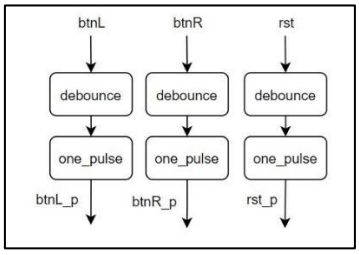
Lab 5

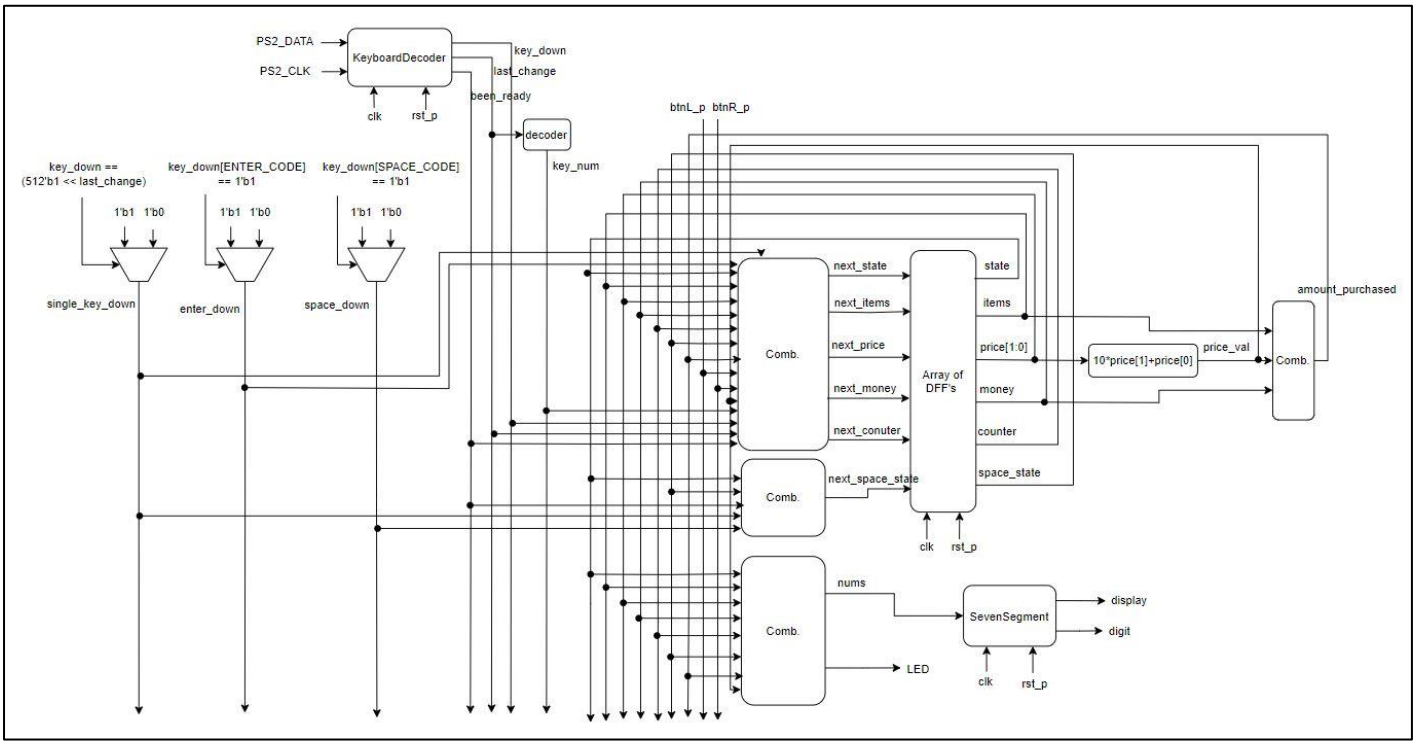
學號: 110011138 姓名: 楊立慈

A. Lab Implementation

下圖為 lab5 中所有 button 的 input 經過 debounce、one_pulse 處理的 block diagram，其中省略的所有 module 所用的 clock source 都是 clk。



