VLSI System Design (Graduate Level)

Fall 2022

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:陸啟倫

Student ID: N26114950

**outline**

**1.Summary 3**

2.Major problem encountered and resolution3

3.Block diagram4

**4.State diagram introduction 6**

(a)Inst. Cache**6**

(b)Data Cache**7**

(c)AXI Write arbiter**8**

(d)AXI Read arbiter**9**

(e)ROM**10**

(f)DRAM**11**

5.Hit rate and IPC13

6.C code explanation15

7.Waveform19

8.AXI ABVIP23

9.Simulation result 23

10.Superlint result 28

11.Performance 30

12.Lesson learned 32

1. **Summary**

這次作業實作data cache和Inst. cache並加入至CPU wrapper中。以運用資料locality的特性，加快指令和資料讀取的速率。也實作了DRAM和ROM作為兩顆系統外的記憶體，並擴充及驗證AXI，以作為module之間的溝通橋樑。在DRAM的實作上，尚須滿足其內部的assertion，以模擬實際DRAM在存取時，需要較長時間的狀況。將系統整合完成後，成功完成booting、RTL prog0~prog3、及syn prog0~prog3的模擬。合成後clock period為14ns(slack=0)。

作業分工為陸啟倫(100%)。

1. Major problem encountered and resolution
2. 在模擬prog1時，發生load及store指令未正確拿取資料的狀況。

Sol: store data部分須根據ALU result[1:0]存取對應的部分，load資料須根據ALU result[1:0]拿取word中適當的部分並視指令決定是否做sign extension。

1. 在使用burst mode傳送從記憶體拿取的資料時，cache中的buffer未正確存取回來的資料。

Sol:需有一個溝通訊號告訴cache這筆回來的資料是可以存的。當RVALID和RREADY握手時，我會將wait訊號設為0。當cache中的狀態機在存資料的狀態並吃到wait=0的訊號時，則需將資料推入buffer。

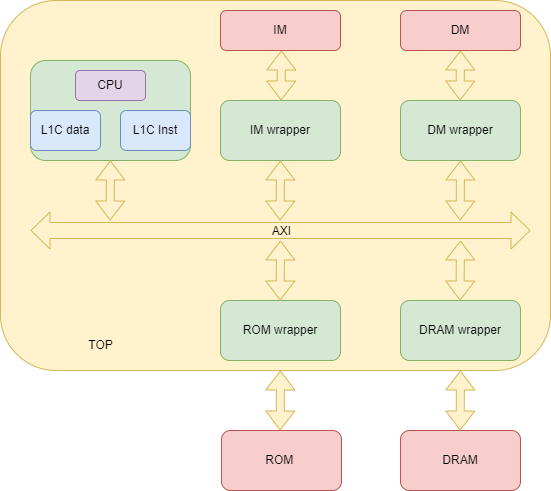
1. Data cache和Inst. cache 同時要求使用bus權限時，需處理優先級問題。

Sol:我在CPU wrapper的狀態機設計上使Inst. cache有較高的優先級，待Inst. cache拿取指令後才會讓data cache獲得存取權。

1. DRAM狀態機在設計時，row hit功能未正確執行

Sol: 我將狀態機修正在Idle狀態判定是否有row hit ，再來決定對應的next state，即可完成此功能。之前無法有row hit功能的原因可能出自，DRAM在比較row hit是否成立的狀態時，Address未正確傳進DRAM。

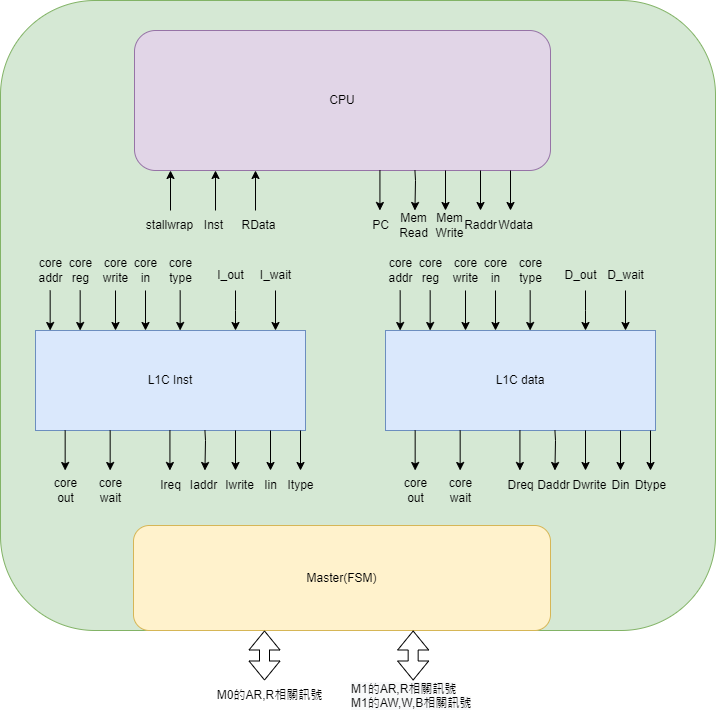
1. **Block diagram**

****(a)top的block diagram

說明:

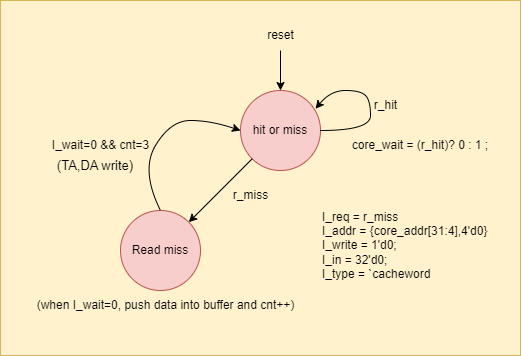
top內部包含一些modules，分別為CPU wrapper(新增兩顆快取)、IM wrapper、DM wrapper、ROM wrapper、DRAM wrapper、AXI。每個wrapper都需透過AXI來進行訊號溝通，以互相傳遞資料。其中IM、DM是屬於系統內的SRAM記憶體，有較快的存取速度，而ROM、DRAM是系統外的DRAM記憶體，有相較SRAM慢的存取速度，但擁有較大的容量。由ROM、DRAM所讀取的資料需由系統外進入ROM wrapper、DRAM wrapper中，而這兩個wrapper內部都有狀態機來控制相關訊號及資料，以和AXI進行溝通。

(b)CPU wrapper的block diagram



說明:

CPU wrapper由CPU、L1C Inst、L1C data 及Master(FSM)組成。當CPU向L1C Inst要求指令或向L1C data要求讀資料時，會檢查發生read miss或read hit，若發生前者，則會由快取發生要求使用 bus的訊號，以向其他的記憶體索取資料，並待bus傳回後存入快取中，再提供給CPU。若後者發生，則直接由快取提供資料，以省下向bus索取資料的時間。當CPU向L1C data要求寫入資料時，則會檢查發生write miss 或 write hit，無論發生何者都一定會向bus發出要求，因我採取的是write through的設計，兩者的差別只在發生write hit時，會將快取內部的數值一起寫入。而Master則是控制與bus的溝通訊號的狀態機。

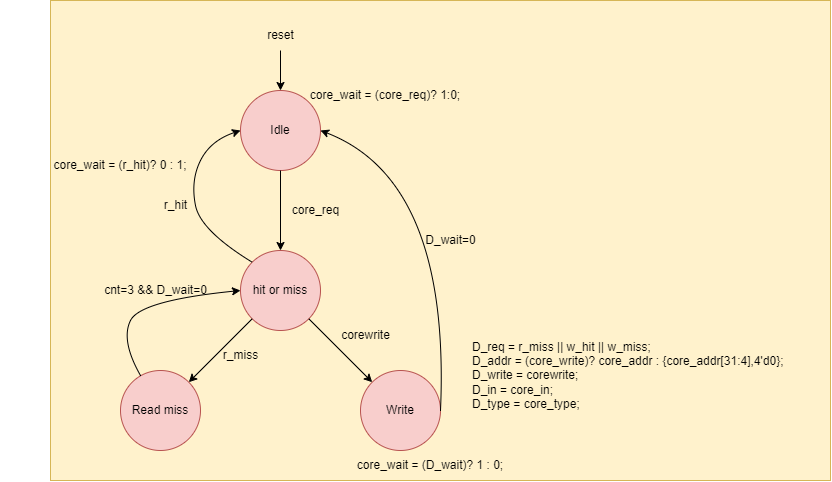
1. **State diagram introduction**
2. L1C Inst. 的狀態機

說明:

上圖為Inst. cache內部的狀態機，只有hit or miss 和 Read miss 兩種種狀態。不考慮設計Idle狀態的原因是CPU會不斷的要求讀取指令，所以沒有必要再新增一個Idle狀態在獲得CPU req時再進入hit or miss 狀態去判斷，這樣只是多等待一個clk。

hit or miss狀態負責判斷目前要讀取的指令在快取中是否存在，若存在則可直接透過快取獲得，並在r\_hit的同時使core\_wait訊號為零，以告訴CPU Ins. Cache已完成工作。

Read miss狀態只有在r\_miss發生時才能進入，在Read miss 狀態時則會向bus要求使用權(I\_req)，並傳送I\_addr等相關訊號至bus，待bus將指令送回後，則會根據bus給的I\_wait訊號來計數已收到的指令數並將收到的指令推進buffer中。由於我們的系統採取的burst length為4，所以需要在cnt數到3(已收到三筆資料)且收到I\_wait訊號為0(第四筆資料剛好回來)時，在下個clk切換狀態至hit or miss。同時會將TA write及DA write設定為寫入(low active)，以確保在回到hit or miss狀態的瞬間， DA和TA內的數值也能成功更新。由於DA和TA已剛被更新，所以此時一定會是r\_hit，則會將指令由cache傳至CPU，並發生core\_wait為零的訊號，以通知CPU Inst. Cache以完成工作。



