VLSI System Design (Graduate Level)

Fall 2023

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

**Contents**

[Summary 3](#_Toc149223010)

[Contribution 3](#_Toc149223011)

[AXI bus explanations 4](#_Toc149223012)

[1. Master 4](#_Toc149223013)

[2. Slave 5](#_Toc149223014)

[3. Bridge 6](#_Toc149223015)

[4. CPU 架構調整 7](#_Toc149223016)

[Verification IP 9](#_Toc149223017)

[1. Master 9](#_Toc149223018)

[2. Slave 9](#_Toc149223019)

[3. Bridge 10](#_Toc149223020)

[Pre-Simulation(wave form) 11](#_Toc149223021)

[Superlint 15](#_Toc149223022)

[Synthesis report 15](#_Toc149223023)

[Post simulation(pass結果) 18](#_Toc149223024)

[Lessons learned 23](#_Toc149223025)

# Summary

本次作業中共分兩個部分，第一部份，我們完成了符合AXI 協定的Master、Bridge和Slave的設計，並通過VIP的驗證。第二部分，我們將RISC V CPU接上我們設計的AXI、完成6個program的測試。最後經過合成，可達到clock period為8ns的performance。

報告中，首先會說明本次作業中的分工，接著介紹AXI的狀態機設計及VIP的驗證和調整方法，還有CPU是如何被調整的，以及6個program的測試結果。最後則會說明我們superlint的調整和結果、合成報告和後模擬通過圖。

# Contribution

吳欣航(N26124395) : 50%

AXI Master設計、AXI Slave設計、CPU架構調整、共同Debug、VIP調整

呂修麟(N26124272): 50%

AXI Bridge設計、CPU架構調整、共同Debug、VIP調整

# AXI bus explanations

在本次作業中，Master 端是 CPU wrapper。而在 wrapper 中，有 CPU 和兩 個 AXI Master 的 module，兩個 AXI master module 分別負責 CPU 對 IM和 DM的存取，需要特別注意的是CPU會針對不同地址對IM和DM作讀取，而寫入只會寫到DM。

## Master

在本次作業中Master即為CPU端，將Master的Read和Write分開設計，分別有Master\_Read與Master\_Write兩個Module。下圖為Master\_Read和Master\_Write的狀態機，兩個的初始狀態都是FREE，若ARVALID拉起來後會跳到下一個狀態，若ARREADY還沒拉起的話，會到WAIT\_HS等待握手，要握手才出的去，握完手則可以開始對Slave傳送address。

在FREE中，CPU會一直跟Master說傳送的資訊，如ARID, ARADDR, ARVALID，並使用暫存器將地址存起來，等到ARVALID拉起來後，address的值會一直保持為暫存器的值，值到傳送結束，ARVALID則是透過CPU發送控制訊號，若ARVALID拉起，MASTER會傳送stall\_IM指令到CPU，叫CPU保持不動，直到動作結束。

WAIT\_HS則是等待握手，要注意的是若ARVALID拉起後不能隨便放下。SEND\_ADDR則是握手完成，會將address傳到slave，並等待Slave傳資料回來，傳回來用RREADY, RVALID, RLAST來判斷是否傳完，傳完才能回到FREE state。

一張含有 圓形, 圓球, 夜晚, 月亮 的圖片

自動產生的描述

Write與Read差不多，只是Write多了一個state用來傳輸response資料，會在FREE把CPU要給出的資訊keep住，SEND\_DATA則會把資料傳給Slave，等到都傳完後會將BVALID拉起，進到下一個state SEND\_B等待BREADY拉起傳送response回來。同樣除了FREE state，皆會發出stall\_W訊號。

一張含有 圓形, 圓球, 夜晚, 月亮 的圖片

自動產生的描述

## Slave

Slave同樣分為Slave\_Read和Slave\_Write兩個module，預設一樣會在FREE state。以Slave\_Read來說，若ARVALID和ARREADY拉起後會到下一個state，這個state經過一個cycle就會跑到下一個state，因為需要等待SRAM的delay，針對不同的記憶體可以有不同的設計。爾後進到DATA，在這一個state，Slave會收到SRAM給回來的data，在這一個state將他傳送回Master。

