Quine-McClusky

version 1.2

November 29, 1994

Overview

This manual provides a brief introduction into a program for using the Quine-McClusky method to reduce a truth table (or set of decimal functions) to a sum-of-products logic structure.

Use and Distribution

Program Limitations

This program was not designed to be user friendly (and it is not) nor was it designed to teach students about the Quine-McClusky method. It is a proof of concept that it is possible to quickly code an algorithm that performs the Quine-McClusky method.

The interface is bare bones text. One responds to prompts for number of variables and the decimal function (both output and don't cares). The algorithm is designed to be made into a library function so that it may be called to solve problems inside a program.

The code is limited to between 2 and 8 variables. It may be extended to 15 variables rather easily (need to change some data type sizes). However,

beyond this point, one may encounter serious memory shortages as the algorithm was not optimized to minimize memory usage.

Quine-McClusky Program

The Quine-McClusky method is an algorithm that reduces the terms of a combinational logic structure to a minimized sum-of-products form. Note that the result may not be the most "efficient" method of implementing the logic.

The best way to use the program is first to derive a truth table for the desired function. List the decimal function with its don't care outputs. The number of inputs to the function, outputs, and don't cares form the input to this program. After processing, the program displays the irredundant form of the prime implicants.

When entering the decimal function, each number should be separated by a space or comma. To specify a range of values, enter the lower value, a dash (‘-’), and the upper boundary. Do not include any spaces when specifying a range. If more than one line is required, terminate the line with a plus sign (‘+’).

The same decimal function may not appear in both the output and don’t cares (since a min term cannot both be an expected output and a don’t care).

Platforms

The source code is strictly ANSI C. This can be compiled on most (if not all) major computing platforms. The code has been compiled and tested on Macintosh and DOS platforms. The Macintosh version is compiled using Metrowerks C 4.5 (and a PowerMac native version is available). The DOS version has been compiled using Borland C++ 2.0.

All development and compiling was performed on a Power Macintosh 6100/80. The DOS version was built by running SoftWindows (a DOS/Windows emulator) on the PowerMac.

Example

Problem Statement

Given four inputs, create a logic structure whose output is one when two inputs are high and zero if no inputs or a single input is high. The output does not matter if more than two inputs are high since this condition should not occur.

Truth Table

The truth table for the problem is shown below. The inputs are *x0* through *x3*, with *x0* being the least significant bit. The output of the circuit is *y*.

decimal	x3	x2	x1	x0		y
0		0	0	0	0	0
1		0	0	0	1	0
2		0	0	1	0	0
3		0	0	1	1	1
4		0	1	0	0	0
5		0	1	0	1	1
6		0	1	1	0	1
7		0	1	1	1	X
8		1	0	0	0	0
9		1	0	0	1	1
10		1	0	1	0	1
11		1	0	1	1	X
12		1	1	0	0	1
13		1	1	0	1	X
14		1	1	1	0	X
15		1	1	1	1	X

Desired Function

The desired output may be specified as a sum of products. Using only the decimal functions, the output function can be written as:

$$y = \Sigma(3, 5, 6, 9, 10, 12) + d(7, 11, 13, 14, 15)$$

Quine-McClusky Program

Start running the program.

When prompted for the number of variables, enter **4** (since the problem has four inputs).

When prompted for the decimal function, enter **3,5,6,9,10,12**.

Finally, enter **7,11,13,14,15** when asked to input the don't cares.

The program will display the irredundant form as:

x1 x0
x2 x0
x2 x1
x3 x0
x3 x1
x3 x2

The logic function is therefore:

$x_1x_0 + x_2x_0 + x_2x_1 + x_3x_0 + x_3x_1 + x_3x_2$