VLSI System Design (Graduate Level)

Fall 2022

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 waveform files in deliverables

Student name: \_\_王昱承(50%)\_\_ \_\_黃昱澄(50%)\_\_

Student ID: \_\_P76111602\_\_ \_\_ P76111238\_\_

Summary

**Simulation result**

|  |  |  |  |
| --- | --- | --- | --- |
|  | rtl | syn | pr |
| Prog0 |  |  |  |
| Prog1 |  |  |  |
| Prog2 |  |  |  |
| Prog3 |  |  |  |
| Prog4 |  |  |  |

Die area = **9988254.46**

Core Area = **9546863.96**

Power = **164.84779888** mW

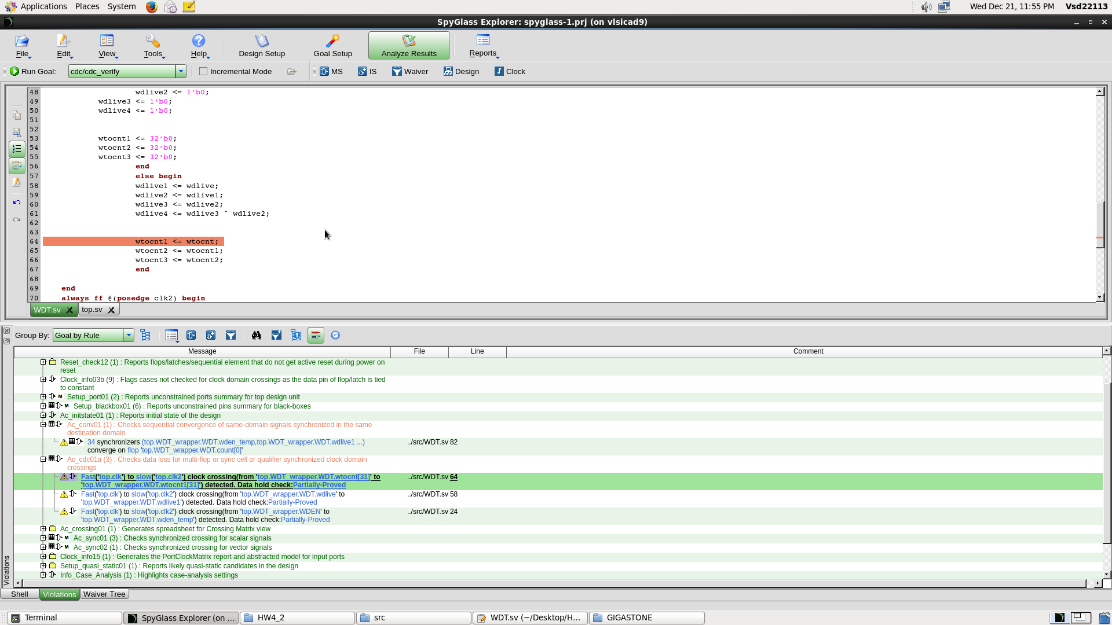
**Spyglass**

一張含有 桌 的圖片

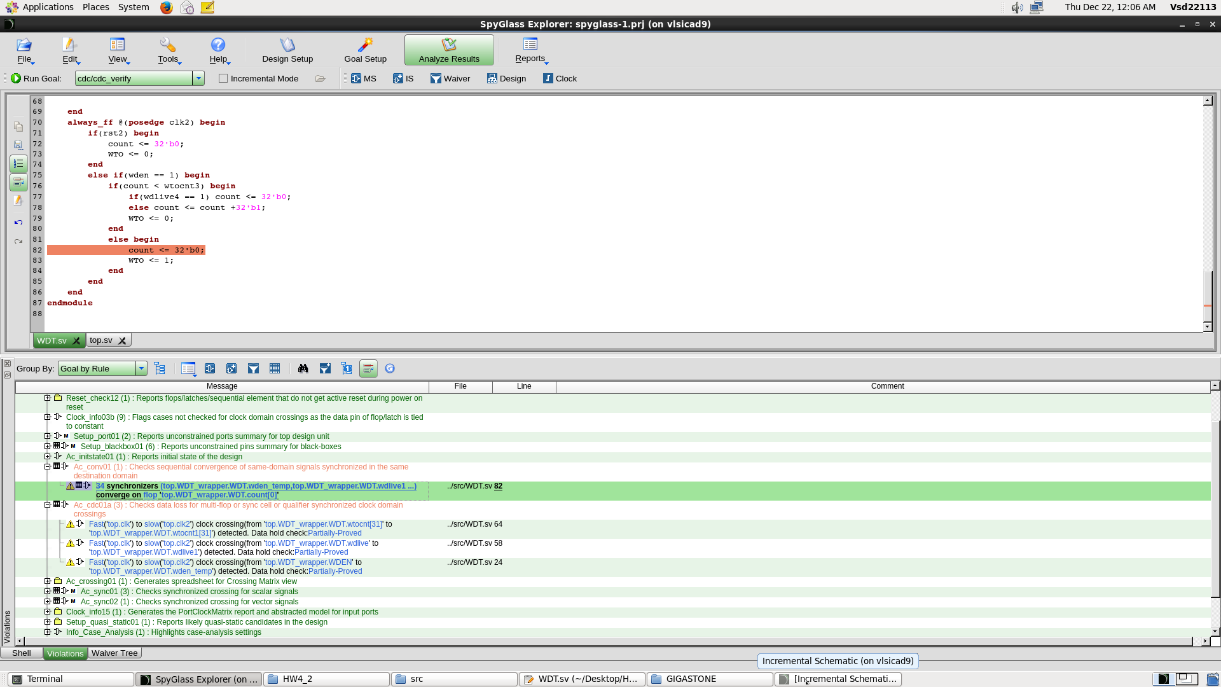
自動產生的描述

一張含有 桌 的圖片

自動產生的描述

