VLSI System Design (Graduate Level)

Fall 2020

HOMEWORK III

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 any waveform files in deliverables

Student name: \_\_黃建智\_\_ \_\_高士鈞\_\_

Student ID: \_\_N26080020\_ \_\_N26094922\_

Contribution: \_\_\_(50%)\_\_\_\_ \_\_\_(50%)\_\_\_

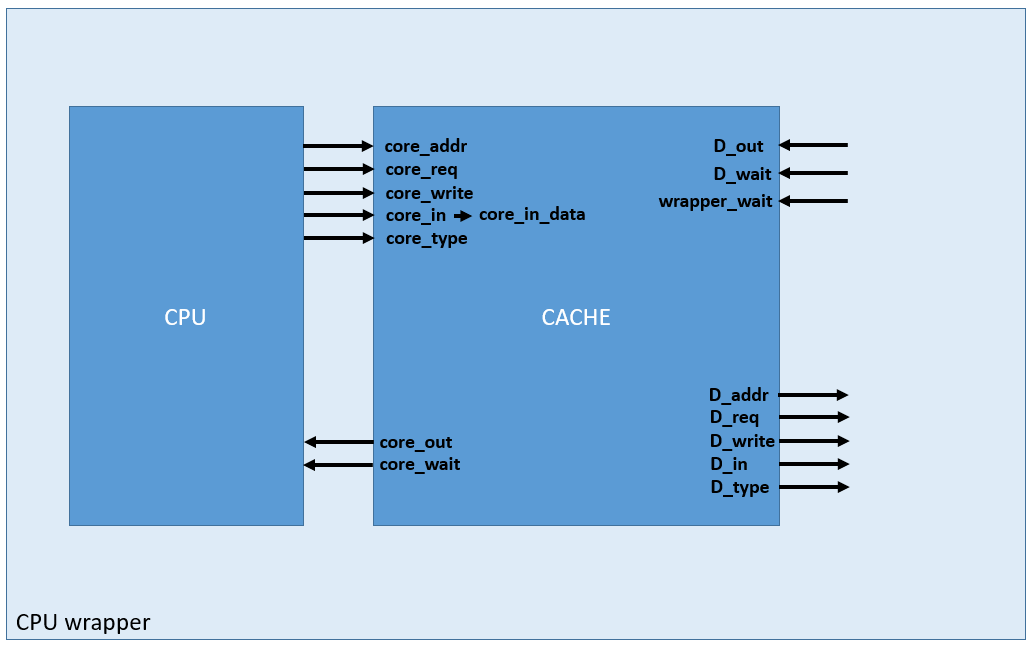
1. **Summary**

* 完成LIC\_data, LIC\_inst，且透過ABVIP驗證無cover與assert。
* 修改CPU\_wrapper, CPU, CPU\_control。
* 結合上述及前次作業之code，通過模擬prog0,1,2,，以及合成後模擬syn0,1,2,。
* 完成CPU在APR上之佈線。

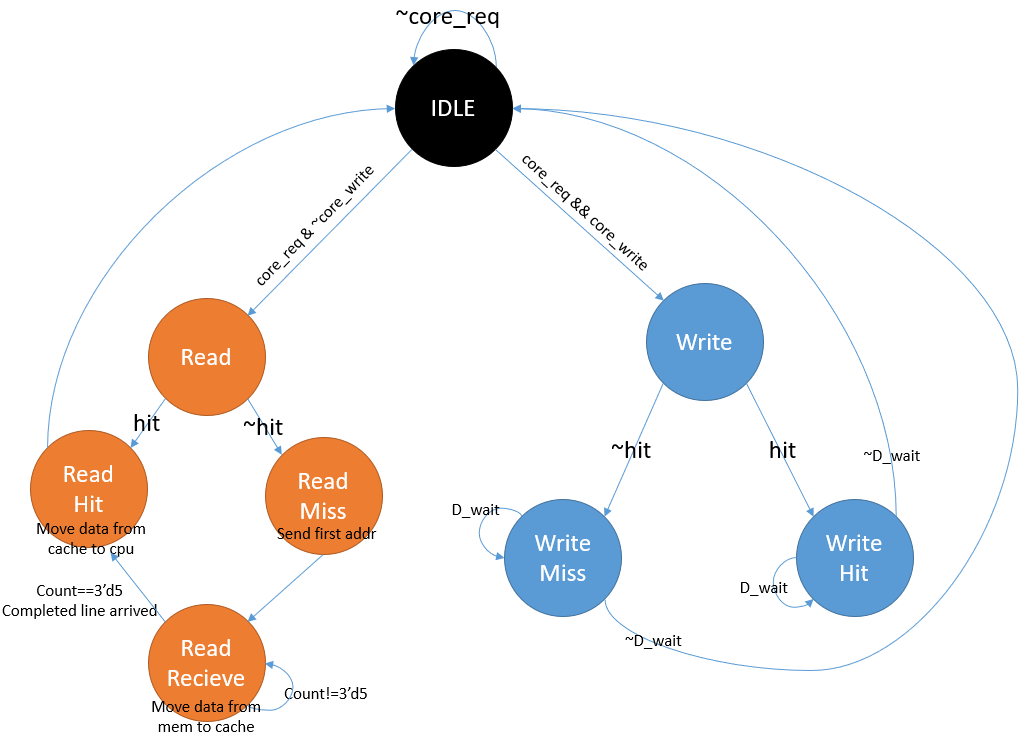
1. **Check list**

|  |  |  |  |
| --- | --- | --- | --- |
| RTL code | | synthesis | |
| prog0 | v | prog0 | v |
| prog1 | v | prog1 | v |
| prog2 | v | prog2 | v |
| prog3 | v | prog3 | v |
| superlint | 97.68% | | |
| APR | **v** | | |

