2. Practice Problem Solutions
The following are solutions to the set of practice problems presented separately. Use these only after attempting the problems on your own.
HW2 - Number Systems and Conversions

1. a. 1110110.010111
   = 1*2^6 + 1*2^5 +1*2^4 + 1*2^2 + 1*2^1 + 1*2^-2 + 1*2^-5 + 1*2^-6
   = 118.359375\text{ }_{10}
   = 001 | 110 | 110.010 | 111_2 = 166.278
   = 0111 | 0110.0101 | 1100_2 = 76.5C_{16}

   b. 15B.35_{16}
   = 1*16^2 + 5*16^1 +11*16^0 + 3*16^{-1} + 5*16^{-2} = 347.20703125_{10}
   = 0001 | 0101 | 1011.0011 | 0101_2
   = 101 | 011 | 011.001 | 101 | 010_2 = 533.1528

2. Use the "Making Change" method
   a. Start by listing the powers of 2 and then find the coefficients of the number by starting with the largest powers and working toward lower powers determining which ones sum to the desired value.

   | 1024 | 512 | 256 | 128 | 64  | 32  | 16  | 8   | 4   | 2   | 1 . 1 0 . 1 |
   | 1  0  1  1  1  0  0  1  1  0  1  1 . 1  0  1  . 2  5 . 1  2  5  |

   (923.625)_{10} = (1110011011.101)_{2}

3. a. 4530.1528
   = 100 101 011 000.001 101 01_2
   = 1001 | 0101 | 1000.0011 | 0101_2 = 958.35_{16}

4. a. DABB.AD00_{16}
   = 1101 1010 1011 1011.1010 1101_2
   = 001 | 101 | 101 | 101 | 011 | 111.101 | 011 | 010 | 010_2 = 155273.5328

   b. BAD.A_{16} = 11*16^2 + 10*16^1 +13*16^0 + 10*16^{-1} = 2989.625_{10}

5. Use the "Making Change" method: Start by listing the powers of 5 and then find the coefficients of the number by starting with the largest powers and working toward lower powers determining which ones sum to the desired value

   a. | 3125 | 625 | 125 | 25  | 5   | 1   | . 3  1  |
   | 3125 | 625 | 125 | 25  | 5   | 1   | . 3  . 2  . 0 4  |

   (2143.64)_{10} = (32033.31)_5
HW3 - Boolean Algebra, Logic Functions, and Canonical Representation, 2-Level Implementations

1. Probably the easiest method is perfect induction (i.e. a truth table)
   \[ F = X + X' = 1 \]
   
<table>
<thead>
<tr>
<th>X</th>
<th>X'</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

2. 
   a. \[ F = WXYZ \cdot (WXYZ' + WX'YZ + WXY'Z + WXYZ) \]
      = \[ WWXXYYZZ' \cdot WWX'YZZ + W'XXXXYZZ + 
      WWXXYY'ZZ \]
      = \[ WXYZZ' + WXX'YZ + WW'XYZ + WXYZ'Z \]
      = 0 + 0 + 0 + 0 + 0
      = 0
      A4’

   b. \[ F = AB + ABC'D + ABDE' + ABC'E + C'D \]
      = \[ AB \cdot (1 + C'D + DE' + C'E) + C'D \]
      = \[ AB \cdot (1 + DE' + C'E) + C'D \]
      = \[ AB + C'D \]
      T2
      T1’

3. 
   a. \[ F = X'Y + X'Y'Z' \]

<table>
<thead>
<tr>
<th>XYZ</th>
<th>X'</th>
<th>X' * Y</th>
<th>Y'</th>
<th>Z'</th>
<th>X' * Y' * Z'</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>010</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>011</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
b. \( F = W' + X' \cdot (Y' + Z) \)

<table>
<thead>
<tr>
<th>WXYZ</th>
<th>W'</th>
<th>Y'</th>
<th>Y'+Z</th>
<th>X'</th>
<th>X' • (Y'+Z)</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0011</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0101</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0110</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0111</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1001</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1100</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1101</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1111</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\( A' + B' + CD \) • \( B + C' + D' \)

<table>
<thead>
<tr>
<th>ABCD</th>
<th>A'</th>
<th>B'</th>
<th>CD</th>
<th>A' + B' + CD</th>
<th>C'</th>
<th>D'</th>
<th>B + C' + D'</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0011</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0100</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0101</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0110</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0111</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1001</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1010</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1011</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1111</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
4.

a.

\[ F = \Sigma_{XYZ} (0,2,3) \]
\[ = m0 + m2 + m3 \]
\[ = X'Y'Z' + X'YZ' + X'YZ \]

\[ = \Pi_{XYZ} (1,4,5,6,7) \]
\[ = M1 \cdot M4 \cdot M5 \cdot M6 \cdot M7 \]
\[ = (X+Y+Z')(X'Y'+Z') \cdot (X+Y+Z) \cdot (X'+Y'+Z) \cdot (X+Y+Z') \]

b.

