VLSI System Design (Graduate Level)

Fall 2023

HOMEWORK IV

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

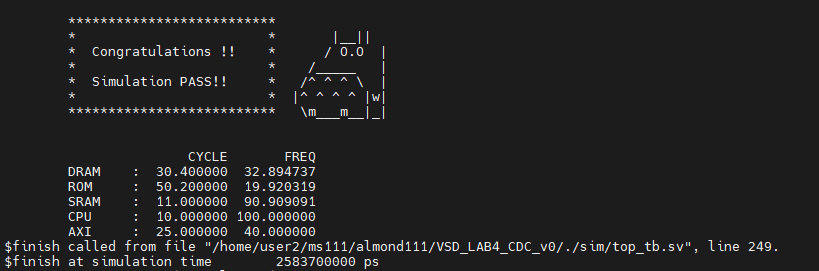
Organize files according to File Hierarchy Requirement

No waveform files in deliverables

|  |  |  |
| --- | --- | --- |
| Student name | 吳欣航 | 呂修麟 |
| Student ID | N26124395 | N26124272 |

|  |  |  |  |
| --- | --- | --- | --- |
| Performance & Area | | | |
|  | rtl | syn | pr |
| Prog0 time | 5270188000ps | 5249790000ps | 5270188000ps |
| Prog1 time | 40597536000ps | 40604914000ps | 40687192000ps |
| Prog2 time | 3842342000ps | 3847466000ps | 3861368000ps |
| Prog3 time | 14190246000ps | 11399682000ps | 12829544000ps |
| Area(um^2)  In APR | 8981332.45 | | |
| CPU cycle | 14ns | | |

Refer to the figure to fill in the simulation time !!



**Contribution**

吳欣航(N26124395) ： 50%

L1 instruction Cache、L1 data Cache、CDC整合與Debug、CPU Wrapper修改、Slave修改、Pre-simulation、Synthesis、 Post-simulation、APR。

呂修麟(N26124272)：50%

AFIFO implement、Cache整合與debug、Spyglass驗證、DRAM wrapper修改、Pre-simulation、Synthesis、 Post-simulation、APR。

**Hardware Design Description**

* System Block Diagram

本次作業除了上次的2個master和6個slave的系統外，還在IF stage和MEM state增加了兩個Level 1 Cache，以增加系統運算的速度。

除了增加了Cache，這次系統有5個clock domain，大大增加設計難度，每個slave和master都和AXI的clock domain不同，因此我們需要在每個master和slave中間加一個afifo 以處理CDC的問題。

最後除了合成外也要使用ICC2來進行APR的設計，並通過所有的模擬驗證。

一張含有 文字, 螢幕擷取畫面, 設計 的圖片

自動產生的描述

**CACHE**

本次作業需要新增Level 1的cache電路，採用Direct map的方式，block size為16bytes。分別儲存IF stage和MEM的資料，且sensor傳過來的資料為uncacheable，因此不進到cache的block裡面，會直接通過。

**DATA CACHE**

接下來會以data cache的狀態做詳細說明。

一張含有 圓形, 螢幕擷取畫面, 文字, 月亮 的圖片

自動產生的描述

**IDLE**: 在data cache中，若是有read\_signal則會進到READ，若是write\_signal則會進到WRITE。我的設計是會在IDLE狀態更新register，並在IDLE就會將TA\_read打開，如果下一個state是READ的話，在當下就可以將addr送進TA和DA裡面，並在下一個cycle判斷有沒有hit，如果有的話就可以直接將直送出來。

**READ**: 此階段會判斷有沒有hit，如果hit的話，將會在這一個cycle將直送出來，並在下一個cycle回到IDLE。若是沒有hit則會進到READ\_BURST。

**READ\_BURST**: 在這個階段，Cache會去向master發出request和address，由於這次cache的block size為16bytes，我們的設計是將address到address+4的四筆資料連續送過來存到cache裡面，相較於以前的AXI只能夠一次傳一筆，這一次作業在slave上做了一些調整，最後收到RLAST則進到READ\_END，在這個state也回同時更新TA裡面的值。

