VLSI System Design (Graduate Level)

Fall 2021

HOMEWORK IV

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 | 劉彥麟 | 李秉軒 |
| Student ID | N26100723 | N26100595 |

Catalog

[Summary 3](#_Toc91598627)

[Contribution 3](#_Toc91598628)

[Design Explanation 4](#_Toc91598629)

[CSR 4](#_Toc91598630)

[Interrupt 5](#_Toc91598631)

[Verification Result 7](#_Toc91598632)

[Waveform 7](#_Toc91598633)

[Simulation Result 12](#_Toc91598634)

[Pre-Synthesize(RTL) 12](#_Toc91598635)

[Post-Synthesize(SYN) 12](#_Toc91598636)

[Post-Place&Route(PR) 13](#_Toc91598637)

[Automatic Placement & Route (APR) 14](#_Toc91598638)

[Program 19](#_Toc91598639)

[Boot 19](#_Toc91598640)

[Program2 19](#_Toc91598641)

[Superlint 19](#_Toc91598642)

[Performance & Area 21](#_Toc91598643)

[Lessons learned 25](#_Toc91598644)

# Summary

本次作業中我們完成了Control Status Register以及Interrupt機制的實作，使得CPU能夠在執行的過程中跳轉至Interrupt Service Routine，並在完成後跳回到原先的程式繼續執行，並且在系統中加入Sensor Wrapper以及Sensor Controller進行Interrupt的相關測試，因應Sensor的新增AXI Bus也進行相對應的修改。

最終我們成功讓系統在Clock Cycle為14.2ns的Constrain之下Synthesize的Worst Negative Slack為0，且之後的Automatic Place & Route也沒有任何Violation，Pre-Synthesize、Post-Synthesize、Post-Place&Route的Simulation皆通過。

# Contribution

劉彥麟(N26100723)：50%

AXI Bus, Sensor controller, Sensor wrapper, Synthesis, APR.

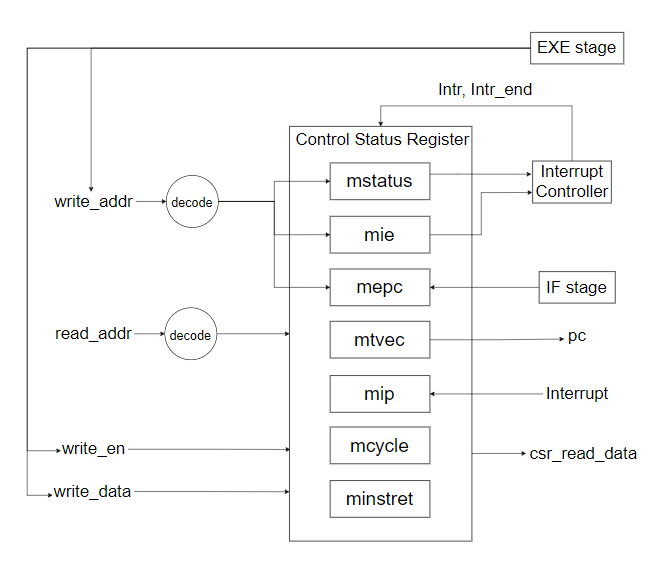
李秉軒(N26100595)：50%

RISC-V CPU, CSR&CSR instructions, Interrupt mechanism, L1 Data Cache, Synthesis.

# Design Explanation

## CSR

由於CSR和Register File相似，因此將我們將CSR實作在ID stage，下圖為CSR之架構圖：



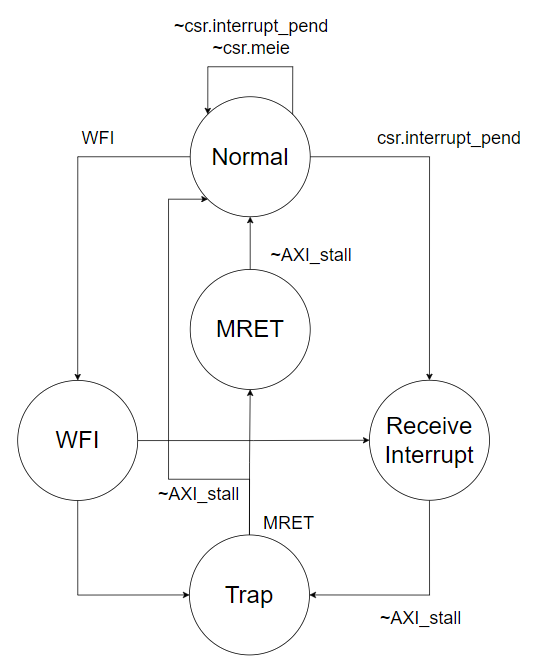
圖(一)、 CSR架構圖

在圖中的左側，會同時有read和write address傳入，其中read address是instruction decode得出，而write address是由EXE stage傳入，原因是寫入CSR的資料不會需要來自Memory，所以可以在EXE stage運算完後就寫回。Address decode後的箭頭指向，代表所有register都是可讀的，但只有mstatus, mie, mepc可被寫入，其餘只能由CSR根據Interrupt自己更新。

發生Interrupt時，Interrupt訊號不會直接接到CSR，而是由Interrupt Controller發送進入/離開ISR的訊號，讓CSR和PC能夠準確發送/接收彼此需要的資訊，此外，Controller需要檢查CSR的資訊，才能決定是否進入ISR。

## Interrupt

由於我們的CPU在實作時，有考慮到在執行wfi指令前就收到interrupt的情況，使得控制較為複雜。因此我們獨立設計一個Interrupt controller(以下簡寫為controller)負責控制整個CPU的interrupt行為(主要是program counter及CSR)，下圖為controller的Finite State Machine。



圖(二)、Interrupt controller FSM

以下簡單介紹各個state的行為：

**Normal** : 在此階段，controller會檢查是否有發生interrupt(透過CSR的mip register)，以及是否decode到wfi指令。當decode到wfi指令，就會進入WFI state，若是檢查到有interrupt，但pc處在無法被更新的狀態，就會進入Receive Interrupt state。

**WFI** :在此階段，PC會固定在wfi指令的下一條指令，但IF傳至ID的register會塞入NOP，直到controller接收到interrupt訊號。

**Receive Interrupt** : 如果剛好在PC無法被更新的時候發生interrupt，controller會進入Receive Interrupt state等待PC。但在這次作業中，Sensor收取資料的速度較慢，導致一定是CPU已經進入WFI狀態才會收到interrupt，因此這部分功能較無法被驗證。

**Trap** : 當pc可以被更新，且controller檢查到interrupt pending，就會進入ISR，此時controller會向pc及CSR發送ctrl\_intr訊號，代表進入ISR。收到該訊號後，pc及CSR需要進行不同的運算。

