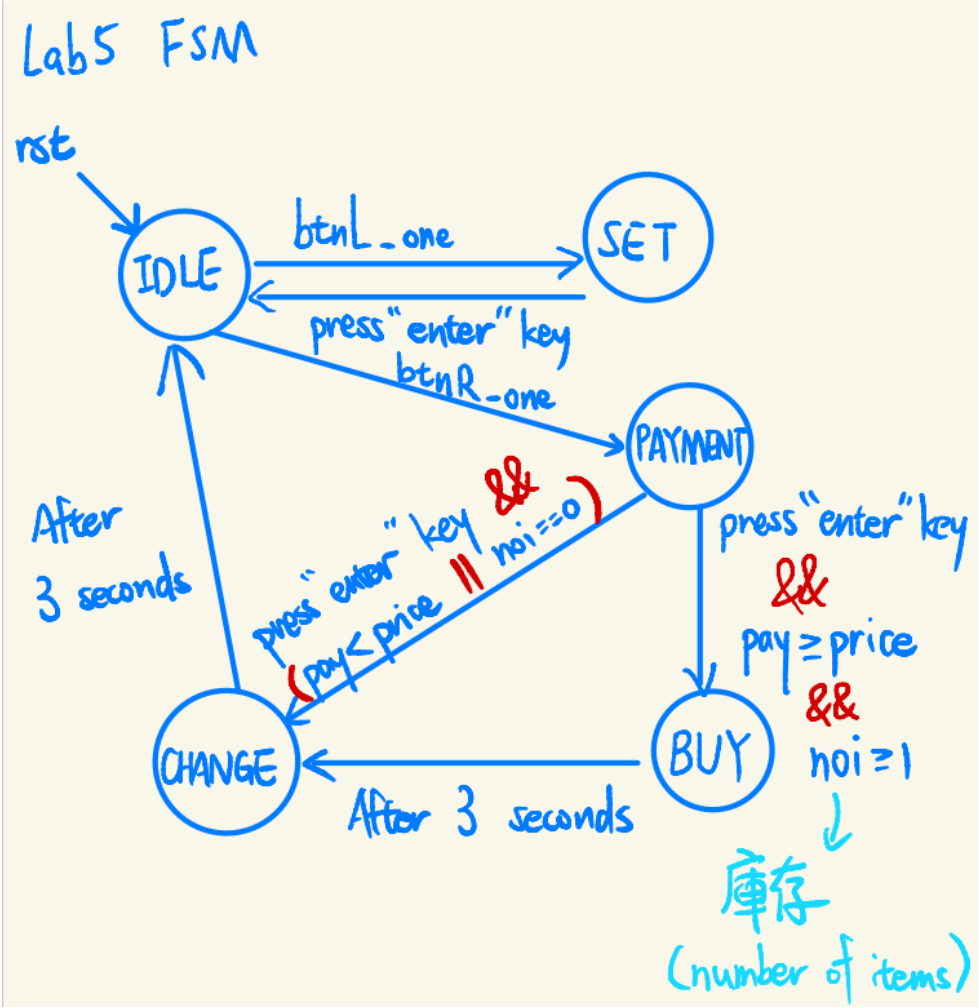
Lab 5

學號: 109021115 姓名: 吳嘉濬

A. Lab Implementation

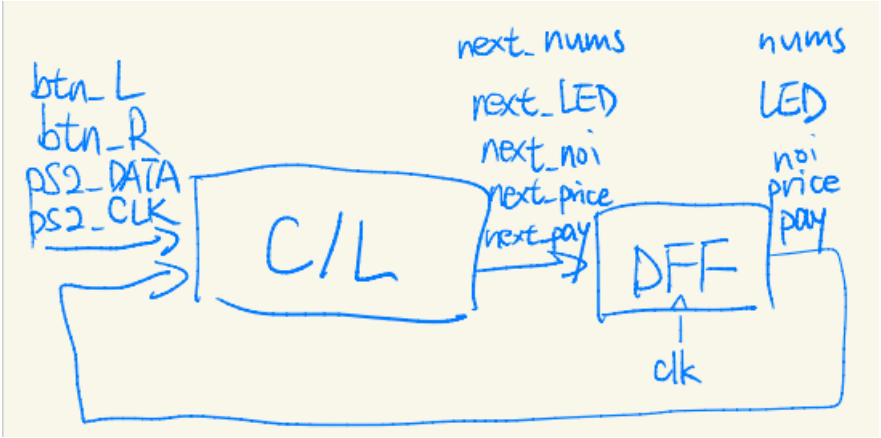
以下為 Lab5 的 FSM：



初始狀態是在 IDLE state。當在 IDLE state 按下 btnL 時，訊號經過 debounce 和 one-pulse 的處理之後，成為 btnL_one 訊號，被 trigger 剛好一個 clock cycle，進入 SET state。在 SET state 中可以設定 noi(庫存)和 price(單價)，按下 enter 鍵回到 IDLE state。如果在 IDLE state 按下 btnR，產生一個 clock cycle 的 btnR_one 訊號，進入 PAYMENT state。在 PAYMENT state，我們可以設定要付的金額，其中按不同數字鍵代表著不同金額的增加或歸零，相關 code 如下：

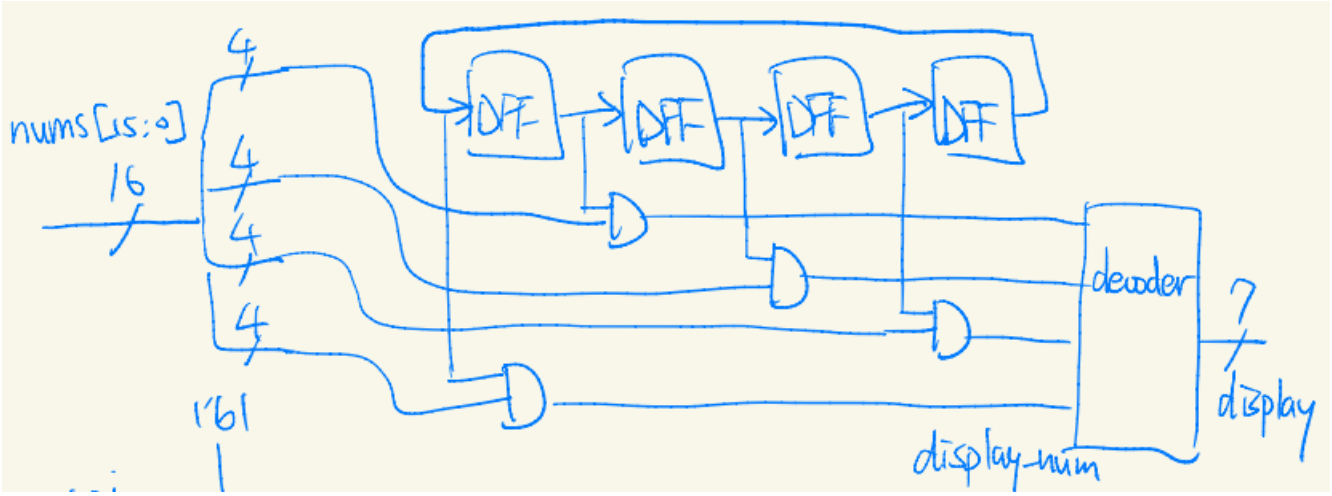
```
if(key_valid && key_down[last_change] == 1'b1 && key_num!=4'b1111 && key_down-key_decode==0) begin  //press number key
    if(key_num==4'b0000) begin  //turn to 0
        next_pay=8'b0;
    end else if(key_num==4'b0001) begin  //+1
        if(pay[7:4]==4'd9 && pay[3:0]==4'd9) begin
            next_pay=pay;  //99
        end else if(pay[3:0]==4'd9) begin
            next_pay={pay[7:4]+1,4'd0};
        end else begin
            next_pay={pay[7:4]+1-1,pay[3:0]+4'd1};  //the most strange bug I ever seen
        end
    end else if(key_num==4'b0010) begin  //+5
        if(pay[7:4]==4'd9 && pay[3:0]>=4'd5) begin
            next_pay=8'b10011001;  //{4'd9,4'd9}
        end else if(pay[3:0]>=4'd5) begin
            next_pay={pay[7:4]+1,pay[3:0]-4'd5};
        end else begin
            next_pay={pay[7:4],pay[3:0]+4'd5};
        end
    end else if(key_num==4'b0011) begin  //+10
        if(pay[7:4]==4'd9) begin
            next_pay=8'b10011001;
        end else begin
            next_pay={pay[7:4]+1,pay[3:0]};
        end
    end else if(key_num==4'b0100) begin  //+50
        if(pay[7:4]>=4'd5) begin
            next_pay=8'b10011001;
        end else begin
            next_pay={pay[7:4]+4'd5,pay[3:0]};
        end
    end else begin  //pressing key other than 0,1,2,3,4
        next_pay=pay;
    end
end
```

