LAB 6

FINITE STATE MACHINES

In Lab 6, you gain more experience designing and analyzing finite state machines (FSMs). You start by designing the brain for a new robotic snail, Robo-Snail. Then you analyze an FSM that controls a *datapath* to perform a complex operation.

This document describes what you need to prepare and demonstrate for Lab 6. Section 6.3 describes the tasks you must complete *before* your lab session. Section 6.6 describes the tasks you complete *during* your lab session. The next section describes lab logistics in more detail.

6.1 Logistics

Even though you work in pairs during your lab session, you are assessed individually on your Lab Preparation ("pre-lab") and Lab Demonstration ("demo"). All pre-lab exercises are submitted electronically before your lab (see the course website for exact due dates, times, and the submission process). So, **before** each lab, you must read through this document and complete all the pre-lab exercises. During the lab, use your pre-lab designs to help you complete all the required in-lab actions. The more care you put into your pre-lab designs, the faster you will complete your lab.

The Lab Preparation must be completed individually and submitted online by the due date. Follow the steps in Section 6.3 for the pre-lab. Remember to **download the starter files**.

You must upload *every required file* for your pre-lab submission to be complete. But you do *not* need to include images that are not on the list of required files (even if those images are in your lab report). If you have questions about the submission process, please ask ahead of time. The required files for Lab 6's pre-lab (Section 6.3) are:

- Your lab report: `lab6_report.tex`, `lab6_report.pdf` (as generated from the tex file)
- Your digital designs: `lab6_part1.circ`,`lab6_part2.circ`

The Lab Demonstration must be completed during the lab session that you are enrolled in. During a lab demonstration, your TA may ask you to: go through parts of your pre-lab, run and simulate your designs in Logisim, and answer questions related to the lab. You may not receive outside help (e.g., from your partner) when asked a question.

6.2 Marking Scheme

Each lab is worth 4% of your final grade, where you will be graded out of 4 marks for this lab, as follows.

- Prelab: 1 mark
- Part I (in-lab): 1 mark
- Part II (in-lab): 2 mark
- Part III (in-lab): 1 mark (bonus)

6.3 Lab Preparation

6.3.1 Part I

In this part, you design a Moore finite state machine for Robo-Snail. Robo-Snail smiles (i.e., outputs 1) if it recognizes two specific sequences of inputs: 1111 **or** 1101. Given an input `W` and an output `Z`, `Z` is 1 when: (1) `W` is 1 for four consecutive clock edges. Or, (2) when the most recent sequence on `W` was 1101 for the last four clock edges. In all other cases, the output `Z` is 0 (i.e., Robo-Snail is not smiling).

Note that overlapping sequences are allowed. For example, if `W` is 1 for five consecutive clock pulses, then `Z` is 1 after both the fourth and fifth clock edge. A state diagram for this FSM's behaviour is shown in Figure 6.1.

In the starter Logisim file, the two modules *part1_FSM* and *part1_state_table* provide a starting point for implementing the required state machine:

- *part1_FSM* is the high-level FSM circuit that has the flip-flops needed to implement the states and a module for the state logic. A module for the output logic is omitted and must be filled in by you.
- *part1_state_table* is the module for the state logic that you must complete.

Study and understand these modules as they provide a model for how to clearly describe a finite state machine that will both simulate and synthesize properly.

Perform the following steps:

1. Derive the state transition table from the diagram in Figure 6.1.
2. Decide on state encodings. **Constraint:** You may not use a simple binary encoding (i.e., A is 000, B is 001, and so on). If you do use a binary encoding, then the order should be different. Otherwise, consider using one-hot or one-cold encoding.

   Include the encodings in your report.
3. Re-do the state transition table using the encodings you have decided on.

   Include this table in your report.

