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: \_\_\_\_張正德\_\_ , \_林晉宇\_\_\_\_\_\_\_

Student ID: \_P76121089\_, \_P76121673

**Summary**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hardware | | | | | | | | |
|  | | | | | | RTL | | synthesis |
| Top | CPU\_wrapper | | CPU | | | ˇ | | ˇ |
| New instructions | | | ˇ | | ˇ |
| SRAM\_wrapper (IM & DM) | | | | | ˇ | | ˇ |
| ROM\_wrapper | | | | | ˇ | | ˇ |
| DRAM\_wrapper | | | | | ˇ | | ˇ |
| AXI | | | | | ˇ | | ˇ |
| Sensor control wrapper | | | | | ˇ | | ˇ |
| Watch Dog Timer | | | | | ˇ | | ˇ |
| Synthesis result | | | | | | | | |
| Area | | | | | Clock cycle(ns) | | | |
| 6056577.051878 | | | | | 20 | | | |
| Firmware & Software | | | | | | | | |
|  | | RTL pass | | syn pass | | | Execution time(ns) | |
| Booting | | ˇ | | ˇ | | | ˇ | |
| Prog 0 | | ˇ | | ˇ | | | 1667740 | |
| Prog 1 | | ˇ | | ˇ | | | 8754300 | |
| Prog 2 | | ˇ | | ˇ | | | 66296740 | |
| Prog 3 | | ˇ | | ˇ | | | 26433190 | |
| Prog 4 | | ˇ | | ˇ | | | 10845740 | |
| Prog 5 | | ˇ | | ˇ | | | 10846900 | |
| Spyglass summary(number of inline messages) | | | | | | | | |
| Information | | Warning | | Error | | | Fatal | |
| 105 | | 2 | | 0 | | | 0 | |
| Superlint(number of inline messages) | | | | | | | | |
| Total lines | | Warning | | Error | | | coverage(%) | |
| 5572 | | 106 | | 0 | | | 98.09% | |

**Validation**

**Rtl & Syn**

|  |  |  |
| --- | --- | --- |
|  | rtl | syn |
| Prog0 | 一張含有 文字, 螢幕擷取畫面, 電腦, 陳列 的圖片  自動產生的描述 | 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 |
| Prog1 | 一張含有 文字, 螢幕擷取畫面 的圖片  自動產生的描述 | 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 |
| Prog2 | 一張含有 文字, 螢幕擷取畫面, 電腦, 陳列 的圖片  自動產生的描述 | 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 |
| Prog3 | 一張含有 文字, 螢幕擷取畫面, 多媒體軟體 的圖片  自動產生的描述 | 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 |
| Prog4 | 一張含有 文字, 螢幕擷取畫面 的圖片  自動產生的描述 | 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 |
| Prog5 | 一張含有 文字, 螢幕擷取畫面 的圖片  自動產生的描述 | 一張含有 文字, 螢幕擷取畫面, 軟體, 數字 的圖片  自動產生的描述 |

**Report Area**

一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述

**Report Timing**

一張含有 文字, 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 的圖片

自動產生的描述

**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

**Superlint**

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

**Contribution**

|  |  |
| --- | --- |
| P76121089 | P76121673 |
| 50% | 50% |

**Hardware Design Description**

* System Block Diagram
* Interrupt mechanism description and flow chart
* DRAM wrapper FSM chart
* WDT & CDC circuit description and diagram

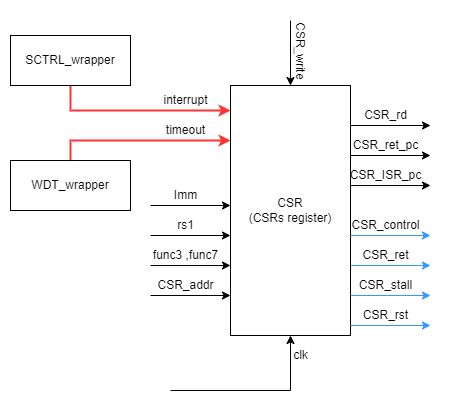
**System Block Diagram**

一張含有 螢幕擷取畫面, 文字, 圖表, 設計 的圖片

自動產生的描述

這次主要新增了WDT及其Wrapper、SCTRL\_wrapper、ROM\_wrapper、RAM\_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(interrupt、timeout)