****

如上圖，因為我多bit的訊號是用4個flip-flop來實作，所以flip-flop間的訊號會有prtially-prove的問題，而這個問題是因為，如果wtocnt變化很頻繁的話，會使得flip-flop間的值是不同的，導致data-hold check，所以產生warning，不過在這次作業中，不會碰到wtocnt不斷變化的問題，所以此warning不會使系統出問題。



而此warning是說多個訊號從clk跨到clk2但是訊號是分開同步的，所以在對這些訊號加上conbinational 電路的時候，就可能因為某個訊號，突然變動，產生期望之外的結果，但是在這次作業中，wtocnt不會突然變動，所以不會有此問題。

**Superlint**

**一張含有 文字 的圖片

自動產生的描述 一張含有 文字, 收據 的圖片

自動產生的描述**

**(43 + 156 + 1 + 90)/ 4805 = 290/4805 = 0.06 => correct rate = 94%**

剩下的Warning主要是(1) inout port的問題以及(2)Interface和內部名稱不同問題。前者因為我們的架構都是用Interface，所以我們的port一定會是inout port。但這也是Interface的優點，也就是做設計時不需要管細部in/output為何，只管架構設計即可。而後者當然可以每一種interface的class就新增一個對應名稱的.sv檔出來。但是我們覺得這對整體系統架構並沒有大礙，加上如果真的分支出來反而會造成檔案過多，不利於理解與維護，所以最終選擇不去de掉這個bug。

