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: 陳俐蓉

Student ID: N26120113

Outline

[一、 Summary 3](#_Toc147236184)

[二、 Block diagram 3](#_Toc147236185)

[三、 Verification waveform 6](#_Toc147236186)

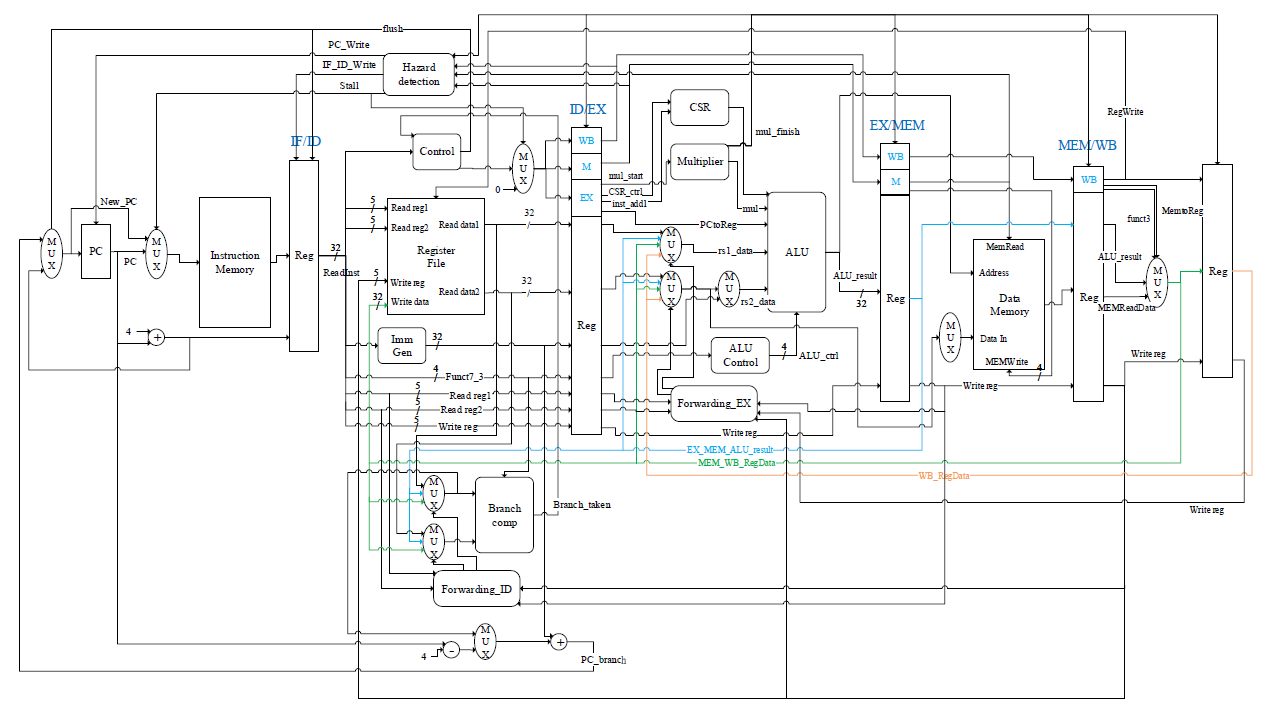
[四、 Synthesize result 11](#_Toc147236187)

[五、 Superlint coverage 13](#_Toc147236188)

[六、 Problems encountered and lessons learned 14](#_Toc147236189)

1. Summary

實現RISC-V ISA 5-stage pipelined CPU，可執行的instruction包含R-type(含乘法)、I-type、S-type、B-type、U-type、J-type、CSR。

1. Block diagram

* Control : 產生各stage所需的控制訊號。其中包含flush訊號，當Branch在ID stage成立時，flush訊號會拉起，使已送進IF stage的指令flush成空指令。
* Hazard detection : 判斷是否需要產生Stall，分為4種狀況

