硬體設計與實驗

期末 Project 報告

指導教授：黃稚存

組別：35
成員：105062132 吳政鴻
105062140 陳建愷

動機

進入大學已經三個學期了，雖然在大一上下學期各有做過程式設計的專題作業，但限於能力不足，做出來的遊戲與平常玩的遊戲相差甚遠，這一次的硬實期末專題作業正好能自行採購材料，做出自己想要的設備，因此就想找一個現實生活中常用到的設備，試著以自己的能力做出來，證明自己的所學是有意義的，足以在現實生活中運用。

會挑選電梯作為期末專題的原因其實有一大部分是因為材料上的關係，找幾顆馬達控制電梯上下及電梯門開關，再用冰棒棍及白膠等材料做出電梯，這樣的做法會使得材料的找尋變得相對簡單，不需要另外去找專業的工廠定做特定的零件。

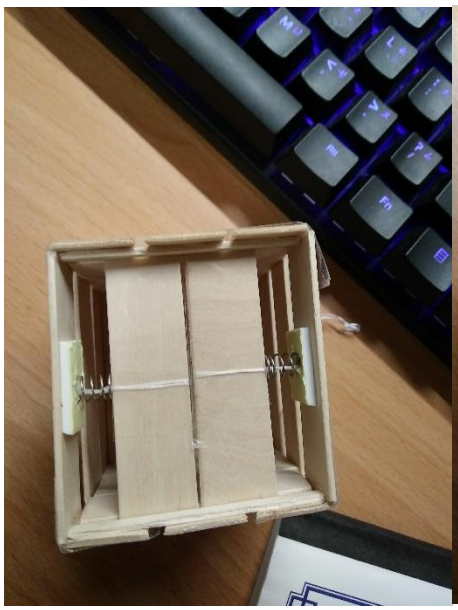
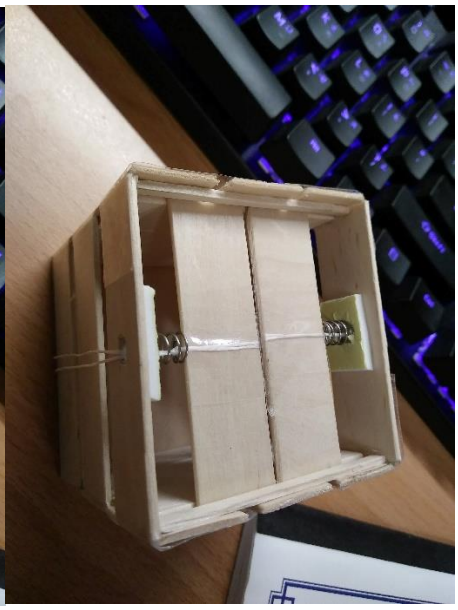
實作過程

電梯實體部分

在剛開始準備電梯材料的時候，原本有想過使用樂高做出電梯的框架，但符合的零件太難找了，加上樂高本身就不便宜，會使得經費增加很多，所以就決定用冰棒棍裁切自己要的樣子，再用白膠黏貼。

而馬達的部分則是到電子材料行購買步進馬達，最剛開始的時候就已設想好一顆馬達控制上下，但在門的部分遇到了困難，因為用一顆麻達接上繩子，綁上門板，馬達轉動時可以拉動繩子，讓門開啟，但遇到關上時，就算馬達反向轉回，也只能讓緊繃的繩子放鬆，並無法讓門歸回原位，如果再加上一顆步進馬達負責門的歸位，會使得整體的設計很凌亂，後來想到的解決方法是在門板與電梯牆壁間的空隙加上一個彈簧，且在門板上裁出一個洞，這樣綁到門板上的繩子就能穿過彈簧及門板，拉到裝在電梯廂後面的馬達並且綁上，這樣馬達轉動時就能讓繩子緊繃，拉動電梯門，反向轉回時也會因為有著彈簧的作用立，使得電梯門能順利關上。

