

Lab 4 - Counters and Shifters II

106033233 資工大四 周聖諺 (Sheng-Yen Chou)

2022-03-28



Contents

Lab 4 - Counters and Shifters II Report	3
Lab 4 - 1: 1Hz 4-bit Synchronous Binary Down Counter	3
Design Specification	3
Design Implementation	3
Lab 4 - 2: 1Hz 4-bit Synchronous Binary Down Counter To 7-Segment Display	7
Design Specification	7
Design Implementation	7
Lab 4 - 3: 0.5Hz 4-bit Synchronous Binary Down Counter To 7-Segment Display	10
Design Specification	10
Design Implementation	11
Lab 4 - 4: 1Hz 8-bit Synchronous Binary Up Counter To 7-Segment Display	14
Design Specification	14
Design Implementation	15

Lab 4 - Counters and Shifters II Report

106033233 資工大四 周聖諺 (Sheng-Yen Chou)

Lab 4 - 1: 1Hz 4-bit Synchronous Binary Down Counter

Design Specification

Source Code

Frequency Divider

Input: rst, clk

Output: clk_out

4-bit Synchronous Binary Down Counter

Input: rst, clk

Output [3:0]q

1Hz 4-bit Synchronous Binary Down Counter

Input: rst, clk

Output [3:0]q

Design Implementation

Frequency Divider

To generate the 1 Hz clock, I use variables counter_in and counter_out to count from 0 to 50M. The counter_in will store the value for the next time step and pass the value to the counter_out when the clock raises. The reason why we need 50M counting is each counting is triggered only when the clock raises, so the circuit will count 1 more for every twice clock pulses.

```
1 `define FREQ_DIV_BITS 30
2 //`define FREQ_DIV_COUNT `FREQ_DIV_BITS'd10000000
3 `define FREQ_DIV_COUNT `FREQ_DIV_BITS'd50000000
4
5 module frequency_divider(
6     clk_out,
7     // counter,
```

```

8     clk,
9     rst
10    );
11
12    input clk;
13    input rst;
14    output clk_out;
15    //    output counter;
16
17    reg clk_in;
18    reg clk_out;
19    reg [`FREQ_DIV_BITS-1:0] counter_in;
20    reg [`FREQ_DIV_BITS-1:0] counter_out;
21
22    always@(counter_out or clk_out)
23        if(counter_out < (`FREQ_DIV_COUNT - 1))
24            begin
25                counter_in <= counter_out + `FREQ_DIV_BITS'd1;
26                clk_in <= clk_out;
27            end
28        else
29            begin
30                counter_in <= `FREQ_DIV_BITS'd0;
31                clk_in <= ~clk_out;
32            end
33
34    always@(posedge clk or negedge rst)
35        if(~rst)
36            begin
37                counter_out <= `FREQ_DIV_BITS'd0;
38                clk_out <= 1'd0;
39            end
40        else
41            begin
42                counter_out <= counter_in;
43                clk_out <= clk_in;
44            end
45    endmodule

```

4-bit Synchronous Binary Down Counter

To implement the binary down counter, I use a variable `q_in` to count from 0 to 15. Whenever the output of the counter `q` changes, the variable `q_in` should be changed to `q - 1`. In addition, when the circuit detects the raise of the clock, the output of the counter will change to the variable `q_in`. On the other hand, if the reset switch to 0 or the counter hits 0, `q` will be reset to the upper limit 15.

Verilog Code

```

1  `define BCD_COUNTER_BITS 4
2

```

```

3  module binary_down_counter(
4      q,
5      clk,
6      rst
7  );
8
9      output [`BCD_COUNTER_BITS-1:0]q;
10     input clk;
11     input rst;
12
13     reg [`BCD_COUNTER_BITS-1:0]q;
14     reg [`BCD_COUNTER_BITS-1:0]q_in;
15
16     always@(q)
17     begin
18         q_in <= q - `BCD_COUNTER_BITS'd1;
19     end
20
21     always@(posedge clk or negedge rst)
22     begin
23         if(~rst)
24             begin
25                 q <= `BCD_COUNTER_BITS'd0;
26             end
27         else
28             begin
29                 q <= q_in;
30             end
31     end
32 endmodule

