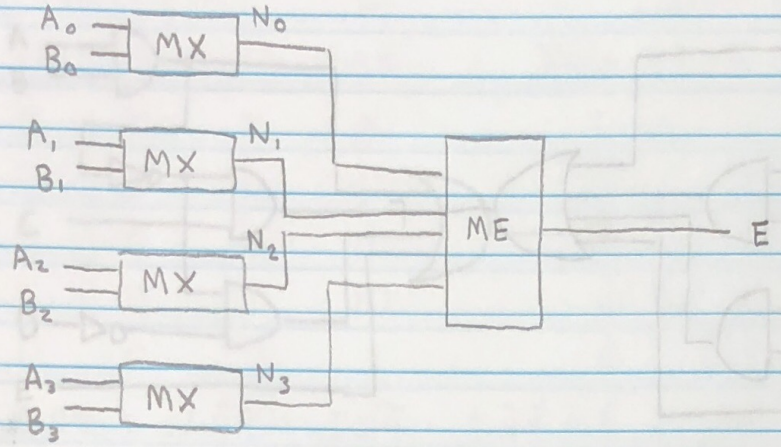
Stewart Dulaney
CS 42
Section 4105
SID: 1545566

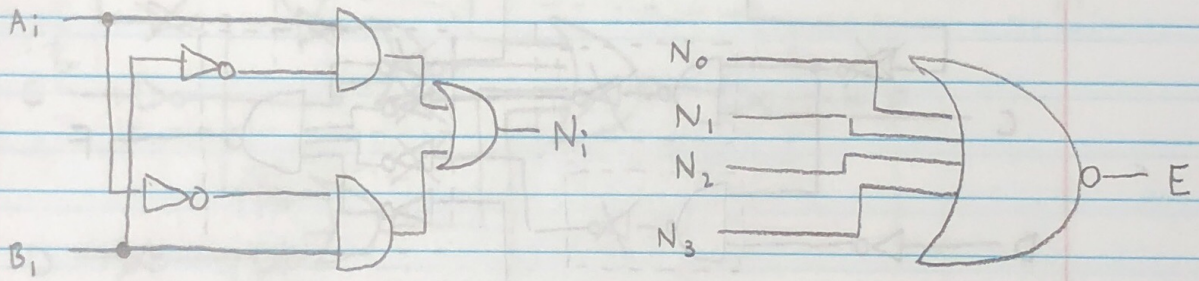
Chapter 3 Problems

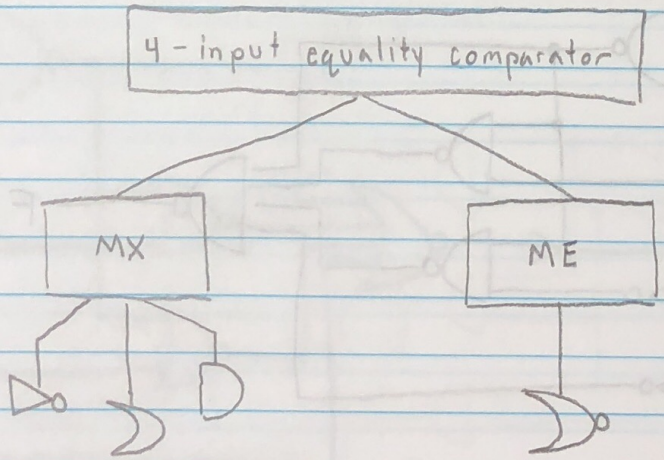
3-1 Design of a 4-Bit Equality Comparator



MX $N_i = \bar{A}_i B_i + A_i \bar{B}_i$

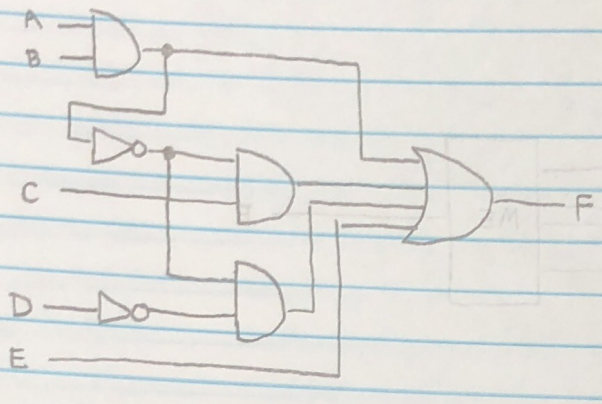
ME $E = \overline{N_0 + N_1 + N_2 + N_3}$

