VLSI System Design (Graduate Level)

Fall 2020

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: \_\_黃建智\_\_ \_\_高士鈞\_\_

Student ID: \_\_N26080020\_ \_\_N26094922\_

Contribution: \_\_\_\_(55%)\_\_\_\_ \_\_\_(45%)\_\_\_\_

1. **Summary**

* 完成CPU\_wrapper, SRAM\_wrapper, AXI，且透過ABVIP驗證無cover與assert，max pending = 1
* 結合CPU\_wrapper, SRAM\_wrapper, AXI，並通過模擬prog0,1,2,3，以及合成後模擬prog0,1,2,3

1. **Check list**

|  |  |  |  |
| --- | --- | --- | --- |
| Rtl code | | synthesis | |
| prog0 | v | prog0 | v |
| prog1 | v | prog1 | v |
| prog2 | v | prog2 | v |
| prog3 | v | prog3 | v |
| superlint | **>97%** | | |

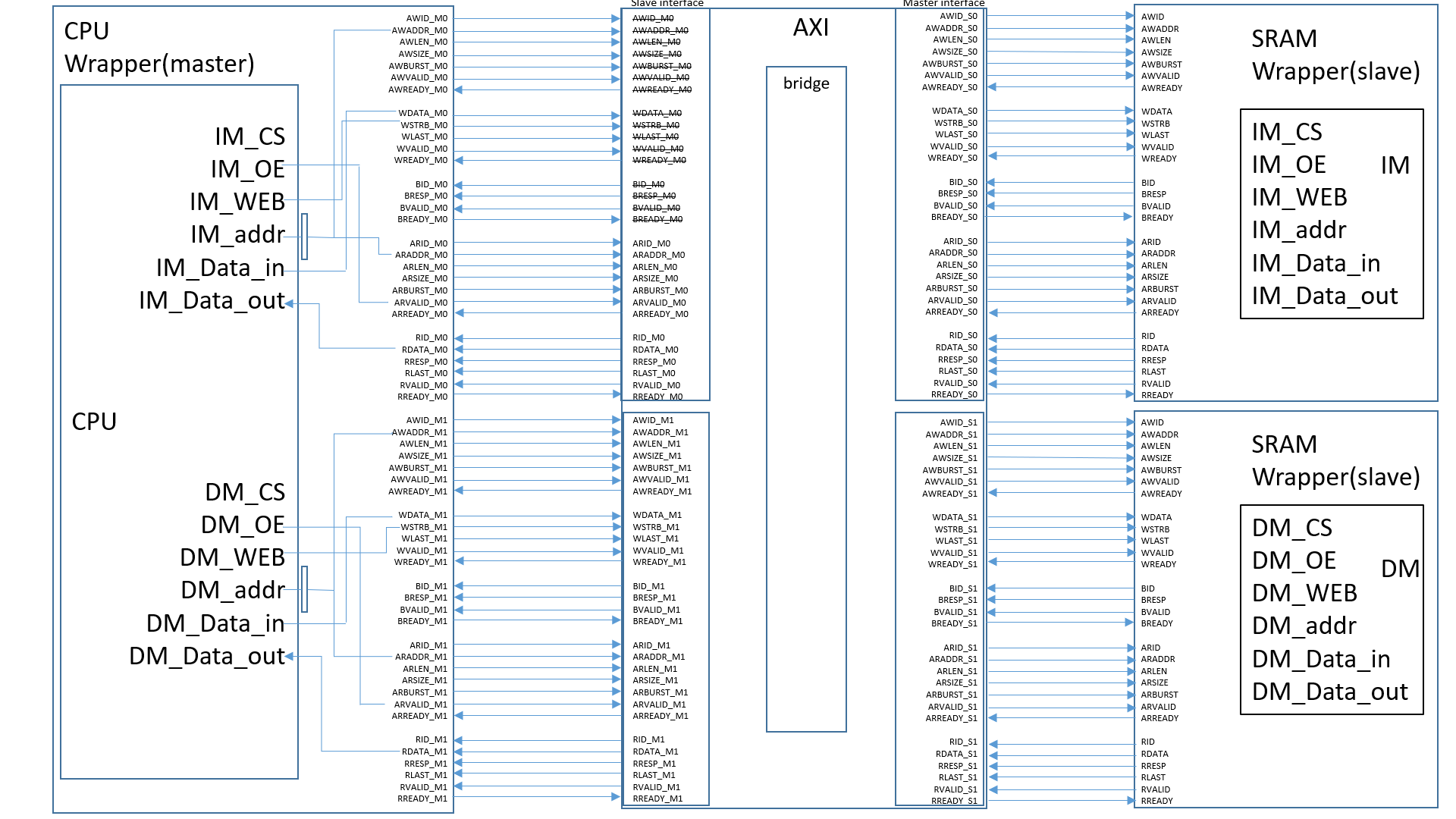
1. **Problem 1**

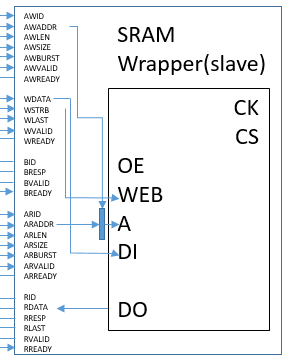
Verify the AXI architecture by using JasperGold AXI ABVIP.

Max pending number = 1

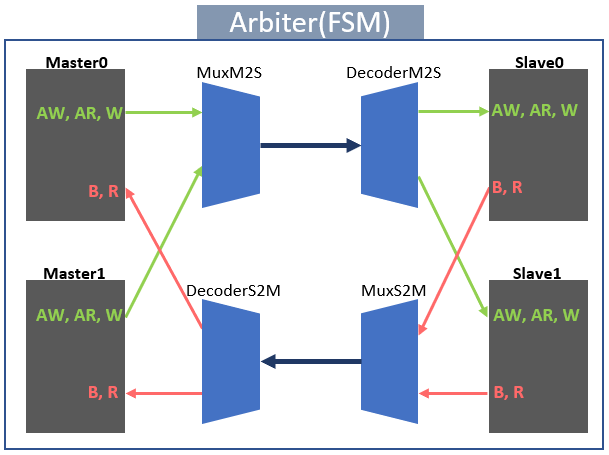
1. **Block diagram**

* **CPU\_wrapper**

由於CPU會向Instructure memory讀取資料，以及向Data memory讀取或寫入資料，因此CPU\_wrapper需要有兩個master port，這邊我們是用FSM將每個步驟區分開，讀值方面分為read addr 以及 read data，寫值方面則分為write addr, write data以及write response，以避免順序錯亂導致ABVIP驗到overflow等問題。由於讀與寫的通道是獨立的，因此讀寫是能同時進行的。當CPU的CS(chip select)與OE(output enable)同時拉高時，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，當CS(chip select)拉高與WEB(write enable)不為4’b1111時，則表示master要進行寫的動作，因此AWVALID則會拉高，等待AWREADY訊號進行寫DATA的動作。