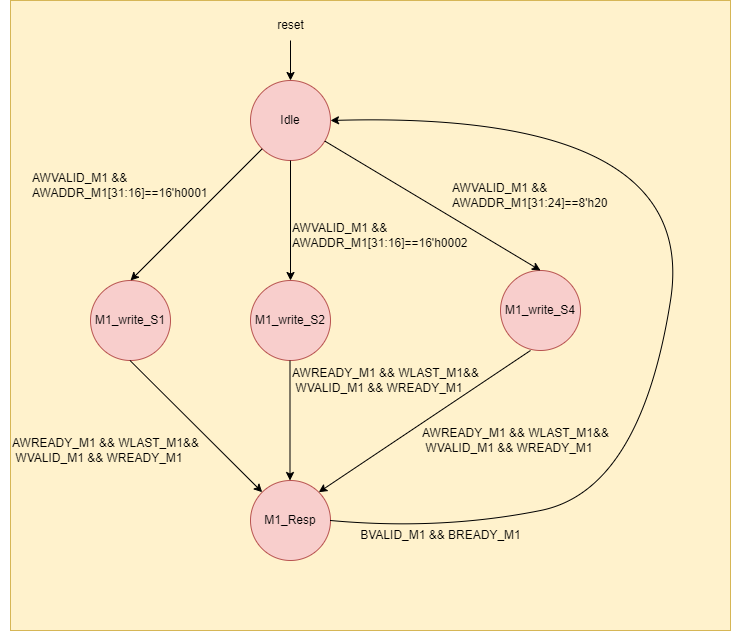
1. L1C data的狀態機

說明:

上圖為L1C data內部的狀態機，總共包含四個狀態分別為Idle、hit or miss、Read miss、Write。在Idle狀態時，當CPU遇到load或store相關指令時，則會發生Mem read或Mem write訊號，當發出兩者任一的訊號時，core\_req就會拉起，表示CPU要對data cache做存取的動作，當快取獲得core\_req訊號時，則會發出core\_wait訊號通知CPU要暫停管線。接著進入hir or miss狀態，在此狀態會判定要讀取的資料使否存在快取內、要寫入的資料是否存在快取內。

在hit or miss狀態時，若發生r\_hit，則在下個clk會回到Idle狀態、將資料給CPU、並發出core\_wait為零的訊號通知CPU快取的工作已結束。當發生r\_miss，則會在下個clk進入Read miss狀態，在此狀態會對bus發出request、並給予bus相關訊號，同時使用D\_wait訊號計數從bus回來的資料並存入buffer中。待cnt=3且D\_wait=0時，則會將DA write 和TA write設成寫入並在下個clk回到hit or miss狀態，在回到hit or miss後則一定會發生r\_hit，再將資料送給CPU。

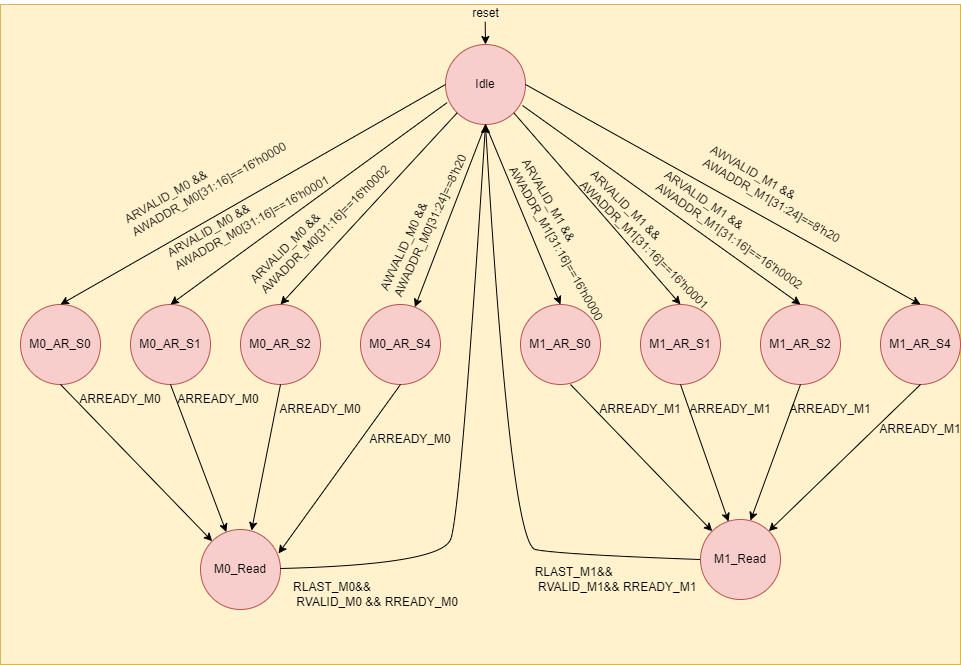
在hit or miss狀態時，無論發生w\_hit或是w\_miss都必須進入Write 狀態，因快取採取的是write through策略。在Write狀態則會對bus發出request，並給予bus相關訊號，待要寫入的資料成功透過bus將Mem的數值更改後，bus則會回傳D\_wait為零的訊號，當快取獲得此訊號時，則會在下個clk回到Idle狀態，並將core\_wait設為零，表示快取的工作已結束。(若發生w\_hit，則需要在D\_wait=0時，同時將DA write訊號設成寫入，以確保回到Idle狀態的瞬間DA能被更新)



1. AXI的狀態機(Write Arbiter)

說明:

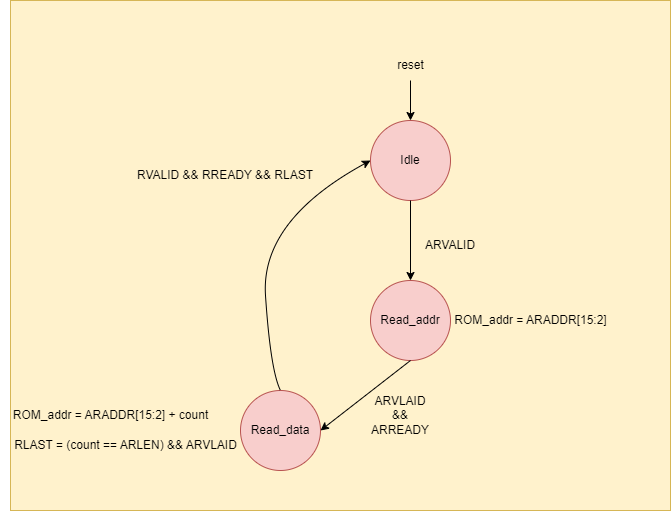
上圖為AXI中控制寫入的狀態機，初始狀態時須根據AWADDR\_M1的數值來決定要將資料寫入slave0、slave1、或slave4(不須考慮AWDDR\_M0的數值，因為只有Master1能夠寫入數值)，再待AWVALID訊號後即可至對應的狀態。接著進入M1\_write\_S1、M1\_write\_S2、或M1\_write\_S4狀態。在這些狀態時，AXI會打開AW及W通道，並將AWID、AWLEN、AWADDR、AWSIZE、AWVALD、AWREADY等AW通道相關資訊及WID、WDATA、WLAST、WVALID、WREADY等W通道相關資訊傳送至對應的master或slave。接著待AW通道及W通道都握到手後，則可進入M1\_Resp。在M1\_Resp中，AXI會打開B通道，並傳送BID、BRESP、BVALID、BREADY等B通道相關資訊至對應的slave或master，待B通道握手後則可再回到初始狀態。



(d)AXI的狀態機(Read Arbiter)

說明:

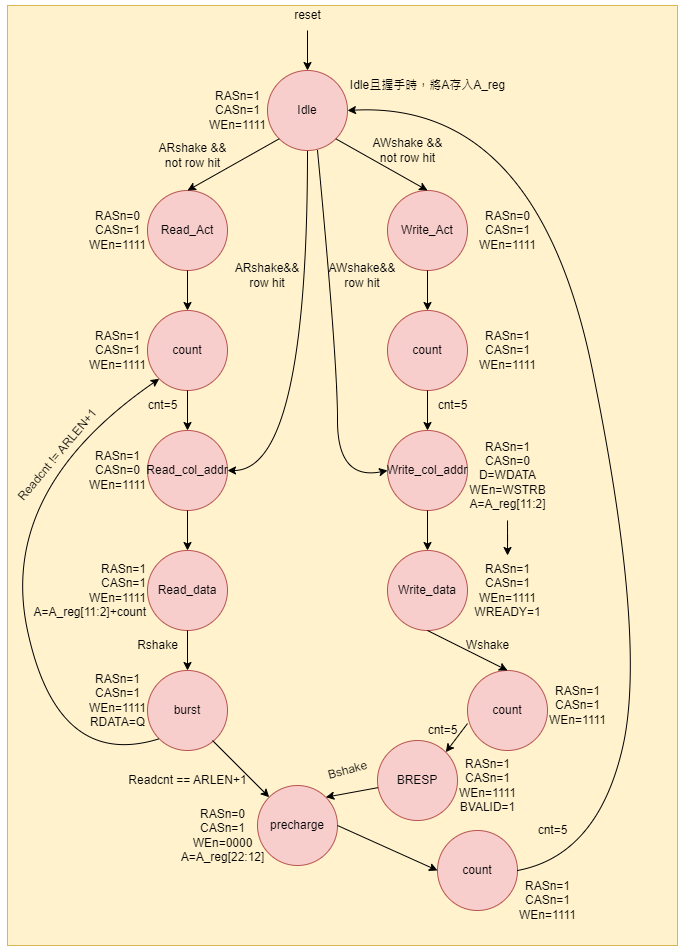
上圖為AXI中負責控制讀取的狀態機，起始狀態時需根據送入AXI的ARADDR\_M1，ARADDR\_M0來判斷這個地址是要送到哪一個slave當中(ROM的addr介於0x0000\_0000~0x0000\_ffff、IM的addr介於0x0001\_0000~0x0001\_ffff，DM的addr介於0x0002\_0000~0x0002\_ffff、DRAM的addr介於0x2000\_0000~0x201f\_ffff)，再待對應的ARVALID後即可進入對應的AR狀態(共有M0傳給S0、M0傳給S1、M0傳給S2、M0傳給S4、M1傳給S0、M1傳給S1、M1傳給S2、M1傳給S4，八種情況)。進入AR狀態時，AXI則會將AR通道打開，並將ARID、ARLEN、ARADDR、ARSIZE、ARVALD、ARREADY等資訊傳送至對應的slave或master。接著開始等待slave傳送ARREADY，代表slave已經準備接收資訊，兩方握手後，資料就能成功傳輸並進入下個狀態(M0\_read或M1\_read)。進入Read狀態後，AXI則會打開Read通道，並將RID、RDATA、RRESP、RLAST、RVALID、RREADY等訊號傳送至對應的slave或master。由於站在CPU讀取資料的角度，可能會有來自slave0、slave1、slave2、slave4的情況，所以需要根據AXI在接收到來自不同Master的ARID時，做不同的高位元的編碼。待slave接收到ARID後則會將高位元的編碼存起來，待傳送RDATA時能順便加在RID的高位元部分，以便AXI分別資料要送至哪個Master。接著等待RVALID、RREADY、RLAST訊號，以確保握手成功及傳送的是最後一筆資料。才能回到初始狀態。



