


---

## Lab 3 - Counters and Shifters I Report

106033233 資工大四 周聖諺 (Sheng-Yen Chou)

2022-03-22



## Contents

<b>Lab 3 - Counters and Shifters I Report</b>	<b>3</b>
Lab 3 - Pre 1: 4-bit Synchronous Binary Up Counter . . . . .	3
Design Specification . . . . .	3
Design Implementation . . . . .	3
Lab 3 - Pre 2: 8-Cascaded Shift Registers . . . . .	5
Design Specification . . . . .	5
Design Implementation . . . . .	5
Lab 3 - 1: $1/2^{27}$ Frequency Divider . . . . .	7
Design Specification . . . . .	7
Design Implementation . . . . .	7
Lab 3 - 2: 1Hz Count-for-50M Frequency Divider . . . . .	8
Design Specification . . . . .	8
Design Implementation . . . . .	9
Lab 3 - 3: 1Hz 4-bit Synchronous Binary Up Counter . . . . .	10
Design Specification . . . . .	10
Design Implementation . . . . .	11
Lab 3 - 4: 1 Hz 8-Cascaded Shift Registers . . . . .	12
Design Specification . . . . .	12
Design Implementation . . . . .	13
Lab 3 - 5: 1 Hz 10 Digit Marquee on 7-Segment Display . . . . .	14
Design Specification . . . . .	14
Design Implementation . . . . .	15

## Lab 3 - Counters and Shifters I Report

106033233 資工大四 周聖諺 (Sheng-Yen Chou)

---

### Lab 3 - Pre 1: 4-bit Synchronous Binary Up Counter

#### Design Specification

Source Code

#### 4-bit Synchronous Binary Up Counter

Input: rst, clk

Output [3:0]q

#### Design Implementation

To implement the binary up 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 hit the upper limit(15), q will be reset to 0.

#### Verilog Code

```
1  `define BCD_COUNTER_BITS 4
2
3  module binary_up_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;
```

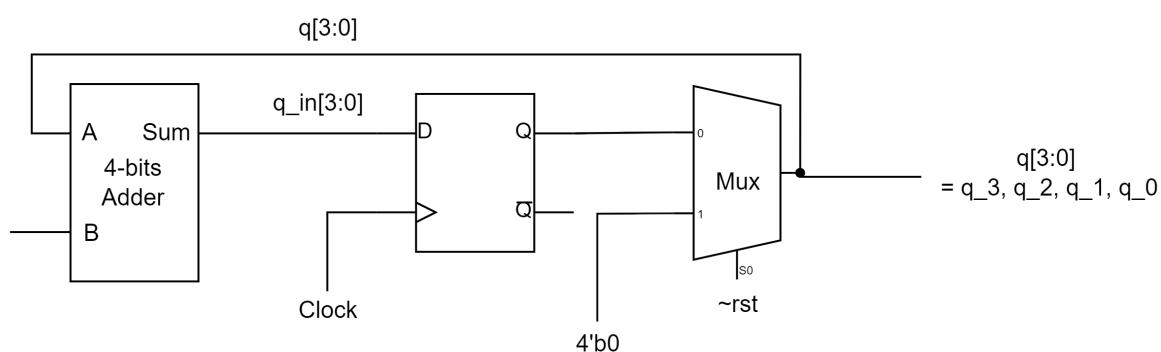
```

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

```

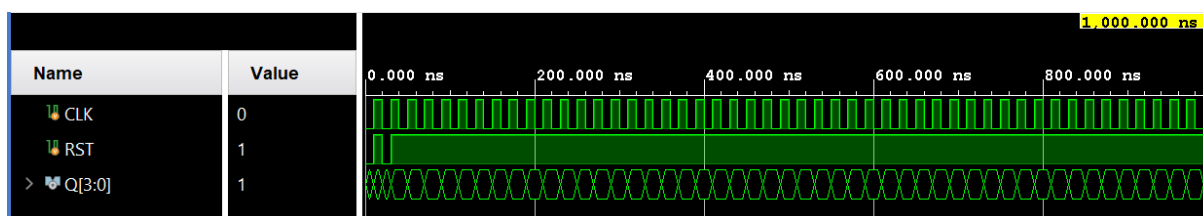
### Block Diagram

## Lab 3 - Pre 1: Binary Up Counter



**Figure 1:** Lab 3-Pre1 Logic Diagram

### RTL Simulation



**Figure 2:** Lab 2-Pre1 RTL Simulation

## Lab 3 - Pre 2: 8-Cascaded Shift Registers

### Design Specification

Source Code

#### D Flip Flop

Input d, clk, rst

Output q

#### 8-Cascaded Shift Registers

Input: rst, clk

Output [7:0]q

### Design Implementation

#### D Flip Flop

I use a variable d to store the input and q as the output of the flip flop. Whenever the clock is raised, the output variable q will be updated with the input variable d. In addition, when the reset is triggered, the output q will be 0.

