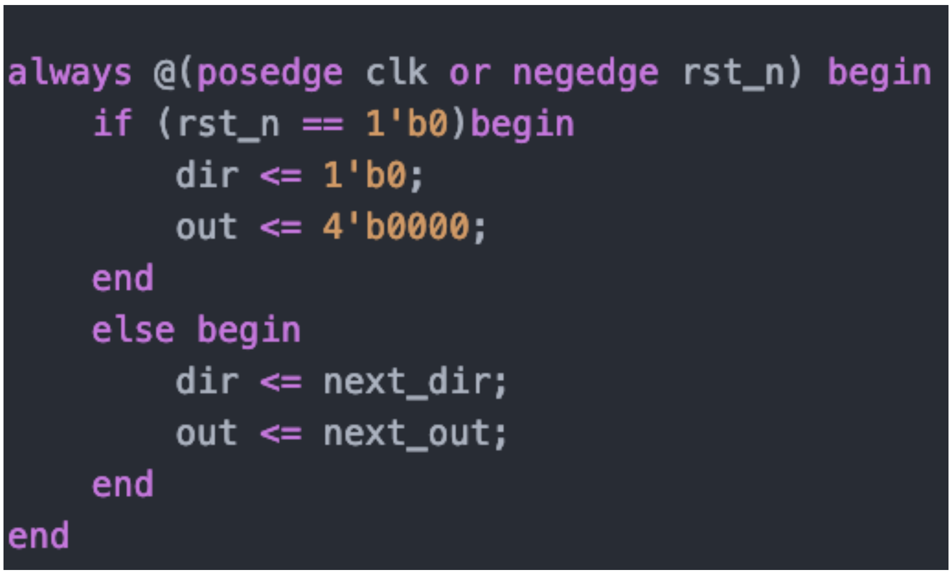
EECS4010 Homework Assignment 01

Student ID: 106062113 Name: 鄭柏文

1. Design Concept

I will split code into two parts and explain them separately.

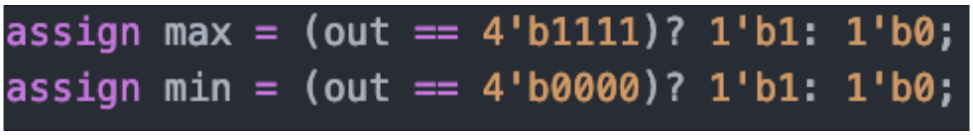
The first part is the sequential part of the circuit. Here, we will implement our DFFs and their reset function. As shown in the picture below. In this assignment we implemented three DFFs: dir and out. They are both positive edge triggered and asynchronous reset registers. When rst_n becomes zero, it both registers will be triggered and reset. The dir register will be reset to 0 which indicates the counter is going to count up. Out register will be reset back to zero.



```verilog
always @(posedge clk or negedge rst_n) begin
    if (rst_n == 1'b0)begin
        dir <= 1'b0;
        out <= 4'b0000;
    end
    else begin
        dir <= next_dir;
        out <= next_out;
    end
end
```

