

11010EECS207001Logic Design Lab

**LAB6 : Peripheral Components: VGA,
Mouse, and Dual FPGA**

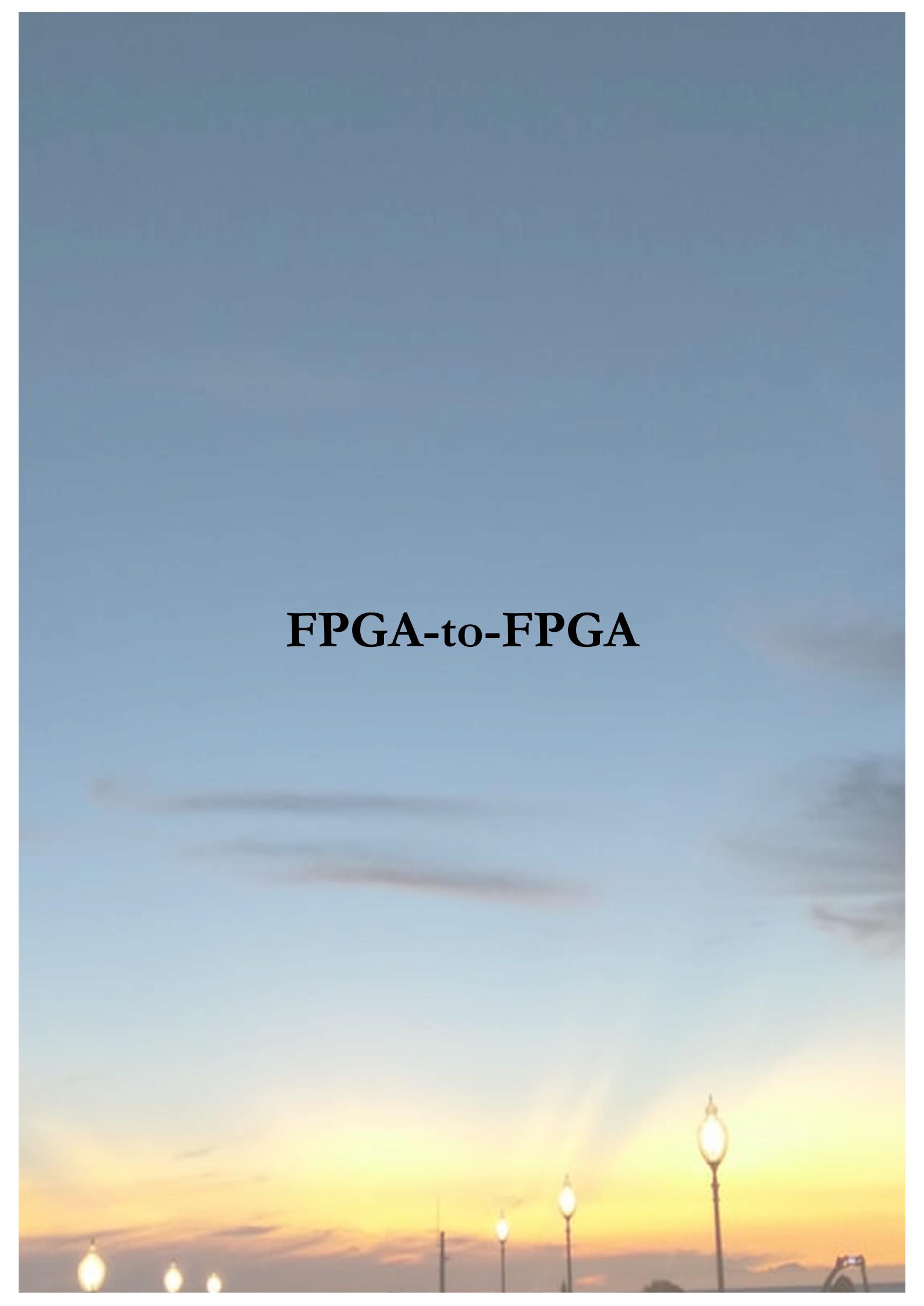
TEAM9

組長：108062213 顏浩昀

組員：106062304 黃鈺舒

Prof. Chun-Yi Lee

2021.12.16

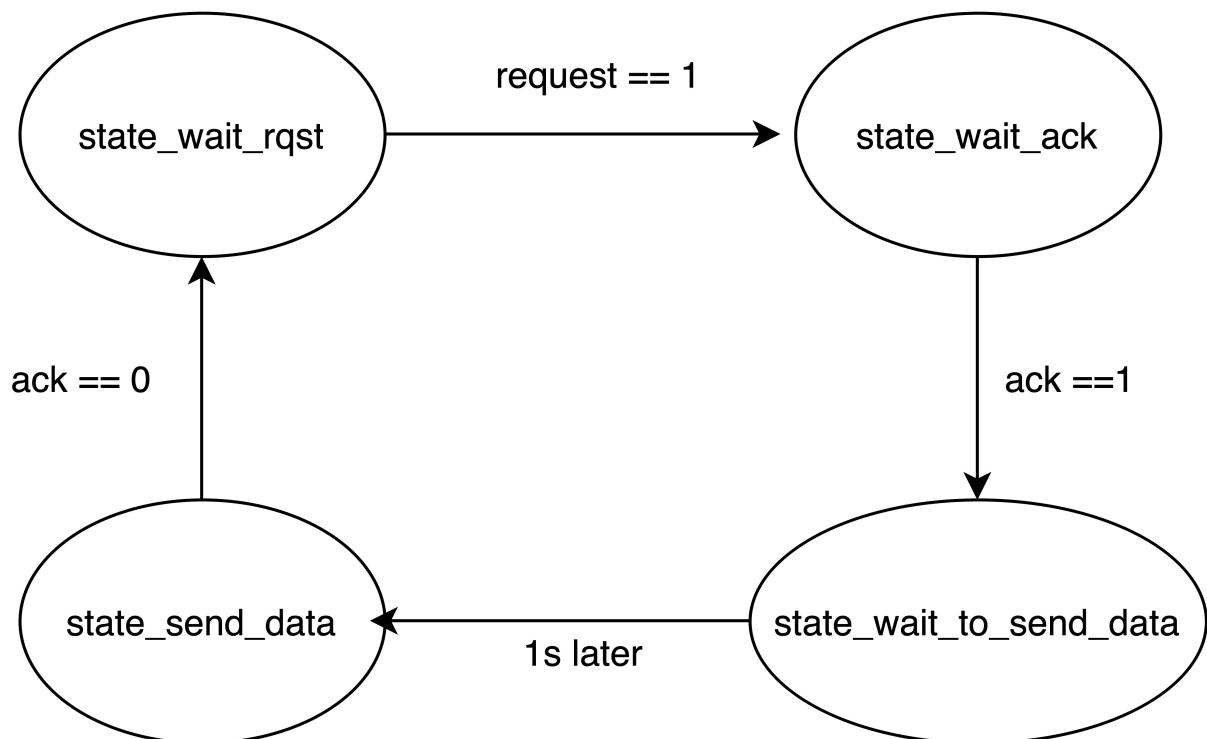
A photograph of a sunset or sunrise sky. The top half of the image is a clear blue, transitioning into a warm orange and yellow glow near the horizon. Several streetlights are visible along the bottom edge, their lights glowing brightly against the darkening sky.

