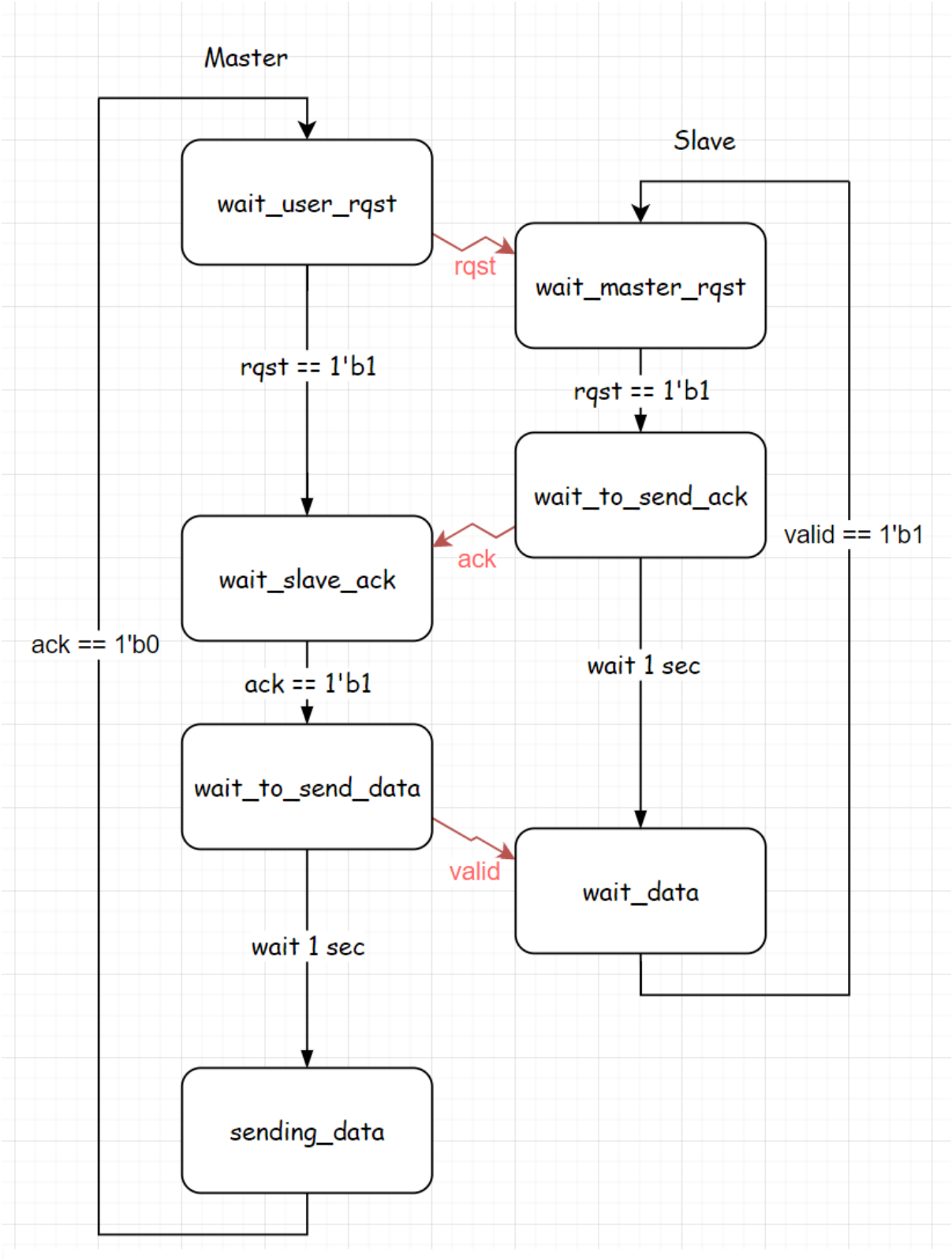
Lab 6 Report

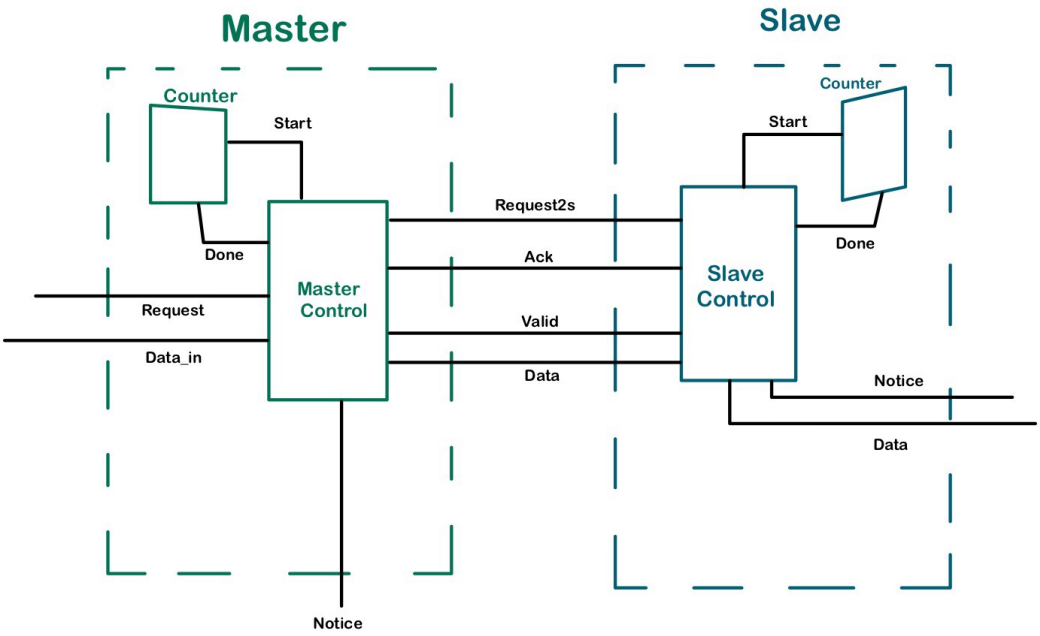
Course	Logic Design Lab
Done	☑
Last edited time	@Dec 18, 2020 10:24 PM
URL	
file	

Chip2Chip

State Diagram



Block Diagram



Master Description

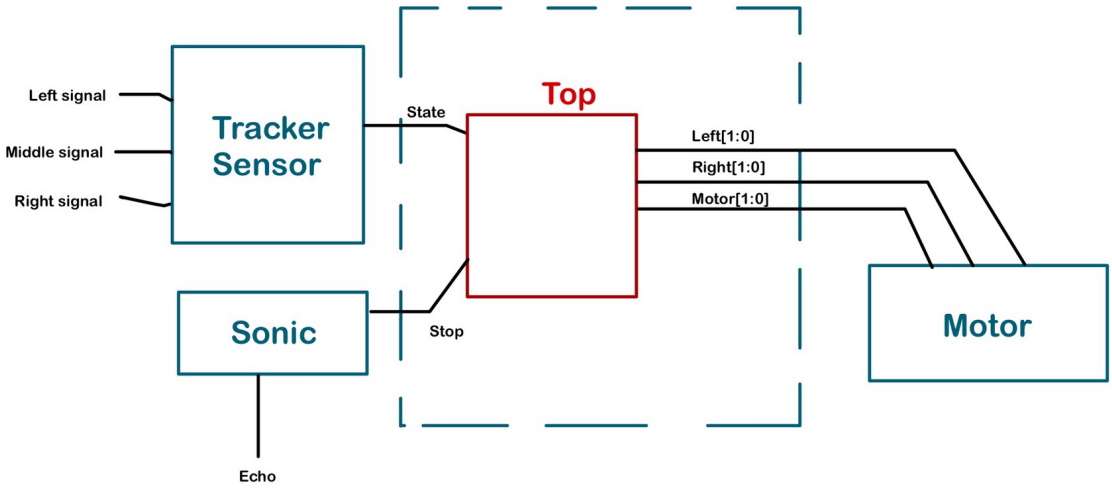
- **wait_uesr_rqst:** 等待使用者按下`request`後，發送`request2s`訊號給**slave**，並進入下一個state。
- **wait_slave_ack:** 在該state等待`ack`訊號，接收到之後進入下一個state。
- **wait_to_send_data:** 開始使用`counter`計時1秒，同時將`notice`設為`1'b1`（LED燈亮），在一秒後進入下一個state。
- **sending_data:** 發送`valid`訊號與`data`給**slave**，並在`ack`訊號變回`1'b0`後，回到第一個state。

