VLSI System Design (Graduate Level)

Fall 2021

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: \_\_\_\_\_陳韋綸\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Student ID: \_\_\_\_\_P76091412\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

* **Summary**

1. 完成cache的jaspergold驗證
2. 完成Booting Program、DRAM\_wrapper、ROM\_wrapper的設計
3. 完成Prog0、Prog1、Prog2、Prog3的RTL驗證
4. 完成Prog0、Prog1、Prog2、Prog3的Gate-Level驗證

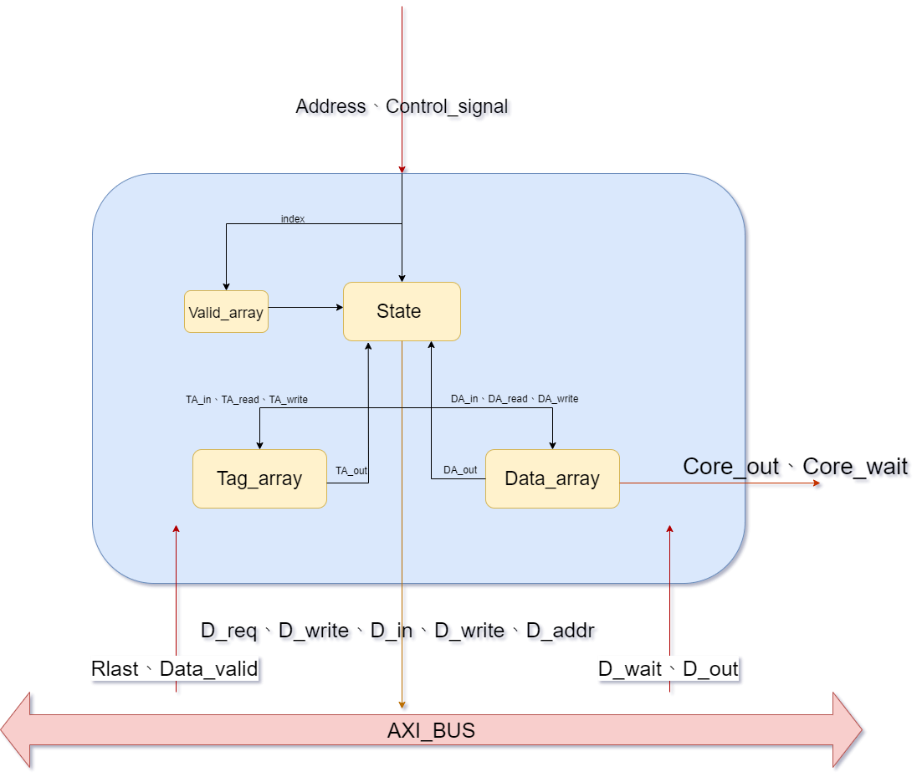
* **Problem1**

1. Cache Description

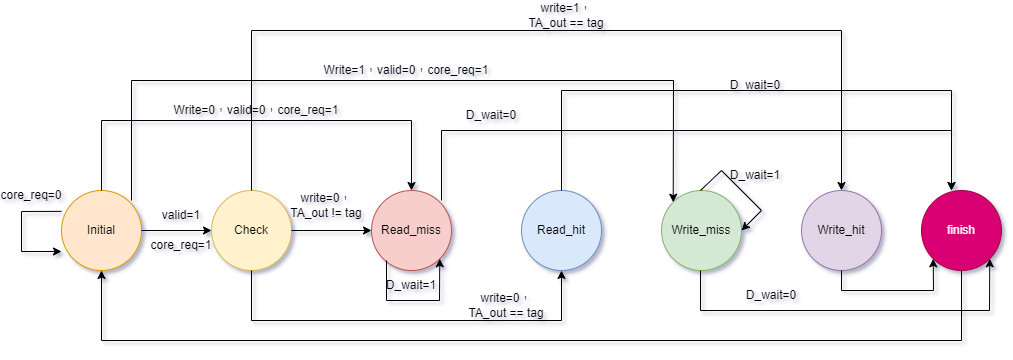
* 在我的Cache中一共有7個state，分別是initial\_state、check\_state、write\_miss\_state、write\_hit\_state、read\_miss\_state、read\_hit\_state、finish\_state，以下會分別介紹每個state的行為:

1. Initial\_state: 在這個state時如果沒有收到來自CPU的request則會一直維持在Initial\_state，一旦接收到來自CPU的request時就會先確認該address的index的valid bit是否為1，若為1則會進入到check\_state確認有沒有hit，若為0則會再依據core\_write判斷現在是write\_miss或是read\_miss，並進到對應的state。
2. Check\_state: 進入到check\_state時TA\_read會被設成1去把那個column所儲存的的tag值讀取出來，接著會馬上把TA\_out和該address的tag相比較，如果相等則會再跟去core\_write去判斷是write\_hit還是read\_hit，反之若不相等則會再用core\_write判斷是write\_miss或是read\_miss。
3. Write\_miss\_state: 因為發生cache write miss時採用的是write around的policy，所以在這個state時不會去更新cache裡面的值，只會把request、address、data、wrtie\_type交給AXI的Master，讓它去和slave溝通完成寫入的行為。
4. Write\_hit\_state:在發生cache write hit時所採用的是write through的policy，所以不但會更新slave的值，也會同步更新cache裡面的值，因此在這個state，cache會發一個request給AXI的Matster並把address、write\_type和data一併傳送過去，而此時cache會跟據CPU傳進來的core\_type把對應的DA\_write設定好，並把core\_in的data放在DA\_in的對應位置讓cache寫入
5. Read\_miss\_state: 發生read\_miss時因為cache的data欄位有4個word的大小，所以每次發生read\_miss時都要像AXI的Master發request並從slave搬4個word的資料量回來，因此要利用一個counter去計算現在讀到第幾個word並把D\_out照順序放在對應的DA\_in，讓cache可以正確的寫入。
6. Read\_hit\_state: 如果發生Read\_hit，則會直接把data\_array的值讀出來並且根據core\_addr[3:2]的block offset去判斷所需要的資料放在DA\_out的哪一個位置。
7. Finish\_state:最後這個state會把從cache讀出來的資料傳遞給core\_out送回去給CPU，如此一來便完成一次的data transfer。
8. Cache Block overview

