VLSI System Design (Graduate Level)

Fall 2023

HOMEWORK I

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

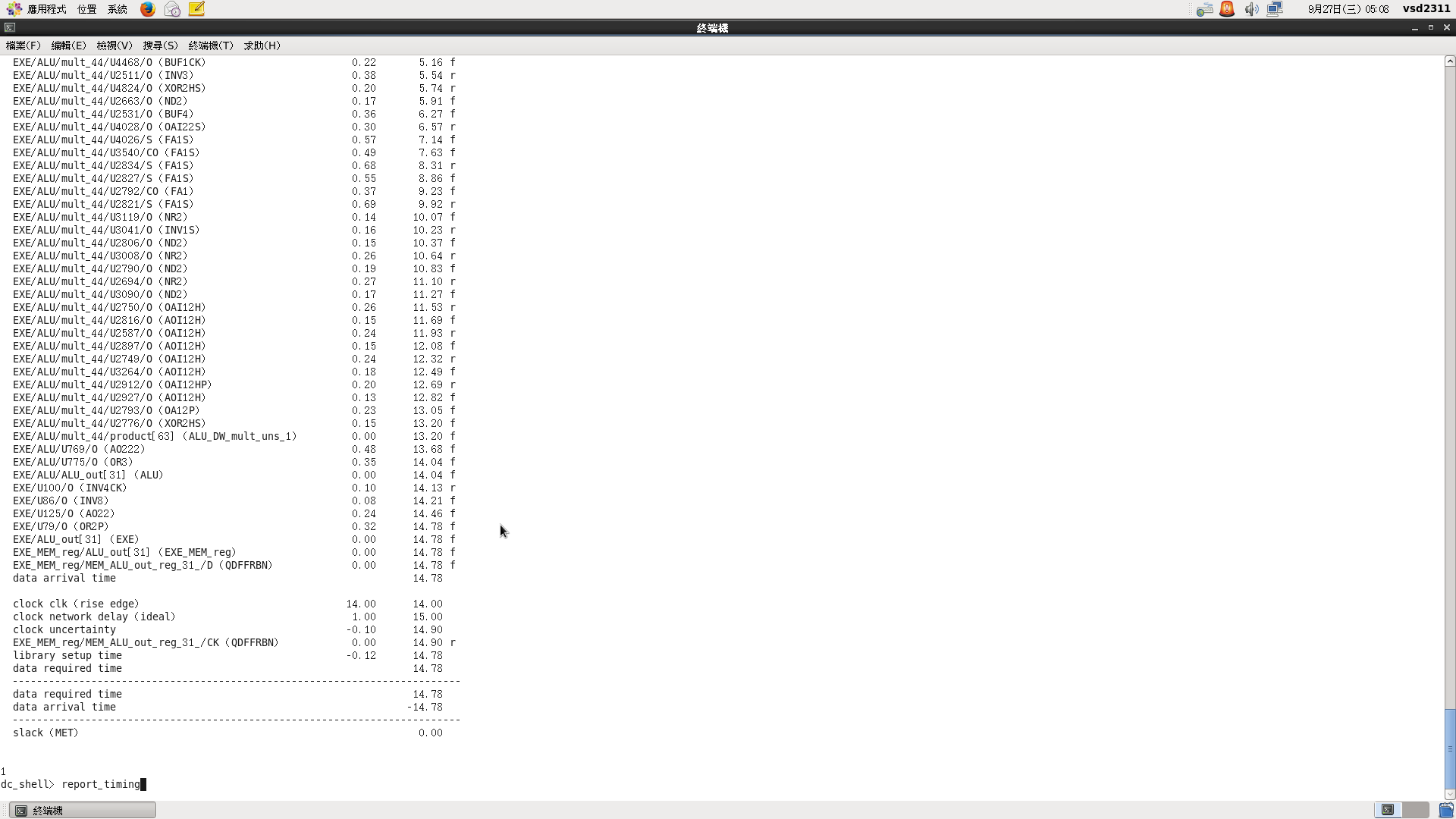
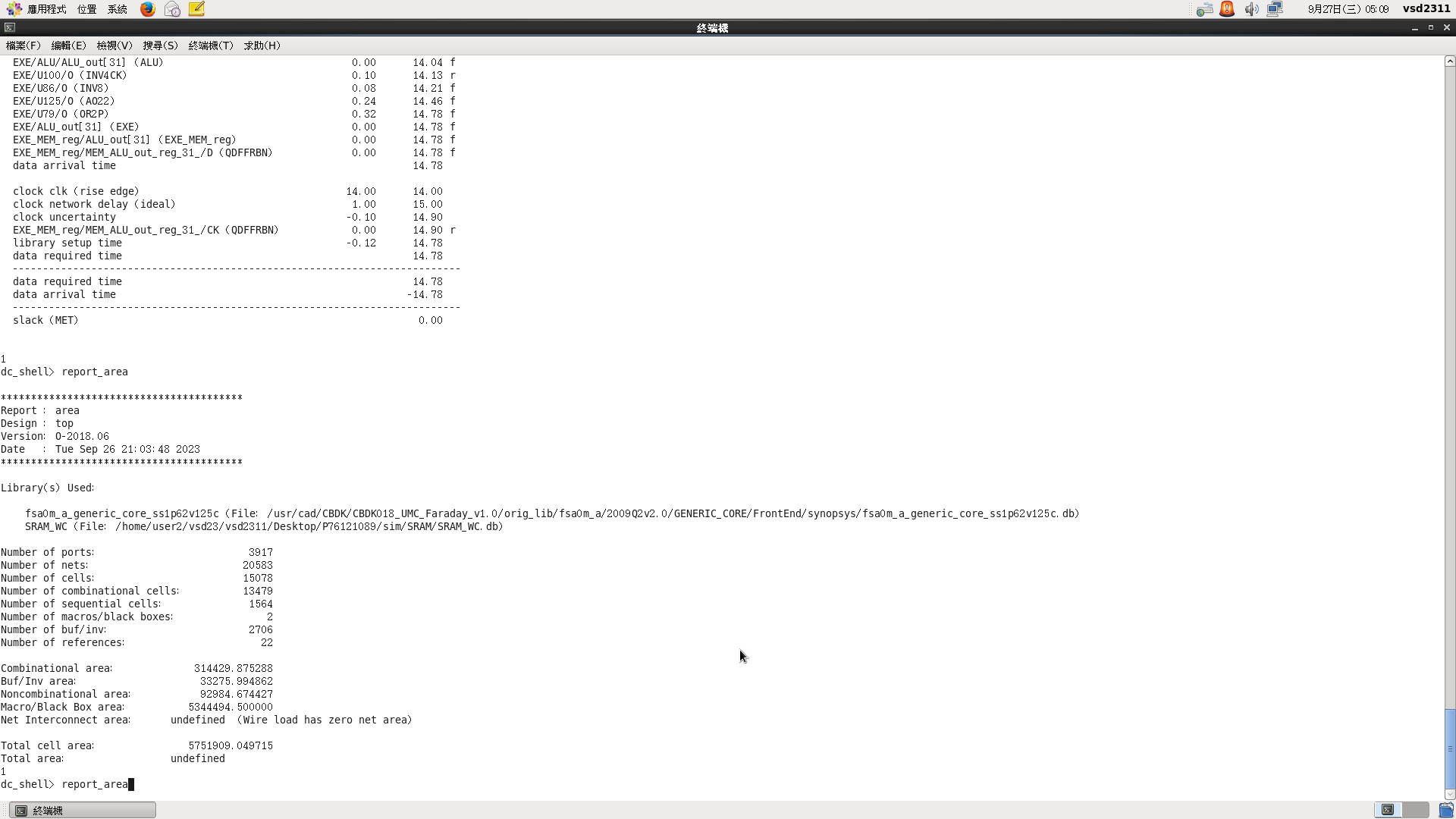
Student name: \_\_張正德\_\_\_\_\_\_\_\_\_

