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: \_\_\_陳語彤\_\_\_ , \_\_陳俐蓉\_\_\_\_

Student ID: \_\_N26121664\_\_, \_\_N26120113\_\_

**Summary**

在這次的作業中在ISA的部份我們主要是將原先寫好的CPU加上I/O與interrupt的處理，並且增加CSR、WFI、MRET相關指令，而在IP的部份我們須增加DRAM Wrapper、ROM Wrapper(用來存放開機指令的記憶體)、Sensor Control Wrapper、WDT Wrapper(用來製造中斷，同時也會有CDC的問題發生)，除了上述硬體的實作外，軟體的部份需實作出sort與gray scale和booting的c程式碼，最後還需要通過AXI的VIP驗證及運用EDA tool來驗證CDC是否符合規範。

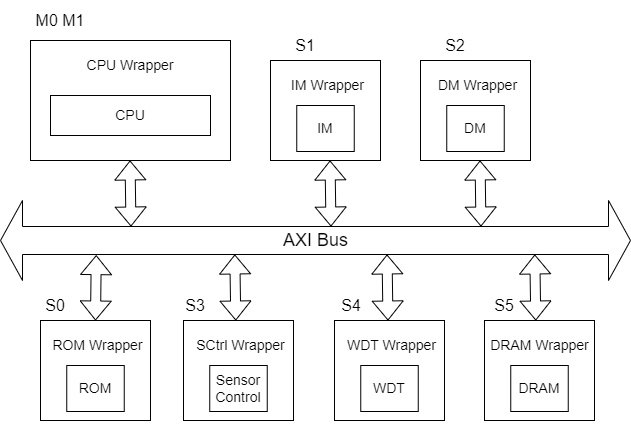
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hardware | | | | | | | | |
|  | | | | | | RTL | | synthesis |
| Top | CPU\_wrapper | | CPU | | | √ | | √ |
| New instructions | | | √ | | √ |
| SRAM\_wrapper (IM & DM) | | | | | √ | | √ |
| ROM\_wrapper | | | | | √ | | √ |
| DRAM\_wrapper | | | | | √ | | √ |
| AXI | | | | | √ | | √ |
| Sensor control wrapper | | | | | √ | | √ |
| Watch Dog Timer | | | | | √ | | √ |
| Synthesis result | | | | | | | | |
| Area | | | | | Clock cycle(ns) | | | |
| 6058639 | | | | | 13.0 | | | |
| Firmware & Software | | | | | | | | |
|  | | RTL pass | | syn pass | | | Execution time(ns) | |
| Booting | | pass | | pass | | |  | |
| Prog 0 | | pass | | pass | | | 1306106 | |
| Prog 1 | | pass | | pass | | | 8326730 | |
| Prog 2 | | pass | | pass | | | 19188386 | |
| Prog 3 | | pass | | pass | | | 15919406 | |
| Prog 4 | | pass | | pass | | | 7281595 | |
| Prog 5 | | pass | | pass | | | 7281998 | |
| Spyglass summary(number of inline messages) | | | | | | | | |
| Information | | Warning | | Error | | | Fatal | |
| 235 | | 12 | | 0 | | | 0 | |
|  | | | | | | | | |
| Superlint(number of inline messages) | | | | | | | | |
| Total lines | | Warning | | Error | | | coverage(%) | |
| 7556 | | 81 | | 0 | | | 98.9% | |

**Contribution**

|  |  |
| --- | --- |
| 陳語彤 50% | 陳俐蓉 50% |
| AXI 程式與VIP驗證、DRAM wrapper、ROM wrapper、SCtrl wrapper、Prog1程式、Prog2程式、cdc 驗證、booting程式碼、撰寫報告 | 新增CPU指令、SRAM wrapper、CPU wrapper、WDT 程式碼、WDT wrapper、cdc 驗證、撰寫報告 |

**Hardware Design Description**

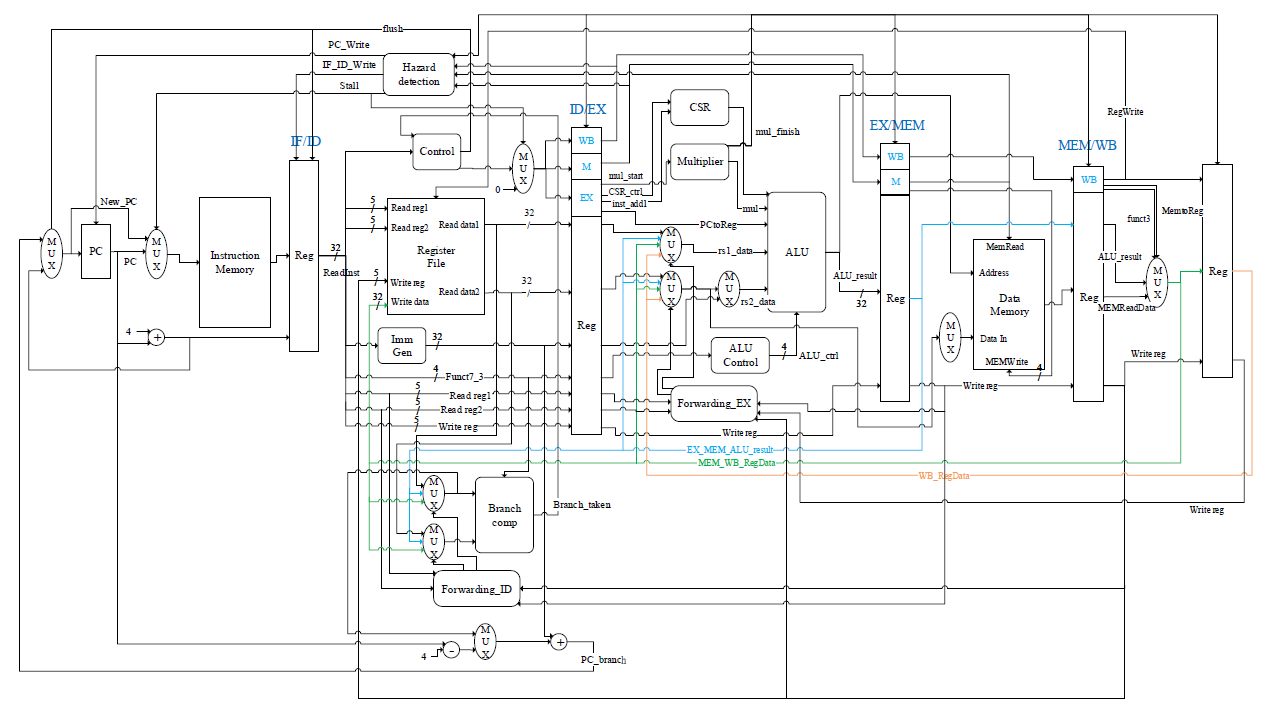
* System Block Diagram



這次作業共有2個master和6個slave，WDT的操作頻率為10MHz (cycle time = 100ns)，CPU和其他wrapper操作頻率需大於50Mhz (cycle time >= 20ns)，以下介紹系統的運作流程:

1. 系統一開始會先進行booting開機，CPU會到ROM讀取開機指令，開機指令會透過load和store將存放在DRAM的program指令和初始化資料搬運到IM和DM。
2. 接著CPU會到IM讀取program指令並開始執行。
3. 過程中若sensor control或WDT發出interrupt，CPU會視情況接受中斷，暫停執行目前的指令，並跳去interrupt的PC，去執行interrupt要求的指令。
4. 中斷事件處理完畢後，CPU會回到中斷前的PC，繼續執行剩下的指令。
5. 最後，CPU會將運算完畢的資料存回DRAM。
6. 重複2~5步驟直到執行完整個program指令。

* CPU Architecture



Interrupt Control

CSR

ALU

CSR

Register file

因為這次實作增加interrupt的功能，所以CPU中需要多增加了CSR Register file、CSR ALU、Interrupt Control三個模組，以及Instruction decode(Control)也需要新增辨識CSR、MRET、WFI指令的功能。

**新增指令:**

1. CSR instruction

|  |  |
| --- | --- |
| Instruction | Description |
| CSRRW | CSR = rs1\_data , rd = CSR |
| CSRRS | CSR = CSR | rs1\_data , rd = CSR |
| CSRRC | CSR = CSR & (~rs1\_data) , rd = CSR |
| CSRRWI | CSR = uimm , rd = CSR |
| CSRRSI | CSR = CSR | uimm , rd = CSR |
| CSRRCI | CSR = CSR & (~uimm) , rd = CSR |

1. WFI instruction

一旦CPU開始執行WFI，就會進入Wait For Interrupt mode(WFI mode)，CPU會暫停執行後續指令，直到interrupt 被taken後，會結束WFI，接著跳到mtvec中記錄的PC去執行interrupt的program。

1. MRET instruction

當interrupt的program處理完畢時，會執行MRET，代表CPU可以返回去繼續執行interrupt前的指令，此時PC會跳到mepc中所記錄的PC。

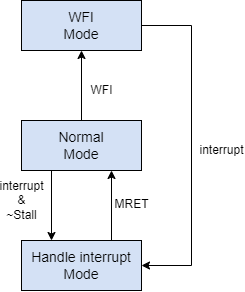
**新增模組:**

1. Interrupt Control

根據以下三種模式決定目前CPU是否可以接受interrupt(本設計中以interrupt\_en訊號表示)

1. Normal mode: 一般模式，尚未有interrupt中斷CPU，此時CPU可以正常執行指令。一般情況下，此時CPU通常可以接受interrupt，但若此時CPU正在Stall，就不會接受interrupt，直到Stall結束。
2. Handle interrupt mode : 中斷模式，表示CPU已接受interrupt，目前正在處理interrupt。當CPU進入此模式後，便不會再接受其他interrupt。
3. WFI mode : 等待中斷模式，此時CPU會停止執行後續指令，直到interrupt taken後，才會結束此模式，並接著處理interrupt。

以下為三種模式的轉換方式



1. CSR Register file

CSR Register file位於ID stage，更新CSR Register file需考慮是否為CSR指令的寫回要求、interrupt taken、MRET三種狀況，CSR Register file會根據狀況做對應的更新，各暫存器功能如下:

1. mstatus : 紀錄Machine level CPU目前的狀態，可分為「正在執行interrupt」以及「從interrupt返回」，目前僅針對MIE、MPIE、MPP做處理，其他位元固定為0。
2. mie : Machine level，用MTIE和MEIE兩個位元分別記錄timer interrupt enable和external interrupt enable，只有interrupt enable為1時，interrupt才能被taken。
3. mtvec : Machine level，紀錄interrupt後要執行的指令的PC(Trap Address)，interrupt被taken後，PC會跳到mtvec所記錄的PC值(目前把其中的值固定為0x0001\_0000)，去處理interrupt需執行的program。
4. mepc : Machine level，在interrupt taken的同時，mepc會紀錄目前被中斷的指令的PC，當interrupt return之後，PC會跳到mepc中紀錄的PC，繼續執行中斷前的program。實際做法是當CPU接受interrupt時，會將目前到ID stage為止的指令執行完，並記錄下個待執行指令的PC。
5. mip : Machine level，負責記錄待處理的interrupt，因為並非interrupt發生馬上就會被taken，目前只針對MEIP、MTIP做設計。
6. mcycleh : Machine level，計算目前CPU執行了多少個cycle(前32bits)。
7. mcycle : Machine level，計算目前CPU執行了多少個cycle(後32bits)。
8. minstreth : Machine level，計算目前CPU執行了多少個instruction(前32bits)。
9. minstret : Machine level，計算目前CPU執行了多少個instruction(後32bits)。
10. rdcycleh：User level，計算目前CPU執行了多少個cycle(前32bits)。
11. rdcycle：User level，計算目前CPU執行了多少個cycle(後32bits)。
12. rdinstreth：User level，計算目前CPU執行了多少個instruction(前32bits)。
13. rdinstret：User level，計算目前CPU執行了多少個instruction(後32bits)。
14. CSR ALU

CSR ALU位於EX stage，根據Instruction decode產生的CSR\_ctrl訊號，處理CSR指令的相關運算，再將結果寫回CSR Register file，這裡採用在EX stage處理完後在同個cycle將結果寫回的方式，避免CSR Register forwarding的問題。但若是source register是屬於Register file，依然需要考慮forwarding，此處和之前實作共用同個forwarding unit即可。

* DRAM wrapper FSM chart

|  |
| --- |
|  |
| 上圖為DRAM的狀態圖，在reset完後會進入到IDLE中等待(AWVALID && AWREADY)或是(ARVALID && ARREADY)就會進到ACT當中，在還未執行第一次的ACT之前row hit的值皆為0，進到ACT後就會進行row寫入的動作，同時也會將寫入的row存入row register中，此值將在後續的執行中用來判斷是否有row hit的情形發生，同時也會判斷是進行讀還是寫，如果是讀就會進到WRITE中，如果是進行讀則會進到READ當中，已進入WRITE為例，在此狀態中會將column作寫入的動作，同時也將資料作寫入的動作，等到counter為4的時候就會跳到PRE狀態中去做充電的動作，同時也會將BVALID與RVALID拉起，而等到counter為4的時候就會跳回IDLE去等待下一次的讀或寫。  而如果是第二次之後進入到DRAM中則會多出row hit的判斷，如果此時為row miss的話就與上述執行步驟相同，如果是row hit的話就會直接跳過ACT的狀態，進到WRITE或是READ當中去做讀或寫，接著當counter為4的時候就會跳過PRE的狀態進到BVAL當中，在此狀態會將BVALID與RVALID拉起，直到(RVALID & RREADY)或是(BVALID & BREADY)就會跳回IDLE去等待下一次的讀或寫。 |

* Sensor Control & External Interrupt

Sensor每隔1024個cycle會產生一筆資料，sensor control中有自己的local memory來儲存sensor產生的資料，如果local memory已滿，sensor control會停止接收資料(sensor\_en = 0)，並對CPU發出external interrupt，CPU接受interrupt後，會開始將資料從sensor control搬運到DM。以下為sensor control的local memory和變數空間的對應儲存位址:

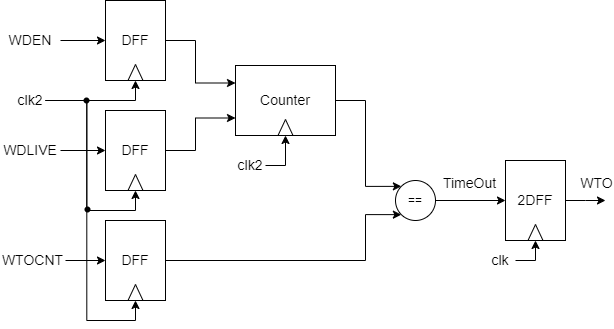
|  |  |
| --- | --- |
| Address | Mapping |
| 1000\_0300 - 1000\_03FF | Mem[0] – mem[63] |
| 1000\_0100 | sctrl\_en |
| 1000\_0200 | sctrl\_clear |

