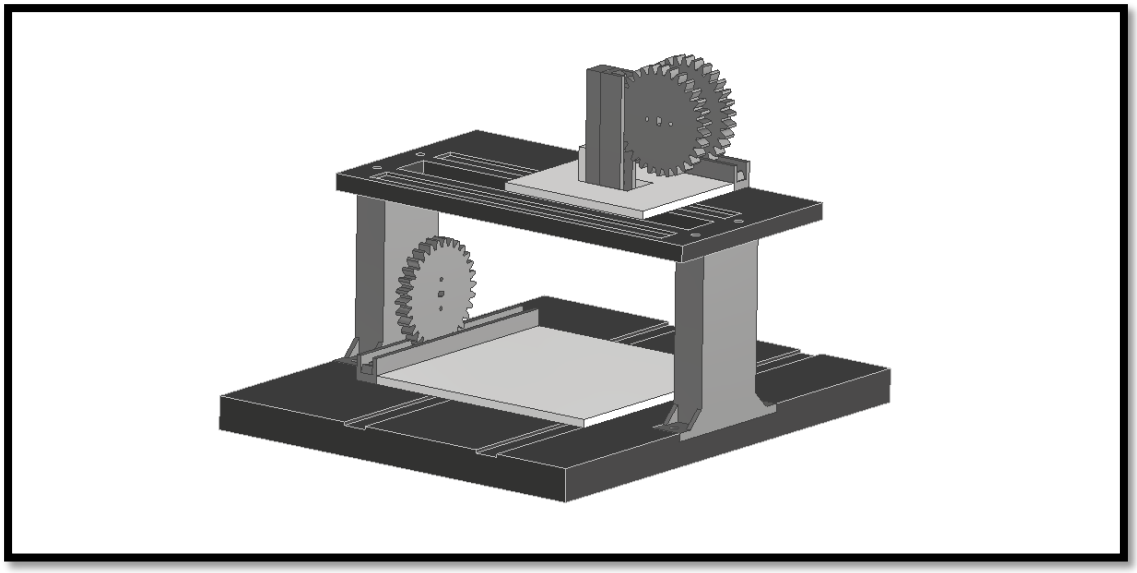
Final Project Report

組長:劉奇泓 109033135

組員:洪聖祥 109062315

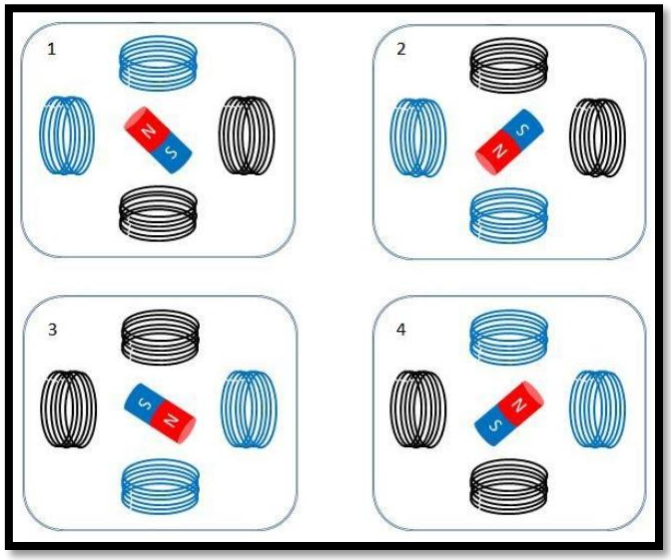
1. 機構介紹



我們使用 Autodesk Inventor 製圖，將機構大致畫完之後將齒條以及齒輪的部分用 3D 列印出來，剩餘的部分基於成本考量，我們購買適合的板子自行製作。示意圖中的兩個白色板子旁邊的齒輪我們使用步進馬達帶動，分別控制寫字機的 x、y 軸；上方的板子有一個缺口可以架上筆，旁邊的齒輪我們則使用伺服馬達帶動，使筆可以上下移動。

- 步進馬達(step motor)

我們使用的步進馬達的型號為 28BYJ-48，並附有驅動板可以對馬達進行控制，在控制板上有兩個接腳是要接電壓以及接地的，我們使用 9V 電池進行供電；另外還有四個接腳可以控制馬達內的四個線圈使馬達轉動，我們為了追求更大的傳動能力而選擇使用二相激磁的方式，如下圖所示，同時對鄰近的兩個線圈通電並以固定的頻率輪流通電，讓馬達上的磁鐵被線圈吸引而轉動。

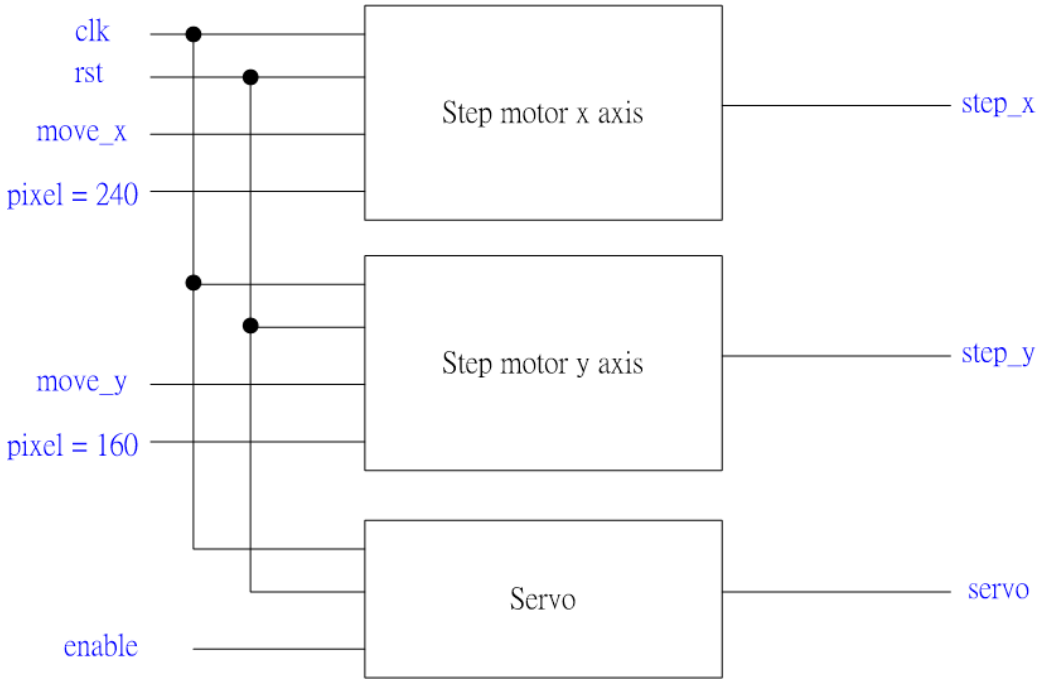


- 伺服馬達(servo motor)

我們使用的伺服馬達的型號是 SG90，有三個接線，棕色的是接地 GND、紅色的是接正電、橘色的是接 PWM 脈衝信號，而控制伺服馬達的方式就是在 FPGA 輸出 PWM，把脈衝信號的週期固定在 50HZ(20ms)，並把高電位的持續時間維持在 1.0ms~2.0ms，就能控制角度在-90°~90°之間。

2. 馬達控制

Block diagram:



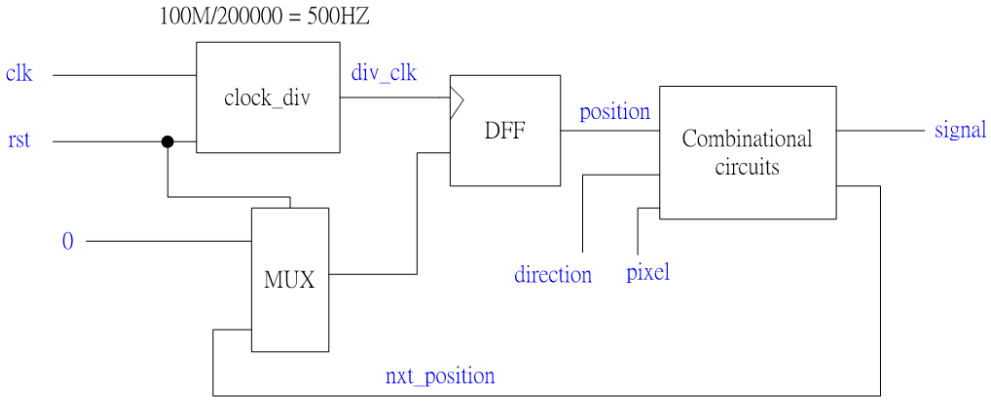
move_x/move_y	
00	不動
01	向前/向右
10	向後/向左
11	回到原點

enable	
0	筆抬起
1	筆放下

傳遞進來的訊號分成 move_x 會讓 x 軸方向的步進馬達移動，move_y 讓 y 軸方向的步進馬達移動，以及 enable 讓伺服馬達控制筆往上或往下，如上表所示。另外有 pixel 可以設定紙面上陣列大小，我們設定 x 方向有 240 格，y 方向有 160 格。輸出 step_x、step_y 是給步進馬達中的線圈的訊號，servo 是給伺服馬達的 PWM。以下再對控制步進馬達和伺服馬達的 module 進行詳細說明。

- Step_motor:

  block diagram:



Step_motor 中有一個 clock divider 可以給出步進馬達供給電流的頻率，也決定了馬達的轉速，我們設定的頻率是 500HZ。再來有一個 MUX 可以將 position 在 reset 時歸零，或者是更新到下個 cycle。position、direction(move_x/y)和 pixel 會被送進一些 combinational circuit 中，輸出 signal 為馬達的訊號。

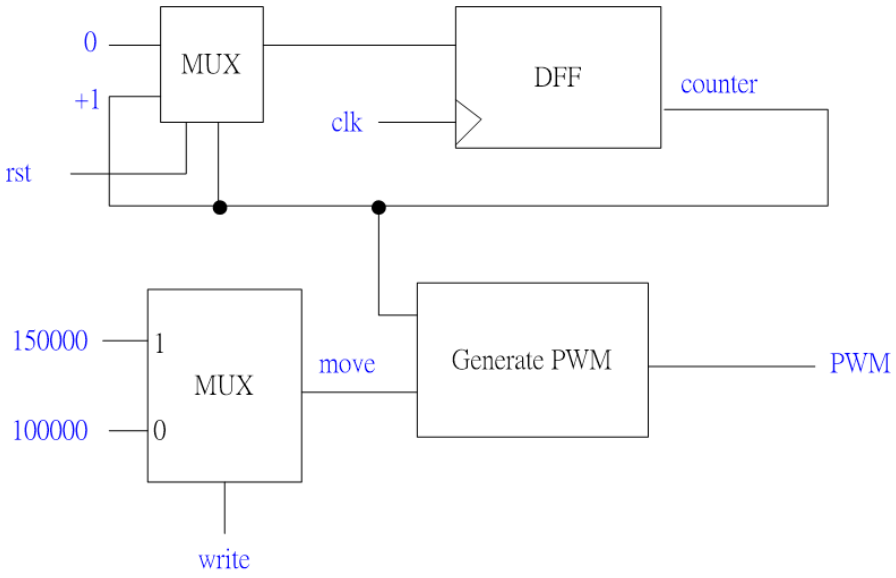
```
always @ (*) begin
    case(direction)
    2'b00: begin
        nxt_position = position;
        signal = 4'd0;
        end
    2'b01 : begin
        if(position < (pixel * 5)) begin
            nxt_position = position + 1'b1;
            case(position[1:0])
            2'b00 : signal = 4'b1000;
            2'b01 : signal = 4'b0100;
            2'b10 : signal = 4'b0010;
            2'b11 : signal = 4'b0001;
            endcase
            end
        else begin
            nxt_position = position;
            signal = 4'd0;
            end
        end

        2'b10, 2'b11 : begin
            if(position > 9'd0) begin
                nxt_position = position - 1'b1;
                case(position[1:0])
                2'b00 : signal = 4'b1000;
                2'b01 : signal = 4'b0100;
                2'b10 : signal = 4'b0010;
                2'b11 : signal = 4'b0001;
                endcase
                end
            else begin
                nxt_position = position;
                signal = 4'd0;
                end
            end
        endcase
    end
```

Direction = 00 時馬達不動，因此 position 不變，direction = 01 時馬達正轉，position 在小於 pixel*5 時每個 cycle 都會加 1(position 走五步等於一格 pixel)，direction = 10、11 時馬達反轉，position 在大於 0 時每個 cycle 都會減 1，其中注意到 position 不變時 signal 的 4bit 都要為 0，否則可能會因為線圈的電流過大而讓驅動模組的電感損壞。設定好 position 後，signal 會隨著

position[1:0]變化，輪流給四個線圈電流，讓步進馬達可以實現正轉和反轉。

- servo_motor:

  block diagram:



servo_motor 中有一個 counter 會可以設定伺服馬達 PWM 的頻率，根據型號我們設定為 50HZ，另外有一個 MUX 可以根據 write 的訊號切換 PWM 高電位的時間，給 1ms 時會固定在-90 度，1.5ms 時則會固定在 0 度，輸出的 move 和 counter 會一起輸入到 combinational circuit 中產生 PWM。

```
always @ (posedge clk) begin
    if (rst == 1'b1 || count == 21'd2000000)
        count <= 21'b0;
    else
        count <= count + 1'b1;
    end

always @ (*) begin
    if (count < move)
        begin
        PWM = 1'b1;
        end
    else
        begin
        PWM = 1'b0;
        end
    end
```

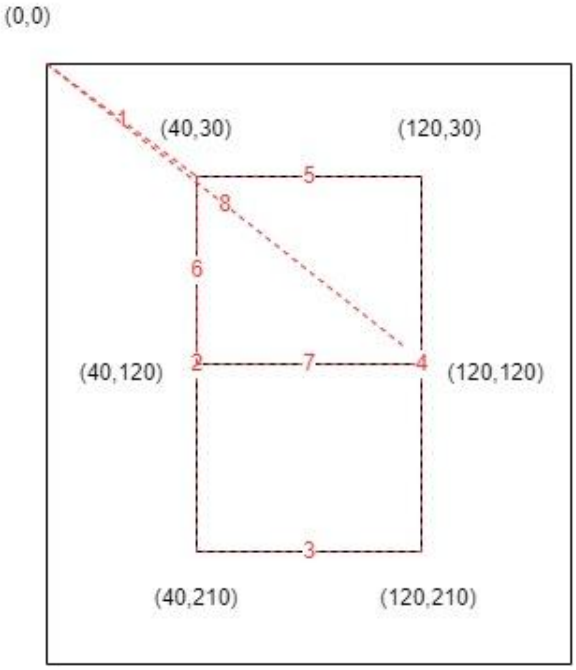
當 count < move 時，表示 PMW 要持續在高電位，大於 move 後 PWM 回到低電位。