Student ID: \_\_P76121089\_\_\_\_\_\_\_\_

1. Summary

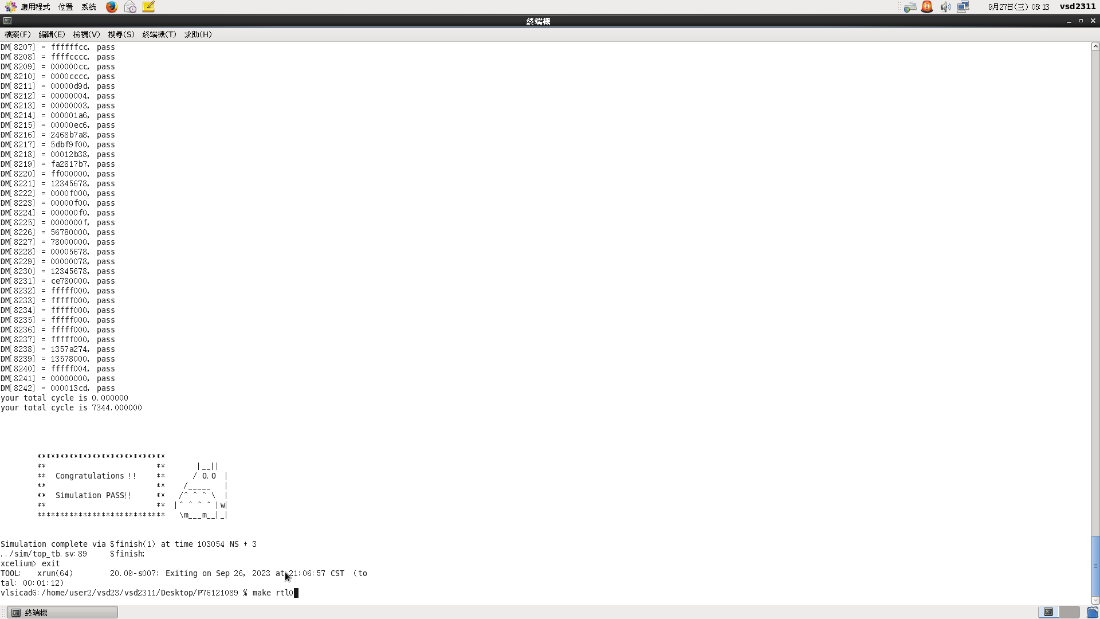
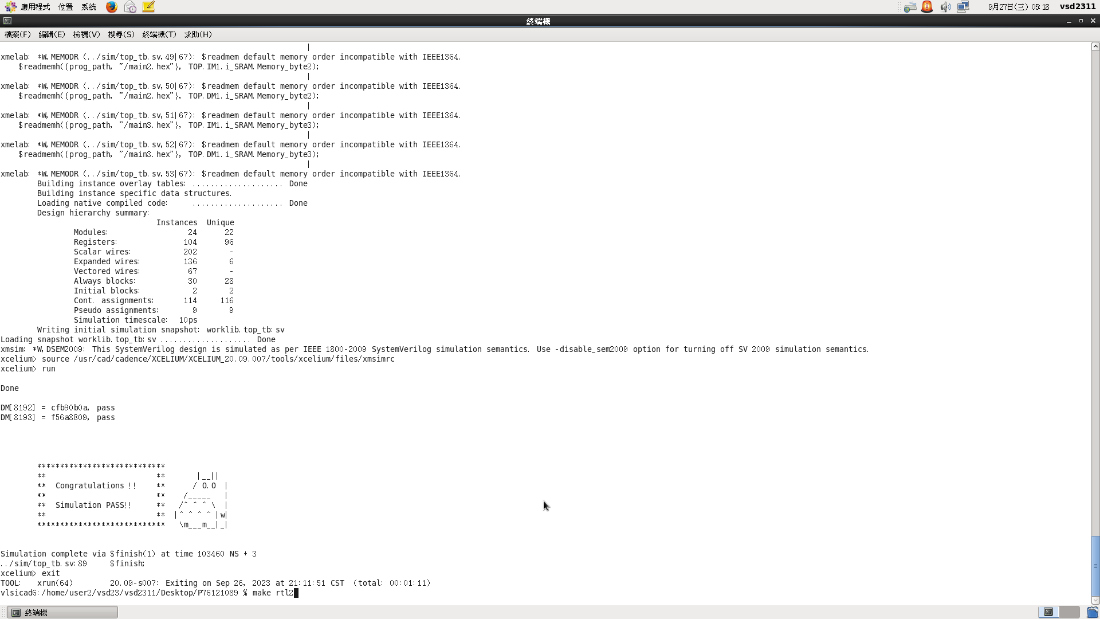
* Hw1 內容為完成一個根據 RISC-V ISA 的 5-stage pipeline CPU。
* 5-stage 分別為：
  + Instruction Fetch
  + Instruction Decode and register read
  + Execution and address generation
  + Memory access
  + Write Back to register
* Harvard architecture
* 實現作業中的45個指令