* Watch Dog Timer & Timer Interrupt

1. WDT

clk domain

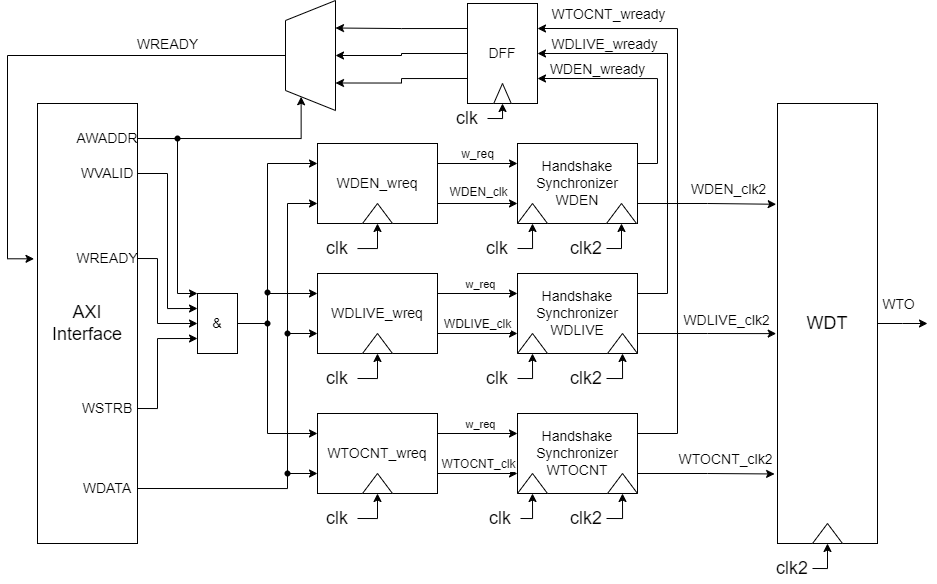
clk2 domain



WDT其實就是一個counter，當WDLIVE=1時，counter會被重置為0；當WDEN=1時，counter會開始計數；當counter數值等於WTOCNT時，WTO會拉起，此時WDT會發出timer interrupt。

因為WDT和CPU屬於不同的clock domain，所以訊號在輸入或輸出WDT前都需要經過CDC處理。WDT的三個輸入訊號(WDEN、WDLIVE、WTOCNT)會在wrapper中先做CDC處理，原本的設計是直接將輸入訊號拿來使用，但這種作法在spyglass中會有data convergence 的warning，因此我們改成將輸入訊號先送入暫存器後再拿來使用，讓訊號維持穩定。輸出訊號WTO因為是1-bit long pulse，所以這裡只需使用兩級flip flop(2DFF)做CDC處理即可。

1. WDT wrapper



clk2 domain

clk domain

WDT wrapper中除了AXI介面以外，因為WDT屬於clk2 domain，而Bus資料屬於clk domain，所以當Bus資料要從clk domain傳輸到clk2 domain時會產生CDC path，需要新增Synchronizer來處理CDC，並透過handshake的方式將Bus資料過渡到clk2 domain，再送入WDT。

這裡會依據AWADDR對WDEN、WDLIVE、WTOCNT三個變數空間寫入資料，每次只會對一個變數做寫入。以下為三個變數在WDT中對應的儲存位址:

|  |  |
| --- | --- |
| Address | Mapping |
| 1001\_0100 | WDEN |
| 1001\_0200 | WDLIVE |
| 1001\_0300 | WTOCNT |

Synchronizer handshake步驟如下:

*(Handshake Synchronizer架構圖參考CDC Circit-1)*

1. 當WVALID和WREADY同時拉起時，代表此時可以開始傳輸寫入資料，在資料送入Synchronizer之前需要先送入暫存器(XXX\_wreq)，讓資料維持穩定。
2. 在Synchronizer接受clk domain的寫入要求(w\_req)後，XXX\_wready會先轉為low，表示此時Synchronizer不再接受新的資料寫入要求，並把目前傳入的資料暫存在Buffer\_clk。
3. 在寫入要求透過Pulse Synchronizer傳輸到clk2 domain後，此時clk2 domain可以接受寫入資料，此時Buffer\_clk2會更新寫入資料。
4. 將clk2 domain已接受寫入要求的訊息透過Pulse Synchronizer傳回clk domain，接著XXX\_wready轉為high，表示Synchronizer可以接受新的資料寫入要求。

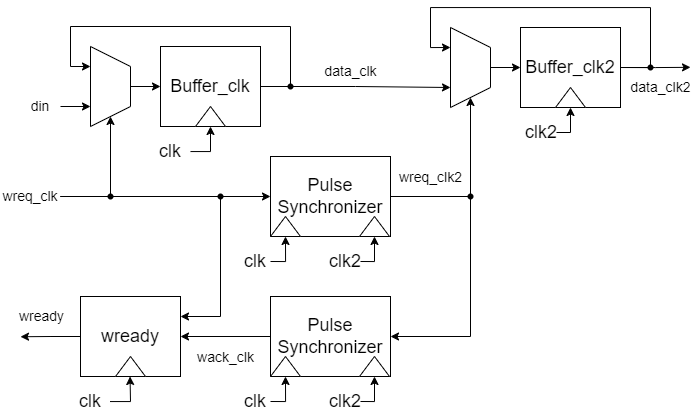
重複以上步驟達到handshake的目的。

* CDC Circuit

1. Handshake Synchronizer

clk2 domain

clk domain



因為Bus傳輸的資料可能是多個bit，無法直接使用2DFF解決CDC問題，且需要做handshake機制，所以這邊使用Handshake Synchronizer做為 WDT wrapper中的CDC電路。Handshake Synchronizer電路運作原理和步驟已在WDT wrapper中詳細說明。

1. Pulse Synchronizer

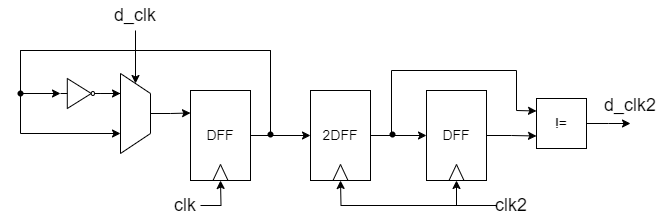
clk2 domain

clk domain

2.

3.

1.

