Design Assignment #1 Report Template (ver 1.0)

Live URL: https://docs.google.com/document/d/18ILMwwcdb6nlfeIKyuxrMokThgd4zXtpOtH_F_ahE4o

Student Name: Le'on Norfleet

Task 1: Who has more 1's?

Joint Submission?

Is this problem a joint submission (Yes/No)? No
If Yes, what is the name of your partner:

Collaboration

List all the students you collaborated with (sought help or provided help). N/A

1. <Name>: <describe what help you gave or received>
2. …

Approach

Briefly describe your design approach, e.g. what is your "algorithm", did you use K-Maps (if so, include here), how did you optimize the design?

My approach was to sum the number of set bits using a chain of adders for A and B. Then I compare both of them for equality using a comparator, looking to see if A is greater than or equal to B's number of bits using an OR gate. This outcome is connected to a multiplexor to both give the output AWON and also select whether A or B will be in DOUT. I was able to optimize the design by being more efficient with my adders and using the best component to reduce the overall amount I needed. For example, I forgot multiplexers were allowed so I implemented a 16-bit OR and 16-bit AND gate to select whether A or B would be the output. This was changed to optimize the circuit.

Task 2: Numeric String to 2's Complement

Joint Submission?

Is this problem a joint submission (Yes/No)? No
If Yes, what is the name of your partner:

Collaboration

List all the students you collaborated with (sought help or provided help). N/A

1. <Name>: <describe what help you gave or received>
2. …

Approach

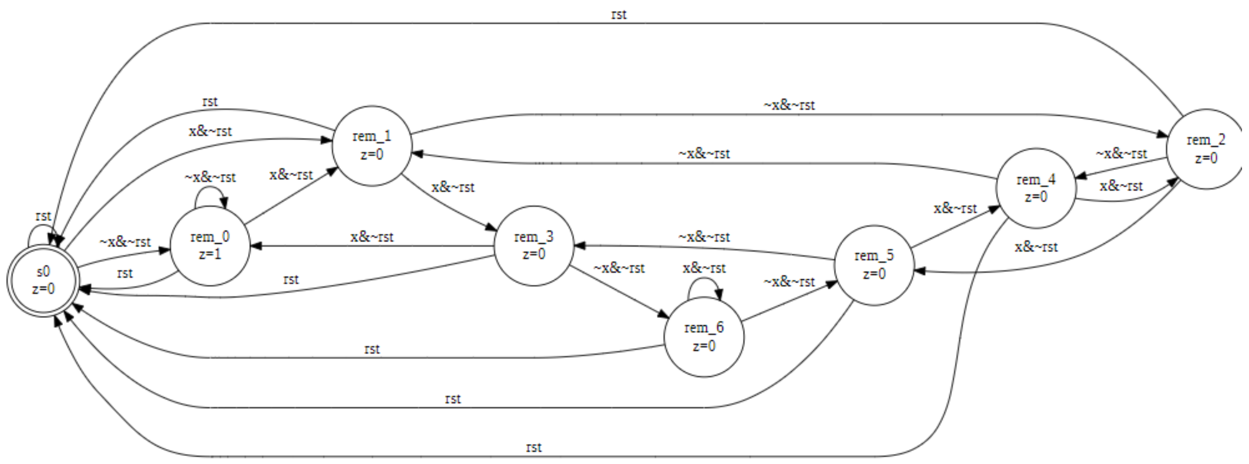
Briefly describe your design approach, e.g. what is your "algorithm", did you use K-Maps (if so, include here), how did you optimize the design?

Check if sign, D1, and D0 are valid(since they are constrained into a specific range of hex values, it is easy to use bit manipulation for this. Multiply D1 by 10 and add it to D0. Use an adder and multiplexors to get the two's complement and see if it should be output, respectively. Use the check if valid circuit to determine NAN, also use it with a multiplexor to select the correct TC output(a constant 0xfff or the number D1D0). I optimized the design by using bit shifting and addition for multiplication instead of implementing a multiplier subcircuit.

