Lab 3: Sequential Circuits

Group 3 : 109060013 張芯瑜　109062328 吳邦寧

Design Explanation

1. Ping Pong Counter

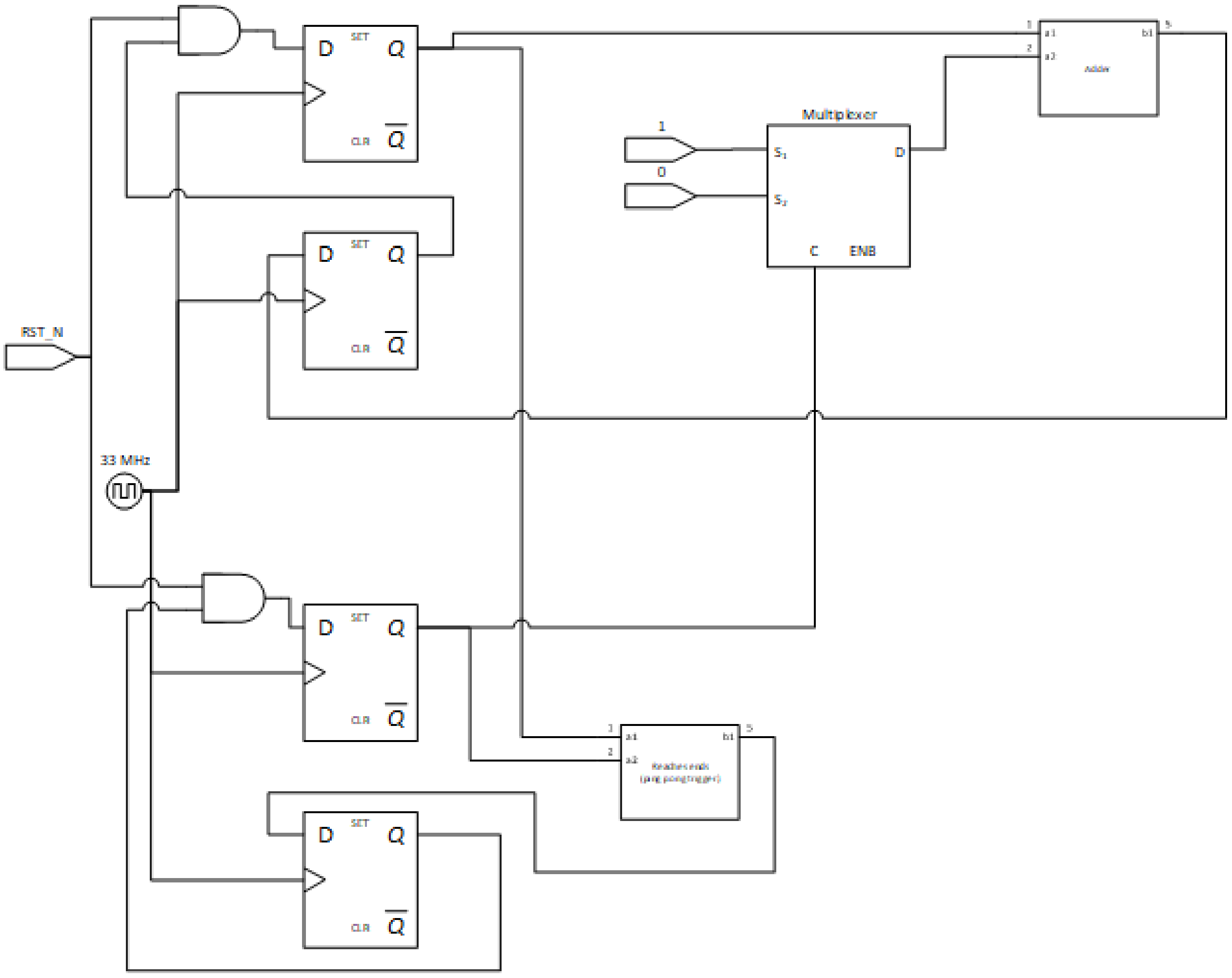
(1) Design Concepts

Reverse the direction when it meets end (at bottom & decreasing or at top & increasing). Reversal can be applied by XOR.

Increment when direction is 1 and decrement when direction is 0.