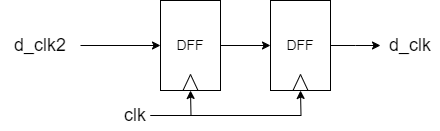


當輸入訊號(d\_clk)為1T pulse訊號時，需使用Pulse Synchronizer，由於送入Handshake Synchronizer的寫入要求(wreq\_clk)可能是1T pulse訊號，所以要用Pulse Synchronizer做訊號跨頻處理。此電路可以分為3個步驟處理:

1. 產生clk domain的long pulse
2. 使用2DFF將clk domain的long pulse跨頻到clk2 domain
3. 產生clk2 domain的1T pulse訊號
4. 2DFF 1-bit Synchronizer

clk domain

clk2 domain



2DFF只能用來處理1-bit訊號的CDC問題，在本設計中除了用在Pulse Synchronizer的跨頻處理外，從WDT送出的WTO訊號也需要做跨頻處理，因為它是1-bit的long pulse訊號，所以用2DFF處理即可。

**Software & Firmware design description**

* Booting

|  |
| --- |
|  |
| 上圖為booting的程式碼，在此次作業中的booting指令放在ROM中，在開機時，CPU會最先access ROM，做開機的動作，此程式碼的功能是將資料進行搬運，把原本存在DRAM中的指令與data複製到instruction memory 與 data memory中，可以看到上圖的第一個for迴圈，此迴圈主要是將DRAM的資(也就是instruction)存到instruction memory當中，而第2.3個for迴圈皆是將DRAM中的資料存到data memory中。 |

* Prog 1

|  |
| --- |
|  |
| 上圖prog1的程式碼，在程式撰寫的過程中其實沒有太大的問題，因為在前幾次作業中就有實作過了，在for迴圈之前我們先將第一個值存入以sort的陣列當中，之後進到for迴圈，在此從原先還未sort的陣列中將第一個值讀出，並且儲存為temp，並將sort index為i的值(就是現在sort完成的陣列要加入新的值的位置)，接著與sort完成的陣列最後進行比較(也就是i-1)的位置，如果遇到比temp大的值就會將值進行互換，直到遇到比temp小的值就會跳出while迴圈，接著就會繼續讀入下一個還未sort的值，直到所有值皆被sort完成。 |

* Prog 2

|  |
| --- |
|  |
| 以下simulation結果皆為在clock cycle=20ns的情形下，與最終clock cycle不同 |
| 上圖為prog2的程式碼，在程式撰寫的過程中遇到許多的問題，首先是for迴圈讀不到值(下圖1)，因為一開始我以為助教在HW3投影片中所提到的\_binary\_image\_bmp\_size是給訂好的值，所以我直接將它當成判斷for迴圈停止的標準，導致在simulation時結果都是為0，叫出波型圖來才發現\_binary\_image\_bmp\_size的值為unknown，所以迴圈根本不會停止，再仔細看過投影片後才發現\_binary\_image\_bmp\_size是要自己去做計算的，計算過程為上圖1.的地方。  接著再2.的部分是將54 byte的header直接存入結果，因為header不須經過轉成灰階的動作，而在4.的部分是將RGB的數值轉成灰階，而在一開始的時候是用gray = ((blue \* 0.11) + (green \* 0.59) + (red \* 0.30))來計算，但執行結果一直不正確(下圖2)，最後看波型圖後才發現因為太多小數的處理導致在時間內無法計算完成，所以才想到可以將其結果全部做完在同時除100，結果為(下圖3)，因想要將其執行時間縮短所以我上網查了一些資料，看到在白色部分也就是RGB三個值皆為0Xff時，再轉到灰階結果也會是0Xff，所以我就增加了一個判斷條件，就是在RGB三值皆為0Xff時就直接將灰階結果給0Xff，這樣就可以減少很多的執行時間(可由下圖3與4做比較) |
|  |
| 圖1 |
|  |
| 圖2 |
|  |
| 圖3 |
|  |
| 圖4 |

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

* Wave forms

|  |
| --- |
| First DRAM Execution or Row Miss |
|  |
| 上圖為第一次寫入或讀取DRAM的過程，在DRAM的狀態機中，因為為第一次寫入所以一定會進到ACT及PRE的狀態去做row的選取及充電，可以看到一開始的狀態為IDLE接著只要AWVALID & AWREADY就會進到ACT的狀態中，在ACT狀態一開始會先將RASn降為低電位，將地址給定為AWADDR[22:12]，而因為DRAM需延遲5個cycle所以等到counter為4時就會跳到WRITE的狀態中，在WRITE的狀態首先會先將CASn降為低電位，並將地址給定為AWADDR[11:2]，同時也須將WEn的值給為0表示同意寫入，到counter為4時就會跳到PRE的狀態中去做充電的動作，會先將RASn降為低電位，定將地址給定為AWADDR[22:12]，同時也須將WEn的值給為0。 |

|  |
| --- |
| DRAM Execution WRITE (Row Hit) |
|  |
| 上圖為第二次以上寫入或讀取DRAM的過程，在IDLE狀態等待AWVALID & AWREADY與ARVALID & ARREADY的同時也會判斷是否row hit，如果row hit的情況下就不用進到ACT的狀態下去做找row的找尋，這樣一次的讀取或寫入就可以省下10cycle的時間，會直接從IDLE的狀態到WRITE狀態中，去做找column的動作，而等待counter為4時就會進到BVAL的狀態中，在此狀態中因為與DRAM的寫入或讀取沒有關係所以不需要有5個cycle的delay，此狀態是為了告訴CPU已做完寫入的動作，在此狀態中只要BVALID & BREADY同時為1時就會回到IDLE的狀態去等待下一次CPU要做讀取或寫入的動作。 |

|  |
| --- |
| DRAM Execution READ (Row Hit) |
|  |
| 上圖為第二次以上寫入或讀取DRAM的過程，在IDLE狀態等待AWVALID & AWREADY與ARVALID & ARREADY的同時也會判斷是否row hit，如果row hit的情況下就不用進到ACT的狀態下去做找row的找尋，會直接從IDLE的狀態到READ狀態中，去做找column的動作，而等待counter為4時就會進到BVAL的狀態中，此狀態是為了告訴CPU已做完讀取的動作，在此狀態中只要RVALID & RREADY & DRAM\_VALID同時為1時就會回到IDLE的狀態去等待下一次CPU要做讀取或寫入的動作。 |
| WFI Instruction |
|  |
| 圖1 |
|  |
| 圖2 |
| 上圖1為wfi指令的執行過程，10500073就是WFI的assemble code，當CPU開始執行WFI指令後，CPU會進入WFI mode，會暫停執行後續指令，並且等待interrupt，在interrupt被拉起後，PC值會跳到mtvec\_PC的地址(interrupt後指令的起始PC)，可以由上圖2看到inst\_PC跳到1\_0000與圖1的mtvec\_PC值相同，此時CPU會結束WFI mode，去執行interrupt要執行的指令。 |

|  |
| --- |
| Sensor Control |
|  |
| 圖1 |
|  |
| 圖2 |
|  |
| 圖3 |
| 上圖為sensor control的執行過程，當sensor control寫入地址為1000\_0100(此地址為sctrl\_en的控制訊號)且寫入的值為1時(表示此時可以生成新的資料了)，代表sensor control被啟動了，而sensor control每1024個cycle會生成一個新的data存在自己的memory中，而當local memory滿了的時候會拉起sensor control interrupt，同時將sctrl\_en的訊號降下，並將sensor control local memory的資料寫入DM當中，而在sensor control寫入地址為1000\_0200(此地址為sctrl\_clear的控制訊號)且寫入的值為1時，sensor control interrupt會降為0。 |

