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 | 劉彥麟 | 李秉軒 |
| Student ID | N26100723 | N26100595 |

Catalog

[Summary 3](#_Toc89228359)

[Contribution 3](#_Toc89228360)

[Problem 1 4](#_Toc89228361)

[Design Explanation 4](#_Toc89228362)

[Verification Result 6](#_Toc89228363)

[Waveform 7](#_Toc89228364)

[Problem 2 8](#_Toc89228365)

[Design Explanation 8](#_Toc89228366)

[Verification Result 11](#_Toc89228367)

[Terminal(Post-Simulation) 11](#_Toc89228368)

[Program 15](#_Toc89228369)

[Boot 15](#_Toc89228370)

[Program1 15](#_Toc89228371)

[Program2 15](#_Toc89228372)

[Program3 15](#_Toc89228373)

[IPC 15](#_Toc89228374)

[Hit rate 17](#_Toc89228375)

[Superlint 18](#_Toc89228376)

[Synthesis Report 20](#_Toc89228377)

[Lessons learned 24](#_Toc89228378)

# Summary

本次作業中共分兩個部分，第一部份，我們完成了L1 Instruction Cache和L1 Data Cache的設計，並通過JasperGold的驗證；第二部分，我們將HW1中的RISC V CPU接上我們在本次作業設計的Caches後再將其接上HW2中的AXI Slave，並且透過由HW2修改而來的AXI Bridge連接我們的Peripheral，如ROM、Instruction SRAM、Data SRAM、DRAM等，這些Peripheral都需要相對應的Wrapper來使其可以與AXI Bridge連接。最後完整的CPU、Cache、AXI bus、ROM、SRAM、DRAM經過合成，可達到clock period為10ns的performance。

在報告中，我們首先會說明本次作業中的分工，接著會介紹L1 Instruction Cache和L1 Data Cache的設計以及其JasperGold 的驗證結果；第二部分會說明如何將CPU、Cache、ROM、SRAM、DRAM等元件透過增加Wrapper使其可以與AXI Bridge連接，以及4個Program的測試結果；最後則會說明我們合成的結果，以及本次作業中遇到幾個比較大的問題與我們的解決辦法。

# Contribution

劉彥麟(N26100723) ： 50%

AXI Bridge、ROM Wrapper、SRAM Wrapper、DRAM Wrapper、Pre-simulation、Synthesis、 Post-simulation。

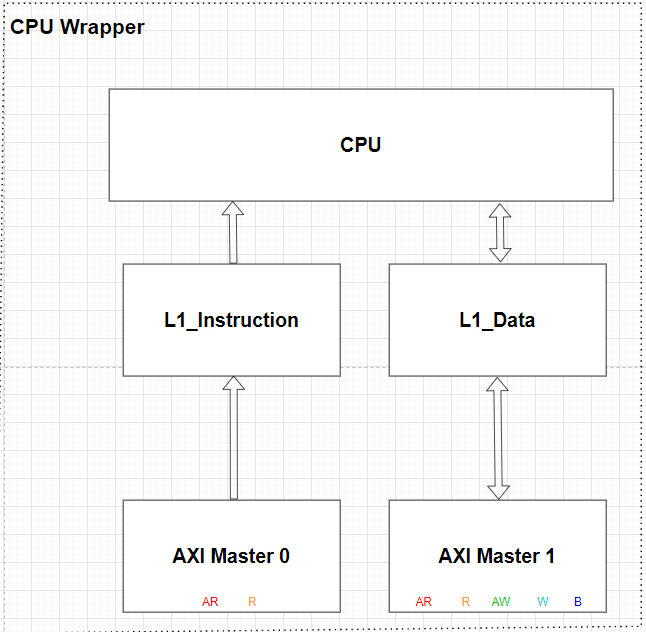
李秉軒(N26100595)：50%

RISC-V CPU、L1 Instruction Cache、L1 Data Cache、CPU Wrapper、Pre-simulation、Synthesis、 Post-simulation。

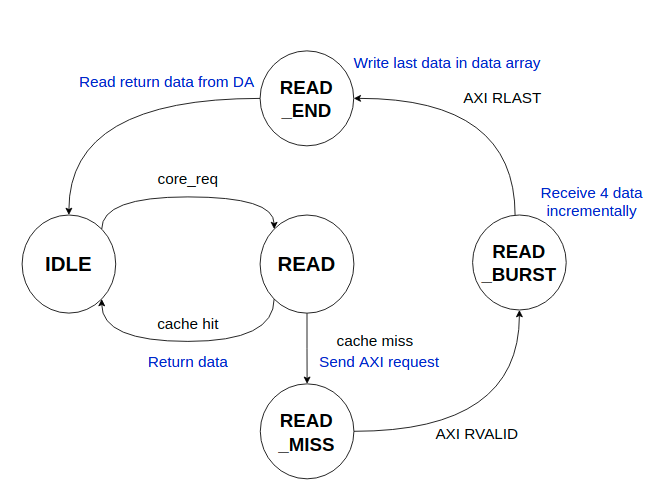
# Problem 1

## Design Explanation

Problem 1的部分主要是L1 Instruction Cache和L1 Data Cache的設計。



本次作業設計了兩個L1 cache，分別儲存IM及DM的快取資料，下圖為Cache的Read state machine及Write state machine。



以下簡單介紹各個state的行為：

**IDLE** : 在此階段，cache會接收來自CPU的request，若有request則會立即stall CPU，根據檢查到是Read或Write，會決定下一個state。同時，該階段會將CPU的address丟給Tag array及檢查valid bit，並在下個cycle檢查是否hit。

**READ** : 此階段會檢查是否cache hit。若是hit，則會把資料回傳給CPU，並解除CPU端的Stall。若是沒有hit，則會將request傳給wrapper，讓wrapper發出AXI的request，並繼續stall CPU。

**READ\_MISS** : 此階段會開始等待AXI回傳資料，當第一筆資料回來，就跳入下個state。

**READ\_BURST :** 由於我們是用burst mode一次存取一個cache line，所以額外使用一個counter計算回來的資料是第幾筆，同時將傳回來的資料存入data array。和AXI溝通的部份主要交由wrapper進行，cache只觀察wait訊號。當counter計算是最後一筆資料，就跳入下個state。

