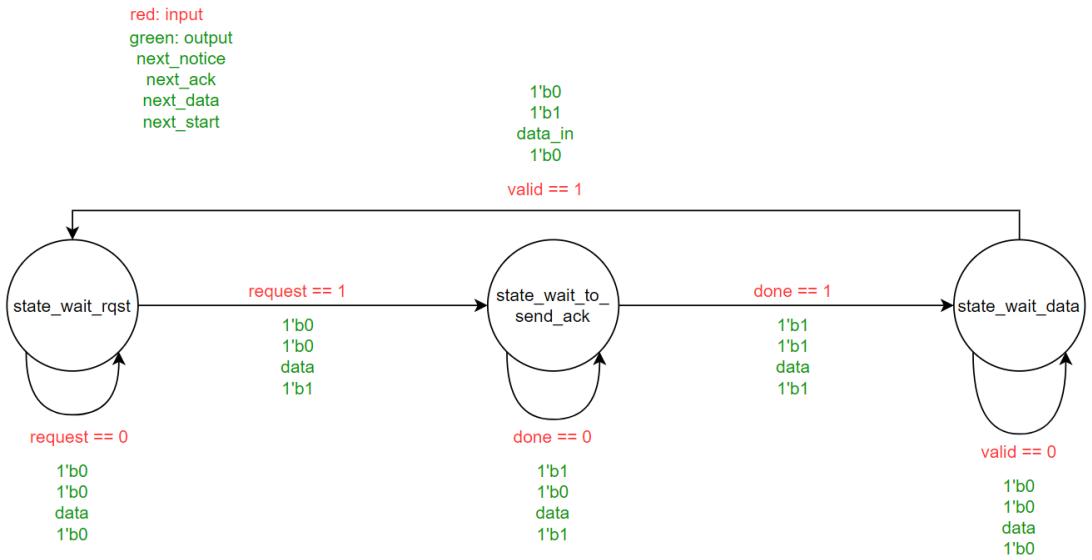
Lab 6 report

組員：110062221 李品萱
110062213 唐翊雯

I. Dual FPGA communication

這題要設計一個 FPGA-to-FPGA 的 communication protocol，基本的步驟與 state spec 都已經給了，template 也完成了大部分功能，我們需要做的部分只有將 slave FPGA 有關 state 的 code 完成，這部分也只要寫出每個 state 的 output 以及 next state 為何即可，其 state diagram 如下：

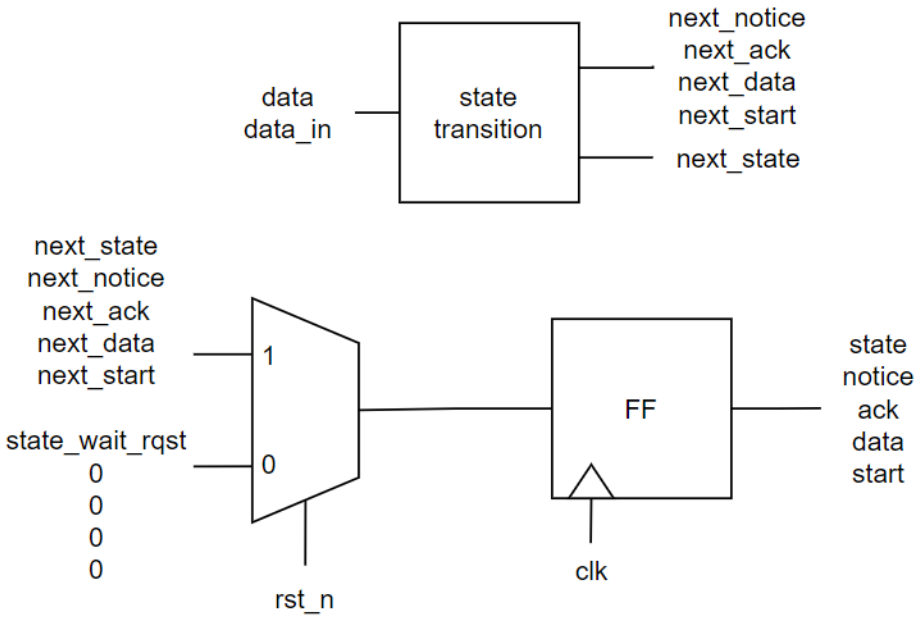


首先在 wait request 的時候，如果 request 為 0，則繼續維持原 state 等待 request，這時 next_notice, next_ack 與 next_start 都設為 0，next_data 則維持為 data。而若 request 為 1，則 next_state 要變成 state_wait_to_send_ack，next_start 也要轉為 1’b1，開始為 notice 亮的時間計時。

在 state_wait_to_send_ack 這個 state 時，若 done 為 0，代表 notice 亮的時間還未結束，此時 next_state 一樣為 state_wait_to_send_ack，next_ack 仍為 0、next_data 仍維持 data，next_start 與 next_notice 則繼續保持為 1’b1。而若 done 為 1，next_state 要變為 state_wait_data，next_ack 要變為 1’b1。