\[ F = \Pi_{ABC} (1,2,4,6) \]
\[ = M1 \cdot M2 \cdot M4 \cdot M6 \]
\[ = (A + B + C') \cdot (A + B' + C) \cdot (A' + B' + C) \cdot (A' + B + C) \]

\[ = \Sigma_{A,B,C} (0,3,5,7) \]
\[ = m0 + m3 + m5 + m7 \]
\[ = A'B'C' + A'BC + AB'C + ABC \]
c.
\[
F = \Sigma_{ABCD}(1,2,5,7) \\
= m_1 + m_2 + m_5 + m_7 \\
= A'B'C'D + A'B'CD' + A'BC'D + A'BCD \\
= \prod_{ABCD} (0,3,4,6,8,9,10,11,12,13,14,15) \\
= M_0 \cdot M_3 \cdot M_4 \cdot M_6 \cdot M_8 \cdot M_9 \cdot M_{10} \cdot M_{11} \cdot M_{12} \cdot M_{13} \cdot M_{14} \cdot M_{15} \\
= (A + B + C + D) \cdot (A + B + C' + D') \cdot (A + B + C' + D) \\
\cdot (A' + B + C' + D') \cdot (A' + B' + C + D) \\
\cdot (A' + B' + C + D') \cdot (A' + B' + C' + D) \\
\cdot (A' + B' + C' + D') \cdot (A' + B' + C' + D')
\]

d.
\[
F = X' + YZ' + YZ' = X' + YZ' \\
= X'(Y + Y')(Z + Z') + (X+X')(YZ') \\
= X'Y'Z' + X'Y'Z + X'YZ' + X'YZ + XYZ' + X'YZ' \\
= m_0 + m_1 + m_2 + m_3 + m_6 \\
= \Sigma_{XYZ}(0,1,2,3,6) \\
= \prod_{X,Y,Z}(4,5,7) \\
= M_4 \cdot M_5 \cdot M_7 \\
= (X' + Y + Z) \cdot (X' + Y + Z') \cdot (X' + Y' + Z')
\]

e.
\[
F = A'B + B'C + AC' \\
= (A'BC + A'BC') + (AB'C + A'B'C) + (ABC' + AB'C') \\
= m_3 + m_2 + m_5 + m_1 + m_6 + m_4 \\
= \Sigma_{ABC}(1,2,3,4,5,6) \\
= \prod_{ABC}(0,7) \\
= M_0 \cdot M_7 \\
= (A + B + C) \cdot (A' + B' + C')
\]

5.

\[
X + Y \\
\begin{array}{c}
\begin{array}{c}
\text{x} \\
\text{y}
\end{array}
\end{array} \\
\cdot \\
\begin{array}{c}
\begin{array}{c}
\text{x} \\
\text{y}
\end{array}
\end{array} \\
= \\
\begin{array}{c}
\begin{array}{c}
\text{x} \\
\text{y}
\end{array}
\end{array}
\]

6. **One possible solution:**
   \[ XY + X'Z + YZ \]
   \[ XY + X'Z + YZ \cdot 1 \quad T1' \]
   \[ XY + X'Z + YZ \cdot (X + X') \quad T5 \]
   \[ XY + X'Z + XYZ + XYZ \quad T8 \]
   \[ (XY + XYZ) + (X'Z + X'ZY) \quad T6 \]
   \[ XY + X'Z \quad T9 \]

7. a. 
   \[ F = X'Z' + (Y(X' + Z))' \]
   \[ = X'Z' + Y' + (X' + Z)' \quad DEMORGANS \]
   \[ = X'Z' + Y' + XZ' \quad DEMORGANS \]
   \[ = (X' + X) \cdot Z' + Y' \quad T8 \]
   \[ = 1 \cdot Z' + Y' \quad T5 \]
   \[ POS = Z' + Y' \quad T1' \]

   b. 
   \[ G = XY + Y'Z' \]
   \[ = (XY + Y')(XY + Z') \quad T8' \]
   \[ = (X + Y')(Y + Y')(X + Z')(Y + Z') \quad T8' \]
   \[ = (X + Y')(X + Z')(Y + Z') \quad T3' \]
   \[ = (X + Y')(Y + Z') \quad T11' \]
   \[ POS = (X + Y')(Y + Z') \]

   c. 
   \[ H = AB \cdot (CD)' + A + D \]
   \[ = (AB \cdot (C' + D')) + A + D \quad DEMORGANS \]
   \[ = (AB + A + D) \cdot (C' + D' + A + D) \quad T8' \]
   \[ = (A + 1 + D) \cdot (1) \]
   \[ = (A + D) \cdot 1 \quad T1' \]
   \[ POS = A + D \]

8. 
   \[ Z = AB + (C' + A'B')' + A'(AB + AC'D') \]
   \[ = AB + (C' + A'B')' + A'AB + A'AC'D' \quad T8 \]
   \[ = AB + C \cdot (A + B) + 0 + 0 \quad DEMORGANS, T5' \]
   \[ SOP = AB + AC + BC \quad T8 \]

