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: \_王劭懷\_\_\_\_\_\_\_\_\_

Student ID: \_M16121077\_\_\_\_\_\_\_\_\_\_

1. **Summary**

在此次作業中，我完成了Master AXI、Bridge AXI和Slave AXI的設計，並順利通過prog0至prog6共7個測試。使用的cycle time為1.0，整體合成面積為18887.767891。在prog0的測試中，運行耗時13735個cycle，小於hw1 CPU的cycle數的2倍，原因是我在本次作業針對CPU做了一些修改，將branch或jump的address移至pc的flip-flop後面，也就是直接給予IM，這樣每次flush時都只要flush 1個cycle即可，對於prog0可節省約一千多個cycle。在Superlint驗證方面，我的程式碼僅有2個warning，代碼總行數超過3000行，warning佔比不到1%。在ABVIP驗證結果中，Slave與Bridge均未出現任何counterexample（CEX）或uncover的項目，而Master雖無CEX，卻有13個uncover。這是因為在AW/AR/W的handshake完成後，Ready信號在下一個cycle立即拉高，無法覆蓋到RVALID && (!RREADY)的情況，因此有uncover存在。

1. **Explanation of my design**

我在本次的作業中，三個Wrapper皆使用FSM做輸出訊號的安排，且分為Read channel 和 Write channel。

**(1)SRAM\_wrapper的Read channel的Finite State Machine如下:**

一張含有 文字, 圖表, 行, 圓形 的圖片

自動產生的描述

在state s\_wait時，若write\_now的flag為0，則ARREADY\_S會維持為1，而若ARVALID\_S和ARRREADY\_S同時為1時，會將A\_read給予address ARADDR\_S，並同時將A\_read暫存至A\_reg，ARID\_S暫存至RID\_S，ARLEN\_S加1暫存至ARLEN\_INCR。

在state s\_SEND\_DATA時，會將ARREADY\_S拉為0，RVALID\_S與readnow拉為1，並且將A\_read給予先前存取的A\_reg避免給予memory的address有任何更動，RDATA\_S則輸出IM的data DO，並持續將DO暫存至register DO\_reg，而當RVALID\_S和RREADY同時為1時，Flip-flop ARLEN\_INCR會減1，當判斷到ARLEN\_S為1時則將RLAST\_S拉為1。

在state s\_response時與s\_SEND\_DATA的操作差不多，多加此一state的用意是與write channel的state數相同，避免同時執行時會有不同步的狀況發生。

**(2)SRAM\_wrapper Write channel Finite State Machine 如下 :**

一張含有 行, 圖表, 圓形, 文字 的圖片

自動產生的描述

在state s\_wait時，會將A\_write設為0，且若readnow為0的話AWREADY\_S會一直拉為1等待handshake。並且當AWVALID\_S和AWREADY\_S同時為1時會將AWADDR\_S和AWID\_S暫存至waddr\_reg和BID\_reg的暫存器中。

在state s\_SEND\_DATA時，將AWREADY\_S拉回0，並會將writenow拉起為1避免read也同時執行，此用意是為了能通過ABVIP的arvalid && (!arready) unreachable的情況，若沒有此控制訊號會導致arvalid為1時arready都一直是為1，故導致此條件無法測試。而當WVALID\_S 和 WREADY\_S同時為1時，會將A\_write 輸出waddr\_reg，BWEB輸出WSTRB\_S，又因為WSTRB\_S只有4個bit，因此會將其每個bit擴展為8個bit而達成32bit的輸出。

在state s\_SEND\_RESPONSE時，因為已做完寫入的動作，故回傳BVALID\_S拉為1，且BID\_S輸出BID\_reg，BRESP\_S輸出2’b00，表示OKAY的資訊。並在BVALIID\_S 和BREADY\_S同時為1時回到s\_wait的state。

