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）

Midterm Report

指道教授: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

8-to-1 MUX 24

16 bit 8-to-1 mux 29

16 bit DFF 34

16 bit Eight-Register Register File 37

Full adder 40

16 bit full adder 43

16 bit full adder(complement) 45

16 bit ALU 48

256x16 memory 52

Complete Data path 56

Hardware cost **113**

Conclusion **116**

How to design an RISC processor

首先，要建造一個 16 bit 的 RISC processor 我們需要先確定好 ISA(Instruction Set Architecture)還有我們想要實現的 cycle 狀態，在這次的 project 中我們選定的是 multi-cycle 的系統因為相較於單周期的系統，單週期系統頃向把最長的指令當作是各個指令的一次執行時間，但這樣會發現對於一些指令他們不需要這麼長的指令執行時間，於是透過多週期指令可以將每一個指令的執行時間縮減，如果進一步想要在優化的話可以還搭配 pipeline 來完成，但是本次的作業是採取多週期系統。

ISA 的部份如下圖，指令集我們會完成基本加減法、load、store 和跳躍指令等等，這次的 processor 位元是 16bit，這些不同的參數都會影響到我們未來在設計上面不同作法。

另外在硬體的成本上面，多週期系統可以有效的重複使用(例如 memory)相較於單週期系統，可以更有效優化單週期系統上的缺點(時間、硬體)。

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.

再來，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將我們的 processor 開始設計出來了。

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

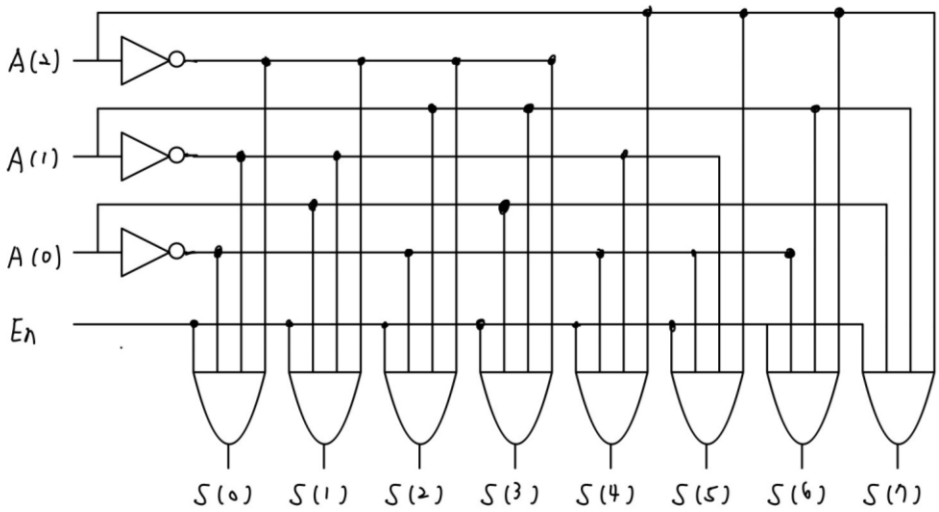
(1) enabled-controlled 3-to-8 noninverting output decoder

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

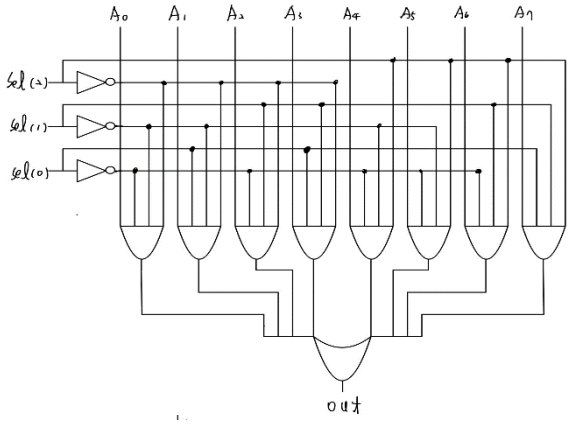
當 En 信號為 0 時輸出全為 0，所以這邊就沒有特別標示出來，可以看到當 En 為 1 時指令才能正常輸出，透過 A(2:0)來選擇現在要輸出為第幾個位置，這樣被選到的暫存器就可以正常輸出他所對應的位置。



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。

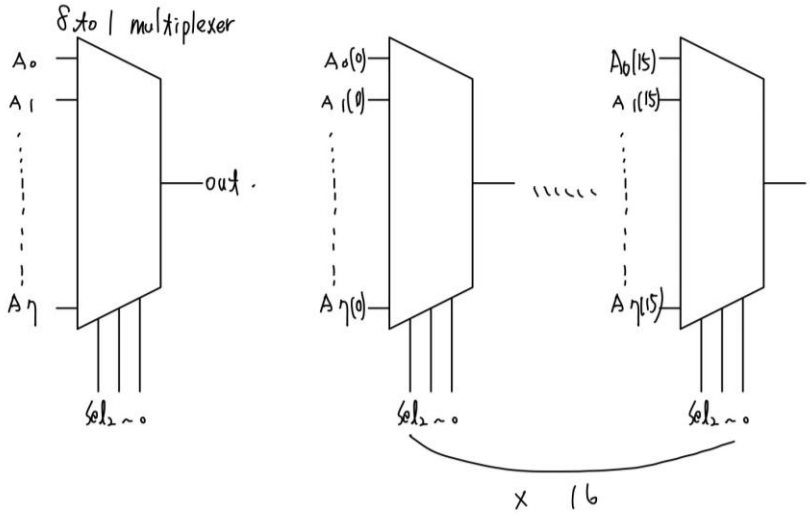
(2) 8-to-1 multiplexer&16 bit 8-to-1 multiplexer

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

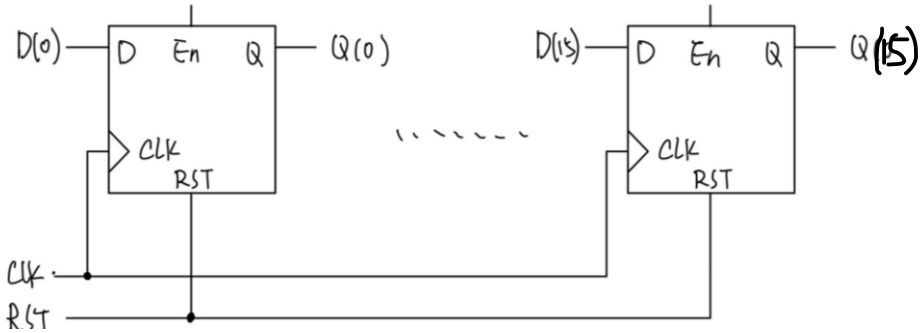
最後將 16 個 8-to-1 多工器合起來即可得到一個 16 bit 的多工器



