



Welcome to The Hardware Design & Lab!

Fall 2024 Lab 3: Sequential Circuits

Prof. Chun-Yi Lee

Department of Computer Science
National Tsing Hua University

Agenda

- Lab 3 Outline
- Lab 3 Basic Questions
- Lab 3 Advanced Questions



Lab 3 Outline

- Basic questions (1.5%)
 - Individual assignment
 - Due on **10/3/2024. In class.**
 - Only demonstration is necessary. Nothing to submit.
- Advanced questions (5%)
 - Group assignment
 - EEClass submission due on **10/10/2024. 23:59:59.**
 - Demonstration on your FPGA board (**In class**)
 - Assignment submission (**Submit to eeclass**)
 - Source codes and testbenches
 - Lab report in PDF

Lab 3 Rules

- Please note that grading will be based on **NCVerilog**
- You can use **ANY** modeling techniques
- If not specifically mentioned, we assume the following SPEC
 - **clk** is **positive edge triggered**
 - Synchronously reset the Flip-Flops when **rst_n == 1'b0, if there exists one rst_n signal in the specification**

Lab 3 Submission Requirements

- Source codes and testbenches
 - Please follow the templates **EXACTLY**
 - We will test your codes by TAs' testbenches
- Lab 3 report
 - Please submit your report in a single **PDF** file
 - Please **draw** the block diagrams of your designs using **software**
 - Please **explain** your designs in detail
 - **Please explain how you test your design**
 - Please describe the contributions of each member
 - What you have **learned** from Lab 3

Agenda

- Lab 3 Outline
- **Lab 3 Basic Questions**
- Lab 3 Advanced Questions

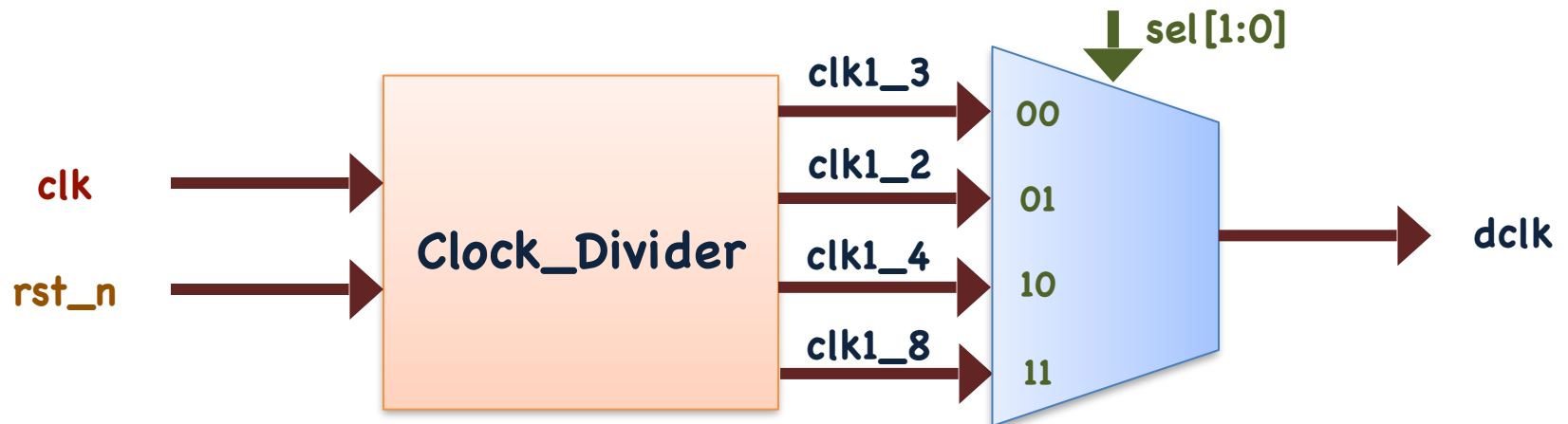
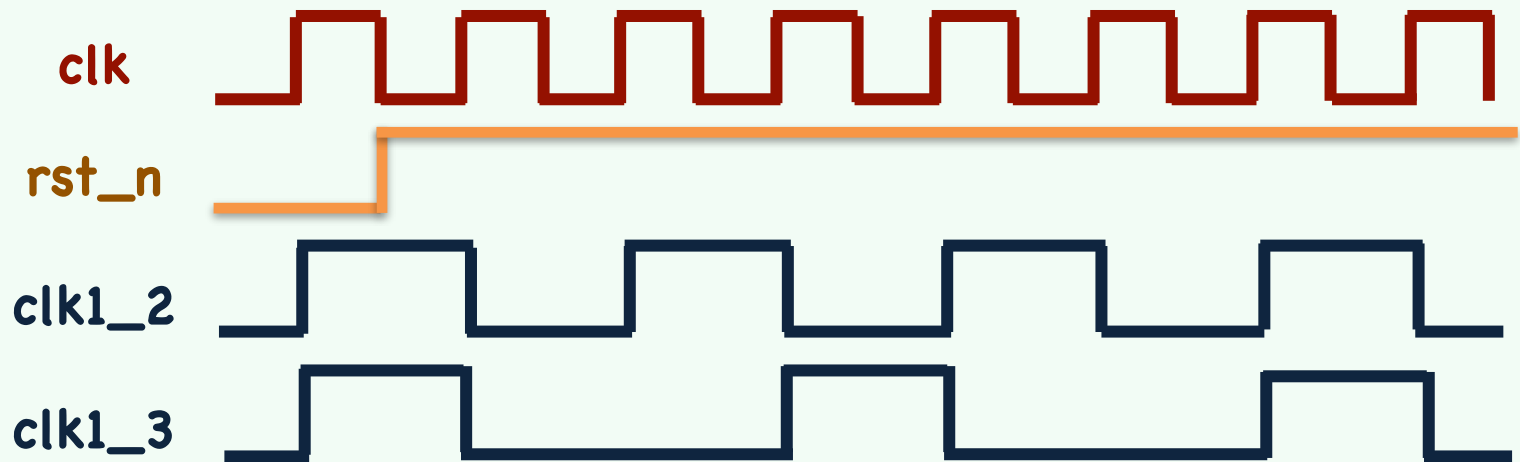


Basic Questions

- Individual assignment
- Verilog questions (due on 10/3/2024. In class.)
 - Clock Divider
 - 128 x 8 Memory Array
- Demonstrate your work by waveforms

