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:陸啟倫

Student ID:N26114950

**outline**

**1.Summary 3**

2.Major problem encountered and resolution3

3.Block diagram4

4.Module introduction6

**5.State diagram introduction 9**

(a)SCtrl Wrapper**9**

(b)WDT Wrapper**10**

(c)AXI Write arbiter**11**

(d)AXI Read arbiter**12**

6.Waveform 13

7.CDC problem 18

8.Spyglass 24

9.Simulation result 27

10.Superlint result 34

11.AXI ABVIP36

12.Layout37

13.Performance 42

14.Lesson learned 44

1. **Summary**

這次作業將CPU加上了CSR的相關指令，加上了Sensor及WDT來模擬實際CPU在運作時，可能因內部或外部的原因，導致要先終止當下執行的程式，先去執行某部分程式的狀況。處理了WDT和WDT Wrapper交換資料時的CDC問題。將上述要求完成後，成功完成booting、rtl prog0~prog4、syn prog0~prog4、及APR prog0~prog4的模擬。合成及APR使用的clock period均為10ns。

作業分工為陸啟倫(100%)。

1. Major problem encountered and resolution
2. 中斷發生時，stall也同時發生，導致中斷訊號未正確執行的情況

Sol: 在中斷發生時，需考慮當下是否有發生stall訊號。若有發生stall訊號，CPU所有管線暫停，CPU會無法更新PC，這代表CPU在stall時會看不到中斷訊號的發生。我新增了存取中斷的暫存器，在中斷和stall同時發生時，將中斷存入暫存器中，確保stall結束後，CPU能看到中斷訊號。

1. Spyglass檢查過程中，報錯許多cdc\_setup\_check相關的error

Sol:在做檢查時，需要先確保整個設計都先改成同步reset以及case或if中的每個訊號是否有寫滿，若沒寫滿則可能會有error的產生。這部分的原因可能是因為latch的產生而導致。

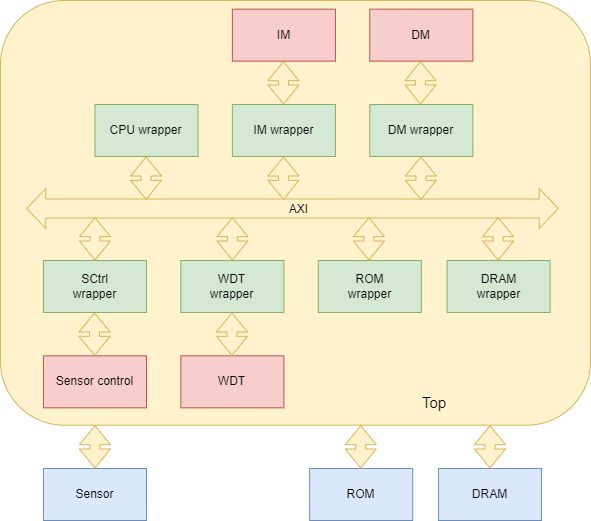
1. 在產生中斷訊號時，中斷訊號會一直拉高，使CPU一直中斷的狀況。

Sol:在產生中斷訊號時，還需要考慮MIE暫存器的數值。因為當中斷發生 時，MIE會被寫入成0，這樣就能確保中斷訊號只會被拉高一次。

1. 在讀取0x1000\_0000~0x1000\_03ff的資料時，未正確從bus讀取

Sol: Data cache需要新增cacheable的判斷，要確保在讀取這段address中的資料時，cache一定會read miss，才能確保會從bus拿回資料。

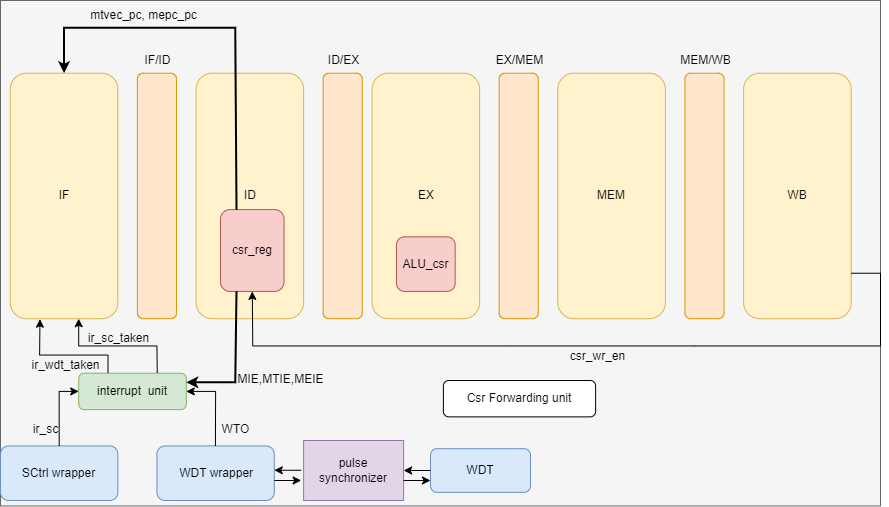
1. **Block diagram**
2. **top的block diagram**



說明:

top內部包含的module如上所示。各wrapper都需透過AXI來進行訊號溝通，以互相傳遞資料。WDT wrapper負責傳送WDLIVE、WTO、WTOCNT、WTO。CPU需固定傳送WDLIVE來告訴WDT ，CPU還在正常運作，若CPU沒有及時restart WDT內部的計數器，則WDT會傳送WTO訊號來告訴CPU要進行中斷。SCtrl wrapper負責傳送Sctrl\_en、Sctrl\_addr、Sctrl\_clear訊號給Sensor Control，當Sctrl\_en傳至Sensor Control，內部counter會開始計數，並傳遞對應的資料回來，過程中若收到Sctrl\_clear則會歸零計數器。當傳送到64筆資料時，則會拉起中斷訊號，通知CPU要進行中斷。

**(b)CSR及中斷相關unit的block diagram**

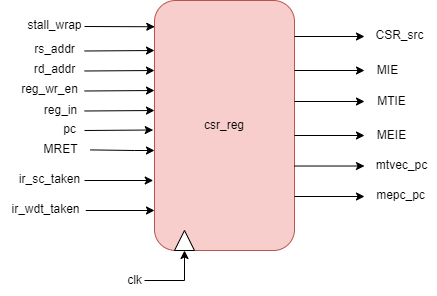


說明:

上圖包含執行CSR指令的相關單元、回饋單元、中斷單元、及會產生中斷訊號的兩個單元。若在執行過程中遇到CSR相關指令，則會先根據CSR的地址，在ID中的csr\_reg提取對應的暫存器數值，接著進入EX，會在ALU\_csr中進行對應的CSR指令運算，並將運算結果一路往後傳到WB，接著會根據csr\_wr\_en訊號的數值，來決定是否要將運算數值寫回crs\_reg，整體的行為如同CPU在處理一般的R-type指令，但在forwarding的判斷上有些差異而已。處理中斷的元件，interrupt unit，會收到來自SCtrl wrapper或WDT wrapper的中斷訊號，這些中斷訊號會由WDT和Sensor control各自的計數器數到特定數值時產生，當interrupt unit吃到外部的中段訊號時，會再根據MIE、MTIE、MEIE的數值來決定是否真的能產生中斷，即ir\_sc\_taken和ir\_wdt\_taken訊號。這麼做的目的是確保中斷只能發生一次，不會有中斷不停發生的狀況。當IF中的PC counter收到ir\_sc\_taken或ir\_wdt\_taken時，則會跳到interrupt service routine去執行相關的程式，完成中斷CPU的目的，而中斷的發生也會使csr\_reg將下個要執行的指令PC值，存放在mepc暫存器中，以確保中斷結束後，IF中的PC counter能回到剛剛執行的指令的下一個指令，繼續運作。值得注意的是在WDT wrapper和WDT之中有經過一個pulse synchronizer，由於WDT操作的頻率相較其他元件慢很多，所以需要通過此單元來解決WTOCNT、WDEN、WDLIVE訊號在傳送到WDT時發生的亞穩態問題，以確保WDT能夠收到穩定且正確的訊號。

1. **Module introduction**

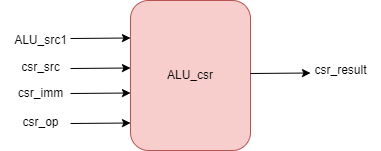
**(a)csr\_reg**



說明:

csr\_reg是專門存取csr暫存器的Regfile，內部包含9個csr暫存器，分別為mstatus、mie、mtvec、mepc、mip、mcycle、ministret、mcycleh、minstreth。這九個暫存器有不同的目的，mstatus會追蹤當前的控制狀態，當CPU發生異常時，會根據中斷訊號改寫mstatus中特定的數值。當中斷發生時，會將MIE的數值存入MPIE、將0存入MIE、將當前的privilege mode存入MPP。當中斷結束時，會將1寫入MPIE、將MPIE寫入MIE、將當前的privilege mode存入MPP。mie會存放中斷的enable資訊。mtvec會存放interrupt service routine的pc位置，而且無法被更改，確保中斷時能夠跳去正確的pc。mepc會存放中斷發生時，正運行到的pc數值，確保中斷結束後能回到原處繼續執行指令。mip會存放不同類型中斷的等待狀態。mcycle和mcycleh會存放系統目前運算的cycle數，兩個reg分別對應[63:32]和[31:0]的數值。ministret和minstreth會存放系統目前執行的指令數，兩個reg分別對應[63:32]和[31:0]的數值。

1. **ALU\_csr**



說明:

ALU\_csr是專門處理CSR指令運算的單元，有三個運算元分別為ALU\_src1、csr\_src、及csr\_imm，ALU\_src1即rs1、csr\_src是從csr\_reg中讀取的數值、csr\_imm是CSR指令的[31:20]部分。ALU\_csr會根據csr\_op來決定現在要進行的是甚麼運算，總共能夠執行6種運算，為CSRRW、CSRRS、CSRRC、CSRRWI、CSRRSI、CSRRCI。結束運算後，則會輸出crs\_result。

