Lab 2: Parameterized Ping-Pong Counter and Data Encoding

Submission Due Dates:

Demo: 2023/09/26 17:20
Source Code: 2023/09/26 18:30
Report: 2023/10/01 23:59

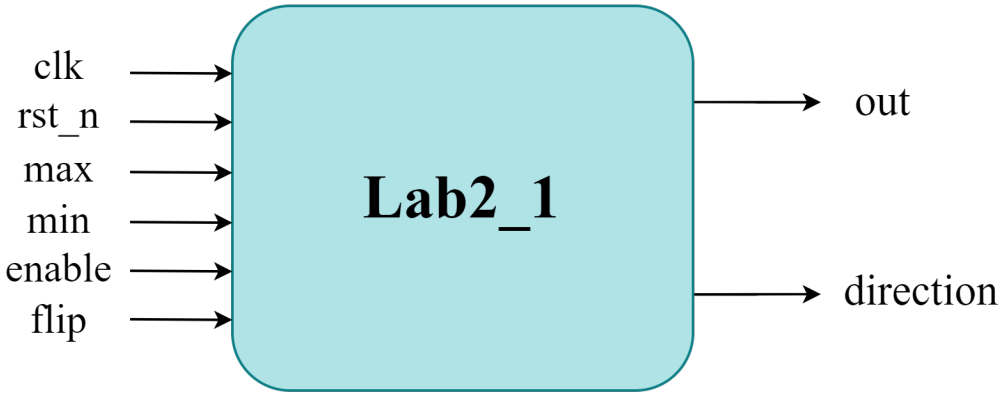
Objective

- Getting familiar with Verilog sequential behavior modeling.
- Practicing counter and FSM designs in Verilog.
- Practicing memory data structure in Verilog.

Action Items

1. lab2_1.v (45%)

Design a 4-bit parameterized ping-pong counter that can count in two directions with max and min signals:



a. IO List and Specification

Input Signals	Bit Width
clk	1
rst_n	1
max	4
min	4
enable	1
flip	1

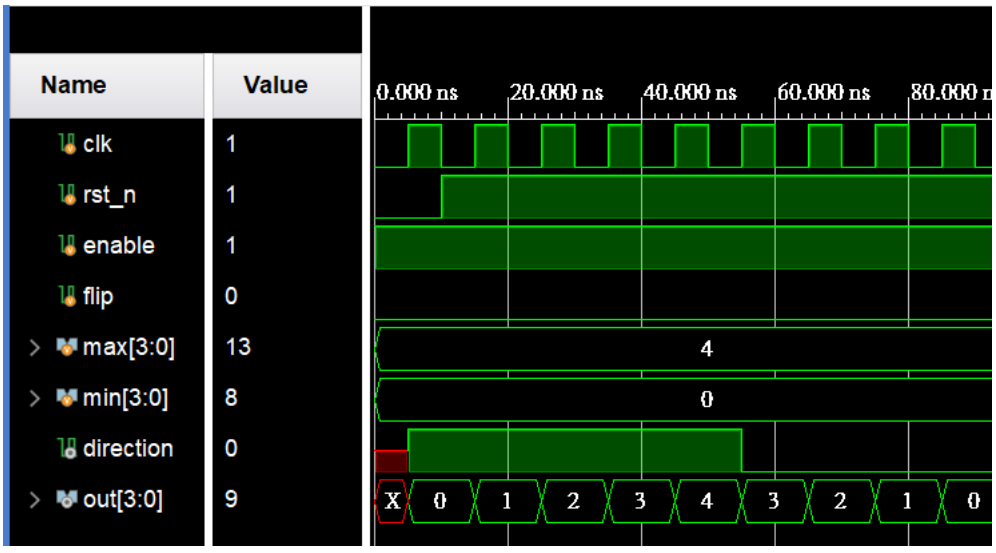
Output Signals	Bit Width
out	4
direction	1