```

1Hz 4-bit Synchronous Binary Down Counter

All we need to do is combine the 1 Hz frequency divider and the 4-bit binary down counter which triggered by the 1 Hz frequency divider.

```

1  `define BCD_COUNTER_BITS 4
2  `define RST_HIGH 1'b1
3
4  module lab4_1(
5      q,
6      rst,
7      clk
8  );
9      output [`BCD_COUNTER_BITS-1:0]q;
10     input rst;
11     input clk;
12
13     // reg [`BCD_COUNTER_BITS-1:0]q;
14     wire DIV_CLK;
15

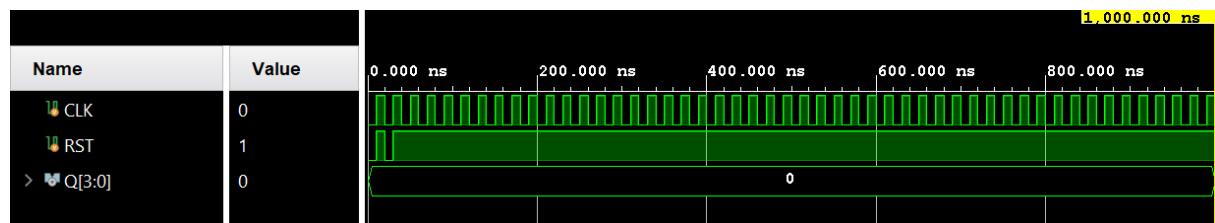
```

I/O Pin Assignment

I/O	clk	rst	q[0]	q[1]	q[2]	q[3]
LOC	W5	V17	U16	E19	U19	V19

[illegible]

RTL Simulation



106033233 資工大四 周聖諺 (Sheng-Yen Chou)

Lab 4 - 2: 1Hz 4-bit Synchronous Binary Down Counter To 7-Segment Display**Design Specification**

Source Code

Frequency Divider

Input: rst, clk

Output: clk_out

4-bit Synchronous Binary Down Counter

Input: rst, clk

Output [3:0]q

1Hz 4-bit Synchronous Binary Down Counter

Input: rst, clk

Output [3:0]q

Binary to 7-Segment Display

Input [3:0] i

Output [3:0]P, [7:0]D

1Hz 4-bit Synchronous Binary Down Counter To 7-Segment Display

Input rst, clk

Output [3:0]q, [3:0]P, [7:0]D

Design Implementation**Frequency Divider**

Same as lab 4-1.

4-bit Synchronous Binary Down Counter

Same as lab 4-1.

1Hz 4-bit Synchronous Binary Down Counter

Same as lab 4-1.

Binary to 7-Segment Display

Convert 4-bit binary number to 7-segment display with switch-case syntax.

```

1  `define INPUT_BITS_N 4
2  `define SEGMENT_7_DISPALY_DIGIT_N 4
3  `define SEGMENT_7_SEGMENT_N 8
4
5  module segment7(
6      input [`INPUT_BITS_N:0] i,
7      output [`SEGMENT_7_DISPALY_DIGIT_N-1:0] P,
8      output [`SEGMENT_7_SEGMENT_N-1:0] D
9  );
10
11     reg [`SEGMENT_7_SEGMENT_N-1:0] D;
12
13     assign P = ~4'b0001;
14     always@(i)
15         case(i)
16             4'd0: D=`SEGMENT_7_SEGMENT_N'b0000001_1;
17             4'd1: D=`SEGMENT_7_SEGMENT_N'b1001111_1;
18             4'd2: D=`SEGMENT_7_SEGMENT_N'b0010010_1;
19             4'd3: D=`SEGMENT_7_SEGMENT_N'b0000110_1;
20             4'd4: D=`SEGMENT_7_SEGMENT_N'b1001100_1;
21             4'd5: D=`SEGMENT_7_SEGMENT_N'b0100100_1;
22             4'd6: D=`SEGMENT_7_SEGMENT_N'b0100000_1;
23             4'd7: D=`SEGMENT_7_SEGMENT_N'b0001111_1;
24             4'd8: D=`SEGMENT_7_SEGMENT_N'b0000000_1;
25             4'd9: D=`SEGMENT_7_SEGMENT_N'b0000100_1;
26             4'd10: D=`SEGMENT_7_SEGMENT_N'b0001000_1;
27             4'd11: D=`SEGMENT_7_SEGMENT_N'b1100000_1;
28             4'd12: D=`SEGMENT_7_SEGMENT_N'b0110001_1;
29             4'd13: D=`SEGMENT_7_SEGMENT_N'b1000010_1;
30             4'd14: D=`SEGMENT_7_SEGMENT_N'b0110000_1;
31             4'd15: D=`SEGMENT_7_SEGMENT_N'b0111000_1;
32             default: D=`SEGMENT_7_SEGMENT_N'b0111000_1;
33         endcase
34
35     endmodule

