

CSE 140 Homework Four

December 4, 2014

*Only **Problem Set Part B** will be graded. Turn in only **Problem Set Part B** which will be due on December 12, 2014 (Friday) at 4:30pm.*

1 Problem Set Part A

- Roth&Kinney, 6th Ed: 9.14
- Roth&Kinney, 6th Ed: 9.18
- Roth&Kinney, 6th Ed: 9.19
- Roth&Kinney, 6th Ed: 9.27
- Roth&Kinney, 6th Ed: 9.29
- Roth&Kinney, 6th Ed: 9.31
- Roth&Kinney, 6th Ed: 9.36
- Roth&Kinney, 6th Ed: 11.11
- Roth&Kinney, 6th Ed: 11.15
- Roth&Kinney, 6th Ed: 11.16
- Roth&Kinney, 6th Ed: 11.18
- Roth&Kinney, 6th Ed: 15.10
- Roth&Kinney, 6th Ed: 15.11
- Roth&Kinney, 6th Ed: 15.13
- Roth&Kinney, 6th Ed: 15.14
- Roth&Kinney, 6th Ed: 15.15
- Roth&Kinney, 6th Ed: 15.17
- Roth&Kinney, 6th Ed: 15.18
- Roth&Kinney, 6th Ed: 15.19
- Roth&Kinney, 6th Ed: 15.24
- Roth&Kinney, 6th Ed: 15.26
- Roth&Kinney, 6th Ed: 15.27
- Roth&Kinney, 6th Ed: 15.28
- Roth&Kinney, 6th Ed: 15.33

2 Problem Set Part B

1 (Reverse Implications)

As you can recall in class, implication tables are used to identify equivalent states in a finite state machine. The steps of using this table follow:

1. Cross off any pairs of states that produce different outputs
2. For each pair (i, j) , write the pairs of states which must be equivalent for i and j to be equivalent
3. For each non-crossed off square (i, j) , check if their equivalence depends on a crossed out square; if so, cross out (i, j)
4. Repeat until examination of all remaining non-crossed off squares yields no further crossing off

(Part A) As a first step, we give you the implication table below on which we've already executed steps 1 and 2. Please apply steps 3-4 and then note the sets of equivalent states.

B	A,B D,E			
C				
D			B,E	
E			A,B B,E	A,B
	A	B	C	D

Current State	Next State		Output
	0	1	
A	B	D	0
B	A	E	0
C	A	B	1
D	A	E	1
E	B	E	1

(Part B) Assume you currently have two equivalence classes Φ and Ψ . In addition you have a set of states, Ω , which transition to states in Ψ on some input i . If some states in Φ are in Ω but others are not, which of the following do you agree with as a possible step in a new state minimization algorithm?

- You must split Ψ into two equivalence classes to ensure Φ does not straddle Ω
- You must split Φ into two equivalence classes to ensure Φ does not straddle Ω
- Both Ψ and Φ must be split to ensure Φ does not straddle Ω
- Neither Ψ nor Φ must split as there is nothing wrong with Φ straddling Ω
- While one of Ψ or Φ must split, further information regarding Φ 's transitions on input i are needed to choose the correct split.

(Part C-E) While exploring the implication table algorithm, your friend who is trying to implement it notes that it seems step 4 could be improved. His observation is that once you know a pair (i, j) is crossed off then every cell with (i, j) written in it can be crossed off. However, since non-equivalence flows against the pointers, we need to repeatedly scan all cells until we can no longer cross things off. To remedy this, your friend suggests reversing the pointers, i.e. if (a, b) will be crossed out if (i, j) is crossed out, then you write (a, b) in (i, j) . This yields the following algorithm:

1. For each pair (i, j) , write the pairs of states that will be non-equivalent if i and j are non-equivalent
2. Cross off any pairs of states that produce distinct outputs
3. Every time you cross off a square, cross off all pairs written in the square (apply this recursively)

(Part C) To explore this algorithm, we ask you to execute step 1 on the FSM you analyzed in (Part A). We've reproduced the next state table and an implication table for you to use. Please populate the pointers generated by the reverse implication table; do not cross off any boxes as that step now takes place in step 2 of the new algorithm.

B				
C				
D				
E				
	A	B	C	D

Current State	Next State		Output
	0	1	
A	B	D	0
B	A	E	0
C	A	B	1
D	A	E	1
E	B	E	1

(Part D) After news of this new algorithm spreads, your classmates begin debating the maximum number of pointers that need to be written. After hours of debate, most of the class falls into 4 camps:

1. The reverse algorithm requires fewer pointers than the conventional algorithm
2. The two algorithms need the same number of pointers
3. The reverse algorithm requires a constant multiple more pointers than the conventional algorithm
4. The reverse algorithm requires an unbounded number of pointers over the conventional algorithm

With which group do you agree? Explain why.

