Logic Design Lab 4

Experiment 1

Construct a 4-bit synchronous binary down counter (b3b2b1b0) with the 1-Hz clock frequency from lab3 and use 4 LEDs for displaying the binary values.

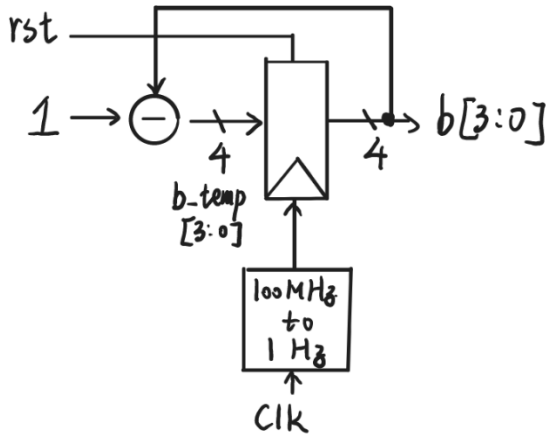
I/O	$f_{crystal}$	b3	b2	b1	b0
Site	W5	V19	U19	E19	U16

Design Specification：

Input: clk,rst.

Output: b [3:0].

Block diagram:



Design Implementation：

Verilog code:

```
23  module lab4_1(
24  b, // output
25  clk, // global clock
26  rst // active high reset
27  );
28  output reg [3:0]b;
29  input clk;
30  input rst;
31  wire clk_new;
32  reg [3:0]b_tmp;
33
34  clk_1HZ U0(.clk_out(clk_new),.clk(clk),.rst_n(rst));
```

```
35
36  // Combinational logics
37  always @*
38      b_tmp = b - 1'b1;
39
40  // Sequential logics: Flip flops
41  always @(posedge clk_new or negedge rst)
42      if (~rst)
43       b<=4'b1111;
44
45      else b<=b_tmp;
46
47  endmodule
```

Discussion：

1. This experiment is based on the previous lab (lab 3). That is to say, we only need to change the “plus 1” from the lab3 to “minus 1” as well as the initial value in the counter (optional) so that we can get the result we want.
2. We skip the discussion of the frequency from 100MHz to 1Hz because it would be well discussed in the lab3.

Conclusion：

This experiment is pretty easy. We only need to copy the code from the lab3 and change some parameters so that we can implement the circuit we want.

.

Experiment 2

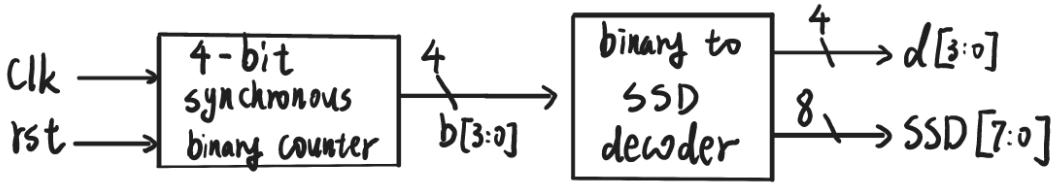
Combine the 4-bit synchronous binary down counter from exp1 with a binary-to-seven-segment-display decoder (from lab2) to display the binary counting in 7-segment display.

Design Specification：

Input: clk,rst.

Output: d [7:0], SSD [7:0] .

Block diagram:



Design Implementation：

1. Verilog code:

Source code

```
23  module lab4_2(clock,rest,d,SSD);
24  input clock;
25  input rest;
26  output [3:0]d;
27  output SSD;
28
29  reg clk;
30  reg rst;
31
32  wire [3:0]b;
33  wire [7:0] SSD;
34  wire [3:0] d;
35
36
37  down_counter U1(.b(b),.clk(clock),.rst(rest));
38  show_SSD U2(.i(b),.d(d),.SSD(SSD));
39
40
41  endmodule
```

```
23  module down_counter(
24  b, // output
25  clk, // global clock
26  rst // active high reset
27  );
28  output reg [3:0]b;
29  input clk;
30  input rst;
31  wire clk_new;
32  reg [3:0]b_tmp;
33
34  lab3_2 U0(.clk_out(clk_new),.clk(clk),.rst_n(rst));
35
36  // Combinational logics
37  always @*
38      b_tmp = b - 1'b1;
39
40  // Sequential logics: Flip flops
41  always @(posedge clk_new or negedge rst)
42      if (~rst)
43       b<=4'b1111;
44
45      else b<=b_tmp;
46
47  endmodule
```

```
23  module lab3_2(
24  clk_out,
25  clk,
26  rst_n
27      );
28
29  output clk_out;
30  input clk;
31  input rst_n;
32
33  reg [25:0]cnt_temp;
34  reg [25:0]cnt;
35  reg clk_t;
36  wire rst_d;
37  wire trigger;
38  wire clk_out;
39
40  // Dividied frequency
41  assign clk_out =clk_t;
42  assign trigger = (cnt== 26'd50000000);
43  assign rst_d =(trigger||rst_n);
44
45  // T-FF
46  always@(posedge trigger or negedge rst_n)begin
47  if (~rst_n)
48  clk_t<=0;
49  else
50  clk_t =~clk_t;
51  end
52
53  // Combinational logics: increment, neglecting overflow
54  always @(cnt)begin
55  cnt_temp = cnt + 1'b1;
56  end
57
58  // Sequential logics: Flip flops
59  always @(posedge clk or negedge rst_d)begin
60  if (~rst_d)
61  cnt<=26'd0;
62  else
63  cnt<=cnt_temp;
64  end
65
66  endmodule
```

