VLSI System Design (Graduate Level)

Fall 2021

HOMEWORK IV

REPORT

Must do self-checking before submission:

Compress all files described in the problem into one tar

All SystemVerilog files can be compiled under SoC Lab environment

All port declarations comply with I/O port specifications

Organize files according to File Hierarchy Requirement

No waveform files in deliverables

Student name: \_陳韋綸(50%)\_ \_王傑世(50%)\_

Student ID: \_P76091412\_ \_P76091690\_

* **Summary**

1. 完成sensnor\_ctrl\_wrapper的設計
2. 完成HW4的APR
3. 完成CSR module的設計
4. 完成Prog0、Prog1、Prog2、Prog3的RTL驗證
5. 完成Prog0、Prog1、Prog2、Prog3的Gate-Level驗證
6. 完成Prog0、Prog1、Prog2、Prog3的Post-layout驗證

* **Architecture**

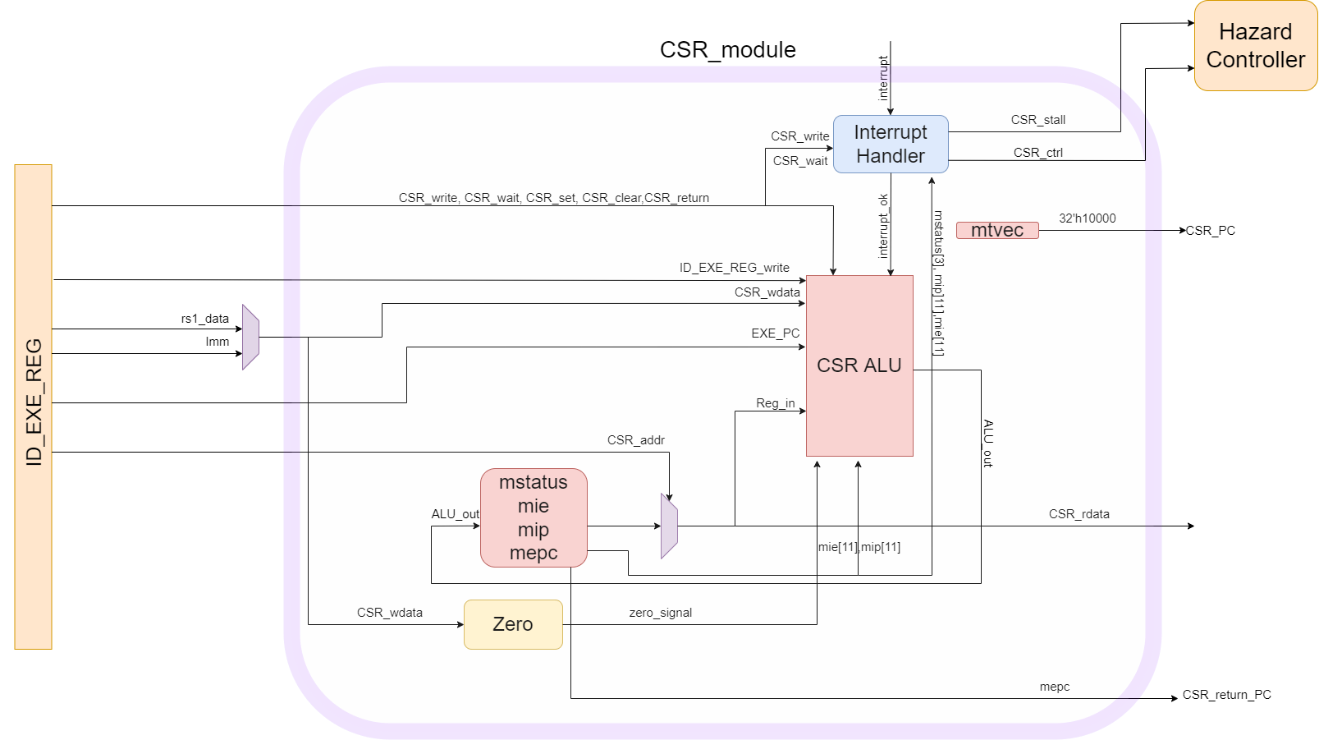
1. **Sensor\_controller\_wrapper**

* 在sensor\_ctrl\_wrapper的設計上，因為它與SRAM\_wrapper的行為類似，所以在實作上我是以SRAM\_wrapper去做修改的，唯一比較不同的地方在於發生write sensor的時候要根據地址分把wdata[0]的資料送到sctrl\_en和sctrl\_clear這2個訊號，讓sensor\_controller開始動作，如下圖所示

