106061151 劉安得

Prelab3.

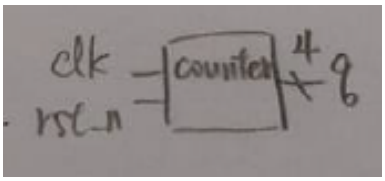
1. **Design Specification**

   4-bits synchronous binary up counter

   Input : clk, rst_n

   Output : q

   Q will count 0 to 15 repeatly

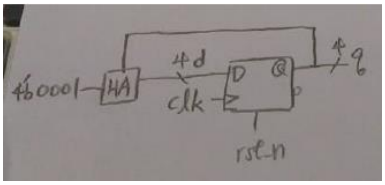


2. **Design Implementation**

   D = q + 1

   if(rst_n = 0), q = 4’b0000

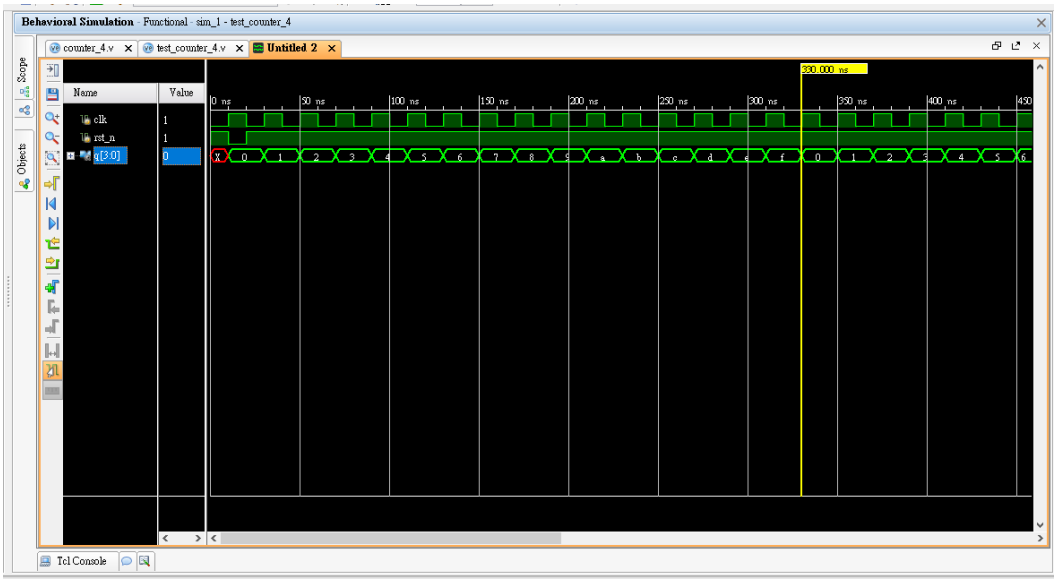
   else, q = d



   D flip-flop synchronous reset

```
always @(posedge clk or negedge rst_n)
        begin
            if(rst_n == 0)
                BCD <= 4'b0000;
            else if(d == 4'b1010)
                BCD <= 4'b0000;
            else
                BCD <= d;
         end
```

3. **simulation**



clk	Rst_n	q
0	1	X
1	0	0
0	1	0
1	1	1
0	1	1
1	1	2
0	1	2
1	1	3
0	1	3
1	1	4
0	1	4
1	1	5
0	1	5
1	1	6
0	1	6
1	1	7
0	1	7
1	1	8
0	1	8
1	1	9
0	1	9
1	1	10
0	1	10
1	1	11

0	1	11
1	1	12
0	1	12
1	1	13
0	1	13
1	1	14
0	1	14
1	1	15
0	1	15
1	1	0

4. Discussion

這次 prelab 只要寫 module 不用燒到 FPGA 板上，因此要寫 textbench 來測試。不過因為 sequential circuit 的 textbench 打法比較不同，因此我有參考之前邏輯設計的參考資料。

然後在打 verilog 前最好先畫一下 block diagram，不然邊想邊打會很容易不知道要打什麼

5. Conclusion

在這之後，我們會開始用到 sequential circuit。這次實驗讓我學會如何 implement counter，還有 D flip-flop verilog 的打法

6. References

Logic Design Lecture 7 sequential textbench

Sequential logic 的 textbench 最大的不同是 clk 需要不斷做 invert，還有 rst_n 記得歸零

```
module test_ex2;
reg clk;
reg rst_n;
reg in;
wire out;

Ex2  U0(.clk(clk), .rst(rst_n), .in(in), .out(out));

always
 #5 clk = ~clk;       // generate periodic clock

initial
begin
 clk=0;rst_n=0;in=0; //set initial input, let FF reset
 #10 rst_n=1; // disable reset
 #10 in=1;
 #10 in=0;
 #10 in=1;
 #10 in=1;
 ...
end

endmodule
```

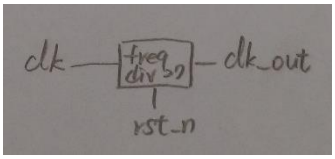
Lab3-1.

