Lab 4: Finite State Machines

CSC258H1 – Computer Organization

This document describes what you need to prepare and demonstrate for Lab 4. Section 2 describes the tasks you must complete *before* your lab session. Section 3 describes the tasks you complete *during* your lab session. The next section describes lab logistics in more detail.

Contents

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.

Begin the pre-lab by following the steps in Section 2. Remember to **download the starter files**. The `.circ` starter files are typically pre-populated with the names of subcircuits and pins for you. **You should not change the names of pins or the order of inputs and outputs.**

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 3's pre-lab (Section 2) are: `lab4_report.pdf`, `lab4_binary.circ`, and `lab4_one_hot.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 Evolution, and answer questions related to the lab. You may not receive outside help (e.g., from your partner) when asked a question.

2 Lab Preparation

By the end of this pre-lab, you should be able to:

- Design a sequence recognizer using a finite state machine.
- Evaluate finite state machines that have the same behaviour but use different encodings in terms of area and delay.

In each pre-lab part, you design a Moore finite state machine for Robo-Snail II. Robo-Snail II 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 II 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 1.

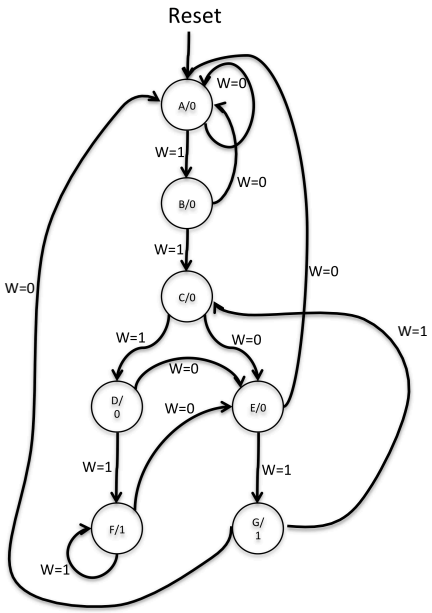


Figure 1: The state diagram for Robo-Snail II.

Element	Propagation Delay (ps)
NOT	20
AND2	25
AND3	35
AND4	45
OR2	25
OR3	35
OR4	45

Table 1: Logic Gate Delays

2.1 Part I

In this part, your states will be encoded using a simple binary encoding. That is, state `A` is 0, `B` is 1, and so forth. You must use the minimum number of bits needed, N, to encode all states. And your schematics are limited to the gates listed in Table 1. Perform the following steps:

1. Derive the state transition table from the diagram in Figure 1 and use a simple binary encoding for the states.
2. Using the encoding of `1` for a smile and `0` for no smile, derive the output table.
3. Derive the minimized Boolean equations for your next state logic and output logic.
4. In the starter file `lab6_binary.circ`, open the sub-circuit called `NextStateLogic`. Design a schematic based on your equations for the next state logic. You should have a 1-bit input, W. You should also have an N-bit input S and an N-bit output Snext.

   Export the subcircuit schematic as an image and include it in your report.
5. In the same file, open the sub-circuit called `OutputLogic`. Design a schematic based on your equation for the output logic. You should have an N-bit input S and a 1-bit output Z.

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

6. Using the `SequenceDetector` sub-circuit, generate a timing diagram. Your timing diagram should include the waveforms for: `CLK, W, StateRegister, Z`. The timing diagram should show that the RoboSnail II correctly smiles for the sequence `1111010`.

   Export the timing diagram as an image and include it in your report.

2.2 Part II

In this part, your states will be encoded using a one-hot encoding. There are seven states, so seven bits are needed. The initial state should be `0000001`, the next should be `0000010`, the next `0000100`, and so on. Note that `0000000` is *not* a valid one-hot encoding. Your schematics are limited to the gates listed in Table 1. Perform the following steps:

1. Derive the state transition table from the diagram in Figure 1 and use a one-hot encoding for the states.
2. Using the encoding of `1` for a smile and `0` for no smile, derive the output table.
3. Derive the minimized Boolean equations for your next state logic and output logic.
4. In the starter file `lab6_one_hot.circ`, open the sub-circuit called `StateRegister`. Design a state register that, when reset, rests to the "initial" one-hot state. Otherwise, this state register behaves just like a typical register. Recall that an N-bit register is made up of N flip-flops that share the same clock.