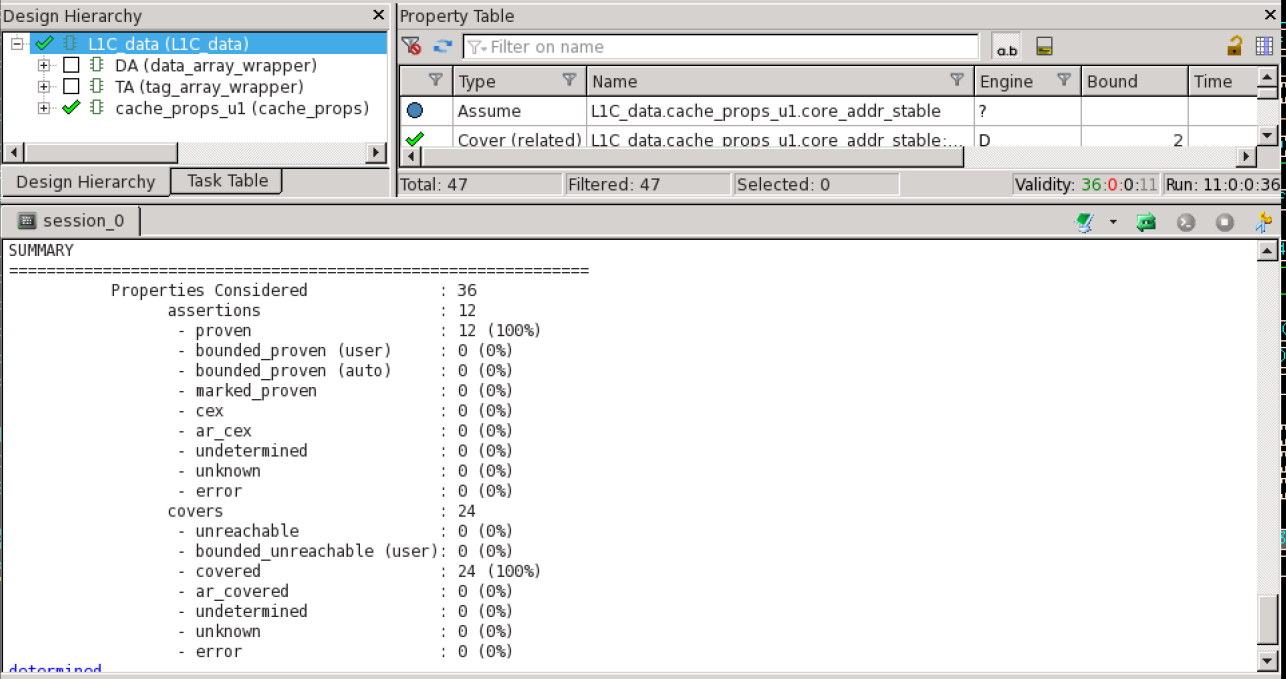
1. **Problem1**

implement a data cache module named “L1C\_data” and an instruction cache module named “L1C\_inst”.

本次作業要實做的是一個cache，目標是加速資料搬運的效率，在此以LIC\_data為例，因為LIC\_inst少了write的部分，所以可以視為是subset，在cache的邏輯中，每當CPU需要讀或寫資料，都要先到cache中檢查是否存在，在此我們用一個FSM來讓整個流程更加完整。

以read為例，當CPU發出core\_req討資料且write low的情況，會進到Read state，然後判斷有無hit，若hit cpu\_wrapper會先到cache看該資料存在與否，若存在則直接從cache讀出資料(Read\_hit)，若不存在(Read\_miss)，則發出第一個地址給AXI後，進入Read\_receive，繼續發出剩餘地址並把整條line的資料讀齊後，便可視為是資料存在，回到Read\_hit，完成此次讀值。

另一個部分是write，此部分是LIC\_data會有的狀況，當cpu發出core\_req及core\_write訊號時，首先會進到write階段，在此cache會先比較cpu給的addr以及cache本身有的addr，若有吻合者，進入Write\_hit，把cpu給的值寫入cache以及memory，並以D\_wait訊號當作完成與否的依據；反正addr不吻合者，進入Write\_miss，在此僅將資料寫入momory，跳過寫入cache。

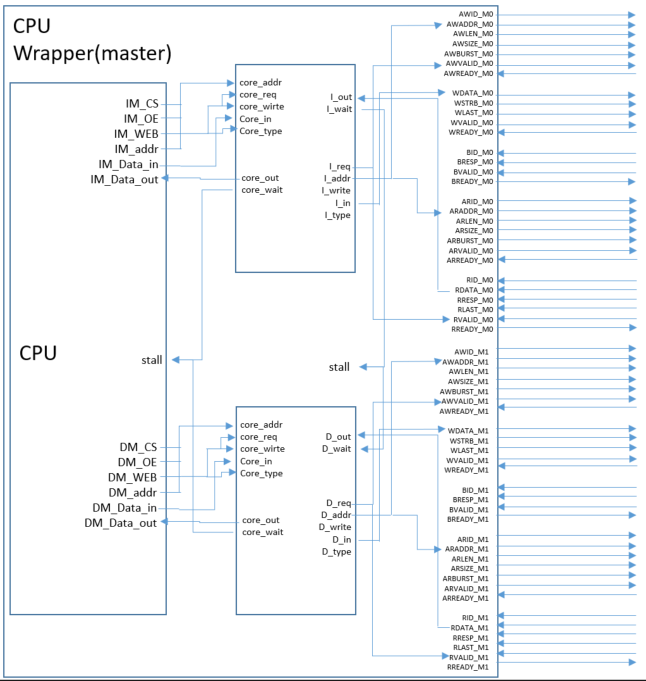
ABVIP - LIC\_data

0 cover 0 assert

ABVIP - LIC\_inst

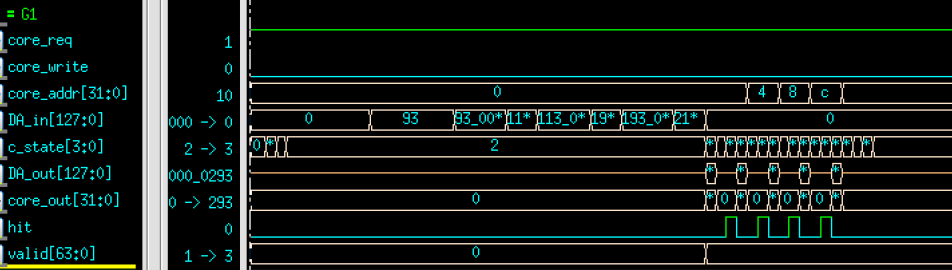
LIC\_inst和LIC\_data架構相同。

1. **Problem2**

****此外，本題也要將cache加入原有的系統中，所以要把CPU及AXI的腳會對應到cache定義的腳位，連結方式如block diagram所示，當CPU需要讀取或寫入資料時，會拉高core\_req以及core\_write(0為讀，1為寫)到cache，若read hit，則直接將cache中的資料讀入CPU，若read miss，cache會拉高I\_req 或 D\_ req訊號告訴CPU\_wrapper需要到AXI讀取資料，此時CPU也會進入stall狀態，等待資料成功寫入cache，當AXI搬運資料的過程，I\_wait或D\_wait訊號會拉高，write miss與write hit皆會拉高I\_req 或 D\_ req，將資料透過AXI寫到memory，差別在於write hit需要更新cache該位置的資料，而write miss則不用，下面會用波型介紹cache不同狀況以及介紹新增的指令。

