VLSI System Design (Graduate Level)

Fall 2020

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: 黃建智

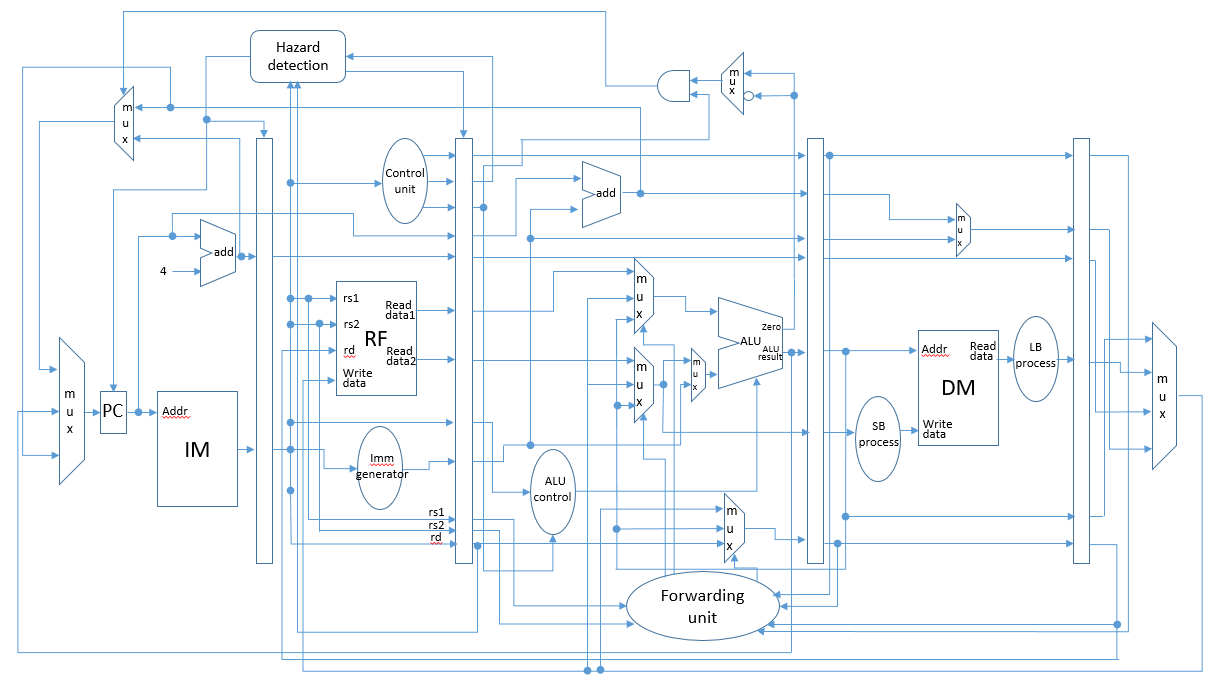
Student ID: N26080020

Summary: 實現五級pipeline RISC-V 32I的架構，33種功能。

|  |  |
| --- | --- |
| rtl0 | pass |
| rtl 1 | pass |
| rtl2 | pass |
| rtl3 | pass |
| syn0 | pass |
| syn1 | pass |
| syn2 | pass |
| syn3 | pass |
| superlint | 99.9% |

**Block diagram :**

下圖為此次作業CPU流程圖，這邊省略部分控制訊號線避免圖片過於雜亂，下面會依序對每個stage做較詳細的解說。



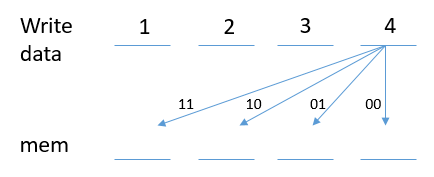
由於此次作業不只有單純的jump與branch還需考慮JALR因此下一個PC除了PC+4, PC+imm還需要額外加入rs1+imm的選項，並用jump,branch以及JALR控制訊號選擇。當PC進入IM下個cycle即會讀出對應的32-bit指令。