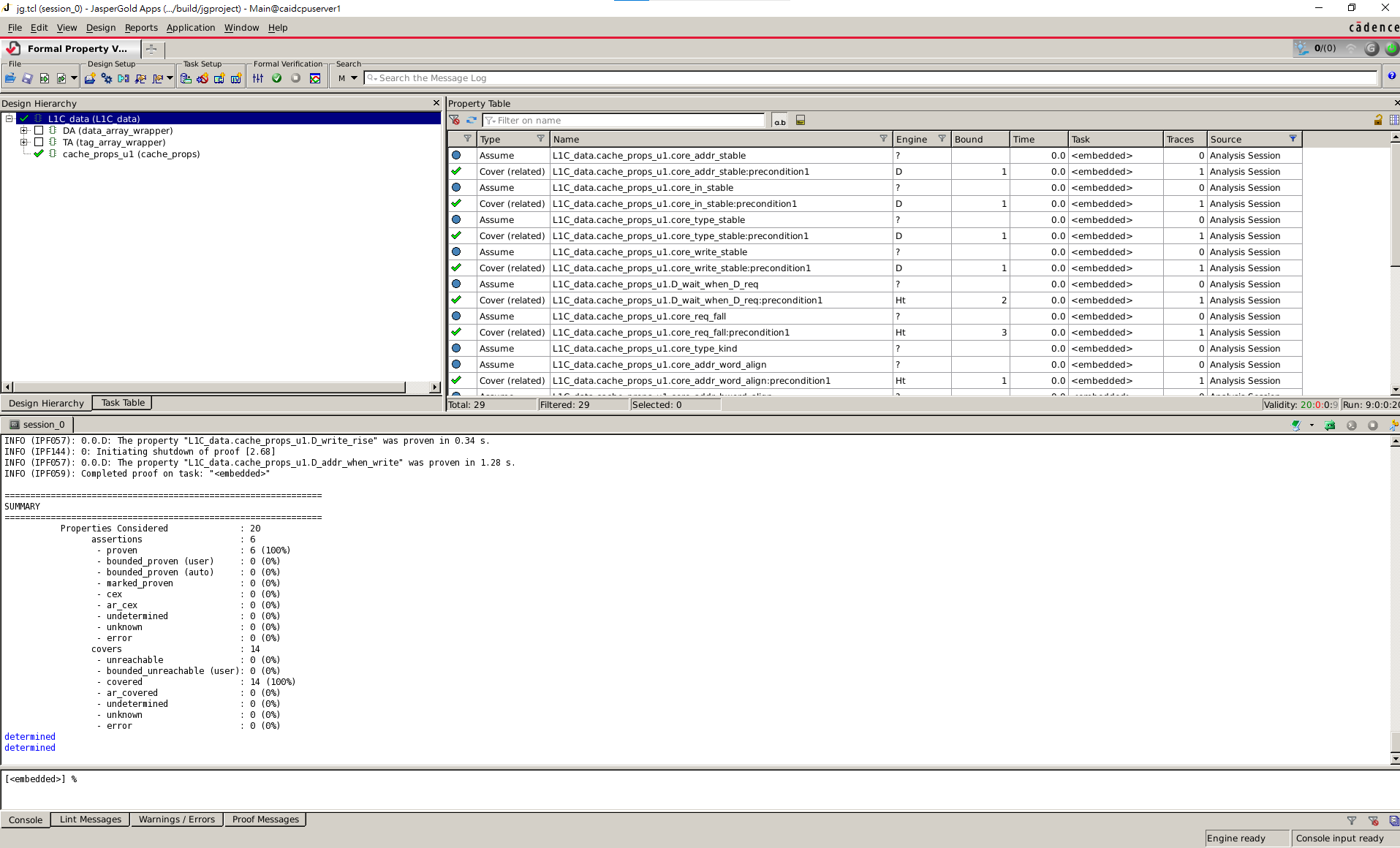
* Cache Block diagram



* FSM of Cache



1. Cache Jaspergold Verification



1. Hit Rate

一張含有 文字 的圖片

自動產生的描述

1. Data\_Cache:

Data\_Cache hit = 4411

Data\_Cache miss = 2928

Data\_Cache hit rate = (4411/7339)\*100% = 60.1%

1. Instr\_Cache:

Instr\_Cache hit = 29176

Instr\_Cache miss = 205

Instr\_Cache hit rate = (29176/29381)\*100%=99.3%

1. Instruction per cycle of CPU



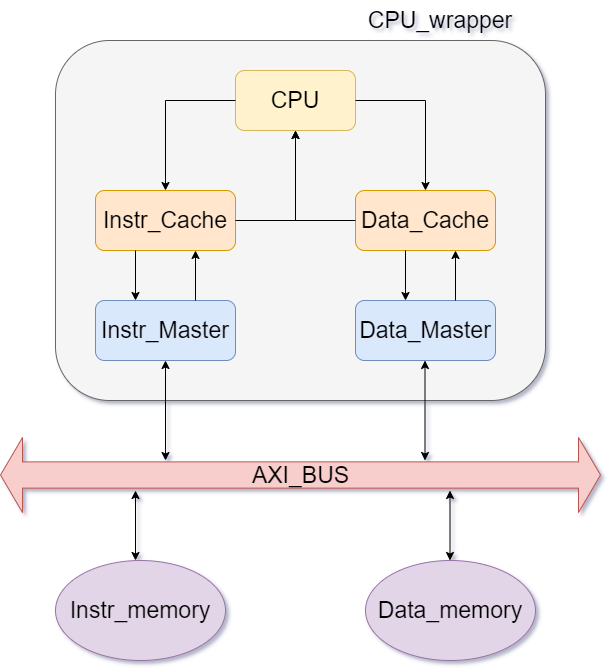


* 1. 在Prog0我的clock cycle是11ns，總執行時間為1560053ns，所以總共的cycle數為1560053/11 = 141823個cycle
  2. 又從上圖得知總共執行的指令為23374個
  3. 因此IPC = 23374/141823 = 0.165
* **Problem2**

1. Design Description

* 因為主要的CPU和AXI BUS的部分都與作業2相差不大，所以在此只針對DRAM\_wrapper和ROM\_wrapper設計說明。
* ROM的行為與SRAM相同，只不過ROM只會進行read的動作，不會有write發生，因此只需要把SRAM\_wrapper裡的Write channel、B channel、AW\_channel拿掉即可，另外在address上因為cache一旦發生Read miss都要連續讀4筆data出來，所以每次handshake完之後都要把address+1，直到rlast訊號拉起來為止。
* 在DRAM\_wrapper方面，我一共分了5個state，IDLE、Action\_row、Read\_col、Write\_col、Precharge

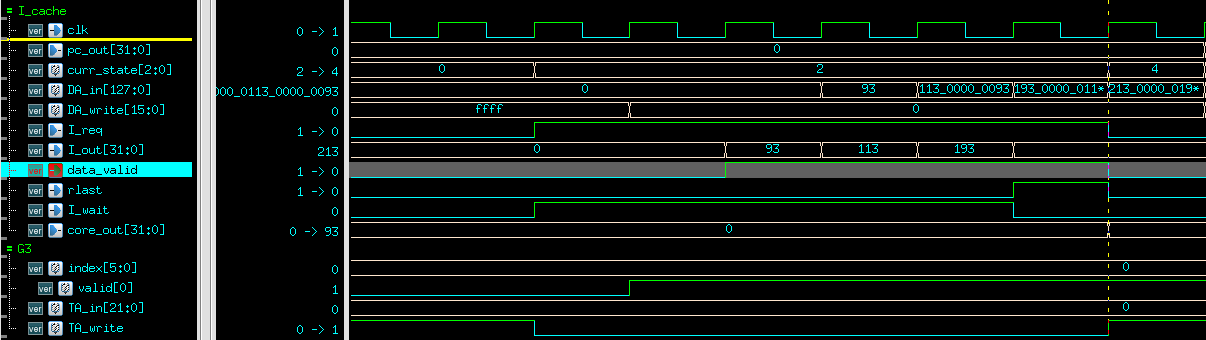
1. IDLE:在沒有收到arvalid或awvalid之前都會一直在這個state，一旦收到這2個訊號之後就會先把address存起來，並且進到下一個state。
2. Action\_row: 因為tRCD的緣故，所以RASn= 1’b0的信號要和CASn = 1’b0相差5個clock以上，因此我會用一個counter數在目前的state執行了幾個cycle，並且會在counter=0的時候就把RASn設成0，之後就會一直維持high，最後在counter=5的時候會再根據剛剛是傳awvalid還是arvalid進來判斷等等是要進行Read或是Write
3. Read\_col:在Read\_col state時在counter=0的時候就會先把CASn設成0並把WEn 設成4’hf，等待4個cycle後DRAM\_valid就會傳進來告訴wrapper 資料已經讀出來了，接著會去檢查rlast是否為1，如果不為1會把DRAM\_addr + 1 並且跟上一次DRAM address的Row\_addr比對檢查是否需要precharge，若Row\_addr相同則先不用進行precharge，只需再把CASn設成0即可，以此類推直到讀出4個值為止
4. Write\_col: 在Write\_col時一樣會在counter = 0的時候把CASn設成0，並且把WEn設成4’h0，去進行寫入，之後會檢查wlast是否為1，確認把Master所需要寫的資料量都寫入後就會結束這個state
5. Precharge: 在這個state時會把先前在arvalid或是awvalid為1時傳入的所存下的ROW address寫回去DRAM裡面，而且會把RASn設為0，CASn設為1，WEn設為4’b0，去進行Precharge的行為，且因為tRP的緣故，每次precharge和下一次RASn的assert都要間隔5個clock以上。
6. Block overview



