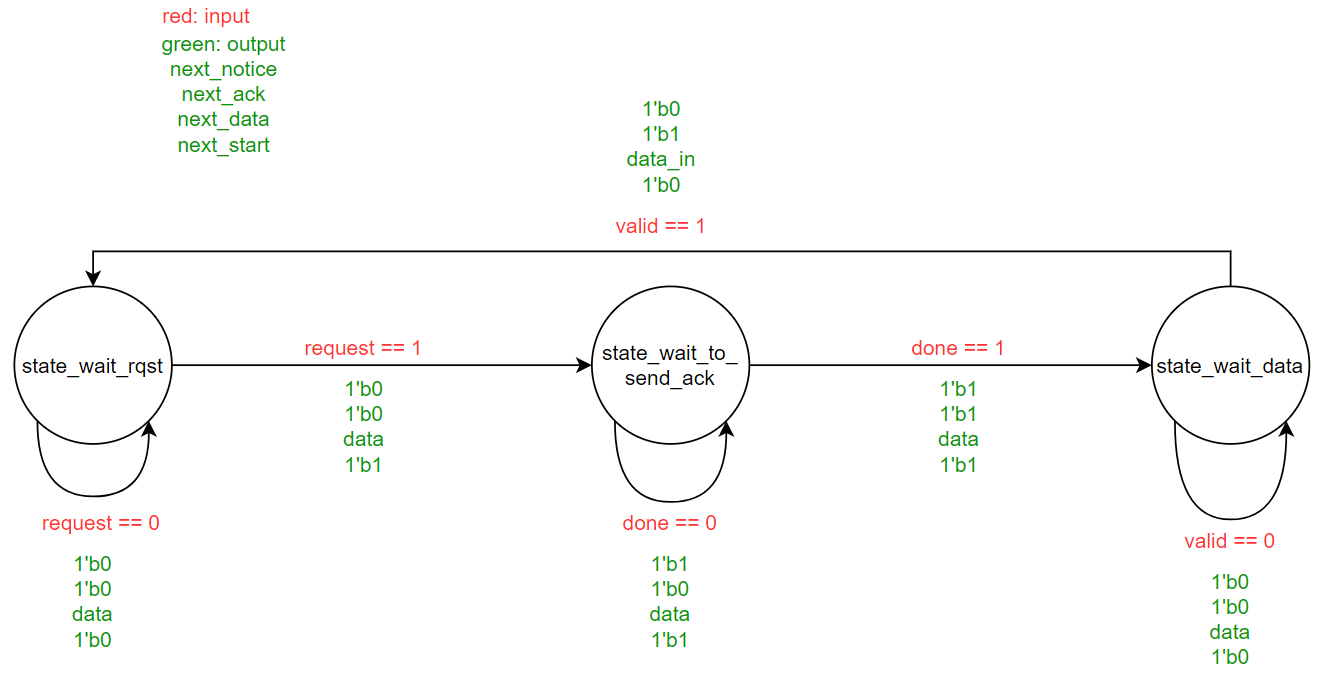
Lab 6 report

組員：110062221李品萱

110062213唐翊雯

1. **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：

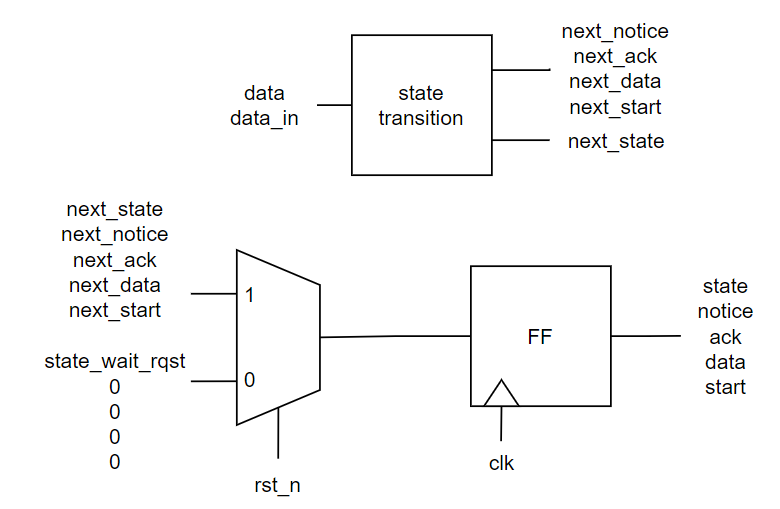


首先在wait request的時候，如果request為0，則繼續維持原state等待request，這時next\_notice, next\_ack與next\_start都設為0，next\_data則維持為data。而若request為1，則next\_state要變成state\_wait\_to\_send\_ack，next\_start也要轉為1’b1，開始為notice亮的時間計時。