**Innovus**

一張含有 文字 的圖片

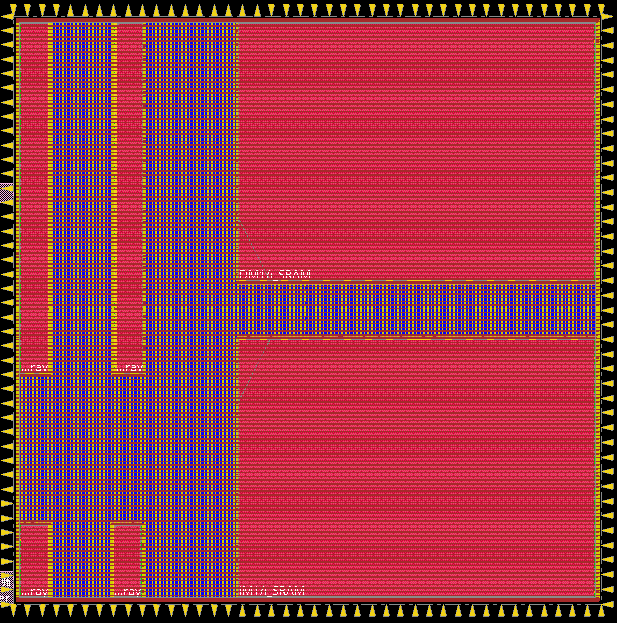
自動產生的描述 DRC :

