Lab 5: Counters and Clocks

CSC258H1 – Computer Organization

This document describes what you need to prepare and demonstrate for Lab 5. 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: `lab5_report.pdf`, `lab5_counter.circ`, and `lab5_clock.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 counter from T Flip-flops using hierarchical design.
- Design a "rate divider" using a counter.
- Represent some Morse code symbols in a specific binary format.

2.1 Part I

In this part, you design a counter using only T flip-flops. A T flip-flop does not have a D input but rather, as the name implies, a T input. If T is 1, then the bit stored in the flip-flop *toggles*. For example, if the bit stored was 0, then it toggles to 1. And if the bit stored was 1, then it toggles to 0.

Consider the 4-bit synchronous counter that uses four T flip-flops in the `incrementer4` subcircuit provided in the starter files. The circuit has three 1-bit inputs: `CountEnable, Clock, Clear` and one 4-bit output: `Q`. The `Clear` input resets all flip-flops to 0 asynchronously. When `CountEnable` is HIGH, `Q` will increase by 1 on each rising clock edge. When `CountEnable` is LOW, the counter is paused (i.e., retains its state across rising clock edges). Simulate the circuit to understand both how the circuit is incrementing `Q` by one and when any one of the flip flops should toggle.

Perform the following steps:

1. In the starter file called `lab5_counter.circ`, open the subcircuit called `counter2`. Implement a 2-bit counter using T flip-flops. You should have the same inputs and outputs as `incrementer4`, but for 2 bits rather than 4 bits.
2. Open the subcircuit called `counter4`. Using *Hierarchical Design*, implement a 4-bit counter using two 2-bit counters (i.e., `counter2`).

   Hint: Think carefully about how to connect the two counters. You may need one or more gates.
3. Simulate the `counter4` circuit using `Simulate -> Timing Diagram`. Make sure you `Reset Simulation` before starting. Demonstrate that the circuit can start at 0 and increment on each rising clock edge until it becomes 0 again.

   Export the timing diagram as an image and include it in your report.
4. Open the subcircuit called `counter8`. Using *Hierarchical Design*, implement an 8-bit counter using two 4-bit counters (i.e., `counter4`). *Hint:* Think carefully about how to connect the two counters. You may need one or more gates.

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

2.2 Part II

Typically, a digital system has one global clock. However, sometimes it is desirable to have some circuits operate at a slower speed than the global clock. It is useful to design a subcircuit, called a "rate divider", that helps you run other circuits at a rate that is slower than the clock. Remember, however, that a fully synchronous circuit is one where every flip-flop in your circuit is connected to the *same* clock. So the output of the rate divider should **not** connect to the clock input of another circuit.

In this part, you design a rate divider that produces a new "clock" (actually an enable) signal. **In this part, we assume that the global clock is 32Hz** (remember to set this in Logisim Evolution by navigating to: `Simulate` then `Auto-Tick Frequency`). This new clock signal is slower than the global clock, and can be used to drive the *enable* inputs of other circuits that need to operate at a slower rate. One of the rate divider's inputs, `Rate`, dictates how much slower the new clock signal will be (see Table 1).

$Rate_1$	$Rate_0$	Speed
0	0	Half (16 Hz)
0	1	1 Hz
1	0	0.5 Hz
1	1	0.25 Hz

Table 1: The supported frequencies.

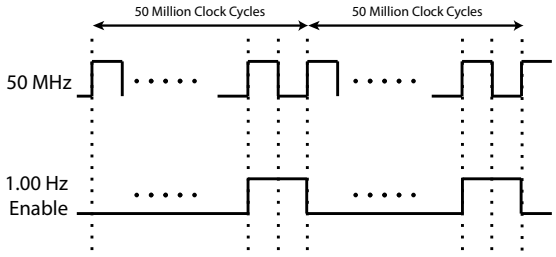

Figure 1: Timing of a 1 Hertz enable signal.

Figure 1 shows a timing diagram that produces a 1 Hz pulse signal from a 50 MHz clock (the kind used on the DE1-SOC board in the lab rooms). The resulting 1 Hz pulse signal would provide the Enable value for, for example, a counter. A common way to provide the ability to change the number of pulses counted is to parallel load the counter with the appropriate value and count down to zero. For example, if you want to count 50 million clock cycles, load the counter with 50 million - 1 (49,999,999). Then subtract one from it until it reaches 0. After that reload 49,999,999.

Perform the following steps:

1. In the file `lab5_clock.circ`, open the subcircuit called `RateDivider`. Design a circuit that has 1-bit inputs, `Enable, Clock`, a 2-bit input, `Rate`, and a 1-bit output `DivOut`. Using a built-in 8-bit counter, implement a Rate Divider as described above.

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

2. Simulate the your rate divider using `Simulate -> Timing Diagram` and include waveforms for `Rate, Enable, CLK, DivOut` and your counter. You should display the counter waveform's values as "Unsigned Decimal". Make sure you `Reset Simulation` before starting.