```

1Hz 4-bit Synchronous Binary Down Counter To 7-Segment Display

It's a simple module and just use the divided clock from frequency divider to trigger the binary down counter. The counting of the down counter will be shown in 7-segment display.

```

1  `define BCD_COUNTER_BITS 4
2  `define RST_HIGH 1'b1
3  `define SEGMENT_7_DISPALY_DIGIT_N 4
4  `define SEGMENT_7_SEGMENT_N 8

```



```

5
6  module lab4_2(
7      q,
8      P,
9      D,
10     rst,
11     clk
12 );
13     output [`BCD_COUNTER_BITS-1:0]q;
14     output [`SEGMENT_7_DISPALY_DIGIT_N-1:0]P;
15     output [`SEGMENT_7_SEGMENT_N-1:0]D;
16     input rst;
17     input clk;
18
19     // reg [`BCD_COUNTER_BITS-1:0]q;
20     wire DIV_CLK;
21
22     assign P = 4'b1110;
23
24     frequency_divider U0(.clk(clk), .rst(rst), .clk_out(DIV_CLK));
25     binary_down_counter U1(.clk(DIV_CLK), .rst(rst), .q(q));
26     segment7 U2(.i(q), .P(P), .D(D));
27 endmodule

```

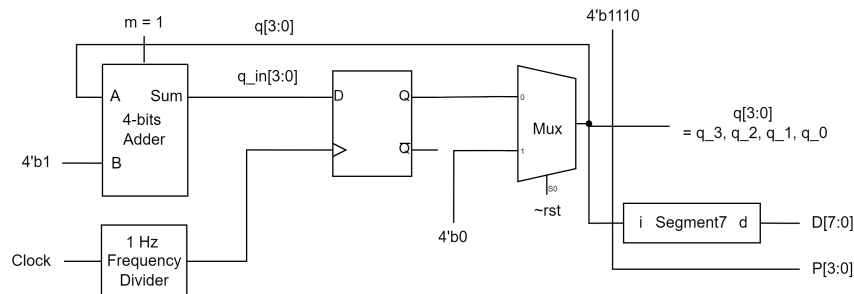
I/O Pin Assignment

I/O	clk	rst	q[0]	q[1]	q[2]	q[3]	P[0]	P[1]	P[2]	P[3]
LOC	W5	V17	U16	E19	U19	V19	U2	U4	V4	W4

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

Block Diagram

Figure 3: Lab 4-2 Logic Diagram



Name	Value
CLK	0
RST	1
> Q[3:0]	0
> P[3:0]	-2
> D[7:0]	113

Figure 4: Lab 4-2 RTL Simulation

Input: rst, clk

Output [3:0]q

0.5Hz 4-bit Synchronous Binary Down Counter

Input: rst, clk

Output [3:0]q

Binary to 7-Segment Display

Input [3:0] i

Output [3:0]P, [7:0]D

0.5Hz 4-bit Synchronous Binary Down Counter To 7-Segment Display

Input rst, clk

Output [3:0]q, [3:0]P, [7:0]D

Design Implementation

Frequency Divider

It's similar to lab 4-2. I only modify the upper limit of the counter to 100M, which means twice divide the frequency.

```

1  `define  FREQ_DIV_BITS 30
2  //`define  FREQ_DIV_COUNT `FREQ_DIV_BITS'd10000000
3  //`define  FREQ_DIV_COUNT `FREQ_DIV_BITS'd500000000
4  `define  FREQ_DIV_COUNT `FREQ_DIV_BITS'd100000000
5
6  module frequency_divider(
7      clk_out,
8      clk,
9      rst
10 );
11
12     input clk;
13     input rst;
14     output clk_out;
15
16     reg clk_in;
17     reg clk_out;
18     reg [`FREQ_DIV_BITS-1:0] counter_in;
19     reg [`FREQ_DIV_BITS-1:0] counter_out;
20
21     always@(counter_out or clk_out)
22         if(counter_out < (`FREQ_DIV_COUNT - 1))
23             begin
24                 counter_in <= counter_out + `FREQ_DIV_BITS'd1;