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：

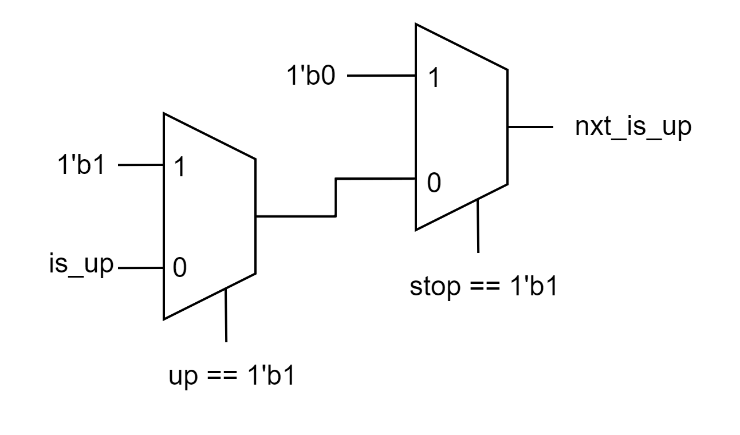


1. **The slot machine**

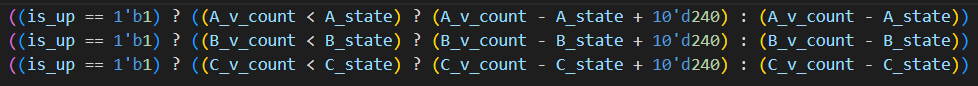
這題我們需要實作一個拉霸機，在sample code的部分已經幫我們寫好大部分的功能，我們只需要修該code使其作出往上跑的部分。Trace了一下sample code之後，我們先將控制向上的按鈕做debounce及onepulse，然後將處理過的up\_op接到state control的module內，在這個module內，我們用一個is\_up的reg去記現在的方向（1為up，0為down），如下圖

（next\_is\_up會在sequential block內接到is\_up）。

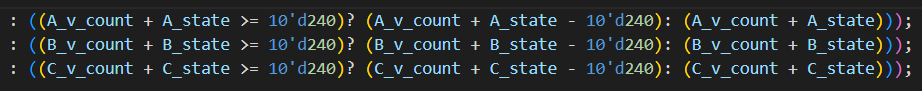




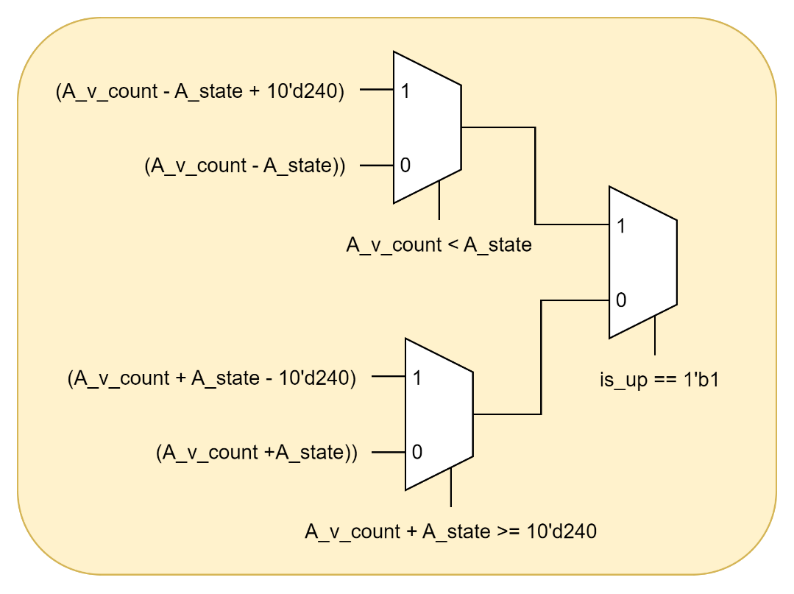
接下來我們需要修改配合給定方向做顯示的部分，也就是next\_A\_v\_count、next\_B\_v\_count、next\_C\_v\_count。首先我們用is\_up判斷現在的方向，如果向上的話接下來需要判斷邊界情況，由於是以向下為正、向上為負，因此向上的部分必須判斷他是正的，維護他在顯示的合法範圍。這部分在sample code做的向下的操作也有類似的處理，而我們對像上的操作如下圖。

