

This homework is due at the *beginning of class* on Thursday, February 9.

*NOTE:* A correct answer without adequate explanation or derivation will have points deducted. To get full credit, (a) write legibly/type, and (b) show all work (label relevant items, show derivations, include explanations).

1. (40 points) **Introduction to Espresso and Espresso-Exact.** This problem is available online as *Handout #9a*. You should also refer to the web page *Getting Starting with SIS 1.3*, as well as a very useful clickable link on that page for the *Espresso man page*.

In this problem, you will go over a brief tutorial on running an exact 2-level logic minimizer *espresso-exact* (also known as *mincov*) and a heuristic 2-level logic minimizer *espresso*, and then will perform and analyze some experiments. (Learning the tool and doing the experimental runs should take you 4-6 hours, with several hours of additional time for assembling results and answering some questions.)

2. (12 points) **Quine-McCluskey Method: Step #3.** Using the Quine-McCluskey Method, reduce and solve the prime implicant table (“constraint matrix”) shown below. *Be sure* to follow the steps in the Quine-McCluskey Handout *in order*. Iterate through as many loops as necessary to produce an empty table. (*Note:* you should not need to apply Petrick’s method in this example.) Show all work, and *clearly label* each step and iteration – i.e. for each operation, indicate which iteration you are on, and which step you are performing (refer to Handout #5 as a guide).

*NOTE:* Finally, be sure to note the convention in Handout #5 (and common in the research literature): columns represent prime implicants, and rows represent ON-set minterms. Follow this convention throughout the course. (Some digital textbooks reverse the above usage!)

	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$	$p_6$
1	0	0	0	1	1	0
2	0	1	1	0	0	1
3	1	0	0	1	0	1
4	1	0	0	1	0	0
5	0	1	0	0	1	0
6	1	0	1	0	0	0
7	0	1	1	0	1	0
8	0	0	1	0	0	1

3. (10 points) **Quine-McCluskey Method: Step #4.** For this problem, refer to the constraint matrix, which is already reduced, given below. This 4x4 matrix is cyclic. Apply Petrick’s method to solve the matrix, as follows: (a) write the constraint equation for the matrix (i.e., product of sums equation); (b) multiply out and solve the equation, finding a minimum-cost cover, using Boolean simplification. (Refer to the Quine-McCluskey Handout, Example #2.) Show all work.

	$p_1$	$p_2$	$p_3$	$p_4$
1	1	1	1	0
2	0	1	0	1
3	0	0	1	1
4	1	0	0	1

4. (16 points) **Multi-Output Functions, Representations, and Minimization.** An incompletely specified 3-output function is given by the following minterm expansions:

$$f_1(a, b, c) = \Sigma m(0, 4) + \Sigma d(2, 3)$$

$$f_2(a, b, c) = \Sigma m(0, 2, 3, 7)$$

$$f_3(a, b, c) = \Sigma m(4, 6, 7) + \Sigma d(0)$$

Given this multi-output function, do the following:

- Write a single multi-output truth table.
  - Write a *PLA input file* for the multi-output function (also called “*cubical representation*” or “*cubical complex*”), using cubes to indicate the ON-set and DC-set (see Handouts #6 and #9a).
  - Draw a *hypercube representation for the multi-output function*, indicating ON-set and DC-set minterms. Mark ON-set minterms with a black dot, DC-set minterms with an *X*, and OFF-set minterms with a white dot.
  - On your figure in part(c), neatly mark all *multi-output prime implicants* (do by inspection; you do not need to use the Quine-McCluskey algorithm to derive these!).  
*Note:* Refer to Handout #6 for discussion of multi-output primes. Remember that “multi-output primes” may contribute to 1, 2, or 3 of these outputs, but in all cases cannot be expanded further in the multi-output function without hitting the OFF-set (i.e. 0 minterms). *Each multi-output prime is maximally expanded.*
  - List the multi-output primes of part(d) in a PLA file representation, as presented in Handouts #6 and #9a.
  - Indicate all *essential multi-output primes*.
  - Draw a *minimum-cost cover* of multi-output prime implicants (do by inspection; you do not need to use the QM algorithm to derive these). Do not modify the primes!
  - Write your solution to part (g) *cubical representation*, as presented in Handout #6 and #9a.
5. (10 points) **Espresso’s EXPAND Step: Cube Ordering Heuristic.** You are given a single-output binary-valued function  $f(a, b, c, d)$ , specified as follows:

$$F^{ON} = a'd' + a'bc + abc'd + a'bc'd$$

$$F^{OFF} = abc + ab' + a'b'd + abd'$$

$$F^{DC} \text{ is empty}$$

For this problem, assume that the initial “seed cover”  $F$  is simply  $F^{ON}$ . You will not do the complete “expand” step in this problem, just the first operation: determining cube order for expansion. (Note: you will not need to use  $F^{OFF}$  for this problem, it is only given for completeness.)

