CSE 140 Homework Four

March 4, 2019

*Only **Problem Set Part B** will be graded. Turn in only **Problem Set Part B** which will be due on March 15, 2019 (***Friday***) at 3:00pm.*

*Submit homework via gradescope <gradescope.com> as a **PDF.** Other formats may or may not be viewed and evaluated accurately.*

1 Problem Set Part A

All questions in this part are from Roth&Kinney, 7th Edition.

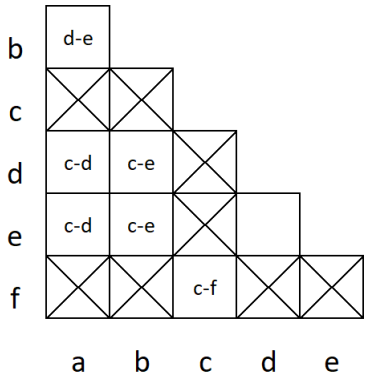
- 15.10, 15.11, 15.13, 15.14, 15.15, 15.17, 15.24, 15.26, 15.27, 15.28, 15.33
- 9.13, 9.14(a), 9.16, 9.18, 9.19, 9.20, 9.21, 9.27, 9.29, 9.31, 9.36

2 Problem Set Part B

I. (State Minimization)

After a stirring quarter of learning digital design, you decide to pack up and throw away all the scratch papers. You find a few raw implication charts and can't help but minimize them. However, you feel that the conventional algorithm requires scanning over the table iteratively, which is cumbersome. You start pondering if you can come up with another algorithm to reduce the complexity.

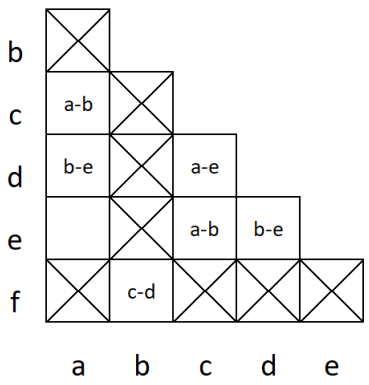
(Part A) You begin by refreshing your memory on the state minimization method using the implication table. Following the standard algorithm taught in class, **cross out non-equivalent entries in the chart and list out all the equivalent states.**



(Part B) Stepping towards an efficient state minimization algorithm, you decide to devise a new train of thought that delivers deterministic information as early as possible. The standard algorithm *constantly prunes out non-equivalent states*, and the entries remaining indeterminable at termination are interpreted as denoting *equivalent* states. You start to wonder if you can design an algorithm that **keeps building up equivalences instead with the remaining indeterminable entries categorized as non-equivalent states this time around.**

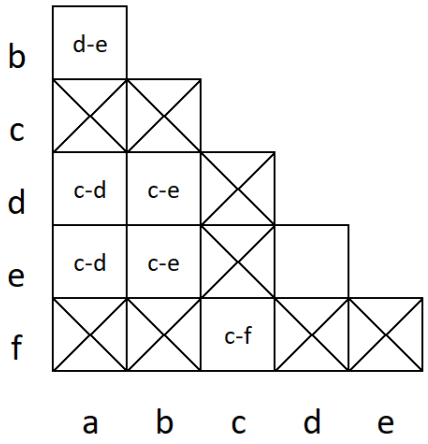
For example, if α and β are equivalent and the entry for γ and ϵ *only* contains $\alpha - \beta$, we can conclude that γ and ϵ are definitely equivalent without checking any other entries.

You may have noticed in the following chart **states a and e are definitely equivalent**, and this observation becomes the natural starting point for the new algorithm. Using the new algorithm, list out the equivalent states **in the order that they are derived.** *If multiple equivalent states are found in one step, put them at the same level.*



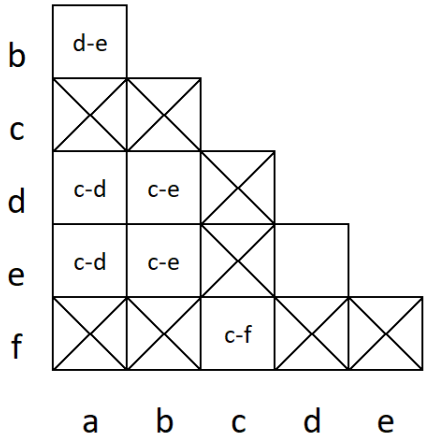
a ≡ e
↓

(Part C) Inspired by your success (or at least partial success), you want to verify your new algorithm on a more complex implication chart which is actually the same one you have seen in (Part A). Using the new algorithm, list out the equivalent states **in the order that they are derived.** *If multiple equivalent states are found in one step, put them at the same level.*



(Part D) Compare your answer in (Part C) with that in (Part A). **Does the new algorithm find all the equivalent states?** If yes, do you think the new algorithm will be correct for any implication table? Explain why. If no, list out all the equivalent states in the example that are **not detected** by the new algorithm, and generalize a condition under which the new algorithm will fail.