9. a. 
   \[ F = x'y' + xy'z + z' \]
   Let us simplify first
   \[ = y'(x' + xz) + z' \]
   \[ = y'(z + z') + xz + z' \quad T1', T5 \]
   \[ = y'(x'z + x'z' + xz) + z' \quad T8 \]
= y'(x'z + x'z + x'z' + xz) + z'  
= y'(z + xz) + z'  
= y'(x' + z) + z'  
T3 (replicate terms)  
T8  
T5,T1’

Now let us convert to POS using T8’
= (z'+y')(z' + x' + z)  
= (z'+y')  
T8’  
T5,T1’

b.  
G = (x'+y')(y)(w'+y+z)
Convert to SOP using T8 (simplifying as we go)
= (x'y + y'y)(w' + y + z)  
= x'y(w' + y + z)  
= w'x'y + x'y + x'yz  
= x'y(w' + 1 + z)  
= x'y  
T8  
T5'/T1  
T8  
T8  
T2 / T1’
HW4 - Circuit Design w/ Karnaugh Maps

1. 
   a. $F = \Sigma_{ABCD}(1,2,4,5,9,10,12,13)$
      
      ![Karnaugh Map for a](image)
      
      $$F_{SOP} = \overline{CD} + \overline{B}C + \overline{BC}\overline{D}$$

   b. $F = \Pi_{MNOP}(0,1,2,8,9)$
      
      ![Karnaugh Map for b](image)
      
      $$F_{SOP} = N + OP + MO$$

   c. $F = \Sigma_{ABCD}(0,1,2,5,8,10,11,15)$
      
      ![Karnaugh Map for c](image)
F_{SOP} = \overline{ACD} + ACD + BD

d. \quad F = \Pi_{wxyz} (3, 6, 7, 12, 14)

\[ F_{SOP} = \overline{WY} + WZ + \overline{XZ} \]

2.

a. \quad F = X'Y' + X'Z' + W'X + XYZ

\[ F_{POS} = (W' + X' + Y)(X + Y' + Z')(W' + X' + Z) \]

b. \quad The 4-bit prime numbers are: 2, 3, 5, 7, 11, 13

\[ F_{POS} = (C+D)(B'+D)(A'+D)(A'+B+C) \]
c. The 4-bit numbers that are not perfect squares or cubes are: 2, 3, 5, 6, 7, 10, 11, 12, 13, 14, 15.

\[ F_{\text{POS}} = (B + C)(A + C + D) \]

d. The 4-bit numbers divisible by 3 or 5 are: 0, 3, 5, 6, 9, 10, 12, 15

\[ F_{\text{POS}} = (A+B+C+D')(A+B'+C+D)(A+B'+C'+D)(A'+B+C+D)(A'+B'+C+D')(A'+B+C+D) \]

3. The K-Maps are below:

\[ F_{\text{SOP}} = \overline{CD} + \overline{BC} + \overline{BCD} \]
\[ F_{POS} = (\overline{C} + \overline{D})(\overline{B} + \overline{C})(B + C + D) \]

a. AND – OR Implementation

![AND-OR Implementation Diagram]

b. OR - AND Implementation

![OR-AND Implementation Diagram]
c. NAND - NAND Implementation

\[ B \]
\[ C' \]
\[ D \]
\[ B' \]
\[ C \]
\[ D' \]


d. NOR - NOR Implementation

\[ D' \]
\[ C' \]
\[ B' \]
\[ B \]
\[ C \]
\[ D \]
4.

a. The values with an odd number of 1s are: 1, 2, 4, 7, 8, 11, 13, 14. The Don’t Care numbers are: 0, 4, 5, 8, 10, 12, and 15.

\[
\text{SOP} = A'C' + BD + AC + B'D'
\]

b. \( F = B'C'D' + BCD' + ABC'D, \ d = A'BC'D + A'B'CD' \)

\[
F = (A+A')B'C'D' + (A+A')BCD' + ABC'D
\]  
\[= AB'C'D' + A'B'C'D' + ABCD' + A'BCD' + ABC'D
\]

\[d = A'BC'D + A'B'CD'
\]

\[
F_{\text{SOP}} = B'C'D' + BC'D + BCD'
\]
5.

a. \( F = A'C + ACE + CAB' + B'A'DE + A'D'E' \)

b. \( F = (AB + A'B') (C'D' + CD) \)
   \( = ABC'D' + A'B'C'D' + ABCD + A'B'CD \)

- **K-Map:**

\[ F_{SOP} = ABC'D' + A'B'C'D' + ABCD + A'B'CD \]
• NAND – NAND Implementation

\[ F = A'B'C'D' + A'B'CD' + A'BC'D' + A'BC'D + AB'CD' \]

• K-Map:

\[ F_{SOP} = A'C'D' + A'BC' + B'CD' \]

• NAND – NAND Implementation
d. The numbers are: 2, 3, 4, 5, 10, 12, 13, 14, 15.