**READ\_END**: 會有READ\_END這一個state就只是在等AXI的最後一筆資料存進DA中而已，此時經過一個cycle就會進到下一個state IDLE，因為TA在READ\_BURST已經順利被更新，因此到下一個READ應該會順利的把值直接吐出來並回到IDLE。在計算hit rate時有把此情況排除，以免送到錯誤的hit rate。

**WRITE**: 在IDLE收到write\_signal後會進到WRITE state，這邊會將address 和 data送進slave做寫入，值得注意的是，我們採用write through的策略，就是每一次寫入都會寫入Master裡，在我們的設計，只有在hit狀況發生時，要寫入cache裡面，以免沒有被更新到，使用write through策略的好處就是資料用遠保持一致，以及設計簡單，收到BREADY和BVALID代表寫完了，則進入WRITE\_END。

**WRITE\_END**: 經過一個cycle後會進到IDLE，當初會這樣設計似乎是因為避免抓太緊，但現在想想這個state好像不是很必要。

**PASS\_READ**: 在IDLE收到read\_signal後，若地址是屬於sensor的範圍，則判斷為uncacheable，此時address並不會進到cache裡面，會直接通往master，master拿回來的資料依樣會直接給CPU，收到RLAST則回到IDLE。

**PASS\_WRITE**: IDLE 收到write\_signal後，若地址是屬於sensor的範圍，則判斷為uncacheable，此時address, data並不會進到cache裡面，會直接通往master，收到BVALID && BREADY則回到IDLE。

**INSTRUTION CACHE**

一張含有 圓形, 螢幕擷取畫面, 文字 的圖片

自動產生的描述

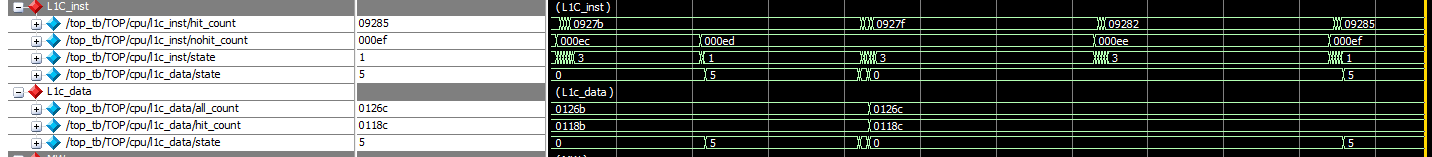
Instruction cache與Data Cache的架構十分相似，但在設計上有些許不一樣，除了state較少外，較為不同的地方為，若一開始read miss，我在READ\_END的state會回到IDLE，在下一個cycle才會去READ拿資料，這部分我想也可以優化，但礙於時程關係沒有做這一部分的調整。

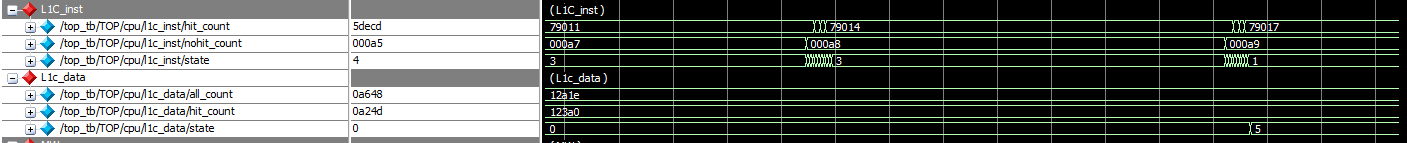
**Hit Rate**

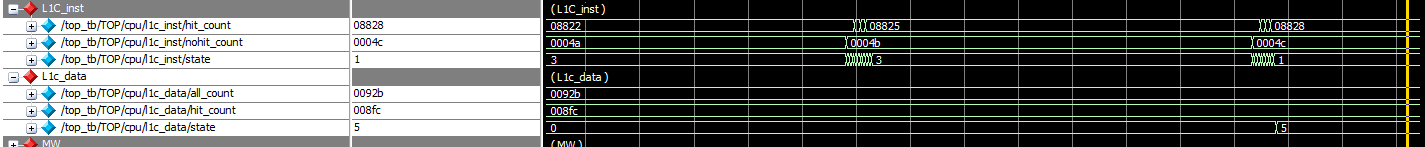
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hit Rate | Prog0 | Prog1 | Prog2 | Prog3 |
| Instruction | 99.36% | 99.60% | 99.78% | 99.85% |
| Data | 95.25% | 97.60% | 97.99% | 97.16% |