一張含有 文字 的圖片

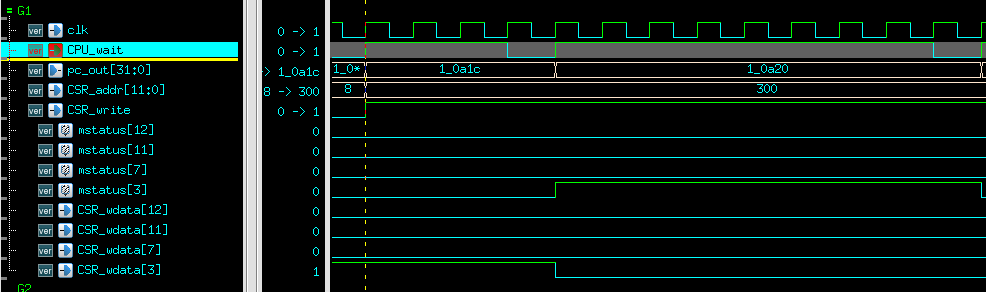
自動產生的描述

1. **CSR module**



* 在我的設計中，CSR module是放在EXE stage當中，因為考量到rs1\_data有可能會發生data hazard的情況，可能會需要forwarding，並且每次跳到ISR的位址或是return from interrupt的時候都要flush掉IF和ID 的pipeline register，這點和Branch/Jump指令一樣，綜合上述2點，我就將CSR instruction設計在EXE stage執行，如此一來可以很好的配合我目前的CPU設計
* 從上圖中可以看到，CSR module會根據由Instruction decoder解碼出來的CSR\_clear、CSR\_write、CSR\_set、CSR\_wait、CSR\_return、CSR\_addr這些訊號線設定CSR ALU要對CSR的哪個Register做寫入，並且會判斷這次wdata的來源是從rs1來或是immediate，之後會判斷CSR\_wdata是否為0，來設定zero signal
* 而當interrupt發生的時候會由interrupt Handler來處理，他會根據mstatus[3]、mie[11]以及mip[11]來判斷現在能不能處理external interrupt，並且根據情況設定好CSR\_ctrl和CSR\_stall的訊號
* CSR\_ctrl是當要處理interrupt的時候要告訴Hazard controller要把IF和ID的register都清掉，讓指令從ISR的位址重新開始執行，而CSR\_stall則是當執行到的指令是wfi時，要告訴Hazard controller，讓它將整個pipeline stall住，等待interrupt的到來
* **Waveform Verification**
* 因為HW4主要是新增了sensor\_wrapper以及CSR相關的指令，而又因為我在sensor\_wrapper的設計與SRAM\_wrapper相差不太，所以在這部分就沒有特別截圖sensor\_wrapper的波型
* 因此在這部分主要針對CSRRW、CSRRS、CSRRC、WFI、MRET和發生interrupt時要CPU會如何處理進行說明

