1. Is the expression given below true or false? Give justification for your answer.

\[ A \cdot (B \oplus C) = (A \oplus B) \cdot (A \oplus C) \]

Make a truth table to determine if the given expression is true or false.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</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>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Left \neq Right \Rightarrow \text{False}

2. Give the 2 level NOR-NOR realization of \( F(a, b, c, d) = \Pi (0, 2, 3, 8, 10, 11, 15) \).
Assume the complement of each Boolean variable is available.

\[ F = \overline{b} \overline{d} + \overline{b} \overline{c} + \overline{a} \overline{c} \overline{d} \]

\[ F = (b+d)(b+c)(\overline{a} + \overline{c} + \overline{d}) \]

3. The following are PI's of a 4 variable Boolean function \( g(w, x, y, z) \). Identify those which are NOT essential PI's.

<table>
<thead>
<tr>
<th>Karnaugh Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

#3 and #2 contain a minimum \((m_5 + m_{15})\) not contained in any other PI, So they are essential.
4. Convert the decimal number 343.24 to base 5

\[
\begin{align*}
343 \div 5 &= 68 \quad \text{R} \\
68 \div 5 &= 13 \quad \text{R} \\
13 \div 5 &= 2 \quad \text{R} \\
2 \div 5 &= 0 \quad \text{R} \\
\text{Fraction} &= \quad 0.24 \div 5 = 0.4 \\
\text{R} &= \quad 0.2 \div 5 = 0.0
\end{align*}
\]

\[2333.11\]

5. Convert the binary number 11010.101 to decimal

\[1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4}\]

\[= 26.625\]

6. Add the 2 hexadecimal numbers. Leave the result in hexadecimal.

\[
\begin{align*}
\begin{array}{c c c c c c c c c c c c c c c c c c}
& & & 1 & 0 & 9 & 8 & 7 & 6 & h \\
\hline
+ & 5 & 8 & 3 & 2 & 1 & h \\
\hline
& & 6 & 1 & 8 & 7 & 7 & h
\end{array}
\end{align*}
\]

1 + 6 = 7
7 + 2 = 9
8 + 3 = 11
9 + 8 = 17 = 16 + 1 \rightarrow 11 \text{ in hex}
1 + 0 + 5 = 6
7. Perform the BCD addition given below. Be sure to show your work, noting any correction steps that are required.

\[
\begin{align*}
498 & \quad + \quad 583 \\
\text{corrections} & \quad \rightarrow \\
\text{if necessary} & \quad \rightarrow \\
\end{align*}
\]

If the sum is greater than 9, correct by adding 6.

\[
\begin{align*}
1000 & \quad + \quad 0011 \\
\rightarrow & \quad \text{cound} \\
\text{answer:} & \quad 1081
\end{align*}
\]

8. Mark each Boolean identity which is incorrect, and give one counter example to illustrate that it is wrong.

<table>
<thead>
<tr>
<th>Mark</th>
<th>Boolean Identity</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>(AB + A'B' = 1)</td>
</tr>
<tr>
<td></td>
<td>(ABC + ABC' + B' = A + B')</td>
</tr>
</tbody>
</table>

1. \(AB + \overline{A}\overline{B} = 1\) when \(B = 1\), the Identity

2. Simplify \(ABC + A\overline{B}C' + \overline{B}\)

\[
\begin{align*}
&= \overline{AB} \left( \overline{C} \oplus C' \right) + \overline{B} \\
&= \overline{B} + A
\end{align*}
\]

9. Express the following Boolean function in product of maxterms canonical form:

1. Write as SOP

\[
F(w, x, y, z) = y'(xz + z') + wx(y + y') + x'y'z = \overline{y}z + \overline{x}z + \overline{w}x + \overline{w}yz + \overline{x'y'}
\]

2. Draw Karnaugh Map

\[
\begin{array}{cccc}
& 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
\end{array}
\]

Max terms correspond to the zeros on the KMAP

\[
F = \bigoplus (M_1, M_2, M_6, M_7, M_9, M_{10})
\]
10. Express $F$ in minimal sum of products form and minimal product of sums form.