```

```

25         clk_in <= clk_out;
26     end
27     else
28     begin
29         counter_in <= `FREQ_DIV_BITS'd0;
30         clk_in <= ~clk_out;
31     end
32
33     always@(posedge clk or negedge rst)
34     if(~rst)
35     begin
36         counter_out <= `FREQ_DIV_BITS'd0;
37         clk_out <= 1'd0;
38     end
39     else
40     begin
41         counter_out <= counter_in;
42         clk_out <= clk_in;
43     end
44 endmodule

```

4-bit Synchronous Binary Down Counter

Same as lab 4-2.

1Hz 4-bit Synchronous Binary Down Counter

Same as lab 4-2.

Binary to 7-Segment Display

Same as lab 4-2.

0.5Hz 4-bit Synchronous Binary Down Counter To 7-Segment Display

We simply combine the previous 3 modules and we can

```

1  `define BCD_COUNTER_BITS 4
2  `define RST_HIGH 1'b1
3  `define SEGMENT_7_DISPALY_DIGIT_N 4
4  `define SEGMENT_7_SEGMENT_N 8
5
6  module lab4_3(
7      q,
8      P,
9      D,
10     rst,
11     clk
12 );
13     output [`BCD_COUNTER_BITS-1:0]q;
14     output [`SEGMENT_7_DISPALY_DIGIT_N-1:0]P;
15     output [`SEGMENT_7_SEGMENT_N-1:0]D;

```

```

16     input rst;
17     input clk;
18
19     // reg [`BCD_COUNTER_BITS-1:0]q;
20     wire DIV_CLK;
21
22     assign P = 4'b1110;
23
24     frequency_divider U0(.clk(clk), .rst(rst), .clk_out(DIV_CLK));
25     binary_down_1digit_counter U1(.clk(DIV_CLK), .rst(rst), .q(q));
26     segment7 U2(.i(q), .P(P), .D(D));
27 endmodule

```

I/O Pin Assignment

I/O	clk	rst	q[0]	q[1]	q[2]	q[3]	P[0]	P[1]	P[2]	P[3]
LOC	W5	V17	U16	E19	U19	V19	U2	U4	V4	W4

