106061151 劉安得

Lab9-1

1. **Specification**

   Display 0~9 & A,S,M in seven segment display with Keyboard

   Input : clk, rst

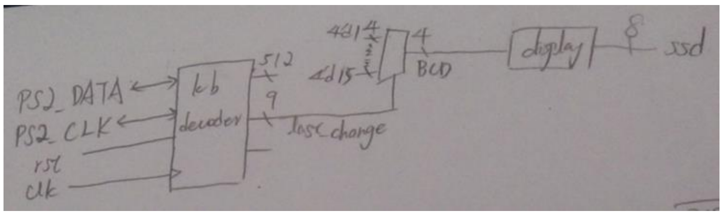
   Output : [7:0] ssd, [3:0] ssd_ctl,

   Inout : PS2_CLK, PS2_DATA

2. **Implementation**

   第一個子實驗比較簡單，我將之前寫的 display 新增了 A, S, M, enter 這些功能，分別用 4-bits binary 的 10, 11, 12, 15 代替

   以下為各 module 間的連結，我在 top module 有寫一個 case 判斷現在的 last change 代表什麼



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

I/O	Ssd_ctl[3]	Ssd_ctl[2]	Ssd_ctl[1]	Ssd_ctl[0]
Pin	W4	V4	U4	U2

I/O	PS2_CLK	PS2_DATA	clk	Rst
Pin	C17	B17	W5	V17

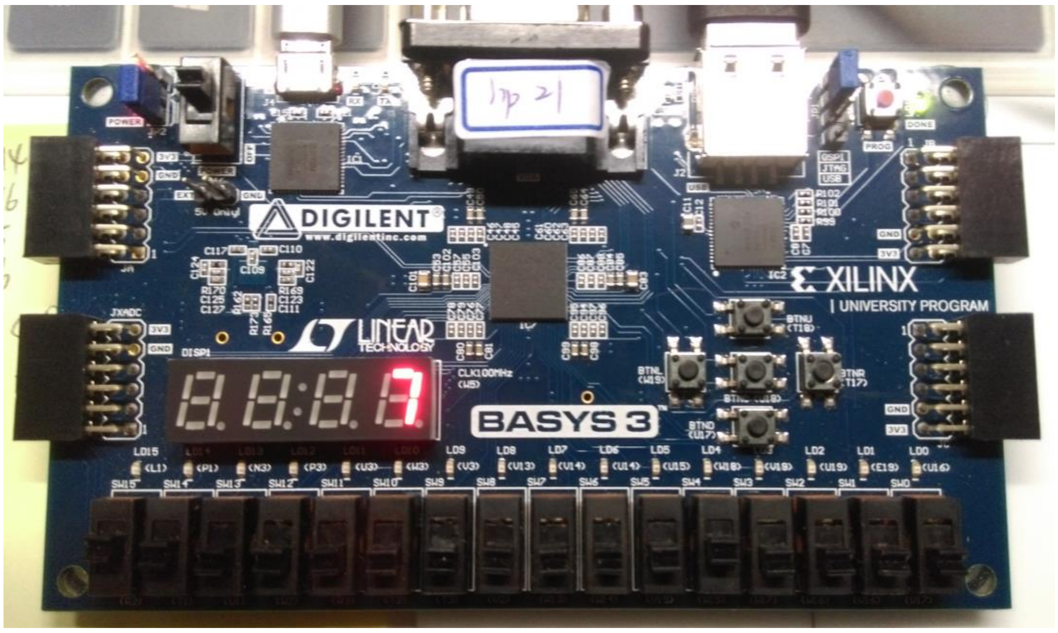
```
case(last_change)
            8'H16 : BCD = 4'd1;
            8'H1E : BCD = 4'd2;
            8'H26 : BCD = 4'd3;
            8'H25 : BCD = 4'd4;
            8'H2E : BCD = 4'd5;
            8'H36 : BCD = 4'd6;
            8'H3D : BCD = 4'd7;
```

