Serialized Comparator

In this problem, you will be walked through the thought process of serialization, in order to better understand serialization and comparators, both of which can be a tricky topic for students to grasp. Suppose we want to compare two binary numbers A and B. Our designed comparator should output a 1 if A $\leq$ B, and 0 otherwise. To start with serialization we first need to review some bit slice design, which is the idea of performing repeated operation on a single bit of A and B to recover some information. We will denote the comparator output as Y.

1. Suppose we have 1-bit inputs A and B to a comparator, fill in the table based on what the output should be. Ask yourself, if the inputs a and b fill the comparator condition of A $\leq$ B

A	B	Y(A $\leq$ B)
0	0	
0	1	
1	0	
1	1	

2. Next, lets recognize some patterns by comparing larger input sequences. You could convert A and B to binary to compare if necessary.

A	B	Y(A $\leq$ B)
11	11	
11	01	
111	101	
1000	0111	

Serialized Comparator

3. The next question we must ask ourselves, is whether we can design some logic that takes the current bit information and the information provided by previous bits, to construct some new output. Suppose we are looking at one bit at a time starting from the LSB, just like part 1. Based on a single bit we can make some conclusion about a numbers comparative magnitude. Next, we can look at the next LSB(the second bit from the left). Can we use information obtained from the magnitude of the first bit to make a conclusion about the magnitude about the second bit? Think about how binary numbers are sums of powers of 2, and what this could imply. The previous Y is available to you as Y_{i-1}. Fill out the table on what the comparative outputs of some bit slice of A and B should be based on their current bits A_i and B_i.

To get started you may want to do some examples. Start with some 2 bit input and pretend you are trying to determine Y for the MSB after obtaining Y from the LSB.

A_i	B_i	Y(A $\leq$ B)
0	0	
0	1	
1	0	
1	1	

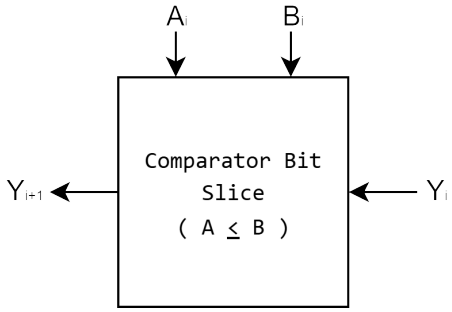
Serialized Comparator

4. Now create output expression for Y based on Y_{i-1}, A_i ,and B_i. Creating a k-map may be necessary.

Y=

5. Now that we have an output for Y based on some expression, draw the digital logic implementation of Y. You may use Y_{i-1}, A_i ,and B_i as inputs.

6a. Now we can place Y inside a box, called a unit bit slice. This bit slice represent a single bit slice of our comparator logic, in which inputs and outputs can be chained together. Explain what is inside the unit bit slice and how this system will produce an output Y based on your previous calculations.



Serialized Comparator

6b. What should the default carry in be for the first bit slice? What should we assume of the comparative magnitudes of numbers that have not necessarily been processed yet?

7a. Suppose we want to compare two 8 bit numbers. How many unit bit slices would we then need?

7b. Suppose we want to compare two 64 bit numbers. How many unit bit slices would we then need?

7c. As the number of bits increases, what happens to the number of bit slices required?

7d. Based on these observations, what inefficiency do you see potentially arising with this bit sliced design. Think about the re-usability of the system with different sized numbers as well as the total area it takes up.

7e. Does a single bit slice rely on data from the previous bit slice?

7f. Have we learned about any computer components that can store bits of data? If so name them.

.

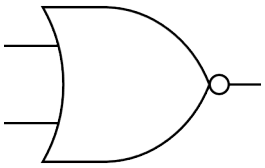
Serialized Comparator

Serialized designs allow us to take a bit slice and reuse it over and over again for each individual bit, rather than have a large array of slices. Based on problem 6, the output of one bit slice feeds into the input of the next. This means that the systems output is defined by the previous state, which can be seen from your expression from question 5. What if we saved this bit in some sort of storage device, then fed in the next set of bits for A and B (the next slice) and then used the saved value of Y from the previous state as part of our calculation. This trick allows use a singular bit slice repeatedly to perform the comparison on an entire bit sequence, regardless of how long it is.

9a. Look at how many bits of data we need to store between bit slices, or how many bits of data are sent from one bit slice to the next?

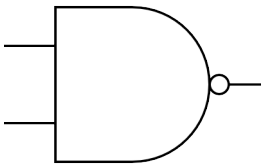
9b. How many flip-flop will be needed in the serialized design? What needs to be stored?