最後在 state_wait_data 這個 state，已經不需要用到 notice 與 start，因此 next_notice 與 next_start 均設為 0。而在 valid 為 0 時，next_state 要維持在 state_wait_data、next_ack 設為 0、next_data 仍設為 data。在 valid 變為 1 後，next_state 變為 state_wait_rqst、next_ack 再次設為 1、next_data 設為 data_in，吃新輸入的 data 進來。

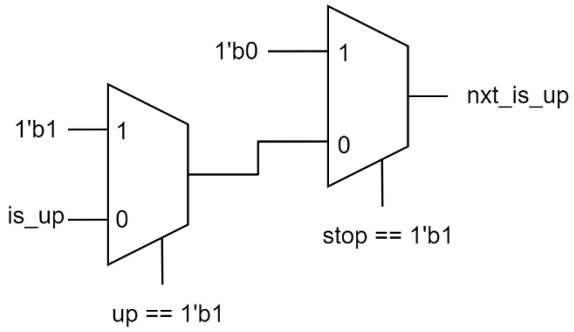
這個 module 的 block diagram 如下：



II. The slot machine

這題我們需要實作一個拉霸機，在 sample code 的部分已經幫我們寫好大部分的功能，我們只需要修該 code 使其作出往上跑的部分。Trace 了一下 sample code 之後，我們先將控制向上的按鈕做 debounce 及 onepulse，然後將處理過的 up_op 接到 state control 的 module 內，在這個 module 內，我們用一個 is_up 的 reg 去記現在的方向（1 為 up，0 為 down），如下圖（next_is_up 會在 sequential block 內接到 is_up）。

```
assign nxt_is_up = ((start == 1'b1) ? 1'b0 : (up == 1'b1 ? 1'b1 : is_up));
```



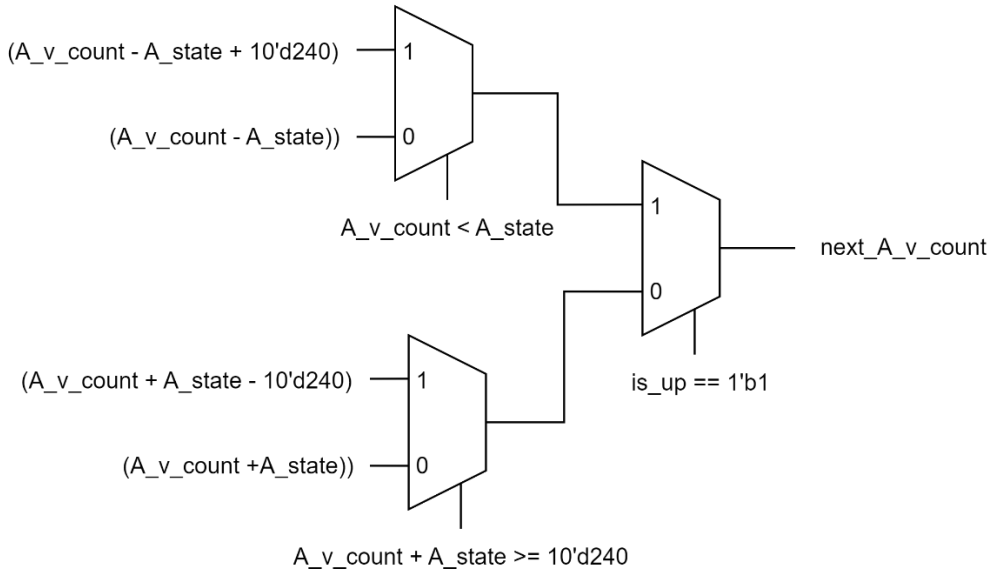
接下來我們需要修改配合給定方向做顯示的部分，也就是 next_A_v_count、next_B_v_count、next_C_v_count。首先我們用 is_up 判斷現在的方向，如果向上的話接下來需要判斷邊界情況，由於是以向下為正、向上為負，因此向上的部分必須判斷他是正的，維護他在顯示的合法範圍。這部分在 sample code 做的向下的操作也有類似的處理，而我們對像上的操作如下圖。

```
((is_up == 1'b1) ? ((A_v_count < A_state) ? (A_v_count - A_state + 10'd240) : (A_v_count - A_state))
((is_up == 1'b1) ? ((B_v_count < B_state) ? (B_v_count - B_state + 10'd240) : (B_v_count - B_state))
((is_up == 1'b1) ? ((C_v_count < C_state) ? (C_v_count - C_state + 10'd240) : (C_v_count - C_state))
```

