LAB 4

SYNCHRONOUS SEQUENTIAL CIRCUITS

In Lab 4, you work with your first synchronous sequential circuits. First, you design and implement gated D latches, and use that to design and implement D- flip flops. For part II, you will extend the ALU from Lab 3 to support 8 operations. Then, you connect the ALU to an 8-bit register and use that register to feed back into an input of the ALU. After that, you design and implement 8- and 16-bit shift registers.

This document describes what you need to prepare and demonstrate for Lab 4. Section 4.3 describes the tasks you must complete *before* your lab session. Section 4.5 describes the tasks you complete *during* your lab session. The next section describes lab logistics in more detail.

4.1 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 4.3 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 4's pre-lab (Section 4.3) are:

- Your lab report: `lab4_report.tex`, `lab4_report.pdf` (as generated from the tex file)
- Your digital designs: `lab4_part1.circ`, `lab4_part2.circ`, `lab4_part3.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.

4.2 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): 1 mark
- Part III (in-lab): 1 mark

4.3 Lab Preparation

The pre-lab for Lab 4 consists of two parts. In Part I, you extend the ALU you designed in Lab 3. In Part II, you design and implement one piece of a shift register.

4.4 Part I

Figure 4.1 shows the circuit for a gated D latch. In this part, you will build the gated D latch using the 7400 chips (as in Lab 1) on the protoboard (breadboard). Refer back to the Lab 1 handout for 7400 chips schematics, specifications, and pin-out.

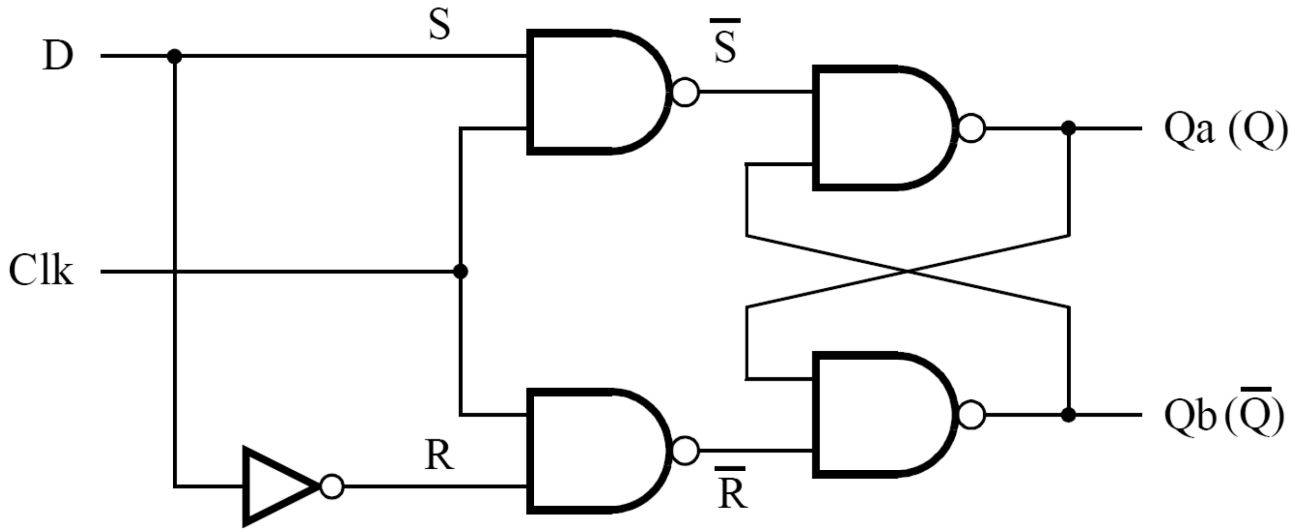

Figure 4.1: Circuit for a gated D latch.

The most common storage element today is the *edge-triggered D flip-flop*. One way to build an edge-triggered D flip-flop is to connect two D latches in series, such that the two D latches use opposite levels of the clock for gating the latch. This is called a master-slave flip-flop, see Fig. 4.2.

The output of the master-slave flip-flop changes on a clock *edge*, unlike the latch, which changes according to the *level* of the clock. For a positive edge-triggered flip-flop, the output changes when the clock edge *rises*, i.e., when clock transitions from 0 to 1.