**pc** : pc需要跳轉至ISR的位址，該位址由CSR提供，同時必須將目前的pc值傳給CSR的mepc register。

**CSR** : CSR需要提供ISR的位址(trap vector)給pc，並將目前的pc存入mepc，此外，必須更新mstatus中的MIE/MPIE/MPP bit。

接著就會開始進行ISR的運算，直到decode到mret(machine return)指令。

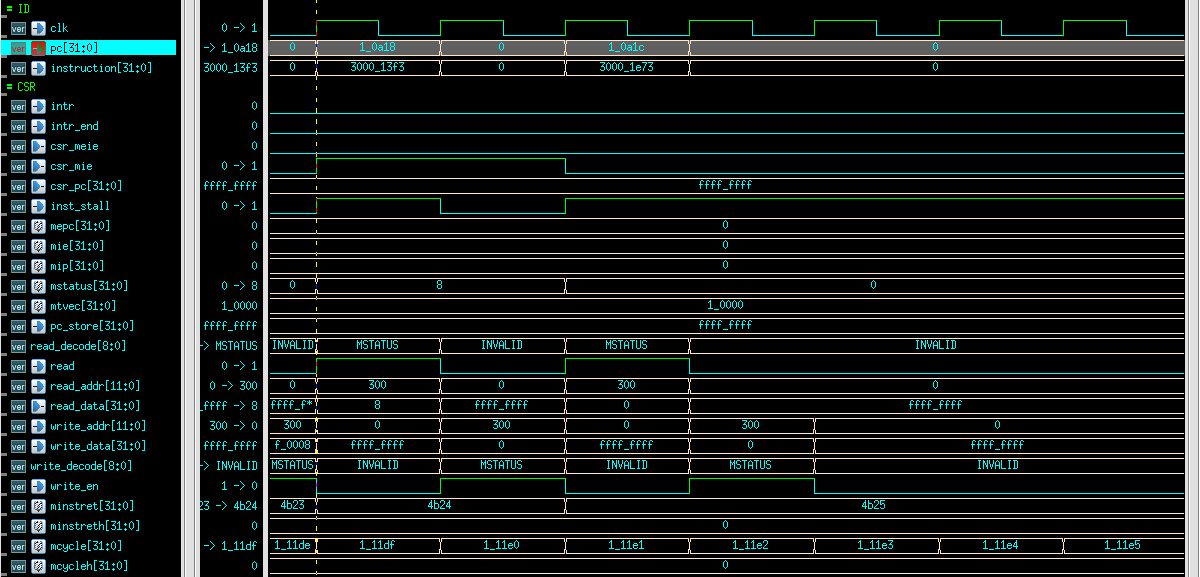
**MRET :** 當decode到mret指令，由於目前的架構，pc一定正在取得下一筆指令，因此controller也會等到pc可被更新，再同時向pc及CSR發送intr\_end訊號，代表離開ISR，state回復到Normal。pc會回到原先跳轉前的指令繼續執行，CSR則會將mstatus的值回復。

這時必須注意，IF stage會傳入mret的下一條指令，因此controller必須傳送一個訊號給ID到EXE的register，表示這條指令是mret後的指令，讓register塞入NOP。

# Verification Result

## Waveform

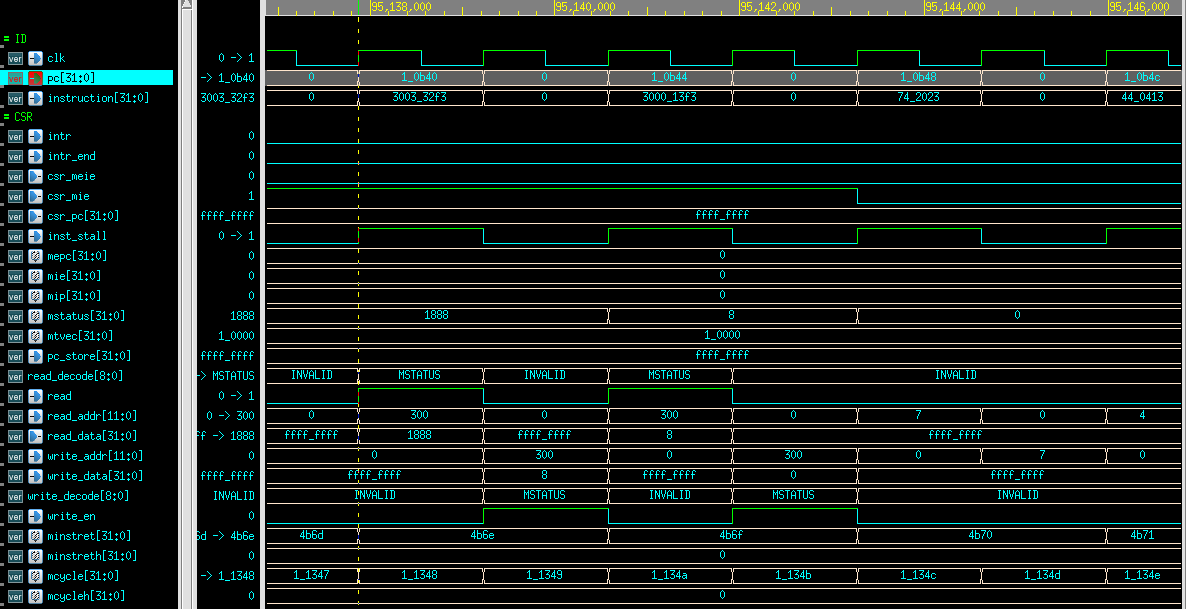
1. CSRRW



(圖) CSRRW指令(PC = 0x10a18)

上圖是一個CSRRW指令，read\_decode結果為讀取mstatus，讀出data為0x8。下一個cycle資料傳入EXE stage運算後，會再傳回CSR的write部分，因此看到write\_decode比read\_decode晚一個cycle，而寫入的資料為0。

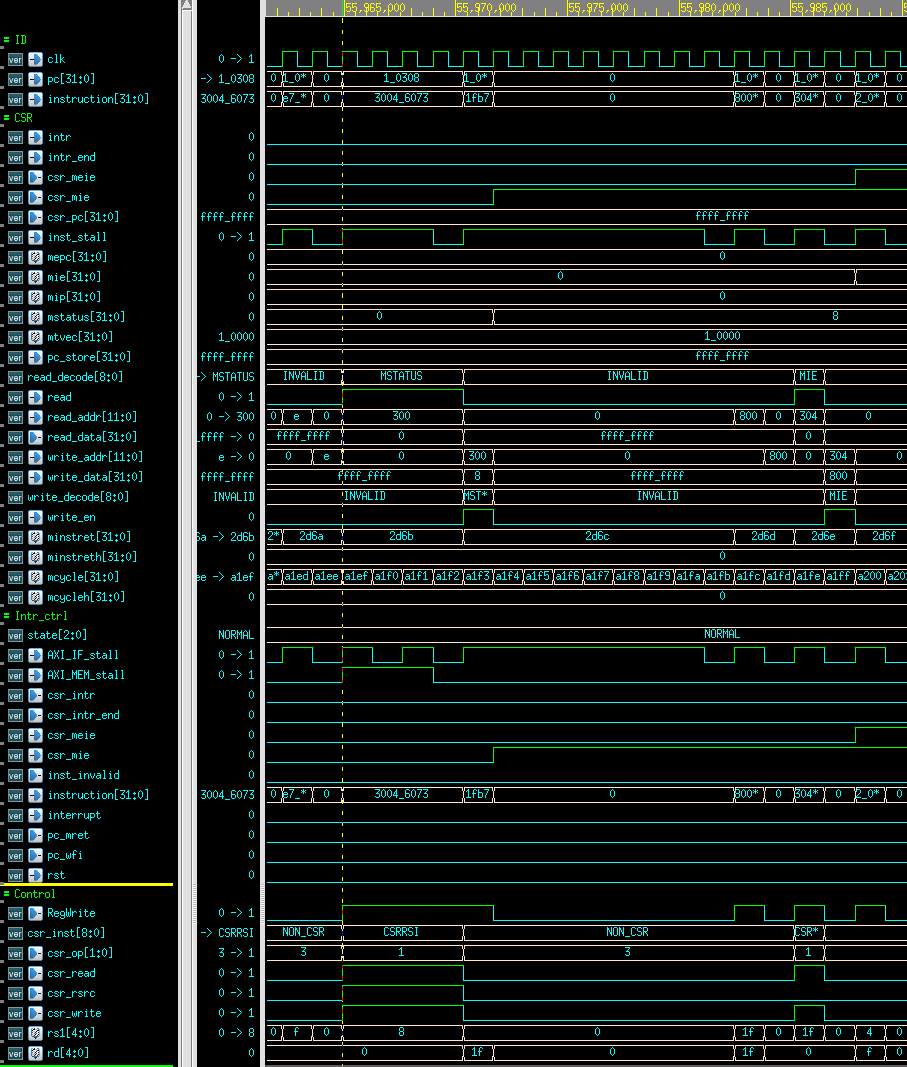
1. CSRRC



(圖) CSRRC指令(PC = 0x10b40)