第一級pipeline後，將指令送往decode stage，在此階段，會將指令透過risc-v的格式，拆為rs1,rs2,rd的address以及要執行的功能與imm，因此RF就是在處理rs1,rs2與rd，將rs1與rs2中的值讀出，並將write back回來的data寫入rd位置，至於功能的部分，Control unit會透過opcode,funct7,funct3 來決定目前要執行的功能為哪一個，並決定每一條控制線應該是多少。由於不同type所用到的imm格式都不盡相同，Imm generator則是用來處理imm的格式問題，並輸出正確的imm。

經過第二級pipeline後，資料則來到了execution stage，主要的運算都在此stage完成，ALU的操作主要由ALU Control的控制線來決定，而不是直接由第二級的Control unit來決定，這種二階段的控制能簡化控制線的複雜度。此外，由於五級pipeline勢必會有pipeline hazard的問題產生，因此在此級加入了Forwarding unit透過forward的方式來解決pipeline hazard的問題，不過LW的hazard沒有辦法使用這種forward的方式完全解決，透過”Hazard detection”先偵測LW hazard發生的時機，並同時發出stall的訊號，將前兩級的pipeline以及PC stall住一個cycle，下個clk就能正常用forward的方式解決hazard的問題。

Branch決定要不要跳的判斷也是在此級完成，若branch的條件成立，會有一控制訊號拉到前面的Program Counter決定下一個PC為PC+imm而不是PC+4，若在第三級判斷branch成立，則前兩級讀進來的指令皆為不須執行的指令，因此會透過flush的方式將前兩級的訊號flush掉，達到branch prediction的效果，就不用每次有branch指令就先停兩個cycle。JAL即JALR也是透過flush的方式將前兩級的訊號flush掉。

在MEM stage，主要功能為將ALU算出來的結果存入DM或是將DM中的值讀出來放到rd中，由於此次作業需特別處理LB以及SB的功能，因此，若須執行SB，在write data寫入DM前須先處理，根據SB指令的imm我們可以得知我們要將write data中的least significant byte放到對應mem中的哪個位置，如下圖所示:



LB功能則是在DM讀出值後處理，只留下least significant byte，並sign extension到32bit 。

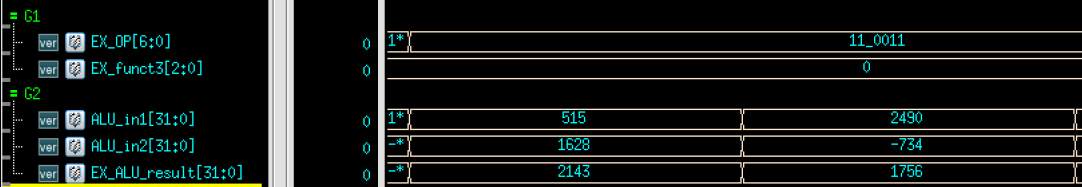
最後一級則是將data 寫回register，共會有四種情況，ALU算完的結果、從DM load出來的值、PC+4以及PC+imm，因此最後一級我用了一個4 to 1的mux來選擇要write back回去的資料。

