

# Homework 2

To pass this assignment, you need to complete the following tasks:

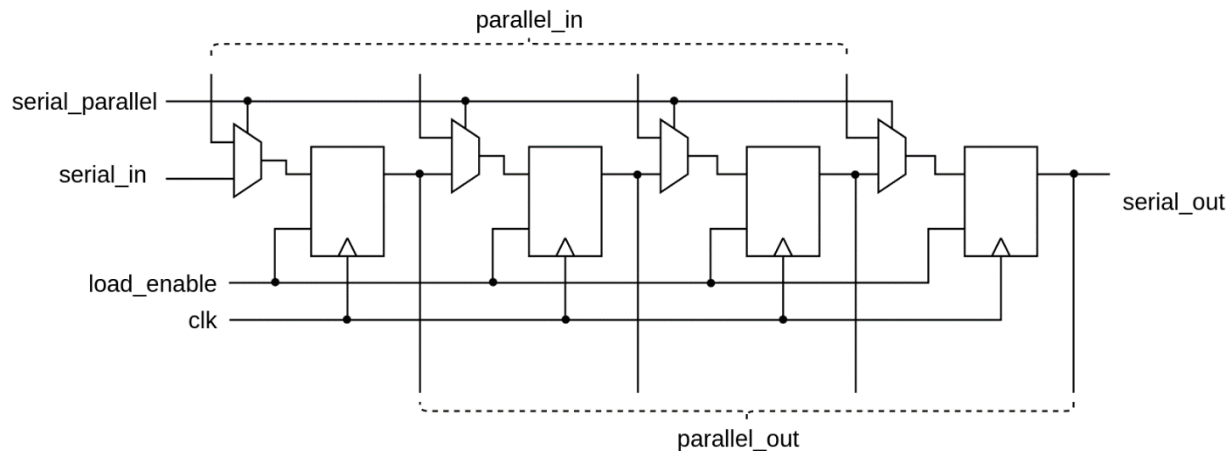
- 1) Solve at least one of the following problems.
- 2) Submit your solution to GitHub. (HDL and schematic when required)
- 3) Review and give feedback on your colleague's solutions assigned to you in Canvas.

## QUESTIONS

### 1.1 QUESTION 1

Use HDL to model a parametric N-bit shift register that supports serial and parallel loading. The flops update only when *load\_enable* is asserted. The flops can receive data in serial or parallel, based on the *serial\_parallel* signal. When *serial\_parallel* = 0, the flops get the serial data as shown in the figure below. When *serial\_parallel* = 1 the flops are loaded with *parallel\_in* data.

- Write a SystemVerilog code for the N-bit shift register described above.
- Write a Testbench that verifies and tests all possibilities of loading data serially and in parallel.



Module pinout

Name	Direction	Width	Control/Data	Description
<b>clk</b>	in	1	Control	Clock signal
<b>rstn</b>	in	1	Control	Asynchronous active low reset

<b>serial_parallel</b>	in	1	Control	If 0, load serially; if 1, load parallel
<b>load_enable</b>	in	1	Control	If 1, flops accept new data
<b>serial_in</b>	in	1	Data	Serial input to the shift register
<b>parallel_in</b>	in	N	Data	Parallel input to the shift register
<b>parallel_out</b>	out	N	Data	Parallel output of the shift register
<b>serial_out</b>	out	1	Data	Serial output of the shift register

```

module shift_register #(parameter N=4)
    (input logic clk,
     input logic rst_n,
     input logic serial_parallel,
     input logic load_enable,
     input logic serial_in,
     input logic [N-1:0] parallel_in,
     output logic [N-1:0] parallel_out,
     output logic serial_out);

    //complete here

endmodule

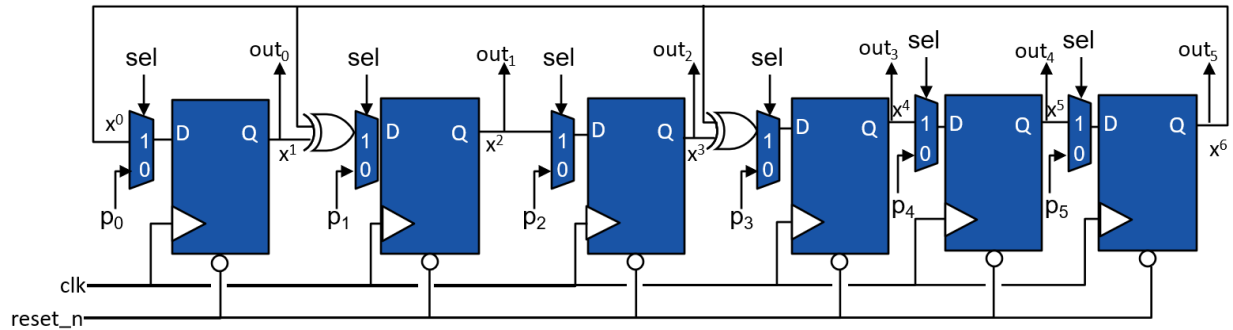
```

## 1.2 QUESTION 2

**LFSR (Linear Feedback Shift Register)** is a digital circuit that shifts bits through a register and uses linear feedback (XOR operations) to generate the next bit. It's widely used in cryptography, error detection, pseudo-random number generation, and digital communication.

