

ECE 2140 – 103

Colby Alley, Jacob Hendrix, and Andrew Marin

ckalley42@tnitech.edu, amarin42@tnitech.edu, and jthendrix42@tnitech.edu

Lab 103

Date: 12/3/25

Due Date: 12/5/25

Lab Instructor: Marim Mahmoud

Part 2 Project Report – Sequence Detector

To continue with the final part of the project, we will design a sequence detector that will detect sequence number one from the previous part's sequence generator, which was 0001. We will have an input, I, and an output, Y. Input I will receive the bits of the sequence generator while the output Y will output zero or one depending on if the full sequence was received. In my implementation, I will use a Mealy approach which will allow us to use fewer flip flops as if we were to use a Moore approach. Since we have four bits, we will need two D flip flops for our four states.

We will be using the 74HC74 dual flip flop integrated chip. Below is figure one that displays the state diagram of the sequence detector.

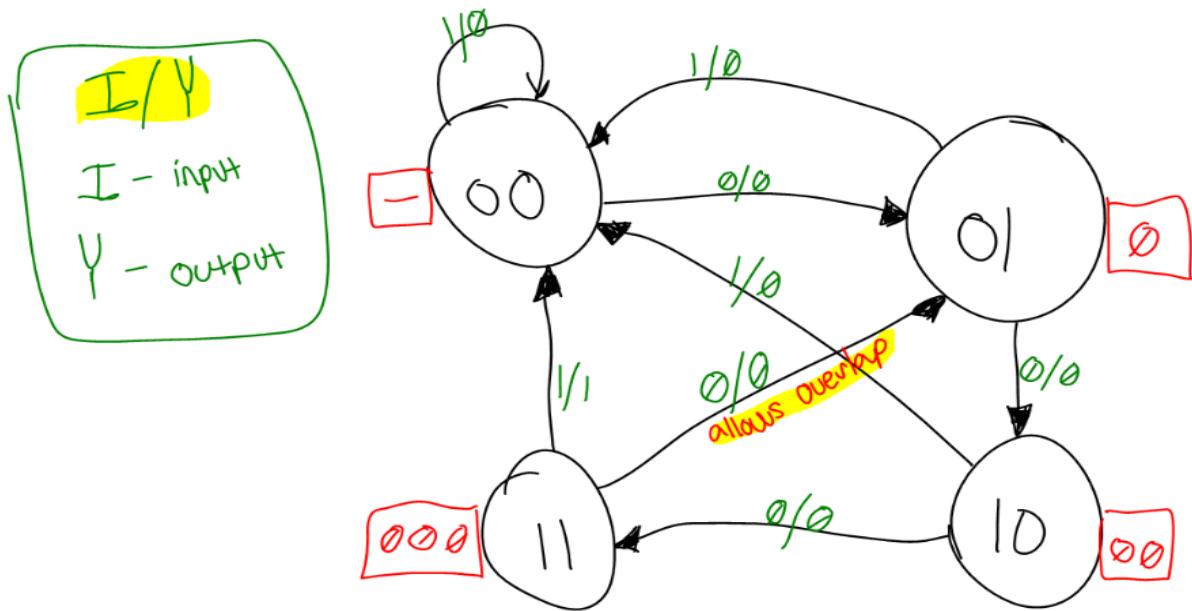


Figure 1: State Diagram

In this diagram, each state alongside its input and output are displayed. When we are in state 00 and our input is 0, that means we have received the first bit of the sequence, so we move onto the next state. This is displayed by the red boxes with numbers of the sequence received.

However, if we received one, we would remain in the first state until the next bit was received. If we are in state 01 and our input is 1, that means we have been reset to receiving the first bit, so we move back to the first state. If we receive zero, that means we have received the second bit, so we move onto the next state 10. If we receive one in state 10, that means we reset again and move back to state 00. If we receive zero, we have received the second to last bit, so we move onto the next state. So far, our output has remained zero since we have not received the entire sequence. Once we are in state 11, if we receive one, we will have received the entire sequence, resetting the state back to 00 and returning output one to show we have detected the full sequence. If we receive zero, we could either return to the first state or the second state. Moving to the first state would mean overlapping is not allowed. So, the second state 01 would allow overlapping, which is how I have implemented it. When it receives zero instead of one, it goes to the second state 01 and knows it has received the first bit of the sequence.

It is now time to build the state table for our state diagram. Below in figure two is our state table determined by our state diagram above. We will now need to use k-maps to build functions based off the input and current states. This confirms our Mealy design circuit. Since we know in D flip flops the characteristic equation is $D(t) = Q(t+1)$, we will be able to implement this easily on our circuit board.

