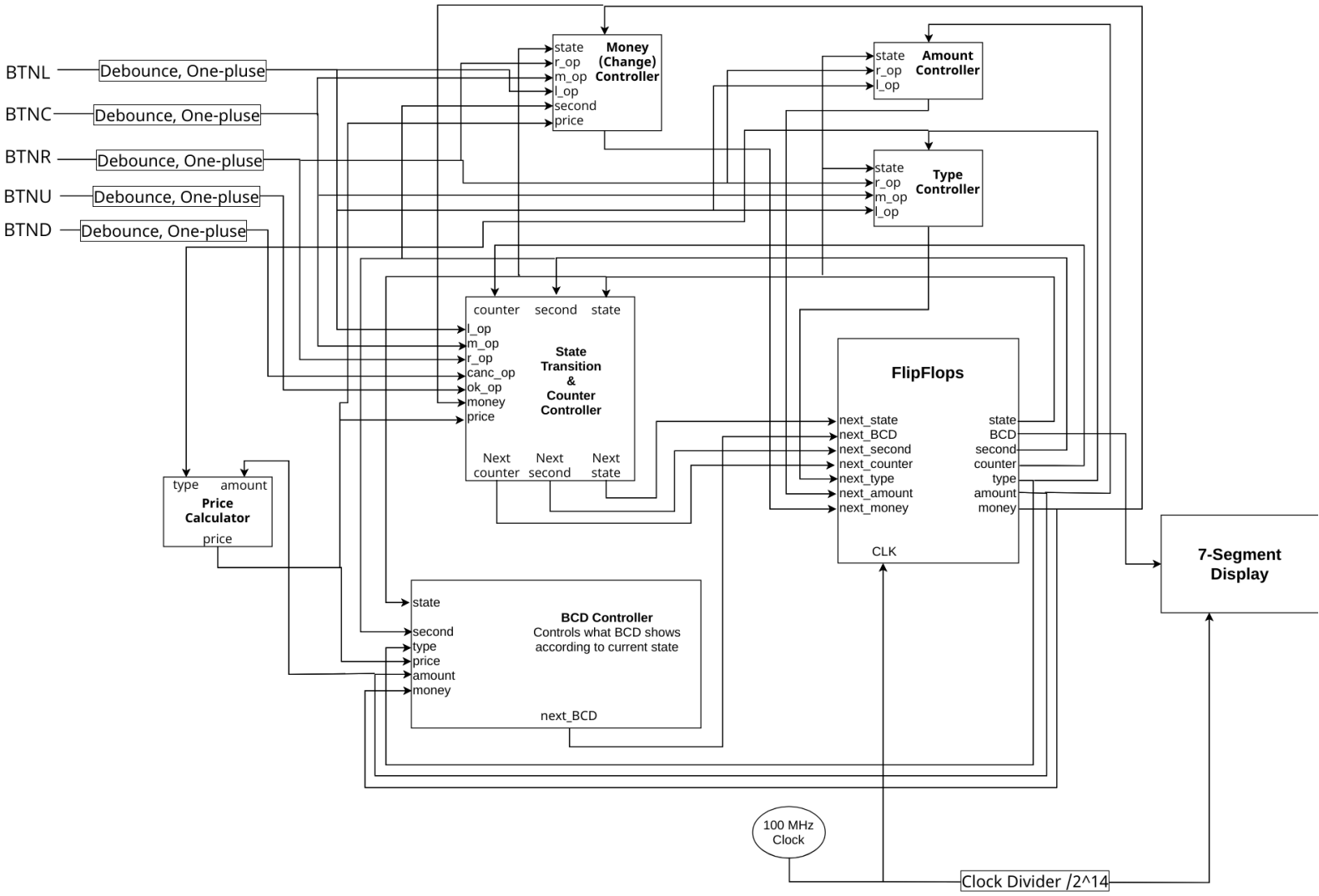