**Interrupt mechanism description and flow chart**

****

* 我們利用CSR module處理interrupt以及timeout，透過CSR指令設定mstatus、mie、mip、mtvec以及mepc暫存器管理Interrupt。
* 若發生interrupt或是timeout，則發出不同的控制訊號，來管理CPU的運作以及pc值的設定，
  + 若發生interrupt :

1. pc要被assign為CSR\_ISR\_pc，跳到interrupt service routine的位址。
2. CSR\_stall拉高，將CPU暫停。
   * 若發生timeout :
3. pc要被assign為0x0000\_0000，整個系統重新開啟。
4. CSR\_rst拉高，將CPU內所有暫存器清空。

**DRAM wrapper FSM chart**

一張含有 圖表, 圓形, 寫生 的圖片

自動產生的描述

**WDT & CDC circuit description and diagram**

一張含有 文字, 螢幕擷取畫面, 圖表, 設計 的圖片

自動產生的描述

當0x10010100寫入非零值時，會將WDEN=1傳入WDT; 當0x10010200寫入非零值時會將WDLIVE=1傳入WDT; 當0x10010300寫入非零值時會將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**與**WTO**是一個持續性的訊號，所以選擇採用最簡單的2flop synchronizer連接，來同步訊號。

**WDLIVE**因為此訊號是一個single cycle的 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計算。

**Software & Firmware design description**

* Prog 1

使用insertion sort，並將指定陣列按照ascending排序。

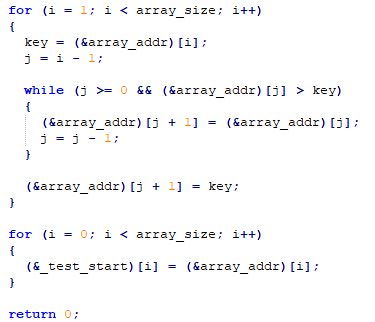
宣告外面變數 `array\_size` 表示陣列大小

宣告外部變數 `array\_addr` 表示陣列起始位址

宣告外部變數 `\_test\_start` 表示排序後陣列的儲存位址。



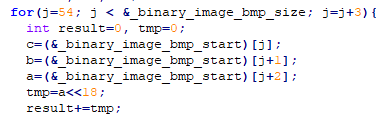
再利用for loop 進行insertion sort，最後存將結果的array依序存進test位址中。