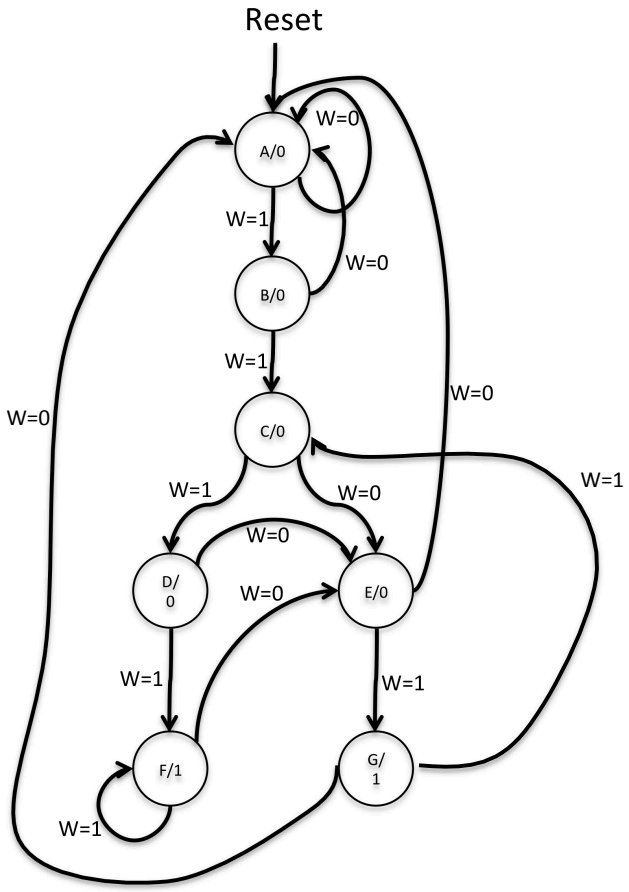

Figure 6.1: The state diagram for Robo-Snail.

4. Using the encoding of `1` for a smile and `0` for no smile, derive the output table.

   Include this table in your report.

5. Derive Boolean equations for your next state logic and output logic.

   Include these equations in your report.

6. Fill in the rest of the circuit in the *part1_state_table* module to implement the state table you derived in Step 2. This module will implement the *state logic* (the combinational circuit that determines what the new flip-flop values should be, based on the previous flip-flop values and the input w).

   Export the subcircuit schematic as an image and include it in your report.

7. Implement the output value circuit for z in *part1_FSM*.

   Export the subcircuit schematic as an image and include it in your report.

6.4 Part II

Processor circuits can be separated into two main components:

1. The *datapath* that connects data storage structures (registers) to processing units (the ALU).

2. The *control path* that operates the datapath signals to determine what data values flow through the datapath and what operations are performed on this data (a finite state machine).

In previous labs, you constructed a simple ALU. In Part I of this lab you constructed a simple *finite state machine* (FSM), which is the most common component used to implement the control path. To complete this part of the lab, you will implement an FSM to control a datapath that performs a common calculation. This is an important step towards building a microprocessor as well as any other computing circuit.

The datapath you will be controlling is provided as part of the starter circuit. A FSM is provided as well, which operates on this datapath to compute $2A^2 + C$. Your task is to create a **different** FSM that controls the datapath provided to perform the following computation:

$$Cx^2 + Bx + A$$

The values of x, A, B and C will be preloaded by the user on the switches before the computation begins.

Figure 6.2 shows the block diagram of the datapath you will be using. There are a few things to note about this diagram:

- Reset signals are not shown to reduce clutter, but make sure to include them when building your circuit in Logisim.

- The datapath will operate on 8-bit unsigned values. Assume that the input values are small enough to not cause any overflows at any point in the computation, i.e., no results will exceed $2^8 - 1 = 255$.

- The ALU only needs to perform addition and multiplication, but you're welcome to use a variation of the ALU you built in previous labs in order to have more operations available for solving other equations (in case you wish to try some things on your own).

- There are four registers R_x, R_A, R_B and R_C that will be loaded at the start with the values of x, A, B and C, respectively. The registers R_A and R_B can be overwritten during the computation.

- There is one output register, R_R, that captures the output of the ALU and displays its value both in binary on the LEDs and in hexadecimal on the HEX displays.

- Two 8-bit-wide, 4-to-1 multiplexers at the inputs to the ALU are used to select which register values are input to the ALU.

- All registers have enable signals to determine when they load new values. They also have an active low asynchronous reset to clear their contents to zero.