FPGA-to-FPGA

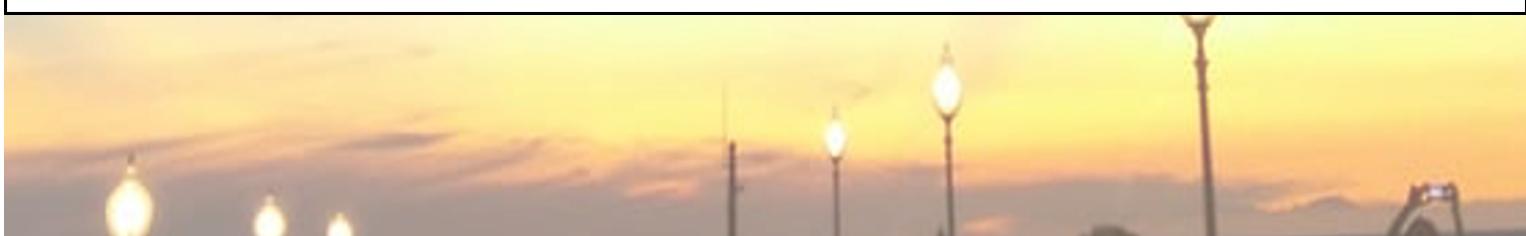
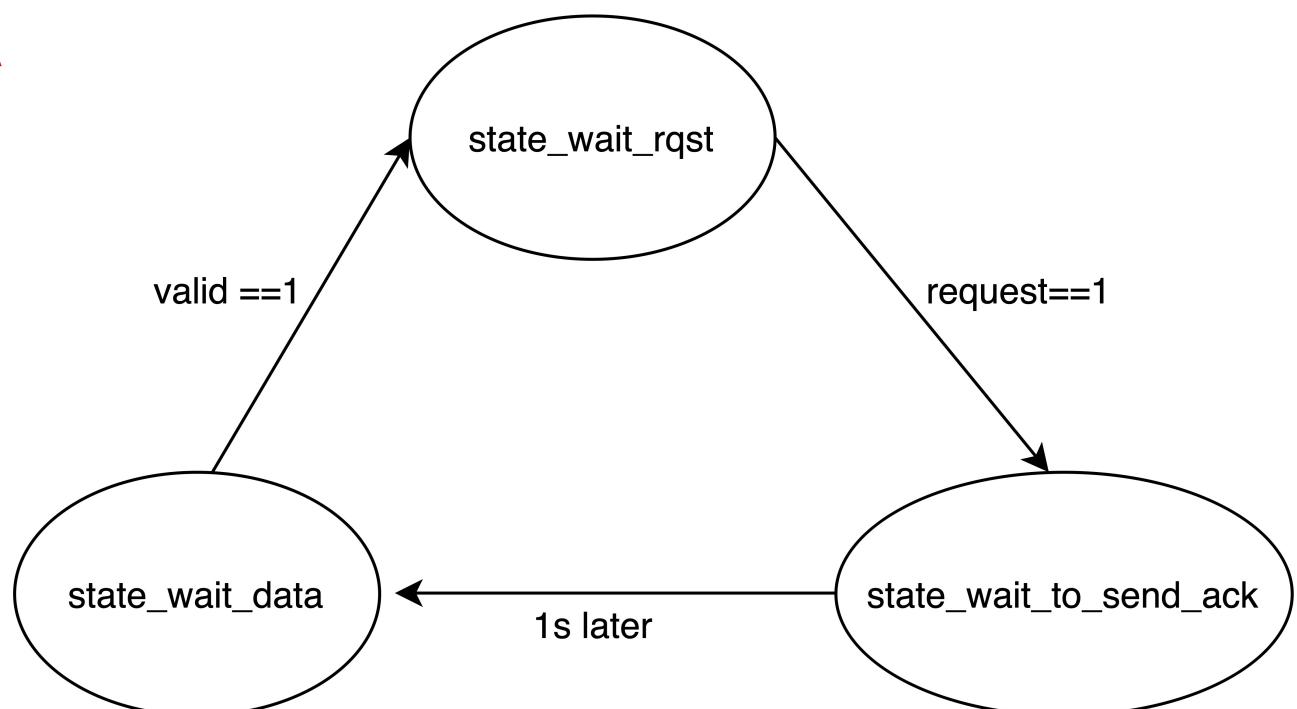
I. Diagram