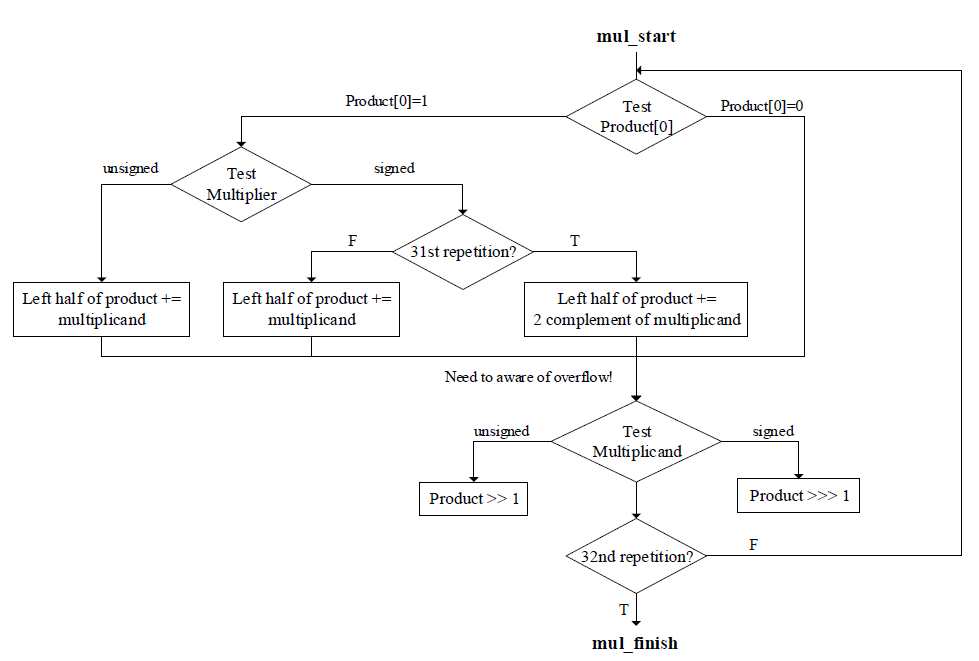
1. Load下個cycle指令的source register為Load的destination register。
2. Load下兩個cycle指令為B type / JALR且source register為Load的destination register。
3. B type / JALR上個cycle指令的destination register為B type / JALR的source register。
4. 此指令為乘法，進行到EX stage時需Stall 32個cycle。

* Imm Gen : 幫immediate做sign extension成32 bits。
* Branch comp : 幫B type指令做比大小，並判斷此Branch是否成立
* Forwarding\_ID : 在B type指令進行到ID stage時，其source register為前面指令的destination register，且此時destination register的值尚未被寫回，發生data hazard，需進行forwarding。需注意的是，若有一種以上的forwarding狀況，最靠近ID stage的資料應該最優先被forwarding，因為最靠近ID stage的資料才是最新的資料。

|  |  |
| --- | --- |
| Forward1 / Forward2 | rs1\_data / rs2\_data |
| 0  (No forwarding happen) | RS1\_data / RS2\_data |
| 1 | EX\_MEM\_ALU\_result |
| 3 | MEM\_WB\_RegData |

* ALU Control : 產生ALU控制訊號(4 bits)，告訴ALU此指令需要做哪種運算。
* ALU : 負責做加、減、AND、OR、XOR、移位、比大小等運算。
* Multiplier : 當乘法指令進行到EX stage，mul\_start會拉起來通知ALU開始做乘法，乘法會拆成32個cycle做，同時Stall住其他stage的指令，做完32個cycle後mul\_finish會拉起來，並解除Stall。

以下為乘法運算流程圖



* CSR : 負責計算cycle數和instruction數，cycle數每經過一個cycle加一，instruction數在Stall或Flush成空指令時不可加一。
* Forwarding\_EX : 指令進行到EX stage時，其source register為前面指令的destination register，且此時destination register的值尚未被寫回，發生data hazard，需進行forwarding。需注意的是，若有一種以上的forwarding狀況，最靠近EX stage的資料應該最優先被forwarding，因為最靠近EX stage的資料才是最新的資料。

|  |  |
| --- | --- |
| Forward1 / Forward2 | rs1\_data / rs2\_data |
| 0  (No forwarding happen) | ID\_EX\_RS1\_data / ID\_EX\_RS2\_data |
| 2 | EX\_MEM\_ALU\_result |
| 1 | MEM\_WB\_RegData |
| 3 | WB\_RegData |

