ASIC Design Laboratory
Lab 5 : Serial Communication Basics, FSM Design, and Data Streaming based Testing
Lab Manual

Fall 2020

The purpose of this lab exercises is to help you become familiar with designing and verifying simple serial communication building blocks and finite-state-machines (FSMs), which will later be common aspects of larger designs.

It is important to note, that given the fundamentally parallel and interactive nature of hardware designs, debugging designs described with HDL code requires a method that strictly identifies and leverages guaranteed cause-effect relationships with in the design's description. Other lazy or speculative debugging methods will generally result in vast amounts of wasted time, effort, and frustration and can easily increase debugging times by a factor of 10x.

In this lab, you will perform the following tasks:

- Design, implement, and verify flexible and scalable versions a Serial-to-Parallel Shift Register.
- Design, implement, and verify flexible and scalable versions a Parallel-to-Serial Shift Register.
- Implement and verify the functionality of the Mealy and Moore Sequence Detector FSMs originally designed in the Lab 1 post-lab.

1 Lab Setup

In a UNIX terminal window, issue the following commands, to setup your Lab 5 workspace:

mkdir –p ~/ece337/Lab5
cd ~/ece337/Lab5
dirset
setup5

The setup5 command is an alias to a script file that will check your Lab 5 directory structure and give you file needed for starting the lab. If you have trouble with this step please ask for assistance from your TA.

IMPORTANT: Make sure to add this new workspace into your 337 Repository, like you did in Lab1.

This way, you will always have the original copy in storage.

2 Flexible Shift Register Design

2.1 Flexible and Scalable Serial-to-Parallel Shift Register Design

In Lab 1's post-lab you created schematics for a 4-bit Most-Significant-Bit (MSB) first Serial-to-Parallel (StP) Shift Register. Based on that schematic (or any needed corrections/revisions) create an RTL diagram for this shift register design.

The example of expected behavior for this shift register design has been replicated from Lab 1 as Figure 1.

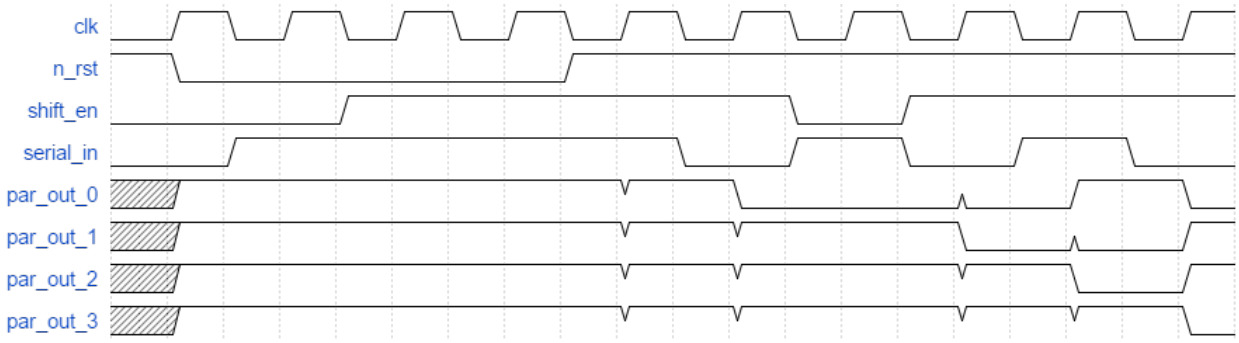

Figure 1: Timing waveform diagram for 4-bit MSB Serial-to-Parallel Shift Register

Once you have completed this RTL diagram, have a TA check your work before moving on to the implementation sections.

NOTE: If there is a queue for TA check-offs, then work on the RTL diagram for the the Parallel-to-Serial Shift Register in Section 2.2 while waiting for the TA.

2.1.1 Flexible Module Implementation

Using parameters and your RTL diagram for a 4-bit MSB StP Shift Registers design a flexible version that has the following behavior:

- Have a parameter called 'NUM_BITS' that determines the number of bits in both the internally stored value and the 'parallel_out' port, with a default value of 4.
- Have a parameter called 'SHIFT_MSB' that defines the shifting direction of the register such that a Boolean value of 'true' assumes input serial data is sent most significant bit first and 'false' assumes input serial data is least significant bit first.

The default values for the parameters for this design must make it the implementation of your 4-bit MSB Serial-to-Parallel Shift Register RTL diagram.

