CMPE 160.01 Digital System Design I

Laboratory Exercise 8

Design and Simulation of a Moore State Machine

Adam Schultzer
3/28/2024

Lab Section:	3
Instructor:	Mr. Lange
TA:	Andy Lin
	Christian Rockwell
	Ruhan Syed
	Matthew Tosi
Lecture Section:	01
Lecture Instructor:	Richard Cliver

Abstract

The goal of this exercise was to implement and simulate a Moore State Machine, which is a state machine who's inputs are dependent only on the internal state of the memory instead of using data from inputs to the circuit directly. Specifically, a Moore State Machine was designed that could detect a specific sequence of values from two inputs. Thi circuit was compiled and tested in ModelSim and on a physical MAX10 FPGA (Field Programmable Gate Array) board by looking at the LED (Light Emitting Diode) output pins, and was verified to be successfully created and functional.

Design Methodology

The goal of the exercise was to create a circuit that could detect when two bits, (A, B) went through the sequence 11, 10, 00. This was done by creating a Moore State Machine with a 2-bit state $Q = (Q_1, Q_0)$. There were four possible states of the state machine. Firstly, $Q = 00$ (Idle), which was for any time that the state machine did not detect the beginning of the sequence and was not already in the middle of the sequence. Second, $Q = 01$ (Got11), which was set as the state after an initial $(A, B) = 11$ was detected. Third, $Q = 11$ (Got10), which was set as the state when the last state was Got11 and $(A, B) = 10$. Fourth, $Q = 10$ (GotAll), which was set as the state when the last state was either Got10 or GotAll and the current input was $(A, B) = 00$. Besides the (A, B) inputs, there was also a RST input that would immedietely set the state of the machine to Idle.

To demonstrate these concepts, a state diagram was created, shown in Figure 1.

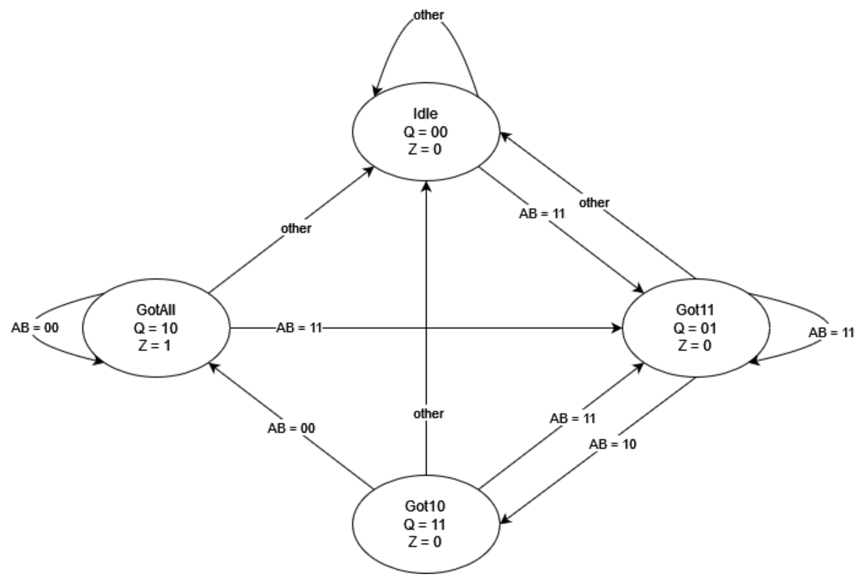

Figure 1: *State Diagram for the Sequence Detector*

Figure 1 shows how each state leads to the next and under what conditions. This helped to solidify the exact behavior of the state machine. In order to turn this concept into a circuit, a state table was created based upon the state diagram, shown in Table 1.

Table 1: *State Table for the Sequence Detector*

RST	Q_1	Q_0	A	B	Q_1^*	Q_0^*	Z
1	*	*	*	*	0	0	0
0	*	*	1	1	0	1	*
0	0	0	*	*	*	*	0
0	0	1	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	0	0	0	1	0	1
0	*	*	*	*	0	0	*

Table 1 shows the intended state changes that should occur within the state machine when designed, and the last row represents that any conditions that were not previously defined in other rows will always lead to the next state being Idle. In order to make the circuit design process simpler, this state table was used to design boolean expressions that describe the state machine. Equations 1-3 describe the state transitions and how the output Z is defined.

$$Q_1^* = \overline{Q_1}Q_0A\overline{B} + Q_1\overline{A}\,\overline{B} \tag{1}$$
$$Q_0^* = \overline{Q_1}Q_0A + AB \tag{2}$$
$$Z = Q_1\overline{Q_0} \tag{3}$$

These equations define the next state of the state machine as a SOP expression of the current state and the current inputs, allowing the entire state machine to be designed around a SOP circuit using minterms. In Figure 2, a circuit designed around these equations is shown.