1. Verification waveform
2. R type -- ADD

128: li t0, -1 // t0 = -1

12C: li t1, -1 // t1 = -1

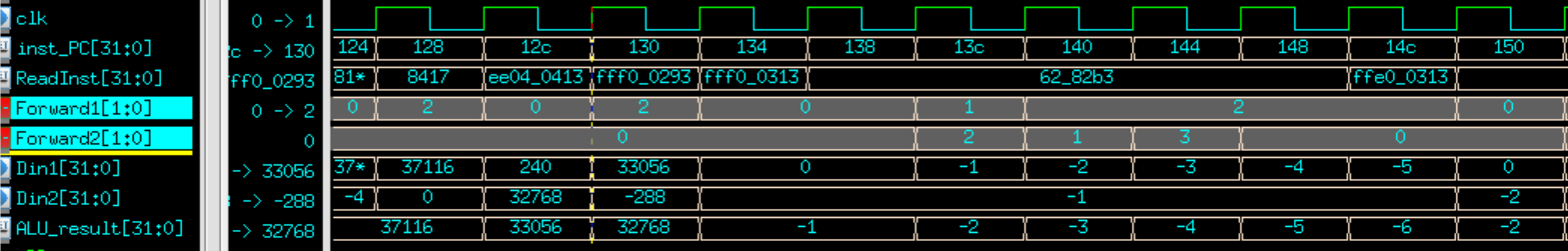
130: add t0, t0, t1 // t0 = -2

134: add t0, t0, t1 // t0 = -3

138: add t0, t0, t1 // t0 = -4

13C: add t0, t0, t1 // t0 = -5

140: add t0, t0, t1 // t0 = -6



框起來的部分為擷取指令在EX stage的運算結果，可以看到指令之間有發生data hazard，並進行forwarding。

1. R type -- MUL

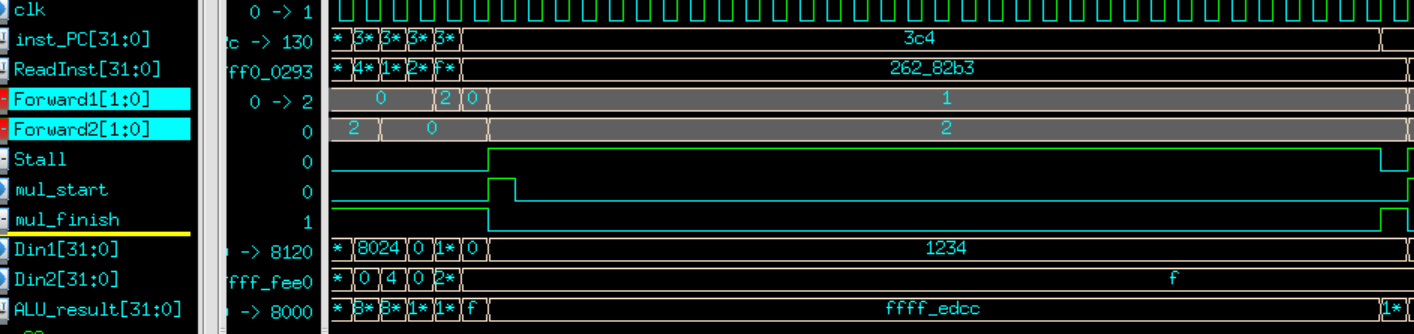
3b0: lui t0, 0x1

3b4: addi t0, t0, 564 // t0 = 1234h

3b8: li t1, 15 // t1 = fh

3bc: mul t0, t0, t1 // t0 = 1110Ch

3c0: mul t0, t0, t1 // t0 = FFFB4h



紅框部分為乘法指令進行到EX stage時需Stall住其他stage，需Stall 32 個cycle。

1. I type -- LW

494: addi t0, sp, 16

498: lw t0, 0(t0)

Stall

49c: lw t0, 0(t0)

Stall

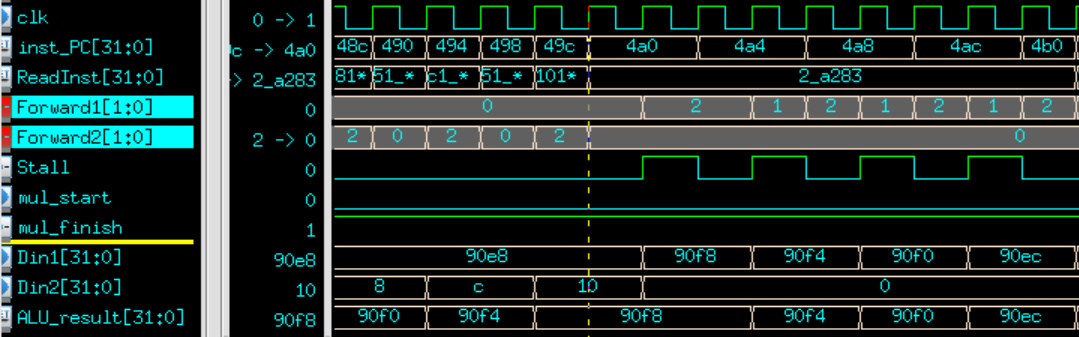
4a0: lw t0, 0(t0)

