硬體設計與實驗 Final project report

1. 組員名字及學號: 103062124　楊翔閔　　103062305　林宛萱

2. 題目以及題目說明

- 題目: 菁英操作提款機
- 題目說明: 選擇實作提款機這個主題是因為，提款機需要完整的應用這學期 lab 所學過的所有技術，包含 FSM、audio 以及 VGA 顯示的部分，並且將這些技術做適當的延伸才能完成，所以我們認為對我們而言這是一個難度適中的 project。

  在著手實作的這段時間裡，我們碰巧聽說了系上有同學遭到詐騙集團的話術欺騙，一不小心將戶頭的餘額全數匯給詐騙集團的悲慘事件，透過這個事件，我們發現到了現代一般使用的提款機可以改進的部分，大多數的提款機都支援到好幾萬塊的提領或是轉帳功能，但考量到一般人以及學生族群正常在使用提款機的狀態下並不會需要使用到如此大筆的款項(即使需要，高額的提領及轉帳功能依然是危險的，如上述同學的例子，不如直接臨櫃進行大筆金額的提領及轉帳)，因此我們設計出了這款安全型提款機，轉帳及匯款最大數目都比一般的提款機來的少，但相對的更安全，也不支援將戶頭內的錢全數轉帳的功能，以減少戶頭被提領一空的悲劇，來服務民眾。

3. 板子上面各個硬體的使用方式

- 插卡:switch 中的 SW0~4，每個 switch 對應一張提款卡。
- 領錢:switch 中的 SW15。
- 機器內有卡片時亮燈:LD0。
- 輸入密碼每輸入一位亮一個燈:LD10~15。
- 輸入密碼、輸入金額錯誤:LD1。
- 戶頭內金額不足:LD2。
- 是否繼續進行其他交易:LD3。
- 轉帳中亮燈:LD4。
- 自行輸入金額亮燈:LD5。
- 決定提領或轉帳之金額亮燈:LD6。
- 輸入欲轉帳之卡號亮燈:LD7。
- 選擇服務項目亮燈:LD8。
- 可以輸入密碼亮燈:LD9。
- 聲音: Pmod JB1~6。
- 顯示金額、倒數秒數:7 segment。
- Reset 功能:Button BTNU。
- 輸入密碼、金額:鍵盤右邊的 0~9。
- 更正輸入錯誤之密碼或金額: 鍵盤 Backspace。

- 確認輸入密碼或金額:鍵盤 Enter。
- 若不想自行輸入金額可返回選取固定金額之返回鍵: 鍵盤 Shift。
- 選取自行輸入金額:鍵盤 A。
- 選取一千元:鍵盤 B。
- 選取兩千元:鍵盤 C。
- 選取五千元:鍵盤 D。
- 選取一萬元:鍵盤 E。
- 確認繼續進行其他交易:鍵盤 O。
- 不需繼續進行其他交易:鍵盤 X。

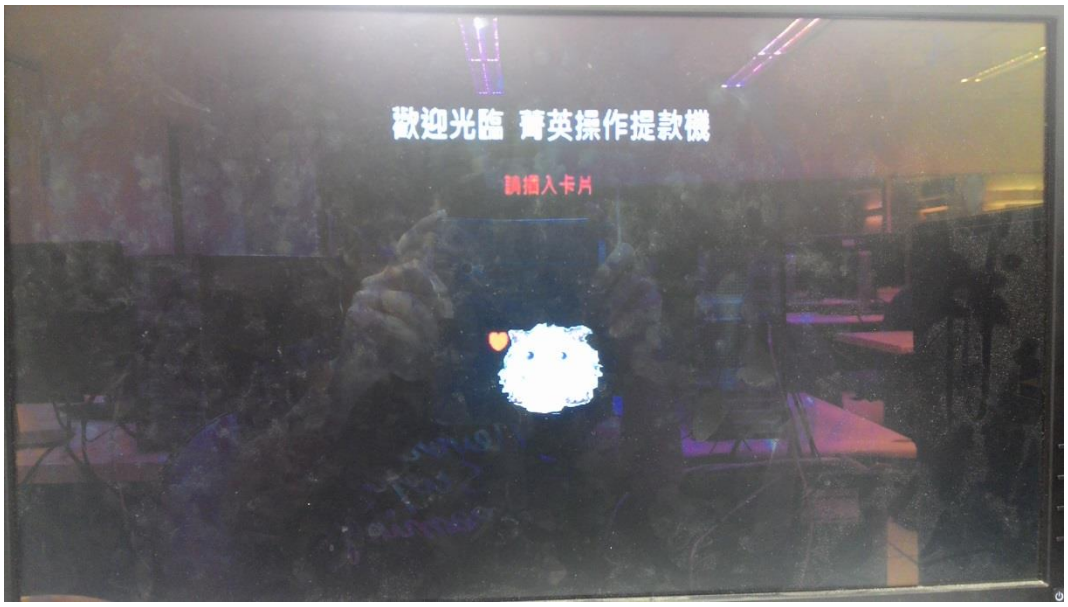
4. 功能描述

- 共五個帳戶、對應五組密碼，每個帳戶之間沒有跨行轉帳需收取手續費之問題。
- 主要功能 : 領錢及匯款、查詢餘額。
- 領錢最多可領一萬元 ---- 防搶劫!!!!
- 匯款最多可匯一萬元 ---- 防詐騙!!!!
- 輸入密碼、金額時的更正鍵 ---- 人性化!!!!
- 自行輸入金額時可按返回鍵 ---- 人性化!!!!
- 確認領錢金額後，若未在 30 秒內收取現金，則現金將被機器

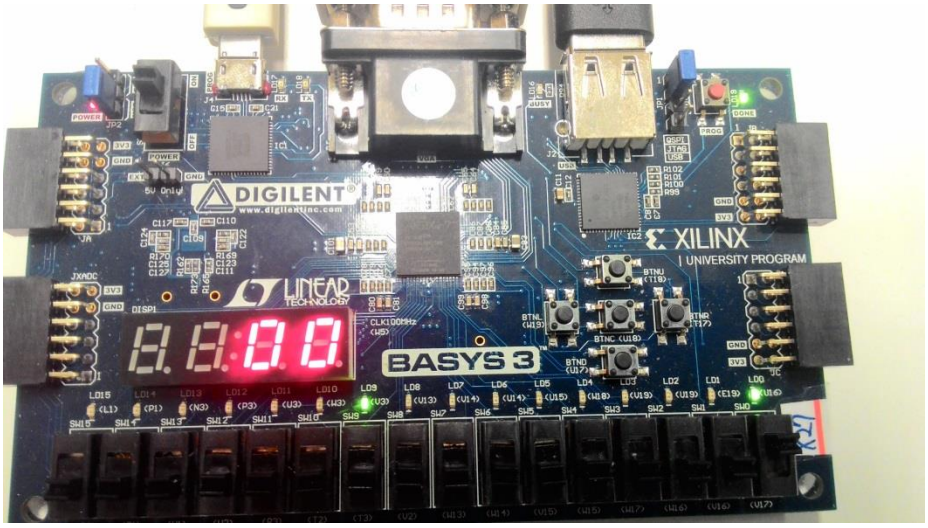
收回，帳戶將不會扣款。

- 退卡後、若未在 10 秒內取卡，則機器將再次自動插入卡片，並進入輸入密碼的狀態。
- 若有進行任何交易動作(提款及轉帳)，但帳戶內餘額不足的話，將會亮燈顯示餘額不足，所進行的任何交易動作皆不成立，接著進入是否繼續進行交易之畫面，並顯示現在所剩之餘額。
- 若輸入錯誤的密碼或不合法的金額輸入(非一千元的倍數且小於等於一萬元)，則會顯示輸入錯誤。
- 示範實作

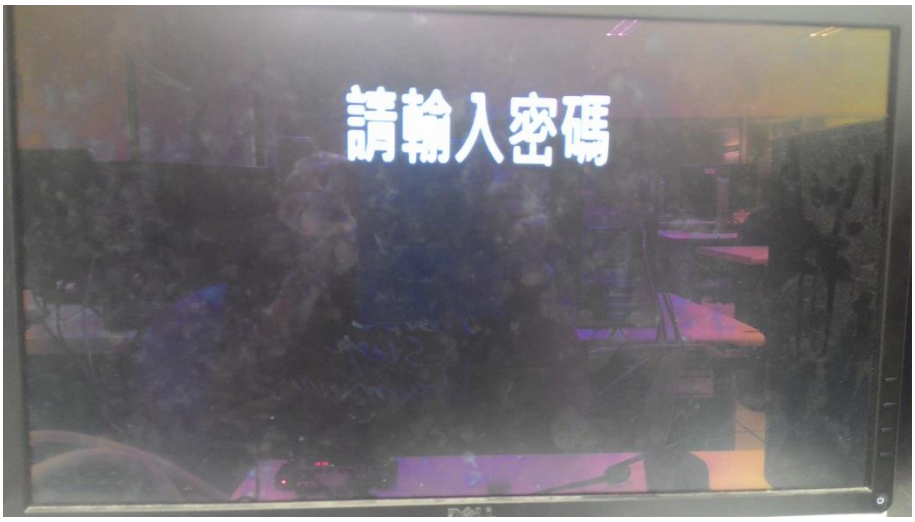
<示範一> 查詢功能



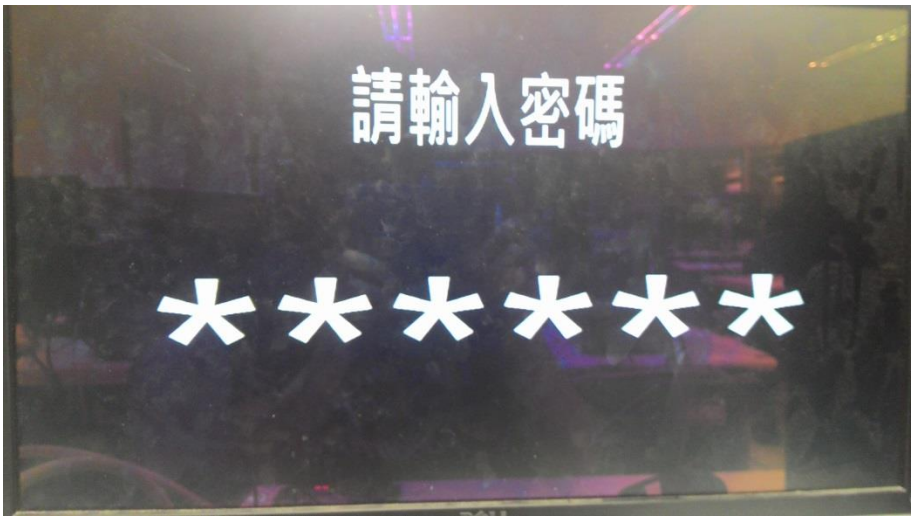
起始畫面



Switch 扳開，插入卡片一之後，warning 會亮起顯示此時無法再插卡



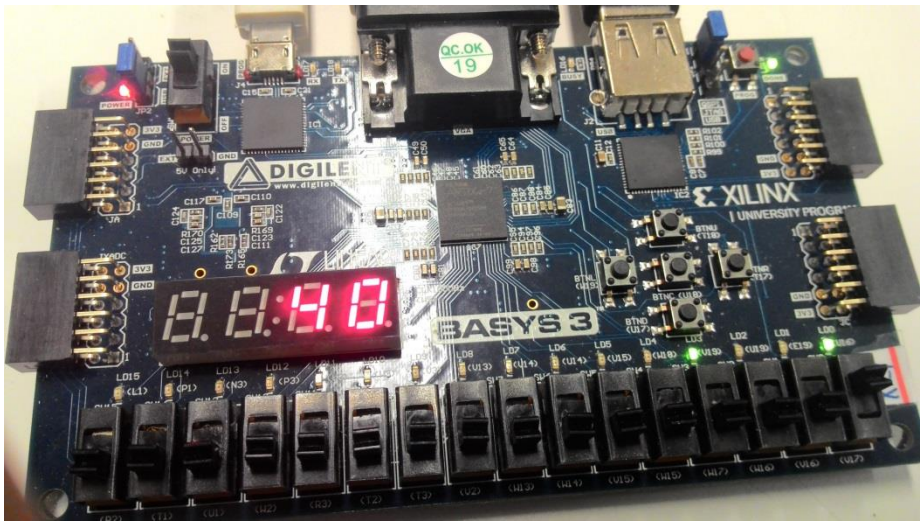
同時螢幕要求輸入密碼



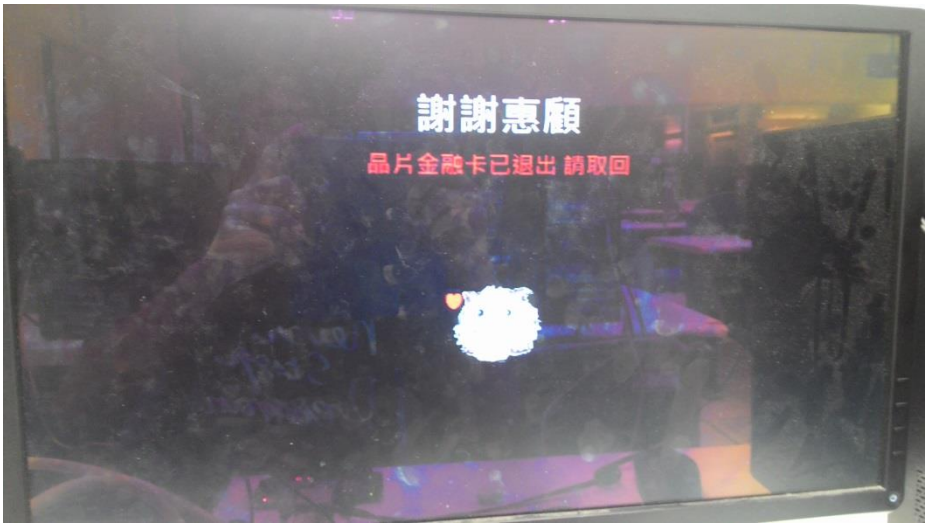
輸入密碼的狀態

進入要求選擇服務項目的畫面

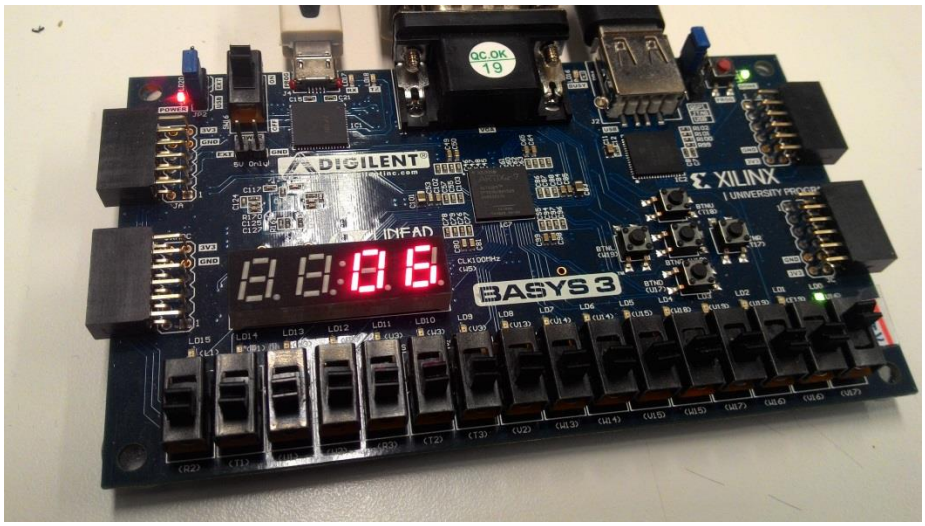
先選擇查詢餘額，螢幕會詢問是否繼續進行其他交易



同時 7 seg 上會顯示餘額，說明卡片一中的餘額有 4 萬元



選擇不繼續進行交易，則會顯示謝謝惠顧的畫面

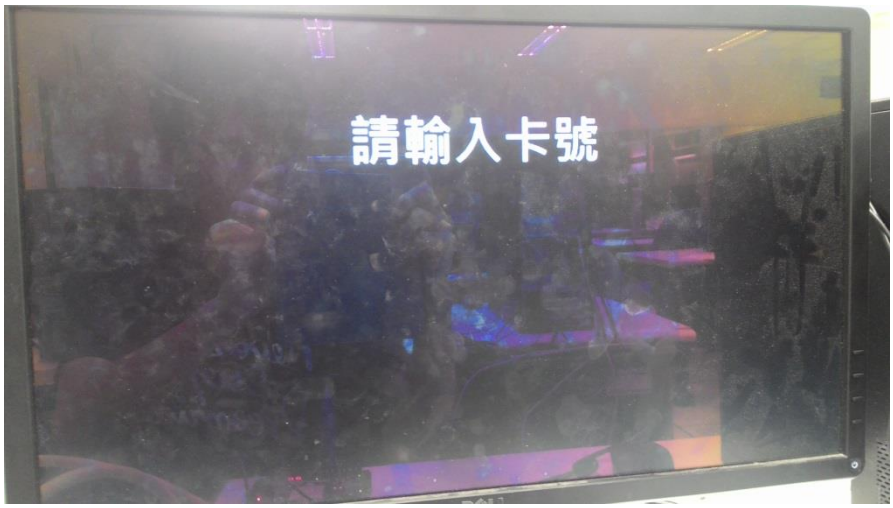


同時 7 seg 會進入倒數，要求在 10 秒內把卡片抽離(把開關扳回)

如此一來，便完成了提款機的查詢功能之示範，接下來，我們接著看查詢的功能。

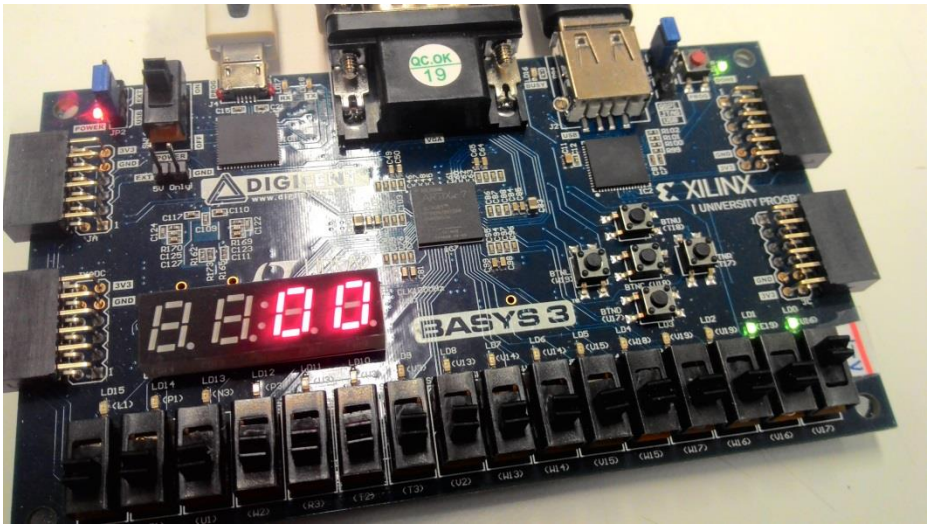
<示範二> 轉帳功能以及餘額不足之警示燈

再次插入卡片一，並在服務項目中選擇轉帳

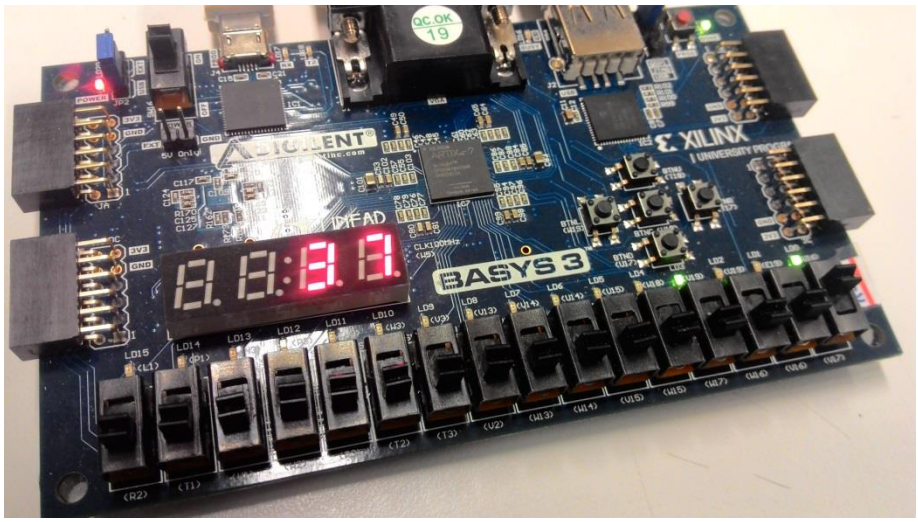


系統會要求使用者輸入欲轉入之卡號

接著會要求使用者輸入欲轉帳之金額

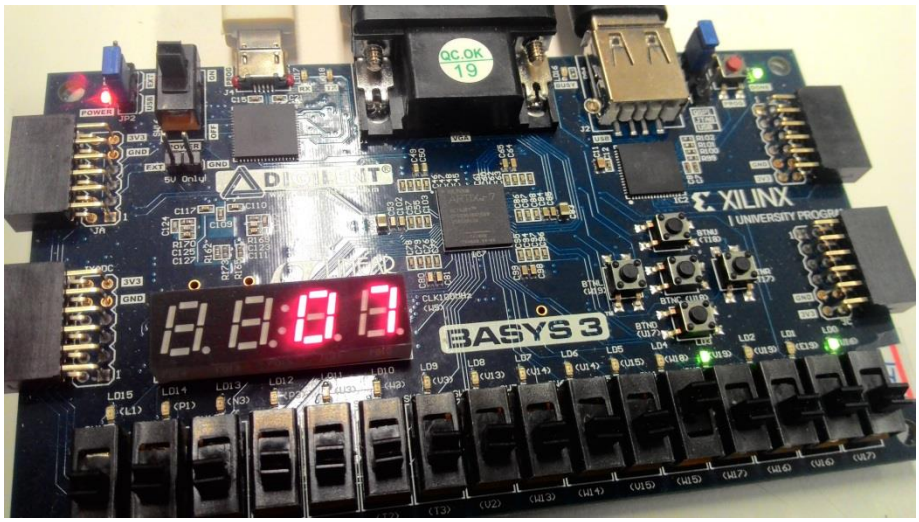


選擇自行輸入金額，但輸入非合法金額的話，系統會顯示警示燈

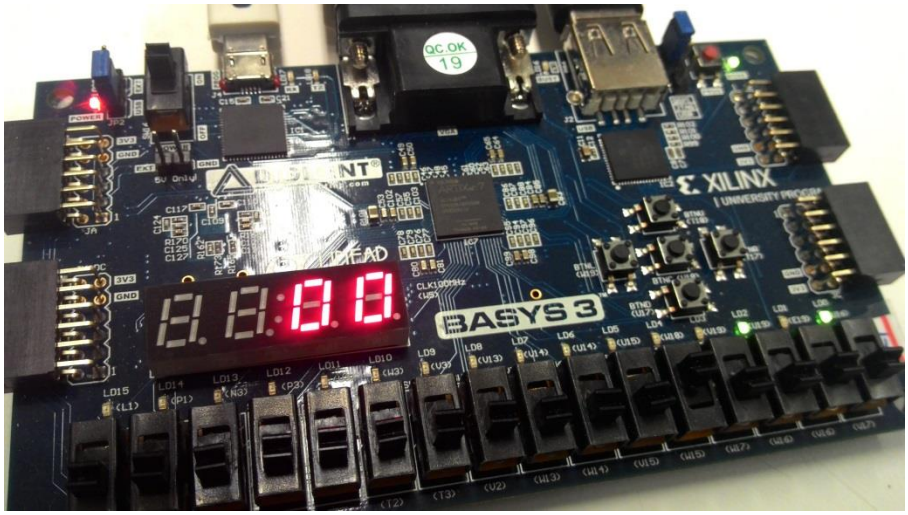


輸入合法金額的話則會正確扣款

並且詢問是否進行其他交易



接著插入卡片五，並執行查詢餘額，會發現已經正確匯款了

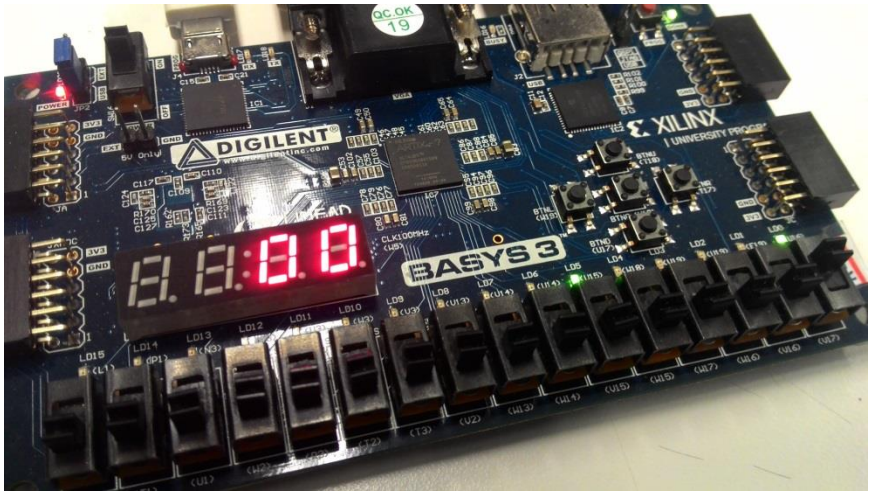


接著我們選擇繼續交易，並領取一萬元，但卡片五中實際上只有七千元，餘額不足的警示燈便會亮起，交易無效

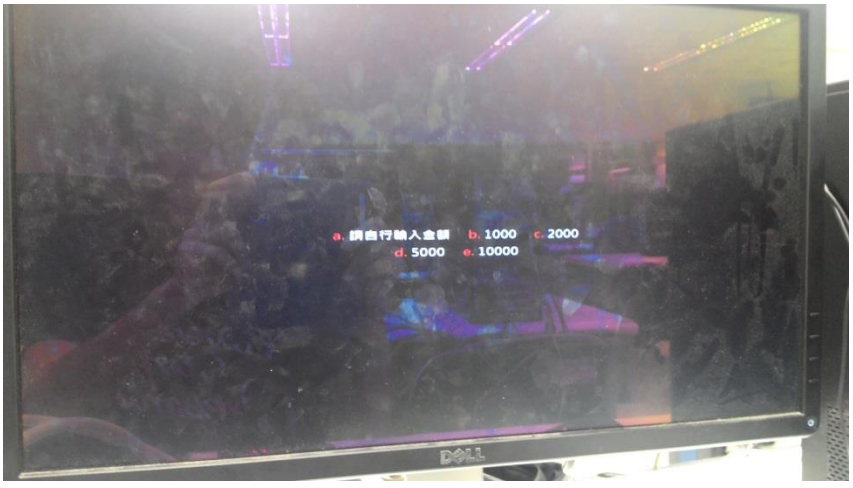
以上便是轉帳及餘額不足之情形實作，接著我們進行提領的示範。

<示範三> 提款

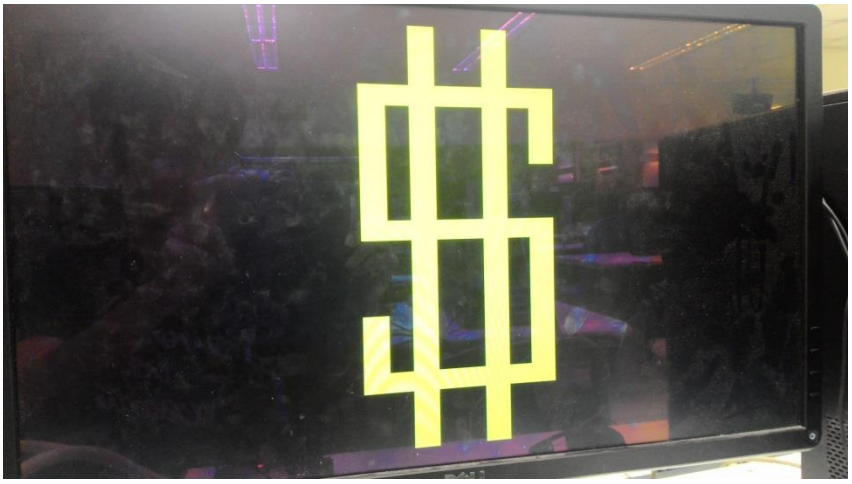
Reset 一次之後插入卡片一，在服務項目中選取領錢便會進入此畫面



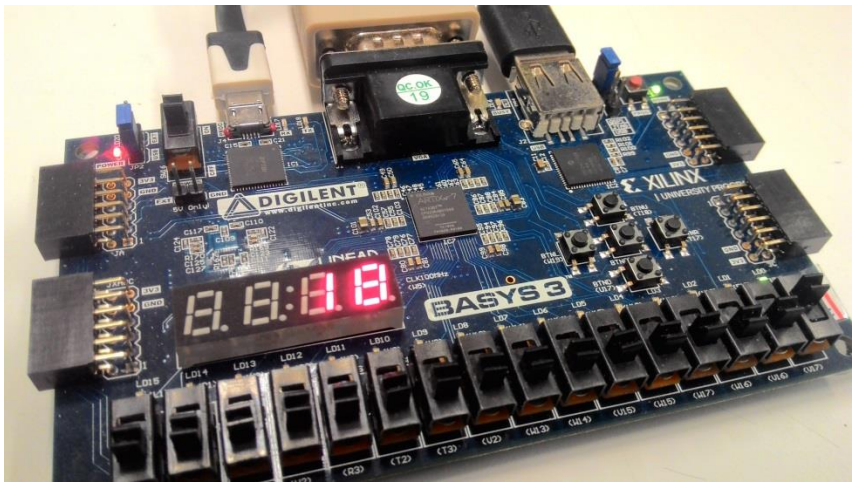
選擇自行輸入金額之後，LD5 會亮起，此時可輸入金額，和轉帳時相同，若輸入不合法的金額，警示燈依然會亮起



若突然想選擇固定金額，按下 Shift 即可返回至本畫面

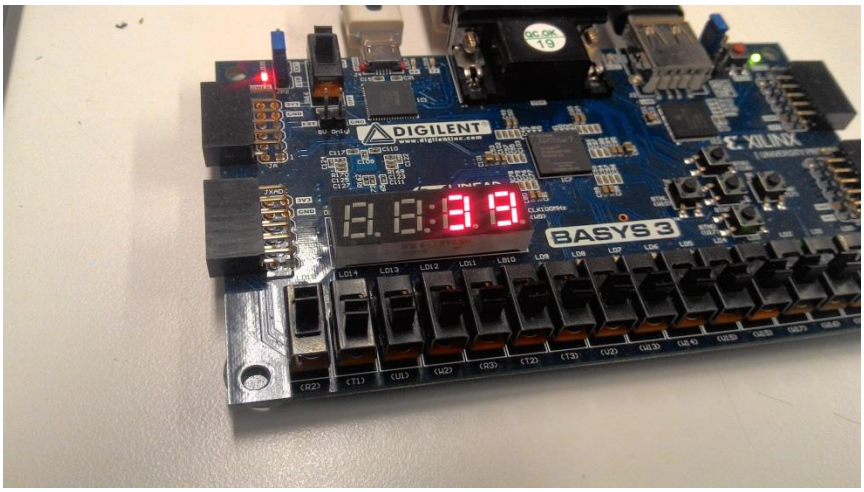


選擇 b 提領一千元之後，會進入此畫面，代表已經可取鈔



7 seg 會進入 30 秒的倒數，需在 30 秒內取鈔，否則鈔票將會收回，

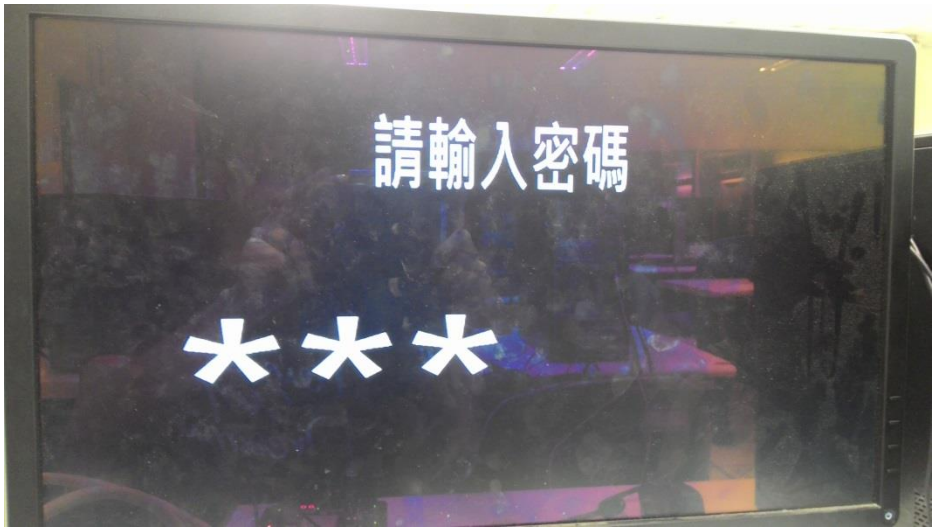
使用者之戶頭將不會被扣款



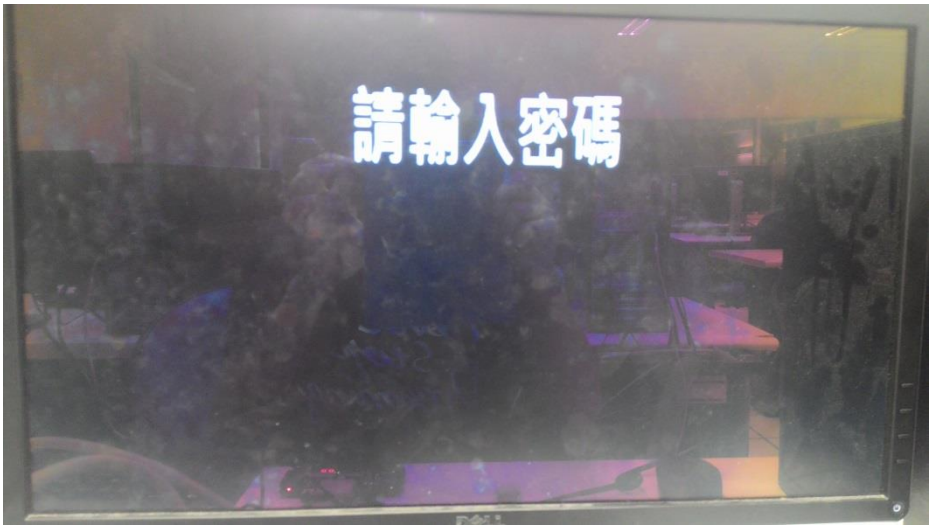
扳上 SW15 代表已取鈔，會顯示餘額，螢幕會詢問是否繼續進行交易

以上即是提款之示範。

<示範四> 輸入錯誤修正



假設在輸入密碼途中發現輸入錯

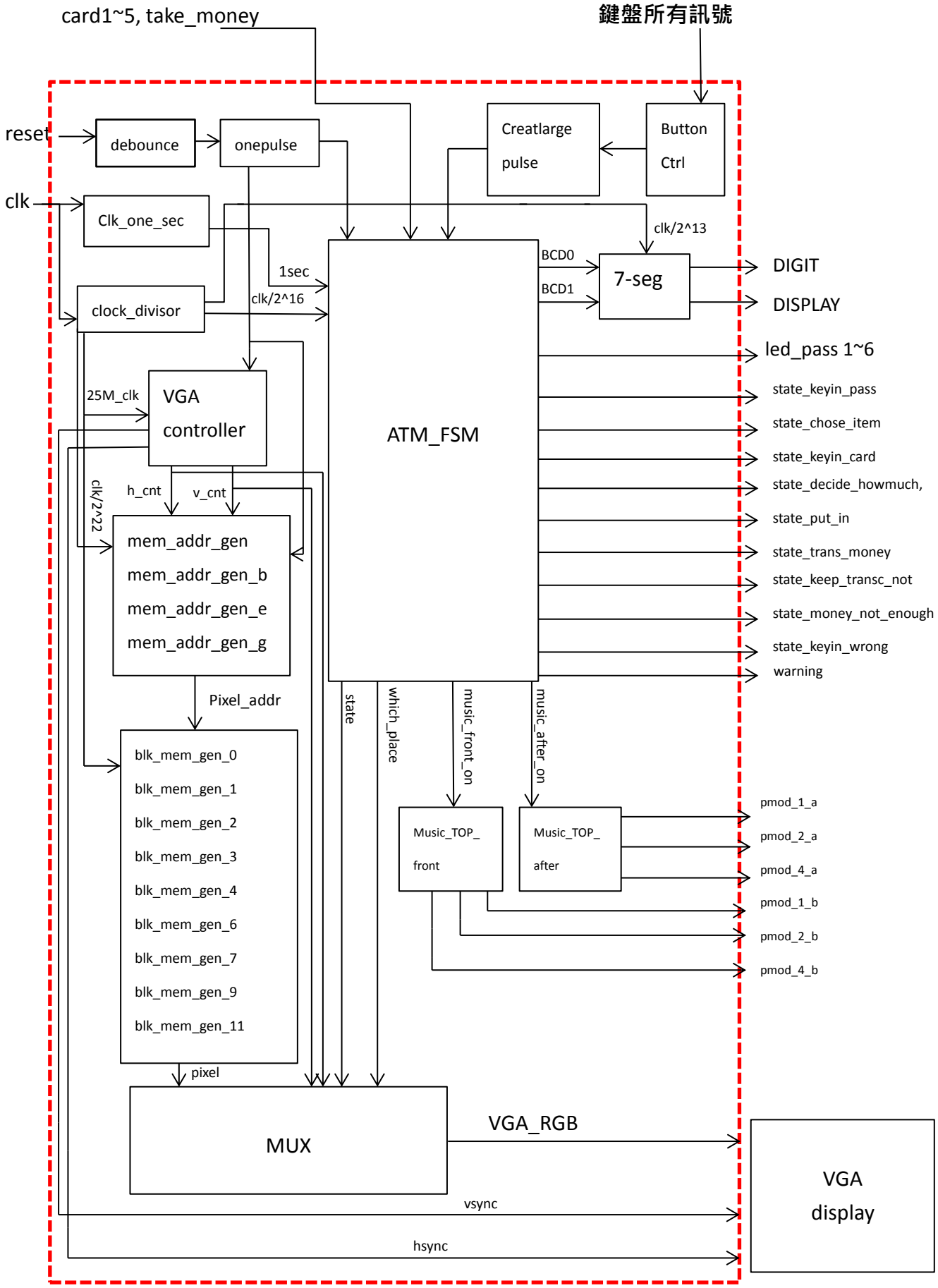


按下鍵盤上 Backspace 鍵即可清空重新輸入，再自行輸入金額的過程

中亦可執行此項操作修正輸入錯誤之金額

以上即是所有示範操作的過程。

5. 設計架構以及電路方塊圖


card1~5, take_money
鍵盤所有訊號
reset
debounce
onepulse
Creatlarge pulse
Button Ctrl
clk
Clk_one_sec
clk/2^13
1sec
clk/2^16
clock_divisor
BCD0
BCD1
7-seg
DIGIT
DISPLAY
led_pass 1~6
25M_clk
VGA controller
ATM_FSM
state_keyin_pass
state_chose_item
state_keyin_card
state_decide_howmuch,
state_put_in
state_trans_money
state_keep_transc_not
state_money_not_enough
state_keyin_wrong
warning
clk/2^22
h_cnt
v_cnt
mem_addr_gen
mem_addr_gen_b
mem_addr_gen_e
mem_addr_gen_g
Pixel_addr
state
which_place
music_front_on
music_after_on
blk_mem_gen_0
blk_mem_gen_1
blk_mem_gen_2
blk_mem_gen_3
blk_mem_gen_4
blk_mem_gen_6
blk_mem_gen_7
blk_mem_gen_9
blk_mem_gen_11
Music_TOP_ front
Music_TOP_ after
pmod_1_a
pmod_2_a
pmod_4_a
pmod_1_b
pmod_2_b
pmod_4_b
pixel
MUX
VGA_RGB
VGA display
vsync
hsync

6. 實作方式

首先，控制整個提款機內部運作的核心部分，我們以一個 finite state machine 來完成，總共有 13 個 state: insert_card (等待插卡)、keyin_pass (輸入密碼)、chose_item (選擇服務項目)、keyin_card (匯款時輸入欲匯入帳號)、decide_howmuch (選擇金額)、put_in(自行輸入金額)、money_not_enough (餘額不足)、trans_money (處理匯款)、keep_transc_not (詢問是否繼續其他交易)、keyin_wrong (密碼輸入錯誤)、keyin_wrong_2 (金額輸入錯誤)、count_down_money(吐錢後倒數 30 秒)、count_down_card(退卡後倒數 10 秒)，現在讓我一一來細說這些 state。

- Insert_card (等待插卡)

```
next_count_second1 <= count_second1;
next_count_second2 <= count_second2;
next_money_card1 <= money_card1;
next_money_card2 <= money_card2;
next_money_card3 <= money_card3;
next_money_card4 <= money_card4;
next_money_card5 <= money_card5;
```

這幾行是後面某幾個 state 會用到的變數，為了避免發生在跑其他 state 時因為沒有給值而被更改成任意值的狀況，所以我們在每一個 state 均放入這幾行讓他維持原來應有的值。

```
if(card_1 == 1) begin
    next_id_card1 <= 1;
    next_state <= keyin_pass;
end
else if(card_2 == 1) begin
    next_id_card1 <= 2;
    next_state <= keyin_pass;
end
```

這邊判斷由 switch 輸入的訊號(card_1、card_2、….、card_5)，來判斷使用者插入哪一張金融卡，並用 id_card1 來記錄現在插的是哪張金融卡以便後面的 state 做判斷。最後讓下個 state 到 keyin_pass，給使用者輸入密碼。

- keyin_pass(輸入密碼)

```
if(key_0 == 1) begin
    next_comp_num_1 <= comp_num_1;
    next_comp_num_2 <= comp_num_2;
    next_comp_num_3 <= comp_num_3;
    next_comp_num_4 <= comp_num_4;
    next_comp_num_5 <= comp_num_5;
    next_comp_num_6 <= comp_num_6;
    if(which_place == 0) begin
        next_comp_num_1 <= 0;
        next_which_place <= which_place + 1;
        next_state <= keyin_pass;
    end
    else if(which_place == 1) begin
        next_comp_num_2 <= 0;
        next_which_place <= which_place + 1;
        next_state <= keyin_pass;
    end

   else if(which_place == 2) begin
       next_comp_num_3 <= 0;
       next_which_place <= which_place + 1;
       next_state <= keyin_pass;
   end
   else if(which_place == 3) begin
       next_comp_num_4 <= 0;
       next_which_place <= which_place + 1;
       next_state <= keyin_pass;
   end
   else if(which_place == 4) begin
       next_comp_num_5 <= 0;
       next_which_place <= which_place + 1;
       next_state <= keyin_pass;
   end
   else if(which_place == 5) begin
       next_comp_num_6 <= 0;
       next_which_place <= which_place + 1;
       next_state <= keyin_pass;
   end
end
```

key_0 為鍵盤傳入的訊號 0，代表使用者輸入密碼 0，其他還有 key_1、key_2、…、key_9，來判斷使用者目前輸入甚麼密碼。comp_num_1、comp_num_2、…、comp_num_6 用來記錄使用

者輸入的六位數密碼，comp_num_1 記錄第一位數，comp_num_2 記錄第二位數，以此類推，例如 ： 使用者小花輸入 046981，則 comp_num_1 = 0、comp_num_2 = 4、comp_num_3 = 6、comp_num_4 = 9、comp_num_5 = 8、comp_num_6 = 1。

which_place 用來記錄使用者目前輸入的是第幾位密碼，一開始未輸入時 which_place 設為 0，在輸入密碼的第一個數字時，就會判斷到 which_place==0，進而把紀錄第一位數的密碼 comp_num_1 設為現在輸入的數字，並把 which_place + 1，再來輸入第二位數時就會判斷到 which_place == 1，進而把 comp_num_2 設為現在輸入的數字，依此類推即可正確紀錄使用者輸入的六位數密碼。

```
else if(key_correct == 1) begin
    next_which_place <= 0;
    next_state <= keyin_pass;
end
```

Key_correct (更正鍵)為鍵盤輸入信號，使用者不小心將密碼輸入錯誤時可按下 key_correct 清空密碼欄位重新輸入，這邊將 which_place 設為 0，如此一來就可再重新從密碼第一位數開始判斷使用者輸入到第幾位密碼並記錄下來。

```
else if(key_enter == 1 && which_place == 6 && id_card1 == 1) begin
    if(comp_num_1 == 9&& comp_num_2 == 4 && comp_num_3 == 2 && comp_num_4 == 7 && comp_num_5 == 6 && comp_num_6 ==0)begin
        next_which_place <= 0;
        next_state <= chose_item;
    end
    else begin
        next_which_place <= 0;
        next_state <= keyin_wrong;
    end
end
```

當使用者將密碼輸入好後可按下鍵盤上確認鍵 key_enter，我們就判斷如果使用者按下確認鍵(key_enter == 1)，而且他正確地輸入了六個密碼(which_place == 6)，而且使用者插入的卡片是卡 1 時，就判斷剛剛用 comp_num_1~comp_num_6 紀錄使用者輸入的密碼是否和我們預設給卡 1 的密碼相符，是的話就可以進入到 chose_item(選擇服務的 state)，如果不符合即代表使用者輸入的密碼錯誤，就會進到 keyin_wrong(密碼錯誤 state)，其他卡 2~卡 5 也是像這樣判斷。

```
else if(key_enter == 1 && which_place !=6)begin
        next_which_place <= 0;
        next_state <= keyin_wrong;
end
```

如果按下 key_enter 但是輸入的密碼小於六個時，也會被告知輸入錯誤。

- keyin_wrong (密碼輸入錯誤)

```
keyin_wrong: begin
    next_count_second1 <= count_second1;
    next_count_second2 <= count_second2;
    next_money_card1 <= money_card1;
    next_money_card2 <= money_card2;
    next_money_card3 <= money_card3;
    next_money_card4 <= money_card4;
    next_money_card5 <= money_card5;
    if(key_enter) begin
        next_state <= keyin_pass;
    end
    else begin
        next_state <= keyin_wrong;
    end
end
```

進到這邊告訴使用者密碼輸入錯誤後，需要使用者按一下 enter 再回到 keyin_pass state 重新輸入密碼。

- chose_item(選擇服務項目)

```
chose_item:begin
    next_count_second1 <= count_second1;
    next_count_second2 <= count_second2;
    next_money_card1 <= money_card1;
    next_money_card2 <= money_card2;
    next_money_card3 <= money_card3;
    next_money_card4 <= money_card4;
    next_money_card5 <= money_card5;
    if(key_a) begin
        next_state <= decide_howmuch;
        next_flag <=0;
    end
    else if(key_b) begin
        next_state <= keyin_card;
        next_flag <=1;
    end
    else if(key_c) begin
        next_state <= keep_transc_not;
        next_flag <=0;
    end
    else begin
        next_state <= chose_item;
        next_flag <=0;
    end
end
```

這邊以使用者按下的鍵來判斷他需要甚麼服務，key_a 為領錢，選擇領錢後，下一個 state 會進到 decide_howmuch 選擇金額；key_b 為匯款，下一個 state 會進入 keyin_card 讓使用者輸入欲匯入卡號；key_c 為查詢餘額，下一個會進入到 keep_transc_not 告訴使用者餘額並詢問使用者是否要繼續進行其他交易。

flag 用來記錄是否為匯款，以便後面的 state 判斷，若使用者選用匯款功能，則把 flag 設為 1，否則為 0。

- keyin_card(匯款時輸入欲匯入帳號)

```
if(key_1) begin
    next_id_card2 <= 1;
    next_state <= keyin_card;
end
else if(key_2) begin
    next_id_card2 <= 2;
    next_state <= keyin_card;
end
else if(key_3) begin
    next_id_card2 <= 3;
    next_state <= keyin_card;
end
else if(key_4) begin
    next_id_card2 <= 4;
    next_state <= keyin_card;
end
else if(key_5) begin
    next_id_card2 <= 5;
    next_state <= keyin_card;
end
```

鍵盤訊號 key_1~key_5 為判斷使用者目前輸入欲匯入的卡號，並用 id_card2 紀錄此卡號以便後面 state 判斷。

```
else if(key_enter && (id_card2 != 0)) begin
    next_state <= decide_howmuch;
end
```

使用者按下 key_enter 確認鍵時，我們以 id_card2 來判斷使用者是否有輸入了卡號，如果正確輸入的話，就可進入 decide_howmuch state 選擇金額；由於 id_card2 的初始值設為 0，若使用者未輸入卡號，id_card2 會依然維持 0，就不會進入下個 state。

```
else if(key_correct) begin
    next_id_card2 <= 0;
    next_state <= keyin_card;
end
```

在這邊也做了更正鍵 key_correct，若使用者發現卡號輸入錯誤時，在按下 enter 前都可以重新再輸入卡號。

- decide_howmuch (選擇金額)

  鍵盤訊號 key_a、key_b、key_c、key_d、key_e，判斷使用者選擇哪種金額，從 a~e 分別是 ： 自行輸入其他金額、領 1000、領 2000、領 5000、領 10000。

```
if(key_a) begin
    next_state <= put_in;
end
```

使用者按下 a 選擇自行輸入金額，即進入 put_in 輸入金額。

而其他的 b~e，這邊以 b 做說明：

```
else if(key_b) begin
        if(id_card1 == 1 && money_card1 >= 1000 && flag == 0) begin
            next_money_card1 <= money_card1 - 1000;
            next_state <= count_down_money;
            next_temp_money <= 1000;
        end
        else if(id_card1 == 2 && money_card2 >= 1000 && flag == 0) begin
            next_money_card2 <= money_card2 - 1000;
            next_state <= count_down_money;
            next_temp_money <= 1000;
        end
        else if(id_card1 == 3 && money_card3 >= 1000 && flag == 0) begin
            next_money_card3 <= money_card3 - 1000;
            next_state <= count_down_money;
            next_temp_money <= 1000;
        end
        else if(id_card1 == 4 && money_card4 >= 1000 && flag == 0) begin
            next_money_card4 <= money_card4 - 1000;
            next_state <= count_down_money;
            next_temp_money <= 1000;
        end
        else if(id_card1 == 5 && money_card5 >= 1000 && flag == 0) begin
            next_money_card5 <= money_card5 - 1000;
            next_state <= count_down_money;
            next_temp_money <= 1000;
        end
```

使用者按下 b 選擇金額 1000 元時，我們用之前紀錄現在插的是哪張卡片的 id_card1 來判斷要扣哪一張卡的錢，且這張卡裡的金額必須大於使用者欲領金額 1000，且 flag == 0(為領錢 非匯錢)，就將此卡內的錢扣款 1000 元(next_money_card <= money_card – 1000)，並將 temp_money 設為 1000 紀錄使用者領的錢(後面的 state 會用到)，然後進入下個 count_down_money state 吐錢給使用者。

```
else if(id_card1 == 1 && money_card1 >= 1000 && flag == 1) begin
    next_money_card1 <= money_card1 - 1000;
    next_state <= trans_money;
    next_temp_money <= 1000;
end
else if(id_card1 == 2 && money_card2 >= 1000 && flag == 1) begin
    next_money_card2 <= money_card2 - 1000;
    next_state <= trans_money;
    next_temp_money <= 1000;
end
else if(id_card1 == 3 && money_card3 >= 1000 && flag == 1) begin
    next_money_card3 <= money_card3 - 1000;
    next_state <= trans_money;
    next_temp_money <= 1000;
end
else if(id_card1 == 4 && money_card4 >= 1000 && flag == 1) begin
    next_money_card4 <= money_card4 - 1000;
    next_state <= trans_money;
    next_temp_money <= 1000;
end
else if(id_card1 == 5 && money_card5 >= 1000 && flag == 1) begin
    next_money_card5 <= money_card5 - 1000;
    next_state <= trans_money;
    next_temp_money <= 1000;
end
```

而如果其他判斷和上上圖一樣但是 flag == 1 時，代表現在是要匯款，所以把進入下個吐錢 state 改為進入 trans_money state 處理匯款。

- put_in(自行輸入金額)

```
if(key_0 && digit < 6) begin
    next_temp_money <= temp_money*10 + 0;
    next_digit <= digit +1;
    next_state <= put_in;
end
else if(key_1 && digit < 6) begin
    next_temp_money <= temp_money*10 + 1;
    next_digit <= digit +1;
    next_state <= put_in;
end
```

```
else if(key_2 && digit < 6) begin
    next_temp_money <= temp_money*10 + 2
    next_digit <= digit +1;
    next_state <= put_in;
end
else if(key_3 && digit < 6) begin
    next_temp_money <= temp_money*10 + 3
    next_digit <= digit +1;
    next_state <= put_in;
end
```

digit 為現在輸入到第幾位數金額，因為最多不能提領超過一萬

元，所以我們判斷 digit<6，而 key_0~9 判斷使用者現在輸入的數字(上圖貼出 key_0~3 示意)。

在使用者輸入數字時，將記錄錢的 temp_money*10 + 現在輸入的數字，如此就能正確將輸入的金錢正確存入 temp_money 內，並且將 digit 加 1 紀錄現在輸入了幾位數。

```
else if(key_enter == 1 && temp_money <= 10000 && temp_money%1000 == 0 && temp_money != 0 && flag == 0) begin
        next_money_card1 <= money_card1;
        next_money_card2 <= money_card2;
        next_money_card3 <= money_card3;
        next_money_card4 <= money_card4;
        next_money_card5 <= money_card5;
        if(id_card1 == 1 && money_card1 >= temp_money) begin
            next_money_card1 <= money_card1 - temp_money;
            next_state <= count_down_money;
        end
        else if(id_card1 == 2 && money_card2 >= temp_money) begin
            next_money_card2 <= money_card2 - temp_money;
            next_state <= count_down_money;
        end
        else if(id_card1 == 3 && money_card3 >= temp_money) begin
            next_money_card3 <= money_card3 - temp_money;
            next_state <= count_down_money;
        end
        else if(id_card1 == 4 && money_card4 >= temp_money) begin
            next_money_card4 <= money_card4 - temp_money;
            next_state <= count_down_money;
        end
        else if(id_card1 == 5 && money_card5 >= temp_money) begin
            next_money_card5 <= money_card5 - temp_money;
            next_state <= count_down_money;
        end
        else begin
            next_state <= money_not_enough;
        end
end
```

使用者輸入完欲提領金錢後，按下 key_enter 確認鍵後， 會判

斷 temp_money <= 10000 (提領金額不能超過一萬) && temp_money% 1000 == 0 (不能提領金錢單位小於一千的金額) && temp_money!= 0 (輸入金額為 0 或沒有輸入金額即為輸入錯誤金額) && flag == 0 (領錢)，必須滿足以上這些條件才會開始判斷目前是插入哪張卡片且卡片餘額需大於提領金額，才會將卡片內的金額扣掉欲提領金額 temp_money，並進入 count_down_money state 吐錢給使用者，否則餘額不足會進入 money_not_enough state。

```
else if(key_enter == 1 && temp_money%1000 == 0 && temp_money <= 10000 && temp_money != 0 && flag == 1)
        next_money_card1 <= money_card1;
        next_money_card2 <= money_card2;
        next_money_card3 <= money_card3;
        next_money_card4 <= money_card4;
        next_money_card5 <= money_card5;
        if(id_card1 == 1 && money_card1 >= temp_money) begin
            next_money_card1 <= money_card1 - temp_money;
            next_state <= trans_money;
        end
        else if(id_card1 == 2 && money_card2 >= temp_money) begin
            next_money_card2 <= money_card2 - temp_money;
            next_state <= trans_money;
        end
        else if(id_card1 == 3 && money_card3 >= temp_money) begin
            next_money_card3 <= money_card3 - temp_money;
            next_state <= trans_money;
        end
        else if(id_card1 == 4 && money_card4 >= temp_money) begin
            next_money_card4 <= money_card4 - temp_money;
            next_state <= trans_money;
        end
        else if(id_card1 == 5 && money_card5 >= temp_money) begin
            next_money_card5 <= money_card5 - temp_money;
            next_state <= trans_money;
        end
        else begin
            next_state <= money_not_enough;
        end
end
```

而這邊判斷與上上圖相同，除了 flag 改成判斷為 1 (匯款)，其他處理方式相同，除了下一個會進入 trans_money state 處理匯款。

```
else if(key_enter == 1 && (temp_money > 10000 || temp_money == 0 || temp_money%1000 != 0)) begin
    next_temp_money <= 0;
    next_digit <= 0;
    next_state <= keyin_wrong_2;
end
```

這邊是判斷若使用者按下確認鍵，但是輸入金額超過一萬元，或是輸入金額等於 0，或是輸入小於一千單位金錢的金額時，將本來儲存的輸入金額 temp_money 和紀錄輸入位數的 digit 重設為 0，並進入 keyin_wrong_2 state 告訴使用者金額輸入錯誤。

```
else if(key_correct) begin
    next_temp_money <= 0;
    next_digit <= 0;
    next_state <= put_in;
end
else if(key_back) begin
    next_temp_money <= 0;
    next_digit <= 0;
    next_state <= decide_howmuch;
end
```

這邊除了做 key_correct 更正鍵，讓使用者可在不小心輸入錯誤時清空原本輸入的金額再重新輸入；還做了 key_back 返回鍵，如果使用者突然不需要自行輸入金額時可返回上一個 decide_howmuch state 重新選擇金額。

- keyin_wrong_2 (金額輸入錯誤)

```
keyin_wrong_2:begin
    next_temp_money <= 0;
    next_digit <= 0;
    next_flag <= flag;
    next_count_second1 <= count_second1;
    next_count_second2 <= count_second2;
    next_money_card1 <= money_card1;
    next_money_card2 <= money_card2;
    next_money_card3 <= money_card3;
    next_money_card4 <= money_card4;
    next_money_card5 <= money_card5;
    if(key_enter) begin
        next_state <= put_in;
    end
    else begin
        next_state <= keyin_wrong_2;
    end
end
```

進入 keyin_wrong_2 後，只要按下 enter 即可回到 put_in state 重新輸入金額。

- money_not_enough (餘額不足)

```
money_not_enough:begin
    next_temp_money <= 0;
    next_digit <= 0;
    next_flag <= flag;
    next_count_second1 <= count_second1;
    next_count_second2 <= count_second2;
    next_money_card1 <= money_card1;
    next_money_card2 <= money_card2;
    next_money_card3 <= money_card3;
    next_money_card4 <= money_card4;
    next_money_card5 <= money_card5;
    if(key_enter) begin
        next_state <= keep_transc_not;
    end
    else begin
        next_state <= money_not_enough;
    end
end
```

使用者進到 money_not_enough state 時，須按下 enter，才會

繼續再到 keep_transc_not state 顯示餘額並詢問是否繼續進行其他交易。

- count_down_money(吐錢後倒數 30 秒)

```
if(take_money == 1) begin
    next_count_second1 <= 0;
    next_state <= keep_transc_not;
end
```

count_secon1 為秒數 30 秒，在機器吐錢後開始倒數 30 秒，使用者須在這 30 秒內把錢取走，否則機器將回收金錢。

Switch 訊號 take_money 表示使用者將提款機吐出的錢取走，這邊是判斷如果使用者將錢取走了，就把 count_second1 重設為 0，且進入 keep_transc_not state 詢問是否繼續進行其他交易。

```
else begin
    next_money_card1 <= money_card1;
    next_money_card2 <= money_card2;
    next_money_card3 <= money_card3;
    next_money_card4 <= money_card4;
    next_money_card5 <= money_card5;
    if(count_second1 > 0) begin
        next_count_second1 <= count_second1 - 1;
        next_state <= count_down_money;
    end
    else if(count_second1 == 0 && id_card1 == 1 ) begin
        next_count_second1 <= 0;
        next_money_card1 <= money_card1 + temp_money;
        next_state <= keep_transc_not;
    end
    else if(count_second1 == 0 && id_card1 == 2 ) begin
        next_count_second1 <= 0;
        next_money_card2 <= money_card2 + temp_money;
        next_state <= keep_transc_not;
    end
```

```
    else if(count_second1 == 0 && id_card1 == 3 ) begin
        next_count_second1 <= 0;
        next_money_card3 <= money_card3 + temp_money;
        next_state <= keep_transc_not;
    end
    else if(count_second1 == 0 && id_card1 == 4 ) begin
        next_count_second1 <= 0;
        next_money_card4 <= money_card4 + temp_money;
        next_state <= keep_transc_not;
    end
    else if(count_second1 == 0 && id_card1 == 5 ) begin
        next_count_second1 <= 0;
        next_money_card5 <= money_card5 + temp_money;
        next_state <= keep_transc_not;
    end
    else begin
        next_count_second1 <= count_second1;
        next_state <= count_down_money;
    end
end
```

而在使用者把錢取走前，如果 count_second1 還沒倒數完 30 秒(即 count_second1 > 0)，count_second1 會持續減 1，直到數到 0 時，使用者卻還沒把錢領走，此時我們須做回收鈔票的動作。判斷到 count_second1 == 0 時，我們要再判斷現在插的是哪張卡片，因為使用者未將金錢取走，所以我們不必對她的帳戶扣款，於是我們必須將之前紀錄領錢且扣過款的 temp_money 加回此帳戶中，以 id_card1 == 1(目前插入卡 1)為例：將 next_money_card1 設成 money_card1+temp_money，如此即機器將金錢回收並不會扣到款項，並進入 keep_transc_not state 詢問是否繼續進行其他交易。

- trans_money (處理匯款)

```
if(id_card2 == 1) begin
    next_money_card1 <= money_card1 + temp_money;
    next_state <= keep_transc_not;
end
else if(id_card2 == 2) begin
    next_money_card2 <= money_card2 + temp_money;
    next_state <= keep_transc_not;
end
else if(id_card2 == 3) begin
    next_money_card3 <= money_card3 + temp_money;
    next_state <= keep_transc_not;
end
else if(id_card2 == 4) begin
    next_money_card4 <= money_card4 + temp_money;
    next_state <= keep_transc_not;
end
else if(id_card2 == 5) begin
    next_money_card5 <= money_card5 + temp_money;
    next_state <= keep_transc_not;
end
```

這邊以之前紀錄的 id_card2 來判斷匯款要匯給哪一張卡片，再把那張卡片的餘額加上欲匯入金額成為新的餘額。

- keep_transc_not (詢問是否繼續其他交易)

```
if(key_yes) begin
    next_temp_money <= 0;
    next_which_place <= 0;
    next_digit <= 0;
    next_count_second1 <= 30;
    next_count_second2 <= 10;
    next_flag <= 0;
    next_state <= chose_item;
end
else if(key_no) begin
    next_temp_money <= 0;
    next_which_place <= 0;
    next_digit <= 0;
    next_count_second1 <= 30;
    next_count_second2 <= 10;
    next_flag <= 0;
    next_state <= count_down_card;
end
```

如果使用者欲進行其他交易，就按下鍵盤上的 O(key_yes)，即

可再進入 chose_item state 選擇服務；如果決定結束服務，則按下鍵盤上的 X(key_yes)，則進入 count_down_card state 退出卡片。

而在此 state 我們也將一些變數初始化，以免發生錯誤。

- count_down_card(退卡後倒數 10 秒)

```
if(card_1 ==0 && card_2 ==0 && card_3 ==0 && card_4 ==0 && card_5 ==0) begin
    next_count_second2 <= 0;
    next_state <= insert_card;
end
```

Count_second2 預設為 10 秒，使用者須在 10 秒內將提款機退出的卡取回，否則將被機器收回。

這邊用 switch 訊號 card_1~5 來判斷使用者是否已把卡片取回，若取回(card_1~5 = 0)，則回到 insert_card state 等待插卡。

```
else begin
    if(count_second2 > 0) begin
        next_count_second2 <= count_second2 - 1;
        next_state <= count_down_card;
    end
    else begin
        next_count_second2 <= 0;
        next_state <= insert_card;
    end
end
```

而這邊為使用者還沒取卡的部分；判斷是否倒數完十秒，若還沒則 count_secont_2 持續減 1，若倒數完十秒，則提款機將卡回收，並回到 insert_card state 等待插卡。

Clock 的部分我們在倒數秒數時(判斷現在是否在 count_down_card 或 count_down_money state)，選用一秒的 clock，其他則均為 1/16clock：

```
assign clk_select = (state == count_down_card  || state == count_down_money) ? clk_1sec: clk16;
```

再來，在板子上的 7-segment 我們顯示了倒數秒數和餘額，我們只用到板子最左邊兩位數；倒數 30 秒和 10 秒即直接在板子顯示 30 和 10，而餘額部分，我們設左邊的數為表示萬元，右邊表示千元，例如下圖表示的即為四萬元：

我們用 BCD1 顯示左邊的萬位數字和 BCD0 顯示右邊的千位數字 ：

```
assign BCD0 = (state == count_down_money)? count_second1 % 10 :
              (state == count_down_card)? count_second2 % 10 :
              (state == keep_transc_not && card_1 == 1)? (money_card1 % 10000)/1000 :
              (state == keep_transc_not && card_2 == 1)? (money_card2 % 10000)/1000 :
              (state == keep_transc_not && card_3 == 1)? (money_card3 % 10000)/1000 :
              (state == keep_transc_not && card_4 == 1)? (money_card4 % 10000)/1000 :
              (state == keep_transc_not && card_5 == 1)? (money_card5 % 10000)/1000 :0;

assign BCD1 = (state == count_down_money)? count_second1 / 10 :
              (state == count_down_card)? count_second2 / 10 :
              (state == keep_transc_not && card_1 == 1)? money_card1 / 10000 :
              (state == keep_transc_not && card_2 == 1)? money_card2 / 10000 :
              (state == keep_transc_not && card_3 == 1)? money_card3 / 10000 :
              (state == keep_transc_not && card_4 == 1)? money_card4 / 10000 :
              (state == keep_transc_not && card_5 == 1)? money_card5 / 10000 :0;
```

在這邊判斷如果 state 在需要倒數秒數的 count_down_money 或

count_down_card state 時，就顯示出秒數；或是在需顯示餘額的 keep_transc_not state 時判斷目前插入的卡號，並顯示此張卡的餘額。

而板子上的最右邊的燈（warning 燈），為判斷是否有卡片是插入的狀態，是則亮燈，否則不亮：

```
assign warning = (card_1 == 0 && card_2 == 0 && card_3 == 0 && card_4 == 0 && card_5 == 0) ? 0 : 1;
```

從右邊 warning 燈往左邊開始數的 9 個表示現在在哪個 state 的 LED 燈，我們用 state 去判斷現在應該亮哪個對應的燈：

```
assign state_keyin_pass = (state == keyin_pass) ? 1:0;
assign state_chose_item = (state == chose_item) ? 1:0;
assign state_keyin_card = (state == keyin_card)?1:0;
assign state_decide_howmuch = (state == decide_howmuch)?1:0;
assign state_put_in = (state == put_in)?1:0;
assign state_trans_money = (state == trans_money) ? 1:0;
assign state_keep_transc_not = (state == keep_transc_not) ? 1:0;
assign state_money_not_enough = (state == money_not_enough) ? 1:0;
assign state_keyin_wrong = (state == keyin_wrong || state == keyin_wrong_2) ? 1:0;
```

而剩下最左邊的六個 LED 燈為輸入密碼時隨著密碼亮燈，以 which_place(紀錄輸入到第幾位密碼)來判斷現在應亮幾個燈：

```
assign led_password_1 = (which_place >= 1) ? 1:0;
assign led_password_2 = (which_place >= 2) ? 1:0;
assign led_password_3 = (which_place >= 3) ? 1:0;
assign led_password_4 = (which_place >= 4) ? 1:0;
assign led_password_5 = (which_place >= 5) ? 1:0;
assign led_password_6 = (which_place >= 6) ? 1:0;
```

再來是音樂的部分，我們在插卡時和退卡時各做了一小段音效

```
assign music_front_on = (state == insert_card) ? 0 : 1;
assign music_after_on = (state == count_down_card)? 0 : 1;
```

music_front_on 代表插卡時的音效是否要開啟，開啟(state = insert_card

時)則將 music_front_on 設為 0，否則為 1；而 music_after_on 代表的是退卡時的音效，判斷方式與插卡音效相同。

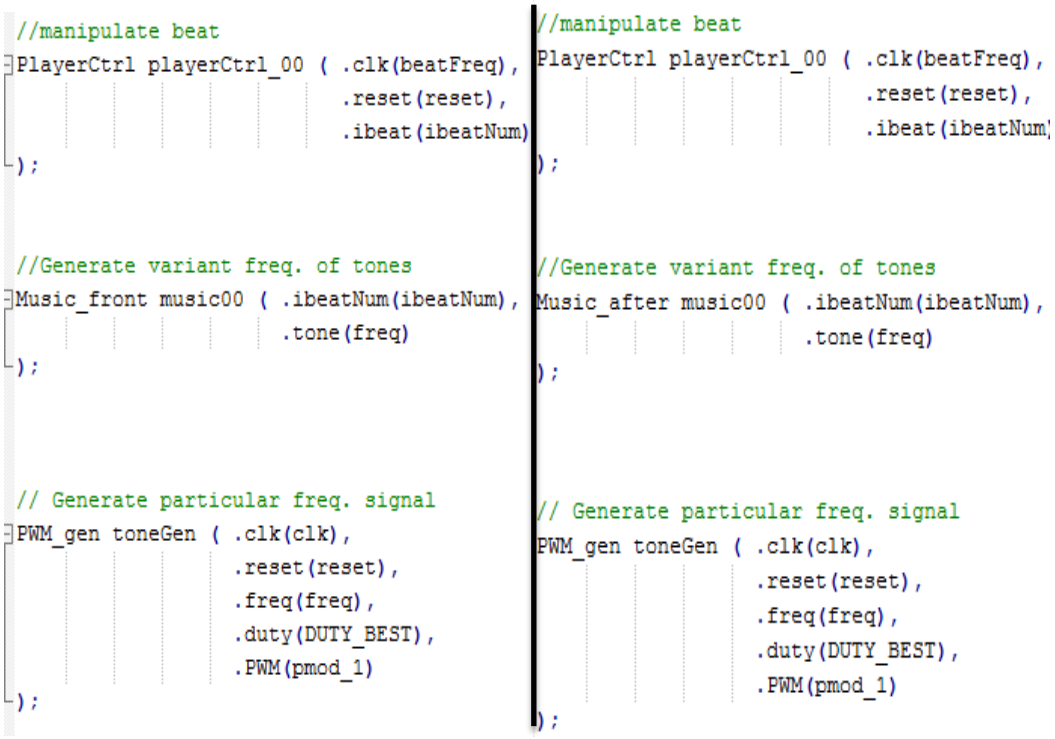
再用此兩個變數來判斷 pmod_1、pmod_2、pmod_4 的值，再 pmod_1 的部分判斷如果現在是撥放插卡音效(music_front_on == 0)，則將 pmod_1 設為 pmod_1_a (由插卡音效產生)，若不是則放退卡音樂，將 pmod_1 設為 pmod_1_b(由退卡音效產生)；pmod_2 也是用同樣的方法，而控制靜音與否的 pmod_4 則判斷現在是否有再撥放這兩個音樂的其中一個音樂，若無則讓音樂靜音：

```
assign pmod_1 = (music_front_on == 0) ? pmod_1_a : pmod_1_b;  // music
assign pmod_2 = (music_front_on == 0) ? pmod_2_a : pmod_2_b;
assign pmod_4 = ( music_front_on == 0 || music_after_on == 0) ? 1 : 0;
```

那我們的 pmod_1_a、pmod_1_b、pmod_2_a、pmod_2_b 又是如何產生的呢？請看以下說明

```
module Music_TOP_front (
    input clk,
    input reset,
    output pmod_1,
    output pmod_2,
    output pmod_4
);
parameter BEAT_FREQ = 32'd8;    //one beat=0.125sec
parameter DUTY_BEST = 10'd512;  //duty cycle=50%

wire [31:0] freq;
wire [7:0] ibeatNum;
wire beatFreq;

assign pmod_2 = 1'd1;   //no gain(6dB)
assign pmod_4 = (reset == 1)? 1'd0 : 1'd1;  //turn-

//Generate beat speed
PWM_gen btSpeedGen ( .clk(clk),
                     .reset(reset),
                     .freq(BEAT_FREQ),
                     .duty(DUTY_BEST),
                     .PWM(beatFreq)
);
```

```
module Music_TOP_after (
    input clk,
    input reset,
    output pmod_1,
    output pmod_2,
    output pmod_4
);
parameter BEAT_FREQ = 32'd8;    //one beat=0.125sec
parameter DUTY_BEST = 10'd512;  //duty cycle=50%

wire [31:0] freq;
wire [7:0] ibeatNum;
wire beatFreq;

assign pmod_2 = 1'd1;   //no gain(6dB)
assign pmod_4 = (reset == 1)? 1'd0 : 1'd1;  //turn-on

//Generate beat speed
PWM_gen btSpeedGen ( .clk(clk),
                     .reset(reset),
                     .freq(BEAT_FREQ),
                     .duty(DUTY_BEST),
                     .PWM(beatFreq)
);
```



```
//manipulate beat
PlayerCtrl playerCtrl_00 ( .clk(beatFreq),
                           .reset(reset),
                           .ibeat(ibeatNum)
);

//Generate variant freq. of tones
Music_front music00 ( .ibeatNum(ibeatNum),
                      .tone(freq)
);

// Generate particular freq. signal
PWM_gen toneGen ( .clk(clk),
                  .reset(reset),
                  .freq(freq),
                  .duty(DUTY_BEST),
                  .PWM(pmod_1)
);
```

```
//manipulate beat
PlayerCtrl playerCtrl_00 ( .clk(beatFreq),
                           .reset(reset),
                           .ibeat(ibeatNum)
);

//Generate variant freq. of tones
Music_after music00 ( .ibeatNum(ibeatNum),
                      .tone(freq)
);

// Generate particular freq. signal
PWM_gen toneGen ( .clk(clk),
                  .reset(reset),
                  .freq(freq),
                  .duty(DUTY_BEST),
                  .PWM(pmod_1)
);
```

左邊是產生 pmod_1_a、pmod_2_a 的 module，即為撥出插卡音效的 module；而右邊為產生 pmod_1_b、pmod_2_b 的 module，即為撥出退卡音效的 module，我們分別為插卡和退卡用了兩個這種 module，而這兩個 module 內唯一不同的內容是 Music_front music00 和 Music_after music00 兩個 module，裡面分別裝了插卡和退卡音效的音樂，如此就能在適當時機撥出我們希望撥出的音樂。

鍵盤的部分因為是拿之前做過的 lab 來加，沒有改變甚麼特別的部分，所以不多做說明。

最後是螢幕部分，我們總共放入四種不同尺寸的圖片，總共九張

圖，同一種尺寸但不同圖片我們用同一種 mem_addr_gen module 來產生 pixel_addr，再傳給不同的 blk_mem_gen module 進而產生不同的 pixel，如下圖，左邊為產生同種尺寸的一張 pixel_e，而右邊為產生兩張同尺寸的 pixel_g、pixel_h：

```
mem_addr_gen_e mem_addr_gen_inst_e(
    .clk(clk22),
    .h_cnt(h_cnt),
    .v_cnt(v_cnt),
    .pixel_addr(pixel_addr_e)
    );

blk_mem_gen_4 blk_mem_gen_0_inst_4(
      .clka(clk_25MHz),
      .wea(0),
      .addra(pixel_addr_e),
      .dina(data[11:0]),
      .douta(pixel_e)
    );
```

```
mem_addr_gen_g mem_addr_gen_inst_g(
    .clk(clk22),
    .h_cnt(h_cnt),
    .v_cnt(v_cnt),
    .pixel_addr(pixel_addr_g)
    );

blk_mem_gen_6 blk_mem_gen_0_inst_6(
      .clka(clk_25MHz),
      .wea(0),
      .addra(pixel_addr_g),
      .dina(data[11:0]),
      .douta(pixel_g)
    );
blk_mem_gen_7 blk_mem_gen_0_inst_7(
      .clka(clk_25MHz),
      .wea(0),
      .addra(pixel_addr_g),
      .dina(data[11:0]),
      .douta(pixel_h)
    );
```

再來用現在在哪個 state 來判斷螢幕需要輸出哪張圖片，比較特別的是輸入密碼時螢幕會隨著密碼輸入產生*字號的效果，也是像 LED 表示密碼的部分一樣判斷現在 which_place(現在輸入第幾位)是多少，然後用 h_cnt 和 v_cnt 去調整圖片位置，產生密碼圖片部分為下圖紅色框框，其他為產生其他圖片的判斷：

```
assign {vgaRed, vgaGreen, vgaBlue} =  (valid==1'b1 && state == 4'b0000 && h_cnt >=220 && h_cnt<=420 && v_cnt>= 60 && v_cnt<= 150) ? pixel:  // insert_card
                                      (valid==1'b1 && state == 4'b0000 && h_cnt >=280 && h_cnt<=360 && v_cnt>= 250&& v_cnt<= 330) ? pixel_b:  // pooral
                                      (valid==1'b1 && state == 4'b0001 && h_cnt >=220 && h_cnt<=420 && v_cnt>= 60&& v_cnt<= 150) ? pixel_d:  // keyin_pass

                                      (valid==1'b1 && state == 4'b0001 && which_place ==1 && h_cnt >=80 && h_cnt<=160 && v_cnt>= 250&& v_cnt<= 330) ? pixel_e:  // password
                                      (valid==1'b1 && state == 4'b0001 && which_place ==2 && h_cnt >=80 && h_cnt<=240 && v_cnt>= 250&& v_cnt<= 330) ? pixel_e:
                                      (valid==1'b1 && state == 4'b0001 && which_place ==3 && h_cnt >=80 && h_cnt<=320 && v_cnt>= 250&& v_cnt<= 330) ? pixel_e:
                                      (valid==1'b1 && state == 4'b0001 && which_place ==4 && h_cnt >=80 && h_cnt<=400 && v_cnt>= 250&& v_cnt<= 330) ? pixel_e:
                                      (valid==1'b1 && state == 4'b0001 && which_place ==5 && h_cnt >=80 && h_cnt<=480 && v_cnt>= 250&& v_cnt<= 330) ? pixel_e:
                                      (valid==1'b1 && state == 4'b0001 && which_place ==6 && h_cnt >=80 && h_cnt<=560 && v_cnt>= 250&& v_cnt<= 330) ? pixel_e:

                                      (state == 4'b1101 && h_cnt >= 240 && h_cnt <= 400  &&  v_cnt>= 80 && v_cnt<= 100 ) ? 12'hff0:
                                      (state == 4'b1101 && h_cnt >= 380 && h_cnt <= 400  &&  v_cnt>= 100 && v_cnt<= 160 ) ? 12'hff0:
                                      (state == 4'b1101 && h_cnt >= 240 && h_cnt <= 260  &&  v_cnt>= 100 && v_cnt<= 240 ) ? 12'hff0:
                                      (state == 4'b1101 && h_cnt >= 260 && h_cnt <= 400  &&  v_cnt>= 220 && v_cnt<= 240 ) ? 12'hff0:
                                      (state == 4'b1101 && h_cnt >= 380 && h_cnt <= 400  &&  v_cnt>= 240 && v_cnt<= 400 ) ? 12'hff0:
                                      (state == 4'b1101 && h_cnt >= 240 && h_cnt <= 380  &&  v_cnt>= 380 && v_cnt<= 400 ) ? 12'hff0:
                                      (state == 4'b1101 && h_cnt >= 240 && h_cnt <= 260  &&  v_cnt>= 320 && v_cnt<= 380 ) ? 12'hff0:
                                      (state == 4'b1101 && h_cnt >= 280 && h_cnt <= 300  &&  v_cnt>= 20 && v_cnt<= 460 ) ? 12'hff0:
                                      (state == 4'b1101 && h_cnt >= 340 && h_cnt <= 360  &&  v_cnt>= 20 && v_cnt<= 460 ) ? 12'hff0:

                                      (valid==1'b1 && state == 4'b0010 && h_cnt >=215 && h_cnt<=425 && v_cnt>= 250 && v_cnt<= 290) ? pixel_g: // chose_abc

                                      (valid==1'b1 && state == 4'b0100 && h_cnt >=215 && h_cnt<=425 && v_cnt>= 250 && v_cnt<= 290) ? pixel_h: // chose_money a~f

                                      (valid==1'b1 && state == 4'b0011 && h_cnt >=220 && h_cnt<=420 && v_cnt>= 60 && v_cnt<= 150) ? pixel_j:  // keyin_card

                                      (valid==1'b1 && state == 4'b1001 && h_cnt >=220 && h_cnt<=420 && v_cnt>= 60 && v_cnt<= 150) ? pixel_m: // keep_not

                                      (valid==1'b1 && state == 4'b1110 && h_cnt >=220 && h_cnt<=420 && v_cnt>= 60&& v_cnt<= 150) ? pixel_c: // thanks
                                      (valid==1'b1 && state == 4'b1110 && h_cnt >=280 && h_cnt<=360 && v_cnt>= 250&& v_cnt<= 330) ? pixel_b: 0; // pooral
```

7.心得

- 楊翔閔：

  此次的 final project 實作過程實在是一段勞心勞力的過程，更讓我了解到在 coding 的過程細心與良好的 coding style 真的很重要。

  此次實作上遇到的最大的困難主要有二，第一個，也是此次困擾我們時間最長的問題，就是在我們設計的 FSM 中，有許多需要在操作過程紀錄的值，我們都因為沒有釐清設置 next_XXX 這個變數的時機，結果我們在某些不需要設 next_XXX 變數設了，該使用的卻沒有設，導致我們的變數隨著 clk 變動而變成 unknow。另一個問題則是關於 VGA 的問題，我們的 VGA 因為比較晚才開始實作，加上我們當時懶得花時間慢慢刻 pixel 檔，因此我們幾乎所有的圖片都是先畫好再燒入 FPGA 板，後來覺得這樣其實比慢慢刻更花時間…，需要不停地調座標，而且我們發現用燒的圖片的品質顯然跟用 code 慢慢刻出來的品質差異有點大，但已經沒甚麼時間再改掉重做，我想這也算學到了一次經驗吧!

  總的來說，這次的 final project 實作，雖然很累，但確實的讓我透徹的再把這學期沒有學通的，不論是語法的錯誤也好，設計

過程中思考上的錯誤也好，或者是在過去 lab 中沒有透徹了解的部分也好，都好好複習了一遍。也要謝謝教授，助教們一學期以來的辛勞!!!

- 林宛萱:

  之前雖然有修過數位邏輯設計寫過一點 verilog，但那時寫好 code 都沒有燒進板子過，也對 verilog 似懂非懂的，所以這次要寫硬體實驗的 Final Project 之前，內心深感惶恐，想說：我真的能做到嗎？！

  回想起一開始剛和隊友面對著一頁全空白的 notepad++時，絞盡腦汁地想，現在要從哪裡開始著手呢？如果寫好卻有一大堆 de 不掉的 bug 怎麼辦哩？

  幸好經我我們縝密的討論後，我們先將一些需要用到的小 module 像 clk_divisor、debounce…等，從 ilms 拿下來，再開始寫我們的主架構 finite state machine。從一開始只加入插卡和退卡兩個 state 和兩個 state 的音效，一直到後來慢慢地把所有 state 完整加入，再 de 了大概四五天這 finite state machine 的大大小小的 bug，最後才開始加入螢幕的部分；螢幕也是從一開始不小心加入太多圖讓記憶體爆掉，再慢慢地裁減圖片、修改大小、調整位置，有的圖片改成用刻的不占用 memory，到最後終於

完成了我們的大作！！

在 debug 的途中，我們真心得出了一個結論，有變數是放在等號右邊要改變值的那種變數，只要在每個 state 加入他的 next 就對了(ex. next_temp_money <= temp_money)，就不用怕發生被更改為任意值或發生 latch 的問題，所以能看到我們的 finite state machine 裡面有很多的 state 都在重複放入好幾行這種東西 XD

做完之後這次 Final 後，真的得到了不少的成就感，花了近兩個禮拜的時間完成，也學到了不少東西，對 verilog 似乎是又多了解了不少，在團隊合作的過程中也互相交換了不少想法，在這些想法的激盪中，產生了我們的「菁英操作提款機」！

8. 分工項目以及對 project 的貢獻百分比

- 所有 code 部分均為一起討論一起完成
- report :第一、二、三、四、五、七、八部分:楊翔閔完成

  第六、七部分:林宛萱完成
- 貢獻度:楊翔閔 50%　林宛萱 50%