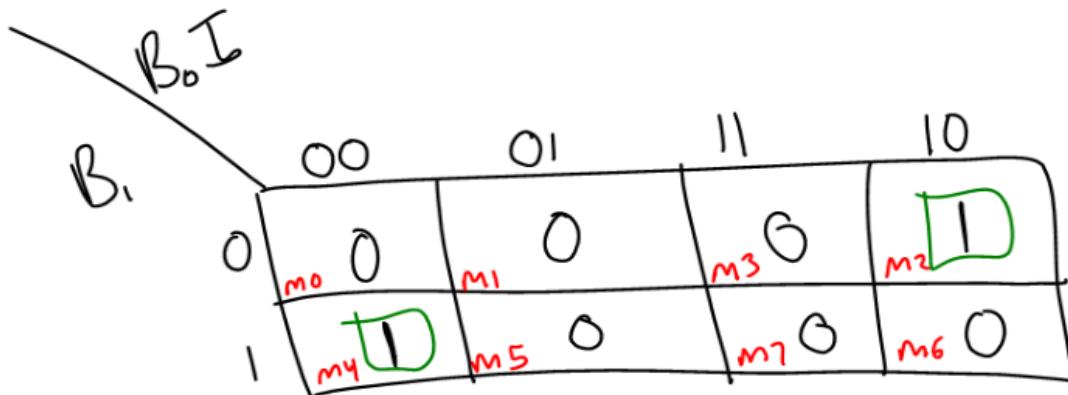
Current States		Inputs	Next State		Output
B_1	B_0	I	B_1	B_0	Y
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	1	0	0
0	1	1	0	0	0
1	0	0	1	1	0
1	0	1	0	0	0
1	1	0	0	1	0
1	1	1	0	0	1

Figure 2: State Table

Next, I will use k-maps to find a relationship between our current states and our input and outputs. Attached below will show the work in that process:

Finding D_{B_1} in terms of

I , current B_0 and current B_1 :



$$D_{B_1}(+) = (B_0 \oplus B_1) I'$$

$$D_{B_0}(+) = (B_0 \oplus B_0) I'$$

Proof:

$$= (B'_0 B_0 + B_0 B'_0) I'$$

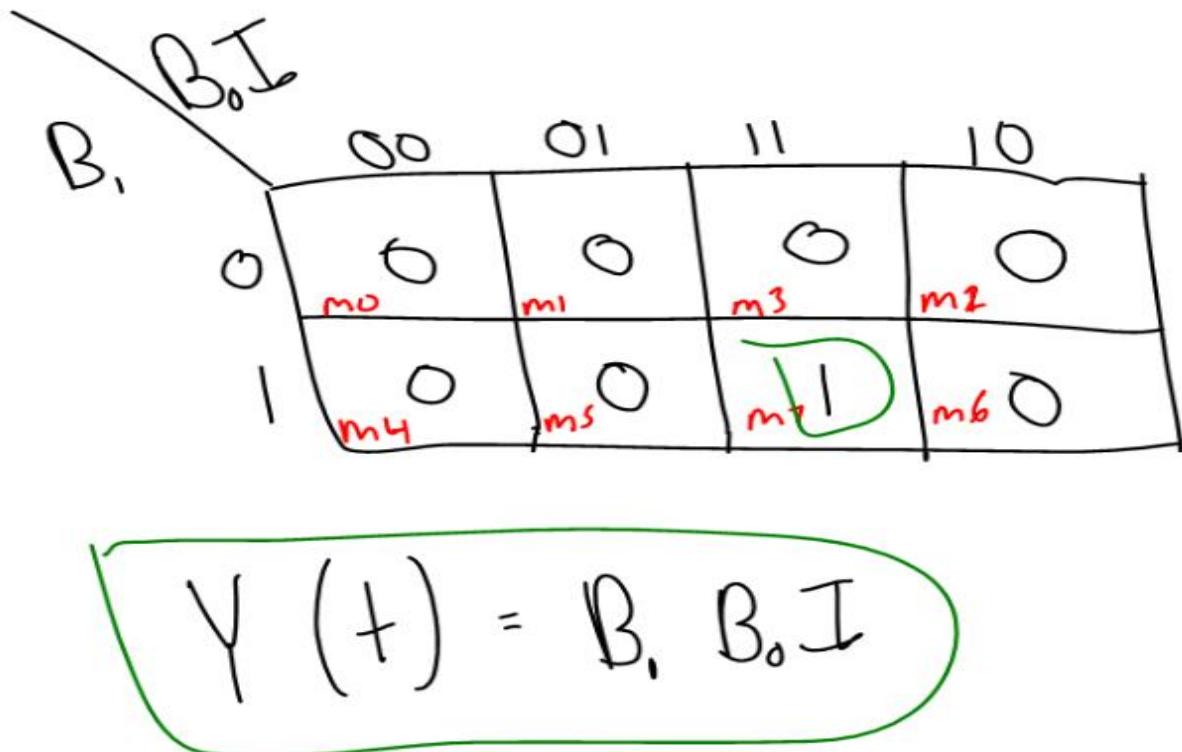
$$= B'_0 B_0 I' + B_0 B'_0 I'$$

$$= (B_0 \oplus B_0) I'$$

Finding D_{B_0} in terms of I , current B_0 and current B_1 :