1. **CSRRW**



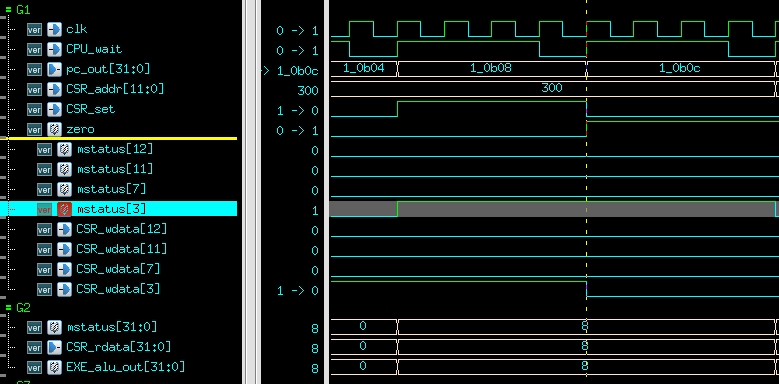
* 當pc\_out = 10a1c的時候代表EXE stage正在處理10a14的 「csrw mstatus,t0」指令，因此會把CSR\_write的訊號拉高，同時因為CSR\_addr = 12’h300(屬於mstatus register)，所以在下一次CPU\_wait=0時就會根據CSRRW的行為把CSR\_wdata[3](rs1\_data[3])、CSR\_wdata[7](rs1\_data[7])、CSR\_wdata[11:12](rs1\_data[11:12])的值assign給mstatus[3]、mstatus[7]、mstatus[11:12]

一張含有 文字, 時鐘, 時鐘收音機 的圖片

自動產生的描述

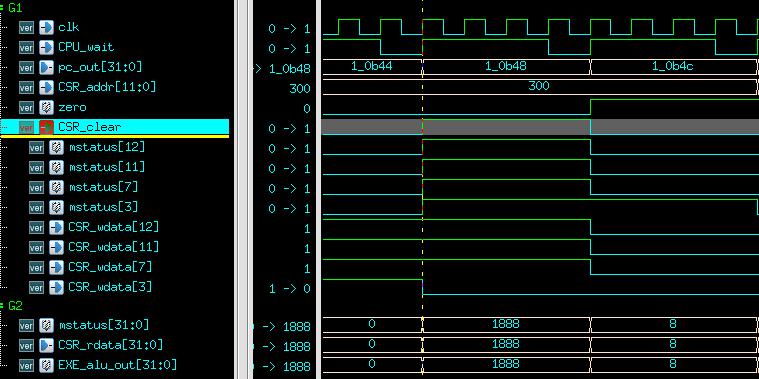
* 並且會把mstatus的值傳送到EX/MEM stage，最後寫回到rd裡面

1. **CSRRS**



* 當pc\_out = 10b08的時候代表EXE stage正在處理10b00的 「csrrs t0,mstatus,t1」指令，因此會把CSR\_set的訊號拉高，同時因為CSR\_addr = 12’h300(屬於mstatus register)且zero=0(判斷rs1\_data是否為0)，所以在下一次CPU\_wait=0時就會根據CSRRS的行為把CSR\_wdata[3](rs1\_data[3])、CSR\_wdata[7](rs1\_data[7])、CSR\_wdata[11:12](rs1\_data[11:12])的值分別和mstatus[3]、mstatus[7]、mstatus[11:12]進行OR運算後寫回mstatus，同時將mstatus的值傳到EX/MEM stage，最後送回rd裡面

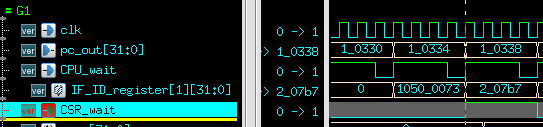
1. **CSRRC**



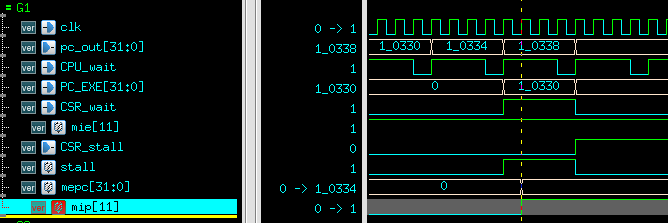
* 當pc\_out = 10b48的時候代表EXE stage正在處理10b40的 「csrrc t0,mstatus,t1」指令，因此會把CSR\_clear的訊號拉高，同時因為CSR\_addr = 12’h300(屬於mstatus register)且zero=0(判斷rs1\_data是否為0)，所以在下一次CPU\_wait=0時就會根據CSRRS的行為把~CSR\_wdata[3](rs1\_data[3])、~CSR\_wdata[7](rs1\_data[7])、~CSR\_wdata[11:12](rs1\_data[11:12])的值分別和mstatus[3]、mstatus[7]、mstatus[11:12]進行AND運算後寫回mstatus，同時將mstatus的值傳到EX/MEM stage，最後送回rd裡面

