VLSI System Design

(Graduate Level)

Fall 2021

HW4 report

Must do self-checking before submission:

Compress all files described in the problem into one tar

All System Verilog 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 any waveform files in deliverables

Student name: 鄭丞祥、陳冠佑

Student ID: N26094891、N26090130

# Problem Description

設計CSR 、增加CPU指令以及sctrl wrapper，並將這些元件與CPU系統整合在一起

# Check List

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| RTL code | | Synthesis | | APR | |
| Prog0 | Pass | Syn0 | Pass | PR0 | Pass |
| Prog1 | Pass | Syn1 | Pass | PR1 | Pass |
| Prog2 | Pass | Syn2 | Pass | PR2 | Pass |
| Superlint | 98 % | | | | |

# Contribution

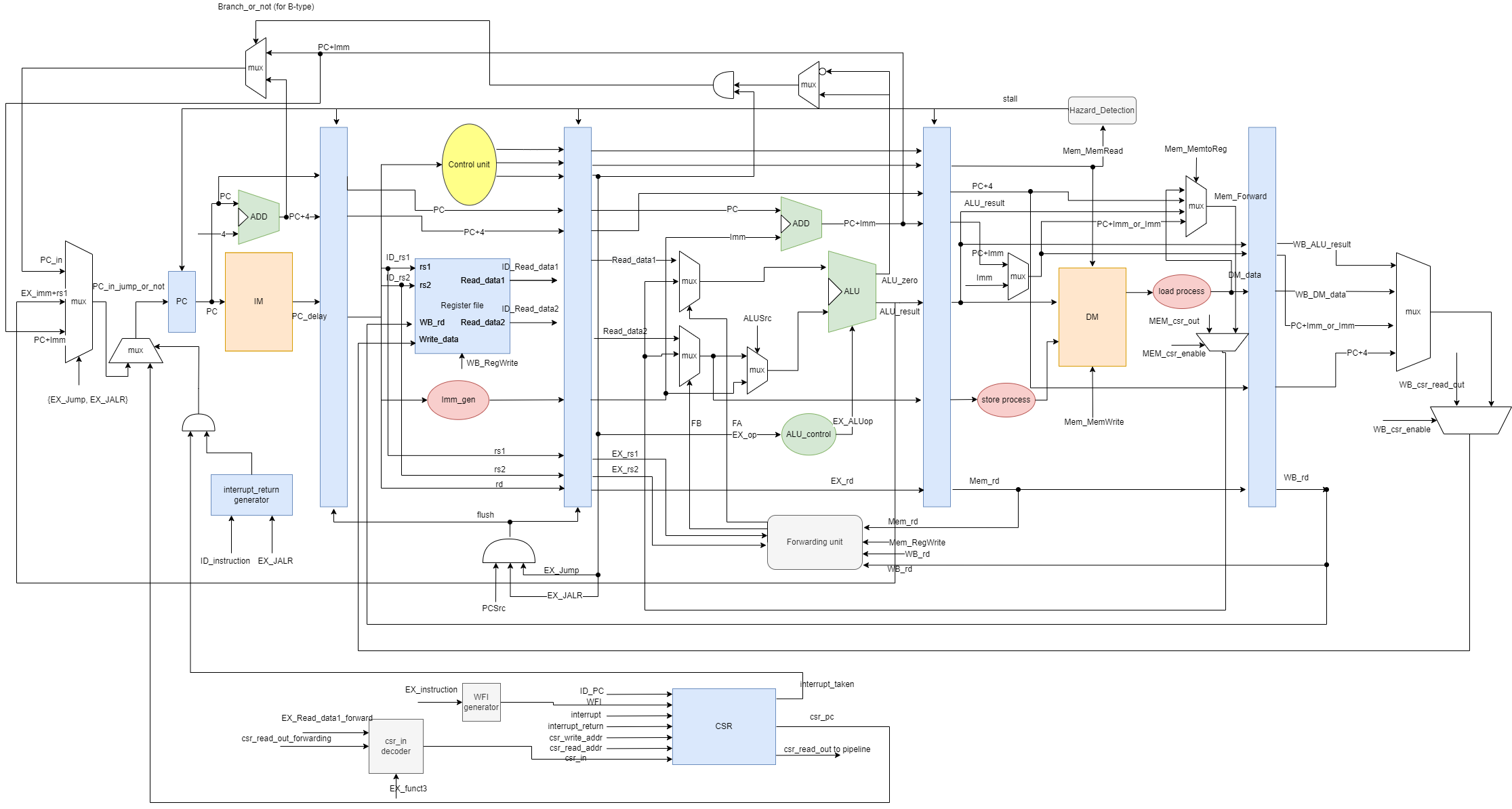
* 鄭丞祥(50%)
* 陳冠佑(50%)

# Problem 1

Booting + instructions verification.

在PROG0中，我們需要實作CSR且要在Instruction中加入對CSR指令的支援，以下各自說明。

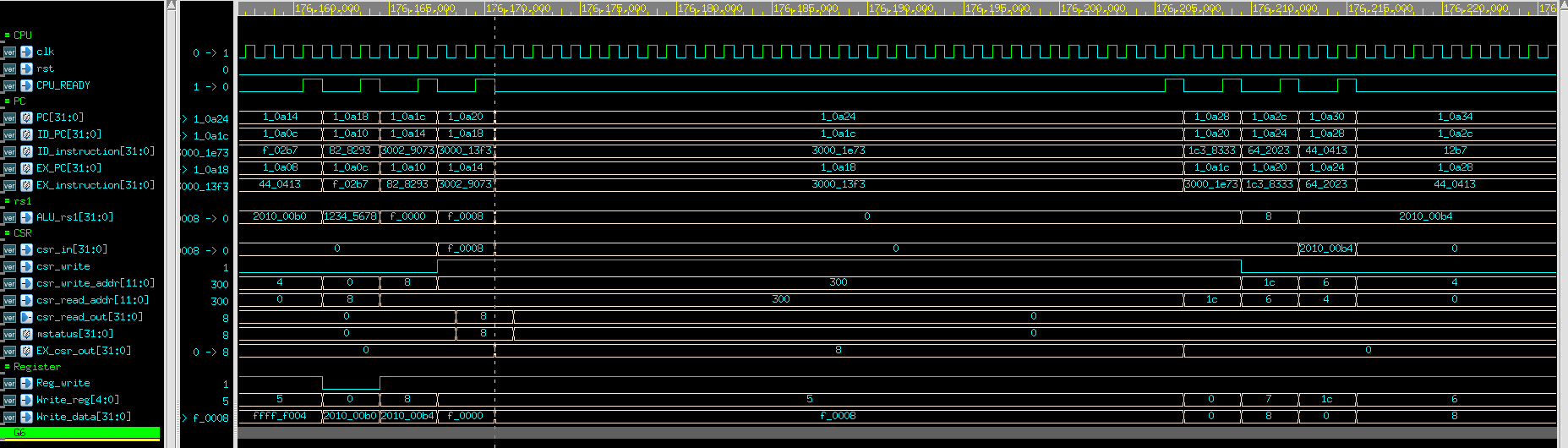
* Block Diagram



上圖為這次作業的block diagram，主要更動的部分為紅框標示處。因為這次作業要求CPU要能處理CSR指令，因此需要一塊CSR模組來處理相關的CSR訊號線。這塊模組主要會接收WFI以及CSR\_IN，以及要暫存的PC位置當作輸入，WFI的用途是讓CPU知道說現在它待會會不會需要interrupt訊號，如果有的話會提早先拉起等待sensor給的interrupt訊號來臨才可以真的跳到中斷位置執行中斷指令，而CSR\_IN的用途而是判斷說如果今天這個指令是CSR相關指令的話，會需要將計算出來的數值存入到CSR裡面那多個相關暫存器，比方說mstatus，mie等暫存器，目的是為了之後interrupt動作結束後能正確回到原本位置以及能處理原本欲處理的指令。而雖然說外部的sensor control會給CPU interrupt訊號，但實際上CPU會不會執行跳躍指令是看interrupt taken以及interrupt return，當這兩個訊號有一者為1時代表待會CPU要跳到中斷程式，而中斷跳躍這項行為我們覺得對於CPU來說可以視為一種JUMP/BRANCH指令，因此會將下個時刻要跳到中斷程式的PC給到PC\_jump\_or\_not來傳遞下去。而在執行CSR指令的硬體時，也需要特別注意到Forwarding的問題，如果今天有連續兩筆CSR指令進行到EX階段和MEM階段時，會有資料相依性的問題存在，因此需要像第一次作業一樣考慮到如何在正確的時間點將資料forward回去，對應於圖中最右邊兩個紅色框框。

