Student Name: Yupu Wang
Student ID:20012100012
HW ID: H00365030
Programme:BEng in Telecommunications Engineering
University: Xidian University & Heriot-Watt University
Year:Year 2
Submission Date:October 27nd,2021

B38DB Digital Design & Programming

Lab 3 – Multifunction Registers

Part 1: Introduction

In Lab_3, we are going to design registers and how to implement multifunctional registers by adding functional capability using combinational logic devices. A register is a memory device that can be used to store more than one bit of information. A register consists of several D-Q flip flops (which store 1 bit each) with common control signals that control the movement of data to and from the register. The main operations on a register as like those on any storage device. These include,

- Load or Store: Put new data into the register
- Read: Retrieve the data stored in the register

Using various logic devices, we can add functional capability allowing us to create up-down counters using registers, shift registers and multifunctional registers capable of various functions depending on the selected control signals.

Part 2: Design and Implementation of a 4-bit Serial-in to Parallel-out (SIPO) register

For the first task, you are required to design a 4-bit Serial-in to Parallel-out (SIPO) register in Logisim using D-Q flip flops. A SIPO register takes data into the register serially (data shifts every clock pulse) and outputs the data simultaneously (all at once).

To start the design, in the object browser, select the memory folder and drag four 'D Flip-Flop's to the workspace. Ensure they are aligned in one row.

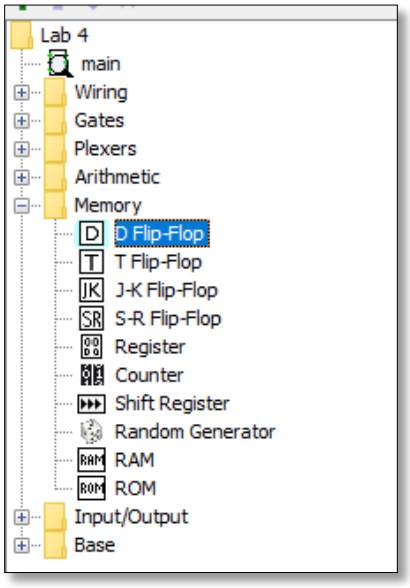


Figure 1: Object browser showing the D Flip-Flop component.

All registers require a clock signal to keep the flip flops in sync. You can create a clock by going to the 'Wiring' section of the object browser and dragging a 'Clock' into the workspace.

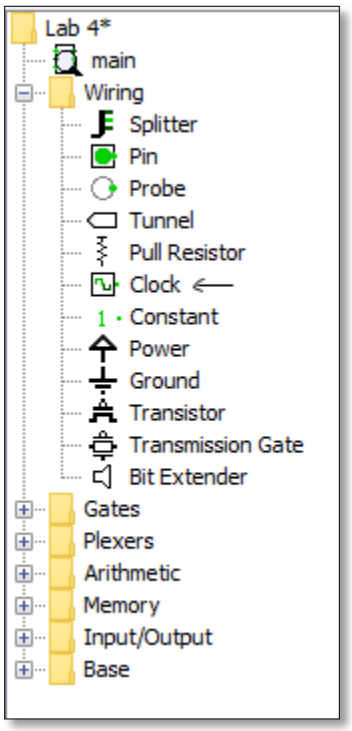


Figure 2: Object browser showing the Clock component.

Now, you should connect the clock signal into the clock input nodes for each of the flipflops. Your workspace should now look something like the one shown in Figure 3.

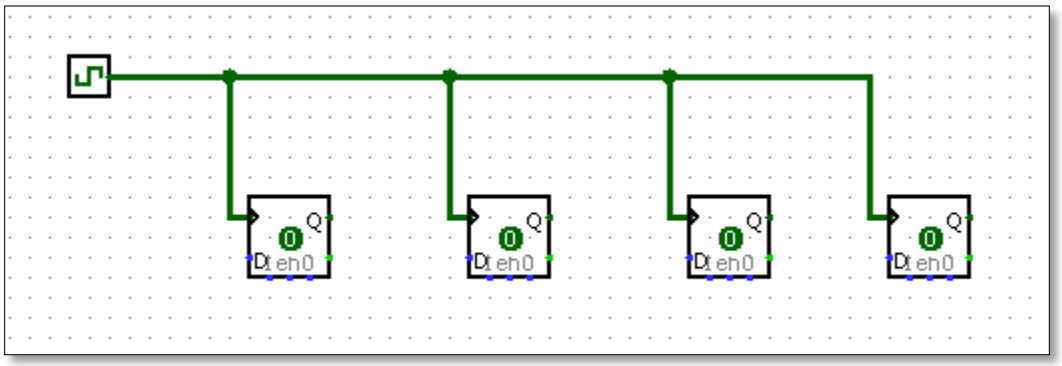


Figure 3: Workspace showcasing the clock signal connected to each of the flipflops.