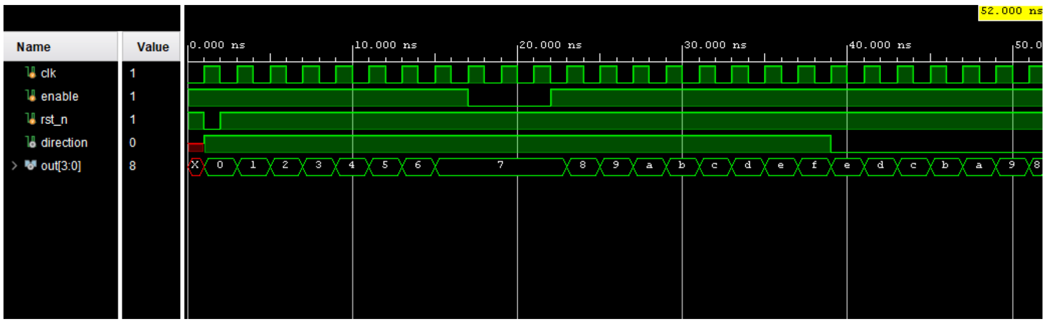
Apply the previous procedures when clock is triggered.

(2) Schematic Graph



(3) Validation

Reset and let it ping pong.



2. First-In First Out (FIFO) Queue

(1) Design Concepts

For the FIFO, we create a register for it to keep the signals. The whole FIFO has 8 place to keep the signals, and each of them can take 8 bits.

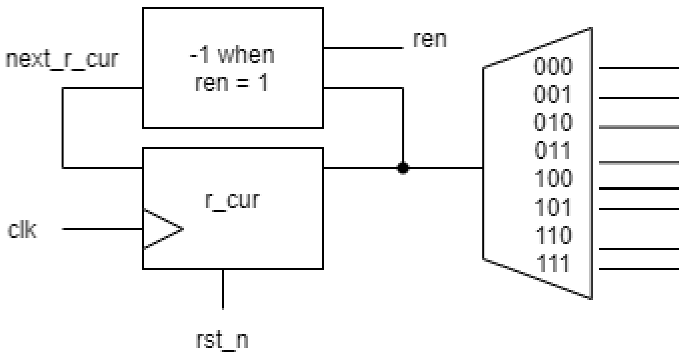
In order to read or write the correct signal we need, we use r_cur for knowing which one we are goning to read out and w_cur for knowing which one we are goning to write in. Both of them move to the next place after doing their job. Also, they can move in the cycle of the FIFO.

Next, the FIFO might be empty or full and the work of reading out and wrting in may go wrong, so we use another register “full” to keep counting how many signals are in iur FIFO now. When we write in a signal, “full” will add one to itself. When we read out a signal, “full” will minus one to itself. By counting “full”, we may check are we reading from an empty FIFO or writing into a full FIFO. Then we can get are error signals.

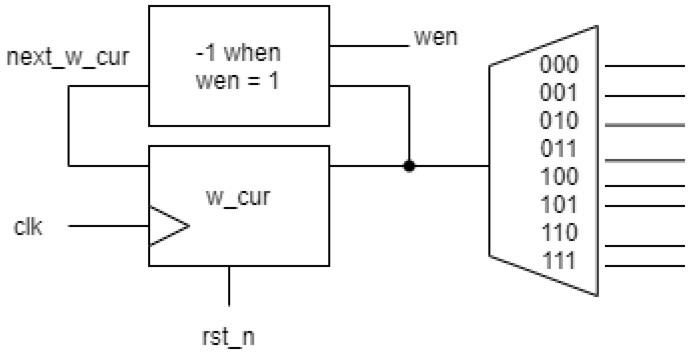
Last, the reset part is easy. While rst_n becomes 0 in this clock, we reset are r_cur and w_cur to the head of our FIFO. And also, we reset “full” to 0 because the FIFO had been reset to an empty FIFO.