上圖是一個CSRRC指令，和CSRRW不同之處在於，寫回CSR的值是CSR和RS1運算後的結果(CSR = CSR&(~RS1))，此處CSR讀出的值為0X1888，RS1讀出為0XF1880。CSR = 0X1888 & (~0XF1880) = 8，結果正確。

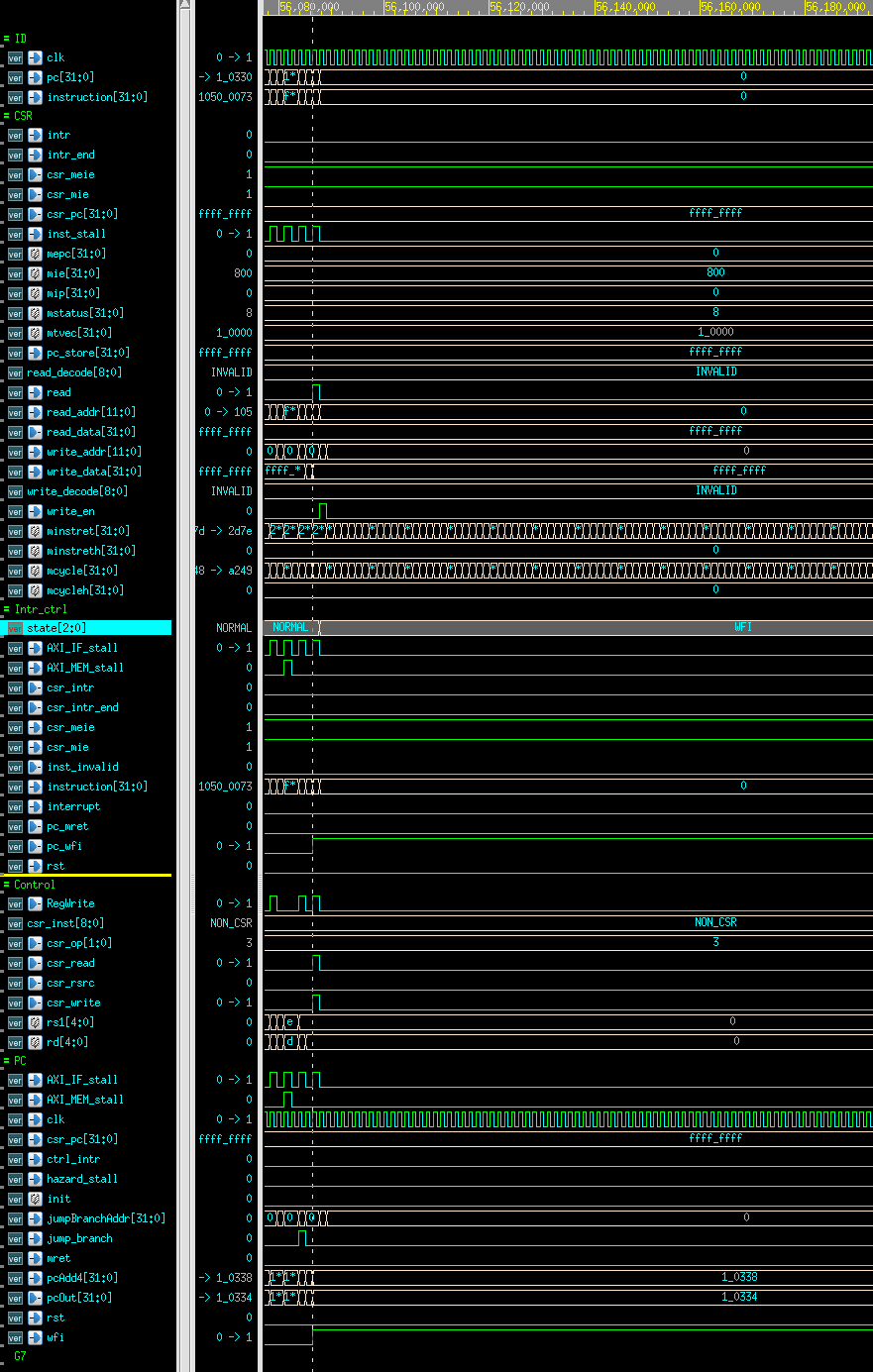
1. CSRRSI



(圖) CSRRSI指令(PC = 0x10308)

再來我們使用CSRRSI說明CSRRS和unsigned-immediate的運算，這張波型中列出了ID stage的Control unit對CSR運算的控制，以csr\_op代表運算種類(W/S/C)，csr\_rsrc代表是使用rs1還是uimm運算，在這個指令中，csr\_rsrc = 1代表使用uimm運算。驗證運算結果，CSR = CSR|uimm，此處CSR讀出為0，uimm為8，CSR = 0X0 | 0X8 = 8，結果正確。