Picture 1

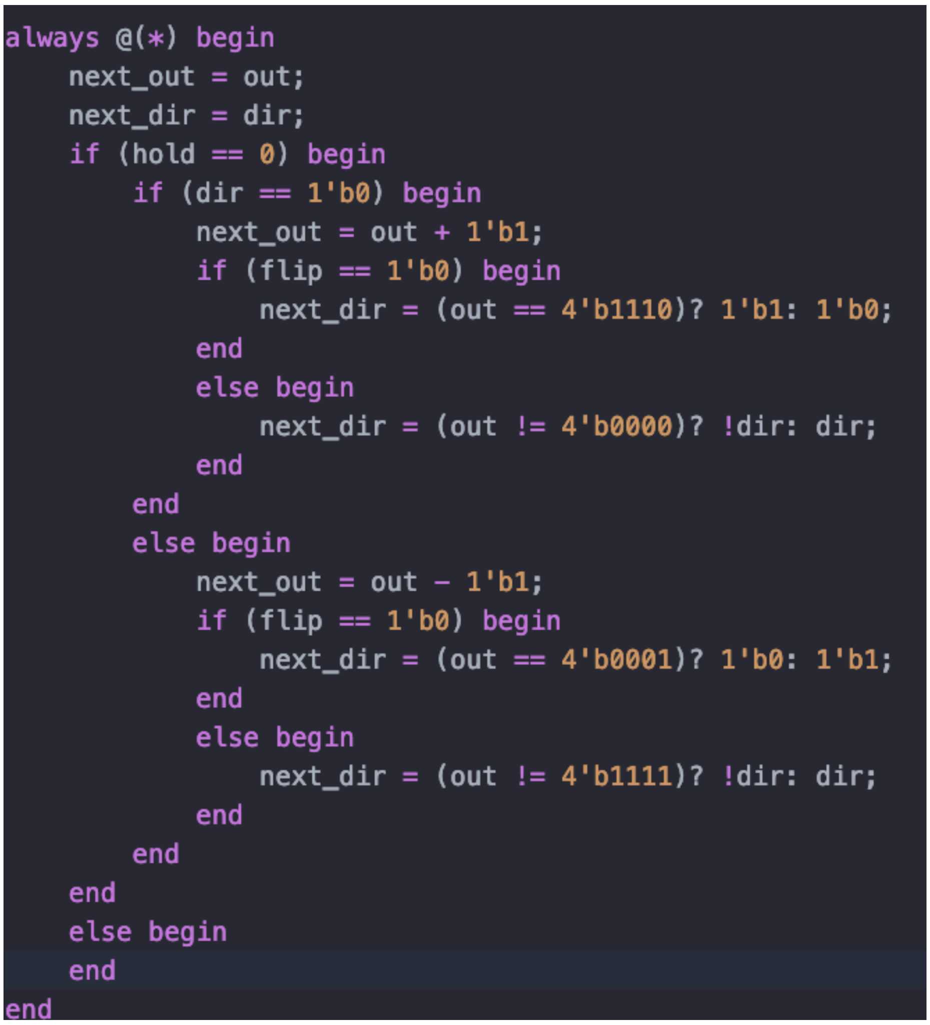
Then theirs is the combinational part of the design. I first tackled min and max signals, since there are pretty straight forward ans easy to implement. The following picture shows the implementation of these two signal in RTL Verilog. When out equals 0, which is our minimum output, the min signal will be asserted. When out equals 15, which is out maximum output value, the max signal will be asserted.



```verilog
assign max = (out == 4'b1111)? 1'b1: 1'b0;
assign min = (out == 4'b0000)? 1'b1: 1'b0;
```

Picture 2

Then we deal with the rest of the functionality of the design. We first consider the value of hold, if the hold signal is asserted, the counter does nothing. Then we check the value of dir since the direction we are counting towards will greatly affect the behavior of our design. If dir equals zero, that means we are counting up, so we set next_out to be out + 1 regardless of the flip signal being one or zero. Direction, however, is affected by the flip signal. If flip is not asserted, dir continue to be zero until we count to 15. When out reaches 15, next cycle it will counts to 16 and start counting down, so we set the value of next_dir to be one. If the flip signal is asserted, we change flips the direction by set next_dir to !dir. We are asked not to consider extreme cases like flipping at 0 or 15, so if flip is asserted when the counter is at 0 or 15, it will be ignored. The counting down part(dir = 1) is pretty similar to the up-counting part with only some minor modification to accommodate the different direction. The picture below shows the implementation of the function I just described.

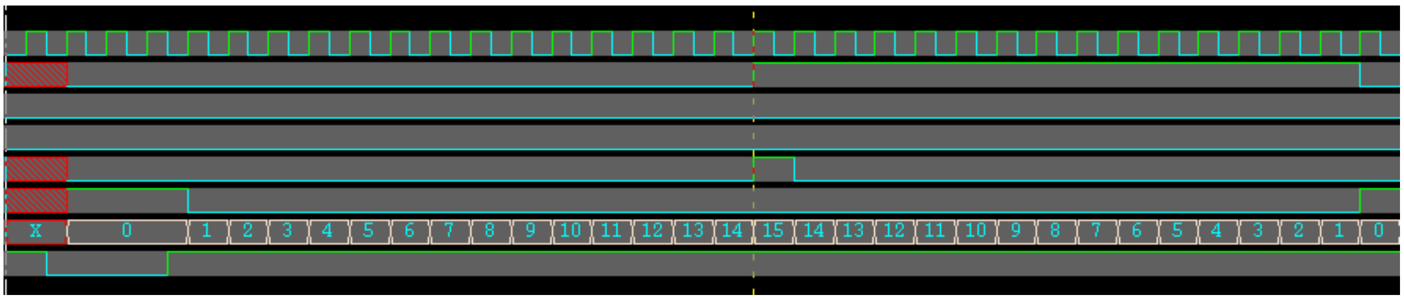


```
always @(*) begin
    next_out = out;
    next_dir = dir;
    if (hold == 0) begin
        if (dir == 1'b0) begin
            next_out = out + 1'b1;
            if (flip == 1'b0) begin
                next_dir = (out == 4'b1110)? 1'b1: 1'b0;
            end
            else begin
                next_dir = (out != 4'b0000)? !dir: dir;
            end
        end
        else begin
            next_out = out - 1'b1;
            if (flip == 1'b0) begin
                next_dir = (out == 4'b0001)? 1'b0: 1'b1;
            end
            else begin
                next_dir = (out != 4'b1111)? !dir: dir;
            end
        end
    end
    else begin
    end
end
```