	00	01	11	10
0	m0 1	0 m1	0 m3	0 m2
1	1 m4	0 m5	0 m7	1 m6

$$D_{B_0}(+) = B'_0 I' + B_0 I'$$



Now we have our output and next states in terms of our current states and input. This is all we need to construct the actual circuit. Since we are using D flip flops, we can use the characteristic equation $D(t) = Q(t+1)$ to determine our D inputs for our flip flops. We have named these B_0 and B_1 . B_1 will be our most significant state while B_0 will be our least significant state. Y will be our output bit. We have the following equations (derived above):

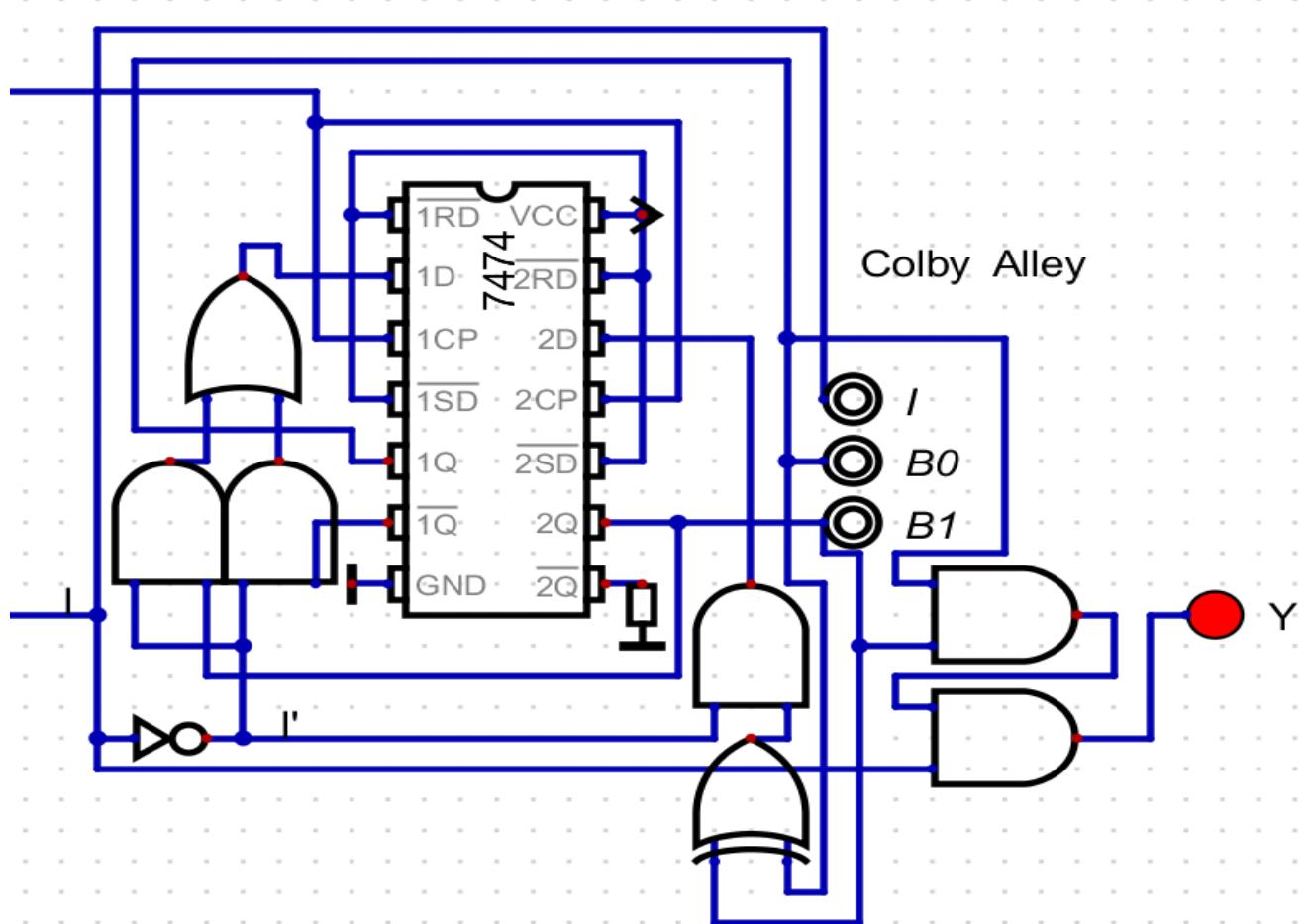
$$DB_0(t) = B_0'I' \text{ OR } B_1I'$$

$$DB_1(t) = (B_1 \text{ XOR } B_0)I'$$

$$Y(t) = B_1B_0I$$

Now, I will use these equations to build our digital tool circuit, test it, and then implement it on my breadboard to ensure it works as intended. Attached below is the digital tool figure. Note: I have the part one sequence generator off to the side to use the output of the sequence generator

for the input of the sequence detector. Since it is not the priority of this part of the project, it is not shown.



