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: 林孟佑

Student ID: N26114324

**HW1報告大綱**

**I. Summary**

**II. Design of 5-stage RISC-V pipelined CPU**

1. **CPU架構圖**
2. **CPU module介紹**

**Part 1. CPU中的各個功能電路簡介**

**Part 2. CPU pipeline register簡介**

**Part 3. CPU controller簡介**

1. **Hazrad 問題描述與解決方式**

**III. Instruction 波型說明**

1. **R-type – add, or**
2. **I-type – addi, jalr**
3. **S-type – sw, sb, sh**
4. **B-type - beq**
5. **U-type – lui, auipc**
6. **J-type - jal**
7. **CSR – rdinst, rdcycle**

**IV. Verification**

**V. Performance and Area**

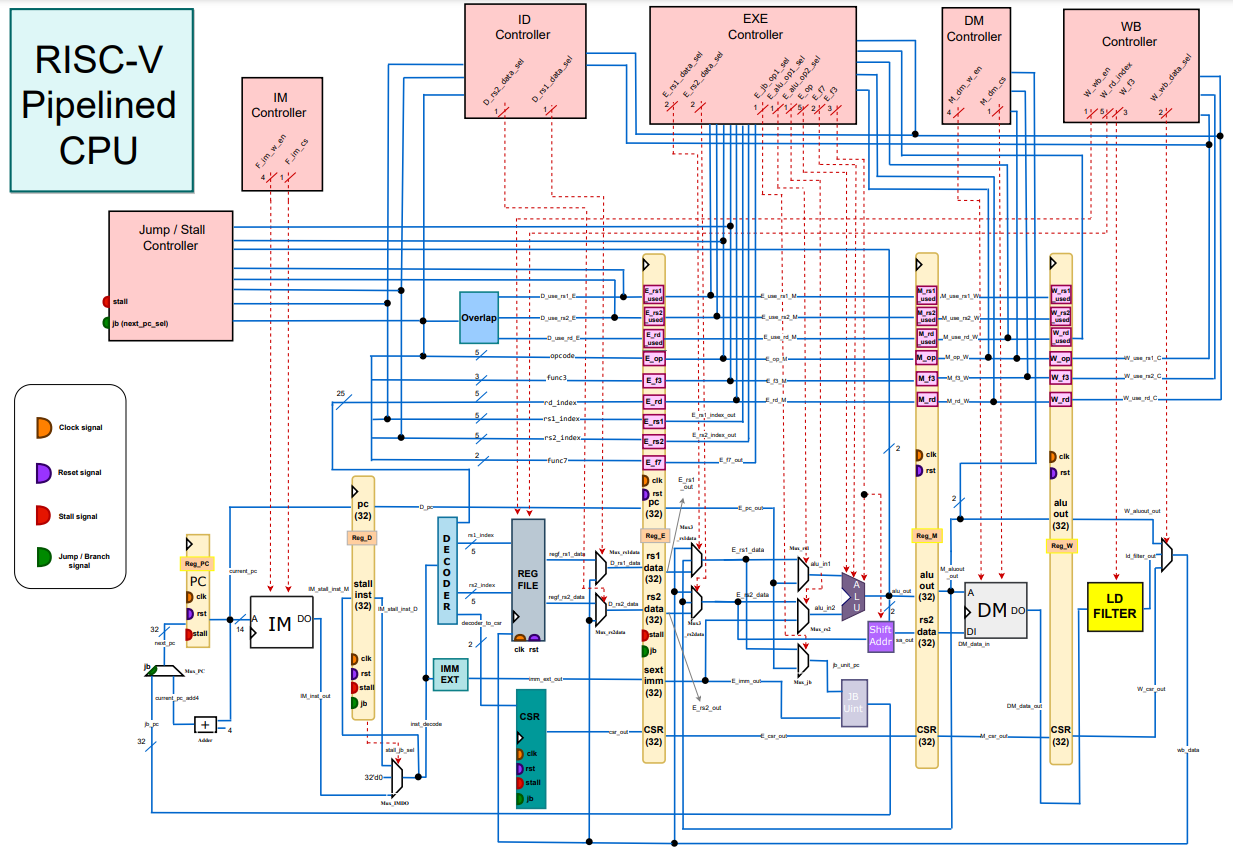
**VI. Superlint**

**VII. Lessons Learned from This Homework**

1. **Summary**

本次作業依據RISC-V指令集，設計出能夠執行41種指令的5-stage pipeline CPU，且能夠處理pipelined架構會產生的structure hazard、data hazard以及control hazard問題，完成後能夠通過prog0-prog5，並在電路合成後通過syn0-syn5，以驗證電路的功能。程式碼在superlint中沒有出現warning與error的比例需在85%以上。

1. **Design of 5-stage RISC-V pipelined CPU**

**1. CPU架構圖**

**2. CPU使用到的所有module:**

**Part1** - Pipelined CPU使用的功能電路簡介:

1. **Adder:**

Combinational電路，32 bits的加法器，將pc與4相加，用來計算非跳轉指令下一條指令的位址。

1. **ALU:**

Combinational電路，算術邏輯單元，對不同指令進行相對應的運算，計算出的結果可能為data、memory位址或branch指令的比較結果。

1. **Decoder:**

Combinational電路，32 bits的解碼器，根據32 bits指令的各區塊將指令解碼並輸出給Register file和 Controller使用。

1. **Imm\_ext:**

Combinational電路，依據不同的指令型式將立即數extend為32 bits的形式。

1. **JB\_unit:**

