LAB4_1

Construct a 4-bit synchronous binary up counter (b3b2b1b0) with the 1-Hz clock frequency from lab2 and use 4 LEDs for display.

I/O	fcrystal	b3	b2	b1	b0
Site	W5	V19	U19	E19	U16

specification:

output:

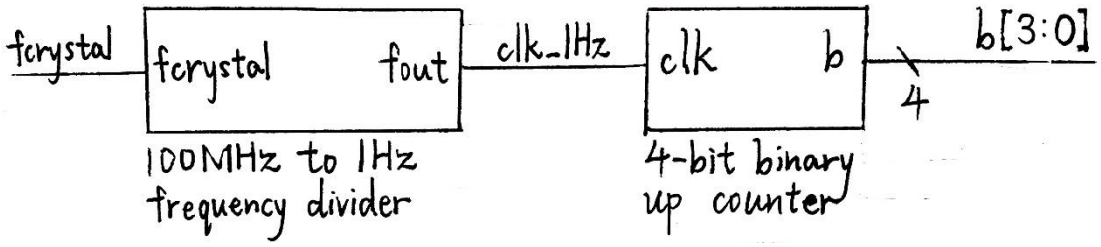
b[3:0] // binary value

input:

fcrystal // 100M Hz global clock

rst_n // active low reset

circuit diagram:



top: 把除頻器的一赫茲輸出連接到計數器作為時脈

```
23 module LAB4_1(                          // TOP
24     output [3:0] b,                     // LEDs display binary 4'b0000 - 4'b1111
25     input        fcrystal,              // 100M Hz global clock
26     input        rst_n                  // active low reset
27     );
28     wire clk_1Hz;                       // 1 Hz 50% duty clock
29
30     frequency_divider_100M FD(          // 100M Hz to 1 Hz frequency divider
31         .fout(clk_1Hz),                 // 1 Hz 50% duty clock
32         .fcrystal(fcrystal),            // 100M Hz global clock
33         .rst_n(rst_n)                   // active low reset
34         );
35
36     counter_up_binary_4bit COUNTER(     // 4-bit binary up counter
37         .q(b),                          // binary 4'b0000 - 4'b1111
38         .clk(clk_1Hz),                  // 1 Hz 50% duty clock
39         .rst_n(rst_n)                   // active low reset
40         );
41
42 endmodule                              // LAB4_1
```

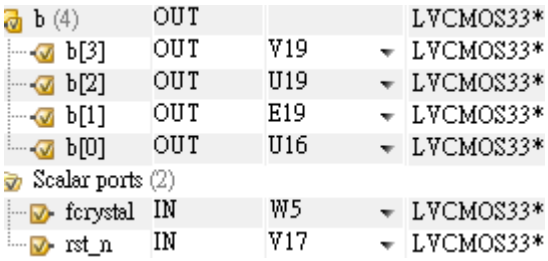
100M Hz to 1 Hz 50% duty frequency divider:
先數到 50M，再連接到一個 TFF 即可

```
23 module frequency_divider_100M(   // 100M Hz to 1 Hz 50% duty frequency divider
24     output reg fout,             // output frequency
25     input      fcrystal,         // crystal frequency
26     input      rst_n             // active low reset
27     );
28     reg [26:0] counter;
29
30     // count for 50M times
31     always @ (posedge fcrystal or negedge rst_n)
32     begin
33         if (~rst_n)
34             counter <= 1;        // reset
35         else
36             counter <= (counter < 50_000_000) ? (counter + 1) : (1);
37     end
38
39     // creat 50% duty output
40     always @ (posedge fcrystal or negedge rst_n)
41     begin
42         if (~rst_n)
43             fout <= 1;           // reset
44         else begin
45             if (counter == 1)
46                 fout <= ~fout;   // 50% duty
47     end end
48
49 endmodule                       // frequency_divider_100M
```

4-bit binary up counter:

```
23 module counter_up_binary_4bit(   // 4-bit binary up counter
24     output reg [3:0] q,
25     input            clk,       // clock
26     input            rst_n      // active low reset
27     );
28
29     // 4-bit binary up counter
30     always @ (posedge clk or negedge rst_n)
31     begin
32         if (~rst_n)
33             q <= 0;              // reset
34         else
35             q <= q + 1;          // add 1
36     end
37
38 endmodule                       // counter_up_binary_4bit
```

implement:



b (4)	OUT		LVCMOS33*
b[3]	OUT	V19	LVCMOS33*
b[2]	OUT	U19	LVCMOS33*
b[1]	OUT	E19	LVCMOS33*
b[0]	OUT	U16	LVCMOS33*
Scalar ports (2)			
fcrystal	IN	W5	LVCMOS33*
rst_n	IN	V17	LVCMOS33*

conclusion:

這一題要做的是一個以一赫茲為時脈的四位元二進位上數計數器。解題思路如下：先拿一個計數器當作除頻器，利用 FPGA 內建的 100M 赫茲時脈產生一赫茲的時脈。再把該一赫茲時脈連接到一個四位元上數計數器。因為 100M 赫茲轉一赫茲的除頻器與四位元上數計數器在上次的 LAB3 就已經做過了，規格一模一樣，所以直接拿來用即可。

LAB4_2

Combine the 4-bit synchronous binary up counter from exp1 with a binary-to-seven-segment-display decoder (from lab2-exp3) to display the binary counting in 7-segment display.

specification:

output:

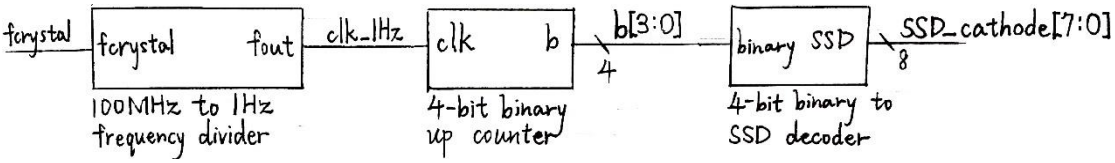
SSD_cathode[7:0] // individual cathodes of seven segment display

input:

fcrystal // 100M Hz global clock

rst_n // active low reset

circuit diagram:



top: 把除頻器產生的一赫茲時脈連接到四位元計數器，再接一個解碼器。

```
23 module LAB4_2(                          // TOP
24     output [7:0] SSD_cathode,           // individual cathodes of seven segment display
25     input        fcrystal,              // 100M Hz global clock
26     input        rst_n                  // active low reset
27     );
28     wire       clk_1Hz;                 // 1 Hz 50% duty clock
29     wire [3:0] binary_4bit;             // 4-bit binary
30
31     frequency_divider_100M FD(          // 100M Hz to 1 Hz 50% duty frequency divider
32         .fout(clk_1Hz),                 // 1 Hz 50% duty clock
33         .fcrystal(fcrystal),            // 100M Hz global clock
34         .rst_n(rst_n)                   // active low reset
35         );
36
37     counter_up_binary_4bit COUNTER(     // 4-bit binary up counter
38         .q(binary_4bit),                // 4-bit binary
39         .clk(clk_1Hz),                  // 1 Hz clock
40         .rst_n(rst_n)                   // active low reset
41         );
42
43     binary_to_SSD_decoder DECODER(      // 4-bit binary to SSD decoder
44         .D(SSD_cathode),                // [7:0] individual cathodes of seven segment display
45         .i(binary_4bit)                 // 4-bit binary
46         );
47
48 endmodule                              // LAB4_2
```

100M Hz to 1 Hz 50% duty frequency divider:

數到 50M，再接一個 TFF。

```
module frequency_divider_100M(  // 100M Hz to 1 Hz 50% duty frequency divider
    output reg fout,            // output frequency
    input      fcrystal,        // crystal frequency
    input      rst_n            // active low reset
    );
    reg [26:0] counter;

    // count for 50M times
    always @ (posedge fcrystal or negedge rst_n)
    begin
        if (~rst_n)
            counter <= 1;       // reset
        else
            counter <= (counter < 50_000_000) ? (counter + 1) : (1);
    end

    // creat 50% duty output
    always @ (posedge fcrystal or negedge rst_n)
    begin
        if (~rst_n)
            fout <= 1;          // reset
        else begin
            if (counter == 1)
                fout <= ~fout;  // 50% duty
    end end

endmodule                       // frequency_divider_100M
```

4-bit binary up counter:

```
module counter_up_binary_4bit(  // 4-bit binary up counter
    output reg [3:0] q,
    input            clk,       // clock
    input            rst_n      // active low reset
    );

    // 4-bit binary up counter
    always @ (posedge clk or negedge rst_n)
    begin
        if (~rst_n)
            q <= 0;         // reset
        else
            q <= q + 1;     // add 1
    end

endmodule                   // counter_up_binary_4bit
```

4-bit binary to SSD decoder:

```
23 module binary_to_SSD_decoder(   // 4-bit binary to SSD decoder
24     output reg [7:0] D,         // 7-segment display decode
25     input      [3:0] i          // 4-bit binary number
26     );
27
28     // binary (0-9, a, b, c, d, e, f) to 7-segment display decoder
29     always @ (i)
30     begin
31         case (i)
32             0:       D = 8'b0000001_1;  // 0
33             1:       D = 8'b1001111_1;  // 1
34             2:       D = 8'b0010010_1;  // 2
35             3:       D = 8'b0000110_1;  // 3
36             4:       D = 8'b1001100_1;  // 4
37             5:       D = 8'b0100100_1;  // 5
38             6:       D = 8'b0100000_1;  // 6
39             7:       D = 8'b0001111_1;  // 7
40             8:       D = 8'b0000000_1;  // 8
41             9:       D = 8'b0000100_1;  // 9
42             10:      D = 8'b0001000_1;  // a
43             11:      D = 8'b1100000_1;  // b
44             12:      D = 8'b0110001_1;  // c
45             13:      D = 8'b1000010_1;  // d
46             14:      D = 8'b0110000_1;  // e
47             15:      D = 8'b0111000_1;  // f
48             default: D = 8'b0000000_0;  // all on
49         endcase
50     end                         // always
51
52 endmodule                       // binary_to_SSD_decoder
```

implement:

SSD_cathode (8)	OUT		LVCMOS33*
SSD_cathode[7]	OUT	W7	LVCMOS33*
SSD_cathode[6]	OUT	W6	LVCMOS33*
SSD_cathode[5]	OUT	U8	LVCMOS33*
SSD_cathode[4]	OUT	V8	LVCMOS33*
SSD_cathode[3]	OUT	U5	LVCMOS33*
SSD_cathode[2]	OUT	V5	LVCMOS33*
SSD_cathode[1]	OUT	U7	LVCMOS33*
SSD_cathode[0]	OUT	V7	LVCMOS33*
Scalar ports (2)			
fcrystal	IN	W5	LVCMOS33*
rst_n	IN	V17	LVCMOS33*

conclusion:

這一題跟上一題一樣要做一個以一赫茲為時脈的四位元二進位上數計數器，但這題要求使用七段顯示器來監控計數器的數值。這題也很單純，只要將上一題的四位元計數器的輸出連接到一個二進位轉七段顯示器的解碼器即可。

LAB4_3

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

3.1 Construct a BCD up counter.

3.2 Construct a BCD-to-seven-segment display decoder (from lab2-exp2).

3.3 Combine the above two together.

specification:

output:

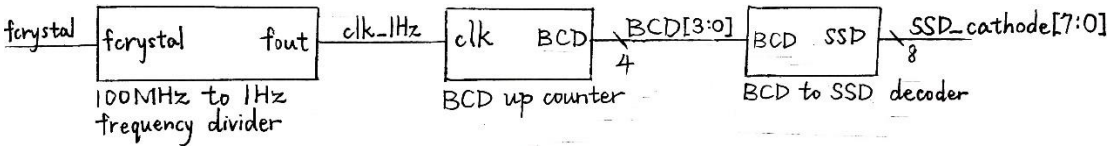
SSD_cathode[7:0] // individual cathodes of seven segment display

input:

fcrystal // 100M Hz global clock

rst_n // active low reset

circuit diagram:



top: 將除頻器產生的一赫茲時脈連接到 BCD 上數計數器，再連接解碼器。

```
23 module LAB4_3(                          // TOP
24     output [7:0] SSD_cathode,           // individual cathodes of seven segment display
25     input        fcrystal,              // 100M Hz global clock
26     input        rst_n                  // active low reset
27     );
28     wire       clk_1Hz;                 // 1 Hz 50% duty clock
29     wire [3:0] BCD;                     // binary-coded decimal
30
31     frequency_divider_100M FD(          // 100M Hz to 1 Hz 50% duty frequency divider
32         .fout(clk_1Hz),                 // 1 Hz 50% duty clock
33         .fcrystal(fcrystal),            // 100M Hz global clock
34         .rst_n(rst_n)                   // active low reset
35         );
36
37     counter_up_BCD COUNTER(             // BCD up counter
38         .BCD(BCD),                      // BCD
39         .clk(clk_1Hz),                  // 1 Hz clock
40         .rst_n(rst_n)                   // active low reset
41         );
42
43     decoder_BCD_to_SSD DECODER(         // BCD to SSD decoder
44         .SSD_cathode(SSD_cathode),      // [7:0] individual cathodes of seven segment display
45         .BCD(BCD)                       // BCD
46         );
47
48 endmodule                              // LAB4_3
```

100M Hz to 1 Hz 50% duty frequency divider:
數到 50M，然後連接到一個 TFF。

```
module frequency_divider_100M(   // 100M Hz to 1 Hz 50% duty frequency divider
    output reg fout,             // output frequency
    input      fcrystal,         // crystal frequency
    input      rst_n             // active low reset
    );
    reg [26:0] counter;

    // count for 50M times
    always @ (posedge fcrystal or negedge rst_n)
    begin
        if (~rst_n)
            counter <= 1;        // reset
        else
            counter <= (counter < 50_000_000) ? (counter + 1) : (1);
    end

    // creat 50% duty output
    always @ (posedge fcrystal or negedge rst_n)
    begin
        if (~rst_n)
            fout <= 1;           // reset
        else begin
            if (counter == 1)
                fout <= ~fout;   // 50% duty
    end end

endmodule                        // frequency_divider_100M
```

BCD up counter:

```
module counter_up_BCD(                          // BCD up counter
    output reg [3:0] BCD,                       // BCD
    input            clk,                       // clock
    input            rst_n                      // active low reset
    );

    always @ (posedge clk or negedge rst_n)
    begin
        if (~rst_n)
            BCD <= 0;                           // reset
        else
            BCD <= (BCD < 9) ? (BCD + 1) : (0); // add 1
    end

endmodule                                       // counter_up_BCD
```

BCD to SSD decoder:

```
23 module decoder_BCD_to_SSD(          // BCD to SSD decoder
24     output reg [7:0] SSD_cathode,   // individual cathodes of seven segment display
25     input      [3:0] BCD            // BCD
26     );
27
28     always @ (BCD)
29     begin
30         case (BCD)
31             0:       SSD_cathode = 8'b0000001_1;  // 0
32             1:       SSD_cathode = 8'b1001111_1;  // 1
33             2:       SSD_cathode = 8'b0010010_1;  // 2
34             3:       SSD_cathode = 8'b0000110_1;  // 3
35             4:       SSD_cathode = 8'b1001100_1;  // 4
36             5:       SSD_cathode = 8'b0100100_1;  // 5
37             6:       SSD_cathode = 8'b0100000_1;  // 6
38             7:       SSD_cathode = 8'b0001111_1;  // 7
39             8:       SSD_cathode = 8'b0000000_1;  // 8
40             9:       SSD_cathode = 8'b0000100_1;  // 9
41             default: SSD_cathode = 8'b0000000_0;  // all on
42         endcase
43     end                             // always
44
45 endmodule                          // decoder_BCD_to_SSD
```

implement:

SSD_cathode (8)	OUT		LVCMOS33*
SSD_cathode[7]	OUT	W7	LVCMOS33*
SSD_cathode[6]	OUT	W6	LVCMOS33*
SSD_cathode[5]	OUT	U8	LVCMOS33*
SSD_cathode[4]	OUT	V8	LVCMOS33*
SSD_cathode[3]	OUT	U5	LVCMOS33*
SSD_cathode[2]	OUT	V5	LVCMOS33*
SSD_cathode[1]	OUT	U7	LVCMOS33*
SSD_cathode[0]	OUT	V7	LVCMOS33*
Scalar ports (2)			
fcrystal	IN	W5	LVCMOS33*
rst_n	IN	V17	LVCMOS33*

conclusion:

這一題跟上一題很像，要做的都是上數計數器，但是這題是 BCD。也就是說，數到九之後的下一個時脈不是變成十，而是變成零，如 counter_up_BCD 的第 34 行所示。此外，因為計數器的數值不會大於九，因此不需要如上一題一樣把四位元二進位的所有可能情況都寫進解碼器，只要保留零到九就好，如 decoder_BCD_to_SSD 的第 31 到 41 行所示。