1. Waveform of Instruction

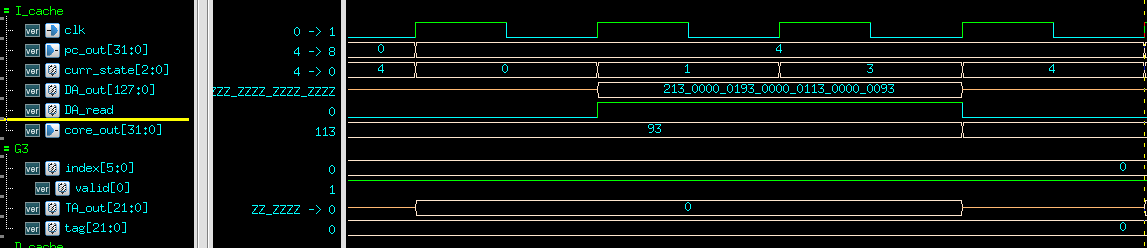
* 因為這次的作業最大的不同點在於多了Cache和DRAM這2個module，其餘的設計以及ROM\_wrapper的波型與前次作業沒有太大的差別，所以只會針對cache 發生Read miss、Read hit、Write miss、Write hit時的waveform，與DRAM如何read、write資料的waveform進行說明

1. Read miss:



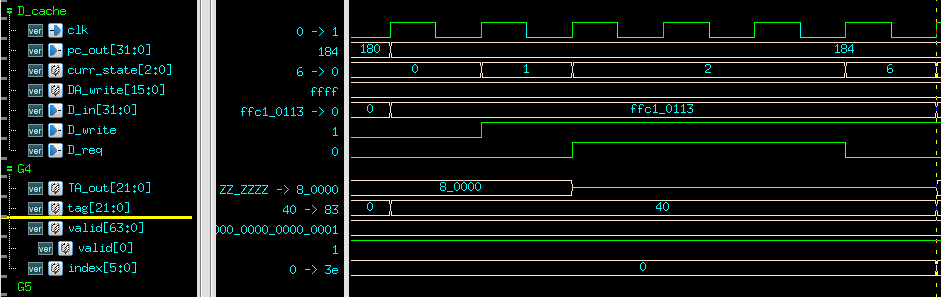
當program counter為0的時候，因為這是第一次對Instruction cache做request，因此這個address所對應到的欄位的valid bit一定是0，必定是miss，因此在進入Read miss state後就會把valid bit 設成1，並且把tag寫入該欄位，然後會向AXI的Master發出I\_req，接著slave就會連續回傳4個值，並且每一筆資料讀出來時都會傳一個data\_valid的訊號告訴cache，cache再照順序將data擺放在DA\_in裡面，之後再把4個word的資料都寫入data\_array中，此外當slave的rlast訊號為1的時候就代表資料都送完了，就會進到最後一個finish\_state裡，再將core\_out依照他的core\_addr從DA\_in輸出他所需要的data。

