

Convert $(857)_{10}$

	Quotient	Remainder	
$857 \div 2 =$	428	1	LSB
$428 \div 2 =$	214	0	
$214 \div 2 =$	107	0	
$107 \div 2 =$	53	1	
$53 \div 2 =$	26	1	
$26 \div 2 =$	13	0	
$13 \div 2 =$	6	1	
$6 \div 2 =$	3	0	
$3 \div 2 =$	1	1	
$1 \div 2 =$	0	1	MSB

Result is $(1101011001)_2$

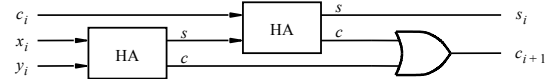
Figure 5.1. Conversion from decimal to binary.

Decimal	Binary	Octal	Hexadecimal
00	0000	00	00
01	0001	01	01
02	0010	02	02
03	0011	03	03
04	0100	04	04
05	0101	05	05
06	0110	06	06
07	0111	07	07
08	1000	10	08
09	1001	11	09
10	1010	12	0A
11	1011	13	0B
12	1100	14	0C
13	1101	15	0D
14	1110	16	0E
15	1111	17	0F
16	10000	20	10
17	10001	21	11
18	10010	22	12

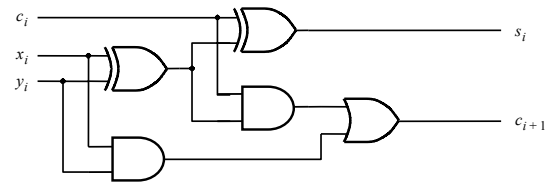
Table 5.1. Numbers in different systems.

$$\begin{array}{r}
 X = x_4x_3x_2x_1x_0 \quad 01111 \quad (15)_{10} \\
 + Y = y_4y_3y_2y_1y_0 \quad 01010 \quad (10)_{10} \\
 \hline
 \quad \quad \quad 1110 \quad \leftarrow \text{Generated carries} \\
 \hline
 S = s_4s_3s_2s_1s_0 \quad 11001 \quad (25)_{10}
 \end{array}$$

Figure 5.3. An example of addition.



(a) Block diagram



(b) Detailed diagram

Figure 5.5. A decomposed implementation of the full-adder circuit.

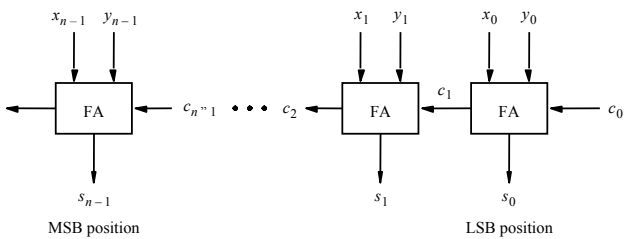
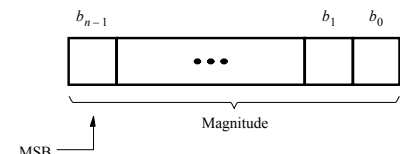
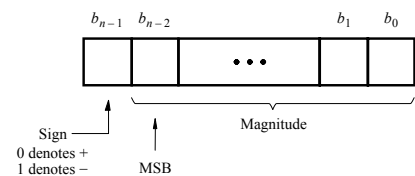


Figure 5.6. An n -bit ripple-carry adder.



(a) Unsigned number



(b) Signed number

Figure 5.8. Formats for representation of integers.

