD;
end
end

always @ (Clk)
begin
         Register <= W;
end

endmodule
```

A.22 We use Figure 3.5 to implement the multiplier. We add a control signal "load" to load the multiplicand and the multiplier. The load signal also initiates the multiplication. An output signal "done" indicates that simulation is done.

```
module MULT(clk, load, Multiplicand, Multiplier, Product, done);
input clk, load;
input [31:0] Multiplicand, Multiplier;
output [63:0] Product;
output done;

         reg [63:0] A, Product;
         reg [31:0] B;
         reg [5:0] loop;
         reg done;

         initial begin
           done = 0; loop = 0;
         end

         always @(posedge clk) begin
           if (load && loop ==0) begin
                  done <= 0;
                  Product <=0;
                  A <= Multiplicand;
                  B <= Multiplier;
                  loop <= 32;
         end

         if(loop > 0) begin
             if(B[0] == 1)
                Product <= Product + A;

             A <= A << 1;
             B <= B >> 1;
             loop <= loop -1;

             if(loop == 0)
               done <= 1;
           end

         end
         endmodule
```

A.23 We use Figure 3.10 for divider implementation, with additions similar to the ones listed above in the answer for Exercise A.22.

```
module DIV(clk, load, Divisor, Dividend, Quotient, Remainder, done);

input clk, load;
input [31:0] Divisor;
input [63:0] Dividend;
output [31:0] Quotient;
input [31:0] Remainder;
output done;

reg [31:0] Quotient;     //Quotient
reg [63:0] D, R;         //Divisor, Remainder
reg [6:0] loop;          //Loop counter
reg done;

initial begin
  done = 0; loop = 0;
end

assign Remainder = R[31:0];

always @(posedge clk) begin
  if (load && loop ==0) begin
    done <= 0;
    R <=Dividend;
    D <= Divisor << 32;
        Quotient <=0;
        loop <= 33;
end

if(loop > 0) begin
        if(R - D >= 0)
          begin
          Quotient <= (Quotient << 1) + 1;
          R <= R - D;
          end
        else
          begin
          Quotient <= Quotient << 1;
        end