- Initialization.* Do the following: (i) *draw the Karnaugh map*, and label each cube of the ON-set and OFF-set; and (ii) *write the PLA file* for seed cover  $F = F^{ON}$ .
- Determine Final Cube Order for Expansion.* Following the method presented in lecture #2, and in the Lecture #2 (part 2) slides, determine the order (from first to last) of the cubes for expansion. *Show all work:* (i) calculate the literal weight vector; (ii) calculate cube weights (for each of the 4 cubes in seed cover  $F$ ), and show the resulting cube weight table; and (iii) sort the cubes by weight (low to high), and show the resulting sorted cube weight table.

6. (12 points) **Binate Covering Problem.** In Handout #5 (Quine-McCluskey Method, example #2), Petrick's method was introduced, which uses a product-of-sums (POS) formula on Boolean propositions (i.e.  $P_i$  variables), or equivalently a constraint matrix, to represent basic covering constraints. Such a POS formula, where propositions (i.e. Boolean variables) are always positive (i.e. uncomplemented), represents a *unate covering problem*.

A *binate covering problem* is a more general type of constraint formulation. In this case, a POS formula is also used, but the propositions (i.e. Boolean variables) can now appear in both *complemented* and *uncomplemented* form. For example, the following is a binate constraint:  $P = (P_1 + P_2')(P_3' + P_4)$ . Binate constraints are used to capture more complex covering requirements, such as *implication*, *mutual exclusion*, or other correlations between the selection and non-selection of various resources. Several important CAD optimization problems can be formulated using binate covering.

*Example: Capturing an Implication with a Binate Constraint.* Binate problems can easily be used to capture implication: i.e. the selection of some variable  $P_i$  may *require* the selection (or non-selection) of another variable  $P_j$ . The basic Boolean identity is:  $(P_i \Rightarrow P_j) \equiv (P_i' + P_j)$ . This identity says that the implication on the left can be equivalently represented as a Boolean sum with the first term complemented. Hence, if we wanted to represent the constraint "if  $P_1$  is selected, then  $P_2$  must also be selected", it is equivalent to the implication  $(P_1 \Rightarrow P_2)$ , which in turn is captured by the simple binate clause  $(P_1' + P_2)$ . This binate clause is identical to the original implication. It can be satisfied by setting  $P_1$  to 0, (i.e.  $P_1'$  to 1), or setting  $P_2$  to 1, or both. However, the clause is not satisfied if  $P_1$  is 1 but where  $P_2$  is 0; this is the case where the implication is also violated.

In this exercise, you are to write a simple POS formula for a binate covering problem, and then solve it.

The following is a unate constraint matrix:

	$P_1$	$P_2$	$P_3$	$P_4$	$P_5$	$P_6$
m1	X				X	
m2			X	X		
m3		X	X			
m4	X	X				
m5					X	X
m6				X		X

This matrix can be mapped to a simple product-of-sum equation, where each term is a single uncomplemented variable:

$$P = (P_1 + P_5)(P_3 + P_4)(P_2 + P_3)(P_1 + P_2)(P_5 + P_6)(P_4 + P_6)$$

For this problem, you will modify this formula, to incorporate *additional binate constraints*:

- (i) if the solution includes  $P_3$ , then it must also include  $P_5$ ;
- (ii) if the solution includes  $P_4$ , then it cannot include  $P_3$ ;
- (iii) the solution must include *at least one* of  $P_2$ ,  $P_3$  and  $P_4$ ;
- (iv) the solution *cannot simultaneously include* all three of  $P_3$ ,  $P_4$  and  $P_6$ ;
- (v) if the solution includes  $P_1$ , then *exactly one* of  $P_3$  or  $P_4$  must be included;
- (vi) if the solution does not include at least one of  $P_4$  or  $P_5$ , then it must include  $P_3$ ;
- (vii) if the solution includes  $P_2$ , then it must include *both*  $P_1$  and  $P_6$ .

*What to Do:*

- (a) For each of the above new constraints, (i) to (vii), *write the corresponding new constraint clause or clauses*. (Note: in some cases, the clauses may be unate; in other cases binate.) In most cases, you will have 1 simple clause per constraint, but in some cases you may have the product of 2 or more simple clauses per constraint.

*Note:* each simple clause must be the OR (i.e. disjunction) only of *propositions* (i.e. variables  $P_i$  or their complements  $P_i'$ ). *No products* of propositions are allowed within a clause.

- (b) Write the resulting binate formula, which combines these new constraints with the original unate constraints, in one composite equation.
- (c) By inspection, determine if the equation has any feasible solution. That is, either give a legal satisfying assignment of 0/1 values to the decision variables ( $P_1$  through  $P_6$ ), which makes  $P$  true; or give a clear argument why no legal solution exists (i.e. the problem formulation is over-constrained and cannot be solved).

*Note: do not multiply out the POS equation!* Instead, by inspection, (i) determine if  $P$  has any satisfying solution; (ii) if it does, provide a minimum-cost solution; and finally, (iii) justify your answer to part (i), by either showing how every unate and binate clause is satisfied, or by demonstrating how the clauses can never all be satisfied.