**(3)Bridge 的 AXI Read channel 的Finite State Machine 如下:**

一張含有 行, 繪圖, 圖表, 圓形 的圖片

自動產生的描述

當ARVALID\_M0 或 ARVALID\_M1成立時，即會進入下個state。在state為1’b0時，若ARVALID\_M0拉為1，則rmaster\_sel為1’b0，若ARVALID\_M1拉為1，則rmaster\_sel為1’b1，若兩者同時為1時，rmaster\_sel為pre\_rmaster\_r的invert，若兩者同時為0時，rmaster\_sel為pre\_rmaster\_r，pre\_rmaster\_r則為前次的master暫存起來的值，此種作法即為round-robbin的arbiter架構，當兩者同時要讀取時，先前完成過handshake的優先次序則排至最後，這樣一來便可達到輪流讀，不會有某一個channel被霸佔的情形。並且根據rmaster\_sel和對應的address[31:16]的部分判斷slave應該選擇哪slave0或slave1，output logic即會根據選擇到的master與slave做對應的連結，如rmaster\_sel為1且rslave\_sel為1時，會將ARVALID\_S給予ARVALID\_M1，以此類推做連接。為了避免rmaster\_sel被更動導致連接變動的情況(如ARVALID\_M1先拉起但下個cycle還沒等到ARREADY時ARVALID\_M0也拉起，會導致rmaster\_sel原先選擇1改成選擇pre\_rmaster\_r的invert)，故會在收到其中一個ARVALID為1時便進入下個state，且同時將rslave\_sel、rmaster\_sel暫存到rslave\_dat\_sel、rmaster\_dat\_sel。

在state為1時，即可將所有AR/R的channel根據rmaster\_dat\_sel與rslave\_dat\_sel做連結，連結的方式如在state0時相同，而當選到的master和slave同時拉起RVALID、RREADY、RLAST便可回到state 0等待下次的handshake。

**(4) Bridge 的 AXI Write channel 的Finite State Machine 如下:**

一張含有 圖表, 圓形, 行, 文字 的圖片

自動產生的描述

在cs\_w為0時，因為master只有一個不需要做arbiter，故只需透過AWVALID\_M1為1時判斷其對應到的slave為何，且將其與對應的slave做連結，並在判斷AWVALID\_M1和AWREADY\_M1同時為1時暫存wslave\_sel於state為1做判斷連結的slave，當BREADY\_M1和BVALID\_M1同時為1時即回到state0

**(5) CPU\_Wrapper master0的read channel FSM如下:**

一張含有 圖表, 行, 文字, 螢幕擷取畫面 的圖片

自動產生的描述

在state s\_wait時，若read\_im為1，則ARVALID\_M0和stall\_im拉為1，且ARADDR\_M0為CPU輸出的iaddr，並同時將iaddr、read\_im存入暫存器iaddr\_reg和read\_now0，且將write\_now0的暫存器存為0。若在s\_wait時read\_im和ARVALID\_M0和ARREADY\_M0同時皆為1，則直接進入s\_data的state，若非，則先進去s\_addr做等待handshake。

在s\_addr的狀態時，ARVALID\_M0拉為與先前read\_now0暫存的值相同，且ARADDR\_M0輸出iaddr\_reg的值，若發生ARVALID\_M0和ARREADY\_M0同時為1則進入s\_data的state，多此一state的用意是將ARREADY和ARADDR暫存起來以確保這些值會維持穩定不會有變動。

在state s\_data時輸出RREADY\_M0為read\_now0，且若RVALID\_M0、RREADY\_M0和RLAST\_M0同時為1且write\_dm為0時，會將stall\_im拉回為0，並將RDATA\_M0和ddata\_reg寫入暫存器inst和ddata，傳回CPU讓其做接下來的運算。若write\_dm為1時，為了保持read/write所花的cycle數相同，此時stall會保持為1，且RDATA\_M0存入inst\_reg的暫存器中，並進入s\_response state。

若是同時為read/write的話，進入s\_response state才將ddata\_reg 寫入ddata和inst\_reg寫入inst，並且在s\_response將stall\_im拉為0。

**(6) CPU\_Wrapper master1的read channel FSM如下:**

一張含有 行, 文字, 螢幕擷取畫面, 圖表 的圖片

自動產生的描述

在state s\_wait時，要先確定目前allow\_read\_dm的flag為1才可進入其他state，而allow\_read\_dm的判斷是read\_dm為1但lock\_dm不為1時，這樣做的用意是怕dm占用channel，故在ARVALID\_M0 和ARVALID\_M1同時為1時，lock\_dm的暫存器會拉為1，又因為arbiter的設計，CPU遇到IM和DM要同時讀取的時候，會先執行DM，並在下一次IM拿到資料前都把DM lock住，進而即使dm的ARVALID又拉為1也不會重複執行DM的handshake。而當ARVALID\_M1、ARREADY\_M1且allow\_read\_dm同時為1時進入s\_data的state，否則只有allow\_read\_dm為1時進入s\_adddr等待handshake，若都不成立則停留在s\_wait的狀態。在s\_wait狀態時的輸出會根據allow\_read\_dm決定ARVALID\_M1的輸出，以及daddr會做為ARADDR\_M1的輸出。

在s\_addr的狀態時，ARADDR\_M1持續輸出daddr，不需要在前面暫存的原因是因為此時的CPU都還是處於stall的狀態，故daddr不會有變動，所以只要持續輸出daddr就好。ARVALID\_M1也持續拉為1。