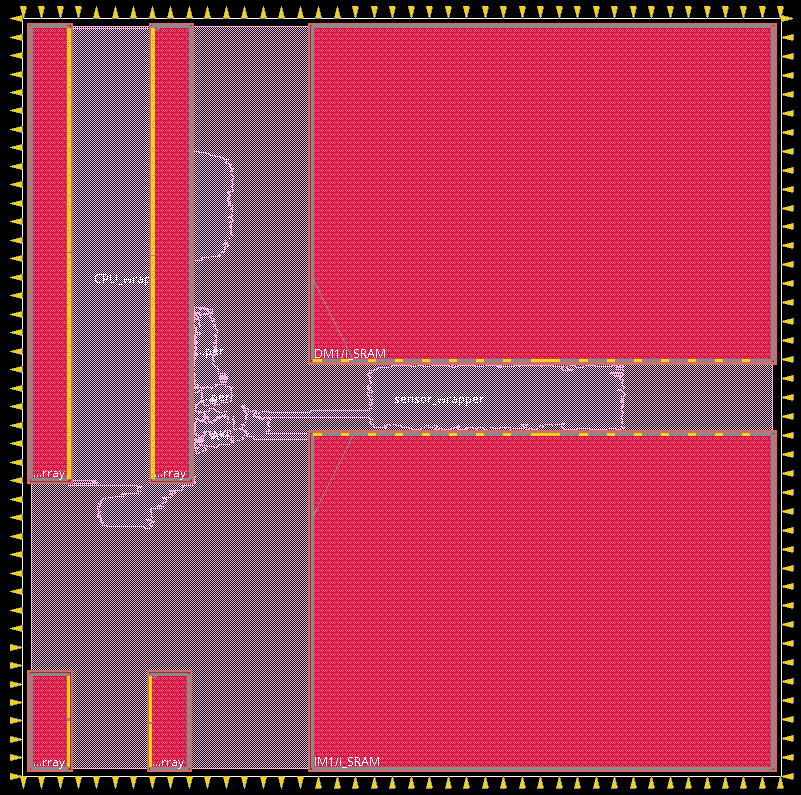
一張含有 桌 的圖片

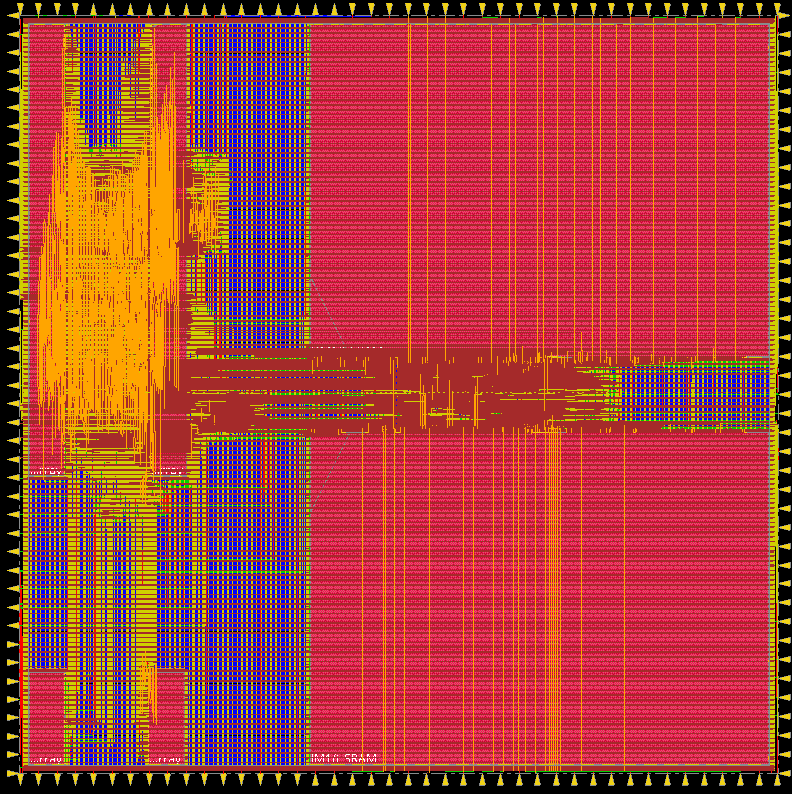
自動產生的描述 LVS ：

一張含有 文字 的圖片

自動產生的描述 VPA ：

FloorPlan view ：

Amoeba ：

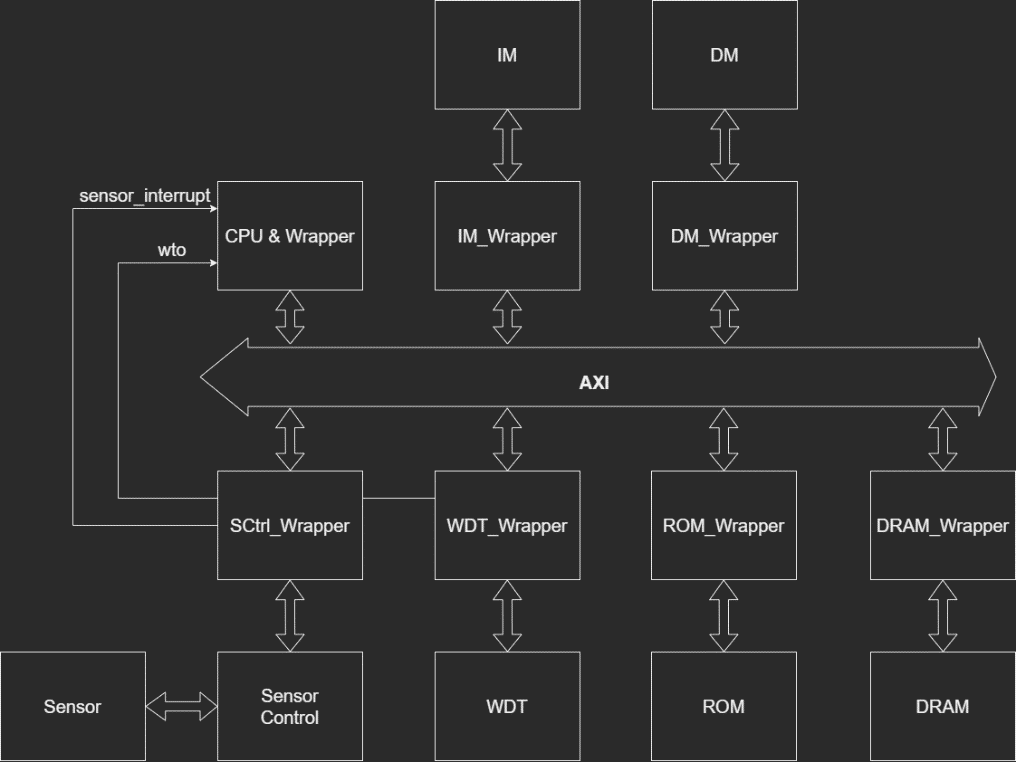
 Physical ：

Lesson learned

這次作業讓我們學習到CDC與CSR的相關知識，這些東西看似簡單，實際上研究之後，發現非常複雜，CDC需要想得很清楚，才能確定做出來的東西到底會不會產生metastable，而且還要通過spyglass大關，而APR看似只要細心跟著做就會成功，實際上也有很多小細節要注意，尤其擺放的位置可謂差之毫釐失之千里，一不小心DRC驗證就不會過，讓我在soc實驗室重勒了5次，擺的位置不夠完美，則setup time & hold time的slack就會是負的，在這邊也讚嘆一下cadence，他的optimize Design真的是魔法，可以各種把負的值轉為正的，可謂非常牛逼。