- clk:
  - Positive edge triggered.
- rst_n:
  - Synchronous negative reset; when rst_n == 1'b0, reset the out to min; the counter will count up after leaving the reset operation; direction is set to 1'b1.
- max and min:
  - All the "out" from the counter must fall within the range between the minimum and maximum values ($min \leq out \leq max$).
  - If out > max, max $\leq$ min, or out < min, the counter should hold its current value.
  - Note that max and min values may change during the counting process.
  - If max == min == output, hold the output and direction.
- enable:
  - If enable == 1'b1, the counter begins its operation. Otherwise, the counter should hold its current value.
- flip:
  - When flip == 1'b1, the counter changes its direction.
- direction:
  - When counting up, direction = 1'b1. When counting down, direction = 1'b0.
- If the value of the counter is out of range, hold the out value and its direction.
- E.g.: max = 5, min = 0, enable = 1'b1, flip = 1'b0

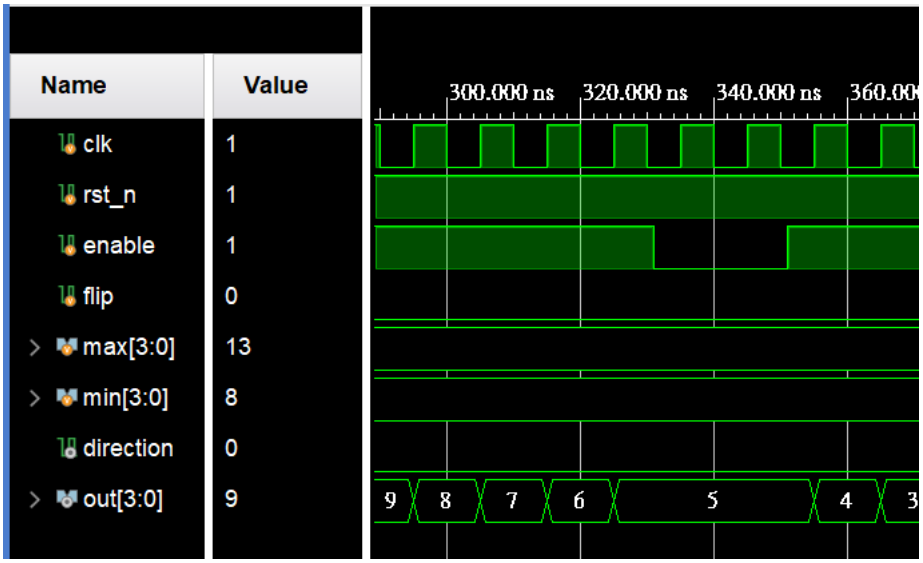
out	0	1	2	3	4	5	4	3	2	1	0	1	2	3
direction	1	1	1	1	1	1	0	0	0	0	0	1	1	1