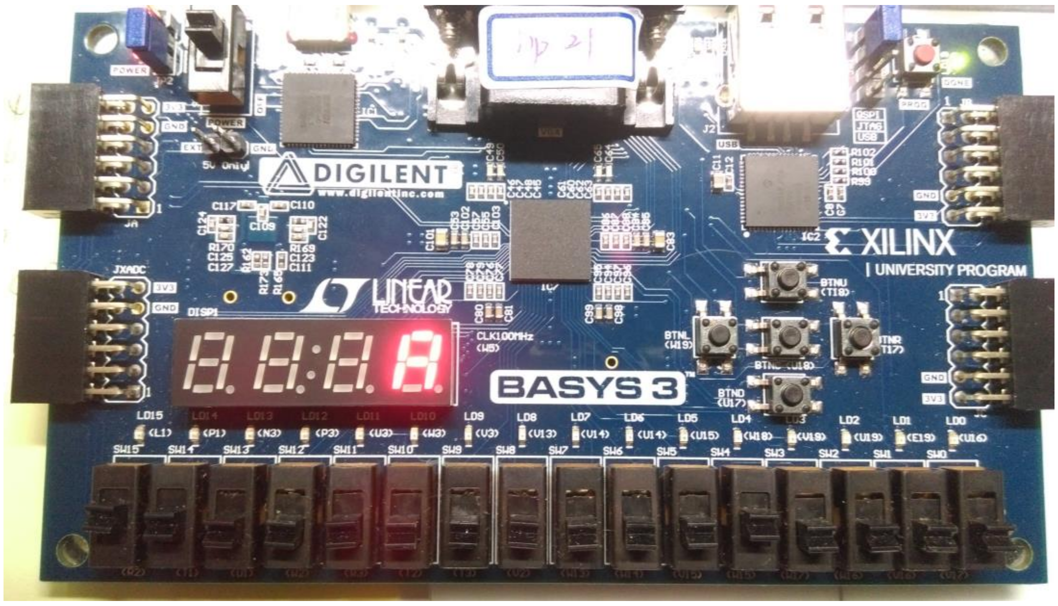
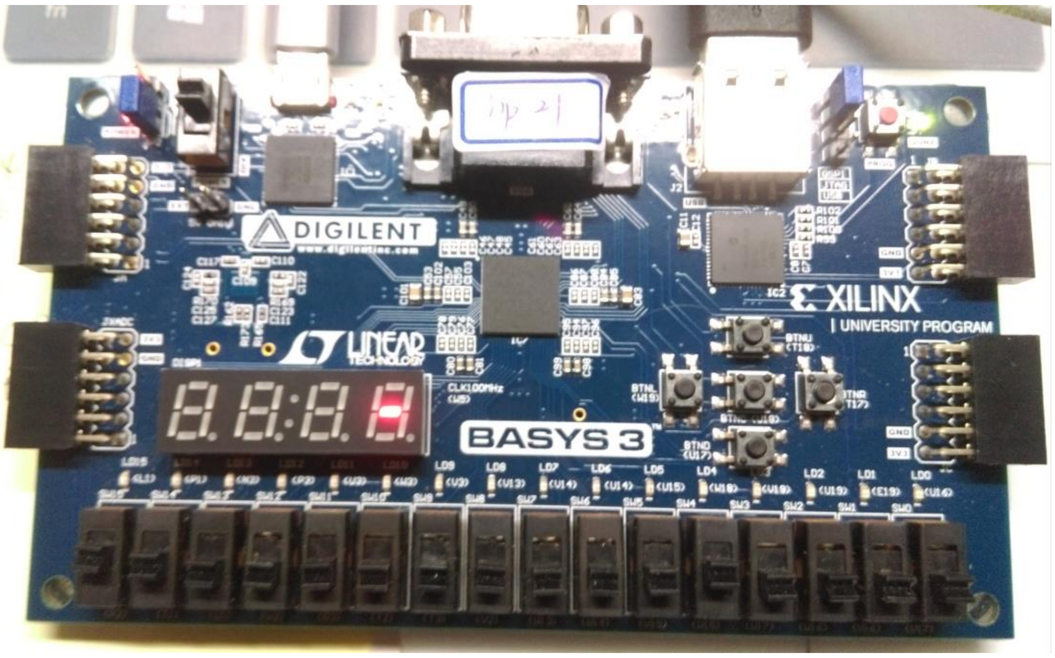
```
        8'H3E : BCD = 4'd8;
        8'H46 : BCD = 4'd9;
        8'H45 : BCD = 4'd0;
        8'H1C : BCD = 4'd10;
        8'H1B : BCD = 4'd11;
        8'H3A : BCD = 4'd12;
        8'H5A : BCD = 4'd15;
    Endcase
```

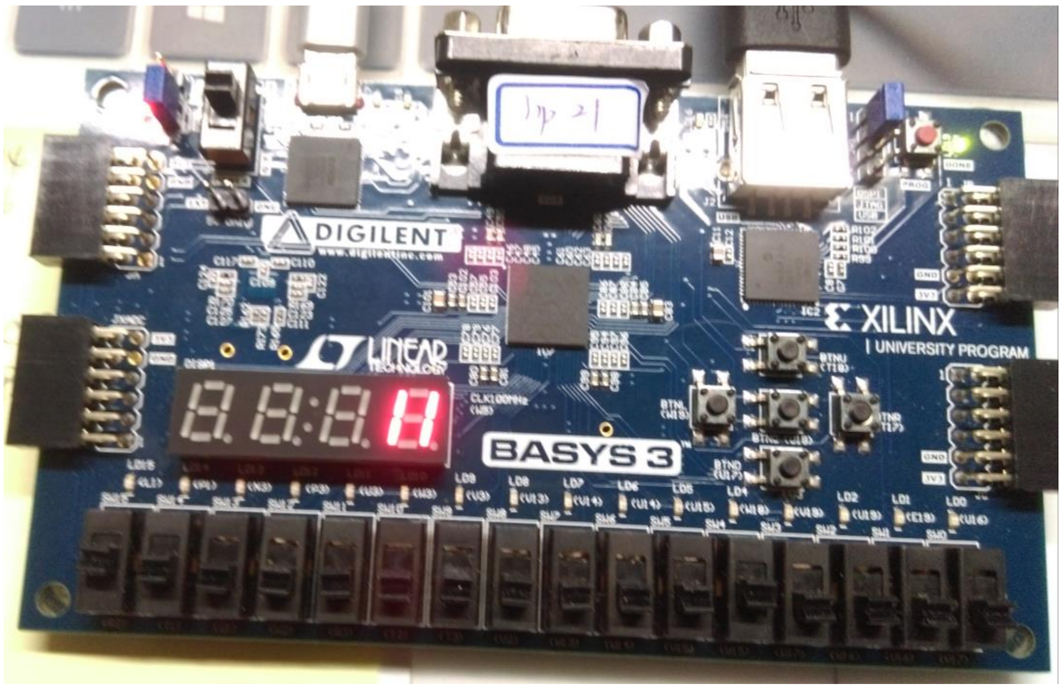
3. Discussion

一開始要理解 keyboard 怎麼運作的需要花比較多時間，理解徹底後就順利許多。其中我把 enter 設成按下後七段顯示器不會亮，而不是像 reset 那樣

4. Result




DIGILENT
LINEAR TECHNOLOGY
XILINX
UNIVERSITY PROGRAM
BASYS 3
CLK100MHz
BTNU
BTNL
BTNR
BTND

Lab9-2

1. **Specification**

   Adder with keyboard，the first and second seven segment display are addend and augend, and the third and forth are the sum

