VLSI System Design (Graduate Level)

Fall 2021

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

內容

[Summary 2](#_Toc87397421)

[Contribution 3](#_Toc87397422)

[Problem 1 3](#_Toc87397423)

[Design Explanation 3](#_Toc87397424)

[Verification Result 9](#_Toc87397425)

[Problem 2 12](#_Toc87397426)

[Design Explanation 12](#_Toc87397427)

[Verification Result 17](#_Toc87397428)

[Terminal 17](#_Toc87397429)

[Waveform 20](#_Toc87397430)

[Superlint 33](#_Toc87397431)

[Synthesis Report 35](#_Toc87397432)

[Lessons learned 37](#_Toc87397433)

# Summary

本次作業中共分兩個部分，第一部份，我們完成了符合AXI protocol的Master、Bridge以及Slave的設計，並通過JasperGold ABVIP的驗證。第二部分，我們將作業一中的RISC V CPU接上我們設計的AXI Bus、完成4個program的測試。最後完整的CPU、AXI bus及SRAM經過合成，可達到clock period為10ns的performance。

在報告中，我們首先會說明本次作業中的分工，接著會介紹AXI bus的設計以及JasperGold ABVIP的驗證結果。第二部分會說明CPU架構如何被調整為透過AXI bus存取Instruction Memory與Data Memory，以及4個program的測試結果。最後則會說明我們合成的結果，以及本次作業中遇到幾個比較大的問題與我們的解決辦法。

# Contribution

劉彥麟(N26100723) ： 50%

AXI bridge、AXI slave設計、Superlint測試、Synthesis和Post-sim測試。

李秉軒(N26100595)：50%

AXI Master設計、CPU架構調整、Pre-sim測試。

# Problem 1

## Design Explanation

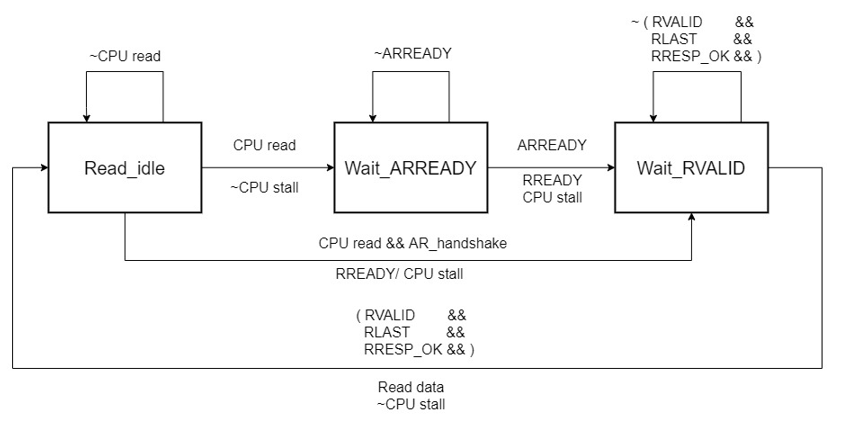
Problem 1的部分主要是AXI bus的設計，以下將分別對Master、Bridge、Slave作介紹 。

1. Master

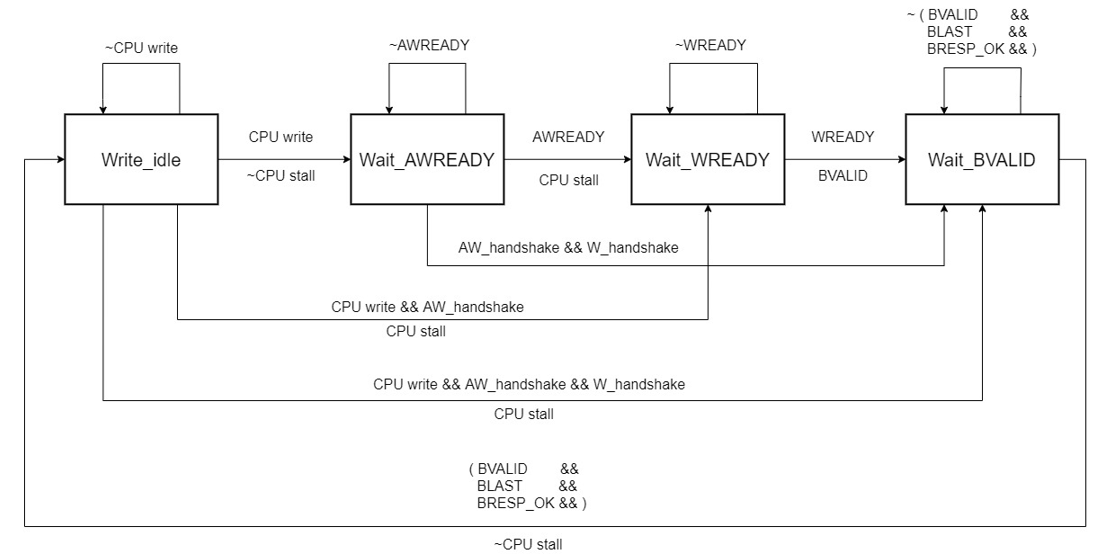
在本次作業中，Master端是CPU wrapper。而在wrapper中，有CPU和兩個AXI Master的module，兩個AXI master module分別負責CPU對Instruction Memory(IM)和Data Memory(DM)的存取。

下圖是整個CPU wrapper的架構圖，可以看到CPU分別有兩組port發出request，而由於IM為read-only，因此在IM端只能發出read request。而master module在處理request的同時，也會發出stall訊號給CPU，CPU內部的stall機制會在Problem 2詳細介紹。

設計上，我們再把一個Master/Slave分為Read, Write，分別處理Read/Write的行為。Read, Write module的設計是採用Finite State Machine的方式，下圖分別為Read, Write的FSM。



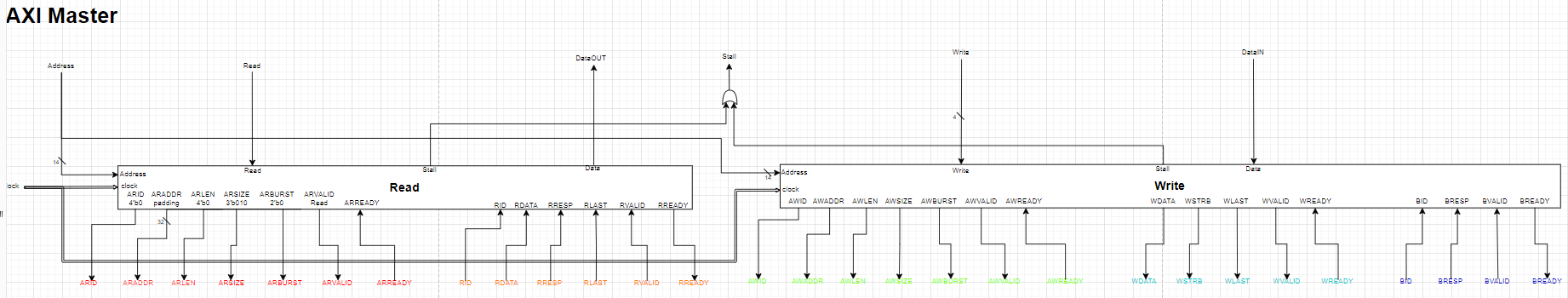
(圖) Master Read FSM



(圖) Master Write FSM

在一般情況，Read/Write module都是在idle狀態，在收到CPU發出的request後，會開始透過Bus和Slave進行溝通。為了簡化CPU stall的機制，我們讓wrapper的output都是combinational，也就是說，CPU發出request到wrapper發出request，會是在同個cycle。而wrapper回傳的stall訊號也會是在同個cycle內。

而因為在ABVIP驗證中，會測試Bridge立即回傳Ready的情況。因此在FSM中才會看到從idle直接跳到交握完成的state。

最後，在Write的情況下，W channel必須比AW channel晚交握完成，因此在write module的設計上，我們強制WVALID訊號必須在AW交握完才能拉起，如此才能符合protocol。

(圖) Master Block Diagram

1. Bridge

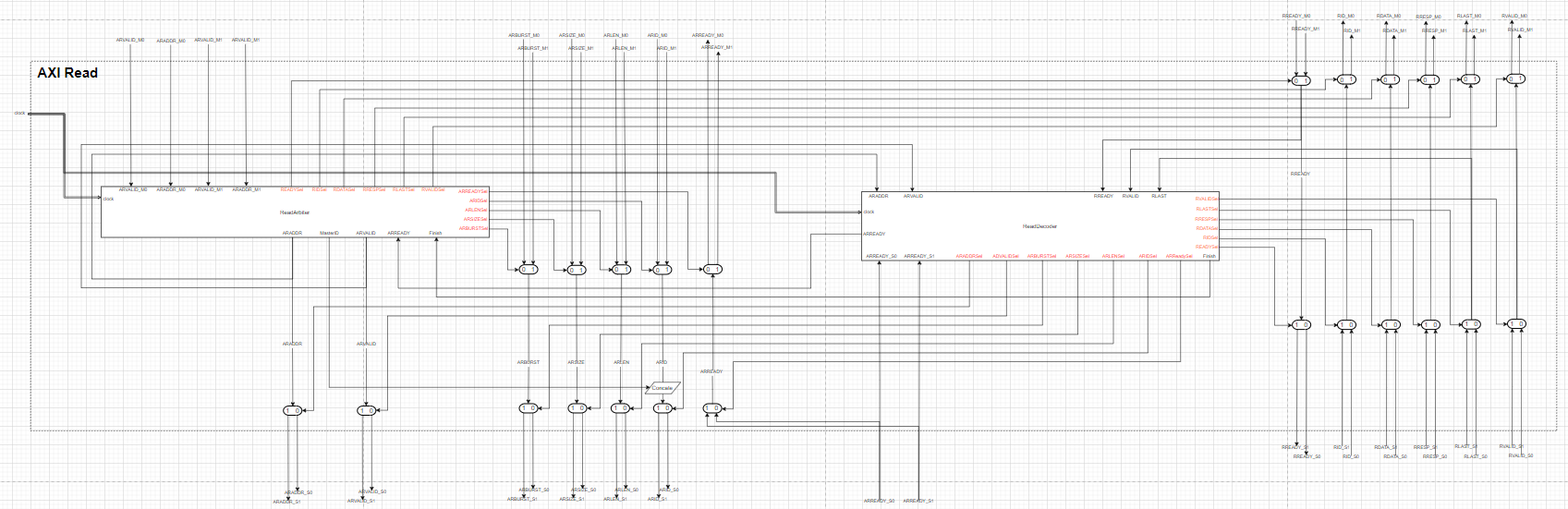
在本次作業中，Bridge分為Read和Write兩個Module：

1. Read Module

Read Module負責AXI Bus中AR Channel和R Channel，Read Module中包含Read Arbiter和Read Decoder，兩者皆使用FSM的方法撰寫，其中Read Arbiter負責Master端的MUX，而Read Decoder則負責Slave端的MUX。

剛開始執行時，Read Arbiter和Read Decoder皆屬於IDLE State，Read Arbiter會檢測Master端是否有將ARVALID設為High，如果只有一個Master將ARVALID設為High，則Arbiter會選擇該Master之ARADDR，如果有大於一個Master將ARVALID設為High，則Arbiter會使用Round Robin的方式去選擇ARADDR；Arbiter選擇完ARADDR後，Decoder會收到ARVALID、ARADDR，如果ARVALID為High，則Decoder會依據ARADDR來判斷該分配給哪個Slave，如果ARADDR介於0x0000\_0000〜0x0000\_FFFF，則會將Slave 0的ARREADY傳到Arbiter，ARADDR介於0x0001\_0000〜0x0001\_FFFF，則會將Slave 1的ARREADY傳到Arbiter，上述的兩種可能的ARREADY為TRUE時，Decoder下個Cycle會進入BUSY state，如果不是上述的兩種結果，則會回傳TRUE給Arbiter並讓Decoder下個Cycle進入WRONGADDRESS state，去處理非法的Address；如果Arbiter收到的ARREADY為TRUE，則Arbiter下個Cycle會進入BUSY state。

如果Arbiter為BUSY state，則其會將Master MUX的Selection Signal固定為Arbiter選擇的Master，如果Decoder為BUSY state，則其會將Slave MUX的Selection Signal固定為ARADDR對應的Slave，如果Decoder為WRONGADDRESS state，則Read Module會使用內部的FSM處理這個ERROR，並且將RRESP設為DECERR。

Arbiter和Decoder會一直等到RREADY、RVALID、RLAST三種Signal皆為TRUE時，才會從BUSY state回到IDLE state。