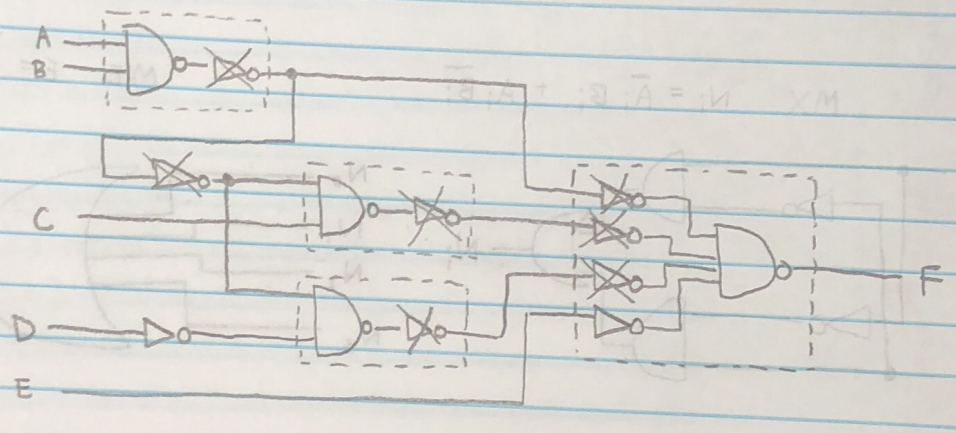


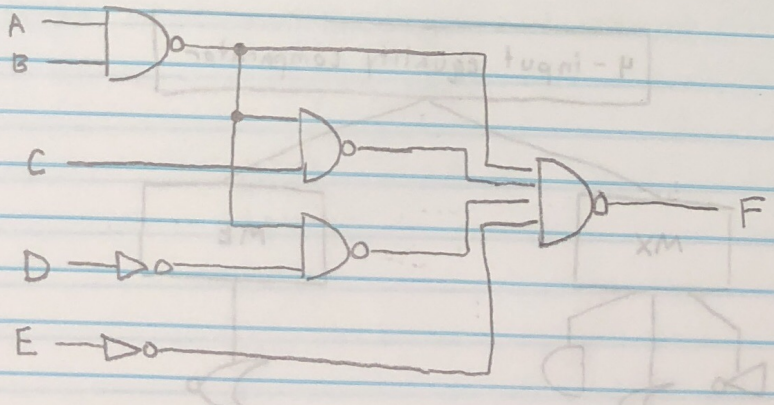


3-2 Implementation with NAND Gates

$$F = AB + \overline{(AB)}\,C + \overline{(AB)}\,\bar{D} + E$$

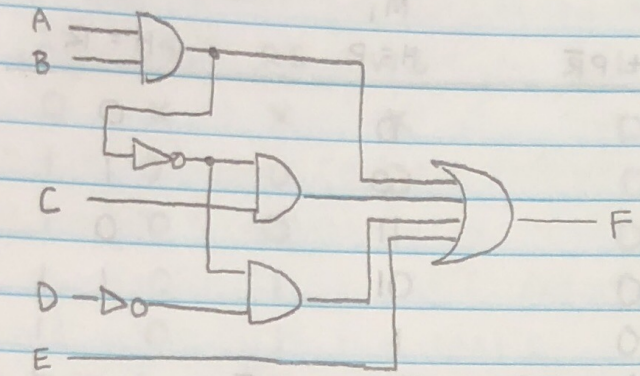


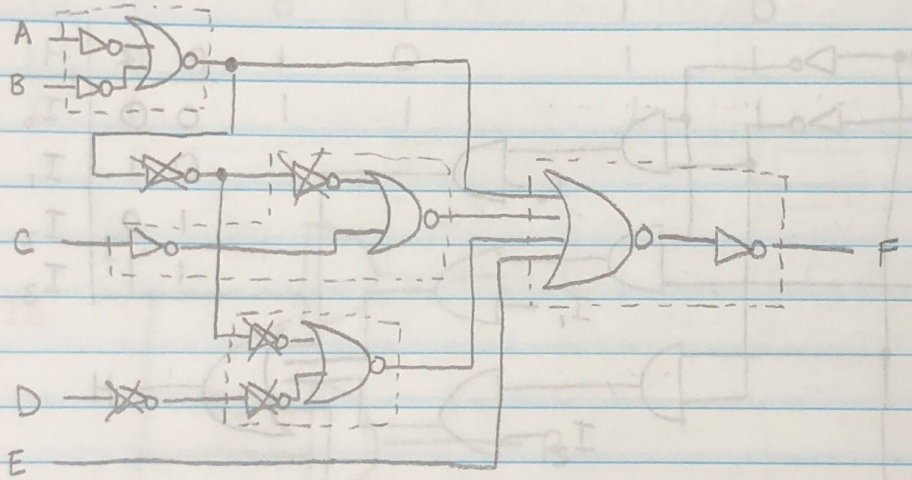


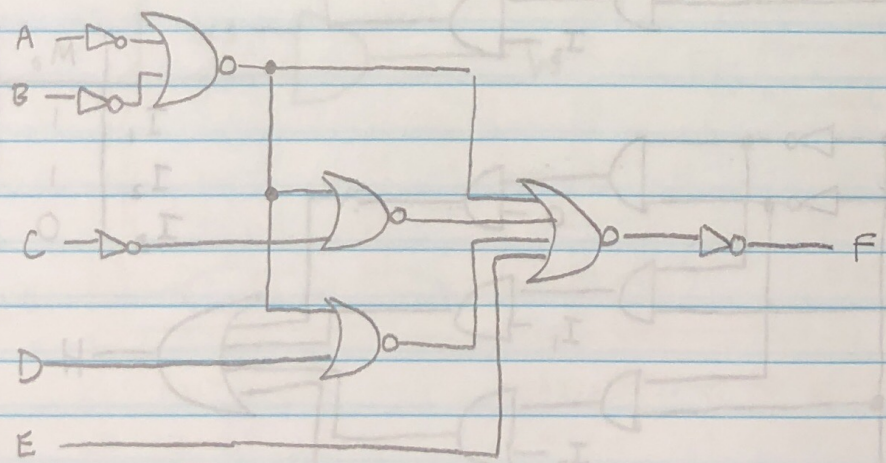


3-3 Implementation with NOR Gates

$F = AB + \overline{(AB)}C + \overline{(AB)}\bar{D} + E$





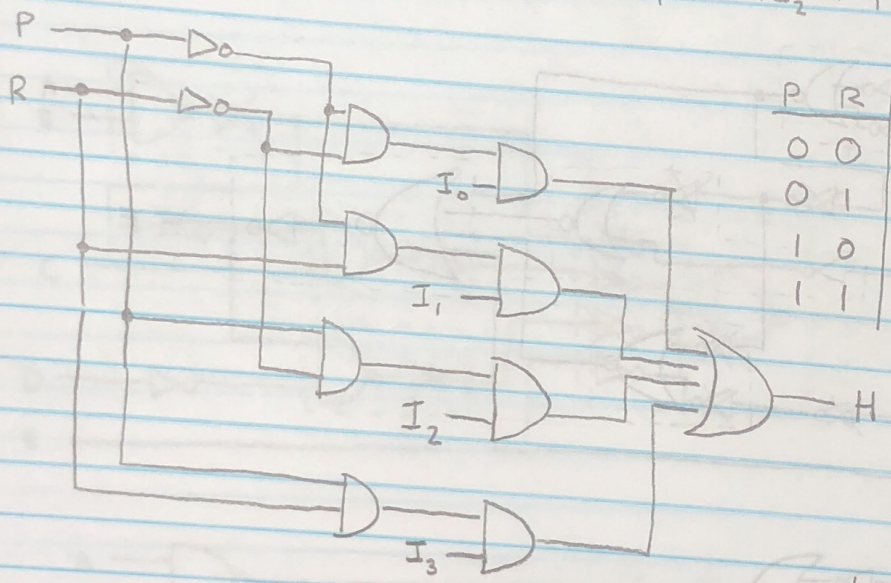


3-4

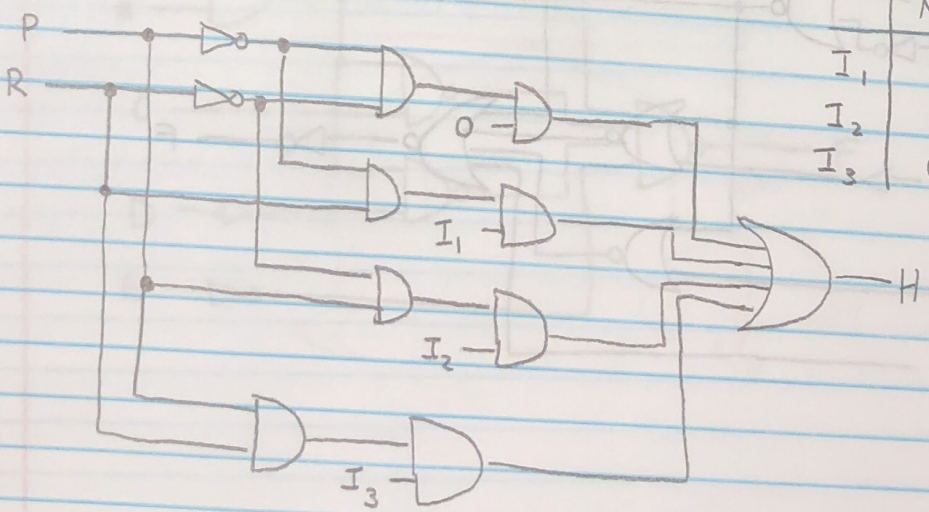
Lecture - Hall Lighting Control Using Value Fixing

Mode:		M_0	M_1	M_2
P	R	$H = \bar{P}R + P\bar{R}$	$H = P$	$H = R$
0	0	0	0	0
0	1	1	0	1
1	0	1	1	0
1	1	0	1	1

$$H(P, R, I_0, I_1, I_2, I_3) = \bar{P}\bar{R}I_0 + \bar{P}RI_1 + P\bar{R}I_2 + PRI_3$$



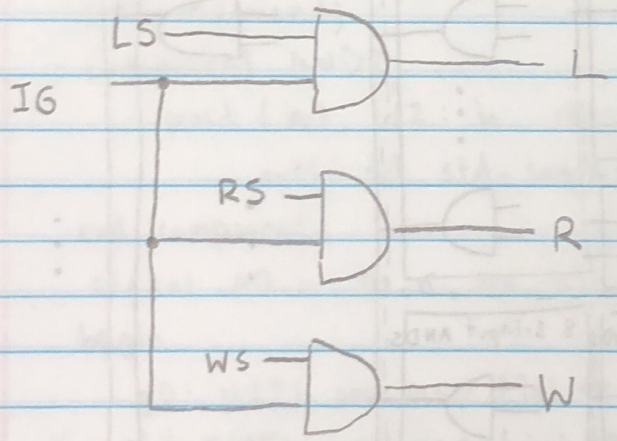
P	R	H
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



	M_0	M_1	M_2
I_1	1	0	1
I_2	1	1	0
I_3	0	1	1

3-5 Car Electrical Control Using Enabling

Input Switches				Accessory Control		
IS	LS	RS	WS	L	R	W
0	X	X	X	0	0	0
1	0	0	0	0	0	0
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	0	1	1
1	1	0	0	1	0	0
1	1	0	1	1	0	1
1	1	1	0	1	1	0
1	1	1	1	1	1	1



3-6 6-to-64-Line Decoder

Typo in text
For output size?