#### 8-Cascaded Shift Registers

I use a variable q with 8 entries to store the value. Whenever the clock is raised, it will shift the value in the previous register to the next one and the last one will be shift to the first register.

### Verilog Code

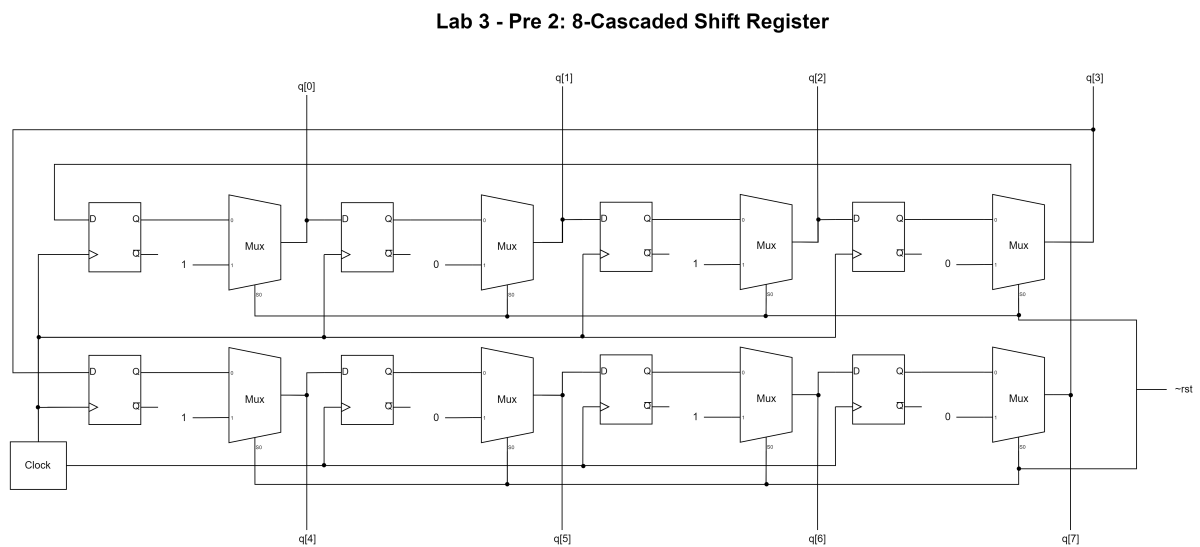
```
1  `define BIT_WIDTH 8
2
3  module shifter(
4      q,
5      clk,
6      rst
7  );
8
9      output [`BIT_WIDTH-1:0]q;
10     input clk;
11     input rst;
12
13     reg [`BIT_WIDTH-1:0]q;
14
15     always@(posedge clk or negedge rst)
16     begin
```

```

17     if(~rst)
18     begin
19         q <= `BIT_WIDTH'b01010101;
20     end
21     else
22     begin
23         q[0] <= q[7];
24         q[1] <= q[0];
25         q[2] <= q[1];
26         q[3] <= q[2];
27         q[4] <= q[3];
28         q[5] <= q[4];
29         q[6] <= q[5];
30         q[7] <= q[6];
31     end
32 end
33 endmodule

```

### Block Diagram



**Figure 3:** Lab 3-Pre2 Logic Diagram

### RTL Simulation

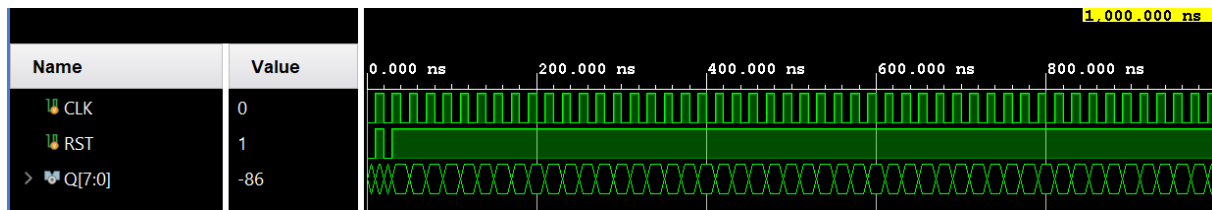


Figure 4: Lab 2-Pre2 RTL Simulation

### Lab 3 - 1: $1/2^{27}$ Frequency Divider

#### Design Specification

Source Code

Input: rst, clk

Output: clk\_out

#### Design Implementation

To implement the frequency divider, I use a variable with 26 bits to count from 0 to  $1^{27} - 1$ . and then back to 0 while the variable hits  $1^{27} - 1$ .