至於如何使彈簧不脫落也是一個問題，因為彈簧的可接觸面積很小，根本無法利用白膠將它與冰幫棍牢固的黏合，就算能黏住不脫落，門打開時彈簧需要受到不小的作用力，可能一被擠壓就脫落了，最後的解決方法則是利用泡棉膠，將泡棉膠挖一個小洞，再將彈簧塞入，最後泡棉膠黏到門板上，這樣便能使得彈簧牢固的黏合在牆壁上，也不會因為擠壓而脫落或者偏離原本的位置。

程式碼部分

這次 final project 用 FPGA 板實作電梯，主要是用到 FSM 的概念。因此設計了幾個 state，分別有 IDLE、UP、DOWN、DOOR_OPEN、DOOR_STOP、DOOR_CLOSE、CHECK，以下會分別介紹，但在介紹這些 state 前，先介紹其他會用到的判斷東西。

為了要判斷電梯的運動狀態，設計了一個 ele_up 的訊號，如果電梯是在上升的階段就把他拉起來為 1；如果再下降階段就把它設為 0；在一樓時因為只能往上，所以也把它拉起來為 1；在三樓時因為只能往下，所以把它設為 0。

```
always@(*)begin
    ele_up_next = ele_up;
    if (state == `UP)
        ele_up_next = 1'b1;
    else if (state == `DOWN)
        ele_up_next = 1'b0;
    else if (floor_cur == 4'd1)
        ele_up_next = 1'b1;
    else if (floor_cur == 4'd3)
        ele_up_next = 1'b0;
end
```

而 floor_cur 則是代表現在電梯所在的樓層，在 IDLE 狀態維持住；在 UP 狀態時當 counter 數到電梯跑一樓的時間時，判斷原本的樓層再加一；數到跑兩樓的時間時直接設為 3，因要上升兩樓一定只有從一樓到三樓；在 DOWN 狀態時當 counter 數到跑一樓的時間時，則判斷原本的樓層後再減一；數到跑兩樓的時間時直接設為 0，因為要下降兩樓一定只有從三樓到一樓。其餘狀態則維持住。

```
always@(*)begin
    floor_cur_next = floor_cur;
    case(state)
        `IDLE : begin
            floor_cur_next = floor_cur;
        end
        `UP : begin
            if (counter == second_1F)begin
                if (floor_cur == 4'd1)
                    floor_cur_next = 4'd2;
                else if (floor_cur == 4'd2)
                    floor_cur_next = 4'd3;
            end
            else if (counter == second_2F)begin
                floor_cur_next = 4'd3;
            end
        end
        `DOWN : begin
            if (counter == second_1F)begin
                if (floor_cur == 4'd3)
                    floor_cur_next = 4'd2;
                else if (floor_cur == 4'd2)
                    floor_cur_next = 4'd1;
            end
            else if (counter == second_2F)begin
                floor_cur_next = 4'd1;
            end
        end
        `DOOR_OPEN : begin
            floor_cur_next = floor_cur;
        end
        `DOOR_STOP : begin
            floor_cur_next = floor_cur;
        end
        `DOOR_CLOSE : begin
            floor_cur_next = floor_cur;
        end
        `CHECK : begin
            floor_cur_next = floor_cur;
        end
    endcase
end
```

為了要判斷各層樓外的上升下降控制面板，設計了 request_up、request_down，皆有 3 個 bit，設為 1 時表示該樓層有需求，最左邊的 bit 表示 3 樓的訊號，再來是二樓與一樓。鍵盤控制都是在右邊的數字區，數字 7 表示一樓的上升，數字 5 表示二樓的下降，數字 8 表示二樓的上升，數字 6 表示三樓的下降。如果按下以上的按鍵，當樓層的 request_up/down 的 bit 就會設為