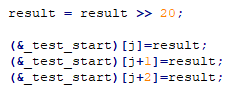
* Prog 2
  + 先將bmp中的header搬入test的開頭



* + 再將BGR資料分別取出，接著個別計算後再加總



* + 最後再將結果存回test中



* Booting

在main.c執行前，藉由boot.c來將DRAM中的資料搬到IM與DM之中。

一開始CPU會去讀取ROM中的指令，而ROM的指令為以下boot.c編譯而成。

**//設定將要搬移的指令數量**

**len = (&\_dram\_i\_end) – (&\_dram\_i\_start) + 1**

**//將dram中的指令搬移到IM中**

**for (i=0; i< len; i++)**

**(&\_imem\_start)[i] = (&\_dram\_i\_start)[i];**

**//設定將要搬移的資料數量**

**len = (&\_\_sdata\_end) – (&\_\_sdata\_start) + 1;**

**//將dram中的資料搬移到DM中**

**for (i=0; i<len; i++)**

**(&\_\_sdata\_start)[i] = (&\_\_sdata\_paddr\_start)[i];**

**//設定將要搬移的資料數量**

**len = (&\_\_data\_end) - (&\_\_data\_start) + 1;**

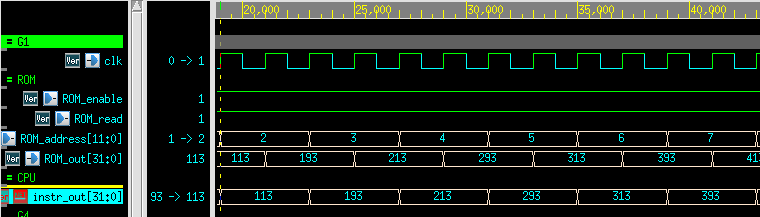
**//將dram中的資料搬移到DM中**

**for(i = 0; i < len; i++)**

**(&\_\_data\_start)[i] = (&\_\_data\_paddr\_start)[i];**

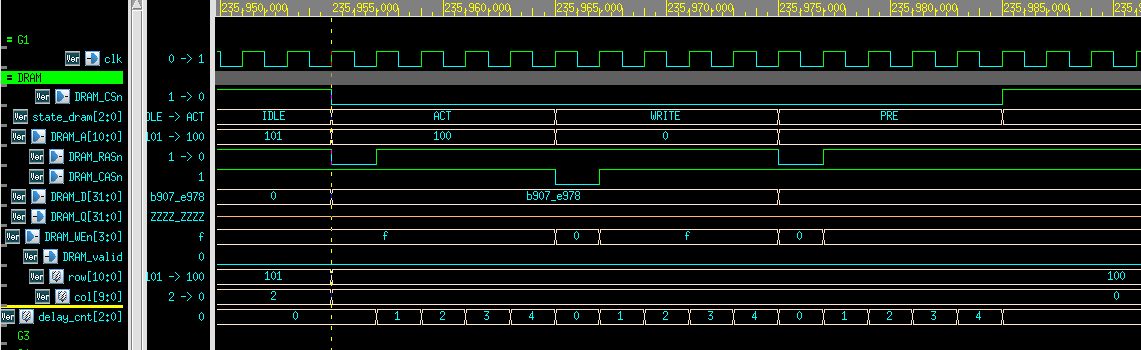
**Screen shot of wave forms and simulation results**

**ROM**

****

CPU的PC從address 0x0開始讀取ROM的資料到CPU的instr\_out，開始執行booting的程式。可以看到上圖中的ROM\_enable以及ROM\_read皆為high，正在被CPU讀取中。

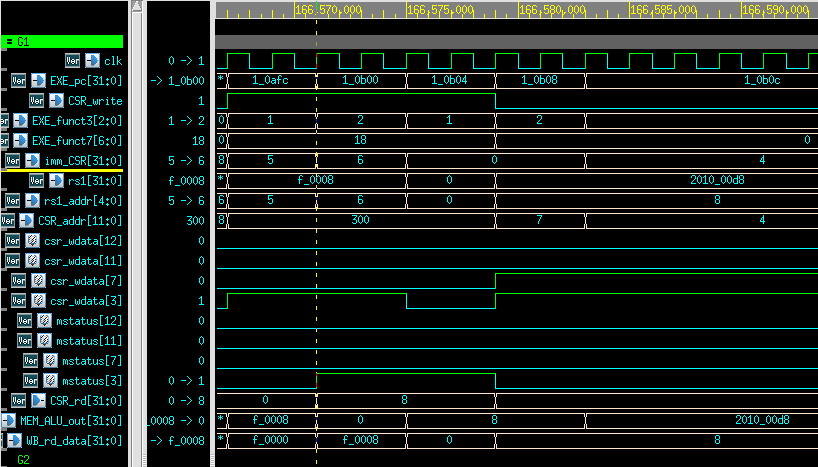
**DRAM**

****

**紅框處可以看出這次是對DRAM做寫入,當state在IDLE時等待AW\_hs\_done訊號後,進入ACT狀態後,將RASn以及CSn設為low****並開始設定ROW address,等待delay\_cnt數到4後進入WRITE state,接著將CASn設定為low,並開始設定Column address,同時將WEn設為4’b0寫入資料,等待delay\_cnt數到4後進入PRE state,接著將RASn設為low並保留原本的ROW address, 同時將WEn設為4’b0, 接著等待delay\_cnt數到4後返回IDLE state**

