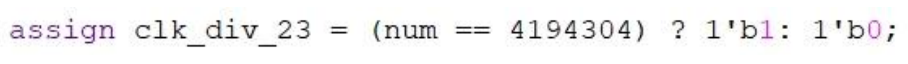
EECS 2070 02 Digital Design Labs 2020
Lab 3

學號：108062281　　姓名：莊晴雯

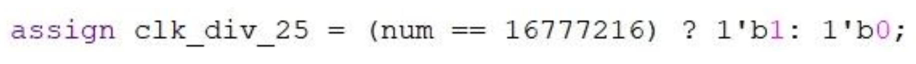
1. 實作過程

- Module: clock_divider
  To design the clock_divider, which will divide the frequency of the input clock by 2**25 to get the output clock, I refer to the example provided in the lecture. Because we want to divide it by 2**25, so the parameter n here is 25, and there is one input and one output, which are clk and clk_div respectively. For the num, we declare it as a reg of 25 bits that holds the value of 25'd0 initially, and declare next_num as 25 bits of wire. Here I used the positive clock edge and inside its always block, we need to set the value of num as next_num. Then outside of the always block, we use assign for the next_num to store the value of num+1 and assign for clk_div to store the value of num[n-1].

  However, I modified the code a bit for lab3_2, and lab3_3. Not only there are 2 kinds of clock divider that we need to implement (2**23 and 2**25), I set the output as either 1 or 0, which acts as something like a flag for representation of each clock divisor. 1 represents that the clock divider of that particular value (either 2**23 and 2**25) is currently on, and it will always be on every time num is reaches to either 2**23 and 2**25 as shown on the picture below, based on the clock divider implemented. The rest are pretty much the same as the original clock divider.



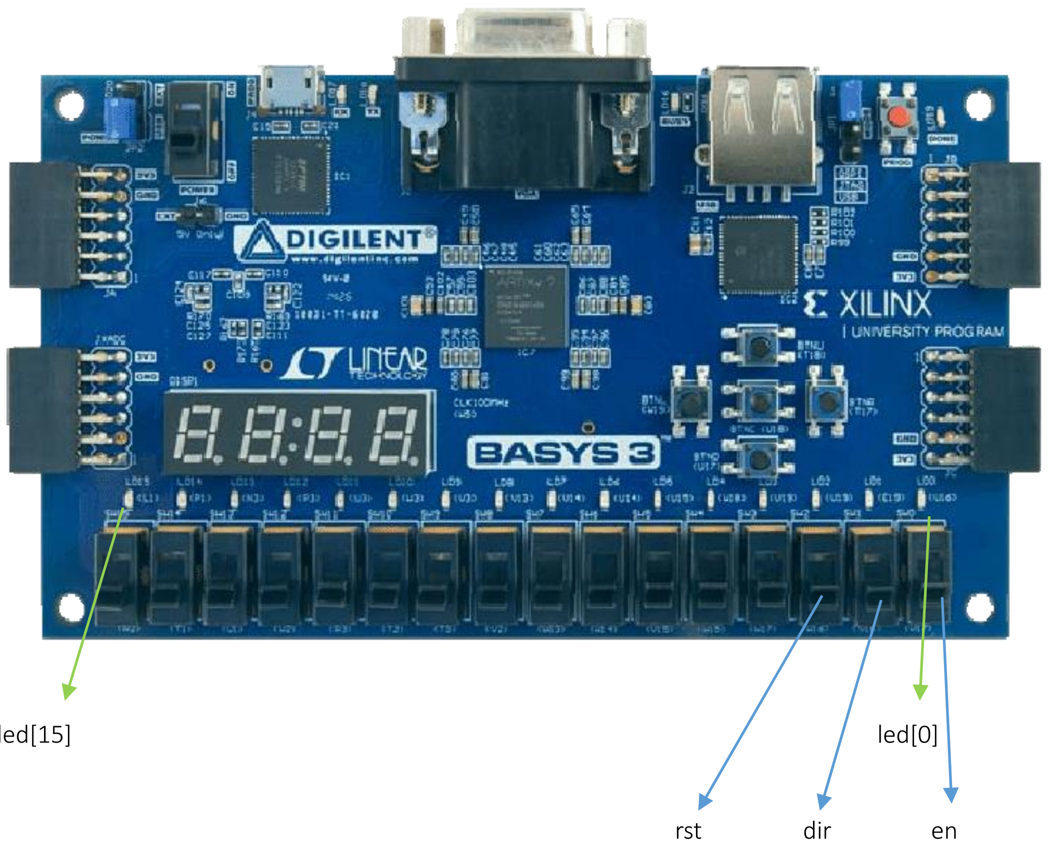
```
assign clk_div_23 = (num == 4194304) ? 1'b1: 1'b0;
```



```
assign clk_div_25 = (num == 16777216) ? 1'b1: 1'b0;
```

- lab3_1
  In lab3_1 we are required to write a Verilog module of the LED controller which is synchronous with the positive clock edges, in which the clock frequency is obtained by dividing the frequency of Basys3's on-board clock. There will be 1 LED on initially on the leftmost side (led[15]), and it will move based on the switch. If rst == 1, in other words, if rst switch is being turned

on, then the left-most LED is on while the others will be off. The LED will then shift to the left or right direction if en is being turned on (en == 1'b1). The direction itself is also determined by the switch dir, in which if dir == 1'b1, the LED will begin to shift from right to left synchronized to the clock. There is also one condition that if the LED reaches the end (led[0] or led[15]), it will continue to the other side of the LED (if reaches led[15], continue to led[0]; if reaches led[0], continue to led[15]). Otherwise, if dir == 1'b0, then the LED will begin to shift from left to the right synchronized to the clock. While if en is equal to 1'b0, then the LED will stay at its place unchanged.



In this design, I implemented the clock divider inside the clock_divider module to get the clock frequency wanted, which the same way as I implemented it the first time, and it will return the value of clk_div, the clock frequency obtained by dividing the frequency of Basys3's on-board clock by 2**25. As for the LED to shift itself, I implemented the design inside the lab3_1 module with the inputs and ouputs provided. I set the inputs as wire and the 16-bit output led as reg, and initialize its value as 16'b1000000000000000 for the leftmost led to be turned on the first time. I then use the module clock_divider and pass clk to it for me to get the output clk_div, which is the clock frequency is obtained by dividing the frequency of Basys3’s on-board clock by 2**25.

```
clock_divider clkdiv(.clk(clk), .clk_div(clk_div));

always@(posedge clk_div or posedge rst)
```

I then use this new clock frequency clk_div for the always block, along with posedge rst to avoid delay when performing rst. Inside the always block, there are two if statements at first, which is when rst is equal to 1’b1 or rst is equal to 1’b0. rst is put at the outmost of the other if statements because it has highest priority. When the others have value too but if rst ==1’b1, then LED will be on at the leftmost side no matter what other conditions are. So here if rst == 1’b1, then we set the value of led to 16'b1000000000000000.

```
if(rst == 1'b1)
begin
    led = 16'b1000000000000000;
end
```