1. **WFI**

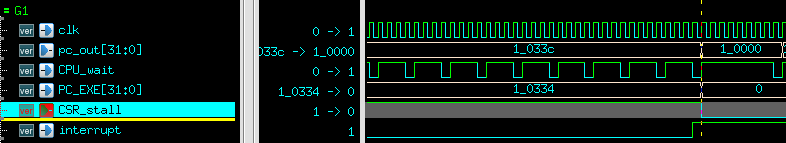
* 當pc\_out=10338的時候代表EXE stage正在處理10330的「wfi」指令



* 從上圖中可以看到當CPU去10330的address拿到instruction後就會存在IF/ID register中，也就是32’h10500073，把這個指令送到Decoder之後就會知道要執行的是wfi指令，因此在下一次CPU\_wait=0的時候就會把CSR\_wait的訊號拉起來，告訴CSR的module要處理wfi

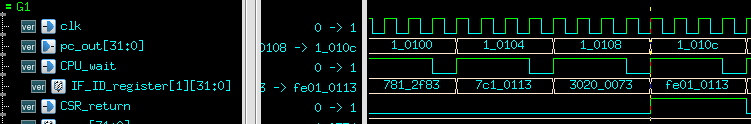


* 上圖中，當CSR\_wait拉起來之後會去判斷mie[‘MEIE]的值，如果mie[‘MEIE]為1，則會stall住整個pipeline，由於CSR\_wait & mie[11]的結果為1，因此在下一次CPU\_wait=0之後就會把整個pipeline都停住
* 同一時刻，當CSR module收到CSR\_wait signal後就會把PC\_EXE(EXE\_stage的program counter)+4寫入到mepc裡面，所以mepc = 10330 + 4 = 10334，而mip[`MEIP]也會根據mie[`MEIE]的值去做改變，如果mie[`MEIE]=1的話，mip[`MEIP]就會變成1，否則不變，而在上圖中因為mie[11]=1，所以在下一個cycle後mip[11]就被設成1了

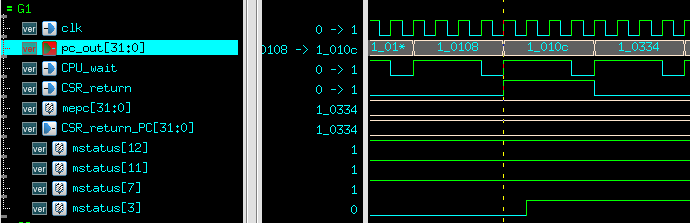


* 接著等interrupt來了之後CSR\_stall就會被拉下來，讓pc跳到處理interrupt的ISR位址

1. **MRET**

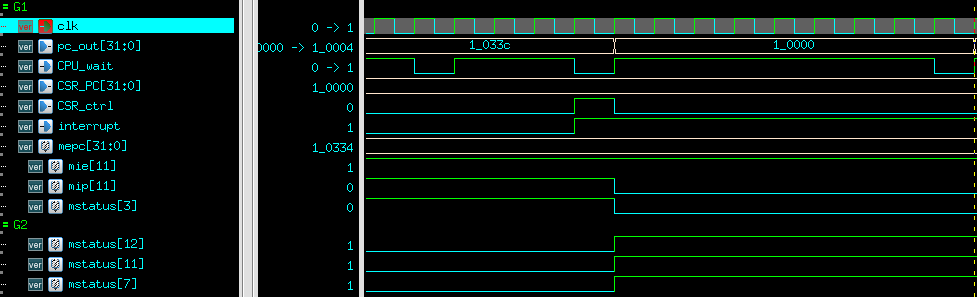


* 從上圖可知，當CPU到address 10104的位址拿data後，會得到32’h30200073的指令，將這個指令送到Decoder解碼後，CPU就會知道要做的是「mret」，因此在下一個CPU\_wait=0的時候就會把CSR\_return的signal送到CSR module裡面

