Lab 6

學號: 108032053 姓名: 陳凱揚

1. 實作過程

在此題中，我使用原始的 100MHz 作為 clock，以接收來自鍵盤的訊號，並及時以 LED 更新 B1、B2 車站等待的人數和目前車上的人數，再使用數 clock cycle 的方式，將其他 flip-flop，如 state、SevenSegment 等能夠以 100/(2^27)MHz 的頻率更新，以下圖 1 的 next_state 為例，只有在 cnt == CNT 時，才會更新 next_state（cnt 為數 clock 的 filp-flop，CNT 為{27{1'b1}}）。

下圖 2 為我建立的 BCD module，用來轉換以 binary 表示的 gas 和 income 至 BCD，由於這些數字最大只有 2 位數，所以只要簡單判斷十位數字，就能得知個位數，並輸出此兩位數的 BCD。

下圖 3 為我判斷鍵盤輸入的方式，當 been_ready 和 key_down[last_change] == 1'b1 時，代表目前有個按鍵被按下了，接著判斷這個鍵是數字 1（KEY_CODE_1）或數字 2（KEY_CODE_2），並對應改變 B1 和 B2 車站的等待人數，其他鍵按下則不會有反應。下圖 4 則是 KeyboardDecorder 的接線方式

在 module lab6 中，我使用了 11 個 flip-flop，功能分別如下：

(1) state：記錄目前狀態，有 6 個狀態，state diagram 如下圖 5 所示，分別為 IDLE（閒置）、GETON（上車）、PAYMENT（付錢）、REFUEL（加油）、MOVE（移動）、GETOFF（下車）。
(2) bus_pos：記錄目前車子的位置，根據 dir 決定移動方向。
(3) wait_B1、wait_B2：B1、B2 車站等待人數，可由鍵盤輸入增加人數，最多至 2 人。
(4) bus_people：記錄目前車上的人數，在上下車時增減人數，最多至 2 人。
(5) gas：記錄目前的油量，每人經過每個加油站減 5，每次加油加 10，最多至 20。
(6) income：記錄目前的金錢，每人上山加 30，下山加 20，每次加油減 10，最多至 90。
(7) nums：16bits 中前 8bits 為 gas_BCD，後 8bits 為 income_BCD。
(8) LED：[15:14]、[12:11]、[10:9]分別為 B1 等待人數、B2 等待人數、車上人數，[6:0]中只有車子所在位置會亮，其他都是暗的，[13]、[8:7]則是永遠保持暗的。
(9) dir：記錄目前車子方向，有 L 跟 R 兩種。
(10) cnt：27bits，每經過一個 clock 就加一。

下圖 5、圖 6 分別為 state diagram 和 block diagram。

```
// next_state
always @* begin
    next_state = state;
    if(cnt == CNT) begin
        case(state)
            IDLE: begin
                if(wait_B1 && wait_B2) next_state = GETON;
                else if(wait_B1) next_state = (bus_pos==B1)?(GETON):(REFUEL);
                else if(wait_B2) next_state = (bus_pos==B2)?(GETON):(REFUEL);
            end
            GETON: begin
                next_state = PAYMENT;
            end
            PAYMENT: begin
                next_state = REFUEL;
            end
            REFUEL: begin
               if(gas==8'd20 || income==8'd0) next_state = MOVE;
            end
            MOVE: begin
                if(bus_people != 2'd0) begin
                    if(next_bus_pos==G1 || next_bus_pos==G3) next_state = GETOFF;
                    else if(next_bus_pos == G2) next_state = REFUEL;
                end else begin
                    if(next_bus_pos==B1 || next_bus_pos==B2) next_state = GETON;
                end
            end
            GETOFF: begin
                if(next_bus_people > 2'd0) next_state = GETOFF;
                else if(wait_B1 && wait_B2) next_state = GETON;
                else if(wait_B1) next_state = (bus_pos==B1)?(GETON):(REFUEL);
                else if(wait_B2) next_state = (bus_pos==B2)?(GETON):(REFUEL);
                else next_state = IDLE;
            end
        endcase
    end
end
```

▲ 圖 1

```
module BCD(
    input [7:0] binary,
    output reg [7:0] BCD);
    wire [7:0] b1, b2, b3, b4, b5, b6, b7, b8, b9;
    assign b1 = binary-8'd10;
    assign b2 = binary-8'd20;
    assign b3 = binary-8'd30;
    assign b4 = binary-8'd40;
    assign b5 = binary-8'd50;
    assign b6 = binary-8'd60;
    assign b7 = binary-8'd70;
    assign b8 = binary-8'd80;
    assign b9 = binary-8'd90;
    // BCD
    always @* begin
        BCD = binary;
        if(binary < 8'd10) BCD = binary;
        else if(binary < 8'd20) BCD = {4'd1, b1[3:0]};
        else if(binary < 8'd30) BCD = {4'd2, b2[3:0]};
        else if(binary < 8'd40) BCD = {4'd3, b3[3:0]};
        else if(binary < 8'd50) BCD = {4'd4, b4[3:0]};
        else if(binary < 8'd60) BCD = {4'd5, b5[3:0]};
        else if(binary < 8'd70) BCD = {4'd6, b6[3:0]};
        else if(binary < 8'd80) BCD = {4'd7, b7[3:0]};
        else if(binary < 8'd90) BCD = {4'd8, b8[3:0]};
        else if(binary < 8'd100) BCD = {4'd9, b9[3:0]};
    end
endmodule
```