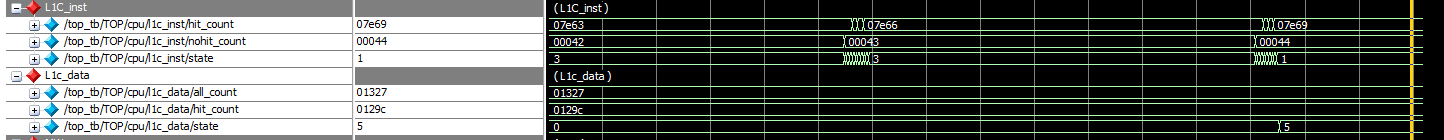
計算出這些hit rate後期時嚇了一跳，覺得非常不合理，但想想因為中間fill block的部分占了一大部分，因此我認為這份數據有參考價值。以下附上我在modelsim上計算hit count的波型，instrution的部分我是分別計算有hit和no hit，data則直接計算全部的數量跟hit count。

Prog0

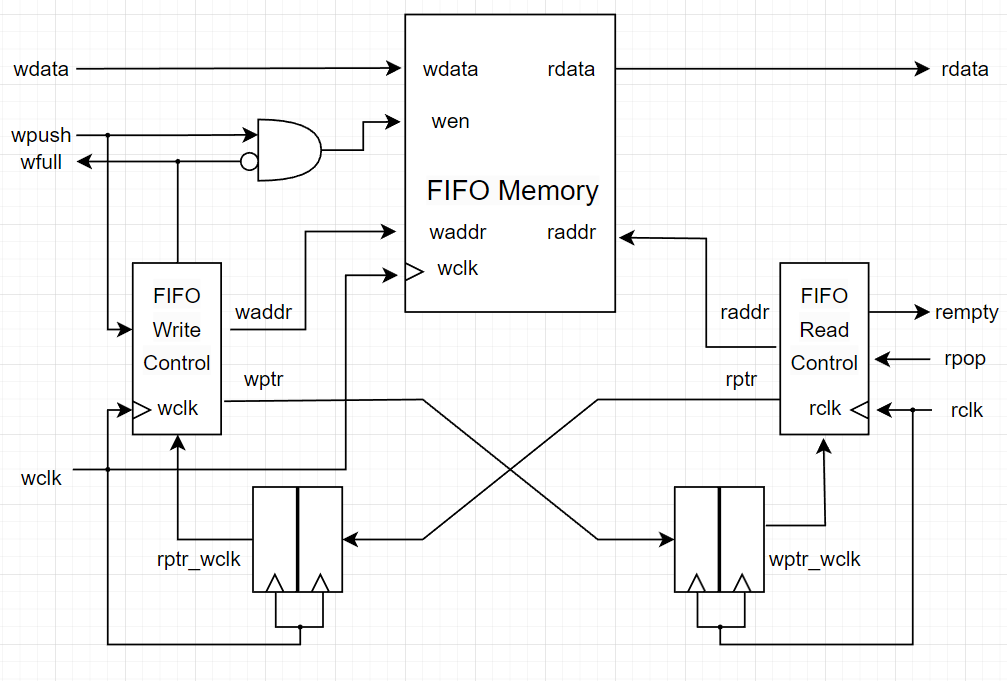


Prog1

Prog2

Prog3

**AFIFO**



上圖為我們AFIFO的架構，主要有五個module所組成:

1. FIFO memory : 控制資料寫入和讀出
2. Write control: 控制和處理 waddr和wptr 之間的gray code轉換，並且將本身的wptr和同步過的rptr\_wclk做wfull的判斷
3. Read control : 控制和處理 raddr和rptr 之間的gray code轉換，並且將本身的rptr和同步過的wptr\_rclk做empty的判斷
4. Sync\_r2w : 將 rclk 的 rptr 經過兩個 DFF同步成rptr\_wclk
5. Sync\_w2r : 將 wclk 的 wptr 經過兩個 DFF同步成wptr\_rclk