   Input : clk, rst

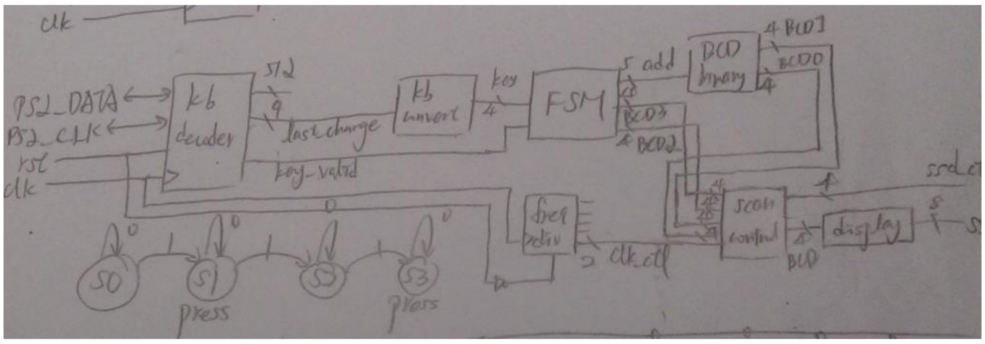
   Output : [7:0] ssd, [3:0] ssd_ctl,

   Inout : PS2_CLK, PS2_DATA

2. **Implementation**

   因為要使用兩次 lab9-1 中的 case 來偵測按的數字，所以我將它寫成一個 module。

   以下為各 module 間的連結，我在 top module 有寫一個 FSM 紀錄現在讀到第幾個數字，並再設一個 wire add 紀錄答案



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

I/O	Ssd_ctl[3]	Ssd_ctl[2]	Ssd_ctl[1]	Ssd_ctl[0]
Pin	W4	V4	U4	U2

I/O	PS2_CLK	PS2_DATA	clk	Rst
Pin	C17	B17	W5	V17

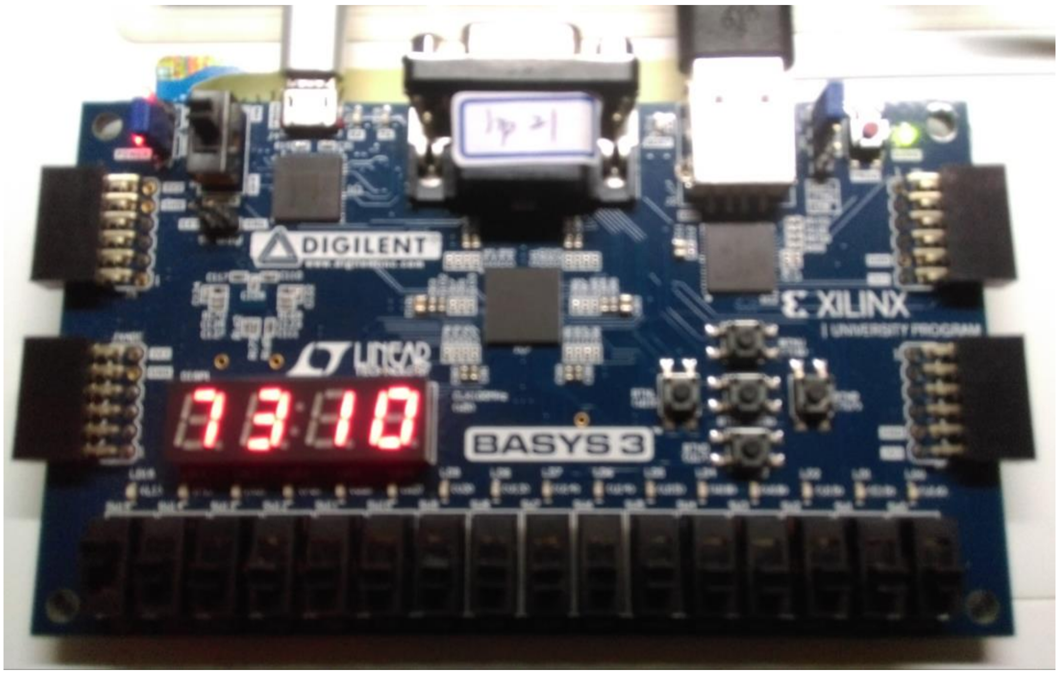
```
assign add = (state == `S1 || state == `S2) ? BCD3 : BCD3 + BCD2;

next_BCD3 = BCD3;
            next_BCD2 = BCD2;
            case(state)
                  `S0 :
                        next_state = (key_valid == 1'b1) ? `S1 : `S0;
```

```
        `S1 : begin
            next_state = (key_valid == 1'b1) ? `S2 : `S1;
            next_BCD3 = key;
            next_BCD2 = 4'd15;
        end
        `S2 :
            next_state = (key_valid == 1'b1) ? `S3 : `S2;
        `S3 : begin
            next_state = (key_valid == 1'b1) ? `S0 : `S3;
            next_BCD2 = key;
        end
    endcase
```

3. Discussion

我想了很久要怎麼偵測到我按下第一個和第二個數字，最後選擇使用 key_valid，偵測到兩次 1 就代表輸入完一個數字，並使用 FSM implement

4. Result



Lab9-3

1. **Specification**

   two-digit decimal adder / subtractor / multiplier with keyboard

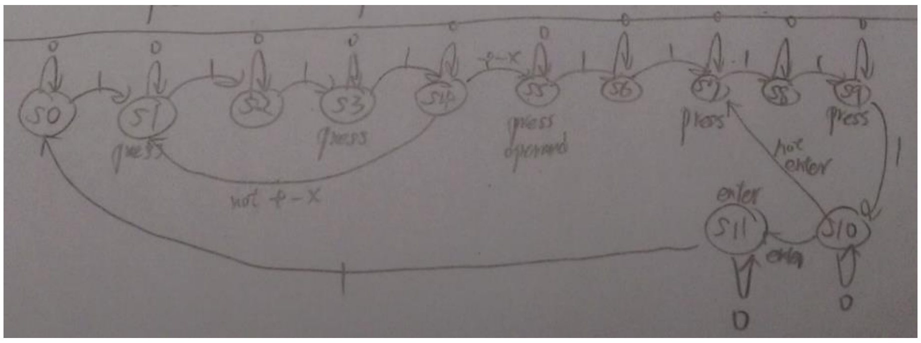
   Input : clk, rst

   Output : [7:0] ssd, [3:0] ssd_ctl, [3:0] LED

   Inout : PS2_CLK, PS2_DATA

2. **Implementation**

   使用的 module 和 9-2 一樣，我在 top module 寫了一個 FSM 紀錄現在讀到哪裡，並設一個 reg ans 紀錄答案，最後我還設了 LED 燈，顯示你在輸入第一個值，運算子，第二個值，還是顯示答案