Combinational電路，用來計算跳轉指令的pc，由於跳轉只會跳到2的倍數位址(能夠支援compressed指令)，因此會做(op1 + op2) & (~32’d1)。

1. **LD\_Filter:**

Combinational電路，依據Load指令的不同類型(lb、lh、lw、lbu和lhu)處理data的bits，使其符合對應的結果。

1. **RegFile:**

Sequential電路與Combinational電路組成。

Sequential電路: 當clk來且rst訊號為high時，將register x0設為0。

當clk來且wb\_en為high時，會將資料write back回rd，wb\_en為low時rd的data則保持不變。此外，corner case為當rd為x0時，不需將資料write back。

Combinational電路: 根據Decoder輸出的rs1\_index與rs2\_index，將register file的data讀取出來。

1. **Shift\_address:**

Combinational電路，用於store指令，會根據ALU算出的地址後兩個bits，判斷store中的sb和sh是否需要存進非4-byte aligned的地址，並對資料進行處理。

1. **Overlap:**

Combinational電路，依據opcode判斷指令是否需要使用rs1、rs2或rd，並將結果傳至pipelined register。

**(10) CSR\_module:**

Sequential電路與Combinational電路組成。

**Sequential電路:** 在clk訊號為high時更新已運行的cycle數與instruction數。

**Combinational電路:** 將cycle數與instruction輸出至pipelined register。

**Part2 –** Pipeline register簡介**:**

**(1) Reg\_PC:**

當rst訊號為high時，將pc設為0(預設0為初始位址)，當clk來時將current\_pc更新為next\_pc。

**(2) Reg\_D:**

接收Reg\_PC傳入的PC位址和保存CPU stall時的指令，當clk為high時把資料傳給下一個stage。

**(3) Reg\_E:**

接收並保存decoder所解碼的指令資訊、Overlap輸出的訊號、Reg\_D傳入的PC位址以及正確的RegFile資料(會透過mux與ID controller的控制訊號決定是否需要forwarding)、Imm\_ext展開的32 bits立即數與CSR資料，這級保存的資料會給EXE controller使用，當clk為high時更新資料並把這級的資料傳給下一級。

**(4) Reg\_M:**

接收並保存Reg\_E傳出的指令資訊、Overlap資訊、CSR資料、ALU輸出的資料、Shift address傳入的資料，這級保存的資料會給DM controller使用，當clk為high時把資料傳給下一個stage。