1. **Csr Forwarding unit**

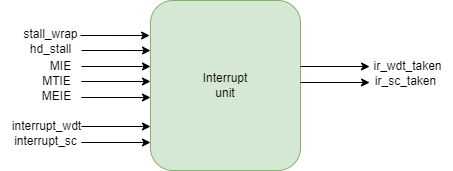
**一張含有 文字 的圖片

自動產生的描述**

說明:

Csr Forwarding unit負責處理Csr相關暫存器的回饋問題。會根據各個stage的opcode和csr address來決定是否需要回饋後一級的數值到前方做運算。總共會產生兩種回饋可能，分別為WB拉回MEM、MEM拉回EX，分別對應FW\_csr\_EXMEM、FW\_csr\_MEMWB。由於Csr暫存器中有些是不可被隨便回饋的，例如mtvec和mip，所以當這些暫存器出現在rd，且與上一級的csr暫存器相依時，回饋會失效。

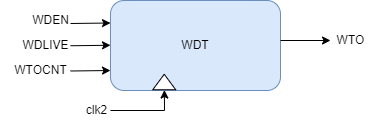
1. **Interrupt unit**



說明:

Interrupt unit負責處理中斷事宜。當中斷單元收到來自SCtrl wrapper或是WDT wrapper的中斷訊號時，會再根據MIE、MTIE、MEIE訊號來決定是否能發生中斷。若收到interrupt\_sc、且MIE和MEIE均為1時，代表可以產生ir\_sc\_taken訊號使PC值跳到interrupt service routine。若收到interrupt\_wdt、且MIE和MTIE均為1時，代表可以產生ir\_wdt\_taken訊號使PC值跳到interrupt service routine。值得注意的是需要考慮stall\_wrap和hd\_stall訊號，stall\_wrap代表來自cache的stall訊號，hd\_stall代表來自CPU內部控制危障的stall訊號。當任一Stall訊號拉起時，CPU的管線會暫停，這代表此時的PC值是無法更改的，換言之CPU看不到interrupt訊號的發生。為了解決這種stall和中斷同時發生的情況，我們需要兩個暫存器分別在stall發生時，將中斷訊號存取起來，以確保Stall結束時，還有中斷訊號被存在暫存器當中，才能使中斷訊號更改PC值，以完成任務。

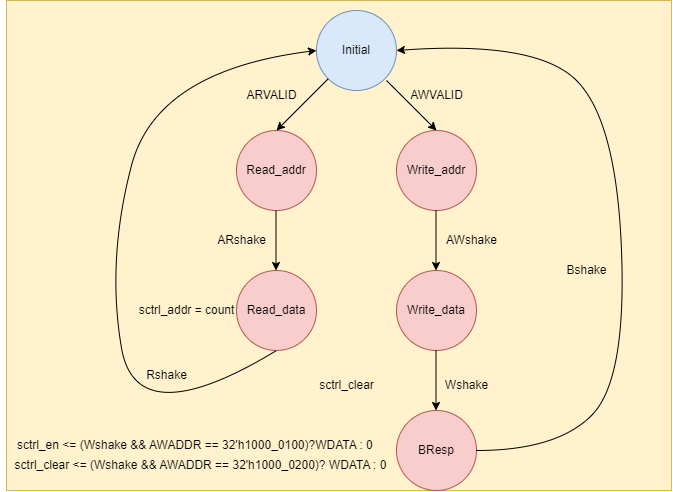
1. **WDT**

****

說明:

WDT內部存在一個計數器，當收到WDEN後則會開始計時，若在計數過程中收到WDLIVE則會歸零，若內部計數器數到WTOCNT的數值時，則會輸出WTO。

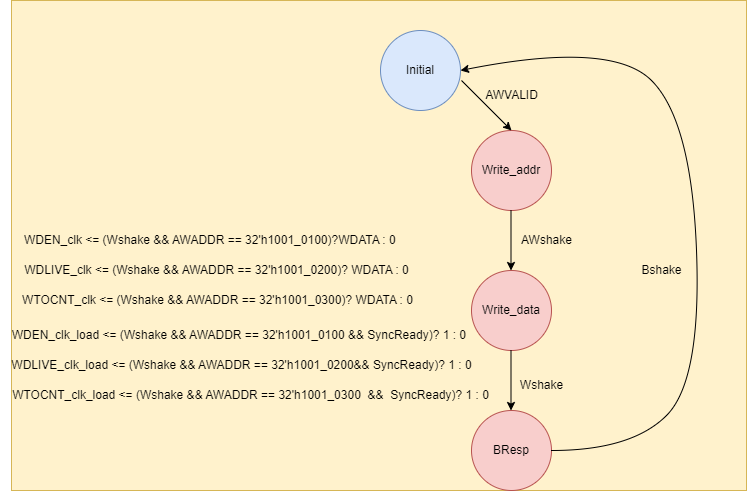
1. **State diagram introduction**
2. **SCtrl wrapper**



說明:

上圖為SCtrl wrapper中的狀態機，在起始狀態時會根據ARVLAID和AWVALID來判斷要進入Read\_addr或是Write\_addr，若進入Read\_addr，則會等待AR訊號握手，再進入Read\_data，由於我們要從Sensor control讀取的數值可以在1個clk內回來，所以只要在Read\_data再給sctrl\_addr即可，而內部的計數器會根據R訊號的握手來將存取數值加一。接著等待R訊號握手後回到初始狀態。若進入Write\_addr，則會等待AW訊號握手，再進入Write\_data，在Write\_data狀態時，會根據W訊號是否握手及AWADDR是否為32’h1000\_0100或32’h1000\_0200來決定是否要將WDATA寫入sctrl\_en或sctrl\_clear。接著等待W訊號握手後，再進入BResp訊號，再等待B訊號握手後，回到初始狀態。

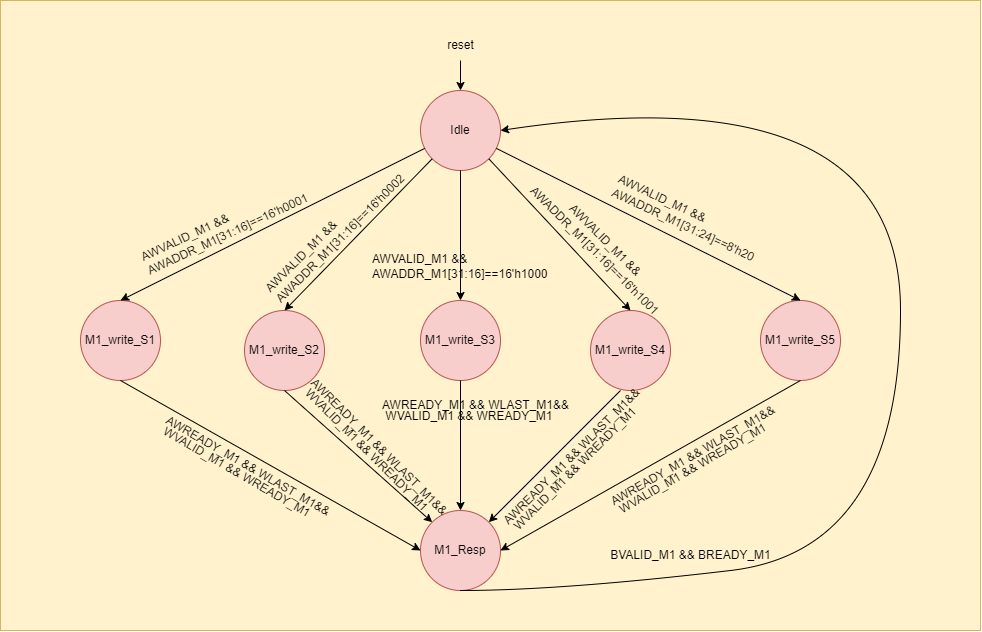
1. **WDT wrapper**



說明:

上圖為WDT wrapper中的狀態機，由於我們不需要從WDT中讀取數值，所以只需要有寫入部分的狀態機。在初始狀態時，若吃到AWVALID訊號，則會進入Write\_addr狀態，接著等待AW訊號握手後再進入Write\_data狀態，在Write\_data狀態中，會根據W訊號是否握手及AWADDR是否為32’h1001\_0100、32’h1000\_0200、或32’h1001\_0300來決定是否要將WDATA寫入WDEN\_clk、WDLIVE\_clk、或WTOCNT\_clk (clk表示這三個訊號目前操作在clk的頻率，還未經synchronizer同步到clk2)。同時也會根據W訊號是否握手及AWADDR是否為32’h1001\_0100、32’h1000\_0200、或32’h1001\_0300來決定WDEN\_clk\_load、WDLIVE\_clk\_load、和WTOCNT\_clk\_load對應的數值，這三個訊號會被用來作為synchronizer的load訊號。接著，W訊號握手後則會進入BResp狀態，再等待B訊號握手後則可回到初始狀態。

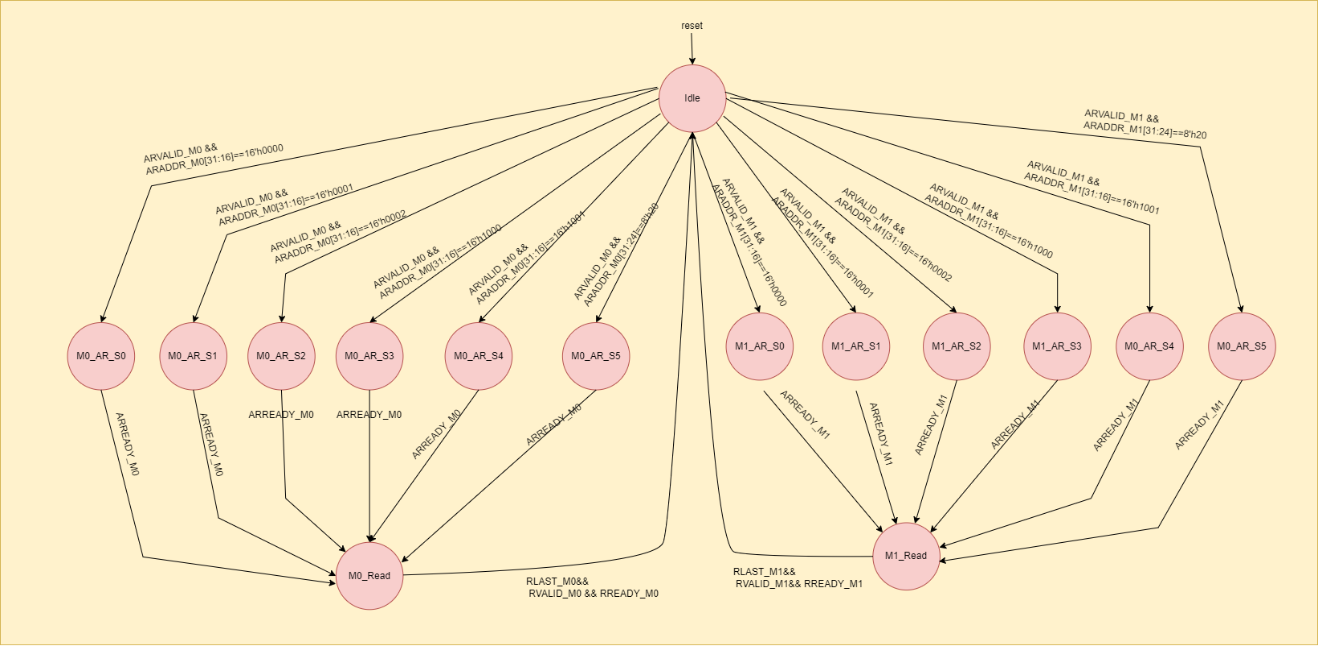
1. **AXI Write arbiter**

****

說明:

上圖為AXI中控制寫入的狀態機，初始狀態時須根據AWADDR\_M1的數值來決定要將資料寫入slave1、slave2、slave3、slave4或slave5，再待AWVALID訊號後即可至對應的狀態。接著進入M1\_write\_S1、M1\_write\_S2、M1\_write\_S3、M1\_write\_S4或M1\_write\_S5狀態。在這些狀態時，AXI會打開AW及W通道，並將AWID、AWLEN、AWADDR、AWSIZE、AWVALD、AWREADY等AW通道相關資訊及WID、WDATA、WLAST、WVALID、WREADY等W通道相關資訊傳送至對應的master或slave。接著待AW通道及W通道都握到手後，則可進入M1\_Resp。在M1\_Resp中，AXI會打開B通道，並傳送BID、BRESP、BVALID、BREADY等B通道相關資訊至對應的slave或master，待B通道握手後則可再回到初始狀態

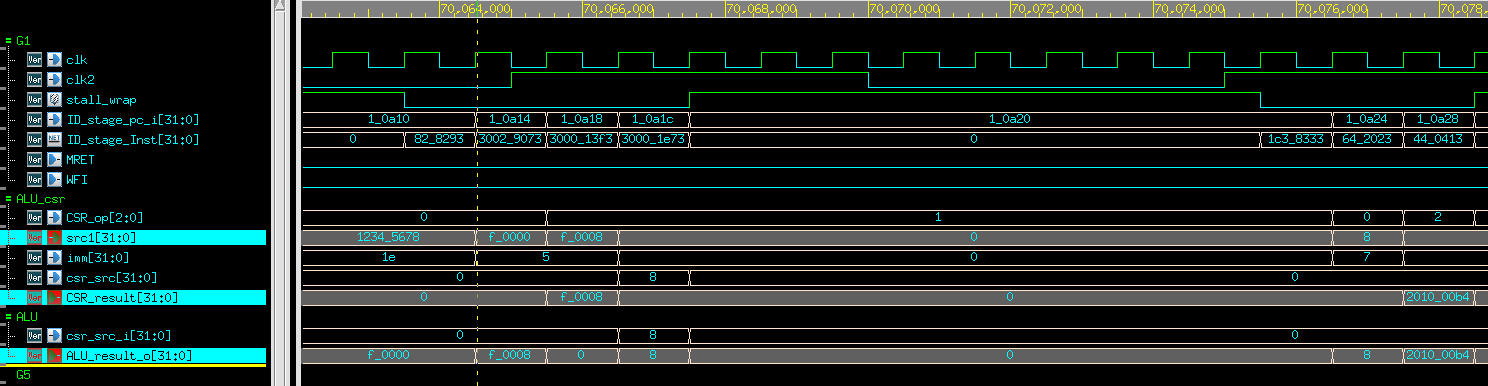
1. **AXI Write arbiter**

****

說明:

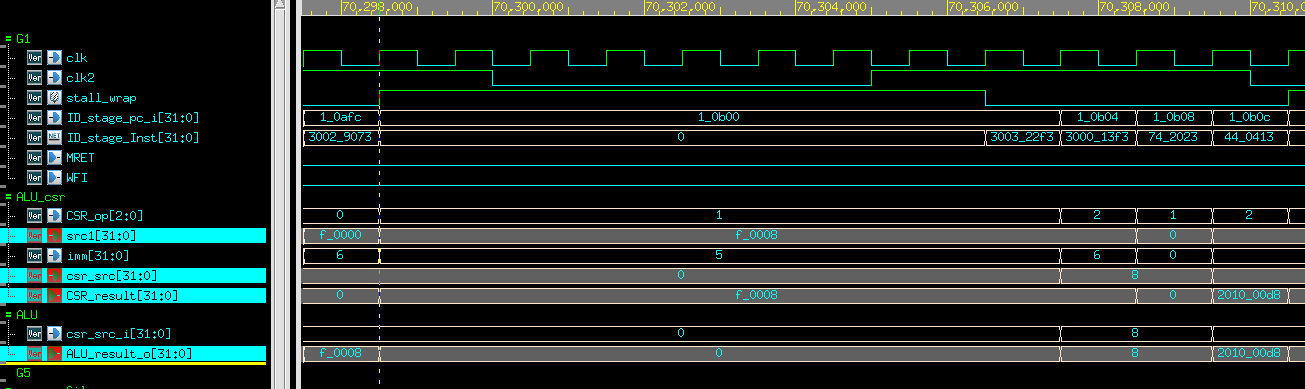
上圖為AXI中負責控制讀取的狀態機，起始狀態時需根據送入AXI的ARADDR\_M1，ARADDR\_M0來判斷這個地址是要送到哪一個slave當中，再待對應的ARVALID後即可進入對應的AR狀態。進入AR狀態時，AXI則會將AR通道打開，並將ARID、ARLEN、ARADDR、ARSIZE、ARVALD、ARREADY等資訊傳送至對應的slave或master。接著開始等待slave傳送ARREADY，代表slave已經準備接收資訊，兩方握手後，資料就能成功傳輸並進入下個狀態(M0\_read或M1\_read)。進入Read狀態後，AXI則會打開Read通道，並將RID、RDATA、RRESP、RLAST、RVALID、RREADY等訊號傳送至對應的slave或master。由於站在CPU讀取資料的角度，可能會有來自slave0、slave1、slave2、slave3、slave4、slave5的情況，所以需要根據AXI在接收到來自不同Master的ARID時，做不同的高位元的編碼。待slave接收到ARID後則會將高位元的編碼存起來，待傳送RDATA時能順便加在RID的高位元部分，以便AXI分別資料要送至哪個Master。接著等待RVALID、RREADY、RLAST訊號，以確保握手成功及傳送的是最後一筆資料。才能回到初始狀態。

1. **Waveform**
2. CSRRW



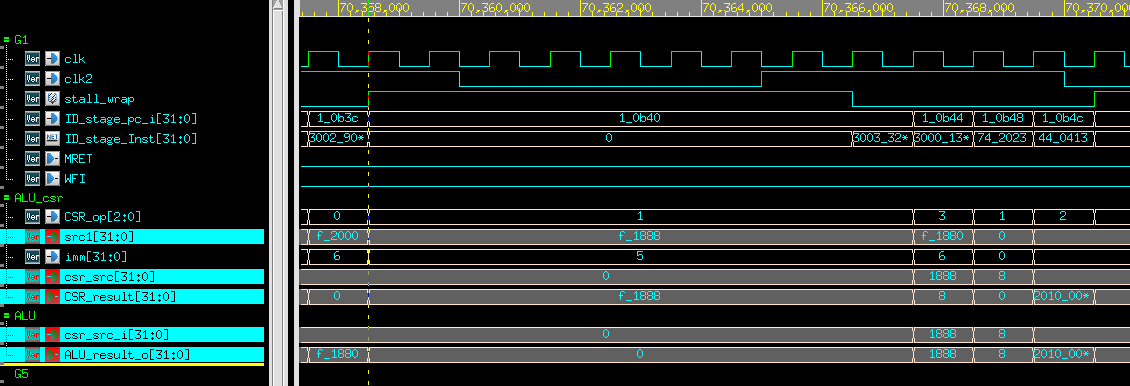
說明:10a14、10a18、10a1c均為CSRRW指令，CSRRW指令會執行兩種操作，分別為rd = csr、csr = rs1，上圖中的紅色框框為將src1數值傳送給CSR\_result(對應csr = rs1的部分)。上圖中的藍色框框為將csr\_src數值傳送給ALU\_result\_o(對應rd = csr的部分)。

1. CSRRS



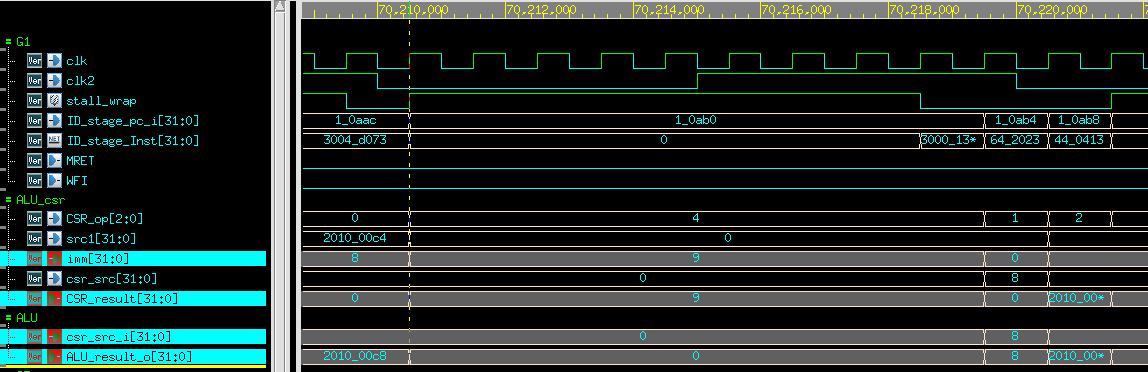
說明:10b00為CSRRS指令，CSRRS指令會執行兩種操作，分別為rd = csr、csr = rs1 | csr，上圖中的紅色框框為將src1和csr\_src做or運算後將結果存在CSR\_result(對應csr = rs1|csr的部分)。上圖中的藍色框框為將csr\_src數值傳送給ALU\_result\_o(對應rd = csr的部分)。