Model a design using HDL that implements a **6-bit LFSR**. The design loads initial values in parallel using the *sel* signal. When the *sel* signal is '0', new bits are loaded to the DFFs in parallel using a 6-bit input. When the *sel* signal is '1', the design does not load new bits from the input but works in the shift mode as an LFSR. The LFSR has a **6-bit output** that is read directly from the output of each flip-flop. The schematic and module definition are given below. All the *sel* signals are connected together.

- Write a SystemVerilog code for the 6-bit LFSR described above.
- Write a Testbench that verifies the described functionalities.



Module pinout

Name	Direction	Width	Control/Data	Description
<b>clk</b>	in	1	Control	Clock signal
<b>rstn</b>	in	1	Control	Asynchronous active low reset
<b>sel</b>	in	1	Control	If 1, load serially; if 0, load in parallel
<b>parallel_in</b>	in	6	Data	Parallel input data
<b>parallel_out</b>	out	6	Data	Parallel output data

```

module LFSR_6bit (
    input logic clk, rst_n,
    input logic sel,
    input logic [5:0] parallel_in,
    output logic [5:0] parallel_out
);
    // ...
    // Add your description here
    // ...
endmodule

```

### 1.3 QUESTION 3

Design and model an N-bit up/down counter in SystemVerilog. The counter has an *up\_down* signal that controls its counting direction. When *up\_down* = 1, the counter counts up; when *up\_down* = 0, it counts down. When *load* = 1, the counter loads an initial value from *input\_load* on the next rising edge of the clock. In an up/down counter, the *carry\_out* signal shows when the counter has wrapped around its range. When counting up, *carry\_out* goes high if the counter reaches its maximum value and then increases again, wrapping back to zero. When counting down, *carry\_out* goes high if the counter reaches zero and then decreases again, wrapping

back to the maximum value. This way, carry\_out indicates that the counter has completed a full cycle in either direction.

- Draw the RTL schematic for a 4-bit up/down counter.
- Write the SystemVerilog code for the discussed counter.
- Write a testbench to verify its functionality in both counting modes (up and down) and for the load/reset operations.

Module pinout

Name	Direction	Width	Control/Data	Description
clk	in	1	Control	Clock signal
rstn	in	1	Control	Asynchronous active low reset
up_down	in	1	Control	If 1, it will count up; if 0 it will count down.
load	in	1	Control	If 1, the counter will be loaded with input load
input_load	in	N	Data	Initial value of your counter
count_out	out	N	Data	Output of the counter
carry_out	out	1	Control	Output of the counter

```
module up_down_counter #(parameter N = 4)
    (input logic clk,
     input logic rst_n,
     input logic up_down,
     input logic load,
     input logic [N-1:0] input_load,
     output logic [N-1:0] count_out,
     output logic carry_out);

    // complete here

endmodule
```

## 1.4 QUESTION 4

Use a 4-bit counter as a building block to create a larger counter and implement a frequency divider. Cascade four 4-bit counters to construct a 16-bit counter. Then, modify the design to act as a frequency divider, alternating between division ratios of 18 and 866.

- Draw the RTL schematic of the design.
- Write the SystemVerilog code for it.
- Write a testbench to verify its functionality.

Module pinout

Name	Direction	Width	Description
<b>clk</b>	in	1	Clock signal
<b>rstn</b>	in	1	Asynchronous active low reset
<b>divider_out</b>	out	1	Divided signal

```

module frequency_divider (input logic clk,
                          input logic rst_n,
                          output logic divider_out);

    // complete here


endmodule

```

## 1.5 QUESTION 5

Use HDL to model an 8-bit register file with a depth of 16 rows. The registerfile has two read ports, each with its own address port. To write in a defined address of a registerfile (*write\_addr*), the *write\_en* should be active(=1). The behaviour of the registerfile is defined according to the following table. **The read is synchronized with the clock.**

- Draw the RTL schematic of the design.
- Write the SystemVerilog code for it.
- Write a testbench to verify its functionality.

Function	clk	rst_n	write_en	Memory	data_out_1	data_out_2
<b>Reset</b>	X	0	X	mem(all) = 0	0	0
<b>Standby</b>	X	1	X	No change	0	0
<b>Read</b>		1	X	No change	mem(read_addr_1)	mem(read_addr_2)