下圖為 lab5 主要的 block diagram，其中所有的 decoder、Comb. Circuit 實作會與 code 一起解釋。圖中 state 訊號代表 FSM 目前的 state、items 訊號代表剩下的商品數量、price 代表商品價格、money 代表使用者付的錢、counter 是用來數秒數的、space_state 是用來在 SET state 切換輸入項目所用、amount_purchased 代表使用者目前可以買的商品數量。左側的 3 個 MUX 負責根據不同條件(如圖中所示)輸出有關鍵盤狀況的資訊，single_key_down 代表目前是否只有 1 個按鍵按下(利用 key_down 是否只包含一個 1 的 bit 判斷)、enter_down 代表 ENTER 鍵是否按下(看 key_down 裡 ENTER_CODE 對應的值是否是 1)、space_down 代表 SPACE 鍵是否按下(看 key_down 裡 SPACE_CODE 對應的值是否是 1)。此外，需要注意的是右側的 price[1:0]是一個 1D array，price[1]代表 price 的十位數、price[0]代表個位數，也因此 price_val，也就是 price 的值，只要單純利用 10*price[1]+price[0]的 Comb. Circuit 就可算出。此外，display、digit 等訊號我是利用 Comb. Circuit 算出 nums 後再利用提供的 SevenSegment module 來輸出，而 LED 訊號則是直接用 Comb. Circuit 算出即可。



下圖是 block diagram 中負責計算 key_num 的 decoder 的 code。主要是根據鍵盤數字鍵的 KEY_CODE 來把 last_change 對應的按鍵轉為數字值，若不是數字鍵則設為 NAN。

```
always @ (*) begin
  case (last_change)
    KEY_CODES[00] : key_num = 4'b0000; // 0
    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; // 9
    KEY_CODES[10] : key_num = 4'b0000; // 0
    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; // 9
    default       : key_num = NAN; // not a number
  endcase
end
```

下圖為 block diagram 中計算 next_space_state 的 Comb. Circuit，概念是在 SET state 裡 SPACE 還沒被按時 space_state 是 0，一旦 SPACE 被按就會 toggle space_state 的值。這樣之後在 SET state 輸入時就可以利用當 space_state==0 時是改變商品數量(items)，當 space_state==1 時是改變商品價格(price)。值得注意的是判斷 SPACE 按下的條件，要確認鍵盤按鍵訊號確實有被處理(been_ready)、只有單一按鍵按下(single_key_down)、SPACE 鍵被按下(space_down)。

```
always @(*) begin
  if (state == SET) begin
      next_space_state = space_state;
      if (been_ready && single_key_down && space_down) next_space_state = ~space_state;
  end
  else next_space_state = 0;
end
```

下圖為 block diagram 中計算可買商品數(amount_purchased)的 Comb. Circuit 實作。首先先處理邊界條件以避免除以 0 的情況出現，當沒商品可買時(items==0)可買數量設為 0，當商品價格是 0 時(price_val==0)可買數量即為商品數量，其餘情況可買數量則是$\min\{items, \left\lfloor \frac{money}{price_val} \right\rfloor\}$。

```
always @(*) begin
  if (items == 0) amount_purchased = 0;
  else if (price_val == 0) amount_purchased = items;
  else amount_purchased = (items < (money / price_val)) ? items : (money / price_val);
end
```

