VLSI System Design (Graduate Level)

Fall 2024

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: \_王劭懷\_\_\_\_\_\_

Student ID: \_M16121077\_\_\_\_\_

1. **Summary**
2. prog0 –測試49個instructions

Simulation time : 10374700 ps

Result : All Pass

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

1. prog1- Sort Algorithm

Simulation time : 38226300 ps

Result : All Pass

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

1. prog2 – 不使用MUL/MUL[[S]U] instructions 完成Multiplication

Simulation time : 10349500 ps

Result : All Pass

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

1. prog3 – Greatest common divisor

Simulation time : 10731700 ps

Result : All Pass

一張含有 文字, 螢幕擷取畫面, 多媒體軟體, 繪圖軟體 的圖片

自動產生的描述

1. prog4 – 使用階乘c code測試rdinstret, rdinstreth, rdcycle, rdcycleh

Simulation time : 10009300 ps

Result : All Pass

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

1. prog5 – 使用MUL/MUL[[S]U] instructions實現Multiplication

Simulation time : 9335900 ps

Result : All Pass

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

1. prog6 – 使用floating point instructions進行運算

Simulation time : 9293900 ps

Result : All Pass

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

1. Lines

Total Lines : 3667

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

1. Superlint

Warning : 16

Superlint warning / line : 16/3667 0.4%

一張含有 文字, 螢幕擷取畫面, 軟體, 網頁 的圖片

自動產生的描述

在Superlint中最常遇到的問題

1. CAS\_NO\_DEFA

解決方法 : 將使用到case的地方加入default ，使條件皆有相對應的結果，進而避免產生latch。

1. SIG\_NO\_USED

解決方法 : 將無使用到的port或signal刪除。

1. INS\_MS\_PSIZ

解決方法 : 將每個input 、 output port 及相連的wire的bit數對齊，不要有宣告長度不同的地方。

1. IDN\_NR\_SVKW

解決方法: 避免使用到verilog的保留字，或是在變數使用全大寫的方式

1. CST\_NO\_BWID

解決方法: 將每個直接assign為常數的bit width宣告與欲放入的logic長度一致

1. Area

Area : 16781.819663

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

1. **Results by waveform**
2. I-type

一張含有 螢幕擷取畫面, 行 的圖片

自動產生的描述

fff0\_0293為addi x5, x0, -1，因此在ID級從IM讀出指令並解碼後後，在EXE級做解碼，將x0暫存器(data = 0)的值與-1 (32’hffff\_ffff)相加，結果為aluout (32’hffff\_ffff)，並在經過MEM級後於WB級寫回暫存器x5，writedata為先前alu執行運算的結果(32’hffff\_ffff)。

1. R-type

一張含有 螢幕擷取畫面, 多媒體軟體 的圖片

自動產生的描述

6282b3的指令為 add x5, x5, x6，因此在ID級IM讀出指令後，進入EX級做運算，但因為x5在前兩個指令(32’hfff0\_0293，見前圖)正好寫回32’hffff\_ffff，因此直接將欲write back的值32’hffff\_ffff做forwarding至aluout的in1，當作欲運算的rs1的值。因為前一個指令(32’hfff0\_0313)為addi x6, x0, -1，將-1的值寫入x6暫存器，因此將MEM級中已運算完欲寫回x6暫存器的值forwarding到alu的in2做運算，並得到結果aluout(32’hffff\_fffe)，最後在經過MEM級後於WB級將aluout的結果寫回x5暫存器。

1. B-type

一張含有 螢幕擷取畫面, 多媒體軟體, 軟體, 繪圖軟體 的圖片

自動產生的描述

530c63的指令為beq x6, x5, 24，但是此時在EXE級的判斷中branch\_check不成立，故address不會更動，仍然執行原先預計的下個指令。

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

628463的指令為beq x5, x6, 8，此時x5與x6暫存器的值相同，命名為data1與data2是因為加入fowarding與src的考量，而相同時會將datapc的值輸入到pc的input中，如EX級中pcin應該為原先的pc+4 (即16’h0918)，然而，因為branch\_check與branch控制訊號同時成立，故將pcin改為datapc的值(16’h0914)，且將原先讀入的兩個指令(16’h910、16’h914 ) flush掉，隔兩個delay後繼續執行datapc的那個指令。

1. U-type

一張含有 螢幕擷取畫面, 文字, 電子產品, 多媒體軟體 的圖片

自動產生的描述

在b78的指令32’h0000\_0297，亦為auipc x5, 0，因此在EXE將{immdata[31:12],12'd0} + {16'd0,pcdata}得出pcimm，並且在WB級寫回register x5。