**(5) Reg\_W:**

接收並保存Reg\_M傳出的指令資訊、Overlap資訊、CSR資料和ALU輸出的資料、當clk為high時把資料傳給下一個stage。

**Part3** - CPU Controller簡介:

**(1) Jump / Stall Controller:**

接收 decoder解碼的instruction opcode、rs1\_index、rs2\_index，ID stage 的Overlap資訊，EXE stage的opcode與rd\_index和ALU的最後1個bit，判斷地址是否需要跳轉或是CPU需要停滯一個cycle，並輸出jump signal或stall signal。

**(2) ID Controller:**

接收 decoder解碼的instruction opcode、rs1\_index、rs2\_index，ID stage 、WB stage的Overlap資訊與WB stage的opcode與rd\_index，並輸出ID stage 的Mux控制訊號，選擇正確的資料。

**(3) EXE Controller:**

接收 EXE stage的instruction opcode、function3、function7、rs1\_index、rs2\_index、rd\_index，IM stage 的instruction opcode、rd\_index，WB stage的instruction opcode、rd\_index資訊，輸出EXE stage 的Mux、ALU、Shift Address控制訊號。

**(4) DM Controller:**

接收IM stage 的instruction opcode、function3與Reg\_M的alu後兩個bits，輸出DM的控制訊號。

**(5) WB Controller:**

接收WB stage 的instruction opcode、function3、rd\_index，輸出WB stage 的Mux控制訊號、LD filter的控制訊號以及Regfile的控制訊號。

**3. Hazard的問題與處理方式:**

**(1) Structure hazard:**

**問題說明:**

由於Pipelined CPU會同時執行不同的指令，兩條指令會同時需要使用memory。

**解決方式:**

複製兩份SRAM(im與dm)，分別讓IF stage與MEM stage使用，避免不同stage的硬體競爭情形。

**(2) Control hazard:**

**問題說明:**

由於branch與jump指令需要在EX stage時計算跳轉地址，而尚未計算出跳轉地址前，接續的指令就已經進到pipeline內，此時可能會使用到錯誤的指令。

**解決方式:**

一律先假設跳轉指令不成立，並在jump / branch controller內加入flush signal(架構中的jb signal)，而當地址需要跳轉時，flush signal為high，此時會將IF stage與ID stage插入nop指令，並在pipeline中傳遞下去，此時可將錯誤的指令清除，讓跳轉指令可以正確執行。

**(3) Data hazard:**

**問題說明:**

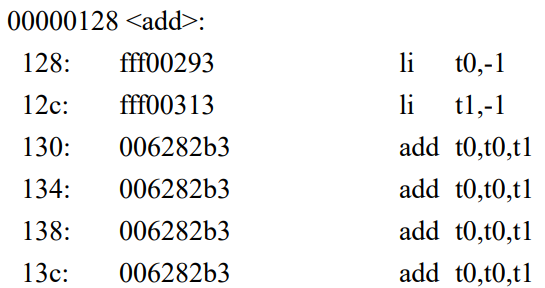
當前後指令具有dependency(rd與rs1或rs2有重疊)，可能無法使用到最新的data，進而導致結果錯誤。

**解決方式:**

根據不同的情況做forwarding，將MEM stage或WB stage的最新data forward回ID stage或EX stage，MEM stage與WB stage皆有重疊時，MEM的順位較優先。(Load指令情形較特別，需要stall一個cycle  在EX stage插入nop，並將IF stage與ID stage的指令卡住，才能正確將data forward回ID stage或EX stage)。

1. **Instruction 波型說明(以prog0為例)**

**(1) R-type:**

**(ex1) add:**

**一張含有 螢幕擷取畫面, 多媒體軟體, 軟體, 鮮豔 的圖片

自動產生的描述**

以PC從128至13c的波型來看，PC為128與13c時li指令有被ALU正確的算出，接著在130-13c時透過add運算出結果 -5(ffff\_fffc)，接著透過WB\_stage的Mux選擇並寫回Regfile。

**一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述(ex2) or:**

**一張含有 螢幕擷取畫面, 鮮豔 的圖片

自動產生的描述**

以PC從324至334的波型來看，PC為324至330時有lui與addi指令有被正確的算出為1234\_5678與fedc\_ba98，接著在PC為334時透過or運算出結果 fefc\_fef8，接著透過WB\_stage的Mux選擇並寫回Regfile。

