CSE 140 Homework Three

July 19, 2011

Only Problem Set Part B will be graded. Turn in only Problem Set Part B which will be due on July 28, 2011 (Thursday) at 4:00pm.

1 Problem Set Part A

- textbook 6.7
- textbook 6.8
- textbook 6.11
- textbook 6.12
- textbook 6.14
- textbook 6.15
- textbook 6.17
- textbook 6.19
- textbook 5.2
- textbook 5.3
- textbook 5.7
- textbook 5.10(a)(b)
- textbook 5.17(a)(b)
- textbook 5.21
- textbook 5.23
2 Problem Set Part B

1 (Sequential logic flip-flops)

Nobody can fire the brother-in-law of a company’s CEO who for *inexplicable* reasons seems to have total job security. To keep the company out of harm’s way, the sole job given to him has been the hardware instantiation at the last stage of sequential logic synthesis. Specifically, his job is to integrate the flip-flops designed into the circuit. This engineer, cognizant of the fact that nobody can fire him and utterly bored with his job, has decided to amuse himself by changing the flip-flops the designers intended the design for. In order not to push his luck utterly, the “engineer” limits himself to flipping flip-flops only within flip-flop classes, i.e., between the 2-input flip-flops, SR to JK and vice versa, and the 1-input flip-flops, D to T and vice versa.

Instead of fixing this problem by firing his brother-in-law, the CEO of the company asked the design team to analyze the effects of his brother-in-law’s mischievery in the hopes of coming up with some solutions in the sequential circuit design stage to ensure that the company does not go bankrupt as a result of his brother-in-law’s actions.

Please help the design team for each part of this question to determine which of the following cases applies:

1) Such a practical joke has no effect on the final circuit implementation, i.e., resulting in a circuit completely equivalent to the desired one in all cases. In this case, the solution would evidently be to continue designing with these flip-flops since brother-in-law’s mischievery has no impact.

   If you think this is the case, explain your reasoning as to why the resultant circuit would be equivalent to the desired one.

2) Even though such a practical joke *can* result in a final circuit with different functionality, the problem can be prevented in an early design stage, so as to preclude the incorrect operation of the resultant circuit.

   If you think this is the case, give a strategy to prevent such a problem in the design stage. Then the CEO will adopt these “design rules” so that his brother-in-law’s mischievery will have no effect.

3) Such a practical joke can result in a final circuit with different functionality, and there’s nothing the design team could do to ensure consistently that the problem can be guaranteed not to occur.

   Obviously, the CEO in this case would outlaw the use of these kinds of flip-flops in his company, since the brother-in-law’s mischievery will result in incorrect designs. Since this is obviously a painful step for the CEO to take, please convince him by showing the impact of the brother-in-law’s mischievery on the FSM shown below if this is your answer by drawing out the resultant FSM of the outcome of the practical joke.
(Part A) A circuit is originally designed with SR flip-flops, but the engineer replaces all the SR flip-flops with JK flip-flops during hardware instantiation.

(Part B) A circuit is originally designed with JK flip-flops, but the engineer replaces all the JK flip-flops with SR flip-flops during hardware instantiation.

(Part C) A circuit is originally designed with D flip-flops, but the engineer replaces all the D flip-flops with T flip-flops during hardware instantiation.

(Part D) A circuit is originally designed with T flip-flops, but the engineer replaces all the T flip-flops with D flip-flops during hardware instantiation.
2 (State Encoding Strategies)

A CAD company is considering the development of a new state encoding tool for D flip-flops that can maximize the probability of generating optimal state encodings for FSMs. A group of engineers is exploiting the various conditions of assigning adjacent encodings (code words with Hamming distance 1) to all states that have a common destination or source, more specifically, the two cases shown in the following figure.