在實際使用上，以AXI和ROM為例講解AR 的channel。除了要接對wclk和rclk以外，最重要的是寫入端如何控制wpush，讀出端如何控制rpop。wpush我們使用諸如ARVALID這種重要的訊號搭配XOR去製造一個clk的pulse。rpop我們讓讀出端透過empty去做判斷，並且搭配reg去儲存我們想要拿到的資料，包含ARVALID和ARADDR等重要資訊。

**Screen shot of wave forms and simulation results**

**Waveform**

**CACHE**

Cache Instruction read (hit)

一張含有 螢幕擷取畫面, 文字, 軟體, 多媒體軟體 的圖片

自動產生的描述

可以看到state為 0 1 0 1 0 1 0 1，就是IDLE READ IDLE READ…在跳，這時Cache不會對master發出任何請求，而是直接將cache 裡面的值給到CPU。

Cache Instruction read (not hit)

一張含有 螢幕擷取畫面, 文字, 軟體, 多媒體軟體 的圖片

自動產生的描述

在這張圖可以看到instruction not hit的狀態，從IDLE跳到READ再跳到READ\_BURST，最後READ\_END，這階段屬於Cache在對master發請求，並將地址給到master，這邊有counter再針對DA\_in的遮罩做處理。

Cache Data read (hit)

一張含有 螢幕擷取畫面, 文字, 軟體, 多媒體軟體 的圖片

自動產生的描述

在此可以看到DATA CACHE hit的狀態，狀態依樣是IDLE READ IDLE在跳，也不會對master發出任何請求，單純從cache裡拿東西。

Cache Data read (not hit)

可以看到instruction not hit的狀態，從IDLE跳到READ再跳到READ\_BURST，最後READ\_END，這階段屬於Cache在對master發請求，並將地址給到master，這邊有counter再針對DA\_in的遮罩做處理。一張含有 螢幕擷取畫面, 軟體, 文字, 多媒體軟體 的圖片

自動產生的描述

可以看到data not hit的狀態，從IDLE跳到READ再跳到READ\_BURST，最後READ\_END，這階段屬於Cache在對master發請求，並將地址給到master，這邊有counter再針對DA\_in的遮罩做處理。

Cache Data write(hit)

一張含有 螢幕擷取畫面, 文字, 軟體, 多媒體軟體 的圖片

自動產生的描述

上面的紅框框代表cache對master發出寫的操作，而因為有hit到，若是不把資料更新到cache裡面可能會造成後面的讀取出錯，因此可以看到下面框框有把資料寫進地址裡。

Cache Data write(not hit)

一張含有 螢幕擷取畫面, 軟體, 文字, 多媒體軟體 的圖片

自動產生的描述

跟上一張圖依樣都是寫的操作，但這一張圖並沒有hit，因此不用更新cache裡面的值。

Sensor Write

一張含有 螢幕擷取畫面, 文字, 軟體, 多媒體軟體 的圖片

自動產生的描述

這一張圖展示了uncacheable的write，這個state會直接將address和data直接通到master，並對master發出write訊號。

Sensor Read

一張含有 螢幕擷取畫面, 文字, 軟體, 多媒體軟體 的圖片

自動產生的描述

這一張圖展示了uncacheable的read，這個state會直接將address直接通到master，並對master發出req訊號。

AXI

**一張含有 軟體, 多媒體軟體, 文字, 繪圖軟體 的圖片

自動產生的描述**

這一次作業也對slave做了一些修改，將原本AX能傳一個改成可以連訊傳四筆資料，這邊使用counter實現設計。

**AFIFO**

Cpu to AXI

一張含有 螢幕擷取畫面, 文字, 軟體, 多媒體軟體 的圖片

自動產生的描述

上圖為CPU中Master的AXI透過AFIFO傳遞到AXI的AW波型，可以看到最上面的AWVALID為一後，會控制wpush將AW的資訊存進AFIFO，然後AXI端在檢測到empty不為零後，會控制rpop將資訊讀出，在上圖可以看到橘色框住的AXI內部AW訊號，在rpop後成功接收到20f60這個地址。