**READ\_END :** 該state為read最後一個state，會將要回傳到CPU的data從data array讀出，並同時解除CPU的stall。最後回到IDLE state。

**WRITE** : 在data cache則是加上了write state，由於write的機制是write through和write around，等於有沒有hit都要寫回memory，因此只要判斷有hit時要寫入data array，而也因為資料寬度只有一個word，所以也不需要burst。

**Cache port :**

這次作業中，我們直接沿用之前CPU和AXI 溝通的方式，將type port 移除，改使用strobe訊號溝通。

**Branch行為 ：**

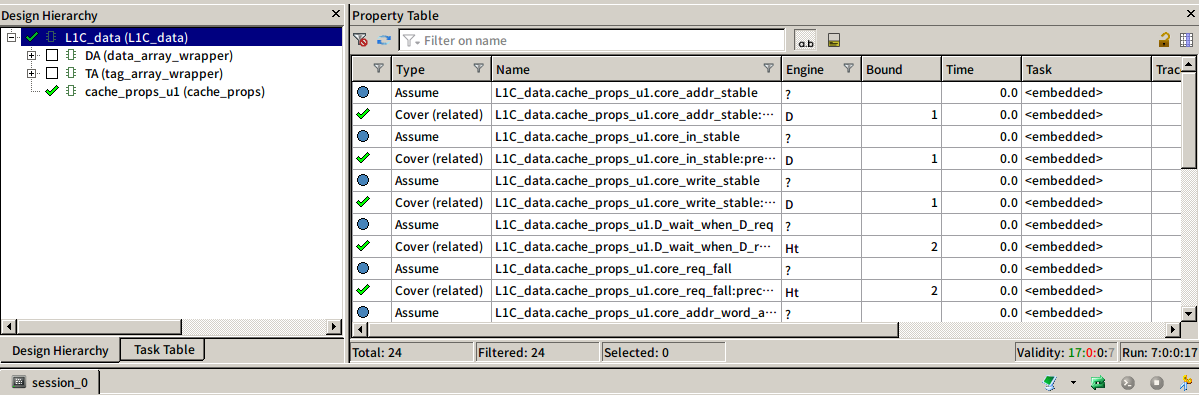
當CPU發現要branch時，又剛好上一條指令發生cache miss，會導致cache去存取一條用不到的cache line。

觀察發現，大部分時候發現branch時，cache都正要對AXI 發出request，因此我們將要發出request的address在同個cycle換為要跳轉的目標address，這樣就能來得及取得有意義的資料。儘管目前的CPU設計會在資料讀取完後，再向cache存取一次跳轉的目標address，但這次存取就會cache hit，依舊能夠得到performance的增益。

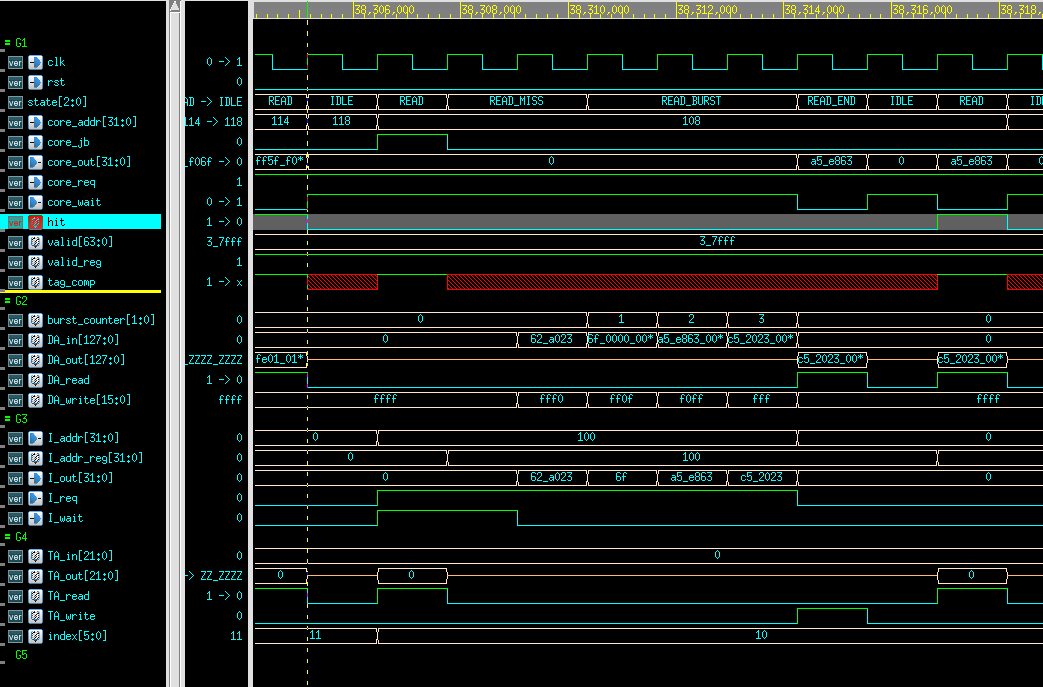
目前的設計上仍有些瑕疵，例如其實可以讓cache在發現有branch時，再重新檢查一次是否hit，經過測試可以讓performance獲得大量提昇，但可惜再作業時間內來不及通過所有program測試。此外，目前的設計會有一條從運算branch到發出AXI request的critical path，也是可以優化的部份。