(e)ROM wrapper的狀態機

說明:

上圖為ROM wrapper中的狀態機，吃到reset訊號時會先進入Idle狀態，若有ARVALID則在下個clk進入Read\_addr，在Read\_addr狀態時，則會將ARREADY傳送出去，同時將收到的ARADDR[15:2]，以ROM\_addr傳遞出系統外。再待AR訊號握手後，則可進入Read\_data狀態，並將RID、RDATA、RVALID等相關訊號傳送出去。在Read\_data狀態時，則會透過count來計數傳送出去的read data個數，當count和ARLEN相等且收到ARVALID訊號時，則會拉起RLAST訊號。再待R訊號握手且RLAST訊號為一時，則可在下個clk回到初始狀態。

%3CmxGraphModel%3E%3Croot%3E%3CmxCell%20id%3D%220%22%2F%3E%3CmxCell%20id%3D%221%22%20parent%3D%220%22%2F%3E%3CmxCell%20id%3D%222%22%20value%3D%22%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3BfillColor%3D%23fff2cc%3BstrokeColor%3D%23d6b656%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22140%22%20y%3D%2215%22%20width%3D%22670%22%20height%3D%22940%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3C%2Froot%3E%3C%2FmxGraphModel%3E

(f)DRAM wrapper的狀態機

說明:

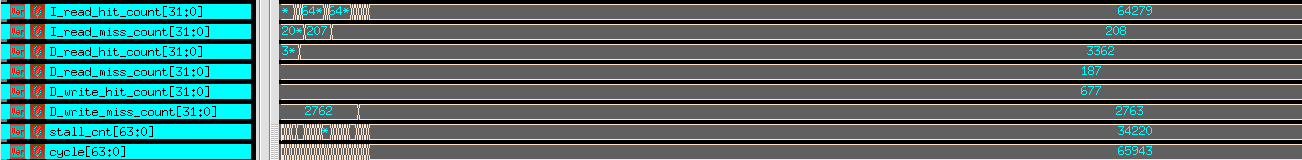
上圖為DRAM wrapper中的控制狀態機，此狀態機分為讀和寫兩部分。在電路初始化時，會先進入Idle狀態。在Idle狀態時，等待握手並判判斷是否發生row hit，以進入對應的狀態，在吃到握手訊號時，把A存入暫存器中以確保地址能穩定。

若在Idle狀態吃到AR握手且not row hit的訊號，則會進入Read\_Act狀態，由於DRAM中給予地址的方式是先給Row address(A[22:12])再給Column address(A[11:2])，所以在此狀態會先將RASn設為零，將Row address的部分傳送至DRAM。接著進入count狀態，此狀態的目的在於模擬DRAM實際在讀取時，會有相較SRAM大的延遲，在此狀態會開始計數五個clk，再來就能進入Read\_col\_addr狀態。在Read\_col\_addr狀態會將CASn設為零，以將Column address的部分傳送給DRAM，由於我們的系統有採取busrt mode，所以還需要一個Readcnt來計數讀取到的資料數，並根據目前已讀到的資料數決定送出的Column address是否需要修正(即A\_reg[11:2]+count)，接著待R訊號握手後，即可進入burst狀態。在burst狀態會判斷Readcnt是否有計數到足夠的資料數，若還沒達到burst length的要求，則會回到count狀態重複讀取資料，直到滿足burst length的要求。

若在Idle狀態吃到row hit 且AR握手，(row hit發生即代表ARADDR的row address部分與我們存在A\_reg的row address部分相同)，若相同則可直接進入Read\_col\_addr狀態，這樣就不需要再傳送一次一樣的row address，以節省clk cycle數量。在burst狀態滿足了burst length的要求後，則會在下個clk進入precharge狀態，在此狀態會將RASn設為零、將Wen設為4’b0000、並給予A\_reg中row address的部分(因在precharge狀態中，需給予和剛剛讀取地址一樣的Row address)，接著會在下個clk進入count狀態，計數五個clk後，再回到Idle狀態。

若在Idle狀態時，吃到AW握手且not row hit的訊號，則會進入Write\_Act狀態。在Write\_Act狀態中，會將RASn設為零，以將Row address的部分傳給DRAM，接著進入count狀態，待計數五個clk後再進入Write\_col\_addr狀態。在Write\_col\_addr狀態時，會將CASn設為零，以將Column address的部分傳去DRAM，並將Wen設為WSTRB的數值(WSTRB會根據要sw、sh、sb而變化)，同時將D設為WDATA。接著在下個clk會進入Write\_data狀態。在此狀態會等待W訊號握手後再進入count狀態。此處有count狀態的原因在於WDATA給予後，需等待DRAM五個clk後才會將數值成功寫入，所以需要有count狀態，才能進入BRESP狀態。在進入BRESP狀態後會等待B訊號握手後再進入precharge狀態。在precharge狀態中，會給予對應的訊號，接著再進入count狀態，最後回到Idle。若在Idle狀態吃到AW握手且row hit的訊號，則可直接進入Write\_col\_addr狀態，直接輸出Column address的部分，不須再給予一次相同的row address，如此就能減少clk cycle的數量。

1. **Hit rate and IPC**
2. **Prog0**

****

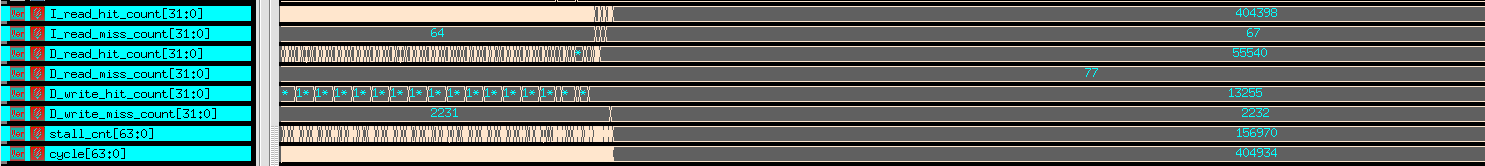
Inst. cache read hit rate:

Data cache read hit rate:

Data cache write hit rate:

IPC (Instruction per cycle):

1. **Prog1**

****

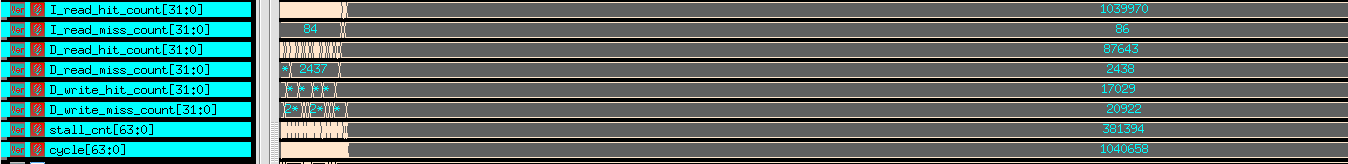
Inst. cache read hit rate:

Data cache read hit rate:

Data cache write hit rate:

IPC (Instruction per cycle):

1. **Prog2**

****

Inst. cache read hit rate:

Data cache read hit rate:

Data cache write hit rate:

IPC (Instruction per cycle):

1. **Prog3**

****

Inst. cache read hit rate:

Data cache read hit rate:

Data cache write hit rate:

IPC (Instruction per cycle):

**7.C code explanation**

一張含有 文字 的圖片

自動產生的描述(a)booting

說明:

booting的程式是存放在ROM當中，其目的在於將DRAM的資料搬到IM和DM當中。第一個for迴圈是使用global pointer的方式將DRAM中的資料搬運到IM當中，使用i計數，直到DRAM中所有要傳給IM的資料都搬運完畢後，即會跳出迴圈。第二和第三個for迴圈也是使用相同的方式計數，將DRAM中要傳給DM的資料都搬運完後，則會跳出迴圈。

一張含有 文字 的圖片

自動產生的描述(b)prog1

說明:

Prog1是half-word的排序演算法。最外層的for迴圈是計算現在選到的是array中的哪個數值，接著將insert設為零，代表資料還沒被insert過，接著進入第二個for迴圈，在此迴圈中，test\_comp會從零開始比較到剛在最外層迴圈中所選到的數值，若在比較過程中發現有數值比目前已排列的數列中的數值小，則會進入第三層for迴圈排序資料，並將剛比較到的資料加入至數列並排序完成，接著會將insert設成1，代表數值有被insert過。

