Pre-lab1 與 Experiment1:

Pre-lab1:
Construct a 30-second down counter with pause function. When the counter goes to 0, all the LEDs will be lighted up. You can use one push button for reset and one other for pause/start function.
1.1 Write the spec (inputs, outputs, and function table) of the design.
1.2 Draw the related block/logic diagram.
1.3 Use a FSM to implement the function of pause/start function. Use one LED to represent current state.
1.4 Use Verilog to implement 1.3 and verify the design with simulation results.

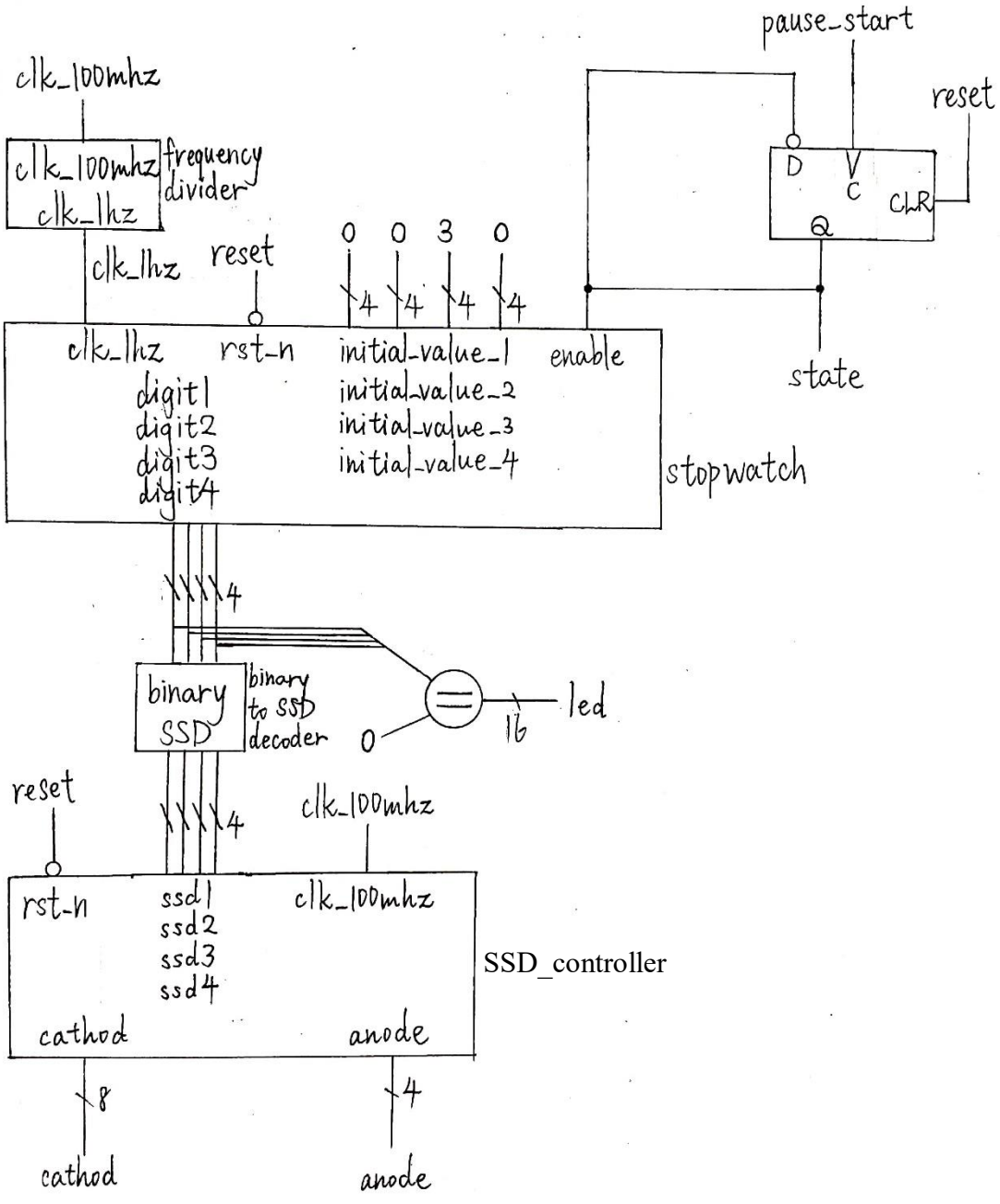
Experiment1:
Construct a 30-second down counter with pause function. When the counter goes to 0, all the LEDs will be lighted up. You can use one push button for reset and one other for pause/start function.
1.1 Implement a periodic 30-second down counter and demo with the FPGA board.
1.2 Implement Prelab 1.3 and demo with the FPGA board.
1.3 Combine 1.2 and 1.3 to finish the experiment.

a. Specification :
Inputs:
- pause_start // 按鈕，按一下開始下數(start)，再按一下會暫停(pause)
- reset // 按鈕，按一下會重置計數器
- clk_100mhz // FPGA 內建的時脈，頻率為 100 M Hz

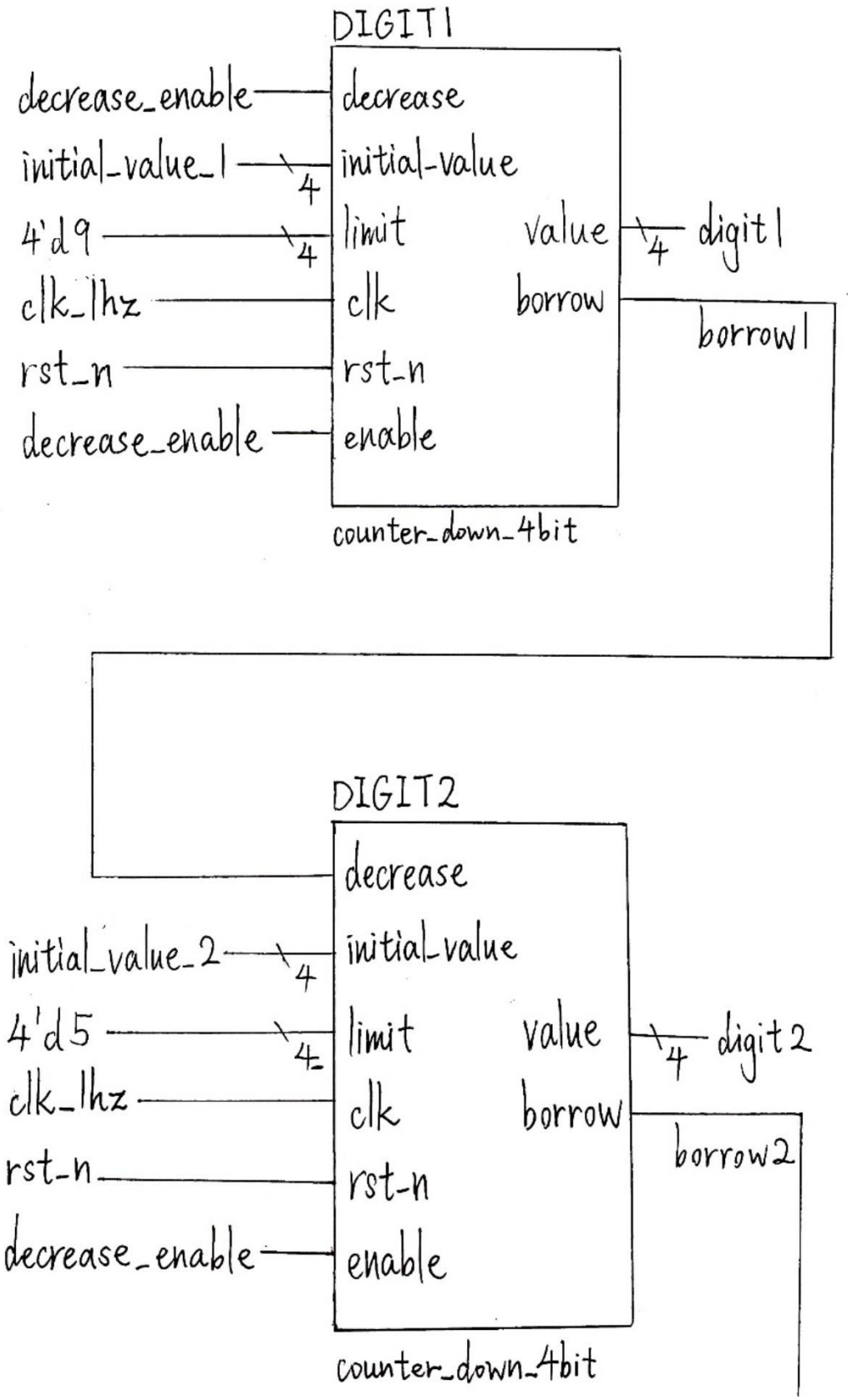
Outputs:
- cathode[7:0] // 控制一個七段顯示器中各個 segment 的亮暗
- anode[3:0] // 控制要啟用哪一個七段顯示器
- led[14:0] // 計數器數到零的時候，亮燈
- state // 顯示計數器當下的狀態是 pause 還是 start

b. Block Diagram :

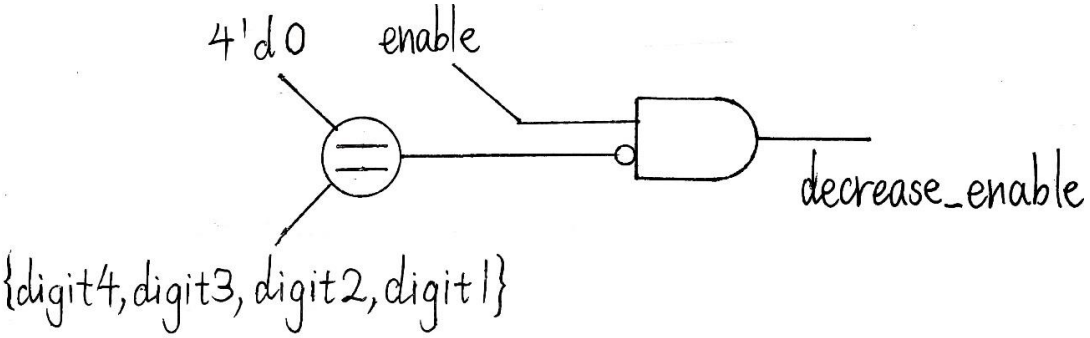
用除頻器製造出一赫茲的時脈。把一赫茲時脈連接到碼表，使碼表的數值每秒減少一。把碼表的二進位制數值以解碼器轉換成七段顯示器的樣式，然後再連接到七段顯示器的控制器。下圖右上角，用一個 T Flip-flop 來實作有限狀態機，用來控制碼表的狀態，判斷當下碼表應該要下數(start)還是不動(pause)。