可以得知當我按數字 0,1,2,3,4 時，對應歸零、+1、+5、+10、+50，並把上限限制在 99。
按下 enter 鍵，會開始判斷付的金額是否夠買至少一個商品且庫存是否至少有一個，如果是的話進入 BUY state，反之直接跳到 CHANGE state。進入 BUY state 之後 7-seg 會顯示購買的數量以及所需花費的金額，led 燈會每 0.5 秒變換一次，以”亮暗亮暗亮暗”的形式在 3 秒過後進入 CHANGE state。進入 CHANGE state 之後 7-seg 會顯示購買的數量以及買家所剩的餘額，led 燈全亮 3 秒過後回到 IDLE state。
以下為 Lab5 的 block diagram：



這個是最簡化的 block diagram，簡單說明了這個 FSM 的運作原理。鍵盤的輸入透過較為底層的 module KeyboardCtrl_0.v 和 Ps2Interface.v，再經過 KeyboardDecoder，最後再進入上層的 Lab5.v module，把 ps2_DATA 和 ps2_CLK 訊號做適當處理，讓我們可以接收來自鍵盤的 input 資料，並做出我們想要的行為。

以下針對細節，個別畫出對應的 block diagram：

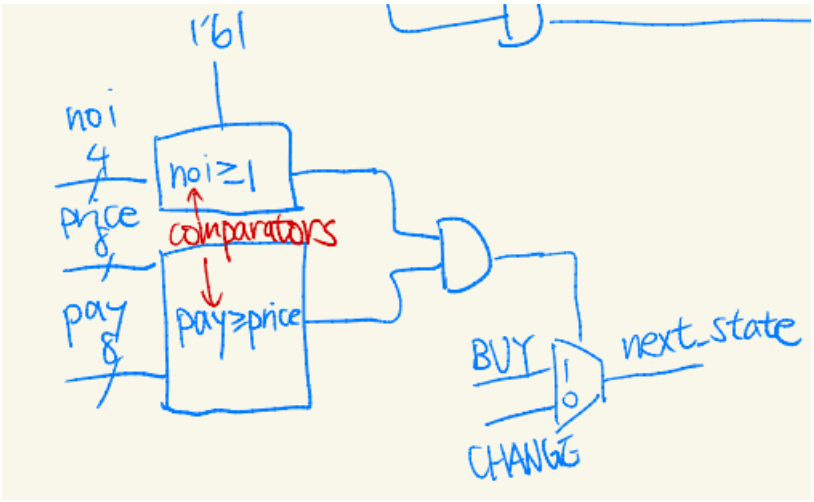


這是 7-seg display 的處理過程。我們把想要顯示的數字資料從左到右 4 個 digit，依次存到 num[15:12], num[11:8], num[7:4], num[3:0]當中，經過 DFFs 的處理我們可以每經過一個 clock cycle 切換一次顯示的 digit，之後我們用寫好的 table 經過 decoder 把它轉換成能在 7-seg 顯示出對應數字樣貌的數值，即 display。以下是對應的 code：

```
always @ (posedge clk_divider[15], posedge rst) begin
    if (rst) begin
        display_num <= 4'b0000;
        digit <= 4'b1111;
    end else begin
        case (digit)
            4'b1110 : begin
                    display_num <= nums[7:4];
                    digit <= 4'b1101;
                end
            4'b1101 : begin
                    display_num <= nums[11:8];