下圖為 block diagram 中 LED、nums 的 Comb. Circuit 實作，由於 code 較長所以這裡以 SET、PAYMENT、BUY、CHANGE states 為例。在 SET state 如同上述，利用 space_state 來決定 LED 要亮左半邊還是右半邊，nums 則單純利用 concatenate 4 個 4-bit 的訊號來決定我們要 display 的值，像是商品數量 items、價錢十位數 price[1]、個位數 price[0]等。在 PAYMENT state 也是類似，不過 nums 的右邊兩位變成展示 money 的十位、個位數。在 BUY state 則是根據不同 round 值來做出閃爍的效果，其中 round 值是利用 counter / 29'd50_000_000 計算的，因此 round==0 or 2 or 4 即為[0, 0.5)、[1, 1.5)、[2, 2.5)秒的時間區間，而此處 nums 可利用 amount_purchased 來展示可買數量、計算需要支付價格(即 price_val * amount_purchased)的十位、個位數。在 CHANGE state，nums 右邊兩位則變成要找的錢(即 money – price_val * amount_purchased)的十位、個位數字。

```
SET: begin
  nums = {items, D_DASH, price[1], price[0]};
  if (space_state == 1'b0) LED = 16'b1111_1111_0000_0000;
  else LED = 16'b0000_0000_1111_1111;
end
PAYMENT: begin
  LED = 16'd0;
  nums[15:8] = {D_DASH, D_DASH};
  nums[7:4] = money / 10;
  nums[3:0] = money % 10;
end
```

```
BUY: begin
  if (round == 0 || round == 2 || round == 4) begin
    LED = 16'b1111_1111_1111_1111;
    nums[15:12] = amount_purchased;
    nums[11:8] = D_DASH;
    nums[7:4] = (price_val * amount_purchased) / 10;
    nums[3:0] = (price_val * amount_purchased) % 10;
  end
  else begin
    LED = 16'd0;
    nums = {D_BLANK, D_BLANK, D_BLANK, D_BLANK};
  end
end
CHANGE: begin
  LED = 16'b1111_1111_1111_1111;
  nums[15:12] = amount_purchased;
  nums[11:8] = D_DASH;
  nums[7:4] = (money - price_val * amount_purchased) / 10;
  nums[3:0] = (money - price_val * amount_purchased) % 10;
end
```

接下來都是 block diagram 中負責計算 next_state、next_items、next_price、next_money、next_counter 的 Comb. Circuit 實作解釋，主要以 SET、PAYMENT、BUY、CHANGE states 舉例。

首先，下圖為 SET state 的 code。利用類似上述判斷 SPACE 鍵按下的條件，我們可以用 been_ready&single_key_down&enter_down 來判斷 ENTER 鍵被按下，並在 ENTER 鍵按下後回到 IDLE state。而 items 則如上述只在 space_state==0 時更新，在確認只有單一按鍵按下後，需要額外確認按下的鍵是否是數字(key_num!=NAN)，是數字才能更新 next_items。處理 next_price[1:0] 也是類似的條件，此外由於我的 price 是存十位數、個位數，所以可以直接把 next_price 的十位設為 price 的個位、next_price 的個位設為輸入的 key_num。

```
SET: begin
  // state
  if (been_ready && single_key_down && enter_down) next_state = IDLE;

  // items
  if (space_state == 1'b0) begin
    if (been_ready && single_key_down && key_down[last_change]) begin
      if (key_num != NAN) next_items = key_num;
    end
  end

  // price
  if (space_state == 1'b1) begin
    if (been_ready && single_key_down && key_down[last_change]) begin
      if (key_num != NAN) begin
        next_price[1] = price[0];
        next_price[0] = key_num;
      end
    end
  end
end
```

下圖為 PAYMENT state 的 code。此處 next_state 除了判斷 ENTER 鍵按下外，要根據使用者付的錢是否夠買至少一個商品來判斷下一個 state。如果有商品可買(items > 0)且錢夠買(money >= price_val)，則進入 BUY state，不然就進入 CHANGE state。而 money 的更新則同樣先判斷單一按鍵按下後，根據 key_num 的值決定 money 要增加多少，若增加後會超過 99 元則只加到 99 元(key_num==NAN 的情況這裡不用列出來是因為 next_money 在此 always block 一開始即設過 default 值為 money)。最後一行的 next_counter=0 則是為了準備 BUY 或 CHANGE state 數秒數。

```
PAYMENT: begin
  // state
  if (been_ready && single_key_down && enter_down) begin
    if (items > 0 && money >= price_val) next_state = BUY;
    else next_state = CHANGE;
  end

  // money
  if (been_ready && single_key_down && key_down[last_change]) begin
    case (key_num)
      4'd0: next_money = 7'd0;
      4'd1: next_money = (money < 7'd99) ? money + 1 : 7'd99;
      4'd2: next_money = (money < 7'd95) ? money + 5 : 7'd99;
      4'd3: next_money = (money < 7'd90) ? money + 10 : 7'd99;
      4'd4: next_money = (money < 7'd50) ? money + 50 : 7'd99;
    endcase
  end

  // counter
  next_counter = 0;
end
```