(2) Schematic Graph

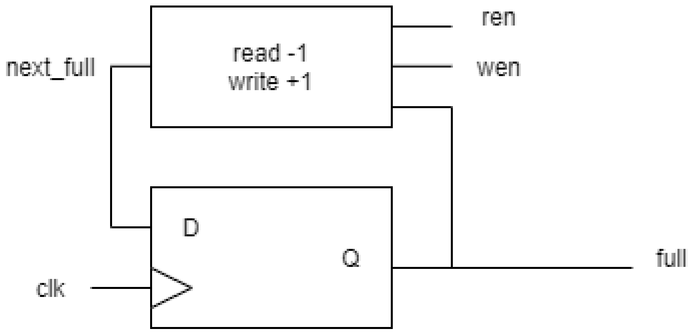
- r_cur



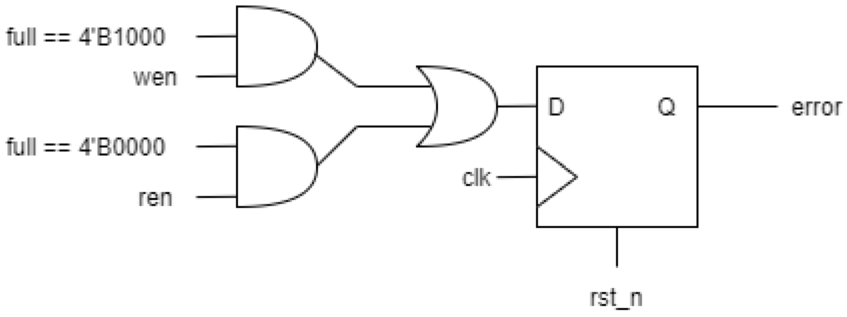
- w_cur



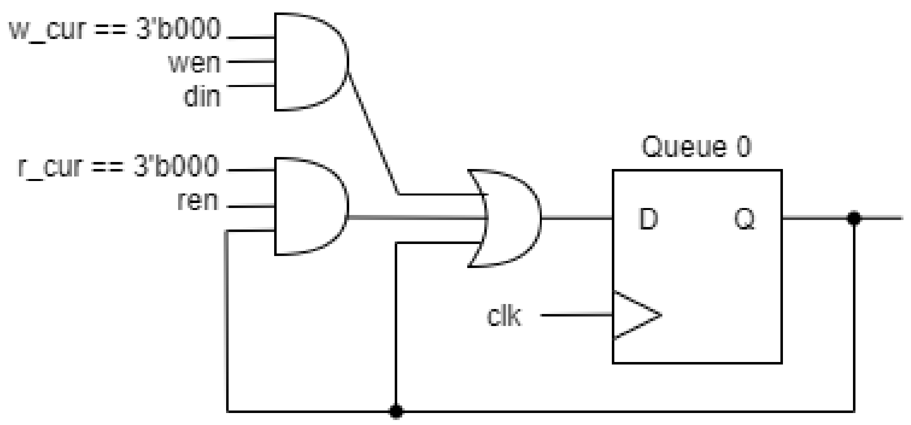
- full



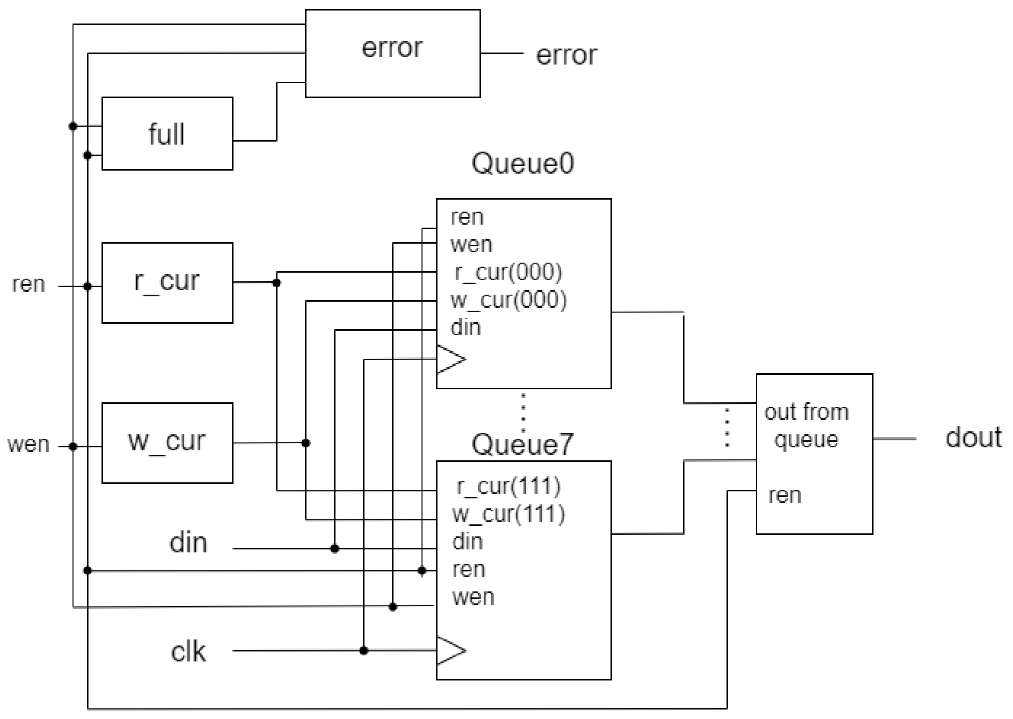
- error



- each of Queue(ex:Queue0)