1. Report\_timing & report\_area
   1. Clock clk : 14, slack(MET) : 0
   2. Total cell area : 5751909.049715

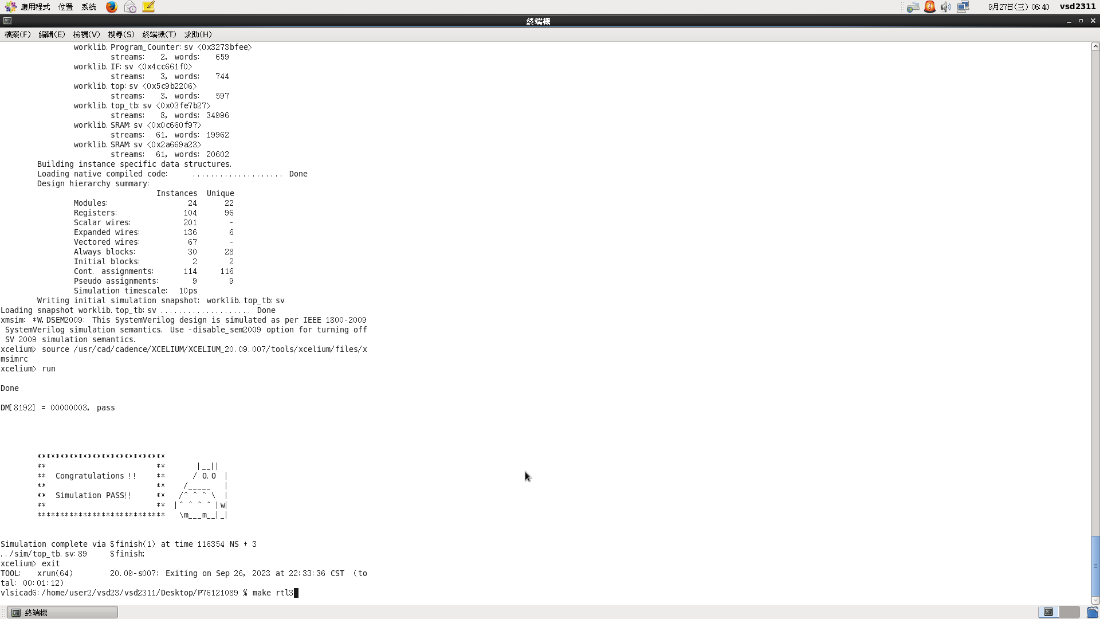
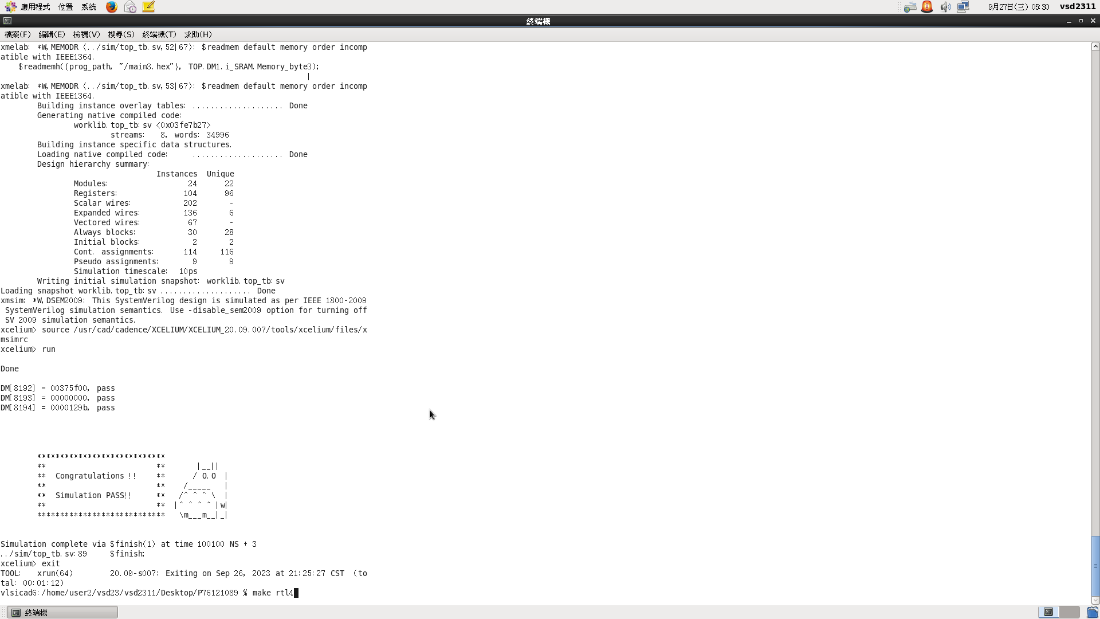
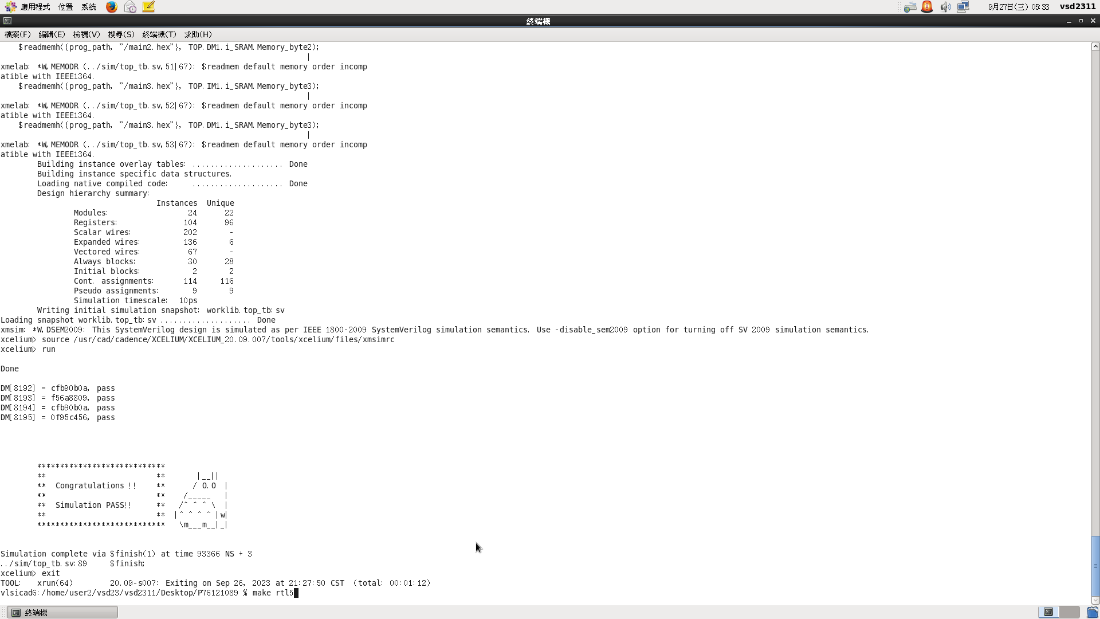
 . 

