Logic design lab

Lab6_Team5_report

2020/12/17

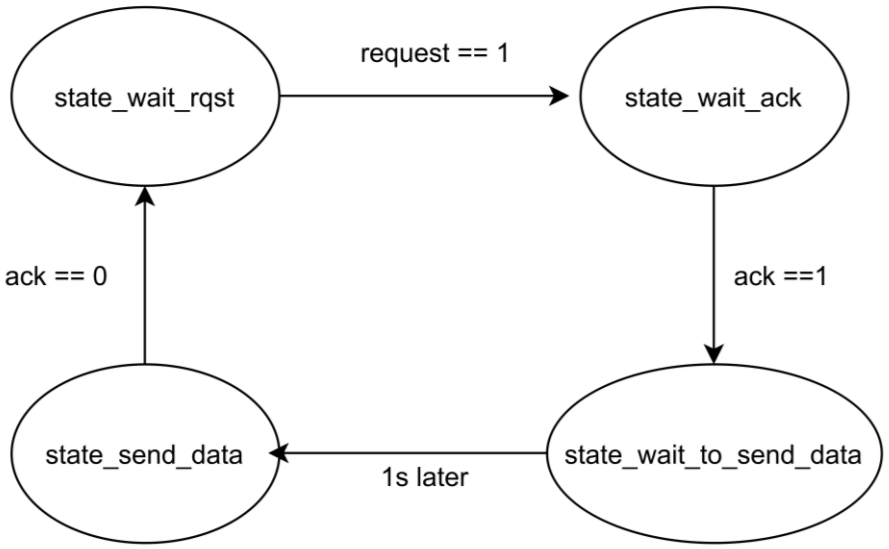
chip2chip

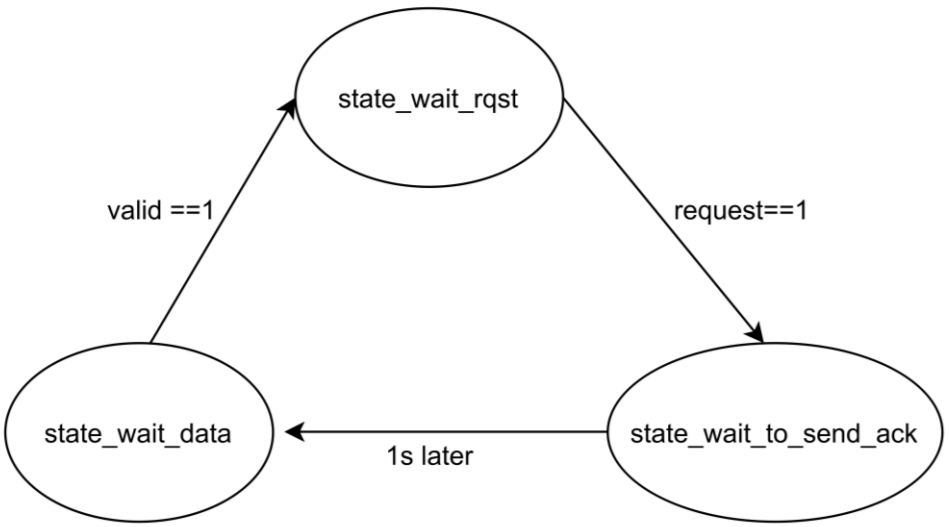
I. Diagram

(i) State diagram

上 : master

下: slave

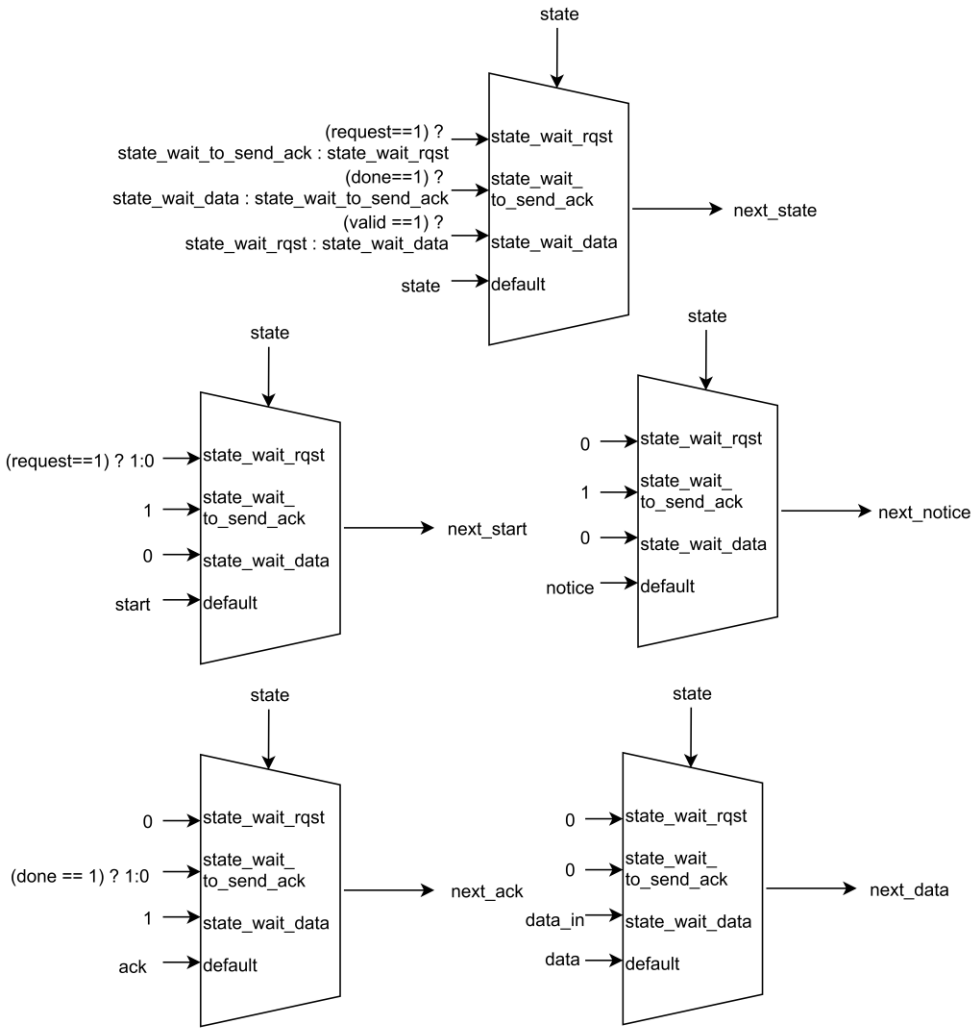




I. Diagram

(ii)電路設計圖

slave_control



II. Explanation

這次chip2chip的實作，主要透過兩塊板子間傳送request、ack作為通訊信號來互相溝通，由master送出request，直到slave收到並送回ack，再由master收到後開始傳送資料，並顯示於slave，其中傳送資料前、顯示數字這兩個state皆會持續一秒，如此循環。