一張含有 文字 的圖片

自動產生的描述(c)prog2

說明:

Prog2是將一BMP的圖檔從彩色轉灰階。Byte\_size0、Byte\_size1、Byte\_size2、Byte\_size3用來抓取BMP file中的資料，之後再將這四筆資料移到對應的位置後，組合成一個word(稱bmp\_byte\_size)，接著再利用for迴圈從int=0~54將header讀取過一次存放在test\_start，之後在下一個for迴圈中將 i設為從54開始累加三(i從54開始代表從點陣圖資料開始)，直到i小於剛剛組成的bmp byte size中所存放的數值(bmp\_byte\_size存放的是點陣圖的大小)，接著再判斷藍、綠、紅是否都為ff如果是則此像素無須多做計算，若將藍、綠、紅都不是ff，則會將藍、綠、紅做計算，最後將計算完的結果存回，直到將整張圖片計算完成。

一張含有 文字 的圖片

自動產生的描述(d)prog3

說明:

Prog3是執行矩陣乘法，由data.s的檔案引入array size i/j/k，最後計算完的結果為兩個short相乘後，再存為integer，若i=8、j=4、k=3，要計算的矩陣是將一個A(83)和B(34)的二維矩陣相乘，最後產生的矩陣會是一個C(84)的矩陣。在C code中最外層的for迴圈是用i做counter，因考慮到矩陣乘法的先後順序，擺放的順序是為先放row再放column，所以會將i(row)的迴圈放在外面，將每次要變動的j(column)放在裡面，將A\_data、B\_data、result初始化後，再進入到下個for迴圈，其中k是資料要相乘相加的次數，A data的計算是將起始位置加上每次進來的i再乘上k的大小(選擇進來的行是哪一行)，最後再加上k(最裡面for迴圈這次跑到哪一個值)。而B data的計算類似，差別在於取資料是從第24筆資料開始計算(起始位址加上A矩陣應該拿完的資料後的位置開始拿資料)，待迴圈跑完後，再放到最後的矩陣中即為運算的結果。

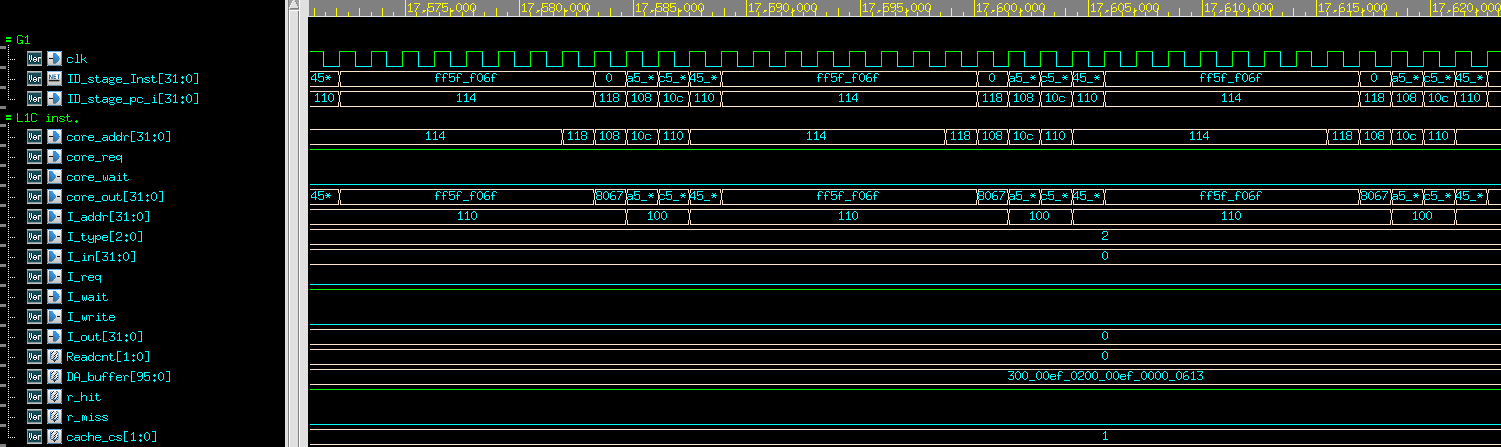
**8.Waveform**



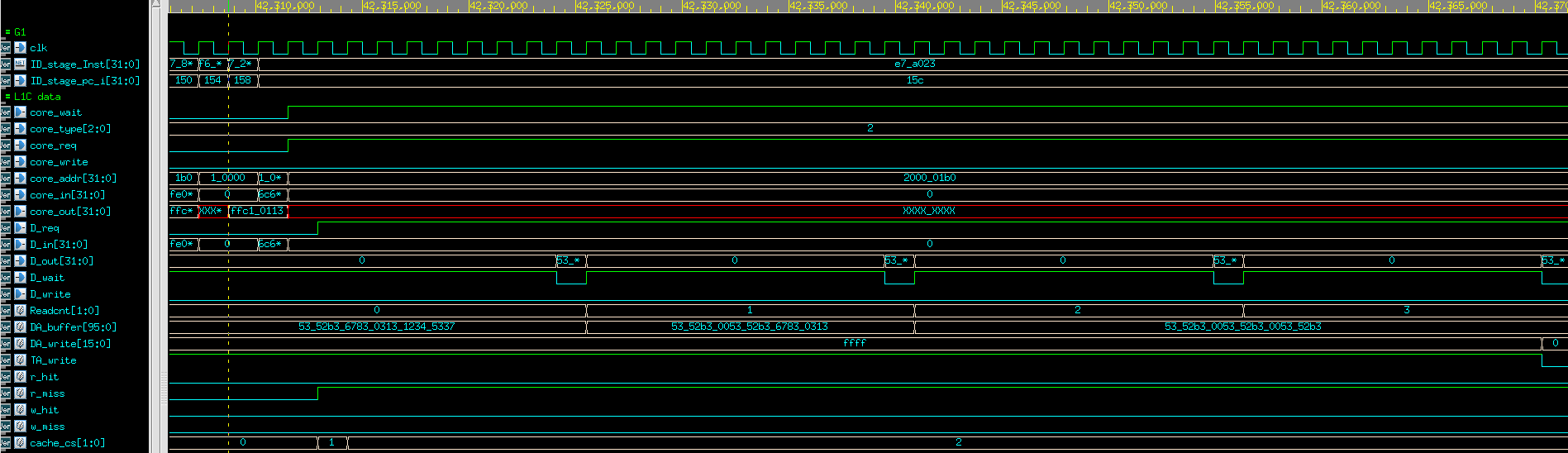
**一張含有 文字, 室內, 黑色, 螢幕擷取畫面 的圖片

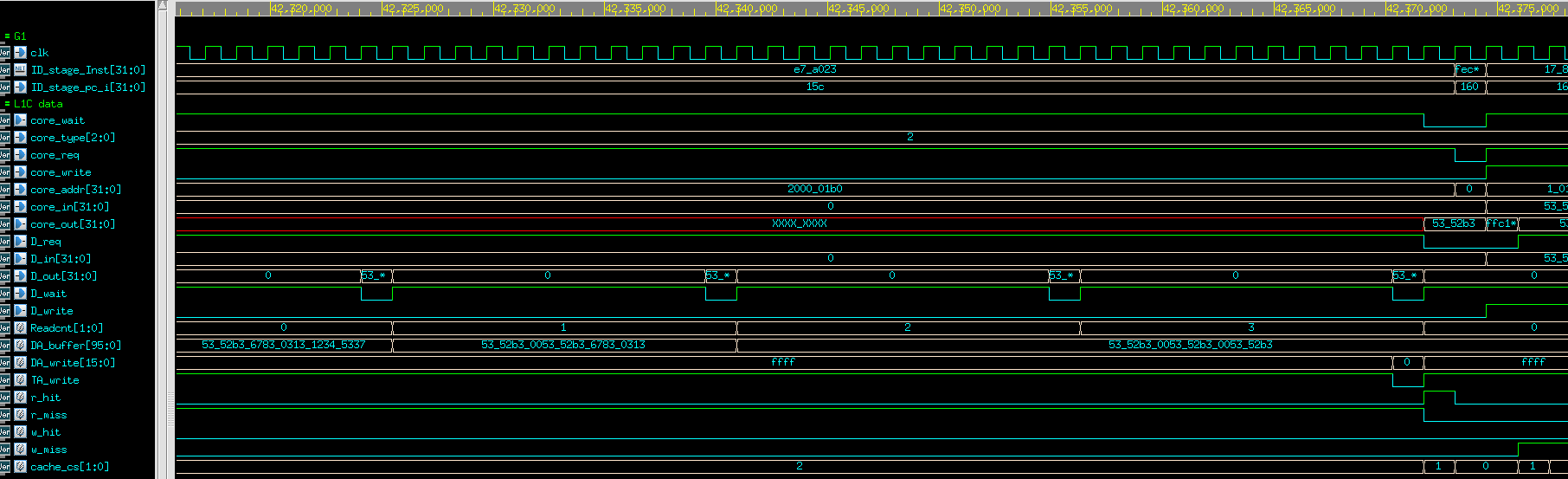
自動產生的描述(a)Inst. cache read miss**

說明:上圖為Inst. cache 發生read miss的狀況。由CPU傳送core\_addr給快取時，發生了r\_miss的情況(紅色框框)，當快取發生r\_miss時，則會直接將core\_wait拉起，以確保CPU的管線會暫停。接著快取會進入cs=2(向bus要求資料，並等到四筆指令都回來時才能回到原本的狀態)，當cs=2時，I\_out則會出現bus傳回來的四筆指令(橘色框框)，快取會同時將這四筆資料存到DA\_buffer之中。待四筆指令都存好後(I\_wait為0且Readcnt會數到3時)，則可回到cs=1，此時由於指令已經被存入TA和DA中，所以一定會發生r\_hit，並順利將剛存的指令送給CPU(藍色框框)，並將core\_wait設為零，以讓CPU繼續運作。