For a SIPO register, you only require a one-bit input since the data is going to be fed serially into the register. Create a one-bit input pin and feed it into the D input node of the first flip flop on the left side. Connect the Q node of the first flop into the D node of the second flip flop. Repeat this until all the flipflops are connected. Your workspace should now look like the one shown in Figure 4.

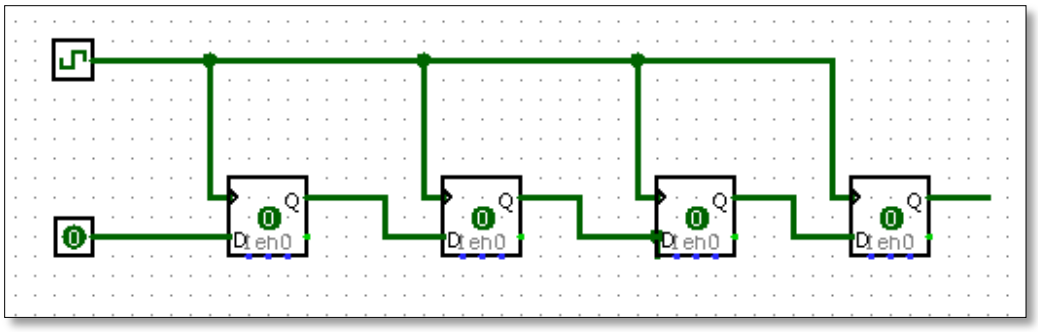


Figure 3: Serial In input established by connecting the inputs and outputs of the D flip flops.

Since the output of the 4-bit register should be parallel out. We require an output pin with 4 bits. Create an output pin of length 4 bits and connect the output of each flipflop to a splitter to join them into one 4-bit output. Your final SIPO register design should look like that shown in Figure 4.

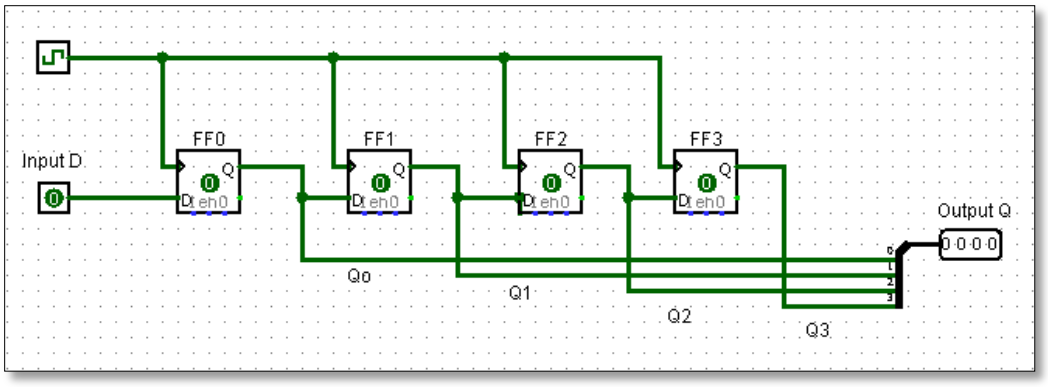


Figure 4: Final design of SIPO 4-bit register.

To test the design, you must simulate the clock and then feed different inputs on every clock pulse. To accomplish this, click on simulate in the menu bar and select “Ticks Enabled” to allow the clock to tick automatically. You can also vary the frequency of the clock signal by changing it from “Tick Frequency” in the same section. For this test, a frequency of 1 Hz is sufficient. However, you may change it as you wish if you find it too quick to change the inputs and record the results. You may also wish to manually click the clock off and on using the probe function to obtain the results. Figure 5 shows the setting on the menu bar.