Task 3: Detect Multiples of Seven

State Transition Diagram

(download and insert below the PNG Image of the state transition diagram of the FSM that you draw in https//edotor.net using the prescribed template available on Piazza)



I want to mention that reset is implemented synchronously, but not as an input to the truth table

State Transition Table

Provide a state transition table showing the output and next state for each combination of current state and inputs. Add rows as needed. For Current and Next States use the same symbolic names as you did in the State Transition Diagram.

I want to mention, these are the T flip flop conversions for the next state. Meaning that the group of flip flops will actually change to THESE values when executing their behavior reset is implemented synchronously to reduce area cost.

curstate	X	rst	T	Z
000	-	1	000	0
000	0	0	001	0
000	1	0	100	0
001	-	1	001	1
001	0	0	000	1
001	1	0	101	1
100	-	1	100	0
100	0	0	001	0
100	1	0	010	0
101	-	1	101	0

101	0	0	010	0
101	1	0	111	0
110	-	1	110	0
110	0	0	101	0
110	1	0	111	0
111	-	1	111	0
111	0	0	011	0
111	1	0	010	0
010	-	1	010	0
010	0	0	100	0
010	1	0	101	0
011	-	1	011	0
011	0	0	001	0
011	1	0	000	0

State Encoding

Briefly describe how you encoded the states into bit vectors and why you picked this encoding.

<000
001
100
101
110
111
010
011

I chose this encoding because I was looking for one that gave me as minimal next state expressions and karnaugh maps as possible. I did not have a specific reason for this one other than the efficiency it gave me. I went through multiple different encodings until I found this one that worked pretty well.>

Circuit Cost

Total Circuit Cost = <*252*>

Details of Circuit Cost Computation

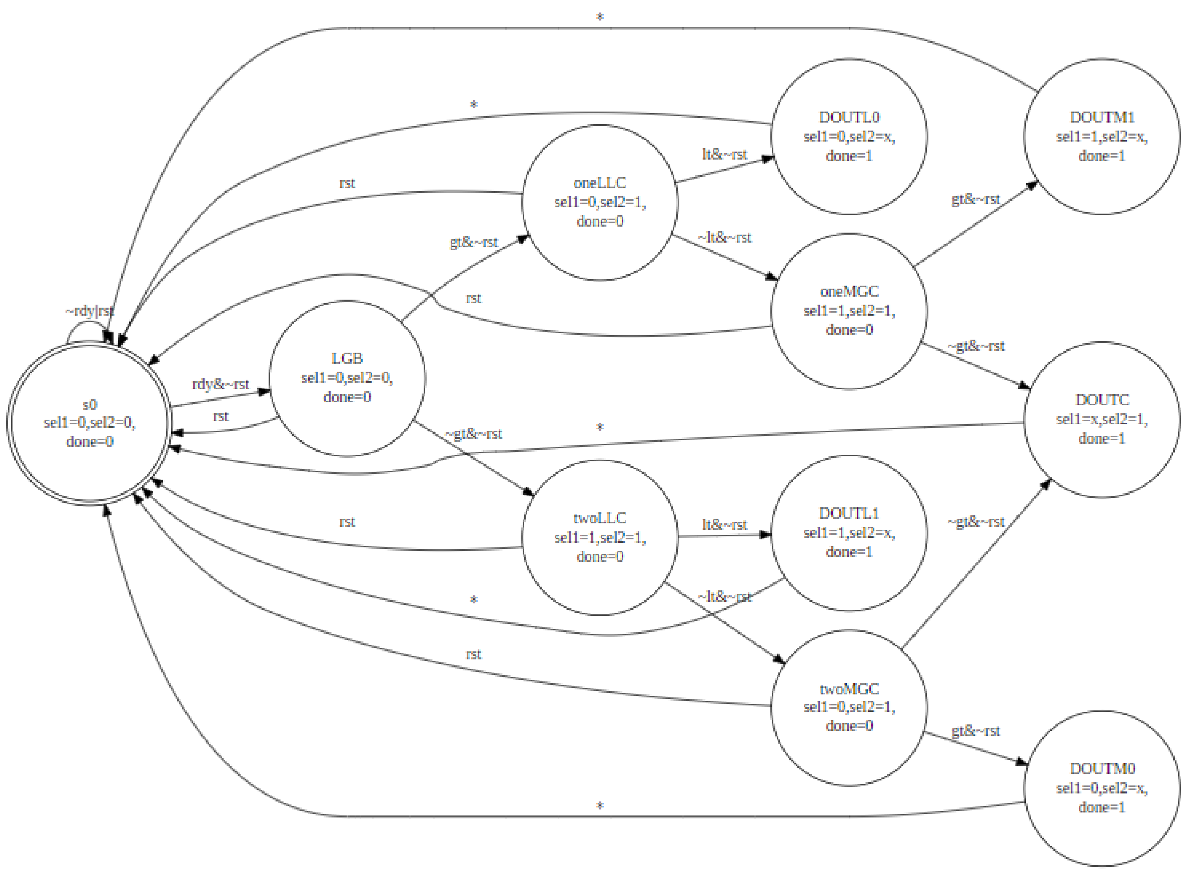
Detail the computation of the cost of your circuit, listing each type of gate, quantity, and cost. Remember to properly account for the cost of any gates in subcircuits.

