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: **\_\_\_周昱佑\_\_\_** \_\_\_\_杜冠勳\_\_\_\_

Student ID: **N26090180** \_ **N26094883**\_\_\_

* **Summary :**

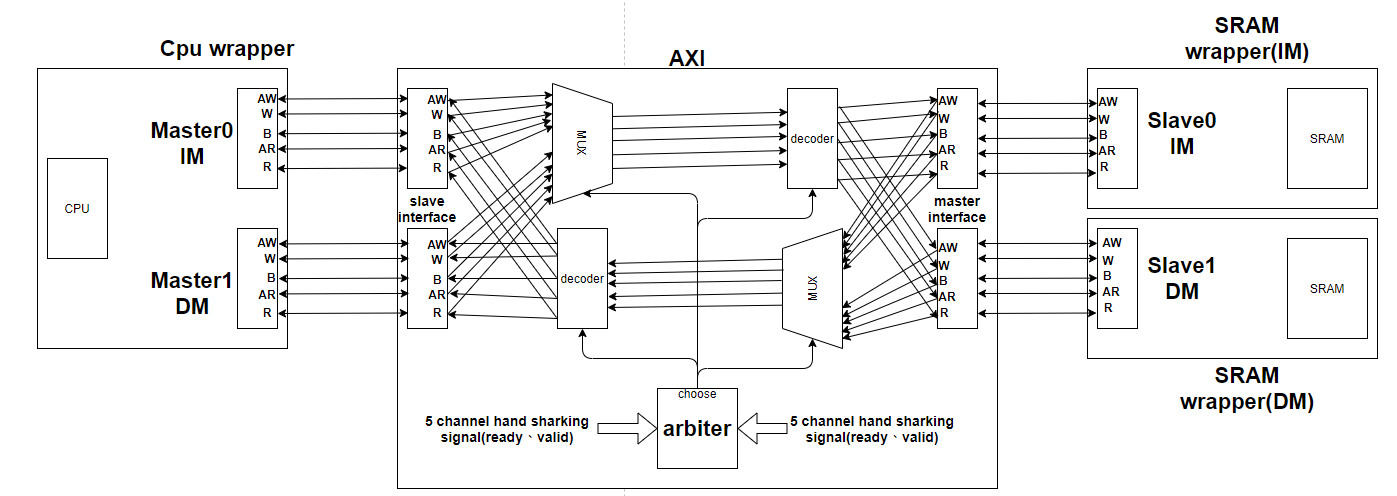
|  |  |  |
| --- | --- | --- |
| 1. | AXI.sv verification(jaspergold) | **DONE** |
| 2. | CPU\_wrapper.sv verification(jaspergold) | **DONE** |
| 3. | SRAM\_wrapper.sv verification(jaspergold) | **DONE** |
| 3.1 | IM=slave1(ID=0) | **DONE** |
| 3.2 | DM=slave2(ID=1) | **DONE** |
| 4. | Outstanding | 1 (In order) |
| 5. | Burst operation | Single transfer only |
| 6. | Transfer type: | No burst transfer |
| 7. | Synthesis | **DONE** |
| 7.1 | Prog0 before and after Synthesis | **PASS** |
| 7.2 | Prog1 before and after Synthesis | **PASS** |
| 7.3 | Prog2 before and after Synthesis | **PASS** |
| 7.4 | Prog3 before and after Synthesis | **PASS** |
| 8. | superlint | **99.24%** |

* **分工 :**

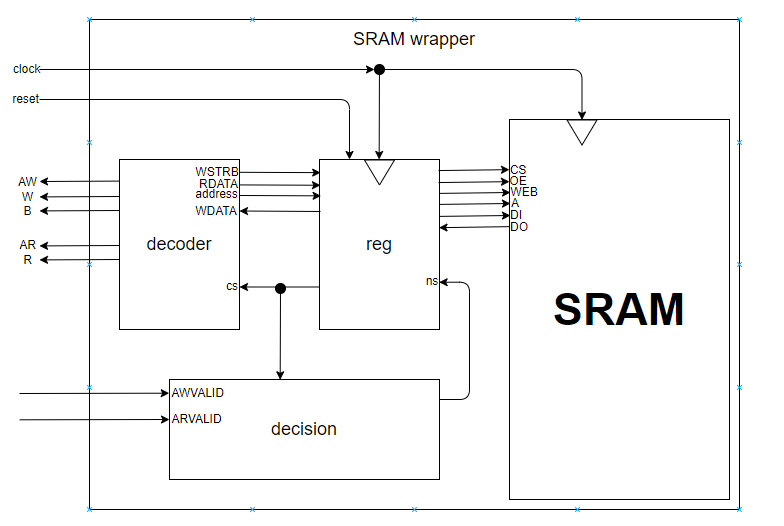
|  |  |  |
| --- | --- | --- |
| **姓名** | **周昱佑 (50%)** | **杜冠勳 (50%)** |
| **負責工作** | **AXI**  **SRAM Wrapper** | **AXI**  **CPU Wrapper** |

* **Architecture :**

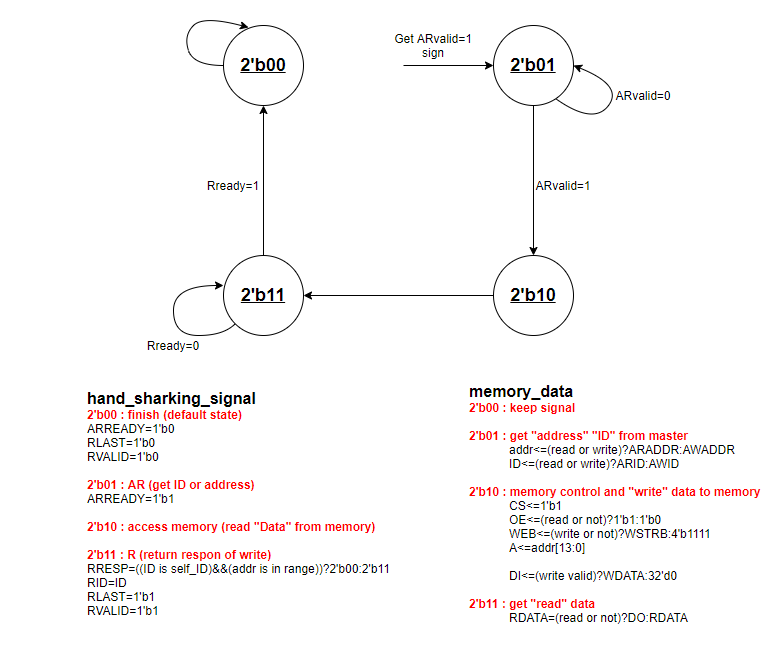
**此次AXI的設計架構以狀態機來做訊號線的控制與讀存資料，以這樣設計的原因為較易於做判定是否要維持訊號與cpu的stall，同時在debug能較清楚的觀察到是哪個狀態的判定上出了問題，狀態的切換依照spec上描述，以valid與ready為主，在slave端會驗證ID與ADDRESS的正確性給予RESP為OK或DECERR，中間AXI的部分以arbiter來決定優先執行的部分，設計上以master\_dm優先於master\_im，write優先於read。**