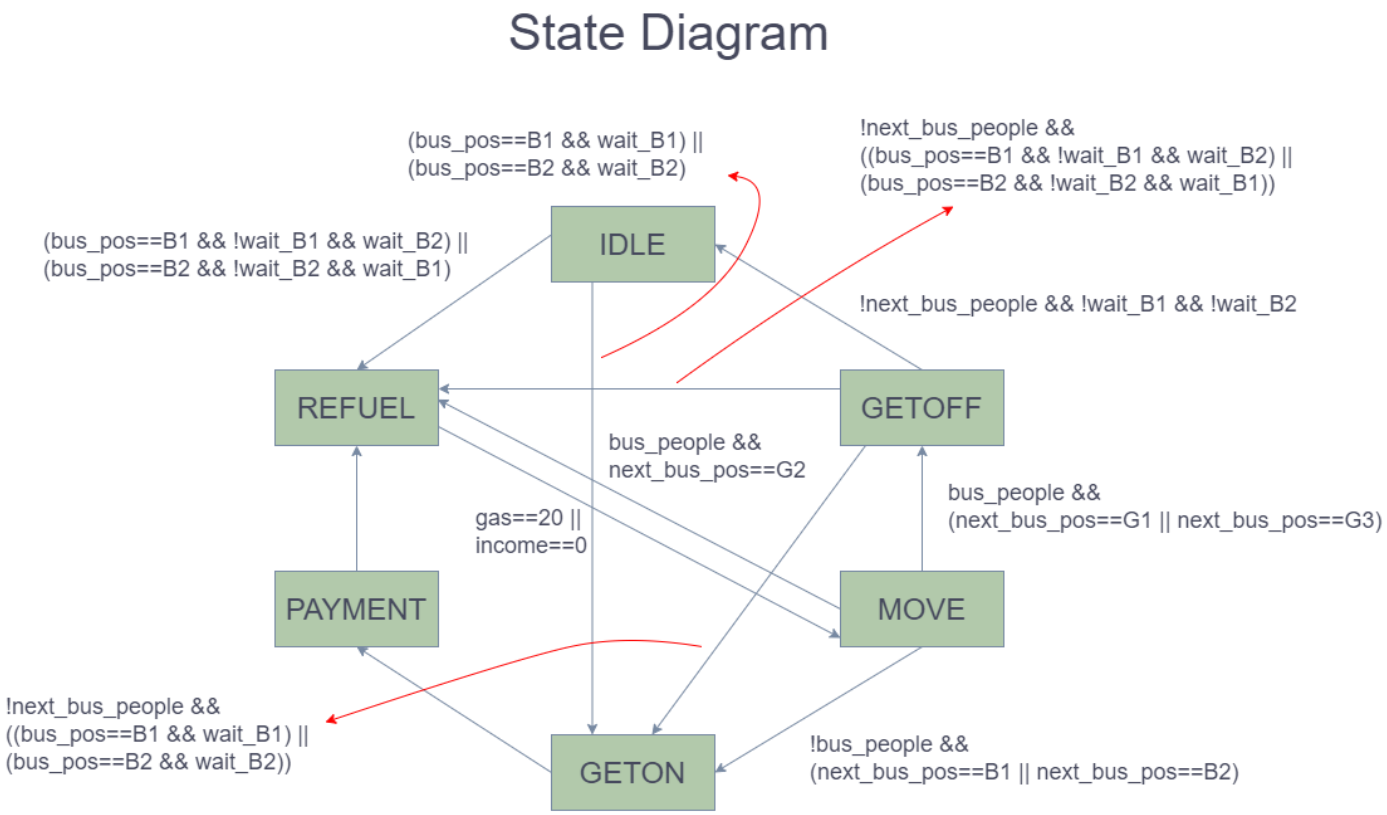
▲ 圖 2

```
end else if(been_ready && key_down[last_change]==1'b1) begin
    if(last_change==KEY_CODE_1 && wait_B1<2'd2) next_wait_B1 = wait_B1+2'd1;
    else if(last_change==KEY_CODE_2 && wait_B2<2'd2) next_wait_B2 = wait_B2+2'd1;
end
```