3. 產生訊號

- draw_line.v（處理一個筆劃的直線**移動**）

要如何讓寫字機移動是接下來的問題。我們在前面已經寫好控制 x y 軸的步進馬達。現在我們需要告訴步進馬達什麼時候要正著轉甚麼時候要反著轉。而寫字機最基本的動作是移動，我決定寫一個 module 來控制步進馬達寫字時要往哪裡移動。我把一個字分成由好幾條直線所構成，也覺得說不定我只需要用一個 module 來控制起點終點，並且在一個筆畫畫完之後只需要改變起點終點這樣就可以只用一個 module 來寫完一個字。

想法:(以數字 8 為例)



我把紙張座標化,這樣可以利用座標來更改起點終點讓寫字機可以知道該往哪裡移動。

上圖以數字 8 為例,第一次移動是從(0,0)(起點)到(40,30)(終點),第二次移動是從(40,30)(起點)到(40,120)(終點),…,共移動 8 次。

1.(0,0) -> (40,30)

2.(40,30) -> (40,210)

3.(40,210) -> (120,210)

4.(120,210) -> (120,30)

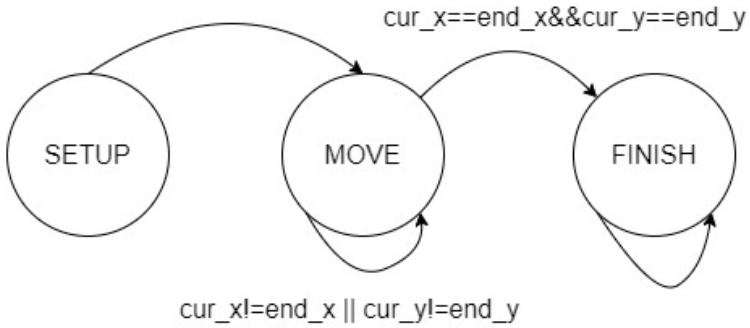
5.(120,30) -> (40,30)

6.(40,30) -> (40,120)

7.(40,120) -> (120,120)

8.(120,120) -> (0,0)

State Diagram



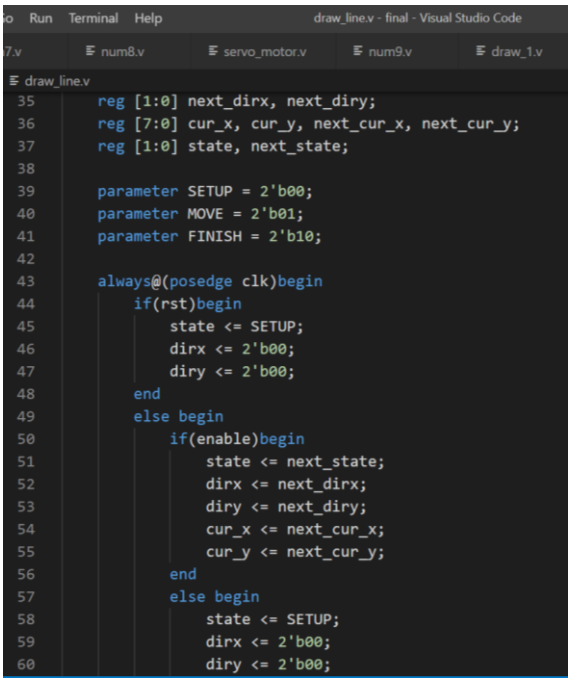
SETUP: 把 input x 、 y 座標讀進來並設值給 cur_x, cur_y

MOVE: cur_x、cur_y 跟 end_x、end_y 比較，如果 cur_x、cur_y 還沒有到達 end_x 、 end_y，繼續在 MOVE state。反之，往 FINISH state。

FINISH: 結束，output 一個 done = 1’b1。如果不是在 FINISH state，done = 1’b0。

Coding Detail

Sequential Part



cur_x 和 cur_y 代表現在的位置移動到哪裡了，dirx 和 diry 是兩個 output signal 要給步進馬達該正著轉還是該反著轉。dirx == 2’b10 代表寫字機的馬達要做到把筆往左移動。反之，寫字機的馬達要做到把筆往左移動。而 diry == 2’b10 寫字機的馬達要做到把筆往前移動。反之，寫字機的馬達要做到把筆往後移動。

Combinational Part

```
always@(*)begin
    case(state)
        SETUP:begin
            next_state = MOVE;
        end
        MOVE:begin
            if(cur_x == endx && cur_y == endy)next_state = FINISH;
            else next_state = MOVE;
        end
        FINISH:begin
            next_state = FINISH;
        end
    endcase
end
```

我希望不會產生 timing violation，不會有很多層 if else 條件，所以我把 always block 分開來寫，每一個 always block 分別處理不同的訊號。上圖是專處理 next_state，如果在 SETUP，我們 next_state 直接進入 MOVE。MOVE 要判斷是否 cur_x==end_x&&cur_y==end_y 才會進入 FINISH。

```
always@(*)begin
    case(state)
        SETUP:begin
            next_cur_x = startx;
            next_dirx = 2'b00;
        end
        MOVE:begin
            if(cur_x != endx)begin
                if(cur_x < endx)begin
                    next_dirx = 2'b01;
                    next_cur_x = cur_x + 8'd1;
                end
                else begin
                    next_dirx = 2'b11;
                    next_cur_x = cur_x - 8'd1;
                end
            end
            else begin
                next_dirx = 2'b00;
                next_cur_x = cur_x;
            end
        end
```

上圖是專門處理 x 軸方向(next_cur_x 和 next_dirx)的 always block。在 SETUP，我要把 input signal startx 讀進來並設值給 next_cur_x。 MOVE 是要看 cur_x 是否等於 end_x，如果 cur_x < end_x，next_dir = 2’b01(筆要往右移動)。反之，next_dir = 2’b10(筆要往左移動)。並且要更新 cur_x。