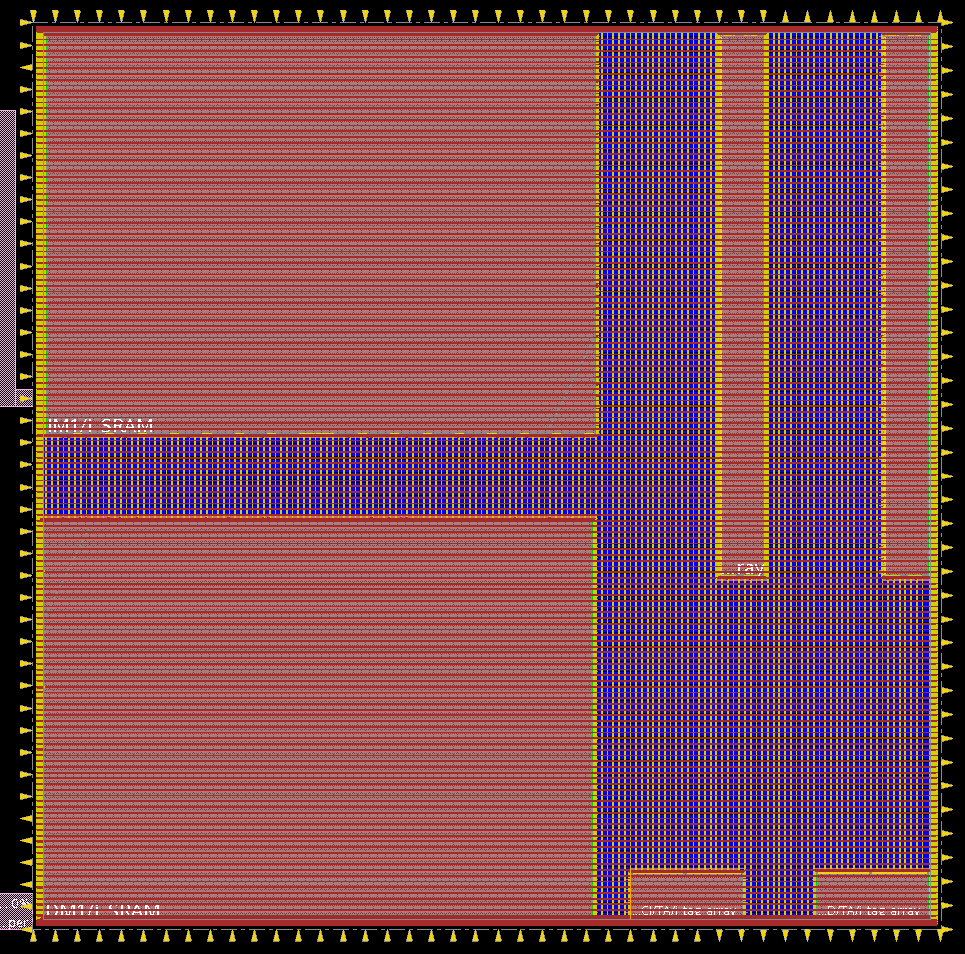


* 從上圖來看，當CSR module收到CSR\_return的信號後就會把mepc裡的address輸出給CSR\_return\_PC，之後再送到progrom counter裡，所以在下一次CPU\_wait=0的時候，pc\_out就會因為收到來自CSR\_return\_PC的address而跳到10334這個位址，同時mstatus[7]會被設成1、mstatus[11:12]會被設成2’b11，而mstatus[3]會被設成mstatus[7]，因為在上圖中mstatus[7]、mstatus[11:12]都分別是1’b1和2’b11的狀態了，所以沒有變化，而mstatus[3]就會因為mstatus[7]=1’b1而被設成1

