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: \_\_\_\_\_王華昀\_\_\_\_\_\_

Student ID: \_\_N26134308\_\_\_\_\_\_\_\_

目錄

1. **SUMMARY**
2. **AXI 系統設計**
   1. **CPU Wrapper**
   2. **AXI Bridge**
   3. **SRAM Wrapper**
3. **CPU系統架構**
4. **Jasper VIP驗證**
   1. **Master**
   2. **Bridge**
   3. **Slave**
5. **nWave波型圖**
6. **Superlint**
7. **Sim & Syn Result**
8. **Lesson Learn**
9. **Summary**

本次作業共分為兩個部分，第一部分為設計符合AXI協定規範的通聯系統，必須包括兩個Master、兩個Slave，並使用Jasper驗證AXVIP。第二部分須將HW1設計之5 stage pipeline RISCV CPU接上AXI，通過prog0至prog6的驗證。最後成功合成。

本報告中，首先會先介紹AXI中Master、Bridge、Slave的設計，其中又分為讀及寫兩種行為及其ABVIP驗證結果。接著會根據AXI調整CPU架構，使其能結合AXI正常運作，最後是各個測資的驗證結果。

1. **AXI 系統設計**

* CPU Wrapper

在本次作業中，將HW1的CPU端給予IM及DM的訊號分別當作系統的兩個Master，而在各個Master中又分為READ及WRITE兩種行為。

* + Master READ

Master端READ行為的狀態機如圖1，三個狀態分別為RD\_IDLE、RD\_ADDR、RD\_DATA。RD\_IDLE狀態為起始狀態，CPU端會持續更新需要傳送的資訊，並透過暫存器存取。

等到ARVALID跳起後表示讀取地址確定，這地址會維持暫存器中的數值，同時並傳送一個stall訊號回CPU，將各級間的暫存器暫停，直到讀取完成才可以繼續。狀態則跳到RD\_ADDR狀態，RD\_ADDR狀態主要是等待AR頻道握手完成，將要讀取的地址傳送過去Slave端。

等到AR頻道握手後，狀態會跳到RD\_DATA，RD\_DATA主要是將從SRAM中讀取出的數值傳送回Master端，此階段的片段主要是透過RLAST，當數值全部船後完畢後RLAST跳起，將狀態返回RD\_IDLE，完成一次讀取，同時將stall訊號跳下，使CPU繼續運作。每一次讀取數值需要4個clk cycle。

一張含有 文字, 圖表, 螢幕擷取畫面, 行 的圖片

自動產生的描述

圖1 Master READ

* + Master WRITE

Master端WRITE行為的狀態機如圖2，其狀態與READ相似，只是多一個WD\_RESP狀態傳輸response資料。各個狀態與READ行為時相同，在WD\_IDLE時會將CPU端會持續更新需要傳送的資訊，並透過暫存器存取。在WRITE行為開始時，同樣也傳送一個stall訊號回CPU，到WD\_ADDR狀態時會將寫入地址傳送去Slave端並等待存取數據，等待AW頻道握手後到WD\_DATA進行寫入數據的傳輸，利用WLAST判斷寫入完成跳到WD\_RESP後，利用B頻道握手判斷response訊號傳輸完成，並回保WD\_IDLE狀態。每一次寫入需要5個clk cycle。

一張含有 文字, 圖表, 行, 字型 的圖片

自動產生的描述

圖2 Master WRITE

* AXI Bridge
  + AXI Bridge READ

AXI READ狀態分為兩個狀態，分別為IDLE及DATA。在READ行為中，此module負責AR頻道以及R頻道的聯通。有兩個Master端及兩個Slave端。當有讀取需求時，等待Master端及Slave端AR頻道握手成功，進入到DATA狀態。驗證VIP時發現到由於會有M0及S0、M0及S1、M1及S0、M1及S1四種情況，並且透過一個暫存器去紀錄存取的Master端及Slave端，當同時兩個Master要讀取數值時，使用輪流的方式進行控制。與CPU wrapper相同，當R頻道握手並且RLAST時，表示讀取完成，將狀態跳回IDLE。