The provided circuit should operate in the following manner:

- After an active low asynchronous reset, you will use the *data_in* bits to load the following values on the first four clock cycles:

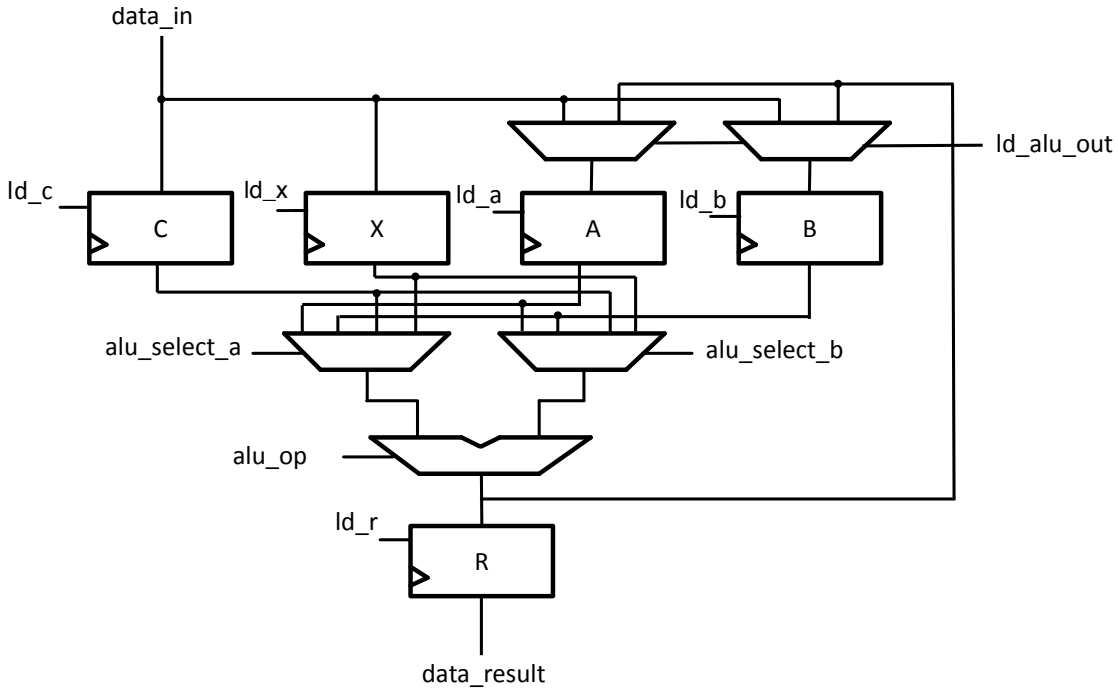

Figure 6.2: Block diagram of datapath.

  - On the first clock cycle, load the value for R_A.
  - On the second clock cycle, load the value for R_B.
  - On the third clock cycle, load the value for R_C.
  - On the fourth clock cycle, load the value for R_X.
- Computation will start after all values are loaded.
- When computation is finished, the final result will be loaded into R_R.
- This final result should be displayed on LEDs in binary and HEX displays in hexadecimal.

Perform the following steps for this part:

1. Examine the provided starter circuits (all the modules for this part start with `part_2`), which is available on Quercus). This is a major step in this part of the lab. You will not need to build the datapath yourself, but you will need to fully understand the datapath embodied by the starter circuit to be able to make your modifications.

2. Determine a sequence of steps similar to the datapath example shown in lecture to control the datapath to perform the required computation. You should draw a table that shows the contents of the registers and the control signal values for each cycle of your computation. Include this table in your prelab.

3. Draw a state diagram for your controller starting with the register load states provided in the example FSM. Include the state diagram in your prelab.

4. Modify the provided FSM to implement your controller and synthesize it. You should only modify the control module, not the datapath. Submit your modified circuit in the prelab.

5. Test your modules with *Poke* to verify its correctness. Include a few screenshots that shows the simulation output.

Again, note that there are multiple ways to implement a FSM in Logisim. Explore these and find the approach that works best for you.

It is fine if your implementation does not use all of the modules provided, but that doesn't mean you should have your entire circuit in one giant module! We will start evaluating the elegance and readability of your design in these later labs, so find ways to divide your solution into smaller modules when possible/sensible.

6.5 Part III (Optional)

Note: Only start working on this part if you already completed other parts. This is an optional part that provides a more challenging exercise for you to further test your knowledge.