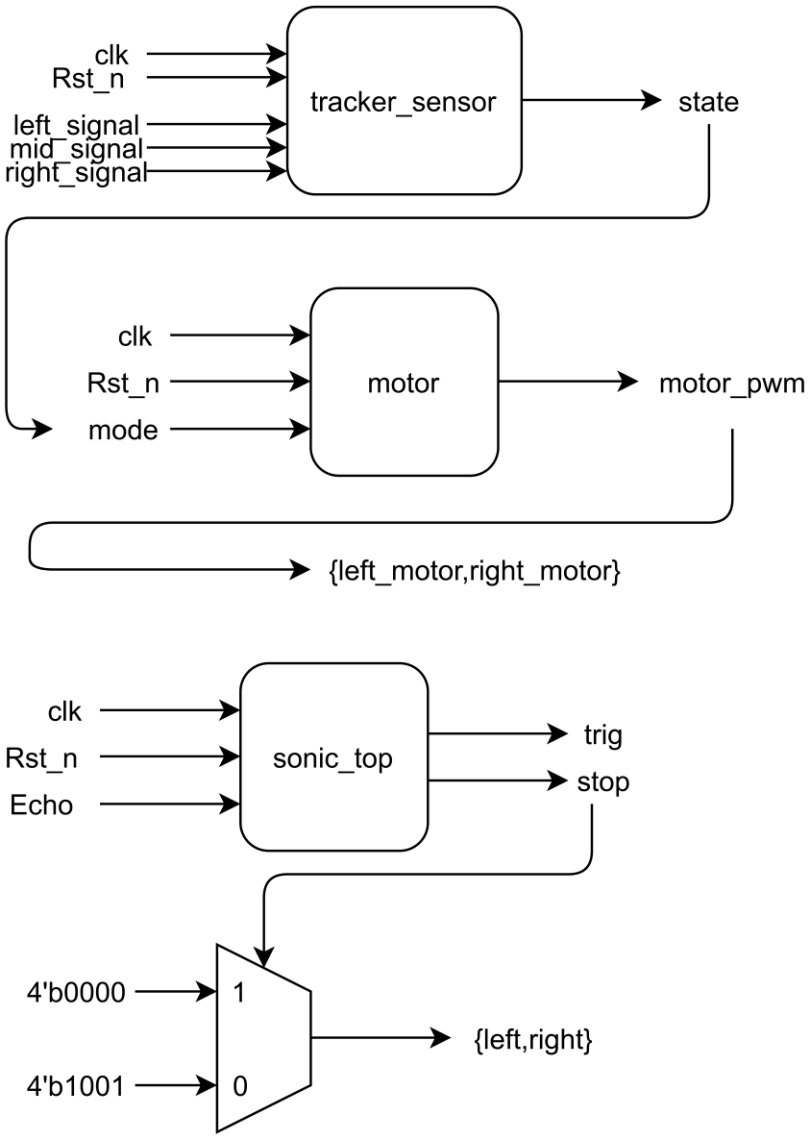
而需要實作的slave_control部分，則是在state_wait_rqst、state_wait_to_send_ack、state_wait_data，三個state轉換，一開始當收到request後切換到state_wait_to_send_ack，並亮起指示燈，當一秒後切換到state_wait_data，傳送ack，並顯示收到的data。

```
case(state)
    state_wait_rqst: begin
        next_state = (request == 1)? state_wait_to_send_ack: state_wait_rqst;
        next_notice = 0;
        next_ack = 0;
        next_data = 0;
        next_start = (request == 1)? 1:0 ;
    end
    state_wait_to_send_ack: begin
        next_state = (done == 1)? state_wait_data : state_wait_to_send_ack ;
        next_notice = 1;
        next_ack = (done == 1) ? 1:0;
        next_data = 0;
        next_start = 1;
    end
    state_wait_data: begin
        next_state = (valid == 1)? state_wait_rqst : state_wait_data;
        next_notice = 0;
        next_ack = 1;
        next_data = data_in;
        next_start = 0;
    end
    default: begin
    end
endcase
```