Step 1) $k = n = 6$

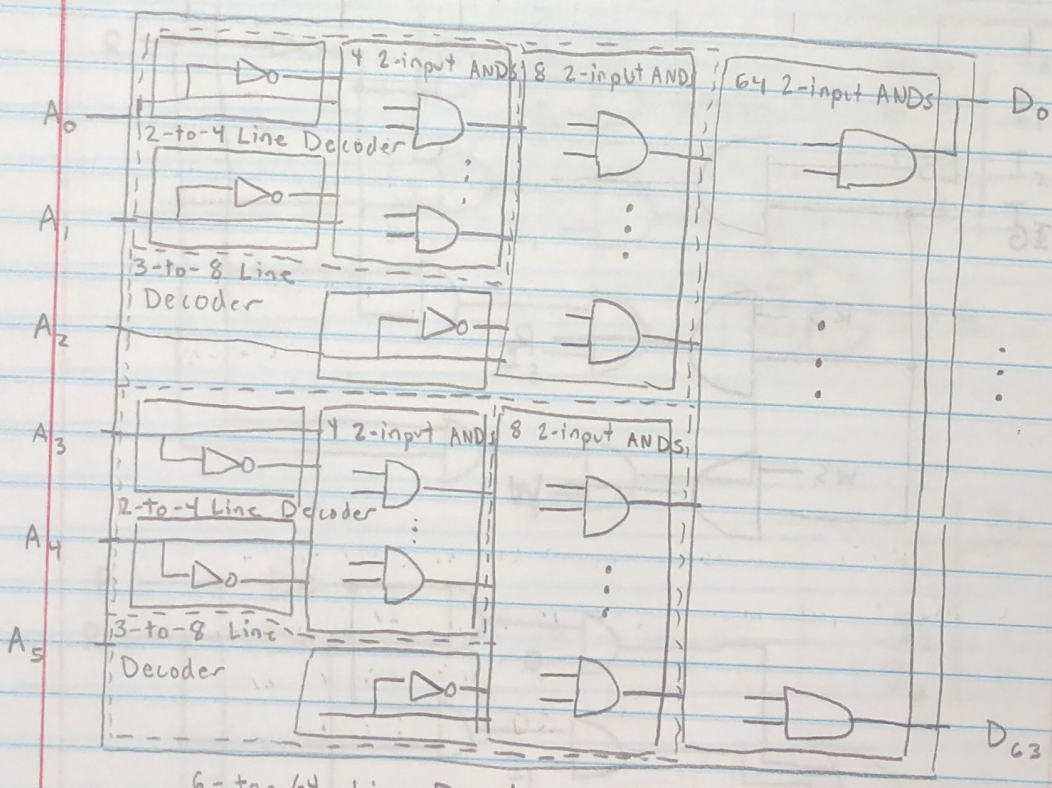
Step 2) k is even $\rightarrow$ $k/2 = 6/2 = 3$. Use $2^6 = 64$ AND gates driven by two decoders of output size $2^k = 2^3 = 8$.

Step 1) $k = n = 3$

Step 2) k is odd $\rightarrow$ $(k+1)/2 = 4/2 = 2$ and $(k-1)/2 = 2/2 = 1$. Use $2^3 = 8$ AND gates driven by a decoder of output size $2^2 = 4$ and a decoder of output size $2^1 = 2$.

Step 1) $k = n = 2$

Step 2) k is even $\rightarrow$ $k/2 = 2/2 = 1$. Use $2^2 = 4$ AND gates driven by two decoders of output size $2^k = 2^1 = 2$.



6-to-64 Line Decoder

3-7 VHDL Models for a 2-to-4-Line Decoder

```
-- 2-to-4-Line Decoder with Enable: Structural VHDL Description
library ieee, ledf_vhdl;
use ieee.std_logic_1164.all, ledf_vhdl.func_prims.all;
entity decoder_2_to_4_w_enable is
    port (EN, A0, A1: in std_logic;
    D0, D1, D2, D3: out std_logic);
end decoder_2_to_4_w_enable;

architecture structural_1 of decoder_2_to_4_w_enable is
  component NOT1
    port ( in1: in std_logic;
           out1: out std_logic);
  end component;
  component AND2
    port (in1, in2: in std_logic;
          out1: out std_logic);
  end component;
  signal A0_n, A1_n, N0, N1, N2, N3: std_logic;
  begin
    g0: NOT1 port map (in1 => A0, out1 => A0_n);
    g1: NOT1 port map (in1 => A1, out1 => A1_n);
    g2: AND2 port map (in1 => A0_n, in2 => A1_n, out1 => N0);
    g3: AND2 port map (in1 => A0, in2 => A1_n, out1 => N1);
    g4: AND2 port map (in1 => A0_n, in2 => A1, out1 => N2);
    g5: AND2 port map (in1 => A0, in2 => A1, out1 => N3);
    g6: AND2 port map (in1 => EN, in2 => N0, out1 => D0);
    g7: AND2 port map (in1 => EN, in2 => N1, out1 => D1);
    g8: AND2 port map (in1 => EN, in2 => N2, out1 => D2);
    g9: AND2 port map (in1 => EN, in2 => N3, out1 => D3);
end structural_1;
```

3-8

Verilog Models for a 2-to-4-Line Decoder

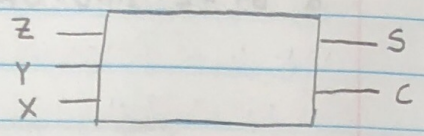
```
// 2-to-4-Line Decoder with Enable: Structural Verilog Desc.
module decoder_2_to_4_st_v (EN, A0, A1, D0, D1, D2, D3);
  input EN, A0, A1;
  output D0, D1, D2, D3;

  wire A0_n, A1_n, N0, N1, N2, N3;
  not
    g0 (A0_n, A0);
    g1 (A1_n, A1);
  and
    g3 (N0, A0_n, A1_n);
    g4 (N1, A0, A1_n);
    g5 (N2, A0_n, A1);
    g6 (N3, A0, A1);
    g7 (D0, N0, EN);
    g8 (D1, N1, EN);
    g9 (D2, N2, EN);
    g10 (D3, N3, EN);
endmodule

// 2-to-4-Line Decoder with Enable: Dataflow Verilog Desc.
module decoder_2-to-4_df_v (EN, A0, A1, D0, D1, D2, D3);
    input EN A0, A1;
    output D0, D1, D2, D3;

    assign D0 = EN & ~A1 & ~A0;
    assign D1 = EN & ~A1 & A0;
    assign D2 = EN & A1 & ~A0;
    assign D3 = EN & A1 & A0;
endmodule
```

