邏輯設計實驗 Lab3 結報

105060012 張育菘

1. Frequency Divider: Construct a 27-bit synchronous binary counter. Use the MSB of the counter, we can get a frequency divider which provides a $1/2^{27}$ frequency output (fout) of the original clock (fcrystal, 100MHz). Construct a frequency divider of this kind.

1.1 Write the specification of the frequency divider.

1.2 Draw the block diagram of the frequency divider.

1.3 Implement the frequency divider with the following parameters.

clk	clk_out	rst
W5	U16	V17

Design Specification

input : rst, clk;

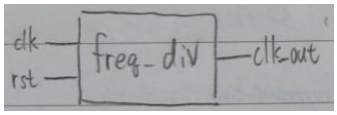
output : clk_out;

reg clk_out;

reg [25:0]cnt;

reg [26:0]cnt_tmp;

block diagram :

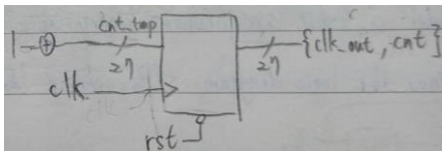


Design Implementation

Logic function :

```
always@* cnt_tmp={clk_out,cnt}+1'b1;
always@(posedge clk or negedge rst)    //DFFs
   if(~rst){clk_out,cnt}<=27'b0;  //當 rst=0 時，{clk_out,cnt} = 0(從 0 開始數)
   else{clk_out,cnt}<=cnt_tmp;
                  //當 rst=1 & clk 從 0→1 時，把 cnt_tmp 存入{clk_out,cnt}
```

Logic diagram :



Result

當 rst=1 時，其閃爍的頻率為 clk 的 $1/2^{27}$ 倍

2.Frequency Divider: Use a count-for-50M counter and some glue logics to construct a 1 Hz clock frequency. Construct a frequency divider of this kind.

2.1 Write the specification of the frequency divider.

2.2 Draw the block diagram of the frequency divider.

2.3 Implement the frequency divider with the following parameters.

clk	clk_out	rst
W5	U16	V17

Design Specification

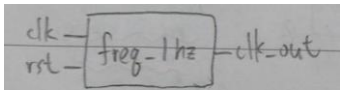
input : rst, clk;

output : clk_out;

reg clk_out;

reg [25:0]cnt;

reg [26:0]cnt_tmp;



Design Implementation

Logic function :

```
always@* cnt_tmp={clk_out,cnt}+1'b1;
always@(posedge clk or negedge rst)
    if(~rst) {clk_out,cnt} <= 27'b0;
    else
    begin
        {clk_out,cnt} <= cnt_tmp;
        if(cnt==49999999)
//當 cnt 數道 49999999 時，clk_out 從 0→1 或是從 1→0，也讓 cnt 重數
            begin
                cnt<=26'b0;
                clk_out<=~clk_out;
```

```
                end
            end
```

Result

當 rst=1 時，其閃爍的頻率為 clk 的 $1/(5*10^7)$倍

Discussion

1. 除頻的方法可以用多個 bit 進行，因為越高位的頻率因加法的特性會變低
2. 在作 DFFs 時需要 next_state 和 state，因此在做的時候拿 cnt_tmp 當作我的 next_state

3.Construct a single digit BCD up counter with the divided clock as the clock frequency and display on the seven-segment display.

3.1 Construct a BCD up counter.

3.2 Construct a BCD-to-seven-segment display decoder.

3.3 Combine the above two together.

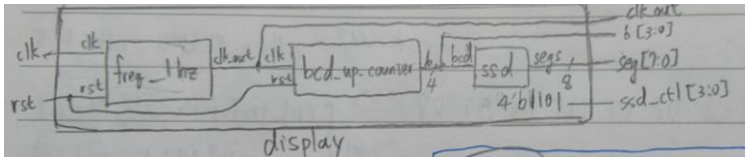
clk	clk_out	b[3]	b[2]	b[1]	b[0]
W5	U16	P3	N3	P1	L1