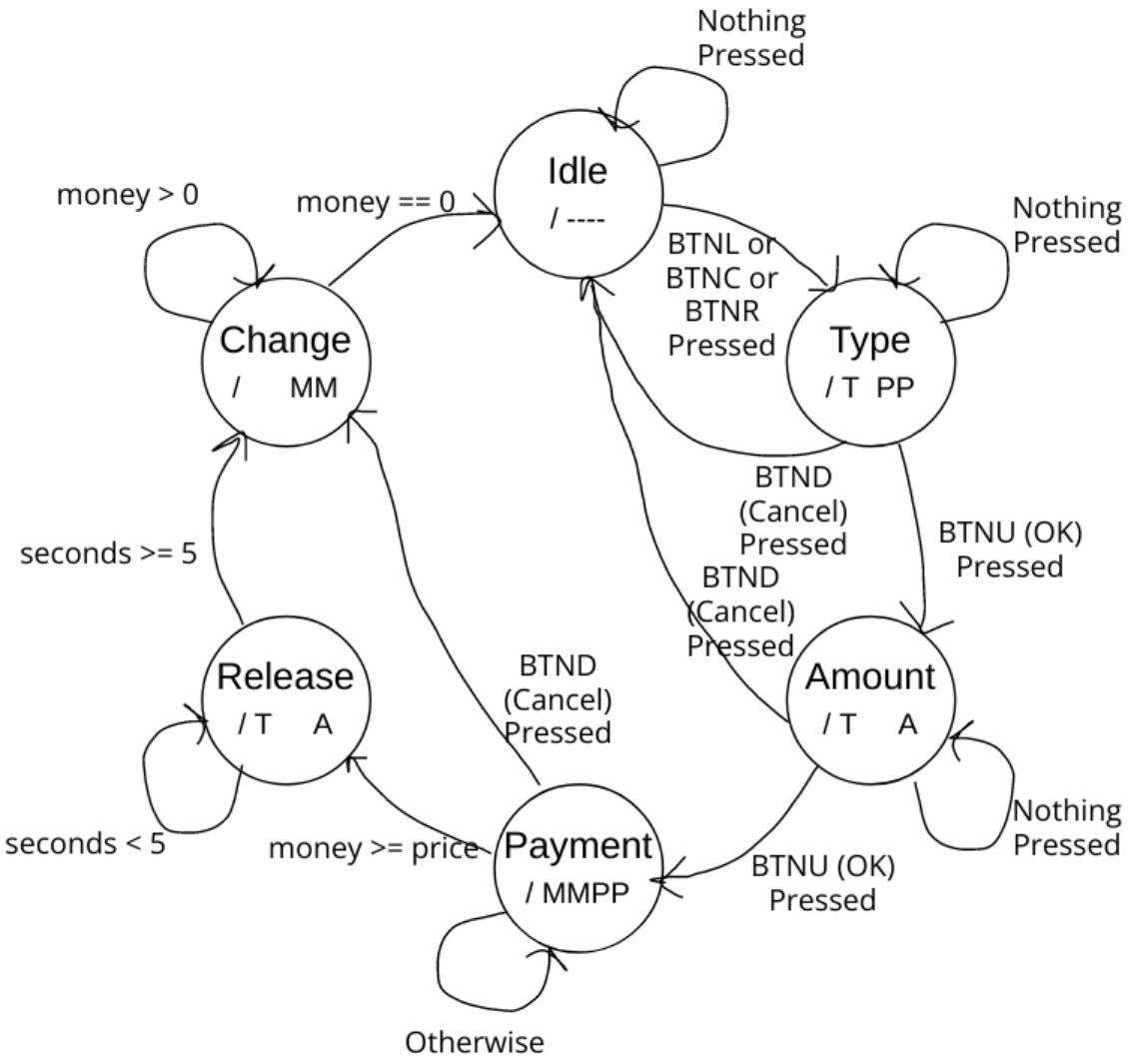
Lab 5