1. CSRRC



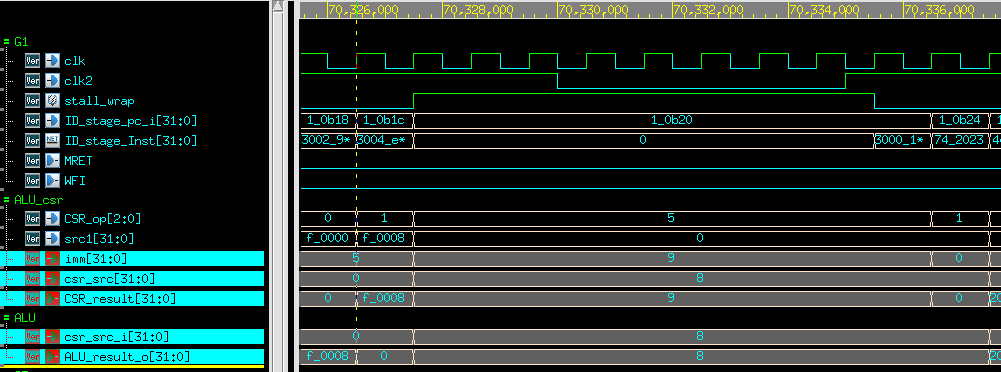
說明:10b40為CSRRC指令，CSRRC指令會執行兩種操作，分別為rd = csr、csr = csr &(~rs1)，上圖中的紅色框框為src1和csr\_src的數值，完成csr\_src&(~src1)的運算後，將結果存在CSR\_result(對應csr = rs1|csr的運算)。上圖中的藍色框框為將csr\_src數值傳送給ALU\_result\_o(對應rd = csr的運算)。

1. CSRRWI



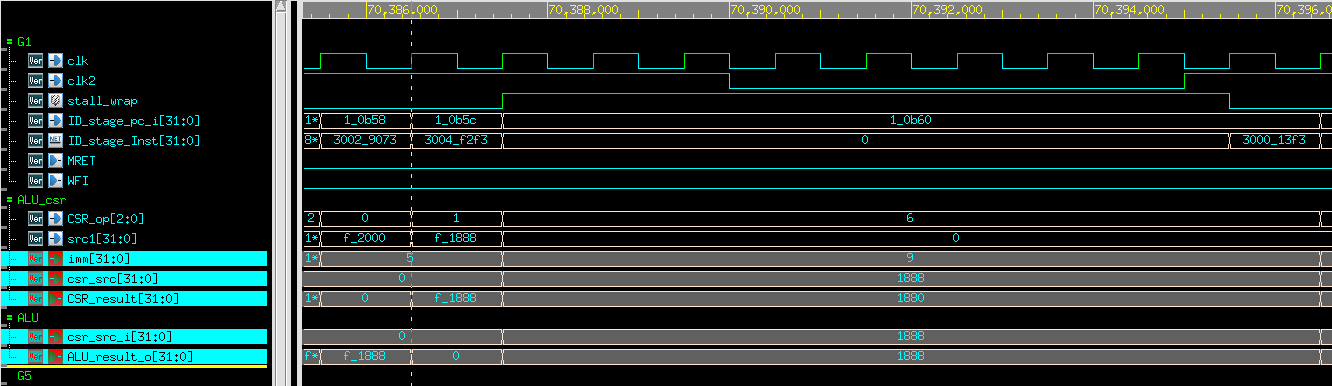
說明:10ab0為CSRRWI指令，CSRRWI指令會執行兩種操作，分別為rd = csr、csr = uimm(zero-extend)，上圖中的紅色框框為將imm的數值傳送到CSR\_result(對應csr = uimm(zero-extend)的運算)。上圖中的藍色框框為將csr\_src數值傳送給ALU\_result\_o(對應rd = csr的運算)。

1. CSRRSI



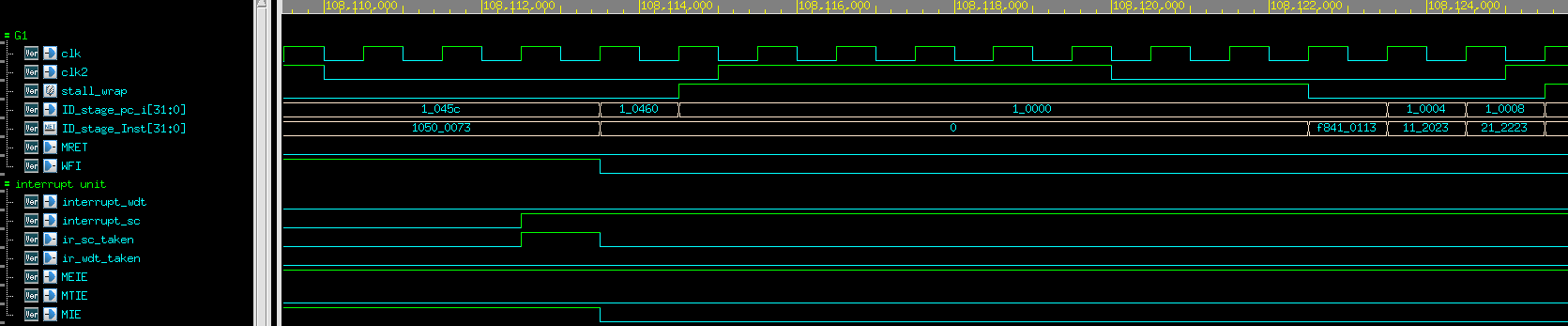
說明:10b20為CSRRSI指令，CSRRSI指令會執行兩種操作，分別為rd = csr、csr = csr | uimm(zero-extend)，上圖中的紅色框框為imm和csr\_src的數值，完成csr\_src|imm的運算後，將結果傳送到CSR\_result(對應csr = csr | uimm(zero-extend)的運算)。上圖中的藍色框框為將csr\_src數值傳送給ALU\_result\_o的部分(對應rd = csr的運算)。

1. CSRRCI

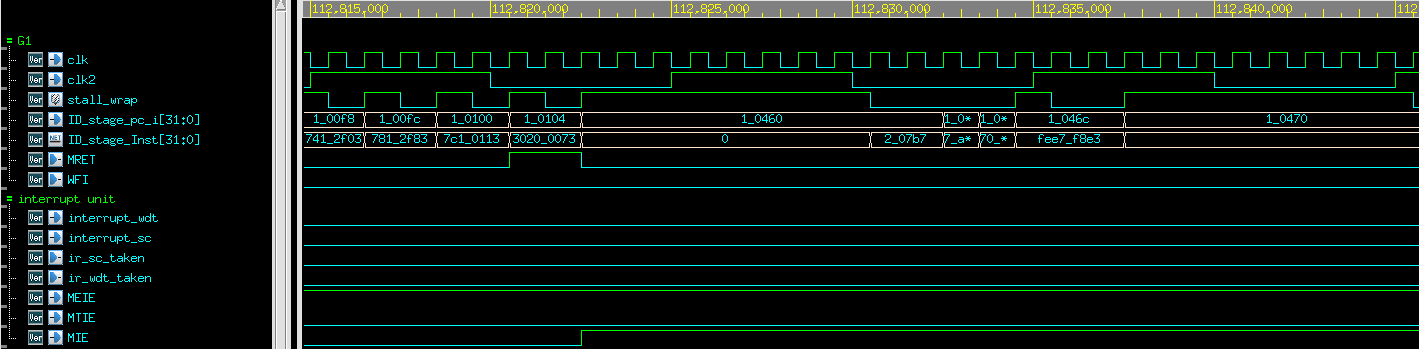


說明:10b5c為CSRRCI指令，CSRRCI指令會執行兩種操作，分別為rd = csr、csr = csr | ~uimm(zero-extend)，上圖中的紅色框框為imm和csr\_src的數值，完成csr\_src| ~imm的運算後，將結果傳送到CSR\_result(對應csr = csr | ~uimm(zero-extend)的運算)。上圖中的藍色框框為將csr\_src數值傳送給ALU\_result\_o的部分(對應rd = csr的運算)。

1. Interrupt from Sensor Ctrl



**(a)發生中斷**

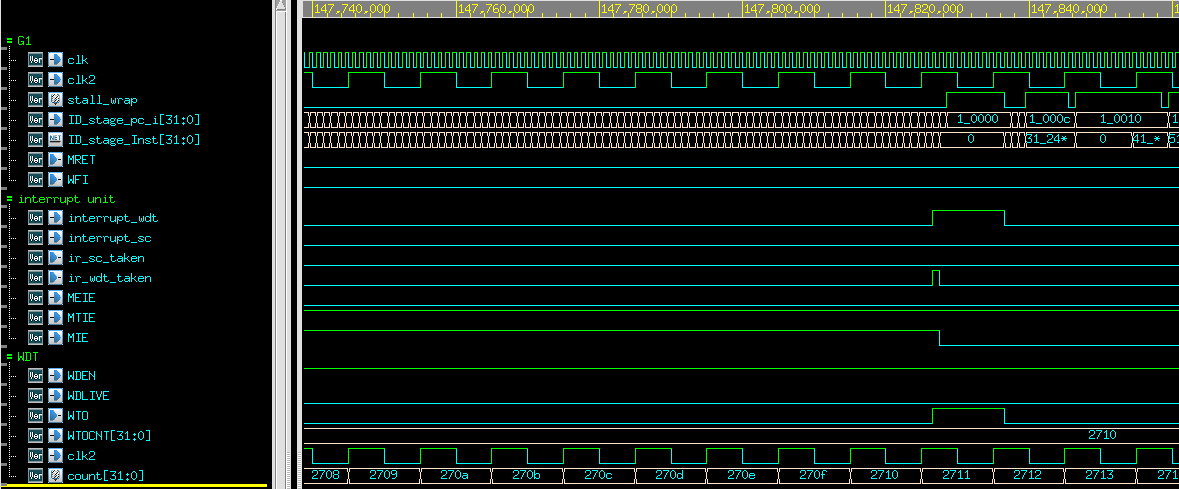


**(b)從中斷跳回**

說明:

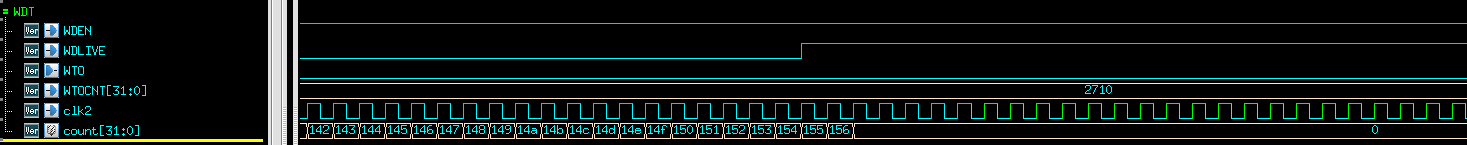
上兩圖為遇到WFI且Sensor Ctrl的中斷發生的狀況，橘色框框的部分為WFI指令與其PC值，可以發現WFI的訊號是拉高的，表示現在遇到WFI指令。藍色框框的部分有兩個訊號均為高電位，interrupt\_sc表示從Sensor Crtl送回的中斷訊號，ir\_sc\_taken為經過interrupt unit結合MIE和MTIE的數值判斷後，產生的中斷訊號(所以ir\_sc\_taken才是真正能使CPU中斷的訊號)。在ir\_sc\_taken拉起後，PC的數值會被更改到0x0001\_0000(紅色框框)，表示進入到interrupt service routine 。可以發現當中斷發生後，MIE的數值會被改為低電位(黃色框框)，以確保中斷不會持續發生。等待中斷執行結束後，則會遇到MRET指令並拉起MRET訊號，表示要從interrupt service routine跳回程式剛剛執行的地方，可以看到紫色框框的部分，PC值10460就是剛剛中斷發生時，程式所執行到的地方。

1. Interrupt from WDT



說明:

上圖為收到WDT發出的中斷訊號的狀況。當WDT收到WDEN=1時，代表內部的計數器要以clk2的時脈開始計時，當內部的計數器數到WTOCNT的數值時(紅色和白色框框)，WTO訊號就會在下一個cycle拉起。當WTO訊號拉起時，interrupt\_wdt也會是高電位，而interrupt unit會根據interrupt\_wdt、MTIE、MIE的數值來決定是否可拉起ir\_wdt\_taken訊號(藍色框框)。當ir\_wdt\_taken訊號拉起時，則會去更改PC counter的數值，所以可在下個cycle見到ID內的PC數值會是0x0001\_0000，即interrupt service routine(橘色框框)。

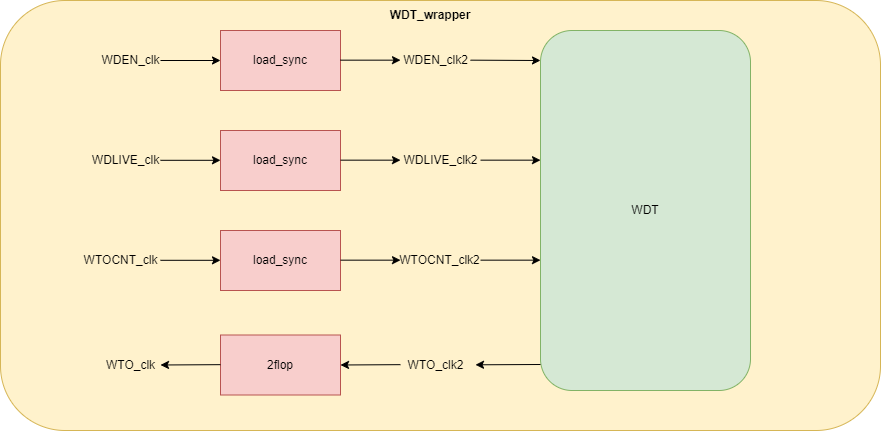


說明:

若WDT的內部計數器在計數過程中收到WDLIVE訊號(橘色框框)，則會將內部計數器的數值歸零(紅色框框)。若內部計數器未數到WTOCNT的數值(2710)，則不會有中斷訊號產生。

1. **CDC problem**

(a)architecture

****

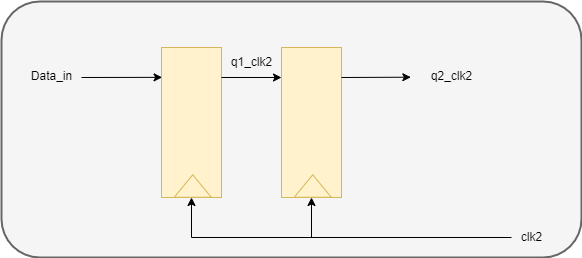
說明:

上圖為我處理CDC問題的硬體架構，我透過結合load訊號的synchronizer來處理位在clk domain的WDEN\_clk、WDLIVE\_clk、和WTOCNT\_clk訊號，經過load synchronizer後，可以得到WDEN\_clk2、WDLIVE\_clk2、WTOCNT\_clk2。這些訊號會成為WDT的輸入，來控制WDT內部的計數器，並在特定情況下輸出WTO\_clk2。由於WTO\_clk2是操作在clk2 domain，我會使用2flop synchronizer，也就是打兩拍的方式來傳送WTO\_clk2到clk domain，即WTO\_clk，並傳入到CPU中進行中斷判斷。

**問題:為什麼在同步WTO\_clk2訊號時單獨使用2flop synchronizer的方法呢?**

Ans:考量到要同步WTO\_clk2的狀況是從操作頻率慢的domain同步到操作頻率快的domain。在這種情況下，clk2 domain的訊號就算只有一個脈衝，clk domain也一定會看到，因為clk的操作頻率遠大於clk2。所以就單獨使用同步CDC中最簡單的架構，2flop synchronizer，就能滿足需求。

(b)2 flop synchronizer內部架構



說明:

上圖為一個2flop synchronizer的架構，由兩個flip flop組成。Data\_in是從clk domain來的資料，經過兩個操作在clk2 domain的flip flop後，得到q2\_clk2，即為同步到clk2後的Data\_in。

**問題:在使用此架構來做CDC同步時，clk和clk2的操作頻率是否有要求?**

Ans:在使用此架構做同步時，必須確定Data\_in的數值能夠穩定足夠的時間。

Data\_in在穩定的狀態時，若能穩定看到clk2 domain中3個edge，則可確定此架構能完成同步。若無法達到3個edge，就代表destination clk看不到Data\_in的變化，也就無法完成訊號同步。

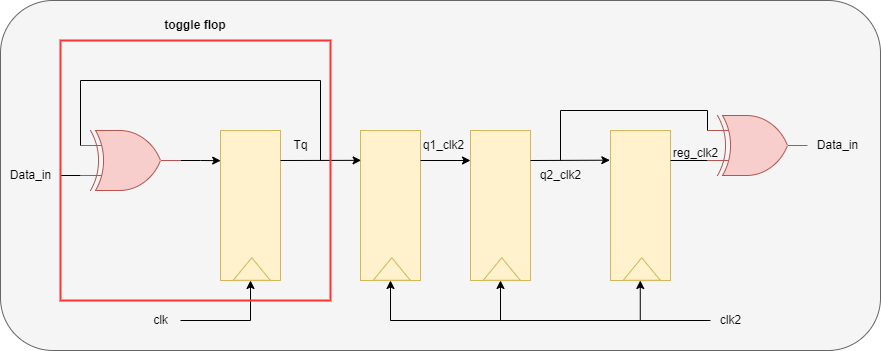
**問題:在使用此架構來做CDC同步時，Data\_in的bit數量是否有要求?**

Ans: 2flop synchronizer只能夠用來同步單bit的訊號，若要同步多bit訊號，則需要load訊號的幫助，才能確保所有bit能穩定，再來做同步。

**問題: 在使用此架構來做CDC同步時，Data\_in的來源有限制嗎?**

Ans: Data\_in需要為暫存器之輸出，若為組合邏輯之輸出，glitch現象會增加第一級flop產生亞穩態的機率，使得同步失敗的機率提高。

(c)sync\_pulse內部架構



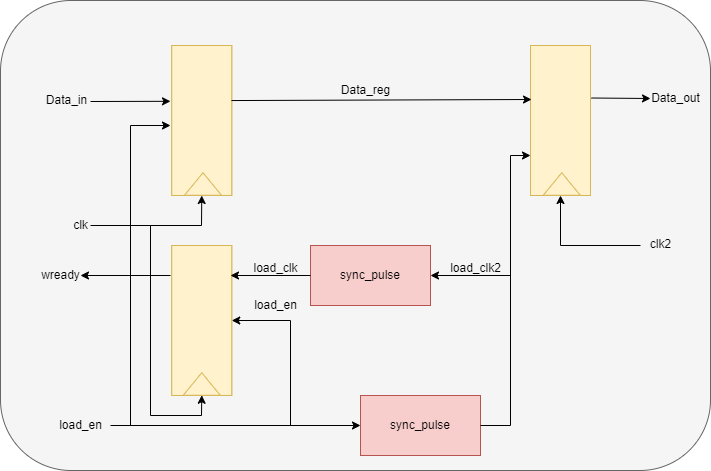
說明:

上圖為pulse synchronizer的內部架構，這個單元的主要目的是要使一個位於clk domain的脈衝同步到clk2 domain後，也會是一個脈衝。透過xor gate與flip flop組成的toggle flop，能夠將收到的pulse 訊號(data\_in)轉換成level 訊號(tq)，轉換為level訊號後，會通過兩個操作在clk2 domain的flip flop(2 flop synchronizer)，來將clk domain的level訊號同步到clk2 domain(q2\_clk2)。接著，再透過一個操作在clk2 domain的flip flop和xor gate來製造出clk2 domain的pulse訊號。