1. **Design Specification**

   2^{-27} Frequency Divider

   Input : clk, rst_n

   Output : clk_out



2. **Design Implementation**

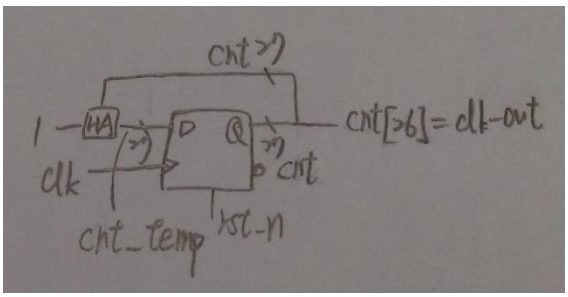
   Construct a 27-bit counter, then the MSB will provide 2^{-27} frequency output

   Cnt_temp = cnt +1

   clk_out = Cnt[26]

   if(rst_n = 0) , cnt = 27 ’d0

   else, cnt = cnt_temp



I/O	Clk_out	clk	Rst_n
Pin	U16	W5	V17

D flip-flop synchronous reset

```
always @(posedge clk or negedge rst_n)
    begin
        if(rst_n == 0)
            cnt <= 27'd0;
        else
            cnt <= cnt_temp;
    end
```

3. **Discussion**

   老師的範例是將[26:0] cnt 分成 {cnt_out, [25:0]cnt}，而我是直接 assign

cnt_out = cnt[27]。

這次 lab，我們剛好利用 27-bit counter 來幫我們 divide frequency，但實際上，應該把 counter 和 frequency divider 分開寫，用一個 top module 來叫出 counter

除頻完的頻率大約是 0.75Hz

一開始我打成[27:0]，但事實上應該是[26:0]，這樣才是 27-bit

4. Conclusion

做完這個實驗讓我學會如何使用 counter 來 implement frequency divider，還有 basic sequential circuit verilog 的打法。