With this next circuit, it is slightly more complex than the previous circuit. We will need extra chips like NOT and OR to be used when implementing the physical circuit. I have tested it with my sequence generator, and it appears to work as intended. Output is connected to my simplified Boolean expression above as well as my current and next states. We will need to use 74HC74 dual flip flop chip, 74HC08 dual AND gate chip, 74HC86 dual XOR chip, 74HC32 dual OR chip, and the 74HC04 NOT chip to implement this circuit. I will use the StaticIO in the WaveForms program for our I input alongside the states (B1 and B0) to share what they are doing during the circuit. I will test both the input I as a switch as well as the input from the sequence

generator to test the overlapping and regular sequence detector. I will use a physical LED to show the sequence detector on our breadboard. Attached below is figure three that shows how the connections will be made using our Analog Discovery Kit. I have just put the input I as a wire coming in alongside the same clock that is used in the sequence generator from part one of the project.

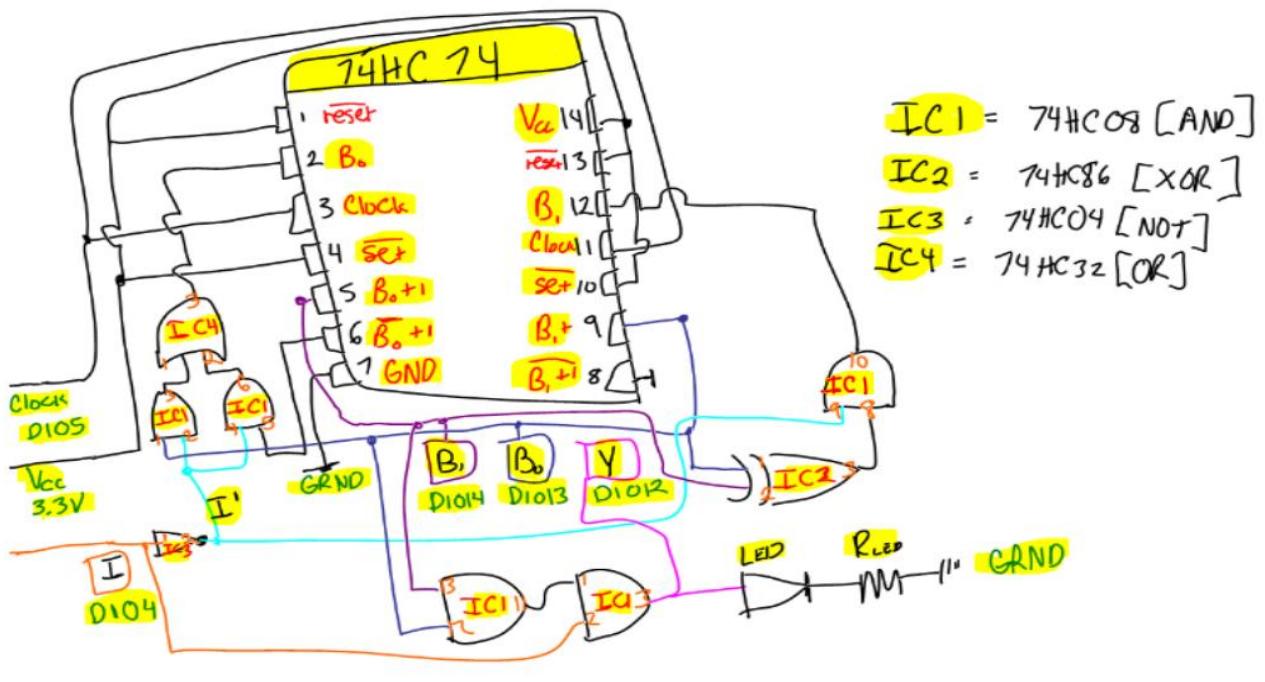
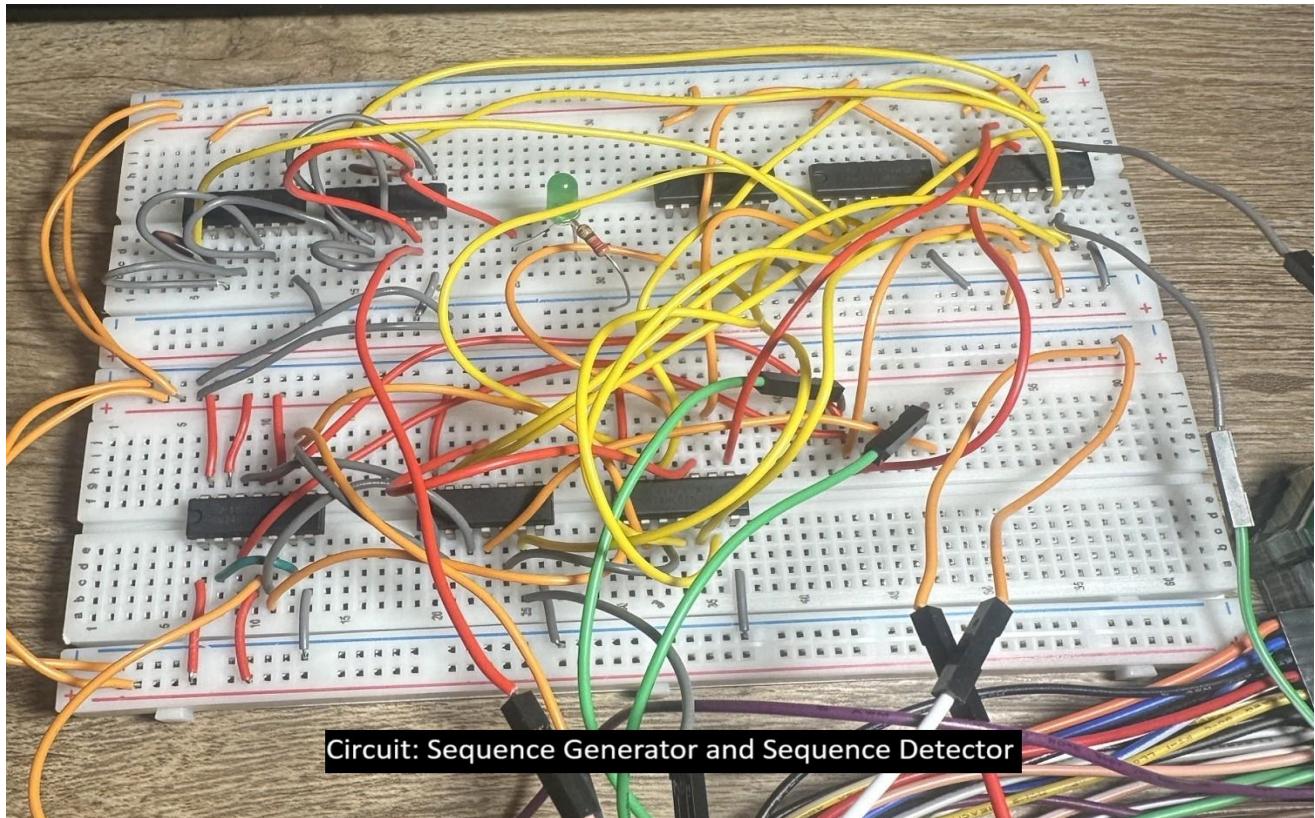