Stall

4a4: lw t0, 0(t0)

Stall

4a8: lw t0, 0(t0)



可以知道這裡Stall的條件為上述的第一個條件，在擷取指令中共需Stall 4次。

1. S type -- SW、SB、SH

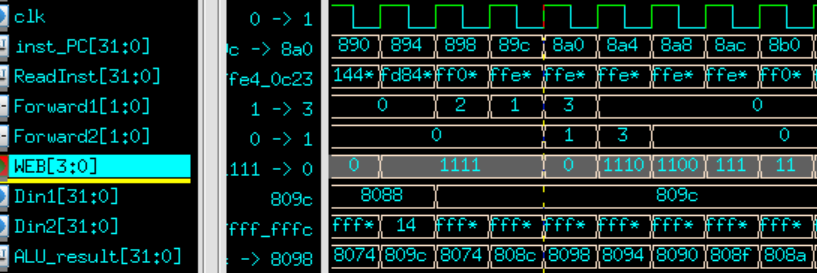
894: sw t5, -4(s0)

898: sb t5, -8(s0)

89c: sh t5, -12(s0)

8a0: sb t5, -13(s0)

8a4: sh t5, -18(s0)



藉由控制WEB的4個bit來決定要寫入記憶體中的哪個位置，WEB的1個bit可以控制記憶體的1個byte是否接受寫入。

1. B type -- BEQ

≪Branch不成立≫

8bc: lui t0, 0xfffff

8c0: lui t1, 0xfffff

8c4: lui t1, 0x1

8c8: addi t1, t1, -16 // t1 = ff0

Stall

8cc: beq t0, t1, 904

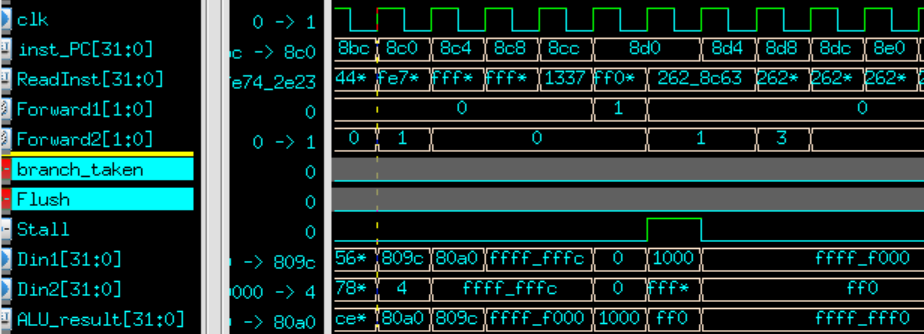
8d0: beq t0, t1, 904

Branch not taken

8d4: beq t0, t1, 904

8d8: beq t0, t1, 904

8dc: beq t0, t1, 904



可以看到此段指令的branch都沒有成立，因此在ID stage時branch\_taken和Flush都沒有拉起來，PC也沒有跳轉為branch成立後的PC。而此處有發生Stall是因為上述狀況中的第三個條件。

≪Branch成立≫

Branch taken

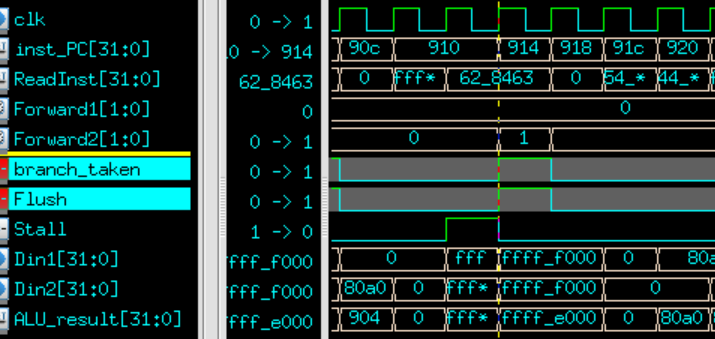
90c: beq t0, t1, 914

Flush

910: ori t0, t0, 2

914: sw t0, 0(s0)