3-7 cont'd

```
-- 2-to-4-Line Decoder: Dataflow VHDL Description
-- Use library, use, and entity entries from 2_to_4_decoder_st;

architecture dataflow_1 of decoder_2_to_4_w_enable is

signal A0_n, A1_n : std_logic;
begin
   A0_n <= not A0;
   A1_n <= not A1;
   D0 <= A0_n and A1_n and EN;
   D1 <= A0 and A1_n and EN;
   D2 <= A0_n and A1 and EN;
   D3 <= A0 and A1 and EN;
end dataflow_1;
```

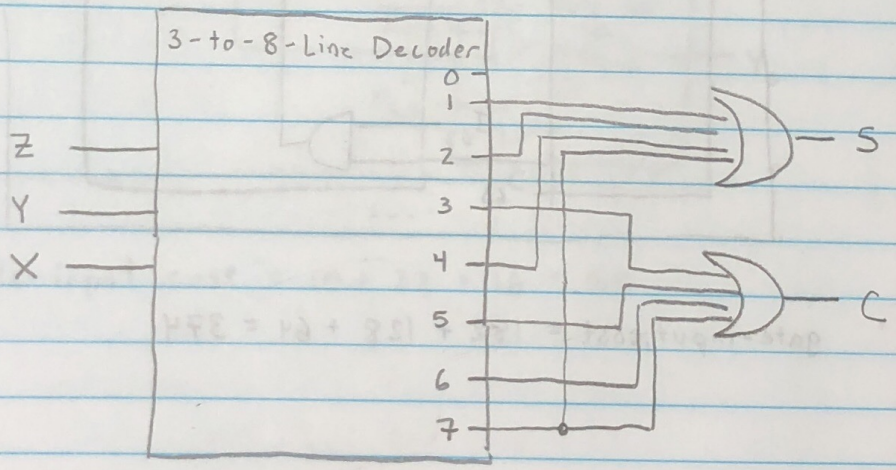
3-9 Decoder and OR-Gate Implementation of a Binary Adder Bit

	X	Y	Z	C	S
m_0	0	0	0	0	0
m_1	0	0	1	0	1
m_2	0	1	0	0	1
m_3	0	1	1	1	0
m_4	1	0	0	0	1
m_5	1	0	1	1	0
m_6	1	1	0	1	0
m_7	1	1	1	1	1



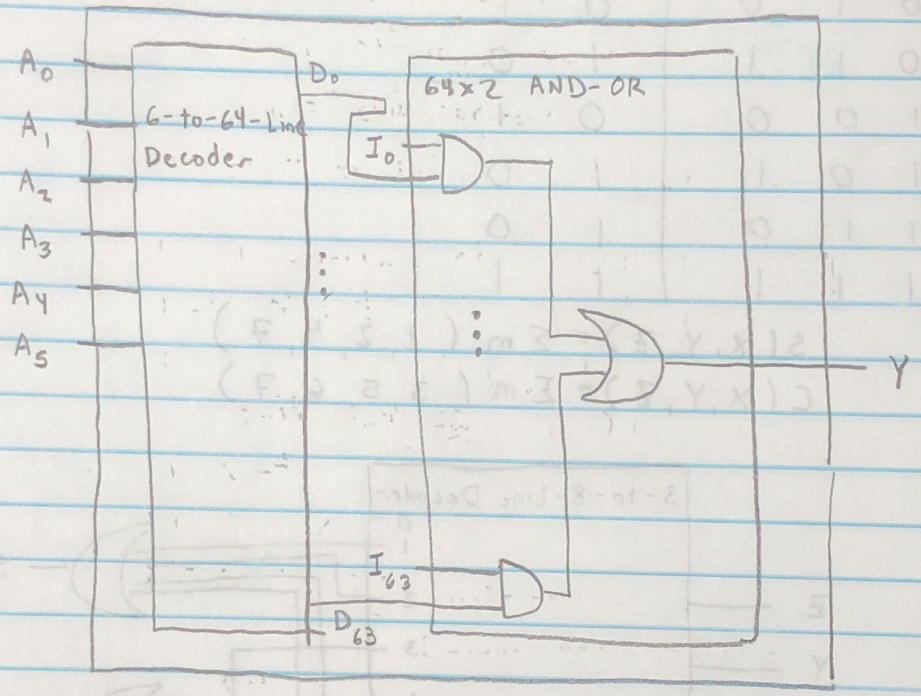
$S(X, Y, Z) = \Sigma m\,(1, 2, 4, 7)$

$C(X, Y, Z) = \Sigma m\,(3, 5, 6, 7)$



3-10 64-to-1-Line Multiplexer

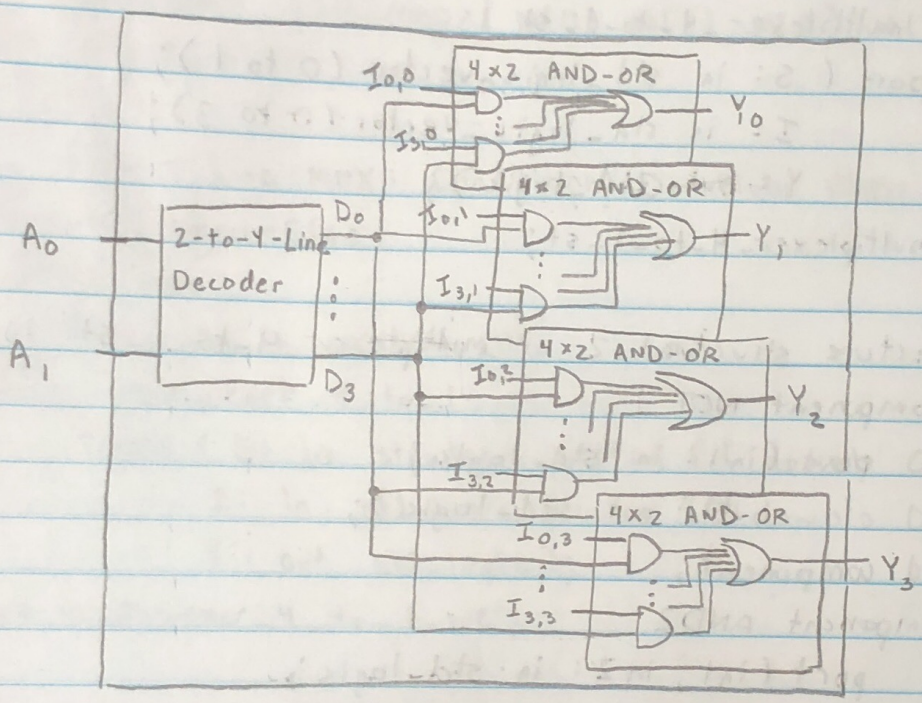
$n = 6$. This will require a 6-to-64-line decoder and a 64×2 AND-OR gate.



gate-input cost = 182 + 128 + 64 = 374

3-11 4-to-1-Line Quad Multiplexer

$n = 2$. The implementation requires a 2-to-4-Line decoder and four 4×2 AND-OR gates.