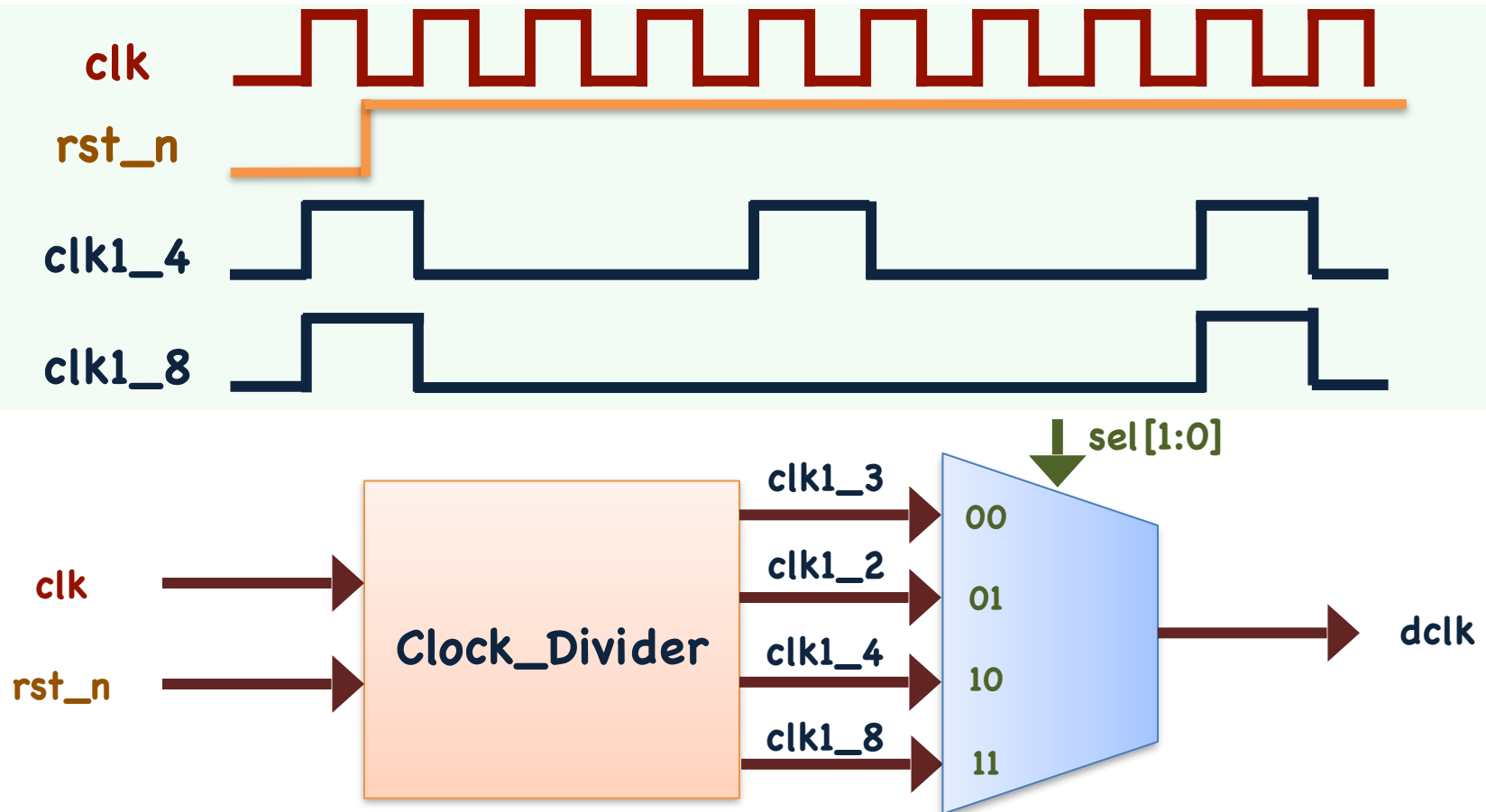
Verilog Basic Question 1

- Clock Divider
 - **sel[1:0]** and the mux are combinational, not triggered by **clk**
 - Outputs: **clk1_2**, **clk1_3**, **clk1_4**, **clk1_8**, **dclk**



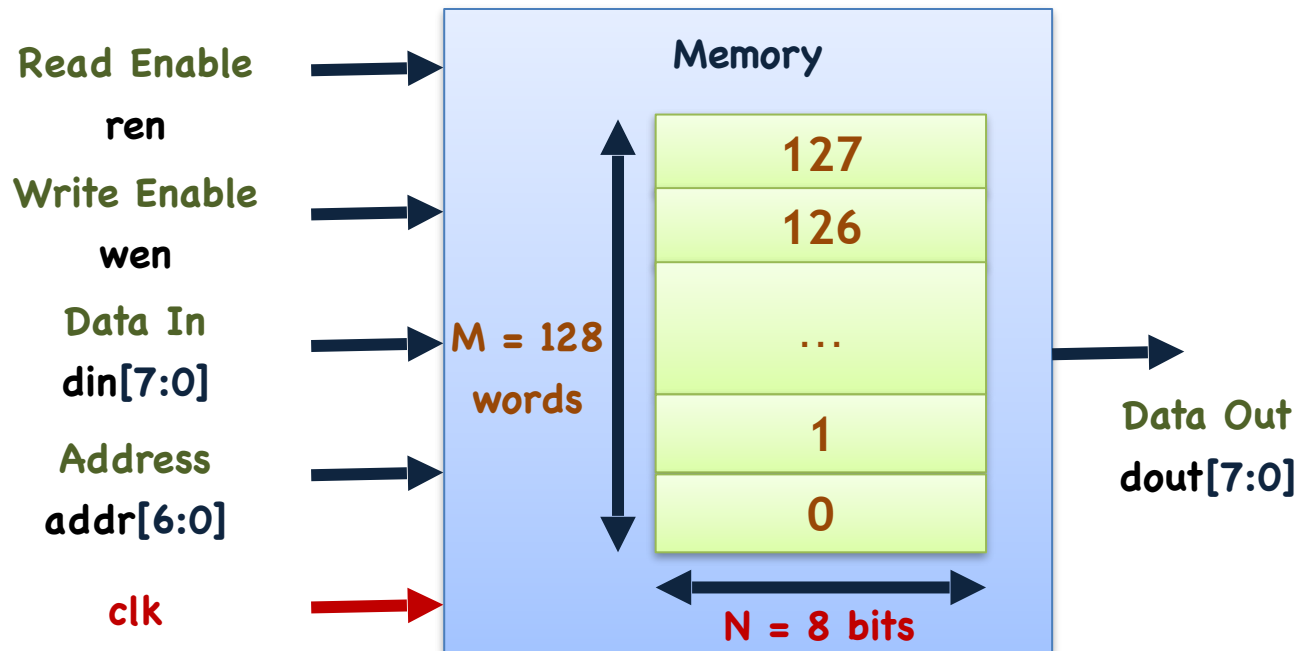
Verilog Basic Question 1(Con't)

- Clock Divider
 - **sel[1:0]** and the mux are combinational, not triggered by **clk**
 - When **rst_n == 1'b0**, all signals out the clock divider are **one**
 - Outputs: **clk1_2**, **clk1_3**, **clk1_4**, **clk1_8**, **dclk**





Verilog Basic Question 2

- 128 x 8 Memory Array **Memory**
- $M = 128$, $N = 8$
 - Inputs: **clk**, **ren**, **wen**, **addr[6:0]**, **din[7:0]**
 - Outputs: **dout[7:0]**