5. References

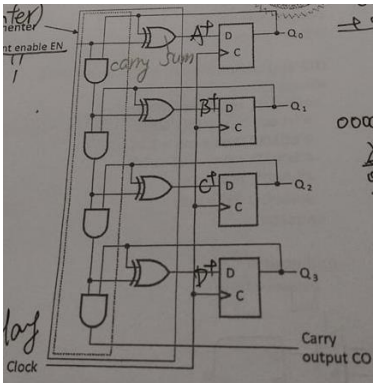
03_Verilog2 P.32

Frequency divider example

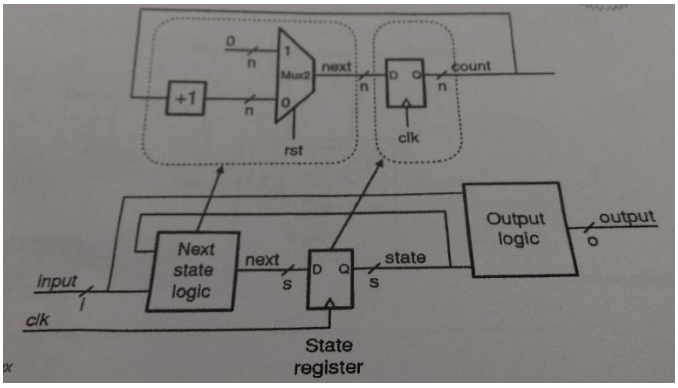
雖然可以直接照抄講義，但我還是有自己理解，用自己的方式打

Logic Design Lecture 8 Registers and Counters

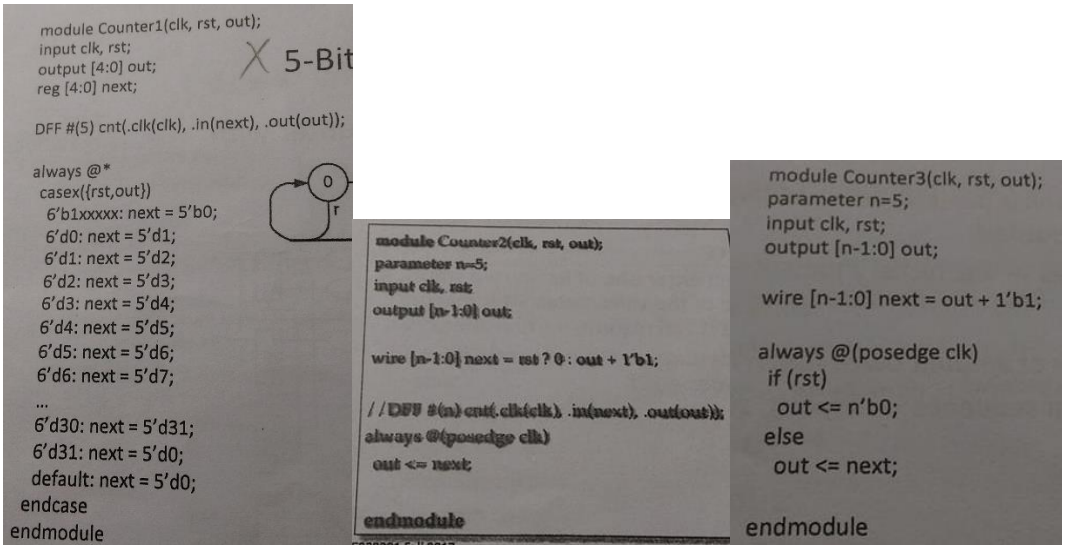
在 verilog 中，簡單的加法其實就代表著一個 adder，我打的 counter 其實是 synchronous counter，就是在 D flip-flop 前加一個 adder。左下圖為 synchronous counter



The schematic of counter



以下是三種 verilog 打法，第一種用 case 寫，比較像是 behavior level，而後兩者為 RTL level，只是兩種不同的 if 語法



```
module Counter1(clk, rst, out);
input clk, rst;
output [4:0] out;
reg [4:0] next;

DFF #(5) cnt(.clk(clk), .in(next), .out(out));

always @*
 casex({rst,out})
  6'b1xxxxx: next = 5'b0;
  6'd0: next = 5'd1;
  6'd1: next = 5'd2;
  6'd2: next = 5'd3;
  6'd3: next = 5'd4;
  6'd4: next = 5'd5;
  6'd5: next = 5'd6;
  6'd6: next = 5'd7;
  ...
  6'd30: next = 5'd31;
  6'd31: next = 5'd0;
  default: next = 5'd0;
 endcase
endmodule
```

5-Bit

```
module Counter2(clk, rst, out);
parameter n=5;
input clk, rst;
output [n-1:0] out;

wire [n-1:0] next = rst ? 0 : out + 1'b1;

//DFF #(n) cnt(.clk(clk), .in(next), .out(out));
always @(posedge clk)
 out <= next;

endmodule
```

```
module Counter3(clk, rst, out);
parameter n=5;
input clk, rst;
output [n-1:0] out;

wire [n-1:0] next = out + 1'b1;

always @(posedge clk)
 if (rst)
  out <= n'b0;
 else
  out <= next;

endmodule
```

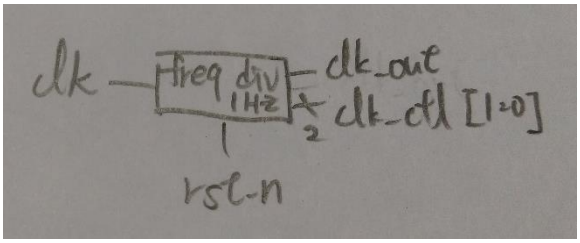
Lab3-2.

1. **Design Specification**

   1Hz Frequency Divider

   Input : clk, rst_n

   Output : clk_out, clk_ctl[1:0]



2. **Design Implementation**

   Construct a 50M counter, when it reach 50M，reset and invert the clock outout. Therefore, the clock will be divided by 100M, and produce a 1Hz clock.

   50M needs 26-bits to store, and the 16-17th bits are the ssd control. The control will be use to implement ssd display later.

