VLSI System Design (Graduate Level)

Fall 2021

HOMEWORK II

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: 王傑世(60%)

Student ID: P76091690

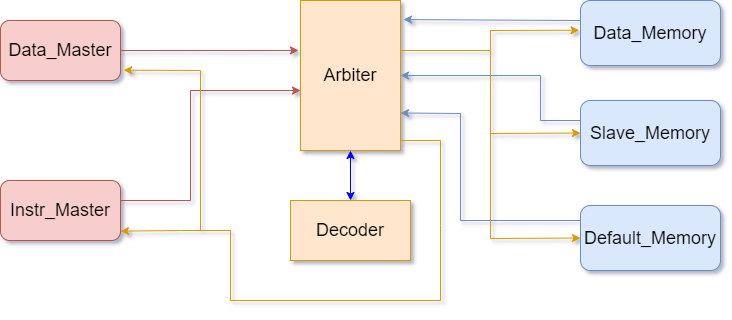
Student name: 陳韋綸(60%)

Student ID: P76091412

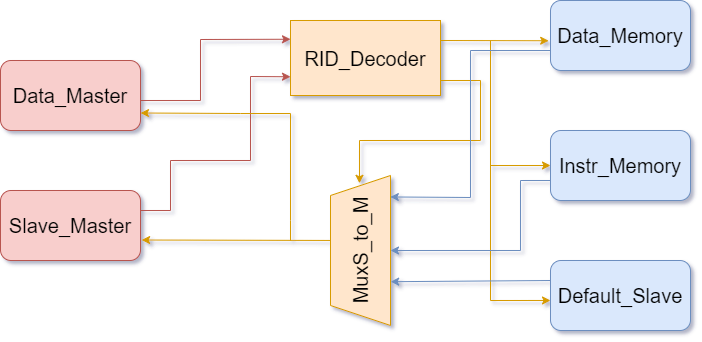
* **Summary**

1. 完成Jaspergold 驗證
2. 完成Prog0~Prog3 RTL 驗證
3. 完成Prog0~Prog3 Gate-level simulation驗證

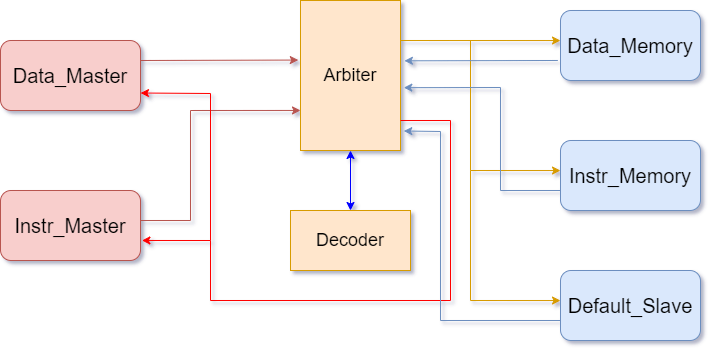
* **AXI Bus System Overview**
* Read Address Channel