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

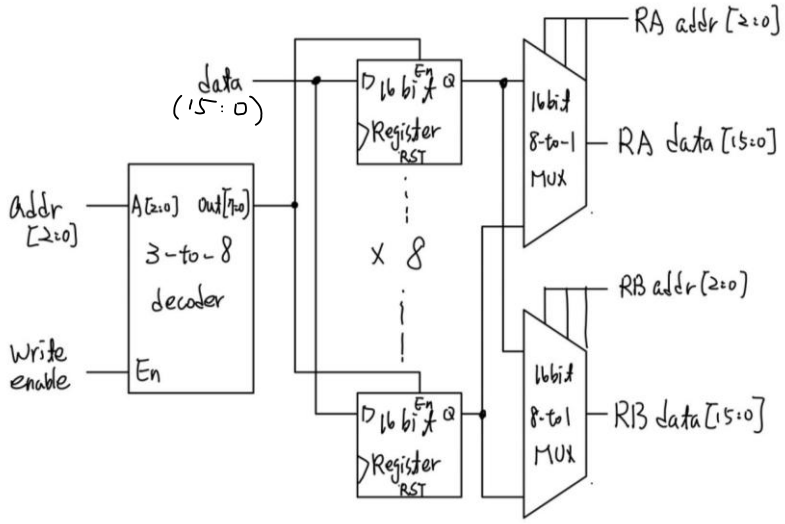
接下來要製作的是 16 bit 的 DFF，採取正緣觸發和搭配 RST，用來記錄 16bit 的資料，總共有 8 組的 DFF。



(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



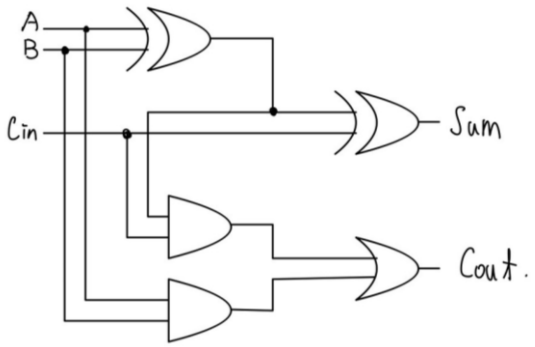
二、A 16-Bit ALU

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

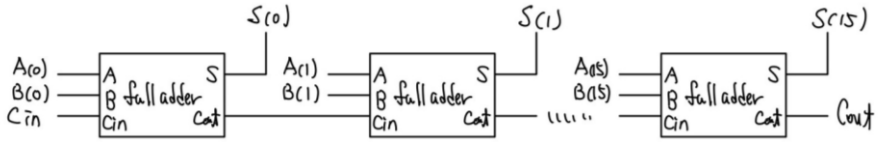
(1) full adder

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。



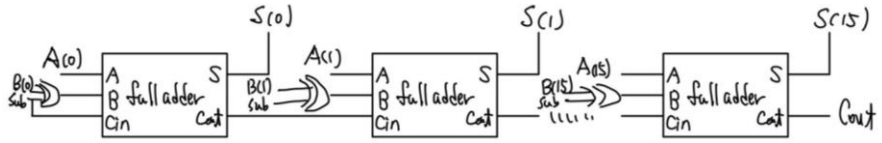
(2) 16-bit adder

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

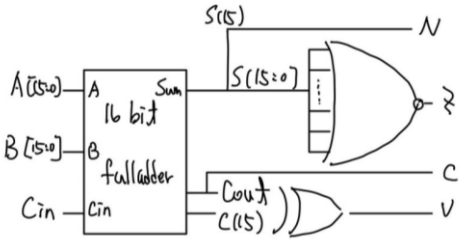
這個步驟是為了要可以實作出減法器，實際做法就是要對 B 取二補數，透過反向加一，將 B 和減法信號透過 XOR 串接起來，再把減法信號接到 CIN 即可。



(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。

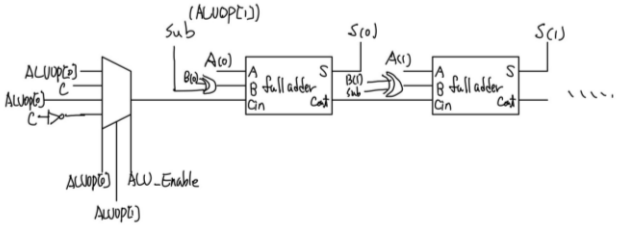
另外作業中雖然不用完成 N 和 V 指令，但是這邊也進一步說明該如何完成。N 指令目的在判斷是否為負號，只要將 S(15)輸出成 N 即可判斷。V 為判斷是否溢位，只要將最後兩級的 COUT 做 XOR 即可完成



(5)16 bit ALU

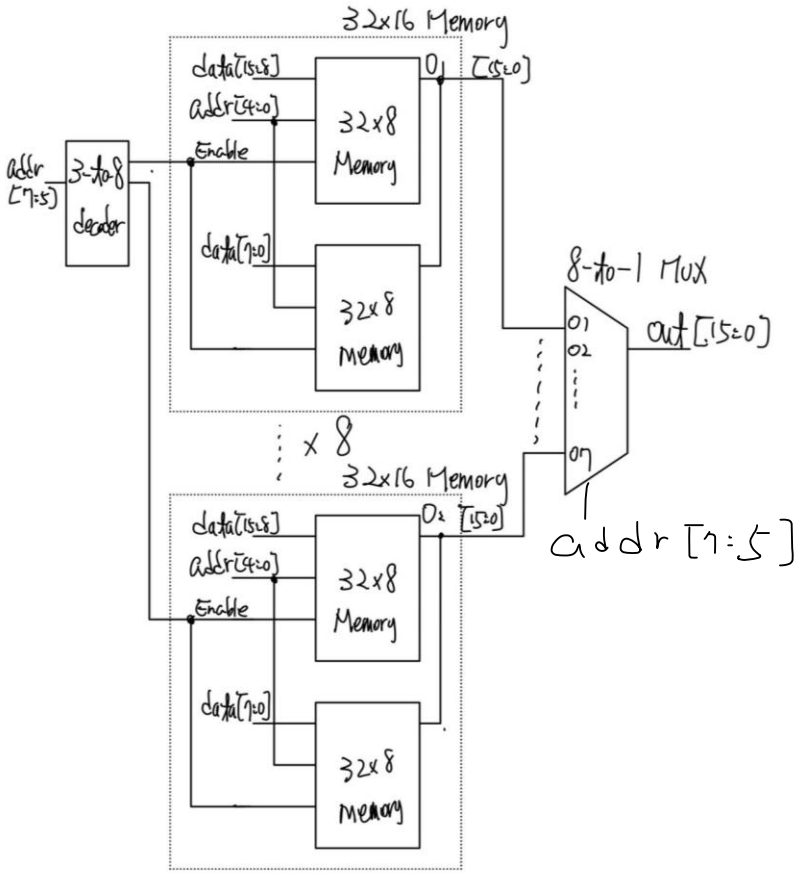
最後將整個根據 ISA 完整將功能實現出來，透過 inst[1]的來控制加減法，inst[0]控制是否要加上 C flag。

