Lab 3

學號 ：109000168 姓名 ： 許媄香

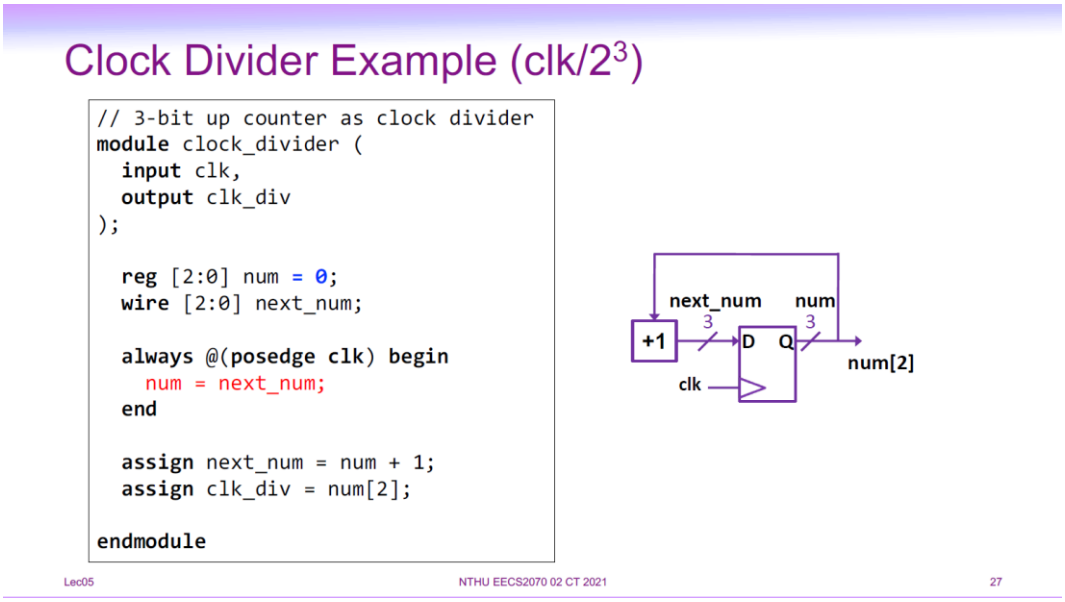
1. 實作過程

- clock_divider
  I didn't do much in clock_divider module because there is a template given by the TAs and example in the professor's lecture class. I just modify the example from the professor's slide to meet the requirements in the pdf.

```
module clock_divider #(parameter n=25) (
  input clk,
  output clk_div
);

  // add your design here

endmodule
```



- lab3_1
  In this lab need to create a synchronous LED controller that turn on and off repeatedly with two different kind of speeds, 2^{24} and 2^{27}, which could be implemented with the previous clock_divider module. Since the clock rate is depend on the speed input, I make a new wire to keep in track to the speed input with the specification below.

  If **speed == 1**: all LEDs turn on and off at the clock rate of (100 MHz / 2^{27}).
  If **speed == 0**: all LEDs turn on and off at the clock rate of (100 MHz / 2^{24}).

I made an always block with positive edge clock or reset sensitivity, and I also made a 16-bit reg to assign the next led condition. If the reset is positive, then the next led will be on. And if the reset is in negative edge, check if the enable is on or off. If it's off, the next led will be the same as the current led condition. But if it's on, the next led will be the opposite as the current one. If the led is on then the next one will be off and vice versa. After the always block is ended, I assign the led to be the next led which is decided in the always block.

```
clock_divider #(24) speed_0 (clk, clk_0);
clock_divider #(27) speed_1 (clk, clk_1);

assign clock = (speed == 0) ? clk_0 : clk_1;

always@(posedge clock, posedge rst) begin
    if(rst) begin
        led_next = 16'b1111_1111_1111_1111;
    end else begin
        if(en) begin
            if(led == 16'b1111_1111_1111_1111) begin
                led_next = 16'b0000_0000_0000_0000;
            end else begin
                led_next = 16'b1111_1111_1111_1111;
            end
        end
        else led_next = led;
    end
end
assign led = led_next;
```

- lab3_2

Here I make 2 always block, one sequential and once combinational. In the sequential block, I only make the current state, count and output to the next one (which compute in the combinational block), except when the reset is '1', I make all of the value to the initial state.

```
always@(posedge clk_div, posedge rst) begin
    if(rst) begin
        state <= S0;
        cnt <= 3'd0;
        count <= 5'd0;
        led <= ON;
    end else begin
        state <= next_state;
        cnt <= next_cnt;
        count <= next_count;
        led <= next_led;
    end
end
```

In the combinational block, I assign all of the next condition to be the current condition in advance. Then it got in into case statement with 3 different states, S0, S1 and S2. S0 is the initial state which do the flashing job.