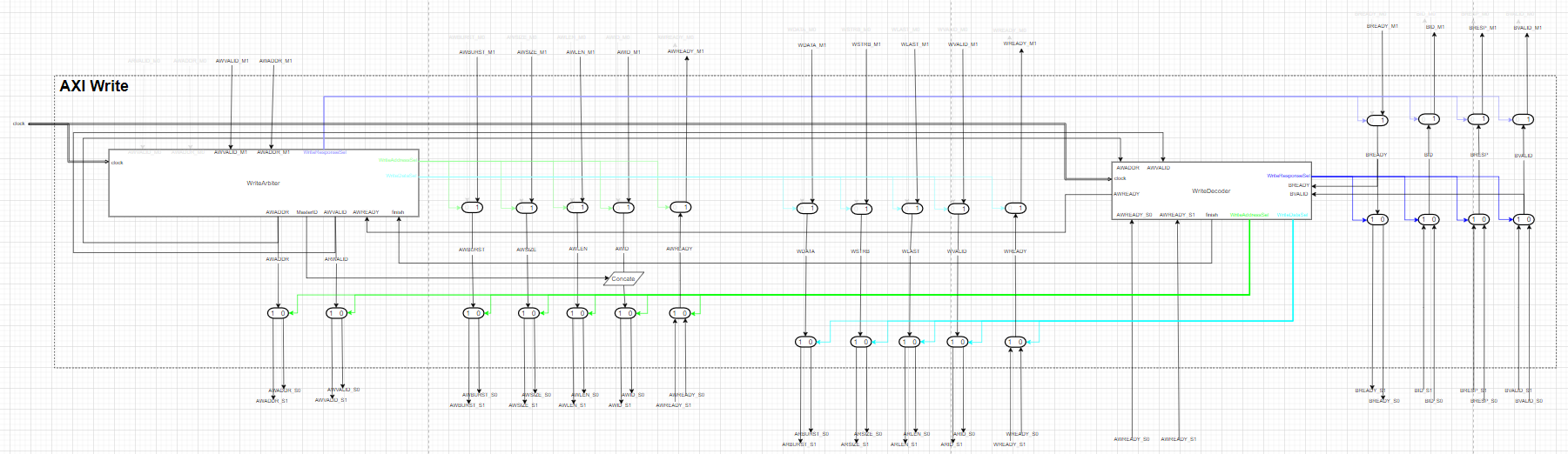
(圖) Read Module Diagram

1. Write Module

Write Module負責AXI Bus中AW Channel、W Channel、B Channel，Read Module中包含Write Arbiter和Write Decoder，兩者皆使用FSM的方法撰寫，其中Write Arbiter負責Master端的MUX，而Write Decoder則負責Slave端的MUX。

剛開始執行時，Write Arbiter和Write Decoder皆屬於IDLE State，Read Arbiter會檢測Master端是否有將AWVALID設為High，如果只有一個Master將AWVALID設為High，則Arbiter會選擇該Master之AWADDR，如果有大於一個Master將AWVALID設為High，則Arbiter會使用Round Robin的方式去選擇AWADDR；Arbiter選擇完AWADDR後，Decoder會收到AWVALID、AWADDR，如果AWVALID為High，則Decoder會依據AWADDR來判斷該分配給哪個Slave，如果AWADDR介於0x0000\_0000〜0x0000\_FFFF，則會將Slave 0的AWREADY傳到Arbiter，AWADDR介於0x0001\_0000〜0x0001\_FFFF，則會將Slave 1的AWREADY傳到Arbiter，上述的兩種可能的AWREADY為TRUE時，Decoder下個Cycle會進入BUSY state，如果不是上述的兩種結果，則會回傳TRUE給Arbiter並讓Decoder下個Cycle進入WRONGADDRESS state，去處理非法的Address；如果Arbiter收到的AWREADY為TRUE，則Arbiter下個Cycle會進入BUSY state。

如果Arbiter為BUSY state，則其會將Master MUX的Selection Signal固定為Arbiter選擇的Master，如果Decoder為BUSY state，則其會將Slave MUX的Selection Signal固定為AWADDR對應的Slave，如果Decoder為WRONGADDRESS state，則Write Module會使用內部的FSM處理這個ERROR，並且將BRESP設為DECERR。

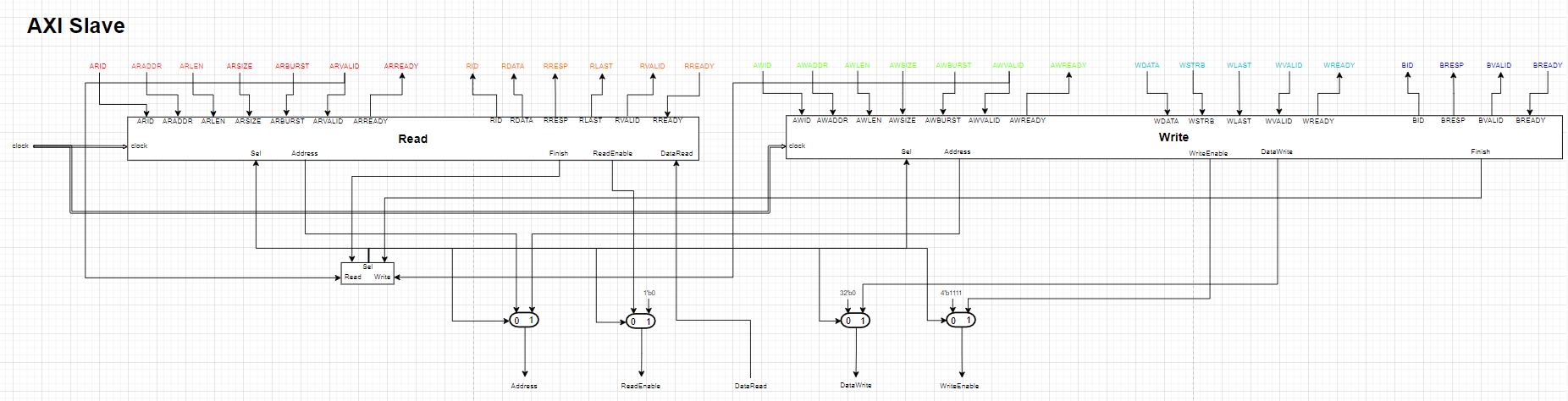
Arbiter和Decoder會一直等到BREADY 、 BVALID兩種Signal皆為TRUE時，才會從BUSY state回到IDLE state。

(圖) Write Module Diagram

1. Slave

在本次作業中，Slave分為Read和Write兩個Module，Read Module負責AR、R兩個Channel，而Write Module負責AW、W、B三個Channel，如果，Slave同時只會讓其中一個Module運作，若ARVALID和AWVALID同時為TRUE時，Slave會優先讓Read Module運作。

Read Module 使用FSM撰寫，state分為ADDRESS state和DATA state，剛開始為ADDRESS state，AR Handshake結束之後會轉為DATA state，如果有需要進行Burst，則Read Module會算出下一個Address，直到最後一筆資料回來Read Module會將RLAST設為TRUE，並且下個Cycle回到DATA state。

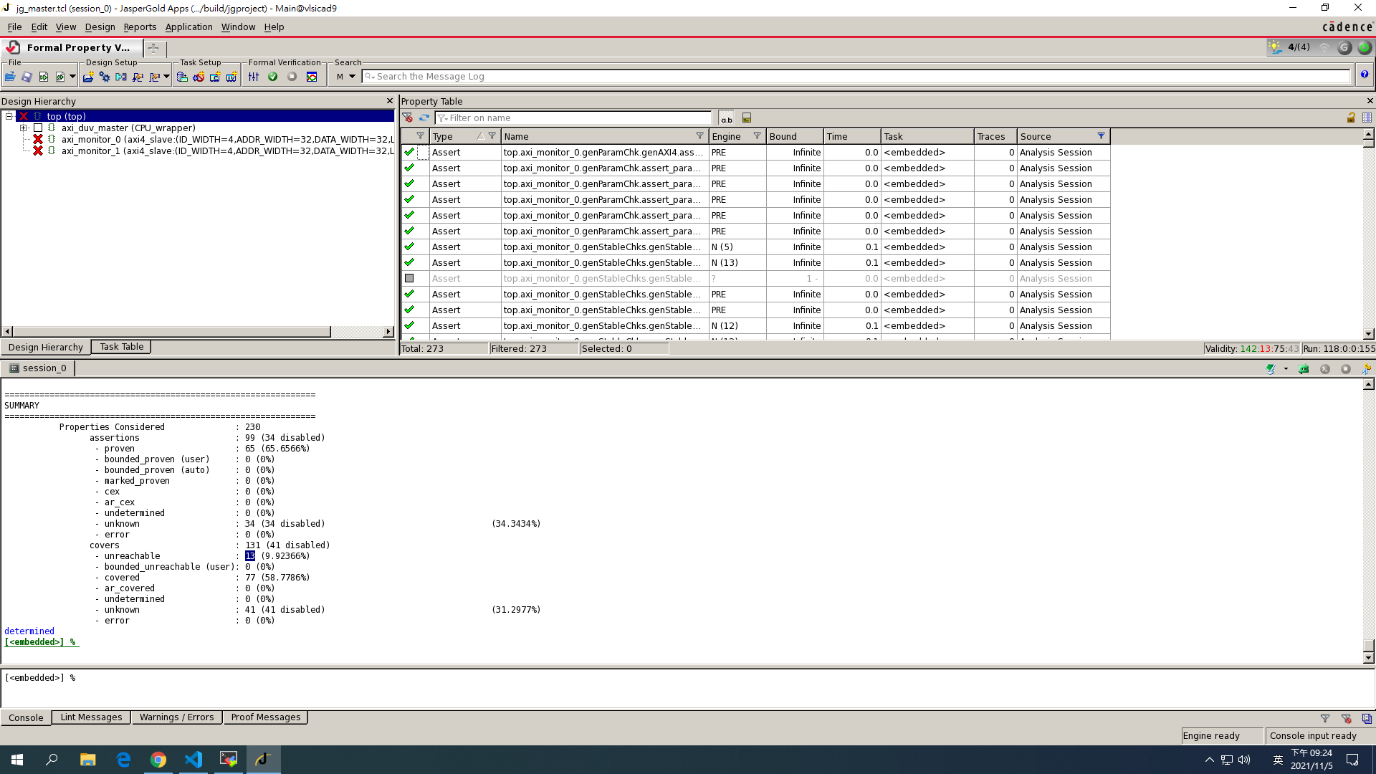
Write Module 使用FSM撰寫，state分為ADDRESSDATA state和RESPONSE state，剛開始為ADDRESSDATA state，AW Handshake和W Handshake結束之後會轉為RESPONSE state，如果有需要進行Burst，則Write Module會算出下一個Address，直到最後一筆資料寫入之後會轉為RESPONSE state，RESPONSE state會將BRESP設為OKAY，並且下個Cycle回到ADDRESSDATA state。

