10/5/2015

Lab 1: Programming Assignment

Quine-McCluskey Method

ENGG*3190 – Logic Synthesis

October 5th, 2015

Professor Omar Ahmed

Group 1

Shirley Javier 0799038

Anthony Leong 0792383

Introduction

The main purpose of this lab was to apply the basic, in-class concept of the Quine-McCluskey method and implement it as a simple CAD tool using C. The program itself must take in an input file containing the number of inputs of a Boolean function, along with its minterms and don't cares. The program must then parse these values and apply the Quine-McCluskey algorithm to generate an output file that appends all prime implicants for the given function at the end of the given input file.

Implementation

In order to generate the prime implicants of the given input using the Quine-McCluskey method, the program was implemented to systematically go through the steps below:

1. Open the input.txt file using the functions "fgets", and parse using the tags, ".i, .m, and .d", which stand of "inputs", "minterms", and "don't cares" respectively. The values were separated by using the string tokenizer function, using the tags given as the delimiters
2. Store the parsed numbers into two arrays: one containing the minterms, and one containing the don't cares. This was achieved by again tokenizing the given input using spaces as delimiters.
3. Use a "toBinary" function to convert the given decimal minterms/don't cares into binary, and store them into the first array.
4. Iterate through the first array of binary numbers and compare two binary strings at a time, while comparing each instance to see what the hamming distance between the two values is. This was achieved using the "compareBinary" function.
5. If the "compareBinary" function returns the integer 3, this means that the two compared binary strings have a hamming distance of one (3 similar bits); the strings are then combined and the proper generalized binary string is generated using the "combinePrimes" function (i.e. 0100 and 0000 become 0-00).
6. The combined binary string must then be moved into a new array, to represent the start of a new column in the algorithm. Before placing the new string into the array, the "alreadyExists" Boolean function is called to check if the string already has been placed in the second array.
7. At any point that the whole array is traversed by one string with no match, it means that the first string being compared is a prime implicant; this minterm is then placed into the primeArray, which holds all prime implicants. This is similar to not checking off a minterm when doing the algorithm by hand; this tells us right away that no reductions can be done and that the mindterm is a prime implicant.
8. The algorithm is repeated for the second column array, and all prime implicants (ones that cannot be combined with other minterms) are appended to the input.txt file, and displayed to the console.

Error Analysis and Limitations

There is currently no method in place for the program to take in more than 4 inputs, since all 16 values of binary are hard-coded. If given more time, a function called “toBinary” can be implemented in order to avoid hard-coding and give the program more room for expansion. The program also currently appends to an input.txt file and does not create a separate output.txt file.

In terms of size management, the program doesn’t currently have groupings in place for the number of ones in the minterm strings. This means that when the minterms are combined in search of prime implicants, the whole column array is traverse as opposed to what is done in class (compare group 0 with group 1, group 1 with group 2, etc.). The program also does not have minimization methods in place for the prime implicants; however, this will be expanded upon in the next laboratory.

Lastly, Valgrind was not used for debugging purposes and because of the several occurrences of dynamic allocation, it is not guaranteed that no memory management bugs are currently present in the program.

Conclusions

This laboratory was a very effective way of exploring the advantages and disadvantages of using the Quine-McCluskey method for Boolean logic synthesis. By implementing the algorithm using a low-level programming language like C, the process and logic behind the method was thoroughly explored.