1。在 DOOR_STOP 狀態時，因已經到達需求樓層，故要把需求的訊號解除，如果是在一樓或三樓，就把該樓的所有訊號解除；如果是在二樓，則判斷現在電梯是上升或下降，解除該樓層該狀態的訊號。同時在 DOOR_STOP 狀態才能接收電梯廂內的訊號，表示想要往哪個樓層，必須按不同於現在樓層才會接收，同時也要與現在的樓層相比，才知道是往上的需求還是往下的需求。

```
always@(*)begin
    request_down_next = request_down;
    request_up_next = request_up;
        if (been_ready && key_down[right_7])begin
            request_up_next[0] = 1'b1;
        end
        if (been_ready && key_down[right_5])begin
            request_down_next[1] = 1'b1;
        end
        if (been_ready && key_down[right_8])begin
            request_up_next[1] = 1'b1;
        end
        if (been_ready && key_down[right_6])begin
            request_down_next[2] = 1'b1;
        end

    if (state== `DOOR_STOP)begin
        if (floor_cur == 4'd1)begin
            request_down_next = {request_down[3:1], 1'b0};
            request_up_next = {request_up[3:1], 1'b0};
        end
        if (floor_cur == 4'd2)begin
            if (ele_up == 1'b0)
                request_down_next = {request_down[3:2], 1'b0, request_down[0]};
            else
            request_up_next = {request_up[3:2], 1'b0, request_up[0]};
        end
        if (floor_cur == 4'd3)begin
            request_down_next = {request_down[3], 1'b0, request_down[1:0]};
            request_up_next = {request_up[3], 1'b0, request_up[1:0]};
        end
        if (been_ready && key_down[right_1])begin
            if(floor_cur!= 4'd1)
                request_down_next[0] = 1'b1;
        end
        if (been_ready && key_down[right_2])begin
            if (floor_cur == 4'd1)
                request_up_next[1] = 1'b1;
            else if (floor_cur == 4'd3)
                request_down_next[1] = 1'b1;
        end
        if (been_ready && key_down[right_3])begin
            if (floor_cur != 4'd3)
                request_up_next[2] = 1'b1;
        end
    end
end
```

為了要判斷電梯接下來要往哪裡跑，設計一個 floor_request 訊號來表示，當 request_up/down 皆為 0 時，表示沒有需求，設為 0；如果電梯是往上的狀態，則先判斷往上的需求，由最低樓層開始看，如果有需求則 floor_request 設為該樓層，若沒有則判斷下降的需求，由最高開始看，有需求則 floor_request 設為該樓層；當電梯是往下的狀態時，則跟往上的狀態相反，先判斷往下的需求，若沒有再判斷往上的需求。

```
always@(*)begin
    floor_request_next = floor_request;
    if (request_down == 4'd0 && request_up == 4'd0)
        floor_request_next = 4'd0;
    else if (ele_up)begin
        if (request_up != 4'd0)begin
            if (request_up[0] == 1'b1)
                floor_request_next = 4'd1;
            else if (request_up[1] == 1'b1)
                floor_request_next = 4'd2;
            else if (request_up[2] == 1'b1)
                floor_request_next = 4'd3;
        end
        else if (request_down!= 4'd0)begin
            if (request_down[2] == 1'b1)
                floor_request_next = 4'd3;
            else if (request_down[1] == 1'b1)
                floor_request_next = 4'd2;
            else if (request_down[0] == 1'b1)
                floor_request_next = 4'd1;
        end
    end
    else if (ele_up == 1'b0)begin
        if (request_down != 4'd0)begin
            if (request_down[2] == 1'b1)
                floor_request_next = 4'd3;
            else if (request_down[1] == 1'b1)
                floor_request_next = 4'd2;
            else if (request_down[0] == 1'b1)
                floor_request_next = 4'd1;
        end
        else if (request_up!= 4'd0)begin
                if (request_up[0] == 1'b1)
                    floor_request_next = 4'd1;
                else if (request_up[1] == 1'b1)
                    floor_request_next = 4'd2;
                else if (request_up[2] == 1'b1)
                    floor_request_next = 4'd3;
        end
    end
end
```

為了要判斷電梯是要上升下降兩樓還是一樓，設計一個 move_2F 的訊號，當 state 不等於 DOOR 相關的狀態時，判斷 floor_request，如果是二樓的話，一定只會移動一樓而已，所以設為 0；要移動兩樓一定是從一樓移動到三樓或是三樓移動到二樓，所以這兩種情況才設為 1；而在二樓時，因為上一次移動是到二樓，所以訊號已經是 0，因此維持住就可以。

```
always@(*)begin
    move2F_next = move2F;
    if (state!= `DOOR_OPEN || state!= `DOOR_STOP || state!= `DOOR_CLOSE)begin
        if (floor_request == 2'd2)
            move2F_next = 1'b0;
        else if ( (floor_cur == 2'd1 && floor_request == 2'd3) ||
            ( floor_cur == 2'd3 && floor_request == 2'd1) )
            move2F_next = 1'b1;
    end
end
```