Flashing mode

All LEDs will turn on and off repeatedly. After 6 on-off cycles, the controller goes to **Shifting mode**.

```
S0 : begin
    if(cnt == 3'd6) begin
        next_state = S1;
        next_count = 5'd0;
        next_led = SHIFT_INITIAL;
    end else begin
        if(led == ON) begin
            next_led = OFF;
        end else begin
            next_cnt = cnt + 3'd1;
            next_led = ON;
            if(cnt == 3'd0) next_cnt = 3'd1;
            else if(cnt == 3'd1) next_cnt = 3'd2;
            else if(cnt == 3'd2) next_cnt = 3'd3;
            else if(cnt == 3'd3) next_cnt = 3'd4;
            else if(cnt == 3'd4) next_cnt = 3'd5;
            else if(cnt == 3'd5) next_cnt = 3'd6;
            else next_cnt = 3'd0;
        end
    end
end
```

Since it need to turn on and off repeatedly for 6 clock cycles, I make a cnt reg to keep in track of it. I add the cnt by one if the current led is off because it always starts with off and ended in on. For the reason S0 will only be access by the S2 state or if the reset happened. Both of that state always ended with full on led, then automatically S0 will started with the led off. And if the cnt value is 6, then it will go to S1 state with the initial shifting output which looks like the explanation below.

Shifting mode

Initially, the odd-indexed LEDs (e.g., LD15, LD13, LD11, ..., LD3, LD1) are ON; the even-indexed LEDs (e.g., LD14, LD12, LD10, ..., LD2, LD0) are OFF.

E.g., the initial state of Shifting mode: (●: LED on, ○: LED off)

(LD15) ●○●○●○●○●○●○●○●○●○●○ (LD0)

Then, the LED will begin to shift from left to right (when **dir==0**) or to shift from right to left (when **dir==1**) synchronized to the clock. When all LEDs are OFF, the controller goes to **Expanding mode**.

For the S1 state, shifting mode, I also keep in track of the shifting with a signed count reg in order to know whether the two tails of the initial led is ended or not. If it reaches the end, then it will just shifting as usual without making a new tail. And if the led is off, the next state will go to S2.

```
S1 : begin
    if(led == OFF) begin
        next_state = S2;
        next_count = 5'd0;
        next_led = EXPAND_INITIAL;
    end else begin
        if(dir) begin
            if(count <= 0) next_led = led << 1;
            else next_led = {led[14:0], !led[0]};
            next_count = count - 5'd1;
        end else begin
            if(count >= 0) next_led = led >> 1;
            else next_led = {!led[15], led[15:1]};
            next_count = count + 5'd1;
        end
    end
end
```

S2, the expanding mode, which works like the below.

Expanding mode

Initially, the middle LEDs (LD8 and LD7) are ON, and others are OFF.

E.g., the initial state of Expanding mode: (●: LED on, ○: LED off)

(LD15) ○○○○○○○●●○○○○○○○ (LD0)

a. When dir==0, the LEDs will begin to expand at each clock edge. To be specific, the two LEDs next to the outermost lighted LEDs will turn on. So, the lightbar will expand gradually.

E.g., the 2nd to 4th states of Expanding mode if dir==0: (●: LED on, ○: LED off)

(LD15) ○○○○○○●●●●○○○○○○ (LD0)

(LD15) ○○○○○●●●●●●○○○○○ (LD0)

(LD15) ○○○○●●●●●●●●○○○○ (LD0)

b. When dir==1, the LEDs will begin to shrink at each clock edge. To be specific, the outermost lighted LEDs will turn off. So, the lightbar will shrink gradually until all LEDs are OFF.
E.g., the next four following states of Expanding mode if dir==1:

(LD15) ○○○○○●●●●●●○○○○○ (LD0)

(LD15) ○○○○○○●●●●○○○○○○ (LD0)

(LD15) ○○○○○○○●●○○○○○○○ (LD0)

(LD15) ○○○○○○○○○○○○○○○○ (LD0)

After all the LEDs go OFF, they will keep OFF. Afterward, if the dir switches to 0, the controller will jump to the initial state of Expanding mode.

The controller will go to **Flashing mode** only after all LEDs are ON.