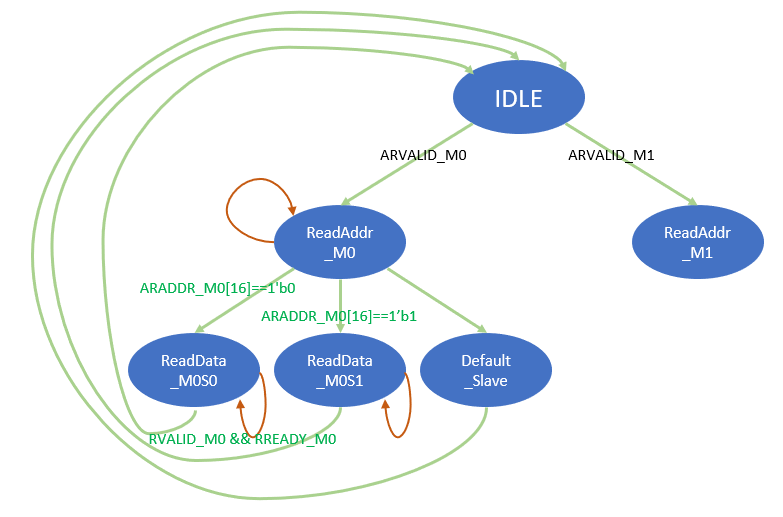
* **SRAM\_wrapper**

Slave端則比較單純，負責接收來自master的VALID訊號以及發送READY訊號給master，由於single port SRAM沒辦法同一時間做讀寫，因此雖然讀寫通道分開，但若兩master要同時讀與寫同一SRAM也是沒辦法的，須等一個做完另一個才能做。