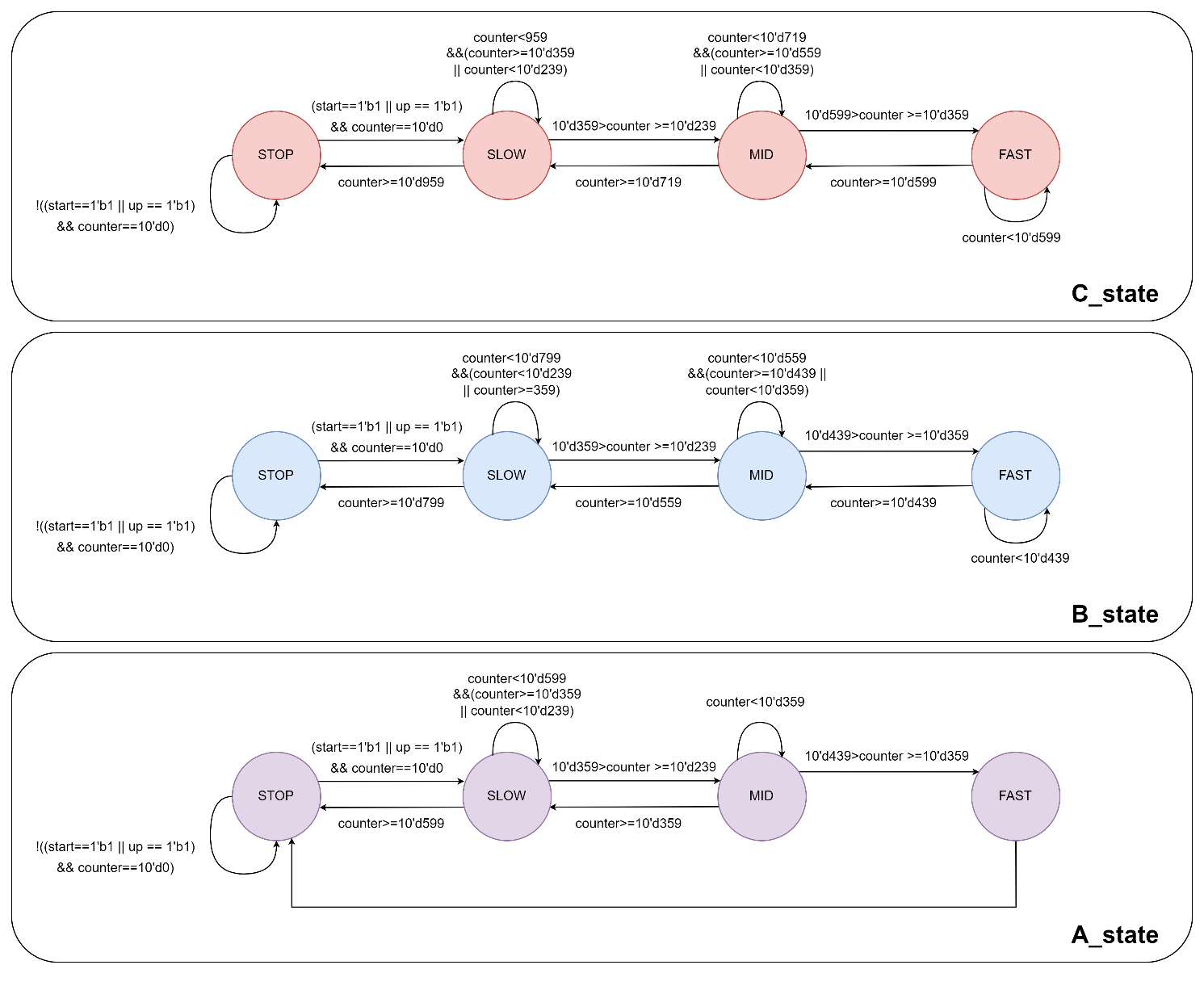


而sample code內的操作如下圖，因為我們的v方向最大值是240，因此需要維護他在向下加的過程中是在合法的顯示範圍內。

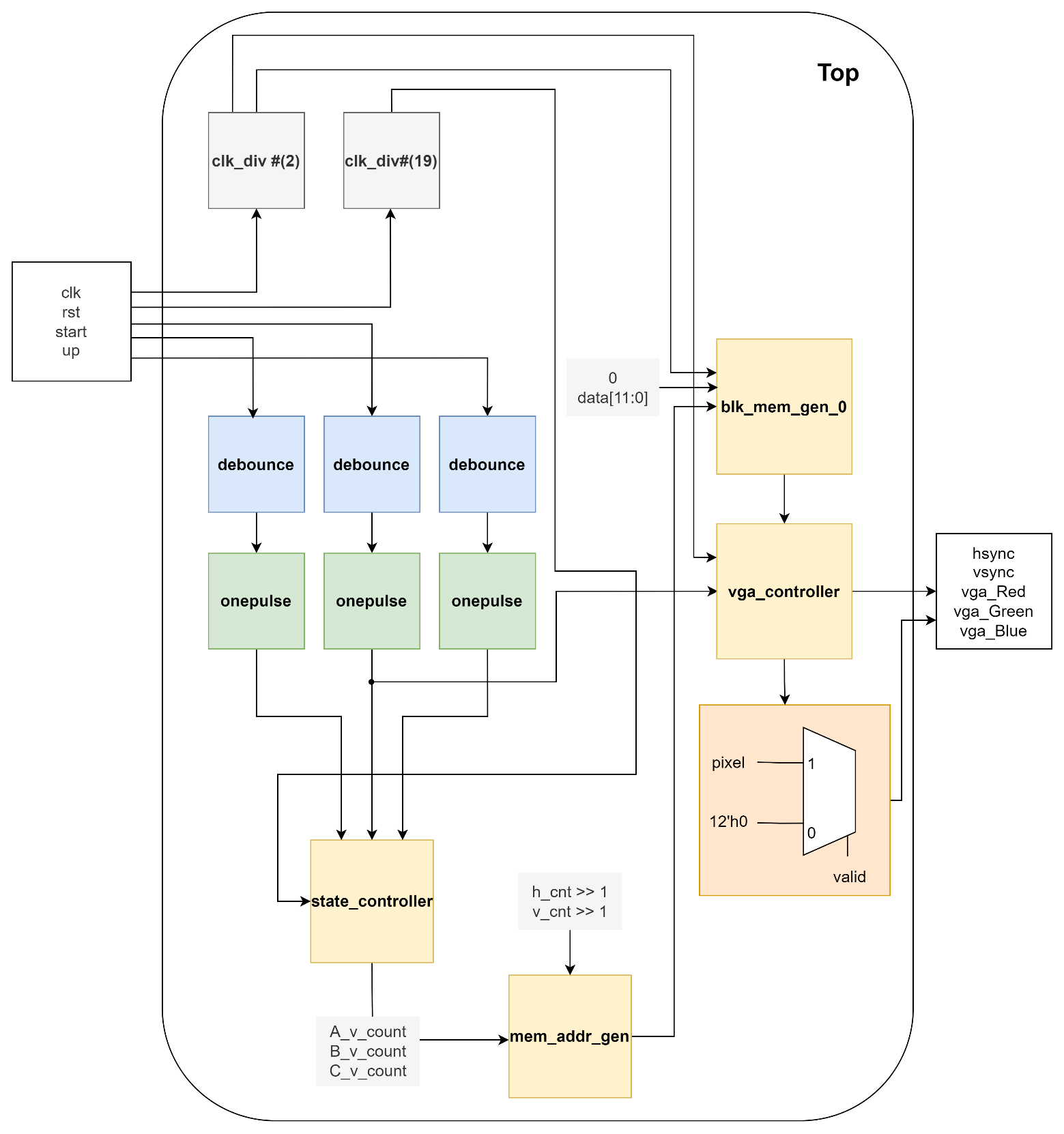


next\_A\_v\_count（next\_B\_v\_count、next\_C\_v\_count同理）的block diagram如下圖。



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。

本題的block diagram如下。

1. **The car**

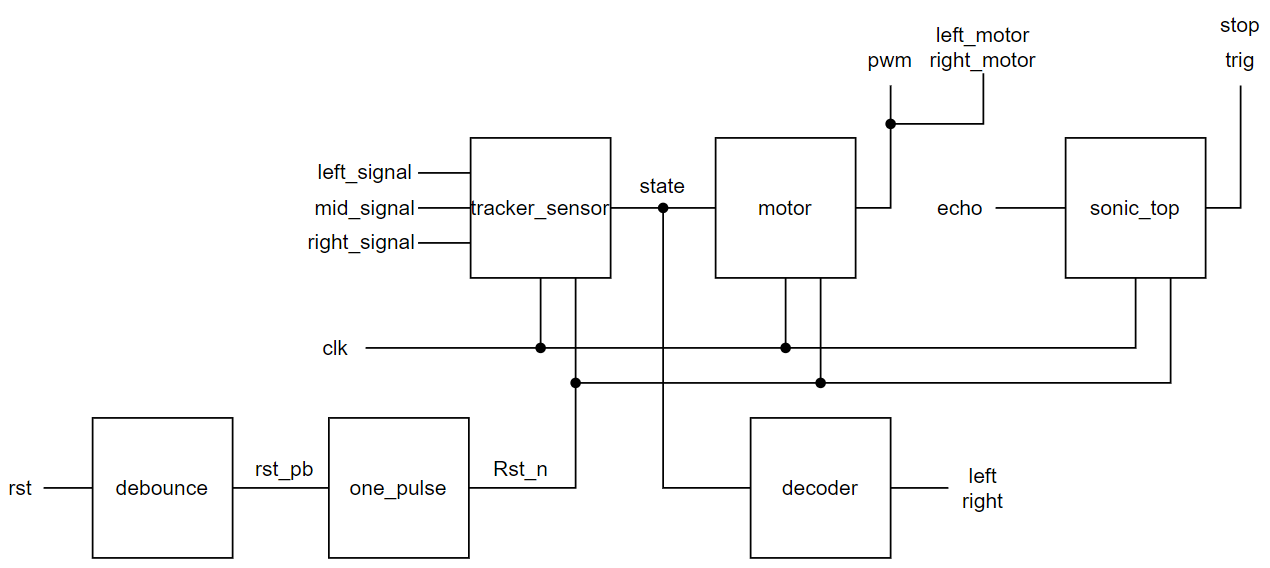
這題要完成一個自走車，在code的部分，主要要完成下列三個目標：

* 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。