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

Block Diagram

Lab 4 - 3: 0.5 Hz 4-bit Synchronous Binary Down Counter To 7-Segment Display

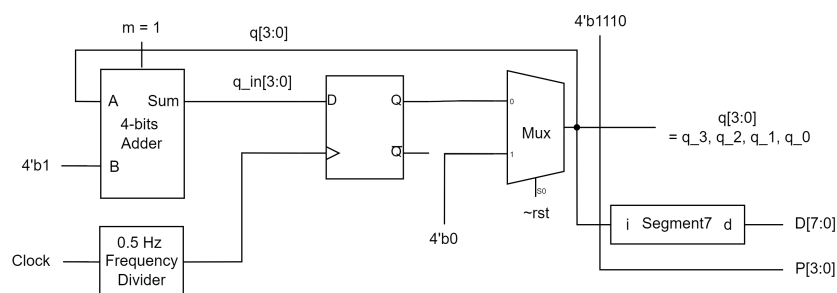


Figure 5: Lab 4-3 Logic Diagram

RTL Simulation

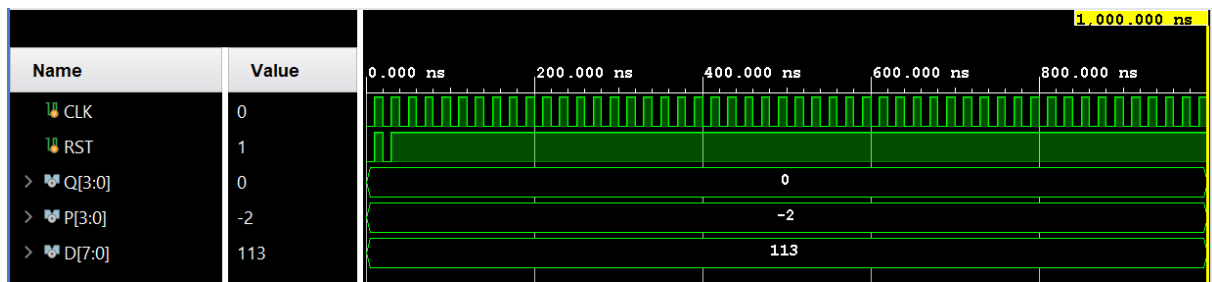


Figure 6: Lab 4-3 RTL Simulation

Lab 4 - 4: 1Hz 8-bit Synchronous Binary Up Counter To 7-Segment Display

Design Specification

Source Code

Frequency Divider

Input: rst, clk

Output: clk_out

8-bit Synchronous Binary Up Counter

Input: rst, clk

Output [7:0]q

1Hz 8-bit Synchronous Binary Up Counter

Input: rst, clk

Output [7:0]q

Extractor

Input [7:0] x

Output [3:0] d1, [3:0] d2

Binary to 7-Segment Display

Input [3:0] i

Output [3:0]P, [7:0]D

7-Segment Display

Output [0:3]d_sel, [7:0]d_out

Input clk, rst, [7:0]d0, [7:0]d1, [7:0]d2, [7:0]d3

1Hz 8-bit Synchronous Binary Up Counter To 7-Segment Display

Input rst, clk

Output [3:0]q, [3:0]P, [7:0]D

Design Implementation

Frequency Divider

Same as lab 4-1.

8-bit Synchronous Binary Up Counter

It's similar to the 4-bit binary down counter in lab 4-1. I only extend the bits array of the counter to 8 bits and use plus 1 instead of minus 1.

1Hz 8-bit Synchronous Binary Up Counter

It's similar to the 1 Hz 4-bit binary down counter in lab 4-1. I only extend the bits array of the counter to 8 bits and use plus 1 instead of minus 1.

Extractor

I use mod of 10 to extract the first decimal digit and use divided by 10 to extract the second decimal digit.

```
1 module extract(  
2     input [7:0] x,  
3     output [3:0] d1,  
4     output [3:0] d2  
5 );  
6  
7     wire [7:0]mod;  
8     wire [7:0]div;  
9     assign mod = x % 10;  
10    assign div = x / 10;  
11  
12    assign d1 = mod[3:0];  
13    assign d2 = div[3:0];  
14 endmodule
```

Binary to 7-Segment Display

Same as lab4-2.

7-Segment Display Controller

Since we can only control one digit of the 7-segment display each time, I design a module that takes the 4-digit patterns as input and shows the 1 digit on the display when the clock raises. Whenever the clock raises, the module will switch the control `d_sel` to different digit and shows the corresponding digit. Take an example, when the first clock raise occur, the module will set `d_sel = 4' b1110` and `d_out = d0`. As for second clock pulse, the module will output `d_sel = 4' b1101` and `d_out = d1` and so on.

```

1  `define DIGIT_N 4
2  `define SEGMENT_N 8
3  `define NONE_BITS `SEGMENT_N'b1111111_0
4  `define EMPTY_BITS `SEGMENT_N'b1111111_1
5
6  module display_7seg(
7      d_sel,
8      d_out,
9      clk,
10     rst,
11     d0,
12     d1,
13     d2,
14     d3
15 );
16
17     output [0:`DIGIT_N-1]d_sel;
18     output [`SEGMENT_N-1:0]d_out;
19     input clk;
20     input rst;
21     input [`SEGMENT_N-1:0]d0;
22     input [`SEGMENT_N-1:0]d1;
23     input [`SEGMENT_N-1:0]d2;
24     input [`SEGMENT_N-1:0]d3;
25
26     reg [0:`DIGIT_N-1]d_sel;
27     reg [`SEGMENT_N-1:0]d_out;
28     reg [0:`DIGIT_N-1]d_sel_temp;
29     reg [`SEGMENT_N-1:0]d_out_temp;
30     wire clk_out;
31
32     //    initial
33     //    begin
34     //        d_sel_temp <= `DIGIT_N'b1110;
35     //        d_out_temp <= `EMPTY_BITS;
36     //    end
37
38     segment7_frequency_divider U0(.clk(clk), .rst(rst), .clk_out(
39         clk_out));
40
41     always@(d_sel)
42     begin