以下為各 module 間的連結

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

I/O	Ssd_ctl[3]	Ssd_ctl[2]	Ssd_ctl[1]	Ssd_ctl[0]
Pin	W4	V4	U4	U2

I/O	LED[3]	LED[2]	LED[1]	LED[0]
Pin	V19	U19	E19	U16

I/O	PS2_CLK	PS2_DATA	clk	Rst
Pin	C17	B17	W5	V17

```
assign num1 = (state == `S3) ? 7'd10*BCD1 + key : num1;
    assign operand = (state == `S5) ? key : operand;
    assign num2 = (state == `S9) ? 7'd10*BCD1 + key : num2;
```

```
	assign LED[3] = ( state == `S1 || state == `S2 || state == `S3 || state == 
`S4) ? 1'b1 : 1'b0;
	assign LED[2] = (state == `S5 || state == `S6) ? 1'b1 : 1'b0;
	assign LED[1] = (state == `S7 || state == `S8 || state == `S9 || state == 
`S10) ? 1'b1 : 1'b0;
	assign LED[0] = (state == `S11 || state == `S0) ? 1'b1 : 1'b0;

	always @*
	begin
		next_BCD3 = BCD3;
		next_BCD2 = BCD2;
		next_BCD1 = BCD1;
		next_BCD0 = BCD0;

		case(state)
			`S0 : begin
				next_state = (key_valid == 1'b1) ? `S1 : `S0;
			end
			`S1 : begin//press
				next_state = (key_valid == 1'b1) ? `S2 : `S1;
				next_BCD3 = 4'd15;
				next_BCD2 = 4'd15;
				next_BCD1 = 4'd15;
				next_BCD0 = key;
			end
			`S2 : begin
				next_state = (key_valid == 1'b1) ? `S3 : `S2;
				if(key_valid == 1'b1)
					next_BCD1 = BCD0;
			end
			`S3 : begin//press
				next_state = (key_valid == 1'b1) ? `S4 : `S3;
				next_BCD0 = key;
			end
			`S4 : begin
				if(key_valid == 1'b0)
					next_state = `S4;
				else if(key == 4'd10 || key == 4'd11 || key == 4'd12)
```

```
                    next_state = `S5;
                else
                    next_state = `S1;
            end
            `S5 : begin//press operand
                next_state = (key_valid == 1'b1) ? `S6 : `S5;
                next_BCD1 = 4'd15;
                next_BCD0 = key;
            end
            `S6 :
                next_state = (key_valid == 1'b1) ? `S7 : `S6;
            `S7 : begin//press
                next_state = (key_valid == 1'b1) ? `S8 : `S7;
                next_BCD1 = 4'd15;
                next_BCD0 = key;
            end
            `S8 :begin
                next_state = (key_valid == 1'b1) ? `S9 : `S8;
                if(key_valid == 1'b1)
                    next_BCD1 = BCD0;
            end
            `S9 : begin//press
                next_state = (key_valid == 1'b1) ? `S10 : `S9;
                next_BCD0 = key;
            end
            `S10 : begin
                if(key_valid == 1'b0)
                    next_state = `S10;
                else if(last_change == 9'H5A)
                    next_state = `S11;
                else
                    next_state = `S7;
            end
            `S11 : begin//enter
                next_state = (key_valid == 1'b1) ? `S0 : `S11;
                next_BCD3 = (operand == 4'd11 && num1 < num2)? 4'd11 :
ans/14'd1000;
                next_BCD2 = (ans%14'd1000) / 14'd100;
```

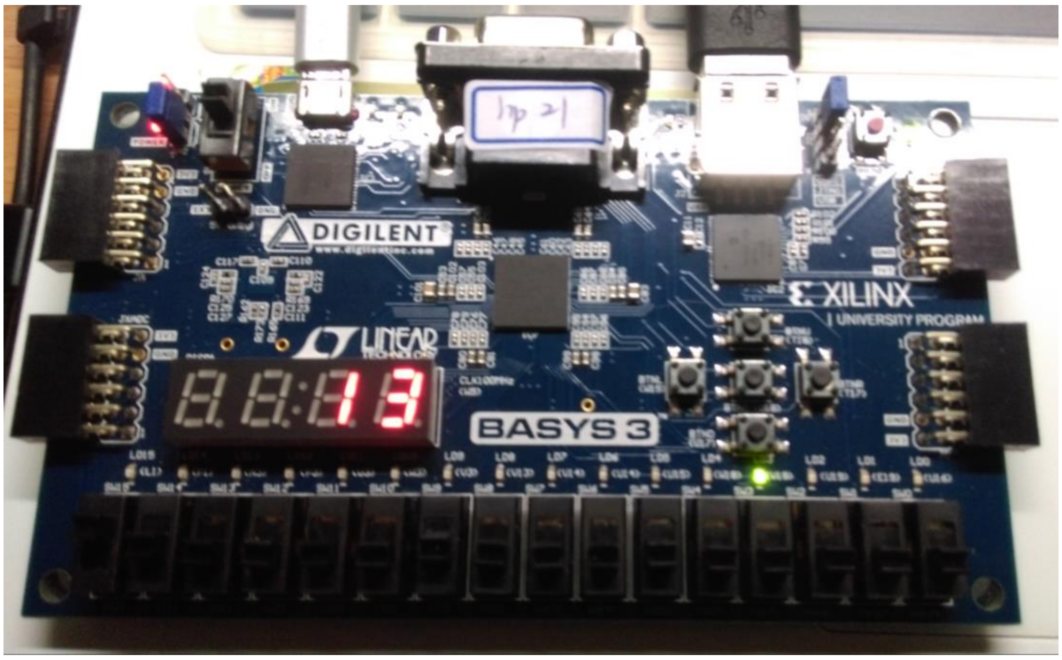
```
                    next_BCD1 = (ans%14'd100) / 14'd10;
                    next_BCD0 = ans%14'd10;
                end
            endcase

            case(operand)
                4'd10 : ans = num1 + num2;
                4'd11 : ans = (num1 >= num2) ? num1 - num2 : num2 - num1;
                4'd12 : ans = num1 * num2;
          Endcase
```