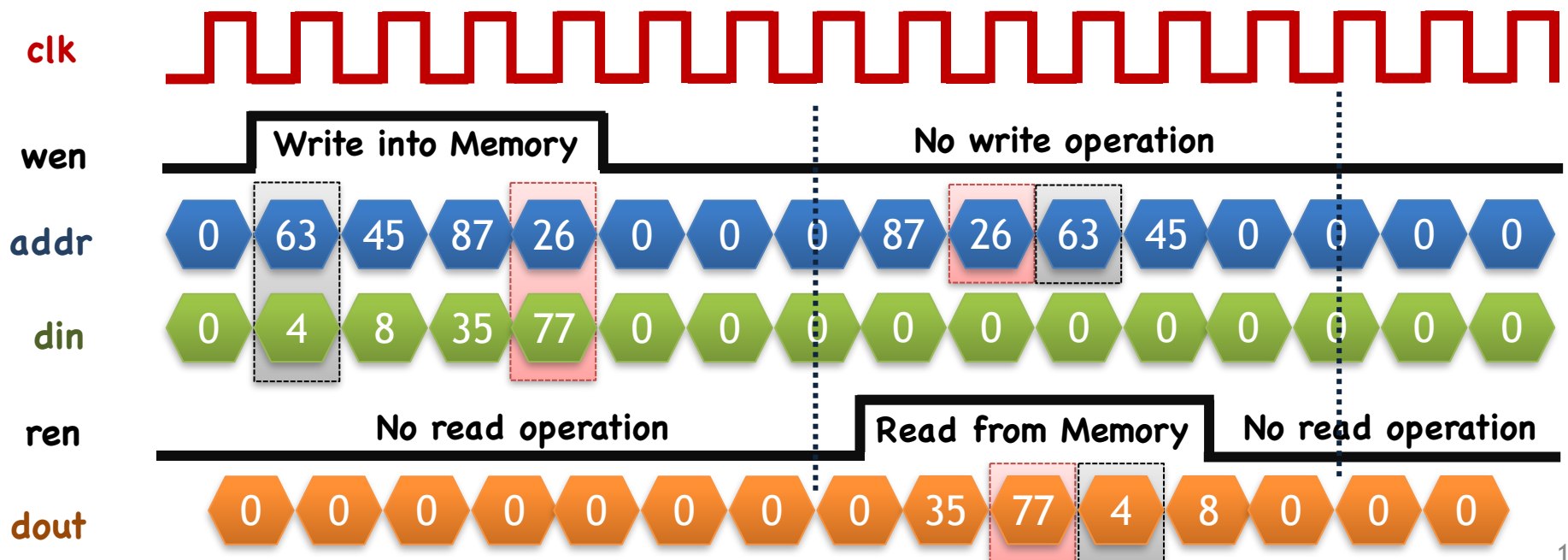


Note: Memory Array in Verilog

- A collection of registers in Verilog to mimic memory arrays
 - In reality, it is **NOT** made from registers
 - Real memory is made from SRAMs or DRAMs
- Declaration
 - Similar to regular reg arrays
 - `reg [N-1:0] Your_Memory [M-1:0];`
 -  **N bits per word**
 -  **M words**
- Access
 - Use your address register **ADDR**
 - E.g., `One_word[N-1:0] = Your_Memory[ADDR]`
 - If your **M** is **256**, you only need **8 bits** for **ADDR** ($2^8 = 256$)

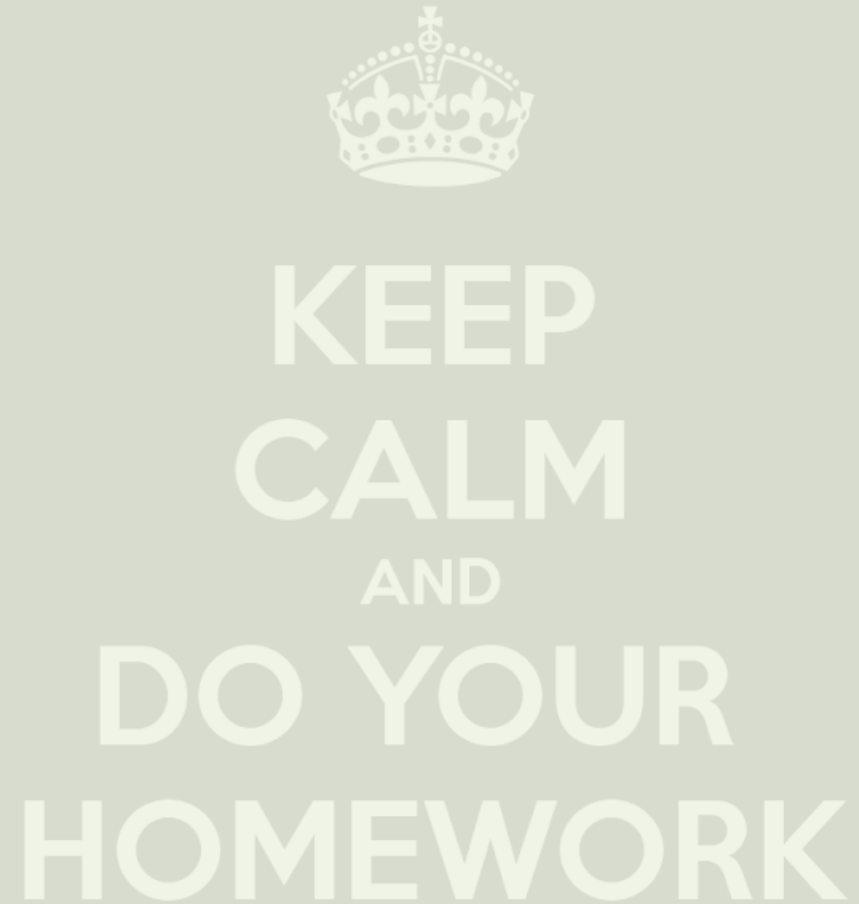
Verilog Basic Question 2 (Con't)

- Specification
 - When **wen** == 1'b1, write **din** to **Memory[addr]**
 - When **ren** == 1'b1, output **Memory[addr]** to **dout**; otherwise **dout** = 8'd0
 - **If both are 1, do only the read operation**
 - **Memory** does not need to be reset



Agenda

- Lab 3 Outline
- Lab 3 Basic Questions
- **Lab 3 Advanced Questions**



Advanced Questions

- Group assignment
- Verilog questions (due on 10/10/2024. 23:59:59.)
 - **Necessary:** 4-bit Ping-Pong Counter
 - **Necessary:** First-In First Out (FIFO) Queue
 - **Necessary:** Multi-Bank Memory
 - **Necessary:** Round-Robin FIFO Arbiter
 - **Necessary:** 4-bit Paramterized Ping-Pong Counter
- FPGA demonstration (due on 10/24/2024. In class.)
 - **Necessary:** 4-bit Paramterized Ping-Pong Counter on FPGA

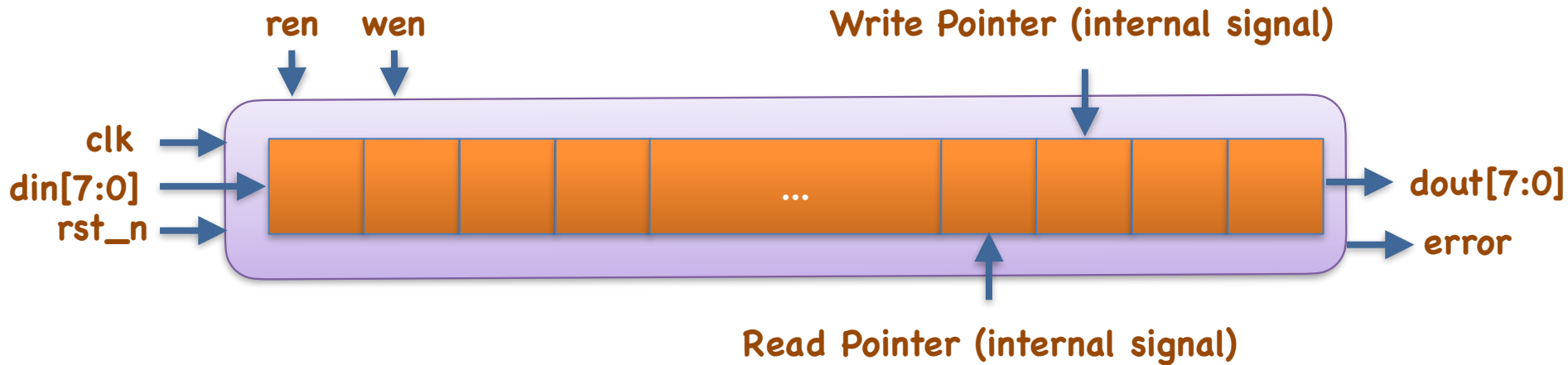
Verilog Advanced Question 1

- Design a 4-bit Ping-Pong Counter
 - **out:** 0,1,2,...,13,14,15,14,13,...,2,1,0,1,2,...
 - **direction:** 1,1,1,...,1, 1, 1, 0, 0,..., 0,0,0,1,1,...
- SPEC
 - When **rst_n** == 1'b0, the counter resets its value to 4'b0000, and the **direction** to 1'b1
 - When **enable** == 1'b1, the counter begins its operation. Otherwise, the counter holds its current value



