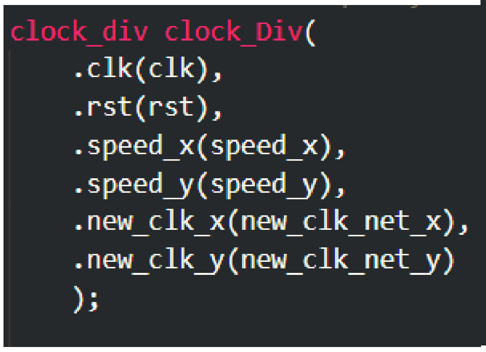
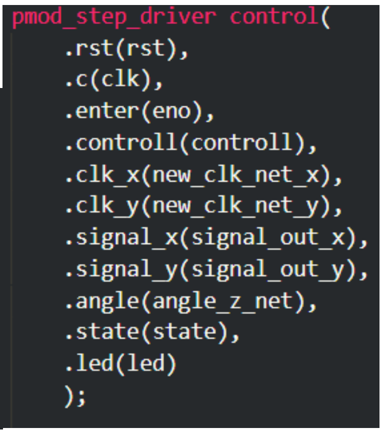
Code 主體

這次的 project 我用到 3 顆步進馬達跟 1 顆伺服馬達，我用一個 top module 包含這兩種馬達的控制，其中伺服馬達需要用到 clock divider 和 pmod_step_driver

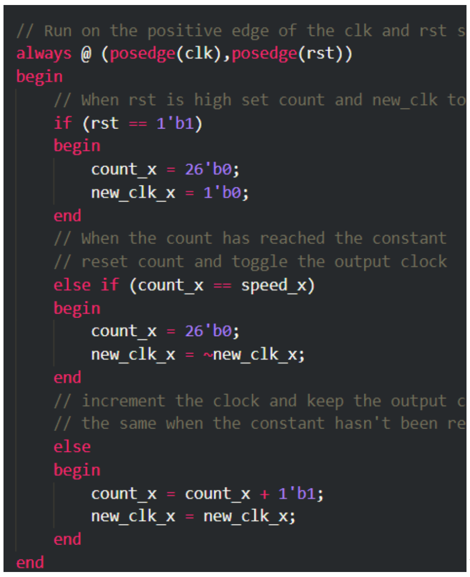


```
clock_div clock_Div(
    .clk(clk),
    .rst(rst),
    .speed_x(speed_x),
    .speed_y(speed_y),
    .new_clk_x(new_clk_net_x),
    .new_clk_y(new_clk_net_y)
    );
```



```
pmod_step_driver control(
    .rst(rst),
    .c(clk),
    .enter(eno),
    .controll(controll),
    .clk_x(new_clk_net_x),
    .clk_y(new_clk_net_y),
    .signal_x(signal_out_x),
    .signal_y(signal_out_y),
    .angle(angle_z_net),
    .state(state),
    .led(led)
    );
```

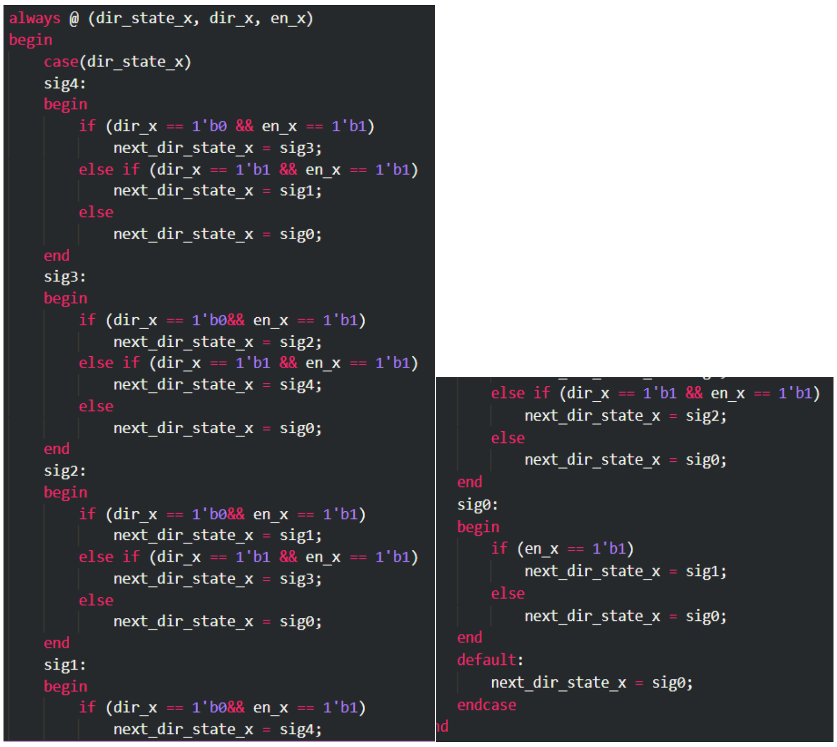
Clock divider 生成一個除頻後的 clock 之後再用類似 seven segment 的方式去控制步進馬達的四個電磁鐵線圈，所以只要控制每個線圈在馬達裡的通電順序就可以控制步進馬達，當 clock 頻率越高轉速越快，相對扭力會縮小，反之則轉速慢扭力大，我的 clock 裡面可以輸入不同的速度(speed_x,speed_y)去控制 clock divider 產生的頻率，進一步控制馬達，以下是 x 方向馬達的 code 跟 y 方向馬的也是一樣的



```
// Run on the positive edge of the clk and rst s
always @ (posedge(clk),posedge(rst))
begin
    // When rst is high set count and new_clk to
    if (rst == 1'b1)
    begin
        count_x = 26'b0;
        new_clk_x = 1'b0;
    end
    // When the count has reached the constant
    // reset count and toggle the output clock
    else if (count_x == speed_x)
    begin
        count_x = 26'b0;
        new_clk_x = ~new_clk_x;
    end
    // increment the clock and keep the output c
    // the same when the constant hasn't been re
    else
    begin
        count_x = count_x + 1'b1;
        new_clk_x = new_clk_x;
    end
end
```

當 counter = speed_x 就改變訊號，也就是產生一個周期為 2*speed_x 的 clk，當我的寫字機需要產生斜率非 1 或-1 的斜直線時就可以透過控制 x 軸和 y 軸馬達的速度來調整角度，字型也會比較漂亮

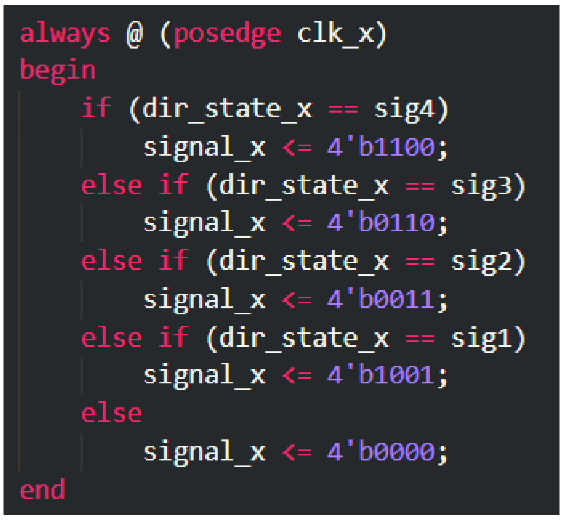
再來是我的 pmod_step_driver 是主要控制寫字機的 module，也是驅動步進馬達並控制馬達正轉或逆轉的主要 module



```
always @ (dir_state_x, dir_x, en_x)
begin
    case(dir_state_x)
    sig4:
    begin
        if (dir_x == 1'b0 && en_x == 1'b1)
            next_dir_state_x = sig3;
        else if (dir_x == 1'b1 && en_x == 1'b1)
            next_dir_state_x = sig1;
        else
            next_dir_state_x = sig0;
    end
    sig3:
    begin
        if (dir_x == 1'b0&& en_x == 1'b1)
            next_dir_state_x = sig2;
        else if (dir_x == 1'b1 && en_x == 1'b1)
            next_dir_state_x = sig4;
        else
            next_dir_state_x = sig0;
    end
    sig2:
    begin
        if (dir_x == 1'b0&& en_x == 1'b1)
            next_dir_state_x = sig1;
        else if (dir_x == 1'b1 && en_x == 1'b1)
            next_dir_state_x = sig3;
        else
            next_dir_state_x = sig0;
    end
    sig1:
    begin
        if (dir_x == 1'b0&& en_x == 1'b1)
            next_dir_state_x = sig4;
```

