**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: \_\_黃昱澄\_\_ \_\_王昱承\_\_

Student ID: \_\_P76111238\_\_ \_\_P76111602\_\_

**Summary**

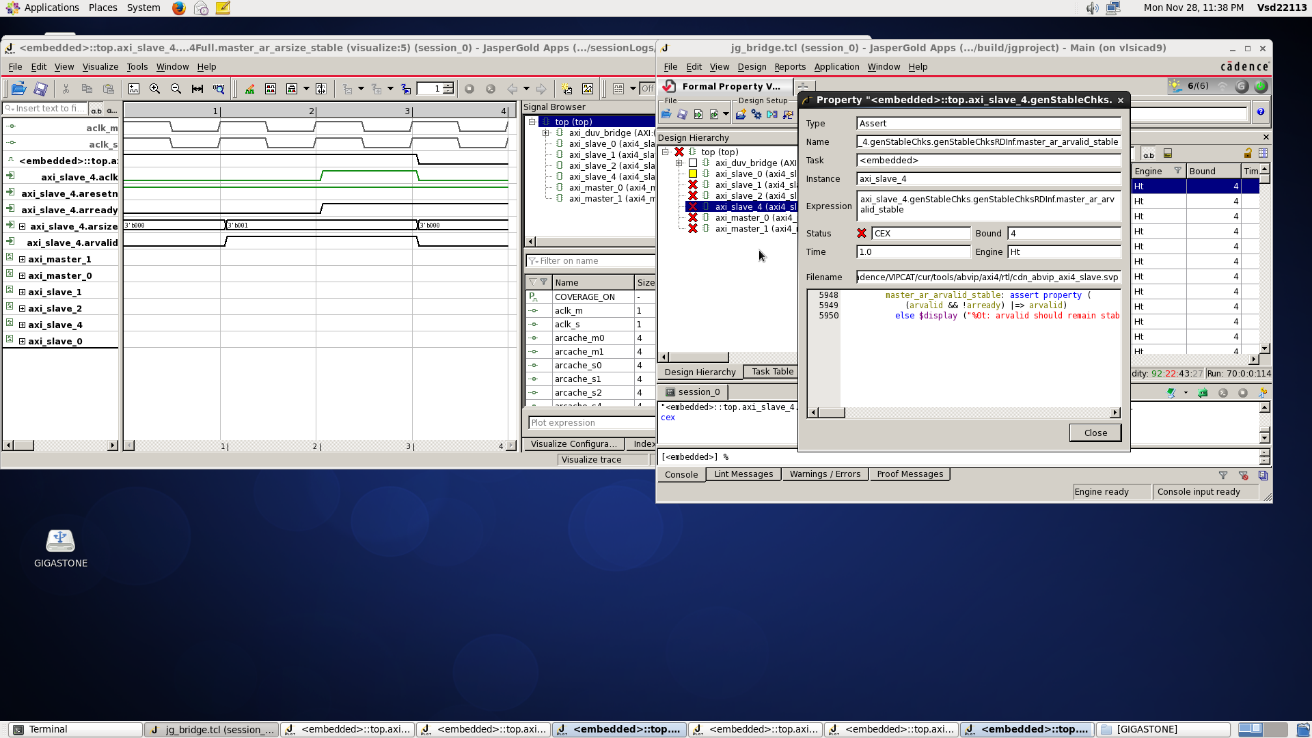
**1. Performance**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prog0** | **Prog1** | **Prog2** | **Prog3** | **Area** |
| 1551460ns | 26434220ns | 62541060ns | 2769820ns | 5670467 |