Verilog Advanced Question 2

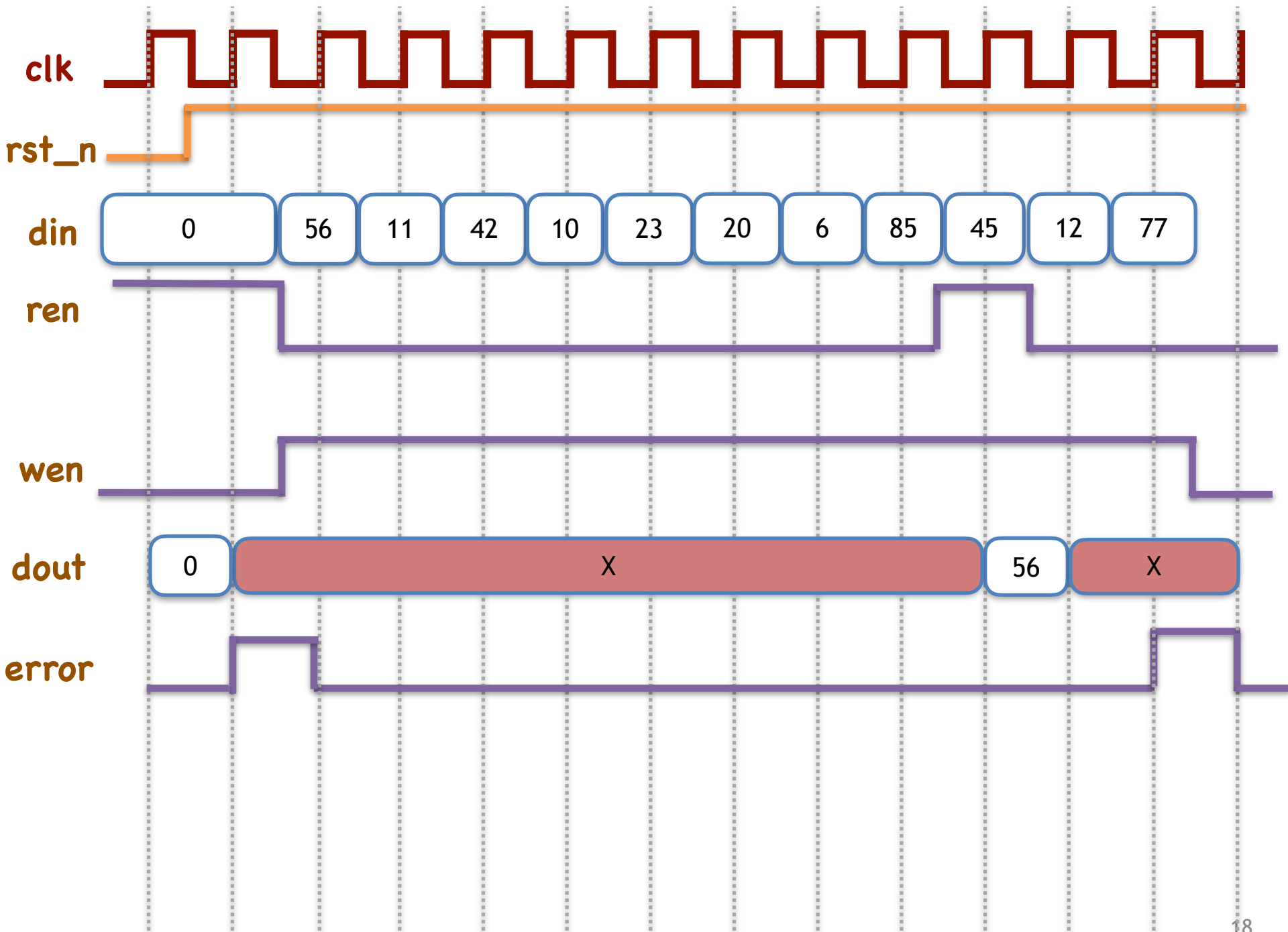
■ First-In First Out (FIFO) Queue



- Design a circular FIFO that stores eight entries of 8-bit data
- The order of the read should follow the FIFO pattern, in which the first data written would be read out first
- The behavior of the FIFO
 - By setting **ren=1'b1**, the FIFO should output the oldest data to **dout**. On the other hand, if **wen=1'b1**, the value of **din** signal is written into the FIFO. If both **ren** and **wen** are set to **1'b1**, **only the read operation is performed**
 - The FIFO should be able to be written when **ren=1'b0** **unless it is full**, and should be able to be read **unless it is empty**

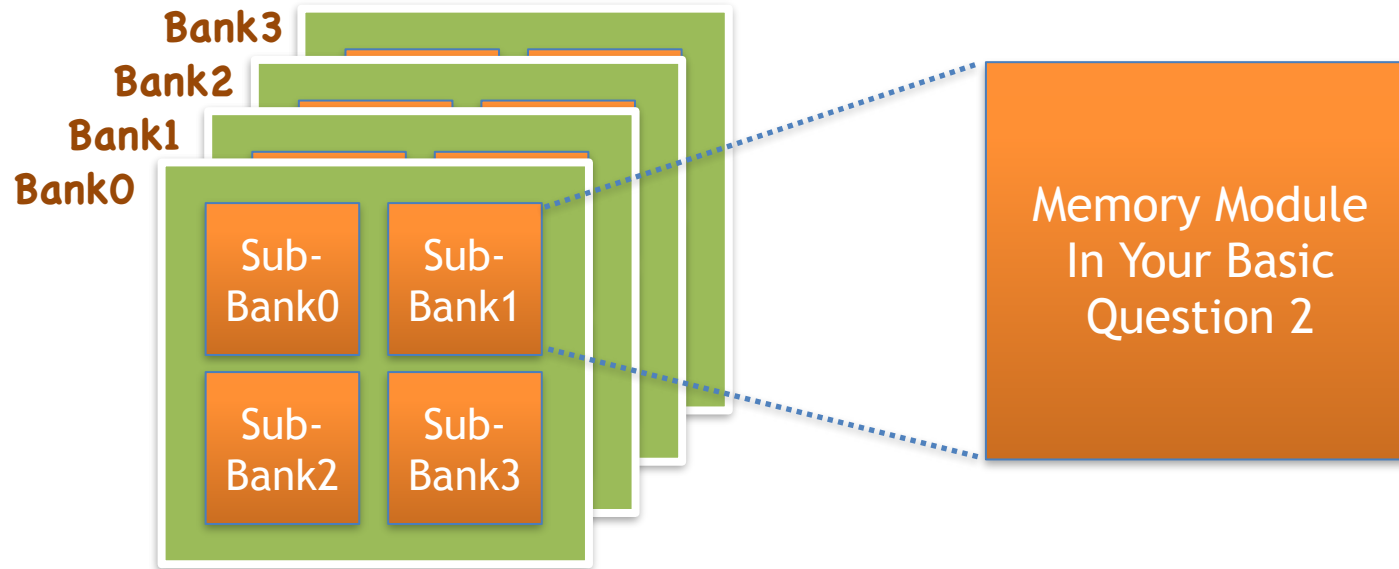
Verilog Advanced Question 2 (Con't)

- Error condition
 - If a **read** / **write** is issued to an **empty** / a **full** FIFO, the **error** bit should be set to **1'b1**. Otherwise, the **read** / **write** is valid and the **error** bit should be set to **1'b0**
- The values of **dout**
 - If there's an error, we do not care about the value of **dout**
 - If the FIFO is performing a write operation, we also do not care about the value of **dout**
 - If both **ren** and **wen** are zero, we also do not care about the value of **dout**
- If **rst_n** == **1'b0**, empty the FIFO, and set both **dout** and **error** to **zero**
- Please note that the values of **dout** and **error** should change synchronously, i.e., **their values should only change at the positive edges of clk**
- Please refer to the next slides for example waveform