   Cnt_temp = cnt +1

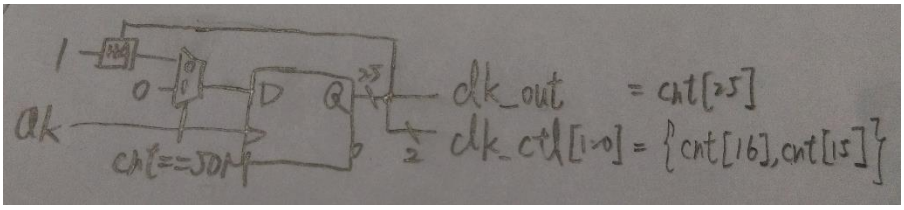
   clk_out = Cnt[25]

   clk_ctl = {cnt[16], cnt[15]};

   if(rst_n = 0) , cnt = 26 ’d0, clk_out = 0

   else, cnt = cnt_temp

   If(cnt = 50M), cnt = 26 ’d0, clk_out = ~clk_out



I/O	Clk_out	clk	Rst_n
Pin	U16	W5	V17

D flip-flop synchronous reset with MUX

```
always @(posedge clk or negedge rst_n)
    begin
        if(rst_n == 0)
        begin
```

```
            cnt <= 26'd0;
            clk_out <= 0;
        end
        else if(cnt == 26'd50000000)
        begin
            cnt <= 26'd0;
            clk_out <= ~ clk_out;
        end
        else
            cnt <= cnt_temp;
    end
```

3. Discussion

這是一個重要的元件，因為很多其他 module 都會需要用秒來當 clock

老師的範例是將 cnt 設成很多部分，來因應兩種 output，而我是直接 assign cnt 對應的位數給 output。

一開始我把 cnt = 50M 和 rst_n = 0 這兩個判斷式寫在一起，我直接在裡面寫 clk_out 做 invert，我忽略了 clk_out 需要歸零，不然會一直是 unknown，所以 cnt = 50M 和 rst_n = 0 需要分開寫。

除此之外，我還會一直忘記 if 如果超過兩行，要打 begin end，這讓我浪費許多時間

4. Conclusion

做完這個實驗讓我學會如何讓 counter 數到特定的數值來 implement frequency divider，還有 basic sequential circuit verilog 的打法

5. References

03_Verilog2 P.32

Frequency divider example

因為範例是直接用 BCD 來 divide frequency，因此只好自己摸索如何算到一半就 reset

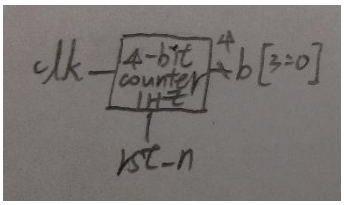
Lab3-3.

1. **Design Specification**

   4-bits counter with 1Hz clock

   Input : clk, rst_n

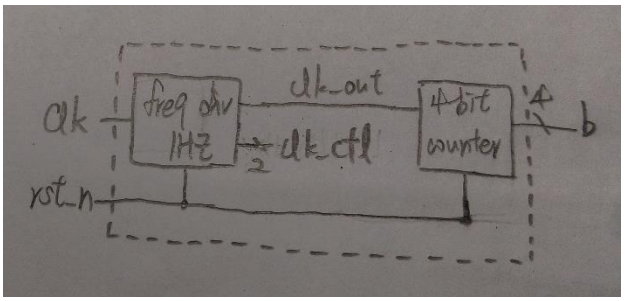
   Output : b[3:0]



2. **Design Implementation**

   使用 prelab 所寫的 4-bits counter 和 lab3-2 的 1Hz frequency divider 來 implement

   Declare a wire “clk_out”, to connect the 1Hz frequency divider and 4-bits counter.



I/O	B[3]	B[2]	B[1]	B[0]	clk	Rst_n
Pin	V19	U19	E19	U16	W5	V17

3. **Discussion**

   因為 bitstream 實在跑太久，所以我都會開下一子實驗先打，但這也造成到時候 bit 檔可能會燒成上個實驗的，讓結果差很多，我也浪費很多時間在這

   也因為只要寫 top module 叫出其他 module，所以不容易出錯

4. **Conclusion**

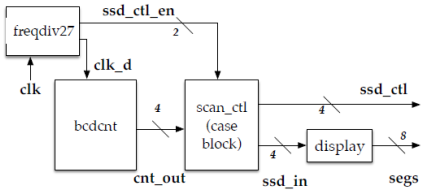
   這次實驗是我第一次使用 top module 來 implement，也讓我體會到使用寫好的 module 的便利性

5. **References**

   03_Verilog2 P.35

The schematic of top module

雖然範例是 BCD_counter，但它幫助我理解 top module 是如何運作的



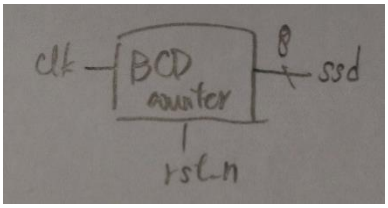
Lab3-4.