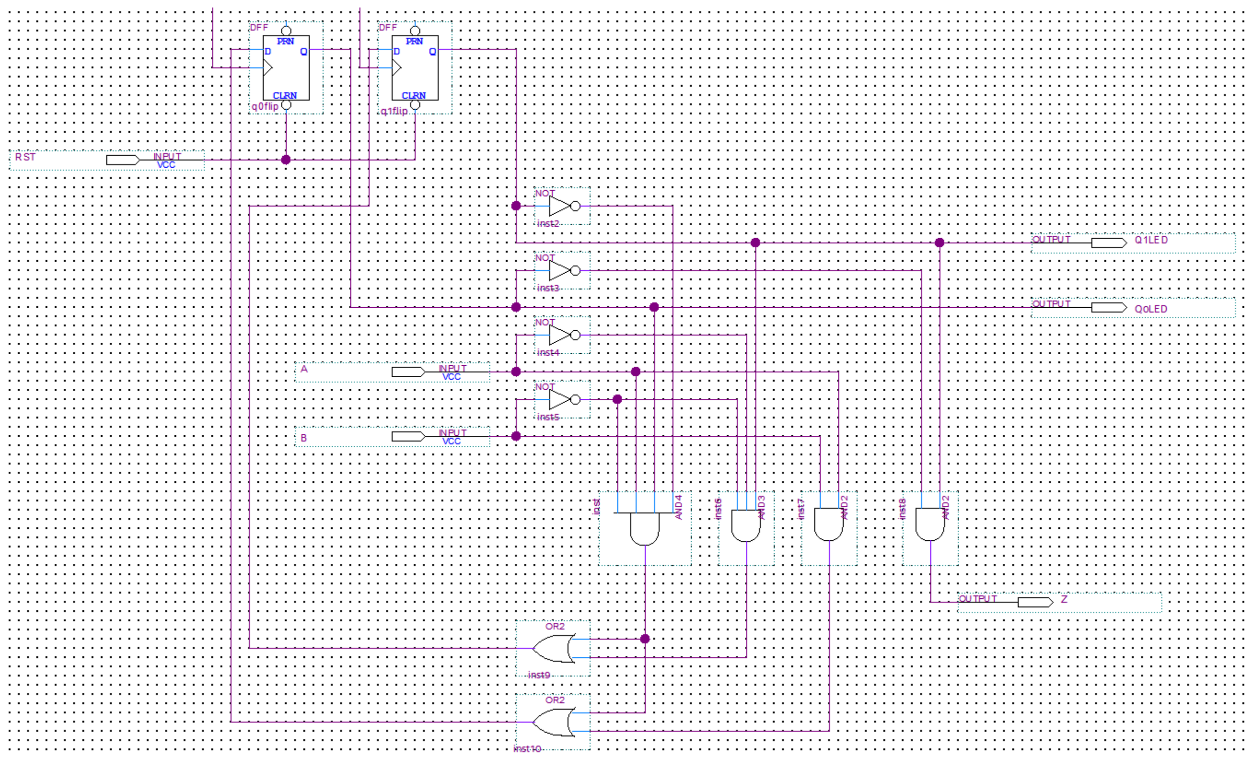

Figure 2: *Implementation of a Sequence Detector in Quartus Prime*

Figure 2 displays the sequence detector as a circuit. It uses three minterms to define the next state of the circuit and one to produce the output of the design. The minterm for $\overline{Q_1}Q_0A$ in Equation 2 could be substituted with the existing minterm for $\overline{Q_1}Q_0A\overline{B}$, as the outputs are equivalent due to the other minterm in Equation 2, AB.

The sequence detector was implemented into the MAX10 FPGA by connecting the signals to the pins shown in Table 2. Two PLLs were also used alongside a 50MHz clock signal to create a 2Hz clock for the clock inputs on the D-flip-flops.

Table 2: *Sequence Detector Pin to Signal Connections*

Signal	MAX10 FPGA Pin
A	SW1
B	SW0
$Q1LED$	LEDR9
$Q0LED$	LEDR8
Z	LEDR0
clk_50MHz	MAX10_CLK1_50
RST	KEY0

Table 2 shows the connections to the FPGA for the sequence detector. A hardware test verified that the design operated correctly on the FPGA.

Results and Analysis

To verify the circuit built in Figure 2, the sequence detector circuit was simulated using ModelSim, using two different sequences. The first of the input sequences produced the waveform shown in Figure 3.