在s\_data的狀態時，RREADY\_M1會拉為1，並等待RVALID\_M1與RREADY\_M1與RLAST\_M1同時為1時回到s\_wait的state。

**(7) CPU\_Wrapper master1的write channel FSM如下:**

一張含有 圖表, 行, 螢幕擷取畫面, 圓形 的圖片

自動產生的描述

在s\_wait的狀態時，AWVALID\_M1會依CPU的輸出write\_dm決定應該輸出1或0，而AWADDR\_M1也輸出CPU的位址daddr，當AWVALID\_M1和AWREADY\_M1同時為1時會進入到s\_data的state做寫入資料，WVALID\_M1拉起為1，WDATA\_M1輸出CPU的output datawrite，WSTRB\_M1則是根據CPU的BWEB output (在我的CPU取名為bit\_enable)每8個bit做or並反向做輸出，並在WVALID\_M1與WREADY\_M1與WLAST\_M1同時為1時輸出進入s\_response的狀態。

在s\_response的狀態就將BREADY\_M1拉起為1，等待BVALID\_M1也為1時回到s\_wait的狀態等待下次handshake。

**(8) Waveform of Read Slave 0:**

一張含有 螢幕擷取畫面, 多媒體軟體 的圖片

自動產生的描述

ARVALID時將iaddr作為ARADDR\_M0，且在下個cycle即會透過在RVALID為1時的RDATA\_M0拿到SRAM回傳的資料，並在前一個cycle的stall為0時回傳到CPU作為下個instruction作執行。

**(9) Waveform of Read Slave 0 & Write Slave 1:**

一張含有 螢幕擷取畫面, 多媒體軟體 的圖片

自動產生的描述

當遇到同時要write DM時，因為write有多一個cycle要等待Response，故在READ的channel兩個cycle做完讀取資料後，仍然會stall住等待write的操作，而write channel是在AWVALID\_M1為1時，AWADDR\_M1傳送CPU欲寫入的地址daddr，並在WVALID\_M1時傳送WDATA\_M1欲寫入的資料datawrite，並在response的handshake成功的同時將stall拉為0，使得下個cycle CPU可執行得到的新instruction。

**(10) Waveform of Read Slave 0 & Read Slave 1:**

一張含有 螢幕擷取畫面, 行 的圖片

自動產生的描述

當遇到同時IM和DM都要讀取時，ARVALID\_M0和ARVALID\_M1同時都為1，但會先讓DM做handshake，傳送CPU欲讀取的資料地址daddr，並在下個cycle取得回傳的資料。為了要保存資料直到IM也做完，會將RDATA\_M1在RVALID\_M1和RREADY\_M1同時為1時取得的值寫入ddata\_reg的暫存器當中，且在執行DM開始時即會將lock\_dm拉起，讓執行完第一次handshake後的M1不會重複執行handshake。並且在M1取得資料後透過arbiter讓M0接著讀取IM的資料，在讀到IM資料的同時才將stall拉回0，且一併將IM和DM的資料寫回CPU當中。而之所以可以每次都先做讀DM再做讀IM的操作是透過Arbiter做處理，因為CPU每個cycle都一定會做讀取IM資料。因此Arbiter做round robbin的設計會將前次做handshake的master記錄下來，而當IM和DM要同時讀取時，會選與前次相反的master，故使一開始master\_sel都會選到DM先做處理後再做IM，slave\_sel則再根據master欲要求存取的地址再做解碼判斷。

1. **Block Diagrams**
2. SRAM\_wrapper的 Block Diagram如下圖所示:

一張含有 文字, 圖表, 行, 平行 的圖片

自動產生的描述

1. CPU\_wrapper的 Block Diagram如下圖所示:

**一張含有 文字, 圖表, 平行, 行 的圖片

自動產生的描述**

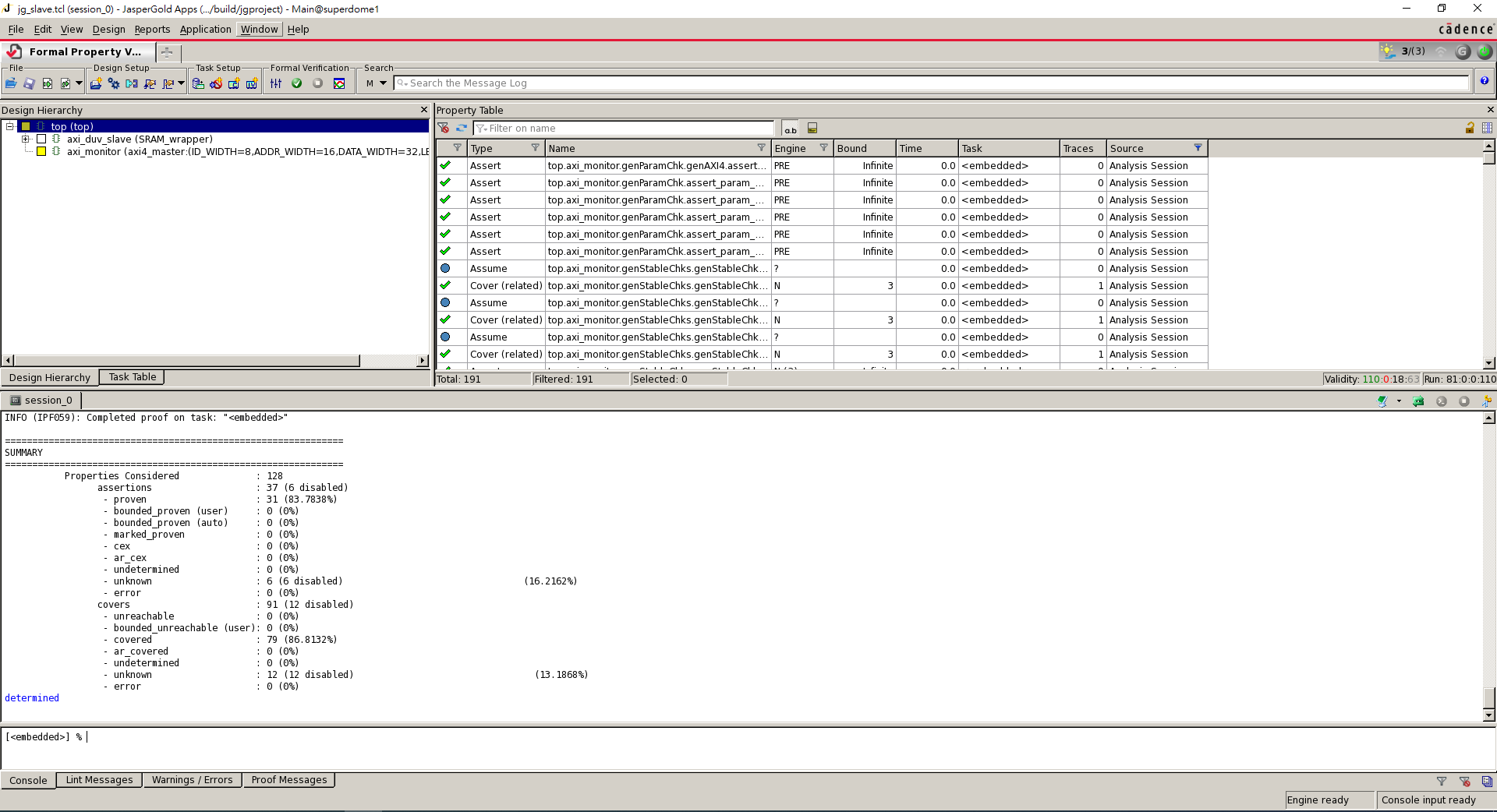
1. Bridge 的 Block Diagram如下圖所示:

**一張含有 圖表, 方案, 工程製圖, 圖解 的圖片

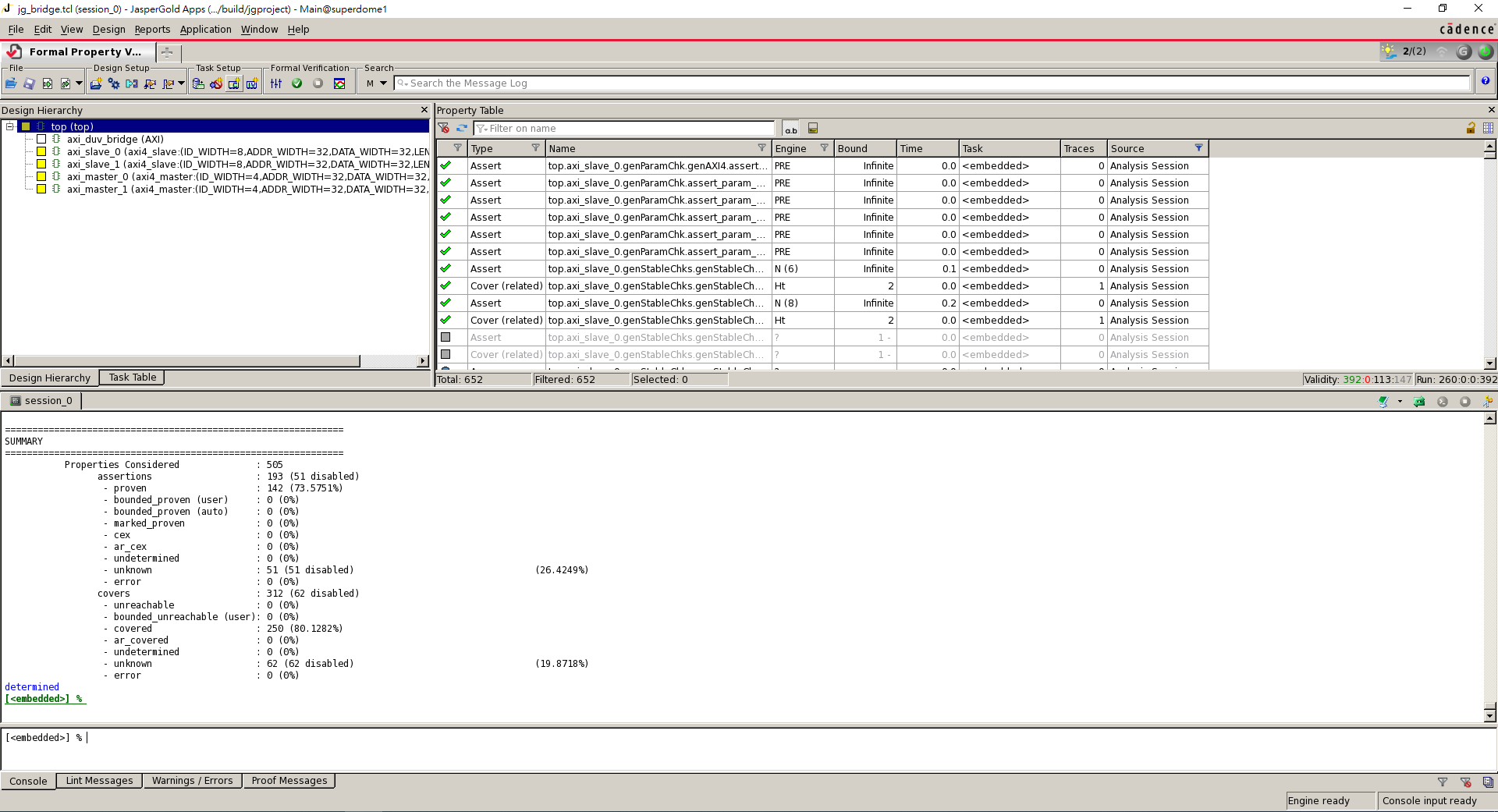
自動產生的描述**

1. **Results of Verification**

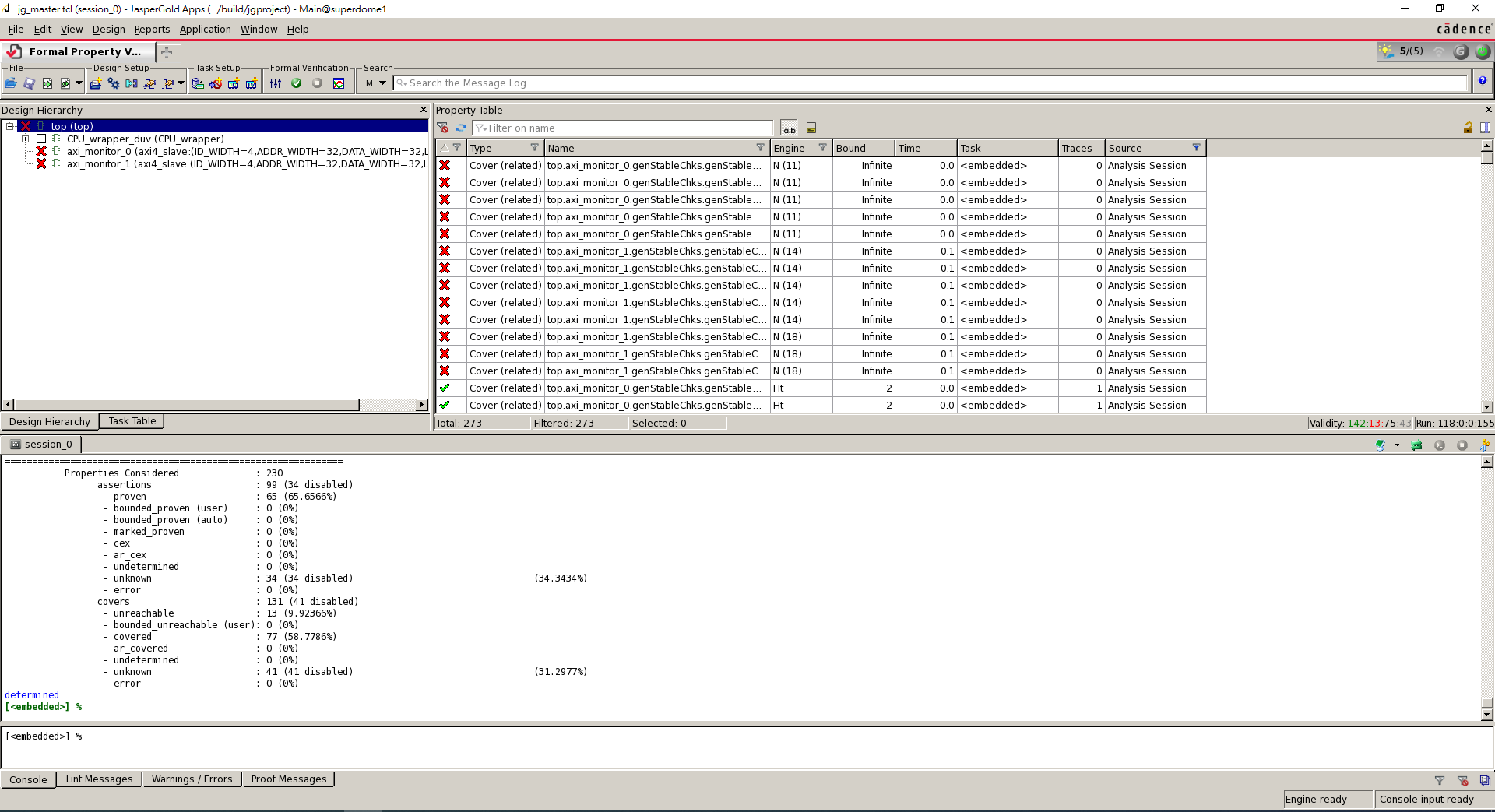
(1)SRAM\_wrapper

****

(2) Bridge

****

(3)CPU\_wrapper

****

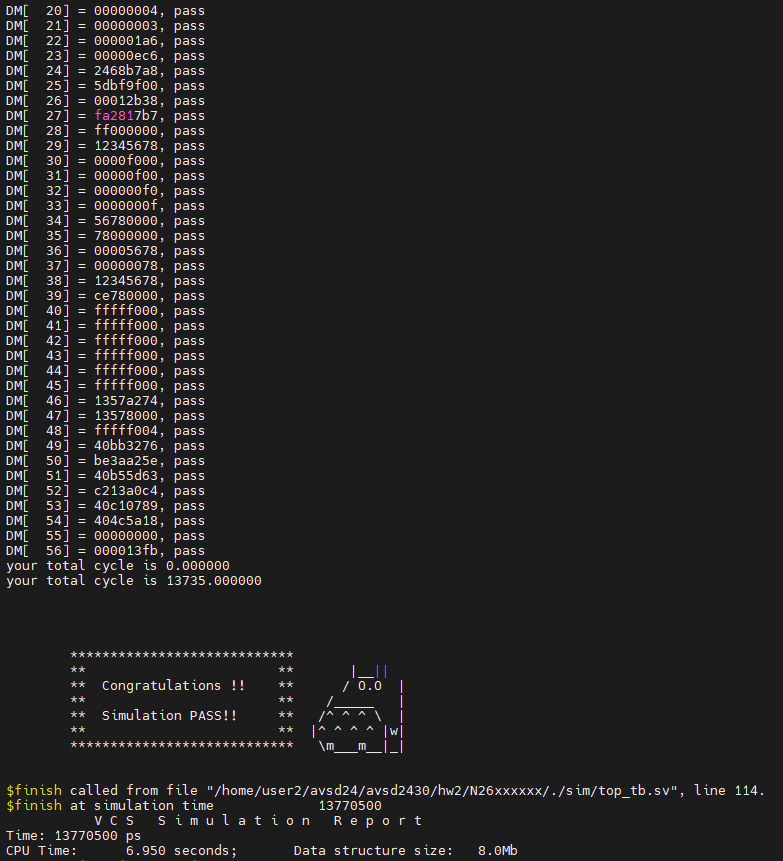
1. **Results of Simulation**

我在prog0到prog1的Simulation結果皆為All Pass

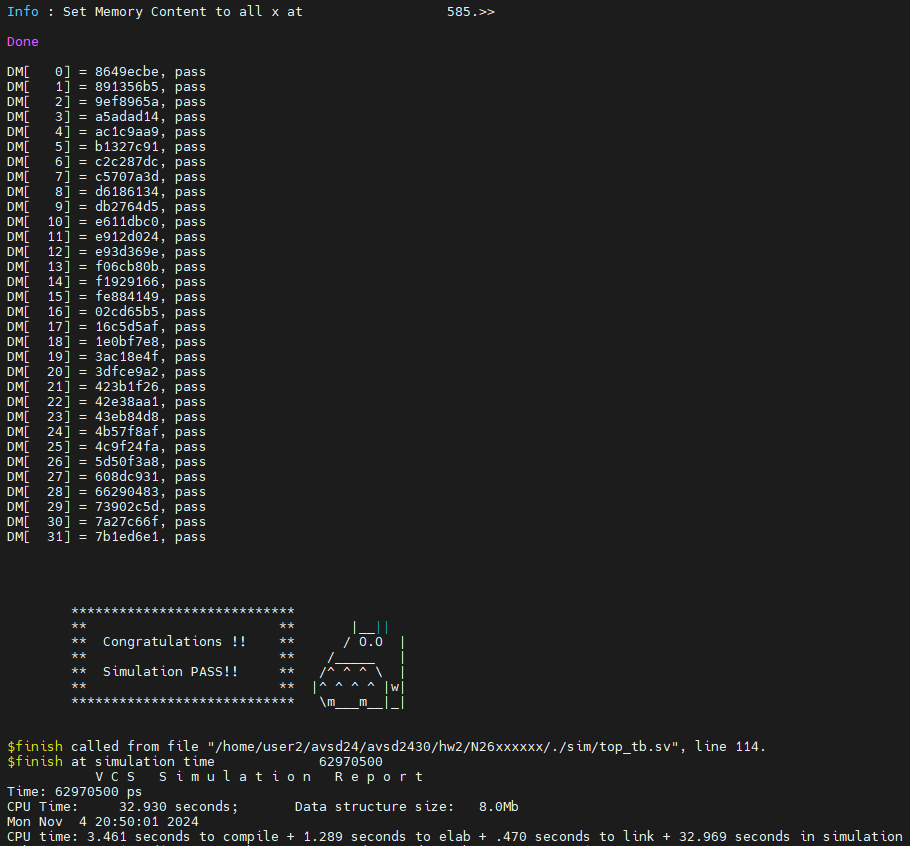
1. Prog0

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述



1. Prog1



1. Prog2



1. Prog3



1. Prog4

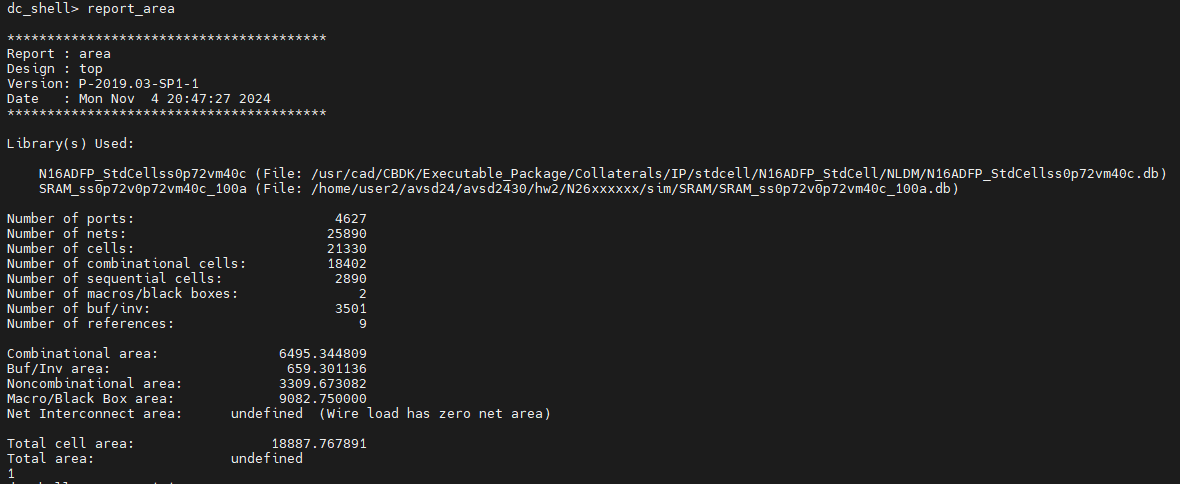