* Waveform

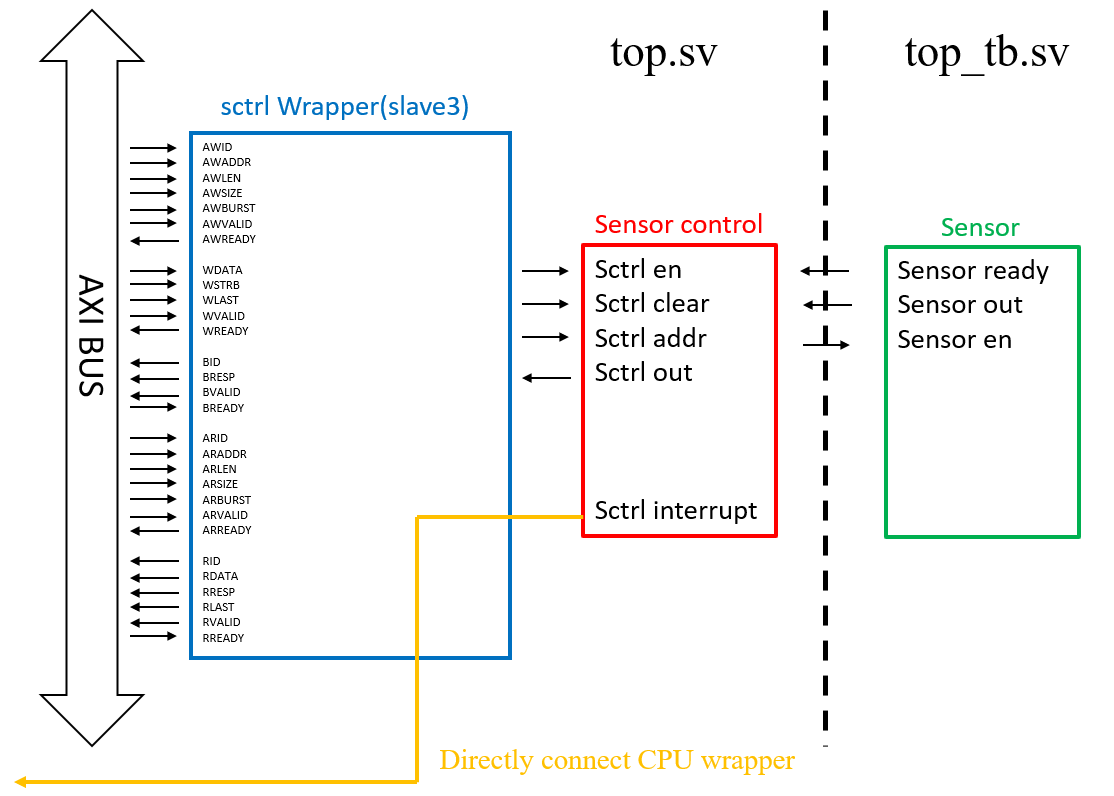
是用來記錄細整個電路的運作狀態的，其中有八個register記載了不同資訊，例如mcycle記錄整電路運行多少cycle，CSR指令除了WFI和MRET以外都大同小異，所以在此只列出其中一個的波形。

此藍色框框10a18為CSRRW指令由圖中紅色框框可以看到將原本CSR(mstatus)的值8寫回rd暫存器(rd = 7)，此外在看橘色框框可看出有將rs1之值更新到CSR中(mstatus 8 0)，代表CSRRW指令執行無誤。

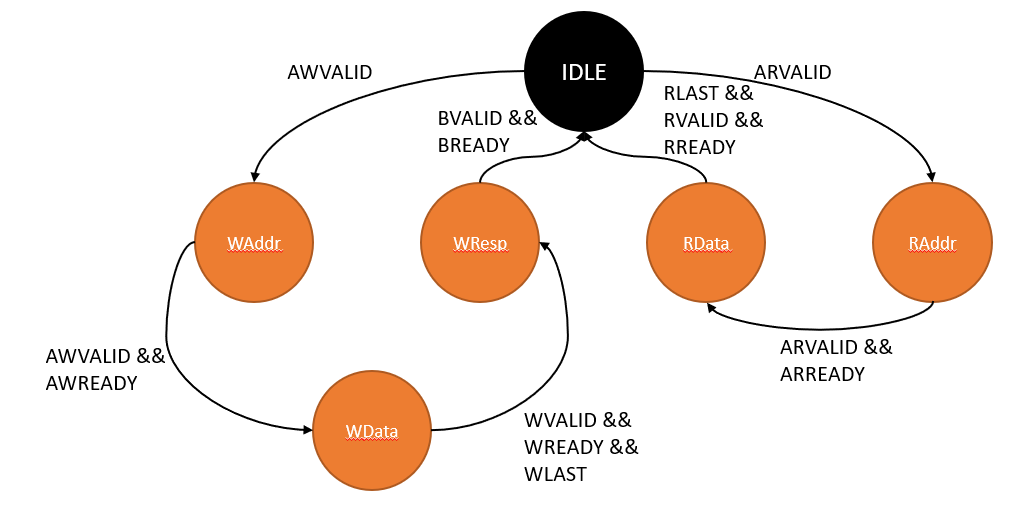
# Problem 2

Booting + interrupt verification.

在PROG0中，我們需要實作出sensor control wrapper以完成中斷程式，以下各自說明。

* Block Diagram
* Sensor control wrapper

Sensor control 的wrapper與SRAM wrapper較相似與bus的溝通都一樣，因此狀態機的設計也是一樣的設計，如下圖可見，比較不一樣的設計部分為sctrl en、 sctrl clear的訊號給法，我們的設計為按照助教規定之用寫入之addr以及寫入資料是否為非零做判定給值，除此之外因為sctrl interrupt訊號為interrupt訊號，因此需要講求即時性，故將他在top module直接對接CPU wrapper給入CPU做中斷。

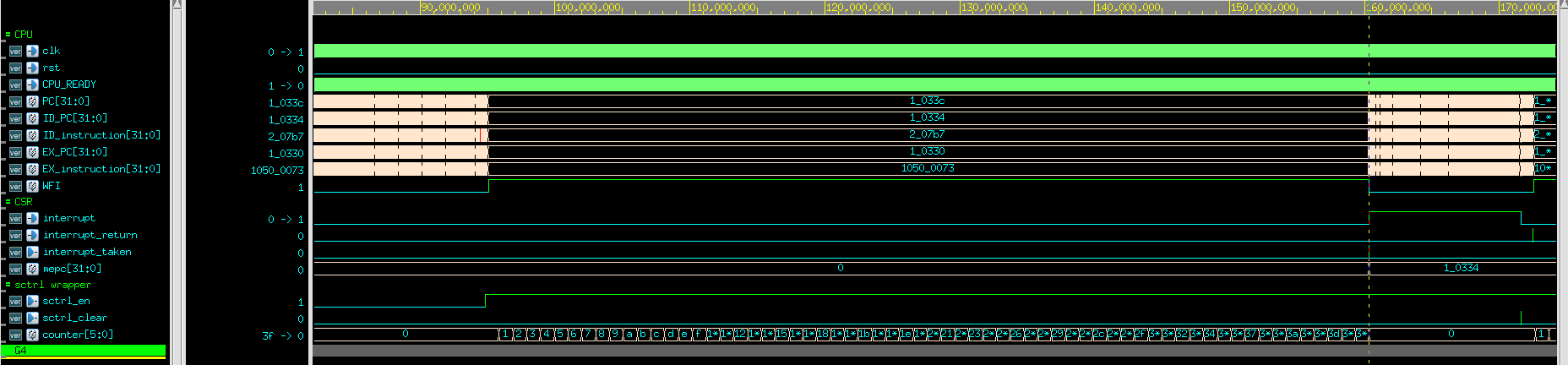


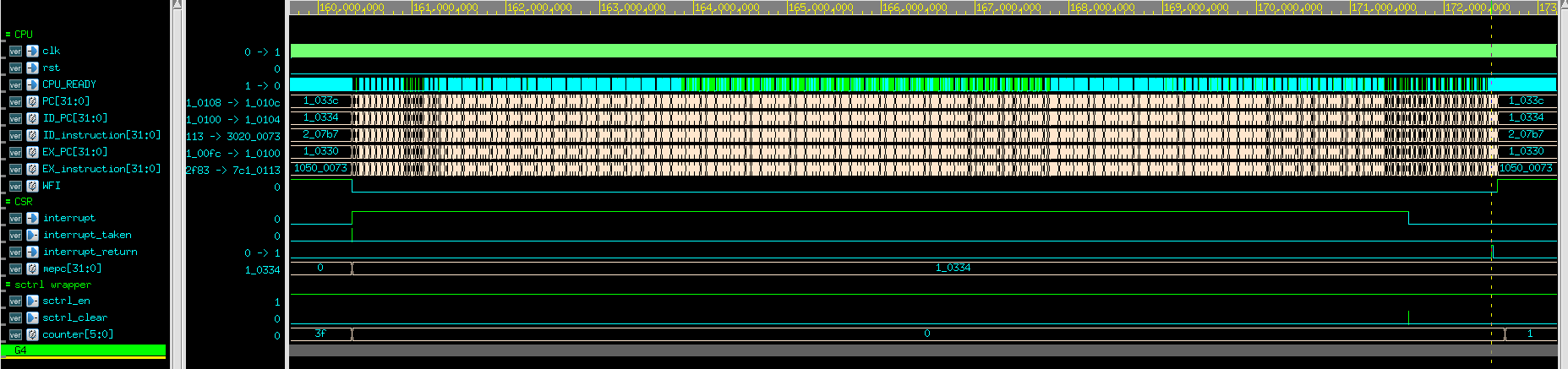
除此之外，因為助教有規定若是要從sensor control拿資料必須by pass cache，因此我們有在data cache裡面多加幾個state利用cacheable去跳。

* Waveform

此兩張波行是想要描述合時會產生interrupt，以及interrupt結束的狀況，後面會在對兩種interrupt進行更進一步的分析，interrupt type1(wait for interrupt)、interrupt type2(隨機interrupt)。