****

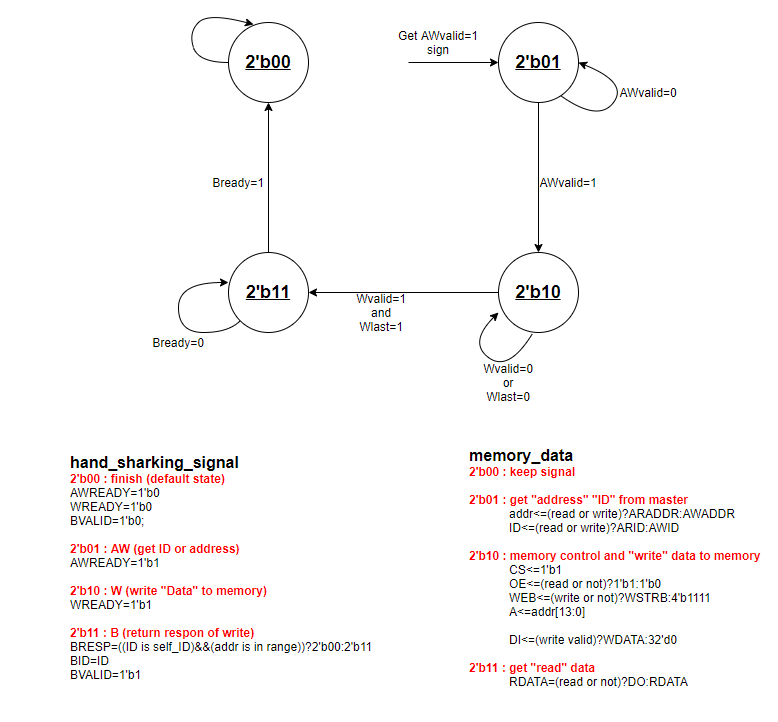
* Slave :



1. Read FSM

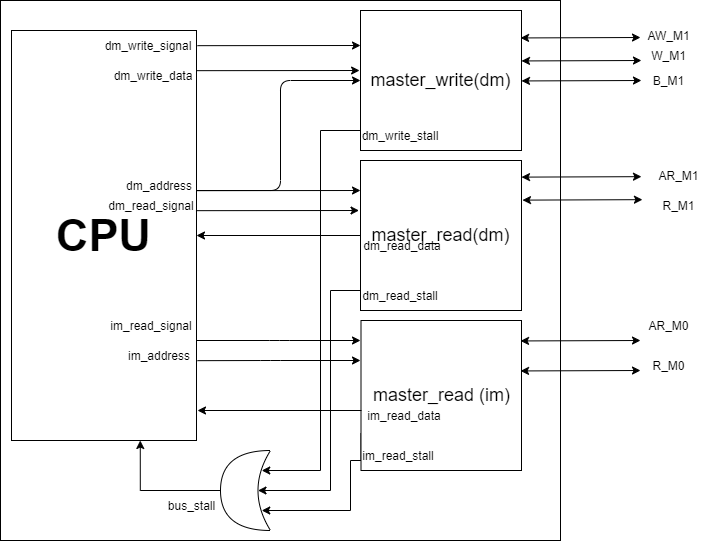


1. Write FSM

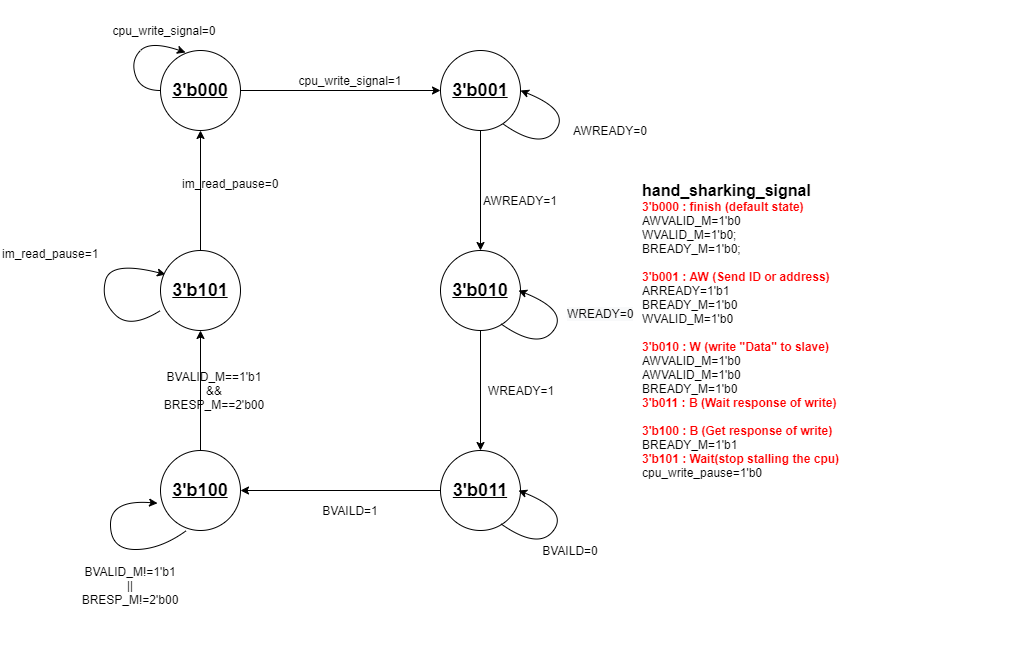


在設計上，有hand sharking 訊號才會進入下一個state，抓取master端送過來的data則也是以達到ready與valid同時拉為1時才抓取，並存入reg中保持穩定，state 01為AW或AR抓取data，state 10為W或R抓取master送過來要寫入memory的data(LW、LB)與訪問memory取得正確address上的data(SW、SB)，當為read時，則會在state 11時把 state 10取得的data送出並拉起RREADY訊號，並回傳RRESP，在write時，state 10則會寫入data，並直接進入state 11傳送BVALID訊號與BRESP。