**(b)Inst. cache read hit**

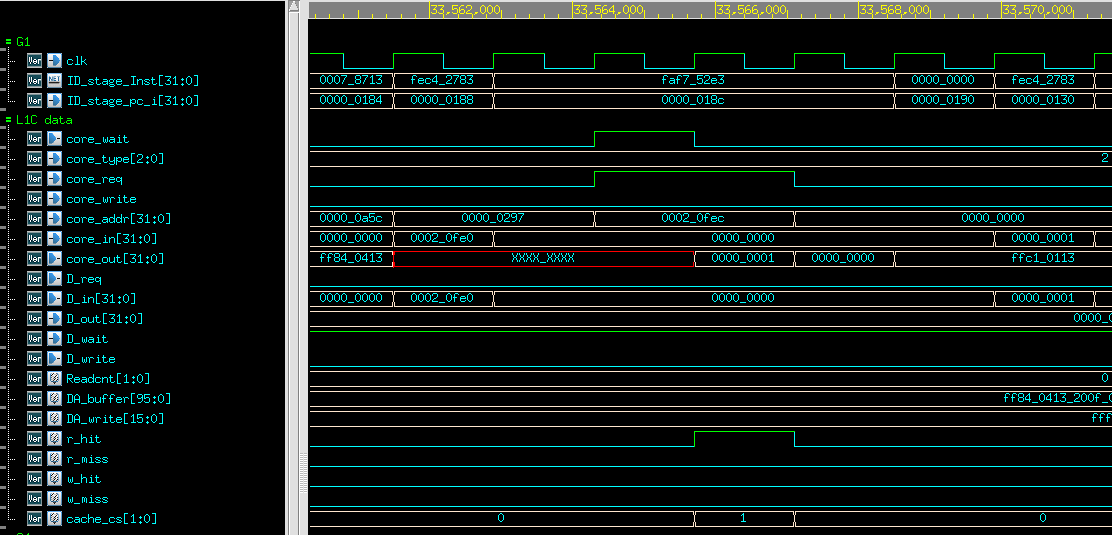
說明:上圖為Inst. cache發生read hit的狀況。由於這部分的程式執行是迴圈(橘色框框)，所以會一直發生r\_hit(紅色框框)。由於一直發生，所以快取會維持在cs=1，當吃到r\_hit訊號時，則會直接送出指令給CPU(藍色框框)。由於快取可以直接送出指令給CPU而不需要延遲clk，所以core\_wait可以一直維持零，讓CPU同時運作。





**(c)Data cache read miss**

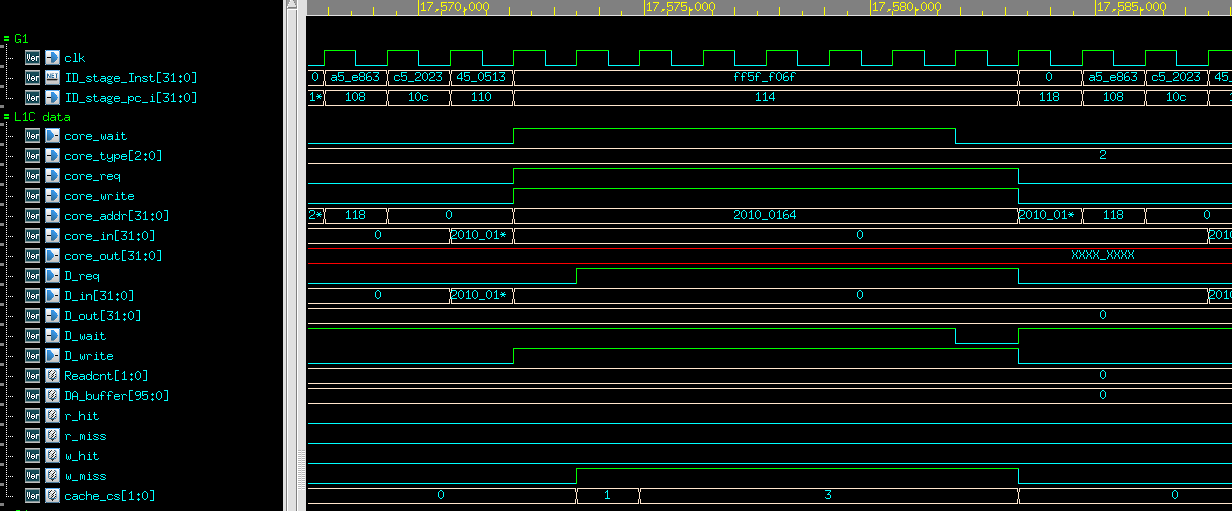
說明:上兩圖為Data cache發生read miss的情況，PC158為lw指令(紅色框框)，經過兩個clk後會發出core\_req=1和core\_write=0訊號，以告知快取需要讀取資料，接著快取會進入cs=1，並判別r\_miss發生(藍色框框)，接著快取會進入cs=2(向bus要求要讀取資料)，bus每次傳回一筆資料則將D\_wait設為零，當快取吃到D\_wait為零的訊號時，則會將數值存入buffer中，同時Readcnt會開始計數已存入了幾筆資料，當存入了三筆資料(Readcnt=3)且第四筆資料(D\_wait=0)送回來時，則會將數值寫入至DA和TA當中(紫色框框)，並在下個clk回到cs=1。由於剛讀取的資料已被寫回DA和TA中，所以此時一定會是r\_hit，當r\_hit發生時則會將core\_wait設為零(橘色框框)、將讀取的數值送給CPU，並在下個clk回到初始狀態。



(d)Data cache read hit

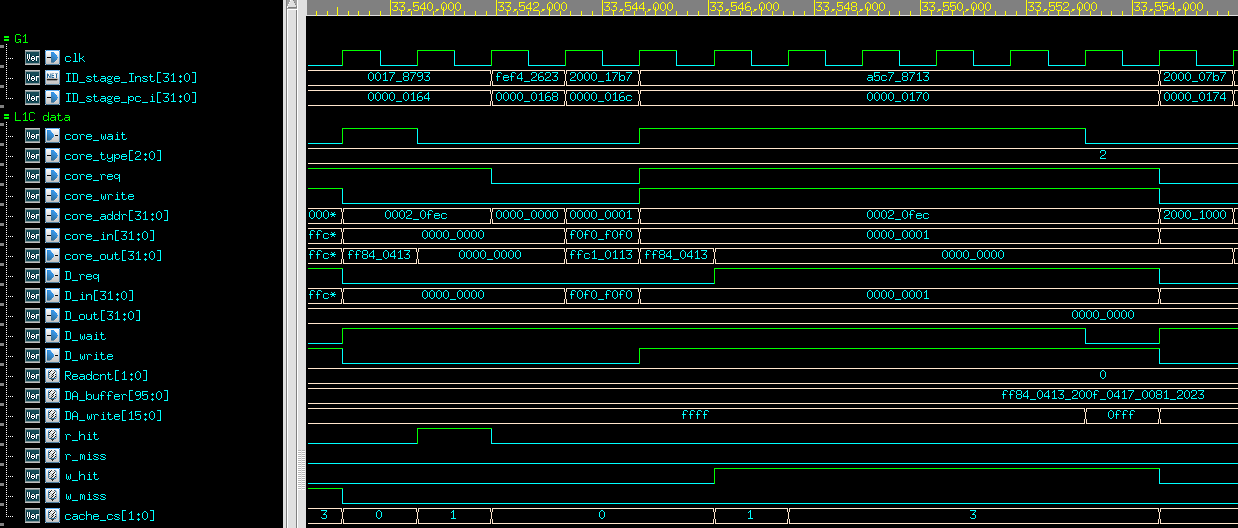
說明:

上圖為Data cache發生read hit的狀況。fec4\_2783為lw指令，經過兩個clk後，快取會吃到core\_req=1和core\_write=0兩個訊號，當吃到core\_req=1訊號的瞬間，快取就會發出core\_wait=1的訊號以暫停CPU中的管線，接著快取會進入cs=1狀態(判斷hit or miss)，經判斷後發生r\_hit，則可直接將CPU要讀取的數值傳給CPU，並在r\_hit的同時將core\_wait設為零，以讓CPU繼續運作。