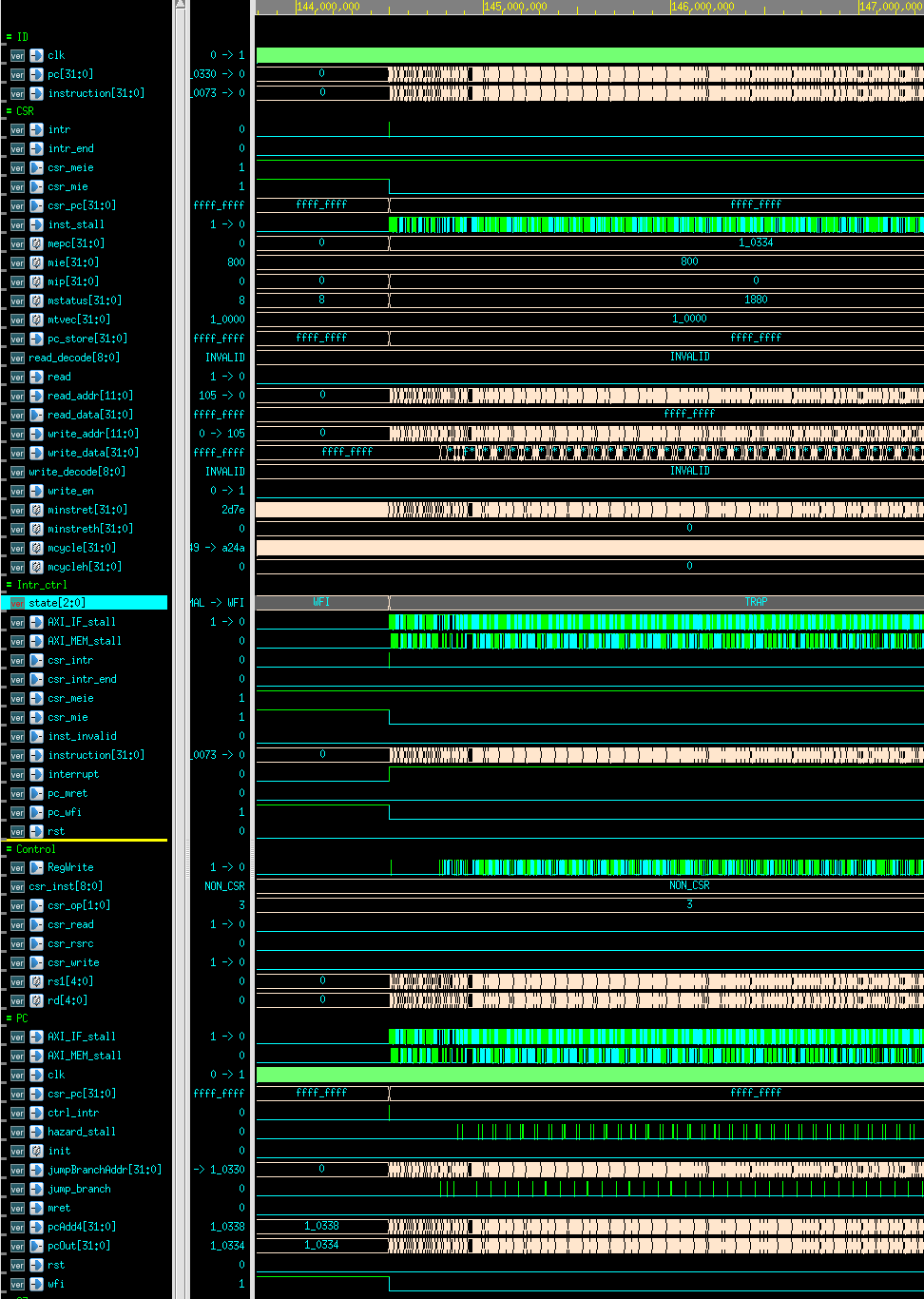
1. WFI



(圖) WFI

上圖所示的是decode到wfi時，各個module的波形圖。最上方ID stage會往後面的stage傳入NOP，而pc則會卡在wfi的下一個指令處等待(波型圖中的pc是ID到EXE stage的register，不是IF stage的pc)。

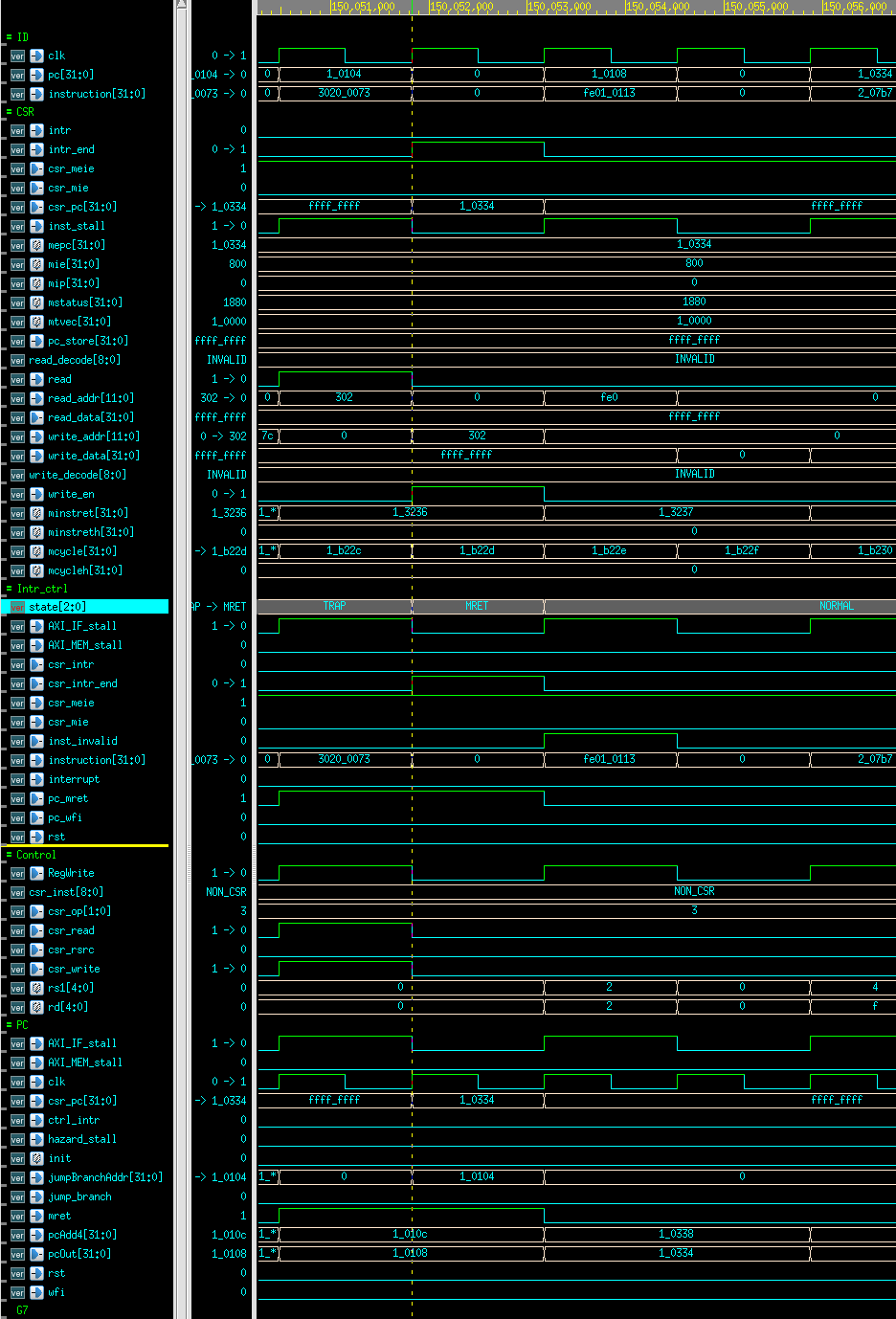
1. ISR



(圖) ISR

接收到interrupt訊號時，controller發送訊號通知pc & CSR進入ISR，pc會進入trap vector， CSR會在進入ISR時更新mstatus的值。

1. MRET

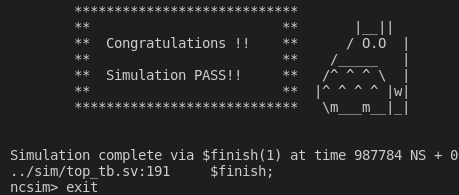


(圖) MRET

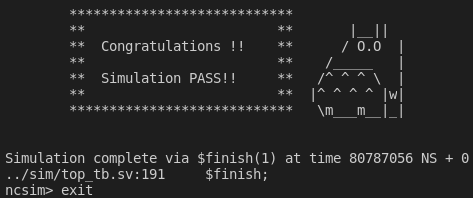
當controller decode到mret指令，再度發送訊號通知pc & CSR回到原本的程式。然而此時pc已經發送下一個指令的request，因此controller需要發送一個訊號(inst\_invalid)告知ID到EXE的register，這條指令無效，改為往後傳NOP。

## Simulation Result

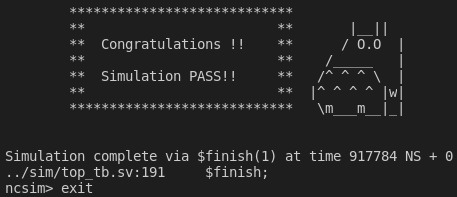
### Pre-Synthesize(RTL)