1. RTL Simulation result (All pass)

Prog0 prog1 prog2

Prog3 prog4 prog5

1. Post-synthesis Simulation (All pass)

Prog0 prog1 prog2

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 prog4 prog5

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

1. Superlint result

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

The number of lines of RTL code : 1622 lines

The number of warnings : 11 lines

1-(11/1622)\*100% = 1-0.6% = 99.4%

1. Block diagram

一張含有 螢幕擷取畫面, 文字, 圖表 的圖片

自動產生的描述

1. Module Description

**Program Counter:**

* 當PC\_Write為1時，pc\_out <= pc\_in；當PC\_Write為0時，pc\_out則不會做更新，保留上一個clk時的值。

**Register File**

* 根據rs1\_addr和rs2\_addr讀出rs1\_data以及rs2\_data。若RegWrite為1，則將rd\_data寫入register[rd\_addr]。位址0x0的暫存器永遠為32’b0，不可被寫入。

**Immediate Generator**

* 根據imm\_Type將instruction中的imm延伸及對應成32bits的immediate值。

**Control Unit**

* 根據opcode產生對應的各個control signals，包括ALUOP, imm\_Type, ALUSrc, Reg\_Write, Mem\_Read, Mem\_Write, MemtoReg, PCtoRegSrc, RdSrc, branch, ALUOutSelect。

**ALU Control**

* 根據Control Unit送出的ALUOP以及funct3、funct7來決定ALU的控制訊號，控制ALU做不同運算。

**ALU**

* 根據ALU\_Control送出的控制訊號對兩個輸入做運算，並輸出運算結果以及Zero\_flag。

**Branch Control**

* 根據Branch 控制訊號以及Zero\_Flag來輸出BranchCtrl決定下一個pc\_in為何。

**Hazard Control**

* Control Hazard: 當Branch跳躍成立時，將Instr\_Flush以及CtrlSignalFlush設為1，沖刷目前在IF-stage以及ID-stage的兩個指令。
* Data Hazard: 利用EXE-stage時的Mem\_Read控制訊號和rd\_addr，以及rs1\_addr和rs2\_addr來判斷是否發生load use Data Hazard，若發生則設定CtrlSignalFlush為1，沖刷位於ID-stage的指令，並且讓PC\_write、IF\_ID pipeline register 皆暫停一個clk(等同加入一個NOP指令)。並利用Forwarding Unit將資料前饋。