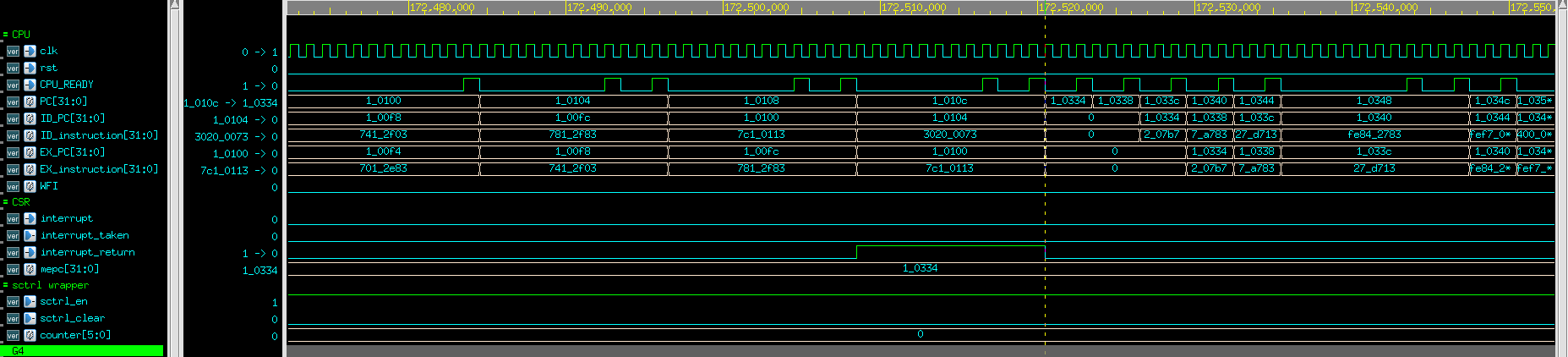
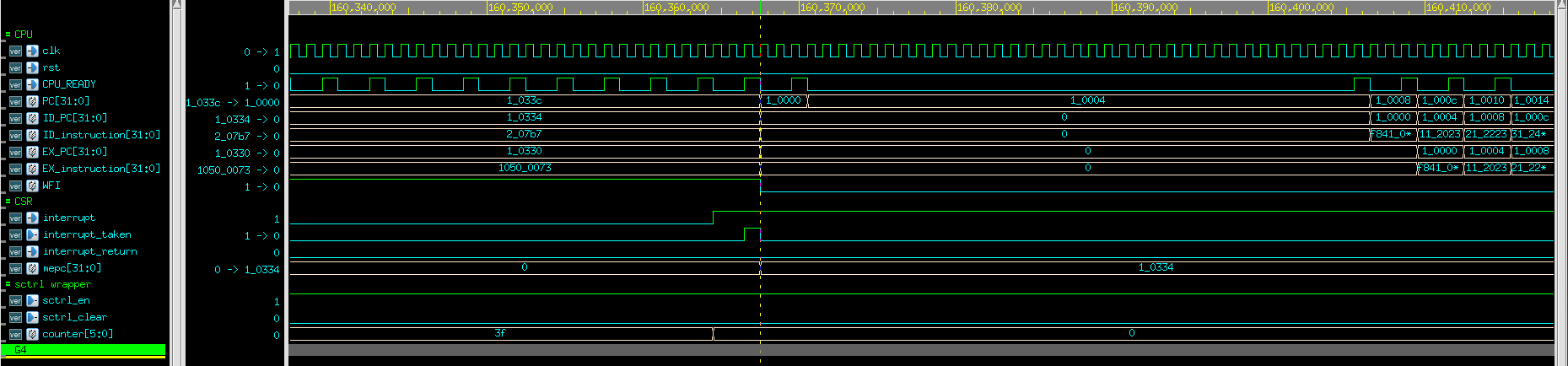
* Interrupt發生

 由圖中可以看出當sctrl\_en拉1 後(紅色框框)counter就會開始數數到63代表64筆data都搬到sensor control的local memory，因為這邊是執行到WFI指令因此中間的timing CPU都會stall住，interrupt訊號及會拉起(藍色框框)，將當前PC存入mepc暫存器中(我們的設計為存ID\_PC)，接著進行中斷指令開始將資料搬到DRAM。

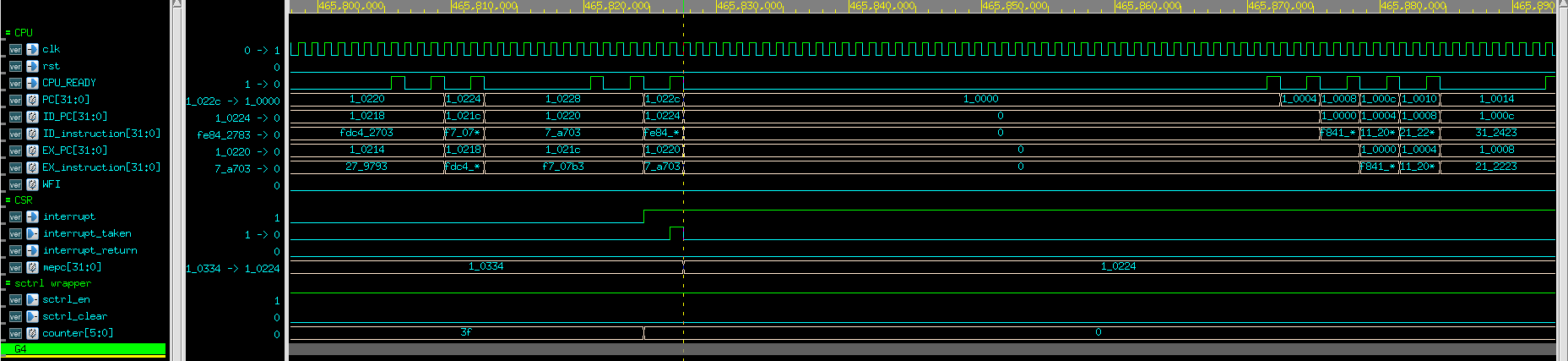
* Interrupt結束

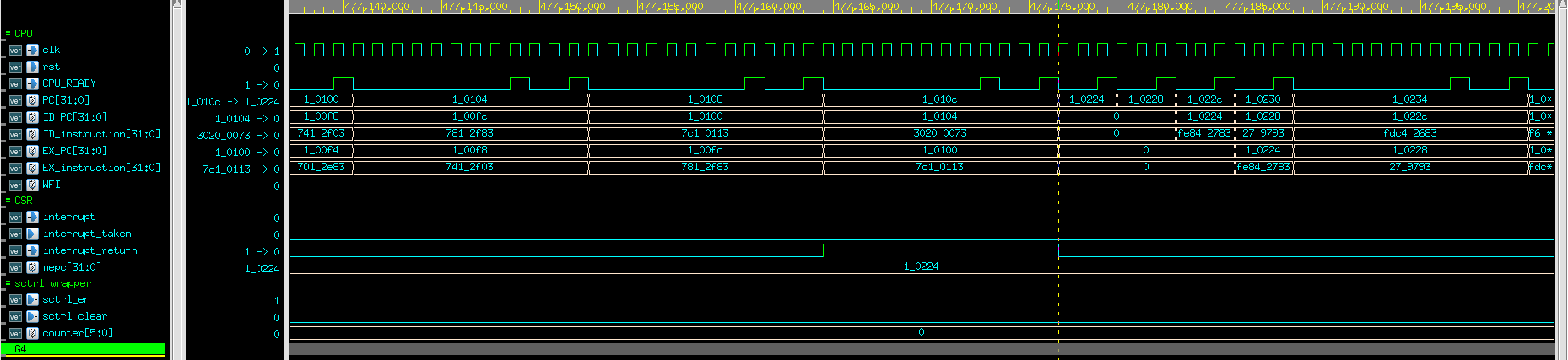
由圖中可以看出中間為在執行interrupt後的指令，搬運data到DRAM當執行interrupt所有指令後，會執行MRET這條指令，當執行這條指令時(藍色框框)，可以看到紅色框框interrupt return訊號拉起，代表interrupt指令執行完，並將mepc給回PC，繼續執行main的指令。

* Interrupt type1(wait for interrupt)

當前執行的指令為WFI指令(10330)，因此CPU stall住，直到interrupt taken(紅色框框)，可以看到mepc將10334存住(wfi的下一個指令)，且也可以看到PC跳到10000執行中斷後的指令。

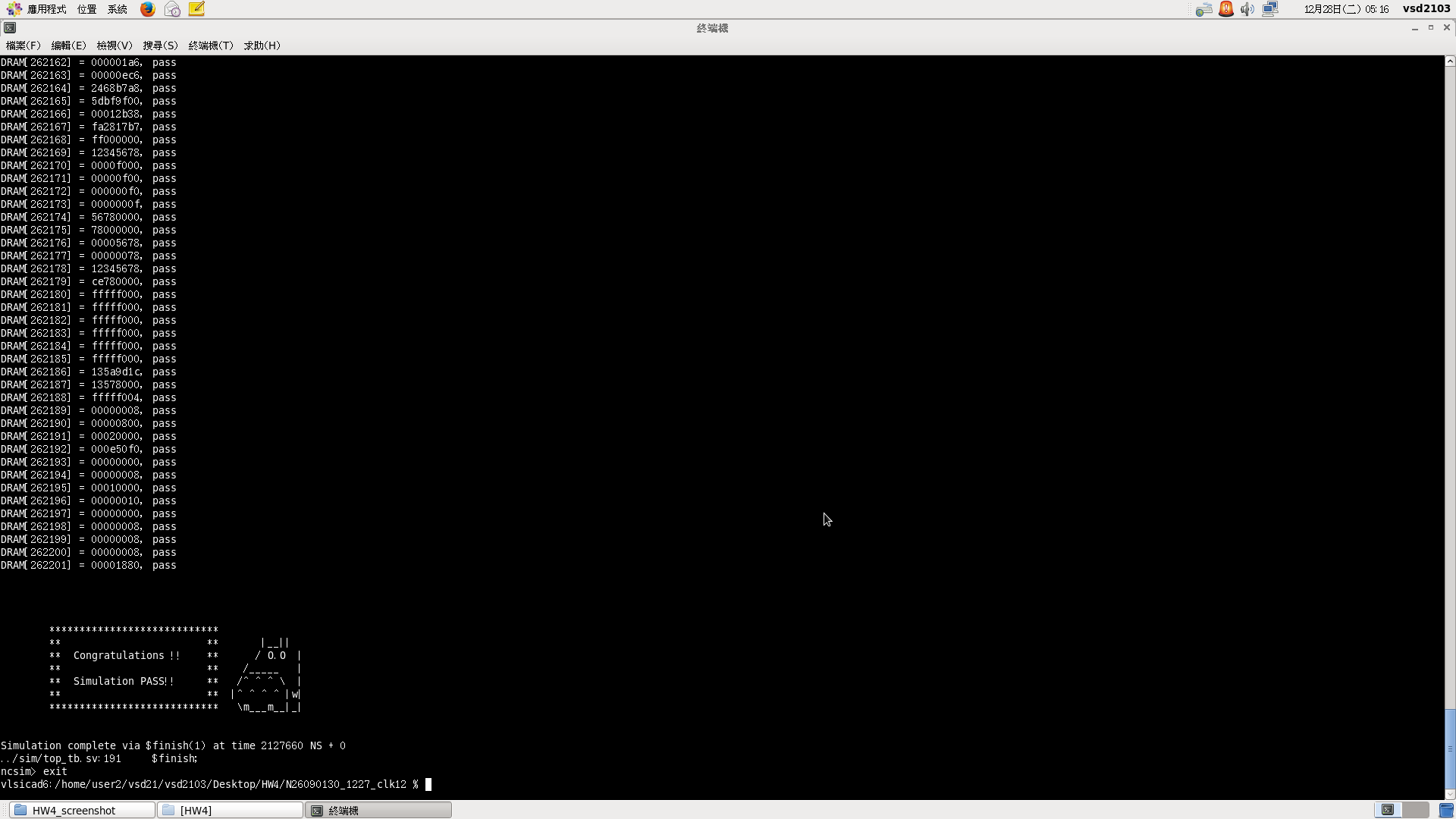
由圖可以看出當中斷程式執行到最後一個MRET(30200073)時代表中斷程式執行完成，由紅色框框可以看到interrupt return拉起，且將mepc之值存回給PC(10334)，CPU繼續執行中斷前之指令。

* Interrupt type2(隨機interrupt)

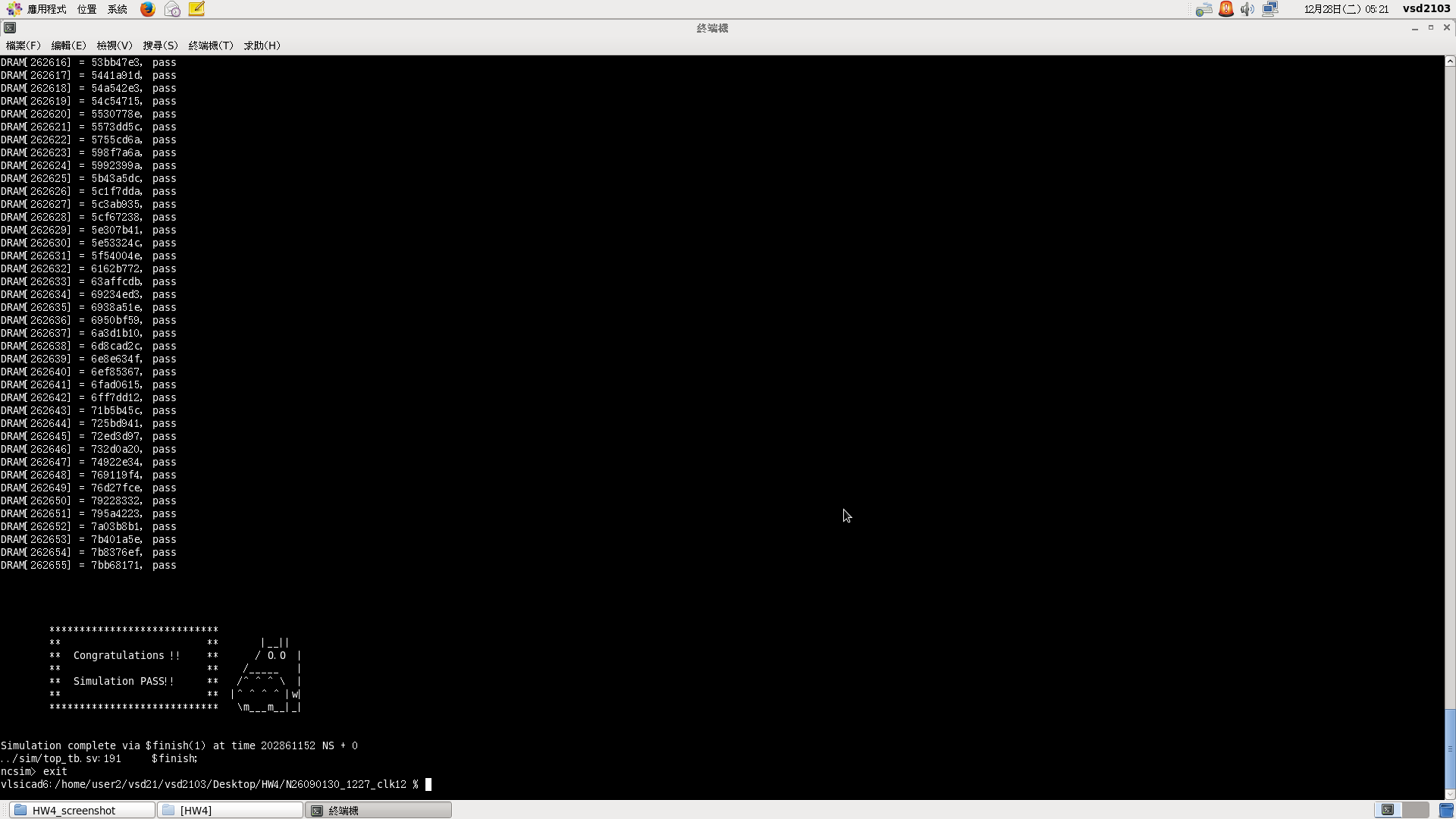
此時不像上述一樣是執行到wfi指令等待interrupt來，而是程式執行到一半interrupt產生，因此必須馬上中斷把當前PC存住，跳到中斷程式去執行中斷程式(10000)，因此圖中可以看出interrupt taken(紅色框框)，可以看到mepc將10224存住，且也可以看到PC跳到10000執行中斷後的指令。

由圖可以看出當中斷程式執行到最後一個MRET(30200073)時代表中斷程式執行完成，由紅色框框可以看到interrupt return拉起，且將mepc之值存回給PC，CPU繼續執行中斷前之指令(10224)。

* Prog0測試結果:

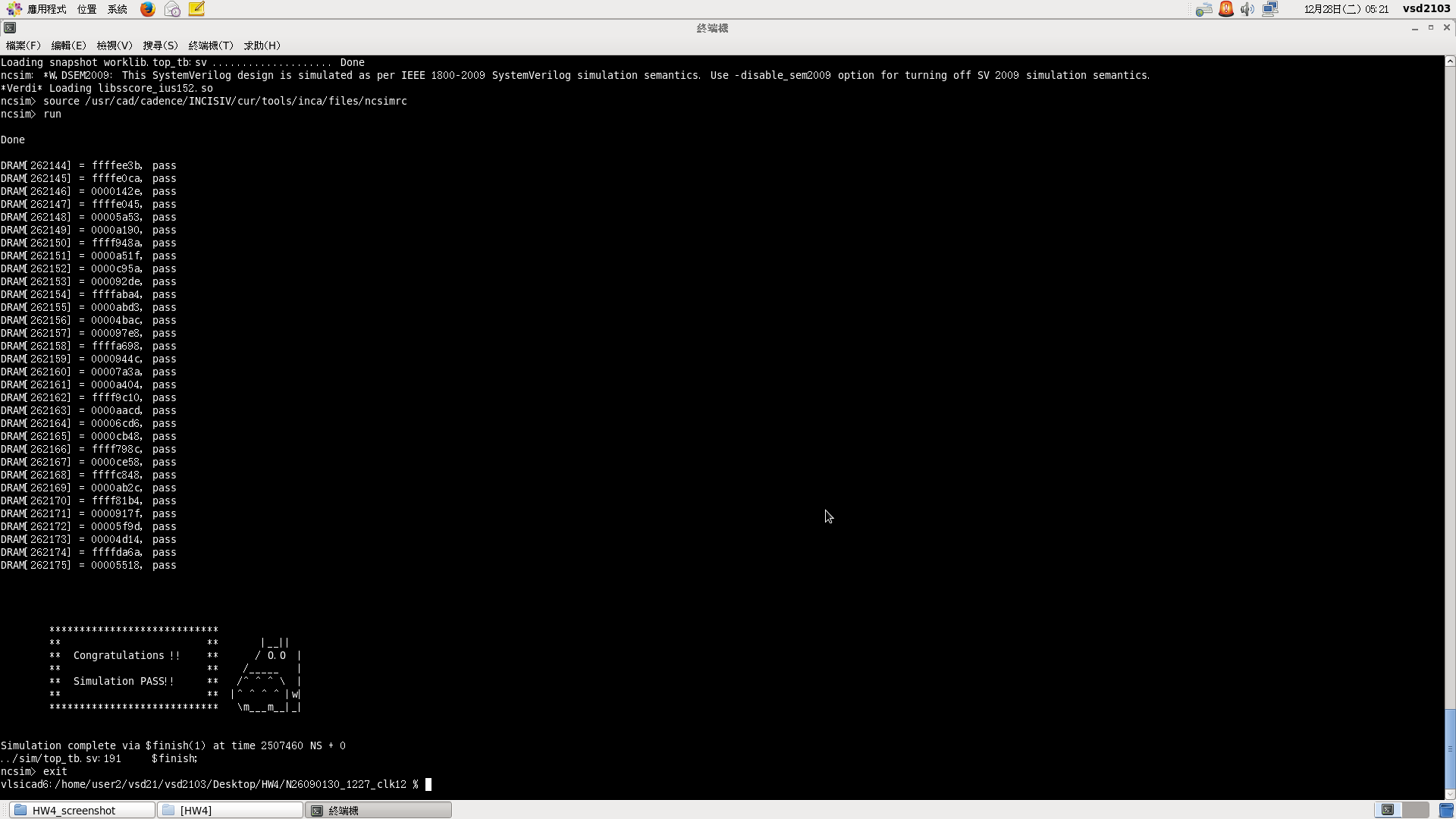


* Prog1: booting + interrupt

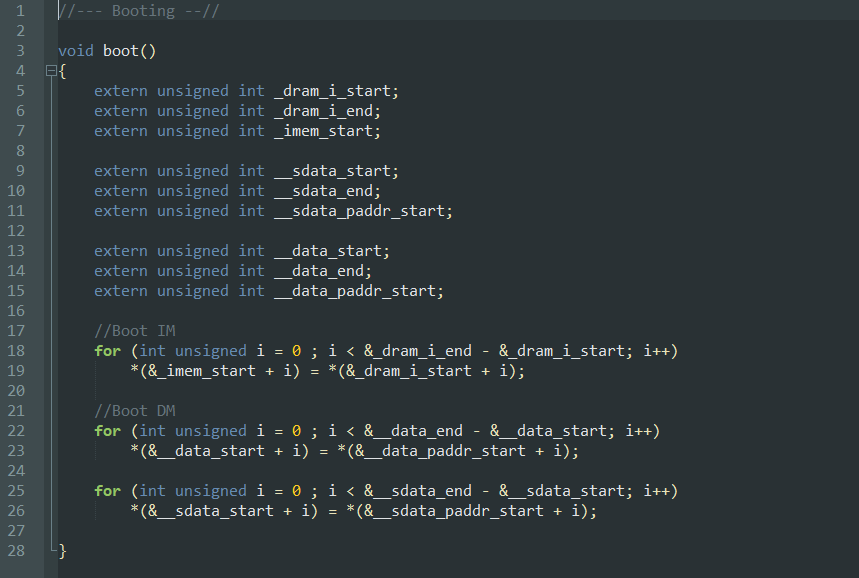


* Prog2: Matrix multiplication

將兩筆二維矩陣透過三層for迴圈來達到矩陣相乘的目的

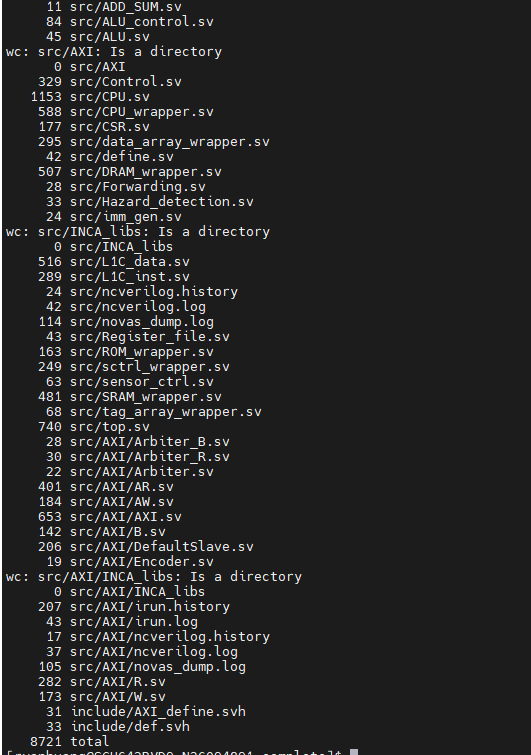


* Boot.c

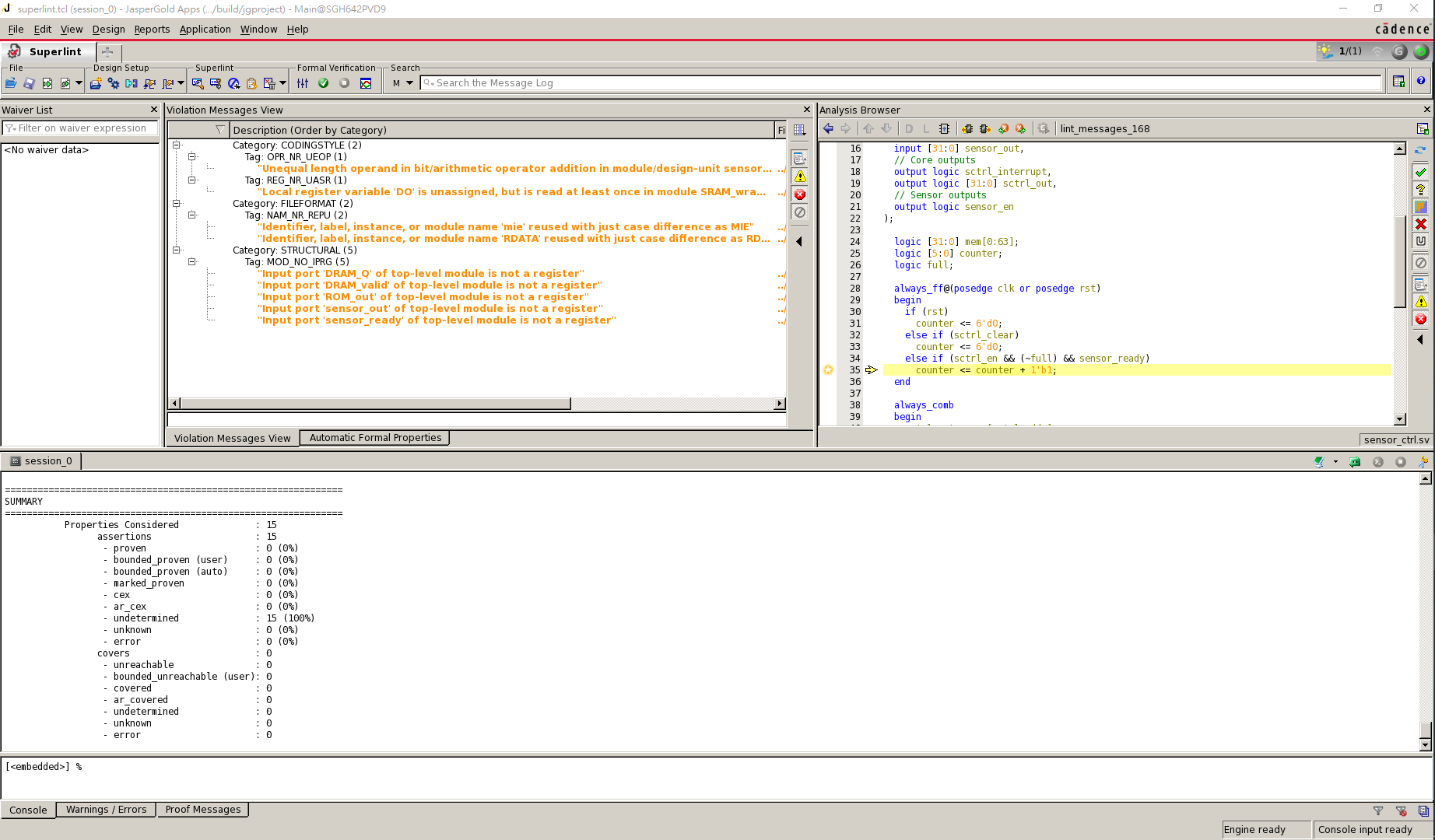


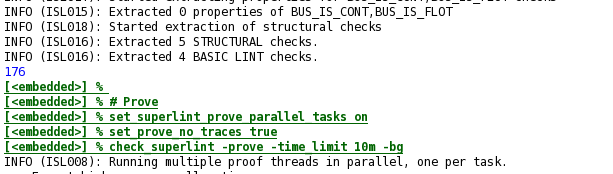
Boot.c其實就如同助教在投影片講述的，大部分都被行為都被定義得很清楚了，所以只要清楚知道要從哪裡開始搬運資料，要把資料搬運到哪邊就可以，例如\_dram\_i\_start就是DRAM的指令起始位置，會把牠搬運到\_imem\_start的相對位置。

# Number of lines of RTL code: 8721行



# Superlint result: 176





第1個warning為sensor\_ctrl裡面的counter壘加的bit數不對counter + 1’b1應為counter + 6’d1但因為sensor\_ctrl不是我們寫的，因此沒有特別做修改。

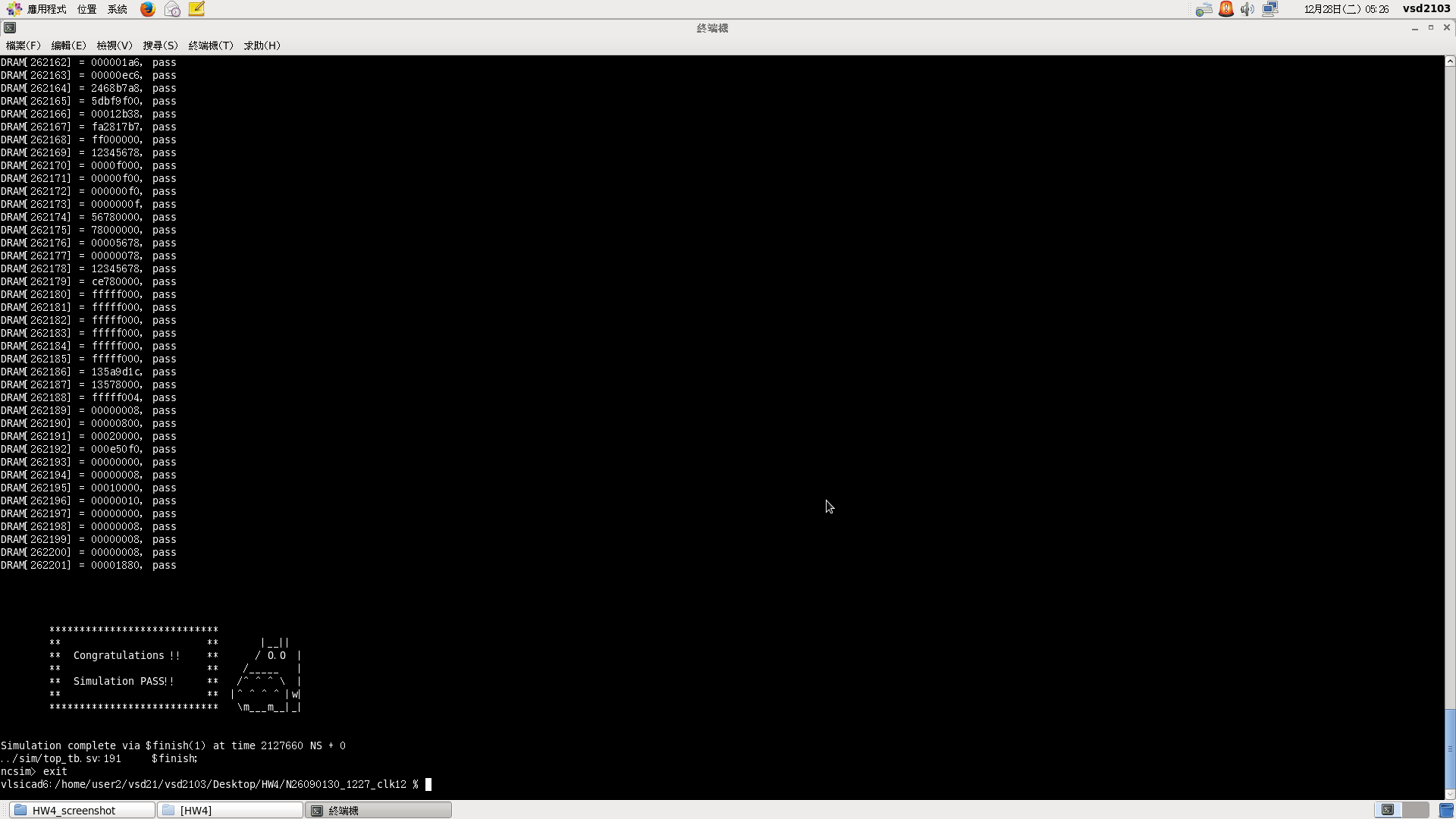
第2個warning為上次就有的warning，原因為DO命名的關係，並沒有assign給他值但卻把它的值給別人，但其實我們是有給DO值的在SRAM中，只是寫法為DO[i]，賦予每一個bit值，不是直接給DO一個值，所以系統出現這個warning提示，並不影響。

第3個warning為定義的腳位出現mie與MIE大小寫差異，提醒我們要清楚這是不同之定義，但並不影響code的執行，所以並沒有特別修改。

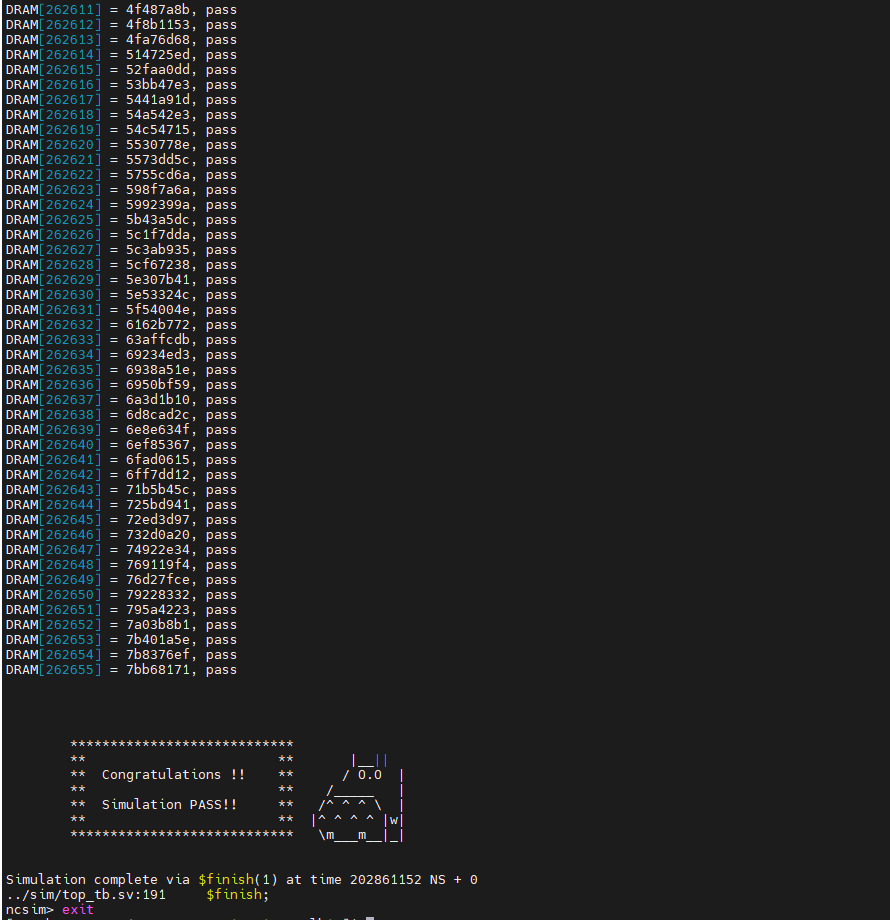
第4個warning為module定義的腳位RDATA與我們的狀態機的一個狀態命名一樣，但並不影響code的執行，所以並沒有特別修改。  
第5~9個warning為DRAM、ROM與sensor control外接的腳位，但因為外接的DRAM、ROM與sensor control不是我們寫的，因此這裡我們也沒特別做修改。

# Synthesis:

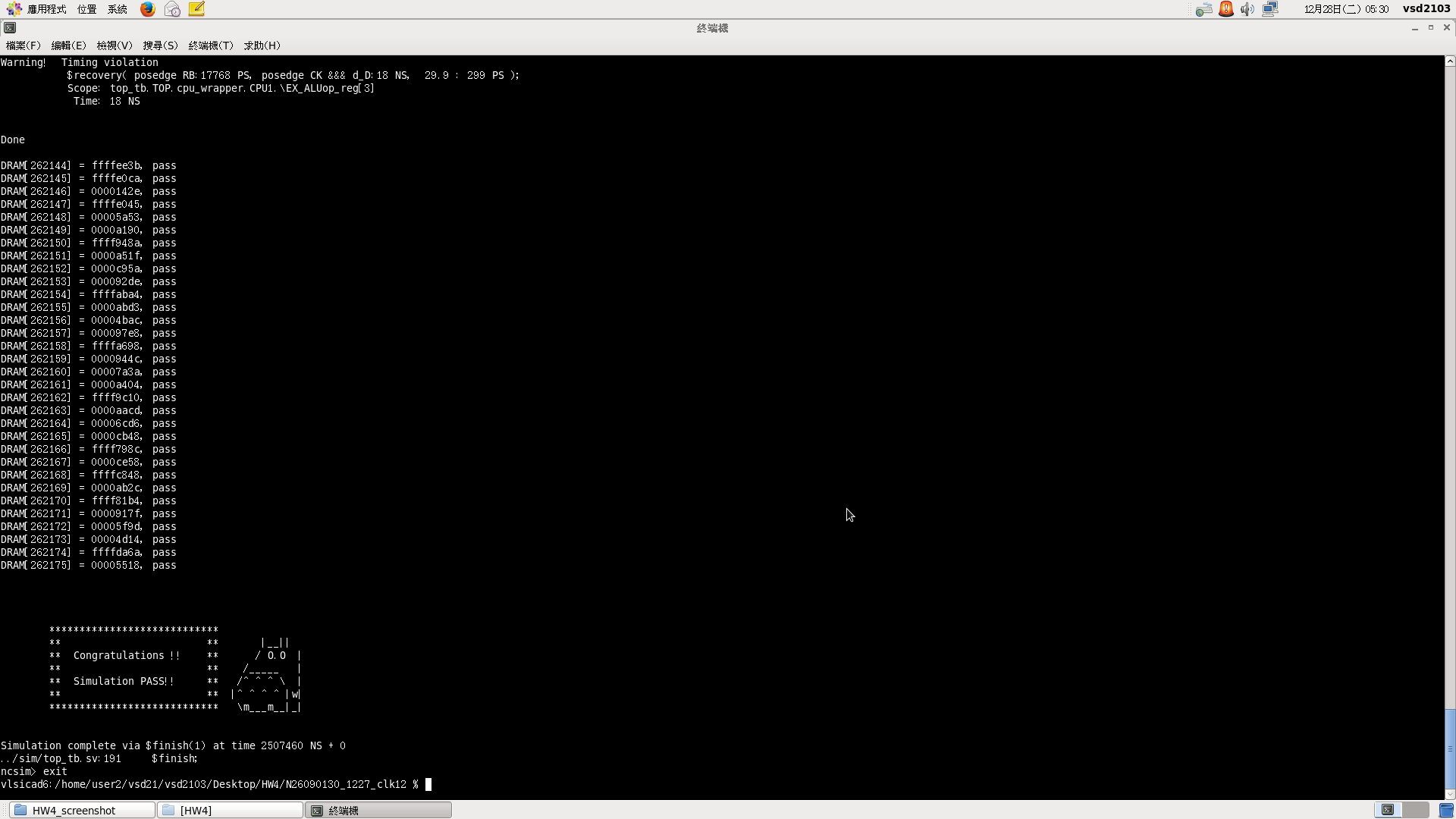
* CLK -> 12
* Prog 0:



* + Prog 1:

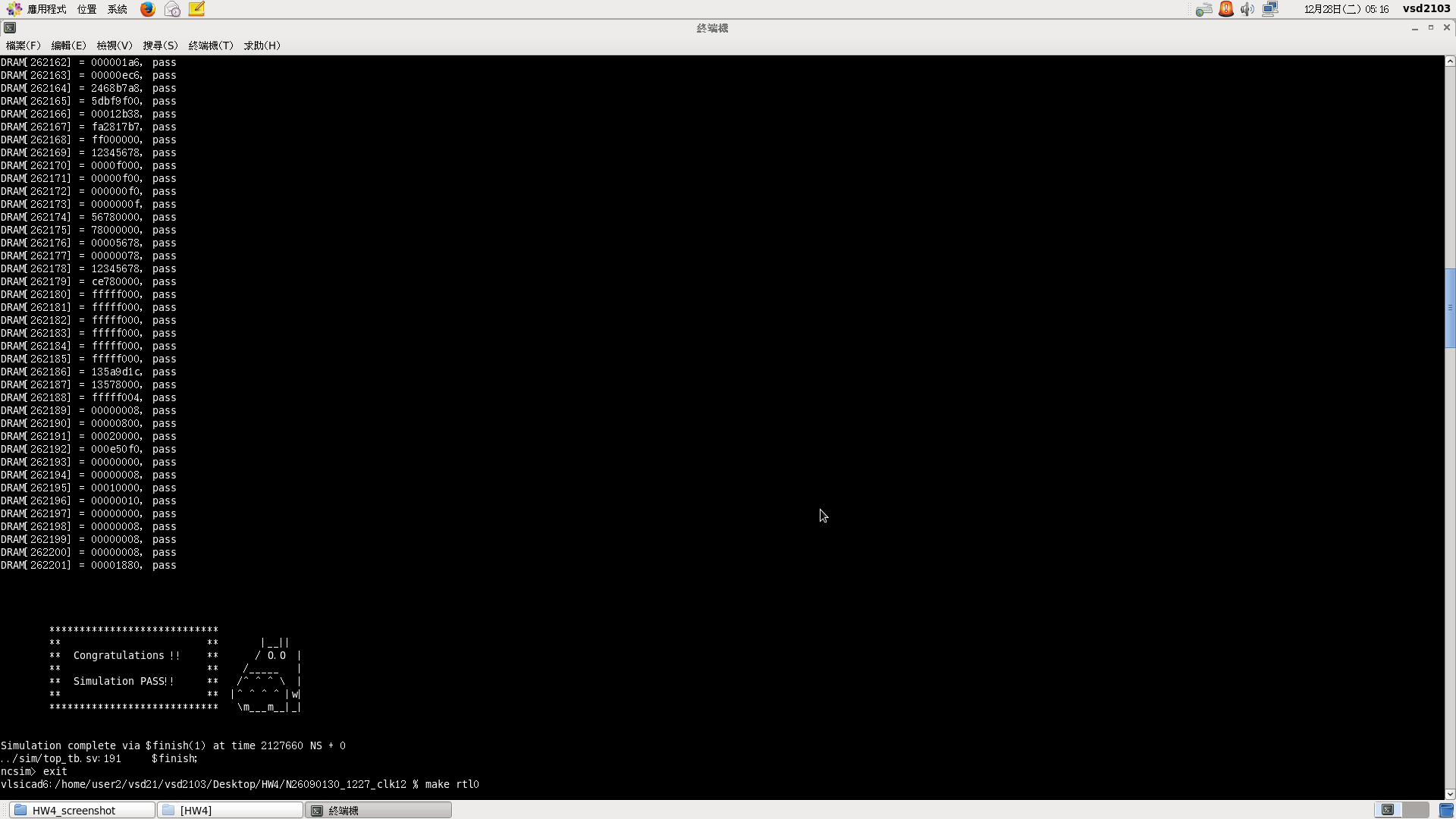


* + Prog 2:

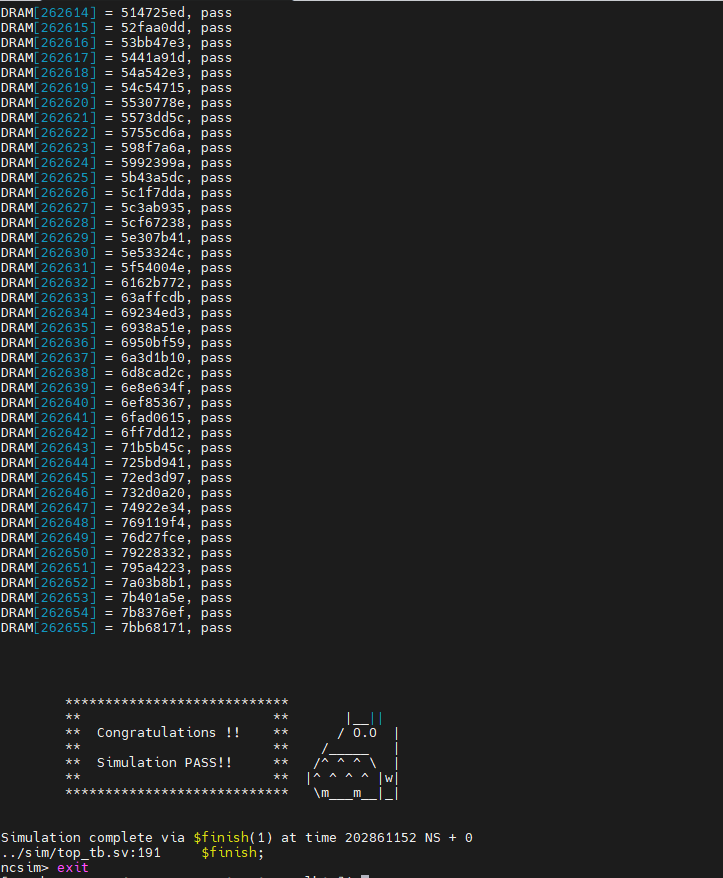


# APR:

* CLK -> 12
* Prog 0:



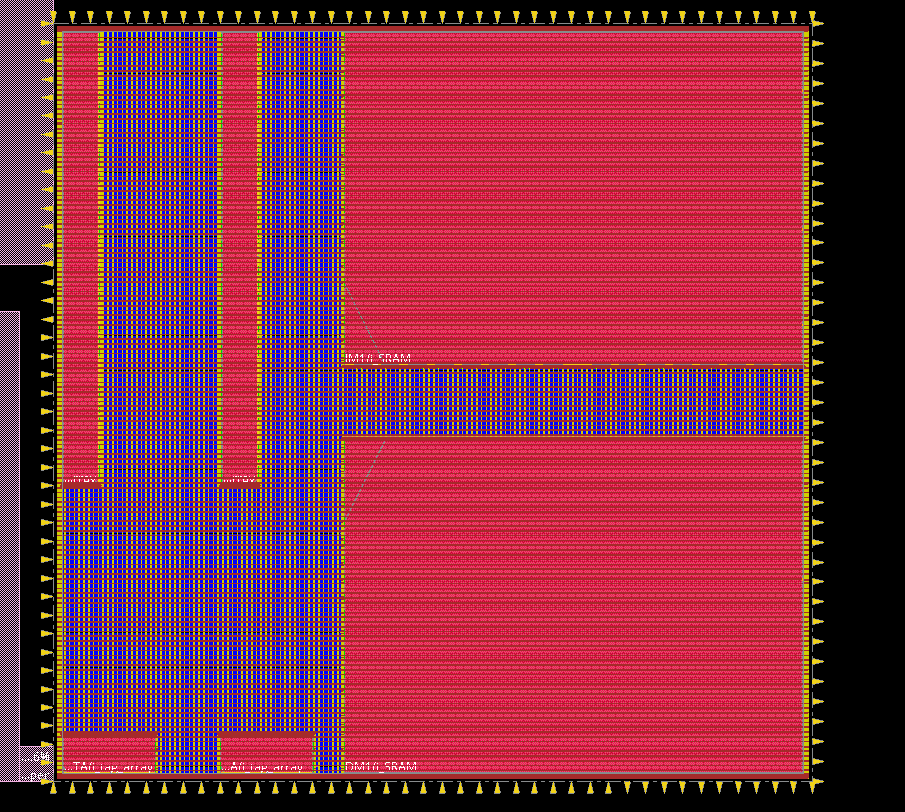
* + Prog 1:

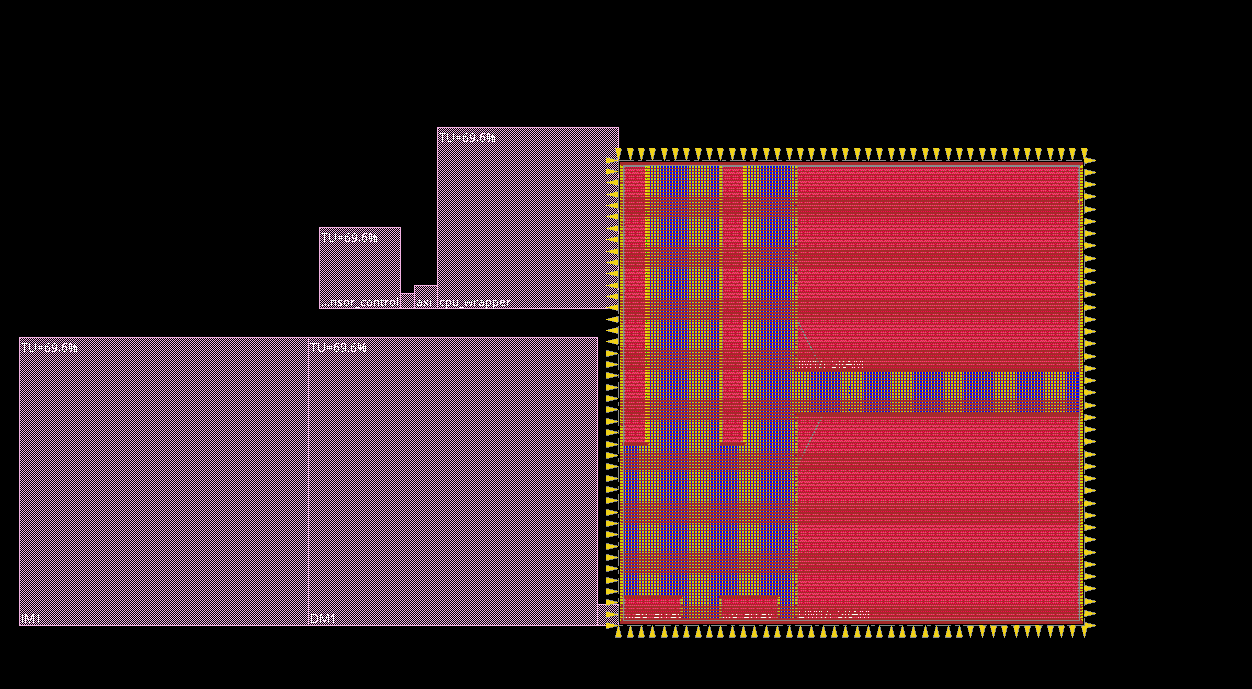


* + Prog 2:

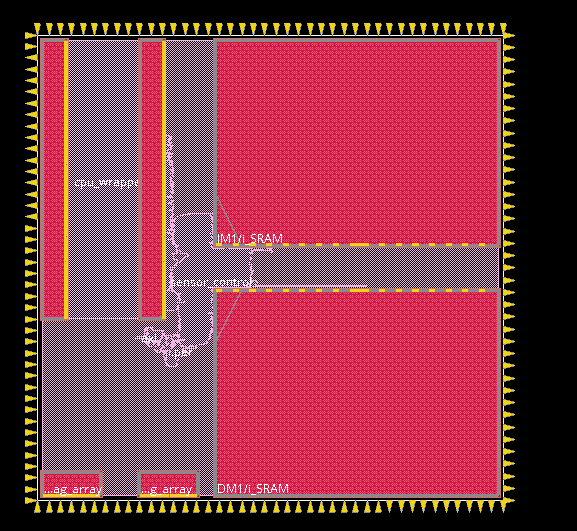


* + Floorplan View:

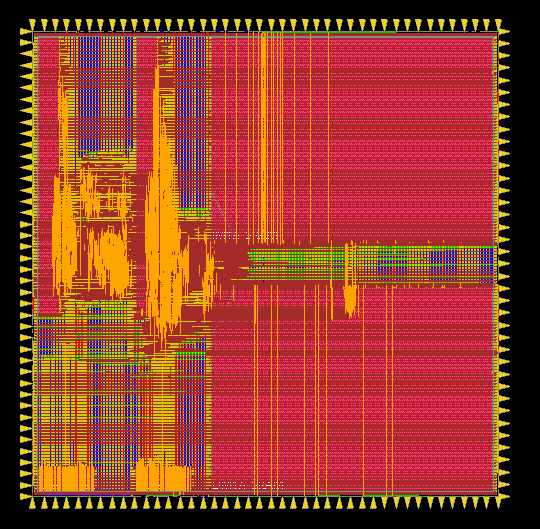




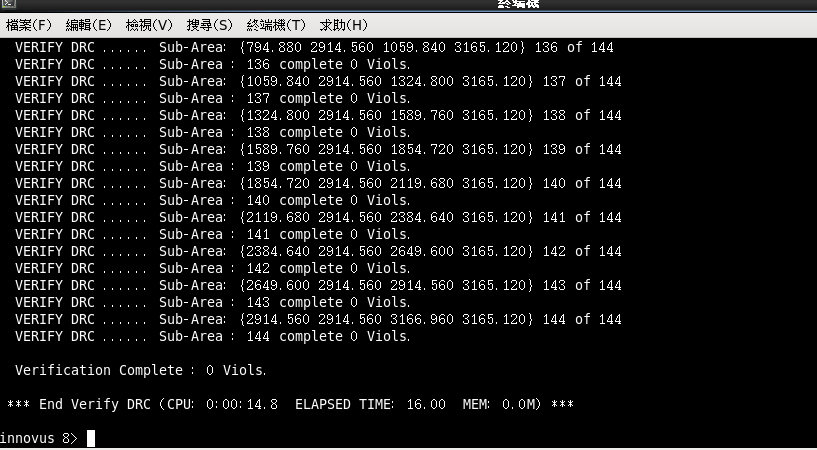
* + Amoeba View



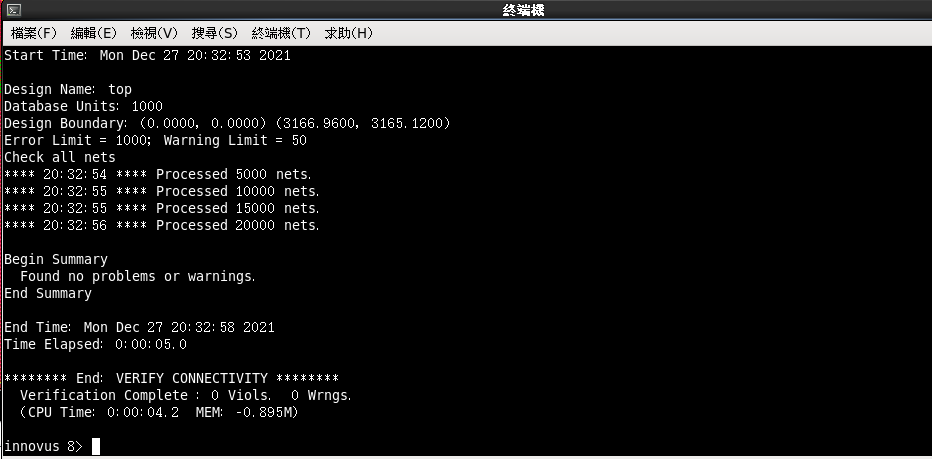
* + Physical View



* + Geometry Verification(DRC)



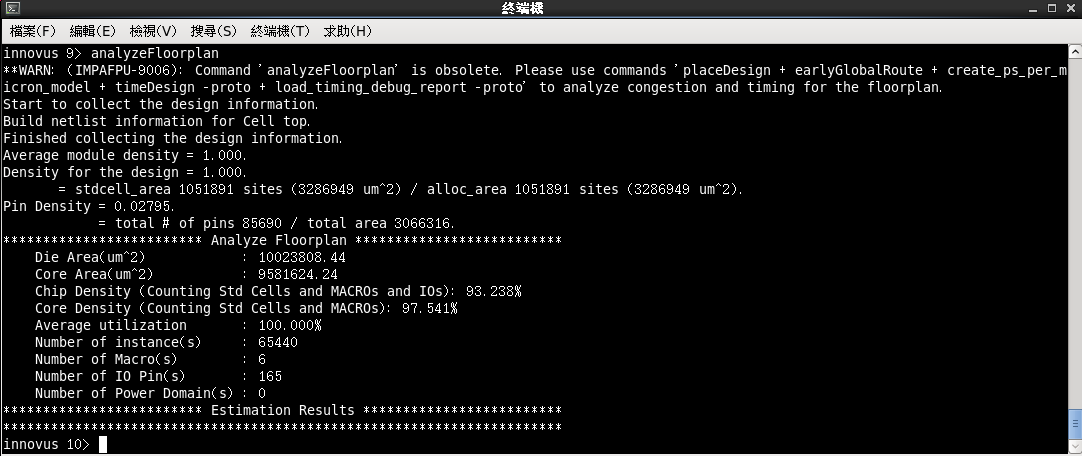
* + Connectivity Verification(LVS)



* + Antenna Verification



* + AREA



Performance: (2127660 + 202861152 + 2507460) = 207496272 NS

AREA: 9581624.24

PA = 207496272 \* 9581624.24 = 1.98\*

# Problems encountered

這次作業遇到的問題可以分為兩個部分，interrupt以及APR。

在CSR的設計時，我們遇到的問題主要是在增加CSR時一開始我們並沒有把CPU的forwarding考慮清楚，因此遇到forwarding的問題，在打開波行後才發現一些forwarding的線沒有考慮到CSR，因此出錯。此外再加上interrupt後，也是遇到很大的問題是一開始我們並沒有很了解interrupt的機制以及總共會interrupt幾次等問題，也是在看了助教的testbench後才比較了解interrupt的運作機制，此外我們這組這次有遇到合成前pre-sim全過但合成後post-sim沒有過的問題，在經歷一番波折後才找到錯誤是因為我們mux的其中一個訊號有一個cycle是unknow但在pre-s im時不會出現問題，但在post-sim會出錯，也讓我們知道pre-sim的一些unknow訊號不能隨便忽略。

對於APR的部分，在這次遇到的最大問題是對於軟體上的使用不熟悉，APR工具的操作流程即為複雜，並且每一步都必須要小心翼翼地去點選，才不會導致DRC和LVS出現錯誤，比方說最初在擺放模組的時候，如果沒有將模組放大最大，將模組完整的貼合到邊線上的話，就相當容易讓DRC沒過，而模組的擺放位置如果擺設不好，也很容易使後續軟體在做place & route的時候優化不易，可能會跑很久，甚至跑了多次一樣會讓WNS為負，這些小細節都是必須要考慮的。此外還有像是APR的script必須要更改也是我們這次APR以為快要做完的時候才知道，以及syn會過得cycle數在APR上不一定可行等等，都讓我們在實作上反覆了好幾次。

# Lesson learned

鄭丞祥: 在這次作業中我們遇到最大也最透痛的問題可以說是interrupt的部分，因為原本對於interrupt到底是怎麼運作得不慎了解，寫了很多版模擬出來的結果不盡理想，為此我們而外上網看了許多教學以及向已經完成的人請教，而在好不容易終於知道interrupt的運行模式後，本來以為可以順利的開始進行合成後，卻遇到了最大的問題就是我們的電路在第二次sorting的時候會無法正常運作。一開始我們毫無頭緒，認為sort沒道理會錯，因為第一次sort對，理論上第二次也要對。但是當我們想要為此開始去debug的時候卻遇到大難題就是這次的測資實在是太多了，導致我們想要產生後面的波形時會需要花上龐大的時間去等待。後來經過一番折磨後以及請教助教後才慢慢找到問題，原來問題出在我們之前的lw設計會導致interrupt在執行時會少flush一個cycle，PC雖然都有正確的flush調，但是那個時間點的指令仍然還在所以導致運算多了一次。另外就是postsim時也遇到大問題就是因為我們電路在presim時就會有些unknow訊號，這些unknow訊號我們起初沒有很在意，沒完全沒想到就是因為它導致我們postsim大錯特錯，花了不少時間。這次作業不僅在interrupt、APR上收穫許多，也讓我們知道一個乾淨、沒有冗餘電路的設計以及debug技巧是多麼重要。

陳冠佑: 這次作業我認為一開始會一直撞牆是因為不太懂interrupt的機制導致不太知道要怎麼設計，也去找了許多的網路上資源幫我更了解中斷，除此之外也讓我知道可以去看testbench就可以比較知道一些行為，當整體電路設計完後，進入debug才是真正噩夢的開始，我認為因為我們之前有一些設計沒有寫得非常乾淨，所以在加上interrupt後，會有一些special case要另外debug，像是我們一開始第一次sort會對第二次sort會錯後來才發現是因為lw指令接interrupt時，我們的設計會少flush一個cycle，因此導致出錯，另外就是上述所說的pre-sim unknow訊號沒有處理乾淨也花了我們不少時間，這次作業讓我學到interrupt的機制以及相關資訊，也讓我更深刻體會到debug的時間永遠都會是設計電路的好幾倍，也謝謝助教這次的幫忙。