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: \_\_\_\_黃冠予\_\_\_\_\_ \_\_\_\_\_俞杉麒\_\_\_\_\_\_

Student ID: \_\_\_M16111064\_\_\_\_ \_\_\_\_P78111519\_\_\_\_\_

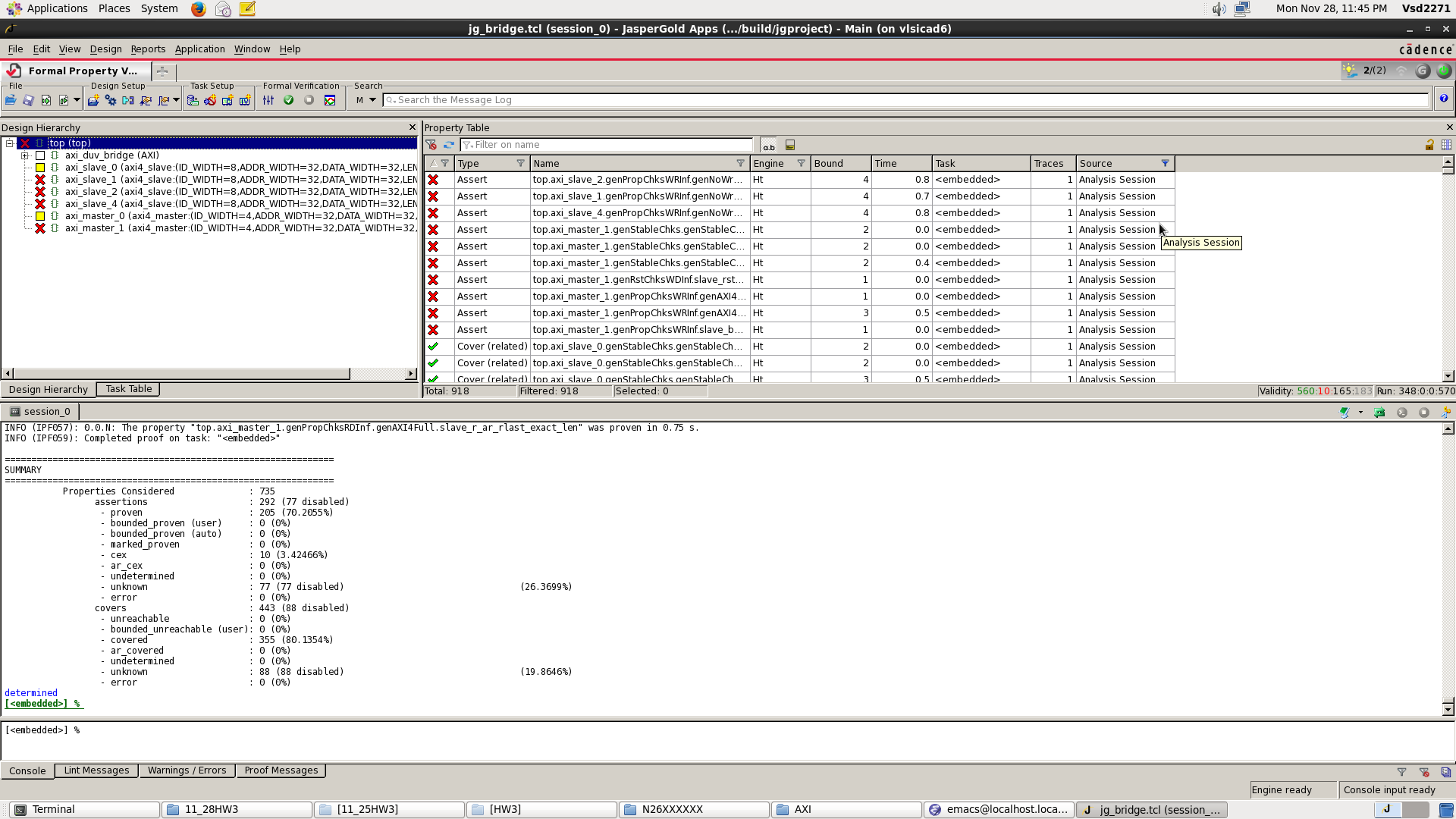
1. Summary
2. Performance

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prog0** | **Prog1** | **Prog2** | **Prog3** | **Area** |
| 2983us | 34539520ns | 162179400ns | 5551720ns | 5694265.8640 |

1. Simulation

|  |  |  |  |
| --- | --- | --- | --- |
| Prog0 | Prog1 | Prog2 | Prog3 |
| pass | pass | pass | pass |
| Syn0 | Syn1 | Syn2 | Syn3 |
| pass | pass | pass | pass |