1. **Design Specification**

   BCD counter

   Input : clk, rst_n

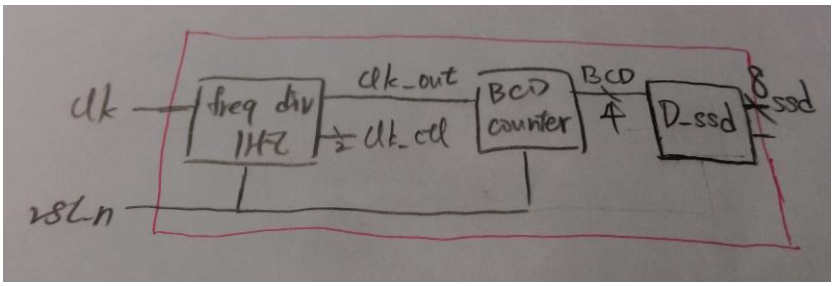
   Output : ssd[7:0]



2. **Design Implementation**

   使用 lab2 所寫的 ssd display，lab3-2 的 1Hz frequency divider，並再寫一個 BCD counter 來 implement

   前面兩者用 wire ”clk_out”連結，後面兩者用 wire ”[3:0] BCD”連結



I/O	Ssd[7]	Ssd[6]	Ssd[5]	Ssd[4]	Ssd[3]	Ssd[2]	Ssd[1]	Ssd[0]
Pin	W7	W6	U8	V8	U5	V5	U7	V7

I/O	clk	Rst_n
Pin	W5	V17

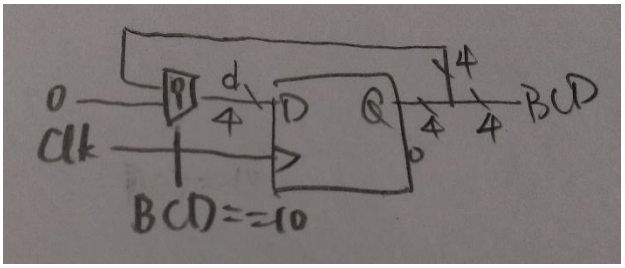
BCD counter

d = BCD +1

if(rst_n = 0) , BCD = 0

else, BCD = d

If(BCD = 10), BCD = 0

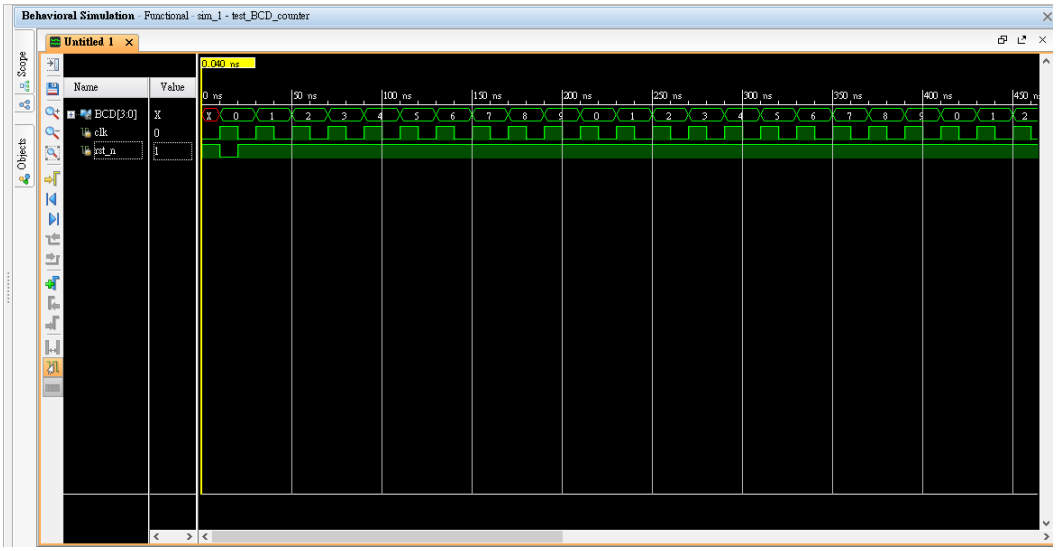


D flip-flop synchronous reset with MUX

```
always @(posedge clk or negedge rst_n)
        begin
            if(rst_n == 0)
                BCD <= 4'b0000;
            else if(d == 4'b1010)
                BCD <= 4'b0000;
            else
                BCD <= d;
        End
```