Verilog Advanced Question 3

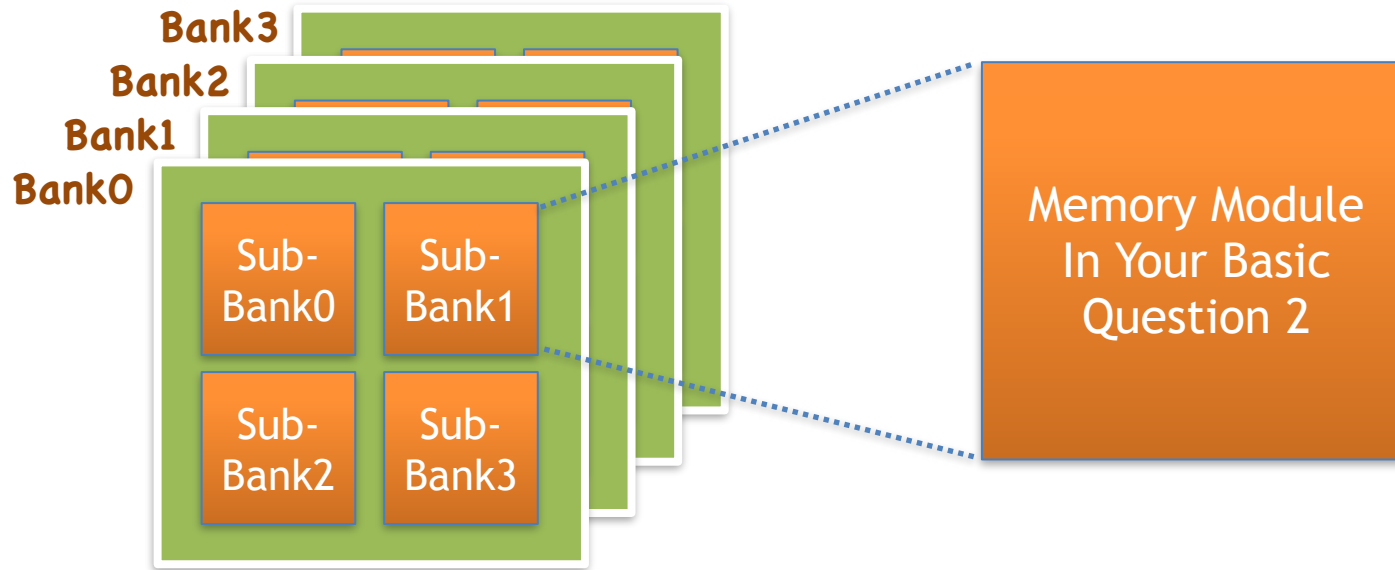
■ Multi-Bank Memory



- Design a memory hierarchy containing **4** banks of memory. Each bank consists of **4** sub-bank memory modules. (A total of **16** sub-banks)
- Points will be deducted if the specified hierarchy is not followed
- Please reuse the module from **Basic Question 2** for each sub-bank
- Input: **clk, ren, wen, raddr[10:0], waddr[10:0], din[7:0]**
- Output: **dout[7:0]**

Verilog Advanced Question 3 (Con't)

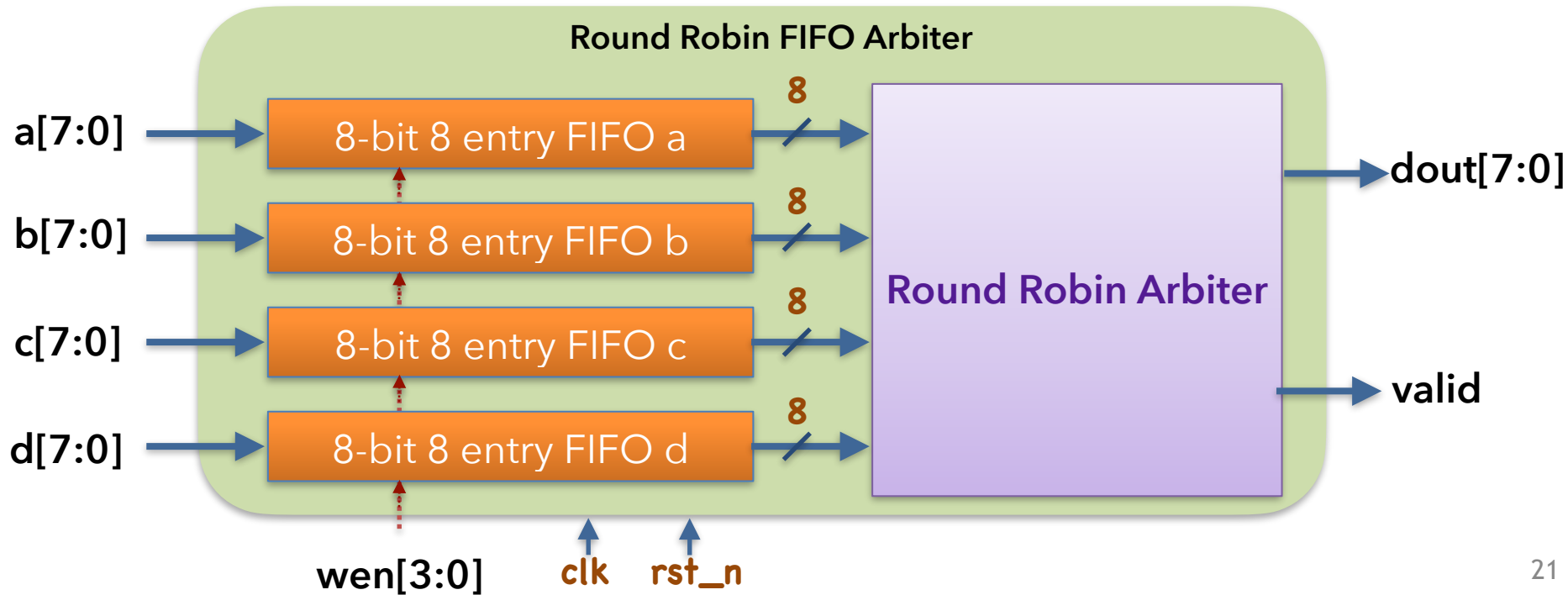
■ Multi-Bank Memory



- The most significant four bits of **raddr** (i.e. **raddr[10:7]**, read address) and **waddr** (i.e. **waddr[10:7]**, write address) address different sub-banks of different banks. For example, **waddr[10:7] == 4'b0110** addresses bank1's (01) sub-bank2 (10)
- When **wen == 1'b1**, write **din** to **Memory[addr]**
- When **ren == 1'b1**, output **Memory[addr]** to **dout**; otherwise **dout = 8'd0**
- When both **wen** and **ren** are **1'b1**, they can be serviced simultaneously if they are directed for **different sub-banks**. Otherwise, only read request is serviced