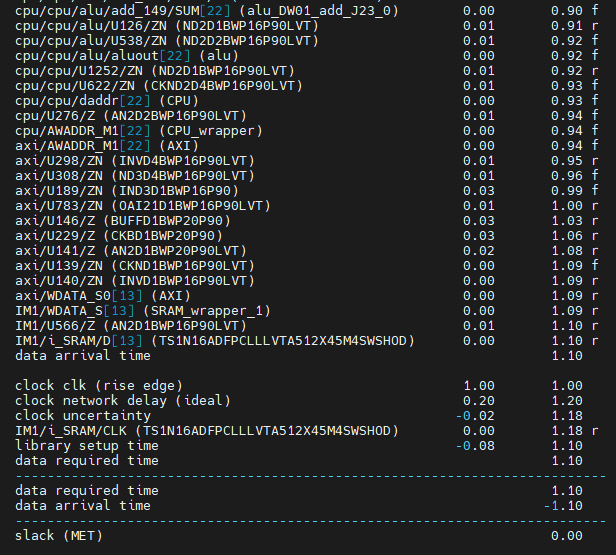
1. Prog5

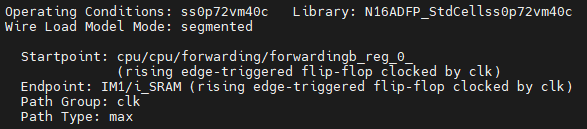


1. Prog6



(8)合成面積為18887.767891

(9)Slack 為0.0

(10)Critical path 為forwardb到IM

1. **Superlint Results**

在執行Superlint中遇到最多的warning和error都是bit數有落差導致的，解決方法就是將其bit數對齊應該對應的bit數，且如2bit的資料若直接寫constant都要寫2’b00之類的，不可直接寫阿拉伯數字。還有一個常遇到的是宣告的變數但沒有用到，只要將多餘宣告的變數刪掉即可解決。

一張含有 文字, 螢幕擷取畫面, 軟體, 陳列 的圖片

自動產生的描述

**一張含有 文字, 字型, 螢幕擷取畫面, 設計 的圖片

自動產生的描述**

Total warnings & errors / Total Lines : 2/3151 = 0.06%

1. **Major Problems**