Figure 3: Schematic of Circuit

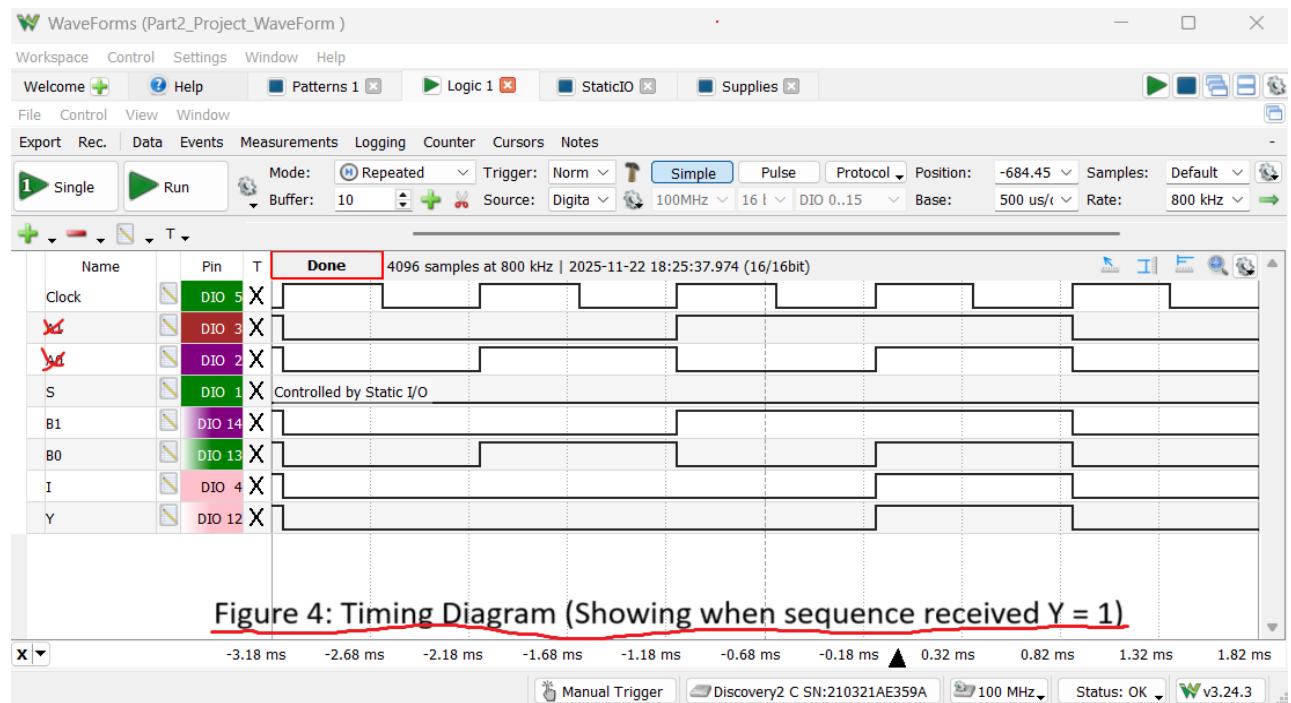
Above I have the states (**B₁** and **B₀**) connected to DIO14 and DIO13 and will use StaticIO to display them as LEDs in the circuit. I will do the same with DIO12 and use a physical LED on the circuit to display the sequence detector's output. I have the same clock connected to DIO5 from the sequence generator and the input connected to the same output of the sequence generator for the new sequence detector. Below is a picture of my physical circuit design. I have implemented it on a separate breadboard for more space to implement the newly used IC chips. I have attached

a photo that shows both the sequence generator and sequence detector. It looks complicated, but it was the simplest way for me to implement both the sequence generator and sequence detector.



Above is the picture of the sequence generator and detector combination. I had to use NAND gates (74HC00 chips) for more AND gate implementations as I ran out of inputs and outputs for my AND chip. It has been personally tested, and I am glad to say it works as intended. It will also be shown to our lab instructor for verification. I used the same pins labeled in the circuit schematic and the circuit works correctly. The only difference was NAND gates were used in place of the AND gates for the output (Y). I tested it with sequence number one (0001), and it detects it correctly. The sequence detector outputs zero unless the entire sequence was detected, and then it will output one. I have this connected to an LED and the LED confirms the output the same way my StaticIO in WaveForms. Figures four and five below show the timing diagram of the circuit when the sequence detector detects the sequence (selector is zero). I will show the timing diagram

for overlapping to the lab instructor during the demonstration as it is more difficult to show that as the frequency is so low to see it. The way WaveForms works it is harder to demonstrate the timing diagram for overlapping due to the higher frequency and it depends on inputs (S from the sequence generator). However, I have drawn a timing diagram for the overlapping case. This is demonstrated in figure six. For the timing diagram, the frequency has been increased to show cycles. The demonstration will show the circuit slowed down so we are able to see the circuit detect the sequence per each state. In these timing diagrams, it is a rising edge clock.



Note: Figure 4 shows when S = 0 as well.