The other condition of rst, which is when rst == 1’b0, there are three other conditions that can be found inside rst == 1’b0, which are:

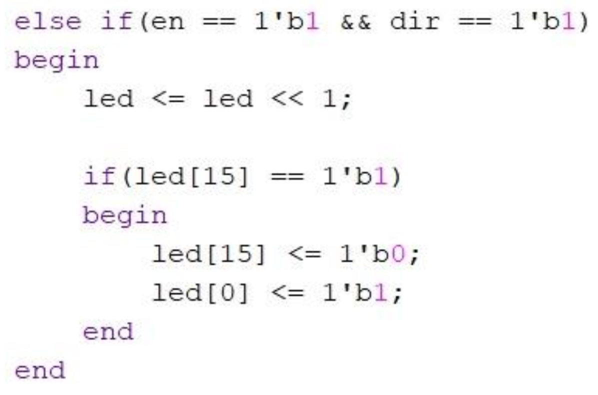
1. en == 1'b0 : LED stays at its position unchanged.

```
if(en == 1'b0)
begin
    led <= led;
end
```

2. en == 1'b1 && dir == 1'b0 : LED shifts to the right by 1 position.

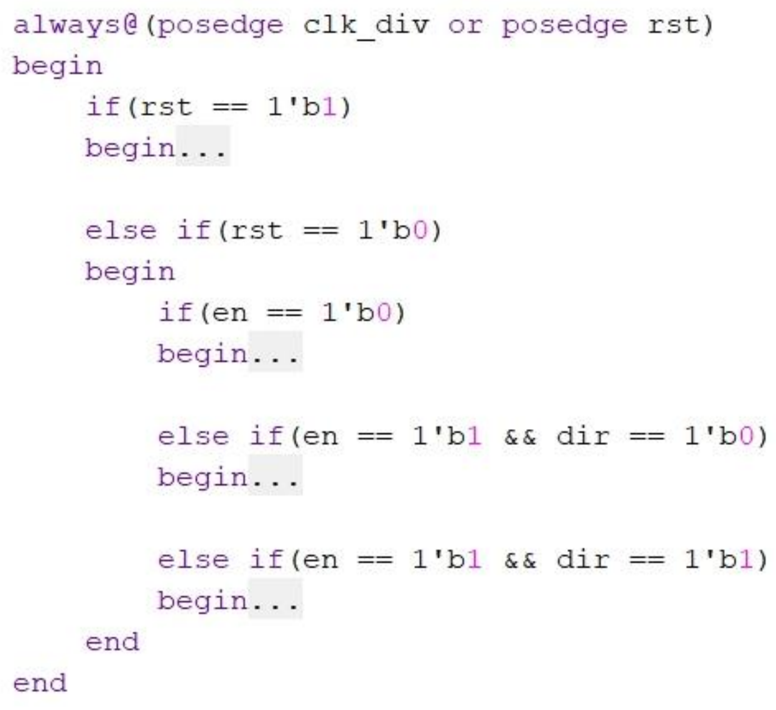
```
else if(en == 1'b1 && dir == 1'b0)
begin
    led <= led >> 1;

    if(led[0] == 1'b1)
    begin
        led[0] <= 1'b0;
        led[15] <= 1'b1;
    end
end
```

3. en == 1'b1 && dir == 1'b1   : LED shifts to the left by 1 position.



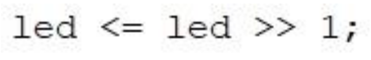
```
else if(en == 1'b1 && dir == 1'b1)
begin
    led <= led << 1;

    if(led[15] == 1'b1)
    begin
        led[15] <= 1'b0;
        led[0] <= 1'b1;
    end
end
```

The conditions mentioned above is implemented like this in my design inside the always block:



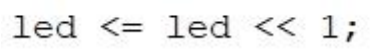
```
always@(posedge clk_div or posedge rst)
begin
    if(rst == 1'b1)
    begin...

    else if(rst == 1'b0)
    begin
        if(en == 1'b0)
        begin...

        else if(en == 1'b1 && dir == 1'b0)
        begin...

        else if(en == 1'b1 && dir == 1'b1)
        begin...
    end
end
```

To shift, I use << to shift led 1 position to the left, and >> to shift led 1 position to the right. Which is why, the code will somehow look like:



```
led <= led >> 1;
```

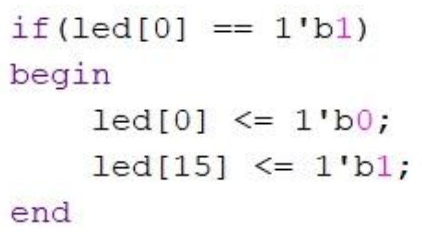
(right shift, then assign it back to led)



```
led <= led << 1;
```

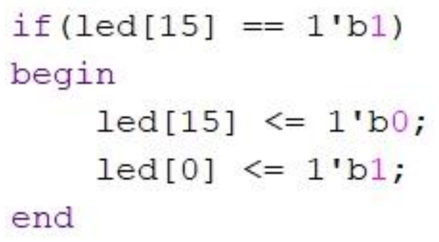
(left shift, then assign it back to led)

Additionally, there are another if statement in each of the en == 1'b1 && dir == 1'b0 and en == 1'b1 && dir == 1'b1 conditions, which is when LED hits the end of the LED lights (led[0] or led[15]). The only collision condition that is possible for when led shifts to the right is that it hits the rightmost LED (led[0] is on, led[0] == 1’b1). Therefore, the LED will continue lighting up at led[15], so we can write it as:



```
if(led[0] == 1'b1)
begin
    led[0] <= 1'b0;
    led[15] <= 1'b1;
end
```

