STARS 2023: Experiment No. 7

Introduction to System Verilog

(Materials are adapted from ECE270 and ECE337 course materials)

Objective:

- To learn how to convert a given sequential circuit design into System Verilog.
- To learn how to use the System Verilog simulator to synthesize and simulate these designs.
- To learn how to verify the outputs of the sequential circuits on GTKWave.

Lab Tasks: Implement the following sequential circuits using behavioral modeling

1. Edge triggered D, JK Flip flop with Asynchronous Preset and Clear
2. 2-bit synchronizer

Lab task1: Edge triggered D, JK Flip flop with Asynchronous Preset and Clear

In experiment 4, you implemented a D flip-flop on a breadboard. Flip-Flops are the basic synchronous storage cell for CMOS designs and form the basis for all 'registers'. The most common is the D-Flip-Flop with Set and Reset signals that allow the design to be cleared/reset/initialized to a known operating state. The syntax for describing a Flip-Flop is rather simple and straightforward but is different from the combinational logic that you have been working with primarily so far. The most common flip-flop used in designs is a rising-edge sensitive flip-flop with an active low reset. You will use an asynchronous, active-low system reset. When this line is asserted (logic '0'), all registers/flip-flops in the device must reset to their initial value.

```
always_ff @ (posedge clk, negedge n_rst)
begin [: <block tag name>]
  if(1'b0 == n_rst) begin
    <Flip-Flop Signal Name> <= <reset value>;
  end
  else begin
    <Flip-Flop Signal Name> <= <Flip-Flop input signal>;
  end
end
```

This syntax implements a flip-flop due to the nature of how the sensitivity list is used and the "always_ff" tells the compiler that you intend for it to be flip-flop so it should give error messages if the code is not correct for a flip-flop. You should always use the "always_comb" block instead of the "always" block for combination logic for similar reasons as the "always_ff" block for flipflops/registers.

Required module name: d_ff
Required port name:

Port name	Direction	Description
clk	input	The system clock.
n_rst	input	This is an asynchronous active low system reset. When this line is asserted (logic '0'), all registers, flip-flops in the device must reset to their initial value.
d_in	input	1-bit data input
d_out	output	1-bit output

On your top module, instantiate the d_ff module and connect its inputs and outputs to push buttons and LEDs on the FPGA as follows-

```
d_ff u1(.clk(hz100), .n_rst(pb[0]), .d_in(pb[1]), .d_out(right[0]));
```

To verify the correctness of your circuit, click on Build target: cram and click on Makefile: Build the current target. On pressing pb[0] and pb[1] simultaneously, you will see the right[0] LED glows. To verify your flip-flop on GTKWave, follow the steps-

1. Since you are only learning SystemVerilog for the first time, we will provide the necessary tools to verify your code. Verification is done by writing a testbench file that instantiates your top module, driving inputs and reading outputs, and printing out any unexpected output values with error messages. For this lab, we provide the test bench needed to verify the modules you just wrote, do not modify it.
2. Change/edit your build target to set it to 'verify'. Run this new target. If you see any "ERROR"s in the VSCode terminal, you'll need to figure out what your modules are missing. This is how you verify your modules. If there is no error, you should see the same compilation output, but instead of flashing your design to the FPGA, a new GTKWave window should appear.
3. On the new window, select the instance name under the tab SST, right-click the signals you want to see on the waveform window and select insert. All the signals and wires should appear on the waveform.
4. Observe how the inputs are changing the outputs. If you correctly implemented your module, you should get the expected outputs. To change the format, you may select the signal under the signals tab, right-click, and change the data format to decimal/hexadecimal/binary or any format you want.

TA check-off point: Show your D flip-flop output to your TA for check-off.

Note: The working of JK flip-flop is given in the study materials. Try to implement it on the simulator. This is for your practice only and no additional check-off may be required. An optional task is modifying the D flip-flop testbench to view the JK flip-flop output on GTKWave.

