Laboratory Exercise 6

Finite State Machines

The purpose of this exercise is to learn how to create and use finite state machines.

Preparation Before the Lab

You are required to complete Parts I to III of the lab by writing and testing Verilog code and compiling it with Quartus II. Show your schematic, Verilog, and simulations for Parts I to III and state diagrams for Parts II and III to the teaching assistants. You must simulate your circuit with ModelSim (using reasonable test vectors using the format shown in the previous lab).

In-lab Work

You are required to implement and test all of Parts I to III of the lab. You need to demonstrate all parts to the teaching assistants.

Part I

We wish to implement a finite state machine (FSM) that recognizes two specific sequences of applied input symbols, namely four consecutive 1s or the sequence 1101. There is an input w and an output z. Whenever $w = 1$ for four consecutive clock pulses, or when the sequence 1101 appears on w across four consecutive clock pulses, the value of z has to be 1; otherwise, $z = 0$. Overlapping sequences are allowed, so that if $w = 1$ for five consecutive clock pulses the output z will be equal to 1 after the fourth and fifth pulses. Figure 1 illustrates the required relationship between w and z.

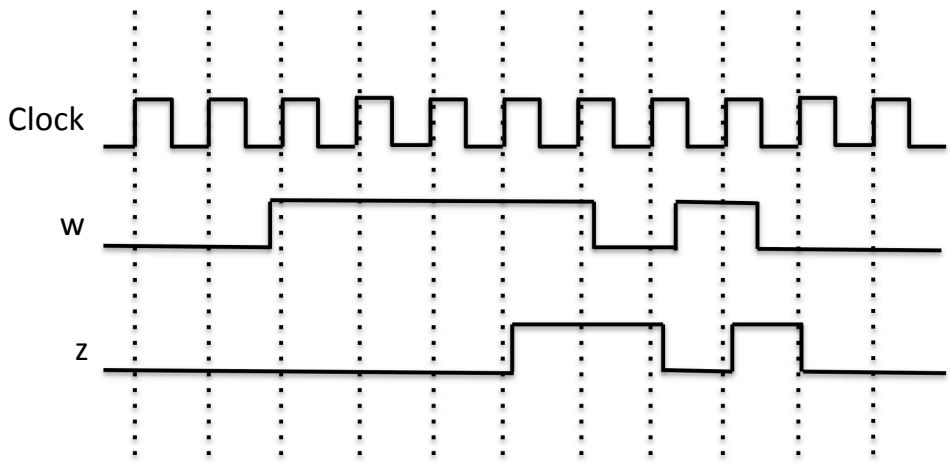

Figure 1: Required timing for the output z.

A state diagram for this FSM is shown in Figure 2. For this part you are to manually derive an FSM circuit that implements this state diagram, including the logic expressions that feed each of the state flip-flops.