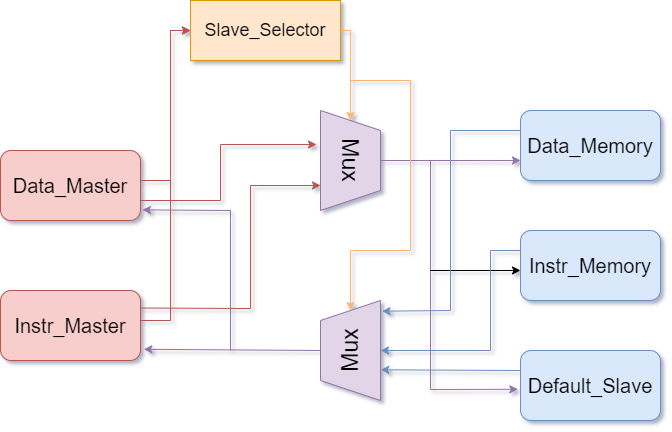
* Read Data Channel



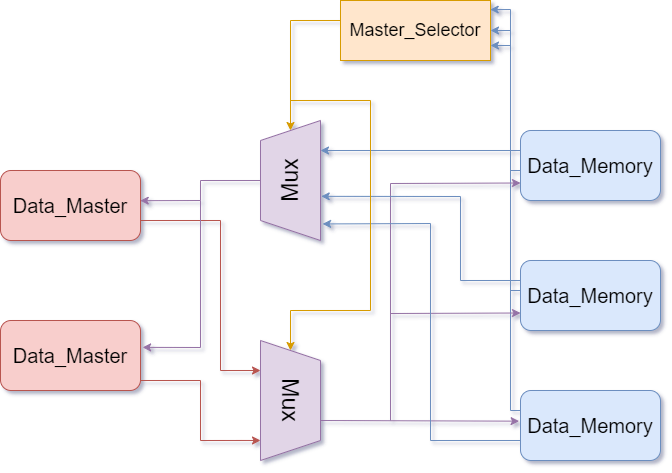
* 在Read Transaction中當任何一個Master要Read data時會在Read Address Channel發出一個ARVALID的signal告訴Arbiter我要進行一個有效的傳輸，並且將Address送入Decoder中，而這時候Arbiter會決定哪個Master可以先進行這次的transaction，並且將ARID、ADDR、ARLEN、ARSIZE、ARBURST一併送入slave，而Decoder則會根據所傳入的address去分析master所想要access的是哪一個slave，接著Arbiter再根據該slave傳入的ARREADY是否為high來判斷這次的Handshake是否完成，再slave完成Address的handshake之後，slave就會根據收到address和ARLEN去準備所需要的data，然後RID decoder會根據ARID把Read Data Channel中正確的資料送回到Master中，此外在每一筆data的transfer中RID\_Decoder也會根據收data的Master是否有把RREADY拉高來決定這一筆資料transfer的handshake是否完成，每當handshake完成後才會開始進行下一筆資料的transfer，最後當slave把Master所需要的最後一筆資料回傳回去之後，slave也會把RLAST設成1，告訴Master所有它需要的資料都已經傳送完成了，另外，在我的架構中當2個Master同時把ARVALID拉高時，我會讓data\_master有較高的優先權可以進行transaction，也就是說instr\_master會一直等待直到data\_master完成transaction才會開始它的data transfer。
* Write Address Channel



* Write Data Channel



* Write Response Channel



* 在Write transaction中，任一個Master要對Slave進行寫入的行為前，都會先在Write Address Channel中發出一個AWVALID告訴Arbiter我要進行一次transaction，並把AWID、AWLEN、AWSIZE、AWBURS也放到Write Address Channel上，之後address被送到decoder中，decoder會根據address去判斷要寫入的是哪一個slave，接著Arbiter再根據該slave所傳入的AWREADY訊號來決定這次的handshake是否完成，在slave 完成Write address channel的handshake之後Master就會開始把WDATA、WSTRB、WVALID放到Write Data Channel上，在Write Data Channel中Slave selector會根據剛剛2個master所傳入的AWVALID去判斷Mux要把哪個Master的DATA送到slave中，而slave則會回傳WREADY告訴Master這筆寫入資料的handshake已經完成了，再繼續進行下一筆資料的寫入，直到Master傳入WLAST告訴slave這是最後一筆要寫入的資料了，Slave再完成WLAST為1的handshake之後就會在Write Response Channel放上BVALID、BRESP以及BID，在Write Response Channel中的Master\_selector就會根據BID去判斷要讓哪個slave的response通過mux，同時也會透過這個BID把Master的BREADY正確的送到slave中告訴slave這個response它收到了，此次write transaction已經全部完成了，最後雖然在這份作業中Instrucion Master不會寫入任何資料到Instruction Slave中，不過我在實作上還是有把Instrucion Master的output接到Write Address Channel、Write Data Channel和Write Response Channel中。
* **Jaspergold VIP Result**
* Bridge
* 在Jaspergold中，我的Bridge驗證全數通過沒有任何violation 一張含有 文字 的圖片

  自動產生的描述
* Slave
* 在Jaspergold中我的slave驗證全數通過，沒有任何Violation 一張含有 文字 的圖片

  自動產生的描述
* Master
* 在Master中一共有8個情況無法cover到，之所以會有一些cover沒辦法通過驗證是因為在我的設計上只有當Slave把Valid訊號拉高時Master才會把Ready訊號同時拉高，也因此在做VIP的驗證時會出現永遠沒有辦法cover到的情況，因為Ready訊號不會比Valid訊號要早拉起來。

1. Bvalid cover violation
2. Bresp cover violation
3. Bid cover violation
4. Rvalid cover violation
5. Rdata cover violation
6. Rresp cover violation
7. Rid cover violation
8. Rlast cover violation