Case : Read hit/miss

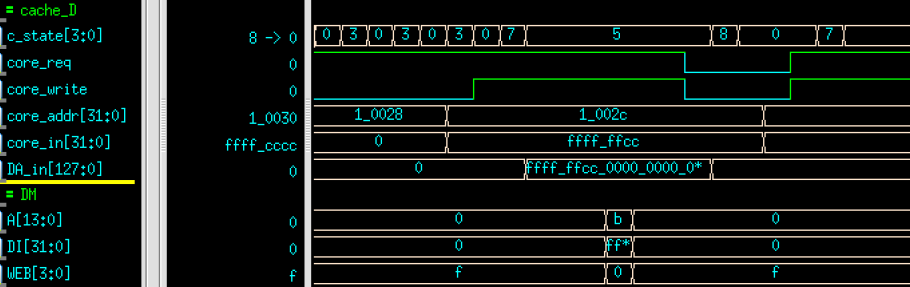
當CPU向cache索取address 0 的Instruction(core\_req=1且core\_write=0時)，但結果為read miss 因為該地址的valid為0，代表cache尚未儲存任何值，此時cache會向IM索取0,4,8,c之地址的資料，4個word共128bit，存入對應的cache記憶體中，因此，當後續要讀0,4,8,c的資料的時候，就會直接到hit狀態，加快資料的存取速度，而不需要每次都經過AXI到global memory搬取資料。



Case : Write hit/miss

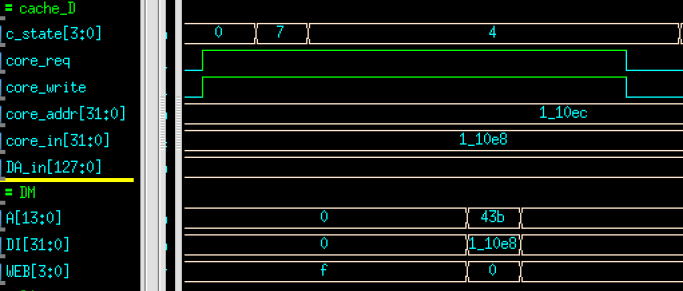
* Write hit:

當core\_req=1且core\_write=1時，表示CPU要向MEM寫入資料，若寫入資料之位置與cache有重複(write hit)，則會更新cache中的資料，並同時將該資料寫入DM，由下圖可看出，core\_addr為1\_002c之位置要寫入core\_in資料0xffff\_ffcc，會同時更新cache中的DA，下方的DM也會在同一個state中寫入同筆資料。



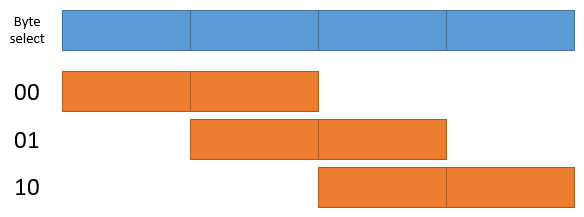
* Write miss:

當core\_req=1且core\_write=1時，表示CPU要向MEM寫入資料，若寫入資料之位置與cache沒有重複(write miss)，則會直接將值寫入 DM並不會改變cache中的儲存的值。如下圖所示，當write miss 並不會將值寫入cache，而是直接將值存入DM中。



* 新增指令

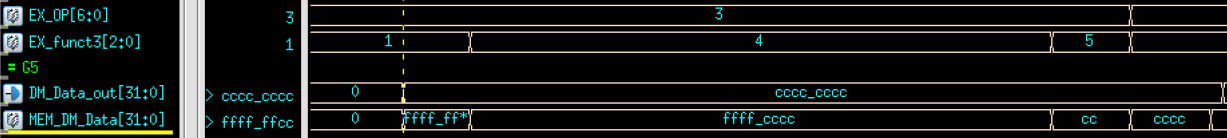
新增的指令中的SH、LH指令，LH、SH指令要讀寫兩個byte，而一個word是四個byte，所以總共有三種儲存位置的選擇，這個位置的選擇須要靠ALU\_result決定，所以要特別注意。



I-type

* LH, LBU, LHU

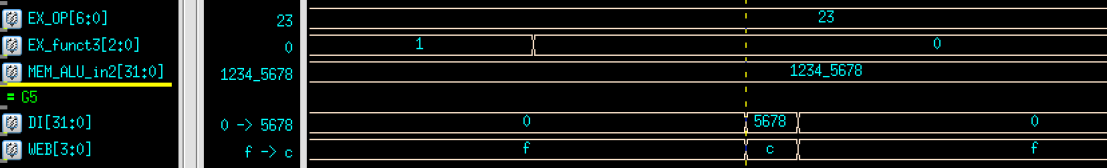
當op code 為0000011，funct3為001 時，指令為LH，因此由下圖可以看出，從mem 讀進來的資料為0xcccc\_cccc，經過處理後變成0xffff\_cccc，當funct3為100時，指令為LBU，0xcccc\_cccc，經過處理後變成0x0000\_00cc，當funct3為101時，指令為LHU，0xcccc\_cccc，經過處理後變成0x0000\_cccc。