$b_3b_2b_1b_0$	Sign and magnitude	1's complement	2's complement
0111	+7	+7	+7
0110	+6	+6	+6
0101	+5	+5	+5
0100	+4	+4	+4
0011	+3	+3	+3
0010	+2	+2	+2
0001	+1	+1	+1
0000	+0	+0	+0
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-0	-1

Table 5.2. Interpretation of four-bit signed integers.

$$\begin{array}{r}
 (+5) \quad 0101 \\
 +(+2) \quad +0010 \\
 \hline
 (+7) \quad 0111
 \end{array}
 \qquad
 \begin{array}{r}
 (-5) \quad 1010 \\
 +(+2) \quad +0010 \\
 \hline
 (-3) \quad 1100
 \end{array}$$

$$\begin{array}{r}
 (+5) \quad 0101 \\
 +(-2) \quad +1101 \\
 \hline
 (+3) \quad 10010 \\
 \quad \quad \quad \leftarrow 1 \\
 \quad \quad \quad 0011
 \end{array}
 \qquad
 \begin{array}{r}
 (-5) \quad 1010 \\
 +(-2) \quad +1101 \\
 \hline
 (-7) \quad 10111 \\
 \quad \quad \quad \leftarrow 1 \\
 \quad \quad \quad 1000
 \end{array}$$

Figure 5.9. Examples of 1's complement addition.

$$\begin{array}{r}
 (+5) \quad 0101 \\
 +(+2) \quad +0010 \\
 \hline
 (+7) \quad 0111
 \end{array}
 \qquad
 \begin{array}{r}
 (-5) \quad 1011 \\
 +(+2) \quad +0010 \\
 \hline
 (-3) \quad 1101
 \end{array}$$

$$\begin{array}{r}
 (+5) \quad 0101 \\
 +(-2) \quad +1110 \\
 \hline
 (+3) \quad 10011 \\
 \quad \quad \quad \uparrow \\
 \quad \quad \quad \text{ignore}
 \end{array}
 \qquad
 \begin{array}{r}
 (-5) \quad 1011 \\
 +(-2) \quad +1110 \\
 \hline
 (-7) \quad 11001 \\
 \quad \quad \quad \uparrow \\
 \quad \quad \quad \text{ignore}
 \end{array}$$

Figure 5.10. Examples of 2's complement addition.

$$\begin{array}{r}
 (+5) \quad 0101 \\
 -(+2) \quad -0010 \\
 \hline
 (+3) \quad 10011 \\
 \quad \quad \quad \uparrow \\
 \quad \quad \quad \text{ignore}
 \end{array}
 \qquad
 \begin{array}{r}
 (-5) \quad 1011 \\
 -(+2) \quad -0010 \\
 \hline
 (-7) \quad 11001 \\
 \quad \quad \quad \uparrow \\
 \quad \quad \quad \text{ignore}
 \end{array}$$

$$\begin{array}{r}
 (+5) \quad 0101 \\
 -(-2) \quad -1110 \\
 \hline
 (+7) \quad 0111
 \end{array}
 \qquad
 \begin{array}{r}
 (-5) \quad 1011 \\
 -(-2) \quad -1110 \\
 \hline
 (-3) \quad 1101
 \end{array}$$

Figure 5.11. Examples of 2's complement subtraction.

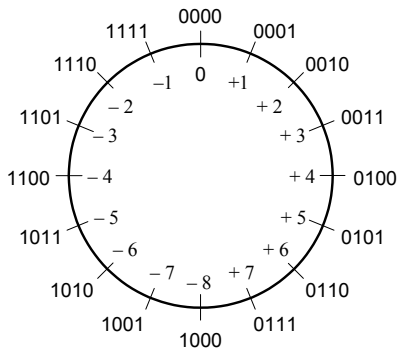


Figure 5.12. Graphical interpretation of four-bit 2's complement numbers.

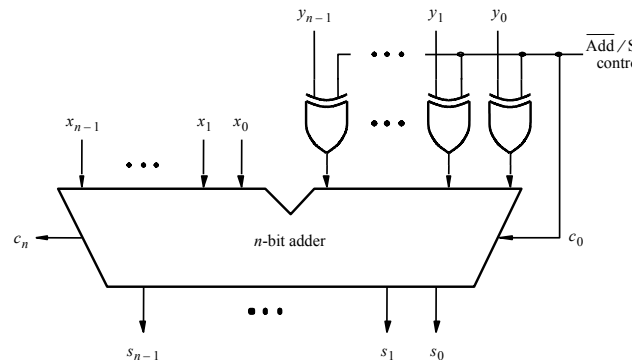


Figure 5.13. Adder/subtractor unit.