**CSR instructions:**

1. **CSRRS :**

****

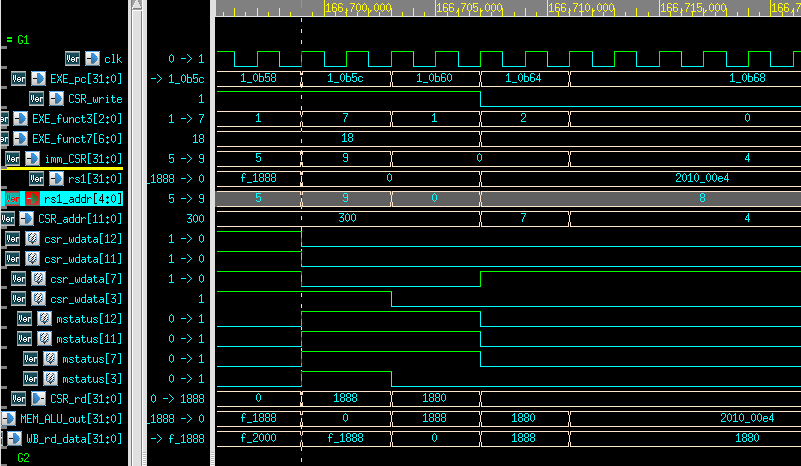
紅色框處EXE\_pc=1\_0b00，見編譯後的main.log可知指令為CSRRS t0,mstatus,t1。

黃色框處可以看到CSR\_addr是0x300為mstatus的位址。

驗證CSRRS指令

* + Rd = csr, if rd!=0 : mstatus的值為0x0….01000，而藍框處可以看到CSR\_rd的值為8。
  + Csr=csr | (rs1) : 0x0…01000 | 0x0…01000 = 0x0…01000，因此可以看到經過當前這個指令後，mstatus的值為0x0…01000。

1. **CSRRCI:**

****

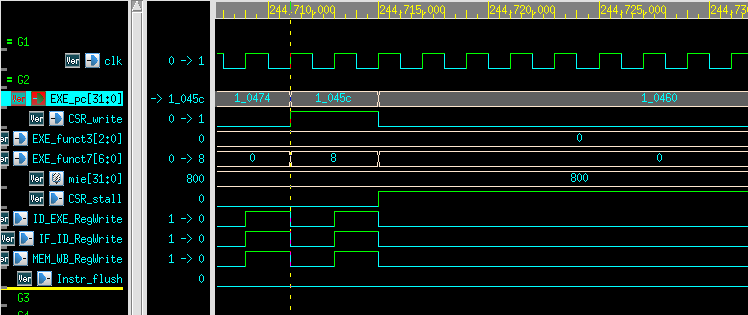
紅色框處EXE\_pc=1\_0b5c，見編譯後的main.log可知指令為csrrci t0,mstatus,9。

黃色框處可以看到CSR\_addr是0x300為mstatus的位址。

驗證CSRRCI指令

* + Rd = csr, if rd!=0 : mstatus的值為1\_1000\_1000\_1000，而藍框處可以看到CSR\_rd的值為1888。
  + Csr=csr &(~uimm) : 1\_1000\_1000\_1000 & (~1001)，因此可以看到經過當前這個指令後，mstatus的值為1\_1000\_1000\_0000。

1. **WFI**

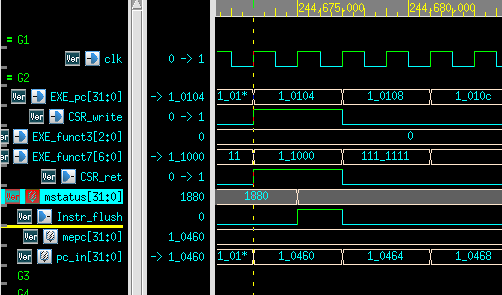
****

紅色框處EXE\_pc=1\_045c，見編譯後的main.log可知指令為wfi。

WFI指令為等待interrupt處理，因此需要將CPU stall住，所以我們新增了CSR\_stall的訊號，在當執行到WFI指令時，將mie[MEIE] assign給CSR\_stall，看是否有interrupt要執行並暫停CPU運作。