接下來介紹 state 的轉換，在 IDLE 狀態時，當 floor_request 大於現在的樓層時，則切到 UP 狀態；當 floor_request 小於現在樓層且大於零時，則切到 DOWN 狀態；當 floor_request 等於現在樓層時，則切到 DOOR_OPEN 狀態；當 fix 拉起時，則直接切到 DOOR_OPEN，其餘則是維持住。

```
case(state)
    `IDLE : begin
        if (floor_request > floor_cur)begin
            state_next = `UP;
        end
        else if (floor_request < floor_cur && floor_request != 2'd0)begin
            state_next = `DOWN;
        end
        else if (floor_request == floor_cur)
            state_next = `DOOR_OPEN;
        else if (fix)
            state_next = `DOOR_OPEN;
        else
            state_next = `IDLE;
    end
```

在 UP 或 DOWN 狀態時，如果 move_2F 拉起來時，當 counter 數到移動兩樓的時間時，切到 DOOR_OPEN，不然則維持住；move_2F 沒有拉起來時，當 counter 數到移動一樓的時間時，則切到 DOOR_OPEN，其餘則是維持住。

```
`UP : begin
    if (move2F)begin
        if (counter == second_2F)begin
            state_next = `DOOR_OPEN;
        end
        else
            state_next = `UP;
    end
    else begin
        if (counter == second_1F)begin
            state_next = `DOOR_OPEN;
        end
        else
            state_next = `UP;
    end
end
`DOWN : begin
    if (move2F)begin
        if (counter == second_2F)begin
            state_next = `DOOR_OPEN;
        end
        else
            state_next = `DOWN;
    end
    else begin
        if (counter == second_1F)begin
            state_next = `DOOR_OPEN;
        end
        else
            state_next = `DOWN;
    end
end
```

在 DOOR_OPEN 狀態時，counter 數到開門的時間時則切到 DOOR_STOP，不然就維持住。

在 DOOR_STOP 狀態時，當 fix 拉起來時，則會一直維持住；若是有按下 close 或 counter 數到等待的時間時，會切到 DOOR_CLOSE，其餘則維持住。

在 DOOR_CLOSE 狀態時，counter 數到關門所需時間時切到 CHECK，不然就維持住。

```
`DOOR_OPEN : begin
    if (counter == second_door)
        state_next = `DOOR_STOP;
    else
        state_next = `DOOR_OPEN;
end
`DOOR_STOP : begin
    if (fix)
        state_next = `DOOR_STOP;
    else begin
        if (close_db || counter == second_wait_door)
            state_next = `DOOR_CLOSE;
        else
            state_next = `DOOR_STOP;
    end
end
`DOOR_CLOSE : begin
    if (counter == second_door)
        state_next = `CHECK;
    else
        state_next = `DOOR_CLOSE;
end
```

在 CHECK 狀態下會檢查下一個個樓層的需求，如果 fix 拉起來，則切到 DOOR_OPEN；若 request_up/down 皆為 0 時，則回到 IDLE 狀態；如果電梯現在是往上的狀態時，判斷如果 floor_request 是否大於現在樓層，若大於則切到 UP 狀態，小於且大於 0 則切到 DOWN；若電梯現在是往下的狀態，判斷 floor_request 是否小於現在樓層且大於 0，若是則切到 DOWN 狀態，若不是則切到 UP 狀態。

```
`CHECK : begin
    if (fix)
        state_next = `DOOR_OPEN;
    else if (request_down == 3'd0 && request_up == 3'd0)
        state_next = `IDLE;
    else begin
        //判斷電梯運動狀態 要往上還往下
        if (fix)
            state_next = `DOOR_OPEN;
        else if (ele_up)begin
            if (floor_request > floor_cur)
            state_next = `UP;
            else if (floor_request < floor_cur && floor_request != 2'd0)
            state_next = `DOWN;
        end
        else begin
            if (floor_request < floor_cur && floor_request != 2'd0)
                state_next = `DOWN;
            else if (floor_request > floor_cur)
                state_next = `UP;
        end
    end
end
```

