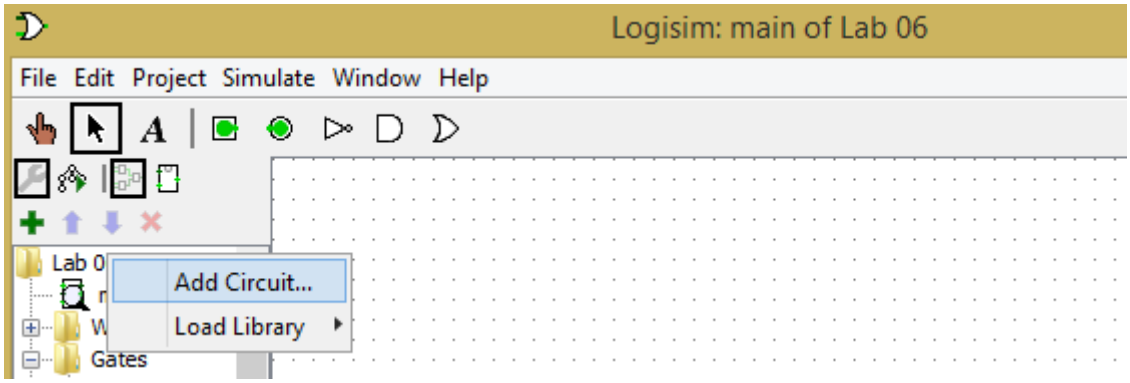
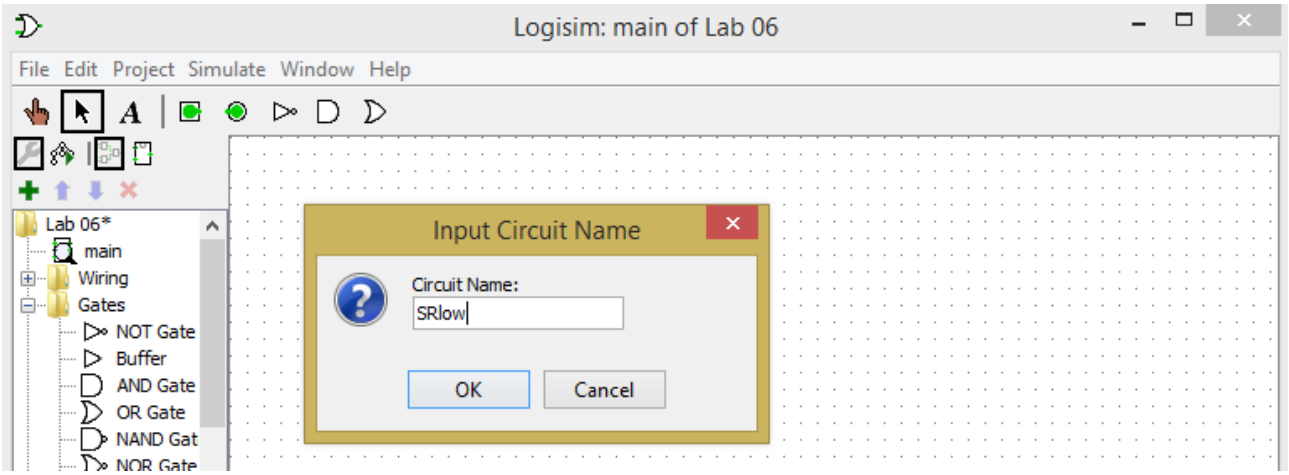
The tasks 1–3 in this lab are done in Logisim. Task 4 is done in Verilog.
Open Logisim. Save your new project as Lab05.circ Right-click on Lab05 and choose Add Circuit...



Task 1

Call a new circuit, and call it SRlow



Build an SR latch using NAND gates (active-low inputs). Label the inputs of the latch as S' and R' and outputs Q and Q'.

Show and explain how the latch remembers a 1.
Show and explain how the latch remembers a 0.

Show that in one case, the latch's output is 1 when both inputs are 1's, while in other case the latch's output is 0 when both inputs are 1's.

Task 2

Add a new circuit to the project, and call it SRhigh

Build an SR latch using NOR gates (active-high inputs).

Show and explain how the latch remembers a 1.
Show and explain how the latch remembers a 0.

