Midterm Homework and CAD Project

This assignment is due at **start of class on Thursday, March 28.**

**Grading.** This entire assignment is worth approximately 20% of your final grade. It consists of three parts: (i) CAD programming problems (Handouts #23a and #23b); (ii) SIS CAD tool multi-level application problem (Handout #23c); and (iii) short written problems (below in this Handout #23). Part (i) is the programming project; it is worth 15% of your final course grade, with part #1 (#23a) worth 10% and part #2 (#23b) worth 5%. Parts (ii) and (iii) together form a small midterm homework assignment, which should not take much work (no more than 5-6 hours); together they are worth 5% of your final course grade.

**Working in Groups.** You are only allowed to work in groups on problem (i): the CAD programming problem (Handouts #23a and #23b). For this part, you can choose either to work in a group-of-two or solo. If you work in a group, you both get the same grade. I encourage you to work in a group-of-two.

However, for parts (ii) and (iii) (SIS and written problems), you must work solo and hand in your own individual solutions.

1. **CAD Programming Mini-Project: Creating a CAD Tool for Single-Cube Extraction.** This problem is the mini-project; it is worth 10% of your final grade. It allows you the opportunity to create and test out your own CAD tool for a key step in multi-level logic optimization: single-cube extraction. See Handout #23a for details.

2. **CAD Programming Mini-Project: Creating a CAD Tool for Function “Similarity” Evaluation.** This problem is the mini-project; it is worth 5% of your final grade. It allows you the opportunity to create and test out your own CAD tool for a novel divide-and-conquer algorithm, to create a tool to rapidly determine the “similarity” of two functions. See Handout #23b for details.

3. (50 points) **CAD Tool Tutorial: Introduction to Multi-Level Optimization Using SIS.** This tutorial and problem is an introduction to multi-level optimization, using the UC Berkeley “SIS” framework. This famous public-domain tool includes many of the optimization algorithms we are covering in class (and much more!). It also forms the skeleton of many of the commercial CAD tools at companies such as Synopsys, Cadence and Mentor Graphics. Over 1600 research papers cite the original 1992 SIS paper by Sentovich et al., many of which used the tool for deriving or comparing research results; type in “SIS Sentovich” under www.googlescholar.com.

   The SIS tool is no longer actively maintained, and has some peculiarities: it has some sensitivity to whether or not you write out/read in a file, since this can reorder the data structure for the circuit; it also has sensitivity to the state of tool – whether it is rebooted or not. The tutorial includes some detailed instructions so that you can generate **reproducible results** for grading. See Handout #23c for details.
   
   (a) Write a cube-free algebraic expression.
   
   (b) Write a non-cube-free algebraic expression.

   \[ x = abd + bbf + abf + bgi. \]

   (c) Give an algebraic divisor which is a factor of \( x \), and give the corresponding quotient (do this by inspection).

   (d) Give an algebraic divisor which is not a factor of \( x \), and give the corresponding quotient and remainder (do this by inspection).

   (e) Given algebraic expression \( y = adhk + abe + abdf + adfh + bhj \), (i) list all kernels of \( y \);

   (ii) for each kernel, indicate its corresponding co-kernel.

5. (20 points) Single-Cube Extraction. You are given the following local functions (each within a separate node), forming a logic network:

   \[ x = abcf + adefj + dehj + aceg \]
   \[ y = acdef + defg + h \]
   \[ z = bcd + acef + i \]

   (a) Deriving co-kernels and kernels: For each local function (\( x \), \( y \), \( z \)), list all the kernel/co-kernel pairs. Be sure to include the trivial kernel/co-kernel if appropriate. Note: You can simply do this by inspection, rather than apply the algorithm, as long as you find the complete solution.

   (b) Single-cube extraction: Based on non-trivial co-kernel intersections, list all possible single cubes that can be used for single-cube extraction of these functions. (Note: recall that the single cube to be extracted must have 2 or more literals, and must be the result of co-kernel intersection across two or more of the local functions.)

   (c) Resulting circuit implementation: Pick the answer to part(b) with optimal 'gain', i.e. a single cube to extract which results in the greatest overall reduction in literal count in the logic network, and draw the resulting logic network after this extraction.

6. (20 points) Multiple-Cube Extraction. You are given the following algebraic expressions, forming a logic network:

   \[ x = cd + fgh + acgh + bgh \]
   \[ y = acdg + dfg + e + bdg \]
   \[ z = bd + abc + bf + h \]

   (a) Deriving co-kernels and kernels: For each local function (\( x \), \( y \), \( z \)), list all the kernel/co-kernel pairs. Be sure to include the trivial kernel/co-kernel if appropriate. Note: You can simply do this by inspection, rather than apply the algorithm, as long as you find the complete solution.

   (b) Multi-cube extraction: Based on non-trivial kernel intersections, i.e. using Brayton/McMullen’s Theorem, list all possible multi-cube expressions that can be used for multi-cube extraction of these functions. (Note: recall that the multi-cube expression to be extracted must have 2 or more cubes, and must be the result of kernel intersection across two or more of the local functions.)

   (c) Resulting circuit implementation: Pick the answer to part(b) with optimal 'gain', i.e. a multi-cube expression to extract which results in the greatest overall reduction in literal count in the logic network, and draw the resulting logic network after this extraction.