LAB4_4

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

4.1 Construct a BCD up counter.

4.2 Construct a BCD-to-seven-segment display decoder (from lab2-exp2).

4.3 Combine the above two together.

specification:

output:

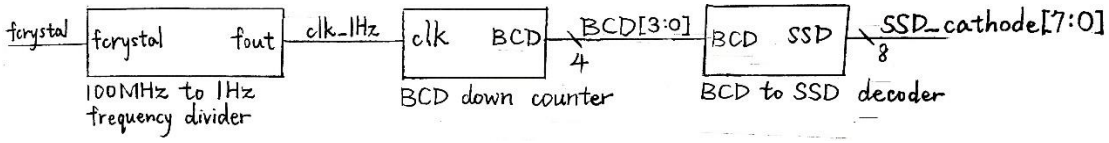
SSD_cathode[7:0] // individual cathodes of seven segment display

input:

fcrystal // 100M Hz global clock

rst_n // active low reset

circuit diagram:



top: 將除頻器所產生的一赫茲時脈連接到 BCD 下數計數器，然後再接解碼器

```
23 module LAB4_4(                          // TOP
24     output [7:0] SSD_cathode,           // individual cathodes of seven segment display
25     input        fcrystal,              // 100M Hz global clock
26     input        rst_n                  // active low reset
27     );
28     wire       clk_1Hz;                 // 1 Hz 50% duty clock
29     wire [3:0] BCD;                     // binary-coded decimal
30
31     frequency_divider_100M FD(          // 100M Hz to 1 Hz 50% duty frequency divider
32         .fout(clk_1Hz),                 // 1 Hz 50% duty clock
33         .fcrystal(fcrystal),            // 100M Hz global clock
34         .rst_n(rst_n)                   // active low reset
35         );
36
37     counter_down_BCD COUNTER(           // BCD down counter
38         .BCD(BCD),                      // BCD
39         .clk(clk_1Hz),                  // 1 Hz clock
40         .rst_n(rst_n)                   // active low reset
41         );
42
43     decoder_BCD_to_SSD DECODER(         // BCD to SSD decoder
44         .SSD_cathode(SSD_cathode),      // [7:0] individual cathodes of seven segment display
45         .BCD(BCD)                       // BCD
46         );
47
48 endmodule                              // LAB4_4
```

100M Hz to 1 Hz 50% duty frequency divider:
先數到 50M，再連接到一個 TFF。

```
23 module frequency_divider_100M(  // 100M Hz to 1 Hz 50% duty frequency divider
24     output reg fout,            // output frequency
25     input      fcrystal,        // crystal frequency
26     input      rst_n            // active low reset
27     );
28     reg [26:0] counter;
29
30     // count for 50M times
31     always @ (posedge fcrystal or negedge rst_n)
32     begin
33         if (~rst_n)
34             counter <= 1;       // reset
35         else
36             counter <= (counter < 50_000_000) ? (counter + 1) : (1);
37     end
38
39     // creat 50% duty output
40     always @ (posedge fcrystal or negedge rst_n)
41     begin
42         if (~rst_n)
43             fout <= 1;          // reset
44         else begin
45             if (counter == 1)
46                 fout <= ~fout;  // 50% duty
47     end end
48
49 endmodule                      // frequency_divider_100M
```

BCD down counter:

```
23 module counter_down_BCD(                         // BCD down counter
24     output reg [3:0] BCD,                        // BCD
25     input            clk,                        // clock
26     input            rst_n                       // active low reset
27     );
28
29     always @ (posedge clk or negedge rst_n)
30     begin
31         if (~rst_n)
32             BCD <= 0;                            // reset
33         else
34             BCD <= (BCD == 0) ? (9) : (BCD - 1); // subtract 1
35     end
36
37 endmodule                                        // counter_down_BCD
```

BCD to SSD decoder:

```
23 module decoder_BCD_to_SSD(          // BCD to SSD decoder
24     output reg [7:0] SSD_cathode,   // individual cathodes of seven segment display
25     input      [3:0] BCD            // BCD
26     );
27
28     always @ (BCD)
29     begin
30         case (BCD)
31             0:       SSD_cathode = 8'b0000001_1;  // 0
32             1:       SSD_cathode = 8'b1001111_1;  // 1
33             2:       SSD_cathode = 8'b0010010_1;  // 2
34             3:       SSD_cathode = 8'b0000110_1;  // 3
35             4:       SSD_cathode = 8'b1001100_1;  // 4
36             5:       SSD_cathode = 8'b0100100_1;  // 5
37             6:       SSD_cathode = 8'b0100000_1;  // 6
38             7:       SSD_cathode = 8'b0001111_1;  // 7
39             8:       SSD_cathode = 8'b0000000_1;  // 8
40             9:       SSD_cathode = 8'b0000100_1;  // 9
41             default: SSD_cathode = 8'b0000000_0;  // all on
42         endcase
43     end                             // always
44
45 endmodule                          // decoder_BCD_to_SSD
```

implement:

SSD_cathode (8)	OUT		LVCMOS33*
SSD_cathode[7]	OUT	W7	LVCMOS33*
SSD_cathode[6]	OUT	W6	LVCMOS33*
SSD_cathode[5]	OUT	U8	LVCMOS33*
SSD_cathode[4]	OUT	V8	LVCMOS33*
SSD_cathode[3]	OUT	U5	LVCMOS33*
SSD_cathode[2]	OUT	V5	LVCMOS33*
SSD_cathode[1]	OUT	U7	LVCMOS33*
SSD_cathode[0]	OUT	V7	LVCMOS33*
Scalar ports (2)			
fcrystal	IN	W5	LVCMOS33*
rst_n	IN	V17	LVCMOS33*

conslusion:

這一題幾乎跟上一題一模一樣，唯一的差別在於上一題是上數，這一題是下數。上一題是不斷加一直到數值為九，再過一時脈變為零，而這一題則是不斷減一直到數值為零，再過一時脈變為九，如 counter_down_BCD 的第 34 行所示。

LAB4_5

(Bonus) Construct a 30-second count down timer (stop at 00).

specification:

output:

cathode[7:0] // cathodes of similar segments on all four displays

anode[3:0] // anodes of the seven LEDs forming each digit

input:

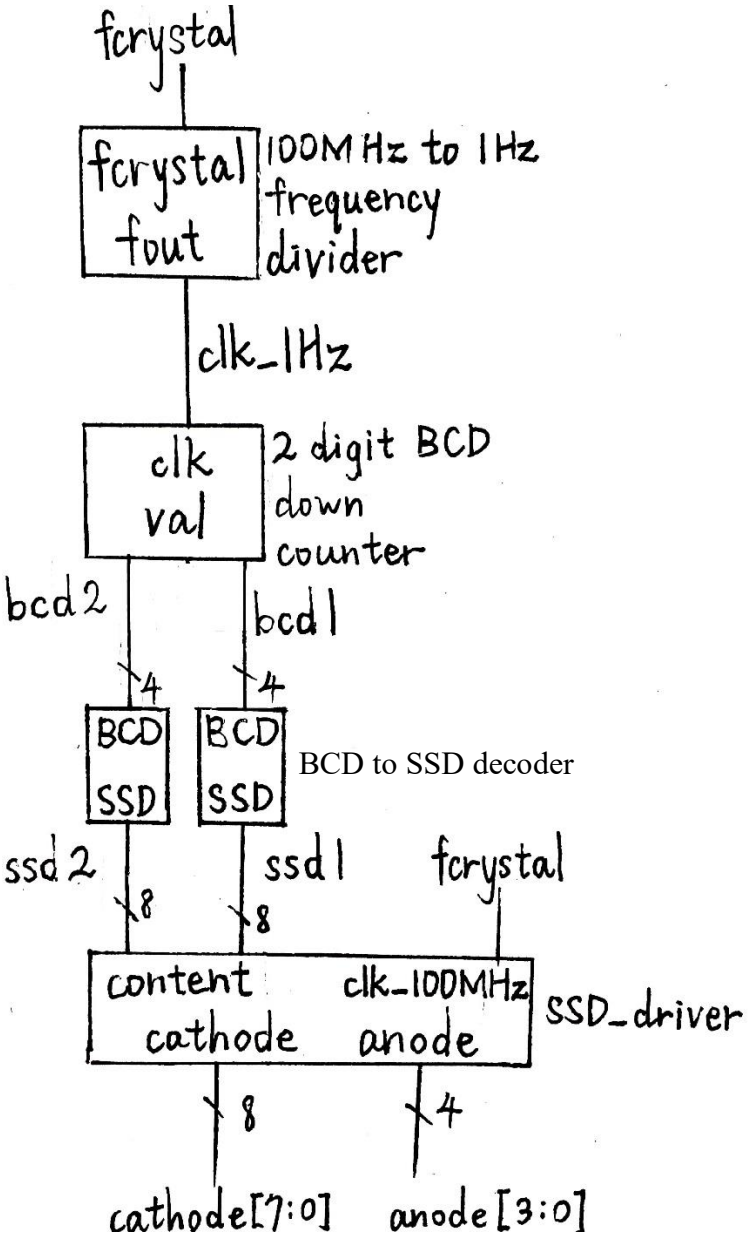
enable // start timing

fcrystal // 100M Hz global clock

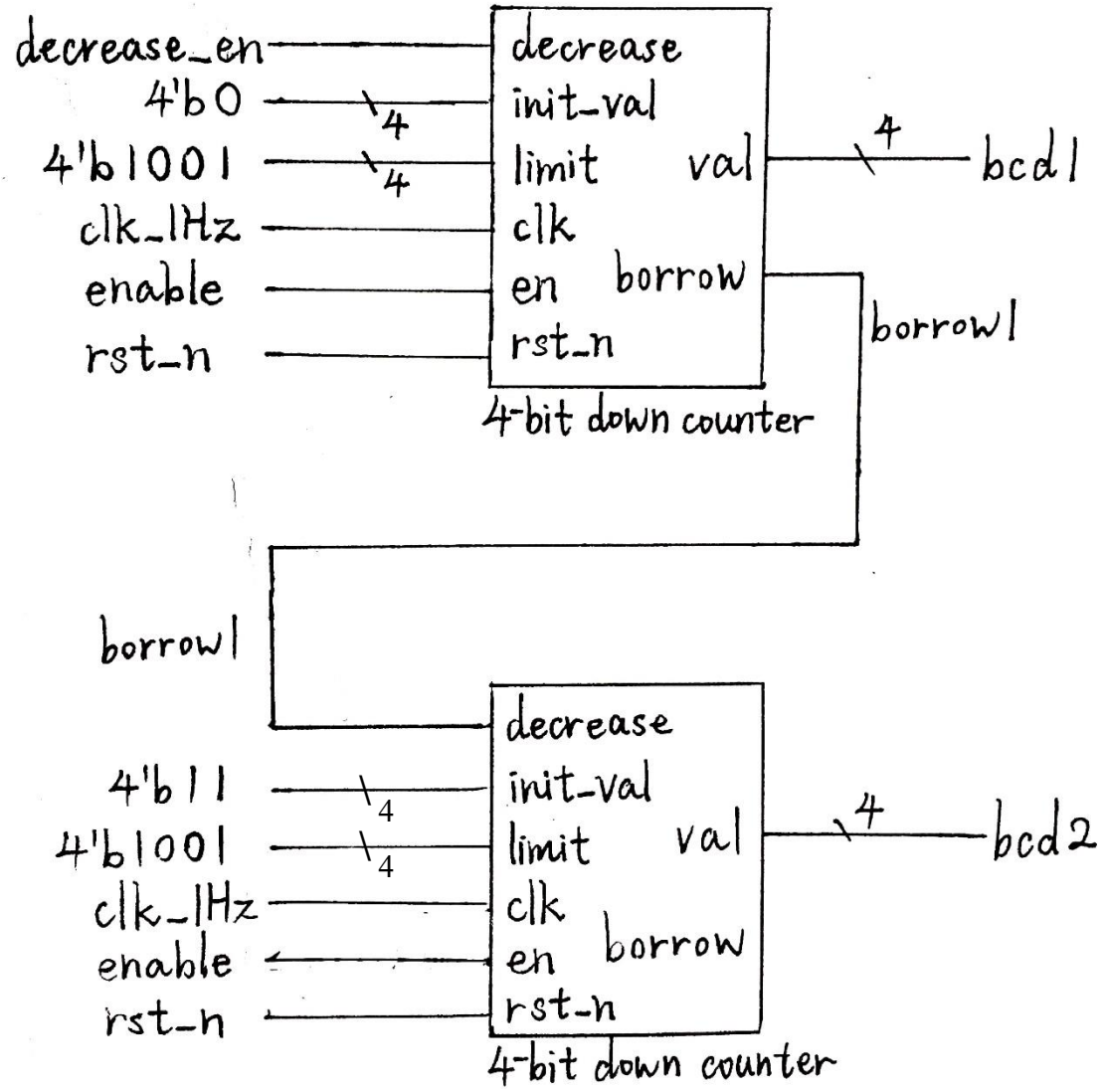
rst_n // active low reset

circuit diagram:

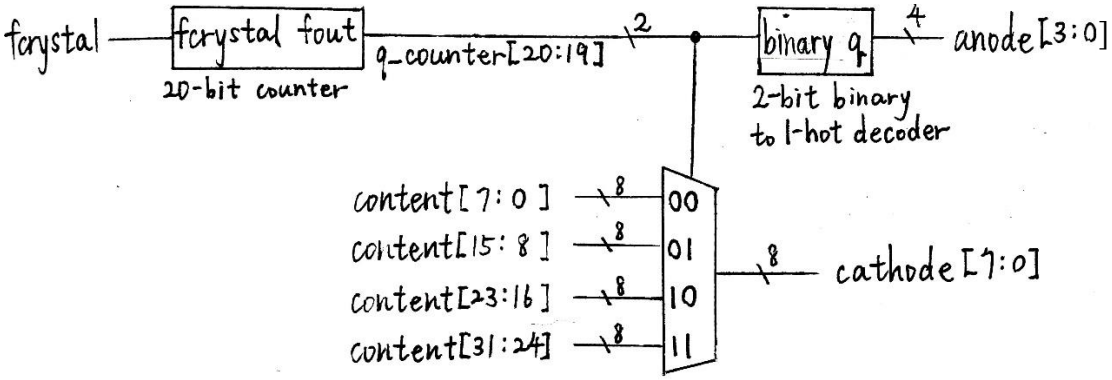
top:



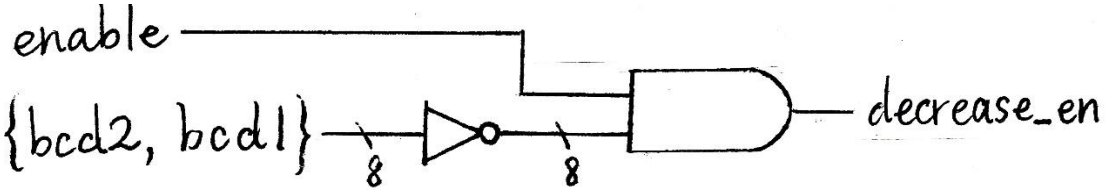
2-digit BCD down counter:



SSD driver:



decrease enable:



100M Hz to 1 Hz 50% duty frequency divider:

把 FPGA 內建的 100M 赫茲時脈連接到一個計數器，只要數到 50M 就重置為一，再將重置的訊號連接到一個 TFF 即可產生一赫茲的時脈。

```
23 module frequency_divider_100M(   // 100M Hz to 1 Hz 50% duty frequency divider
24     output reg fout,             // output frequency
25     input      fcrystal,         // crystal frequency
26     input      rst_n             // active low reset
27     );
28     reg [26:0] counter;
29
30     // count for 50M times
31     always @ (posedge fcrystal or negedge rst_n)
32     begin
33         if (~rst_n)
34             counter <= 1;        // reset
35         else
36             counter <= (counter < 50_000_000) ? (counter + 1) : (1);
37     end
38
39     // creat 50% duty output
40     always @ (posedge fcrystal or negedge rst_n)
41     begin
42         if (~rst_n)
43             fout <= 1;           // reset
44         else begin
45             if (counter == 1)
46                 fout <= ~fout;   // 50% duty
47     end end
48
49 endmodule                       // frequency_divider_100M
```

4-bit down counter:

四位元下數計數器的 code 如下圖。其中，借位的訊號較單純，條件為：啟用（enable）且要減一（decrease）且當前的數值為零（value==0）。使用一般的組合邏輯電路即可，如下圖第 36 行所示。而下一刻的數值（val_next）則要使用 sequential logic 來設計：使用四個 flip-flop 來儲存當前的數值（val），過一個時脈後將下一個數值（val_next）存入，如下圖第 52 到 58 行所示。下一個時脈的數值（val_next）要使用組合邏輯電路來產生，可分為四種情況：如果不啟用（enable），則重設為初始數值（initial value）；如果啟用但不用減一（decrease），則維持當前數值；如果需要減一且當前數值非零，則直接減一；如果需要減一但當前數值為零，則將數值設為最大值（limit），如下圖第 39 到 49 行所示。

```
23 module counter_down_4bit(
24     output reg [3:0] val,        // value
25     output           borrow,     // subtraction carry
26     input            decrease,   // minus 1
27     input      [3:0] init_val,   // initial value
28     input      [3:0] limit,      // highest possible value
29     input            clk,        // clock
30     input            en,         // enable
31     input            rst_n       // active low reset
32     );
33     reg [3:0] val_next;
34
35     // combinational logic for borrowing
36     assign borrow = en & decrease & (val == 0);
37
38     // combinational logic before the FF of value
39     always @ (val or decrease or en)
40     begin
41         if (en && decrease && (val == 0))
42             val_next <= limit;
43         else if (en && decrease)
44             val_next <= val - 1;
45         else if (en)
46             val_next <= val;
47         else
48             val_next <= init_val;
49     end
50
51     // FF for the sequential logic of value
52     always @ (posedge clk or negedge rst_n)
53     begin
54         if (~rst_n)
55             val <= 0;
56         else
57             val <= val_next;
58     end
59
60 endmodule                        // counter_down_4bit
```

4-bit binary to SSD pattern decoder:

其實這裡使用 BCD to SSD decoder 就好，也就是說，下圖第 42 到 47 行是不必要的，因為前面連接的是一個只有可能產生 0~9 的下數計數器。

```
23 module decoder_4bit_binary_to_SSD(   // 4-bit binary to SSD pattern decoder
24     output reg [7:0] SSD,            // 7-segment display decode
25     input      [3:0] binary_4bit     // 4-bit binary number
26     );
27
28     // binary (0-9, a, b, c, d, e, f) to 7-segment display decoder
29     always @ (binary_4bit)
30     begin
31         case (binary_4bit)
32             0:       SSD = 8'b0000001_1;  // 0
33             1:       SSD = 8'b1001111_1;  // 1
34             2:       SSD = 8'b0010010_1;  // 2
35             3:       SSD = 8'b0000110_1;  // 3
36             4:       SSD = 8'b1001100_1;  // 4
37             5:       SSD = 8'b0100100_1;  // 5
38             6:       SSD = 8'b0100000_1;  // 6
39             7:       SSD = 8'b0001111_1;  // 7
40             8:       SSD = 8'b0000000_1;  // 8
41             9:       SSD = 8'b0000100_1;  // 9
42             10:      SSD = 8'b0001000_1;  // a
43             11:      SSD = 8'b1100000_1;  // b
44             12:      SSD = 8'b0110001_1;  // c
45             13:      SSD = 8'b1000010_1;  // d
46             14:      SSD = 8'b0110000_1;  // e
47             15:      SSD = 8'b0111000_1;  // f
48             default: SSD = 8'b0000000_0;  // all on
49         endcase
50     end                                  // always
51
52 endmodule                               // decoder_4bit_binary_to_SSD
```

controller for common-anode and individual-cathode of seven-segment display:
有了前面那幾個模組之後，從三十開始倒數到零就只差最後一步了。這個模組主要是為了解決 FPGA 的七段顯示器的那四個 digits 無法同時顯示四個不同 pattern 的問題。我們在這裡可以利用人眼視覺暫留，以足夠快的更新頻率輪流在不同 digit 顯示各個 pattern。實作如下：先使用除頻器將 FPGA 內建的 100M 赫茲時脈處理成適當的頻率後（$100M \div 2^{19}$），連接到一個 1-hot 解碼器與一個多功器，分別用以控制七段顯示器的 anode 與 cathode。

```
23 module SSD_driver(                          // control common-anode and individual-cathode of seven-segment display
24     output reg [7:0]  cathode,              // cathodes of similar segments on all four displays
25     output reg [3:0]  anode,                // anodes of the seven LEDs forming each digit
26     input      [31:0] content,              // SSD pattern to display
27     input             clk_100MHz,           // 100M Hz global clock
28     input             rst_n                 // active low reset
29     );
30     reg [20:0] q_counter;                   // frequency divider
31
32     // frequency divider
33     always @ (posedge clk_100MHz or negedge rst_n)
34     begin
35         if (~rst_n)
36             q_counter <= 0;                 // active low reset
37         else
38             q_counter <= q_counter + 1;     // add 1
39     end
40
41     // anode enable
42     always @ (q_counter[20:19])             // take a suitable refresh frequency
43     begin
44         case (q_counter[20:19])
45             2'b00:   anode <= 4'b1110;      // 1st digit
46             2'b01:   anode <= 4'b1101;      // 2nd digit
47             2'b10:   anode <= 4'b1011;      // 3rd digit
48             2'b11:   anode <= 4'b0111;      // 4th digit
49             default: anode <= 4'b0000;      // all digit
50         endcase
51     end
52
53     // cathode selection
54     always @ (q_counter[20:19])             // take a suitable refresh frequency
55     begin
56         case (q_counter[20:19])
57             2'b00:   cathode <= content[ 7: 0]; // 1st digit
58             2'b01:   cathode <= content[15: 8]; // 2nd digit
59             2'b10:   cathode <= content[23:16]; // 3rd digit
60             2'b11:   cathode <= content[31:24]; // 4th digit
61             default: cathode <= 8'b0000000_0;   // all on
62         endcase
63     end
64
65 endmodule                                  // SSD_driver
```

