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: \_\_\_\_\_林晉宇\_\_\_\_\_\_

Student ID: \_\_\_\_P76121673\_\_\_\_\_\_

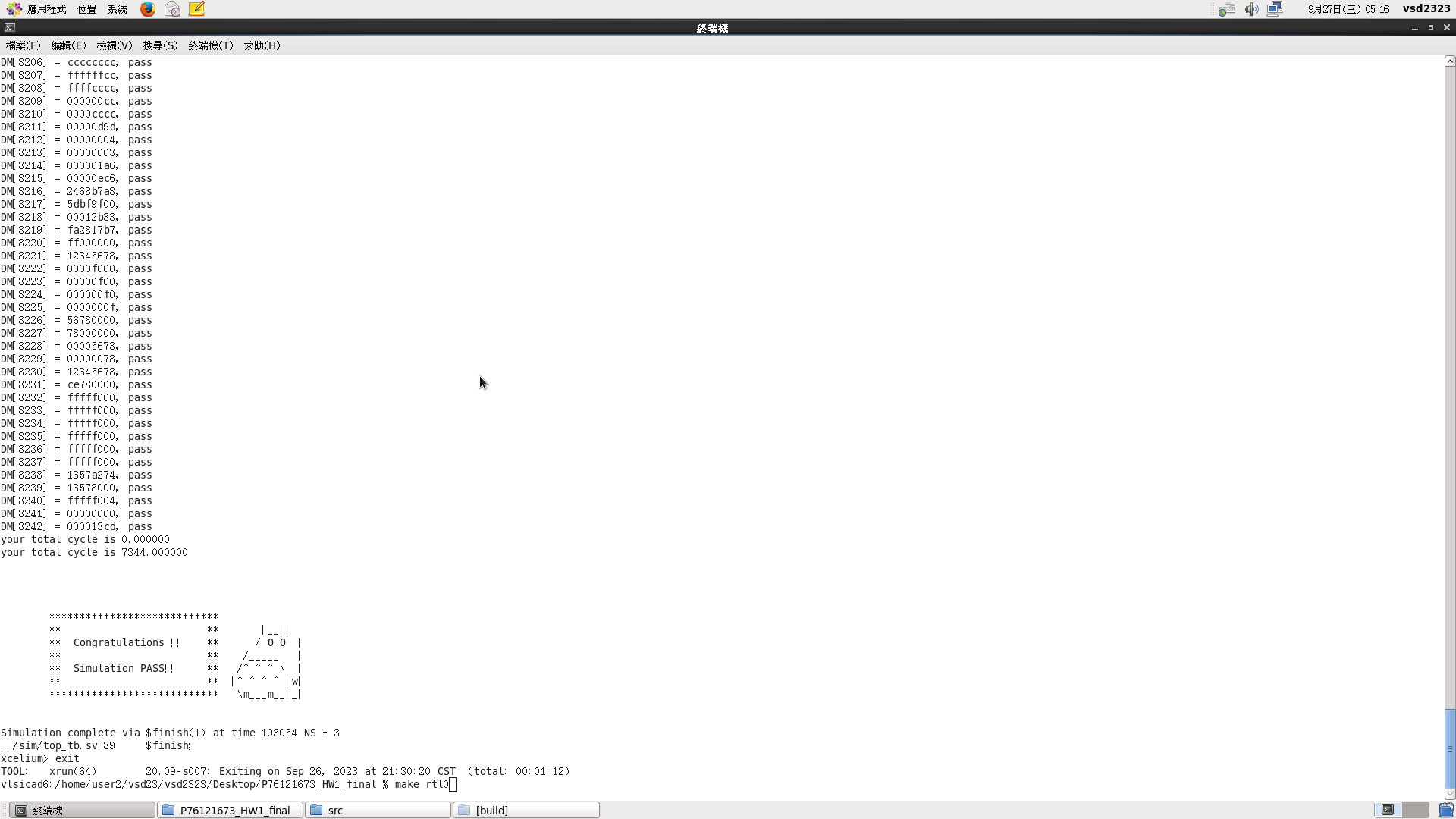
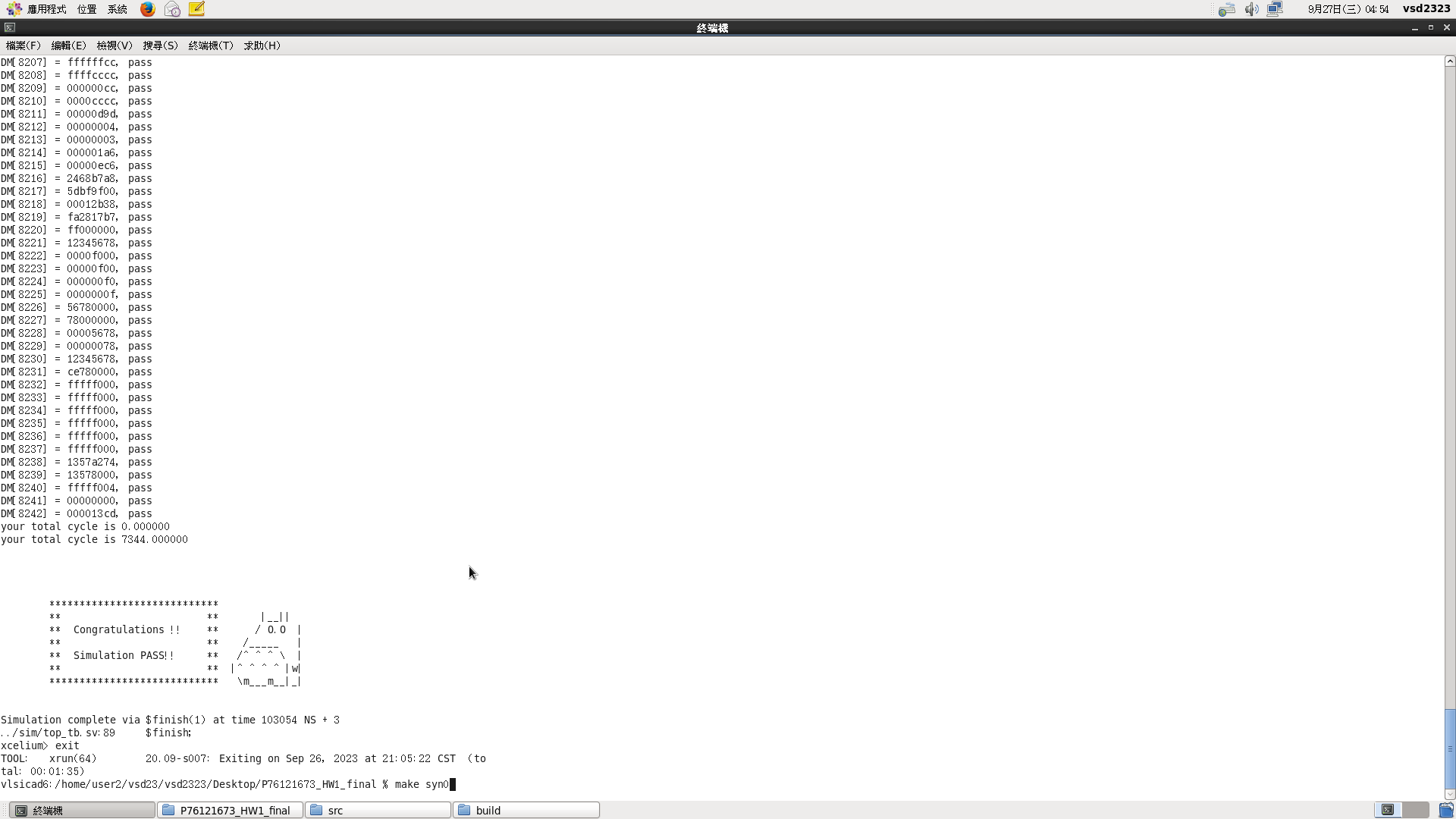
1. Summary

* RISC-V 5-stage pipeline CPU
* 主要將作業一分成五個module (IF、ID、EXE、MEM、WB)以及負責控制訊號的module(Branch Control、Forward Unit、Hazard Control)
* 有些module則會包含負責處理資料的module
* IF module中包含PC
* ID module中包含Control Unit、Register File、Immediate Generator
* EXE module中包含ALU