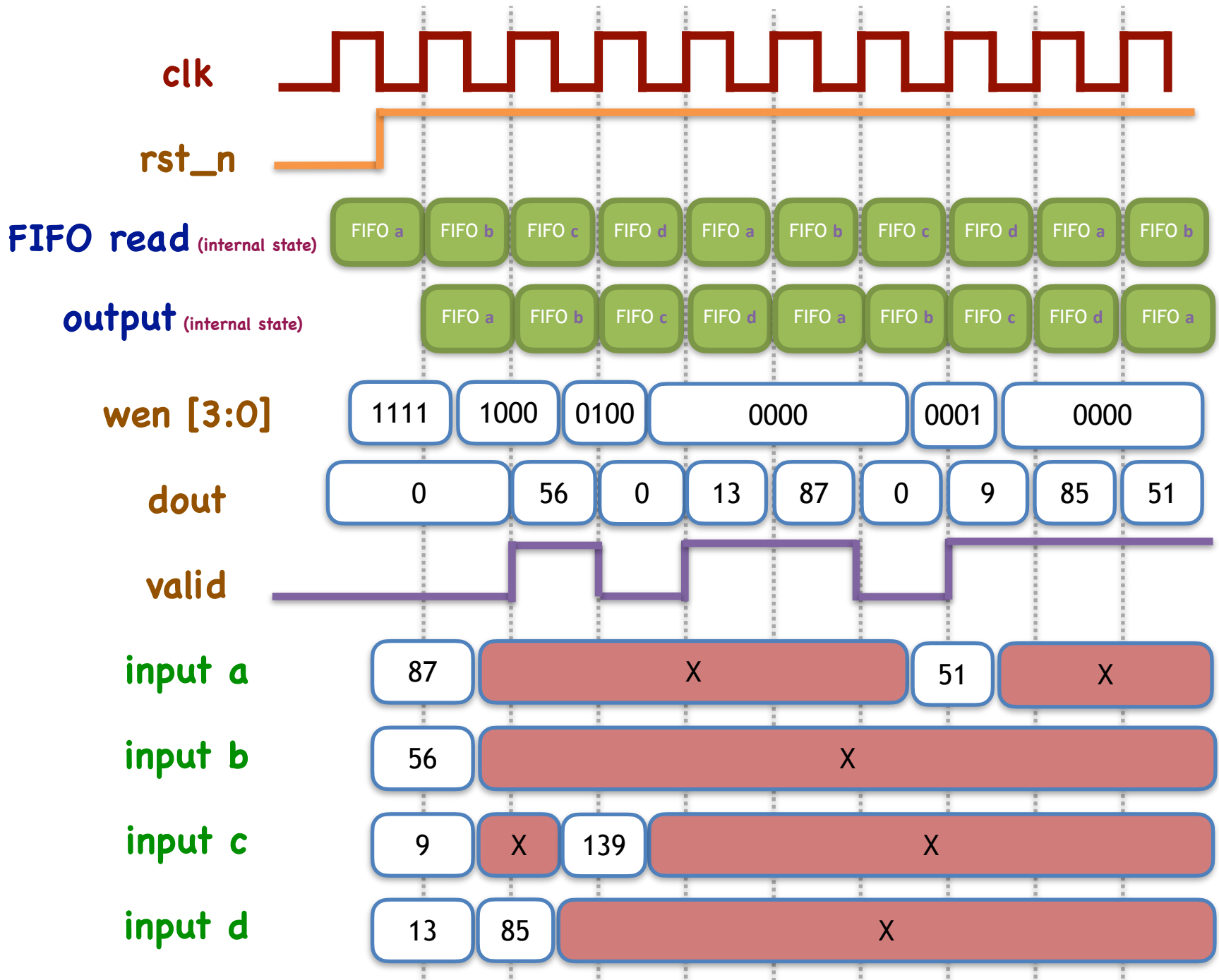
Verilog Advanced Question 4

- Design a **Round-Robin FIFO Arbiter** based on **Advanced Q2**
 - Input: `clk, rst_n, wen[3:0], a[7:0], b[7:0], c[7:0], d[7:0]`
 - output: `valid, dout[7:0]`
- Four FIFOs in advanced question Q2 are connected to a round robin arbiter, which controls their **ren** signals to make them output their contents via **dout** in a round robin fashion.



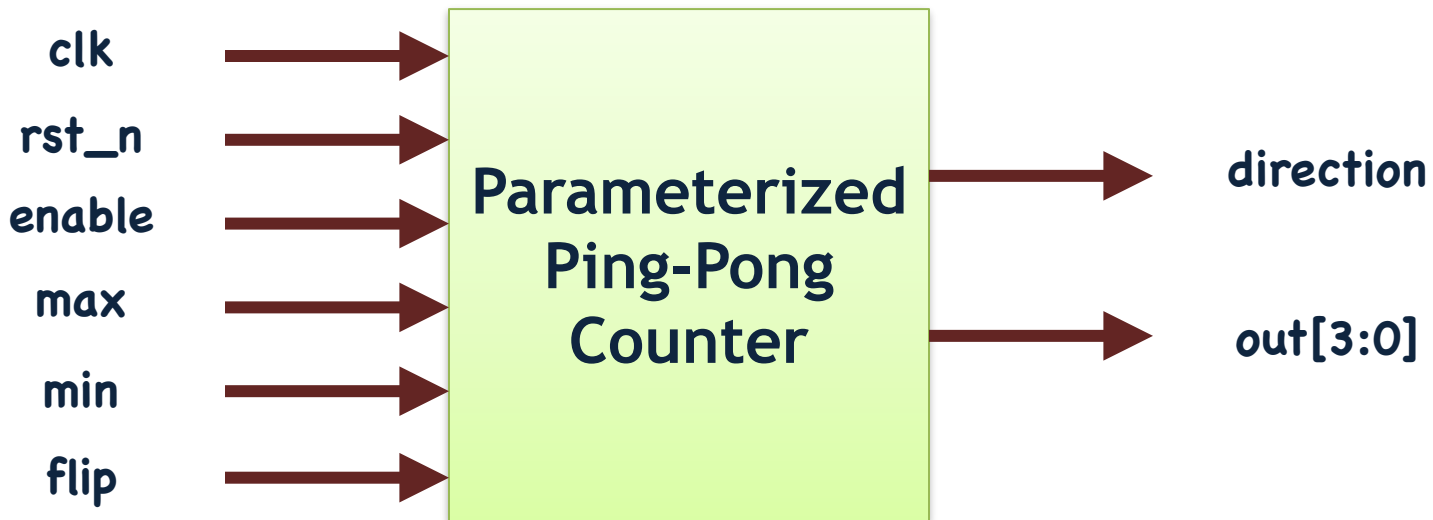
Verilog Advanced Question 4 (Con't)

- Each FIFO is written independently by setting the corresponding bit in **wen** to **1'b1**, e.g., setting **wen** to 4'b0001 will write **a** to FIFO a, 4'b1001 will write **d** to FIFO d and **a** to FIFO a
- The input data of FIFOs a, b, c, and d are supplied via input ports **a**, **b**, **c**, and **d**, respectively
- However, if the FIFO that is being accessed by the arbiter is also being written or its **error** signal is **1'b1**, the access is considered invalid. In such a situation, the **valid** and the **dout** signal should be set to **1'b0** and **no data is read out from the FIFO**. Otherwise, the read access is valid and **valid** should be set to **1'b1**.
- Please note that the values of **dout** and **valid** should change synchronously, i.e., **their values should only change at the positive edges of clk**.
- When **rst_n == 1'b0**, **dout** and **valid** should be set to **8'b0** and **1'b0**, respectively.
- Please refer to the next slide for a sample waveform.



Verilog Advanced Question 5

- Design a **4-bit Parameterized Ping-Pong Counter** with **max** and **min**
 - Input: `clk, rst_n, enable, flip, max[3:0], min[3:0]`
 - `out[3:0]`: `0,1,2,...,7,8,9,8,7,...,2,1,0,1,2,...`
 - `direction`: `1,1,1,...,1,1,1,0,0,...,0,0,0,1,1,...`
 - In the above example, **max** is 9 and **min** is 0



Verilog Advanced Question 5 (Con't)

■ **rst_n** and **enable**

- When **rst_n** == 1'b0, resets **out** to **min** and **direction** to 1'b1
- When **enable** == 1'b1, the counter begins its operation. Otherwise, the counter holds its current value

■ **max** and **min**

- **max** and **min** values are the maximum and minimum values for the counter
- **max** > **min**. Otherwise, the counter holds its current value
- When counter > **max** or counter < **min**, counter holds its current value

■ **flip**

- When **flip** == 1'b1, counter flips its direction
- Flip can occur anytime, but it only takes effects when counter < **MAX** and counter > **MIN**