	T Flip Flop	NOT Gate	NAND Gate	
	3	20 1-bit	13 1-bit 2-input	
			9 1-bit 3-input	
			1 1-bit 5-input	
SUM	32*3	20*2*1	(2*1*2*2)+(2*9*1*3)+(2*1*5*1)	96+40+116=252

Task 4: Compute the Median

State Transition Diagram

(download and insert below the PNG Image of the state transition diagram of the controller FSM that you draw in https//edotor.net using the prescribed template available on Piazza)


s0 sel1=0,sel2=0, done=0
~rdy|rst
rdy&~rst
rst
LGB sel1=0,sel2=0, done=0
gt&~rst
~gt&~rst
oneLLC sel1=0,sel2=1, done=0
twoLLC sel1=1,sel2=1, done=0
lt&~rst
~lt&~rst
DOUTL0 sel1=0,sel2=x, done=1
oneMGC sel1=1,sel2=1, done=0
DOUTL1 sel1=1,sel2=x, done=1
twoMGC sel1=0,sel2=1, done=0
gt&~rst
~gt&~rst
DOUTM1 sel1=1,sel2=x, done=1
DOUTC sel1=x,sel2=1, done=1
DOUTM0 sel1=0,sel2=x, done=1
*
rst

State Transition Table

Provide a state transition table showing the output and next state for each combination of current state and inputs. Add rows as needed. For Current and Next States use the same symbolic names as you did in the State Transition Diagram.

Notice: `-` and `x` mean don't care for inputs and outputs respectively. It is logism's notation not mines
The way I use the GO1,GO2,GO3, they do not directly impact the transition of states unless all three have previously been high for 1 cycle. The history of being high is stored with a register for each GO and does not interact with the circuit until all three registers are high at which case rdy will be 1. So for the following table in the context of state transition: GO1,GO2,GO3 do not matter, and neither does rdy unless rst is low and we are at the start state. Another thing to add, I use 11 out of the 16 possible states. The states not mentioned do not matter, please treat them as having - or x in all of their slots for input and output respectively. Finally, reset is implemented synchronously in a different way to reduce area cost.