b. Waveform examples

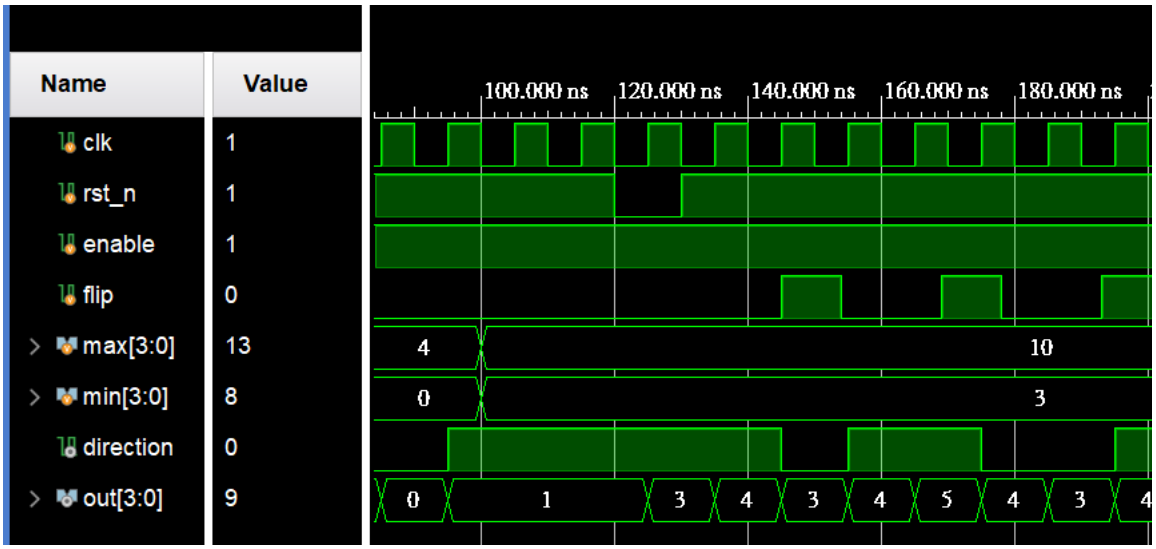
- When enable == 1'b1, flip == 1'b0

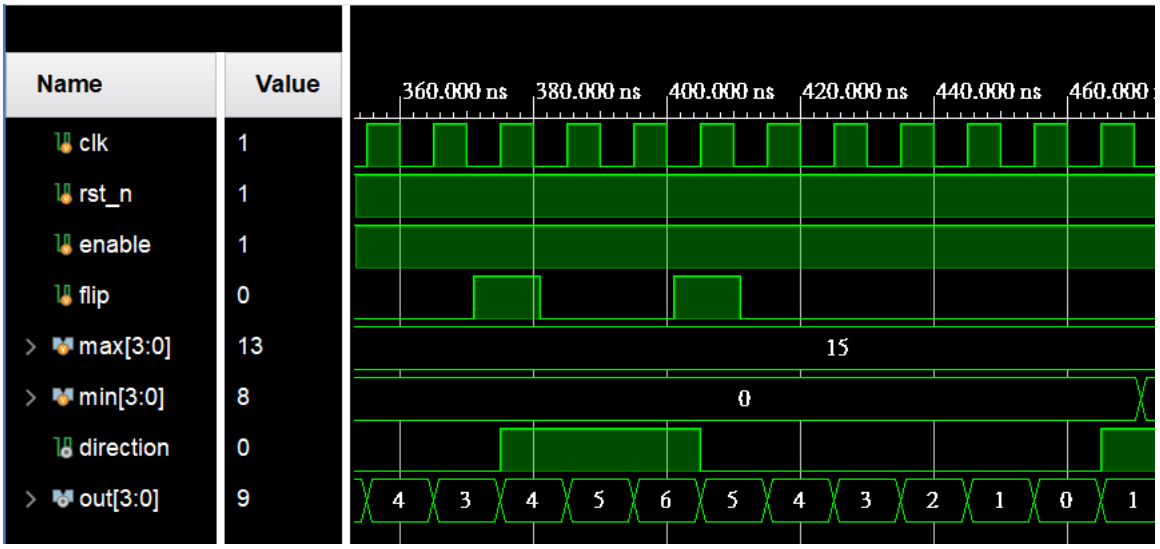


- When enable == 1'b0



- Example of how rst_n, flip, min, max affect the direction and out





c. You must use the following template for your design

```
`timescale 1ns/1ps
```

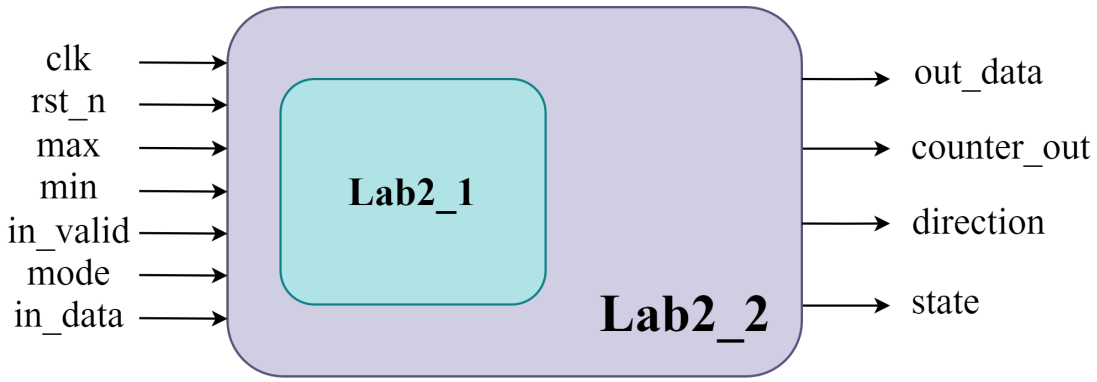
```
module Parameterized_Ping_Pong_Counter (
        input clk,
        input rst_n,
        input enable,
        input flip,
        input [3:0] max,
        input [3:0] min,
        output direction,
        output [3:0] out
);

    // Output signals can be reg or wire
    // add your design here

endmodule
```