(圖) Program0

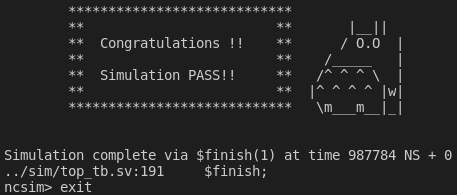


(圖) Program1

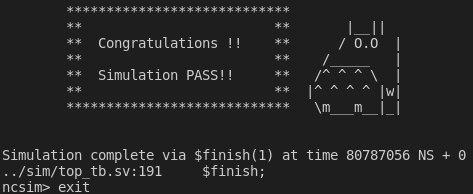


(圖) Program2

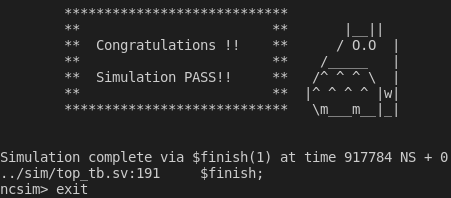
### Post-Synthesize(SYN)



(圖) Program0

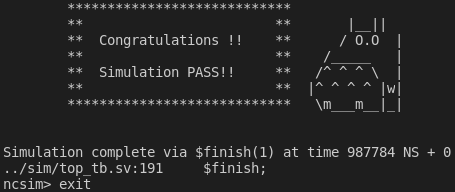


(圖) Program1

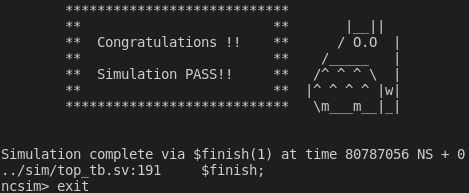


(圖) Program2

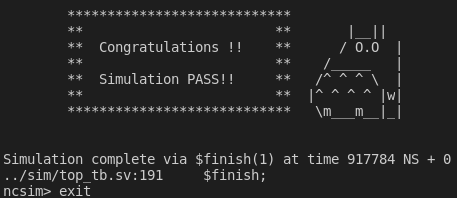
### Post-Place&Route(PR)



(圖) Program0

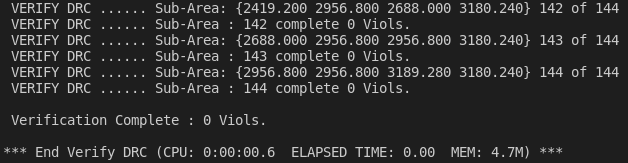


(圖) Program1

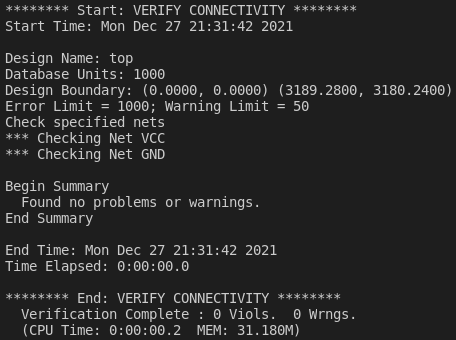


(圖) Program2

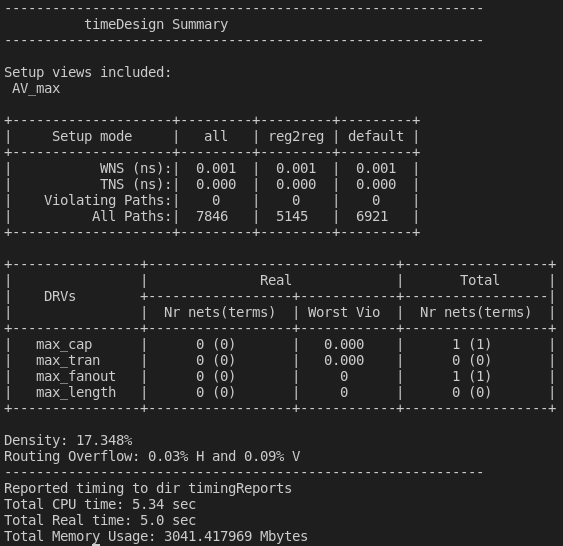
# Automatic Placement & Route (APR)