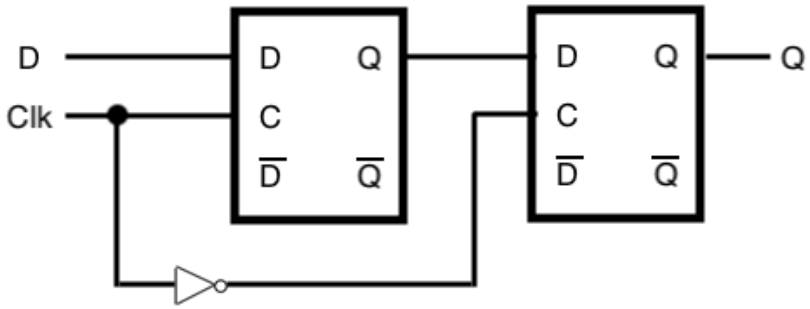

Figure 4.2: Circuit for a master slave flip flop using gated D latches.

Perform the following steps:

1. In Logisim, build this gated D latch from Fig. 4.1 and the master slave flip-flop from Fig. 4.2, each in its own module. The flip-flop should be implemented using the module you create for the D latch, and the latch should make use of interconnected 7400-series chips. For this exercise, you are allowed to use NAND gates. Recall from Lab 1 what a gate-level schematic looks like.

   Some of you have noticed that there is a special Clock signal in the *Wiring* set. You do not need to use that for this part (you will in future labs). For now, just use the default input pin type for the *Clk* input signal (the one that you've been using up to this point).

2. In your pre-lab, include a schematic of the gated D latch using interconnected 7400-series chips.

3. Study the behavior of the latch for different D and clock (Clk) settings.

4. For the D latches and the flip flops, are there any input combinations of Clk and D that should NOT be the first you test with the *Poke* tool? Explain this in your prelab and list them if applicable.

4.4.1 Part IIa

In this part, you extend the ALU you designed in Lab 3 so that it modifies one operation and supports two new operations: operations 6 and 7 (Table 4.1). **Notice that operation 5 has changed**. This brings the total number of supported operations to 8.

Perform the following steps:

1. In a file called `lab4_part2.circ`, use `File -> Merge` and select your `lab3.circ` from last lab. This should import all the subcircuits from `lab3.circ`. **When prompted, you you should say Yes** to overwrite the main in your `lab4_part2.circ` file. The main circuit was completed during the demonstration of Lab 3 - refer to it if you need to refresh your memory.

Table 4.1: Modified and new ALU Operations

Operation	Subcircuit Name	Input(s)	Behaviour
5	op5	`A[4], B[4]`	The output, ALUout, equals B left shifted by A bits (i.e., a "logical left shift"). This operation must use the shifter from Logisim Evolution found under `Arithmetic`.
6	op6	`A[4], B[4]`	The output, ALUout, equals B right shifted by A bits (i.e., a "logical right shift"). This operation must use the shifter from Logisim Evolution found under `Arithmetic`.
7	op7	`A[4], B[4]`	The output, ALUout, equals $A \times B$ (i.e., multiplication). This operation must use the multiplier from Logisim Evolution found under `Arithmetic`.

CAUTION

When using `File -> Merge`, check each subcircuit by hand before continuing. Sometimes, Logisim Evolution does not merge every subcircuit correctly (e.g., if you had edited a subcircuit's appearance, it apparently does not get merged).

Perform the following steps **for each operation** described in Table 4.1:

2. In the same `lab4_part2.circ` file, create (or modify, in the case of operation 5) a subcircuit that implements the behaviour as described in Table 4.1. **Every operation must have one 8-bit output** called `OPout` (inputs may vary, see Table 4.1). You may need to use a `Splitter` and/or `Bit Extender` in Logisim Evolution.

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

3. Simulate your subcircuit using test vectors with *intelligently* chosen values for the input. Take a screenshot of your test vector results and include it in your report.

   Be prepared to justify and explain your test cases during the demonstration.

Two of the operations (5 and 6) involve shifting. When the function asks you to "Shift B by A bits", B contains the value that you are meant to shift. The value in A tells you how much to shift. For example, if `B=0101` and `A=0001`, a right shift would result in `00000010`. In contrast, a left shift would result in `00001010`.

