Lab6_Team1_Report

組員

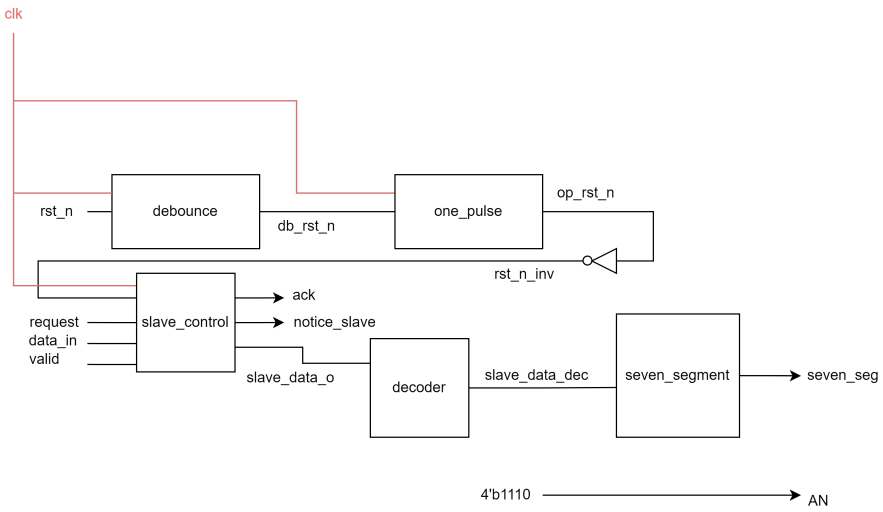
107071016 施泳瑜
109062320 朱季葳

FPGA Demonstration 1

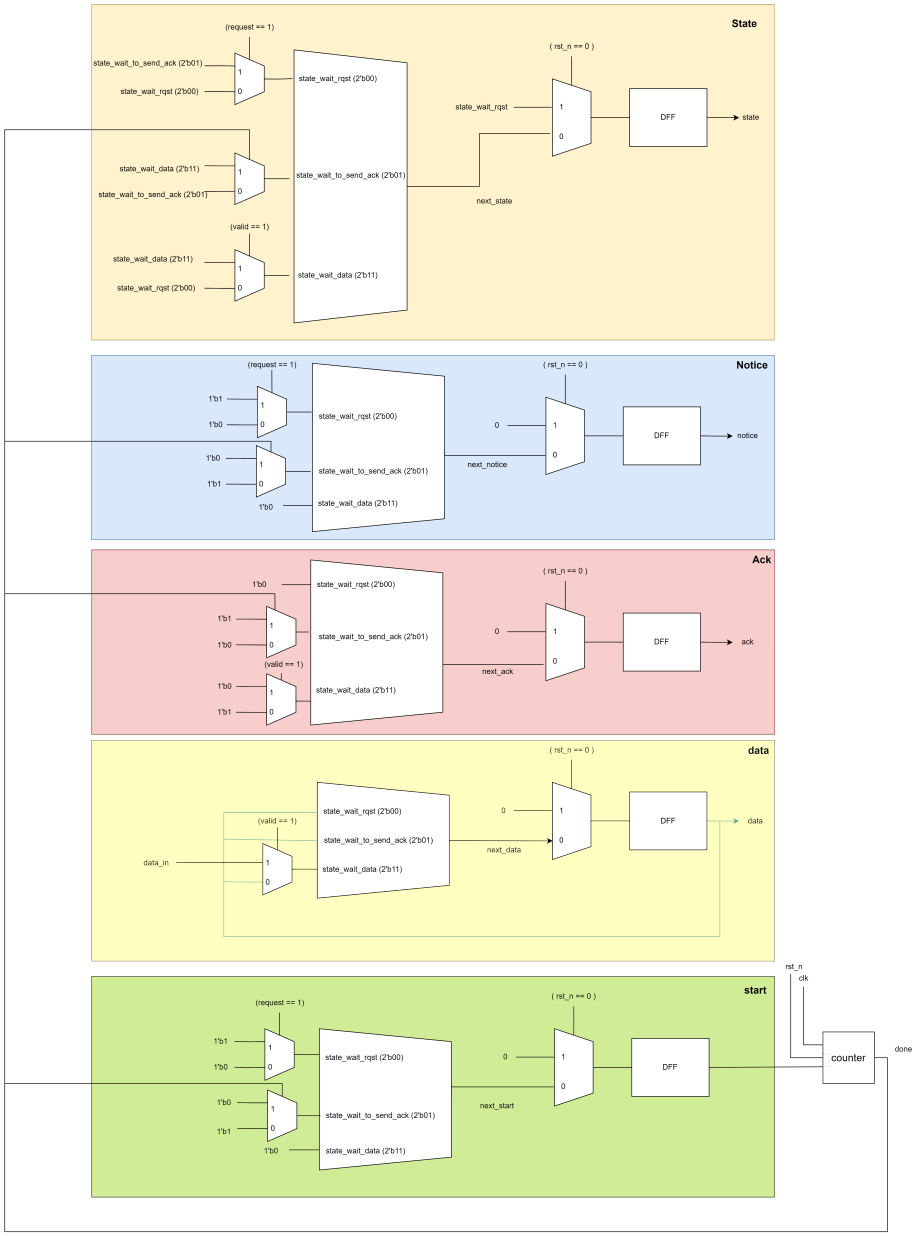
Drawing of the design of FPGA Demonstration 1