- **K-Map:**

```
  00 01 11 10
AB 0  1  2  3
  1  5  7  15
CD 0  1  1  1
   1  3  1  9
```

\[
\text{SOP} = AB + BC' + A'B'C + ACD'
\]

- **NAND – NAND Implementation**
6.

a. \( F = A'C + ACE + CAB' + B'ADE + A'D'E' \)

- **K-Map:**

```
<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>DE</td>
<td>0</td>
<td>1</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>BC</td>
<td>0</td>
<td>1</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ F_{\text{POS}} = (A' + C')(A' + B' + E)(B' + C + E')(C + D + E')(C + D' + E) \]

b. \( F = (AB + A'B')(C'D' + CD) \)

- **K-Map:**

```
<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>CD</td>
<td>0</td>
<td>9</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>3</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ F_{\text{POS}} = (A + B')(A' + B)(C + D')(C' + D) \]
NOR – NOR Implementation

\[ F = A'B'C'D' + A'B'CD' + A'BC'D' + A'BC'D + AB'CD' \]

- K-Map:

\[ F_{\text{POS}} = (B + D')(B' + C')(A' + C) \]

- NOR – NOR Implementation:
d. The numbers are: 2, 3, 4, 5, 10, 12, 13, 14, 15.

- K-Map:

<table>
<thead>
<tr>
<th>CD</th>
<th>AB</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>10</td>
</tr>
</tbody>
</table>

$$\text{POS} = (B + C)(A + B' + C')(A' + B + D')$$

- NOR – NOR Implementation:

7. A and B are both 2-bit signed numbers. They range from -2 to +1. Thus $F = A + B$ is ranging from -4 to +2. We need 3 bits to represent the signed number F.

a. The block diagram is below:
b. The truth table is below:

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>B1</th>
<th>B0</th>
<th>F2</th>
<th>F1</th>
<th>F0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

K-Map of output “F_2”

K-Map of output “F_1”

F_{2\text{ SOP}} = A_1A_0' + B_1B_0' + A_1B_1 + A_1B_0' + A_0'B_1

F_{1\text{ SOP}} = A_1B_1'B_0' + A_1A_0'B_1' + A_1'A_0'B_1 + A_1'B_1B_0' + A_1'A_0B_1'B_0 + A_1A_0B_1B_0
K-Map of output “F₀”

\[
F₀_{SOP} = A₀B₀' + A₀'B₀
\]

d. AND-OR implementation

Implementation of F₂

Implementation of F₁

The implementation consists of four 3-input AND gates, two 4-input AND gates and one 6-input OR gate.

Implementation of F₀
NAND – NAND Implementation

(1) Implementation of $F_2$

(2) Implementation of $F_1$

The implementation consists of four 3-input NAND gates, two 4-input NAND gates and one 6-input NAND gate.

(3) Implementation of $F_0$
8. X is a 3-bit unsigned number. It ranges from 0 to 7. Thus Y = 3X + 1 is ranging from 1 to 22. We need 5 bits to represent the unsigned number Y.

a. The block diagram is below:

```
X2 -> Y4
X1 -> Y3
X0 -> Y2
```

b. The truth table is below:

<table>
<thead>
<tr>
<th>X</th>
<th>X2</th>
<th>X1</th>
<th>X0</th>
<th>Y4</th>
<th>Y3</th>
<th>Y2</th>
<th>Y1</th>
<th>Y0</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>22</td>
</tr>
</tbody>
</table>

c. K-Map of output “Y4”

```
Y4_{POS} = X_2(X_1 + X_0)
```

K-Map of output “Y3”

```
Y3_{POS} = (X_2 + X_1)(X_1' + X_0)(X_2' + X_0')
```

K-Map of output “Y2”

```
Y2_{POS} = (X_2 + X_1 + X_0)(X_2 + X_1' + X_0')(X_2' + X_1' + X_0)(X_2' + X_1 + X_0')
```
K-Map of output “Y₁”

Y₁POS = X₁

K-Map of output “Y₀”

Y₀POS = X₀'

d.  
  - OR – AND Implementation

Implementation of Y₄

X₁
X₀
X₂

Implementation of Y₃

X₂
X₁
X₁'
X₀
X₂'
X₀'
Conversion of Y1 and Y0 to NOR Gates

Convert all gates in the above implementations to NOR gates. For Y1 and Y0 just use wires.
e. This will not be covered at this point of the class since we have not yet discussed adders. Ignore this question.

9. X is a 3-bit signed number. It ranges from -4 to +3. Thus, \( Z = X^2 + 2X + 1 = (X + 1)^2 \) is ranging from 0 to +16. We need 6 bits to represent signed number Z.

a. The block diagram is below:

![Block Diagram](image)

b. The truth table is below:

<table>
<thead>
<tr>
<th></th>
<th>X2</th>
<th>X1</th>
<th>X0</th>
<th>Z5</th>
<th>Z4</th>
<th>Z3</th>
<th>Z2</th>
<th>Z1</th>
<th>Z0</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>+4</td>
</tr>
<tr>
<td>+2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>+9</td>
</tr>
<tr>
<td>+3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+16</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+0</td>
</tr>
<tr>
<td>-2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+1</td>
</tr>
<tr>
<td>-3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+4</td>
</tr>
<tr>
<td>-4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>+9</td>
</tr>
</tbody>
</table>

c. No K-Map needed for Z5. “Z5” = 0

K-Map of output “Z4”

\[
Z_{4\text{SOP}} = X_2'X_1X_0
\]

K-Map of output “Z3”

\[
Z_{3\text{SOP}} = X_2'X_1X_0' + X_2X_1X_0'
\]
K-Map of output “Z₂”

\[
\begin{array}{c|cccc}
X_0 & 00 & 01 & 11 & 10 \\
\hline
0 & 0 & 2 & 6 & 4 \\
1 & 3 & 7 & 5 &
\end{array}
\]

\[Z_{2\text{SOP}} = X_2'X_1'X_0\]

Note: \(Z_1 = 0\) (constant)

K-Map of output “Z₀”

\[
\begin{array}{c|cccc}
X_0 & 00 & 01 & 11 & 10 \\
\hline
0 & \text{C} & 1 & 2 & 6 \\
1 & 3 & 7 & 5 &
\end{array}
\]

\[Z_{0\text{SOP}} = X_2'X_0' + X_1'X_0'\]
10. Design a 1-bit comparator that takes in a bit X and a bit Y and outputs X<Y, X>Y, X=Y. Use a single 2-to-4 decoder and 1 single OR gate.

Consider the truth table of each of these simple functions. We can implement each as a sum of minterms. And since a decoder implements the minterms of the input variables, we can arrive at the circuit above.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X&lt;Y</th>
<th>X&gt;Y</th>
<th>X==Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
HW5 - Signed Representations and Arithmetic

1. 
   a. 
   \[
   \begin{array}{c}
   1111 \\
   10100 = 20 \\
   + 11101 = 29 \\
   \text{Carry}=1 \\
   110001 = 49 \\
   \end{array}
   \]
   
   b. 
   \[
   \begin{array}{c}
   11100 \quad 28 \\
   - 10100 \quad -20 \\
   01000 \quad 8 \\
   \text{2’s comp} \quad 01011 \\
   + 1 \\
   01000 \\
   \end{array}
   \]

2. 
   b. 
   \[
   \begin{array}{c}
   11111_\text{2’s comp} \quad 11100 \\
   - 101.011 \\
   + 011.010 \\
   1000.101 \\
   \end{array}
   \]

3. For unsigned subtraction we still use take the 2’s complement of the bottom number (but the check for overflow differs for unsigned vs. 2’s complement)
   a. \(0110110_2 - 0100100_2\) cannot be performed as is because we cannot subtract a larger number from a smaller number in unsigned representation.
   \[
   \begin{array}{c}
   0110110 \quad +54 \\
   - 0100100 \quad -60 \\
   \text{2’s comp} \quad 1011011 \\
   + 1 \\
   0010010 \\
   \end{array}
   \]

4. 
   a. Unsigned binary: \(2^8 < 283_{10} < 2^9\). Therefore, 9 bits. \(283_{10} = 100011011_2\)
   b. 2’s complement: with \(n\)-bits we can make \(-(2^{n-1})\) to \(+(2^{n-1})\). With \(n = 9\), the range is -256 to +256. With \(n = 10\), the range is -512 to +511. Thus 10 bits. \(283_{10} = 1000110111_2\)

5. Consider the following decimal numbers +21, +55, +121, -32, -99, -128
   a. What are the corresponding 8-bit signed-magnitude representation?.
a. What are the corresponding 8-bit 2's complement representation?.
b. Which of the above numbers can be represented in 6-bit signed-magnitude, 6-bit 1's complement, 6-bit 2's complement representations, explain?

<table>
<thead>
<tr>
<th></th>
<th>21</th>
<th>55</th>
<th>121</th>
<th>-32</th>
<th>-99</th>
<th>-128</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>00110111</td>
<td>01110001</td>
<td>10100000</td>
<td>11100011</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>00010101</td>
<td>00110111</td>
<td>01111001</td>
<td>11100000</td>
<td>10011101</td>
<td>10000000</td>
</tr>
<tr>
<td>C</td>
<td>Y,Y,Y</td>
<td>N,N,N</td>
<td>N,N,N</td>
<td>N,N,Y</td>
<td>N,N,N</td>
<td>N,N,N</td>
</tr>
</tbody>
</table>

010101 in all cases

6. What are the corresponding decimal representations for the following binary numbers: 01011011, 11010010, if
   a. The binary numbers are in 8-bit signed-magnitude format?
   b. The binary numbers are in 8-bit 2's complement format?

   a) 64+16+8+2+1=91
   b) 64+16+8+2=82

7. Perform the following addition problems for the following 2’s complement numbers. State whether overflow does or does not occur for each problem. Justify your answer for why overflow does or does not occur. Check your work by converting each number to decimal.