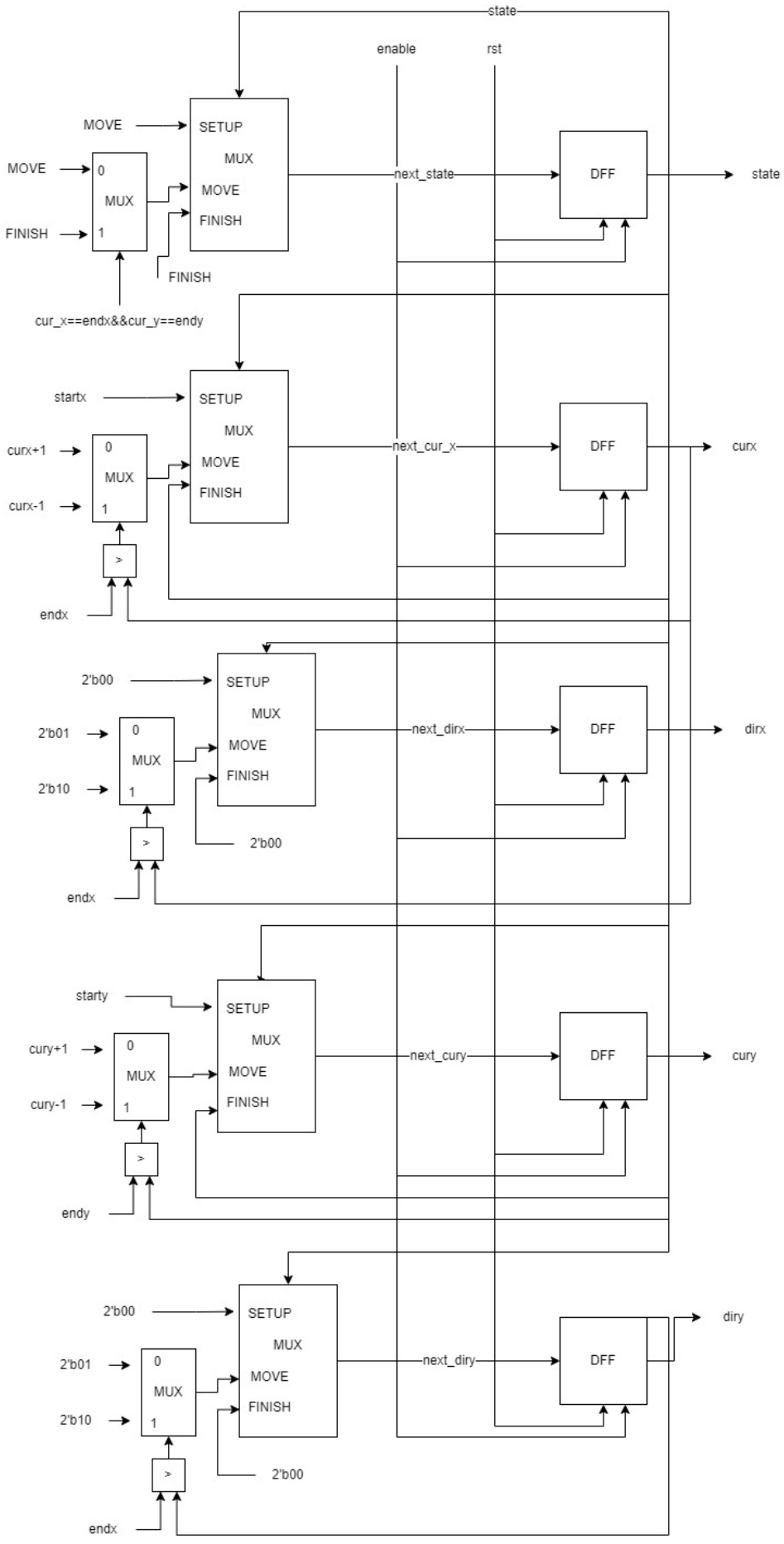
```
always@(*)begin
    case(state)
        SETUP:begin
            next_cur_y = starty;
            next_diry = 2'b00;
        end
        MOVE:begin
            if(cur_y != endy)begin
                if(cur_y < endy)begin
                    next_diry = 2'b01;
                    next_cur_y = cur_y + 8'd1;
                end
                else begin
                    next_diry = 2'b11;
                    next_cur_y = cur_y - 8'd1;
                end
            end
            else begin
                next_diry = 2'b00;
                next_cur_y = cur_y;
            end
        end
        FINISH:begin
            next_diry = 2'b00;
```

上圖的邏輯跟 x 軸的移動邏輯一模一樣，只差在是處理 y 軸的訊號而已。

```
assign done = (state == FINISH) ? 1'b1 : 1'b0;
```

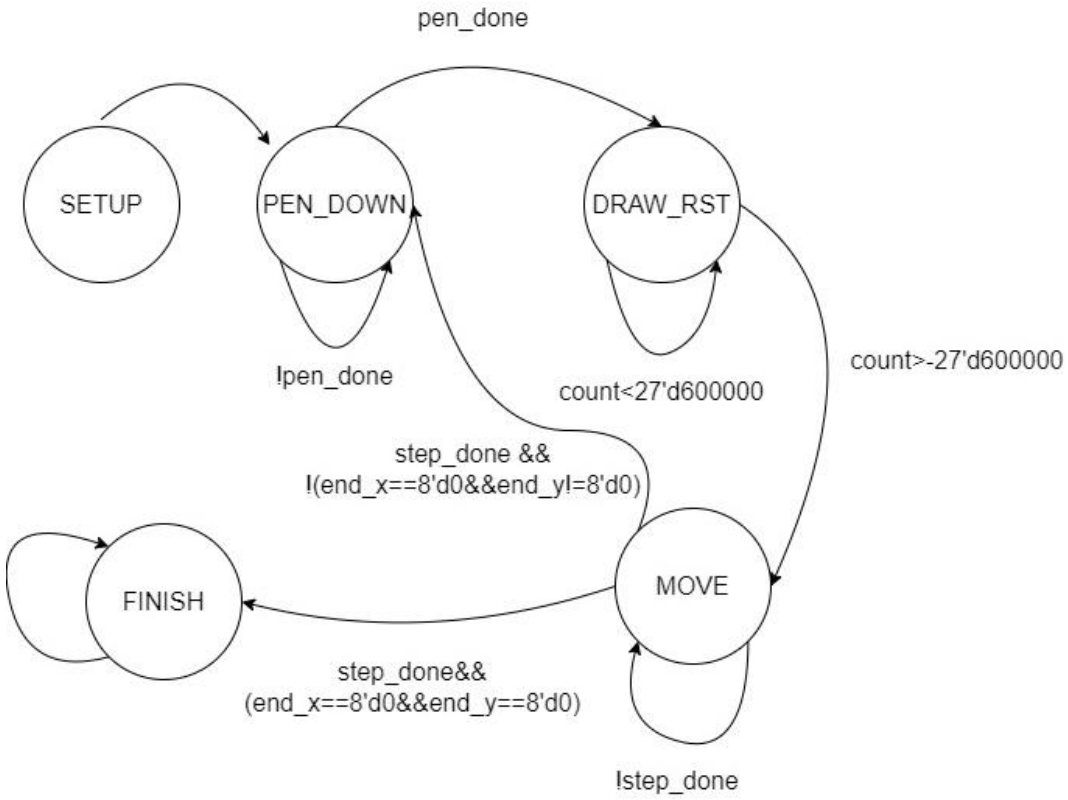
而我要 output 一個 done signal 給更外層的 module，才知道這個筆化已經移動完成了。

Block Diagram


state
enable
rst
MOVE
SETUP
MUX
MOVE
FINISH
MOVE
0
MUX
1
FINISH
FINISH
next_state
DFF
state
cur_x==endx&&cur_y==endy
startx
curx+1
curx-1
endx
>
next_cur_x
curx
2'b00
2'b01
2'b10
next_dirx
dirx
starty
cury+1
cury-1
endy
next_cury
cury
next_diry
diry

- draw_1.v（利用前面的 draw_line.v 處理全部的筆劃、以及筆是否放下）

State Diagram



SETUP: 初始化各個訊號

PEN_DOWN: 在移動筆之前決定是否要將筆放下

DRAW_RST: 要讓 draw_line.v 這個 module reset ，才不會讓 draw_line.v 這個 module 卡在 FINISH state。這也是重複使用 draw_line.v 這個 module 的關鍵。

MOVE: 讓筆畫一直線， draw_line 這個 module 畫完一直線的時候 output done 會等於 1’b1，所以只要判斷 done 是否是 1’b1 知道一筆劃是否結束了。而畫完一個字的時候一定最後會回到原點(0,0)，所以只要知道終點是(0,0)就可以到 FINISH state。

FINISH: 結束所有筆劃的過程，output done == 1’b1。

Coding Detail

Sequential Part

```
parameter DRAW_RST = 3'b010;
parameter MOVE = 3'b011;
parameter FINISH = 3'b100;

always@(posedge clk)begin
    if(rst)begin
        state <= SETUP;
        count <= 27'd0;
        idx <= 4'd0;
        pen_down = 1'b0;
    end
    else begin
        if(enable)begin
            state <= next_state;
            count <= next_count;
            idx <= next_idx;
            pen_down <= next_pen_down;
        end
        else begin
            state <= SETUP;
            count <= 27'd0;
            idx <= 4'd0;
            pen_down = 1'b0;
        end
    end
end
```

next_count 是一個 counter。因為 draw_line 這個 module 的 clock 是 5*10^5 個 fpga clock，而在 reset draw_line 要 trigger positive edge，所以要>=5*10^5 個 fpga clock 維持 reset signal