一張含有 圓形, 圓球, 天體, 夜晚 的圖片

自動產生的描述

Slave\_Write也同樣有3個state，一開始也是在FREE state，握手後接收到資料WLAST拉起後，進到下一個state SEND\_B傳送response資訊。一張含有 圓形, 圓球, 天體, 月光 的圖片

自動產生的描述

## Bridge

Bridge分為 Bridge\_read和Bridge\_write。Bridge\_read和Bridge\_write的狀態機只有FREE和DATA，在FREE狀態，會去偵測來自MASTER的ARVALID和AWVALID訊號，並透過判斷ARADDR或AWADDR是否要進入DATA的狀態;在DATA，Bridge會讓其與對應的SLAVE進行handshake，然後偵測RLAST、RVALID或是BREADY、BVALID來判斷是否Master和Slave通聯已經結束，可以回到FREE的狀態。

**Bridge\_read** 負責 AR和R channel 的通道管理，在這個作業中，有兩個Master和兩個Slave，雖然負責IM的Master不會去read DM，但是為了架構的完整性和VIP的正確度，Bridge在設計上要支持兩個Master去分別讀取兩個Slave的邏輯判斷，透過一個reg去記住上一個掌控AXI的Master，並且透過ARREADY和ARVALID去判斷哪一個Master和Slave被選擇。

一張含有 圓形, 螢幕擷取畫面 的圖片

自動產生的描述

**Bridge\_write** 負責 WR、W和B channel 的通道管理，在這個作業中，有兩個Master和兩個Slave，為了架構的完整性和VIP的正確度，Bridge在設計上要支持兩個Master去分別寫入兩個Slave的邏輯判斷，透過AWREADY和AWVALID去判斷哪一個Master和Slave被選擇。

一張含有 圓形, 螢幕擷取畫面 的圖片

自動產生的描述

## CPU 架構調整

這一次作業使用吳欣航同學的CPU作修改，主要的修改有三個點，分別是stall的處理，以及Read/Write signal判斷，除此之外架構依舊。Read signal在IM時都是設為1，而在DM時使用opcode進行判斷，若M\_op=LOAD的話就將read signal設為1。Write signal也同樣使用M\_op進行判斷，M\_op=STORE會將write signal拉起。

由於原先設計針對SRAM有一個cycle delay作處理，這一次作業將CPU與SRAM溝通用AXI取代，由於讀取資料的cycle一定大於1，因此不用對SRAM delay作處理，正確地來說，SRAM delay問題在AXI設計已處理，因此可以將拿到的資料直接存到pipeline register。

Stall的處理在本次作業是接收到read signal或是write signal後會一直拉起，直到在XLAST訊號拉高時拉低，只有在stall\_IF(全部Master的stall訊號)為low的時候CPU會動作，這邊的stall\_IF行為跟原CPU的stall訊號有些不一樣，stall\_IF訊號將所有registers 保持住就好，因此所有有用到sequential的電路皆需要特別處理。

若有兩組需要讀取，由於AXI只能有一組通過，因此需要針對此狀況作設計，若是M0與M1同時都要讀資料的話，假設先讓M1通過，M0進到非FREE的state時會將stall拉起，這邊CPU會一直keep住資料，爾後使用M1的rvalid去判斷要不要拿資料，此時M1的stall依然拉著，因此不能進行運算，但需要將拿好的資料存進reg裡面，並且keep住，直到下一個instruction才能使用更新過的值。Stall\_IF會在RLAST傳回的那個cycle拉低並進行運算，運算完進到下一個指令。

一張含有 文字, 圖表, 方案, 工程製圖 的圖片

自動產生的描述若是同時讀跟同時寫，我們為了簡化設計流程，將讀寫所需要的cycle訂為一致，如此一來就可以將read和write對齊，因此不用特別對這個狀況作處理。

# Verification IP (Problem 1)

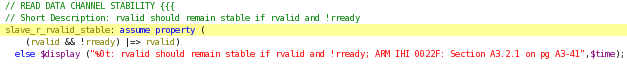
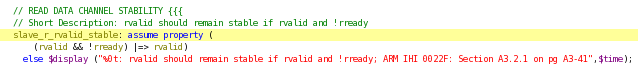
本設計的Master, Bridge, Slave都通過JasperGold ABVIP的驗證，以下為驗證結果。