下圖是碼表內部的局部接線。碼表內有四個四位元的計數器，每個四位元計數器負責一個位數。第四、三位數(digit4, digit3)是分鐘的 BCD，第二、一位數(digit2, digit1)是秒的 BCD。



為了使碼表在數到零的時候自動停下來，碼表內的每一個四位元計數器的 enable 不可以與碼表外部的 enable 直接連接，而是應該要多判斷一下。如下圖，必須要確定這四個四位元計數器所代表的倒數時間不是零，才繼續下數。



c. FSM :

計數器有兩個狀態，S1 是正在下數，S2 是暫停。可以用一個位元來表示計數器當下的狀態 state，當按鈕 pause_start 被按下時，會切換計數器的狀態：

state	pause_start	state(next)
0	0	0
1	0	1
0	1	1
1	1	0

pause_start	state(next)
0	state
1	~state

d. Implement :

除頻器的作法跟上次的 Lab 一樣。使用一個計數器，每當數到 50M 的時候，就觸發 T Flip-flop。如下圖，這樣就成功地製造出 50% duty 的一赫茲時脈了。

```
1  module frequency_divider_100M(      // 100M Hz to 1 Hz 50% duty frequency divider
2      output reg clk_1hz,             // output frequency
3      input      clk_100mhz,          // crystal frequency
4      input      rst_n                // active low reset
5      );
6      reg  [27:0] counter;
7
8      always @ (posedge clk_100mhz or negedge rst_n)
9          if (~rst_n)
10             counter <= 0;
11         else
12             counter <=
13                 (counter == 50_000_000) ? (1) : (counter + 1);
14
15     always @ (posedge clk_100mhz or negedge rst_n)
16         if (~rst_n)
17             clk_1hz <= 0;
18         else if (counter == 1)
19             clk_1hz <= ~clk_1hz;
20
21 endmodule                           // frequency_divider_100M
```

四位元下數計數器的硬體描述語言如下圖，跟上一次 LAB 的一模一樣。第 14 到 22 行判斷下一個狀態的數值：如果要重置，就把數值設定成初始值；否則，如果不要下數，就把保持相同的數值；否則，如果還沒有數到零，就把數值減少一；否則，把數值設定成最大的允許的數值。第 25 到 29 行的 D Flip-flop 用來儲存和更新數值。第 32 到 33 行用來判斷是否有需要向上一位數借位：如果要繼續下數，但是當下的數值已經數到零了，就需要輸出借位的訊號給上一位數的四位元計數器。

```
1  module counter_down_4bit(              // a 4-bit binary counter
2      output reg [3:0] value,            // the binary value of this counter
3      output           borrow,           // this digit is zero and required to continue to count down
4      input            decrease,         // whether to enable the counting function
5      input      [3:0] initial_value,    // the value to be assigned to this counter when rst_n
6      input      [3:0] limit,            // the highest allowed value
7      input            clk,              // the frequency of decreasing by 1
8      input            rst_n,            // the active low reset
9      input            enable            // whether to enable counting and borrowing function
10 );
11     reg [3:0] value_temp;
12
13     // combinational logic for next value
14     always @ *
15         if (~rst_n)
16             value_temp <= initial_value;
17         else if (~decrease)
18             value_temp <= value;
19         else if (value)
20             value_temp <= value - 1;
21         else
22             value_temp <= limit;
23
24     // sequential logic for value
25     always @ (posedge clk or negedge rst_n)
26         if (~rst_n)
27             value <= initial_value;
28         else
29             value <= value_temp;
30
31     // combinational logic for subtraaction carry
32     assign borrow =
33         enable && decrease && value == 0;
34
35 endmodule                              // counter_down_4bit
```

下圖是碼表的輸出、輸入與一些內部的線路名稱。

```
1  module stopwatch(                    // a down-counting stopwatch, able to handle 59:59 – 00:00
2      output [3:0] digit1,             // the binary value of 1st digit
3      output [3:0] digit2,             // the binary value of 2nd digit
4      output [3:0] digit3,             // the binary value of 3rd digit
5      output [3:0] digit4,             // the binary value of 4th digit
6      input  [3:0] initial_value_1,    // the value to be assigned to 1st digit when rst_n
7      input  [3:0] initial_value_2,    // the value to be assigned to 2nd digit when rst_n
8      input  [3:0] initial_value_3,    // the value to be assigned to 3rd digit when rst_n
9      input  [3:0] initial_value_4,    // the value to be assigned to 4th digit when rst_n
10     input        clk_1hz,            // 1 Hz clock
11     input        rst_n,              // active low reset
12     input        enable              // whether to enable the counting process of this counter
13 );
14     wire decrease_enable;            // to determine whether to enable the counting and borrowing function
15     wire borrow1;                    // between digit 1 & 2
16     wire borrow2;                    // between digit 2 & 3
17     wire borrow3;                    // between digit 3 & 4
```

第 20 到 21 行用來判斷是否要讓碼表內的那四個四位元計數器繼續下數：如果碼表外部沒有 enable 訊號，或是這四個計數器所代表的數值已經數到零，就不要讓計數器繼續下數(decrease_enable)。

```
19     // combinational logic: to determine whether to enable the counting and borrowing function
20     assign decrease_enable =
21         {digit4, digit3, digit2, digit1} != 0 && enable;
```

下圖是碼表內那四個四位元計數器的線路連接方法。這四個計數器的 enable 不是與碼表外部的 enable 直接連接，而是與判斷過當下數值是否已經下數到零的 decrease_enable 連接。

```
23      counter_down_4bit DIGIT1(               // a 4-bit binary down counter
24          .value        (digit1),             // the binary value of this counter
25          .borrow       (borrow1),            // whether this digit is zero and required to continue to count down
26          .decrease     (decrease_enable),    // whether to enable the counting function
27          .initial_value(initial_value_1),    // the value to be assigned to this counter when rst_n
28          .limit        (9),                  // the highest allowed value
29          .clk          (clk_1hz),            // the frequency of decreasing by 1
30          .rst_n        (rst_n),              // active low reset
31          .enable       (decrease_enable)     // whether to enable counting and borrowing function
32          );
33
34      counter_down_4bit DIGIT2(               // a 4-bit binary down counter
35          .value        (digit2),             // the binary value of this counter
36          .borrow       (borrow2),            // whether this digit is zero and required to continue to count down
37          .decrease     (borrow1),            // whether to enable the counting function
38          .initial_value(initial_value_2),    // the value to be assigned to this counter when rst_n
39          .limit        (5),                  // the highest allowed value
40          .clk          (clk_1hz),            // the frequency of decreasing by 1
41          .rst_n        (rst_n),              // active low reset
42          .enable       (decrease_enable)     // whether to enable counting and borrowing function
43          );
44
45      counter_down_4bit DIGIT3(               // a 4-bit binary down counter
46          .value        (digit3),             // the binary value of this counter
47          .borrow       (borrow3),            // whether this digit is zero and required to continue to count down
48          .decrease     (borrow2),            // whether to enable the counting function
49          .initial_value(initial_value_3),    // the value to be assigned to this counter when rst_n
50          .limit        (9),                  // the highest allowed value
51          .clk          (clk_1hz),            // the frequency of decreasing by 1
52          .rst_n        (rst_n),              // active low reset
53          .enable       (decrease_enable)     // whether to enable counting and borrowing function
54          );
55
56      counter_down_4bit DIGIT4(               // a 4-bit binary down counter
57          .value        (digit4),             // the binary value of this counter
58          .borrow       (),                   // whether this digit is zero and required to continue to count down
59          .decrease     (borrow3),            // whether to enable the counting function
60          .initial_value(initial_value_4),    // the value to be assigned to this counter when rst_n
61          .limit        (5),                  // the highest allowed value
62          .clk          (clk_1hz),            // the frequency of decreasing by 1
63          .rst_n        (rst_n),              // active low reset
64          .enable       (decrease_enable)     // whether to enable counting and borrowing function
65          );
66
67  endmodule                                   // stopwatch
```

解碼器的硬體描述語言如下圖，跟上一次 LAB 的一模一樣。

```
module decoder_binary_to_SSD(   // 4-bit binary to SSD pattern decoder
    output reg [7:0] SSD,       // 7-segment display decode
    input      [3:0] binary     // 4-bit binary number
);

    always @ *                  // binary (0-9, a, b, c, d, e, f) to 7-segment display decoder
        case (binary)
            0:       SSD = 8'b0000001_1;  // 0
            1:       SSD = 8'b1001111_1;  // 1
            2:       SSD = 8'b0010010_1;  // 2
            3:       SSD = 8'b0000110_1;  // 3
            4:       SSD = 8'b1001100_1;  // 4
            5:       SSD = 8'b0100100_1;  // 5
            6:       SSD = 8'b0100000_1;  // 6
            7:       SSD = 8'b0001111_1;  // 7
            8:       SSD = 8'b0000000_1;  // 8
            9:       SSD = 8'b0000100_1;  // 9
            10:      SSD = 8'b0001000_1;  // a
            11:      SSD = 8'b1100000_1;  // b
            12:      SSD = 8'b0110001_1;  // c
            13:      SSD = 8'b1000010_1;  // d
            14:      SSD = 8'b0110000_1;  // e
            15:      SSD = 8'b0111000_1;  // f
            default: SSD = 8'b0000000_0;  // all on
        endcase

endmodule                       // decoder_binary_to_SSD
```