1. JasperGold



1. 貢獻度

|  |  |
| --- | --- |
| 黃冠予 | 俞杉麒 |
| M16111064 | P78111519 |
| 50% | 50% |

1. IPC, Hit rate, Miss rate
2. Prog0
3. I-cache read: 73524

I-cache hit: 73314

I hit rate: 99.71%

1. D-cache read: 7468

D-cache hit: 4517

D hit rate: 60.48%

1. IPC: 0.702
2. Prog1
3. I-cache read: 863087

I-cache hit: 863007

I hit rate: 99.99%

1. D-cache read: 159132

D-cache hit: 156672

D hit rate: 98.45%

1. IPC: 0.799
2. Prog2
3. I-cache read: 4053944

I-cache hit: 4053836

I hit rate: 99.99%

1. D-cache read: 617987

D-cache hit: 594518

D hit rate: 96.20%

1. IPC: 0.776
2. Prog3
3. I-cache read: 138382

I-cache hit: 138300

I hit rate: 99.94%

1. D-cache read: 17747

D-cache hit: 15425

D hit rate: 86.92%

1. IPC: 0.654
2. Lesson Learned

在這次作業中，學到了如何去設計Cache，了解從CPU傳進Cache的Address該如何劃分，Cache中Valid，Tag，Data之間是如何運作，譬如如何判定Hit，Miss，以及發生Hit或Miss該發出何種訊號等等。此次Cache設計的size為1KB , Line Bits為128Bits , Associativity為Direct Map,並且Write Policy為Hit時使用Write Through , Miss時使用Write around.

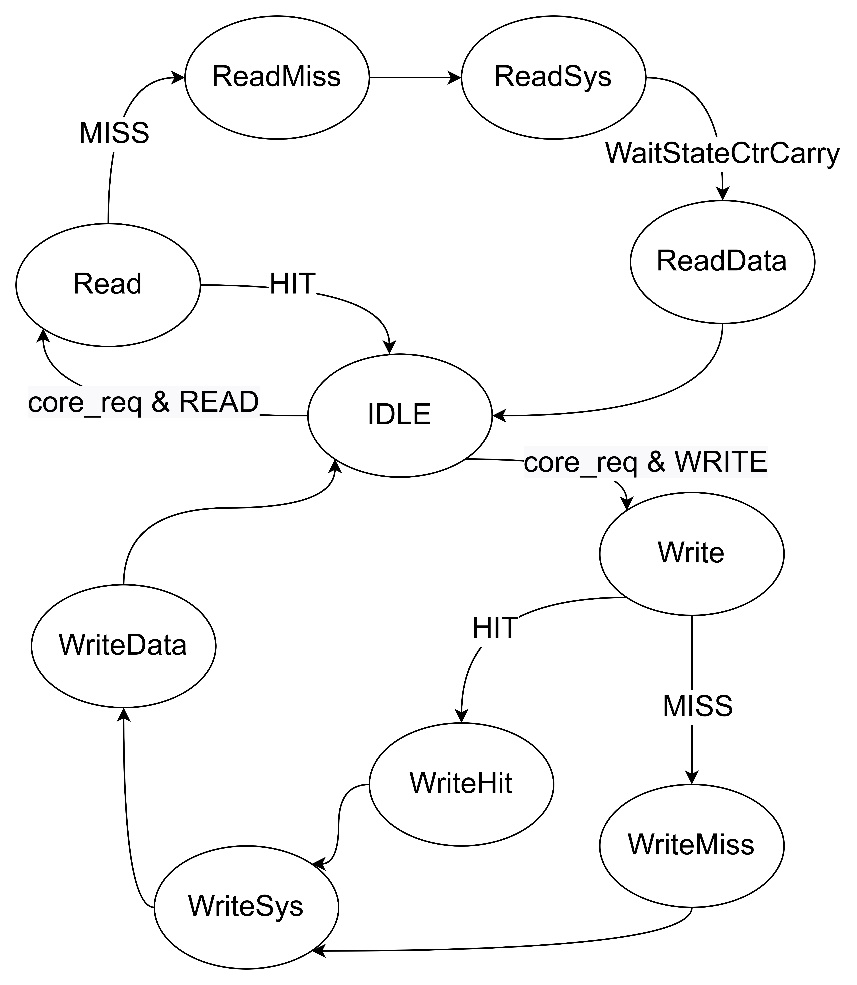
除了實做Cache以及前次作業的SRAM Wrapper，此次也實作ROM Wrapper以及配合Dram和AXI Bus相接的Dram Wrapper。而ROM Wrapper運作原理和SRAM Wrapper 相似，需要多注意的為Dram Wrapper，Dram的操作和SRAM有部分的差異，除了需要原本的Read/Write訊號，還需要考慮Active row, Access column address, Pre-charge，並且在每個階段都需配合Dram delay 5個clk。