(圖) Slave Block Diagram

## Verification Result

本設計之Master, Bridge, Slave皆經過JasperGold ABVIP驗證，以下為驗證結果：

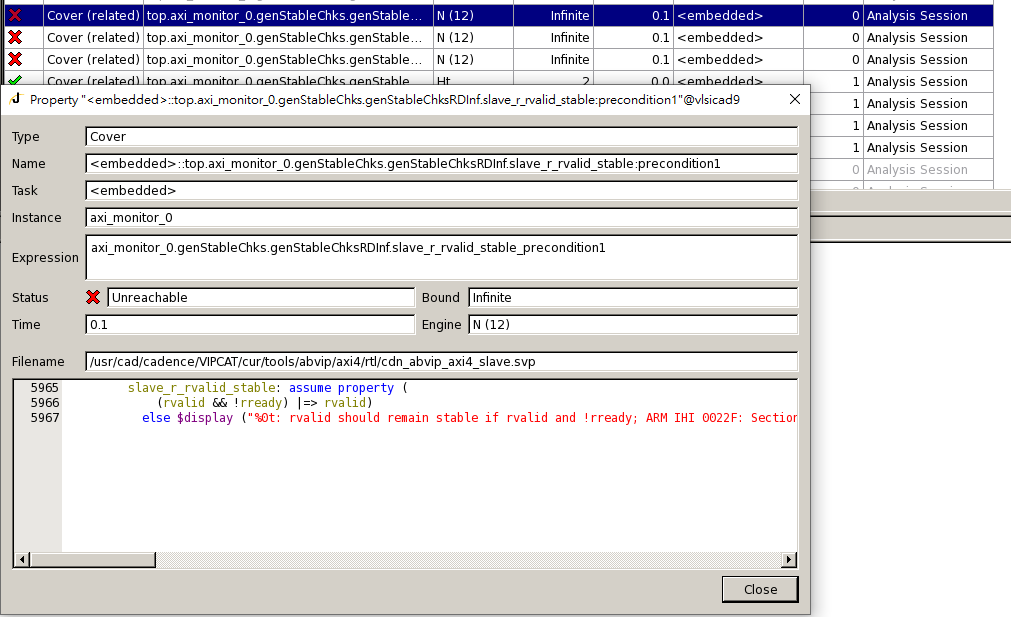
1. Master

(圖) Master VIP result

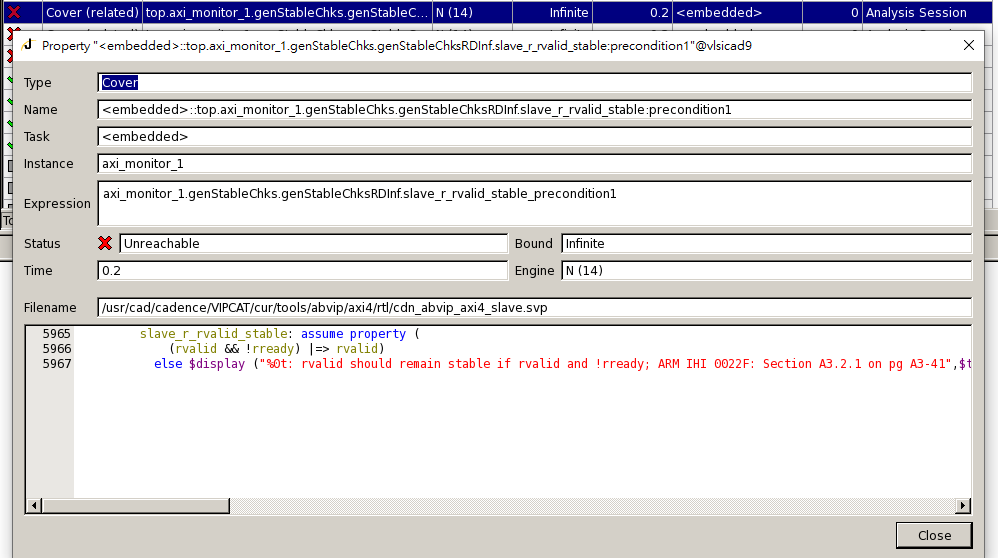
Master的部分由於設計關係，有部分Cover無法被滿足。在我們的設計中，只要完成AR/AW/W channel的handshake，下個cycle就會將RREADY/BREADY訊號拉high，因此無法測試到Slave端訊號在handshake前要保持stable的部分。總共的Cover錯誤數共有：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | M0 | M1 |  | |
| R channel | 5 | 5 |  | |
| B channel |  | 3 |  | |
| Total covers | 5 | 8 | 13 |

以下為部分Cover截圖：

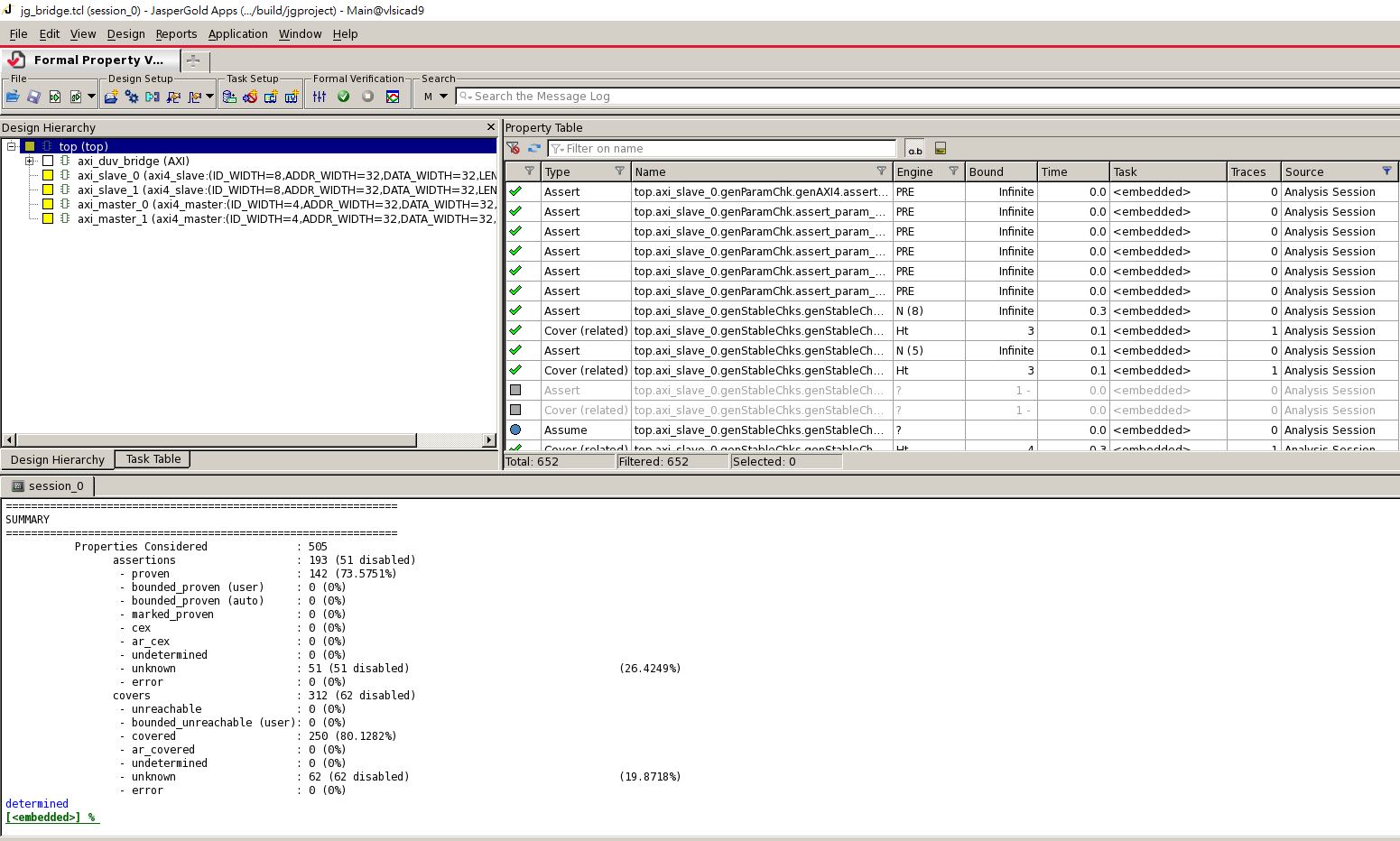


(圖) Master cover error : rvalid of master0



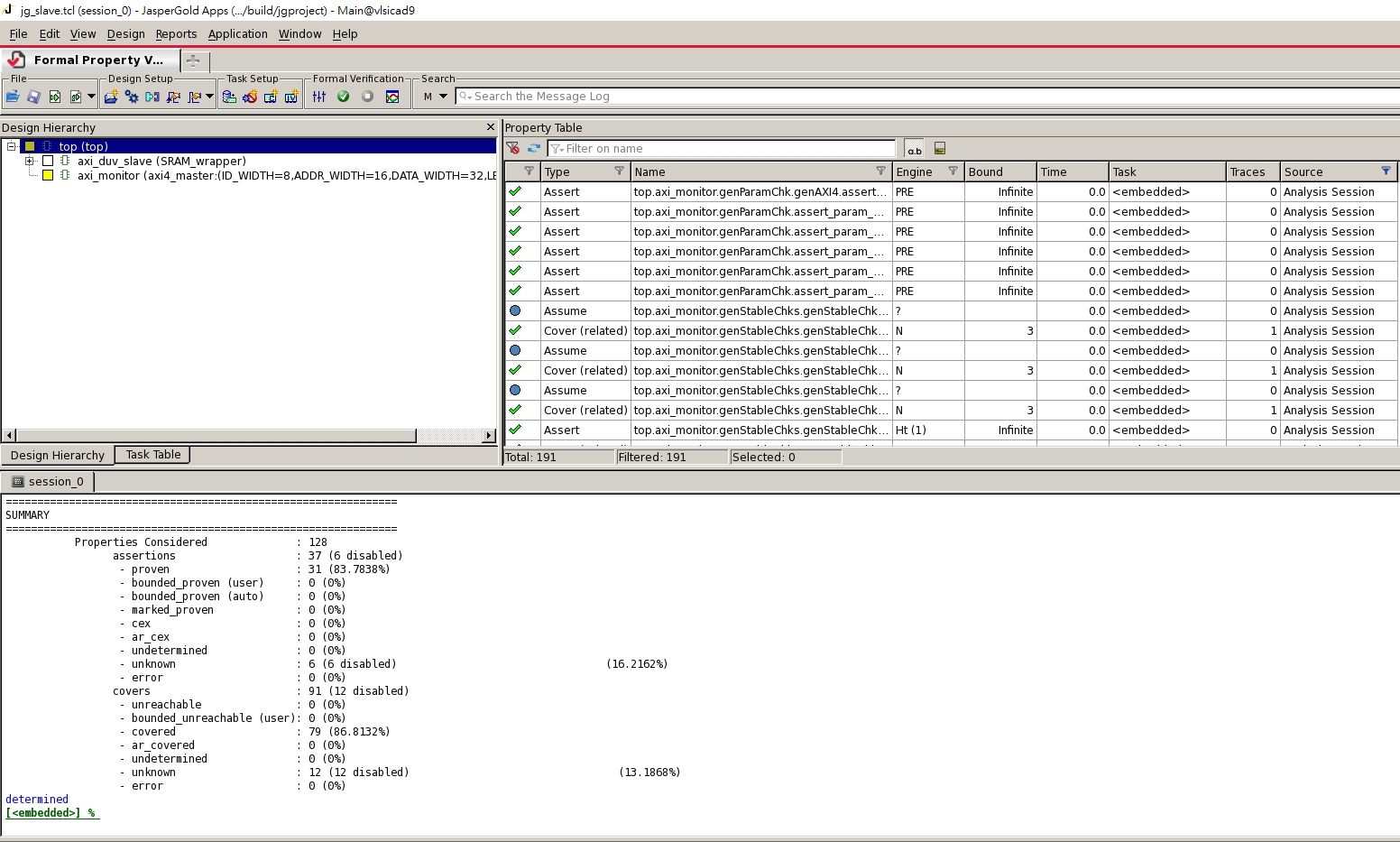
(圖) Master cover error : rvalid of master1

1. Bridge



(圖) Bridge VIP result

1. Slave

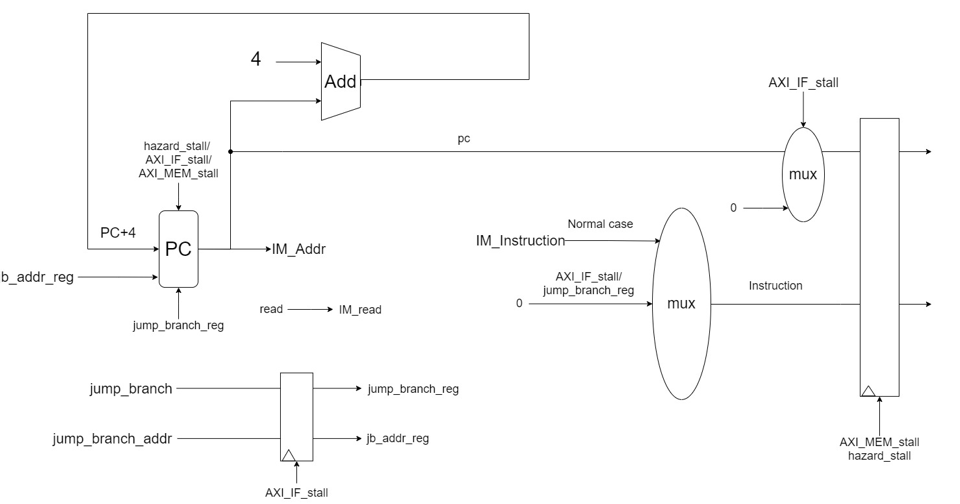


(圖) Slave VIP result

# Problem 2

## Design Explanation

Problem 2的設計主要是將作業一的CPU加入CPU wrapper中，本次作業使用的CPU是李秉軒同學在作業一中的CPU。和作業一相比，主要的差異在Mem stage的stall以及跳轉指令的實作，以下介紹更改後的CPU 各級架構圖。



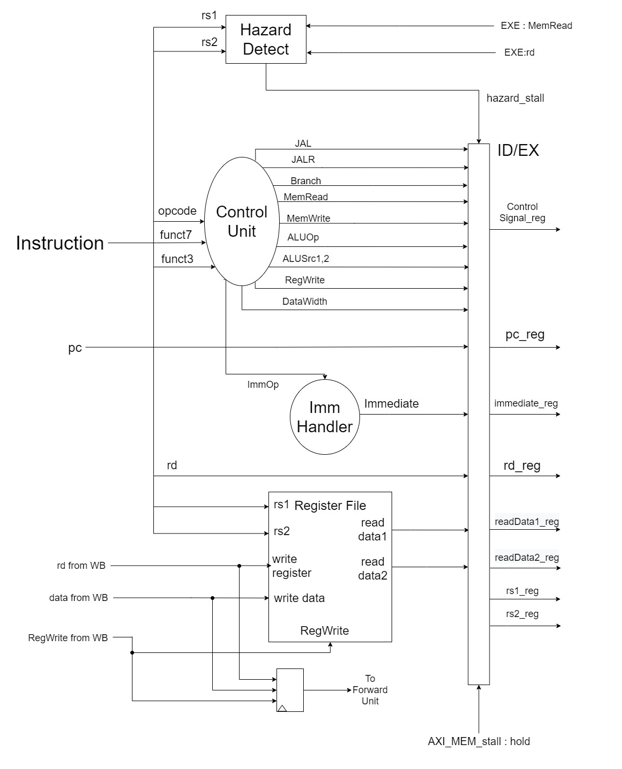
(圖) IF stage 架構

首先介紹CPU的IF stage架構，也是CPU中主要修改的部分。由於我們AXI bus的設計會使得指令在3個cycle後才回來，因此必須加入一些判斷讓IF級往後送NOP instruction，此處用來判斷的訊號取名為AXI\_IF\_stall，由Wrapper控制。當AXI\_IF\_stall為high時，IF級會往後傳送NOP指令，並stall住PC；當AXI\_IF\_stall為low，代表指令已經回到CPU，並且會在下個cycle送至下一級。