To design this is a bit simple, with two conditions, when the dir = 0, then the center's 2-bit of the next_led should be one bit and the left will use the led[14:8] and the right side is led[7:1] since we need to get rid of the both tail. For the dir = 1, only if the led not off yet, I assign the first and last bit to be 0, and taking the led[15:9] and led[6:0] to be the center. But if the led is off then do nothing.

```
S2 : begin
    if(led == ON) begin
        next_state = S0;
        next_cnt = 3'd0;
        next_led = OFF;
    end else begin
        if(dir) begin
            if(led != OFF) begin
                next_led = {1'b0 , led[15:9], led[6:0], 1'b0};
            end
        end else begin
            next_led = {led[14:8], 2'b11, led[7:1]};
        end
    end
end
```

After the case statement is finished, I make a if condition for the en. If the en = 0, then all of the next conditions will be the same as the current conditions.

```
if(!en) begin
    next_state = state;
    next_count = count;
    next_cnt = cnt;
    next_led = led;
end
```

In addition, in lab3_2, I also need to change the port name in the Basys3's constraints from speed to dir.

```
set_property PACKAGE_PIN V16 [get_ports dir]
set_property IOSTANDARD LVCMOS33 [get_ports dir]
set_property PACKAGE_PIN V17 [get_ports en]
set_property IOSTANDARD LVCMOS33 [get_ports en]
set_property PACKAGE_PIN W16 [get_ports rst]
set_property IOSTANDARD LVCMOS33 [get_ports rst]
```

- lab3_3

In this lab, the LED Controller is designed with synchronous positive-edge-trigged whose clock frequency is obtained by dividing the frequency of Basys3's clock, 100MHz, by 2^{23}, 2^{24}, 2^{25} or 2^{26}, but depends on the speed input. It designed with the requirement which is shown below.

There are two LED runners, called Mr. 1 and Mr. 3.

- ✓ Mr. 1 and Mr. 3 race on the 16 LEDs of the FPGA Demo board.
- ✓ Use one single LED to represent Mr. 1.
- ✓ Use three consecutive LEDs to represent Mr. 3. (His position is determined by the middle LED.)
- ✓ If **rst** == **1**:
  - Mr. 1's position is at LD15.
  - Mr. 3's position is at LD1.

  E.g., the initial state: (●: LED on, ○: LED off)

  Mr. 1: (LD15) ●○○○○○○○○○○○○○○○ (LD0)

  Mr. 3: (LD15) ○○○○○○○○○○○○○●●● (LD0)
- ✓ If **en** == **0**: Mr. 1 and Mr. 3 will all hold at their current positions.
- ✓ If **en** == **1**:
  - ■ **Mr. 1** will begin to rotate from left to right. Its position after LD0 is LD15.
  - ■ **Mr. 3** will begin to rotate from right to left. Its position after LD15 is LD0.
  - If **speed** == **0**: **Mr. 3** runs at the clock rate of (100 MHz / 2^{25}); **Mr. 1** runs at the clock rate of (100 MHz / 2^{23}), respectively.
  - If **speed** == **1**: **Mr. 3** runs at the clock rate of (100 MHz / 2^{26}); **Mr. 1** runs at the clock rate of (100 MHz / 2^{24}), respectively.

Same as the lab3_1, I assign the clock beforehand, so then it's easier to use it in the always block. In this lab, three always blocks are used. Two of them are sequential blocks, one is for Mr.1 and the others is Mr.3. Both of the always block is similar, just to assigning the current condition to the next. However, I add a dir reg for Mr.1 since it will go to different direction if it collides with Mr.3's led. Given the hint from the TAs (position of Mr.3), I can make my code much more simple.

```
always@(posedge clk1) begin
    if(rst) begin
        led1 <= 16'b1000_0000_0000_0000;
        dir <= 1'b0;
        pos1 <= 4'd15;
    end else begin
        led1 <= led1_next;
        dir <= dir_next;
        pos1 <= pos1_next;
    end
end
```

```
always@(posedge clk3) begin
    if(rst) begin
        led3 <= 16'b0000_0000_0000_0111;
        pos3 <= 4'd1;
    end else begin
        led3 <= led3_next;
        pos3 <= pos3_next;
    end
end
```

Apart from it, there is one more combinational block for computing the next condition. Because Mr.3 direction never change, then I shift it to the left every time when en = 1 also for the position will added by 1. The rest is for the Mr.1 led which change direction every time it collide. It can be easily to check because of the position reg. I added it inside the if else condition.