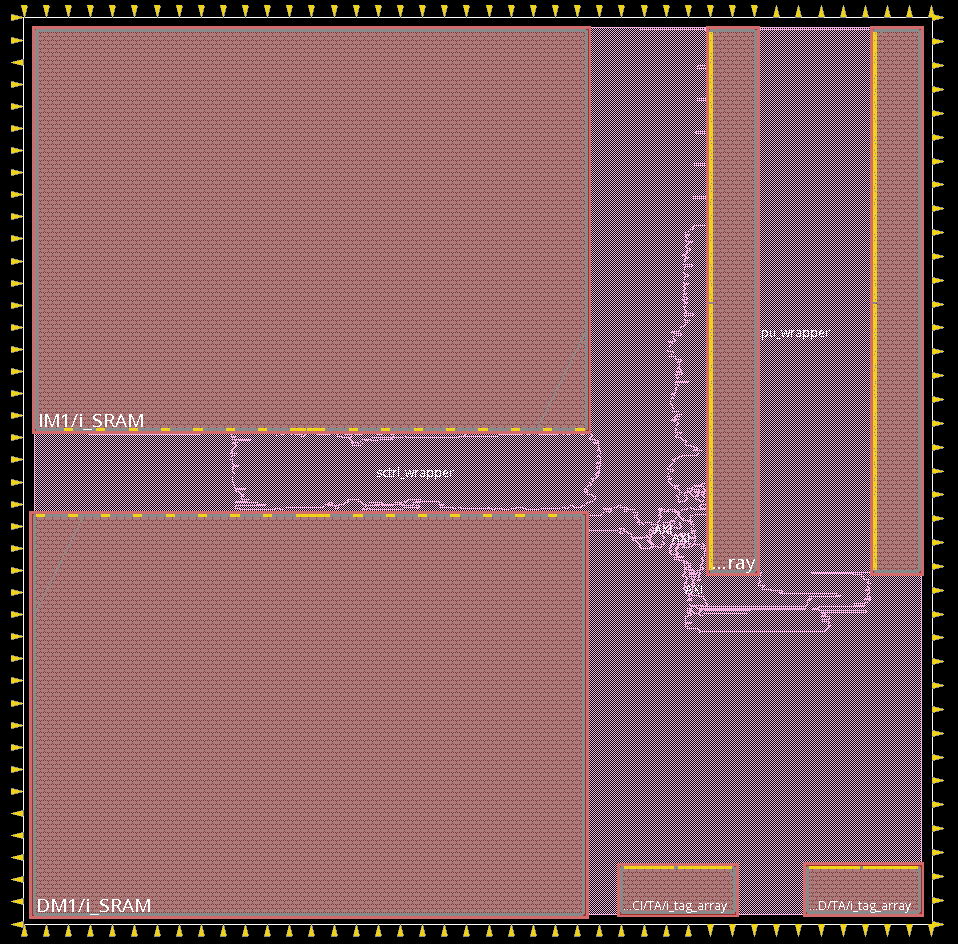
1. **Interrupt**



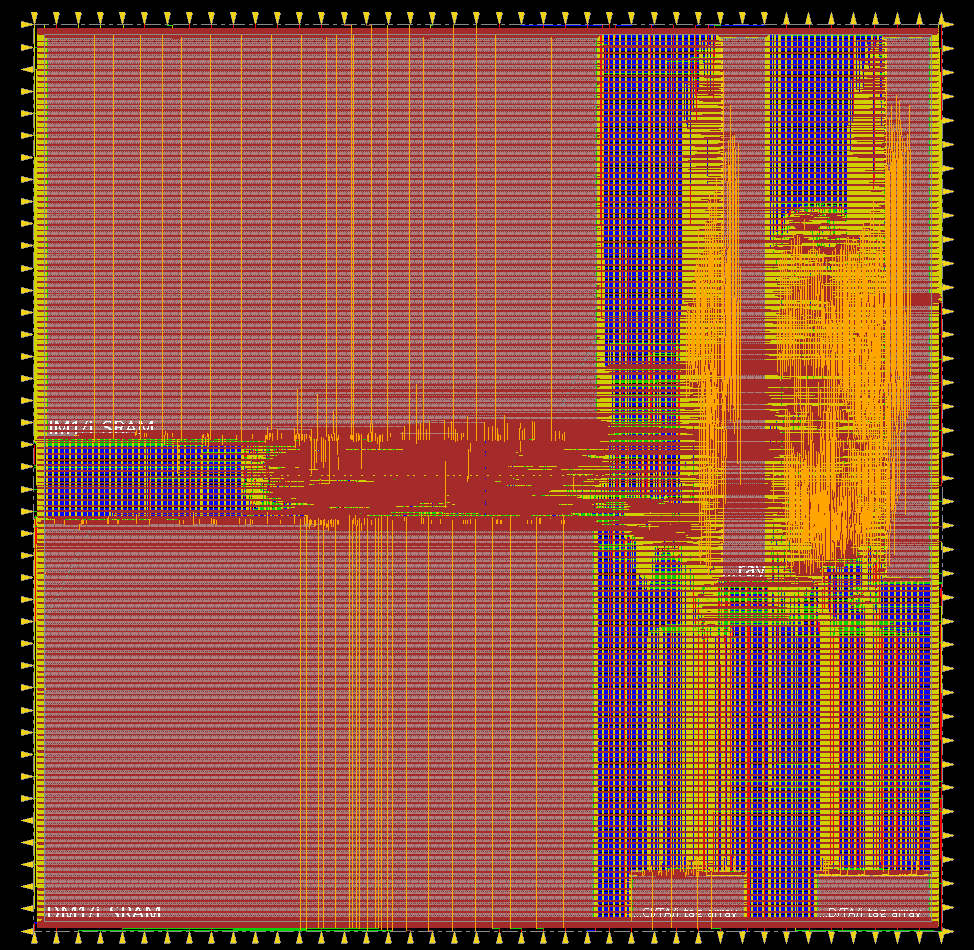
* 當sensor\_controller因為local memory滿了後發出interrupt訊號時，CSR module會根據mstatus[3]、mip[11]和mie[11]的訊號來判斷CPU現在可不可以處理來自external的interrupt，因此上圖中的CSR\_ctrl就是把mstatus[3]、mip[11]、mie[11]以及interrupt這4個signal做AND運算，如果結果為1，代表可以處理external interrupt，接著CSR module就會把mtvec=10000的位址assign給CSR\_PC，並且在下一次CPU\_wait=0的時候把CSR\_PC assign給program counter，讓CPU可以跳到ISR的位址處理interrupt
* 發生interrupt的同一時間mstatus也會根據mip[11]的值做改變，從上圖來看，發生interrupt時mip[11]=1’b1，所以mstatus[7]會變成mstatus[3]=1’b1，mstatus[3]會變成1’b0，mstatus[11:12]會變成2’b11，最後mip[11]會變成1’b0，如此一來就算完成interrupt發生時CSR需要進行的處理
* **Automatic Placement and Route**
* **Floorplan View**



* **Amoeba View**



* **Physical View**



* **DRC verification**

一張含有 文字 的圖片