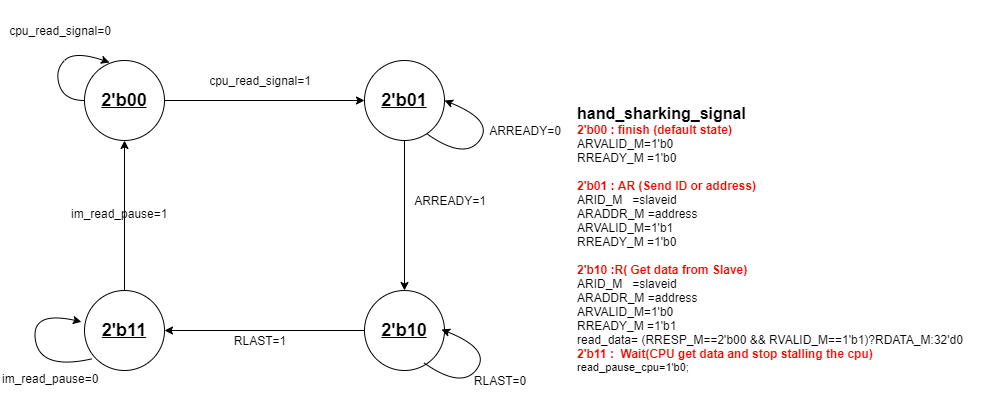
* Master:



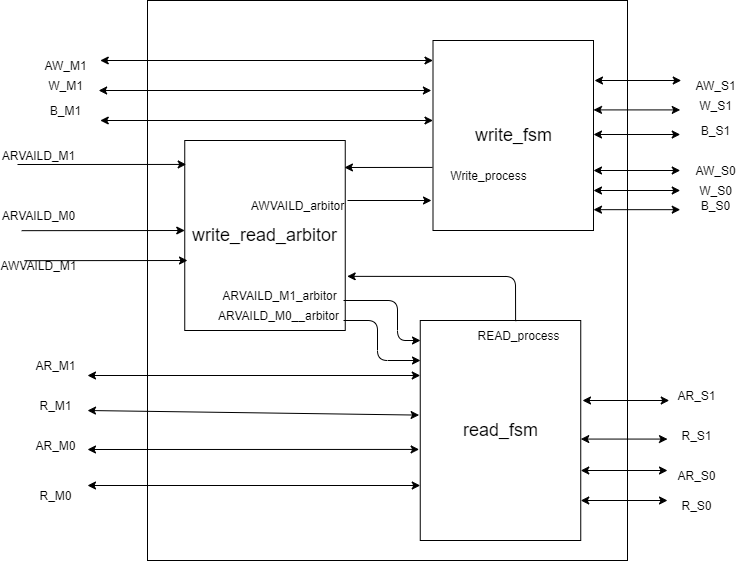
1. Master\_write



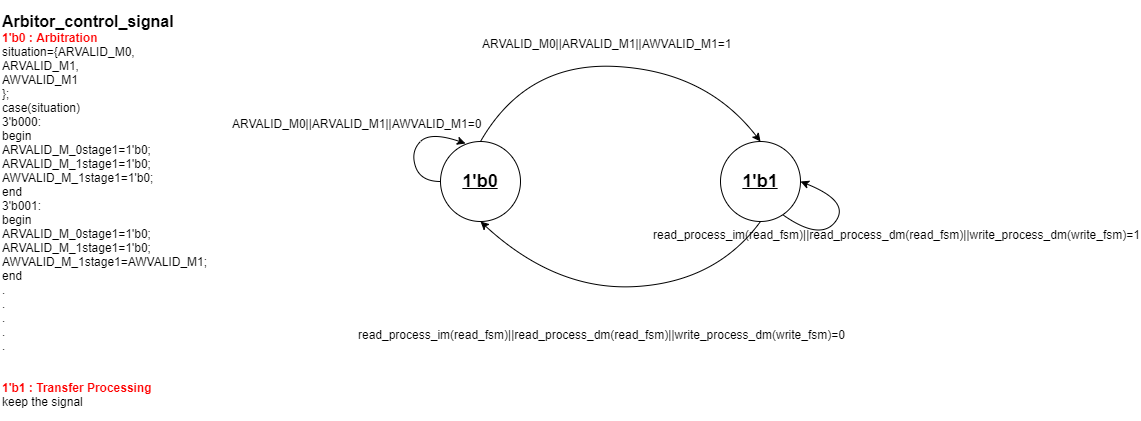
1. Master\_read



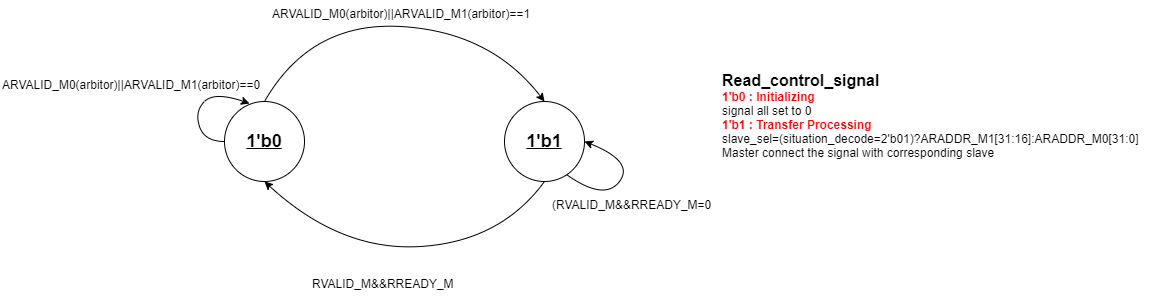
* AXI:

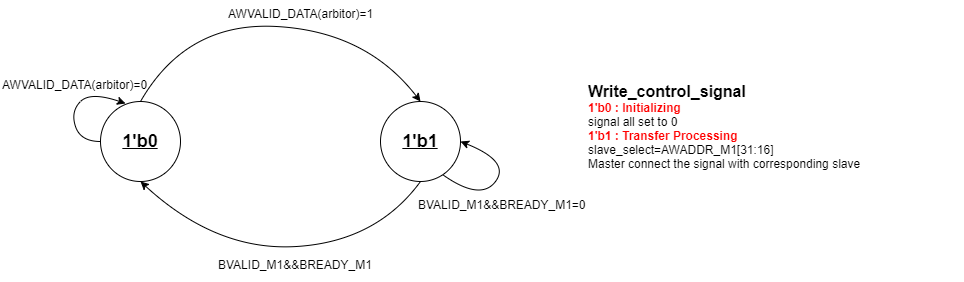


1. write\_read\_aribitor



2. read\_fsm



1. write\_fsm

* **Major problem & resolutions :**
* Slave :

1. RVALID、ARREADY、AWREADY等訊號線為unstable?

<sol.>等待master端送出相對應的hand sharking訊號才進入下一個狀態。

1. RID、RDATA、BID等資料為unstable?

<sol.>在轉換與接收data的state時才做改變並將資料栓鎖進reg中，其他時候則輸出栓鎖的數值。

1. 輸入訊號線出現亂跳的現象，port與線的數值不相同?