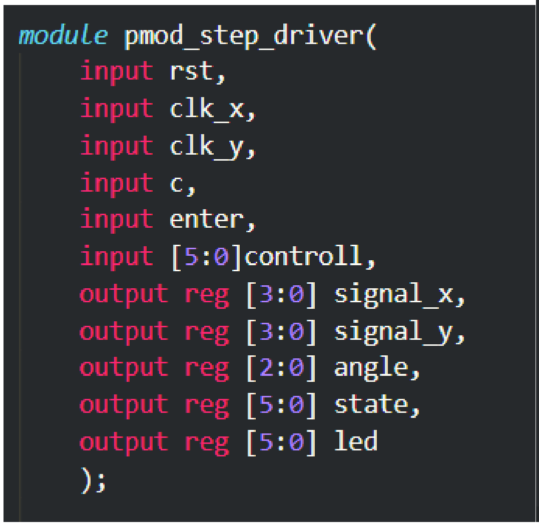
```
        else if (dir_x == 1'b1 && en_x == 1'b1)
            next_dir_state_x = sig2;
        else
            next_dir_state_x = sig0;
    end
    sig0:
    begin
        if (en_x == 1'b1)
            next_dir_state_x = sig1;
        else
            next_dir_state_x = sig0;
    end
    default:
        next_dir_state_x = sig0;
    endcase
nd
```

上圖是控制步進馬達四相線圈的 state machine，當我的 dir = 1 時 state 的變化順序是從 sig0->sig1->sig3->sig4->sig0，反之 state 則反方向變化，所以我可以透過控制正逆轉來實現寫字功能，而當 en = 1，馬達可以轉動，en = 0 時馬達停止

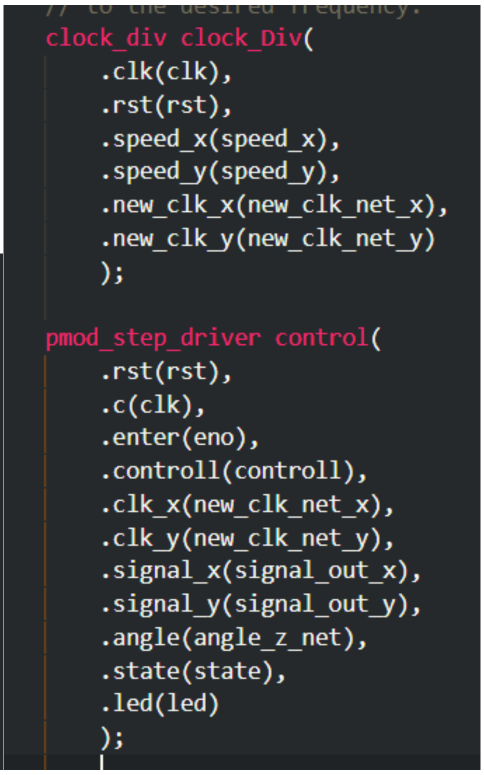


```
always @ (posedge clk_x)
begin
    if (dir_state_x == sig4)
        signal_x <= 4'b1100;
    else if (dir_state_x == sig3)
        signal_x <= 4'b0110;
    else if (dir_state_x == sig2)
        signal_x <= 4'b0011;
    else if (dir_state_x == sig1)
        signal_x <= 4'b1001;
    else
        signal_x <= 4'b0000;
end
```

圖中的 4bit signal 就是用來控制馬達的 output，類似於 seven segment，透過連續變換來驅動馬達，我採用的是 2 相驅動法，也就是一次讓兩個線圈通電，因為我只用板子的 3.3v 來供電，所以用單向驅動馬達發現扭力不足，才使用 2 相驅動，這樣可以加強扭力，但缺點是耗電量大，而 clk_x 就是驅動馬達的 clk，透過 clk divider 調整輸入的 clk 頻率來實現控制馬達的速度，就可以讓字型更好看，在 pmod_step_driver 有兩個 input: clk_x 跟 clk_y，分別是用來控制 x,y 軸馬達，而輸入的訊號由 top module 的 clock divider 送出

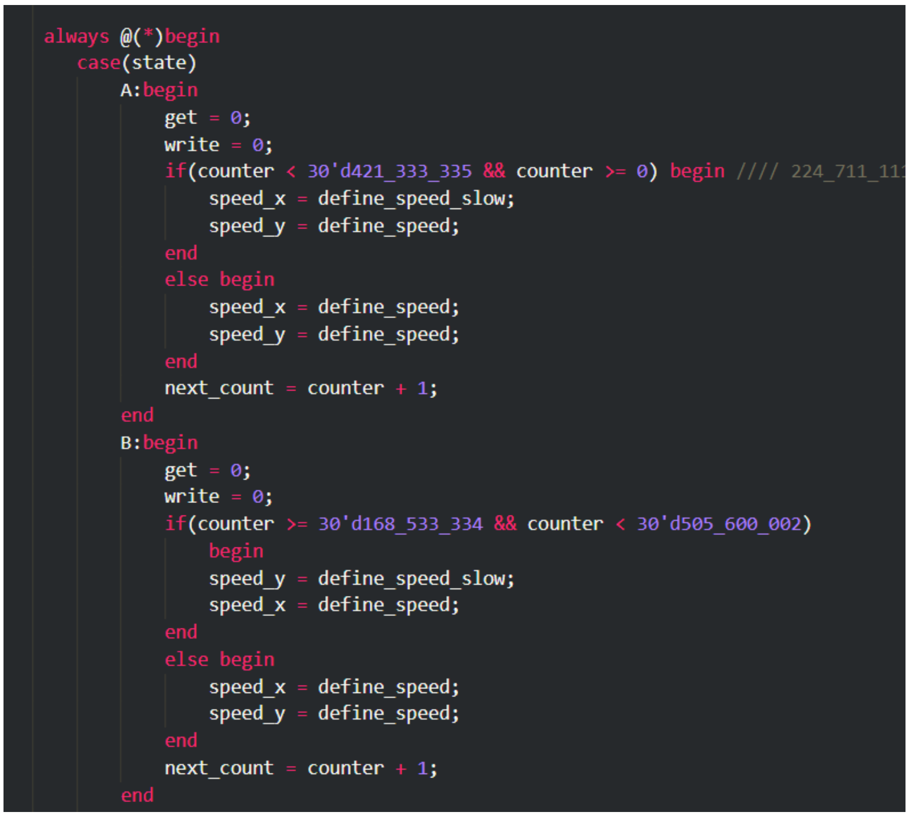


```verilog
module pmod_step_driver(
    input rst,
    input clk_x,
    input clk_y,
    input c,
    input enter,
    input [5:0]controll,
    output reg [3:0] signal_x,
    output reg [3:0] signal_y,
    output reg [2:0] angle,
    output reg [5:0] state,
    output reg [5:0] led
    );
```