1. Read hit:



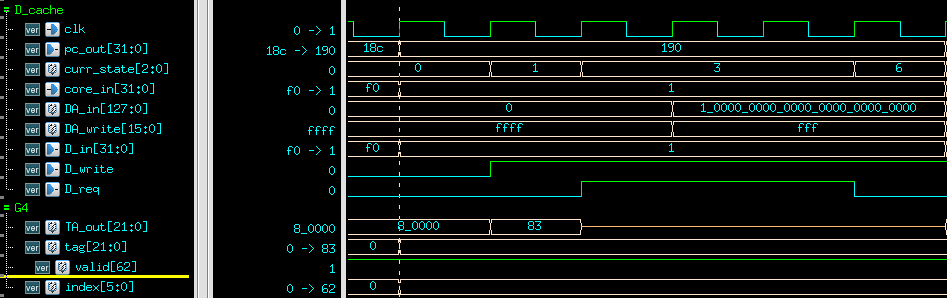
當pc\_out=4時，因為該address所對應的valid bit為1，所以再結束initial\_state之後會先進到check state檢查TA\_out與address的tag是否相同，因為TA\_out與tag相同且write又是0，所以會進到Read\_hit\_state，接著在Read hit state中會把DA\_read設成1去讀該index下的data\_array，然後cache再根據core\_addr[3:2]的block offset去判斷他要的資料是在DA\_out的哪一個部分，最後到finish\_state後再把data輸出給core\_out。

1. Write miss:



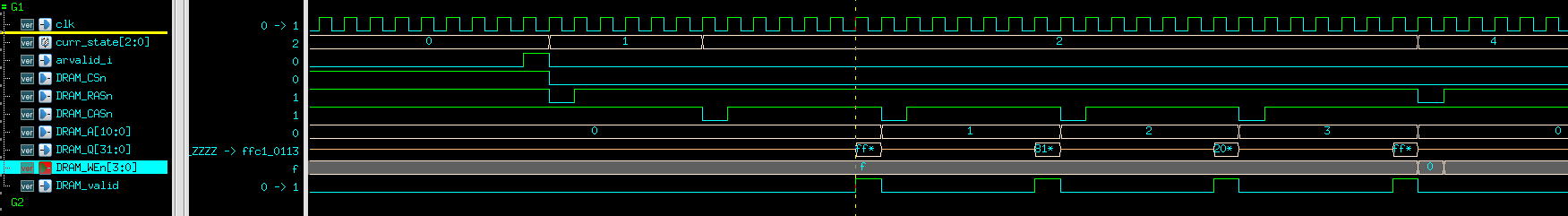
當pc\_out=184時代表MEM stage在執行sw a4,0(a5)的指令，因此，此時會把core\_req送到Data\_cache中，又根據該address的index發現他對應的valid bit為1，所以接著會進入check state檢查該address的tag和TA\_out是否相同，但發現2者並不相同此外D\_write也為1，所以接著會進到write miss state，因為我們在cache write miss所採取的policy為write around，因此不會去更新data\_array裡面的值，所以DA\_write都是4’hffff，只會更新slave裡面的data，於是D\_cache會向AXI的Master發出request並告訴它要write into slave，同時也把D\_in設成CPU所傳入data，然後傳送給Master去做寫入，寫完之後再進到finish\_state，就完成整個write miss的處理。

1. Write hit:



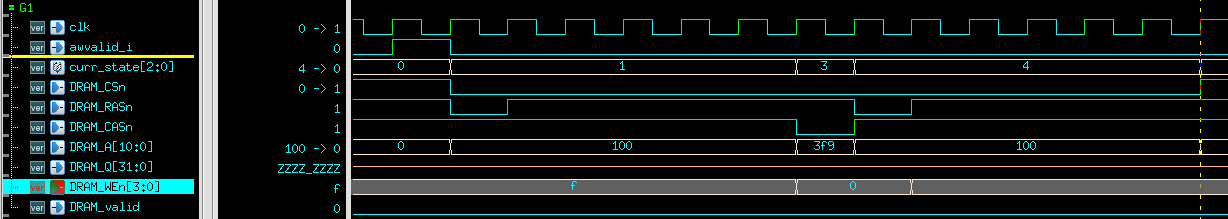
當pc\_out為190的時候，代表MEM stage 正在處理sw a5,-20(s0)，此時Data\_cache一樣會先去檢查該address index所對應的valid bit是否為1，因為它的valid bit為1，所以接著會進到check state，檢查TA\_out與tag是否相同，這時候他們2者也相同，所以接著會再進到write hit state，在發生cache write hit的時候所採用的是write through policy，所以不但會更新data array裡面的值，同時也會更新slave裡面的值，因此在write hit state時，會根據core\_addr[3:2]的block offset把core\_in存入DA\_in中正確的位置，DA\_write則會根據core\_addr[1:0]的byte offset以及core\_write設定要存入該block中哪一些byte，同時，也會向AXI的Master發送一個write request並把data、address和D\_type一併送入，等到Master寫完收到Bresp之後Cache就會進到finish state，代表完成了write hit 的處理。