   Hint: you should make use of the S and/or R inputs using Logisim Evolution's built-in D Flip-Flop.
5. In the same file, open the sub-circuit called `NextStateLogic`. Design a schematic based on your equations for the next state logic. You should have a 1-bit input, W. You should also have an N-bit input S and an N-bit output Snext.

   Export the subcircuit schematic as an image and include it in your report.
6. In the same file, open the sub-circuit called `OutputLogic`. Design a schematic based on your equation for the output logic. You should have an N-bit input S and a 1-bit output Z.

   Export the subcircuit schematic as an image and include it in your report.
7. Using the `SequenceDetector` sub-circuit, generate a timing diagram. Your timing diagram should include the waveforms for: `CLK, W, Z` and `Q` from the `StateRegister` sub-circuit. The timing diagram should show that the RoboSnail II correctly smiles for the sequence `1111010`.

   Export the timing diagram as an image and include it in your report.

2.3 Part III

In this part, you compare your designs from Sections 2.2 and 2.3. Perform the following steps:

1. Calculate the the propagation delay of your `NextStateLogic` and `OutputLogic` sub-circuits from Section 2.1. Use the delays given in Table 1 and explicitly describe (or use a figure to show) the critical path. Show your work and the final result in your report.
2. Calculate the the propagation delay of your `NextStateLogic` and `OutputLogic` sub-circuits from Section 2.2. Use the delays given in Table 1 and explicitly describe (or use a figure to show) the critical path. Show your work and the final result in your report.
3. Compare the area of your FSM from Section 2.1 and 2.2.

   You do not need to exactly calculate the area of each design, but reasoning about the number of transistors, gate cost, and/or gate cost + inverters may be useful. Complete marks are reserved for a correct and concise comparison that considers all relevant aspects of the designs. *Hint: don't forget about your state registers.*

3 Lab Demonstration

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 unit 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 this demonstration, you implement a calculator for the greatest common divisor.[1] By the end of this lab demonstration, you should be able to:

- Analyse a given "Datapath".
- Design a "Control Unit"; that is, a finite state machine that controls another circuit.

When demonstrating a part to your TA, be ready to answer questions **individually**.

[1] Based on Pedroni, Volnei A. Finite state machines in hardware: theory and design (with VHDL and SystemVerilog). MIT press, 2013.

3.1 Part I

In this part, we analyse the design of a datapath. Open the starter file: `lab4_demo.circ` and analyse the sub-circuits `ArithmeticUnit` and `Datapath`. Perform the following steps:

1. Describe the behaviour of the `ArithmeticUnit` circuit by writing out a truth table for every combination of `Op` and describing `Result` in terms of `A` and/or `B`. Then describe `Sign` in terms of `Result` (e.g., $Result = 0$, $Result > 0$, etc.)

Op	Description of Result	Sign	Description of Sign w.r.t. Result
00		00	
01		01	
10		10	
11		11	N/A

2. Generate a timing diagram using `Datapath` that computes $12 - 4$ when $A = 12$ and $B = 4$. You must only use the input pins (i.e., do not poke the registers) and use the minimal amount of rising clock edges. Export the timing diagram and save it for your demonstration.

3. Generate a timing diagram using `Datapath` that computes $12 - 4$ when $A = 4$ and $B = 12$. You must only use the input pins (i.e., do not poke the registers) and use the minimal amount of rising clock edges. Export the timing diagram and save it for your demonstration.

Make sure you understand the datapath and how to use it in Logisim Evolution before demonstrating to your TA.

3.2 Part II

Design a Moore FSM to control the datapath - see subcircuit `ControlUnit` for the FSM template. You will need to sketch schematics for `NextStateLogic` and `OutputLogic` to implement the Greatest Common Divisor (GCD) algorithm:

$$GCD(A, B) = \begin{cases} A & A = B \\ GCD(A - B, B) & A > B \\ GCD(A, B - A) & B > A \end{cases}$$

Before you can begin computing the GCD, however, you must load values into the datapath's registers. The general flow chart is shown in Figure 2. A "Ready" signal indicates to the control unit that inputs A and B should be stored into the registers. Careful, the registers for A and B are in the datapath (they are not state registers).