**2. Simulation**

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

**3. JasperGold**

因為我們的module都是用Interface所串起來的，所以我們在驗證時會出現下圖的狀況。就是波型都是正確，但是卻依然會報出紅字。

**4. IPC、HIT rate、MISS rate**

**1. Prog0**

(1) I-cache read : 39383

I-cache hit : 37977

I hit rate : 37977/39383 = 96.4%

(2) D-cache read : 9494

D-cache hit : 3836

D hit rate : 3836/9494 = 40.4%

(3) IPC : 0.702

**2. Prog1**

(1) I-cache read : 661073

I-cache hit : 660555

I hit rate : 660555/661073 = 99.9%

(2) D-cache read : 132758

D-cache hit : 130983

D hit rate : 130982/132758 = 98.6%

(3) IPC : 0.798

**3. Prog2**

(1) I-cache read : 1563819

I-cache hit : 1563123

I hit rate : 1563123/1563819 = 99.9%

(2) D-cache read : 408778

D-cache hit : 370198

D hit rate : 370198/408778 = 90.5%

(3) IPC : 0.775

**4. Prog3**

(1) I hit rate : 99.2%

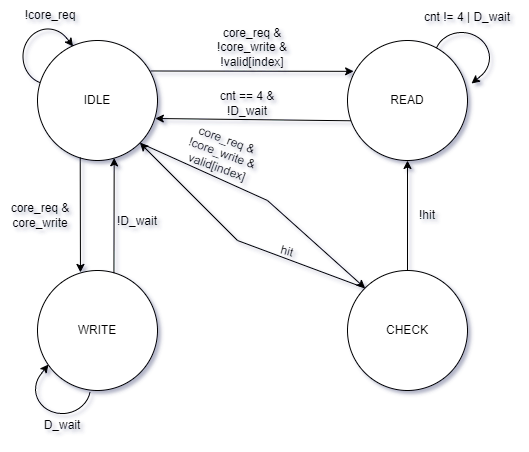
(2) D hit rate : 87%

(3) IPC : 0.654

**Lesson Learned**

**1. Cache**

**(1) State Diagram**

****

**i. IDLE :** 在這個state，會先檢查CPU是否有需要讀或是寫，如果說沒有任何要求，那麼就會一直維持在**IDLE**狀態。反之如果有需求，會先檢查是否為寫，如果是就會到**WRITE**的狀態。否則就會檢查其對應的valid值是否為1。如果valid=1, 則在**CHECK**的狀態中檢查是否成功hit。反之如果valid=0， 表示此時在cache中並沒有CPU所想要的資料，所以就會到**READ**的狀態去把資料讀出來。

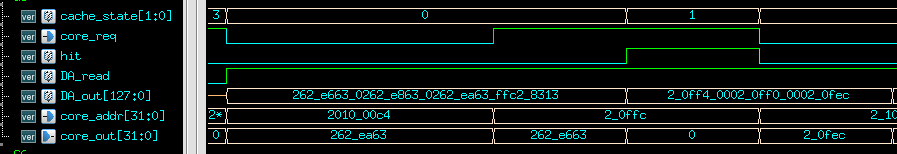
**ii. READ :** 會由cnt來決定是否讀取完成。Cache會透過Master根據位址向AXI發出並於DRAM或是SRAM中接收。接收到之後會連續讀取4筆data，再透過AXI向Cache發送data。

**iii. CHECK :** 當valid=1的時候，表示這筆資料有可能在cache中，所以需要**CHECK**是否有hit。於是便會在此階段對tag進行比對，如果hit就直接將data傳輸給CPU，反之會進到**READ**中透過AXI向RAM取值。