(Part E) Your friend claims that this algorithm has no need to repeatedly search for squares to cross out as required in the conventional algorithm. Is this correct? In particular, will all non-equivalences be crossed off after examining every state once?

2 (State Encoding)

As you have learned in class, the first two rules of the prioritized adjacency state encoding heuristic attempt to cluster common values in order to enable the generation of larger cubes to simplify the next-state logic. The rules are reproduced below for your convenience:

1. If two states S_i and S_j both transition to state S_k on the same input a , then the encodings of S_i and S_j should be hamming adjacent.
2. If a state S_i transitions to state S_j on input a and to state S_k on input b , and a and b are hamming adjacent, then the encodings of S_j and S_k should be hamming adjacent.

The benefits of these rules are readily apparent should one apply them in a D flip-flop implementation, as the first rule induces commonalities in all bits in question of the next-state logic on the Karnaugh Map, while the second rule induces commonalities in all but one bit on the parts of the next-state logic affected by the rule on the Karnaugh Map. In this question, we ask that you consider the impact of these heuristics with respect to the other 3 flip-flop implementations: T, SR, and JK.

Since the two rules we discussed induce adjacent commonalities in the Karnaugh Map, we ask you to focus in your answers on the effect of flip-flop selection on the number of commonalities. As you know, a commonality is violated when the two adjacent bits differ, i.e. one is a “0” while the other is a “1”. We will stick to the same definition of commonality even in the face of “don’t-cares,” implying that the only time when a commonality is violated is when the adjacent bits have complementary values. Consequently, not only pairs such as “00” and “11” but also “0X”, “X0”, “X1”, “1X”, and “XX” should also be construed to display commonality.

Some of you may find it helpful to draw Karnaugh Maps displaying the induced commonalities in the case of D flip-flops and think about the impact of the various flip-flops on adjacent pairs displaying these commonalities.

(Part A) The first implementation to consider is that constructed with **T flip-flops**. Because T flip-flops have the same number of inputs as D flip-flops, at first glance one would expect the number of commonalities to be identical to that of the D flip-flop. However, further analysis might yield one of the cases enumerated below. **For each of the two aforementioned rules, please choose an option below and provide adequate justification** as to how the number of pairs with guaranteed commonalities in a T flip-flop implementation compares with that of a D flip-flop implementation. Your answers for the two rules may not be the same.

- I:** The number of commonalities induced by the rule is identical to what is present in a D flip-flop implementation.
- II:** The number of commonalities induced by the rule exceeds those in the D flip-flop implementation by 1.
- III:** The number of commonalities induced by the rule is 1 less than those in the D flip-flop implementation.
- IV:** The number of commonalities induced by the rule exceeds those in the D flip-flop implementation by a fixed, known quantity greater than 1 (but you don't need to report the exact quantity).
- V:** The number of commonalities induced by the rule is less than those in the D flip-flop implementation by a fixed, known quantity greater than 1 (but you don't need to report the exact quantity).
- VI:** The number of commonalities induced by the rule exceeds those in the D flip-flop implementation by a quantity that can only be determined on a case-by-case basis.
- VII:** The number of commonalities induced by the rule is less than those in the D flip-flop implementation by a quantity that can only be determined on a case-by-case basis.

(Part B) The next implementation to consider is that constructed with **SR flip-flops**. Because SR flip-flops double the number of inputs, one would expect that by using the 2 rules above, the commonalities in the bits of the next-state logic should double as well. However, further analysis might yield one of the cases enumerated below. **For each of the two aforementioned rules, please choose an option below and provide adequate justification** as to how the number of pairs with guaranteed commonalities in an SR flip-flop implementation compares with that of a D flip-flop implementation. Your answers for the two rules may not be the same.

- I:** The number of commonalities induced by the rule is exactly double what is present in a D flip-flop implementation.
- II:** The number of commonalities induced by the rule exceeds double those in the D flip-flop implementation by 1.
- III:** The number of commonalities induced by the rule is less than double those in the D flip-flop implementation by 1.
- IV:** The number of commonalities induced by the rule exceeds double those in the D flip-flop implementation by a fixed, known quantity greater than 1 (but you don't need to report the exact quantity).
- V:** The number of commonalities induced by the rule is less than double those in the D flip-flop implementation by a fixed, known quantity greater than 1 (but you don't need to report the exact quantity).
- VI:** The number of commonalities induced by the rule exceeds double those in the D flip-flop implementation by a quantity that can only be determined on a case-by-case basis.
- VII:** The number of commonalities induced by the rule is less than double those in the D flip-flop implementation by a quantity that can only be determined on a case-by-case basis.