- whole

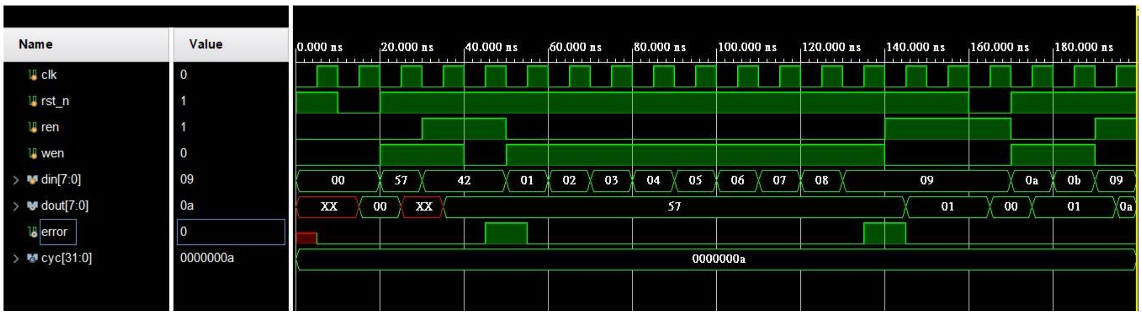


(3) Validation

In the testbench, we check a couple of things below:

i. ren and wen are both 1, then we only read
ii. reading an empty FIFO will get an error signal
iii. writing into a full FIFO will get an error signal
iv. reset can go well
v. reading and writing correctly

Then set the inputs to satisfy the condition above.



3. Memory Bank

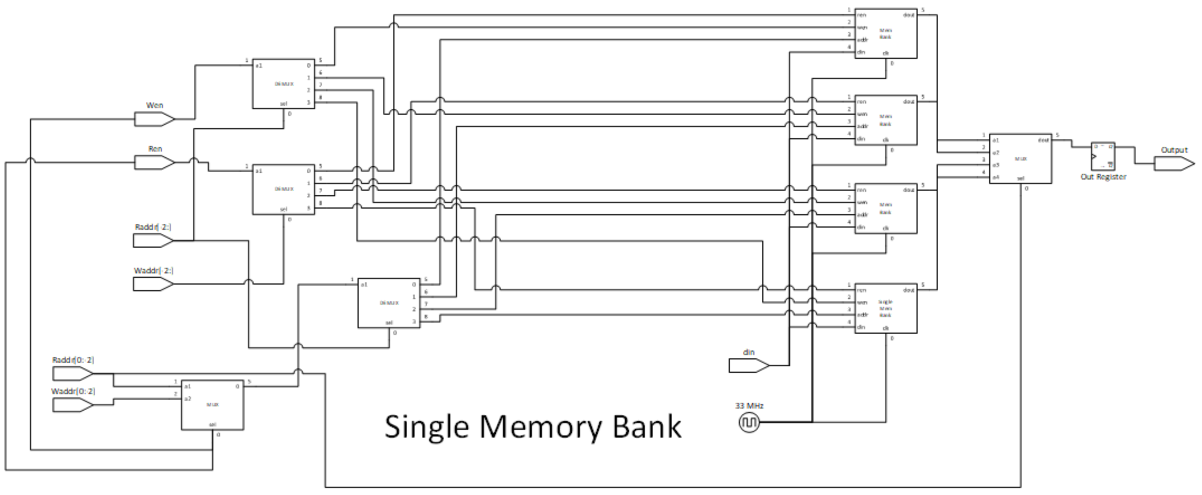
(1) Design Concepts

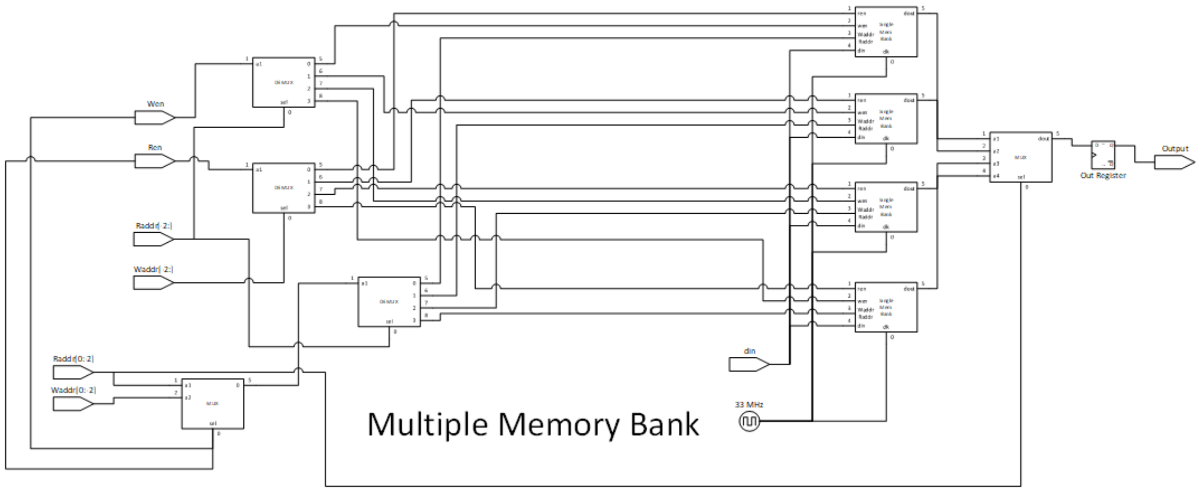
We design 2 modules, each has 4 children.