**問題:為什麼需要將clk的pulse轉換成level訊號再做同步處理呢?**

Ans:若我們直接將clk domain的pulse訊號做同步處理，2flop synchronizer可能會看不到這個脈衝而無法同步成功，這部分和clk、clk2的操作頻率有關，若兩者相差非常多倍，則無法同步的可能性就越大，所以若轉成level訊號再去做同步，就能避免這樣的問題發生。

(d)load synchronizer架構

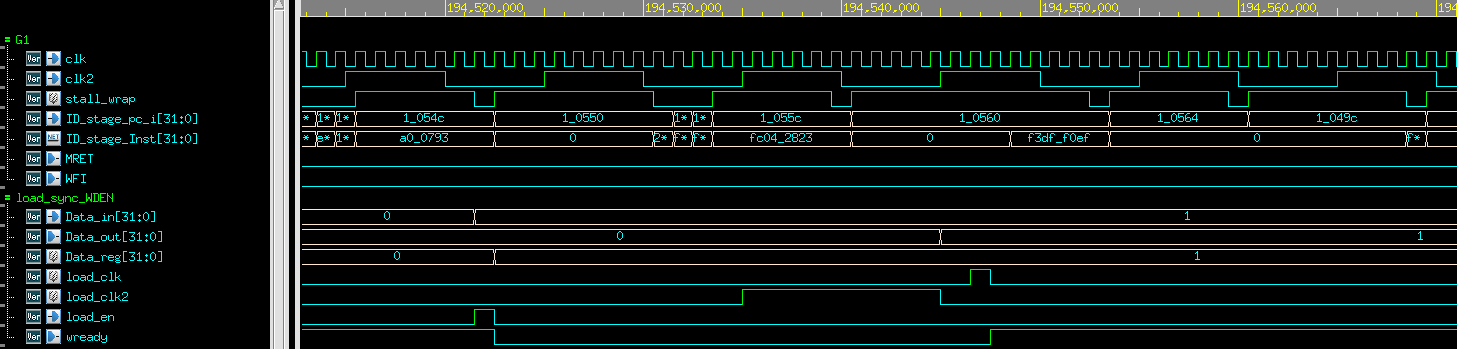


說明:

上圖為load synchronizer的架構，由兩個操作在clk domain的flip flop、兩個pulse synchronizer、和一個操作在clk2 domain的flip flop。這兩個操作在clk domain的flip flop分別要存取Data\_in和load\_en，Data\_in為我們要同步的訊號

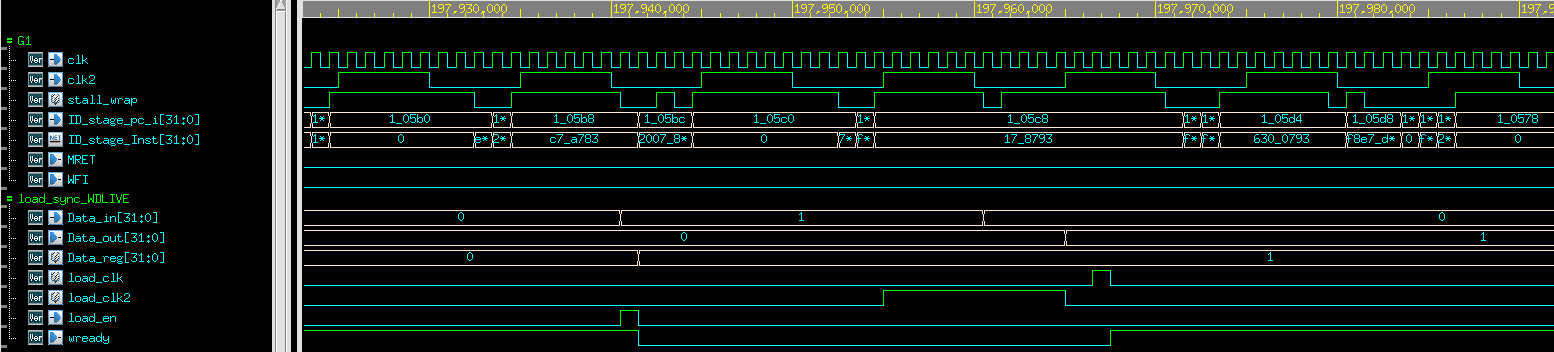
，而load\_en則是為了確保Data\_in(多bit資料)是穩定的，而新增的載入訊號。只有當load\_en為高電位時，Data\_in才能被暫存器存入到Data\_reg。待load\_en經過sync\_pulse後，會產生load\_clk2訊號(pulse in clk domainpulse in clk2 domain)，當load\_clk2訊號為高電位時，暫存器才能將Data\_reg的數值存到Data\_out，而完成同步。當load\_clk2訊號產生，則會再進入一個sync\_pulse，並產生load\_clk訊號，做反饋的目的是要確保數值被同步到 clk2 domain了，這樣才能進行下一筆資料的同步。當load\_en為1，wready會在下個clk變成0，表示不可再同步資料。當load\_clk為1時，wready會在下個clk變成1，表示可再同步資料。

1. 同步訊號的相關波型



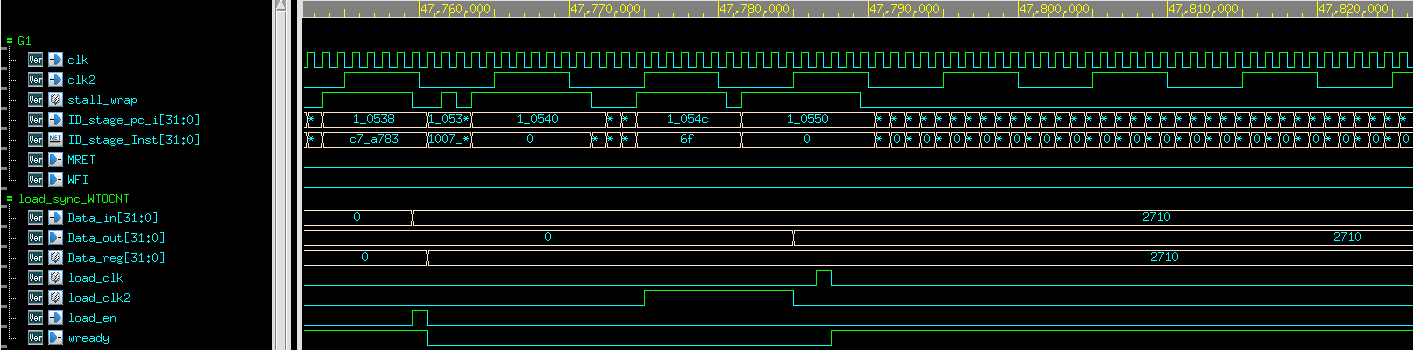
**(i)WDEN同步波型**

說明:上圖為WDEN訊號從clk domain同步到clk2 domain的波型，橘色框框為Data\_in和load\_en訊號，由於load\_en為1，所以會在下個clk將Data\_in的數值存入Data\_reg、並將wready存為0。接著等待load\_en訊號經過pulse\_sync單元得到load\_clk2(藍色框框)，由於load\_clk2為1，所以或在下個clk2將數值同步到Data\_out。load\_clk再經過pulse\_sync單元同步後會得到load\_clk，由於load\_clk為1，所以會在下個clk將wready回復成高電位，表示可以繼續同步資料。

****

**(ii)WDLIVE同步波型**

說明:上圖為WDLIVE訊號從clk domain同步到clk2 domain的波型，橘色框框為Data\_in和load\_en訊號，由於load\_en為1，所以會在下個clk將Data\_in的數值存入Data\_reg、並將wready存為0。接著等待load\_en訊號經過pulse\_sync單元得到load\_clk2(藍色框框)，由於load\_clk2為1，所以或在下個clk2將數值同步到Data\_out。load\_clk再經過pulse\_sync單元同步後會得到load\_clk，由於load\_clk為1，所以會在下個clk將wready回復成高電位，表示可以繼續同步資料。

****

**(iii)WTOCNT同步波型**

說明:上圖為WTOCNT訊號從clk domain同步到clk2 domain的波型，橘色框框為Data\_in和load\_en訊號，由於load\_en為1，所以會在下個clk將Data\_in的數值存入Data\_reg、並將wready存為0。接著等待load\_en訊號經過pulse\_sync單元得到load\_clk2(藍色框框)，由於load\_clk2為1，所以或在下個clk2將數值同步到Data\_out。load\_clk再經過pulse\_sync單元同步後會得到load\_clk，由於load\_clk為1，所以會在下個clk將wready回復成高電位，表示可以繼續同步資料。

**一張含有 文字, 時鐘 的圖片

自動產生的描述**

**(iv)WTO同步波型**

說明:上圖為WTO訊號從clk2 domain同步到clk domain的波型。產生WTO\_clk2訊號後，經過一個clk後會存入WTO\_reg，再經過一個clk後會存入WTO，完成打兩拍的同步方法。