918: addi s0, s0, 4



Instruction

machine code

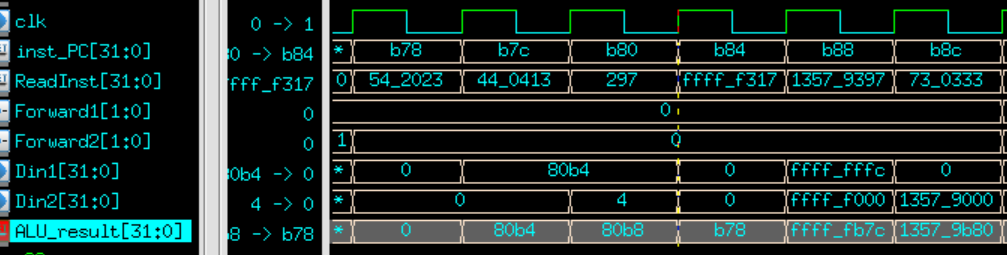
這裡可以看到PC為90c的beq成立後，在ID stage時branch\_taken和Flush都拉起來，接著下個cycle的指令被Flush為空指令。

1. U type -- AUIPC

b78: auipc t0, 0x0 // t0 = 0xb78 + 0x0 = b78

b7c: auipc t1, 0xfffff // t1 = 0xb7c + fffff000 = fffffb7c

b80: auipc t2, 0x13579 // t2 = 0xb80 + 13579000 = 13579b80



紅框部分為擷取指令進行到EX stage時ALU的運算結果。

1. J type -- JAL

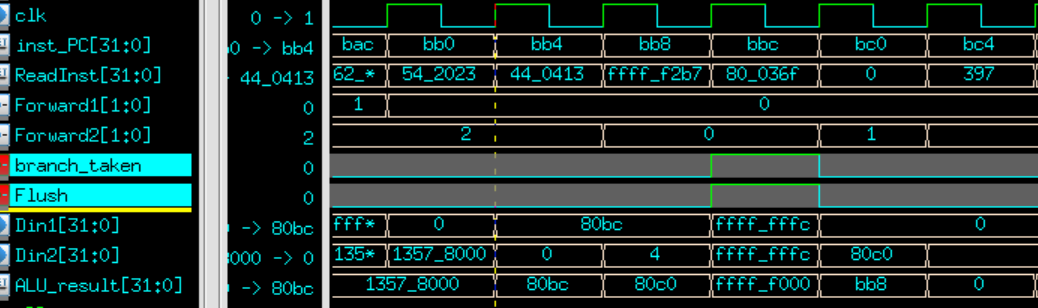
bb0: lui t0, 0xfffff

bb4: jal t1, bbc

Flush