一張含有 文字, 圖表, 圓形, 螢幕擷取畫面 的圖片

自動產生的描述

圖3 AXI Bridge READ

* + AXI Bridge WRITE

與AXI Bridge READ相似，狀態同樣分為IDLE及DATA。在WRITE行為中，此module負責WR、W、B頻道的聯通，理想上與AXI Bridge READ一樣需要考慮到四種不同Master及Slave的組合，而由於在本次的作業中並不會有M0寫入的情況發生，因此需要考慮的組合不會有M0的訊號。

一張含有 文字, 圖表, 螢幕擷取畫面, 圓形 的圖片

自動產生的描述

圖4 AXI Bridge WRITE

* SRAM Wrapper

在本次作業中，將HW1的IM及DM分別當作系統的兩個Slave，而在各個Slave中又分為READ及WRITE兩種行為。

* + Slave READ

Slave端READ行為的狀態機如圖5，三個狀態分別為RD\_IDLE、RD\_ADDR、RD\_DATA。RD\_IDLE狀態為起始狀態，與Master端不同的是，由於Slave端是被動接收訊號，並且在本次作業中還會有一個clk的延遲才能將資料讀出來，因此在AR通道握手後，進入RD\_ADDR狀態，此時這個狀態將要讀取的地址告知SRAM並等待一個clk後直接進入RD\_DATA，同樣使用RLAST判斷數值傳輸完畢，回到RD\_IDLE。

一張含有 文字, 圖表, 螢幕擷取畫面, 行 的圖片

自動產生的描述

圖5 Slave READ

* + Slave WRITE

Slave端WRITE行為的狀態機如圖6。與Master端較為不同，由於Slave是被動接收訊號，因此當有寫入需求後，可以將些入地址及寫入數值同時傳輸過來，也就是在AW頻道握手後，進入WD\_TRANS狀態，再等到W頻道握手並且WLAST跳起，說明寫入地址及寫入數值都已完成，再進入到WD\_RESP，從Slave端傳輸response訊後回去Master端。再B頻道握手後說明寫入行為完成，再跳回WD\_IDLE。

一張含有 文字, 圖表, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

圖6 Slave READ

1. **CPU系統架構**

**一張含有 文字, 圖表, 方案, 工程製圖 的圖片

自動產生的描述**

圖7 系統架構圖

本次作業中使用HW1做的CPU作為系統Master端，由於需加上AXI通聯，因此有針對以下幾點做更動。

* SRAM讀取延遲問題:

在HW1時，由於是直接將CPU接上SRAM進行通聯，因此SRAM的讀取延遲問題要在CPU內部進行處理。而在HW2時改為使用AXI進行通聯，在上述描述中，有提到在SRAM READ已經有針對讀取的數值延遲問題進行處理，因此從AXI回傳的RDATA可以直接使用。

* AXI 讀取及寫入時stall CPU:

在本次作業中使用AXI進行通聯，由於每次讀取及寫入都需要等待讀取及寫入動作完成，而在我設計的AXI系統讀取需要4個clk cycle寫入需要5個clk cycle，因此在等待AXI動作時需要將CPU各級的數值暫停運作。除此之外，由於AXI要進行甚麼行動是透過CU中MEM\_read及MEM\_write進行決定，因此這部分的訊號不能只是單純暫停，當AXI運作完成後要將訊號拉回CPU內進行判斷，使CPU不會一直卡住。

* 遇到的困難:

在測資中會有Master DM端讀取Slave IM的情況，會造成Hazard的產生，因此導致CPU系統中同時有兩個stall訊號出現，AXI的stall及原本CPU端的stall訊號。這部分的處理方法是將CPU中的所有訊號都與AXI\_stall對齊。如同將HW1中的每個clk cycle都直接拉成本次作業的4個clk cycle，這樣做雖然會導致CPU運行的cycle變多，但是可以確保CPU內部沒有數值會由於沒有對齊而讀取到錯誤的數值。本次作業幾乎都是在IF及MEM進行更動，而由於在HW1中有特別將Hazard Unit拉出五階架構外，因此在使用AXI\_stall將各階間的暫存器暫停就成功解決很多問題。其餘部分如PC或其他暫存器也必須同時用AXI\_stall暫停。

1. **Jasper VIP驗證**

本次作業中需要針對Master、Bridge、Slave端進行JasperGold ABVIP的驗證。

* Master

在Master端有13個cover未滿足，是因為當AR頻道、AW頻道、W頻道握手後，下個cycle READY訊號就會跳高，因此在握手後無法偵測到訊號的stable，其中13個都是這部分的問題，其餘部分cex為0。

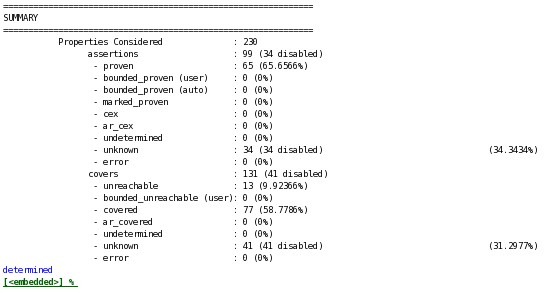


圖8 Master VIP

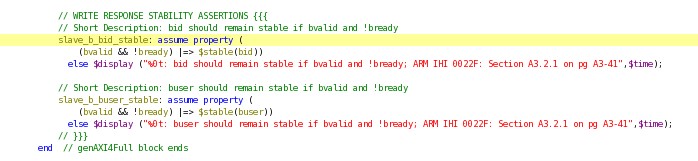
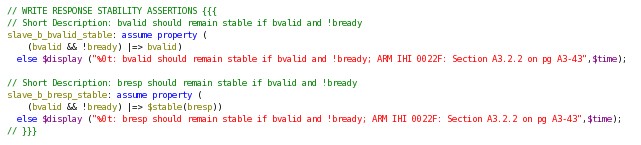
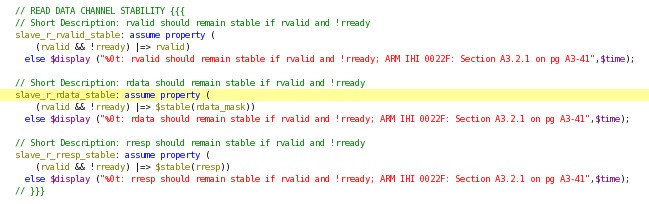
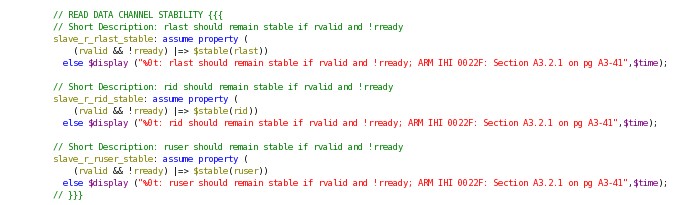
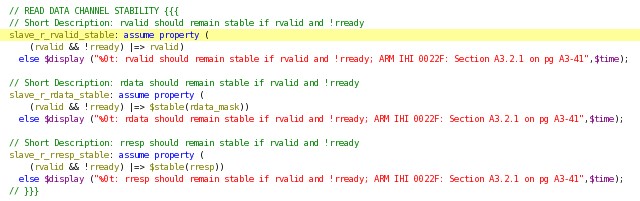


圖9 Master VIP 13個unreachable

* Bridge

在Bridge端的VIP驗證，如同上述AXI設計提到，由於在本次CPU的架構中其實有很多組合是不會使用到的，因此最開始在設計時會有一部分沒有cover到，因此需要針對那部分增加更多握手情況，最後cex及uncover皆為0。

