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, but you will be graded out of 8 marks for this lab, as follows.

- Prelab + Simulations: 3 marks
- Part I: 1 mark
- Part II: 2 mark
- Part III: 2 marks

3 Preparation Before the Lab

You are required to complete the prelab for Part I of the lab as you would have prepared for Lab 1. Parts II and III of the lab require you to build and test your Logisim modules. Include your schematics, Logisim modules, and simulations (where applicable) for Parts I to III in the prelab. 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 Logisim has difficulty testing sequential inputs. Test vectors are still appropriate for the combinational circuits in your design.

You are required to implement and test all of Parts I to III of the lab. You need to demonstrate all parts to TAs after you tested them yourselves.

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.

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.

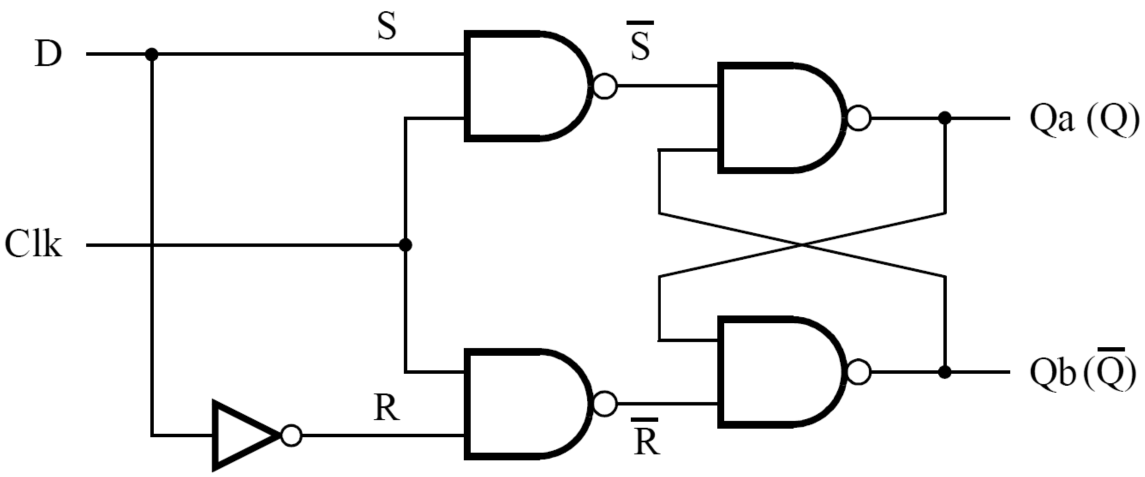

Figure 1: Circuit for a gated D latch.

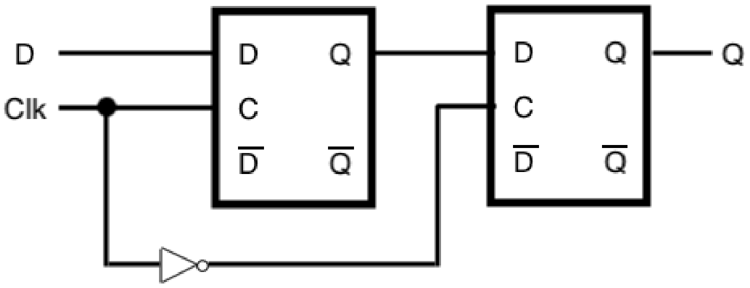

Figure 2: Circuit for a master slave flip flop using gated D latches.

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. Note that for this part you should still use the default input type for signal *Clk* (not the special clock signal in the *Wiring* set). **(PRELAB)**

2. Study the behaviour of the latch for different D and clock (Clk) settings by using *Poke*(👆). **(PRELAB)**

3. 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? Explain this in your prelab and list them if applicable.

5 Part II

Starting with the circuit you built for Lab 3 Part III, build an ALU that supports the eight operations shown in the table below. The output of the ALU is to be stored in an 8-bit *register* and the four least-significant bits of the register output are to be connected 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, and 0 (8'b00000000) if all the bits are low (use a reduction OR operator)
5	Left shift B by A bits Details about the shifter component can be found in `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	A × B using the × operator Details about the multiplier component can be found in `http://www.cburch.com/logisim/docs/2.6.0/en/libs/arith/multiplier.html`