(Part C) The final implementation to consider is that constructed with **JK flip-flops**. Again, because JK flip-flops double the number of inputs, one would expect that by using the 2 heuristics above, the commonalities in the bits of the next-state logic should double as well. However, further analysis might yield one of the cases enumerated below. **For each of the two aforementioned rules, please choose an option below and provide adequate justification** as to how the number of pairs with guaranteed commonalities in a JK flip-flop implementation compares with that of a D flip-flop implementation (the choices are identical to that in Part B). Your answers for the two rules may not be the same.

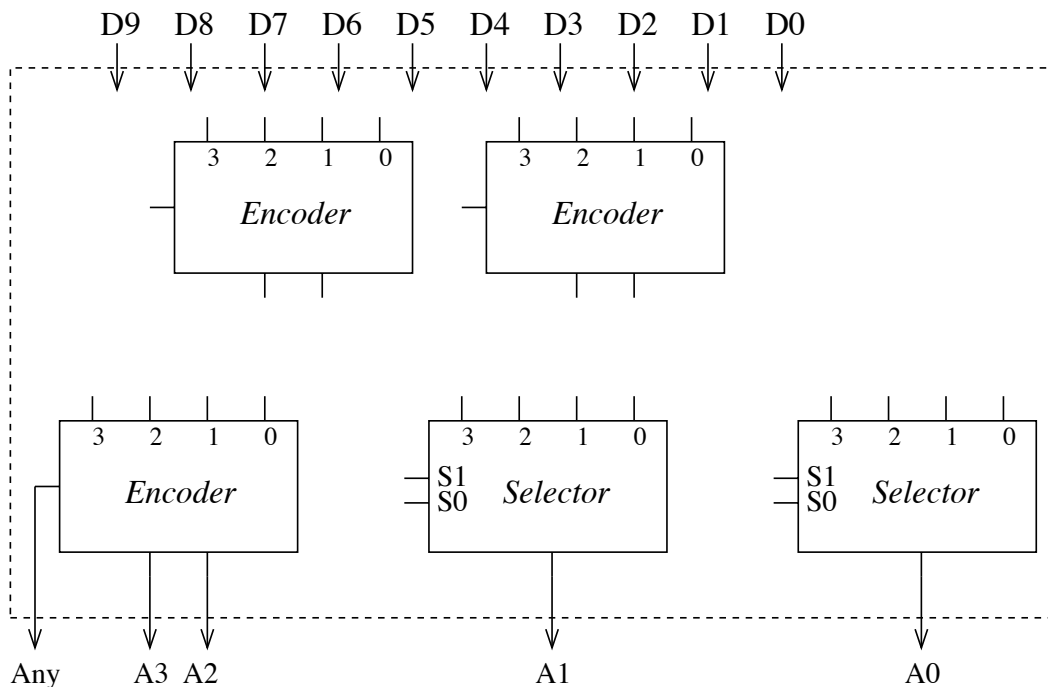
- I:** The number of commonalities induced by the rule is exactly double what is present in a D flip-flop implementation.
- II:** The number of commonalities induced by the rule exceeds double those in the D flip-flop implementation by 1.
- III:** The number of commonalities induced by the rule is less than double those in the D flip-flop implementation by 1.
- IV:** The number of commonalities induced by the rule exceeds double those in the D flip-flop implementation by a fixed, known quantity greater than 1 (but you don't need to report the exact quantity).
- V:** The number of commonalities induced by the rule is less than double those in the D flip-flop implementation by a fixed, known quantity greater than 1 (but you don't need to report the exact quantity).
- VI:** The number of commonalities induced by the rule exceeds double those in the D flip-flop implementation by a quantity that can only be determined on a case-by-case basis.
- VII:** The number of commonalities induced by the rule is less than double those in the D flip-flop implementation by a quantity that can only be determined on a case-by-case basis.