## Master

Master 因為設計因此有部分 Cover 無法被滿足。在設計中，只要完成 AR/AW/W的握手，下個 cycle 就會將 READY 訊號拉高，因此無法測試到 Slave在 handshake前要保持stable的部分，總共的Cover錯誤數共有13個，皆是因為此原因，而cex則為0。

一張含有 文字, 螢幕擷取畫面, 數字, 字型 的圖片

自動產生的描述



## Slave

在原先設計中，會有無法測試到 Master在 handshake前要保持stable的部分，因為AXREADY在都是保持為1，因此測不到AXREADY為0的狀況。針對這個狀況作了一些修改，在SRAM Wrapper中加了一個狀態機，這個狀態機可以透過select訊號選擇此操作是read還是write，若是read的話，WREADY則會被設為0，反之亦然操作是write的話，RREADY則會被設為0。透過這個狀態機可以成功將uncover變成cover。Cex和uncover皆為0。

一張含有 文字, 螢幕擷取畫面, 數字, 字型 的圖片

自動產生的描述

## Bridge

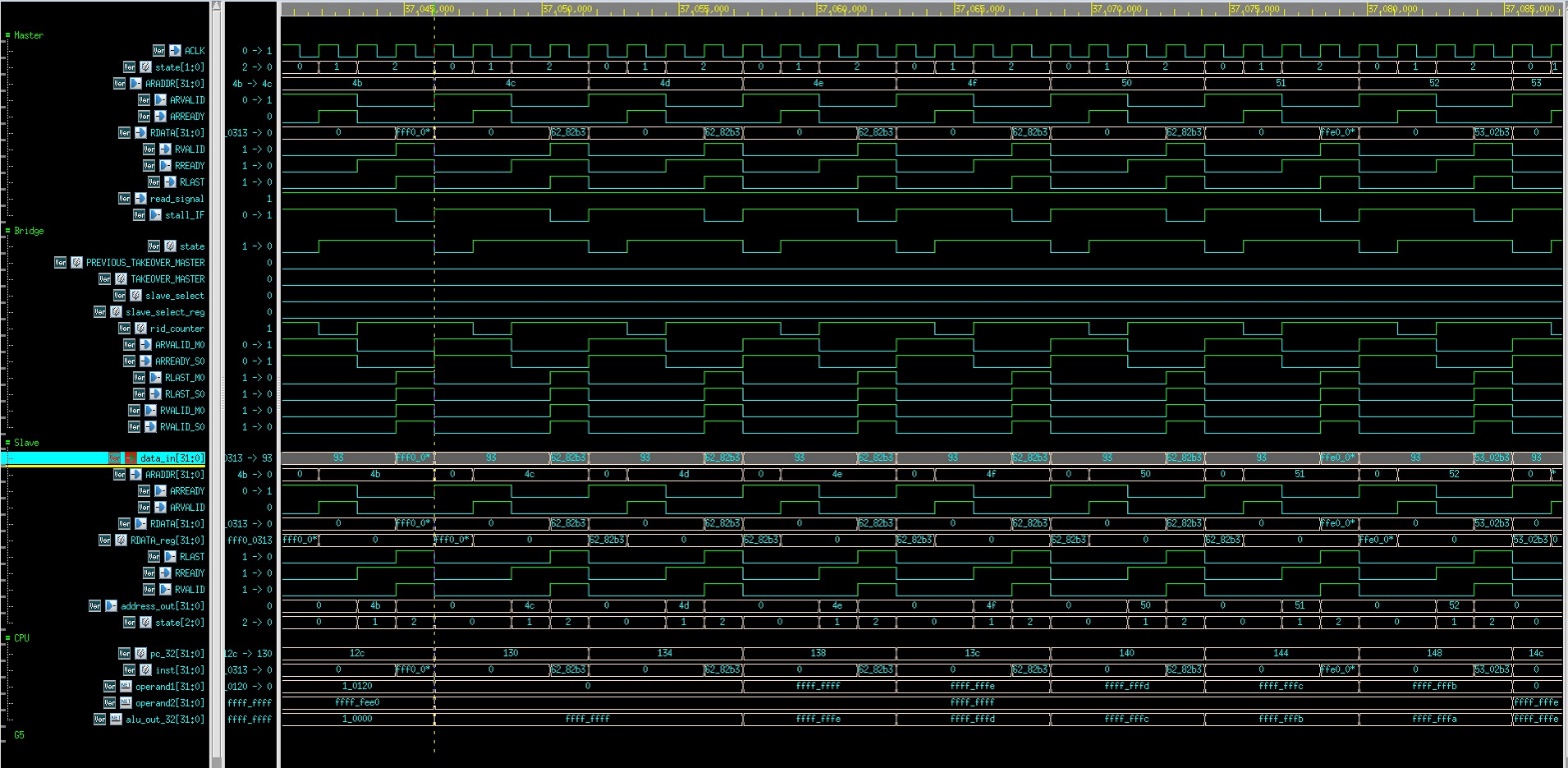
在原本設計中，只有針對homework的CPU架構去作撰寫，並沒有像spec上的AXI一樣靈活，因此遇到了許多uncover，我修改Bridge的邏輯判斷讓每個MASTER都有機會去存取每個SLAVE後，就解決了這些uncover。我也發現MASTER的ARADDR在VIP中，在handshake後可能換自己改變，我因此增加了reg去儲存有效的ARADDR，才避免了錯誤。我還發現許多問題難以解決，最後發現是線沒接好導致的。Cex和uncover皆為0。一張含有 文字, 螢幕擷取畫面, 數字, 字型 的圖片