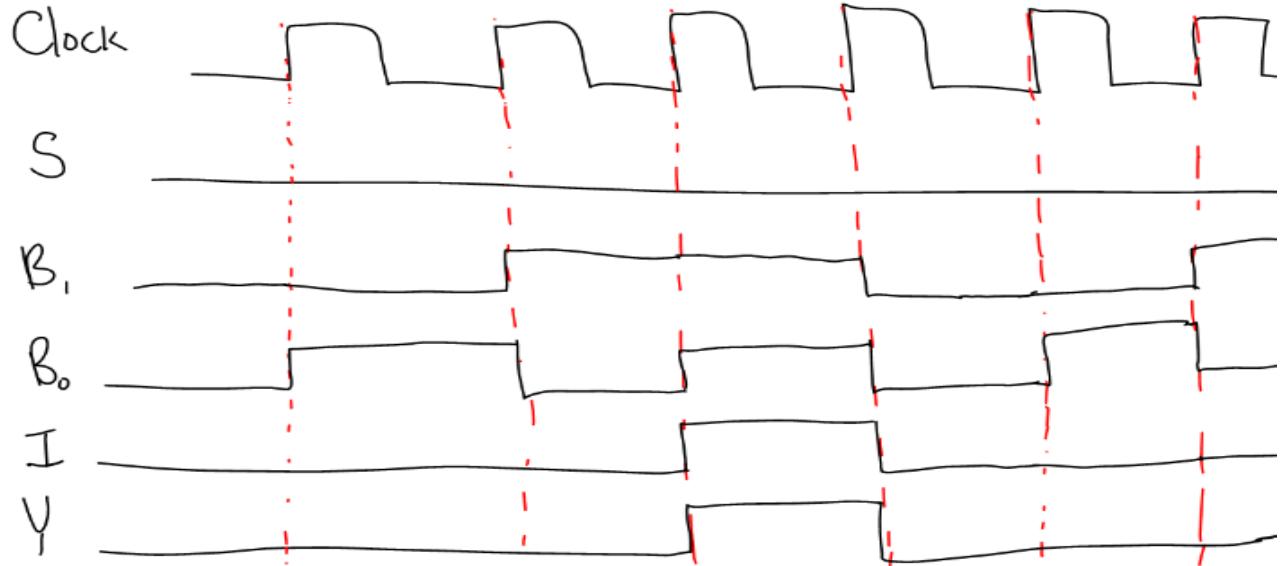


Figure 5: Timing Diagram
(Showing when sequence is received $Y=1$)

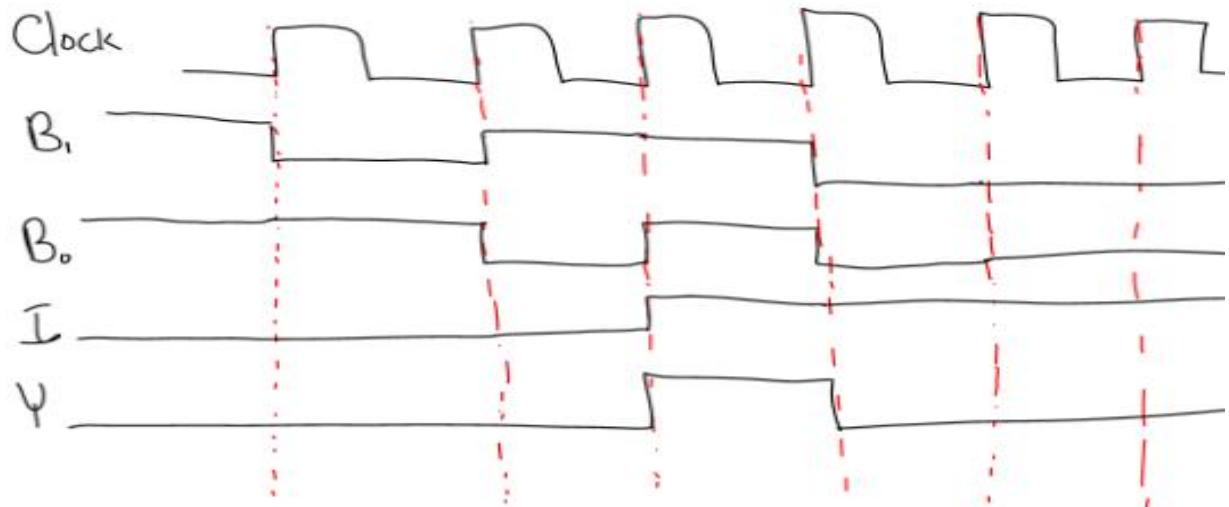


Figure 6: Timing Diagram
(Showing when sequence is received with overlapping)

Now, here are the timing diagrams to show when the selector equals one. Figure seven shows the Logic Analyzer timing diagram while figure eight shows my drawn diagram. These figures also slightly show the overlap when the input is zero during this sequence (1110).