1. Cache
2. Cache Spec:

一張含有 桌 的圖片

自動產生的描述

1. Cache Controller State Diagram



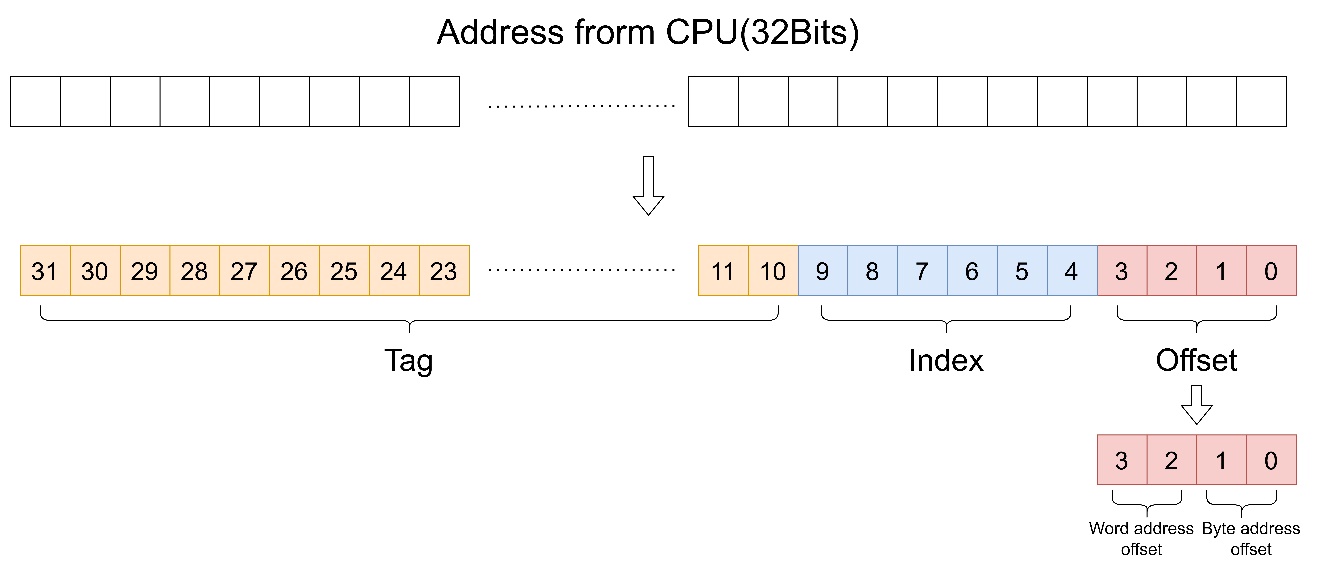
1. **IDLE**:當CPU沒有請求時，Cache狀態將會停留在此，當CPU發出Read請求時，將會進入**Read** ，反之CPU發出Write請求時，將進入**Write**。
2. **Read**:此狀態對比CPU發出的Address是否與Cache中所儲存的Tag一致，並且確認該Index之Valid是否為1。如皆成立則達成HIT，向CPU發出在Cache中其Index所儲存之Data，並返回狀態**IDLE**。

