Lab 5

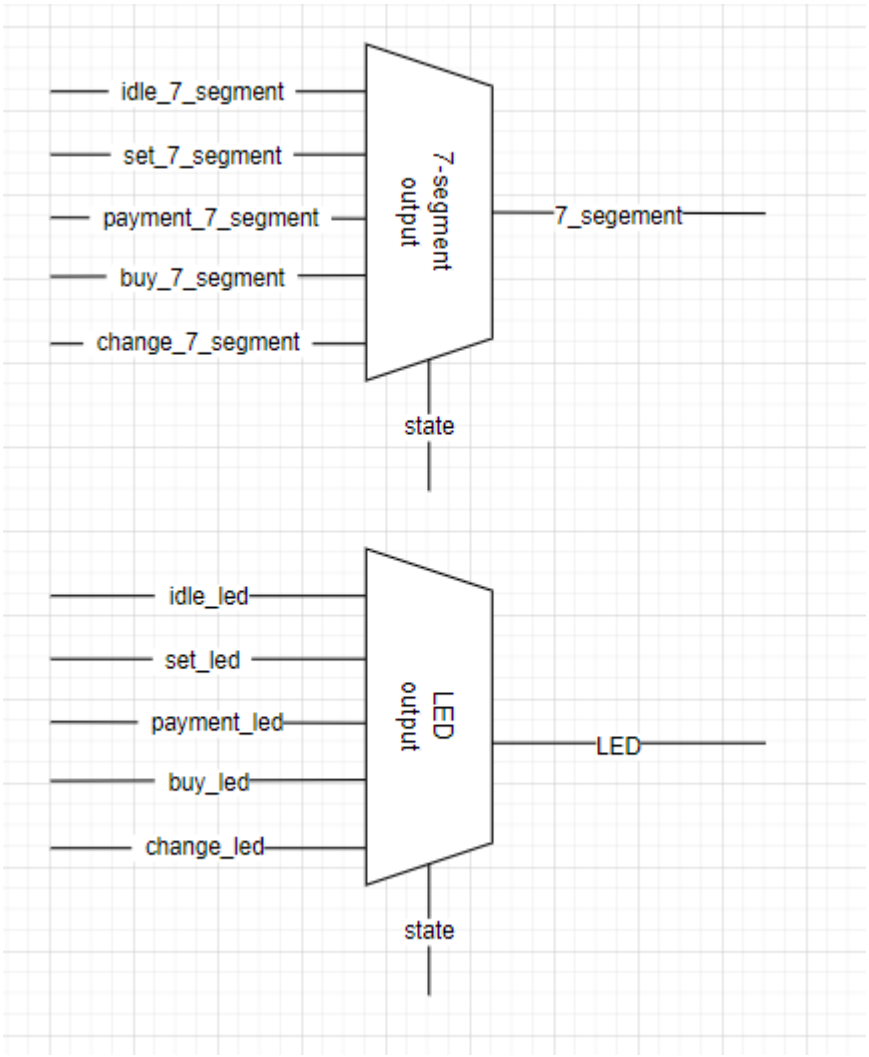
學號: 109062173 姓名: 葉昱揚

A. Lab Implementation

1. Block diagram

State 很多，因此分段解釋 block diagram。

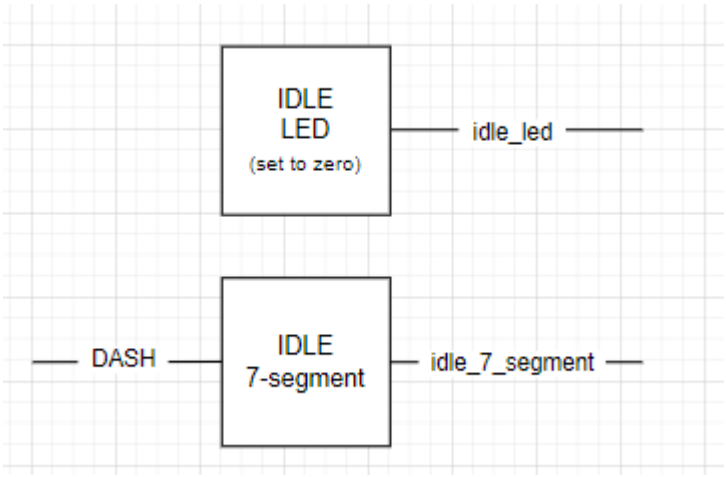
OUTPUT:



FSM 的每個 state 都有自己的 7-segment 和 led，並根據 FSM 的規則變化計算出 state，讓真正的 output 7-segment 和 output led 輸出對應 state 的數據。

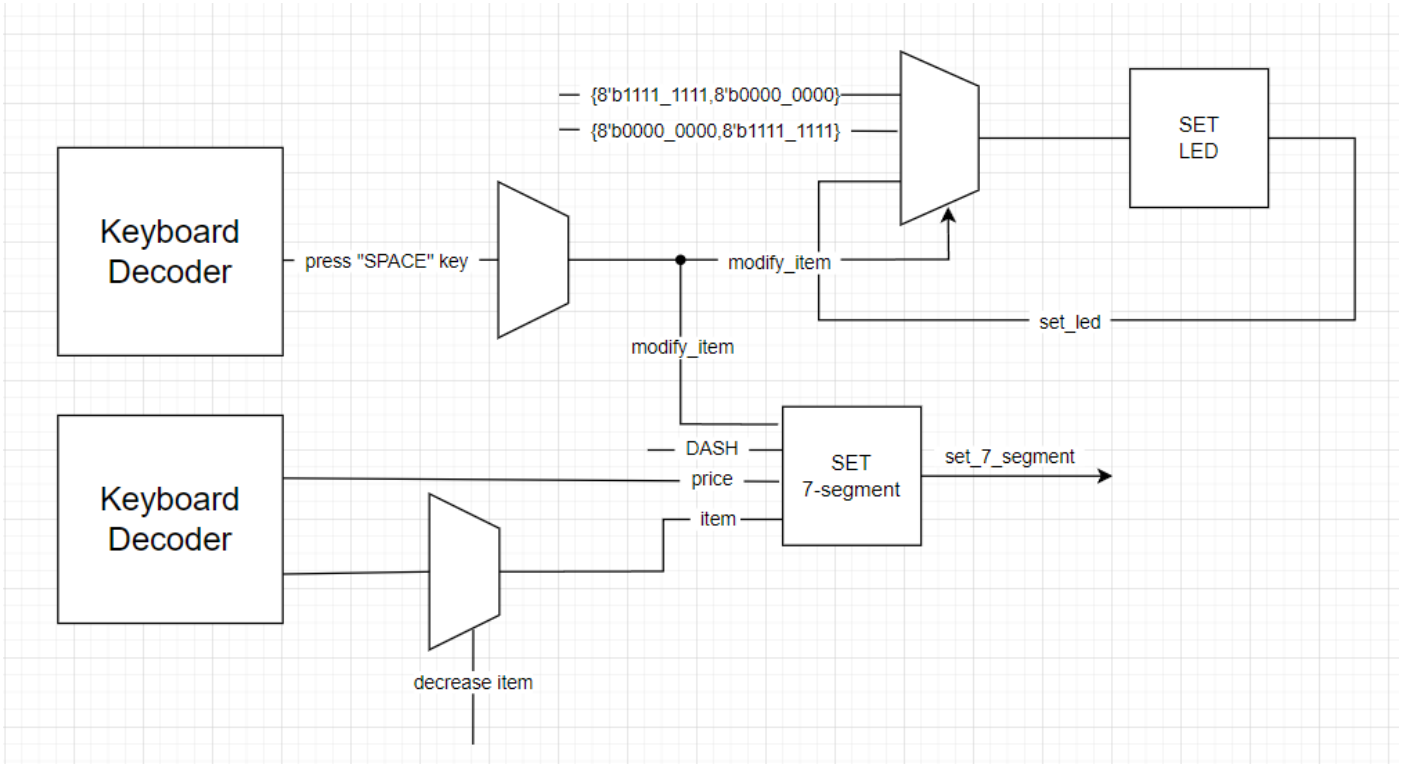
稍後各個 state 的 block diagram 的 7-segment 和 led 都不會把線拉到這邊
(為了讓版面更乾淨，也不是需要表達的重點)。

IDLE state:



這個階段很簡單，不需要做任何操作。
需要 led 全部設為熄滅(all zero) 和 7-segment 全部設為 DASH。

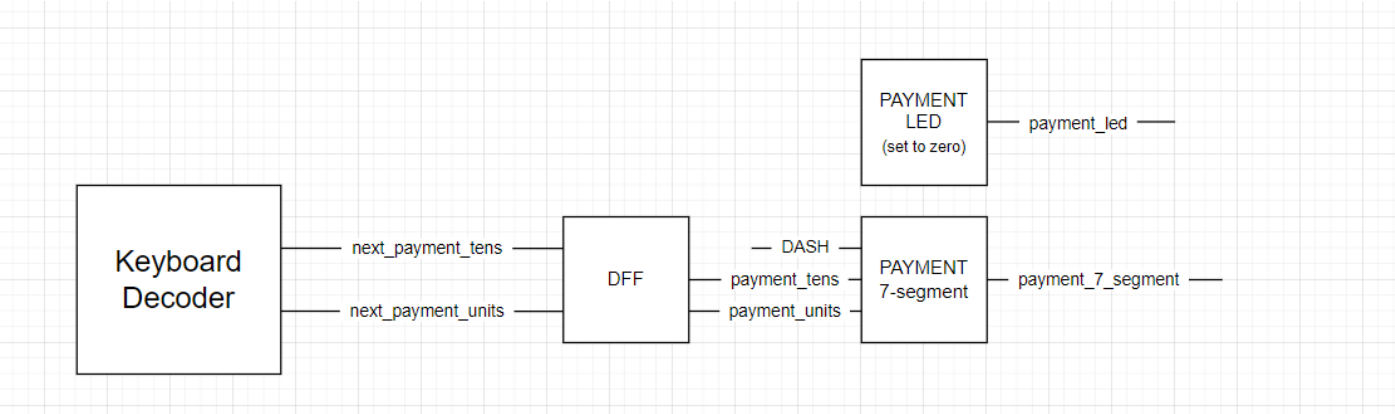
SET state:



這個 state 需要設定商品數量、商品價格。
空白鍵可以控制要設定數量 or 價格，上圖的 modify_item 就是控制這件事情的 enable signal。
price=價格 、item=數量，他們的值等同於按下的鍵盤代表的值。
decrease item 是用來控制”如果賣出 n 件商品，則商品數量需要扣掉 n 件”這件事情。

set_led 在設定數量時，為左邊 8 個燈亮起、右邊 8 個燈熄滅。
set_led 在設定價格時，為左邊 8 個燈熄滅、右邊 8 個燈亮起。

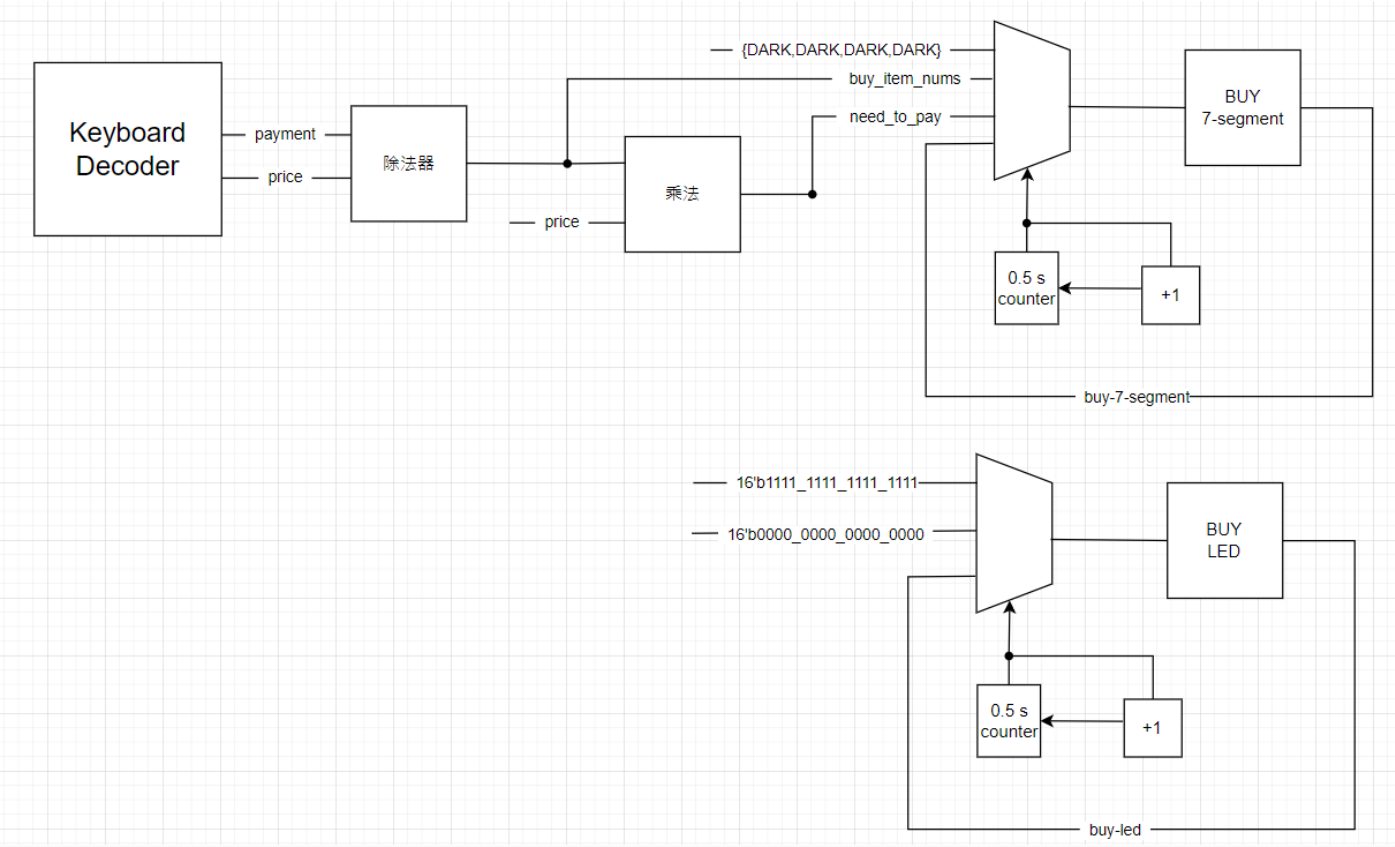
PAYMENT state:



這個 state 需要輸入付多少錢。
next_payment_tens 代表付的錢的十位數、next_payment_units 代表付的錢的個位數，這兩個數值用壓下的按鍵得來，並顯示在 7-segent 之後。

payment_led 在這邊保持全暗(all zero)。

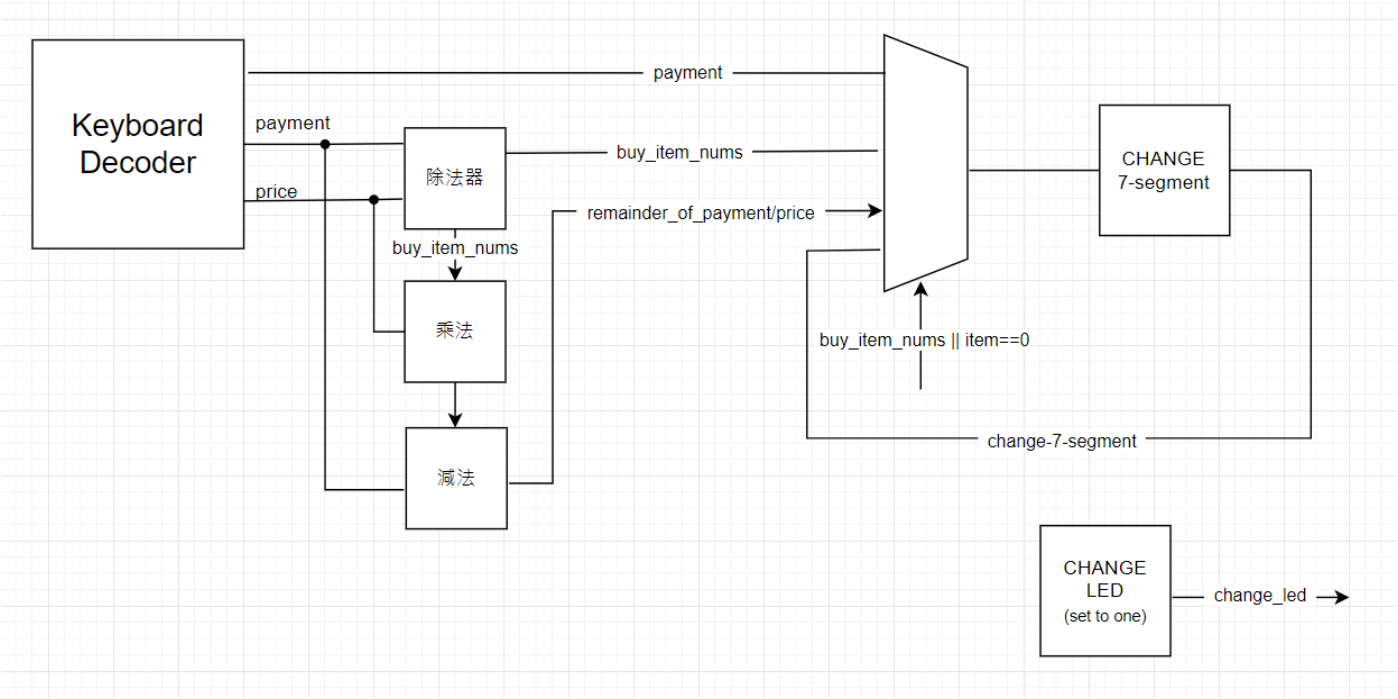
BUY state:



這個 state 需要顯示購買數量(buy item nums)和花費金錢(need to pay)。
這邊除法比較複雜，不能直接使用 payment/price，因此額外寫了一個除法器 module。
buy_item_nums 是 payment、price 放進除法器算出來的商，need_to_pay 是 buy_item_nums*price。
這兩項數值需要根據 0.5s counter 來顯示在 7-segment 之上。0.5 counter 可以準確計算經過的時間，每經過 0.5s，7-segment 會 flash and off(亮起和熄滅)。

Led 同理，每經過 0.5 秒就會亮起/熄滅。

CHANGE state:



這個 state 需要顯示購買數量(buy item nums)和剩餘金錢(remainder)。
進入這個階段有兩個條件，有買到東西 or 商品數量為 0。
IF 有買到東西，7-segment 需要顯示 (1)買了多少東西(buy_item_nums) (2)購買後剩餘金錢
IF 商品數量為 0，7-segment 需要顯示 (1)支付多少錢買商品(payment)

Led 在這個階段為全亮(set to one)

2. Partial code screenshot with the explanation

```
wire [7:0] payment_dec;
assign payment_dec = 4'd10*payment_tens + payment_units;

wire [7:0] price_dec;
assign price_dec = 4'd10*price[7:4] + price[3:0];

wire [7:0] quotient_of_payment_div_price;
wire [7:0] remainder_of_payment_div_price;

division payment_div_price(.A(payment_dec),.B(price_dec),.Res(quotient_of_payment_div_price));
assign remainder_of_payment_div_price = payment_dec - price_dec*buy_item_nums;

reg [3:0] buy_item_nums;
always@(*)begin
    if(quotient_of_payment_div_price > item) buy_item_nums = item;
    else buy_item_nums = quotient_of_payment_div_price[3:0];
end
```

(除法器使用展示)
正常來說，想要得到 payment 和 price 的商，會做 payment/price。但我把這樣做的結果燒到板子上後結果會錯。後來發現"/"在 verilog 的合成很複雜，於是弄了個除法器的 module，並把 payment 的

十進位 payment_dec、price 的十進位 price_dec 丟進去，得到的商 quotient，再去和目前擁有的商品數比較，得出真正的購買商品數量(buy_item_nums)。

```
reg counter;
always@(posedge clk) begin
    if(key_down[last_change]) begin
        counter <= 1;
    end
    else begin
        counter <= 0;
    end
end
```

(解決長壓連續觸發)
處理方法是每當壓下一個按鈕，這個按鈕只有 1 clock cycle 的時間能夠操作，後面的時間都不會繼續反應。
如上圖當經過 1 clk 後，counter 為 1，這個 counter 會是一把鎖住鍵盤輸入的鎖(lock)，避免後續的輸入，如下圖 always block 最後一句 if(counter==1) lock=1。
無論壓著同個按鍵多久，都只會在壓下的第一個 clk 內做反應，後面的時間都會忽略。

```
reg lock;
integer i;
always@(*) begin
    if(been_ready && key_down[last_change] == 1'b1) begin
        lock = 0;
        for(i=0;i<=150;i=i+1)begin
            if(key_down[i] && (i!=last_change)) lock = 1;
        end
    end
    if(counter==1) lock=1;
end
```

(解決按下按鈕後再按其他按鈕也有反應)

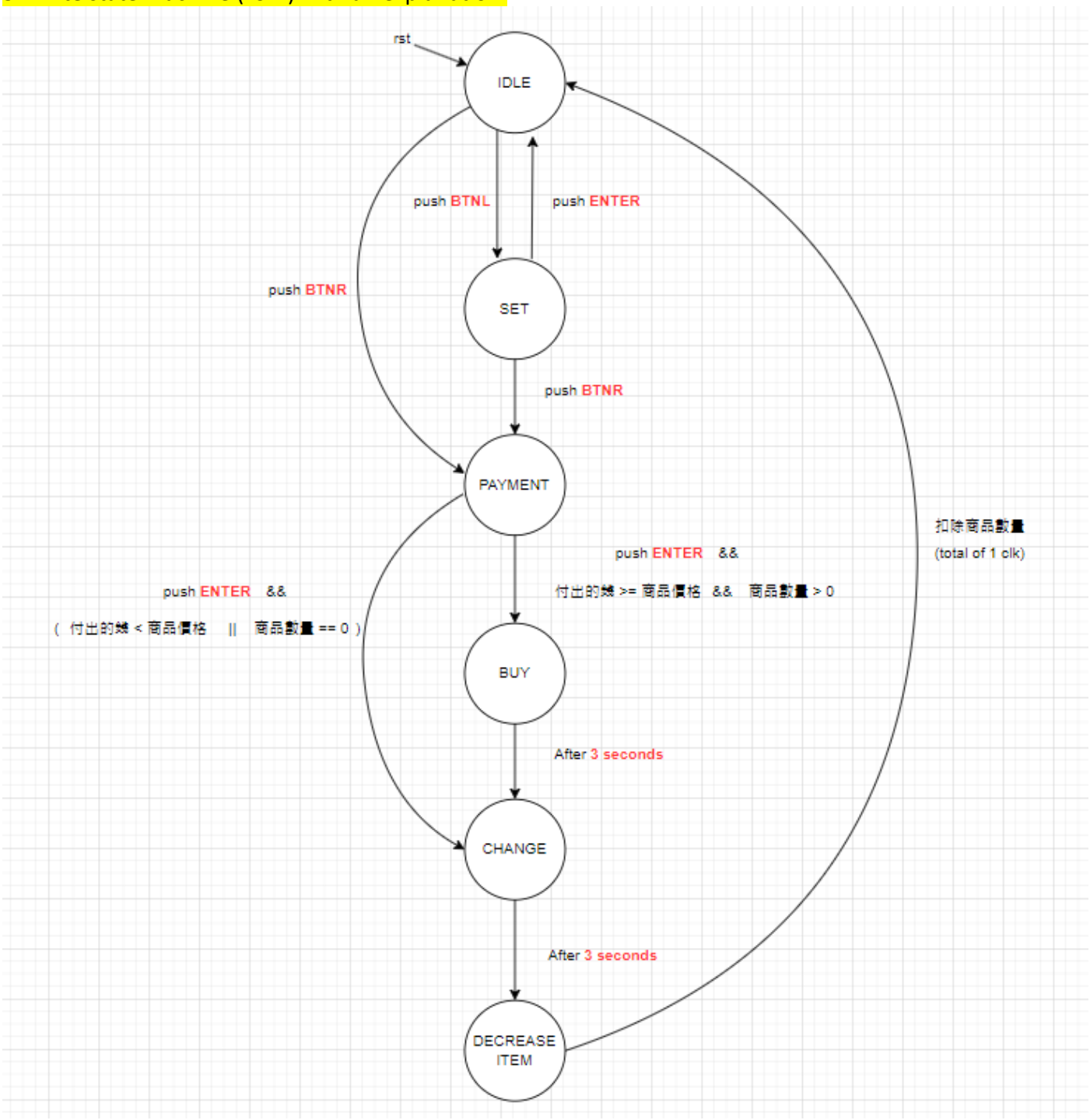
處理方法是每壓下一個按鍵，檢查 key_down 裡面除了"當前壓下的按鈕"以外"是否有其他按鈕也壓下"。更直觀一點，檢查全部 key_dow 是否只有 0 or 1 bit 為 1，若超過就鎖住鍵盤輸入(lock=1)

```
if (!lock && been_ready && key_down[last_change] == 1'b1) begin
    if (key_num != 4'b1111)begin
        case(state)
            SET:begin
                if(modifyItem) begin
                    item <= key_num;
                end
                else begin
                    price <= {price[3:0],key_num};
                end
            end
```

可以看到 if statement 裡面有個!lock，即是配合上面兩個方法擋住鍵盤輸入。

原本的寫法沒有!lock，所以長壓或按多個按鍵時，都可以通過 if statement 裡面的 been ready&&key_down[last_chaneg]，造成本次 Lab 我們不想看到的效果。

3. Finite state machine (FSM) with an explanation.



設計了 6 個 state，為 IDLE、SET、PAYMENT、BUY、CHANGE、DECREASE_ITEM。

IDLE:

Rst 會回到這個 state，是整個 FSM 的起點。

按下 BTNL，會從 IDLE 跳到 SET

按下 BTNR，會從 IDLE 跳到 PAYMENT

SET:

按下 ENTER，會從 SET 跳到 IDLE

按下 BTNR，會從 SET 跳到 PAYMENT

PAYMENT:

需要審查付出的錢和商品價格才能決定要跳到哪一個 state。

IF 成功買到商品且按下 ENTER，跳到 BUY

IF 沒買到商品(付太少錢 or 商品賣完) 且按下 ENTER，跳到 CHANGE

BUY:

3 秒後跳到 CHANGE。

CHANGE:

3 秒後跳到 DECREASE_ITEM。

DECREASE_ITEM:

1 clock cycle (100MHZ) 後跳到 IDLE。

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.

處理多個按鍵: 檢查全部的 key_down 的值是否超過 1 bit 為 1。
因為壓下 n 個按鍵就會有 n 個 bit 為 1，我們的目標是只讓第一個按鍵反應其他都忽略，可以檢查全部的 key_down 的值是否超過 1 bit 為 1。
若"按下當前按鍵"不超過 1 bit 為 1，則在板子上反應按鍵輸入結果。
若"按下當前按鍵"超過 1 bit 為 1 ，則忽略第 2~n 個進來的按鍵輸入。

處理長壓按鍵: 只有在按鍵輸入進來的第一個 clk 做操作。
最小的時間單位就是 1 clk，但凡長壓都會超過 1 clock cycle，甚至說我們理解中的"按一下鍵盤"也會超過 1 clokc cycle。無論如何，解法都是只在壓下後的第一個 clock cycle 做對應的操作，後面長壓通通忽略掉。

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.

目標是允許多個按鍵同時壓下，但都不會連續觸發。

處理長壓按鍵一樣是: 只有在按鍵輸入進來的第一個 clk 做操作。
允許多個按鍵我認為可以幫每一個按鍵都開一個新的 counter vector，紀錄已經被壓下多久。
比如說 A 被壓下，則經過 1 clk 後 counter[A] = 1，同時 B 被壓下，則經過 1 clk 後 counter[B]=1。
counter 的數值直到對應的按鍵放開才會重製為 0。
這期間我們要做的判斷就是，每當有按下被壓下，檢查它的 counter:
If counter==0，在板子上做出對應的反應持續 1 clk cycle。
If counter==1，忽略這個按鍵輸入。

C. Problem Encountered

```
25
26
27    module KeyboardDecoder(
28        output reg [150:0] key_down,
29        output wire [8:0] last_change,
30        output reg key_valid,
31        inout wire PS2_DATA,
32        inout wire PS2_CLK,
33        input wire rst,
34        input wire clk
35        );
36
```

(降低 keyboardDecoder 使用空間)
原本為 out reg [511:0] key_down，但是會無法合成。經過助教提示後知道是使用空間過多，在確認本次 lab 鍵盤輸入的 make code 都不超過 150 後，把空間降到 150。

可以從下圖得知，make code 最大值是 7D，10 進位是 125。

```
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
};
```

另外的問題就是怎麼處理長壓和按壓多個按鍵，我已經放在 partial code with screen shot 那邊，覺得放在那邊更適合。

D. **Suggestions**

No suggestions.

附上迷因