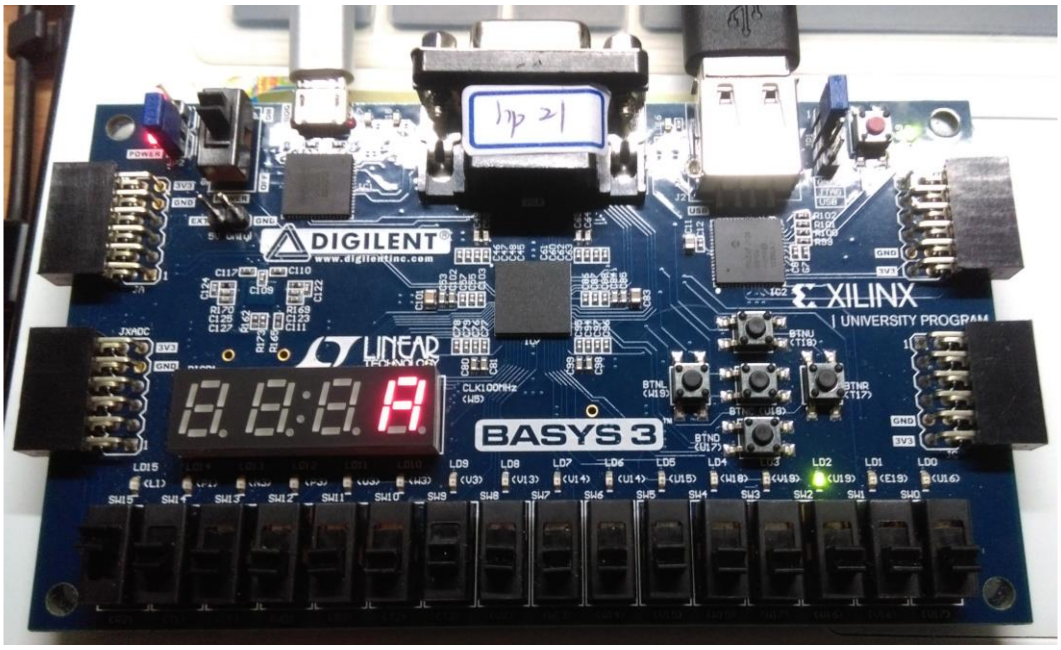
3. **Discussion**

  這次 lab 的 FSM 我寫了 11 個 state 為了記錄我現在讀到哪裡，我覺得或許有更好的做法，畢竟 11 個 state 有點多，但我一時想不到。

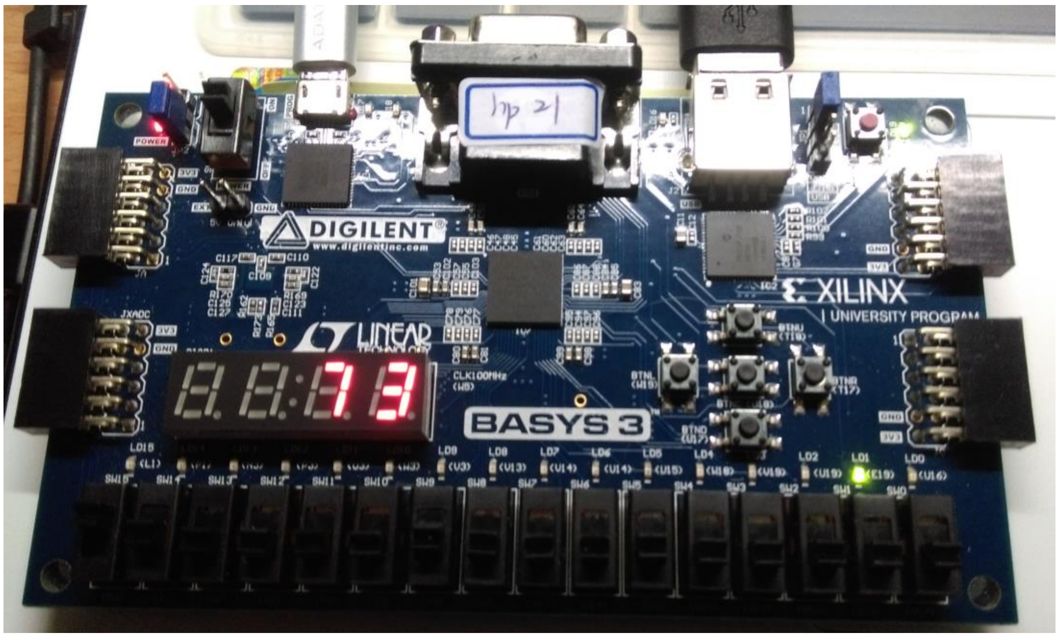
  再來是設想各個 state 要做什麼事的時候，很容易亂掉，一亂掉就會讓顯示錯誤，不過透過顯示找到哪裡出錯，多改幾次就會成功