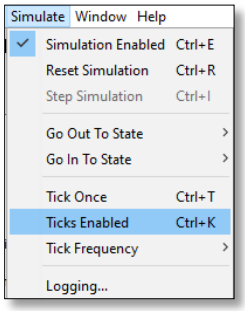


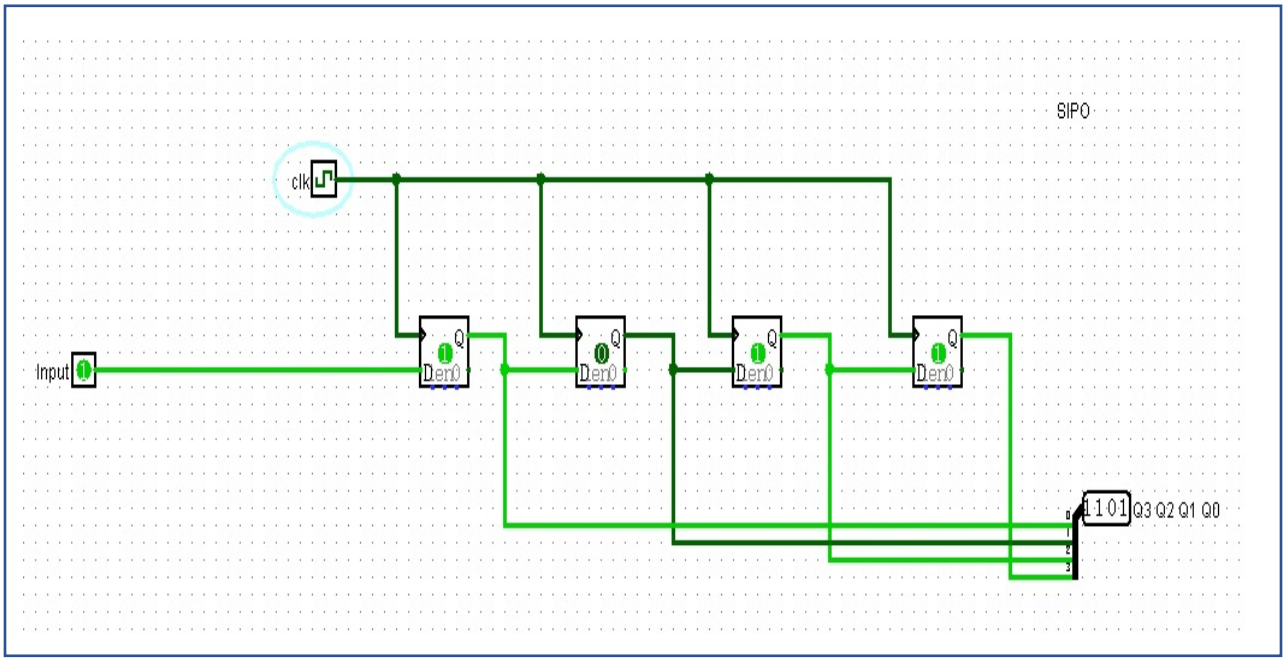
Figure 5: Menu bar highlighting the “Ticks Enabled” option to allow for simulation of the clock signal.

Once you see the clock switching between on and off, you can proceed to change the inputs. Remember that the flip flops will trigger on the rising edge (off -> on) of the clock pulse. **Use the input stream '100110111' and record the results of your simulation in Table 1.**

Table 1: SIPO Simulation Results

Clock Pulse Rising Edge	1	2	3	4	5	6	7	8	9
Input	1	0	0	1	1	0	1	1	1
Q0	1	0	0	1	1	0	1	1	1
Q1	0	1	0	0	1	1	0	1	1
Q2	0	0	1	0	0	1	1	0	1
Q3	0	0	0	1	0	0	1	1	0

Screenshot of SIPO 4-bit Register on the 7th clock pulse:



Part 3: Design and Implementation of SISO, PISO and PIPO Registers

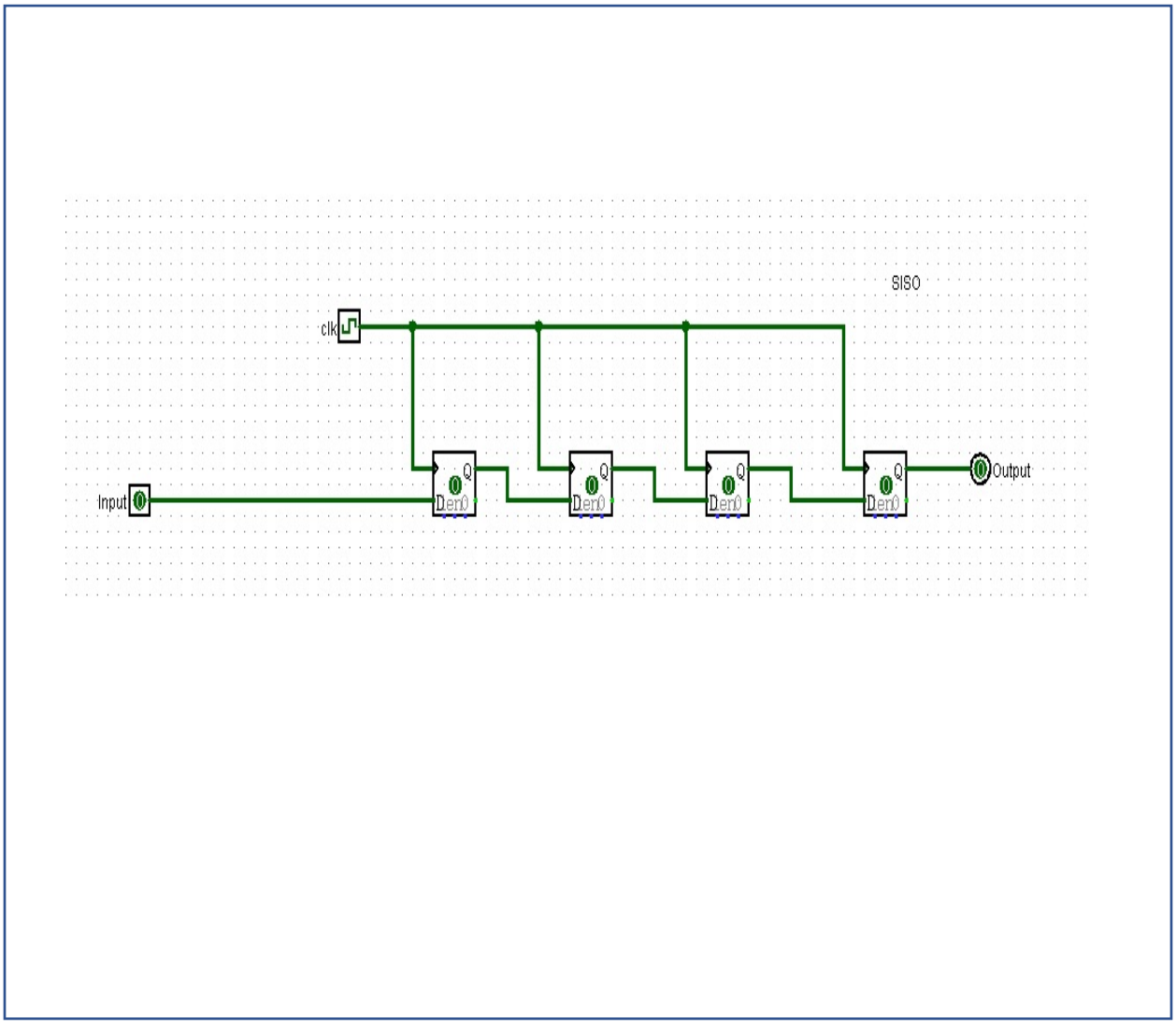
In this section, you are required to design, implement, and test the three other types of 4-bit registers. These are,

- Serial-in to Serial-out (SISO): data is shifted Serially "IN" and "OUT" on every clock pulse.
- Parallel-in to Serial-Out (PISO): data is loaded into the register simultaneously and is shifted out of register serially one bit at a time under clock control.
- Parallel-in to Parallel-out (PIPO): data is loaded simultaneously into the register and to the outputs by the same clock pulse.