The module and file requirements are on the following page.

Required Module Name: flex_stp_sr

Required Filename: flex_stp_sr.sv

Required Ports:

Port name	Direction	Description
clk	input	The system clock. (Maximum Operating Frequency: 400Mhz)
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.
shift_enable	input	This is the active-high enable signal that allows the shift register to shift in the value of the serial_in sinal
serial_in	input	This is the serial input signal and thus will contain the value to be shifted into the register. The default/idle value for this signal is a logic '1', since this is the common idle value used in serial communications.
parallel_out[#:0]	output	This is the data that is currently held by the shift register. The actual port declaration should use the NUM_BITS parameter value to determine the value of the '#'

2.1.2 Testing your flexible Serial-to-Parallel Shift Register Design

To assist you in testing your new flexible design, the setup5 script you ran earlier has provided you with a starter test bench called 'tb_stp_sr_4_msb.sv', which is a test bench built around testing the 'default settings' version of your flexible serial-to-parallel shift register via the 'stp_sr_4_msb.sv' wrapper file. This setup, with a default settings wrapper file, was done so that (1) the test bench will both match up a design file that designates it's parameter settings and (2) you can easily copy and update this test bench for directly testing the non-default setting versions via other wrapper files (such as the provided 'stp_sr_8_msb.sv' and 'stp_sr_4_lsb.sv' ones). Remember to update the variables in the makefile prior to using the 'sim_full_source' and 'sim_full_mapped' make commands to simulate the different versions of your design.

NOTE: You will need to add more test cases to the provided test bench before you can be confident that your design is fully working, as only two example test cases (other than the power-on-reset one) are provided for you.

NOTE: The test bench provided to you uses 'streaming tasks' to consistently managing the timing of 'streaming' successive bit values through the shift-register design being tested. For more information on their operation and purpose, please look over Section 3.

2.1.3 Automated Grading of the Flexible Serial-to-Parallel Shift Register

To submit your flexible serial-to-parallel shift register design for grading, issue the following command at the terminal (can be from anywhere).

submit Lab5fs

2.2 Flexible and Scalable Parallel-to-Serial Shift Register Design

In Lab 1's post-lab you created schematics for a 4-bit Most-Significant-Bit (MSB) first Parallel-to-Serial (PtS) Shift Register. Based on that schematic (or any needed corrections/revisions) create an RTL diagram for this shift register design.

The example of expected behavior for this shift register design has been replicated from Lab 1 as Figure 2.

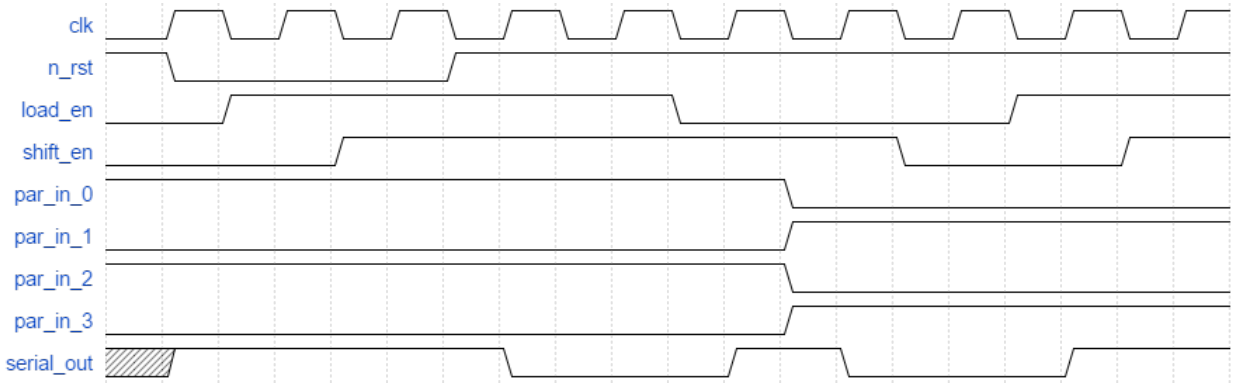

Figure 2: Timing waveform diagram for a 4-bit MSB Parallel-to-Serial Shift Register

Once you have completed this RTL diagram, have a TA check your work before moving on to the implementation sections.