        D <= D >> 1;
        loop <= loop - 1;
```

```
                if(loop == 0)
                  done <= 1;
          end

        end
        endmodule
```

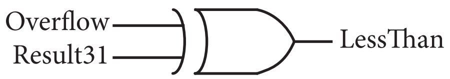
Note: This code does not check for division by zero (i.e., when Divisior = = 0) or for quotient overflow (i.e., when Divisior < = Dividiend [64:32]).

A.24 The ALU-supported set less than (`slt`) uses just the sign bit. In this case, if we try a set less than operation using the values -7_{ten} and 6_{ten}, we would get $-7 > 6$. This is clearly wrong. Modify the 32-bit ALU in Figure 4.11 on page 169 to handle `slt` correctly by factor in overflow in the decision.

If there is no overflow, the calculation is done properly in Figure 4.17 and we simply use the sign bit (Result31). If there is overflow, however, then the sign bit is wrong and we need the inverse of the sign bit.

Overflow	Result31	LessThan
0	0	0
0	1	1
1	0	1
1	1	0

$$LessThan = Overflow \oplus Result31$$



Overflow	Result31	LessThan
0	0	0
0	1	1
1	0	1
1	1	0

A.25 Given that a number that is greater than or equal to zero is termed positive and a number that is less than zero is negative, inspection reveals that the last two rows of Figure 4.44 restate the information of the first two rows. Because A – B = A + (–B), the operation A – B when A is positive and B negative is the same as the operation A + B when A is positive and B is positive. Thus the third row restates the conditions of the first. The second and fourth rows refer also to the same condition.

Because subtraction of two's complement numbers is performed by addition, a complete examination of overflow conditions for addition suffices to show also when overflow will occur for subtraction. Begin with the first two rows of Figure 4.44 and add rows for A and B with opposite signs. Build a table that shows all possible combinations of Sign and CarryIn to the sign bit position and derive the CarryOut, Overflow, and related information. Thus,

Sign A	Sign B	Carry In	Carry Out	Sign of result	Correct sign of result	Over-flow?	Carry In XOR Carry Out	Notes
0	0	0	0	0	0	No	0	
0	0	1	0	1	0	Yes	1	Carries differ
0	1	0	0	1	1	No	0	$\|A\| < \|B\|$
0	1	1	1	0	0	No	0	$\|A\| > \|B\|$
1	0	0	0	1	1	No	0	$\|A\| > \|B\|$
1	0	1	1	0	0	No	0	$\|A\| < \|B\|$
1	1	0	1	0	1	Yes	1	Carries differ
1	1	1	1	1	1	No	0	

From this table an Exclusive OR (XOR) of the CarryIn and CarryOut of the sign bit serves to detect overflow. When the signs of A and B differ, the value of the CarryIn is determined by the relative magnitudes of A and B, as listed in the Notes column.

A.26 C1 = c4, C2 = c8, C3 = c12, and C4 = c16.

$c4 = G_{3,0} + (P_{3,0} \cdot c0)$.

c8 is given in the exercise.

$c12 = G_{11,8} + (P_{11,8} \cdot G_{7,4}) + (P_{11,8} \cdot P_{7,4} \cdot G_{3,0}) + (P_{11,8} \cdot P_{7,4} \cdot P_{3,0} \cdot c0)$.

$$c16 = G_{15,12} + (P_{15,12} \cdot G_{11,8}) + (P_{15,12} \cdot P_{11,8} \cdot G_{7,4}) + (P_{15,12} \cdot P_{11,8} \cdot P_{7,4} \cdot G_{3,0}) + (P_{15,12} \cdot P_{11,8} \cdot P_{7,4} \cdot P_{3,0} \cdot c0).$$

A.27 The equations for c4, c8, and c12 are the same as those given in the solution to Exercise 4.44. Using 16-bit adders means using another level of carry lookahead logic to construct the 64-bit adder. The second level generate, G0′, and propagate, P0′, are

$$G0' = G_{15,0} = G_{15,12} + P_{15,12} \cdot G_{11,8} + P_{15,12} \cdot P_{11,8} \cdot G_{7,4} + P_{15,12} \cdot P_{11,8} \cdot P_{7,4} \cdot G_{3,0}$$

and

$$P0' = P_{15,0} = P_{15,12} \cdot P_{11,8} \cdot P_{7,4} \cdot P_{3,0}$$

Using G0′ and P0′, we can write c16 more compactly as

$$c16 = G_{15,0} + P_{15,0} \cdot c0$$

and

$$c32 = G_{31,16} + P_{31,16} \cdot c16$$
$$c48 = G_{47,32} + P_{47,\,32} \cdot c32$$
$$c64 = G_{63,48} + P_{63,48} \cdot c48$$

A 64-bit adder diagram in the style of Figure B.6.3 would look like the following:

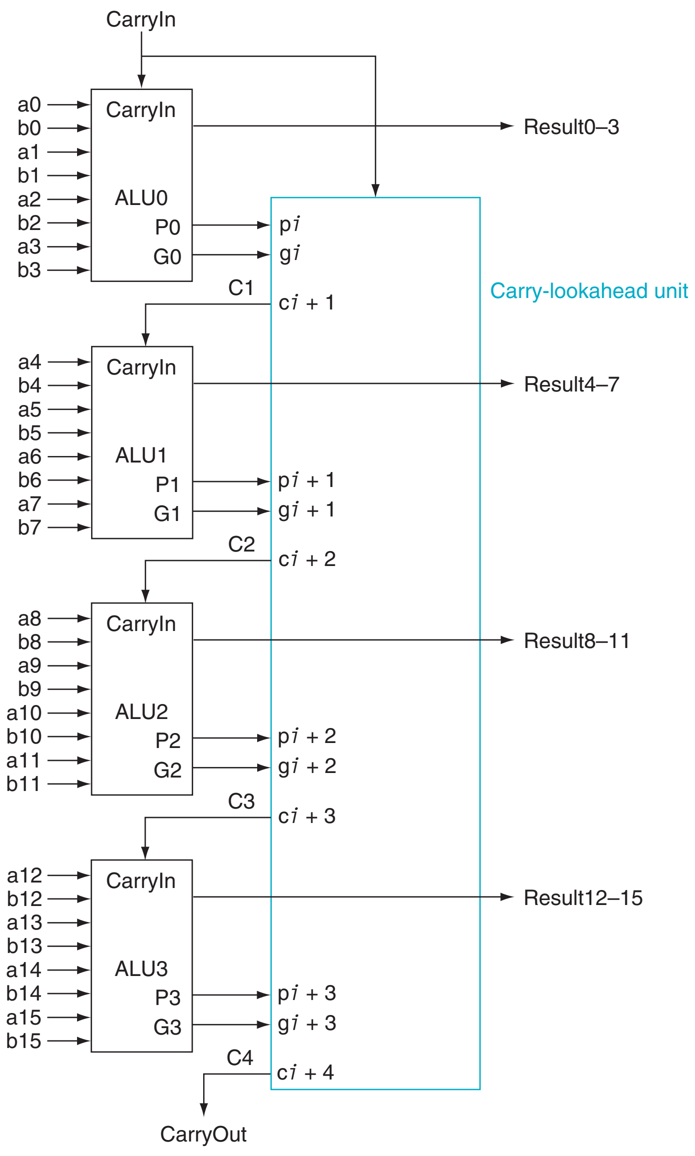


FIGURE A.6.3 Four 4-bit ALUs using carry lookahead to form a 16-bit adder. Note that the carries come from the carry-lookahead unit, not from the 4-bit ALUs.

A.28 No solution provided.

A.29 No solution provided.

A.30 No solution provided.

A.31 No solution provided.

A.32 No solution provided.

A.33 No solution provided.

A.34 The longest paths through the top (ripple carry) adder organization in Figure A.14.1 all start at input a0 or b0 and pass through seven full adders on the way to output s4 or s5. There are many such paths, all with a time delay of $7 \times 2T = 14T$. The longest paths through the bottom (carry save) adder all start at input b0, e0, f0, b1, e1, or f1 and proceed through six full adders to outputs s4 or s5. The time delay for this circuit is only $6 \times 2T = 12T$.