9c. We decided in part 6b, that the system needs a constant carry in for the first bit. However, we need the bit slice to accept the input Y_{i-1}. Suppose you are given an input signal F, which equals one when the current bit slice is the first bit, and 0 otherwise. Fill in the following circuits by specifying the inputs such that the listed input is forced when F=1, and Y_{i-1} is forced when F=0. Complemented inputs are available. The two circuit provide a different input depending on what the input F is. Indicate which should be used based on your answer from problem 6b.



Initialize to 0 when F=1

Otherwise pass Y_{i-1}



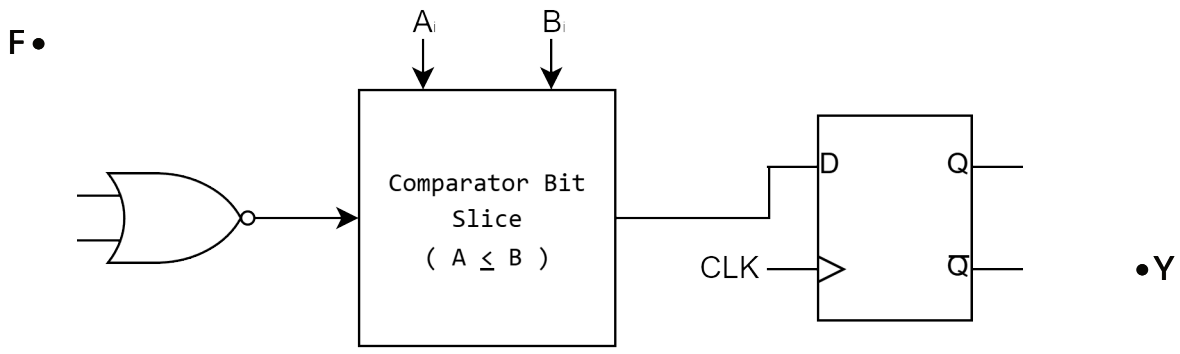
Initialize to 1 when F=1

Otherwise pass Y_{i-1}

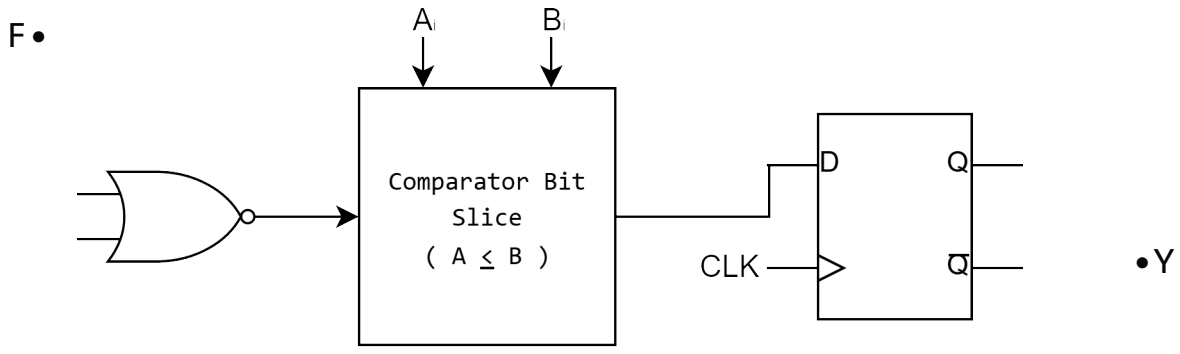
Serialized Comparator

10a. We now have all of the necessary parts to make the serialized comparator. Fill in the following diagram to complete the design. Choose the relevant bit carry-in enforcing mechanism from the previous problem. Feed in any relevant inputs, and ensure all components are well defined. You may select from the list of following components, though not everything may be used. All provided inputs and outputs are to be used.

NOR(carry in enforcement) , Bit Slice, Flip Flop(Stage Storage)



10b. Suppose we want to modify the comparator such that it outputs a 1 when A > B. How can we modify the existing circuit to do this. You may not add any additional gates to your solution from part 10a. Think about the relation between A > B and A $\leq$ B.



Increasing Monotonic Sequence Checker

For this problem, you will be designing a system that checks whether a sequence of numbers is an increasing monotonic sequence. This means that numbers can only increase in the sequence, but never decrease. You will start checking from the MSB of this number, and end at the LSB. You will be given an input number A and must calculate an output Y that determines whether A is a monotonic sequence.

VALID Increasing Monotonic Sequence: 0 0 0 1 1 1 1 1

INVALID Increasing Monotonic Sequence: 0 1 0 0 0 1 1 0

1a. Suppose we want to check whether a sequence of numbers is strictly increasing. For any given bit, do we need to know what the previous bit was?

1b. Given your answer above fill out the following table. You have the inputs A_{i-1} and A_i available to you as inputs as well as Y_{i+1}. Use these to determine if the sequence is then increasing or decreasing.