自動產生的描述

# Pre-Simulation(wave form)

在這一章節我們會針對透過AXI對SRAM讀取的不同狀況作討論，分別是只讀IM，和同時讀IM以及DM，還有同時讀IM和寫DM，使用較為巨觀的波型作說明。

## Only read IM



在這個情況下我們討論了指讀取IM的狀況。由波型圖可以觀察到，當Master 握手後(紅色圈圈)，state會進入到第2個state，這時候Slave會進入到第1個state，向SRAM傳送地址，1個cycle後進到下一個state，Slave就會讀取到SRAM給的data，在同一個cycle透過Bridge將資料傳給Master，由於收到RLAST，因此Master會將stall\_IF拉低(黃色圈圈)，讓CPU可以繼續作運算，並在下一個cycle回到FREE。我們的設計是在CPU給出地址後，3個cycle資料會回來。

## Read IM and read DM

一張含有 螢幕擷取畫面, 陳列, 軟體, 多媒體軟體 的圖片

自動產生的描述

一張含有 螢幕擷取畫面, 行, 鮮豔, 平行 的圖片

自動產生的描述

圖中可以看到，Master 0 (IF stage) 和 Master 1(MEM stage)的arvalid同時拉起，此時Bridge會根據其內部先前儲存的reg，判斷要選擇Master 1先作處理，原理和上一小節一樣，接收到arready時握手(黃色圈圈)，經過3個cycle後，Slave1會把資料從DM抓到並傳回CPU，這時CPU會用暫存器將值存起來，由於此時stall\_IF還是拉高的狀況，因此我們拉了RLAST出去給CPU作判斷可不可以存值，但要注意的時這個更新後的值是要在下一個instruction進來後才能作運算(淺黃色箭頭)。

在抓取DM資料的時候，由於Bridge是通Slave1，因此Master0將看不到ARREADY，因此會進到state 1保持著，直到DM做完處理，Bridge將通道變成可以通Master0，才會進到握手環節(白色圈圈)，拿到資料後會將stall\_IF拉低，通知CPU可以開始作運算。