Discussion：

1. This lab is the combination of the lab4_1 and the binary to SSD decoder. Thus we only need to use the module made in lab4_1 and combine it with the decoder and then we can get the circuit.

2. The decoder I used is the conventional logic design way to find the Boolean function of each segment. However, there are another way to implement the circuit with higher language and it will be used in the latter experiments.

Conclusion：

Because the experiment is quite simple. I think it is a good chance to recall the implementation of SSD decoder, which can be seen as the base of the following experiment.

Experiment 3

Construct a single digit BCD down counter with a 2-Hz clock as the clock frequency and display on the seven-segment display.

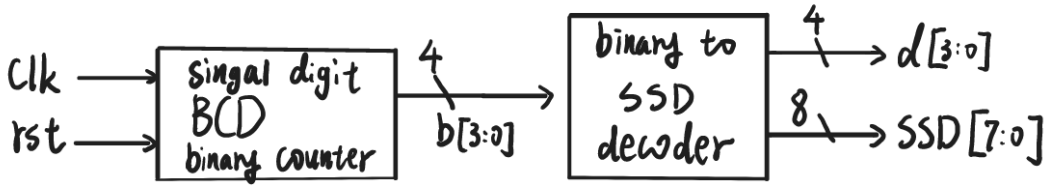
I/O	$f_{crystal}$	f_{out}	rst
Site	W5	U16	V17

Design Specification：

Input: clk,rst.

Output: d [7:0], SSD [7:0].

Block diagram:



Design Implementation：

1. BCD counter

```
23  module bcd_counter(
24  b, // output
25  clk, // global clock
26  rst // active high reset
27  );
28  output reg [3:0]b;
29  input clk;
30  input rst;
31  wire clk_new;
32  reg [3:0]b_tmp;
33
34  clk_2HZ(.clk_out(clk_new),.clk(clk),.rst_n(rst));
35
```

```
36  // Combinational logics
37  always @*
38      if(b!=4'b0000)  b_tmp = b - 1'b1;
39      else   b_tmp=4'b1001;
40
41  // Sequential logics: Flip flops
42  always @(posedge clk_new or negedge rst)
43       if (~rst) b<=4'b1001;
44       else   b<=b_tmp;
45
46  endmodule
```

2. 2HZ clock.

```
module clk_2HZ(clk_out,clk,rst_n);

output clk_out;
input clk;
input rst_n;

reg [25:0]cnt_temp;
reg [25:0]cnt;
reg clk_t;
wire rst_d;
wire trigger;
wire clk_out;

// Dividied frequency
assign clk_out =clk_t;
assign trigger = (cnt== 26'd25000000);
assign rst_d =(trigger||rst_n);

// T-FF
always@(posedge trigger or negedge rst_n)begin
if (~rst_n)
clk_t<=0;
else
clk_t =~clk_t;
end


// Combinational logics: increment, neglecting overflow
always @(cnt)begin
cnt_temp = cnt + 1'b1;
end

// Sequential logics: Flip flops
always @(posedge clk or negedge rst_d)begin
if (~rst_d)
cnt<=26'd0;
else
cnt<=cnt_temp;
end

endmodule
```

3. SSD decoder

```
module BCD_To_SSD(q,d,SSD);
output [7:0] SSD;
output d;
input [3:0] q;
reg [3:0] d;
integer  idx ;
reg [7:0] SSD;
always@(q)
begin
for(idx=0; idx < 4; idx = idx +1) //assign 4-bit binary
    d[idx] <= q[idx];
case (q)
4'd0 : SSD<=8'b00000011;
4'd1 : SSD<=8'b10011111;
4'd2 : SSD<=8'b00100101;
4'd3 : SSD<=8'b00001101;
4'd4 : SSD<=8'b10011001;
4'd5 : SSD<=8'b01001001;
4'd6 : SSD<=8'b01000001;
4'd7 : SSD<=8'b00011111;
4'd8 : SSD<=8'b00000001;
4'd9 : SSD<=8'b00001001;
default SSD<=8'b00000000;
endcase
end
endmodule
```

Discussion：

1. Compared to the lab4_2, we need consider that we need to set the limit for BCD counter.
2. We are also assigned to implement 2HZ clock, and it can be made by dividing the parameter we used in 1 HZ clock by 2.