Note that, for left and right shifts, you need to *extend* B first before shifting it by whatever value is in A. You should use *sign extension*. For example, if `B=1001` (i.e., -7_{10}), sign extending B will result in `11111001` (also -7_{10}, but now 8 bits long).

When are done implementing the operations, then:

4. Update the `ALU` subcircuit to support operations 6 and 7. Verify that operation 5 works correctly (check the inputs and outputs carefully).

4.4.2 Part IIb

Rather than having an input pin for B in our main circuit, we instead want to use the four least significant bits of `ALUout` to be the input for B. However, doing so would create a cycle in our combinational circuit. To avoid the cycle, we will design a synchronous sequential circuit using an 8-bit register. Figure 4.3 shows our intended circuit at a high level.

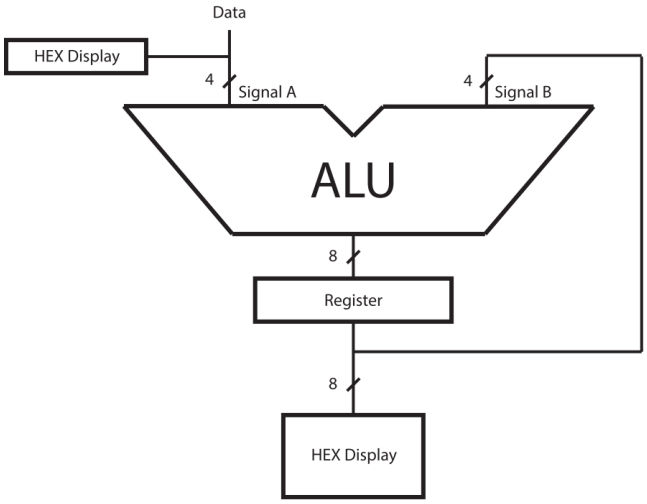

Figure 4.3: The ALU connected to a register.

To create the synchronous sequential circuit, perform the following steps:

1. In the main circuit in file `lab4_part2.circ`, remove the input pin, `B`, and its associated `HEX_DECODER` and 7-segment display.

   Note: This step assumes you have merged in the main circuit from your Lab 3 demonstration, as described in Section 4.4.1.

2. Instead of connecting ALUout directly to the `HEX_DECODER`s for `HEX_ALU1` and `HEX_ALU0`, connect it instead to an 8-bit register called `ALUreg`. The 8-bit register should connect to the decoders for `HEX_ALU1` and `HEX_ALU0`. **You should also connect the 4 least significant bits of the register to the input B of your ALU**.

3. Simulate the main circuit using `Simulate -> Timing Diagram`. Make sure you `Reset Simulation` before starting. Demonstrate the following behaviour: ALUreg starts at 0x00 and increases by 1 on each rising clock edge until it reaches 0x0f. Make sure the values for ALUreg are visible by right-clicking it in the Timing Diagram and selecting `Hexadecimal`.

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

4. Simulate the main circuit using `Simulate -> Timing Diagram`. Make sure you `Reset Simulation` before starting. Use Operation 1 to help you initialize ALUReg to 0x01. Then, **using a shifting operation**, demonstrate the following behaviour: ALUreg goes from 0x01 and doubles on each rising clock edge until it reaches 0x00. Make sure the values for ALUreg are visible by right-clicking it in the Timing Diagram and selecting `Hexadecimal`.

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

4.4.3 Part III

In this part of the lab, you will create an 8-bit shift-register that has an optional arithmetic shift.

A shift register is a row of flip-flops where each flip-flop moves its contents to the next flip-flop in the row on the rising clock edge. Figure 4.4 shows one bit of this shift register. It contains a positive edge-triggered flip-flop that stores the ShifterBit's current value and two multiplexers that determine the source of the ShifterBit's next value.

To create an 8-bit shift-register, you will need eight instances of the circuit in Figure 4.4. Connect these together to design your 8-bit shift-register with optional arithmetic shift and parallel load as shown in Figure 4.5.