而若HIT未成立，則MISS將拉起，進入狀態**ReadMiss**。

1. **ReadMiss**:此狀態把來自CPU之Address放上AXI

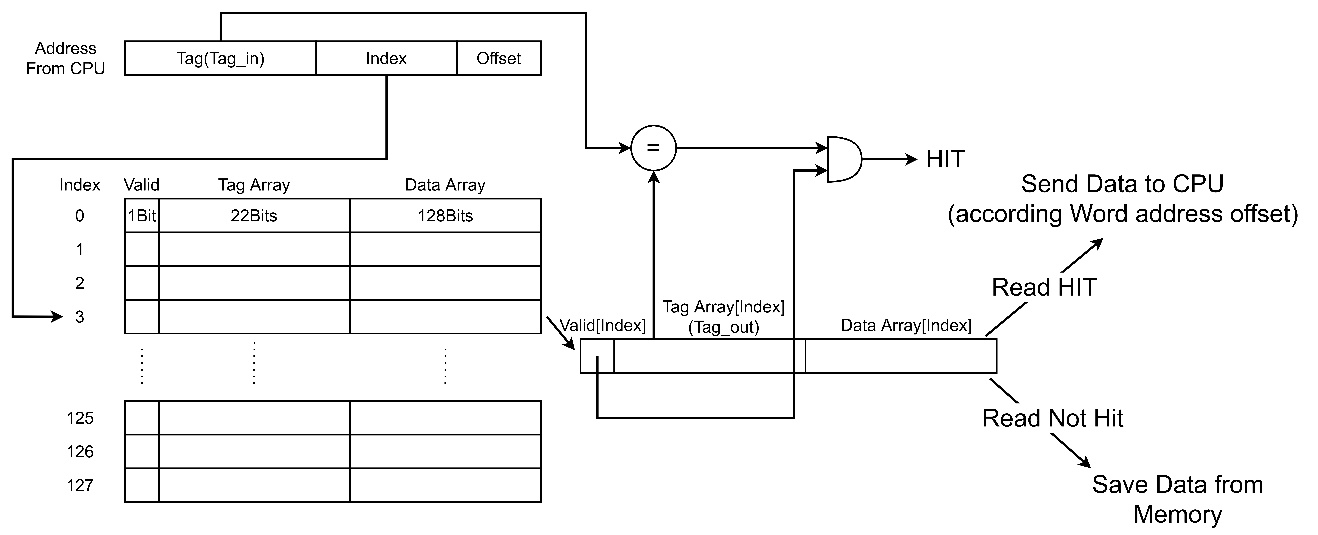
，並發出Read請求。

1. **ReadSys**:開始接收經由AXI Bus，來自Memory的資料。由於Burst模式為Incremental，且Length為3，即連續發送4筆資料，因此需要使用WaitStateCtrCarry來計算資料發送次數，當四筆資料皆發送完成並寫入Cache Data，WaitStateCtrCarry將拉起，並進入狀態**ReadData**。
2. **ReadData**:將CPU所需要的資料，從Cache Data取出並發送回CPU。
3. **Write**:此狀態如同狀態**Read**，比對是否HIT，如HIT則進入**WriteHit**，Miss則進入**WriteMiss**。
4. **WriteHit**:將CPU所發出的資料寫入Cache Data中，並將要發向AXI Bus的資料，
5. **WriteMiss**:擺上準備向AXI Bus所發送之資料。
6. **WriteSys**:對AXI Bus 發出Write 請求。
7. **WriteData**:等待資料寫入完成，如完成將返回狀態**IDLE**。
8. Address from CPU



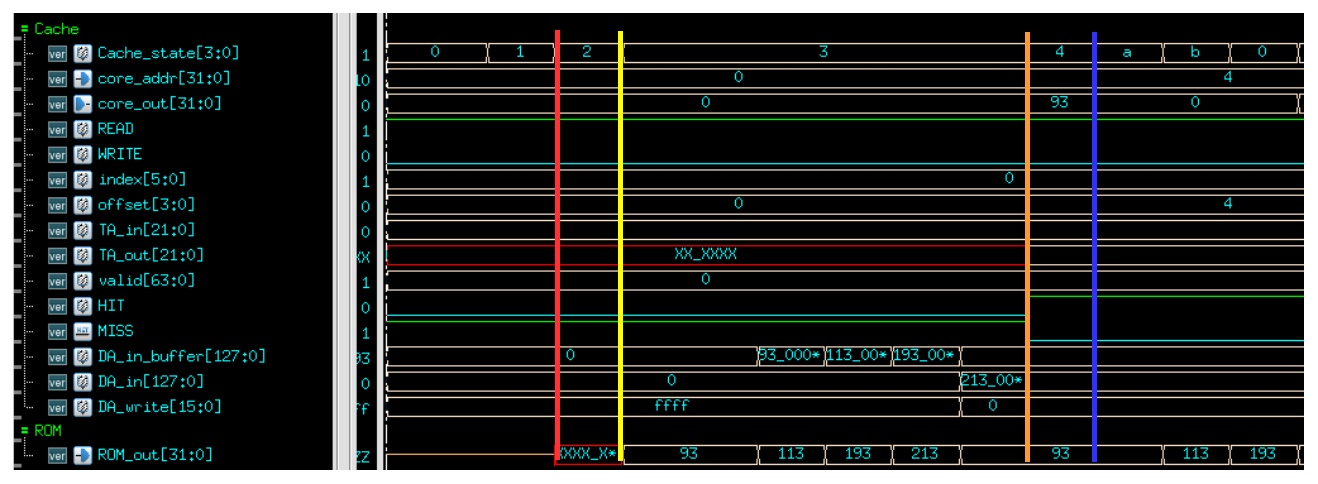
將CPU所發出的Address切分成Tag,Index,Offset，其中Tag為address[31:10],Index為address[9:4],Offset為address[3:0]，其中Offset[3:2]為Word address offset,Offset[1:0]為Byte address offset。

1. Hit



比對Tag\_in和Cache中的Tag是否一致並且Valid bit為1，成立則拉起HIT。

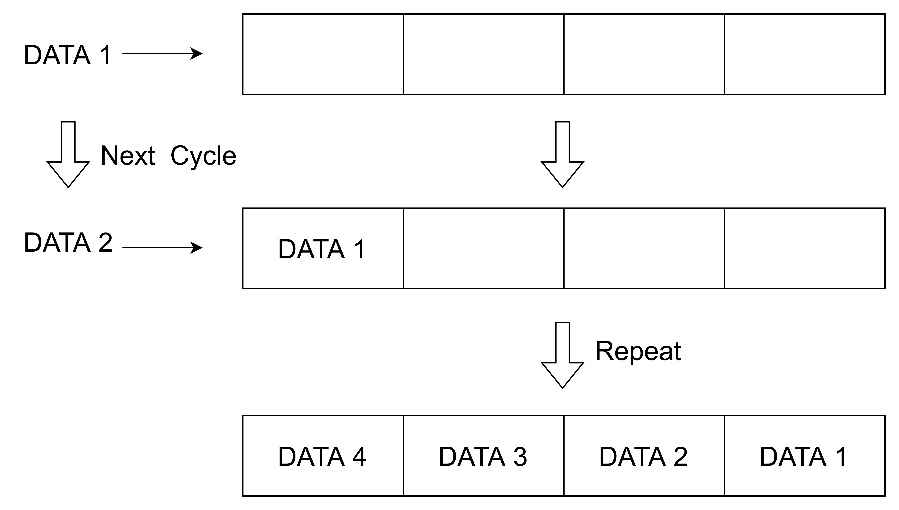
1. Cache Waveform
2. ReadMiss:



* 1. 當Cache 進入**Read**(state == 1)時，比對Tag\_in和Tag\_out是否一致，且Validbit是否拉起，此處皆不成立因此為MISS，進入狀態**ReadMiss**(state == 2)(**紅線處**)
  2. **ReadMiss**(state == 2)向AXI發送Read請求(**黃線處**)
  3. **ReadSys**(state == 3)開始接收來自AXI Bus的資料

(此處發送資料者為ROM)，在Cache中使用DA\_in\_buffer來暫時存取。(**橘線處**)

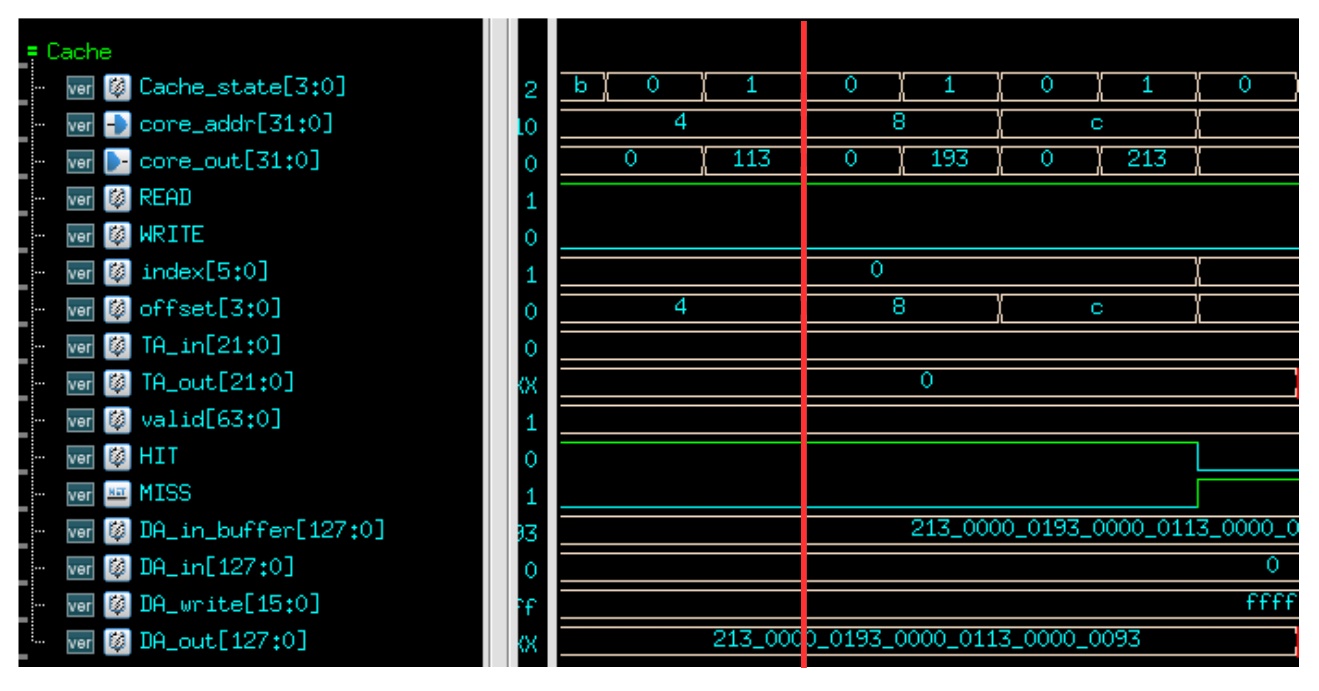
而DA\_in\_buffer使用ShiftRegister來實現，其原理如下