才能 reset draw_line。idx 是我要知道這個數字的筆畫到哪一個步驟了。而我有 instaniate 一個 module 是只要 input idx 就會 output 這個步驟的起點(start_x, start_y)、終點(end_x, end_y)以及筆是否要放下。

Combinational Part

```
always@(*)begin
    case(state)
        SETUP:begin
            draw_rst = 1'b0;
            draw_enable = 1'b0;
            next_count = 27'd0;
            next_idx = 5'd0;
            next_state = PEN_DOWN;
            next_pen_down = pen_down;
        end
        PEN_DOWN:begin
            draw_rst = 1'b0;
            draw_enable = 1'b0;
            next_idx = idx;
            next_pen_down = pen;
            if(count >= 27'd199999999)begin
                next_count = 27'd0;
                next_state = DRAW_RST;
            end
            else begin
                next_count = count + 27'd1;
                next_state = PEN_DOWN;
            end
        end
```

在 SETUP state，要初始化 next_idx，讓寫字機知道這個字要從第一個筆劃開始。draw_rst 和 draw_enable 是要控制 draw_line 這個 module 的 reset 和 enable，只有在 DRAW_RST 才會設為 1’b1。

在 PEN_DOWN state，是要讓寫字機讓筆放下或抬起，這邊利用我 instatiate 的一個 module(number)來決定是否要讓筆放下或抬起，放下設為 1’b1，抬起設為 1’b0。在這個 state 我有設一個 counter，要先經過 2 秒才會進入下一個 state。因為伺服馬達在 2 秒內一定會將比抬起或放下，因此我大約抓個時間 2 秒當作分界

點，讓伺服馬達有足夠的時間能完成動作。

```
DRAW_RST:begin
    draw_rst = 1'b1;
    draw_enable = 1'b1;
    next_idx = idx;
    next_pen_down = pen_down;
    if(count >= 27'd600000)begin
        next_count = 27'd0;
        next_state = MOVE;
    end
    else begin
        next_count = count + 27'd1;
        next_state = DRAW_RST;
    end
end
MOVE:begin
    draw_rst = 1'b0;
    draw_enable = 1'b1;
    next_count = 27'd0;
    next_pen_down = pen_down;
    if(step_done)begin
        if(end_x == 8'd0 && end_y == 8'd0)begin
            next_idx = 5'd0;
            next_state = FINISH;
        end
```

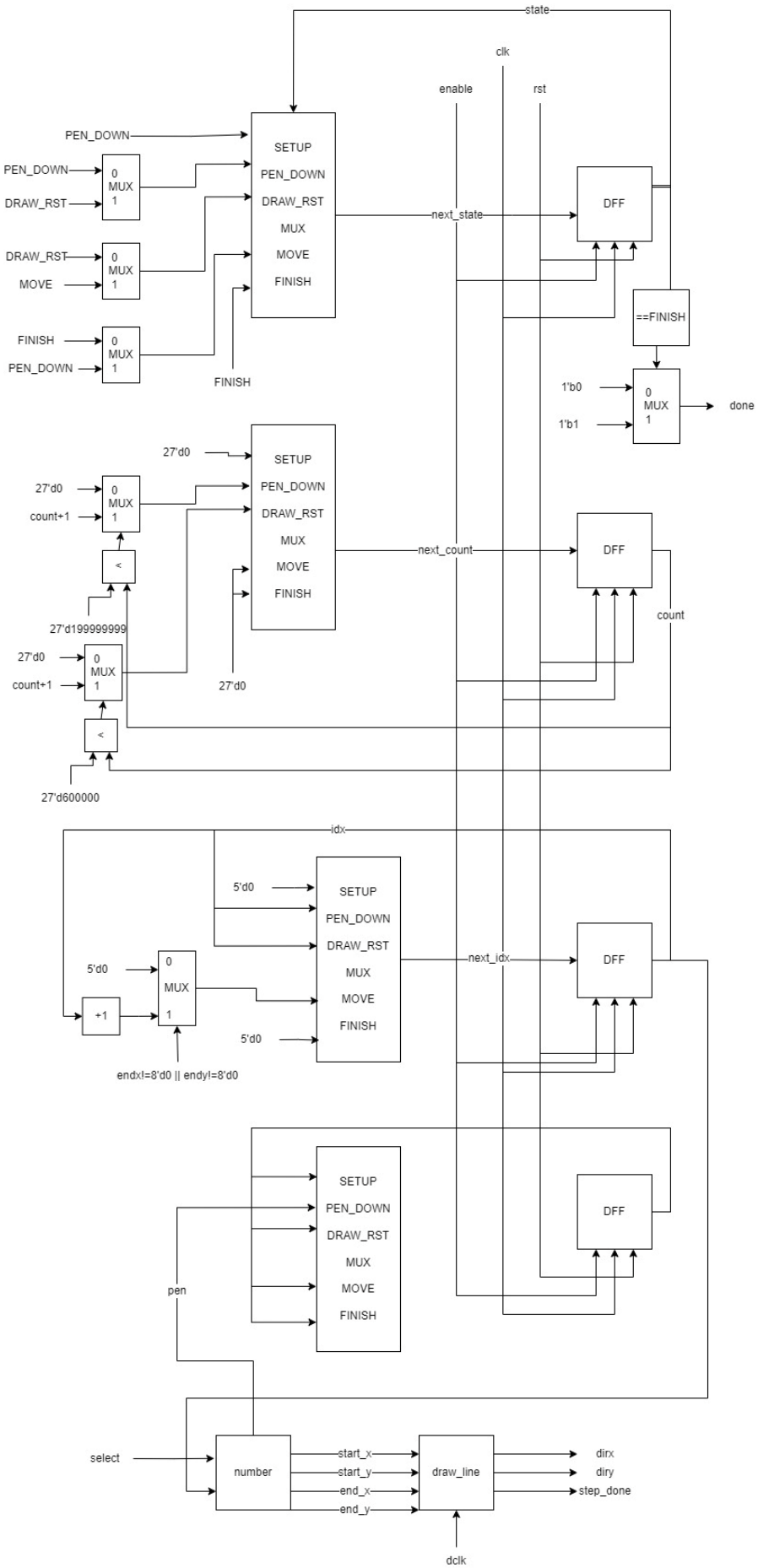
DRAW_RST state 是要 reset draw_line 這個 module。因為我設計 draw_line 這個 module 是如果寫完一筆劃會一直停留在 FINISH state，所以如果要接著寫下一個筆劃的話要 reset 重新回到 draw_line 的 SETUP state。

MOVE state 是要讓 draw_line 這個 module 開始運作(enable = 1’b1)。而我判斷寫完這個字的方法是用 draw_line 是否已經寫完這個筆劃(step_done)以及 end_x end_y 是否都是 8’d0。(前面提過 end_x end_y 是寫這個筆劃的終點位置)。

```
            FINISH:begin
                draw_rst = 1'b0;
                draw_enable = 1'b0;
                next_count = 27'd0;
                next_idx = 5'd0;
                next_pen_down = pen_down;
                next_state = FINISH;
            end
        endcase
    end
    assign done = (state == FINISH) ? 1'b1 : 1'b0;
    draw_line d0(dclk, draw_rst, draw_enable, start_x, start_y, end_x, end_y, dirx, diry, step_done);
    number n0(idx, select, start_x, start_y, end_x, end_y, pen);
endmodule
```