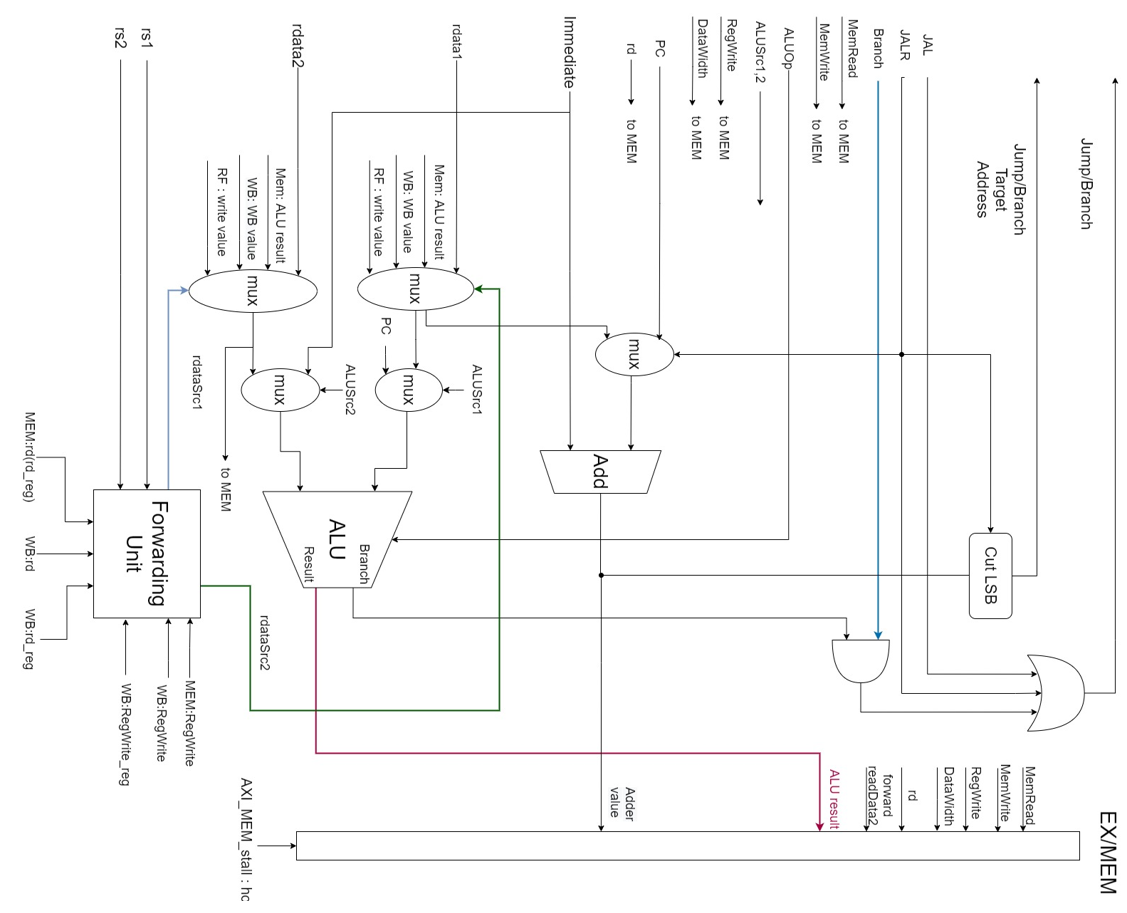
Branch/Jump :

當EXE級偵測到指令要跳轉時，會傳一個訊號和要跳轉的地址到IF級，分別為jump\_branch 和 jump\_branch\_addr。然而此時IF級已經發出下個指令的request，必須等到該條指令回來後，再發出跳轉指令的request。因此，我們使用register儲存jump\_branch訊號，並等到當前錯誤的request結束，用來當作遮蓋掉錯誤指令的判斷訊號，並在當個cycle使用後被清空。當AXI\_IF\_stall被拉為low，代表request已經結束，該register就會在下個cycle被清空。

接下來的ID stage, EXE stage則和上次作業沒有差異，因此不多作介紹。



(圖) ID stage 架構



(圖) EXE stage 架構

一張含有 桌 的圖片

自動產生的描述

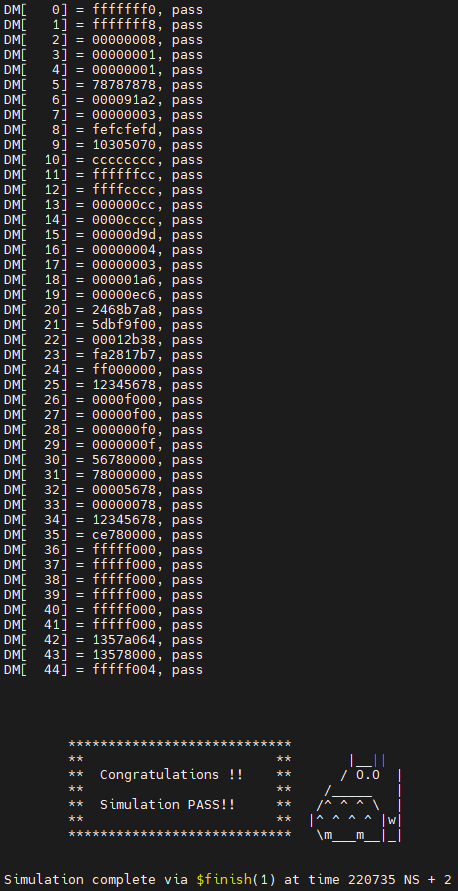
(圖) MEM stage 架構

再來是MEM stage，同樣有判斷是否需要stall的訊號，稱為AXI\_MEM\_stall，同樣由wrapper控制。

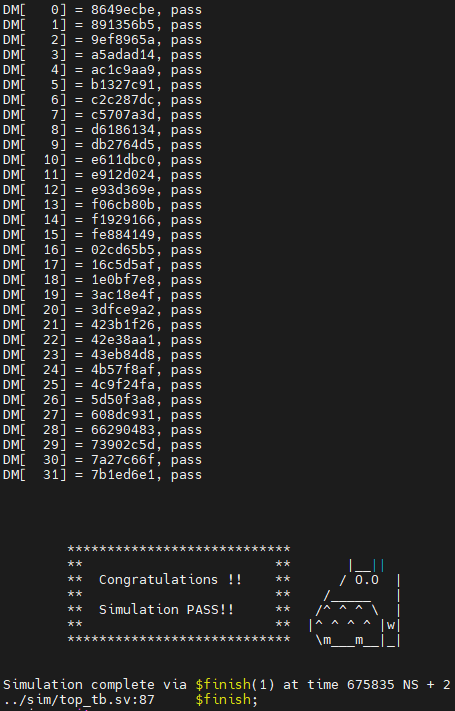
當MEM級在等待request被完成時，前面的stage都必須被停住，實現的方法是，讓前面幾級的pipeline registers都輸出原本的值，不要輸出新的值。而至於MEM後面的stage，則同樣輸出NOP就好。

## Verification Result

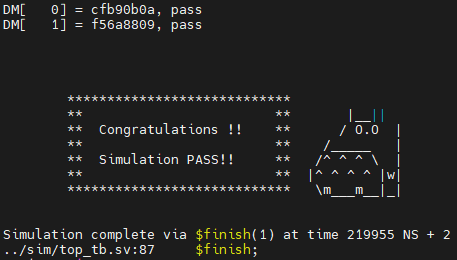
### Terminal



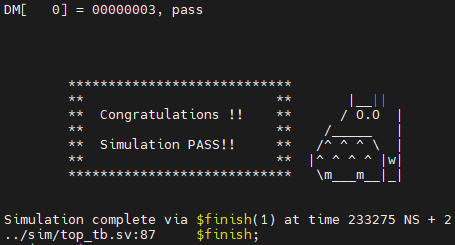
(圖) Program0



(圖) Program1



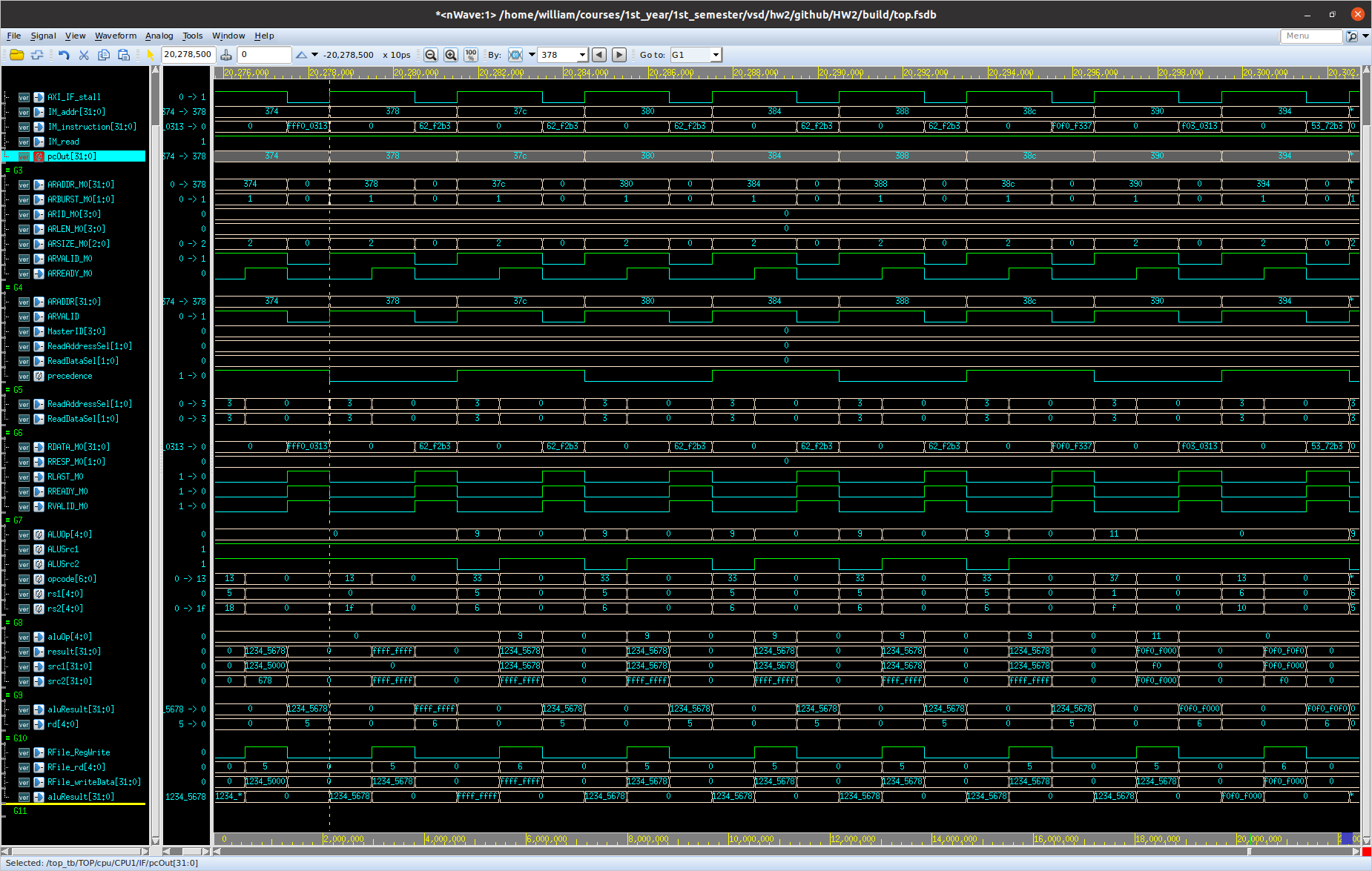
(圖) Program2



(圖) Program3

### Waveform

R-Type:



(圖) R-TYPE指令: AND, pc = 0x378

如上圖是一個R-TYPE指令的波型，以下就不同Group作介紹。首先Group 2是IF級訊號，IM前綴代表與IM存取相關的訊號，AXI\_IF\_stall則是wrapper控制CPU stall的訊號。當pc\_out更新，CPU就會對IM發出新request，然後被wrapper stall住，直到指令(IM\_instruction)回來。

Group 3則是wrapper裡的訊號，在指令級只會使用到AR及R channel。ARBURST = 1代表burst的地址是increment，但在本次設計中不會用到。ARSIZE = 2代表資料為4byte。ARLEN代表不會用到burst。