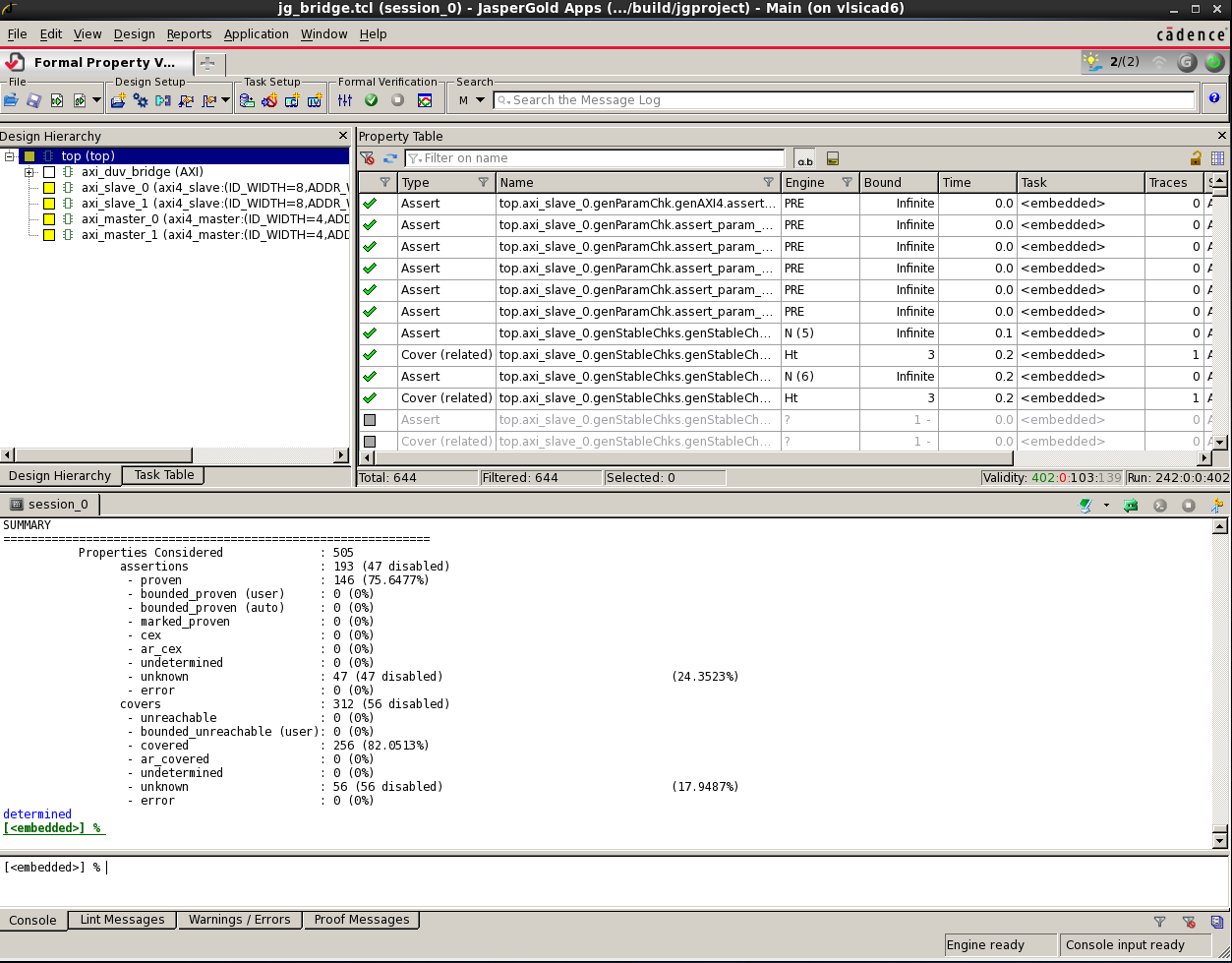
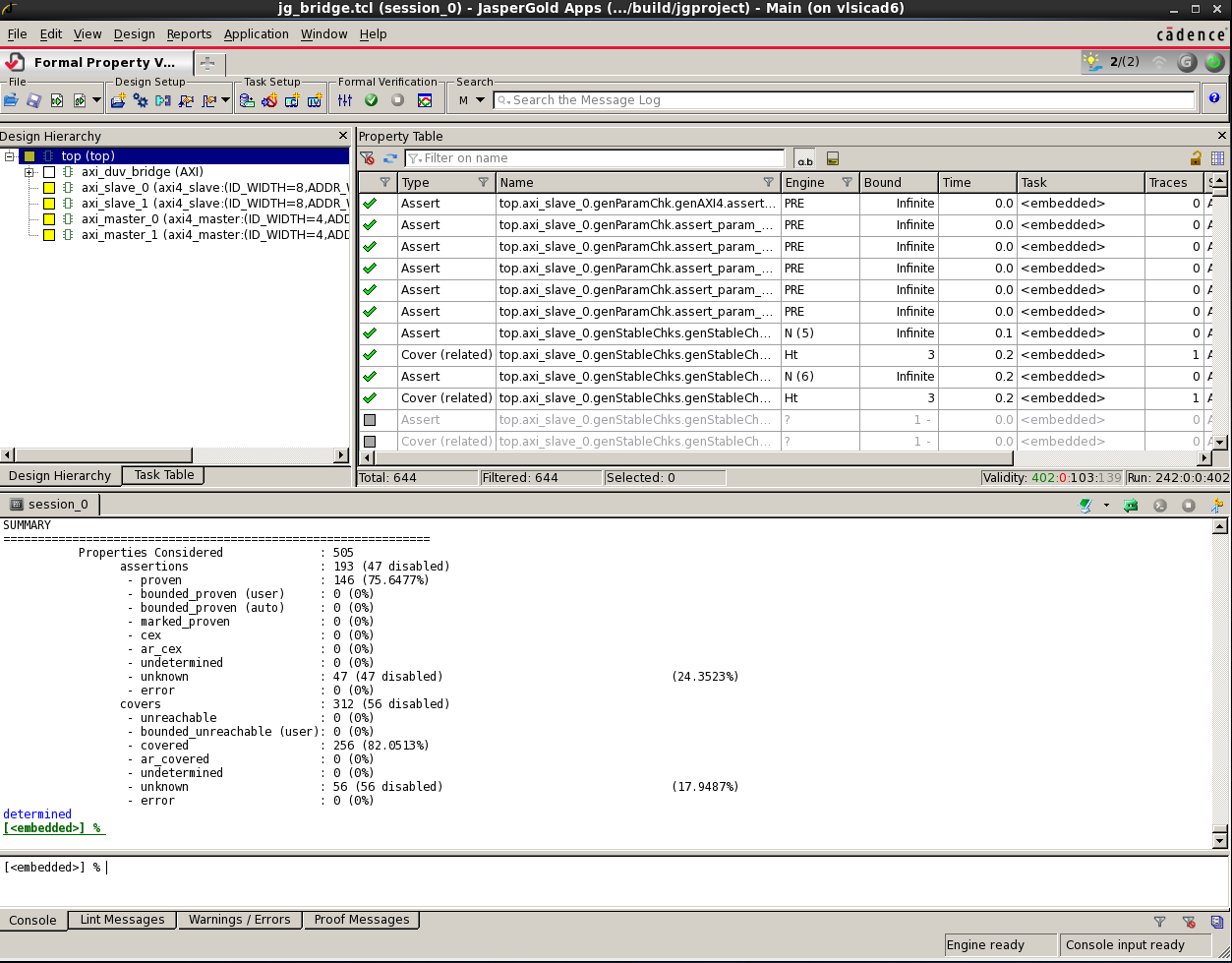
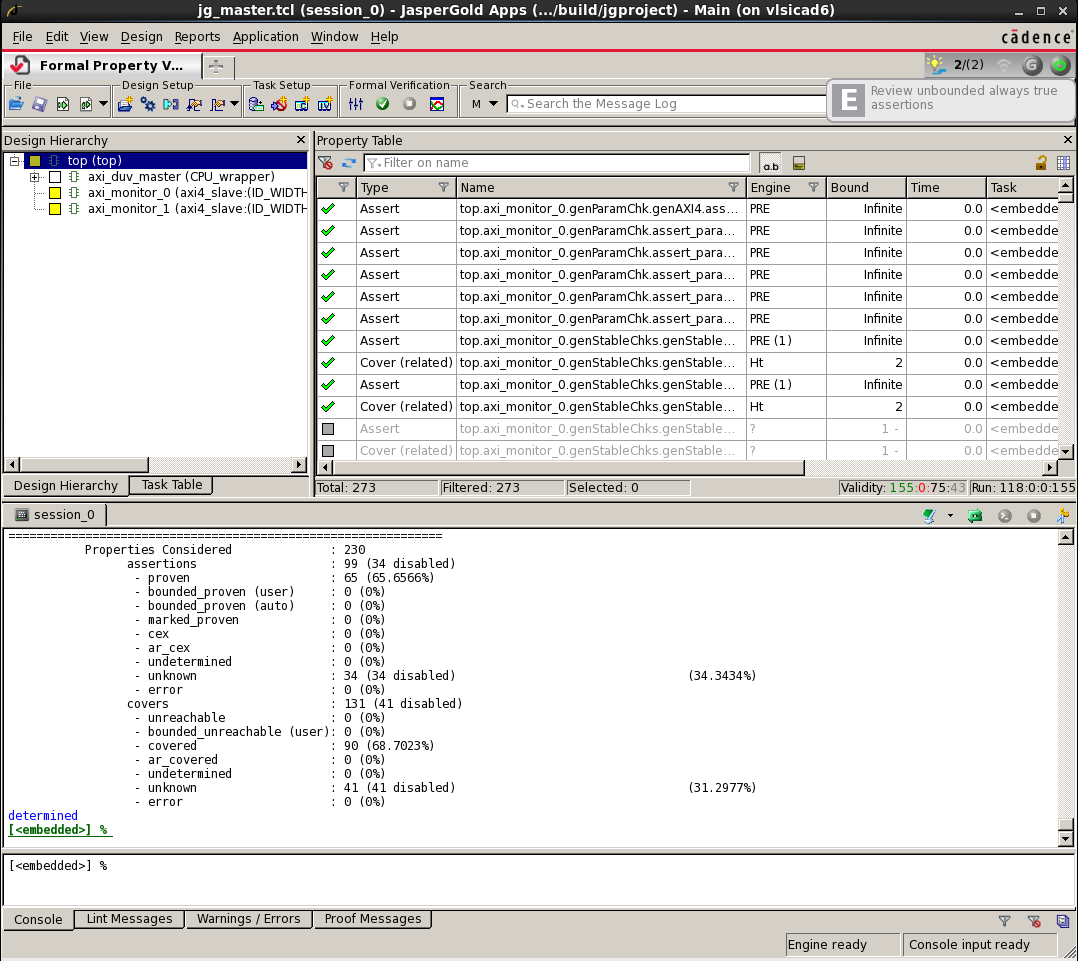
