VLSI System Design (Graduate Level)

Fall 2023

HOMEWORK III

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: \_\_陳燦仁\_ , \_\_陳彥佑\_

Student ID: P76121631 , P76121411

**Summary**

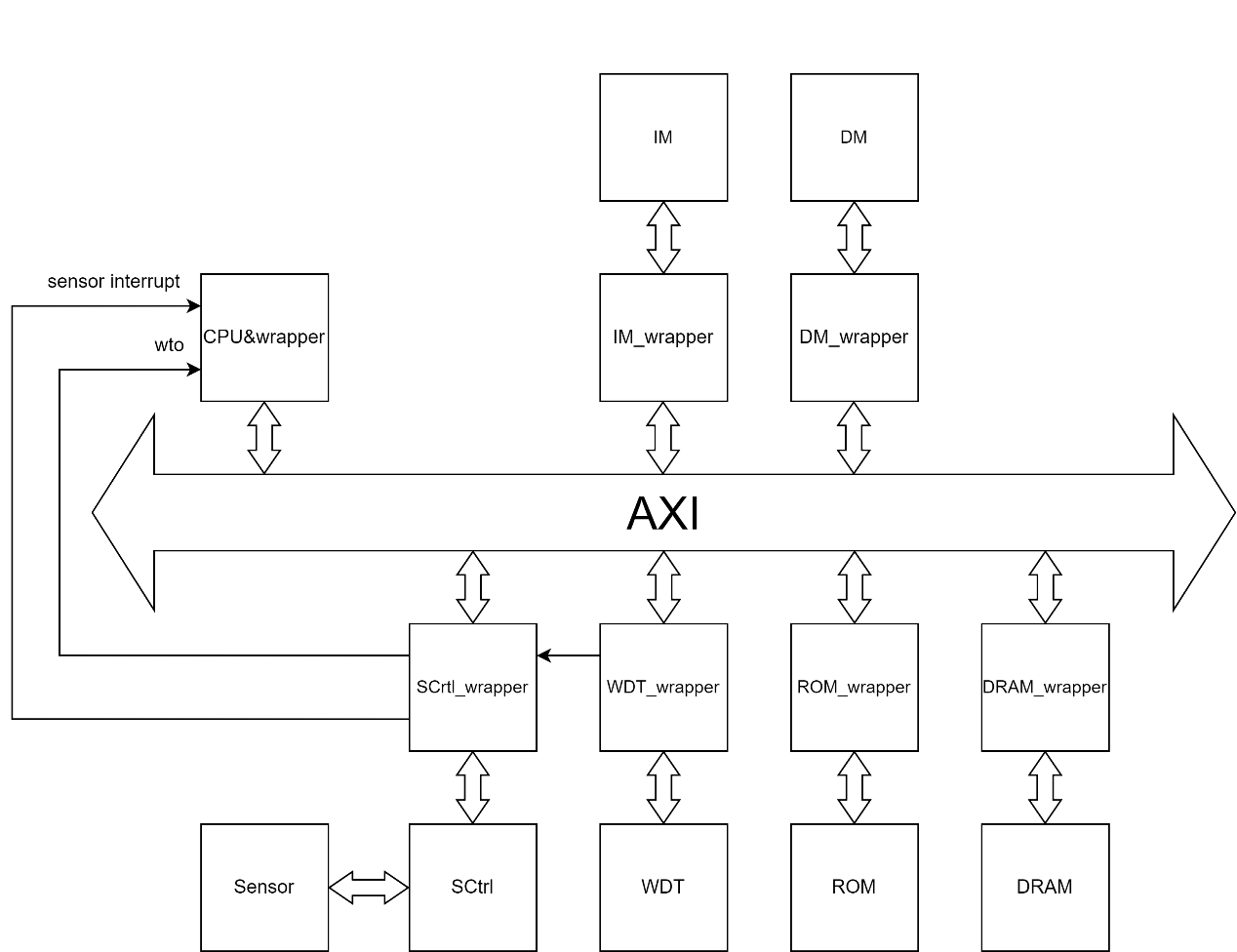
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hardware | | | | | | | | |
|  | | | | | | RTL | | synthesis |
| Top | CPU\_wrapper | | CPU | | |  | |  |
| New instructions | | |  | |  |
| SRAM\_wrapper (IM & DM) | | | | |  | |  |
| ROM\_wrapper | | | | |  | |  |
| DRAM\_wrapper | | | | |  | |  |
| AXI | | | | |  | |  |
| Sensor control wrapper | | | | |  | |  |
| Watch Dog Timer | | | | |  | |  |
| Synthesis result | | | | | | | | |
| Area | | | | | Clock cycle(ns) | | | |
| Total : 6059386.246540 | | | | | 18 | | | |
| Firmware & Software | | | | | | | | |
|  | | RTL pass | | syn pass | | | Execution time(ns) | |
| Booting | |  | |  | | | - | |
| Prog 0 | |  | |  | | | 2241300-2241300 | |
| Prog 1 | |  | |  | | | 907134-9071340 | |
| Prog 2 | |  | |  | | | 6838382000-68383820000 | |
| Prog 3 | |  | |  | | | 2681908-26819080 | |
| Prog 4 | |  | |  | | | 1168940-11689400 | |
| Prog 5 | |  | |  | | | 11685014-14270540 | |
| Spyglass summary(number of inline messages) | | | | | | | | |
| Information | | Warning | | Error | | | Fatal | |
| 137 | | 7 | | 1 | | | 0 | |
| Superlint(number of inline messages) | | | | | | | | |
| Total lines | | Warning | | Error | | | coverage(%) | |
| 7225 | | 96 | | 0 | | | 99.987% | |

**Contribution**

|  |  |
| --- | --- |
| XXX 50% | XXX 50% |
| 陳燦仁 50% | 陳彥佑 50% |

**Hardware Design Description**

* System Block Diagram



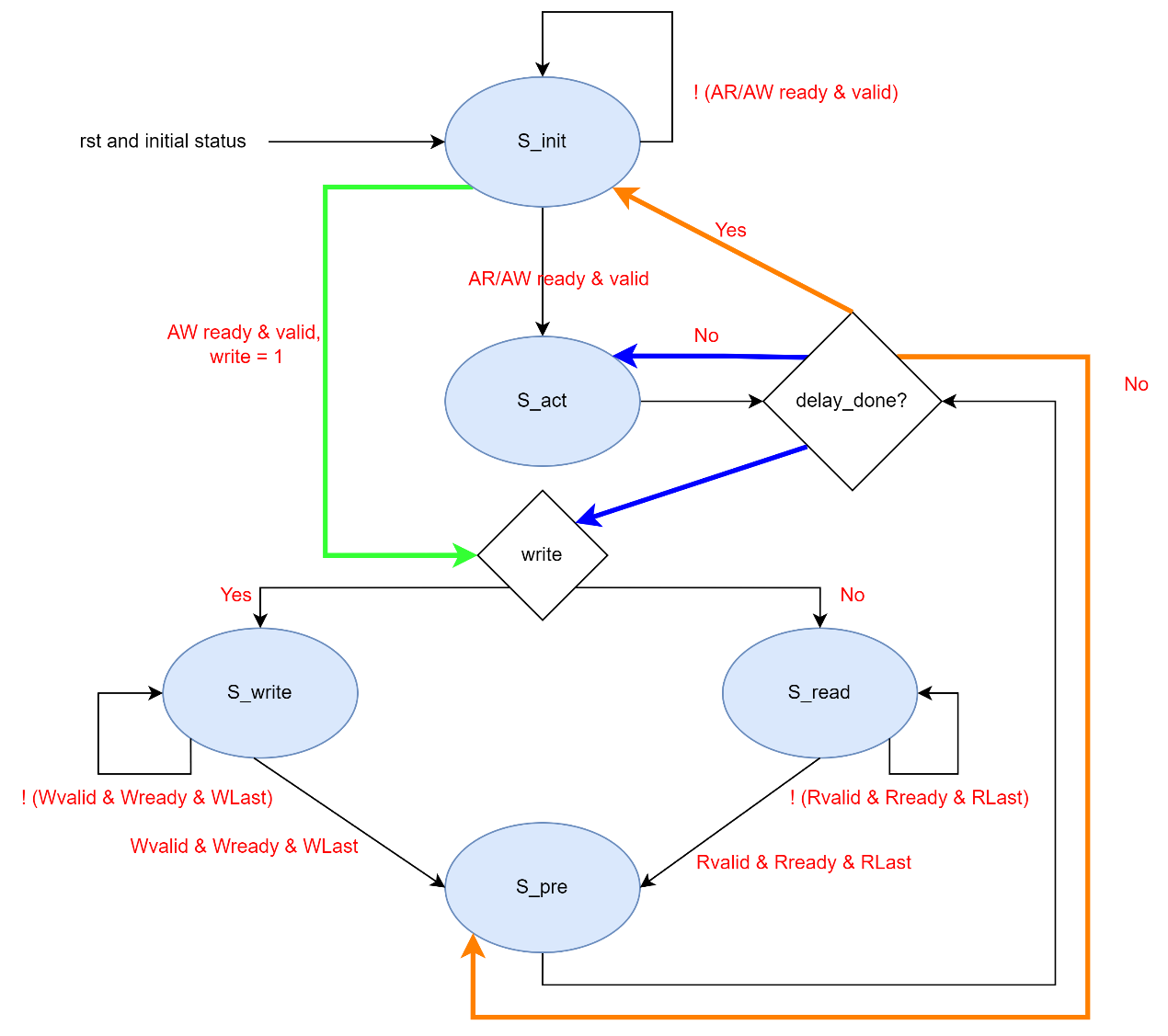
上圖是這次HW3的整個系統架構，這次主要新增了WDT及其Wrapper、Sensor Control及其Wrapper以及CPU中新增了CSR。另外WDT以及SCtrl分別將其output直接接上CPU\_Wrapper，而非不經過AXI。並修改其餘相關部分程式碼，在此不予贅述。