<sol.>檢查top.sv的接線，發現有誤，下次接完後要再細心的檢查。

1. 當訊號線讀跟寫同時送入slave端?

<sol.>多設定一個flag訊號來做優先度的判定，以write為優先再來才是read，達到防止狀態機亂跳狀態而送錯訊號。

1. Read data時，資料沒有在cpu stall時回傳，反而多延後一個stall區間才收到?

<sol.>當memory在讀出資料時，address送入為1個clock，送出資料(RDATA)為1個clock，所以在狀態機state10時，多停留1個clock讓資料被讀出並在state11時將資料送出。

1. 收到的address在某些時候讀錯data?

<sol.>由於我們原先處理address為cpu做完除以4後才傳出給slave，但是在做prog0時發現master\_dm會去讀slave\_im的數值，造成我們事先處理的address會使AXI的abiter開錯通道，使資料讀錯並回傳，再改成由slave端統一處理時已經解決。

* + AXI :

1. Output訊號unstable(aw、arvalid與aw、arready)?

<sol.>由於在沒有選擇到要開啟通道的slave與master，jaspergold仍然會給予訊號，所以需要將其輸出的訊號做栓鎖，若有出現hand sharking的時候才能改變其訊號。

1. valid與ready訊號的先後順序?

<sol.>測試 jaspergold時，valid與ready訊號並沒有指定先後，若valid訊號先送出，則要保持與其一同送出的資料，所以加上一個flip-flop與狀態機來做資料的栓鎖。

1. AR、R與AW、W的先後順序問題?

<sol.>由於AW和W可以同時送出，但是AR與R有先後送出的問題，由於我們在設計通道上，但是jaspergold會提前拉起B與R的hand sharking訊號，如此，則會出現通道過早關閉，所以將read與write的hand sharking訊號放入arbiter內來做通道的開關判定。

1. Decoder條件沒有寫滿?

<sol.>造成latch產生與訊號線的值可能亂跳，在問題1.有提到的栓鎖問題有解決但是仍要把沒用到的線補上，防止產生latch。

* **Results of verification :**
* Max pending number :  **1**
* **Jaspergold for AXI 4.0 protocol :**