黃框處可以看到當mie[MEIE]為1且執行到WFI指令時，CSR\_stall會拉高，並且將每一個stage間的pipeline register也暫停寫入。

1. **MRET**

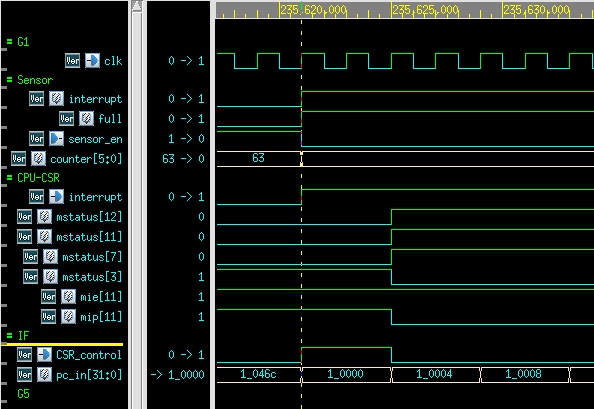
****

紅色框處EXE\_pc=1\_0104，見編譯後的main.log可知指令為MRET。

此指令是當interrupt結束後，要返回原本的pc位址。因此我們設計一個CSR\_ret的訊號，在指令為MRET時會拉高，作為pc選擇時的一條選擇控制線。

因此可以從黃框處看到，當指令被執行時，pc\_in會從mepc取回之前執行到的記憶體位址，返回到1\_0460繼續執行。

**Sensor Control and Interrupt**

****

紅色框處可以看到當sensor\_control中的counter數到63後，即代表sensor中的memory已滿，full和interrupt訊號會同時被拉起。

此時interrupt被傳送到CPU中，可以在黃框處看到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 被拉高且IF中pc\_in跳到0x1\_0000開使執行中斷處理。

**Watch Dog Timer**

**Timeout**

**一張含有 螢幕擷取畫面 的圖片

自動產生的描述**

紅框處可以看到WDEN被設為1代表,準備開始使用WDT計算是否timeout

,由clk觸發的WDEN會先被打一拍到也是由clk觸發的wden\_a register,接著再由clk2觸發的wden\_b1,wden\_b2打兩拍到clk2 domain。

**一張含有 螢幕擷取畫面, 多媒體軟體, 軟體, 繪圖軟體 的圖片

自動產生的描述**

黃框處可以看到WDLIVE藉由toggle flop來延長訊號,再透過wdlive\_b1,wdliveb2,wdliveb3進行CDC的處理,最後再由wdlive\_b3及wdlive\_b4做XOR轉回pulse。