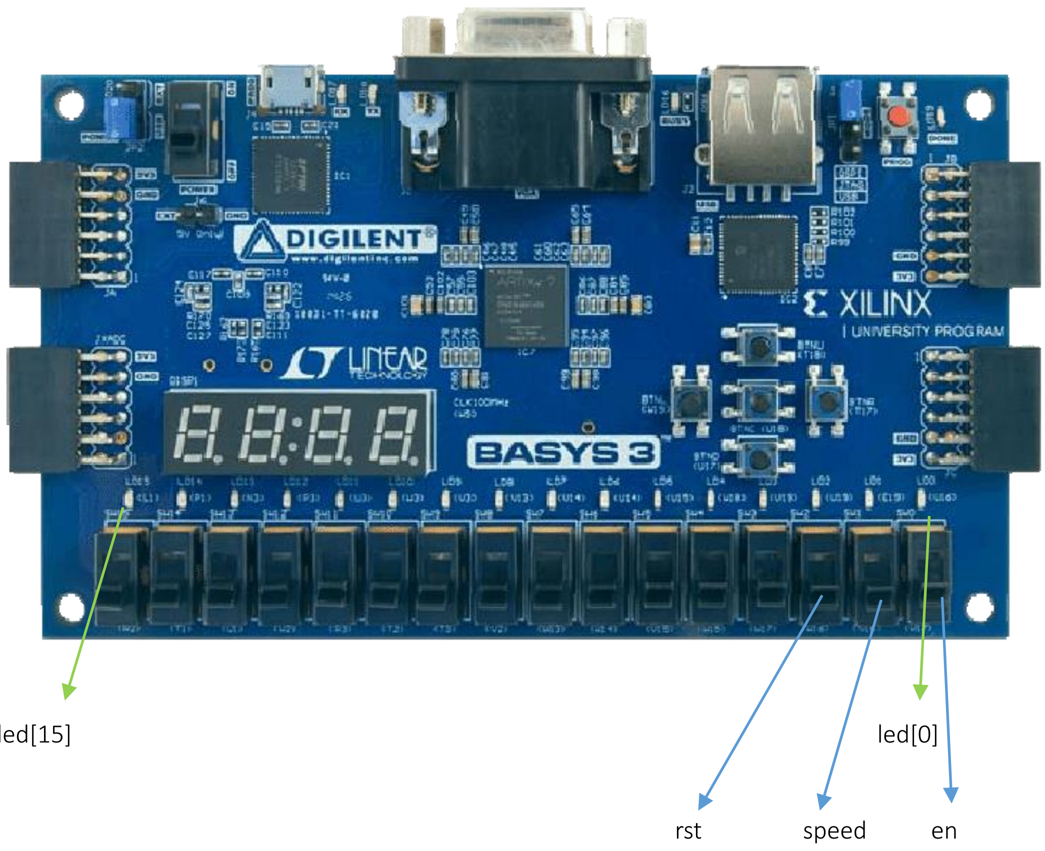
When led[0] lights up, we turn if off again and at the same time turn led[15] on, which is why I used the non-blocking assignment here. The other condition is when led shifts to the left and it hits the leftmost LED (led[15] is on, led[15] == 1'b1), the LED continues to light up at led[0], and the code are as follows:



```
if(led[15] == 1'b1)
begin
    led[15] <= 1'b0;
    led[0] <= 1'b1;
end
```

When led[15] lights up, we turn if off again and at the same time turn led[0] on, which is why I used the non-blocking assignment here.

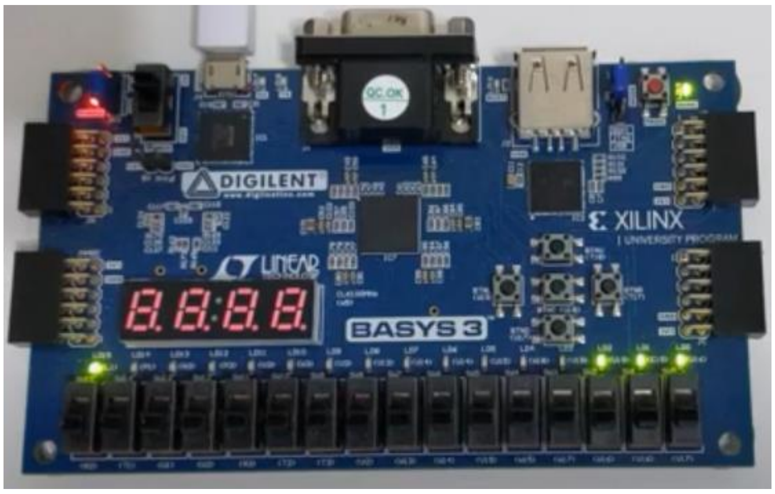
For lab3_2 and lab3_3:



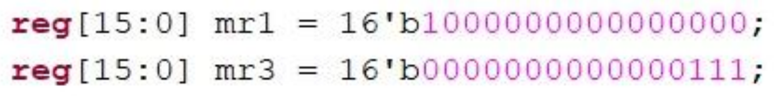
- lab3_2

  In lab3_2 we are required to write a Verilog module of the LED Controller which is synchronous with the clock whose frequency is obtained by dividing the frequency of Basys3's on-board 100MHz clock by either 2**23 or 2**25. This problem is implemented with having two LED runners, Mr. 1 and Mr. 3, who will race on the 16 LEDs of the FPGA Demo board.

  Mr. 1 is represented by one single LED and his initial position is at led[15], while Mr. 3 is represented by three consecutive LEDs and his initial position covers led[2], led[1], and led[0], as shown on the picture below.

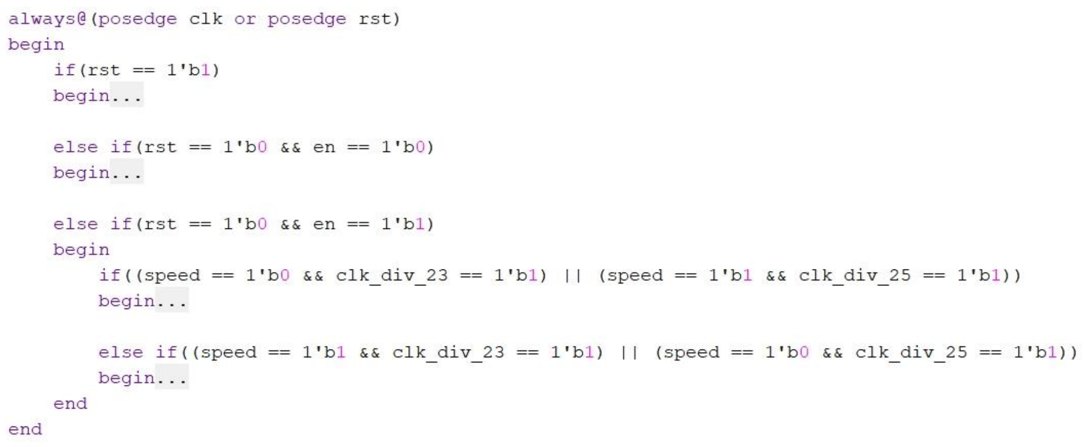


To set their initial position so that the LED lights can light up like above picture, I need to declare an initial value for both Mr. 1 and Mr. 3:



```
reg[15:0] mr1 = 16'b1000000000000000;
reg[15:0] mr3 = 16'b0000000000000111;
```