Counter 則是在 IDLE 和 CHECK 狀態設為 0，其餘則是若狀態沒有改變則數字加一，有改變則歸零。在 DOOR_STOP 下，若有按下 open 則 counter 會歸零，延長開門的時間。

```
always@(*)begin
    case(state)
        `IDLE : begin
            counter_next = 0;
        end
        `UP : begin
            counter_next = counter + 1;
            if (state_next != `UP)
                counter_next = 0;
        end
        `DOWN : begin
            counter_next = counter + 1;
            if (state_next != `DOWN)
                counter_next = 0;
        end
        `DOOR_OPEN : begin
            counter_next = counter + 1;
            if (state_next != `DOOR_OPEN)begin
                counter_next = 0;
            end
        end
        `DOOR_STOP : begin
            counter_next = counter + 1;
            if (state_next != `DOOR_STOP)begin
                counter_next = 0;
            end
            else if (open_db)begin
                counter_next = 0;
            end
        end
        `DOOR_CLOSE : begin
            counter_next = counter + 1;
            if (state_next != `DOOR_CLOSE)begin
                counter_next = 0;
            end
        end
        `CHECK : begin
            counter_next = 0;
        end
    endcase
end
```

馬達控制則是依據不同的狀態給予不同的值，在 IDLE 狀態時 en 全部為 0；在 UP 和 DOWN 狀態 en_ele 皆為 1，en_door 皆為 0；在 DOOR_OPEN 和 DOOR_CLOSE 狀態 en_ele 為 0，en_door 為 1；DOOR_STOP 則皆為 0；CHECK 則

皆為 0。

```
always@(*)begin
    case(state)
        `IDLE : begin
            en_door_next = 1'b0;
            en_ele_next = 1'b0;
        end
        `UP : begin
            en_door_next = 1'b0;
            en_ele_next = 1'b1;
        end
        `DOWN : begin
            en_door_next = 1'b0;
            en_ele_next = 1'b1;
        end
        `DOOR_OPEN : begin
            en_door_next = 1'b1;
            en_ele_next = 1'b0;
        end
        `DOOR_STOP : begin
            en_door_next = 1'b0;
            en_ele_next = 1'b0;
        end
        `DOOR_CLOSE : begin
            en_door_next = 1'b1;
            en_ele_next = 1'b0;
        end
        `CHECK : begin
            en_door_next = 1'b0;
            en_ele_next = 1'b0;
        end
    endcase
end
```

馬達方向控制則是根據所需方向給予不同的值，根據實驗得到質與方向的對應。

```
always@(*)begin
    case(state)
        `IDLE : begin
            direction_door_next = 1'b0;
            direction_ele_next = 1'b0;
        end
        `UP : begin
            direction_door_next = 1'b0;
            direction_ele_next = 1'b0;
        end

        `DOWN : begin
            direction_door_next = 1'b0;
            direction_ele_next = 1'b1;
        end

        `DOOR_OPEN : begin
            direction_door_next = 1'b0;
            direction_ele_next = 1'b0;
        end
        `DOOR_STOP : begin
            direction_door_next = 1'b0;
            direction_ele_next = 1'b0;
        end
        `DOOR_CLOSE : begin
            direction_door_next = 1'b1;
            direction_ele_next = 1'b0;
        end
        `CHECK : begin
            direction_door_next = 1'b0;
            direction_ele_next = 1'b0;
        end
    endcase
end
```

七段顯示器四位數字分別表示有按上的樓層、有按下的樓層、電梯要前往的樓層和現在的樓層。

```
always@(posedge clk_16_div)begin
    case(DIGIT)
        4'b1110:begin
            value=floor_request;
            DIGIT=4'b1101;
        end
        4'b1101:begin
            value=request_down;
            DIGIT=4'b1011;
        end
        4'b1011:begin
            value=request_up;
            DIGIT=4'b0111;
        end
        4'b0111:begin
            value=floor_cur;
            DIGIT=4'b1110;
        end
        default:begin
            value=4'd0;
            DIGIT=4'b1110;
        end
    endcase
end
```