**(2) I-type:**

**一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述(ex1) slli:**

**一張含有 螢幕擷取畫面 的圖片

自動產生的描述**

以PC從6d8至6ec的波型來看，PC為6d8時li在ALU有算出值10(0000\_000a)，並在接下來進行5次左移運算，接著透過WB\_stage的Mux選擇並寫回Regfile。

**(ex2) jalr:**

**一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述**

**一張含有 螢幕擷取畫面, 電子產品 的圖片

自動產生的描述**

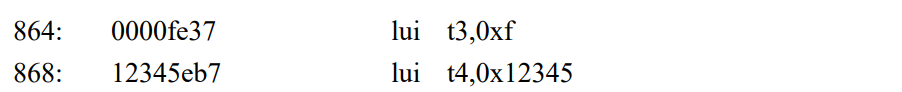
以PC從7d0至7dc的波形來看，PC為7d0至7d8時在ALU有算出值(0000\_07f4)，並在7dc進行跳轉運算，ALU算出跳回地址(7dc + 4 = 7e0)，JB\_unit則算出跳轉地址。

**(3) S-type:**

**(ex) sw, sh, sb:**

**一張含有 文字, 字型, 螢幕擷取畫面, 收據 的圖片

自動產生的描述**

****

**一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述**

**一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述**

PC從854至884一開始將資料load入register t0-t4，接著將資料以word型式存入register s0所存之地址 + offset， PC從888至8a4則將register t5的資料以不同型式存入register s0所存之地址 + offset，要注意的是，在sh與sb的情況可能會出現存入地址非4-byte aligned的問題， 資料需透過Shift\_Addr module進行處理。

**一張含有 螢幕擷取畫面, 電子產品 的圖片

自動產生的描述**

以PC從88c至89c的波形來看，Shift\_Addr有分別處理要寫入DM資料以sw, sb, sh的形式以及寫入DM的對應地址，資料處理後為1234\_5678、7800\_0000與5678\_0000，後續再寫入DM。

**(4) B-type:**

**一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述(ex) beq:**

**一張含有 螢幕擷取畫面, 軟體, 多媒體軟體 的圖片

自動產生的描述**

以PC從8bc至8dc的波形來看，PC為8bc至8c8時在ALU有算出register t0與t1值，並在PC為8cc至8dc時進行比較，ALU算出的值皆為0，此時地址不進行跳轉，next\_pc維持為PC+4。

**(5) U-type:**

**(ex1) lui:**

**一張含有 文字, 字型, 螢幕擷取畫面, 白色 的圖片

自動產生的描述**

**一張含有 螢幕擷取畫面, 多媒體軟體, 軟體 的圖片

自動產生的描述**

以PC從b94至b9c的波形來看，lui將immediate放到最高位的5個bits，ALU算出的結果分別為0000\_0000、ffff\_f000、1357\_9000 。

**(ex2) auipc:**

**一張含有 文字, 字型, 螢幕擷取畫面, 白色 的圖片

自動產生的描述**

**一張含有 螢幕擷取畫面, 軟體, 多媒體軟體 的圖片

自動產生的描述**

以PC從b78至b80的波形來看，auipc將immediate放到最高位的5個bits再與PC相加，ALU算出的結果分別為0000\_0b78、ffff\_fb7c、1357\_9b80 。