Y_{i+1}	A_i	A_{i-1}	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Increasing Monotonic Sequence Checker

2a. Using your truth table from 1a give a boolean expression for Y

2b. Draw the digital logic implementation of Y using a 3:8 decoder, and a single OR gate of any bit width.

Increasing Monotonic Sequence Checker

3. Now we can place Y inside a box, called a unit bit slice. This bit slice represent a single bit slice of our sequence checker logic, in which inputs and outputs can be chained together. Explain what is inside the unit bit slice and how this system will produce an output Y based on your previous calculations.

3b. What should the default carry in be for the first bit slice? What should we assume of a sequence of numbers that have not necessarily been processed yet?

4a. Suppose we want to check a 12 bit numbers. How many unit bit slices would we then need?

4b. Suppose we want to check a 128 bit numbers. How many unit bit slices would we then need?

4c. As the number of bits increases, what happens to the number of bit slices required?

4d. Based on these observations, what inefficiency do you see potentially arising with this bit sliced design. Think about the re-usability of the system with different sized numbers as well as the total area it takes up.

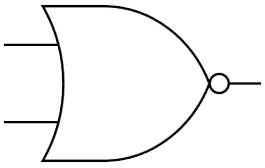
4e. Does a single bit slice rely on data from the previous bit slice?

Increasing Monotonic Sequence Checker

5a. Look at how many bits of data we need to store between bit slices, or how many bits of data are sent from one bit slice to the next?

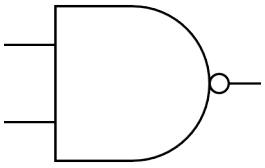
5b. How many flip-flop will be needed in the serialized design? What needs to be stored?

5c. We decided in part 3b, that the system needs a constant carry in for the first bit. However, we need the bit slice to accept the input Y_{i+1}. Suppose you are given an input signal F, which equals one when the current bit slice is the first bit, and 0 otherwise. Fill in the following circuits by specifying the inputs such that the listed input is forced when F=1, and Y_{i+1} is forced when F=0. Complemented inputs are available. The two circuit provide a different input depending on what the input F is. Indicate which should be used based on your answer from problem 3b.



Initialize to 0 when F=1

Otherwise pass Y_{i+1}



Initialize to 1 when F=1

Otherwise pass Y_{i+1}

Serialized Increasing Monotonic

10a. We now have all of the necessary parts to make the serialized comparator. Fill in the following diagram to complete the design. Choose the relevant bit carry-in enforcing mechanism from the previous problem. Feed in any relevant inputs, and ensure all components are well defined. You may select from the list of following components, though not everything may be used. All provided inputs and outputs are to be used.

NAND(carry in enforcement) , Bit Slice, Flip Flop(Stage Storage)