一張含有 螢幕擷取畫面, 電子產品 的圖片

自動產生的描述

ffff\_f337的指令為lui x6, -1，在EX執行將immdata候補12’d0的值放入datafromalu，並寫在WB級時將此output寫入register x5中。

1. J-type

一張含有 螢幕擷取畫面, 多媒體軟體 的圖片

自動產生的描述

80036f的指令為jal x6, 8，於EXE級計算出PC+imm的值{16’d0,pcdata}+immdata，因為計算方式與auipc不同，故需一個auipc的訊號控制。而將計算完的datapc放入pc的input作為下一個要執行的指令pc，且將pcnext(命名為pcout)於WB級中放入暫存器x6中。而在jump執行時與branch相同會先讀取兩個指令，因此欲先讀取進來的指令(bb8、bbc)會因為Jump的成立而被flush掉。

1. F-type

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

32787的指令為flw f15, 0(x6)，此時讀出registerfile中x6位址的值，並與Immdata(0)做相加作為Data Memory的位址要求，而因為EX級計算出來的aluout應該除以4才是Address，故ce0對應的addresss assign為338，並在下一個cycle(MEM級)輸出data (32’hbe3a\_a25e)，在WB級時寫回暫存器f15，而因為本次的要求，我將兩個32個位址的 register file合併為一個，並將floating point的暫存器儲存在原暫存器位址加32的地方。如上圖所示，因為WB時需放在Floating point的位址，故registerfile同時收到兩訊號後會將原位址5’h0f改為6’h2f做存放。

一張含有 螢幕擷取畫面, 文字, 設計 的圖片

自動產生的描述

Fee4\_2e27的指令為fsw f14, -4(x8)，在IM級讀取暫存器rs2為floating point的暫存器(f14)，因此register file收到float\_rd2的控制訊號為1，此時將原先Input位置5’h0e更改為6’h2e，讀取到floating point位置的暫存器，並在EX級時讀出該暫存器的值，直接放入DM的Data In(DI)中。在Memory位置的計算上為普通暫存器x8加上-4，而因為前面的指令將會寫入x8新的值，故會做forwarding，如黃色框起處所示。

一張含有 螢幕擷取畫面, 文字, 行 的圖片

自動產生的描述

F7\_77d3的指令為fadd.s f15, f14 , f15，由於前面一個指令是flw fa5, 0(t1)，有Load\_use hazard，故會暫停一個cycle，因此實際的ID、EX、MEM、WB都會延後一個cycle，而在ID級因為rs1，rs2都是floating point的暫存器，故此時floatrd1、floatrd2的flag都為1，並讀出對應位置的data，在EXE做IEEE-754 floating point的計算後得出aluout，並在WB級寫回floating point暫存器中。於驗證方面則透過numeral systems的網站做核對，加速驗證的時間，如下圖所示。

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

自動產生的描述

1. CSR

一張含有 螢幕擷取畫面, 電子產品 的圖片

自動產生的描述

C0202373指令為csrrs x6, instret, x0，即將instret[31:0]寫入register x6中，且instret的作法為持續加1，遇到flush或load\_use\_hazard時則不加，然而，因為我存下來的instret是在EXE級儲存的，還要經過兩個cycle才會完成，因此在WB級時要將原先存的值減2才會是實際執行該指令前所執行的所有指令數量。RDCYCLE的做法同理，只是counter會持續加1，中間不會暫停。而再根據instruction的imm做判斷決定要儲存High的data還是low的data。

1. **Block Diagram**

一張含有 圖表, 方案, 工程製圖, 圖解 的圖片

自動產生的描述

黃色部分為register或Memory

白色部分則為Combinational circuit

1. **Lessons Learned**

在這次的作業中，我親手設計並實現了一個屬於自己的5 stage Pipelined RISC-V CPU。雖然我們在計算機組織課程中主要學習的是 MIPS 架構，但實際上，RISC-V 與 MIPS 的架構有許多相似之處。主要的差異在於能提供不同功能的運算，位址與資料的解法方式不同，與之前所學不同的還有指令記憶體（Instruction Memory，IM）和資料記憶體（Data Memory，DM）的運作方式更接近真實的記憶體模型，它們具有一個時鐘週期的延遲。也就是說，在給定地址輸入後，需要在下一個週期才能獲得對應的輸出資料。