## Verification Result

本設計之L1 Data Cache經過JasperGold 之驗證，以下為驗證結果：



### Waveform

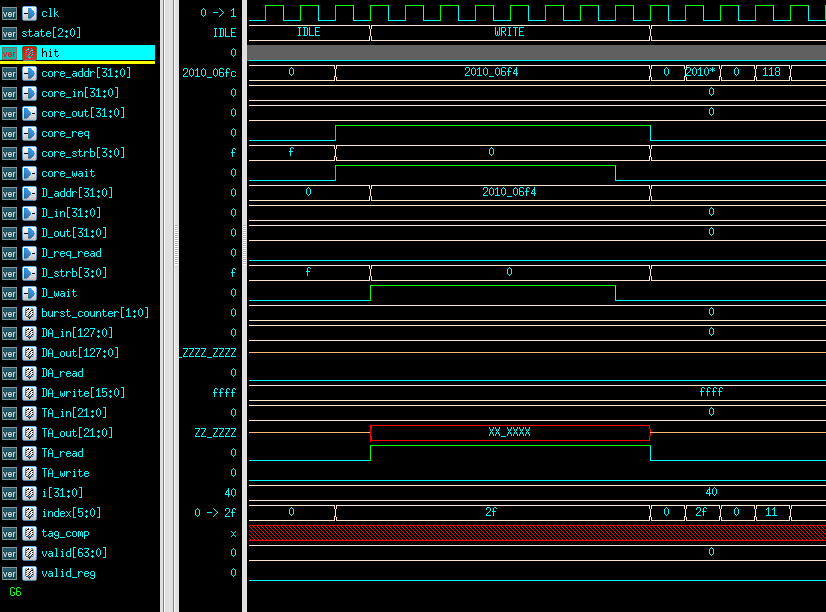


上圖為i\_cache的波形圖，首先看到core\_addr為118時，cache讀到core\_req，此時會將index丟給Tag RAM。

等下個cycle進入READ state，會比對Tag和address是否hit，但在波形中可以看到core\_jb訊號被拉起，以及core\_addr改變，代表CPU發現要執行跳轉，因此cache會將此次比對結果強行轉為miss，並對AXI發出對跳轉指令的request。由於cache對memory存取的方式是一次一條cache line，因此發送的地址必須與line對齊。

進入到READ\_MISS state，cache會等待直到第一筆資料回來，當第一筆資料返回，會進入到READ\_BURST state，並同時將讀回的資料寫入data array。而burst\_counter訊號就是讓cache用來紀錄目前是第幾筆資料。

讀完4筆資料後，進入到READ\_END state，再將要傳回給CPU的值從data array傳回。然而，儘管已經傳回正確的資料，CPU會假設這次資料是錯的，這是由於上次作業使用AXI bus，是來不及取回正確指令的。因此CPU會再發送一次指令，但這次就會hit，取回正確資料。



上圖為data cache的波形圖，詳細的機制已在上方說明。

# Problem 2

## Design Explanation

Problem 2的設計將RISC V CPU接上我們在本次作業設計的Caches後再將其接上HW2中的AXI Slave，並且透過由HW2修改而來的AXI Bridge連接我們的Peripheral，如ROM、Instruction SRAM、Data SRAM、DRAM等，這些Peripheral都需要相對應的Wrapper來使其可以與AXI Bridge連接，以下將進行詳細的介紹：

1. AXI Bridge

本次的AXI Bridge沿用HW2的AXI Bridge，將原本的2個Master＋2個Slave改為2個Master＋4個Slave，由於Master的數量沒有改變，所以我們不需要修改Arbiter，只需要修改Decoder，本次作業我將我們Decode的Address改為：

ROM → 0x0000\_0000～0x0000\_1FFF

IM(SRAM) → 0x0001\_0000～0x0001\_FFFF

DM(SRAM) → 0x0002\_0000～0x0002\_FFFF

DRAM → 0x2000\_0000～0x201F\_FFFF

上述的Address的範圍會轉接至該Peripheral之Wrapper，其餘的Address均由Default Slave負責。

1. ROM Wrapper

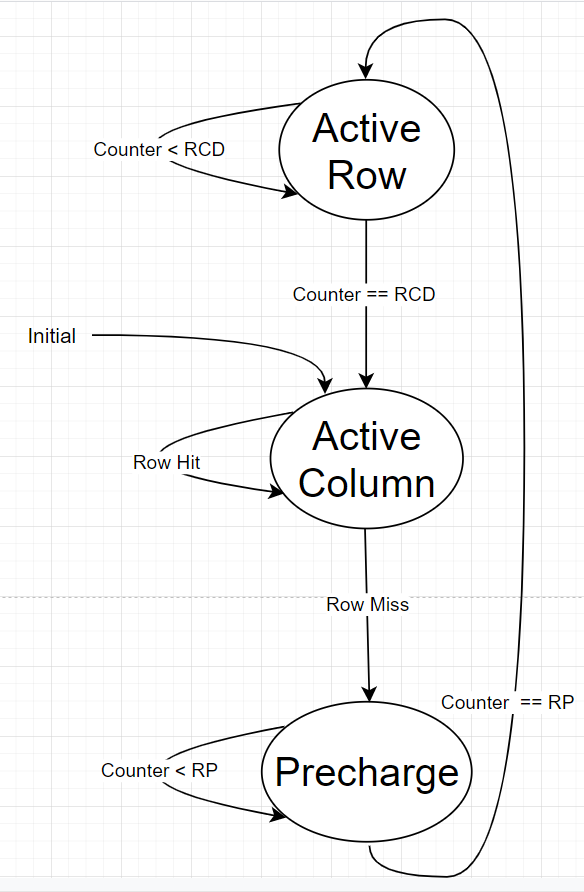
本次作業的Booting程式儲存於ROM，由於是Read-Only Memory，因此只需要Read不會Write，又其Read的行為和HW2之SRAM一樣，所以可以直接沿用HW2的AXI Slave，只需要將Address由14-bit改為11-bit即可，此處都使用Word Addressing。

