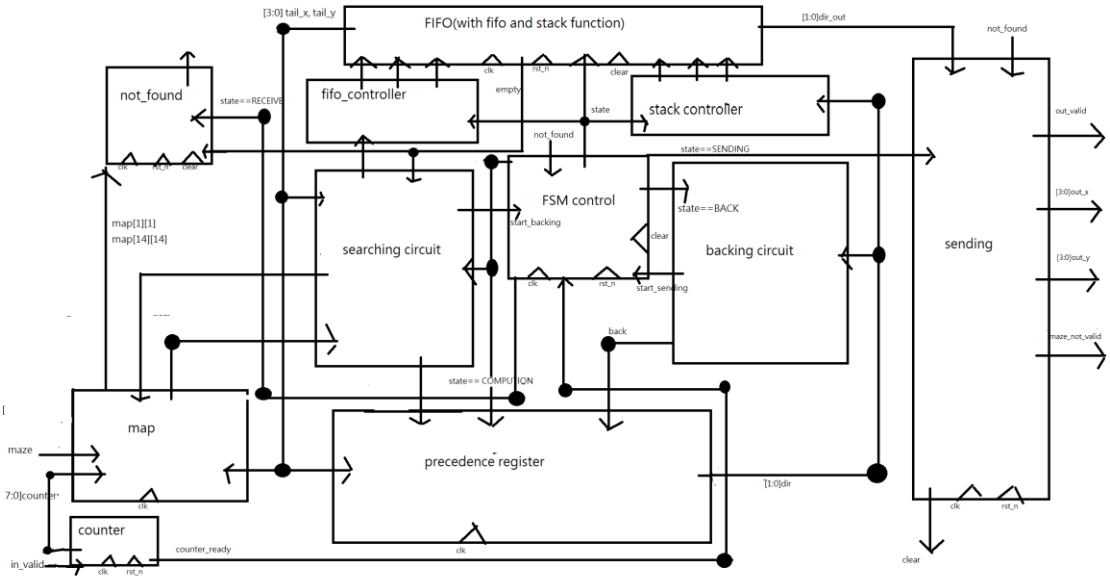
組別:27

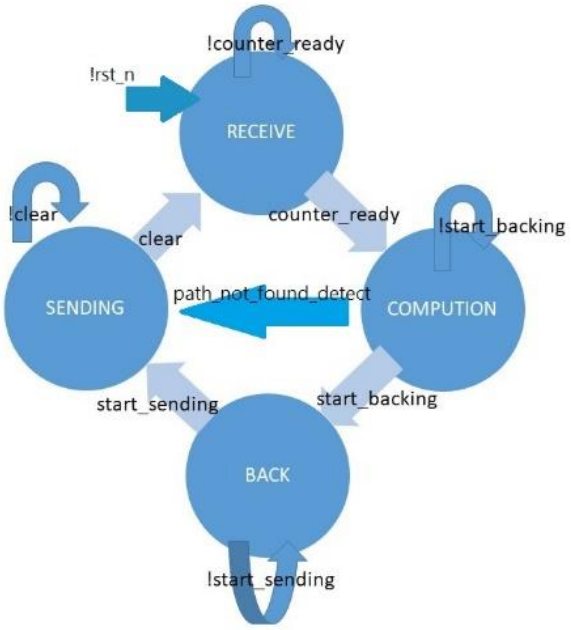
架構與原理

演算法是先從尾巴利用 bfs 與上左下右優先順序找路徑，再反推，最後送出。
下圖是架構圖:



FSM control:

將電路分成四個階段，接收、bfs、反推路徑、送出資料，並利用 fsm 控制。每個階段結束前一個 cycle 由一個訊號觸發，使 fsm 往下一個 state 移動。
Compution state 時，若 not_found 且 !in_valid(避免在 in_valid 時開始送資料)，下一個 cycle 往 sending 移動。



receiving:

用一個 8 bit counter，當 in_valid 時 counter+1。當 state==RECEIVE 時，通過 counter 來對 map 給值，每次 rst_n 和 clear 時使 map[1][1]和 map[14][14]兩個 DFF 值=0，此外 map 外圍值固定成 1。

Not_found:

由於每次 rst_n 或 clear 時，map[1][1]=0 且 map[14][14]=0，因此當 state==RECEIVE 時，偵測到 map[1][1]==1 or map[14][14]==1 時讓 not_found=1。此外，由於每次 rst_n 或 clear 時 fifo 大小均不=0(原因在 fifo)，只有當 searching 找不到路後 fifo 大小才會=0，因此 fifo empty 時讓 not_found=1。

searching:

當 state== COMPUTION 時，每個 clk 正緣觸發，map 接收 fifo tail 座標，並將 tail 四周位置資料回傳給 searching circuit，同時 searching circuit 將四周有更新的位置通過 fifo controller 加入 fifo，直到(1,1)加入 fifo(即 tail=(1,2), (2,1))，此時送出訊號給 fsm。

此外，map, precedence 接收 tail 位置，並由 searching circuit 控制周圍是否更新。

Backing:

當 state==BACK 時，每個 clk 正緣觸發，back 從(1,1)開始向 precedence 讀相對位置，back 利用相對位置更新 back 位置，同時將相對位置通過 stack controller 加入 stack。當 back==(14,14)時，送出訊號給 fsm。

FIFO:

利用 fsm 控制，可利用分時(共用記憶體)方法同時有 fifo 和 stack，當 state==COMPUTION 時當 fifo，當 state==BACK 或 state==SENDING 時當 stack。

當 fifo 時，會接收四組有優先順序的絕對位置，以及要加入 fifo 的數量，每次正緣觸發時，依據要加的數量，從優先順序大的加到優先順序小的

當 stack 時，接收是否讀以及是否寫的資料(每當讀一次資料刪除一次)，以及一組相對位置，並回傳最後加入的相對位置。

為了減少 cycle，當 clear 或 rst_n 時，將(14,14)加入 fifo，因此 fifo 大小=1，此外歸零 stack。

實現方法

針對 fifo，用 head_pointer 和 tail_pointer，當 empty 時 head_pointer==tail_pointer，增加資料時 head_pointer 加上加入的資料量，刪除一個資料時 tail_pointer 加一，因此從 tail_pointer 到 head_pointer-1 都是有效的資料。輸出資料為 tail_pointer 位置資料。

針對 stack，用一個 pointer，增加資料時 pointer+1，減少資料時 pointer-1，因此從 0 到 pointer-1 都是有效的資料。輸出資料為 pointer-1 位置資料。

記憶體控制電路分成兩個部分，並且由 fsm 控制電路是否有效。

FIFO_controller:

當 COMPUTION 時，通過接收 searching circuit 四個方位的位置，以及四個方位各別要不要加入訊號，和 state 訊號，並通過組合邏輯方式，產生要加入 fifo 的數量，並將要加入的資料依優先順序進行排序，並傳入 fifo。

Stack_controller:

通過 state 控制是否讀或寫，只有當 state==BACK 時要寫，只有 state==SENDING 時要讀。同時將資料直接送入 fifo。

Sending:

當 state==SENDING 時，每次正緣觸發，從(14,14)開始，利用 stack 的資料決定下個位置。用 not_found 判斷是否找到路徑。結束前一個 cycle 送出 clear 訊號，clear 後歸零必要的 DFF。

out_valid == state==SENDING。

clear==maze_not_valid || (out==(1,1))。

maze_not_valid== out_valid && not_found。

減少面積方法

1. precedence 只記錄相對位置
2. map 和 distance 共用，只要=1 就不能走
3. map 和 precedence 周圍 DFF 固定成常數
4. precedence[14][14]固定成常數
5. stack, fifo 共用記憶體
6. 大部分 DFF 不 rst_n
7. 判斷 state 是哪個共用電路
8. COMPUTION 時找 tail 旁四個位置電路共用加法器
9. BACK 時找下一個 back 位置電路共用加法器
10. SENDING 時找下一個 out 位置電路共用加法器

加速方法

1. rst_n 或 clear 之後，就直接加(14,14)到 fifo 內
2. 不等 in_valid 結束就進入運算
3. 偵測到 not_found 且!in_valid 就直接 sending

心得報告

我覺得這次作業比較麻煩的地方是，不能從頭找到尾，因為經過我多次嘗試，任何的優先順序，都有可能跟題目的規定找到的不同。此外，這次 project 對於 fifo 和 stack 大小決定很重要，太大面積會太大，太小會 fail，經過計算與測試，我選定 fifo size=8*32，stack size=2*128，兩者剛好一樣，且應該是最佳解(當大小=2^x)。