一張含有 文字 的圖片

自動產生的描述

* 在Read data channel中可以看到沒有cover到的條件都是(rvalid && !rready) |=> $stable(/\*----signal----\*/)，這是因為在我的架構中Rvalid一拉起來Rready也會馬上拉起來並且完成handshake，也就是說這3件事情會在同一時間進行，因此不會出現Rvalid為1但Rready為0的情況，所以不論是rlast、rid、rresp、rdata、rvalid都會無法cover到上述的情況。

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

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

一張含有 文字 的圖片

自動產生的描述

* 在Write Response channel中，cover所違反的條件與read data channel類似，也都是(bvalid && !bready) |=> $stable(/\*---signal---\*/)，這也是因為在我Master的設計中當bvalid為1時，bready也會馬上變成1，並且完成handshake，因此我的電路中不會出現bvalid為1但是bready為0的情況，所以才會發生bid、bresp、bvalid永遠不會被cover到的情況。

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

一張含有 文字 的圖片

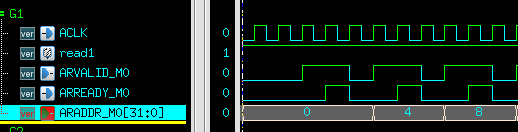
自動產生的描述

* **CPU with AXI Bus Overview**

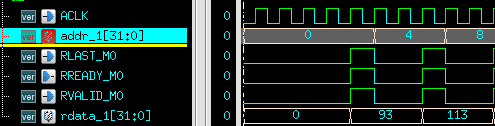
一張含有 文字, 標誌, 綠色, 螢幕擷取畫面 的圖片

自動產生的描述

* 在TOP module之下有4個component，分別是CPU wrapper、AXI Bus、IM以及DM，CPU wrapper會把從CPU送出來的訊號，像是address、read、write之類的signal轉換成AXI BUS上的訊號，再透過Bridge和Instruction Memory或是Data Memory完成資料的傳輸



* 如圖所示，當CPU想要access Memory時，它會透過CPU wrapper發出一個ARVALID到BUS上，並且把Address一併傳到BUS中，等到slave回傳ARREADY時就代表handshake完成了，這次Read Address signal的transfer就完成了

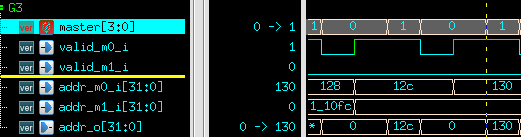


* 接著當Slave收到由Master傳來的Address之後便會把它所要的資料準備好，同時把RLAST和RVALID拉高，並且這時候Master也會把RREADY拉高，代表這時候handshake完成，此次Read data transfer成功完成
* **How AXI Bridge transfer signal and data between Master and Slave**
* 在這部分會利用Read Transaction說明AXI BUS中的bridge如何連接Master和Slave雙方
* 以第130條指令add t0,t0,t1為例

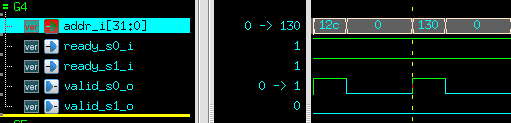
一張含有 文字, 時鐘 的圖片

自動產生的描述

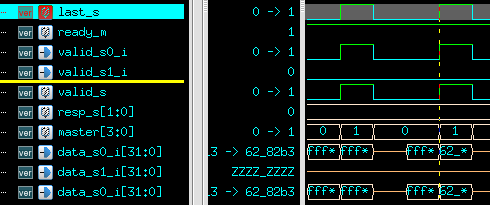
* 在Read Address Channel中的Arbiter首先會收到由2個Master送入的訊號，並且利用AWVALID訊號判斷現在是哪個Master要Read Instruction Memory，然後再把該Master的Address等訊號送出去給Slave



* 此時Decoder也會根據送入的Address判斷要讀取的是哪個Slave，並且根據該Slave的ready值來判斷此次Read Address的handshake是否完成，若該slave尚未ready則Arbiter需要把Master的address和valid都保存好，直到slave變成ready並且完成handshake為止，如下圖所示，因為slave0的狀態為ready，就代表可以完成這次Address和其它signal的傳輸