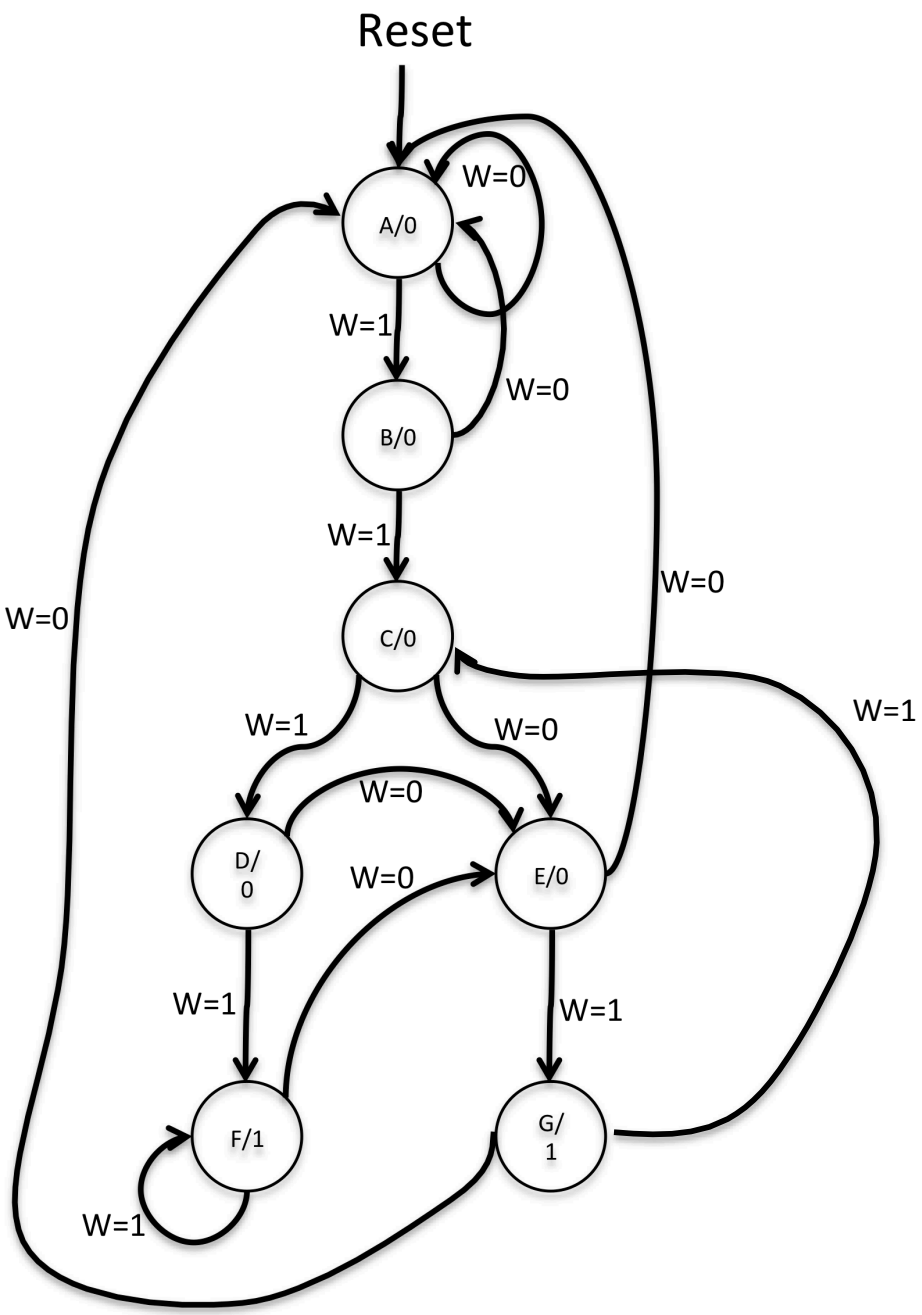

Figure 2: A state diagram for the FSM.

Perform the following steps:

1. Draw a schematic outlining the hierarchies you will use and explain them to the TA as part of your prelab.
2. Write a Verilog file that realizes the behaviour described above. Use the toggle switch SW_0 on the DE1-SoC board as an active-low synchronous reset input for the FSM, use SW_1 as the w input, and the pushbutton KEY_0 as the clock input which is applied manually. Use the red light $LEDR_9$ as the output z, and assign the state flip-flop outputs to $LEDR_{3-0}$.
3. Simulate your circuit with ModelSim for a variety of input settings, ensuring the output waveforms are correct. You must show this to the TA as part of your prelab.

4. Compile the project.
5. Download the compiled circuit into the FPGA chip. Test the functionality of the circuit.

Part II

For this section, you will be designing a 4-bit serial divider using a finite state machine.

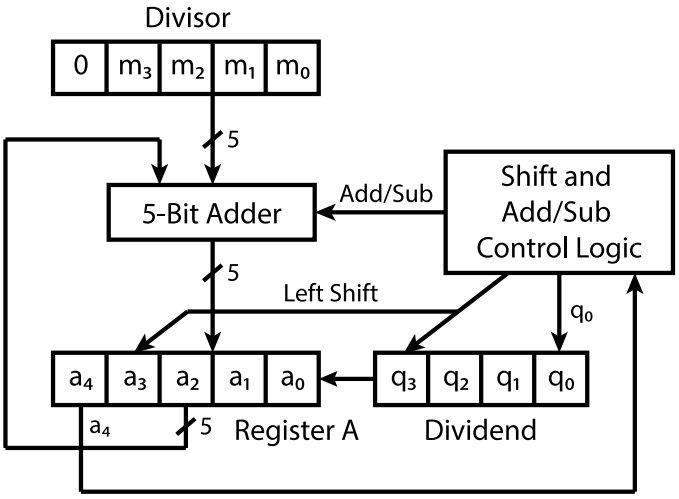

Figure 3: Schematic of serial divider.

