Only Problem Set Part B will be graded. Turn in only Problem Set Part B which will be due on November 23, 2009 (Monday) at 4:00pm.

1 Problem Set Part A

- textbook 4.7(b)
- textbook 4.9(b)
- textbook 4.10(b)
- textbook 4.17(a)
- textbook 4.18
- textbook 5.3
- textbook 5.9
- textbook 5.10
2 Problem Set Part B

1 (Tabulation Method)

(Part A) The tabulation method for the identification of the prime implicants is being applied on a Boolean function of \( n \) variables. To assess your understanding of the process, we are providing below a number of statements. For each of them, please state whether the statement is True (T) or False (F) by circling either T or F. To jolt your memory (unnecessary though it probably is), we remind you that an ‘\( i \)-subcube’ denotes a subcube with \( 2^i \) minterms, while each \( G_k \) group contains subcubes which are denoted in the tabulation method by entries whose 1’s count equals \( k \).

T/F A 0-subcube in \( G_0 \) can always be merged with a 0-subcube in \( G_1 \) to generate a 1-subcube in \( G_0 \).

T/F A 0-subcube in \( G_n \) can always be merged with a 0-subcube in \( G_{n-1} \) to generate a 1-subcube in \( G_n \).

T/F It is possible to merge together four 0-subcubes, one in \( G_n \), two in \( G_{n-1} \) and one in \( G_{n-2} \) to generate a 2-subcube in \( G_{n-1} \).

T/F It is possible to merge together two 0-subcubes in \( G_0 \) and two 0-subcubes in \( G_1 \) to generate a 2-subcube in \( G_0 \).

(Part B) We have been given a set of implicants that are purportedly prime, generated by the tabulation method for several Boolean functions. The only information regarding the functions and the given implicants we can rely on consists of the following two pieces:

- The set of implicants we have been provided covers all the minterms.
- For each function, any multiple invocation of a particular variable (such as \( u \), or \( p \), or \( q \)) denotes the identical value, even though the exact value referred to is not specified.

We are asking you in this part to figure out for each function, whether the specified implicants in either set are prime or not. If you think any implicant is not prime, please correct the set by making the appropriate implicant(s) prime.

PIs of the 1st function: 
\[
\begin{array}{cccccc}
\text{u} & \text{} & \text{p} & 0 & \text{q} \\
\text{u} & 1 & \text{p} & 1 & \text{q}
\end{array}
\]

PIs of the 2nd function: 
\[
\begin{array}{cccccc}
0 & \text{} & 0 & \text{} & u \\
\text{} & \text{} & 1 & 1 & u \\
1 & 1 & \text{} & 1 & u
\end{array}
\]
(Part C) Once again, we have been given a set of implicants generated by the tabulation method. The only information regarding the functions and the given implicants we can rely on consists of the following three pieces:

- The set of implicants we have been provided covers all the minterms.
- All given implicants are certifiably prime.
- For each function, any multiple invocation of a particular variable (such as $u$) denotes the identical value, even though the exact value is not specified.

In this part, your job is to figure out for each function, whether the PI set is complete or not. For each function, if you think there are any missing PIs that have not been listed, please identify them.

PIs of the 1st function: $1 - u - 0$

$1 - u 1 -$

$- 1 u 0 1$

PIs of the 2nd function: $u 0 - 0 -$

$u - 1 1 -$

$u - 1 - 1$
(Part D) We have applied the tabulation method on a set of 4-variable Boolean functions. For each function, the number and types of subcubes found at the end are listed below. As you can see, each function has only two PIs, from which fact it can be immediately inferred that both PIs are essential. Accordingly, no further work is needed in identifying the minimal cover, thus enabling us to proceed directly to the implementation. For this purpose, we plan to use 2-input and 3-input NAND gates to implement each function. As you have learned from class, the NAND implementations of sum-of-product representations require inverters if (1) we need to complement an input variable for any product term, or (2) we need to decompose a 4-input AND (or OR) gate into two gates.

Your job is to figure out the minimum number of NAND gates and the minimum number of inverters needed for each function. For each case, please also briefly provide your reasoning.

- A 0-subcube in $G_1$ and a 1-subcube in $G_1$.

- A 0-subcube in $G_2$ and a 1-subcube in $G_2$.

- A 0-subcube in $G_2$ and a 2-subcube in $G_1$. 
2 (Karnaugh Maps & Boolean Decomposition)

As we have discussed in class, every Boolean function can be expressed using AND, OR and NOT gates in the standard 2-level sum-of-products or product-of-sums forms. However, in terms of gate count, the total number of gates (not counting the NOT gates) used in this standard implementation may not be minimal when compared to nonstandard representations. As a comparison, function decomposition enables us to generate factored representations, thus possibly reducing the total gate count. This decomposition can be attained through the utilization of any 2-input Boolean functions.

In this question, we examine possible decompositions using implication and inhibition gates. Recall that implication (denoted as \( +' \)) is defined as \( x +' y = x + y' \), while inhibition (denoted as \( *' \)) is defined as \( x *' y = xy' \).