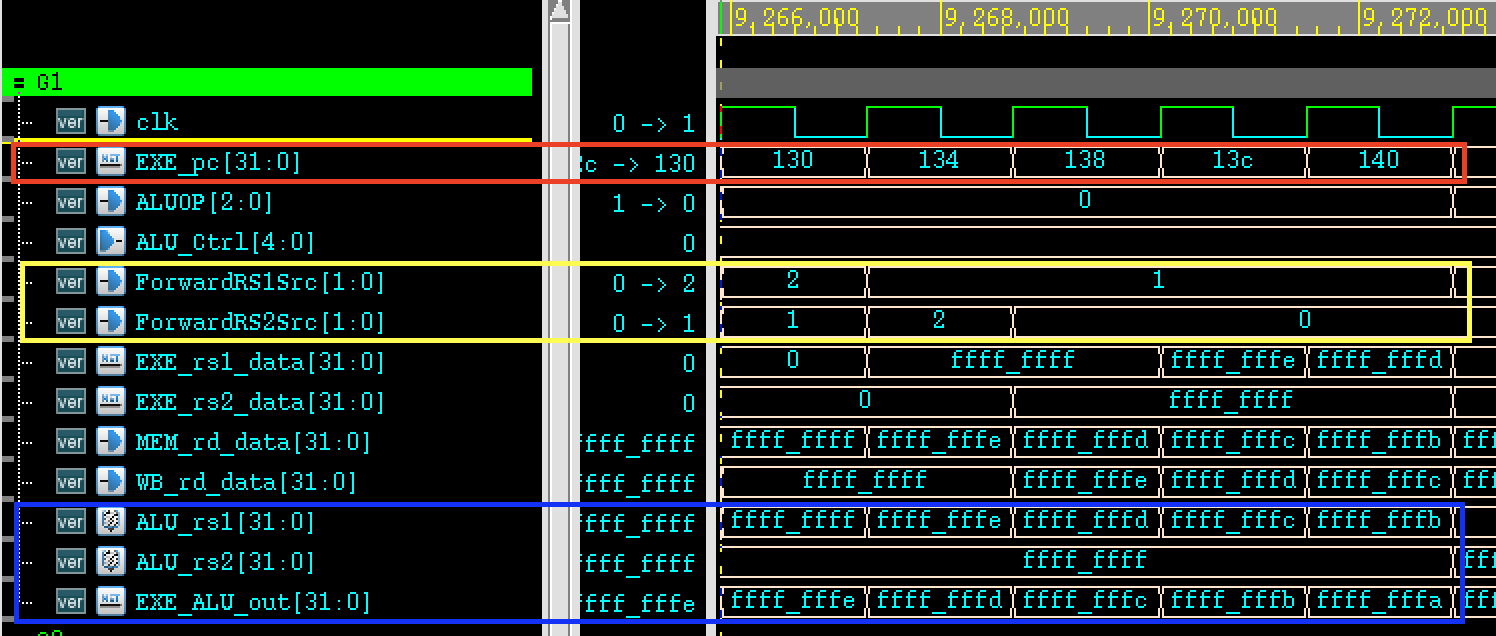
**Forwarding Unit**

* 在EXE-stage時，判斷當前要使用的rs1\_data以及rs2\_data是否在MEM-stage和WB-stage有被更新，若有發生上述Data dependency，則利用ForwardRS1Scr、ForwardRS2Scr控制訊號控制當前要運算的Src，將MEM\_rd\_data或是WB\_rd\_data前饋到EXE-stage。
* 判斷方法: 利用MEM-stage以及WB-stage的RegWrite、rd\_data來和當前EXE-stage的rs1\_addr以及rs2\_addr做比對。

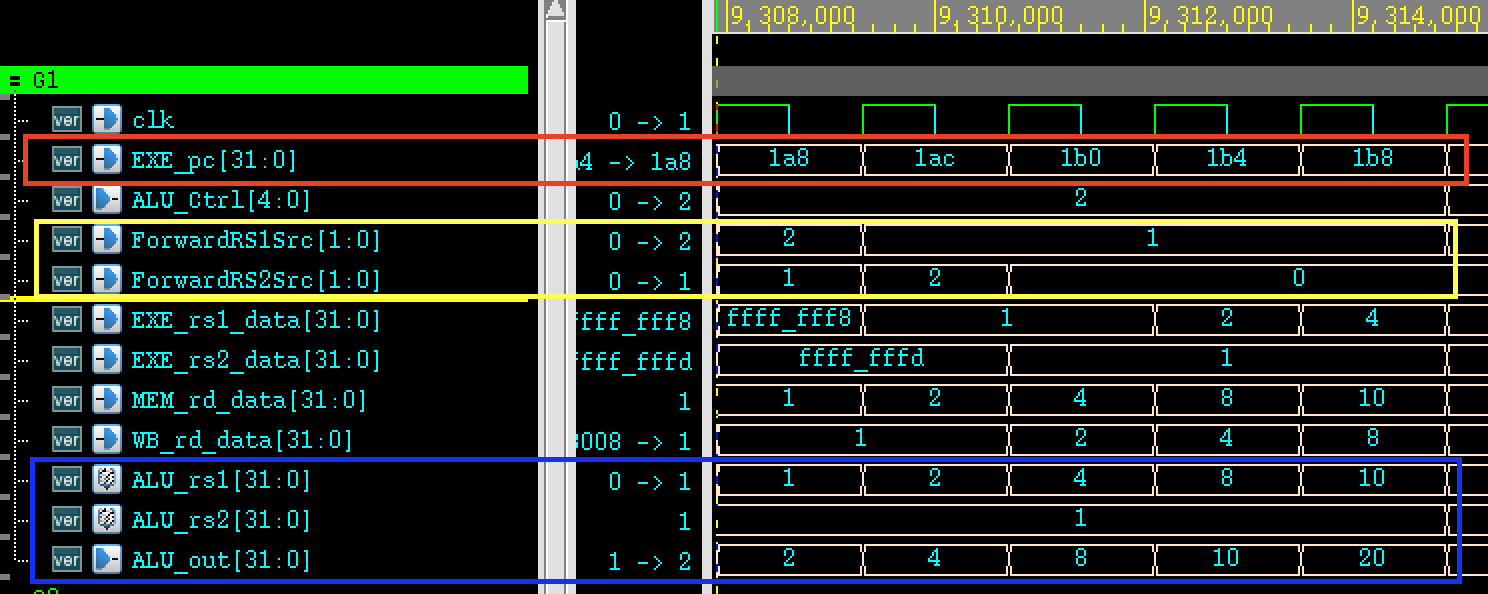
**CSR Calculator**

* 存在兩個暫存器分別為instret以及cycle，用來計算和儲存當前指令數目以及clock cycle數。並且在Branch跳躍成立時將instret暫存器數量減1(Flush 2 instruction)，當load use hazard發生時將instret保持不變，其餘皆遞增1。Cycle暫存器則每當posedge clk時遞增1。
* 另外，此module位在EXE-stage，因此在cycle數大於1之後才開始計算instruction數量。