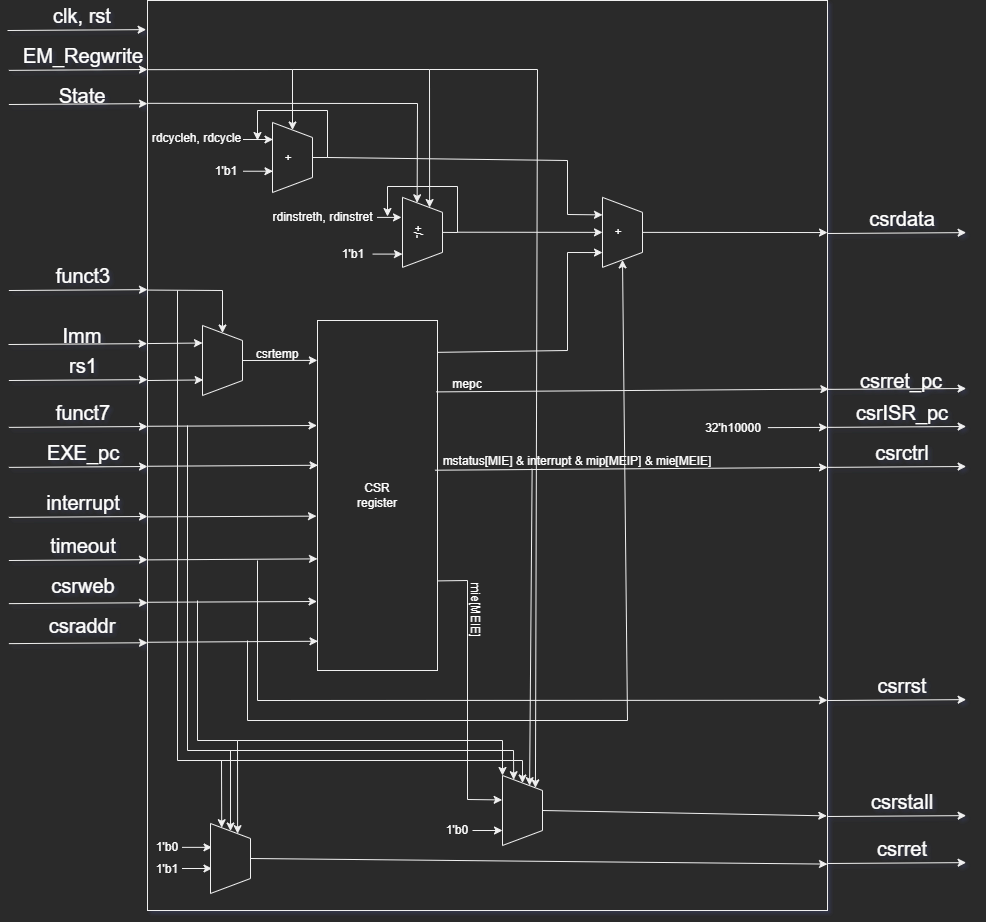
Architecture diagram

**Full Architecture**



上圖是這次HW4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。

**CSR Design**



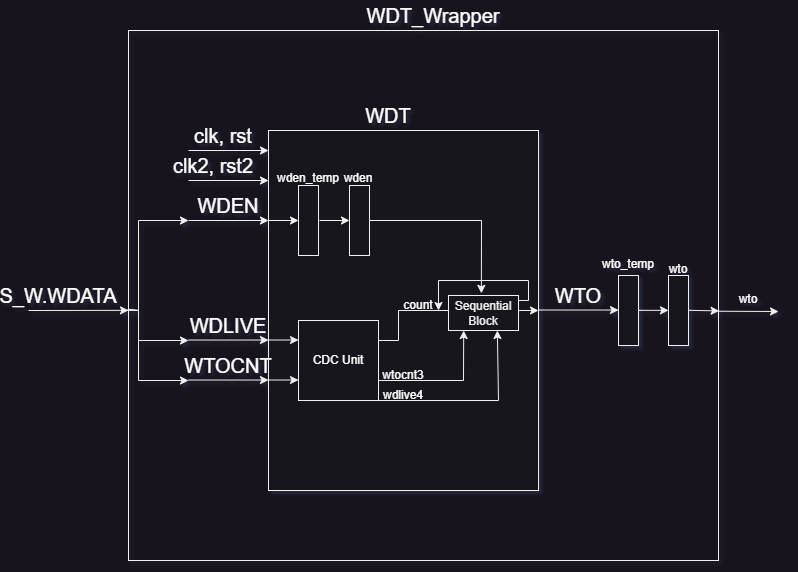
延續之前的作業部分，CSR我們是放在EXE中，因為會需要使用到CSR的站存器與rs1做運算，以及將相關的值存進rd中。另外在最易開始做Branch判斷的地方也是EXE，所以最終把CSR放置於EXE中。

由上圖可知CSR主要由CSR Registe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。

**SCtrl\_Wrapper**

因為SCtrl\_Wrapper與SRAM類似，並於之前report有詳細介紹SRAM架構，在此即不再贅述。

**WDT & WDT\_Wrapper**



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0x10010200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**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。

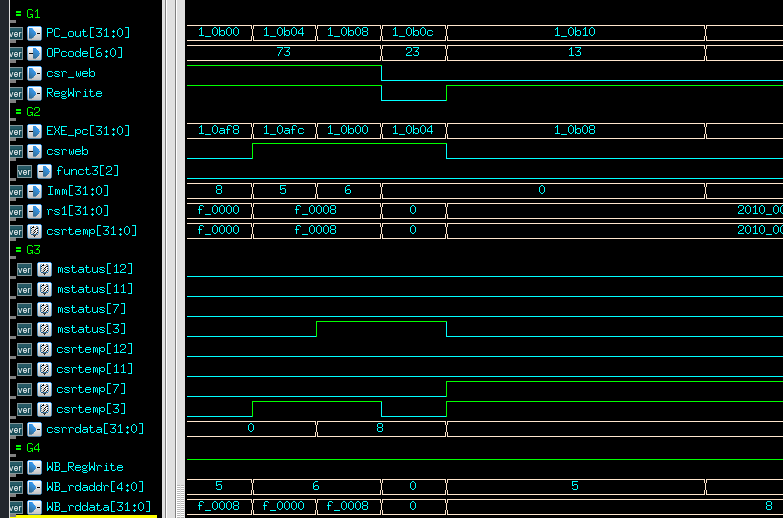
**WDLIVE**因為此訊號是一個pulse 所以使用了toggle flop將pulse轉變成level就可以進行CDC的處理，在output 端再使用XOR和一個flop來重新轉回pulse。

