Logic Design Lab 8

Experiment 1

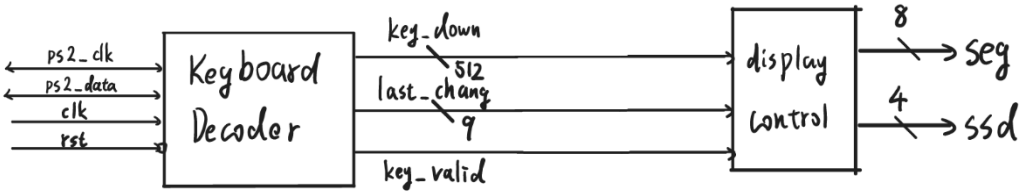
Implement keyboard using the left-hand-side keyboard (inside the black blocks).

Design Specification：

Input: clk, rst_n.

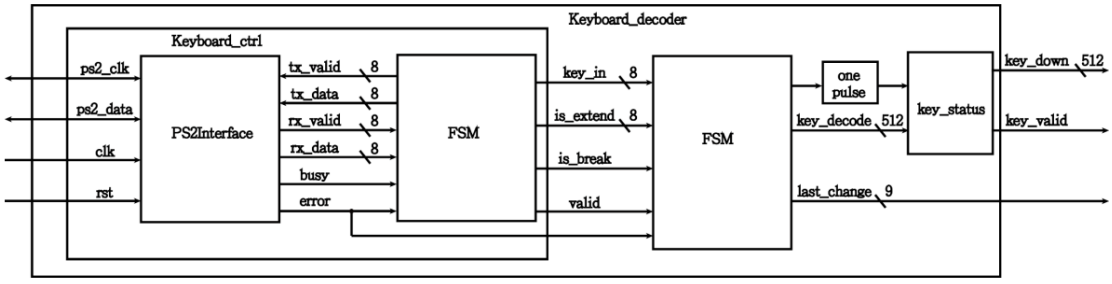
Output: arduino_mclk, arduino_lrlk, arduino_sck, arduino_sdin.

Block diagram:



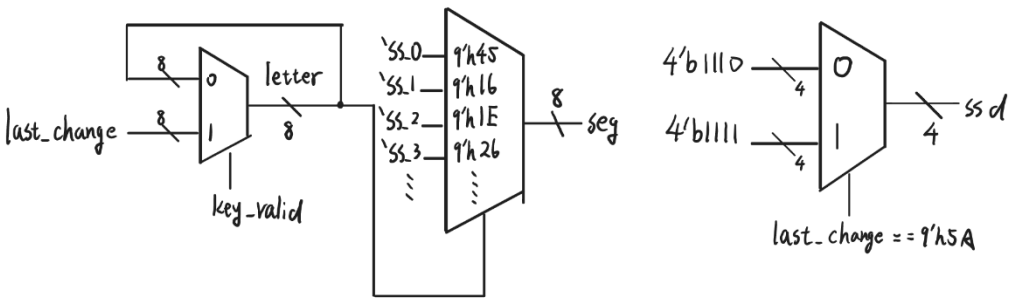
Design Implementation：

1. Keyboard decoder：



(Copied form the handout)

2. Display control：



Discussion：

1. Keyboard decoder：
   The keyboard decoder blocks (IP) is offered by lecturer. We only need to know what parameters in the blocks stands for.

2. Display control：
   (1) From the block diagram, we can use key valid and last change to determine which button be pressed, and then we can show the pattern corresponding to the button we press.
   (2) Besides, when we press “enter”, which means that the last change is enter and we can clear our SSD by set our ssd to 4’d1111(all closed).