bb8: ori t0, t0, 1

bbc: auipc t2, 0x0



JAL屬於unconditional branch，一定會成立，所以在JAL進行到ID stage時branch\_taken和Flush都會拉起來。

1. CSR -- RDINSTRET

bd4: rdinstreth t0

be0 ID

bdc EX

be0 ID

Stall EX

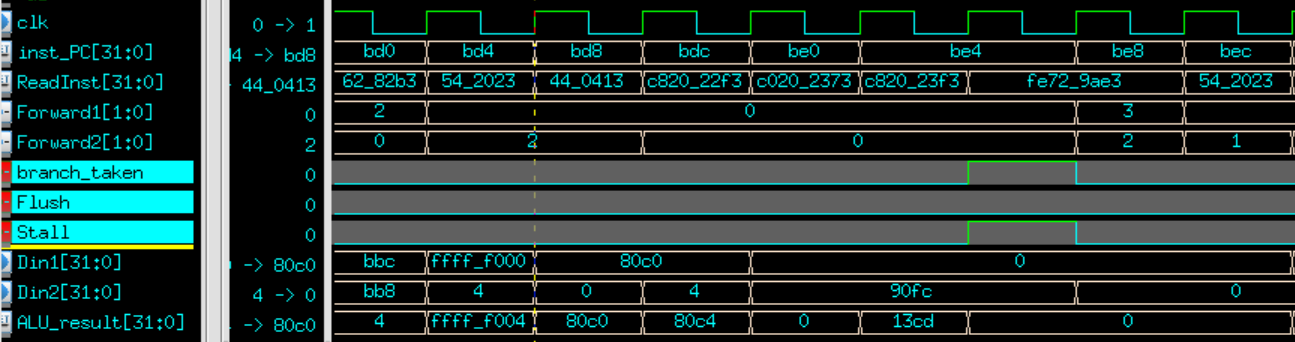
bdc MEM

bd8: rdinstret t1

bdc: rdinstreth t2

Stall

be0: bne t0, t2, bd4

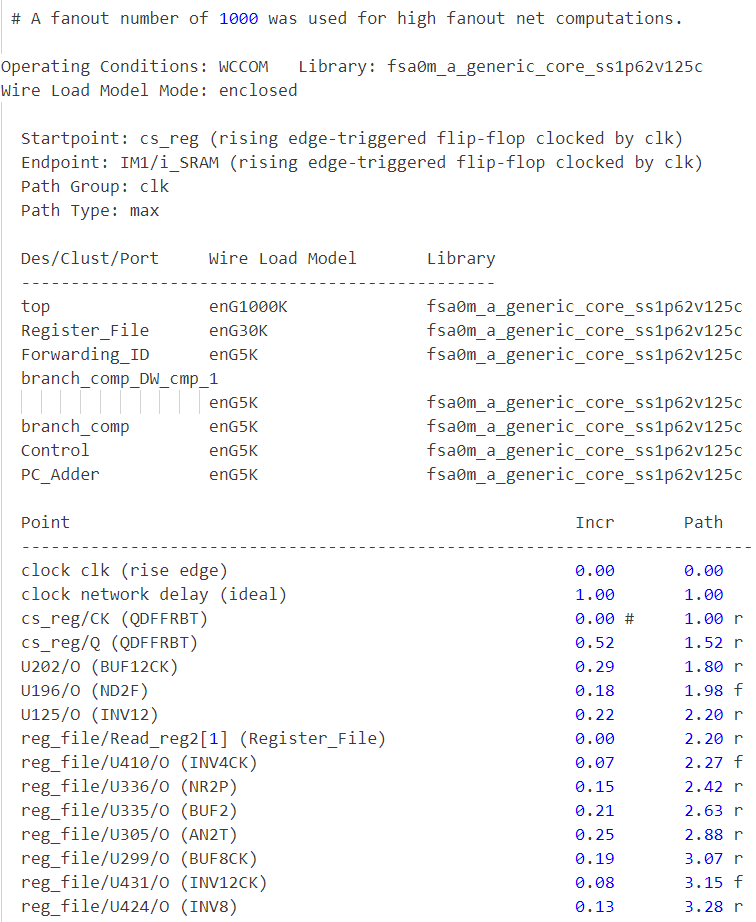


已知在be0進行到ID stage、bdc進行到EX stage時，會插入一個Stall，在此同時be0的branch會成立，但此時t2的正確數值尚未被forwarding寫回，這樣會導致指令發生錯誤，因此當Stall發生時，Flush就不可以拉起，需等待Stall過後的branch判斷才是正確的。