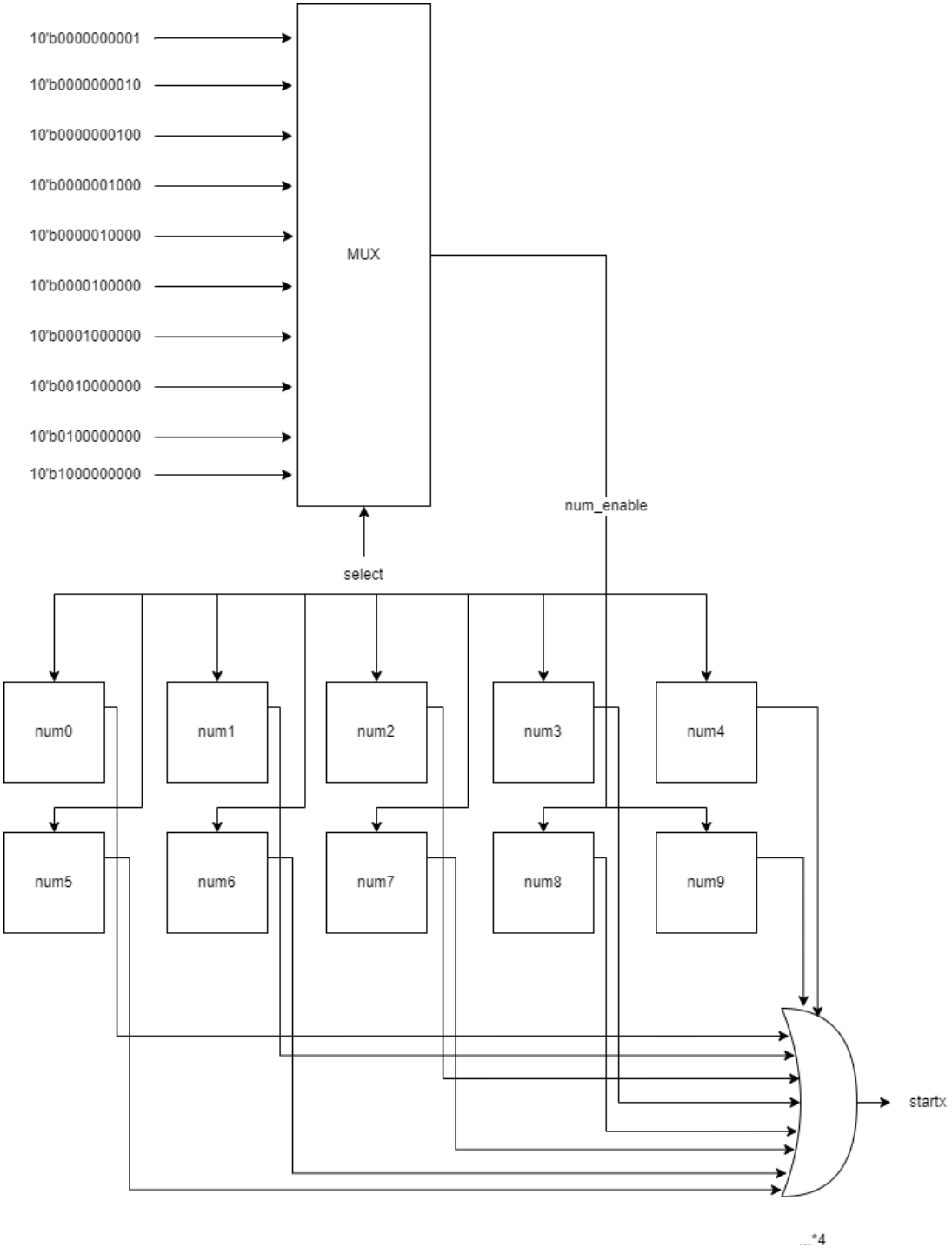
FINISH state 是要讓更外層的 module 知道已經寫完一個字了。所以我設了一個訊號是 done 當在 FINISH 會是 1’b1 不是的話是 1’b0。

Block Diagram


state
clk
enable
rst
PEN_DOWN
SETUP
PEN_DOWN
DRAW_RST
MUX
MOVE
FINISH
PEN_DOWN
DRAW_RST
0
MUX
1
DRAW_RST
MOVE
FINISH
PEN_DOWN
FINISH
next_state
DFF
==FINISH
1'b0
1'b1
done
27'd0
count+1
<
27'd199999999
27'd600000
next_count
count
idx
5'd0
+1
endx!=8'd0 || endy!=8'd0
next_idx
pen
select
number
start_x
start_y
end_x
end_y
draw_line
dirx
diry
step_done
dclk

- number.v

Block Diagram



Coding Details

number 這個 module 的功用是 input 自己決定要畫的字以及自己要的這個數字的第幾個筆劃，就會 output 這個數字在這個筆劃的起點和終點位置。

```
reg [9:0] num_enable;
wire [7:0] tmp_startx [0:9];
wire [7:0] tmp_starty [0:9];
wire [7:0] tmp_endx [0:9];
wire [7:0] tmp_endy [0:9];
wire [9:0] tmp_pen;
always@(*)begin
    num_enable = 10'd0;
    case(select)
        4'd0:num_enable[0] = 1'b1;
        4'd1:num_enable[1] = 1'b1;
        4'd2:num_enable[2] = 1'b1;
        4'd3:num_enable[3] = 1'b1;
        4'd4:num_enable[4] = 1'b1;
        4'd5:num_enable[5] = 1'b1;
        4'd6:num_enable[6] = 1'b1;
        4'd7:num_enable[7] = 1'b1;
        4'd8:num_enable[8] = 1'b1;
        4'd9:num_enable[9] = 1'b1;
    endcase
end
```

在這個 module 中我有 instatiate 10 個 submodule 分別表示 0-9 數字的第幾個筆劃的起點終點位置，所有 submodule 都有 enable 這個 input。所以我上面的 code 代表我 input 哪一個數字，那一個數字的 module enable 會設成 1’b1。這樣就可以正確選擇那個數字在哪一個筆劃的起點終點。

```
    num0 n0(idx, num_enable[0], tmp_startx[0], tmp_starty[0], tmp_endx[0], tmp_endy[0], tmp_pen[0]);
    num1 n1(idx, num_enable[1], tmp_startx[1], tmp_starty[1], tmp_endx[1], tmp_endy[1], tmp_pen[1]);
    num2 n2(idx, num_enable[2], tmp_startx[2], tmp_starty[2], tmp_endx[2], tmp_endy[2], tmp_pen[2]);
    num3 n3(idx, num_enable[3], tmp_startx[3], tmp_starty[3], tmp_endx[3], tmp_endy[3], tmp_pen[3]);
    num4 n4(idx, num_enable[4], tmp_startx[4], tmp_starty[4], tmp_endx[4], tmp_endy[4], tmp_pen[4]);
    num5 n5(idx, num_enable[5], tmp_startx[5], tmp_starty[5], tmp_endx[5], tmp_endy[5], tmp_pen[5]);
    num6 n6(idx, num_enable[6], tmp_startx[6], tmp_starty[6], tmp_endx[6], tmp_endy[6], tmp_pen[6]);
    num7 n7(idx, num_enable[7], tmp_startx[7], tmp_starty[7], tmp_endx[7], tmp_endy[7], tmp_pen[7]);
    num8 n8(idx, num_enable[8], tmp_startx[8], tmp_starty[8], tmp_endx[8], tmp_endy[8], tmp_pen[8]);
    num9 n9(idx, num_enable[9], tmp_startx[9], tmp_starty[9], tmp_endx[9], tmp_endy[9], tmp_pen[9]);
    assign start_x = tmp_startx[0] | tmp_startx[1] | tmp_startx[2] | tmp_startx[3] | tmp_startx[4] | t
    assign start_y = tmp_starty[0] | tmp_starty[1] | tmp_starty[2] | tmp_starty[3] | tmp_starty[4] | t
    assign end_x = tmp_endx[0] | tmp_endx[1] | tmp_endx[2] | tmp_endx[3] | tmp_endx[4] | tmp_endx[5] |
    assign end_y = tmp_endy[0] | tmp_endy[1] | tmp_endy[2] | tmp_endy[3] | tmp_endy[4] | tmp_endy[5] |
    assign pen_down = tmp_pen[0] | tmp_pen[1] | tmp_pen[2] | tmp_pen[3] | tmp_pen[4] | tmp_pen[5] | tm
endmodule
```