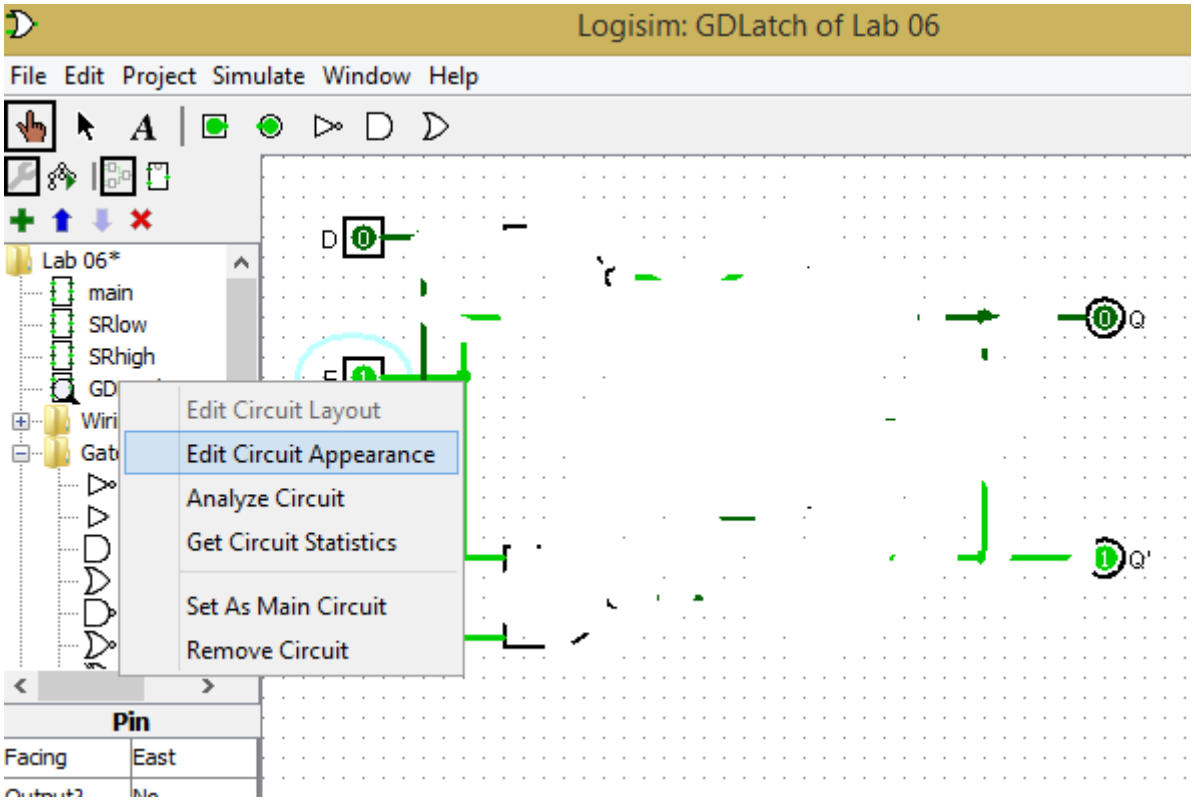
Show that in one case, the latch's output is 1 when both inputs are 0's, while in other case the latch's output is 0 when both inputs are 0's.

Task 3

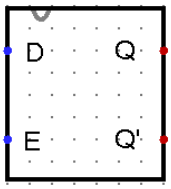
Add a new circuit to the project and call is GDLatch

Build a gated D latch.

Create a symbol for this circuit: right-click on the circuit name and choose Edit Circuit Appearance



Make the following symbol for this circuit:



Add a new circuit to the project and call is Dffpos
This circuit will be a positive-edge-triggered D flip-flop
Build a circuit for a positive-edge-triggered D flip-flop using gated D latches that you have just created.
Add an enable signal to your circuit through a multiplexer so that it could remember a binary value.
Explain how to store 1 or 0 in your flip-flop.

Task 4

In this task you will start learning how to build sequential circuits using Verilog.

A Verilog `always` statement is written in the form

```
always @ (sensitivity list)
statement;
```

The `statement` is executed only when the event specified in the `sensitivity list` occurs.

In `always` statements, signals keep their old value until an event in the `sensitivity list` takes place that explicitly causes them to change. Hence, such code, with appropriate sensitivity lists, can be used to describe sequential circuits with memory.

In contrast, continuous assignment statements (`assign`) are reevaluated anytime any of the inputs on the right hand side changes. Therefore, such code necessarily describes combinational logic.

This is the code for a gated D latch:

```
module dlatch (data, en, q);
    input data, en;
    output reg q;

    always @ (en or data)
        if (en) begin
            q <= data;
        end

endmodule
```

Save this code into a file with the name dlatch.v in the folder iverilog\samples

`q <= data` is pronounced “q gets data”
`<=` is called a **nonblocking assignment**. It is used instead of `assign` inside an `always` statement.

All signals on the left hand side of `<=` or `=` in an `always` statement must be declared as `reg`

In this example, `q` is both an `output` and a `reg`, so it is declared as `output reg q`

Declaring a signal as `reg` does not mean the signal is actually the output of a register (registers are arrays of flip-flops). All it means is that the signal appears on the left hand side of an assignment in an `always` statement. We will see later examples of `always` statements describing combinational logic in which the output is declared `reg` but does not come from a flip-flop or a latch.

Add a testbench to the same file dlatch.v. The testbench should save the waveform into the file dlatch.vcd

You should demonstrate me the work of a gated D latch using the waveform that you obtain in this task. In particular, your waveform must show the following. First, store 0 into the latch. Then, switch the Enable signal off, and show that in this case the latch remembers the stored value and does not react to any changes in the input. After that, switch the Enable signal on, store a 1 in the latch, and show that the latch can remember the value 1 when the Enable signal is low.