Lab task2: 2-bit synchronizer

Whenever a design has input signals which are asynchronous to the system clock, these input signals need to be synchronized. Assuming the inputs are read on the rising edge of the system clock, synchronization will prevent reading a transition in the input signal. To synchronize a signal, the input must go through two flip-flops clocked on the system clock. In general, the synchronizer circuit has two purposes:

1. Synchronize the input signal to the system clock domain
2. Reduce the metastability of the signal.

Metastability (the unknown value of a signal) is an issue when synchronizing an asynchronous signal into a flip-flop device. If the incoming asynchronous signal happens to change its state (logic value) at the same time the flip-flop device is capturing the data or during the setup or hold regions, an indeterminate value on the output of the flip-flop may be seen. Now we ask, "How does this affect the functionality of the circuit?" It can adversely affect the circuit if the captured data is fanned out to multiple places within the circuit. The indeterminate value of the captured signal can be interpreted differently (either a logic '1' or '0') by different logic gates within the receiving circuit, thus leading to a possible malfunction of the circuit. To try to reduce the possibility of this occurring, a second flip-flop is attached in series to the data-capturing flip-flop. It will be the output of this second flip-flop that is used inside the receiving circuit. Please note that adding the second flip-flop does not guarantee the output of the second flip-flop to be stable but the chances of the output being metastable are greatly reduced.

See the Reset to Logic Low Synchronizer schematics and the timing waveform below.

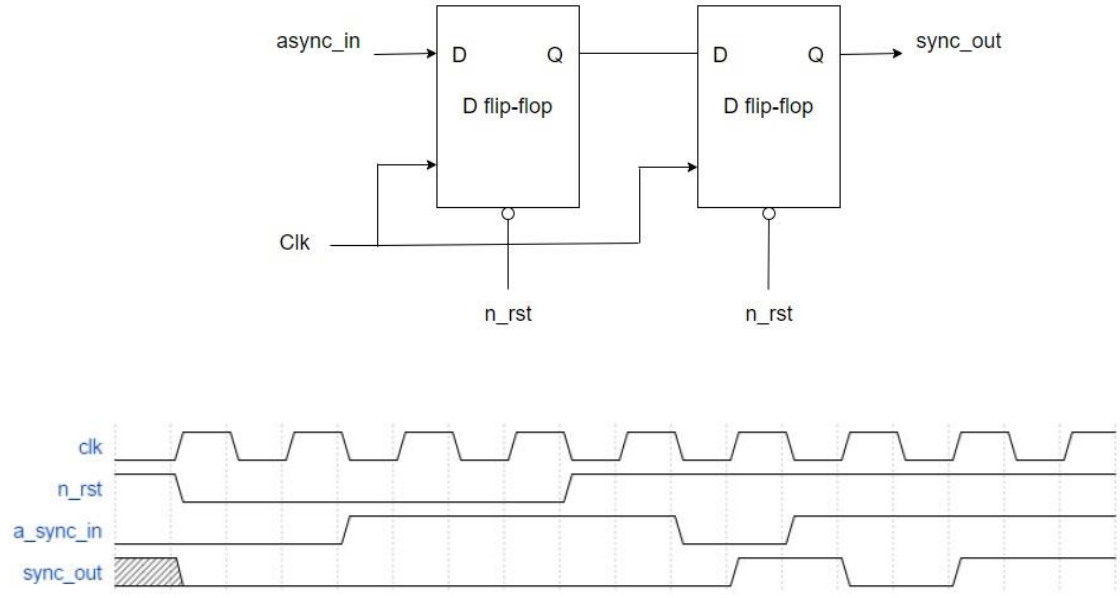

Figure 1: Schematics and timing waveform for 2-stage synchronizer for an active high input

Required module name: sync_low
Required port name:

Port name	Direction	Description
clk	input	The system clock.
n_rst	input	This is an asynchronous active low system reset. When this line is asserted (logic '0'), all registers, flip-flops in the device must reset to their initial value.
async_in	input	This is the original signal which is not synchronized to the supplied clock signal.
sync_out	output	This is the form of the input that is now synchronized with the supplied clock signal.

On your top module, instantiate the sync_low module and connect its inputs and outputs to push buttons and LEDs on the FPGA as follows-

```
sync_low u1(.clk(hz100), .n_rst(pb[0]), .async_in(pb[1]), .sync_out(right[0]));
```

Verify your circuit the same way you verified your D flip-flop. A test bench is provided.

TA check-off point: Show your reset to logic low synchronizer output to your TA for checkoff.