LAB 5: SYNCHRONOUS SEQUENTIAL CIRCUITS

This lab will focus on the implementation and uses of memory elements (in this case, D Flip-Flops) and combinational circuits, and the design of synchronous sequential circuits for a variety of purposes in the context of a sleepy seaside resort town.

On that note, congratulations! You've saved enough for a plane ticket, packed up, and finally got away from your ECE 465 demons. Unfortunately, the town isn't big, and everyone's clocked you out to have been an engineering graduate, so everyone's coming to you to solve their electric engineering problems. They're your family, too, so you can't hit them with the 'I tender my resignation, effective immediately'...so get to it! Mang Tomas' Electronics Shop will give you access to a good number of logic gates (AND, OR, NOT, XOR, XNOR), and D Flip-Flops, as well as digital oscilloscopes that can serve as Clock inputs.

Mang Tomas is great, but he's also really temperamental and eccentric, *so the availability of components will vary from task to task*.

As in previous labs, use Logisim-evolution when instructed to implement or design a circuit.

Part 1: Tita Josie's Sari-Sari Store

Your Tita Josie will be gone for the next few weeks, and has thus left the care of her store to your cousin, Kuya Jerome. Unfortunately for you, your Kuya Jerome has been watching a few too many business and technology TikToks and has badgered you into automating the store. Against your better judgment, you finally give in, and decide to show him how to implement a **barcode reader**.

You will have to start with a prototype, that can recognize a barcode for the store's most popular product—a shampoo sachet.

For your input, you will have a scanner that reads from left to right, and returns 1 when a black stripe is detected, and 0 when a black stripe is not detected. After doing some digging, you also find that every barcode for the sachets has at least one black stripe, followed by a blank space, then followed by one black stripe. The output must be able to recognize this sequence and returns 1 when it is detected. Every barcode provides an input sequence of length 5. ***Design and implement a synchronous sequential circuit that achieves this purpose.***

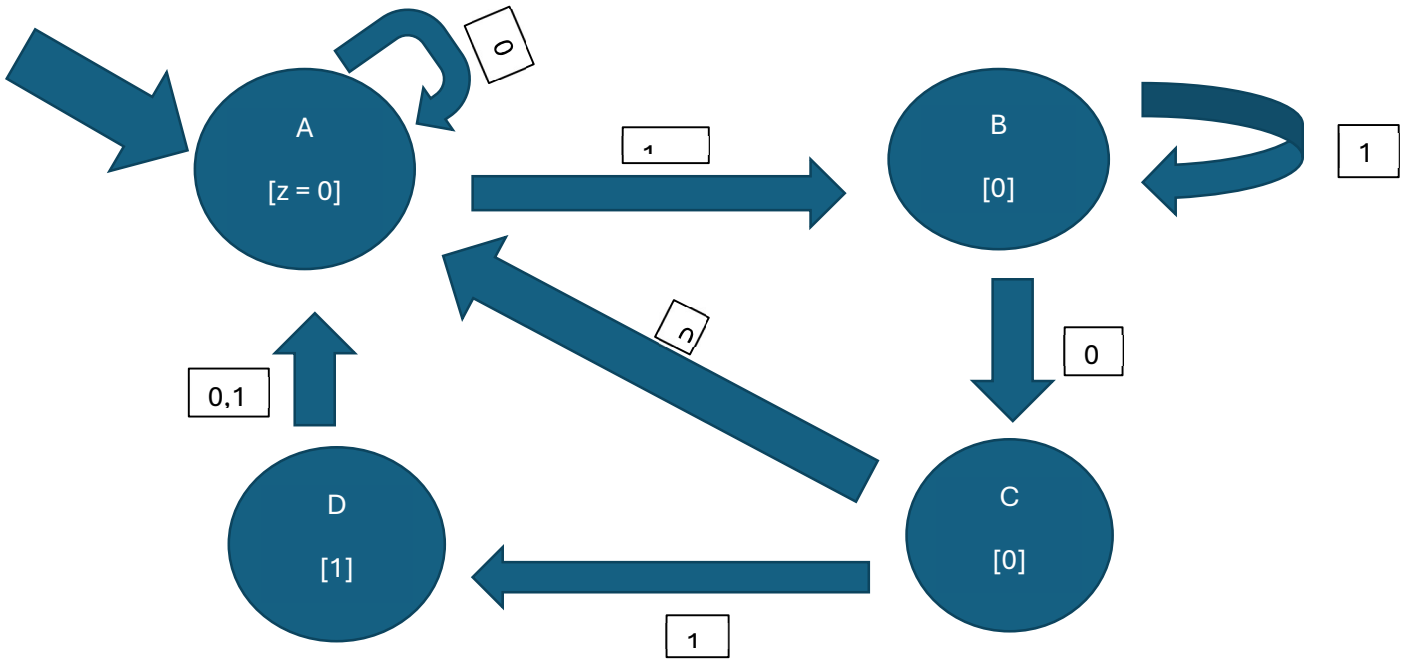
Unfortunately, the soap opera that Mang Tomas has been watching had a bad ending, and so has decided to shut off much of his shop. You will therefore only have access to NAND gates and D Flip Flops for the implementation.

