

# ECE 474A/574A Programming Project 3 - 2-Level Logic Minimizer (80 pts)

## Due November 30, 2009 (Monday) at 8:00 pm

### DELIVERABLES

#### ECE 474A Deliverables

- Brute-force and Branch-and-Bound algorithms implemented in C/C++
- Makefile

#### ECE 574A Deliverables

- Espresso Expand coupled with your own logic minimization algorithm implemented in C/C++
- Makefile
- 3-4 page report

### PROJECT DESCRIPTION

The goal of this project is to implement your own logic minimization algorithm (or a phase within logic optimization). Your tool takes in a functional description of a n-input, 1-output Boolean function, and determines the optimized equivalent. The algorithm should also output the corresponding circuit in either the Espresso file format or as a circuit specified in Verilog.

### LOGIC MINIMIZATION

Logic minimization is an active area of research, both in two-level logic minimization and multi-level logic minimization, with many exact algorithms and heuristics available. In lecture we have looked at several of these techniques including Quine-McCluskey, Petrick's method, iterated and recursive consensus method, row/column dominance, Espresso, Branch-and-Bound, and Simulated Annealing.

ECE474A students, your project will entail the implementation of two algorithms. The first algorithm will implement the brute force approach which basically looks at all possible combinations and determines the optimal solution. You will additionally implement the Branch-and-Bound algorithm which should also determine the optimal solution, ideally in less time.

ECE 574A students, you will implement the Espresso expand operation as a starting point. Using the expanded function you will then develop your own methodology drawn from previous work, a combination of techniques, or a completely novel technique developed on your own to optimize the function. Just remember that your methodology will be part of your score, if you simply append Simulated Annealing to the end of your algorithm you will not get as many points as a group who has put more thought into their algorithm. In addition to the logic minimization tool, ECE 574A students will need to provide a 3-4 page report of your logic minimizer in standard IEEE 2-column format (a sample is provided at [http://www.ieee.org/portal/cms\\_docs/pubs/confpubcenter/pdfs/samplems.pdf](http://www.ieee.org/portal/cms_docs/pubs/confpubcenter/pdfs/samplems.pdf)). Your report should minimally include a discussion (and citation) of relevant previous work, an overview of your minimization strategy, and results discussing the effectiveness of your methodology illustrated through experiments. *NOTE: You should not just implement an existing methodology, but design your own based on the algorithms we've discussed or you've researched.*

### INPUT FILE

Your minimizer will read in an input file that specifies the functionality of a two-level Boolean function. The logic function is described as a character matrix with keywords embedded in the input to specify the size of the matrix, input and output variable names, and an optional end of file marker. Keywords are specified in Figure 1. The input file format is based on the input file format for ESPRESSO. Many other options exists for ESPRESSO, but we limit this project to consider only the keywords listed in Figure 1.

Figure 2 provides a sample input file utilizing the keywords outlined in Figure 1. The first three lines are comments, and strictly for readability. The entry “.i 2” specifies there are two input variables to the function, and “.ilb a b” indicates the names of these two input variables are a and b. Additionally, the entries “.o 1” and “.ob F” specifies this is a single output function with output variable name F. The last four entries specify the functionality of F, basically constructing a truth table indicating the corresponding output value given an input combination. Variables are represented as “1” (on), “0” (off), and “-” (don't care).

Additional input files can be downloaded from the course page. When parsing these files, remember not all files will be exactly the same. For example, comments can appear anywhere in the file as long as they are preceded by a #. Additionally, an input file can have multiple empty lines or no empty lines. Make sure your parser handles these (and possibly other) cases.

Keyword	Description
# comment	Comments are allowed within the input by placing a pound sign (#) as the first character on a line.
.i [d]	Specifies the number of input variables, where [d] denotes a decimal number.
.o [d]	Specifies the number of output functions, where [d] denotes a decimal number.
.ilb [s1] [s2] ... [sn]	Gives the names of the binary valued variables. This must come after .i and .o. There must be as many tokens following the keyword as there are input variables.
.ob [s1] [s2] ... [sn]	Gives the names of the output functions. This must come after .i and .o. There must be as many tokens following the keyword as there are output variables.
.e	Optionally specifies the end of the file

Figure 1: Input file keywords.

## Output File

Your logic minimization tool should allow the user to specify the input file, the output file type, and the output file, using the following command line argument.

```
logic_min <input file> -flag <output file>
```

Two file types are possible, a Verilog file (indicated by `-v`) or a text file (indicated by `-t`). If the user enters the `-v` flag on the command line, your tool should provide an output file implementing the circuit as a Verilog module. The user should be able to compile and simulate the Verilog file in Xilinx WebPACK. *You do not need to provide a testbench.* Alternatively, if the users enters a `-t` flag on the command line, your tool should provide an output file in a text format adhering to the specifications in Figure 1. Essentially, your output file is another input file. Generating another input file can be useful because this file can be read in again by the logic minimizer (or other tools) to perform multiple optimization passes. ECE574A your program should also handle a `-e` flag which outputs the function after the Espresso expand operation ONLY, not the optimized circuit. The output file will again be a text file in the Espresso file format.

## Compilation Requirements

Your minimizer must be written in C or C++, and able to run on ece3.ece.arizona.edu. A sample makefile and test programs are provided on the course page as a reference, you may provide your own makefile is you want. You should be able to logon to ece3.ece.arizona.edu, type “make”, and execute the program provided. Similarly, the grader should only have to type “make” to compile your project and use the command line outlined above to specify the input file, output file format, and output file. *Again, file names and file format are command line arguments. You will loose points if you hardcode these parameters or utilize a menu system in your code.* All projects will be compiled and executed on ece3.ece.arizona.edu. Please make sure you either work on this machine or *frequently* test your code on this machine as you develop your project.

## Additional Resources

I will be more than happy to discuss the espresso input specification, optimization methodologies and/or algorithms, or high-level implementation details. However, ***I will not answer C/C++ syntax questions or figure out why your code does not compile*** (or seg faults). You will be responsible for the C/C++ language portion of the project. Language details can be found in a good reference book or online resources.

Figure 2: Sample input file.

```
# test1.txt
# F = a'b + ab' + ab
# F = a + b (opitmized)

.i 2
.o 1
.ilb a b
.ob F

00 0
01 1
10 1
11 1
```