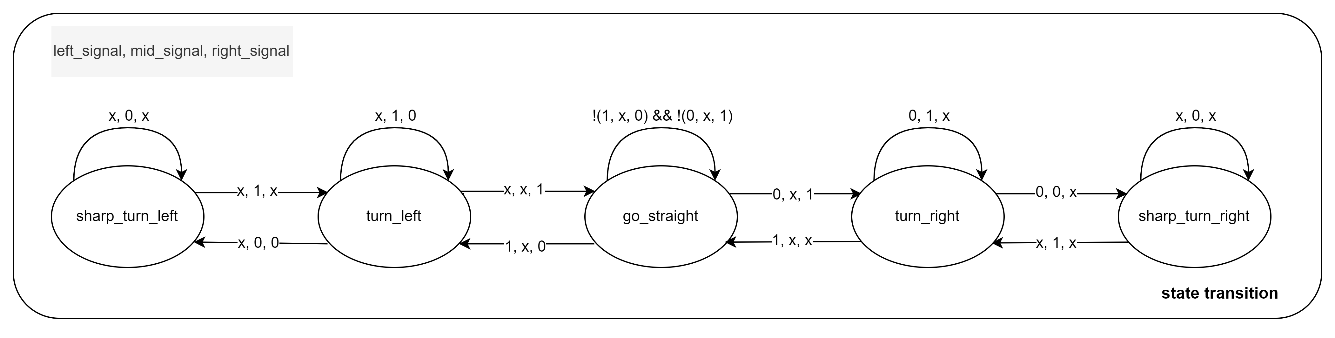
* top module

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：



* motor & tracker\_sensor

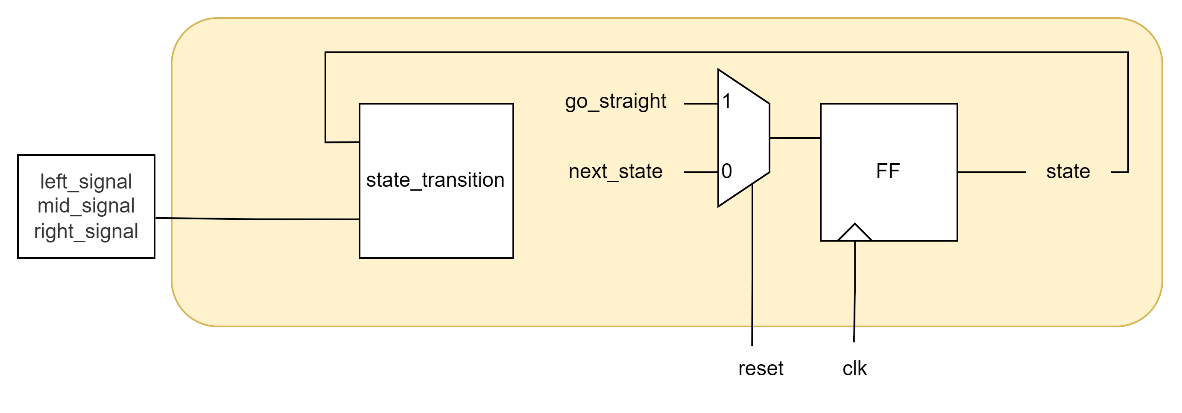
首先我們看tracker\_sensor的部分，這邊會藉由紅外線偵測車車是否在正確的賽道上，紅外線snesor偵測到黑色會回傳0，偵測到白色會回傳1，一開始我們使用的方法是用input信號直接output車子的state，但這樣會使車子在急轉時需要比較多的時間。因此為了讓車子在轉彎時較為順利，我們嘗試用FSM讓車子做出對應的行為，如下圖。