**Verification**

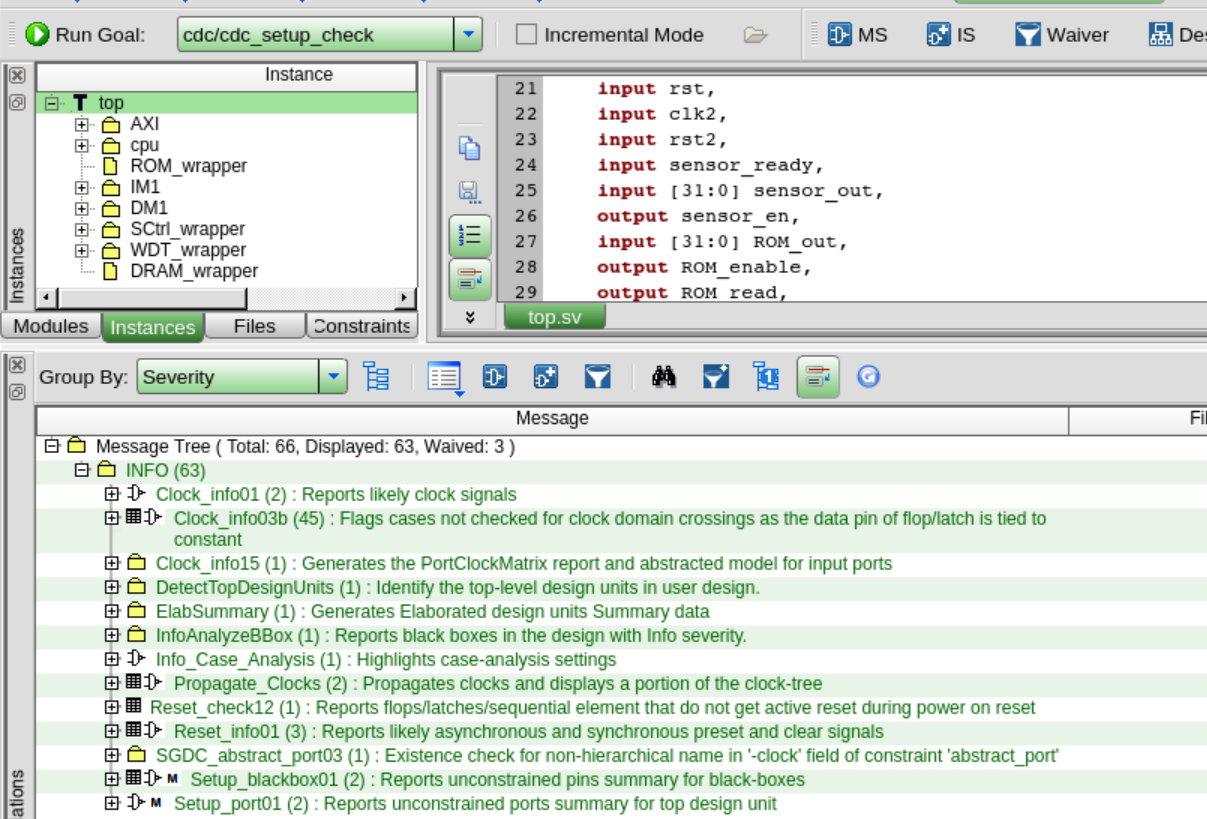
* AXI VIP

|  |
| --- |
|  |
| 圖1 |
|  |
| 圖2 |
| 上圖1為這次AXI的架構圖，在這次AXI的實作中基本觀念與上次的相同，因為在上次的作業實作中對AXI的理解錯誤只寫了一個AXI的sv檔，而在這次的作業中將AXI改為5個channel，2個用來讀資料，3個用來寫資料，並且透過2個arbiter來控制要讀取哪個slave，這次的作業只需增加4個slave的輸出及輸入，並且需更改arbiter的判斷，需要依照slave的優先權去做更改，將優先權高的slave優先判斷。 |

