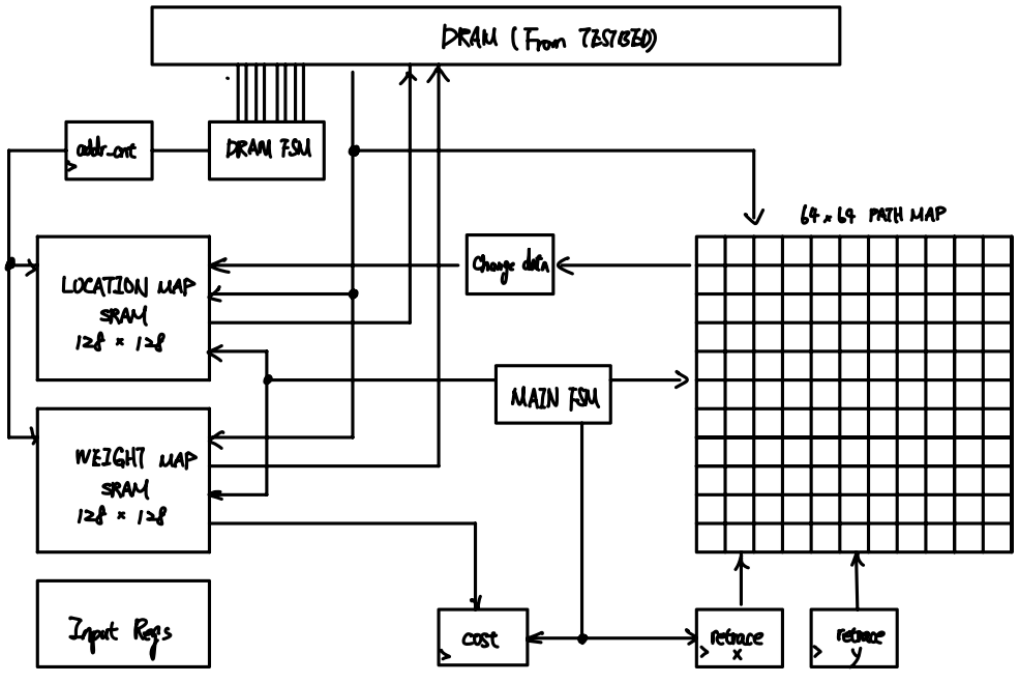
1. 電路架構



(1) LOCATION_MAP：用來儲存目前繞線結果的 SRAM

(2) WEIGHT_MAP：儲存目前 MAP 所對應之 WEIGHT 的 SRAM

(3) Addr_cnt：當下 SRAM 的位址控制器，以讀取對應的輸入

(4) Change data：當 PATH_MAP 要把結果記錄在 LOCATION_MAP 時，因為 SRAM 是 128-bit/word，而只要把其中的 4bit 換成現在的 netid，因此使用 Change data 這個單元來達成只更換 4bit 的功能。

(5) PATH_MAP：負責計算繞線過程的二維計算器

(6) MAIN_FSM：控制整體電路的 FSM

(7) DRAM_FSM：與 DRAM 溝通的 FSM，判斷何時需要寫入何時讀出

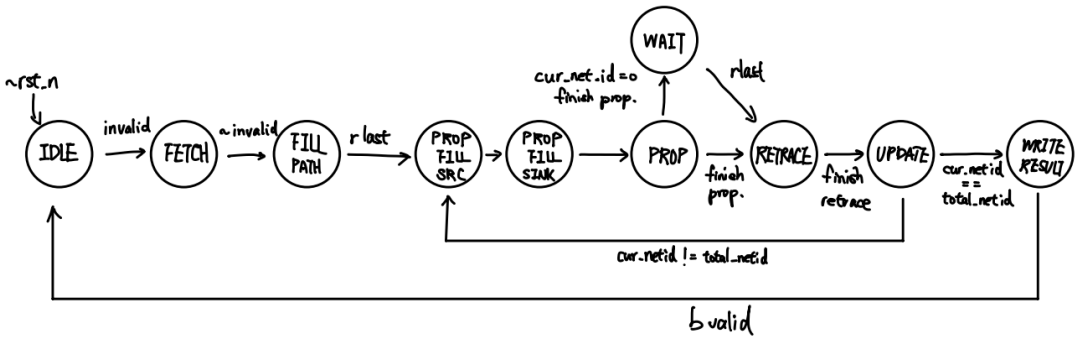
(8) Input Regs：存取所有輸入的資料

(9) Cost：計算最後 cost 的暫存器

(10) Retrace_x, Retrace_y：在 RETRACE 階段，需要追蹤當下 retrace 的座標點，分別為 PATH_MAP 上的 x 與 y 座標。