## Read IM and write DM

一張含有 電子產品, 螢幕擷取畫面, 電路, 電腦 的圖片

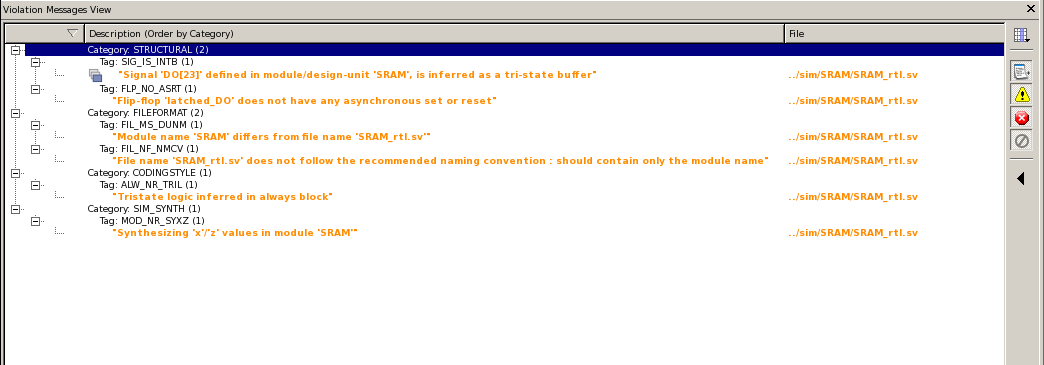
自動產生的描述一張含有 螢幕擷取畫面, 行, 軟體, 電子產品 的圖片

自動產生的描述

上圖中我們以Master 0作為Read的參照，由於AXI有R和W通道，因此我們為了簡化設計，將read和write所需的cycle數設計成一樣，如此一來就可以對齊，read和write皆是在送地址資料後3個cycle stall會拉下，CPU因此能進行運算。

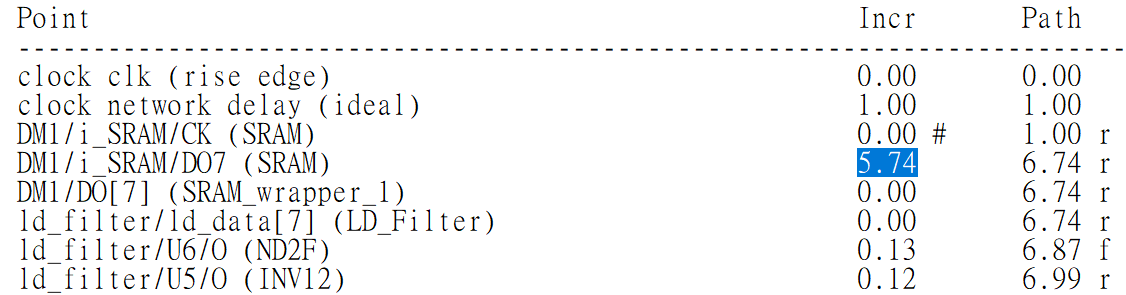
# Superlint

在superlint中，我們修正了許多case或是if沒有完全賦值或是缺少else的問題，我們也修正了MASTER和SLAVE的ID寬度不同而導致的width unmatched的問題，除此之外並沒有嚴重的coding style問題。我們的superlint在經過修正後，除了SRAM的部分以外，均無問題，因此通過率為100%。

