國立臺灣科技大學
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# FPGA 系統設計實務

（FPGA System Design）

## Final Report

執導教授:林銘波

學生:M11302155 鄭立揚

# 目錄

**How to design an RISC processor** ..... **3**

A 16-Bit Eight-Register Register File ..... 4

A 16-Bit ALU ..... 6

A 256x16 Memory Module ..... 8

PC Circuitry ..... 8

Complete Datapath ..... 9

Controller ..... 13

**Implement schematic entry** ..... **20**

3-to-8 decoder ..... 20

16 bit 8-to-1 mux ..... 23

16 bit DFF ..... 29

16 bit Eight-Register Register File ..... 32

Full adder ..... 35

16 bit ALU ..... 37

256x16 memory ..... 41

Complete Data path ..... 45

**Test** ..... **68**

Find the minimum and maximum from two numbers in memory. ..... 68

Add two numbers in memory and store the result in another memory location. ..... 69

Add ten numbers in consecutive memory locations. ..... 70

Mov a memory block of N words from one place to another. ..... 71

**Hardware cost** ..... **73**

**Conclusion** ..... **75**

# How to design an RISC processor

根據期中所做的 schematic，我們進一步將其轉化為硬體描述語言的方式來寫，細節上大部分的狀況都差不多，要先將指令 bit 數決定好之後，再來決定每一個元件的輸入、輸出及 enable 信號，組成 datapath 後，設計好 controller，然後就可以將兩個部分合起來變成最後的 multi-cycle processor。

選定的語言為 verilog HDL，使用的參數與期中一致，根據作業 pdf 的步驟一步一步將各個元件描述出來，並且此次的期末報告有將期中未達到的細節補上，參數定義更清楚，controller 的完善。

Table 1: The basic instruction set of the 16-bit RISC processor.

| Mnemonic | Operation | N | Z | V | C | Instruction format |
|---|---|---|---|---|---|---|
| LHI Rd, #imm8 | Rd ← {imm8, Rd[7:0]}; (imm8 = 0 to 255) | - | - | - | - | 00001_ddd_iiiiiiii |
| LLI Rd, #imm8 | Rd ← {8'h0, imm8}; (imm8 = 0 to 255) | - | - | - | - | 00010_ddd_iiiiiiii |
| LDR Rd, [Rm, #imm5] | Rd ← Mem[Rm + imm5]; | - | - | - | - | 00011_dddmmm_iiiii |
| LDR Rd, [Rm, Rn] | Rd ← Mem[Rm + Rn]; | - | - | - | - | 00100_dddmmmnnn_00 |
| STR Rd, [Rm, #imm5] | Mem[Rm + imm5] ← Rd; | - | - | - | - | 00101_dddmmm_iiiii |
| STR Rd, [Rm, Rn] | Mem[Rm + Rn] ← Rd; | - | - | - | - | 00110_dddmmmnnn_00 |
| ADD Rd, Rm, Rn | Rd ← Rm + Rn; | * | * | * | * | 00000_dddmmmnnn_00 |
| ADC Rd, Rm, Rn | Rd ← Rm + Rn + C; | * | * | * | * | 00000_dddmmmnnn_01 |
| SUB Rd, Rm, Rn | Rd ← Rm - Rn; | * | * | * | * | 00000_dddmmmnnn_10 |
| SBB Rd, Rm, Rn | Rd ← Rm - Rn - $\overline{C}$; | * | * | * | * | 00000_dddmmmnnn_11 |
| CMP Rm, Rn | Rm - Rn; | * | * | * | * | 00110_xxxmmmnnn_01 |
| ADDI Rd, Rm, #imm5 | Rd ← Rm + #imm5; (imm5 = 0 to 31) | * | * | * | * | 00111_dddmmm_iiiii |
| SUBI Rd, Rm, #imm5 | Rd ← Rm - #imm5; (imm5 = 0 to 31) | * | * | * | * | 01000_dddmmm_iiiii |
| MOV Rd, Rm | Rd ← Rm; | - | - | - | - | 01011_dddmmm_xxx_xx |
| BCC label | $\overline{C}$: PC ← PC + SignExtend(label); | - | - | - | - | 1100_0011_disp8 |
| BCS label | $C$: PC ← PC + SignExtend(label); | - | - | - | - | 1100_0010_disp8 |
| BNE label | $\overline{Z}$: PC ← PC + SignExtend(label); | - | - | - | - | 1100_0001_disp8 |
| BEQ label | $Z$: PC ← PC + SignExtend(label); | - | - | - | - | 1100_0000_disp8 |
| B[AL] label | PC ← PC + SignExtend(label); | - | - | - | - | 1100_1110_disp8 |
| JMP label | PC[10:0] ← label; | - | - | - | - | 10000_label11 |
| JAL Rd,label | Rd ← PC; PC ← PC + SignExtend(label); | - | - | - | - | 10001_ddd_disp8 |
| JAL Rd,Rm | Rd ← PC; PC ← Rm; | - | - | - | - | 10010_dddmmm_xxx_yy |
| JR Rd | PC ← Rd; | - | - | - | - | 10011_ddd_xxxxxxxx |
| OutR Rm | OutR ← Rm; | - | - | - | - | 11100_xxx_mmm_xxx_00 |
| HLT | Set done flag to 1 and halt CPU; | - | - | - | - | 11100_xxxx_xxxxx_01 |

Note that the memory address in each of LDR and STR instructions corresponds to a word.

與期中一樣，因為涉及到了 multi-cycle 的系統，我們需要將一個指令執行的所有階段分切成五個步驟，如下:

1.(IF)instruction fetch : read instruction memory

2.(ID/RF)instruction decode and register fetch : decode control signal and read the value from register

3.(EXE)Execution：pure ALU calculate；memory address calculate（pass by sign extension & shift & ALU）；branch or not （by ALU's zero bit）

4.(MEM)memory access or R-format : rewrite the result of ALU to register

5.(Write back, WB)load : memory read and rewrite the result of ALU to register

最後，確定好所有細節之後就可以透過硬體描述語言將我們的 processor 開始設計出來了。(以下設計與期中相同)

## 一、A16-Bit Eight-Register Register File

首先，我們要來製作 16bit 的暫存器資料夾，也就是說我可以指定 R0-R7 的暫存器當作計算上使用的一部分，接下來的步驟就是如何 step by step 的製作。

1. 製作 enabled-controlled 3-to-8 noninverting output decoder
2. 製作 8-to-1 multiplexer
3. 將 8-to-1 multiplexer 合成出一個 16 bit 的 8-to-1 multiplexer
4. 16-bit D-flip-flop register with clock-enable
5.將以上元件組合完成一個 register file

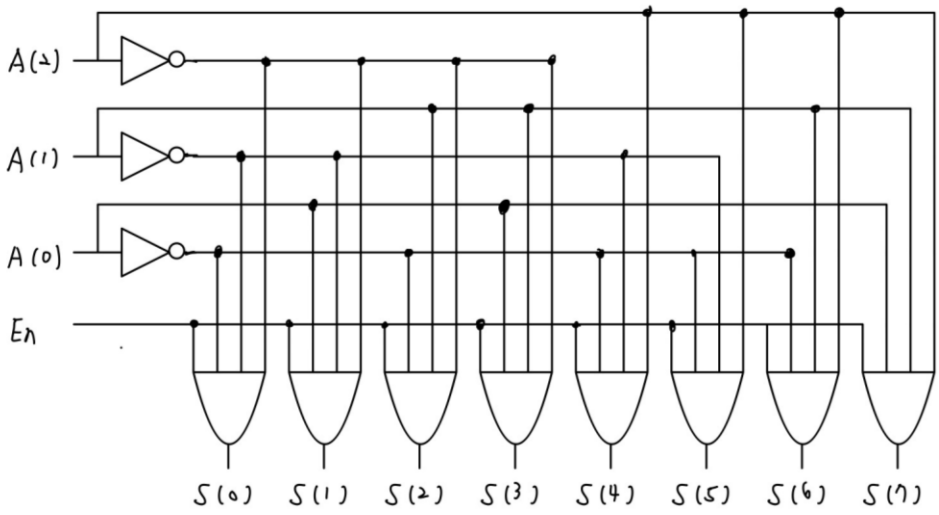
(1) enabled-controlled 3-to-8 noninverting output decoder

製作一個 decoder 我們要首先將 truth table 寫出來，用來選擇暫存器要輸入的位置。

**Truth Table**

| En | A(2) | A(1) | A(0) | S(7) | S(6) | S(5) | S(4) | S(3) | S(2) | S(1) | S(0) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

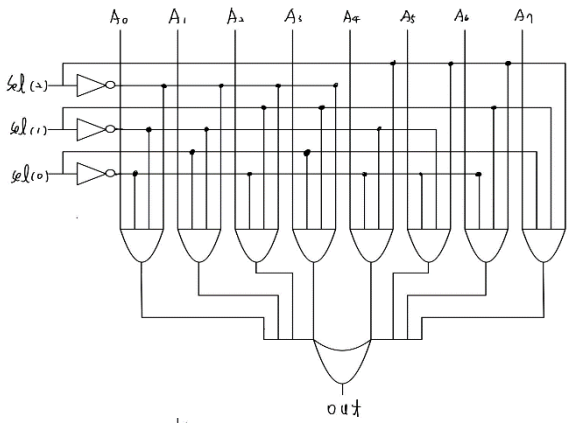
當 En 信號為 0 時輸出全為 0，所以這邊就沒有特別標示出來，可以看到當 En 為 1 時指令才能正常輸出，透過 A(2:0)來選擇現在要輸出為第幾個位置，這樣被選到的暫存器就可以正常輸出他所對應的位置。




在開始製作前先將示意圖畫出，可以看到當 A(2:0)為 000 時 S(0)為 1，以此類推下去，報告的最後會將所有實際接線圖展示。

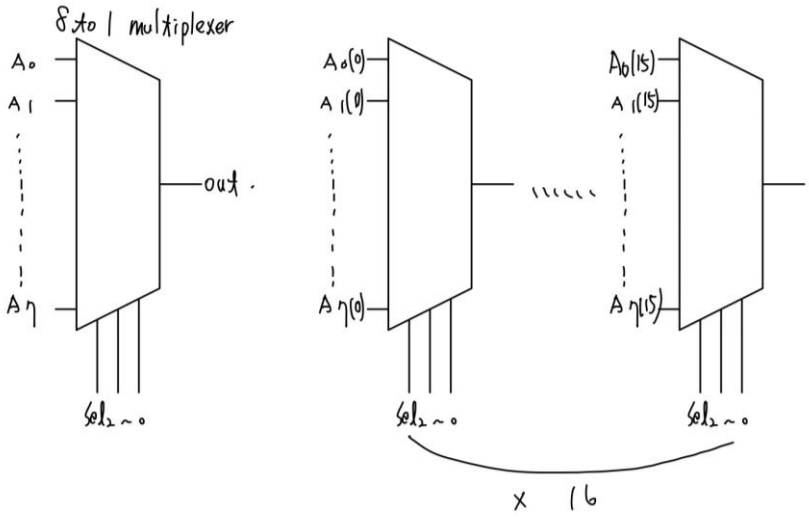
(2) 8-to-1 multiplexer&16 bit 8-to-1 multiplexer

將 3-to-8 的 decoder 稍作改動就可以做出一個 8-to-1 的 multiplexer，用來選擇現在要輸出哪一個暫存器用的，多工器透過 sel 的選擇來決定 An 要被輸出出去。




| Sel2 | Sel1 | Sel0 | out |
|---|---|---|---|
| 0 | 0 | 0 | A0 |
| 0 | 0 | 1 | A1 |
| 0 | 1 | 0 | A2 |
| 0 | 1 | 1 | A3 |
| 1 | 0 | 0 | A4 |
| 1 | 0 | 1 | A5 |
| 1 | 1 | 0 | A6 |
| 1 | 1 | 1 | A7 |

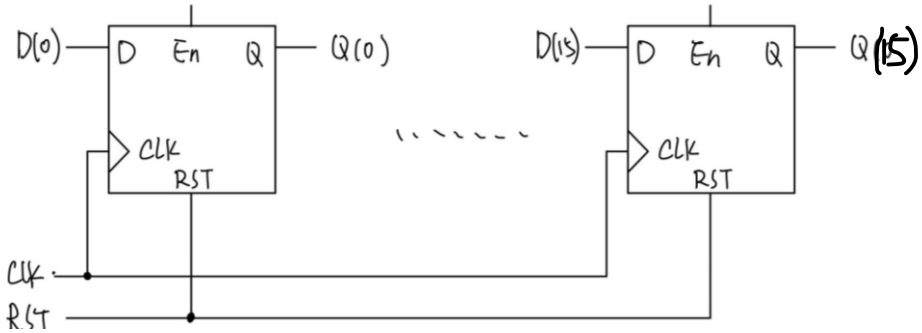
最後將 16 個 8-to-1 多工器合起來即可得到一個 16 bit 的多工器




另外我還有運用相同邏輯額外製作 16 bit 2-to-1/4-to-1 都是為了日後有一些指令(ALU source 或是 輸入信號來源等)方便使用。

(3) 16-bit D-flip-flop register with clock-enable

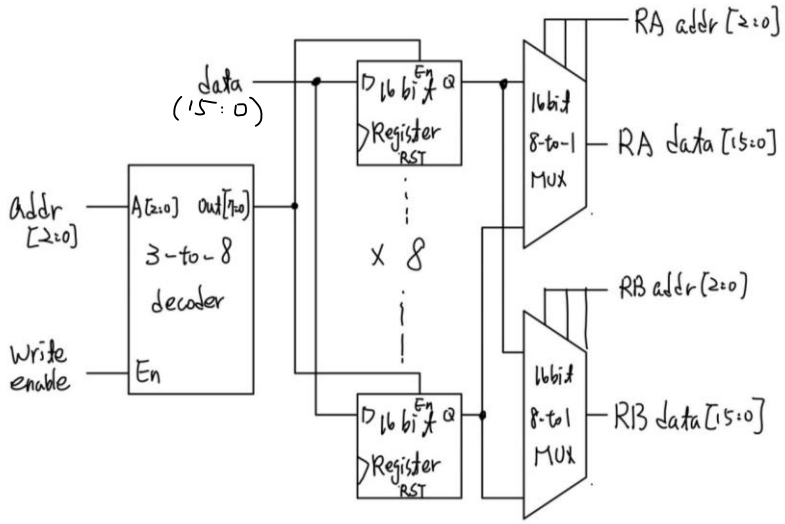
接下來要製作的是 16 bit 的 DFF，採取正緣觸發和搭配 RST，用來記錄 16bit 的資料，總共有 8 組的 DFF。



(5) combine

最後將所有元件搭配起來就完成了 register file，透過 write enable 決定是否可以寫入資料，addr 為寫入的位置，當 CLK 正緣觸發以及 Enable 即會將 data 對應的 register。

再來讀取資料時不用等 enable 或是 CLK，只要直接將 RA/RB addr 輸入，即會輸入對應的 register 資料



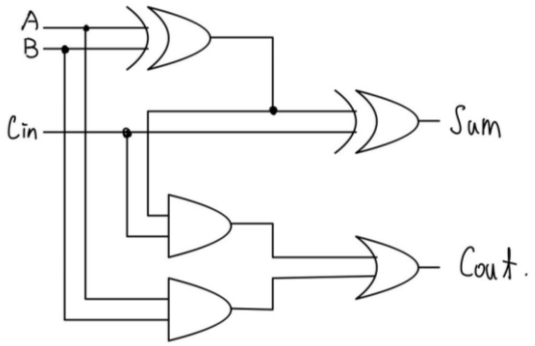

## 二、A 16-Bit ALU

ALU 為指令集的計算核心單位，要能夠實作出加減法(ADD、SUB、ADC、SBB)所以 ALU OP 的部分為兩個 bit 控制另外還要將 zero(相減為 0)和 C(進位)指令顯示出來，所以我們要完成:

1. full adder
2. 16-bit adder
3. 16-bit two’s complement adder
4. Z and C flags

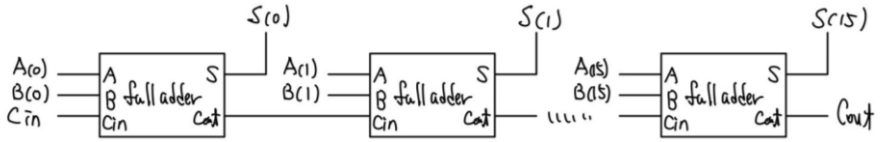
(1) full adder

要完成 fulladder 需要將 XOR、AND 和 OR gate 搭配，如圖所示就完成了一個簡單的 full adder。



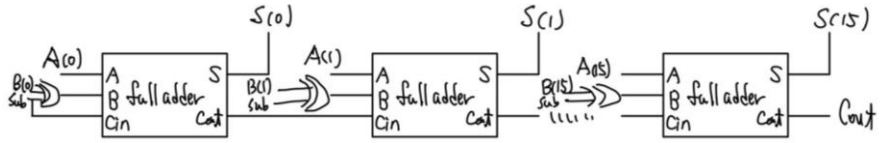
(2) 16-bit adder

由於系統為 16 bit 的系統，所以我們在 ALU 的處理上都要實現 16 bit，這部分我們採取 ripple carry 的方式將 16 個 full adder 串接起來，如此以來就可以實現一個 16bit 的 full adder。




(3) 16-bit two's complement adder

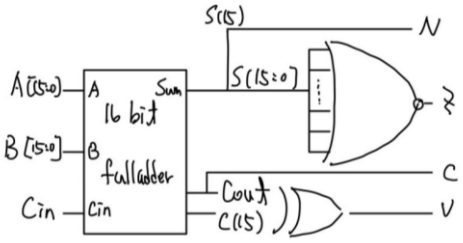
這個步驟是為了要可以實作出減法器，實際做法就是要對 B 取二補數，透過反向加一，將 B 和減法信號透過 XOR 串接起來，再把減法信號接到 CIN 即可。




(4) Z and C flags

最後一步要將 Z 跟 C 指令做出來非常簡單，Z 指令的部分只要將 SUM 的信號全部透過 OR gate 接起來之後再進行反向(全 0 輸出 0 再反向為 1)即可完成。C 指令的部分即為 COUT 信號。

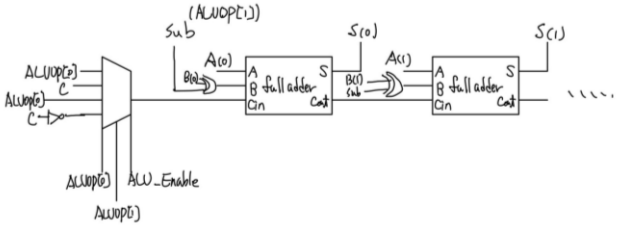
另外作業中雖然不用完成 N 和 V 指令，但是這邊也進一步說明該如何完成。N 指令目的在判斷是否為負號，只要將 S(15)輸出成 N 即可判斷。V 為判斷是否溢位，只要將最後兩級的 COUT 做 XOR 即可完成




(5)16 bit ALU

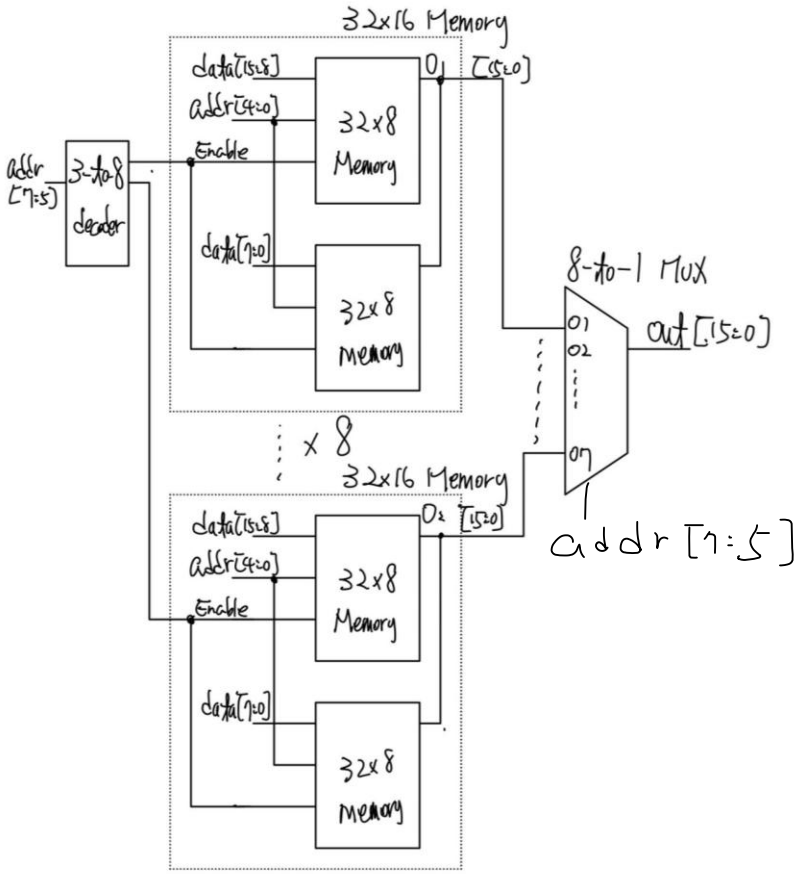
最後將整個根據 ISA 完整將功能實現出來，透過 inst[1]的來控制加減法，inst[0]控制是否要加上 C flag。

| | | | |
|---|---|---|---|
| ADD Rd, Rm, Rn | Rd ← Rm + Rn; | * * * * | 00000_dddmmmnnn_00 |
| ADC Rd, Rm, Rn | Rd ← Rm + Rn + C; | * * * * | 00000_dddmmmnnn_01 |
| SUB Rd, Rm, Rn | Rd ← Rm - Rn; | * * * * | 00000_dddmmmnnn_10 |
| SBB Rd, Rm, Rn | Rd ← Rm - Rn - $\overline{C}$; | * * * * | 00000_dddmmmnnn_11 |
| CMP Rm, Rn | Rm - Rn; | * * * * | 00110_xxxmmmnnn_01 |



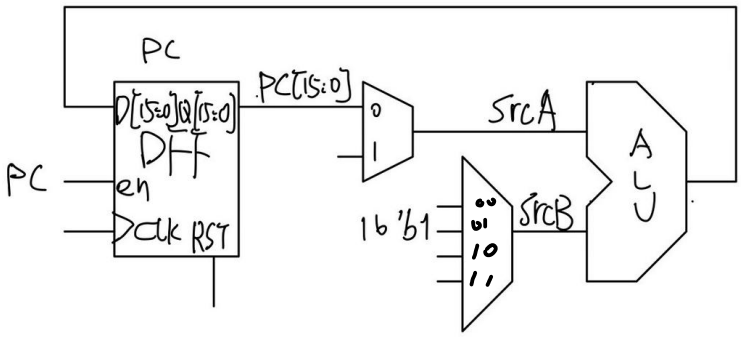
## 三、A 256x16 Memory Module

這個部分我想了一陣子應該要如何處理，後來上網查了許多資料以及跟同學稍加討論之後，才得出結論，如果要實現一個 256x16 的 memory 的話需要先透過一個 3-to-8 的將 addr[7:5]轉換成要儲存的位置的 enable 信號，再來透過兩個 32x8 的 memory 串接變成一個 32x16 的 memory 最後再透過 8 組 32x16 的 memory 完成 256x16 的 memory，由 addr[4:0] 來控制要儲存的 32x16 memory 的位置，即可完成 256x16 的 memory。




## 四、PC Circuitry

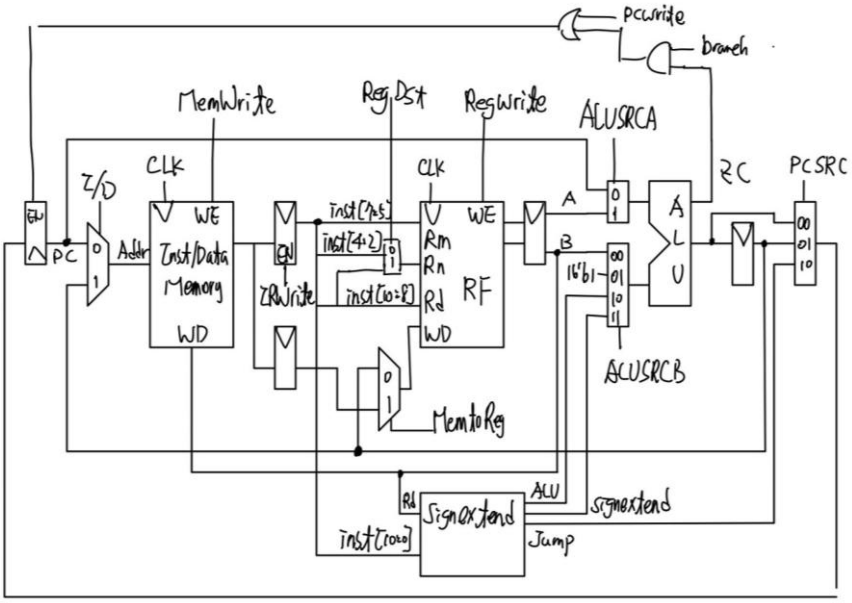
PC 循環其實並沒有太難，只要透過一個暫存器(DFF)以及 ALU 就可以完成，PC 全名是 Program Counter 顧名思義就是要記錄程式現在執行到第幾個，所以將 PC 傳到 ALU 然後再加一，傳到 PC 的 DFF 即可完成一次 PC 紀錄。



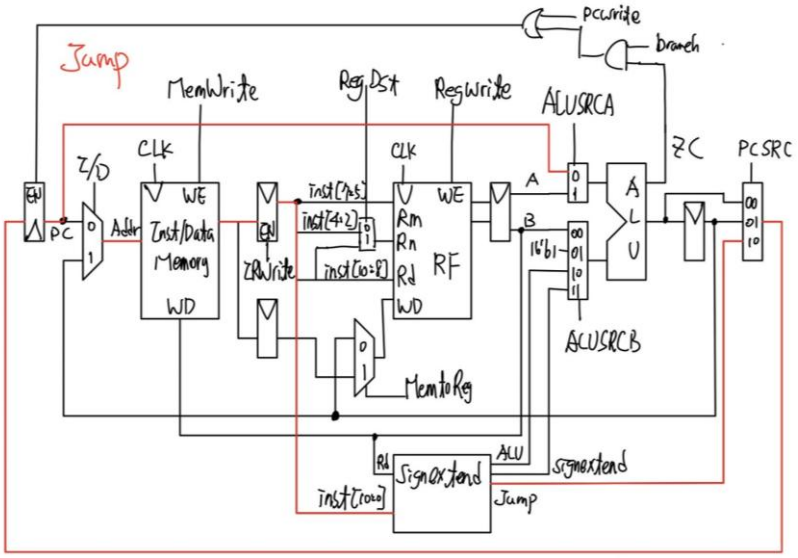
## 五、Complete Datapath

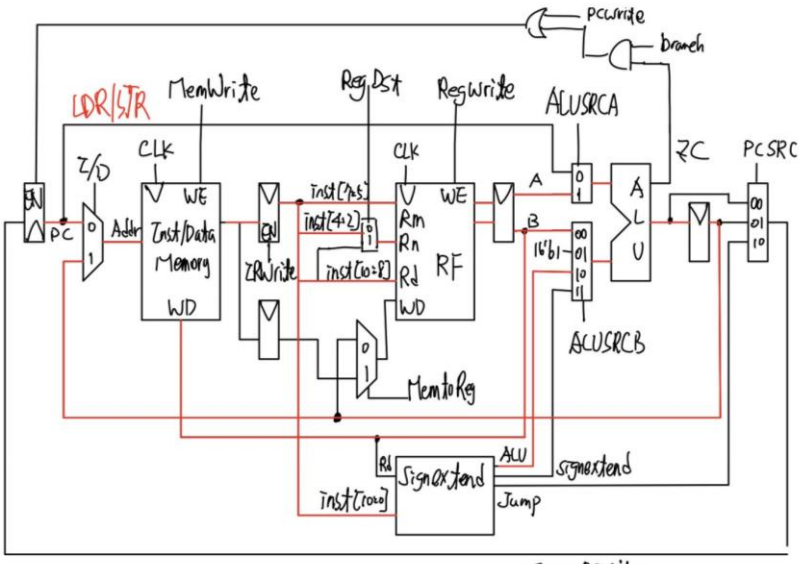
有了上面的基本元件之後就可以將完整的 processor 組合出來了，因為我們是使用 multi-cycle 的方式，所以要將我們整個 data path 用四個 DFF 暫存器切分成五個部分(IF、ID、EXE、MEMORY、WB)，將資料停下來等待下一個階段送出。

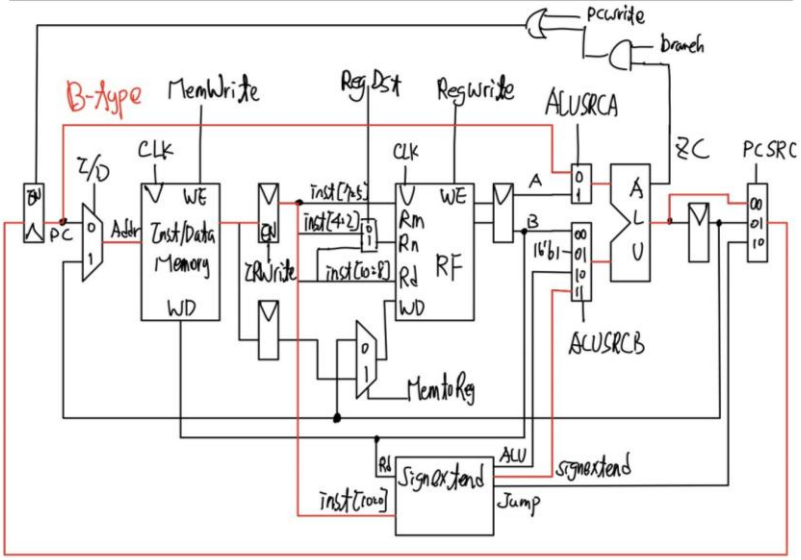
首先，我會先把需要完整五階段的指令實作出來，因為這樣就可以取得大致上的架構，當需要在增加其他指令時只需要增加一些 MUX 即可，這邊先考慮 LDR 以及 STR 指令再來是 R-type 指令以及跳躍指令，完成這些指令之後就可以將大概的架構完成了。

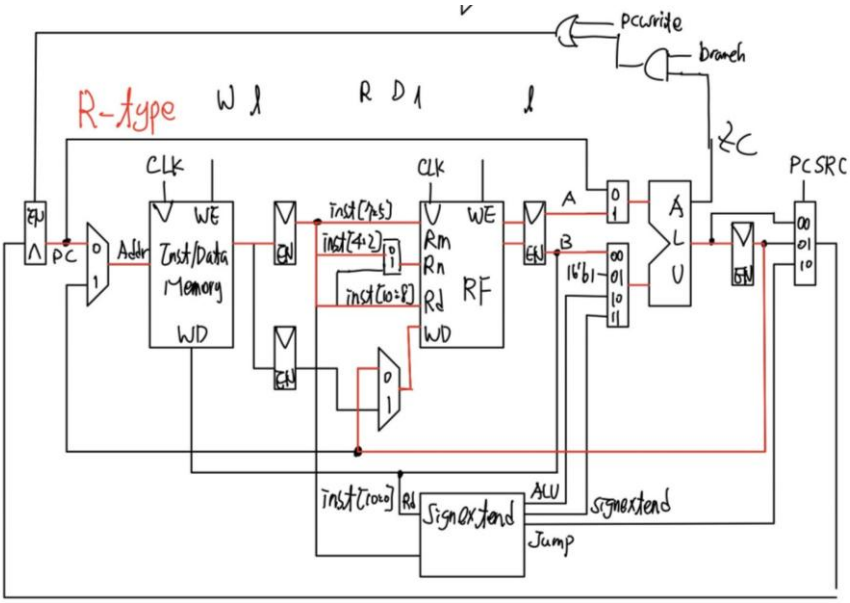



以下為指令會經過的路徑:

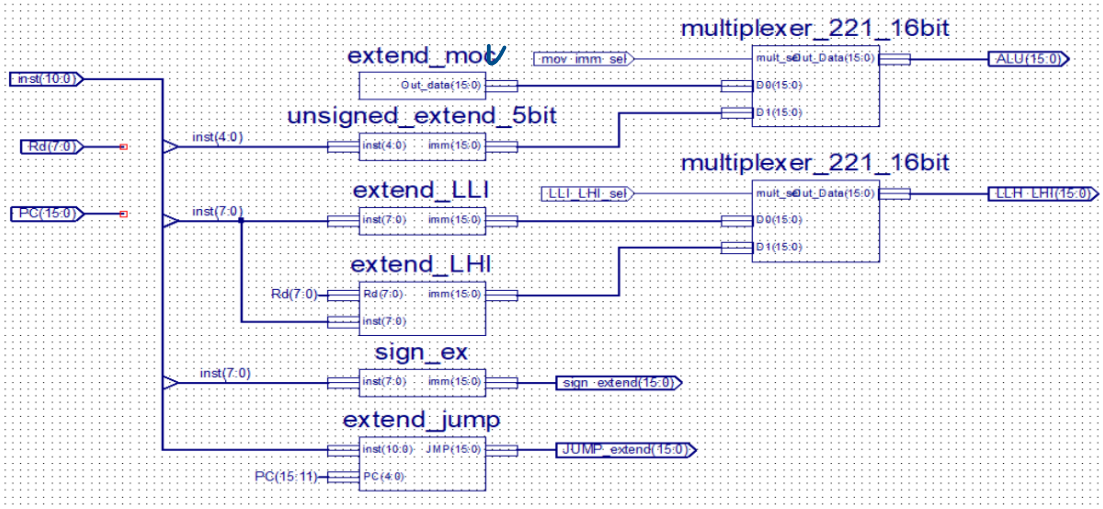



PCWrite
branch
LDR/STR
MemWrite
RegDst
RegWrite
ALUSRCA
ZC
PCSRC
Z/D
CLK
WE
PC
Addr
Inst/Data Memory
WD
EN
ZRWrite
inst[7:5]
inst[4:2]
inst[10:8]
Rm
Rn
Rd
RF
A
B
16'b1
ALU
ALUSRCB
MemtoReg
Rd
Sign extend
ALU
signextend
inst[10:0]
Jump


PCWrite
branch
B-type
MemWrite
RegDst
RegWrite
ALUSRCA
ZC
PCSRC
Z/D
CLK
WE
PC
Addr
Inst/Data Memory
WD
EN
ZRWrite
inst[7:5]
inst[4:2]
inst[10:8]
Rm
Rn
Rd
RF
A
B
16'b1
ALU
ALUSRCB
MemtoReg
Rd
Sign extend
ALU
signextend
inst[10:0]
Jump

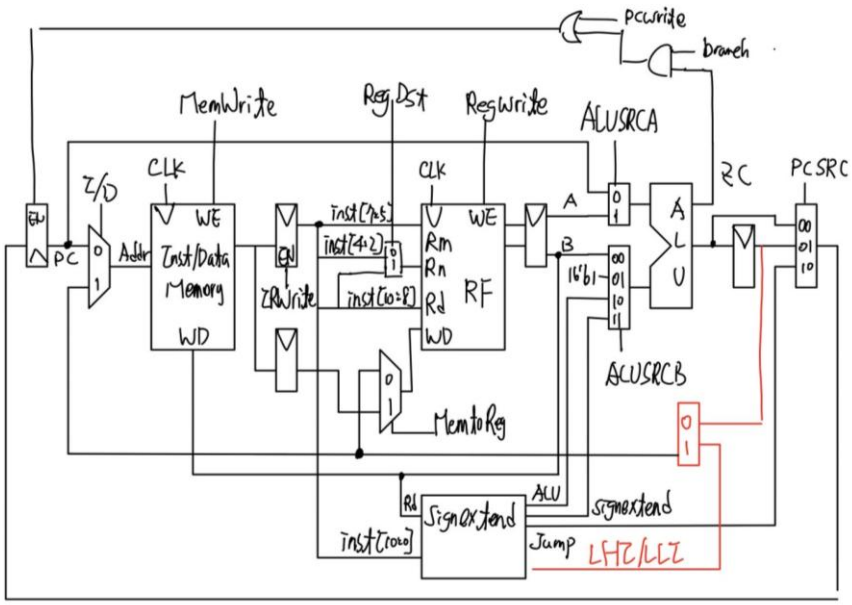

PCWrite
branch
R-type
ZC
PCSRC
CLK
WE
PC
Addr
Inst/Data Memory
WD
EN
inst[7:5]
inst[4:2]
inst[10:8]
Rm
Rn
Rd
RF
A
B
16'b1
ALU
inst[10:0]
Rd
Sign extend
ALU
signextend
Jump

另外 sign extend 的部分，我將 LHI/LLI/MOV/跳躍指令(J)/跳躍指令(B)並且分別透過 mov_imm、LLI_LHI 來控制要輸出那些值，分成了數個部份操作，MOV 傳送 16b0 即可，LLI 為將 inst[7:0]擴展成(0000_0000_inst[7:0])，LHI 為(inst[7:0]_Rd[7:0])，JMP 為(PC[15:11]_inst[10:0])，將會擴展的指令分開來執行我認為會更加清楚。內部元件:



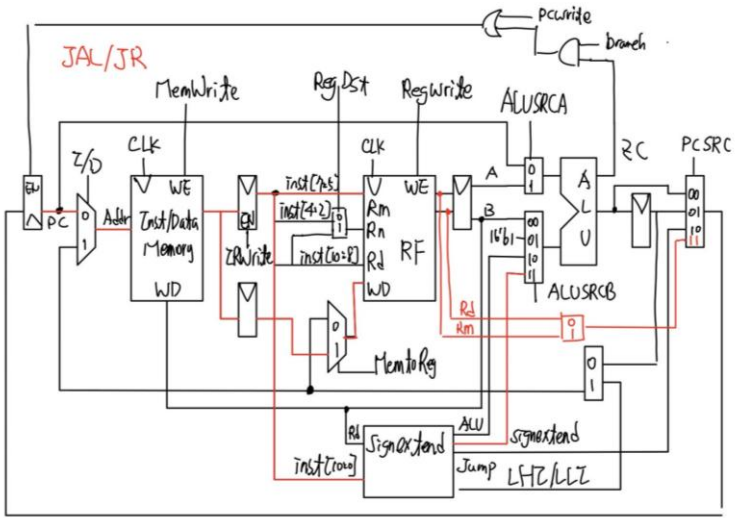

接下來就完成剩下的指令，首先是 LHI/LLI，需要將 sign extend 計算的值輸出，並且加上一個 MUX 去決定要將 ALU 的值還是 LHI/LLI 的值輸出到 PC 使用

| | | | |
|---|---|---|---|
| LHI Rd, #imm8 | Rd ← {imm8, Rd[7:0]}; (imm8 = 0 to 255) | - - - - | 00001_ddd_iiiiiiii |
| LLI Rd, #imm8 | Rd ← {8'h0, imm8}; (imm8 = 0 to 255) | - - - - | 00010_ddd_iiiiiiii |



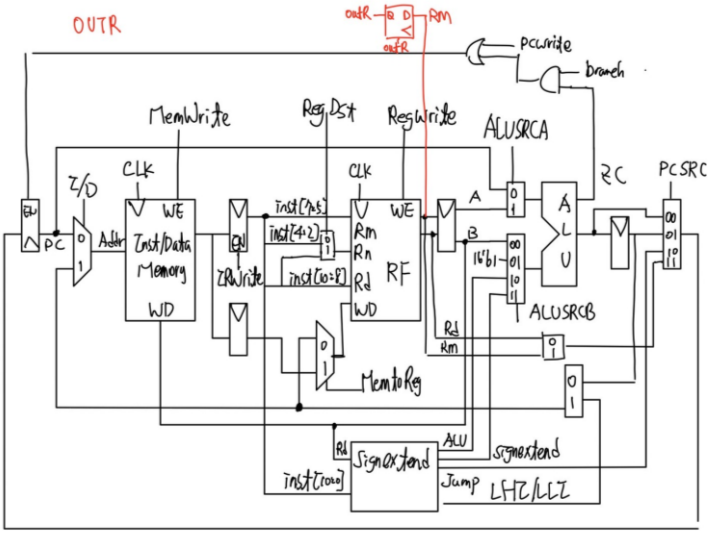

再來要處理 JAL/JR，JAL 為先將 PC 的值給 Rd 後再將 PC 加上 signextend or Rm 之後送到 PC，所以我們需要先將 PCSRC 的部分變成 4-to-1 的多工器，並且將 Rm or Rd 拉出來用一個 MUX 輸出給 PCSRC，使用路徑及新增的元件如下。

| | | | |
|---|---|---|---|
| JAL Rd,label | Rd ← PC; PC ← PC + SignExtend(label); | - - - - | 10001_ddd_disp8 |
| JAL Rd,Rm | Rd ← PC; PC ← Rm; | - - - - | 10010_dddmmm_xxx_yy |
| JR Rd | PC ← Rd; | - - - - | 10011_ddd_xxxxxxxx |
| OutR Rm | OutR ← Rm; | - - - - | 11100_xxx_mmm_xxx_00 |

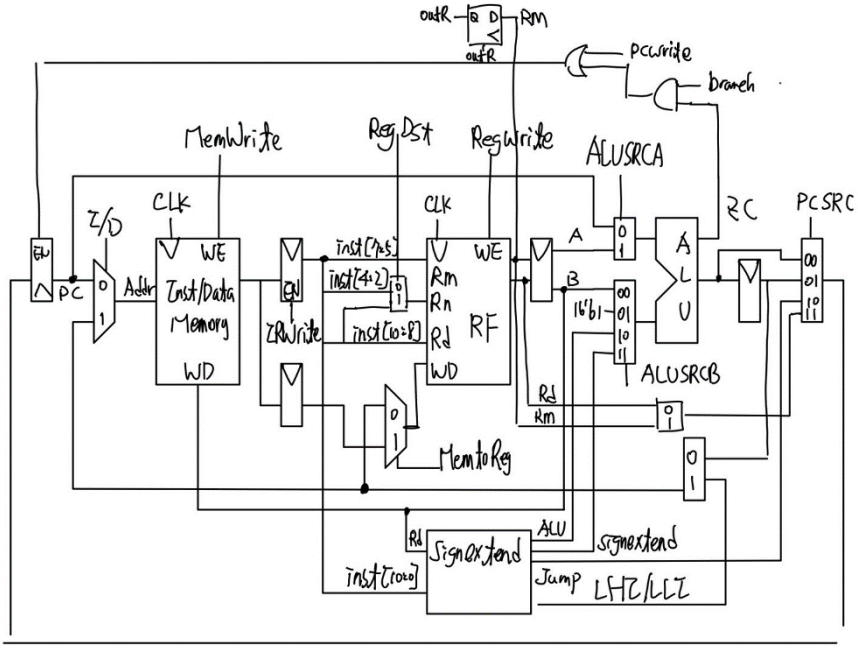



最後一個指令為 OutR、HLT，OutR 只要讓 Rm 拉出來觀察即可，所以我加上了一個 DFF 把 Rm 的值拉出來。

| | | | |
|---|---|---|---|
| OutR Rm | OutR ← Rm; | - - - - | 11100_xxx_mmm_xxx_00 |
| HLT | Set done flag to 1 and halt CPU; | - - - - | 11100_xxxx_xxxxx_01 |



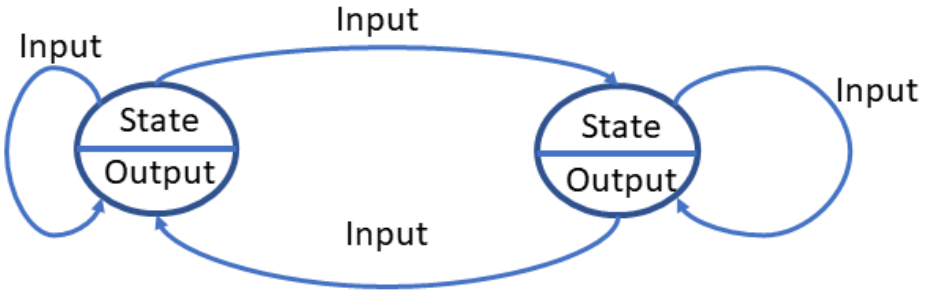

完整 data path:



## 六、Controller

因為這次的系統為 multi-cycle 的系統，每一個指令都有不同的階段周期來完成，每一個週期也都不太一樣，所以我們需要透過設計一個 finite state machine 來協助完成 controller，不同於單週期的 controller 只要將指令寫成 truth table 然後再轉換成 gate 就好。

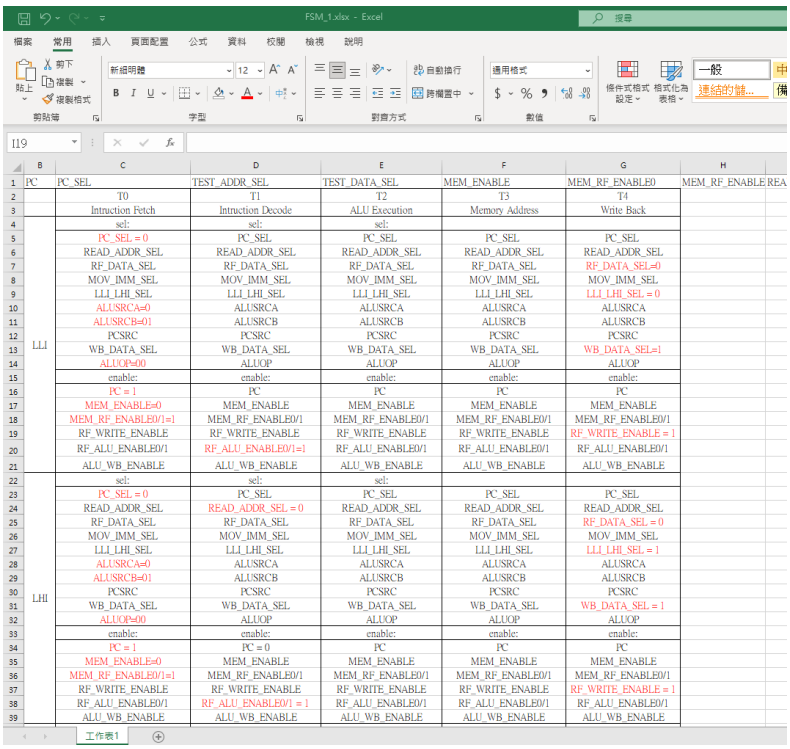
此次我設計的架構是根據摩爾有限狀態機去做設計的，每一個狀態由 CLK 控制，只要當前狀態改變，就會改變輸出的值，非常適合運用在這次的多週期的系統上面。




Generic Moore model

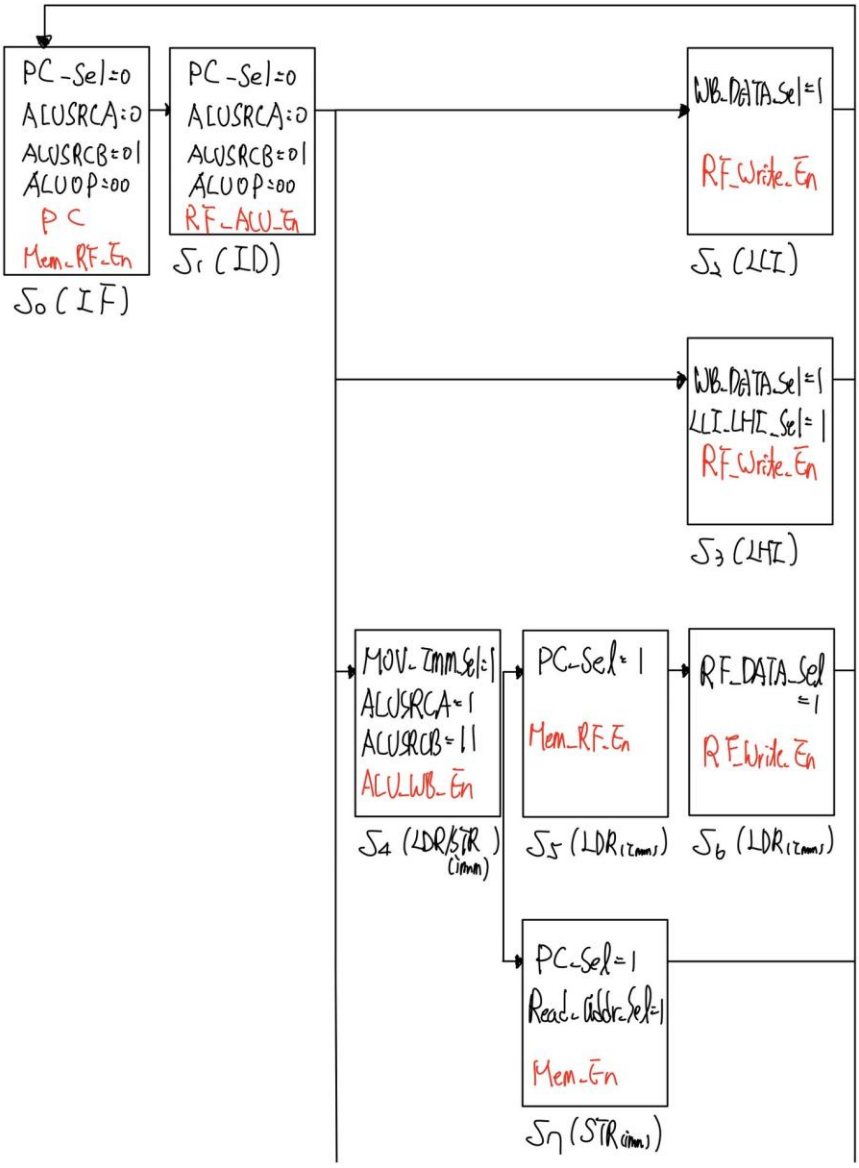
這裡的 input 即為 data path 內的 instruction 根據 instruction 來決定下一個 state，而 output 就是提供給不同的元件的信號。

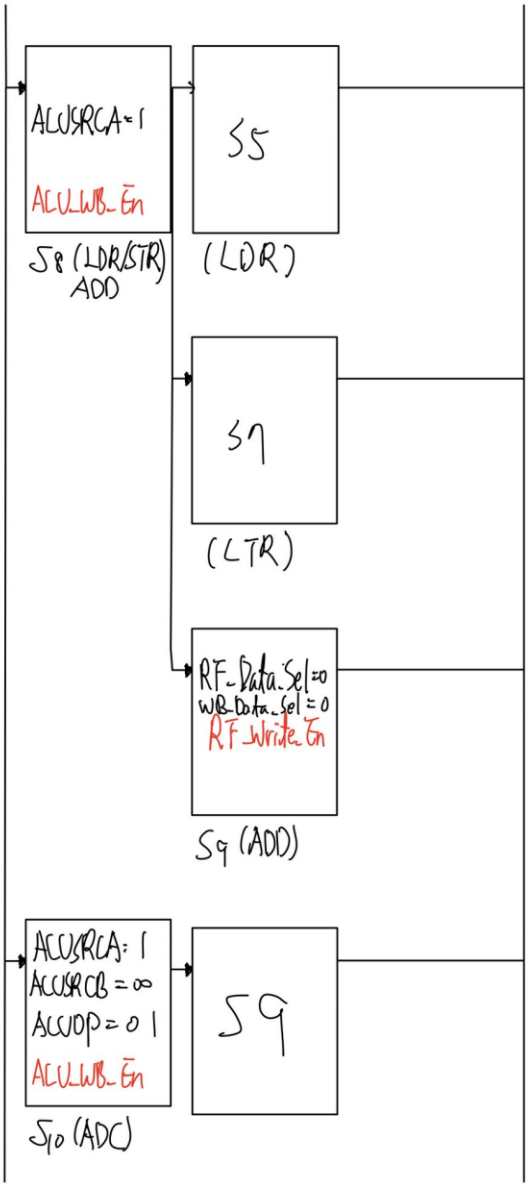
在這裡，我會先講每一個指令的所有狀態先透過 excel 打出所需要的指令信號，然後再將每個指令共同的地方整理出來方便節省 FSM 的流程，最後再將表格轉化成流程圖實現 controller。

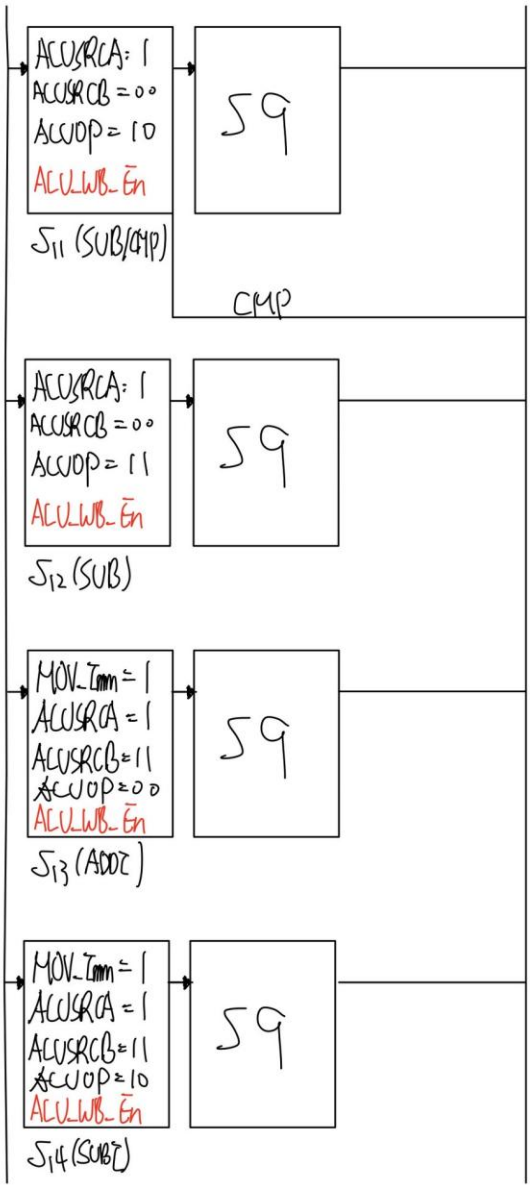


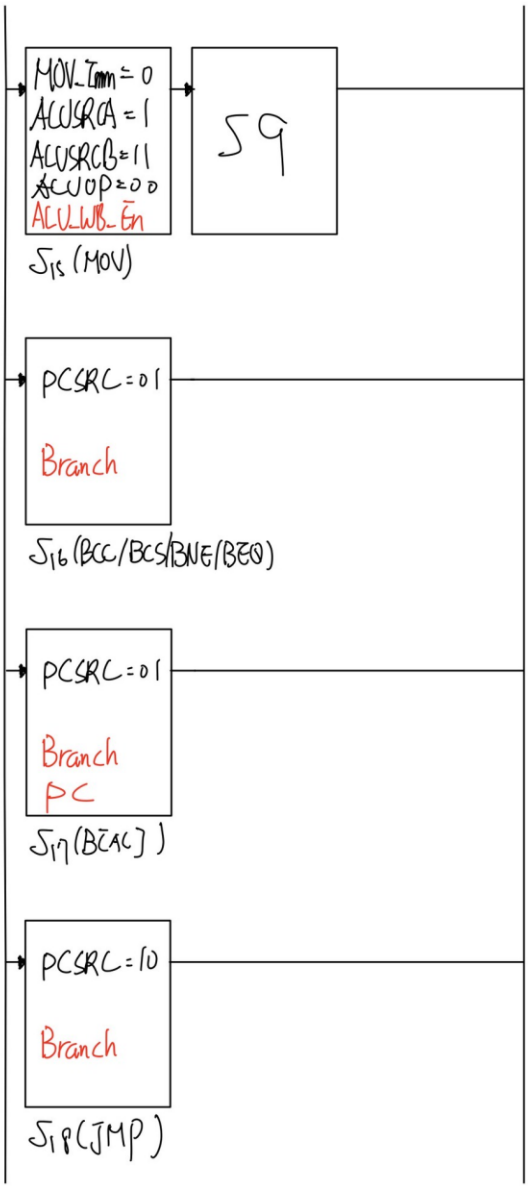
| | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| 1 | PC | PC_SEL | TEST_ADDR_SEL | TEST_DATA_SEL | MEM_ENABLE | MEM_RF_ENABLE0 | MEM_RF_ENABLE REA |
| 2 | | T0 | T1 | T2 | T3 | T4 | |
| 3 | | Intruction Fetch | Intruction Decode | ALU Execution | Memory Address | Write Back | |
| 4 | | sel: | sel: | sel: | | | |
| 5 | | PC_SEL = 0 | PC_SEL | PC_SEL | PC_SEL | PC_SEL | |
| 6 | | READ_ADDR_SEL | READ_ADDR_SEL | READ_ADDR_SEL | READ_ADDR_SEL | READ_ADDR_SEL | |
| 7 | | RF_DATA_SEL | RF_DATA_SEL | RF_DATA_SEL | RF_DATA_SEL | RF_DATA_SEL=0 | |
| 8 | | MOV_IMM_SEL | MOV_IMM_SEL | MOV_IMM_SEL | MOV_IMM_SEL | MOV_IMM_SEL | |
| 9 | | LLI_LHI_SEL | LLI_LHI_SEL | LLI_LHI_SEL | LLI_LHI_SEL | LLI_LHI_SEL = 0 | |
| 10 | | ALUSRCA=0 | ALUSRCA | ALUSRCA | ALUSRCA | ALUSRCA | |
| 11 | | ALUSRCB=01 | ALUSRCB | ALUSRCB | ALUSRCB | ALUSRCB | |
| 12 | | PCSRC | PCSRC | PCSRC | PCSRC | PCSRC | |
| 13 | LLI | WB_DATA_SEL | WB_DATA_SEL | WB_DATA_SEL | WB_DATA_SEL | WB_DATA_SEL=1 | |
| 14 | | ALUOP=00 | ALUOP | ALUOP | ALUOP | ALUOP | |
| 15 | | enable: | enable: | enable: | enable: | enable: | |
| 16 | | PC = 1 | PC | PC | PC | PC | |
| 17 | | MEM_ENABLE=0 | MEM_ENABLE | MEM_ENABLE | MEM_ENABLE | MEM_ENABLE | |
| 18 | | MEM_RF_ENABLE0/1=1 | MEM_RF_ENABLE0/1 | MEM_RF_ENABLE0/1 | MEM_RF_ENABLE0/1 | MEM_RF_ENABLE0/1 | |
| 19 | | RF_WRITE_ENABLE | RF_WRITE_ENABLE | RF_WRITE_ENABLE | RF_WRITE_ENABLE | RF_WRITE_ENABLE = 1 | |
| 20 | | RF_ALU_ENABLE0/1 | RF_ALU_ENABLE0/1=1 | RF_ALU_ENABLE0/1 | RF_ALU_ENABLE0/1 | RF_ALU_ENABLE0/1 | |
| 21 | | ALU_WB_ENABLE | ALU_WB_ENABLE | ALU_WB_ENABLE | ALU_WB_ENABLE | ALU_WB_ENABLE | |
| 22 | | sel: | sel: | sel: | | | |
| 23 | | PC_SEL = 0 | PC_SEL | PC_SEL | PC_SEL | PC_SEL | |
| 24 | | READ_ADDR_SEL | READ_ADDR_SEL = 0 | READ_ADDR_SEL | READ_ADDR_SEL | READ_ADDR_SEL | |
| 25 | | RF_DATA_SEL | RF_DATA_SEL | RF_DATA_SEL | RF_DATA_SEL | RF_DATA_SEL = 0 | |
| 26 | | MOV_IMM_SEL | MOV_IMM_SEL | MOV_IMM_SEL | MOV_IMM_SEL | MOV_IMM_SEL | |
| 27 | | LLI_LHI_SEL | LLI_LHI_SEL | LLI_LHI_SEL | LLI_LHI_SEL | LLI_LHI_SEL = 1 | |
| 28 | | ALUSRCA=0 | ALUSRCA | ALUSRCA | ALUSRCA | ALUSRCA | |
| 29 | | ALUSRCB=01 | ALUSRCB | ALUSRCB | ALUSRCB | ALUSRCB | |
| 30 | | PCSRC | PCSRC | PCSRC | PCSRC | PCSRC | |
| 31 | LHI | WB_DATA_SEL | WB_DATA_SEL | WB_DATA_SEL | WB_DATA_SEL | WB_DATA_SEL = 1 | |
| 32 | | ALUOP=00 | ALUOP | ALUOP | ALUOP | ALUOP | |
| 33 | | enable: | enable: | enable: | enable: | enable: | |
| 34 | | PC = 1 | PC = 0 | PC | PC | PC | |
| 35 | | MEM_ENABLE=0 | MEM_ENABLE | MEM_ENABLE | MEM_ENABLE | MEM_ENABLE | |
| 36 | | MEM_RF_ENABLE0/1=1 | MEM_RF_ENABLE0/1 | MEM_RF_ENABLE0/1 | MEM_RF_ENABLE0/1 | MEM_RF_ENABLE0/1 | |
| 37 | | RF_WRITE_ENABLE | RF_WRITE_ENABLE | RF_WRITE_ENABLE | RF_WRITE_ENABLE | RF_WRITE_ENABLE = 1 | |
| 38 | | RF_ALU_ENABLE0/1 | RF_ALU_ENABLE0/1 = 1 | RF_ALU_ENABLE0/1 | RF_ALU_ENABLE0/1 | RF_ALU_ENABLE0/1 | |
| 39 | | ALU_WB_ENABLE | ALU_WB_ENABLE | ALU_WB_ENABLE | ALU_WB_ENABLE | ALU_WB_ENABLE | |

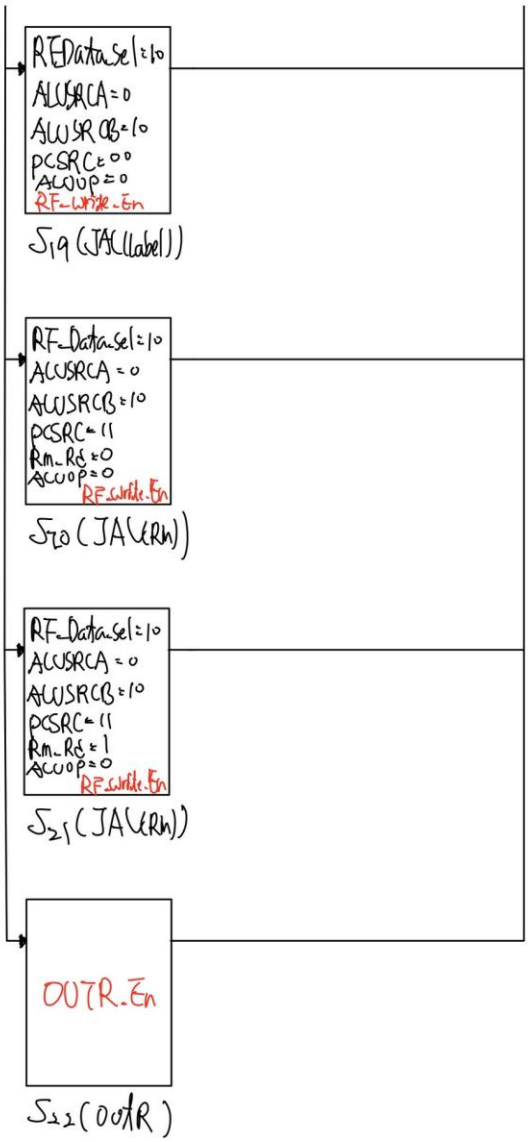
流程圖(這邊的信號為我實際使用的信號，與流程圖有些微差異):




ALUSRCA=1
ALU_WB_En
S8 (LDR/STR) ADD
S5
(LDR)
S7
(LTR)
RF_Data_Sel=0
WB_Data_Sel=0
RF_Write_En
S9 (ADD)
ALUSRCA: 1
ALUSRCB=00
ALUOP=01
ALU_WB_En
S10 (ADC)
S9


ACCSRCA: 1
ACCSRCB = 00
ACCOP = 10
ACU_WB_En
S9
S11 (SUB/CMP)
CMP
ACCSRCA: 1
ACCSRCB = 00
ACCOP = 11
ACU_WB_En
S9
S12 (SUB)
MOV_Imm = 1
ACCSRCA = 1
ACCSRCB = 11
ACCOP = 00
ACU_WB_En
S9
S13 (ADDZ)
MOV_Imm = 1
ACCSRCA = 1
ACCSRCB = 11
ACCOP = 10
ACU_WB_En
S9
S14 (SUBZ)


MOV_Imm = 0
ALUSRCA = 1
ALUSRCB = 11
ALUOP = 00
ALU_WB_En
S9
S15 (MOV)
PCSRC = 01
Branch
S16 (BCC/BCS/BNE/BEQ)
PCSRC = 01
Branch
PC
S17 (BCAC)
PCSRC = 10
Branch
S18 (JMP)




上方 FSM 流程圖的 S0 為 IF 階段，S1 為 ID 階段，因為指令的這兩個階段都一樣所以可以共用，其他部分就會根據指令的不同而有所不同的狀態，沒寫到的部分即為 don't care 所以我只有將特別要更改的指令標出而已。

接下來為了要實際將 controller 真實的 schematic 畫出來，我們需要先將每個指令跟輸出、狀態的 truth table 標示出來，最後再採取 Programmable Logic Array 的方式將邏輯閘畫出來。

| now state | | | | | input | | | | | | | | | | | | | | | | | Next state | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S4 | S3 | S2 | S1 | S0 | I[15] | I[14] | I[13] | I[12] | I[11] | I[10] | I[9] | I[8] | I[7] | I[6] | I[5] | I[4] | I[3] | I[2] | I[1] | I[0] | rst | S4' | S3' | S2' | S1' | S0' |
| 0 | 0 | 0 | 0 | 0 | | | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | | | | | | | | | | | | | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | | | | | | | | | | | | | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | | | | | | | | | | | | | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | | | | | | | | | | | | | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | | | | | | | | | | | | | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | | | | | | | | | | | | | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | | | | | | | | | | | | | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | | | | | | | | | | 0 | 0 | | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | | | | | | | | | | 0 | 0 | | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | | | | | | | | | | 0 | 0 | | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | | | | | | | | | | 0 | 1 | | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | | | | | | 0 | 1 | | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | | | | | | | | | | 1 | 0 | | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | | | | | | | | | | 0 | 1 | | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | | | | | | | | | | 1 | 0 | | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | | | | | | | | | | 1 | 1 | | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | | | | | | 1 | 1 | | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | | | | | | | | | | | | | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | | | | | | | | | | | | | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | | | | | | | | | | | | | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | | | | | | | | | | | | | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | | | | | | | | | | | | | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | | | | | | | | | | | | | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | | | | | | | | | | | | | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | | | | | | | | | | | | | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | | | | | | | | | | | | | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | | | | | | | | | | | | | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | | | | | | | | | | | | | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | | | | | | | | | | | | | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | | | | | | | | | | 0 | 0 | | 1 | 0 | 1 | 1 | 0 |

以上表格即為將上頁流程圖轉換的 truth table，每個狀態最後都會回到 S0 的階段(沒有再寫出來)，接下來就要製作每個階段所對應信號的 truth table。

| now state | | | | | signal | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S4 | S3 | S2 | S1 | S0 | PC | MEM_enable | MEM_RF_enable | branch | RF_Write_enable | RF_ALU_enable0/1 | ALU_WB_enable | Rm_Rd | OutR | RF_data_sel | Read_addr_sel | PC_sel | Mov_imm | LLI_LHI | ALUSRCA | ALUSRCB | WB_data_sel | Pc_src | ALUOP |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 11 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 10 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 11 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 11 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 11 | 0 | 0 | 10 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 11 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 11 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 10 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 11 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

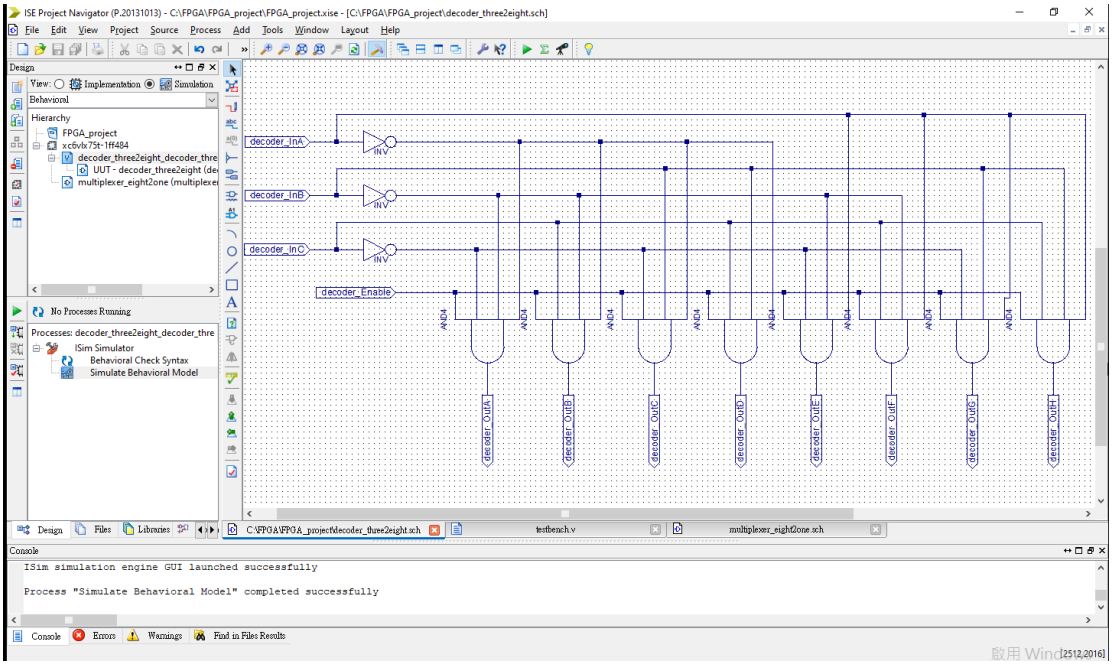
完成了這兩個表格之後，基本上最後一步就是根據 Explicit FSM 將所有狀態表示出來，另外採取 modeling style 2 的撰寫方式。

# Implement schematic entry

測試資料的部分由於有些信號名字不同(部分信號進行優化)，所以模擬的部分可能稍微不同，我有將期中的部份留下以供參考。

## 1. 3-to-8 decoder

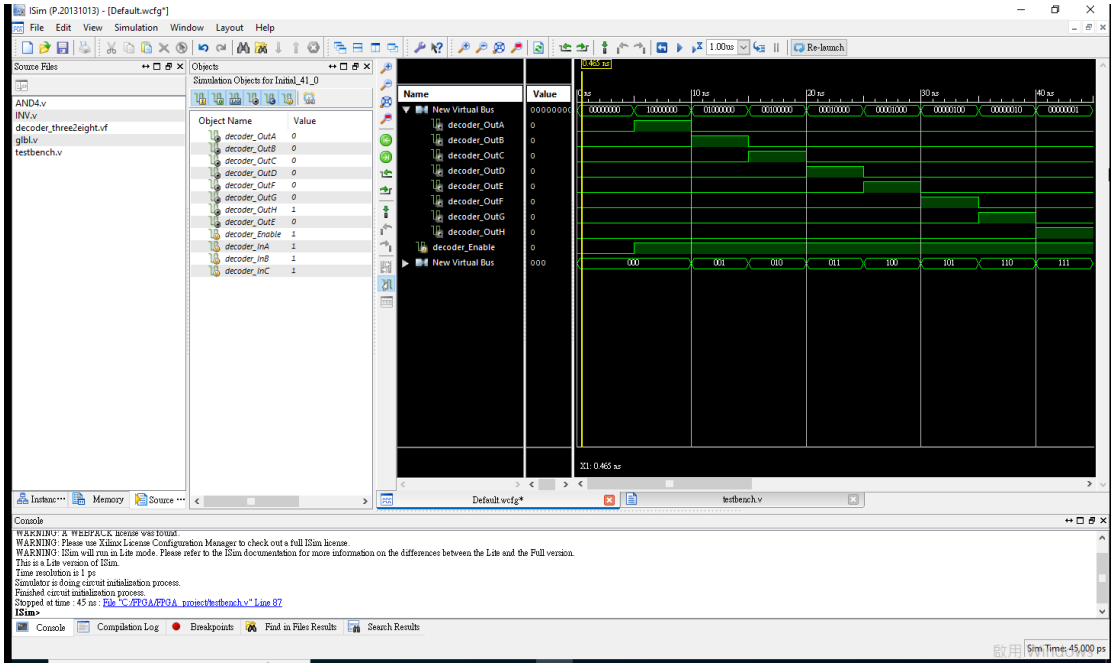
### Schematic



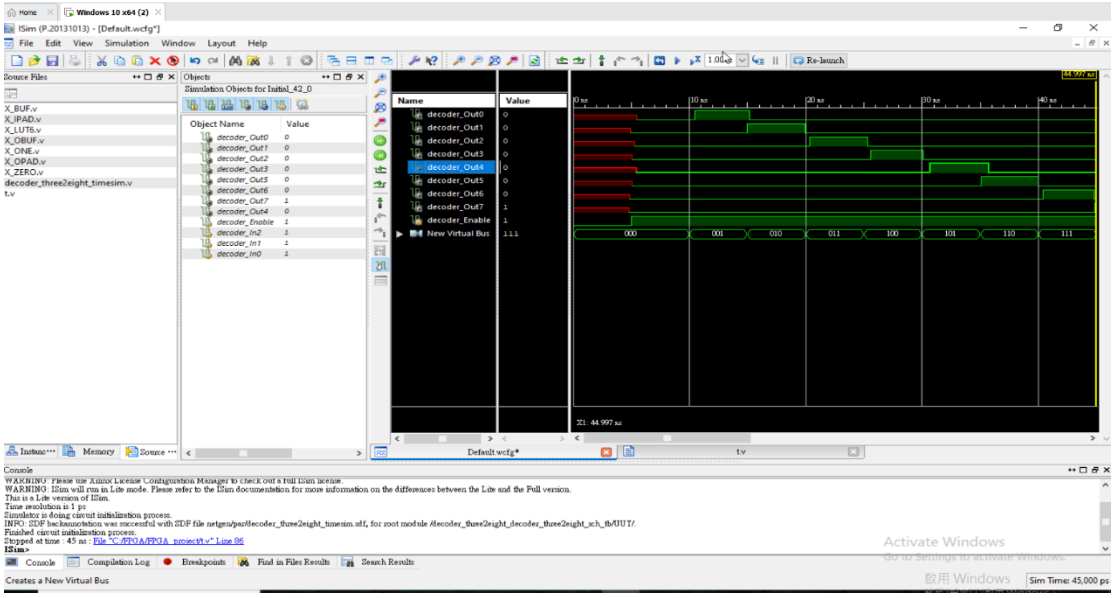

**Simulation:**

根據 3 bit 選擇的位置輸出的 out 改為 0。

**Behavior:**



## Post-Route:




## VHDL

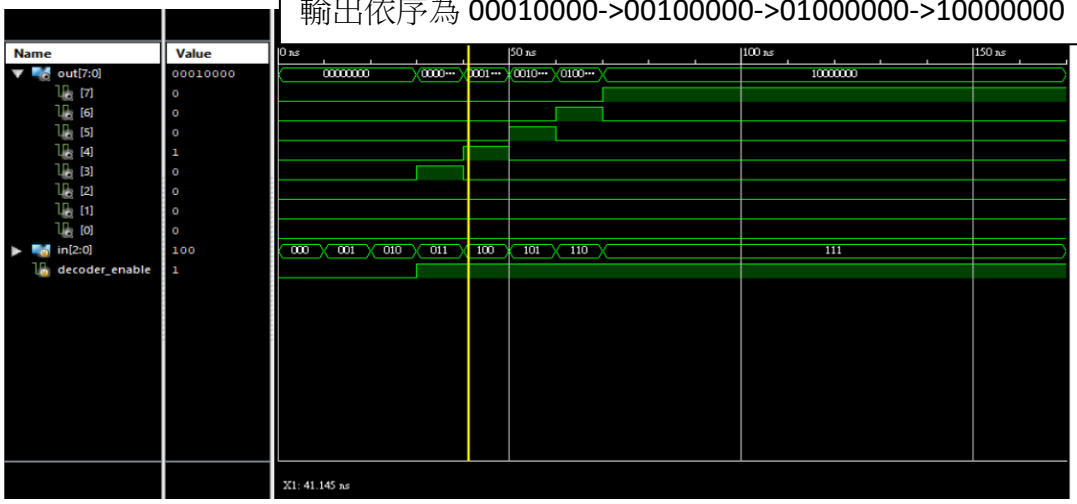
```
1   `timescale 1ns / 1ps
2   module decoder_3to8(
3       input [2:0]in,
4       input decoder_enable,
5       output reg [7:0]out
6       );
7       always@(*)
8       if(decoder_enable)
9           case(in)
10              3'b000:out=8'b0000_0001;
11              3'b001:out=8'b0000_0010;
12              3'b010:out=8'b0000_0100;
13              3'b011:out=8'b0000_1000;
14              3'b100:out=8'b0001_0000;
15              3'b101:out=8'b0010_0000;
16              3'b110:out=8'b0100_0000;
17              3'b111:out=8'b1000_0000;
18          endcase
19      else
20          out = 8'b0;
21
22
23  endmodule
24
```

**Testbanch:**

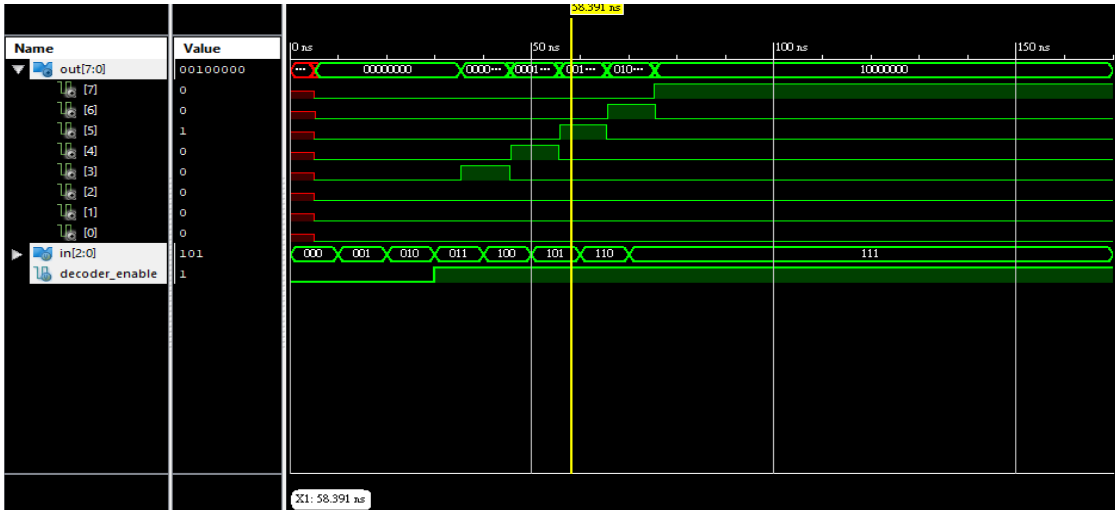
```
1   `timescale 1ns / 1ps
2   module decoder_3to8_t;
3
4       // Inputs
5       reg [2:0] in;
6       reg decoder_enable;
7
8       // Outputs
9       wire [7:0] out;
10
11      // Instantiate the Unit Under Test (UUT)
12      decoder_3to8 uut (
13          .in(in),
14          .decoder_enable(decoder_enable),
15          .out(out)
16      );
17
18      initial begin
19          // Initialize Inputs
20          in = 0;
21          decoder_enable = 0;
22          #10;in = 1;
23          #10;in = 2;
24          #10;in = 3;
25          decoder_enable = 1;
26          #10;in = 4;
27          #10;in = 5;
28          #10;in = 6;
29          #10;in = 7;
30          #100
31          $finish;
32      end
33
34  endmodule
35
```

**Behavior:**

輸出依序為 00010000->00100000->01000000->10000000

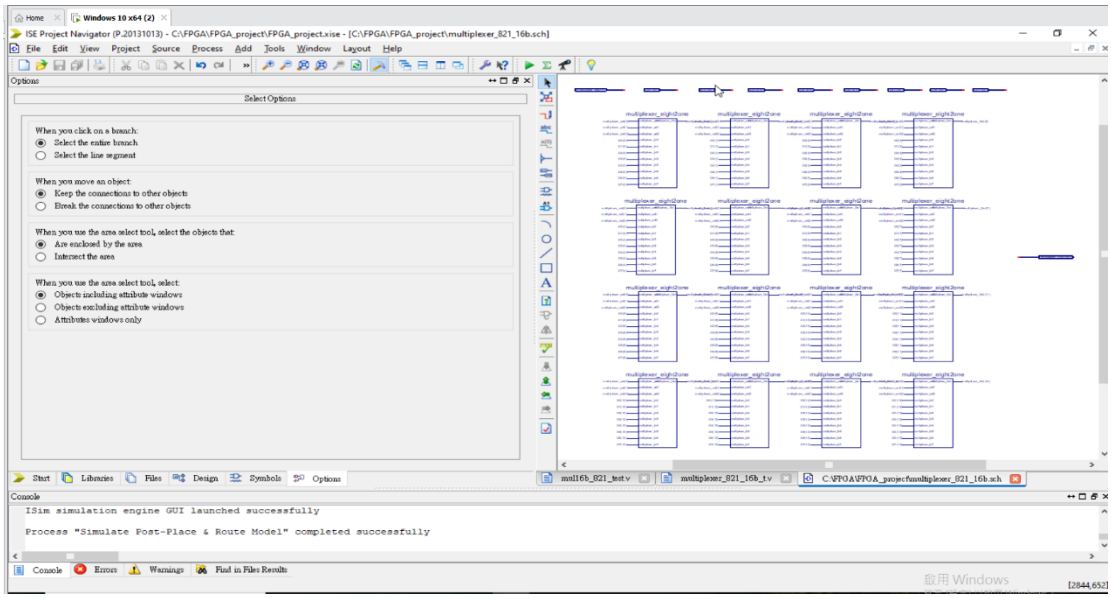


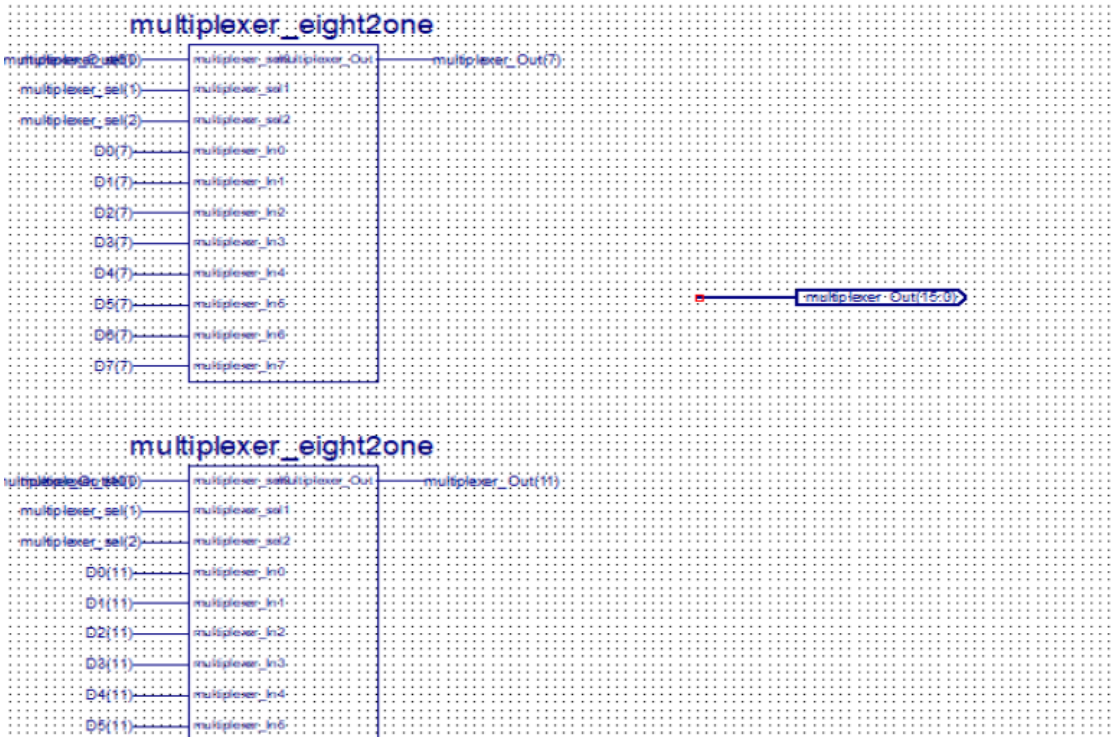
**Post_Route:**




# 2. 16 bit 8-to-1 mux

**Schematic:**

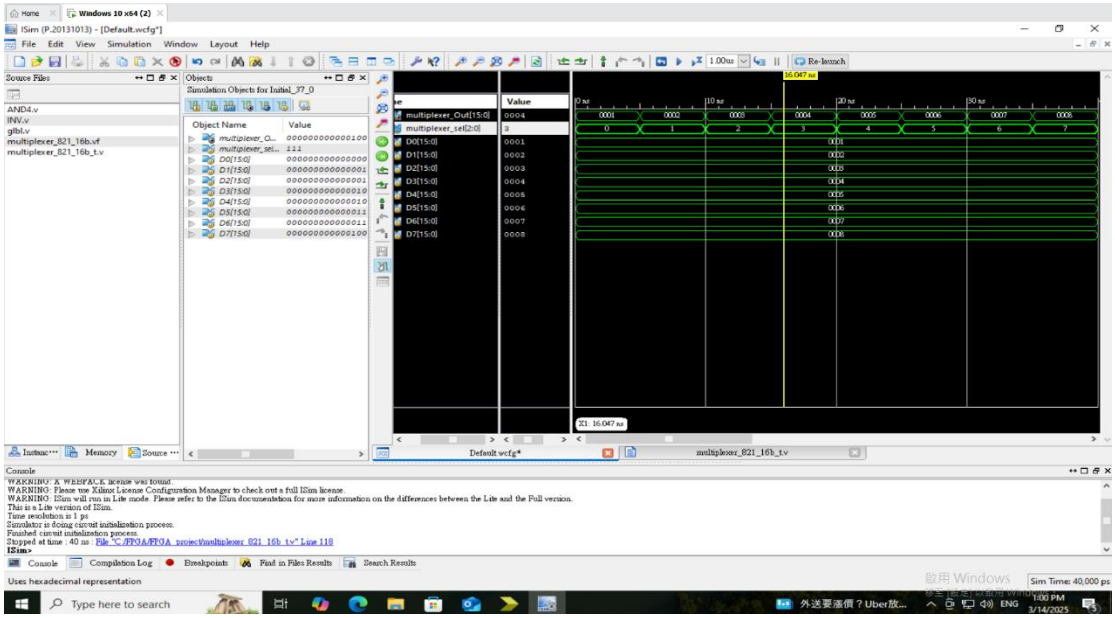




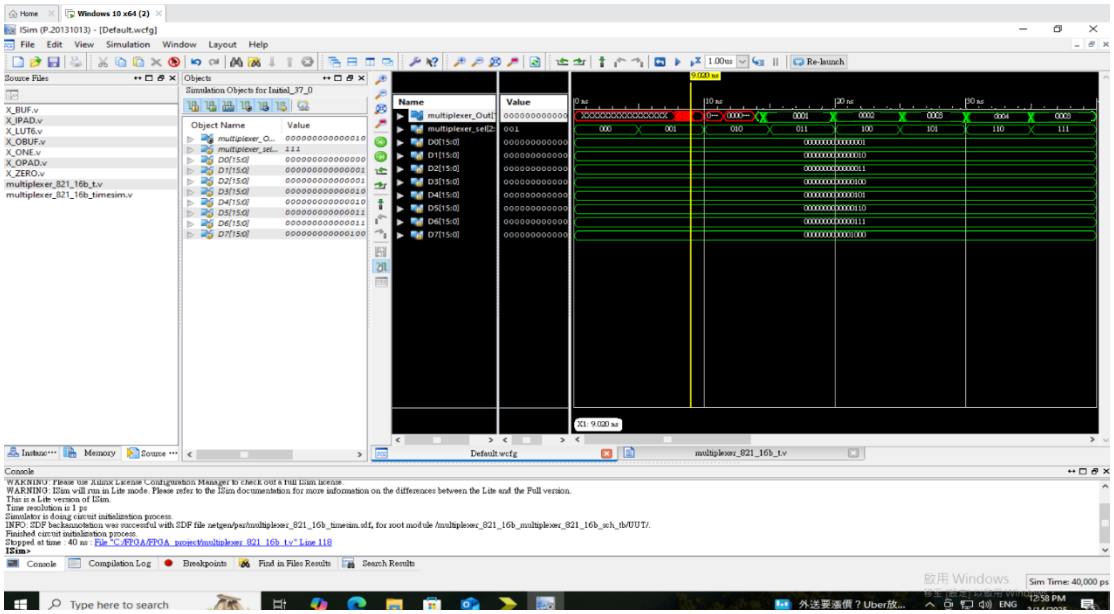

**Simulation:**

改成 16bit 的方式

**Behavior:**



**Post-Route:**



# VHDL

```
`timescale 1ns / 1ps

module mux_8to1(
    input [2:0]sel,
    input [15:0]in0,in1,in2,in3,in4,in5,in6,in7,
    output reg [15:0]out
    );
   always@(*)
   case(sel)
        3'b000:out=in0;
        3'b001:out=in1;
        3'b010:out=in2;
        3'b011:out=in3;
        3'b100:out=in4;
        3'b101:out=in5;
        3'b110:out=in6;
        3'b111:out=in7;
    endcase

endmodule
```

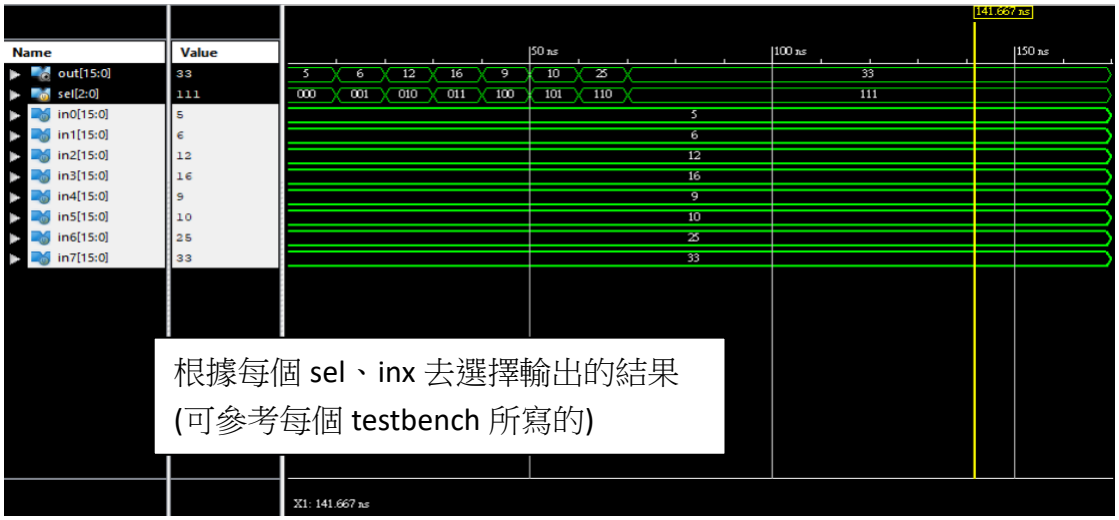
**Testbench:**

```
1   `timescale 1ns / 1ps
2   module mux_8to1_t;
3       // Inputs
4       reg [2:0] sel;
5       reg [15:0] in0;
6       reg [15:0] in1;
7       reg [15:0] in2;
8       reg [15:0] in3;
9       reg [15:0] in4;
10      reg [15:0] in5;
11      reg [15:0] in6;
12      reg [15:0] in7;
13      // Outputs
14      wire [15:0] out;
15
16      // Instantiate the Unit Under Test (UUT)
17      mux_8to1 uut (
18          .sel(sel),
19          .in0(in0),
20          .in1(in1),
21          .in2(in2),
22          .in3(in3),
23          .in4(in4),
24          .in5(in5),
25          .in6(in6),
26          .in7(in7),
27          .out(out)
28      );
29      initial begin
30          // Initialize Inputs
31          sel = 0;
32          in0 = 5;
33          in1 = 6;
34          in2 = 12;
35          in3 = 16;
36          in4 = 9;
37          in5 = 10;
38          in6 = 25;
39          in7 = 33;
40          #10
41          sel = 1;
```

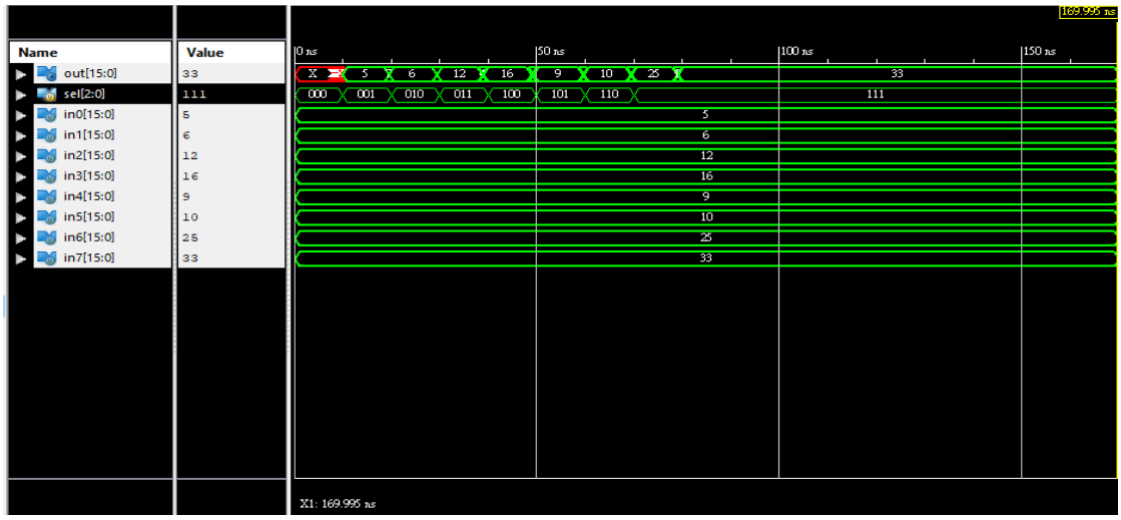
```
41          sel = 1;
42          in0 = 5;
43          in1 = 6;
44          in2 = 12;
45          in3 = 16;
46          in4 = 9;
47          in5 = 10;
48          in6 = 25;
49          in7 = 33;
50          #10
51          sel = 2;
52          in0 = 5;
53          in1 = 6;
54          in2 = 12;
55          in3 = 16;
56          in4 = 9;
57          in5 = 10;
58          in6 = 25;
59          in7 = 33;
60          #10
61          sel = 3;
62          in0 = 5;
63          in1 = 6;
64          in2 = 12;
65          in3 = 16;
66          in4 = 9;
67          in5 = 10;
68          in6 = 25;
69          in7 = 33;
70          #10
71          sel = 4;
72          in0 = 5;
73          in1 = 6;
74          in2 = 12;
75          in3 = 16;
76          in4 = 9;
77          in5 = 10;
78          in6 = 25;
79          in7 = 33;
80          #10
81          sel = 5;
```

```
        in7 = 33;
        #10
        sel = 5;
        in0 = 5;
        in1 = 6;
        in2 = 12;
        in3 = 16;
        in4 = 9;
        in5 = 10;
        in6 = 25;
        in7 = 33;
        #10
        sel = 6;
        in0 = 5;
        in1 = 6;
        in2 = 12;
        in3 = 16;
        in4 = 9;
        in5 = 10;
        in6 = 25;
        in7 = 33;
        #10
        sel = 7;
        in0 = 5;
        in1 = 6;
        in2 = 12;
        in3 = 16;
        in4 = 9;
        in5 = 10;
        in6 = 25;
        in7 = 33;
        // Wait 100 ns for global reset to finish
        #100;
        $finish;
        // Add stimulus here

    end

endmodule
```

**Behavior:**

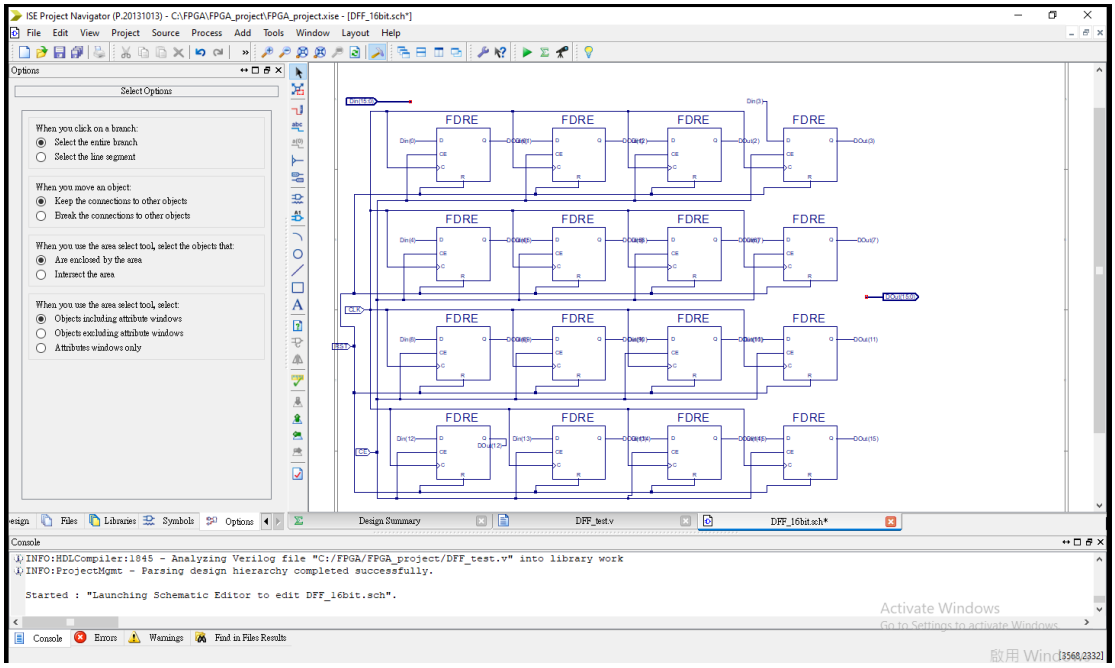


**Post-Route:**

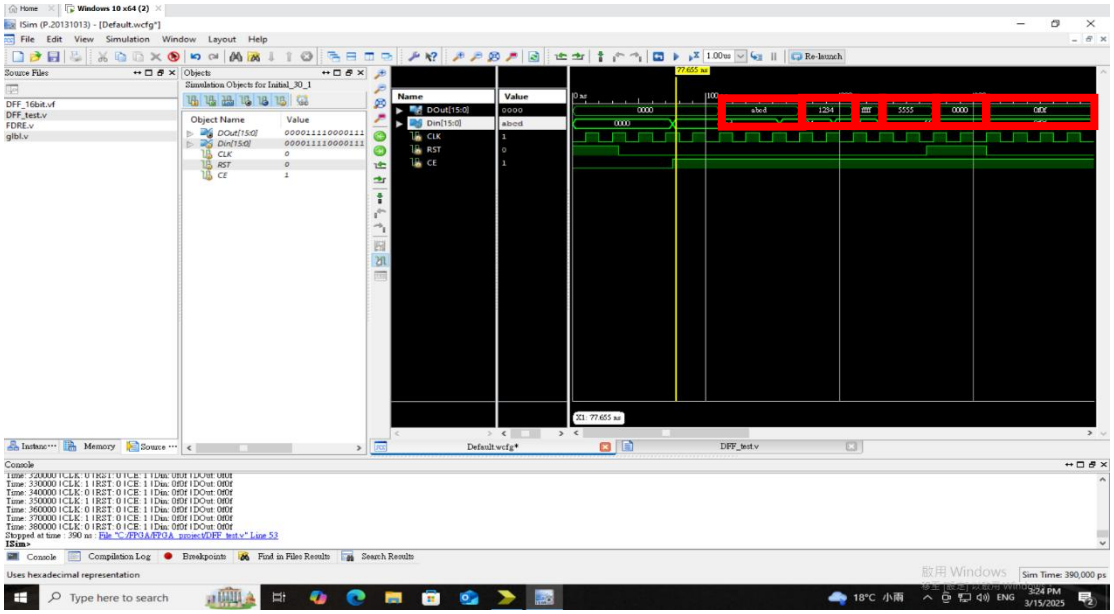



# 3. 16 bit DFF

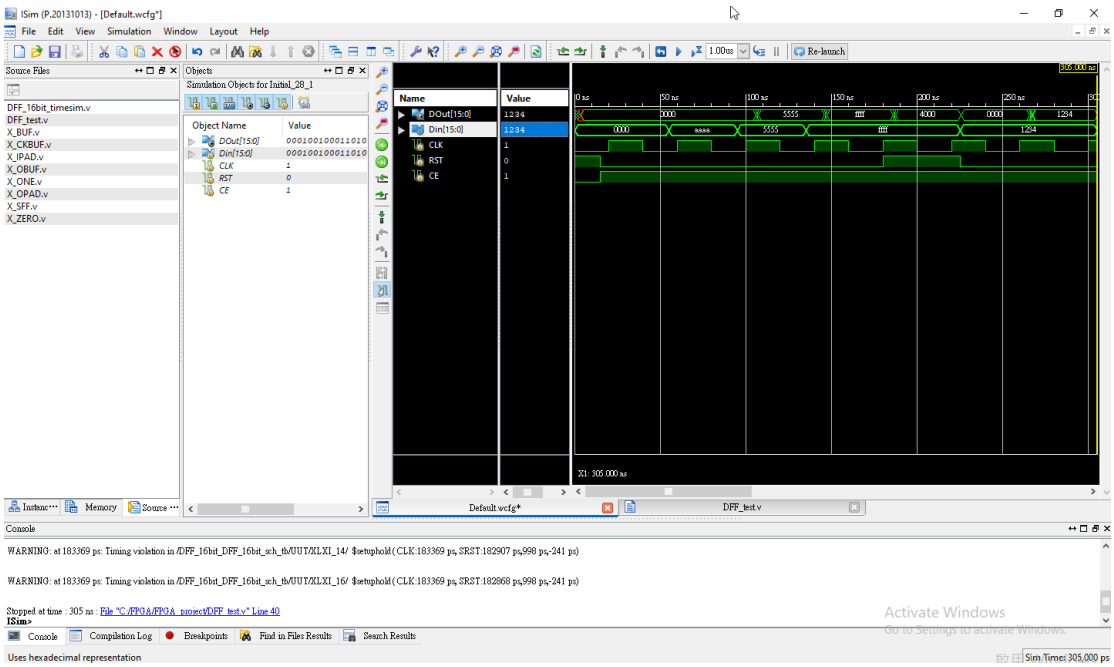
## Schematic




**Behavior:**



## Post-Route:



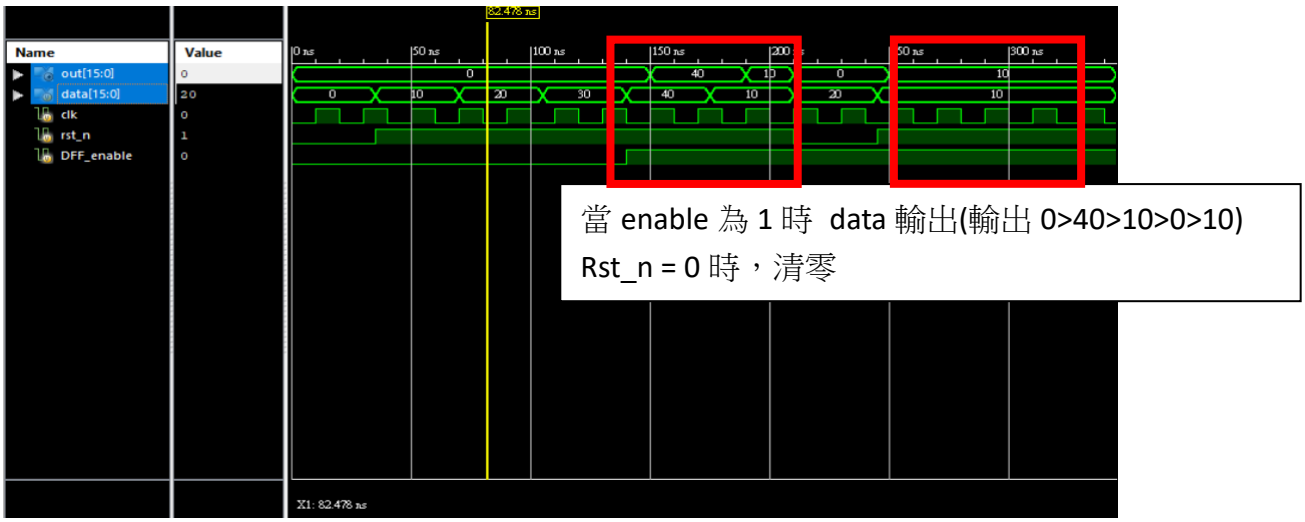
## VHDL

```
`timescale 1ns / 1ps

module DFF_16b(
    input [15:0]data,
    input clk,rst_n,DFF_enable,
    output reg [15:0]out
    );
    always@(posedge clk or negedge rst_n)
        if(!rst_n) out <= 16'b0;
        else if(DFF_enable) out <= data;
        else out <= out;

endmodule
```

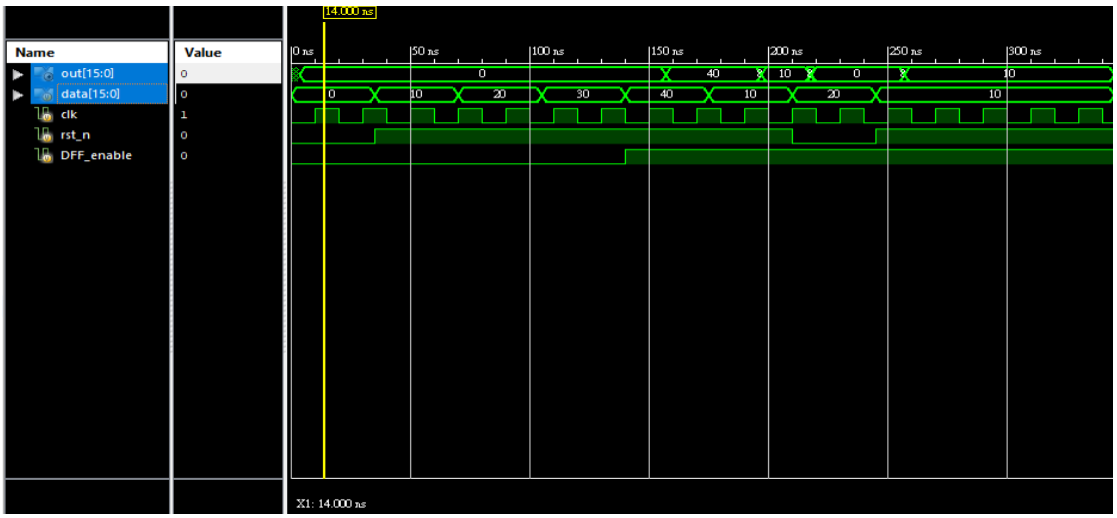
**Testbench:**

```
`timescale 1ns / 1ps
module DFF_16b_t;
    reg [15:0] data;
    reg clk;
    reg rst_n;
    reg DFF_enable;
    wire [15:0] out;
    DFF_16b uut (
        .data(data),
        .clk(clk),
        .rst_n(rst_n),
        .DFF_enable(DFF_enable),
        .out(out)
    );
    always begin
        #10 clk = ~clk;
    end
    initial begin
        data = 0;
        clk = 0;
        rst_n = 0;
        DFF_enable = 0;
        #35
        rst_n = 1;
        data = 10;
        #35
        data = 20;
        #35
        data = 30;
        #35
        DFF_enable = 1;
        data = 40;
        #35
        data = 10;
        #35
        rst_n = 0;
        data = 20;
        #35
        rst_n = 1;
        data = 10;
        #100;
        $finish;
    end
endmodule
```

**Behavior:**

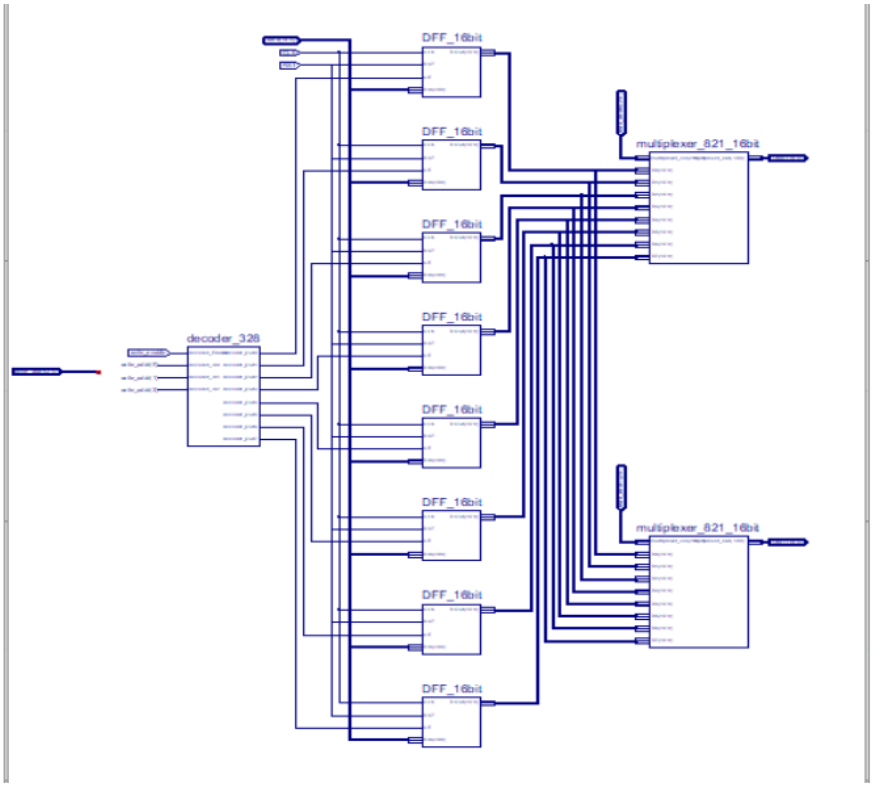


**Post-Route:**

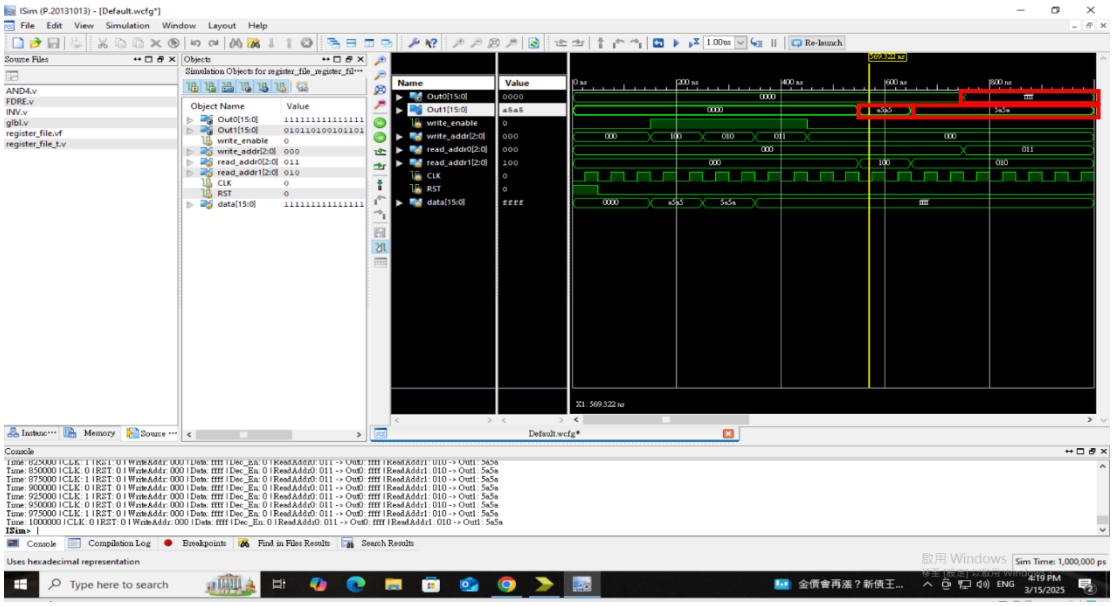



# 4. 16 bit Eight-Register Register File

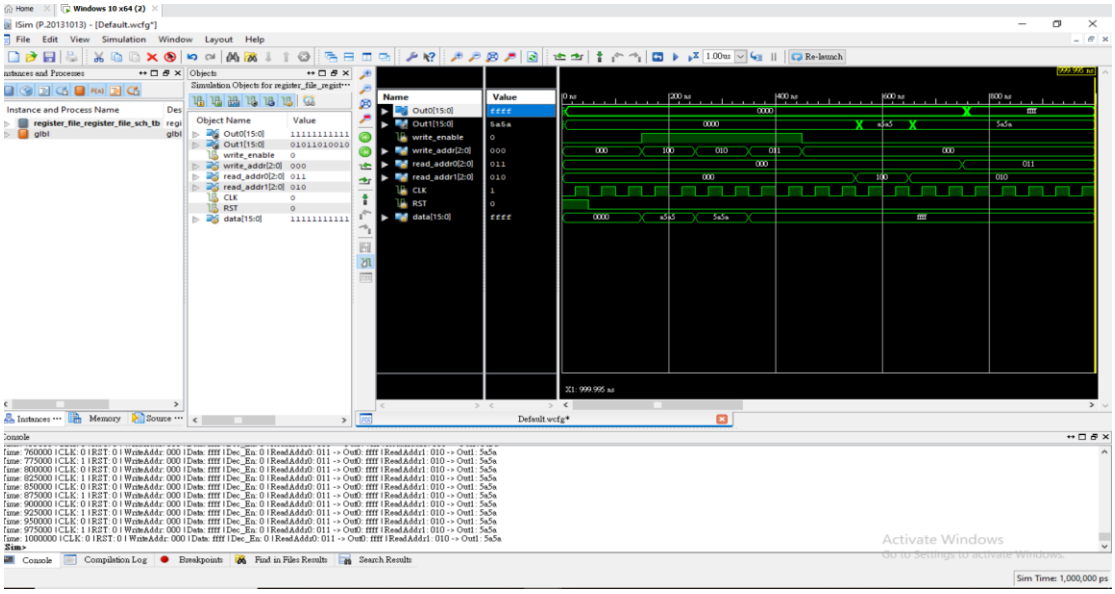
## Schematic



## Behavior:



## Post-Route:



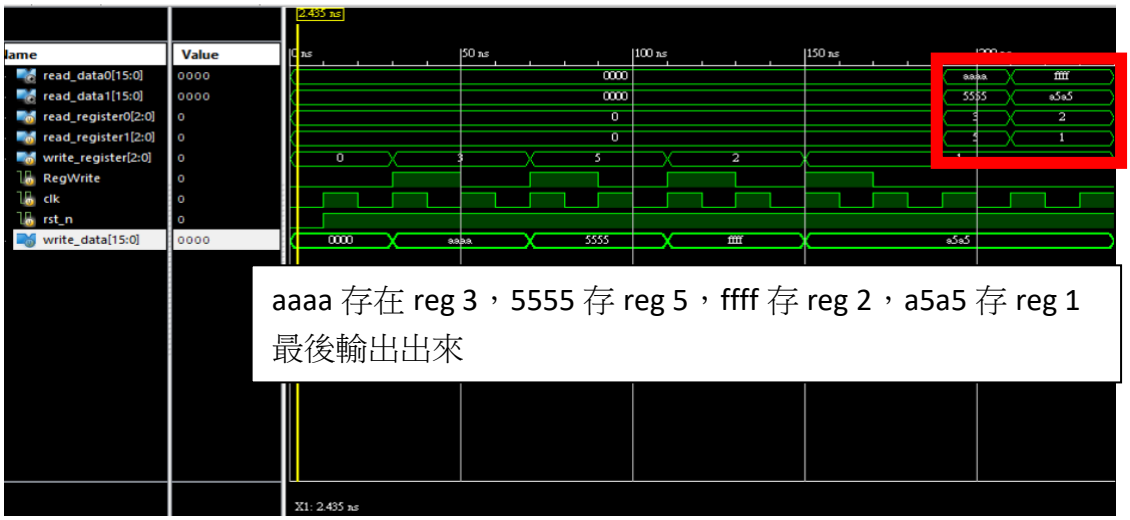
# VHDL

```vhdl
`timescale 1ns / 1ps
module RF_16b(
    input [2:0]read_register0,read_register1,write_register,
    input RegWrite,clk,rst_n,
    input [15:0]write_data,
    output [15:0]read_data0,read_data1
    );
    wire [7:0]DFF_enable;
    wire [15:0]read_data[7:0];
    decoder_3to8 decoder(.in(write_register),.decoder_enable(RegWrite),.out(DFF_enable));
    DFF_16b DFF0(.data(write_data),.clk(clk),.rst_n(rst_n),.DFF_enable(DFF_enable[0]),.out(read_data[0]));
    DFF_16b DFF1(.data(write_data),.clk(clk),.rst_n(rst_n),.DFF_enable(DFF_enable[1]),.out(read_data[1]));
    DFF_16b DFF2(.data(write_data),.clk(clk),.rst_n(rst_n),.DFF_enable(DFF_enable[2]),.out(read_data[2]));
    DFF_16b DFF3(.data(write_data),.clk(clk),.rst_n(rst_n),.DFF_enable(DFF_enable[3]),.out(read_data[3]));
    DFF_16b DFF4(.data(write_data),.clk(clk),.rst_n(rst_n),.DFF_enable(DFF_enable[4]),.out(read_data[4]));
    DFF_16b DFF5(.data(write_data),.clk(clk),.rst_n(rst_n),.DFF_enable(DFF_enable[5]),.out(read_data[5]));
    DFF_16b DFF6(.data(write_data),.clk(clk),.rst_n(rst_n),.DFF_enable(DFF_enable[6]),.out(read_data[6]));
    DFF_16b DFF7(.data(write_data),.clk(clk),.rst_n(rst_n),.DFF_enable(DFF_enable[7]),.out(read_data[7]));
    mux_8to1 mux0(.sel(read_register0),.in0(read_data[0]),.in1(read_data[1]),.in2(read_data[2]),.in3(read_data[3]),.in4(read_data[4]),.in5(read_data[5]),.in6(read_data[6]),.in7(read_data[7]),.out(read_data0));
    mux_8to1 mux1(.sel(read_register1),.in0(read_data[0]),.in1(read_data[1]),.in2(read_data[2]),.in3(read_data[3]),.in4(read_data[4]),.in5(read_data[5]),.in6(read_data[6]),.in7(read_data[7]),.out(read_data1));

endmodule
```

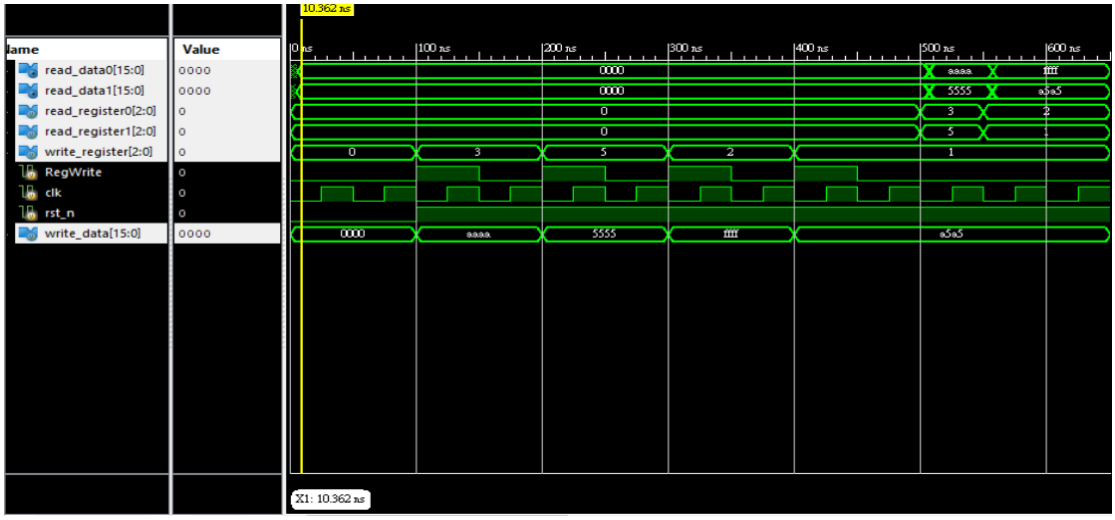
**Testbench:**

```
1   `timescale 1ns / 1ps
2   module RF_16b_t;
3       reg [2:0] read_register0;
4       reg [2:0] read_register1;
5       reg [2:0] write_register;
6       reg RegWrite;
7       reg clk;
8       reg rst_n;
9       reg [15:0] write_data;
10      wire [15:0] read_data0;
11      wire [15:0] read_data1;
12      RF_16b uut (
13          .read_register0(read_register0),
14          .read_register1(read_register1),
15          .write_register(write_register),
16          .RegWrite(RegWrite),
17          .clk(clk),
18          .rst_n(rst_n),
19          .write_data(write_data),
20          .read_data0(read_data0),
21          .read_data1(read_data1)
22      );
23      always begin
24          #25 clk = ~clk;
25      end
26      initial begin
27          read_register0 = 0;
28          read_register1 = 0;
29          write_register = 0;
30          RegWrite = 0;
31          clk = 0;
32          rst_n = 0;
33          write_data = 0;
34          #100 rst_n = 1;RegWrite = 1;write_register = 3;write_data = 16'hAAAA;
35          #50 RegWrite = 0;
36          #50 RegWrite = 1;write_register = 5;write_data = 16'h5555;
37          #50 RegWrite = 0;
38          #50 RegWrite = 1;write_register = 2;write_data = 16'hffff;
39          #50 RegWrite = 0;
40          #50 RegWrite = 1;write_register = 1;write_data = 16'ha5a5;
41          #50 RegWrite = 0;
42          #50 read_register0 = 3;read_register1 = 5;
43          #50 read_register0 = 2;read_register1 = 1;
44       #100 $stop;
45      end
46  endmodule
```

**Behavior:**

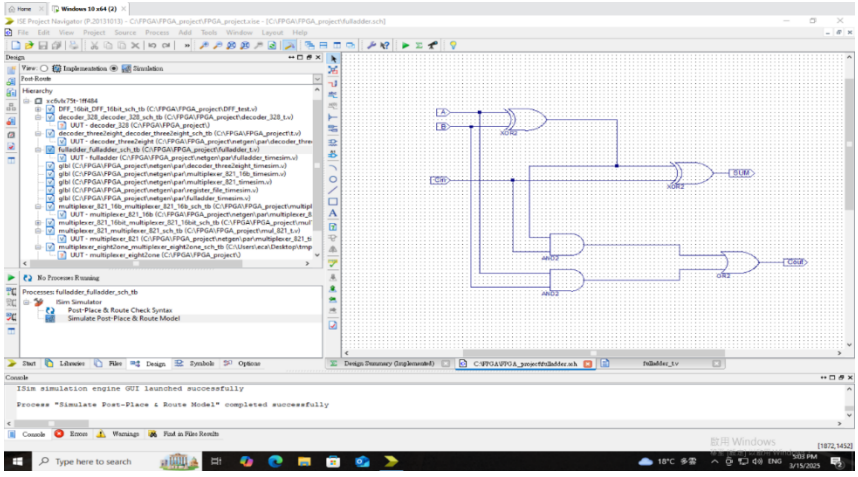


**Post-Route:**




# 5. Full adder

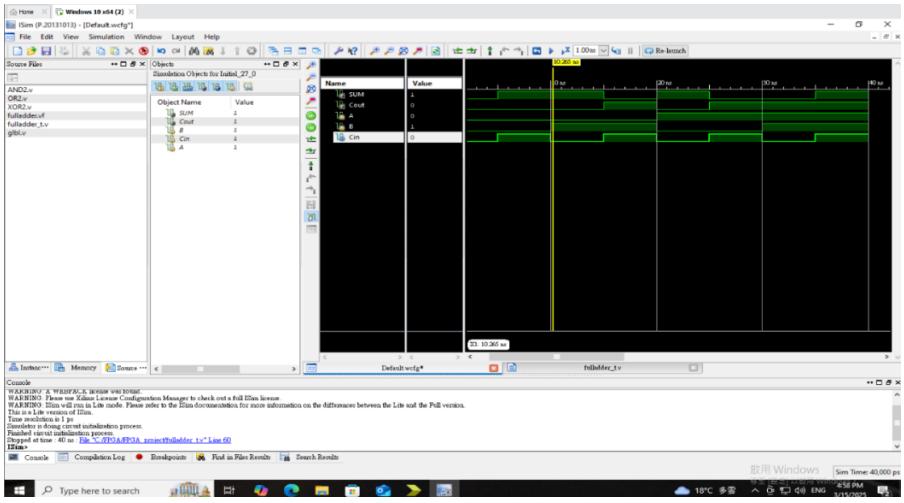
## Schematic



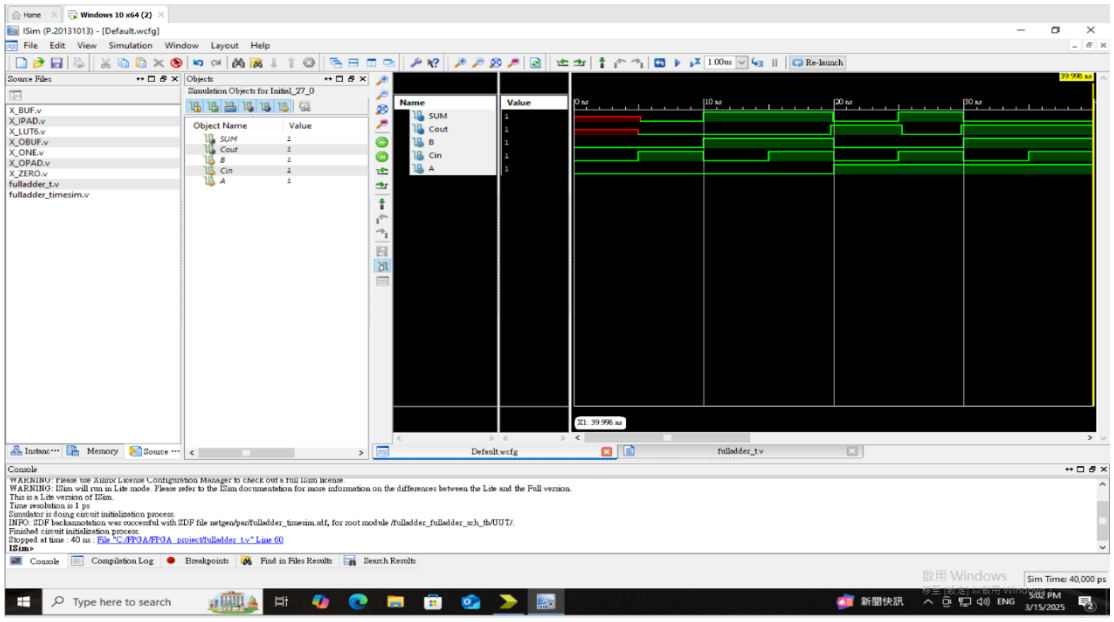
**Simulation:**

基本加法

**Behavior:**



## Post-Route:



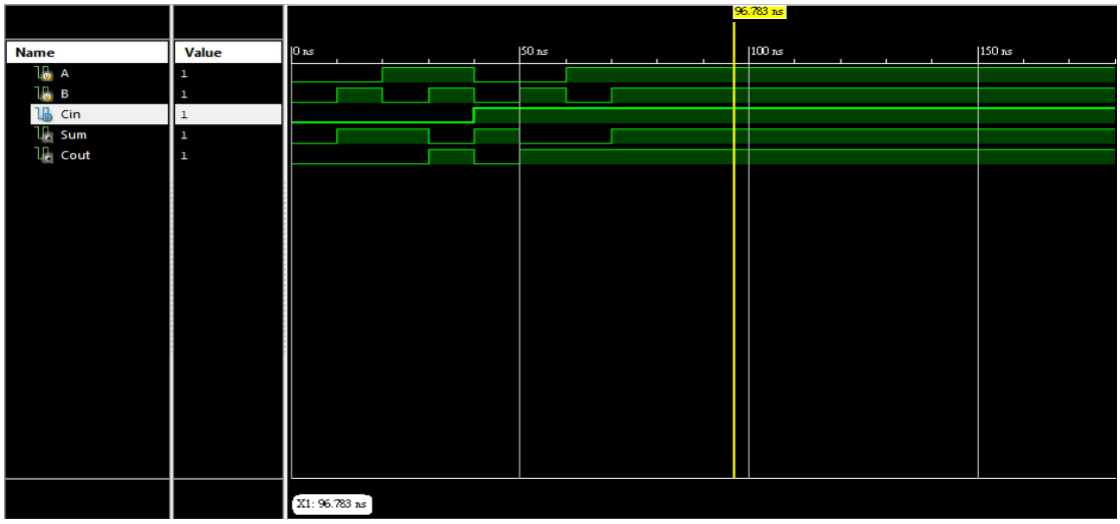
## VHDL

```
`timescale 1ns / 1ps
module full_adder(
    input A,B,Cin,
    output reg Sum,Cout
    );
    always@(*)begin
        Sum = A^B^Cin;
        Cout = (A&B)|((A^B)&Cin);
    end

endmodule
```

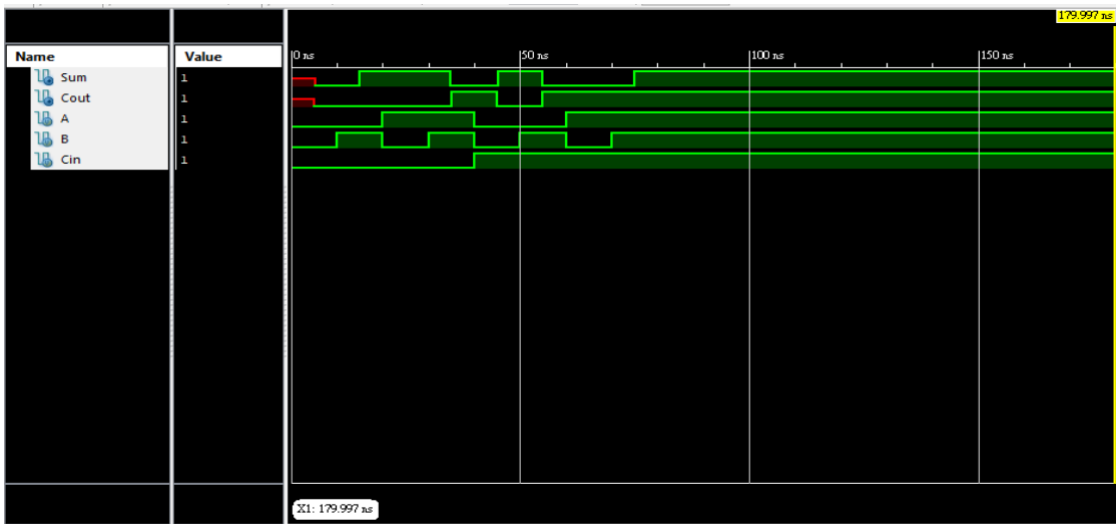
## Testbench:

```
`timescale 1ns / 1ps
module ful_adder_t;
   // Inputs
   reg A;
   reg B;
   reg Cin;
   // Outputs
   wire Sum;
   wire Cout;
   // Instantiate the Unit Under Test (UUT)
   full_adder uut (
      .A(A),
      .B(B),
      .Cin(Cin),
      .Sum(Sum),
      .Cout(Cout)
   );

   initial begin
      // Initialize Inputs
      A = 0;
      B = 0;
      Cin = 0;
      #10 A = 0;B = 1;Cin = 0;
      #10 A = 1;B = 0;Cin = 0;
      #10 A = 1;B = 1;Cin = 0;
      #10 A = 0;B = 0;Cin = 1;
      #10 A = 0;B = 1;Cin = 1;
      #10 A = 1;B = 0;Cin = 1;
      #10 A = 1;B = 1;Cin = 1;
      #100;
        $finish;
   end

endmodule
```

**Behavior:**

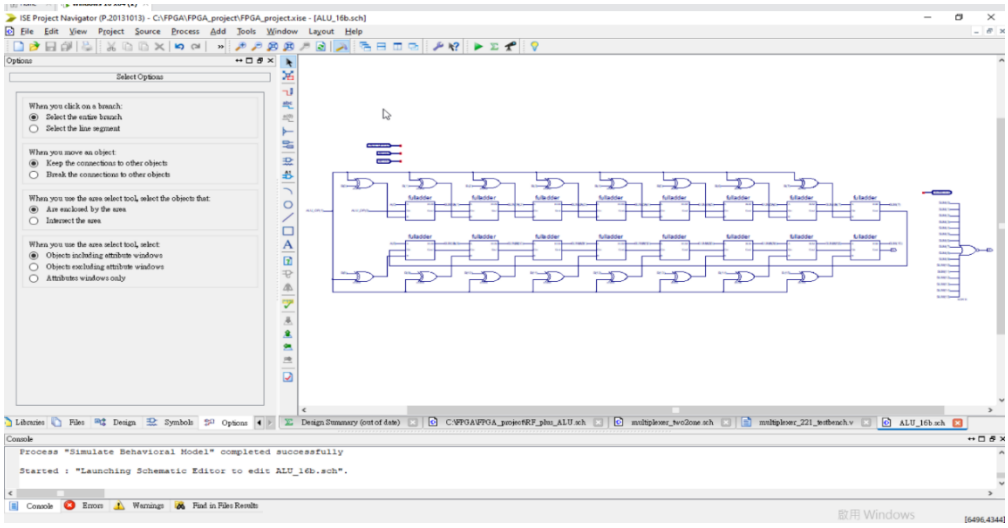


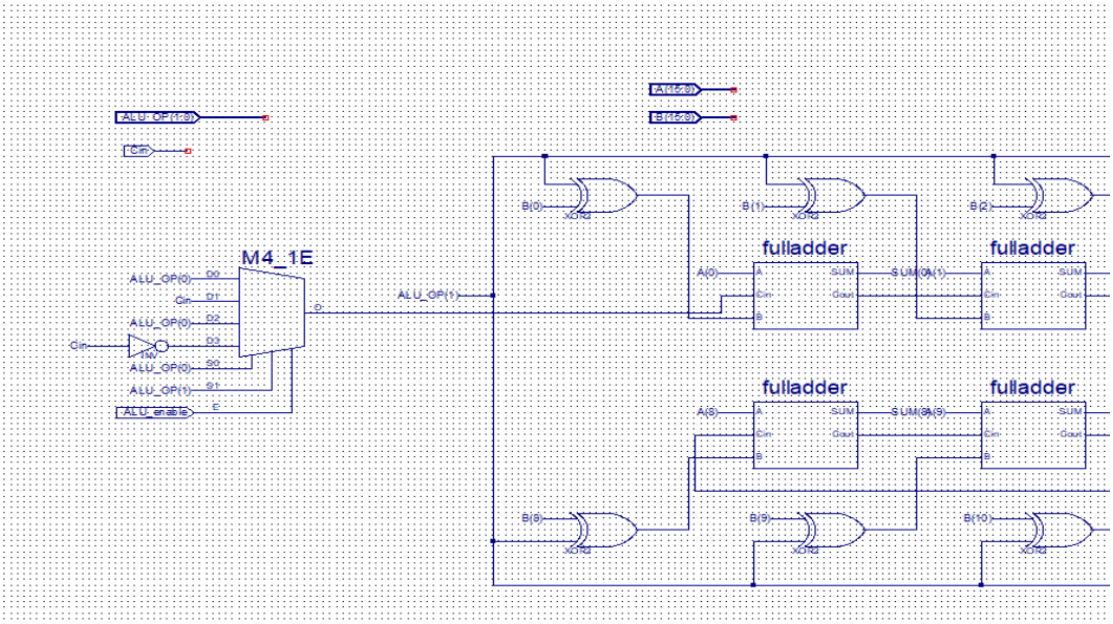

**Post-Route:**



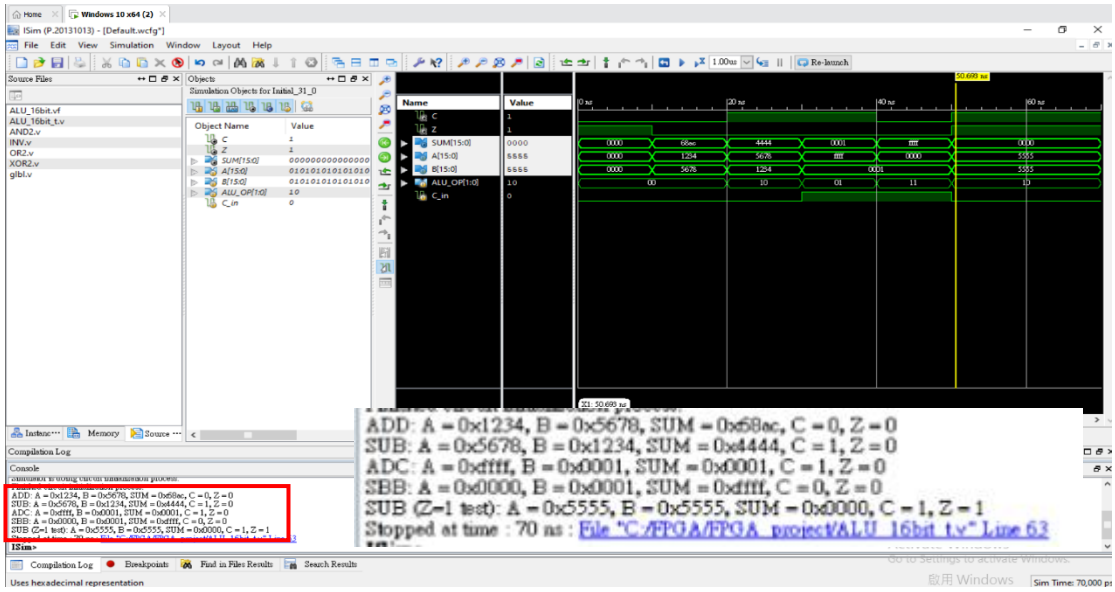

# 6. 16 bit ALU

## Schematic

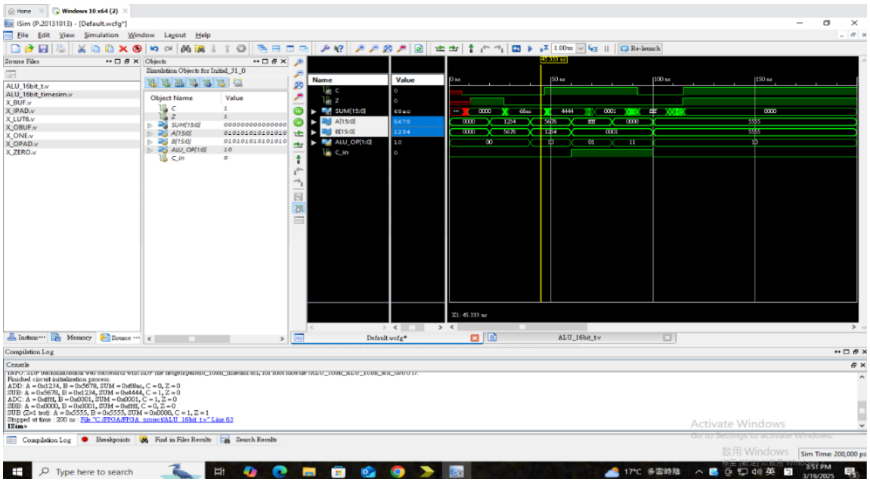





## Behavior:




## Post-Route:



# VHDL

```
module ALU_16b(
    input [15:0] dataA, dataB,
    input [1:0] sel,
    input Cin,
    output [15:0] Sum,
    output C,
    output Z, N, V
);
    wire [15:0] B;
    wire cin;

    wire [14:0] cout;

    assign cin = (sel == 2'b10) ? 1'b1 : //SUB
                 (sel == 2'b11) ? ~Cin : //SBB
                 (sel == 2'b01) ? Cin :  //ADC
                 1'b0;
    genvar i;
    generate
        for (i = 0; i < 16; i = i + 1) begin : gen_xor
            xor (B[i], dataB[i], sel[1]);
        end
    endgenerate
    full_adder fa0 (.A(dataA[0]), .B(B[0]), .Cin(cin),      .Sum(Sum[0]), .Cout(cout[0]));
    full_adder fa1 (.A(dataA[1]), .B(B[1]), .Cin(cout[0]), .Sum(Sum[1]), .Cout(cout[1]));
    full_adder fa2 (.A(dataA[2]), .B(B[2]), .Cin(cout[1]), .Sum(Sum[2]), .Cout(cout[2]));
    full_adder fa3 (.A(dataA[3]), .B(B[3]), .Cin(cout[2]), .Sum(Sum[3]), .Cout(cout[3]));
    full_adder fa4 (.A(dataA[4]), .B(B[4]), .Cin(cout[3]), .Sum(Sum[4]), .Cout(cout[4]));
    full_adder fa5 (.A(dataA[5]), .B(B[5]), .Cin(cout[4]), .Sum(Sum[5]), .Cout(cout[5]));
    full_adder fa6 (.A(dataA[6]), .B(B[6]), .Cin(cout[5]), .Sum(Sum[6]), .Cout(cout[6]));
    full_adder fa7 (.A(dataA[7]), .B(B[7]), .Cin(cout[6]), .Sum(Sum[7]), .Cout(cout[7]));
    full_adder fa8 (.A(dataA[8]), .B(B[8]), .Cin(cout[7]), .Sum(Sum[8]), .Cout(cout[8]));
    full_adder fa9 (.A(dataA[9]), .B(B[9]), .Cin(cout[8]), .Sum(Sum[9]), .Cout(cout[9]));
    full_adder fa10(.A(dataA[10]),.B(B[10]),.Cin(cout[9]),.Sum(Sum[10]),.Cout(cout[10]));
    full_adder fa11(.A(dataA[11]),.B(B[11]),.Cin(cout[10]),.Sum(Sum[11]),.Cout(cout[11]));
    full_adder fa12(.A(dataA[12]),.B(B[12]),.Cin(cout[11]),.Sum(Sum[12]),.Cout(cout[12]));
    full_adder fa13(.A(dataA[13]),.B(B[13]),.Cin(cout[12]),.Sum(Sum[13]),.Cout(cout[13]));
    full_adder fa14(.A(dataA[14]),.B(B[14]),.Cin(cout[13]),.Sum(Sum[14]),.Cout(cout[14]));
    full_adder fa15(.A(dataA[15]),.B(B[15]),.Cin(cout[14]),.Sum(Sum[15]),.Cout(C));
    assign Z = (Sum == 16'b0)?1'b1:1'b0;
    assign N = Sum[15];
    assign V = (dataA[15] & B[15] & ~Sum[15]) | (~dataA[15] & ~B[15] & Sum[15]);
endmodule
```

## Testbench:

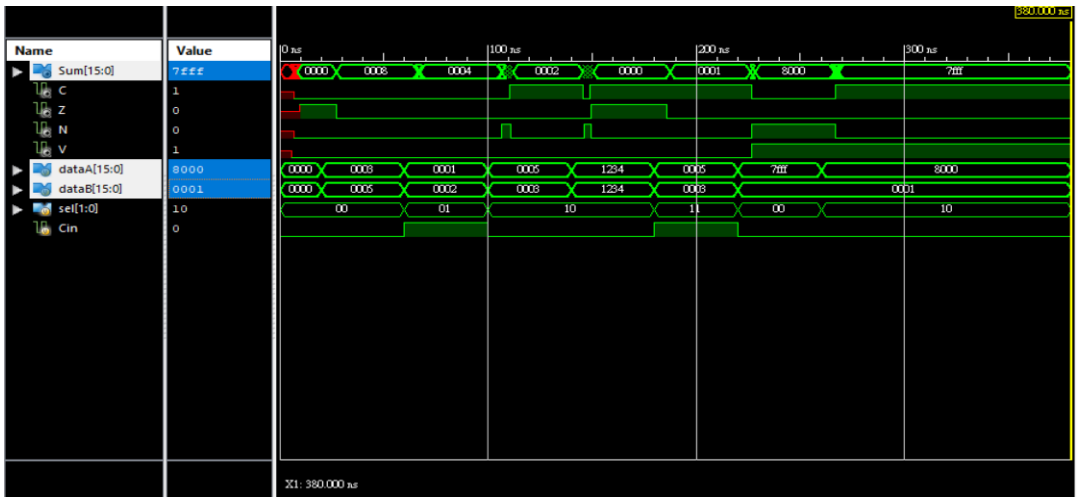
```
`timescale 1ns / 1ps
module ALU_16b_t;

    // Inputs
    reg [15:0] dataA;
    reg [15:0] dataB;
    reg [1:0] sel;
    reg Cin;

    // Outputs
    wire [15:0] Sum;
    wire C;
    wire Z;
    wire N;
    wire V;

    // Instantiate the Unit Under Test (UUT)
    ALU_16b uut (
        .dataA(dataA),
        .dataB(dataB),
        .sel(sel),
        .Cin(Cin),
        .Sum(Sum),
        .C(C),
        .Z(Z),
        .N(N),
        .V(V)
    );

    initial begin
        // Initialize Inputs
        dataA = 0;
        dataB = 0;
        sel = 0;
        Cin = 0;
        $display("Time\tsel\tCin\tA\tB\t=> Sum\t C Z N V");
```

```
        sel = 0;
        Cin = 0;
        $display("Time\tsel\tCin\tA\tB\t=> Sum\t C Z N V");

        // 測試 ADD：A + B
        #20 dataA = 16'h0003; dataB = 16'h0005; sel = 2'b00; Cin = 0;
        #20 $display("%t\t%2b\t%b\t%h\t%h\t=> %h\t%b %b %b %b", $time, sel, Cin, dataA, dataB, Sum, C, Z, N, V);

        // 測試 ADC：A + B + 1
        #20 dataA = 16'h0001; dataB = 16'h0002; sel = 2'b01; Cin = 1;
        #20 $display("%t\t%2b\t%b\t%h\t%h\t=> %h\t%b %b %b %b", $time, sel, Cin, dataA, dataB, Sum, C, Z, N, V);

        // 測試 SUB：A - B
        #20 dataA = 16'h0005; dataB = 16'h0003; sel = 2'b10; Cin = 0;
        #20 $display("%t\t%2b\t%b\t%h\t%h\t=> %h\t%b %b %b %b", $time, sel, Cin, dataA, dataB, Sum, C, Z, N, V);

        // 測試 SUB 為 0
        #20 dataA = 16'h1234; dataB = 16'h1234; sel = 2'b10; Cin = 0;
        #20 $display("%t\t%2b\t%b\t%h\t%h\t=> %h\t%b %b %b %b", $time, sel, Cin, dataA, dataB, Sum, C, Z, N, V);

        // 測試 SBB：A - B - 1
        #20 dataA = 16'h0005; dataB = 16'h0003; sel = 2'b11; Cin = 1;
        #20 $display("%t\t%2b\t%b\t%h\t%h\t=> %h\t%b %b %b %b", $time, sel, Cin, dataA, dataB, Sum, C, Z, N, V);

        // 測試 overflow：正數 + 正數 => 負數
        #20 dataA = 16'h7FFF; dataB = 16'h0001; sel = 2'b00; Cin = 0;
        #20 $display("%t\t%2b\t%b\t%h\t%h\t=> %h\t%b %b %b %b", $time, sel, Cin, dataA, dataB, Sum, C, Z, N, V);

        // 測試負數減正數 => 更負
        #20 dataA = 16'h8000; dataB = 16'h0001; sel = 2'b10; Cin = 0;
        #20 $display("%t\t%2b\t%b\t%h\t%h\t=> %h\t%b %b %b %b", $time, sel, Cin, dataA, dataB, Sum, C, Z, N, V);
        #100;
        $finish;
    end

endmodule
```

## Behavior:

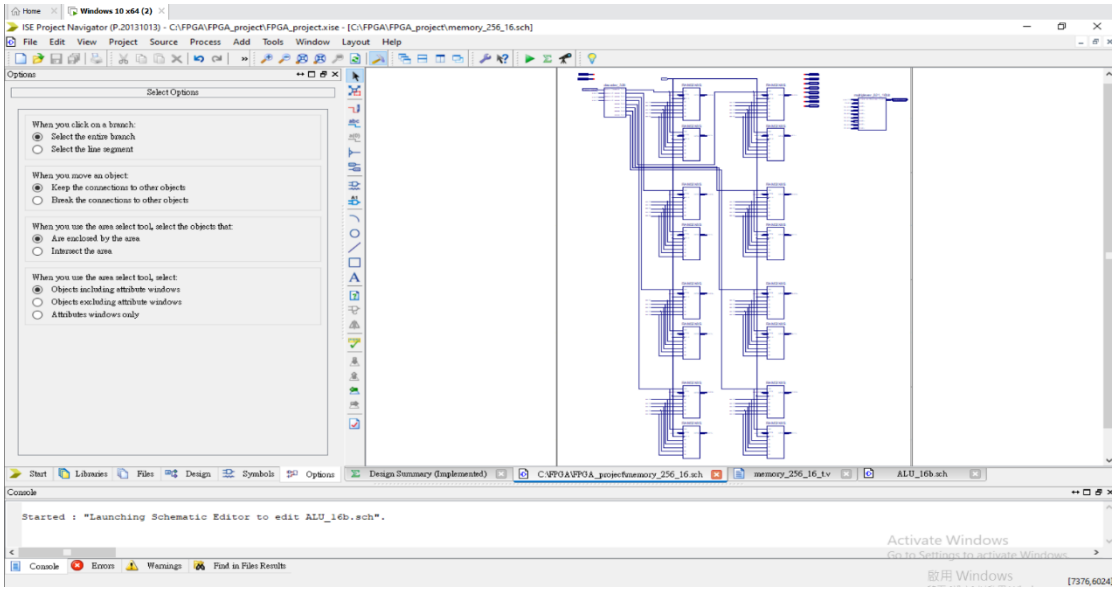
```
Simulator is doing circuit initialization process.
Time        sel     Cin     A       B       => Sum      C Z N V
Finished circuit initialization process.
    20000   00      0       0003    0005    => 0008     0 0 0 0
    40000   01      1       0001    0002    => 0004     0 0 0 0
    60000   10      0       0005    0003    => 0002     1 0 0 0
    80000   10      0       1234    1234    => 0000     1 1 0 0
   100000   11      1       0005    0003    => 0001     1 0 0 0
   120000   00      0       7fff    0001    => 8000     0 0 1 1
   140000   10      0       8000    0001    => 7fff     1 0 0 1
```

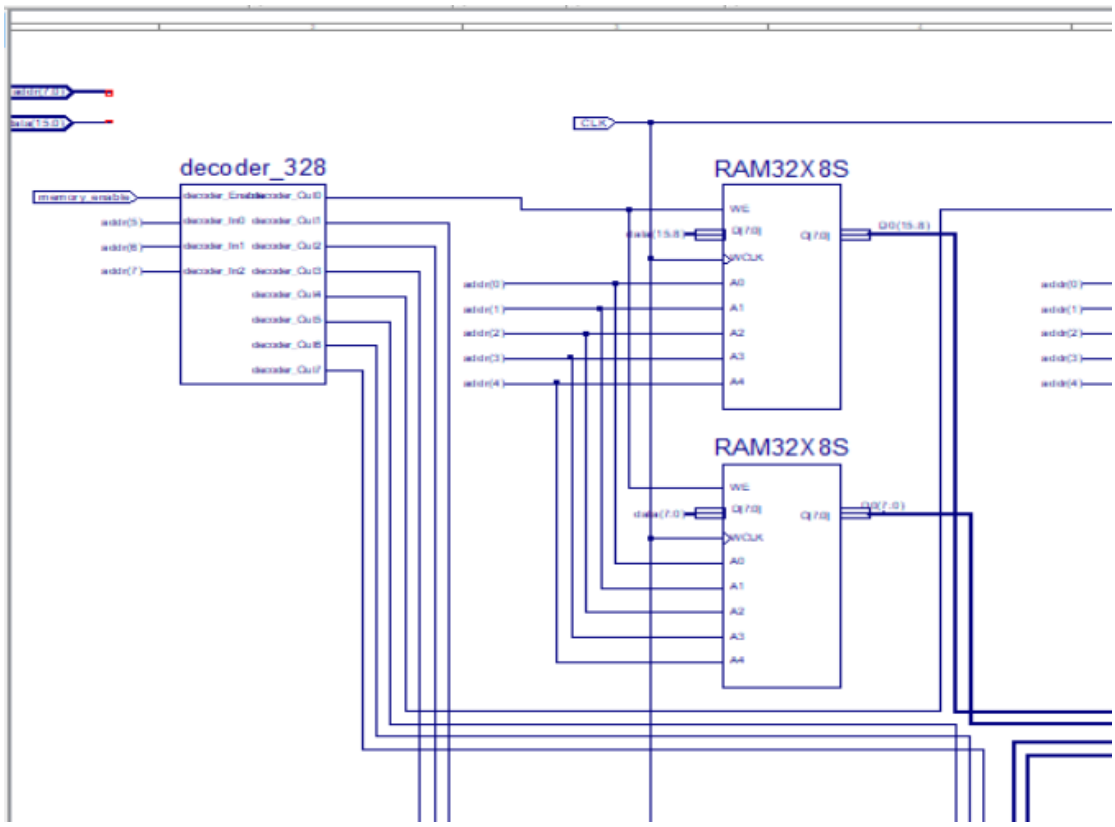
## Post-Route:

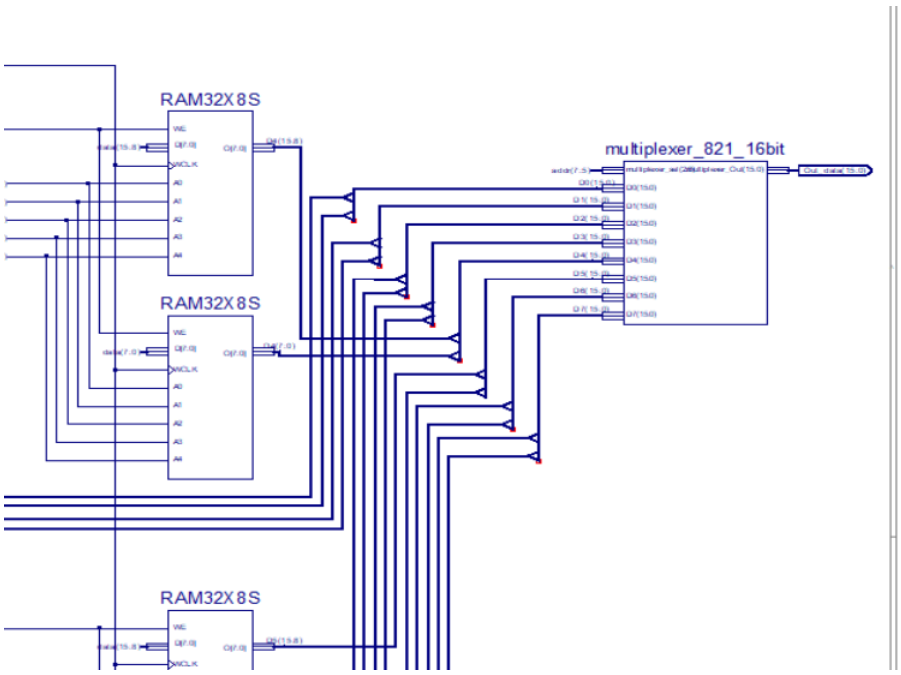


# 7. 256x16 memory

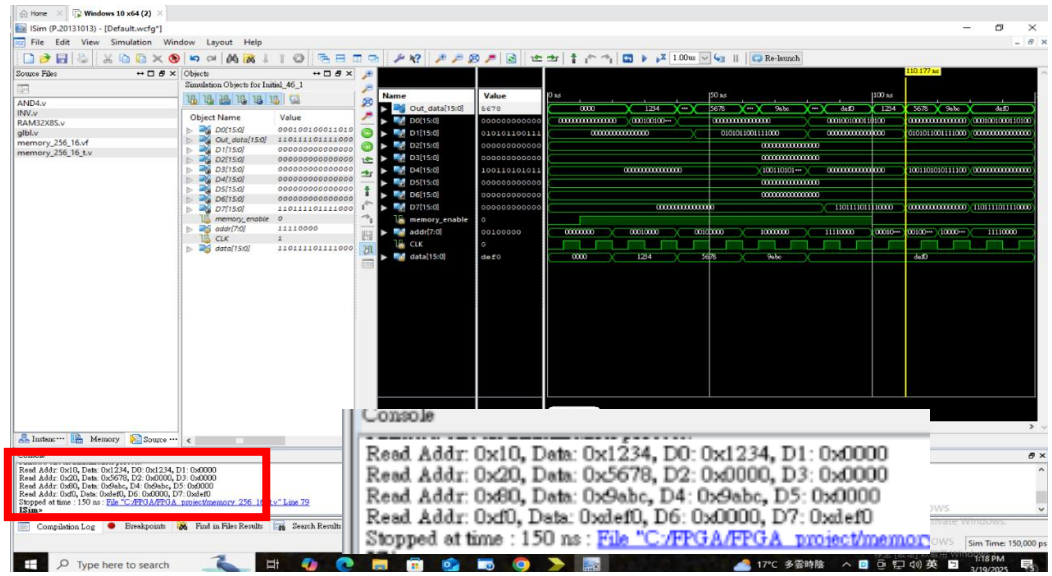
## Schematic



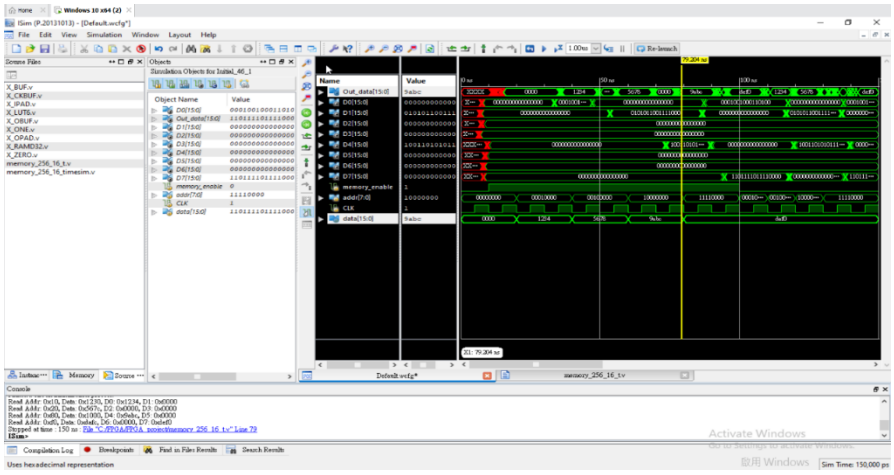






**Behavior:**




**Post-Route:**



# VHDL

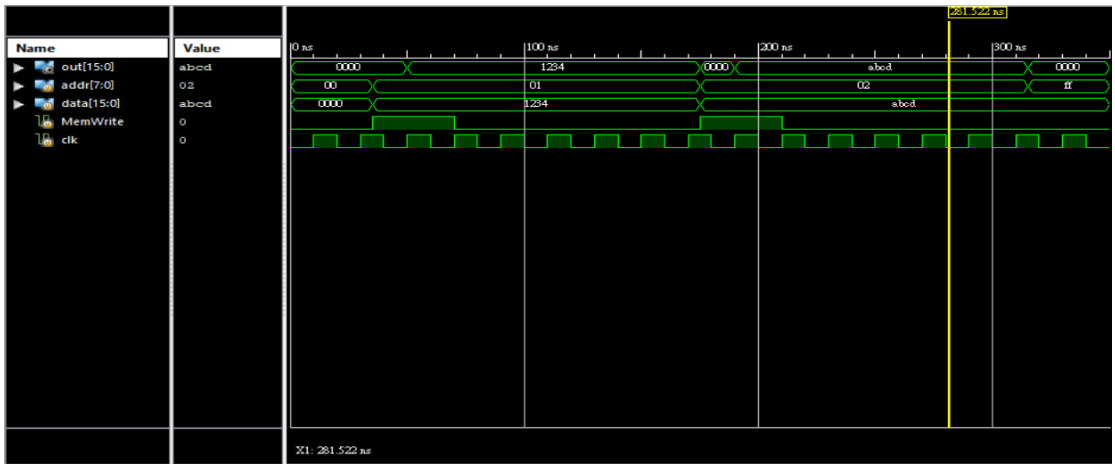
```
`timescale 1ns / 1ps
module MEM_256to16(
    input [7:0]addr,
    input [15:0]data,
    input MemWrite,clk,
    output [15:0]out
    );

    reg [15:0]mem[0:255];
    assign out = mem[addr];
    always @(posedge clk)
        if(MemWrite) mem[addr] <= data;
endmodule
```

**Testbench:**

```
`timescale 1ns / 1ps
module MEM_256to16_t;
   // Inputs
   reg [7:0] addr;
   reg [15:0] data;
   reg MemWrite;
   reg clk;
   // Outputs
   wire [15:0] out;
   // Instantiate the Unit Under Test (UUT)
   MEM_256to16 uut (
      .addr(addr),
      .data(data),
      .MemWrite(MemWrite),
      .clk(clk),
      .out(out)
   );
   always begin
       #22.5 clk = ~clk;
   end
   initial begin
      // Initialize Inputs
      addr = 0;
      data = 0;
      MemWrite = 0;
      clk = 0;
      // === Write 0x1234 to address 0x01 ===
      #45 addr = 8'h01; data = 16'h1234;MemWrite = 1;
      #45  // write occurs here
      MemWrite = 0;
      #45;
      addr = 8'h01;
      #45
      $display("READ @01 = %h (expected: 1234)", out);
      // === Write 0xABCD to address 0x02 ===
      #45 addr = 8'h02;data = 16'hABCD;MemWrite = 1;
```

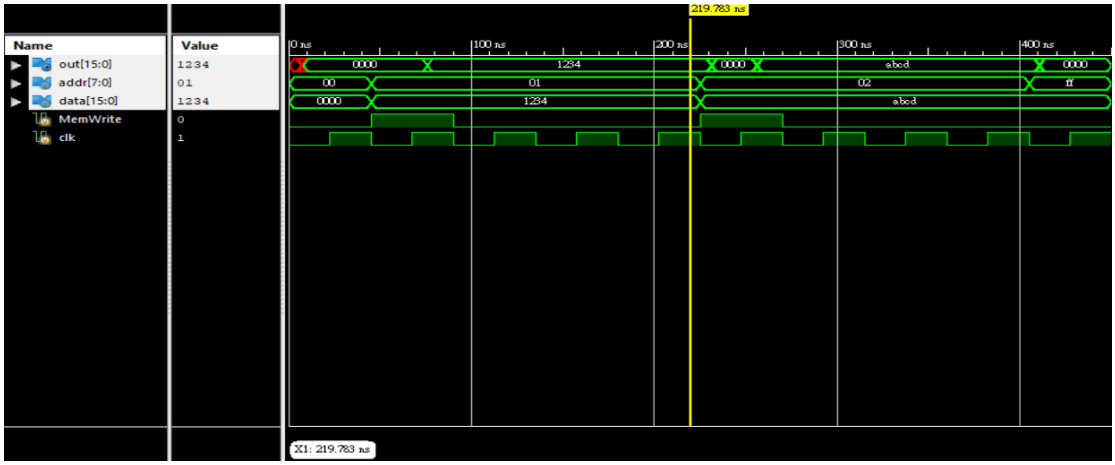
```
        #45 addr = 8'h02;data = 16'hABCD;MemWrite = 1;
        #45
        MemWrite = 0;
        #45
        addr = 8'h02;
        #45
        $display("READ @02 = %h (expected: ABCD)", out);
        // === Read uninitialized address 0xFF ===
        #45
        addr = 8'hFF;
        #45
        $display("READ @FF = %h (expected: 0000)", out);
        $finish;
    end
endmodule
```

**Behavior:**



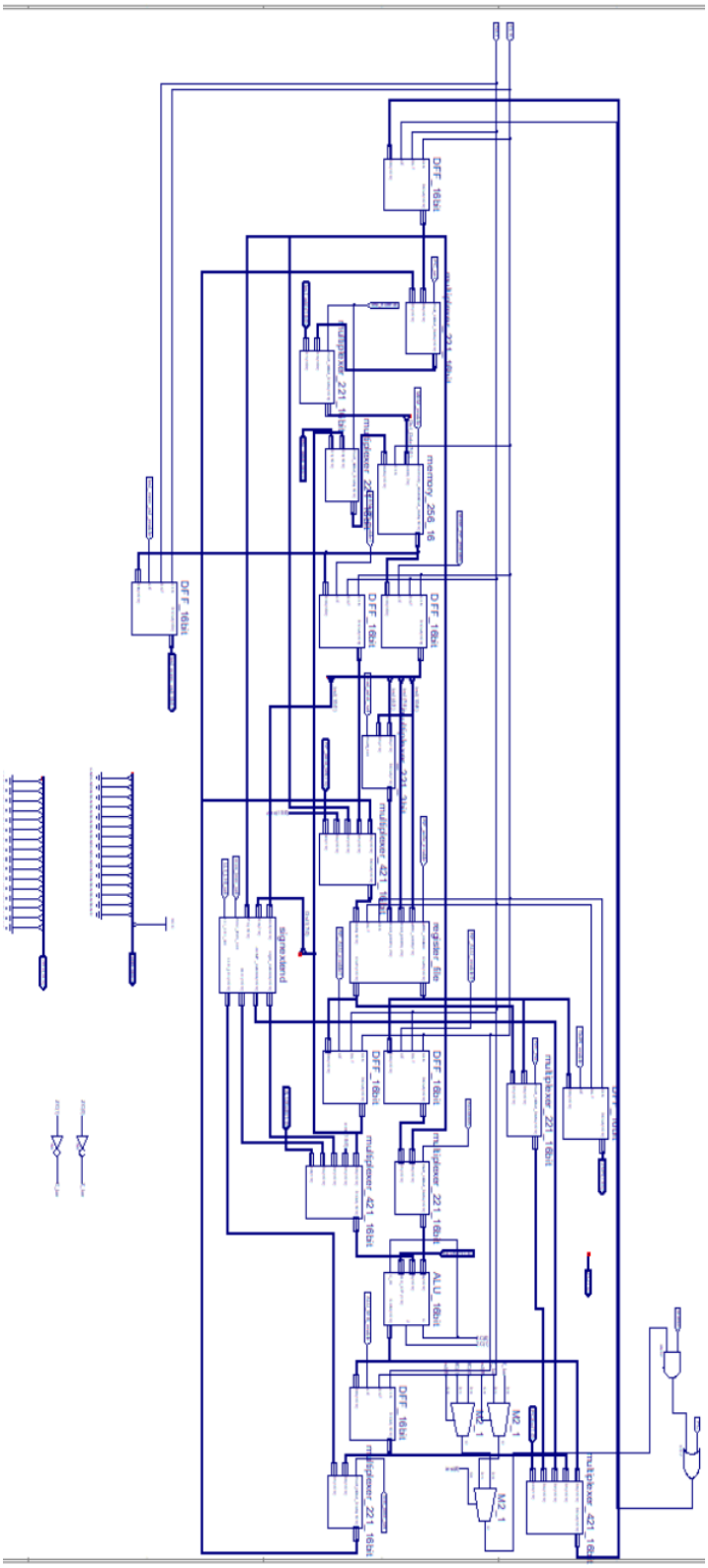

```
Finished circuit initialization process.
READ @01 = 1234 (expected: 1234)
READ @02 = abcd (expected: ABCD)
READ @FF = 0000 (expected: 0000)
```

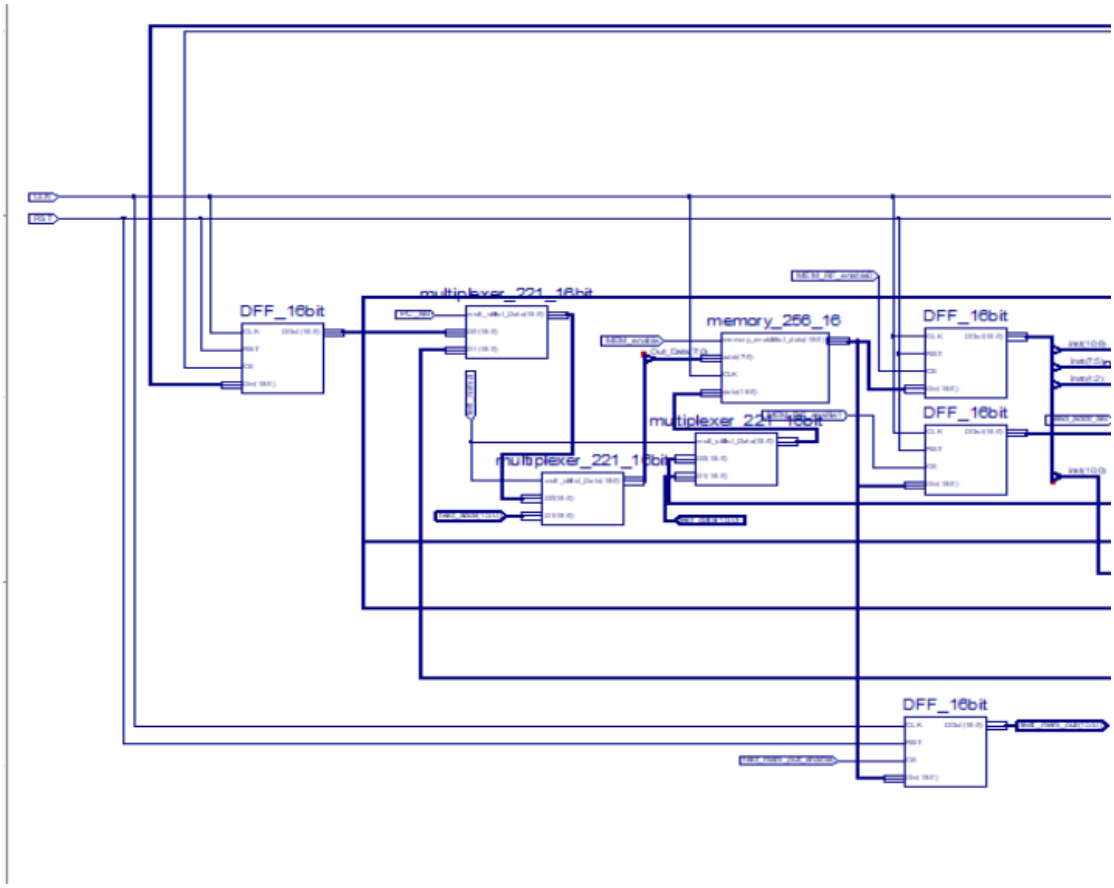
**Post-Route:**

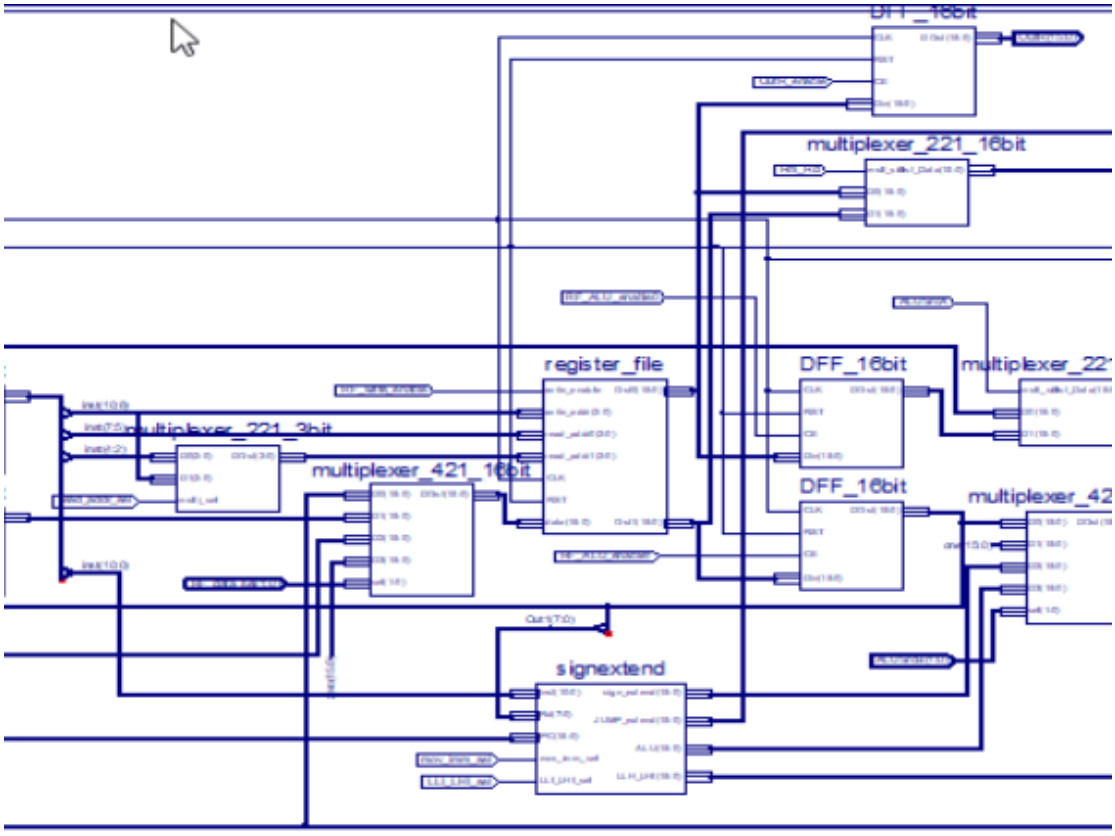


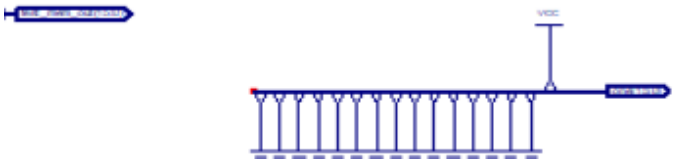
# 8. Complete Data path

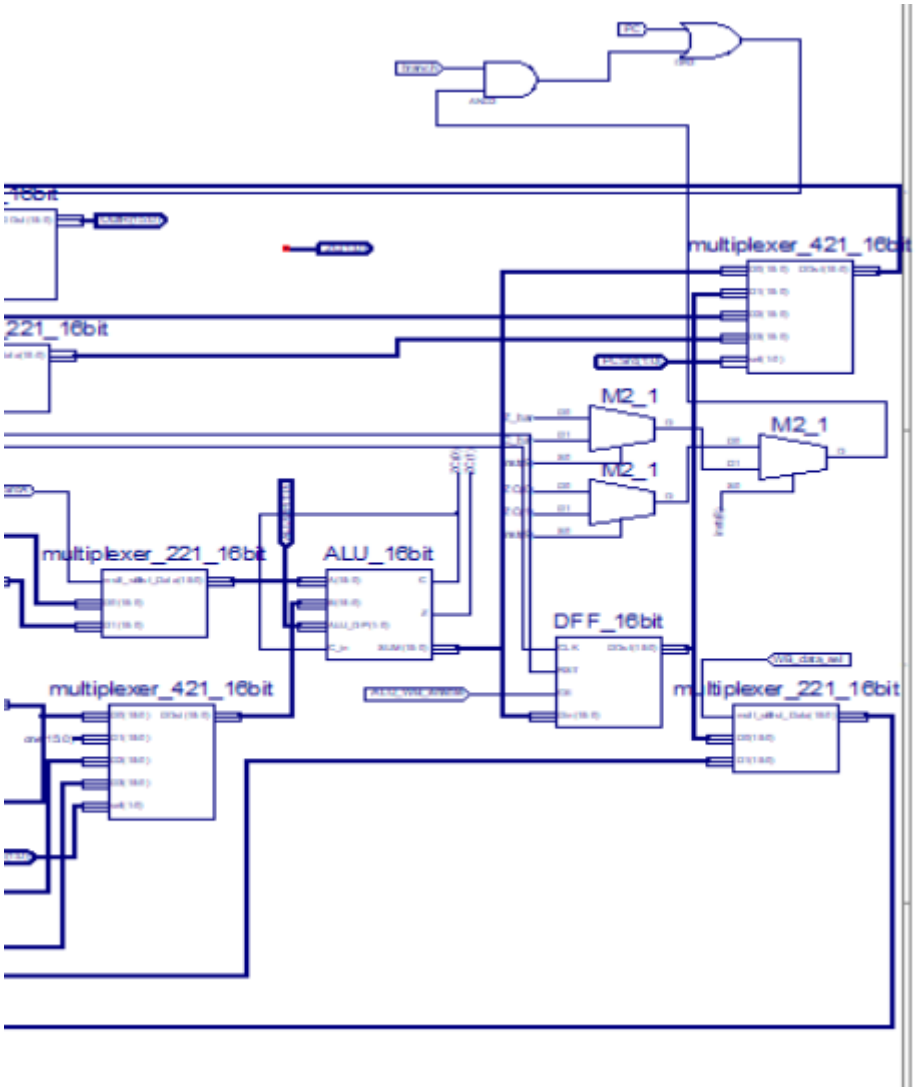
**Schematic**




DFF_16bit
multiplexer_221_16bit
memory_256_16
DFF_16bit
DFF_16bit
multiplexer_221_16bit
multiplexer_221_16bit
DFF_16bit


DFF_16bit
multiplexer_221_16bit
register_file
DFF_16bit
multiplexer_221_3bit
multiplexer_421_16bit
DFF_16bit
signextend


VCC




## Simulation

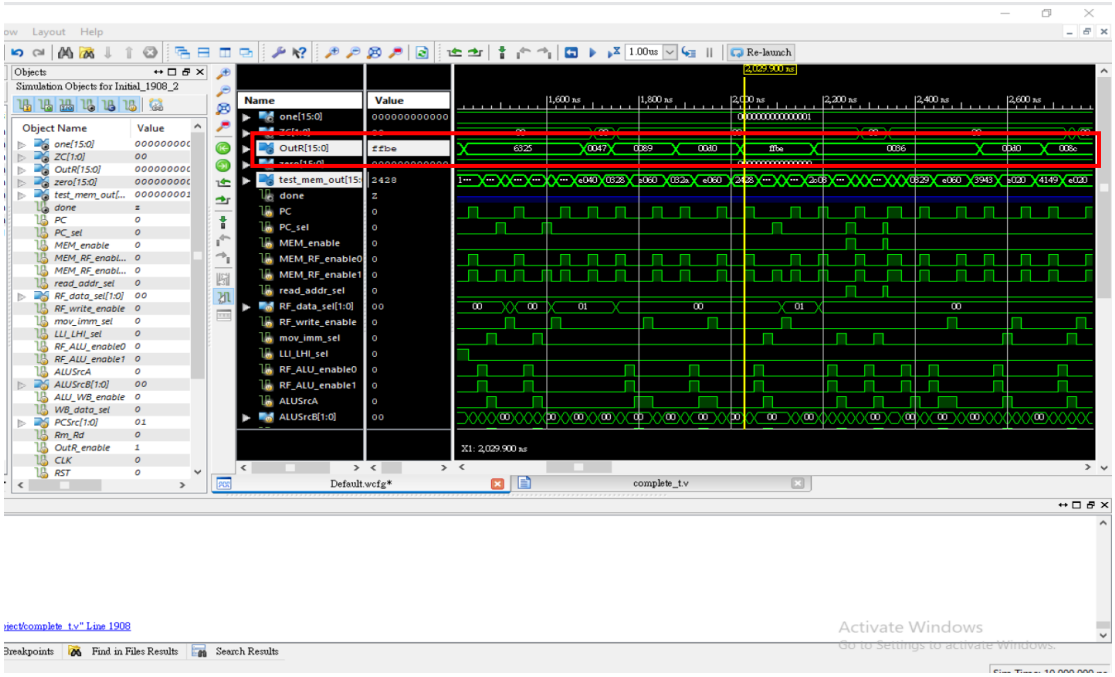
除了老師給的測試範例 我另外寫了其他的範例用 test 隔出來，也有將應該輸出的值標示。

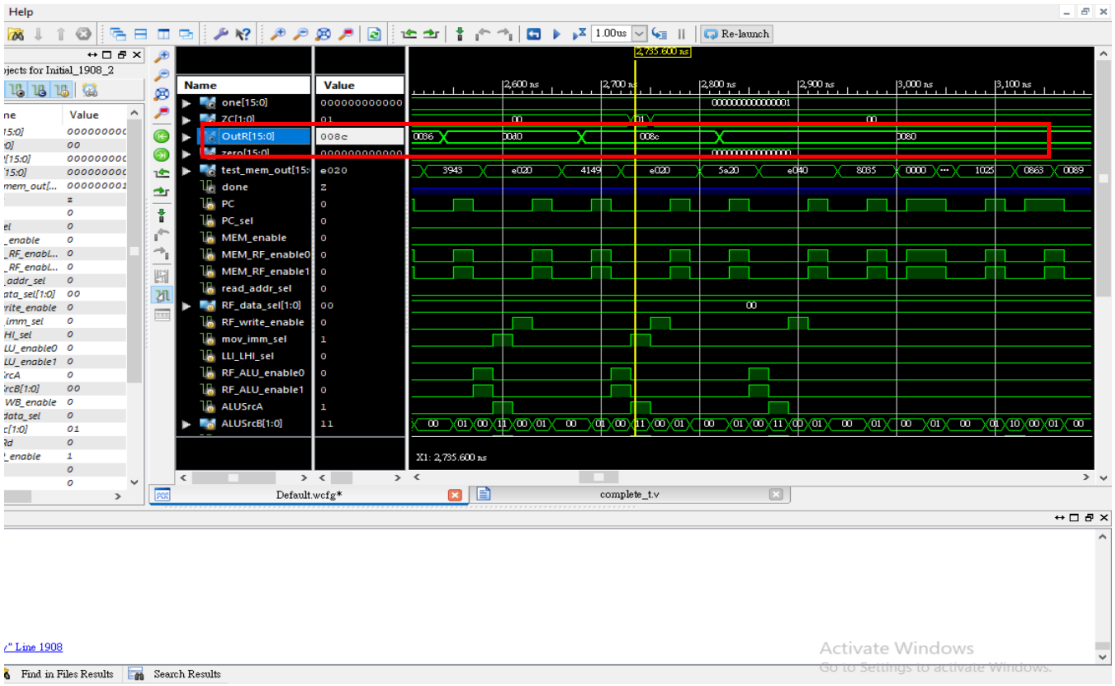
```
write_mem(16'h0,16'b0001_0000_0010_0101); // LLI R0,#25
write_mem(16'h1,16'b0000_1000_0110_0011); // LHI R0,#63
write_mem(16'h2,16'b1110_0000_0000_0000); // OUT R0 (6325H)
write_mem(16'h3,16'b0001_1001_0000_0000); // LDR R1,R0,#0
write_mem(16'h4,16'b0001_1010_0000_0001); // LDR R2,R0,#1
write_mem(16'h5,16'b1110_0000_0010_0000); // OUT R1 (47H)
write_mem(16'h6,16'b1110_0000_0100_0000); // OUT R2 (89H)
write_mem(16'h7,16'b0000_0011_0010_1000); // ADD R3,R1,R2
write_mem(16'h8,16'b1110_0000_0110_0000); // OUT R3 (D0H)
write_mem(16'h9,16'b0000_0011_0010_1010); // SUB R3,R1,R2
write_mem(16'hA,16'b1110_0000_0110_0000); // OUT R3 (FFBEH)
///////////////test///////////////
write_mem(16'hB,16'b0010_0100_0010_1000); // LDR R4,R1,R2
```

```
write_mem(16'hC,16'b1110_0000_1000_0000); // OUT R4 (36H)
write_mem(16'hD,16'b0010_1100_0000_0011); // STR R4 R0 #3(16'h28 = 36H)
write_mem(16'hE,16'b0011_0011_0010_1000); // STR R3 R1 R2(16'hD0 = FFBEH)
write_mem(16'hF,16'b0011_0000_0010_1001); // CMP R1 R2
write_mem(16'h10,16'b0000_0011_0010_1001); // ADC R3 R1 R2
write_mem(16'h11,16'b1110_0000_0110_0000); // OUT R3 (D0H)
write_mem(16'h12,16'b0011_1001_0100_0011); // ADDI R1,R2 #3
write_mem(16'h13,16'b1110_0000_0010_0000); // OUT R1 (8CH)
write_mem(16'h14,16'b0100_0001_0100_1001); // SUBI R1,R2 #9
write_mem(16'h15,16'b1110_0000_0010_0000); // OUT R1 (80H)
write_mem(16'h16,16'b0101_1010_0010_0000); // MOV R2 R1
write_mem(16'h17,16'b1110_0000_0100_0000); // OUT R2 (80H)
write_mem(16'h18,16'b1000_0000_0011_0101); // JMP pc = 16'h35
write_mem(16'h35,16'b1110_0000_0100_0000); // OUT R2 (80H)
write_mem(16'h40,16'b1100_1110_0000_0010); // B[AL] pc = pc + 3 (to pc:16)
write_mem(16'h43,16'b1110_0000_0100_0000); // OUT R2 (80H)
write_mem(16'h25,16'h47); // data (25h, 47h)
write_mem(16'h26,16'h89); // data (26h, 89h)
write_mem(16'hD0,16'h36); // data (D0h, 39h)
```

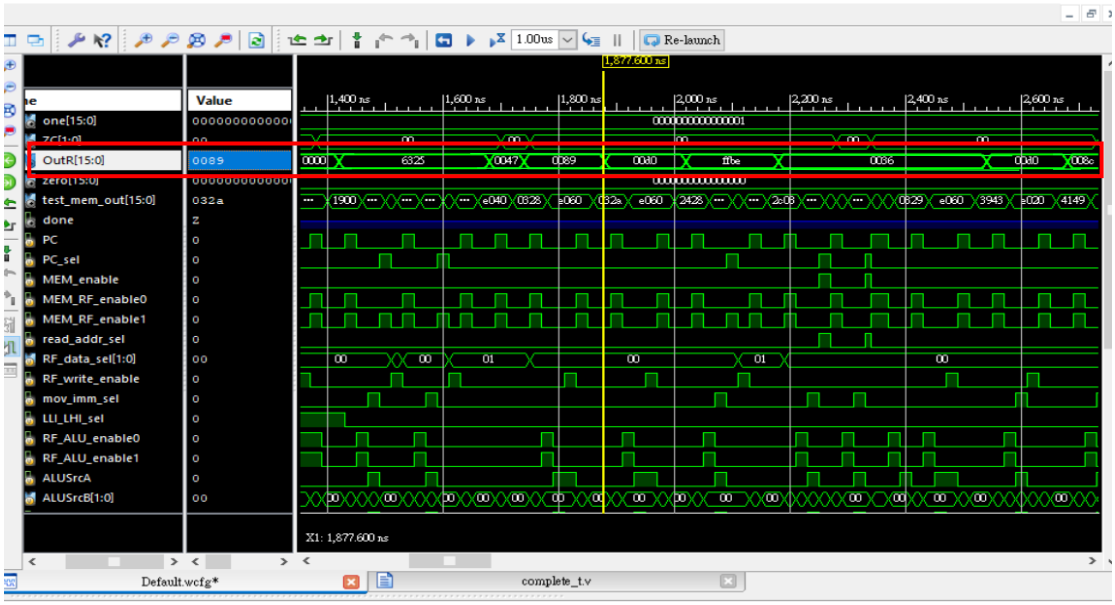
**Behavior:**

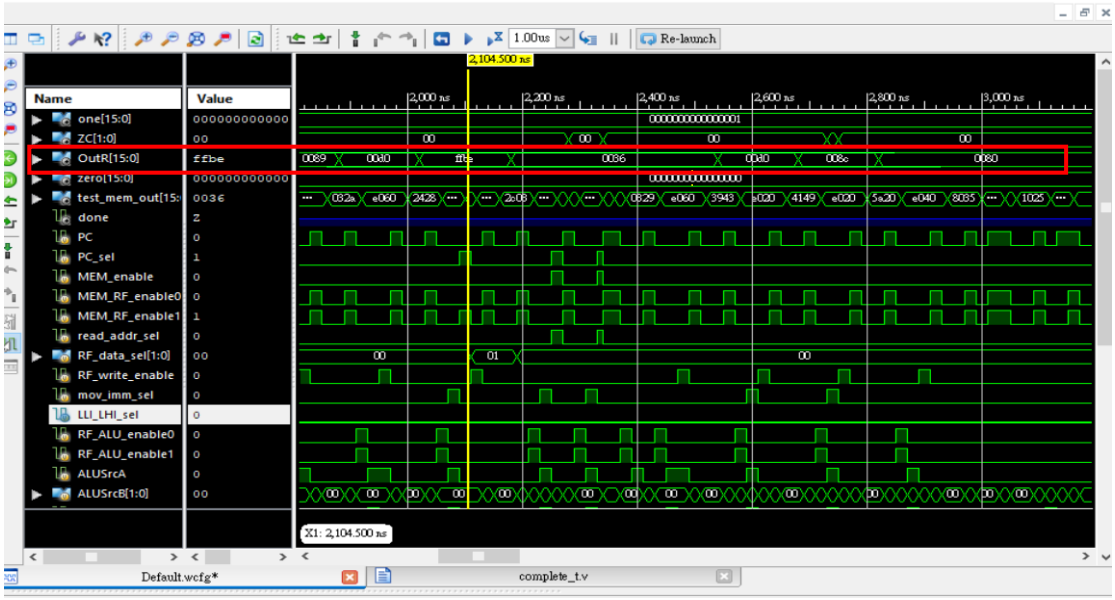
6325->0047->0089->00D0->FFBE->0036->00D0->008C->0080




Help
Re-launch
Name
Value
one[15:0]
ZC[1:0]
OutR[15:0]
zero[15:0]
test_mem_out[15:
done
PC
PC_sel
MEM_enable
MEM_RF_enable0
MEM_RF_enable1
read_addr_sel
RF_data_sel[1:0]
RF_write_enable
mov_imm_sel
LLI_LHI_sel
RF_ALU_enable0
RF_ALU_enable1
ALUSrcA
ALUSrcB[1:0]
X1: 2,755.600 ns
Default.wcfg*
complete_t.v
Find in Files Results
Search Results
Activate Windows

## Post-Route:


Re-launch
Value
one[15:0]
ZC[1:0]
OutR[15:0]
zero[15:0]
test_mem_out[15:0]
done
PC
PC_sel
MEM_enable
MEM_RF_enable0
MEM_RF_enable1
read_addr_sel
RF_data_sel[1:0]
RF_write_enable
mov_imm_sel
LLI_LHI_sel
RF_ALU_enable0
RF_ALU_enable1
ALUSrcA
ALUSrcB[1:0]
X1: 1,877.600 ns
Default.wcfg*
complete_t.v




# VHDL

```
1   `timescale 1ns / 1ps
2   module datapath(
3       input rst_n,clk,
4       input MemWrite      ,
5       input IRWrite       ,
6       input Imm_5or8      ,
7       input RegWrite      ,
8       input OutREn        ,
9       input [1:0]ALUop    ,
10      input Branch        ,
11      input PSWEn         ,
12      input PCWrite       ,
13      input IorD          ,
14      input RegDst        ,
15      input [1:0]MemtoReg ,
16      input LLorLH        ,
17      input ALUSrcA       ,
18      input [1:0]ALUSrcB  ,
19      input JAorJR        ,
20      input [1:0]PCSrc    ,
21      input [15:0]ext_addr,
22      input [15:0]ext_data,
23      input test,
24      input TestMem,
25      output [15:0]OutR,
26      output [15:0]TestMemout,
27      output [15:0]opcode
28      );
29      wire [15:0]tomemaddr;
30      wire [15:0]tomemdata;
31      wire [15:0]PCout;
32      wire [15:0]pcnext;
33      wire PCEn;
34      wire [15:0]IorDout;
35      wire [15:0]adr;
36      wire [15:0]ALUout;
```

```
36      wire [15:0]ALUout;
37      wire [15:0]Memout;
38      wire [15:0]Instr;
39      wire [15:0]Data;
40      wire [2:0]Rn;
41      wire [15:0]LLLHout;
42      wire [15:0]toreg;
43      wire [15:0]Regout0;
44      wire [15:0]Regout1;
45      wire [15:0]RA;
46      wire [15:0]RB;
47      wire [15:0]OutR;
48      wire [15:0]SrcA;
49      wire [15:0]Signout;
50      wire [15:0]SrcB;
51      wire N_flag,V_flag;
52      wire tmpc,tmpz;
53      wire toALU;
54      wire[15:0]ALUResult;
55      wire [15:0]JAJRout;
56      //pc//
57      assign PCEn = (PCWrite | (Branch & ((Instr[8]) ? ((Instr[9]) ? ~C_flag:~Z_flag):((Instr[9]) ? C_flag:Z_flag))));
58      PC pc(.clk(clk),.rst_n(rst_n),.PC_enable(PCEn),.PC_next(pcnext),.PC_result(PCout));
59
60      //IorD//
61      MUX_2to1 mux2to10(.in0(PCout),.in1(ALUout),.select(IorD),.out(adr));
62
63      //mem//
64      //assign testmemwrite = test ? 1'b1:MemWrite;
65      MEM_256to16 mem256to16(.clk(clk),.addr(tomemaddr[7:0]),.data(tomemdata),.MemWrite(MemWrite),.out(Memout));
66
71      DFF_16b dff1(.clk(clk),.rst_n(rst_n),.data(Memout),.DFF_enable(1'b1),.out(Data));
72
73      //regdst · memtoreg//
74      MUX_2to1_3b mux2to13b(.in0(Instr[4:2]),.in1(Instr[10:8]),.select(RegDst),.out(Rn));
75
76      MUX_4to1 mux4to10(.in0(ALUout),.in1(Data),.in2(LLLHout),.in3(PCout),.select(MemtoReg),.out(toreg));
77
78      //registerfile//
79      RF_16b rf(.clk(clk),.rst_n(rst_n),.read_register0(Instr[7:5]),.read_register1(Rn),.write_register(Instr[10:8]),.write_data(toreg),.RegWrite(RegWrite),.read_data0(Regout0),.read_data1(Regout1));
80
81      //LLLH//
82      MUX_2to1 mux2to12(.in0({8'h0,Instr[7:0]}),.in1({Instr[7:0],RB[7:0]}),.select(LLorLH),.out(LLLHout));
83
84      //RA RB//
85      DFF_16b dff2(.clk(clk),.rst_n(rst_n),.data(Regout0),.DFF_enable(1'b1),.out(RA));
86      DFF_16b dff3(.clk(clk),.rst_n(rst_n),.data(Regout1),.DFF_enable(1'b1),.out(RB));
87
88      //outr//
89      DFF_16b dff4(.clk(clk),.rst_n(rst_n),.data(Regout0),.DFF_enable(OutREn),.out(OutR));
90
91      //ALUSrcA/B//
92      MUX_2to1 mux2to11(.in0(PCout),.in1(RA),.select(ALUSrcA),.out(SrcA));
93      MUX_4to1 mux4to11(.in0(RB),.in1(16'b1),.in2(Signout),.in3(16'b0),.select(ALUSrcB),.out(SrcB));
94
95      //signextend//
96      SignExtend se(.imm_5b(Instr[4:0]),.imm_8b(Instr[7:0]),.imm_5or8(Imm_5or8),.out(Signout));
97
98      //ALU//
99      wire C_flag,Z_flag;
100     DFF_16b dff7(.clk(clk),.rst_n(rst_n),.data(tmpz),.DFF_enable(PSWEn),.out(Z_flag));
101     DFF_16b dff8(.clk(clk),.rst_n(rst_n),.data(tmpc),.DFF_enable(PSWEn),.out(C_flag));
102     MUX_2to1_1b mux2to11b(.in0(1'b0),.in1(tmpc),.select(ALUop[0]),.out(toALU));
103     ALU_16b alu16b(.dataA(SrcA),.dataB(SrcB),.Cin(toALU),.sel(ALUop),.Sum(ALUResult),.C(tmpc),.Z(tmpz),.N(N_flag),.V(V_flag));
104     DFF_16b dff5(.clk(clk),.rst_n(rst_n),.data(ALUResult),.DFF_enable(1'b1),.out(ALUout));
105
106     //JAorJR · pcsrc//
107     MUX_4to1 mux4to12(.in0(ALUResult),.in1(ALUout),.in2({PCout[15:11],Instr[10:0]}),.in3(JAJRout),.select(PCSrc),.out(pcnext));
108     MUX_2to1 mux2to13(.in0(RA),.in1(RB),.select(JAorJR),.out(JAJRout));
109
110     //testmem&data//
111     MUX_2to1 mux2to14(.in0(adr),.in1(ext_addr),.select(test),.out(tomemaddr));
112     MUX_2to1 mux2to15(.in0(RB),.in1(ext_data),.select(test),.out(tomemdata));
113
114 endmodule
115
```

**Testbench:**

這次的 **testbench** 跟期中測試的測資一樣。

```
write_mem(16'h0,16'b0001_0000_0010_0101); //LLI R0,#25
write_mem(16'h1,16'b0000_1000_0110_0011); //LHI R0,#63
write_mem(16'h2,16'b1110_0000_0000_0000); //OUT R0 (6325H)
write_mem(16'h3,16'b0001_1001_0000_0000); // LDR R1,R0,#0
write_mem(16'h4,16'b0001_1010_0000_0001); // LDR R2,R0,#1
write_mem(16'h5,16'b1110_0000_0010_0000); // OUT R1 (47H)
write_mem(16'h6,16'b1110_0000_0100_0000); // OUT R2 (89H)
write_mem(16'h7,16'b0000_0011_0010_1000); // ADD R3,R1,R2
write_mem(16'h8,16'b1110_0000_0110_0000); // OUT R3 (D0H)
write_mem(16'h9,16'b0000_0011_0010_1010); // SUB R3,R1,R2
write_mem(16'hA,16'b1110_0000_0110_0000); // OUT R3 (FFBEH)
write_mem(16'hB,16'b0010_0100_0010_1000); // LDR R4,R1,R2
write_mem(16'hC,16'b1110_0000_1000_0000); // OUT R4 (39H)
write_mem(16'hD,16'b0010_1100_0000_0011); // STR R4 R0 #3(16'h28 = 39H)
```

```
write_mem(16'hE,16'b0011_0011_0010_1000); // STR R3 R1 R2(16'hD0 = FFBEH)
write_mem(16'hF,16'b0011_0000_0010_1001); // CMP R1 R2
write_mem(16'h10,16'b0000_0011_0010_1001); // ADC R3 R1 R2
write_mem(16'h11,16'b1110_0000_0110_0000); // OUT R3 (D0H)
write_mem(16'h12,16'b1100_1110_0000_0011); // B[AL] (to pc:16)
write_mem(16'h16,16'b1110_0000_0100_0000); // OUT R2 (89H)
write_mem(16'h17,16'b0011_1001_0100_0011); // ADDI R1,R2 #3
write_mem(16'h18,16'b1110_0000_0010_0000); // OUT R1 (8CH)
write_mem(16'h19,16'b0100_0001_0100_1001); // SUBI R1,R2 #9
write_mem(16'h1A,16'b1110_0000_0010_0000); // OUT R1 (80H)
write_mem(16'h1B,16'b0101_1010_0010_0000); // MOV R2 R1
write_mem(16'h1C,16'b1110_0000_0100_0000); // OUT R2 (80H)
write_mem(16'h1D,16'b0000_0000_0010_1011); // SBB R0,R1,R2
write_mem(16'h1E,16'b1100_0000_0000_0011); // BEQ pc = pc + 3(to pc:22)
write_mem(16'h22,16'b1110_0000_0110_0000); // OUT R3 (D0H)
write_mem(16'h23,16'b1000_0000_0011_0101); // JMP pc = 16'h35
write_mem(16'h35,16'b1110_0000_0100_0000); // OUT R2 (80H)
write_mem(16'h25,16'h47); // data (25h, 47h)
write_mem(16'h26,16'h89); // data (26h, 89h)
write_mem(16'hD0,16'h39); // data (D0h, 39h)
```

```
1  timescale 1ns / 1ps
2  module datapath_t;
3     // Inputs
4     parameter clk_period = 20;
5     parameter delay_factor = 2;
6     reg rst_n;
7     reg clk;
8
9     reg MemWrite;
10    reg IRWrite;
11    reg Imm_5or8;
12    reg RegWrite;
13    reg OutREn;
14    reg [1:0] ALUop;
15    reg Branch;
16    reg PCWrite;
17    reg IorD;
18    reg RegDst;
19    reg [1:0] MemtoReg;
20    reg LLorLH;
21    reg ALUSrcA;
22    reg [1:0] ALUSrcB;
23    reg JAorJR;
24    reg [1:0] PCSrc;
25    reg [15:0] ext_addr;
26    reg [15:0] ext_data;
27    reg test;
28    reg TestMem;
29    reg PSWEn;
30    // Outputs
31    wire [15:0] OutR;
32    wire [15:0] TestMemout;
33    wire [15:0] opcode;
34    integer i;
35    // Instantiate the Unit Under Test (UUT)
36    datapath uut (
```

```
36      datapath uut (
37          .rst_n(rst_n),
38          .clk(clk),
39
40          .MemWrite(MemWrite),
41          .IRWrite(IRWrite),
42          .Imm_5or8(Imm_5or8),
43          .RegWrite(RegWrite),
44          .OutREn(OutREn),
45          .ALUop(ALUop),
46          .Branch(Branch),
47          .PSWEn(PSWEn),
48          .PCWrite(PCWrite),
49          .IorD(IorD),
50          .RegDst(RegDst),
51          .MemtoReg(MemtoReg),
52          .LLorLH(LLorLH),
53          .ALUSrcA(ALUSrcA),
54          .ALUSrcB(ALUSrcB),
55          .JAorJR(JAorJR),
56          .PCSrc(PCSrc),
57          .ext_addr(ext_addr),
58          .ext_data(ext_data),
59          .test(test),
60          .TestMem(TestMem),
61          .OutR(OutR),
62          .TestMemout(TestMemout),
63          .opcode(opcode)
64      );
65      always begin
66          #(clk_period/2)clk <= 1'b0;
67          #(clk_period/2)clk <= 1'b1;
68      end
69      initial begin
70          // Initialize Inputs
71          rst_n = 0;
```

```
71          rst_n = 0;
72          clk = 0;
73
74          MemWrite = 0;
75          IRWrite = 0;
76          Imm_5or8 = 0;
77          RegWrite = 0;
78          OutREn = 0;
79          ALUop = 0;
80          Branch = 0;
81          PSWEn = 0;
82          PCWrite = 0;
83          IorD = 0;
84          RegDst = 0;
85          MemtoReg = 0;
86          LLorLH = 0;
87          ALUSrcA = 0;
88          ALUSrcB = 0;
89          JAorJR = 0;
90          PCSrc = 0;
91          ext_addr = 0;
92          ext_data = 0;
93          test = 0;
94          TestMem = 0;
95
96          rst_n <= 1'b0;test = 1'b0;
97          repeat (2) @(posedge clk)
98                      #(clk_period/delay_factor)rst_n <= 1'b0;
99          rst_n <= 1'b1;test = 1'b1;
100
101
102         write_mem(16'h0,16'b0001_0000_0010_0101); //LLI R0,#25
103         write_mem(16'h1,16'b0000_1000_0110_0011); //LHI R0,#63
104         write_mem(16'h2,16'b1110_0000_0000_0000); //OUT R0 (6325H)
105
106         write_mem(16'h3,16'b0001_1001_0000_0000); // LDR R1,R0,#0
```

```
106        write_mem(16'h3,16'b0001_1001_0000_0000); // LDR R1,R0,#0
107        write_mem(16'h4,16'b0001_1010_0000_0001); // LDR R2,R0,#1
108        write_mem(16'h5,16'b1110_0000_0010_0000); // OUT R1 (47H)
109        write_mem(16'h6,16'b1110_0000_0100_0000); // OUT R2 (89H)
110        write_mem(16'h7,16'b0000_0011_0010_1000); // ADD R3,R1,R2
111        write_mem(16'h8,16'b1110_0000_0110_0000); // OUT R3 (D0H)
112        write_mem(16'h9,16'b0000_0011_0010_1010); // SUB R3,R1,R2
113        write_mem(16'hA,16'b1110_0000_0110_0000); // OUT R3 (FFBEH)
114
115        ///////////////test///////////////
116        write_mem(16'hB,16'b0010_0100_0010_1000); // LDR R4,R1,R2
117        write_mem(16'hC,16'b1110_0000_1000_0000); // OUT R4 (39H)
118        write_mem(16'hD,16'b0010_1100_0000_0011); // STR R4 R0 #3(16'h28 = 39H)
119        write_mem(16'hE,16'b0011_0011_0010_1000); // STR R3 R1 R2(16'hD0 = FFBEH)
120        write_mem(16'hF,16'b0011_0000_0010_1001); // CMP R1 R2
121
122        write_mem(16'h10,16'b0000_0011_0010_1001); // ADC R3 R1 R2
123        write_mem(16'h11,16'b1110_0000_0110_0000); // OUT R3 (D0H)
124
125        write_mem(16'h12,16'b1100_1110_0000_0011); // B[AL] pc = pc + 3 (to pc:16)
126        write_mem(16'h16,16'b1110_0000_0100_0000); // OUT R2 (89H)
127
128        write_mem(16'h17,16'b0011_1001_0100_0011); // ADDI R1,R2 #3
129        write_mem(16'h18,16'b1110_0000_0010_0000); // OUT R1 (8CH)
130        write_mem(16'h19,16'b0100_0001_0100_1001); // SUBI R1,R2 #9
131        write_mem(16'h1A,16'b1110_0000_0010_0000); // OUT R1 (80H)
132
133        write_mem(16'h1B,16'b0101_1010_0010_0000); // MOV R2 R1
134        write_mem(16'h1C,16'b1110_0000_0100_0000); // OUT R2 (80H)
135        write_mem(16'h1D,16'b0000_0000_0010_1011); // SBB R0,R1,R2
136        write_mem(16'h1E,16'b1100_0000_0000_0011); // BEQ pc = pc + 3(to pc:22)
137        write_mem(16'h22,16'b1110_0000_0110_0000); // OUT R3 (D0H)
138        write_mem(16'h23,16'b1000_0000_0011_0101); // JMP pc = 16'h35
139        write_mem(16'h35,16'b1110_0000_0100_0000); // OUT R2 (80H)
140
141        write_mem(16'h25,16'h47); // data (25h, 47h)
```

```
141        write_mem(16'h25,16'h47); // data (25h, 47h)
142        write_mem(16'h26,16'h89); // data (26h, 89h)
143        write_mem(16'hD0,16'h39); // data (D0h, 39h)
144
145        @(posedge clk)#(clk_period/delay_factor) MemWrite = 1'b0;
146         test = 1'b1;
147        for(i = 0 ; i < 14 ; i = i + 1)
148           @(posedge clk) #(clk_period/delay_factor) ext_addr = i;
149         test = 1'b0;
150        rst_n = 1'b1;
151        repeat (2) @(posedge clk)
152                   #(clk_period/delay_factor) rst_n = 1'b0;
153        rst_n = 1'b1;
154        TestMem = 1;
155        //LLI R0,#25//
156        @(posedge clk)
157        #10 test = 0; IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
158        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
159        @(posedge clk)
160        #10 PCWrite = 0; IRWrite = 0;
161        @(posedge clk)
162        #10 LLorLH = 0; MemtoReg = 2'b10; RegWrite = 1;
163
164        //LHI R0,#63//
165        @(posedge clk)
166        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
167        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
168        @(posedge clk)
169        #10 PCWrite = 0; IRWrite = 0;
170        @(posedge clk)
171        #10 RegDst = 1;
172        @(posedge clk)
173        #10 LLorLH = 1; MemtoReg = 2'b10; RegWrite = 1;
174
175        //OUT R0 (6325H)//
176        @(posedge clk)
```

```
176        @(posedge clk)
177        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
178        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
179        @(posedge clk)
180        #10 PCWrite = 0; IRWrite = 0;
181        @(posedge clk)
182        #10 OutREn = 1;
183
184        // LDR R1,R0,#0
185        @(posedge clk)
186        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
187        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
188        @(posedge clk)
189        #10 PCWrite = 0; IRWrite = 0;
190        @(posedge clk)
191        #10 ALUSrcA = 1; ALUSrcB = 2'b10;ALUop = 2'b00;Imm_5or8 = 0;
192        @(posedge clk)
193        #10 IorD = 1;
194        @(posedge clk)
195        #10 MemtoReg = 2'b01; RegWrite = 1;
196
197
198        // LDR R2,R0,#1
199        @(posedge clk)
200        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
201        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
202        @(posedge clk)
203        #10 PCWrite = 0; IRWrite = 0;
204        @(posedge clk)
205        #10 ALUSrcA = 1; ALUSrcB = 2'b10;ALUop = 2'b00;Imm_5or8 = 0;
206        @(posedge clk)
207        #10 IorD = 1;
208        @(posedge clk)
209        #10 MemtoReg = 2'b01; RegWrite = 1;
210        // OUT R1 (47H)
211        @(posedge clk)
```

```
211        @(posedge clk)
212        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
213        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
214        @(posedge clk)
215        #10 PCWrite = 0; IRWrite = 0;
216        @(posedge clk)
217        #10 OutREn = 1;
218
219        // OUT R2 (89H)
220        @(posedge clk)
221        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
222        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
223        @(posedge clk)
224        #10 PCWrite = 0; IRWrite = 0;
225        @(posedge clk)
226        #10 OutREn = 1;
227        // ADD R3,R1,R2
228        @(posedge clk)
229        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
230        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
231        @(posedge clk)
232        #10 RegDst = 0; PCWrite = 0; IRWrite = 0;
233        @(posedge clk)
234        #10 ALUSrcA = 1;ALUSrcB = 2'b00; ALUop = 2'b00;
235        @(posedge clk)
236        #10 MemtoReg = 2'b00; RegWrite = 1;
237
238        // OUT R3 (D0H)
239        @(posedge clk)
240        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
241        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
242        @(posedge clk)
243        #10 PCWrite = 0; IRWrite = 0;
244        @(posedge clk)
245        #10 OutREn = 1;
246        // SUB R3,R1,R2
```

```
246        // SUB R3,R1,R2
247        @(posedge clk)
248        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
249        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
250        @(posedge clk)
251        #10 RegDst = 0; PCWrite = 0; IRWrite = 0;
252        @(posedge clk)
253        #10 ALUSrcA = 1;ALUSrcB = 2'b00; ALUop = 2'b10;
254        @(posedge clk)
255        #10 MemtoReg = 2'b00; RegWrite = 1;
256        // OUT R3 (FFBEH)
257        @(posedge clk)
258        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
259        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
260        @(posedge clk)
261        #10 PCWrite = 0; IRWrite = 0;
262        @(posedge clk)
263        #10 OutREn = 1;
264
265        // LDR R4,R1,R2
266        @(posedge clk)
267        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
268        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
269        @(posedge clk)
270        #10 PCWrite = 0; IRWrite = 0;
271        @(posedge clk)
272        #10 ALUSrcA = 1; ALUSrcB = 2'b00;ALUop = 2'b00;Imm_5or8 = 0;
273        @(posedge clk)
274        #10 IorD = 1;
275        @(posedge clk)
276        #10 MemtoReg = 2'b01; RegWrite = 1;
277        // OUT R4 (39H)
278        @(posedge clk)
279        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
280        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
281        @(posedge clk)
```

```
281        @(posedge clk)
282        #10 PCWrite = 0; IRWrite = 0;
283        @(posedge clk)
284        #10 OutREn = 1;
285        // STR R4 R0 #3(16'h28 = 39H)
286        @(posedge clk)
287        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
288        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
289        @(posedge clk)
290        #10 IRWrite = 0;RegDst = 1; PCWrite = 0;
291        @(posedge clk)
292        #10 ALUSrcA = 1;ALUSrcB = 2'b10;ALUop = 2'b00;Imm_5or8 = 0;
293        @(posedge clk)
294        #10 IorD = 1;MemWrite = 1;
295        // STR R3 R1 R2(16'hD0 = FFBEH)
296        @(posedge clk)
297        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
298        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
299        @(posedge clk)
300        #10 IRWrite = 0;RegDst = 0; PCWrite = 0;
301        @(posedge clk)
302        #10 RegDst = 1; ALUSrcA = 1;ALUSrcB = 2'b00;ALUop = 2'b00;
303        @(posedge clk)
304        #10 IorD = 1;MemWrite = 1;
305        // CMP R1 R2
306        @(posedge clk)
307        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
308        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
309        @(posedge clk)
310        #10 IRWrite = 0; PCWrite = 0; RegDst = 0;
311        @(posedge clk)
312        #10 ALUSrcA = 1;ALUSrcB = 2'b00; ALUop = 2'b10; PSWEn = 1;
313
314        // ADC R3 R1 R2
315        @(posedge clk)
316        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
```

```
316        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
317        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
318        @(posedge clk)
319        #10 RegDst = 0; PCWrite = 0; IRWrite = 0;
320        @(posedge clk)
321        #10 ALUSrcA = 1;ALUSrcB = 2'b00; ALUop = 2'b01;PSWEn = 1;
322        @(posedge clk)
323        #10 MemtoReg = 2'b00; RegWrite = 1; PSWEn = 0;
324
325        // OUT R3 (D0H)
326        @(posedge clk)
327        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
328        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
329        @(posedge clk)
330        #10 PCWrite = 0; IRWrite = 0;
331        @(posedge clk)
332        #10 OutREn = 1;
333
334        // B[AL] pc = pc + 3 (to pc:16)
335        @(posedge clk)
336        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
337        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
338        @(posedge clk)
339        #10 ALUSrcA = 0; ALUSrcB = 2'b10; Imm_5or8 = 1; ALUop = 2'b00; IRWrite = 0; PCWrite = 0;
340        @(posedge clk)
341        #10 PCSrc = 2'b01; Branch = 1; PCWrite = 1;
342        // OUT R2 (89H)
343        @(posedge clk)
344        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
345        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
346        @(posedge clk)
347        #10 PCWrite = 0; IRWrite = 0;
348        @(posedge clk)
349        #10 OutREn = 1;
350
351        // ADDI R1,R2 #3
```

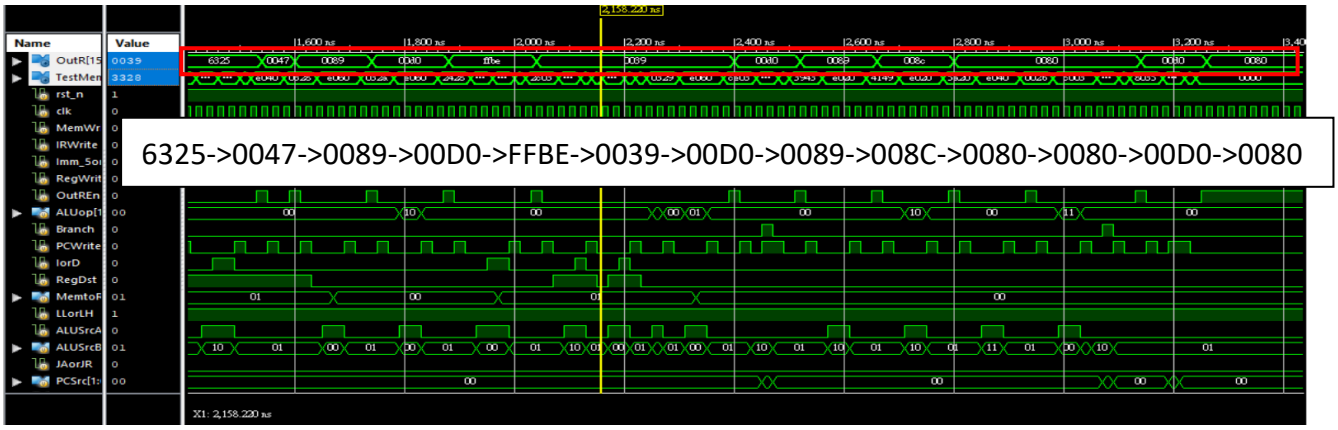
```
351        // ADDI R1,R2 #3
352        @(posedge clk)
353        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
354        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
355        @(posedge clk)
356        #10 PCWrite = 0; IRWrite = 0;
357        @(posedge clk)
358        #10 ALUSrcA = 1;ALUSrcB = 2'b10; Imm_5or8 = 0 ; ALUop = 2'b00;
359        @(posedge clk)
360        #10 MemtoReg = 2'b00; RegWrite = 1; PSWEn = 0;
361        // OUT R1 (8CH)
362        @(posedge clk)
363        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
364        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
365        @(posedge clk)
366        #10 PCWrite = 0; IRWrite = 0;
367        @(posedge clk)
368        #10 OutREn = 1;
369        // SUBI R1,R2 #9
370        @(posedge clk)
371        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
372        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
373        @(posedge clk)
374        #10 PCWrite = 0; IRWrite = 0;
375        @(posedge clk)
376        #10 ALUSrcA = 1;ALUSrcB = 2'b10; Imm_5or8 = 0 ; ALUop = 2'b10;
377        @(posedge clk)
378        #10 MemtoReg = 2'b00; RegWrite = 1; PSWEn = 0;
379        // OUT R1 (80H)
380        @(posedge clk)
381        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
382        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
383        @(posedge clk)
384        #10 PCWrite = 0; IRWrite = 0;
385        @(posedge clk)
386        #10 OutREn = 1;
```

```
386        #10 OutREn = 1;
387
388        // MOV R2 R1
389        @(posedge clk)
390        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
391        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
392        @(posedge clk)
393        #10 IRWrite = 0; PCWrite = 0;
394        @(posedge clk)
395        #10 ALUSrcA = 1; ALUSrcB = 2'b11; ALUop = 2'b00;
396        @(posedge clk)
397        #10 MemtoReg = 2'b00; RegWrite = 1;
398        // OUT R2 (80H)
399        @(posedge clk)
400        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
401        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
402        @(posedge clk)
403        #10 PCWrite = 0; IRWrite = 0;
404        @(posedge clk)
405        #10 OutREn = 1;
406
407        // SBB R0,R1,R2
408        @(posedge clk)
409        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
410        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
411        @(posedge clk)
412        #10 RegDst = 0; PCWrite = 0; IRWrite = 0;
413        @(posedge clk)
414        #10 ALUSrcA = 1;ALUSrcB = 2'b00; ; ALUop = 2'b11; PSWEn = 1;
415        @(posedge clk)
416        #10 MemtoReg = 2'b00; RegWrite = 1; PSWEn = 0;
417        // BEQ pc = pc + 3(to pc:22)
418        @(posedge clk)
419        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
420        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
421        @(posedge clk)
```

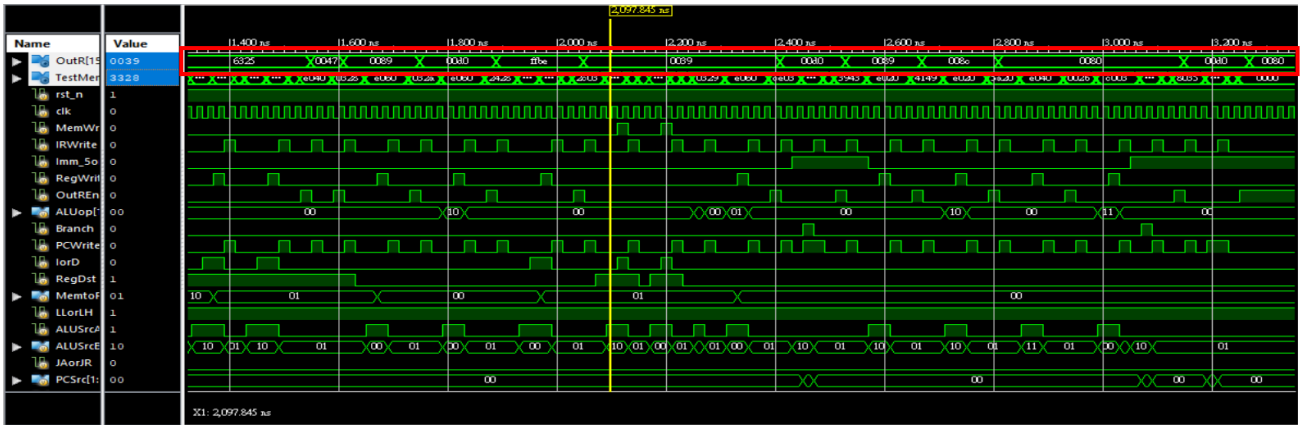
```
        @(posedge clk)
        #10 ALUSrcA = 0; ALUSrcB = 2'b10; Imm_5or8 = 1; ALUop = 2'b00; IRWrite = 0; PCWrite = 0;
        @(posedge clk)
        #10 PCSrc = 2'b01; Branch = 1;
        // OUT R0 (00H)
        @(posedge clk)
        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
        @(posedge clk)
        #10 PCWrite = 0; IRWrite = 0;
        @(posedge clk)
        #10 OutREn = 1;
        // JMP pc = 16'h35
        @(posedge clk)
        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
        @(posedge clk)
        #10 PCWrite = 0; IRWrite = 0;
        @(posedge clk)
        #10 PCSrc = 10;PCWrite = 1;
        // OUT R2 (80H)
        @(posedge clk)
        #10 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
        Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
        @(posedge clk)
        #10 PCWrite = 0; IRWrite = 0;
        @(posedge clk)
        #10 OutREn = 1;
    end
    task write_mem;
    input [15:0] addr,data;
    begin
        @(posedge clk)#(clk_period/delay_factor)begin
            test = 1'b1;
            MemWrite = 1;
            TestMem = 1'b1;ext_addr = addr;
```

```
            TestMem = 1'b1;ext_addr = addr;
            ext_data = data;
        end
    end
    endtask

    initial #3500 $finish;
    initial
        $monitor($realtime ,"ns %h %h %h %h %h %h %h %h %h \n ",clk,rst_n,TestMem,test,ext_addr,ext_data,TestMemout,OutR);


endmodule
```

**Behavior:**




**Post-Route:**



# 9. Controller(include instruction decoder)(期中未做出來)

**VHDL**

```
1   `timescale 1ns / 1ps
2   module controller(
3       input rst_n,clk              ,
4       input [15:0]opcode           ,
5       input start                  ,
6       output reg MemWrite          ,
7       output reg IRWrite           ,
8       output reg Imm_5or8          ,
9       output reg RegWrite          ,
10      output reg OutREn            ,
11      output reg [1:0]ALUop        ,
12      output reg Branch            ,
13
14      output reg PSWEn             ,
15      output reg PCWrite           ,
16      output reg IorD              ,
17      output reg RegDst            ,
18      output reg [1:0]MemtoReg     ,
19      output reg LLorLH            ,
20      output reg ALUSrcA           ,
21      output reg [1:0]ALUSrcB      ,
22      output reg JAorJR            ,
23      output reg [1:0]PCSrc        ,
24
25      output reg [4:0]state
26      );
27      //設定指令的opcode//
28      parameter LHI = 5'b00001          ;
29      parameter LLI = 5'b00010          ;
30      parameter LDR_imm = 5'b00011      ;
31      parameter LDR = 5'b00100          ;
32      parameter STR_imm = 5'b00101      ;
33      parameter STR = 5'b00110          ;
34      parameter ADD = 5'b00000          ;
35      parameter ADC = 5'b00000          ;
36      parameter SUB = 5'b00000          ;
```

```
36      parameter SUB = 5'b00000        ;
37      parameter SBB = 5'b00000        ;
38      parameter CMP = 5'b00110        ;
39      parameter ADDI = 5'b00111       ;
40      parameter SUBI = 5'b01000       ;
41      parameter MOV = 5'b01011        ;
42      parameter BCC = 8'b1100_0011    ;
43      parameter BCS = 8'b1100_0010    ;
44      parameter BNE = 8'b1100_0001    ;
45      parameter BEQ = 8'b1100_0000    ;
46      parameter BAL = 8'b1100_1110    ;
47      parameter JMP = 5'b10000        ;
48      parameter JAL_label = 5'b10001  ;
49      parameter JAL = 5'b10010        ;
50      parameter JR = 5'b10011         ;
51      parameter OutR = 5'b11100       ;
52
53
54
55    always @(posedge clk)begin
56       if(!rst_n || !start) state <= 5'd0;
57       else
58       begin
59           case(state)
60               5'd0:
61                   state <= 5'd1;
62               5'd1:
63                   if(opcode[15:11] == LHI || opcode[15:11] == LDR_imm || opcode[15:11] == STR_imm || opcode[15:11] == JR)begin
64                       state <= 5'd2;
65                   end
66                   else if(opcode[15:11] == LLI)begin
67                       state <= 5'd8;
68                   end
69                   else if(opcode[15:11] == LDR || (opcode[15:11] == STR && opcode[1:0] == 2'b00))begin
70                       state <= 5'd9;
71                   end
```

```
71                  end
72                  else if(opcode[15:11] == ADD && opcode[1:0] == 2'b00)begin
73                      state <= 5'd12;
74                  end
75                  else if(opcode[15:11] == ADC && opcode[1:0] == 2'b01)begin
76                      state <= 5'd14;
77                  end
78                  else if((opcode[15:11] == SUB && opcode[1:0] == 2'b10)|| (opcode[15:11] == CMP && opcode[1:0] == 2'b01))begin
79                      state <= 5'd15;
80                  end
81                  else if(opcode[15:11] == SBB && opcode[1:0] ==2'b11)begin
82                      state <= 5'd16;
83                  end
84                  else if(opcode[15:11] == ADDI)begin
85                      state <= 5'd17;
86                  end
87                  else if(opcode[15:11] == SUBI)begin
88                      state <= 5'd18;
89                  end
90                  else if(opcode[15:11] == MOV)begin
91                      state <= 5'd19;
92                  end
93                  else if(opcode[15:8] == BCC || opcode[15:8] == BCS || opcode[15:8] == BNE || opcode[15:8] == BEQ)begin
94                      state <= 5'd20;
95                  end
96                  else if(opcode[15:8] == BAL)begin
97                      state <= 5'd21;
98                  end
99                  else if(opcode[15:11] == JMP)begin
100                     state <= 5'd22;
101                 end
102                 else if(opcode[15:11] == JAL_label)begin
103                     state <= 5'd23;
104                 end
105                 else if(opcode[15:11] == JAL)begin
106                     state <= 5'd24;
```

```
106                         state <= 5'd24;
107                     end
108                     else if(opcode[15:11] == OutR)begin
109                         state <= 5'd25;
110                     end
111                 5'd2:
112                     if(opcode[15:11] == LHI)begin
113                         state <= 5'd3;
114                     end
115                     else if(opcode[15:11] == LDR_imm)begin
116                         state <= 5'd4;
117                     end
118                     else if(opcode[15:11] == STR_imm)begin
119                         state <= 5'd6;
120                     end
121                     else if(opcode[15:11] == JR)begin
122                         state <= 5'd7;
123                     end
124                 5'd4:
125                     state <= 5'd5;
126                 5'd9:
127                     if(opcode[15:11] == LDR)begin
128                         state <= 5'd10;
129                     end
130                     else if(opcode[15:11] == STR)begin
131                         state <= 5'd11;
132                     end
133                 5'd10:
134                     state <= 5'd5;
135                 5'd12,5'd14,5'd16,5'd17,5'd18,5'd19:
136                     state <= 5'd13;
137                 5'd15:
138                     if(opcode[15:11] == SUB)begin
139                         state <= 5'd13;
140                     end
141                     else if(opcode[15:11] == CMP)begin
```

```
141                 else if(opcode[15:11] == CMP)begin
142                     state <= 5'd0;
143                 end
144             default:
145                 state <= 5'd0;
146         endcase
147     end
148 end
149 always @(state)begin
150     Imm_5or8 = 0;
151     IorD = 0;
152     ALUSrcA = 0;
153     ALUSrcB = 2'b00;
154     ALUop = 2'b00;
155     PCSrc = 2'b00;
156     Branch = 0;
157     PCWrite = 0;
158     MemWrite = 0;
159     IRWrite = 0;
160     PSWEn = 0;
161     OutREn = 0;
162     RegWrite = 0;
163     RegDst = 0;
164     MemtoReg = 2'b00;
165     LLorLH = 0;
166     JAorJR = 0;
167     case(state)
168         5'd0:
169             begin
170                 IorD = 0; ALUSrcA = 0; ALUSrcB = 2'b01; ALUop = 2'b00; PCSrc = 2'b00;
171                 Branch = 0; PCWrite = 1; MemWrite = 0; IRWrite = 1; PSWEn = 0;OutREn = 0; RegWrite = 0;
172             end
173         5'd1:
174             begin
175                 Imm_5or8 = 1;IorD = 0;ALUSrcA = 0;ALUSrcB = 2'b10;ALUop = 2'b00;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
176                 MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
```

```
176                 MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
177             end
178         5'd2:
179             begin
180                 Imm_5or8 = 0;IorD = 0;ALUSrcA = 1;ALUSrcB = 2'b10;ALUop = 2'b00;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
181                 MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 0;RegDst = 1;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
182             end
183         5'd3:
184             begin
185                 Imm_5or8 = 0;IorD = 0;ALUSrcA = 0;ALUSrcB = 2'b00;ALUop = 2'b00;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
186                 MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 1;RegDst = 0;MemtoReg = 2'b10;LLorLH = 1;JAorJR = 0;
187             end
188         5'd4:
189             begin
190                 Imm_5or8 = 0;IorD = 1;ALUSrcA = 0;ALUSrcB = 2'b00;ALUop = 2'b00;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
191                 MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
192             end
193         5'd5:
194             begin
195                 Imm_5or8 = 0;IorD = 0;ALUSrcA = 0;ALUSrcB = 2'b00;ALUop = 2'b00;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
196                 MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 1;RegDst = 0;MemtoReg = 2'b01;LLorLH = 0;JAorJR = 0;
197             end
198         5'd6:
199             begin
200                 Imm_5or8 = 0;IorD = 1;ALUSrcA = 0;ALUSrcB = 2'b00;ALUop = 2'b00;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
201                 MemWrite = 1;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
202             end
203         5'd7:
204             begin
205                 Imm_5or8 = 0;IorD = 0;ALUSrcA = 0;ALUSrcB = 2'b00;ALUop = 2'b00;PCSrc = 2'b11;Branch = 0;PCWrite = 1;
206                 MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 1;
207             end
208         5'd8:
209             begin
210                 Imm_5or8 = 0;IorD = 0;ALUSrcA = 0;ALUSrcB = 2'b00;ALUop = 2'b00;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
211                 MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 1;RegDst = 0;MemtoReg = 2'b10;LLorLH = 0;JAorJR = 0;
```

```
211                 MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 1;RegDst = 0;MemtoReg = 2'b10;LLorLH = 0;JAorJR = 0;
212             end
213         5'd9:
214             begin
215                 Imm_5or8 = 0;IorD = 0;ALUSrcA = 1;ALUSrcB = 2'b00;ALUop = 2'b00;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
216                 MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 0;RegDst = 1;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
217             end
218         5'd10:
219             begin
220                 Imm_5or8 = 0;IorD = 1;ALUSrcA = 0;ALUSrcB = 2'b00;ALUop = 2'b00;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
221                 MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
222             end
223         5'd11:
224             begin
225                 Imm_5or8 = 0;IorD = 1;ALUSrcA = 0;ALUSrcB = 2'b00;ALUop = 2'b00;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
226                 MemWrite = 1;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
227             end
228         5'd12:
229             begin
230                 Imm_5or8 = 0;IorD = 0;ALUSrcA = 1;ALUSrcB = 2'b00;ALUop = 2'b00;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
231                 MemWrite = 0;IRWrite = 0;PSWEn = 1;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
232             end
233         5'd13:
234             begin
235                 Imm_5or8 = 0;IorD = 0;ALUSrcA = 0;ALUSrcB = 2'b00;ALUop = 2'b00;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
236                 MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 1;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
237             end
238         5'd14:
239             begin
240                 Imm_5or8 = 0;IorD = 0;ALUSrcA = 1;ALUSrcB = 2'b00;ALUop = 2'b01;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
241                 MemWrite = 0;IRWrite = 0;PSWEn = 1;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
242             end
243         5'd15:
244             begin
245                 Imm_5or8 = 0;IorD = 0;ALUSrcA = 1;ALUSrcB = 2'b00;ALUop = 2'b10;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
246                 MemWrite = 0;IRWrite = 0;PSWEn = 1;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
```

```
                MemWrite = 0;IRWrite = 0;PSWEn = 1;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
            end
        5'd16:
            begin
                Imm_5or8 = 0;IorD = 0;ALUSrcA = 1;ALUSrcB = 2'b00;ALUop = 2'b11;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
                MemWrite = 0;IRWrite = 0;PSWEn = 1;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
            end
        5'd17:
            begin
                Imm_5or8 = 0;IorD = 0;ALUSrcA = 1;ALUSrcB = 2'b10;ALUop = 2'b00;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
                MemWrite = 0;IRWrite = 0;PSWEn = 1;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
            end
        5'd18:
            begin
                Imm_5or8 = 0;IorD = 0;ALUSrcA = 1;ALUSrcB = 2'b10;ALUop = 2'b10;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
                MemWrite = 0;IRWrite = 0;PSWEn = 1;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
            end
        5'd19:
            begin
                Imm_5or8 = 0;IorD = 0;ALUSrcA = 1;ALUSrcB = 2'b11;ALUop = 2'b00;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
                MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
            end
        5'd20:
            begin
                Imm_5or8 = 0;IorD = 0;ALUSrcA = 0;ALUSrcB = 2'b00;ALUop = 2'b00;PCSrc = 2'b01;Branch = 1;PCWrite = 0;
                MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
            end
        5'd21:
            begin
                Imm_5or8 = 0;IorD = 0;ALUSrcA = 0;ALUSrcB = 2'b00;ALUop = 2'b00;PCSrc = 2'b01;Branch = 1;PCWrite = 1;
                MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
            end
        5'd22:
            begin
                Imm_5or8 = 0;IorD = 0;ALUSrcA = 0;ALUSrcB = 2'b00;ALUop = 2'b00;PCSrc = 2'b10;Branch = 0;PCWrite = 1;
                MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
```

```
                MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
            end
        5'd23:
            begin
                Imm_5or8 = 1;IorD = 0;ALUSrcA = 0;ALUSrcB = 2'b10;ALUop = 2'b00;PCSrc = 2'b00;Branch = 0;PCWrite = 1;
                MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 1;RegDst = 0;MemtoReg = 2'b11;LLorLH = 0;JAorJR = 0;
            end
        5'd24:
            begin
                Imm_5or8 = 0;IorD = 0;ALUSrcA = 0;ALUSrcB = 2'b00;ALUop = 2'b00;PCSrc = 2'b11;Branch = 0;PCWrite = 1;
                MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 0;RegWrite = 1;RegDst = 0;MemtoReg = 2'b11;LLorLH = 0;JAorJR = 0;
            end
        5'd25:
            begin
                Imm_5or8 = 0;IorD = 0;ALUSrcA = 0;ALUSrcB = 2'b00;ALUop = 2'b00;PCSrc = 2'b00;Branch = 0;PCWrite = 0;
                MemWrite = 0;IRWrite = 0;PSWEn = 0;OutREn = 1;RegWrite = 0;RegDst = 0;MemtoReg = 2'b00;LLorLH = 0;JAorJR = 0;
            end
        default:
            begin
                Imm_5or8 = 0;
                IorD = 0;
                ALUSrcA = 0;
                ALUSrcB = 2'b00;
                ALUop = 2'b00;
                PCSrc = 2'b00;
                Branch = 0;
                PCWrite = 0;
                MemWrite = 0;
                IRWrite = 0;
                PSWEn = 0;
                OutREn = 0;
                RegWrite = 0;
                RegDst = 0;
                MemtoReg = 2'b00;
                LLorLH = 0;
                JAorJR = 0;
            end
    endcase
  end

endmodule
```

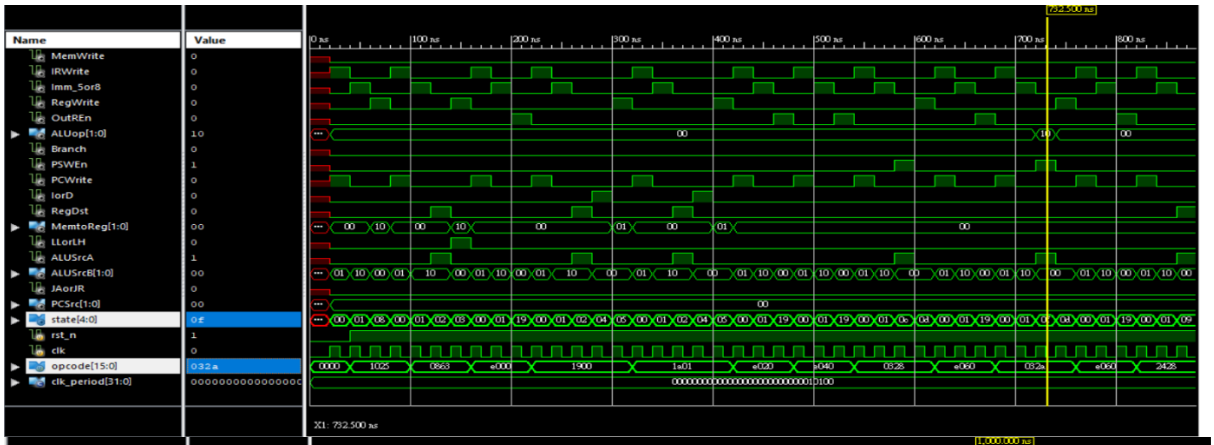
## Testbench:

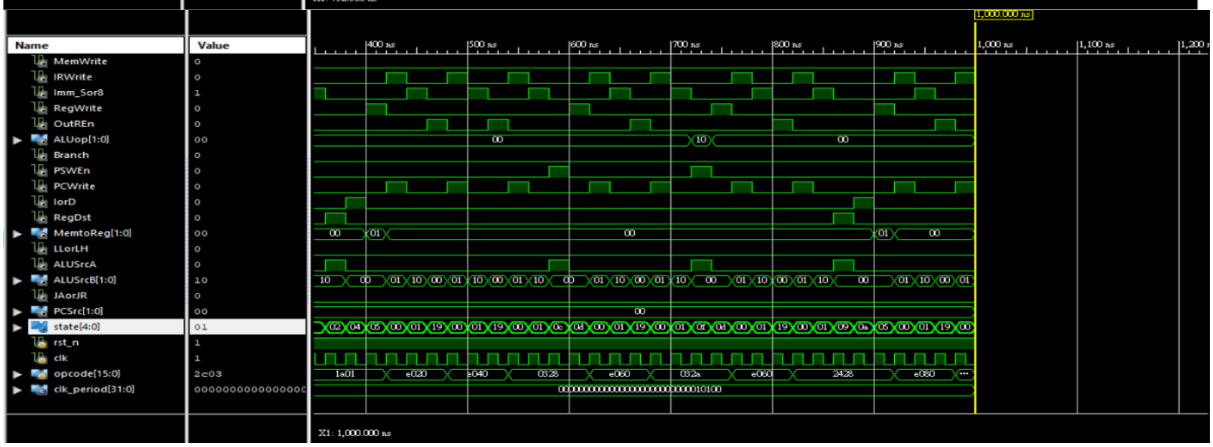
```
`timescale 1ns / 1ps
module controller_t;
    parameter clk_period = 20;
    // Inputs
    reg rst_n;
    reg clk;
    reg [15:0] opcode;
    // Outputs
    wire MemWrite;
    wire IRWrite;
    wire Imm_5or8;
    wire RegWrite;
    wire OutREn;
    wire [1:0] ALUop;
    wire Branch;
    wire PSWEn;
    wire PCWrite;
    wire IorD;
    wire RegDst;
    wire [1:0] MemtoReg;
    wire LLorLH;
    wire ALUSrcA;
    wire [1:0] ALUSrcB;
    wire JAorJR;
    wire [1:0] PCSrc;
    wire [4:0] state;
    // Instantiate the Unit Under Test (UUT)
    controller uut (
        .rst_n(rst_n),
        .clk(clk),
        .opcode(opcode),
        .MemWrite(MemWrite),
        .IRWrite(IRWrite),
        .Imm_5or8(Imm_5or8),
        .RegWrite(RegWrite),
        .OutREn(OutREn),
```

```
36        .OutREn(OutREn),
37        .ALUop(ALUop),
38        .Branch(Branch),
39        .PSWEn(PSWEn),
40        .PCWrite(PCWrite),
41        .IorD(IorD),
42        .RegDst(RegDst),
43        .MemtoReg(MemtoReg),
44        .LLorLH(LLorLH),
45        .ALUSrcA(ALUSrcA),
46        .ALUSrcB(ALUSrcB),
47        .JAorJR(JAorJR),
48        .PCSrc(PCSrc),
49        .state(state)
50    );
51    always begin
52       #(clk_period/2)clk <= 1'b0;
53       #(clk_period/2)clk <= 1'b1;
54    end
55    initial begin
56     $monitor(
57         "T=%0t | state=%0d | opcode=%b | MemWrite=%b | IRWrite=%b | Imm_5or8=%b | RegWrite=%b | OutREn=%b | ALUop=%b | Branch=%b | PSWI
58         $time, state, opcode, MemWrite, IRWrite, Imm_5or8, RegWrite, OutREn, ALUop, Branch, PSWEn, PCWrite, IorD, RegDst, MemtoReg, LL
59     );
60    end
61    initial begin
62       // Initialize Inputs
63       rst_n = 0;
64       clk = 0;
65       opcode = 0;
66       #40
67       rst_n = 1;
68       repeat (3) @(posedge clk)  opcode = 16'b0001_0000_0010_0101;// LLI R0,#25
69       repeat (3) @(posedge clk)  opcode = 16'b0000_1000_0110_0011;// LHI R0,#63
70       repeat (3) @(posedge clk)  opcode = 16'b1110_0000_0000_0000;// OUT R0 (6325H)
71       repeat (5) @(posedge clk)  opcode = 16'b0001_1001_0000_0000;// LDR R1,R0,#0
```

```
70          repeat (3) @(posedge clk)  opcode = 16'b1110_0000_0000_0000;// OUT R0 (6325H)
71          repeat (5) @(posedge clk)  opcode = 16'b0001_1001_0000_0000;// LDR R1,R0,#0
72          repeat (5) @(posedge clk)  opcode = 16'b0001_1010_0000_0001;// LDR R2,R0,#1
73          repeat (3) @(posedge clk)  opcode = 16'b1110_0000_0010_0000;// OUT R1 (47H)
74          repeat (3) @(posedge clk)  opcode = 16'b1110_0000_0100_0000;// OUT R2 (89H)
75          repeat (4) @(posedge clk)  opcode = 16'b0000_0011_0010_1000;// ADD R3,R1,R2
76          repeat (3) @(posedge clk)  opcode = 16'b1110_0000_0110_0000;// OUT R3 (D0H)
77          repeat (4) @(posedge clk)  opcode = 16'b0000_0011_0010_1010;// SUB R3,R1,R2
78          repeat (3) @(posedge clk)  opcode = 16'b1110_0000_0110_0000;// OUT R3 (FFBEH)
79          repeat (5) @(posedge clk)  opcode = 16'b0010_0100_0010_1000;// LDR R4,R1,R2
80          repeat (3) @(posedge clk)  opcode = 16'b1110_0000_1000_0000;// OUT R4 (39H)
81          repeat (4) @(posedge clk)  opcode = 16'b0010_1100_0000_0011;// STR R4 R0 #3(16'h28 = 39H)
82          repeat (4) @(posedge clk)  opcode = 16'b0011_0011_0010_1000;// STR R3 R1 R2(16'hD0 = FFBEH)
83          repeat (3) @(posedge clk)  opcode = 16'b0011_0000_0010_1001;// CMP R1 R2
84          repeat (4) @(posedge clk)  opcode = 16'b0000_0011_0010_1001;// ADC R3 R1 R2
85          repeat (3) @(posedge clk)  opcode = 16'b1110_0000_0110_0000;// OUT R3 (D0H)
86          repeat (3) @(posedge clk)  opcode = 16'b1100_1110_0000_0011;// B[AL] pc = pc + 3 (to pc:16)
87          repeat (3) @(posedge clk)  opcode = 16'b1110_0000_0100_0000;// OUT R2 (89H)
88          repeat (4) @(posedge clk)  opcode = 16'b0011_1001_0100_0011;// ADDI R1,R2 #3
89          repeat (3) @(posedge clk)  opcode = 16'b1110_0000_0010_0000;// OUT R1 (8CH)
90          repeat (4) @(posedge clk)  opcode = 16'b0100_0001_0100_1001;// SUBI R1,R2 #9
91          repeat (3) @(posedge clk)  opcode = 16'b1110_0000_0010_0000;// OUT R1 (80H)
92          repeat (4) @(posedge clk)  opcode = 16'b0101_1010_0010_0000;// MOV R2 R1
93          repeat (3) @(posedge clk)  opcode = 16'b1110_0000_0100_0000;// OUT R2 (80H)
94          repeat (4) @(posedge clk)  opcode = 16'b0000_0000_0010_1011;// SBB R0,R1,R2
95          repeat (3) @(posedge clk)  opcode = 16'b1100_0000_0000_0011;// BEQ pc = pc + 3(to pc:22)
96          repeat (3) @(posedge clk)  opcode = 16'b1110_0000_0110_0000;// OUT R3 (D0H)
97          repeat (3) @(posedge clk)  opcode = 16'b1000_0000_0011_0101;// JMP pc = 16'h35
98          repeat (3) @(posedge clk)  opcode = 16'b1110_0000_0100_0000;// OUT R2 (80H)
99          #100;
100         $finish;
101      end
102
103   endmodule
```

## Behavior:

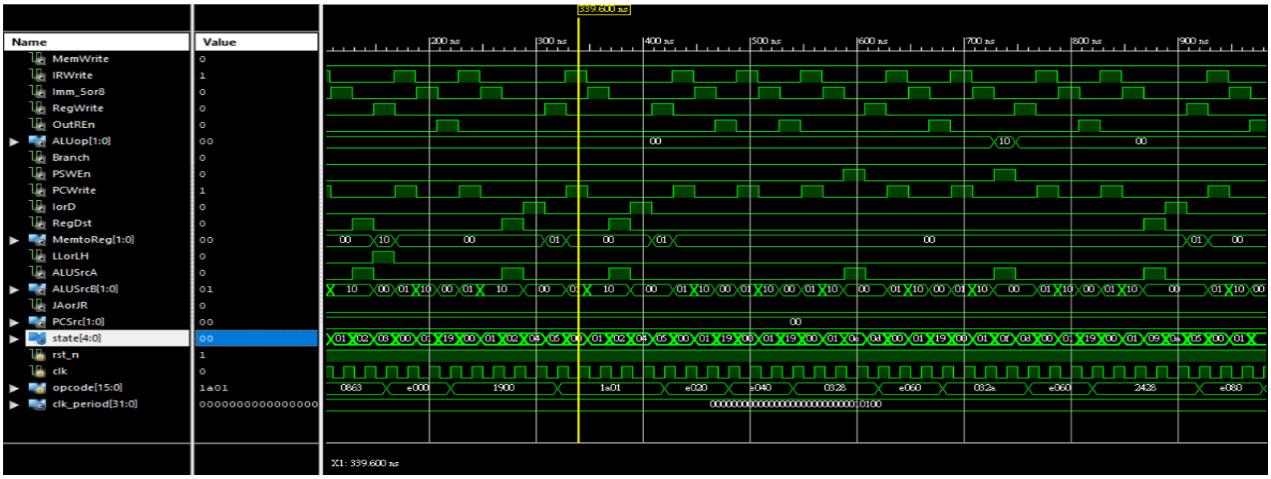




```
Finished circuit initialization process.
T=0 | state=x | opcode=0000000000000000 | MemWrite=x | IRWrite=x | Imm_Sor8=x | RegWrite=x | OutREn=x | ALUop=xx | Branch=x | PSWEn=x | PCWrite=x | IorD=x | RegDst=x | MemtoReg=xx | LLorLH=x | ALUSrcA=x | ALUSrcB=xx | JAorJR=x | PCSrc=xx
T=20000 | state=0 | opcode=0000000000000000 | MemWrite=0 | IRWrite=1 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=1 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=40000 | state=1 | opcode=0001000000100101 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=60000 | state=6 | opcode=0001000000100101 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=10 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=80000 | state=0 | opcode=0001000000100101 | MemWrite=0 | IRWrite=1 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=1 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=100000 | state=1 | opcode=0000100001100011 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=120000 | state=2 | opcode=0000100001100011 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=1 | MemtoReg=00 | LLorLH=0 | ALUSrcA=1 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=140000 | state=3 | opcode=0000100001100011 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=10 | LLorLH=1 | ALUSrcA=0 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=160000 | state=0 | opcode=1110000000000000 | MemWrite=0 | IRWrite=1 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=1 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=180000 | state=1 | opcode=1110000000000000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=200000 | state=25 | opcode=1110000000000000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=1 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=220000 | state=0 | opcode=0001100100000000 | MemWrite=0 | IRWrite=1 | Imm_Sor8=0 | RegWrite=0 | OutREn=1 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=1 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=240000 | state=1 | opcode=0001100100000000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=260000 | state=2 | opcode=0001100100000000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=1 | MemtoReg=00 | LLorLH=0 | ALUSrcA=1 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=280000 | state=4 | opcode=0001100100000000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=1 | RegDst=0 | MemtoReg=01 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=300000 | state=5 | opcode=0001100100000000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=1 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=01 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=320000 | state=0 | opcode=0001101000000001 | MemWrite=0 | IRWrite=1 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=1 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=340000 | state=1 | opcode=0001101000000001 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=360000 | state=2 | opcode=0001101000000001 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=1 | MemtoReg=00 | LLorLH=0 | ALUSrcA=1 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=380000 | state=4 | opcode=0001101000000001 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=1 | RegDst=0 | MemtoReg=01 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=400000 | state=5 | opcode=0001101000000001 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=1 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=01 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=420000 | state=0 | opcode=1110000000100000 | MemWrite=0 | IRWrite=1 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=1 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=440000 | state=1 | opcode=1110000000100000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=460000 | state=25 | opcode=1110000000100000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=1 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=480000 | state=0 | opcode=1110000000100000 | MemWrite=0 | IRWrite=1 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=1 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=500000 | state=1 | opcode=1110000000100000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=520000 | state=25 | opcode=1110000000100000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=1 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=540000 | state=0 | opcode=0000001100101000 | MemWrite=0 | IRWrite=1 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=1 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=560000 | state=1 | opcode=0000001100101000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=580000 | state=12 | opcode=0000001100101000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=1 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=1 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=600000 | state=13 | opcode=0000001100101000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=1 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=1 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=620000 | state=0 | opcode=1110000001100000 | MemWrite=0 | IRWrite=1 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=1 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=640000 | state=1 | opcode=1110000001100000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=660000 | state=25 | opcode=1110000001100000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=1 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=680000 | state=0 | opcode=0000001100101010 | MemWrite=0 | IRWrite=1 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=1 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=700000 | state=1 | opcode=0000001100101010 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
```

```
T=240000 | state=1 | opcode=0001100100000000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=260000 | state=2 | opcode=0001100100000000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=1 | MemtoReg=00 | LLorLH=0 | ALUSrcA=1 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=280000 | state=4 | opcode=0001100100000000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=1 | RegDst=0 | MemtoReg=01 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=300000 | state=5 | opcode=0001100100000000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=1 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=01 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=320000 | state=0 | opcode=0001101000000001 | MemWrite=0 | IRWrite=1 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=1 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=340000 | state=1 | opcode=0001101000000001 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=360000 | state=2 | opcode=0001101000000001 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=1 | MemtoReg=00 | LLorLH=0 | ALUSrcA=1 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=380000 | state=4 | opcode=0001101000000001 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=1 | RegDst=0 | MemtoReg=01 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=400000 | state=5 | opcode=0001101000000001 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=1 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=01 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=420000 | state=0 | opcode=1110000000100000 | MemWrite=0 | IRWrite=1 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=1 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=440000 | state=1 | opcode=1110000000100000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=460000 | state=25 | opcode=1110000000100000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=1 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=480000 | state=0 | opcode=1110000000100000 | MemWrite=0 | IRWrite=1 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=1 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=500000 | state=1 | opcode=1110000000100000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=520000 | state=25 | opcode=1110000000100000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=1 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=540000 | state=0 | opcode=0000001100101000 | MemWrite=0 | IRWrite=1 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=1 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=560000 | state=1 | opcode=0000001100101000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=580000 | state=12 | opcode=0000001100101000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=1 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=1 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=600000 | state=13 | opcode=0000001100101000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=1 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=1 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=620000 | state=0 | opcode=1110000001100000 | MemWrite=0 | IRWrite=1 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=1 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=640000 | state=1 | opcode=1110000001100000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
T=660000 | state=25 | opcode=1110000001100000 | MemWrite=0 | IRWrite=0 | Imm_Sor8=0 | RegWrite=0 | OutREn=1 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=00 | JAorJR=0 | PCSrc=00
T=680000 | state=0 | opcode=0000001100101010 | MemWrite=0 | IRWrite=1 | Imm_Sor8=0 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=1 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=01 | JAorJR=0 | PCSrc=00
T=700000 | state=1 | opcode=0000001100101010 | MemWrite=0 | IRWrite=0 | Imm_Sor8=1 | RegWrite=0 | OutREn=0 | ALUop=00 | Branch=0 | PSWEn=0 | PCWrite=0 | IorD=0 | RegDst=0 | MemtoReg=00 | LLorLH=0 | ALUSrcA=0 | ALUSrcB=10 | JAorJR=0 | PCSrc=00
ISim>
```

## Post-Route:



# 10. Complete processor

## VHDL

```
1   timescale 1ns / 1ps
2   module CPU(
3       input clk,
4       input rst_n,
5       input [15:0]ext_addr,
6       input [15:0]ext_data,
7       input test,
8       input tMemWrite,
9       input  TestMem,
10      input start,
11      output [15:0]OutR,
12      output [4:0]state,
13      output [15:0]opcode,
14      output [15:0]TestMemout
15      );
16      wire MemWrite          ;
17      wire IRWrite           ;
18      wire Imm_5or8          ;
19      wire RegWrite          ;
20      wire OutREn            ;
21      wire [1:0]ALUop        ;
22      wire Branch            ;
23
24      wire PSWEn             ;
25      wire PCWrite           ;
26      wire IorD              ;
27      wire RegDst            ;
28      wire [1:0]MemtoReg     ;
29      wire LLorLH            ;
30      wire ALUSrcA           ;
31      wire [1:0]ALUSrcB      ;
32      wire JAorJR            ;
33      wire [1:0]PCSrc        ;
34      wire test_memwirte;
35      assign dp_memwrite = test?tMemWrite:MemWrite;
36      datapath dp(
```

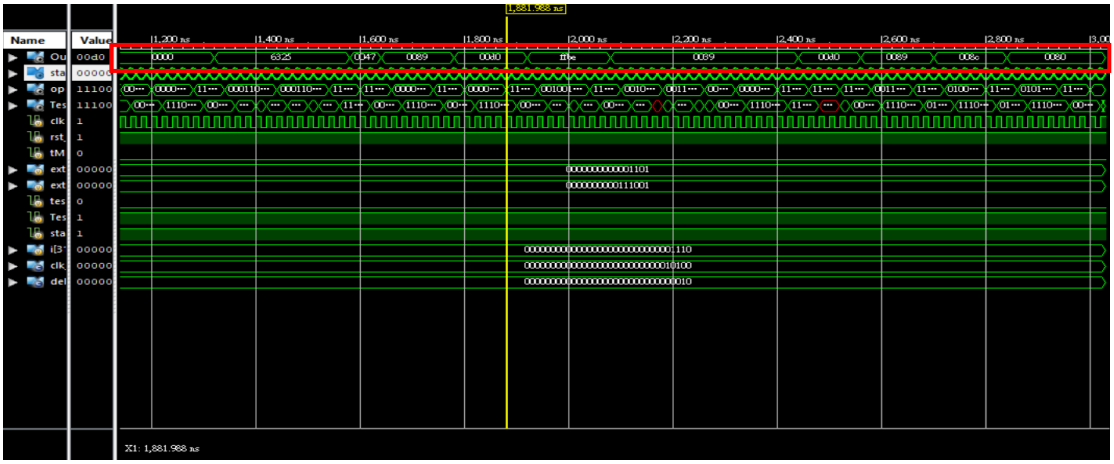
```
36      datapath dp(
37      .rst_n(rst_n)           ,
38      .clk(clk)               ,
39      .MemWrite(dp_memwrite)  ,
40      .IRWrite(IRWrite)       ,
41      .Imm_5or8(Imm_5or8)     ,
42      .RegWrite(RegWrite)     ,
43      .OutREn(OutREn)         ,
44      .ALUop(ALUop)           ,
45      .Branch(Branch)         ,
46      .PSWEn(PSWEn)           ,
47      .PCWrite(PCWrite)       ,
48      .IorD(IorD)             ,
49      .RegDst(RegDst)         ,
50      .MemtoReg(MemtoReg)     ,
51      .LLorLH(LLorLH)         ,
52      .ALUSrcA(ALUSrcA)       ,
53      .ALUSrcB(ALUSrcB)       ,
54      .JAorJR(JAorJR)         ,
55      .PCSrc(PCSrc)           ,
56      .ext_addr(ext_addr)     ,
57      .ext_data(ext_data)     ,
58      .test(test)             ,
59      .TestMem(TestMem)       ,
60      .OutR(OutR)             ,
61      .TestMemout(TestMemout),
62      .opcode(opcode)
63      );
64
65      wire clk_n = ~clk;
66      controller ctr(
67      .rst_n(rst_n)           ,
68      .clk(clk_n)             ,
69      .start(start)           ,
70      .opcode(opcode)         ,
71      .MemWrite(MemWrite)     ,
```

```
        .MemWrite(MemWrite)      ,
        .IRWrite(IRWrite)        ,
        .Imm_5or8(Imm_5or8)      ,
        .RegWrite(RegWrite)      ,
        .OutREn(OutREn)          ,
        .ALUop(ALUop)            ,
        .Branch(Branch)          ,
        .PSWEn(PSWEn)            ,
        .PCWrite(PCWrite)        ,
        .IorD(IorD)              ,
        .RegDst(RegDst)          ,
        .MemtoReg(MemtoReg)      ,
        .LLorLH(LLorLH)          ,
        .ALUSrcA(ALUSrcA)        ,
        .ALUSrcB(ALUSrcB)        ,
        .JAorJR(JAorJR)          ,
        .PCSrc(PCSrc)            ,
        .state(state)
        );

endmodule
```

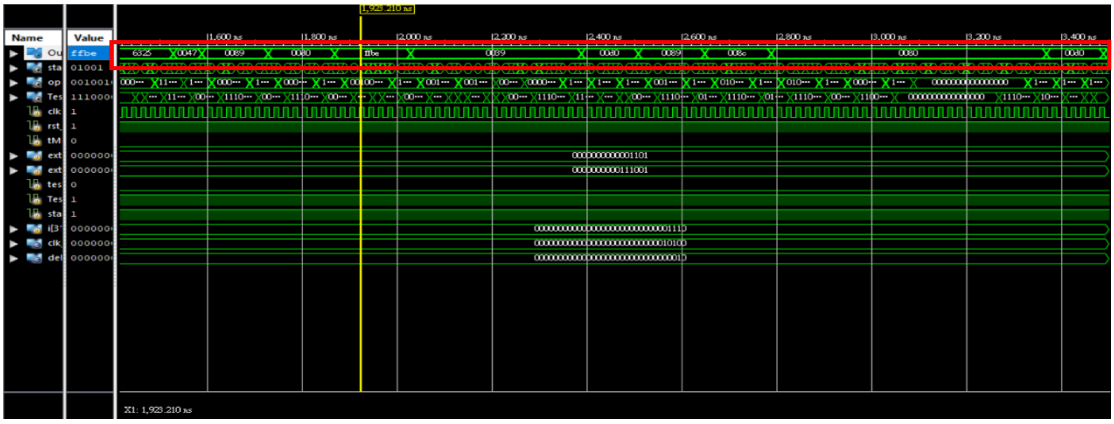
**Testbench:**

**與 datapath 相同**

**Behavior:**



**Post-Route:**

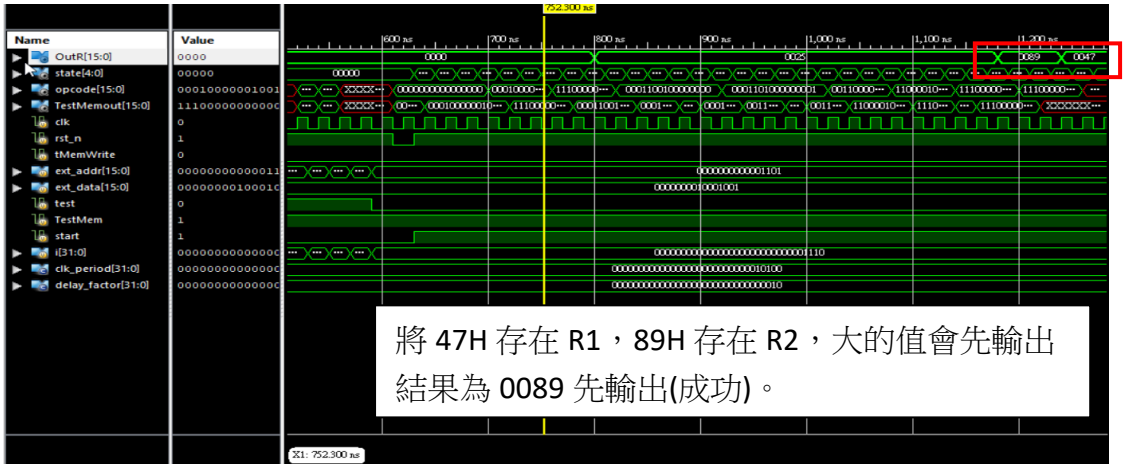


1. **Find the minimum and maximum from two numbers in memory.**

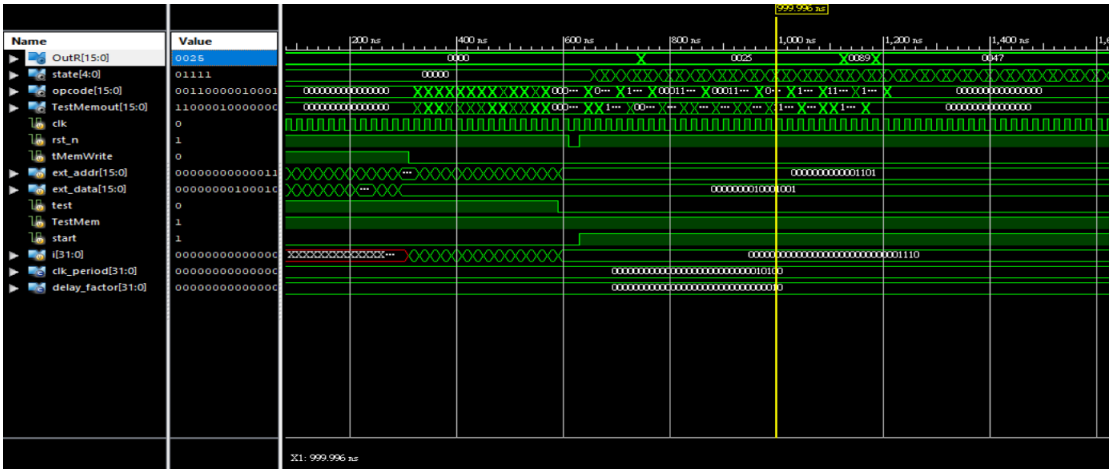
**Testbench:**

```
        write_mem(16'h0,16'b0001_0000_0010_0101); // LLI R0,#25
        write_mem(16'h1,16'b1110_0000_0000_0000); // OUT R0 (25H)
        write_mem(16'h2,16'b0001_1001_0000_0000); // LDR R1,R0,#0
        write_mem(16'h3,16'b0001_1010_0000_0001); // LDR R2,R0,#1
        write_mem(16'h4,16'b0011_0000_0100_0101); // CMP R2 R1
        write_mem(16'h5,16'b1100_0010_0000_0011); // BCS pc = pc + 2 (true)

        //R1 > R2
        write_mem(16'h6,16'b1110_0000_0010_0000); // OUT R1 (47H)
        write_mem(16'h7,16'b1110_0000_0100_0000); // OUT R2 (89H)

        //R2 > R1
        write_mem(16'h9,16'b1110_0000_0100_0000); // OUT R2 (89H)
        write_mem(16'hA,16'b1110_0000_0010_0000); // OUT R1 (47H)


        write_mem(16'h25,16'h47); // data (25h, 47h)
        write_mem(16'h26,16'h89); // data (26h, 89h)
```

**Behavior:**




**Post-Route:**

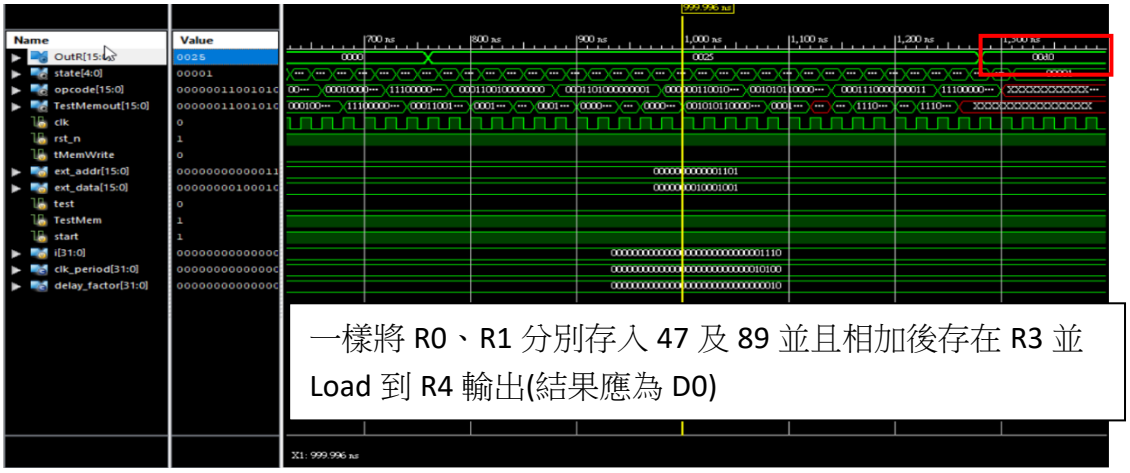


2. **Add two numbers in memory and store the result in another memory location.**

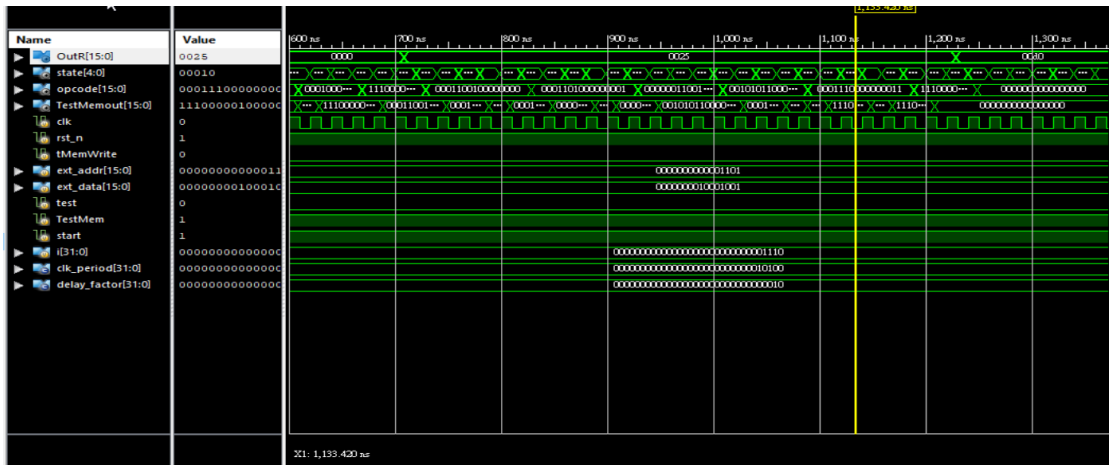
**Testbench:**

```
write_mem(16'h0,16'b0001_0000_0010_0101); // LLI R0,#25
write_mem(16'h1,16'b1110_0000_0000_0000); // OUT R0 (25H)
write_mem(16'h2,16'b0001_1001_0000_0000); // LDR R1,R0,#0
write_mem(16'h3,16'b0001_1010_0000_0001); // LDR R2,R0,#1
write_mem(16'h4,16'b0000_0011_0010_1000); // ADD R3,R1,R2
write_mem(16'h5,16'b0010_1011_0000_0011); // STR R3 R0 #3(16'h28 = D0H)
write_mem(16'h6,16'b0001_1100_0000_0011); // LDR R4,R0,#3
write_mem(16'h7,16'b1110_0000_1000_0000); // OUT R4 (D0H)
write_mem(16'h25,16'h47); // data (25h, 47h)
write_mem(16'h26,16'h89); // data (26h, 89h)
```

**Behavior:**




**Post-Route:**

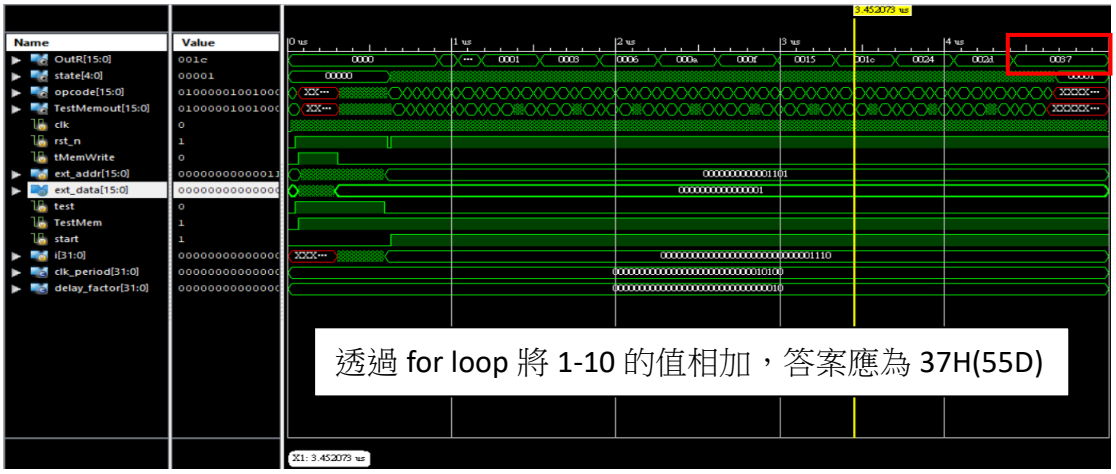


## 3. Add ten numbers in consecutive memory locations.

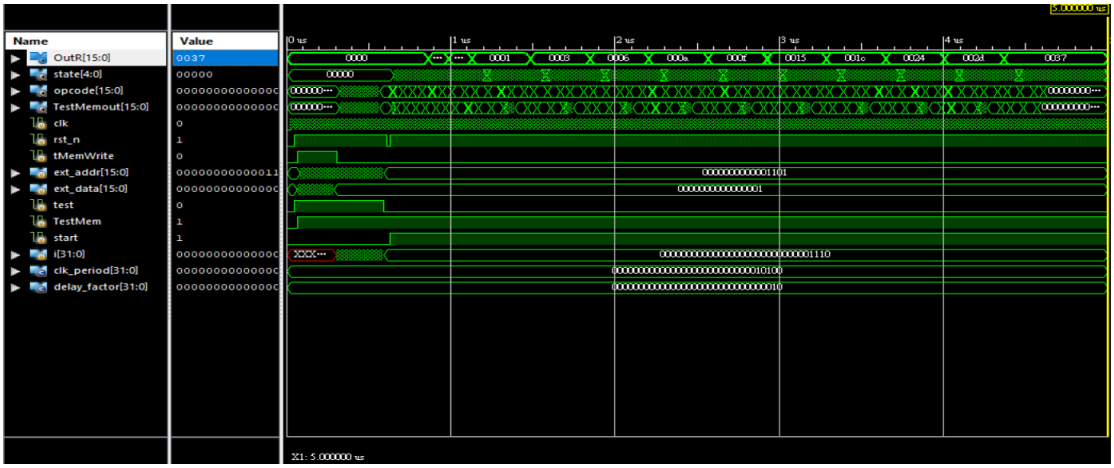
### Testbench:

```
write_mem(16'h0,16'b0001_0011_0000_0000); // LLI R3,0H
write_mem(16'h1,16'b1110_0000_0110_0000); // OUT R3 (0H)
write_mem(16'h2,16'b0001_0000_0000_0001); // LLI R0,#1
write_mem(16'h3,16'b1110_0000_0000_0000); // OUT R0 (1)
write_mem(16'h4,16'b0001_0001_0000_1010); // LLI R1,0AH
write_mem(16'h5,16'b1110_0000_0010_0000); // OUT R1 (0AH)


write_mem(16'h6,16'b0000_0011_0110_0000); // ADD R3,R3,R0
write_mem(16'h7,16'b1110_0000_0110_0000); // OUT R3
write_mem(16'h8,16'b0011_1000_0000_0001); // ADDI R0,R0 #1
write_mem(16'h9,16'b0100_0001_0010_0001); // SUBI R1,R1 #1

write_mem(16'ha,16'b1100_0001_1111_1011); // BNE (R1 == 0)?end:pc = pc - 5

write_mem(16'h25,16'h1); // data (25h, 1h)
```

### Behavior:




### Post-Route:

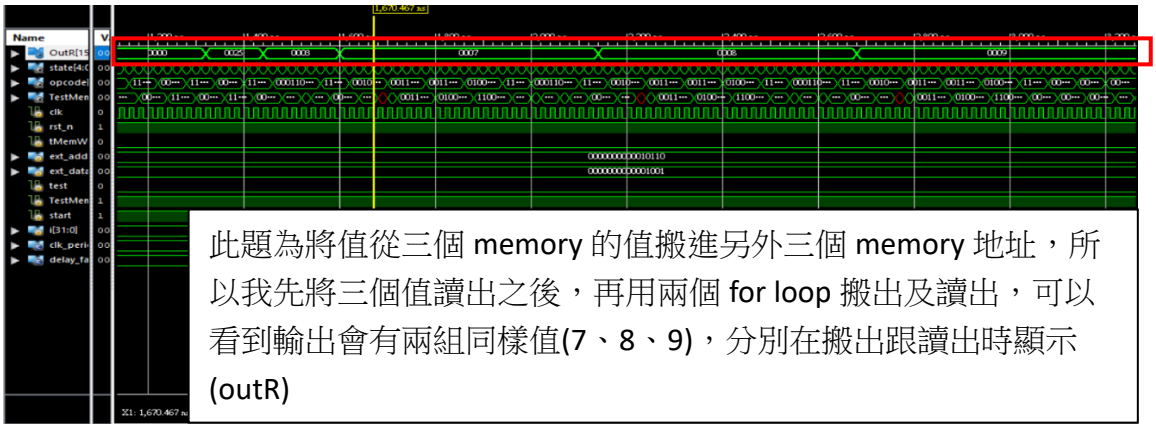


## 4. Mov a memory block of N words from one place to another.

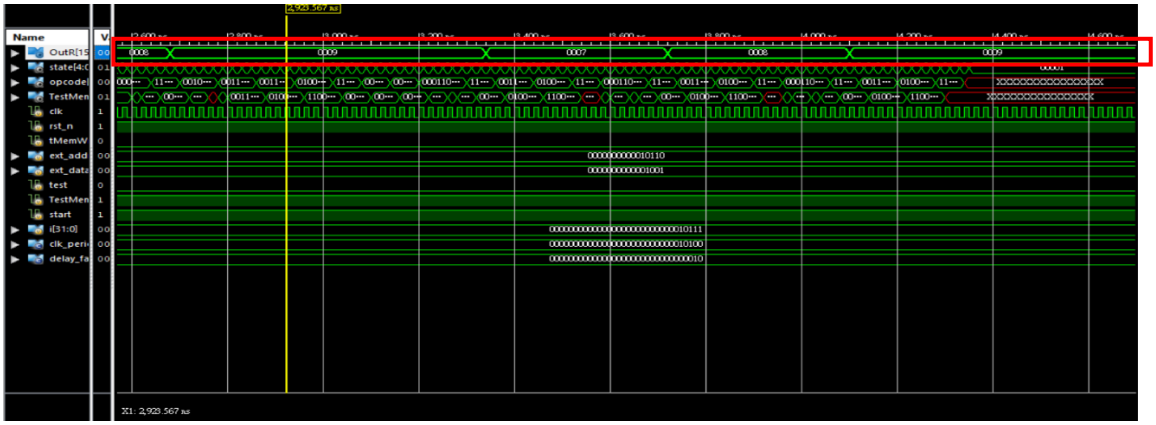
**Testbench:**

```
write_mem(16'h0,16'b0001_0000_0000_0000); //LLI R0,0h//迴圈初始
write_mem(16'h1,16'b1110_0000_0000_0000); //OUT R0 (0h)
write_mem(16'h2,16'b0001_0010_0010_0101); //LLI R2,25H//地址
write_mem(16'h3,16'b1110_0000_0100_0000); //OUT R2 (25H)
write_mem(16'h4,16'b0001_0001_0000_0011); //LLI R1,#3//三個值
write_mem(16'h5,16'b1110_0000_0010_0000); //OUT R1 (3H)

//move
write_mem(16'h6,16'b0001_1011_0100_0011); //LDR R3,R2,#3
write_mem(16'h7,16'b1110_0000_0110_0000); //OUT R3 (7H)
write_mem(16'h8,16'b0010_1011_0100_0000); //STR Mem\[R2+#0]<- R3//把mem的值存在暫存器
write_mem(16'h9,16'b0011_1000_0000_0001); //ADDI R0,R0,#1
write_mem(16'hA,16'b0011_1010_0100_0001); //ADDI R2,R2,#1
write_mem(16'hB,16'b0100_0001_0010_0001); //SUBI R1,R1,#1
write_mem(16'hC,16'b1100_0001_1111_1001); //BNE if R1 != 0
//PC <- PC-7

write_mem(16'hD,16'b0001_0000_0010_0101); //LLI R0,25H
write_mem(16'hE,16'b0001_0001_0000_0011); //LLI R1,#3
//read
write_mem(16'hF,16'b0001_1010_0000_0000); //LDR R2,R0,#0
write_mem(16'h10,16'b1110_0000_0100_0000); //OUT R2
write_mem(16'h11,16'b0011_1000_0000_0001); //ADDI R0,R0,#1
write_mem(16'h12,16'b0100_0001_0010_0001); //SUBI R1,R1,#1
write_mem(16'h13,16'b1100_0001_1111_1011); //BNE if R1 != 0
//PC <- PC-5
write_mem(16'h28,16'h7); //data (31h, 07h)
write_mem(16'h29,16'h8); //data (32h, 08h)
write_mem(16'h2a,16'h9); //data (33h, 09h)
```

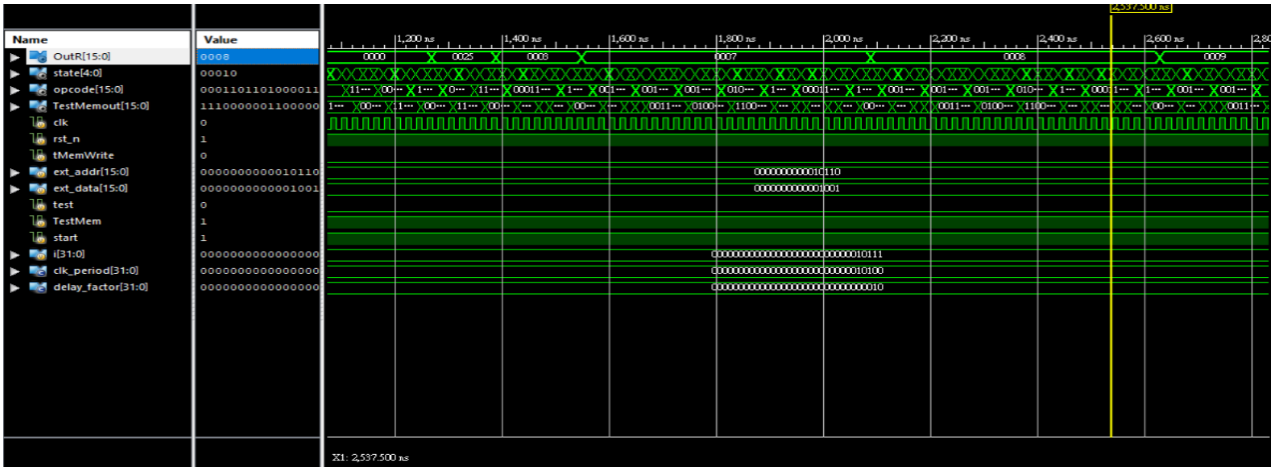
**Behavior:**

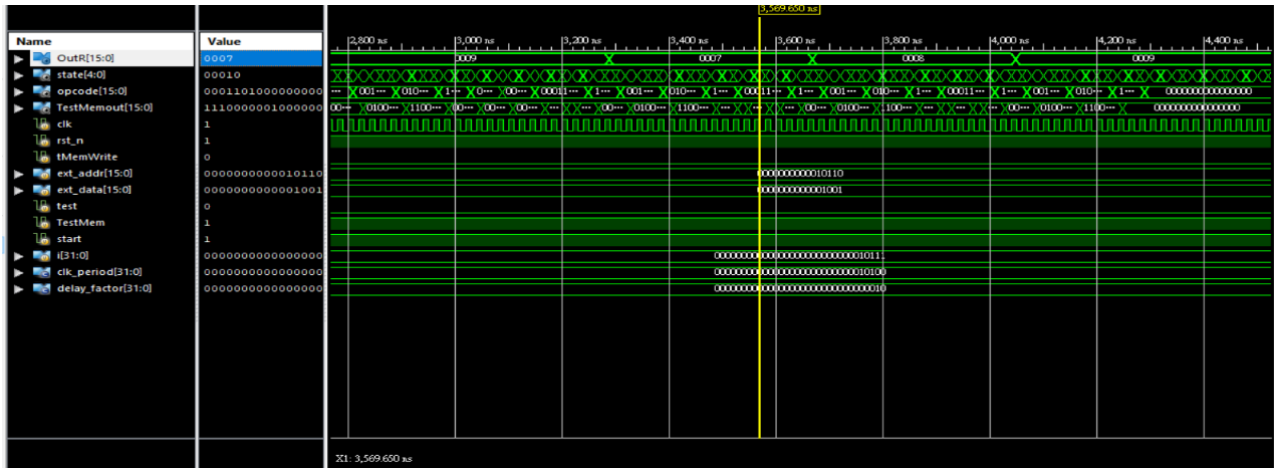


此題為將值從三個 memory 的值搬進另外三個 memory 地址，所以我先將三個值讀出之後，再用兩個 for loop 搬出及讀出，可以看到輸出會有兩組同樣值(7、8、9)，分別在搬出跟讀出時顯示(outR)



## Post-Route:


Name
Value
OutR[15:0]
0008
state[4:0]
00010
opcode[15:0]
0001101101000011
TestMemout[15:0]
1110000001100000
clk
0
rst_n
1
tMemWrite
0
ext_addr[15:0]
0000000000010110
ext_data[15:0]
0000000000001001
test
0
TestMem
1
start
1
i[31:0]
0000000000000000
clk_period[31:0]
0000000000000000
delay_factor[31:0]
0000000000000000
X1: 2,537.500 ns


Name
Value
OutR[15:0]
0007
state[4:0]
00010
opcode[15:0]
0001101000000000
TestMemout[15:0]
1110000001000000
clk
1
rst_n
1
tMemWrite
0
ext_addr[15:0]
0000000000010110
ext_data[15:0]
0000000000001001
test
0
TestMem
1
start
1
i[31:0]
0000000000000000
clk_period[31:0]
0000000000000000
delay_factor[31:0]
0000000000000000
X1: 3,569.650 ns

# Hardware cost

Xilinx Design Summary Page 1 of 3

| CPU Project Status | | | |
|---|---|---|---|
| **Project File:** | FPGA_CPU_VHDL.xise | **Parser Errors:** | No Errors |
| **Module Name:** | CPU | **Implementation State:** | Placed and Routed |
| **Target Device:** | xc6vlx75t-1ff484 | • **Errors:** | No Errors |
| **Product Version:** | ISE 14.7 | • **Warnings:** | 65 Warnings (0 new) |
| **Design Goal:** | Balanced | • **Routing Results:** | All Signals Completely Routed |
| **Design Strategy:** | Xilinx Default (unlocked) | • **Timing Constraints:** | All Constraints Met |
| **Environment:** | System Settings | • **Final Timing Score:** | 0 (Timing Report) |

| Device Utilization Summary | | | | [-] |
|---|---|---|---|---|
| **Slice Logic Utilization** | **Used** | **Available** | **Utilization** | **Note(s)** |
| Number of Slice Registers | 287 | 93,120 | 1% | |
| Number used as Flip Flops | 287 | | | |
| Number used as Latches | 0 | | | |
| Number used as Latch-thrus | 0 | | | |
| Number used as AND/OR logics | 0 | | | |
| Number of Slice LUTs | 353 | 46,560 | 1% | |
| Number used as logic | 287 | 46,560 | 1% | |
| Number using O6 output only | 237 | | | |
| Number using O5 output only | 0 | | | |
| Number using O5 and O6 | 50 | | | |
| Number used as ROM | 0 | | | |
| Number used as Memory | 64 | 16,720 | 1% | |
| Number used as Dual Port RAM | 0 | | | |
| Number used as Single Port RAM | 64 | | | |
| Number using O6 output only | 64 | | | |
| Number using O5 output only | 0 | | | |
| Number using O5 and O6 | 0 | | | |
| Number used as Shift Register | 0 | | | |
| Number used exclusively as route-thrus | 2 | | | |
| Number with same-slice register load | 2 | | | |
| Number with same-slice carry load | 0 | | | |
| Number with other load | 0 | | | |
| Number of occupied Slices | 194 | 11,640 | 1% | |
| Number of LUT Flip Flop pairs used | 504 | | | |
| Number with an unused Flip Flop | 220 | 504 | 43% | |

file:///C:/Users/eca/Downloads/FPGA/CPU_summary.html 5/13/2025

Xilinx Design Summary Page 2 of 3

| | | | | |
|---|---|---|---|---|
| Number with an unused LUT | 151 | 504 | 29% | |
| Number of fully used LUT-FF pairs | 133 | 504 | 26% | |
| Number of unique control sets | 17 | | | |
| Number of slice register sites lost to control set restrictions | 17 | 93,120 | 1% | |
| Number of bonded IOBs | 83 | 240 | 34% | |
| Number of RAMB36E1/FIFO36E1s | 0 | 156 | 0% | |
| Number of RAMB18E1/FIFO18E1s | 0 | 312 | 0% | |
| Number of BUFG/BUFGCTRLs | 1 | 32 | 3% | |
| Number used as BUFGs | 1 | | | |
| Number used as BUFGCTRLs | 0 | | | |
| Number of ILOGICE1/ISERDESE1s | 0 | 360 | 0% | |
| Number of OLOGICE1/OSERDESE1s | 0 | 360 | 0% | |
| Number of BSCANs | 0 | 4 | 0% | |
| Number of BUFHCEs | 0 | 72 | 0% | |
| Number of BUFIODQSs | 0 | 36 | 0% | |
| Number of BUFRs | 0 | 18 | 0% | |
| Number of CAPTUREs | 0 | 1 | 0% | |
| Number of DSP48E1s | 0 | 288 | 0% | |
| Number of EFUSE_USRs | 0 | 1 | 0% | |
| Number of FRAME_ECCs | 0 | 1 | 0% | |
| Number of GTXE1s | 0 | 8 | 0% | |
| Number of IBUFDS_GTXE1s | 0 | 6 | 0% | |
| Number of ICAPs | 0 | 2 | 0% | |
| Number of IDELAYCTRLs | 0 | 9 | 0% | |
| Number of IODELAYE1s | 0 | 360 | 0% | |
| Number of MMCM_ADVs | 0 | 6 | 0% | |
| Number of PCIE_2_0s | 0 | 1 | 0% | |
| Number of STARTUPs | 1 | 1 | 100% | |
| Number of SYSMONs | 0 | 1 | 0% | |
| Number of TEMAC_SINGLEs | 0 | 4 | 0% | |
| Average Fanout of Non-Clock Nets | 4.89 | | | |

| Performance Summary | | | | [-] |
|---|---|---|---|---|
| **Final Timing Score:** | 0 (Setup: 0, Hold: 0) | **Pinout Data:** | Pinout Report | |
| **Routing Results:** | All Signals Completely Routed | **Clock Data:** | Clock Report | |

file:///C:/Users/eca/Downloads/FPGA/CPU_summary.html 5/13/2025

| | | | |
|---|---|---|---|
| **Timing Constraints:** | All Constraints Met | | |

| Detailed Reports | | | | | [-] |
|---|---|---|---|---|---|
| **Report Name** | **Status** | **Generated** | **Errors** | **Warnings** | **Infos** |
| Synthesis Report | Current | Tue May 13 15:38:10 2025 | 0 | 65 Warnings (0 new) | 3 Infos (0 new) |
| Translation Report | Current | Tue May 13 15:38:17 2025 | 0 | 0 | 0 |
| Map Report | Current | Tue May 13 15:38:52 2025 | 0 | 0 | 8 Infos (0 new) |
| Place and Route Report | Current | Tue May 13 15:39:08 2025 | 0 | 0 | 3 Infos (0 new) |
| Power Report | | | | | |
| Post-PAR Static Timing Report | Current | Tue May 13 15:39:17 2025 | 0 | 0 | 4 Infos (0 new) |
| Bitgen Report | | | | | |

| Secondary Reports | | [-] |
|---|---|---|
| **Report Name** | **Status** | **Generated** |
| ISIM Simulator Log | Current | Tue May 13 15:40:39 2025 |
| Post-Place and Route Simulation Model Report | Current | Tue May 13 15:39:59 2025 |

**Date Generated:** 05/13/2025 - 17:13:11

## Conclusion

這次期末作業的目標，是將期中所完成的 schematic 電路設計轉換為 Verilog 硬體描述語言。在轉換過程中，由於我對整個硬體架構已經非常熟悉，因此整體進行算是順利。儘管過程中仍花了不少時間進行 debug，但這些挑戰也轉化為寶貴的學習經驗。

我遵循模組化設計的方式，首先建立基本元件模組（如 MUX、Decoder、Register File、Memory 等），接著實作期中未完成的 Controller，最終成功整合為一個完整的 Processor。我認為這個過程非常有成就感，特別是在 Controller 的實作上，只需將已完成的 FSM 架構用 case 敘述實現，並清楚定義每一狀態所需控制訊號，就能順利完成。

一樣我整理出了一些在這次的作業上我遇到比較棘手的問題:

1. 語法熟悉與思維轉換：

雖然我過去有寫過 VHDL，但一開始未能立即轉換至 Verilog 的思維，尤其在 blocking/non-blocking 的使用、output 是否需加 reg 等語法細節上花了不少時間重新適應。不過在中後段進入狀況後，開發速度也有明顯提升。

2. Datapath 整合困難與訊號管理：

與 schematic 相比，Verilog 中的訊號需手動命名並傳入模組，無法像圖形化方式直接拖線接上。因此在整合 datapath 所有元件時，debug 花了很多時間確認訊號是否傳遞正確。我後來的做法是「需要哪個 wire 再宣告哪個」，避免過多冗餘，但同時也需注意宣告順序，避免未定義或 race condition 的情況。

3. Controller 實作與解碼設計：

在將 FSM 實作成 case 結構時，我選擇將 instruction decoder 內嵌於 Controller 中，這樣可以集中管理控制訊號。此做法雖然設計上需高度清楚各指令格式與對應訊號，但由於期中已完整設計 FSM 與指令解碼邏輯，因此這部分進行得相當順利。

4. 整合時的訊號接錯與同步問題：

整合 Controller 和 Datapath 時，有時會遇到訊號雖已宣告並在 Controller 中產生，卻沒有正確傳入 Datapath，或是因 wire 未定義導致競爭條件發生。這些問題多靠波形模擬與訊號對照逐一解決，最終成功完成老師所提供的所有測試程式。

相較於期中，這次期末作業的難度相對較低，因為只要期中基礎架構完整，期末幾乎只是延伸與整合而已。我也藉此機會進一步優化整體架構，例如改善跳躍指令處理、Sign Extend 等細節，並成功完成 Controller 的設計。我對這次的成果還算滿意，也很感謝老師提供這樣的實作練習，不僅加強了我在硬體語言上的基礎，也讓這份作業成為我作品集中值得紀錄的一部分。