3 (Encoders for Decimal Codes)

We have studied in class the design of primary encoders for binary numbers. In this question, we examine the design of primary encoders for the **Excess-3** and **2421** decimal codes through the use of binary encoders and selectors. (*For those of you who need a refresher, the Excess-3 and 2421 decimal codes are provided at the end of the homework.*)

(Part A) In this part, you are asked to implement a 10-to-4 primary encoder that generates an **Excess-3** coded decimal number, $A_3A_2A_1A_0$, as its output. Specifically, if input line D_i is high and none of the inputs D_k ($k > i$) is high, the output $A_3A_2A_1A_0$ should represent a decimal digit i in Excess-3 representation.

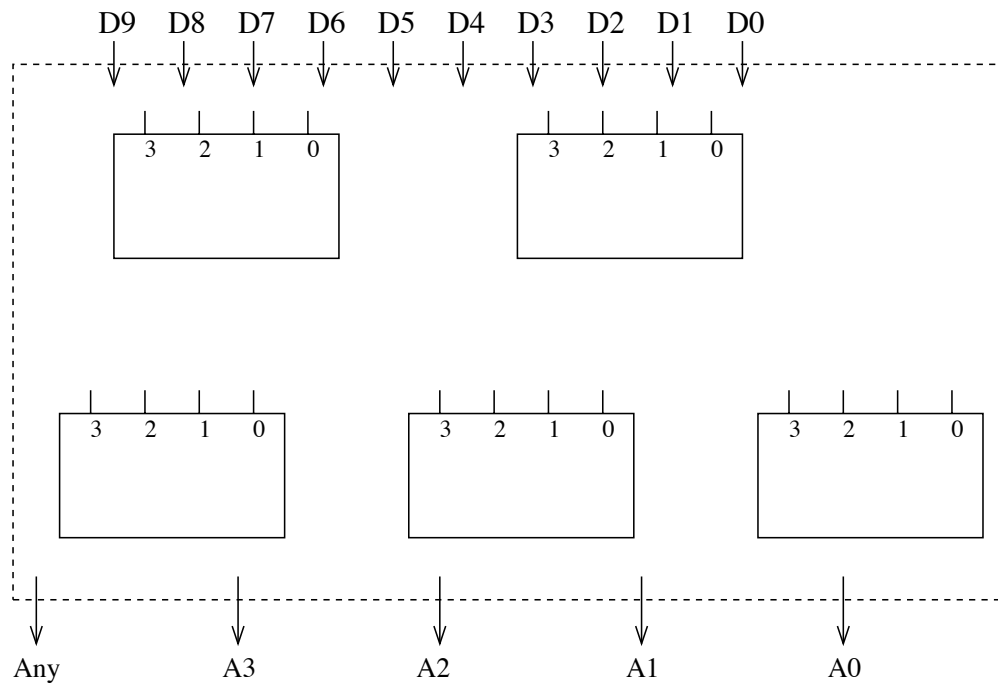
You are asked to design this Excess-3 primary encoder using a number of 4-to-2 binary encoders and 4-to-1 selectors. To simplify your work, the skeleton of the primary encoder is provided below. Please complete this implementation by drawing the connections among the blocks. *Please note that in this implementation, there exist two blocks whose two input lines should be driven by constant values, i.e., 0 or 1.*



Implement an Excess-3 encoder with 4-to-2 encoders and 4-to-1 selectors

(Part B) In this part, you are asked to implement a 10-to-4 primary encoder that generates a **2421** coded decimal number, $A_3A_2A_1A_0$, as its output. Specifically, if input line D_i is high and none of the inputs D_k ($k > i$) is high, the output $A_3A_2A_1A_0$ should represent a decimal digit i in 2421 representation.

You are asked to design this 2421 primary encoder using a number of 4-to-2 binary encoders and 4-to-1 selectors. You are provided with the following encoder skeleton. Yet this time the type of each component is not clearly specified. Your job is to specify the inputs, the outputs, the type of each block as either **Encoder** or **Selector**, and additionally draw the connections among the blocks and the input/output lines so as to implement this encoder. *Please note that in this implementation as well, there exist two blocks whose two input lines should be driven by constant values, i.e., 0 or 1.*



Implement a 2421 encoder with 4-to-2 encoders and 4-to-1 selectors

The Excess-3 and 2421 codes for decimal digits:

Decimal	Excess-3	2421
0	0 0 1 1	0 0 0 0
1	0 1 0 0	0 0 0 1
2	0 1 0 1	0 0 1 0
3	0 1 1 0	0 0 1 1
4	0 1 1 1	0 1 0 0
5	1 0 0 0	1 0 1 1
6	1 0 0 1	1 1 0 0
7	1 0 1 0	1 1 0 1
8	1 0 1 1	1 1 1 0
9	1 1 0 0	1 1 1 1