* Interrupt mechanism description and flow chart



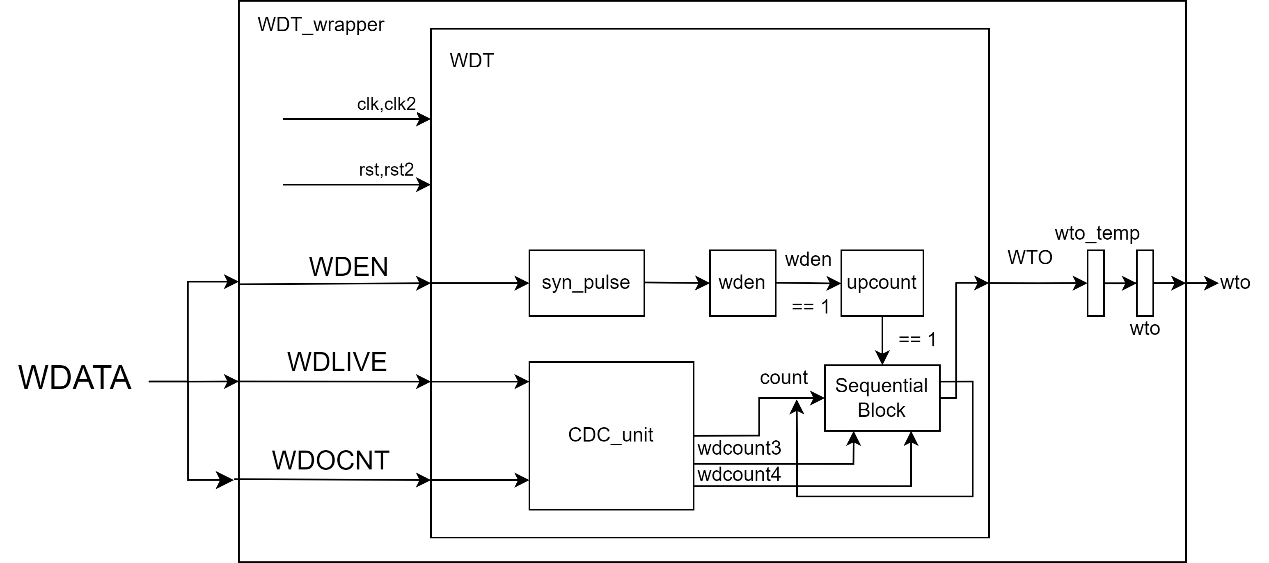
延續之前的作業部分，CSR我們是放在EXE中，因為會需要使用到CSR的暫存器與rs1做運算，以及將相關的值存進rd中。另外在最易開始做Branch判斷的地方也是EXE，所以最終把CSR放置於EXE中。

由上圖可知CSR主要由CSR Register作為主導，對於Interrupt來說，首先就是對於其中數值進行CSR運算，接著發出WFI做pc+4以及根據mie對mip相對應位置做賦值，並且將csrstall pull HIGH。接著收到Interrupt之後，再將csrstall pull LOW 並根據mip對mstatus相對應位置做賦值，之後pc就會到ISR位址做中斷處理。最後在用MERT來return原本的pc繼續做未完成的事。

* DRAM wrapper FSM chart

初始狀態設為S\_init，當AWvalid和AWready為1時拉起write訊號，或是ARready和ARvalid為1的話會進到S\_act，若是delay已經完成了就會依照write訊號是否為1來決定要進入S\_write或是S\_ready，在此二狀態會進行讀取跟寫入，當RLAST和WLAST都為1時就會進入S\_pre，若此時delay完成的話就會回到S\_init否則待在S\_pre。

* WDT & CDC circuit description and diagram

****

WDT\_Wrapper與slave\_wrapper相似，當0x10010100進來時會將WDEN=1傳入WDT; 當0x10010200進來時會將WDLIVE=1傳入WDT; 當0x10010300進來時會將WTOCNT傳入WDT，以下分別概述WDT是如何將此三種訊號做CDC的處理。

**WDEN**我們先使用syn\_pluse來是使它的持續長度達到 clk2的一個週期，而upcount 是由 WDEN 啟動的時候來變成1 直到你有輸出WTO 才會變成0。

**WDLIVE**因為此訊號是一個pulse 所以使用了toggle flop將pulse轉變成level就可以進行CDC的處理，在output 端再使用XOR和一個flop來重新轉回pulse。

**WTOCNT**通常多bit的訊號都會在clk內產生一個load 訊號，當load為high才把多bit的訊號打入clk2，但是這次沒有load訊號，而且WTOCNT又很穩定，所以我在2flop synchronizer後再打兩拍，確定這三級的值是穩定的，再輸出給後面的sequential circuit。

**Software & Firmware design description**

* Prog 1

Insertion Sort.

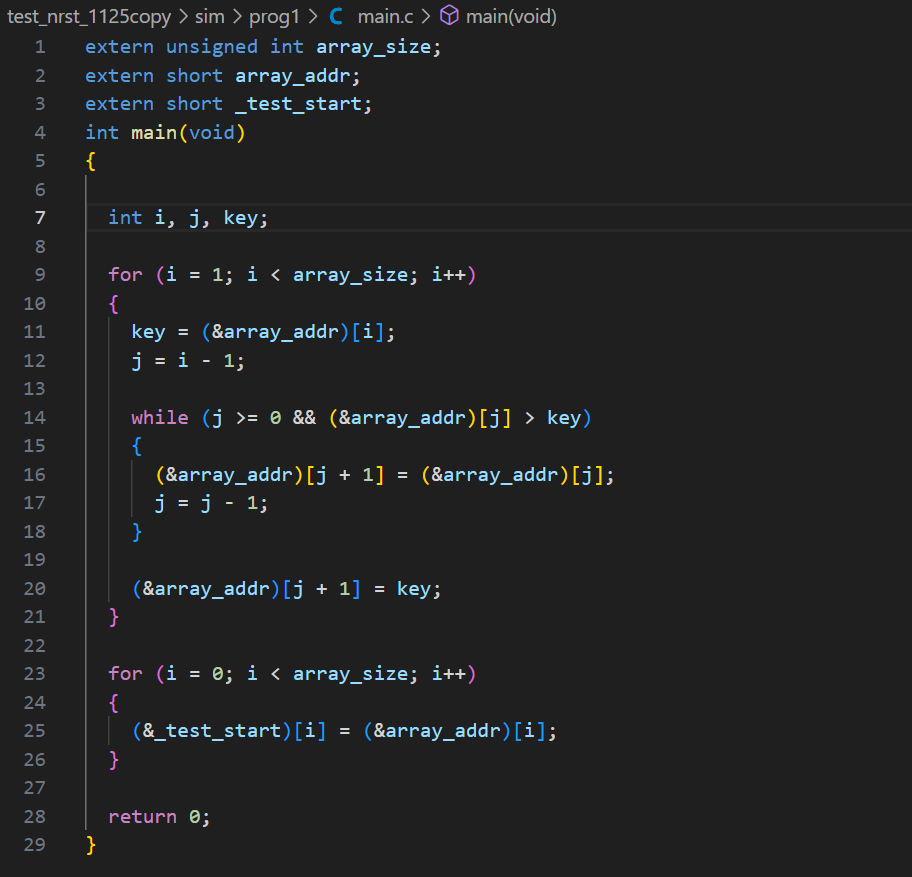
**Pseudo code :**

**For i = 1 to array\_size:**

**j = i-1;**

**if j > 0 and (&array\_data)[j] > (&array\_data)[i] :**

**swap ((&array\_data)[j], (&array\_data)[i]);**



簡單來說我們就是運用了 insertion sort來完成排序。我們透過將 array\_addr 把當中的值提取出來當作 key。再透過 while來將剩下的值進行排序。最後再將排序完的data搬到 \_test\_start。