Slave Description

- **wait_master_rqst:** 等待來自**master**的`request2s`訊號，接收到之後進入下一個state。
- **wait_to_send_ack:** 開始使用`counter`計時1秒，同時將`notice`設為`1'b1`（LED燈亮），在一秒後發送`ack`訊號給**master**，並進入下一個state。
- **wait_data:** 在該state等待`valid`訊號，接收到之後將資料替換為`data_in`，並回到第一個state。

Tracing Car

Block Diagram



Description

- Tracker Sensor: Combinational Circuit，根據傳進的Signal決定State。

```
always @(*) begin   // 0 is black, 1 is white.
    case (signal)
        3'b000: state = TURN;
        3'b001: state = RRRRIGHT;
        3'b011: state = RIGHT;
        3'b111: state = Straight;
        3'b110: state = LEFT;
        3'b100: state = LLLLEFT;
        default: state = Straight;
    endcase
end
```

- Sonic: 根據傳回的Echo，判定有沒有 <40cm。
- Top: 根據Tracker Sensor與Sonic回傳的結果，決定最終State傳給Motor。

```
parameter Straight = 3'd0;
// parameter Stop = 3'd1;
parameter LLLLEFT = 3'd2;
parameter LEFT = 3'd3;
parameter RIGHT = 3'd4;
parameter RRRRIGHT = 3'd5;
parameter TURN = 3'd6;
```

```
assign update_state = (stop ? 3'd1 : state);

always @(*) begin
    case (state)
        3'd2: next_last_left = 1'b0;
        3'd3: next_last_left = 1'b0;
        3'd4: next_last_left = 1'b1;
        3'd5: next_last_left = 1'b1;
        default: next_last_left = last_left;
    endcase

    left = (state == 3'd6 && last_left == 1'b0) ? 2'b01 : 2'b10;
    right = (state == 3'd6 && last_left == 1'b1) ? 2'b01 : 2'b10;
```

這邊寫了一個比較特別的State叫做TURN，主要是在Sensor偵測到3'b000（全黑）時，根據我們上次記憶的轉彎方向，可以得知是由賽道的哪一邊衝出去，就可以強制車子做大幅度迴轉回賽道的動作。

- Motor: 根據傳進的state與last_left（最後一次轉彎的方向，會在TURN使用到），決定馬達速度。

```
always @(*) begin
    case (mode)

    3'd0: begin
        next_left_motor = 10'd1023;
        next_right_motor = 10'd1023;
    end

    3'd1: begin
        next_left_motor = 10'd0;
        next_right_motor = 10'd0;
    end

    3'd2: begin
        next_left_motor = 10'd470;
        next_right_motor = 10'd1023;
    end

    3'd3: begin
        next_left_motor = 10'd670;
        next_right_motor = 10'd1023;
    end

    3'd4: begin
        next_left_motor = 10'd1023;
        next_right_motor = 10'd670;
    end

    3'd5: begin
        next_left_motor = 10'd1023;
        next_right_motor = 10'd470;
    end

    3'd6: begin
        next_left_motor = (last_left ? 10'd1023 : 10'd800);
        next_right_motor = (last_left ? 10'd800 : 10'd1023);
    end

    endcase
end
```

Slot Machine

原本的 sample code 已經做好了 down 的方向，接者我們只要完成 up 就 OK 了，所以基本上改動的地方主要是在 state_control 這個 module 裡面。

```
//signals
debounce DB0(.s(start), .s_db(start_db), .clk(clk));
debounce DB1(.s(rst), .s_db(rst_db), .clk(clk));
debounce DB2(.s(start2), .s_db(start2_db), .clk(clk));
onepulse OP0(.s(start_db), .s_op(start_op), .clk(clk_d22));
onepulse OP1(.s(rst_db), .s_op(rst_op), .clk(clk_d22));
onepulse OP2(.s(start2_db), .s_op(start2_op), .clk(clk_d22));
```