In the first module, 4 naïve memory bank is connected. Corresponding signals are attached by mux and demux.

In the second module, 4 single memory bank is connected. Correspond signals are attached by demux. The second module is nearly identical to the first module.

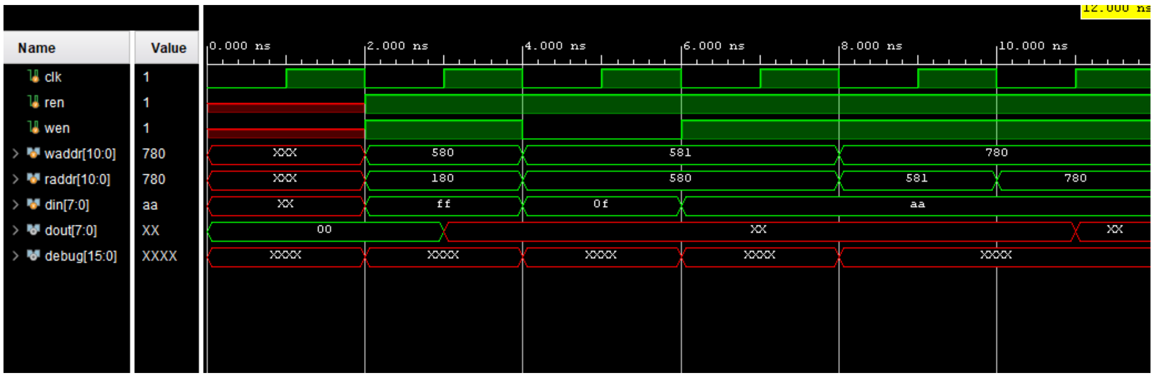
(2) Schematic Graph





(3) Validation

Randomly write some info in and write some info out.



4. Round-Robin FIFO Arbiter

(1) Design Concepts

We reuse the FIFO from question2, and create 4 FIFOs for this module.

For reading, we want to read one signal at each clock. Because the order need to be from a to d, so we count the next_ren at every clock, and set ren to next_ren after using ren to decide which one to be read.

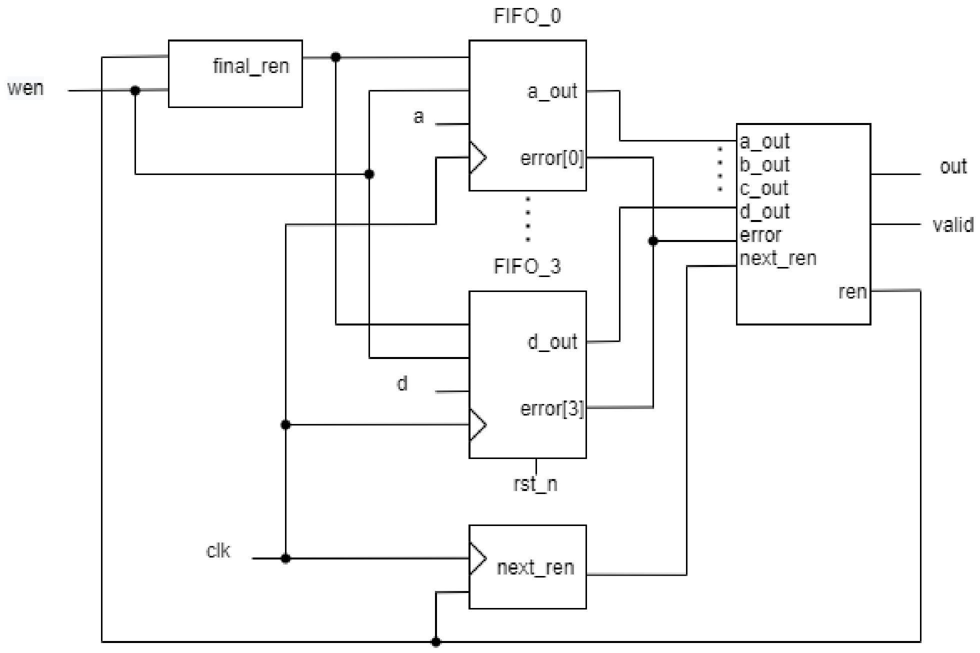
In order to check if we are writing or reading at this clock, we use “final_ren” to check. When the ren or wen signals change, we get a new “final_ren” to know are we going to read at this clock.

ren	0	1	0	1
wen	0	0	1	1
final_ren	0	1	0	0

At every clock, we write, read or do nothing to the FIFOs. It depends on the signal of final_ren and wen. After doing their job, we choose the output that we want by ren signal. If we get an error signal from the FIFO that we are reading or the final_ren is 0, then we output an invalid signal and 0 for dout. Otherwise, we output the one we want and valid signal.