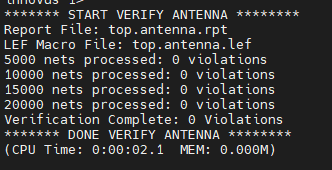
自動產生的描述

* **Connectivity Verification**

一張含有 文字 的圖片

自動產生的描述

* **Antenna Verification**



* **Booting program explanation**

一張含有 文字 的圖片

自動產生的描述

* 在Booting program中，主要的行為就是把存放在DRAM當中的資料搬到IM以及DM裡面，因此我的c code寫法也很簡單，
* \_imem\_start: 首先會計算DRAM當中instruction start的起始位置和instruction end的結束位置，把2者相減就可以得知DRAM當中存放幾條指令，接著再用一個for迴圈從\_dram\_i\_start的起始位置開始依序把instruction搬到\_imem\_start裡面對應的index位址。
* \_\_data\_start: 同上，將\_data\_start和\_data\_end相減就可以得知Main data當中要存放多少的data數量，接著再利用一個for迴圈從\_data\_paddr\_start的開始位置依序把\_\_data\_paddr\_start裡的資料寫進到\_\_data\_start中相對應的index位址。
* \_\_sdata\_start: 同理，先利用\_sdata\_start和\_sdata\_end這2者的位置相減，就可以得知DM的Main data裡面應該要存放多少資料量，接著再利用一個for迴圈依序將\_sdata\_paddr\_start裡面的資料寫入到\_\_sdata\_start對應的index位址中。
* **Prog2 explanation**

