Project: Modeling and Simulation with SV

---

a) Write a Systemverilog model.
b) Simulate the design using an HDL simulator.
c) 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.

Task 1. 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:

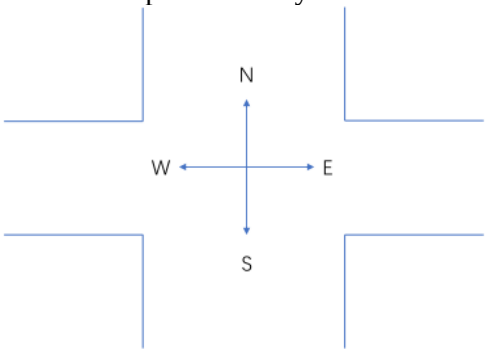
NAME	DIRECTION / SIZE	TYPE	DESCRIPTION
rst	Input / 1 bit	Active high	Asynch global reset
clk	Input	-	Controller clock
wr	Input / 1 bit	Active high	From external device wanting to write data into FIFO
rd	Input / 1 bit	Active high	From external device wanting to read data from FIFO
wr_en	Output / 1 bit	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.

Task 2: A Traffic Light Controller

This design represents a simple traffic light controller for a North-South and East-West intersection. The North-South is the main road, and is given the GREEN light unless a sensor on the East-West Street is activated. When that occurs, and the North-South light was GREEN for sufficient time, then the light will change to give way to the East-West traffic.

The design also takes into account emergency vehicles that can activate an emergency sensor. When the emergency sensor is activated, then the North-South and East-West lights will turn RED, and will stay RED for a minimum period of 3 cycles.



The model has the following interfaces. The model makes use of the following type definition;

```
typedef enum { OFF,             // power off
               RED,             // red state
               YELLOW,          // yellow state
               GREEN,// green state
               PRE_GREEN} // state before green
               lights_t;

module trafficlight
(
output lights_t ns_light, // North/South light status, Main road
output lights_t ew_Iight,         // East/West light status
input ew_sensor,                        // East/West sensor for new car
input emgcy_sensor,             // emergency sensor
input reset_n,                          // synchronous reset
input clk                               // master clock
);
timeunit 1ns;
timeprecision 100ps;
parameter FAIL = 1'b0;
```

Since the traffic lights are timed, two timers are used in this first implementation:

```
logic [1:0] ns_green_timer;     // timer for NS light in Green
logic [1:0] ew_green_timer;     // timer for EW light in GREEN
```

The architecture includes two FSMs and the timers. The operation of the machine is very simple:

1. North-South remains GREEN unless one of the sensors is activated.
2. If the North-South light is RED and the North-South GREEN timer is 3, then the light will switch to GREEN.
3. If the North-South light is YELLOW, it will switch to RED.

4. If the North-South light is GREEN, and the emergency sensor is activated, the light will switch to YELLOW. Also, if the North-South timer is 3, and the East-West sensor is activated, the North-South light will switch to YELLOW.
5. The North-South GREEN timer is reset to zero at reset, or when the North-South light is YELLOW, Otherwise, it increments at every clock until It reaches the maximum count of 3.
6. The East-West light switches from RED to a PRE_GREEN state (to allow the North-South light to go to YELLOW) if the North-South timer is 3 and the East-West sensor is activated.
7. If the East-West is PRE_GREEN; it will switch to GREEN at the next clock.
8. If the East-West light is YELLOW, it will switch to RED.
9. If the East-West is GREEN and either the emergency sensor is activated or the East-West timer reaches a count of 3 then it switches to YELLOW.
10. The East-West GREEN timer is reset to zero at reset or when the East-West light is YELLOW. Otherwise, it increments at every clock until it readies the maximum count of 3.

a) Write a Systemverilog model. b) Simulate the design using an HDL simulator.

Task 3:

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. Implement the detector by an FSM-based model in Systemverilog.