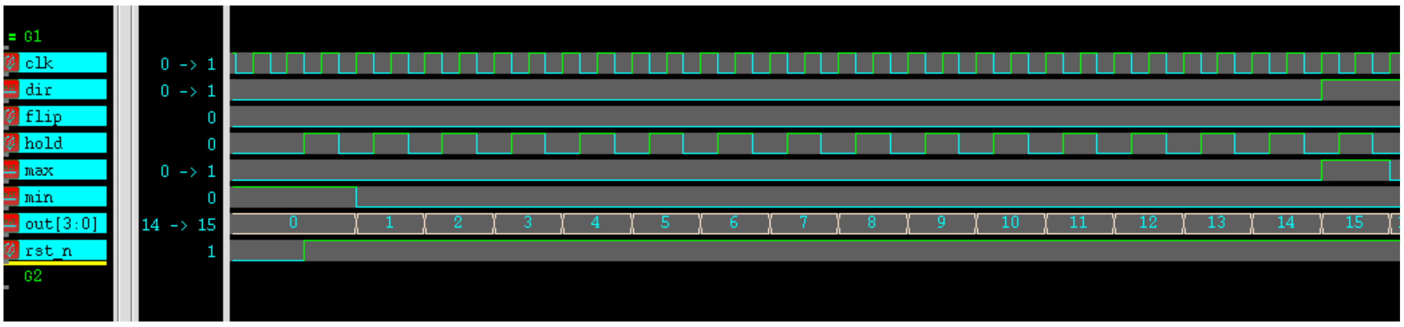
Picture 3

2. Simulation and Discussion

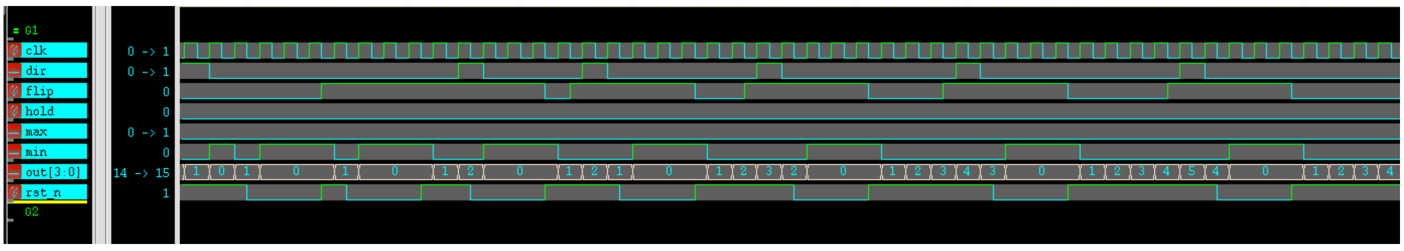
For simulation, I split the test pattern into 3 parts. First is the part that no hold or flip is asserted, the counter counts all the way op and all the way down to see whether we implement the counting function right or not.



Then in the second part, we add an extra cycle to each output by asserting hold, which is picked up by the design at the second positive edge of clock. This is to test whether we implemented the hold function correct or not.



In the final part of the test. We asserted flip for all the output value to check if we implemented the flip function correctly. I also tested for the special case where out equals zero or 15 and whether the output counts back and forth if we asserted flip for multiple cycle.



While Doing the assignment, I didn't run into much problem, the only problem I ran into is how I can make my code more readable and more compact. As shown in the picture above, there's a lot of nested if else statement in my design, which is not a good practice. I tried creating one more register called state, and define the circuit’s behavior based on the state the circuit’s currently in. However, by doing so, I had to add a lot more code and signals into my design and end up making my design less readable and messy. In the end, I just went back to my original code.

3. Summary

Assignment 1 was not hard for me since I have done similar problem in Hardware Design lab before.but this time we are using workstation instead of vivado as our working environment. This brings back good old memory from my logic design class, in which I had a ton of fun. Different from last time, however, is that I step up the testbench design a lot. I used to only tests for a couple of values for my design and that’s it, but this time I came up with my own design pattern and tried to test my design as thoroughly as possible. I hope I can come up with better design and test pattern in the future.