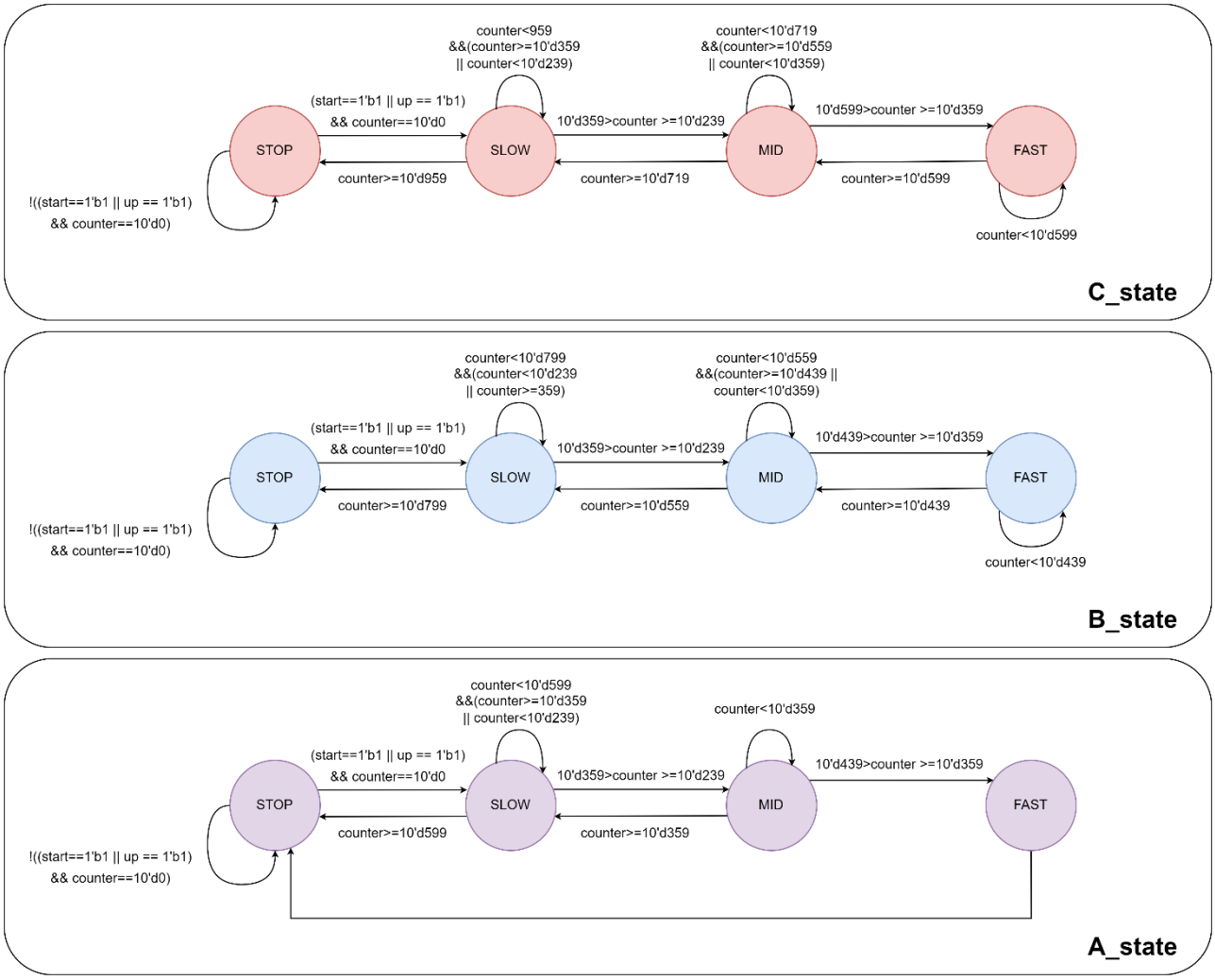
而 sample code 內的操作如下圖，因為我們的 v 方向最大值是 240，因此需要維護他在向下加的過程中是在合法的顯示範圍內。

```
: ((A_v_count + A_state >= 10'd240)? (A_v_count + A_state - 10'd240): (A_v_count + A_state)));
: ((B_v_count + B_state >= 10'd240)? (B_v_count + B_state - 10'd240): (B_v_count + B_state)));
: ((C_v_count + C_state >= 10'd240)? (C_v_count + C_state - 10'd240): (C_v_count + C_state)));
```