1. **Spyglass**

(a)cdc\_setup\_check

一張含有 文字 的圖片

自動產生的描述

(b)clock\_reset\_integrity

一張含有 文字 的圖片

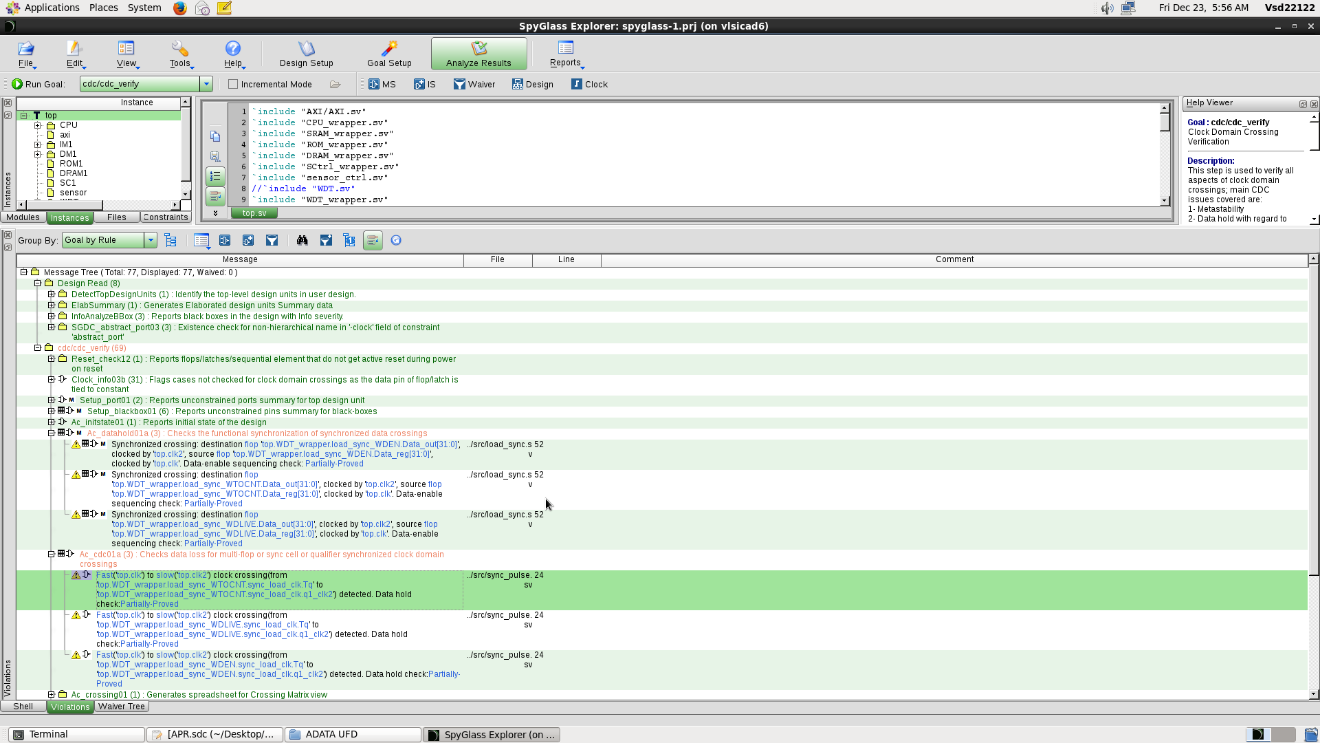
自動產生的描述

(c)cdc\_verify\_struct

一張含有 文字 的圖片

自動產生的描述

(d)cdc\_verify



**Partially prove解釋**:Partially prove代表的是spyglass無法在指定時間內驗出是否能無錯誤發生，所以我在tcl.中有加入**set parameter fa\_atime 100**的指令，但檢驗後的結果依然是partially prove，但實際電路在操作上，WDEN、WDLIVE、WTOCNT都能正確的傳送到clk2 domian(波型圖請見p22.p23)，所以這些warning我認為是可以忽略的。

(e)cdc\_abstract

一張含有 文字 的圖片

自動產生的描述

1. spyglass.sgdc

一張含有 文字 的圖片

自動產生的描述

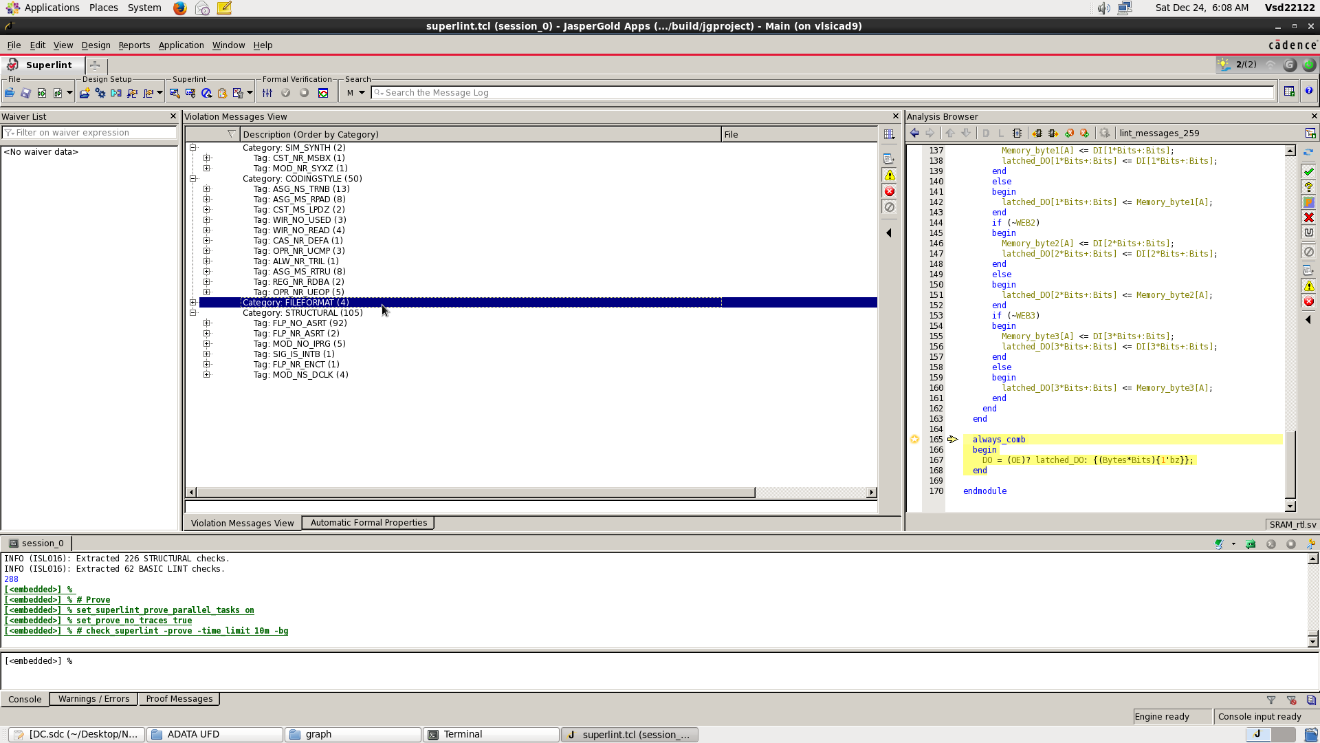
說明:在sgdc檔案的設定上，我有新增一些指令來使進行CDC同步的單元中的暫存器數值歸零，這麼做的原因是為了解決datahold的error，若不初始化這些暫存器，會在0ns的暫存器數值中發現問題。

1. **Simulation result**

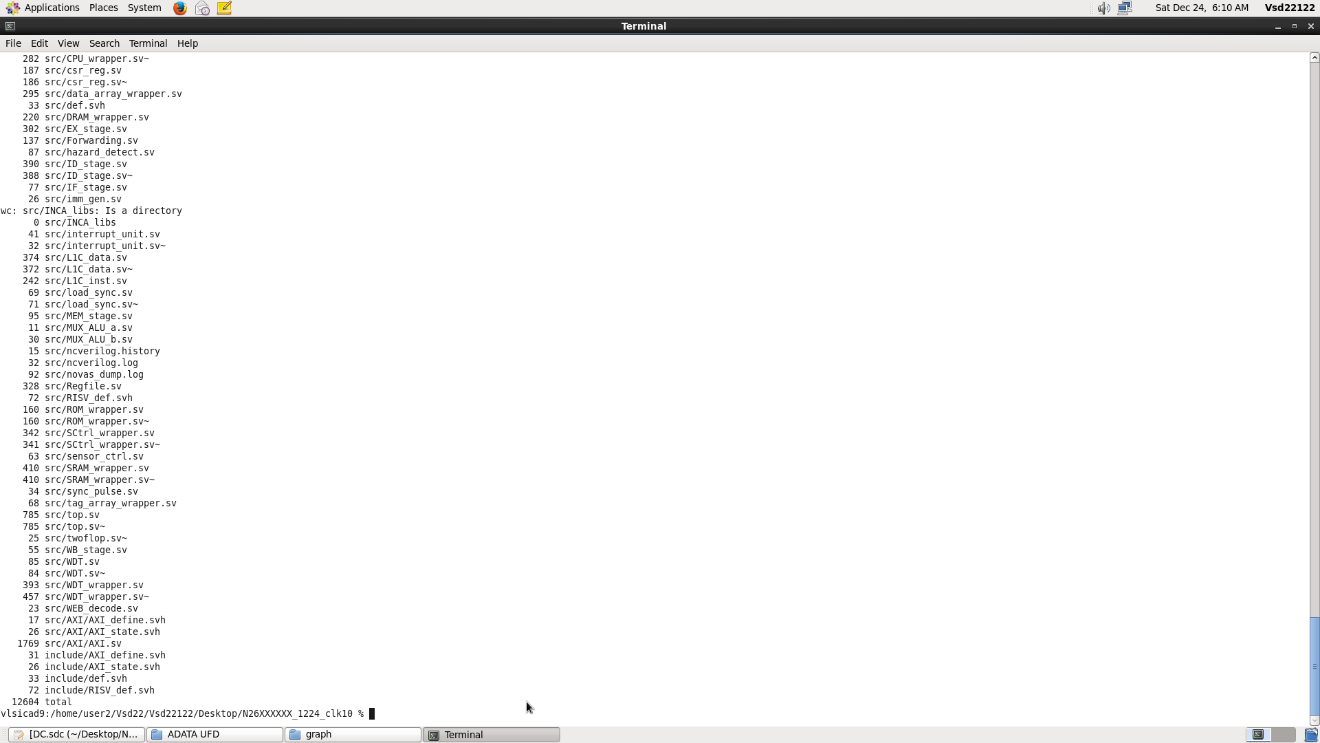
|  |  |
| --- | --- |
|  |  |
| rtl0 | rtl1 |
|  |  |
| rtl2 | rtl3 |
|  |  |
| rtl4 | syn0 |
|  |  |
| syn1 | syn2 |
|  |  |
| syn3 | syn4 |
|  |  |
| pr0 | pr1 |
|  |  |
| pr2 | pr3 |
|  |  |
| pr4 |  |

