# Project: Modeling and Simulation with SV

---

**Tasks:**
For the following designs, 1) Write a Systemverilog model. 2) Simulate the design using an HDL simulator.
**The Report:**
1) Write **your NAME** and your email address on the first page of your report.
2) Turn in your report to the D2L **by the deadline** specified at the D2L Dropbox section.
3) Only one report is needed for each group.
4) Your report must contain at least:
   - Design specification: explain the details about your project design, etc.
   - LEGIBLE results for all Test benches (test cases).
   - The Systemverilog model code.

---

**D1:** In digital communication, a special synchronization pattern, known as a preamble, is used to indicate the beginning of a packet. For example, the Ethernet I1 preamble includes eight repeating octets of "10101010". We wish to design an FSM that generates the "10101010" pattern. The circuit has an input signal, start, and an output, data-out. When start is 'l', the "10101010" will be generated in the next eight clock cycles. The output width is 1 bit. It is not a vector. Output will keep sending the pattern if the condition is satisfied. When input signal is interrupted at any stage, FSM should reset to the initial state. Write a Systemverilog model. Your test-bench should show the interrupt case at least 2 times in simulation. "Start" signal's sequence can be like '1111000111111100' or '1111111111110000'.

**D2:** 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.

**D3:** 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.