$\begin{array}{r} (+7) \quad 0111 \\ + (+2) \quad +0010 \\ \hline (+9) \quad 1001 \\ c_4 = 0 \\ c_3 = 1 \end{array}$	$\begin{array}{r} (-7) \quad 1001 \\ + (+2) \quad +0010 \\ \hline (-5) \quad 1011 \\ c_4 = 0 \\ c_3 = 0 \end{array}$
$\begin{array}{r} (+7) \quad 0111 \\ + (-2) \quad +1110 \\ \hline (+5) \quad 10101 \\ c_4 = 1 \\ c_3 = 1 \end{array}$	$\begin{array}{r} (-7) \quad 1001 \\ + (-2) \quad +1110 \\ \hline (-9) \quad 10111 \\ c_4 = 1 \\ c_3 = 0 \end{array}$

Figure 5.14. Examples of determination of overflow.

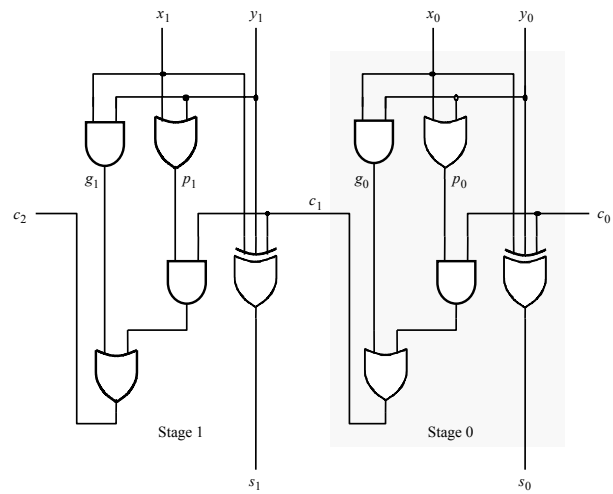


Figure 5.15. A ripple-carry adder with generate/propagate signals.

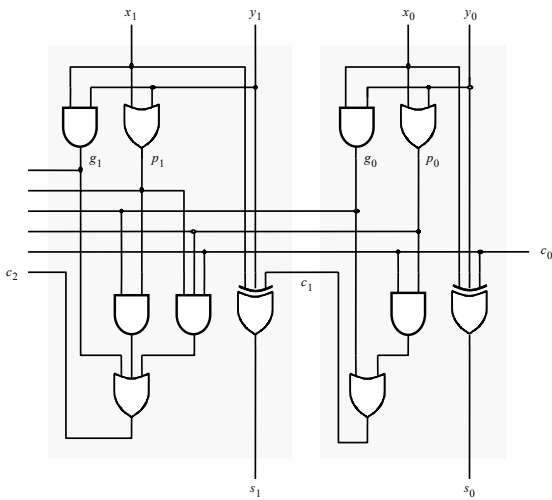


Figure 5.16. The first two stages of a carry-lookahead adder.

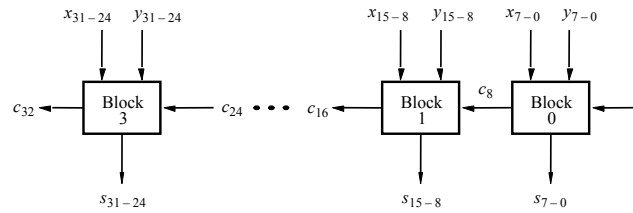


Figure 5.17. A hierarchical carry-lookahead adder with ripple-carry between blocks.