(i) State diagram

Master
FPGA

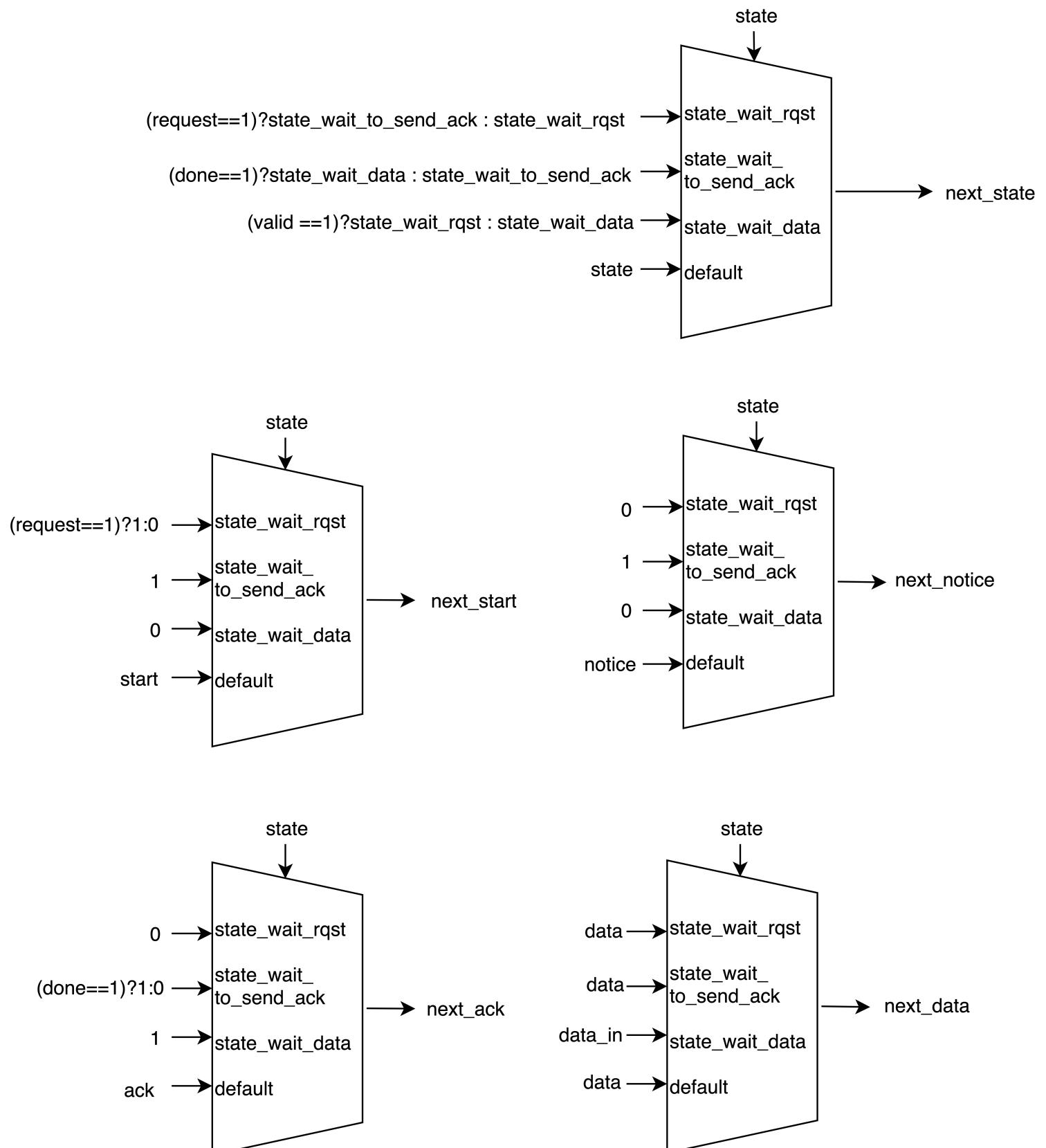


Slave
FPGA



I. Diagram

(ii) Slave FPGA block diagram



II. Explanation

兩個FPGA分別有自己的state與各自的行为，master的部分sample code已經完成，但在sample code中master top部分data to slave output沒有透過master control的data與data_in去控制，導致slave端數字提早變換，因此將data to slave output皆為master control的output(data)。（下圖為data to slave output的更動）

```
// assign data_to_slave_o = data_to_slave;  
master_control ms_ctrl_0(.clk(clk), .rst_n(rst_n_inv), .request(op_request), .ack(ack),  
|.data_in(data_to_slave), .notice(notice_master), .data(data_to_slave_o), .valid(valid), .request2s(request2s))
```

而這次主要要修改的是slave control的部分。Slave的部分共有三個state，分別是wait request, wait to send ack, wait data。在wait request時，要等待master傳request signal過來，根據前面的block diagram可知此時notice, ack都是0，data則是hold住不變，start及state則是在接收到request signal後有所改變。在收到request後，state會進入wait to send ack，這時候data仍不變，notice及start則維持1。這時候透過module counter計算1 second後將done signal以1傳出，done signal會影響的是ack以及state的部分，在done == 1 (after 1 second)，ack=1用來傳回給master表示slave有確實收到request，並且將state轉為wait data，等待master傳入真正的data。

在master收到ack後，會先亮起notice的LED燈號，然後透過module counter計算1 second，然後使valid = 1並且在此時將valid與data一起傳給slave，並熄滅LED燈號。這時slave會因為傳入的data_in改變7 segment，並在收到valid=1之後回到wait request state。（下圖為slave control實作）