[chip2chip_slave]

- Top module



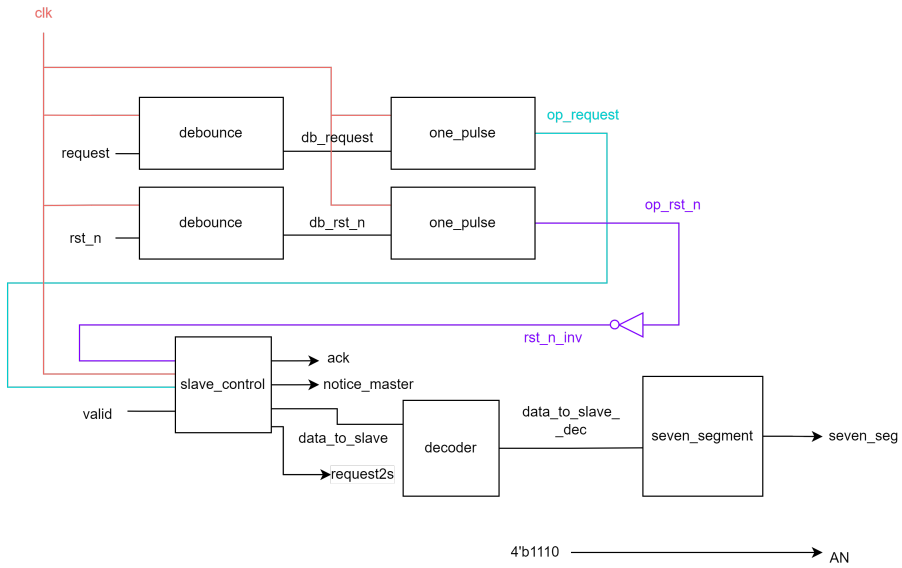
- slave control



- debounce / one_pulse / decoder / seven_segment：by sample code
- counter：by sample code

[chip2chip_master]

- Top module



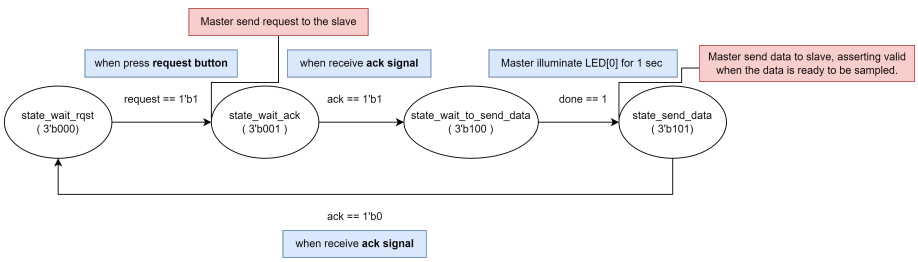
- master control：by sample code
- debounce / one_pulse / decoder / seven_segment：by sample code
- encoder：by sample code

Design Explanation

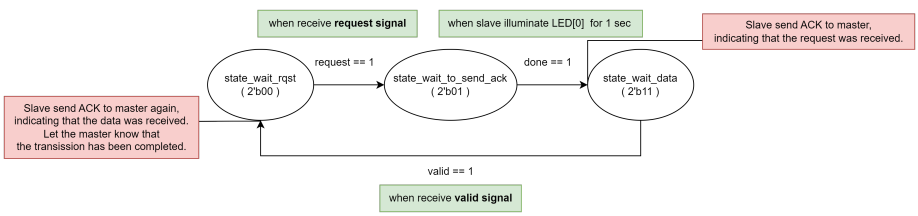
從 sample code 給的檔案中，只修改了必須更動的 slave_control 這項模組，其餘程式碼未更動。

- 我們先比照 master control 的 code 後，畫出相對應的 State Diagram 設計作為 slave control 的參考，並依此對 slave control 劃出了以下 State Diagram 設計。