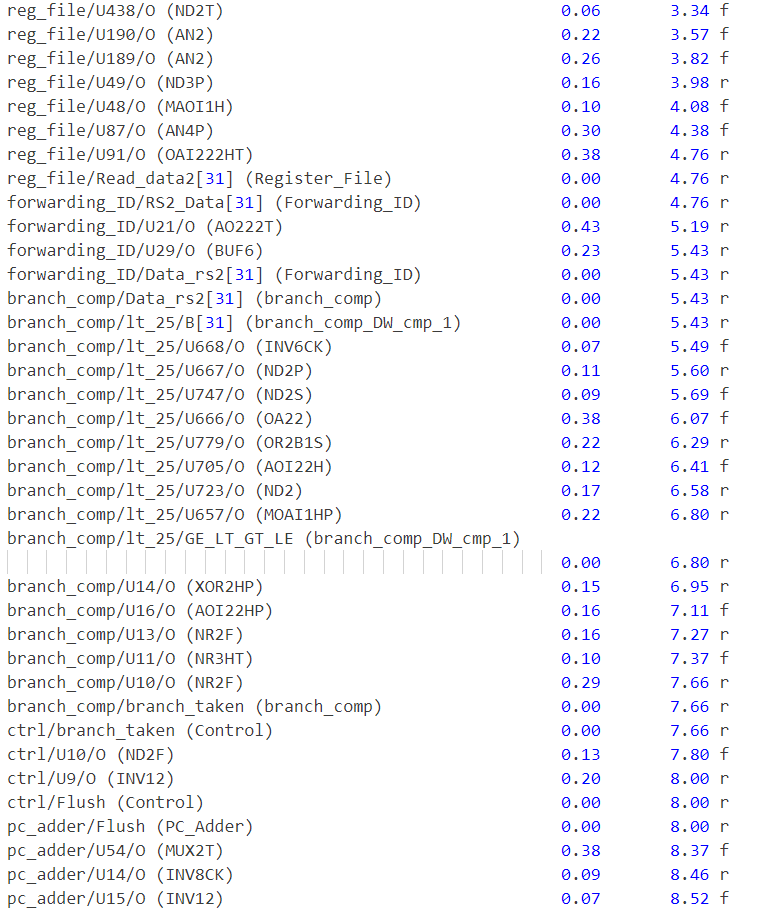
1. Synthesize result

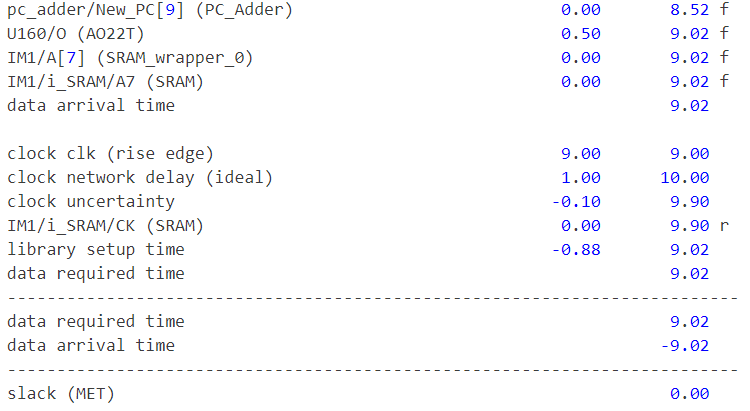
* Cycle time : 9 ns

以prog0來說，總執行時長 = 9ns \* 6773cycles = 60957ns

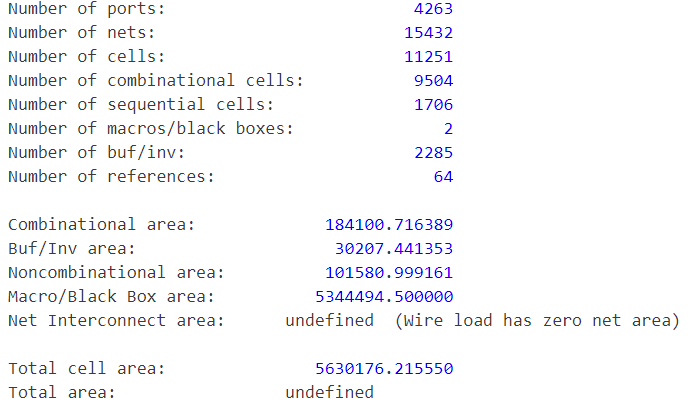


(接下頁)