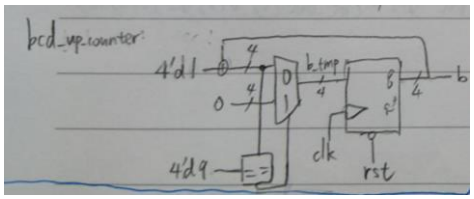
Design Specification

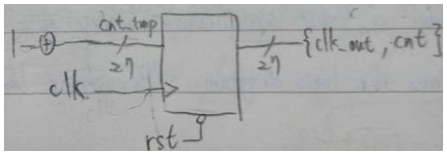
input : clk,rst;

output : clk_out,[3:0]b, [3:0]ssd_ctl, [7:0]seg;

block diagram :







(freq_1hz)

Design Implementation

Logic function :

1. display：

   此為 top module，專門拿來呼叫其他小 moudule 的，並把 freq_1hz 除頻過後的 clk_out 作為 bcd_up_counter 的 clk，讓 bcd_up_counter 在每 1hz 時把數值加一。並只讓顯示器第 3 個數字亮(ssd_ctl = 4'b1101)。

2. freq_1hz：

   其運作原理同第二題，當 cnt 數到 49999999 時，clk=~clk 以及讓 cnt 從 0 開始重數，而 clk_out 即是除頻過後的產物，頻率為 1hz。

3. bcd_up_counter：

   先設一個 b_tmp[3:0]作為 DFFs 的 next_state，而 b[3:0]作為 DFFs 的 state。當 b 數到 9 時，把 b 的值歸零，用此達到從 0 數到 9 後再從 0 開始的效果。而其 reset 的值在此設為 0。

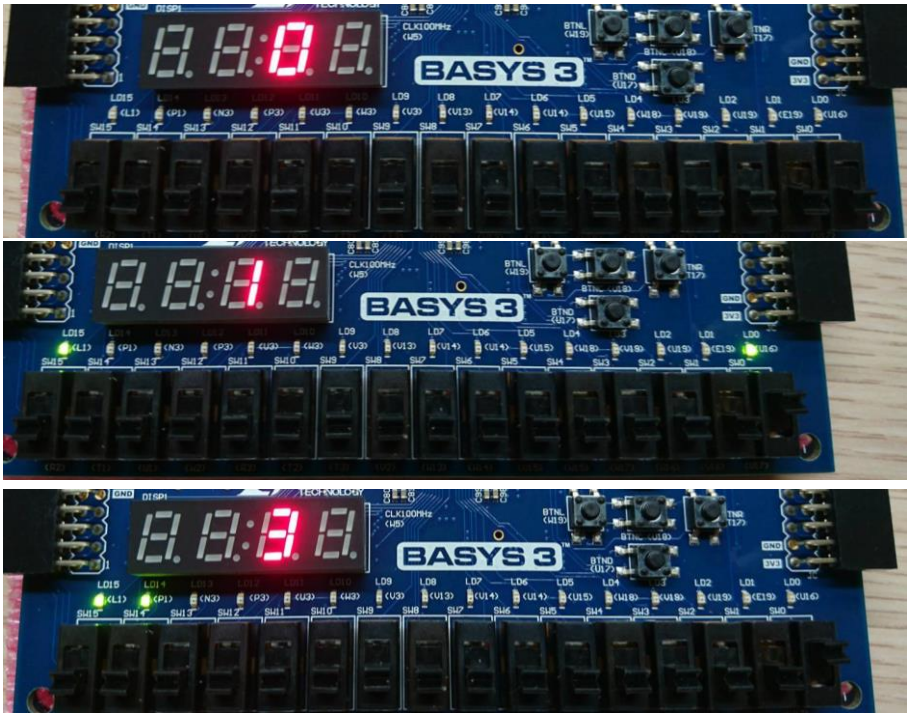
```
always@*
        b_tmp=b+1'b1;
always@(posedge clk or negedge rst)
            if(~rst) b <= 4'b0;
            else
            begin
                b <= b_tmp;
                if(b==9)
                begin
                    b<=4'b0;
                end
            end
```