**Prog0: Verification of the functionality of instructions**

* R-type

1. 測試ADD功能: opcode=0110011, funct3=000

ALU的input分別為515及1628，加完結果為2143

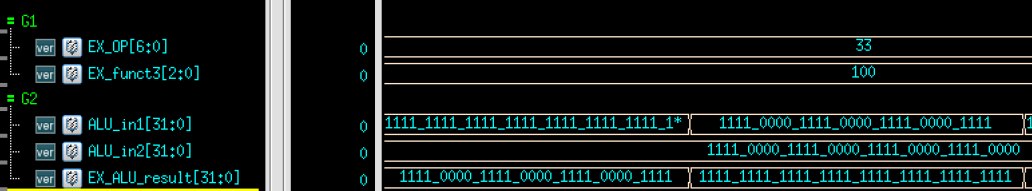


1. 測試XOR功能: opcode=0110011, funct3=100

ALU\_in1(rs1)=32’b0000\_1111\_0000\_1111\_0000\_1111\_0000\_1111

ALU\_in2(rs2)=32’b1111\_0000\_1111\_0000\_1111\_0000\_1111\_0000

ALU\_result =32’b1111\_1111\_1111\_1111\_1111\_1111\_1111\_1111



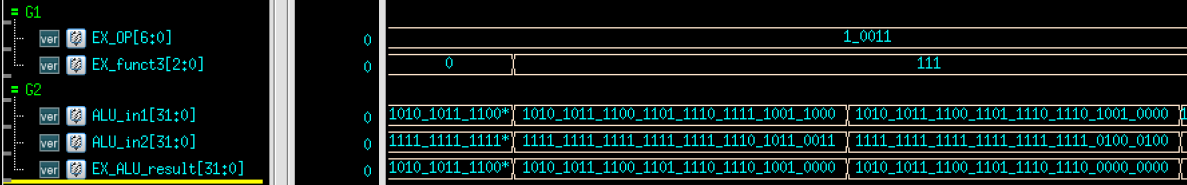
* I-type

1. 測試ANDI功能: opcode=0010011, funct3=111

ALU\_in1 (rs1) =32’b1010\_1011\_1100\_1101\_1110\_1111\_1001\_1000

ALU\_in2(imm)=32’b1111\_1111\_1111\_1111\_1111\_1110\_1011\_0111

ALU\_result =32’b1010\_1011\_1100\_1101\_1110\_1110\_1001\_0000

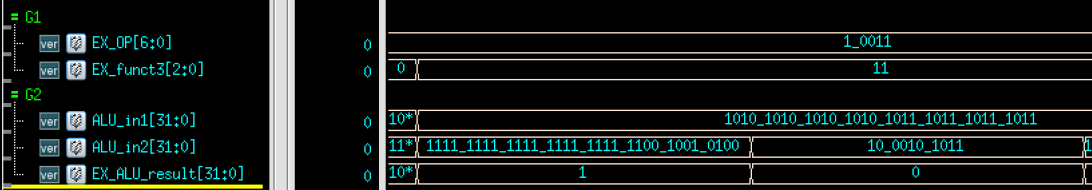


1. 測試SLTIU功能: opcode=0010011, funct3=011

ALU\_in1 (rs1) =32’b1010\_1010\_1010\_1010\_1011\_1011\_1011\_1011

ALU\_in2(imm)=32’b1111\_1111\_1111\_1111\_1111\_1100\_1001\_0100

結果為1，因<



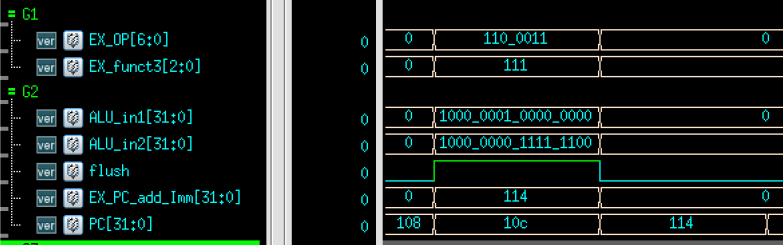
* B-type

1. 測試BGEU功能: opcode=1100011, funct3=111

ALU\_in(rs1)=32’b0000\_0000\_0000\_0000\_1000\_0001\_0000\_0000

ALU\_in(rs2)=32’b0000\_0000\_0000\_0000\_1000\_0000\_1111\_1100

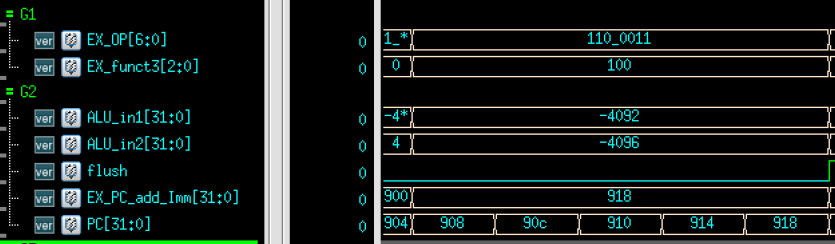
由於()成立，因此下個PC為PC+Imm=32’h114，此時flush訊號拉起，將前兩級訊號flush掉。