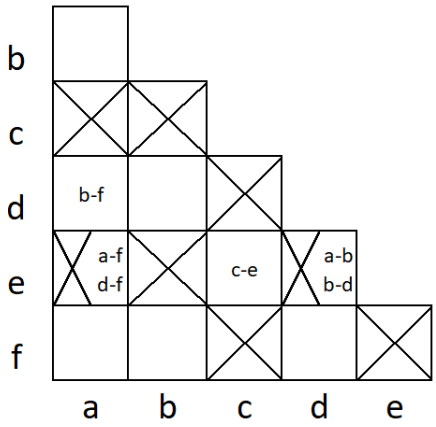
The same implication chart of (Part A/C) is reproduced for your convenience.



(Part E) You turn the scratch paper around and a quite unexpected weird implication chart hits you. You notice that some of the entries marked as non-equivalent with a big X also have state pairs scribbled but to the side of the entry. While you are used to seeing entries that are subsequently crossed over once non-equivalence is established, this implication chart looks different. You recall that this implication chart is an experiment on your hypothetical **opposite chasing method.** In this method, instead of showing the conditions for two states to be equivalent, the entries actually represent the **implied non-equivalent states when the coordinate states are non-equivalent.** Another way to visualize the difference in the initial construction of this table is that the entries represent the source states that transition to the coordinate states rather than the sink states (always on the same inputs) that we put for the technique studied in class.

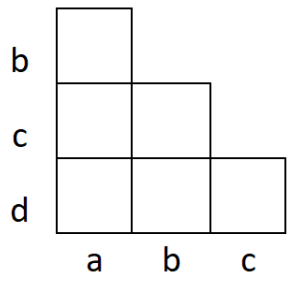
For example, in this new representation when γ and ϵ are non-equivalent and the entry for γ and ϵ contains $\alpha - \beta$, we can conclude that α and β are also non-equivalent. You quickly realize that the reason for experimenting with this approach of filling in the table must have been the speed with which non-equivalence information can be propagated without having to scan the whole table.

Given the construction rule for the mutated implication chart, find out all the equivalent states.



(Part F) Construct a minimal reduced state transition diagram for the FSM in (Part E).

(Part G) **Do you think that the new method can always rule out all the non-equivalent states?** If yes, explain how this method avoids pruning equivalent states. If no, fill in the implication chart using the new construction rule such that the *opposite chasing method* will fail.



II. (D and T Flip-Flop State Encoding)

In class, you heard about two priority adjacency heuristics (H_1 and H_2) that are used to help simplify the next-state logic of D flip-flops. Applying these heuristics helps maximize the *number of commonalities* between the next state logic of *hamming adjacent states* and *hamming adjacent input combinations* on the next-state Karnaugh Map. These heuristics are reproduced below for your convenience:

1. H_1: If two states S_i and S_j transition to the same state S_k on the same input I, then the encodings of S_i and S_j should be hamming adjacent.
2. H_2: If a state S_i transitions to S_j on input I_0 and to S_k on I_1, and I_0 and I_1 are hamming adjacent, the encodings of S_j and S_k should be hamming adjacent.

The number of commonalities induced in the Karnaugh Map for each of these heuristics is the number of bits that the next state designations in each of the heuristics have in common with each other. However, if the starting state designations are greater than hamming distance 2 apart, then the number of commonalities is 0 (because these commonalities are not useful).

For example if the state "1100" transitions to "0000" and "0001" on hamming adjacent inputs, there are 3 commonalities. If "1100" and "1111" transition to "1110", there are no commonalities because "1100" and "1111" are hamming distance 2 apart.

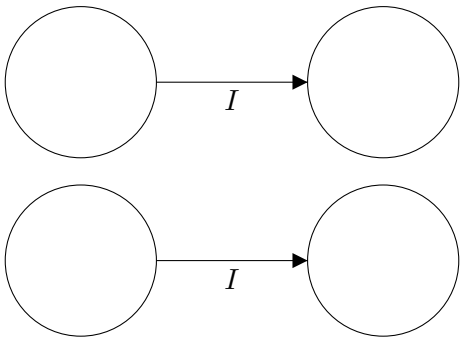
(Part A) Fill out the table below with the number of commonalities induced per application of the H_1 and H_2 heuristics in a **D** flip-flop implementation in terms of n, the number of bits in the state encoding.