                    digit <= 4'b1011;
                end
            4'b1011 : begin
                    display_num <= nums[15:12];
                    digit <= 4'b0111;
                end
            4'b0111 : begin
                    display_num <= nums[3:0];
                    digit <= 4'b1110;
                end
            default : begin
                    display_num <= nums[3:0];
                    digit <= 4'b1110;
                end
        endcase
    end
end
```

```
always @ (*) begin
    case (display_num)
        0 : display = 7'b1000000;   //0000
        1 : display = 7'b1111001;   //0001
        2 : display = 7'b0100100;   //0010
        3 : display = 7'b0110000;   //0011
        4 : display = 7'b0011001;   //0100
        5 : display = 7'b0010010;   //0101
        6 : display = 7'b0000010;   //0110
        7 : display = 7'b1111000;   //0111
        8 : display = 7'b0000000;   //1000
        9 : display = 7'b0010000;   //1001
        10: display = 7'b0111111;   //1010  "-"
        default : display = 7'b1111111;
    endcase
end
```

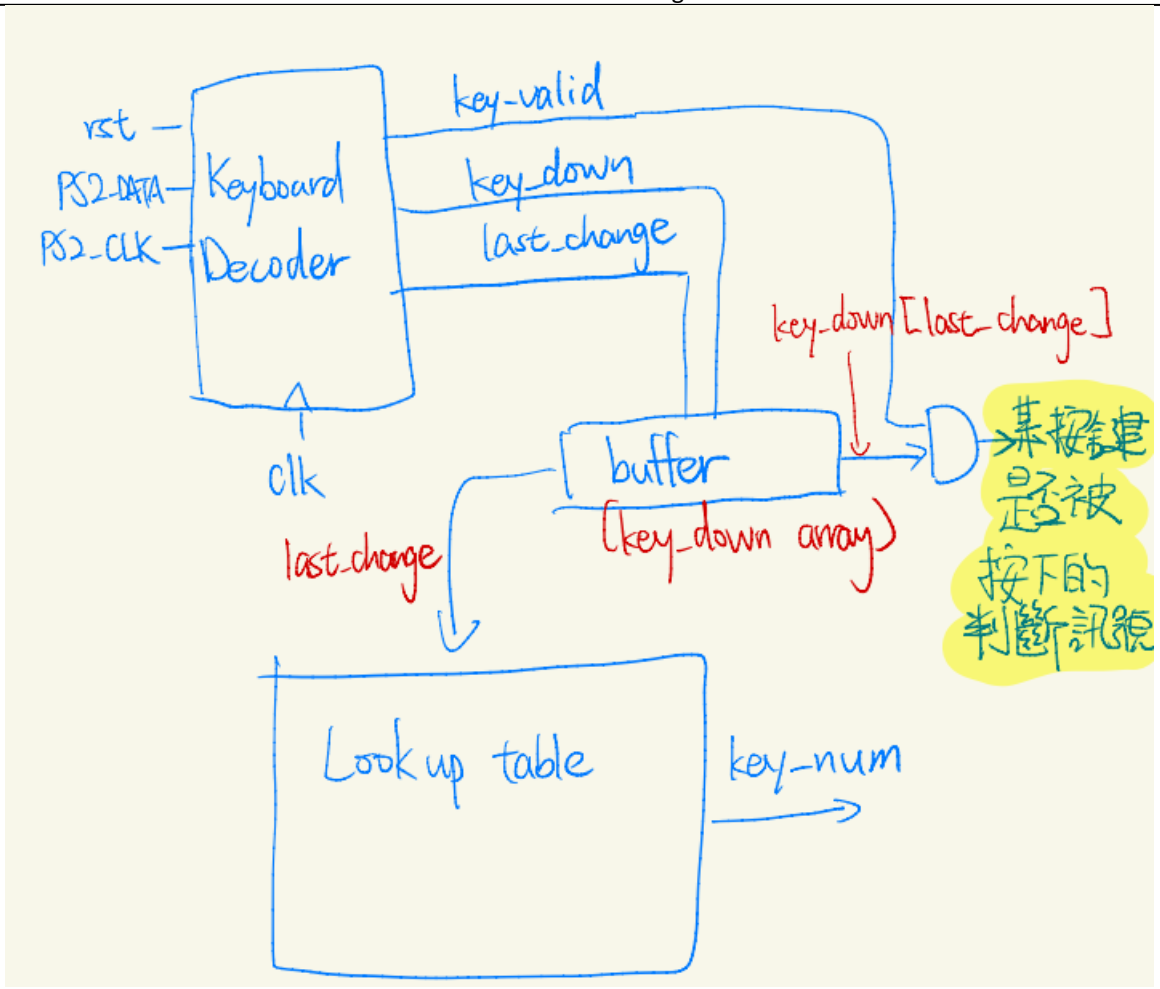
以下是在 PAYMENT state 按下 enter 鍵之後判斷要進入哪一個 state 對應的電路：



