# CSC258 - Lab 4

**Latches, Flip-flops, and Registers**

## 1 Learning Objectives

The purpose of this lab is to investigate the fundamental synchronous logic elements: latches, flip-flops, and registers.

## 2 Marking Scheme

Each lab is worth 6% of your final grade, which is broken down as follows:

- Prelab + Simulations: 3 marks
- Part I (in-lab): 0.5 marks
- Part II (in-lab): 1.5 marks
- Part III (in-lab): 1 mark

## 3 What You Need To Do

### 3.1 Preparation Before the Lab

As a refresher, carefully review the "Preparation Before the Lab" instructions in the Lab 2 handout.

You are required to complete Parts I to III of this lab by building and testing your circuit in Logisim. Make sure to complete all circuit diagrams and Logisim implementations outlined by the sections in Parts I to III marked as **(PRELAB)**.

As part of this lab (and future labs), you must test your Lab 3 circuits in Logisim. The nature of the circuits in this lab will require you to do most of the testing for your Logisim circuits using *Poke*( ) in the tool bar because the test vectors in Logisim have difficulty testing sequential inputs. Test vectors are still appropriate for the combinational circuits in your design, and the tests you conduct on those circuits should consist of multiple test vectors, sufficient to demonstrate that your design functions as intended. Make sure to submit both the test vector files for your combinational circuits as well as screenshots of your sequential circuit testing.

You are also recommended to find information about *Wiring* in logisim_reference.pdf. Wiring components will be useful for this lab.

### 3.2 In-lab Work

You are required to implement and test Parts I to III of the lab. You need to show your designs and demonstrate the operation of each part to the teaching assistants, including your test vectors.

### 3.3 Summary of Lab Requirements

The following is a summary of what you need to perform for this lab (and labs in general):

- Make sure to include the following in the pre-lab report:
  - Answers to any question posed in the prelab sections
  - Diagrams of your circuit designs (when asked)
  - Any .circ files that you created to implement your designs
  - Test vector files (when possible). Note: test vectors don't work on sequential circuits, but some components (such as your ALU) are combinational circuits that can be tested more effectively with test vectors.
  - Screenshots of your circuit's performance (when test vectors aren't possible). Since you're only meant to include a sample of the possible test cases here, consider which input combinations would best illustrate the behaviour of your circuit.
- What to do in demo sessions:
  - Review your pre-lab report (as necessary)
  - Running and simulating your designs, using the Poke tool and/or test vectors
  - Answer questions about your design and/or the handout
- Files required to upload:
  - Pre-lab report (including circuit diagrams, answers to questions, etc)
  - Logisim files (.circ)
  - Test vectors (.txt)

## 4 Part I

To explore the behaviour of latches, in this lab you will create a latch using *NAND* and *NOT* logic gates. Figure 1 shows the circuit for a gated D latch.

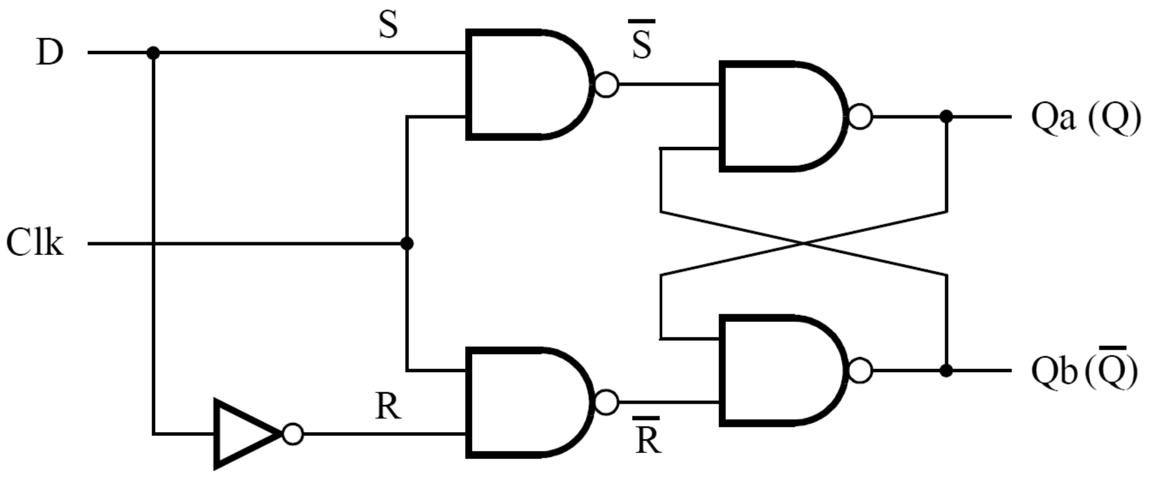

Figure 1: Circuit for a gated D latch.

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. 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.

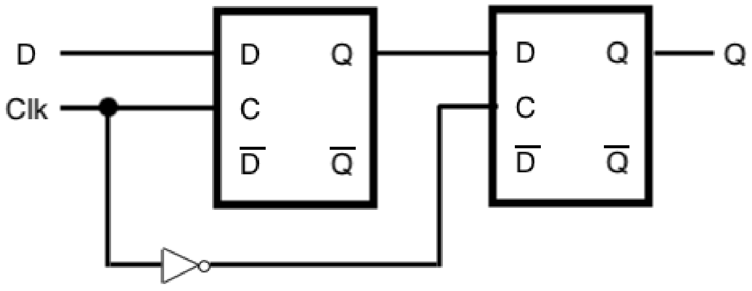

Figure 2: Circuit for a master slave flip flop using gated D latches.

For this part of the lab, you must perform the following steps:

1. In Logisim, build this gated D latch from Fig. 1 and the master slave flip-flop from Fig. 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 the logic gates available in the *Gates* component set. 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). **(PRELAB)**
2. Study the behaviour of the latch for different D and clock (Clk) settings by using *Poke*( ). **(PRELAB)**
3. Try creating a test vector to test this circuit. In your prelab report, describe what happens when you test various input combinations with a test vector. **(PRELAB)**
4. For the D latch and the flip flop, 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.

## 5 Part II

Starting with the circuit you built for Part III of Lab 3, build an ALU that supports the eight operations shown in the table below. The main difference is that the output of the ALU is to be stored in an 8-bit *register* and the four least-significant bits of the register output will feed back to the *B* input of the ALU. Figure 3 shows the required connections.

| Function values | Logic |
|---|---|
| 0 | Make the output equal to A+1, using the adder circuit from Part II of Lab 3. |
| 1 | A + B using the adder from Part II of Lab 3 |
| 2 | A + B using the '+' operator found in *Arithmetic* |
| 3 | A XOR B in the lower four bits, A OR B in the upper four bits |
| 4 | Output 1 (8'b00000001) if any of the 8 bits in either A or B are high.<br>Output 0 (8'b00000000) if all the bits are low (use a reduction OR operator) |
| 5 | Left shift B by A bits<br>Details about the shifter component can be found in<br>`http://www.cburch.com/logisim/docs/2.6.0/en/libs/arith/shifter.html` |
| 6 | Right shift B by A bits (logical right shift) |
| 7 | Output the product of A $\times$ B using the $\times$ operator.<br>Details about the multiplier component can be found in<br>`http://www.cburch.com/logisim/docs/2.6.0/en/libs/arith/multiplier.html` |

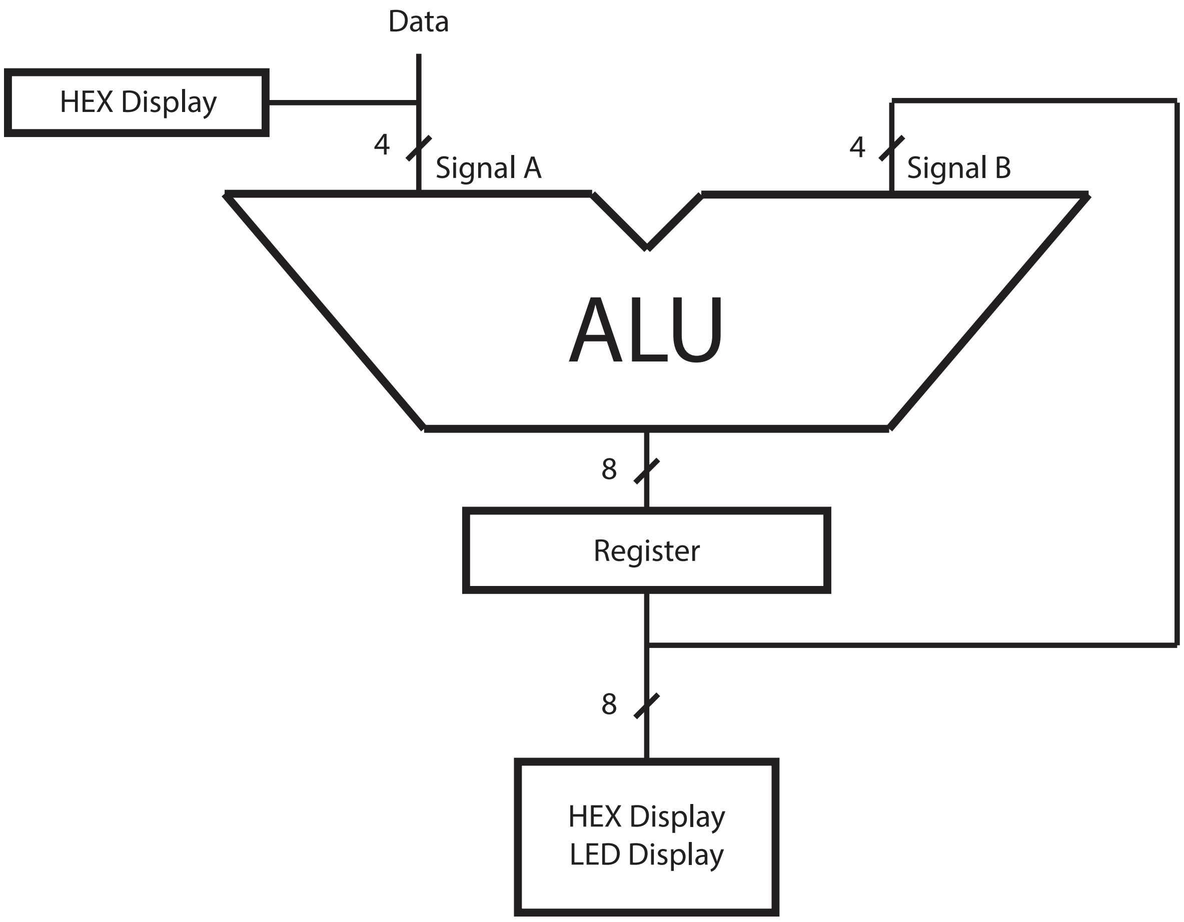

Figure 3: Simple ALU with register circuit for Part II.

One of the new operations here is shifting. When the function asks you to ”Shift B by A bits”, B contains the value that you're meant to shift and the value in A tells you how much to shift. So if B=0101 and A=0001, a right shift would result in 00000010, whereas a left shift would result in 00001010. Note that in each case you need to extend B first to 8 bits before shifting it by whatever value is in A.

Two more things to note about shifting:

- When extending B from 4 bits to 8 bits, you need to maintain its sign (known as ”sign extending”). The Bit Extender in Logisim starts off automatically with Sign as its Extension Type, so you don't need to think about it. But this means that if B is 1001 (-7 in decimal), sign extending B will result in 11111001 (also -7 in decimal, but 8 bits long)
- Values of A that are greater than 7 will shift all the original bits of B out of the result. What will the output be for the left and right shift in the case where A is greater than 7? Include the answer to this in your report. **(PRELAB)**

For this part of the lab, you must perform the following steps.