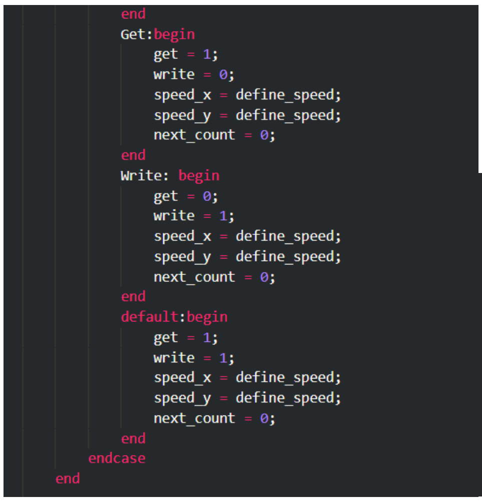


```verilog
// to the desired frequency.
clock_div clock_Div(
    .clk(clk),
    .rst(rst),
    .speed_x(speed_x),
    .speed_y(speed_y),
    .new_clk_x(new_clk_net_x),
    .new_clk_y(new_clk_net_y)
    );

pmod_step_driver control(
    .rst(rst),
    .c(clk),
    .enter(eno),
    .controll(controll),
    .clk_x(new_clk_net_x),
    .clk_y(new_clk_net_y),
    .signal_x(signal_out_x),
    .signal_y(signal_out_y),
    .angle(angle_z_net),
    .state(state),
    .led(led)
    );
```

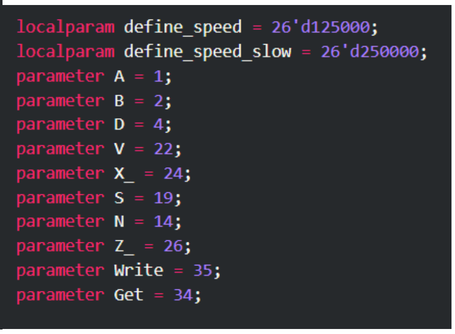
而 clock divider 輸出的 clk 由 speed 決定，speed 又根據我從 pmod_step_driver 送出的目前的 state 來決定馬達什麼時候該用正常的速度，甚麼時候該用漫一點的速度，下圖就是我控制馬達速度的方式



```verilog
always @(*)begin
    case(state)
        A:begin
            get = 0;
            write = 0;
            if(counter < 30'd421_333_335 && counter >= 0) begin //// 224_711_11
                speed_x = define_speed_slow;
                speed_y = define_speed;
            end
            else begin
                speed_x = define_speed;
                speed_y = define_speed;
            end
            next_count = counter + 1;
        end
        B:begin
            get = 0;
            write = 0;
            if(counter >= 30'd168_533_334 && counter < 30'd505_600_002)
                begin
                speed_y = define_speed_slow;
                speed_x = define_speed;
            end
            else begin
                speed_x = define_speed;
                speed_y = define_speed;
            end
            next_count = counter + 1;
        end
```



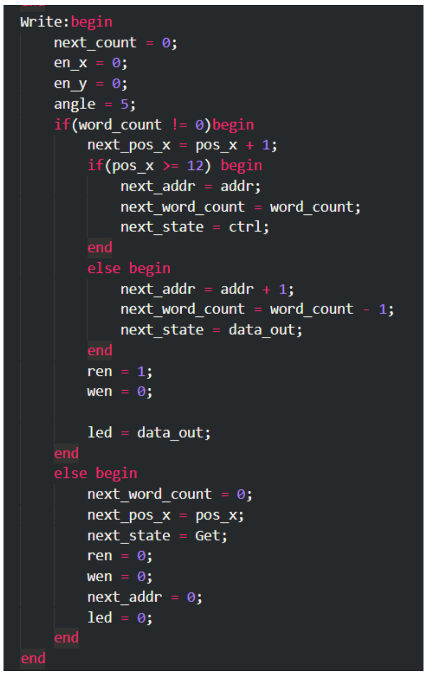
```verilog
            end
            Get:begin
                get = 1;
                write = 0;
                speed_x = define_speed;
                speed_y = define_speed;
                next_count = 0;
            end
            Write: begin
                get = 0;
                write = 1;
                speed_x = define_speed;
                speed_y = define_speed;
                next_count = 0;
            end
            default:begin
                get = 1;
                write = 1;
                speed_x = define_speed;
                speed_y = define_speed;
                next_count = 0;
            end
        endcase
    end
```



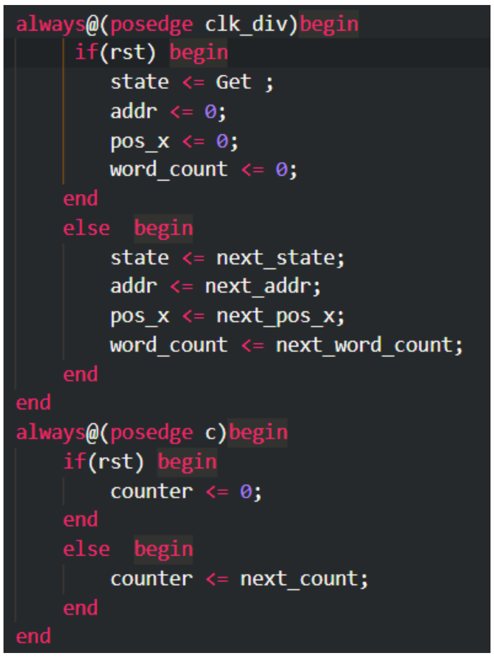
```verilog
localparam define_speed = 26'd125000;
localparam define_speed_slow = 26'd250000;
parameter A = 1;
parameter B = 2;
parameter D = 4;
parameter V = 22;
parameter X_ = 24;
parameter S = 19;
parameter N = 14;
parameter Z_ = 26;
parameter Write = 35;
parameter Get = 34;
```

為了字型的美觀，我必須畫出斜率不為 1 或-1 的斜線，所以我用一個 slow speed，速度為 speed 的 1/2，代表我可以畫出斜率為 1/2, -1/2, 2, -2 的線，需要用到斜率不同的線的字為 A,B,D,V,X,S,N,Z 所以我依照 pmod_step_driver 傳出目前的 state 在用 top module 的 always block 來決定哪個筆畫應該用速度較慢的 speed，哪個該用快一點

然後是我的寫字主要部分，我的寫字機可以寫出 26 個英文字母跟數字還有底線和空白鍵，所以我用了 38 個 state 去處理我的主要輸入法，每一個字的筆劃軌跡都是透過我的 x,y 軸馬達來寫出來的，我用比較低頻率的 clock 來驅動 state，在用 system clock 來做 counter，counter 的功用是來控制馬達轉動的角度，我先用估算的方式算出馬達轉動一公分的距離大概是經過 112355556 個 clock cycle，接下來我就依這這個標準去控制我的字型跟筆畫，write state 還有另一個功能，就是當我輸入的字數超過 12 個字時就會進入自動換行的 state，這裡我用 pos_x 來記錄 x 方向上已經有幾個字，如果超過 12 個字則先保存目前的 address，等到換行完之後再繼續寫，並且將 pox_x 歸零