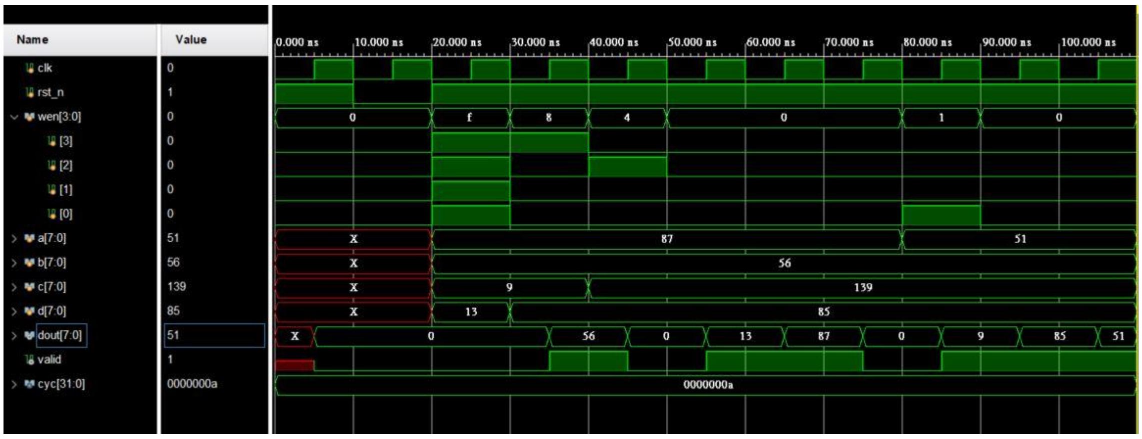
Last, the reset part is easy. When we get a negedge rst_n signal, we reset ren to the start(4’b0001) ,set dout to 0, and output invalid.

(2) Schematic Graph



(3) Validation

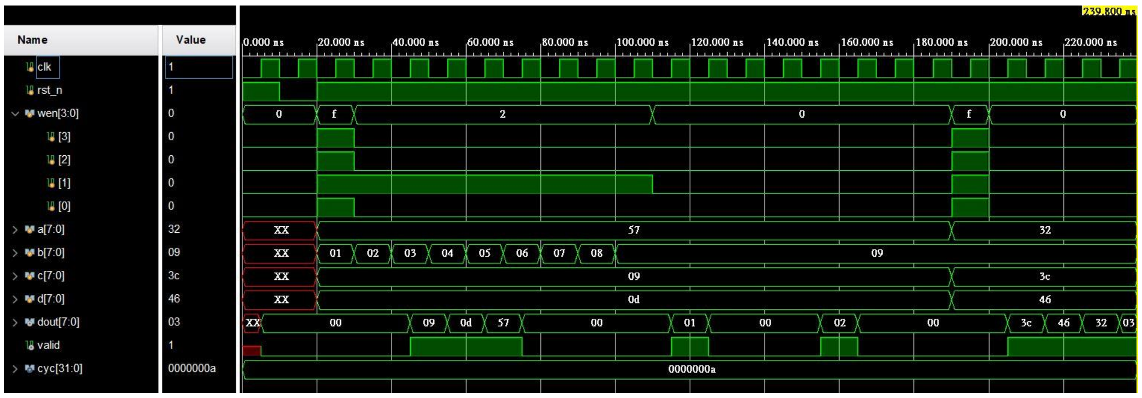
First, we tried to get the same wave as the ppt wrote. We success!!!



Second, we try to check a couple of things below:

i. ren and wen are both 1, then we only write and get invalid signal
ii. reading an empty FIFO will get an error signal and get invalid
iii. reading and writing are both correctly and get valid signal

Then set the inputs to satisfy the condition above.



Name	Value
clk	1
rst_n	1
wen[3:0]	0
[3]	0
[2]	0
[1]	0
[0]	0
a[7:0]	32
b[7:0]	09
c[7:0]	3c
d[7:0]	46
dout[7:0]	03
valid	1
cyc[31:0]	0000000a