**一張含有 螢幕擷取畫面, 行 的圖片

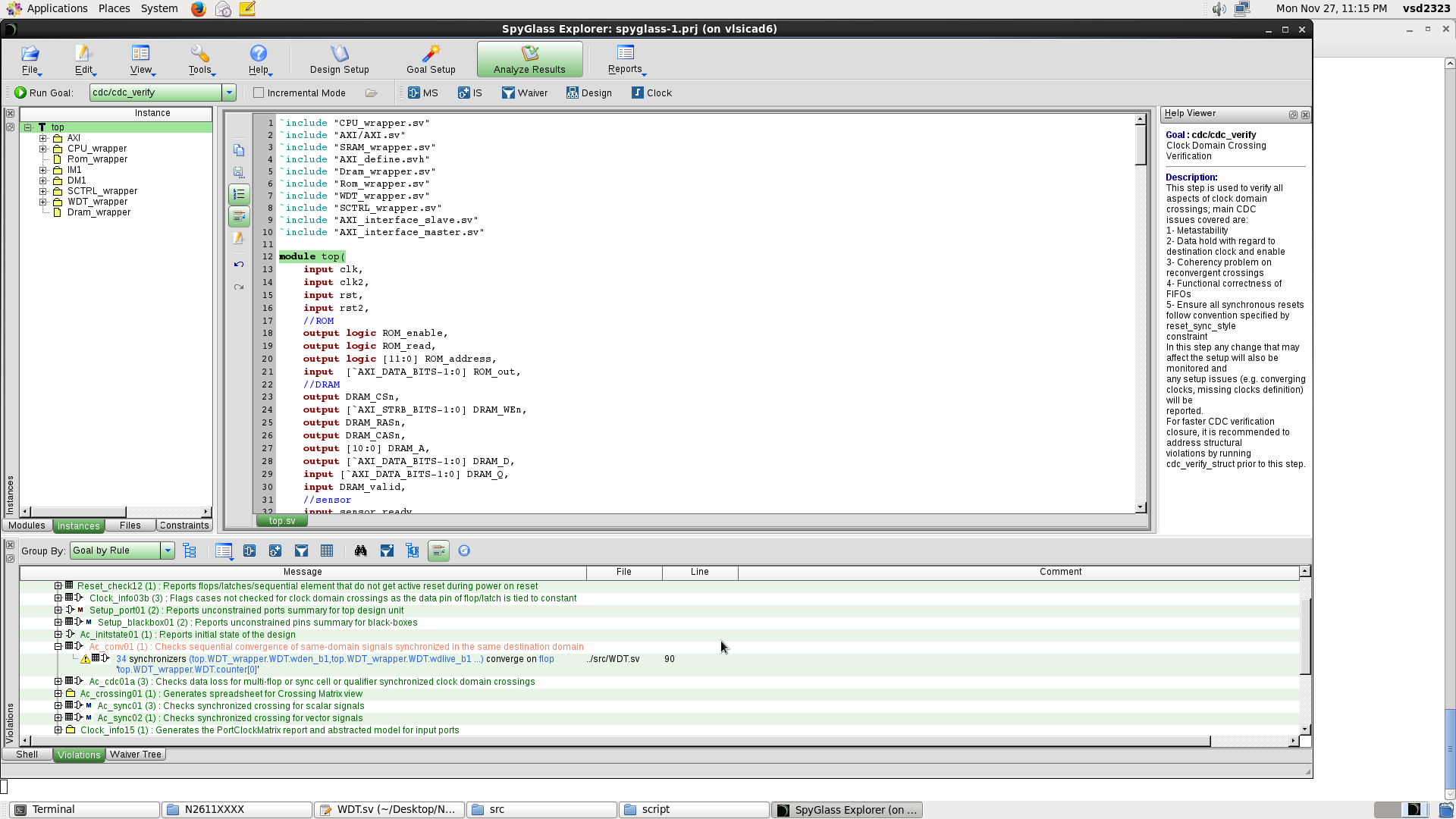
自動產生的描述**

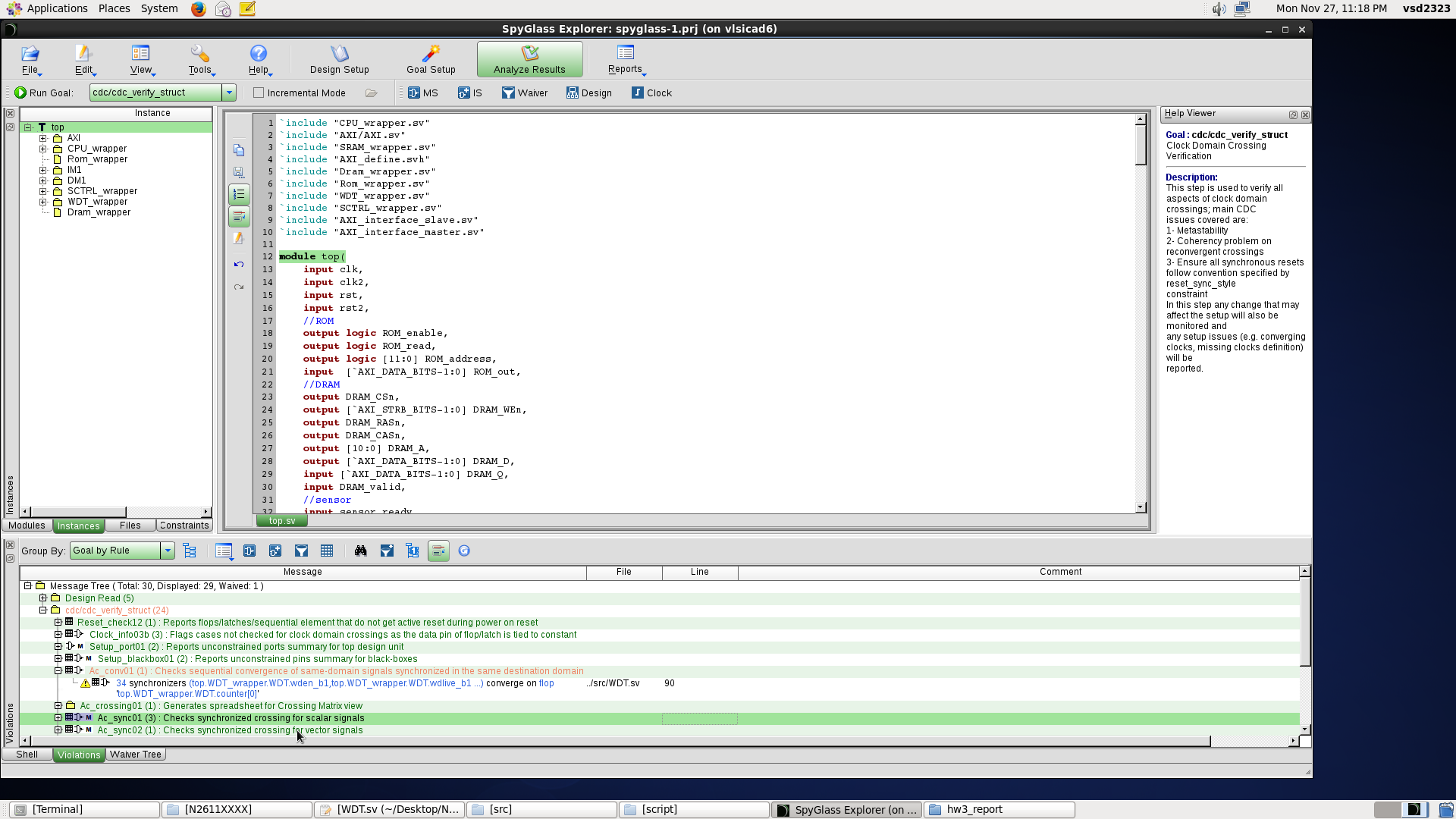
藍框處可以看到WDOCNT為多bit訊號,由於WDOCNT訊號很穩定,所以多打一拍來確保不會發生亞穩態。

**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**

紅框處可以看到當counter累加到wtocnt\_b3的值後會發出wto訊號通知cpu發生timeout並reset整個cpu 系統,由pc\_in可以看出成功重啟CPU。**Spyglass CDC :**

****

****

我們遇到的兩個warnings都是Ac\_conv01的問題，此問題是在說兩個以上經過不同數量flip-flop的資料，聚合在同一個combinational的電路中，可能會導致輸出結果不如預期，可能需要使用FIFO的Gray code同步方法來解決此問題。

**Problems to answer**

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

Mcycle: 是RISC-V指令集架構中的一個計數器，用於計算CPU執行的時鐘周期數，也可以用來追蹤處理器運行的指令數量。

