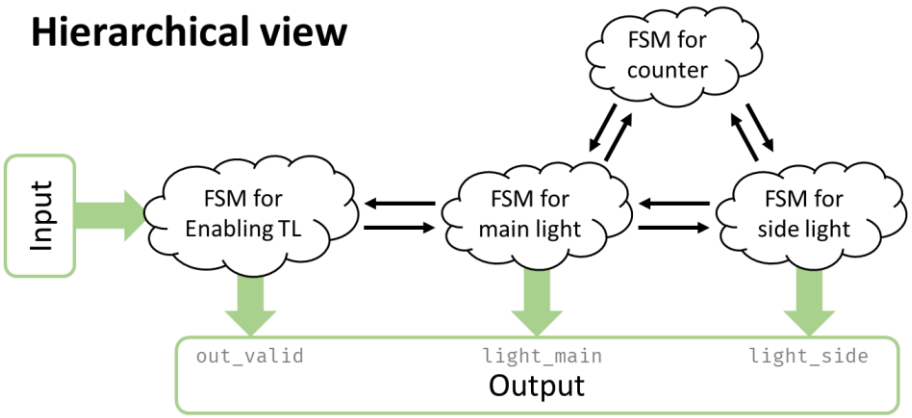
數位電路與系統 HW3 Report

110511233 李承宗

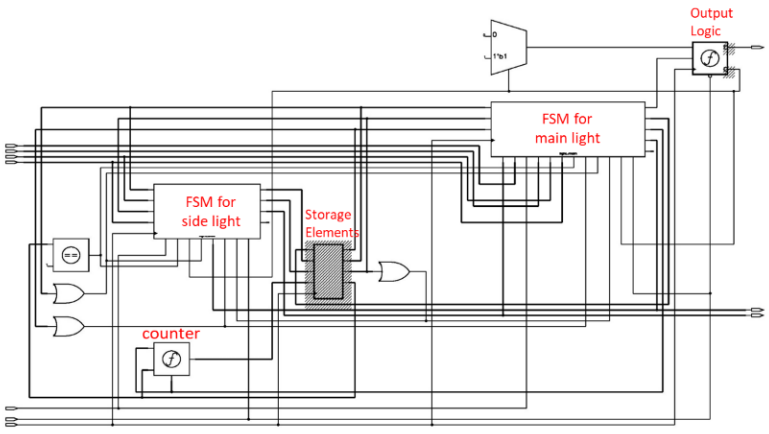
作業三的 FSM 設計難度比 HW2 以及我在修邏輯設計時寫過的題目要難上許多，所以我寫這次作業的時間幾乎都是花在設計、修改 FSM。看完助教提供的影片之後，我開始使用階層式的 FSM 來設計這次作業，但可能我對這次作業的邏輯以及控制流程比較沒感覺，所以很多時候都是發現測資沒過、遇到 corner case，我才知道有少考慮到的情況。因此這次我是先拼拼湊湊將 code 寫完，再來進行優化，設計出這次作業的 FSM。

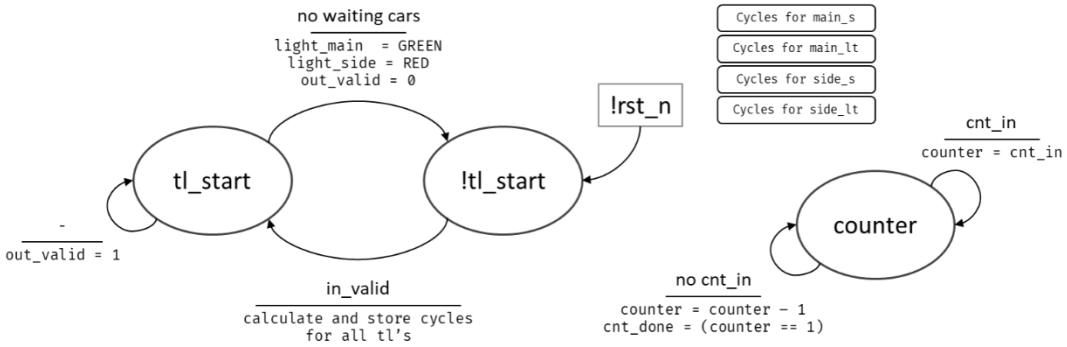
在經過一番嘗試之後，我設計出來的 FSM 整體的階層關係如下圖所示，主要分成：用來啟動紅綠燈（處理輸入）、計算倒數時間、控制*light_main*以及控制*light_side*這四個部分的狀態機。



