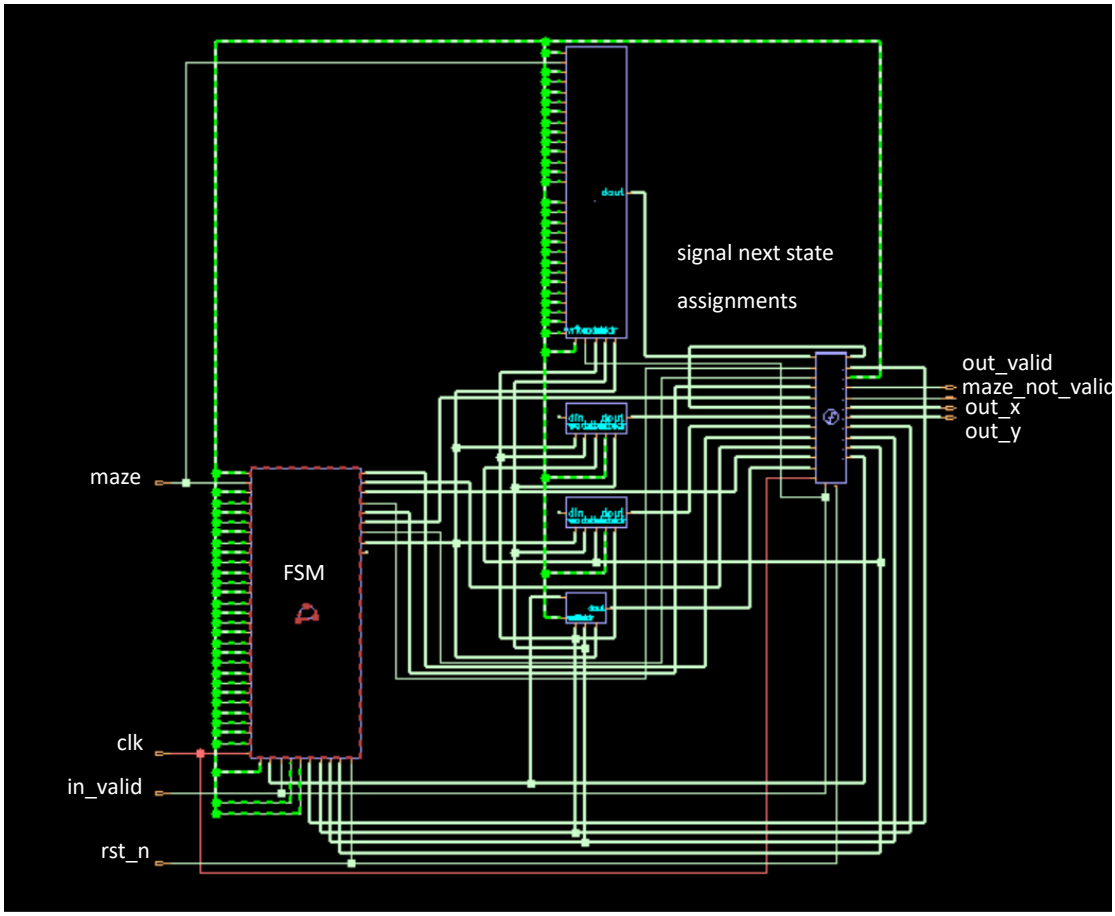
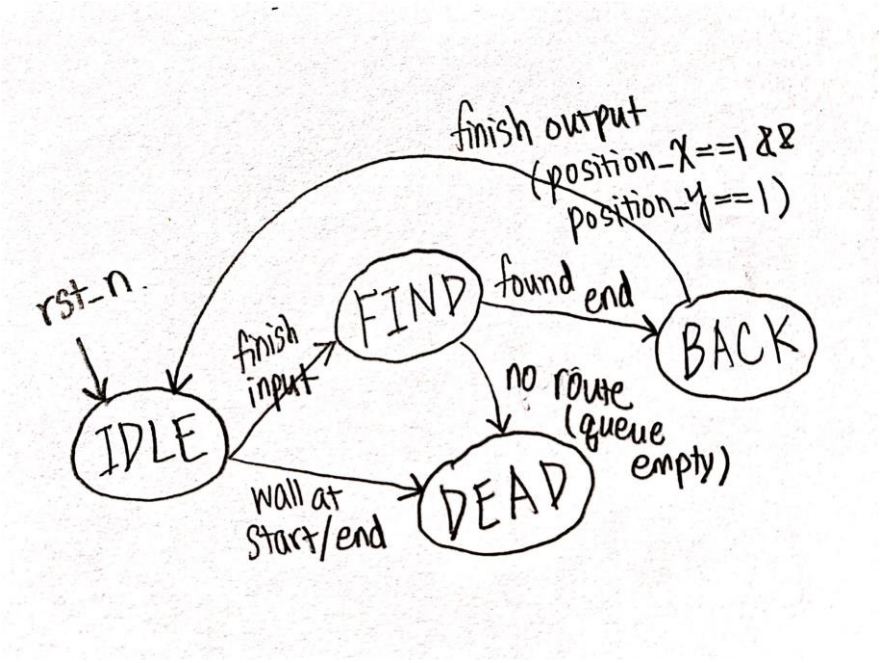
Block Diagram