* Prog 2

灰階處理

**Pseudo code :**

**For j = 54 to &\_binary\_image\_bmp\_size, j = j+3:**

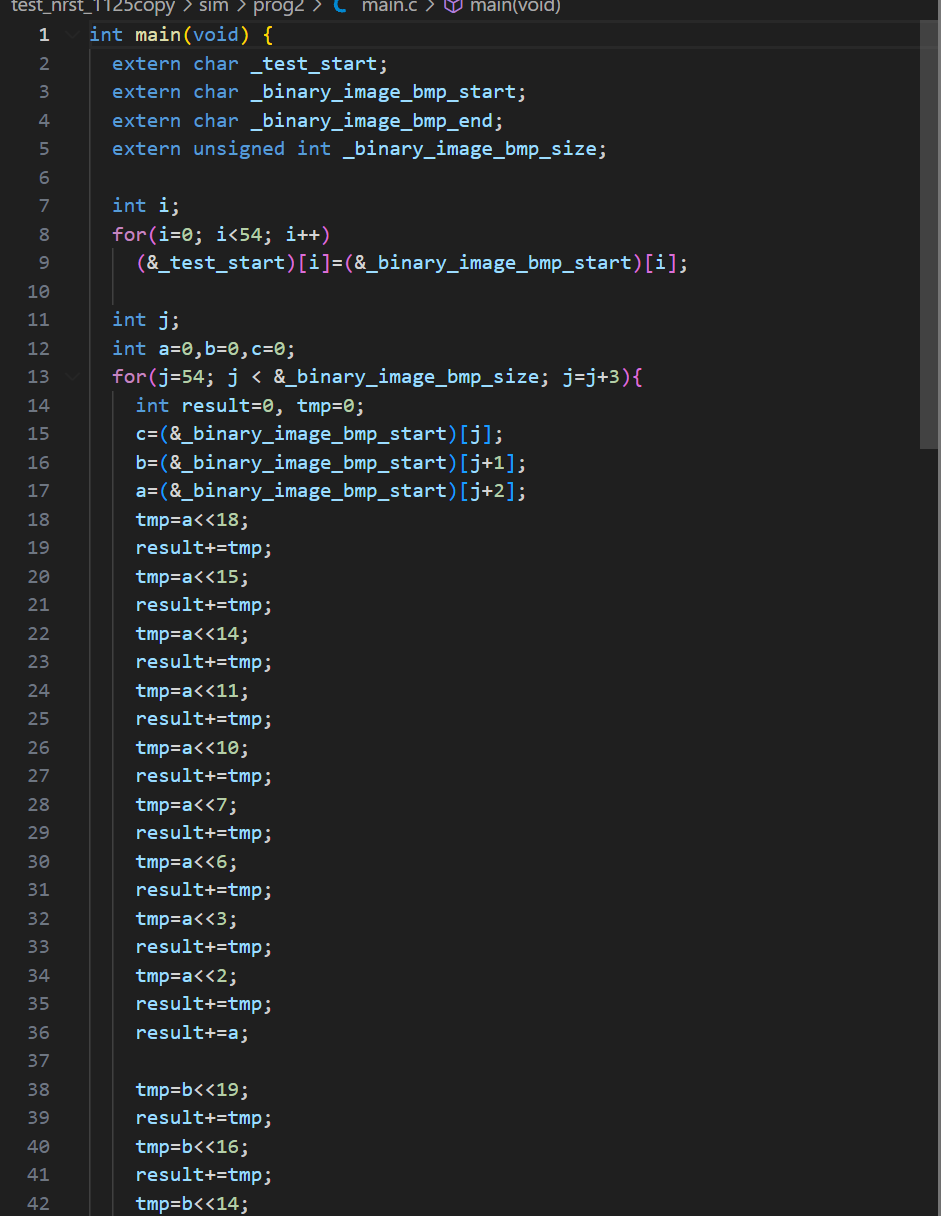
**c=(&\_binary\_image\_bmp\_start)[j];**

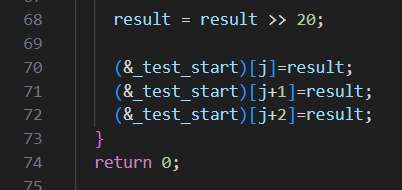
**b=(&\_binary\_image\_bmp\_start)[j+1];**

**a=(&\_binary\_image\_bmp\_start)[j+2];**

**tmp=a<<位移量;tmp1=b<<位移量;tmp2=c<<位移量**

**result+=tmp,tmp1,tmp2;**





我們將bit 3bit3bit進行相加後的操作。並且進行最終的操作。然後再進行一個偏移的動作。並且將data放在 \_test\_start中。

* Booting

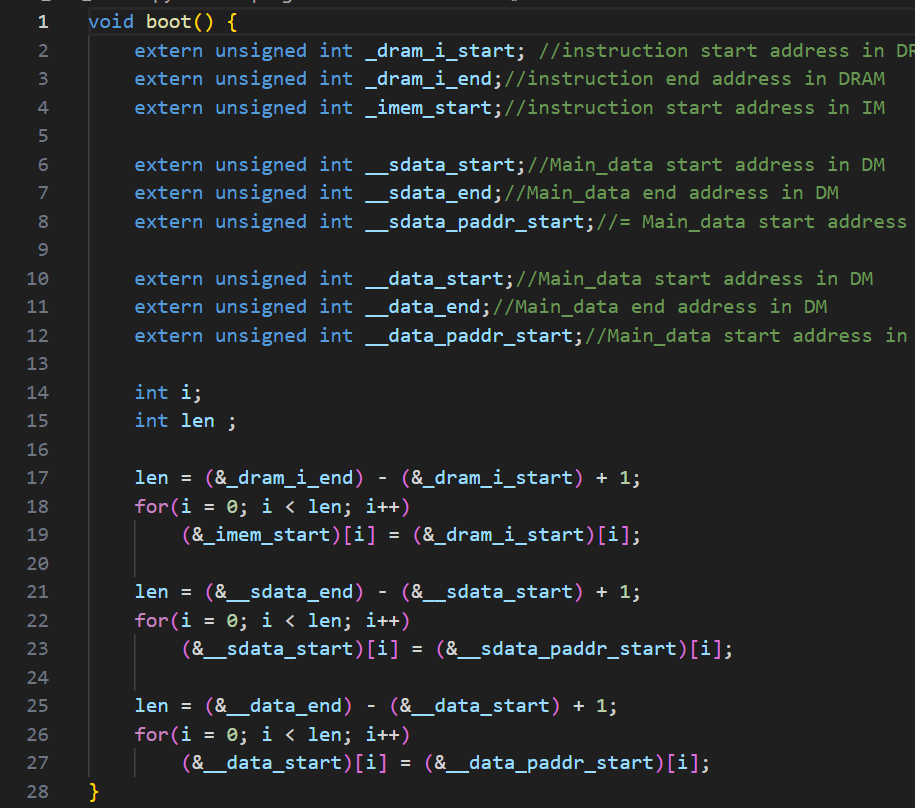
在程式執行之前，會需要藉由boot.c來將DRAM中的data搬移到SRAM給我們做指令讀取或是做資料讀寫。一開始boot.c會存在ROM之中，我們可以觀察到指令都是放置於\_dram\_i\_end ~ \_dram\_i\_start之間，並且將這些指令都放到\_imem\_start之後才能正確讀取指令並執行。Data則被放置到\_sdata\_end-\_sdata\_start 以及 \_data\_end-\_data\_start，同樣的我們需要藉由boot來將這兩處data分別放置於\_\_sdata\_paddr\_start 以及 \_\_data\_paddr\_start 之後。所以我們可以使用迴圈來將作資料搬移。