Heuristic	$n = 2$	$n = 3$	n
H_1			
H_2			

(Part B) Now suppose we apply these same heuristics to a **T** flip-flop implementation of an FSM. Fill out the table below with the number of commonalities induced per application of the H_1 and H_2 heuristics in a **T** flip-flop implementation in terms of n, the number of bits in the state encoding.

Heuristic	$n = 2$	$n = 3$	n
H_1			
H_2			

(Part C) There is another heuristic that when applied to a T flip-flop implementation produces a perfect number of commonalities in the next-state logic (i.e., n commonalites in the n number of bits in the state encoding) when you transition on the same input I. Please fill in the states (denoted by circles) in the diagram below with an example of this new heuristic by giving each state your favorite 4 bit encoding so as to induce perfect commonality. In addition, please explain the heuristic in general terms just as we have explained the H_1 and H_2 heuristics at the beginning of the question.



(Part D) Not only do these priority adjacency heuristics help create bigger prime implicant cubes within each next-state Karnaugh Map, but they also help create chances for sharing prime implicants across the Karnaugh Maps of the different flip-flops. For example, if after applying H_1 and H_2 the next state logic for the flip-flop D_0 is $ab + bc'$ and the next state logic for the flip-flop D_1 is $ab + ab'c'$, then the prime implicant ab is shared. This sharing can come in handy if we were to implement the FSM's next-state logic in a PLA.

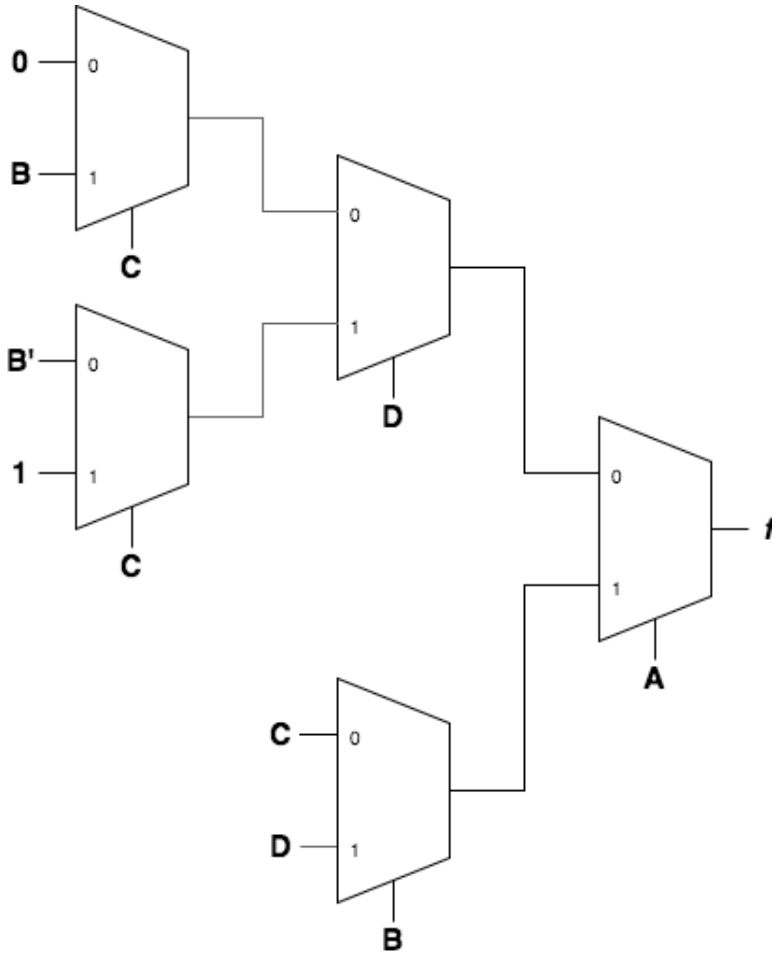
This is great for any 1s that are shared across Karnaugh Maps, but what about the 0s that are shared? The commonalities help make bigger implicants on some of the Karnaugh Maps, but they also help create bigger implicates on others. Suppose we are allowed to insert inverters at the outputs of the implicant/implicate gates of the 2-level form of the next state logic. Can we utilize these inverters in order to share the implicants and the implicates across Karnaugh Maps? Explain how this change could accommodate this sharing, or if it cannot, explain why not.