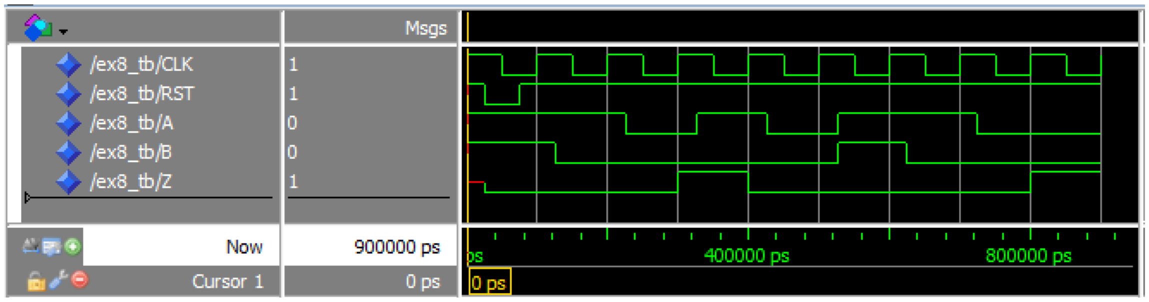

Figure 3: *Waveforms Produced by the Sequence Detector in ModelSim for Sequence 1*

Figure 3 shows the waveform output of the sequence detector when the input sequence is (A,B) = (1,1), (1,0), (0,0), (1,0), (0,0), (1,1), (1,0), (0,0), (0,0). It demonstrates the full sequence being detected for one clock cycle, and a partial sequence being detected later. In order to test another feature of the detector, a second sequence was tested. The waveforms produced when testing this second sequence are shown in Figure 4.

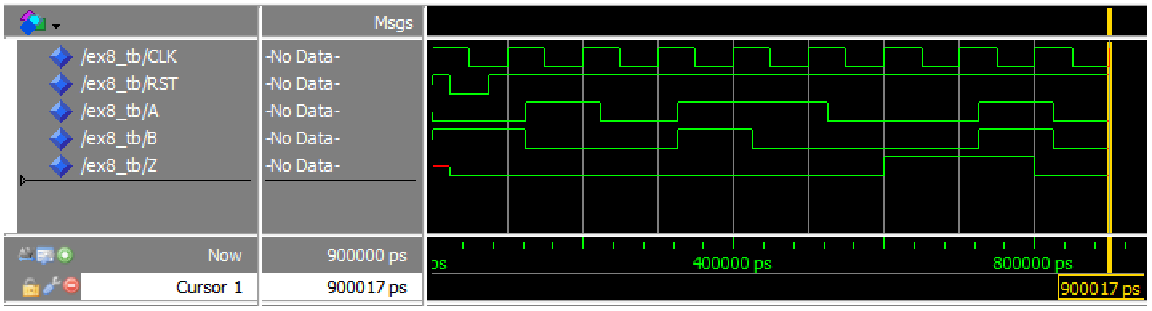

Figure 4: *Waveforms Produced by the Sequence Detector in ModelSim for Sequence 2*

Figure 4 shows the waveform output of the sequence detector when the input sequence is (A,B) = (0,1), (1,0), (0,0), (1,1), (1,0), (0,0), (0,0), (1,1), (0,0). This sequence demonstrates how the detector's output continues to be high as long as the input continues to be (A,B) = (0,0). The waveforms in Figures 3 and 4 demonstrate expected behavior and therefore the exercise was successful.

Conclusion

Sequence detectors are a very common part of modern electronics, although may sometimes be defined in software rather than hardware. For example, the handshakes that happen at the beginning of many IO interactions, like when a USB device is plugged in, use sequence detection in order to fully define what is happening and to set a starting point for communications. This exercise was successful, as the sequence detector successfully performed the operations expected and reacted to inputs as expected.

Questions

1. Repeat the synthesis process of your state machine where Idle = 00, Got11 = 01, Got10 = 10, and GotAll = 11

The state diagram is almost the exact same, with the only change being that GotAll and Got10 have switched Q values. Table 3 shows the new state table with the new encoding.

Table 3: *State Table for the Sequence Detector with new encoding*

RST	Q_1	Q_0	A	B	Q_1^*	Q_0^*	Z
1	*	*	*	*	0	0	0
0	*	*	1	1	0	1	*
0	0	0	*	*	*	*	0
0	0	1	1	0	1	0	0
0	1	0	0	0	1	1	0
0	1	1	0	0	1	1	1
0	*	*	*	*	0	0	*

Table 3 shows the new state table. Using this new state table, new equations can be made. Equations 4-6 show these new equations.

$$Q_1^* = \overline{Q_1}Q_0A\overline{B} + Q_1\overline{A}\,\overline{B} \tag{4}$$

$$Q_0^* = Q_1\overline{A}\,\overline{B} + AB \tag{5}$$

$$Z = Q_1Q_0 \tag{6}$$

Equations 4-6 show that Q_1^* is unchanged, and Z and Q_1^* are slightly modified to fit the new table.

2. How many different ways are there to encode a state machine with four states that is implemented using two D-flip flops? Note that the answer should be a number, not ”one-hot” or ”gray”.

A state machine with four states and two D-flip flops could be encoded $4 * 3 * 2 * 1 = 24$ different ways.

3. What is ”one-hot” encoding style?

”One-hot” encoding is when a single bit is high for each individual state, and thus requires the same number of bits as states.