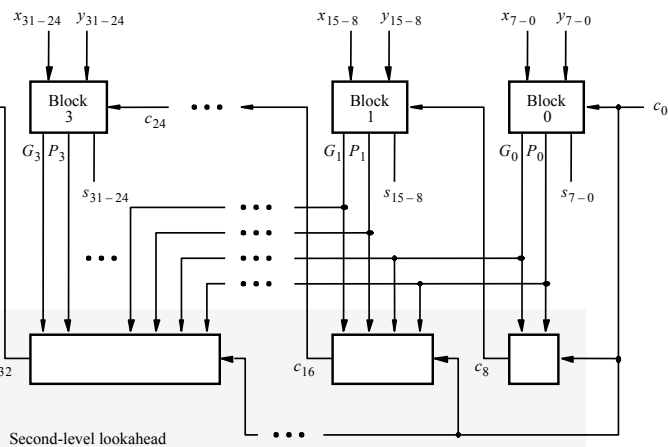


Figure 5.18. A hierarchical carry-lookahead adder.

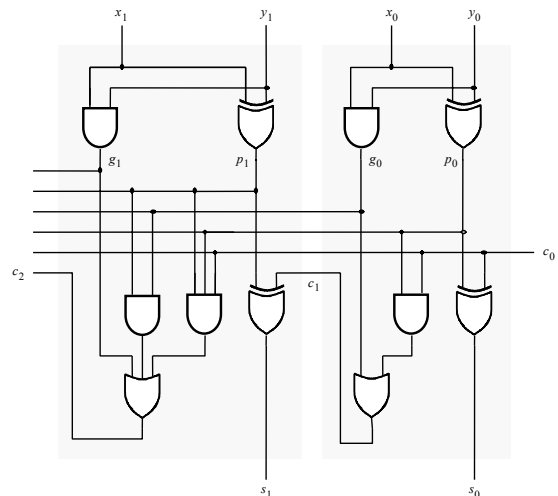


Figure 5.19. An alternative design for a carry-lookahead adder.

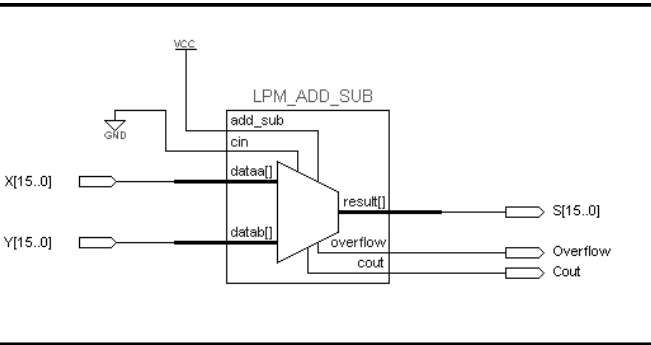


Figure 5.20. Schematic using an LPM adder/subtractor module.

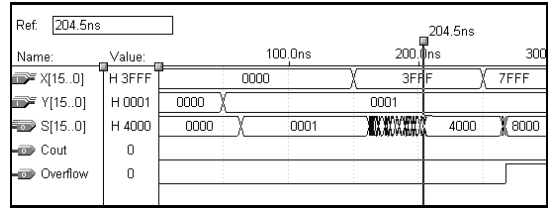


Figure 5.21. Simulation results for the LPM adder optimized for cost.

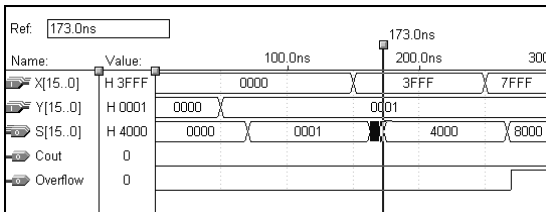


Figure 5.22. Simulation results for the LPM adder optimized for speed.

```

module fulladd (Cin, x, y, s, Cout);
  input Cin, x, y;
  output s, Cout;

  xor (s, x, y, Cin);
  and (z1, x, y);
  and (z2, x, Cin);
  and (z3, y, Cin);
  or (Cout, z1, z2, z3);

endmodule

```

Figure 5.23. Verilog code for the full-adder using gate level primitives.

```

module fulladd (Cin, x, y, s, Cout);
  input Cin, x, y;
  output s, Cout;

  xor (s, x, y, Cin);
  and (z1, x, y),
      (z2, x, Cin),
      (z3, y, Cin);
  or (Cout, z1, z2, z3);

endmodule