1. 測試BLT功能: opcode=1100011, funct3=100

ALU\_in1=-4092, ALU\_in2=-4096

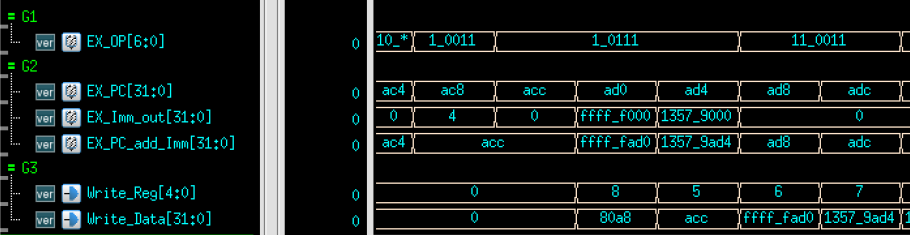
由於()並**不**成立，因此下個PC維持PC+4，並不會branch。



* U-type

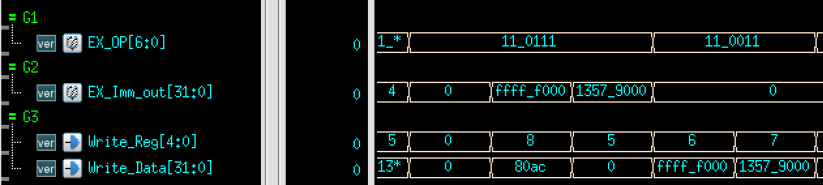
1. 測試AUIPC功能: opcode=0010111

PC+Imm的值在2個cycle之後會write back到rd中



1. 測試LUI功能: opcode=0110111

Imm的值在兩個cycle之後會存進rd如下圖所示



* J-type

測試 JAL功能: opcode=1101111

下個PC為PC+imm即 (b08+8=b10)而PC+4 即(b08+4=b0c)

則會在兩個cycle後存進rd

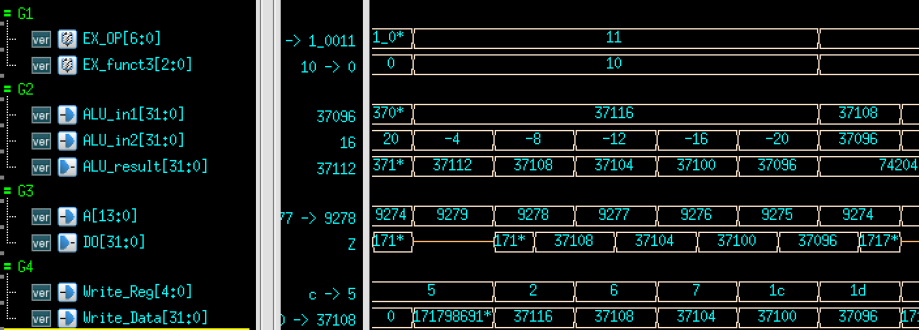


+4

* LW/LB

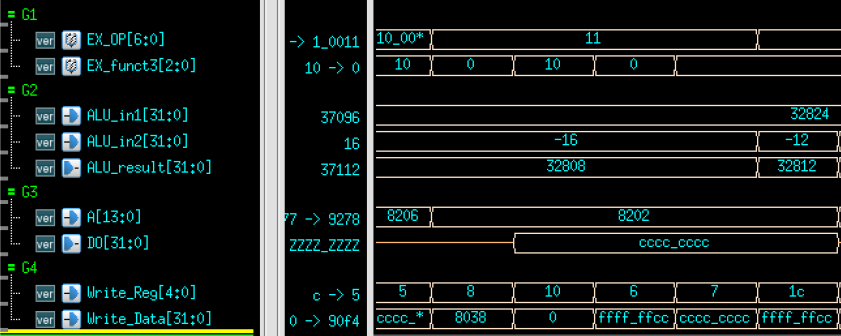
1. 測試LW功能:opcode=0000011 funct3=010

ALU\_in1(rs1)與ALU\_in2(imm)相加後得到ALU\_result，下個cycle會在Data memory讀出M[ALU\_result/4]中的值如下圖所示，再下一個cycle則會將此值寫回到register