利用 2 個 comparator 去比較庫存是否>=1 還有付的錢是否>=商品單價(即買得起>=1 件商品)，之後再把它 and 起來，如果此 signal==1'b1，next_state=BUY；如果此 signal==1'b0，next_state=CHANGE。對應 code 如下：

```
if(pay10>=price10 && noi>=4'd1) begin  //buy at least one item
    next_state=BUY;
    can_bought=pay10/price10;
    bought = (noi>=can_bought) ? can_bought : noi ;
    paid10=price10*bought;
    back10=pay10-paid10;
    paid1=paid10/10;
    paid0=paid10%10;
    back1=back10/10;
    back0=back10%10;
    next_nums={bought,4'b1010,paid1,paid0};
    next_noi=noi-bought;
    next_LED=16'b1111111111111111;
    next_cnt=32'd0;
end else begin
    next_state=CHANGE;
    next_nums={4'b0,4'b1010,pay};
    next_LED=16'b1111111111111111;
    next_cnt=32'd0;
end
```

補充一下這幾行 code 的意思。pay10、price10、back10 分別是指 pay、price、back 在 10 進位下的數值，can_bought 是指可以購買的數量，bought 是指實際購買的數量，這會受限於庫存是否充足，這會由第 4 行的 code 進行相關處理。



以上是"判斷某按鍵是否被按下"以及"如何得知按了哪個數字鍵"的相關電路。經過 KeyboardDecoder module，我們可以得到 key_valid, key_down, last_change 的訊號，尋找在 key_down array 中 index 為 last_change 的那一個訊號看是否為高電位且當下的 key_valid 是否也是高電位，如果都是的話，這意味著最新按下的那個按鍵正在被 trigger，進而採取對應的動作。以下為對應的 code：

```
if(key_valid && key_down[last_change] == 1'b1 && key_num!=4'b1111 && key_down-key_decode==0) begin  //press number key
```