**iv. WRITE :** 當core\_req同時core\_write時表示要對SRAM進行資料寫入，並且同時寫入cache之中，避免cache與SRAM資料不一致。

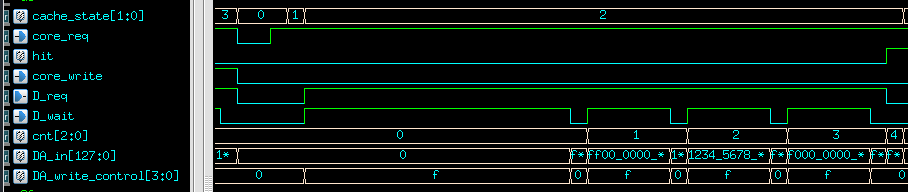
**(2) Waveform Analysis**

**i. Read Hit**

****

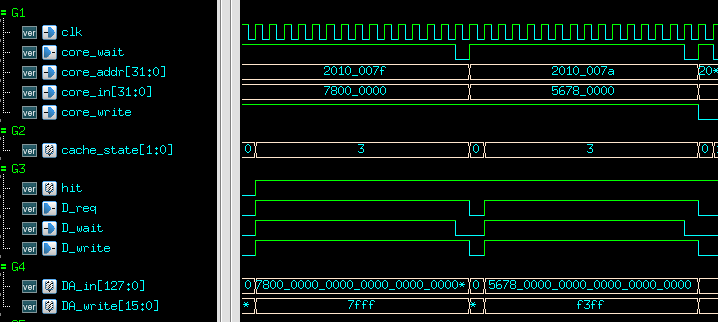
當**IDLE(0)**時候，DA\_read active HIGH表示可以讀取。接著當req且valid時到狀態**CHECK(1)(紅框處)。**此時發生了hit=1，表示tag匹配，這時就直接拿取cache中的值來給CPU**(黃框處)**。此時會回傳一筆128bit的block，再根據core\_addr[3:2]來表示所要取的資料位於此block中的哪個位置**(藍框處)**。最後把有效data傳給core\_out，即回傳CPU。

**ii. Read Miss**

****

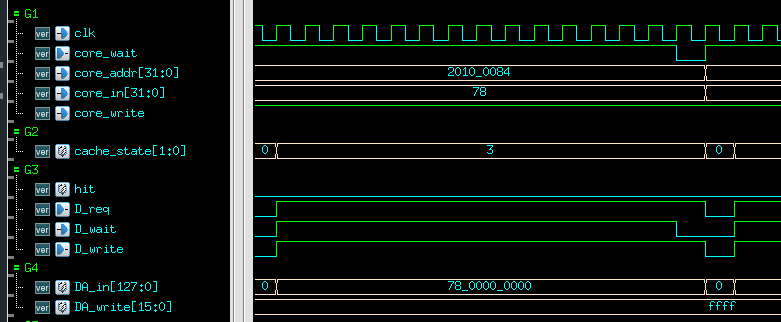
同上相似處就不予贅述，差別在於這次狀態**CHECK(1)**發生了hit=0且core\_write=0，表示tag不匹配，因此進到狀態**READ(2)(紅框處)。**接著因為要向DRAM要4筆word，且每當data運送成功時，會將**D\_wait pull low(黃框處)。**讀完4次之後，會將4筆共128bit data用DA\_in存起來**(藍框處)**。

**iii. Write Hit**

****

當**IDLE(0)**時候，發生了core\_req且core\_write=1，因此進到狀態**WRITE(3)(紅框處)。**此時hit=1，接著向slave **pull up D\_req**來得到要寫入的位址，此時DRAM會 **pull up D\_wait**讓cache暫停**(黃框處)**。直到DRAM **pull down D\_wait，**表示data已經完成寫入**(藍框處)**。此外也會將core\_in一同寫入cache中，再根據core\_type或是offset來決定DA\_write行為。

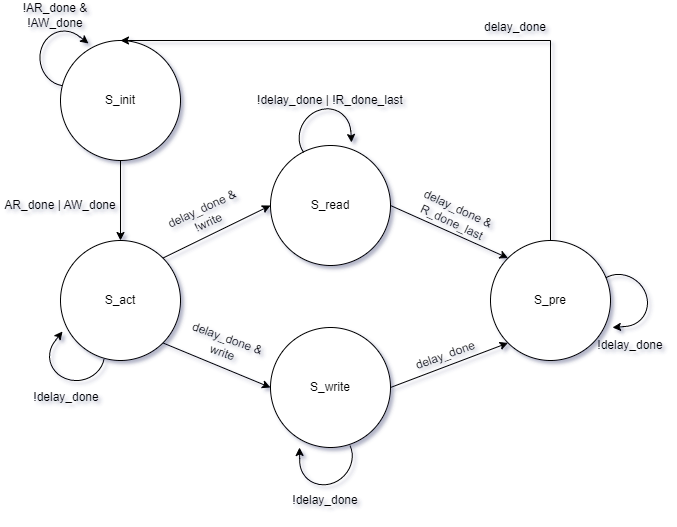
**iv. Write Miss**

****

同上相似處就不予贅述，差別在於這次hit=0**(紅框處)**。因為不會再另外寫入cache之中，所以說此時可以觀察到DA\_write為4`hFFFF**(黃框處)**，表示不寫入cache。最後**pull down D\_wait**來表示寫入完成。

