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: \_\_陳柏廷(50%)\_\_ \_\_顧芳宜(50%)\_\_

Student ID: \_\_P76091284\_\_ \_\_ P78101556\_\_

Summary

RTL part

|  |  |
| --- | --- |
|  | Complete CSR module(include CSR instruction and interrupt mechanism) |
|  | Complete sensor ctrl wrapper |
|  | Added “cacheable” wire and “UNCACHE” state in L1C\_data cache |
|  | Complete APR and simulate |

Simulation result

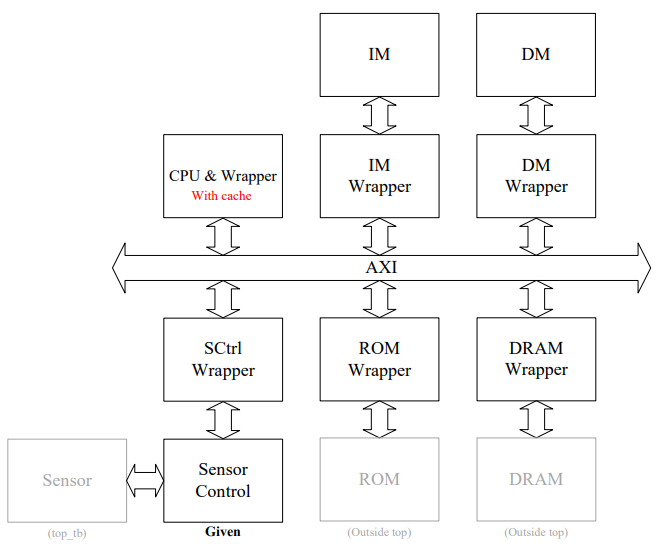
|  |  |  |  |
| --- | --- | --- | --- |
|  | rtl | syn | pr |
| Prog0 |  |  |  |
| Prog1 |  |  |  |
| Prog2 |  |  |  |

Lesson learned

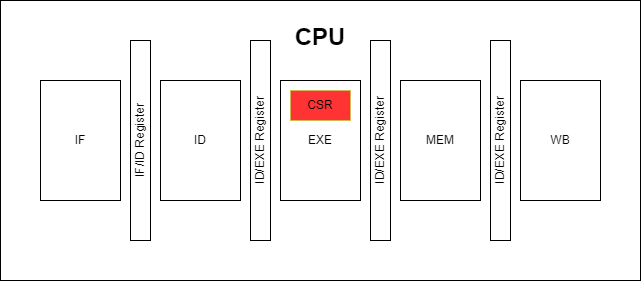
* Register務必要做reset，否則使得合成後的模擬產生錯誤。
* 若top有input 或output建議在input進來之後或output出去之前加register，才不會讓你的導致在做APR之後發現critical path在input 或 output的wire上。
* 如教innovus助教所說，hard macro適當的擺放可以減少在verify geometry或connectivity較不會產生violation，在做APR的過程中嘗試過非常多次。