在本次作業中遇到最大的問題是Bridge的安排以及ABVIP的驗證，因為初始資料有些存在IM中，故有可能原先CPU欲同時讀取IM和DM的資料，但是都要和同個slave做access，因此需要對此情況做規劃，對master做優先順序的排列，後在做block住讓另外一個master執行，使通道不會被占用。再者是在設計時，即使功能正確，但為了符合ABVIP的要求，需持續做更改，將設計改成能支援真正的AXI，因為我的設計CPU和AXI的VALID和READY幾乎都會同時拉為1，不會有VALID為1但READY為0的情況，因此為了符合ABVIP的要求，需要將某些資料暫存並卡在某一個state，才不會使VALID時的資料有變動。

1. **Lessons Learned**
2. **深入理解AXI匯流排協議**：透過親手實作AXI匯流排的各個元件，我對ARM的AMBA 2.0規範和AXI協議的特性有了更深入的了解。
3. **AXI元件的設計與實現**：在設計仲裁器、解碼器和預設從屬模組時，我學會了如何處理匯流排仲裁、地址解碼和錯誤處理等關鍵問題。
4. **主從設備的包裝與整合**：透過設計CPU wrapper和SRAM wrapper，我體會到如何讓現有的CPU和記憶體模組與AXI匯流排相容，並解決了整合過程中的各種挑戰。