5. Parameterized Ping Pong Counter

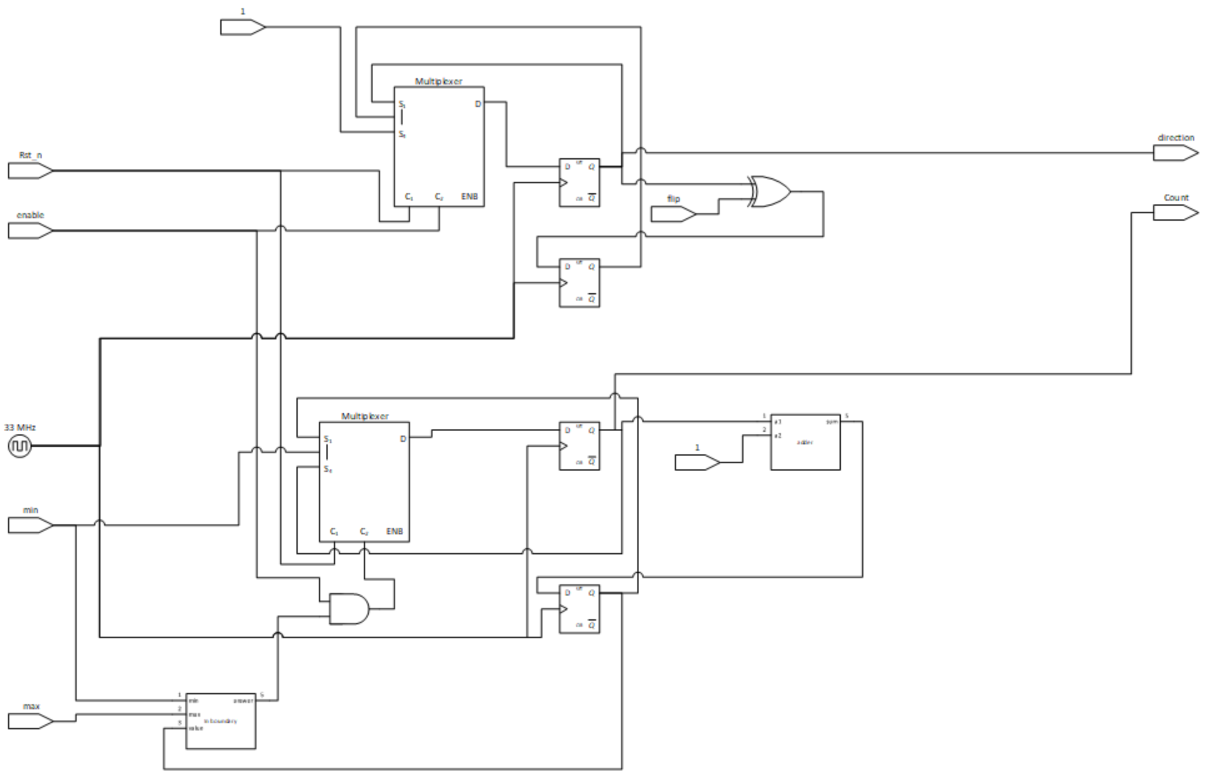
(1) Design Concepts

Reverse the direction when it meets end (at bottom & decreasing or at top & increasing). Reversal can be applied by XOR.

Increment when direction is 1 and decrement when direction is 0.

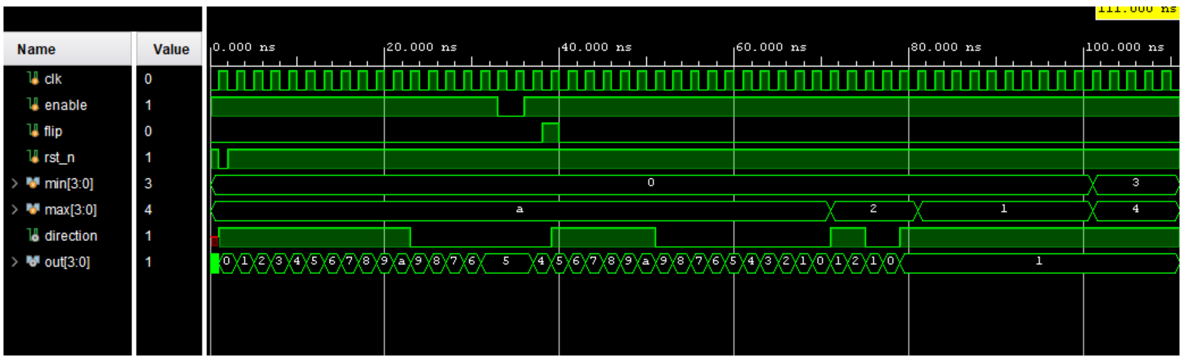
Apply the previous procedures when clock is triggered.

(2) Schematic Graph



(3) Validation

First, we test the reset signals. Second, we test the enable signal. Third, we test the flip signal. Last, we test the min/max signals and the boundary conditions.



6. FPGA Design

(1) Design struggle:

As we all know, messing around with clock signals is never a recommended development approach. But the soul of science and the ambition of exploration drives me to alter the clock signal.