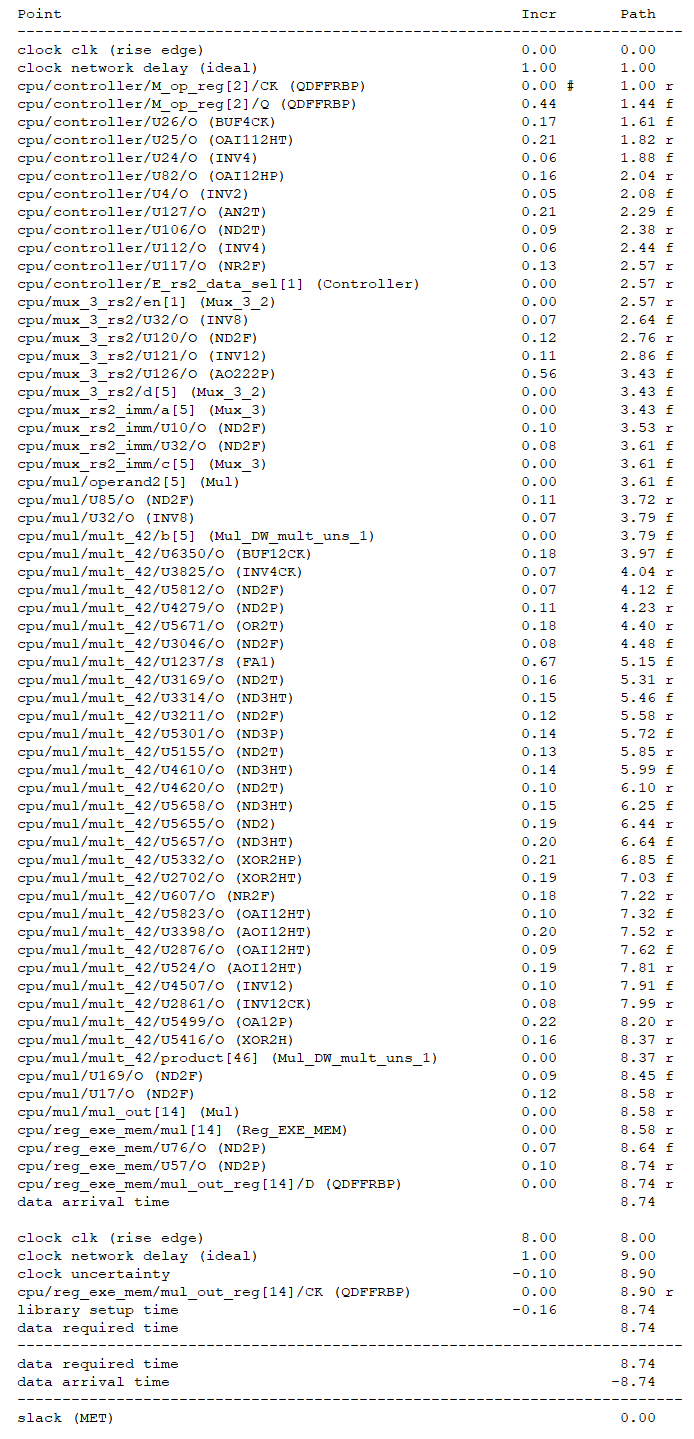
****

# Synthesis report

在這次作業我們發現電路cycle time可以壓到比作業1更低，但我們的CPU架構只有小改而已，原先以為是工作站變強了，但之後檢查作業1的timing report發現作業1為直接接到SRAM，因此SRAM的預設delay就會影響critical path甚多，如下圖5.74。原先作業1 cycle time為13ns，現在則可以跑到8ns，符合我們的預期。