   Demonstrate that the circuit's `DivOut` produces a "clock", but at a rate of 1 Hz (e.g., Figure 1). Begin by ensuring `Enable` is TRUE and toggling the clock once; this should load the appropriate value (according to `Rate`) into the counter. Then toggle the clock, observing `DivOut` in relation to `CLK`.

   Export the timing diagram as an image and include it in your report. The timing diagram may be 'clipped' in that it does not show the very beginning of time. But it should show that `DivOut` is TRUE at the right time along with the internal values inside your counter.

The circuit `RateDividerChecker` can help you see if one counter is progressing slower than another. However, there is no built-in rate divider in Logisim Evolution. So there is no "ground truth" to compare against in the circuit simulation.

2.3 Part III

Morse code uses patterns of short and long pulses to represent a message. Each letter is represented as a sequence of dots (a short pulse), and dashes (a long pulse). For example, the last 8 letters of the English alphabet have the following representation:

S	• • •
T	—
U	• • —
V	• • • —
W	• — —
X	— • • —
Y	— • — —
Z	— — • •

Let us first look into how we store the Morse code representation for each letter. Suppose all the times for dot, dash and pause are multiples of 0.5 seconds. We can use this to our advantage and represent each Morse code as a sequence of bits. Each bit corresponds to a display duration of (e.g., an LED turning on for) 0.5 seconds. Therefore a single 1 bit will correspond to a dot (the LED stays on for 0.5 seconds), while three 1s in a row corresponds to a dash (the LED should stay on for 3×0.5 seconds). In order to differentiate between a dot and a dash, or between multiple successive dots or multiple successive dashes, we will inject zeros between them (*i.e.*, the LED stays off for 0.5 seconds – the time required between pulses).

An LED that is off signifies either a pause (e.g., a transition between a Morse dash and a dot), the end of a transmission, or no transmission. Using this representation, the Morse code for letter X would be stored as `1110101011100000`, assuming we use 16-bits to represent it. Observe that a different number of bits is needed for each letter. For letters that do not require the maximum number of bits, you may simply set the last few bits to 0.

Perform the following steps:

1. Write the Morse code binary representation of all letters specified above (S to Z) in a table following the same approach used for X. You must also decide on the maximum number of bits needed when accounting for all letters (S to Z). The last bit of any Morse code representation should be 0.

   Fill in a table with your binary representation of each letter from S to Z. Include this table in your report.

3 Lab Demonstration

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

- Demonstrate two counters incrementing values at different rates on the DE1-SoC board.
- Design a circuit that flashes Morse code letters through an LED.

3.1 Part I

In `lab5_clock.circ`, merge in a previous lab that includes your `HexDecoder` sub-circuit (and its associated sub-circuits). Then, create a new sub-circuit similar to the `RateDividerChecker` with the goal of synthesising the design to the DE1-SoC board. So you will need output pins for two different seven segment displays, a reset button, and switches for the enable and the rate.

Follow the normal steps for synthesis (recall Lab 2), with an added step: Make sure you set the frequency in the `Synthesize and Download > Clock Settings` to 32 Hz. The "Divider Value" will be set on its own. This divider value will divide the board's clock, which is 50 MHz, by a value to output the 32 Hz clock for your design.

3.2 Part II

Open `lab5_clock.circ` and perform the following steps:

1. Create a new subcircuit in `lab5.circ` called `MorseLUT`. Implement a Look-up Table (LUT) that has a 3-bit input, `Char` to select a letter (i.e., one of S to Z; let 000 correspond to S, 001 correspond to T, and so forth up to 111 for Z). The output, `MorseCode`, should be a multi-bit output whose width is what you decided on in the pre-lab. Based on the input `Char`, `MorseCode` outputs the binary representation for the specified letter. You can use `Constant`s in Logisim to store these output values.

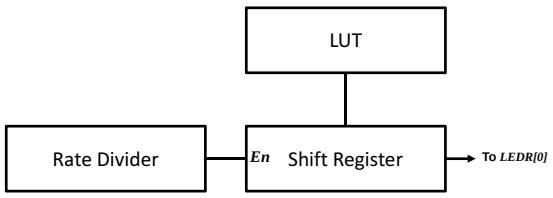

Figure 2: A high-level block diagram of the Morse code circuit.

2. Create a new subcircuit called `MorseCode`. It should include: two 1-bit inputs `LoadEnable` and `Enable`, a 3-bit input `CharSelect`, and a clock. Add a `MorseLUT` and `rate_divider` module, as well as a shift register from Logisim Evolution (you can find it under `Memory`). You may find Figure 2 helpful.

   Configure the shift register to match the bit width of your `MorseLUT`'s output. Also, **configure the rate divider** to so that the shift register shifts every 0.5 seconds for a 32Hz clock. The output of the circuit should be a single LED that flashes the morse code being selected by `CharSelect`.

When you are ready, demonstrate the simulation to your TA.