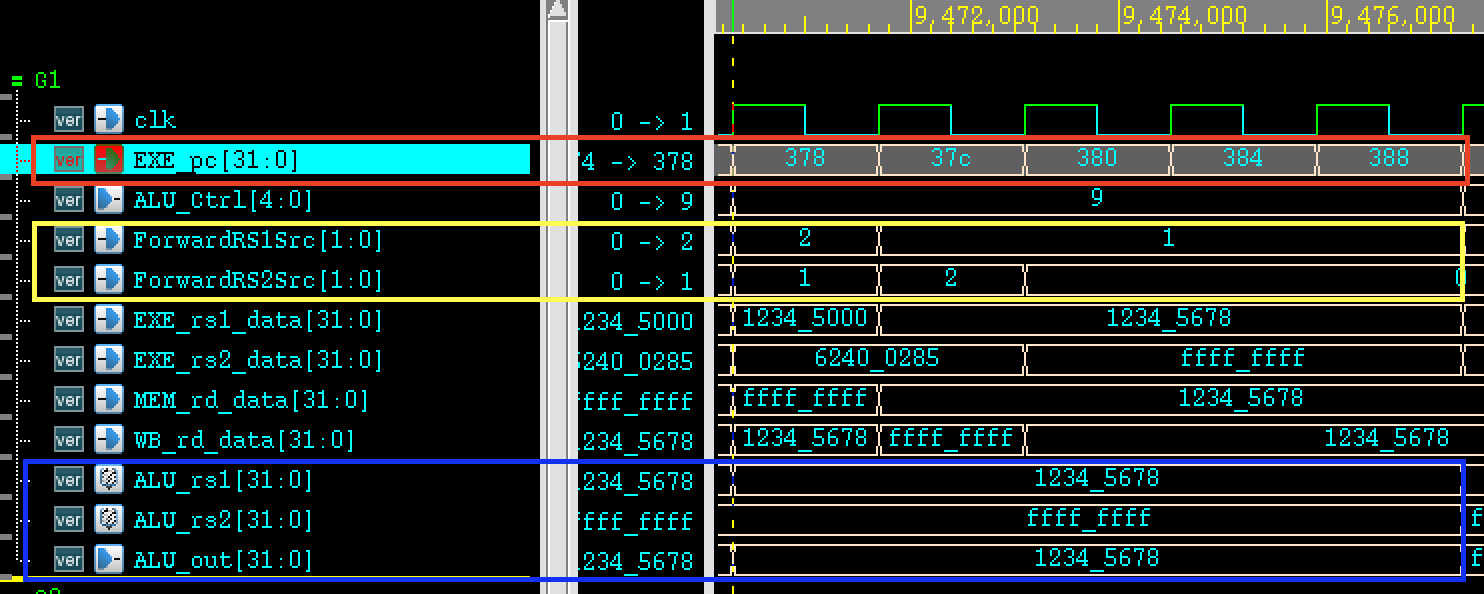
1. 波型驗證
   1. R-Type
      1. 驗證 **ADD instruction**



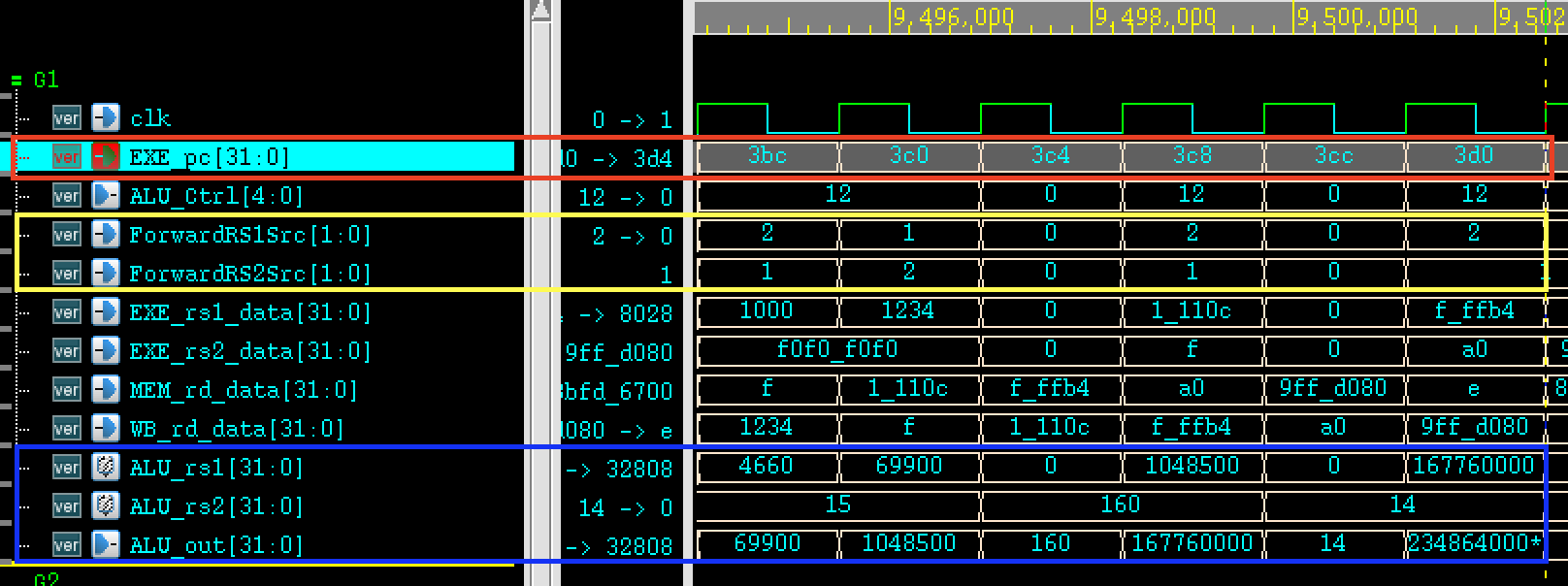
* 紅色框起來的為當前進到EXE stage時的pc，根據main.log可得pc在130~140位址時的指令皆為 add t0, t0, t1。
* ALU\_Ctrl 值為0，控制 ALU 做加法運算。
* 此時因為連續五個指令中都使用t0、t1暫存器，因此必定需要利用Forwarding解決。黃框處觀察ForwardRS1Src以及ForwardRS2Src，當值等於0時ALU resource為從ID stage傳來的EXE\_rs1/2\_data；當值等於1時ALU resource 為MEM\_rd\_data；等於2時則為 WB\_rd\_data。
* 最後驗證ALU\_rs1 + ALU\_rs2 等於 EXE\_ALU\_out (example : ffff\_ffff + ffff\_ffff = ffff\_fffe)
  + 1. 驗證 **SLL instruction**