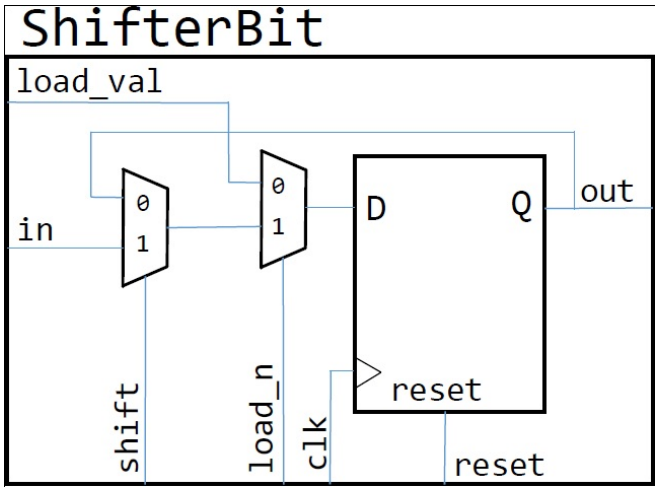

Figure 4.4: Single-bit shift-register

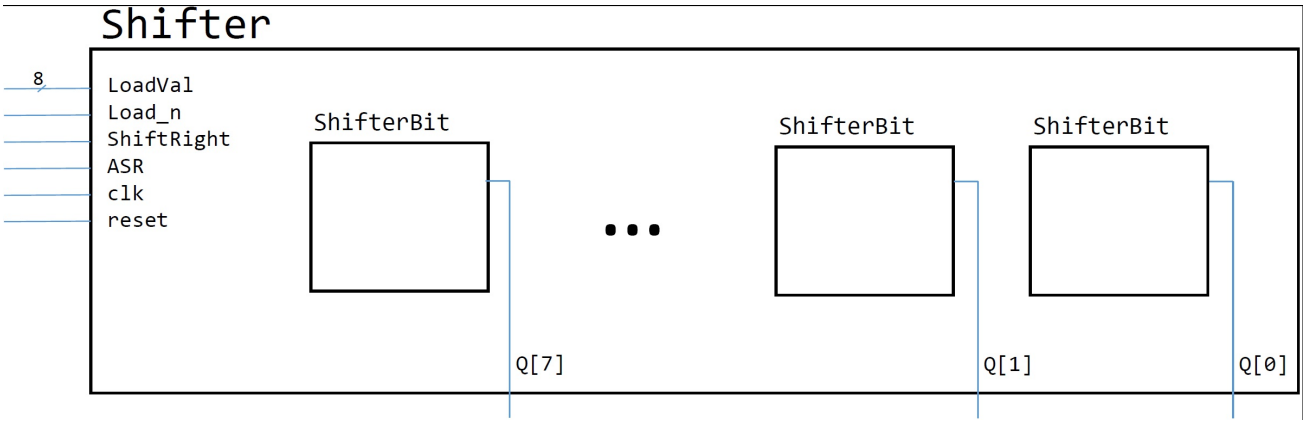

Figure 4.5: 8-bit shift-register of Part III. **None** of internal connections are shown here.

Note: When creating the circuit for the ShifterBit, you may use the D flip flop in *Memory > D Flip Flop*. But you must not use the built-in Shift Register, doing so will earn you 0 marks for this part.

When bits are shifted in the shift register, it means that the bit is copied from the current ShifterBit to the next one on the right. This also implies that the flip-flop in this ShifterBit loads its new value from the flip-flop to its left when the positive clock edge occurs.

What happens to the left-most ShifterBit? When performing a right-shift operation, the flip-flop at the left end of the register has no left neighbour from which to get its new value. So what value gets shifted in? One option is to load a zero, but what if the value in the register is storing a signed value? In this case we should perform *sign-extension*. When we perform the sign-extension, this shift operation is called an *Arithmetic Shift Right* (ASR).

In the Shifter module, create an 8-bit-wide register (i.e. 8 connected ShifterBits) with the following inputs and outputs:

1. An 8-bit input *LoadVal*, whose wires are connected to the `load_val` inputs of each ShifterBit.

2. An 8-bit-wide output *Q*, which is the output of the ShifterBit instances.

3. The *ShiftRight* input which feeds into the *shift* input of all eight instances of the ShifterBit circuit in Figure 4.4.

4. The inputs *load_n*, *clock (clk)*, and *reset*, which feed into the corresponding inputs of each ShifterBit.

5. For each ShifterBit, the *in* port of should be connected to the *out* port of the instance to its left.

For the leftmost ShifterBit, you should design a circuit that will perform sign-extension when the signal *ASR* is high (arithmetic right shift) or load zeros if *ASR* is low (logic right shift). This special circuit is not shown in Figure 4.5.