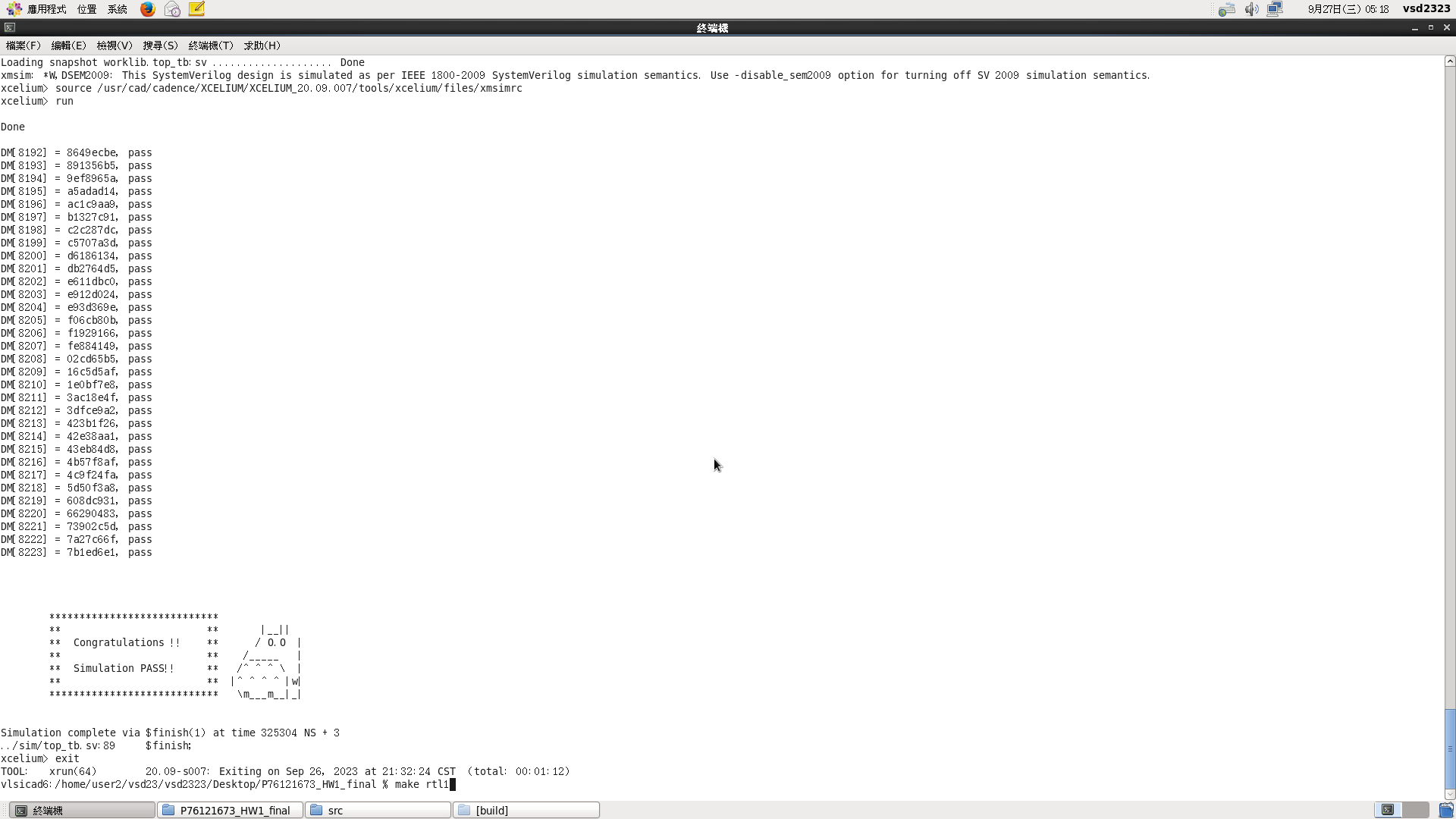
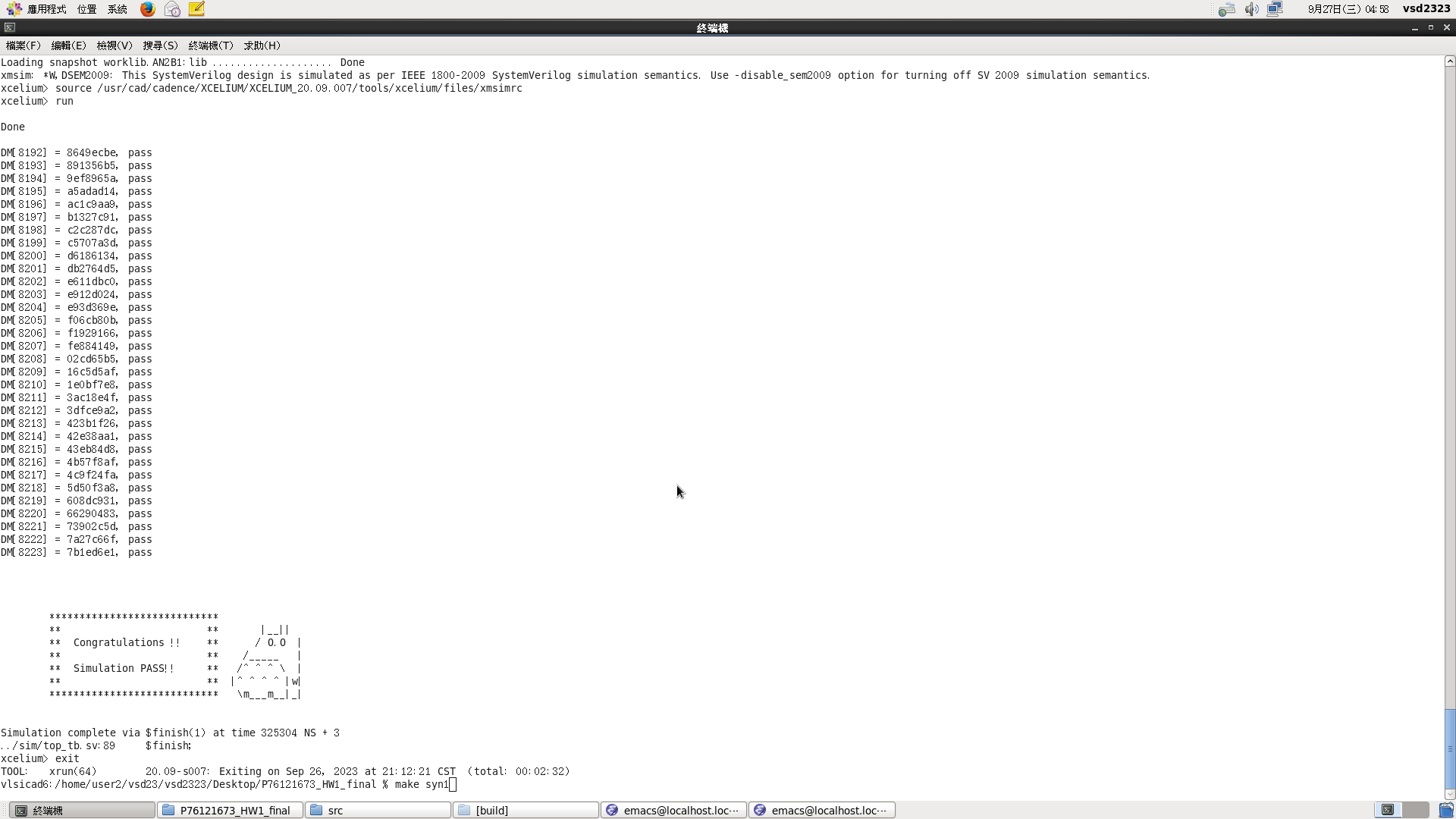
1. RTL & Gate-level simulation

RTL : Gate-level:

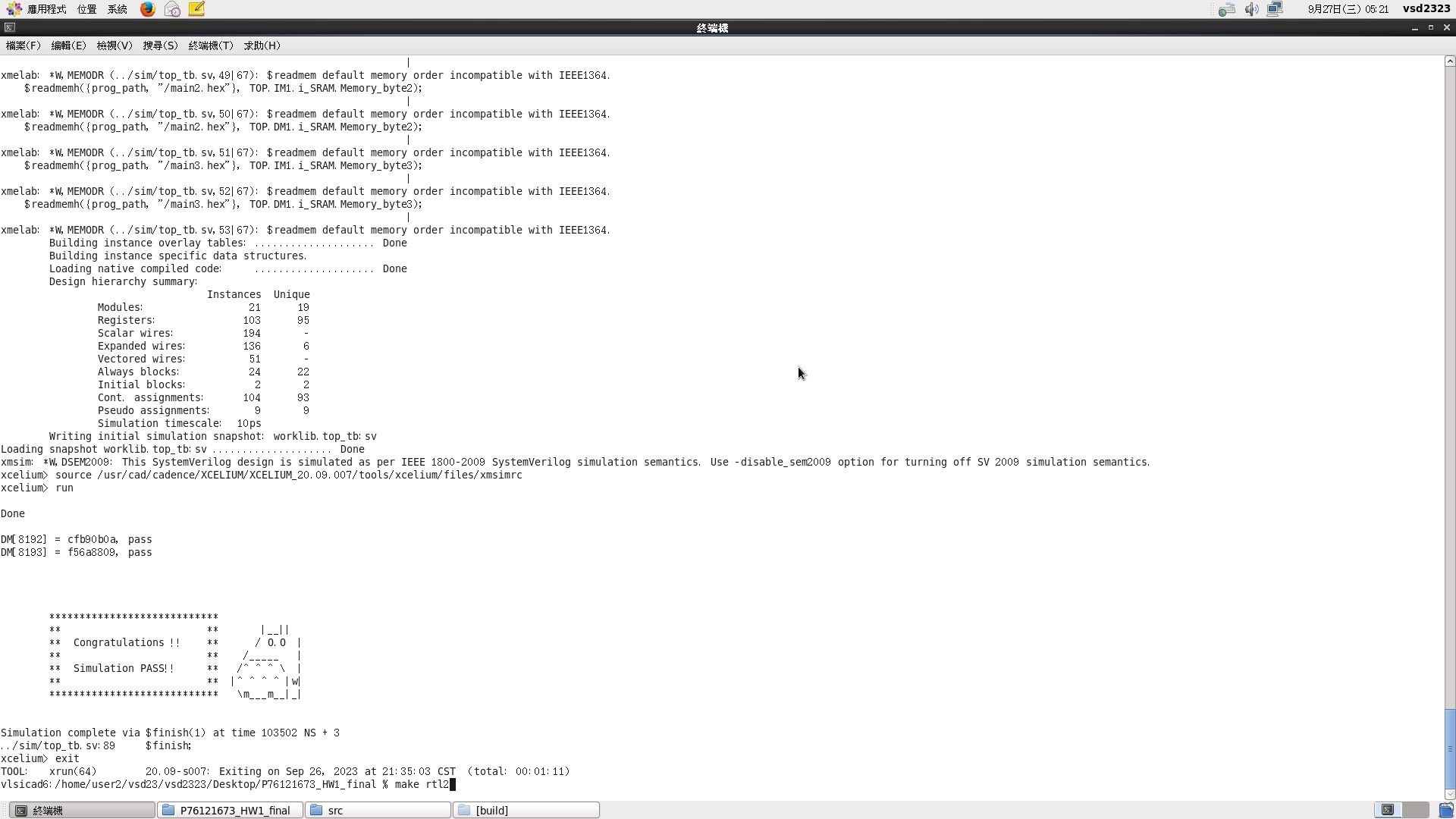
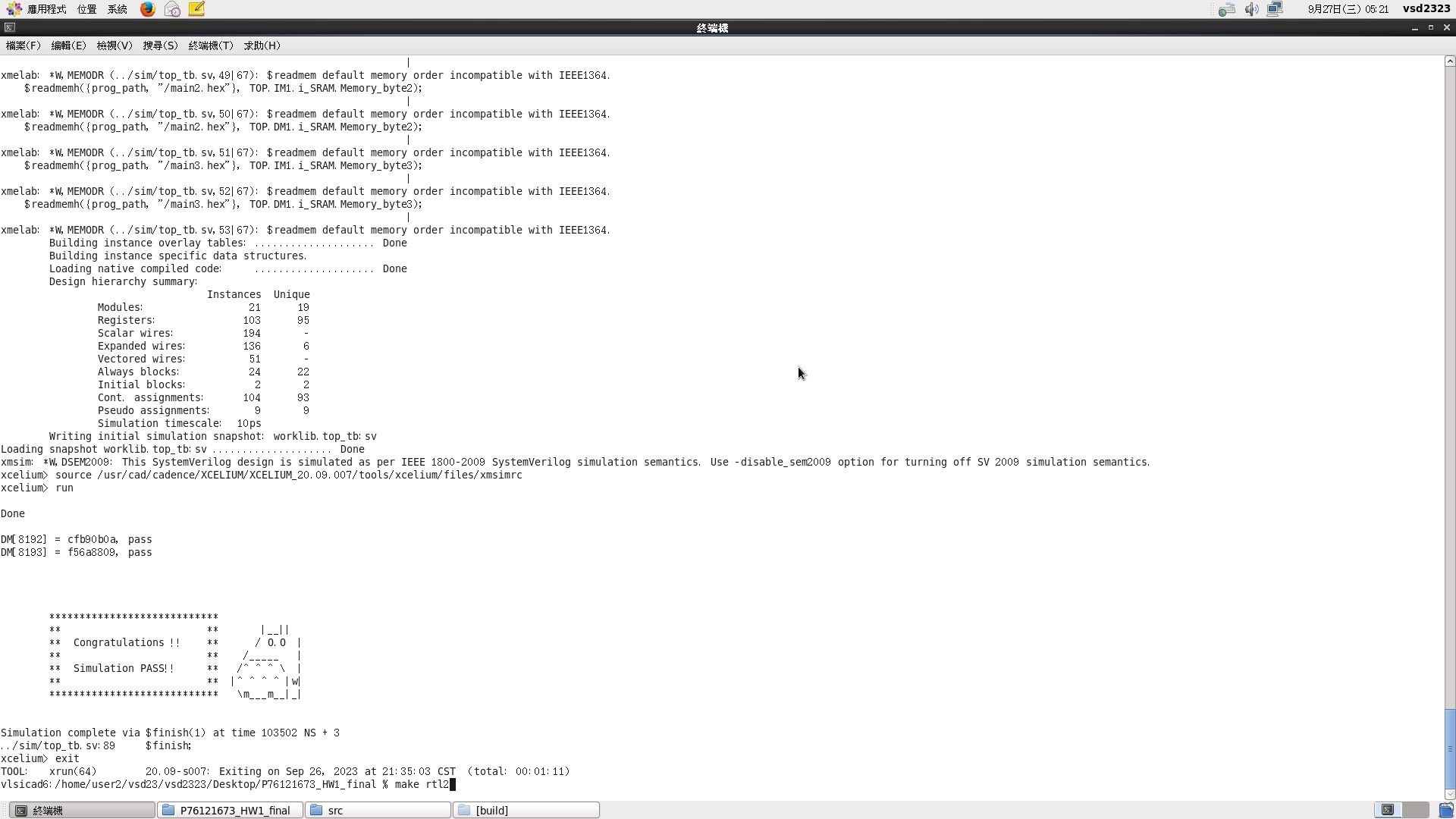
Progrm0:

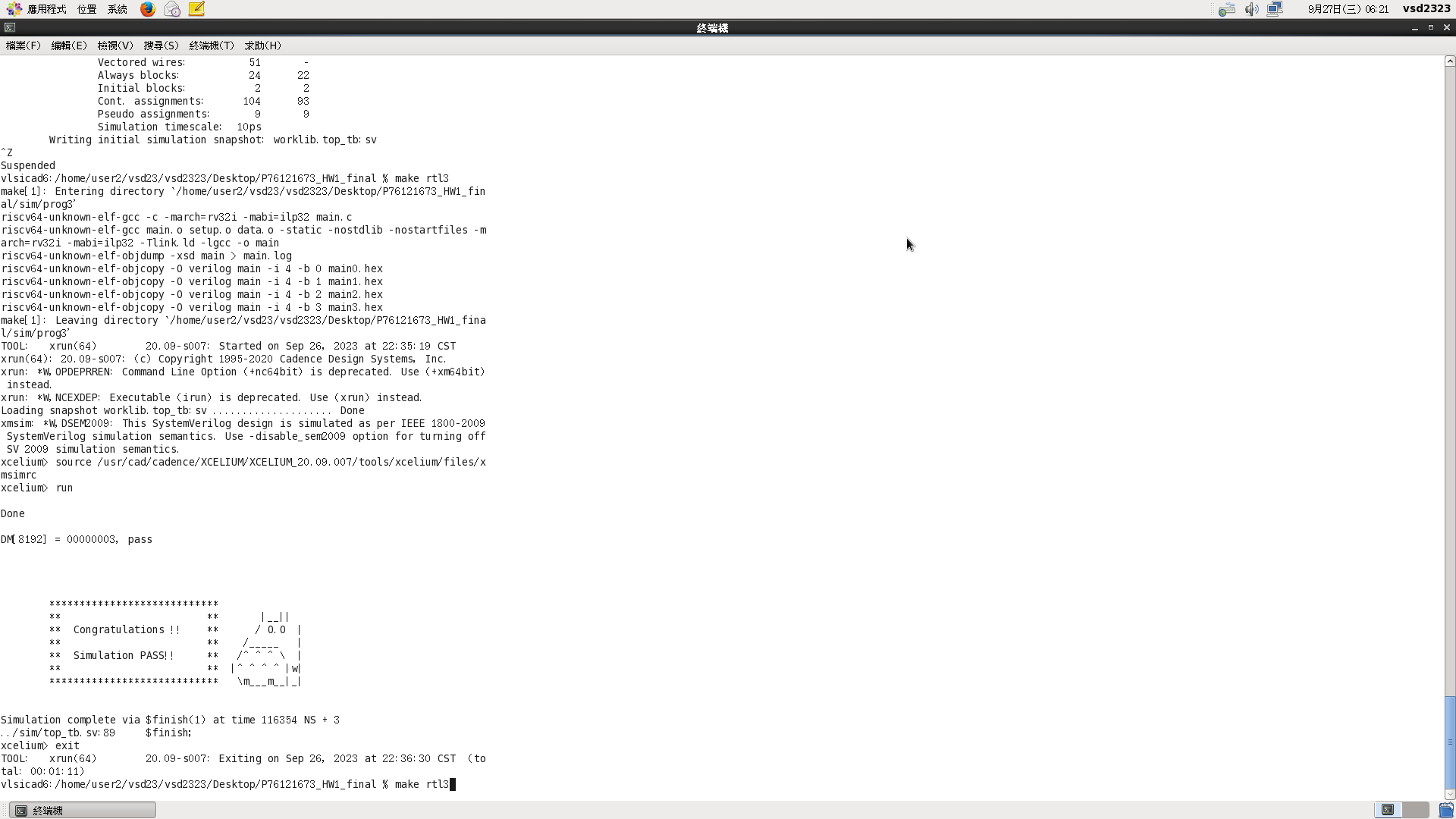
Program1:

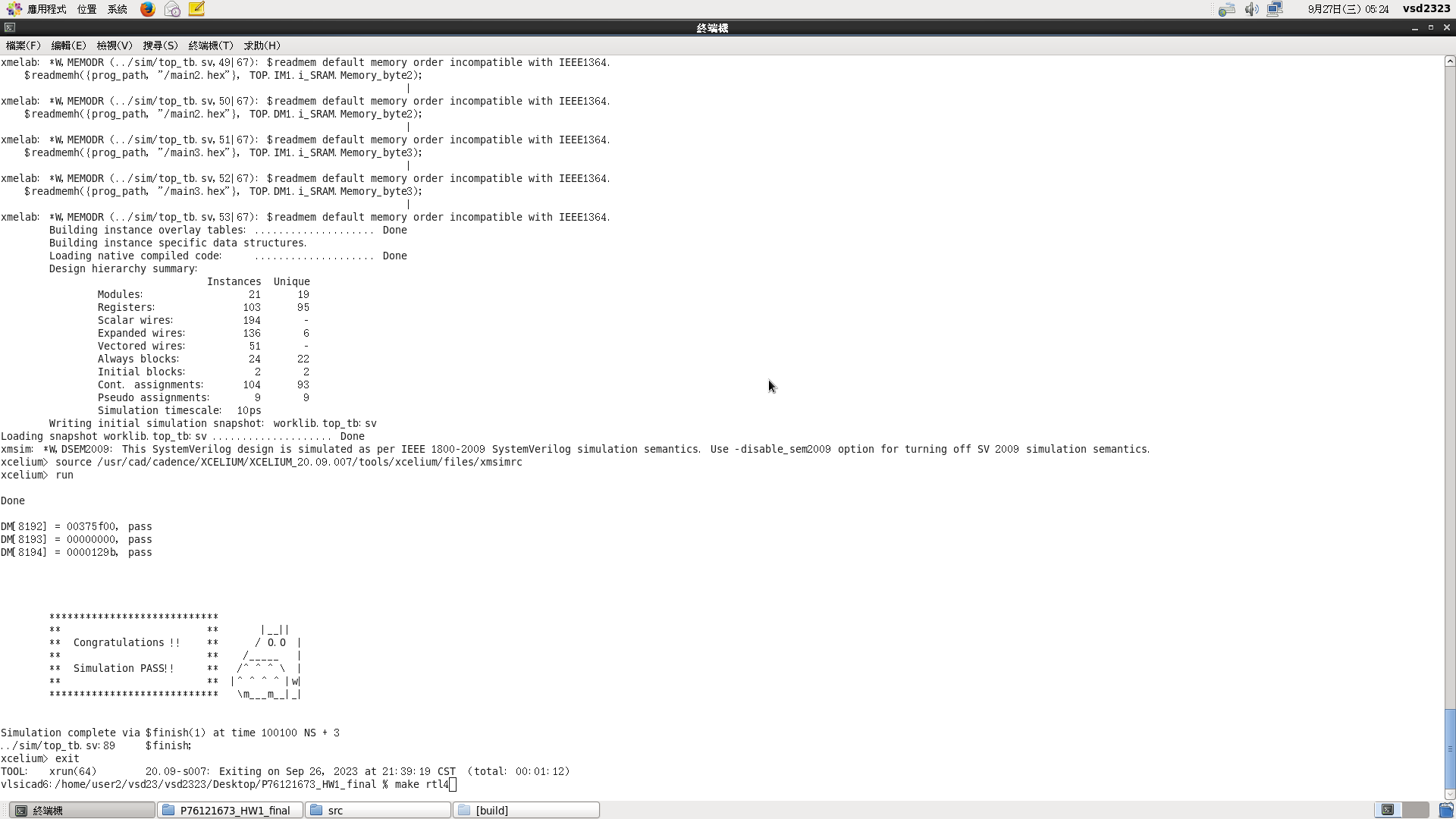
Program2:

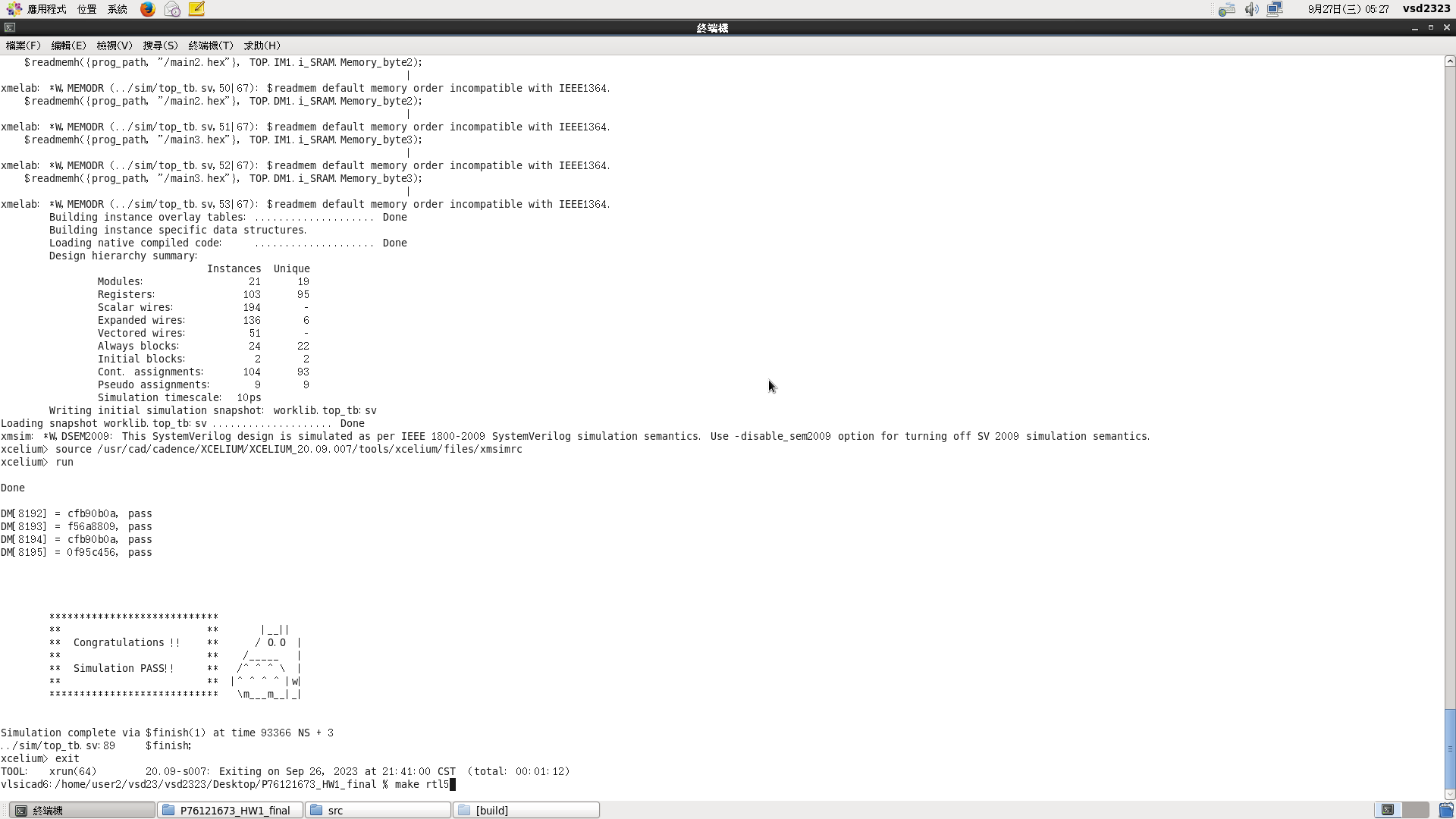
Program3:

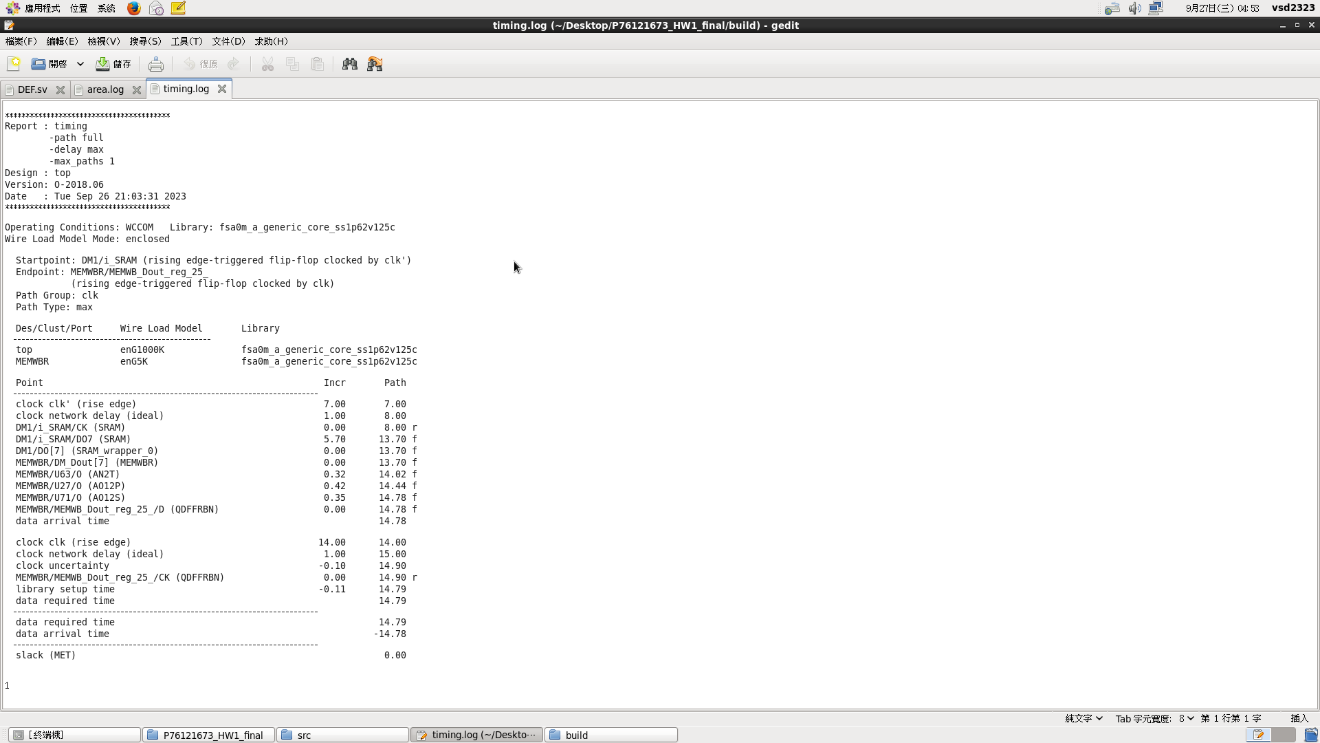
Program4:

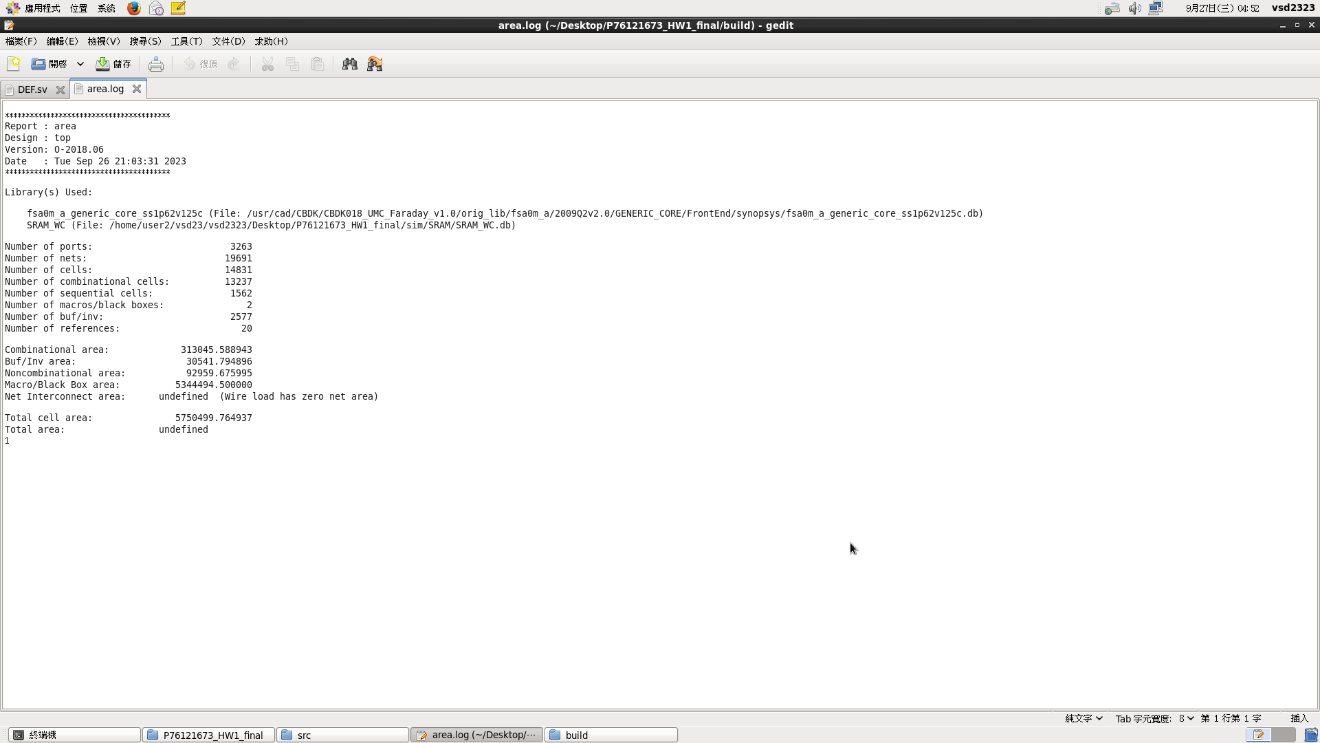
Program5:

1. Timing report :



1. Area report :

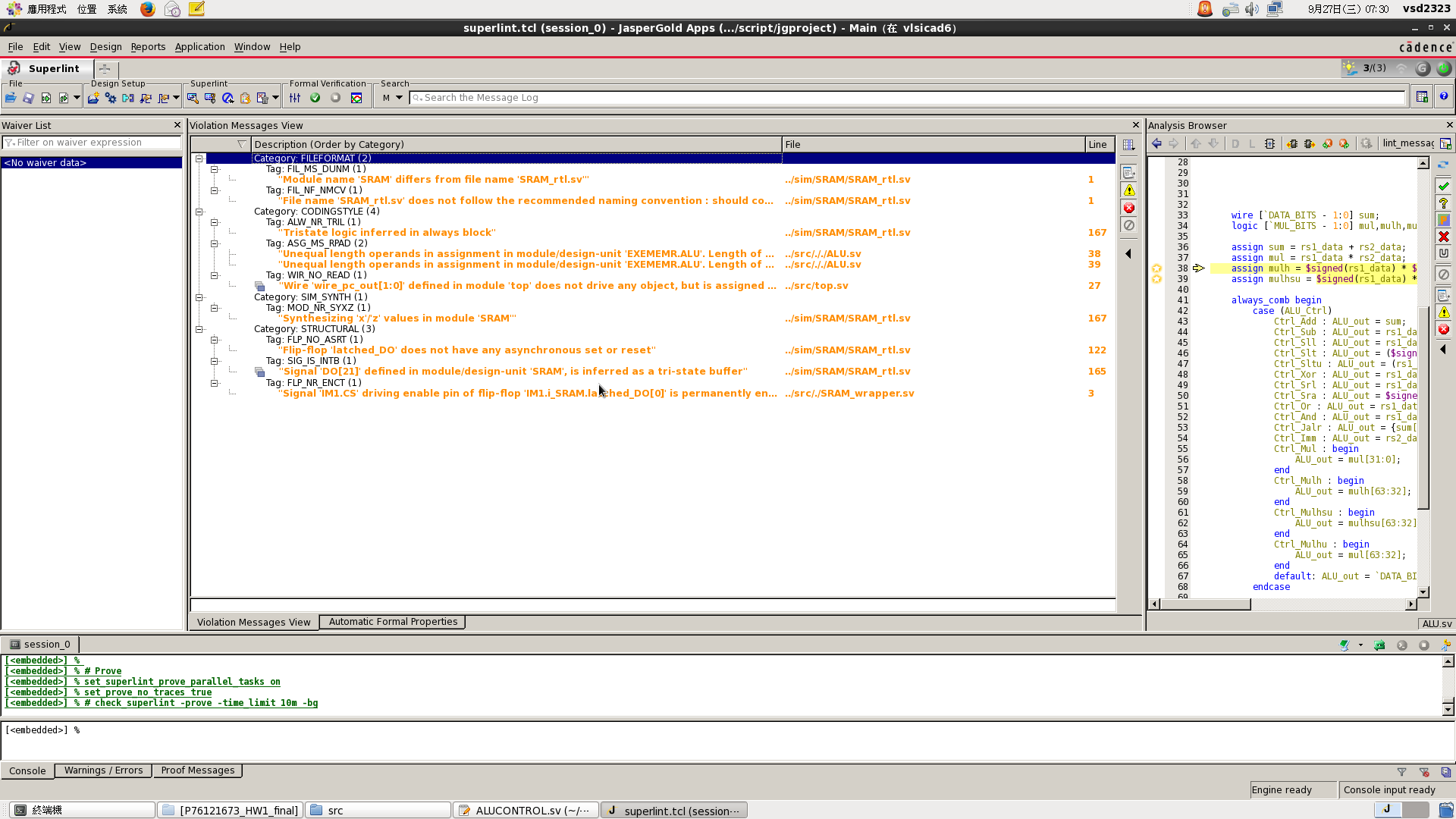


1. Superlint report :

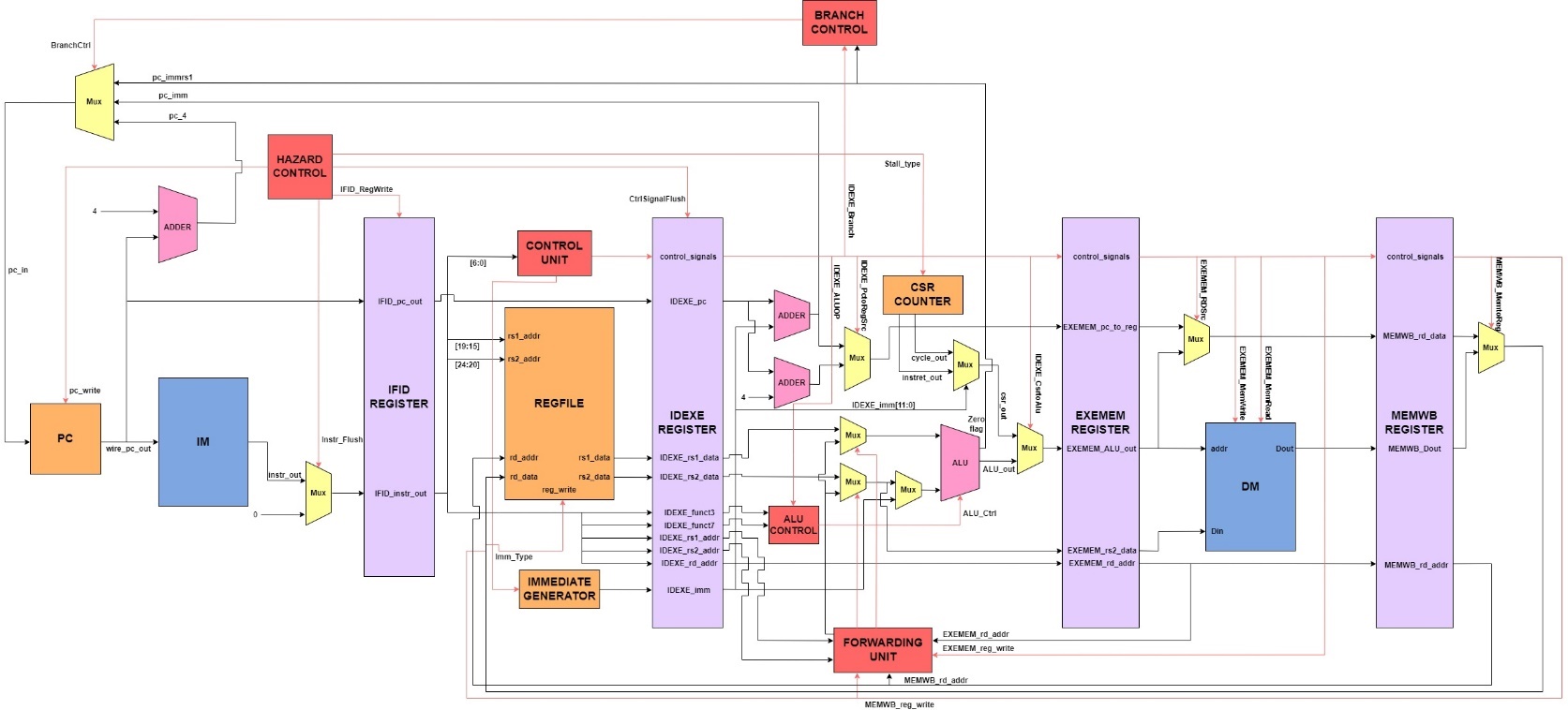
總行數 : 1476

Warning數 : 10

1. (10/1476) \* 100% = 99.33%



1. Block diagram :



1. Description of my module :

* PC : 在rst = 1的時候會初始化pc\_out = 0，每個clk 正緣來的時候就會讓pc\_out = pc\_in
* Register File : 在rst = 1的時候會初始化所有暫存器x0 = 0，read data階段會根據收到的addr給出相對應的data，write data階段則是會在clk正緣時，根據RegWrite的值，將data用不同的方式(LW, R-type…)存進Register File
* Immediate Generator : 會根據Control Unit所傳過來的ImmType將Instruction轉換成對應的Immediate。
* CSR Counter : 根據stall\_type來計算instruction及cycle的數量並輸出instret\_out及cycle\_out

1. 若沒有stall發生時，則instruction數量+1
2. stall\_type為Data Hazard時，因為下一個instruction 是NOP，所以instruction數量不變
3. stall\_type為Control Hazard時，因為下兩個instruction 是NOP，所以instruction數量需要減-1

* ALU : 根據ALU Control傳來的ALU\_Ctrl將兩個input作相對應的運算並輸出ALU\_out及Zeroflag
* Hazard\_Control : 根據Control Unit產生的MemRead以及Branch Control 產生的BranchCtrl加上上一個instruction的destination register的addr來決定IF以及ID 兩個stage是否要flush