輸入會經由最上層（邏輯上）的紅綠燈控制器來處理，並經由一些邏輯運算來決定出主幹道的狀態、流程。得到控制器提供的資料之後，*light_side*會維持在紅燈，同時主幹道的控制器就會開始運作（最開始一定會是主幹道的紅綠燈先動作）。其中會調用*counter*來計算經過時間，以及在主幹道的流程結束之後，*light_main*會維持在紅燈，並將控制權交給支幹道的控制器。整個流程結束（包括支幹道）之後會由主幹道的控制器來送出結束訊號，並且讓整個紅綠燈回到原始的狀態（等待*in_valid*和資料輸入），一個週期的流程至此結束。

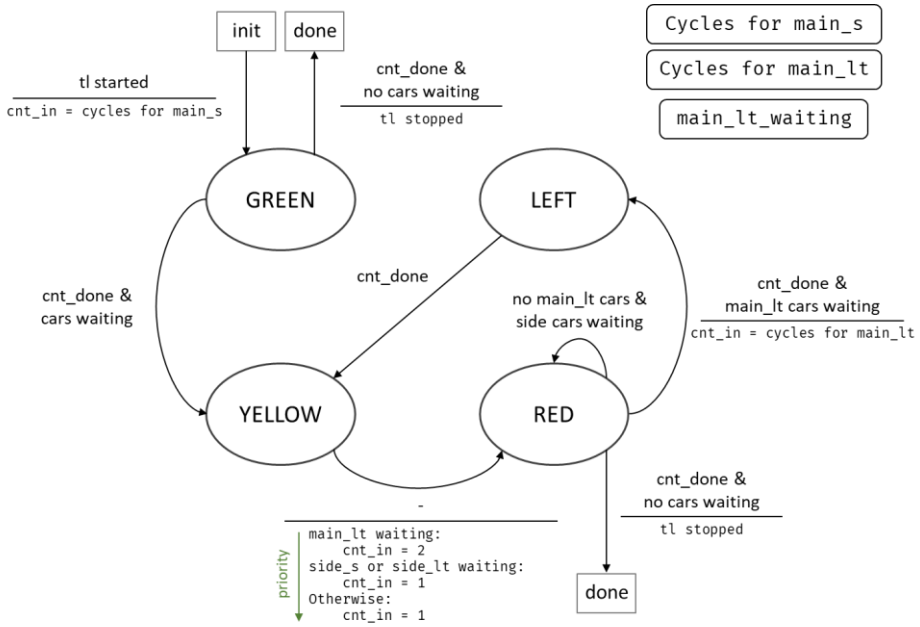
接下來是各階層 FSM 設計的部分。（下圖為 Verdi 輸出的 schema）





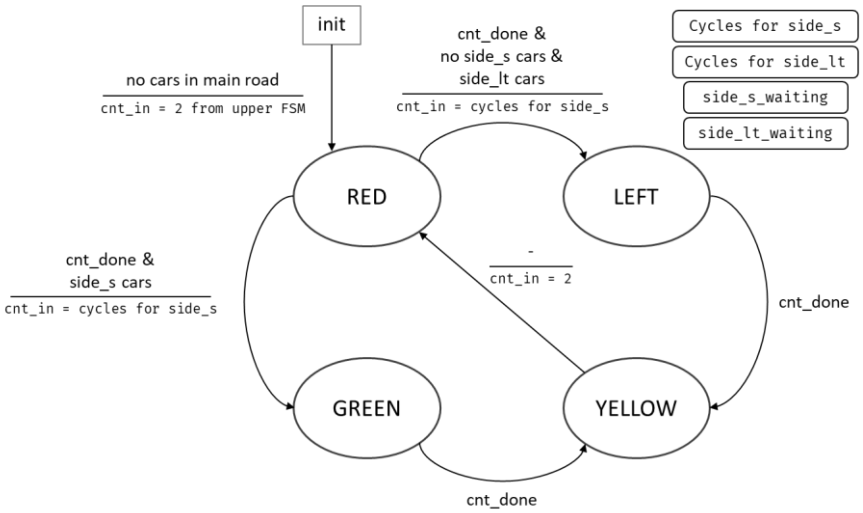
上圖是紅綠燈控制器以及*counter*的*state diagram*。可以看到紅綠燈控制器的部分我使用了*tl_start*變數來控制狀態，初始狀態會在!*tl_start*的部分，在接收到*in_valid*訊號之後，資料會經過一系列的處理（存成 n 的整數倍、輸入為 0 時要做特殊處理等等…）並儲存起來（右上四個*label*所相對應的資料），並且將狀態轉為*tl_start*。因為*out_valid*要在資料處理完之後才能拉到*high*的狀態（不能跟*in_valid*重疊），因此會在切換狀態之後才開始控制。

