CSE 140 Homework One

January 16, 2019

Only Problem Set Part B will be graded. Turn in only Problem Set Part B which will be due on January 29, 2019 (Tuesday) at 3:00pm.

1 Problem Set Part A

All questions in this part are from Roth&Kinney, 7th Edition.

- 1.1, 1.3, 1.4, 1.5, 1.7, 1.15, 1.19, 1.23, 1.26, 1.29, 1.34, 1.42, 1.44
- 2.1, 2.3, 2.5, 2.6, 2.7, 2.8, 2.11, 2.22, 2.30
- 3.6, 3.9, 3.10, 3.12, 3.15, 3.16, 3.18, 3.19, 3.20, 3.21, 3.29, 3.30, 3.33, 3.34, 3.36, 3.37, 3.38
2 Problem Set Part B

1 (Mad Multipliers)

After taking CSE 140, you are offered an internship at an obscure circuit research lab. You decide to take the job, and you quickly realize you are in for more than you bargained for. The lab is an abandoned extraterrestrial project, and has been colonized by eccentric, crazy scientists and their extraterrestrial brethren. They do not leave often, so their trends in implementation strategies for basic circuits have diverged from those of the rest of the world. This is where weird ideas go when they die.

Your employers assign you to a team of scientists developing extremely eccentric multiplication designs. In this problem, you will be working with n-bit multipliers. In the following problems, you will be evaluating some crazy multiplier designs that your teammates have cooked up. You must decide if these designs produce the correct output or not; in case you decide they don’t, then you must figure out whether it’s possible to correct the error by initializing the partial product at a particular value. This initialization value must not depend on the value of the multiplier, but you can use any information about the multiplicand you wish. If you feel unsure about your answer, you may wish to include an example of a 4-bit multiplication that illustrates your hypothesis. You may assume that the accumulator (partial product) is 2n bits long, as is your adder, so that you do not have to ponder the treatment of sign extension logic.

(Part A) Your first task is to test a new Robertson’s Multiplication circuit developed by a scientist on your team. You probe the circuit, and discover that it exhibits the following quirky behavior; it subtracts the multiplicand from the partial product when it encounters a 0 in the multiplier, and performs no operation when it encounters a 1. The only slight irregularity is that if a 0 is present in the MSB of the multiplier, it instead adds the multiplicand to the partial product. You ask the designer about it, and he explains that this is the intended functionality, although his explanation involves something about birds and background bits, and goes over your head. You are not quite sure this circuit will work as intended.

Will this behavior produce the correct product of the multiplier and multiplication? If not, can you initialize the accumulator to a value that corrects the circuit, and if so, to what value? Justify your answer.
(Part B) Another scientist asks you to evaluate his new **Booth’s Multiplication** circuit. Before you attempt this however, you decide to refresh your memory by building the Booth’s operation table. Fill out the table below with what operation each pair of bits corresponds to.

<table>
<thead>
<tr>
<th>Bit Combination</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Booth’s Multiplication Operation Table

(Part C) Now that you remember how Booth’s is supposed to work, you turn your attention to the new circuit. While perusing the schematic, you notice that instead of appending a “0” on the right of the multiplier, the circuit always appends a “1”. When you ask the designer about this, he mumbles something inaudible and wanders away in thought.

Will this circuit output the correct product? If not, can you correct the output by initializing the partial product to a value, and if so, what value? Justify your answer.

(Part D) Yet another scientist asks you to look at his Modified Booth’s circuit. Before delving into the circuit, as before, refresh your memory of Modified Booth’s Algorithm by filling in the table below to specify which operation to do on each multiplier-bit/flip-flop ($F$) value combination, and how to update $F$ on each step.

<table>
<thead>
<tr>
<th>$x_{i+1}$</th>
<th>$x_{i}$</th>
<th>$F$</th>
<th>Operation</th>
<th>Next $F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2: Modified Booth’s Multiplication Table
(Part E) Now, looking at the schematic for the circuit in (Part D), you see again that the bit appended to the right of the multiplier is initialized to “1” instead of “0”. You ask the designer about it, but your hopes of receiving a cogent answer are dashed when he does not even look up from his workbench to answer you.

Does this circuit output the correct product? If not, can you initialize the partial product to a value to correct the output, and if so, what value? Justify your answer.

(Part F) You have barely finished with the last circuit when the same designer drops another schematic on your desk and leaves without another word. This circuit is a Modified Booth’s circuit with a new kind of defect; the right-most bit is correctly initialized to “0”, but the multiplier is extended with the complement of the sign bit, instead of the sign bit.