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next 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，因為直走、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。

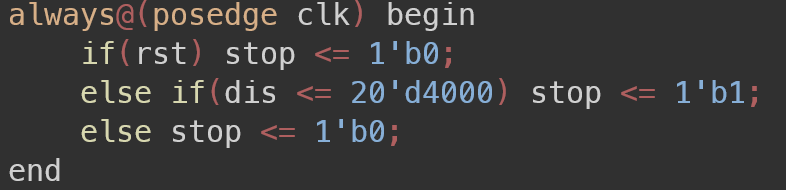
轉彎的部分我們以左轉說明，首先在turn\_left的部分，因為在這個state內left\_signal會是1，所以我們不判left\_signal，若mid\_signal為0則為左急轉，若為1則維持一般的左轉，而若right\_signal為1則代表已經轉完，回到直走的狀況。接下來在左急轉的部分，由於只由mid\_signal決定是否為急轉，所以我們不對另外兩個signal進行判斷，若mid\_signal讀到黑色代表需要急轉，讀到白色則代表進入左轉的state，這麼做的原因是，我們發現如果只依靠input signal判斷的話，由於default是直走，在轉彎時會反覆在轉彎、直走、急轉三個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，運作上也流暢許多。右轉、右急轉原理相同這邊就不再贅述。

該module的block diagram如下。

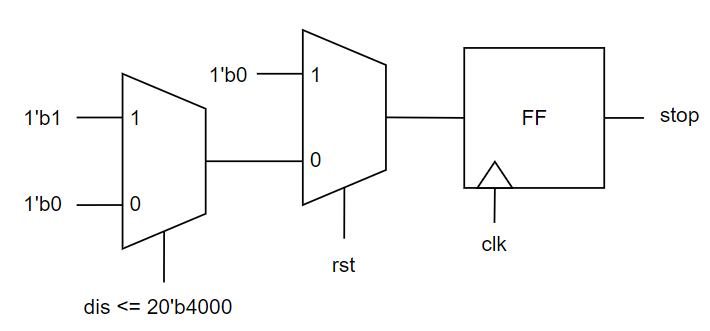


* sonic\_top

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。



這部分的block diagram如下：



1. **Summary**



1. **Contributions**

* **Code**

Dual FPGA communication by 李品萱

The slot machine by唐翊雯

The car

* **Report**

各自描述負責的題目