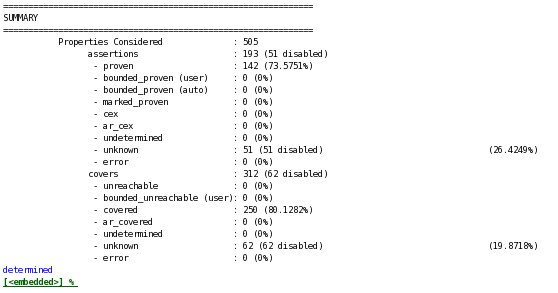


圖10 Bridge VIP

* Slave

與Master端一樣，是因為當AR頻道、AW頻道、W頻道握手後，下個cycle READY訊號就會跳高，因此在握手後無法偵測到訊號的stable，這是由於READY訊號在握手前一直為1，因此沒有READY訊號為0的情形。解決方法為在SRAM wrapper中加入contr訊號判斷現在是READ還是WRITE，當READ時，WREADY會設為0；當WRITE時，RREADY設為0，使得各個情況都有cover到。

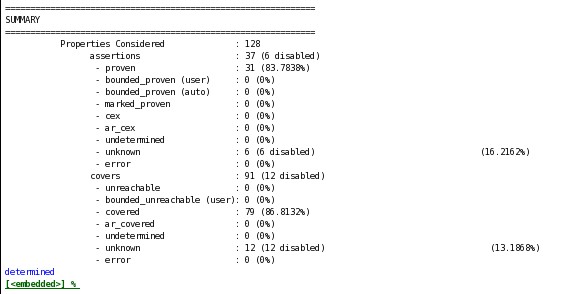


圖11 Slave VIP

1. **波型圖**

* **AXI stall 及clk的關係**

從下方波型圖可以看出，在進行instruction讀取時將HW1的1個clk cycle擴增成4個clk cycle，將所有數值對齊使CPU同步不會有問題產生。

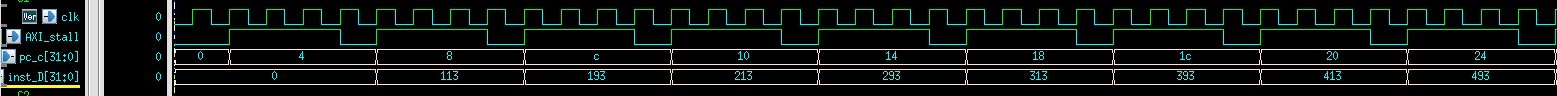


圖12 AXI stall 及clk的關係

* **同時讀取IM及DM**

從下方波型圖可以看出，在r\_en\_IM及r\_en\_DM同時跳為1時，需要連續做兩次讀取的動作，因此clk cycle暫停7個cycle在第8個cycle讀出。其中bridge的Master選擇從M0跳至M1再跳回M0，完成輪流使用Bridge。