1. Data Hazard :

PC\_write = 0x0 //因為下一個PC不用進來

Instr\_flush = 0x0 //保留原本的Instruction

IFID\_RegWrite = 0x0 //IF-stage的instruction再做一次

CtrlSignalFlush = 0x1 //ID-stage的instruction再做一次

1. Control Hazard :

PC\_write = 0x1 //下一個PC進來

Instr\_flush = 0x1 //Flush Instruction

IFID\_RegWrite = 0x1 //Flush IF-stage的instruction

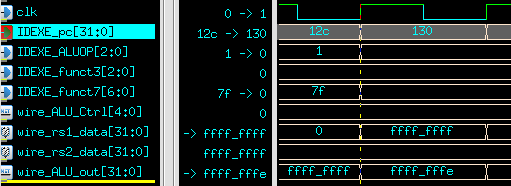
CtrlSignalFlush = 0x1 //Flush ID-stage的instruction

* Control Unit : 根據opcode(Instruction[6:0])產生相對應的控制訊號
* Branch Control : 根據Control Unit產生的Branch以及ALU產生的Zeroflag決定接下來的Branch種類(PC+4 or PC+imm or rs1+imm)
* ALU Control : 根據ALUOP以及funct3(Instruction[14:12])和funct7(Instruction[31:25])決定要傳到ALU的ALU\_Ctrl
* Forwarding Unit : 根據EXE stage的rs1 addr以及rs2 addr和MEM stage 以及 WB stage的rd\_addr來決定EXE stage當下的rs1 data 以及rs2 data是否需要Forwarding

1. Waveform Verification:

**R-type :**

**ADD :**



根據main.log可得知PC = 0x130為 add t0,t0,t1，

ALU\_Ctrl = 0x0 (ALU做ADD運算)，

ALUOP = 0 (R-type) ，

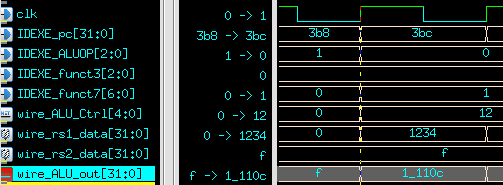
funct3 = 0x0，funct7 = 0x0，

rs1\_data = 0xffffffff

rs2\_data = 0xffffffff

ALU\_out = 0xfffffffe

**MUL :**



根據main.log可得知PC = 0x3bc為 mul t0,t0,t1，

ALU\_Ctrl = 0x12 (ALU做MUL運算)，

ALUOP = 0 (R-type) ，

funct3 = 0x0，funct7 = 0x1，

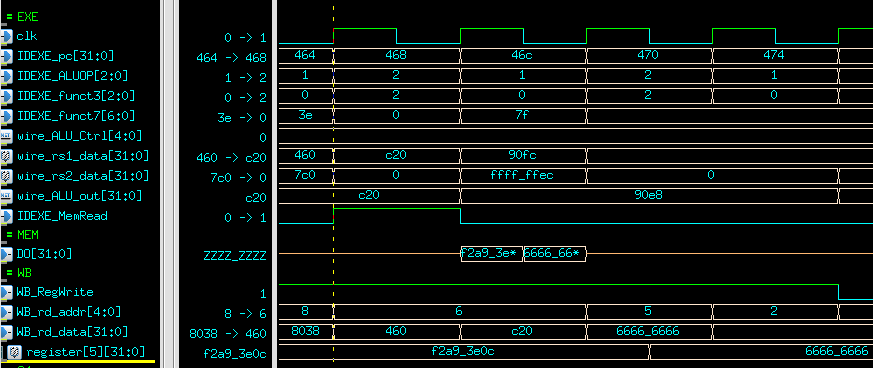
rs1\_data = 0x00001234

rs2\_data = 0x0000000f

ALU\_out = 0x0001110c

**I-type :**

**LW :**

****

根據main.log可得知PC = 0x468為 lw t0,0(t1)，

ALU\_Ctrl = 0x0(ALU做ADD運算)，

ALUOP = 2 (ADD-type) ，

funct3 = 0x2，funct7 = 0x0，

rs1\_data = 0x00000c20

rs2\_data = 0x00000000

ALU\_out = 0x00000c20 //計算存取位址

MemRead = 0x1

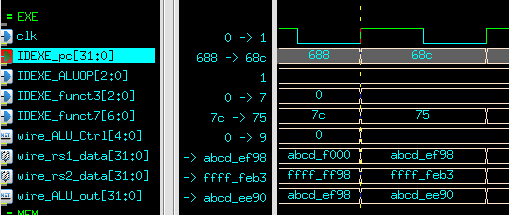
DO = 0x66666666 //取出Data Memory中0xc20位置的值

WB\_RegWrite = 0x1

WB\_rd\_addr = 0x5 //存回register t0

將DO = 0x66666666 存回t0 (register[5])

**ANDI :**

****

根據main.log可得知PC = 0x68c為 andi t0,t0,-333，

ALU\_Ctrl = 0x9 (ALU做AND運算)，

ALUOP = 1 (I-type) ，

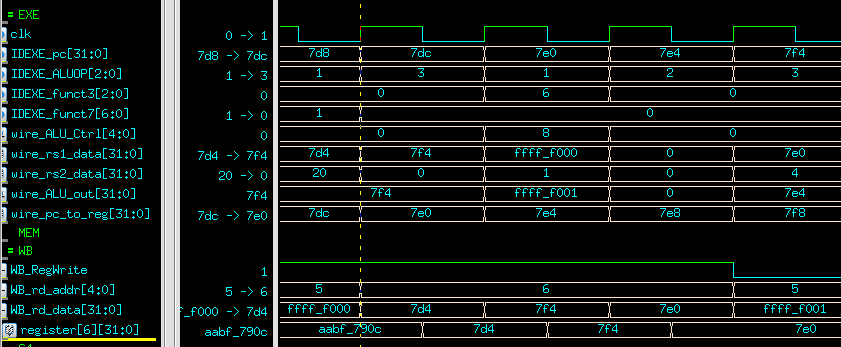
funct3 = 0x7，funct7 = 0x75，

rs1\_data = 0xABCDEF98

rs2\_data = 0xFFFFFEB3 // -333

ALU\_out = 0xABCDEE90

**JALR :**

****

根據main.log可得知PC = 0x7dc為jalr t1,t1，

ALU\_Ctrl = 0x0(ALU做ADD運算)，

ALUOP = 3 (JALR-type) ，

funct3 = 0x0，funct7 = 0x0，

rs1\_data = 0x000007f4 // rs1

rs2\_data = 0x00000000 // imm

ALU\_out = 0x000007f4 // rs1+imm

wire\_pc\_to\_reg = 0x7e0 // 選擇PC+4 (0x7e0 = 0x7dc + 0x4)

WB\_RegWrite = 0x1

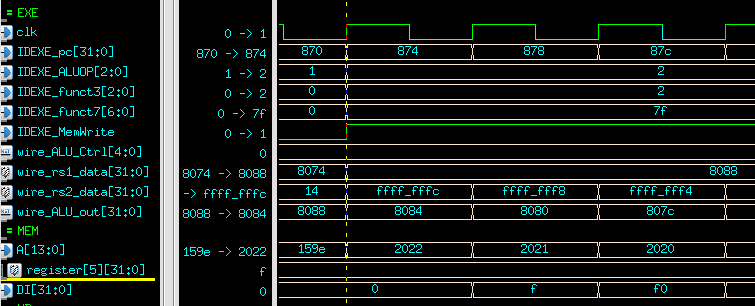
WB\_rd\_addr = 0x6 //存回register t1

將WB\_rd\_data = 0x7e0存回t1 (register[6])

藍色框中的PC為 0x7f4 (PC = rs1 + imm)，jump指令必須在EXE stage才可算出，所以兩個clock後PC值才會正確

**S-type :**

**SW :**

****

根據main.log可得知PC = 0x874為 sw t0,-4(s0)，

ALU\_Ctrl = 0x0(ALU做ADD運算)，

ALUOP = 2 (ADD-type) ，

funct3 = 0x2，funct7 = 0x7f，

rs1\_data = 0x00008088 // rs1

rs2\_data = 0xFFFFFFFC // -4

ALU\_out = 0x00008084 // 計算存取位址

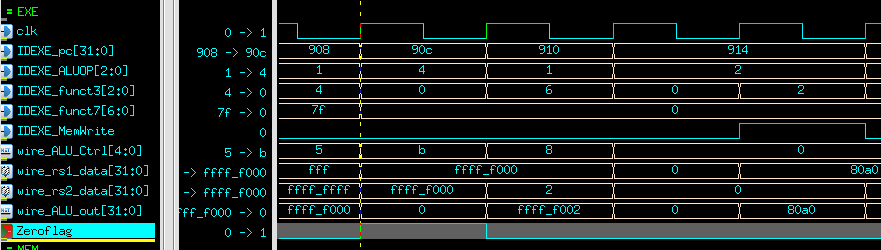
MemWrite = 0x1

DI = 0x0000000f //要存入Data Memory的值(來自register[5])