Architecture diagram

整體大架構引用HW IV中的圖

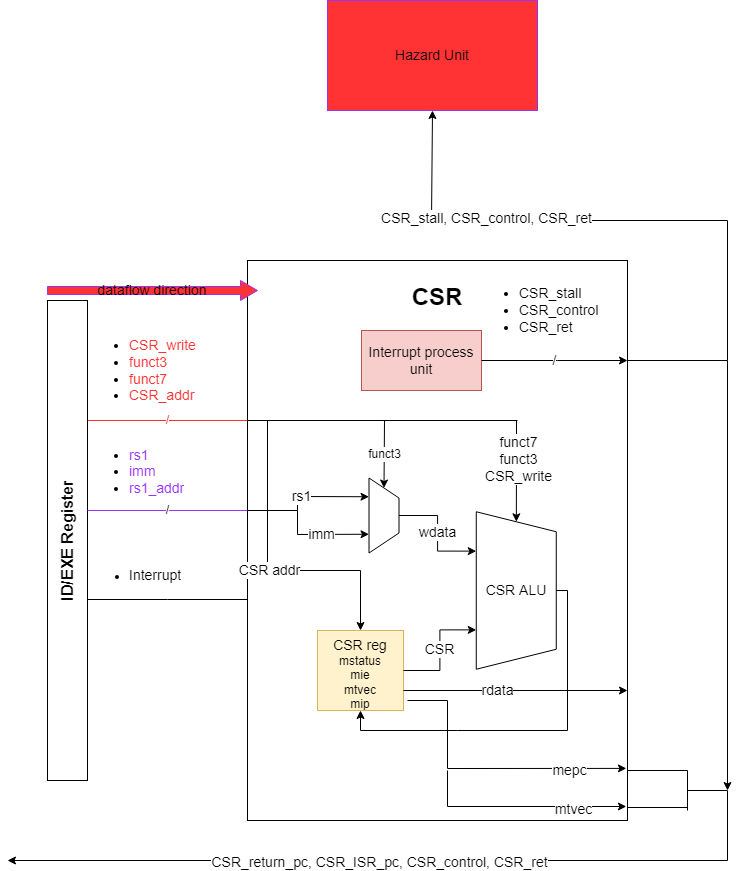


而其中此次重點部分為CSR module 新增於CPU當中，如下圖所示紅色所示。



從上圖可知，我們將CSR置於EXE stage當中，其中的考量包含了會對CSR中的CSR register與rs1做運算並且將CSR register的value存進rd，與ALU之運算有相似之處。而另一個考量之處則是由於interrupt或是return from trap發生時，會需要branch的動作判斷，而之前的設計也是於EXE stage做判斷，因此認為將CSR module置於此處是最佳的選擇。

而細部設計如下圖，



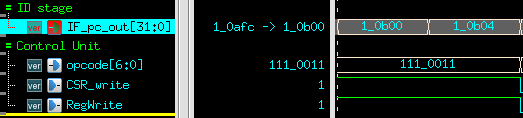
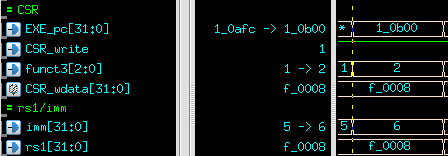
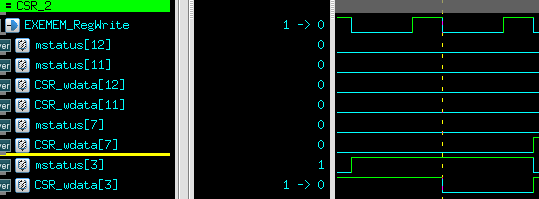
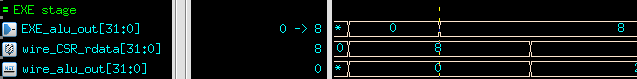
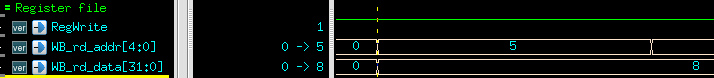
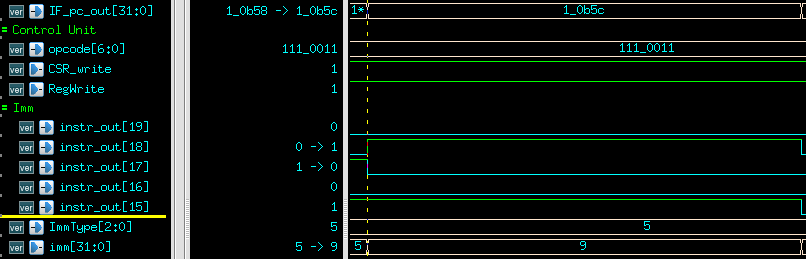
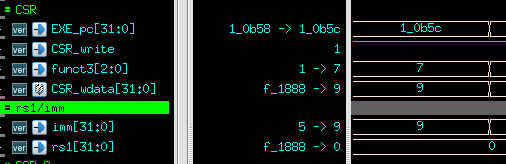