AXI to ROM

一張含有 螢幕擷取畫面, 文字, 軟體, 多媒體軟體 的圖片

自動產生的描述

上圖為AXI傳遞到ROM的AR波型，可以看到最上面的ARVALID為一後，會控制wpush將AR的資訊290存進AFIFO，然後AXI端在檢測到empty不為零後，會控制rpop將資訊290讀出。AR之後是R channel，在上圖可以看到橘色框住的ROM DO，在經過AFIFO同步到上面的RDATA\_M0。

AXI to SRAM

一張含有 螢幕擷取畫面, 文字, 軟體, 多媒體軟體 的圖片

自動產生的描述

上圖為AXI傳遞到SRAM的AR波型，可以看到wpush後會將AR的資訊104c0存進AFIFO，然後AXI端在檢測到empty不為零後，會控制rpop將資訊104c0讀出。AR之後是R channel，在上圖可以看到橘色框住的SRAM DO，在經過AFIFO同步到上面的RDATA\_M0。

AXI to DRAM

一張含有 螢幕擷取畫面, 文字, 多媒體軟體, 行 的圖片

自動產生的描述

上圖為AXI傳遞到DRAM的AR波型，可以看到wpush後會將AR的資訊2010\_02b0存進AFIFO，然後AXI端在檢測到empty不為零後，會控制rpop將資訊2010\_02b0讀出，並且開始DRAM的控制，例如拉低CAS，以及拿到valid反饋訊號。AR之後是R channel，在上圖可以看到橘色框住的DRAM RDATA\_out，在經過AFIFO同步到上面的RDATA\_M1。

AXI to sensor

一張含有 設備, 文字, 螢幕擷取畫面, 電子產品 的圖片

自動產生的描述

上圖為AXI傳遞到SENSOR的AR波型，可以看到wpush後會將AR的資訊ARVALID存進AFIFO，然後AXI端在檢測到empty不為零後，會控制rpop將資訊ARVALID讀出，並且開始讀出sctrl\_out的控制。AR之後是R channel，在上圖可以看到橘色框住的sctrl\_out，在經過AFIFO同步到上面的RDATA\_M1。

**Timing Report (synthesis)**

一張含有 文字, 螢幕擷取畫面, 字型, 陳列 的圖片

自動產生的描述

一張含有 文字, 螢幕擷取畫面, 字型, 陳列 的圖片

自動產生的描述

一張含有 文字, 螢幕擷取畫面, 陳列, 字型 的圖片

自動產生的描述

一張含有 文字, 螢幕擷取畫面, 陳列, 字型 的圖片

自動產生的描述

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

每一個clock domain皆沒有violation。

**Power Report (synthesis)**

一張含有 文字, 螢幕擷取畫面, 軟體, 多媒體軟體 的圖片

自動產生的描述

**Area Report (synthesis)**

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

合成後面積使用了 9247946.36。

**APR**

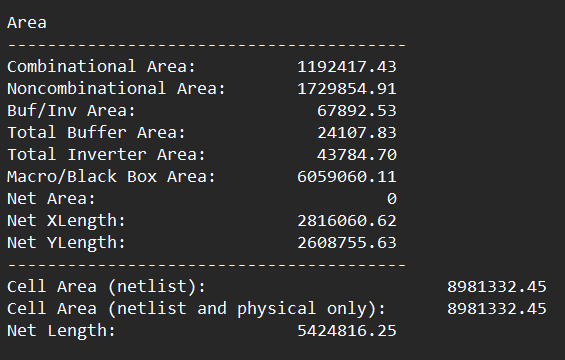
這一次使用icc2作為APR的工具，我們直接使用助教提供的script進行APR的工作。