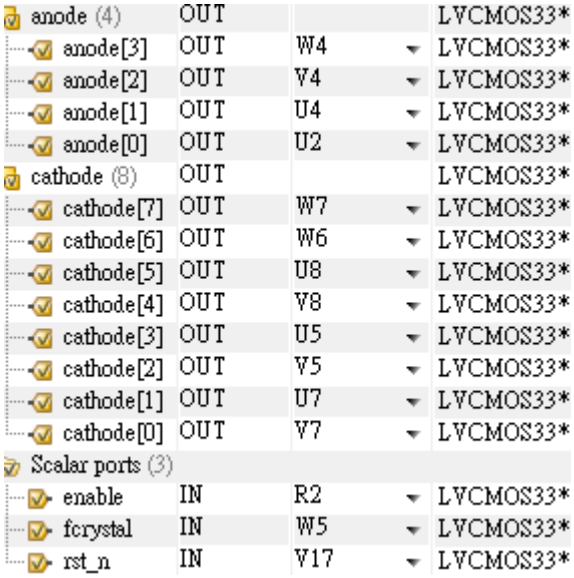
top:

```
23 module LAB4_5(
24     output [7:0] cathode,          // cathodes of similar segments on all four displays
25     output [3:0] anode,            // anodes of the seven LEDs forming each digit
26     input        enable,           // start timing
27     input        fcrystal,         // 100M Hz global clock
28     input        rst_n             // active low reset
29     );
30     wire       clk_1Hz;            // 1 Hz 50% duty clock
31     wire [3:0] bcd1;               // BCD of 1st digit
32     wire [3:0] bcd2;               // BCD of 2nd digit
33     wire       borrow1;            // subtraction carrying
34     wire [7:0] ssd1;               // SSD pattern of 1st digit
35     wire [7:0] ssd2;               // SSD pattern of 2nd digit

        // some modules

85 endmodule                          // LAB4_5
```

最後，只要將剛剛建立好的那些模組串起來就大功告成了。由於三十秒的倒數計時器需要使用到七段顯示器的其中兩個 digits，所以我在下圖第 78 行處將連接到 SSD_driver 的剩下的另外兩個 digits 用 1 補滿。另外，在第 83 行處，我將 decrease_en 設計成僅當有需要下數的時候才將開始下數的訊號傳入第 46 行的計數器。

```
37      frequency_divider_100M FD(                  // 100M Hz to 1 Hz 50% duty frequency divider
38          .fout(clk_1Hz),                         // 1 Hz 50% duty clock
39          .fcrystal(fcrystal),                    // 100M Hz global clock
40          .rst_n(rst_n)                           // active low reset
41          );
42
43      counter_down_4bit COUNT_DIGIT1(             // 4-bit binary down counter for 1st digit
44          .val(bcd1),                             // BCD of 1st digit
45          .borrow(borrow1),                       // subtraction carrying
46          .decrease(decrease_en),
47          .init_val(0),                           // initial value
48          .limit(9),                              // highest possible value
49          .clk(clk_1Hz),                          // 1 Hz 50% duty clock
50          .en(enable),                            // start timing
51          .rst_n(rst_n)                           // active low reset
52          );
53
54      counter_down_4bit COUNT_DIGIT2(             // 4-bit binary down counter for 2nd digit
55          .val(bcd2),                             // BCD of 2nd digit
56          .borrow(),
57          .decrease(borrow1),
58          .init_val(3),                           // initial value
59          .limit(9),                              // highest possible value
60          .clk(clk_1Hz),                          // 1 Hz 50% duty clock
61          .en(enable),                            // start timing
62          .rst_n(rst_n)                           // active low reset
63          );
64
65      decoder_4bit_binary_to_SSD DECODE_DIGIT1(   // 4-bit binary to SSD pattern decoder for 1st digit
66          .SSD(ssd1),                             // SSD pattern of 1st digit
67          .binary_4bit(bcd1)                      // 4-bit binary of 1st digit
68          );
69
70      decoder_4bit_binary_to_SSD DECODE_DIGIT2(   // 4-bit binary to SSD pattern decoder for 2nd digit
71          .SSD(ssd2),                             // SSD pattern of 2nd digit
72          .binary_4bit(bcd2)                      // 4-bit binary of 2nd digit
73          );
74
75      SSD_driver DRIVE_SSD(                       // control common-anode and individual-cathode of seven-segment display
76          .cathode(cathode),                      // cathodes of similar segments on all four displays
77          .anode(anode),                          // anodes of the seven LEDs forming each digit
78          .content({16'b1111111111111111, ssd2, ssd1}),
79          .clk_100MHz(fcrystal),                  // 100M Hz global clock
80          .rst_n(rst_n)                           // active low reset
81          );
82
83      assign decrease_en = ((enable) && ({bcd2, bcd1} != 0));
```

implement:



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*
Scalar ports (3)			
enable	IN	R2	LVCMOS33*
fcrystal	IN	W5	LVCMOS33*
rst_n	IN	V17	LVCMOS33*

conclusion:

這一題要做的是一個三十秒的倒數計時器。這一題與前幾題相較之下複雜許多，考驗分析問題的能力。如果碰到像這種功能較複雜的題目，可以先試著將大問題分解成數個小問題，在針對各個小問題構造出對應的模組，最後再整合起來。雖然寫硬體語言和寫軟體語言有很大的不同，但卻有類似的解題思路：例如寫軟體時可以把大問題解剖成數個小問題，再針對各個小問題寫出 function，最後用 main()整合起來。