One thing to note is that the signal *load_n* is *active-low*, meaning that it performs its load operation when its value is 0.

Here is an example of how these signals are used to operate the circuit:

1. When *Load_n = 0*, the value on *LoadVal* is stored in the flip-flops on the next rising clock edge (called parallel load behaviour).

2. When *Load_n = 1*, *ShiftRight = 1* and *ASR = 0*, the bits of the register shift to the right on each positive clock edge. Assuming that the initial value in the flip-flops at cycle 0 is A, with bits A_7 through A_0, the values in the two subsequent cycles would be:

	$Q[7]$	$Q[6]$	$Q[5]$	$Q[4]$	$Q[3]$	$Q[2]$	$Q[1]$	$Q[0]$
Cycle 0:	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
Cycle 1:	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1
Cycle 2:	0	0	A_7	A_6	A_5	A_4	A_3	A_2
...								

3. When *Load_n = 1*, *ShiftRight = 1* and *ASR = 1* the bits of the register shift to the right on each positive clock edge but the most significant bit is replicated. This is called an *Arithmetic shift right*:

	$Q[7]$	$Q[6]$	$Q[5]$	$Q[4]$	$Q[3]$	$Q[2]$	$Q[1]$	$Q[0]$
Cycle 0:	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
Cycle 1:	A_7	A_7	A_6	A_5	A_4	A_3	A_2	A_1
Cycle 2:	A_7	A_7	A_7	A_6	A_5	A_4	A_3	A_2
...								

Do the following steps:

1. What is the behaviour of the 8-bit shift register shown in Figure 4.5 when *Load_n = 1* and *ShiftRight = 0*? Briefly explain in your prelab.

2. Draw a schematic for the 8-bit shift register shown in Figure 4.5 including the necessary connections. Your schematic should contain eight instances of the one-bit shifter (i.e. the ShifterBit) shown in Figure 4.4 and all the wiring required to implement the desired behaviour. Label the signals on your schematic with the same names you will use in your Logisim circuit.

3. In a file called `lab4_part3.circ`, add a subcircuit called `shifter_bit`. Starting with the built-in positive edge-triggered D flip-flop found at *Memory > D Flip Flop*, use this D flip-flop with instances of the *mux2to1* module from Lab 2 to build the one-bit shifter shown in Figure 4.4.

4. In `lab4_part3.circ`, create a new subcircuit called `shift_register`. Build your Logisim module for the shift register that instantiates and connects eight instances of the ShifterBit. This module should match with the schematic in your prelab.

5. Simulate your modules with *Poke*. Choose test cases that make you feel confident about your shifter's correctness, in preparation for your in-lab demo. Make sure to include a few selected screenshots of these cases when you hand in your prelab.

   In your simulation, you should perform the reset operation on the first clock cycle, then do a parallel load of your register on the next cycle. Finally, clock the register for several cycles to demonstrate both types of shifts. *(NOTE: If you do not perform a reset first, your simulation will not work! Try simulating without doing reset first and see what happens. Can you explain the results?)*

4.5 Lab Demonstration

The lab demonstration consists of three parts. Each part corresponds to Parts I, II and III from the pre-lab. In Part III, you implement an 8-bit shift register. When demonstrating a part to your TA, be ready to answer questions **individually**.

4.5.1 Part I

Build the gated D latch and D flip-flop using the chips on the protoboard. Use switches to control the clock (Clk) and D inputs. Use lights (LEDs) to make Qa and Qb visible. Don't forget to hook up the power and ground on all of your chips! Demonstrate your latch and flip-flop implementation to the TA after you have tested it.

4.5.2 Part II

Discuss with your partner the ALU you worked on during the pre-lab (`lab4_part2.circ`). **Make sure you understand the timing diagrams and, if asked, can generate a timing diagram from scratch.** When you are ready, demonstrate your pre-lab designs to your TA.

4.5.3 Part III

Discuss with your partner the 8-bit shift register you worked on during the pre-lab (`lab4_part3.circ`). When you are ready, demonstrate your 8-bit shift registers to your TA.