next_A_v_count（next_B_v_count、next_C_v_count 同理）的 block diagram 如下圖。



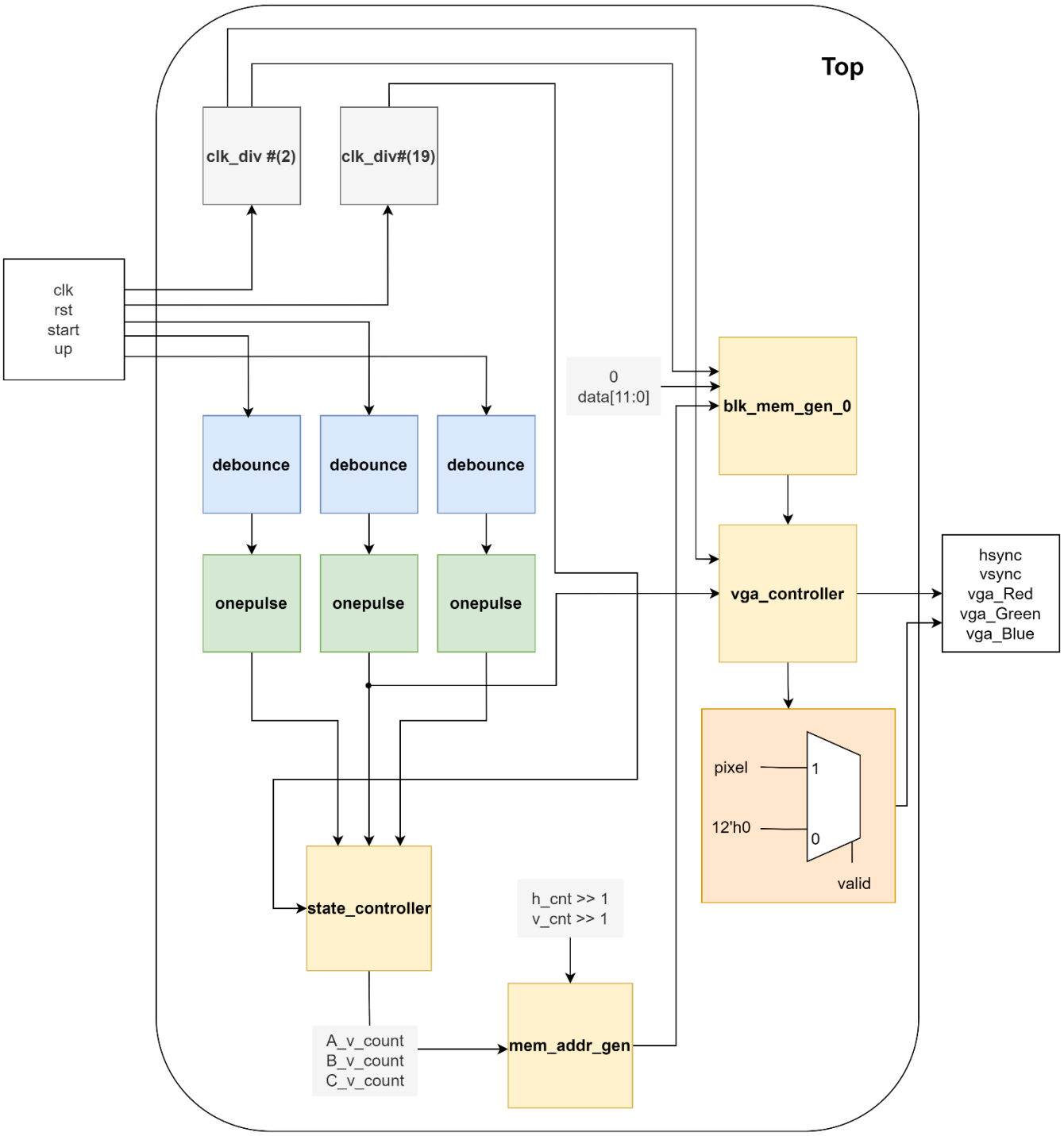
在顯示速度的部分，我們只對 A、B、C 的 stop state 加上 up == 1’b1 的判斷條件，這邊的 state diagram 如下圖。