1. DRAM Read:



當DRAM收到來自Master的arvalid之後就會進入Action\_row的state，g此時會在counter=0的時候就把RASn設成0，並且row\_address為0，接著過了5個clock之後就會進到read\_col state，此時一樣會在counter=0得時候把CASn設成0，此時的column address為0，在過了4個cycle之後DRAM\_valid被拉起來了，代表data讀出來了，但是因為此時還沒有讀到4個word的資料量因此又再進行了一次read\_col的行為，只不過現在的column address變成1了，依此類推直到第4次的DRAM\_valid被拉起來為止，此時4個word的資料量都被讀出來了，接著就會進到precharge state，在這個state因為此次的Active row address為0所以我會把Wen設成4’h0，RASn也設成0，而DRAM\_A則是這次Transfer的Active row address同樣也是0，之後等到5個cycle過了之後就會回到IDLE狀態，這次的Read operation也就完成了。

1. DRAM\_write:



當Master發送awvalid訊號給DRAM\_wrapper時就代表要對DRAM做寫入的動作，此時DRAM\_wrapper會進入Action\_row state，把RASn設成0，row\_address為100，接著過了5個cycle之後就會進入write\_col state，在這個state時會把CASn設成0，WEn也設成4’h0，此時的DRAM\_A為colum address是3f9，最後再寫完data之後就會進到precharge state，在這個state裡，RASn會再次被設成0，但是WEn會被設成4’h0，而此時的DRAM\_A是這次Transfer的Activation ROW，因此是100，之後在過了5個cycle之後就會回到IDLE狀態了，也代表這次的write operation完成了。

1. Prog1、Prog2、Prog3 Explanation
2. Prog1

一張含有 文字 的圖片

自動產生的描述

首先將array\_addr和\_test\_start都宣告為short型態，如此一來在對array\_addr或是\_test\_start做讀取或是寫入就都會以Half-word的形式去操作，之後再利用bitwise的方式做bubble sort，最後都排序完成時就會以ascending的順序輸出，並且依序寫入到\_test\_start的array中。

1. Prog2

* 在program2中因為要將原本RGB的圖片轉成灰階，所以要對圖片中的每一個pixel的RGB channel都乘上一個小數，由於在實作CPU的時候並沒有做乘法指令以及浮點數的運算，因此在小數部分我會採用近似的方式去表達，而乘法的部分則會利用加法來取代。



* 在表示0.11\*b上，因為0.11\*b會近似於(115344)/(115344)\*b，此算式又可以表達成((115344)\*b) >> 20 = (1110000101001000\*b) >> 20，而1110000101001000\*b可以利用左移以及相加得到，所以1110000101001000\*b可以表示成(b<<3 + b<<6 + b<<8 + b<<13 + b<<14 + b<<15)，如此一來就得到0.11\*b的結果。

一張含有 文字 的圖片

自動產生的描述

* 在表示0.59\*g上，因為0.59\*g會近似於(618660)/(115344)\*g，此算式又可以表達成((618660)\*g) >> 20 = (10010111000010100100\*g) >> 20，而10010111000010100100\*g可以利用左移以及相加得到，所以10010111000010100100\*g可以表示成(g<<2 +g<<5 + g<<7 + g<<12 + g<<13 + g<<14 + g<<16 + g<<19)，如此一來就得到0.59\*g的結果。

一張含有 文字 的圖片

自動產生的描述

* 在表示0.3\*r上，因為0.3\*r會近似於(314573)/(115344)\*r，此算式又可以表達成((314573)\*r) >> 20 = (1001100110011001101\*r) >> 20，而1001100110011001101\*r可以利用左移以及相加得到，所以1001100110011001101\*r可以表示成(r + r<<2 +r<<3 + r<<6 + r<<7 + r<<10 + r<<11 + r<<14 + r<<15 + r<<18)，如此一來就得到0.3\*r的結果。

一張含有 文字 的圖片

自動產生的描述

* 而在main部分中的第16行的add t0,t0,t1會去讀取整個BMP image的大小，這也是我們整個資料結尾的所在，而在第19行的lhu t1,10(s1)則是去讀取offset，這個offset是代表BMP image的RGB資料落在哪一個位置，但布包括header。

一張含有 文字 的圖片

自動產生的描述

* 在write\_header的section主要是將BMP的header寫入到\_test\_start中

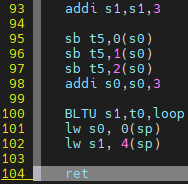
一張含有 文字 的圖片

自動產生的描述

* 在將header寫入到\_test\_start裡面後就會開始進行灰階化的處理，首先會先把RGB的資料load進來，之後再透過前面所提過的利用近似和邏輯左移以及加法的方式去完成小數的乘法