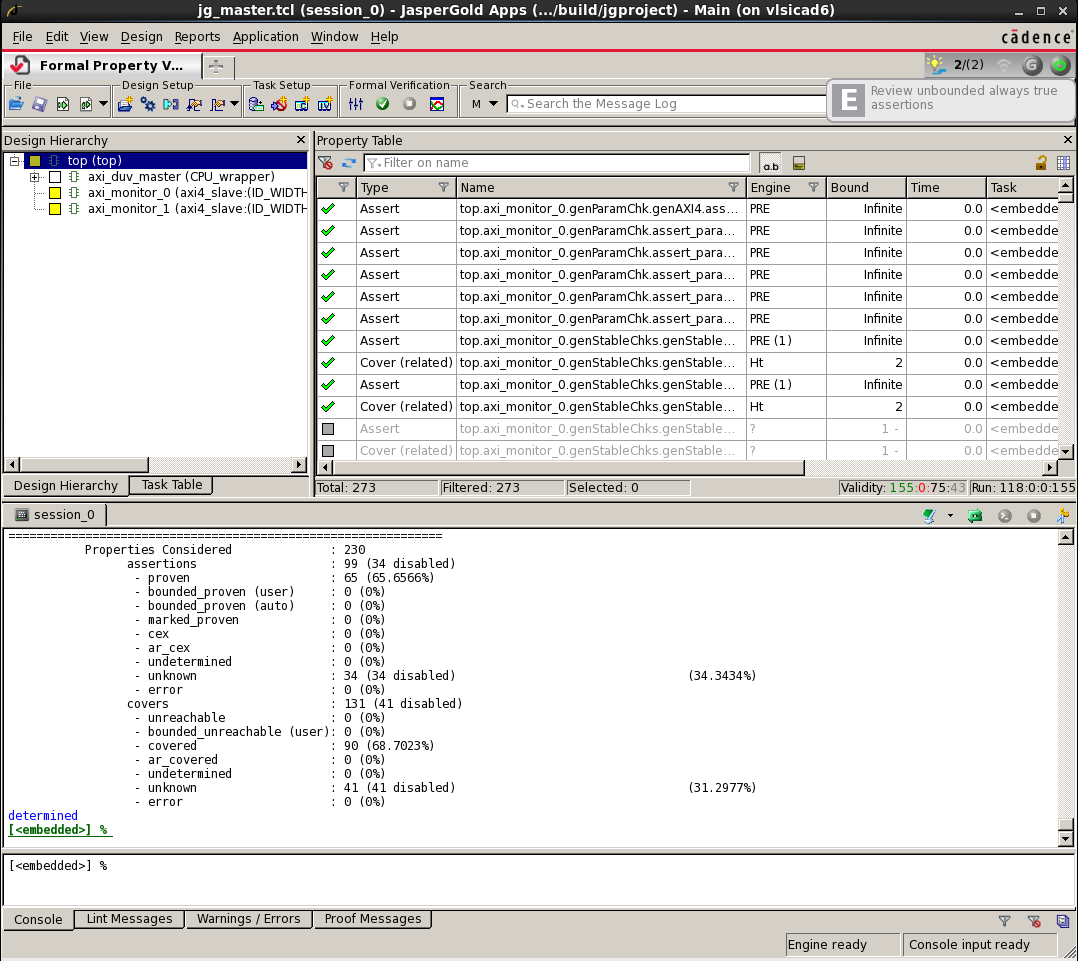
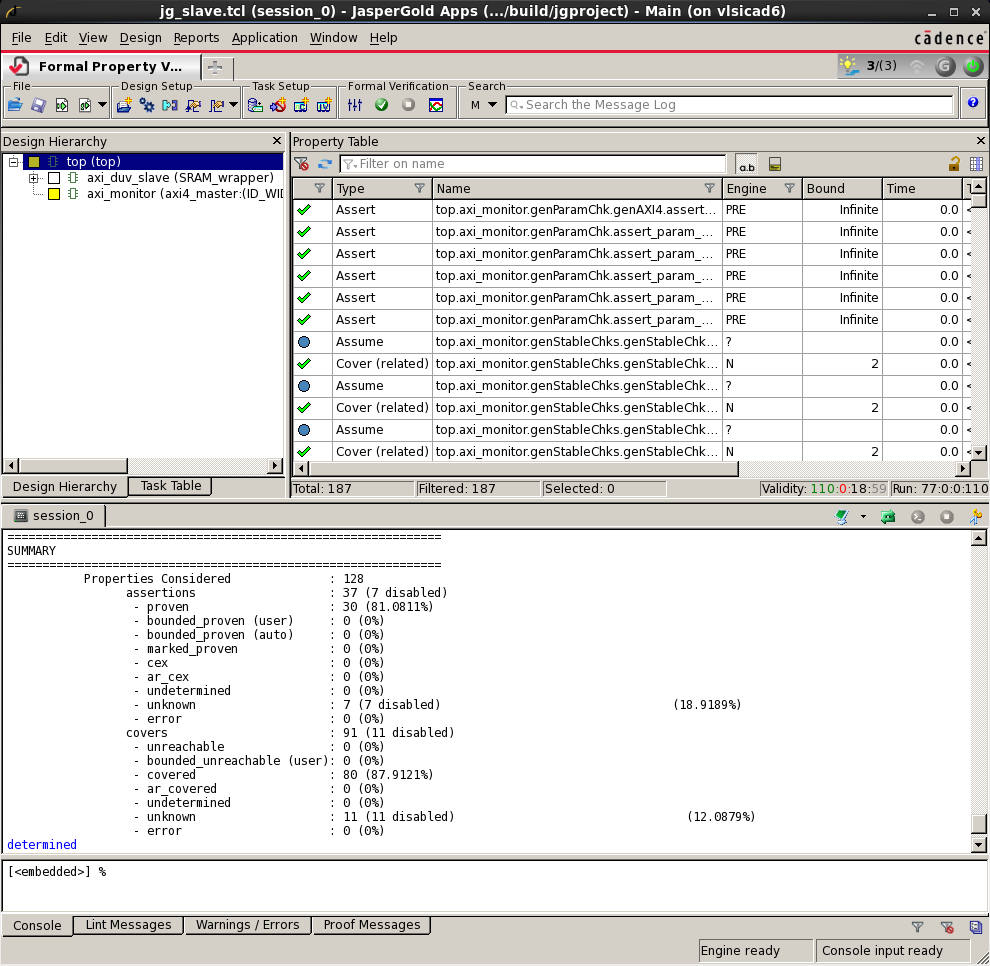
* **AXI**