Label nets to make it easier.

\[ F = \overline{AB} + \overline{A} \overline{C} \]

\[ \begin{align*}
Y &= \overline{AB} \\
X &= A \overline{y} = A \overline{AB} \\
Z &= C \overline{y} = C \overline{AB}
\end{align*} \]

\[ F = XYZ = (A \overline{AB})(C \overline{AB}) \]

11. If $A$, $B$, and $C$ are Boolean variables, and $A + B = 1$ and $B + C = 0$, solve for $A$, $B$, $C$

\[ B + C = 0 \rightarrow \text{Since OR is only zero if both inputs are zero} \rightarrow \]

\[ B = 0 \quad C = 0 \]

\[ A + B = 1 \rightarrow A + 0 = 1 \quad A = 1 \]

12. Give the dual of the following Boolean expression in sum of minterm format:

\[ \text{Dual} \]

\[ \text{OR} \rightarrow \text{AND} \]

\[ \text{AND} \rightarrow \text{OR} \]

\[ 0 \rightarrow 1 \]

\[ 1 \rightarrow 0 \]

\[ \begin{align*}
&\text{Minterm: } x'y + x'z + yz \\
&\text{Dual: } \overline{xy} + \overline{xz} + \overline{yz} \\
&\text{Minterm: } (x + y)(x + z)(y + z) \\
&\text{Dual: } (xy + xz + yz)(y + z) \\
&\text{Minterm: } yx + xy + yz + yz + xz + yz \\
&\text{Dual: } \leq \{0, 3, 5, 7\} \\
\end{align*} \]
13. Compliment the following Boolean expression and represent the result in sum of product format using De Morgan's Law.

\[ F = (a' + (b + c'd')) + e \]

\[ F = \overline{a \cdot (b + c' \cdot d')} + e = \overline{a \cdot (b + c' \cdot d')} \cdot \overline{e} = \left[ a + \overline{b + c' \cdot d'} \right] \cdot \overline{e} = \left[ a + \left( \overline{b + c' \cdot d'} \right) \cdot \overline{e} \right] \cdot \overline{e} = a \cdot \overline{e} + b \cdot \overline{c} \cdot \overline{e} + b \cdot d \cdot e \]

14. Perform subtraction of the following two unsigned binary numbers by taking the 2's compliment of the subtrahend.

\[
\begin{array}{c}
\text{100} \\
\hline
\text{0100} \to \text{0110} \\
- \text{1010} \\
\hline
\text{-6}
\end{array}
\]

15. Implement the following Boolean function using AND to 1 INX with S being the selection control and I_1 and I_0 being the 2 inputs.

\[ F(a, b, c) = ab' + bc' + c'a' \]

\[ \text{The trick: want every term to contain } a \text{ or } \overline{a} \text{ so introduce } (a + \overline{a}) \]

16. If \((10n01)_2 = 33\), find both \(n\) and \(r\).

\[
\begin{align*}
2^5 \cdot n + 2^4 \cdot 2^4 &= 3r + 3 \\
16 + 4n + 1 &= 3r + 3 \\
\text{Now, } n \text{ can only be } 0 \text{ or } 1 \text{ since it's binary} \quad \text{Case 2: } \begin{cases} n = 1 \Rightarrow 21 = 3r + 3 \\ 3r = 18 \Rightarrow \boxed{r = 6} \end{cases}
\end{align*}
\]
17. $F(a, b, c)$'s prime implicants are listed below. Which of the four are essential?

- $a'b'$
- $a'c$
- $bc$
- $ab$

$a'b'$ is 4, only $P_4$ to contain $m_6$.
$a'c$ is 4, only $P_4$ that contains $m_6$.

18. Find the Boolean function $F(w, x, y)$ and express it in simplified SOP format.

$w(x'y) + \overline{w(x+y)} = F$

$F = wx'y + wx + wy$

$F = \overline{w}x'y + \overline{w}x + \overline{w}y$

**MUX work like this:** $S.I_1 + \overline{S.I_0} = F$

So when $S = 0$, $I_0$ is selected.
And when $S = 1$, $I_1$ is selected.