(e)Data cache write miss

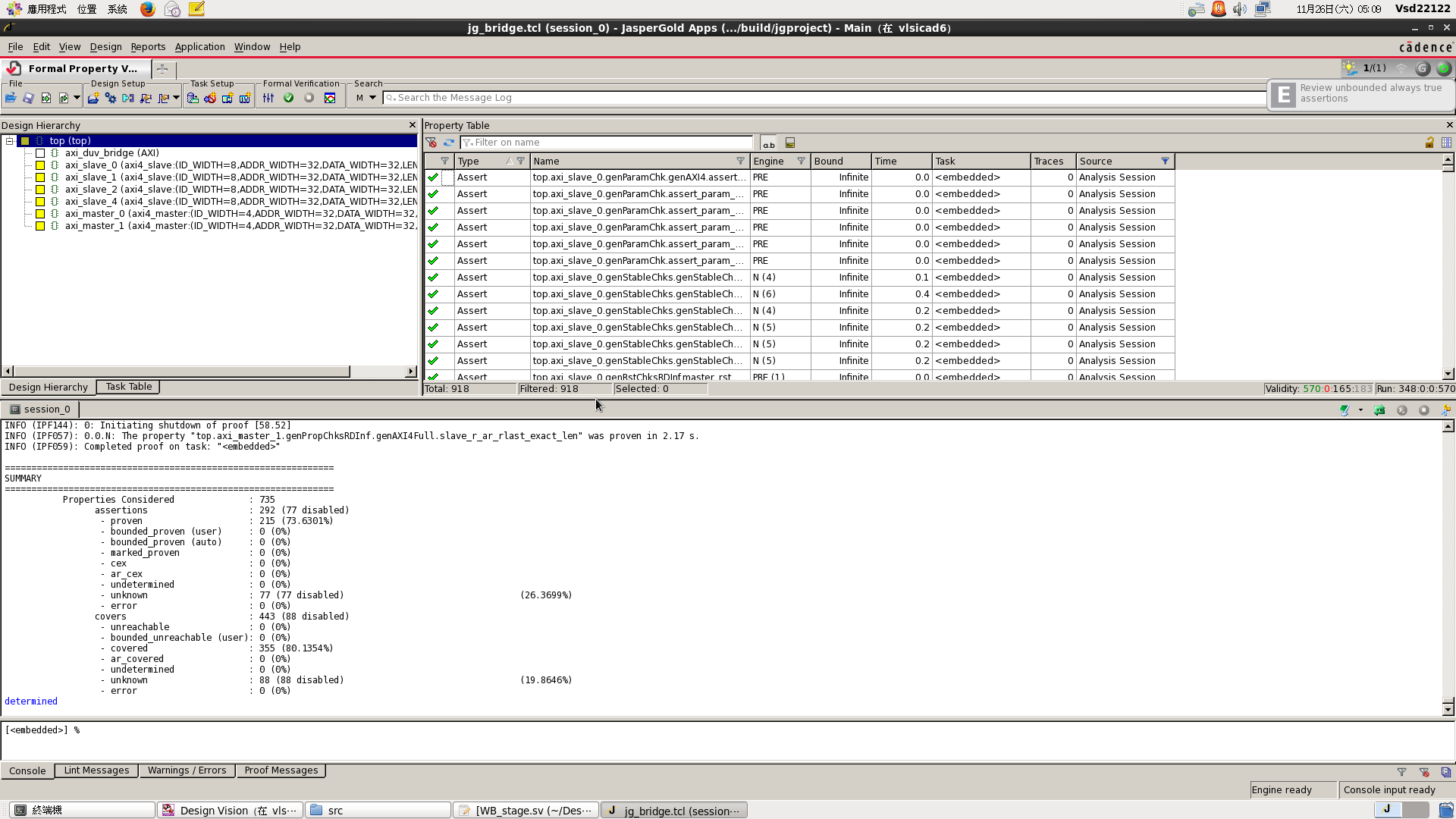
說明:上圖為Data cache發生write miss的狀況。c5\_2023是sw指令(橘色框框)，經過兩個clk後，會發現core\_req拉起(紅色框框)，因sw會在EX/MEM管線拉出core\_write的訊號(紅色框框)，所以core\_req會拉起。當快取獲得core\_req訊號後，則會進入cs=1，此時判斷為w\_miss(黃色框框)，所以需要進入cs=3(對bus發出request以將數值write through至記憶體中)。待bus對記憶體寫入數值後，則會給予D\_wait=0的訊號，告訴快取資料已成功透過bus寫入記憶體。接著快取就能回到cs=0，使core\_wait=0(藍色框框)，讓CPU繼續運作。



(f)Data cache write hit

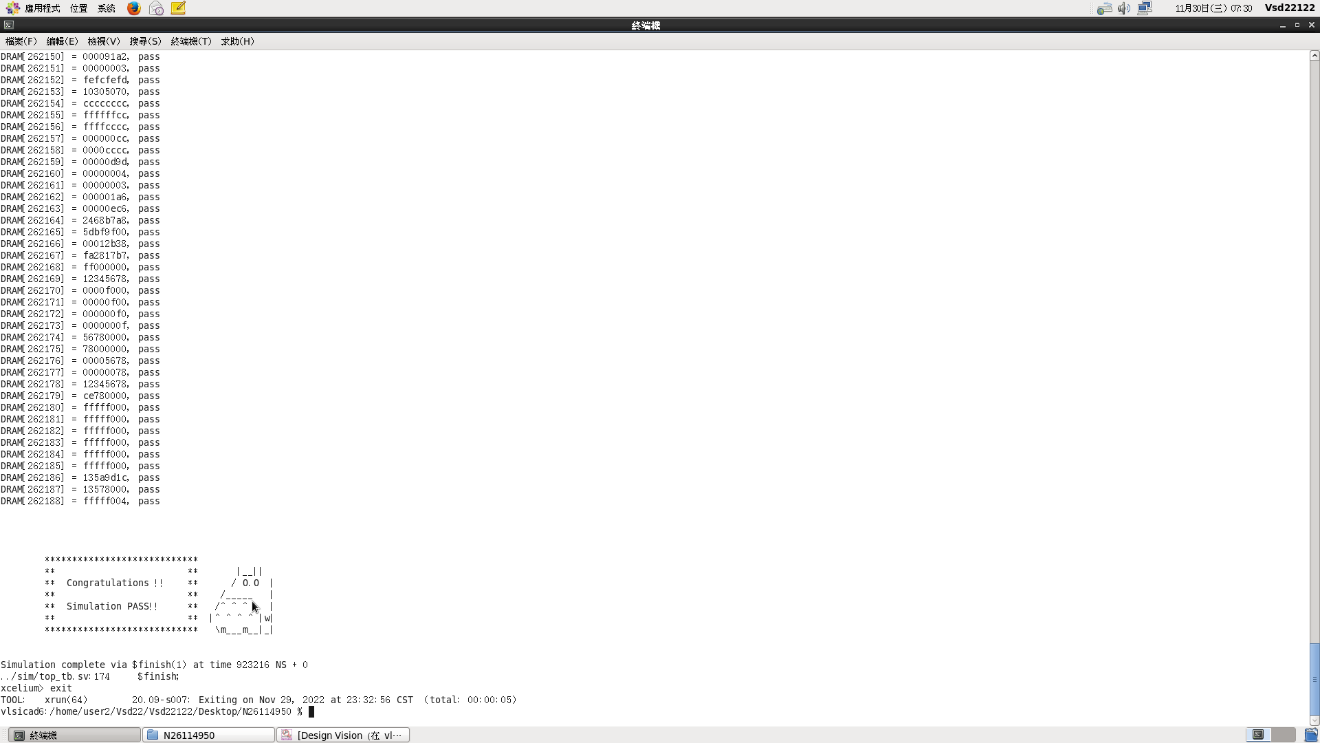
說明:上圖為Data cache發生write hit的狀況。fef4\_2623是sw指令(橘色框框)，再經過兩個clk後會發出core\_req和core\_write訊號(紅色框框)，當快取吃到這兩個訊號後，則會進入cs=1，此時快取會發現w\_hit發生(黃色框框)，但由於快取是採取write through策略，所以還是要進入cs=3(對bus發出request以將數值write through至記憶體中)，待bus結束工作後則會發出core\_wait=0訊號(藍色框框)。由於w\_hit還需要將數值寫入快取中，所以在cs=3的最後一個cycle，會將DA\_write設成0fff(白色框框，0的部分代表寫入的word位置)，以確保回到初始狀態時，數值能順便寫入DA。