一張含有 文字 的圖片

自動產生的描述



* 最後當整個pixel的灰階化都處理完成後就會將運算完的結果寫入到\_test\_start裡面，整個程式便執行結束。

1. Prog3

一張含有 文字 的圖片

自動產生的描述

* 在prog3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。

1. 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，

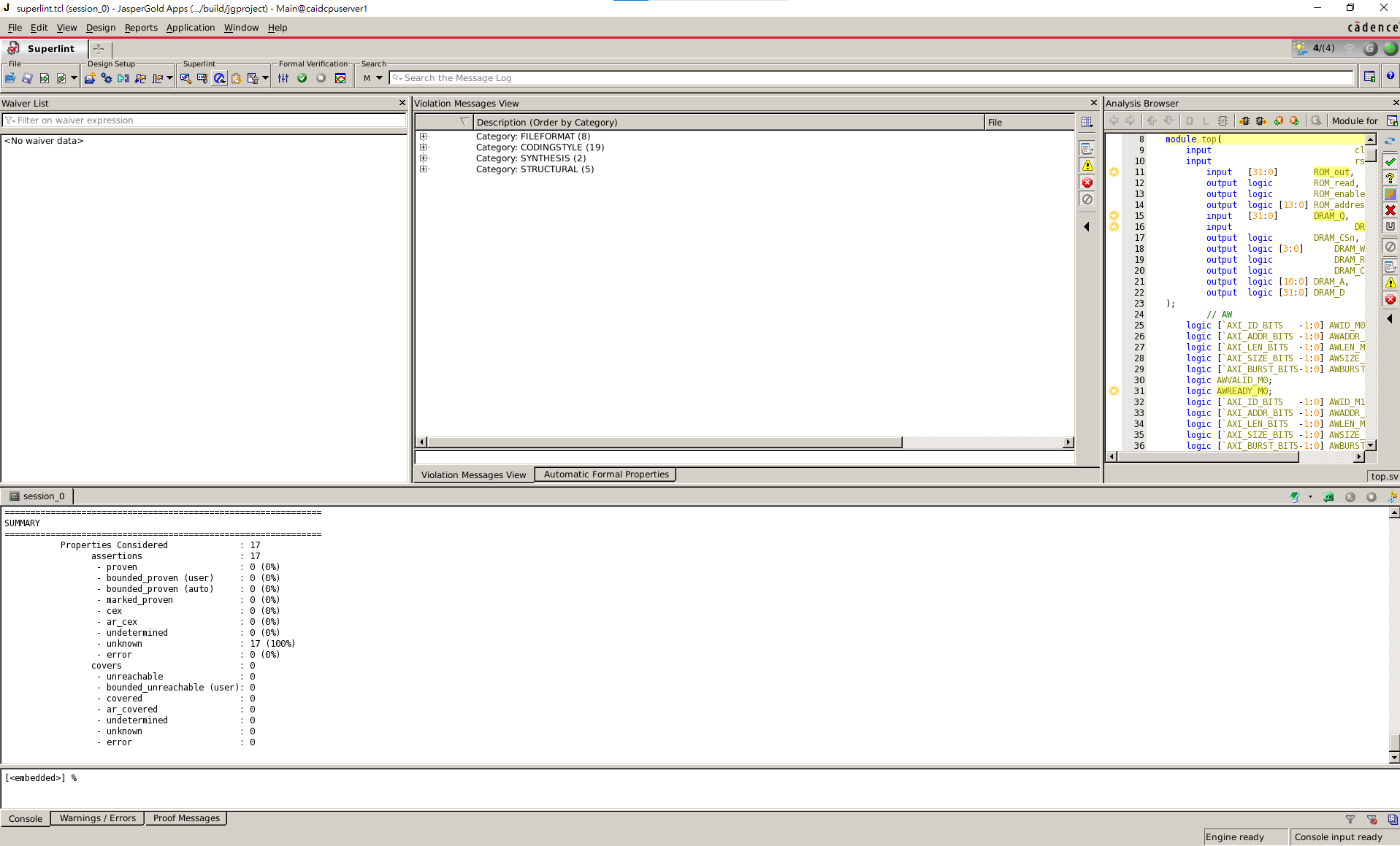
* \_imem\_start: 首先會計算DRAM當中instruction start的起始位置和instruction end的結束位置，把2者相減就可以得知DRAM當中存放幾條指令，接著再用一個for迴圈從\_dram\_i\_start的起始位置開始依序把instruction搬到\_imem\_start裡面對應的index位址。
* \_\_data\_start: 同上，將\_data\_start和\_data\_end相減就可以得知Main data當中要存放多少的data數量，接著再利用一個for迴圈從\_data\_paddr\_start的開始位置依序把\_\_data\_paddr\_start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。