![Encoding Case A](image1)
![Encoding Case B](image2)

**Figure 1: Two potential candidates for adjacent encodings**

The engineers do remember a rather eccentric professor from two dozen years ago discussing some similar looking diagrams, but time has taken its toll, and they are no longer quite sure under what input conditions these diagrams should be applied. As ignorance could be a great breeding ground for creativity (you do have a lot more options after all), the engineers are passionately arguing on the types of input conditions (i.e., the values of \(i\) and \(j\)) under which adjacent encodings should be assigned to \(S_0\) and \(S_1\). There are four prevailing ideas for each of the two cases:

- **Idea W**: adjacent encodings should \textbf{only} be assigned to \(S_0\) and \(S_1\) if \(i\) and \(j\) are the \textbf{same} input combination.
- **Idea X**: adjacent encodings should \textbf{only} be assigned to \(S_0\) and \(S_1\) if \(i\) and \(j\) are \textbf{adjacent} input conditions of Hamming distance 1.
- **Idea Y**: adjacent encodings should \textbf{only} be assigned to \(S_0\) and \(S_1\) if \(i\) is a \textbf{superset} of \(j\) (or vice versa), for example, \(i = 1X\) and \(j = 10\).
- **Idea Z**: adjacent encodings should \textbf{always} be assigned to \(S_0\) and \(S_1\) \textbf{independent} of the values of \(i\) and \(j\).

You are in a rather unfortunate position as the manager has tasked you with helping her make a decision as to which of these idea(s) are right. Unfortunately, as you remember from class, optimality is a probabilistic issue, implying that specific decisions may turn out best for a particular FSM but may fail on the average. The probabilistic aspect deprives you of the ability to identify the correct answer by looking at a particular example. Therefore, you need to evaluate, based on your conceptual understanding of the material, for each of the four ideas whether:

- **A** The engineers involved in promoting this idea should be \textbf{fired} because the input condition suggested by them will \textbf{never happen} in any FSM.
- **B** The engineers involved in promoting this idea should be \textbf{demoted} because their suggested input condition \textbf{cannot} generate the best encoding result probabilistically speaking for most FSMs.
- **C** The engineer involved in promoting this idea should be \textbf{promoted} because their suggested input condition on the average does end up generating the best encoding result for most FSMs.
(Part A) For Case A (two states with a common destination) shown in the left of Figure 1, please clearly present your evaluation for each engineer and also give a brief reasoning (otherwise the manager will not believe in your judgment!).

Your evaluation for Idea W:
Your reasoning:

Your evaluation for Idea X:
Your reasoning:

Your evaluation for Idea Y:
Your reasoning:

Your evaluation for Idea Z:
Your reasoning:

(Part B) For Case B (two states with a common source) shown in the right of Figure 1, please clearly present your evaluation for each engineer and also give a brief reasoning to convince the manager.

Your evaluation for Idea W:
Your reasoning:

Your evaluation for Idea X:
Your reasoning:

Your evaluation for Idea Y:
Your reasoning:

Your evaluation for Idea Z:
Your reasoning:
(Part C) After figuring out the input conditions for each of the two adjacent encoding cases, the engineers in the company now are wondering about the application priority of the two cases. More specifically, if the two cases imply conflicting encoding guidelines, one would need to adjudicate among the competing demands of the two heuristics.

Please help these engineers order the two cases by analyzing the maximum amount of benefit that each case can deliver if adjacent encodings are first assigned to the two source states in Case A or alternatively first to the two destination states in Case B.
This question concerns the comparison of two 8-bit numbers. Figure 2 presents the implementation of a 2-bit magnitude comparator that generates the greater-than and less-than indicators for two numbers $a_1a_0$ and $b_1b_0$. By serially connecting 7 of these 2-bit comparators, an 8-bit magnitude comparator can be implemented (shown in Figure 3), wherein $G_7 = 1$ iff $X > Y$ and $L_7 = 1$ iff $X < Y$.