Current State	RST	$GO1$	$GO2$	$GO3$	<signals from datapath to controller>			Next State	$DONE$	<signals from controller to datapath>	
					rdy	gt	lt			sel1	sel2

0000	1	-	-	-	-	-	-	0000	0	x	x
0000	0	-	-	-	0	-	-	0000	0	0	0
0000	0	-	-	-	1	-	-	0001	0	0	0
0001	1	-	-	-	-	-	-	0000	0	x	x
0001	0	-	-	-	-	1	-	0010	0	0	0
0001	0	-	-	-	-	0	-	0011	0	0	0
0010	1	-	-	-	-	-	-	0000	0	x	x
0010	0	-	-	-	-	-	1	0100	0	0	1
0010	0	-	-	-	-	-	0	1001	0	0	1
0011	1	-	-	-	-	-	-	0000	0	x	x
0011	0	-	-	-	-	-	1	0110	0	1	1
0011	0	-	-	-	-	-	0	1010	0	1	1
0100	-	-	-	-	-	-	-	0000	1	0	x
0101	-	-	-	-	-	-	-	0000	1	0	x
0110	-	-	-	-	-	-	-	0000	1	1	x
0111	-	-	-	-	-	-	-	0000	1	1	x
1000	-	-	-	-	-	-	-	0000	1	x	1
1001	1	-	-	-	-	-	-	0000	0	x	x

1001	0	-	-	-	-	1	-	0111	0	1	0
1001	0	-	-	-	-	0	-	1000	0	1	0
1010	1	-	-	-	-	-	-	0000	0	x	x
1010	0	-	-	-	-	1	-	0101	0	0	0
1010	0	-	-	-	-	0	-	1000	0	0	0

State Encoding

Briefly describe how you encoded the states into bit vectors and why you picked this encoding.

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010

11 states, 2^4 = 16 so it makes sense. They are bit vectors simply by consequence of logism’s table conversion + easier inputs and efficiency. They indexes of the vectors will be split into individual bits anyways so there is no point in partitioning them on a table. I picked this encoding because gray code was the simplest. There were probably more efficient ones but I was able to use a low amount of NAND gates with this encoding, which is why I picked it.

Circuit Cost

Total Circuit Cost = *2402*

Design Approach

Give a brief description of your controller and datapath design approach, including how you split between the two, what is the information exchange between them, and how are the two coordinated.

The design starts with the datapath. Using three 1-bit registers and some combinational logic, I was able to create a ‘group ready’ system where the controller won’t begin calculating the median until all three DIN values have been loaded/all three GO signals have been asserted. When this is done, a rdy signal is sent to the controller to begin calculating the median. The system ‘keeps track’ of 2 ‘variables’, L and M which indicate largest and median. Both are initially set to DIN1 for convenience. The circuit has branching paths similar to if/else statements based on the outcomes of the comparator logic. When a comparison is completed, the datapath sends two signals, gt and lt, which determine the navigation of the controller’s various states. The numbers chosen for the comparison is dictated by the controller and changes based on the state it is in, meaning that it may do DIN1 > DIN2 one cycle, and then do DIN2 < DIN3 another cycle. When the controller computes the median, it sends outputs a DONE signal for 1 clock cycle and also sends it to the datapath, so it can clear the ‘group ready’ system and be ready for the next calculation in its idle state. L and M change based on the outcome of the comparison of the N-1th clock cycle, unless the system has been reset, in which case it will resort to the default starting comparison of DIN1 and DIN2.

Performance Summary of Your Design

How many clock cycles does your design take in the worst case to produce the Median from the point the latest of the three inputs is available? Is this delay dependent on the values? If so, what is the best case delay? When do the best and worst-case delays occur?

It takes 4 clock cycles to produce the Median in the worst case. The delay is not dependent on the variables. The median algorithm I created does 3 comparisons overall, meaning the delay is based on the inputs in which the numbers used for the median are placed. The best-case delay is 3 clock cycles. Best case delay occurs when DIN1 > DIN2, DIN2 < DIN3 and DIN1 < DIN2,DIN2 < DIN3. The worst-case delay occurs in all other scenarios.