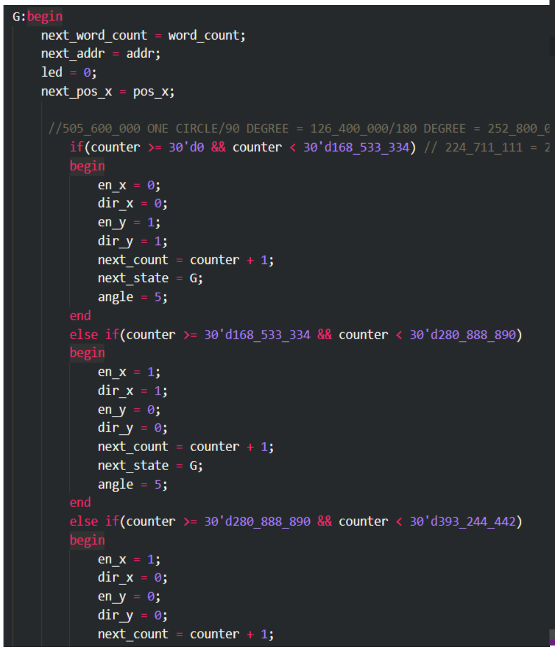


```verilog
Write:begin
    next_count = 0;
    en_x = 0;
    en_y = 0;
    angle = 5;
    if(word_count != 0)begin
        next_pos_x = pos_x + 1;
        if(pos_x >= 12) begin
            next_addr = addr;
            next_word_count = word_count;
            next_state = ctrl;
        end
        else begin
            next_addr = addr + 1;
            next_word_count = word_count - 1;
            next_state = data_out;
        end
        ren = 1;
        wen = 0;

        led = data_out;
    end
    else begin
        next_word_count = 0;
        next_pos_x = pos_x;
        next_state = Get;
        ren = 0;
        wen = 0;
        next_addr = 0;
        led = 0;
    end
end
```

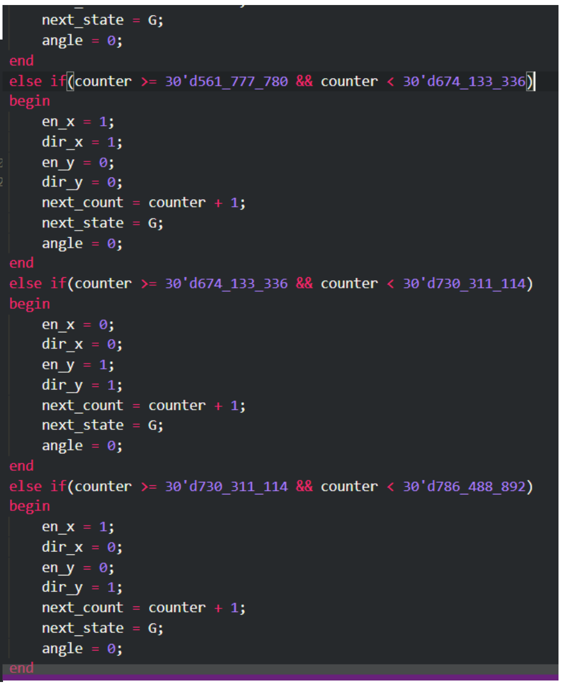


```verilog
always@(posedge clk_div)begin
    if(rst) begin
        state <= Get ;
        addr <= 0;
        pos_x <= 0;
        word_count <= 0;
    end
    else  begin
        state <= next_state;
        addr <= next_addr;
        pos_x <= next_pos_x;
        word_count <= next_word_count;
    end
end
always@(posedge c)begin
    if(rst) begin
        counter <= 0;
    end
    else  begin
        counter <= next_count;
    end
end
```

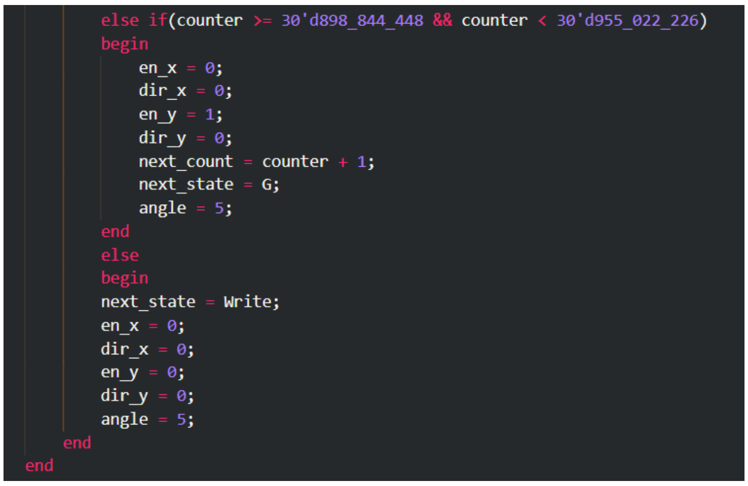
下圖是字母 E 的 code，每個字母都是用類似的方法完成一個字的 state



```verilog
G:begin
    next_word_count = word_count;
    next_addr = addr;
    led = 0;
    next_pos_x = pos_x;

     //505_600_000 ONE CIRCLE/90 DEGREE = 126_400_000/180 DEGREE = 252_800_0
        if(counter >= 30'd0 && counter < 30'd168_533_334) // 224_711_111 = 2
        begin
            en_x = 0;
            dir_x = 0;
            en_y = 1;
            dir_y = 1;
            next_count = counter + 1;
            next_state = G;
            angle = 5;
        end
        else if(counter >= 30'd168_533_334 && counter < 30'd280_888_890)
        begin
            en_x = 1;
            dir_x = 1;
            en_y = 0;
            dir_y = 0;
            next_count = counter + 1;
            next_state = G;
            angle = 5;
        end
        else if(counter >= 30'd280_888_890 && counter < 30'd393_244_442)
        begin
            en_x = 1;
            dir_x = 0;
            en_y = 0;
            dir_y = 0;
            next_count = counter + 1;
```