APR Result

一張含有 螢幕擷取畫面, 紫色, 鮮豔 的圖片

自動產生的描述

Total area



面積總共8981332.45。

Power

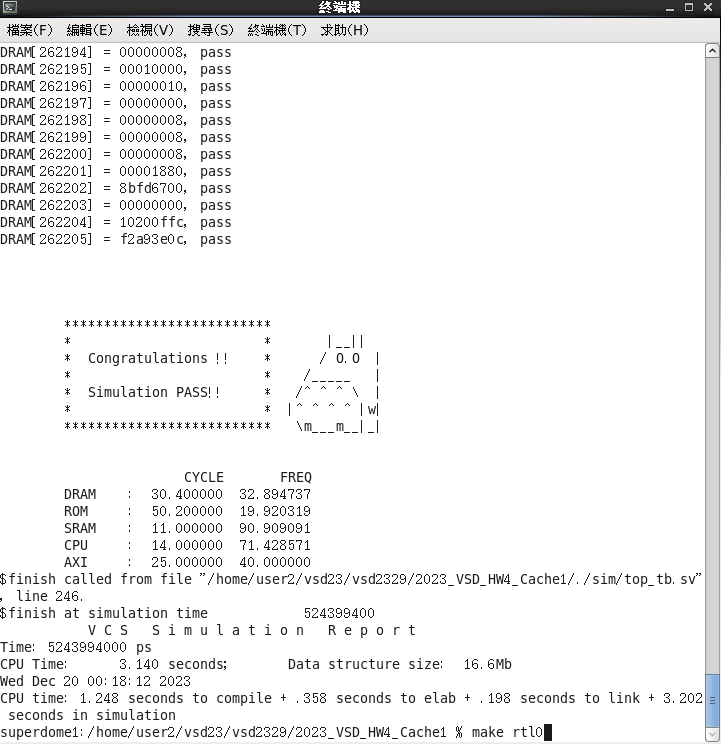
一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

這一次在效能上的結果不盡理想，希望能在未來有更進一步的探討。

**Simulation Results**

**Pre-sim**

Prog0

Prog1一張含有 文字, 螢幕擷取畫面, 陳列, 軟體 的圖片

自動產生的描述

Prog2一張含有 文字, 螢幕擷取畫面, 陳列, 軟體 的圖片

自動產生的描述

Prog3一張含有 文字, 螢幕擷取畫面, 軟體, 數字 的圖片

自動產生的描述

**Post-sim**

Prog0一張含有 文字, 螢幕擷取畫面, 陳列, 軟體 的圖片

自動產生的描述

Prog1一張含有 文字, 螢幕擷取畫面, 陳列, 軟體 的圖片

自動產生的描述

Prog2一張含有 文字, 螢幕擷取畫面, 陳列, 軟體 的圖片

自動產生的描述

Prog3一張含有 文字, 螢幕擷取畫面, 陳列, 軟體 的圖片

自動產生的描述

**Pr-sim**

Prog0一張含有 文字, 螢幕擷取畫面, 陳列, 軟體 的圖片

自動產生的描述

Prog1一張含有 文字, 螢幕擷取畫面, 陳列, 數字 的圖片

自動產生的描述

Prog2一張含有 文字, 螢幕擷取畫面, 陳列, 軟體 的圖片

自動產生的描述

Prog3一張含有 文字, 螢幕擷取畫面, 軟體, 陳列 的圖片

自動產生的描述

**Superlint**

一張含有 文字, 螢幕擷取畫面, 軟體, 網頁 的圖片

自動產生的描述

在superlint只留下幾個warning，其中大多數都是屬於使用synchronous reset造成的warning，會使用synchronous reset是為了符合spyglass的規範。值得一提的是，使用synchronous reset會讓superlint沒辦法檢查到錯誤，因此我們先將reset變成aynchronous到superlint上跑，改掉錯誤後再將reset改回synchronous reset。

此外可以看到MOD\_NO\_IPRG這個，我們一開始忽略這些消息，若是把這些訊號直接接上CPU會造成timing violation，因此我們的做法是在中間加上buffer。

剩下的warning幾乎都是位元沒有對其的小問題。

**Spyglass**

一張含有 文字, 螢幕擷取畫面, 陳列, 軟體 的圖片

自動產生的描述

一張含有 文字, 螢幕擷取畫面, 軟體, 陳列 的圖片

自動產生的描述

我們的CDC電路使用AFIFO來達成，在經過spyglass的CDC 檢測後，只有warning，並沒有error。Warning的部分可能是因為AFIFO和週邊訊號較多，所以他們沒辦法檢測到error，但我們的電路可以達成partially proved，這就表示我們的CDC電路的能力足夠讓spyglass檢測不出error。