上圖的 code 表示我 instatiate 10 個 submodule 分別是 0-9 的起點終點。因為選擇一個數字只有那個數字的 enable 會設成 1’b1 其他會是 1’b0。所以把所有 0-9 output 接 or gate 會得出想要的

結果。

- num1.v(舉數字 1 為例)

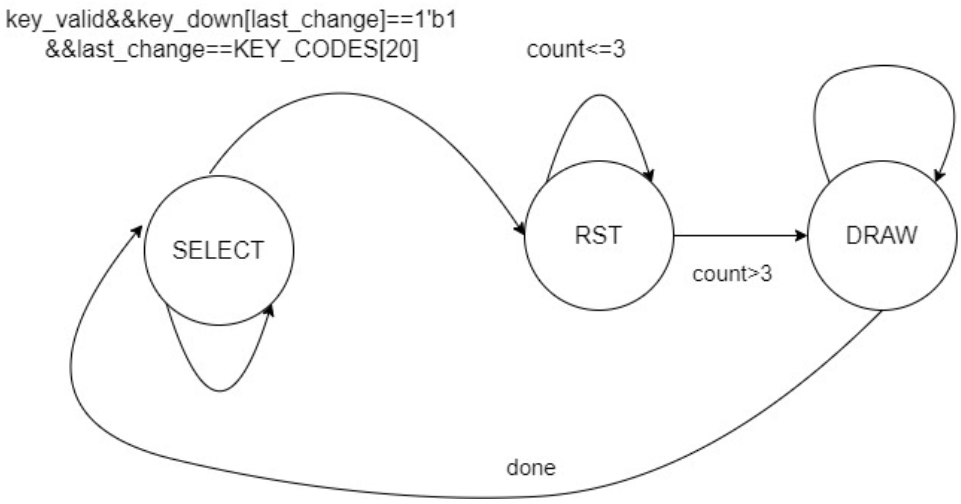
```
        if(enable)begin
            case(idx)
            5'd0:begin
                start_x = 8'd0;
                start_y = 8'd0;
                end_x = 8'd60;
                end_y = 8'd80;
                pen_down = 1'b0;
            end
            5'd1:begin
                start_x = 8'd60;
                start_y = 8'd80;
                end_x = 8'd180;
                end_y = 8'd80;
                pen_down = 1'b1;
            end
            5'd2:begin
                start_x = 8'd180;
                start_y = 8'd80;
                end_x = 8'd0;
                end_y = 8'd0;
                pen_down = 1'b0;
            end
        endcase
        else begin
            start_x = 8'd0;
            start_y = 8'd0;
            end_x = 8'd0;
            end_y = 8'd0;
            pen_down = 1'b0;
        end
    end
endmodule
```