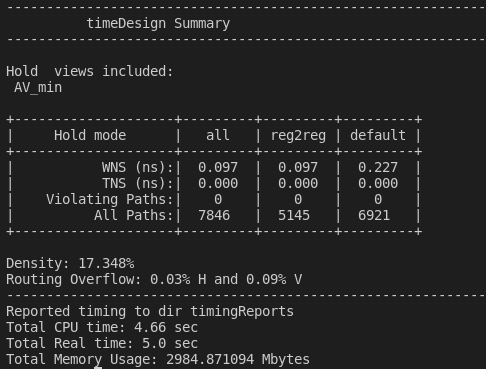
(圖) Powerplan DRC



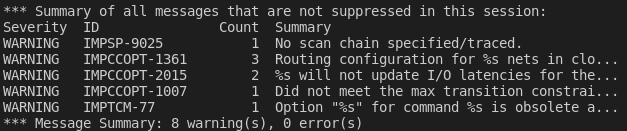
(圖) Powerplan Connectivity



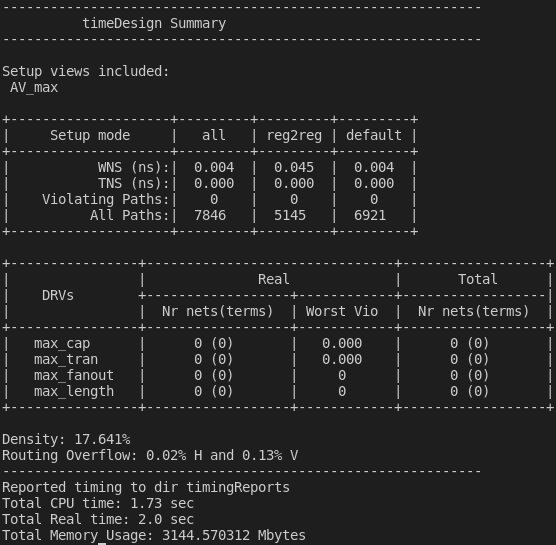
(圖) Pre-CTS Setup Timing Report



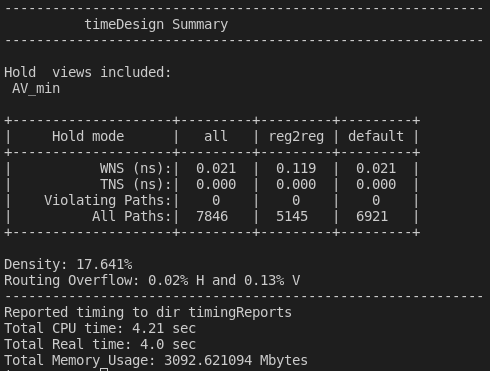
(圖) Pre-CTS Hold Timing Report



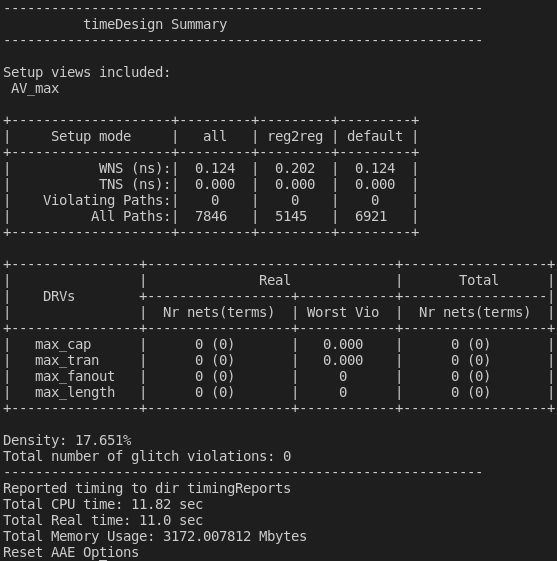
(圖) CCOpt no ERROR



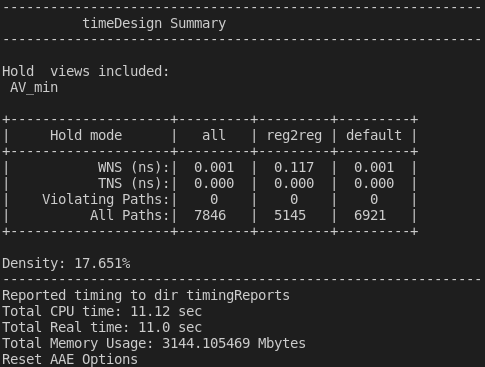
(圖) Post-CTS Setup Timing Report



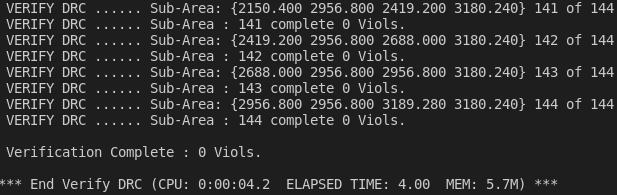
(圖) Post-CTS Hold Timing Report



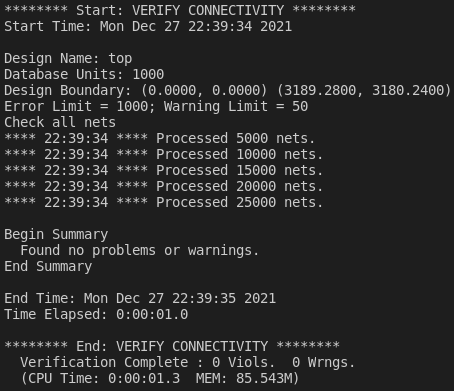
(圖) Post-Route Setup Timing Report



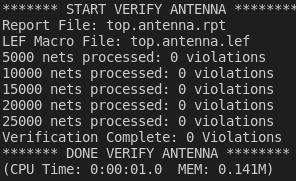
(圖) Post-Route Hold Timing Report



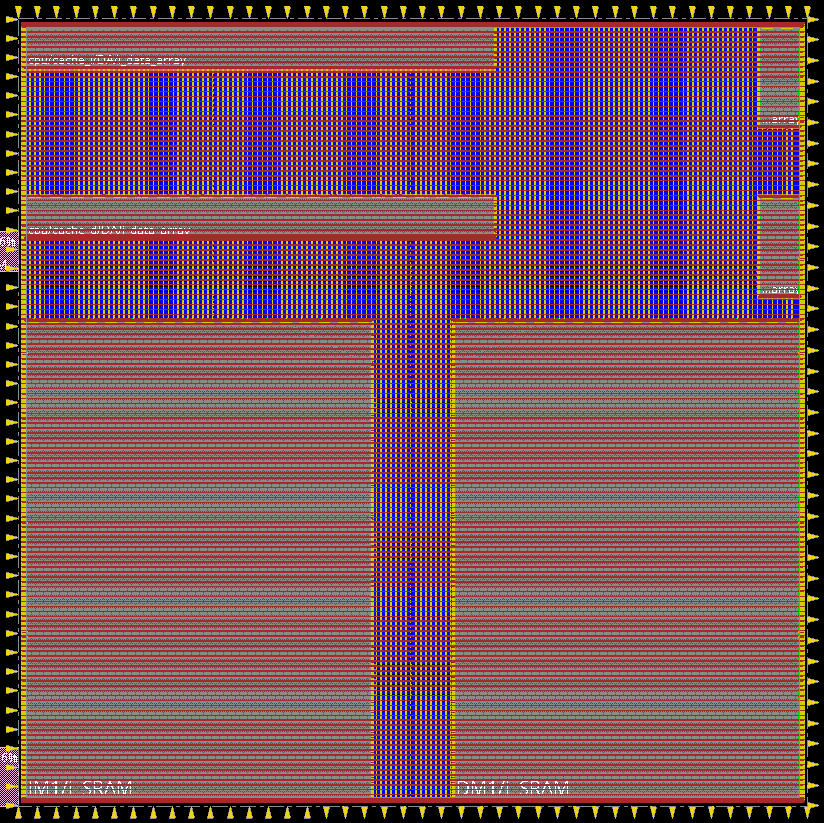
(圖) Routing DRC



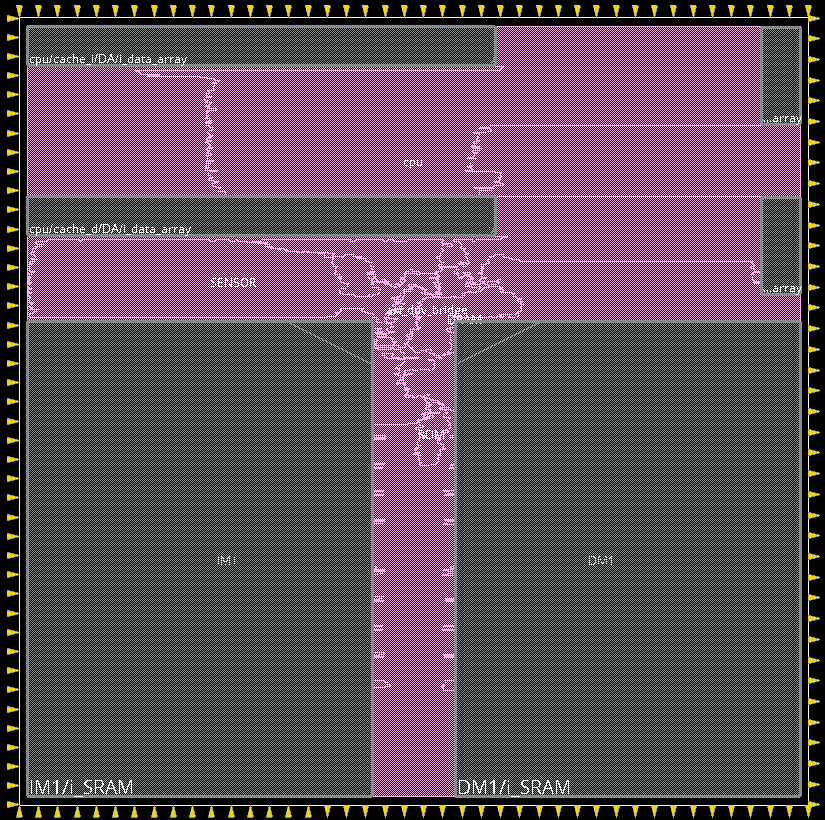
(圖) Routing Connectivity



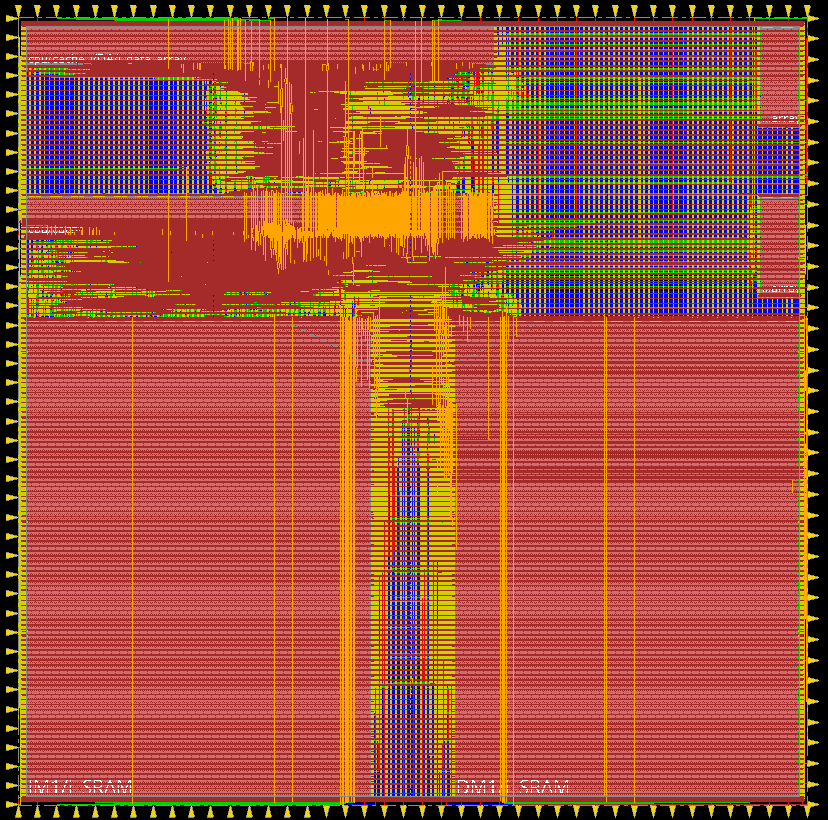
(圖) Routing Antenna



(圖) Floorplan View



(圖) Amoeba View



(圖) Physical View

# Program

## Boot

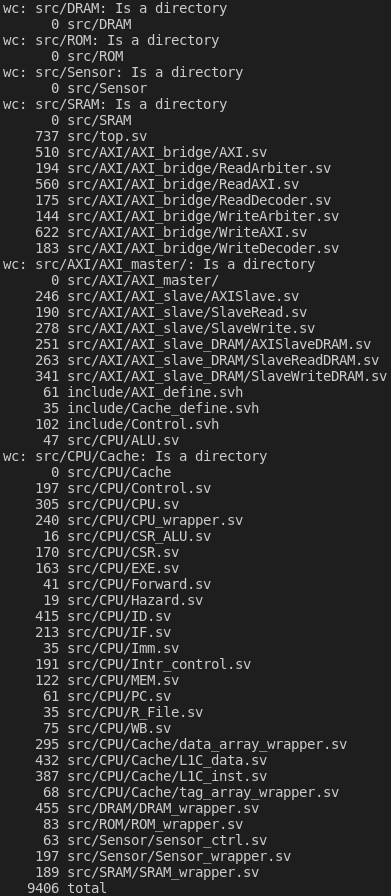