**Pseudo code :**

**For i = 0 to len(DRAM\_instructions):**

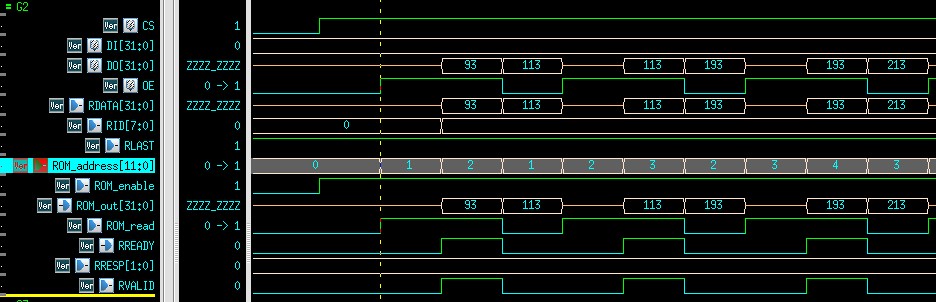
**&instr\_mem\_start[i] = & instre\_DRAM\_start[i];**

**// The “instr” can be replace with “data” or “sdata”**



**Screen shot of wave forms and simulation results**

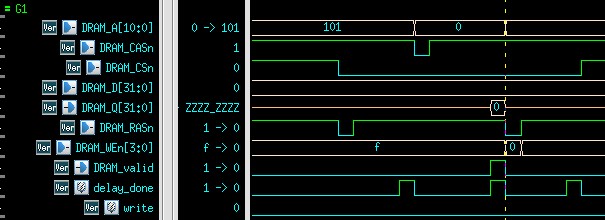
**ROM operation**

****

**我們可以看到當 ROM 紅色框框當中的 OE 1被拉高的時候代表 output enable，那此時可以發現 ROM\_address 跟 下一個的 ROM\_out開始有值，可以看到這是一個明顯的ROM相關操作。**

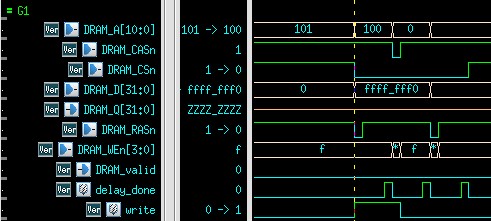
**DRAM operation**

**DRAM READ:**

****

**當 DRAM 做 READ 的時候，我們可以發現DRAM valid 並 DRAM\_Q拉起，並且RASn使的 DRAM\_Wen拉為0，此後將DATA輸出。**

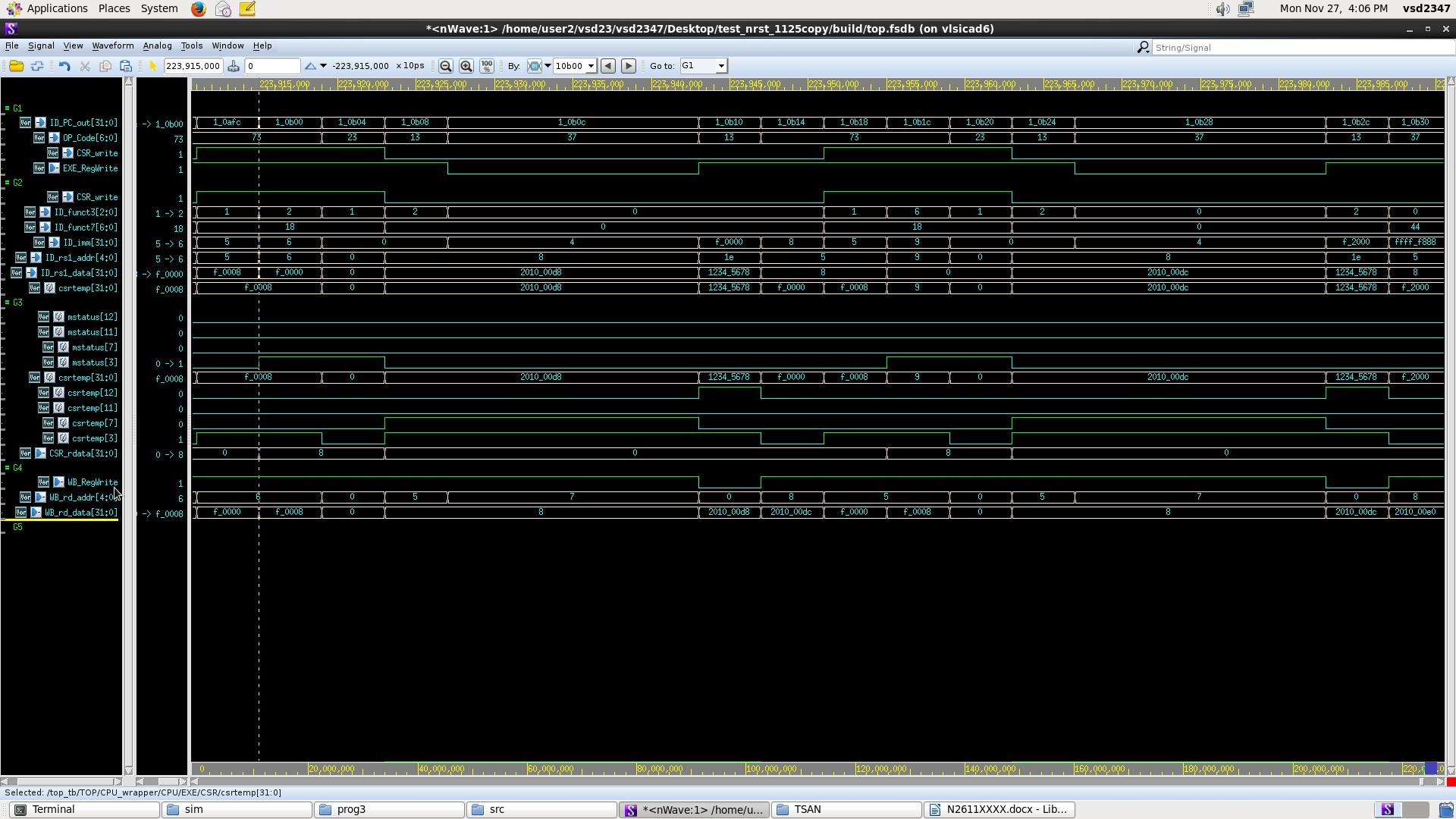