1. SRAM Wrapper

本次作業的使用的SRAM和HW2之SRAM一樣，所以可以直接沿用HW2的AXI Slave。

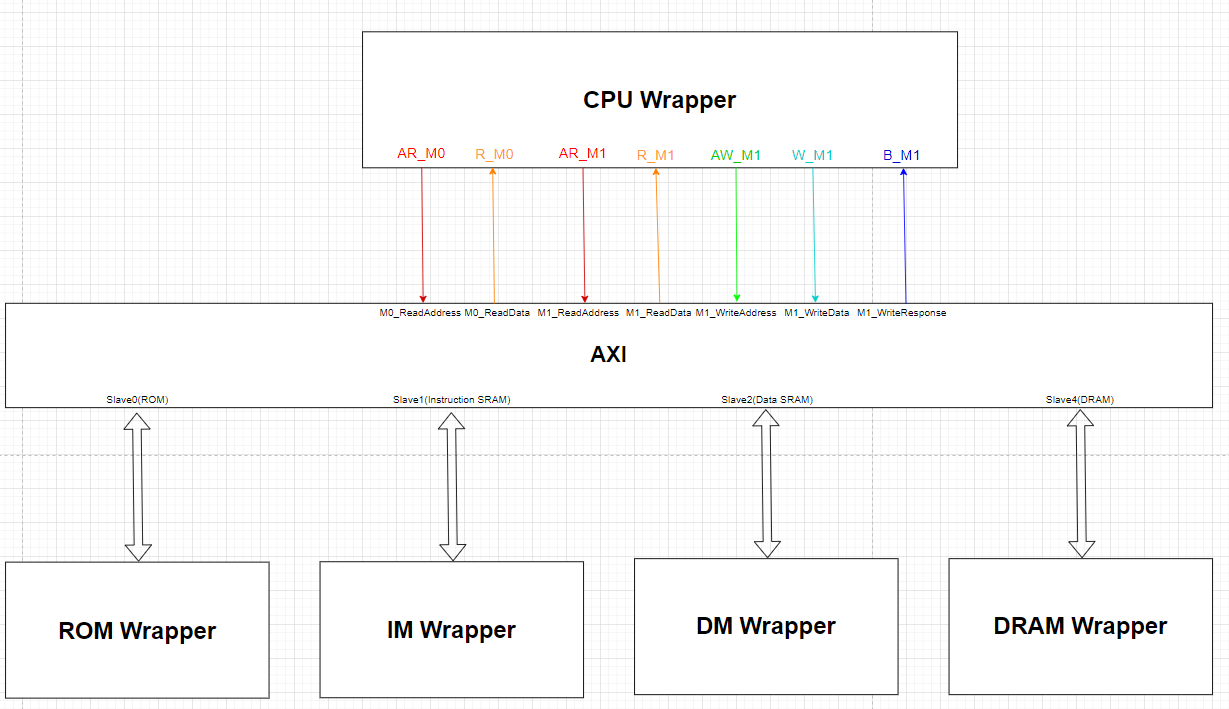
1. DRAM Wrapper

本次作業的Program和Data都存在DRAM中，DRAM並非同SRAM，其需要先Active Row再Active Column，如果是Row Miss則需要先進行Row的Precharge，之後再Active新的Row，為此需要設計一個DRAM Controller，我們DRAM Controller的State Machine如下：



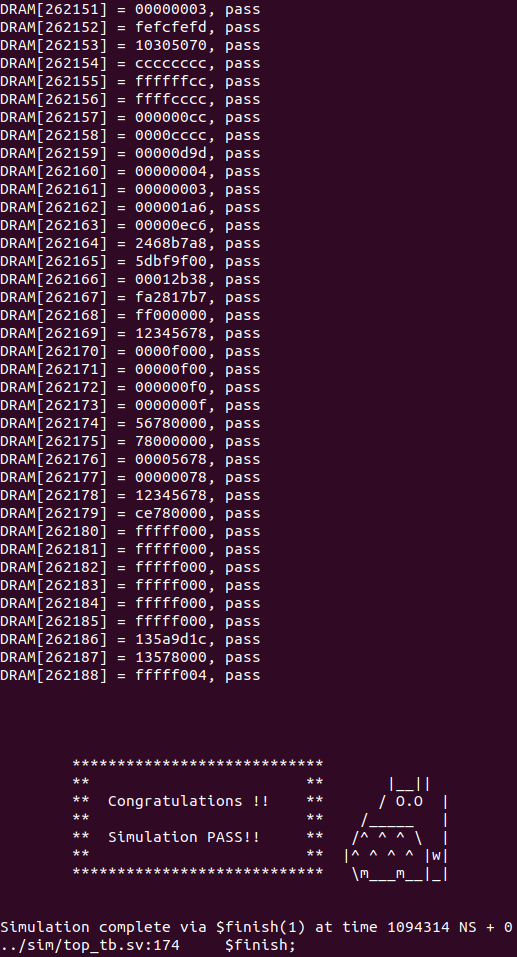
每個State都需要5個Cycle，因此DRAM最理想的Read/Write Delay是5個Cycle，最差是15個Cycle，即Access Request有Row Hit是Best Case，相反的Access Request如果Row Miss，則需要先Precharge再Active Row再Active Column。

將上述Problem1的CPU、Cache等元件組合而成的CPU Wrapper和Problem2的AXI Bridge、ROM Wrapper、SRAM Wrapper、DRAM Wrapper結合起來後就會如下：

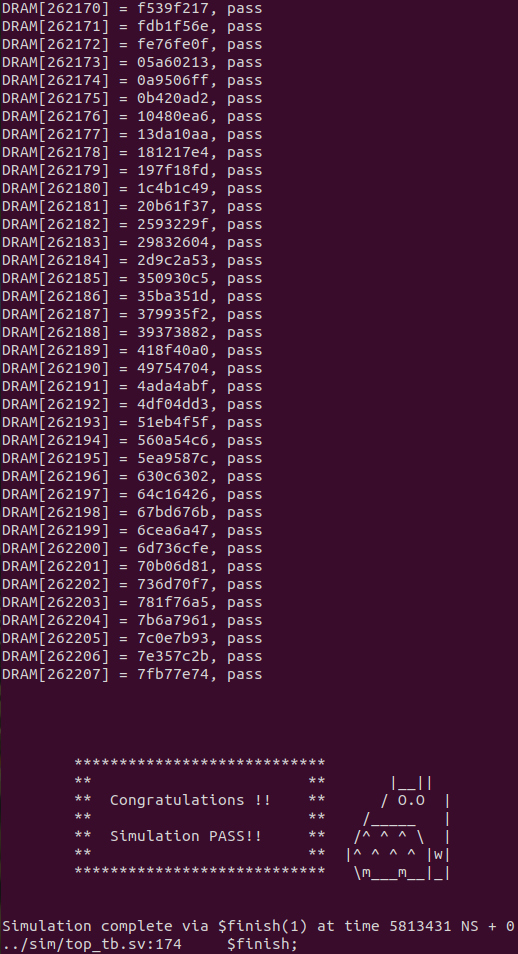


## Verification Result

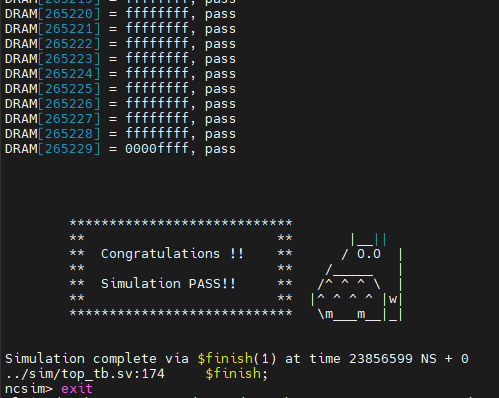
### Terminal(Post-Simulation)