4. ssd：

```
當 bcd 為 4'd0: segs = 8'b00000011;   //七段顯示器顯示 0
當 bcd 為 4'd1: segs = 8'b10011111;   //七段顯示器顯示 1
當 bcd 為 4'd2: segs = 8'b00100101;   //七段顯示器顯示 2
當 bcd 為 4'd3: segs = 8'b00001101;   //七段顯示器顯示 3
當 bcd 為 4'd4: segs = 8'b10011001;   //七段顯示器顯示 4
當 bcd 為 4'd5: segs = 8'b01001001;   //七段顯示器顯示 5
當 bcd 為 4'd6: segs = 8'b01000001;   //七段顯示器顯示 6
當 bcd 為 4'd7: segs = 8'b00011111;   //七段顯示器顯示 7
當 bcd 為 4'd8: segs = 8'b00000001;   //七段顯示器顯示 8
當 bcd 為 4'd9: segs = 8'b00001001;   //七段顯示器顯示 9
default: segs = 8'b00000000;          //七段顯示器顯示 8.
```

Result(顯示器只有第 3 個數字亮)

1. rst=0→初始化設為 0
2. rst=1→每當 U16 的 LED 燈亮的霎那，七段顯示器顯示的數字開始往上數，顯示 b 的燈也跟著往上跳一個數字



Discussion

1. 在呼叫其他小 module 時，括號內的 I/O 變數為大 module 的 I/O 變數；括號外的為被呼叫的小 module 的 I/O 變數。Ex. ssd U2(.bcd(b),.segs(seg));　”U2”為名稱、”b”為 display 的 I/O 變數、”bcd”為 ssd 的 I/O 變數。
2. 最好用一個 top module 包小 module，這樣在跑實驗時比較不會跑出不是理想中的結果。

4. (Bonus) Construct a 30 seconds count down timer (stop at 00).

clk	clk_out	cnt_ten	cnt_ten	cnt_ten	cnt_ten
W5	U16	cnt_ten	cnt_ten	cnt_ten	cnt_ten

cnt_one[3]	cnt_one[2]	cnt_one[1]	cnt_one[0]
V13	V3	W3	U3

Design Specification

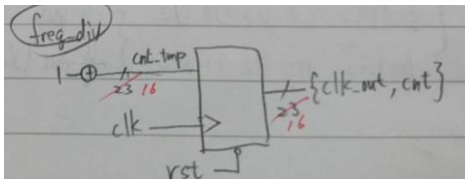
input : clk,rst;

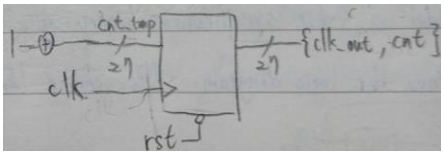
output : clk_out, [3:0]cnt_one,[3:0]cnt_ten, [3:0]ssd_ctl, [7:0]seg;

wire clk_bit;

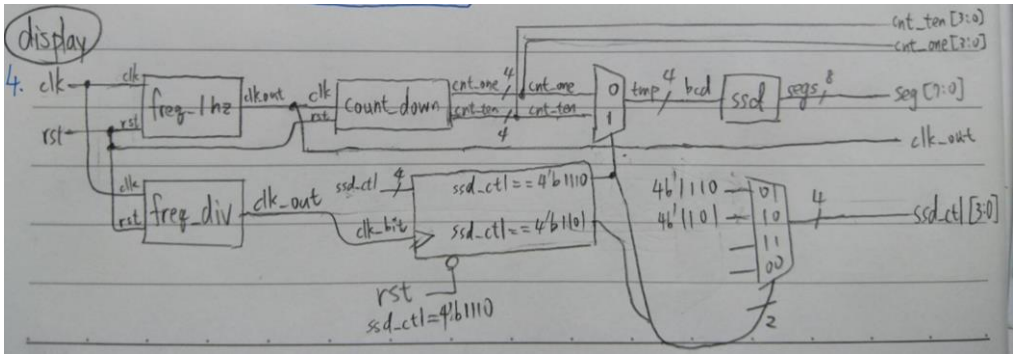
reg [3:0]tmp, [3:0]ssd_ctl;

block diagram :





(freq_1hz)