ϟ Master State Transition Diagram ϟ



ϟ Slave State Transition Diagram ϟ



- 撰寫各 STATE
  - state_wait_rqst (2'b00)：等著接收 request。
    - next_state = (request == 1)? state_wait_to_send_ack : state_wait_rqst; ➡ 接收到 request 時改變狀態到 state_wait_to_send_ack，讓 LED[0] 對開始亮做準備。
    - next_notice = (request == 1)? 1'b1 : 1'b0; ➡ 接收到 request 之前 LED[0] 不亮 / 接收到時 LED[0] 亮
    - next_ack = 1'b0; ➡ 不用向 master 傳送 ack
    - next_data = data; ➡ hold 本來的 data
    - next_start = (request == 1)? 1'b1 : 1'b0; ➡ 接收到 request 時開始透過 counter 計算秒數來確保 LED[0] 亮一秒
  - state_wait_to_send_ack (2'b01)：讓亮燈持續一秒。
    - next_state = (done == 1)? state_wait_data : state_wait_to_send_ack; ➡ 燈亮一秒之後改變狀態到 state_wait_data，轉換至即將發送 ack 的狀態。
    - next_notice = (done == 1)? 1'b0 : 1'b1; ➡ LED[0] 持續亮一秒 / LED[0]一秒後熄滅
    - next_ack = (done == 1)? 1'b1 : 1'b0; ➡ 亮的時候不用向 master 傳送 ack ，LED[0]亮一秒後馬上傳 ack 給 master
    - next_data = data; ➡ hold 本來的 data
    - next_start = (done == 1)? 1'b0 : 1'b1; ➡ 繼續透過 counter 計算秒數來確保 LED[0] 亮一秒，完成後不再需要透過 counter 計算 LED[0] 亮的秒數而關閉 counter 功能
  - state_wait_data (2'b11)：發送出 ack 訊號來告訴 master 現在是可以接收 data 的狀態。
    - next_state = (valid == 1)? state_wait_rqst : state_wait_data; ➡ 接收完 data 之後改變狀態到

state_wait_rqst，繼續等待下次接收 request。

- next_notice = 1’b0; ➨ LED[0] 不亮
- next_ack = (valid == 1) ? 1’b0 : 1’b1; ➨ 向 master 傳送 ack 表示有收到訊息
- next_data = (valid == 1)? data_in : data; ➨ 接收到 data 之時更改 data 值為現在的 data_in
- next_start = 1’b0; ➨ 一樣不需要透過 counter 計算 LED[0] 亮的秒數

I/O pin assignment

[chip2chip_master]

input	port
clk	W5
rst_n	T18
valid	P17
ack	L17
notice_slave	U16
request	M19

output	port
data_in[2]	A15
data_in[1]	A17
data_in[0]	C15
in[7]	W13
in[6]	W14
in[5]	V15
in[4]	W15
in[3]	W17
in[2]	W16
in[1]	V16
in[0]	V17

[chip2chip_slave]

input	port
clk	W5
rst_n	T18
valid	P17
ack	L17
notice_master	U16
request	U18
request2s	M19

output	port
data_to_slave_o[2]	A15
data_to_slave_o[1]	A17
data_to_slave_o[0]	C15
in[7]	W13
in[6]	W14
in[5]	V15
in[4]	W15
in[3]	W17
in[2]	W16
in[1]	V16
in[0]	V17

[7-segment display for master & slave]

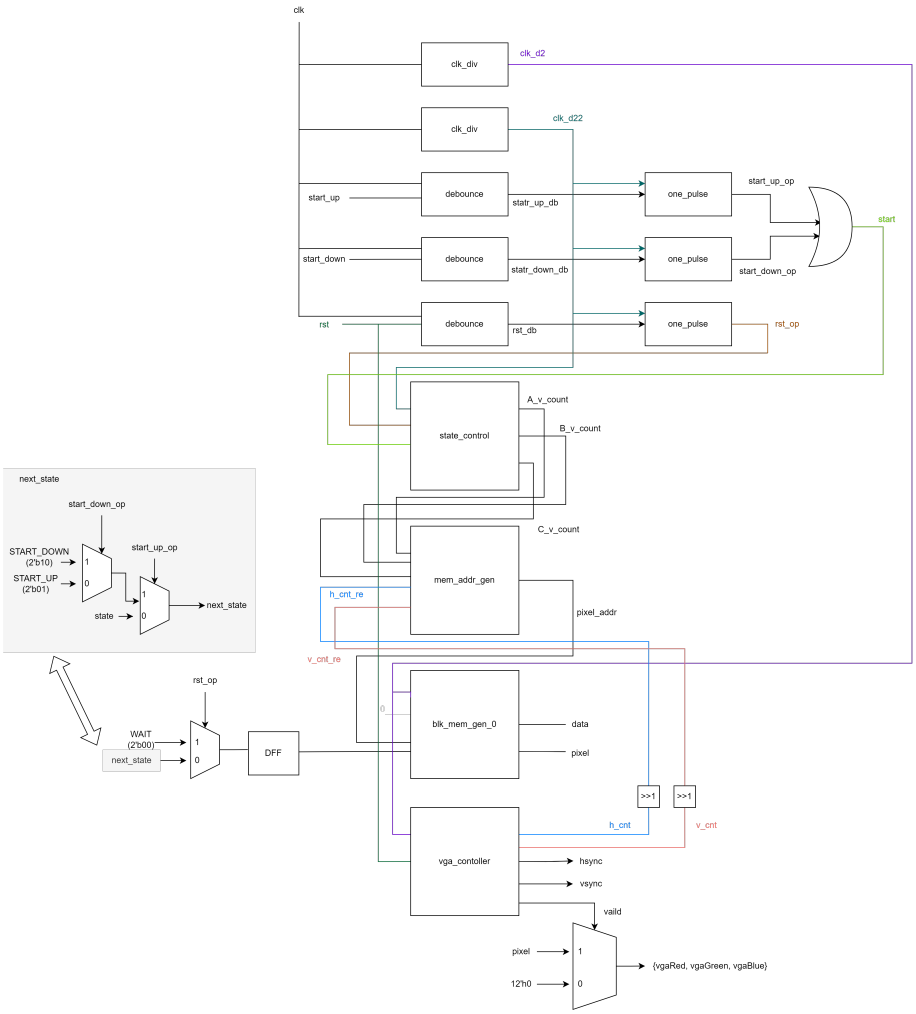
Output	package pin
AN[3]	W4
AN[2]	V4
AN[1]	U4
AN[0]	U2
segment[6]	U7
segment[5]	V5
segment[4]	U5
segment[3]	V8
segment[2]	U8
segment[1]	W6
segment[0]	W7