A = 0x2021 //存入Data Memory中的位置 (0x8084/4 = 0x2021)

**B-type :**

**BEQ :**

****

根據main.log可得知PC = 0x90c為beq t0,t1,914，

ALU\_Ctrl = 0xb(ALU做BEQ運算)，

ALUOP = 4 (B-type) ，

funct3 = 0x0，funct7 = 0x0，

rs1\_data = 0xFFFFF000 // t0

rs2\_data = 0xFFFFF000 // t1

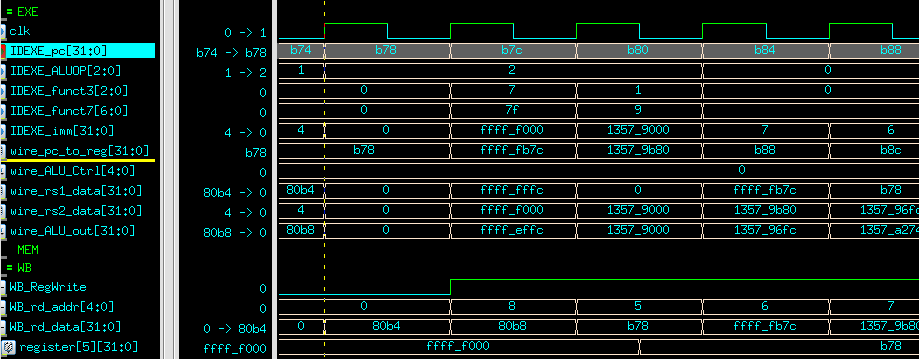
ALU\_out = 0x0 // 計算是否branch

ZeroFlag = 0x1 //需要branch

藍色框中的PC為 0x914 (PC = PC + imm)，branch指令必須在EXE stage才可算出，所以兩個clock後PC值才會正確

**U-type :**

**AUIPC :**



根據main.log可得知PC = 0xb78為 auipc t0,0x0，

ALU\_Ctrl = 0x0(ALU做ADD運算)，

ALUOP = 2 (ADD-type) ，

funct3 = 0x0，funct7 = 0x0，

PC = 0xb78

imm = 0x00000000

PC\_to\_reg = 0xb78

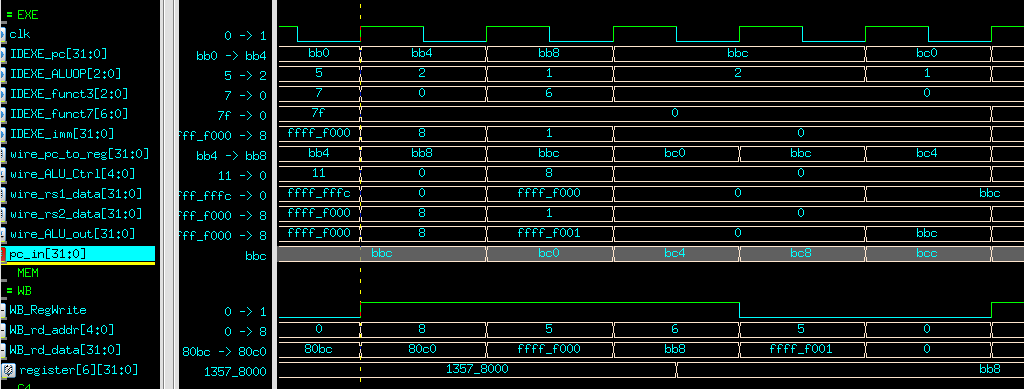
WB\_RegWrite = 0x1

WB\_rd\_addr = 0x5 //存回register t0

將WB\_rd\_data = 0xb78 存回t0 (register[5])

**J-type :**

**JAL :**

****

根據main.log可得知PC = 0xbb4為jal t1,bbc，

ALU\_Ctrl = 0x0(ALU做ADD運算)，

ALUOP = 2 (ADD-type) ，

funct3 = 0x0，funct7 = 0x0，

wire\_pc\_to\_reg = 0xbb8 // 選擇PC+4 (0xbb8 = 0xbb4 + 0x4)

imm = 0x8

pc\_in = 0xbbc // 選擇 PC+imm (0xbbc = 0xbb4 + 0x8)

WB\_RegWrite = 0x1

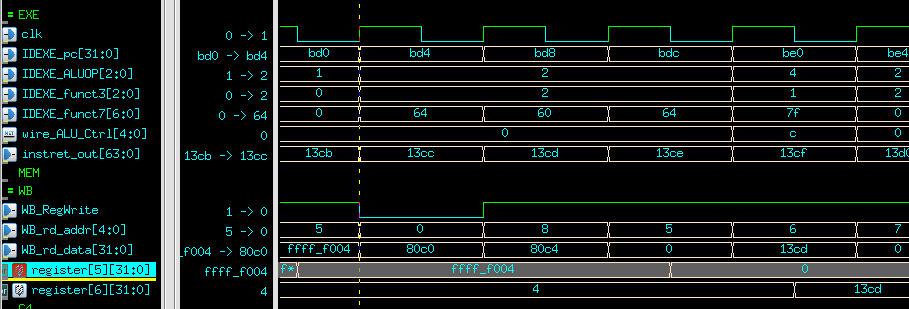
WB\_rd\_addr = 0x6 //存回register t1

將WB\_rd\_data = 0xbb8存回t1 (register[6])

藍色框中的PC為 0xbbc (PC = PC + imm)，jump指令必須在EXE stage才可算出，所以兩個clock後PC值才會正確

**CSR-type :**

**RDINSTRETH , RDINSTRET :**

****

根據main.log可得知PC = 0xbd4為 rdinstreth t0 ,

PC = 0xbd8為 rdinstret t1，

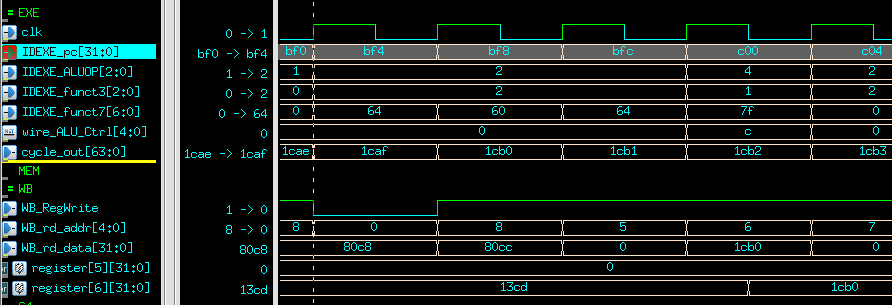
PC = 0xbd4時 , instruction的數量為0x000013cc

所以存回t0(register[5])的值為0x0

PC = 0xbd8時 , instruction的數量為0x000013cd

所以存回t1(register[6])的值為0x13cd

**RDCYCLEH , RDCYCLE :**

****

根據main.log可得知PC = 0xbf4為 rdcycleh t0 ,

PC = 0xbf8為 rdcyclet t1，

PC = 0xbf4時 , cycle的數量為0x00001caf

所以存回t0(register[5])的值為0x0

PC = 0xbf8時 , cycle的數量為0x00001cb0

所以存回t1(register[6])的值為0x1cb0

1. **Lessons learned**

在實作RISC-V ISA的5-stage pipeline CPU的作業中，在這次的作業中學到了許多東西：

* RISC-V指令集架構：我重新複習了RISC-V指令集的基本結構、指令格式、不同的指令類型（例如R型、I型、S型、B型等）和功能。
* Pipeline處理：我重新複習了CPU的pipeline處理概念，包括五個不同階段（IF、ID、EXE、MEM、WB，有助於提高指令處理效率和性能。
* Control Unit：我學到了如何設計和實現CPU的控制單元，以協調各個stage的操作，並根據指令的opcode來生成相應的控制信號。
* Waveform verification：在使用波形來找出錯誤的過程中，讓我更深入的理解這次作業的觀念跟設計。
* Verilog : 這次的作業中都是由Verilog去撰寫，從最基本的使用提升到大專案上，讓我獲得更進階的經驗。

在實作RISC-V ISA的5-stage pipeline CPU的作業中，我獲得了有關計算機結構、指令集架構、Pipeline、硬體設計等方面的寶貴知識和實踐技能。所以經過這個作業以後對CPU變得非常了解，也為後面的作業打好基礎。