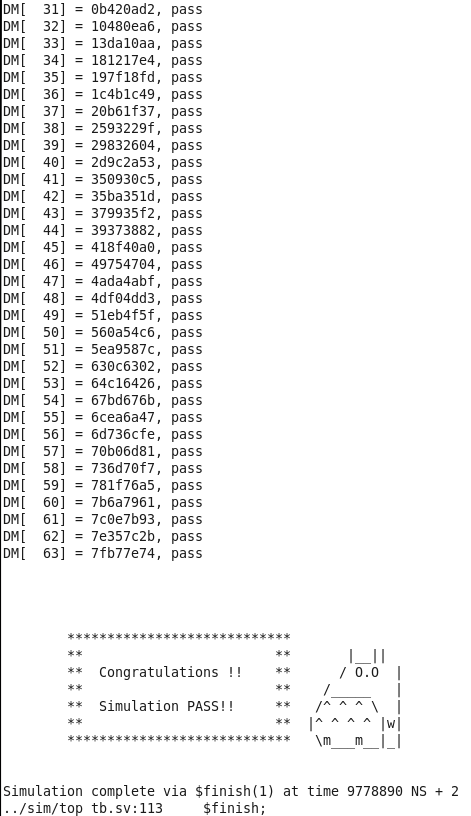


S-type

* SH

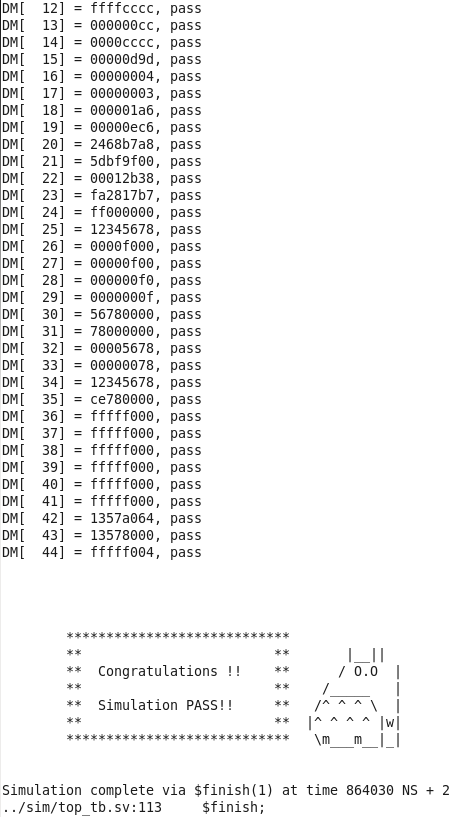
當op code 為0100011，funct3為001 時，指令為SH，由下圖可以看出，原來數值為0x1234\_5678，經處理變為0x0000\_5678存入MEM。

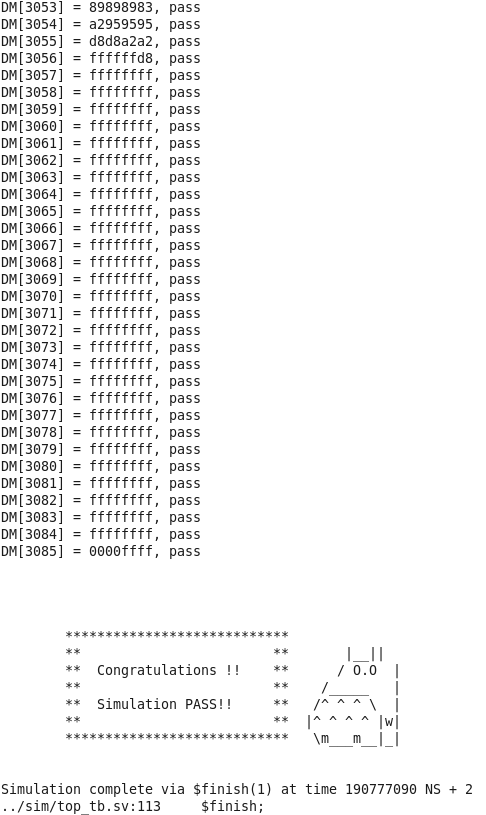
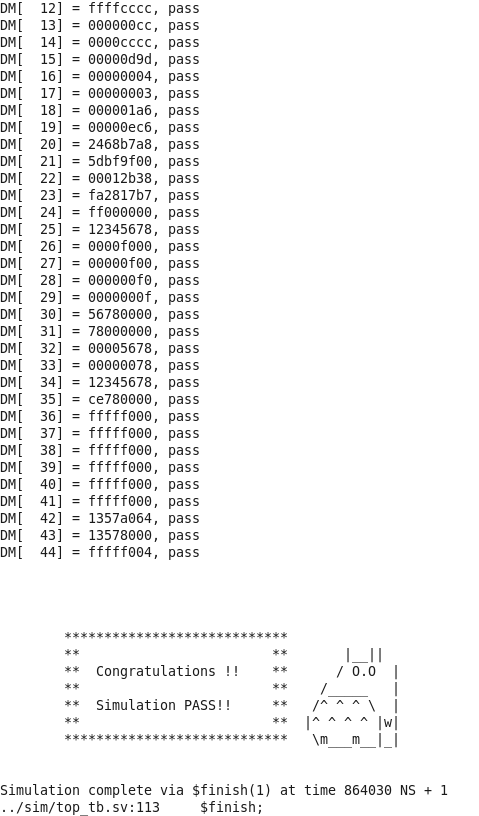