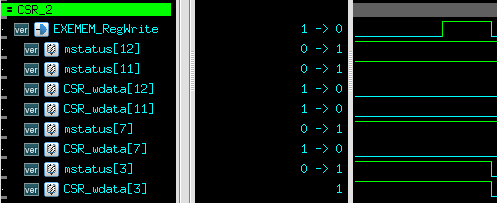
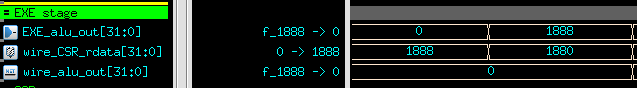
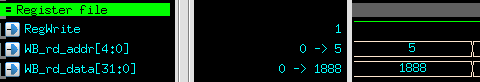
有上圖可知CSR module最主要的元件為CSR register，根據CSR的instruction將CSR register與imm / rs1做logic 運算，並將結果存回CSR register且輸出CSR register 運算前的值。  
而interrupt相關的運作未列在上圖，只以一個interrupt processing unit，其中包括了WFI, return from trap, interrupt發生時的相關輸出，可以看到CSR\_stall, CSR\_control, CSR\_ret這幾條wire表示。  
CSR\_stall 發生在當WFI指令來時，並確認mie中的MEIE bit，則會將CSR\_stall設為1，直到interrupt觸發時，才又在設為0。  
CSR\_control則在sensor直接送入的interrupt訊號觸發時，同時判斷”mstatus的MIE bit, mip的MEIP bit, mie的MEIE bit”是否為1，若條件成立，則會將CSR\_control設為1表示在下一個cycle，CPU將會跳到預設的ISR start的位址。