4. **result**

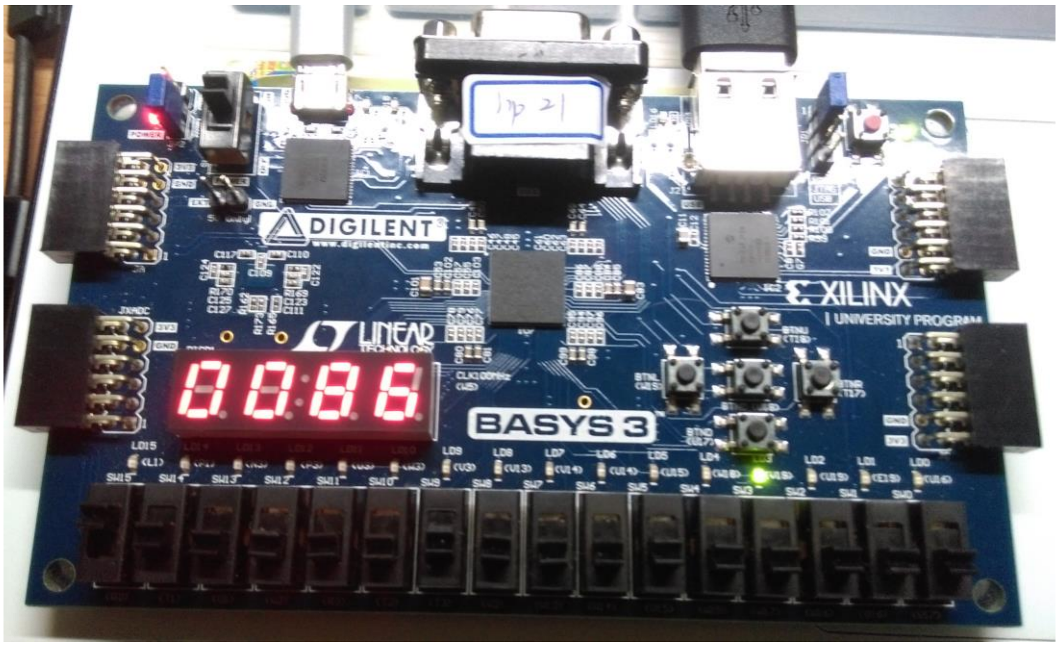


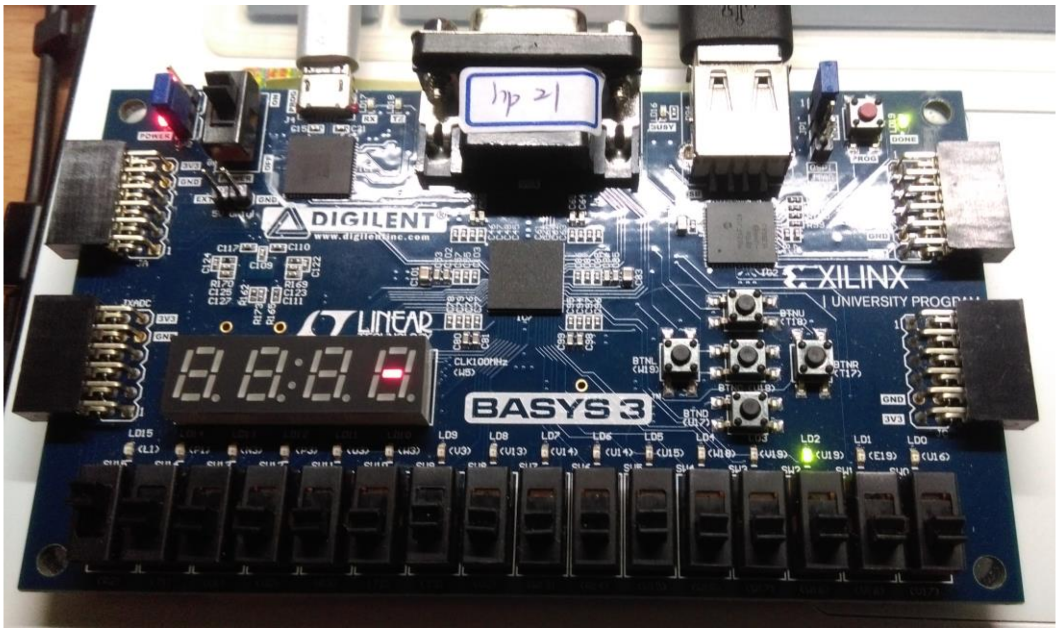


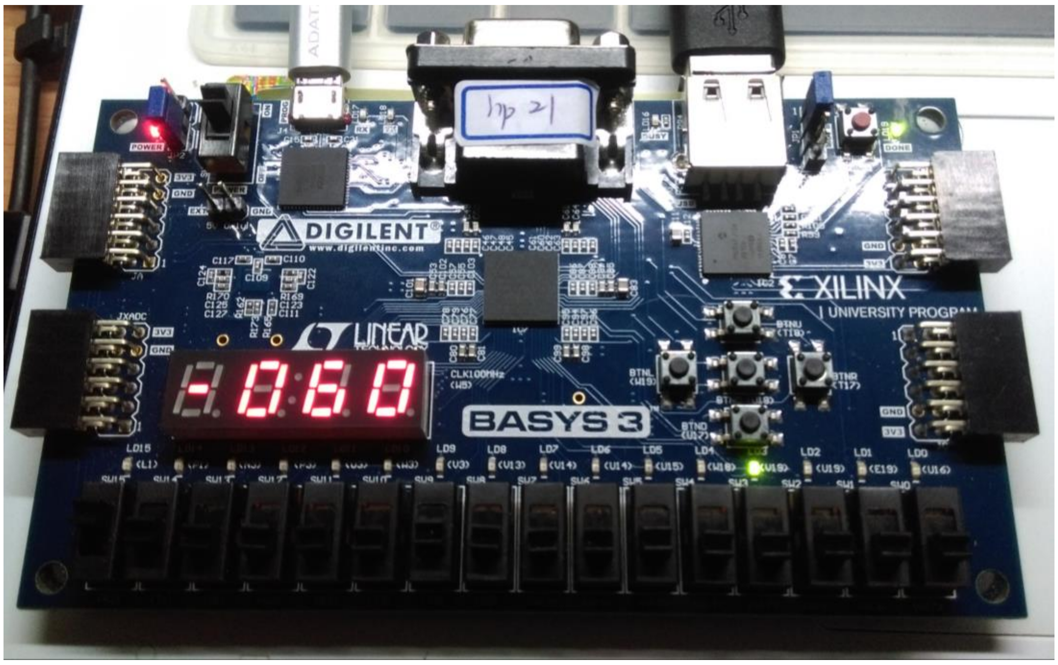
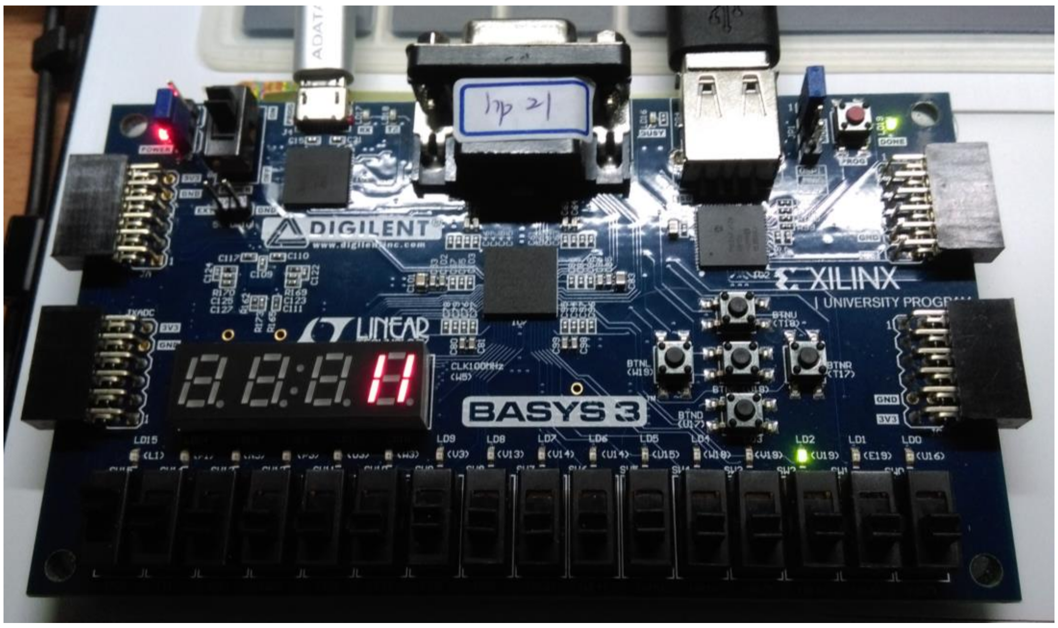
DIGILENT
LINEAR TECHNOLOGY
XILINX
UNIVERSITY PROGRAM
BASYS 3
A