FSM Diagram



我們這次的走迷宮是使用 BFS 演算法，我們在此結報中就不再針對 BFS 原理細做說明，而較著重針對我們遇到的個案和優化過程去做探討。

遇到困難和如何解決:

1. 路徑判斷

   我們在找路徑的時候依照「右上左下」(原因另作說明) 的順序在 combinational 去做一一的判斷，在 code 的方面就用四個 if-else，但若沒有多加判斷，而只看是否有牆壁擋住，會使每次的判斷都卡在判斷優先的位置上，而不會在下一個 clk 往後判斷，所以我們就另創一個變數(direction)存取現在判斷的位置(上、下、左或右)，然後在 if 的判斷裡面檢查 1)時否為牆壁 2)是否為當下在走的方向，就能順利的完成當下那一格鄰近四個位置的 BFS，但事後我們也有找到更好的優化方式，將在「優化過程」的「路徑判斷」進一步說明。

2. 往回過程

   當我們走到終點時，原先的寫法是按照走過的路以上左下右優先順序往回走，但其實我們後來發現這種走法要得到最短路徑只會在某些例子成立，而並不會保證走出最佳解。所以我們又另建一個 13*13 的矩陣去記下走到迷宮中那一格對應位置是從甚麼方向來的，而從終點往回的時候就依照每一格內存的「來源方向」遞迴式的往回走，直到走到起點，就完成輸出啦!

3. 上左下右/左上右下

   BFS 也是有優先順序的，因為我們有額外存方向來方便走回去，如果想要走回來時依照特定方向優先順序，就要好好想想該怎麼設計，一開始設計時就是照著助教說的上左下右的順序把位置存進 queue 裡，但是自己驗證時卻發現走回去的方向優先順序並非上左下右，而是左上右下，於是經過了一番推導，證明了如果 BFS 優先順序是左上右下的話，就會依照上左下右的規則走回去，這樣回程非常簡單，只要照來的分向往回走就行了，不用再做一次 BFS 判斷。

優化過程:

1. 路徑判斷

   先前提到的變數(direction)是為了避免重複判斷，但其實要避免重複判斷另有方法而且能節省不少面積，後來直接將判斷完的格子在存取迷宮的 reg 中改標為牆壁，就不會再次走到，而 output 往回走的時候也不會因此而受影響因為往回走是直接看當下要走的方向，不再判斷那一個為牆壁與否，因為這件事是在當初尋找路徑的時候就做好了。

心得

有修過資料結構真的有差，寫 C++的 pattern 時又把資料結構的東西複習了一次，在演算法的理解上沒有甚麼太大的阻礙，能夠快速掌握要運用 queue 的關鍵。這樣讓我們得知，其實寫程式這件事是牽動的，就算是不同的語言往往要運用一樣的邏輯或是概念，難怪天碩從來沒有寫過 python 也能幫社團的學妹 debug！把寫程式的邏輯訓練好就能一魚多吃，學習不一樣的 syntax 和 language 就能快速得心應手，所以，俗話說的好：「Verilog 寫地好，coding 沒煩惱。」跟我們的結報一樣，結報寫的好，助教給分沒煩惱，耶 XD~