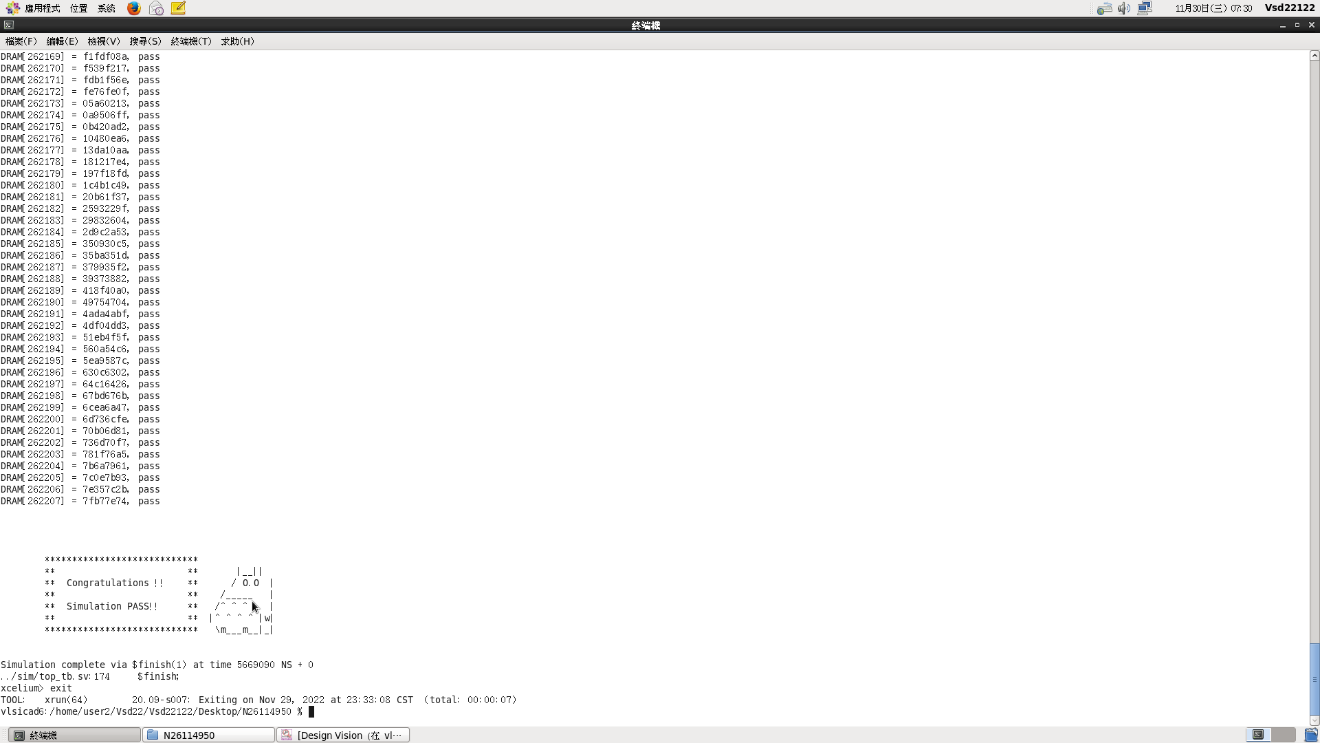
**8.AXI ABVIP**



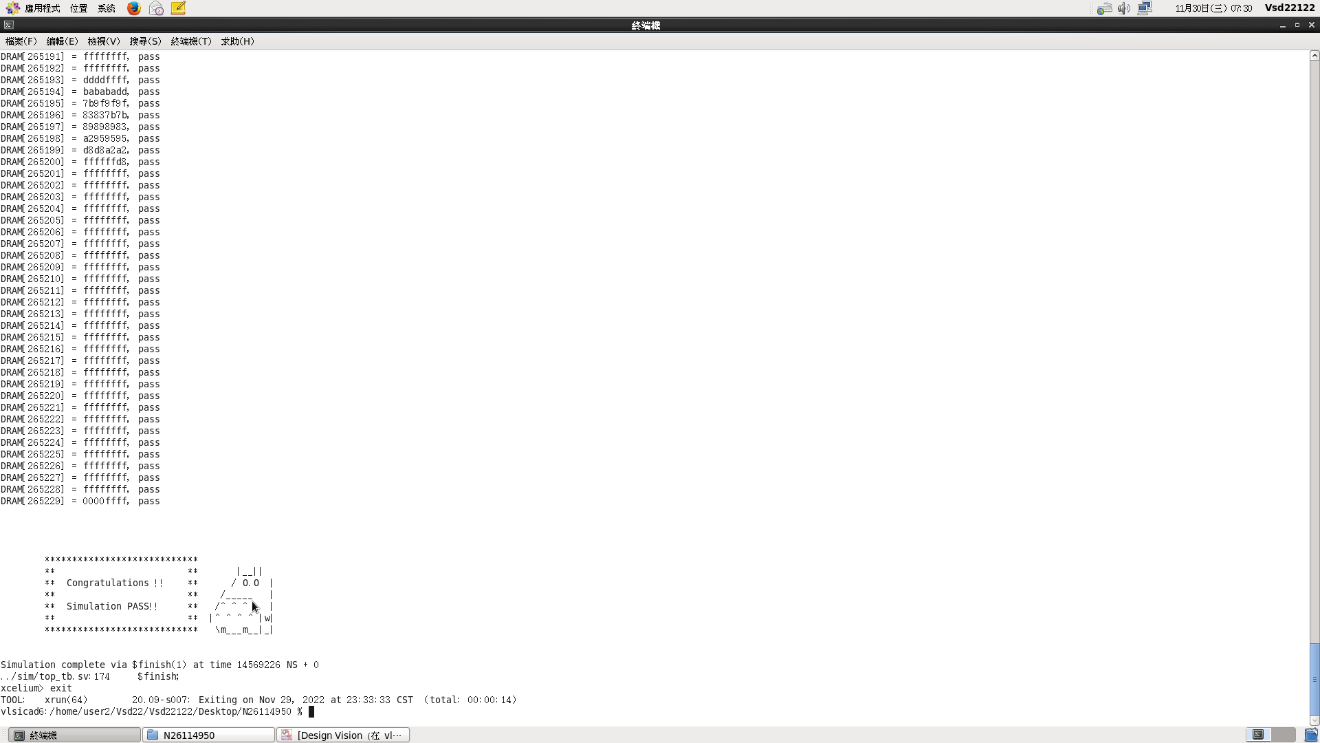
AXI的驗證結果如上圖所示，相關的assert和cover都有pass。

**9.Simulation result**

1. rtl0模擬成功
2. rtl1模擬成功



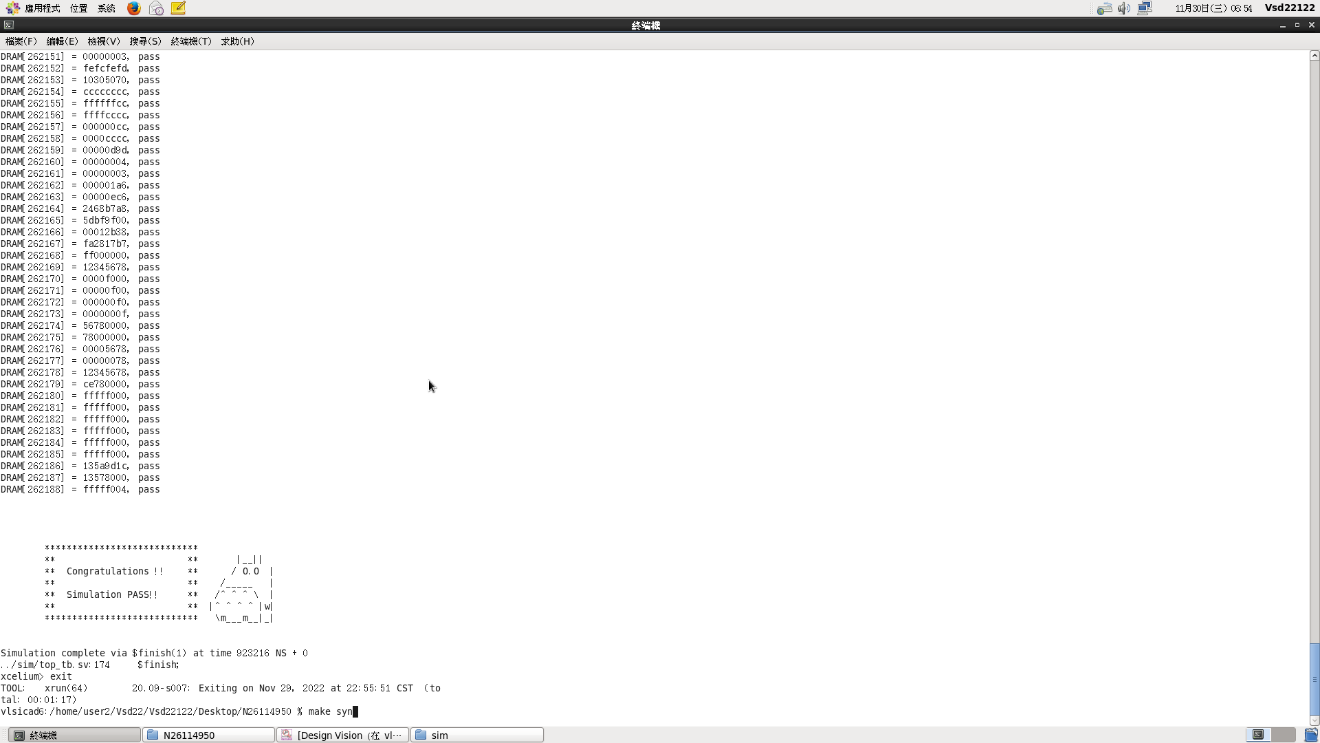
1. rtl2模擬成功



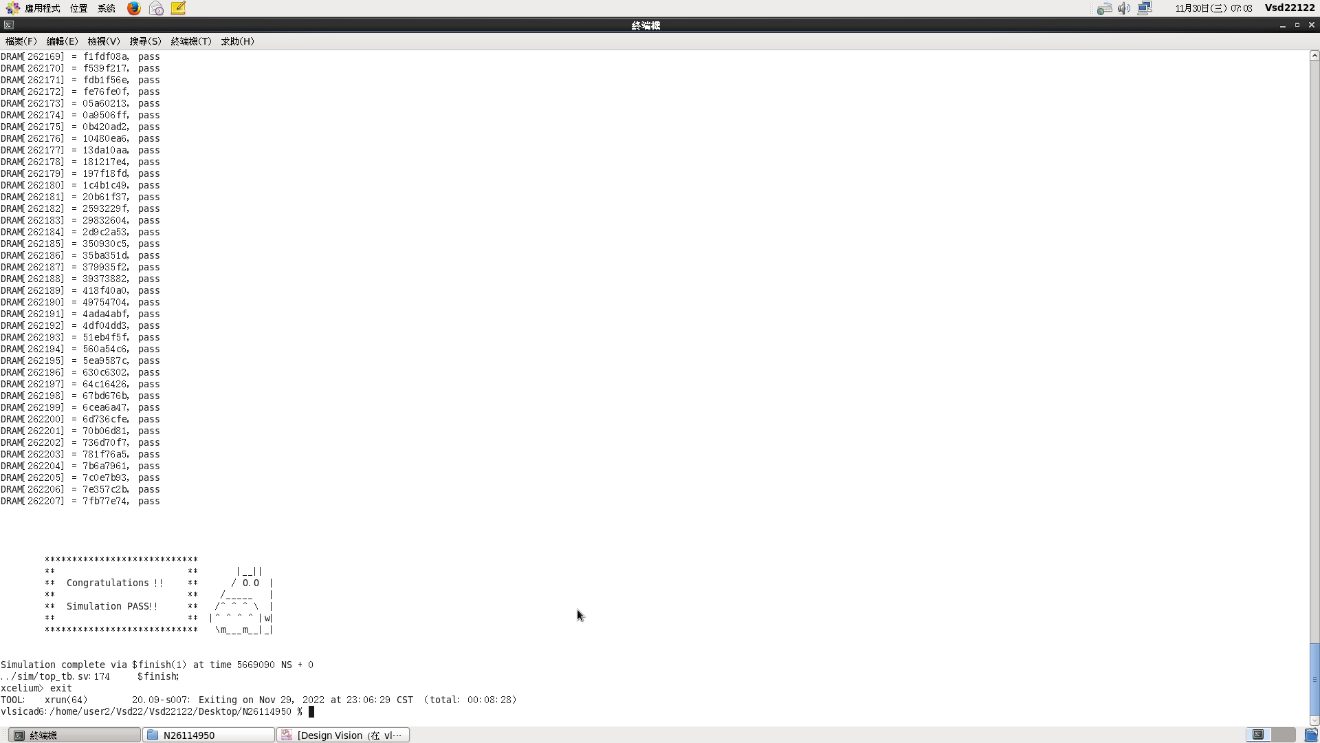
一張含有 文字 的圖片

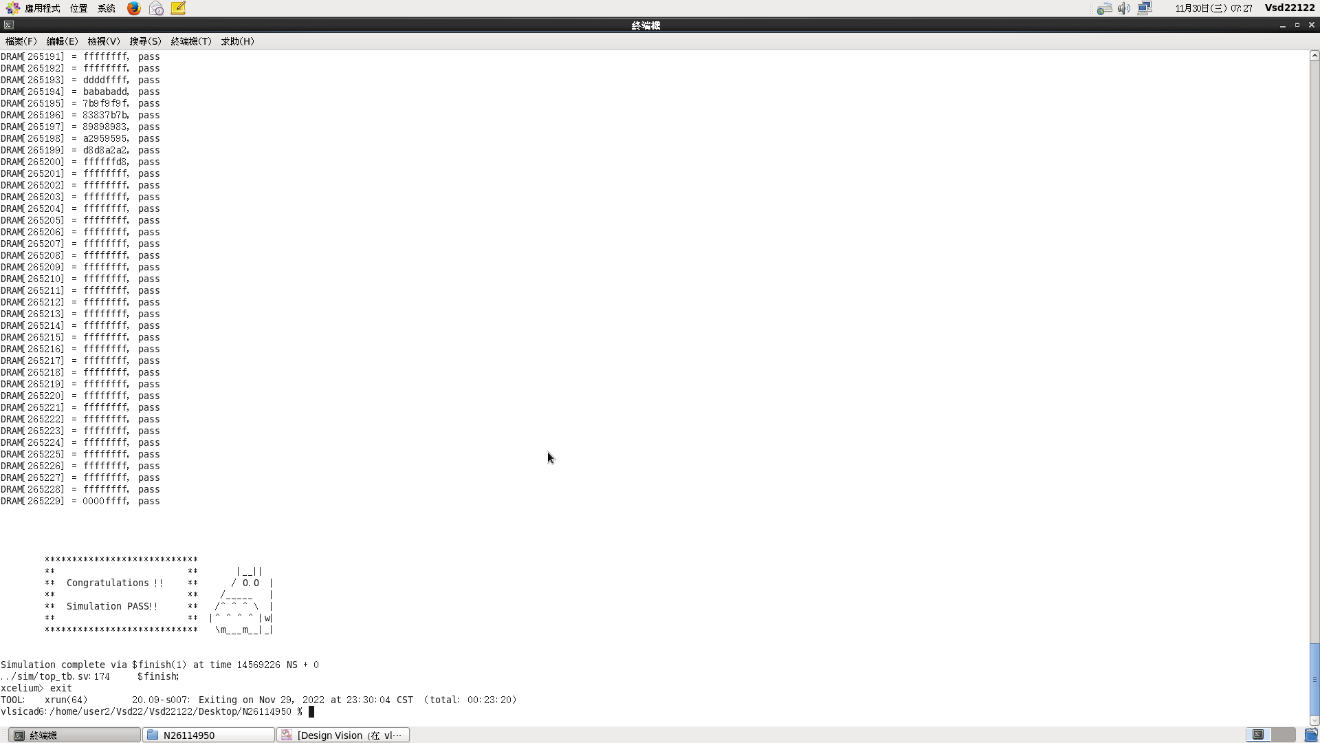
自動產生的描述(d)rtl3模擬成功

1. syn0模擬成功

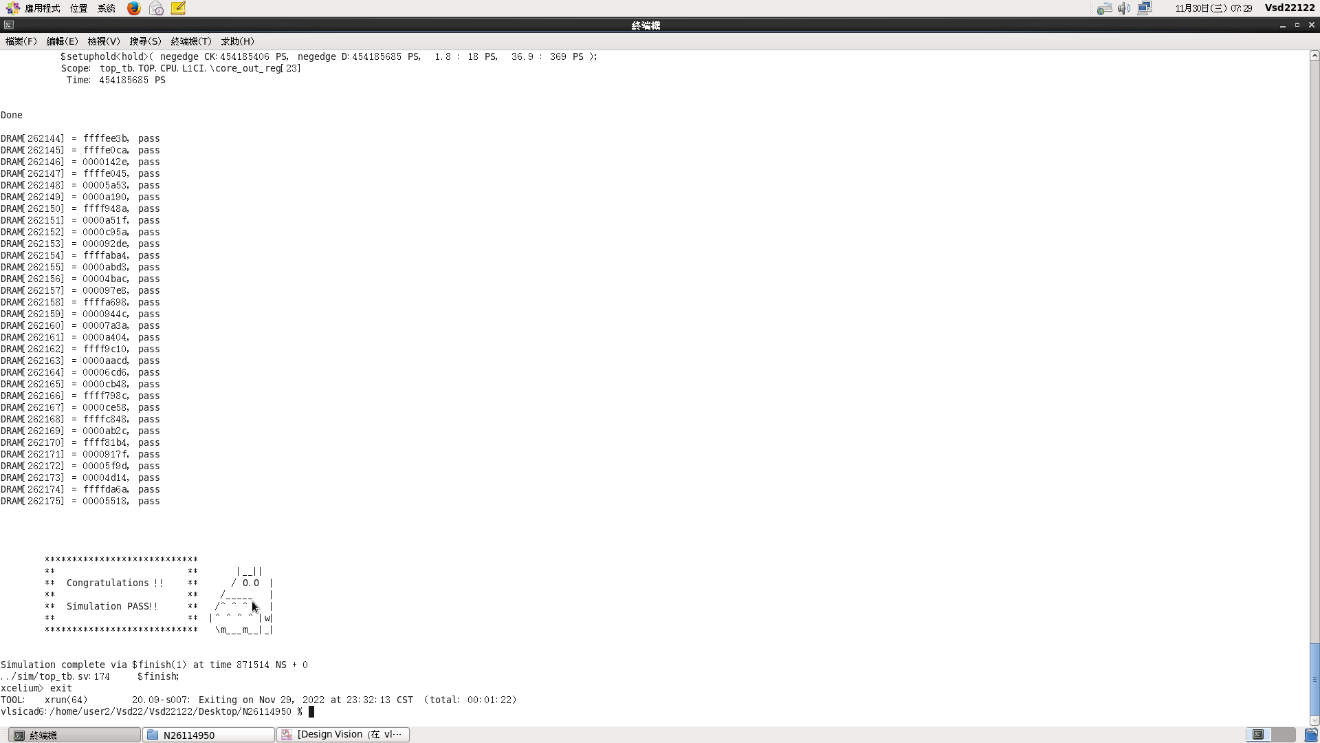


1. syn1模擬成功



1. syn2模擬成功

(g)syn3模擬成功



**10.Superlint result**

一張含有 文字 的圖片

自動產生的描述一張含有 文字 的圖片

自動產生的描述(a)總體程式行數為8832

(b)總體warning為55個

**Most frequency warning or error in supertlint**

(a)遇到case狀況未寫滿的問題。

Sol:由於這次狀態機比較多，所以有幾個狀態機漏寫到default的情況，事後有將default的狀況加上後，即可修掉這些error。

(b)case中的判斷使用到x等不可合成語法(例如3’b0xx)

Sol:我將這條判斷修成default狀態的寫法後，就能將error修掉，上網查詢後發現這種語法只有提供在rtl模擬層級。

(c)等號右邊與等號左邊的暫存器位元數大小不一致

Sol:這是因宣告時不小心而導致的warning，只要將兩邊的位元數宣告正確即可將此warning消除。

經計算後，整體superlint中的執行結果約為**99.37%**(warning不佔總體程式行數的百分率)