Actually,  $1/2^{27}$  frequency divider can also be implemented by cascading 26 D-type flip flop. Whenever we cascade one flip flop, the frequency can be divided by 2.

#### Verilog Code

```

1  `define FREQ_DIV_BITS 1
2
3  module lab3_1(
4      clk_out,
5      clk,
6      rst
7  );
8
9      output clk_out;
10     input clk;
11     input rst;
12
13     reg clk_out;
14     reg [`FREQ_DIV_BITS-1:0]count_out;
15     reg [`FREQ_DIV_BITS:0]count;
16
17     always@(clk_out or count_out)

```

```

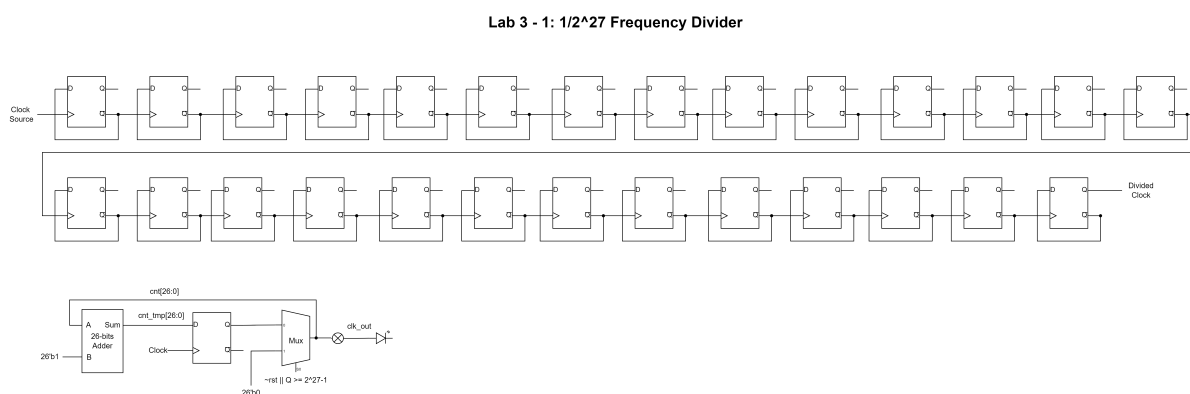
18     begin
19         count = {clk_out, count_out} + `FREQ_DIV_BITS'd1;
20     end
21
22     always@(posedge clk or negedge rst)
23     begin
24         if(~rst)
25             begin
26                 {clk_out, count_out} = `FREQ_DIV_BITS'd0;
27             end
28         else
29             begin
30                 {clk_out, count_out} = count;
31             end
32         end
33     endmodule

```

### I/O Pin Assignment

I/O	clk	rst	clk_out
LOC	W5	V17	U16

### Block Diagram



**Figure 5:** Lab 3-1 Logic Diagram

### Lab 3 - 2: 1Hz Count-for-50M Frequency Divider

#### Design Specification

#### Source Code



Input: rst, clk

Output: clk\_out

### Design Implementation

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.

### Verilog Code

```
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 lab3_2(
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
```

## I/O Pin Assignment

### Block Diagram

### Lab 3 - 3: 1Hz 4-bit Synchronous Binary Up Counter

### 4-bit Synchronous Binary Up Counter

Input: rst, clk

Output [3:0]q;

### 1Hz 4-bit Synchronous Binary Up Counter

Input: rst, clk

Output [3:0]q;

## Design Implementation

### Frequency Divider

Same as Lab3-2.

### 4-bit Synchronous Binary Up Counter

Same as Lab3-pre1.