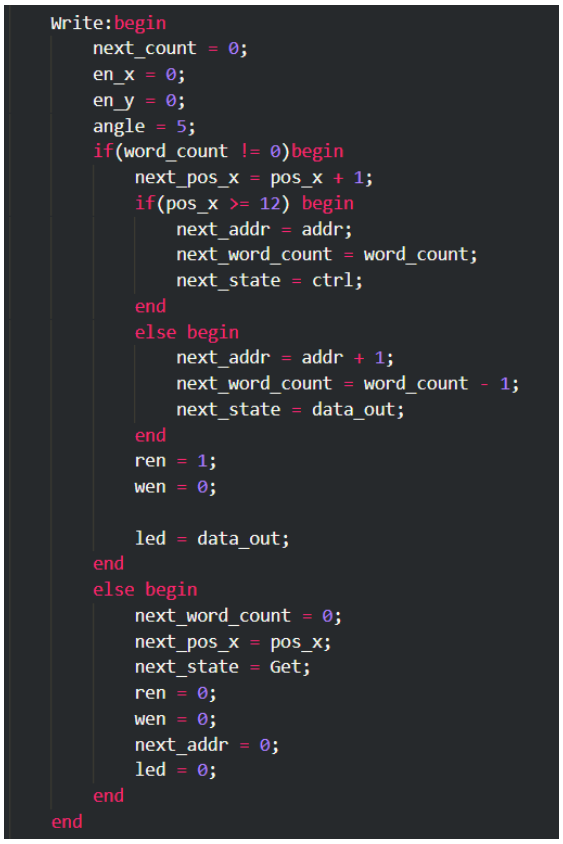


```verilog
    next_state = G;
    angle = 0;
end
else if(counter >= 30'd561_777_780 && counter < 30'd674_133_336)
begin
    en_x = 1;
    dir_x = 1;
    en_y = 0;
    dir_y = 0;
    next_count = counter + 1;
    next_state = G;
    angle = 0;
end
else if(counter >= 30'd674_133_336 && counter < 30'd730_311_114)
begin
    en_x = 0;
    dir_x = 0;
    en_y = 1;
    dir_y = 1;
    next_count = counter + 1;
    next_state = G;
    angle = 0;
end
else if(counter >= 30'd730_311_114 && counter < 30'd786_488_892)
begin
    en_x = 1;
    dir_x = 0;
    en_y = 0;
    dir_y = 1;
    next_count = counter + 1;
    next_state = G;
    angle = 0;
end
```



```
        else if(counter >= 30'd898_844_448 && counter < 30'd955_022_226)
        begin
            en_x = 0;
            dir_x = 0;
            en_y = 1;
            dir_y = 0;
            next_count = counter + 1;
            next_state = G;
            angle = 5;
        end
        else
        begin
        next_state = Write;
        en_x = 0;
        dir_x = 0;
        en_y = 0;
        dir_y = 0;
        angle = 5;
    end
end
```

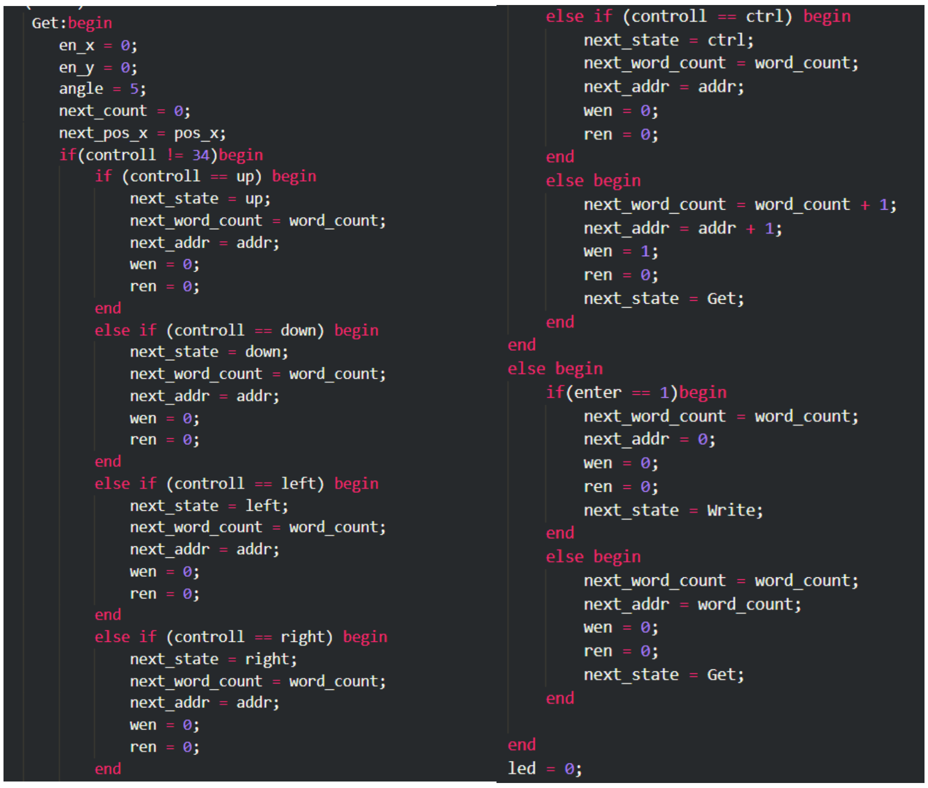
每一個 if 或是 else if 都是一個筆畫或是動作，而 angle 是用來控制 z 軸，可以做出下筆和提筆的動作，用伺服馬達驅動，當我要下筆的時候 angle = 0,提筆的時候 angle = 5，而每一個 if 或是 else if 都是用 counter 來計算筆畫或是移動的長度，有些字沒辦法一筆畫完成，所以就需要提筆移動之後再下筆，所以會有很多 if else，為了減少 code 的長度，我盡量讓字母可以一筆畫完成，或是盡可能減少移動的距離，所以寫字的筆畫會跟一般手寫的筆畫不太一樣，而且每次寫完字之後都會停在字的右下角而且空出 0.5 公分的距離，這樣不管字做什麼排列組合都可以接得起來，每次下筆都從上一個字的右下角開始寫，寫完之後就會回到 write state，也就是輸入字母的 state，下圖為 write state



```
Write:begin
    next_count = 0;
    en_x = 0;
    en_y = 0;
    angle = 5;
    if(word_count != 0)begin
        next_pos_x = pos_x + 1;
        if(pos_x >= 12) begin
            next_addr = addr;
            next_word_count = word_count;
            next_state = ctrl;
        end
        else begin
            next_addr = addr + 1;
            next_word_count = word_count - 1;
            next_state = data_out;
        end
        ren = 1;
        wen = 0;

        led = data_out;
    end
    else begin
        next_word_count = 0;
        next_pos_x = pos_x;
        next_state = Get;
        ren = 0;
        wen = 0;
        next_addr = 0;
        led = 0;
    end
end
```

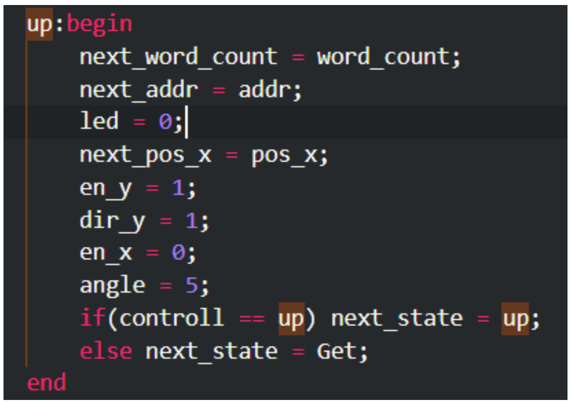
這裡的 write state 負責連續寫字，我用 lab3 寫過的 memory 來當作鍵盤輸