**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。

Waveform & Explanation

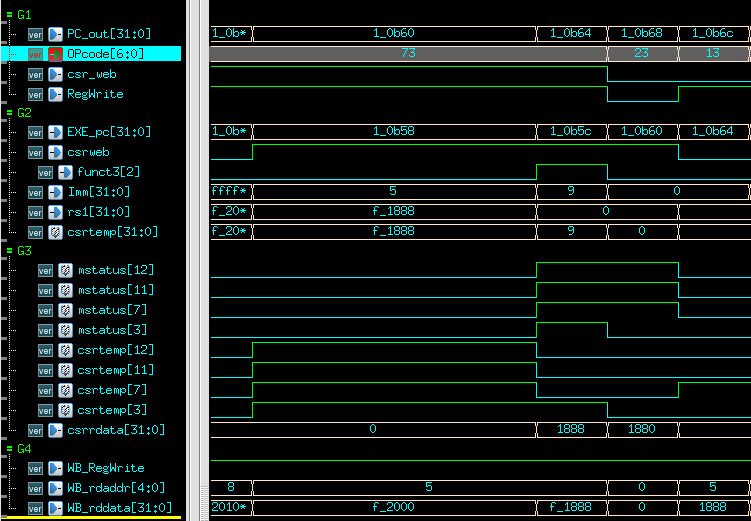
**CSR operation**

CSRRS



首先PC\_out = 1\_0b00，在這一條的對應指令為 csrrs t0,mstatus,t1。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funct3[2] == 1`b0，所以 csrtemp = rs1 = 0xf\_0008。然後可以在黃框處發現csrrdata = mstatus[x] | csrtemp[x]，x = 3、7、11以及12。最後綠框處可以發現有將最終的結果0x8寫回reg中。

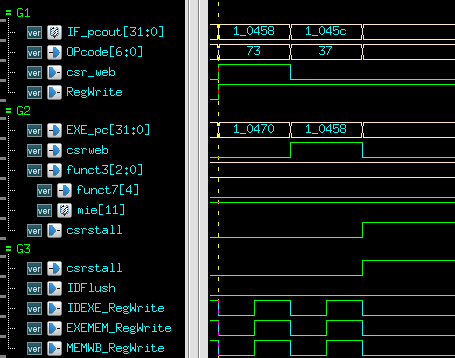
CSRRCI



首先PC\_out = 0x1\_0b5c，在這一條的對應指令為 csrrci t0,mstatus,9。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funct3[2] == 1`b1，所以 csrtemp = Imm = 9。然後可以在黃框處發現csrrdata = mstatus[x] & ~csrtemp[x]，x = 3、7、11以及12。最後在綠框處可以發現有將最終的結果0x1888寫回reg中。

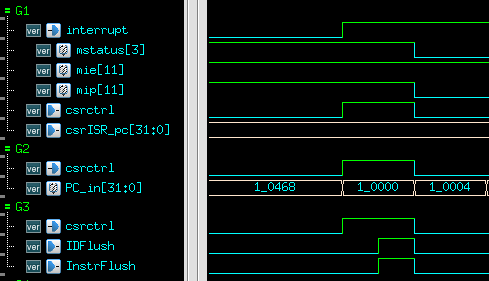
**Interrupt processing**

WFI



首先PC\_out = 0x1\_0458，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funct3 == 3`b0且funct7[4] == 1`b0，還有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csrctrl 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csrstall pull HIGH，且IF、ID的Flush全部pull HIGH。

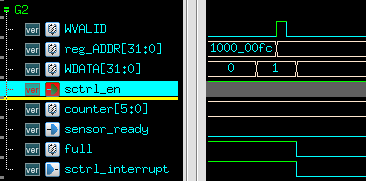
MERT

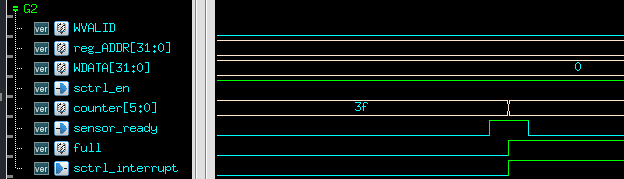
一張含有 文字, 計分板 的圖片

自動產生的描述

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，以及吃到CSR operation時將csr\_web 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。