- pos1 – pos3 = 2 (touch)
  - ➢ Collision Situation 1 (Touch): Mr. 1 touches one end of Mr.3

  E.g.

  Mr. 1: (LD15) ○○○  ●○○○○○○○○○○○(LD0)

  Mr. 3: (LD15) ○○○○○○●●●○○○○○○○(LD0)
- pos1 – pos3 = 1 (overlap)
  - ➢ Collision Situation 2 (Overlap): Mr. 1 overlaps one end of Mr. 3

  E.g.

  Mr. 1: (LD15) ○○○○  ●○○○○○○○○ (LD0)

  Mr. 3: (LD15) ○○○○○○●●●○○○○○○ (LD0)
- pos1 – pos3 = 0 (overlap)
  - ➢ Collision Situation 3 (Overlap): Mr. 1A overlaps the middle of Mr. 3.

  E.g.

  Mr. 1: (LD15) ○○○○○○○●○○○○○○○○ (LD0)

  Mr. 3: (LD15) ○○○○○○●●●○○○○○○○ (LD0)
- and vice versa

```
always@(*) begin
    if(en) begin
        led3_next[0] = led3[15];
        led3_next[15:1] = led3[14:0];
        pos3_next = (pos3 == 4'd15) ? 4'd0 : pos3 + 4'd1;
        if(!dir && (pos1 - pos3 == 4'd2 || pos1 - pos3 == 4'd1 || pos1 - pos3 == 4'd0)) begin
            led1_next[0] = led1[15];
            led1_next[15:1] = led1[14:0];
            pos1_next = (pos1 == 4'd15) ? 4'd0 : pos1 + 4'd1;
            dir_next = 1'b1;
        end else if(dir && (pos3 - pos1 == 4'd2 || pos3 - pos1 == 4'd1 || pos3 - pos1 == 4'd0)) begin
            led1_next[15] = led1[0];
            led1_next[14:0] = led1[15:1];
            pos1_next = (pos1 == 4'd0) ? 4'd15 : pos1 - 4'd1;
            dir_next = 1'b0;
        end else begin
            if(dir) begin
                led1_next[0] = led1[15];
                led1_next[15:1] = led1[14:0];
                pos1_next = (pos1 == 4'd15) ? 4'd0 : pos1 + 4'd1;
                dir_next = dir;
            end else begin
                led1_next[15] = led1[0];
                led1_next[14:0] = led1[15:1];
                pos1_next = (pos1 == 4'd0) ? 4'd15 : pos1 - 4'd1;
                dir_next = dir;
            end
        end
    end else begin
        led1_next = led1;
        led3_next = led3;
        dir_next = dir;
        pos1_next = pos1;
        pos3_next = pos3;
    end
end
```

For the en = 0, it is the same as lab3_2, assigning the next condition to the current. And at last is assigning the led output from Mr.1 and Mr.3 led with or gate.

2. 學到的東西與遇到的困難

In lab3_1, when I first finished my code and run it in FPGA board, it works normally. However, I got some warning in Vivado synthesis showing there is a latch, can't set and reset at the same time. Then I tried to change my code by adding the led_next reg and after synthesis, everything seems fine without warning.

For the lab3_2, I spent a lot of time in this lab because of my coding style problem. First problem happened when I run it in my FPGA board. Even though the waveform seems normal in testbench, but when I programmed it in the board, the led lights are very weird. It seems like the bulb that don't have enough battery. It keeps randomly blinking with low light. I suspect it happened because of latch. So, I erase my code and redo it. After finished it and have tried to check the waveform, I programmed it in FPGA board as well. Unfortunately, the led keep on like stuck in the reset state. I really have no idea what is happening. For this reason, I skip lab3_2 and do the lab3_3. But things don't go as the way I want. I couldn't concentrate doing the lab3_3 because I keep thinking about lab3_2. Therefore, I spent my whole day doing lab3_2 and tried everything I can. At last, I assign all the next condition like next state and next led to current condition. Surprisingly, it works and I just found out that I need to assign the next condition although it seems like it won't be used. Maybe because HDL is really different from high level language.

The challenging part of doing lab3_3 is to think of the idea keep in track of the movement. Luckily, thanks to the hint from the TAs, pointing the position of Mr.3. Before that, I was confused about how to check if Mr.3 led was separated like this, 16'b1000_0000_0011. I was thinking of making a long if else statement. But all become simple after using a position to check the collision.

3. 想對老師或助教說的話

None