3. Simulation

BCD counter



clk	Rst_N	BCD
0	1	X
1	0	0000(0)
0	1	0000(0)
1	1	0001(1)
0	1	0001(1)

1	1	0010(2)
0	1	0010(2)
1	1	0011(3)
0	1	0011(3)
1	1	0100(4)
0	1	0100(4)
1	1	0101(5)
0	1	0101(5)
1	1	0110(6)
0	1	0110(6)
1	1	0111(7)
0	1	0111(7)
1	1	1000(8)
0	1	1000(8)
1	1	1001(9)
0	1	1001(9)
1	1	0000(0)

4. Discussion

我發現我們之前沒有打過 BCD_counter，只好自己打一個，並再用 top module 把需要的 module 包起來

因為這次只有 1-9 所以我讓七段顯示器，同時顯示同個數字，但如果是 bonus 的話就不能這樣打

由於是第四個子實驗了，而且跟前面很像，漸入佳境，這次沒什麼出錯就做好了

我在事後跑 simutation 時，我發現不能直接在本來的 project 跑，要另外開一個新的 project

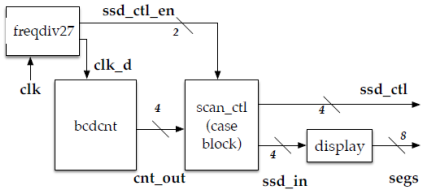
5. Conclusion

這次實驗算是總結之前所學，除頻器，counter，七段顯示器，top module，全部合在一起應用

6. References

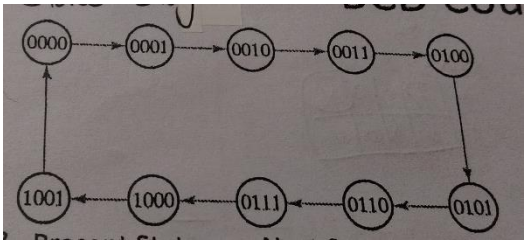
03_Verilog2 P.35

The schematic of BCD counter



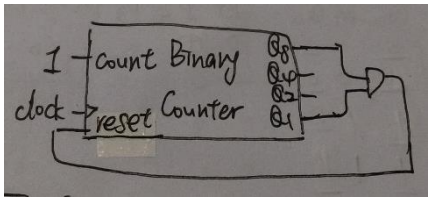
Logic Design Lecture 8 Registers and Counters

The state diagram of BCD counter



The block diagram of BCD counter

它是用 combinational circuit，將 output 接到 reset。當 1010 時，reset 就會被啟動而我則是使用 MUX，當 1010 時，D flip-flop 的 D = 0



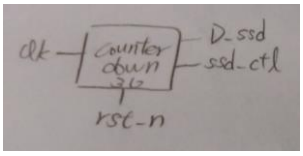
Lab3-5.

1. **Design Specification**

   30 到 00 的倒數計時器

   Input : clk, rst_n

   Output : [7:0] D_ssd, [3:0] D_ssd

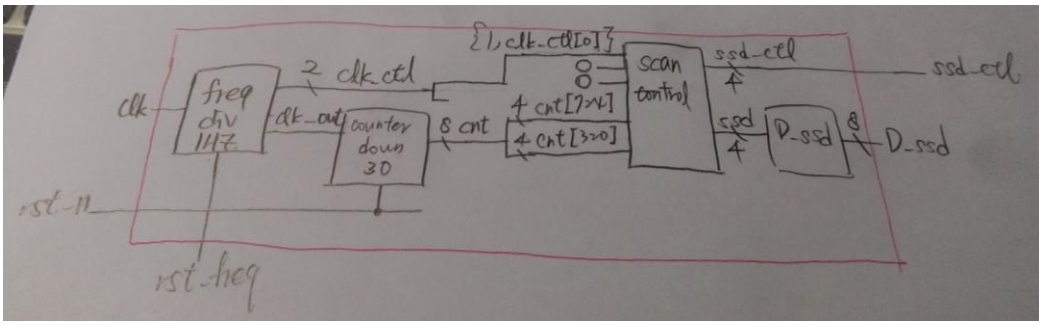


2. **Design Implementation**

   使用 lab2 所寫的 ssd display，lab3-2 的 1Hz frequency divider，並再寫 scan control 和從 30 開始倒數的 down counter 來 implement