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



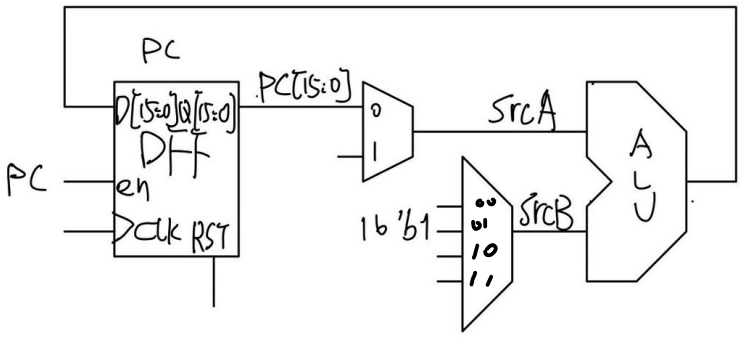
三、A 256x16 Memory Module

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

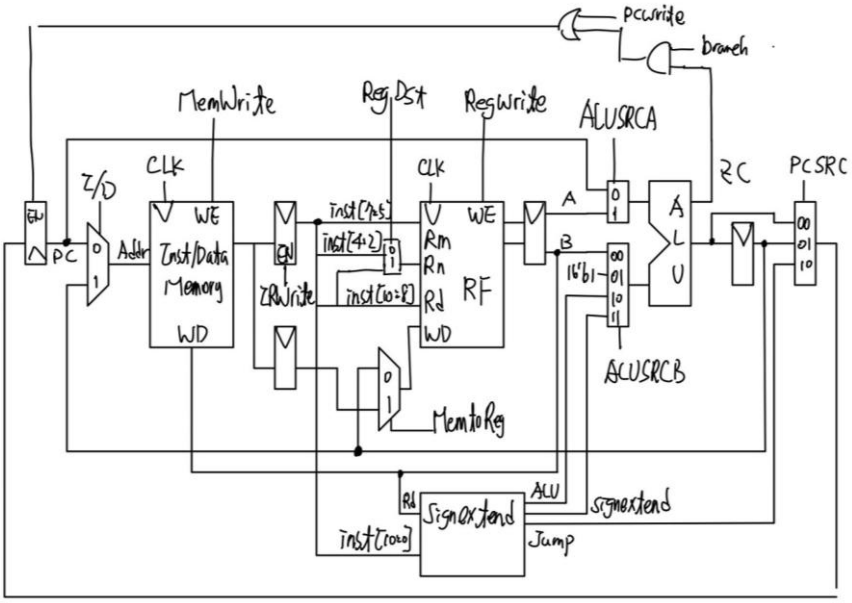
PC 循環其實並沒有太難，只要透過一個暫存器(DFF)以及 ALU 就可以完成，PC 全名是 Program Counter 顧名思義就是要記錄程式現在執行到第幾個，所以將 PC 傳到 ALU 然後再加一，傳到 PC 的 DFF 即可完成一次 PC 紀錄。



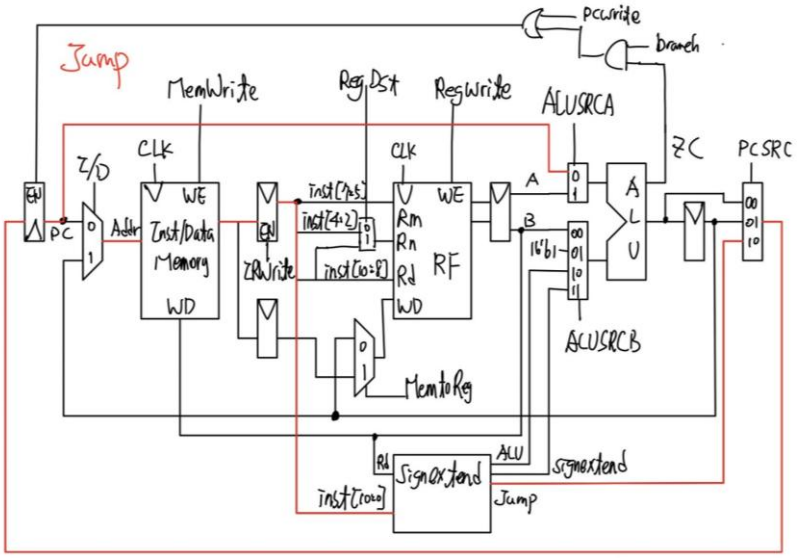
五、Complete Datapath

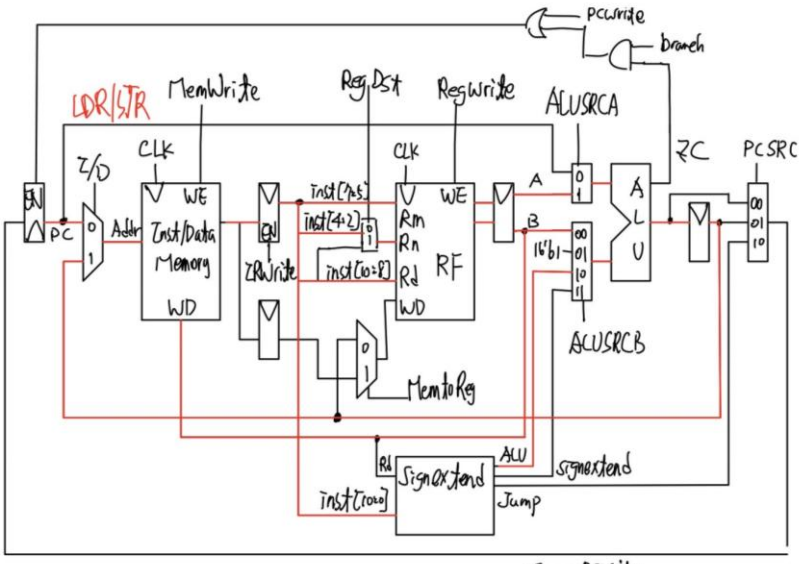
有了上面的基本元件之後就可以將完整的 processor 組合出來了，因為我們是使用 multi-cycle 的方式，所以要將我們整個 data path 用四個 DFF 暫存器切分成五個部分(IF、ID、EXE、MEMORY、WB)，將資料停下來等待下一個階段送出。

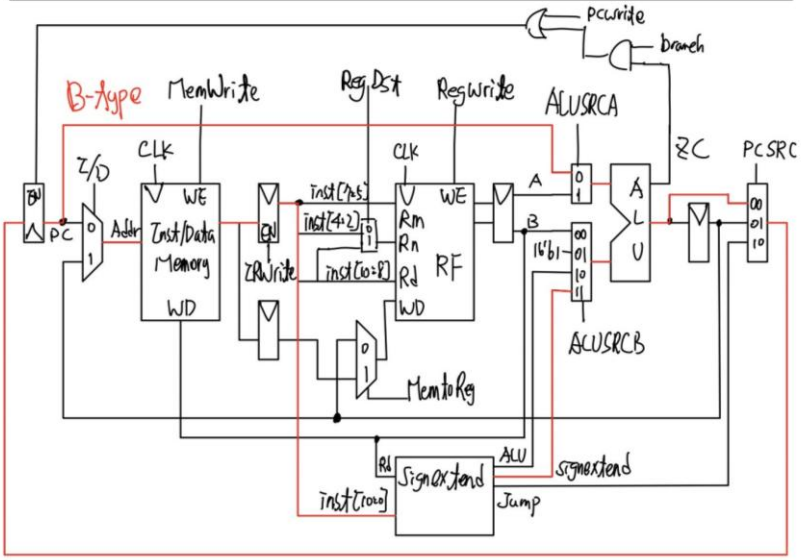
首先，我會先把需要完整五階段的指令實作出來，因為這樣就可以取得大致上的架構，當需要在增加其他指令時只需要增加一些 MUX 即可，這邊先考慮 LDR 以及 STR 指令再來是 R-type 指令以及跳躍指令，完成這些指令之後就可以將大概的架構完成了。

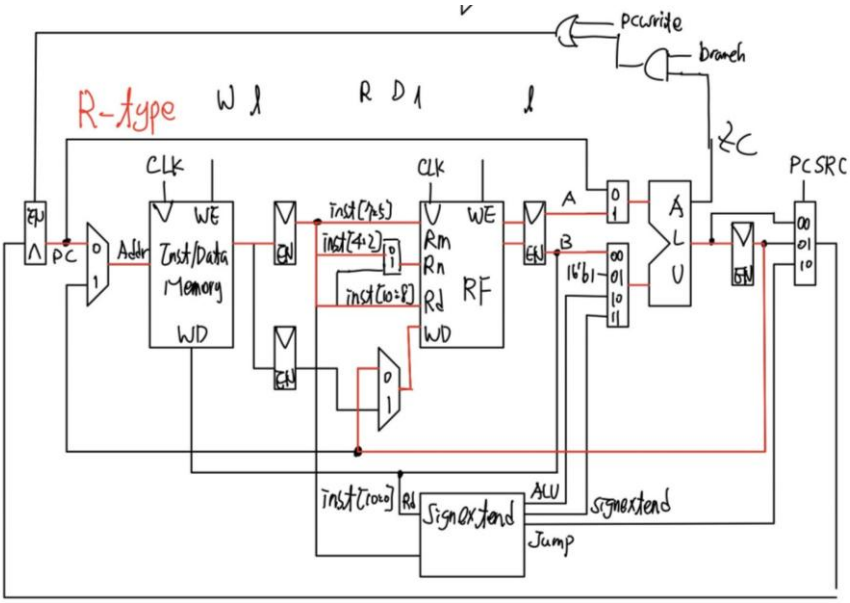


以下為指令會經過的路徑:

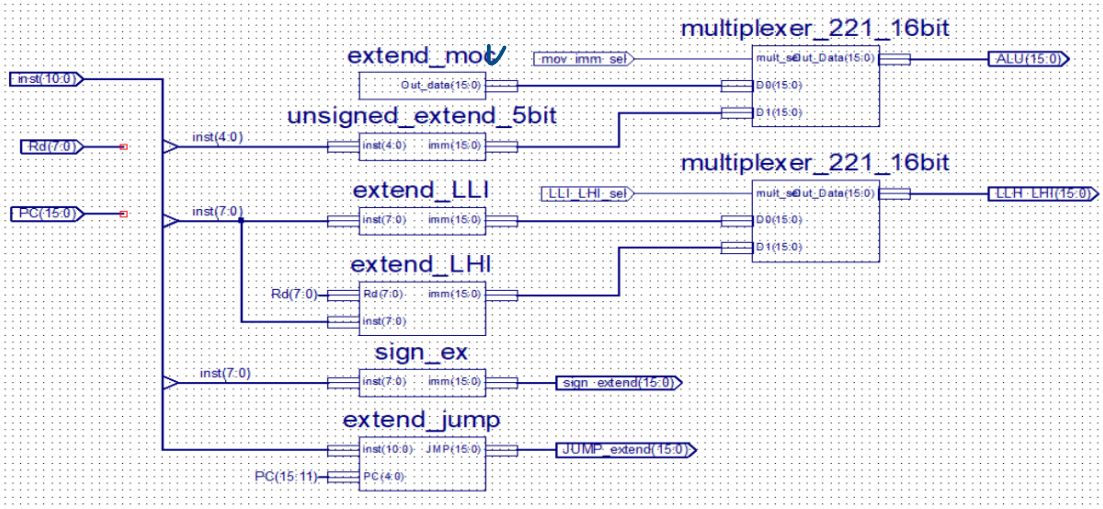



PCWrite
branch
LDR/STR
MemWrite
RegDst
RegWrite
ALUSRCA
ZC
PCSRC
Z/D
CLK
WE
PC
Addr
Inst/Data Memory
WD
EN
ZRWrite
inst[7:5]
inst[4:2]
inst[10:8]
Rm
Rn
Rd
RF
A
B
16'b1
ALU
ALUSRCB
MemtoReg
Rd
Sign extend
ALU
signextend
inst[10:0]
Jump


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

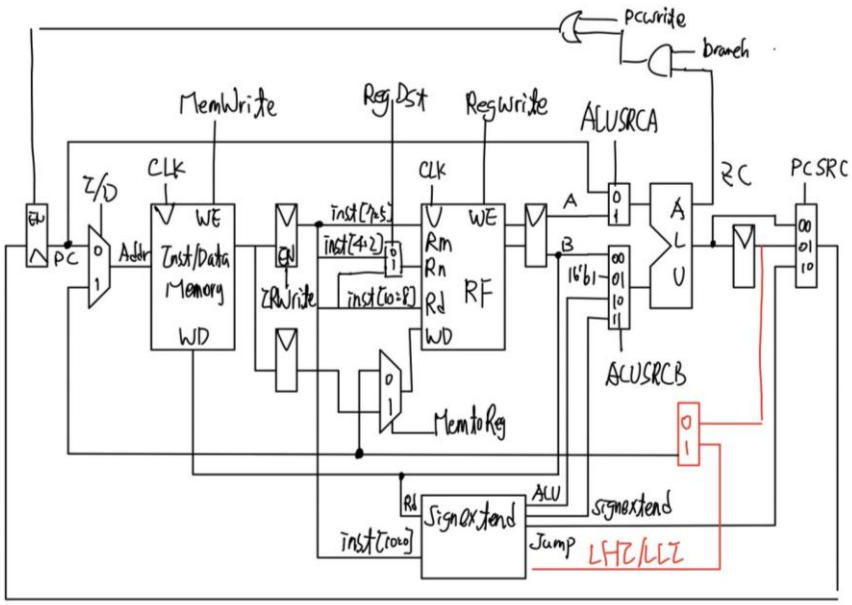

PCWrite
branch
R-type
ZC
PCSRC
CLK
WE
PC
Addr
Inst/Data Memory
WD
EN
inst[7:5]
inst[4:2]
inst[10:8]
Rm
Rn
Rd
RF
A
B
16'b1
ALU
inst[10:0]
Rd
Sign extend
ALU
signextend
Jump

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:



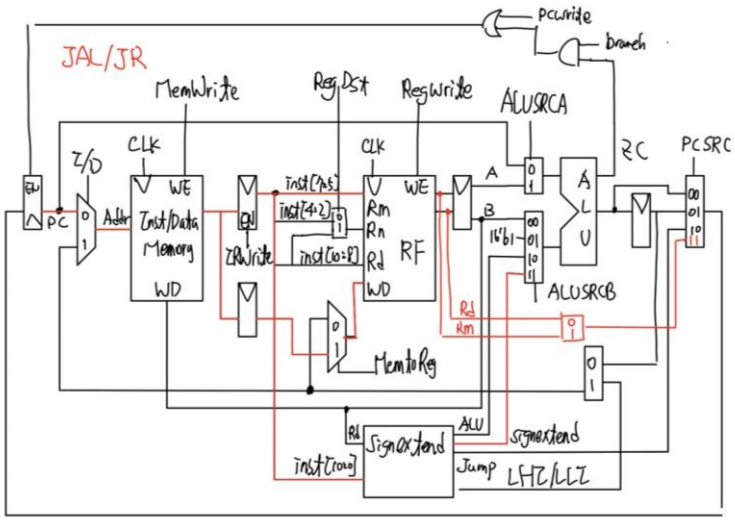
接下來就完成剩下的指令，首先是 LHI/LLI，需要將 sign extend 計算的值輸出，並且加上一個 MUX 去決定要將 ALU 的值還是 LHI/LLI 的值輸出到 PC 使用

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



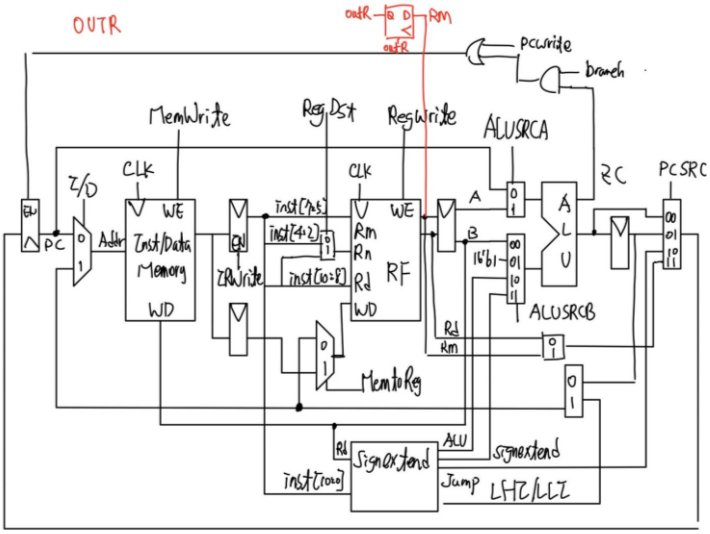
再來要處理 JAL/JR，JAL 為先將 PC 的值給 Rd 後再將 PC 加上 signextend or Rm 之後送到 PC，所以我們需要先將 PCSRC 的部分變成 4-to-1 的多工器，並且將 Rm or Rd 拉出來用一個 MUX 輸出給 PCSRC，使用路徑及新增的元件如下。

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

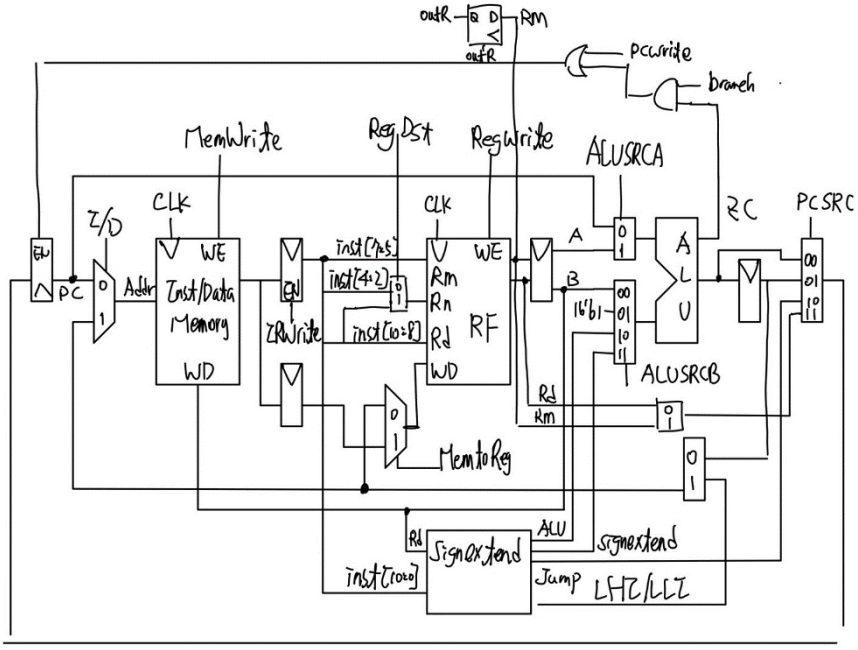


最後一個指令為 OutR、HLT，OutR 只要讓 Rm 拉出來觀察即可，所以我加上了一個 DFF 把 Rm 的值拉出來。

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



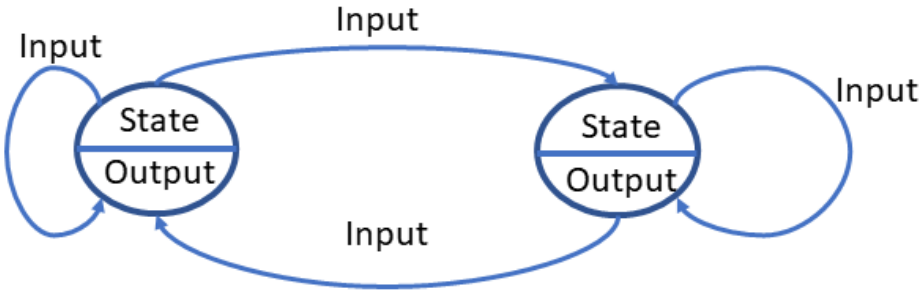
完整 data path:



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。

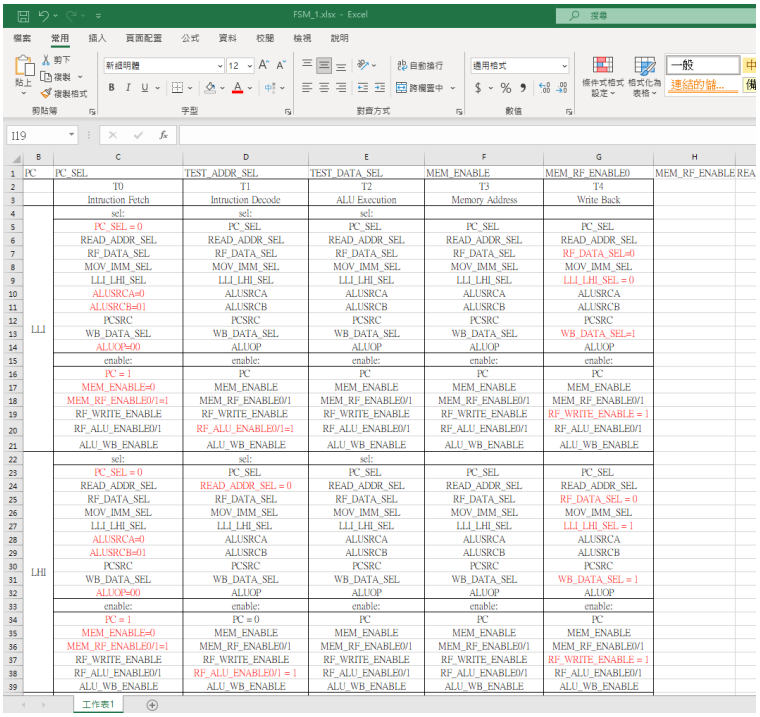
此次我設計的架構是根據摩爾有限狀態機去做設計的，每一個狀態由 CLK 控制，只要當前狀態改變，就會改變輸出的值，非常適合運用在這次的多週期的系統上面。



Generic Moore model

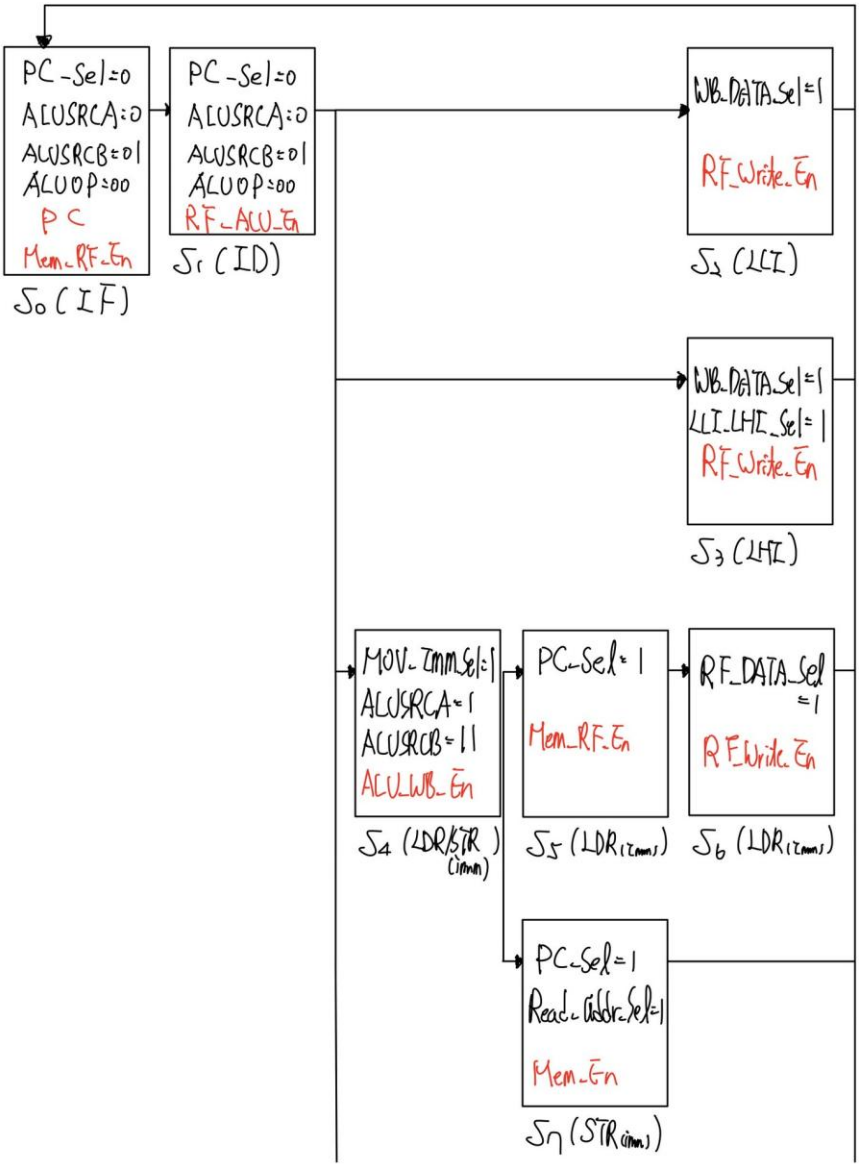
這裡的 input 即為 data path 內的 instruction 根據 instruction 來決定下一個 state，而 output 就是提供給不同的元件的信號。

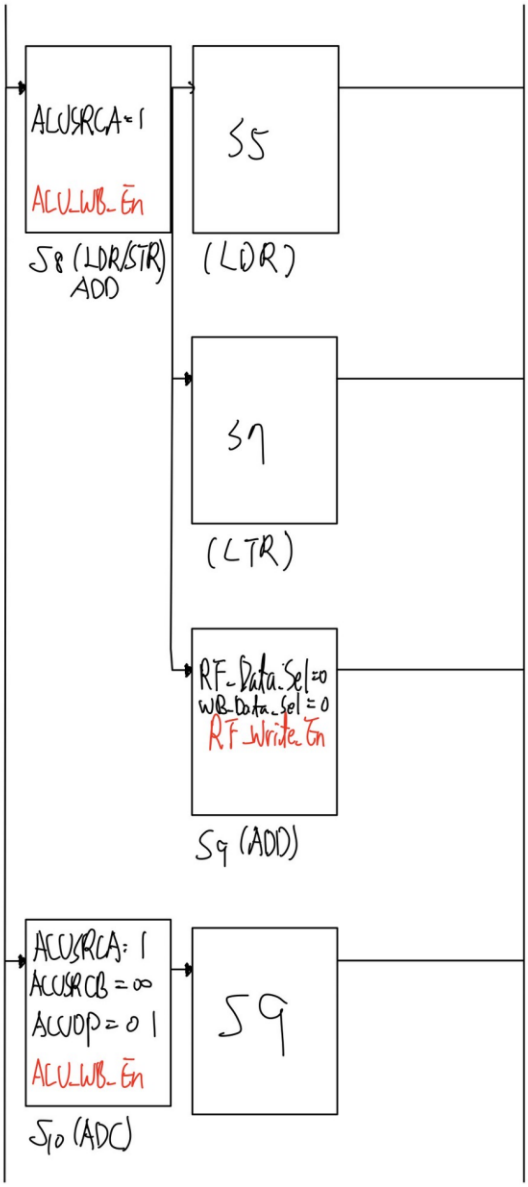
在這裡，我會先講每一個指令的所有狀態先透過 excel 打出所需要的指令信號，然後再將每個指令共同的地方整理出來方便節省 FSM 的流程，最後再將表格轉化成流程圖實現 controller。

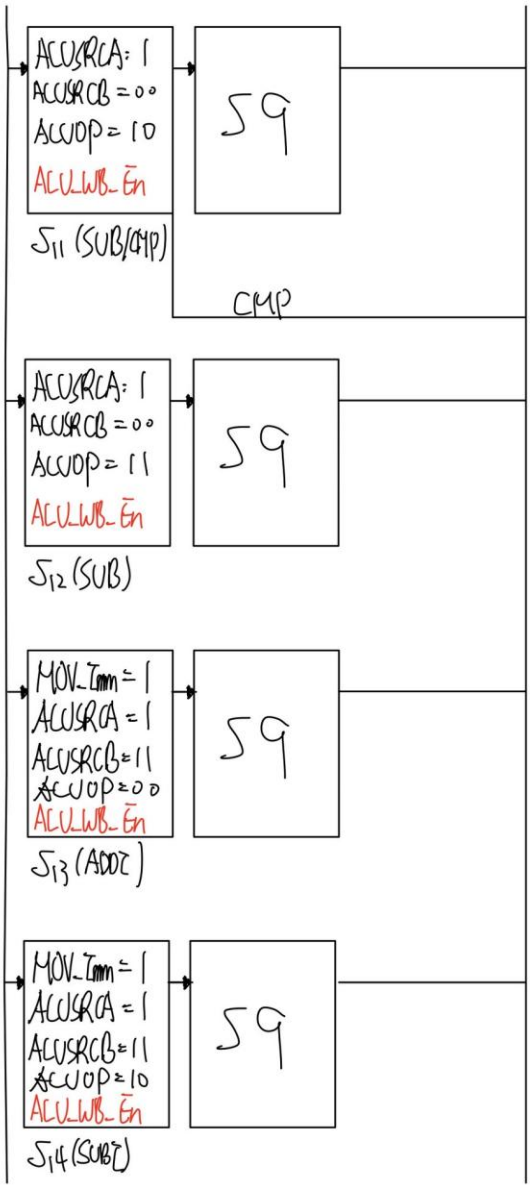


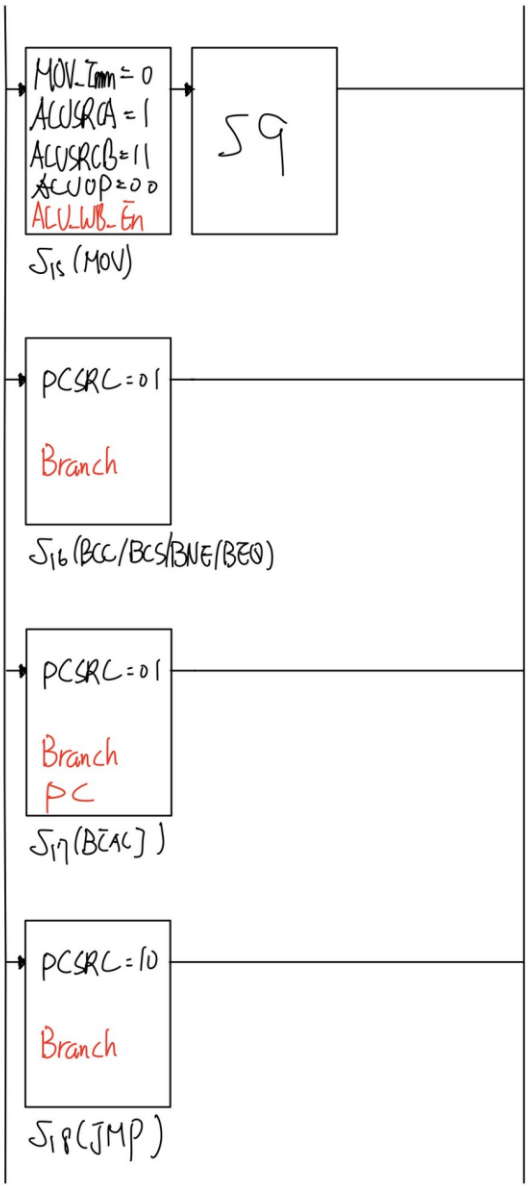
	B	C	D	E	F	G	H
1	PC	PC_SEL	TEST_ADDR_SEL	TEST_DATA_SEL	MEM_ENABLE	MEM_RF_ENABLE0	MEM_RF_ENABLE REA
2		T0	T1	T2	T3	T4	
3		Intruction Fetch	Intruction Decode	ALU Execution	Memory Address	Write Back	
4		sel:	sel:	sel:			
5		PC_SEL = 0	PC_SEL	PC_SEL	PC_SEL	PC_SEL	
6		READ_ADDR_SEL	READ_ADDR_SEL	READ_ADDR_SEL	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	