FPGA Demonstration 2

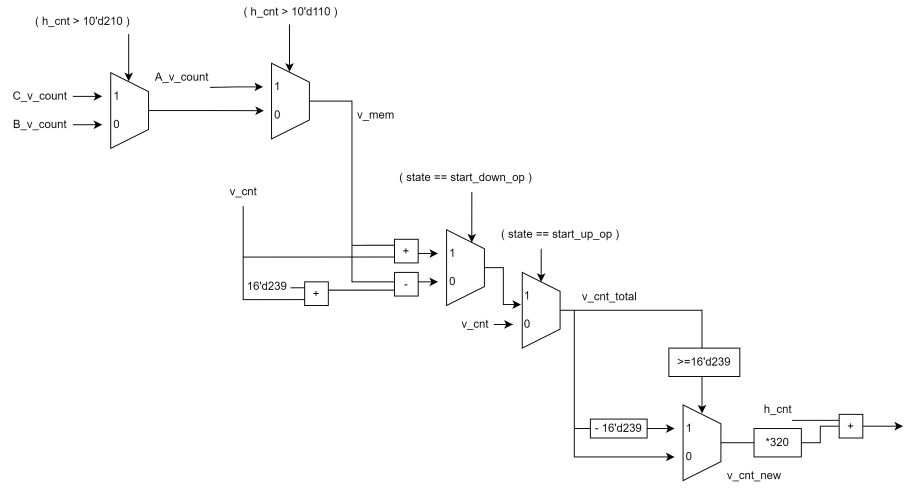
Drawing of the design of FPGA Demonstration 2