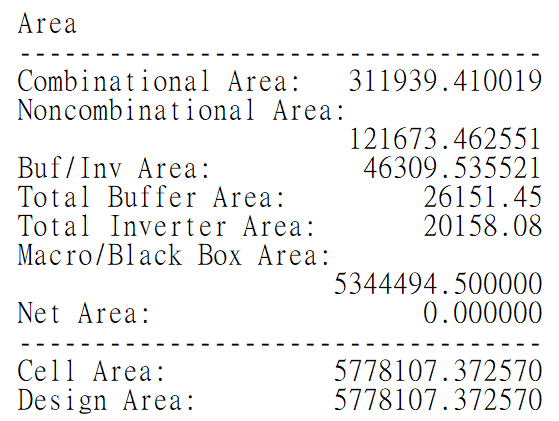


## Timing Report



電路合成的cycle time可以壓到8，並且可以看到slack為0，表示沒有出現violation。

## Area Report

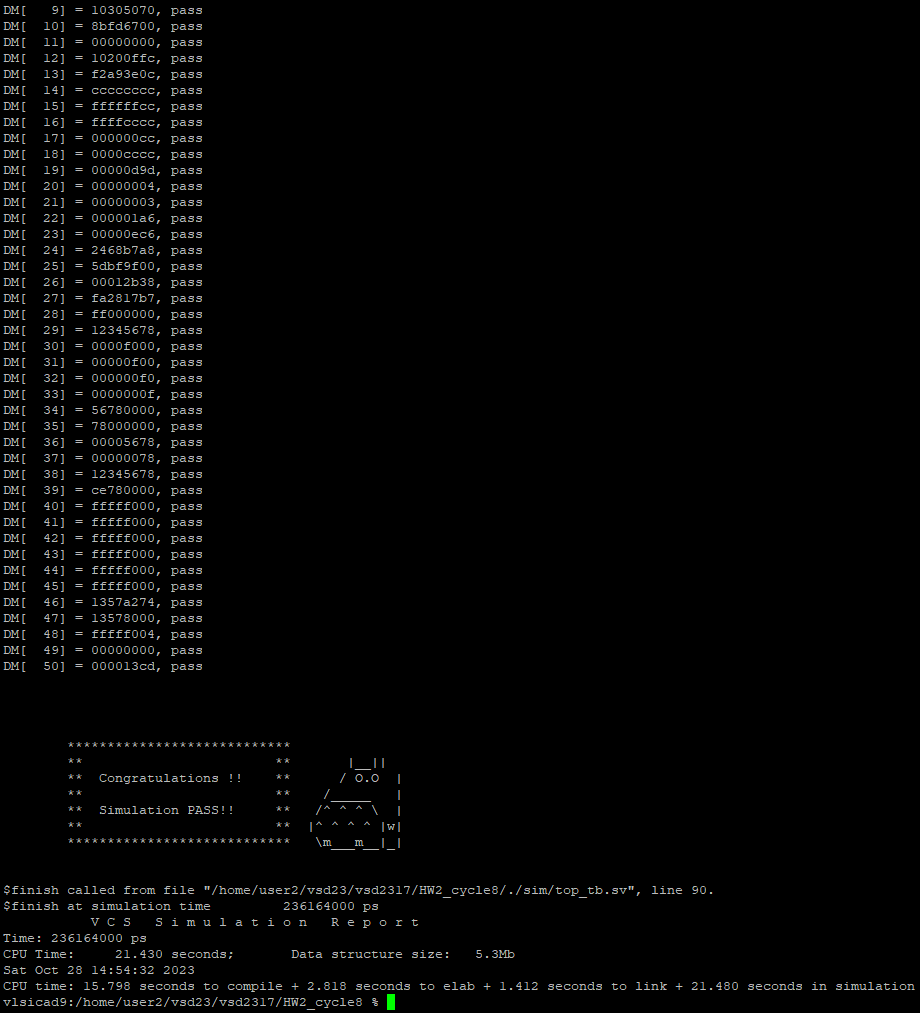


合成後的面積為5778107。

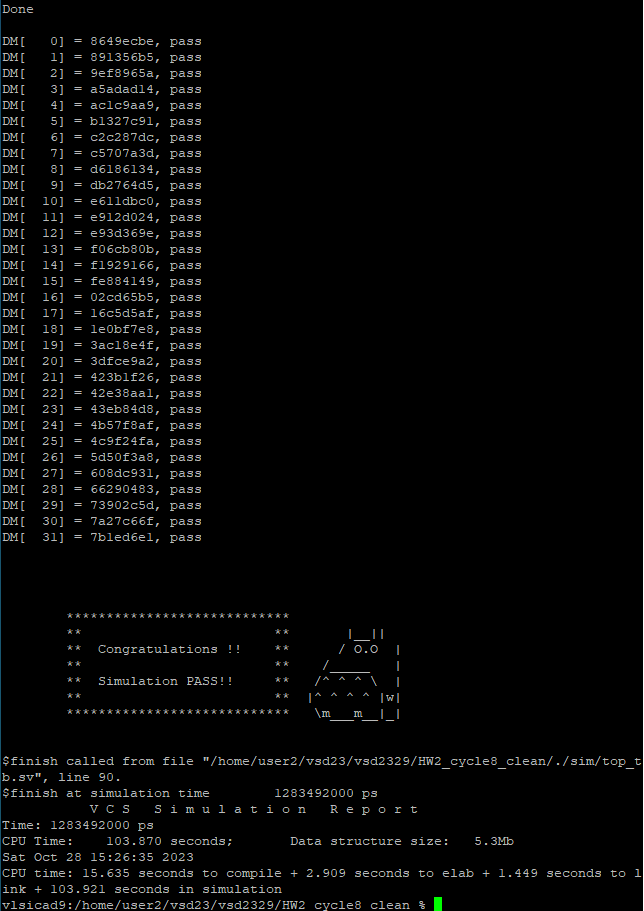
# Post simulation (Problem 2)