1. **Master wrapper :**

一張含有 文字 的圖片

自動產生的描述

1. **AXI :**

**一張含有 文字 的圖片

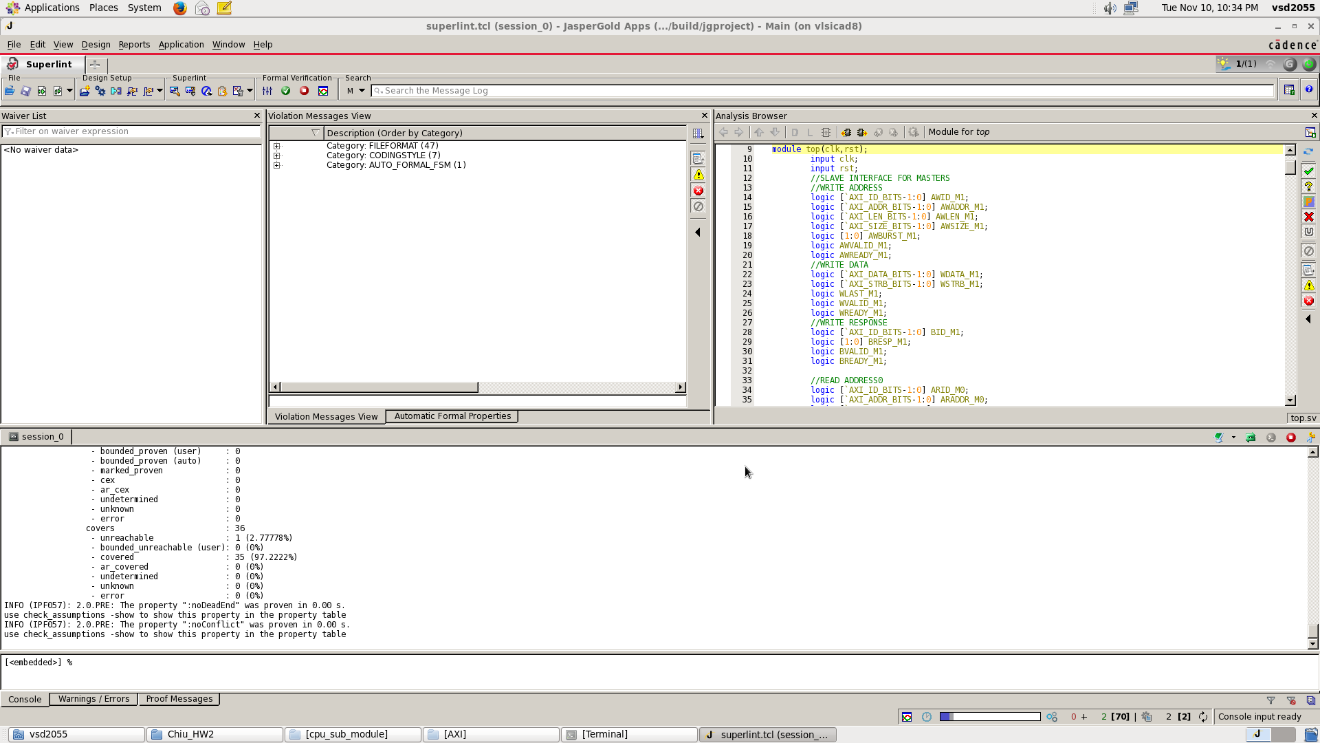
自動產生的描述**

1. **SRAM wrapper :**

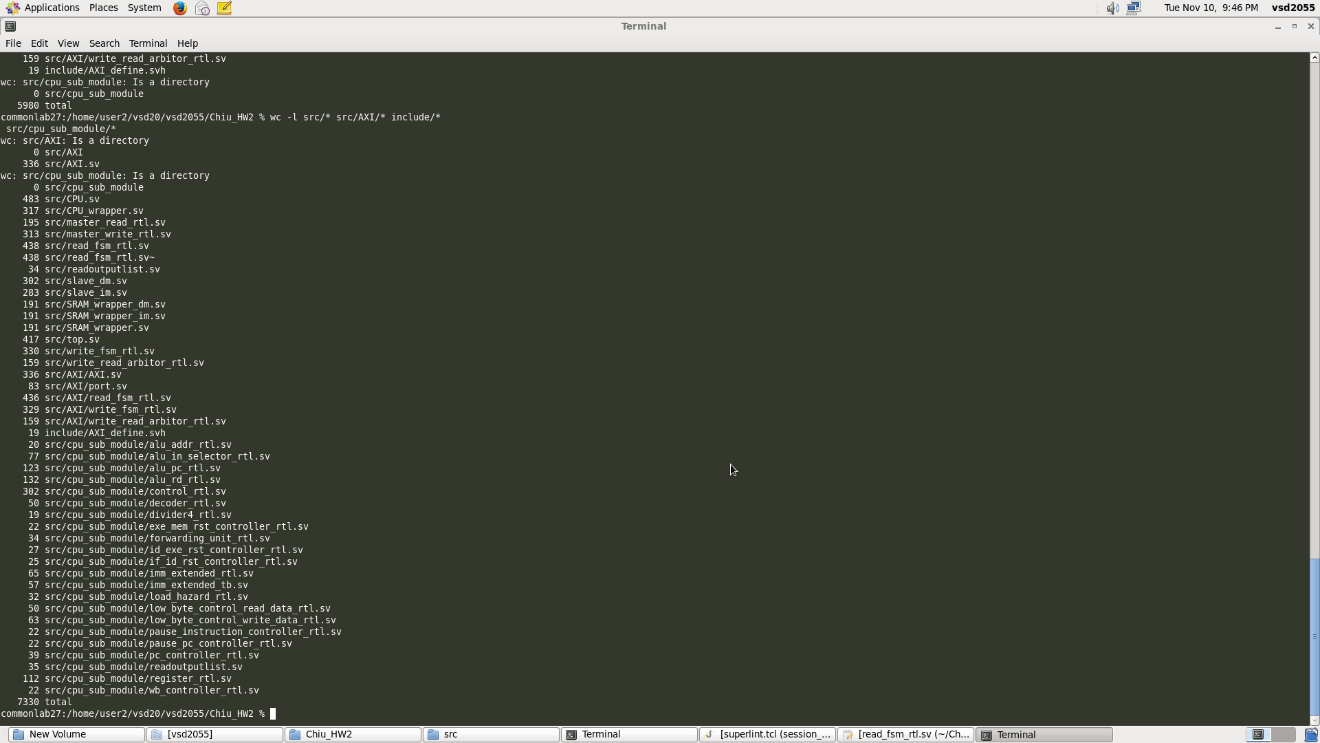
一張含有 文字 的圖片

自動產生的描述

* **Superlint :**



Total line of warning : 55



Total line of code : 7330

Total line of code : 7330

Total line of warning : 55 (47+7+1)

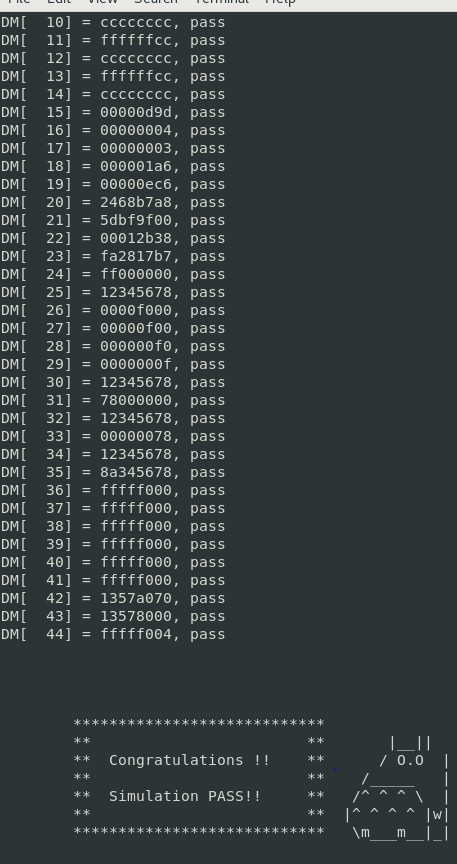
※Correct rate of code : (7330-55)/7330=0.992496 (99.24%)

修正錯誤與warning:

1. Master 端的送出hand sharking訊號有latch產生，有53條warning，補齊訊號線在always\_comb後，已被修正。
2. 上圖所示的47條warning，為file formate上的問題，檔案的命名上為該warning出現的原因，由於檔案命名不影響，故按照其設計命名沒有做修正。

* **Test for Prog0~3**

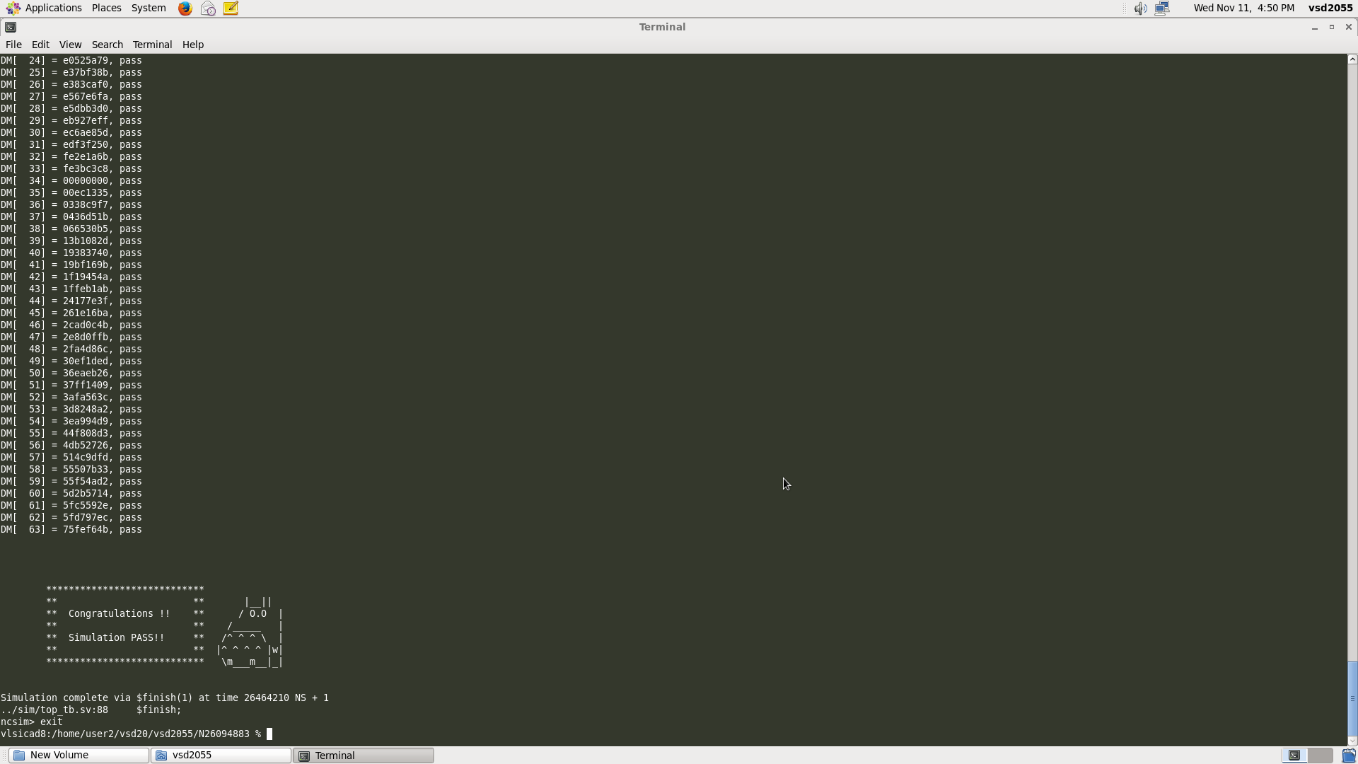
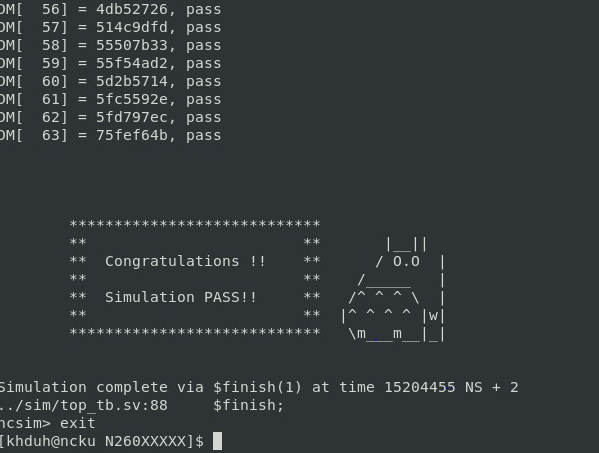
1. **Prog0**