2. lab2_1_t.v (15%)

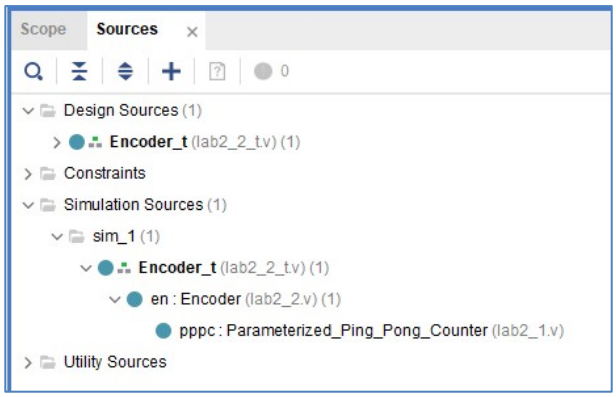
Write a testbench (lab2_1_t.v) by yourself to verify the design. Remember to test **boundary conditions**. In your report, please explain how you built your testbench.

3. lab2_2.v (25%)

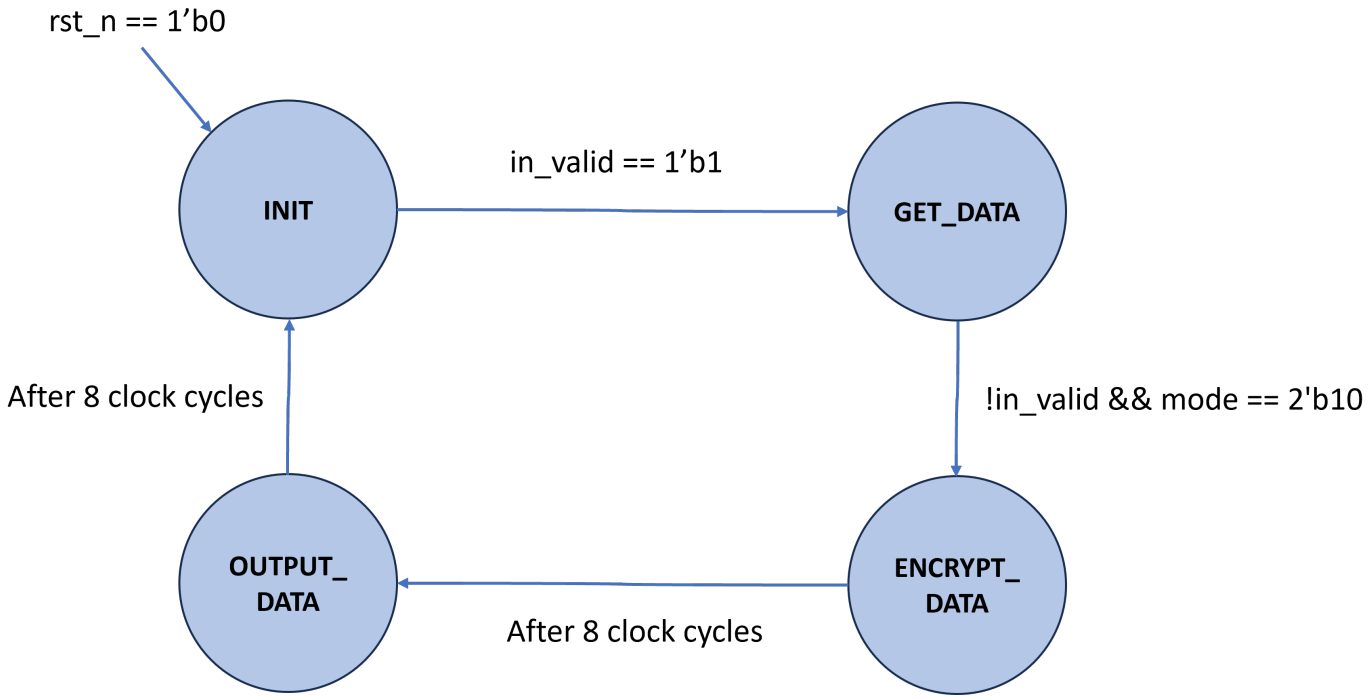
Caesar cipher and Hamming codes encoding



Use the counter that you design in the previous question, and don't copy-and-paste the lab2_1 module in your lab2_2.v file. That is, you will see the hierarchy as the following figure.