Use the techniques you have learnt from Part 2 and from lectures to obtain the results.

a) Serial-in to Serial-out (SISO) Register:

Screenshot of SISO register design:



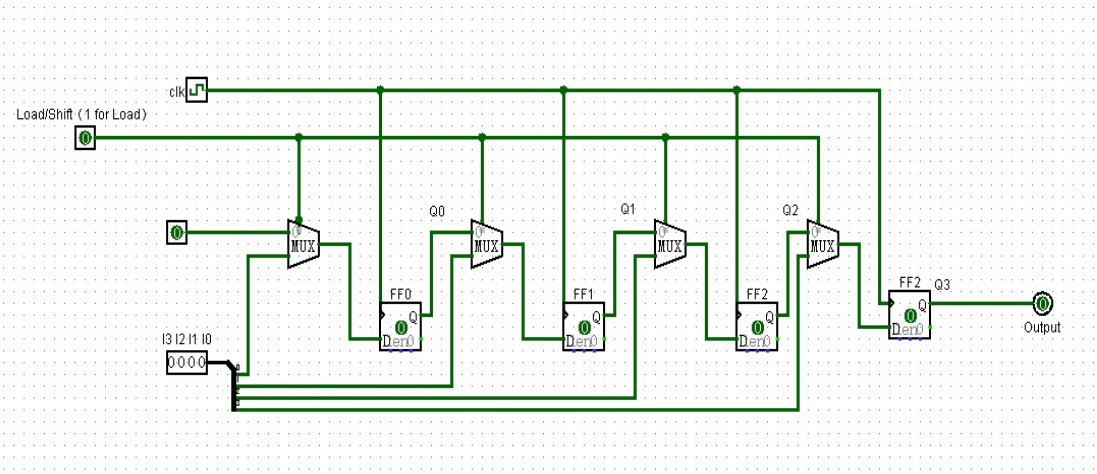
Use input stream '11010O011' and fill in Table 2 with the observed results.

Table 2: SISO Simulation Results

Clock Pulse Rising Edge	1	2	3	4	5	6	7	8	9
Input Stream	1	1	0	1	0	0	0	1	1
Output	0	0	0	1	1	0	1	0	0

b) Parallel-in to Serial-Out (PISO): (Inital value for first Mux=0)

Screenshot of PISO register design:



Use inputs as shown in Table 3 for each clock pulse and record the results observed.

Table 3: PISO Simulation Results

Clock Pulse Rising Edge	1	2	3	4	5	6	7	8	9
I0	1	1	1	1	1	1	1	0	0
I1	0	0	0	1	0	1	1	0	0
I2	1	1	1	0	0	1	1	1	0
I3	1	1	1	0	0	1	1	0	0
Load/Shift (1 for Load)	1	0	0	1	1	1	0	1	0
Output	1	1	D	0	0	1	1	0	1

c) Parallel-in to Parallel-Out (PIPO):

Screenshot of PIPO register design:

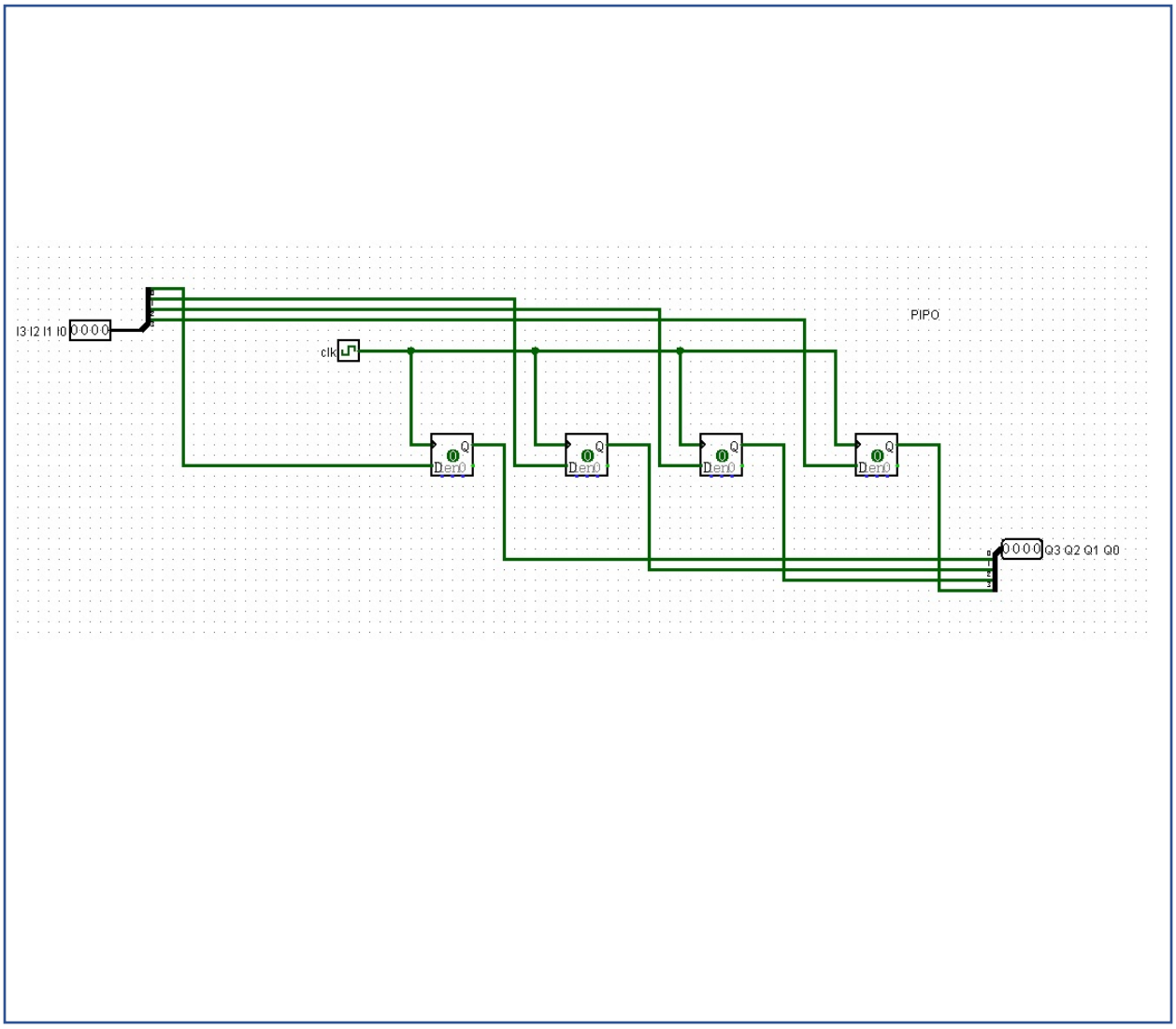


Table 4: PIPO Simulation Results

Clock Pulse Rising Edge	1	2	3	4	5	6	7	8	9
I0	1	1	1	1	1	1	1	0	0
I1	0	0	0	1	0	1	1	0	0
I2	1	1	1	0	0	1	1	1	0
I3	1	1	1	0	0	1	1	0	0
Q0	1	1	1	1	1	1	1	0	0
Q1	0	0	0	1	0	1	1	0	0
Q2	1	1	1	0	0	1	1	1	0
Q3	1	1	1	0	0	1	1	0	0

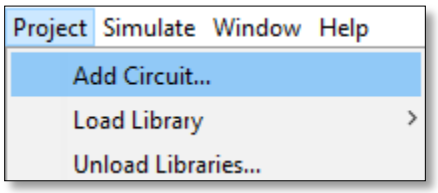
Part 3: Design and Implementation of a 4-bit Multifunction Register with Separate Control Inputs

In this section, we are going to look at designing a 4-bit multifunction register with separate control inputs. The multifunction register is expected to have four functions. These are,

- Maintain present value (when all inputs are 0)
- Shift right (when shr value is set to 1)
- Shift left (when shl value is set to 1 AND shr value is set to 0)
- Parallel load (when ld value is set to 1)

First, you are required to design the combinational logic that will determine the control signal inputs to the multiplexors. This will be created as a sub-circuit. Here are the steps for a refresher on how to do this.

Step 1: Create a sub circuit by going to Project -> Add circuit.



Step 2: Name the circuit, “Combinational Logic”.

Step 3: Ensure that the Combinational Logic circuit is being edited.

Using the rules stated in the bulleted list at the start of this section, fill in the truth table below.

Table 5: Truth Table for Multifunction Register

Inputs			Outputs		Operation
ld	shr	shl	s1	s0	
0	0	0	0	0	Maintain present value
0	0	1	1	1	Shift left
0	1	0	1	0	shift right
0	1	1	1	0	shift right
1	0	0	0	1	Parallel load
1	0	1	0	1	Parallel load
1	1	0	0	1	Parallel load
1	1	1	0	1	Parallel load