III. (MUX Ado About MUXes)

One fine Sunday morning, you get up and head off to your normal job, recalling that there seems to be some large sequential project in the works. As you arrive, you are filled in on the details, and told that your branch would be focusing on the datapath elements of the circuitry. As various engineers give their brief presentations, you cannot help but continue to stare at one group who had a major screw up on the last company project.

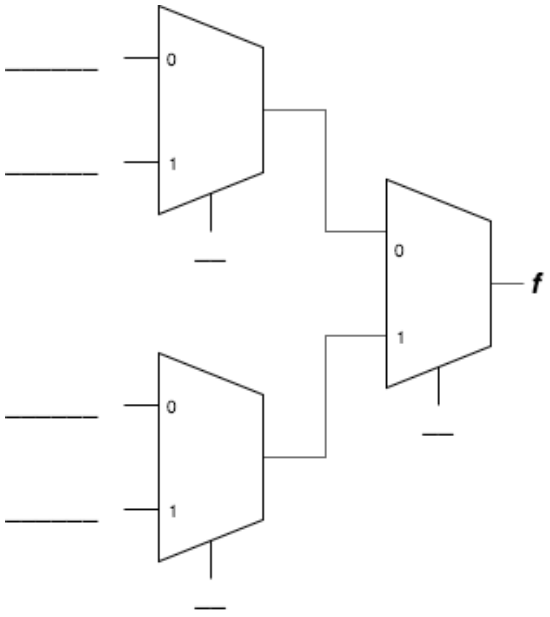
(Part A) Looking at their MUX circuit blueprint, you shake your head in disbelief and decide to investigate their circuit. **Examine the function that the circuit implements, and give its minimal SOP representation.**

Hint: You may always make use of a K-map to ensure minimality after obtaining the function implemented by this circuit. As a sanity check, ensure that your function has 8 minterms.

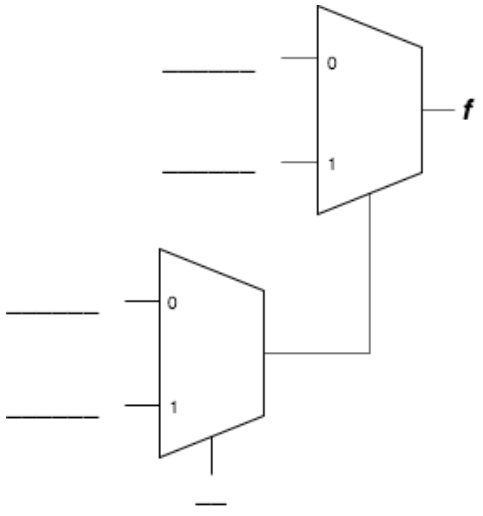


(***SOP***) $f =$

(Part B) Wanting to best that group, your colleague convinces you to implement the circuit yourself and reminds you that Shannon Decomposition often comes in useful in MUXifying equations. You recall that in performing this decomposition, you have to make choices about the order in which you factor the variables, and decide to try decomposing the circuit from (Part A) with a different ordering than the one used in (Part A). **Please fill in the MUX circuit below in such a way as to only use 3 MUXes, and only have literals at the inputs (you may have complemented literals as well).**



(Part C) After finishing your decomposition, you hear a loud cheer from the other end of the workshop. Apparently, those fellows actually managed to reduce their own circuit to the same size. Reluctant to give up, your colleague tells you to try harder. A little stumped, **you wonder if putting a function into the selector line** may further simplify the circuit to two MUXes, and notice that ***A*** **and** ***B*** **are the most common literals** across all of the SOP terms. **Please complete the circuit below such that you only have literals at the inputs (you may have complemented literals as well).**



(Part D) As you walk into your supervisor's office to show him your MUX design, you witness one of your more reclusive engineers argue that MUXes cannot be used for the project as they are prone to static hazards. **He argues that in a basic 2-to-1 MUX, the presence of the inverter on the selector line leads to unwanted intermediate outputs when switching between states**, and that this is dangerous for the project. As he leaves the room, your supervisor tells you to investigate this issue. You decide to examine the general case of a **2-to-1 MUX with a selector line S, and inputs A at $S = 0$, and B at $S = 1$. Recall that the function of this MUX is $AS' + BS$.**

(i) Please fill in the following tables with the value of the function (symbolically as functions of A & B) depending on the selector line. For both of the tables, the middle column represents the intermediate value of the function which occurs when one line has changed its value, but the other not yet due to extra delay of the inverter. The first entry has been filled in for you.

selector line:	0	$0 \to 1$	1		1	$1 \to 0$	0
function value:	A						

(ii) Based on the table that you have just completed, was the engineer correct in his concern about static hazards? **If so, state the input combinations under which a hazard can occur. If not, explain why there are no hazards.**

(Part E) Your supervisor decides to err on the side of caution and tasks you with developing a MUX-like component that **functionally implements the function $AS' + BS$ but is guaranteed not to contain static hazards.** Please develop such a component using as few extra logic gates as possible.