⛧ 畫出有修改過的 module

- Top module



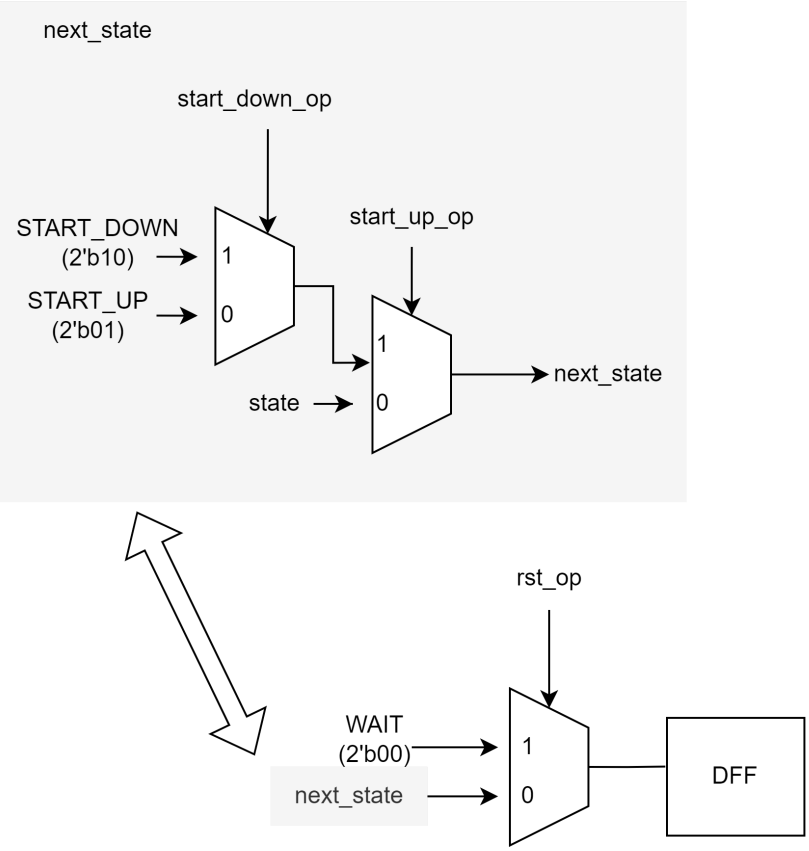
- mem_addr_gen



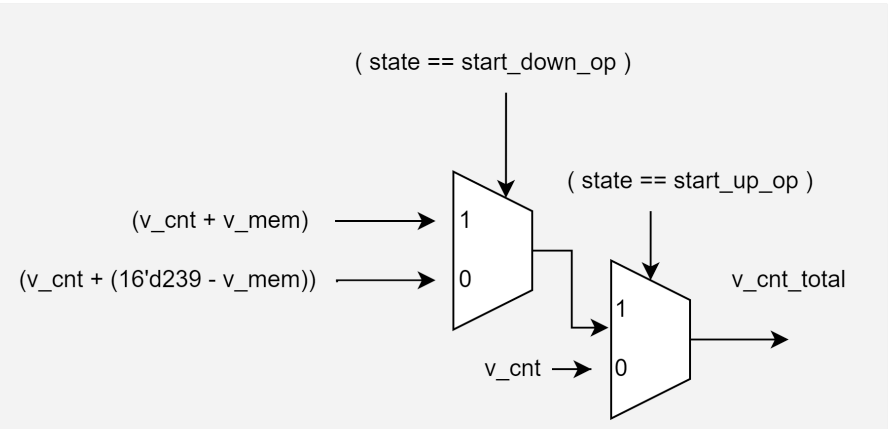
Design Explanation

1. 本來只有一個 start 按鈕，為滿足 spec 所要求的 slot machine 可以 run 以下兩個狀態而加上兩個按鈕 start_up, start_down ，並做 debounce & onepulse 處理後得到 start_up_op, start_down_op 兩個訊號。
    - upwared direction
    - downward direction

2. 按照是否 start_up_op, start_down_op 兩個訊號啟動 start 訊號於 state_control module 中。