**(6) J-type:**

**(ex) jal:**

**一張含有 文字, 字型, 螢幕擷取畫面, 白色 的圖片

自動產生的描述**

**一張含有 螢幕擷取畫面, 電子產品 的圖片

自動產生的描述**

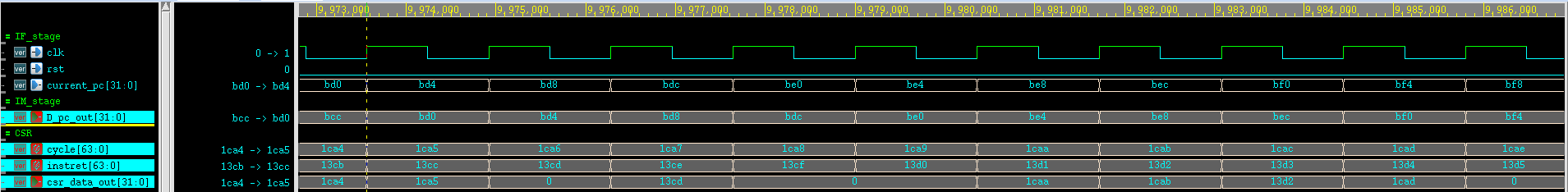
以PC從bb4的波形來看，jal的ALU算出PC + 4 = bb8並存回 Regfile，JB\_unit則算出跳轉地址 = bbc。

**(7) CSR:**

**(ex1) rdinst**

**一張含有 文字, 字型, 螢幕擷取畫面, 白色 的圖片

自動產生的描述**

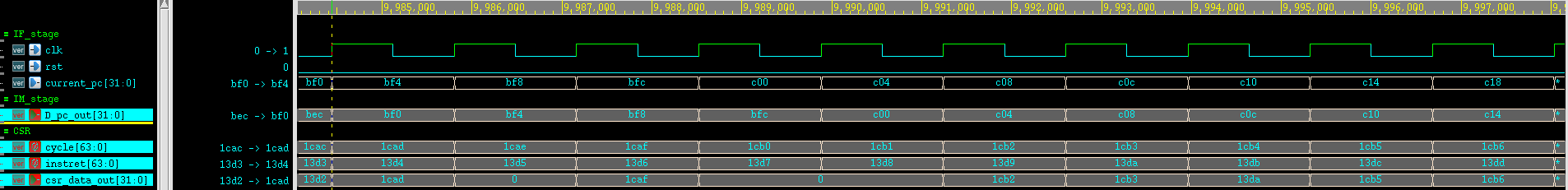
****

以PC從bd4-bdc的波形來看，csr\_data\_out根據rdinstreth或rdinstret輸出64-bit的instruction register的高32位元或低32位元的值。

**(ex2) rdcycle**

**一張含有 文字, 字型, 螢幕擷取畫面, 白色 的圖片

自動產生的描述**

****

以PC從bf4-bfc的波形來看，csr\_data\_out根據rdcycleh或rdcycle輸出64-bit的cycle register的高32位元或低32位元的值。