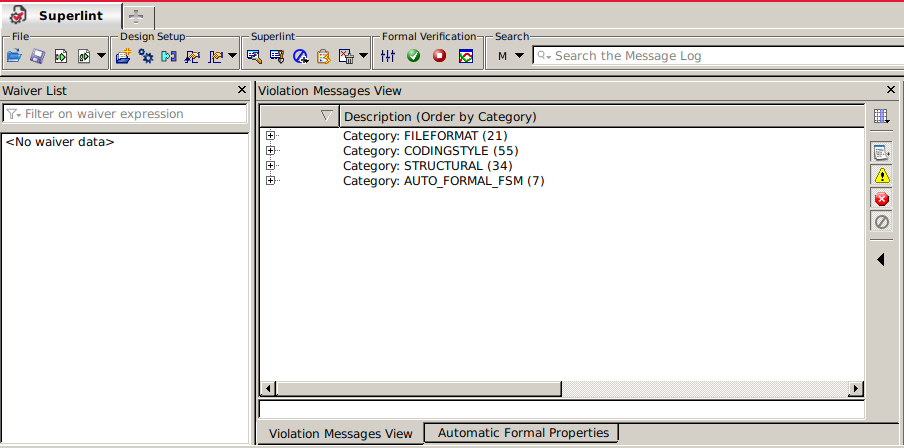
Boot這個程式必須將DRAM中的Instruction、sdata、data等Data存到Instruction SRAM和Data SRAM，因此此程式使用3個While Loop完成。

## Program2

Program2是Matrix Multiplication，此程式使用3層的For Loop完成Matrix A和Matrix B每個Element的Traverse。

# Superlint

Overall Report：

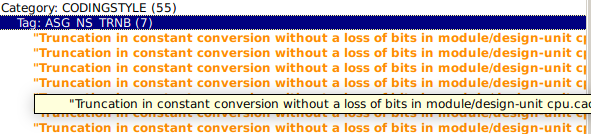


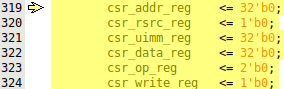
Total Violation = 117

Code Length = 9406

Score = 98.75%

Violation1：ASG\_NS\_TRNB

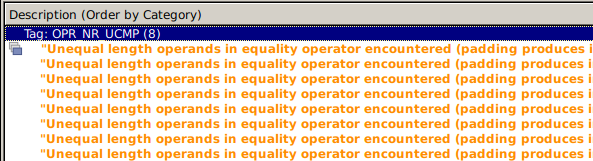




csr\_addr\_reg <= 32’b0;

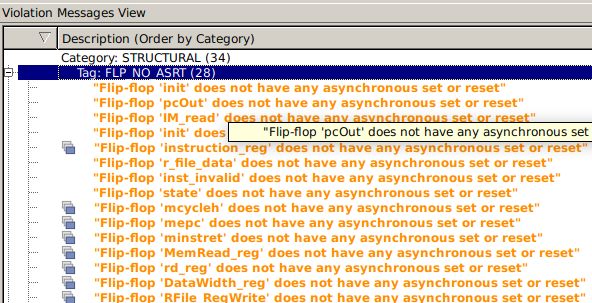
should be csr\_addr\_reg <= 32’d0;

Violation2：OPR\_NR\_UCMP



read == 0 Should be read == 1’b0

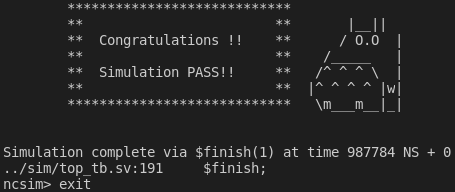
Violation3：FLP\_NO\_ASRT



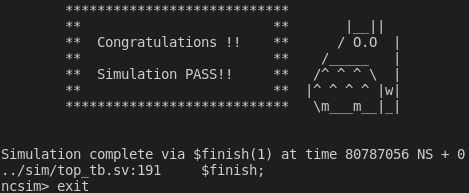
always\_ff @(posedge clock) should be always\_ff @(posedge clock or posedge reset)

# Performance & Area

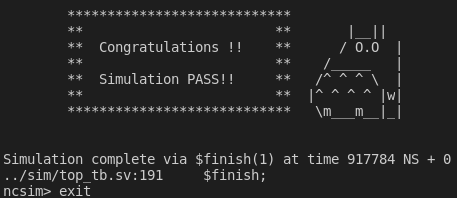
1. Performance(Simulation Time)