On the left K-map, please circle the implicants in the original MUX. On the right K-map, circle the implicants in your minimal hazard-proof MUX design.

original MUX

S \ AB	00	01	11	10
0				
1				

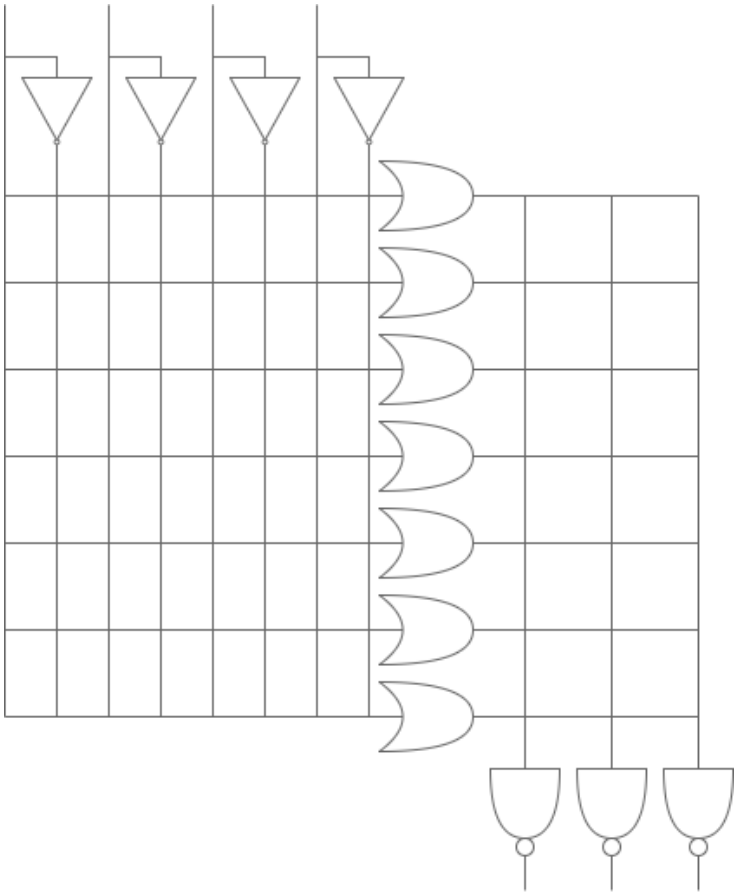
hazard-proof MUX

S \ AB	00	01	11	10
0				
1				

(Part F) Although the typical practice in your firm is to use LUTs for functions with four or fewer variables, your supervisor is fond of these new OR-NAND PLA components that the firm just got ahold of, and asks you to make use of one of them. Although you can argue that optimizing for largest implicants/implicates does not matter in a PLA, **please implement the function given by the K-map below in the PLA by taking the largest implicant/implicate terms nonetheless. Make sure to clearly indicate the inputs and the outputs, and put a cross on the interconnects that you choose to use.**

Hint: you may think of an OR-NAND PLA as an OR-AND PLA with the output complemented.

CD \ AB	00	01	11	10
00	1	1	0	0
01	1	1	1	1
11	0	0	1	0
10	0	1	1	0



(Part G) Shortly after production, it emerges that there is a minor flaw in the design, but as PLAs are designed to be re-programmable, you suppose that this should not be an issue. It turns out, however, that these were purchased at a large discount price and all of their re-programming logic is completely broken, meaning that you cannot modify the circuit already implemented in (Part F) (you can still program unused row and column lines of course).

The actual problem is that one of the minterms in the K-map you just saw in (Part F) was in an incorrect position: **the correct function should output a 0 given an ABCD input of 0101, and a 1 given an ABCD input of 0111.**

Your colleague proposes that, since only one output was used, a simple solution would be to use one of the other output lines to **implement the correct behavior on top of the existing circuitry from (Part F)** and simply ignore the erroneous output line. Is this a viable fix? **If yes, please complete the new PLA schematic below (including your lines from (Part F)). If no, please elaborate on why this cannot be done.**