19. For the function $F(a, b, c) = b'c + ab$, find the static and functional hazards.

- Make KMAP of SOP - only static 1's!

- Marked w/ $\Diamond$ when input des from $abc = 101$ to $abc = 111$ and vice versa.

- Functional 1 - $F = bc + ab + ac$

- $ABC = 001 \leftrightarrow ABC = 111$

- $ABC = 101 \leftrightarrow ABC = 110$

Functional 0:

- $ABC = 000 \leftrightarrow ABC = 011$

$0 \rightarrow 1 \rightarrow 0$
4. (16 points) Specifications and decoder based realization

(a) (8 points) Specifications to Truth Table

A combinational circuit accepts a one digit BCD input \((w,x,y,z)\) and produces two outputs \(F\) and \(G\). The circuit outputs meet the following conditions:

- \(F = 1\) and \(G = 0\) if the BCD input is \(\leq 3\)
- \(F = 0\) and \(G = 1\) if the BCD input is \(\geq 4\) but is \(\leq 6\)
- \(F = 1\) and \(G = 1\) if the BCD input is \(\geq 7\)

Write the truth table of this function:

<table>
<thead>
<tr>
<th>BCD sum</th>
<th>(W)</th>
<th>(X)</th>
<th>(Y)</th>
<th>(Z)</th>
<th>(F)</th>
<th>(G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Note: \(FG = 10\) if \(BCD \leq 3\)
\(FG = 01\) if \(3 < BCD \leq 6\)
\(FG = 11\) if \(BCD \geq 7\)
(b) (8 points) Decoder bases realization
Realize the specifications given in (a) above using a 3-to-8 decoder, and OR gates only. For your convenience a decoder with appropriate connections to the inputs and OR gates are drawn below. You are required to make connections on the figure provided. More gates are provided than necessary.

\[ \text{For } F \text{ - Notice that when } W = 1, F \text{ is either 1 or don't care, so we can hook } W \text{ directly into the OR, and add the necessary minterms (6,1,3,7,11) } \]

\[ \text{For } G \text{ - Notice that when } W = 1, G \text{ is also either 1 or don't care, so we can tie in } W \text{ directly, then add the required decoder outputs where } G \text{ is 1 and } W = 0 \text{, (4,5,6,7)} \]

Want to include minterms in the OR gate where \( F \) or \( G \) are 1, and also factor in \( W \).
5. (8 points) **Mux based realization**

Use the 8-to-1 multiplexer below to implement an exclusive-or function for four bits. This is also known as an "odd" function. The function \( \text{exor}(w,x,y,z) \) is to be:
\[
\text{exor}(w,x,y,z) = w \oplus x \oplus y \oplus z
\]

Draw the circuit in the space below. You may use only constant values (0, 1) as needed and one NOT gate. Note that the variable "w" has been factored out. Also, pay special attention to the connections of \( x, y, z \) to \( S2, S1, S0 \).

\[\begin{array}{cccccccc}
\bar{w} & & & & D7 & & & \\
| & & | & & | & & | & \\
& w & & & D6 & & & \\
| & & | & & | & & | & \\
& w & & & D5 & & & \\
| & & | & & | & & | & \\
& w & & & D4 & & & \\
| & & | & & | & & | & \\
& w & & & D3 & & & \\
| & & | & & | & & | & \\
& w & & & D2 & & & \\
| & & | & & | & & | & \\
& w & & & D1 & & & \\
| & & | & & | & & | & \\
& w & & & D0 & & & \\
\end{array}\]

\[\begin{array}{cccc}
w & x & y & z \\
\end{array} \quad \text{8-to-1 MUX} \quad \text{Out} \quad \text{exor}(w,x,y,z)\]

Write \( \bar{w} \) as such:

\[\begin{array}{cccccccc}
\bar{w} & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\hline
\text{Sel}_0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\text{Sel}_1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
\text{Sel}_2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\text{Sel}_3 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
\text{Sel}_4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Sel}_5 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
\text{Sel}_6 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
\text{Sel}_7 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
\text{Sel}_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Sel}_9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Sel}_{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Sel}_{11} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Sel}_{12} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Sel}_{13} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Sel}_{14} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Sel}_{15} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}\]

\[\begin{array}{cccc}
xy2 & w & \text{Out} \\
\end{array} \]