Write a Verilog module that can apply the following encryption algorithm and error correction code. **This module should have a finite state machine with four states:**



- **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.
  - In this state, mode can be any value. However, it can only go to the ENCRYPT_DATA

state when mode == 2'b10.
- Use the counter in Practice to help you sequentially store the input data in a 1D array.

- **ENCRYPT_DATA**
  - Follow the encryption rules (Caesar Cipher and Hamming Codes) to process the input data you stored in the GET_DATA state.
  - In this state, the mode signal will maintain a value of 2'b10.
  - Encrypt one data in each clock cycle and store the encrypted data in a 1D array.
  - Since you have 8 data to encode, there will be 8 clock cycles in this state.
  - After 8 clock cycles, go to the OUTPUT_DATA state.

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

a. IO List

Input Signals	**Bit Width**
clk	1
rst_n	1
max	4
min	4
in_valid	1
mode	2
in_data	8

Output Signals	**Bit Width**
out_data	12
counter_out	4
direction	1
state	3

b. Rules

- in_data:
  - The length of the in_data is 8; $4'd1 \leq in\_data \leq 4'd255$
- Each state other than INIT should have 8 clock cycles.
- The clk, rst_n, max and min follow the same rules as mentioned in Question 1.
- in_valid:

- If in_valid == 1'b1, get the input data, in_valid will be high only in the GET_DATA state.

- mode:
  - When mode == 2'b00, flip signal of the counter is disabled.
  - When mode == 2'b01, flip signal of the counter is enabled; the direction of the counter should be changed accordingly.
  - When mode == 2'b10, hold the current value of the counter (out, direction)
- Encryption process:
  Here, we will design a revised and simplified version of the Caesar cipher.
  - The encryption process uses the value of the counter through the following equation:
    $$e_i = (d_i + n) \% 256$$
    where e stands for the encrypted message, d is the input data, n is the value we obtained from the counter, and i is the index of current data, $0 \leq i \leq 7$.
  - The encryption process will begin only when in_valid is low and mode == 2'b10.
  - E.g.:
    $d = 248, 249, 250, \ldots, 253, 254, 255;$
    $n = 3;$
    $e = 251, 252, 253, \ldots, 0, 1, 2$
  - After applying the encryption equation, we need to use Hamming codes to construct error correction code. For the encoding rules, please refer to the following **c. Error correction code**.

c. Error correction code

- Introduction:
  Data transmission in real life will suffer from data corruption. To address this problem, scientists have come up with various error detection and correction algorithms.

  In this lab, we will use Hamming codes to encode the encrypted data. This method involves adding redundant bits to the original data and using the parity check to detect and correct the error bits. In this lab, we use even parity to construct the error correction code.

  For more information, please check the following link:
  Hamming code: https://en.wikipedia.org/wiki/Hamming_code
  Parity check: https://en.wikipedia.org/wiki/Parity_bit

- Hamming Code encoding rules:
  As in the following table, the Hamming code adds 4 redundant bits to the 8-bit data,

where $r1, r2, r3, r4$ are the redundant bits and $d0, d1, d2, d3, d4, d5, d6, d7$ are the data bits.

Bit	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1
Notation	d7	d6	d5	d4	r4	d3	d2	d1	r3	d0	r2	r1

- 4 redundant (parity) bits are added as the following equations:
  - $r1 = B3 \oplus B5 \oplus B7 \oplus B9 \oplus B11$
  - $r2 = B3 \oplus B6 \oplus B7 \oplus B10 \oplus B11$
  - $r3 = B5 \oplus B6 \oplus B7 \oplus B12$
  - $r4 = B9 \oplus B10 \oplus B11 \oplus B12$

For example, an input data $\{101\}_{10}$ can be represented as $\{01100101\}_2$.