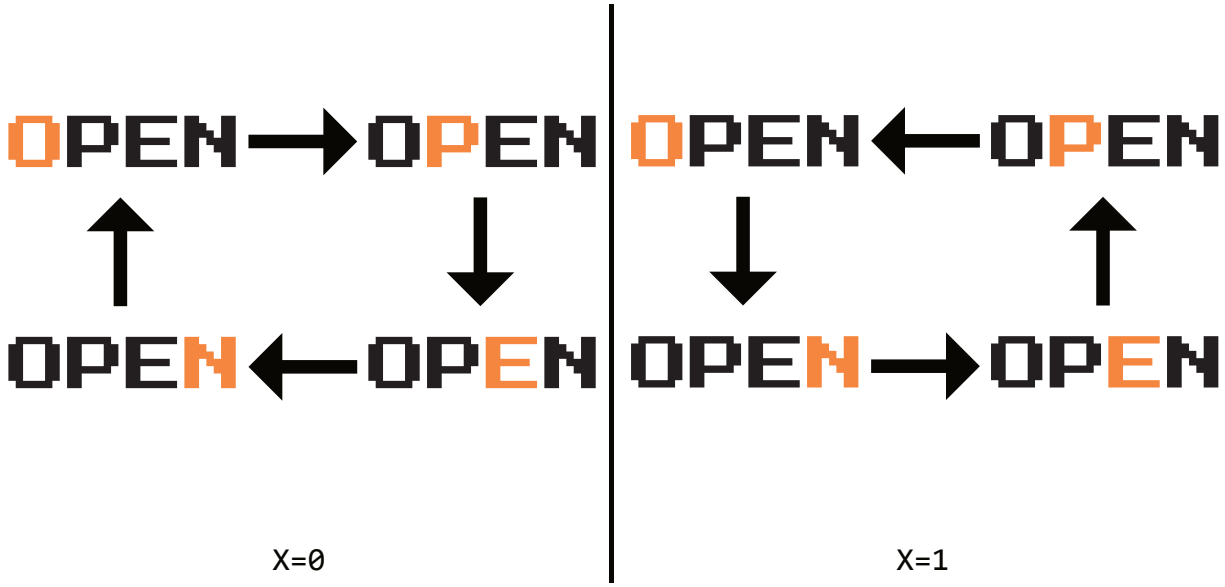
LED OPEN Sign FSM Analysis

The Daily Byte wants to ad an OPEN Sign to attract more customers They have designed a system to create an LED sign where each individual letter is lit up in a cycle. The effect is the ‘O’ is lit up followed, by the ‘P’, followed by the ‘E’, followed by the ‘N’, with it looping back to the ‘O’ after. The staff have included the lighting system below. The lighting for the sign can be represented as a 4 bit sequence [OPEN], where a 1 means that letter is lit and a 0 means the letter is not.

You are tasked with extending this FSM to create a different flashing LED sign. You are given an input X, which switches the FSM from a cycling LED to a flashing LED. When X=0, you should continue to cycle the LED as above. When X=1, you should then flash the entire sequence on and OFF. A visualization and outline for the LED operations have been outlined below.

1. Fill in the output operation for the extended system.

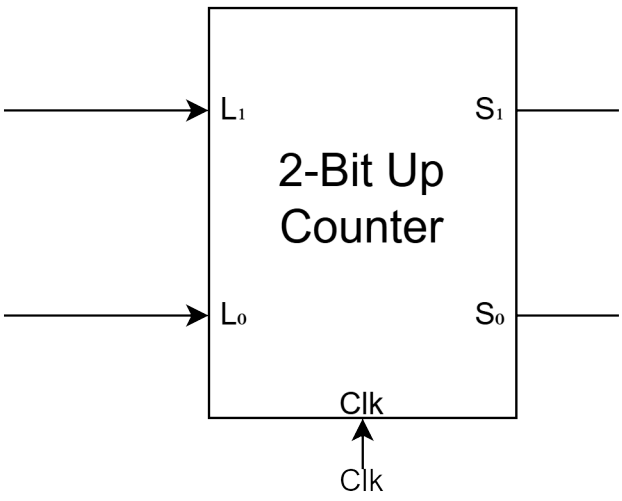
State(S1S0)	Output[OPEN] X=0	Output[OPEN] X=1
00	1000	1000
01	0100	
10	0010	
11	0001	



LED OPEN Sign FSM Analysis

2. Given X=0, derive expressions for the outputs O,P,E, and N from inputs S1 and S0.

3. The original FSM uses a 2 bit-up counter, which resets to some load input after exceeding its maximum value. Use you derived logic above to complete the non-extended FSM (X=0). You may use any amount of 4-input ANDs to implement the output logic.

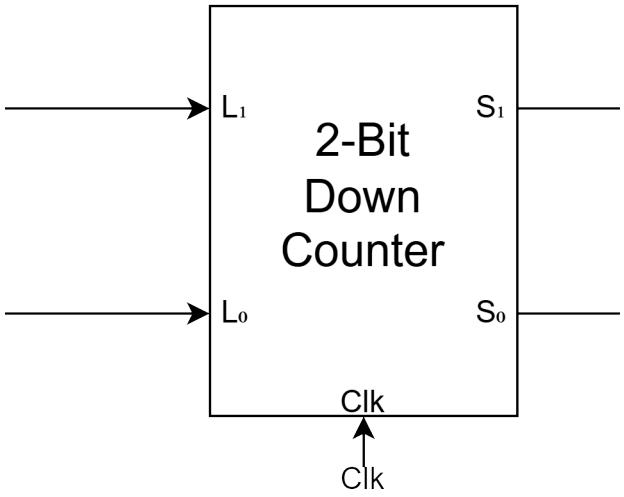


- **O**
- **P**
- **E**
- **N**

LED OPEN Sign FSM Analysis

4. Now you must design the extended FSM. You are given a 2-bit down counter, which resets to some load input after going below the minimum value. Your logic for this FSM should use the same output logic for O,P,E,N as part 3.

- **O**
- **P**
- **E**
- **N**



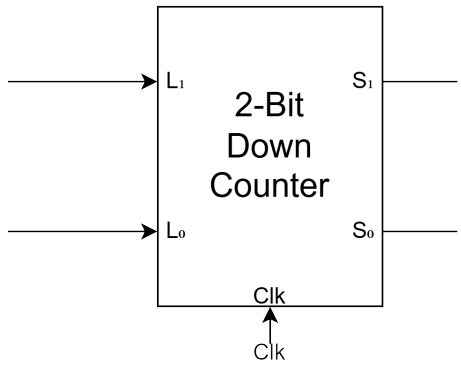
You now have the two distinct FSM for the two cycling cases, when X=0, and X=1. You must now combine the FSMs by selectively combining the FSMs based on the desired output.

LED OPEN Sign FSM Analysis

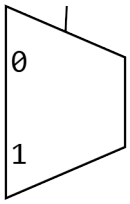
5. Complete the full FSM by filling in inputs and implementing the necessary output logic.

X•

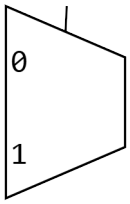
•O



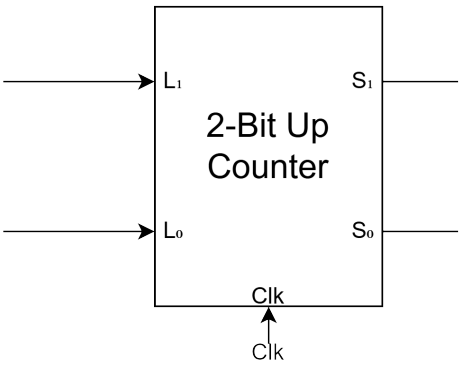
•P



•E

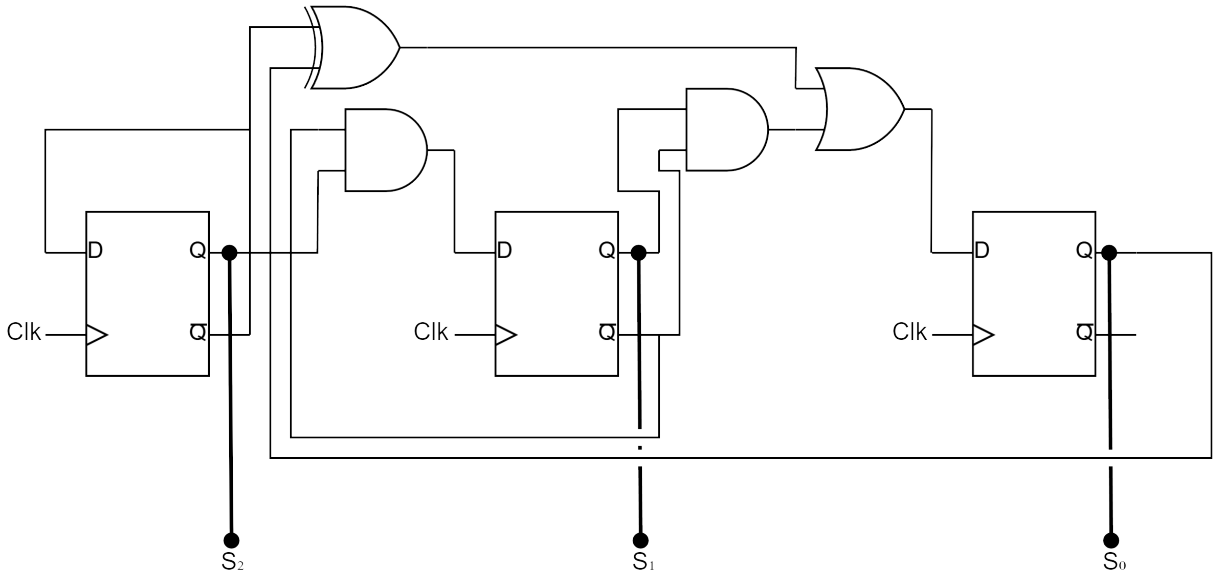


•N



Random FSM Analysis

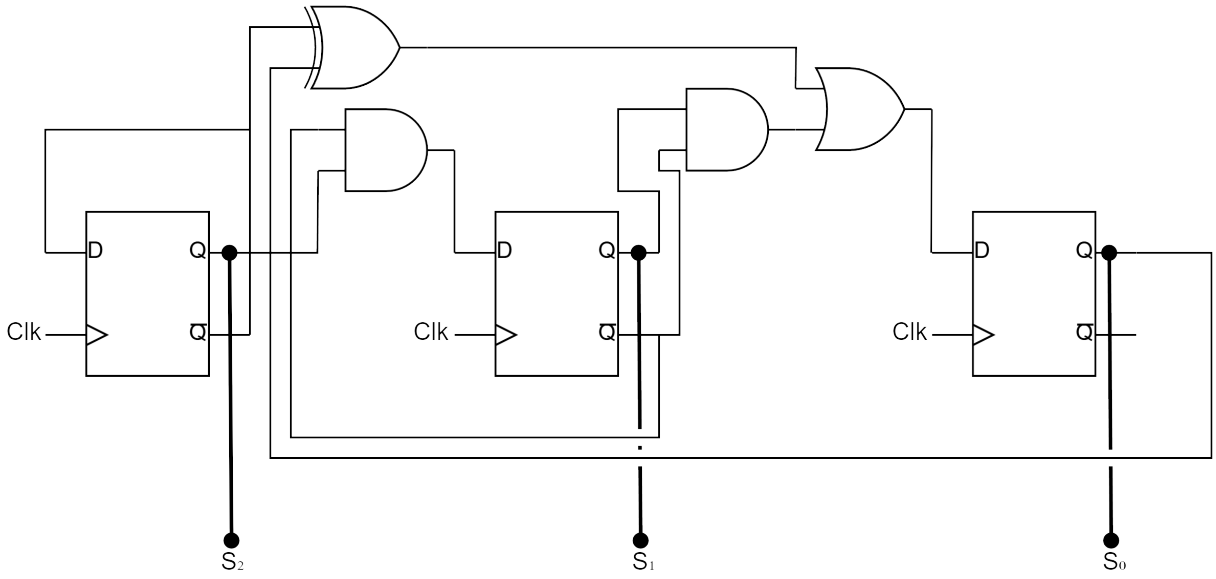
You are given the following FSM. You are asked to analyze various components.