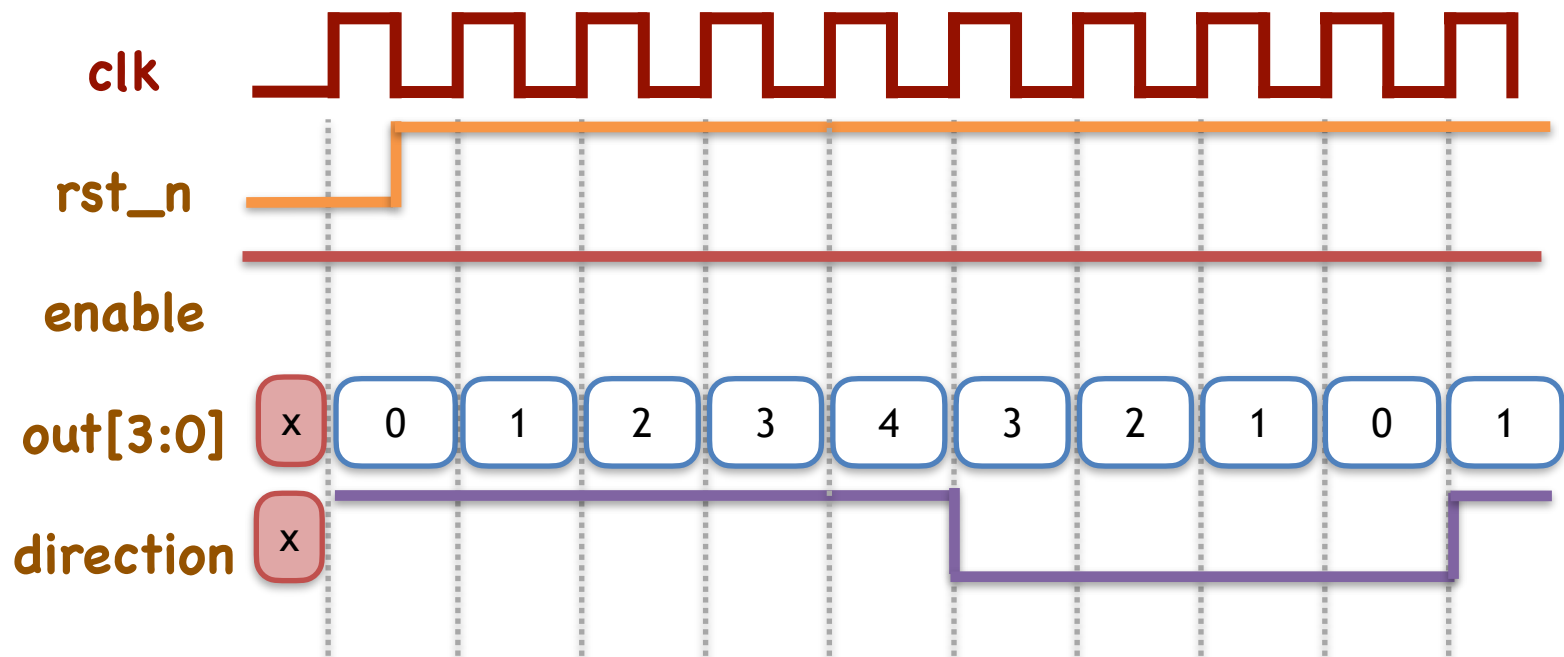
Verilog Advanced Question 5 (Con't)

■ Notes

- Be careful that **max** and **min** will change during counting
- Once the value of the counter is out of range, hold the value and direction
- If **max == min == output**, please hold the **output** and **direction**
- The following slides provide some example waveforms

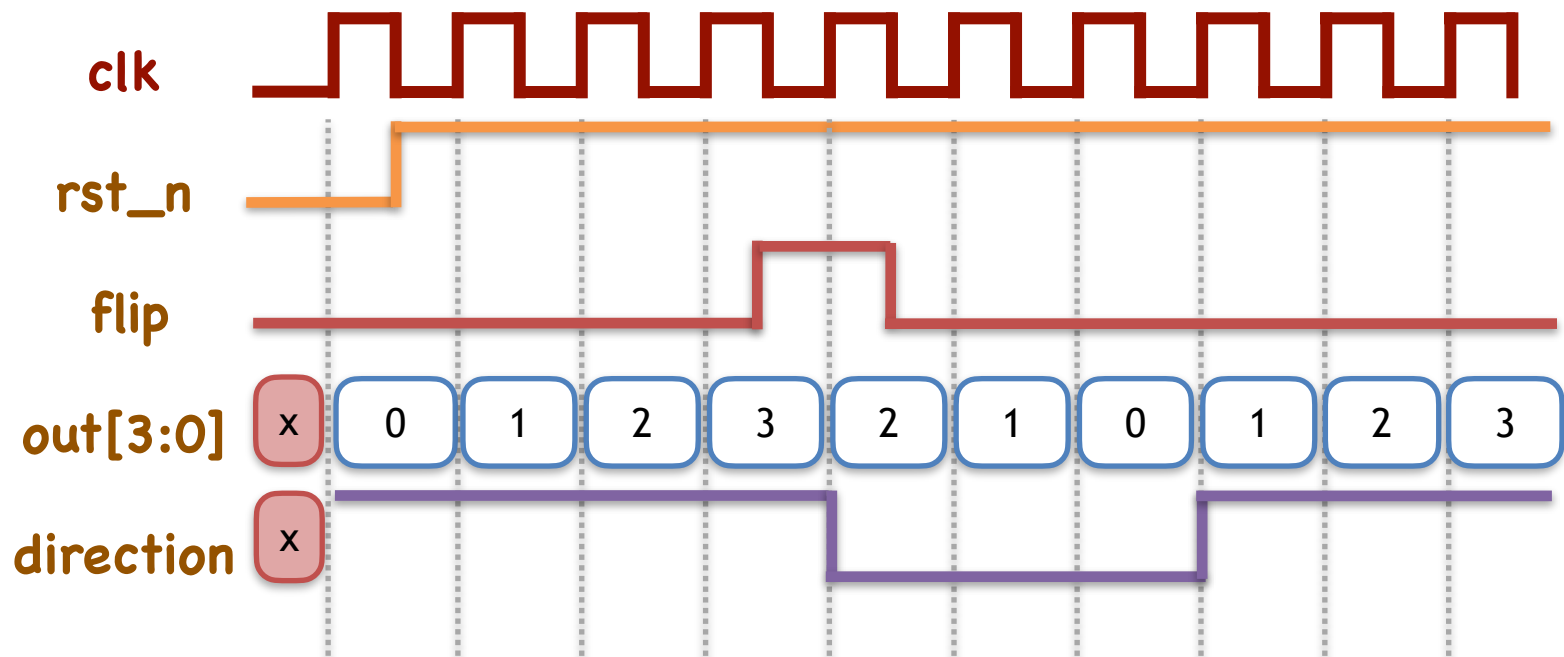
Verilog Advanced Question 5 (Con't)

- An example waveform where **flip** is set to 1'b0 and **enable** is set to 1'b1
- In this example **min** = 4'd0 and **max** = 4'd4



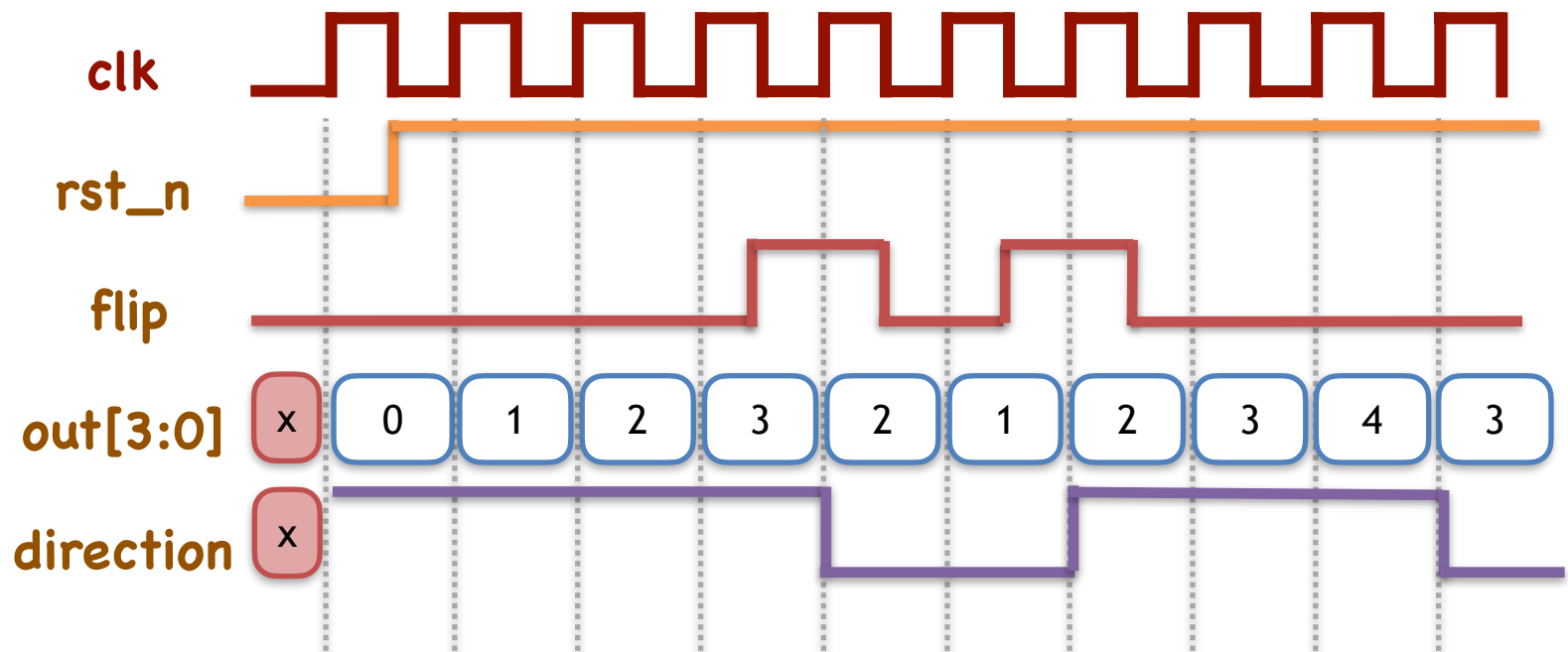
Verilog Advanced Question 5 (Con't)