3. 新增三個狀態，並依下圖做變化，此訊號將用於 mem_addr_gen module 中。
    - WAIT(2’b00)
    - START_UP(2’b01)
    - START_DOWN(2’b10)



4. 因 mem_addr_gen module 多了 state signal， v_cnt_total 改動按照下圖做變化



    - state = START_UP ： v_cnt 加上 v_mem

- state = START_DOWN ：v_cnt 減去 v_mem，並加回 16’239 確保值為正數
- state = 其他狀態：維持原本的 v_cnt

I/O pin assignment

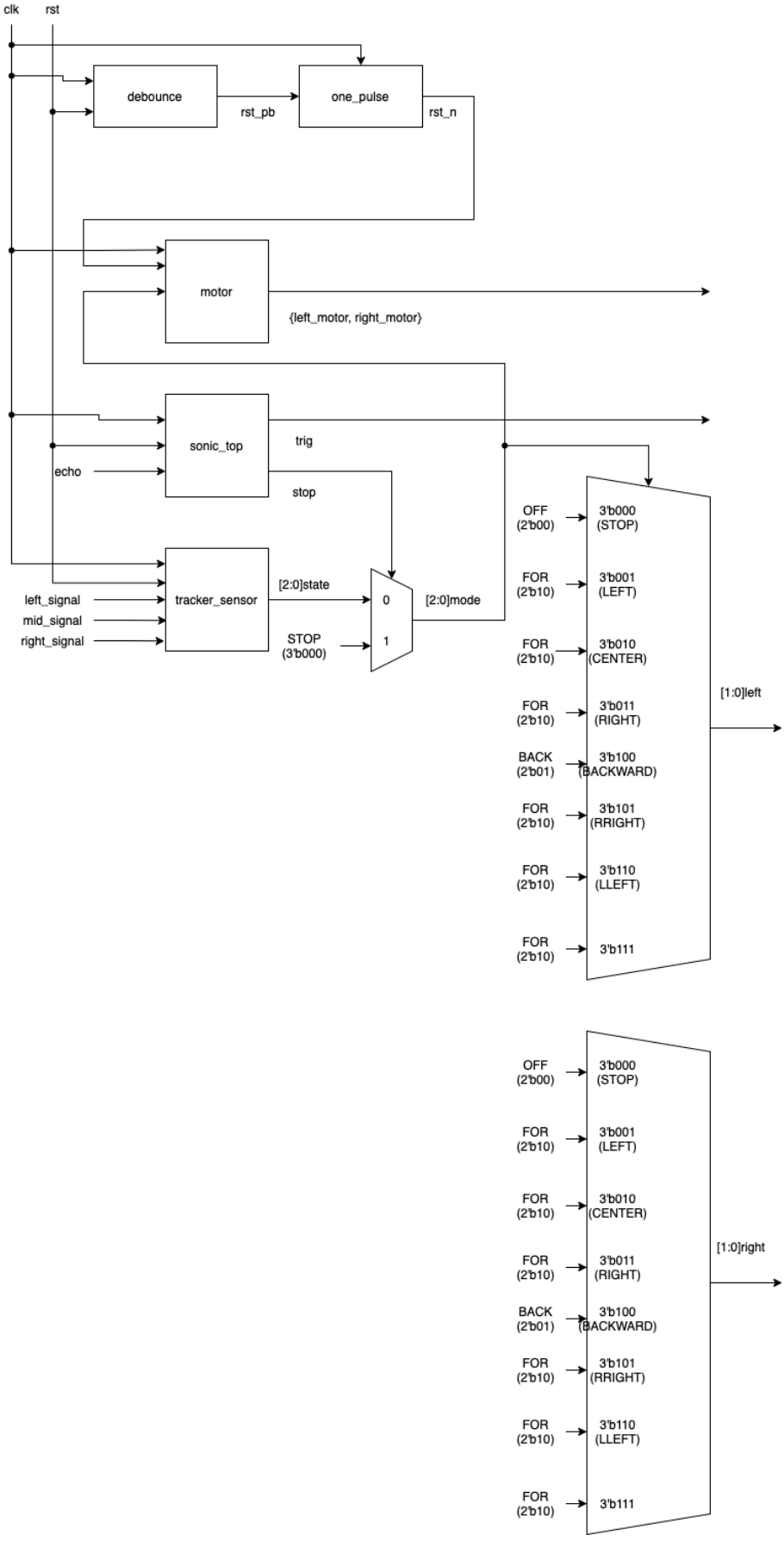
input	port
clk	W5
rst_n	T18
hsync	P19
vsync	R19
up_btn	W19
down_btn	T17