而 LED 燈則是顯示當前的狀態，依照先前介紹的順序，每次皆多亮一顆燈。

```
always@(*)begin
    case(state)
        `IDLE : LED = 16'b1000_0000_0000_0000;
        `UP : LED = 16'b1100_0000_0000_0000;
        `DOWN : LED = 16'b1110_0000_0000_0000;
        `DOOR_OPEN : LED = 16'b1111_0000_0000_0000;
        `DOOR_STOP : LED = 16'b1111_1000_0000_0000;
        `DOOR_CLOSE : LED = 16'b1111_1100_0000_0000;
        `CHECK : LED = 16'b1111_1110_0000_0000;
        default : LED = 16'b1000_0000_0000_0000;
    endcase
end
```

馬達控制

步進馬達一共有四個訊號需要控制，想要讓馬達能順利轉動只需要將這四個訊號輪流輸出為 1，因此只需要一個 FSM 控制，讓狀態輪流切換，在特定狀態輸出特定 PIN 孔的訊號即可，而馬達反向則是將 FSM 的切換順序反向

```
always@(*)begin
    case(present_state)
    sig4:       next_state = (dir && en) ? sig1 : (en) ? sig3 : sig0;
    sig3:       next_state = (dir && en) ? sig4 : (en) ? sig2 : sig0;
    sig2:       next_state = (dir && en) ? sig3 : (en) ? sig1 : sig0;
    sig1:       next_state = (dir && en) ? sig2 : (en) ? sig4 : sig0;
    sig0:       next_state = (en) ? sig1 : sig0;
    default:    next_state = sig0;
    endcase
end

always@(posedge clk, posedge rst)begin
    if (rst)    present_state = sig0;
    else        present_state = next_state;
end

always@(*)begin
    case(present_state)
        sig0:       signal = 4'b0000;
        sig1:       signal = 4'b0001;
        sig2:       signal = 4'b0010;
        sig3:       signal = 4'b0100;
        sig4:       signal = 4'b1000;
        default:    signal = 4'b0000;
    endcase
end
```

上面的 code 中，sig0~sig4 代表 state，signal 則是控制馬達的 output，en 與 dir 則是為了程式方便撰寫所設置的訊號

在 en 與 dir 皆為 1 時，sig0 會變為 sig1，sig1 變為 sig2，sig2 變為 sig3，sig3 變為 sig4，sig4 再變為 sig1；在 en 為 1，dir 為 0 時，sig0 變為 sig1，sig1 變為 sig4，sig4 變為 sig3，sig3 變為 sig2，sig2 變為 sig1，sig1 再變為 sig4，也就是狀態反向切換，這樣馬達便能反向運轉，馬達的狀態切換需要的 clock 也不能過快或過慢，兩者皆會導致馬達無法運轉，目前找到適合的 clock 是 2^18。

馬達的供電需求是 5V，由於一次需要運作兩組馬達，所以一塊電源模組需要個別分出一個電源給馬達，最剛開始找的電源供應器為 5V 的，但發現不足以一次運轉兩顆馬達，馬達在運轉上會顯得相當緩慢，解決方法是找了一個 12V 的電源供應器，電源模組可以支援到 12V，在測試過後發現此方法可行，兩顆馬達也可以順利以正常的速度轉動。

遇到的困難與感想

一開始 vivado 一直找不到其他的 module，試了很多次也檢查過好幾次 code，都沒有問題，後來是在 vivado 裡面新增檔案，原本是直接加進去，這樣就解決了，應該是 vivado 的 bug，浪費了 2 、3 個小時在 de 這個 bug。

還有更改樓層的問題，原本是用加的，發現這樣很容易會 overflow，於事都直接設定值給他，這樣就比較不會有 bug。

不過最難的還是想出這整套的演算法，雖然沒有到很完美，但基本的判斷都有了，蠻有成就感的，這一學期 verilog 也深入了解不少，可說是收穫滿滿。