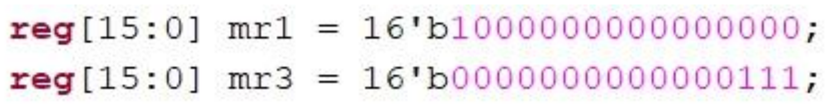
Mr. 1 will always shift from left to right, while Mr. 3 will always shift from right to left. As for the switches, if rst == 1’b1 then Mr. 1 and Mr. 3 will go back to their initial position (Mr.1 at led[15], Mr. 3 at led[2], led[1], led[0]), and if rst is not equal to 1’b1 and en == 1’b0, then Mr. 1 and Mr. 3 stays at their position they are in right now. As for when rst == 1’b0 and en == 1’b1, there is one more switch condition that will determine the speed they will be moving at, one is when speed == 1’b0, Mr. 3 will run at the clock rate of 100 MHz/(2**25) and Mr. 1 will run at the clock rate of 100 MHz/(2**23). The other condition is that when speed == 1’b1, Mr. 3 will run at the clock rate of 100 MHz/(2**23) while Mr. 1 will run at the clock rate of 100 MHz/(2**25). The structure of the design based on the condition mentioned above somehow looked like this:



```
always@(posedge clk or posedge rst)
begin
    if(rst == 1'b1)
    begin...

    else if(rst == 1'b0 && en == 1'b0)
    begin...

    else if(rst == 1'b0 && en == 1'b1)
    begin
        if((speed == 1'b0 && clk_div_23 == 1'b1) || (speed == 1'b1 && clk_div_25 == 1'b1))
        begin...

        else if((speed == 1'b1 && clk_div_23 == 1'b1) || (speed == 1'b0 && clk_div_25 == 1'b1))
        begin...
    end
end
```

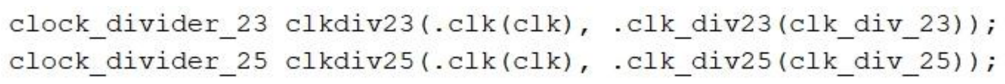
In this design, I implemented the clock divider using two different modules, clock_divider_23 and clock_divider_25, each for the frequencies of 100 MHz/(2**23) and 100 MHz/(2**25) respectively. This time, the output of the two modules are either 0 or 1, which will act like a flag as a representative of each clock divisor. It will become 1 when num reaches 2**23 or 2**25 (depending on which clock divisor it is). Again, I used posedge rst inside the always@ to avoid delay when performing rst.

Inside the lab3_2 module, I declare the 16-bit led as wire for me to be able to assign its value using assign outside the always block. I also declare rst as wire, along with clk_div_23 and clk_div_25. For the new variables mr1 and mr3 which is used to indicate the two runners, I declared them as a 16-bit reg with their own initialized starting value which is their starting position.



```
reg[15:0] mr1 = 16'b1000000000000000;
reg[15:0] mr3 = 16'b0000000000000111;
```

I then used the two clock divider modules (clock_divider_23 and clock_divider_25) to get the value of the flag of the clock frequency as a mark for when it is on or not, then store them in clk_div_23 and clk_div_25 respectively.



```
clock_divider_23 clkdiv23(.clk(clk), .clk_div23(clk_div_23));
clock_divider_25 clkdiv25(.clk(clk), .clk_div25(clk_div_25));
```

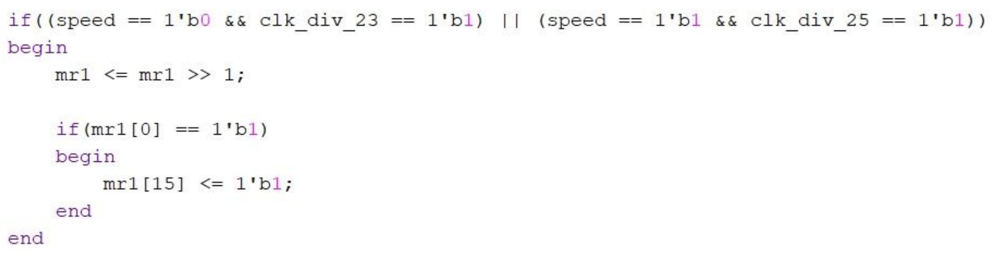
Inside the always@(posedge clk or posedge rst) block, there are a few conditions that should be applied:

1. rst == 1'b1
   - ➢ Sets mr1 and mr3 back to their initial position:
     mr1 <= 16'b1000000000000000;
     mr3 <= 16'b0000000000000111;

2. rst == 1'b0 && en == 1'b0
   - ➢ mr1 and mr3 stays at their current position:
     mr1 <= mr1;
     mr3 <= mr3;

3. rst == 1'b0 && en == 1'b1:
   There are two other conditions inside this if statement:
   - ❖ (speed == 1'b0 && clk_div_23 == 1'b1) ||
     (speed == 1'b1 && clk_div_25 == 1'b1)
   - ➢ When speed == 1'b0 and clk_div_23 == 1 OR when speed == 1'b1 and clk_div_25 == 1, then mr1 will shift to the right. The condition is as above because when speed is equal to 1'b0, then mr1 moves at the clock rate of 100 MHz/(2**23), and when speed is equal to 1'b1, then then mr1 moves at the clock rate of 100 MHz/(2**25).

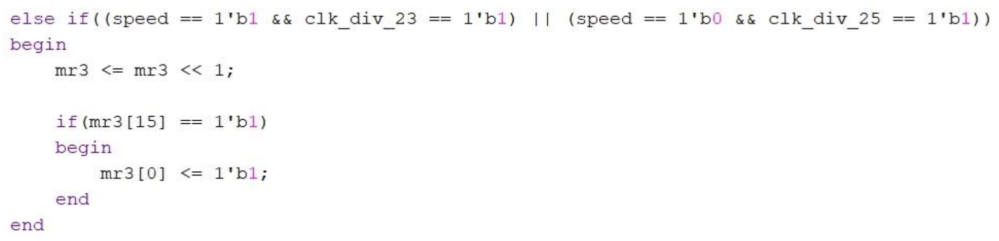
Inside, there is still another condition when mr1 runs until he reaches led[0]. He will then continue his run from led[15], which is why if(mr1[0] == 1'b1), then mr1[15] <= 1'b1, mr1[15] is assigned to the value 1’b1.



```
if((speed == 1'b0 && clk_div_23 == 1'b1) || (speed == 1'b1 && clk_div_25 == 1'b1))
begin
    mr1 <= mr1 >> 1;

    if(mr1[0] == 1'b1)
    begin
        mr1[15] <= 1'b1;
    end
end
```

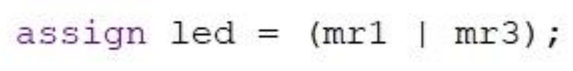
- ❖ (speed == 1'b1 && clk_div_23 == 1'b1) ||
  (speed == 1'b0 && clk_div_25 == 1'b1)
- ➢ When speed == 1 and clk_div_23 == 1 OR when speed == 0 and clk_div_25 == 1, then mr3 will shift to the left. The condition is as above because when speed is equal to 1, then mr3 moves at the clock rate of 100 MHz/(2**23), and when speed is equal to 0, then then mr3 moves at the clock rate of 100 MHz/(2**25).

  Inside, there is still another condition when mr3 runs until he reaches led[15]. He will then continue his run from led[0], which is why if(mr3[15] == 1'b1), then mr3[0] <= 1'b1, mr3[0] is assigned to the value 1’b1.



```
else if((speed == 1'b1 && clk_div_23 == 1'b1) || (speed == 1'b0 && clk_div_25 == 1'b1))
begin
    mr3 <= mr3 << 1;

    if(mr3[15] == 1'b1)
    begin
        mr3[0] <= 1'b1;
    end
end
```

At the end, we can see that every position of Mr. 1 and Mr. 3 is stored separately inside mr1 and mr3 respectively, when they move, stays, etc. They will then be combined together and put inside led for the LED lights on the FPGA board to light up. I combine them using OR (|) and directly assign it to led.