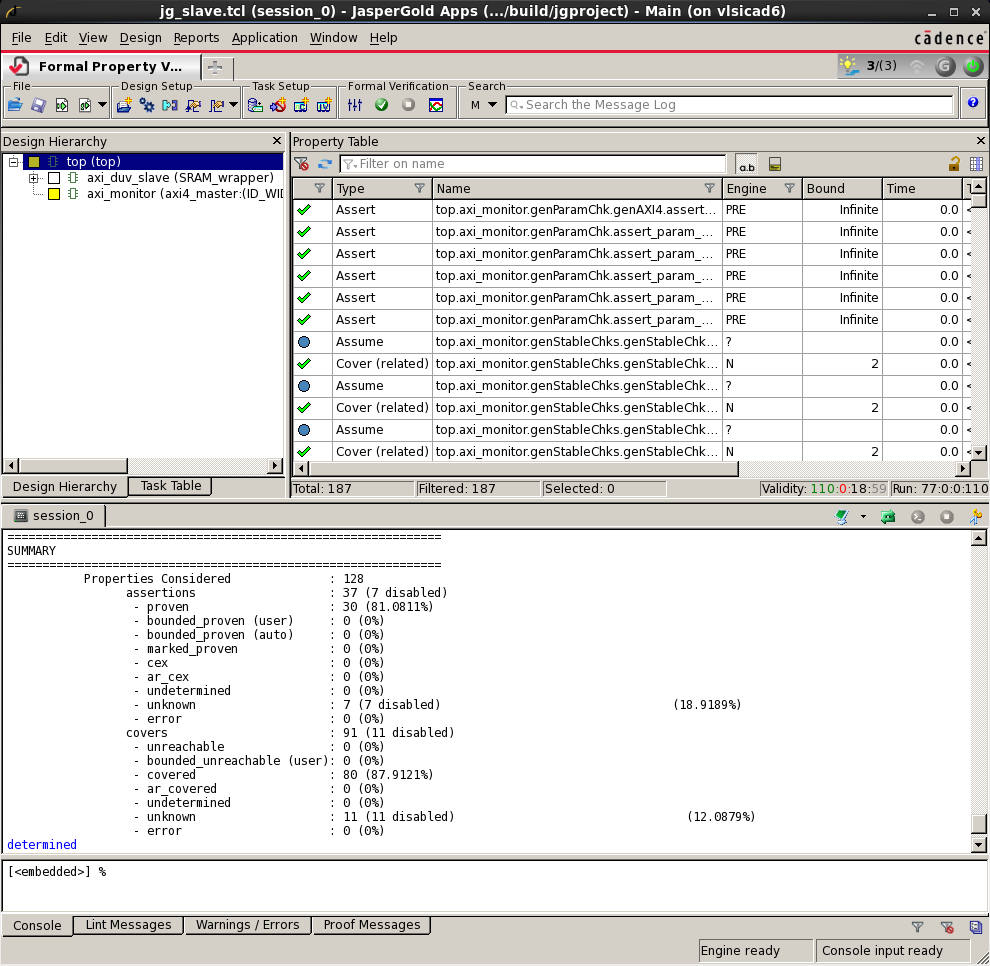
(綠色路徑: 請求及寫入資料，紅色路徑: 回傳資料)