make syn0

make rtl0

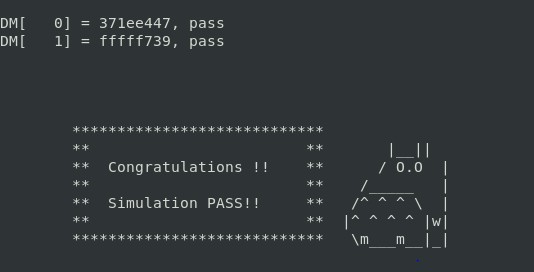
1. **Prog1**



make syn1

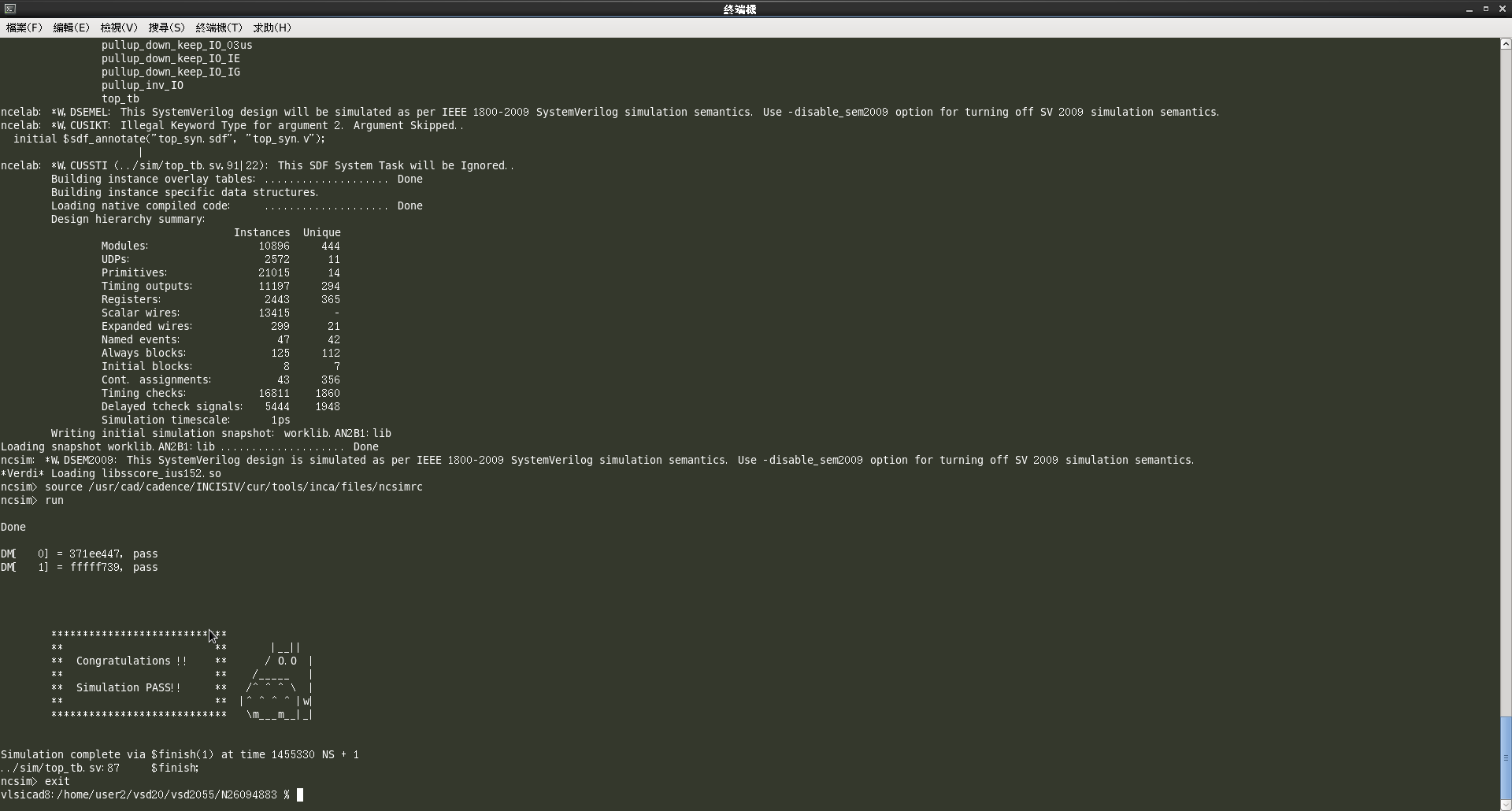
make rtl1

1. **Prog2**

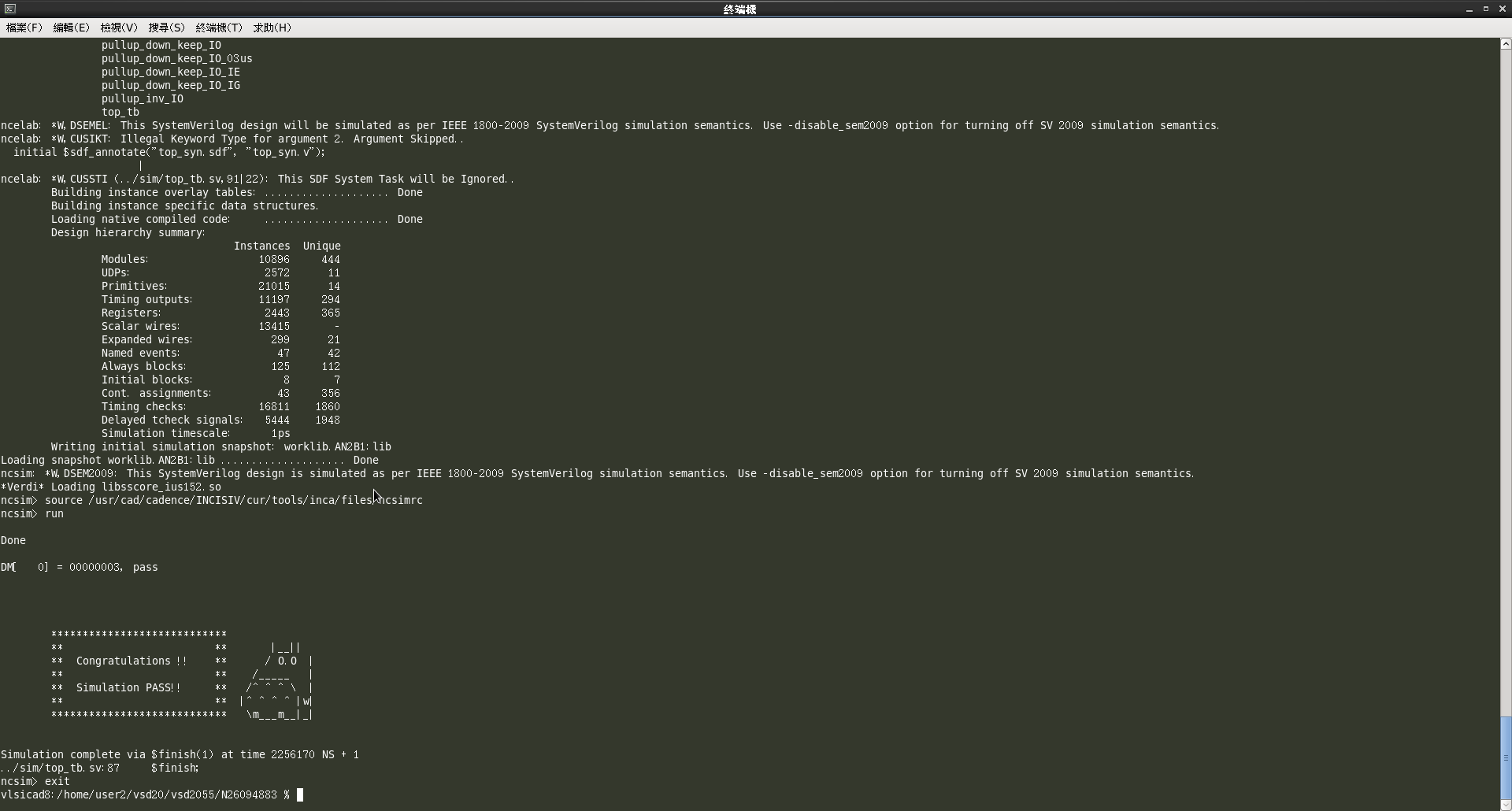