而CSR\_ret則是在trap結束後返回的mret指令所觸發，CPU會跳回進入trap之前所執行的指令位址。

Waveform & Explanation

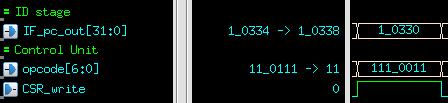
此部分以rtl0及rtl1之波型作說明

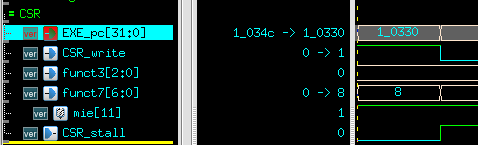
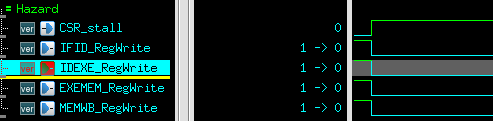
rtl0因作用類似，因此只取CSRRS及CSRRCI 指令

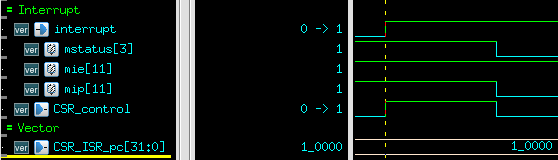
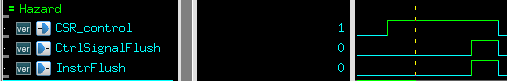
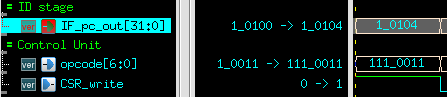
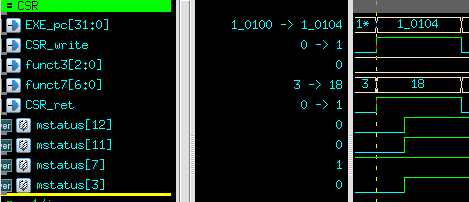
1. CSRRS < csrrs t0,mstatus,t1>  
   001100000000 00110 010 00101 1110011  
    CSR(mstatus) rs1(t1) csrrs rd(t0) opcode(CSR instr)
   1. ID stage
      1. 
         1. 上圖紅框處(1\_0b00)為csrrs t0, mstatus, t1指令在control unit解碼出相關訊號
         2. Opcode = 111\_0011表示CSR instruction  
            因此將CSR\_write(判斷時否為CSR instr), RegWrite(因需要將CSR[mstatus]之值寫回rd[t0]) 設為1，並且存入ID/EXE Reg，以便在EXE stage及WB stage作使用。
   2. EXE stage
      1. 
         1. 上圖為csrrs經過decode進入EXE stage之CSR module
         2. 首先CSR\_write為1表示目前為CSR instruction。
         3. CSR\_wdata為要與CSR reg運算之運算元，利用funct3[2]決定imm/rs1，此時funct3[2]為0，因此CSR\_wdata=rs1。
      2. 
         1. 上圖為將mstatus, CSR\_wdata(rs1)之[12:11], [7], [3]bit作 | 運算。
         2. 將結果在EXEMEM\_RegWrite為1時寫回mstatus，因為EXEMEM\_RegWrite若為0則為stall狀態，若提前寫入則使得值提前寫入而使得最後存入Register file的值錯誤。
      3. 
         1. 上圖說明當指令為CSR instruction時，則mux會選擇wire\_CSR\_rdata作為送入EXE/MEM Reg的wire，並且在WB stage 存入Register file。
   3. WB stage
      1. 
         1. 將mstatus之值存入rd register(t0)
2. CSRRCI  
   001100000000 01001 010 00101 1110011  
    CSR(mstatus) imm(0-ext) csrrci rd(t0) opcode(CSR instr)
   1. ID stage
      1. 
         1. 上圖為(1\_0b5c)為csrrci t0,mstatus,9指令在control unit解碼出相關訊號
         2. Opcode = 111\_0011表示CSR instruction  
            因此將CSR\_write(判斷時否為CSR instr), RegWrite(因需要將CSR[mstatus]之值寫回rd[t0]) 設為1，並且存入ID/EXE Reg，以便在EXE stage及WB stage作使用。
         3. 然因為為CSR之I type指令，因此Imm generator根據解碼出來的Imm\_type=C\_Imm將instr\_out[19:15]作zero-extend，結果以imm wire輸出存入ID/EXE register。
   2. EXE stage
      1. 
         1. 上圖為csrrci經過decode進入EXE stage之CSR module
         2. 首先CSR\_write為1表示目前為CSR instruction。
         3. CSR\_wdata為要與CSR reg運算之運算元，利用funct3[2]決定imm/rs1，此時funct3[2]為1，因此CSR\_wdata=imm。
      2. 
         1. 上圖為將mstatus, ~CSR\_wdata(imm)之[12:11], [7], [3]bit作 & 運算。
         2. 將結果在EXEMEM\_RegWrite為1時寫回mstatus，因為EXEMEM\_RegWrite若為0則為stall狀態，若提前寫入則使得值提前寫入而使得最後存入Register file的值錯誤。
      3. 
         1. 上圖說明當指令為CSR instruction時，則mux會選擇wire\_CSR\_rdata作為送入EXE/MEM Reg的wire，並且在WB stage 存入Register file。
   3. WB stage
      1. 
         1. 將mstatus之值存入rd register(t0)

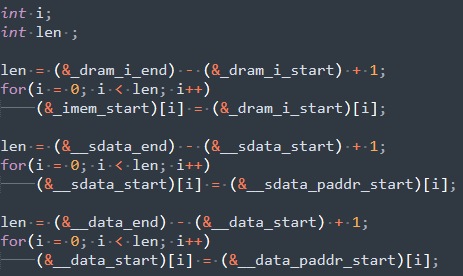
再來是rtl跟WFI, interrupt以及return from trap的部分

在rtl1 WFI指令來之前已經將mstatus[3]以及mie[11]設為1，即WFI指令來CPU即作CSR\_stall的動作。

1. WFI  
   0001000 00101 00000 000 00000 1110011  
    WFI
   1. Decode
      1. 
         1. 上圖為WFI 指令在Control Unit decode情形，看到將CSR\_write設為1，告知CSR module 此為CSR instruction。