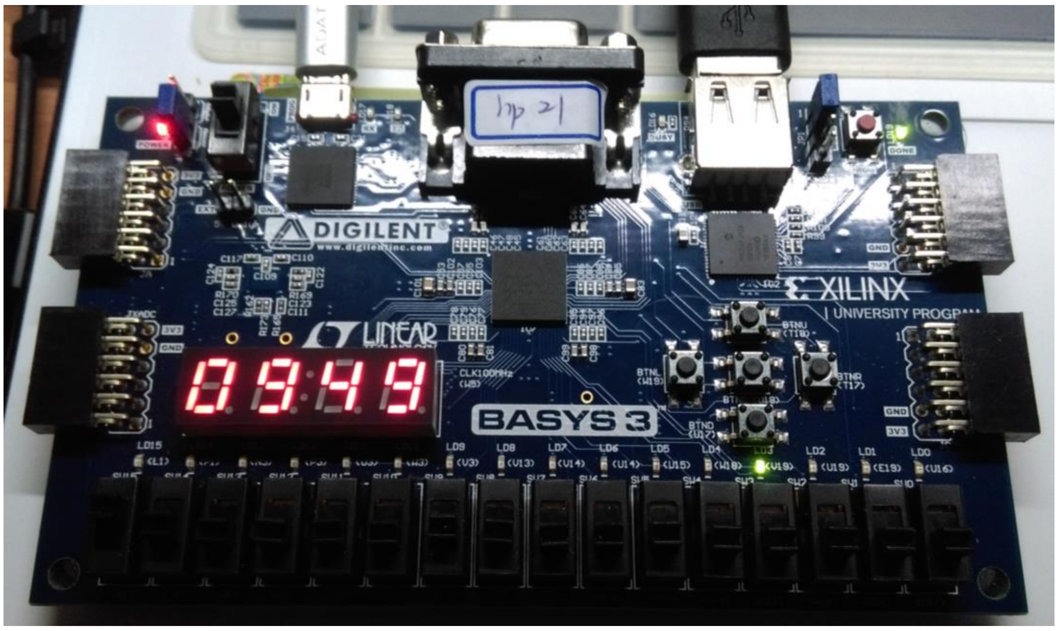


DIGILENT
LINEAR TECHNOLOGY
XILINX
UNIVERSITY PROGRAM
BASYS 3
73


DIGILENT
www.digilentinc.com
LINEAR TECHNOLOGY
0086
BASYS 3
XILINX
UNIVERSITY PROGRAM


DIGILENT
www.digilentinc.com
LINEAR
BASYS 3
XILINX
UNIVERSITY PROGRAM


lnp 21
DIGILENT
XILINX
UNIVERSITY PROGRAM
BASYS 3
-060

lnp 21
DIGILENT
XILINX
UNIVERSITY PROGRAM
BASYS 3
H


hp 21
DIGILENT
LINEAR
0949
BASYS 3
XILINX
UNIVERSITY PROGRAM

Lab9-4

1. **specifation**

   display ASCII code with caps and shift

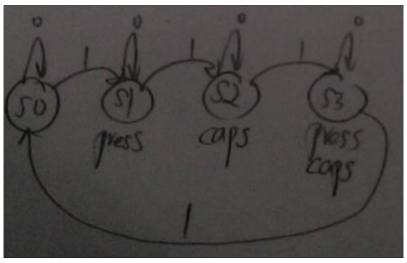
   Input : clk, rst

   Output : [7:0] ssd, [3:0] ssd_ctl, caps_lock, [6:0] ASCII

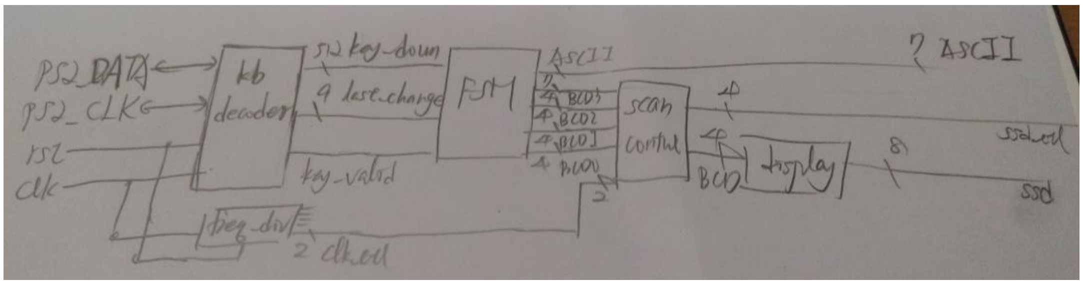
   Inout : PS2_CLK, PS2_DATA

2. **implementation**

   我在 top module 寫了一個 FSM 紀錄 caps 的狀態，並當 caps 和 shift 異號時為大寫，也就是 caps = 1 且 shift 沒被按下，或者 caps=0，shift 有按下。再用 case 判斷 last change 為多少。最後我還設了 seven segment display，顯示 ASCII code，這樣比較好 debug