而為了讓 slot machine 可以連續觸發，不須按下 reset 才能跑下一次，我們修改了 counter 的部分，讓他在符合條件時歸零，如下圖。

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

本題的 block diagram 如下。



III. The car

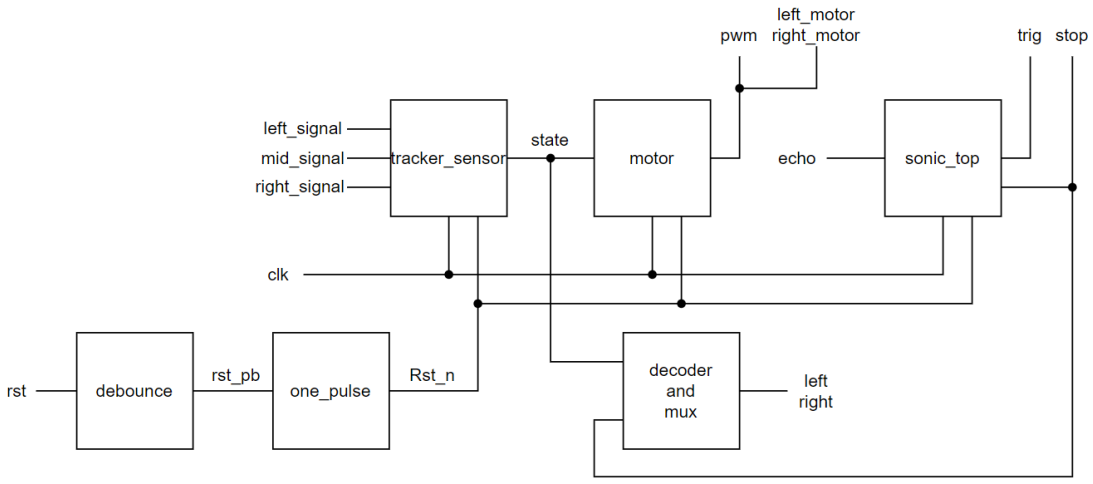
這題要完成一個自走車，在 code 的部分，主要要完成下列三個目標：

- 將 top module 裡面各個用到的 module 接好，使車車能正常運作
- 完成 motor 與 tracker_sensor 兩個 module，使車車能偵測並走在白色賽道上
- 完成 sonic_top 這個 module，使車車遇到前方障礙物時自動停下

以下將分別解釋三部分的實作。

- **top module**

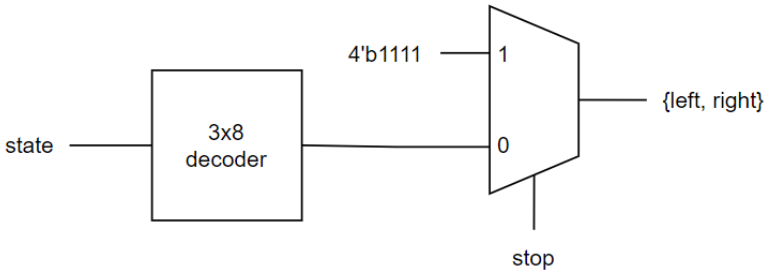
  首先要將 tracker_sensor、motor 與 sonic_top 三個主要 module 接好。除了 tracker_sensor 的 state(output) 與 motor 的 mode(input)以外，其他都照 top module 自己的 input 與 output 接好就好。而 tracker_sensor 的 state 會根據接收到的 left_signal、mid_signal 與 right_signal，輸出相對應的 state 後再傳給 motor，讓 motor 根據不同的 state 做不同的事情。這個 module 整體的 block diagram 如下：



  而 left 與 right 的部分，我們根據 stop 的值與現在的 state 來決定。首先如果 stop 為 1，則車子要停下來。而如果 stop 為 0，我們則會根據現在的 state 決定兩個輪子要如何移動，具體的移動方式如下：

State	左輪	右輪
直走	向前	向前
左轉	向前（速度較慢）	向前（速度較快）
右轉	向前（速度較快）	向前（速度較慢）
左急轉	向後	向前
右急轉	向前	向後
向後	向後	向後

  而根據 car tutorial，若希望輪子向前，需要將 output 設為 2’b10；若希望輪子向後，需要將 output 設為 2’b01；若希望輪子停下，則要將 output 設為 2’b11。我們使用 case 完成這部分的 code，block diagram 如下：

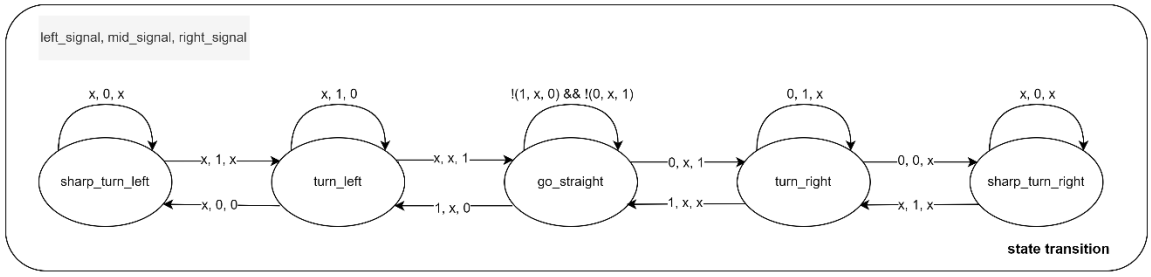


為了方便 debug 與優化車子的速度，我們多接了一些 input 與 output 來顯示與控制各種資訊。我們分別將 left_signal、mid_signal、right_signal、left、right 與 stop 使用 LED 燈顯示出來。stop 的部分，我們還另外開了一個 wire: stop_ctrl，使我們可以用 switch 控制是否要開啟 stop 的功能，方便我們測試。我們也使用 7-segment display 來 output 車子現在的 state 為何，方便我們做改進。

- **motor & tracker_sensor**

  I. tracker_sensor

  首先在 tracker_sensor 的部分，會藉由紅外線偵測車車是否在正確的賽道上，紅外線 snesor 偵測到黑色會回傳 0，偵測到白色會回傳 1，一開始我們使用的方法是用 input 信號直接 output 車子的 state，但這樣會使車子在急轉時需要比較多的時間。因此為了讓車子在轉彎時較為順利，我們嘗試用 FSM 讓車子做出對應的行為，如下圖。

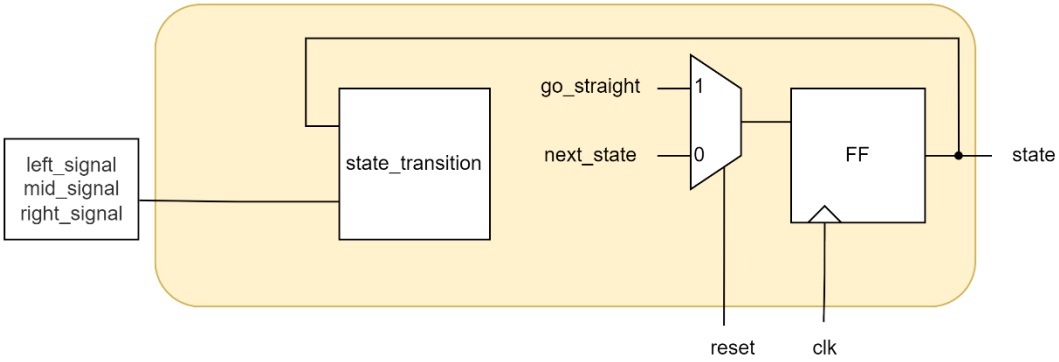


上圖中首先我們看 go_straight，車子進到這個 state 的條件是偵測到全白，而我們讓這個 state 不會是急轉的下一個或上一個 state，因此我們只看左邊跟右邊的訊號，若都是 1 則直走，而在 demo 時為了通過 bonus 的賽道 intersection 的部分，我們確定直走、轉彎與急轉的狀況我們都已經維護好，因此不會有衝出白色賽道的狀況，所以我們嘗試讓車子 default 是直走，也成功通過該賽道。

轉彎的部分我們以左轉說明，首先在 turn_left 的部分，因為在這個 state 內 left_signal 會是 1，所以我們不判 left_signal，若 mid_signal 為 0 則為左急轉，若為 1 則維持一般的左轉；而若 right_signal 為 1 則代

表已經轉完，回到直走的狀況。接下來在左急轉的部分，由於只由 mid_signal 決定是否為急轉，所以我們不對另外兩個 signal 進行判斷，若 mid_signal 讀到黑色代表需要急轉，讀到白色則代表進入左轉的 state，這麼做的原因是，我們發現如果只依靠 input signal 判斷的話，由於 default 是直走，在轉彎時會反覆在轉彎、直走、急轉三個 state 之間一直變換，轉速變化會落差太大造成轉彎不流暢，而修改過後則可以限制車車下一個 state 的行為，讓他在轉的過程再去判斷自己要不要急轉，運作上流暢許多。右轉、右急轉原理相同這邊就不再贅述。

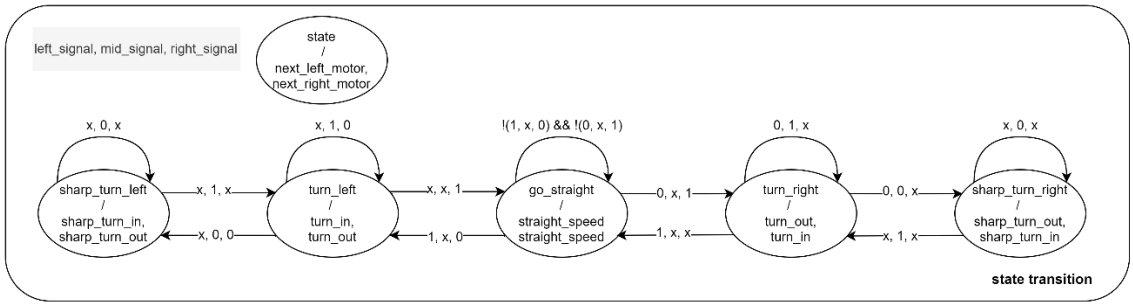
該 module 的 block diagram 如下。



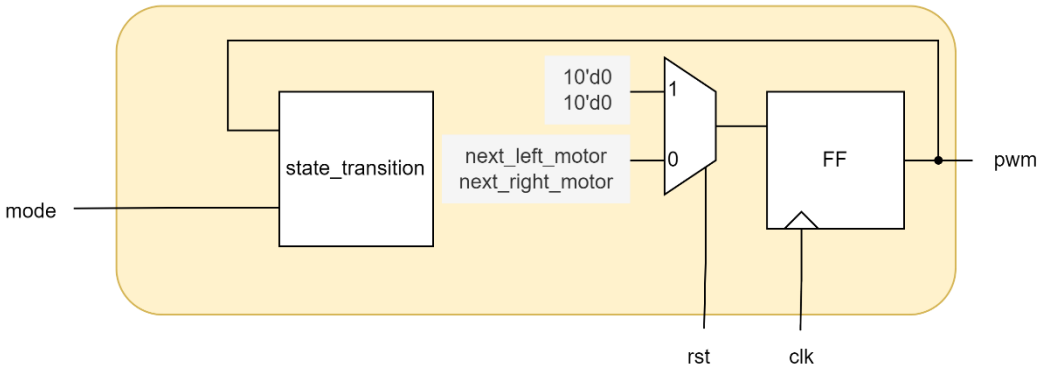
II. motor

這個 module 會決定輪子的轉速，我們需要做的是根據每個 state 給出相對應的參數，一開始不知道該給多少，大概 trace 了一下確定他是給越大的數值會有越大的轉速，在嘗試了幾次後我們觀察到 left_motor 及 right_motor 大概在 700 會是緩慢的行進，5、600 幾乎不會動。