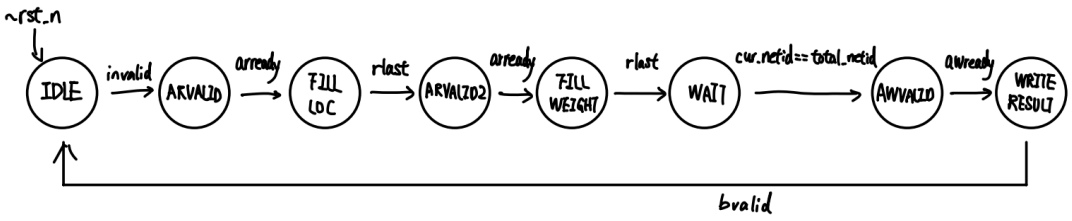
2. FSM

(1) Main FSM：



1. IDLE：初始狀態
2. FETCH：在 invalid 為 1 時，進行抓取資料的過程
3. FILL_PATH：
   DRAM 在送資料進來時，在這個狀態下會把 MAP 的結果存入 LOCATION_MAP 和 PATH_MAP 中。
4. PROP_FILL_SRC：
   要開始尋找路徑時，先將 source 變成起始點。
5. PROP_FILL_SINK：
   終點也要事先挖成一個空格，後續將判斷 sink 的位置上是否已經被填滿。
6. PROP：
   從 source 開始以 2、2、3、3 的方式向外擴散，直到碰到 sink 為止，此時 finish_propagate 會拉起。
7. WAIT：
   在第一次做完 PROP 時，時間上會對應到 WEIGHT_MAP 還在讀取，因此要先等到做完 WEIGHT 讀取後才可以繼續後續動作。
8. RETRACE：
   從 sink 開始，沿著計數器當下的紀錄開始往回推，沿路過程中都要把 PATH_MAP 上的值改成 1(Blocked)。
9. UPDATE：
   RETRACE 結束後，會觀察現在使否已經將所有 net 都繞線完成，如果沒有的話，要回到 PROP_FILL_SRC 來做下個 net 的繞線，而如果已經完成，就可以將答案輸出出去。
10. WRITE_RESULT：
    將最後繞線的結果從 SRAM 讀出後寫到 DRAM 上。

(2) DRAM_FSM：



1. IDLE：初始狀態
2. ARVALID：
   在準備好 raddr 時等待 DRAM 送出 arready。
3. FILL_LOC：
   由於已經設定好 burst 是可以連續送 128 筆 128-bit 資料，過程中可以將 raddr 向上計數，直到收到 rlast 後就跳到下一個狀態
4. ARVALID2：
   再次準備 raddr，等待 DRAM 送出 arready
5. FILL_WEIGHT：
   步驟與 FILL_LOC 相同，只是這個狀態讀取 weight 的值並寫入到 SRAM。
6. WAIT：
   等待 Main FSM 將繞線的過程做完，也就是當 cur_netid 和 total_netid 相同時，表示已經繞完所有的線，因此跳到下個狀態。
7. AWVALID：
   準備好 waddr 並等待 awready 的訊號
8. WRITE_RESULT：
   與 Main FSM 的 WRITE_RESULT 相同，在這個狀態下會把 SRAM 裡儲存的繞線結果抓出來後寫入 DRAM。

3. 優化方法

(1) PROP 終止的判斷條件：

在原本的設計中，會判對 sink 的**上下左右**是否被改成 2 或 3 了，而不是 sink 的位置是否改成 2 或 3。這麼做的目的是因為防止 PROP 多向外擴散一格時，會影響到 RETRACE 上的判斷準確度，進而提早判斷結束的條件。但經過時測候，只要能夠校正好 PROG 銜接到 RETRACE 的 counter，向外多擴散一格不會造成行為錯誤。

而由於前一個要取得 4 個 PATH_MAP 的值，而要獲取 PATH_MAP 的 MUX 本身也非常龐大，因此光是從獲取 4 個 PATH_MAP 的值變成 1 個，面積就有明顯的下滑。

(2) RETRACE 和計算 Cost 在同一狀態運算：

原本的設計中，會先將 RETRACE 的過程紀錄好，並獨立出一個狀態來將繞線的 cost 計算出來。但是獨立出來時，沒辦法知道當下 retrace 的座標，因此必須整張 PATH_MAP 都掃過，才能算出真整的 cost。

改良的方法就是在 RETRACE 時，先讀取當下 retrace 座標點所對應的 cost，並在下一個 cycle 將 cost 加上。這麼做雖然在 RETRACE 需要 2 個 cycle 才能做完，但是整體來看 latency 是可以明顯下降的，因為可以不用讀取整張 map。

(3) 更新 PATH_MAP 每格所需要的 bit 數：

先前已經有提及 PATH_MAP 在擴散出去的時候，是將內部填 2 或 3。而計數的方式就是 2、2、3、3。由於 PATH_MAP 本身就已經足夠龐大，如果使用 3 個 bit 來表示狀態，面積就會超出這次 spec 的要求。因此使用了 2-bit 來完成這次的 Midterm。其中 0 -> EMTPY、1 -> BLOCKED、2 -> CNT0、3 -> CNT1。

(4) 關於這次 Midterm 的教訓

在重要的輸出訊號如 busy，一定要以 DFF 來儲存，因為在合成過後，如果是 combinational 電路勢必會有中間 transition 的過程，而這次 Midterm 不巧就遇到了，這次因為一個 busy 的 glitch 就導致 fail 了這次的 Midterm，將 busy 改成 DFF 後，就能通過 gate sim 了。