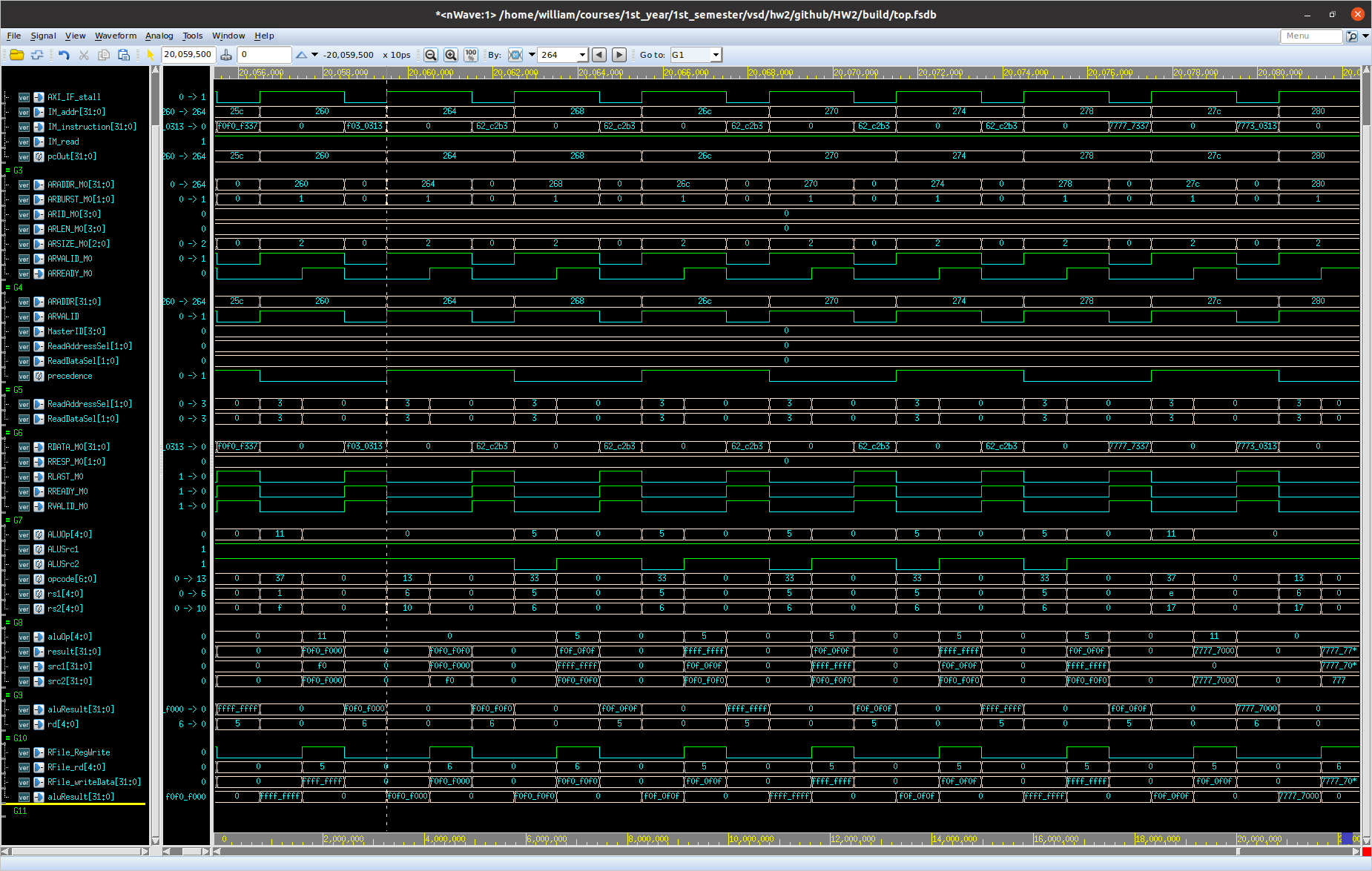
Group 4是Bridge裡的Arbiter，當有大於一個Master向Bus發出Request時，Arbiter會根據precedence選擇Master，收到Request後Arbiter會將ARADDR、ARVALID送到Bridge的Decoder，並且輸出Arbiter目前選擇的Master之ID，Arbiter選擇Master後會將Selection Signal輸出給Master MUX，並且根據目前的state選擇是否需要Hold Selection Signal。

Group 5 是Bridge裡的Decoder，Decoder會根據目前的state和ARADDR選擇是否輸出Selection Signal。

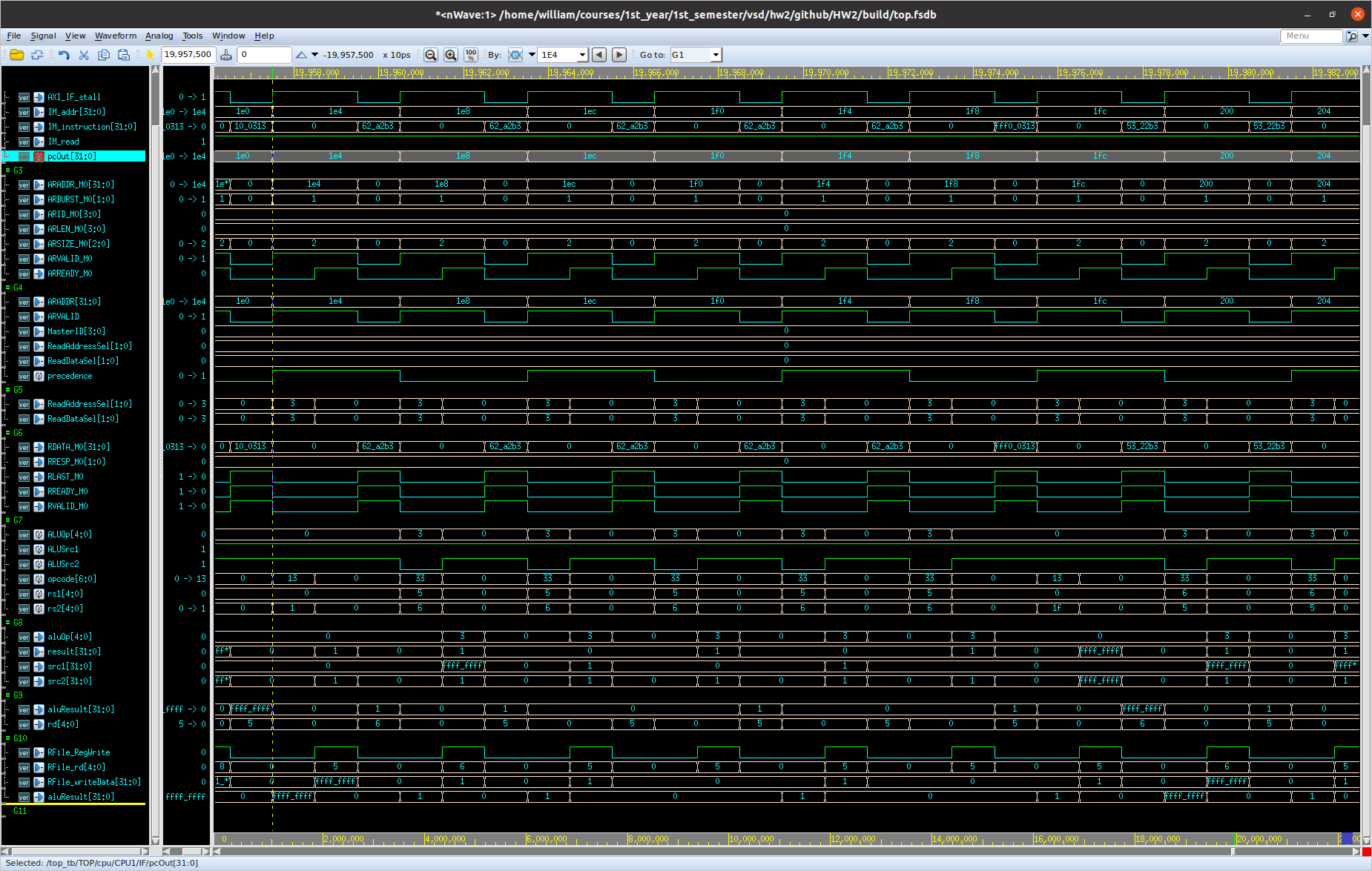
Group 6 是回到wrapper的R channel資料，可以看到回來正確的指令資料，且必須判斷為最後一筆burst資料(RLAST = 1)，RRESP不可以是error。皆判斷完後，wrapper就會將AXI\_IF\_stall設為low，IF級的pipeline registers就會在下個cycle將指令送去ID級。

Group 7 是ID級，會decode出為AND指令(ALUOP = 9)。Group 8 則是EXE級，會算出(0x1234\_5678 & 0xffff\_ffff = 0x1234\_5678)。Group 9則是MEM級，在R-TYPE指令中不會使用。Group 10則是write back回Register File。

下圖則是其他R-TYPE指令波型。

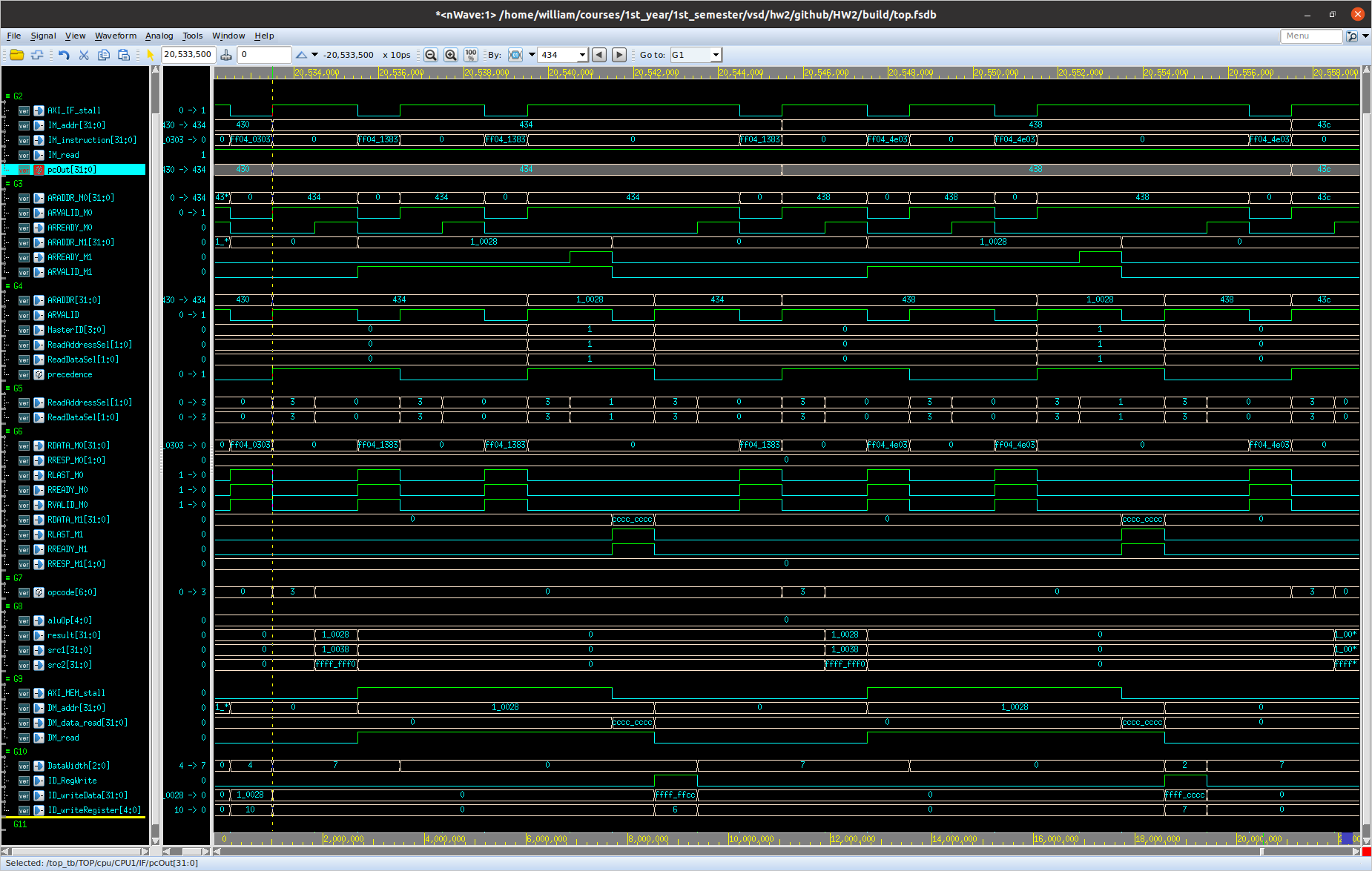


(圖) R-TYPE指令: XOR, pc = 0x264



(圖) R-TYPE指令: SLT, pc = 0x1E4

接下來則是I-Type指令，這邊以Load指令來做為舉例，下圖為LH指令波形圖：



(圖) I-TYPE指令: LH, pc = 0x434

和R-Type不同的是，Load指令需要存取Data memory， 因此可以看到在Wrapper(G2)，ARVALID\_M1訊號被拉高(此處由於上個指令也是load指令，所以在時間軸上比較靠近pc = 0x434的其實是上個指令的request)。