我們也可以利用 last_change 的值在 lookup table 中找到對應的 key_num，得知我們按了哪個數字鍵。以下為對應的 code：

```
parameter [8:0] KEY_CODES [0:19] = {
    9'b0_0100_0101, // 0 => 45
    9'b0_0001_0110, // 1 => 16
    9'b0_0001_1110, // 2 => 1E
    9'b0_0010_0110, // 3 => 26
    9'b0_0010_0101, // 4 => 25
    9'b0_0010_1110, // 5 => 2E
    9'b0_0011_0110, // 6 => 36
    9'b0_0011_1101, // 7 => 3D
    9'b0_0011_1110, // 8 => 3E
    9'b0_0100_0110, // 9 => 46

    9'b0_0111_0000, // right_0 => 70
    9'b0_0110_1001, // right_1 => 69
    9'b0_0111_0010, // right_2 => 72
    9'b0_0111_1010, // right_3 => 7A
    9'b0_0110_1011, // right_4 => 6B
    9'b0_0111_0011, // right_5 => 73
    9'b0_0111_0100, // right_6 => 74
    9'b0_0110_1100, // right_7 => 6C
    9'b0_0111_0101, // right_8 => 75
    9'b0_0111_1101  // right_9 => 7D
};
```

```
always@(*) begin
    case(last_change)
        KEY_CODES[00] : key_num = 4'b0000;
        KEY_CODES[01] : key_num = 4'b0001;
        KEY_CODES[02] : key_num = 4'b0010;
        KEY_CODES[03] : key_num = 4'b0011;
        KEY_CODES[04] : key_num = 4'b0100;
        KEY_CODES[05] : key_num = 4'b0101;
        KEY_CODES[06] : key_num = 4'b0110;
        KEY_CODES[07] : key_num = 4'b0111;
        KEY_CODES[08] : key_num = 4'b1000;
        KEY_CODES[09] : key_num = 4'b1001;
        KEY_CODES[10] : key_num = 4'b0000;
        KEY_CODES[11] : key_num = 4'b0001;
        KEY_CODES[12] : key_num = 4'b0010;
        KEY_CODES[13] : key_num = 4'b0011;
        KEY_CODES[14] : key_num = 4'b0100;
        KEY_CODES[15] : key_num = 4'b0101;
        KEY_CODES[16] : key_num = 4'b0110;
        KEY_CODES[17] : key_num = 4'b0111;
        KEY_CODES[18] : key_num = 4'b1000;
        KEY_CODES[19] : key_num = 4'b1001;
        default       : key_num = 4'b1111;
    endcase
end
```

B. Questions and Discussions

A. Regarding Note 2, we only need to handle the first key press and ignore the subsequent ones. How can this be achieved? How can you prevent continuous detection of a positive signal when a key is pressed and held down? E.g., in the SET state, pressing and holding '2' will add 5 dollars only once.

我參考 sample code 並在我的 module 中定義 wire [127:0] key_decode = 1 << last_change;，利用 key_decode 去 represent 哪個按鍵剛被 press/release，如此，我在所有判斷按鍵是否被按下的 if statement 中額外加上"key_down-key_decode==0"的判斷條件(也就是 key_down==key_decode)，即可確保在同一時間只有一個按鍵能有反應，因為如果在同一時間有超過兩個按鍵被按住，key_down 就不會等於 key_decode 了。

在 KeyboardDecoder.v 的 FSM 中有以下 code：

```
GET_SIGNAL_DOWN : begin
        state <= WAIT_RELEASE;
        key <= {been_extend, been_break, key_in};
        been_ready  <= 1'b1;
    end
WAIT_RELEASE : begin
        if (valid == 1) begin
            state <= WAIT_RELEASE;
        end else begin
            state <= WAIT_FOR_SIGNAL;
            been_extend <= 1'b0;
            been_break  <= 1'b0;
        end
    end
```

如果不去理會 valid 的值，無條件讓 state<=WAIT_FOR_SIGNAL 的話，如此一來，即使是長按按鍵不放，下一個 clock cycle 也會進入 WAIT_FOR_SIGNAL state，因此只會接收到按一次按鍵的資訊。下方為修正後的 code：

```
WAIT_RELEASE : begin
        state <= WAIT_FOR_SIGNAL;
        been_extend <= 1'b0;
        been_break  <= 1'b0;
    end
```

B. Regarding Question A, what can we do if we ignore Note 2? In this case, when a key is pressed first, another key can still become active (e.g., pressing two keys simultaneously.) You can explain your thoughts or use part of the code to illustrate.