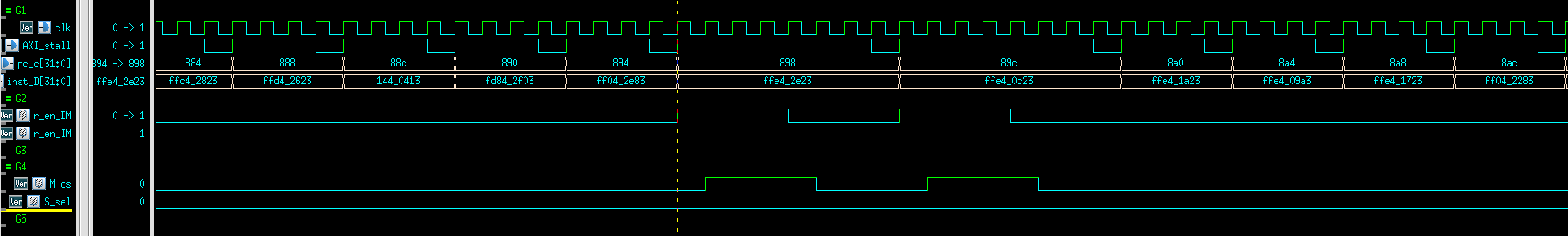


圖13 同時讀取IM及DM

* **寫入DM後直接讀取DM**

從下方波型圖可以看出，在w\_en跳為1後寫入數值，緊接著讀取同一個地址的數值，並根據BWEB判斷讀取byte，有成功讀取數值。

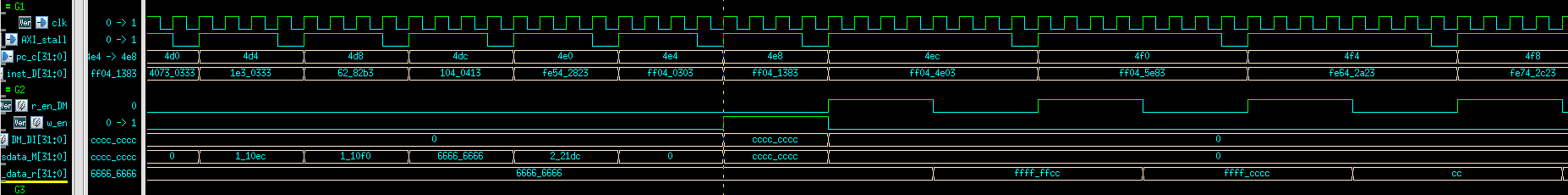


圖14 寫入DM後直接讀取DM

* **Hazard發生時stall訊號處理**

原先CPU中Hazard發生時的stall訊號及AXI stall訊號打結，只需將其進行對其即可解決CPU停擺問題。

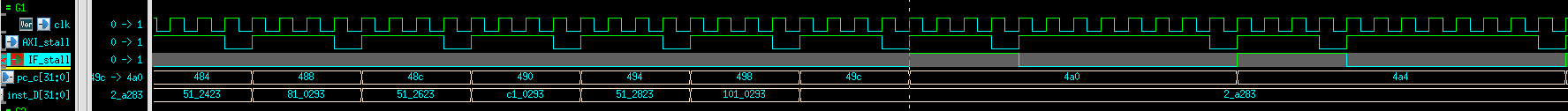


圖15 Hazard發生時stall訊號處理

* **AXI訊號判斷w\_en**

前面章節有提到需要將AXI訊號拉入CPU判斷DM讀取完畢，避免CPU停擺，是將RVALID\_M拉至MEM stage，當RVALID跳起時，w\_en結束。

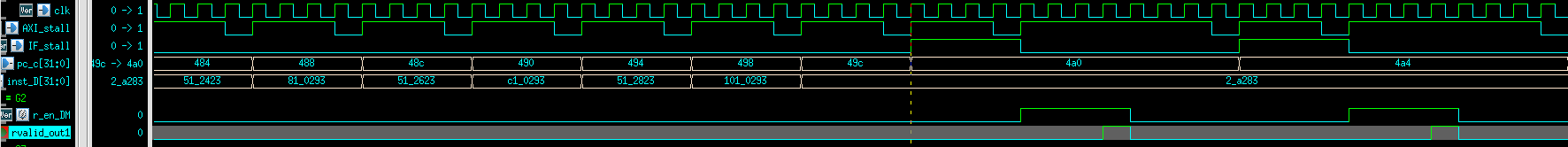


圖16 AXI訊號判斷w\_en

1. **Superlint**

問題與HW1時大致相同，因此很快就修正完畢，除了其中在SRAM的DO，跳出Logic register variable “DO” is unassigned，但由於我在SRAM wrapper中並沒有將DO使用DFF，而是直接用combination將DO直接拉出來，因此推測為SRAM本身的問題。