5. **系統級設計與整合**：將CPU、IM、DM和AXI匯流排結合成一個完整的迷你系統，讓我對嵌入式系統的架構和設計流程有了更全面的認識。
6. **硬體描述語言的運用**：在撰寫可合成的RTL程式碼時，我更加熟練地使用Verilog，並學會了如何遵守設計規範和時序約束。
7. **使用專業驗證工具**：透過使用JasperGold AXI ABVIP驗證我的設計，我學習了如何利用專業的驗證工具，並理解了assertion和property檢查的重要性。
8. **解決驗證問題與波形分析**：在驗證過程中遇到問題時，我培養了分析波形和除錯程式的能力，提升了問題解決的技巧。
9. **遵守設計規範與約束**：確保我的程式碼符合Superlint的要求，讓我更了解業界對於程式碼品質的高標準。
10. **合成與時序分析**：在進行設計合成和時序分析時，我理解了時序約束對於硬體設計的重要性，以及如何優化設計以滿足時序要求。
11. **報告撰寫與結果分析**：準備撰寫報告時，我意識到清楚記錄設計過程、驗證結果和解決方法的重要性，這有助於提升我的技術溝通能力。

總的來說，這次作業讓我在數位系統設計與驗證方面獲得了寶貴的實踐經驗，為未來更複雜的VLSI系統設計打下了堅實的基礎。