### 1Hz 4-bit Synchronous Binary Up Counter

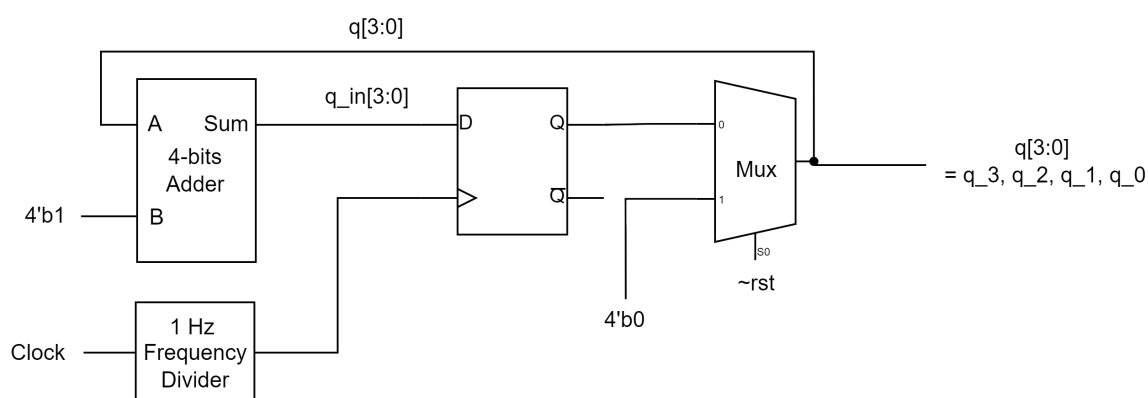
All we need to do is combine the 1 Hz frequency divider and the 4-bit binary up counter which triggered by the 1 Hz frequency divider.

## Verilog Code

```
1  `define BCD_COUNTER_BITS 4
2  `define RST_HIGH 1'b1
3
4  module lab3_3(
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
16     frequency_divider U0(.clk(clk), .rst(rst), .clk_out(DIV_CLK));
17     binary_up_counter U1(.clk(DIV_CLK), .rst(rst), .q(q));
18 endmodule
```

## I/O Pin Assignment

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

**Block Diagram****Lab 3 - 3: 1 Hz Binary Up Counter****Figure 7:** Lab 3-3 Logic Diagram**Lab 3 - 4: 1 Hz 8-Cascaded Shift Registers****Design Specification**

Source Code

**Frequency Divider**

Input: rst, clk

Output: clk\_out

**8-Cascaded Shift Registers**

Input: rst, clk

Output [7:0]q

**1 Hz 8-Cascaded Shift Registers**

Input: rst, clk

Output [7:0]q

## Design Implementation

### Frequency Divider

Same as Lab3-2.

### 8-Cascaded Shift Registers

Same as Lab3-Pre2.

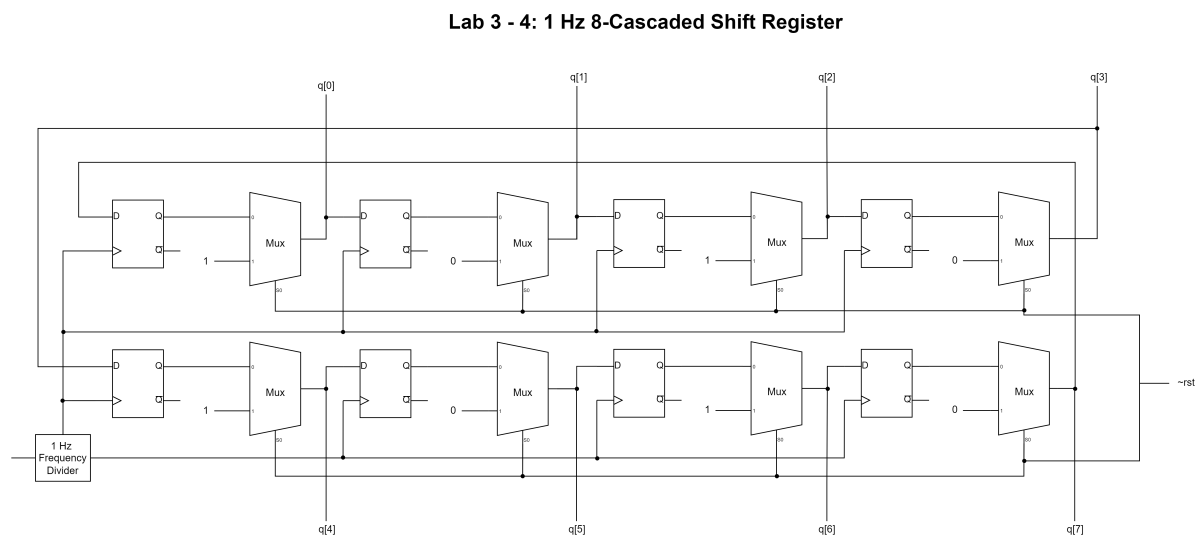
### 1 Hz 8-Cascaded Shift Registers

We can achieve this functionality with the 1 Hz frequency divider as the clock trigger of the 8-cascaded shift registers.