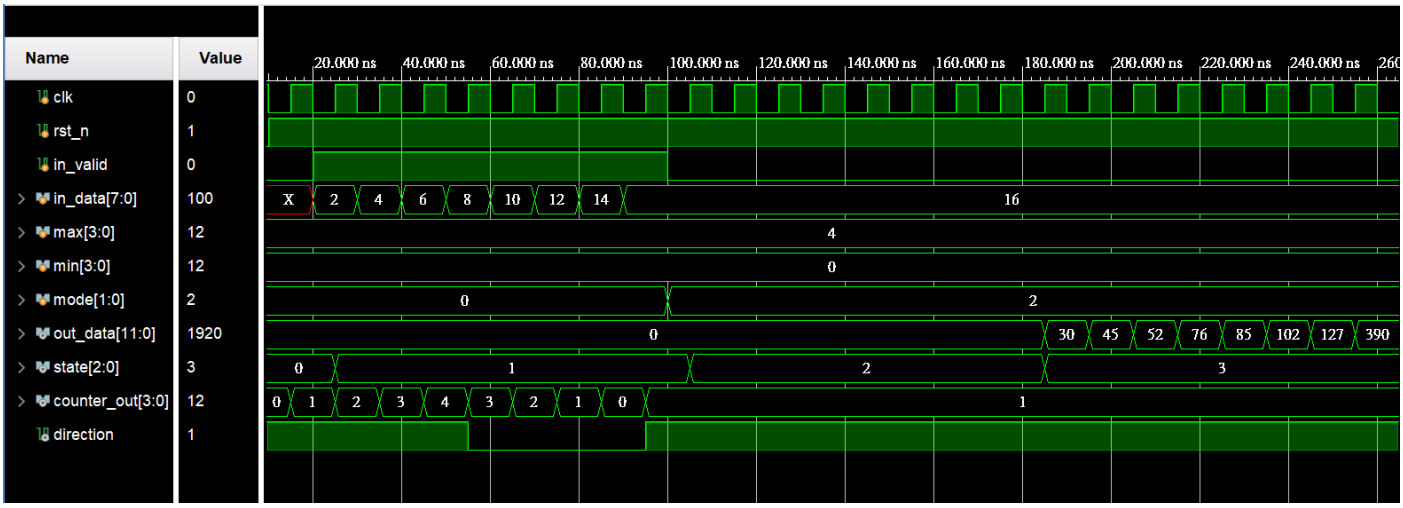
Bit	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1
Notation	d7	d6	d5	d4	r4	d3	d2	d1	r3	d0	r2	r1
Data	0	1	1	0	?	0	1	0	?	1	?	?

After applying Hamming codes:

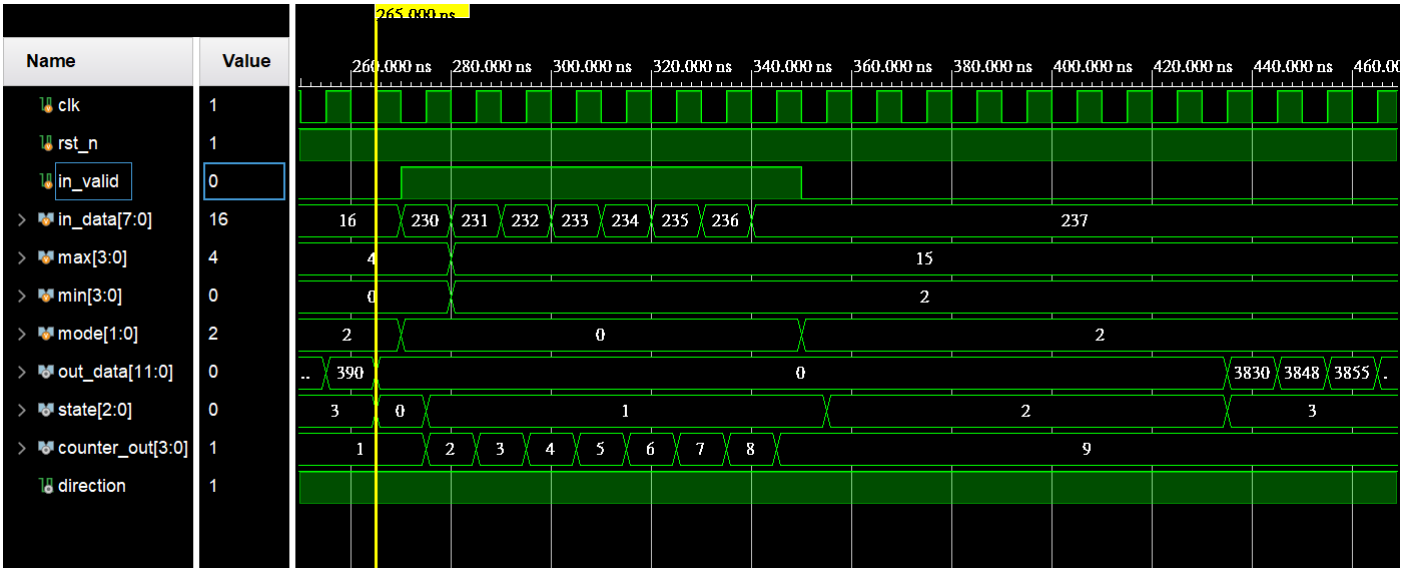
Bit	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1
Notation	d7	d6	d5	d4	r4	d3	d2	d1	r3	d0	r2	r1
Data	0	1	1	0	0	0	1	0	1	1	0	0