Does this circuit output the correct product? If not, can you initialize the partial product to a value to correct the output, and if so, what value? Justify your answer.
(Parts G-H) As was stated in the exposition, the n-bit multiplication circuits have an accumulator (partial product) and adder that are both 2n bits. In these last two parts, the multiplicand is also 2n bits long, preloaded with the multiplicand value in the left word, and the right word initialized with 0s. Furthermore, in contrast to the typical multiplication algorithms which normally update the accumulator before the accumulator is shifted leftward, the multiplicand is shifted rightward before the accumulator — which no longer shifts at all — is updated.

(Part G) Another circuit. You feel your psyche degrading quickly, and soon you will be as mad as these mad scientists. This crazy circuit is a **Robertson’s Multiplication** circuit which scans the multiplier from left to right instead of from right to left. Operations are performed when a “1” is encountered, as usual; all operations are additions except an operation performed on the MSB, which is a subtraction.

Does this circuit output the correct product? If not, can you initialize the partial product to a value to correct the output, and if so, what value? Justify your answer.

(Part H) The next circuit is a **Booth’s Multiplication** circuit which also scans from left to right. At each bit position, it performs the operation specified by the normal Booth’s operation table (Figure 1 in Part B). The correct “0” value is appended to the right side of the multiplier at the beginning of each multiplication.

Does this circuit output the correct product? If not, can you initialize the partial product to a value to correct the output, and if so, what value? Justify your answer.
2 (Gray Codes)

Seeing your friends struggle with internship applications and obsess over resume reviews, you instead decide to have some peace of mind and indulge in exploring the limitations of the representational power of gray codes on a midterm. As you recall from class, the defining property of the gray code encoding is that the representations of adjacent values are only 1 bit apart from each other. **Do keep in mind that the gray code adjacency behavior wraps around.**

(Part A) **Give a valid gray code mapping scheme for the blanks in the following table.** If you are confident that it is **impossible** to assign a mapping to some value, put an **X** and briefly explain why this cannot be done.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>111</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>010</td>
</tr>
</tbody>
</table>

(Part B) **Give a valid gray code mapping scheme for the blanks in the following table.** If you are confident that it is **impossible** to assign a mapping to some value, put an **X** and briefly explain why this cannot be done.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>110</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

(Part C) Suppose that you wanted to represent all of the digits **from 0 to 7 inclusive** in some form of gray code encoding. **Would that be possible with some number of bits?** If yes, please give the smallest number of bits that would be needed AND give some valid gray code encoding for all the digits. If not, please briefly justify why.
(Part D) How about encoding all of the digits from 0 to 8 inclusive? Is this possible using some number of bits? If yes, please give the smallest number of bits that would be needed AND give some valid gray code encoding for all the digits. If not, please briefly justify why.

(Part E) Hopefully, you have started to see the pattern and have gotten an idea about the underlying reason underpinning the feasibility of gray codes. Now, let us ascend to the next level. Suppose that instead of having a code where the encodings of all adjacent digits are exactly one bit apart from each other (as in gray code), they would be exactly 2 bits apart.

Given this condition, would it be possible to come up with some \textit{hamming distance 2} code scheme for the values 0 to 3 inclusive with 3 digits? If yes, please give such an encoding map (i.e. an encoding for each value 0 to 3); if not, briefly explain the problems that arise.

(Part F) Finally, with the same constraints (3 digits and a hamming distance of 2), can you find a way to represent all of the digits from 0 to 5 inclusive? If yes, please give such an encoding map (i.e. an encoding for each value 0 to 5); if not, briefly explain the problems that arise.
3 (X(N)OR Axioms)

The theorems you have covered in class revolve around two operators: OR (+) and AND (·). However, you know there is more to Boolean Algebra than just OR and AND. Your recent work with Hamming Codes has fixed your interest towards XOR (⊕) gates, and their XNOR (⊙) counterparts. You start to wonder how some of the Theorems and Axioms you have studied change after exchanging ORs for XORs and ANDs for XNORs. After all, you’re basically just circling the operator, it can’t be that different, right?

Use ONLY XNOR and XOR operations for the duration of this question.

(Part A) You begin by refreshing your memory on the exact function of XOR and XNORs. Fill out the truth table below.