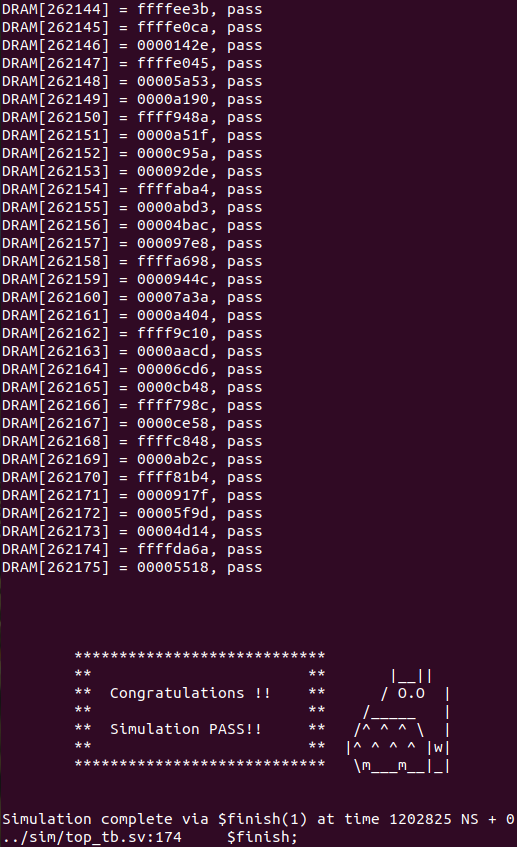
(圖) Program0



(圖) Program1



(圖) Program2



(圖) Program3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Comparison | prog0(ns) | prog1(ns) | prog2(ns) | prog3(ns) |
| Ours | 1094314 | 5813413 | 23856599 | 1202825 |
| TA | 458680 | 4096390 | 10990320 | 452970 |

(表) Execution Time Comparison

# Program

## Boot

Boot這個程式必須將DRAM中的Instruction、sdata、data等Data存到Instruction SRAM和Data SRAM，因此此程式使用3個While Loop完成。

## Program1

Program1是Half Word的Sorting，此程式使用的Algorithm是Insertion Sort，因此我們使用兩個While Loop完成。

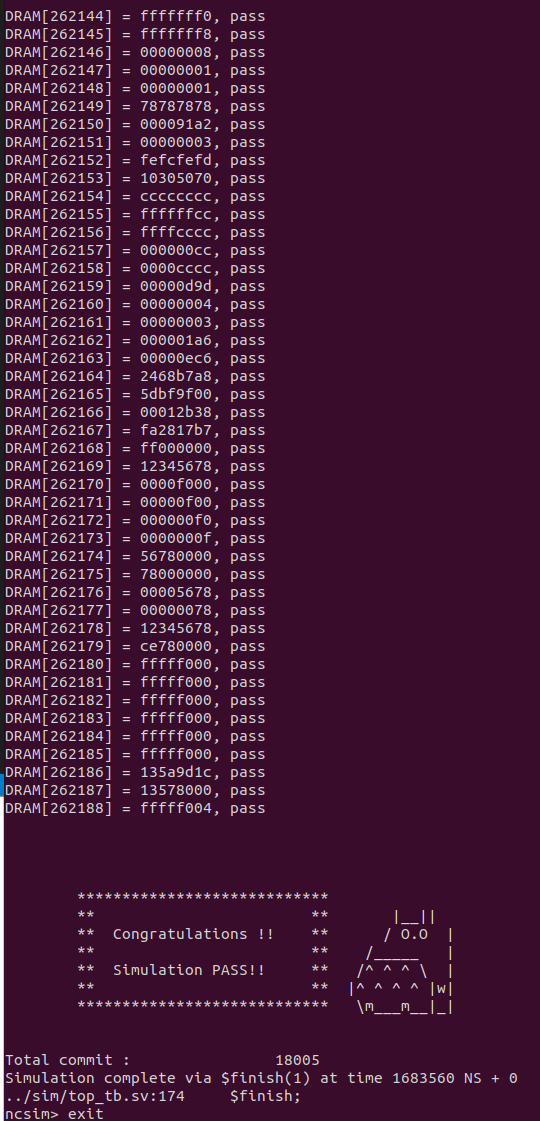
## Program2

Program2是將BMP圖轉成Gray Scale，計算的公式是0.11 × Blue＋0.59 × Green＋0.3 × Red，將Output Picture的每個Pixel中Blue、Green、Red，都設為上述公式的結果，此程式使用While Loop來Traverse Input Picture。

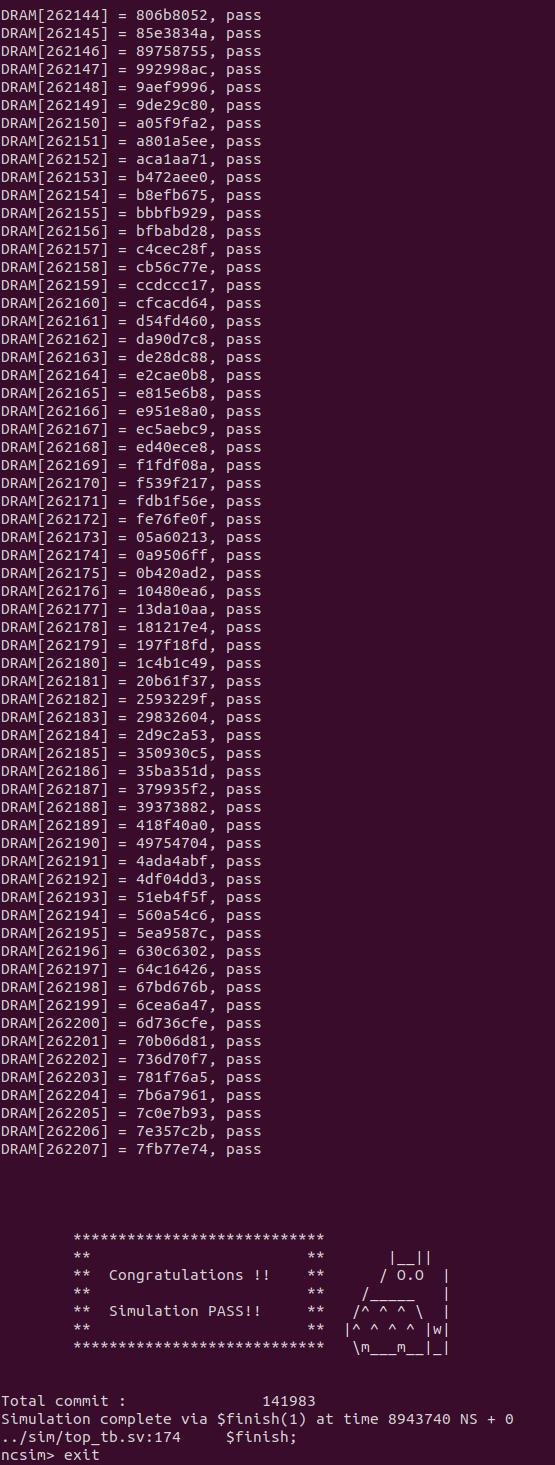
## Program3

Program3是Matrix Multiplication，此程式使用3層的For Loop完成Matrix A和Matrix B每個Element的Traverse。

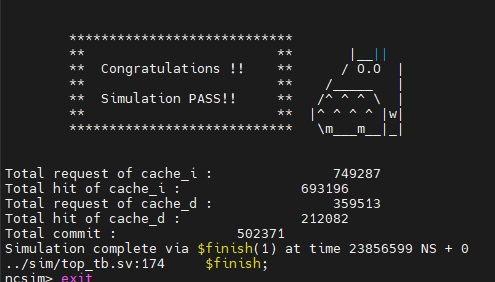
## IPC



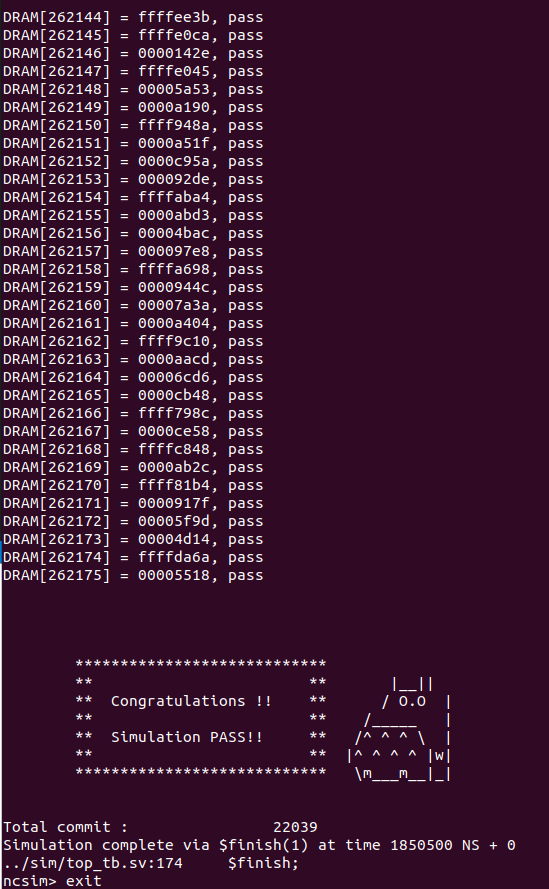
(圖)Program 0



(圖)Program 1



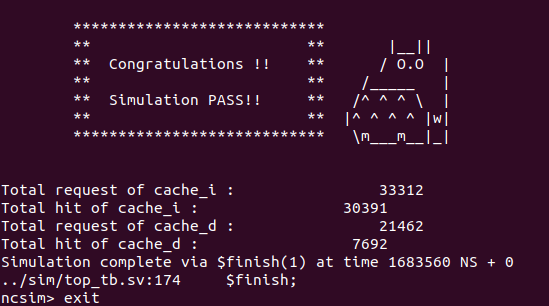
(圖)Program 2



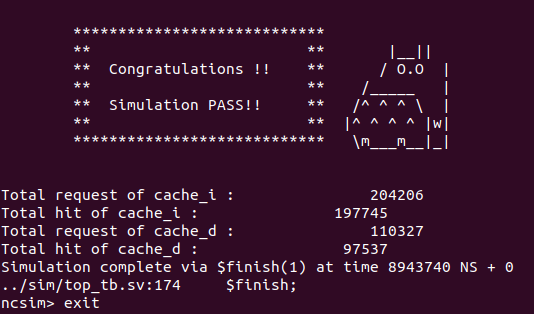
(圖)Program 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | prog0 | prog1 | prog2 | prog3 |
| IPC | 0.21 | 0.3175 | 0.273 | 0.2381 |

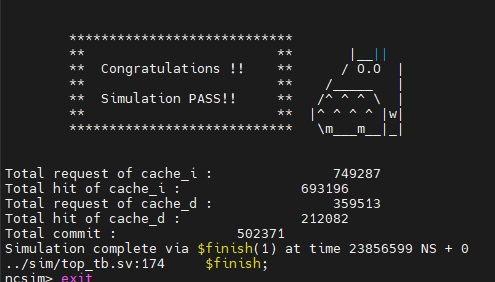
## Hit rate



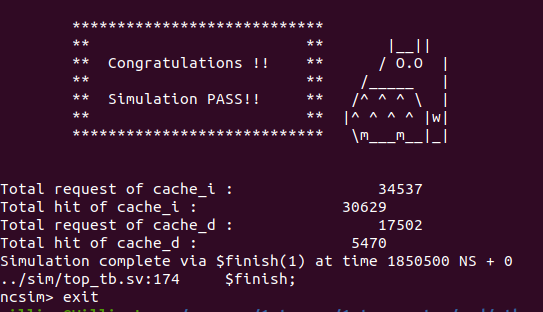
(圖)Program 0



(圖)Program 1



(圖)Program 2

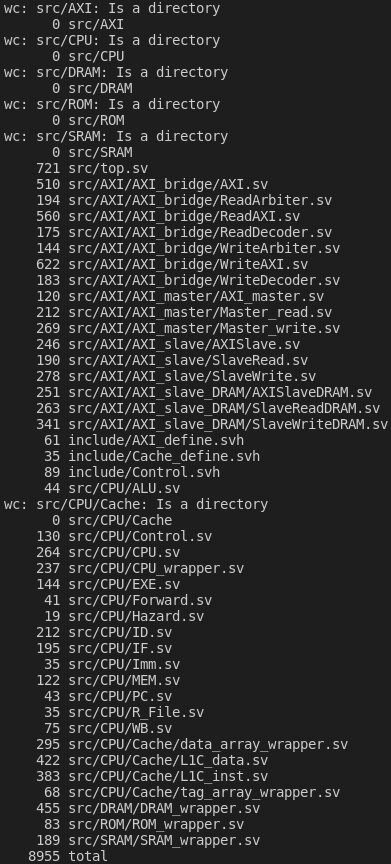
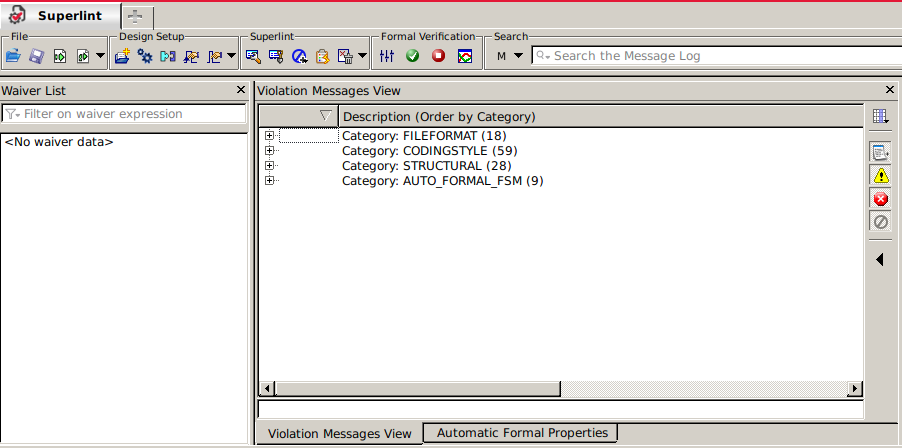


(圖)Program 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cache Hit Rate | prog0 | prog1 | prog2 | prog3 |
| i\_cache | 0.91 | 0.968 | 0.925 | 0.87 |
| d\_cache | 0.358 | 0.884 | 0.589 | 0.312 |

# Superlint

Overall Report：

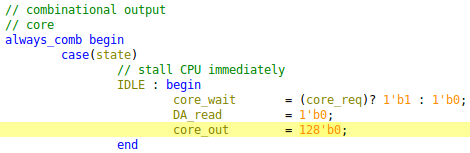
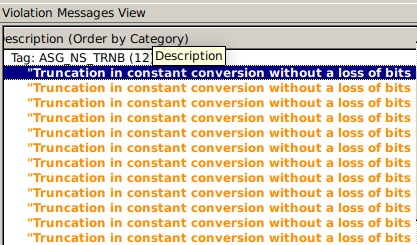


Total Violation = 114

Code Length = 8955

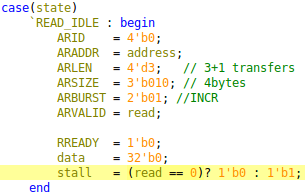
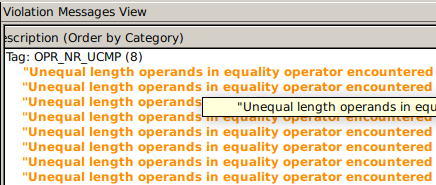
Score = 98.7%

Violation1：ASG\_NS\_TRNB



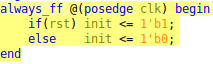
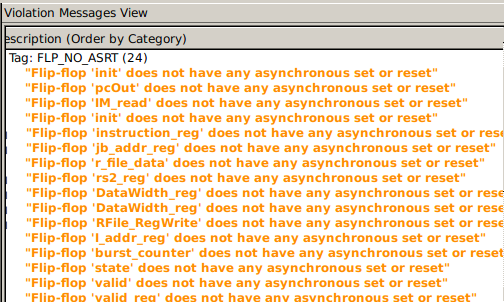
core\_out = 128’b0; should be core\_out = 128’d0;

Violation2：OPR\_NR\_UCMP



read == 0 Should be read == 1’b0

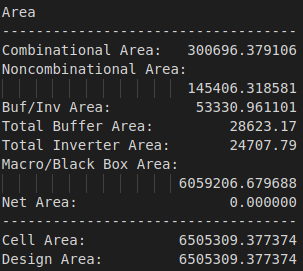
Violation3：FLP\_NO\_ASRT



always\_ff @(posedge clock) should be always\_ff @(posedge clock or posedge reset)