2.2.1 Flexible Module Implementation

Using parameters and your RTL diagram for a 4-bit MSB PtS Shift Registers design a flexible version that has the following behavior:

- Have a parameter called 'NUM_BITS' that determines the number of bits in both the internally stored value and the 'parallel_in' port, with a default value of 4.
- Have a parameter called 'SHIFT_MSB' that defines the shifting direction of the register such that a Boolean value of 'true' assumes output serial data is to be sent most significant bit first and 'false' assumes output serial data is to be sent least significant bit first.

The default values for the parameters for this design must make it the implementation of your 4-bit MSB Parallel-to-Serial Shift Register RTL diagram.

The module and file requirements are on the following page.

Required Module Name: flex_pts_sr

Required Filename: flex_pts_sr.sv

Required Ports:

Port name	Direction	Description
clk	input	The system clock. (Maximum Operating Frequency: 400Mhz)
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.
shift_enable	input	This is the active-high enable signal that allows the shift register to shift in the value of the serial_in sinal
load_enable	input	This is the active-high enable signal that allows shift register to load the value at the parallel_in port into the register. If both the shift_enable and the load_enable are active, the load_enable takes priority.
parallel_in[#:0]	input	This is the data that will be loaded into the shift register. The actual port declaration should use the NUM_BITS parameter value to determine the value of the '#'
serial_out	output	This is the serial output signal and thus will contain the value that is being serially transmitted from the shift register.

2.2.2 Testing your flexible Parallel-to-Serial Shift Register Design

Just like for the serial-to-parallel shift register, the setup5 script has provided a test bench (tb_pts_sr_4_msb.sv) and wrapper files for testing your flexible parallel-to-serial shift register design. Remember to update the variables in the makefile prior to using the 'sim_full_source' and 'sim_full_mapped' make commands to simulate the different versions of your design.

NOTE: You will need to add more test cases to the provided test bench before you can be confident that your design is fully working, as only two example test cases (other than the power-on-reset one) are provided for you.

NOTE: The test bench provided to you uses 'streaming tasks' to consistently managing the timing of 'streaming' successive bit values through the shift-register design being tested. For more information on their operation and purpose, please look over Section 3.

2.2.3 Automated Grading of the Flexible Parallel-to-Serial Shift Register

To submit your flexible parallel-to-serial shift register design for grading, issue the following command at the terminal (can be from anywhere).

submit Lab5fp

3 Data-Streaming Tasks for Efficient Verification of Serial Designs

In the prior lab, tasks were used to manage and structure test benches for efficiently and consistently repeating work and timing of individual actions among test cases. In this lab we are dealing with designs that work with streams of data/bits which also must be consistently handled, and thus the provided test bench has be crafted to use 'streaming tasks' to do this. Streaming tasks are not a special form of syntax, but rather a special style of task usage where fixed or variable length streams/vectors of values are provided for the task to iterate through. This style of task usage, rather than replicated loops directly, enables clean and easy to update test cases for these common streaming scenarios, as well as certainty of having consistent timing handling these streams even when the data in the stream changes in the different test cases.

A 'streaming task' for dealing with the serial stream of bits that would be feed into the serial-to-parallel shift register should be created in a way that it uses another task for explicit timing of individual bits (as shown in Source Code 1. The reason for this is that you will need to create test cases that also working with either sending individual bits or change what happens between successive bits being sent, and should still be consistent with how each bit is sent when sending it.

```
1  task send_stream;
2      input logic bit_stream [];
3  begin
4      // Coniguously stream out all of the bits in the provided input vector
5      for(tb_bit_num = 0; tb_bit_num < bit_stream.size(); tb_bit_num++) begin
6        // Send the current bit
7        send_bit(bit_stream[tb_bit_num]);
8      end
9  end
10 endtask
```

Source Code 1: Serial-to-Parallel Shift Register Test Bench's task for sending a variable length stream of bits through the Shift Register.

Source Code 2 shows the provided test bench's task for sending a bit. Before setting the value of the bit to send, one should always synchronize away from the rising edge of the system clock (unless you are explicitly wanting to create and test for timing violation handling). Since shift enables allow the shift register to shift it's internal values on every rising edge while the enable is active, this task should explicitly only pulse the enable and end with it off. This way it will correctly and safely work for both back-to-back and isolated usage.

