LAB #2: Two-Level Logic Minimization and ESPRESSO

This lab is due at 5pm in the EE locker for CS/EE 3700 on Thursday, February 4, 1999. NO LATE HOMEWORK WILL BE ACCEPTED.

1 Laboratory Objectives

In this laboratory, we will learn to do two-level logic minimization both by hand and using the tool espresso. Before using the tool, do the homework problems listed in the back of the handout. For the indicated problems, redo them using the tool espresso.

The goal of the laboratory portion of this assignment is to learn how to:

- Create truth table inputs and interpret truth table outputs for the tool Espresso;
- Use Espresso to compute the minimized sum of products form of the Boolean function;
- Apply Espresso to Boolean expressions with multiple output functions;

It is assumed that you are familiar with UNIX, and that you know how to log in, navigate your way around directories, and edit files. Place the following lines in your startup file (seek the help of a teaching assistant or a colleague in the course if none of this makes sense to you):

```
set path = (~cmyers/bin $path);
```

It is important that you attempt to minimize the equations using the pencil and paper techniques before you sit down at a computer terminal. It is quite all right if you do not obtain the same solutions as Espresso. However, for the purposes of this tutorial, we will be using Espresso primarily as a checking tool.

2 Using Espresso

Espresso takes as input a two-level representation of a Boolean function, and produces a minimal equivalent representation. The command line typed at the UNIX prompt looks like the following: espresso [options] [file].

Espresso reads the file provided (or standard input if no file is specified), performs the minimization, and writes the minimized result to standard output. If you want to keep the output, for example, to print it out for later reference, you should redirect it to a file.

Espresso comes with a rather large choice of options. Most of these are for the expert user, and will not be of concern for us (a copy of the full espresso manual page is on the webpage). We only summarize the most important options here:

```
-o[type] Selects the output format. By default, only the ON-set (i.e., type f) is output after the minimization, which is mostly what we are after. However, [type] can also be one of f, d, r, fd, dr, fr, or fdr to select any combination of the ON-set (f), the OFF-set (r), or the DC-set (d). [type] can also be eqntott to output algebraic equations acceptable to eqntott.

-eop Swaps the ON-set and OFF-set of the function after reading the function. This can be used to minimize the OFF-set of the function.
```
Input to espresso can be created in two ways, by using equations or by creating a truth table form directly. An equation file is a collection of Boolean equations separated by semicolons. For example, the expressions for the full adder would be written as:

\[
\text{sum} = \text{ain} \text{ bin'} \text{ cin} + \text{ain} \text{ bin'} \text{ cin'} + \text{ain} \text{ bin'} \text{ cin'} + \text{ain} \text{ bin} \text{ cin}; \\
\text{cout} = \text{ain} \text{ bin} \text{ cin'} + \text{ain} \text{ bin} \text{ cin} + \text{ain} \text{ bin} \text{ cin} + \text{ain} \text{ bin'} \text{ cin};
\]

+ = OR, ' = COMPLEMENT, adjacent literals = AND. Parentheses are optional if you write your expressions in Sum of Products form, since complement has higher precedence than AND, which has higher precedence than OR. However, you may use them to change the default precedence.

Before you can run espresso, however, you must translate the equation file into truth table form. This is accomplished by running sis, a program we will learn more about in the next tutorial. Type the following command:

\text{sis -t eqn -T pla < [eqn file] > [truth table file]}

This will produce the following truth table file:

```
.i 3 -- number of input variables, e.g., ain, bin, cin
.o 2 -- number of output functions, e.g., sum, cout
.ilb ain bin cin -- input variable names, separated by spaces
.ob sum cout -- output function names, separated by spaces
.p 8 -- number of non-zero truth table entries
111 10 -- truth table row ain=1, bin=1, cin=1: sum=1, cout=0
001 10
010 10
100 10
111 01
011 01
101 01
110 01
.e -- end of table
```

Note that there are some redundant truth table rows in this file (e.g., 111 is a term for sum as well as cout). Do not be concerned. These will be eliminated by espresso.

For many examples, especially if you are given them in a truth table or minterm index form, it is easier to type the truth table file directly rather than use eqntot. Further, the truth table file is the only way to express don’t care conditions. These are represented by placing a “-” in the truth table entry for the given input conditions and function.

Running espresso on the output of the above file will yield the following:
.i 3
.o 2
.na [filename]
.ilb ain bin cin
.ob sum cout
.p 7 -- number of unique reduced product terms
 001 10
 010 10
-11 01 -- cout = bc +
 100 10
 1- 01 -- ac +
 11- 01 -- ab
 111 10
.e

A "." in the truth table index part means that the particular input variable does not participate in that reduced product term. A function's reduced sum of products description is the OR of all product terms which have a 1 in the output column for that function. Espresso could not find a reduced expression for sum, but did eliminate a term and some literals from the expression for cout.

3 Experiments

After completing the problems at the end of this handout, check the last six using espresso. Using the input format as described above, prepare an input file describing the function. Run Espresso and examine the output truth table it produces. How does the result compare with your hand determined result? For the first problem, try this again, using the -o eqntot option to produce Boolean equations rather than truth tables as outputs.

Since Espresso uses the sum of products form internally, it cannot compute the product of sums form directly. However, it is possible to request that the function's complement be produced as output, using the parameter described above. You will need to apply DeMorgan's Law to obtain the result in the desired form. Try it for the first problem. What answer do you obtain? If they are not exactly the same, it may not be because you made a mistake, but rather that espresso found an alternate cover. If they are different, is the complexity of the covers the same, that is, do both solutions use the same number of product terms and literals?

Repeat the process of finding the minimum sum of products form for the rest of the problems.

4 Review

In this laboratory, we have introduced espresso, a powerful tool for minimizing Boolean expressions. Given a description of the expressions in terms of a stylized truth table, espresso produces a reduced truth table, from which the minimized Boolean functions can be determined. It can also minimize multiple functions and functions with don't cares.

The procedure of this lab has been to minimize the example equations first by hand, and then with espresso to check the results. This makes it easier to understand the espresso output format, when you already know what the answer is supposed to be. Just in case you observed a different solution, you were asked to explain why this may have happened. The most likely cause is a mistake on your part, especially in the small examples we have looked at here (its awfully hard to beat espresso at its own game), but it can be because espresso has identified an alternative solution that is just as good as yours.
Once you have the confidence that you can specify and interpret equations in espresso’s desired form, you should be able to use this tool for all your future laboratories and design projects.
5 Homework Problems

Complete the following problems by hand first. Check the last six using the tool espresso. Turn in both your hand written minimization and the input and output files from espresso. Be sure to show all your work. Include your name, your section, and your TAs name on the handin.

1. CLD, problem 2.9
2. CLD, problem 2.10(g)
3. CLD, problem 2.13(e)
4. CLD, problem 2.14
5. CLD, problem 2.18(a) (check your answer with espresso)
6. CLD, problem 2.19(a) (check your answer with espresso)
7. CLD, problem 2.23 (check your answer with espresso)
8. CLD, problem 2.26 (check your answer with espresso)
9. CLD, problem 2.27 (check your answer with espresso)
10. CLD, problem 2.28 (check your answer with espresso)