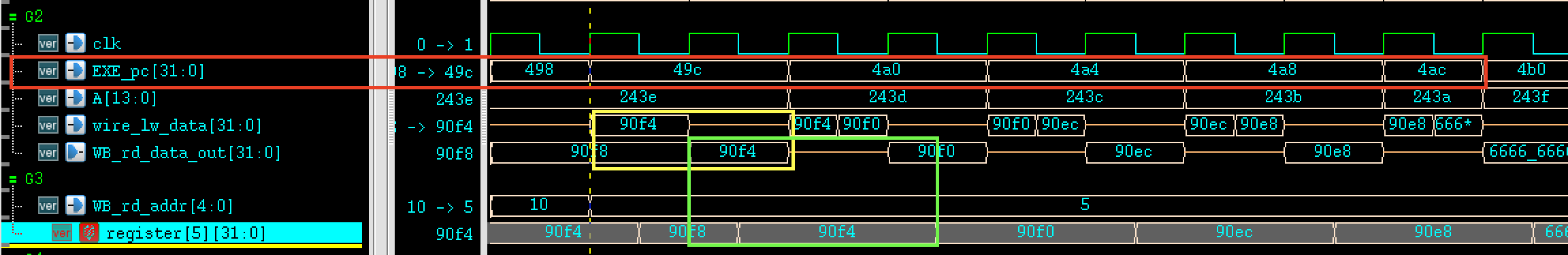
* 紅色框起來的為當前進到EXE stage時的pc，根據main.log可得pc在1a8~1b8位址時的指令皆為 sll t0, t0, t1。
* ALU\_Ctrl 值為 2，控制 ALU 做左移運算。
* 與上述ADD指令相同，因為連續五個指令中都使用t0、t1暫存器，因此必定需要利用Forwarding解決。黃框處ForwardRS1Src以及ForwardRS2Src當值等於0時ALU resource為從ID stage傳來的EXE\_rs1/2\_data；當值等於1時ALU resource 為MEM\_rd\_data；等於2時則為 WB\_rd\_data。
* 最後驗證ALU\_rs1 << ALU\_rs2 等於 EXE\_ALU\_out (example：以下數值表示法為十六進制: 8 << 1= 10)
  + 1. 驗證 **AND instruction**



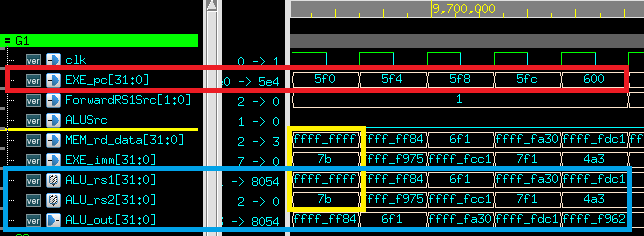
* 紅色框起來的為當前進到EXE stage時的pc，根據main.log可得pc在378~388位址時的指令皆為 and t0, t0, t1。
* ALU\_Ctrl 值為 9，控制 ALU 做 AND 運算。
* 與上述其他指令相同，因為連續五個指令中都使用t0、t1暫存器，因此必定需要利用Forwarding解決。黃框處ForwardRS1Src以及ForwardRS2Src當值等於0時ALU resource為從ID stage傳來的EXE\_rs1/2\_data；當值等於1時ALU resource 為MEM\_rd\_data；等於2時則為 WB\_rd\_data。
* 最後驗證ALU\_rs1 & ALU\_rs2 等於 EXE\_ALU\_out (example：以下數值表示法為十六進制: 1234\_5678 & ffff\_ffff = 1234\_5678)
  + 1. 驗證 **MUL instruction**



* 紅色框起來的為當前進到EXE stage時的pc，根據main.log可得pc在3bc、3c0、3c8、3d0的位址時，指令皆為 mul t0, t0, t1。
* ALU\_Ctrl 值為 12，控制 ALU 做 MUL 運算。
* 因前兩個指令為連續的mul指令且皆使用t0、t1暫存器，因此需要利用Forwarding解決。黃框處ForwardRS1Src以及ForwardRS2Src當值等於0時ALU resource為從ID stage傳來的EXE\_rs1/2\_data；當值等於1時ALU resource 為MEM\_rd\_data；等於2時則為 WB\_rd\_data。
* 最後驗證ALU\_rs1 \* ALU\_rs2 等於 EXE\_ALU\_out (example：圖中藍色框處數值表示法為十進制: 4660 \* 15 = 69900)
  1. I-Type
     1. 驗證 **LW instruction**



* 紅框處 EXE\_pc 在位址 498～4a8 的指令皆為 lw t0, 0(t0)。在下一個clk進到 MEM-stage 時開始讀出 Data memory 的資料。上述五個指令中，第二個到第五個都有發生load use data hazard，可以看到除了第一個lw指令以外皆需2 clk才跳到下一個stage。
* 可以看到在 EXE\_pc 為 498 的下一個 clk 黃色方框處 lw\_data 讀出 904f ，再下一個 clk 傳給 WB-stage，再依照 WB\_rd\_addr(5) 的位址寫入 register[5] 。
  + 1. 驗證 **XORI instruction**