1. **Testbench**

**RTL1**

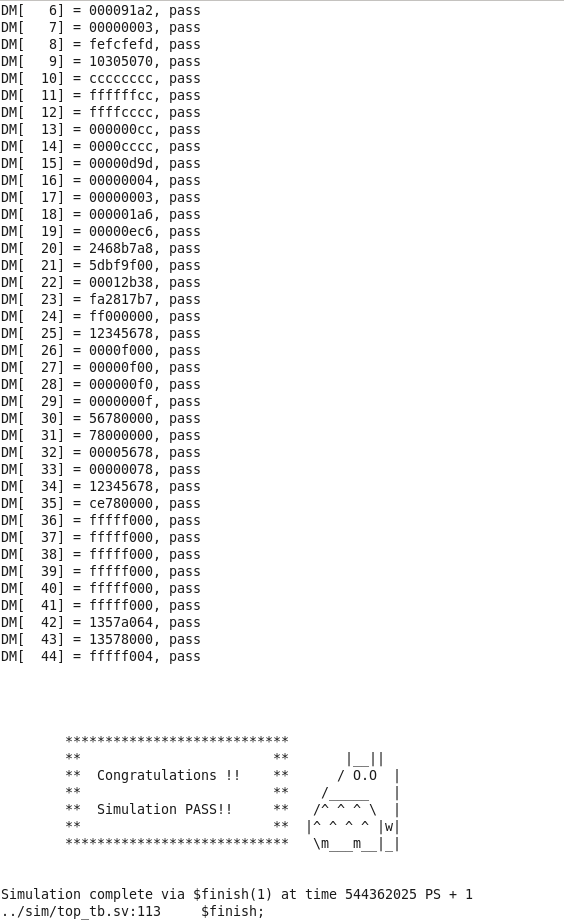
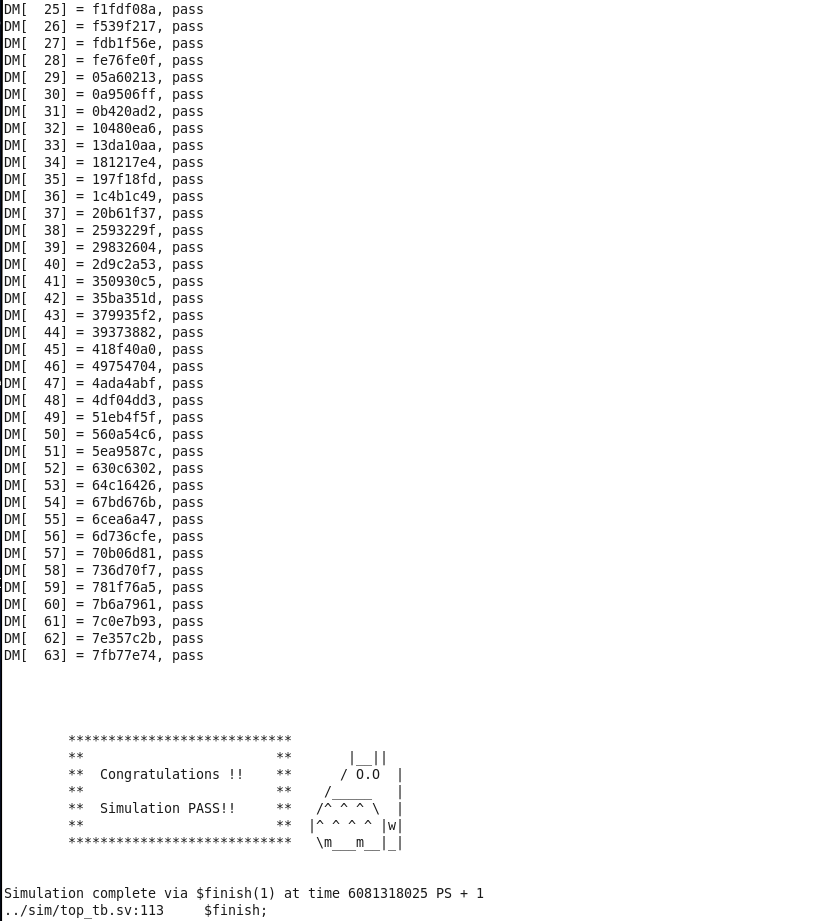
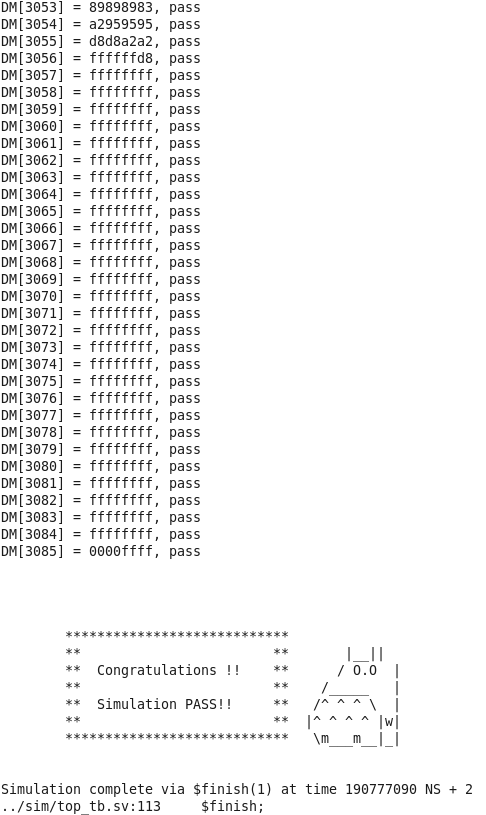
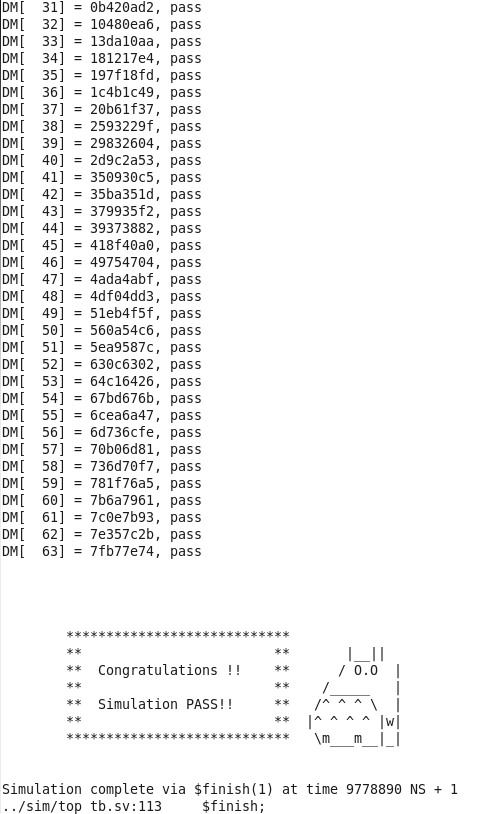
**RTL0**