1. Build the Logisim module for the ALU described above (you are strongly encouraged to extend your design from Lab 3). **(PRELAB)**
2. Include answers to the following questions in your prelab report: **(PRELAB)**
   (a) How would this ALU behave if you didn't include the register in your diagram (i.e. if the ALU output went straight into ALU input B)?
   (b) How would this ALU behave if the register was implemented using clocked latches instead of edge-triggered flip-flops?
   (c) When multiplying two $n$-bit binary numbers, how many bits will you need to store the result?
   (d) (from the shift section above) What will the outputs be for the left and right shift operations when A is greater than 7?
3. Test your modules with *Poke*( ). Choose test cases that make you feel confident about your ALU's correctness in preparation for you in-lab demo.
   (a) Use test vectors to test the operation of your ALU in isolation (i.e. independent of the register and the feedback of the output into the ALU input B). Submit this test vector file. **(PRELAB)**
   (b) When testing the circuit in its entirety, you can't use test vectors. Instead, document your testing of this circuit in your prelab report by providing a list of the test sequences you used for each ALU operation to verify its correctness. **(PRELAB)**
   (c) For the new operations that weren't implemented in Lab 3, include a few select screenshots with your prelab report of test cases that effectively demonstrate that these functions are operating correctly. **(PRELAB)**

## 6 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 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. Connect these together to design your 8-bit shift-register with optional arithmetic shift and parallel load as shown in Figure 5.

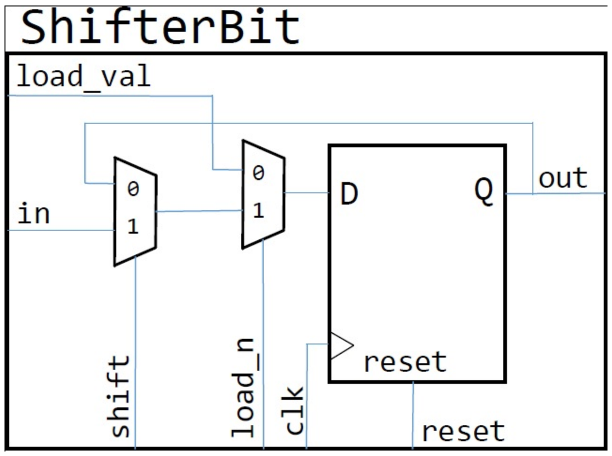

Figure 4: Single-bit shift-register

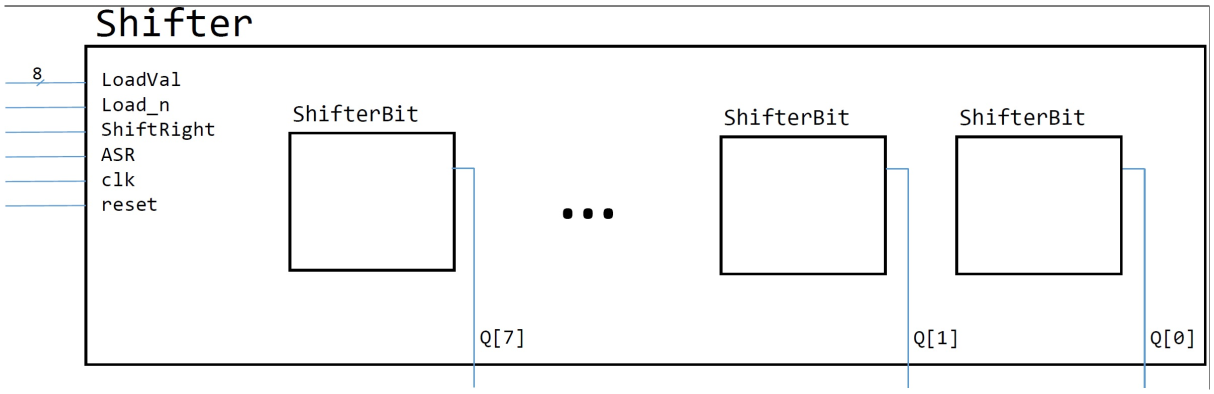

Figure 5: 8-bit shift-register of Part III. **None** of internal connections are shown here.

**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. 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 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 5 when *Load_n = 1* and *ShiftRight = 0*? Briefly explain in your prelab. **(PRELAB)**
2. Draw a schematic for the 8-bit shift register shown in Figure 5 including the necessary connections. Your schematic should contain eight instances of the one-bit shifter (i.e. the ShifterBit) shown in Figure 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. **(PRELAB)**
3. 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. **(PRELAB)**
4. 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. **(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?)* Include one (or a few) screenshot of simulation output in your prelab. **(PRELAB)**