1 Implement a stopwatch function (00:00-59:59) with the FPGA board.
1.1 Use the four (Seven-Segment Displays, SSDs) as the display. The left two digits represent the minute and the right two digits represent the second.
1.2 Use two push buttons to control the function. Use one button to control start/stop and the other to control the lap and reset. When the stopwatch counts, press the 'lap' button will freeze the SSDs but the stopwatch continues counting, and when press the 'lap' button again, the SSDs will start to show current time.

1. Specification
內容: 寫下你的電路中的 inputs, outputs 以及其 bit widths，名稱必須跟你的 verilog code 中相同。

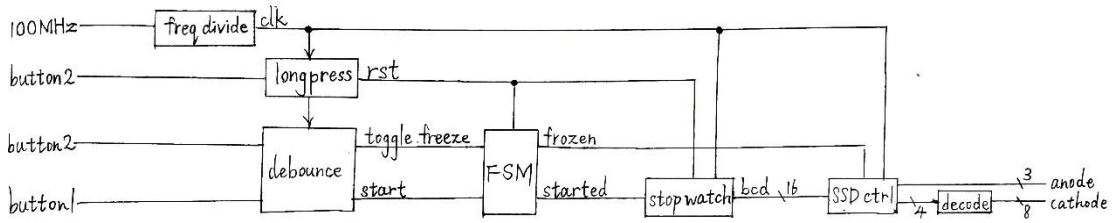
```
module LAB6_1(
  output [7:0] cathode,     // 控制七段顯示器
  output [3:0] anode,       // 控制七段顯示器
  input        push1,       // 按一下會開始計時
  input        push2,       // 按一下會凍結或解除凍結
  input        f_100mhz,    // 內建的時脈
  input        rst_n        // 可強制關機
);

module upcounter(
  output reg [3:0] value,   // 現在顯示的數值
  output           c_out,   // 下一個時脈要進位
  input      [3:0] limit,   // 最大允許的數值
  input            c_in,    // 下一個時脈要加一
  input            clk,     // 時脈
  input            rst_n    // 重置
);

module decoder_BCD_to_SSD(
  output reg [7:0] ssd,     // 七段顯示器的樣式
  input      [3:0] bcd      // 二進位的樣式
);
```

```
module stopwatch(
  output [7:0] ssd4,    // 分鐘的十位數字
  output [7:0] ssd3,    // 分鐘的個位數字
  output [7:0] ssd2,    // 秒鐘的十位數字
  output [7:0] ssd1,    // 秒鐘的個位數字
  input        en,      // 啟用，允許上數
  input        f_1hz,   // 時脈
  input        rst_n    // 重置，變成 00:00
);

module SSD_controller(
  output reg [7:0] cathode,    // 控制七段顯示器
  output reg [3:0] anode,      // 控制七段顯示器
  input     [7:0] ssd1,        // 秒鐘的個位數字
  input     [7:0] ssd2,        // 秒鐘的十位數字
  input     [7:0] ssd3,        // 分鐘的個位數字
  input     [7:0] ssd4,        // 分鐘的十位數字
  input           freezed,     // 凍結顯示器
  input     [1:0] clk          // 時脈
);

module FSM(
  output paused,         // 控制碼表是否上數
  output freezed,        // 控制是否凍結顯示器
  input  debounced1,     // 按一下會開始計時
  input  debounced2,     // 按一下會凍結或解除凍結
  input  rst_n           // 重設狀態
);

module frequency_divider(
  output reg        f_1hz,         // 一赫茲的時脈(50% duty)
  output reg [25:0] f_counter,     // 各種不同頻率的時脈
  input             f_100mhz,      // 內建於 FPGA 的時脈
  input             rst_n          // 重置
);
```

```
module debounce(
  output debounced,  // 經穩定後的訊號
  input  push,       // 未經處理的訊號
  input  clk,        // 時脈
  input  rst_n       // 重置
);
```

2. Block Diagram

內容: 電路中的 Block diagram(可以用手畫拍照或電腦繪圖)。

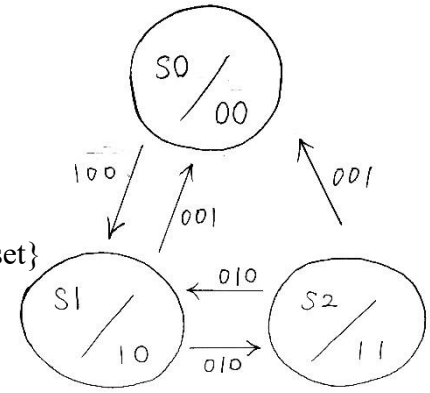


3. Finite state machine

內容: 電路中的 Finite state machine，若無則寫無。

state0: reseted
state1: started
state2: frozen

moore machine inputs: {start, toggle_freeze, reset}
moore machine outputs: {started, frozen}



4. Implement

內容: 請列出相關的 logic function、詳細用文字解釋電路的運作方法、結果等等，可以貼 code 解釋或拍 FPGA 輔助解釋(但不能只貼 code 跟 FPGA 結果)。

下圖是除頻器，跟上次 Lab 的差不多。第 8 到 11 行是計數器。第 13 到 15 行用來製造 50% duty 的一赫茲時脈。將除頻器的 counter 當作 output，讓 top 的其他模組可以參照各種頻率。

```
1  module frequency_divider(
2    output reg        f_1hz,
3    output reg [25:0] f_counter,
4    input             f_100mhz,
5    input             rst_n
6  );
7
8    always @ (posedge f_100mhz or negedge rst_n)
9      if (~rst_n)                    f_counter <= 0;
10     else if (f_counter < 49_999_999) f_counter <= f_counter + 1;
11     else                           f_counter <= 0;
12
13   always @ (posedge f_100mhz or negedge rst_n)
14     if (~rst_n)             f_1hz <= 0;
15     else if (f_counter == 0) f_1hz <= ~f_1hz;
16
17 endmodule
```

因為按鈕按下的瞬間會有擾動，有時候雖然只按一下，但卻會形成數個震盪。所以可以先把未處理的按鈕訊號連接到一個去除雜訊的模組。下圖是用來排除雜訊的，第 9 行到第 11 行是 shifter。第 13 行的 AND 可以用來判斷按鈕是否處於按下的狀態超過數個時脈。如果偵測到連續按下，那就代表按鈕已經穩定。

```
1  module debounce(
2    output debounced,
3    input  push,
4    input  clk,
5    input  rst_n
6  );
7    reg [9:0] window;
8
9    always @ (posedge clk or negedge rst_n)
10     if (~rst_n) window <= 0;
11     else        window <= {window, push};
12
13   assign debounced = &{window, push};
14
15 endmodule
```

下圖是有限狀態機，只有三個狀態。因為 output 只與當下的狀態有關，所以是 moore machine。而且，這三個狀態的 output 兩兩相異，所以可以直接用 output 的組合來表示當下的狀態，如第 13 行。可以用 parameter 來建立狀態與 output 之間的關係，如第 9 到 11 行。狀態是 RESETED 時，碼表不要上數，所以是 2'b10；狀態是 FREEZED 時，碼表要持續下數，而顯示器要凍結，所以是 2'b01；狀態是 STARTED 時，碼表要下數且顯示器不凍結，所以是 2'b00。第 15 到第 18 行用來判斷下一個狀態。只有當按鈕被按下，或是要重置的時候，才需要改變狀態，如第 15 行。如果要重置，就把狀態變成 RESETED，如第 16 行。如果當下的狀態是 STARTED，而且第二個按鈕(toggle_frozen)被按下了，則把狀態變成 frozen (FREEZED)，如第 17 行。其餘的狀況，例如第一個按鈕(start)被按下，會使狀態變成 STARTED，如第 18 行。

```
1  module FSM(
2    output paused,
3    output freezed,
4    input  debounced1,
5    input  debounced2,
6    input  rst_n
7  );
8    reg [1:0] state;
9    parameter RESETED = 2'b10;
10   parameter FREEZED = 2'b01;
11   parameter STARTED = 2'b00;
12
13   assign {paused, freezed} = state;
14
15   always @ (posedge debounced1 || debounced2 or negedge rst_n)
16     if      (~rst_n)                          state <= RESETED;
17     else if (debounced2 && state == STARTED) state <= FREEZED;
18     else                                      state <= STARTED;
19
20 endmodule
```

碼表有四位數，每一個位數由一個四位元上數計數器組成。下圖是上數計數器。第 10 到 14 行用來判斷下一個時脈過後要變成什麼數值。如果沒有 carry in (c_in)，就不動。否則如果當下的數值小於最大允許數值，就加一。其餘的狀況就重置為零。第 16 行判斷要產生進位 carry out (c_out)的條件，條件為 carry in (c_in) (代表要加一)且當下數值已經到達最大允許數值。

```
1  module upcounter(
2    output reg [3:0] value,
3    output           c_out,
4    input      [3:0] limit,
5    input            c_in,
6    input            clk,
7    input            rst_n
8  );
9
10   always @ (posedge clk or negedge rst_n)
11     if (~rst_n)            value <= 0;
12     else if (~c_in)        value <= value;
13     else if (value < limit) value <= value + 1;
14     else                   value <= 0;
15
16   assign c_out = c_in && (value == limit);
17
18 endmodule
```

下圖是碼表 stopwatch 的接線方法。只要把四個上數計數器和四個解碼器接好即可。如下，把計數器的 carry in (c_in)和 carry out (c_out)兩兩相接。

```
module stopwatch(
// output [15:0] debug,
  output [7:0] ssd4,
  output [7:0] ssd3,
  output [7:0] ssd2,
  output [7:0] ssd1,
  input        en,
  input        f_1hz,
  input        rst_n
);
  wire [3:0] bcd4;
  wire [3:0] bcd3;
  wire [3:0] bcd2;
  wire [3:0] bcd1;

// assign debug = {bcd4, bcd3, bcd2, bcd1};

  upcounter DIGIT4(
    .value(bcd4),
    .c_out(),
    .limit(5),
    .c_in (carry3),
    .clk  (f_1hz),
    .rst_n(rst_n)
  );
  upcounter DIGIT3(
    .value(bcd3),
    .c_out(carry3),
    .limit(9),
    .c_in (carry2),
    .clk  (f_1hz),
    .rst_n(rst_n)
  );
  upcounter DIGIT2(
    .value(bcd2),
    .c_out(carry2),
    .limit(5),
    .c_in (carry1),
    .clk  (f_1hz),
    .rst_n(rst_n)
  );
  upcounter DIGIT1(
    .value(bcd1),
    .c_out(carry1),
    .limit(9),
    .c_in (en),
    .clk  (f_1hz),
    .rst_n(rst_n)
  );

  decoder_BCD_to_SSD DECODER4(
    .ssd(ssd4),
    .bcd(bcd4)
  );
  decoder_BCD_to_SSD DECODER3(
    .ssd(ssd3),
    .bcd(bcd3)
  );
  decoder_BCD_to_SSD DECODER2(
    .ssd(ssd2),
    .bcd(bcd2)
  );
  decoder_BCD_to_SSD DECODER1(
    .ssd(ssd1),
    .bcd(bcd1)
  );

endmodule
```

用來控制七段顯示器的模組如下，跟前幾次 Lab 的差不多。在此，將 freezed 作為 input，可以在第 17 到 23 行使用一些 D latch 來凍結顯示器。

```
1  module SSD_controller(
2    output reg [7:0] cathode,
3    output reg [3:0] anode,
4    input      [7:0] ssd1,
5    input      [7:0] ssd2,
6    input      [7:0] ssd3,
7    input      [7:0] ssd4,
8    input            freezed,
9    input      [1:0] clk
10 );
11   reg [7:0] ssd4_q;
12   reg [7:0] ssd3_q;
13   reg [7:0] ssd2_q;
14   reg [7:0] ssd1_q;
15
16   // Inferring D Latches
17   always @ *
18     if (~freezed) begin
19       ssd4_q <= ssd4;
20       ssd3_q <= ssd3;
21       ssd2_q <= ssd2;
22       ssd1_q <= ssd1;
23     end
```

```
25   //MUX
26   always @ *
27     case (clk)
28       0:       anode <= 4'b1110;
29       1:       anode <= 4'b1101;
30       2:       anode <= 4'b1011;
31       3:       anode <= 4'b0111;
32       default: anode <= 4'b0000;
33     endcase
34
35   //MUX
36   always @ *
37     case (clk)
38       0:       cathode <= ssd1_q;
39       1:       cathode <= ssd2_q;
40       2:       cathode <= ssd3_q;
41       3:       cathode <= ssd4_q;
42       default: cathode <= 0;
43     endcase
44
45 endmodule
```

下圖是 top，把剛剛建構好的那些模組接起來即可。第 75 到 77 行用來判斷按鈕長按，如果連續按下超過數個時脈，就發出重設訊號。除了 stopwatch 以一赫茲為時脈之外，其餘模組運作的時脈以除頻器所產生的的 f_counter[18]為主，頻率為 10^9/2^19，約 1907 赫茲。

```
1  module LAB6_1(
2  // output [15:0] debug,
3    output [7:0] cathode,
4    output [3:0] anode,
5    input        push1,
6    input        push2,
7    input        f_100mhz,
8    input        rst_n
9  );
10 // wire [15:0] debugSW;
11   wire        debounced1;
12   wire        debounced2;
13   wire        f_1hz;
14   wire [25:0] f_counter;
15   wire [ 7:0] ssd4;
16   wire [ 7:0] ssd3;
17   wire [ 7:0] ssd2;
18   wire [ 7:0] ssd1;
19   wire        paused;
20   wire        freezed;
21   reg  [ 1:0] delay;
22
23 // assign debug = {debugSW, debounced1, debounced2, p
24
25   debounce DEBOUNCED1(
26     .debounced(debounced1),
27     .push     (push1),
28     .clk      (f_counter[18]),
29     .rst_n    (rst_n)
30   );
31   debounce DEBOUNCED2(
32     .debounced(debounced2),
33     .push     (push2),
34     .clk      (f_counter[18]),
35     .rst_n    (rst_n)
36   );
37
38   FSM FSM(
39     .paused    (paused),
40     .freezed   (freezed),
41     .debounced1(debounced1),
42     .debounced2(debounced2),
43     .rst_n     (rst_n & ~&{delay, debounced2})
44   );
45
46   frequency_divider FD(
47     .f_1hz    (f_1hz),
48     .f_counter(f_counter),
49     .f_100mhz (f_100mhz),
50     .rst_n    (rst_n)
51   );
52
53   stopwatch SW(
54 //   .debug(debugSW),
55     .ssd4 (ssd4),
56     .ssd3 (ssd3),
57     .ssd2 (ssd2),
58     .ssd1 (ssd1),
59     .en   (!paused),
60     .f_1hz(f_1hz),
61     .rst_n(rst_n & ~&{delay, debounced2})
62   );
63
64   SSD_controller CTRL(
65     .cathode(cathode),
66     .anode  (anode),
67     .ssd4   (ssd4),
68     .ssd3   (ssd3),
69     .ssd2   (ssd2),
70     .ssd1   (ssd1),
71     .freezed(freezed),
72     .clk    (f_counter[20:19])
73   );
74
75   always @ (posedge f_1hz or negedge push2)
76     if (~push2) delay <= 0;
77     else        delay <= {delay, debounced2};
78
79 endmodule
```

5. Conclusion

內容: 可以寫下你的這個 lab 的想法、遇到的問題、解決方法、心得等等，請自由發揮。

這一題跟 LAB4 很類似，所以建構的過程很順利，沒有遇到困難。然而，我在建構這題的模組時，有許多 flip flop 的 clk 是 async 的，這使得我難以將本題的模組直接沿用在下一題。即使成功套用了，也會出一些莫名其妙的 bug，很難 debug。

2 Implement a timer (can support as long as 23:59) with the following functions.
2.1 Use one DIP switch as the 'setting' control. When the 'setting' is ON, you can use two buttons to set the minute and second.
2.2 Use other two buttons to control the timer operation. One button for start/stop and the other button for pause/resume.
2.3 When the time goes to 0, light up all the LEDs.

1. Specification
內容: 寫下你的電路中的 inputs, outputs 以及其 bit widths，名稱必須跟你的 verilog code 中相同。

```
module LAB6_2_top(
  output [ 7:0] cathode,    // 控制顯示器
  output [ 3:0] anode,      // 控制顯示器
  output [15:0] led,        // 控制 LED 的明滅

  input         button1,    // 按一下會開始倒數或增加一小時
  input         button2,    // 按一下會暫停/繼續倒數或增加一分鐘
  input         setting,    // 切換模式

  input         rst_n,      // 強制重置
  input         clk_100mhz // 內建於 FPGA 的時脈
);

module frequency_divider(
  output reg [26:0] clk_g,      // 提供各種頻率給其餘模組使用
  input             clk_100mhz, // 內建於 FPGA 的時脈
  input             rst_n       // 強制重置
);

module debounce(
  output reg debounced, // 經處理後的訊號
  input      push,      // 未經處理的訊號
  input      clk,       // 時脈
  input      rst_n      // 重置
);
```

```
module onepulse(
  output reg onepulse,  // 經處理後的脈衝
  input      debounced, // 去除了繞動後的訊號
  input      clk,       // 時脈
  input      rst_n      // 重置
);

module debounce_onepulse(
  output onepulse,   // 經處理後的脈衝
  output debounced,  // 經處理後的訊號
  input  push,       // 未經處理的、帶有雜訊的訊號
  input  clk,        // 時脈
  input  rst_n       // 重置
);

module fsm(
  output started,           // 要開始數
  input  onepulse_button1,  // 要開始數或加一小時
  input  onepulse_button2,  // 要暫停/繼續或加一分鐘
  input  onepulse_button2_longpressed,   // 要重置
  input  debounced_setting, // 用來判斷現在是否處於設定模式
  input  rst_n              // 強制重置
);

module up_counter(
  output reg [3:0] value,      // 顯示的數值
  output           carryout,   // 需要進位
  input            carryin,    // 需要加一

  input      [3:0] init,       // 初始數值
  input      [3:0] limit,      // 最大允許數值

  input            clk,        // 時脈
  input            enable,     // 允許加一
  input            rst_n       // 重設或初始化
);
```

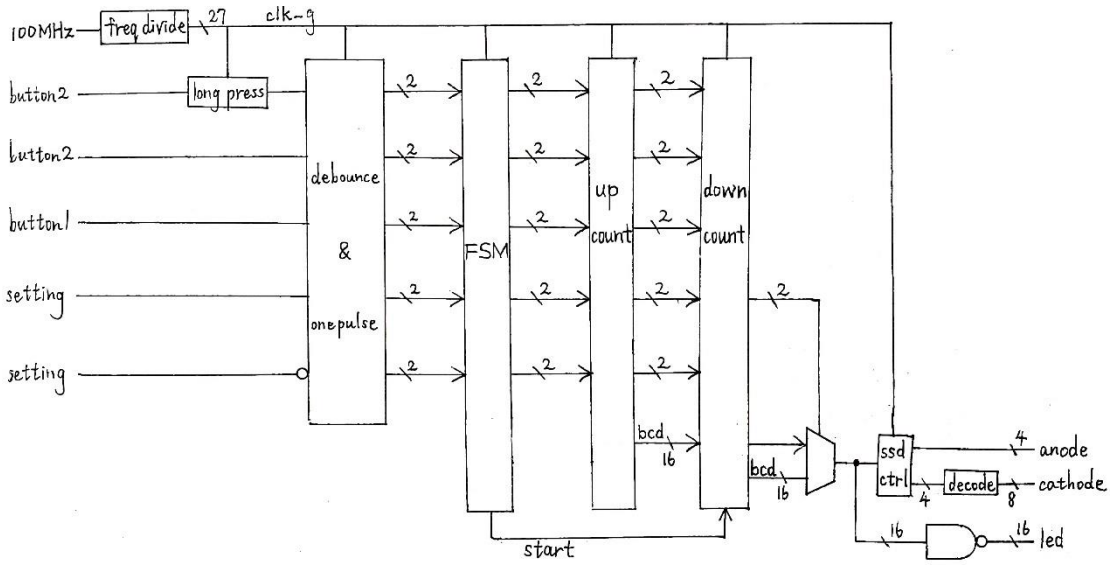
```
module down_counter(
  output reg [3:0] value,      // 顯示的數值
  output           carryout,   // 需要借位
  input            carryin,    // 需要減一

  input      [3:0] init,       // 初始數值
  input      [3:0] limit,      // 最大允許數值

  input            clk,        // 時脈
  input            enable,     // 允許減一
  input            rst_n       // 重設或初始化
);

module minute_up_counter(
  output [3:0] bcd0,        // 分鐘的個位數的數值
  output [3:0] bcd1,        // 分鐘的十位數的數值

  output       carryout,    // 要進位一小時
  input        carryin,     // 要加一分鐘

  input  [3:0] init0,       // 分鐘的個位數的初始數值
  input  [3:0] init1,       // 分鐘的十位數的初始數值

  input        clk,         // 時脈
  input        enable,      // 允許加一分鐘
  input        rst_n        // 重置或初始化
);
```

```
module hour_up_counter(
  output [3:0] bcd0,        // 小時的個位數的數值
  output [3:0] bcd1,        // 小時的十位數的數值

  output       carryout,    // 要進位一天
  input        carryin,     // 要加一小時

  input  [3:0] init0,       // 小時的個位數的初始數值
  input  [3:0] init1,       // 小時的十位數的初始數值

  input        clk,         // 時脈
  input        enable,      // 允許加一小時
  input        rst_n        // 重置或初始化
);

module minute_down_counter(
  output [3:0] bcd0,        // 分鐘的個位數的數值
  output [3:0] bcd1,        // 分鐘的十位數的數值

  output       carryout,    // 要借位一小時
  input        carryin,     // 要減一分鐘

  input  [3:0] init0,       // 分鐘的個位數的初始數值
  input  [3:0] init1,       // 分鐘的十位數的初始數值

  input        clk,         // 時脈
  input        enable,      // 允許減一分鐘
  input        rst_n        // 重置或初始化
);
```

```
module hour_down_counter(
  output [3:0] bcd0,        // 小時的個位數的數值
  output [3:0] bcd1,        // 小時的十位數的數值

  output       carryout,    // 要借位一天
  input        carryin,     // 要減一小時

  input  [3:0] init0,       // 小時的個位數的初始數值
  input  [3:0] init1,       // 小時的十位數的初始數值

  input        clk,         // 時脈
  input        enable,      // 允許減一小時
  input        rst_n        // 重置或初始化
);

module ssd_control(
  output     [7:0] cathode,    // 控制顯示器
  output reg [3:0] anode,      // 控制顯示器
  input      [3:0] bcd0,       // 分鐘的個位數的數值
  input      [3:0] bcd1,       // 分鐘的十位數的數值
  input      [3:0] bcd2,       // 小時的個位數的數值
  input      [3:0] bcd3,       // 小時的十位數的數值
  input            freezed,    // 凍結顯示器 (本題沒用到)
  input      [1:0] clk         // 時脈
);
```

2. Block Diagram

內容: 電路中的 **Block diagram**(可以用手畫拍照或電腦繪圖)。



3. Finite state machine

內容: 電路中的 Finite state machine，若無則寫無。

只有兩個狀態，正在下數或不動。有四個輸入，分別代表「第一個按鈕被按下」、「第二個按鈕被按下」、「第二個按鈕被持續按下超過數個時脈」、「正處於設定模式」。此 FSM 有一個輸出，用來控制下數計數器是否要以一赫茲為頻率減一分鐘。由於這是 moore machine，而且狀態跟輸出之間的關係很單純，所以可以把狀態直接與輸出連接起來，如第 15 行。當狀態是不動(PAUSED)時，計數器不要下數，所以是 1'b0，如第 12 行。當狀態是正在下數(STARTED)時，計數器要下數，所以是 1'b1，如第 13 行。只有當輸入有改變時，才需要更新 FSM 當前的狀態，所以將時脈與輸入連接，如第 23 到 26 行所示。而第 17 到 21 行會依照當下的輸入與狀態來判定下一個狀態。若第一個按鈕被按下，則下一個狀態為 STARTED。如果第二個按鈕被按下，且當下狀態是 PAUSED，則下一個狀態也要變成 STARTED。其餘情況都會使下一個狀態變成 PAUSED。

```
1  module fsm(
2    output started,
3    input  onepulse_button1,
4    input  onepulse_button2,
5    input  onepulse_button2_longpressed,
6    input  debounced_setting,
7    input  rst_n
8  );
9    reg  [1:0] state;
10   wire [1:0] state_tmp;
11   wire       clk;
12   parameter PAUSED  = 1'b0;
13   parameter STARTED = 1'b1;
14
15   assign started = state;
16
17   assign state_tmp = (onepulse_button1)                  ? (STARTED) :
18                      (onepulse_button2 & state == PAUSED ) ? (STARTED) :
19                      (onepulse_button2 & state == STARTED) ? (PAUSED)  :
20                      (onepulse_button2_longpressed)        ? (PAUSED)  :
21                      (debounced_setting)                   ? (PAUSED)  : (PAUSED);
22
23   assign clk = onepulse_button1
24              | onepulse_button2
25              | onepulse_button2_longpressed
26              | debounced_setting;
27
28   always @ (posedge clk or negedge rst_n)
29     if   (~rst_n) state <= PAUSED;
30     else          state <= state_tmp;
31
32 endmodule
```

4. Implement

內容: 請列出相關的 logic function、詳細用文字解釋電路的運作方法、結果等等，可以貼 code 解釋或拍 FPGA 輔助解釋(但不能只貼 code 跟 FPGA 結果)。

下圖是除頻器。把 FPGA 內建的 100M Hz 連接到一個 27 位元計數器，從零開始數，最高數到 100M - 1，並直接把計數器的數值當作輸出。如果有其他模組需要一赫茲的時脈，只要直接接入 clk_g[26]即可。其他位元就當作各種不同頻率的時脈，讓其餘有特殊需求的模組各取所需。

```
module frequency_divider(
  output reg [26:0] clk_g,
  input             clk_100mhz,
  input             rst_n
);
  reg [26:0] clk_g_tmp;

always @*
  if (clk_g == 99_999_999) clk_g_tmp <= 0;
  else                     clk_g_tmp <= clk_g + 1;

always @ (posedge clk_100mhz or negedge rst_n)
  if (~rst_n) clk_g <= 0;
  else        clk_g <= clk_g_tmp;

endmodule
```

下圖可以穩定訊號，用來消除按鈕的震動。原理跟上一題一樣。

```
module debounce(
  output reg debounced,
  input      push,
  input      clk,
  input      rst_n
);
  reg  [3:0] window;
  wire       debounced_tmp;

always @ (posedge clk or negedge rst_n)
  if (~rst_n) window <= 0;
  else        window <= {window, push};

assign debounced_tmp = (window == 4'b1111);

always @ (posedge clk or negedge rst_n)
  if (~rst_n) debounced <= 1'b0;
  else        debounced <= debounced_tmp;

endmodule
```

下圖是用來產生脈衝的。如此一來，模組產生訊號給有限狀態機後，有限狀態機比較能夠輕鬆而不易出錯地更新到下一個狀態。其主要目的是為了讓有限狀態機內運作的頻率跟外部其他模組盡可能地同步。

```
module onepulse(
  output reg onepulse,
  input      debounced,
  input      clk,
  input      rst_n
);
  reg delay;

always @ (posedge clk or negedge rst_n)
  if (~rst_n) delay <= 0;
  else        delay <= debounced;

assign onepulse_tmp = (debounced & ~delay);

always @ (posedge clk or negedge rst_n)
  if (~rst_n) onepulse <= 1'b0;
  else        onepulse <= onepulse_tmp;

endmodule
```

下圖只是把剛剛那兩個模組包裝在一起而已。只要把一個訊號連接到這個模組，這個模組就會產生去除按鈕跳動等雜訊後的乾淨訊號，以及時長為一個時脈的脈衝。

```
1  module debounce_onepulse(
2    output onepulse,
3    output debounced,
4    input  push,
5    input  clk,
6    input  rst_n
7  );

9    debounce DEBOUNCED(
10     .debounced(debounced),
11     .push     (push),
12     .clk      (clk),
13     .rst_n    (rst_n)
14   );

16   onepulse ONEPULSE(
17     .onepulse (onepulse),
18     .debounced(debounced),
19     .clk      (clk),
20     .rst_n    (rst_n)
21   );

23 endmodule
```

以下兩張圖片分別是四位元上數計數器和四位元下數計數器，結構相同。第 18 到 21 行判斷下一個時脈過後的數值應變為多少，然後第 28 到第 30 行在時脈來的時候更新數值。而第 23 到 26 行則是判斷使否要輸出進位的訊號。

```
1  module up_counter(
2    output reg [3:0] value,
3    output           carryout,
4    input            carryin,
5
6    input      [3:0] init,
7    input      [3:0] limit,
8
9    input            clk,
10   input            enable,
11   input            rst_n
12 );
13   wire [3:0] value_tmp;
14
15   localparam TRUE  = 1'b1;
16   localparam FALSE = 1'b0;
17
18   assign value_tmp = (~enable)       ? (value)     :
19                      (~carryin)      ? (value)     :
20                      (value <  limit) ? (value + 1) :
21                      (value == limit) ? (0)         : (0);
22
23   assign carryout  = (~enable)       ? (FALSE)     :
24                      (~carryin)      ? (FALSE)     :
25                      (value <  limit) ? (FALSE)     :
26                      (value == limit) ? (TRUE )     : (FALSE);
27
28   always @ (posedge clk or negedge rst_n)
29     if (~rst_n) value    <= init;
30     else        value    <= value_tmp;
31
32 endmodule
```

四位元上數計數器和四位元下數計數器只有一點點不同，只要使用文字編輯器將所有圓括號內的「limit」與「0」互換，再將「+」改成「-」，即為四位元下數計數器。

下圖是兩位數的上數計數器，負責控制分鐘的個位數字和分鐘的十位數字。分鐘的個位數字最高到 9，如第 22 行。分鐘的十位數字最高到 5，如第 33 行。把個位數和十位數之間的進位訊號接好，如第 19、31 行。其他接線直接與 input 或 output 連接即可。

```
1  module minute_up_counter(
2    output [3:0] bcd0,
3    output [3:0] bcd1,
4
5    output       carryout,
6    input        carryin,
7
8    input  [3:0] init0,
9    input  [3:0] init1,
10
11   input        clk,
12   input        enable,
13   input        rst_n
14 );
15   wire carry;
16
17   up_counter up_counter_minute_digit_0(
18     .value   (bcd0),
19     .carryout(carry),
20     .carryin (carryin),
21     .init    (init0),
22     .limit   (4'd9),
23     .clk     (clk),
24     .enable  (enable),
25     .rst_n   (rst_n)
26   );
27
28   up_counter up_counter_minute_digit_1(
29     .value   (bcd1),
30     .carryout(carryout),
31     .carryin (carry),
32     .init    (init1),
33     .limit   (4'd5),
34     .clk     (clk),
35     .enable  (enable),
36     .rst_n   (rst_n)
37   );
38
39 endmodule
```

下圖也是兩位數的上數計數器，但負責控制的是小時的個位數字和小時的十位數字。小時的個位數字的最大允許數值由小時的十位數字決定，如第 18、19、25 行。如果十位數是 2，那個位數的最大允許數值就是 3，否則為 9。而小時的十位數字則最高到 2，如第 36 行。把個位數和十位數之間的進位訊號接好，如第 22、34 行。其他接線直接與 input 或 output 連接即可。

```
1  module hour_up_counter(
2    output [3:0] bcd0,
3    output [3:0] bcd1,
4
5    output       carryout,
6    input        carryin,
7
8    input  [3:0] init0,
9    input  [3:0] init1,
10
11   input        clk,
12   input        enable,
13   input        rst_n
14 );
15   wire       carry;
16   wire [3:0] digit_0_limit;
17
18   assign digit_0_limit =
19       (bcd1 == 2) ? (4'd3) : (4'd9);
20   up_counter up_counter_hour_digit_0(
21     .value   (bcd0),
22     .carryout(carry),
23     .carryin (carryin),
24     .init    (init0),
25     .limit   (digit_0_limit),
26     .clk     (clk),
27     .enable  (enable),
28     .rst_n   (rst_n)
29   );
30
31   up_counter up_counter_hour_digit_1(
32     .value   (bcd1),
33     .carryout(carryout),
34     .carryin (carry),
35     .init    (init1),
36     .limit   (4'd2),
37     .clk     (clk),
38     .enable  (enable),
39     .rst_n   (rst_n)
40   );
41
42 endmodule
```

另外兩個下數計數器跟剛剛提到的兩個上數計數器只有一點點差別。用文字編輯器把所有「up」取代為「down」，即為另外兩個下數計數器。

下圖是控制七段顯示器的模組，跟之前的原理一樣。因為這題不用凍結顯示器，所以實際上只需要那兩個 MUX 就好，不用第 18 到 24 行的 D Latch。其中的 decoder 功能跟上一題一樣，可直接沿用。

```
1  module ssd_control(
2    output     [7:0] cathode,
3    output reg [3:0] anode,
4    input      [3:0] bcd0,
5    input      [3:0] bcd1,
6    input      [3:0] bcd2,
7    input      [3:0] bcd3,
8    input            freezed,
9    input      [1:0] clk
10 );
11   reg  [3:0] bcd0_q;
12   reg  [3:0] bcd1_q;
13   reg  [3:0] bcd2_q;
14   reg  [3:0] bcd3_q;
15   reg  [3:0] bcd;
16
17   // Inferring D Latches
18   always @*
19     if (~freezed) begin
20       bcd0_q <= bcd0;
21       bcd1_q <= bcd1;
22       bcd2_q <= bcd2;
23       bcd3_q <= bcd3;
24     end
27   always @*
28     case (clk)
29       0:       anode <= 4'b1110;
30       1:       anode <= 4'b1101;
31       2:       anode <= 4'b1011;
32       3:       anode <= 4'b0111;
33       default: anode <= 4'b0000;
34     endcase
35
36   //MUX
37   always @*
38     case (clk)
39       0:       bcd <= bcd0_q;
40       1:       bcd <= bcd1_q;
41       2:       bcd <= bcd2_q;
42       3:       bcd <= bcd3_q;
43       default: bcd <= 0;
44     endcase
45
46   decoder DECODER(
47     .ssd(cathode),
48     .bcd(bcd)
49   );
50
51 endmodule
```

接下來只要把上述模組連接起來就好。LAB6_2_top 的線路名稱如下圖。

```
module LAB6_2_top(
  output [ 7:0] cathode,
  output [ 3:0] anode,
  output [15:0] led,

  input         button1,
  input         button2,
  input         setting,

  input         rst_n,
  input         clk_100mhz
);
  wire [26:0] clk_g;
  wire        reset;

  reg  [ 7:0] window_button2;
  wire        button2_longpressed;

  wire        debounced_button1;
  wire        debounced_button2;
  wire        debounced_button2_longpressed;
  wire        debounced_setting;
  wire        debounced_setting_n;

  wire        onepulse_button1;
  wire        onepulse_button2;
  wire        onepulse_button2_longpressed;
  wire        onepulse_setting;
  wire        onepulse_setting_n;

  wire        started;

  wire [ 3:0] minute_up_counter_bcd0;
  wire [ 3:0] minute_up_counter_bcd1;
  wire [ 3:0]   hour_up_counter_bcd0;
  wire [ 3:0]   hour_up_counter_bcd1;
  wire [ 3:0] minute_down_counter_bcd0;
  wire [ 3:0] minute_down_counter_bcd1;
  wire [ 3:0]   hour_down_counter_bcd0;
  wire [ 3:0]   hour_down_counter_bcd1;

  wire        minute_down_counter_carryout;
  wire        is_all_zero;

  reg  [ 3:0] bcd0;
  reg  [ 3:0] bcd1;
  reg  [ 3:0] bcd2;
  reg  [ 3:0] bcd3;
```

先用除頻器製造出所有模組可能會用到的時脈。

```
50  frequency_divider frequency_divider(
51    .clk_g     (clk_g),
52    .clk_100mhz(clk_100mhz),
53    .rst_n     (rst_n)
54  );
```

處理所有 input，把彈跳過濾掉，並產生脈衝訊號。這些模組的時脈仿照上一題，使用 clk_g[18]，頻率為 10^9/2^19，大約 1907 赫茲。

```
56  debounce_onepulse debounce_onepulse_button1(
57    .onepulse (onepulse_button1),
58    .debounced(debounced_button1),
59    .push     (button1),
60    .clk      (clk_g[18]),
61    .rst_n    (rst_n)
62  );
63
64  debounce_onepulse debounce_onepulse_button2(
65    .onepulse (onepulse_button2),
66    .debounced(debounced_button2),
67    .push     (button2),
68    .clk      (clk_g[18]),
69    .rst_n    (rst_n)
70  );
71
72  debounce_onepulse debounce_onepulse_setting(
73    .onepulse (onepulse_setting),
74    .debounced(debounced_setting),
75    .push     (setting),
76    .clk      (clk_g[18]),
77    .rst_n    (rst_n)
78  );
79
80  debounce_onepulse debounce_onepulse_setting_n(
81    .onepulse (onepulse_setting_n),
82    .debounced(debounced_setting_n),
83    .push     (~setting),
84    .clk      (clk_g[18]),
85    .rst_n    (rst_n)
86  );
```

為了偵測按鈕長按，使用一些 shifter 來判定按鈕是否被按下超過連續數個時脈。並且也將長按的訊號處理成脈衝訊號。

```
88  always @ (posedge clk_g[23] or negedge debounced_button2)
89    if (~debounced_button2) window_button2 <= 0;
90    else                    window_button2 <= {window_button2, debounced_button2};
91
92  assign button2_longpressed = (window_button2 == 8'b1111_1111);
93
94  debounce_onepulse debounce_onepulse_button2_longpressed(
95    .onepulse (onepulse_button2_longpressed),
96    .debounced(debounced_button2_longpressed),
97    .push     (button2_longpressed),
98    .clk      (clk_g[18]),
99    .rst_n    (rst_n)
100 );
```

要 reset 的情形包含：強制重置、第二個按鈕被按下過久、或切換設定模式時。

```
102 assign reset = ~rst_n
103              | onepulse_button2_longpressed
104              | onepulse_setting
105              | onepulse_setting_n;
```

把經過處理的 input 連接到有限狀態機。輸出 started 用來控制計數器是否下數。

```
107 fsm fsm(
108   .started                    (started),
109   .onepulse_button1           (onepulse_button1),
110   .onepulse_button2           (onepulse_button2),
111   .onepulse_button2_longpressed(onepulse_button2_longpressed),
112   .debounced_setting          (debounced_setting),
113   .rst_n                      (~reset)
114 );
```

把所有計數器的接好。下數計數器的分鐘要向小時傳送借位訊號，如第 145、158 行。第一個按鈕被按下時，上數計數器的小時數值會加一，如第 135 行。第二個按鈕被按下時，上數計數器的分鐘數值會加一，如第 123 行。而因為 clk_g[26]的頻率為一赫茲，故適合作為下數計數器的時脈，如第 149、161 行。其中，上數計數器只有在設定模式下允許數值加一，如第 124、136 行。下數計數器則只有在有限狀態輸出 started 時才允許下數，如第 150、162 行。且上數計數器的數值連接到下數計數器的初始數值，如此一來，當下數計數器收到重置訊號時，就會把數值設定成上數計數器的數值，如第 147、148、159、160 行。

```
116   minute_up_counter minute_up_counter(
117     .bcd0    (minute_up_counter_bcd0),
118     .bcd1    (minute_up_counter_bcd1),
119     .carryout(),
120     .carryin (1'b1),
121     .init0   (minute_up_counter_bcd0),
122     .init1   (minute_up_counter_bcd1),
123     .clk     (onepulse_button2),
124     .enable  (debounced_setting),
125     .rst_n   (~reset)
126   );
127
128   hour_up_counter hour_up_counter(
129     .bcd0    (hour_up_counter_bcd0),
130     .bcd1    (hour_up_counter_bcd1),
131     .carryout(),
132     .carryin (1'b1),
133     .init0   (hour_up_counter_bcd0),
134     .init1   (hour_up_counter_bcd1),
135     .clk     (onepulse_button1),
136     .enable  (debounced_setting),
137     .rst_n   (~reset)
138   );
```

```
142   minute_down_counter minute_down_counter(
143     .bcd0    (minute_down_counter_bcd0),
144     .bcd1    (minute_down_counter_bcd1),
145     .carryout(minute_down_counter_carryout),
146     .carryin (~is_all_zero),
147     .init0   (minute_up_counter_bcd0),
148     .init1   (minute_up_counter_bcd1),
149     .clk     (clk_g[26]),
150     .enable  (started),
151     .rst_n   (~reset)
152   );
153
154   hour_down_counter hour_down_counter(
155     .bcd0    (hour_down_counter_bcd0),
156     .bcd1    (hour_down_counter_bcd1),
157     .carryout(),
158     .carryin (minute_down_counter_carryout),
159     .init0   (hour_up_counter_bcd0),
160     .init1   (hour_up_counter_bcd1),
161     .clk     (clk_g[26]),
162     .enable  (started),
163     .rst_n   (~reset)
164   );
```

另外，為了避免下數計數器數到零之後還繼續下數，必須考慮數值是否已經全部為零，如果還沒數到零，才允許繼續下數。因此，將以下的判斷條件連接到下數計數器的 carry_in，如第 146 行。

```
140   assign is_all_zero = {bcd3, bcd2, bcd1, bcd0} == 16'b0;
```

至此，計數器都已經配置完成。接下來只要建構好顯示器的部分即可。將剛剛那些計數器的數值連接到 MUX，當處於設定狀態下時，把上數計數器的數值連接到顯示器控制器，反之，把下數計數器的數值連接到顯示器控制器。

```
166   always @ *
167     if (setting) begin
168        bcd0 <= minute_up_counter_bcd0;
169        bcd1 <= minute_up_counter_bcd1;
170        bcd2 <=   hour_up_counter_bcd0;
171        bcd3 <=   hour_up_counter_bcd1;
172     end else begin
173        bcd0 <= minute_down_counter_bcd0;
174        bcd1 <= minute_down_counter_bcd1;
175        bcd2 <=   hour_down_counter_bcd0;
176        bcd3 <=   hour_down_counter_bcd1;
177     end
```

最後連接顯示器控制器，大功告成。

```
179   ssd_control ssd_control(
180     .cathode(cathode),
181     .anode  (anode),
182     .bcd0   (bcd0),
183     .bcd1   (bcd1),
184     .bcd2   (bcd2),
185     .bcd3   (bcd3),
186     .freezed(1'b0),
187     .clk    (clk_g[20:19])
188   );
189
190 // assign led = {bcd3[1:0], bcd2[1:0], bcd.
191   assign led = {16{is_all_zero}};
192
193 endmodule
```

5. Conclusion

內容: 可以寫下你的這個 lab 的想法、遇到的問題、解決方法、心得等等，請自由發揮。

這一題花了我較多的時間，我捨棄上一題的作法，並盡量讓所有訊號都與 clk_g[18]同步，以避免各個計數器抓不到進位或借位時機的問題。我一開始試著沿用上一題的模組，但後來發現上一題的模組的性質不夠好，即使我把重寫部分的 counter，整合上數和下數的功能，在我硬著頭皮把所有 top 的接線連接起來之後，仍然造成了如下窘境：

```
57    counter DIGIT4(
58      .value   (bcd4),
59      .o_carry (),
60      .o_borrow(),
61      .limit   (2),
62      .i_carry (carry3),
63      .i_borrow(borrow3),
64      .plus    ( setting & debounced1),
65      .minus   (~setting & f_1hz & ~paused & |{bcd4, bcd3, bcd2, bcd1}),
66      .rst_n   ( rst_n & ~&{delay, debounced2} & ~(bcd4 == 2 & bcd3 == 4) )
67    );
68    counter DIGIT3(
69      .value   (bcd3),
70      .o_carry (carry3),
71      .o_borrow(borrow3),
72      .limit   (9),
73      .i_carry ( setting),
74      .i_borrow(borrow2),
75      .plus    ( setting & debounced1),
76      .minus   (~setting & f_1hz & ~paused & |{bcd4, bcd3, bcd2, bcd1}),
77      .rst_n   ( rst_n & ~&{delay, debounced2} & ~(bcd4 == 2 & bcd3 == 4) )
78    );
79    counter DIGIT2(
80      .value   (bcd2),
81      .o_carry (),
82      .o_borrow(borrow2),
83      .limit   (5),
84      .i_carry (carry1),
85      .i_borrow(borrow1),
86      .plus    ( setting & debounced2),
87      .minus   (~setting & f_1hz & ~paused & |{bcd4, bcd3, bcd2, bcd1}),
88      .rst_n   (rst_n & ~&{delay, debounced2})
89    );
90    counter DIGIT1(
91      .value   (bcd1),
92      .o_carry (carry1),
93      .o_borrow(borrow1),
94      .limit   (9),
95      .i_carry ( setting),
96      .i_borrow(~setting & ~paused),
97      .plus    ( setting & debounced2),
98      .minus   (~setting & f_1hz & ~paused & |{bcd4, bcd3, bcd2, bcd1}),
99      .rst_n   (rst_n & ~&{delay, debounced2})
00    );
```

我花了一個下午試著添加更多判斷條件在計數器的接口上（當然，這樣是行不通的），最後還是無法成功 debug，反而還讓我的 bug 更難找了。