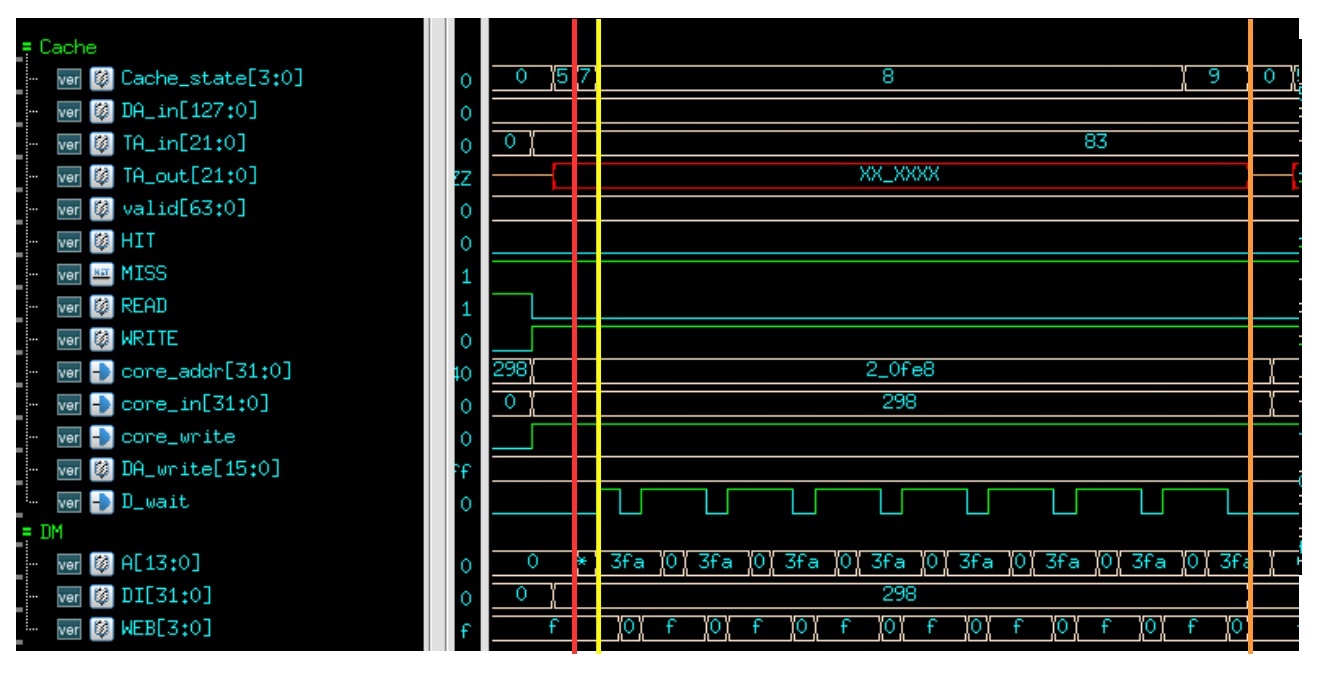
當接收完4筆資料，會將此DA\_in\_buffer傳給DA\_in並打開DATA array的Write enable，讓此資料寫入DATA array，並更新Tag array和Valid。

* 1. ReadData(state == 4)將在狀態Read(state == 1)未成功HIT的Address再次比對，由於在上一個狀態已成功存入，因此此處可以看見HIT拉起，並將該Address之值傳回CPU(core\_out)。(**藍線處**)

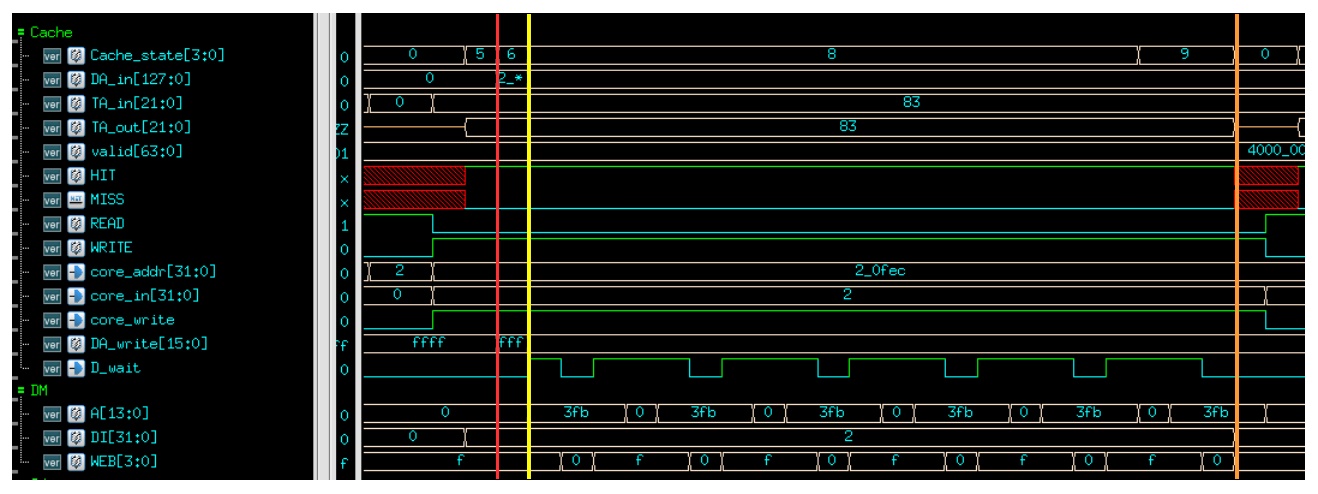
1. ReadHit:



1. 進入Read(state == 1)時，成功比對Tag\_in和Tag\_out，並且valid也為1，因此HIT拉起，而CPU所傳送的Address為4，藉由前部分對Address的說明可得知Word offset為1，因此選擇Cache DATA給出128Bits(DA\_out)中第63到32Bits的位置(0x113)輸出給CPU(core\_out)。(紅線處)
2. WriteMiss:



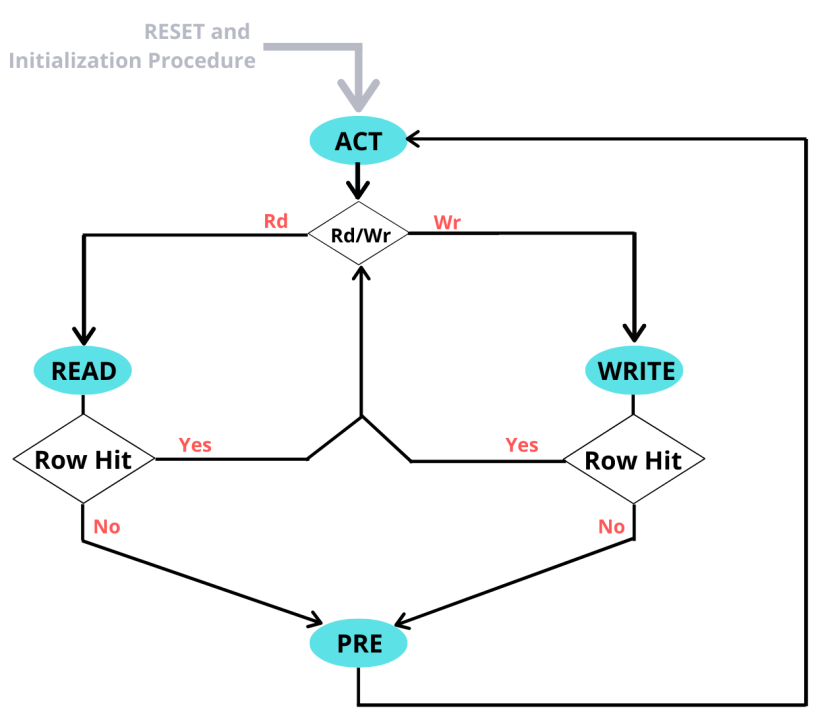
1. 如同**Read**狀態，對TA\_in和TA\_out進行比對並確認Validbit是否拉起，此處比對未成功因此MISS拉起 。(紅線處)
2. **ReadMiss**狀態，此處可以發現Cache Data的Write enable未啟動(DA\_write =ffff)，並且Cache向AXI發出Write的Request。(黃線處)
3. **WriteData**狀態(state == 9)等待AXI寫入Memory(此處寫入DM)完成，當寫入完成D\_wait會pull down，回到**IDLE**狀態。
4. WriteHit:



* + 1. 對TA\_in和TA\_out進行比對並確認Valid bit是否拉起，此處比對成功因此HIT拉起 。(紅線處)
    2. **ReadHIT(state == 6)**狀態，此處可以發現Cache Data的Write enable有啟動 (DA\_write =0xfff)，代表成功寫入Cache Data，並且Cache向AXI發出Write的Request。(黃線處)
    3. **WriteData**狀態(state == 9)等待AXI寫入Memory(此處寫入DM)完成，當寫入完成D\_wait會pull down，回到**IDLE**狀態。(橘線處)

**4. DRAM\_Wrapper**

**(1) State Diagram**

****

**i. S\_init:** When the AR channel or AW channel finishes the handshake process, we will enter the “S\_act” state from “S\_init.” This state is used to initialize signals of DRAM and wait for signals from AXI.

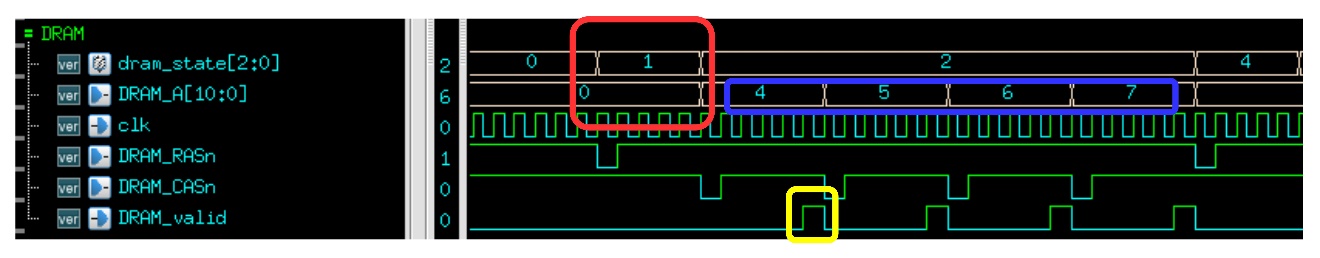
**ii.S\_act:** We will distinguish whether to read or write via the “write” and “delay\_done” from AXI. As for choosing row, DRAM\_RASn is set as 0 if delay\_cnt == 3’b0, and 1 for other scenarios.