d. Waveform examples

- In the following example, when in_valid is high, in_data 2, 4, 6, 8, 10, 12, 14 are received, and after 8 cycles of encoding, we will get the out_data 30, 45, 52, 76, 85, 102, 127, 390.



- Set the out_data to 0 when it is not in the OUTPUT_DATA state.



e. You must use the following template for your design

```
`timescale 1ns/1ps
module Encoder (
    input clk,
    input rst_n,
    input [3:0] max,
    input [3:0] min,
    input in_valid,
    input [1:0] mode,
    input [7:0] in_data,
    output [11:0] out_data,
    output [2:0] state,
    output [3:0] counter_out,
    output direction
    );

    // Output signals can be reg or wire
    // add your design here

endmodule
```

4. lab2_2_t.v

Write a testbench (lab2_2_t.v) by yourself to verify the design. Remember to test **boundary conditions**. In your report, please explain how you built your testbench.

5. Questions and Discussion

Please answer the following questions in your report.

1. In this lab, our reset signal is a synchronous reset. What if it is an asynchronous reset? And how to modify your design to implement an asynchronous reset?
2. Why do we need both combinational circuits and sequential circuits in our design? What are the differences between a combinational circuit and a sequential circuit? Please explain how each of them works in detail.

6. Hint

- You can use the storage/memory (in this case, flip-flops) to store input/output data.
- You can use Practice as a template to design your lab2.

7. Report Guidelines

Your report should include but not limit to the following items.

A. Lab Implementation (35%)

You may elaborate on the following.

1. Block diagram of the design with explanation.
2. Partial code screenshot with the explanation: you don't need to paste the entire code into the report. Just explain the kernel part.
3. How you test your design: waveform screenshot with explanation.
4. Event-based FSM with the explanation

B. Questions and Discussions (50%)

Provide your answer to the Questions and Discussions in the lab assignment.

C. Problem Encountered (10%)

Describe the problems you encountered, solutions you developed, and the discussion. Explaining them with code segments or diagrams is recommended.

D. Suggestions (5%)

Any suggestions for this course are more than welcome.
(If not, you may also post a joke (a funny one, please). It is not mandatory and has nothing to do with the grading. But it would undoubtedly amuse us. 😉)

Attention

- ✓ DO NOT copy-and-paste code segments from the PDF materials to compose your design. It may also paste invisible non-ASCII characters, leading to hard-to-debug syntax errors.
- ✓ You should hand in four source files, including **lab2_1.v, lab2_1_t.v, lab2_2.v and lab2_2_t.v**. Upload each source file individually. DO NOT hand in any compressed ZIP files, which will be considered an incorrect format.
- ✓ You should also hand in your report as **lab2_report_StudentID.pdf** (i.e., lab2_report_111456789.pdf).
- ✓ You should be able to answer questions from TA during the demo.

- ✓ Before the simulation, you may need to change the runtime to 10000ns (or a large enough period) in “Simulation Settings”.