**2. DRAM\_Wrapper**

**(1) State Diagram**

****

**i. S\_init :** 如果說AR channel或是 AW channel完成handshake的時候，就會從**S\_init**狀態進入**S\_act**狀態。此狀態用於初始化DRAM相關訊號來等待AXI送訊號進來。

**ii. S\_act :** 此時會根據AXI送來的write以及delay\_done來分辨是要做read或是write。且如果說delay\_cnt == 3'b0，則將DRAM\_RASn設為0，反之設為1，來對ROW進行選擇。

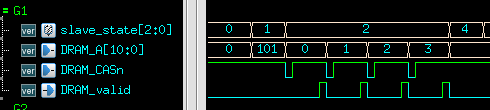
**iii. S\_read :** 當delay\_done且不是write時會進到此狀態。由於以經在前一狀態中對ROW做選擇，接下來因為一次讀4筆data都在同一選擇ROW上，所以當delay\_cnt == 3'b0，則將DRAM\_CASn設為0，反之設為1，來對COL進行選擇。

**iv. S\_write :** 與**S\_read**狀態類似，相同部分不予贅述。比較特殊的是在DRAM\_Wen，主要根據write strob和byte offset，決定要寫入多少。且一樣要當delay\_cnt == 3'b0時AXI傳來的WSTRB才有效，其餘狀況皆設為1`hF防止寫入。

**v. S\_pre :**  最後需要先precharge，所以在delay\_cnt == 3'b0狀態下重設DRAM\_RASn以及DRAM\_Wen，使得下一次的讀寫能正常運作。

**(2) Waveform Analysis**

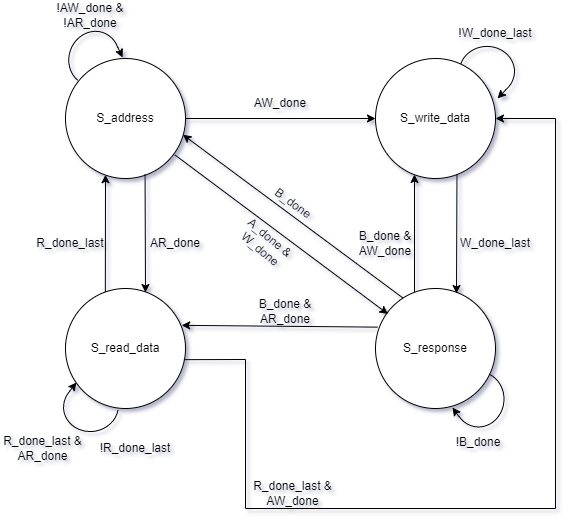
**Read Miss :**

****

以上圖為例，由於cache read miss，所以會到DRAM中去做讀取。首先在狀態**S\_act(1)**的時候選定好所需要的**ROW(11`h101)(紅框處)。**之後在狀態**S\_read(2)**的時候將**CASn pull low**來將選定的COL傳給DRAM，之後會將**DRAM\_valid pull HIGH** 來表示所讀取到的data為有效(黃框處)。因為reaad miss 一次就會需要讀取連續的4筆data，且此4筆data都位於同一ROW上，只要更改COL就可以取得不同的值(藍框處)。

**3. ROM\_Wrapper**

**(1) State Diagram**

****

使用SRAM\_Wrapper架構作修改，其中因為ROM為唯讀記憶體，並不會有寫入的動作，所以將write channel都設為不動作

**4. Boot**

在程式執行之前，會需要藉由boot.c來將DRAM中的data搬移到SRAM供我們做指令讀取或是做資料讀寫。一開始boot.c會存在ROM之中，我們可以觀察到指令都是放置於\_dram\_i\_end - \_dram\_i\_start之間，並且將這些指令都放到\_imem\_start之後才能正確讀取指令並執行。接著data被放置於兩處，分別是\_sdata\_end-\_sdata\_start 以及 \_data\_end-\_data\_start，同樣的我們需要藉由boot來將這兩處data分別放置於\_\_sdata\_paddr\_start 以及 \_\_data\_paddr\_start 之後。所以我們可以使用迴圈來將作資料搬移

**Pseudo code :**

**For i = 0 to len(DRAM\_instructions):**

**&instr\_mem\_start[i] = & instre\_DRAM\_start[i];**

**// The “instr” can be replace with “data” or “sdata”**

**Simulation**

**1. RTL & SYN**

|  |  |  |
| --- | --- | --- |
|  | rtl | syn |
| Prog0 | 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 |
| Prog1 | 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 |
| Prog2 | 一張含有 文字, 電子用品, 螢幕擷取畫面 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 |
| Prog3 | 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 |