4 inputs \( \Rightarrow 2^3 = 8 \rightarrow 1 \text{ Mux}

Exor \( \Rightarrow 1 \) when there are an odd \# of 1's.
Remember: 
\[ S_i = (X_i \oplus Y_i) \oplus C_{i-1} = P_i \oplus C_{i-1} \]

14) In a 4-bit carry look-ahead adder, the carry out bit \( C_4 \) can be expressed as:

\[ C_4 = G_{0-3} + P_0 \cdot C_0 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0 \]

where \( G_{0-3} = G_1 + P_0 \cdot G_0 \) is called a group generate function and \( P_0 \cdot P_1 \cdot P_2 \cdot P_3 \) is called a group propagate function. Suppose we use four of such carry look-ahead adders to perform addition of 16-bit binary numbers. We have

\[ C_4 = G_{0-3} + P_0 \cdot C_0, \quad C_8 = G_{4-7} + P_4 \cdot C_4, \quad \text{and} \quad C_{12} = G_{8-11} + P_8 \cdot C_8. \]

(a) (9 points) Derive a two-level SOP realization for \( C_4 \) and \( C_{12} \) in terms of \( C_0 \) and group generate and group propagate functions. We will ignore the carry out bit \( C_0 \).

\[ C_8 = G_{5-7} + P_5 \cdot C_5 + C_4 = G_{4-7} + P_4 \cdot C_{0-3} + P_4 \cdot P_3 \cdot C_0 \]

\[ C_{12} = G_{5-11} + P_5 \cdot C_5 + C_8 = G_{4-7} + P_4 \cdot C_{0-3} + P_4 \cdot P_3 \cdot P_2 \cdot P_1 \cdot C_0 \]

Assume that an XOR gate contributes 2 gate delays. It takes 2 gate delays to evaluate \( P_4 \) and \( G_4 \) in each 4-bit carry look-ahead adder. What is the maximum gate delay to compute the result using the 16-bit hierarchical carry look-ahead adder described above?

\[ 10 + 4 = 14! \]

- 2 delays to calculate all \( P_i \)s and \( G_i \)s in parallel.
  \[ P_0 = A_0 \oplus B_0, \quad G_0 = A_0 \cdot B_0 \]

- 2 delays to calculate all group \( P_i \)s and \( G_i \)s for all CLA (2 level logic like above)

- 2 delays to calculate carry into 4-bit CLA \( (C_4, C_8, C_{12}) \)

- 2 delays to calculate internal carrying \( \text{w/in CLA} \)
  \[ C_5 = G_3 + P_3 \cdot C_3 \]
  \[ C_6 = G_5 + P_5 \cdot G_4 + P_5 \cdot P_4 \cdot C_4 \]
  \[ C_7 = G_6 + P_6 \cdot G_5 + P_6 \cdot P_5 \cdot G_4 + P_6 \cdot P_5 \cdot P_4 \cdot C_4 \]

- 2 delays to calculate \( S_i = P_i \oplus C_{i-1} \)
2. (20 points) Synchronous sequential circuit synthesis, simulation

Given the state diagram of a sequential circuit as follows:

![State Diagram]

(a) (10 points) The table below is a hand simulation of a sequential circuit. Complete this table.

<table>
<thead>
<tr>
<th>Clock Cycle, t:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Present State:</td>
<td>11</td>
<td>10</td>
<td>10</td>
<td>00</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>Present Input x(t):</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Present Output y(t):</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(b) (10 points) Derive the corresponding two-dimensional state table of this sequential circuit.

<table>
<thead>
<tr>
<th>PS</th>
<th>x = 0</th>
<th>x = 1</th>
<th>x = 0</th>
<th>x = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>01</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>10</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>00</td>
</tr>
</tbody>
</table>
Tabular Method

(a) (12 points) Prime implicant generation

Let: \( F(v,w,x,y,z) = \sum m(9,11,12,13,14,15,19,20,28,30) \)

Execute the Quine-McCluskey algorithm to find all the Prime Implicants of the function.