學號: 109062302 姓名: 郭品毅

1. 實作過程
   - 實作使用 Finite State Machine 的 Sequential Circuit



- State Diagram



-

- Flip Flops

```
// Flip Flops
always @(posedge clk, posedge rst) begin
    if (rst) begin
        state <= S_Idle;
        second <= 1;
        counter <= 100000000 - 1;
        type <= 0;
        amount <= 1;
        money <= 0;
        for (i = 0; i < 4; i = i + 1)
            BCD[i] <= BCD_OFF;
    end else begin
        state <= next_state;
        second <= next_second;
        counter <= next_counter;
        type <= next_type;
        amount <= next_amount;
        money <= next_money;
        for (i = 0; i < 4; i = i + 1)
            BCD[i] <= next_BCD[i];
    end
end
```

- 所有的時序電路都使用這樣的 Pattern
- 如果 reset 是 active, 就將所有 state reset
- 每當 clock 來時就將 next_state pass 成目前的 state, 更新為目前的狀態
- BCD 為一個 Array，使用 For Loop 設定 Array 各個項目的值

- State Transition & Counter

```
// State transition logic
reg [2:0] next_state;
reg [3:0] next_second;
reg [28:0] next_counter;
always @* begin
    next_state = state;
    next_second = second;
    next_counter = counter + 1;

    if (next_counter == 100000000) begin // 1 second
        next_second = second + 1;
        next_counter = 0;
    end

    case (state)
        S_Idle: begin
            if (l_op || m_op || r_op)
                next_state = S_Type;
        end
        S_Type: begin
            if (canc_op)
                next_state = S_Idle;
            else if (ok_op)
                next_state = S_Amount;
        end
        S_Amount: begin
            if (canc_op)
                next_state = S_Idle;
            else if (ok_op)
                next_state = S_Payment;
        end
        S_Payment: begin
            if (canc_op)
                next_state = S_Change;
            else if (money >= price)
                next_state = S_Release;
        end
        S_Release: begin
            if (second == 5)
                next_state = S_Change;
        end
        S_Change: begin
            if (second == 1) begin
                next_second = 0; // One-pulse 1 second counter
                if (money == 0) // Needs to stay at 00 for 1 second
                    next_state = S_Idle;
            end
        end
    endcase

    // Reset counter to 0 on state change
    if (next_state != state) begin
        next_second = 0;
        next_counter = 0;
    end
end
```

- 這個電路負責 State 的轉移，以及計算時間用的 Counter 的變化
- 計算時間用的 Register 有兩個：counter 以及 second
  - 每一個 cycle, counter 都會 +1
  - 每當 counter 數到 100M, second 就會 +1, 然後 reset counter 為 0, 數下一個 1秒
    - 因為整個電路跑在 100 MHz 的 Clock, 每 100M 個 Cycle 就會是 1秒
    - 雖然老師上課說電路可以不用跑在那麼快的 Clock 上，但是如果跑在 100M/2^13 Hz 之類的 Clock，要計算 1秒 就要數到一個很奇怪的數字，所以我還是讓他跑在 100MHz, 這樣數到 100M 就會是 1秒了
  - 當 state 轉移時 (next_state != state), counter 以及 second 就會 reset 為 0，重新為下一個 state 算時間
  - 這個方式是使用上課時教的 Mixed Clock 的第一個方法：在 State 裡面算 counter
  - 在 Change 的 State 時，因為要顯示最後找完錢的 00，因此不能讓 money == 0 時就馬上跳到 Idle, 一樣要等 1 秒之後再轉移

- Speed control

```
// Controls BCD
reg [3:0] next_BCD [3:0];
always @* begin
    for (i = 0; i < 4; i = i + 1) begin
        next_BCD[i] = BCD_OFF;
    end

    case (state)
        S_Idle: begin
            for (i = 0; i < 4; i = i + 1) begin
                next_BCD[i] = (!(second & 1))? BCD_DASH: BCD_OFF;
            end
        end
        S_Type: begin
            next_BCD[3] = 10 + type;

            next_BCD[1] = price / 10;
            next_BCD[0] = price % 10;
        end
        S_Amount, S_Release: begin
            next_BCD[3] = 10 + type;

            next_BCD[0] = amount;
        end
        S_Payment: begin
            next_BCD[3] = money / 10;
            next_BCD[2] = money % 10;
            next_BCD[1] = price / 10;
            next_BCD[0] = price % 10;
        end
        S_Change: begin
            next_BCD[1] = money / 10;
            next_BCD[0] = money % 10;
        end
    endcase
end
```

  - 負責控制 BCD 目前顯示的內容
  - 針對目前在哪一個 State, 顯示相對應的資訊
  - 由於 A, S, C 的 BCD 分別為 10, 11, 12, 而 type 為 0, 1, 2，所以顯示 Type 的時候直接用 10 + type 就不需要再分許多 case 了

- Type Controller

```
// Controls type
reg [2:0] next_type;
always @* begin
    next_type = type;
    if (state == S_Idle || state == S_Type) begin
        if (r_op)
            next_type = 0;
        else if (m_op)
            next_type = 1;
        else if (l_op)
            next_type = 2;
    end
end
```