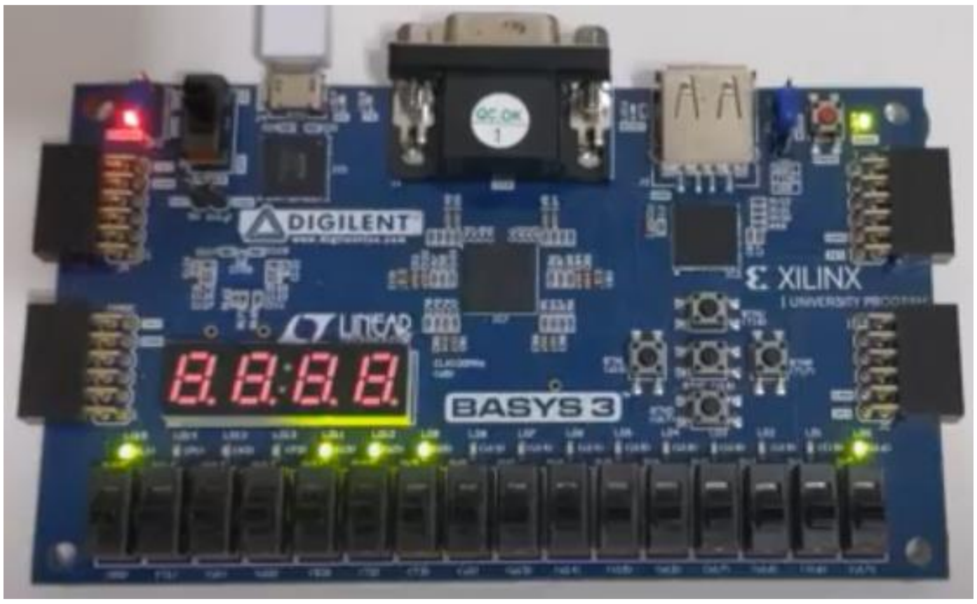


```
assign led = (mr1 | mr3);
```

- lab3_3

  In this lab3_3, we are required to write a Verilog module of the LED Controller which is synchronous at the clock rates of either 100 MHz/(2**23) or 100 MHz/(2**25), dividing from Basys3's 100MHz clock.

  This time, there are three LED runners, Mr. 1A, Mr. 1B, and Mr. 3, with their initial position led[15] for Mr. 1A, led[11], led[10], led[9] for Mr. 3, and led[0] for Mr. 1B, as shown on the picture below.

  Here, if rst == 1'b1, then each runner will be set back to their initial positions. While for when rst == 1'b0, there will be several conditions, namely when en == 1'b0, Mr. 1A, Mr. 1B, and Mr. 3 will stay at their current positions, and if en == 1'b1, Mr. 1A will begin to shift from left to right, Mr. 1B will begin to shift from right to left, and Mr. 3 will begin to shift from left to right.

  The speed of each runner will also depend on the speed switch. If speed == 1'b0, Mr. 3 runs at the clock rate of 100 MHz/(2**25), and Mr. 1A and Mr. 1B run at the clock rate of 100 MHz/(2**23). If speed == 1'b1, then Mr. 3 runs at the clock rate of 100 MHz/(2**23), and Mr. 1A and Mr. 1B run at the clock rate of 100 MHz/(2**25).

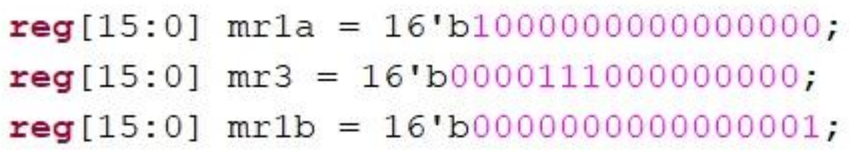
Mr. 1A and Mr. 1B will automatically change their running direction when they reach the right-most and left-most end, and Mr. 3's moves is limited to led[10] and led[5] (take the middle of Mr. 3 as the position), so he will change his direction when he reaches his own end. When Mr. 1A or Mr. 1B collides with Mr. 3, the one collides with Mr. 3 will change his direction but Mr. 3 will not. There are 3 conditions for overlapping:

1. Mr. 1 touches the edge of Mr. 3.
2. Mr. 1 overlaps one side of Mr. 3, this happens when there is still one spot left untouched by the runners which is right in the middle of them, and they both simultaneously touch it together, resulting in their collision there on the one spot of the LED board.
3. Mr. 1 overlaps the middle of Mr. 3, this happens when one of them runs faster and resulting in the collision with the middle of Mr.3.

I implemented the clock divider using two different modules, clock_divider_23 and clock_divider_25, each for the frequencies of 100 MHz/(2**23) and 100 MHz/(2**25) respectively. This time, the output of the two modules are either 0 or 1, which will act like a flag as a representative of each clock divisor. It will become 1 when num reaches 2**23 or 2**25 (depending on which clock divisor it is). Again, I used posedge rst inside the always@ to avoid delay when performing rst.

Then inside the lab3_3 module, I declare the 16-bit led as wire for me to be able to assign its value using assign outside the always block. I also declare rst as wire, along with clk_div_23 and clk_div_25.

For the new variables mr1a, mr1b, and mr3 which is used to indicate the three runners, I declared them as a 16-bit reg with their own initialized starting value which is their starting position.

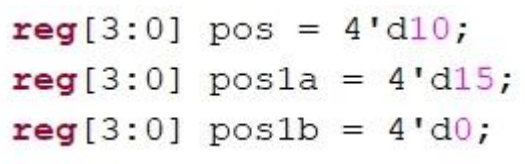


```
reg[15:0] mr1a = 16'b1000000000000000;
reg[15:0] mr3 = 16'b0000111000000000;
reg[15:0] mr1b = 16'b0000000000000001;
```

I also declared flags (flag1a, flag1b, flag3) for each of the runners to determine the direction of their movements, which I first set their values to 1’b0. The rules for the flags are as follows:

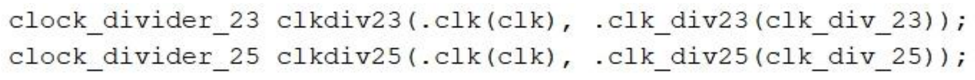
1. flag1a == 1’b0, mr1a shift left; flag1a == 1’b1, mr1a shift right.
2. flag1b == 1’b0, mr1b shift right; flag1b == 1’b1, mr1b shift left.
3. flag3 == 1’b0, mr3 shift left; flag3 == 1’b1, mr3 shift right.