此次作業我們參考了AXI ISA所規範的規格，設計出了一個有基礎功能的AXI介面，我們的設計主要包含了五個部分，包含負責控制整個電路運作的Arbiter，Arbiter主要是用Finite state machine的方式讓整個電路能操作在不同模式之下，兩個MUX則是根據當前Arbiter決定的優先順序，決定讓何筆資料進入BUS，Decoder再跟據資料的擁有者，把BUS的資料給予正確的slave或master。

FSM的部分這裡以Read為例(Write較複雜作圖雜亂)，預設都為IDLE狀態，直到有任何一個master發出valid訊號，則跳出IDLE，進入ReadAddr state後，會檢查slave是否回傳ready，確定ready後，會透過addr的MSB知道這個ADDR的目標slave是誰，已決定讀取對象，確認讀取對象後會進入ReadData state，在這會一直同流直到資料讀取完成的訊號出現，方跳回IDLE，Write和Read該念相同，但要特別注意Instruction memory是無法被寫入值的。

1. **ABVIP**

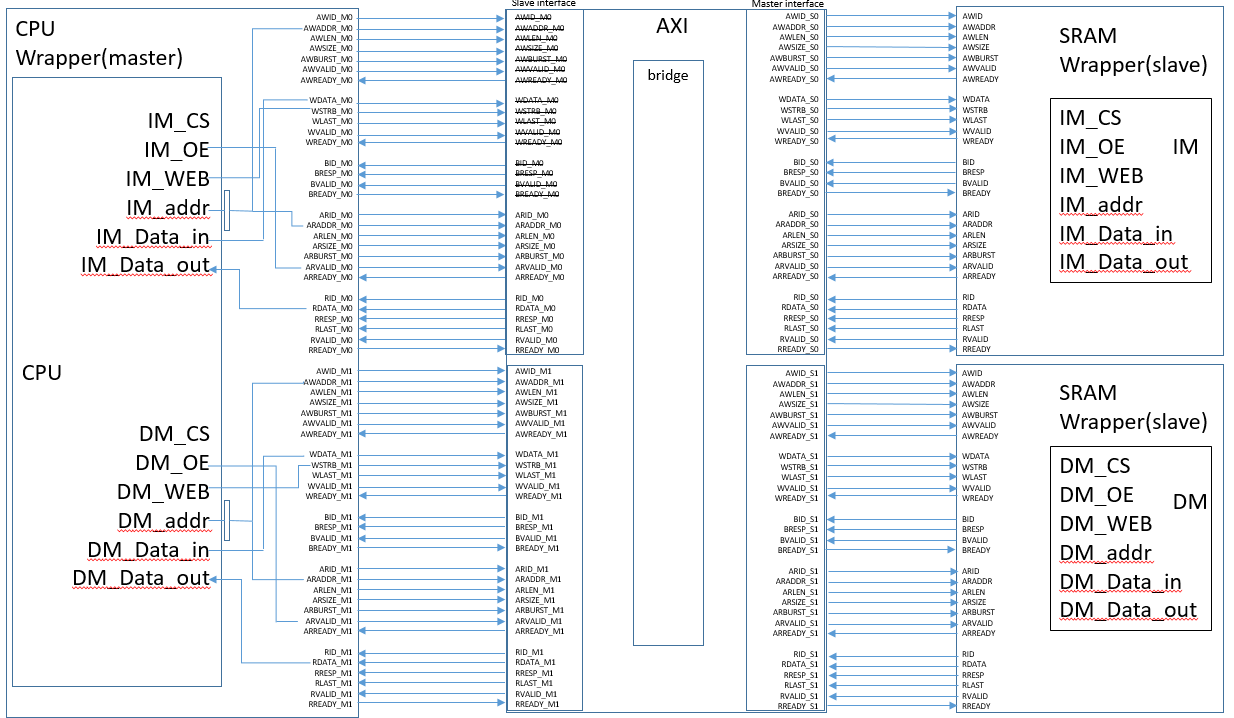
* AXI (Bridge) no cover or asser
* CPU wrapper (Master) no cover or assert
* Sram wrapper (Slave) no cover or assert



1. **Problem 2**

Combine AXI with your CPU and SRAM in Homework 1.

1. **Block diagram**

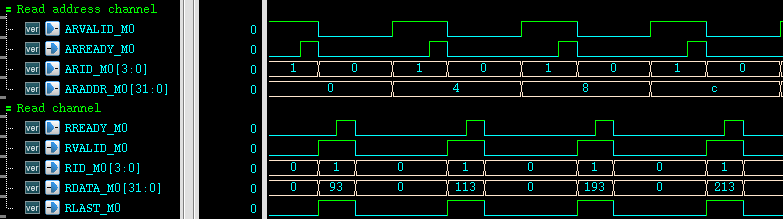


由於CPU是透過AXI向IM以及DM要資料或是寫資料，而不像作業一的CPU可以在1個clk就拿到想要的資料，因此在讀取資料的過程，CPU內部是必須stall住的，若還沒拿到正確的資料就繼續往下做，就會造成錯誤，所以我們的做法是當master0和master1都IDLE沒有讀取或寫入資料的時候，CPU才會運轉，其他時候都在stall的狀態，當兩master要同時讀或同時寫，由於共用channel，AXI的仲裁器則會判斷應該讓哪個master grant，而另一個沒有grant的則會持續發著request(VALID)等到收到slave端回傳READY才會繼續往下做。另一種情況則是其中一個master要讀取slave0而另一個master也要寫入slave0，雖然讀寫通道分開，不過single port 的memory一次只能寫或讀，因此一樣會透過AXI的仲裁器來決定先後順序，因此這邊我們是用FSM的方法來考慮各種情況，會比較周全，不過缺點是會比較慢，因為FSM的一個state就至少1個cycle。

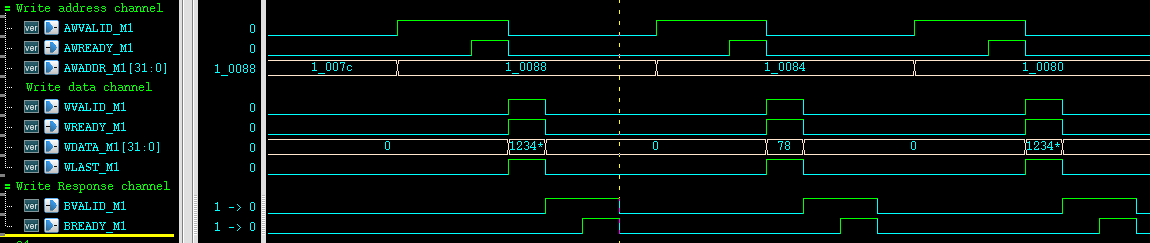
1. **Waveform**

此部分以master0 和 master1為例，觀察不同狀況之波型

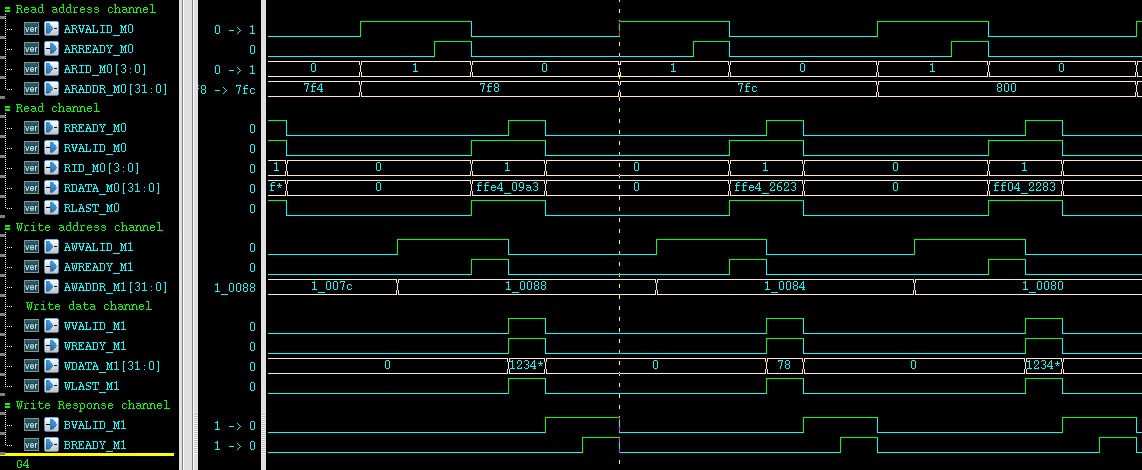
**Read data :** 由下圖可觀察出，當AR channel handshake，slave端確定收到地址後，RVALID才會拉高，當master確定收到要讀的資料，R channel handshake完成，RLAST拉高。