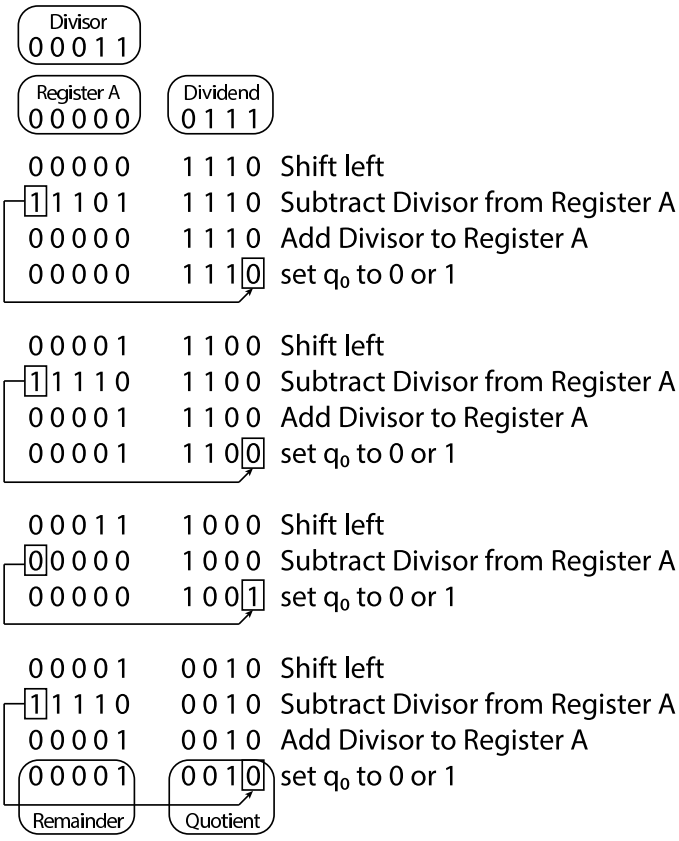

Figure 4: An example of functionality of serial divider.

Shown in the example above, the serial divider starts with *Register A* set to *0*. The *Dividend* is shifted left. Any overflowing value from the left shift will be carried over to *Register A* (as shown in the schematic).

The *Divisor* is then subtracted from *Register A*. The left most bit (called the most significant bit or MSB) is saved. If the MSB is a *1*, then we restore *Register A* back to its original value by adding *Divisor* to *Register A*, and set the right most bit (called the least significant bit or LSB) of the *Dividend* to *0*. Else, we do not perform the addition and immediately set the LSB of the *Dividend* to *1*.

This cycle is performed until all the bits of the *Dividend* have been shifted out. Once the process is complete, the new value of in place of the location of the *Dividend* is the *Quotient*, and *Register A* will hold the value of the *Remainder*

To implement this part, you will use SW_{3-0} for the divisor value and SW_{7-4} for the dividend value. Use *CLOCK_50* to for the clock signal, KEY_0 as a synchronous active high reset, and KEY_1 as the go signal to start computation. The output of the *Divisor* will be displayed on *HEX0*, *Dividend* will be displayed on *HEX2*, *Quotient* on *HEX4*, and *Remainder* on *HEX5*. Set the remaining HEX displays to *0*. Also display the *Quotient* on *LEDR*.

You are allowed to use the add and subtract operator symbol. It is recommended that you have the state transitions in one *case* statement, while having all other operations in another *case* statement controlled by *CLOCK_50*.

Perform the following steps.

1. Draw a schematic & state diagram outlining the hierarchies you will use and explain them to the TA as part of your prelab.
2. Write a Verilog file that realizes the behaviour described above.
3. Simulate your circuit with ModelSim for a variety of input settings, ensuring the output waveforms are correct. You must show this to the TA as part of your prelab.
4. Compile the project.
5. Download the compiled circuit into the FPGA chip. Test the functionality of the circuit.

Part III

A common use for finite state machines is inside processors where there exists a datapath. In the previous labs, you learned how to construct a simple ALU. In this section, you will be given a schematic of a datapath. You are to construct the datapath and to build a finite state machine which performs the quadratic function

$$Ax^2 + Bx + C$$

using given datapath. The value of *x* will be preloaded along with the other constants.

The datapath consists of four registers to store the constants of *A*, *B*, and *C* and a value for *x*. The four registers are selected by two 4 to 1 multiplexers for inputs to the ALU, and the ALU can perform addition or multiplication. The output of the ALU connects to two registers which then feed back into register storing value of *A* and *B*.

All registers have enable signals which can turn on and off the register's functionality. You will have in total 10 control signals as seen in the schematic below (note that the ALU also has a control signal which is not shown).

To start the operation, you will preload registers *RX* and *RA*, then registers *RB* and *RC*. This will take 2 cycles. You are free to perform operations in any order as long as if the result is correct.