- An example waveform where there is one **flip** and **enable** is set to 1'b1
- In this example **min** = 4'd0 and **max** = 4'd4



Verilog Advanced Question 5 (Con't)

- An example waveform where there are two **flips** and **enable** is set to 1'b1
- In this example **min** = 4'd0 and **max** = 4'd4



Advanced Questions

- Group assignment
- Verilog questions (due on 10/10/2024. 23:59:59.)
 - **Necessary:** 4-bit Ping-Pong Counter
 - **Necessary:** First-In First Out (FIFO) Queue
 - **Necessary:** Multi-Bank Memory
 - **Necessary:** Round-Robin FIFO Arbiter
 - **Necessary:** 4-bit Paramterized Ping-Pong Counter
- **FPGA demonstration** (due on 10/24/2024. In class.)
 - **Necessary:** 4-bit Paramterized Ping-Pong Counter on FPGA

FPGA Demonstration 1

- 4-bit Paramterized Ping-Pong Counter on FPGA
- **Behavior specification**
 - In the beginning, the digits showing on the 7-segment display should be the value of **min**
 - Once **enable** is on, the Ping-Pong Counter starts counting
 - When **enable** is off, the Ping-Pong Counter holds its value
 - The Ping-Pong Counter only counts when **max** > **min**
- **Switches**
 - **SW[15]** stands for **enable**
 - **SW[14:11]** stand for **max**
 - **SW[10:7]** stand for **min**

FPGA Demonstration 1

■ Buttons

■ "DOWN" button stands for **flip**

- Once flip occurs, you should change your direction

- Flip only occurs when **min** \leq output \leq **max**

■ "UP" button stands for **rst_n**

- Once the button is pushed, the output is set to the value of **min**, which is determined by **SW[10:7]**

- The direction is set to "**counting up**"

■ Please present your output signal on the two leftmost 7-segment displays

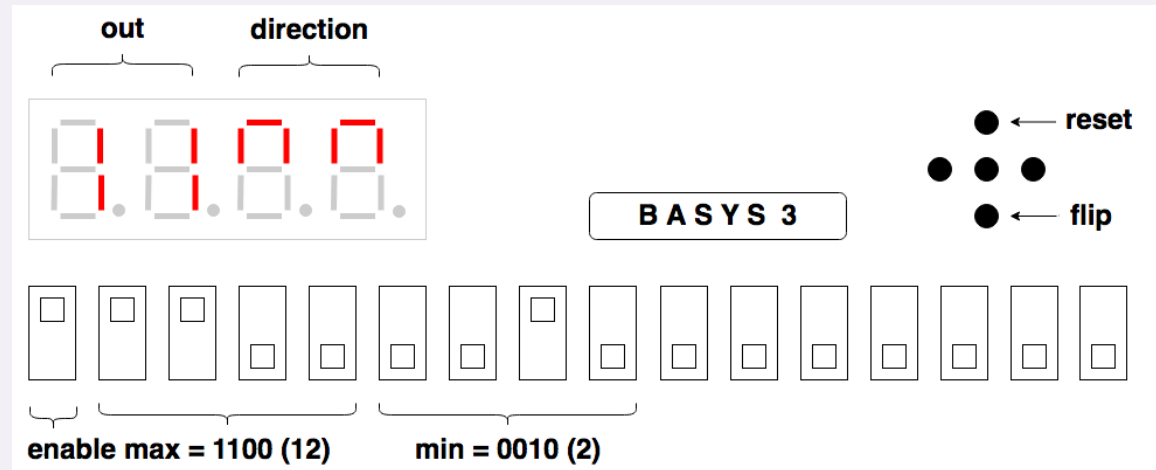
FPGA Demonstration 1

- 7-segment display

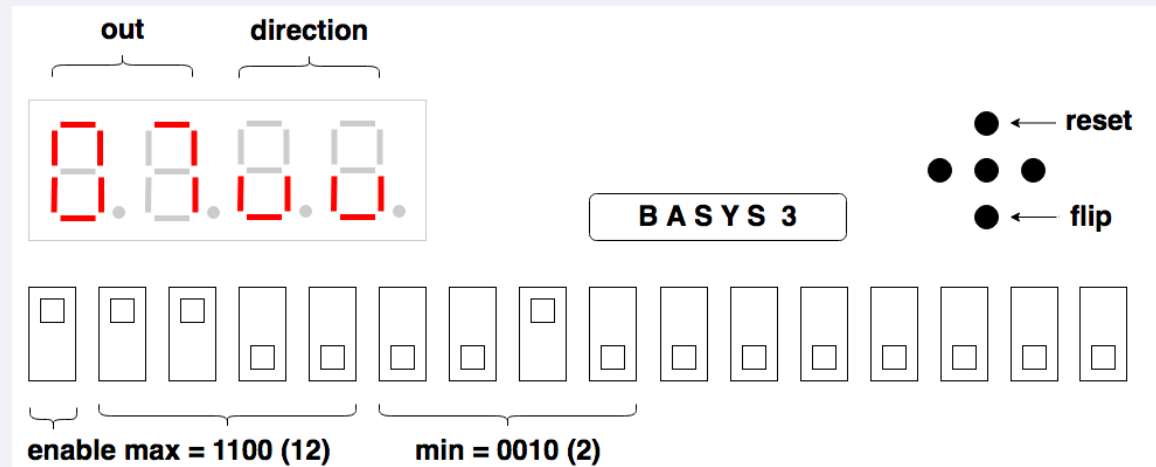
- The rightmost two digits of the 7-segment displays stand for **direction**
- Please illuminates the upper three segments when counting up, and illuminates the lower three segments otherwise
- Please see the figure on the next page for more details

FPGA Demonstration 1

Counting Up



Counting Down



FPGA Demonstration 1

■ Notes

- Be careful that **max** and **min** will change during counting
- Once the value of the counter is out of range, hold the value and direction
- If **max == min == output**, please hold the **output** and **direction**
- You **MUST** add debounce and one-pulse circuits for your buttons
- **Remember to add debounce and one-pulse circuits to your design**
- We use the **100MHz** clock which is provided by the FPGA board. Please set **clk** as input and connect it with the **W5** port on the FPGA board.
- Your counter should count in an observable frequency so that TAs can tell whether your design is correct or not



Thank you for your attention!

*Balloon Festival at Reno, Nevada, USA
This picture is taken by Chün-Yi Lee himself, who is also a fan of photography