**11.Performance**

**一張含有 文字, 桌 的圖片

自動產生的描述**(a)area report

一張含有 桌 的圖片

自動產生的描述(b)power report

**一張含有 桌 的圖片

自動產生的描述**(c)timing report

**12.Lesson learned**

1. 快取的相關細節與設計方法

這次實作了快取，採取write through及one direct的架構。之前在上計算機組織時有接觸過這方面的知識，但真正在實作時，需要考慮更多細節和時脈的問題。透過這次作業，除了複習了快取知識的部份也讓我在實作方面有些經驗。

1. 快取對performance的影響(即Mem delay問題的應對方式)

在CPU wrapper中加上快取後，快取能夠先預存些可能會用到的指令或資料，若CPU 成功向快取要求內部已存取的資料，就能直接回傳給CPU，而不用透過bus去access SRAM或是DRAM，因為這樣會需要耗時更多的cycle 數。由於我們在存取資料時有locality的特性，若快取在CPU要執行for迴圈或 while迴圈所轉成的組合語言指令時，就能發揮更好的效果，節省大量的cycle 數。

(c)DRAM與SRAM的互補之處

DRAM和SRAM本身在存取速度和容量上就有不同，但最大的不同是SRAM的成本很高，若要使用到SRAM達到能媲美DRAM的容量，則會需要非常高額的成本。在計算機組織中有提到Memory hierarchy的觀念，將存取速度快但容量小的記憶單元放在靠近CPU端，存取速度慢但容量大的記憶單元放在最外層，則可將速度最大化，又達到節省成本的目的。

(d)Booting程式的認識與系統的行為

這次功課是第一次接觸到booting的觀念，可以把booting想像成電腦在重新開機時，需要先去執行存放在ROM BIOS的指令，才能夠順利開機，以將資料搬移到正確的地方。之前幾次功課，助教都是透過Tb直接幫我們把對應的資料吃進去，但這算是很理想的情況，實際是不會發生的。經過這次的功課後，整體已經開始有系統的感覺，雖然跟真實情況的booting比較起來只能算是初步認識，但也讓我對這方面有些許認識。