<table>
<thead>
<tr>
<th>a</th>
<th>1010 0111</th>
<th>b</th>
<th>1001 0110</th>
<th>c</th>
<th>0101 1100</th>
<th>d</th>
<th>0101 1101</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1110 0100</td>
<td>+1011 0011</td>
<td>+1011 0101</td>
<td>+0110 1001</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>÷1100 1011</td>
<td>÷0100 1001</td>
<td>÷0001 0001</td>
<td>1100 0110</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

n+n=n
n+n=p
p+n=p
p+p=n

<table>
<thead>
<tr>
<th>cin=cout=1</th>
<th>cin=0,cout=1</th>
<th>cin=cout=1</th>
<th>cin=1,cout=0</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Overflow</td>
<td>Overflow</td>
<td>NoOverflow</td>
<td>Overflow</td>
</tr>
</tbody>
</table>

8. This is an exercise to help you remember what gates are used to create a full adder. Using a full adder as shown below and no other gates, can you produce the function Z = A•B (Hint: Write out the logical equations for S and Cout and see if you can hook up the inputs to produce Z).

   ANSWER:

<table>
<thead>
<tr>
<th>Cin</th>
<th>X</th>
<th>Y</th>
<th>S</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
We want $Z=XY$, and if we set $Cin=0$ then the formulas of $S$ and $Cout$ becomes as follows:

$S = X \oplus Y \oplus Cin$

$Cout = XY + XCin + YCin$

Thus, $Cout = Z = XY$
9. Using 1 half-adder and a minimal number of 4-bit binary adders, design a circuit to calculate \( Y = 25\times X \), where \( X \) is a 4-bit inputs.

**ANSWER:**

If a number is multiplied by a number that is a power of 2, then the number rewritten using the above formula:

\[
2^n X = X \underbrace{0...0}_{n}
\]

\[
Y = 25\times X = 16\times X + 8\times X + X = X0000+X000+X
\]

Also adding the information that \( X \) is a 4-bit number above formula can be re-written

\[
X = X_3X_2X_1X_0
\]

\[
Y = X_3X_2X_1X_0000 + X_3X_2X_1X_000 + X_3X_2X_1X_0
\]

This can be written as addition by 4 bit adders:

\[
\begin{array}{cccc}
X_3 & X_2 & X_1 & X_0 \\
+ & X_3 & X_2 & X_1 & X_0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Since one row in each addition is all zeros, they can be eliminated and the other two numbers can be added using a 4-bit adder. However, the lower bits require only a single bit adder. The carry from the first addition is transferred to the next one.
10. Using a minimal number of 4-bit adders, design a circuit that implements $Y=20*X+107$, where $X$ is a 3-bit unsigned number.

**ANSWER:**

$X = X_2X_1X_0$, and thus $20X = 16X + 4X$. These numbers are achieved by inserting zeros at the end. Once we add these there numbers, we see that not all bits require addition. We actually need only a single 4-bit adder and nothing more.

\[
\begin{array}{ccccccc}
1 & 1 & 0 & 1 & 1 & 1 & \\
& & X_2 & X_1 & X_0 & 0 & 0 \\
+ & & X_2 & X_1 & X_0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Which can be collapsed to the following addition:

\[
\begin{array}{ccccccc}
1 & 1 & X_2 & X_1 & X_0 & 0 & 0 & \\
& & X_2 & X_1 & X_0 & 1 & 0 & 1 & 1 \\
\end{array}
\]
11. Design a minimal circuit using AND/OR/NOT gates to implement the comparison $A > 10$ where $A$ is a 4-bit number.
   a. Start by writing out the logical algorithm for when $A > 10$ then implement it using gates
   b. Check that your work is minimal by using a K-Map.

**ANSWER:**
Let’s assume that the number $A = A_3A_2A_1A_0$.

Logical Algorithm: For any number $A$ to be greater than a number $B$ (in this case $1010_2$), a more significant bit of $A$ must be greater than $B$. However, in the case of $A$ compared with $1010$, we realize $A_3$ and $A_1$ can never be greater than the 1’s in those places for 10. Similarly, $A_2$ and $A_0$ can never be less than the 0’s in those places. Thus for $A > 10$, there are only 2 cases that need to be checked:

- $A_3 = 1$ and $A_2 > 0$ (i.e. $A_2 = 1$)
- $A_3=1$ and $A_1 = 1$ and $A_0 > 0$ (i.e. $A_0 = 1$).

Thus $A > 10 = A_3A_2 + A_3A_1A_0$.

Note that we don’t have to check $A_2$ in the second case because it is DEFINITELY greater than or equal to the 0 in that place of $1010_2$.

A K-Map will also show the same equation
$\text{SUM}(11,12,13,14,15)$.
HW 6 – Latches, and Flip-Flops

1. Complete the following waveforms for negative edge-triggered D Flip-Flops.

   ![Waveform Diagram]

   

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>XY = 00</td>
<td>XY = 01</td>
<td>XY = 10</td>
</tr>
<tr>
<td>Q1</td>
<td>Q0</td>
<td>Q1*</td>
</tr>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>0 0</td>
<td>0 0</td>
</tr>
</tbody>
</table>

2. We should find the state diagram following the steps in the class notes
   a. Convert the diagram to a table.
Convert Q* to D

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 Q0</td>
<td>XY = 00</td>
<td>D1 D0</td>
</tr>
<tr>
<td></td>
<td>XY = 01</td>
<td>D1 D0</td>
</tr>
<tr>
<td></td>
<td>XY = 10</td>
<td>D1 D0</td>
</tr>
<tr>
<td></td>
<td>XY = 11</td>
<td>D1 D0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q0</th>
<th>D1</th>
<th>D0</th>
<th>D1</th>
<th>D0</th>
<th>D1</th>
<th>D0</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

b. Perform K-maps to find equations for D1 and D0.

\[ D_0 = Q_1 \cdot Y + Q_0 \cdot X \]

\[ D_1 = Q_1' \cdot Q_0' \cdot X \]

c. Draw the circuit and implement the initial state (reset) condition.
3. Because we have 2 states in the state diagram, we use one D flip-flop to implement state memory. We design the combinational logic for the next-state logic and the output function logic by building up the state/transition table.