car

I. Diagram

Block Diagram

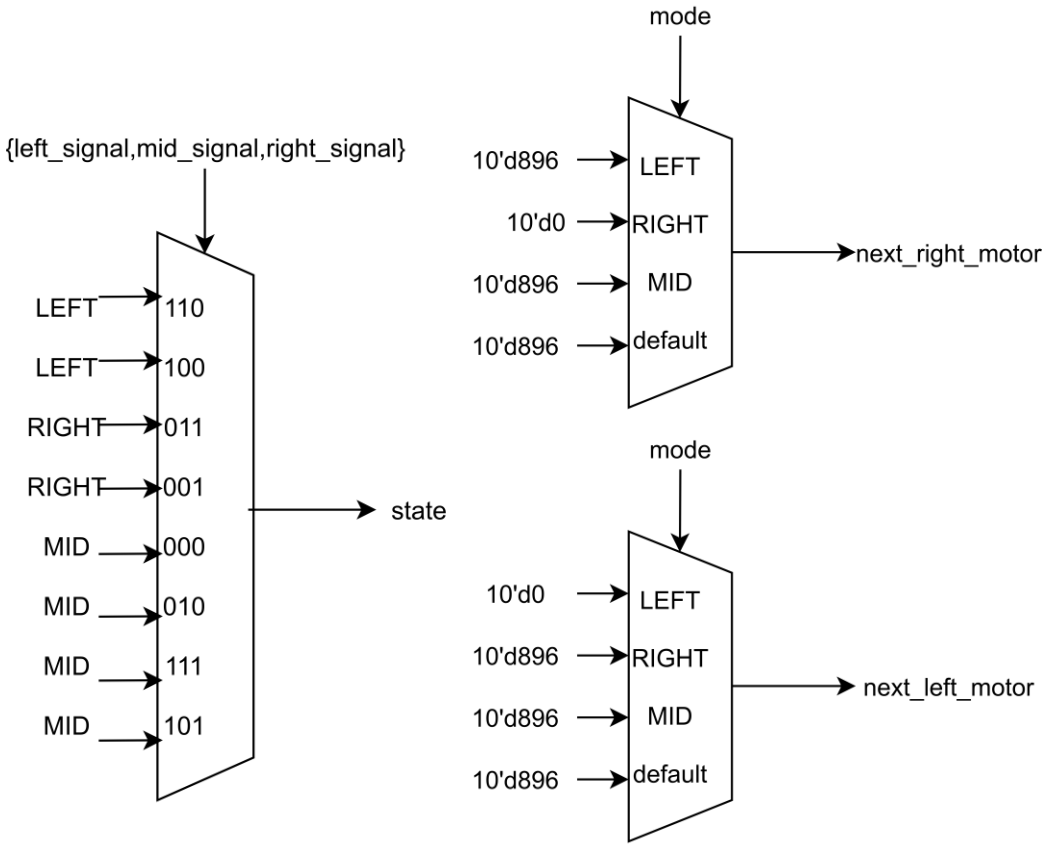


I. Diagram

Block Diagram

左:tracker_sensor

右:motor



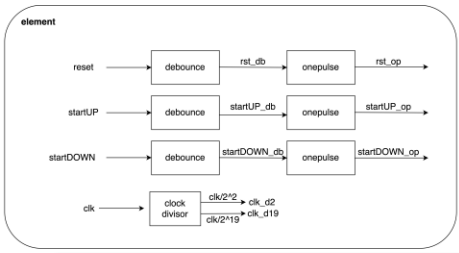
II. Explanation

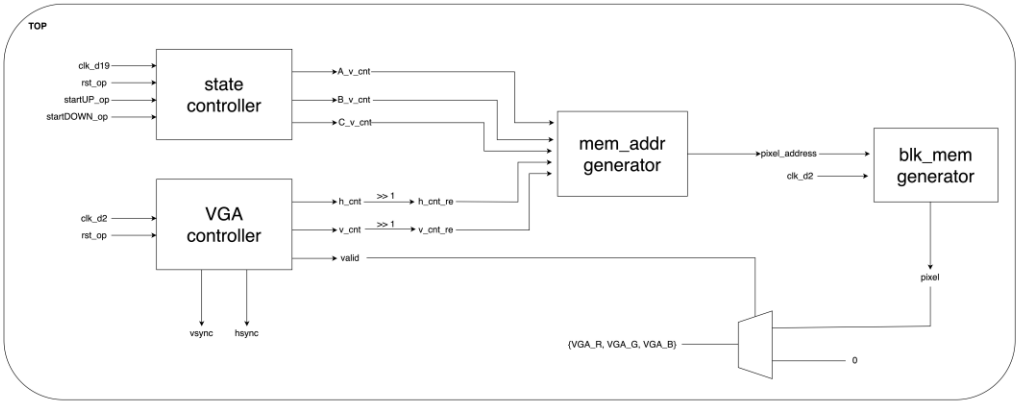
這次的car實作，主要藉由tracker_sensor、sonic判斷目前狀態，並交由motor控制馬達，以達成在不同狀態表現出不同行為的設計，模組中tracker_sensor收到左中右三個訊號後，會依據訊號判斷目前應該要直走、左轉、或是右轉，並傳送出目前的狀態，由motor接受後，會依據狀態給予馬達不同的PWM duty circle，以達成不同轉速的實作，而sonic則是用以判斷物體是否距離車身40cm以內，運作原理是定期打出超音波，以超音波回彈的時間長度去判斷，若是在40cm以內，sonic會傳送stop訊號給top，top就會直接將控制馬達的in0、in1設置為0，直接停止馬達的運作，以實現停車的效果。