Lastly, I declared position variables (pos for Mr. 3, pos1a for Mr. 1A, pos1b for Mr. 1b) to get the position of each runners which will decrease or increase in every moves of each runners, which is used to determine the collision between mr1a and mr1b with mr3. I initialized their starting value as their starting positions for each of the runners. For Mr. 3, I take the middle part of the three LED lights as his position.



```
reg[3:0] pos = 4'd10;
reg[3:0] pos1a = 4'd15;
reg[3:0] pos1b = 4'd0;
```

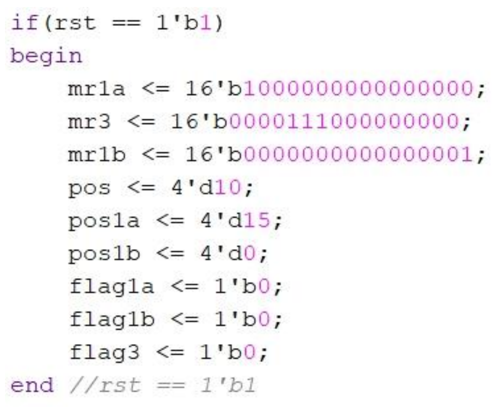
I then used the two clock divider modules (clock_divider_23 and clock_divider_25) to get the value of the flag of the clock frequency as a mark for when it is on or not, then store them in clk_div_23 and clk_div_25 respectively.



```
clock_divider_23 clkdiv23(.clk(clk), .clk_div23(clk_div_23));
clock_divider_25 clkdiv25(.clk(clk), .clk_div25(clk_div_25));
```

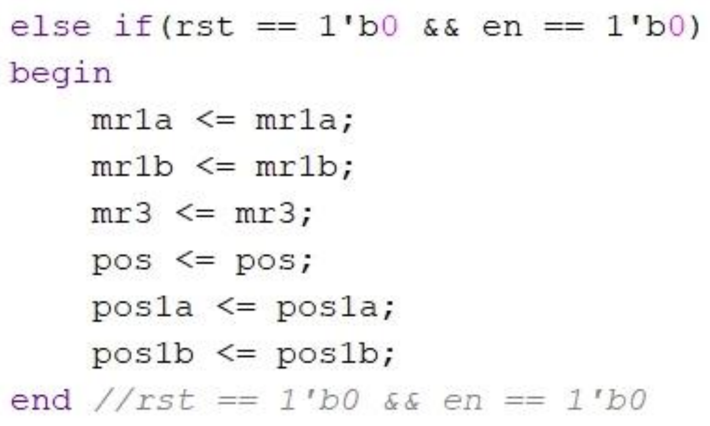
Inside of the always@(posedge clk or posedge rst) block, there are 3 big if conditions:

1. if(rst == 1'b1)
    - ➢ Everything is set back to its first place conditions. Mr1a, mr1b, and mr3 are set to their first initial positions, the same thing goes for each of their positions, and all flags are set back to 1’b0.



```
if(rst == 1'b1)
begin
    mr1a <= 16'b1000000000000000;
    mr3 <= 16'b0000111000000000;
    mr1b <= 16'b0000000000000001;
    pos <= 4'd10;
    pos1a <= 4'd15;
    pos1b <= 4'd0;
    flag1a <= 1'b0;
    flag1b <= 1'b0;
    flag3 <= 1'b0;
end //rst == 1'b1
```

2. if(rst == 1'b0 && en == 1'b0)
    - ➢ Everything stays at their current state. mr1a, mr1b, and mr3 stays at their current positions, each position variables also remain where they are.



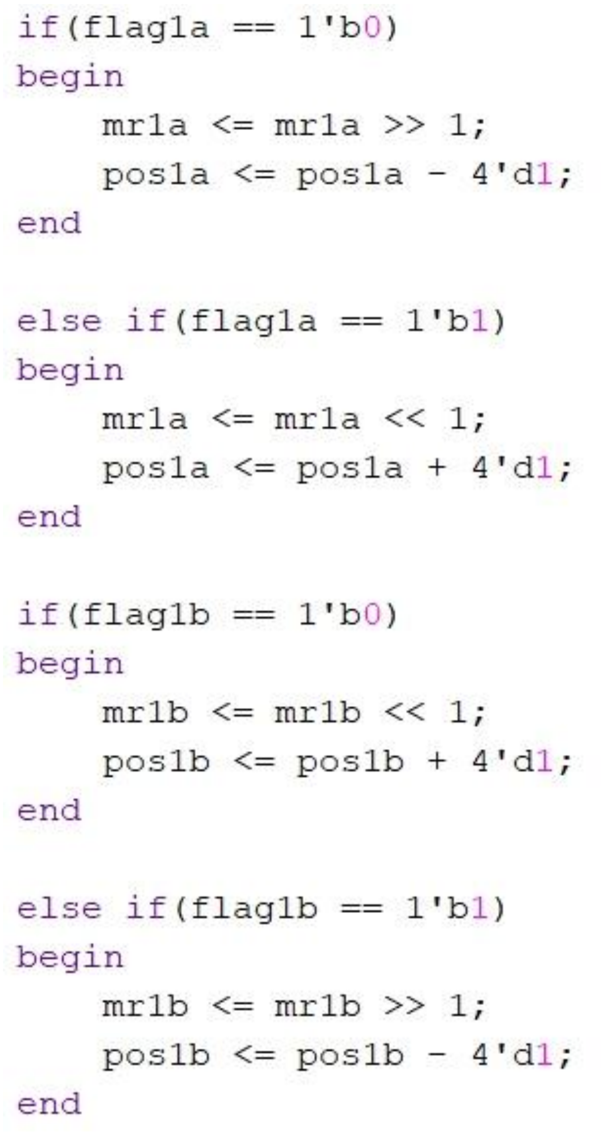
```
else if(rst == 1'b0 && en == 1'b0)
begin
    mr1a <= mr1a;
    mr1b <= mr1b;
    mr3 <= mr3;
    pos <= pos;
    pos1a <= pos1a;
    pos1b <= pos1b;
end //rst == 1'b0 && en == 1'b0
```

3. if(rst == 1'b0 && en == 1'b1)
    - ➢ Inside here there are two conditions of speed, which are when speed == 1’b0 or speed == 1’b1.