下圖為 BUY、CHANGE states 的 code。由於這兩個 states 主要是用來 display 數量、金額等資訊的，因此沒有太多訊號的 next 值要計算，主要是 counter 要增加一，並在數到三秒

(counter==29'd299_999_999)時歸零，以及在數三秒後換 state(BUY state 換到 CHANGE state、CHANGE state 換到 IDLE state)。此外就是在 CHANGE state 時要在正準備換 state 時(因為太早更新會造成 items 變成新的值造成 7-segment 印出錯的值)，更新 next_items 為原本商品數量減去購買數量(items - amount_purchased)。

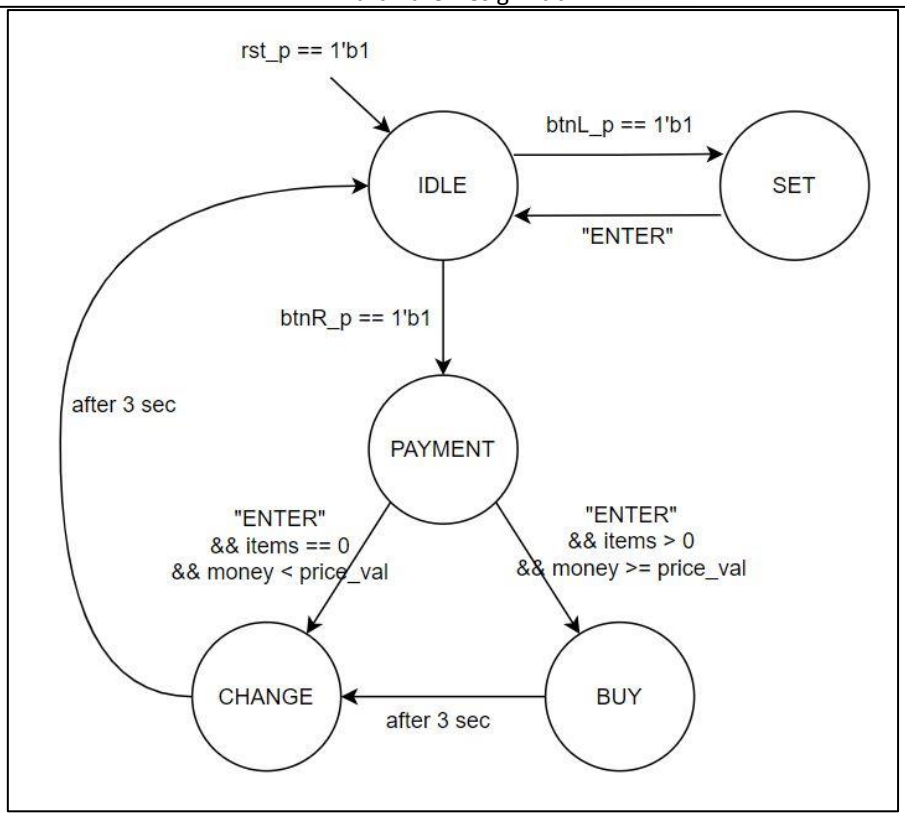
```
BUY: begin
  // state
  if (counter == 29'd299_999_999) next_state = CHANGE;

  // counter
  if (counter == 29'd299_999_999) next_counter = 0;
  else next_counter = counter + 1'b1;
end
CHANGE: begin
  // state
  if (counter == 29'd299_999_999) next_state = IDLE;

  // items
  if (counter == 29'd299_999_999) next_items = items - amount_purchased;

  // counter
  if (counter == 29'd299_999_999) next_counter = 0;
  else next_counter = counter + 1'b1;
end
```

