Lab 2: Design Practice and Guide

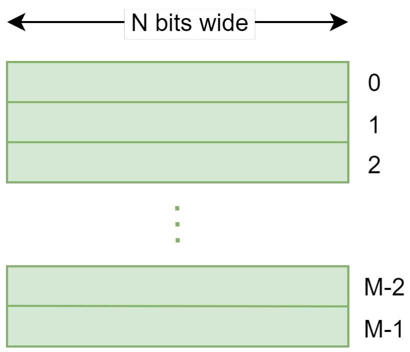
You don't need to submit this design practice.

Objective

1. Getting familiar with sequential Verilog behavior modeling.
2. Practicing finite-state machines (FSMs) in Verilog.
3. Practicing the memory data structure in Verilog.
4. Practicing Verilog testbench design.

Guide

- **Memory Array**
  - Verilog uses a set of registers to simulate the memory array.
  - Declaration:
    - E.g., to implement a memory to store M data with each of N bits, we can use reg [N-1:0] output_tmp [M-1:0];



  - Access:
    - Process the register to store the address of the data you want to access.
    - E.g., we use offset_cnt register to access the specific data: output_tmp[offset_cnt] <= nxt_output_tmp[offset_cnt];

- **Testbench**
  - A testbench is used to simulate and verify your design without the need for any physical hardware.
  - Please refer to the Practice_1_t.v and Practice_2_t.v for more details.

- **Concatenation Operator**
  - In Verilog, we can use {} to concatenate two or more vectors as one.
  - E.g., a is an 8-bit data, while b and c are 4-bit data. Under some circumstances, we

can use concatenation techniques such as a = {b, c} or a = {4’b0, b}.

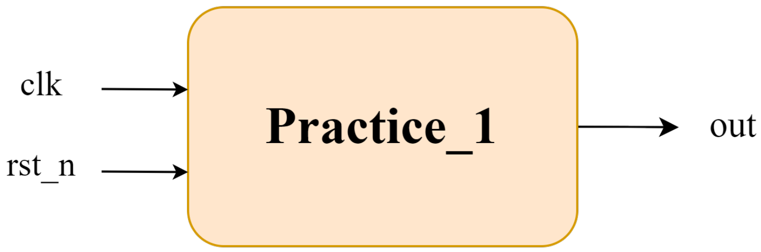
- **Module Instantiation**
  - Bigger and more complex designs are built by integrating modules in a hierarchical manner. Modules can be instantiated within other modules; ports of these instances can be connected with other signals inside the parent module.
    - E.g.,

```
module My_Module(in, out);
input in;
output wire out;
wire a, b; // The wires that connect the two module
    // Specify which module you want to instantiate and give it a name
    My_Second_Module msm(in, a, b);
endmodule

module My_Second_Module(in, a, b);
input in;
output reg a, b;
endmodule
```

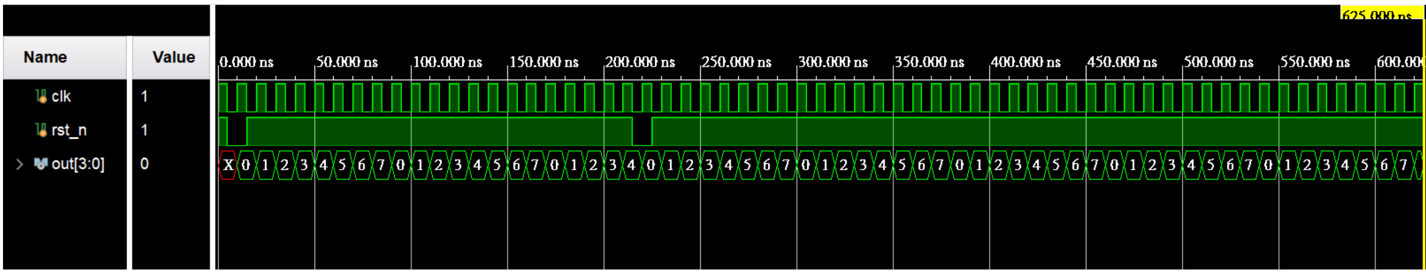
Practice

1. Design a 4-bit counter that counts from 0 to 7.

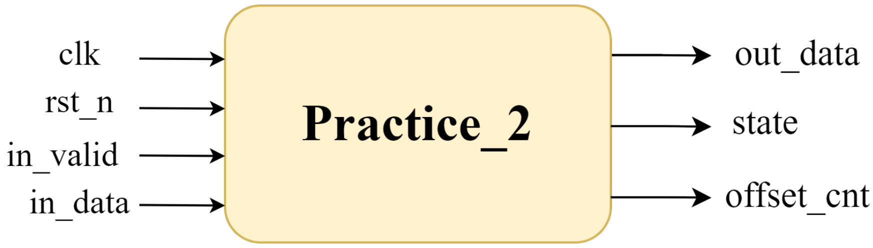


- clk:
  - Positive edge triggered.
- rst_n:
  - Synchronous negative reset; if rst_n == 1’b0, reset the out to 4’d0. The counter will start to count after leaving the reset operation.
- For each clock cycle, the counter will increase its original value by 1 until it reaches its max value of 7.
- When the counter value reaches 7, the next counting value is 0. Repeat the counting process, e.g., 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, …, 7, 0, 1, 2, …
- Note that the counting rules are different from the rules in Problem lab2_1.
- Refer to the Practice_1.v file for more hints.