```

Figure 5.24. Another version of Verilog code from Figure 5.23.

```

module fulladd (Cin, x, y, s, Cout);
  input Cin, x, y;
  output s, Cout;

  assign s = x ^ y ^ Cin;
  assign Cout = (x & y) | (x & Cin) | (y & Cin);

endmodule

```

Figure 5.25. Verilog code for the full-adder using continuous assignment.

```

module fulladd (Cin, x, y, s, Cout);
  input Cin, x, y;
  output s, Cout;

  assign s = x ^ y ^ Cin,
         Cout = (x & y) | (x & Cin) | (y & Cin);

endmodule

```

Figure 5.26. Another version of Verilog code from Figure 5.25.

```

module adder4 (carryin, x3, x2, x1, x0, y3, y2, y1, y0, s3, s2, s1, s0, carryout);
  input carryin, x3, x2, x1, x0, y3, y2, y1, y0;
  output s3, s2, s1, s0, carryout;

  fulladd stage0 (carryin, x0, y0, s0, c1);
  fulladd stage1 (c1, x1, y1, s1, c2);
  fulladd stage2 (c2, x2, y2, s2, c3);
  fulladd stage3 (c3, x3, y3, s3, carryout);

endmodule

module fulladd (Cin, x, y, s, Cout);
  input Cin, x, y;
  output s, Cout;

  assign s = x ^ y ^ Cin,
         Cout = (x & y) | (x & Cin) | (y & Cin);

endmodule

```

Figure 5.27. Verilog code for a four-bit adder.

```

module adder4 (carryin, X, Y, S, carryout);
  input carryin;
  input [3:0] X, Y;
  output [3:0] S;
  output carryout;
  wire [3:1] C;

  fulladd stage0 (carryin, X[0], Y[0], S[0], C[1]);
  fulladd stage1 (C[1], X[1], Y[1], S[1], C[2]);
  fulladd stage2 (C[2], X[2], Y[2], S[2], C[3]);
  fulladd stage3 (C[3], X[3], Y[3], S[3], carryout);

endmodule

```

Figure 5.28. A four-bit adder using vectors.

```

module addern (carryin, X, Y, S, carryout);
  parameter n=32;
  input carryin;
  input [n-1:0] X, Y;
  output [n-1:0] S;
  output carryout;
  reg [n-1:0] S;
  reg carryout;
  reg [n:0] C;
  integer k;

  always @(X or Y or carryin)
  begin
    C[0] = carryin;
    for (k = 0; k < n; k = k+1)
    begin
      S[k] = X[k] ^ Y[k] ^ C[k];
      C[k+1] = (X[k] & Y[k]) | (X[k] & C[k]) | (Y[k] & C[k]);
    end
    carryout = C[n];
  end

endmodule

```

Figure 5.29. A generic specification of a ripple-carry adder.

```

module addern (carryin, X, Y, S);
  parameter n = 32;
  input carryin;
  input [n-1:0] X, Y;
  output [n-1:0] S;
  reg [n-1:0] S;

  always @(X or Y or carryin)
    S = X + Y + carryin;

endmodule

```

Figure 5.30. Specification of an n -bit adder using arithmetic assignment.

```

module addern (carryin, X, Y, S, carryout, overflow);
  parameter n = 32;
  input carryin;
  input [n-1:0] X, Y;
  output [n-1:0] S;
  output carryout, overflow;
  reg [n-1:0] S;
  reg carryout, overflow;

  always @(X or Y or carryin)
  begin
    S = X + Y + carryin;
    carryout = (X[n-1] & Y[n-1]) | (X[n-1] & ~S[n-1]) | (Y[n-1] & ~S[n-1]);
    overflow = carryout ^ X[n-1] ^ Y[n-1] ^ S[n-1];
  end

endmodule