```



```

42         case((d_sel << 1) | (d_sel >> (`DIGIT_N-1)))
43             `DIGIT_N'b1110: d_out_temp <= d0;
44             `DIGIT_N'b1101: d_out_temp <= d1;
45             `DIGIT_N'b1011: d_out_temp <= d2;
46             `DIGIT_N'b0111: d_out_temp <= d3;
47             default: d_out_temp <= `NONE_BITS;
48         endcase
49         d_sel_temp <= (d_sel << 1) | (d_sel >> (`DIGIT_N-1));
50     end
51
52     always@(posedge clk_out or negedge rst)
53     begin
54         if(~rst)
55             begin
56                 d_out <= `EMPTY_BITS;
57                 d_sel <= `DIGIT_N'b1110;
58             end
59         else
60             begin
61                 d_out <= d_out_temp;
62                 d_sel <= d_sel_temp;
63             end
64         end
65     end
66 endmodule

```

1Hz 8-bit Synchronous Binary Up Counter To 7-Segment Display

I combine the all modules mentioned above. First, I generate a 1 Hz clock to trigger the 8-bit up counter and output the number of counting. Then, extract the both decimal digits of the binary number of counting. Finally, show the decimal digits on the 7-segment display.

```

1  `define BCD_COUNTER_BITS 8
2  `define RST_HIGH 1'b1
3
4  `define INPUT_BITS_N 4
5  `define SEGMENT_7_DISPALY_DIGIT_N 4
6  `define SEGMENT_7_SEGMENT_N 8
7
8  `define P 4'b1111
9  `define NONE_SEG7 `SEGMENT_7_SEGMENT_N'b1111111_1
10
11 module lab4_4(
12     q,
13     D_SEL,
14     D_OUT,
15     rst,
16     clk
17 );
18     output [`BCD_COUNTER_BITS-1:0]q;
19     output [`SEGMENT_7_DISPALY_DIGIT_N-1:0]D_SEL;

```

```

20     output [`SEGMENT_7_SEGMENT_N-1:0]D_OUT;
21     input rst;
22     input clk;
23
24     // reg [`BCD_COUNTER_BITS-1:0]q;
25     wire DIV_CLK;
26     wire [`INPUT_BITS_N-1:0]D1_BINARY;
27     wire [`INPUT_BITS_N-1:0]D2_BINARY;
28     wire [`SEGMENT_7_SEGMENT_N-1:0]D1_SEGMENT7;
29     wire [`SEGMENT_7_SEGMENT_N-1:0]D2_SEGMENT7;
30
31     // 2-Digits Binary up counter
32     frequency_divider U0(.clk(clk), .rst(rst), .clk_out(DIV_CLK));
33     binary_up_2digit_counter U1(.clk(DIV_CLK), .rst(rst), .q(q));
34
35     // Extract digits
36     extract U2(.x(q), .d1(D1_BINARY), .d2(D2_BINARY));
37
38     // Convert binary to 7-segment
39     segment7 U3(.i(D1_BINARY), .D(D1_SEGMENT7));
40     segment7 U4(.i(D2_BINARY), .D(D2_SEGMENT7));
41
42     // Show
43     display_7seg U5(.clk(clk), .rst(rst), .d0(D1_SEGMENT7), .d1(
        D2_SEGMENT7), .d2(`NONE_SEG7), .d3(`NONE_SEG7), .d_sel(D_SEL), .
        d_out(D_OUT));
44 endmodule

```

I/O Pin Assignment

I/O	clk	rst	q[0]	q[1]	q[2]	q[3]	P[0]	P[1]	P[2]	P[3]
LOC	W5	V17	U16	E19	U19	V19	U2	U4	V4	W4

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

Block Diagram

RTL Simulation