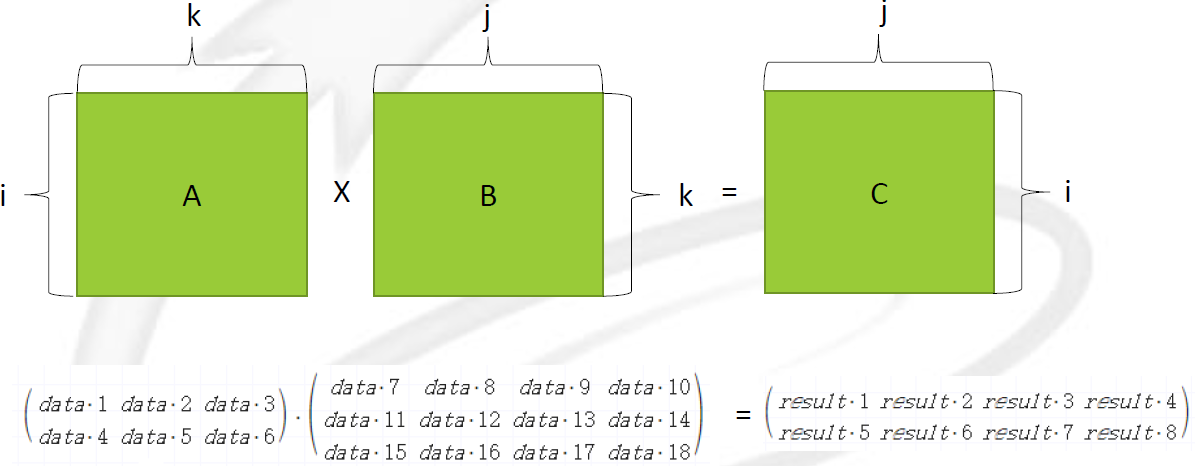
* 1. EXE stage
     1. 
        1. 此為WFI instruction至EXE stage
        2. CSR為0表示CSR instruction  
           funct3 = 3’b0 和 funct7 = 7’d8表示WFI
        3. 此是根據mie[11]=1’b1，因此將會在下一個cycle將CSR\_stall拉起。
  2. Hazard Unit
     1. 
        1. 因為WFI 將CSR\_stall設為1，因此CPU必須整個停下，因此將IF/ID, ID/EXE, EXE/MEM, MEM/WB 的RegWrite 訊號拉為0使得所以Register bar維持目前的值。
        2. 直到interrupt來之前都必須將CPU stall。

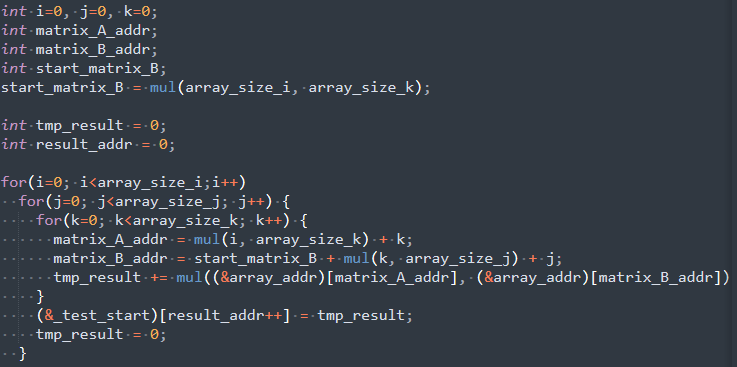
1. Interrupt
   1. CSR
      1. 
         1. Interrupt來之後判斷mstatus[3], mie[11], mip[11]為1，若條件成立則將CSR\_control拉為1。
   2. IF
      1. 
         1. CSR\_control拉為1之後，pc\_in則送入CSR module 的mtvec 訊號，開始執行對應的指令。
   3. Hazard
      1. 
         1. CSR control 拉為1之後，由於IF必須跳至相對應的address，因此需要清除已經在IF的指令，同時需要將ID/EXE的訊號線也作清除。
2. Return from trap  
   0011000 00010 00000 000 00000 1110011  
   return from trap
   1. ID
      1. 
         1. Opcode 判斷為CSR instruction，因此將CSR\_write設為1
   2. EXE
      1. 
         1. 根據funct3 = 3’b0, funct7=7’d18(2’b11000)得知為wret指令。
         2. 因此將CSR\_ret拉為1，並分別送至IF及Hazard unit
         3. 同時將mstatus的相關bit設定至interrupt之前的狀態。
   3. IF
      1. 
         1. CSR\_ret拉起後送至IF module，pc\_in設為interrupt之前的instruction address
   4. Hazard
      1. 
         1. 如interrupt時動作，作了jump的動作，因此需要將IF中的指令flush並將ID/EXE的register清除。