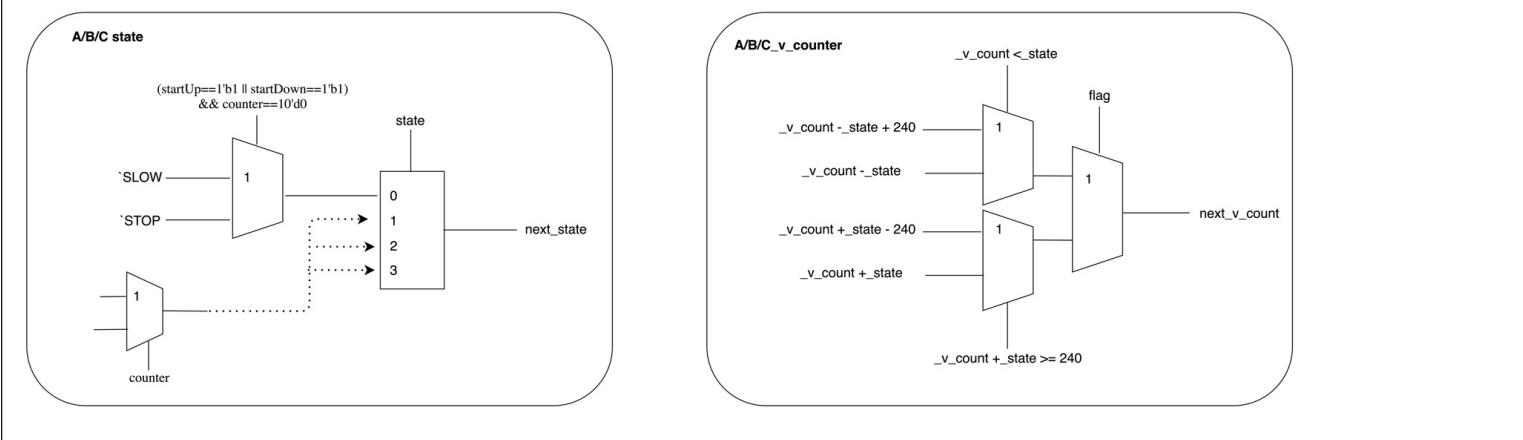
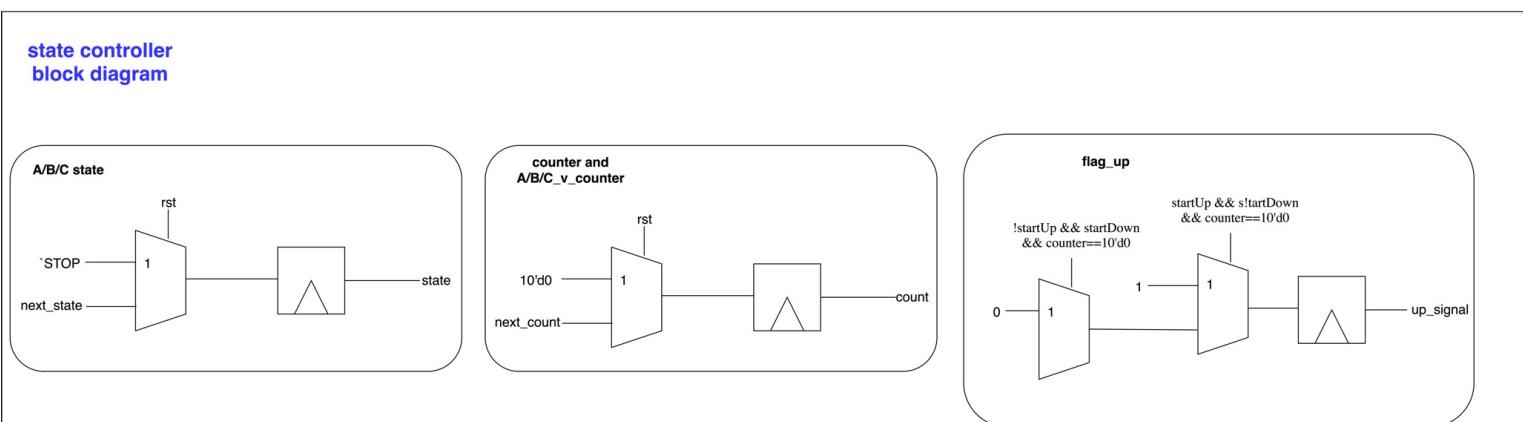
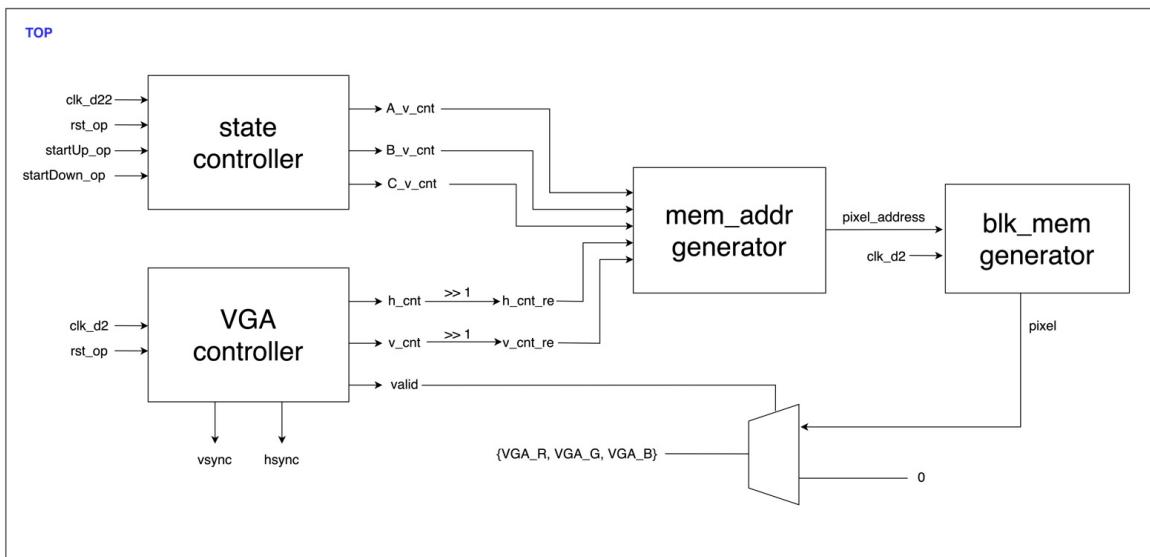
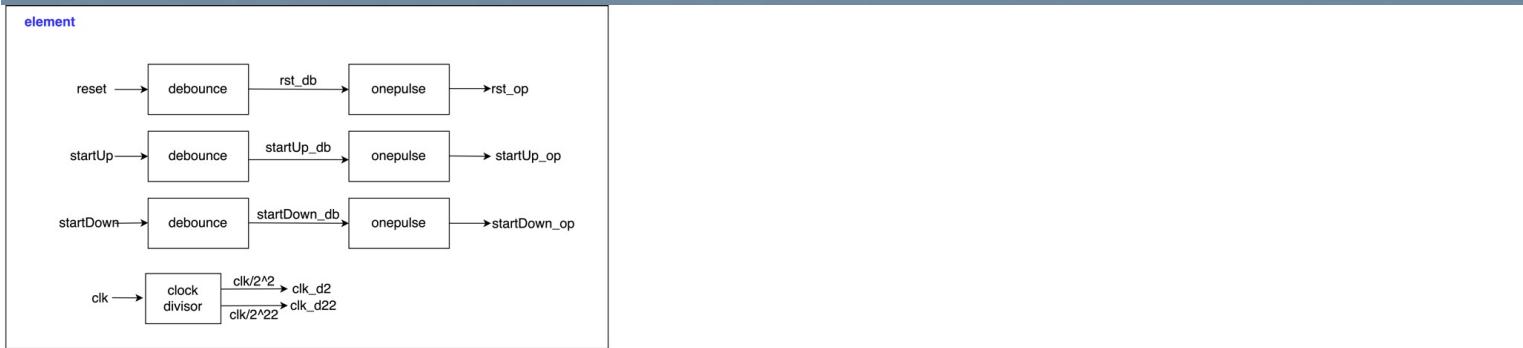
```
state_wait_rqst: begin  
    next_state = (request == 1)? state_wait_to_send_ack: state_wait_rqst;  
    next_notice = 1'b0;  
    next_ack = 1'b0;  
    next_data = data;  
    next_start = (request == 1)? 1'b1: 1'b0;  
    next_start2 = 1'b0;  
end  
state_wait_to_send_ack: begin  
    next_state = (done == 1)? state_wait_data : state_wait_to_send_ack;  
    next_notice = 1'b1;  
    next_ack = (done == 1)? 1'b1 : 1'b0;  
    next_data = data;  
    next_start = 1'b1;  
end  
state_wait_data: begin  
    next_state = (valid == 1)? state_wait_rqst : state_wait_data;  
    next_notice = 1'b0;  
    next_ack = 1'b1;  
    next_data = data_in;  
    next_start = 1'b0;  
end
```