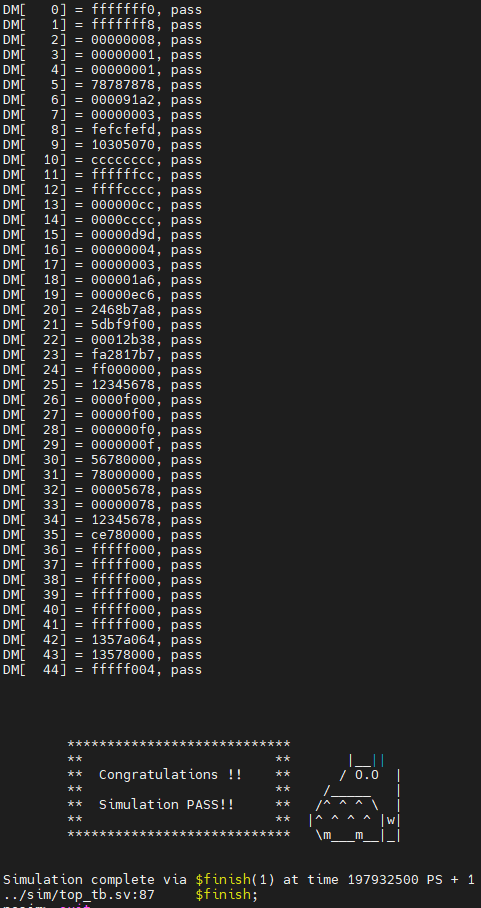


* 接著在Read Data Channel中，會根據ARID去決定現在要送資料的目的地是哪一個Master，並且會根據Slave傳進來的RVALID來判斷是data是由哪個Slave送的並讓它通過Mux，最後再根據Master所送入的RREADY來判斷data是否有成功送回，如下圖所示，因為只有Slave\_0是Rvalid的狀態，所以最後送回Master的data就是Slave所送入的data，並且會根據ARID判斷出目前要送回的Master為Master\_0，然後把RLAST也送回到Master，而且此時的Master的Ready值也為High，代表Read data channel也完成了這筆資料的Handshake。



* **Program Gate-Level-simulation Result(ALL PASS)**

☑syn\_0

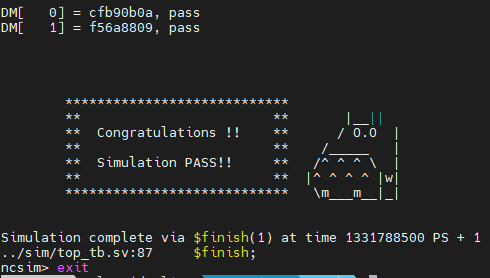


☑syn\_1

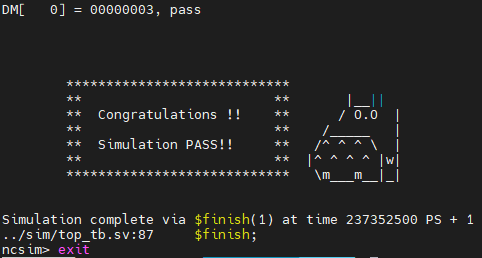
一張含有 文字 的圖片

自動產生的描述

☑syn\_2



☑syn\_3



* **Timing Report**

一張含有 文字 的圖片

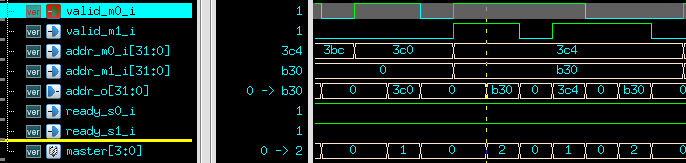
自動產生的描述

* **Area Report**

一張含有 文字, 螢幕擷取畫面, 室內 的圖片

自動產生的描述

* **Program 0 Instruction**
* 在上面的部份已經簡單介紹了當CPU想讀取Instruction Memory的資料時整個AXI BUS是如何建立起Master和Slave之間的溝通，而所有的指令都是透過相同的方式獲得的，因此在這個部分會著重於解釋當CPU要存取Data Memory時是AXI BUS如何運作的，以及當Instruction\_Master和Data\_Master同時都要讀取Slave時AXI BUS會如何處理
* 首先是Load Word指令，以3b8的lw t0, 0(t1)為例
* 當Mem\_stage在處理3b8指令時IF\_stage也正在處理3c4指令所以這2個Master同時會傳入一個Address和AWVALID訊號，而因為在我的架構中Data Master有較高的優先權，所以雖然2個slave的ARREADY訊號都為High，但是Arbiter會把IM\_Slave的ready訊號處理為low讓data\_master可以先傳訊號到slave中，如下圖所示雖然ready\_s0和ready\_s1都為high但是經過Arbiter之後，最後取得BUS的使用權的是master\_2，所以當完成handshake的時候slave收到的是Address是Master2的3c0