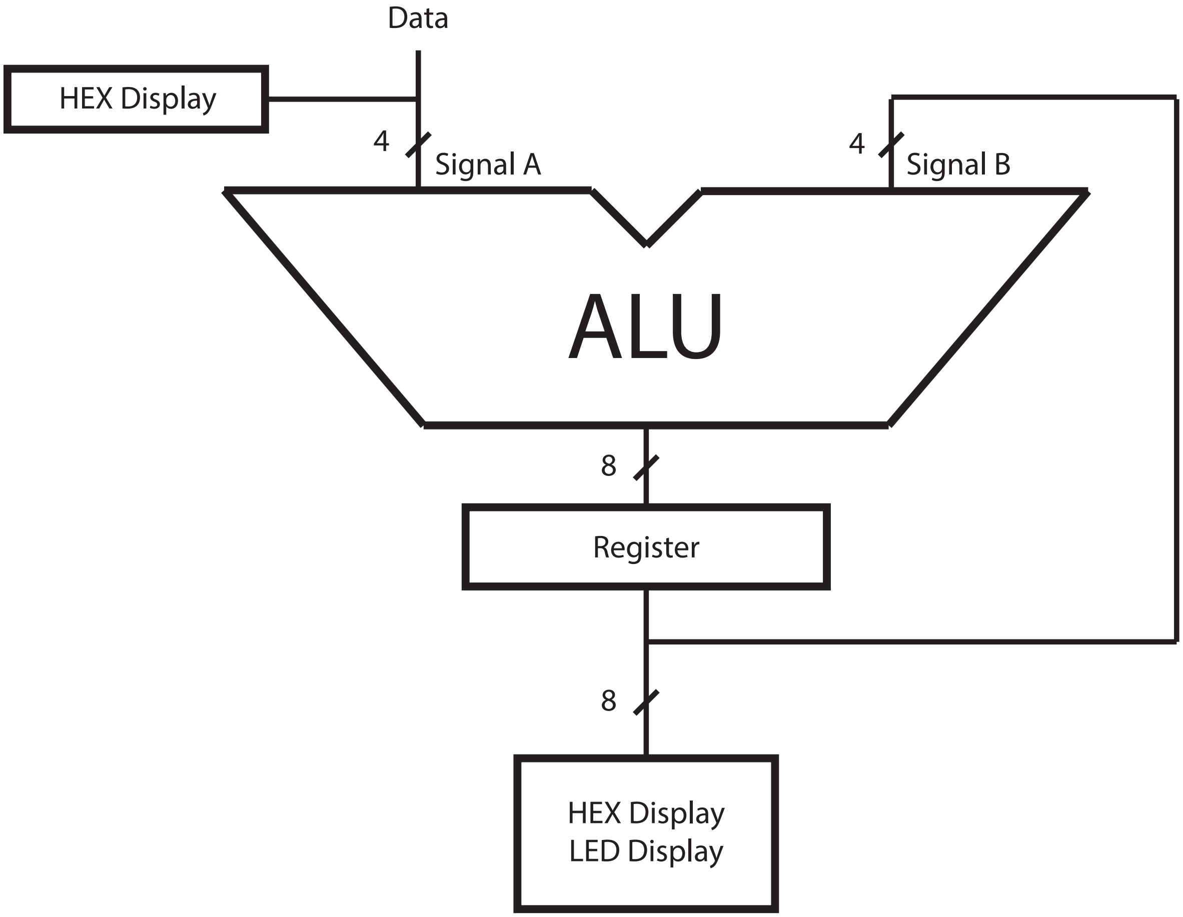

Figure 3: Simple ALU with register circuit for Part II.

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) What would happen if you didn't include the register in your diagram?
   (b) When multiplying two n-bit binary numbers, how many bits will you need to store the result?
3. Test your modules with *Poke*(👆). Choose test cases that make you feel confident about your ALU's correctness in preparation for you demo.
   (a) Document these test cases in your prelab report by providing a list of the test sequences you used for each ALU operation to verify its correctness. **(PRELAB)**
   (b) 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 the correct operation of these functions. **(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 sequence of flip-flops that move their contents one flip-flop down the row on each rising clock edge. Figure 4 shows one bit of this shift-register. It contains a positive edge-triggered flip-flop to store the ShifterBit's value and two multiplexers that determine the source of the ShifterBit's contents.

To create an 8-bit shift-register, you will use eight instances of the circuit in Figure 4 to design your 8-bit shift-register with optional arithmetic shift and parallel load as shown in Figure 5.

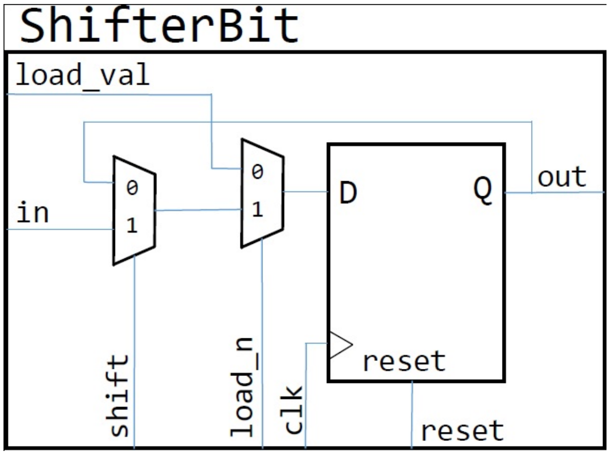

Figure 4: Single-bit shift-register

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

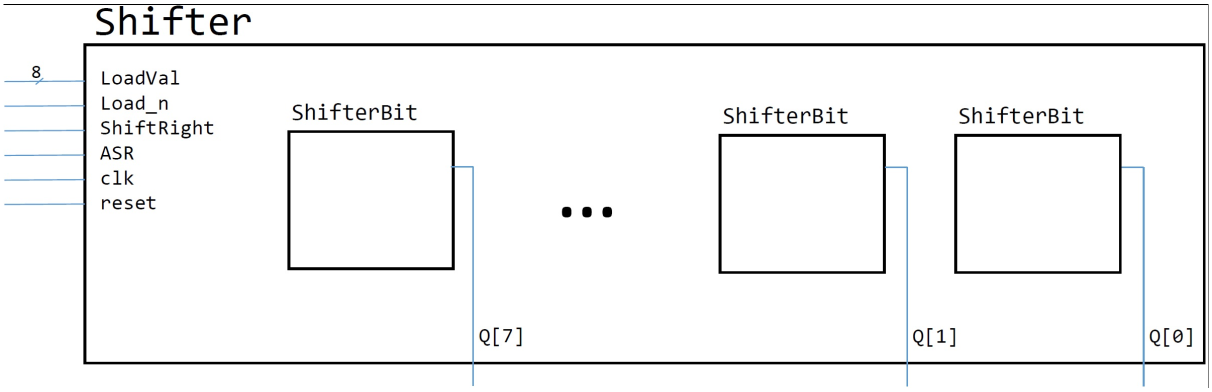

Figure 5: 8-bit shift-register of Part III. **None** of internal connections are shown here.

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 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)**