接下來由於兩個輪子是各自供應動能，與坦克的行進原理很像，因此我們試著將坦克轉彎的方法應用到車車上面。我們將直走與急轉的 left_motor 及 right_motor 都設為 10'd1023，一般轉彎的內輪設為 10'd723，這麼做即可使轉彎時兩輪 motor 相差 10'd300，而急轉彎時因為在 top module 內我們已經使此時輪子是一輪向前，一輪向後，因此急轉時兩輪速差會是最大，即為 motor 相差 10'd2046。這個部份我們花了許多時間調整數值：應該讓轉彎時速差小一點以維持最高均速，還是讓轉彎轉完整一點以最大化走直線（車速會最快）的時間？這個問題困擾了我們一段時間，而以上是我們嘗試出來認為能均衡兩者並讓車車盡量發揮其最大效能的參數。在調參數的過程中，為了方便起見，我們用 parameter 的方式給出轉彎、急轉彎與直走時內外輪的各自數值，motor 的 state diagram 如下圖。



該 module 的 block diagram 如下圖，

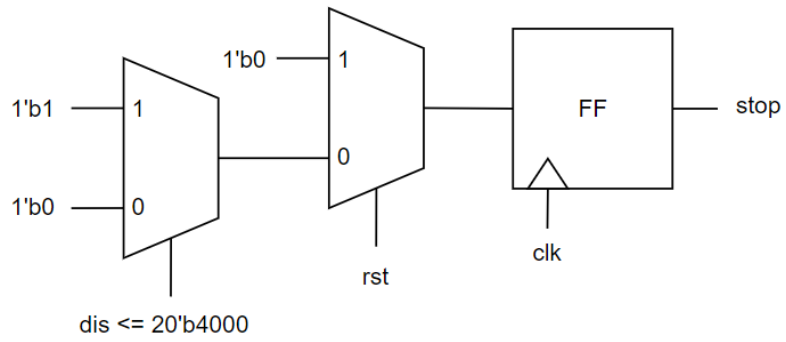


- **sonic_top**

  判斷是否要停下，只需要根據 dis 的大小，即與前方障礙物的距離，來決定 stop 的值即可。首先在 reset 的時候 stop 要設成 1’b0，而若 dis <= 20’d4000（這是反覆測試得出的結果），即 spec 規定的 40cm，則車子要停下來，stop 設成 1’b1，反之車子要繼續前進，stop 一樣設為 1’b0。

```
always@(posedge clk) begin
    if(rst) stop <= 1'b0;
    else if(dis <= 20'd4000) stop <= 1'b1;
    else stop <= 1'b0;
end
```

這部分的 block diagram 如下：



- **Problems solving**

在做這一題的過程中，我們主要遇到了兩個大問題：
第一個是車子在跑的時候，總是只能左轉或是只能右轉。這讓我們感到很疑惑，因為在 code 中控制兩輪的 code 應該非常相似且對稱，沒道理一邊能跑另一邊不行。我們反覆檢查 code 所有可能出問題的地方，也再三確認過線沒有接錯。在多次試驗後才發現是 3-Way Line Tracking IR 出了問題，left_signal 與 mid_signal 永遠被偵測為 1。在丟棄兩塊 sensor 後，車子的行走才變為正常，在鎖 sensor 時還因為鎖成靠車身的洞造成車車會偵測太慢而衝出賽道，後來發現後才成功解決。這也教會我們在遇到 bug 的時候要大膽假設小心求證，給我們上了一課。

第二個問題是車子在直走時，會自動偏向某一邊，導致車子在不該轉彎時轉彎。因此我們在 demo 前花了不少時間調整直走時的餒外輪速度差讓車車可以走直線，但不知道為什麼兩輪的速度差在不同天表現出來卻不一樣，因此讓我們在 debug 時找不出問題，非常困擾，尤其在跑 bonus 賽道時車子換了一顆全新的電池又可以正常的以相同轉速走直線，使得我們原本為了讓他走直線而調整的參數反而是錯誤的。