$$G = a_1b'_1 + a_1a_0b'_0 + b'_1a_0b_0$$

$$L = a'_1b_1 + a'_1a'_0b_0 + b_1a'_0b_0$$

Figure 2: 2-bit magnitude comparator

Figure 3: Serial implementation of an 8-bit magnitude comparator

One crucial shortcoming of the 8-bit comparator presented in Figure 3 is the delay value, which is equal to seven times the delay of a single 2-bit comparator. In this question we focus on various optimization techniques that can be applied to reduce this delay value.
(Part A) One way to shorten the delay of a \( n \)-bit magnitude comparator is to change the serial implementation to a parallel implementation, so that the inputs only need to ripple through \( \lceil \log_2 n \rceil \) comparators instead of \((n - 1)\) comparators. Please implement this type of 8-bit parallel comparator by making all the necessary connections and complete the following figure. In your implementation \( G_7 \) should equal \( 1 \) \text{iff} \( X > Y \), while \( L_7 \) should equal \( 1 \) \text{iff} \( X < Y \). \text{You cannot add any more gates.}

(Part B) For the 8-bit parallel comparator presented in (Part A), it turns out that except for the four 2-bit comparators on the 1\(^{st}\) row, the other three comparators can be furthermore optimized so that each comparator can be implemented within \textbf{two 2-input AND}, \textbf{two 2-input OR}, and \textbf{two inverters}. Please first briefly tell us the insight that you think enables this optimization, and then develop the consequent logic by filling in the two maps and deriving the two minimal expressions.

The insight that enables this optimization is:

\[
\begin{array}{c|c|c|c|c}
\hline
 a_1 b_1 & a_0 b_0 & 00 & 01 & 11 & 10 \\
\hline
 00 & 00 & & & & \\
 01 & & & & & \\
 11 & & & & & \\
 10 & & & & & \\
\end{array}
\]

\[
G = \]

\[
\begin{array}{c|c|c|c|c}
\hline
 a_1 b_1 & a_0 b_0 & 00 & 01 & 11 & 10 \\
\hline
 00 & & & & & \\
 01 & & & & & \\
 11 & & & & & \\
 10 & & & & & \\
\end{array}
\]

\[
L = \]
(Part C) Another way to shorten the delay of the **serial** comparator shown in Figure 3 is to redesign the 2-bit comparator, so that the delay values of $G_{i-1} \rightarrow G_i$ and $L_{i-1} \rightarrow L_i$ can be minimized. To achieve this goal, we decide to utilize the idea of **pre-computation** that has proven effective in CLA adders. More specifically, we want to break the computation of both $G_i$ and $L_i$ into two parts, the magnitude-**generate** functions, $g_i^G$ and $g_i^L$, and the magnitude-**propagate** functions, $p_i^G$ and $p_i^L$, so that the equations for computing $G_i$ and $L_i$ can be re-written in the following form:

\[
G_i = g_i^G + p_i^G G_{i-1} \quad (1)
\]

\[
L_i = g_i^L + p_i^L L_{i-1} \quad (2)
\]

The effectiveness of this technique depends on whether we can generate the four functions $g_i^G$, $g_i^L$, $p_i^G$ and $p_i^L$ independent of the values of $G_{i-1}$ and $L_{i-1}$. In other words, the values of $g_i^G$, $g_i^L$, $p_i^G$ and $p_i^L$ should only depend on $a_1$ and $b_1$, since $G_{i-1}$ and $L_{i-1}$ are connected to the inputs $a_0$ and $b_0$ of each 2-bit comparator (except for the rightmost one). Herein you are asked to figure out the minimal **sum-of-products** expressions for each of the four functions $g_i^G$, $g_i^L$, $p_i^G$ and $p_i^L$, and subsequently to evaluate the effectiveness of this **pre-computation** technique.

\[
g_i^G = \quad g_i^L =
\]

\[
p_i^G = \quad p_i^L =
\]

The effectiveness of the **pre-computation** technique:

(Part D) After evaluating the idea of **pre-computation** for the **serial** implementation of the 8-bit comparator, we wonder whether we can apply the same idea to the **parallel** comparator shown in (Part A). Please determine whether a pre-computation of the magnitude-**generate** and the magnitude-**propagate** functions can reduce the delay of **any** of the seven 2-bit comparators shown in (Part A). If you think pre-computation is useful, please identify **at least one** 2-bit comparator that can benefit from this technique. Otherwise, please briefly explain why pre-computation is of no use in reducing the overall delay for the **parallel** implementation shown in (Part A).