      speed == 0

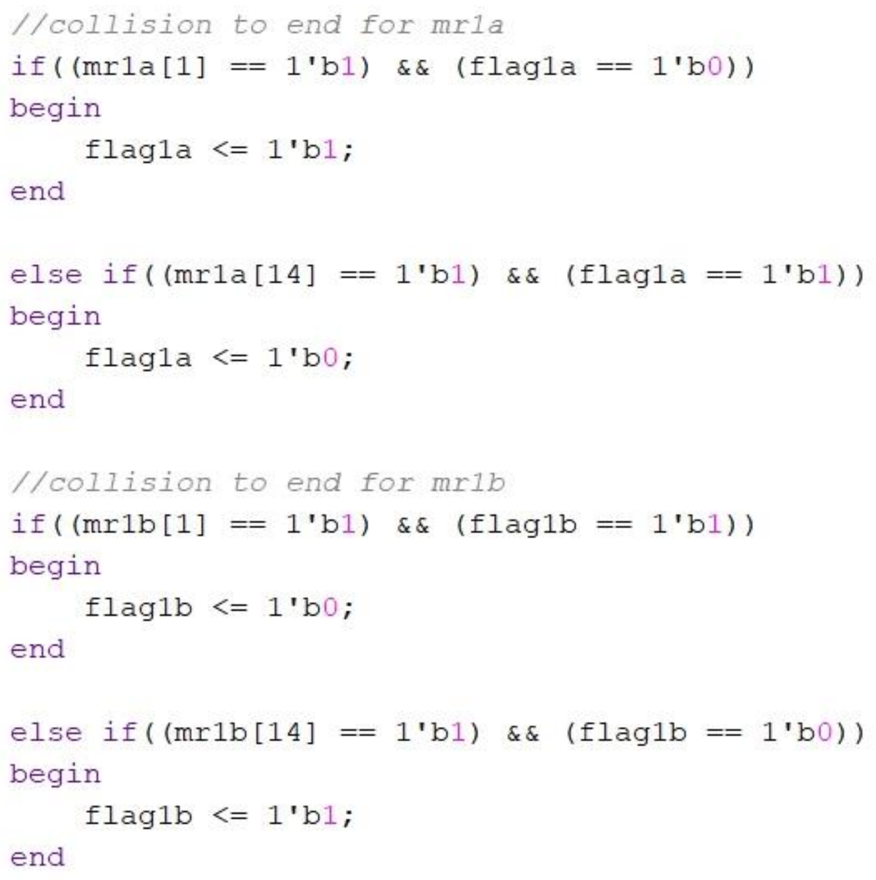
      mr1a and mr1b runs at the clock rate of 100 MHz/(2**23), and mr3 runs at the clock rate of 100 MHz/(2**25), so I used two more if statements for clk_div_23 and clk_div_25.

Inside the if(clk_div_23 == 1'b1), the movements of mr1a and mr1b is also specified again. If flag1a (flag for mr1a) == 1’b0, then mr1a shifts to the right for one position and his position (pos1a) will decrease by one at the same time. If flag1a == 1’b1, then mr1a shifts to the left by one position and his position will increase by one. For mr1b, if flag1b == 1’b0, then mr1b will shift to the left by one position and pos1b will increase by one. If flag1b == 1’b1, then mr1b will shift to the right by one position and pos1b will decrease by one simultaneously, which is why I used non-blocking assignment.



```
if(flag1a == 1'b0)
begin
    mr1a <= mr1a >> 1;
    pos1a <= pos1a - 4'd1;
end

else if(flag1a == 1'b1)
begin
    mr1a <= mr1a << 1;
    pos1a <= pos1a + 4'd1;
end

if(flag1b == 1'b0)
begin
    mr1b <= mr1b << 1;
    pos1b <= pos1b + 4'd1;
end

else if(flag1b == 1'b1)
begin
    mr1b <= mr1b >> 1;
    pos1b <= pos1b - 4'd1;
end
```

There are also conditions to detect when mr1a and mr1b reaches the end. If mr1a[1] == 1’b1 and the flag of mr1a == 1’b0, then the flag1a will change to 1’b1 for it to change its direction and update the position. If mr1a[14] == 1’b1 and the flag of mr1a == 1’b1, then the flag1a will change to 1’b0 for it to change its direction and update the position. If mr1b[1] == 1’b1 and the flag of mr1b == 1’b1, then the flag1b will change to 1’b0 for it to change its direction and update the position. Lastly, when mr1b[14] == 1’b1 and the flag of mr1b == 1’b0, then the flag1b will change to 1’b1 for it to change its direction and update the position.



```
//collision to end for mr1a
if((mr1a[1] == 1'b1) && (flag1a == 1'b0))
begin
    flag1a <= 1'b1;
end

else if((mr1a[14] == 1'b1) && (flag1a == 1'b1))
begin
    flag1a <= 1'b0;
end

//collision to end for mr1b
if((mr1b[1] == 1'b1) && (flag1b == 1'b1))
begin
    flag1b <= 1'b0;
end

else if((mr1b[14] == 1'b1) && (flag1b == 1'b0))
begin
    flag1b <= 1'b1;
end
```

As for the collisions of mr1a or mr1b to mr3, we will use the positions of each runner to detect it. The position of mr3 is the middle of the three LEDs. When mr1 touched the edge of mr3, their difference for the position must be 2. When mr1 overlaps one side of mr3, then the difference of their position must be 1. Lastly, when mr1 overlaps the middle part of mr3, then they will not have any difference in their position. Therefore, pos1a == pos for mr1a and mr3, and pos == pos1b for mr1b and mr3. Because when flag == 1’b0 I set every runner’s direction to their initial running direction, then when each flag is equal to 1’b1 then the runner will run in the opposite direction of its initial direction. Because Mr. 1 first runs using their initial direction, when they collide with Mr. 3 they automatically will change their running direction, which is the opposite of their initial direction. Therefore, their flag will be equal to 1’b1. This is implemented by the code below:

```
//collision of mr1 and mr3
if((pos1a - pos == 4'd2) || (pos1a - pos == 4'd1) || (pos1a == pos))
begin
    flag1a <= 1'b1;
end

else if((pos - pos1b == 4'd2) || (pos - pos1b == 4'd1) || (pos == pos1b))
begin
    flag1b <= 1'b1;
end
```

When clk_div_25 == 1, then it is time for mr3 to run. mr3 runs to the left direction when flag3 == 1’b0, so his position will increase by one. When mr3 touches the end when it shifts to the left, the only condition for mr3 to hit the end is when led[11] lit up, which is if(mr3[11] == 1'b1). Inside this if statement, then mr3 will shift back to the right, flag3 will become 1’b1 and his position will change to 4’d9. The other condition is when flag3 == 1’b1. This time, mr3 shifts to the right and his position will decrease by one. When mr3 touches the end when it shifts to the right, the only condition for mr3 to hit the end is when led[4] lit up, which is if(mr3[4] == 1'b1). Inside this if statement, then mr3 will shift back to the left, flag3 will become 1’b0 and his position will change to 4’d6.