1. Find the next state expressions for S2, S1, and S0.

Random FSM Analysis

You are given the following FSM. You are asked to analyze various components.



2. Fill out the following table

S2	S1	S0	S2+	S1+	S0+
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Random FSM Analysis

3. Draw the FSM diagram for this system. You can ignore the system output in labeling.

4. Suppose the system is in some unknown state to start. Indicate whether the system can end up in a particular state after a certain number of clock cycles.

After 0 Clock Cycles:

000 ____ 001 ____ 010 _____ 011 ____ 100 ____ 101 ____ 110 ____ 111 ____

After 1 Clock Cycle:

000 ____ 001 ____ 010 _____ 011 ____ 100 ____ 101 ____ 110 ____ 111 ____

After 2 Clock Cycles:

000 ____ 001 ____ 010 _____ 011 ____ 100 ____ 101 ____ 110 ____ 111 ____

After 3 Clock Cycles:

000 ____ 001 ____ 010 _____ 011 ____ 100 ____ 101 ____ 110 ____ 111 ____

Divisible-By-2 FSM

A system receives a serial stream of binary digits (one bit per clock cycle) and needs to determine whether the number formed by the bits read so far is divisible by 2. The FSM should continuously process incoming bits and indicate divisibility at each step. The bits will be sent starting from the MSB and ending with the LSB. This problem will walk you through all of the step that comes with creating a FSM, which can be applied to many different problem types. Some questions may seem obvious, but they l walk you through the proper thought process for problem solving.

1.Rewrite/Identify what conditions should this FSM output a 1.

2. How can we tell whether a binary number is divisible by 2? Start by doing some examples and try to identify a pattern, then write out in words how you can identify whether a binary number is divisible by 2.

0110: (Even / Odd)

0111: (Even / Odd)

01: (Even / Odd)

0000: (Even / Odd)

1001: (Even / Odd)

100: (Even / Odd)

3. Now that you know what makes a binary number divisible by 2, identify what states you might need to track along the way. Recognize that the FSM should continuously process incoming bits and indicate divisibility at each step. Thus, as a new bit comes in, you can assume that bit is the new LSB, and then process the number accordingly. How many states might you need to track whether a number is even or odd? List them out below, assigning them numbers and meanings.

State	Meaning	Output

Divisible-By-2 FSM

4. Now that you have identified your states, draw an Moore Machine state transition diagram. Remember, you will need one bubble for each state. Moore Machine diagrams label bubble using STATE/OUTPUT notation, so make sure to including what the output should be at each state based on what you defined in the previous problem.

5. Create a next state transition table based on your FSM diagram. This table should define what your next state should be given your current state and some input..

Input(A)	S0	S0+	Output(Y)
0	0		
1	1		
0	0		
1	1		

Divisible-By-2 FSM

6. Use you next state table to derive a next state expression for S1+. You have one input A, which is the input bit for the number sequence, and one output bit Y, which dictates whether the number detected so far is even. Then implement your design using digital control logic. Remember, you will need a D Flip-Flop for every bit you want to store. Ask yourself: what needs to be stored vs. what can I calculate only using my current state?

7. Notice that there is no reliance on previous states to derive an output. Explain how this system represents serialized design. What logic is going on in a FSM unit? What is serialization and how is it being used in this system?

Divisible-By-2 FSM

8. Given our expression from question 5, lets draw the output of the system given the inputs. Fill in the diagram, with the necessary control logic.

Serial Input[A] •

• Output(Even/Odd)

S1 •

9. Check through your design using your examples from before. As you feed this into your design from MSB to LSB, does your design accurately output whether the currently known number is even or odd?