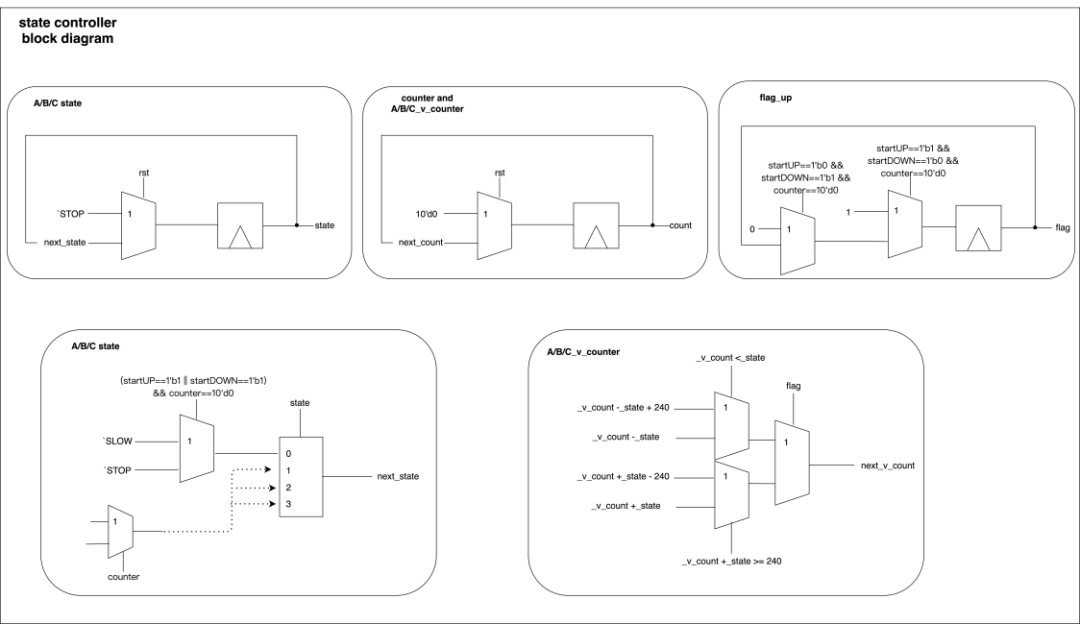
Slot Machine FPGA

I. Diagram

(i)Block Diagram







因A/B/C_state、count的行為類似，故不一一列出。

II. Explanation

如Block diagram所述，我們設計了三個input按鈕，分別是reset、startUP、startDOWN，並將他們依序進行debounce和onepulse處理。

接著state_controller為本題最重要的控制module。透過A/B/C_state去代表三個圖片的移動速度，counter與按鈕input則決定state的種類。

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

（上圖代表next_counter、next_state的處理方式）

（上圖以C_state為例）

而A/B/C_v_count則是表示圖片中一點的垂直位置移動次數。當圖形至最底下後，若再向下移動則需顯示於螢幕的最上層，因此v_count+state >= 240時，

next_v_count = v_count+state-240使之維持於螢幕上。同理，若圖形位於螢幕最上層而繼續向上移動時，就要使之出現於螢幕最下方，故v_count < state時，要讓

next_v_count = v_count-state+240使之維持於螢幕上。

```
if(flag_up == 1'b1) begin
    if(A_v_count < A_state) begin
        next_A_v_count = A_v_count - A_state + 10'd240;
    end
    else begin
        next_A_v_count = A_v_count - A_state;
    end
```