SOLUTION:

Four states will be needed:

A	An initial/reset state indicating a blank memory; output returns 0
B	Indicating the first "1" from the "101" sequence has been detected; output still returns 0
C	Now "10" has been detected; output still returns 0
D	"101" has been detected—output returns 1 to indicate the product is detected, and can reset to get ready for next scan

With this info, a state diagram can be drawn (edges marked by x-values that cause transitions displayed by edges; values in brackets represent the outputs associated with each state):



With four states needed, two flip flops will be needed, each returning an output y_1 and y_2 respectively, so the states are encoded as follows:

State	y_1y_2
A (initial)	00
B	01
C	10
D	11

Using the state diagram, we get the state transition table (now, instead of labeling each state with its letter, we will label them by their associated y_1y_2 configuration):

Present State (y_1y_2)	Next State/Output (y_1y_2/z)	Next State/Output (y_1y_2/z)
	X=0	X=1
00	00/0	01/0
01	10/0	01/0
10	00/0	11/0
11	00/1	00/1

Then, for the next state $\widetilde{y_1}\widetilde{y_2}$ and output z, we can use the above table to get the K-maps for $\widetilde{y_1}$,

y_1y_2 → X ↓	00	01	11	10
0	0	1	0	0
1	0	0	0	1

$\widetilde{y_1} = x'y_1'y_2 + xy_1y_2'$

For $\widetilde{y_2}$,

y_1y_2 → X ↓	00	01	11	10
0	0	0	0	0
1	1	1	0	1

$\widetilde{y_2} = xy_1' + xy_1y_2'$

And z:

y_1y_2 → X ↓	00	01	11	10
0	0	0	1	0
1	0	0	1	0

Z = y_1y_2

END OF SOLUTION

Part 2: Kuya Kokoy's Traysikol

Another cousin, Kuya Kokoy, drives a *traysikol* around town, and with money being tight, needs to keep careful track of exactly how much to charge his riders, so as to avoid both losing money for charging too little, and avoid losing trust by charging too much. He can easily do much of the math *but needs a way to track the distance that he is driving for his riders*.

Kuya Kokoy already has access to a Wheel Meter attached to the wheels, which will briefly return a value of 1 whenever he has driven a distance of 1 km—which is important, as he will have to charge riders a certain value per kilometre. (You can assume that there will be no charge for fractions of a kilometre.) The vehicle also has an Old Display, which takes in a binary input, and will display the decimal representation of said binary input (e.g., if its input is 0, it will display "0", and if its input is 1001, it will display "9", etc.). ***Design and implement a Rate Calculator circuit to connect these two components, such that the Old Display reads the distance traveled through the trip.*** ***Bonus points if you can implement an asynchronous sequential circuit instead of a synchronous one (i.e., remove the need for a Clock input).***

You may also include a "reset" input for Kuya Kokoy, such that when this input is 1, the machine resets and the distance reading is set back to 0.

You have also done some surveying of the neighborhood, and determined that the longest drive Kuya Kokoy can make is 7 kilometres.

Luckily for you, Mang Tomas has finally hit the jackpot in the semi-legal neighborhood gambling bets. It won him a grand total of 5 pesos, but it's enough glee for him to give you an unusually steep discount on this go round, so for Part 2, you will have access to all of the components in the shop. (In the case of Logisim-evolution, you will have access to all of the components available, as opposed to just the logic gates and D Flip Flops, as stated earlier)

SOLUTION:

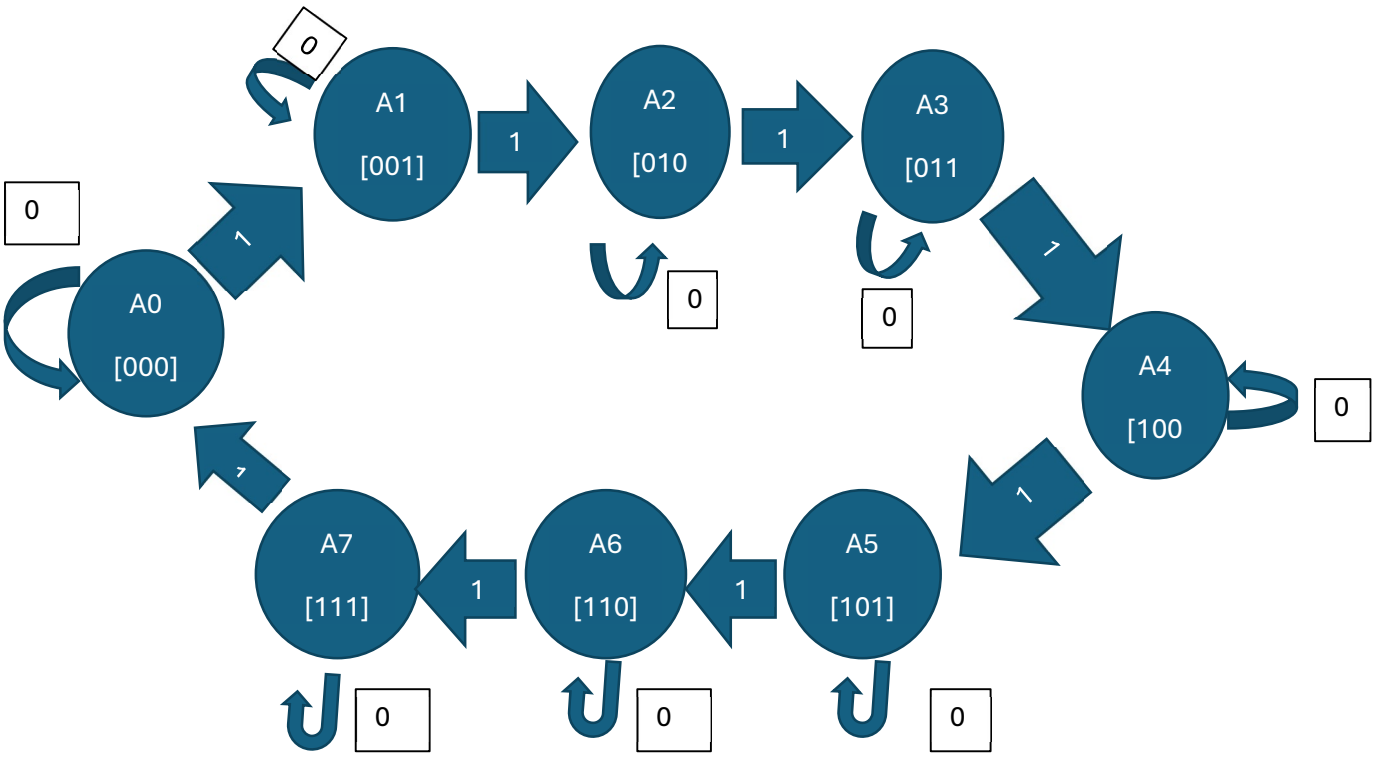
For a scenario like this, it is more natural for there to be 8 states—each one associated with the number of kilometres that have been driven up to that point, all the way from 0 to

7, and based on the information given about the Old Display, the output will have to be a three-bit that is the binary representation of the number of kilometres driven.

We can thus encode each of these states, such that the output z will be the same as the state $y_1y_2y_3$:

State	Encoding/Output ($y_1y_2y_3$)
A0	000
A1	001
A2	010
A3	011
A4	100
A5	101
A6	110
A7	111

Based on how we define the states, the contour of the state diagram and transition table is clear:



Present State ($y_1y_2y_3$)	Next State/Output ($y_1y_2y_3$)	Next State/Output ($y_1y_2y_3$)

	X = 0	X = 1
000 (initial)	000	001
001	001	010
010	010	011
011	011	100
100	100	101
101	101	110
110	110	111
111	111	000

Note that in this design, the state (and thus the output) only changes when the input x is 1. As such, we can easily implement an asynchronous version of the machine by removing the Clock input, and connecting the input x to the Clock inputs of the D Flip Flops.

Similarly, the next state is always the same as the previous state + 001. We can thus calculate the next state/output by using a Bit Extender to extend the one-bit x input into a three-bit, and then use this new three-bit x and the previous state ($y_1 y_2 y_3$) as inputs to a three-bit adder, the output of which will be the next state/output.

Using a “reset_button” input and the Reset input of the D Flip Flops, we can also incorporate the reset feature, forcing the state to return to the initial 000 state whatever the next x is.

END OF SOLUTION