Wrapper傳送request到Bridge後，Arbiter必須決定誰可以獲得存取權，在該圖中，剛好round robin的優先度輪到M0，所以會讓M0再一次發送指令的request，但由於MEM級stall，實際上這次request回來的指令也無法使用，導致cycle被浪費，這部分是我們的arbitration機制可以被改進的部分。

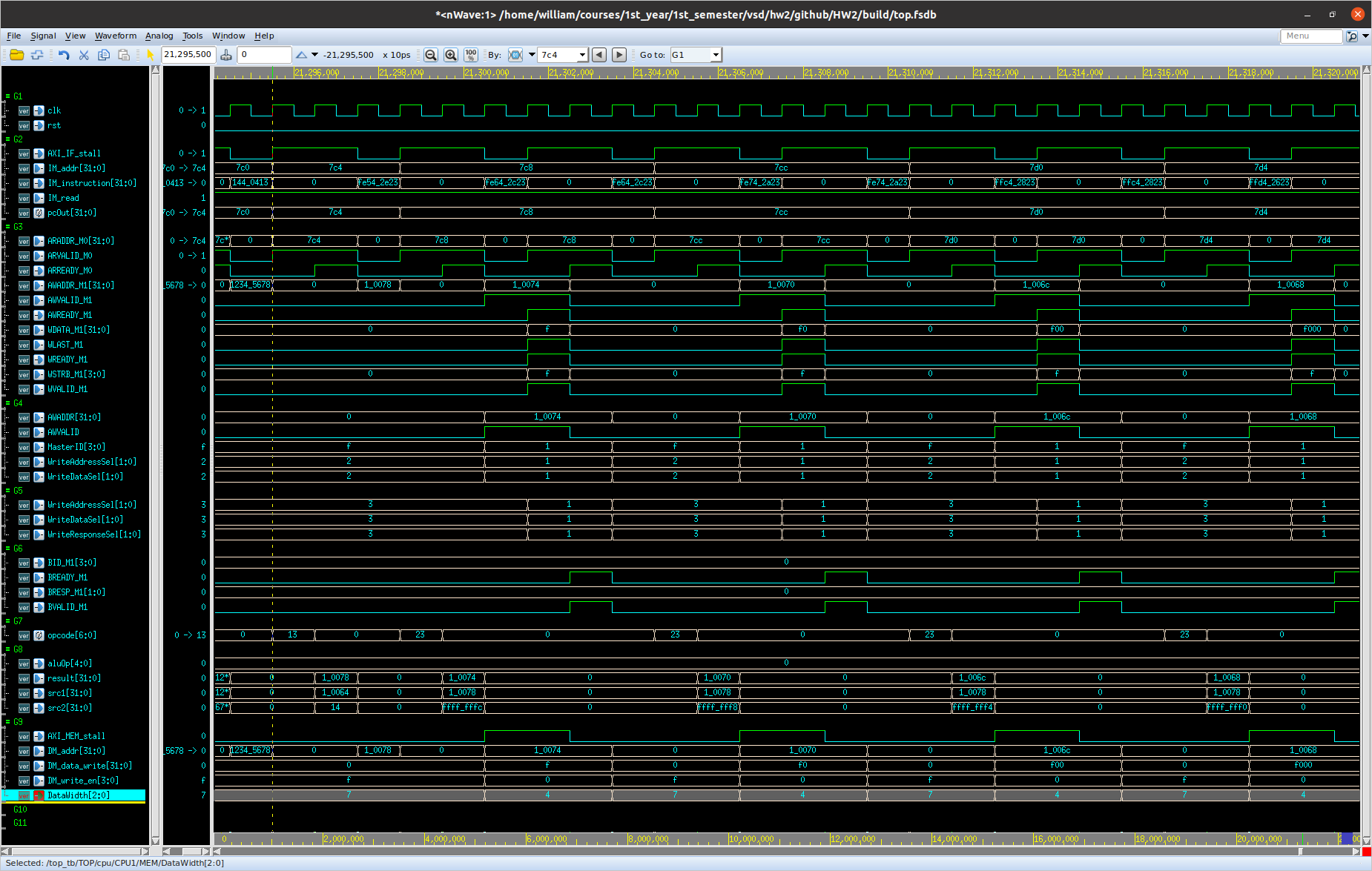
Arbiter決定request權限後，剩下就和IM的存取差異不大。資料讀取回來後，MEM級的AXI\_MEM\_stall訊號被拉低，讀回來的資料會在WB級被處理，由於該指令為LH，WB級的電路會根據memory address的最低位2bit判斷，有效的16bit資料位在讀回來的32bit資料中的哪個部分，並決定是否要sign-extend。