1. **Verification**
   1. 一張含有 文字, 螢幕擷取畫面, 陳列, 電腦 的圖片

      自動產生的描述 Make syn0 (2) Make syn1

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(3) Make syn2 (4) Make syn3

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

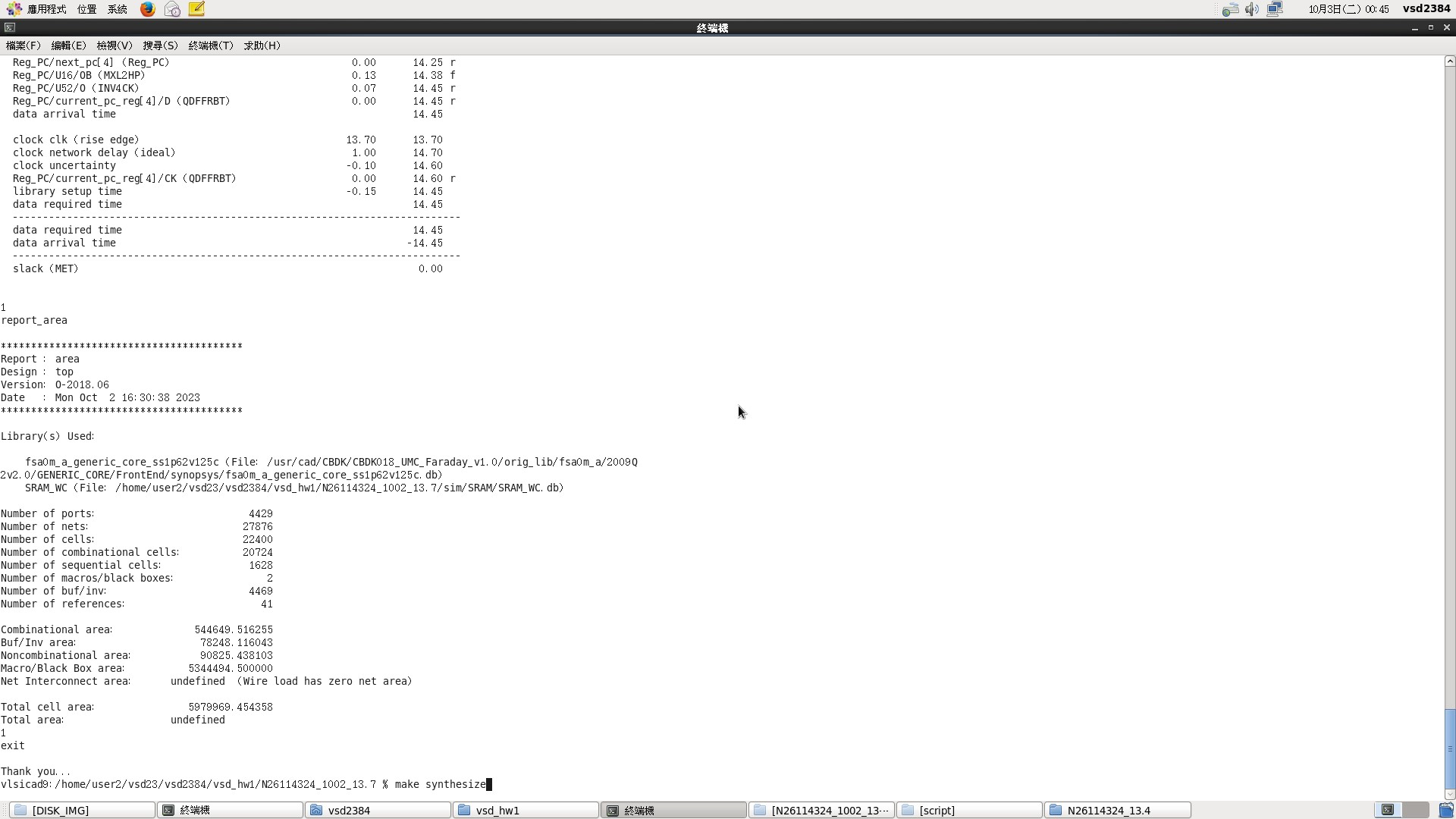
自動產生的描述(5) Make syn4 (6) Make syn5

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. **Performance and Area**

Clock Cycle:13.7 ns



Area: 5979969.454358 um2

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**

依據以下方式修改warning的code:

* 1. 所有sequential電路都改為非同步reset。
  2. case的條件必須要寫滿或是加入default，即便不在意輸出結果，也必須輸出值(補0)，否則會合出latch，而當case的條件有寫滿時不用加入default。
  3. 將數字之bit width補上，等號左值與右值為相同bit width。

最後程式碼總行數為2193條，warning為9條

Warning佔總行數百分比為99.59%

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. **Lessons Learned from This Homework**

本次作業透過實作讓我對RISC-V指令集與Pipelined CPU有了更深的認識，這也是我第一次學習工作站與EDA tool的使用，有非常多新的知識需要學習，這次作業遇到最大的難關就是在第一次模擬全部pass且在clock cycle為20.0下合成slack為正數的電路後，模擬卻沒有通過，在不斷的檢查與嘗試後才發現，sequential電路沒有完全初始化，或是combinational電路的運算太複雜導致算不完，都可能導致合成後的電路，在需要考量cycle時間的情形下無法正常動作。這些細節都是日後需要詳細思索與注意的，在完成作業後有嘗試將cycle時間壓低，發現在時間為13.7以內時slack為負值，但電路仍可以通過合成後模擬，因此把critical path縮短應可以再降低clock cycle時間，也是日後可以持續努力的方向。