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State / Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol/Code</td>
<td>Symbol/Code</td>
</tr>
<tr>
<td>XY = 00</td>
<td>XY = 01</td>
</tr>
<tr>
<td>XY = 10</td>
<td>XY = 11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F / 0</th>
<th>F / 0</th>
<th>G / 1</th>
<th>G / 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q0(t)</td>
<td>Q0*</td>
<td>Q0*</td>
<td>Q0*</td>
</tr>
<tr>
<td>D0</td>
<td>D0</td>
<td>D0</td>
<td>D0</td>
</tr>
</tbody>
</table>

After we get the state/transition table, we build the K-Maps to simplify expressions for D0, and H.

- **D0 K-Map:**

\[ D_0 = YQ_0' + XQ_0' + XY \]

\[ H = Q_0 \]
4.

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State &amp; Flop-Flop Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Next State</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X = 0</td>
<td>X = 1</td>
</tr>
<tr>
<td>A/00</td>
<td>B/01</td>
<td>D/11</td>
</tr>
<tr>
<td>B/01</td>
<td>C/10</td>
<td>B/01</td>
</tr>
<tr>
<td>C/10</td>
<td>B/01</td>
<td>A/00</td>
</tr>
<tr>
<td>D/11</td>
<td>B/01</td>
<td>C/10</td>
</tr>
</tbody>
</table>

After we get the state/transition table, we build the K-Maps to simplify expressions for D1, D0, and Z.

- **D1 K-Map:**

  \[ D_1 = X'Q_1'Q_0 + XQ_0' \]

  ![D1 K-Map Diagram]

- **D0 K-Map:**

  \[ D_0 = X'Q_1 + X'Q_0' + XQ_1' \]

  ![D0 K-Map Diagram]

- **Z K-Map:**

  \[ Z = Q_1Q_0' \]

  ![Z K-Map Diagram]
a. The waveform is below:
5.
   a. We need 4 states and each of them stores one 2-bit binary number ranging from 00 to 11.

<table>
<thead>
<tr>
<th>State</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

The state diagram is below:

b. Because we have 4 states in the state diagram, we use two D flip-flops to implement state memory. We design the combinational logic for the next-state logic and the output function logic by building up the state/transition table.

<table>
<thead>
<tr>
<th>Current State</th>
<th>U·D = 00</th>
<th>U·D = 01</th>
<th>U·D = 10</th>
<th>U·D = 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0/00</td>
<td>S0/00</td>
<td>S3/11</td>
<td>S1/01</td>
<td>S0/00</td>
</tr>
<tr>
<td>S1/01</td>
<td>S1/01</td>
<td>S0/00</td>
<td>S2/10</td>
<td>S1/01</td>
</tr>
<tr>
<td>S2/10</td>
<td>S2/10</td>
<td>S1/01</td>
<td>S3/11</td>
<td>S2/10</td>
</tr>
<tr>
<td>S3/11</td>
<td>S3/11</td>
<td>S2/10</td>
<td>S0/00</td>
<td>S3/11</td>
</tr>
<tr>
<td>Q1(t),Q0(t)</td>
<td>Q1(t+1),Q0(t+1)</td>
<td>Q1(t+1),Q0(t+1)</td>
<td>Q1(t+1),Q0(t+1)</td>
<td>Q1(t+1),Q0(t+1)</td>
</tr>
<tr>
<td></td>
<td>D1,D0</td>
<td>D1,D0</td>
<td>D1,D0</td>
<td>D1,D0</td>
</tr>
</tbody>
</table>

After we get the state/transition table, we build the K-Maps to simplify expressions for D1, and D0.
- **D₁ K-Map:**

\[
D₁ = U·D'Q₁ + D·Q₁Q₀ + U·Q₁Q₀' + U'·DQ₁'Q₀' + U·D'Q₁'Q₀
\]

- **D₀ K-Map:**

\[
D₀ = U'·D'Q₀ + U·DQ₀ + U'·DQ₀' + U·D'Q₀'
\]

c. The waveform is below:
6. Complete the waveform for the 4-bit counter presented in class.