Addition, subtraction and multiplication are much easier to build in hardware than division. Division is a much more complex operation to implement in hardware. For this part, you will design a 4-bit restoring divider using a finite state machine.

Figure 6.3 shows an example of how the restoring divider works. This mimics what you do when you do long division by hand. In this specific example, number 7 (*Dividend*) is divided by number 3 (*Divisor*). The restoring divider starts with *Register A* set to *0*. The *Dividend* is shifted left and the bit shifted out of the left most bit of the *Dividend* (called the most significant bit or MSB) is shifted into the least significant bit (LSB) of *Register A* as shown in Figure 6.4.

The *Divisor* is then subtracted from *Register A*. This is equivalent to adding the 2's complement of the *Divisor* (11101 for the example in Figure 6.3) to *Register A*. If the MSB of *Register A* is a *1*, then we restore *Register A* back to its original value by adding the *Divisor* back to *Register A*, and set the LSB of the *Dividend* to *0*. Else, we do not perform the restoring 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 the *Dividend* register is the *Quotient*, and *Register A* will hold the value of the *Remainder*.

Structure your code in the same way as you were shown in Part II and follow these steps for this part:

1. Draw a schematic for the datapath of your circuit. It will be similar to Figure 6.4. You should show how you will initialize the registers, where the outputs are connected to, and include all the control signals that you require.

2. Draw the state diagram that controls your datapath.

3. Draw the schematic for your controller module.

4. Draw the top-level schematic showing how the datapath and controller are connected as well as the inputs and outputs to your top-level circuit.

5. Build your circuit in Logisim.

6. Simulate your circuit using a variety of input settings.

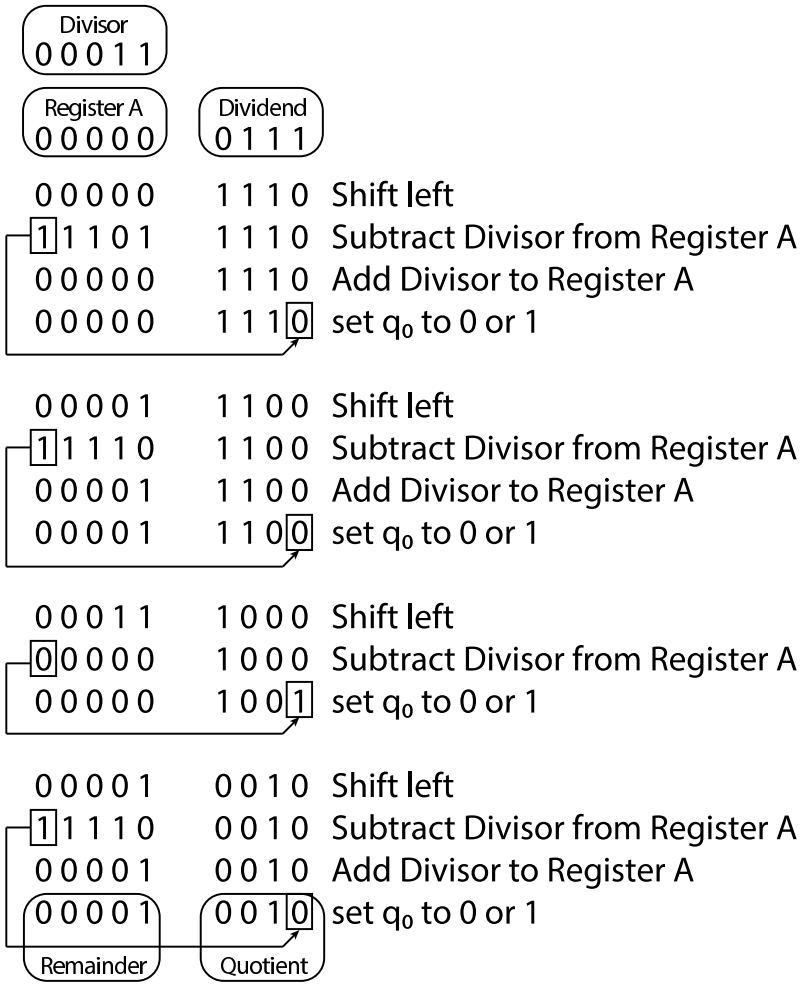

Figure 6.3: An example of how a restoring divider works.