0110: (Even / Odd)

0111: (Even / Odd)

01: (Even / Odd)

0000: (Even / Odd)

1001: (Even / Odd)

100: (Even / Odd)

3-Bit Up/Down Counter

For this problem you will design a 3-bit Up/Down Counter. Counters are able to count numbers in a predefined order. Your design should simply count a 3-bit number from 000 to 111 in ascending order of magnitude (0-7). Your design takes a single input C. When C=1, your design should count upwards. When C=0. You design should count downwards. The design should wrap around, which means that if you count up past 111, the system will go to 000, and if you count down past 000, your system will got to 111.

1a. How many states will you need for the counter design?

1b.How many numbers will you need to count?

1c. How many bits will you need to represent the system state?

2. Draw the state transition diagram for this counter. Label all transitions with respect to the input C. There is no relevant output for this system, and the output can be omitted from the diagram. Try to draw the diagram without making a table.

3-Bit Up/Down Counter

3. Modify the FSM to include a synchronous reset R. That is, when R = 1, the counter should reset to 000 regardless of C. When R = 0, it should function normally as an up/down counter. Your system input can now be represented as a two bit input RC, and you diagram should be labeled accordingly.

Pattern Detector

In this problem, you will design a pattern detector, which detects the pattern ‘1011’ in some serialized input. The bits will be fed as a serialized input starting from the LSB and ending at the MSB. Each time the pattern is detected, the system should output a ‘1’. The output can be denoted as Y and the input bit of the current serialized input can be denoted as X.

1. Draw a state transition diagram for this system. Ensure inputs are well-defined for each state and be careful with handling cases with overlapping occurrences. Determine how many states you need, fill out the state-meaning table, and draw a diagram. You should use the minimum number of states. Only use as many rows as needed in the following table.

State	Meaning	Output

2. Given the following serialized input, indicate the output of the FSM along the way.

X = 1 0 1 1 0 1 1 0 0 0 1 0 1 0 0 1 0 1 1 0 1 1

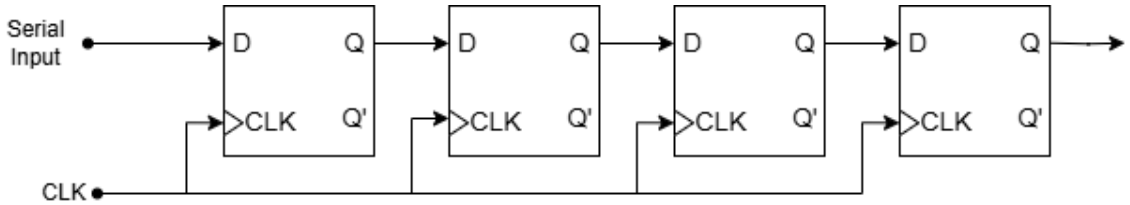
Z =

Pattern Detector

3a. The input sequence will be provided as a serialized input. This can be through a shift register, in which the LSB will be shifted in to start, followed by next least significant bit, ending with the most significant bit. Notice, that bits will start getting thrown out after they trickle through the registers and new bits are shifted in. Use a single gate of any input-width to represent the output Y from these shift registers.

•

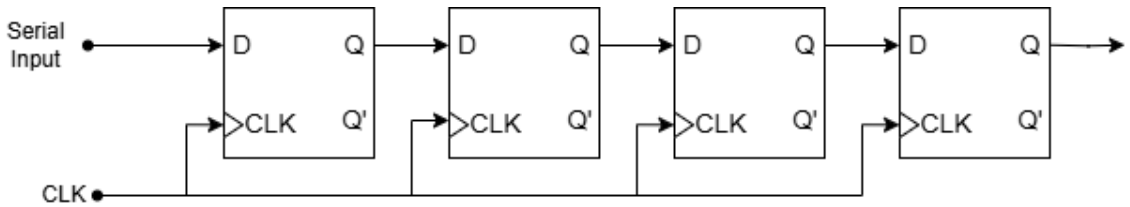
Y



3b. You are given the same set of shift registers. Represent the output using a single 4:16 Decoder. You can simplify the decoder by only labeling the used terminals, but indicate the input and output terminal indices for the ones you choose to use.

•

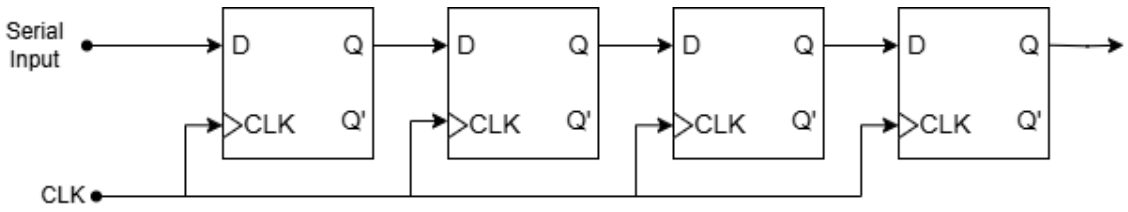
Y



Pattern Detector

3c. You are given the same set of shift registers. Represent the output using a single 16:1 MUX. You can simplify the MUX by only labeling the used terminals, but indicate the select and input terminal indices for the ones you choose to use.