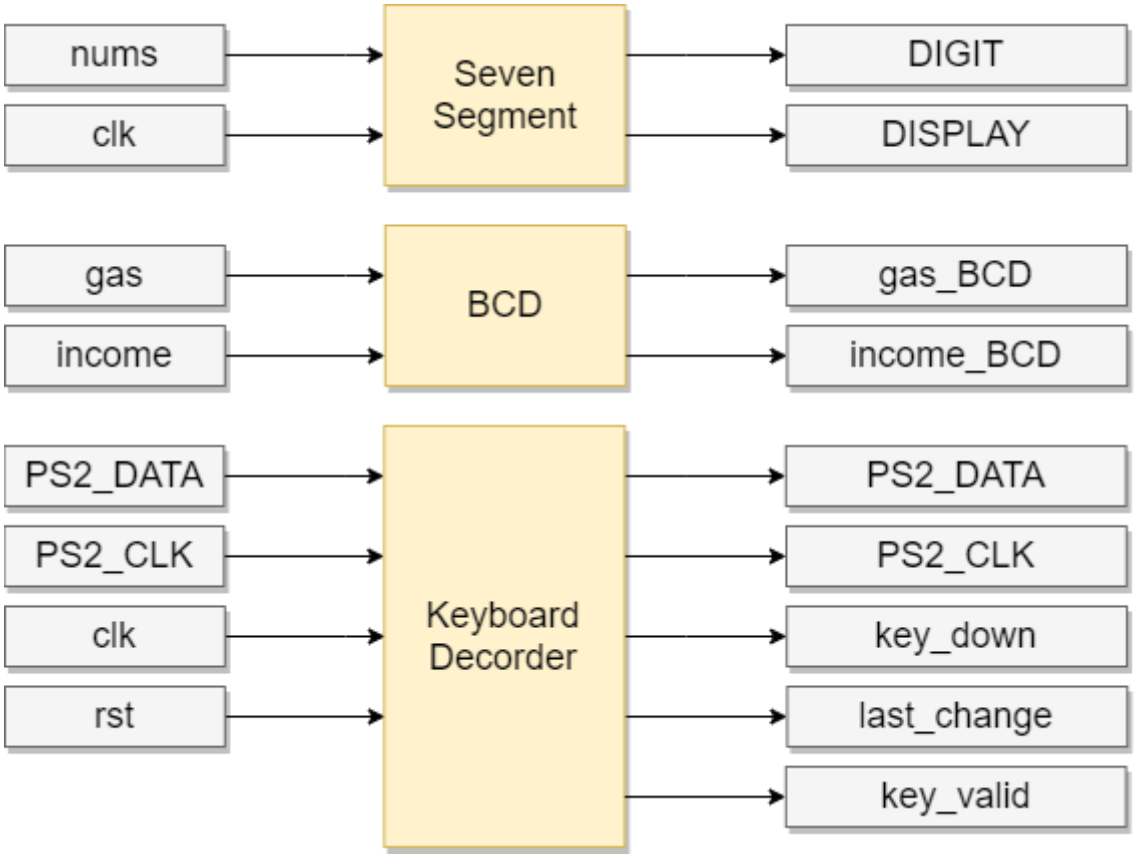
▲ 圖 3

```
KeyboardDecoder key_de(
    .key_down(key_down),
    .last_change(last_change),
    .key_valid(been_ready),
    .PS2_DATA(PS2_DATA),
    .PS2_CLK(PS2_CLK),
    .rst(rst),
    .clk(clk)
);
```