IV. Summary

這次的三題都是 FPGA 的題目，我們練習到了更多 FPGA 板的應用。

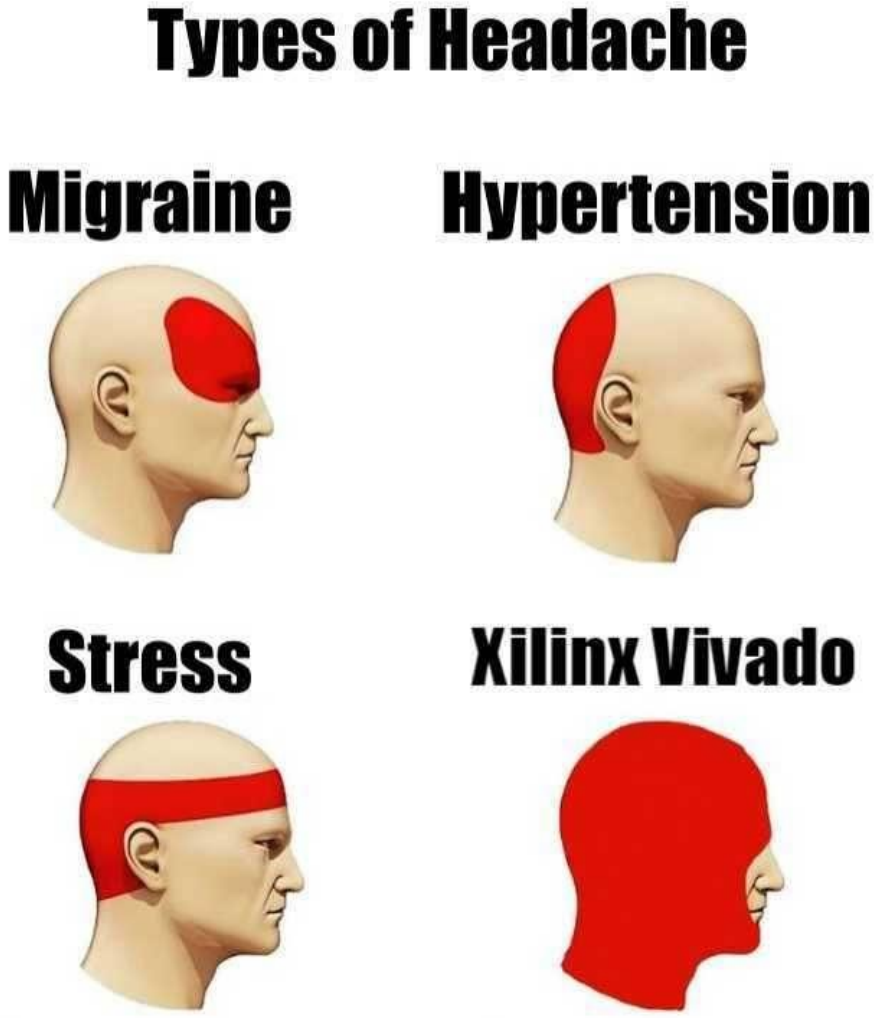
在 chip2chip 這一題，我們練習了如何在兩個板子之間傳遞訊號。傳遞 data 與 ack 的做法恰好與我現在在計算機網路概論上學到的資料傳輸方法非常有關聯，讓我覺得很有趣。

在 slot machine 的部分，我們學到了螢幕的 display，第一次接觸這部分讓我對螢幕有更進一步的了解，看到 slot machine 可以正常運作也覺得很酷很有成就感。

車子的部分花了不少時間，由於我們的電池盒兩條線都斷掉了，因此學會了剪漆包線的技巧，換 sensor 也讓我們學會了這個新技能，以及一開始換電池時總是要推很久，後來熟能生巧就可以很快換好，這次 Lab 除了 code 本身也讓我們學到其他的小東西，更能夠體驗到實際上的硬體操作，下圖是汰換的電池和 sensor 們，以茲紀念。

自己做出能完整跑完整個賽道的車子很有成就感，看到優化過的 code 可以讓車子更流暢的跑也是，不過一直因為硬體設備碰壁一度令人感到相當煩躁與挫折，如下圖：

也因為這樣，我們花費大把時間在盯著沒有出錯的 code，令人相當頭痛，如下圖。



整體來說，這次 Lab 需要考慮到比較多的東西，出現 bug 的時候要用適當的方法找出問題點，例如我們採取的接出 output 讓它顯示在板子上等等，也需要仔細觀察才不會被自己的主觀認知誤導，這部分的確花了我們不少時間，但也有所收穫，在擬定行進策略的部分，很幸運因為 final project 想做坦克所以有事先查了相關的資料，因此對由兩個馬達驅動的車車有較為明確的概念，雖然後來發現輪子沒辦法拆下來所以無法做出真正的坦克，但這個先備知識還是讓我們在操作上稍有餘力。這次 Lab 也特別感謝同學的十字起支援及 demo 的電池支援。

V. Contributions

- **Code**

  Dual FPGA communication by 李品萱

  The slot machine by 唐翊雯

  The car 偵測障礙物 by 李品萱，偵測賽道 by 唐翊雯，debug 及測試由兩人共同完成

- **Report**

  各自描述負責的題目