To illustrate function decomposition more concretely, let’s take a look at the following example that shows one possible implication gate-based decomposition of the original Karnaugh map into two maps. As you can notice in this example, decomposition can afford us multiple degrees of freedom, as exemplified by the emergence of don’t cares in the resultant Karnaugh maps, thus possibly enabling leaner implementations.

\[
y' + zw' + xz + xw'
\]

We therefore can circle the prime implicates of the second map and the prime implicants of the third map, and hence arrive at the factored solution:

\[
F = x(z + w') +' y(z' + w)
\]
(Part A) In this part we consider function decomposition using implication (denoted as \( +' \)) gates. Given the following **Karnaugh map** of the function, \( F \), please decompose the function \( F \) into 2 parts, \( s_1 \) and \( s_2 \), such that \( F = s_1 +' s_2 \), as depicted in the following graph. Both \( s_1 \) and \( s_2 \) should be in minimum **product-of-sums** forms. As a hint and to guide you in this process, an incomplete version of the corresponding schematic is also provided. Please identify the decomposition, fill out the maps of \( s_1 \) and \( s_2 \), and complete the schematic by marking the correct inputs. (Assume inverted inputs for all literals are available at no cost.)

\[
\begin{array}{c|cccc}
  ab & cd & 00 & 01 & 11 & 10 \\
  \hline
  00 & 0 & 1 & 1 & 1 \\
  01 & 0 & 0 & 1 & 1 \\
  11 & 0 & 0 & 0 & 0 \\
  10 & 0 & 0 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{c|cccc}
  ab & cd & 00 & 01 & 11 & 10 \\
  \hline
  00 & \text{ } & \text{ } & \text{ } & \text{ } \\
  01 & \text{ } & \text{ } & \text{ } & \text{ } \\
  11 & \text{ } & \text{ } & \text{ } & \text{ } \\
  10 & \text{ } & \text{ } & \text{ } & \text{ } \\
\end{array}
\]

\[
\begin{array}{c|cccc}
  ab & cd & 00 & 01 & 11 & 10 \\
  \hline
  00 & \text{ } & \text{ } & \text{ } & \text{ } \\
  01 & \text{ } & \text{ } & \text{ } & \text{ } \\
  11 & \text{ } & \text{ } & \text{ } & \text{ } \\
  10 & \text{ } & \text{ } & \text{ } & \text{ } \\
\end{array}
\]

\[
F = ( \quad ) +' ( \quad )
\]
(Part B) The aforementioned decomposition process can also be used in the dual sense. Instead of implication gates, we can use inhibition (denoted as \(*'\) gates.

For the same Karnaugh map given in (PART A), the function \(F\) can also be decomposed into 2 parts, \(p_1\) and \(p_2\), such that \(F = p_1 *' p_2\), as depicted in the following graph. This time around, both \(p_1\) and \(p_2\) should be in minimum sum-of-products form. As a hint and to guide you in this process, an incomplete version of the corresponding schematic is also provided. Please identify the decomposition, fill out the maps of \(p_1\) and \(p_2\), and complete the schematic by marking the correct inputs. (Assume inverted inputs for all literals are available at no cost.)

\[
\begin{array}{c|c|c|c|c|c|c}
ab & cd & 00 & 01 & 11 & 10 \\
\hline
00 & 0 & 1 & 1 & 1 \\
01 & 0 & 0 & 1 & 1 \\
11 & 0 & 0 & 0 & 0 \\
10 & 0 & 0 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c}
ab & cd & 00 & 01 & 11 & 10 \\
\hline
00 & & & & \\
01 & & & & \\
11 & & & & \\
10 & & & & \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c|c}
ab & cd & 00 & 01 & 11 & 10 \\
\hline
00 & & & & \\
01 & & & & \\
11 & & & & \\
10 & & & & \\
\end{array}
\]

\[
F = ( ) *' ( )
\]
This question concerns the logic unit which performs all 16 Boolean functions of two variables. These functions, denoted as $f_0, f_1, ..., f_{15}$, are presented in the table below. Four variables, $S_3, S_2, S_1$ and $S_0$, are used to select one of the 16 functions. These selection variables are encoded in such a way that $S_i \ (0 \leq i \leq 3)$ is equal to 1 iff the minterm $m_i = 1$ for that function. A particular implementation with AND, OR and INV gates is also shown in the following figure.
For each of these 16 logic functions, please identify whether the presented logic unit implementation has static hazards or not. In other words, assuming the 4-bit selection signal $S_3S_2S_1S_0$ is stable, please identify whether the presented logic unit implementation has static hazards under *single bit flip* of either $X_i$ or $Y_i$. If you think that a function has static hazards, please furthermore identify the hazard type (static-0 or static-1), as well as the transition(s) that cause the hazard. As an example, the classification for $f_5$ has already been provided for you.

<table>
<thead>
<tr>
<th>Hazard?</th>
<th>Type</th>
<th>$x_i$ 1 to 0</th>
<th>$x_i$ 0 to 1</th>
<th>$y_i$ 1 to 0</th>
<th>$y_i$ 0 to 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_5$</td>
<td>Yes</td>
<td>static-1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_7$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_8$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_9$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_{11}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_{12}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_{13}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_{14}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_{15}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>