gate-input cost = 10 + 32 + 16 = 58

3-12

VHDL Models for a 4-to-1 Multiplexer

```
-- 4-to-1-Line Multiplexer: Structural VHDL Description
library ieee, ledf_vhdl;
use ieee.std_logic_1164.all, ledf_vhdl.func_prims.all;
entity multiplexer_4_to_1_st is
    port ( S: in std_logic_vector (0 to 1);
           I: in std_logic_vector (0 to 3);
           Y: out std_logic);
end multiplexer_4_to_1_st;

architecture structual_2 of multiplexer_4_to_1_st is
  component NOT1
      port (in1: in std_logic;
            out1: out std_logic);
  end component;
  component AND2
      port (in1, in2: in std_logic;
            out1: out std_logic);
  end component;
  component OR4
      port (in1, in2, in3, in4: in std_logic;
            out1: out std_logic);
  end component;
  signal S_n: std_logic_vector (0 to 1);
  signal D, N: std_logic_vector (0 to 3);
begin
   g0: NOT1 port map (S(0), S_n(0));
   g1: NOT1 port map (S(1), S_n(1));
   g2: AND2 port map (S_n(1), S_n(0), D(0));
   g3: AND2 port map (S_n(1), S(0), D(1));
   g4: AND2 port map (S(1), S_n(0), D(2));
   g5: AND2 port map (S(1), S(0), D(3));
```

3-12 cont'd

```
    g6: AND2 port map (D(0), I(0), N(0));
    g7: AND2 port map (D(1), I(1), N(1));
    g8: AND2 port map (D(2), I(2), N(2));
    g9: AND2 port map (D(3), I(3), N(3));
    g10: OR4 port map (N(0), N(1), N(2), N(3), Y);
end structural_2;

-- 4-to-1-Line MUX: Conditional Dataflow VHDL Description
-- Using When-Else
library ieee;
use ieee.std_logic_1164.all;
entity multiplexer_4_to_1_we is
    port (S: in std_logic_vector (1 downto 0);
          I: in std_logic_vector (3 downto 0);
          Y: out std_logic);
end multiplexer_4_to_1_we;

architecture function_table of multiplexer_4_to_1_we is
begin
    Y <= I(0) when S = "00" else
         I(1) when S = "01" else
         I(2) when S = "10" else
         I(3) when S = "11" else
         'X';
end function_table;
```

3-13 Verilog Models For a 4-to-1-Line Multiplexer

```
// 4-to-1-Line Multiplexer: Structural Verilog Description
module multiplexer_4_to_1_st_v ( S, I, Y);
   input [1:0] S;
   input [3:0] I;
   output Y;

   wire [1:0] not_S;
   wire [0:3] D, N;

not
    gn0 (not_S[0], S[0]);
    gn1 (not_S[1], S[1]);

and
   g0 (D[0], not_S[1], not_S[0]),
   g1 (D[1], not_S[1], S[0]),
   g2 (D[2], S[1], not_S[0]),
   g3 (D[3], S[1], S[0]),
   g0 (N[0], D[0], I[0]),
   g1 (N[1], D[1], I[1]),
   g2 (N[2], D[2], I[2]),
   g3 (N[3], D[3], I[3]);

or
   go (Y, N[0], N[1], N[2], N[3]);

endmodule
```

3-12 cont'd

```
-- 4-to-1-Line Mux: Conditional DataFlow VHDL Description
-- Using with Select
library ieee;
use ieee.std_logic_1164.all;
entity multiplexer_4_to_1_ws is
    port ( S: in std_logic_vector (1 downto 0);
           I: in std_logic_vector (3 downto 0);
           Y: out std_logic);
end multiplexer_4_to_1_ws;

architecture function_table_ws of multiplexer_4_to_1_ws is
begin
    with S select
      Y <= I(0) when "00",
           I(1) when "01",
           I(2) when "10",
           I(3) when "11",
           'X' when others;
end function_table_ws;
```

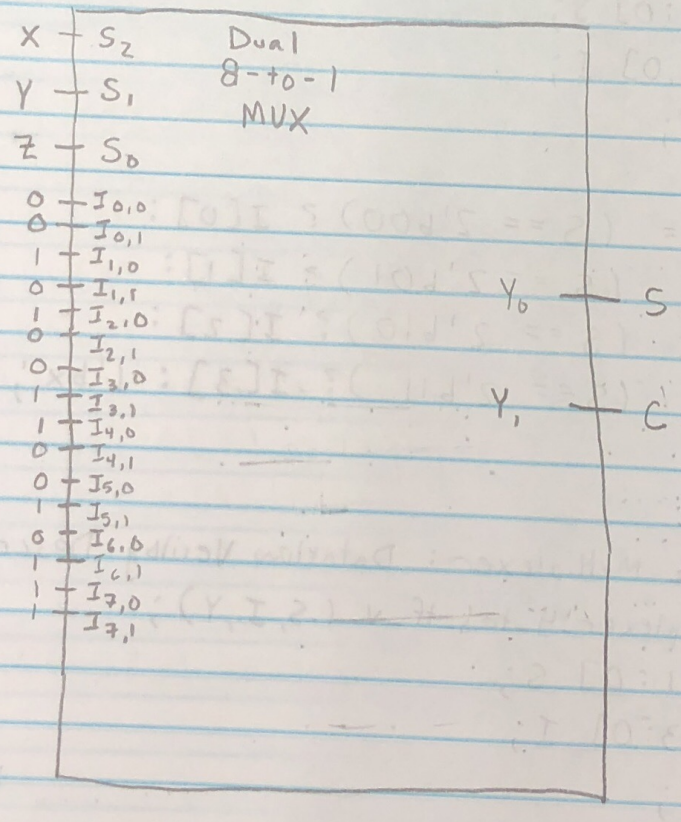
3-13 cont'd

```
// 4-to-1 Line Multiplexer: Dataflow Verilog Description
module multiplexer_4-to-1_dF_v (S, I, Y);
    input [1:0] S;
    input [3:0] I;
    output Y;

    assign Y = (~S[1] & ~S[0] & I[0]) | (~S[1] &
            S[0] & I[1]) | (S[1] & ~S[0] & I[2]) |
            (S[1] & S[0] & I[3]);
end module

// 4-to-1 Line Multiplexer: Dataflow Verilog Description
module multiplexer_4_to_1_cP_v (S,I,Y);
    input [1:0] S;
    input [3,0] I;
    output Y;

    assign Y = (S == 2'b00) ? I[0]:
               (S == 2'b01) ? I[1]:
               (S == 2'b10) ? I[2]:
               (S == 2'b11 ) ? I[3]: 1'bx;
end module

// 4-to-1 Line Multiplexer: Dataflow Verilog Description
module multiplexer_4_to_1_tf_v (S, I, Y);
    input [1:0] S;
    input [3:0] I;
    output Y;

    assign Y = S[1] ? (S[0] & I[3] : I[2]):
               (S[0] ? I[1] : I[0]);
end module
```

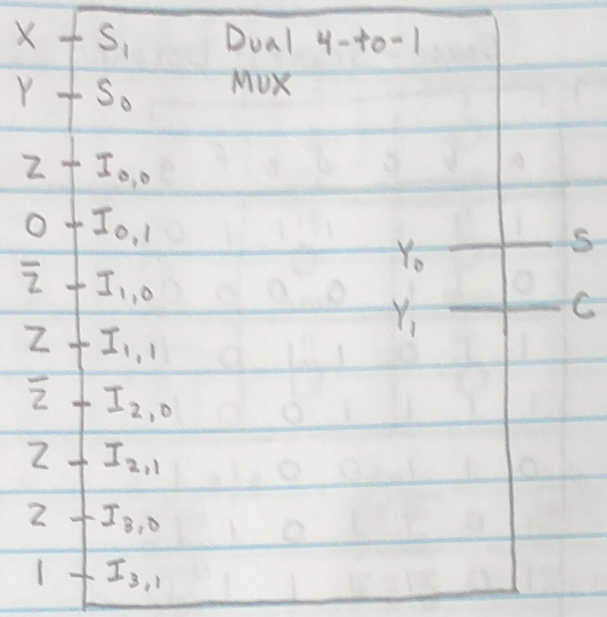
3-14 Security System Sensor Selection using Multiplexers

Some of the sensors can be connected to multiplexer inputs and some directly to microcontroller inputs. One possible solution that minimizes the number of multiplexers is to use two 8-to-1 multiplexers, each connected to a microcontroller input. The two multiplexers handle 16 sensors and require three microcontroller outputs as selection inputs. Since there are 15 sensor outputs, the unused 16th multiplexer input can be attached to 0. The number of microcontroller input/outputs used is $2+3=5$.