Conclusion：

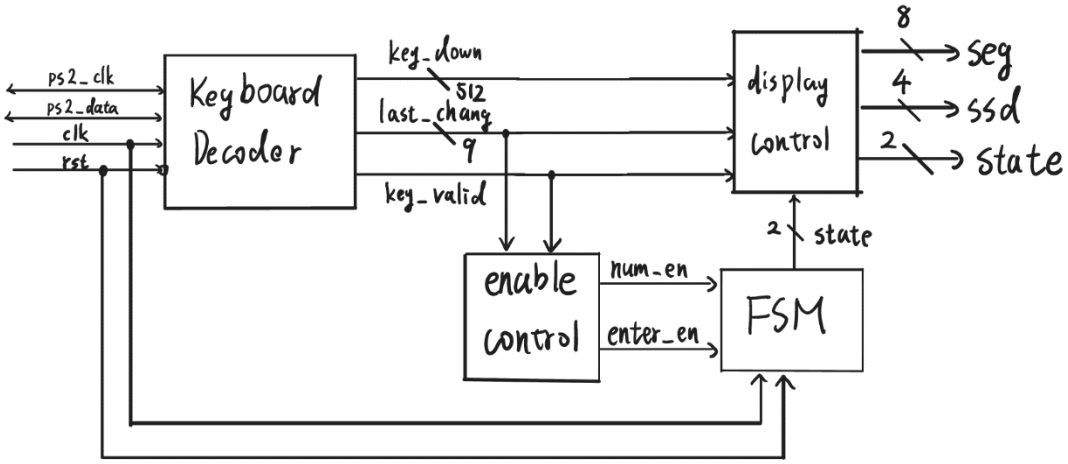
This experiment is the fundamental model of this lab. However, although it seems to be simple, it also took me some time to get used to what keyboard transmits data.

Experiment 2

Implement a single digit decimal adder using the keyboard as the input and display the results on the 7-segment display (The first two digits are the addend/augend, and the last two digits are the sum).

Design Specification：

Input: ps2_clk, ps2_data, clk, rst.
Output: [7:0] seg, [3:0] ssd, [1:0] state.
Block diagram:



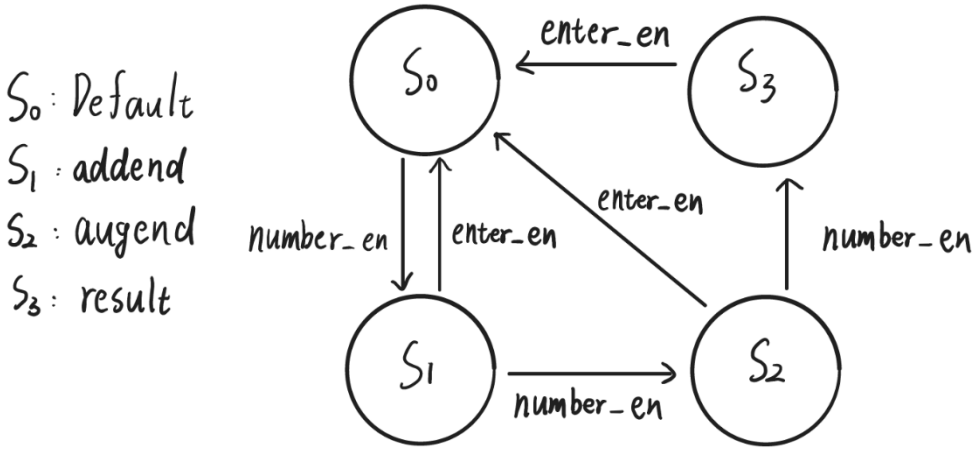
Design Implementation：

1. Enable control：

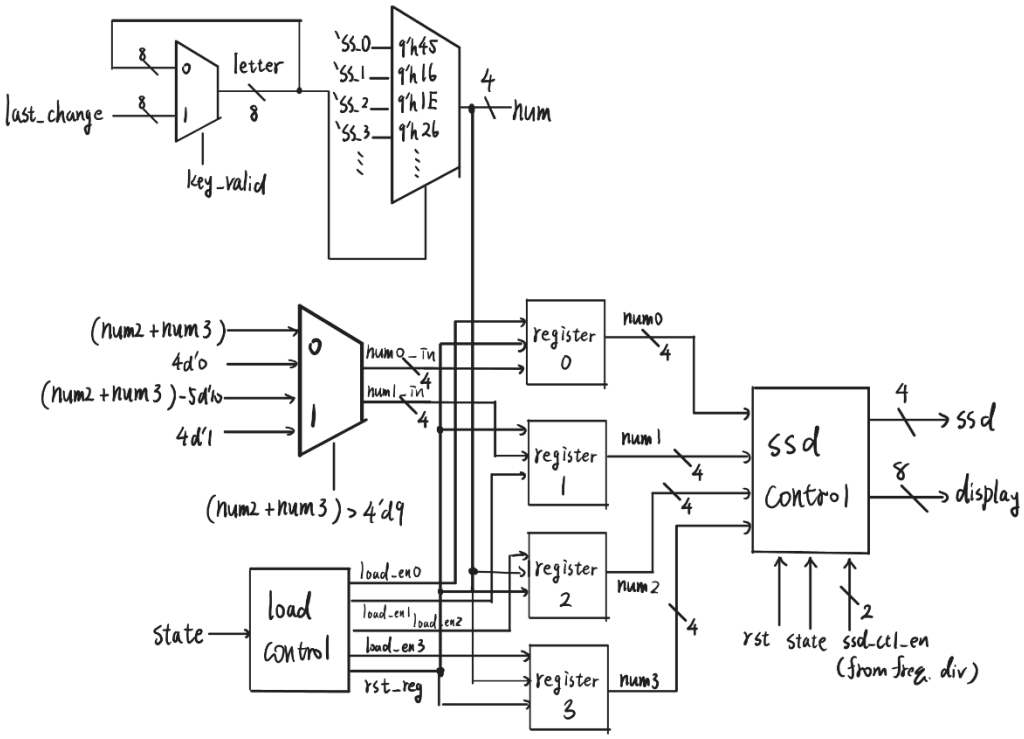
```
assign num_en= key_valid & ((last_change==9'h70)|(last_change==9'h69)|(last_change==9'h72)|(last_change==9'h7A)|(last_change==9'h6B)
| (last_change==9'h73)|(last_change==9'h36)|(last_change==9'h6c)|(last_change==9'h75)|(last_change==9'h7D));

assign enter_en= key_valid & (last_change==9'h5A);
```

2. FSM：



3. Display control：



Discussion：

1. Enable control：a simple combination circuit to control FSM.
2. Display control：
   (1) Register：simple combination circuit to hold value when load enable if off and read the value from the MUX when the load enable is on

```
module register(reg_out,reg_in,load_en,rst);
output reg [5:0]reg_out;
input [5:0]reg_in;
input load_en,rst;

always@*begin
if(~rst)
reg_out = 5'd0;
else if(load_en)