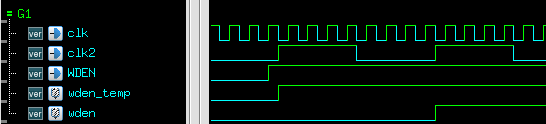
**Sensor Control**

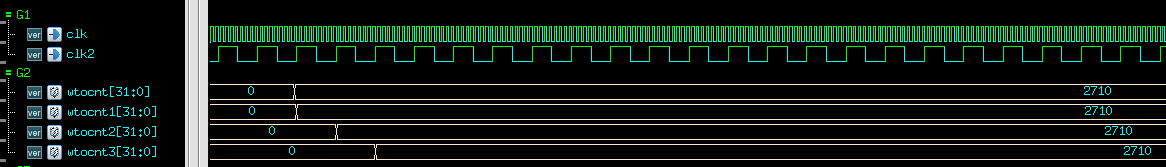
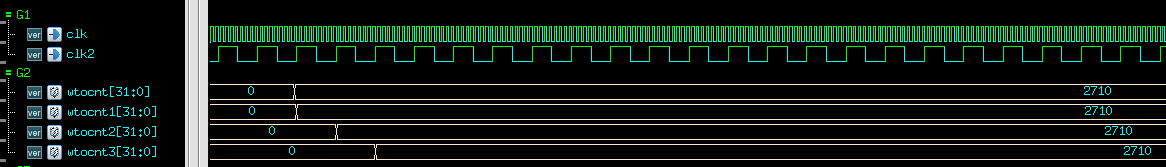
****

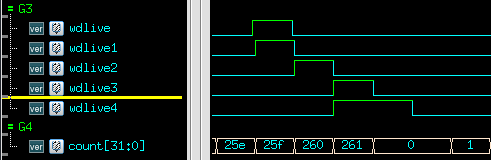
****

當pull up WVALID時，可以發現圖中紅框處根據reg\_ADDR將WDATA[0]賦值給sctrl\_en。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\_resdy以及counter = 0x3f來pull full HIGH，最終發出interrupt。

**Watch Dog Timer**



…



一張含有 文字, 時鐘 的圖片

自動產生的描述

從最上方紅框處可見，為clk與clk2不同，所以先把所得WDEN傳給wden\_temp，在於clk2 pull HIGH時將temp的值賦予wden。接著在兩個藍框處可以發現，我們通過延遲3拍來確保不會發生亞穩態。另外可以在黃框處發現當wdlive4 pull HIGH 時，count小於Timeout的數值時(261 < 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。

C code of booting process

在程式執行之前，會需要藉由boot.c來將DRAM中的data搬移到SRAM供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。接著data被放置於兩處，分別是\_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”**

Performance

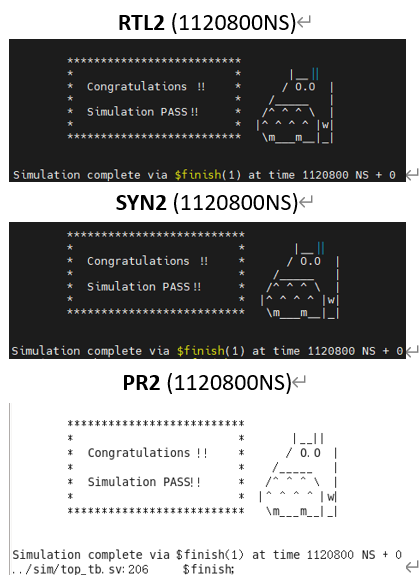
**Prog0 Prog1**

一張含有 文字 的圖片

自動產生的描述一張含有 文字 的圖片

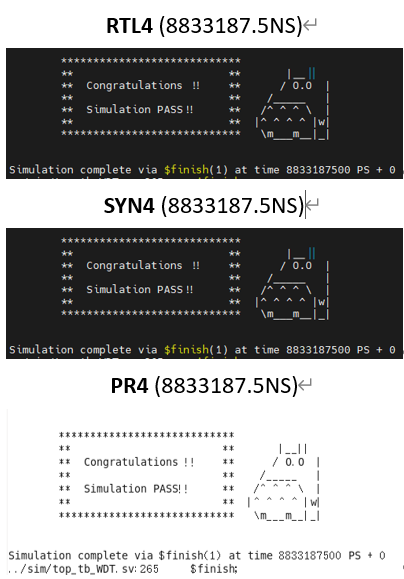
自動產生的描述

**Prog2 Prog3**

一張含有 文字 的圖片

自動產生的描述

**Prog4**



**Area**

一張含有 文字 的圖片

自動產生的描述

**Power**

一張含有 桌 的圖片

自動產生的描述