### Verilog Code

```
1  `define BIT_WIDTH 8
2
3  module shift_register(
4      q,
5      clk,
6      rst
7  );
8
9      output [`BIT_WIDTH-1:0]q;
10     input clk;
11     input rst;
12
13     wire [`BIT_WIDTH-1:0]q;
14     wire CLK_OUT;
15
16     frequency_divider U0(.clk(clk), .rst(rst), .clk_out(CLK_OUT));
17     shifter U1(.clk(CLK_OUT), .rst(rst), .q(q));
18 endmodule
```

### Block Diagram

**Figure 8:** Lab 3-4 Logic Diagram**I/O Pin Assignment**

I/O	clk	rst	q[0]	q[1]	q[2]	q[3]	q[4]	q[5]	q[6]	q[7]
LOC	W5	V17	U16	E19	U19	V19	W18	U15	U14	V14

**Lab 3 - 5: 1 Hz 10 Digit Marquee on 7-Segment Display****Design Specification**

Source Code

**Frequency Divider**

Input: rst, clk

Output: clk\_out

**10-Cascaded Shift Registers**

Input: rst, clk

Output [9:0]q

**1Hz 10-Cascaded Shift Registers**

Input: rst, clk

Output [9:0]q

### 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

### 1 Hz 10 Digit Marquee on 7-Segment Display

Output [0:3]d\_sel, [7:0]d\_out

Input clk, rst

## Design Implementation

### Frequency Divider

Same as Lab3-2.

### 10-Cascaded Shift Registers

Simply extend the 8-digit shift registers in the Lab3-Pre2 to 10-digit. Each digit contains 8 bits to store the pattern shown on the 7-segment display.

### 1Hz 10-Cascaded Shift Registers

Simply extend the 8-digit shift register in Lab3-4 to the 10-digit one.

## Verilog Code

```
1  `define SHIFTER_WIDTH 10
2  `define REG_SIZE 8
3
4  module shift_register(
5      q0, q1, q2, q3, q4, q5, q6, q7, q8, q9,
6      clk,
7      rst
8  );
9
10     output [`REG_SIZE-1:0]q0;
11     output [`REG_SIZE-1:0]q1;
12     output [`REG_SIZE-1:0]q2;
13     output [`REG_SIZE-1:0]q3;
14     output [`REG_SIZE-1:0]q4;
15     output [`REG_SIZE-1:0]q5;
16     output [`REG_SIZE-1:0]q6;
17     output [`REG_SIZE-1:0]q7;
18     output [`REG_SIZE-1:0]q8;
19     output [`REG_SIZE-1:0]q9;
20     input clk;
```

```

21     input rst;
22
23     wire [`REG_SIZE-1:0]q0;
24     wire [`REG_SIZE-1:0]q1;
25     wire [`REG_SIZE-1:0]q2;
26     wire [`REG_SIZE-1:0]q3;
27     wire [`REG_SIZE-1:0]q4;
28     wire [`REG_SIZE-1:0]q5;
29     wire [`REG_SIZE-1:0]q6;
30     wire [`REG_SIZE-1:0]q7;
31     wire [`REG_SIZE-1:0]q8;
32     wire [`REG_SIZE-1:0]q9;
33     wire CLK_OUT;
34
35     frequency_divider U0(.clk(clk), .rst(rst), .clk_out(CLK_OUT));
36     shifter U1(.clk(CLK_OUT), .rst(rst), .q0(q0), .q1(q1), .q2(q2), .q3
        (q3), .q4(q4), .q5(q5), .q6(q6), .q7(q7), .q8(q8), .q9(q9));
37 endmodule

```

## 7-Segment Display

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.