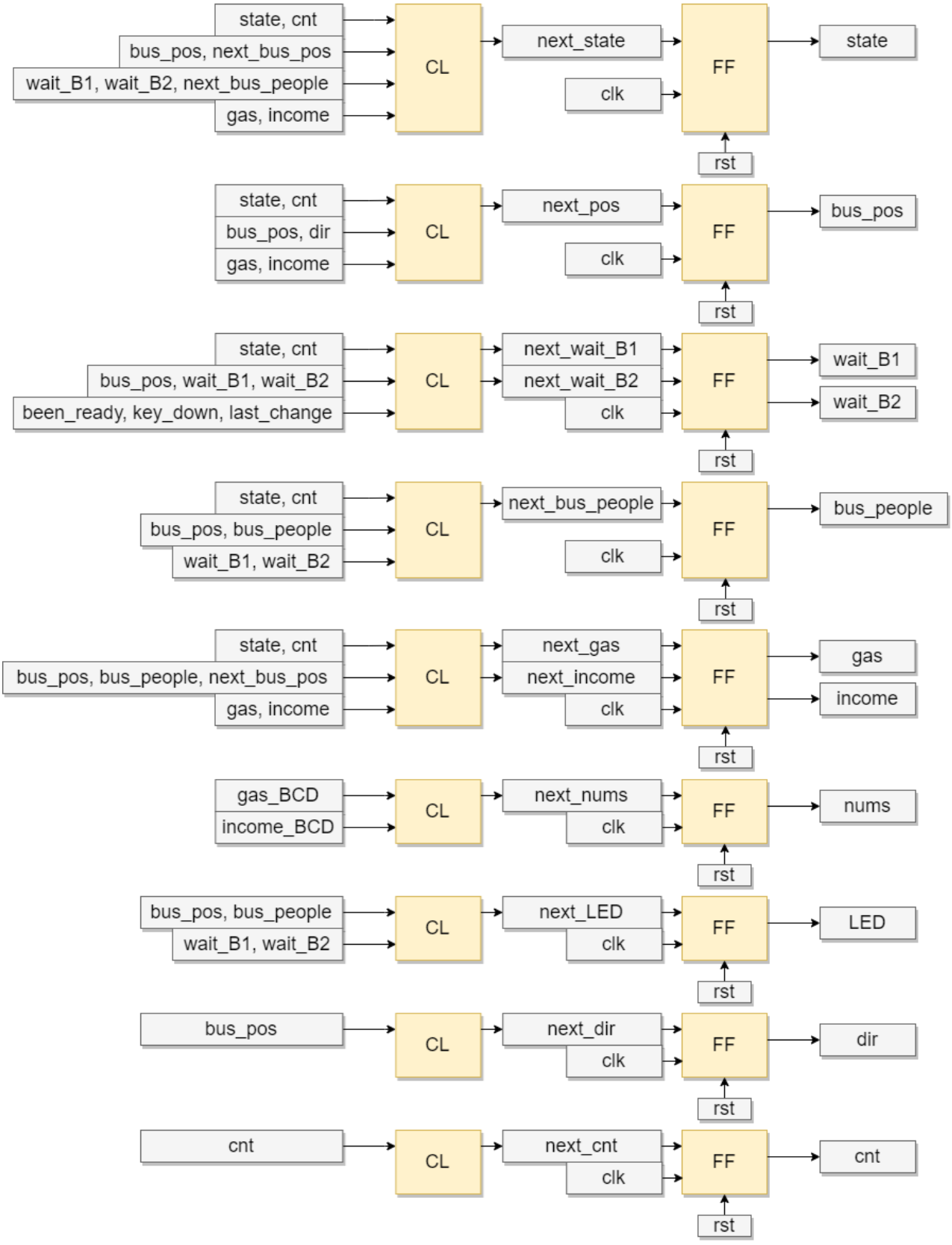
▲ 圖 4


State Diagram
(bus_pos==B1 && wait_B1) ||
(bus_pos==B2 && wait_B2)
!next_bus_people &&
((bus_pos==B1 && !wait_B1 && wait_B2) ||
(bus_pos==B2 && !wait_B2 && wait_B1))
(bus_pos==B1 && !wait_B1 && wait_B2) ||
(bus_pos==B2 && !wait_B2 && wait_B1)
IDLE
!next_bus_people && !wait_B1 && !wait_B2
REFUEL
GETOFF
bus_people &&
next_bus_pos==G2
bus_people &&
(next_bus_pos==G1 || next_bus_pos==G3)
gas==20 ||
income==0
PAYMENT
MOVE
!next_bus_people &&
((bus_pos==B1 && wait_B1) ||
(bus_pos==B2 && wait_B2))
GETON
!bus_people &&
(next_bus_pos==B1 || next_bus_pos==B2)

▲ 圖 5


nums
clk
Seven Segment
DIGIT
DISPLAY
gas
income
BCD
gas_BCD
income_BCD
PS2_DATA
PS2_CLK
clk
rst
Keyboard Decorder
PS2_DATA
PS2_CLK
key_down
last_change
key_valid

Block Diagram



▲ 圖 6

2. 學到的東西與遇到的困難

在這次 lab 中，我覺得 FSM 的複雜度與前幾次不會差太多，但多了不少條件判斷，而且需要自己完整從零設計出來，不像前幾次都有既定流程，我覺得這讓我們可以對設計流程更加熟悉，畢竟通常要求的是功能，而不會有人幫我們設定好完整流程。而我覺得這次最大的困難在於不同 clock 的整合，因為處理鍵盤的輸入，需要 100MHz 的 clock，而車站的等待人數不僅可以使用鍵盤馬上增加人數，也需要除頻過的 clock，在上車時每隔一段時間減少等待人數。一開始我使用兩種 clock 的混用，但處理相對複雜，也出現了不少 bug 無法解決，接著我改採只使用 100MHz 的 clock，加上數 clock cycle 的方式，才順利的完成所需的功能，因此我認為整個 design 還是只使用一個 clock 還是會比較好，不應該偷懶使用多個 clock，造成許多問題。

3. 想對老師或助教說的話