Project: Modeling and Simulation with SV

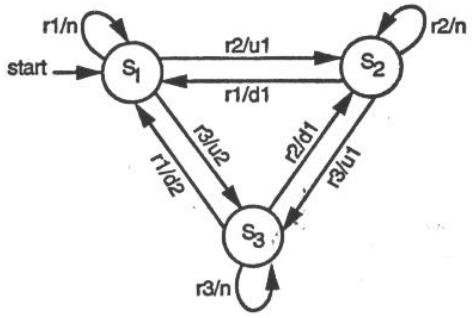
---

1) Write a Systemverilog model.
2) Simulate the design using an HDL simulator.
3) Design your test bench for each model and justify your test case choice and results in your report.

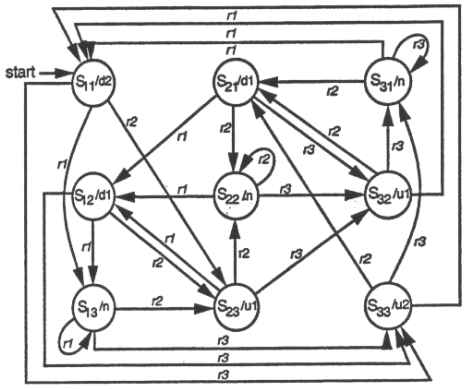
Turn in your report to the D2L **by the deadline** specified at the D2L Dropbox section.
Only one report is needed for each group.

Problem 1. Elevator Controller with three floors
Inputs={r1, r2, r3}: floor requests, Outputs={d2, d1, n, u1, u2}: instructions for the elevator:
di: go-down i floors, ui: go-up i floors.

a) Mealy Machine



b) Moore Machine:



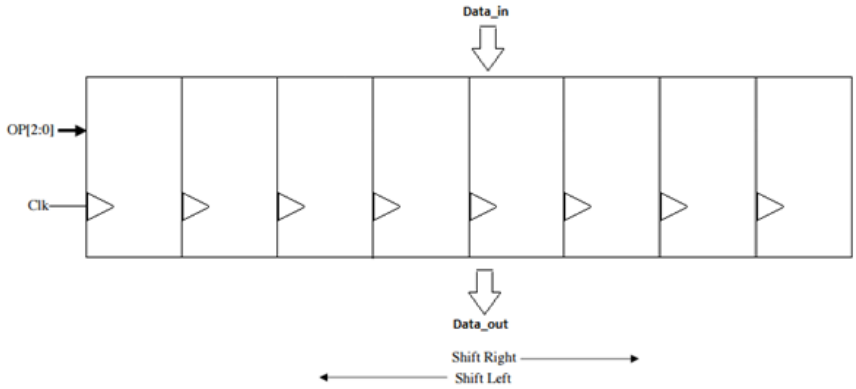
Task: Model and simulate the Mealy and Moore machines in SV, respectively.

Problem 2:
Design a sequencer detector. It compares the input sequence with its own built-in sequence bit by bit. If the input sequence is identical to the built-in sequence, it will display a message "matched", otherwise it will display a message "not-matched". When the mismatched bit occurs, the detector will return to the initial state and process the next input bit as the beginning of a new sequence.

Your built-in sequence is a 8-bit BCD code created by converting the last two digits of the PSU ID number of one member of your group. Model and simulate the detector by an FSM-based model in Systemverilog.

Problem 3: Consider the following design Model_1:

a) a 8-bit input data_in [7:0]
b) 8 register outputs data_out [7:0]
c) Clock(CLK)
d) a 3-bit operation code OP [2:0].



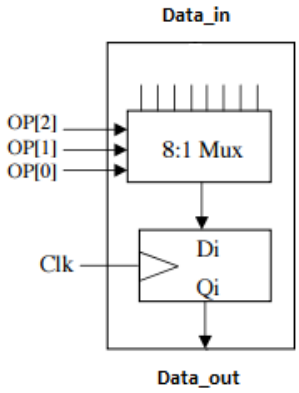
The function of the OP code is defined as:

- ✧ 000: Reset all registers to 0
- ✧ 001: Arithmetic shift right (shift right and keep the highest bit value)
- ✧ 010: Arithmetic shift left (shift left and fill the lowest bit with 0)
- ✧ 011: Shift right (shift right lowest bit wraps to the highest bit)
- ✧ 100: Shift left (shift left the highest bit wraps to the lowest bit)
- ✧ 101: Keep current registers' data
- ✧ 110: Default (You can define your own logic/arithmetic operations)

Tasks:

- ✧ Write and simulate a synthesizable 8-bit shifter register model in Systemverilog .

Problem 4: Assume that each slice of Model_1 (**Problem 3**) is built by an 8:1 Mux and a D flip-flop. The block diagram is shown as follows.



Tasks:
4.1. Model the single-bit shifter in SystemVerilog.
4.2. Write a structural model Model_2 to combine the single-bit shifters implemented by (4.1.) to realize the same functionality of Model_1.
4.3. Simulate the design using an HDL simulator.

Problem 5:
The FIFO is a type of memory that stores data serially, where the first word read is the first word that was stored. Write a Systemverilog model of a logical circuit (FIFO **Controller**) which controls the reading and writing of data from/into a FIFO.

The FIFO is a two-port RAM array having separate *read* and *write* data buses, separate *read* and *write* address buses, a *write* signal and a *read* signal. The size of the RAM array is 32 x 8 bits. Data is read from and written into the FIFO at the same rate (a very trivial case of the FIFO). The FIFO controller has the following input and output signals:

NAME	DIRECTION / SIZE	TYPE	DESCRIPTION
rst	Input / 1 bit	Active high	Asynch global reset
clk	Input	-	Controller clock
wr	Input / 1 bit	Active high	From external device wanting to write data into FIFO
rd	Input / 1 bit	Active high	From external device wanting to read data from FIFO
wr_en	Output / 1 bit	Active high	To FIFO as *write* signal
rd_en	Output / 1 bit	Active high	To FIFO as *read* signal
rd_ptr	Output / 5 bits	-	*read* address bus to FIFO
wr_ptr	Output / 5 bits	-	*write* address bus to FIFO
emp	Output / 1 bit	Active high	Indicates that FIFO is empty
full	Output / 1 bit	Active high	Indicates that FIFO is full

The read pointer **rd_ptr** contains the address of the next FIFO location to be read while the write pointer **wr_ptr** contains the address of the next FIFO location to be written. At reset, both pointers are initialized to point to the first location of the FIFO, **emp** is made high and **full** is made low. If an external device wishes to read data from the FIFO by asserting **rd**, then the controller asserts **rd_en** only if **emp** is deasserted. A similar logic exists for the write operation. The crux of this design is in determining the conditions which lead to the assertion/deassertion of the **emp** and **full** signals.
a) Write a Systemverilog model. b) Simulate the design using an HDL simulator.
Your testbench should contain the following test scenarios.
1. continue to write until full
2. continue to read until empty
3. Mixing read and write operations