1. 測試LB功能:opcode=0000011 funct3=000

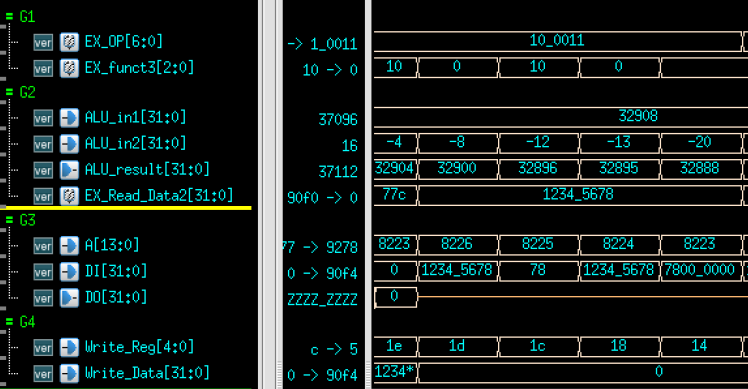
ALU\_in1(rs1)與ALU\_in2(imm)相加後得到ALU\_result，下個cycle會在Data memory讀出M[ALU\_result/4]中的值如下圖所示，LB會將讀出的值留下最後一個byte，其他部分做sign extension到32bit，以下圖為例，在M[8202]中讀出的值為0xcccc\_cccc因為LB存回register的值為0xffff\_ffcc。



* SW/SB

1. 測試SW功能:opcode=0100011 funct3=010

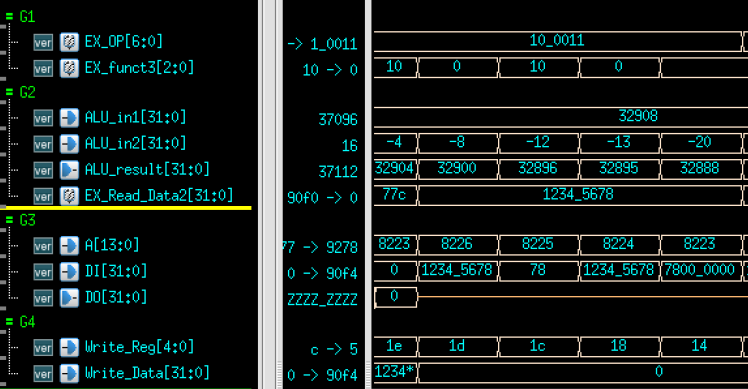
ALU\_in1(rs1)與ALU\_in2(imm)相加後得到ALU\_result，下個cycle會將EX\_Read\_Data2(rs2)的值在下一個cycle寫入M[ALU\_result/4]中如下圖所示。



1. 測試SB功能:opcode=0100011 funct3=000

ALU\_in1(rs1)與ALU\_in2(imm)相加後得到ALU\_result，下個cycle會將EX\_Read\_Data2(rs2)的值在下一個cycle寫入M[ALU\_result/4]中如下圖所示。

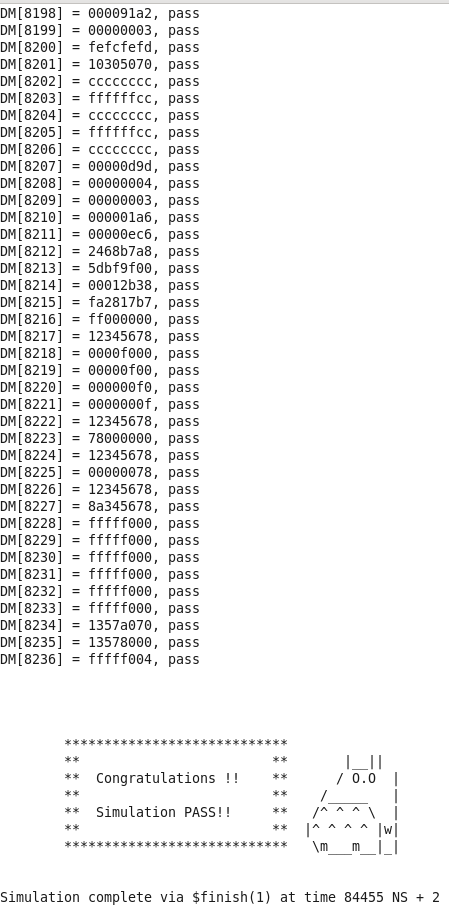
SB將會根據ALU\_result之最後兩個bit來決定要將最後一個byte寫入DM的哪個位置，以下圖為例，當rs2為0x1234\_5678，當ALU\_result之最後兩個bit為00，則78存到該mem的最後一個byte，而當ALU\_result之最後兩個bit為11時，78則存到mem中的第一個byte如下圖所示。