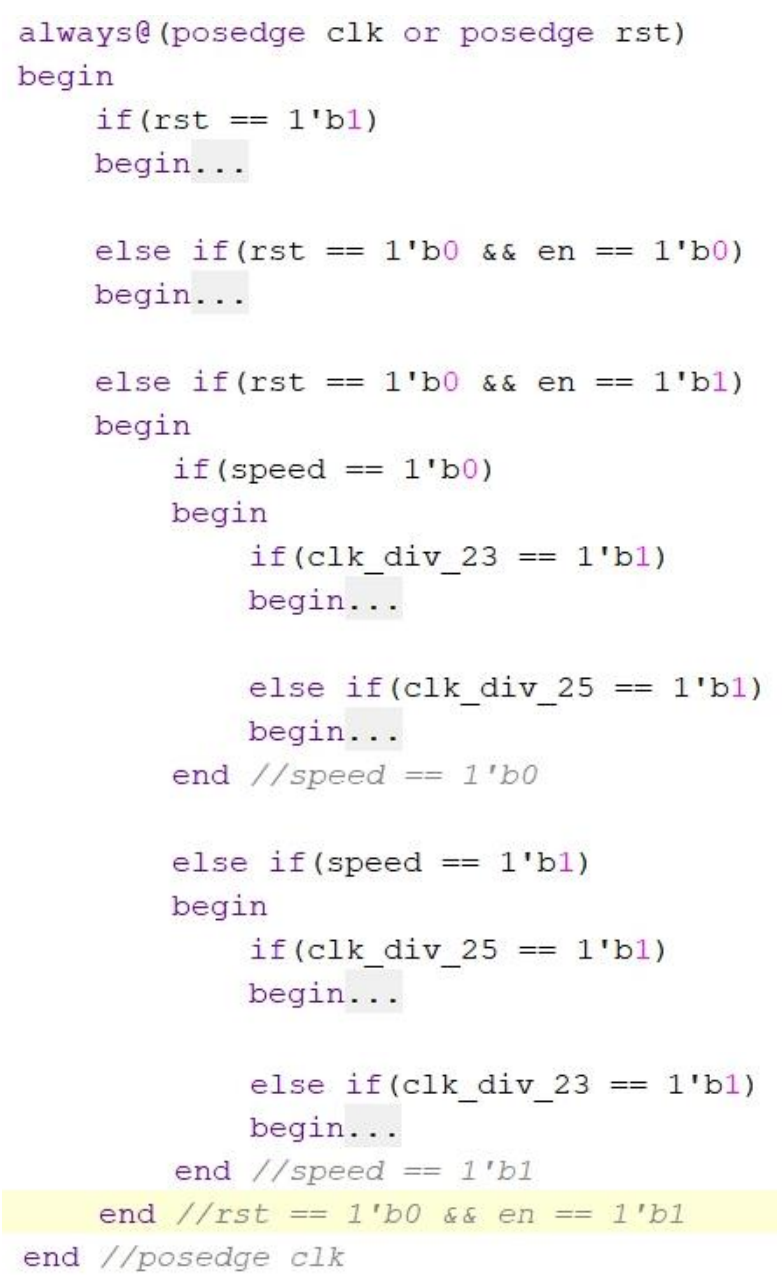
```
if(flag3 == 1'b0)
begin
    mr3 <= mr3 << 1;
    pos <= pos + 4'd1;

    //mr3 touch end
    if(mr3[11] == 1'b1)
    begin
        mr3 <= mr3 >> 1;
        flag3 <= 1'b1;
        pos <= 4'd9;
    end
end //flag3 == 1'b0
```

```
else if(flag3 == 1'b1)
begin
    mr3 <= mr3 >> 1;
    pos <= pos - 4'd1;

    //mr3 touch end
    if(mr3[4] == 1'b1)
    begin
        mr3 <= mr3 << 1;
        flag3 <= 1'b0;
        pos <= 4'd6;
    end
end //flag3 == 1'b1
```

speed == 1

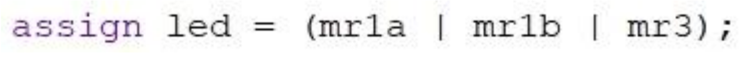
The conditions are really similar to when speed == 1’b0. It is just mr1a and mr1b runs at the clock rate of 100 MHz/(2**25), and mr3 runs at the clock rate of 100 MHz/(2**23), so I just flipped the conditions of the if statements. The rest are all the same as speed == 1’b0.

The overall structure for the design based on the conditions is as follows:



```
always@(posedge clk or posedge rst)
begin
    if(rst == 1'b1)
    begin...

    else if(rst == 1'b0 && en == 1'b0)
    begin...

    else if(rst == 1'b0 && en == 1'b1)
    begin
        if(speed == 1'b0)
        begin
            if(clk_div_23 == 1'b1)
            begin...

            else if(clk_div_25 == 1'b1)
            begin...
        end //speed == 1'b0

        else if(speed == 1'b1)
        begin
            if(clk_div_25 == 1'b1)
            begin...

            else if(clk_div_23 == 1'b1)
            begin...
        end //speed == 1'b1
    end //rst == 1'b0 && en == 1'b1
end //posedge clk
```

In the end, outside of the always block, we just need to assign led as (mr1a | mr1b | mr3) to put all the runners inside the LED board on the FPGA so that every runner can be displayed correctly.



```
assign led = (mr1a | mr1b | mr3);
```

- lab3_1, lab3_2, lab3_3
  I must also modify the constraint so that the design we code can work perfectly on the Basys3 FPGA board when programming the board. We need to set the correct variable on the correct place for the device to program correctly, in this case, we use switch, clk, and led. So I need to uncomment the switch, clk, and led part on the constraint and write the correct variable as the parameter which is the same variable I used on the source code.

2. 學到的東西與遇到的困難

I was at first confused on how to work with the Basys3 FPGA board, until I re-read the teaching materials, consult with my friend, and looked for answers on the online forum to solve my problems. I was also confused on how to work on the xdc file (constraint) until I read the teaching materials carefully again. I learned a lot on how to work with the xdc file and work with the LED lights on the FPGA board. After learning and knowing how it works while programming the lab3_1, I started getting more understanding about how everything works here.

After finishing lab3_1 and the clock_divider module without many problems encountered, I started moving on to lab3_2 and lab3_3, and here is where problems starts to kick in and more questions started to pop out. At first I separated clk_div_23 and clk_div_25 and joined the variables together with an OR gate on the normal posedge clk. While using this method, I got errors, wrong implementations that are hard to debug, and many confusing things such as too much LED lights appearing all at once, LED lights suddenly disappeared, LED lights skipped while shifting or shifted too fast that it looked like it skipped a few steps, delay while restart, and other errors encountered. I tried consulting with my friend and he told me to not separate clk_div_23 and clk_div_25 as it might result in unwanted delays or errors. Therefore, I treated clk_div_23 and clk_div_25 like a flag instead, so the value will either only be 1 or 0, which is like how I implemented it in my design right now.

I realized too that I need simplify my code and avoid duplicating, which I fixed in the end and got some of my errors solved. Last thing, I also learned that I need to make my if and else statements more organized, since there are so many if statements used in this lab03. My if statements used to be so messy that it results in a wrong display of the LED lights. After re-arranging the if statements, everything worked as how it is supposed to be. Last thing, during the demo, one of the TAs told me to type in posedge rst inside always@ to avoid delays when performing reset. As a conclusion, I need to be more careful and organized in typing programs to avoid unwanted errors.

3. 想對老師或助教說的話

Everything was going really well so far. The labs are getting more challenging but the level of difficulty increased gradually, which makes us learn new skills every time which will be used in the next question of the lab.