3-15 Multiplexer Implementation of a Binary Adder Bit



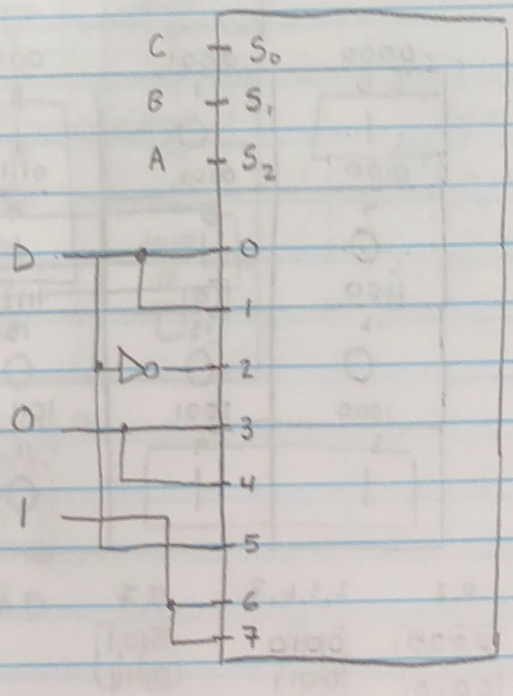
3-16 Alternative Multiplexer Implementation of a Binary Adder Bit



3-17 Multiplexer Implementation of 4-Variable Function

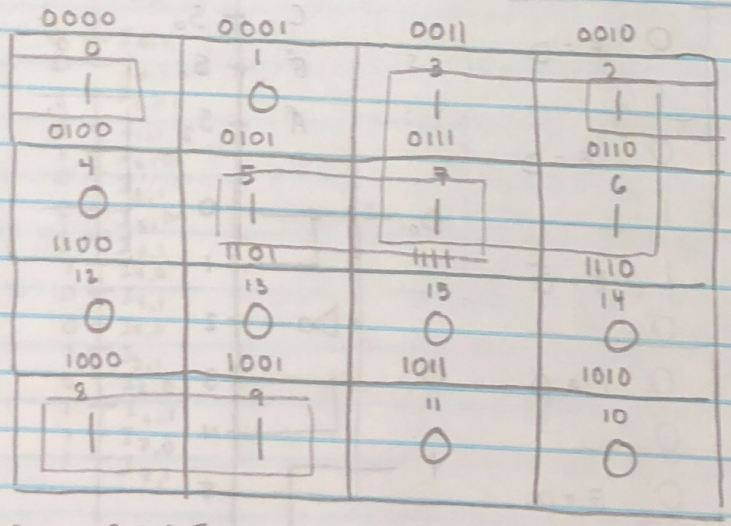
$F(A,B,C,D) = \Sigma m\,(1, 3, 4, 11, 12, 13, 14, 15)$

	A	B	C	D	F	
m_0	0	0	0	0	0	F = D
m_1	0	0	0	1	1	
m_2	0	0	1	0	0	F = D
m_3	0	0	1	1	1	
m_4	0	1	0	0	1	$F = \bar{D}$
m_5	0	1	0	1	0	
m_6	0	1	1	0	0	F = 0
m_7	0	1	1	1	0	
m_8	1	0	0	0	0	F = 0
m_9	1	0	0	1	0	
m_{10}	1	0	1	0	0	F = D
m_{11}	1	0	1	1	1	
m_{12}	1	1	0	0	1	F = 1
m_{13}	1	1	0	1	1	
m_{14}	1	1	1	0	1	F = 1
m_{15}	1	1	1	1	1	



3-18 Design of a BCD-to-Seven-Segment Decoder

	BCD Input				Seven-Segment Decoder						
	A	B	C	D	a	b	c	d	e	f	g
m_0	0	0	0	0	1	1	1	1	1	1	0
m_1	0	0	0	1	0	1	1	0	0	0	0
m_2	0	0	1	0	1	1	0	1	1	0	1
m_3	0	0	1	1	1	1	1	1	0	0	1
m_4	0	1	0	0	0	1	1	0	0	1	1
m_5	0	1	0	1	1	0	1	1	0	1	1
m_6	0	1	1	0	1	0	1	1	1	1	1
m_7	0	1	1	1	1	1	1	0	0	0	0
m_8	1	0	0	0	1	1	1	1	1	1	1
m_9	1	0	0	1	1	1	1	1	0	1	1
All other inputs					0	0	0	0	0	0	0



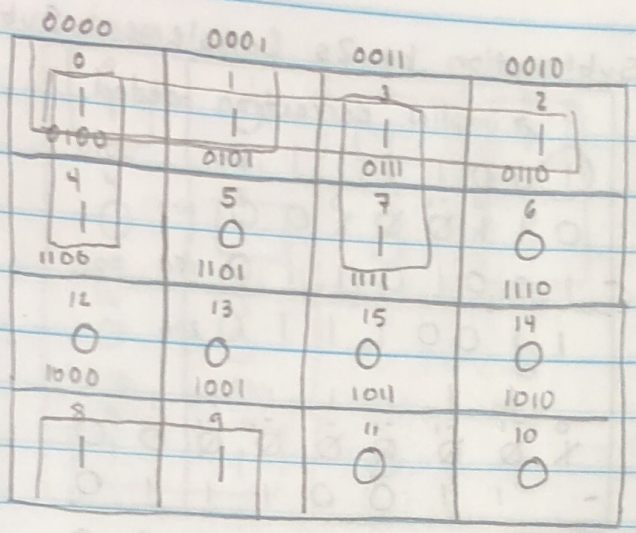
16 8 4 2 1

0,2 2,3,6,7 5,7 8,9

$\bar{A}\bar{B}\bar{D}$ $\bar{A}\bar{C}$ $\bar{A}BD$ $A\bar{B}\bar{C}$

$a = \bar{A}C + \bar{A}BD + \bar{A}\bar{B}\bar{D} + A\bar{B}\bar{C}$

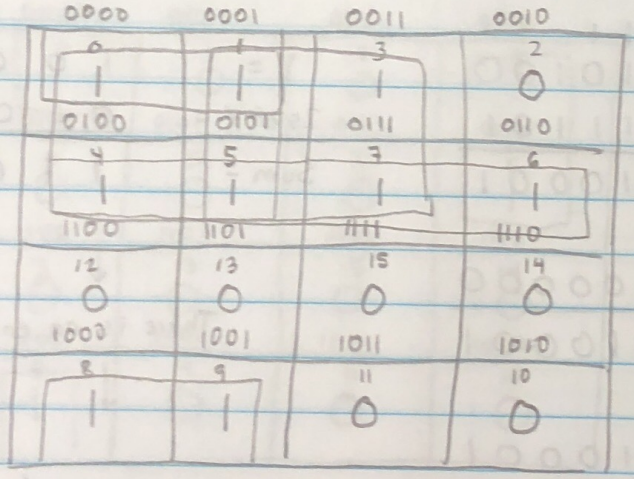
3-18 cont'd



16 8 4 2 1

0,1,2,3 → $\bar{A}\bar{B}$ 　 0,1,8,9 → $\bar{B}\bar{C}$ 　 0,4 → $\bar{A}\bar{C}\bar{D}$ 　 3,7 → $\bar{A}CD$