# Synthesis Report

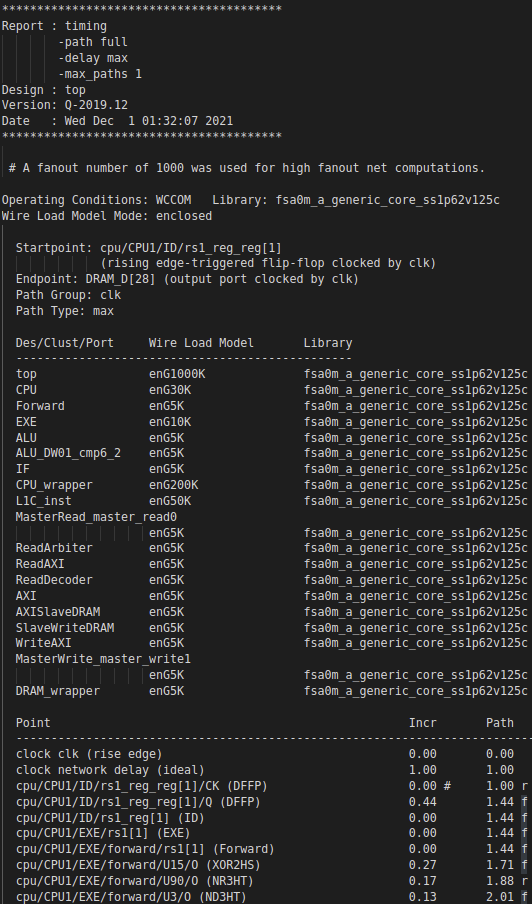
1. Area

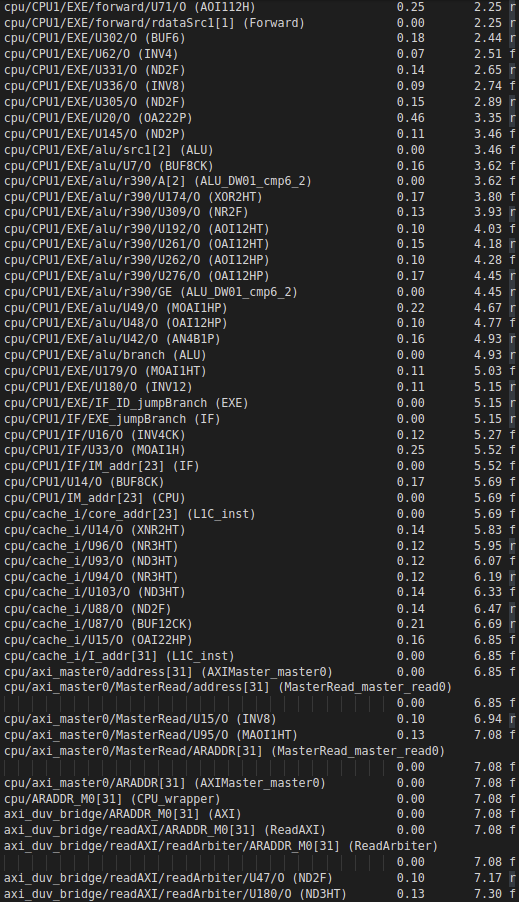


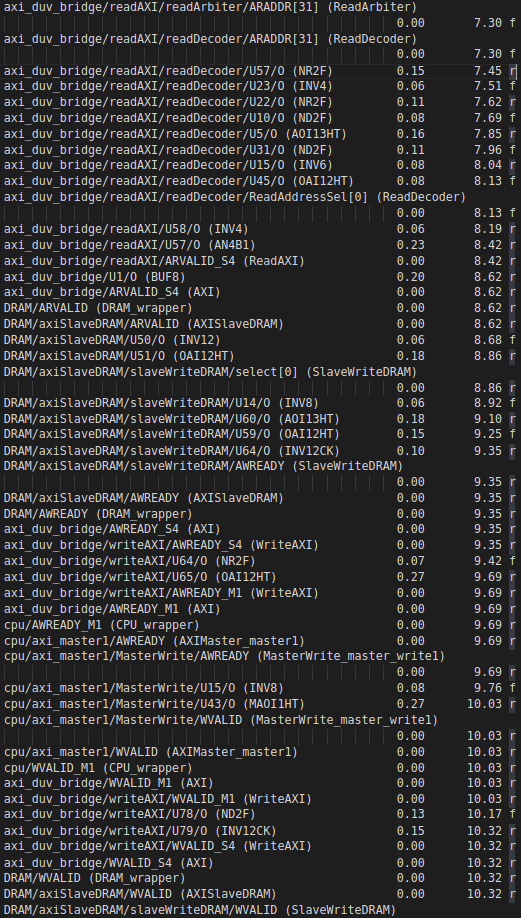
|  |  |
| --- | --- |
| Comparison | Area |
| Ours | 6505310 |
| TA | 6530417 |

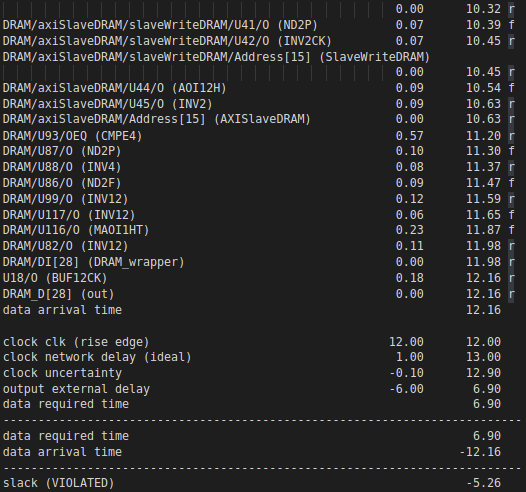
(表) Area Comparison

1. Timing









# Lessons learned

李秉軒：

這次的作業第一次開始有「系統」的感覺，完成簡單的程式，但要先在各種記憶體中跑來跑去，因此必須要對整個流程更清楚才能完成作業。這次主要負責cache的部分，其實蠻有趣的，例如Branch可以來得及打正確request，其實是為了解combinational loop誤打誤撞的結果。也還有很多想嘗試的優化，希望下次有機會可以大展拳腳。

劉彥麟：

這次的作業真的非常的趕，幾乎只有兩週的時間去完成這次作業，由於我在HW2就是負責AXI Bus和Slave Wrapper，因此做這次作業時比較沒有遇到撞牆的問題，但是在撰寫DRAM Controller時，由於沒有考慮周全，因此導致第一版的DRAM Wrapper行為怎麼改都不會對，之後又花了兩天重寫一版新的DRAM Wrapper，感謝我的組員耐心等待我修好Bug，也感謝他陪我趕報告到凌晨5：30。