*Counter*的部分就相對的比較簡單了。為了節省資源，所以這個*counter*比較陽春：設計是直接利用輸入的值來控制狀態，沒有額外的*enable*輸入；邏輯也非常簡單，只要一直執行減 1 的動作即可。會讓$cnt_done = (counter == 1)$是因為這裡的輸出實際上會影響到的是下一個 cycle 的狀態，因此*counter*會在輸入資料之後開始倒數，在數到 1 時，即是 n+1 狀態（次態）時的倒數完畢。



接下來是主幹道控制器的部分，右上角為使用到的一些變數，其中兩個*cycles count*變數為輸入，一個*waiting*是這個 FSM 會控制的變數（沒有*main_s_waiting*是因為主幹道一進來就會是綠燈，所以不需要額外的變數來判斷綠燈的狀態是否已經執行。），可以用來在 state transition 的時候判斷是否要

出現左轉等號，簡化狀態機的邏輯。*Init*是從前一個 FSM 送出的控制訊號，開始的時候會將變數準備好，並且設定*counter*的值來開始倒數時間。由於會出現輸入全為 0，只執行1 *cycle*的情況，因此可能在綠燈的時候就跳出流程。黃燈的地方只需要切換到紅燈即可，不需要計數，但是接下來的紅燈可能會把控制權交給下一層的支幹道控制器，因此會有兩種不同的 case 產生（有點複雜，但我不知道要怎麼改了@@）：用來銜接左轉燈或是支幹道的紅綠燈。在紅燈狀態下如果只剩下支幹道的車，主幹道就會維持紅燈，並在流程結束之後跳出。切換到左轉的邏輯非常簡單，區別是需要控制*waiting*變數來分辨是否亮過左轉燈號。



最後一個是支幹道的控制器，由於到這裡的時候整個流程只剩下支幹道的燈號需要控制，因此邏輯會比主幹道的控制器要簡單很多，但因為支幹道的輸出不一定會亮綠燈，所以得多一個*side_s_waiting*變數來判斷。其他切換的狀態就非常簡單：只需要照著流程，並且完成最後亮兩個 cycle 的紅燈，剩下的部分交給上面兩層的 FSM 即可。

這次的作業相較於前兩次難度算是提升很多，加上我剛開始設計的時候沒看清楚題目跟輸出的要求，導致我在寫的時候還會邊給自己挖坑。寫完這次作業讓我感受到了 pattern 的重要性。幸好有了助教提供的 pattern，我才知道原來這個題目並沒有我最初想像中的那麼簡單，也不至於讓我設計出來的東西跟助教想要的差距太大。設計途中我也發現其實我對階層式 FSM 的設計並沒有那麼熟悉，很常寫一寫就出現兩個 FSM 交纏在一起的情況：當我想改其中一個地方的時候，另外一邊的 FSM 就會亂掉。做報告的時候，我也察覺到我設計出來的架構並沒有那麼的“階層式”，反而更像三四個串接在一起的 FSM，但是這次作業剛好夾在我幾科期中考之間，我也只能將就於此了，之後有時間的話再來多練練階層式 FSM 的設計吧。

終於在 deadline 當天的半夜把作業的 code 跟報告趕完了，能夠放心睡覺了。