1. Number of Lines of RTL code and Superlint result
2. Number of Line of RTL code = 6255

一張含有 文字 的圖片

自動產生的描述

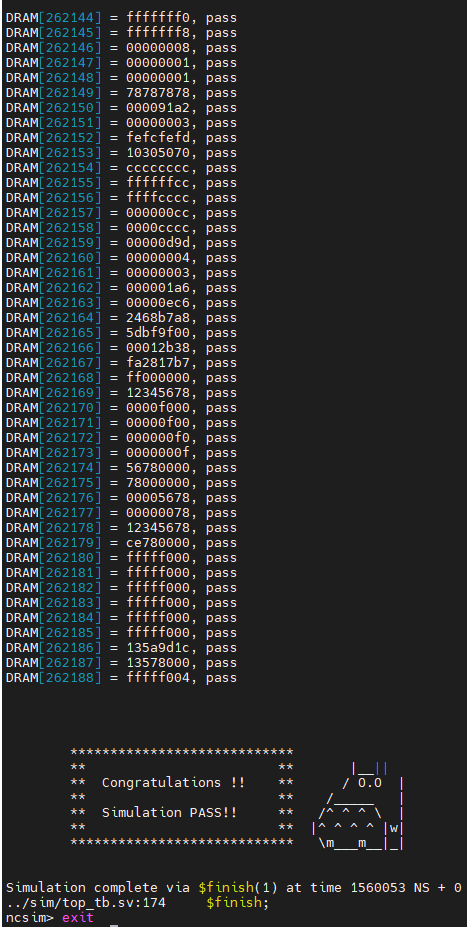
1. Superlint result



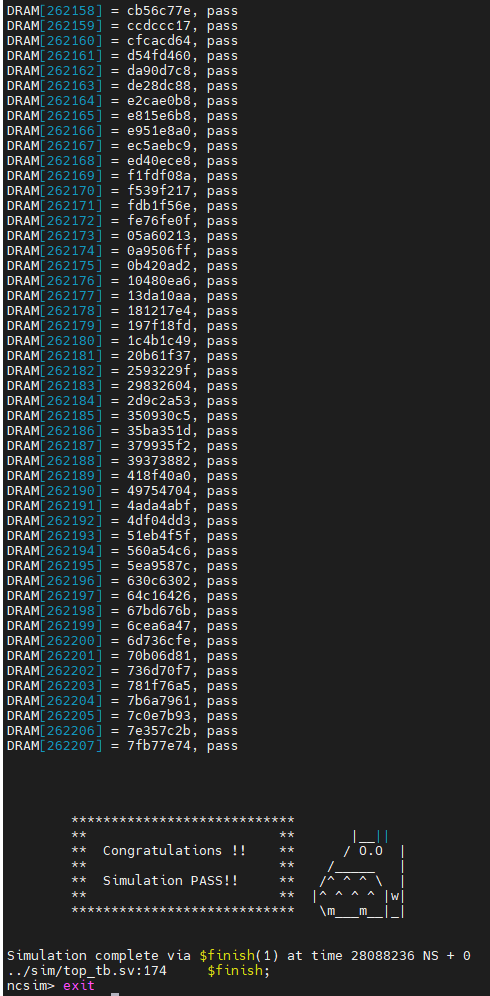
1. Performance and Area

* Performance

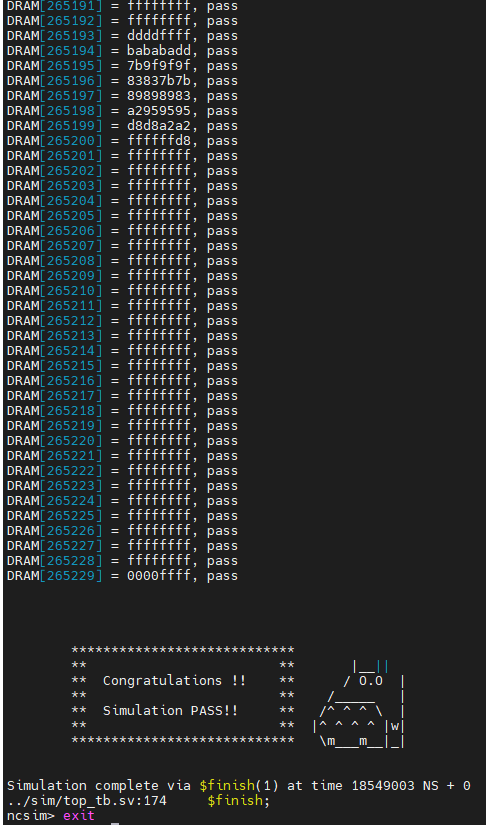
1. Prog0:



1. Prog1:



1. Prog2:



1. Prog3:

一張含有 文字 的圖片

自動產生的描述

* Area:

一張含有 文字, 監視器, 室內, 螢幕擷取畫面 的圖片

自動產生的描述

* Result Comparison with TA:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Prog0(ns) | Prog1(ns) | Prog2(ns) | Prog3(ns) | Area |
| TA | 458680 | 4096390 | 10990320 | 452970 | 6530417 |
| Me | 1560053 | 28088236 | 18549003 | 2876467 | 6490347 |

* **Lesson Learned**
* 在這次作業中我了解到Cache是如何運作的，以及要如何實做出一個符合spec的cache
* 學會了實作出DRAM\_wrapper、ROM\_wrapper讓DRAM和ROM的行為可以符合AXI的slave
* 了解資料是如何從DRAM搬運到SRAM當中讓程式去使用
* 了解到booting program對於一個程式的重要性，以及booting program的撰寫