（左圖以圖A為例）

```
else begin
    if(A_v_count + A_state >= 10'd240) begin
        next_A_v_count = A_v_count + A_state - 10'd240;
    end
    else begin
        next_A_v_count = A_v_count + A_state;
    end
```

II. Explanation

而因為只在STOP的情況下能選擇UP或DOWN，但在各時間皆需知道圖案應向上或向下移動，因此設計flag_up去紀錄圖形要向何處移動。

```
if(startUP==1'b1 && startDOWN==1'b0 && counter==10'd0) begin
    flag_up <= 1'b1;
end
else if(startUP==1'b0 && startDOWN==1'b1 && counter==10'd0)begin
    flag_up <= 1'b0;
end
else begin
    flag_up <= flag_up;
end
```

這個module最後一部分為reset。因應討論區提問，我們也將其設計為在完成一次運作後，無須先reset就可直接進行下一次運作。而我們所使用的方式是當counter>=1000時（代表已經運作完成），將A/B/C_v_count以及counter本身都歸0，又因為state本來就會進入STOP，各條件都與reset時相等，因此可以不用而外進行reset。

```
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;
    flag_up <= 1'b0;
```

```
if(counter >= 10'd1000)begin
    next_A_v_count <= 10'd0;
    next_B_v_count <= 10'd0;
    next_C_v_count <= 10'd0;
end
```

（counter的歸零已於上一頁有圖，故不再次提出）

II. Explanation

state controller處理完的output（A/B/C_v_count），會與VGA controller處理完的output（h_cnt、v_cnt）共同成為mem_addr generator的input，並且透過mem_addr generator的運算，得出pixel應在的位置。這部分運算是透過h_cnt得知pixel在螢幕上的水平位置，進而知道這個pixel屬於A/B/C何者。再由v_count決定垂直移動的距離，這樣就可以知道新的pixel_address為何。

```
assign v_mem = (h_cnt < 10'd110)? A_v_count :(h_cnt > 10'd210)? C_v_count : B_v_count;
assign v_cnt_total = v_cnt + (16'd239 - v_mem);
assign v_cnt_new = (v_cnt_total >= 16'd239)? v_cnt_total - 16'd239 : v_cnt_total;

assign pixel_addr = v_cnt_new*320 + h_cnt;
```

最後透過VGA_rgb處理、blk_mem等處理，便可藉由將運算的結果呈現於圖形上，並藉由VGA輸出。

```
assign {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel : 12'h0;
```

What are we learned from this Lab

這次lab6需要實作三個FPGA實作題，而助教也提供了sample code，因此最主要花費時間的地方便在於trace code以及處理各種硬體bug，在trace code時，在兩分code有發現兩處混用“=”以及”<=”的狀況，一開始有些困惑，後來才發現應該是助教忙中出錯。

而在實作car時因為電池的續航力比預期來的少上許多，因此一開始還曾經以為是code寫壞了、沒有起作用，後來才意識到是電力不足，這個過程也花費了不少時間，而後來因為還有其他活動占用準備時間、為了盡速提早完成，在助教發放額外電池前，我們還購買了額外的電池以確保在coding時能夠排除其他硬體狀況，也算是因為自己誤算了中間會出現的狀況而產生的額外費用，算是得不償失吧，尤其一顆9V電池還真的是不便宜，一次性的購買足夠的量產生的費用也是相當可觀，下次應該要提早向助教回報狀況，以確定該如何處理，這次算是成功地當了一次課金玩家，花錢以減少自己花費的額外時間，也算是為自己買一份經驗吧。

Cooperation

108062211 黃子瑋：

car FPGA、chip2chip FPGA實作與report、各題 debug

108062213 顏浩昀：

slot_machine FPGA實作與report