**DRAM WRITE:**

****

**當write 拉高時，我們會依照 DRAM\_A的 address 去相對應的位置去寫入值，而寫入的值是由 DRAM\_D 給傳進 DRAM對應的位置上。**

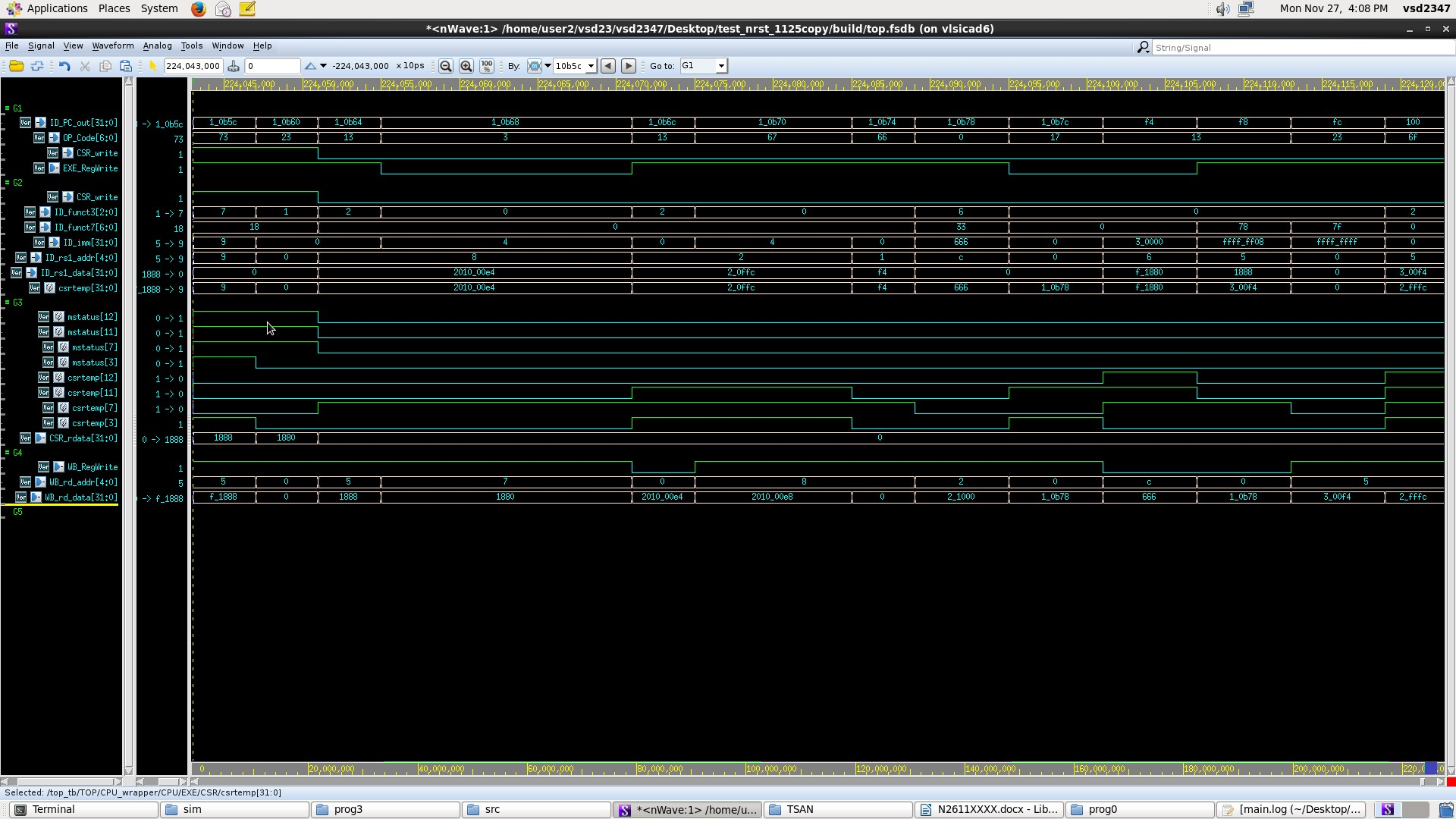
**CSR operation**

**CSRRS**



**我們可以詳細的看到，我們的 ID\_PC\_out=1\_0afc，這邊對應到的指令就為 Csrrs t0,mstatue,t1。所以我們可以在紅色框框的地方看到 OP\_Code 為 0x73=7’b1110011，和得到CSR 動作得時候將CSR\_write 拉高成 High。然後你可以發現在藍色框框確認到，csrtemp=rs1=0xf\_0008。並且就可以在黃色框框的地方發現 CSR\_rdata=mstatus[w]|csrtemp[w]，w=3,7,11,12。最後綠色框框的地方可以發現在最後的結果 8 給寫回去reg當中。此就完成了CSRRS。**

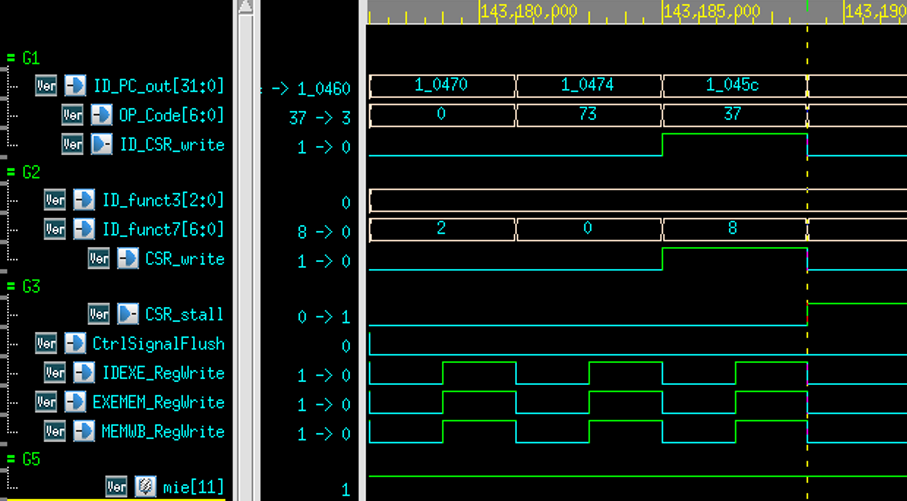
**CSRRSI**



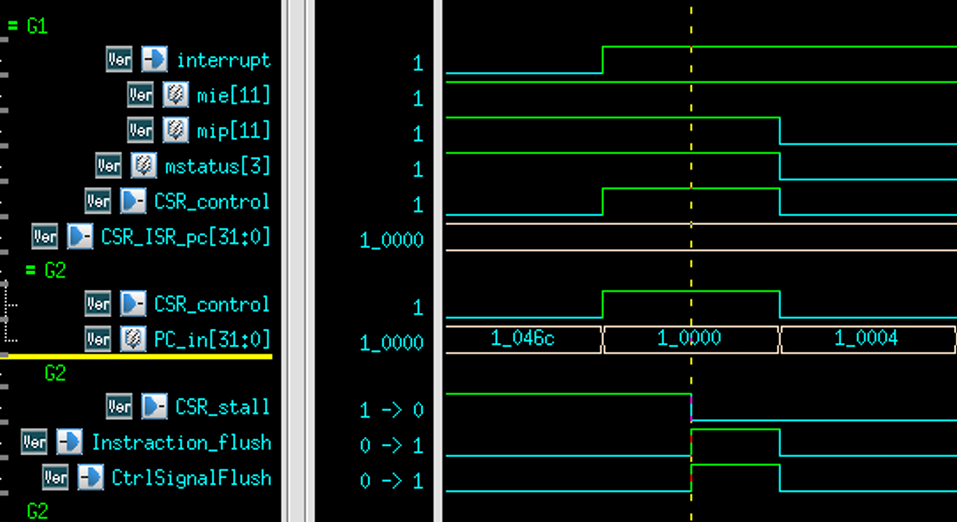
**一開始 ID\_PC\_out=0x1\_0b5c，我們可以知道指令為 csrrsi t0,mstatue,9。所以當我們可以看到紅色框框中的OP\_Code為0x73=7’b1110011，和得到CSR\_write訊號被拉高的訊號。就可以看到因為黃色框框的方要看的東西為**