Output	package pin
vgaBlue[3]	J18
vgaBlue[2]	K18
vgaBlue[1]	L18
vgaBlue[0]	N18
vgaGreen[3]	D17
vgaGreen[2]	G17
vgaGreen[1]	H17
vgaGreen[0]	J17
vgaRed[3]	N19
vgaRed[2]	J19
vgaRed[1]	H19
vgaRed[0]	G19

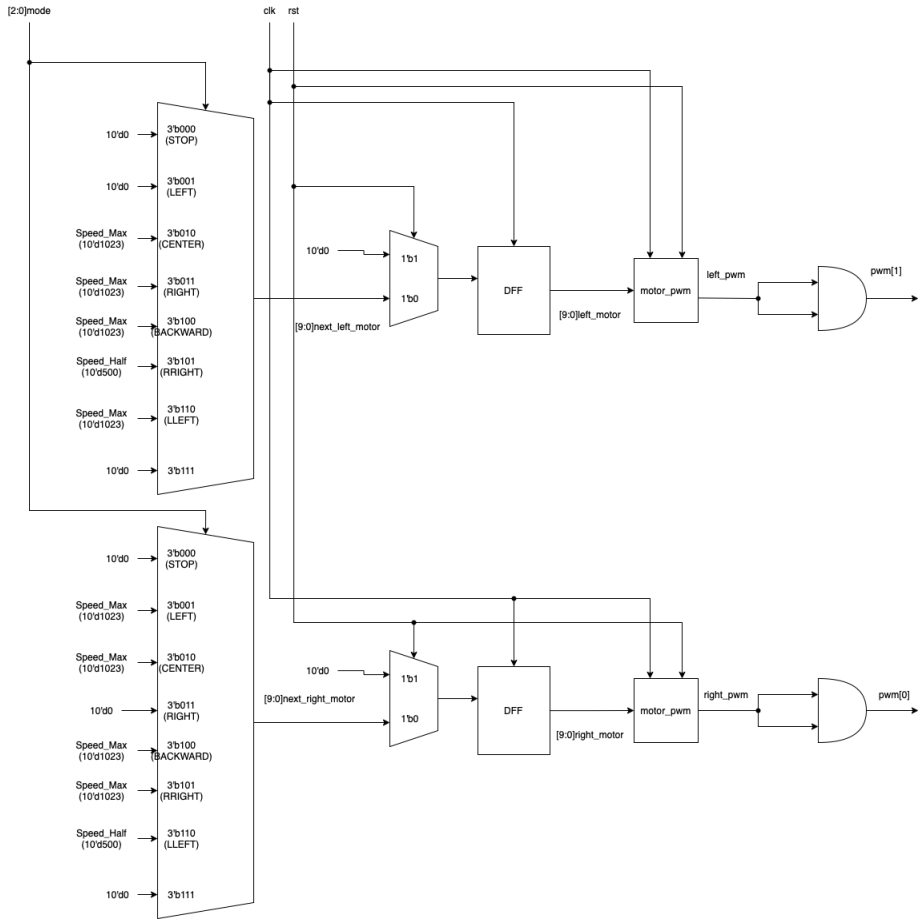
FPGA Demonstration 3

Drawing of the design of FPGA Demonstration 3

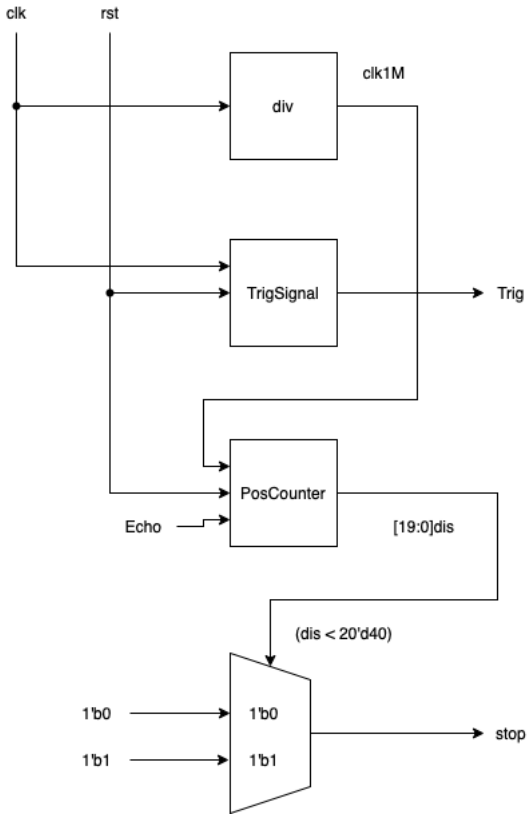
- Top module



- Motor module



- sonic_top



- tracker_sensor

Design Explanation

1. Top module
   在這裡我們把相對應的訊號接入各個module的input, output port，然後依據mode的值來給定left, right（left, right用來控制車車左右方向）的值，而mode的值代表的意思以及給定的left, right的值如下：

- mode == 3'b000(STOP)
  - 在這裡我們想要讓車車停止不前進，所以要讓馬達為off
    - left = OFF(2'b00), 依據car tutorial的指示，(input1,input2) == (0, 0)代表motor off
    - right = OFF(2'b00)
- mode == 3'b001(LEFT)
  - 在這裡我們想讓車車純左轉，但是要讓整台車移動必須要左右的馬達才行，所以都設為Forward(FOR)
    - left = FOR(2'b10), 依據car tutorial的指示，(input1,input2) == (1, 0)代表Forward
    - right = FOR
- mode == 3'b010(CENTER)
  - 在這裡我們想讓車車直走，但是要讓整台車移動必須要左右的馬達才行，所以都設為Forward(FOR)
    - left = FOR

        - right = FOR
- mode == 3’b011(RIGHT)
    - 在這裡我們想讓車車純右轉，但是要讓整台車移動必須要左右的馬達才行，所以都設為Forward(FOR)
        - left = FOR
        - right = FOR
- mode == 3’b100(BACKWARD)
    - 在這裡我們想讓車車往後退，但是要讓整台車移動必須要左右的馬達才行，所以都設為Backward(BACK)
        - left = BACK(2’b01), 依據car tutorial的指示，(input1,input2) == (0, 1)代表Backward
        - right = BACK
- mode == 3’b101(RRIGHT)
    - 在這裡我們想讓車車右轉，但是要讓整台車移動必須要左右的馬達才行，所以都設為Forward(FOR)
        - left = FOR
        - right = FOR
- mode == 3’b110(LLEFT)
    - 在這裡我們想讓車車左轉，但是要讓整台車移動必須要左右的馬達才行，所以都設為Forward(FOR)
        - left = FOR
        - right = FOR

而在left_motor以及right_motor，我們藉由trace motor module裡面的code可以發現pwm = {left_pwm, right_pwm}，而pwm正是用來控制車車左右馬達速度的，正好對應上left_motor以及right_motor的功能。
此外，在top module裡面，我們依據接到sonic_top module的output port的stop訊號的值來給定mode的值：

- stop == 1’b1: mode = STOP(3’b000)
- stop == 1’b0: mode = tracker_state(connected to output port of track_sensor module)

最後在trig的部分，我們把他接到sonic_top module的output port(接到Trig)並得到他的值。
2. Motor module
首先在[9:0]left_motor, [9:0]right_motor的部分，我們像在Top module裡一樣，依據mode的值來給定

[9:0]next_left_motor, [9:0]next_right_motor的值，並於postive trigger時update [9:0]left_motor, [9:0]right_motor，而詳細的情況如下。