To slow down the process for demonstration in naked eyes, I decided to slow down the clock for Parameterized Ping Pong Counter to 1Hz and the clock for 7 segment display to 100Hz.

Clock divider can be easily achieved by registers and equality. Apply clock divider, we shall gain the desired frequency.

But the output in simulator was different to the output in FPGA. After trials and error, experiments and inductions, I found glitches in the circuit.

Extra current is specially provided for clock so parallel energy consumption would not lead to unstable outputs. After messing around with clock signals, the power output on a XOR gate was insufficient to drive all the circuit, therefore the glitches were presented.

Remove the XOR gate and replace it with a register, the circuit works exactly as expected!

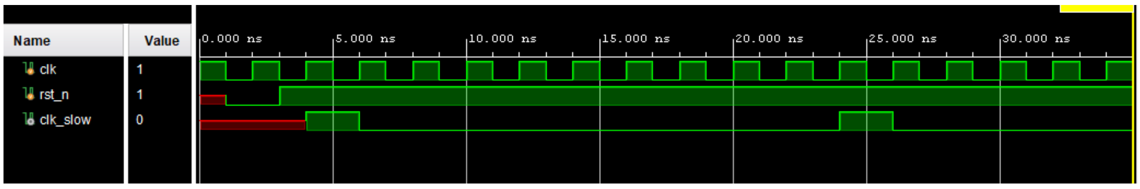
(2) Design concepts:

There is 3 modules for FPGA Demo. The first one is clock divider, which handles the clock signals and to slow down the circuit for naked eyes demonstration. The second one is main chip, which is parameterized ping pong counter. The third one is the 7 segment display module, which handles the transcription between main chip and display.

(3) Validation

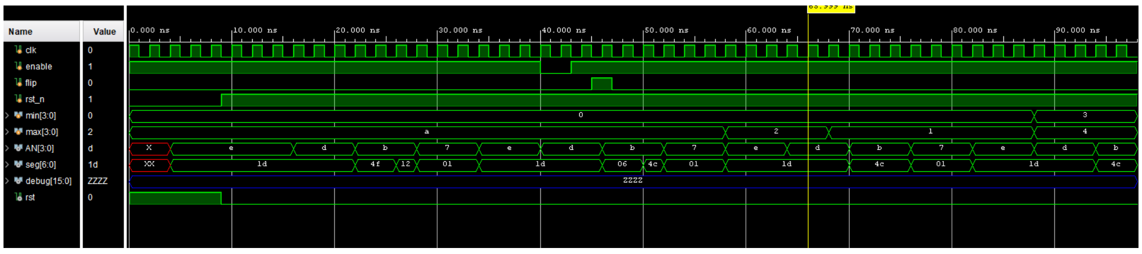
a. Clock Divider testbench

Divide the clock cycle into 1/10. Manual validation is needed.



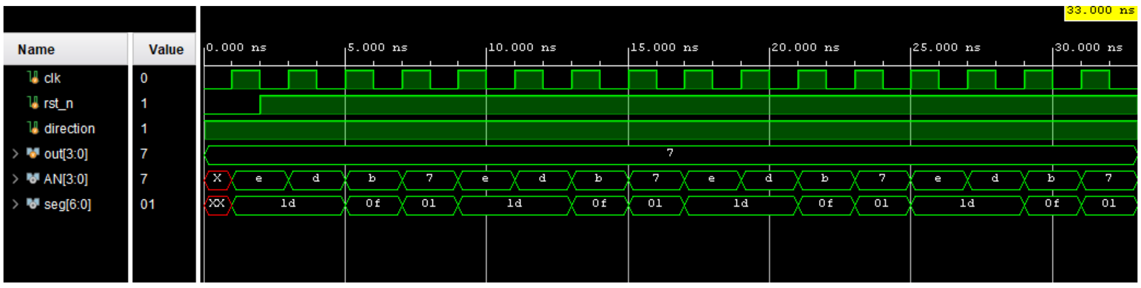
b. FPGA testbench

Identical to parameterized ping pong counter.



c. 7 segment display testbench

Randomly give some circuit output and check the transcription manually.



Contribution

1. Lawrence Wu

Problem 1, Problem 3, Problem 5 and FPGA demonstration.

2. Ariel Chang

Problem 2, Problem 4, integrating report.

What have we learned?

1. Never ever mess around with clock signals
2. Always respect those hardware engineers
3. Splitting modules is important
4. Don’t be rude to TA

5. Behavior on FPGA and Behavior on Simulator might be different
6. Follow the KISS (Keep It Simple and Stupid) rule and it will save your weekend
7. Generating bitstream takes a long time, so always debug on simulator when possible
8. Always fill the default case, otherwise synthesizer would do stupid stuff
9. Insufficient sleep + bug = more bugs