* 紅框處 EXE\_pc 在位址 5f0～600 的指令皆為 xori t0, t0 ,imm。
* 因5個指令為連續的XORI指令且皆使用t0暫存器，因此需要利用Forwarding解決。黃框處ForwardRS1Src當值等於1，因此將 MEM\_rd\_data 的值 assign 給 ALU\_rs1。
* ALUSrc為0，將EXE\_imm的值 assign 給 ALU\_rs2。
* 最後驗證ALU\_rs1 ^ ALU\_rs2 (imm) 等於 EXE\_ALU\_out (example：圖中藍色框處數值表示法為十六進制: ffff\_ffff ^ 7b = ffff\_ff84)
  + 1. 驗證 **JALR instruction**

一張含有 螢幕擷取畫面, 文字, 電子產品 的圖片

自動產生的描述

* 紅框處 EXE\_pc 在位址 7dc 的指令為jalr t1,t1。
* 黃框處驗證 PC=imm+rs1
* 藍框處可以看到EXE\_pc\_to\_reg 為pc+4 (7dc+4=7e0)，並且接續傳到MEM-stage、WB-stage，最終存入register[6](rd)。
  1. S\_type
     1. 驗證**SW instruction**

一張含有 文字, 螢幕擷取畫面, 多媒體軟體, 軟體 的圖片

自動產生的描述

* 紅框處EXE\_pc 在位址 874~884 的指令分別為sw t0, -4(s0), sw t0, -8(s0), sw t0, -12(s0), sw t0, -16(s0), sw t0, -20(s0)。
* 黃框處可以看到ALU\_out = EXE\_imm + rs1，並且在下一個 clk 傳給 MEM 作為 DM 的 Address (藍框處的MEM\_ALU\_out)。
* 最後觀察藍框處在 MEM-stage 將MEM\_rs2\_data assign給DI後寫入MEM\_ALU\_out此address。
  1. B\_type
     1. 驗證**BEQ instruction**

一張含有 文字, 螢幕擷取畫面, 多媒體軟體, 軟體 的圖片

自動產生的描述

* 紅框處EXE\_pc 在位址 90c 的指令為 beq  t0,t1, imm。
* 藍框處可以看到rs1 == rs2，因此下一個指令應為pc+imm。
* 因branch是在EXE-stage計算決定，可以看到在後兩個clk時的pc為914(pc+imm)。
  1. U\_type
     1. AULPC instruction

一張含有 螢幕擷取畫面, 鮮豔, 多媒體軟體 的圖片

自動產生的描述

* 紅框處EXE\_pc 在位址 b78~b80 的指令分別為 auipc  t0,0x0 、auipc  t1,0xfffff、auipc  t2,0x13579。
* 黃框處驗證pc\_to\_reg = pc + imm。
* 藍框處可以看到將pc\_to\_reg的值接續傳到MEM-stage以及WB-stage。
* 最後存入register[rd\_addr]中(粉紅色框處)。
  1. J\_type
     1. **JAL instruction**

**一張含有 螢幕擷取畫面, 多媒體軟體, 鮮豔 的圖片

自動產生的描述**

* 紅框處EXE\_pc 在位址 bb4 的指令為jal t1, bbc。
* 黃框處第一個clk可驗證EXE\_pc\_to\_reg = pc+4。
* 黃框處可以看到將pc\_to\_reg的值接續傳到MEM-stage以及WB-stage。
* 最後存入register[rd\_addr]中(藍框處)。
* 並且可在粉紅色框看到在後兩個clk將pc跳到pc+imm的位址(bbc=bb4+8)。
  1. CSR type
     1. **RDINSTRETH & RDINSTRET**

**一張含有 螢幕擷取畫面, 鮮豔, 電子產品, 多媒體軟體 的圖片

自動產生的描述**

* 紅框處EXE\_pc 在位址 bd4 的指令為rdinstreth t0，bd8的指令為rdinstret  t1。
* 黃框處可以看到ALUOutSelect為high，控制ALU\_out選擇的輸出為wire\_instret，並且依照指令不同的imm選擇前32bits或是後32bits。
* 藍框處可以看到將ALU\_out的值接續傳到MEM-stage以及WB-stage。
* 最後存入register[rd\_addr]中(粉紅色框處)。
  + 1. **RDCYCLE & RDCYCLEH**

**一張含有 螢幕擷取畫面, 多媒體軟體, 鮮豔 的圖片

自動產生的描述**

* 紅框處EXE\_pc 在位址 bf4 的指令為rdcycleh t0，bf8的指令為rdcycle  t1。
* 黃框處可以看到ALUOutSelect為high，控制ALU\_out選擇的輸出為wire\_cycle，並且依照指令不同的imm選擇前32bits或是後32bits。
* 藍框處可以看到將ALU\_out的值接續傳到MEM-stage以及WB-stage。
* 最後存入register[rd\_addr]中(粉紅色框處)。

1. Lessons learned

本次作業為實作RISC-V ISA的5-stage pipeline CPU的作業，這項專案對於我個人和專業成長都有著深遠的影響。

透過這次作業學到許多 :

1. 硬體細節的理解: 這個作業要求我深入瞭解計算機硬體的細節，包括各個Data Path、Control Unit和Hazard的處理等。我學會了如何設計這些元件以實現CPU的基本功能。
2. 時序和時脈管理: 實作一個5-stage pipeline CPU需要精確的時序控制，以確保每個階段在正確的時機執行。我學到了如何設計和控制時脈信號，以確保整個系統的穩定性。
3. HDL編程技能: 雖然以往也有接觸Verilog語言，但是並未實作過如此完整且較大型的專案，在合成的結果探討和使用Superlint工具之下，也學到了更多、更細節且專業的設計技巧。
4. 效能優化: 在設計和實作過程中，起初的timing是非常不理想的，結果未能在限制內，最終也跟學長和實驗室同學一起探討，想出不同解決方案，了解更多設計方法來優化架構，讓clock cycle可以下降許多。

這個作業提供了一個獨特的學習機會，讓我深入瞭解了計算機硬體設計的方方面面，同時也培養了許多實用的技能。