The algorithm has been started for you below (minterms are listed in increasing 1's count order). Complete the algorithm and **circle** the PRIME IMPlicants. You will be penalized severely if any of the implicant or prime implicant generated by is not an implicant or prime implicant.

<table>
<thead>
<tr>
<th>( m )</th>
<th>Index Order</th>
<th>( \checkmark )</th>
<th>1-Cubes</th>
<th>( \checkmark )</th>
<th>2-Cubes</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>01001</td>
<td></td>
<td>010-1</td>
<td></td>
<td>(01-1)</td>
</tr>
<tr>
<td>12</td>
<td>01100</td>
<td></td>
<td>01-01</td>
<td></td>
<td>01-1-</td>
</tr>
<tr>
<td>20</td>
<td>10100</td>
<td></td>
<td>0110-</td>
<td></td>
<td>011-</td>
</tr>
<tr>
<td>11</td>
<td>01011</td>
<td></td>
<td>011-0</td>
<td></td>
<td>01-0</td>
</tr>
<tr>
<td>13</td>
<td>01101</td>
<td></td>
<td>01100</td>
<td></td>
<td>(1-0-0)</td>
</tr>
<tr>
<td>14</td>
<td>01110</td>
<td></td>
<td>(0-1-0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>10011</td>
<td></td>
<td>01-1-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>11100</td>
<td></td>
<td>011-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>01111</td>
<td></td>
<td>011-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>11110</td>
<td></td>
<td>01100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Look for terms that differ in one place. Mark the place with a `=` to check off the term.

---

**PI's** are the unchecked terms: \( PI's = 10011, 1-100, 01--1, 01-1, -11-0 \)
A set of seven Prime Implicants were generated for a Boolean function. The cover table below is then derived for this function. Use this cover table to categorize the PIs into (i) Essential PI(s) (EPI), (ii) Less-than PI(s) (LTPi), (iii) Secondary Essential PI(s) (SEPI), or (iv) Redundant PI(s) (RPI).

<table>
<thead>
<tr>
<th>Prime Imp</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
<th>A7</th>
<th>PI category</th>
</tr>
</thead>
<tbody>
<tr>
<td>P11</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>EPI</td>
</tr>
<tr>
<td>P12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>EPI</td>
</tr>
<tr>
<td>P13</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LTPi</td>
</tr>
<tr>
<td>P14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RPI</td>
</tr>
<tr>
<td>P15</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LTPi</td>
</tr>
<tr>
<td>P16</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>SEPI</td>
</tr>
<tr>
<td>P17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SEPI</td>
</tr>
</tbody>
</table>

(c) (1 Point) Did the function above have any don't cares? Yes or No
Yes, because P17 contains three minterms, so must have don't care

(d) (2 points) Give reason for your answer (how did you draw the conclusion you have marked) in (c) above.

1. Look for EPIs by scanning down Y4 columns, looking for minterms covered only by 1 PI. Mark PI essential, and cross out all minterms covered by it.
2. Look down columns for remaining unchecked minterms. Do a comparison between PI's containing the specific minterm. A PI is less than another if it covers fewer unchecked minterms.
3. Cross out LTPi and find new secondary essential.
4. RPI - Leftover PI's once all Y4 minterms are checked.
2) Below is a logic diagram of three different types of clocked latches and flip-flops. Complete the timing diagram. Neglect propagation delay between input and output of latches or flip-flops.

When $S=R=1$, the circuit doesn't become unstable until key drop.
(b) (8 points) Synchronous sequential circuit

Design a serial parity-bit generator. Assume the input \( x \) is received sequentially. The parity bit generator will convert every third bit of the input sequence to the even parity bit of the first two bits. For example, if the inputs are 11001010000... where \( b \) denotes don't cares, then the corresponding outputs are 11001100000... where the parity bits are underscored. The state diagram has 5 states, A, B, C, D, and E, as shown below. Label on each arc the corresponding input and output. You may use \( b \) as one of the input symbol to represent don't cares.