(圖) Program0



(圖) Program1

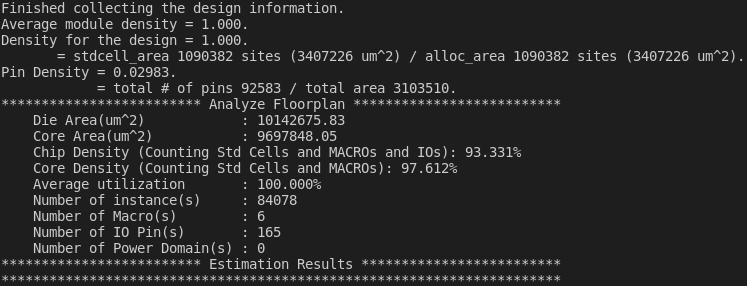


(圖) Program2

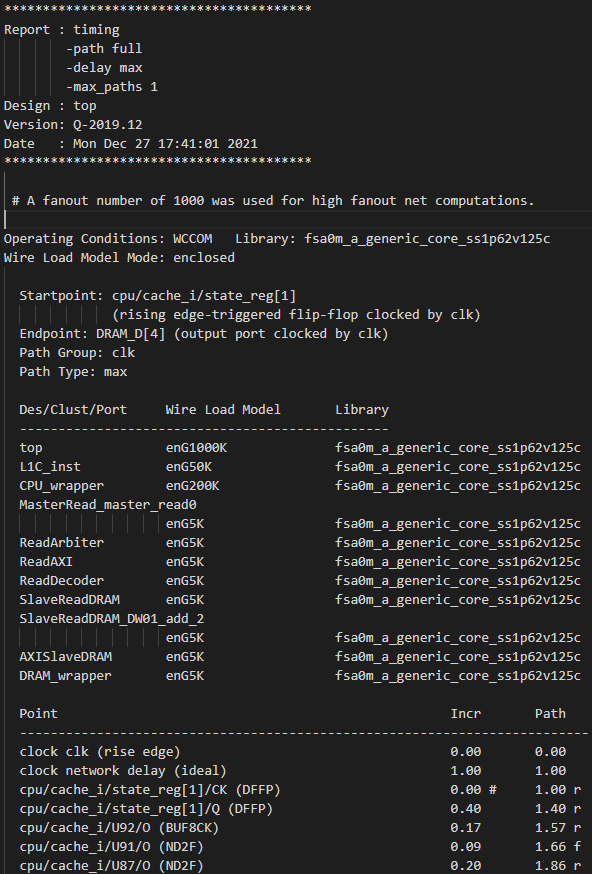
|  |  |  |  |
| --- | --- | --- | --- |
|  | Program0 | Program1 | Program2 |
| Execution Time | 987784(ns) | 80787056(ns) | 917784(ns) |

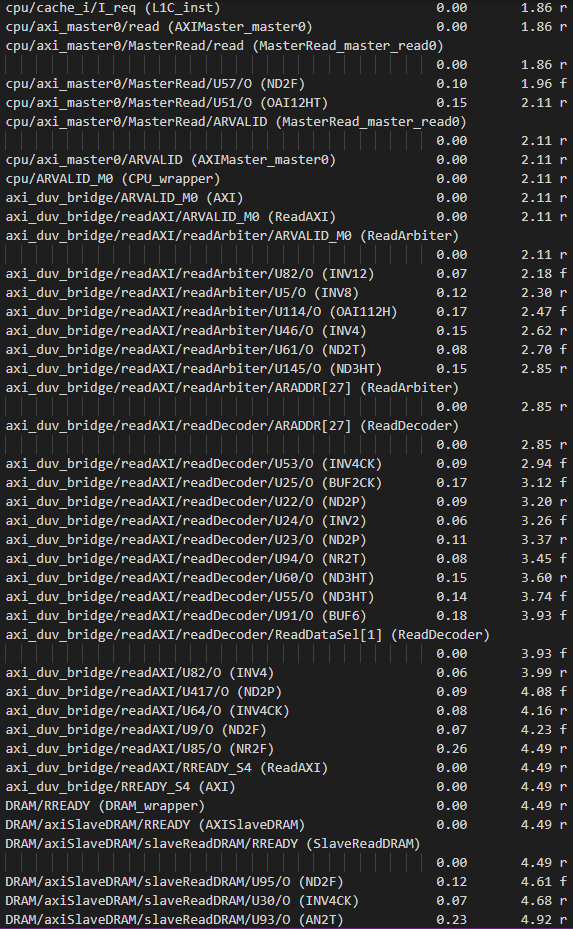
(表) Execution Time of Programs

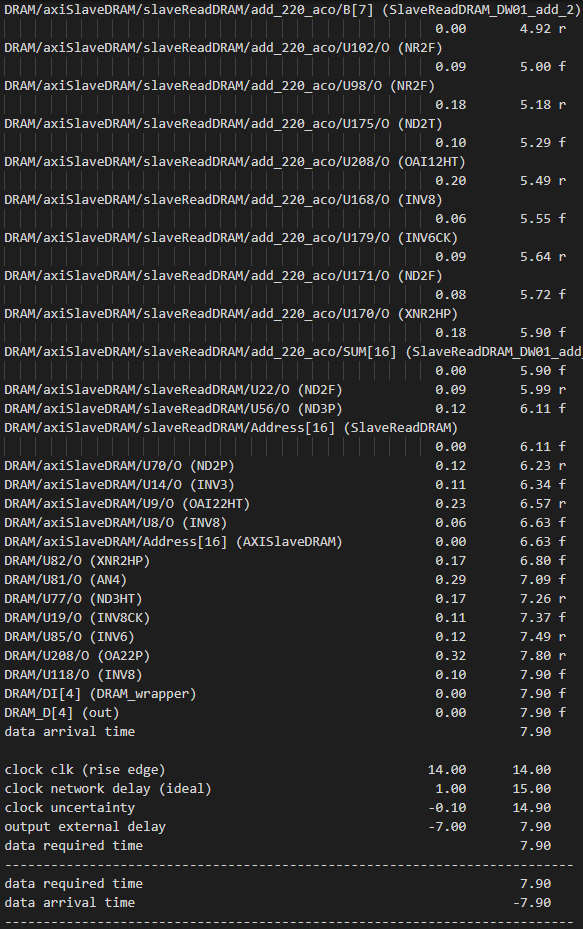
1. Area



1. Timing Report







# Lessons learned

李秉軒：

這次的作業真的是「摸著石頭過河」，開始寫之前，根本對Interrupt一點概念都沒有，也完全沒聽過CSR，光是了解其中的機制跟每個register的意義就花了好幾天。由於上次作業的cache無法合成到20ns以內，因此完成這次RTL的部分後，便回頭去修正上次作業的cache部分，也讓自己原本所想的優化方法可以在這次作業實現。過程中意外發現自己CPU的部分有問題，是過去幾次作業都沒碰到的corner case，只能說驗證真的很重要。

修改完cache後，終於能夠合成到15ns，但問了其他同學才得知，好像很多人都可以合成到10ns以內，雖然也有點想嘗試再優化，但已經筋疲力竭了，接下來就休息一下，好好準備Final project吧！

劉彥麟：

這次的作業跟上次的比要實作的內容少了許多，但是由於要APR，因此原本在Homework3中Synthesize Slack是負的就要改掉，所以我們花了非常多的時間在修改之前的Architecture，至於APR的部分，由於沒有用Cadence的APR軟體，因此花了一些時間熟習整個操作過程，其中Floorplan的部分非常耗時間和體力，一剛開始我遇到Violation就是整個重做，後來經過嘗試後才逐漸摸索出解Violation的方法；Homework終於告一段落了，接下來就終於可以專心做Final Project啦！再次感謝隊友以及助教的幫忙。