**CSR\_rdata=mstatus[w]&~csrtemp[w]，w=3,7,11,12。在之後可以發現最後的結果WB\_RegWrite是 High 後將結果寫回 reg中。**

**WFI**

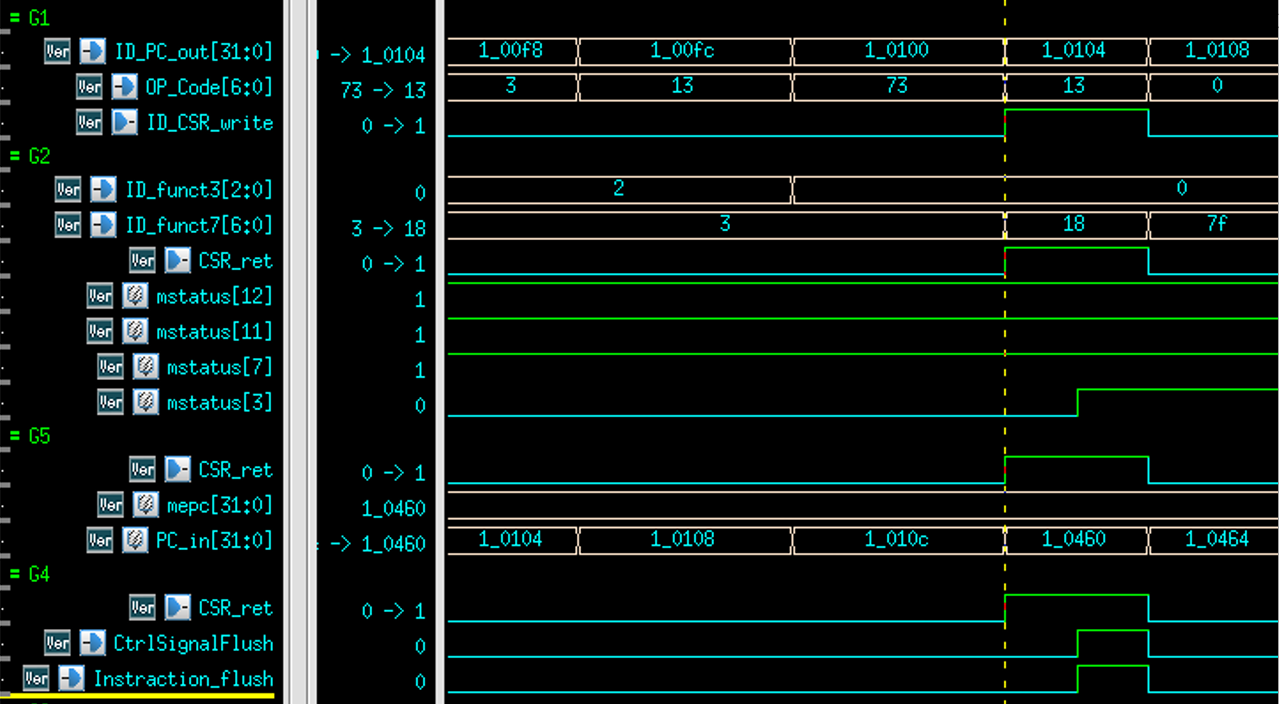
****

**首先ID\_PC\_out = 0x1\_0474，在這一條的對應指令為 wfi。所以我們可以在紅框處看到OP\_code為0x73 = 7’b1110011，以及吃到CSR operation時將csr\_web pull HIGH。接著在圖中藍框處可以發現mie[MEIE] = 1’b1。然後可以在黃框處發現csrstall pull HIGH，且CPU的RegWrite全部pull LOW。**

**Interrupt** 

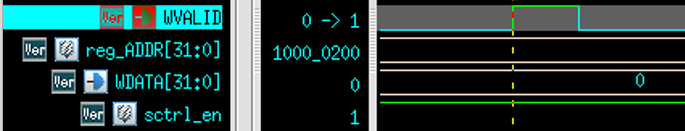
**首先Interrupt pull HIGH。可以在紅框處看到mstatus[MIE] = 1`b1且mie[MEIE] = 1’b1以及mip[MEIP] = 1’b1。接著在圖中藍框處可以發現CSR\_control pull HIGH且IF中PC\_in跳到0x10000開使執行中斷處理。然後可以在黃框處發現CSR\_stall pull Low，且IF、ID的CtrlSignalFlush和Instraction\_Flush全部pull HIGH。**

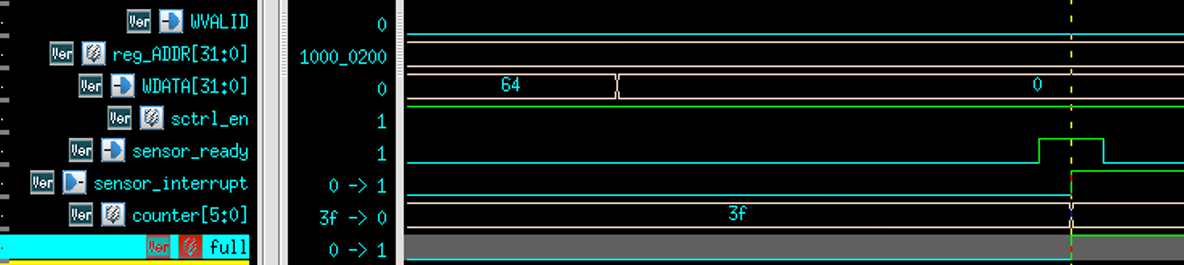
**MERT**

****

**首先PC\_out = 0x1\_0104，在這一條的對應指令為 mret。所以我們可以在紅框處看到OP\_code為0x73 = 7’b1110011，以及CSR operation時將CSR\_write pull HIGH。接著在圖中藍框處可以發現funct3 == 3`b0且funct7[4] == 1`b1，並且將mstatus轉換為Interrupt前的狀態。然後可以在黃框處發現PC\_in 由mepc賦值回去，以執行中斷前未完成指令，並將IF、ID的CtrlSignalFlush和Instraction\_Flush全部pull HIGH。**

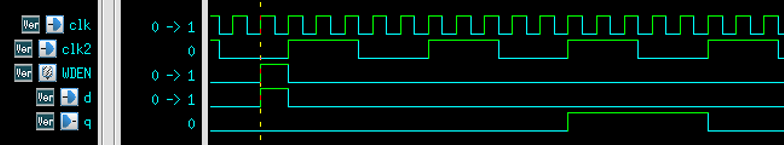
**Sensor Control**

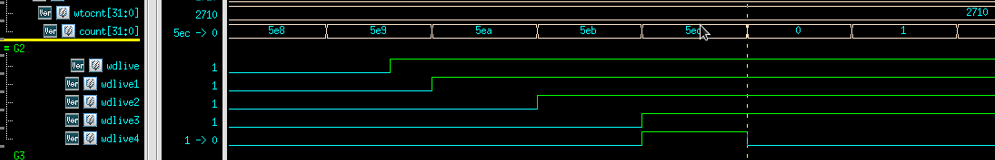
****

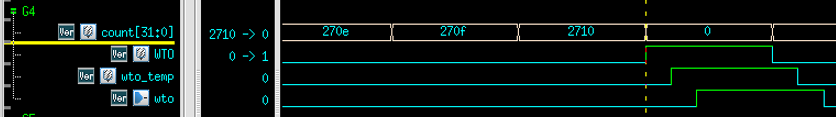
****

**當pull up WVALID時，可以發現圖中紅框處根據reg\_ADDR將sctrl\_en Pull High。接著在藍框可以看到counter 開始累加，直到0x3f = 6`d63時，再根據sctrl\_en、sensor\_ready以及counter = 0x3f來pull full HIGH，最終發出interrupt。**

**WDT**

****

****

****

**從最上方紅框處可見，為clk與clk2不同，所以先把所得WDEN傳給syn\_pulse(d)中將wden(p)的長度從一個clk變成一個clk2的長度。接著在兩個黃框處可以發現，我們通過延遲3拍來確保不會發生亞穩定態。另外可以在藍框處發現當wdlive4 pull HIGH 時，count小於Timeout的數值時(5ec < 2710)，將count歸0。最後可以在綠框處發現，當count大於等於Timeout的數值時，將WTO pull HIGH，並先延遲1拍傳輸給temp，在發出timeout 給CPU。**