* CDC Spyglass

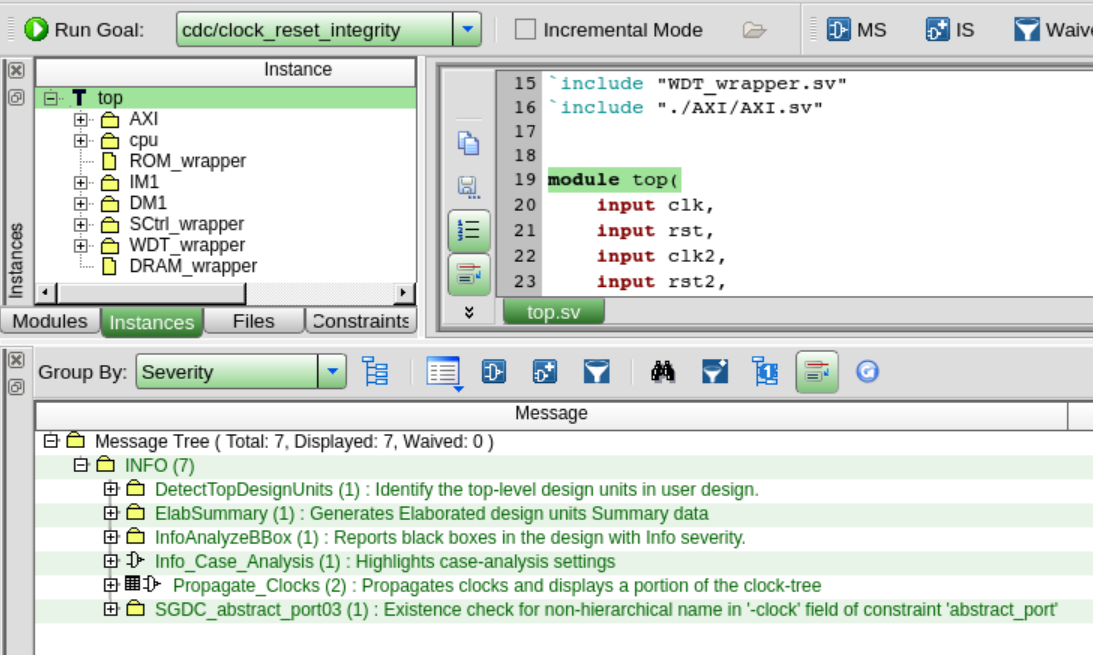
1. CDC Setup Check

0 warning



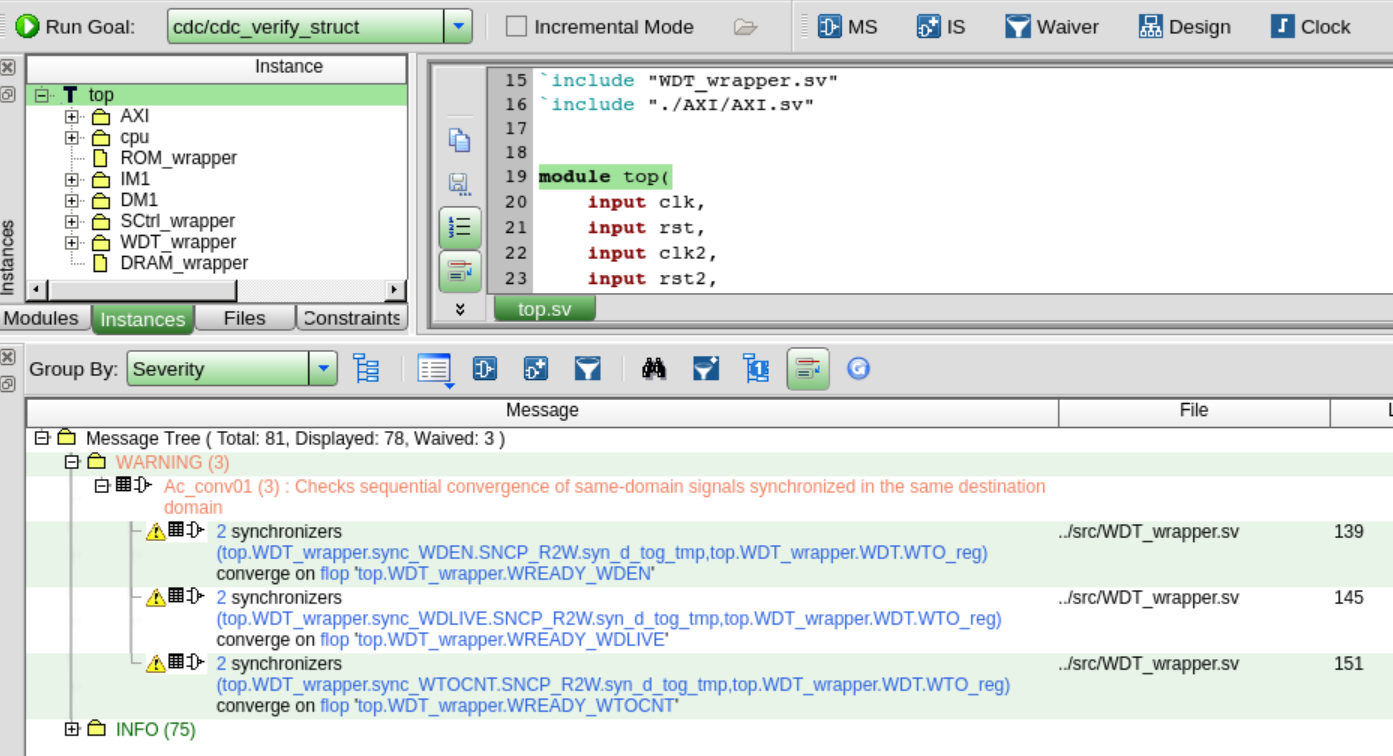
1. Clock Reset Integrity

0 warning

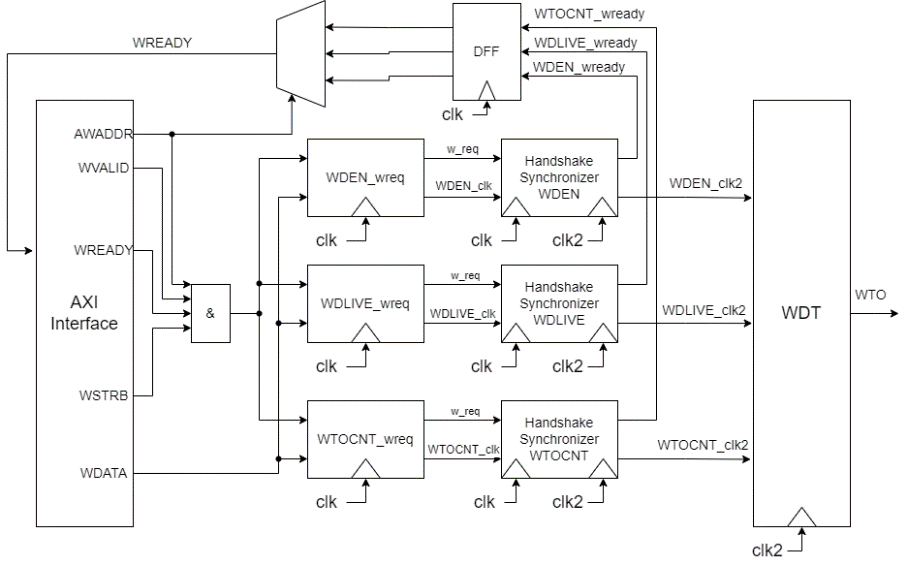


1. CDC Verify Struct

3 warning : Checks sequential convergence of same-domain signals synchronized in the same destination domain



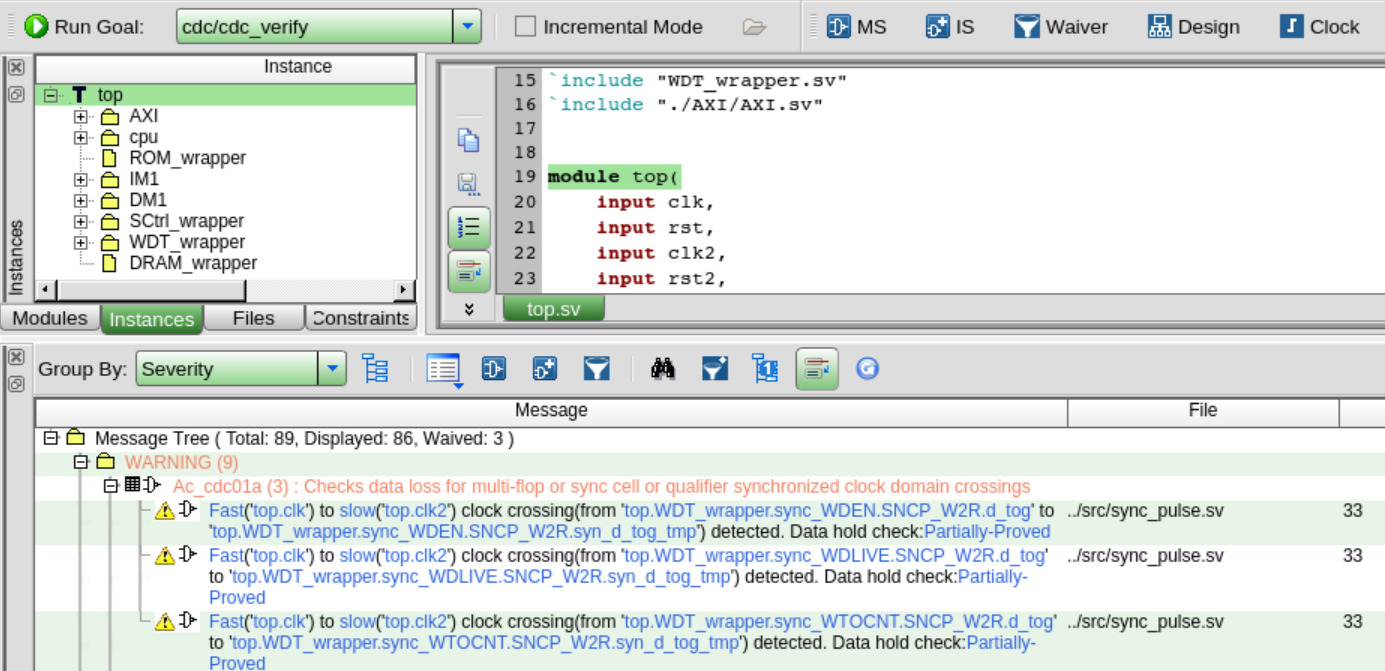
以下為WDT wrapper的架構圖，spyglass報warning的位置位於紅框處，原因是三個XXX\_wready訊號分別從三個暫存器後會一起送入MUX的combinational電路，形成convergence，但若是實際應用上這三個訊號不會同時變化，就不會造成CDC問題。而在本次設計中這三個wready訊號確實不會同時變化，因為CPU每次只會要求寫入WDT的一個位址空間，所以一次也只有一個wready訊號會改變，基於這個原因，我們認為這個waring可以不需要解決。