An example of a test case using this streaming task setup is shown in Source Code 3. The general flow is:

1. Reset the DUT and make inputs 'idle'/inactive values
2. Define/update the test data stream/vector for this test case
3. Update the expected output for checking at the end of the test case
4. Run the streaming task for this test cases stream
5. Check that everything streamed through correctly

NOTE: In the stp_sr test bench, the test data signal is declared as an unpacked array so that it can be dynamically sized across test cases and thus is assigned a value using the 'unpacked concatenation' method for declaring a vector of bits even though they are all zeros for this test case.

```
task send_bit;
    input logic bit_to_send;
begin
    // Synchronize to the negative edge of clock to prevent timing errors
    @(negedge tb_clk);

    // Set the value of the bit
    tb_serial_in = bit_to_send;
    // Activate the shift enable
    tb_shift_enable = 1'b1;

    // Wait for the value to have been shifted in on the rising clock edge
    @(posedge tb_clk);
    #(PROPAGATION_DELAY);

    // Turn off the Shift enable
    tb_shift_enable = 1'b0;
end
endtask
```

Source Code 2: Serial-to-Parallel Shift Register Test Bench's task for sending a bit to the Shift Register.

```
// ********************************************************************
// Test Case 2: Normal Operation with Contiguous Zero Fill
// ********************************************************************
tb_test_num  = tb_test_num + 1;
tb_test_case = "Contiguous Zero Fill";
// Start out with inactive value and reset the DUT to isolate from prior tests
tb_serial_in = 1'b1;
reset_dut();

// Define the test data stream for this test case
tb_test_data = '{SR_SIZE_BITS{1'b0}};

// Define the expected result
tb_expected_ouput = '0;

// Contigously stream enough zeros to fill the shift register
send_stream(tb_test_data);

// Check the result of the full stream
check_output("after zero fill stream");
```

Source Code 3: Serial-to-Parallel Shift Register Test Bench test case showing usage of streaming tasks.

NOTE: In the pts_sr test bench, the streaming task based test cases are structured slightly differently to account for the parallel load and serial output flow of the design. The main changes are that the expected output updating and output checking is done within the streaming task after each shift and there is final check after all of the shifts to make sure the correct value was used to 'fill in' while values were 'shifted out'.

4 Introduction to Implementing Finite-State-Machines in Verilog

In order to help get you more comfortable with designing finite state machine Verilog code like what will be used in the creation of various controller modules for larger designs, you will be implementing and verifying the '1101' Sequence Detector design from Lab 1's post-lab.

As a reminder, the '1101' detector's purpose is to assert a logic '1' output whenever the sequence "1101" is detected in a serial input data stream. For example, the sequence "11011011" will output a logic '1' twice.

4.1 Moore Machine '1101' Detector Design

4.1.1 Moore Machine '1101' Detector Specifications

Required Module Name: moore

Required Filename: moore.sv

Required Ports:

Port name	Direction	Description
clk	input	The system clock. (Maximum Operating Frequency: 400Mhz)
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.
i	input	The serial input for the input stream to process.
o	output	The current result from processing the serial stream so far.

4.1.2 Simulating and Verifying the Moore Machine '1101' Detector

You will need to create your own test bench for simulating and verifying your Moore '1101' detector design. This test bench will be graded via code coverage of a known properly working design during the automated grading submission.

You will need to use data streaming tasks similar to those used in the shift register test benches and will need to show their usage to a TA.

4.1.3 Automated Grading of the Moore Machine '1101' Detector

To submit your Moore '1101' detector design and test bench for grading, issue the following command at the terminal (can be from anywhere).

submit Lab5mo

4.2 Mealy Machine '1101' Detector Design

4.2.1 Mealy Machine '1101' Detector Specifications

Required Module Name: mealy

Required Filename: mealy.sv

Required Ports:

Port name	Direction	Description
clk	input	The system clock. (Maximum Operating Frequency: 400Mhz)
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.
i	input	The serial input for the input stream to process.
o	output	The current result from processing the serial stream so far.

4.2.2 Simulating and Verifying the Mealy Machine '1101' Detector

You will need to create your own test bench for simulating and verifying your Moore '1101' detector design. This test bench will be graded via code coverage of a known properly working design during the automated grading submission.

You will need to use data streaming tasks similar to those used in the shift register test benches and will need to show their usage to a TA.

4.2.3 Automated Grading of the Mealy Machine '1101' Detector

To submit your Mealy '1101' detector design and test bench for grading, issue the following command at the terminal (can be from anywhere).

submit Lab5me

5 Closing Remarks

- Turn in your check-off sheet at the beginning of Lab Lab 6 .