最後處理完的資料便會被寫回Register File，下圖是其他load指令的波形圖。

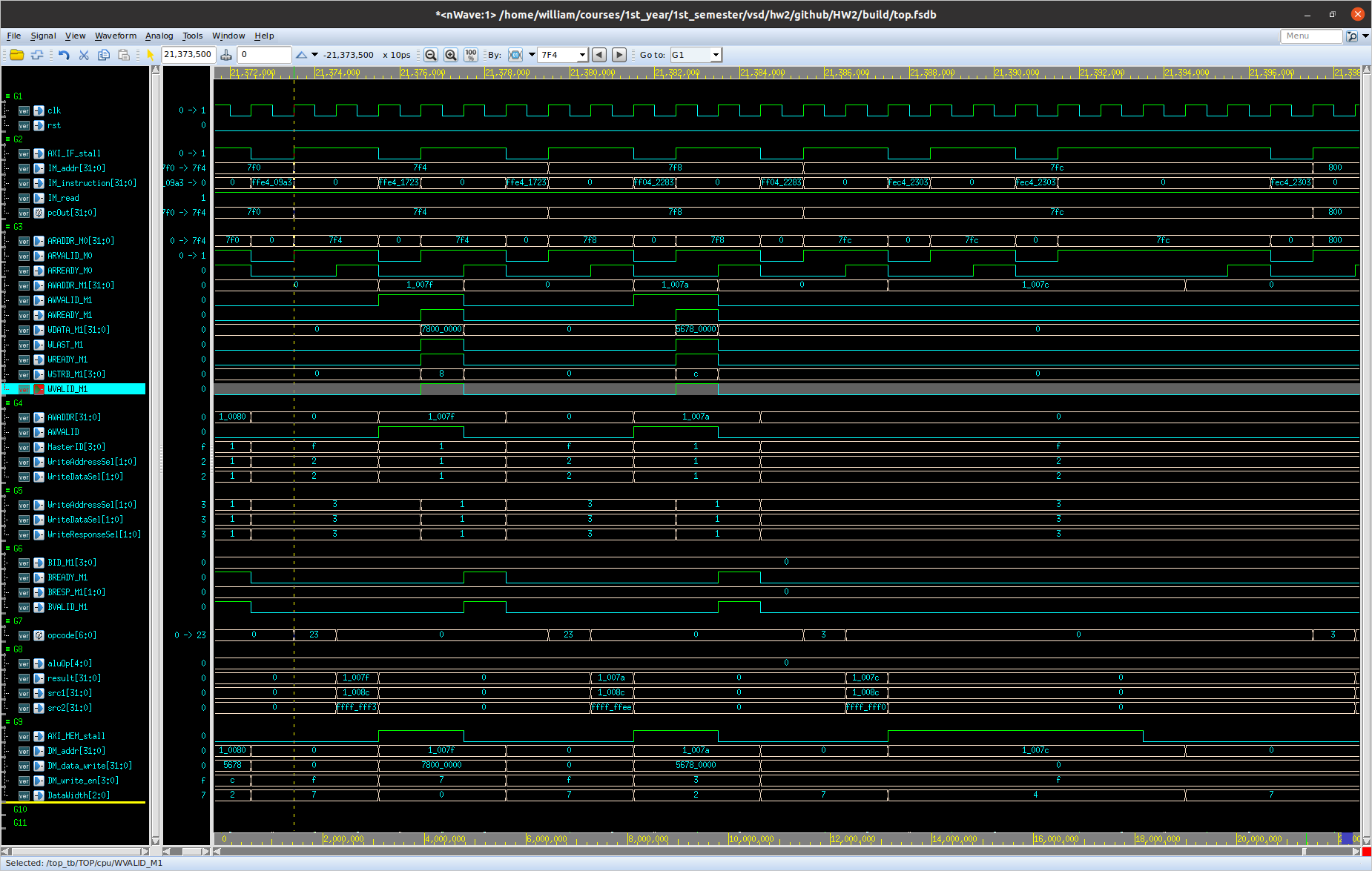
一張含有 文字, 電子用品, 電腦 的圖片

自動產生的描述

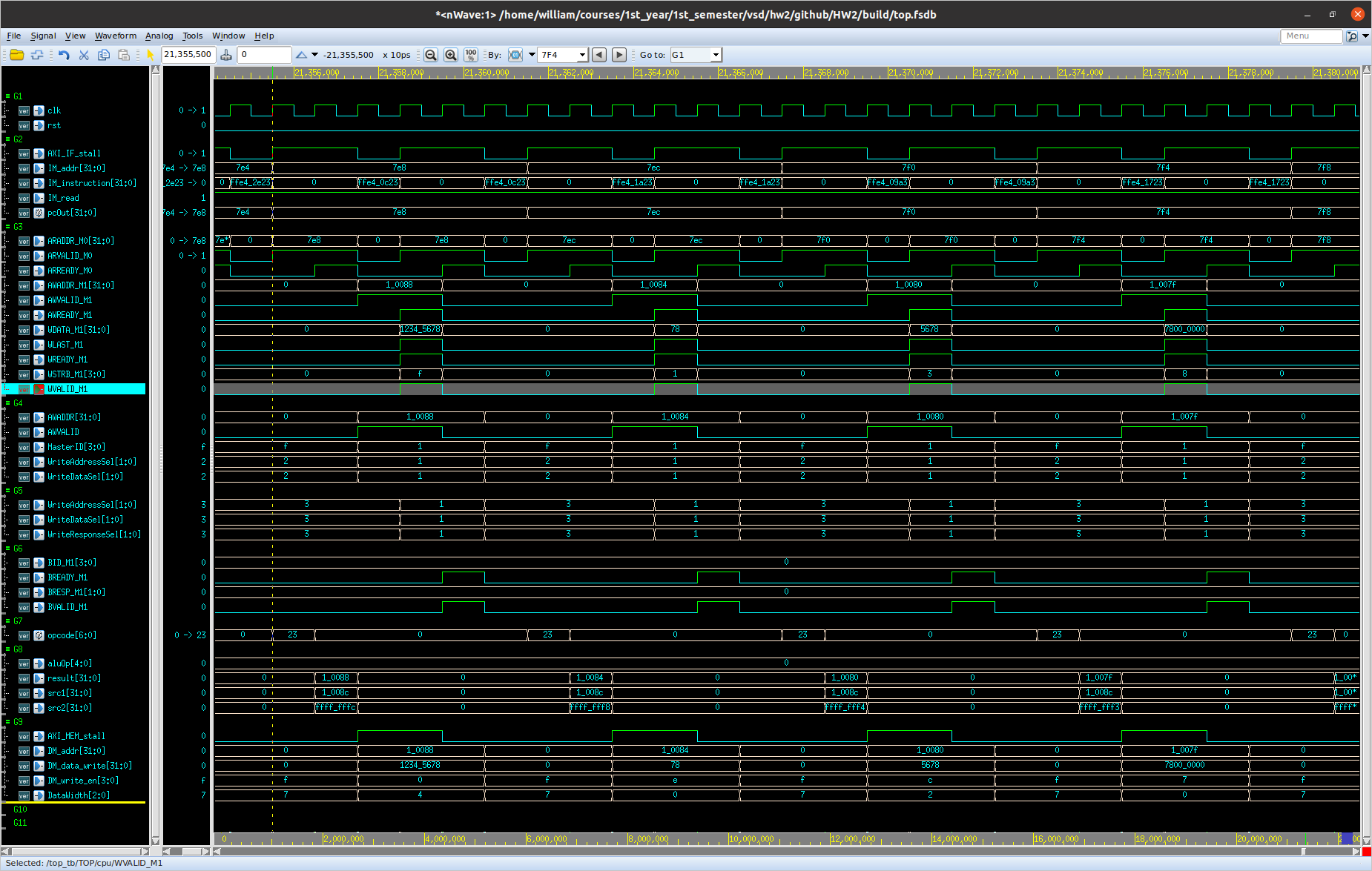
(圖) I-TYPE指令: LBU, pc = 0x438



(圖) S-TYPE指令: SW, pc = 0x7C4

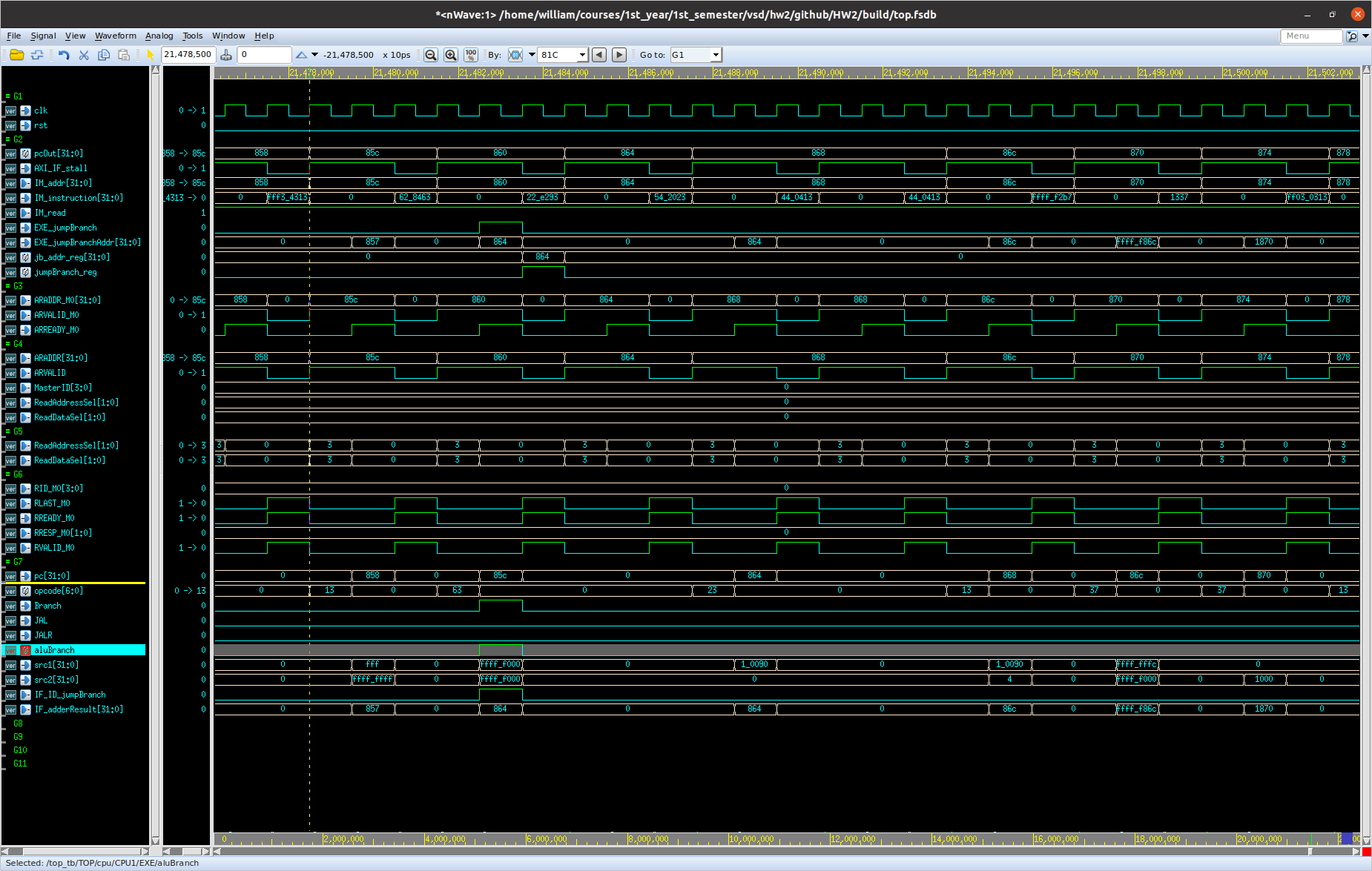


(圖) S-TYPE指令: SH, pc = 0x7F4



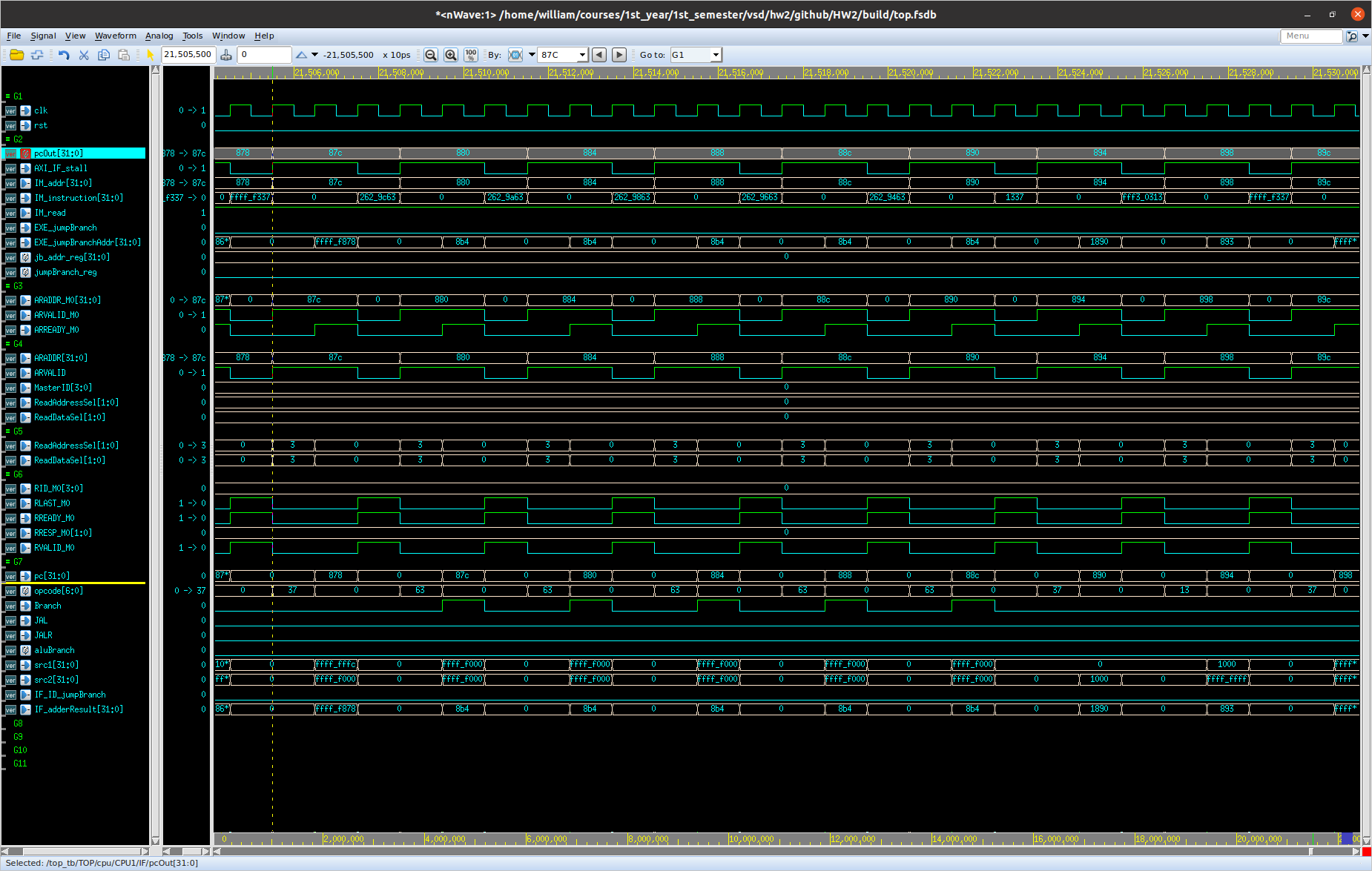
(圖) S-TYPE指令: SB, pc = 0x7E8

接下來介紹Branch指令的機制， 下圖為Branch指令波形圖。

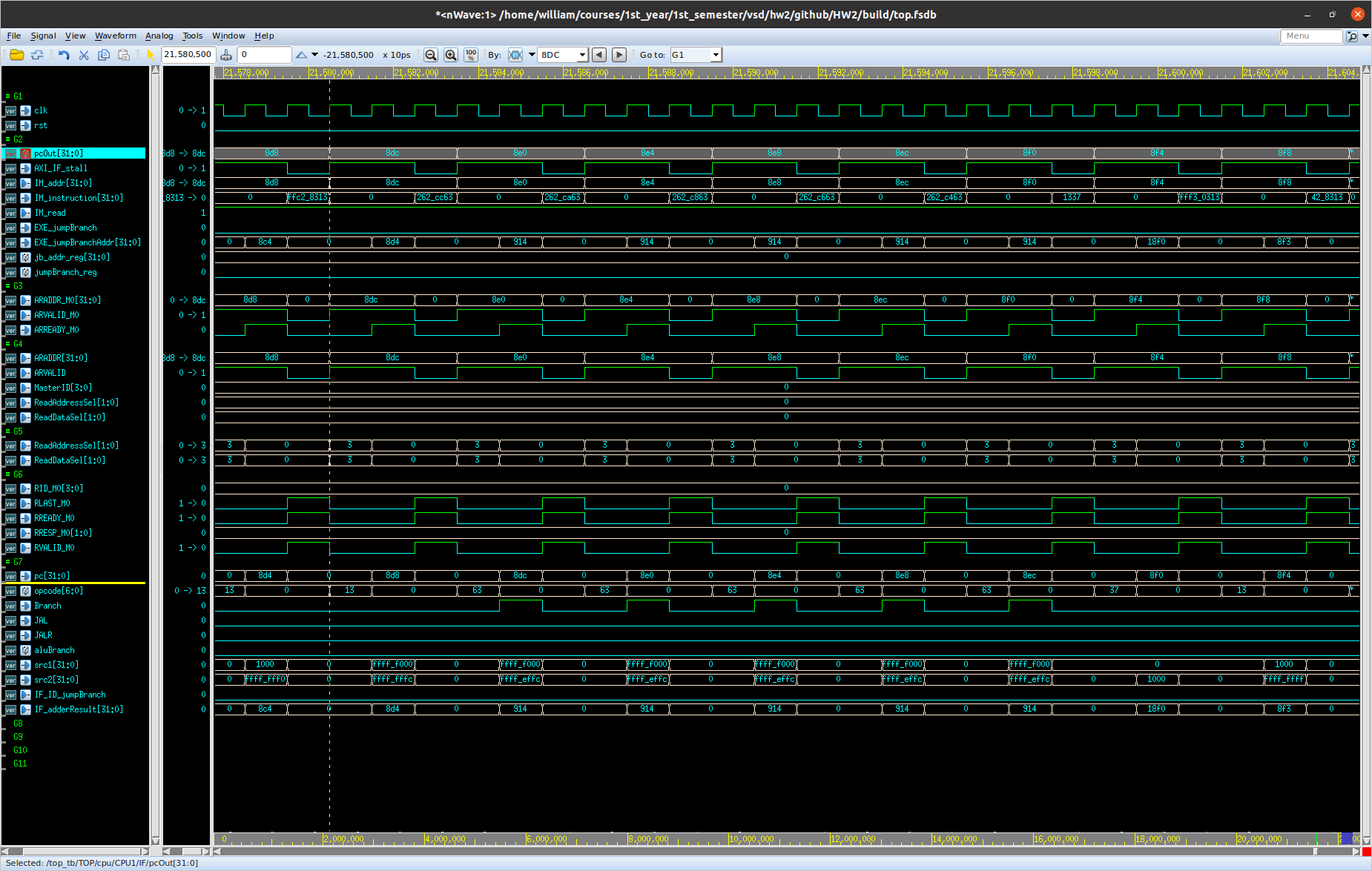


(圖) B-TYPE指令: BEQ, pc = 0x85C

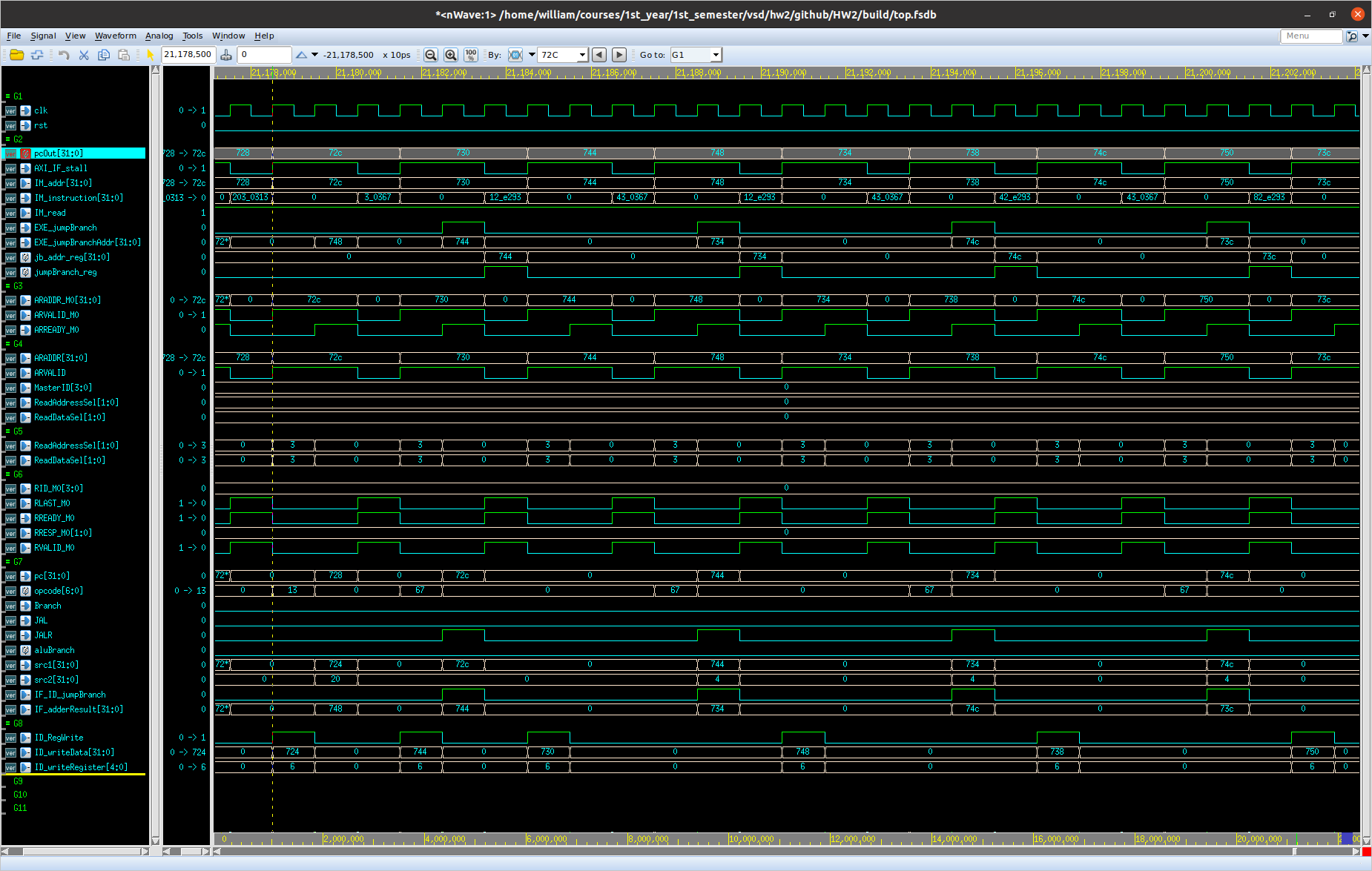
本次作業的CPU是在EXE stage判斷Branch結果，由於一個指令的request會delay 3個cycle，當EXE判斷出要進行branch時，剛好新的request正在進行中，因此必須存住要跳去的地址和要branch的訊號，稱為jump\_branch\_reg和jb\_addr\_reg。當錯誤的request完成，branch的地址會被寫入pc，而錯誤的指令會被遮蓋掉。



(圖) B-TYPE指令: BNE, pc = 0x87C



(圖) B-TYPE指令: BLT, pc = 0x8DC

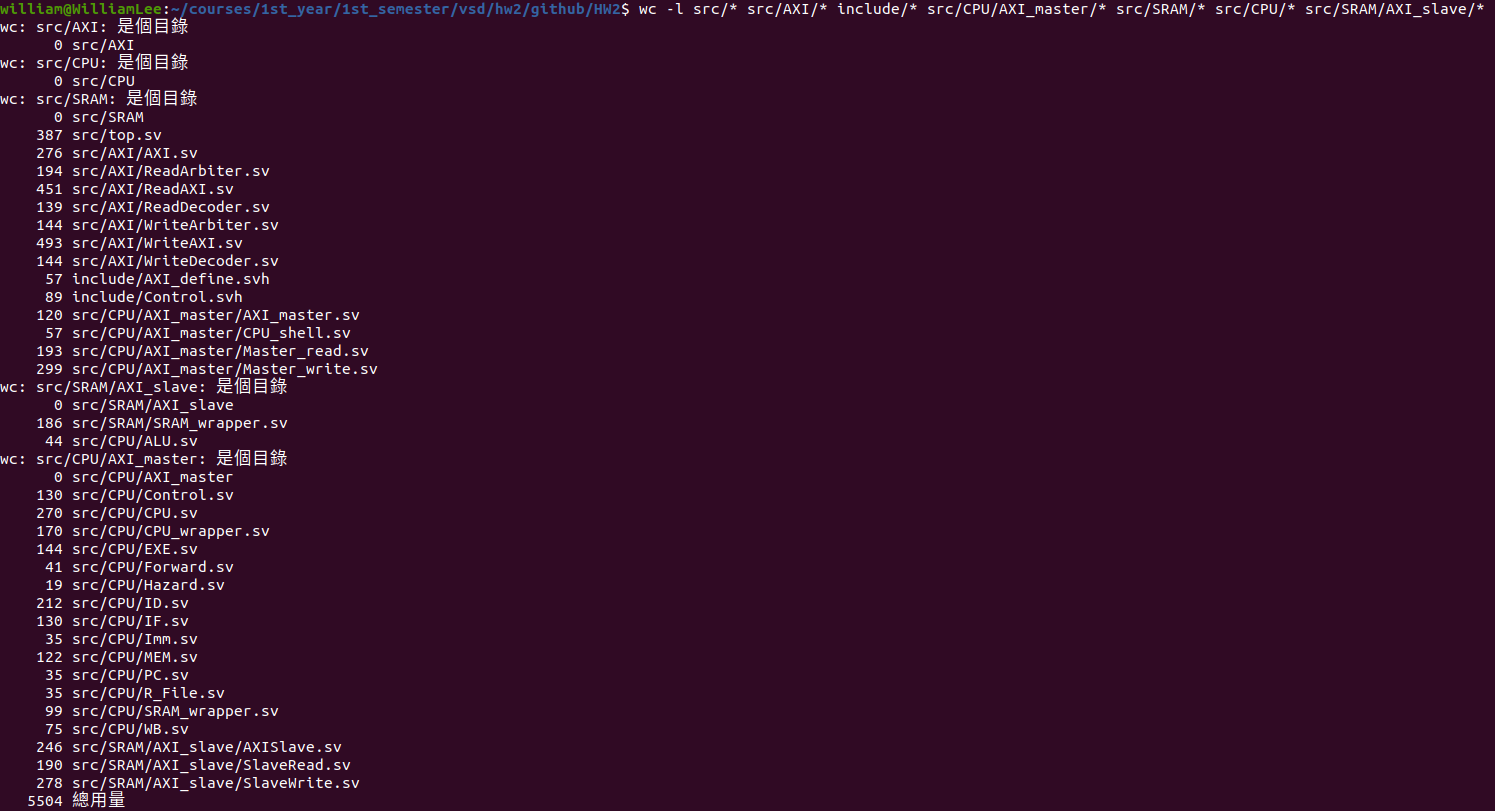


(圖) J-TYPE指令: JALR, pc = 0x72C

# Superlint

Overall Report：

一張含有 文字 的圖片

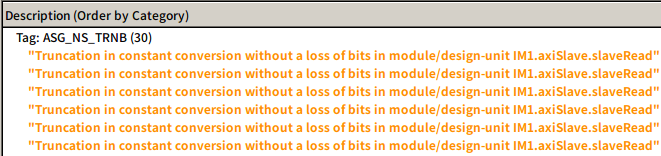
自動產生的描述

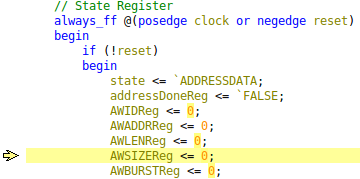