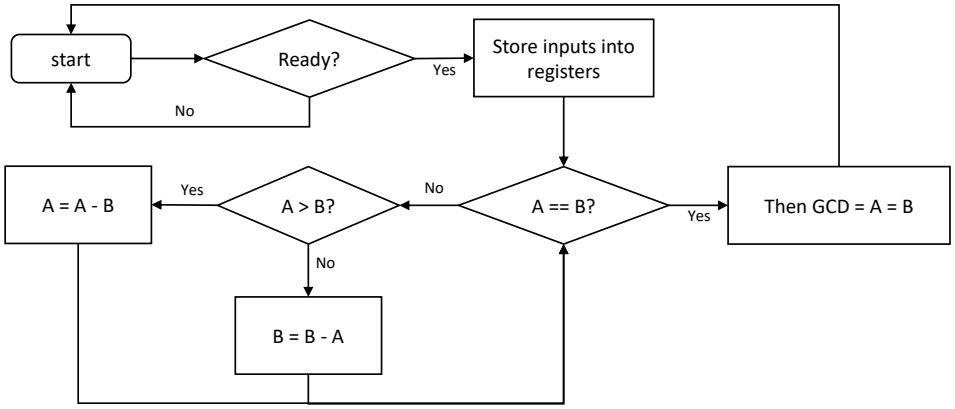


Figure 2: A flowchart showing the GCD algorithm, including steps to load the data.

Once the data is saved, the algorithm is applied until the greatest common divisor is found. Then, the user can store new data by indicating the Ready signal again.

A timing diagram of the expected behaviour is shown in Figure 3. The encodings shown in `StateRegister` are only one example; you may use a different encoding if you wish. But pay close attention to the transitions between states and how that impacts the values stored in the datapath's registers. In particular, in which states do the registers change in value and in which states do they not? Inspect the `GreatestCommonDivisor` subcircuit to see how output from the datapath is used as input to the control unit. Why?

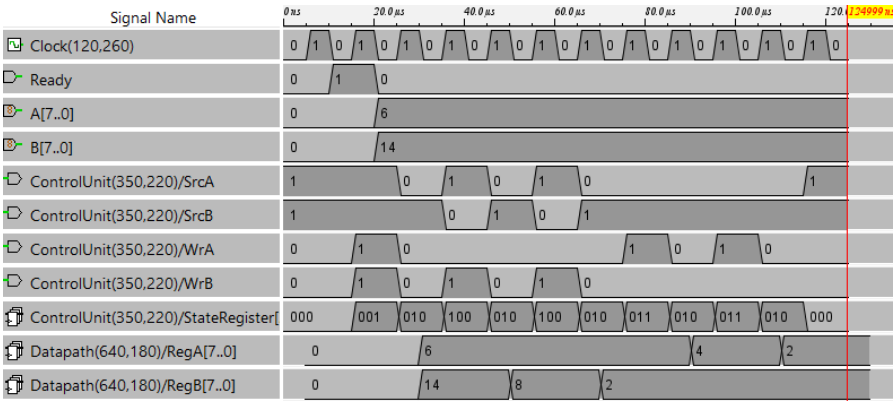
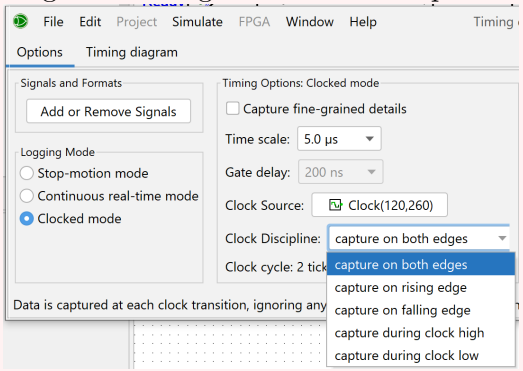


Figure 3: A timing diagram of the expected behaviour for `GreatestCommonDivisor`.

CAUTION

Do to a visualization bug with timing diagrams in Logisim Evolution, our timing diagram is using the "Clock Discipline" setting of "capture on both edges".



Ideally, we would use "capture on rising edge" to be consistent with what we have learned, but then we don't see the clock toggling in the timing diagram. Because we are using "capture on both edges", you will notice that the datapath registers in Figure 3 are changing on the falling edge. You do not need to know the mechanics behind this.

Perform the following steps:

1. Draw a state transition diagram, using Figures 2 and 3 as a guide. See if you can come up with useful English names for each state.
2. Based on your state transition diagram, derive the state transition table and output table.
3. Derive the next state and output equations from the previous tables.
4. Design the schematics in Logisim Evolution.
5. Test your `NextStateLogic`, `OutputLogic`, and `ControlUnit`.

Make sure you understand the control unit, how it controls the datapath, and how to use it in Logisim Evolution before demonstrating to your TA.