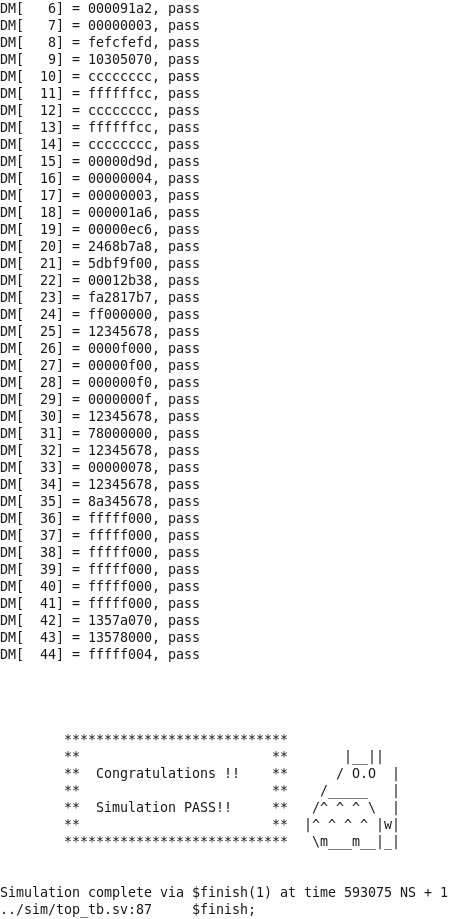
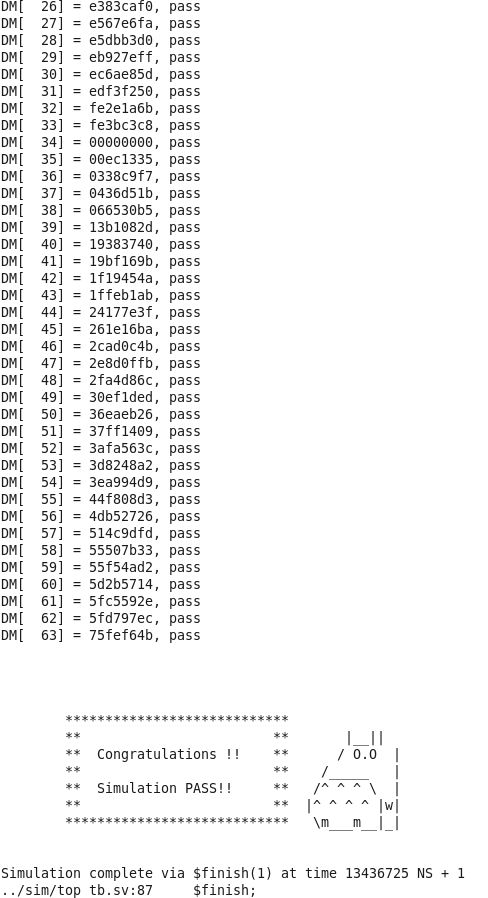
**Write data :** 由於我們是使用FSM來控制整個電路，因此，雖然從CPU發出來的address與write data是同時發出，不過我們的狀態是需先經過AW channel handshake才會開始進行寫資料的動作，因此這部分會比預想中慢一些，當W channel handshake後，會進入response的狀態，response handshake完成後，寫資料才算是正式完成。

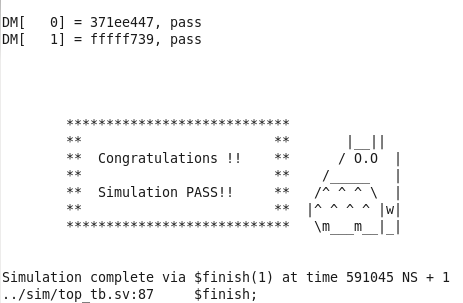


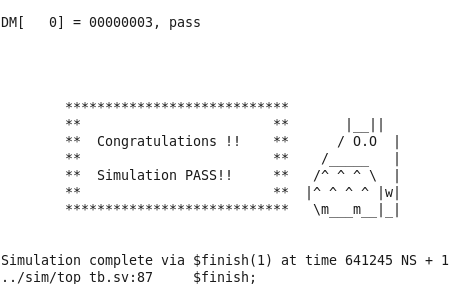
**Read data and write data :** 此狀況為master0讀取slave0資料，同一時間，master1寫入slave1資料，因為AXI的讀寫通道分開，且沒有同時讀寫同一個SRAM，因此兩master是可以同時做讀寫的動作，如下圖所示，master0持續在讀IM的instruction，而此時master1正在寫入資料到DM中。