$b = \bar{A}\bar{B} + \bar{B}\bar{C} + \bar{A}\bar{C}\bar{D} + \bar{A}CD$



16 8 4 2 1

0,1,8,9 → $\bar{B}\bar{C}$ 　 4,5,6,7 → $\bar{A}B$ 　 1,3,5,7 → $\bar{A}D$

$c = \bar{B}\bar{C} + \bar{A}B + \bar{A}D$

3-19 Unsigned Binary Subtraction by 2s Complement Subtract

↗ implies correction needed

Borrows into: (1) 0 0 1 1 1 1 0

Minuend: 0 1 $\overset{0}{\cancel{1}}$ $\overset{-1}{\cancel{0}}$ $\overset{-1}{\cancel{0}}$ $\overset{0}{\cancel{1}}$ 0 0 100

Subtrahend: − 1 0 0 1 0 1 1 0 150

1 1 0 0 1 1 1 0 206

1 1 1 1 1 1 1 1

2^8 $\overset{0}{\cancel{1}}$ $\overset{-1}{\cancel{0}}$ $\overset{-1}{\cancel{0}}$ $\overset{-1}{\cancel{0}}$ $\overset{-1}{\cancel{0}}$ $\overset{-1}{\cancel{0}}$ $\overset{-1}{\cancel{0}}$ 0 0

Initial Result − 1 1 0 0 1 1 1 0

Final Result (−) 0 0 1 1 0 0 1 0

↘ result is negative

3-20 Unsigned Binary Subtraction by 2s Complement Addition

X = 1010100, Y = 1000011

1 1 1 1

X = 1 0 1 0 1 0 0

2s of Y = + 0 1 1 1 1 0 1

Sum = (1) 0 0 1 0 0 0 1

↗ end carry

Discard end carry 2^7 − 1 0 0 0 0 0 0 0

0 0 0 1 0 0 0 1

Answer: X − Y 0 0 1 0 0 0 1

Y = 1 0 0 0 0 1 1

2s of X = 0 1 0 1 1 0 0

Sum = 1 1 0 1 1 1 1

↓

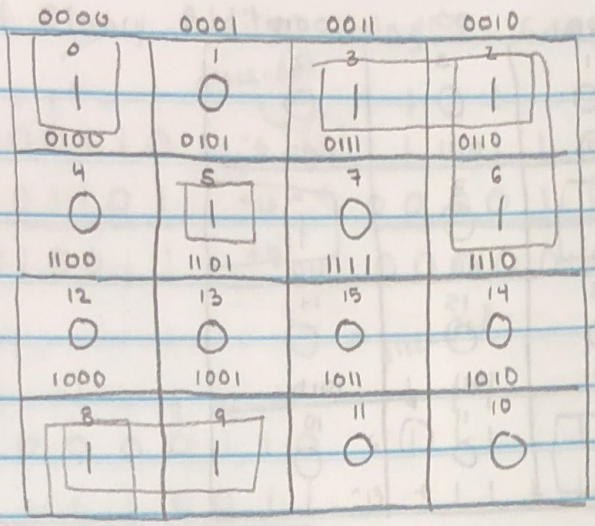
There is no end carry

↓

Answer: Y − X = − (2s complement of 1101111)

= − 0 0 1 0 0 0 1

3-18 cont'd



16 8 4 2 1

0,8 → 0000, 1000 → $\bar{B}\bar{C}\bar{D}$

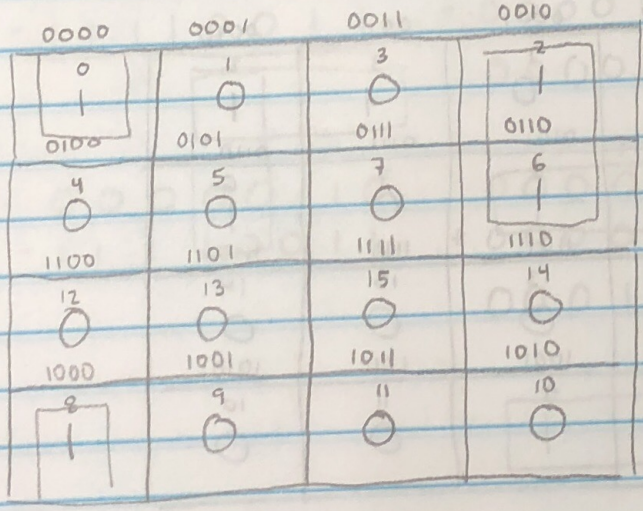
5 → 0101 → $\bar{A}B\bar{C}D$

2,3 → 0010, 0011 → $\bar{A}\bar{B}C$

2,6 → 0010, 0110 → $\bar{A}C\bar{D}$

8,9 → 1000, 1001 → $A\bar{B}\bar{C}$

$d = \bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}\bar{B}C + \bar{A}C\bar{D} + A\bar{B}\bar{C}$

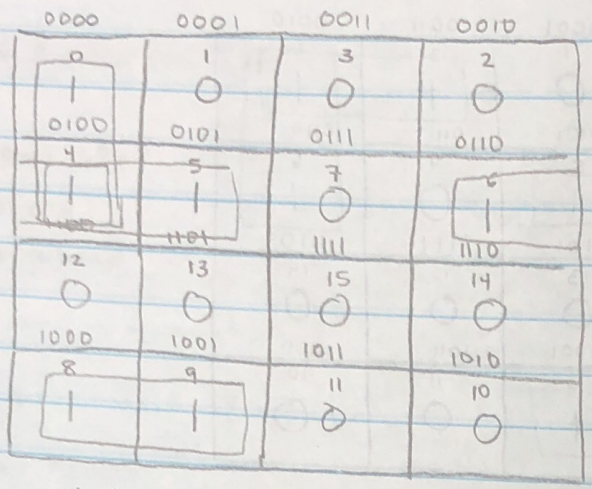


16 8 4 2 1

0,8 → 0000, 1000 → $\bar{B}\bar{C}\bar{D}$

2,6 → 0010, 0110 → $\bar{A}C\bar{D}$

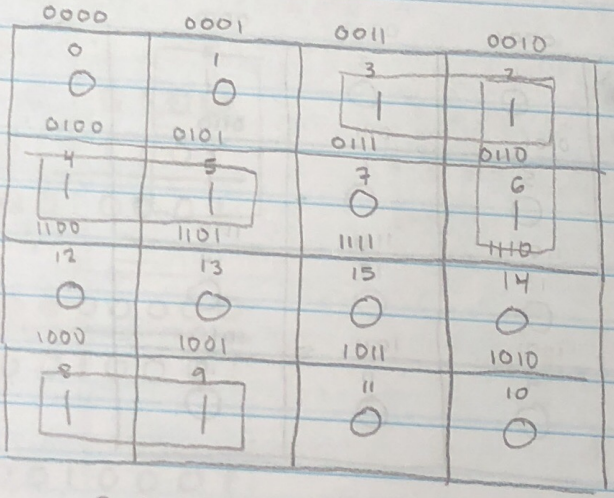
$e = \bar{B}\bar{C}\bar{D} + \bar{A}C\bar{D}$