除了原本的 clk 跟 start 之外，這裡多了 start2，start2 是 left button(up)的輸入訊號

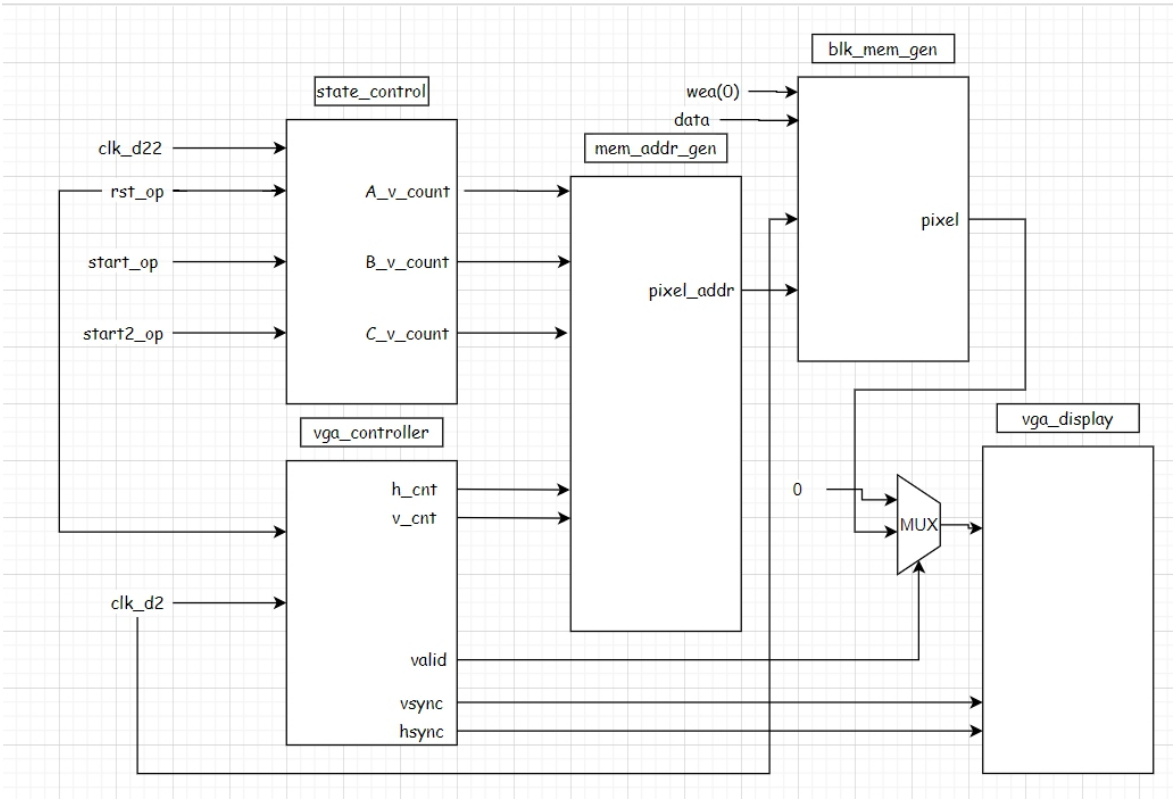
```
//control
state_control SC0(
    .clk(clk_d22),
    .rst(rst),
    .start(start_op),
    .start2(start2_op),
    .A_v_count(A_v_count),
    .B_v_count(B_v_count),
    .C_v_count(C_v_count)
);

mem_addr_gen MAG(
    .h_cnt(h_cnt_re),
    .v_cnt(v_cnt_re),
    .A_v_count(A_v_count),
    .B_v_count(B_v_count),
    .C_v_count(C_v_count),
    .pixel_addr(pixel_addr)
);

//display
blk_mem_gen_0 BMG0(
    .clka(clk_d2),
    .wea(0),
    .addra(pixel_addr),
    .dina(data[11:0]),
    .douta(pixel)
);

vga_controller VC0(
    .pclk(clk_d2),
    .reset(rst),
    .hsync(hsync),
    .vsync(vsync),
    .valid(valid),
    .h_cnt(h_cnt),
    .v_cnt(v_cnt)
);
```

這是整個 code 產生出圖片會需要執行的 module

State_control 負責處理 A、B、C(從左至右)三個部份的 state，並計算他們當前的圖片速度

mem_addr_gen 是把 state_control 計算完 A、B、C 的位置放入，並選擇現在要處理哪個位置的圖片

以下是 module block diagram 的流程



最主要修改的核心在 state_control 裡面，只需要改寫裡面的部分就可以完成 Up、Down 的顯示

sequential 的部分：

```
always@(posedge clk)begin
    if(rst)begin
        A_state <= `STOP;
        B_state <= `STOP;
        C_state <= `STOP;
        counter <= 10'd0;
        A_v_count <= 10'd0;
        B_v_count <= 10'd0;
        C_v_count <= 10'd0;
    end
    else begin
        A_state <= next_A_state;
        B_state <= next_B_state;
        C_state <= next_C_state;
        counter <= next_counter;
        A_v_count <= next_A_v_count;
        B_v_count <= next_B_v_count;
        C_v_count <= next_C_v_count;
    end
end
```