<table>
<thead>
<tr>
<th>(a)</th>
<th>(b)</th>
<th>(a \oplus b)</th>
<th>(a \odot b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To determine whether the Theorems and Axioms taught for AND and OR hold with XNOR/XOR, you figure that the best way to work is to use boolean algebra to determine equivalence between the given equation and the theorem’s presumed analogue. If the algebraic answer is equivalent to the presumed analogue, the Theorem holds. To obtain the presumed analogue, change all ANDs to XNORs and all ORs to XORs. An example is shown below:

Original Distributivity: \((w + x) \cdot (y + z) = (w \cdot y) + (w \cdot z) + (x \cdot y) + (x \cdot z)\)

\(+ \rightarrow \oplus, \cdot \rightarrow \odot\)

Presumed Analogue: \((w \oplus x) \odot (y \oplus z) = (w \odot y) \oplus (w \odot z) \oplus (x \odot y) \oplus (x \odot z)\)

(Part B) Now that you have refreshed your memory on XOR and XNOR you begin validating the various theorems and axioms of boolean algebra. You choose to start with Complement Elements, Idempotency, Identity, and Identity Absorption. For each equation fill the blank to the left of the slash according to the XOR/XNOR analogues of the Theorems and Axioms of · and +, and on the right fill the blank with the algebraically correct answer. Then state which theorems hold.

\[
\begin{array}{l}
\text{Presumed/Actual} \\
\hline
x \oplus 0 = \underline{____} / \underline{____} \\
x \oplus 1 = \underline{____} / \underline{____} \\
x \oplus x = \underline{____} / \underline{____} \\
x \oplus x' = \underline{____} / \underline{____} \\
\hline
\text{Complement Element: } \underline{____} \\
\text{Idempotency: } \underline{____} \\
\end{array}
\]

\[
\begin{array}{l}
\text{Presumed/Actual} \\
\hline
x \odot 0 = \underline{____} / \underline{____} \\
x \odot 1 = \underline{____} / \underline{____} \\
x \odot x = \underline{____} / \underline{____} \\
x \odot x' = \underline{____} / \underline{____} \\
\hline
\text{Identity: } \underline{____} \\
\text{Identity Absorption: } \underline{____} \\
\end{array}
\]
(Part C) Having tackled the basics, you try to handle DeMorgan’s theorem, as it will help you in future X(N)OR manipulations. **On the left, fill in the presumed values according to DeMorgan’s rule. On the right enter the values that make the equation algebraically valid. Determine whether De Morgan’s theorem holds after switching OR/AND for XOR/XNOR based on your two answers.**

\[(x \oplus y)' = \underline{\quad} / \underline{\quad}\]

While playing around with complements, you notice something strange about XOR and XNOR. You begin to suspect that there may be another solution to the complementation above using different inputs. After further investigation you realize that the following equation is also true:

\[(x \oplus y)' = x \odot y\]

(Part D) Although it is usually shown with only two inputs (such as in the earlier part of the question, and in the Boolean Theorems sheet at the back of the test), DeMorgan’s Theorem works for more than two inputs. **Can you generalize the equation provided \((x \oplus y)' = x \odot y\) to create a 3+ input form?** Place the presumed generalization on the left, and the algebraically correct value on the right.

\[(x \oplus y \oplus z)' = \underline{\quad} / \underline{\quad}\]
(Part E) With working rules for (Part B) and (Part C), you feel ready to tackle distributivity. Determine whether the modified distributivity below holds, and justify.

\[(w \oplus x) \circ (y \oplus z) = (w \circ y) \oplus (w \circ z) \oplus (x \circ y) \oplus (x \circ z)\]

(Part F) Encouraged by your previous success, you sink your teeth into absorption. Place the presumed absorption analogue on the left, and place the algebraically correct value on the right. Does Absorption hold?

\[x \circ (x \oplus y) = \boxed{} / \boxed{}\]

(Part G) You are finally ready to look at the consensus theorem. Perform the following equation by using the presumed consensus analogue on the left, and determining the real value for the right. Does Consensus hold?

\[(x \circ y) \oplus (x' \circ z) \oplus (y \circ z) = \boxed{} / \boxed{}\]
(Part II) All of this work has shown you several ways to alternate between XOR and XNOR. As your mind wanders, circling around the potential applications of such versatile functions, you return to your initial starting point of Hamming Codes. Standard implementations use XOR gates, but what if you chose to use XNORs instead? **Start with the hamming code parity and correction equations taught in class, and then change all XORs to XNORs to propose a new error correction scheme below.**

\[
\begin{align*}
p_0 &= \\
p_1 &= \\
p_2 &= \\
C_0 &= \\
C_1 &= \\
C_2 &= 
\end{align*}
\]

Will these modified equations still work for error correction? **If it will work, detail how to read the error codes. If it will not work, explain why.**

You can generalize the 7-4 Hamming Code to create 15-11 Hamming Code (15 total, 11 data, 4 parity). **If you play the same XOR \(\rightarrow\) XNOR gate flipping on 15-11 Hamming code, can you still use error correction?** If so, detail how to read the error codes created by the XNOR 15-11 Hamming code. If not, why not?