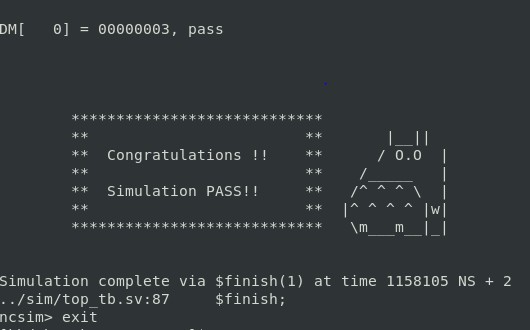


make syn2

make rtl2



1. **Prog3**

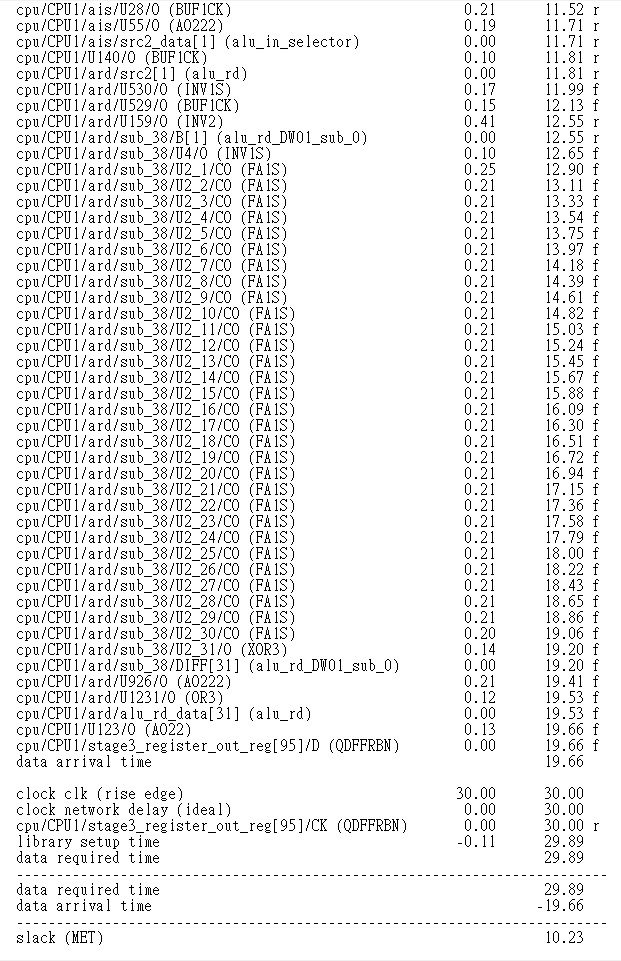
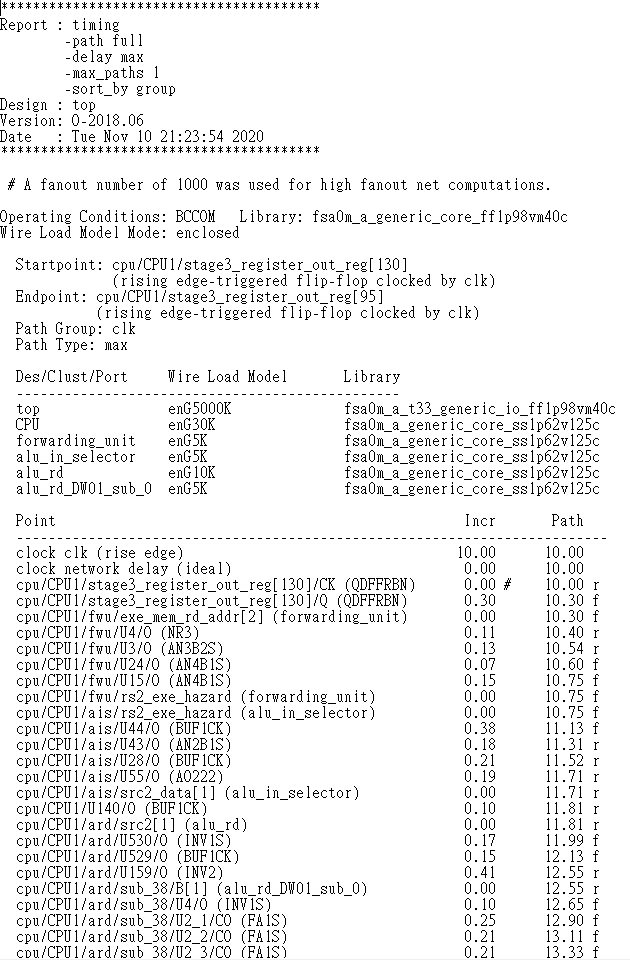