  - 負責控制 Type
  - 預設 type 不變
  - 只有在 S_Idle 以及 S_Type 是按按鈕才會切換 type
- Amount Controller

```
// Controls amount
reg [1:0] next_amount;
always @* begin
    next_amount = amount;
    if (state == S_Idle)
        next_amount = 1;
    else if (state == S_Amount) begin
        if (l_op && amount != 1)
            next_amount = amount - 1;
        else if (r_op && amount != 3)
            next_amount = amount + 1;
    end
end
```

  - 負責控制 Amount
  - 在 S_Idle 時，將 Amount reset 回 1
  - 只有在 S_Amount 時，按左右按鈕才會改變 Amount
- Price Calculator

```
// Calculate price
assign price = (type == 0? 15:
                type == 1? 10: 5) * amount;
```

  - Price 並不是一個 Flip Flop, 是一個 Combinational Circuit 用來計算總票價
  - 因為有很多地方都需要用到這個運算，所以把他獨立分出來

- LED
  - ```
    always @* begin
        for (i = 0; i < 16; i = i + 1)
            LED[i] = (state == S_Idle || state == S_Release) && !(second & 1);
    end
    ```
  - 在 S_Idle 及 S_Release 時每秒閃爍
- 7 Segment Display
  - ```
    reg [3:0] value;

    always @(posedge clk14) begin
        case (DIGIT)
            4'b1110: begin
                value = BCD[1];
                DIGIT = 4'b1101;
            end
            4'b1101: begin
                value = BCD[2];
                DIGIT = 4'b1011;
            end
            4'b1011: begin
                value = BCD[3];
                DIGIT = 4'b0111;
            end
            4'b0111: begin
                value = BCD[0];
                DIGIT = 4'b1110;
            end
            default: begin
                value = BCD[0];
                DIGIT = 4'b1110;
            end
        endcase
    end

    always @* begin
        case (value)
            4'd0: DISPLAY = 7'b100_0000;
            4'd1: DISPLAY = 7'b111_1001;
            4'd2: DISPLAY = 7'b010_0100;
            4'd3: DISPLAY = 7'b011_0000;
            4'd4: DISPLAY = 7'b001_1001;
            4'd5: DISPLAY = 7'b001_0010;
            4'd6: DISPLAY = 7'b000_0010;
            4'd7: DISPLAY = 7'b111_1000;
            4'd8: DISPLAY = 7'b000_0000;
            4'd9: DISPLAY = 7'b001_0000;
            4'd10: DISPLAY = 7'b000_1000; // A
            4'd11: DISPLAY = 7'b001_0010; // S
            4'd12: DISPLAY = 7'b100_0110; // C
            4'd13: DISPLAY = 7'b011_1111; // -
            default: DISPLAY = 7'b111_1111;
        endcase
    end
    ```
  - 使用教材提供的 7-Segment code

2. 學到的東西與遇到的困難
   - 這次主要是再把之前寫的 FSM 和 Sequential Circuit 複習一邊，都是用差不多的流程，好好的把電路寫出來就完成了
     - 這次因為東西比較龐大，我把各個功能分成不同的 Block 處理，每個 Block 都有他負責的信號輸出，這樣 Combitional Circuit 比較不會那麼攏長，哪個 Function 遇到問題就去找哪個 Function 負責的 Block，這樣也比較好 Debug
   - 也遇到了很多粗心的問題：
     - Operator 的優先度問題
     - & (Bitwise and) 的優先度比 == 低, 所以 second & 1 == 0 會是 second & (1 == 0)
       - 以後遇到不確定的 Operator 優先度，就全部加括號，才不會遇到不是我們要的結果
     - money, price 之類的 Register, Bit length 設太小，發現不管怎麼投錢都投不到目標 XD
       - 以後要記得算好需要多少 Bits
     - next_BCD 忘記宣告成 Array, 而且因為 Array 選擇第幾項的語法與選擇哪一個 Bit 的語法相同 (都是用 reg[i]), 所以沒有任何錯誤訊息。
       - Debug 了很久都不知道為什麼 BCD 只顯示 0 或 1
       - 以後要注意這種事情

3. 想對老師或助教說的話
   - 謝謝老師和助教～
   - 感謝助教出了那麼多有趣的題目！