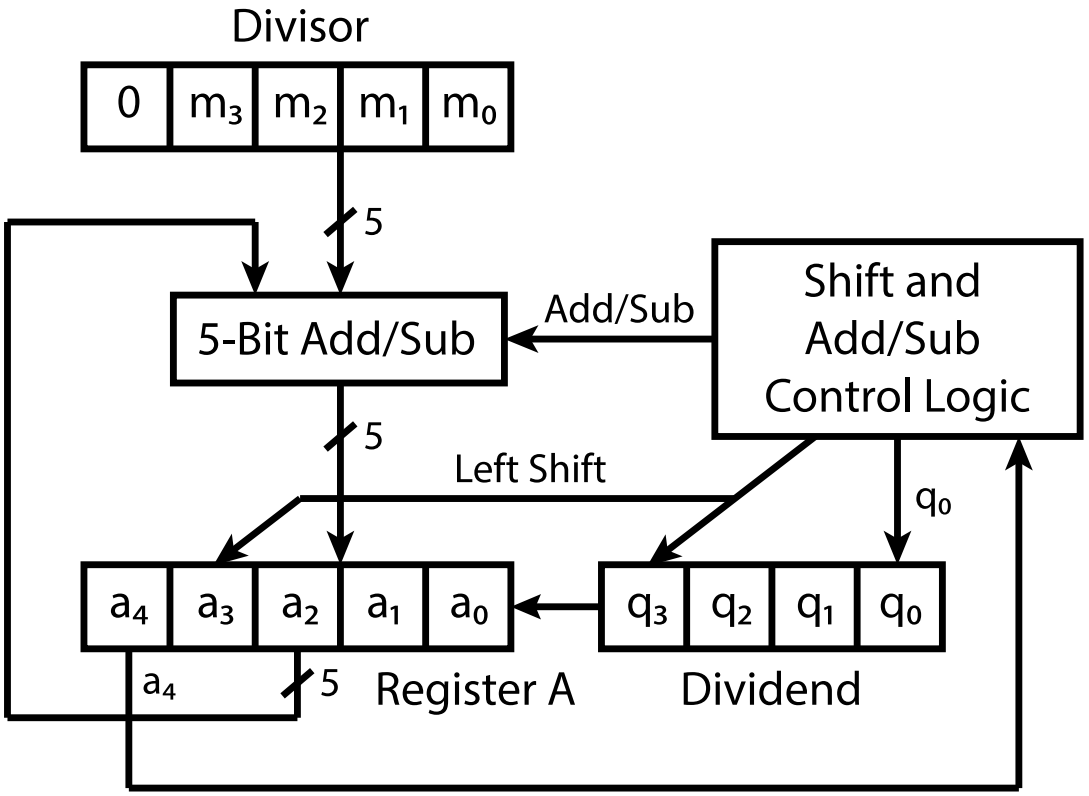

Figure 6.4: Block diagram of restoring divider.

6.6 Lab Demonstration

6.6.1 Part I

Discuss with your partner the encoding you used for your Finite State Machine. Make sure you understand how different encodings could impact the next state or output logic. Choose one of your designs to use for the demonstration.

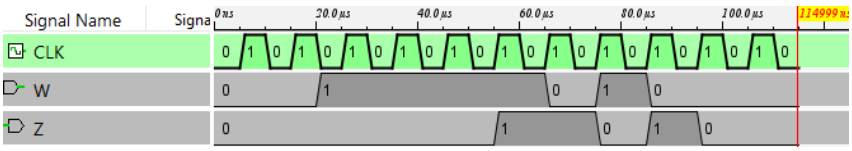

Figure 6.5: An example timing diagram for Robo-Snail.

Check that you can reproduce the timing diagram in Figure 6.5. When you are ready, demonstrate your final designs to your TA.

6.6.2 Part II

Discuss with your partner your analysis of the datapath and control unit. **Make sure you understand the main circuit and, if asked, can generate a timing diagram from scratch.** When you are ready, demonstrate your understanding to your TA.

6.6.3 Part III (optional)

Make sure you understand the main divider circuit and, if asked, can explain its behavior. When you are ready, demonstrate your understanding to your TA.