一張含有 文字 的圖片

自動產生的描述

* 在prog2的乘法部分，我一樣是利用邏輯左移和累加的方式完成，只不過會再額外判斷Input是否有負數，如果有負數則要將輸出結果加上負號，而在main方面則會利用3個for迴圈去完成矩陣乘法。
* Adder\_A: Adder\_A主要是要從矩陣A取得要乘的數的位置在哪，因此當p為0的時候代表現在要乘的是矩陣A的第0列，r則是在算它是這一列的第幾個，前面之所以還要一個mul(p, array\_size\_k)是因為，當現在要讀的是第1列以上的數值就會需要加上一個offset，這個offset就是前面行數讀取過的全部data的數量，如此一來就可以很容易就知道當前row的第一個數值是第幾個data了。
* Adder\_B: Adder\_B在做的事情跟Adder\_A相同，只不過因為存資料的順序是先存Adder\_A再存Adder\_B，因此Adder\_B第一個data的index就不會是0，而是矩陣A的大小，算出矩陣B的起始位置之後再把起始位置和現在在讀第幾個data以及前面row讀出的所有data數量總和相加就可以知道現在矩陣B要乘的數是存在array中的哪一個位置了
* Result\_temp: Result\_temp則是做一個相乘累加的行為，再執行完Adder\_A和Adder\_B之後就得到矩陣A和矩陣B所要相乘的數的位置，因此就把array\_addr中對應位子的數值做相乘並且累加起來，值到r=array\_size\_k的時候就代表說這個列x行已經完成了，所以就把result\_temp寫回到&\_test\_start中，然後再把result\_temp歸零。
* **Number of Lines of RTL code and Superlint result**
  + **Number of Lines of RTL code**

**一張含有 文字 的圖片

自動產生的描述**

* + **Superlint result**

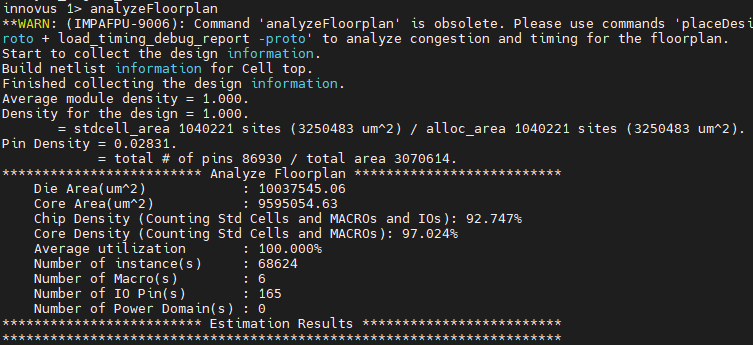
**一張含有 文字 的圖片

自動產生的描述**

* **Performance and Area**
* **Performance**

|  |  |  |
| --- | --- | --- |
| **Rtl0** | **Rtl1** | **Rtl2** |
|  |  |  |
| **Syn0** | **Syn1** | **Syn2** |
|  |  |  |
| **Pr0** | **Pr1** | **Pr2** |
|  |  |  |
| 908784(ns) | 80036464(ns) | 1554088(ns) |

* **Area**

****

* Lesson Learned

1. 學到了sensor\_wrapper的設計
2. 了解到CSR指令在RISC-V CPU是如何運作的
3. 學到了Innovus的使用方法
4. 了解到要如何對合成後的module做APR
5. 知道當發生Interrupt時CPU應該要如何處理