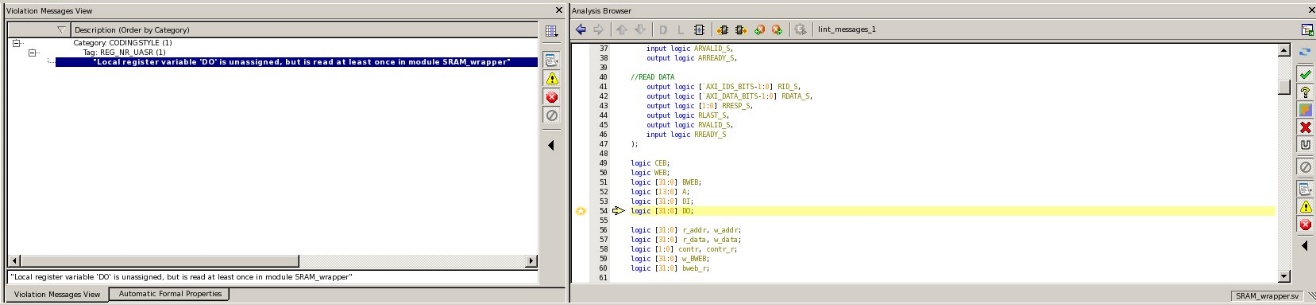
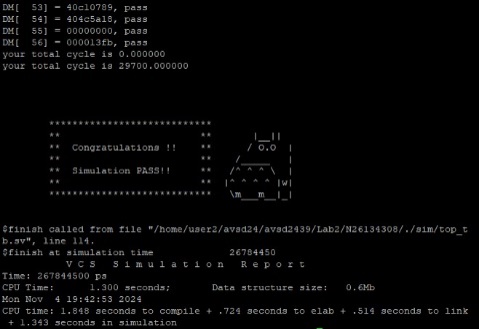
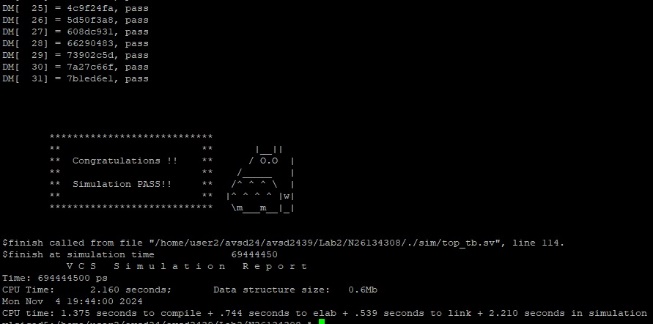
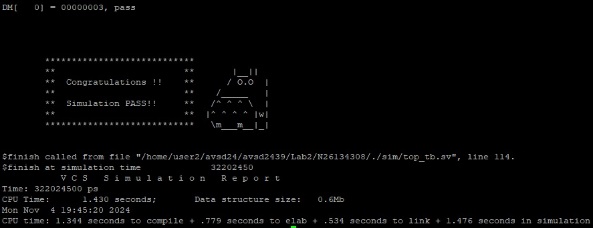