* 如下圖所示，雖然2個slave都回傳Ready，但是在經過Arbiter之後只有送到m1裡的ready為1，送到m0的ready為0，因此只有Data\_Master有完成Read Address的handshake，要到下一個cycle才會輪到Instruction Master使用BUS

一張含有 文字, 時鐘, 儀錶 的圖片

自動產生的描述

* 此時因為Instruction\_Master沒有完成handshake所以它的ARVALID要一直維持high，直到輪到它完成handshake後才會放下

一張含有 文字, 時鐘 的圖片

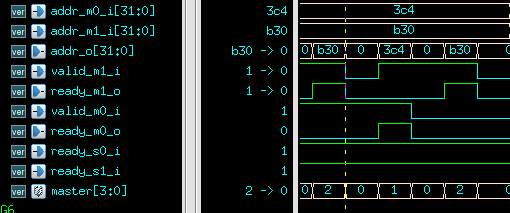
自動產生的描述

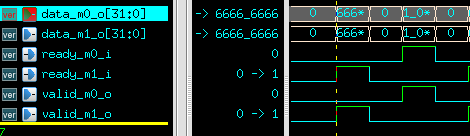
* 在Read Data Channel方面也是，因為Decoder知道Master現在要存取的是slave1所以只有送到Master1的valid訊號為1，而且也只有Master1的ready訊號為1，因此雖然Instruction\_memory和Data\_memory都有data送出，但是只有來自Data\_memory的data會通過Mux送回到Data\_Master中

一張含有 文字, 時鐘 的圖片

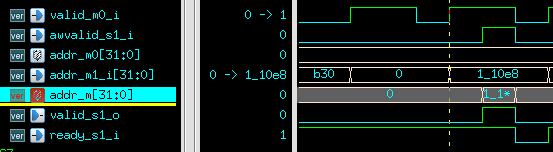
自動產生的描述

* 再下一個Cycle時就輪到Instruction\_Master使用BUS了，Address等signal的傳遞和上述說明相同

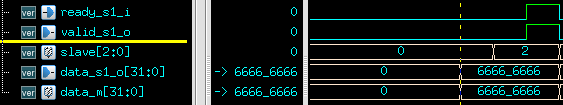




* 接著利用3c0的sw t0,0(sp)指令來說明CPU如何透過BUS對Slave進行寫入
* 當Mem\_stage要執行SW指令時，同時IF\_stage也要執行3cc的Instruction Fetch，也就是說此時CPU要同時對Instruction Memory做讀取並對Data\_Memory做寫入，而因為在AXI中Read和Write會占用不同的Channel，因此可以同時進行，於是這時候IM\_Master的ARVALID和DM\_Master的AWVALID會同時為high，於是乎在Write Address channel只有Master1的Address、Data以及其它signal會送到Slave中，而Write Channel中的Decoder也會根據Address判斷要寫入的為Slave\_1，所以會將Slave\_1的AWREADY訊號傳回到Master中，如下圖所示



* 接著在Write Data Channel中就會根據現在要寫入的data是哪個slave把data和valid訊號一起送到該slave裡，並且也從該slave收到ready訊號，以完成Handshake，結束這次的寫入動作



* 最後再透過BID選擇Master把Bvalid以及Bresp送到該Master中完成handshake確認

一張含有 文字, 時鐘 的圖片

自動產生的描述

* **Lesson Learned**
* 了解AXI BUS如何利用Handshake機制來確保5個Channel中資料的傳輸
* 了解AXI BUS channel中每個component如何運作來完成Master和Slave之間的溝通
* 學習使用Jaspergold 的VIP，來讓自己設計的RTL code可以符合spec中Master、Slave、Bridge運作的行為，並且可以知道assert、violation、和cover這3個result所代表的不同意義
* 了解CPU再接上Bus之後是如何利用BUS和SRAM做communication
* **Superlint**
* Total Lines of RTL code

一張含有 文字 的圖片

自動產生的描述

* Total warning/errors

一張含有 文字 的圖片

自動產生的描述