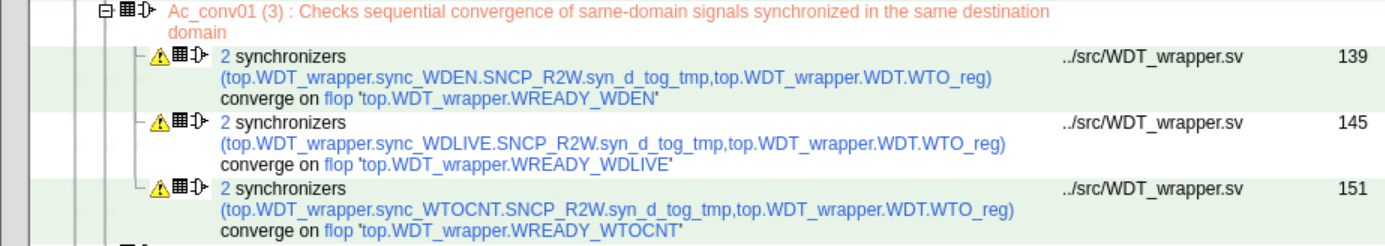
1. CDC Verify

9 warning:

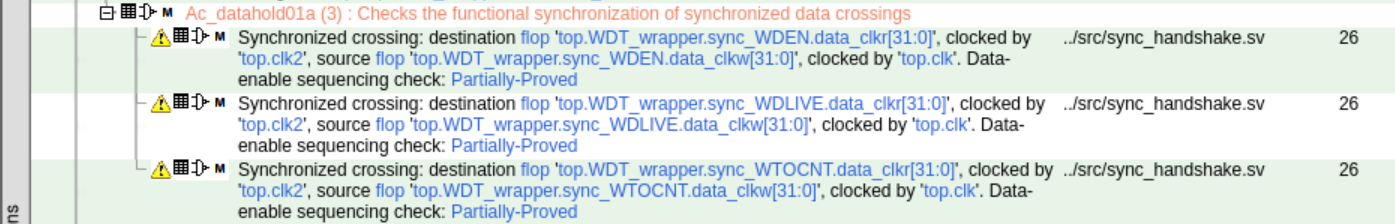
1. Ac\_cdc01a



1. Ac\_conv01



1. Ac\_datahold01a

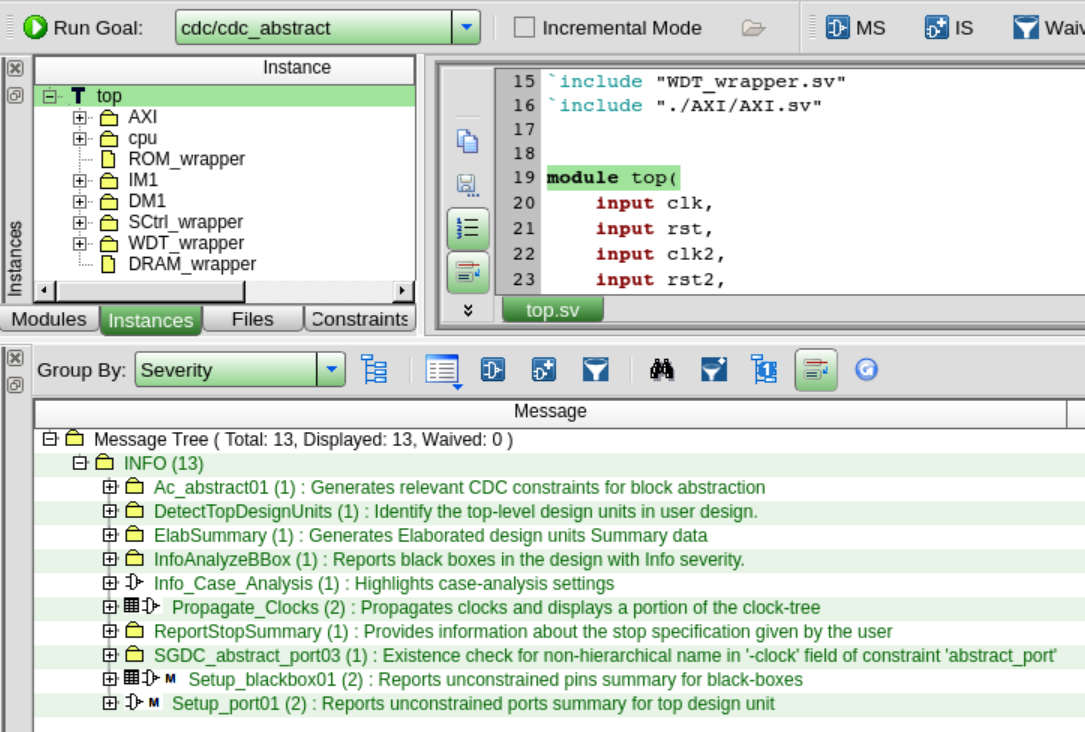


第1點和第3點的warning屬於partially proved，我們有試著下”set\_parameter fa\_atime 100”這個指令來延長驗證時間，最後結果也是partially proved，可以保證在time 100以內不會發生CDC問題。