1. **Testbench**

**Prog0 Prog1**

**Prog2**

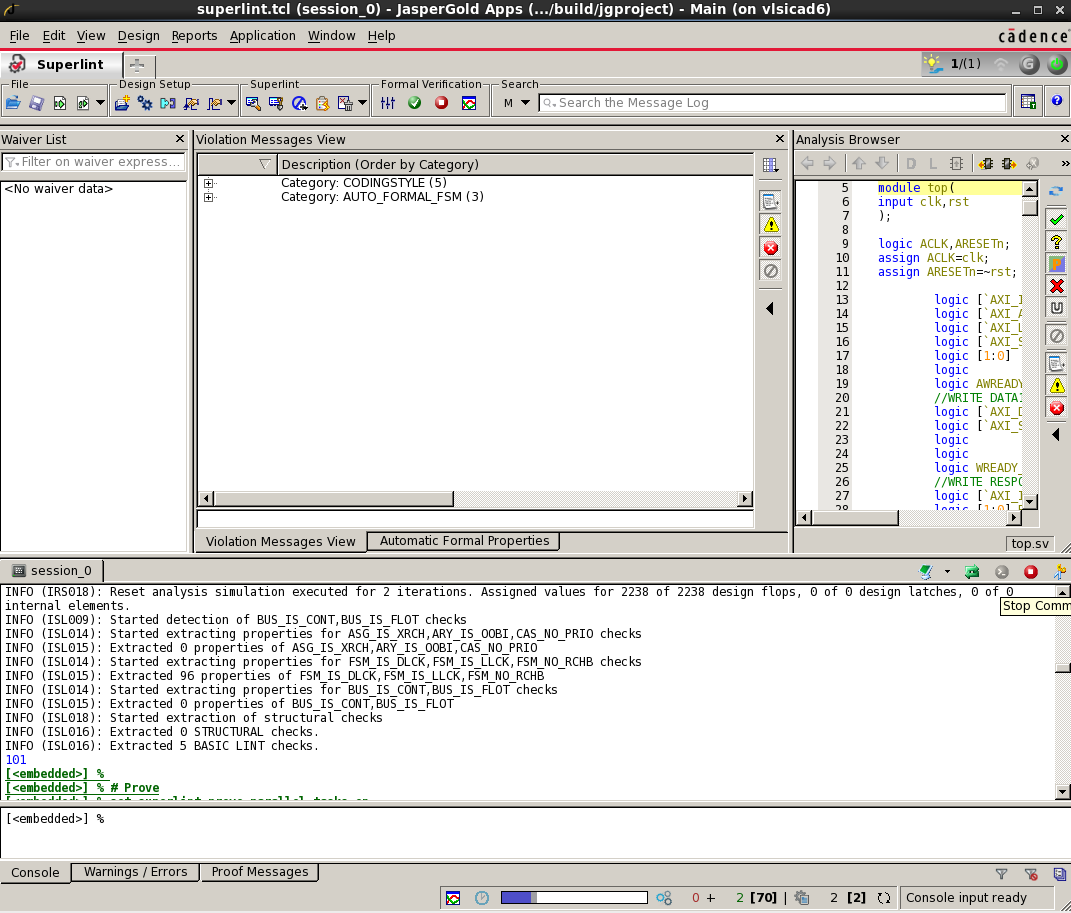
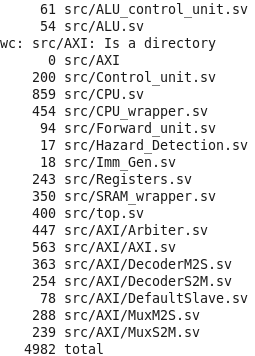
**Prog3**

**Superlint result**

常見的warning:

* + - 1. AXI在slave端與master端的ID長度不同，因此會遇到等號左右長度不一致的問題

🡪 用0將長度補滿，或是只取最後4bit以符合左右長度相同。

* + - 1. 沒有寫1’b0或1’b1而是直接寫0或1 🡪 數字前記得要加1’b 或是1’d

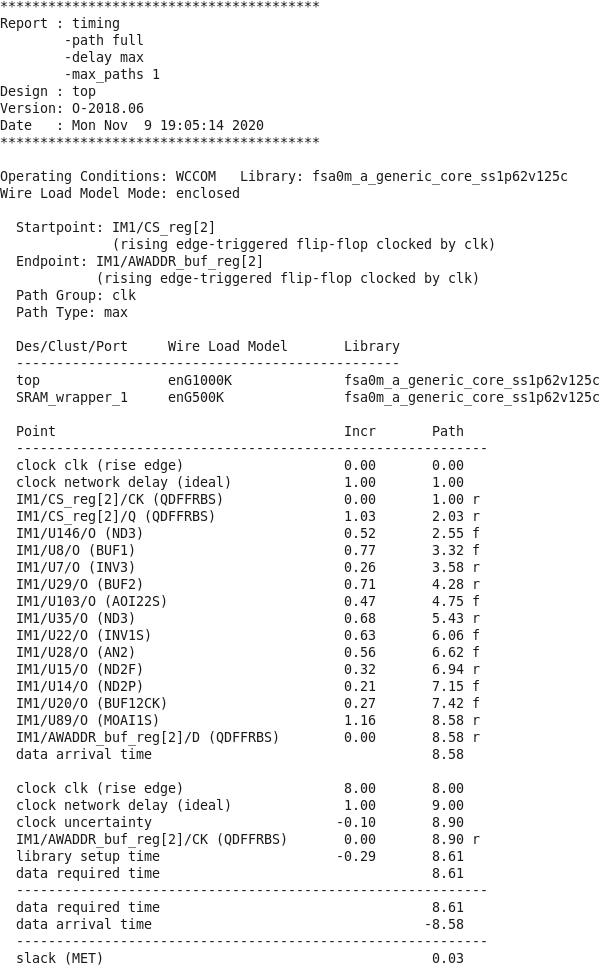
**總行數 4982**

**Warning 101**

**Complete >97%**

**Synthesis:**

Clock period =8

****Timing report:

**Describe the major problems you encountered and your resolutions:**

此次作業最大的困難為我們對AXI的協定不那麼熟悉，因此花了很多時間在了解AXI以及JasperGold的ABVIP上，若ABVIP通過之後，接起來就會容易許多。ABVIP的一些assert以及cover的constraint 一開始也花了蠻多時間在理解這些constraint的意思，才慢慢知道是哪邊沒有考慮周全，最後才決定採用FSM的寫法，一開始的版本沒有使用FSM，有許多情況因為我們沒有考慮周全而導致VIP驗不過。而Problem2比較困難的地方則是timing的問題，要讓CPU在對的時間拿到對的值，prog0有一些特殊情況像是連續LW或是連續SW就會有問題，後來透過timing上的一些修改就可以順利完成!

**Lesson learned:**

上次的作業為一個module CPU，這次則較像是一個系統的雛型，此次作業不僅對完全不熟的AXI變得非常熟悉，也對JasperGold的tool有初步的了解，這些以後我們應該也都會常常接觸到，所以把握這個機會好好學習。