如果忽略 Note 2 的要求，我們可以在同個時間點讓不止一個被同時按下的按鍵一起被偵測到(利用 key_down array 可以做相對應的判斷)，這樣的話，我們可以像 sample 一樣定義當按著 shift 鍵的同時按數字鍵以及沒有按 shift 鍵時按數字鍵時，產生不同的 display 效果。至於如何達到先按的按鍵不放時，後按的按鍵也可以有效果，方法很簡單，利用 last_change 的值可以幫助我們得知最近一次 press/release 的按鍵是哪個，進而做出對應的操作。

C. Problem Encountered

一開始在實作"判斷按鍵是否有被按下"的功能時，我使用以下 code：

```
if(key_valid && key_down[last_change] == 1'b1 && key_num!=4'b1111 && key_down-key_decode==0) begin  //press number key
    if(LED[15]==1'b1) begin  //set noi
        next_noi=key_num;
    end else begin  //set price
        next_price={price[3:0],key_num};
    end
end
```

第一行的 if statement 在判斷數字按鍵是否有被按下(以及是否只有單一按鍵同時被按下)。結果實際用鍵盤測試時發現我按數字鍵時沒有產生任何變化，7-seg 上沒有顯示我 press 的 key number。但因為我當時是參照助教給的 sample，還蠻肯定出錯的地方應該不在這裡，所以我認真重新 trace 一遍助教的 sample code 和我的 code 的差異處，才發現在 KeyboardDecoder.v 中，press 或是 release 一個按鍵會使 key_valid signal 被 trigger 剛好一個 clock cycle，而 key_valid 所使用的 clock frequency 和 FSM's clock frequency 是一致的。而在我起初的 design 當中，兩者的 clock frequency 並不一致，我的 FSM 的 clock frequency 遠低於 key_valid signal 的 clock frequency，導致當 key_valid 被 trigger 一個 clock cycle 時，我的 FSM 不一定會遭遇 posedge，所以無法把 key_valid==1'b1 的資訊 update 到 FSM 當中，進而影響到上方 if statement 的判斷。以下兩張圖的 clock frequency 必須一致：

```
always@(posedge clk or posedge rst) begin  //clk_div1
    if(rst) begin
        state<=IDLE;
        nums<=16'b1010101010101010;  //"----"
        LED<=16'b0;
        cnt<=32'd0;
        noi<=4'd9;
        price<={4'd1,4'd0};
        pay<=8'd0;
    end else begin
        state<=next_state;
        nums<=next_nums;
        LED<=next_LED;
        cnt<=next_cnt;
        noi<=next_noi;
        price<=next_price;
        pay<=next_pay;
    end
end
```

```
always @ (posedge clk, posedge rst) begin
    if (rst) begin
        key_valid <= 1'b0;
        key_down <= 511'b0;
    end else if (key_decode[last_change] && pulse_been_ready) begin
        key_valid <= 1'b1;
        if (key[8] == 0) begin
            key_down <= key_down | key_decode;
        end else begin
            key_down <= key_down & (~key_decode);
        end
    end else begin
        key_valid <= 1'b0;
        key_down <= key_down;
    end
end
```

D. Suggestions

每次在寫 report 時，遇到最棘手的問題就是如何畫好 block diagram，雖然最近有提供之前 lab2, lab3 的 report 範本，但還是希望可以在課堂上教學如何畫 block diagram，不然每次都不確定自己是否畫得夠清楚或是畫得太細微。