make rtl3

make syn3

* **Synthesis :**

合成上我們使用的clock cycle為20ns，結果如下附圖



**※於報告的附檔中有截錄time、power、area的txt檔報告**

在執行合成時會看到一些warning或error的出現，代表有些無法合成的部份或多餘的code，例如 : 原先在master wrapper內always\_ff(posedge clk)begin end，這種原先在寫code時沒有刪除掉的code，進行修正。

* **Lesson learned :**
* 周昱佑 :

在這次的作業中對於slave與master之間的溝通介面有了更多的了解，這次我們所學的為AXI4.0的協定，在這之中從hand sharking的了解是最主要的設計要點，當valid與ready同時為1時才能將資料當作正確的讀入或送出，所以我以這兩種訊號作為狀態機進下一個state的條件，這個條件設計原本我並沒有考慮到，是因為在測試jaspergold時，發現到這兩個訊號如果其中一者沒有拉為1，則另一者不能放下原本拉起的訊號，由於原本由狀態機的設計是依照valid來在給一個clock time的ready，所以在這裡修正進下一個state的狀態，並依照valid來判定資料抓取的確認訊號，並用address與AW、ARID來判定訪問正確的slave，由於我負責設計slave端的部分，同時考慮到要留一個clock time給memory作讀寫的動作，所以考慮到之後合成不能有multi-edge clock所以給一個stage作memory訪問用，另外一個修正點在AW、ARvalid同時送入時要先執行哪一個，所以多用了一個變數flag來作判斷，總結來說最後我將送至slave的訊號分為hand sharking(控制、確認)與data(要以register栓鎖)，由於一開始對這個協定不是很了解，所以花了不少時間跟實驗室的同學一起研究了一下spec並討論中間AXI的設計，並了解到其中並不能用純組合電路的方式來做設計，需要考慮到master訪問的優先度、valid與ready訊號先後問題、未被選擇的interface訊號保持的問題等，基本上都要加上register來作資料得儲存，在AXI最後的設計上才知道AW與W是有可能同時送出資料的，所以在arbiter上判斷條件又作了點修正。

在做prog0的debug時也發現很多沒注意到的，例如:address，若是在cpu優先處理好除以4再送出，會使AXI有判定通道開啟上的錯誤，進而讀錯資料回傳給master，另外在master端送資料過來時，我的slave能否在我所設計的state栓鎖住data並且保持穩定，同時在default時也保持住直到下一次訪問才能更新栓鎖的data，這些設計上的小缺漏在整合之後才會看到的。

同時也感謝助教的協助解答了我不少關於AXI protocol的問題，讓我對這次陌生的作業學習到不少。

* 杜冠勳:

此次作業學習到到了 AMBA AXI 的溝通架構 是以handshake 的基礎上完成的 而這次作業中我設計了AXI 架構 以及CPUWrapper架構

在設計AXI 架構時 ,使用了三個狀態機互相送出訊號去做溝通,三個狀態機分別為arbitor ,讀取 ,以及寫入 ,arbiter 收到來自master的訊號時,會仲裁出通道使用權,而沒被選中的通道則是被控制在讀0, 讀取 ,以及寫入狀態機 接收來自arbitor狀態機的訊號,線決定slave端口是否接上。

在設計AXI架構時因為過去不好的習慣導致在使用狀態機時產生了latch以及一些recursive的電路 經過這次的練習修正了許多觀念 也成功設計出來 。

在設計CPUWrapper架構時 因為要考慮到cpu stall住的問題 所以一開始就有將CPU 放入wrapper 中 設計 只是沒想到我把reset接為0 導致等於沒有接上,但當時候只測wrapper時 jg 是有全過的,如附圖,但當跑完4個prog 回去測jg 時發現不會過了,原因可能如下

我們的CPU設計與 Wrapper 溝通 除了read write 訊號 還有來自 wrapper 的 bus\_stall訊號線,而bus\_stall訊號線的設計是 當 read write訊號來時就會拉起,而dm 優先做的關係 導致在設計 讀寫dm的狀態機必須多一個state來做為判斷是否回狀態0(等候read write訊號狀態)的依據

因為如果回到狀態0由於IM在讀取指令中 造成 pipeline register 會 stall 住 而造成那段時間的dm\_read 或者是dm\_write為1 會讓 dm端口 一直送出request 造成無窮迴圈 故在這次的設計中 兩個master 端口有互傳控制訊號的情形出現導致某一些state無法被測到(uncover )像是dm 端口先拉ARVaild AWVaild的情形 就不會有

* + 原因一 CPU 會一直讀取指令 故im\_read 當初設計為恆拉為一
  + 原因2 儘管後來有改成在load\_hazard避免再向 IM讀取指令 但是考量到設計的架構,我們將 bus stall 統一在wrapper做處理, wrapper 僅會送一個控制訊號給cpu使用不用多作處理 才是符合wrapper設計 這導致 我們不用在cpu去協調 送出 IM DM Request 的優先順序

另一種寫法則是使用控制cpu 讀寫的順序進wrapper ,但控制流程較為複雜,而wrapper 還是要送訊號回來給cpu只是狀態機不用多一個state 判斷 優先順序,而是CPU端口處理,由於我們的設計是選前者,故會讓CPU 只會有IM DM 端口同時送出訊號的情形出現,或是單獨IM 送出請求訊號的情形,造成jg有uncover。

在與 CPU 整合上 須注意 bus\_stall 情形必須優先於 其他影響 pipeline register 的情形出現 像是 load word 或是 jump 的情況

另外 write\_data 回register 時為了確保資料安全性,亦可在stall 後一刻再拉起訊號 下一個posedge clk 即可寫回。