接下來則是此 lab 的 state diagram(如下圖)。由於是 async positive reset，因此不論何時當 rst_p==1 時代表要 reset 並回到 IDLE state。在 IDLE state 時，如果左側按鈕被按下(btnL_p==1'b1)則進入 SET state，如果是右側按鈕被按下(btnR_p==1'b1)，則進入 PAYMENT state。

在 SET state 則較單純，當 ENTER 鍵被按下時回到 IDLE state，而具體判斷 ENTER 鍵按下的條件如先前所述(即 been_ready 且 single_key_down 且 enter_down)。

在 PAYMENT state 在 ENTER 鍵按下時(判斷條件如上所述)，根據不同情況要進到不同 state。如果商品數量大於 0 且使用者的錢足夠支付至少一個商品的價格(即 items > 0 && money >= price_val)則進入 BUY state 去處理購買，否則(即商品數量是 0 且錢不夠支付商品價格)就進入 CHANGE state 找錢。

在 BUY、CHANGE state 都同樣單純等待 3 秒後就進入下一個 state (BUY state 進入 CHANGE state、CHANGE state 回到 IDLE state)，此處 3 秒如先前所述是利用 counter 數的。



B. Questions and Discussions

(a) 多使用一個 Sequential 的 output last_key_down 訊號來記錄上一個 clk cycle 的 key_down 值。由於有同個按鍵一直被按著不放的話，key_down 的值在每個 cycle 更新後會跟上個 cycle 一樣，因此我們可以用 key_down != last_key_down 來確認按下的按鍵不是同一個。

```
reg [511:0] last_key_down;
wire not_same_key = (key_down != last_key_down) ? 1'b1 : 1'b0;

always @(posedge clk, posedge rst_p) begin
  if (rst_p) begin
    last_key_down <= 0;
  end
  else begin
    last_key_down <= key_down;
  end
end
```

再來就是在所有有關按鍵按壓的判斷條件裡都加上 not_same_key==1’b1 這個條件就好，舉例來說檢測 ENTER 按下的條件改為如下圖。

```
if (been_ready && single_key_down && not_same_key && enter_down) next_state = IDLE;
```

(b) 我們希望同樣可以檢測重複按下的不是同個按鍵，但是如果在某個按鍵重複按壓的情況下，按壓另一個按鍵要能有反應。我們可以同樣使用 a 小題的 code，但是在所有有關按鍵按壓的判斷條件裡去掉 single_key_down 的條件，這樣就可以檢測 multiple key down 的情況，舉例來說檢測 ENTER 按下的條件改為如下圖。

```
if (been_ready && not_same_key && enter_down) next_state = IDLE;
```

C. Problem Encountered

在寫計算 next_space_state 的 Comb. Circuit 時，我一開始設 default value 的邏輯想錯，造成測試時按 SPACE 鍵都沒反應，錯誤 code 如下圖所示。因為希望只有在 SET state 才 toggle 值，因此我原本想說初始值都設為 0 就好。但這樣會造成按下 SPACE 鍵後，只有一個 cycle 的 space_state 值是 1，之後的 cycle 因為初始值是 0 且沒有偵測到 SPACE 鍵按下，所以 space_state 又維持在 0，也才造成好像 SPACE 按下都沒反應。

```
always @(*) begin
  next_space_state = 0;
  if (state == SET) begin
      if (been_ready && single_key_down && space_down) next_space_state = ~space_state;
  end
end
```

D. Suggestions

好像建議都在之前的 report 寫過了，所以附個療癒狗狗影片好了: youtube_link。