- mode == 3'b000(STOP)
  - 在這裡我們想要讓車車停止不前進，所以車車的左右輪速度為0
    - [9:0]next_left_motor = 10'd0
    - [9:0]next_right_motor = 10'd0
- mode == 3'b001(LEFT)
  - 在這裡我們想讓車車純左轉，所以讓左輪的速度為0（原本的構想是因為想說要左轉應該要讓右輪去驅動車子就好了，但是後來發現要左轉是靠左輪去驅動的，所以我們後來把兩個motor接相反來達成左轉的功能）
    - [9:0]next_left_motor = 10'd0
    - [9:0]next_right_motor = Max_Speed(10'd1023)
- mode == 3'b010(CENTER)
  - 在這裡我們想讓車車直走，所以把左右輪速度開到最大
    - [9:0]next_left_motor = Max_Speed
    - [9:0]next_right_motor = Max_Speed
- mode == 3'b011(RIGHT)
  - 在這裡我們想讓車車純右轉，所以讓右輪的速度為0
    - [9:0]next_left_motor = Max_Speed
    - [9:0]next_right_motor = 10'd0
- mode == 3'b100(BACKWARD)
  - 在這裡我們想讓車車往後退，所以把左右輪速度開到最大
    - [9:0]next_left_motor = Max_Speed
    - [9:0]next_right_motor = Max_Speed
- mode == 3'b101(RRIGHT)
  - 在這裡我們想讓車車右轉，但是這裡是卡在比較緩的轉彎，所以讓右邊的車車速度為左邊的一半
    - [9:0]next_left_motor = Max_Speed
    - [9:0]next_right_motor = Max_Half(10'd500)
- mode == 3'b110(LLEFT)

    - 在這裡我們想讓車車左轉，但是這裡是卡在比較緩的轉彎，所以讓左邊的車車速度為右邊的一半
        - [9:0]next_left_motor = Max_Half
        - [9:0]next_right_motor = Max_Speed

3. sonic_top
   在這裡分為四個部分
   a.instantiate a div module to divide the system clock and get a clk with 1M Hz
   b. instantiate a TrigSignal module 然後把Trig接到output port
   c. instantiate a PosCounter module 然後把dis接到output port，而在這裡我們有對PosCounter內部做了細微更改，因為我們經過trace code之後發現，distance_register是用來計算從發出聲波(start == 1'b1 == echo_reg1 & ~echo_reg2, 其中echo_reg1是當前的echo訊號，而echo_reg2是上一個cycle時的echo訊號，所以當echo_reg1 == 1'b1 且 echo_reg2 == 1'b0代表此cycle開始發出聲波)到收到回聲(finish == 1'b1 == ~echo_reg1 & echo_reg2)所花的clock數量，而因為接入PosCounter module的clk是1M Hz的clock, 1M Hz = 10^(-6) sec, 所以distance_register所代表的就是以μs為單位的秒數，而我們知道聲波在空氣中的傳播速度為340(m/s)，所以我們可以得到下列式子並改寫模板的code:
    - dis(10^(-2)m) = distance_register(10^(-6)s)* 340(m/s) / 10^(4) / 2
      d. 最後再判斷dis的長度是否小於40(cm)，如果是的話就把stop的訊號拉起來，否則降下來
4. tracker_sensor
   在這裡我們依據{left_signal, mid_signal, right_signal}的訊號來給定state的值

- 3'b000:代表前面沒有路，所以往後退
    - state = BACKWARD;
- 3'b001:代表前面只有右轉的路
    - state = RIGHT;
- 3'b010:代表前面只有中間的路
    - state = CENTER;
- 3'b011:代表前面可能為往右的弧形道路，並非絕對地向右
    - state = RRIGHT;

- 3'b100:代表前面只有左轉的路
  - state = LEFT;
- 3'b101:因為在car tutorial中寫到沒有square corner的情形，所以在這裡我們直接右轉（但其實左轉也可以）
  - state = RIGHT;
- 3'b110:代表前面可能為往左的弧形道路，並非絕對地向左
  - state = LLEFT;
- 3'b111:在這裡是十字路口，依據car tutorial，此時要直接向前走
  - state = CENTER;

I/O pin assignment

input	port
clk	W5
rst	U18
echo	P18
left_signal	B16
mid_signal	B15
right_signal	A16

output	port
trig	N17
left_motor	J1
left[1]	L2
left[0]	J2
right_motor	H1
right[1]	K2
right[0]	H2

Contribution

施泳瑜 : FPGA1, 2, 3 implementation, FPGA1, 2report
朱季葳 : FPGA1, 2, 3 implementation, FPGA3report

What we have learned from Lab6

1. 這次在車車這題真的栽在蠻多細節上的，尤其是發現車子轉彎所需要的馬達跟我們預想不同的時候，因為我們下意識地覺得車子要右轉就應該驅動左邊的馬達，結果後來發現事實並非如此，所以我們就在接線上面做了一些小改動，還有在右轉左轉的部分，原本我們只有設置前後左右四個state，但是發現這樣在轉彎的時候車子需要不斷地倒車然後改方向的幅度非常小，所以最後才再加入兩個state(LLEFT, RRIGHT)來使得車子在轉彎上面比較順。