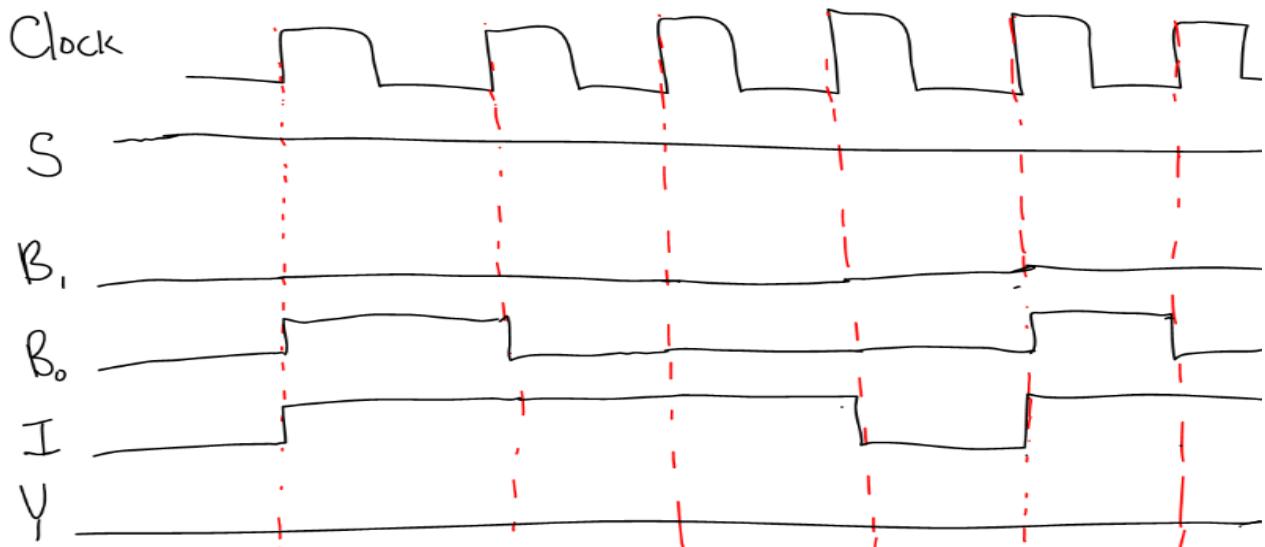
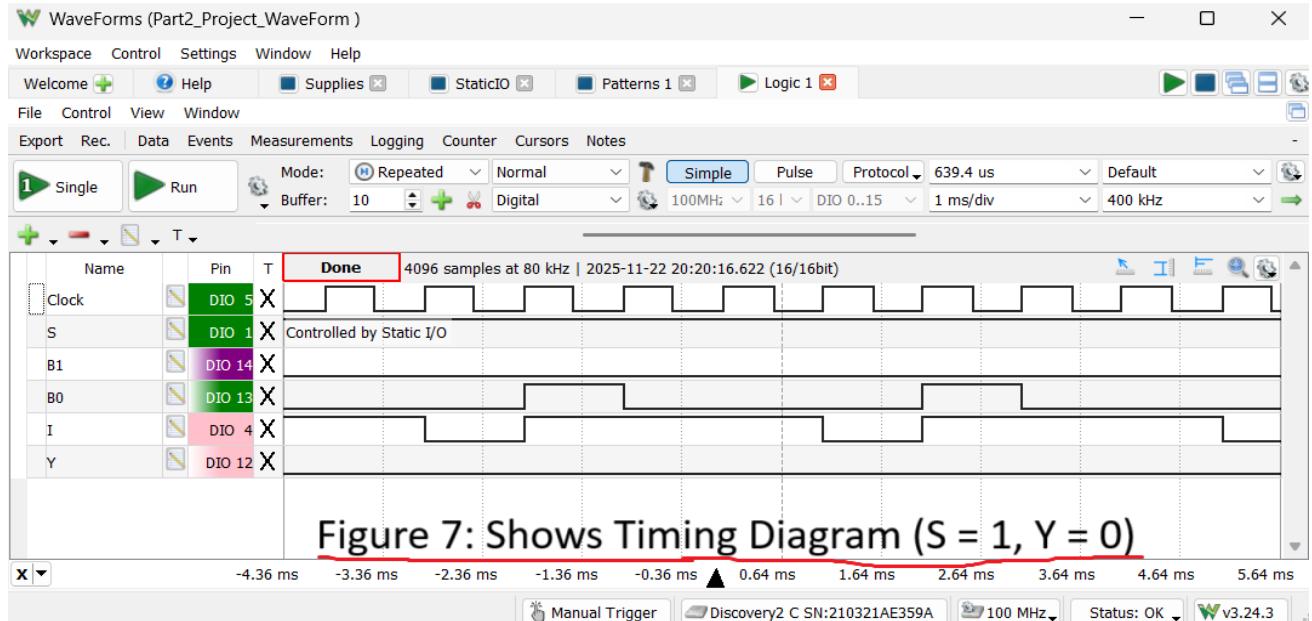


Figure 7 : Timing Diagram
(Shows Timing Diagram (S = 1, Y = 0))

Conclusion

To conclude, the sequence detector worked as intended after the design was executed. The combination of the sequence generator and sequence detector works correctly. The timing diagram matches correctly with the output of the sequence detector. The overlapping case will be demonstrated to the lab instructor during my project demonstration. This is more difficult to demonstrate in the WaveForms generator due to how my circuit relates to the sequence generator. The detector correctly detects sequence one (0001) through StaticIO as well as the Logic Analyzer. The LED on my breadboard also outputs one when the sequence is fully detected. This was trickier to implement as there were many more integrated chips to use during the implementation. However, in the end, I was able to make it correctly work with my state diagram and state table.