Design Implementation

Logic function :

1. display：

此為 top module，專門拿來呼叫其他小 moudule 的，並把 freq_1hz 除頻過後的 clk_out 作為 count_down 的 clk，讓 count_down 在每 1hz 時把數值減一。還有把經過 freq_div 除頻過後的 clk_out(週期為內建時間的 2^{16} 倍)作為自己的 clk_bit。

內部還有一個多功器，功能即是當顯示其顯示十位數時讓顯示器的第 2 個數字亮；當顯示其顯示個位數時讓顯示器的第 1 個數字亮。

```
always@(posedge clk_bit or negedge rst)
        begin
            if(ssd_ctl == 4'b1110)
```

//當顯示器第一個數字亮時同時把 cnt_ten 存入 tmp 以及讓顯示器第 1 個數字亮的訊號存入 ssd_ctl，讓下一次 clk_bit 從 0→1 時，顯示此結果。

```
            begin
                tmp <= cnt_ten;
                ssd_ctl <= 4'b1101;
            end
            else if(ssd_ctl == 4'b1101)
            begin
                tmp <= cnt_one;
                ssd_ctl <= 4'b1110;
            end
            else;
```

end

2. freq_1hz：
   其運作原理同第二題，當 cnt 數到 49999999 時，clk=~clk 以及讓 cnt 從 0 開始重數，而 clk_out 即是除頻過後的產物，頻率為 1hz。
3. freq_div：
   其運作原理同第一題，cnt_tmp 為一 16bits 的數，因此其最高為即為 clk_out，即是除頻過後的產物，週期為 clk 的 2^{16} 倍。
4. count_down：
   這是同時數兩個變數，個位數 cnt_one 跟十位數 cnt_ten 進行往下數。當 cnt_one=0 時把 cnt_ten 的數值減一以及 cnt_one 的數值變成 9；當 cnt_one 跟 cnt_ten 都為 0 時，讓兩個數都維持 0。而初始化(rst=0)就是把樹設為 30(cnt_ten=3 & cnt_one=0)。
5. ssd：
   當 bcd 為 4'd0: segs = 8'b00000011; //七段顯示器顯示 0
   當 bcd 為 4'd1: segs = 8'b10011111; //七段顯示器顯示 1
   當 bcd 為 4'd2: segs = 8'b00100101; //七段顯示器顯示 2
   當 bcd 為 4'd3: segs = 8'b00001101; //七段顯示器顯示 3
   當 bcd 為 4'd4: segs = 8'b10011001; //七段顯示器顯示 4
   當 bcd 為 4'd5: segs = 8'b01001001; //七段顯示器顯示 5
   當 bcd 為 4'd6: segs = 8'b01000001; //七段顯示器顯示 6
   當 bcd 為 4'd7: segs = 8'b00011111; //七段顯示器顯示 7
   當 bcd 為 4'd8: segs = 8'b00000001; //七段顯示器顯示 8
   當 bcd 為 4'd9: segs = 8'b00001001; //七段顯示器顯示 9
   default: segs = 8'b00000000; //七段顯示器顯示 8.

Result(顯示器只有第 3 個數字亮)

1. rst=0→初始化設為 30
2. rst=1→每當 U16 的 LED 燈亮的霎那，七段顯示器顯示的數字開始往下數，顯示 cnt_one 以及 cnt_ten 的燈也跟著往下跳一個數字
3. 當數到 00 時，顯示器保持 00。

Discussion

1. 這次實驗用的 clk_bit 的用意為顯示兩個不同的數，因為七段顯示器一次只能顯示一種數次，因此需要一個頻率介於內建時間到 1hz 的 clk_bit 產生同時顯示兩個數的錯覺。
2. clk_bit 不能為內建的 clk，因為此轉換時間太快了，兩個數的殘像讓眼睛無法分辨兩個數，就像兩個數若有似無的疊在一起似的，導致我們在觀察時看到不對的結果。

Conclusion：

這次的 lab 更深入帶我們了解到如何除頻，以及如何同時顯示兩個不同的數，雖然過程中挫折不斷，但是打完真的是成就感滿滿，真的很開心。