## Verilog Code

```

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     segment7_frequency_divider U0(.clk(clk), .rst(rst), .clk_out(
33         clk_out));
34
35     always@(d_sel)
36     begin
37         case((d_sel << 1) | (d_sel >> (`DIGIT_N-1)))
38             `DIGIT_N'b1110: d_out_temp <= d0;
39             `DIGIT_N'b1101: d_out_temp <= d1;
40             `DIGIT_N'b1011: d_out_temp <= d2;
41             `DIGIT_N'b0111: d_out_temp <= d3;
42             default: d_out_temp <= `NONE_BITS;
43         endcase
44         d_sel_temp <= (d_sel << 1) | (d_sel >> (`DIGIT_N-1));
45     end
46
47     always@(posedge clk_out or negedge rst)
48     begin
49         if(~rst)
50         begin
51             d_out <= `EMPTY_BITS;
52             d_sel <= `DIGIT_N'b1110;
53         end
54         else
55         begin
56             d_out <= d_out_temp;
57             d_sel <= d_sel_temp;
58         end
59     end
60 endmodule

```

### 1 Hz 10 Digit Marquee on 7-Segment Display

To implement the marquee, just simply combine the 1 Hz 10-cascaded shift registers and the 7-segment display. The 1 Hz 10-cascaded shift registers will store the patterns of each time step and shift the digits when the clock raises. Then I inject the digit 0 ~ 4 into the 7-segment display module to show the patterns.

```

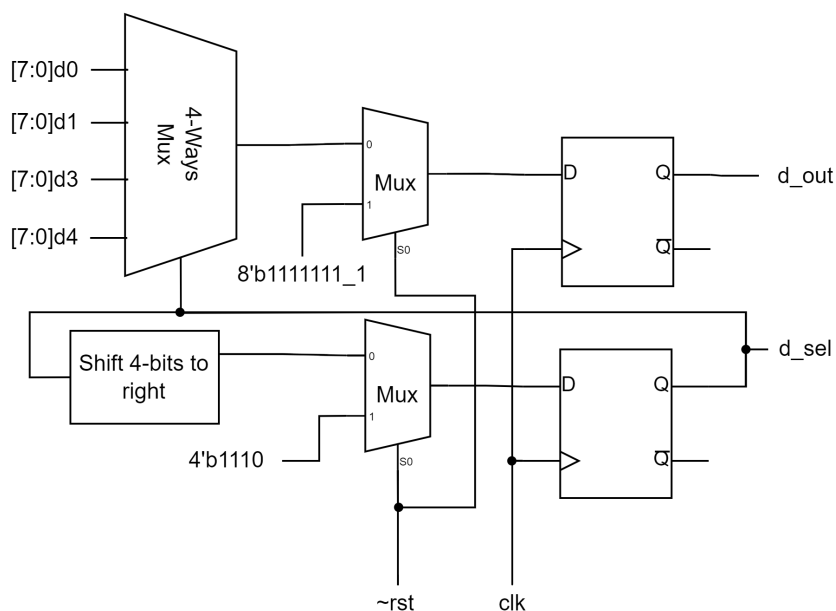
1  `define DIGIT_N 4
2  `define SEGMENT_N 8

```

```
3
4 `define H_BITS `SEGMENT_N'b1001000_1
5 `define N_BITS `SEGMENT_N'b1101010_1
6 `define T_BITS `SEGMENT_N'b1110000_1
7 `define U_BITS `SEGMENT_N'b1000001_1
8
9 module lab3_5(
10     d_sel,
11     d_out,
12     clk,
13     rst
14 );
15
16     output [`DIGIT_N-1:0]d_sel;
17     output [`SEGMENT_N-1:0]d_out;
18     input  clk;
19     input  rst;
20
21     wire [`SEGMENT_N-1:0]Q0;
22     wire [`SEGMENT_N-1:0]Q1;
23     wire [`SEGMENT_N-1:0]Q2;
24     wire [`SEGMENT_N-1:0]Q3;
25
26     shift_register U0(.clk(clk), .rst(rst), .q0(Q0), .q1(Q1), .q2(Q2),
27         .q3(Q3));
28     display_7seg U1(.clk(clk), .rst(rst), .d0(Q0), .d1(Q1), .d2(Q2), .
29         d3(Q3), .d_sel(d_sel), .d_out(d_out));
30 endmodule
```

### Block Diagram

## Lab 3 - 5: 1 Hz 10 Digit Marquee on 7-Segment Display



**Figure 9:** Lab 3-5 Logic Diagram

## I/O Pin Assignment

I/O	clk	rst	d_out[0]	d_out[1]	d_out[2]	d_out[3]	d_out[4]
LOC	W5	V17	V7	U7	V5	U5	V8
d_out[5]	d_out[6]	d_out[7]	d_sel[0]	d_sel[1]	d_sel[2]	d_sel[3]	
U8	W6	W7	U2	U4	V4	W4	