```

Figure 5.31. An n -bit adder with carry-out and overflow signals.

```

module addern (carryin, X, Y, S, carryout, overflow);
  parameter n = 32;
  input carryin;
  input [n-1:0] X, Y;
  output [n-1:0] S;
  output carryout, overflow;
  reg [n-1:0] S;
  reg carryout, overflow;
  reg [n:0] Sum;

  always @(X or Y or carryin)
  begin
    Sum = {1'b0,X} + {1'b0,Y} + carryin;
    S = Sum[n-1:0];
    carryout = Sum[n];
    overflow = carryout ^ X[n-1] ^ Y[n-1] ^ S[n-1];
  end

endmodule

```

32. An alternative specification of an n -bit adder with carry-out and overflow signals.

```

module addern (carryin, X, Y, S, carryout, overflow);
  parameter n = 32;
  input carryin;
  input [n-1:0] X, Y;
  output [n-1:0] S;
  output carryout, overflow;
  reg [n-1:0] S;
  reg carryout, overflow;

  always @(X or Y or carryin)
  begin
    {carryout, S} = X + Y + carryin;
    overflow = carryout ^ X[n-1] ^ Y[n-1] ^ S[n-1];
  end

endmodule

```

Figure 5.33. Simplified complete specification of an n -bit adder.

```

module fulladd (Cin, x, y, s, Cout);
  input Cin, x, y;
  output s, Cout;
  reg s, Cout;

  always @(x or y or Cin)
    {Cout, s} = x + y + Cin;

endmodule

```

Figure 5.34. Behavioral specification of a full-adder.

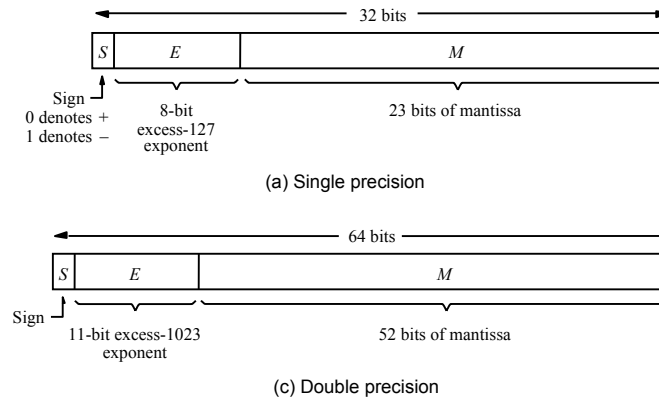


Figure 5.38. IEEE standard floating-point formats.

Decimal digit	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Table 5.3. Binary-coded decimal digits.

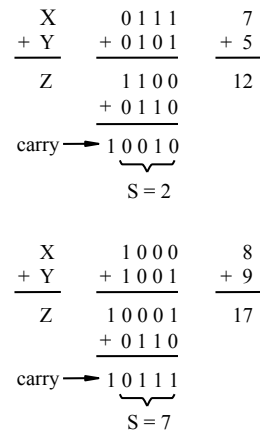


Figure 5.39. Addition of BCD digits.

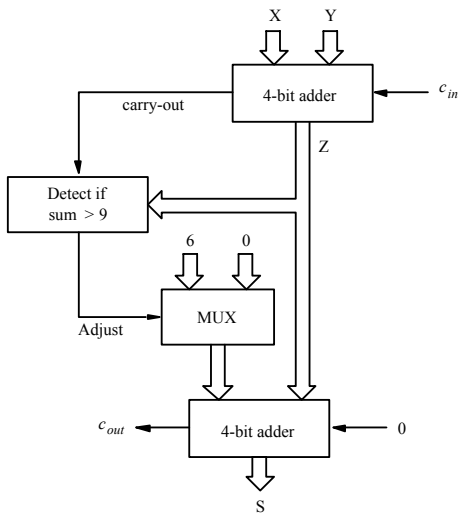


Figure 5.40. Block diagram for a one-digit BCD adder.

```

module bcdadd (Cin,X, Y, S, Cout);
input Cin;
input [3:0] X,Y;
output [3:0] S;
output Cout;
reg [3:0] S;
reg Cout;
reg [4:0] Z;

always@ (X or Y or Cin)
begin
    Z = X + Y + Cin;
    if (Z < 10)
        {Cout,S} = Z;
    else
        {Cout,S} = Z + 6;
end

endmodule

```

Figure 5.41. Verilog code for a one-digit BCD adder.