num1 這個 module 是 input 筆劃就會 output 這個筆劃的起點終點以及筆是否要放下。

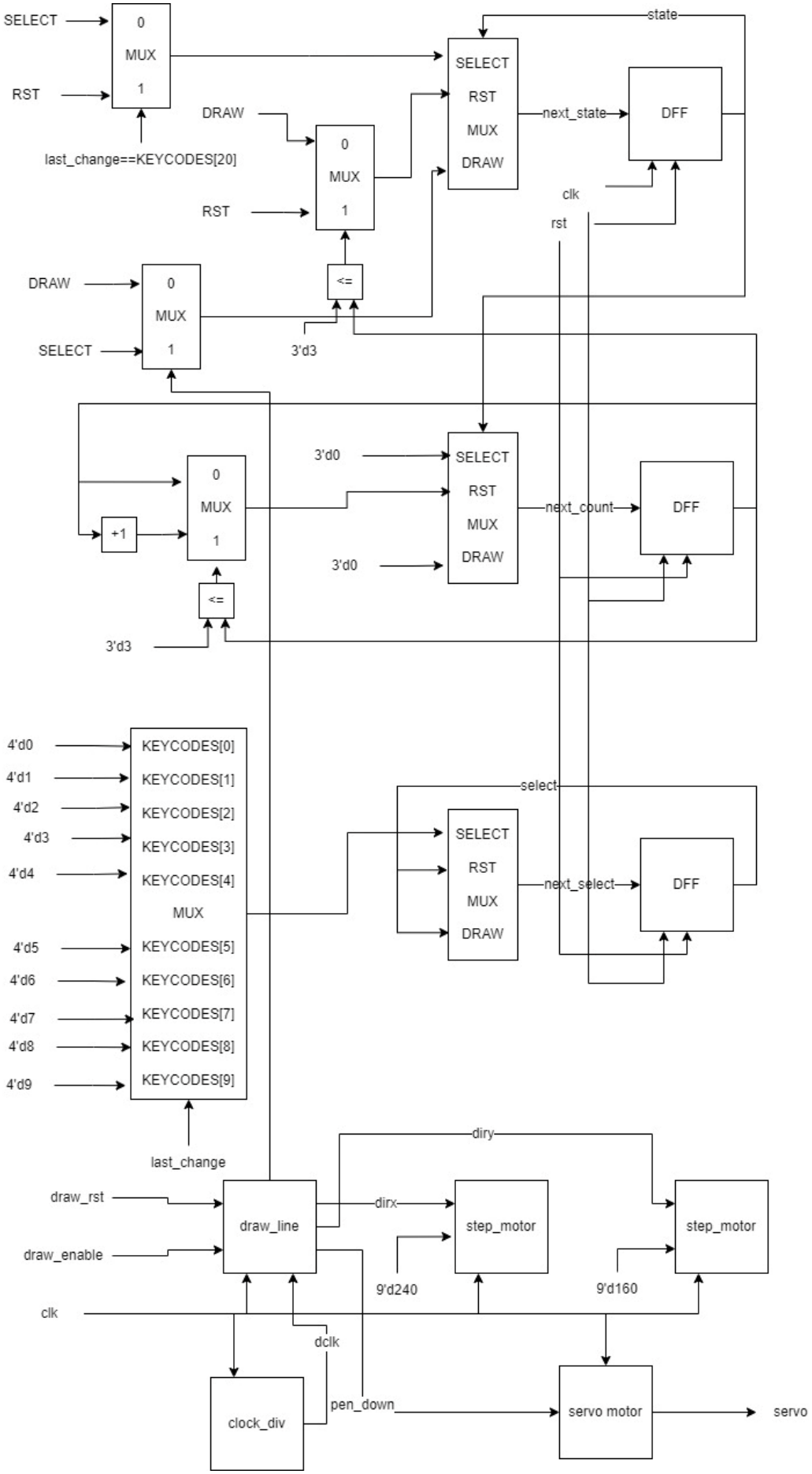
如果要寫更多字只要多加這個 module，更改起點終點，就能寫出更多字了。

- Top.v

State Diagram



Block Diagram


SELECT
0
MUX
1
RST
last_change==KEYCODES[20]
DRAW
0
MUX
1
RST
SELECT
RST
MUX
DRAW
state
next_state
DFF
clk
rst
DRAW
0
MUX
1
SELECT
<=
3'd3
3'd0
SELECT
RST
MUX
DRAW
next_count
DFF
3'd0
0
MUX
1
+1
<=
3'd3
4'd0
4'd1
4'd2
4'd3
4'd4
KEYCODES[0]
KEYCODES[1]
KEYCODES[2]
KEYCODES[3]
KEYCODES[4]
MUX
4'd5
4'd6
4'd7
4'd8
4'd9
KEYCODES[5]
KEYCODES[6]
KEYCODES[7]
KEYCODES[8]
KEYCODES[9]
last_change
select
SELECT
RST
MUX
DRAW
next_select
DFF
diry
draw_rst
draw_enable
clk
draw_line
dirx
step_motor
step_motor
9'd240
9'd160
dclk
clock_div
pen_down
servo motor
servo

Coding Detail

Top module 是整合所有之前的 module 所構成的 module。我希望可以將鍵盤結合，鍵盤按一個數字接著再按下 Enter 鍵就會開始寫。因此我設計 3 個 state，SELECT、RST 和 DRAW。SELECT state 就是按下數字選擇字自己想要寫的字並再按下 ENTER，寫字機就會知道自己想要寫的數字是甚麼了。RST state 就跟我前面說的想法一樣。只要 reset module 就能重複利用這個 module(draw_1)的功能進行寫字。DRAW state 是讓這個 draw_1 module 開始運作，如果 draw_1 module 寫完字了，也就是到 FINISH state，draw_1 module 會 output done = 1’b1 的 signal 給 Top module，這樣會回到 SELECT state。

Sequential Part

```
clock_div dclk0(clk, rst, 26'd500000, dclk);
step_motor x_axis (clk, rst, dirx, 9'd240, step_x);
step_motor y_axis (clk, rst, diry, 9'd160, step_y);
servo_motor write (clk, rst, pen_down, servo);
draw drw(clk, dclk, draw_rst, draw_enable, select, dirx, diry, pen_down, done, draw_state, idx);
//draw_line d(dclk, rst, enable_draw, 0, 0, 200, 100, dirx, diry, done);
SevenSegment s(clk, rst, select, Seven_Segment, AN);
always@(posedge clk)begin
    if(rst)begin
        state <= SELECT;
        select <= 4'd0;
        count <= 3'd0;
    end
    else begin
        select <= next_select;
        state <= next_state;
        count <= next_count;
    end
end
```

count 這個訊號是要在 RST state 維持 3 個 clock cycle 才會進入下一個 state。select 這個訊號是按下鍵盤上的數字鍵後 select 就

會依照自己按下的按鍵來決定是什麼數字(ex. 按下 1 select 會是 1)

Combinational Part

```
always@(*)begin
    case(state)
        SELECT:begin
            next_count = 3'd0;
            draw_enable = 1'b0;
            draw_rst = 1'b0;
            next_select = select;
            if(key_valid && key_down[last_change] == 1'b1)begin
                next_state = SELECT;
                if(last_change == KEY_CODES[0] || last_change == KEY_CODES[10])begin
                    next_select = 4'd0;
                end
                else if(last_change == KEY_CODES[1] || last_change == KEY_CODES[11])begin
                    next_select = 4'd1;
                end
                else if(last_change == KEY_CODES[2] || last_change == KEY_CODES[12])begin
                    next_select = 4'd2;
                end
                else if(last_change == KEY_CODES[3] || last_change == KEY_CODES[13])begin
                    next_select = 4'd3;
                end
                else if(last_change == KEY_CODES[4] || last_change == KEY_CODES[14])begin
                    next_select = 4'd4;
                end
                else if(last_change == KEY_CODES[5] || last_change == KEY_CODES[15])begin
                    next_select = 4'd5;
```

SELECT state 的邏輯就是按下 0-9 選擇數字再按下 ENTER 會進入 RST state。

```
        RST:begin
            draw_enable = 1'b1;
            draw_rst = 1'b1;
            next_select = select;
            if(count <= 3'd3)begin
                next_count = count + 3'd1;
                next_state = RST;
            end
            else begin
                next_count = count;
                next_state = DRAW;
            end
        end
```

RST state 是要將 draw_1 這個 module reset(draw_enable = 1’b1、draw_rst = 1’b1）以便重複利用這個 module，我設定經過 3 個 clock cycle 才會進入下一個 state。

```
        DRAW:begin
            next_count = 3'd0;
            draw_enable = 1'b1;
            draw_rst = 1'b0;
            next_select = select;
            if(done)begin
                next_state = SELECT;
            end
            else begin
                next_state = DRAW;
            end
        end
    endcase
```

DRAW state 顧名思義就是要讓 draw_1 運作寫字(draw_rst = 1’b0、draw_enable = 1’b1)，並在寫完字後(done = 1’b1)回到 SELECT state。

4. 心得:

洪聖祥:這次的 final project 讓我徹底的運用這學期的所學，尤其是 finite state machine 的部分。從畫一條直線到把所有線條整合起來雖然想得蠻久的但還是作出一個完整的作品。但我覺得硬體的部分我實在是不拿手，還好隊友直接 carry 一波把機台直接架起來，讓我能專心在軟體的部分。但完成這個 project 也不是很順利，因為我們前一天要測試的時候居然伺服馬達沒有動起來，我一直以為是我 code 寫錯，但把馬達拿起來才發現馬達得軸心居然跟三秒膠黏在一起。還好最後有借到另一個伺服馬達，不然的話後果不堪設想。最後謝謝助教及教授的指導。

劉奇泓: 本次 final project 其實我們本來想要做的是寫遊戲，但想到我本身 coding 不是很厲害，而且感覺大多數人都能做出功能強

大又好玩的遊戲，因此我想到了可以運用我以前在動機系所學的一些東西來製作機構，加上洪聖祥超強的 coding 能力，軟硬體整合出一個寫字機!遇到的麻煩其中之一是我經過設想畫出來的機構與實際上有些出入，例如 3D 列印的成本問題，或是一些沒有設想到的小細節可能都會成為執行寫字過程中誤差的來源；其二是在研究控制馬達以及供電的部分，除了要確認好自己寫的 code 是不是馬達能接受的東西，也要注意供電以及給電的頻率會不會把馬達弄壞，為此我也跑了幾次電料行買材料問東問西，但這樣的過程也讓我對各種馬達的運作原理有更多認識了。而在 demo 前一個晚上我甚至還不小心把伺服馬達內部的齒輪以及軸心用三秒膠黏住了整個毀掉，還好趕快請動機系的朋友支援才向實驗室借到額外的伺服馬達，否則我們很可能直接放棄了。很感謝 demo 的時候助教們以及教授覺得我們做出了酷酷的東西，這樣的支持也讓我覺得這幾個星期的努力一下變得苦中回甘，最後再次感謝這學期助教們以及教授的細心指導!

5. 分工

劉奇泓:設計寫字機並架設機構 servomotor.v stepmotor.v

洪聖祥:Top.v draw_1.v draw_line.v number.v num(0-9).v sevensegment.v