16 8 4 2

0,4 → $\bar{A}\bar{C}\bar{D}$; 4,5 → $\bar{A}B\bar{C}$; 4,6 → $\bar{A}B\bar{D}$; 8,9 → $A\bar{B}\bar{C}$

$f = \bar{A}\bar{C}\bar{D} + \bar{A}B\bar{C} + \bar{A}B\bar{D} + A\bar{B}\bar{C}$



16 8 4 2 1

2,3 → $\bar{A}\bar{B}C$; 2,6 → $\bar{A}C\bar{D}$; 4,5 → $\bar{A}B\bar{C}$; 8,9 → $A\bar{B}\bar{C}$

$g = \bar{A}\bar{B}C + \bar{A}C\bar{D} + \bar{A}B\bar{C} + A\bar{B}\bar{C}$

3-21 Signed Binary Addition Using 2s Complement

```
+6    00000110         -6   (1)11111010   ← discarded
+13   00001101         +13  + 00001101
+19   00010011         +7     00000111
```

```
                                discarded
+6      00000110       -6   (1)11111010
-13   + 11110011       -13  + 11110011
-7      11111001       -19    11101101
```

3-22 Signed Binary Subtraction Using 2s Complement

```
                                 discarded
-6         11111010            (1)11111010
-(-13)   - 11110011              + 00001101
+7                                 00000111
```

```
+6         00000110              00000110
-(-13)   - 11110011            + 00001101
+19                              00010011
```

3-23 Electronic Scale Feature

a) What arithmetic logic is required?

A: The scale is mensuring the gross weight. The displayed result is the net weight. So a subtractor is needed to form:

Net Weight = Gross Weight - (stored) Tare Weight

b) How many bits are required for the operands, assuming the gross weight capacity of the scale is 2200 grams with one gram as the smallest unit?

Assuming the weights and the subtraction are in binary, 12 bits are required to represent 2200 grams.

□ Why? → IF they are represented in BCD, then $2 + 3 \times 4 = 14$ bits are required.

3-24 Hierarchical VHDL for a 4-Bit Ripple Carry Adder

```
-- 4-bit Adder: Hierarchical Dataflow/Structural
library ieee;
use ieee.std_logic_1164.all;
entity half_adder is
    port (x, y: in std_logic;
          s, c: out std_logic);
end half_adder;

architecture dataflow_3 of half_adder is
    begin
        s <= x xor y;
        c <= x and y;
end dataflow_3;

library ieee;
use ieee.std_logic_1164.all;
entity full_adder is
    port (x, y, z: in std_logic;
          s, c: out std_logic);
end full_adder;
```

3-24 cont'd

```
architecture struc_dataflow_3 of Full_adder is
    component half_adder
    port ( x, y : in std_logic;
           s, c : out std_logic );
    end component;
    signal hs, hc, tc : std_logic;
    begin
      HA1 : half_adder
       port map (x, y, hs, hc );
      HA2 : half_adder
       port map ( hs, z, s, tc );
      c <= tc or hc;
end struc_dataflow_3;

library ieee;
use ieee.std_logic_1164.all;
entity adder_4 is
    port ( B, A : in std_logic_vector (3 downto 0);
           CO : in std_logic;
           S : out std_logic_vector (3 downto 0);
           C4 : out std_logic );
end adder_4;
```

3-25

Behavioral VHDL for a 4-Bit Ripple Carry Adder

```
-- 4-bit Adder: Behavioral Description
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity adder_4_b is
    port (B, A : in std_logic_vector (3 downto 0);
          CO : in std_logic;
          S : out std_logic_vector (3 downto 0);
          C4 : out std_logic);
end adder_4_b;

architecture behavioral of adder_4_b is
signal sum: std_logic_vector (4 downto 0);
begin
    sum <= ('0' & A) + ('0' & B) + ("0000" & CO);
    C4 <= sum (4);
    S <= sum (3 downto 0);
end behavioral;
```

3-24 cont'd

```
architecture structural_4 of adder_4 is
  component full_adder
    port (x, y, z : in std_logic;
          s, c : out std_logic);
  endcomponent;
  signal C : std_logic_vector (4 downto 0);
  begin
  Bit0 : full_adder
   port map ( B(0), A(0), C(0), S(0), C(1));
  Bit1 : full_adder
   port map ( B(1), A(1), C(1), S(1), C(2));
  Bit 2 : full_adder
   port map ( B(2), A(2), C(2), S(2), C(3));
  Bit3 : full_adder
   port map (B(3), A(3), C(3), S(3), C(4));
   C(0) <= CO;
   C4 <= C(4);
end structural_4;
```

3-26

Hierarchical Verilog for a 4-Bit Ripple Carry Adder

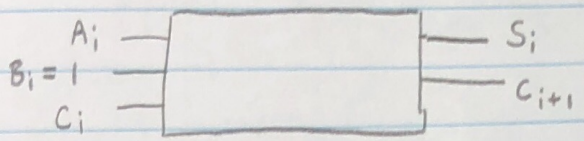
```
// 4-bit Adder: Hierarchical Dataflow / Structural
module half_adder_v (x, y, s, c);
    input x,y;
    output s,c;

    assign s = x ^ y;
    assign c = x & y;
endmodule

module full_adder_v (x,y,z,s,c);
    input x,y,z;
    output s,c;

    wire hs, hc, tc;

    half_adder_v HA1 (x,y, hs, hc),
                 HA2 (hs, z, s, tc);
    assign c = tc | hc;
endmodule

module adder_4_v (B,A, C0, S, C4);
    input [3:0] B, A;
    input C0;
    output [3:0] S;
    output C4;

    wire [3:1] C;
    full_adder_v Bit0 (B[0], A[0], C0, S[0], C[1]),
                 Bit1 (B[1], A[1], C[1], S[1], C[2]),
                 Bit2 (B[2], A[2], C[2], S[2], C[3]),
                 Bit3 (B[3], A[3], C[3], S[3], C4);

endmodule
```

3-27

Behavioral Verilog for a 4-Bit Ripple Carry Adder

```
// 4-bit Adder: Behavioral Verilog Description
module adder_4_b_v (A, B, CO, S, C4);
  input [3:0] A, B;
  input CO;
  output [3:0] S;
  output C4;

  assign {C4, S} = A + B + CO;
endmodule
```

3-28 Contraction of Full-Adder Equations



$$S_i = A_i \oplus B_i \oplus C_i$$
$$= A_i \oplus 1 \oplus C_i$$
$$= \overline{A_i \oplus C_i}$$

□ Why?

$$C_{i+1} = A_i B_i + A_i C_i + B_i C_i$$
$$= A_i \cdot 1 + A_i C_i + 1 \cdot C_i$$
$$= A_i (1 + C_i) + C_i$$
$$= A_i + C_i$$

Suppose this circuit is used in place of each of the four full adders in a 4-bit ripple carry adder. Instead of $S = A + B + C_0$, the computation being performed is $S = A + 1111 + C_0$. In 2s complement, this computation is $S = A - 1 + C_0$. If $C_0 = 0$, this implements the decrement operation $S = A - 1$, using considerably less logic than a 4-bit addition or subtraction.