**RTL/SYN**

**Prog0 Prog1**

RTL

SYN

**一張含有 文字, 收據, 字型, 圖表 的圖片

自動產生的描述一張含有 文字, 收據, 螢幕擷取畫面, 字型 的圖片

自動產生的描述**

**一張含有 文字, 收據, 螢幕擷取畫面, 字型 的圖片

自動產生的描述一張含有 文字, 收據, 圖表, 字型 的圖片

自動產生的描述**

**Prog2 Prog3**

RTL

SYN

**一張含有 文字, 收據, 螢幕擷取畫面, 字型 的圖片

自動產生的描述一張含有 文字, 螢幕擷取畫面, 字型, 圖表 的圖片

自動產生的描述**

**一張含有 文字, 螢幕擷取畫面, 字型, 收據 的圖片

自動產生的描述一張含有 文字, 收據, 字型, 圖表 的圖片

自動產生的描述**

**Prog4 Prog5**

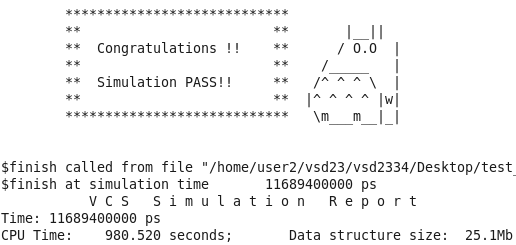
RTL

SYN

**一張含有 文字, 收據, 螢幕擷取畫面, 字型 的圖片

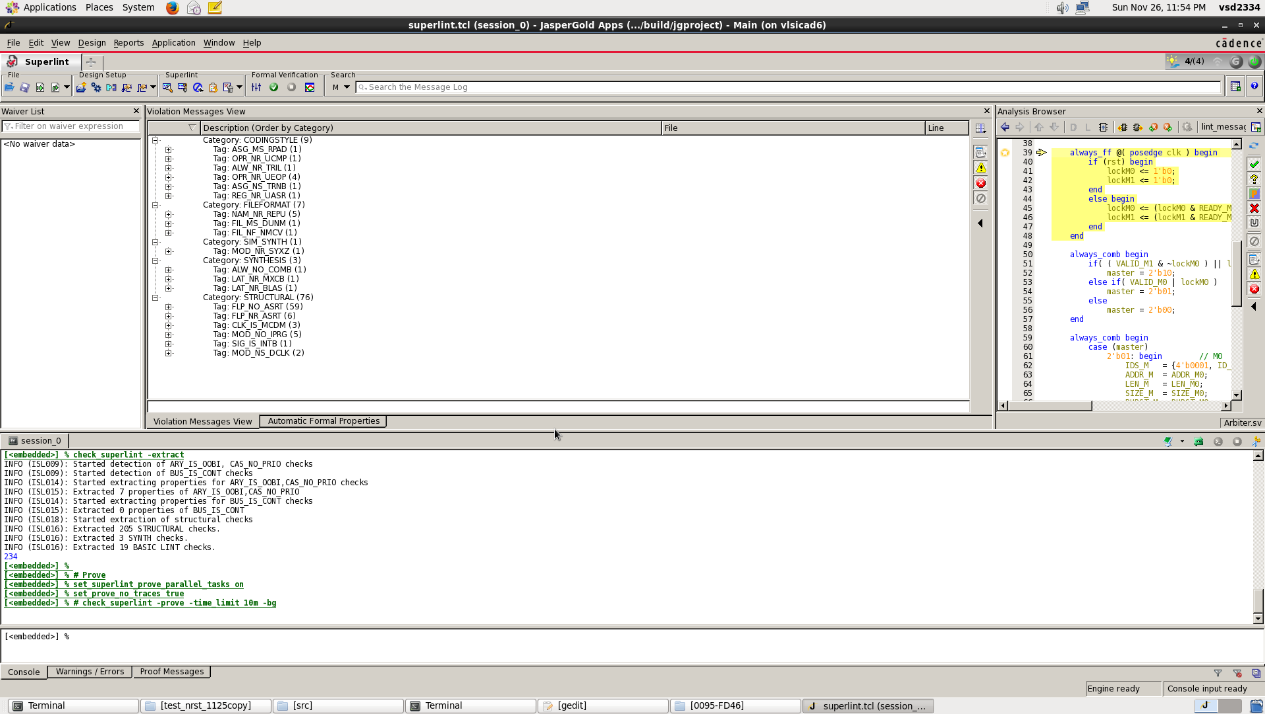
自動產生的描述一張含有 文字, 收據, 螢幕擷取畫面, 字型 的圖片

自動產生的描述**

**一張含有 文字, 收據, 字型, 螢幕擷取畫面 的圖片

自動產生的描述**

**Superlint :**



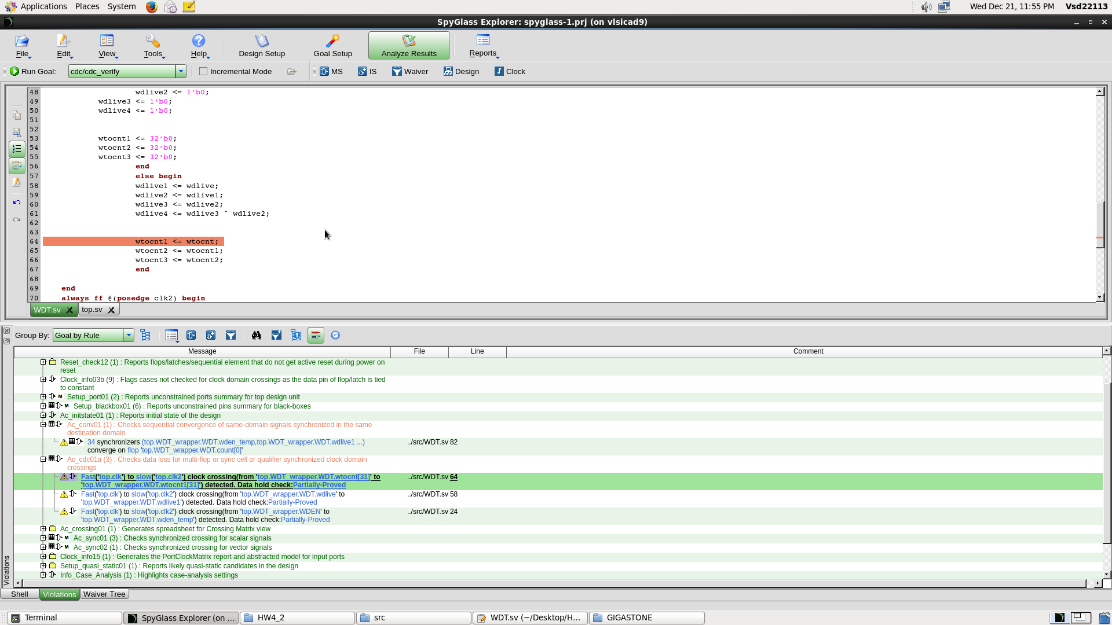
**我們的superlint總共為(9+7+1+3+76)/7225=0.013 -> correct rate = 99.987%**