**Lesson learned**

**吳欣航**:

這次做作業的過程可以說是一波三折，要在前三次作業的基礎上新增cache，這無疑是一大挑戰，因為先前的cpu 對兩個master stall訊號已經花了非常多力氣在處理，現在加了cache進來後，這些訊號要重對花了我們蠻多的時間，讓我們在實作上遭受到一些挑戰。這一次作業我主要負責cache的撰寫，我認為cache的概念不難，然而用FSM控制他必須非常的小心謹慎，一不小心就可能會造成program的卡住，或是program跳不出回圈，會這樣說的原因是因為一開始覺得cache是一件不會太難的事，所以我就沒有很仔細的思考每個state該做甚麼事，一開始接上去雖然很快就動起來了，行為也蠻正確的，但是就是有一寫bug怎麼解都解不了，甚至在上面卡了兩天之久，code改了又改，改了又改，改道我已經不認識我的code為甚麼這樣寫了，最終痛定思痛，審慎思考後重寫一份，果不其然接上去後過了一個小時的debug就通過prog0了，這一次的經驗讓我學到以後要做設計前不能抱著僥倖的心態，需要全盤考量後再動手。

結束cache之後我們才將AFIFO接上去，理論上接上去應該就要可以用才對，但我們疏忽了對RVALID的控制，造成他會一次讀很多筆資料出來所幸這個問題算是明顯，因此解決的算順利。我一開始合成的時候也出現問題，上次作業我們為了要通過spyglass的驗證，將asynchronous reset調整成synchronous reset，而這一次作業我們一樣使用synchronous reset，但這造成superlint會檢查不到錯誤，所以一開始合成時我總是會依值出錯，也找不到解決方法，後來將synchronous reset全部換成asynchronous reset後，丟到superlint跑發現有蠻多地方沒有初始化到，等superlint都解完了再將asynchronous reset條回asynchronous reset，就可以成功動起來了。

我覺得較為可惜的是APR的部分，由於先前完全沒有接觸過APR因此有點不知所措，這次作業光寫完rtl code就要花很多時間了，因此沒有很仔細的鑽研APR的部分，只用助教提供的script來實做，所幸最終是有通過模擬。就當我以為一切都要結束時，交功課前一天我在截波形寫報告的時後發現我們的data cache從沒有hit過，這件事以前完全沒有發現，我就趕緊花了一個晚上的時間找出問題在哪，原來發現是我在write的時候不小心會更新他的TA，因此他就會一直miss，所幸最後有找到問題並成功通過所有模擬。

累歸累，但我覺得十分充實，透過自己的能力完成一個系統的搭建屬實不易，但真的學到非常多，接下來就好好準備final project吧! 感謝助教很用心地提供許多資訊以及步驟讓我們能順利完成這次作業。

**呂修麟**:

在這次作業中，在前三次作業的基礎上，增加了cache，並且有多個clock domain，因此我們必須使用AFIFO去解決CDC問題。在cache的部分感謝吳欣航同學解決了最重要的cache這個部分，由於以前AXI一次只有傳一筆資料，這次我們花了不少時間討論這部分的議題，我們在有了cache的基礎上才進行後續的AFIFO整合和設計到AXI之中。這次作業我主要是負責AFIFO設計和整合的設計，最大的挑戰是要能精通AFIFO的運作原理，並且將其妥善地整合到我們電路中，其中RVALID這種重要訊號的控制也讓我們花了很多時間才發現設計上處理RVALID的瑕疵。AFIFO擺放的位置也很重要，否則會需要花時間去接多餘的wire，在設計和除錯上也都會是挑戰。這次的作業讓我們學習到了在整合許多module的系統時，若有額外需求，須注意到的大大小小的事情，要看的波型也非常多，但也學到很多經驗，希望Final順利。最後感謝助教很用心地提供許多資訊讓我們能順利完成這次作業。