以下為各 module 間的連結



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

I/O	Ssd_ctl[3]	Ssd_ctl[2]	Ssd_ctl[1]	Ssd_ctl[0]
Pin	W4	V4	U4	U2

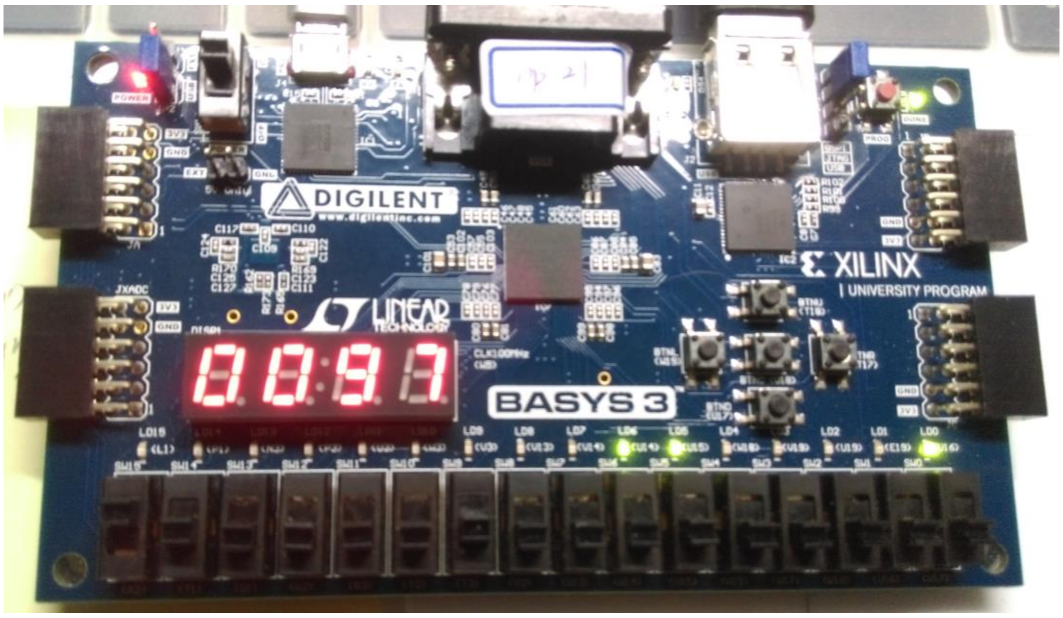
I/O	ASCII[6]	ASCII[5]	ASCII[4]	ASCII[3]	ASCII[2]	ASCII[1]	ASCII[0]
Pin	U14	U15	W18	V19	U19	E19	U16

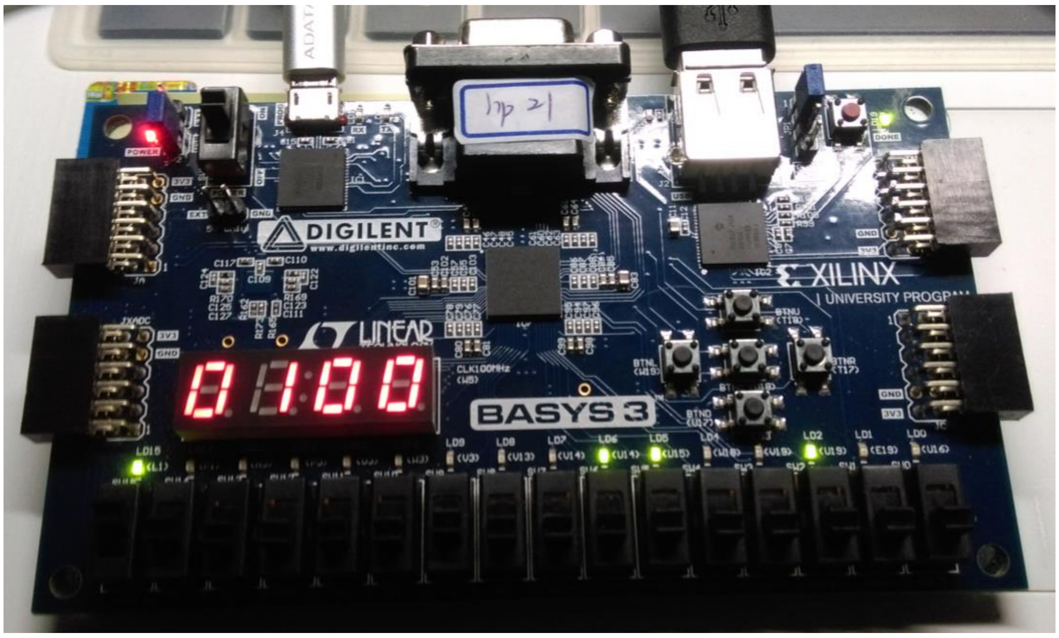
I/O	Caps_lock	PS2_CLK	PS2_DATA	clk	Rst
Pin	L1	C17	B17	W5	V17

3. discussion

雖然 ASCII code 看起來複雜，但只要處理好 caps 和 shift，其實和前面的 lab 差不多。為了 debug 方便我直接顯示 ASCII code，我覺得這樣不管是助教或是學生都比較方便。

4. result





Conclusion :

做完這次實驗，讓我學會如何使用 keyboard，keyboard 因為有特定的傳輸協定，因此處理上比較麻煩。聽說之後還要做電子琴，結合上次的 speaker 和這次的 keyboard

Reference :

09_keyboard

P. 4 the Pin of keyboard

P. 9 the scan code of keyboard

p. 11 IO of keyboard