**iii.S\_read:** We will enter this state when delay\_done == 1 and !write. Notice that we’ve already chosen the row from the previous state, and consecutive 4 data reads are placed at the same row, we choose the column by setting DRAM\_RASn as 0 if delay\_cnt == 3’b0, and 1 for other scenarios.

**iv.S\_write:** Similar to the “S\_read state.” Notice that DRAM\_Wen is determined based on write strobe and byte offset. WSTRB from the AXI is valid only when delay\_cnt == 3’b0.

**v.S\_pre:** Since precharge is required at the end, we reset DRAM\_RASn and DRAM\_Wen when delay\_cnt == 3’b0 so as to have proper operation in the next data transfer.

**(2) Waveform Analysis**

**Read Miss:**

As the figure shown above, DRAM access for read is required if cache read miss occurs. First, we select the required row at state “S\_act” as the red frame shown. Next, we pull down the DRAM\_CASn signal at state “S\_read” to send the selected column to DRAM. DRAM\_valid is then pulled high to show the read data is valid as the yellow frame shown.

4 consecutive data will be read after a read miss. Notice that 4 data are placed at the same row, we can simply obtain different value by changing the column value as the blue frame shown.

**5. ROM\_Wrapper**

We modified the architecture from SRAM\_wrapper. There will be no write activities since ROM is an abbreviation for 'read-only memory', therefore write channel is set to idle.

**6. Boot**

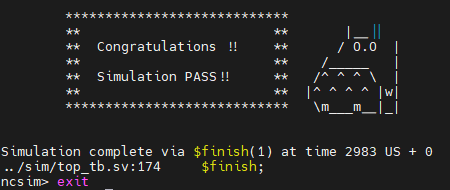
Prior to the program being executed, “boot.c” is required to transfer the data from DRAM to SRAM for the sake of instruction fetching or data accessing. At the beginning, “boot.c” is saved in ROM and instructions are stored between \_dram\_i\_end - \_dram\_i\_start. Notice that instructions should be placed after \_imem\_start so as to be correctly fetched and executed. On the other hand, data are stored at 2 places, \_sdata\_end - \_sdata\_start and \_data\_end - \_data\_start, respectively. Similarly, we place the 2 data pieces after \_\_sdata\_paddr\_start and \_\_data\_paddr\_start respectively via “boot.c”.

Loops are utilized to achieve data transfers.

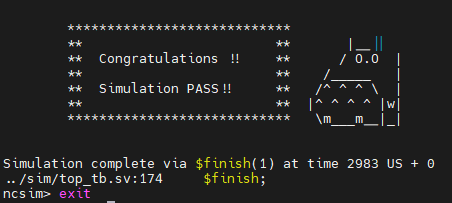
**7. Simulation**

Prog0:

rtl:

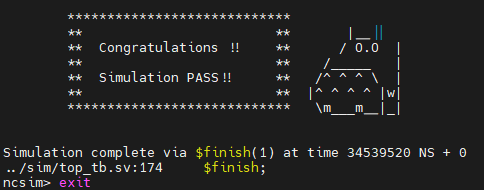


syn:

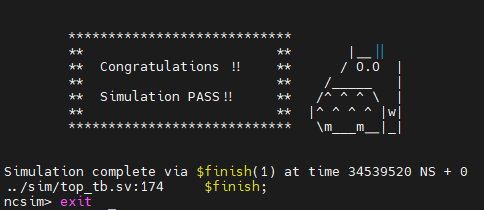


Prog1:

rtl:

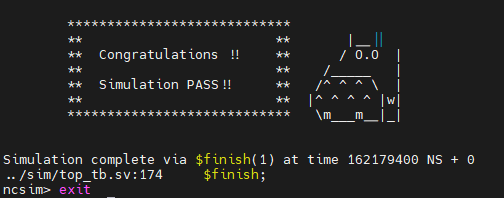


syn:

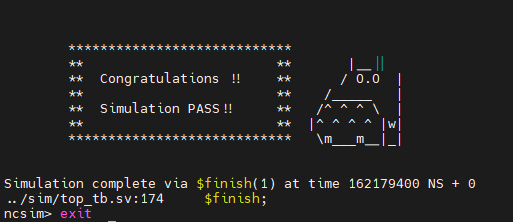


Prog2:

rtl:

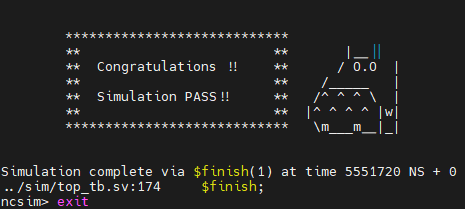


syn:



Prog3:

rtl:



syn:

