Advanced VLSI System Design (Graduate Level)

Fall 2024

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: \_\_黃于睿\_\_

Student ID: \_M16121158\_\_

# Summary

* 設計與驗證 AXI（Advanced Extensible Interface）通訊協議的功能，包括讀寫通道、Handshake流程及其順序性控制。採用 VALID/READY 訊號進行資料傳輸。傳輸僅在 VALID 和 READY 訊號皆為Level High時發生。
* Read有兩個channel（地址與數據），而Write有channel（地址、數據、Response）。各通道均有 VALID 和 READY 信號，確保資料傳輸同步。
* 支援（Outstanding Transactions）及（Out-of-Order Transactions），使資料能亂序亂序傳回，不需要等待當筆資料傳回才繼續，提高其傳輸效率。
* 使用 JasperGold Verification IP 驗證 CPU Wrapper、AXI 及 SRAM Wrapper 的功能。採用排序、乘法、最大公因數計算及浮點運算進行詳細驗證。

# Design Report

1. **RTL (All Pass)**

|  |  |  |
| --- | --- | --- |
| **Prog 0** | **Prog 1** | **Prog 2** |
|  |  |  |
| **Prog 3** | **Prog 4** | **Prog 5** |
|  |  |  |
| **Prog 6** |  |  |

1. **Gate Level Simulation (All Pass)**

|  |  |  |
| --- | --- | --- |
| **Prog 0** | **Prog 1** | **Prog 2** |
|  |  |  |
| **Prog 3** | **Prog 4** | **Prog 5** |
|  |  |  |
| **Prog 6** |  |  |

1. **PA Report**

|  |  |
| --- | --- |
| **Cycle Period** | **2ns** |
| **Area: 19607.410764** |  |

1. **JasperGold Superlint Coverage**

|  |  |
| --- | --- |
| **Coverage: (5396-129)/5396 = 97.61%** |  |

1. **JasperGold VIP Verification**

|  |  |
| --- | --- |
| Master |  |
| AXI Bridge |  |
| Slave  (No error) |  |

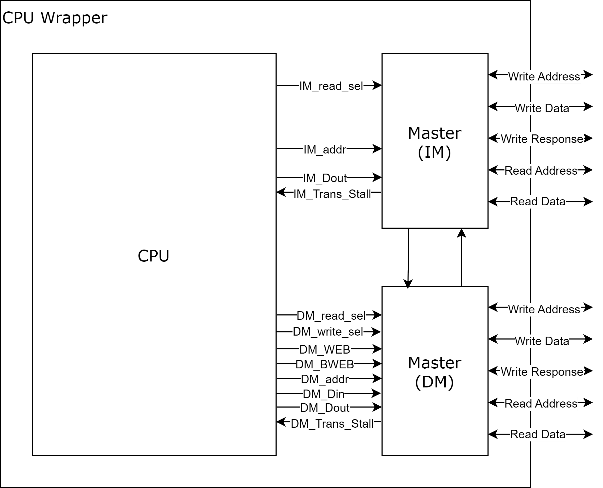
# Design Architecture-

下圖為本次設計完整設計圖。設計中主要以三部分組成，1) CPU Wrapper 2) AXI Bus 3) SRAM Wrapper，當中所能溝通主要通道分為5個 1) Read Address 2) Read Data 3)Write Address 4) Write Data 5) Write Response。



# Sub-module Functional Explanation

1. **CPU Wrapper**

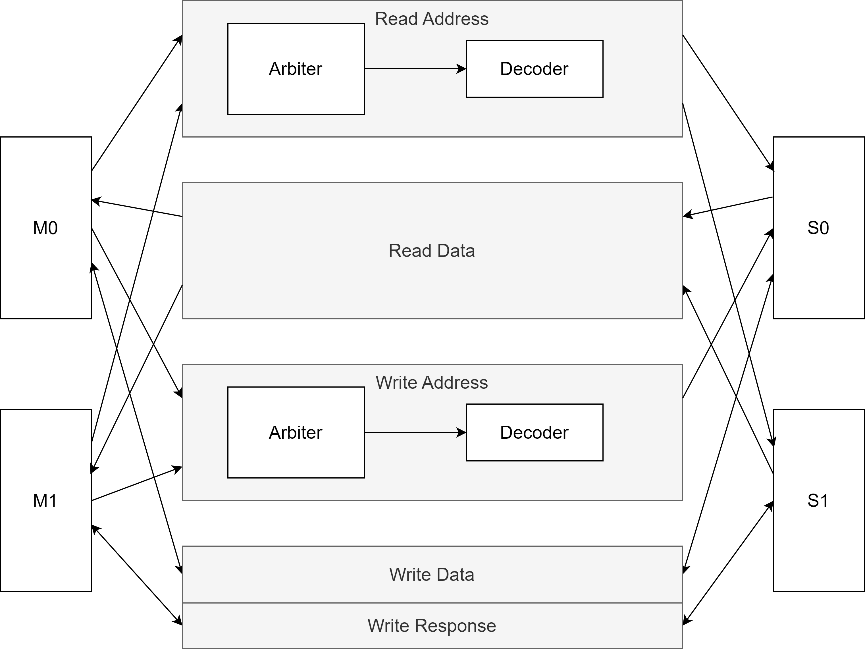
CPU Wrapper為主要建立CPU與他人溝通的(Master)橋樑。詳細設計如右圖所示。其中組成分為3大部份1) CPU 本體架構 2) Master module for IM 3) Master module for DM。由2) 3) 分別負責IM及DM的Handshake訊號、Address、以及Data傳送、接收。

1. CPU
2. Master module for IM/Master module for DM

狀態機中共有5個State，分別為IDLE、RADDR、

1. **AXI bus**

本次設計上採用Shared Address Multiple Data (SAMD)方式，內部簡易連接圖如下所示:



* 1. **Read /Write Address**

此部分由2部分組成，1) Arbiter、 2) Decoder。由於為SAMD方式進行設計，Arbiter

* 1. **Read Data**

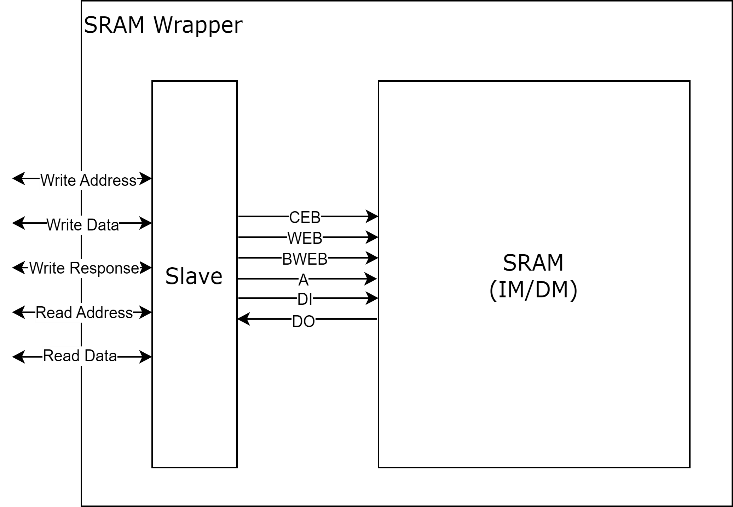
此部分說明Read Data設計為由Master 端發起讀取請求，Handshake 成功後，將資料從Slave傳送至Master端。

* 1. **Write Data**

此部分說明Write Data設計為由Master 端發起讀取請求，Handshake成功後，將資料從Master傳送至 Slave端。

* 1. **Write Response**

此部分說明Write Channel 在傳輸後，需要回覆是否此筆資料有寫入成功。

1. **SRAM Wrapper**

SRAM Wrapper為主要建立Memory與他人溝通的(Slave)橋樑。詳細設計如右圖所示。其中組成分為2大部分，1) SRAM Slave 2) IM/DM(SRAM)。由1) 3) 負責接收處理來自Master端的Handshake訊號、Address、以及Data傳送、接收。

1. **資料傳輸衝突問題**

下表中有為資料若同時須進行時，需要注意的可能

|  |  |
| --- | --- |
| **Read/Read** | 此狀況同時使用Read Channel，須利用AXI 內 Round Robin  方式進行輪流，在DM進行Address Handshake後，即可接續IM進行Address Handshake。 |
| **Read/Write**  **(不同Slave)** | 此狀況由於Write 狀態機比Read Channel多一個Response狀態，故Read Channel需要多等1 cycle，已讓設計中Stall對齊，才不會發生進入deadlock。 |
| **Read/Write**  **(同Slave)** | 此狀況同時使用Read Channel，由於並非相同通道，故有優先權問題。DM優先權比IM高，故先讓DM進行傳輸後，才會換IM。 |
| **Write/Write** | 此作業不會有此狀況 |

# Waveform Verification

1. **CPU Wrapper**
   1. **CPU**

此處會將IM、DM所需的訊號送出。由於IM在作業二都僅負責進行讀取動作，故下圖以DM進行舉例。

* **DM Read**

下圖為舉實際CPU進行Read舉例，當PC\_EXE\_in (在EXE Stage進行的pc)(0x4b4)，故MEM Stage 需減4(執行0x4b0)，此時有Read (active high)發生，並WEB == 1’b1 (active high)，此時所要傳輸的address為 DM\_addr (0x1\_10f8)。對應要輸出的Data為 DM\_Dout(0x6666\_6666)。同時CPU端會收到需要Stall的訊號。

一張含有 螢幕擷取畫面, 軟體, 多媒體軟體, 電子產品 的圖片

自動產生的描述

* **DM Write**

下圖為舉實際CPU進行Write舉例，當PC\_EXE\_in(在EXE Stage進行的pc)(0x10c)，故MEM Stage 需減4(執行0x108)，此時有write(active high)發生，並為全部Bit都需寫入🡺BWEB == 32’b0 (active low)，此時所要傳輸的address為 DM\_addr(0x1\_010c) 。對應要傳輸的Data為 DM\_Din(0x0)。同時CPU端會收到需要Stall的訊號。

一張含有 螢幕擷取畫面, 文字, 行, 多媒體軟體 的圖片

自動產生的描述

* **CPU 內部新增 – Hazard 對Pipeline register 控制**

Hazard 新增了Stall控制訊號，當從Master端發出IM或DM的Stall訊號，代表CPU運作需暫停，此狀況會將pipeline register 都進行鎖住CPU將會維持當下做的指令，直到雙方Stall皆放下時，pipeline register才會更新。

一張含有 螢幕擷取畫面, 多媒體軟體, 軟體, 繪圖軟體 的圖片

自動產生的描述

* 1. **Master (CPU\_wrapper內)**
* **狀態機:**共分為6個狀態，可分為

**IDLE (1):** 在未進入Read、Write 狀態時，在此等候CPU傳來發起Read/Write，為各別觸發對應的Start\_burst\_(read/write)。

1. **進入Read 狀態:**

下圖為舉實際CPU進行Read舉例，當PC\_EXE\_in(在EXE Stage進行的pc)(0x6c)，故MEM Stage 需減4(執行0x68)，此時有Read(active high)發生，並可看到Start\_burst\_read被進行觸發，狀態機的S\_nxt也正確被進行進入Read模式(State = 1)。此時在(State = 1)的時候，ARValid才會正式啟動。

一張含有 螢幕擷取畫面, 多媒體軟體, 軟體 的圖片

自動產生的描述

1. **進入Write 狀態:**

下圖為舉實際CPU進行Write舉例，當PC\_EXE\_in(在EXE Stage進行的pc)(0x10c)，故MEM Stage 需減4(執行0x108)，此時有Write (active high)發生，並可看到Start\_burst\_Write被進行觸發，狀態機的S\_nxt也正確被進行進入Write模式(State = 3)。此時在(State = 3)的時候，AWValid才會正式啟動。

**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**

**Read (2):**

下圖為舉實際CPU進行Read舉例，當PC\_EXE\_in(在EXE Stage進行的pc)(0x6c)，故MEM Stage 需減4(執行0x68)，此時有Read(active high)發生，並可看到Start\_burst\_read被進行觸發，狀態機的S\_nxt也正確被進行進入Read模式(State = 1)。此時在(State = 1)的時候，ARValid才會正式啟動，等待ARReady 進行回覆，當接收到訊號後，啟動RReady，等待RValid傳送成功Handshake，Handshake成功時可看到RData開始傳輸(0xf13)。

一張含有 螢幕擷取畫面, 多媒體軟體, 軟體 的圖片

自動產生的描述

**Write (3):**

下圖為舉實際CPU進行Write舉例，當PC\_EXE\_in(在EXE Stage進行的pc)(0x10c)，故MEM Stage 需減4(執行0x108)，此時有Write (active high)發生，並可看到Start\_burst\_write被進行觸發，狀態機的S\_nxt也正確被進行進入Write模式(State = 3)。此時在(State = 3)的時候，AWValid才會正式啟動，等待AWReady 進行回覆，當接收到訊號後，啟動WValid，等待WReady傳送成功Handshake，Handshake成功時可看到WData開始傳輸(0xf13)，傳完後進入Response模式(State = 5)，來確認是否有寫成功。

**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**

* **資料衝突 狀態機對應:**

1. **Read/Read**

下圖為舉實際CPU進行IM Read/ DM Read舉例，當PC\_EXE\_in(在EXE Stage進行的pc)(0x208)，故MEM Stage 需減4(執行0x204)，此時有Read/ Read (active high)發生，並可看到Start\_burst\_Read皆被進行觸發，狀態機的S\_nxt也正確被進行進入Read模式(State = 1)。此時在(State = 1)的時候，可以看到 DM Read🡺State = 1較快完成，直接進入狀態2，並等待IM做完後在一起結束狀態。

**一張含有 文字, 螢幕擷取畫面, 電子產品, 軟體 的圖片

自動產生的描述**

1. **Read/Write(不同Slave)**

下圖為舉實際CPU進行IM Read/ DM Write舉例，當PC\_EXE\_in(在EXE Stage進行的pc)(0x10c)，故MEM Stage 需減4(執行0x108)，此時有Read/ Write (active high)發生，並可看到Start\_burst\_Read及Start\_burst\_Write皆被進行觸發，狀態機的S\_nxt也正確被進行進入Read模式(State = 1) 及Write模式(State = 3)。此時在IM因為少一狀態，故相 較DM會早一拍完成，如果不進行處理時，會導致連續讀寫中出現錯誤。可以看到 IM Read🡺State = 0晚1cycle進入狀態，並與DM同時做完後在一起結束狀態(State = 2 & State = 5)。

**一張含有 螢幕擷取畫面, 文字 的圖片

自動產生的描述**

1. **Read/Write(同Slave)**

下圖為舉實際CPU進行IM Read/ DM Write舉例，當PC\_EXE\_in(在EXE Stage進行的pc)(0x148)，故MEM Stage 需減4(執行0x144)，此時有Read/ Write (active high)發生，並可看到Start\_burst\_Read及Start\_burst\_Write皆被進行觸發，狀態機的S\_nxt也正確被進行進入Read模式(State = 1) 及Write模式(State = 3)。此時因為兩者傳送目標Slave相同，故在IM需要等待DM做完後才能接續，故相較DM，IM(State = 0)較晚進入(State = 1)。如果不進行處理時，會導致連續讀寫中出現錯誤。並與DM同時做完後在一起結束狀態(State = 2 & State = 5)。

**一張含有 螢幕擷取畫面, 文字, 設計 的圖片

自動產生的描述**

1. **SRAM Wrapper**
   1. **Slave Wrapper**

* **狀態機:**共分為4個狀態，可分為

**Read (2)**

下圖為舉實際CPU進行Read舉例，當PC\_EXE\_in(在EXE Stage進行的pc)(0x13c)，故MEM Stage 需減4(執行0x138)，此時有Read(active high)發生，狀態機的S\_nxt也正確被進行再SADDR模式(State = 0)。此時在(State = 0)的時候，等待Addr Handshake成立才會正式啟動，等當接收到訊號後，啟動RValid，等待RReady傳送成功Handshake，Handshake成功時可看到RData開始傳輸(0xffe0\_0313)，此時可以看到CEB啟動(active low)，讀取位置A(0x50)，並在DO送值出來(0xffe0\_0313)。

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

**Write (3)**

下圖為舉實際CPU進行Write舉例，當PC\_EXE\_in(在EXE Stage進行的pc)(0x54c)，故MEM Stage 需減4(執行0x548)，此時有Write (active high)發生，並可看到Start\_burst\_write被進行觸發，狀態機的S\_nxt也正確被進行進入Write模式(State = 3)。此時在(State = 3)的時候，AWValid才會正式啟動，等待AWReady 進行回覆，當接收到訊號後，啟動WValid，等待WReady傳送成功Handshake，Handshake成功時可看到WData開始傳輸(0xd9d)，傳完後進入Response模式(State = 5)，來確認是否有寫成功。此時可以看到CEB啟動(active low)，寫入位置A(0x13)，並在DI送值送進Memory (0xd9d)。

一張含有 螢幕擷取畫面, 文字, 軟體 的圖片

自動產生的描述

1. **Bridge**
   1. **Read /Write Address**

下圖為舉實際CPU進行Read舉例，當PC\_EXE\_in(在EXE Stage進行的pc)(0x54c)，故MEM Stage 需減4(執行0x548)，此時有Read (active high)發生，並可看到M0\_ARValid啟動，等待ARReady 進行回覆，但M1\_ARValid並沒有進行請求。

**Arbiter:**

可以看到Arbiter 的Master\_sel(1)，代表成功選擇M0\_ARAddr， Handshake成功時可看到O\_Addr開始傳輸(0x550)。

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

**Decoder:**

可以看到Decoder皆收來自Arbiter的輸入後，利用Address決定送自何處，下圖為成功送至S0

一張含有 螢幕擷取畫面, 文字, 陳列, 多媒體軟體 的圖片

自動產生的描述

* 1. **Read Data**

下圖為舉實際CPU進行Read舉例，當PC\_EXE\_in(在EXE Stage進行的pc)(0x54c)，故MEM Stage 需減4(執行0x548)，此時有Read (active high)發生，並可看到M0\_RValid啟動，等待RReady 進行回覆，但M1\_RValid並沒有進行請求，並轉送至S0\_RValid。可看到其讀出Data為(0x44\_0413)

**一張含有 螢幕擷取畫面, 軟體, 多媒體軟體, 文字 的圖片

自動產生的描述**

* 1. **Write Data**

下圖為舉實際CPU進行Write舉例，當PC\_EXE\_in(在EXE Stage進行的pc)(0x4e0)，故MEM Stage 需減4(執行0x4dc)，此時有Write (active high)發生，狀態機的S\_nxt也正確被進行進入Write模式(State = 3)。並可看到S1\_WValid啟動，等待S1\_WReady 進行回覆，並轉送至M1\_WValid，回覆M1\_WReady。可看到其寫入S1 Data為(0xcccc\_cccc)

**一張含有 螢幕擷取畫面, 多媒體軟體, 軟體, 繪圖軟體 的圖片

自動產生的描述**

* 1. **Write Response**

下圖為舉實際CPU進行Write舉例，當PC\_EXE\_in(在EXE Stage進行的pc)(0x4e0)，故MEM Stage 需減4(執行0x4dc)，此時有Write (active high)發生，狀態機的S\_nxt也正確被進行進入Write模式(State = 3)，最後進入Write Data模式(State = 4)前，Master端就已經將Response Ready拉起，等待最後進入Response模式(State = 5)，Slave端將Response Valid拉起。

**一張含有 螢幕擷取畫面, 多媒體軟體, 繪圖軟體, 軟體 的圖片

自動產生的描述**

# Lessons Learned

* **Verilog 語法:**

|  |  |
| --- | --- |
| 設計系統化: | 有用上macro 去對參數進行設計跟檔案include。這次再加上使用LAB已經定義好的AXI Include File，這當中以前只會把它當參數使用，這次才知道其實也可以當作bit數直接使用。  在設計上也在部分地方嘗試使用function，去對會重複使用的功能直接再利用。 |
| 語法學習: | 在課堂上有教到if 的priority 問題，並用unique if來做設計上的優化 |
| Interface: | 由於接下來在AXI中相似線路將越來越多，將會採用Interface進行設計，來縮減在設計中容易線路接錯問題 |

* **Tool使用:**

|  |  |
| --- | --- |
| Superlint 使用: | 用來檢查是否有過多語法錯誤，可能在宣告時常常寫得不夠嚴謹，或是有設計上的疏漏時就很好用。 |
| JasperGold VIP: | 原先設計的bus 傳輸其實會有很多問題，經過vip的驗證後，會發現有些design 在設計中並想的那麼周全。 如 channel default case 因為設計時多寫處理Default Slave以外的狀況，所以導致有訊號提在被下拉，導致錯誤。 |

* **AXI Bus + Wrapper設計**

一開始設計時並沒有想那麼多，覺得應該不會有資料衝突問題，頂多需要在Arbiter內進行輪流處理即可，但這限於個別的Channel，並沒有去處理到不同Channel 間的問題。後來才發現其實也有可能讀(IM)、寫(DM)同時發生，需要有優先權的問題。尤其在下表Read/Write(同Channel狀態下)，此進發生在Program 1中，在設計中很面才想起，需要再Slave端進行控制。

|  |  |
| --- | --- |
| **Read/Read** | 此狀況同時使用Read Channel，須利用AXI 內 Round Robin  方式進行輪流，在DM進行Address Handshake後，即可接續IM進行Address Handshake。 |
| **Read/Write**  **(不同Slave)** | 此狀況由於Write 狀態機比Read Channel多一個Response狀態，故Read Channel需要多等1 cycle，已讓設計中Stall對齊，才不會發生進入deadlock。 |
| **Read/Write**  **(同Slave)** | 此狀況同時使用Read Channel，由於並非相同通道，故有優先權問題。DM優先權比IM高，故先讓DM進行傳輸後，才會換IM。 |
| **Write/Write** | 此作業不會有此狀況 |

* **線路優化設計:**

在Floating point 及CSR 設計上，為了避免增加過多選擇器及控制訊號中，rd部分將其與pc及Integer ALU選擇調整到同一個Mux，在控制訊號處理上要更加小心，並且要搭配register file寫到哪一個都要同步處理，但整體效果算不錯的，不用再多加很多層Mux去處理。

在Floating Point LW/SW 的address計算，由於運算上皆與原本的LW/SW相同，故僅在最後的data利用Control Signal 進行處理，這樣可以減少需要再拉線路去FP\_ALU重複的運算。