七段顯示器控制器的硬體描述語言如下，跟上一次 LAB 的一模一樣。第 14 到 18 行是除頻器，用來產生適當的頻率。雖然這四個七段顯示器是輪流顯示，並非同時，但是因為人眼的視覺暫留，看起來就像同時顯示了四個不同的數字。第 21 到 28 行用來控制要讓四個顯示器中的哪一個顯示器顯示數字，而第 31 到 38 行則控制要顯示的樣式。

```
1  module SSD_controller(                          // control common-anode and individual-cathode
2      output reg [7:0] cathode,                   // cathodes of similar segments on all four displays
3      output reg [3:0] anode,                     // anodes of the seven LEDs forming each digit
4      input      [7:0] digit1,                    // SSD pattern to display
5      input      [7:0] digit2,                    // SSD pattern to display
6      input      [7:0] digit3,                    // SSD pattern to display
7      input      [7:0] digit4,                    // SSD pattern to display
8      input            clk_100mhz,               // 100M Hz global clock
9      input            rst_n                     // active low reset
10 );
11     reg [20:0] q_counter;                       // frequency divider
12
13     // frequency divider
14     always @ (posedge clk_100mhz or negedge rst_n)
15         if (~rst_n)
16             q_counter <= 0;                     // active low reset
17         else
18             q_counter <= q_counter + 1;         // add 1
19
20     // anode enable
21     always @ *                                  // take a suitable refresh frequency
22         case (q_counter[20:19])
23             2'b00:   anode <= 4'b1110;          // 1st digit
24             2'b01:   anode <= 4'b1101;          // 2nd digit
25             2'b10:   anode <= 4'b1011;          // 3rd digit
26             2'b11:   anode <= 4'b0111;          // 4th digit
27             default: anode <= 4'b0000;          // all digit
28         endcase
29
30     // cathode selection
31     always @ *                                  // take a suitable refresh frequency
32         case (q_counter[20:19])
33             2'b00:   cathode <= digit1;  // 1st digit
34             2'b01:   cathode <= digit2;  // 2nd digit
35             2'b10:   cathode <= digit3;  // 3rd digit
36             2'b11:   cathode <= digit4;  // 4th digit
37             default: cathode <= 8'b0000000_0;     // all on
38         endcase
39
40 endmodule                                       // SSD_controller
```

構造出最上層的有限狀態機並規劃好接線之後，用以上五個模組就足以用來完成這個 LAB 的所有題目。接下來的幾張圖片是依照 Pre-lab1 的要求所構造出來的硬體描述語言。輸出、輸入與一些內部接線的名稱如下：

```
1  module LAB5_pre1(                  // TOP
2      output [ 7:0] cathode,         // cathodes of similar segments on all four displays
3      output [ 3:0] anode,           // anodes of the seven LEDs forming each digit
4      output [14:0] led,             // When the counter goes to 0, all the LEDs will be lighted up
5      output        state,           // (1.3) Use one LED to represent current state
6      input         pause_start,     // a button to pause or resume the counting process
7      input         reset,           // a button to stop the counting process and reset the value to 30 sec.
8      input         clk_100mhz       // a 100 MHz global clock
9  );
10     wire [3:0] bcd1;               // between the stopwatch and the decoder
11     wire [3:0] bcd2;               // between the stopwatch and the decoder
12     wire [3:0] bcd3;               // between the stopwatch and the decoder
13     wire [3:0] bcd4;               // between the stopwatch and the decoder
14     wire [7:0] ssd1;               // between the decoder and the controller
15     wire [7:0] ssd2;               // between the decoder and the controller
16     wire [7:0] ssd3;               // between the decoder and the controller
17     wire [7:0] ssd4;               // between the decoder and the controller
18     wire       clk_1hz;            // 1Hz 50% duty clock produced by the frequency divider
19     reg        enable;             // whether to count or pause
```

第 21 到 39 行，把除頻器所製造出來一赫茲時脈的連接到碼表。

```
21     frequency_divider_100M FD(    // 100M Hz to 1 Hz 50% duty frequency divider
22         .clk_1hz   (clk_1hz),     // output frequency
23         .clk_100mhz(clk_100mhz),  // crystal frequency
24         .rst_n     (~reset)       // active low reset
25     );
26
27     stopwatch SW(                      // a down-counting stopwatch, able to handle 59:59 – 00:00
28         .digit1         (bcd1),        // the binary value of 1st digit
29         .digit2         (bcd2),        // the binary value of 2nd digit
30         .digit3         (bcd3),        // the binary value of 3rd digit
31         .digit4         (bcd4),        // the binary value of 4th digit
32         .initial_value_1(0),           // the value to be assigned to 1st digit when rst_n
33         .initial_value_2(3),           // the value to be assigned to 2nd digit when rst_n
34         .initial_value_3(0),           // the value to be assigned to 3rd digit when rst_n
35         .initial_value_4(0),           // the value to be assigned to 4th digit when rst_n
36         .clk_1hz        (clk_1hz),     // 1 Hz clock
37         .rst_n          (~reset),      // active low reset
38         .enable         (enable)       // whether to enable the counting process of this counter
39     );
```

第 41 到 70 行，碼表的數值經過解碼器處理後，連接到七段顯示器控制器。

```
41      decoder_binary_to_SSD DECODER1(   // 4-bit binary to SSD pattern decoder
42          .SSD    (ssd1),               // 7-segment display decode
43          .binary(bcd1)                 // 4-bit binary number
44      );
45
46      decoder_binary_to_SSD DECODER2(   // 4-bit binary to SSD pattern decoder
47          .SSD    (ssd2),               // 7-segment display decode
48          .binary(bcd2)                 // 4-bit binary number
49      );
50
51      decoder_binary_to_SSD DECODER3(   // 4-bit binary to SSD pattern decoder
52          .SSD    (ssd3),               // 7-segment display decode
53          .binary(bcd3)                 // 4-bit binary number
54      );
55
56      decoder_binary_to_SSD DECODER4(   // 4-bit binary to SSD pattern decoder
57          .SSD    (ssd4),               // 7-segment display decode
58          .binary(bcd4)                 // 4-bit binary number
59      );
60
61      SSD_controller SSD_CTRL(          // control common-anode and individual-cathode
62          .cathode   (cathode),         // cathodes of similar segments on all four displays
63          .anode     (anode),           // anodes of the seven LEDs forming each digit
64          .digit1    (ssd1),            // SSD pattern to display
65          .digit2    (ssd2),            // SSD pattern to display
66          .digit3    (ssd3),            // SSD pattern to display
67          .digit4    (ssd4),            // SSD pattern to display
68          .clk_100mhz(clk_100mhz),      // 100M Hz global clock
69          .rst_n     (~reset)           // active low reset
70      );
```

到目前為止，我們已經構造出能開始下數與暫停的 30 秒倒數馬錶了。應 Pre-lab1 的題目要求，第 72 到 78 行用來控制 LED 燈的明滅：當碼表數到零時，所有 LED 燈亮起；並使用一個 LED 燈來顯示當下碼表的狀態是正在下數還是暫停。

```
72      // combinational logic: to determine whether to light up all the LEDs
73      assign led = {16{
74          {bcd4, bcd3, bcd2, bcd1} == 0
75      }};
76
77      // (1.3) Use one LED to represent current state
78      assign state = enable;
```

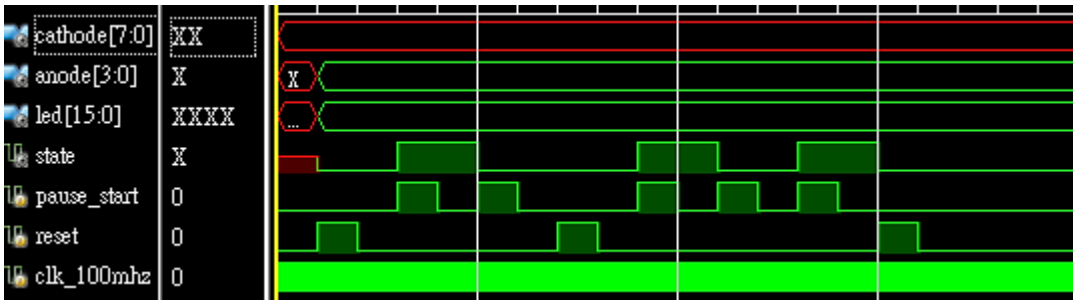
第 80 到 85 行是一個有限狀態機，用來控制碼表應該要下數還是暫停。

```
80      // T flip-flop: changes state whenever the pause_start button is pressed
81      always @ (posedge pause_start or posedge reset)
82          if (reset)
83              enable <= 0;
84          else
85              enable <= ~enable;
86
87  endmodule                       //LAB5_pre1
```

下圖是用來測試 Pre-lab1 的 testbench。

```
1   `timescale 1ps / 1ps
2
3   module LAB5_pre1_test();
4       wire [ 7:0] cathode;
5       wire [ 3:0] anode;
6       wire [15:0] led;
7       wire        state;
8       reg         pause_start;
9       reg         reset;
10      reg         clk_100mhz;
11
12      LAB5_pre1 DUT(
13          .cathode(cathode),
14          .anode(anode),
15          .led(led),
16          .state(state),
17          .pause_start(pause_start),
18          .reset(reset),
19          .clk_100mhz(clk_100mhz)
20      );
22      always #1 clk_100mhz = ~clk_100mhz;
23
24      initial #00_0000 pause_start = 0;
25      initial #00_0000 reset = 0;
26      initial #00_0000 clk_100mhz = 0;
27
28      initial #01_0000 reset = 1;
29      initial #02_0000 reset = 0;
30
31      initial #03_0000 pause_start = 1;
32      initial #04_0000 pause_start = 0;
33
34      initial #05_0000 pause_start = 1;
35      initial #06_0000 pause_start = 0;
36
37      initial #07_0000 reset = 1;
38      initial #08_0000 reset = 0;
39
40      initial #09_0000 pause_start = 1;
41      initial #10_0000 pause_start = 0;
42
43      initial #11_0000 pause_start = 1;
44      initial #12_0000 pause_start = 0;
45
46      initial #13_0000 pause_start = 1;
47      initial #14_0000 pause_start = 0;
48
49      initial #15_0000 reset = 1;
50      initial #16_0000 reset = 0;
51
52  endmodule
```

波型圖如下。cathode 要等到一個一赫茲的時脈產生之後才會有值，然而，整個 stimulus 的過程只有數秒鐘，不足以使除頻器裡的計數器從一數到五千萬。因此，在整個 stimulus 的數秒鐘內，cathode 都是紅的。特別留意當 pause_start 或 reset 被按下的時候，state 是如何改變的。



至於 Experiment1，只要微調 module LAB_pre1 的輸出，如下圖，並刪掉原本用來顯示狀態的第 77 與 78 行即可。

```
1  module LAB5_1(                  // TOP
2      output [ 7:0] cathode,      // cathodes of similar segments on all four displays
3      output [ 3:0] anode,        // anodes of the seven LEDs forming each digit
4      output [15:0] led,          // When the counter goes to 0, all the LEDs will be lighted up
5      input         pause_start,  // a button to pause or resume the counting process
6      input         reset,        // a button to stop the counting process and reset the value to 30 sec.
7      input         clk_100mhz    // a 100 MHz global clock
8  );
```

下圖是用來 demo Pre-lab1 和 Experiment1 的。

anode (4)	OUT		LVCMOS33*
anode[3]	OUT	W4	LVCMOS33*
anode[2]	OUT	V4	LVCMOS33*
anode[1]	OUT	U4	LVCMOS33*
anode[0]	OUT	U2	LVCMOS33*
cathode (8)	OUT		LVCMOS33*
cathode[7]	OUT	W7	LVCMOS33*
cathode[6]	OUT	W6	LVCMOS33*
cathode[5]	OUT	U8	LVCMOS33*
cathode[4]	OUT	V8	LVCMOS33*
cathode[3]	OUT	U5	LVCMOS33*
cathode[2]	OUT	V5	LVCMOS33*
cathode[1]	OUT	U7	LVCMOS33*
cathode[0]	OUT	V7	LVCMOS33*
led (16)	OUT		LVCMOS33*
led[15]	OUT	L1	LVCMOS33*
led[14]	OUT	P1	LVCMOS33*
led[13]	OUT	N3	LVCMOS33*
led[12]	OUT	P3	LVCMOS33*
led[11]	OUT	U3	LVCMOS33*
led[10]	OUT	W3	LVCMOS33*
led[9]	OUT	V3	LVCMOS33*
led[8]	OUT	V13	LVCMOS33*
led[7]	OUT	V14	LVCMOS33*
led[6]	OUT	U14	LVCMOS33*
led[5]	OUT	U15	LVCMOS33*
led[4]	OUT	W18	LVCMOS33*
led[3]	OUT	V19	LVCMOS33*
led[2]	OUT	U19	LVCMOS33*
led[1]	OUT	E19	LVCMOS33*
led[0]	OUT	U16	LVCMOS33*
Scalar ports (3)			
clk_100mhz	IN	W5	LVCMOS33*
pause_start	IN	W19	LVCMOS33*
reset	IN	T17	LVCMOS33*

e. Conclusion :

這一題最讓我感到困難的是碼表的 enable。一開始我直接把碼表內計數器的 enable 直接與碼表外部的 enable 連接，但是發現會有問題。因為當碼表內的計數器數到零秒的時候，所有計數器的 enable 都還是開啟的狀態。這會使得計數器輸出 borrow 的訊號，造成下一個時脈的數值變成 9999。為了讓這四個計數器能在數到零秒的時候自動停下來，我先試著另外使用一個 and gate，先判斷碼表外部有 enable 訊號，而且當下四個計數器的數值都不為零，才輸出訊號到第一個計數器的 decrease。

Experiment2:

The same function as Exp. 1. Instead of using two push buttons for reset/pause/start, try to use just one push button to finish the design. (Hint: You can press the push button longer to represent the reset)

a. Specification :

Inputs:

button // 短按切換狀態(下數/暫停)，長按重置

clk_100mhz // FPGA 內建的時脈，頻率為 100 M Hz

Outputs:

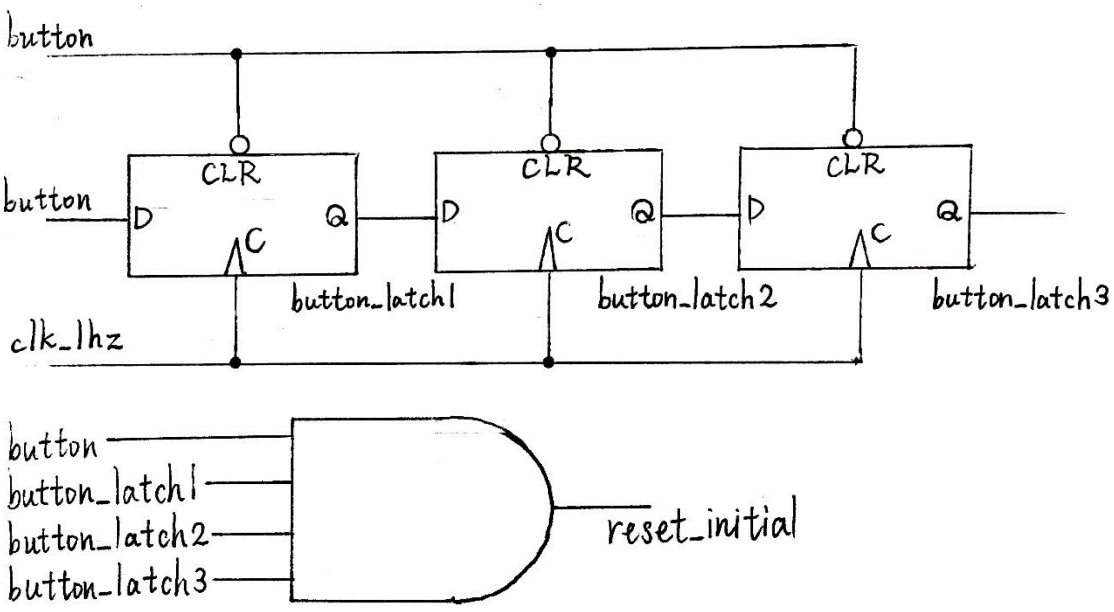
cathode[7:0] // 控制一個七段顯示器中各個 segment 的亮暗

anode[3:0] // 控制要啟用哪一個七段顯示器

led[15:0] // 計數器數到零的時候，亮燈

b. Block Diagram :

原本 Experiment1 中，所有 reset 訊號由獨立的一個按鈕控制。為了將 reset 功能按鈕與 pause_start 合併成單一按鈕 button，我們可以按久一點來表示 reset 信號。以 Experiment1 為基礎，只要再多使用三個 D Flip-flop 就可以偵測按鈕被長按。如下圖，如果三個 D Flip-flop 都是 1，且 button 被按下，就輸出重置信號(reset_initial)給其他模組。



c. FSM :

跟 Experiment1 一模一樣，用一個 T Flip-flop 來切換模式。

d. Implement :

直接沿用 Experiment1 的 module，僅需微調 LAB5_1 即可。接下來的幾張圖片是微調後的 LAB5_2。把 Experiment1 的 input pause_start 和 input reset 合併成單一按鈕 input button；新增了一些接線：wire reset_initial 用來代替 Experimemt1 中原本的 input reset，且多了 reg button_latch 用來偵測按鈕長按，如下圖。

```
1  module LAB5_2(                    // TOP
2      output [ 7:0] cathode,        // cathodes of similar segments on all four displays
3      output [ 3:0] anode,          // anodes of the seven LEDs forming each digit
4      output [15:0] led,            // When the counter goes to 0, all the LEDs will be lighted up
5      input         button,         // a button to reset, pause, start, or resume the counting process
6      input         clk_100mhz      // a 100 MHz global clock
7  );
8      wire [3:0] bcd1;              // between the stopwatch and the decoder
9      wire [3:0] bcd2;              // between the stopwatch and the decoder
10     wire [3:0] bcd3;              // between the stopwatch and the decoder
11     wire [3:0] bcd4;              // between the stopwatch and the decoder
12     wire [7:0] ssd1;              // between the decoder and the controller
13     wire [7:0] ssd2;              // between the decoder and the controller
14     wire [7:0] ssd3;              // between the decoder and the controller
15     wire [7:0] ssd4;              // between the decoder and the controller
16     wire       clk_1hz;           // 1Hz 50% duty clock produced by the frequency divider
17     wire       reset_initial;     // whether to reset counter and reinitialize
18     reg  [2:0] button_latch;      // to determine how long the button has been pressed and not released
19     reg        enable;            // whether to count or pause (the counter)
```

與其他模組的接線沒有太多更動，把原本的 reset 都改成 reset_initial 即可。

```
21      frequency_divider_100M FD(          // 100M Hz to 1 Hz 50% duty frequency divider
22          .clk_1hz   (clk_1hz),           // output frequency
23          .clk_100mhz(clk_100mhz),        // crystal frequency
24          .rst_n     (~reset_initial)     // active low reset
25      );
26
27      stopwatch SW(                               // a down-counting stopwatch, able to handle 59:59 – 00:00
28          .digit1         (bcd1),                 // the binary value of 1st digit
29          .digit2         (bcd2),                 // the binary value of 2nd digit
30          .digit3         (bcd3),                 // the binary value of 3rd digit
31          .digit4         (bcd4),                 // the binary value of 4th digit
32          .initial_value_1(0),                    // the value to be assigned to 1st digit when rst_n
33          .initial_value_2(3),                    // the value to be assigned to 2nd digit when rst_n
34          .initial_value_3(0),                    // the value to be assigned to 3rd digit when rst_n
35          .initial_value_4(0),                    // the value to be assigned to 4th digit when rst_n
36          .clk_1hz        (clk_1hz),              // 1 Hz clock
37          .rst_n          (~reset_initial),       // active low reset
38          .enable         (enable)                // whether to enable the counting process of this counter
39      );
40
41      decoder_binary_to_SSD DECODER1(     // 4-bit binary to SSD pattern decoder
42          .SSD   (ssd1),                  // 7-segment display decode
43          .binary(bcd1)                   // 4-bit binary number
44      );
45
46      decoder_binary_to_SSD DECODER2(     // 4-bit binary to SSD pattern decoder
47          .SSD   (ssd2),                  // 7-segment display decode
48          .binary(bcd2)                   // 4-bit binary number
49      );
50
51      decoder_binary_to_SSD DECODER3(     // 4-bit binary to SSD pattern decoder
52          .SSD   (ssd3),                  // 7-segment display decode
53          .binary(bcd3)                   // 4-bit binary number
54      );
55
56      decoder_binary_to_SSD DECODER4(     // 4-bit binary to SSD pattern decoder
57          .SSD   (ssd4),                  // 7-segment display decode
58          .binary(bcd4)                   // 4-bit binary number
59      );
60
61      SSD_controller SSD_CTRL(            // control common-anode and individual-cathode of seven-segment display
62          .cathode   (cathode),           // cathodes of similar segments on all four displays
63          .anode     (anode),             // anodes of the seven LEDs forming each digit
64          .digit1    (ssd1),              // SSD pattern to display
65          .digit2    (ssd2),              // SSD pattern to display
66          .digit3    (ssd3),              // SSD pattern to display
67          .digit4    (ssd4),              // SSD pattern to display
68          .clk_100mhz(clk_100mhz),        // 100M Hz global clock
69          .rst_n     (~reset_initial)     // active low reset
70      );
```

相較於 Experiment1，Experiment2 多了第 77 到 86 行：第 77 到 82 行是三個 D Flip-flop，將 clear 與反相的 button 連接，用來記錄按鈕已被連續按下多久。第 85 到 86 行用來判斷是否已經被連續長按，如果按鈕被連續長按，就產生重置訊號(reset_initial)。

```
72      // combinational logic: to determine whether to light up all the LEDs
73      assign led = {16{
74          {bcd4, bcd3, bcd2, bcd1} == 0
75      }};
76
77      // shifter: record how long the button has been pressed and not released
78      always @ (posedge clk_1hz or negedge button)
79          if (~button)
80              button_latch <= 0;
81          else
82              button_latch <= {button_latch, button};
83
84      // press the push button longer to represent the reset
85      assign reset_initial =
86          {button_latch, button} == 4'b1111;
87
88      // T flip-flop: changes state of enable whenever the pause_start button is pressed
89      always @ (posedge button or posedge reset_initial)
90          if (reset_initial)
91              enable <= 0;
92          else
93              enable <= ~enable;
94
95  endmodule                                   //LAB5_2
```

這樣就構造完畢了，其他未提及的模組都跟上一題 Experiment1 的一模一樣。

下圖是用來 demo Experiment2 的。

anode (4)	OUT		LVCMOS33*
anode[3]	OUT	W4	LVCMOS33*
anode[2]	OUT	V4	LVCMOS33*
anode[1]	OUT	U4	LVCMOS33*
anode[0]	OUT	U2	LVCMOS33*
cathode (8)	OUT		LVCMOS33*
cathode[7]	OUT	W7	LVCMOS33*
cathode[6]	OUT	W6	LVCMOS33*
cathode[5]	OUT	U8	LVCMOS33*
cathode[4]	OUT	V8	LVCMOS33*
cathode[3]	OUT	U5	LVCMOS33*
cathode[2]	OUT	V5	LVCMOS33*
cathode[1]	OUT	U7	LVCMOS33*
cathode[0]	OUT	V7	LVCMOS33*
led (16)	OUT		LVCMOS33*
led[15]	OUT	L1	LVCMOS33*
led[14]	OUT	P1	LVCMOS33*
led[13]	OUT	N3	LVCMOS33*
led[12]	OUT	P3	LVCMOS33*
led[11]	OUT	U3	LVCMOS33*
led[10]	OUT	W3	LVCMOS33*
led[9]	OUT	V3	LVCMOS33*
led[8]	OUT	V13	LVCMOS33*
led[7]	OUT	V14	LVCMOS33*
led[6]	OUT	U14	LVCMOS33*
led[5]	OUT	U15	LVCMOS33*
led[4]	OUT	W18	LVCMOS33*
led[3]	OUT	V19	LVCMOS33*
led[2]	OUT	U19	LVCMOS33*
led[1]	OUT	E19	LVCMOS33*
led[0]	OUT	U16	LVCMOS33*
Scalar ports (2)			
button	IN	U18	LVCMOS33*
clk_100mhz	IN	W5	LVCMOS33*

e. Conclusion :

建議往後在著手建構 Verilog 前，先完整的瀏覽過所有題目，掌握該次 LAB 要實作的所有內容。畢竟這些 LAB 都是經過精心安排的，性質相近的題目通常會放在一起。如果一開始就把所有功能都考慮進來，就不會發生某個 module 在前幾題可堪使用，但不適用後面的題目的情況。這樣一來，就可以達到一個模組重複利用的優勢，節省下重複開發類似功能模組的時間與心力。

Experiment3:

(Bonus) Use two push buttons to control a multi-function stop timer (mode selection, reset, start, stop). The stop timer has two modes: 30-second/1-minute countdown. When being reset, the seven-segment display shows the digits 30/1:00. When the timer counts to 0, it will stop.

3.1 List the specification of the detector.

3.2 Design the FSM used in this design.

3.3 Draw the block diagram/logic schematic.

3.4 Implement the stop timer with FPGA demo board.

a. Specification :

Inputs:

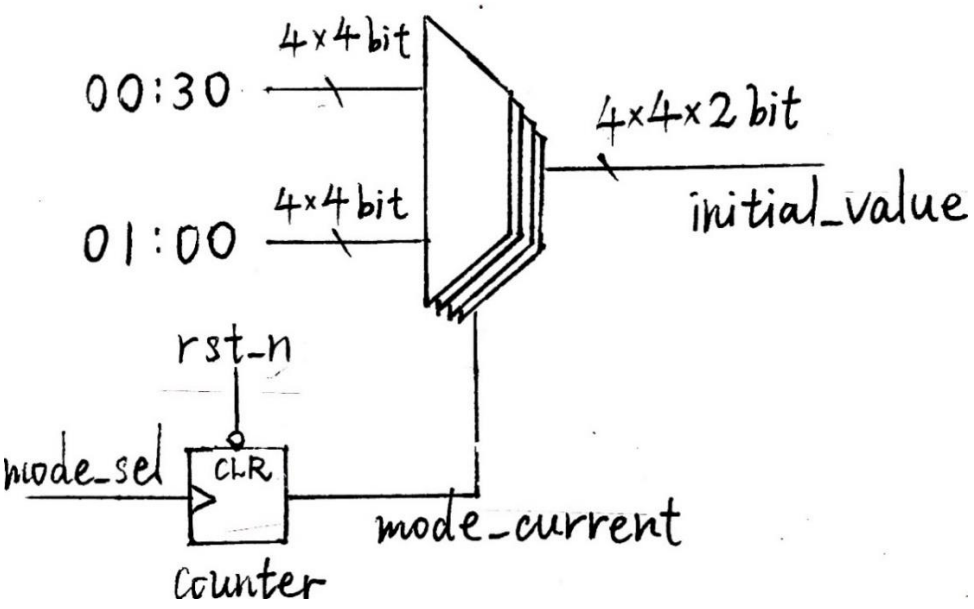
button	// 短按切換狀態(下數/暫停)，長按重置
mode_select	// 按一下會切換倒數 30 秒或一分鐘
clk_100mhz	// FPGA 內建的時脈，頻率為 100 M Hz

Outputs:

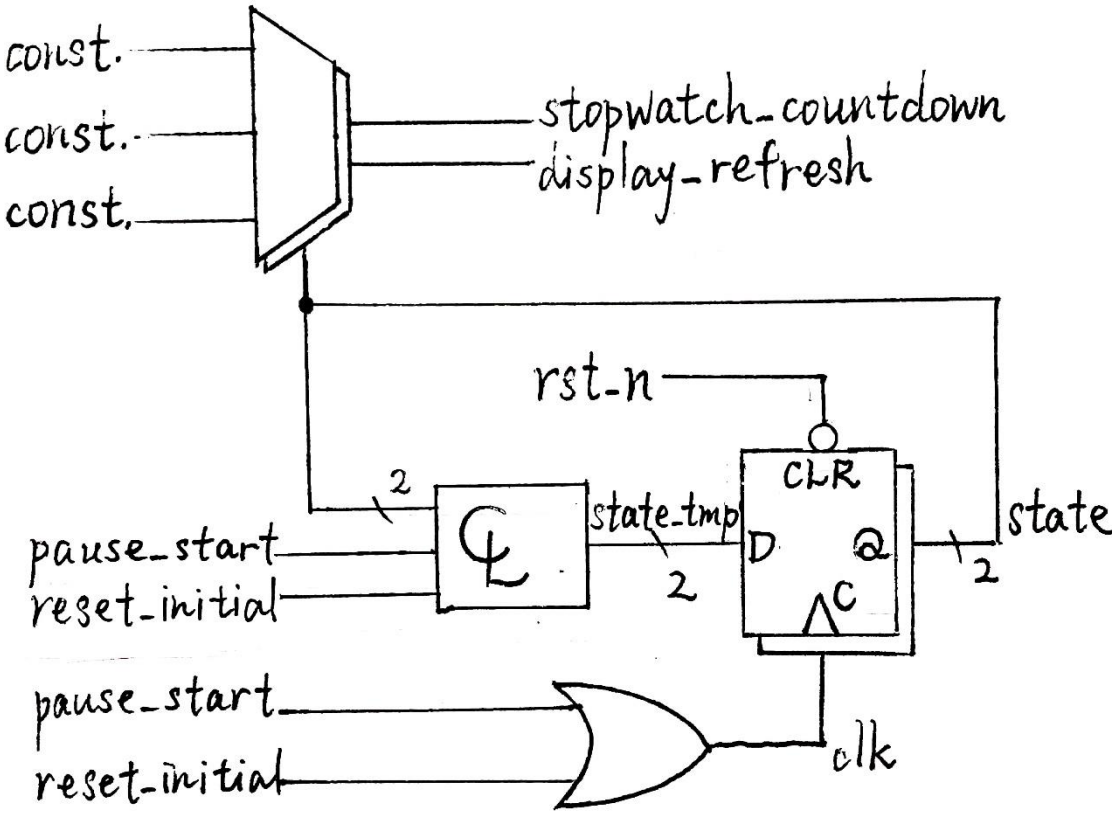
cathode[7:0]	// 控制一個七段顯示器中各個 segment 的亮暗
anode[3:0]	// 控制要啟用哪一個七段顯示器
led[15:0]	// 計數器數到零的時候，亮燈

b. Block Diagram :

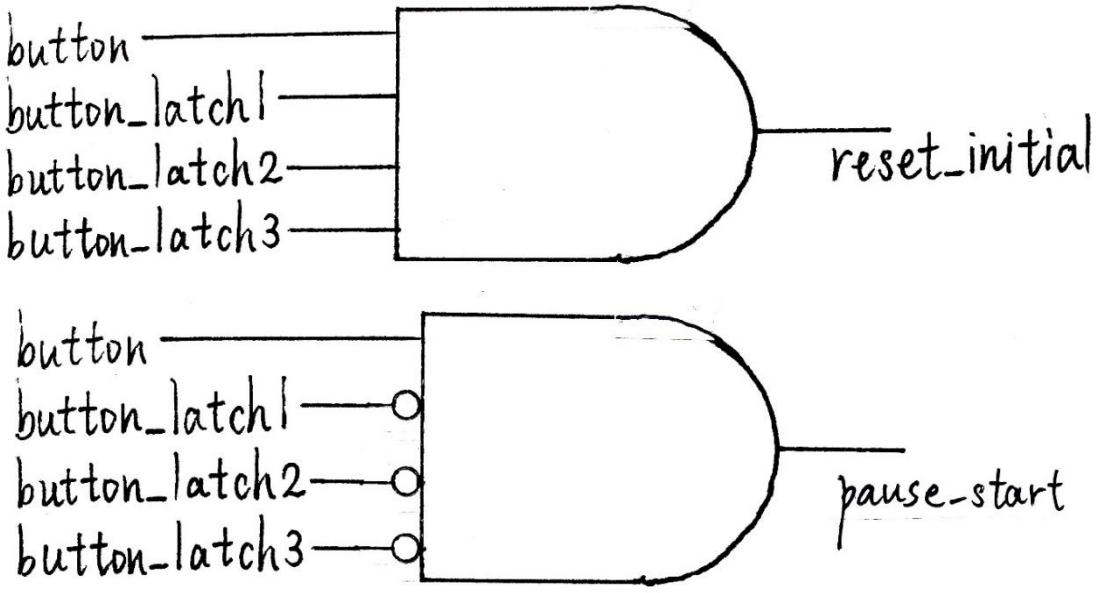
較 Experiment2，多了 mode_selector 和 FSM，其餘一模一樣。mode_selector 由兩部分組成，先用一個計數器來儲存目前的模式(mode_current)，再連接到一個 MUX 來產生該模式對應的初始數值(initial_value)。如果未來有需要新增更多 mode，只要增加計數器的位元數和 MUX 的選項即可。但其實因為這一題只有兩個 mode，所以可以用一個 T Flip-flop 來代替計數器。



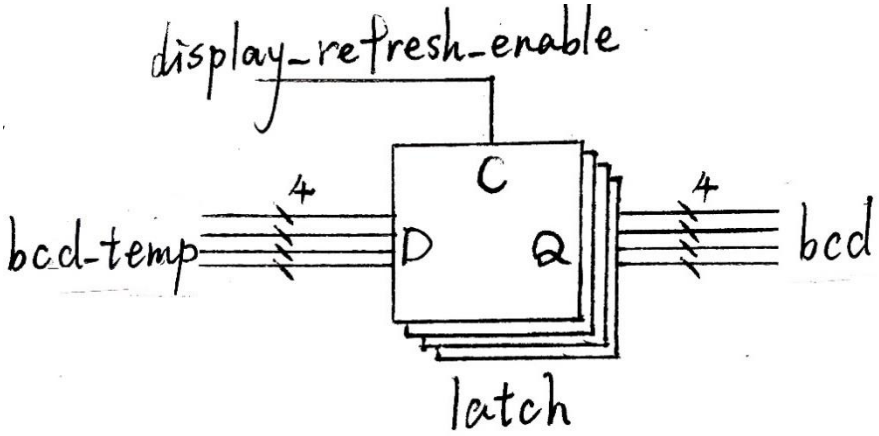
這題的 FSM 輸出只由當前的狀態所確定，也就是 Moore machine。共三個狀態：正在倒數、凍結顯示器、重置，所以需要兩個 D Flip-flop 來儲存當下的狀態。當 FSM 收到切換 pause 或 start 的訊號，或是收到 reset 的信號，就更新狀態。所以 D Flip-flop 的 C 連接到 pause_start | reset_initial。



reset 的訊號跟 Experiment2 一樣，條件是連續按下按鈕達三個時脈以上；而切換 pause 或 start 的訊號則是短按一下按鈕，如下圖。



為了使七段顯示器能凍結，可以在把碼表的數值傳送到解碼器之前就先檢查當下的狀態是否為凍結。如下圖，如果要凍結，則不更新 latch 的數值。



其他未提及的部分跟 Experiment2 一樣，直接沿用即可。

c. FSM :

state:

S0: RESETED

S1: COUNTING

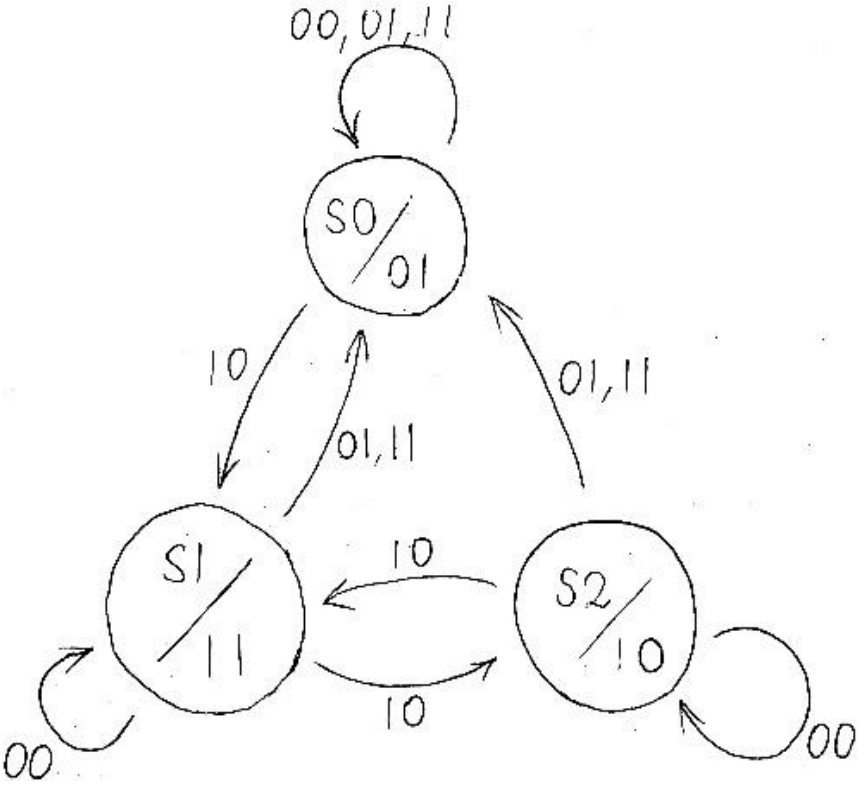
S2: FREEZED

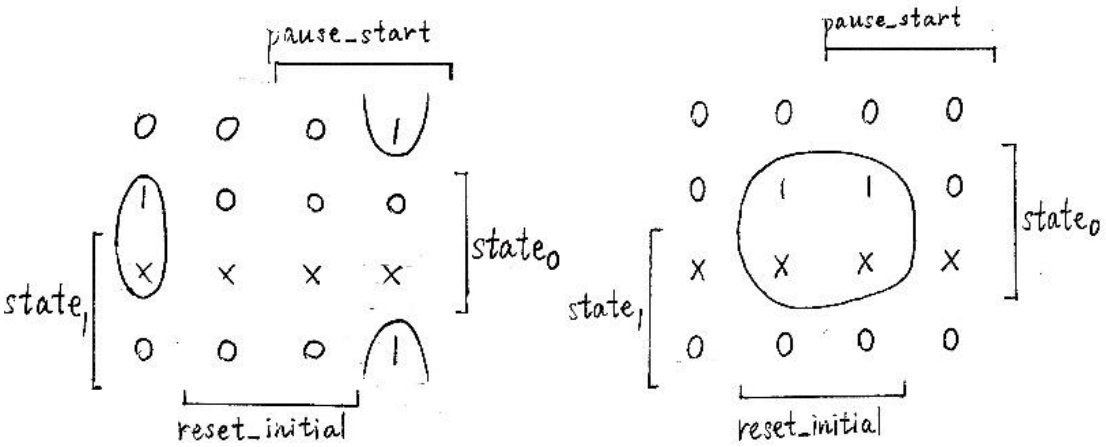
input:

pause_start/reset_initial

output:

stopwatch_countdown_enable/display_refresh_enable





Next state equations/Excitation equations:

$state_0(t+1) = D_0(state_1, state_0, pause\_start, reset\_initial)$

$= \Sigma(2, 4, 10)$

$= state_0 \times pause\_start' \times reset\_initial' +$

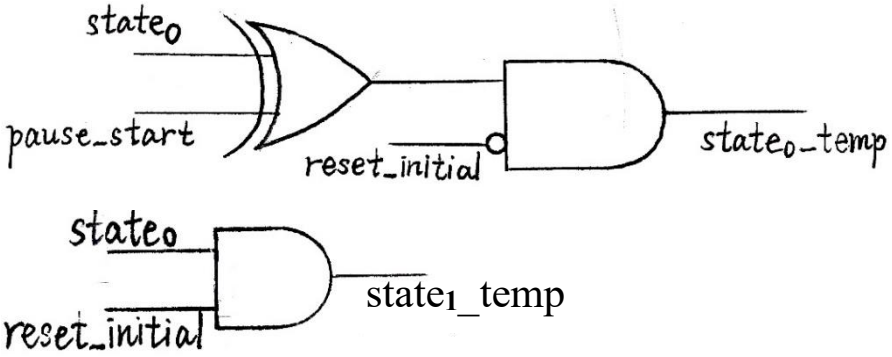
$state_0' \times pause\_start \times reset\_initial'$

$= (state_0 \oplus pause\_start) \times reset\_initial'$

$state_1(t+1) = D_1(state_1, state_0, pause\_start, reset\_initial)$

$= \Sigma(6, 8)$

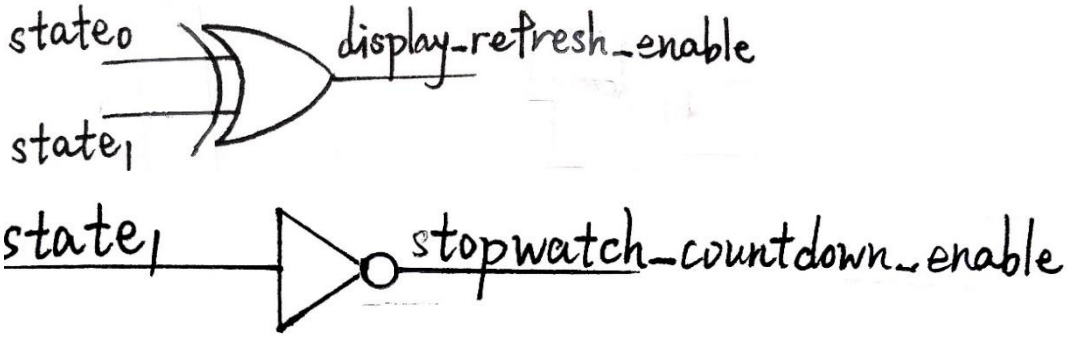
$= state_0 \times reset\_initial$



Output equations:

$stopwatch\_countdown\_enable = state_1'$

$display\_refresh\_enable = state_0 \oplus state_1$



d. Implement :

這題只要多建構兩個模組 mode_selector、FSM，並稍加修改 Experiment1 的 LAB_2 就好，其餘的可以直接沿用。mode_selector 在不同的模式下，要將不同的初始數值送到碼表內。可以用 MUX，如下圖。

```
module mode_selector(
    output reg [3:0] initial_value_1,  // the value to be assigned to the 1st digit when rst_n and ~reset_initial
    output reg [3:0] initial_value_2,  // the value to be assigned to the 2nd digit when rst_n and ~reset_initial
    output reg [3:0] initial_value_3,  // the value to be assigned to the 3rd digit when rst_n and ~reset_initial
    output reg [3:0] initial_value_4,  // the value to be assigned to the 4th digit when rst_n and ~reset_initial
    input            mode_select,      // The stop timer has two modes: 30-second/1-minute countdown
    input            rst_n             // active low reset
);
    reg mode_current;

    // MUX: combinational logic for the mode (01:00 or 00:30)
    always @ *
        case (mode_current)
            0:       {initial_value_4,
                      initial_value_3,
                      initial_value_2,      //  0    0    3    0
                      initial_value_1} <= 16'b0000_0000_0011_0000;
            1:       {initial_value_4,
                      initial_value_3,
                      initial_value_2,      //  0    1    0    0
                      initial_value_1} <= 16'b0000_0001_0000_0000;
            default: {initial_value_4,
                      initial_value_3,
                      initial_value_2,      //  F    F    F    F
                      initial_value_1} <= 16'b1111_1111_1111_1111;
        endcase

    // counter: sequential circuit to toggle between (possibly) multiple modes
    always @ (posedge mode_select or negedge rst_n)
        if (~rst_n)
            mode_current <= 0;
        else
            mode_current <= mode_current + 1;

endmodule                               // mode_selector
```

接下來的幾張圖片是模組 FSM 的硬體描述語言。FSM 的輸出、輸入與一些內部線路的名稱如下。

```
module FSM(                                 // finite state machine
    output reg stopwatch_countdown_enable,  // to determine whether to enable the countdown function of the stopwatch
    output reg display_refresh_enable,      // to determine whether not to freeze the display of the counter
    input      pause_start,                 // to determine whether to freeze or resume the display of counting process
    input      reset_initial,               // to determine whether to reset and reinitial the counting process
    input      rst_n                        // active low reset
);
    parameter TRUE  = 1;
    parameter FALSE = 0;
    parameter STATE_RESETED  = 0;
    parameter STATE_COUNTING = 1;
    parameter STATE_FREEZED  = 2;
    wire      clk;                  // to update current state when rising edge of pause_start or reset_initial hits
    reg [1:0] state;                // current state
    reg [1:0] state_temp;           // next state
```

因為是 moore machine，所以輸出不受輸入影響，可由狀態直接決定。如下圖，可以用 MUX 來依照當下的狀態來產生對應的輸出。

```
    // combinational logic: to determine whether to enable the countdown function of the stopwatch in the next state
    always @ *
        case (state)
            STATE_RESETED:  stopwatch_countdown_enable <= FALSE;
            STATE_COUNTING: stopwatch_countdown_enable <= TRUE;
            STATE_FREEZED:  stopwatch_countdown_enable <= TRUE;
            default:        stopwatch_countdown_enable <= FALSE;
        endcase

    // combinational logic: to determine whether not to freeze the display of the counter in the next state
    always @ *
        case (state)
            STATE_RESETED:  display_refresh_enable <= TRUE;
            STATE_COUNTING: display_refresh_enable <= TRUE;
            STATE_FREEZED:  display_refresh_enable <= FALSE;
            default:        display_refresh_enable <= TRUE;
        endcase
```

下圖是下一個狀態的 combinational logic。在正常情況下，pause_start 與 reset_initial 不會同時為 1，而且第 60 到 67 行的 D Flip-flop 只有在 pause_start 或 reset_initial 為 1 時才更新狀態。因此，只需要明確列出 01 和 10 的 case 即可。

```
35      // combinational logic: to determine the next state
36      always @ *
37          case (state)
38              STATE_RESETED:
39                  case ({pause_start, reset_initial})
40                      2'b01:   state_temp <= STATE_RESETED;
41                      2'b10:   state_temp <= STATE_COUNTING;
42                      default: state_temp <= STATE_RESETED;
43                  endcase
44              STATE_COUNTING:
45                  case ({pause_start, reset_initial})
46                      2'b01:   state_temp <= STATE_RESETED;
47                      2'b10:   state_temp <= STATE_FREEZED;
48                      default: state_temp <= STATE_RESETED;
49                  endcase
50              STATE_FREEZED:
51                  case ({pause_start, reset_initial})
52                      2'b01:   state_temp <= STATE_RESETED;
53                      2'b10:   state_temp <= STATE_COUNTING;
54                      default: state_temp <= STATE_RESETED;
55                  endcase
56              default: state_temp <= STATE_RESETED;
57          endcase
```

第 60 行的邏輯閘使得第 60 到 67 行的 D Flip-flop 僅在 pause_start 或 reset_initial 的 rising edge 時更新 FSM 的狀態。

```
59      // combinational logic: to determine when to update the current state to the next state
60      assign clk = pause_start | reset_initial;
61
62      // D Flip-flop: store the current state and update to the next state when pause_start or reset_initial hits
63      always @ (posedge clk or negedge rst_n)
64          if (~rst_n)
65              state <= STATE_RESETED;
66          else
67              state <= state_temp;
68
69  endmodule                           // FSM
```

至此已萬事具備，剩下的只要微調 LAB5_2 就好。接下來的幾張圖片是修改後的 LAB5_3。輸出、輸入和一些內部接線如下。

```
module LAB5_3(                  // TOP
    output [ 7:0] cathode,      // cathodes of similar segments on all four displays
    output [ 3:0] anode,        // anodes of the seven LEDs forming each digit
    output [15:0] led,          // When the counter goes to 0, all the LEDs will be lighted up
    input         button,       // a button to reset, freeze, or start the display of counting process
    input         mode_select,  // The stop timer has two modes: 30-second/1-minute countdown
    input         clk_100mhz,   // a 100 MHz global clock
    input         rst_n         // active low reset
);
    wire [3:0] initial_value_1; // the value to be assigned to 1st digit when rst_n
    wire [3:0] initial_value_2; // the value to be assigned to 2nd digit when rst_n
    wire [3:0] initial_value_3; // the value to be assigned to 3rd digit when rst_n
    wire [3:0] initial_value_4; // the value to be assigned to 4th digit when rst_n
    reg  [3:0] bcd1;            // the binary value of 1st digit to display (may be freezed)
    reg  [3:0] bcd2;            // the binary value of 2nd digit to display (may be freezed)
    reg  [3:0] bcd3;            // the binary value of 3rd digit to display (may be freezed)
    reg  [3:0] bcd4;            // the binary value of 4th digit to display (may be freezed)
    wire [3:0] bcd1_temp;       // the actual binary value of 1st digit
    wire [3:0] bcd2_temp;       // the actual binary value of 2nd digit
    wire [3:0] bcd3_temp;       // the actual binary value of 3rd digit
    wire [3:0] bcd4_temp;       // the actual binary value of 4th digit
    wire [7:0] ssd1;            // between the decoder and the controller
    wire [7:0] ssd2;            // between the decoder and the controller
    wire [7:0] ssd3;            // between the decoder and the controller
    wire [7:0] ssd4;            // between the decoder and the controller
    wire       clk_1hz;         // 1Hz 50% duty clock produced by the frequency divider
    wire       pause_start;     // to determine whether to freeze or resume the display of counting process
    wire       reset_initial;               // to determine whether to reset and reinitial the counting process
    wire       stopwatch_countdown_enable;  // to determine whether to enable the countdown function of the stopwatch
    wire       display_refresh_enable;      // to determine whether not to freeze the display of the counter
    reg  [2:0] button_latch;                // to determine how long the button has been pressed and not released
```

新加入兩個模組，mode_selector 和 FSM。mode_selector 決定當下的倒數計時模式，在一分鐘和三十秒之間切換，且輸出對應的初始數值給碼表作重置用。而 FSM 則決定當下碼表的狀態，會依照輸入的不同，決定按按鈕之後會變成哪一個狀態。

```
    mode_selector MODE_SEL(                 // The stop timer has two modes: 30-second/1-minute countdown
        .initial_value_1(initial_value_1),  // the value to be assigned to the 1st digit when rst_n and ~reset_initial
        .initial_value_2(initial_value_2),  // the value to be assigned to the 2nd digit when rst_n and ~reset_initial
        .initial_value_3(initial_value_3),  // the value to be assigned to the 3rd digit when rst_n and ~reset_initial
        .initial_value_4(initial_value_4),  // the value to be assigned to the 4th digit when rst_n and ~reset_initial
        .mode_select    (mode_select),      // The stop timer has two modes: 30-second/1-minute countdown
        .rst_n          (rst_n)             // active low reset
    );

    FSM FSM(                                                    // finite state machine
        .stopwatch_countdown_enable(stopwatch_countdown_enable), // to determine whether to enable the countdown function of the stopwatch
        .display_refresh_enable    (display_refresh_enable),    // to determine whether not to freeze the display of the counter
        .pause_start               (pause_start),               // to determine whether to freeze or resume the display of counting process
        .reset_initial             (reset_initial),             // to determine whether to reset and reinitial the counting process
        .rst_n                     (rst_n)                      // active low reset
    );
```

須留意碼表的數值輸出後須先經過第 107 到 114 行的 latch 判斷當下狀態是否為凍結顯示器，再連接到解碼器。

```
50      frequency_divider_100M FD(    // 100M Hz to 1 Hz 50% duty frequency divider
51          .clk_1hz   (clk_1hz),     // output frequency
52          .clk_100mhz(clk_100mhz),  // crystal frequency
53          .rst_n     (rst_n)        // active low reset
54      );
55
56      stopwatch SW(                                   // a down-counting stopwatch, able to handle 59:59 – 00:00
57          .digit1         (bcd1_temp),                // the binary value of the 1st digit
58          .digit2         (bcd2_temp),                // the binary value of the 2nd digit
59          .digit3         (bcd3_temp),                // the binary value of the 3rd digit
60          .digit4         (bcd4_temp),                // the binary value of the 4th digit
61          .initial_value_1(initial_value_1),          // the value to be assigned to the 1st digit when rst_n and ~reset_initial
62          .initial_value_2(initial_value_2),          // the value to be assigned to the 2nd digit when rst_n and ~reset_initial
63          .initial_value_3(initial_value_3),          // the value to be assigned to the 3rd digit when rst_n and ~reset_initial
64          .initial_value_4(initial_value_4),          // the value to be assigned to the 4th digit when rst_n and ~reset_initial
65          .clk_1hz        (clk_1hz),                  // 1 Hz clock
66          .rst_n          (rst_n && ~reset_initial),  // active low reset
67          .enable         (stopwatch_countdown_enable) // whether to enable the counting process of this counter
68      );
```

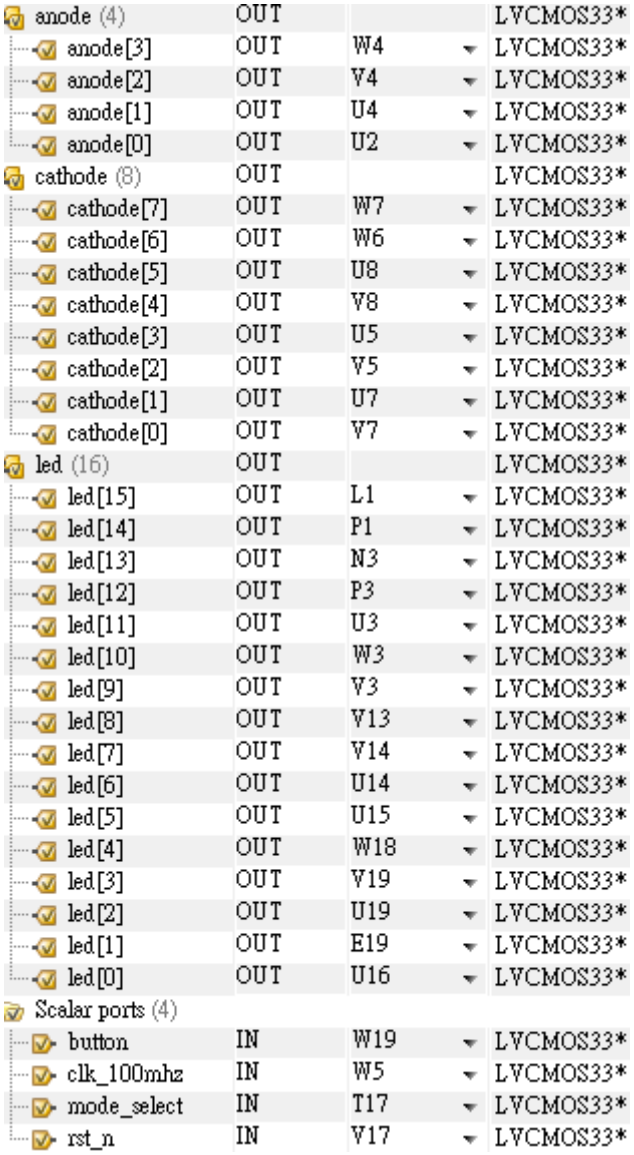
其餘模組的接線保持不變。

```
70      decoder_binary_to_SSD DECODER1(  // 4-bit binary to SSD pattern decoder
71          .SSD   (ssd1),               // 7-segment display decode
72          .binary(bcd1)                // 4-bit binary number
73      );
74
75      decoder_binary_to_SSD DECODER2(  // 4-bit binary to SSD pattern decoder
76          .SSD   (ssd2),               // 7-segment display decode
77          .binary(bcd2)                // 4-bit binary number
78      );
79
80      decoder_binary_to_SSD DECODER3(  // 4-bit binary to SSD pattern decoder
81          .SSD   (ssd3),               // 7-segment display decode
82          .binary(bcd3)                // 4-bit binary number
83      );
84
85      decoder_binary_to_SSD DECODER4(  // 4-bit binary to SSD pattern decoder
86          .SSD   (ssd4),               // 7-segment display decode
87          .binary(bcd4)                // 4-bit binary number
88      );
89
90      SSD_controller SSD_CTRL(         // control common-anode and individual-cathode of seven-segment display
91          .cathode   (cathode),        // cathodes of similar segments on all four displays
92          .anode     (anode),          // anodes of the seven LEDs forming each digit
93          .digit1    (ssd1),           // SSD pattern to display
94          .digit2    (ssd2),           // SSD pattern to display
95          .digit3    (ssd3),           // SSD pattern to display
96          .digit4    (ssd4),           // SSD pattern to display
97          .clk_100mhz(clk_100mhz),     // 100M Hz global clock
98          .rst_n     (rst_n)           // active low reset
99      );
```

第 102 到 104 行有微調。原本是拿 bcd4、bcd3、bcd2、bcd1 來判斷是否已經數到零了，然而由於有可能因為持續保持凍結顯示器的狀態，有可能會出現明明已經過了 30 秒或一分鐘，燈仍然不會亮起。因此，應改為以 latch 前面的數值為判斷依據，如此一來，縱使顯示器的時間被凍結了，一旦碼表數到零，燈還是會亮起。第 107 到 114 是用來凍結顯示器的 latch。另須留意，第 129 行的判定相較 Experiment2 之下，還需要考慮 mode_select 是否有被按下。這樣無論碼表是否在計時，無論顯示器是否被凍結，只要按下 mode_select 按鈕，就會立刻重置，並顯示切換後該模式的計時秒數。其餘未提及的部分與 Experiment2 一模一樣，直接沿用即可。

```
101     // combinational logic: to determine whether to light up all the LEDs
102     assign led = {16{
103         {bcd4_temp, bcd3_temp, bcd2_temp, bcd1_temp} == 0
104     }};
105
106     // latch: to freeze the display when refreshing is not enabled
107     always @ *
108         if (display_refresh_enable)
109         begin
110             bcd4 <= bcd4_temp;
111             bcd3 <= bcd3_temp;
112             bcd2 <= bcd2_temp;
113             bcd1 <= bcd1_temp;
114         end
115
116     // shifter: record how long the button has been pressed and not released
117     always @ (posedge clk_1hz or negedge button)
118         if (~button)
119             button_latch <= 0;
120         else
121             button_latch <= {button_latch, button};
122
123     // press the push button shorter to represent the pause (freeze) and the start (resume)
124     assign pause_start =
125         {button_latch, button} == 4'b0001;
126
127     // press the push button longer to represent the reset
128     assign reset_initial =
129         {button_latch, button} == 4'b1111 || mode_select;
130
131 endmodule                           //LAB5_3
```

下圖是用來 demo Experiment3 的。



anode (4)	OUT		LVCMOS33*
anode[3]	OUT	W4	LVCMOS33*
anode[2]	OUT	V4	LVCMOS33*
anode[1]	OUT	U4	LVCMOS33*
anode[0]	OUT	U2	LVCMOS33*
cathode (8)	OUT		LVCMOS33*
cathode[7]	OUT	W7	LVCMOS33*
cathode[6]	OUT	W6	LVCMOS33*
cathode[5]	OUT	U8	LVCMOS33*
cathode[4]	OUT	V8	LVCMOS33*
cathode[3]	OUT	U5	LVCMOS33*
cathode[2]	OUT	V5	LVCMOS33*
cathode[1]	OUT	U7	LVCMOS33*
cathode[0]	OUT	V7	LVCMOS33*
led (16)	OUT		LVCMOS33*
led[15]	OUT	L1	LVCMOS33*
led[14]	OUT	P1	LVCMOS33*
led[13]	OUT	N3	LVCMOS33*
led[12]	OUT	P3	LVCMOS33*
led[11]	OUT	U3	LVCMOS33*
led[10]	OUT	W3	LVCMOS33*
led[9]	OUT	V3	LVCMOS33*
led[8]	OUT	V13	LVCMOS33*
led[7]	OUT	V14	LVCMOS33*
led[6]	OUT	U14	LVCMOS33*
led[5]	OUT	U15	LVCMOS33*
led[4]	OUT	W18	LVCMOS33*
led[3]	OUT	V19	LVCMOS33*
led[2]	OUT	U19	LVCMOS33*
led[1]	OUT	E19	LVCMOS33*
led[0]	OUT	U16	LVCMOS33*
Scalar ports (4)			
button	IN	W19	LVCMOS33*
clk_100mhz	IN	W5	LVCMOS33*
mode_select	IN	T17	LVCMOS33*
rst_n	IN	V17	LVCMOS33*

e. Conclusion :

其實這一題的 FSM 可以像老師的 ppt 一樣按照推導出來的 table 直接使用 case 和 if…else…列出來就好，不需要像剛剛提到的一樣分成兩個 output 的 MUX 和一個 next state 的 MUX 來寫。但因為這題是 moore machine，output 不受 input 影響，可以單由 state 就決定，所以即使把硬體描述展開來寫，也不會感到過度雜亂。一開始在建構 FSM 的時候，我將 D Flip-flop 的 C 連接到 TOP 的一赫茲時脈，造成按鈕被按下後，還要等到下一個時脈過了才會更新到下一個狀態。後來，我將 input 直接連接到 D Flip-flop 的 C，即達成「按鈕按下的當下就有反應」的效果。