Total Violation = 86

Code Length = 5504

Score = 98.4%

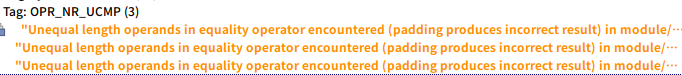
Violation1：ASG\_NS\_TRNB

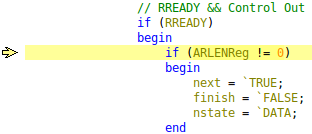




AWIDReg <= 0; should be AWIDReg <= 1’b0;

Violation2：OPR\_NR\_UCMP



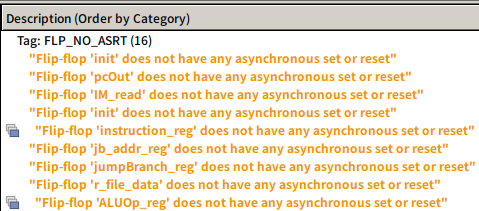


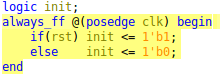
ARLENReg != 0 Should be

ARLENReg != 1’b0

AI != 5’d0

Violation3：FLP\_NO\_ASRT

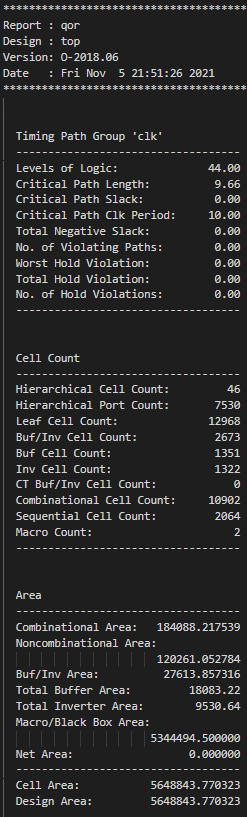




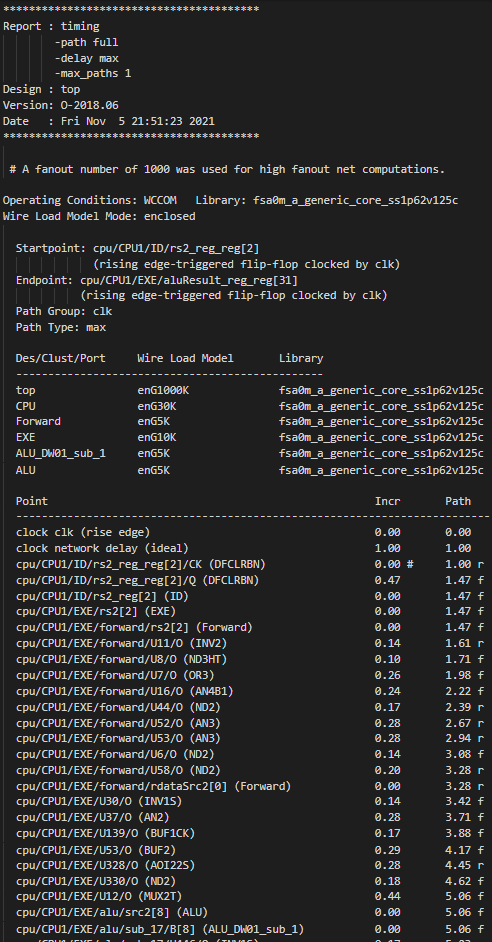
always\_ff @(posedge clock) should be always\_ff @(posedge clock or posedge reset)

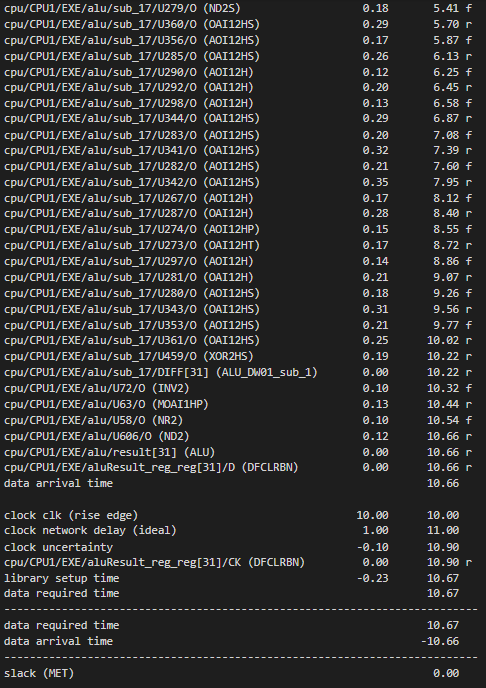
# Synthesis Report

1. QOR (Quality Of Results)



1. Timing





# Lessons learned

李秉軒：

相比上次的pipeline CPU，這次的AXI是完全沒有任何背景知識的開始，所以一開始相當掙扎，相比上次作業一直打Code，這次花比較多時間在想，所幸我的組員十分Carry，把許多事前要考慮的部分都考慮得很清楚。這次除了學習到使用AXI溝通之外，也對於整個系統Memory的定址有了一些概念。

劉彥麟：

我透過這次的練習更了解AXI這個Protocal，並且學到了如何使用ABVIP來測驗自己的Bridge等Module，也讓我發現ABVIP對於皆線錯誤沒有進行檢測，但是最後都有順利的解出所有的Bug讓Cover全部都通過，感謝助教和同學熱心的幫忙。