Y

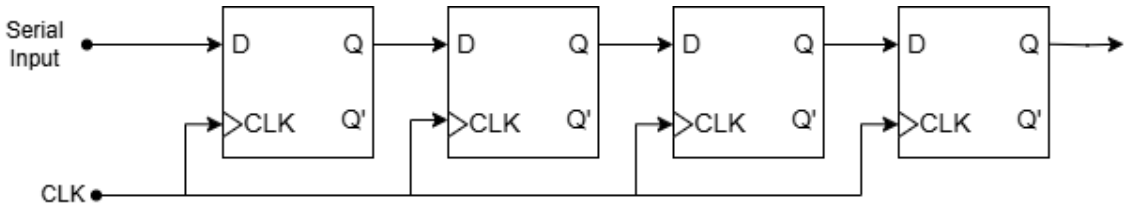


3d. Suppose our input pattern is setup such that it can only create input as chunks of ‘1111’ or ‘1011’. Represent Y using only the given flip flops. You may not use any additional gates.

Valid Sequence: 1111 1011 1111 1011 1011 1111 1011 1111

Invalid Sequence: 0000 0000 0101 0111 1011 1111 1000 1010

Y



Lost at Sea

You and your friends are stranded at sea in a small motorized boat after a storm knocked out your automatic control system. You discover that the boat's functionality can be recovered by creating a control system to steer the boat. Seeing as you only care about getting home, and not necessarily fixing the boat, you decide that you can confidently get home if you can get the boat to at at least go forward, go left, go right, and sit idle. When switching directions from left to right, you must set the boat to idle in between to prevent the boat from tipping. Your system thus has four input and can be denoted by some state U_1U_0. A input of U_1U_0 should attempt to drive the system towards the state S_1S_0. The system is controlled by two motors. The engine propels the boat forward goes idle according to output E. The Rudder steers the boat right, left, or straight, according to output R1R0 . If R1R0 is [00], the boat should go straight. The left or right bit of this two bit sequence should then be set depending on if the boat is going left or right (i.e. left is [10]). If the boat is idle, the output should be set such that the boat is steering straight.

1. Fill out the following state table to define the system's inputs and outputs. Items that are not specifically defined can be chose by you.

State(S_1S_0)	Meaning	E	R_1R_0
00	Idle		
01	Right		
10	Left		
11	Forward		

2. Create a state diagram using the states you defined above. Use the input U1U0 as the state transitions. Use the 3 bit number ER1R0 as an output for the system. The input states are set to have the same meanings as the system states you defined above. That is, whatever meaning is defined by state [00] the input U1U0 should drive the system to this state.

Lost at Sea

1. Derive the next state expressions S1+ and S0+ for the system.

2. Derive the output expressions for E, R1, and R0.

3. Draw the digital logic implementation for the entire FSM.

Automatic Bird Feeder

1. You are tasked with designing an automatic bird feeder system. The system is supposed to feed birds, and only, birds during the daytime. To do this, the system is equipped with a bird sensor, which detects if a bird arrives, and a daylight sensor, which detects when it is daytime. The bird sensor will give a positive signal when a bird is detected and a negative signal otherwise, available to you as the input B. The daylight sensor will give a positive signal when it is daytime and a negative signal otherwise, available to you as the input D. The system has output F, which dictates whether feed should be dispensed or not. The input pair is available as a two bit number [BD]. When the feeder detects a bird, it should dispense feed for 2 clock cycles, before stopping. Once the feeder dispenses feed for the day, it should not dispense any more feed until the following day.

1. Fill out a meaning table and draw a state transition diagram to represent this system. You only need 4 states to complete this problem.

State(S_1S_0)	Meaning	Output(F)
00		
10		
01		
11		

Automatic Bird Feeder

2. Derive the next state expressions S1+ and S0+ for the system.

3. Derive an expression for the output F

4. Draw the digital logic implementation of this FSM.

Automatic Bird Feeder

5. Your feeder becomes popular among the birds, and you start to feel guilty only feeding them once per day. You add an override switch which bypasses the control logic that ensures the feeder only feeds the birds once per day. This input is available as V, and is 1 when you want to override the delay, and 0 otherwise. The system should still wait for a bird to arrive before dispensing feed, but it should now allow you to override the recharging period. Modify your implementation from part 4 by using two additional logic gates and the new input V.