   Scan control 的目的是利用視覺暫留來讓七段顯示器同時顯示不同的數字，因為七段顯示器只能同時顯示一個數字，只好用視覺暫留的方式

   各 module 中間的 wire 見下圖，比較值得注意的是，rst_freq 是設成 reg，因為必須在一開始就重設 frequency divider，否則 scan crontrol 的 clk_ctl 只會收到 10，變成 reset 的時候只會顯示 3，不會顯示 30，所以要讓 frequency divider 一開始就要動。



I/O	D_ssd [7]	D_ssd [6]	D_ssd [5]	D_ssd [4]	D_ssd [3]	D_ssd [2]
Pin	W7	W6	U8	V8	U5	V5

I/O	D_ssd [1]	D_ssd [0]	Ssd_ctl[3]	Ssd_ctl[2]	Ssd_ctl[1]	Ssd_ctl[1]
Pin	U7	V7	W4	V4	U4	U2

I/O	clk	Rst_n
Pin	W5	V17

scan control

透過 case，收到不同的 clk_ctl 亮不同的七段顯示器，舉 00 為例，則將 ssd_ctl 設為 0111，並將 ssd 設為對應的 in0

```
always @*
    case(clk_ctl)
    2'b00:
    begin
        ssd_ctl = 4'b0111;
        ssd = in0;
    end
    2'b01:
    begin
        ssd_ctl = 4'b1011;
        ssd = in1;
    end
    2'b10:
    begin
        ssd_ctl = 4'b1101;
        ssd = in2;
    end
    2'b11:
    begin
        ssd_ctl = 4'b1110;
        ssd = in3;
    end
    default:
    begin
        ssd_ctl = 4'b0000;
        ssd = in0;
   end
```

從 30 開始倒數的 **down counter**

直接使用 2-bits BCD 來記數，這樣就不用再將 binary 轉成 BCD 了

當個位數為 0 時，要從十位數借位。最後數到剩 00，要停在 00 不動

cnt_temp = cnt - 1;

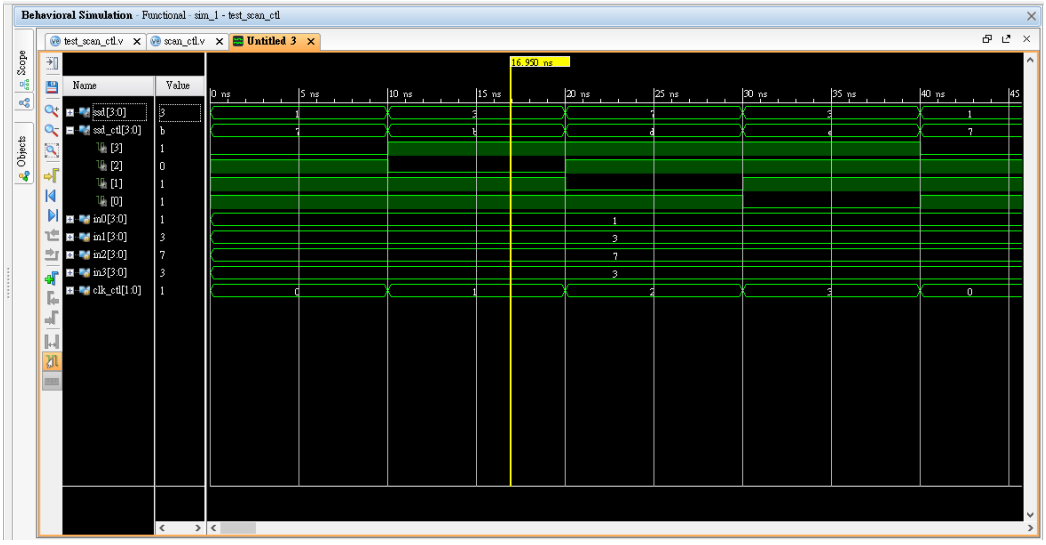
if(rst_n = 0), cnt = 00110000(30)

else if(cnt = 00000000(00)), cnt = 00000000(00)
else if(cnt = xxxx0000(x0)), cnt[7:4] = cnt[7:4] -1, cnt[3:0] = 1001(9)
else, cnt = cnt_temp

```
if(rst_n == 0)
                cnt <= 8'b00110000;
        else if(cnt == 8'd0)
                cnt <= 8'd0;
        else if(cnt[3:0] == 4'b0000)
        begin
                cnt[7:4] <= cnt[7:4] -1;
                cnt[3:0] <= 4'b1001;
        end
        else
             cnt <= cnt_temp;
```