第2點和CDC verify struct是相同的warning，已在CDC verify struct中詳細說明。

1. CDC Abstract

0 warning



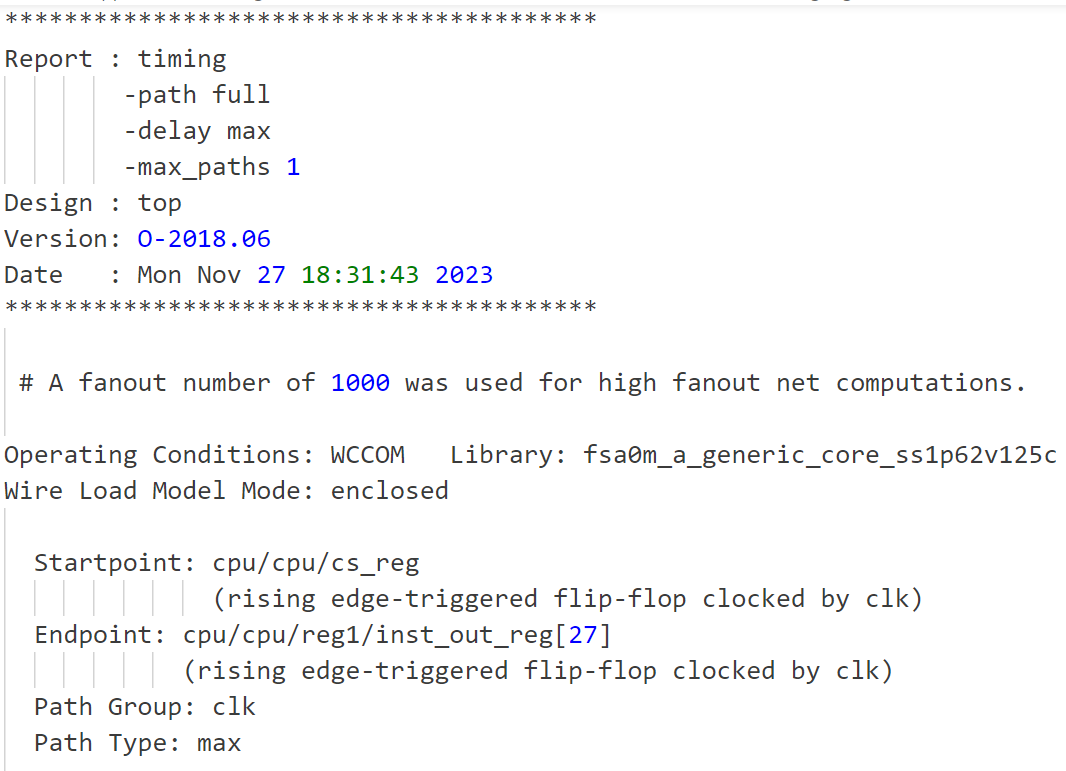
**Synthesize Result**

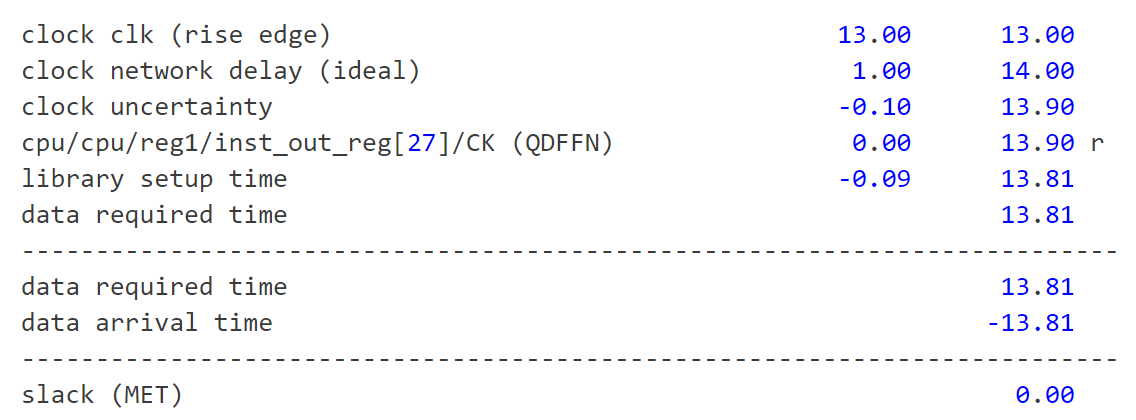
* Area



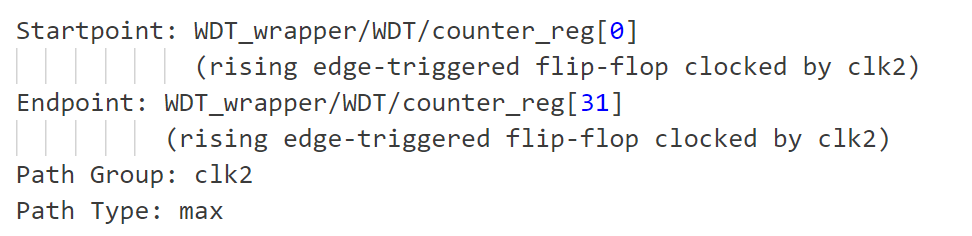
* Timing

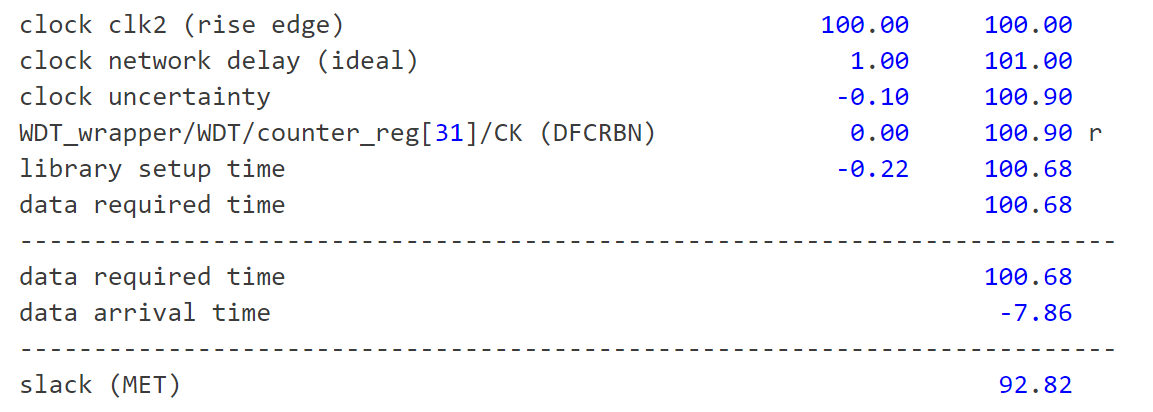
clk domain:





clk2 domain:





* Simulation result

|  |  |
| --- | --- |
| Program0 | Program1 |
|  |  |
| Program2 | Program3 |
|  |  |
| Program4 | Program5 |
|  |  |

**Problems to answer**

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

ANS1:

Mcycle是一種計數器（Machine Counter），是64bit的CSR暫存器，用於計算週期的數量，在每執行一個clock cycle後都會自動增加，可以用來計算出CPU的效能及程式執行的時間。

Timer的clock是固定的，且可以與CPU的clock不同，因此在使用上可能需要做CDC處理。Timer的值是一個只會增加的計數器，表示自某一時間點以來的clock週期數，而其主要的功能為生成中斷，通常用於執行特定的任務，像是特定時間需觸發的事件等，多用於時間相關的功能。

1. What is “Potential Qualifier” in Spyglass?

ANS2:

Potential Qualifier是指在進行Spyglass設計分析時，偵測到問題或一些不合乎規範問題的訊號，像是在這次作業中我們可能會有Metastability、Reconvergence、Data hold problem(資料丟失問題)等timing的問題，而不僅僅是時間的問題像是合成、area、coding等問題也會在Spyglass顯示出來，告知設計者一些要注意及改善的問題，有助於提高設計的可靠性及性能。

**Lesson learned**

1. 在這次的作業當中我們做了AXI的改進，在上次的作業當中因為我們對AXI概念的不足，所以是以一個sv檔來完成它的功能，而在經過老師上課過程的講解，才知道AXI的功能與應該要如何去實現，所以在這次的實作過程中我們有正確的把AXI分成5個channel跟2個arbiter，用arbiter去控制AXI應該如何再master與slave之間傳輸資料，然後把資料跟地址分成不同的channel做傳輸。
2. 我們在實作過程中遇到了不少的問題，最重要的部分是CPU，因為在之前的作業當中有很多的問題是不會遇到的，像是DRAM的delay或是寫回資料的延遲等，除了上述問題外，我們還需要增加I/O與interrupt的處理，此時就需要增加CSR指令，很多其實CPU應該要具備的功能我們其實在原先的CPU沒有執行的很好，而這次CPU又需要去切換模式、功能，來處理更多的問題，導致在這次實作的過程中遇到了很多的問題。
3. 這次在合成時有發生因為訊號輸入和輸出都沒經過暫存器，導致合成時路徑會加上input external delay和output external delay，會大幅降低電路的速度，導致我們剛開始就算cycle time合到20ns依然會有timing violation，所以我們後來將輸入訊號和輸出訊號都先送入暫存器，再拿來使用，這樣就可以避免被input external delay或output external delay影響，合成路徑上就只會有我們電路真正的路徑，不會加上input external delay和output external delay。