- Reference waveform for Practice_1:

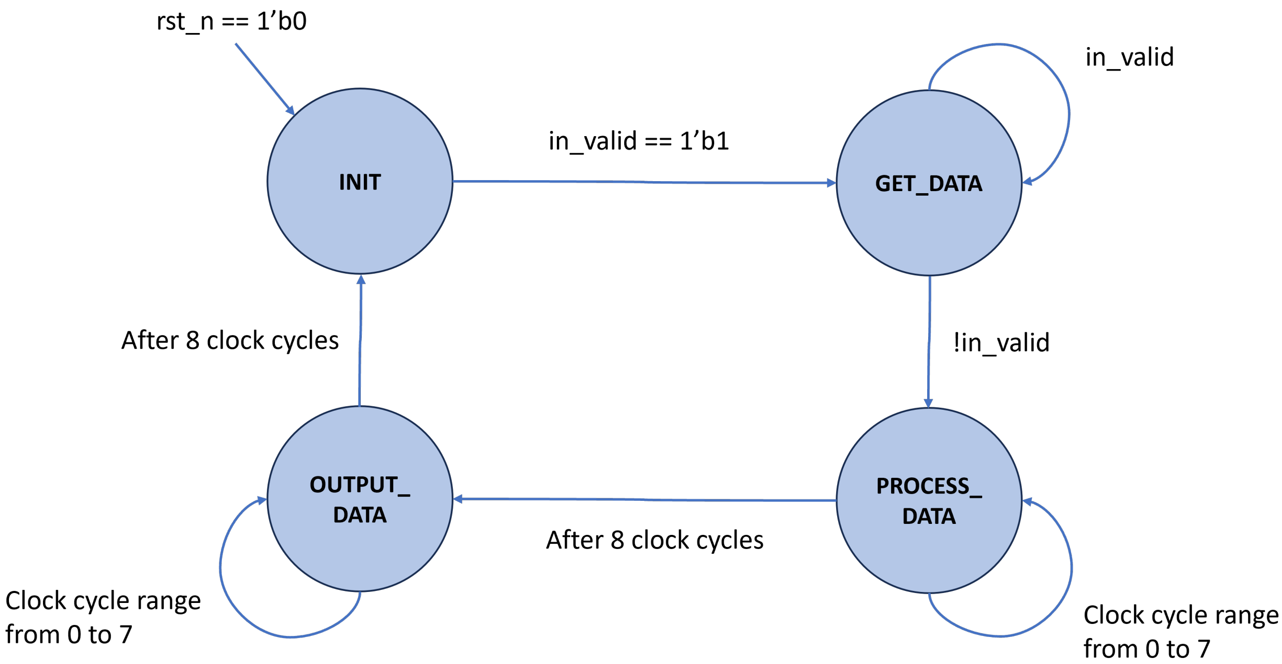


2. Design a simple FSM



- in_data
  - in_data is valid only when in_valid is high.
  - $6'd0 \leq in\_data \leq 6'd60$
- offset_cnt
  - Reset to 0 when entering the new state.
  - Start counting during GET_DATA, PROCESS_DATA, and OUTPUT_DATA state.
  - This is similar to the counter that was designed in Practice_1.
- out_data
  - The data you need to output in the output state.
- state
  - Represent the current state.

- Follow the state diagram below to implement your design.



- State description:
  - **INIT:**
    - ➢ After being reset, the FSM will be in the INIT state waiting for the in_valid to be high.
    - ➢ Whenever the in_valid becomes high, the state machine will enter the GET_DATA state to process the input data.

  - **GET_DATA**
    - ➢ In this state, in_data will be fetched at each clock cycle.
    - ➢ in_valid signal will be high for exactly 8 clock cycles. I.e., a total of eight data will be collected.
    - ➢ Use the counter in the previous question to help you sequentially store the input data in a 1D array.

  - **PROCESS_DATA**
    - ➢ In this state, increase the data value you collected by 1'b1.
    - ➢ Since you have 8 data to process, there will be 8 clock cycles in this state.
    - ➢ After 8 clock cycles, go to the OUTPUT_DATA state.

  - **OUTPUT_DATA**
    - ➢ Output the data (out_data) to the test stimulus (testbench) for verification.
    - ➢ Since you have 8 data to output, there will be 8 clock cycles in this state.
    - ➢ After outputting these 8 data, go back to the INIT state.

- Refer to the Practice_2.v file for more hints.
- Please note that this practice is highly associated with Lab_2_2.
- Reference waveform for Practice_2: