1 Finish the time display function supporting 24-hour (00-23).

1.1 Can display as hour:minute and second, and use a push button or DIP switch to switch the display.

1.2 Support two modes: AM/PM and 24-hour

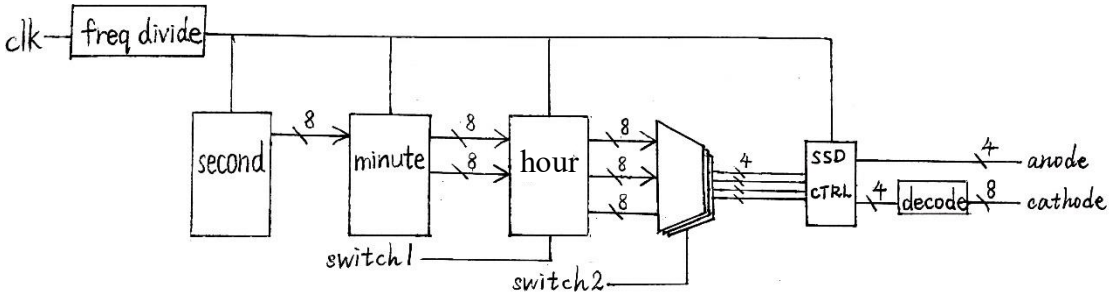
1. Specification

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 LAB7_1_top(
  output [7:0] cathode,                  // 七段顯示器
  output [3:0] anode,                    // 七段顯示器

  input add_second_undebounced,          // 按一下會加一秒
  input add_minute_undebounced,          // 按一下會加一分鐘
  input add_hour___undebounced,          // 按一下會加一小時

  input switch_setting,                  // 切換設定時間
  input switch_hour_minute_second,       // 切換顯示小時、分鐘或秒
  input switch_12_24,                    // 切換顯示 12/24 小時制
  input [3:0] speedup,                   // demo 時可以加速

  input clk_100mhz,                      // FPGA 內建的時脈
  input rst_n                            // 強制重置、初始化
);
```

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。

無

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)。

先用一個 27 位元的計數器構造一個除頻器。可製造出各種不同頻率的時脈，最慢的是一赫茲。如下圖，跟上次 Lab 的差不多。第 8 到 12 行是計數器。直接將除頻器的 clk_g 當作 output，讓 top 的其他模組可以參照各種頻率。這個除頻器裡的計數器每數到 100M 就會重置，所以如果有其他模組有需要使用到一赫茲的時脈的話，可以連接這個除頻器的計數器的最高位數 clk_g[26]。

```
1  module frequency_divider(
2    output reg [26:0] clk_g,
3    input             clk_100mhz,
4    input             rst_n
5  );
6    wire [26:0] clk_g_tmp;
7
8    assign clk_g_tmp = (clk_g < 27'd99_999_999) ? (clk_g + 1'b1) : ('b0);
9
10   always @(posedge clk_100mhz or negedge rst_n)
11     if (~rst_n) clk_g <= 'b0;
12     else        clk_g <= clk_g_tmp;
13
14 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  undebounced,
4    input  clk,
5    input  rst_n
6  );
7    reg [3:0] undebounced_window;
8
9    always @(posedge clk or negedge rst_n)
10     if (~rst_n) undebounced_window <= 'b0;
11     else        undebounced_window <= {undebounced_window, undebounced};
12
13   assign debounced = (undebounced_window == 4'b1111);
14
15 endmodule
```

接下來是用來計時的四位元計數器。跟之前 LAB 的做法很像，在 clk 的 rising edge 的時候，利用 carry_in 來判斷是否需要加一。如果要加一，再利用 upper_bound 來判斷是否要進位。

```
1  module counter(
2    output reg [3:0] value,
3
4    output carry_out,
5    input  carry_in,
6
7    input [3:0] upper_bound,
8
9    input clk,
10   input rst_n
11 );
12   wire [3:0] next_value;
```

第 14 到 16 行是下一個數值的判定。第 18 到 20 行是 D Flip-flop，用來更新數值。第 22 行判定是否需要通知上一位進位。

```
14   assign next_value = (~carry_in)             ? (value)         :
15                       (value <  upper_bound) ? (value + 1'b1) :
16                       (value == upper_bound) ? ('b0)           : (4'hF);
17
18   always @ (posedge clk or negedge rst_n)
19     if (~rst_n) value <= 'b0;
20     else        value <= next_value;
21
22   assign carry_out = (carry_in & value == upper_bound);
23
24 endmodule
```

這樣的一個四位元計數器可以 BCD 表示一個十進位數字，如果要表示小時、分鐘、秒，就必須要有六個四位元計數器。

可以把兩個四位元計數器包裝在同一個模組裡面，因為小時、分鐘、秒鐘都各由兩個十位數組成。把個位數的 carry_out 連接到十位數的 carry_in，這樣個位數要進位時，十位數就會加一。

```
1  module two_digits(
2    output [3:0] ones_value,
3    output [3:0] tens_value,
4
5    output carry_out,
6    input  carry_in,
7
8    input [3:0] ones_upper_bound,
9    input [3:0] tens_upper_bound,
10
11   input clk,
12   input rst_n
13 );
14   wire carry_ones_tens;
```

```
16   counter ones_counter(
17     .value       (ones_value),
18     .carry_out   (carry_ones_tens),
19     .carry_in    (carry_in),
20     .upper_bound(ones_upper_bound),
21     .clk         (clk),
22     .rst_n       (rst_n)
23   );
24
25   counter tens_counter(
26     .value       (tens_value),
27     .carry_out   (carry_out),
28     .carry_in    (carry_ones_tens),
29     .upper_bound(tens_upper_bound),
30     .clk         (clk),
31     .rst_n       (rst_n)
32   );
33
34 endmodule
```

小時、分鐘、秒鐘都各自包裝成模組。這樣要使用時，外界只需要向內部連接時脈和進位訊號就好，避免在一個模組裡同時有太多接線。秒鐘的個位數上界是 9，十位數是 5。

```
module second(
  output [3:0] ones_value,
  output [3:0] tens_value,

  output carry_out,
  input  carry_in,

  input clk,
  input rst_n
);

  two_digits two_digit_second(
    .ones_value       (ones_value),
    .tens_value       (tens_value),
    .carry_out        (carry_out),
    .carry_in         (carry_in),
    .ones_upper_bound(4'd9),
    .tens_upper_bound(4'd5),
    .clk              (clk),
    .rst_n            (rst_n)
  );

endmodule
```

分鐘和秒鐘一樣，個位數上界是 9，十位數是 5。

```
module minute(
  output [3:0] ones_value,
  output [3:0] tens_value,

  output carry_out,
  input  carry_in,

  input clk,
  input rst_n
);

  two_digits two_digit_minute(
    .ones_value       (ones_value),
    .tens_value       (tens_value),
    .carry_out        (carry_out),
    .carry_in         (carry_in),
    .ones_upper_bound(4'd9),
    .tens_upper_bound(4'd5),
    .clk              (clk),
    .rst_n            (rst_n)
  );

endmodule
```

小時的個位數判斷較複雜。第 14 到 15 行是小時的各位數上界判定，如果小時的十位數是 0 或 1，那麼個位數上界就是 9，如果十位數是 2，個位數上界為 3。十位數上界則固定為 2。

```
module hour(
  output [3:0] ones_value,
  output [3:0] tens_value,

  output carry_out,
  input  carry_in,

  input clk,
  input rst_n
);
  wire [3:0] ones_upper_bound;
  wire [3:0] tens_upper_bound;

  assign ones_upper_bound = (tens_value == tens_upper_bound)
                          ? (4'd3) : (4'd9);
  assign tens_upper_bound = (4'd2);

  two_digits two_digit_hour(
    .ones_value       (ones_value),
    .tens_value       (tens_value),
    .carry_out        (carry_out),
    .carry_in         (carry_in),
    .ones_upper_bound(ones_upper_bound),
    .tens_upper_bound(tens_upper_bound),
    .clk              (clk),
    .rst_n            (rst_n)
  );

endmodule
```

小時、分鐘、秒鐘的模組都已經建構完畢。接下來就可以把上述三個模組包裝在一起。這個模組的功能主要有二：1.將 carry 的接線接好。2.判定設定時間的 clk 接線。

```
module timer(
  output [3:0] ones_value_second,
  output [3:0] tens_value_second,
  output [3:0] ones_value_minute,
  output [3:0] tens_value_minute,
  output [3:0] ones_value_hour,
  output [3:0] tens_value_hour,

  output carry_out,
  input  carry_in,
  input add_second,
  input add_minute,
  input add_hour,

  input switch_setting,

  input clk,
  input rst_n
);
  wire carry_second_to_minute;
  wire carry_minute_to_hour;
```

收到 switch_setting 的訊號時，要設定時間，為了讓設定時間的過程更順暢，每個 clk 都要連同 switch_setting 一起考慮，阻止一赫茲的時脈與小時、分鐘、秒鐘直接連接，並使得小時、分鐘、秒鐘只有在對應的按鈕被按下時才加一。進位的 carry_in 也要同 switch_setting 一起考慮，在設定模式中，暫時阻止小時和分鐘以及分鐘和秒鐘之間進位，且使進位訊號恆為 1'b1，這樣按鈕被按下時，對應的小時、分鐘、秒鐘才會加一。

```
  second second(
    .ones_value(ones_value_second),
    .tens_value(tens_value_second),
    .carry_out (carry_second_to_minute),
    .carry_in  (carry_in | ~switch_setting),
    .clk       (clk & switch_setting | add_second),
    .rst_n     (rst_n)
  );

  minute minute(
    .ones_value(ones_value_minute),
    .tens_value(tens_value_minute),
    .carry_out (carry_minute_to_hour),
    .carry_in  (carry_second_to_minute | ~switch_setting),
    .clk       (clk & switch_setting | add_minute),
    .rst_n     (rst_n)
  );

  hour hour(
    .ones_value(ones_value_hour),
    .tens_value(tens_value_hour),
    .carry_out (carry_out),
    .carry_in  (carry_minute_to_hour | ~switch_setting),
    .clk       (clk & switch_setting | add_hour),
    .rst_n     (rst_n)
  );
```

把上述的判定訊號與小時、分鐘、秒鐘的模組連接好，這樣 timer 就基本完成了。

```
endmodule
```

至此，我們已處理完計時的部分。只剩下顯示的部分還沒建構。

顯示小時的模式有兩個，一個是二十四小時制（不用轉換），一個是十二小時（超過十二的話要減十二）。

```
1  module switch_hour(
2    output [3:0] ones_value_hour,
3    output [3:0] tens_value_hour,
4    input  [3:0] ones_value_hour_24,
5    input  [3:0] tens_value_hour_24,
6    input switch_12_24
7  );
8    wire [7:0] unconverted;
9    reg  [7:0]   converted;
10
11   assign unconverted = {tens_value_hour_24, ones_value_hour_24};
```

case 不多，直接列出來，用類似解碼器的方法即可。第 30 到 32 行判斷當下是否要轉換。

```
13   always @*
14     case (unconverted)
15       {4'd1, 4'd3}: converted = {4'd0, 4'd1};
16       {4'd1, 4'd4}: converted = {4'd0, 4'd2};
17       {4'd1, 4'd5}: converted = {4'd0, 4'd3};
18       {4'd1, 4'd6}: converted = {4'd0, 4'd4};
19       {4'd1, 4'd7}: converted = {4'd0, 4'd5};
20       {4'd1, 4'd8}: converted = {4'd0, 4'd6};
21       {4'd1, 4'd9}: converted = {4'd0, 4'd7};
22       {4'd2, 4'd0}: converted = {4'd0, 4'd8};
23       {4'd2, 4'd1}: converted = {4'd0, 4'd9};
24       {4'd2, 4'd2}: converted = {4'd1, 4'd0};
25       {4'd2, 4'd3}: converted = {4'd1, 4'd1};
26       {4'd0, 4'd0}: converted = {4'd1, 4'd2};
27       default:      converted = unconverted;
28     endcase
29
30     assign {tens_value_hour, ones_value_hour} = (switch_12_24)
31                                                 ? (unconverted)
32                                                 : (converted);
33 endmodule
```

至此，已經有了所有時間的 BCD 訊號。接下來要把這些 BCD 訊號轉成七段顯示器的樣式，且利用視覺暫留來使七段顯示器看起來可以同時顯示四個不同的數字。其中第 30 到 33 行的 decoder 的內容跟上次的 LAB 一模一樣。

```
1  module ssd_ctrl(
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      [1:0] clk
9  );
10   reg  [3:0] bcd;
11
12   always @*
13     case (clk)
14       2'd0:    anode = 4'b1110;
15       2'd1:    anode = 4'b1101;
16       2'd2:    anode = 4'b1011;
17       2'd3:    anode = 4'b0111;
18       default: anode = 4'b0000;
19     endcase
21   always @*
22     case (clk)
23       2'd0:    bcd = bcd0;
24       2'd1:    bcd = bcd1;
25       2'd2:    bcd = bcd2;
26       2'd3:    bcd = bcd3;
27       default: bcd = 4'b1111;
28     endcase
29
30   decoder decoder(
31     .ssd(cathode),
32     .bcd(bcd)
33   );
34
35 endmodule
```

上述的三個模組（timers、witch_hour、ssd_ctrl）可以包裝在一個模組裡面，避免同一層級內的接線太多。

```
module electronic_clock(
  output [7:0] cathode,
  output [3:0] anode,

  input add_second,
  input add_minute,
  input add_hour,

  input switch_setting,
  input switch_hour_minute_second,
  input switch_12_24,
  input [3:0] speedup,

  input [26:0] clk_g,
  input        rst_n
);
```

```
  reg clk_timer;

  wire [3:0] ones_value_second;
  wire [3:0] tens_value_second;
  wire [3:0] ones_value_minute;
  wire [3:0] tens_value_minute;
  wire [3:0] ones_value_hour_24;
  wire [3:0] tens_value_hour_24;
  wire [3:0] ones_value_hour;
  wire [3:0] tens_value_hour;

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

為了 demo 方便，我設置了一些加速的選項。平時連接到 timer 的時脈是一赫茲的，如果要加速的話，只要使用 MUX 改將頻率更快的時脈傳送到 timer 即可加速。

```
  always @*
    case (speedup)
      4'd0:    clk_timer = clk_g[26];
      4'd1:    clk_timer = clk_g[25];
      4'd2:    clk_timer = clk_g[24];
      4'd3:    clk_timer = clk_g[23];
      4'd4:    clk_timer = clk_g[22];
      4'd5:    clk_timer = clk_g[21];
      4'd6:    clk_timer = clk_g[20];
      4'd7:    clk_timer = clk_g[19];
      default: clk_timer = clk_g[18];
    endcase
```

timer 的接腳有些是不會用到的，例如 carry_out。而 carry_in 恆為 1’b1，這樣只要有 clk_timer 或要 add_second、add_minute 或 add_hour，tiemr 裡面的計數器都會判斷是否需要加一或進位。

```
  timer timer(
    .ones_value_second(ones_value_second),
    .tens_value_second(tens_value_second),
    .ones_value_minute(ones_value_minute),
    .tens_value_minute(tens_value_minute),
    .ones_value_hour  (ones_value_hour_24),
    .tens_value_hour  (tens_value_hour_24),
    .carry_out        (),
    .carry_in         (1'b1),
    .add_second       (add_second),
    .add_minute       (add_minute),
    .add_hour         (add_hour),
    .switch_setting   (switch_setting),
    .clk              (clk_timer),
    .rst_n            (rst_n)
  );
```

其中，timer 的 hour 的數值是 24 小時制，必須要先通過 switch_hour 判斷當下的 switch_12_24，決定是否需要減十二，不需要的話才直接將數值傳送到 ssd_crtl。

判斷是否需要轉換成十二小時制之後，還要檢查 switch_hour_minute_second，以決定要顯示小時跟分鐘，還是顯示分鐘跟秒。

```
63   switch_hour switch_hour(
64     .ones_value_hour    (ones_value_hour),
65     .tens_value_hour    (tens_value_hour),
66     .ones_value_hour_24(ones_value_hour_24),
67     .tens_value_hour_24(tens_value_hour_24),
68     .switch_12_24       (switch_12_24)
69   );
70
71   always @*
72     if (switch_hour_minute_second) begin
73       bcd0 = ones_value_minute;
74       bcd1 = tens_value_minute;
75       bcd2 = ones_value_hour;
76       bcd3 = tens_value_hour;
77     end else begin
78       bcd0 = ones_value_second;
79       bcd1 = tens_value_second;
80       bcd2 = ones_value_minute;
81       bcd3 = tens_value_minute;
82     end
```

最後再把訊號傳送到 ssd_ctrl 即可。

```
84   ssd_ctrl ssd_ctrl(
85     .cathode(cathode),
86     .anode  (anode),
87     .bcd0   (bcd0),
88     .bcd1   (bcd1),
89     .bcd2   (bcd2),
90     .bcd3   (bcd3),
91     .clk    (clk_g[20:19])
92   );
93
94 endmodule
```

至此，完成第一題的所有模組都已經建構好了，只要再建構出 top 即可。

```
1  module LAB7_1_top(
2    output [7:0] cathode,
3    output [3:0] anode,
4
5    input add_second_undebounced,
6    input add_minute_undebounced,
7    input add_hour___undebounced,
8
9    input switch_setting,
10   input switch_hour_minute_second,
11   input switch_12_24,
12   input [3:0] speedup,
13
14   input clk_100mhz,
15   input rst_n
16 );
```

```
17   wire [26:0] clk_g;
18   wire add_second;
19   wire add_minute;
20   wire add_hour;
```

由於該建構的都幾乎建構完畢了，因此這題的 top 很單純。先除頻。

```
22   frequency_divider frequency_divider(
23     .clk_g     (clk_g),
24     .clk_100mhz(clk_100mhz),
25     .rst_n     (rst_n)
26   );
```

除出來的頻率用來 debounce 除了 clk_100mhz 之外的所有 input 訊號。

```
28   debounce debounce_add_second(
29     .debounced   (add_second),
30     .undebounced(add_second_undebounced),
31     .clk         (clk_g[18]),
32     .rst_n       (rst_n)
33   );
34
35   debounce debounce_add_minute(
36     .debounced   (add_minute),
37     .undebounced(add_minute_undebounced),
38     .clk         (clk_g[18]),
39     .rst_n       (rst_n)
40   );
41
42   debounce debounce_add_hour(
43     .debounced   (add_hour),
44     .undebounced(add_hour___undebounced),
45     .clk         (clk_g[18]),
46     .rst_n       (rst_n)
47   );
```

把 debounce 過的訊號連接到剛剛建構的 electronic_clock，這樣就大功告成了。

```
49   electronic_clock electronic_clock(
50     .cathode         (cathode),
51     .anode           (anode),
52     .add_second      (add_second),
53     .add_minute      (add_minute),
54     .add_hour        (add_hour),
55     .switch_setting  (switch_setting),
56     .switch_hour_minute_second(switch_hour_minute_second),
57     .switch_12_24    (switch_12_24),
58     .speedup         (speedup),
59     .clk_g           (clk_g),
60     .rst_n           (rst_n)
61   );
62
63 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。

有了上次 LAB 的經驗，我這題按部就班的一個一個模組慢慢寫。果然，雖然一開始的進度緩慢，但好處是，最後完全不用 debug，第一次 generate bitstream 測試就成功！比較困難的是下一題，我原本試著沿用本題辛辛苦苦寫好的模組，但卻因為月份和日期上下界的複雜度超乎我所預料，所以只好全部重寫。

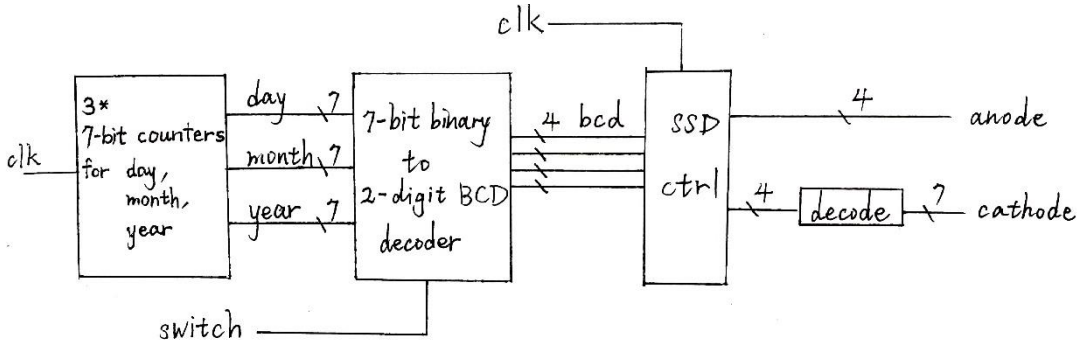
2 For the date functions in clock (no leap year), we have the following functions:
o Day (Jan/March/May/July/Aug/Oct/Dec: 1-31,
Feb: 28, Apr/June/Sept/Nov: 30),
o Month (1-12),
o Year (00-99).
Implement the following functions:
2.1 Month-Day function display in the 4 7-segment displays.
2.2 Combine the Year and 1.1 to finish a Year-Month-Day, and use one DIP switch to select the display of Year (2 Seven-Segment Displays, SSDs) or Month-Day (4 SSDs).

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 LAB7_1_top(
  output [7:0] cathode,                          // 七段顯示器
  output [3:0] anode,                            // 七段顯示器

  input undebounced_add_day,                     // 按一下加一天
  input undebounced_add_month,                   // 按一下加一個月
  input undebounced_add_year,                    // 按一下加一年

  input undebounced_switch_setting,              // 切換設定模式
  input undebounced_switch_day_month_year,       // 切換顯示年月日
  input [3:0] undebounced_speedup,               // demo 時加速用

  input clk_100mhz,                              // 內建的時脈
  input rst_n                                    // 重置、初始化
);
```

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

結構跟上一題一樣，只差在進位判斷和上界判斷的邏輯。但因為月份跟日期的上界判斷較複雜，所以本題捨棄以往一個位數用一個計數器的方式，而改成一個時間單位就使用一個計數器。也就是說，年月日總共只需要用三個計數器，每一個計數器含七個位元。



由計數器產生出來的七位元二進位數字再各自連接到一個七位元二進位轉二位數 BCD 的解碼器，最後才連接到七段顯示器的控制模組。

3. Finite state machine

內容: 電路中的 Finite state machine，若無則寫無。

無

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)。

直接把除頻器的計數器當作 output，提供其他模組各種頻率的時脈。

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

  assign clk_g_tmp = (clk_g < 27'd99_999_999) ? (clk_g + 1'b1) : ('b0);

  always @(posedge clk_100mhz or negedge rst_n)
    if (~rst_n) clk_g <= 'b0;
    else        clk_g <= clk_g_tmp;

endmodule
```

除頻器和 debounce 的模組都直接沿用上一題的就好，不須修改。

```
module debounce(
  output debounced,
  input  undebounced,
  input  clk,
  input  rst_n
);
  reg [3:0] undebounced_window;

  always @(posedge clk or negedge rst_n)
    if (~rst_n) undebounced_window <= 'b0;
    else        undebounced_window <= {undebounced_window, undebounced};

  assign debounced = (undebounced_window == 4'b1111);

endmodule
```

由於年月日的上下界判斷較複雜，如果要把每個時間單位拆開成兩個位數的話，會讓邏輯變得複雜。如果改成直接使用三個七位元的計數器分別記錄年月日的話，就不用各別考慮個位數的上下界了。

```
1  module counter(
2    output reg [6:0] value,
3
4    input      [6:0] upper_bound,
5    input      [6:0] lower_bound,
6
7    input            clk,
8    input            rst_n
9  );
10   wire [6:0] next_value;
11
12   assign next_value = (value < upper_bound)
13                     ? (value + 7'b1)
14                     : (lower_bound);
15
16   always @(posedge clk or negedge rst_n)
17     if (~rst_n) value <= 7'b0;
18     else        value <= next_value;
19
20 endmodule
```

結構跟上一題的四位元計數器一樣，只是新增了下界作為輸入。稍微修改第 12 到 14 行的判斷，當數值已經到達上界時，下一個時脈不是歸零，而是變成下界。

把負責年月日的三個七位元計數器包裝在同一個模組裡。

```
1  module timer(
2    output [6:0] day_value,
3    output [6:0] month_value,
4    output [6:0] year_value,
5
6    input add_day,
7    input add_month,
8    input add_year,
9
10   input switch_setting,
11   input [3:0] speedup,
12
13   input [26:0] clk_g,
14   input rst_n
15 );
16   wire [6:0] day_upper_bound;
17
18   wire day_clk;
19   wire month_clk;
20   wire tear_clk;
```

由於有大月跟小月，所以日期的上界要根據所在月份做調整。

```
22   determine_day_upper_bound determine_day_upper_bound(
23     .day_upper_bound,
24     .month_value
25   );
```

而為了 demo 方便，仿照上一題的模式，不直接把一赫茲的時脈連接到年月日的 clk。建構一個頻率選擇的模組，把要加速的倍率連接到計時的模組。

```
27  speedup_for_demo speedup_for_demo(
28    .day_clk,
29    .speedup,
30    .clk_g
31  );
```

由於負責進位判斷沒有內建在剛剛的年月日的七位元計數器內，所以要另外寫。可以直接把年跟月的 clk 改成連接到一個比較器，如果前一位數的數值為 7'd1，那就代表前一位數進位了。我將比較器的結果連接到 debounce 模組之後才再接到年跟月的 clk，主要是考量到比較器的輸出結果在輸入有變化時會有不穩定的數個波動，有點類似按按鈕時的彈簧震動，會使輸出快速的在 1'b0 跟 1'b1 之間切換。因此，如果不經過 debounce 模組就直接把比較器的結果輸出到月跟日的 clk 的話，常常會遇到明明前一位進位只需加一，但是卻因為比較器的輸出有震盪而增加了三、四次。

```
33  debounce_onepulse debounce_onepulse_month_clk(
34    .onepulsed(month_clk),
35    .debounced(),
36    .undebounced(day_value == 7'd1),
37    .clk(clk_g[14]),
38    .rst_n
39  );
40
41  debounce_onepulse debounce_onepulse_year_clk(
42    .onepulsed(year_clk),
43    .debounced(),
44    .undebounced(month_value == 7'd1),
45    .clk(clk_g[14]),
46    .rst_n
47  );
```

最後，把上述判斷出來的上界、clk、和下界指定給各個時間單位即可。

```
49  counter day_counter(
50    .value(day_value),
51    .upper_bound(day_upper_bound),
52    .lower_bound(7'd1),
53    .clk(switch_setting & day_clk | add_day),
54    .rst_n
55  );
56
57  counter month_counter(
58    .value(month_value),
59    .upper_bound(7'd12),
60    .lower_bound(7'd1),
61    .clk(switch_setting & month_clk | add_month),
62    .rst_n
63  );
64
65  counter year_counter(
66    .value(year_value),
67    .upper_bound(7'd99),
68    .lower_bound(7'd0),
69    .clk(switch_setting & year_clk | add_year),
70    .rst_n
71  );
72
73 endmodule
```

上述模組其中 determine_day_upper_bound 直接根據月份來判斷日期的上界。

```
module determine_day_upper_bound(
  output [6:0] day_upper_bound,
  input  [6:0] month_value
);

  assign day_upper_bound = (month_value == 7'd01) ? (7'd31) :
                           (month_value == 7'd02) ? (7'd28) :
                           (month_value == 7'd03) ? (7'd31) :
                           (month_value == 7'd04) ? (7'd30) :
                           (month_value == 7'd05) ? (7'd31) :
                           (month_value == 7'd06) ? (7'd30) :
                           (month_value == 7'd07) ? (7'd31) :
                           (month_value == 7'd08) ? (7'd31) :
                           (month_value == 7'd09) ? (7'd30) :
                           (month_value == 7'd10) ? (7'd31) :
                           (month_value == 7'd11) ? (7'd30) :
                           (month_value == 7'd12) ? (7'd31) : (7'hF);

endmodule
```

用來 demo 時加速的模組也是用 MUX 組成。

```
module speedup_for_demo(
  output reg day_clk,
  input [3:0] speedup,
  input [26:0] clk_g
);

  always @*
    case (speedup)
      4'd0:    day_clk = clk_g[26];
      4'd1:    day_clk = clk_g[25];
      4'd2:    day_clk = clk_g[24];
      4'd3:    day_clk = clk_g[23];
      4'd4:    day_clk = clk_g[22];
      4'd5:    day_clk = clk_g[21];
      4'd6:    day_clk = clk_g[20];
      4'd7:    day_clk = clk_g[19];
      default: day_clk = clk_g[18];
    endcase

endmodule
```

至此，timer 已經建構完成，接下來要建構用控制顯示器的一些模組。

由於 FPGA 上的七段顯示器只有四個位數，所以需要使用一個 switch 來切換顯示年月或是月日。依照 switch_year_month_day 來判斷要將年、月、日三者中的哪兩個訊號傳送到顯示器。

```
module switch_display(
  output [6:0] binary0,
  output [6:0] binary1,

  input  [6:0] day_value,
  input  [6:0] month_value,
  input  [6:0] year_value,

  input  switch_day_month_year
);

  assign binary0 = (switch_day_month_year) ? (month_value) : (day_value);
  assign binary1 = (switch_day_month_year) ? (year_value)  : (month_value);

endmodule
```

並使用七位元二進位轉二位數 BCD 的解碼器來把時間訊號轉換成 BCD。

```
module two_digits(
  output [3:0] ones,
  output [3:0] tens,
  input [6:0] binary
);
```

這裡有很多方法可以套用。我簡單依照十位數把所有情況分成十種 case。當然，這裡也可以直接把一百種情況列出來，只是我覺得那樣會有點太冗長。

```
  assign tens = (binary < 7'd10) ? (4'd0) :
                (binary < 7'd20) ? (4'd1) :
                (binary < 7'd30) ? (4'd2) :
                (binary < 7'd40) ? (4'd3) :
                (binary < 7'd50) ? (4'd4) :
                (binary < 7'd60) ? (4'd5) :
                (binary < 7'd70) ? (4'd6) :
                (binary < 7'd80) ? (4'd7) :
                (binary < 7'd90) ? (4'd8) :
                                   (4'd9) ;

  assign ones = (binary < 7'd10) ? (binary - 7'd00) :
                (binary < 7'd20) ? (binary - 7'd10) :
                (binary < 7'd30) ? (binary - 7'd20) :
                (binary < 7'd40) ? (binary - 7'd30) :
                (binary < 7'd50) ? (binary - 7'd40) :
                (binary < 7'd60) ? (binary - 7'd50) :
                (binary < 7'd70) ? (binary - 7'd60) :
                (binary < 7'd80) ? (binary - 7'd70) :
                (binary < 7'd90) ? (binary - 7'd80) :
                                   (binary - 7'd90) ;

endmodule
```

把以上三個模組（timer、switch_display、two_digits）包裝在一起，就完成了本題的計時功能與顯示功能。

```
module electronic_clock(
  output [7:0] cathode,
  output [3:0] anode,

  input add_day,
  input add_month,
  input add_year,

  input switch_setting,
  input switch_day_month_year,
  input [3:0] speedup,

  input [26:0] clk_g,
  input        rst_n
);

  wire clk_timer;

  wire [6:0] day_value;
  wire [6:0] month_value;
  wire [6:0] year_value;

  wire [6:0] binary0;
  wire [6:0] binary1;

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

連接方法很直觀。為了能控制要顯示年月還是月日模式，timer 的時間訊號先連接到 switch_display 判斷要保留年月日三者中的哪兩個訊號。

```
31   timer timer(
32     .day_value,
33     .month_value,
34     .year_value,
35     .add_day,
36     .add_month,
37     .add_year,
38     .switch_setting,
39     .speedup,
40     .clk_g,
41     .rst_n
42   );
```

```
44   switch_display switch_display(
45     .binary0,
46     .binary1,
47     .day_value,
48     .month_value,
49     .year_value,
50     .switch_day_month_year
51   );
```

再把保留的訊號經過解碼器轉成總共四個 BCD 之後，連接到控制七段顯示器的模組。其中七段顯示器的模組（ssd_ctrl）跟上一題一模一樣，直接沿用即可。

```
53   two_digits two_digit_binary0(
54     .ones(bcd0),
55     .tens(bcd1),
56     .binary(binary0)
57   );
58
59   two_digits two_digit_binary1(
60     .ones(bcd2),
61     .tens(bcd3),
62     .binary(binary1)
63   );
```

```
65   ssd_ctrl ssd_ctrl(
66     .cathode(cathode),
67     .anode   (anode),
68     .bcd0    (bcd0),
69     .bcd1    (bcd1),
70     .bcd2    (bcd2),
71     .bcd3    (bcd3),
72     .clk     (clk_g[20:19])
73   );
74
75 endmodule
```

至此，我們已經把所有計時和顯示的功能包裝好在一個模組（electronic_clock）裡了。下一步就是將 top 建構出來。

跟上一題一樣，由於我們已經把該建構的功能都建構出來了，我們接下來只需要將一些 FPGA 上面的 input 連接到 electronic_clock 上即可。

```
 1 module LAB7_1_top(
 2   output [7:0] cathode,
 3   output [3:0] anode,
 4
 5   input undebounced_add_day,
 6   input undebounced_add_month,
 7   input undebounced_add_year,
 8
 9   input undebounced_switch_setting,
10   input undebounced_switch_day_month_year,
11   input [3:0] undebounced_speedup,
12
13   input clk_100mhz,
14   input rst_n
15 );
```

```
16   wire [26:0] clk_g;
17
18   wire add_day;
19   wire add_month;
20   wire add_year;
21
22   wire switch_setting;
23   wire switch_day_month_year;
24   wire [3:0] speedup;
```

FPGA 內建的時脈 clk_100mhz 不用 debounce，直接連接到除頻器，再將除頻器的輸出與 electronic_clock 接好。

```
26   frequency_divider frequency_divider(
27     .clk_g,
28     .clk_100mhz,
29     .rst_n
30   );
```

先把所有可以用手操作的 input 連接到 debounce 模組。

```
  debounce_all_input debounce_all_input(
    .add_day,
    .add_month,
    .add_year,
    .switch_setting,
    .switch_day_month_year,
    .speedup,
    .undebounced_add_day,
    .undebounced_add_month,
    .undebounced_add_year,
    .undebounced_switch_setting,
    .undebounced_switch_day_month_year,
    .undebounced_speedup,
    .clk(clk_g[18]),
    .rst_n
  );
```

debounce 完畢的訊號再與 electronic_clock 對應的 port 連接。

```
  electronic_clock electronic_clock(
    .cathode,
    .anode,
    .add_day,
    .add_month,
    .add_year,
    .switch_setting,
    .switch_day_month_year,
    .speedup,
    .clk_g,
    .rst_n
  );

endmodule
```

其中，debounce_all_input 就只是把所有訊號 debounce 後再輸出而已。

```
module debounce_all_input(
  output add_day,
  output add_month,
  output add_year,

  output switch_setting,
  output switch_day_month_year,
  output [3:0] speedup,

  input undebounced_add_day,
  input undebounced_add_month,
  input undebounced_add_year,

  input undebounced_switch_setting,
  input undebounced_switch_day_month_year,
  input [3:0] undebounced_speedup,

  input clk,
  input rst_n
);
```

用來增加天數的按鈕。年和月的只要法照並把 day 改成 year 和 month 即可。

```
  debounce debounce_add_day(
    .debounced   (add_day),
    .undebounced(undebounced_add_day),
    .clk,
    .rst_n
  );
```

用來控制顯示和設定模式的兩個 switch。

```
43  debounce debounce_switch_setting(
44    .debounced  (switch_setting),
45    .undebounced(undebounced_switch_setting),
46    .clk,
47    .rst_n
48  );
49
50  debounce debounce_switch_day_month_year(
51    .debounced  (switch_day_month_year),
52    .undebounced(undebounced_switch_day_month_year),
53    .clk,
54    .rst_n
55  );
```

四個控制 demo 速度的 switch，其他三個只要把 0 改成 1、2、3 即可。

```
57  debounce debounce_add_speedup0(
58    .debounced  (speedup[0]),
59    .undebounced(undebounced_speedup[0]),
60    .clk,
61    .rst_n
62  );
```

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。

這題我折騰了半天，到了隔天早上我才 debug 完畢。用來計時年月日 couunter 的那部份我原本想直接沿用上一題的模組，但是無論我怎麼修改判斷個位數的下界的邏輯，都會莫名其妙不進位或是一直卡在零然後不上數。有鑑於上一次 LAB 的經驗，我直接毅然決然直接重頭開始再建構一次，並採用了新的方法，不再需要冗長而複雜的上下界判斷。

此外，這一題讓我學到一個很重要的觀念：這是硬體描述語言，不能單憑之前學過的軟體的觀念來寫，還必須考慮到實際上的硬體層面的問題。例如：原本在我的模組 timer 裡面，年和月份 carry_in 沒有 debounce 就直接連接了，這就造成了很嚴重的問題，那就是比較器需要一點點的時間才能反映出當下真正的數值。

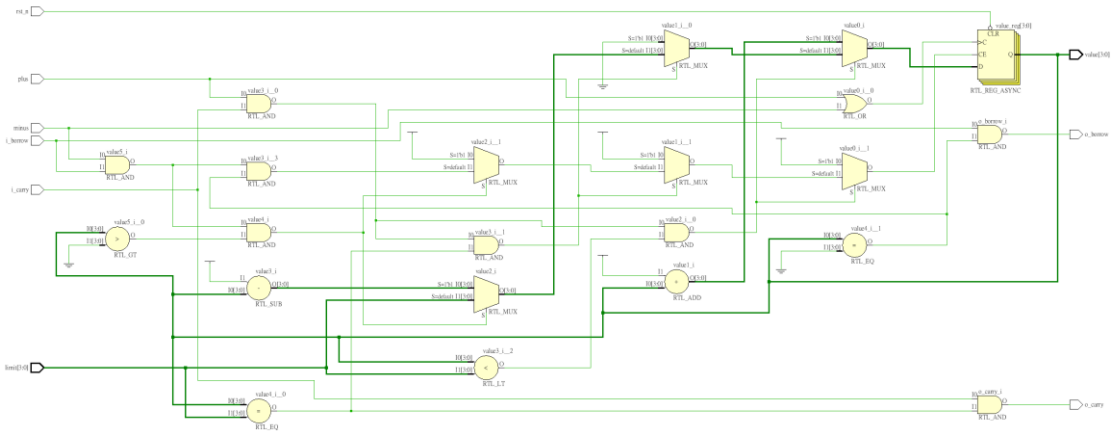
換句話說，如果要把非天生就是時脈的訊號，例如非 clk_100mhz、非除頻器製造的 clk_g[26:0]等等，連接到某個 Flip-flop 的 c 當作時脈的話，就建議要考慮是否有需要 debounce 或甚至 debounce + onepulse。這讓我驚覺原來我學期剛開始的那幾個倒計時的 LAB 會一直做不出來的原因，例如，當時還沒有意識到以下形式的硬體描述語言背後的真正含意：

```
reg q;
wire a, b, c, d, …;
…
always @(posedge a | b | c | …)
  q <= (o/p of some CL w/ a, b, c, … as inputs);
```

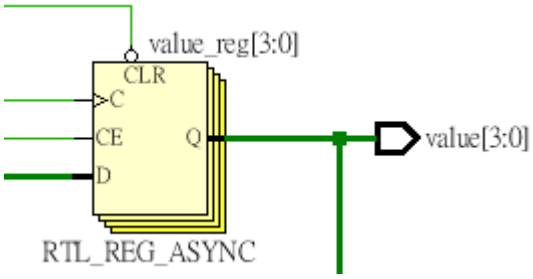
更直接一點的例子（上數兼下數計數器，來源：LAB6 的第二個失敗版本）：

```
 1 module counter(
 2   output reg [3:0] value,
 3   output           o_carry,
 4   output           o_borrow,
 5   input            i_carry,
 6   input            plus,
 7   input            i_borrow,
 8   input            minus,
 9   input      [3:0] limit,
10   input            rst_n
11 );
12
13   always @ (posedge plus | minus or negedge rst_n)
14     value <= (~rst_n                              ) ? (0         ) :
15              (plus  & i_carry  & value <  limit) ? (value + 1) :
16              (plus  & i_carry  & value == limit) ? (0         ) :
17              (minus & i_borrow & value >  0    ) ? (value - 1) :
18              (minus & i_borrow & value == 0    ) ? (limit     ) :
19                                                    (value     ) ;
20
21   assign o_carry  = i_carry  & (value == limit);
22   assign o_borrow = i_borrow & (value == 0);
23
24 endmodule
```

我原本所期待的是，每個 plus 和 minus 的 rising edge 都會使得 value 的數值增加一或減少一。雖然依照軟體語言的解釋方法，這段（第 13 到 19 行）描述完全沒有問題。然而，由於 plus 和 minus 同時出現在 D Flip-flop 的 C 和 D 前面，很有可能在 D 還沒有反應完畢當下 plus 或 minus 改變所造成的不同輸出之前，因為 C 已經先有 rising edge 了，這個 Flip-flop 就把還沒經過 combinational logic 運算完畢的「舊的」D 值存入 value。從下圖 RTL analysis 的 schematic 可以看出來：



請特別留意上圖右上角的 Flip-flop。



由於 D 前面是一大串 combinational logic 的運算結果，有可能在 plus 或 minus 改變後、D 更新完畢當下應有的值之前，這個 D Flip-flop 就把舊的 D 值存入 value 了。要改善這種情況其實很簡單，只要把原本直接連接到 C 的 plus 與 minus 先經過 debounce 之後才接到 C 即可。如此一來，D 前面的一大串 CL 就能有數個 clk 的時間可以確保提供已經運算完畢的新 D 值給 Flip-flop。

由上述觀察可以發現，debounce 或 onepulse 模組不僅僅可以用來穩定 FPGA 外來的非 clk 的 input，還可以用在有需要拿非 clk 的 signal 用來當作 Flip-flop 的 C 的時候使用，用來穩定 CL 輸出（如第一個本題的比較器與 carry_in 的例子）、或是用來延遲 C 收到 rising edge 的時間（如第二個上數兼下數計數器的例子）。

此外，我去實驗室 demo 時，現場的助教檢查了一下我的 verilog。助教說，D Flip-flop 的 CLK 盡量只連接 FPGA 內建的時脈（clk_100mhz）或是經過除頻器處裡過後的時脈（clk_g）。CLK 盡量不要直接與除了時脈之外的 input 連接，比較理想的替代做法應該是，把所有 input 訊號先傳到 FSM，再由 FSM 傳訊號給 Flip-flop。