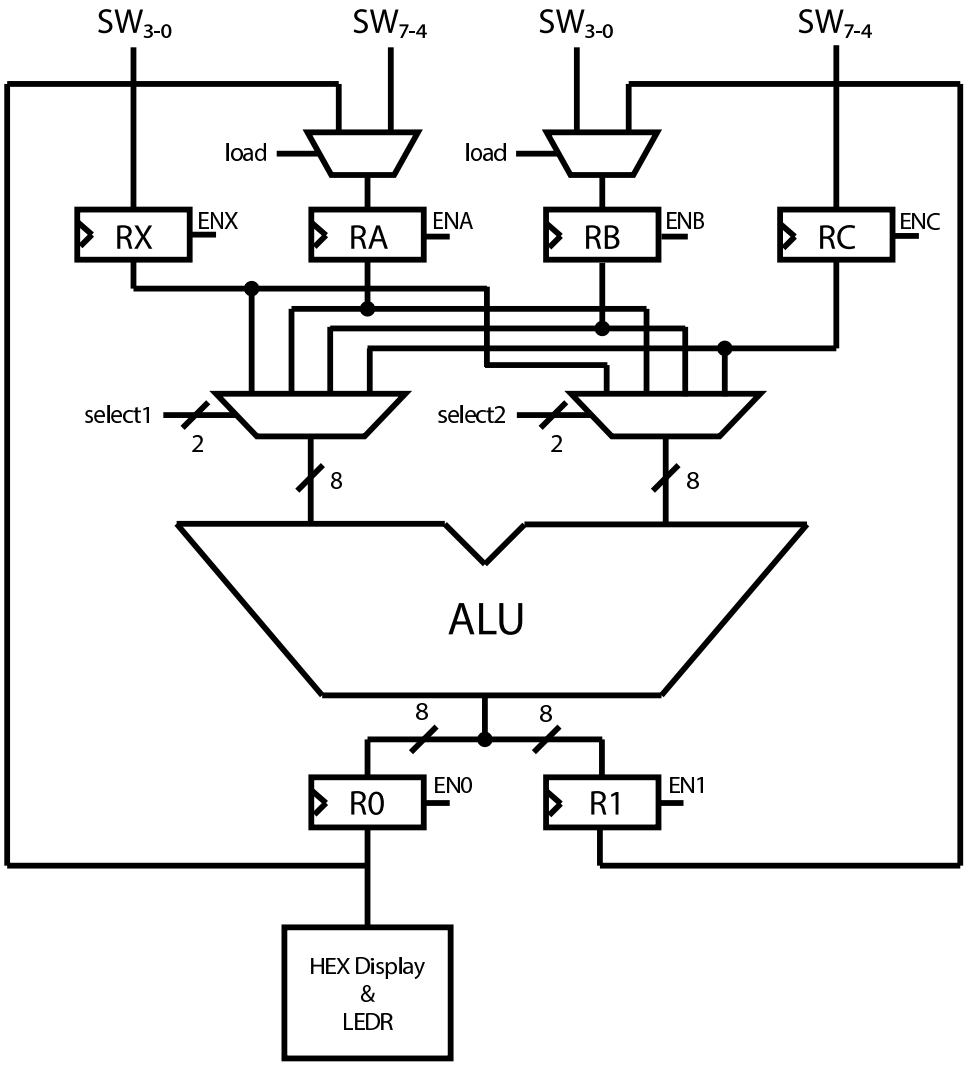

Figure 5: Schematic of datapath.

To implement this part, you will use SW_{3-0} for value of x and value of B, SW_{7-4} for value of A and C. You will use *CLOCK_50* as your clock, KEY_0 as an active high synchronous reset, KEY_1 to load values of x and B, and KEY_2 to load values of A and C. The result is displays on $LEDR_{7-0}$ and *HEX0* and *HEX1*.

It is recommended that you separate the datapath from the control unit (the finite state machine). The control unit should be broken up into two case statements similar to the previous section. One case statement handles the state transitions and the other case statement turns on and off enable bits controlling the datapath.

Perform the following steps.

1. Draw a schematic & state diagram outlining the hierarchies you will use and explain them to the TA as part of your prelab.
2. Write a Verilog file that realizes the behaviour described above.
3. Simulate your circuit with ModelSim for a variety of input settings, ensuring the output waveforms are correct. You must show this to the TA as part of your prelab.
4. Compile the project.
5. Download the compiled circuit into the FPGA chip. Test the functionality of the circuit.