****

****

**SYN0**

**RTL2**

****

**SYN2**

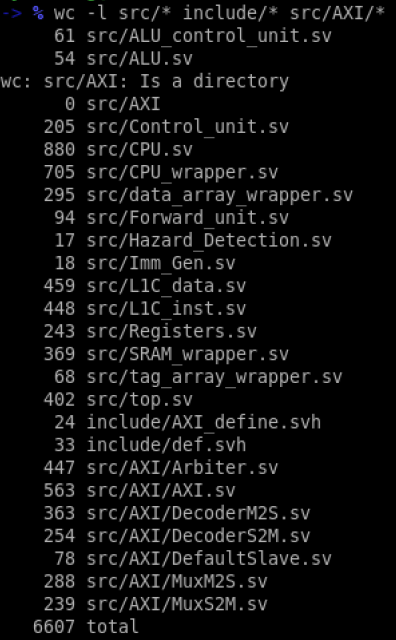
**SYN1**

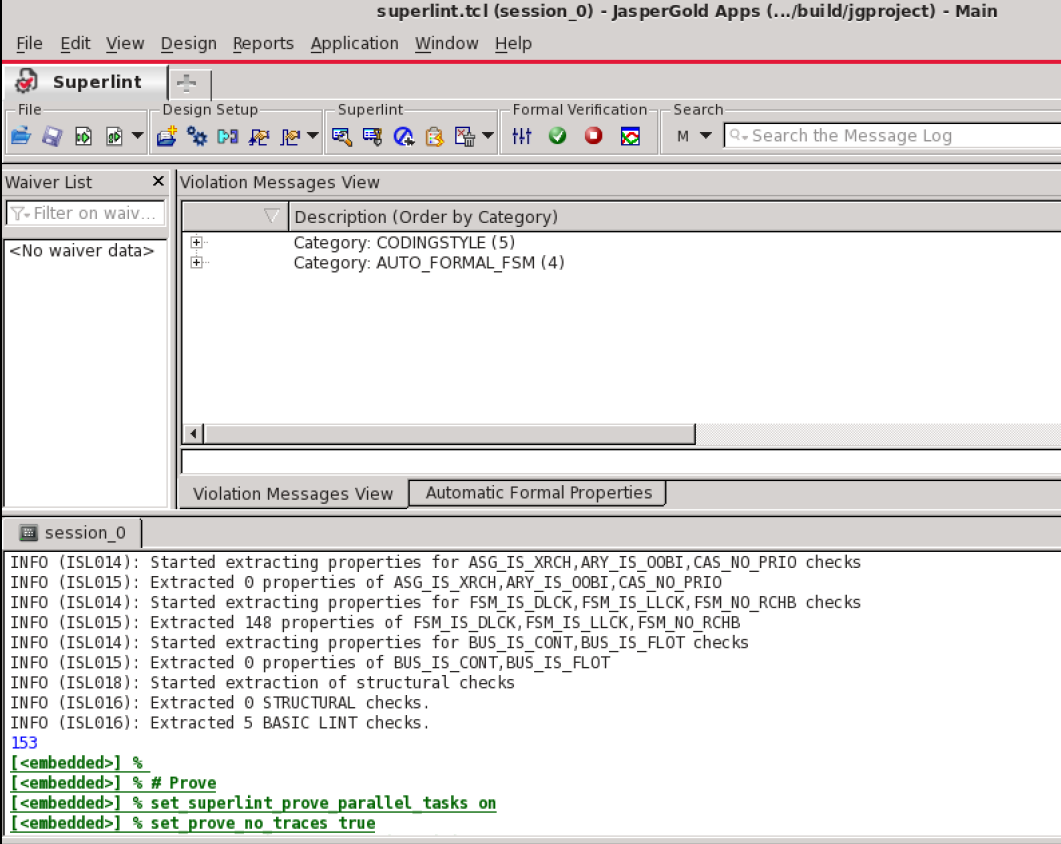
**PR1**

**PR0**

1. **Superlint**

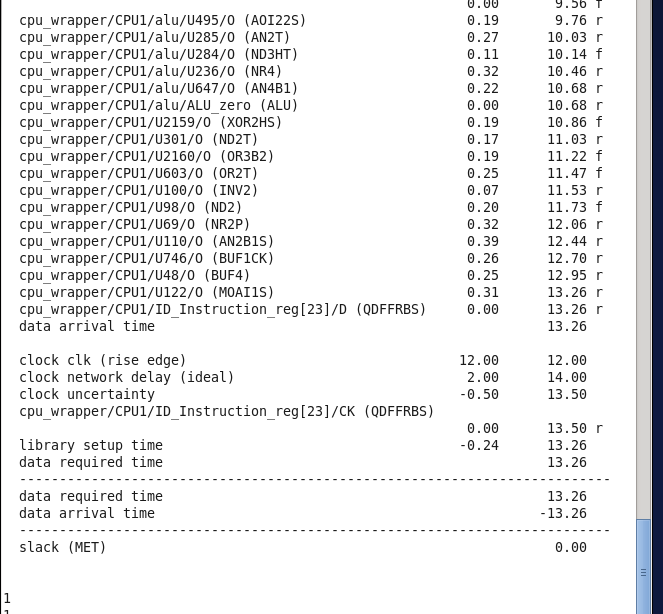
總行數 : 6657





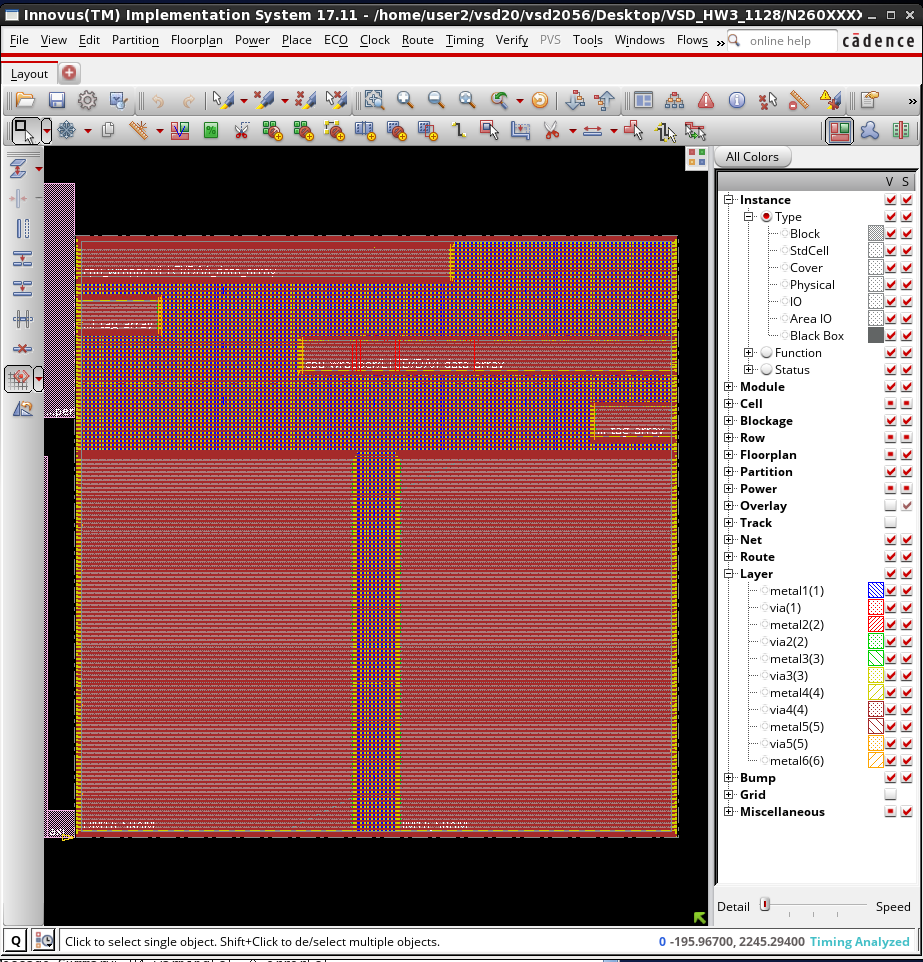
1-153/6657=97.68%

1. **Synthesis**

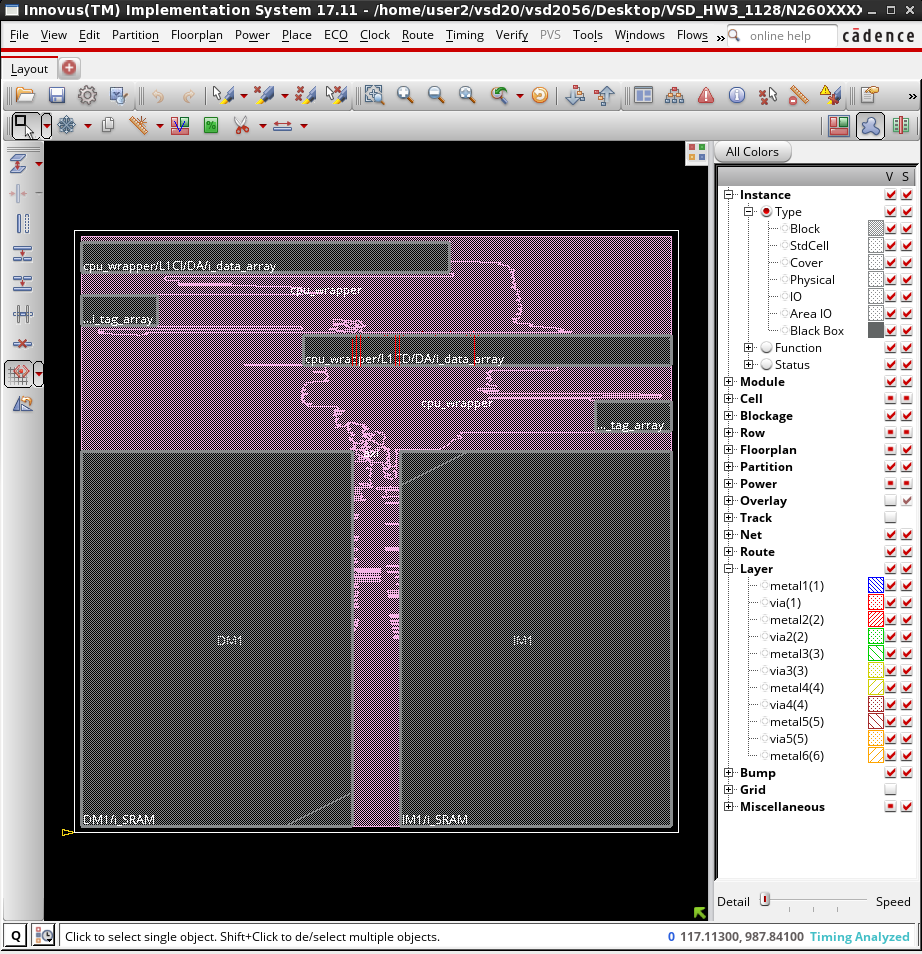
****CLOCK PERIOD = 12

1. **APR**

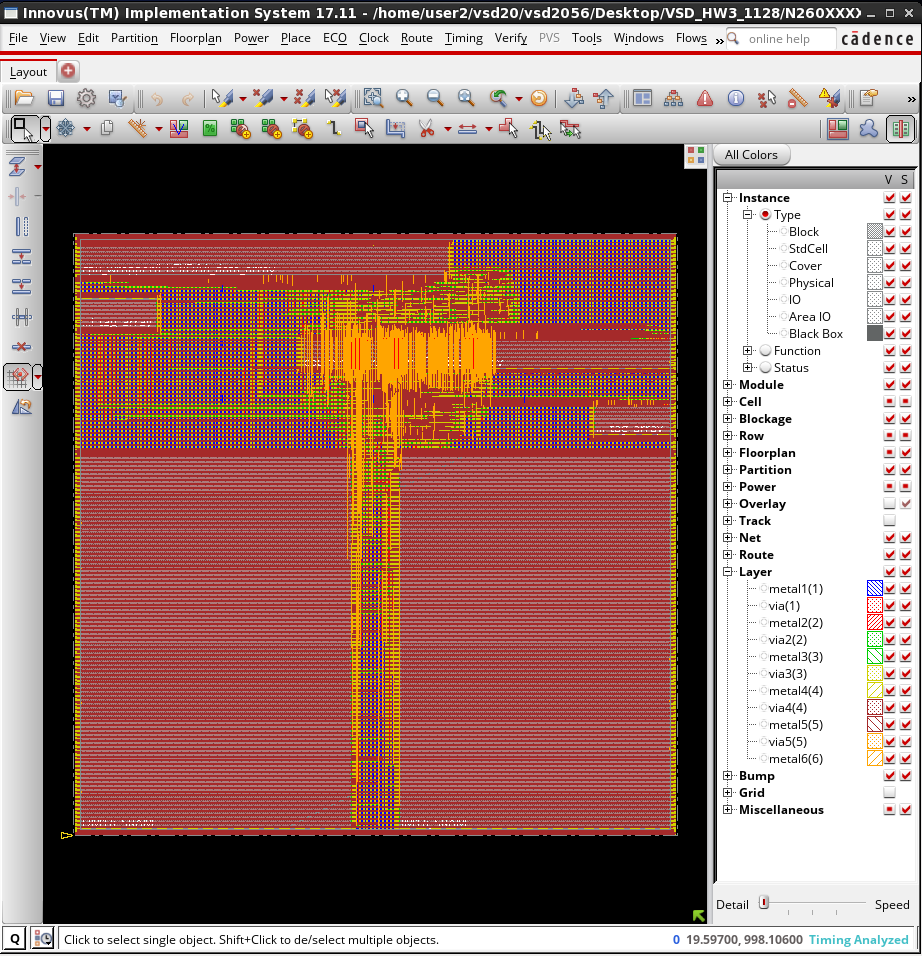