Timer: 是一個通用的術語，指的是計時器，用於測量時間，可以是硬體計時器或軟體實現的計時功能。

What is “Potential Qualifier” in Spyglass?

在Spyglass中，當分析工具偵測到一些模式、規則或潛在問題時，它可能會將其標記為“Potential Qualifier”。這個標記表明該問題或情況可能存在，但需要進一步的審查、確認或處理來確定其是否是真正的問題或只是一個潛在的可能性。

**Lesson learned**

這次作業多了4個新的slave，每個都是一大挑戰，各自都有不同的工作以及限制。尤其是WDT，因為CDC是之前從未接觸的新領域，從0開始學習CDC的知識以及解決方法。加上後面spyglass工具的驗證，讓我們更加清楚不同clock domain間的資料傳輸該如何才能更加穩定，避免產生metastability或是發生不同步的現象。

DRAM的部分，按照規範實作FSM，也讓我們更了解整體DRAM的運作模式以及規範。

CSR也是這次作業中我們認為比較難的地方，他的指令雖然都不難處理，但是在處理interrupt以及timeout發生時，摸索了很長的時間才了解整個和CPU之間各個stage的控制訊號線該如何處理。

最後這次在make syn0卡住了一天半，找了好久，看了好多密密麻麻的波型，最後錯誤竟然是沒有reset到某一條signal，這讓我們深刻體會到reset的重要性。