00

11

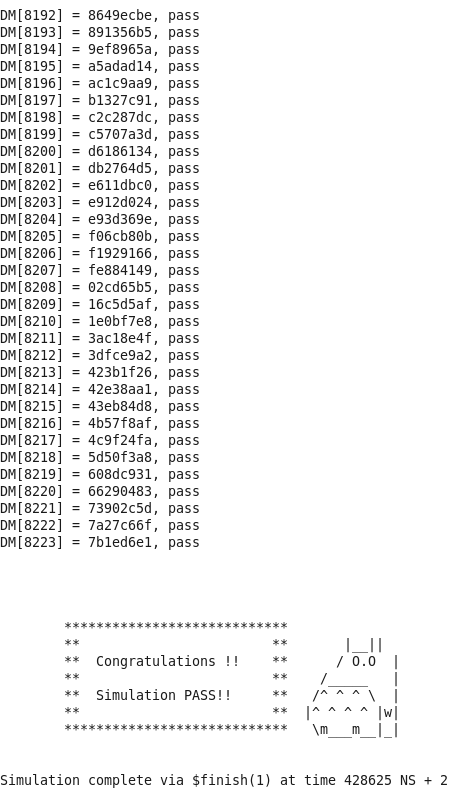
Prog0 測試結果:



Prog1: Sort Algorithm

利用簡單的bubble sort來完成，比較第k個值與第k+1個值的大小(k從0到 array\_size)，若大於則交換，小於則不變，重複此步驟值到sort結束後存進相對應的\_test\_start位置中。

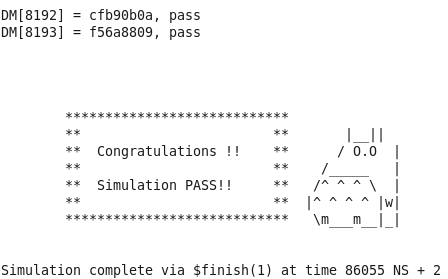
測試結果如下:



Prog2: Multiplication

由於32-bit的兩數相乘結果會是64-bit，因此相乘後需分成兩部分存入32-bit的mem，利用位移(>>)之方式即可達成。

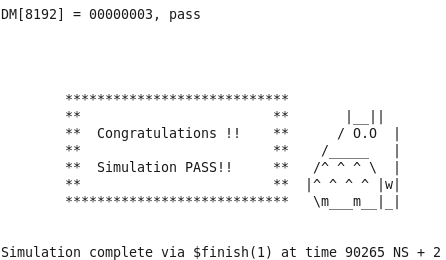
測試結果如下:



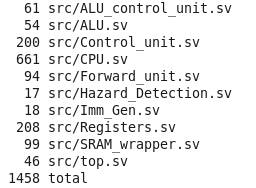
Prog3: GCD

透過輾轉相除法的方式達成，不斷透過兩數相除(大數除以小數)所得的餘數，作為下次除法的input，遞迴執行當最後除到餘數為0時，此時除數即為最大公因數。

測試結果如下:



**Number of lines of RTL code: 1458**

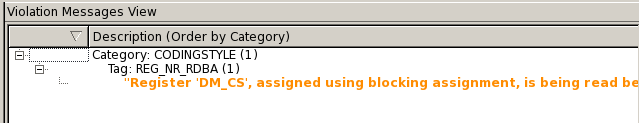


**Superlint result:**

比較常見的warning:

1. case如果條件有寫滿，是可以不用寫default的。🡪把default 拿掉

2.沒有寫1’b0或1’b1而是直接寫0或1。 🡪 數字前記得要加1’b 或是1’d

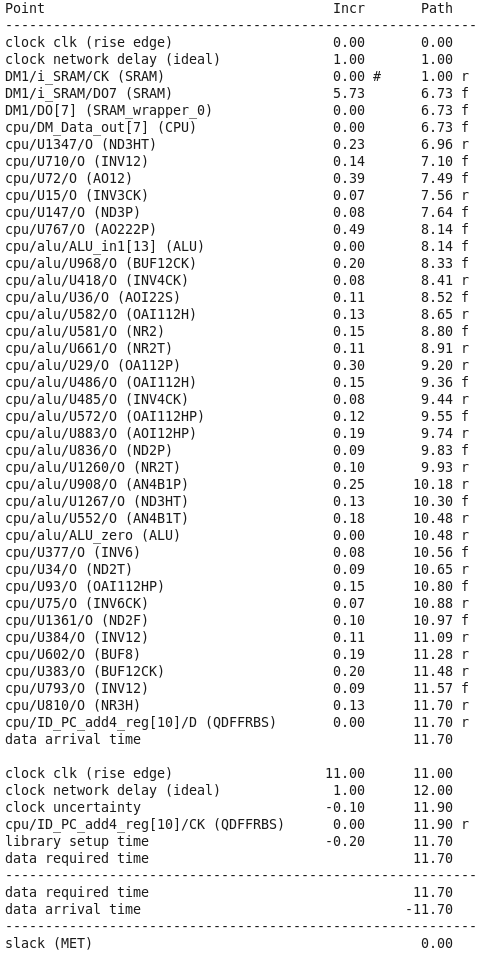


Percentage: 1 - =99.93%

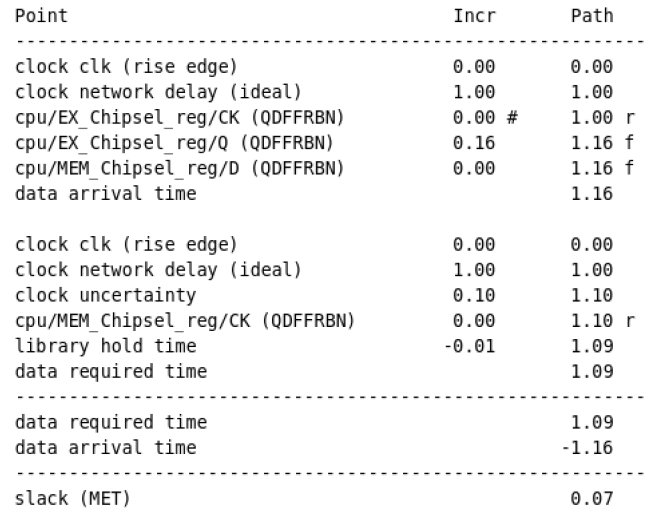
**Synthesis:**

clk 🡪 11

Set up time report:



Hold time report:



Describe the major problems you encountered and your resolutions:

此次作業主要為實現一顆RISC-V 32I的架構，有33種功能，並達到五級pipeline，由於是pipeline架構，因此不可避免的要解決許多pipeline hazard的問題，透過forward以及stall能解決大部分問題，另一個比較大的問題則是timing的問題，若CPU為正緣觸發，Register以及其他的MEM皆採用正緣觸發，在timing上會比原來預期慢一個clk才輸出正確的值，此時forword以及一些pipeline的設計可能就需要做一些相對應的修改，而不是像大學所學可以在上半clk讀值，下半個clk寫值，因此在這部分的改良花了蠻多的時間。此外，合成時遇到的問題則是，第一個clk 正緣會吃到還沒準備好的資料(unknown)，導致後續IM沒有讀出正確的指令，因此我在程式中，讓IM的addr晚一個clk等資料準備好之後再進入IM，就能解決上述問題。

Lesson learned:

由於平常比較少做到這種比較大的作業，有許多細節若沒仔細考慮，都有可能造成最後結果失敗，完成這次作業後，學到最多的是對RISC-V指令級架構的了解，以及如何處理Control訊號，還有debug的能力，因為功能非常多，測試的pattern也是非常長，要如何從茫茫大海中迅速找到錯誤的地方，是一件蠻有挑戰的事情。