Conclusion：

This experiment is also simple. However, it is a little awkward that when I demoed the experiment to TAs, it turned out that the clock I made is 2s but not 2HZ. This remind me that even if the experiment isn't complicated, we are also encouraged to assign our parameters and design our circuit carefully.

Experiment 4

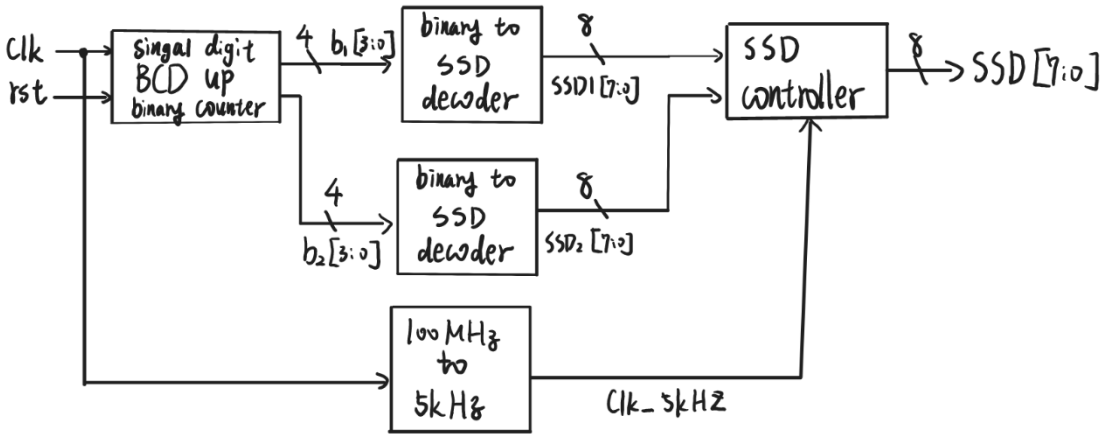
Construct a two-digit BCD up counter with a 1-Hz clock as the clock frequency and display on the seven-segment display.

Design Specification：

Input: clk,rst.

Output: SSD [7:0].

Block diagram:



Design Implementation：

2-digit BCD up counter:

```
module BCD_UP(
b1, // output
b2, // output
clk, // global clock
rst // active high reset
);
output reg [3:0]b1;
output reg [3:0]b2;
input clk;
input rst;
wire clk_new;
reg [3:0]b1_tmp;
reg [3:0]b2_tmp;

clk_1OHZ U0(.clk_out(clk_new),.clk(clk),.rst_n(rst));

// Combinational logics
always @*
    if((b2==4'd9)&&(b1==4'd9))begin
    b1_tmp=4'd0;
    b2_tmp=4'd0;
    end

    else if(b1!=4'd9)begin
     b1_tmp = b1 + 1'b1;
     b2_tmp = b2;
     end
    else if(b1!=4'd9)begin
     b1_tmp = b1 + 1'b1;
     b2_tmp = b2;
     end
    else
    begin
     b1_tmp=4'd0;
     b2_tmp = b2 + 1'b1;
    end

// Sequential logics: Flip flops
always @(posedge clk_new or negedge rst)
     if (~rst)begin
      b1<=4'd0;
      b2<=4'd0;
      end
     else
      begin
      b1<=b1_tmp;
      b2<=b2_tmp;
      end
endmodule
```

Implementation of clocks (skip)
SSD and SSD control (skip)

Discussion：

1. When we program the Verilog code, it is important that when we use higher language, we need to be careful that all the conditions are included in the implement code.

2. After thinking how to implement 2-digit BCD up counter, I think it is easier to implement the circuit by dividing 2-digit and considering each digit as a number instead of considering the 2-digit as a number, which means we separate the 2-digit number to ten digit and unit digit. Because, by use the method, we can use two SSD decoders with SSD control to implement the circuit. However, if we use the method, we need to think throughout and logically to avoid omitting the case we don't consider.

Conclusion：

From the block diagram, it is obvious that this experiment is more complex than the previous experiments. However, I think this experiment is the core experiment of this lab because it helps us integrate the skills of how to implement clock with different frequencies, SSD as well as SSD control and BCD counter.

Reference：

1. The handout and the assignment of logic design
   To review some basic concept of the combination circuit and the problem bulls and cows.
2. The handout of logic design lab
   To program our FPGA board by following the method on the handout of logic design lab.