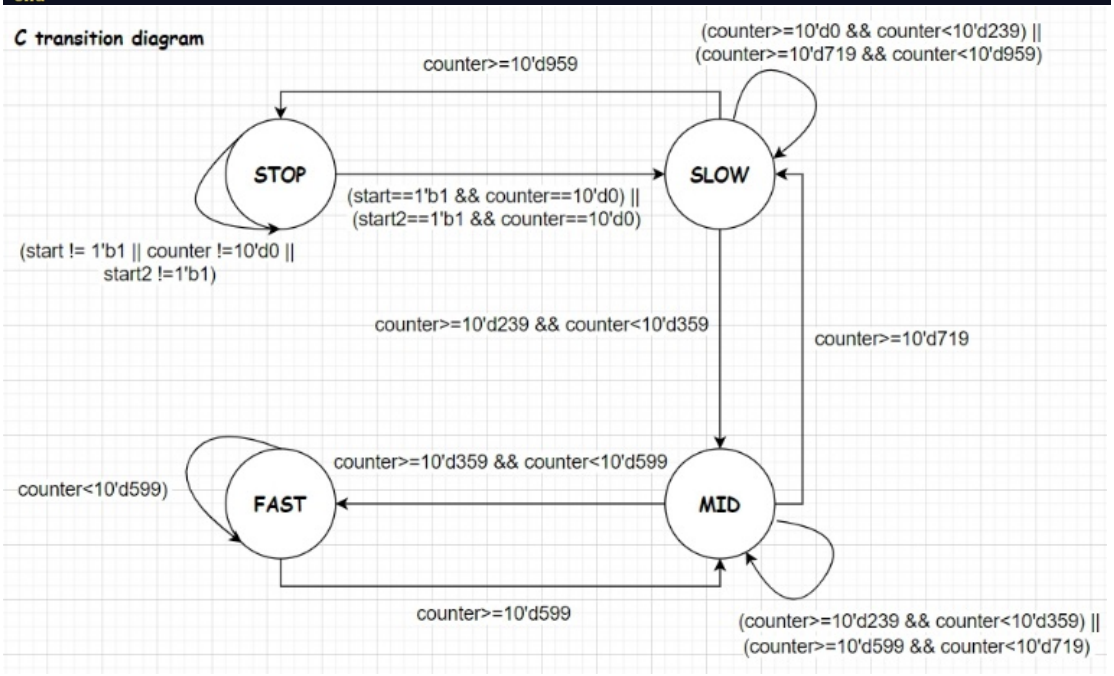
當 rst 為 1 的時候，所有的狀態都會回到 STOP，count 都會變為 0。
不是 1 的時候，A、B、C 就會變成下一個狀態，count 也一樣。

再來是 Combinational 的部分：

前面三張分別是 C、B、A 依據 counter 來做下一個狀態的維持或改變。
因為從左至右分別為 A、B、C，而 C 是最早啟動的部分，最晚停下來的，A 則是最晚啟動，最早停下來的，所以判斷的 counter 不同