**設計初期的挑戰**

在設計之初，缺乏全面的規劃，導致我在 IM 之後額外插入了一級取指（IF）階段，在暫存器檔案（Register File）之後又多加入了解碼（ID）階段的暫存器。雖然在一般的運算中沒有太大問題，但在處理分支（branch）、跳躍（jump）、前饋（forwarding）、危險檢測（hazard detection）等情況時，設計變得格外複雜，否則就必須承受更多不必要的延遲。

由於一開始沒有仔細思考，採用了邊寫邊想的方式，導致出現了許多 bug，而且對問題的根源感到困惑。於是，在颱風天的那幾天裡，我幾乎每分每秒都沉浸在除錯的過程中，甚至熬夜到天亮（儘管那幾天的天氣並不晴朗）。這段經歷讓我深刻體會到，沒有充分的規劃會給後續的工作帶來巨大的麻煩。

**重新審視設計，解決問題**

為了解決這些問題，我決定重新審視整個設計，特別是時序的安排。經過仔細分析，我發現可以將暫存器檔案與 ID 階段的暫存器合併，這樣可以簡化數據路徑。同時，取指（IF）階段只需要在處理跳躍、危險、分支時才需要特殊處理。當我釐清了整個架構後，設計工作變得順利了許多，也讓我深刻認識到在開始執行之前，充分思考和規劃的重要性。

**合成過程中的發現**

在進行電路合成時，我又遇到了新的挑戰。我發現，在第一個時鐘週期中，因為 IM 還沒有收到有效的輸入，會輸出未知的訊號（XX），這可能導致後續的錯誤。因此，我需要在 Register File 之前的多工器（MUX）中，選擇先輸出一條 addi x0, x0, 0 的指令作為佔位，確保電路在啟動時有一個穩定的狀態。

此外，在課堂上，老師提到使用 int 型別的變數比較好，因為它們只有兩種狀態，不會出現 x 或 z 的未知狀態。因此，我在宣告變數時採用了 int。然而，在進行合成時，發現過程耗費了極長的時間，而且無法成功完成。我反覆檢查程式碼，卻找不到問題所在。直到我查看了合成日誌檔 syn\_compile.log，才發現 int 型別的變數是不可合成的。這讓我學到了在使用任何語法或工具之前，必須先確認其是否適用於目標環境。盲目地採用可能帶來潛在的問題，增加除錯的難度。

**提升效能的嘗試**

為了提升 CPU 的效能，特別是在這次需要進行性能分析（Performance Analysis）的比較中，我嘗試了多種方法。例如，共用乘法器和其他硬體資源，減少硬體的使用量；將較長的運算，如果彼此沒有相關性，拆分成平行處理，以縮短臨界路徑（Critical Path）的時間；或者將某些判斷單元，如 Forwarding Unit，改為使用觸發器（Flip-Flop）來實現。雖然這可能會略微增加硬體的複雜度，但可以有效地縮短臨界路徑。因為如果使用組合電路來處理訊號判斷和計算，訊號需要經過較長的路徑；而採用觸發器則相當於將運算分攤到兩個時鐘週期，減少了每個週期內的運算量。

透過實際的合成和測試，我學到了許多在功能模擬層面無法發現的問題，這讓我對實際電路設計的挑戰有了更深入的了解。我相信，這些寶貴的經驗將成為我未來在業界發展的基石。

**學習新工具，拓展視野**

除此之外，這次作業還讓我接觸並學習了許多設計相關的工具，加深了我對驗證流程的理解。使用像 Superlint 這樣的工具，可以在合成之前預先發現可能的問題，提前進行修正，還能改善我的程式碼風格，降低未來出錯的機會。使用 Design Compiler 進行合成，不僅可以生成實際的電路圖（Schematic），還能透過 TCL 指令瞭解合成過程中的細節，讓我能夠找到臨界路徑和佔用面積較大的部分，無論是組合電路還是非組合電路，方便我進行電路的優化。使用 Verdi 工具，可以查看 nWave 和電路架構圖，對於除錯和性能分析都非常有幫助。

**結語**

總的來說，我非常慶幸能夠修讀這門課程，從第一個作業開始，就讓我學到了許多寶貴的知識和技能，為我打下了堅實的基礎。我深刻體會到充分規劃和學習的重要性，也相信這些經驗將讓我在未來的學習和工作中，不斷地進步和提升。期待能夠將所學應用於實際，為公司的發展貢獻一份力量。