入的 buffer，這樣就不需要輸入一個字寫一個字，可以一次打完再一次寫完，實現全自動寫字的功能，當我輸入完要打的字進 memory 按下 enter 之後，進入 write state，每次進入 write state 都會從 memory 取一個字的 state，並且讓輸入進 memory 的 adderess 加一，計算輸入字數的 counter 減一，當 counter = 0，機器就會停止重新等待輸入，進入到寫字的 state 之後等到字寫完就會回到 write state 這時候 address 已經加一，所以又會再寫下一個輸入的字，重複循環直到 counter 歸零，下圖為 Get state



```
Get:begin
    en_x = 0;
    en_y = 0;
    angle = 5;
    next_count = 0;
    next_pos_x = pos_x;
    if(controll != 34)begin
        if (controll == up) begin
            next_state = up;
            next_word_count = word_count;
            next_addr = addr;
            wen = 0;
            ren = 0;
        end
        else if (controll == down) begin
            next_state = down;
            next_word_count = word_count;
            next_addr = addr;
            wen = 0;
            ren = 0;
        end
        else if (controll == left) begin
            next_state = left;
            next_word_count = word_count;
            next_addr = addr;
            wen = 0;
            ren = 0;
        end
        else if (controll == right) begin
            next_state = right;
            next_word_count = word_count;
            next_addr = addr;
            wen = 0;
            ren = 0;
        end
        else if (controll == ctrl) begin
            next_state = ctrl;
            next_word_count = word_count;
            next_addr = addr;
            wen = 0;
            ren = 0;
        end
        else begin
            next_word_count = word_count + 1;
            next_addr = addr + 1;
            wen = 1;
            ren = 0;
            next_state = Get;
        end
    end
    else begin
        if(enter == 1)begin
            next_word_count = word_count;
            next_addr = 0;
            wen = 0;
            ren = 0;
            next_state = Write;
        end
        else begin
            next_word_count = word_count;
            next_addr = word_count;
            wen = 0;
            ren = 0;
            next_state = Get;
        end

    end
    led = 0;
```

在 Get state 裡面我用 controll 來表示輸入的字元，之後會解釋 controll 的來源，這裡我把上下左右鍵當作控制 x,y 軸馬達的按鍵，如果我的寫字機在寫字中偏離水平線我就可以控制上下左右鍵作微調，下圖是讓 y 軸往上的 up state，按著方向鍵不放，筆尖就會往該方向移動直到放開按鍵，如果是按下 ctrl，就會做換行的動作，換行會根據我儲存的 pos_x，也就是計算 X 方向已經寫了多少個字，這樣才可以知道換行的時候要往後退多少格再換行

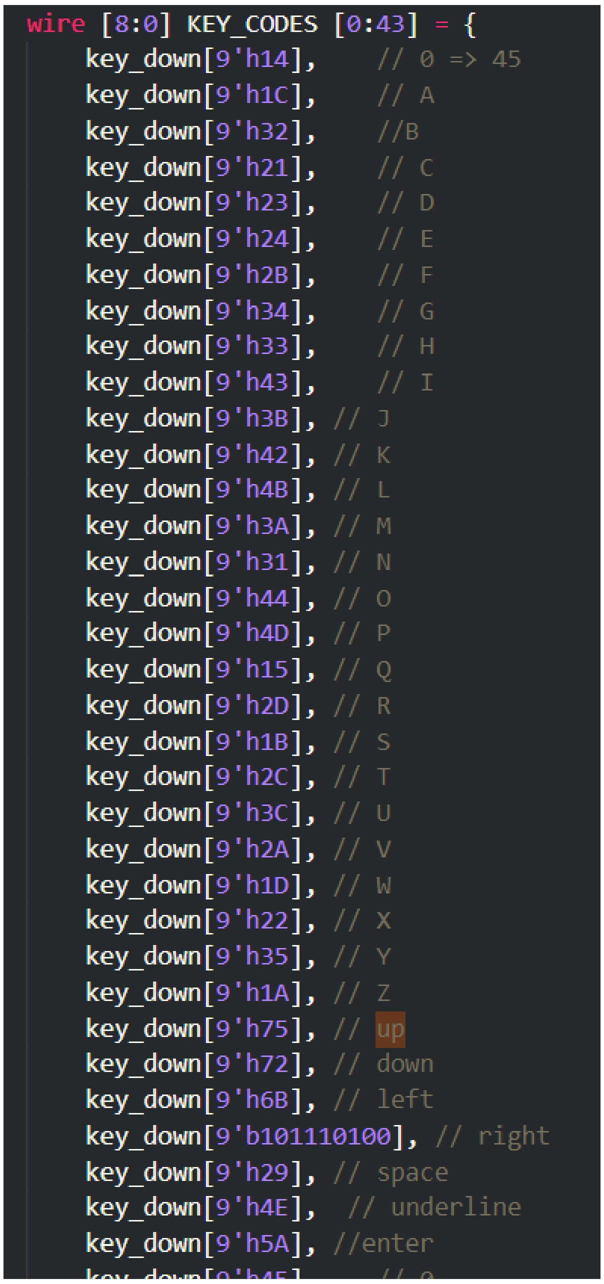


```
up:begin
    next_word_count = word_count;
    next_addr = addr;
    led = 0;
    next_pos_x = pos_x;
    en_y = 1;
    dir_y = 1;
    en_x = 0;
    angle = 5;
    if(controll == up) next_state = up;
    else next_state = Get;
end
```

如果在 Get state 裡面輸入的是字母或數字，那就會存進 memory 裡面，並且 adderess 加一，計算輸入字數的 counter 加一，如果按下 enter，address 就會歸零進到 write state 開始寫字

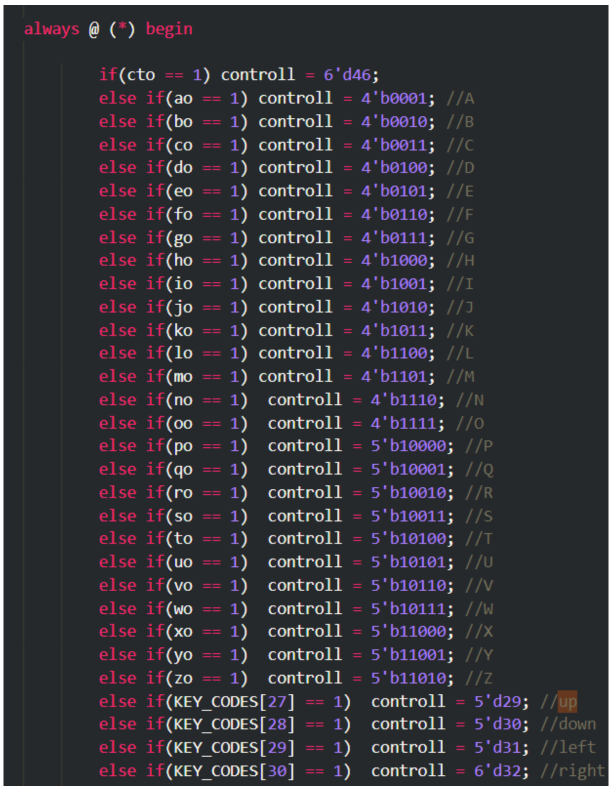
接下來是鍵盤輸入，我用一個二維陣列記錄每一個 keycode，當按下相應按鍵，就會有對應的 controll，然後再加上 onepulse

二維陣列:



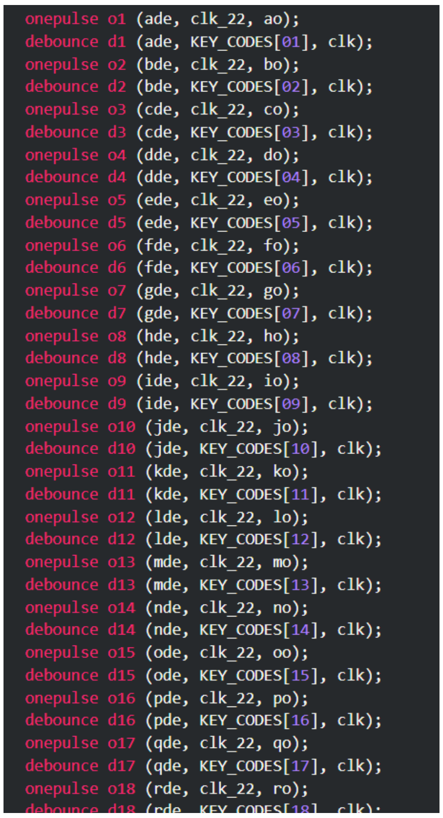
```
wire [8:0] KEY_CODES [0:43] = {
    key_down[9'h14],    // 0 => 45
    key_down[9'h1C],    // A
    key_down[9'h32],    //B
    key_down[9'h21],    // C
    key_down[9'h23],    // D
    key_down[9'h24],    // E
    key_down[9'h2B],    // F
    key_down[9'h34],    // G
    key_down[9'h33],    // H
    key_down[9'h43],    // I
    key_down[9'h3B], // J
    key_down[9'h42], // K
    key_down[9'h4B], // L
    key_down[9'h3A], // M
    key_down[9'h31], // N
    key_down[9'h44], // O
    key_down[9'h4D], // P
    key_down[9'h15], // Q
    key_down[9'h2D], // R
    key_down[9'h1B], // S
    key_down[9'h2C], // T
    key_down[9'h3C], // U
    key_down[9'h2A], // V
    key_down[9'h1D], // W
    key_down[9'h22], // X
    key_down[9'h35], // Y
    key_down[9'h1A], // Z
    key_down[9'h75], // up
    key_down[9'h72], // down
    key_down[9'h6B], // left
    key_down[9'b101110100], // right
    key_down[9'h29], // space
    key_down[9'h4E],  // underline
    key_down[9'h5A], //enter
    key_down[9'h45],    // 0
```

Controll:



```
always @ (*) begin

        if(cto == 1) controll = 6'd46;
        else if(ao == 1) controll = 4'b0001; //A
        else if(bo == 1) controll = 4'b0010; //B
        else if(co == 1) controll = 4'b0011; //C
        else if(do == 1) controll = 4'b0100; //D
        else if(eo == 1) controll = 4'b0101; //E
        else if(fo == 1) controll = 4'b0110; //F
        else if(go == 1) controll = 4'b0111; //G
        else if(ho == 1) controll = 4'b1000; //H
        else if(io == 1) controll = 4'b1001; //I
        else if(jo == 1) controll = 4'b1010; //J
        else if(ko == 1) controll = 4'b1011; //K
        else if(lo == 1) controll = 4'b1100; //L
        else if(mo == 1) controll = 4'b1101; //M
        else if(no == 1)  controll = 4'b1110; //N
        else if(oo == 1)  controll = 4'b1111; //O
        else if(po == 1)  controll = 5'b10000; //P
        else if(qo == 1)  controll = 5'b10001; //Q
        else if(ro == 1)  controll = 5'b10010; //R
        else if(so == 1)  controll = 5'b10011; //S
        else if(to == 1)  controll = 5'b10100; //T
        else if(uo == 1)  controll = 5'b10101; //U
        else if(vo == 1)  controll = 5'b10110; //V
        else if(wo == 1)  controll = 5'b10111; //W
        else if(xo == 1)  controll = 5'b11000; //X
        else if(yo == 1)  controll = 5'b11001; //Y
        else if(zo == 1)  controll = 5'b11010; //Z
        else if(KEY_CODES[27] == 1)  controll = 5'd29; //up
        else if(KEY_CODES[28] == 1)  controll = 5'd30; //down
        else if(KEY_CODES[29] == 1)  controll = 5'd31; //left
        else if(KEY_CODES[30] == 1)  controll = 6'd32; //right
```

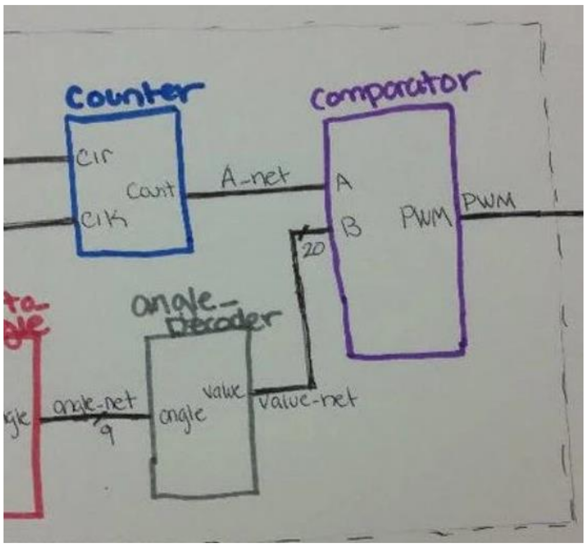
Onepulse:



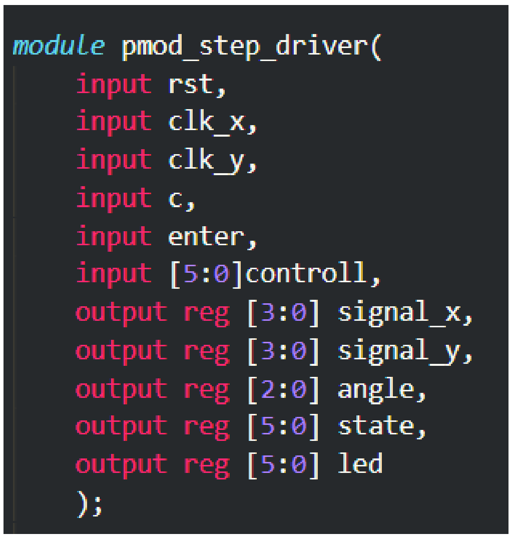
```
onepulse o1 (ade, clk_22, ao);
debounce d1 (ade, KEY_CODES[01], clk);
onepulse o2 (bde, clk_22, bo);
debounce d2 (bde, KEY_CODES[02], clk);
onepulse o3 (cde, clk_22, co);
debounce d3 (cde, KEY_CODES[03], clk);
onepulse o4 (dde, clk_22, do);
debounce d4 (dde, KEY_CODES[04], clk);
onepulse o5 (ede, clk_22, eo);
debounce d5 (ede, KEY_CODES[05], clk);
onepulse o6 (fde, clk_22, fo);
debounce d6 (fde, KEY_CODES[06], clk);
onepulse o7 (gde, clk_22, go);
debounce d7 (gde, KEY_CODES[07], clk);
onepulse o8 (hde, clk_22, ho);
debounce d8 (hde, KEY_CODES[08], clk);
onepulse o9 (ide, clk_22, io);
debounce d9 (ide, KEY_CODES[09], clk);
onepulse o10 (jde, clk_22, jo);
debounce d10 (jde, KEY_CODES[10], clk);
onepulse o11 (kde, clk_22, ko);
debounce d11 (kde, KEY_CODES[11], clk);
onepulse o12 (lde, clk_22, lo);
debounce d12 (lde, KEY_CODES[12], clk);
onepulse o13 (mde, clk_22, mo);
debounce d13 (mde, KEY_CODES[13], clk);
onepulse o14 (nde, clk_22, no);
debounce d14 (nde, KEY_CODES[14], clk);
onepulse o15 (ode, clk_22, oo);
debounce d15 (ode, KEY_CODES[15], clk);
onepulse o16 (pde, clk_22, po);
debounce d16 (pde, KEY_CODES[16], clk);
onepulse o17 (qde, clk_22, qo);
debounce d17 (qde, KEY_CODES[17], clk);
onepulse o18 (rde, clk_22, ro);
debounce d18 (rde, KEY_CODES[18], clk);
```