圖17 Superlint

1. **Sim & Syn Result**

**1. 2 3.4 **

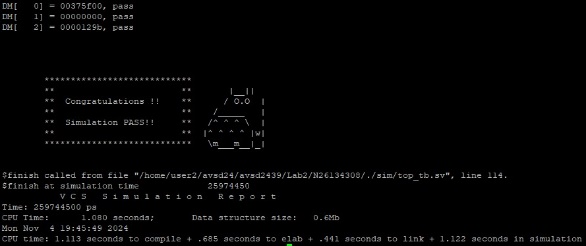
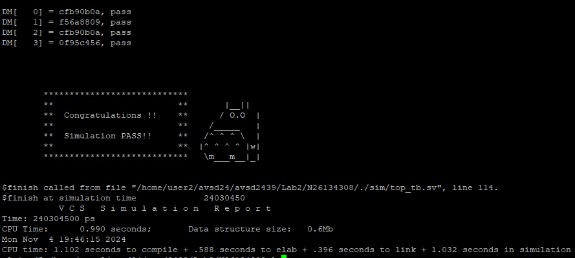
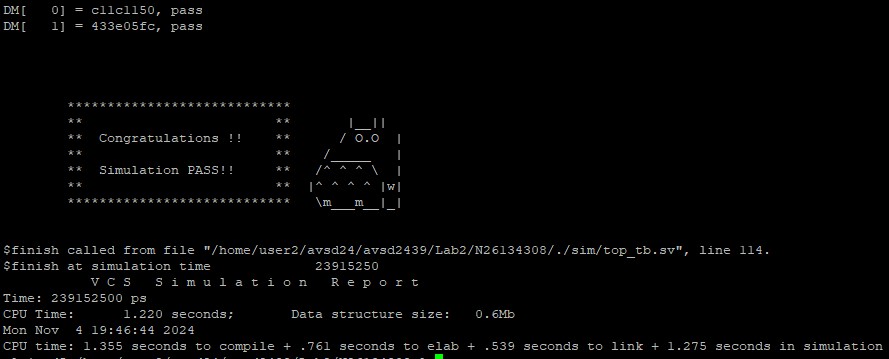
**5. 6.7.**

圖18 sim result

**Area report**

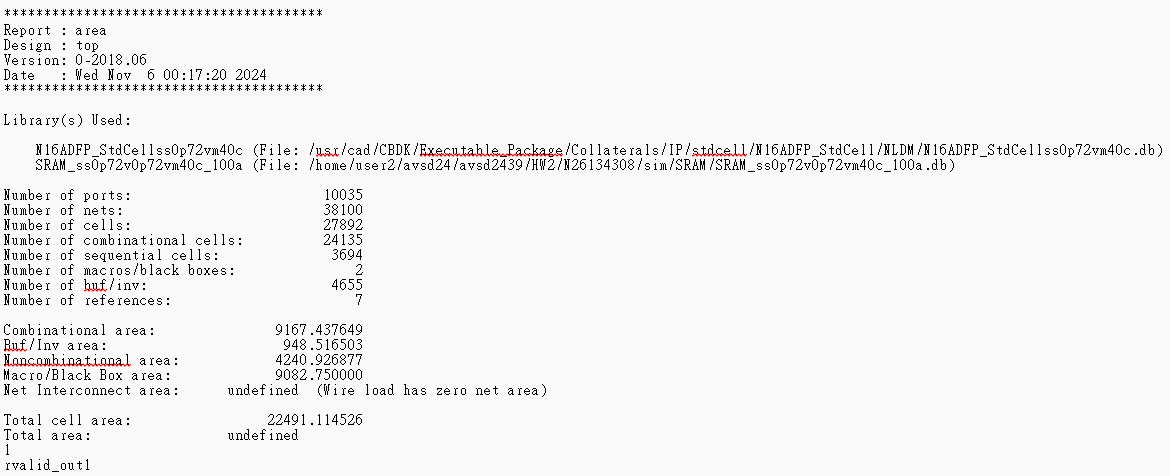
****

圖19 syn result

1. **Lesson Learn**

在本次作業前我是完全沒有接觸過AXI協定，因此前面兩個禮拜都只是在瞭解各個訊號的用途，再加上需要將AXI的設計結合到CPU上面，由於一開始在跑過VIP銜接上CPU後，RTL一直都沒辦法成功通過，導致我又從波型圖最開始看到錯進去修改CPU架構，最後發現其實只要將所有register暫停住，就等於把HW1中1個clk cycle變成4個clk cycle就可以解決問題，不需要針對CPU架構大改，浪費很多時間在debug。下次要把東西再加在這個CPU架構時，我應該先思考清楚核心的問題是在哪裡，而不是將原本對的東西又去做更改。