reg_out = reg_in;
else
reg_out = reg_out;
end
endmodule
```

   (2) Load enable：determining whether the load enable is on or not according to the present state.

   (3) SSD control：Like load enable and the ssd control in previous lab, the only difference between the previous ones is that the ssd_ctl will close some pattern on ssd depending on the state.

Conclusion：

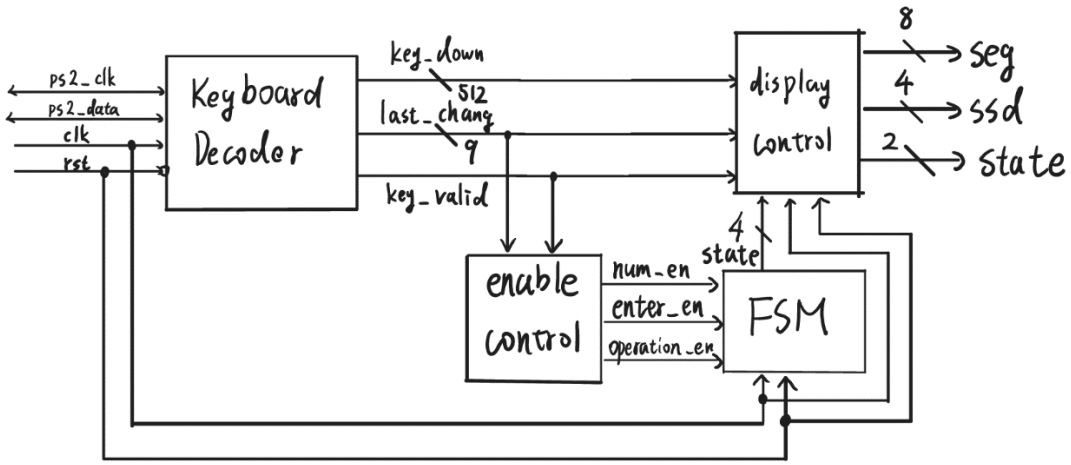
In this experiment, we start to use FSM to help us to solve the problem. Except for the traditional FSM we have practiced from time to time, we also need to deal with the issue that the carry in the decimal adder and the control over different positions on SSD which is similar to the electronic clock in lab6. Besides, the next lab is based on this lab. Thus, if we can get more familiar with this experiment, it may help us to do the experiment3 with more ease.

Experiment 3

Implement a two-digit decimal adder/subtractor/multiplier using the right-hand-side keyboard (inside the red block). You don't need to show all inputs and outputs at the same time in the 7-segment display. You just need to show inputs when they are pressed and show the results after "Enter" is pressed.

Design Specification：

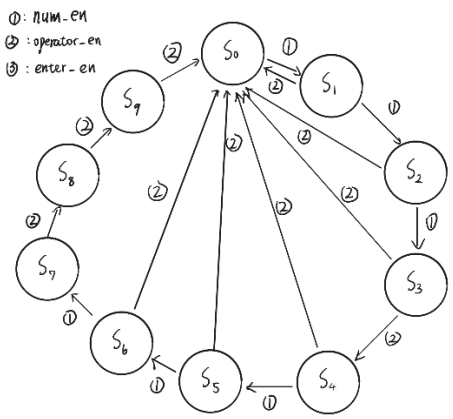
Input: ps2_clk, ps2_data, clk, rst.
Output: [7:0] seg, [3:0] ssd, [3:0] state.
Block diagram:



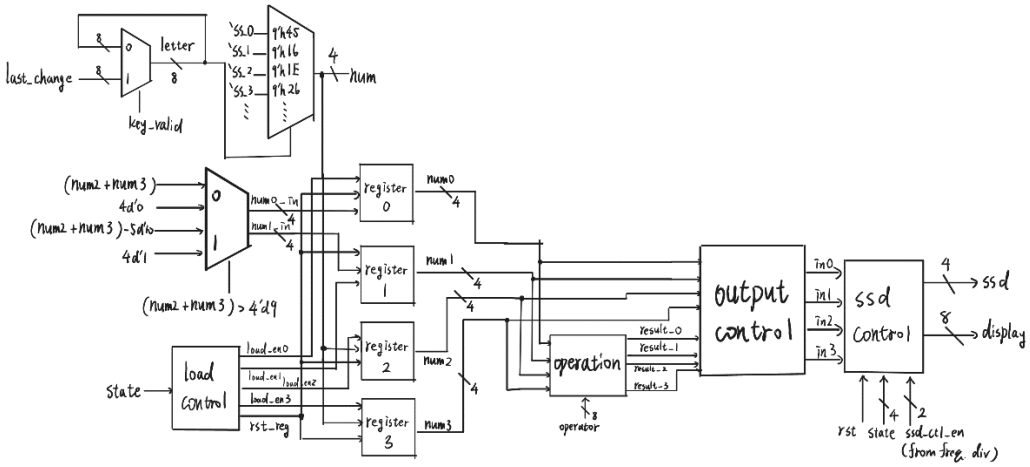
Design Implementation：

1. Enable control：
   Similar to lab8_2. Skip.

2. FSM：



3. Display control：



Discussion：

1. Display control：

(1) Operation：Because this experience requires us to do different operations, thus we need a combination circuit to decide which operation we use and the result of the calculation.

```
always@*begin
 if(operator==`SS_a) begin

        result_0= (((10*num1)+num0)+((10*num3)+num2))-(100*result_2)-(10*result_1);
        result_1= (((10*num1)+num0)+((10*num3)+num2)-(100*result_2))/10;
        result_2= (((10*num1)+num0)+((10*num3)+num2))/100;
        result_3= 4'd0;

  end
  else if(operator==`SS_s) begin
     if(((10*num1)+num0)<((10*num3)+num2))begin
        result_0= ((10*num3)+num2)-((10*num1)+num0)-(10*result_1);
        result_1= (((10*num3)+num2)-((10*num1)+num0))/10;
        result_2= 4'd0;
        result_3= 4'd11;
     end
```

(2) Output control：After we calculate the result, we use this block to decide outputs.

```
    else if(state == `S1)begin
        in0=0;
        in1=0;
        in2=0;
        in3=0;
    end
    else if(state == `S2)begin
        in0=seg0;
        in1=seg1;
        in2=0;
        in3=0;
    end
    else if(state == `S3)begin
        in0=seg0;
        in1=seg1;
        in2=0;
        in3=0;
```

(3) SSD control：Similar to lab8_2.

```
else if (state==`S2) begin
    case(ssd_ctl_en)
    2'b00: display_c = 4'b1111;
    2'b01: display_c = 4'b1101;
    2'b10: display_c = 4'b1111;
    2'b11: display_c = 4'b1111;
    default : display_c = 4'b1111;
    endcase
end

else if (state==`S3) begin
    case(ssd_ctl_en)
    2'b00: display_c = 4'b1110;
    2'b01: display_c = 4'b1101;
    2'b10: display_c = 4'b1111;
    2'b11: display_c = 4'b1111;
    default : display_c = 4'b1111;
    endcase
end
```

Conclusion：

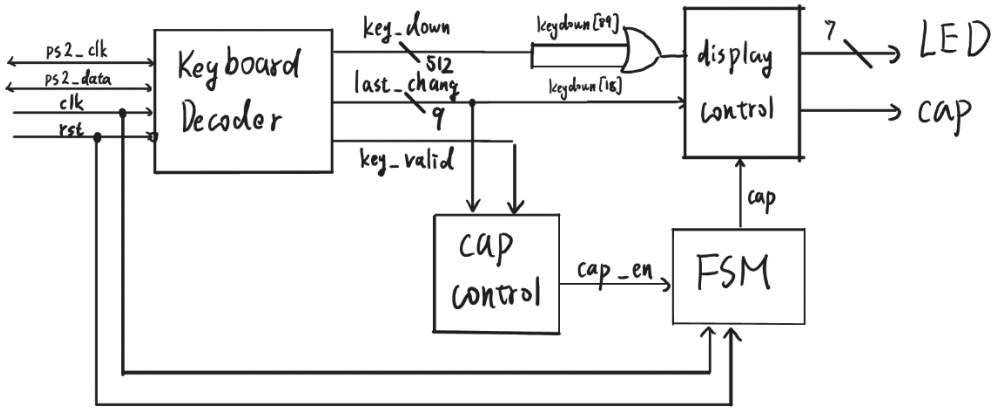
This experiment is similar to lab8_2. However, the reason why I use so many states is that when we press the button, it usually will jump two states. Thus, the approach I use to solve the problem is to add buffer states to solve the problem. Nonetheless, there may exist some method to solve the problem.

Experiment 4

Implement the "Caps" control in the keyboard. When you press A-Z and a-z in the keyboard, the ASCII code of the pressed key (letter) is shown on 7-bit LEDs.

Design Specification：

Input: ps2_clk, ps2_data, clk, rst.
Output: [7:0] seg, [3:0] ssd, [1:0] state.
Block diagram:



Discussion：

Cap decision：

(1) When key down [89] or key down [18] are on, we press the button of shift. Thus, when we press the shift, the output turns to be capital.
(2) And when we press the caps lock on the keyboard, we use FSM to decide the cap state.

Conclusion：

This experiment is based on the lab8_1, and we only change some input/ output condition and the SSD decoder so that we can get the circuit we want.

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.