我的馬達除了步進馬達之外，還有伺服馬達來控制 Z 軸方向，控制伺服馬達需要一個 angle decorder 用來把角度用公式換算成一個 value，還需要一個

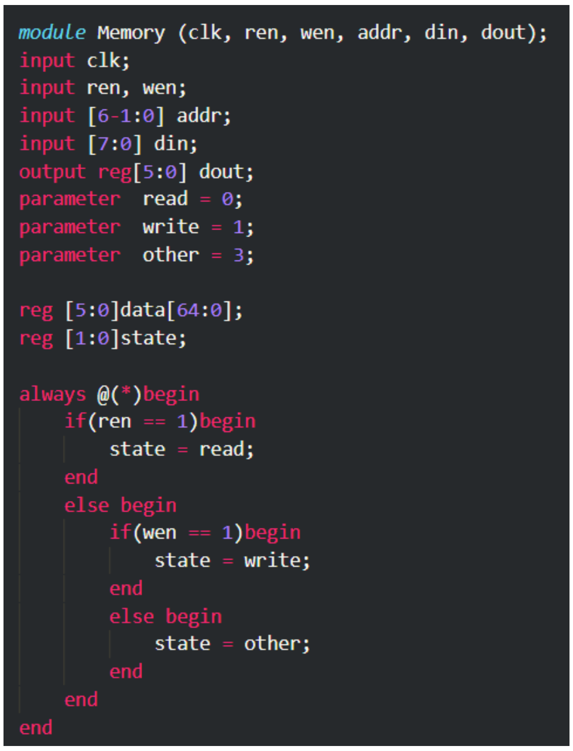
counter 用來當作 clock divider 的感覺，最後用 comparator 比較 counter 和 value 的大小，最後生成 pwm



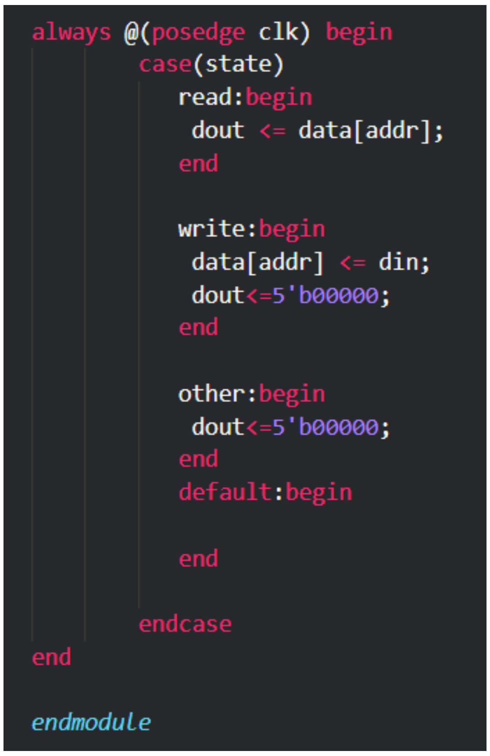
至於 angle 的來源就來自於我的 pmod_step_driver，我在每個字的 state 裡面都輸出一個 angle 的值:0 或是 5 來給 angle decorder 進一步生成 pwm


```
module pmod_step_driver(
    input rst,
    input clk_x,
    input clk_y,
    input c,
    input enter,
    input [5:0]controll,
    output reg [3:0] signal_x,
    output reg [3:0] signal_y,
    output reg [2:0] angle,
    output reg [5:0] state,
    output reg [5:0] led
    );
```

補充一下 lab3 學到的 memory，是用 ren 跟 wen 來告訴 memory 現在要寫入資料還是讀資料，當 ren = 1,wen = 0，就是讀資料功能，當 wen = 1,ren = 0 就是謝入資料功能，再讀或寫的 state 都需要輸入一個 address 來決定要讀/寫入 memory 的記憶體位置，我的 memory 最多可以存入 64 個 6bits 的資料

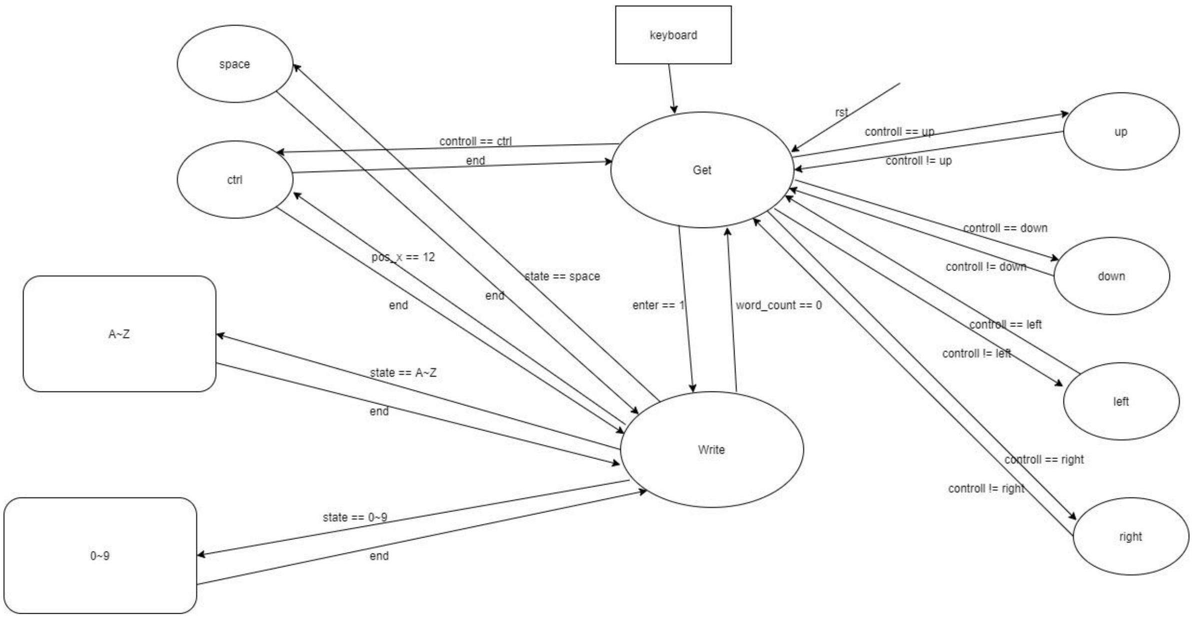


```verilog
module Memory (clk, ren, wen, addr, din, dout);
input clk;
input ren, wen;
input [6-1:0] addr;
input [7:0] din;
output reg[5:0] dout;
parameter  read = 0;
parameter  write = 1;
parameter  other = 3;

reg [5:0]data[64:0];
reg [1:0]state;

always @(*)begin
    if(ren == 1)begin
        state = read;
    end
    else begin
        if(wen == 1)begin
            state = write;
        end
        else begin
            state = other;
        end
    end
end
```



```verilog
always @(posedge clk) begin
        case(state)
           read:begin
            dout  <= data[addr];
           end

           write:begin
            data[addr] <= din;
            dout<=5'b00000;
           end

           other:begin
            dout<=5'b00000;
           end
           default:begin

           end

        endcase
end

endmodule
```

寫字機的 state transition disgram:



keyboard
space
ctrl
Get
rst
controll == ctrl
end
controll == up
controll != up
up
controll == down
controll != down
down
controll == left
controll != left
left
controll == right
controll != right
right
pos_x == 12
end
state == space
end
enter == 1
word_count == 0
A~Z
state == A~Z
end
Write
0~9
state == 0~9
end