**Floorplan view**

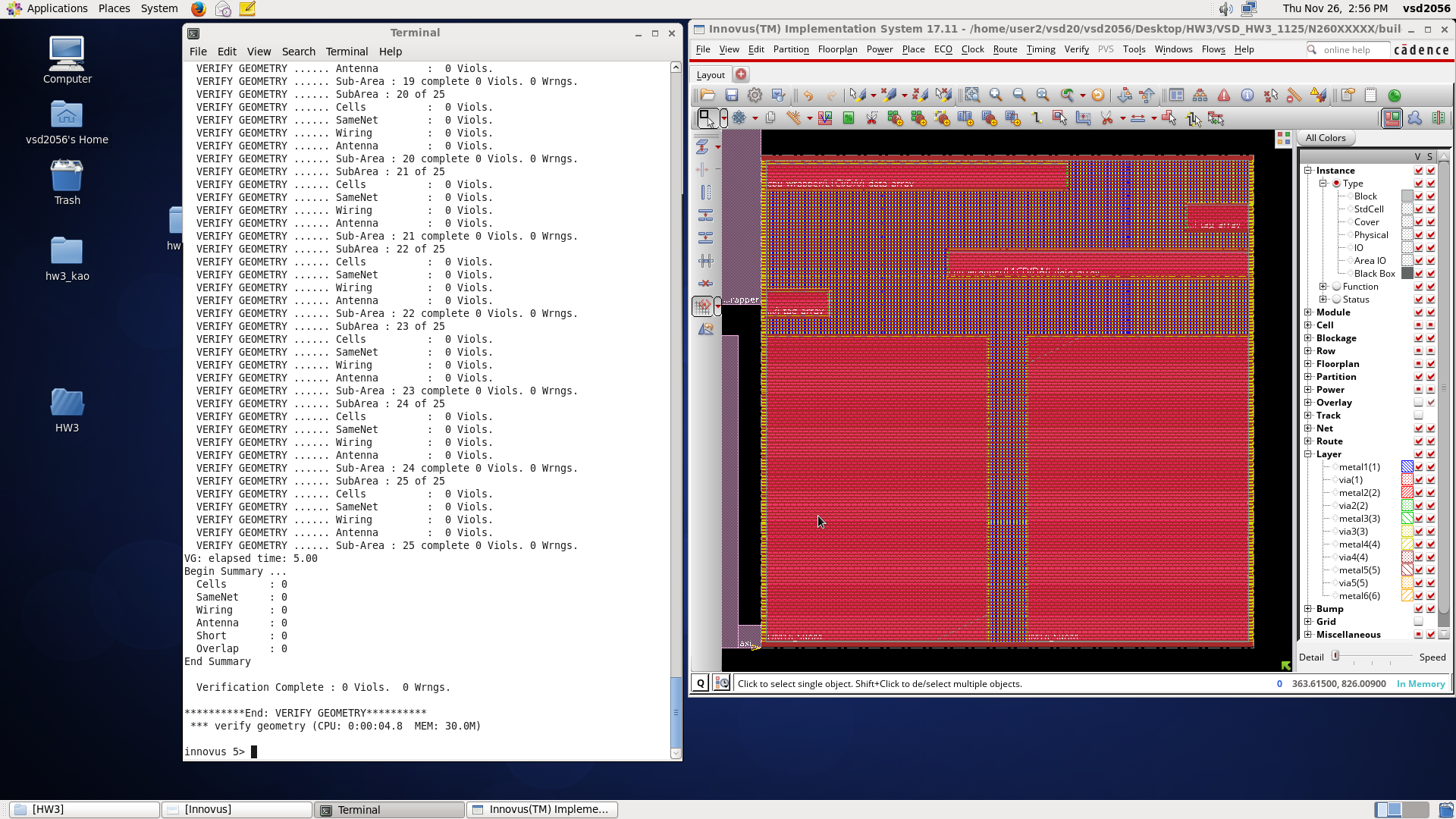


**Amoeba View**

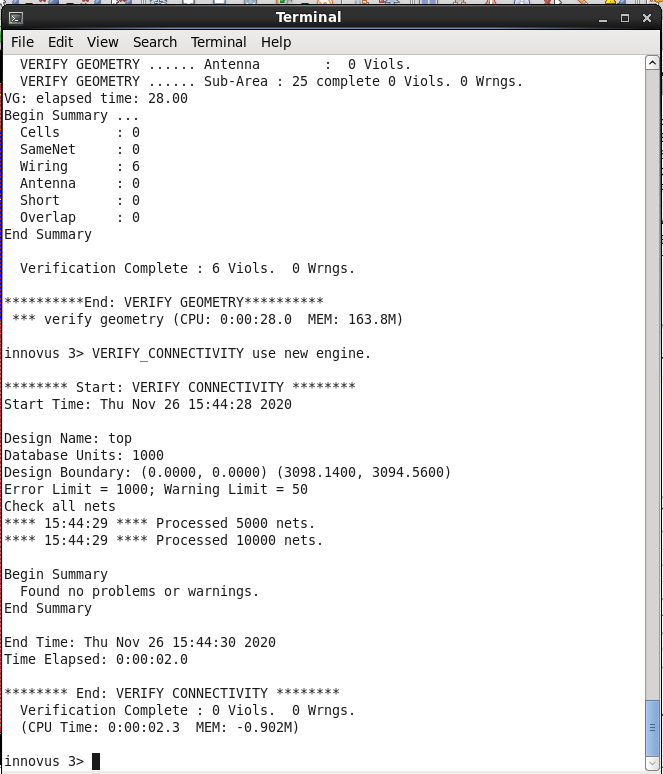


**Physical View**

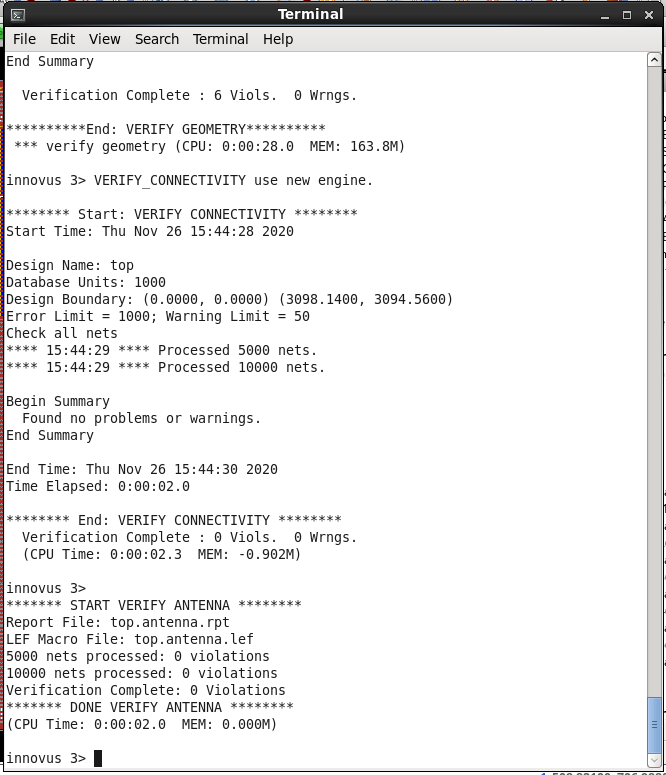


**Geometry Verification**

No violation

**Connectivity Verification**

No violation

**Antenna Verification**

No violation

**Describe the major problems you encountered and your resolutions:**

這次作業是要時做出一個cache，但是作業恰好在課堂教到cache之前就派下來了，所以在前期設計時碰到了一些阻礙，例如cache的state machine，一開始設計時在IDLE之後我會直接依據core\_write以及hit訊號判斷下個state是miss or hit，但後來上課後才聽到老師講解才發現老師的版本多了很多個state，但當初jasperGold有順利通過所以沒有想太多，一直到把cache接上cpu後才發現少了這個state會遇到一些時序上的問題，所以這算是前期困擾我們很大的一個問題，以及cache的I/O一開始放進CPU\_wrapper與CPU和AXI訊號線對接的時候，有時會有timing對不上的問題，因此有些地方會需要加buffer才能順利達到預期的行為。

**Lesson learned:**

這次作業除了實做出常常聽聞的cache架構，確切了解它能讓我們的cpu運作更加迅速的秘密，我們覺得APR這部分也很新奇，因為這是我們第一次透過自動化的tool把數位電路layout畫出來，和之前做類比電路的經驗不太一樣，雖然說有些不線上的問題無法找到教學順利解掉，但我們覺得至少我們對APR的流程及原理有更加了解了。