1. **Superlint result**

(a)Superlint check result

****

說明:共有161個warning，0個error。

(b)code line

說明:程式總行數為12604。

**Most frequency warning or error in supertlint**

(a)flipflop中未使用非同步reset的warning

Sol:由於在spyglass的檢查中，需要將所有的暫存器改成同步reset，所以這部分的Warning沒有要做修改。

(b)case未寫滿的問題

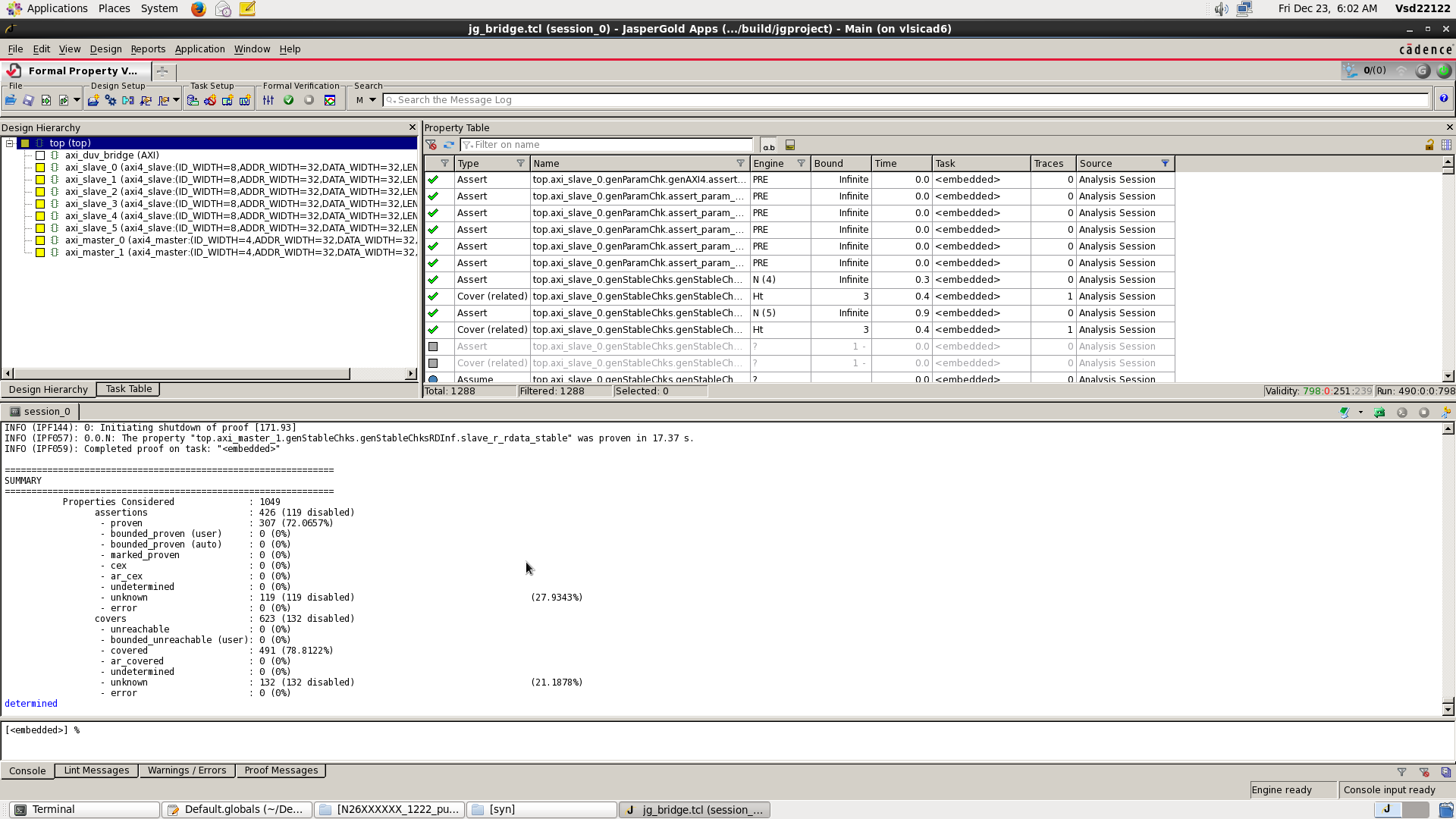
Sol:將case加上default後，即可解決此問題。

(c)等號右邊與等號左邊的暫存器位元數大小不一致

Sol:這是因宣告時不小心而導致的warning，只要將兩邊的位元數宣告正確即可將此warning消除。

經計算後，整體superlint中的執行結果約為**98.72%**(warning不佔總體程式行數的百分率)

1. **AXI ABVIP**

****

AXI 驗證全部通過

**12.Layout**

(a)DRC

一張含有 文字 的圖片

自動產生的描述

(b)Connectivity

一張含有 桌 的圖片

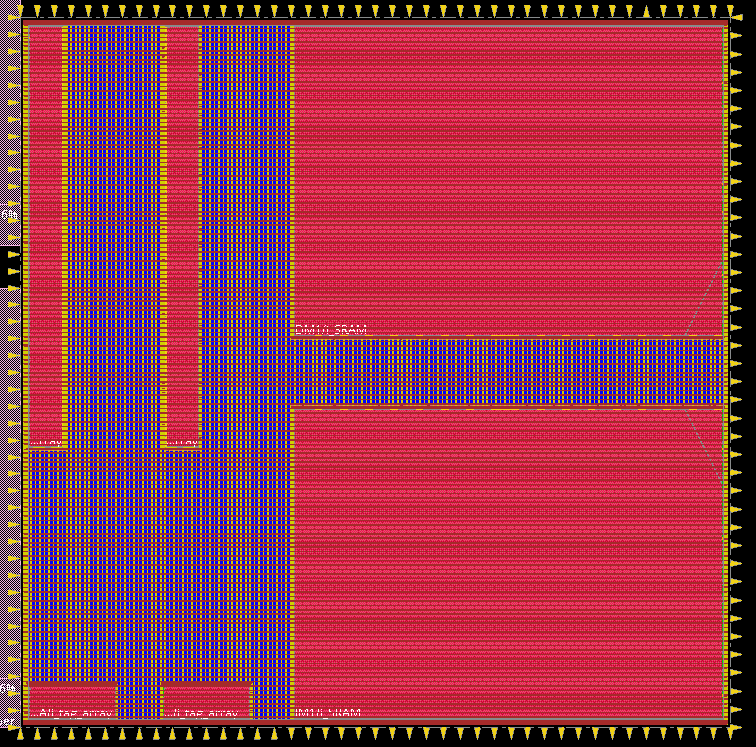
自動產生的描述

(c)Antenna

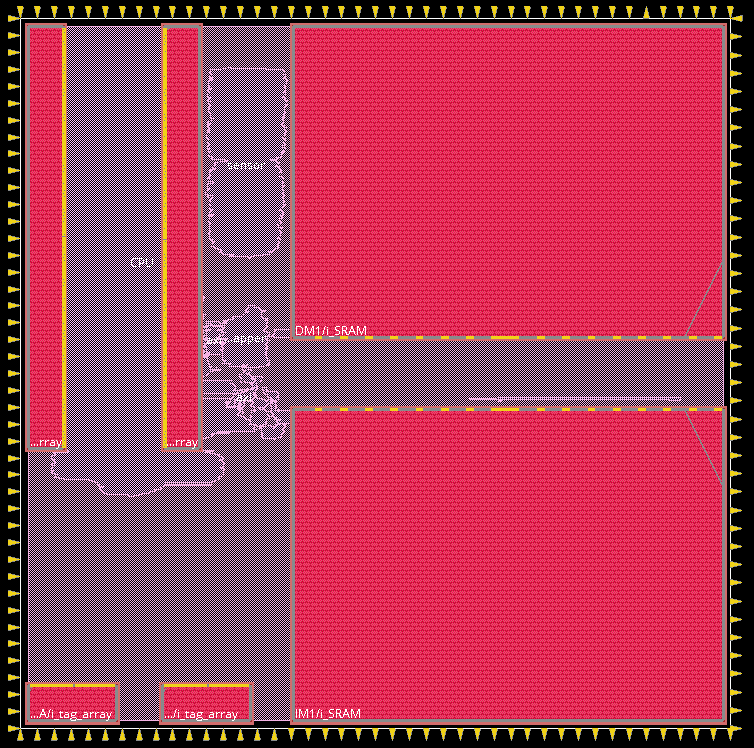
一張含有 文字 的圖片

自動產生的描述

(d)Floorplan



(e)amoeba



(f)physical

一張含有 文字, 電子用品 的圖片

自動產生的描述

**13.Performance**

1. Area (gate-level)

一張含有 文字 的圖片

自動產生的描述

1. Timing(gate-level)

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

1. Area(APR)

一張含有 文字 的圖片

自動產生的描述

1. Power(APR)

一張含有 桌 的圖片

自動產生的描述

**14.Lesson learned**

1. CDC的處理方法

這次是我在數位IC設計的學習過程中，第一次接觸到需要跨時脈溝通的問題。

實際在做系統的時候，各module的操作頻率難免會不相同，如何將資料正確的傳送另個運作時脈且不發生亞穩態的問題是很值得學習的經驗，這次我總共實作了三種跨時脈的處理方法，從最簡單的打兩拍、pulse synchronizer、到非同步FIFO都有用過，但最後選擇了pulse synchronizer作為我的架構。

1. Spyglass的使用

在設計完CDC的處理單元後，還需要使用spyglass來做CDC的檢查，據我了解Spyglass是業界中很用來檢查CDC的EDA tool。雖然我在debug的過程中非常問號，常常報錯的地方在A，但實際發生問題的地方是B，但經過這次的使用經驗，也算是對這個EDA tool有初步的認識。

1. CPU的中斷機制與運作方式

這次會發出中斷的單元有兩個，分別為Sensor ctrl和WDT。透過實際設計中斷訊號，讓我更了解當初在計算機組織中學習到的中斷相關知識。說穿了中斷就是一種jump行為，跳到另一塊程式碼來執行相關操作。

1. APR的操作方式

這算是我第二次接觸到APR，第一次是在大學下線的時候，但當時設計的晶片還不是系統，也沒有記憶體，所以在APR的操作步驟上還是有一些不同的。這次不但有CPU還有一些記憶體，在擺放macro的時候有一些竅門，像是最好要對齊晶片的邊線，這些都是我之前不知道的。心得是很感謝有自動化tool幫忙繞線，能讓數位設計者省很多的心力。