The slot machine

I. Diagram

(i) block diagram

因 A/B/C_state, A/B/C_v_count 所做行為類似，僅僅是改變條件不同，在 block diagram 部分以同一張圖表示。



II. Explanation

首先更動的是startUp與startDown的部分，除了多做出一個按鈕以外，重點是需要清楚知道pixel要維持向上移動還是向下移動，因此 module state control中新增up_signal去紀錄現在的移動方式。

```
if(startUp && !startDown && counter == 10'd0) begin //counter to check init state
| up_signal <= 1'b1; //slot up
end
else if(!startUp && startDown && counter == 10'd0) begin
| up_signal <= 1'b0; //slot down
```

本題的一大重點為counter。counter代表的是一個計時器，當按下start時counter會從0開始向上數值到1000為止，並在數字後歸0並停止。

```
assign next_counter = ((startUp==1'b0 && startDown==1'b0 && counter==10'd0) || (counter >= 10'd1000))? 10'd0 : counter+1'b1;
```

而counter的用處有兩個。第一個是只有在counter為0時，按下startUp、startDown button才有效果，避免當slot machine運作到一半時被改變方向。第二個作用是改變圖片移動速度。根據A,B,C state變化條件，在counter為在某個區間時各個A,B,C分別有自己的移動速度（已在一开始定義好STOP, SLOW, MID, FAST）。

```
(以C_state為例)
case(C_state)
`STOP:begin
    if(startUp && !startDown && counter == 10'd0) begin
        next_C_state = `SLOW;
    end
    else if(!startUp && startDown && counter == 10'd0) begin
        next_C_state = `SLOW;
    end
    else begin
        next_C_state = `STOP;
    end
    // C_to = (start==1'b1 && counter==10'd0)? `SLOW : `STOP;
end
`SLOW:begin
    next_C_state = (counter>=10'd959)? `STOP : (counter>=10'd239 && counter<10'd359)? `MID : `SLOW;
end
`MID:begin
    next_C_state = (counter>=10'd719)? `SLOW : (counter>=10'd359 && counter<10'd599)? `FAST : `MID;
end
`FAST:begin
    next_C_state = (counter>=10'd599)? `MID : `FAST;
end
endcase
```

II. Explanation

接著是要判斷圖片更新位置，透過的是v_count與state，v_count代表現在pixel的位置，而state代表現在pixel移動的速度。首先如果是向上移動，因為螢幕左上為原點(0,0)的位置，如果現在的v_count < state，代表下一個時間點pixel便會超出螢幕上邊界，因此要將pixel位置移動至螢幕的最下方，也就是v_count+240-state。相反的，若是向下移動，如果v_count+state超過240代表下一時間點pixel會超出螢幕的下邊界，因此要將pixel位置移到螢幕最上方，也就是v_count+state-240。如果判斷不會超出螢幕邊界，僅需要將v_count與state簡單加減就可以求出新的位置。

(下方以A state, A_v_count為例)

```
if(A_v_count < A_state) begin //out of upside window
    next_A_v_count = A_v_count + 10'd240 - A_state;
end
else begin
    next_A_v_count = A_v_count - A_state;
end

if(A_v_count+A_state >= 10'd240) begin //out of downside window
    next_A_v_count = A_v_count + A_state - 10'd240;
end
else begin
    next_A_v_count = A_v_count + A_state;
end
```

主要實作的state control module大致實作如上，接下來簡單說明影響pixel位置與A,B,C的mem_addr_gen module。這個module是用來顯示pixel位置。會透過state control與 vga_controller的output作為這個module的input，計算出垂直pixel變化的顯示位置。並且前面說到的A,B,C也是在此module，根據h_cnt（水平方向）做出分別。最後的pixel位置計算是根據v_cnt_new（新的垂直高度）*320 + h_cnt。從這裡我們可以知道pixel的計算是從左至右從上至下累加。V_cnt代表的是index of row，在找出第幾列後，再根據h_cnt去表示該列的詳細位置。

What are we learned from this Lab

這次的題目都是FPGA的實作，我們遇到的問題是在chip to chip那題。似乎因為連接線接觸不良的關係，在多個小時絞盡腦汁的debug中，始終無法維持很好的正確率，總是有時候正確有時候master與slave的數字會不一樣。最後發現是接觸不良時雖然有些無奈，但也慶幸在這次提前發現問題，因為在Final Project我們也會用到這些線材，讓我們知道要更注意這方面的問題，正所謂工欲善其事，必先利其器。

除此之外，這次的Lab就是用於熟悉Final Project需要的工具，我覺得從Audio, keyboard到VGA, chip相接，可說是功能多樣且有趣，雖然每次的fpga題目都不算簡單，也總是需要花費許多時間debug，但希望這些能成為我們順利完成final project的養分。



Cooperation

108062213 顏浩昀：

chip2chip 實作、slot machine 實作、電路圖繪製、report 製作

106062304 黃鈺舒：

chip2chip debug、電路圖繪製、report 製作

