VLSI System Design

(Graduate Level)

Fall 2021

HW2 report

Must do self-checking before submission:

Compress all files described in the problem into one tar

All System Verilog 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: 鄭丞祥、陳冠佑

Student ID: N26094891、N26090130

# Problem Description

Please complete a simplified Advanced eXtensible Interface (AXI) with 2 masters

and 2 slaves. You DO NOT need to implement cache functions, atomic accesses, protection units. You have to verify the AXI architecture by using Jasper Gold AXI

ABVIP.

# Check List

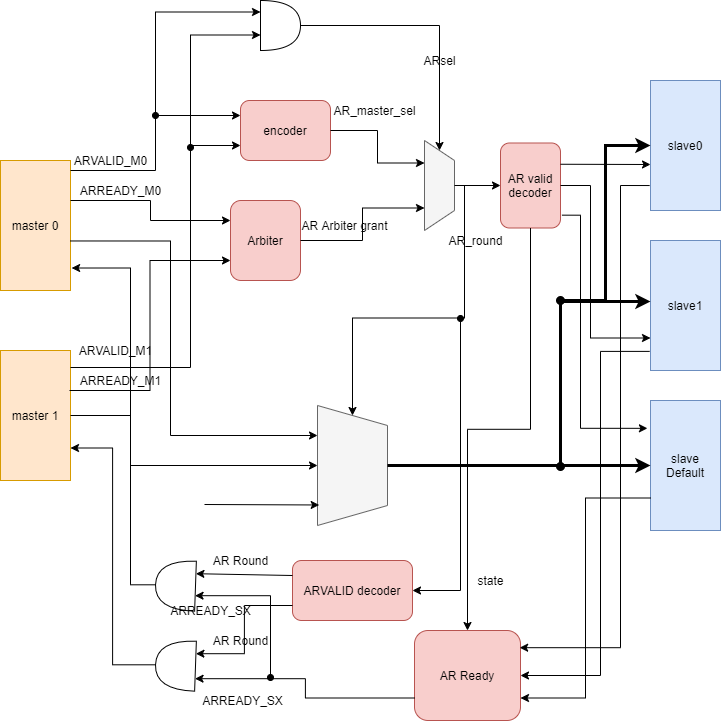
|  |  |  |  |
| --- | --- | --- | --- |
| RTL code | | Synthesis | |
| Prog0 | Pass | Syn0 | Pass |
| Prog1 | Pass | Syn1 | Pass |
| Prog2 | Pass | Syn2 | Pass |
| Prog3 | Pass | Syn3 | pass |
| Superlint | 99.97 % | | |
| AXI\_VIP | PASS | | |
| CPU\_Wrapper\_VIP | PASS | | |
| Slave\_Wrapper\_VIP | PASS | | |

# Contribution

* 鄭丞祥(50%)
* 陳冠佑(50%)

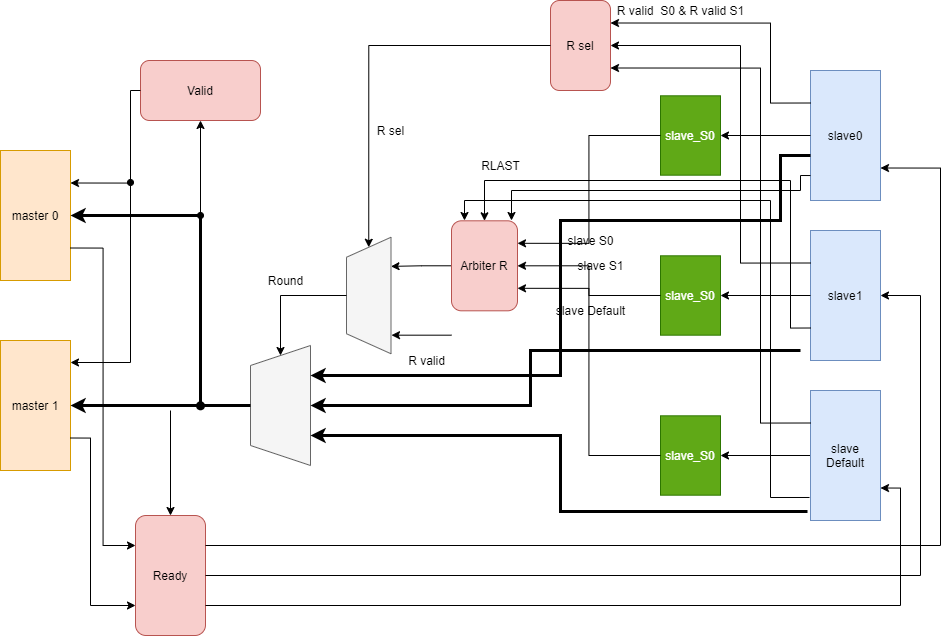
# Problem 1

* Block Diagram
* AR channel



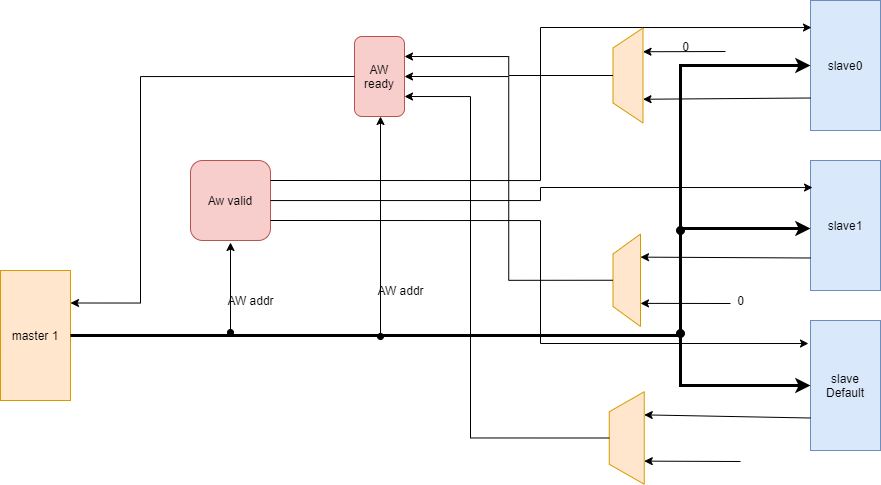
AR channel主要行為如上圖所示，今天假設只有其中一個master要做事情的話，valid訊號透直接透過encoder的模組傳出去，但是假設今天有同時兩個master想要做事情的話，就會需要透過Arbiter的方式來選擇今天到底誰能先做，而決定的方式是透過Round Robin，也就是做的優先權會輪替。透過Arbiter，電路可以得知當下獲得grant的master是誰，而獲得grant訊號的master會將所有的訊號傳遞給所有的slave，但是能真正拿到這筆資料的slave只有獲得ARvalid的人，也就是從把ARaddr透過decoder解碼後把ARvalid傳遞給相對應的slave。至於ARready的部分也一樣會透過ARaddr來去判斷給選擇哪個slave給的訊號線，送到master的訊號則是透過grant來決定。

* R channel



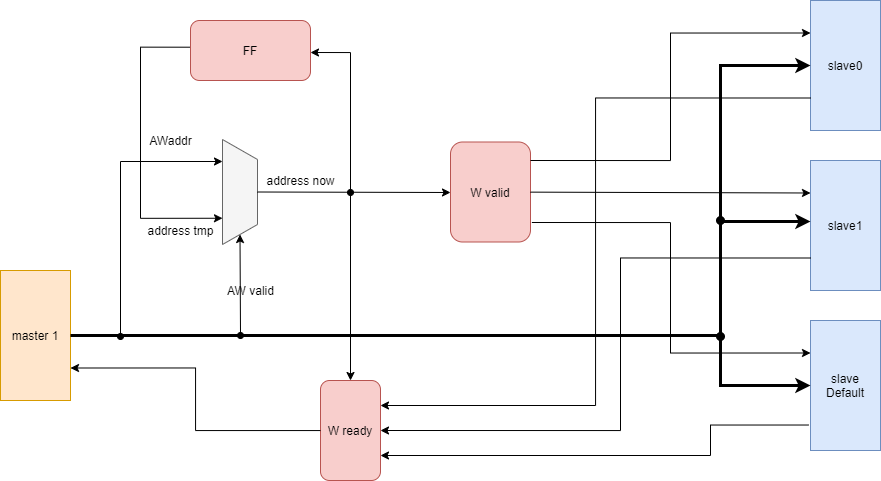
R channel 架構圖如上所示，當今天只有一個slave想做事情的時候就會直接讓他去做，但是若有多個slave想做的話就會透過arbiter來解決，採用方法一樣是round robin，被grant到的slave會把所有的訊號都送到master端，但真正可以拿到訊號的人只有拿到Rvalid的那個master。Rvalid 訊號是透過RID來做解碼才決定要給哪個master。Rready也相同，也是透過RID來解碼才決定今天要給哪個master此訊號線。

* AW channel



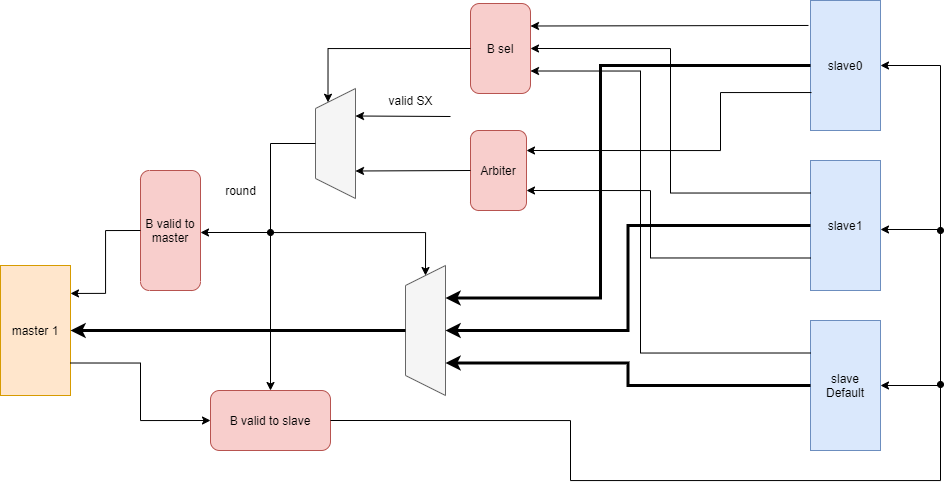
AW 架構圖如上圖所示，在AW中與前兩者不同的地方在於因為會做寫入的master只有一個，所以說不用像前兩者一個需要利用arbiter來決定此刻的優先權為誰，可以直接將所有的除了AWvalid的訊號傳遞給slave，而誰能真正拿到這筆資料則是透過AWvalid來決定，決定該給哪個slave valid訊號是透過AWaddr來決定的，同樣的AWready也會需要透過AWaddr來，再加上valid來決定哪個slave回傳的ready訊號有效。

* W channel



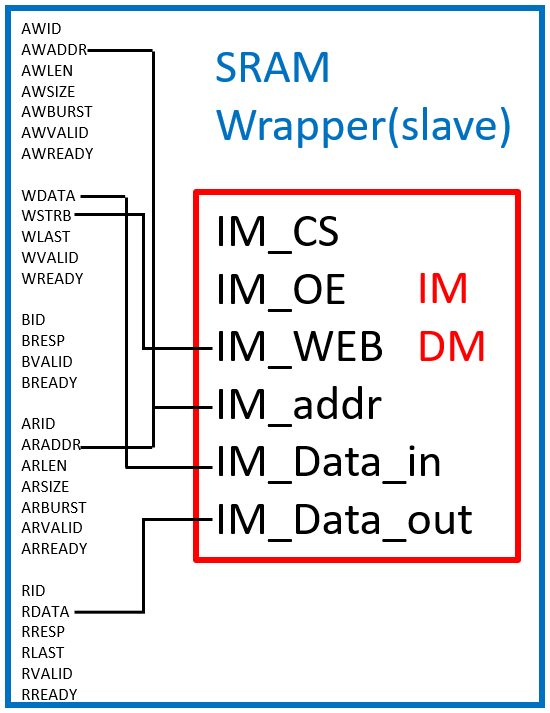
W channel 架構圖如圖所示，因為一樣只有一個master會進行寫的動作，所以master會把所有除了valid的訊號都送到slave端，而誰能採用到這些訊號則是透過address來決定。比較特別的是需要注意到，因為需要確認當次的寫值結束與否，如果尚未結束的話會需要將數值buffer住，以讓valid可以被相對應的slave接收到。而W ready可以藉由地址來決定哪個slave訊號可以被送回master端來完成此次的handshake。

* B channel



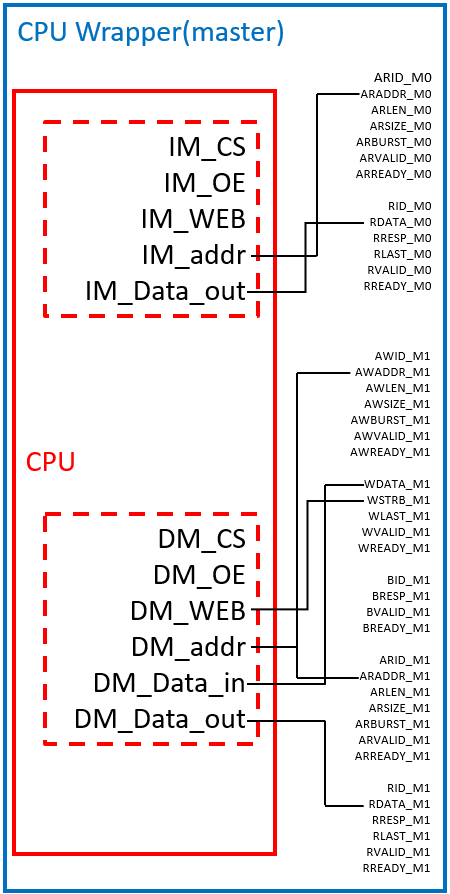
B channel 架構圖如上所示，當只有一個slave想做事情的時候會直接讓他做，但當今天有兩個以上的slave想做事情的時候就會需要透過arbiter來決定優先權落在誰身上，決定方式跟之前一樣。而被選到的slave會把所有資料都送到master身上，B ready的部分則一樣是透過arbiter的方式來決定要送到哪個slave身上。

* SRAM wrapper

slave 端較單純，即分別包兩顆SRAM，一顆做為IM，另一顆為DM，再讀資料時AR channel master端會發送valid訊號給SRAM wrapper，等待slave端發送Ready訊號，進入ReadData channel，等待最後一筆資料時拉起Rlast訊號，完成讀資料之傳輸。再寫資料時，同樣的在AW Channel master 會發出valid訊號，slave端則發出Ready訊號，進行handshake進入WriteData channel，在資料寫入完成後，slave端會發出response給master端告訴master完成資料寫入的動作。

另外由於SRAM為single port因此沒辦法同時進行讀寫，master若要同時讀寫同一SRAM (DM)，需等待一個做完另一個才能執行。另外burst length 2的問題我們是使用讓傳輸完一筆資料後valid訊號會拉下來1個cycle，由於SRAM讀data也需要一個cycle的時間，故利用此cycle準備下一筆data。

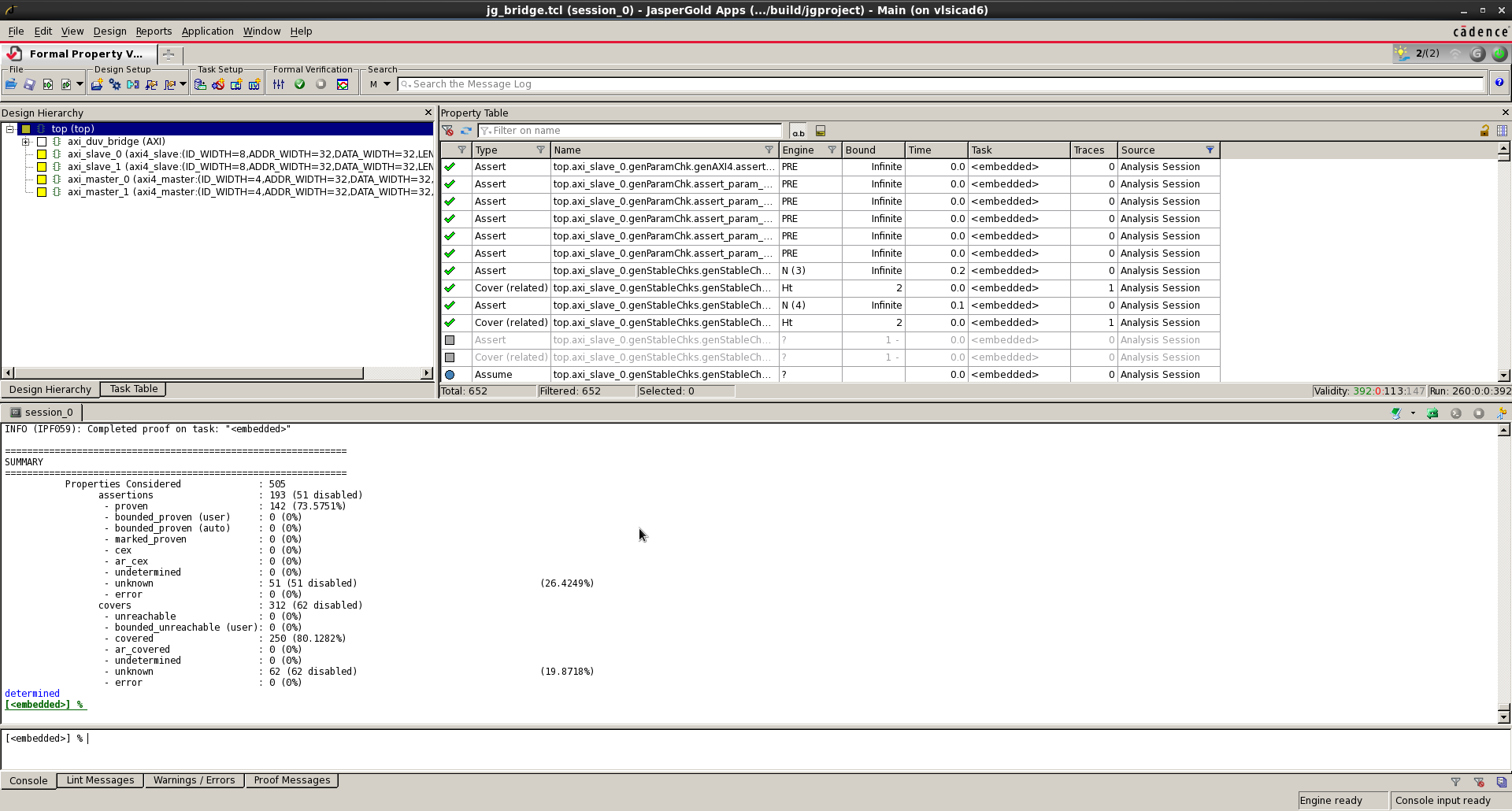
* CPU wrapper

CPU wrapper我們這邊的設計為由於CPU必須向IM讀取指令，並對DM進行LW或SW等指令讀寫記憶體，因此相當於2個master，這邊即可看出我們的CPU wrapper裡面有兩個master但由於master 0只會對IM進行讀指令並不會寫入IM，因此腳位叫DM少許多，另外master1則為對DM進行讀寫。

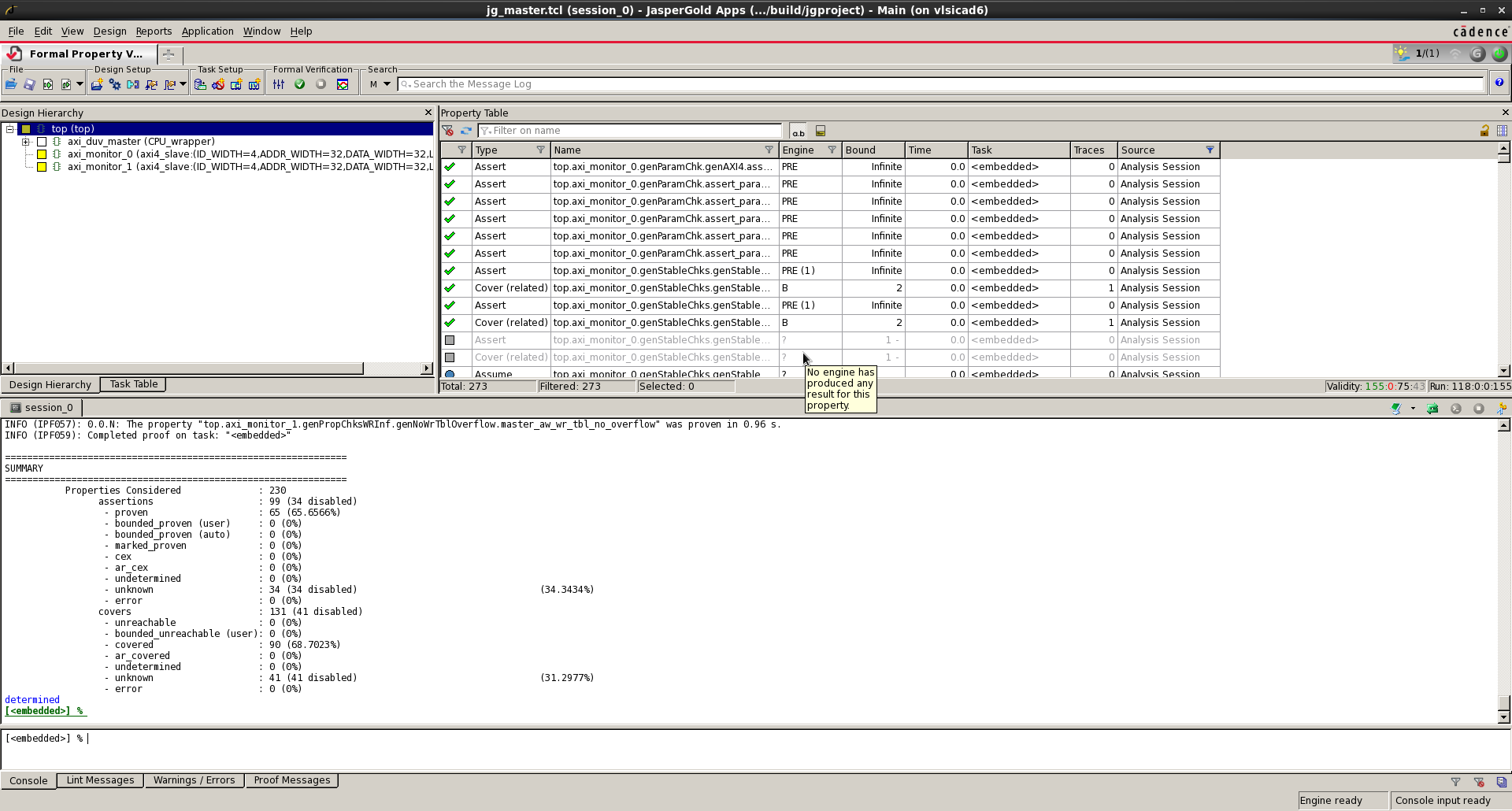
這邊我們的設計為利用兩個FSM去做控制，分別控制master 0 (IM)及master 1 (DM) 的state，在利用兩者之state來控制CPU READY來告訴CPU bus上的資料傳輸是否已經完成。

控制FSM中讀資料的state為透過CPU的CS(chip select)與OE(output enable)同時拉高，代表master端要進行讀的動作，因此ARVALID會拉高等待ARREADY方進行讀DATA的動作，同樣的透過CPU的CS為1與WEB不為4’b1111時，則代表master要進行寫的動作，因此AWVALID會拉高，等待AWREADY進行寫DATA的動作。

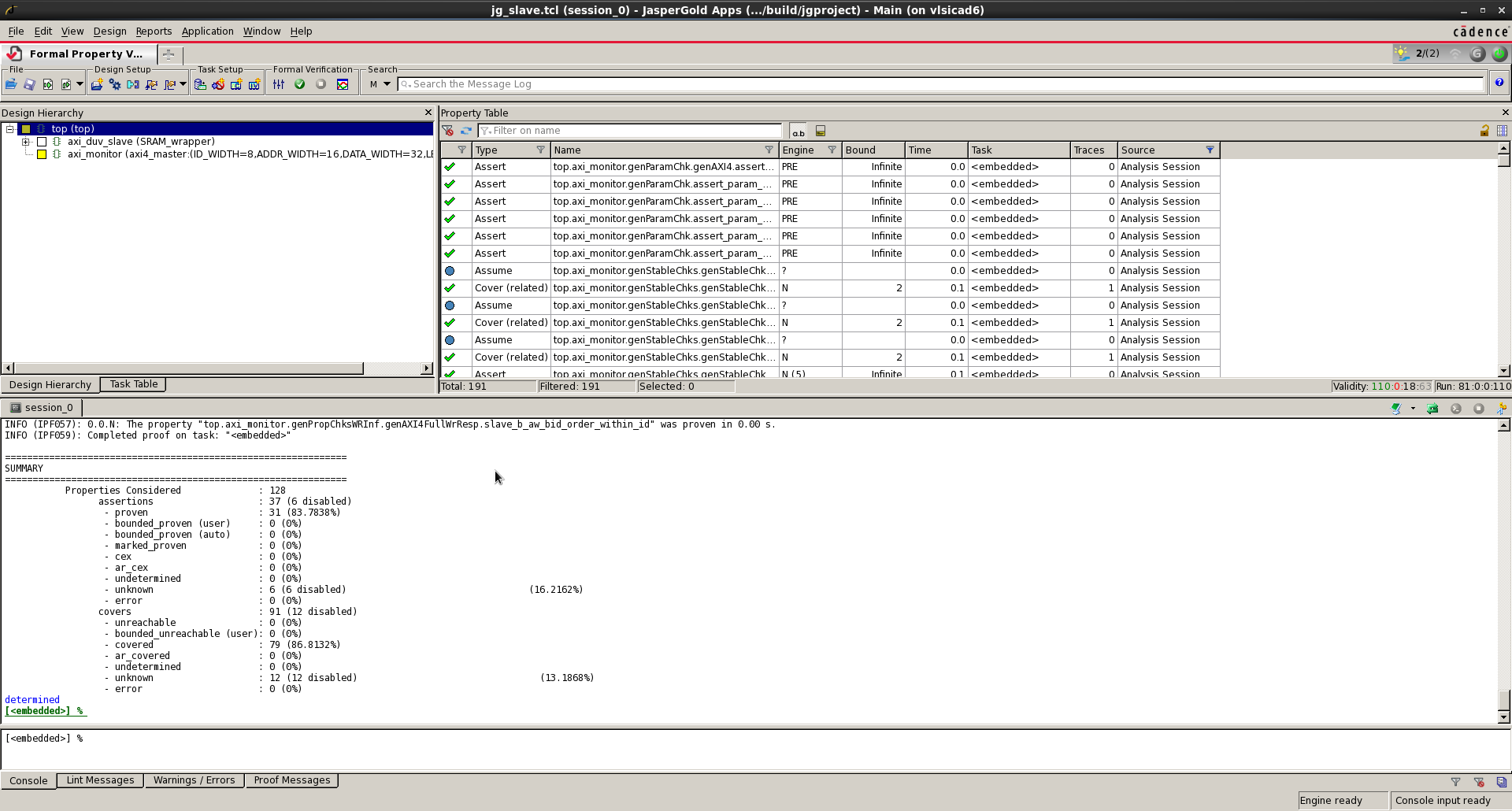
* VIP B verification: AXI的assertion全數通過



* VIP M verification: Master的assertion全數通過

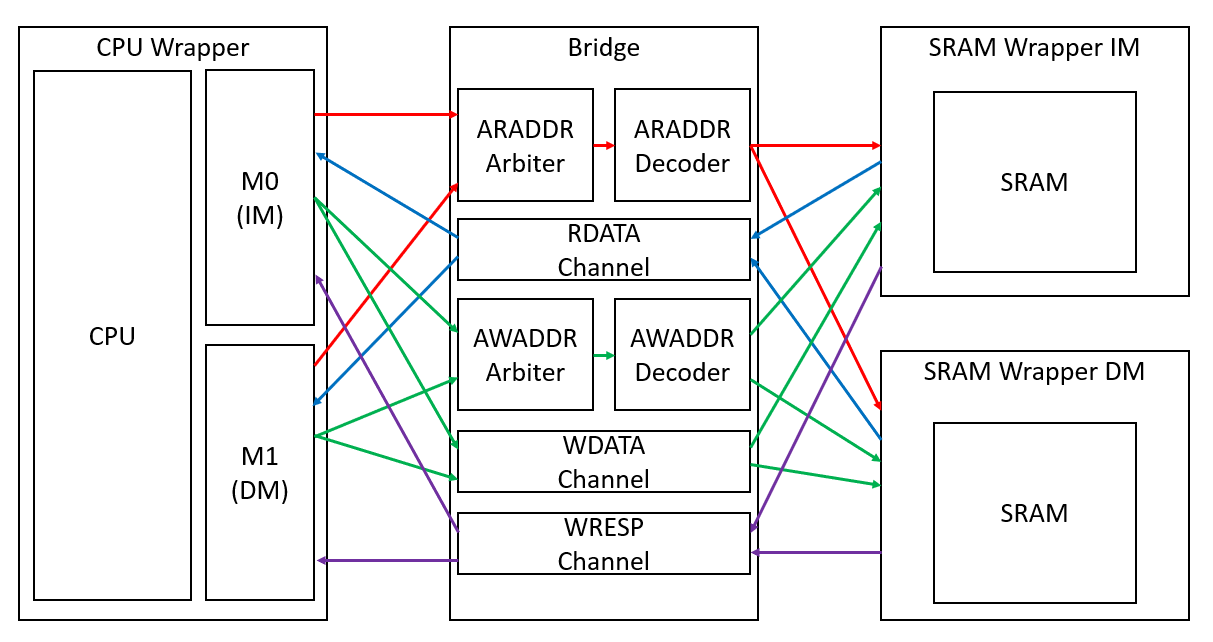


* VIP S verification: Slave的assertion全數通過



# Problem 2

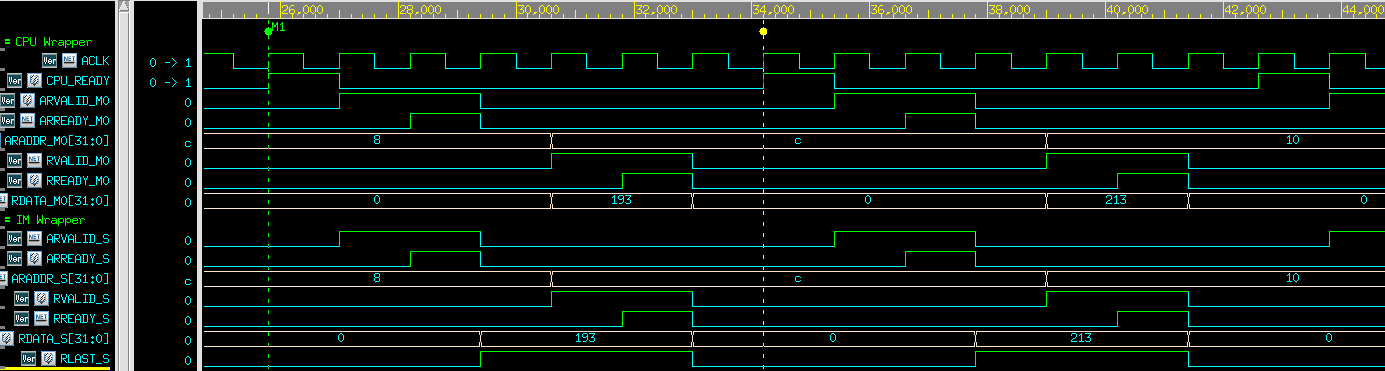
* Block Diagram



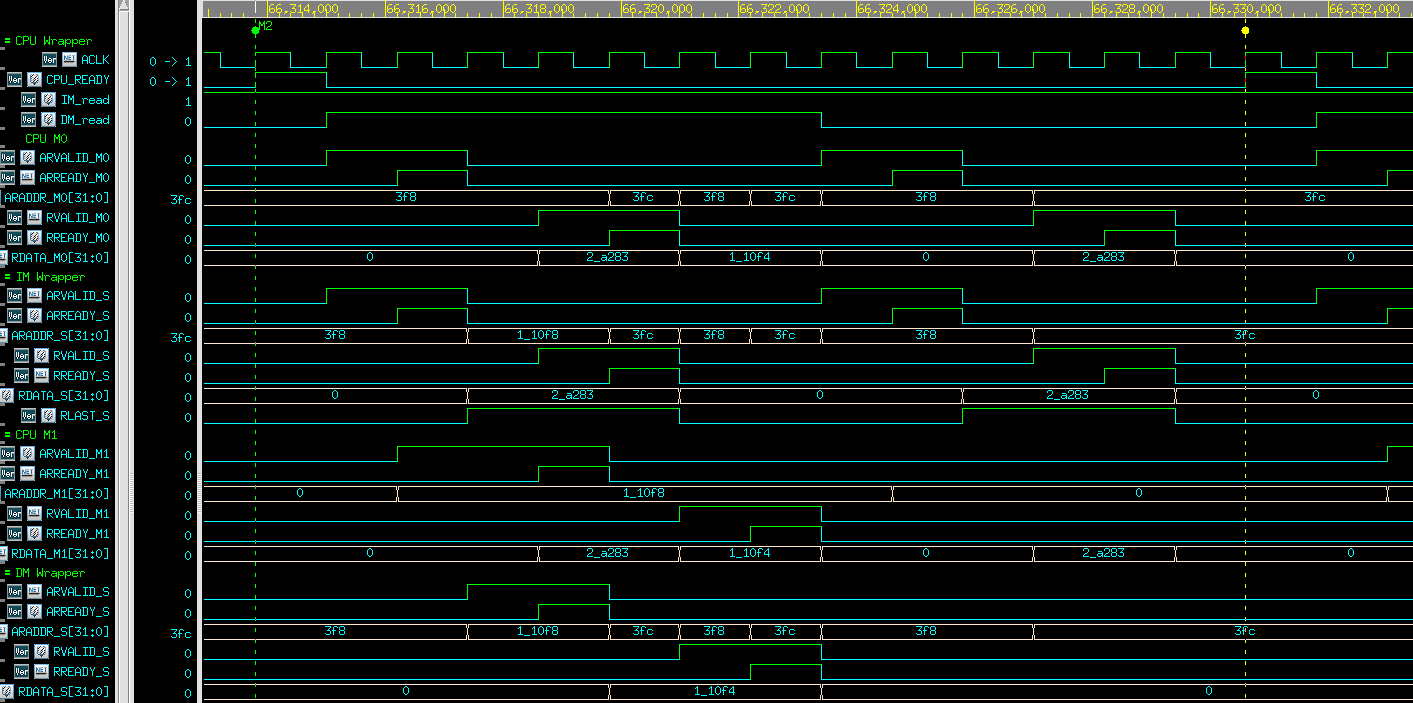
ARADDR  
Channel

AWADDR  
Channel

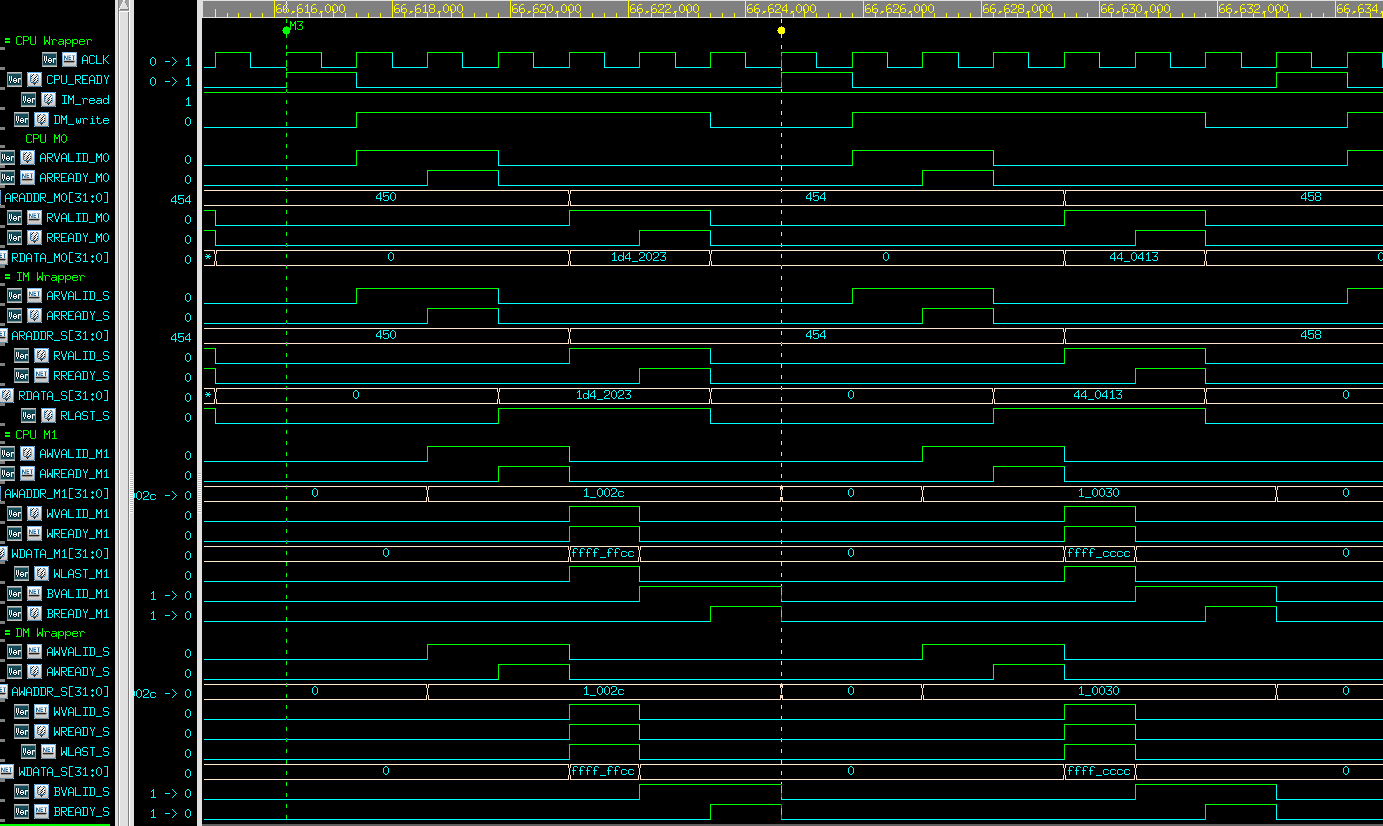
由於CPU現在必須透過AXI Bus來跟SRAM進行資料讀寫的溝通，而不像作業一可以在一個cycle拿到或寫入資料，因此我在CPU Wrapper的設計中有設計若兩個FSM都為IDLE state，會發出一個訊號CPU READY給CPU告訴CPU可以繼續執行，而當CPU READY等於0時，則代表data還在bus上進行傳輸，因此所有CPU的pipeline或有吃clk的訊號我們都將他stall住，暫停動作，等到bus所有傳輸完成，才會繼續動作。  
若是IM以及DM都要同時讀寫的話(IM讀、DM讀或寫)，AXI的仲裁器會決定先後順序，因為一次只能有一筆資料在bus上傳輸，因此我們用狀態機來控制，但也會導致若兩個memory同時要動作，需要stall較長的時間。

* Waveform
  + IM Read (instruction Fetch):

由上述波型圖可以看出當第一次AR channel handshake完成後，才會進入到R channel，在透過AR channel之addr(8)讀取指令後做handshake將資料(193)傳回CPU，CPU READY才會再次拉起。

* + IM Read and DM Read:

此部分的波型圖可以看到此時的動作為IM要進行讀指令，DM也要進行load word，可以看到由於bus上一次只能有一筆資料傳輸，紅色框框為IM的AR channel handshake及 R channel handshake，讀取addr (3f8)的data(2\_a283)，黃色框框為DM的AR channel handshake及 R channel handshake，讀取addr (1\_10f8)的data(1\_10f4)，後面多的cycle為狀態機設計導致不影響行為。

* + IM Read and DM Write:

此部分的波型圖可以看到此時的動作為IM要進行讀指令，DM要進行store word，可以看到由於bus上一次只能有一筆資料傳輸，紅色框框為IM的AR channel handshake及 R channel handshake，讀取addr (450)的data(1d4\_2023)，黃色框框為DM的AW channel handshake、W channel handshake及Response channel handshake，為在addr (1\_002c)寫入data(ffff\_ffcc)。

* Prog0測試結果:



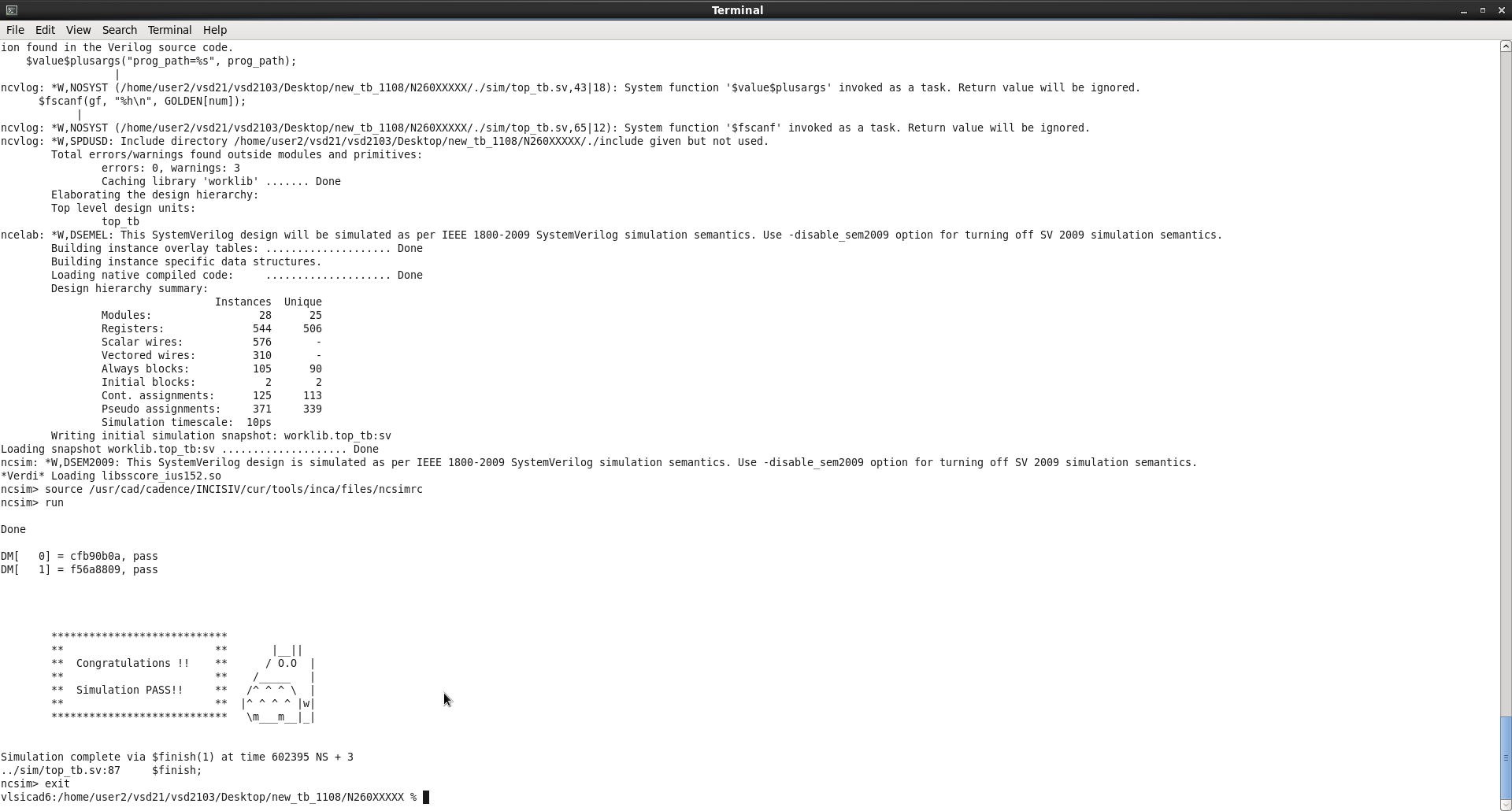
* Prog1: Sort Algorithm

此測試程式可以利用單純的bubble sort完成，基本原理就是不斷地和自己隔壁的數字做比較，當我數字比隔壁的大時就做交換，比較小時就不做任何行為，不斷重複此行為，直到所以的數字都彼此比較過後即完成此演算法。



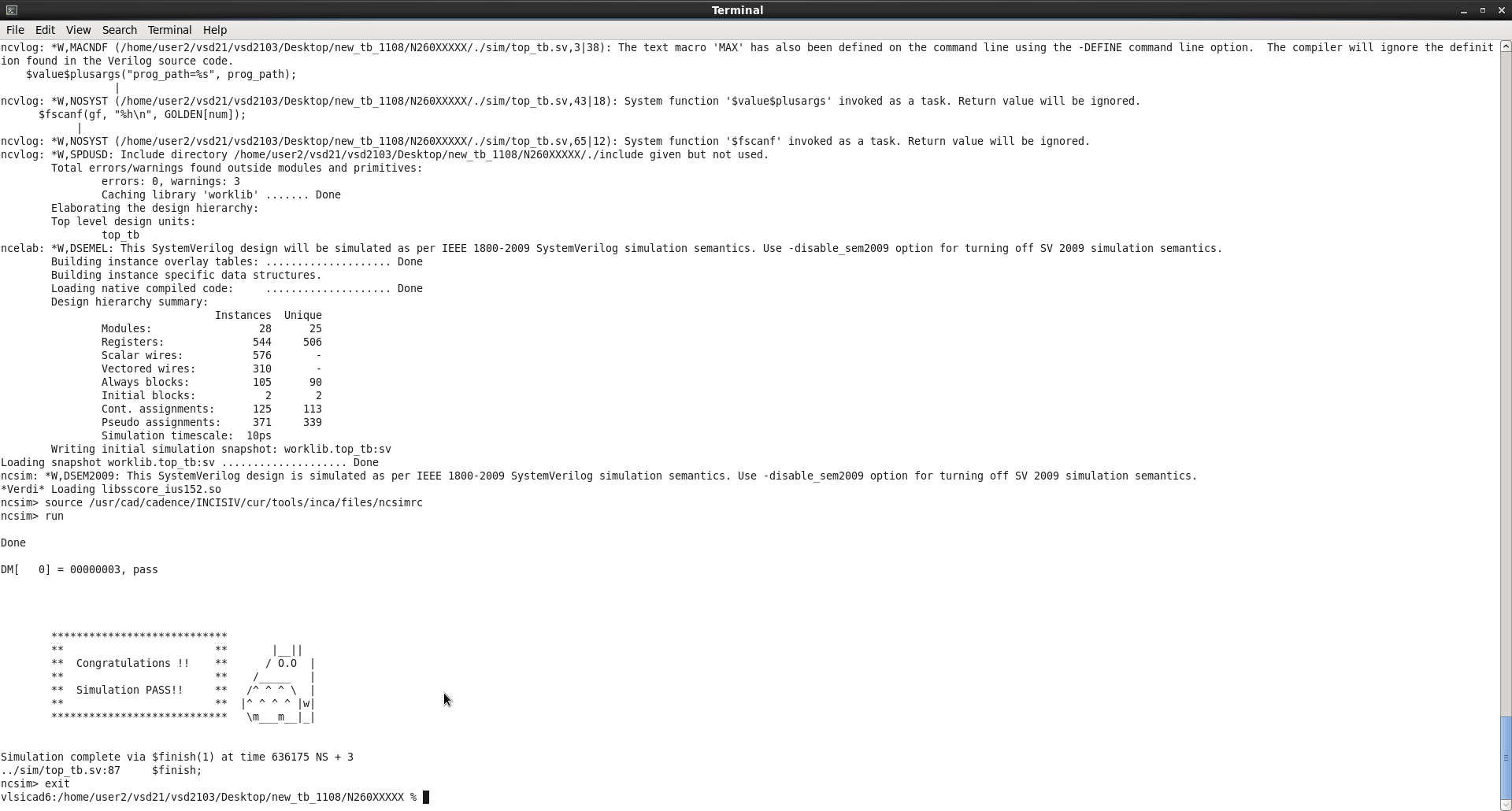
* Prog2: Multiplication

因為此測試程式要求的行為模式為將兩個32bit的數字做相乘，而32bit x 32bit所得到的答案會是64bits 長。 因此相乘後需要將答案分成兩部分存入32bit 的Memory，分成兩部分32bit長的運算可以透過shift(>>)來達成。

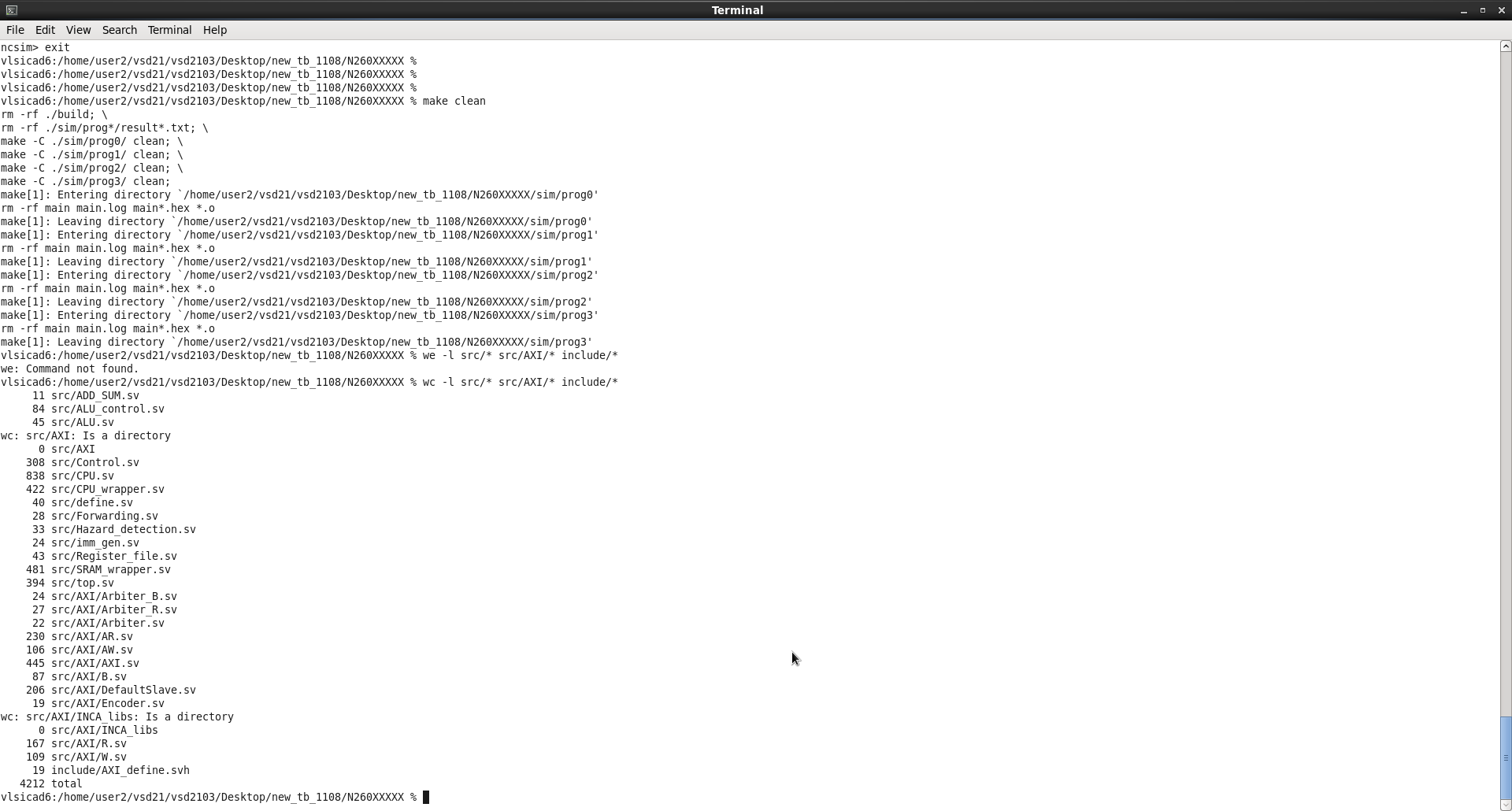


* Prog3: Greatest common divisor (GCD)

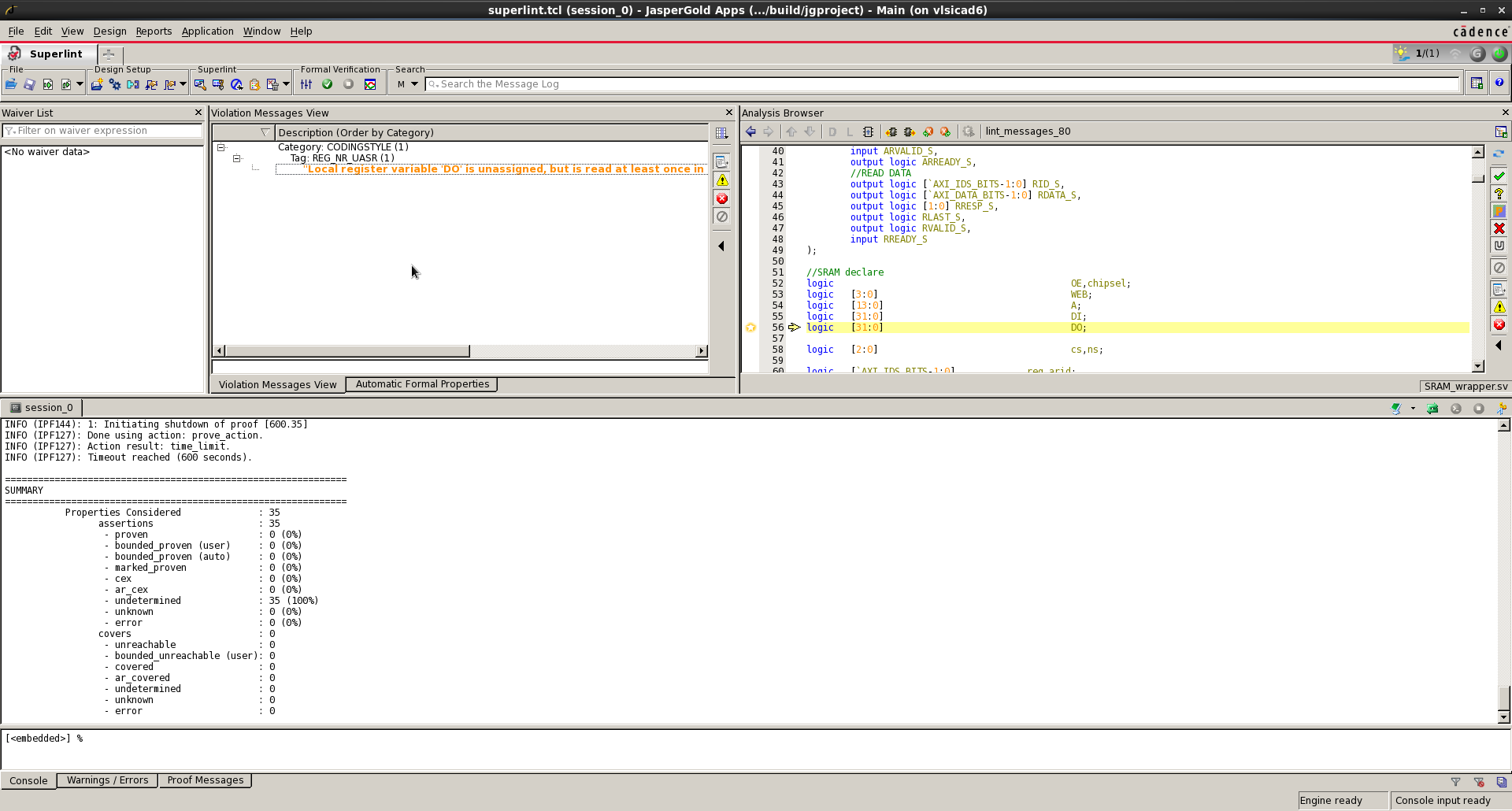
此範例程式的要求可以透過利用輾轉相除法來完成，輾轉相除法的概念就是透過將兩數相除後所得到的餘數做為下一次除法的輸入，不斷地執行，直到餘數為0的時候，此時的除數即為最大公因數。



# Number of lines of RTL code: 4212



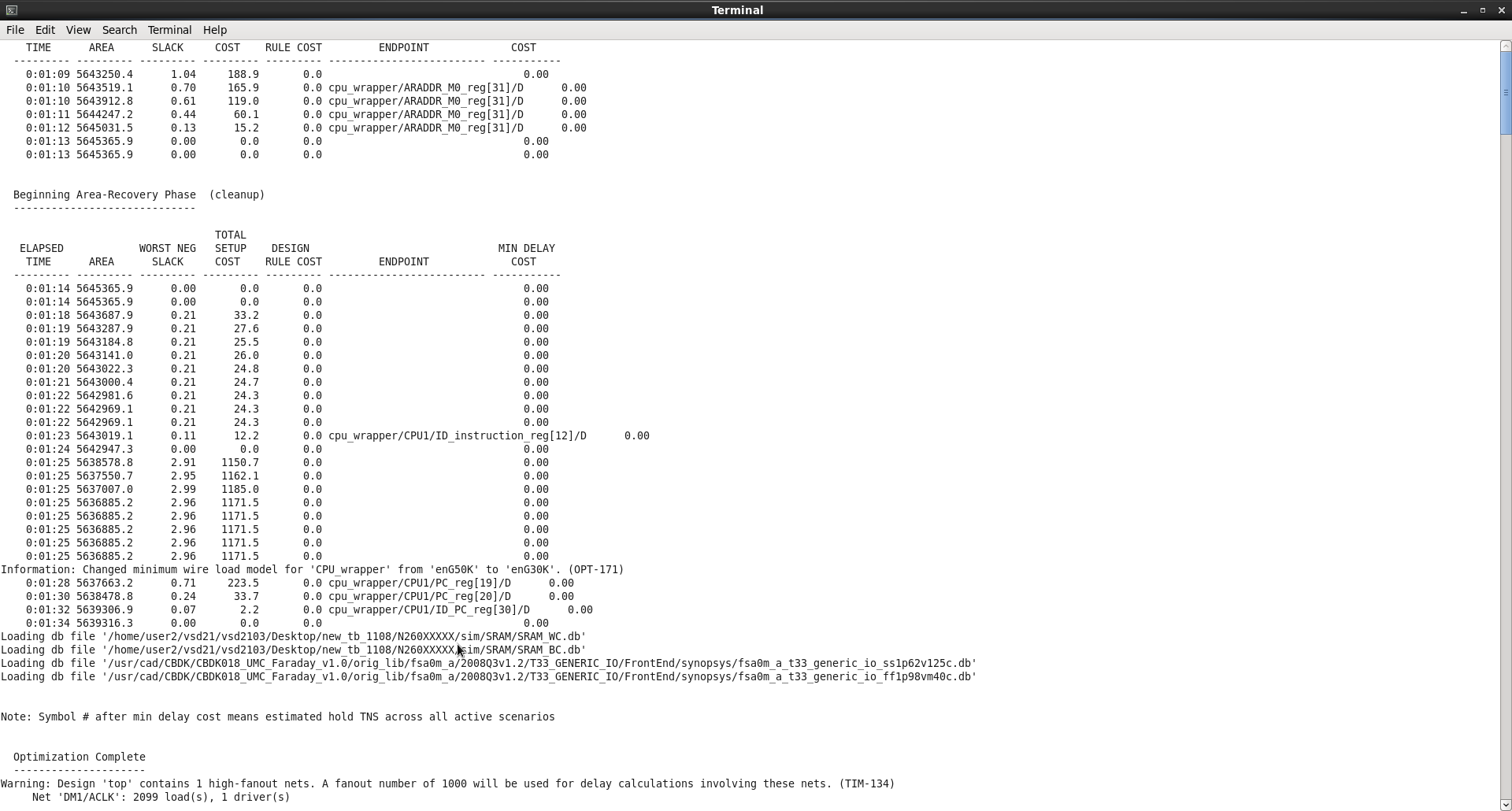
# Superlint result:



常見的warning有: 此warning為DO命名的關係，並沒有assign給他值但卻把它的值給別人，但其實我們是有給DO值的在SRAM中，只是寫法為DO[i]，賦予每一個bit值，不是直接給DO一個值，所以系統出現這個warning提示，並不影響。

# Synthesis:

* CLK -> 10
* Synthesis Area: 5639316.3



* Simulation time:
  + Prog 0: 591375 NS



* + Prog 1: 2489935 NS



* + Prog 2: 602395 NS



* + Prog 3: 636175 NS



* PA:
  + (t0 + t1 + t2 + t3) \* Area = (591375 + 2409935 + 602395 + 636175) \* 5639316.3 = 2.391 \*

# Problems encountered

在這次作業中最為令人頭疼的地方不得不說一定是AXI協定，因為對於BUS的行為本身就不熟悉，而且AXI的port數量又相當多，所以一開始對於究竟該如何下手老實說一點頭緒都沒有，光是為了對整體架構有個概念就跟同學們反覆討論過相當多次，才終於慢慢有的雛形。而寫是一回事，要去驗ABVIP又更是另外一回事，最初在驗VIP的時候常常對於它所給訊號線以及錯誤訊息毫無頭緒，印象最深刻的第一名一定是valid = 1 & ready != 1 =| valid =1 這種錯誤訊息，點開波形來看知道是valid錯了，但絲毫不知道自己錯在哪裡，因此常常必須審視整個AXI架構，來看看自己到底是哪邊出了問題，而不是只單單看那條線附近的相關變數，總結來說就是為了瞭解assert說明的問題，花上了巨大的時間去理解。  
Wrapper的部分，在寫SRAM Wrapper時遇到最大的問題就是burst length等於2的時候，必須考慮到是連續讀兩筆資料，又或者中間ready有拉下來的情況，因此最後我們決定將valid拉下1個cycle來解決這個問題，CPU Wrapper的部分遇到最大的問題是在街上CPU後跑prog0時會遇到許多在寫wrapper時不會考慮到的問題，像是連續store word與連續load word都會有timing的問題必須解決，後來我們的作法是將wrapper中的狀態機的state也一起拉進CPU做判斷，才解決問題。

# Lesson learned

鄭丞祥: 跟上次只是做為一個系統功能單元的CPU比較起來，於這次的作業加上了BUS後慢慢開始有了系統架構的雛型出來，模組之前彼此的溝通不再像是以前都假設為理想情況，資料想從誰那邊拿就從誰那邊拿，而是開始模擬真實情況要考量到許多不同模組之間的溝通問題。且AXI這項作業也讓我們學習到BUS是如何可以確保資料正確且有效的傳輸，也就是handshake，以及利用仲裁法來提升效率的實作方式，獲益良多。

陳冠佑:這次作業我主要是負責寫wrapper的部分，我覺得在這次作業中學到的是在整個系統整合debug的部分，在單獨測SRAM wrapper及CPU wrapper時會考慮的沒這麼完善，且測VIP時也都是考慮單一個module的情況，因此我們在完成VIP後，將整著系統接起來後也花了非常多時間debug，更讓我知道在系統整合的一些timing細節真的需要很多經驗，才能完成這個作業。