C code of booting process  


Booting的話，主要是將DRAM中的資料搬移至SRAM中。若是搬移到IM的話，實作的方式是算出DRAM記憶體的儲存空間的大小，並從記憶體第一筆資料的位置將資料依序搬移到對應的SRAM中。若是搬移到DM，則是算出DM記憶體空間的儲存大小，在依序將資料從DRAM第一筆資料位置搬到相對應的SRAM中。

C code of program2



此矩陣乘法運算需要注意資料擺放的位置。從A左上角開始依照row擺放，到B右下角。結果也是從C左上角開始輸出，到右下角。因此結果運算需要從A第一個row對上B第一個column開始，到A第一個row對上B第二個column，以此類推。  


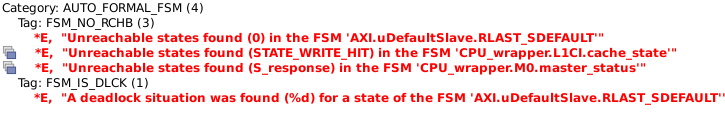
因此，main.c中，我們用了三個for loop，最外層是A的row，中間為B的column，最內層迴圈為一個row /column中的每個element。知道第幾行/列後，我們需要先找到相對應A和B 矩陣的相對應起始位置，才能開始做乘法運算。最後再將答案加總至結果後，即可算出C的一個element。

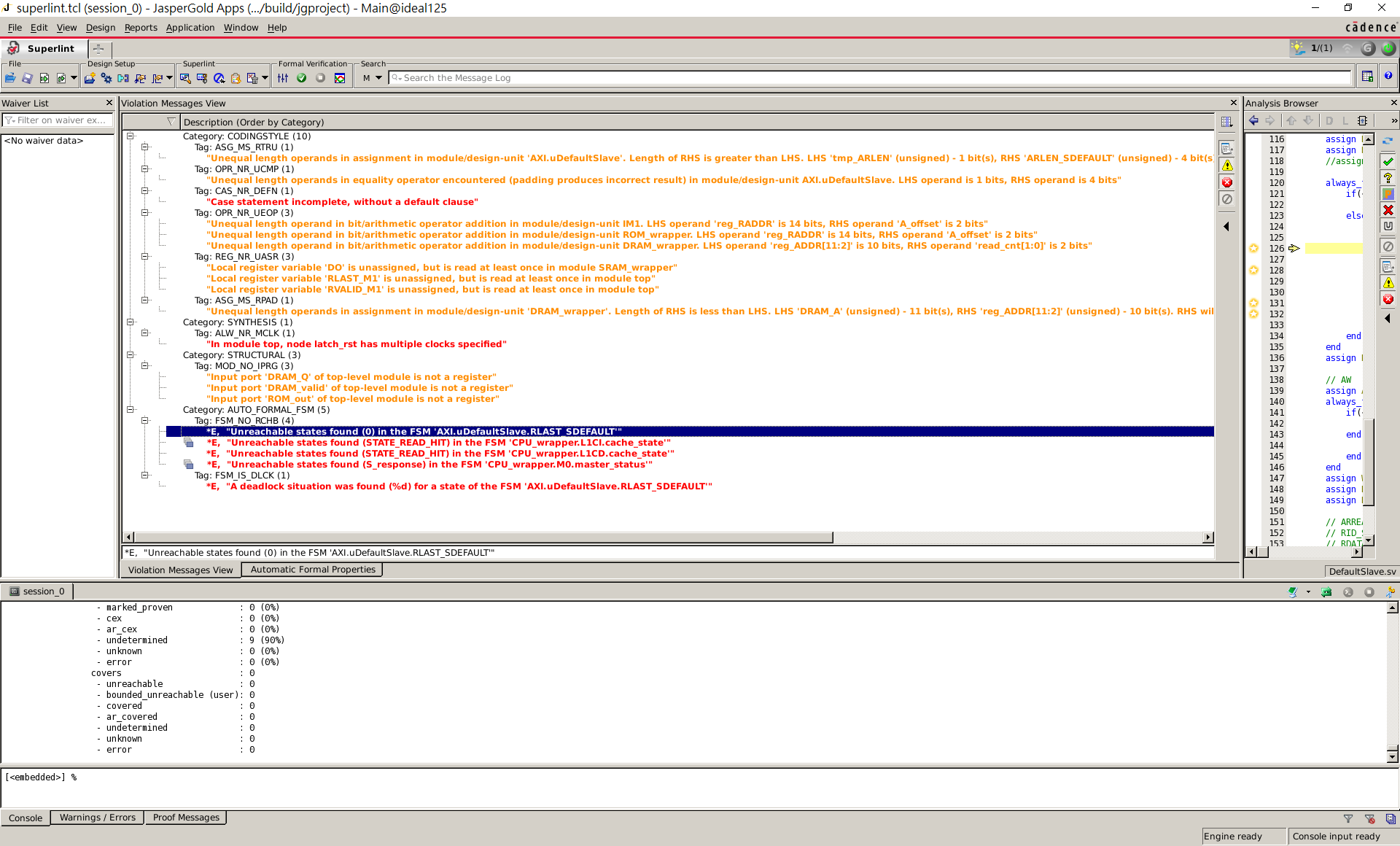
Interrupt mechanism

1. 在WFI及interrupt來之前，需先將mie[11](External interrupt enable)及mstatus[3](global interrupt enable)設為1。
2. WFI來時檢查mie[11]是否為1，條件符合的話則會將整個CPU stall直到external interrupt出現。  
   此時紀錄pc+4至mepc並設定mip[11]。
3. 當interrupt來時，檢查mip[11], mie[11]及mstatus[3]是否為1，若條件則interrupt成立。  
   並且將mstatus[7](MPIE)儲存mstatus[3](MIE)之值，而mstatus[3]設為0。  
   最後將CPU執行的位址跳到ISR起始位址，並開始執行，直到ISR執行完成。
4. ISR執行完畢後，執行mret指令回到interrupt之前執行的位址。  
   並將mstatus register復原(mstatus[3](MIE)復原回mstatus[7](MPIE)的值)。

Superlint

移除錯誤前：



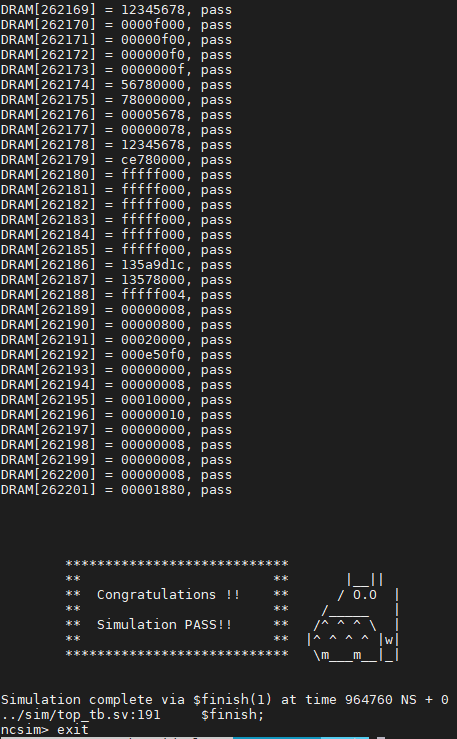


* Unequal length operands in bit/arithmetic operator in module/design-unit:
  + 使兩邊式子bit length相同
* Unreachable states found
  + Cache: STATE\_READ\_HIT & STATE\_WRITE\_HIT (instruction cache only)

這是因為我們將cache中的read hit的動作移到STATE\_CHECK中執行，因此這個state就被忽略，因此只需要移除即可。至於STATE\_WRITE\_HIT則是因為在instruction cache中並沒有write的動作，因此也可以移除。

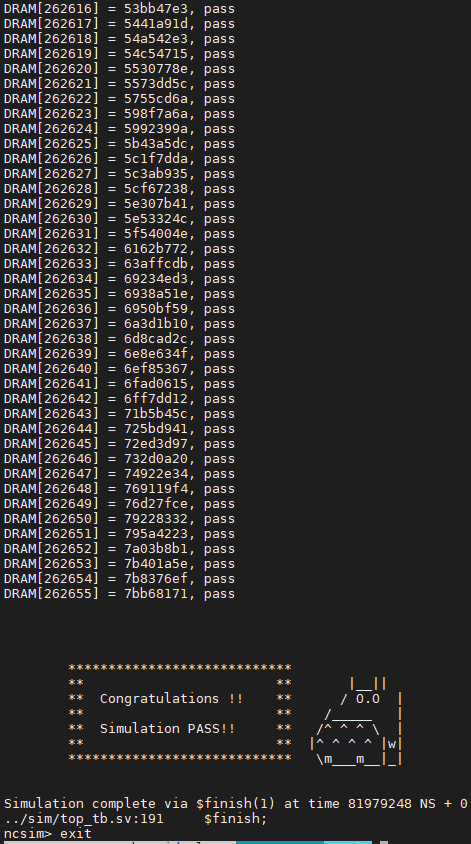
Performance and Area

Prog0 – make pr0



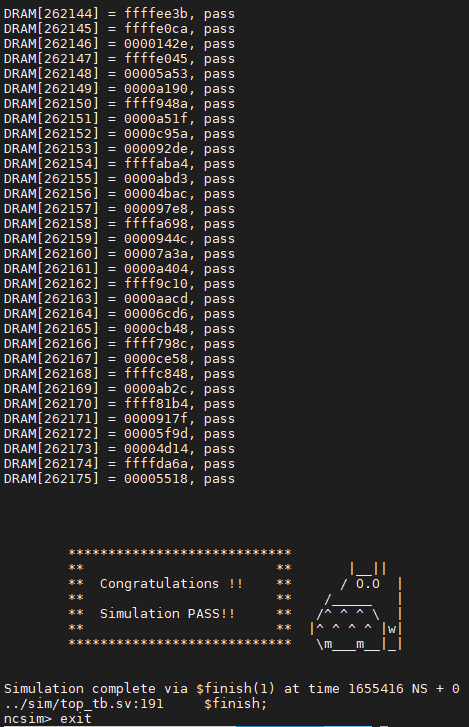
Simulation time: 964760 NS

Prog1 – make pr1



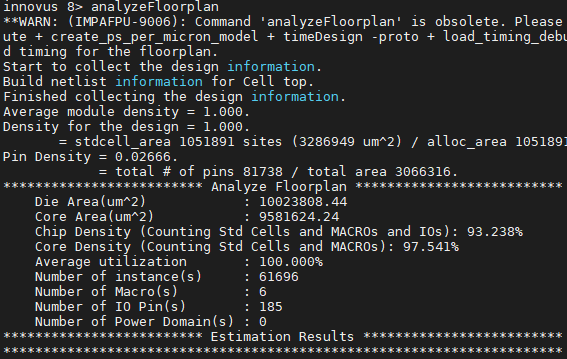
Simulation time: 81979248 NS

Prog2 – make pr2



Simulation time: 1655416 NS

Area

  
Area: 9581624.24