合成後我們通過Prog0~Prog5的測資，Prog0~Prog5與HW1基本雷同。

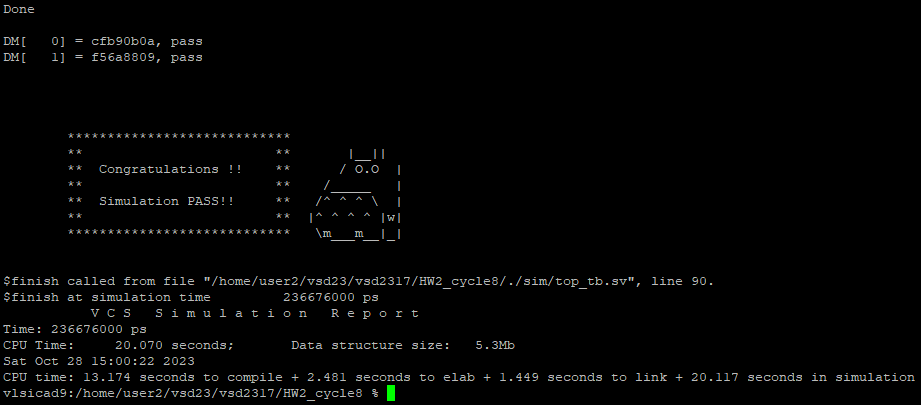
## Prog0



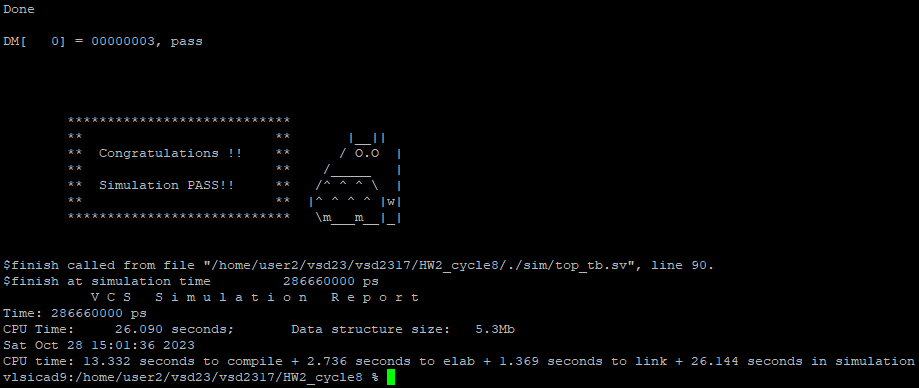
## Prog1



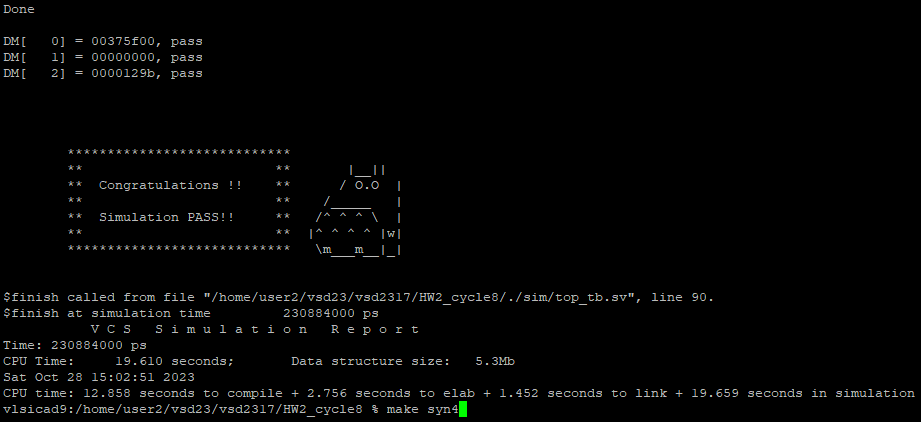
## Prog2



## Prog3



## Prog4



## Prog5



# Lessons learned

吳欣航:

本次作業是在沒有學習過AXI協定下完成，需要先熟讀AXI的spec才能開工，作業1只須熟悉電路的走線即可，而實現AXI需要使用狀態機，讓我更進一步的融會貫通。我們在設計時是先通過functional simulation才去跑VIP，但我認為未來在實現這類協定的電路時，如果有tool就該好好利用，應該並行讓JasperGold成為我們的助力，而不單單是驗證的工具。總的來說，這次作業熟悉了AXI的規格與實現，受益良多。

呂修麟:

之前在使用Xilinx的ZYNQ時就認識到了AXI的協議，但是這次作業不只是使用它，而是要設計出來。我們先花了好幾天討論AXI和CPU於本此作業的架構和設計方法，然後才開始進行coding和debug，debug我們幾乎是一起debug的，有時候兩個人debug會比一個人還要快。我們在使用VIP驗證時，一開始跳出許多cex，但是在共同努力下終於順利完成本次作業。