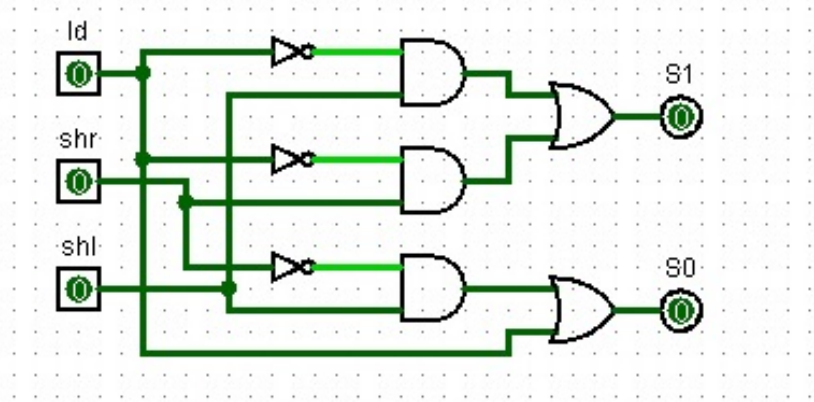
Using the truth table from Table 5, identify the simplified logic equation for s1 and s0.

s1 = $ld' \cdot shr' \cdot shl + ld' \cdot shr \cdot shl' + ld' shr \cdot shl = ld' \cdot shl + ld' \cdot shr$

s0 = $ld' \cdot shr' \cdot shl + ld \cdot shr' \cdot shl' + ld \cdot shr' \cdot shl + ld \cdot shr \cdot shl' + ld \cdot shr \cdot shl = ld + shr' \cdot shl$

Use the logic equations for s1 and s0 to implement the combinational logic circuit.

Screenshot for combinational logic circuit (ensure to use clear labels to identify inputs and outputs):



Once the combinational logic circuit is completed, you can go back to the main circuit and continue working on the multifunction register. The multifunction register consists of four D-Q flip flops with four 4-1 multiplexors preceding it. Your initial workspace should look like that shown in Figure 5. (Note that the MSB is at the bottom and LSB is at the top)

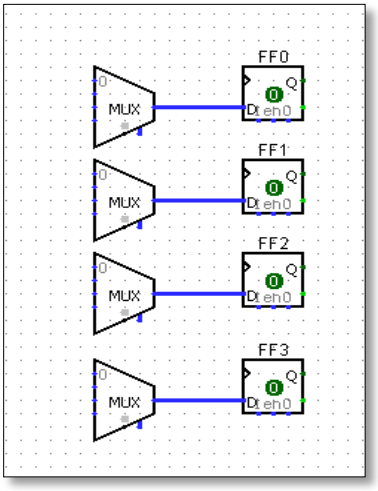


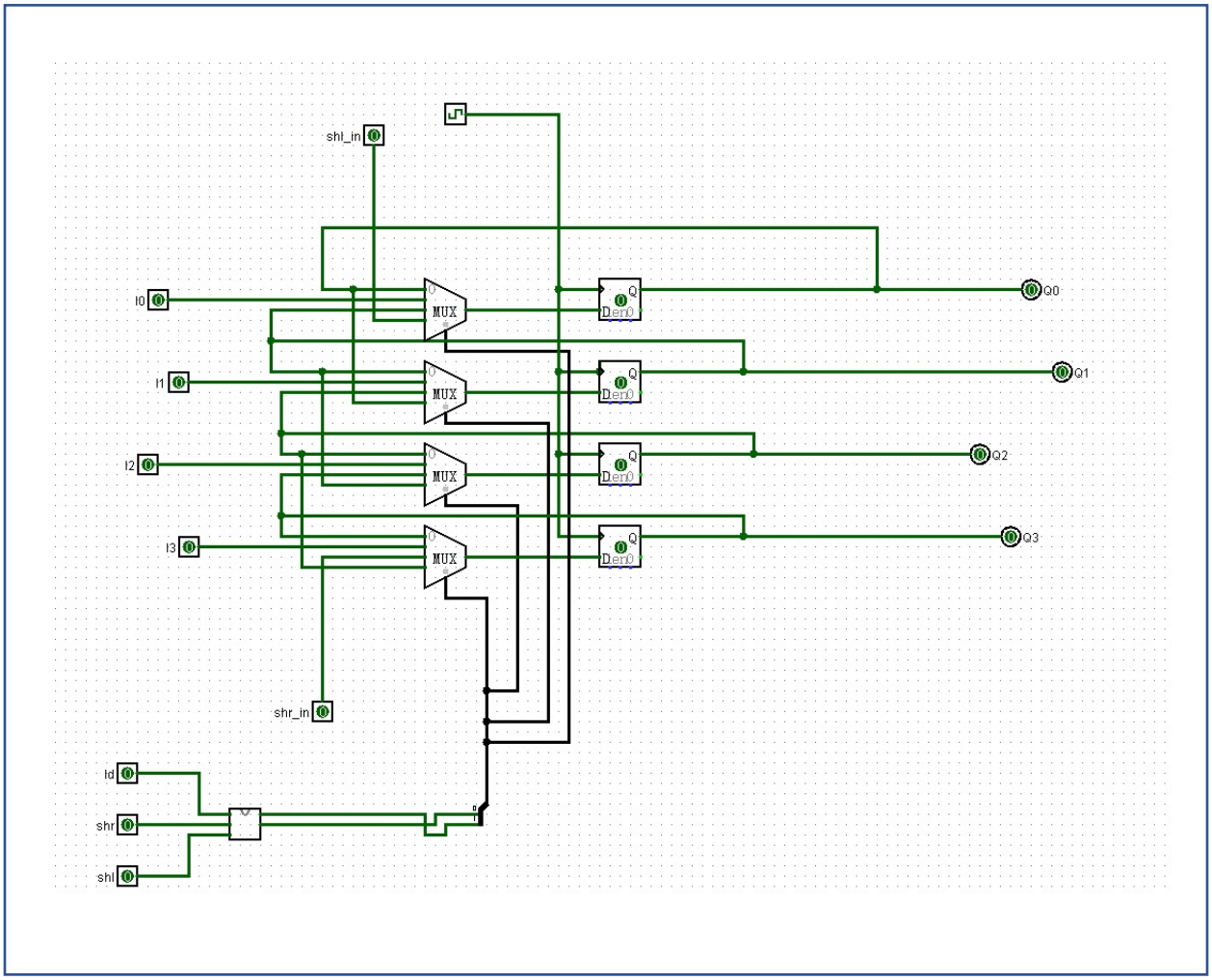
Figure 5: Initial workspace setup for the 4-bit Multifunction register

Your next steps are as follows,

1. Add the combinational logic circuit block under your circuit and connect the s0 and s1 outputs to the multiplexors using a splitter to combine the outputs into a`single 2-bit wire.
2. Create three inputs for *ld, shr* and *shl* to feed into the combinational logic circuit block input nodes.
3. Create four inputs for i0, i1, i2 and i3 to act as inputs to the multiplexor's 2nd input node.
4. Create two inputs for *shl_in* and *shr_in*.
5. Add a clock and connect it to the clock node for all the flip-flops.
6. Create four outputs for Q0, Q1, Q2 and Q3.
7. ~~P~~roceed to complete the connections as required to complete the 4-bit multifunction register circuit.

Refer to the lecture slides for more information if required.

Screenshot of the completed Multifunction Register:



Simulate the Multifunction Register using the inputs for each rising clock pulse shown in Table 6 and note down the observed outputs and operation that occurred on that clock pulse.

Table 6: 4-bit Multifunction Register Simulation Results

Clock Pulse Rising Edge	1	2	3	4	5	6	7	8	9
i0	0	1	1	1	1	1	1	0	0
i1	0	0	1	0	1	1	1	0	0
i2	1	0	1	0	0	1	1	1	0
i3	1	1	1	0	0	1	1	0	0
shl_in	1	1	1	1	0	0	0	1	1
shr_in	0	0	0	1	1	1	1	1	1
Ld	1	1	0	0	1	0	0	0	0
Shr	0	0	1	0	0	0	1	1	0
Shl	0	0	1	1	0	1	0	0	0
Q0	0	1	0	1	1	0	1	1	1
Q1	0	0	0	0	1	1	1	0	0
Q2	1	0	1	0	0	1	0	1	1
Q3	1	1	0	1	0	0	1	1	1
Operation	Parallel load	Parallel load	shift right	Shift left	Parallel load	Shift left	shift right	shift right	Maintain present value

Part 4: Conclusion

In this lab, you have learnt how to create storage elements in the form of DQ Flip-flops and how to extend them into multi bit storage elements known as registers. You have also learnt how to create various types of shift registers. Moreover, you have learnt how to create multi-functional registers using combinational logic circuitry to control the functions.