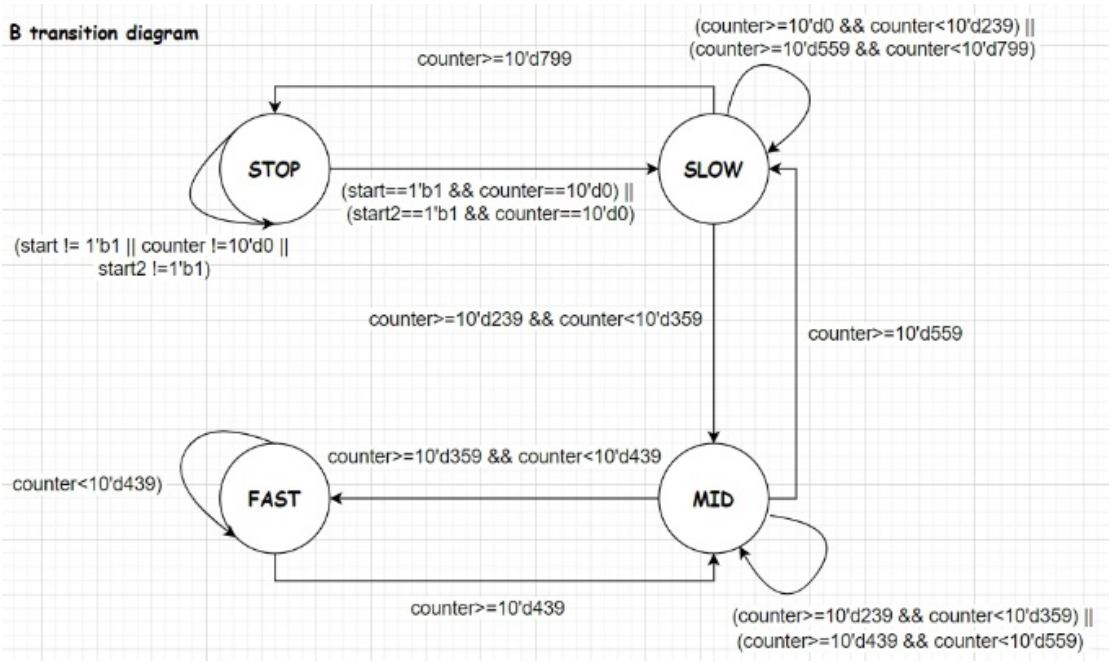
C_state:

```
always@(*)begin
    case(C_state)
        `STOP:begin
            C_to = ((start==1'b1 && counter==10'd0) || (start2==1'b1 && counter==10'd0)) ? `SLOW : `STOP;
            down = (start==1'b1) ? 1 : 0;
            up = (start2==1'b1) ? 1 : 0;
        end
        `SLOW:begin
            C_to = (counter>=10'd959)? `STOP : (counter>=10'd239 && counter<10'd359)? `MID : `SLOW;
            down = down;
            up = up;
        end
        `MID:begin
            C_to = (counter>=10'd719)? `SLOW : (counter>=10'd359 && counter<10'd599)? `FAST : `MID;
            down = down;
            up = up;
        end
        `FAST:begin
            C_to = (counter>=10'd599)? `MID : `FAST;
            down = down;
            up = up;
        end
    endcase
end
```