Clock  
SR  
CEP  
CET  
PE  
P3-P0  
Q3-Q0

7. Complete the waveform for a 4-bit D-Register with Data (Load) Enable and asynchronous reset.

CLK  
AR  
EN  
D[3:0]  
Q[3:0]
8.1. Unit 7 – Datapath Design

1. Implement a circuit that takes in a 4-bit number X[3:0] and produces a 4-bit value Z[3:0] according to the following function:

   \[
   \begin{align*}
   &\text{if } X < 8 \text{ then} \\
   &\quad Z = X + 5; \\
   &\text{else} \\
   &\quad Z = X - 2; \\
   \end{align*}
   \]

   Using the building blocks below and at most 1 inverter (no other gates), implement this function.

Note: To check X<8, subtract X-8 by taking the 2’s complement of 8. The 2’s comp. of 8 is 0111+1. Recall, a carry of 0 when subtracting unsigned numbers (A-B) indicates overflow which can only occur in unsigned subtraction if the result needed to be negative (i.e. A < B). So C4 will only be 1 if X is not less-than 8. We can invert it to get a signal that is true when X < 8. Use those signals to either add 5 (0101) or subtract 2 (add 1110).
2. Design a circuit that takes in a 4-bit number \(X[3:0]\) and outputs a number \(Y\), and performs the operation below. Assume these numbers are unsigned. Use a 4-bit adder and 2-to-1 muxes. Hint: Multiplying by 4 (i.e. \(2^2\)) in binary can be done with no gates just as multiplying by 100 (i.e. \(10^2\)) in decimal can be done simply and easily.

\[
\text{if}(X < 0011_2) \\
\quad Y = 4X \\
\text{else} \\
\quad Y = X
\]

\(4X\) would require 6 outputs and be \(X3\ X2\ X1\ X0\ 0\ 0\), thus even if we pass \(X\) we need to produce 6 outputs. Recall subtracting 3=0011 bin. is the same as adding the 2's complement. Next, a carry of 0 when subtracting unsigned numbers \((A-B)\) indicates overflow which can only occur in unsigned subtraction if the result needed to be negative (i.e. \(A < B\))
3. Design a circuit that takes in two 4-bit numbers, X[3:0] and Y[3:0] along with two function select bits, FS1 and FS0, and produces an output, Z[3:0], according to the following table:

<table>
<thead>
<tr>
<th>FS1,FS0</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td>X+Y</td>
</tr>
<tr>
<td>0,1</td>
<td>X-Y</td>
</tr>
<tr>
<td>1,0</td>
<td>Y-X</td>
</tr>
<tr>
<td>1,1</td>
<td>don’t care</td>
</tr>
</tbody>
</table>

Use (1) 4-bit adder, 4-bit wide 2-to-1 muxes, and any basic gates you desire.

Approach: Take a 4-bit adder and for each operation listed in the table above identify what should be passed to the A and B inputs of the adder. Then use muxes to perform that function. Design logic for the select bits of the muxes and the carry-in of the adder that use FS1 and FS0 as input.

Note: In the solution below the A and B inputs of the muxes are like the 0th and 1st inputs of 4 2-to-1 muxes, respectively. Ignore the G input.
4. Design a circuit that takes in two 4-bit *signed magnitude* numbers, X[3:0] and Y[3:0] and produces two 4-bit outputs, A[3:0] and B[3:0] according to the following algorithm:

\[
\begin{align*}
\text{if } X & \geq Y \text{ then} \\
A &= X \text{ and } B = Y \\
\text{else} \\
A &= Y \text{ and } B = X
\end{align*}
\]

Use a 4-bit adder to subtract and perform the comparison, 4-bit wide 2-to-1 muxes, and basic logic gates. Assume the input 1000 = -0 will never occur on X or Y.

Approach: First think about how to compare signed magnitude numbers by using unsigned comparison techniques and how to precondition the inputs to allow the use of unsigned comparison. For example, 1001 = -1 in signed magnitude and 0100 = +4 in signed magnitude. So 1001 is really less than 0100. Use the comparison result to control the selects of the muxes.
5. Design a circuit that adds two 4-bit, unsigned numbers: \(X[3:0]\) and \(Y[3:0]\) and outputs either the sum if there is no overflow or 1111 (=15\(_{10}\), the maximum unsigned 4-bit number) if there is overflow.

Unsigned overflow is found by looking at Cout. You could use a mux to select the sum or 1111 but realize you can just use OR gates for this function.
1. We would like to build an adder whose inputs are in units of pennies and produces sum in nickels and pennies. Essentially, this is a base-5 adder. The 2 inputs of this adder are $A = \{A_2A_1A_0\}$ and $B = \{B_2B_1B_0\}$. Assume that the maximum input number that $A$ and $B$ can be is 4 pennies. Design an adder to produce the correct nickels/pennies sum, $C = \{[C_N] ; [C_{P2}C_{P1}C_{P0}]\}$. To implement this design you may assume you can use as many 4-bit adders as you like.

\[
\begin{align*}
\text{if } X+Y > 4 \text{ dec, then} \\
&CN = 1, \ CP[2:0] = X+Y - 5 \\
\text{else} \\
&CN = 0, \ CP[2:0] = X+Y
\end{align*}
\]

To be designed by you

Block diagram for the nickel/penny adder you are going to design
"Need actual inverters here, just didn't draw them."

CN
(4 < X+Y)