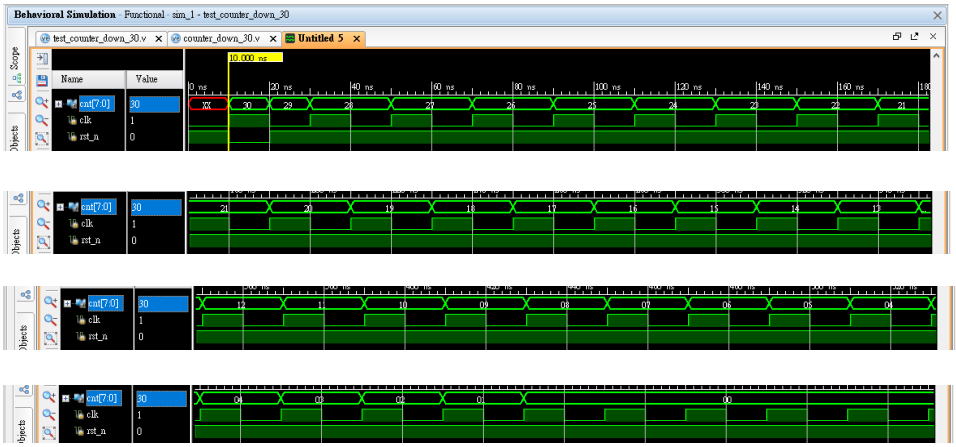
3. Simulation

Scan control



Ssd[3:0]	Ssd_ctl[3:0]	In0[3:0]	In1[3:0]	In2[3:0]	In3[3:0]	Clk_ctl[3:0]
0001(1)	0111	0001(1)	0011(3)	0111(7)	0011(3)	00
0011(3)	1011	0001(1)	0011(3)	0111(7)	0011(3)	01
0111(7)	1101	0001(1)	0011(3)	0111(7)	0011(3)	10
0011(3)	1110	0001(1)	0011(3)	0111(7)	0011(3)	11
0001(1)	0111	0001(1)	0011(3)	0111(7)	0011(3)	00

down counter



Cnt 從 30 倒數到 00，然後一直維持在 00

4. Discussion

這絕對是 lab3 中最難的，它難的地方在於要如何讓七段顯示器顯示不同的數字。

然後我發現 frequency divider 需要一開始就被啟動，不然 clk_ctl 會一直等於 0，這樣就只會顯示一個數字 3，而不是初始應該要顯示的值 30。我在 initial 裡面觸發 rst_n 來解決這個 bug。其實以後 frequency divider 可以被內建啟動好，反正它的目的是提供 clk。

最後，我打 textbench 時，4’b0011 打成 0011，結果好像就真的出錯，原來不打 4’b 真的會比較容易出問題，我以後要記得打

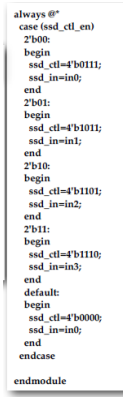
5. Conclusion

做完這次實驗，讓我學到如何讓七段顯示器顯示不同的數字，在 initial 裡面賦值，以及 down counter 的作法。做出來後很有成就感。

6. References

03_Verilog2 P.37

Scan control verilog



```
always @*
 case (ssd_ctl_en)
  2'b00:
  begin
   ssd_ctl=4'b0111;
   ssd_in=in0;
  end
  2'b01:
  begin
   ssd_ctl=4'b1011;
   ssd_in=in1;
  end
  2'b10:
  begin
   ssd_ctl=4'b1101;
   ssd_in=in2;
  end
  2'b11:
  begin
   ssd_ctl=4'b1110;
   ssd_in=in3;
  end
  default:
  begin
   ssd_ctl=4'b0000;
   ssd_in=in0;
  end
 endcase

endmodule
```

Logic Design Lecture 8 Registers and Counters

The schematic of UDL counter

如果想要之後 implement 更多功能的話，可以寫一個 UDL counter，這樣不管是 up, down, load 都可以 implement