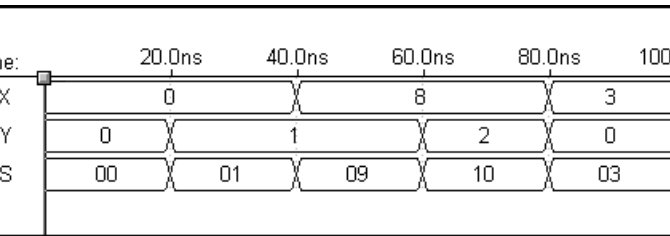


Figure 5.42. Functional simulation of the Verilog code in Figure 5.41.

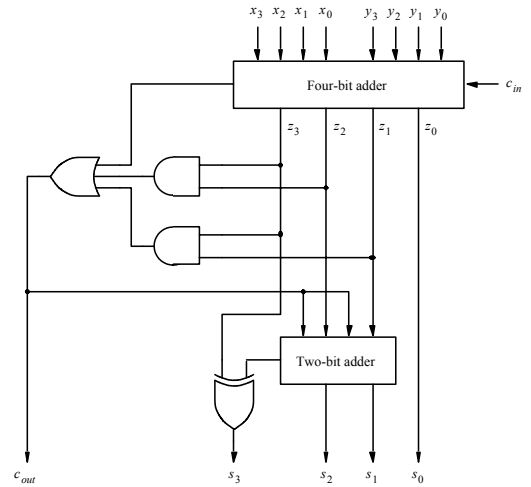


Figure 5.43. Circuit for a one-digit BCD adder.

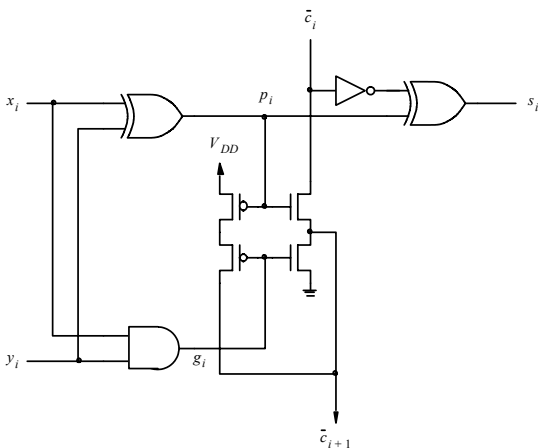


Figure P5.1. Circuit for problem 5.11.

```

module problem5_17 (IN, OUT);
input [3:0] IN;
output [3:0] OUT;
reg [3:0] OUT;

always @(IN)
    if (IN == 4'b0101) OUT = 4'b0001;
    else if (IN == 4'b0110) OUT = 4'b0010;
    else if (IN == 4'b0111) OUT = 4'b0011;
    else if (IN == 4'b1001) OUT = 4'b0010;
    else if (IN == 4'b1010) OUT = 4'b0100;
    else if (IN == 4'b1011) OUT = 4'b0110;
    else if (IN == 4'b1101) OUT = 4'b0011;
    else if (IN == 4'b1110) OUT = 4'b0110;
    else if (IN == 4'b1111) OUT = 4'b1001;
    else OUT = 4'b0000;

endmodule

```

Figure P5.2. The code for problem 5.17.

<i>A B</i>	<i>Carry</i>	<i>Sum</i>
0 0	0	0
0 1	0	1
0 2	0	2
1 0	0	1
1 1	0	2
1 2	1	0
2 0	0	2
2 1	1	0
2 2	1	1

Figure P5.3. Ternary half-adder.