EECS 144 & 244, Fall 2011 **Lee, Roychowdhury, Seshia**

HW 3: Logic Optimization & BDDs

Assigned: September 26, 2011 *Due: October 5, 2011*

1. (10 points) Consider the following 2 functions:

$$\begin{aligned} F &= abc + ac\overline{e} + b\overline{d} + acd + \overline{e}\overline{d} \\ G &= ac + \overline{d} \end{aligned}$$

   Show how to divide F by G using algebraic division, using the algorithm from the class slides on multi-level optimization. Show the following steps in the algorithm: (1) preprocessing by renaming complemented variables, (2) the partially computed quotient Q on each iteration of the **for** loop and the cube corresponding to that iteration, (3) the final quotient and remainder expressions.

2. (5 points) Consider the function $F = abd + abe + bc + cd$. List all the kernels and cokernels of F.

3. (10 points)

   Prove $F_x \oplus G_x \equiv (F \oplus G)_x$, where $\oplus$ denotes the XOR operation. That is, prove that the cofactor of XOR is the XOR of cofactors.

4. (30 points)

   This problem involves using PerlDD, a Perl wrapper for CUDD, the Binary Decision Diagram package from the University of Colorado at Boulder. Thanks to Fabio Somenzi for both PerlDD and CUDD. Stub code is provided for this part of the assignment.

   (a) Implement an 8-bit ripple-carry adder. Print the BDDs for each sum bit and the last carry bit to a DOT file. You should do this with a single call to `dump_dot_to_file`. An example can be found at the end of the carry-lookahead adder function.

   (b) The combinational equivalence checking problem is to determine whether two combinational circuits generate equal outputs for every possible input combination. Perform combinational equivalence checking between the ripple-carry adder implemented in the previous step and the carry-lookahead adder provided in the assignment code. *Hint: construct a circuit that is true (false) when the output bits between the two adder implementations are equal (different).* Print the BDD representing the equivalence circuit to a DOT file.

   (c) Implement the circuit shown in Figure 1. Additionally, find the optimal variable order. Report the optimal variable ordering and the number of nodes in the BDD when the

optimal ordering is used. Note, in the solution (code and written), use the same variable names as given below. Print the BDD for the output of the 8-to-1 multiplexor to a DOT file.

IMPORTANT INSTRUCTIONS:
When submitting the programming portion of this assignment, only submit the perl script, but make sure to include functions to print the BDDs to DOT format for the BDD nodes stated above! The functions to do this already exist and are in the appropriate location. All you need to do is set the arguments properly; see the example/stub functions for more information.

INSTALLATION INSTRUCTIONS:

1. Download PerlDD-0.09.tar.gz from `ftp://vlsi.colorado.edu/pub/PerlDD-0.09.tar.gz`
2. Unpack PerlDD: `tar -xzf PerlDD-0.09.tar.gz`
3. Install PerlDD: `perl Makefile.PL; make; make install`

STUB CODE:
Stub code is provided for this assignment, it can be found under Assignments/HW5 on bSpace. There are 4 examples provided in the stub code. They contain all the necessary commands that you will need to complete the assignment. If you need more information, read the man page: `man Cudd 3`. The usage instructions for `hw5.pl` are:

```
Usage   : hw5.pl [options]
Options :
          --example=1       : Example 1. OR gate
          --example=2       : Example 2. AND gate
          --example=3       : Example 3. XOR gate
          --example=4       : Example 4. 4-bit less than circuit
          --exercise=1      : 8-bit ripple carry adder
          --exercise=2      : Combinational equivalence check
          --exercise=3      : 8-to-1 multiplexor
          --dot=<filename>  : Dump DOT file to filename
                            : [default:filename.dot]
NOTE: --example and --exercise can not be specified together,
      however one must be specified. The --dot argument is optional.
```

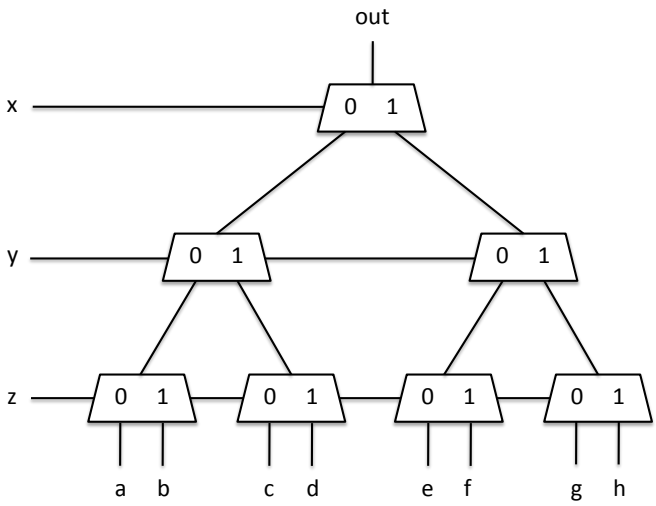

Figure 1: 8-to-1 multiplexor implemented with 2-to-1 multiplexors.