**主要遇到的問題為助教給的 module 就有的warning，另外的warning大部分為不同bit相加的整數。例如 7bit+3bit的數字，但因為是用整數加法，所以我們選擇性忽略掉這個問題。**

**Spyglass :**

一張含有 文字, 螢幕擷取畫面, 陳列, 軟體 的圖片

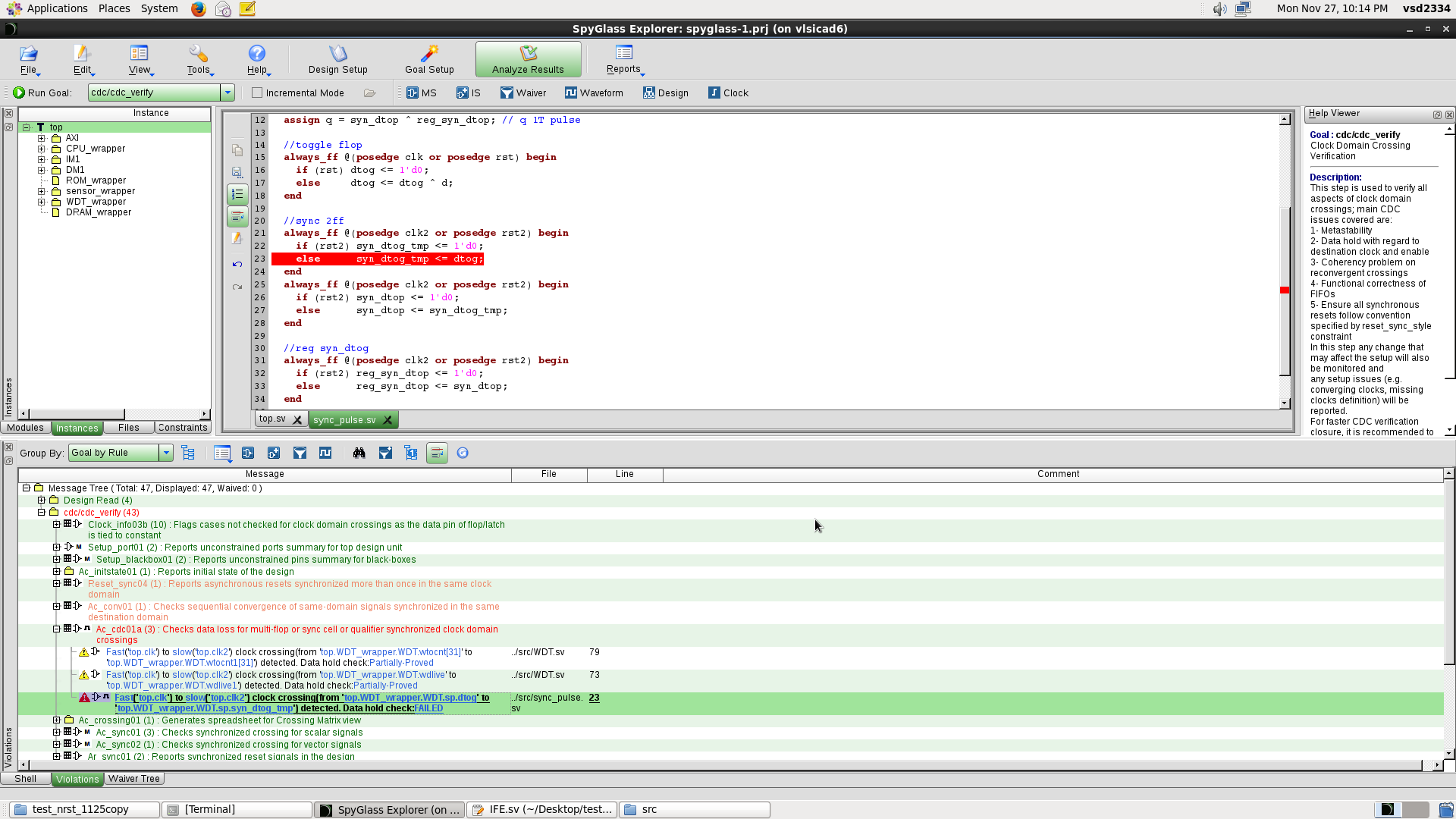
自動產生的描述

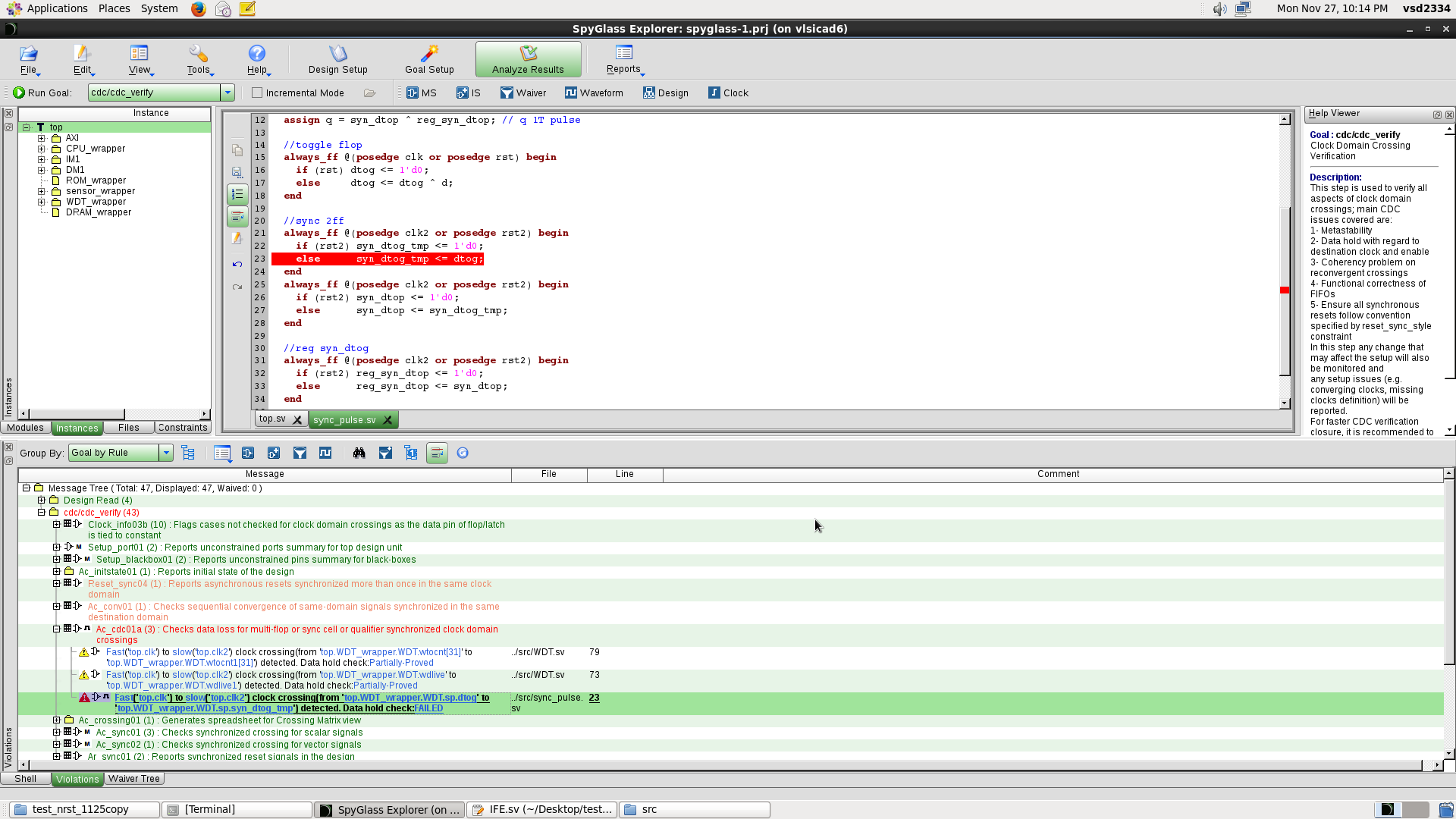
****

**WARNING: 7**

**我們可以看到我是用4個flip-flop來做得，所以 flip-flop之間會有 partially-prove的問題。這個問題的原因是因為若 wtocnt常常進行變換的話，那 flip-flop間的值會不一樣，就會造成一個 data-chold check的現象而產生一個 warning，但我去看了這次的測資發現並不會有這個問題，所以不影響我們的系統。**

**ERROR: 1**

****

****

**我們可以看到我們的 error 是 Ac\_cdc01a ，經過尋找和查看提供的 CDC pdf後，我們可以知道是因為這邊是因為把某個值從 clk 給過渡到 clk2 長度的原因，他怕會導致 data loss，但本身我們這個程式的功能就是要將這個訊號的長度延伸並過渡到 clk2 的 clock rate，所以產生這個問題。所以透過此error並不會影響到我們的程式。**

**Problems to answer**

1. What is the deference between mcycle and timer? When is mcycle used?

ANS:

mcycle : 算CPU重設後共運行了幾個cycle。

timer : 算CPU重設後共運行了多少時間，timer使用的是已知的固定頻率時鐘，eg.32768Hz的clock。

mcycle統計的是CPU周期數，驅動mcycle的是CPU的核心時鐘，核心clock可能是動態調整的，例如繁忙狀態下可以把頻率調高，空閒的話就調低，所以依據mcycle是無法確定CPU總共運行了多少時間的。

而驅動timer的是一個有固定頻率的clock，所以可以用來確定(算出)CPU運行了多少時間。

1. What is “Potential Qualifier” in Spyglass?

ANS:

Potential qualifiers 如果存在的話，會是一個 debug hint for unsynchronized crossing

**Lesson learned**

這次的作業主要是讓我們了解到從 ROM、DRAM等data搬運和相關的 addr處理，和怎麼運用到所謂的 boot.c去做出開機時的 data搬運，我覺得這些都是不錯的相關知識。那這次作業我覺得主要是CDC(WDT)和CSR相關的運用，這些看起來雖然是一些簡單的東西，但你如果仔細去思考的話，會發現比想像中的複雜許多。CSR要思考在甚麼的狀況下要進行 CSR stall，CSR control，CSR return，CSR reset，並且參考所給的 mstatus 相關的 bit，如:MPP，MPIE、MIE、MEIE、MTIE、MTIP。這些 bit 來決定 interrupt的種種操作。CDC我們是運用多個 flip-flop 來進行延遲的操作來讓 clk 的data 在傳送到 clk2 時，不會因為 clk 頻率較快而改變data使得讓 clk2 接收不到正確的訊號，讓船進 clk2 的訊號要保持一段時間史的他可以接收正確的結果，避免錯誤。Spyclass 也是一個問題，很多沒有考慮過的 clk or reset 相關問題也可以從當中進行學習，學到不少東西。