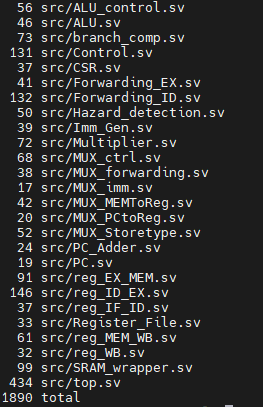


* Area : 5630176 um\*um

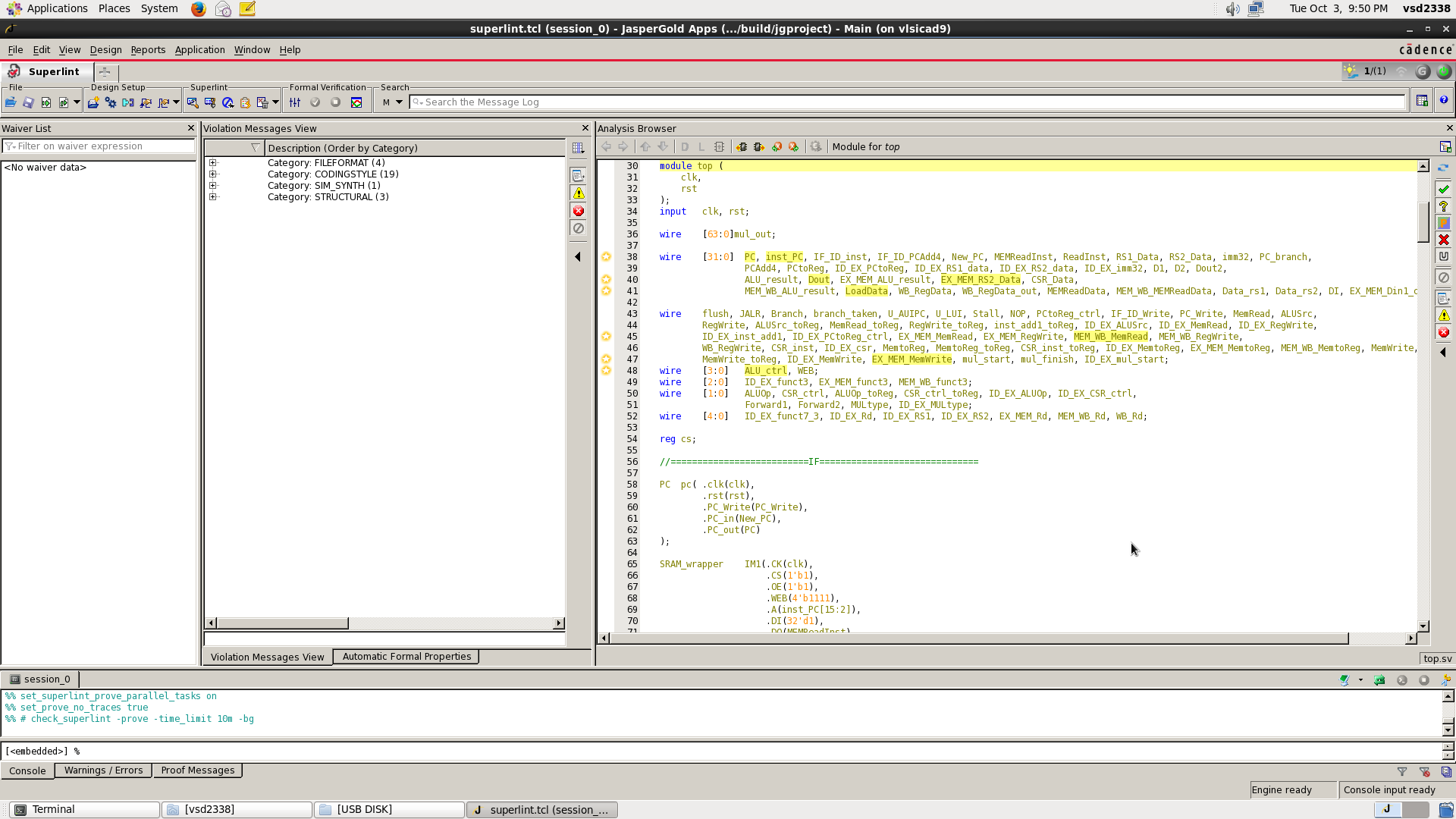


1. Superlint coverage

* Number of total lines = 1890



* Superlint warning lines = 27



* Superlint coverage = (1-27/1890)\*100% = 98.6%
* 跑superlint最常出現的warning

1. 同一個檔案有兩個以上的module
2. 檔案名稱和module名稱不同
3. 常數沒有標出有幾bits
4. Problems encountered and lessons learned

這次實作讓我更了解RISC-V的指令，因為之前學得計組版本是MIPS的，剛開始在理解RISC-V的指令有花了一點時間，不過電路架構大致上是差不多的，但實作的時候會發現計組裡學的架構只是大概，有些細節之前學的時候其實沒有畫上去，像是:

1. 如果把branch指令放在ID stage的話，在ID stage也要做forwarding。
2. 就算前面的指令已經進行到WB stage，後面的指令做到EX stage時還是可能會發生data hazard，因為後面指令在ID stage讀出來的數值依然是還沒被寫回更新的數值，因此在EX stage需要forwarding的條件其實有4個。

另外，合成後有針對critical path和面積做以下優化:

1. 這裡的記憶體有加delay，如果記憶體讀取的資料沒有送進pipeline的暫存器，而是直接送到下個stage，在合成的時候會拖慢cycle的時間，導致很難合到10ns以內。但因為記憶體讀取資料的時間會比送進位址的時間慢一個cycle，如果讀取的資料又要送進pipeline，那麼送進位址的時間就要比原本的架構再提早一個cycle，才能避免讀取資料的時間慢一個cycle。
2. 加Stall的第三個條件並不是因為無法做forwarding，如果要在這裡做forwarding需要直接從EX stage的ALU result拉回ID stage，這樣會導致critical path變長，為了加快合成的cycle time，決定多Stall一次取pipeline送出的數值。
3. 由於乘法器為位於critical path上，而且面積很大，所以我有針對乘法器做pipeline，在EX stage把乘法指令拆成32個cycle做，並且讓4個乘法指令共用一個乘法器，這樣就只會用到一個32 bits的加法器和移位器，面積縮小很多，也能縮短critical path，cycle time可以壓到9ns，但缺點是這樣會讓cycle數變多，如果做很多乘法指令的話時間反而可能比做pipeline前更慢。