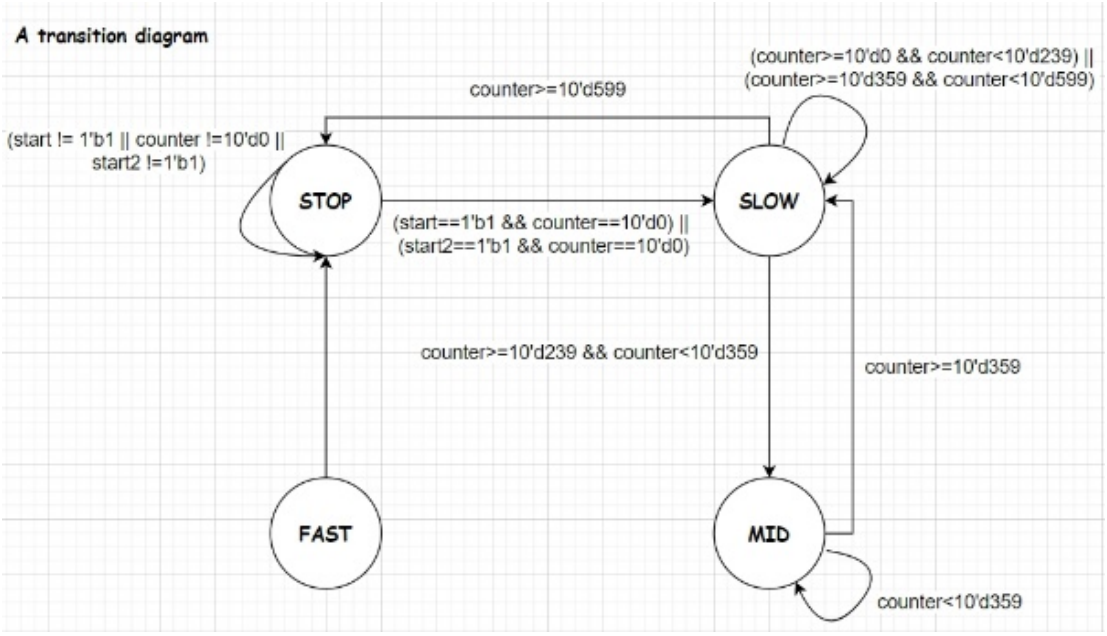


啟動的瞬間，需要去判斷有沒有按下 start(down)或是 start2(up)，才開始啟動。
在 C_state 中有 down 跟 up 的檢測，因為 C 是最早啟動的部分，所以可以知道後續每一個部分要往哪個方向去移動，如果 start 的訊號為 1，down = 1’b1，如果 start2 的訊號為 1，則 up = 1’b1。

```
always@(*)begin
    case(B_state)
        `STOP:begin
            B_to = ((start==1'b1 && counter==10'd0) || (start2==1'b1 && counter==10'd0)) ? `SLOW : `STOP;

        end
        `SLOW:begin
            B_to = (counter>=10'd799)? `STOP : (counter>=10'd239 && counter<10'd359)? `MID : `SLOW;
        end
        `MID:begin
            B_to = (counter>=10'd559)? `SLOW : (counter>=10'd359 && counter<10'd439)? `FAST : `MID;
        end
        `FAST:begin
            B_to = (counter>=10'd439)? `MID : `FAST;
        end
    endcase
end
```




```
always@(*)begin
    case(A_state)
        `STOP:begin
            A_to = ((start==1'b1 && counter==10'd0) || (start2==1'b1 && counter==10'd0)) ? `SLOW : `STOP;
        end
        `SLOW:begin
            A_to = (counter>=10'd599)? `STOP : (counter>=10'd239 && counter<10'd359)? `MID : `SLOW;
        end
        `MID:begin
            A_to = (counter>=10'd359)? `SLOW : `MID;
        end
        `FAST:begin
            A_to = `STOP;
        end
    endcase
end
```



B_state 跟 A_state 的部分沒有其他的改變，只有在判斷開始的時候必須考慮上往上的訊號。

```
always @(*) begin
   if(down == 1'b1 && up == 1'b0)begin
    next_A_v_count = (A_v_count + A_state >= 10'd240)? A_v_count + A_state - 10'd240: A_v_count + A_state;
    next_B_v_count = (B_v_count + B_state >= 10'd240)? B_v_count + B_state - 10'd240: B_v_count + B_state;
    next_C_v_count = (C_v_count + C_state >= 10'd240)? C_v_count + C_state - 10'd240: C_v_count + C_state;
   end
   else if(down == 1'b0 && up == 1'b1)begin
    next_A_v_count = (A_v_count - A_state + 10'd3 < 10'd3)? 10'd240 + A_v_count - A_state: A_v_count - A_state;
    next_B_v_count = (B_v_count - B_state + 10'd3 < 10'd3)? 10'd240 + B_v_count - B_state: B_v_count - B_state;
    next_C_v_count = (C_v_count - C_state + 10'd3 < 10'd3)? 10'd240 + C_v_count - C_state: C_v_count - C_state;
   end
   else begin
    next_A_v_count = A_v_count;
    next_B_v_count = B_v_count;
    next_C_v_count = C_v_count;
   end
end
```

根據 down 跟 up 的訊號來判斷下一個狀態，在往下的部分 sample code 已經完成，而往上的部分因為 v_count 是往回減的，所以必須判斷是否會小於等於 0，小於等於 0 的話就要變回 240 減上多出來的部分，反之 count 會繼續減少。

```
assign next_counter = ((start==1'b0 && start2==1'b0 && counter==10'd0) || (counter >= 10'd1000))? counter : counter+1'b1;
assign next_C_state = C_to;
assign next_B_state = B_to;
assign next_A_state = A_to;
```

Next_counter 這裡要多考慮到往上的訊號，其他則沒有變。

心得

這一次的 lab 非常的有趣，感覺越來越貼近生活了，很多科目學了也不知道怎麼應用在生活上，這一次的實驗課讓我們收穫良多，也覺得很有樂趣，尤其是車子的部分，但是有時候真的不知道是電池問題還是線接錯又或者是 code 寫錯，會一直糾結在一個地方上，結果發現不是這裡錯的時候真的很幹，但是做出來非常有成就感，尤其是在額外關卡，我們的車子只花 8.7 秒就完成一圈，真的很令人開心，不枉費沒有睡覺的那些夜晚。

分工

Chip2Chip 莊景堯
Car 莊景堯
Slot Machine 林諭震
Report 莊景堯/林諭震