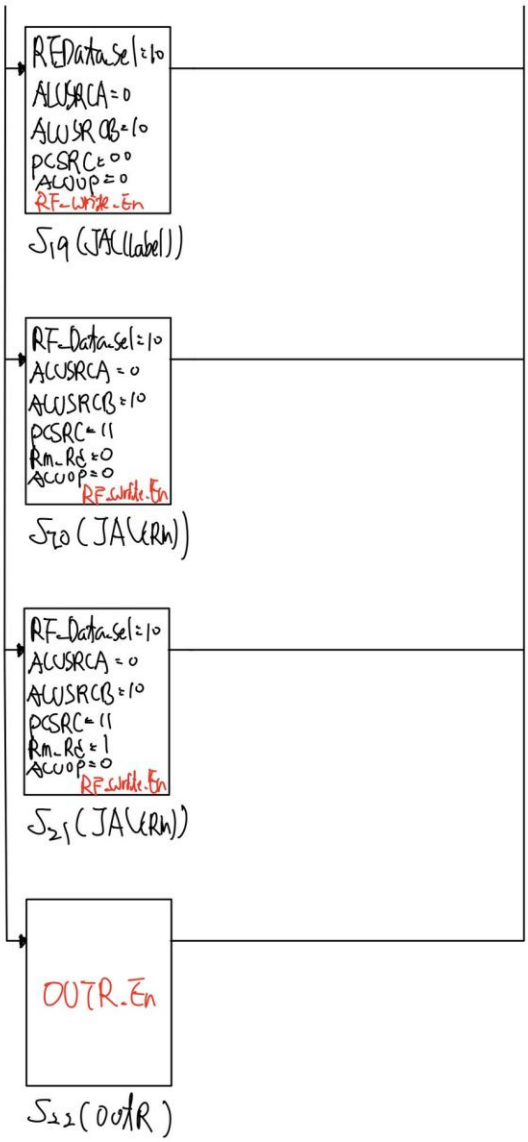
流程圖(這邊的信號為我實際使用的信號，與流程圖有些微差異):




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。

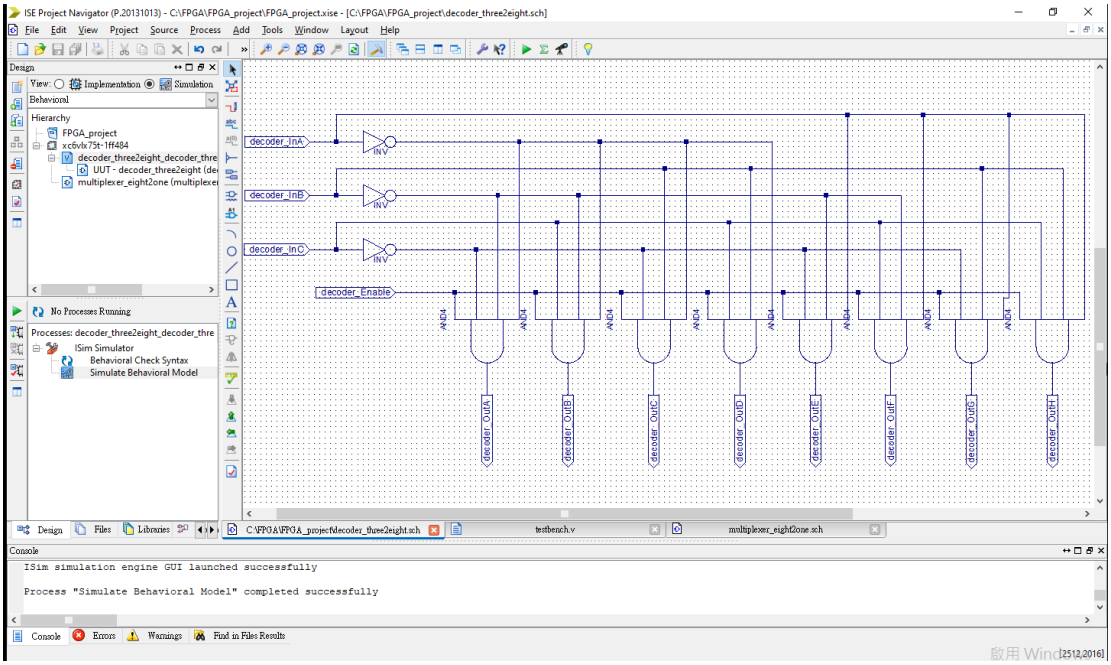
now state					signal																		
S4	S3	S2	S1	S0	PC	MEM_enable	MEM_RF_enable	branch	RF_Write_enable	RF_ALU_enable0/1	ALU_WB_enable	Rm_Rd	OutR	RF_data_sel	Read_addr_sel	PC_sel	Mov_imm	LLI_LHI	ALUSRCA	ALUSRCB	WB_data_sel	Pc_src	ALUOP
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	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透過 PLA(AND、OR)的方式將邏輯閘畫出來了。

Implement schematic entry

1. 3-to-8 decoder

Schematic:



Testbench:

// Verilog test fixture created from schematic C:\FPGA\FPGA_project\decoder_328.sch - Fri Mar 14 15:40:47 2025

```
`timescale 1ns / 1ps

module decoder_328_decoder_328_sch_tb();

// Inputs
    reg decoder_Enable;
    reg decoder_In0;
    reg decoder_In1;
    reg decoder_In2;

// Output
    wire decoder_Out0;
    wire decoder_Out3;
    wire decoder_Out5;
    wire decoder_Out6;
    wire decoder_Out7;
```

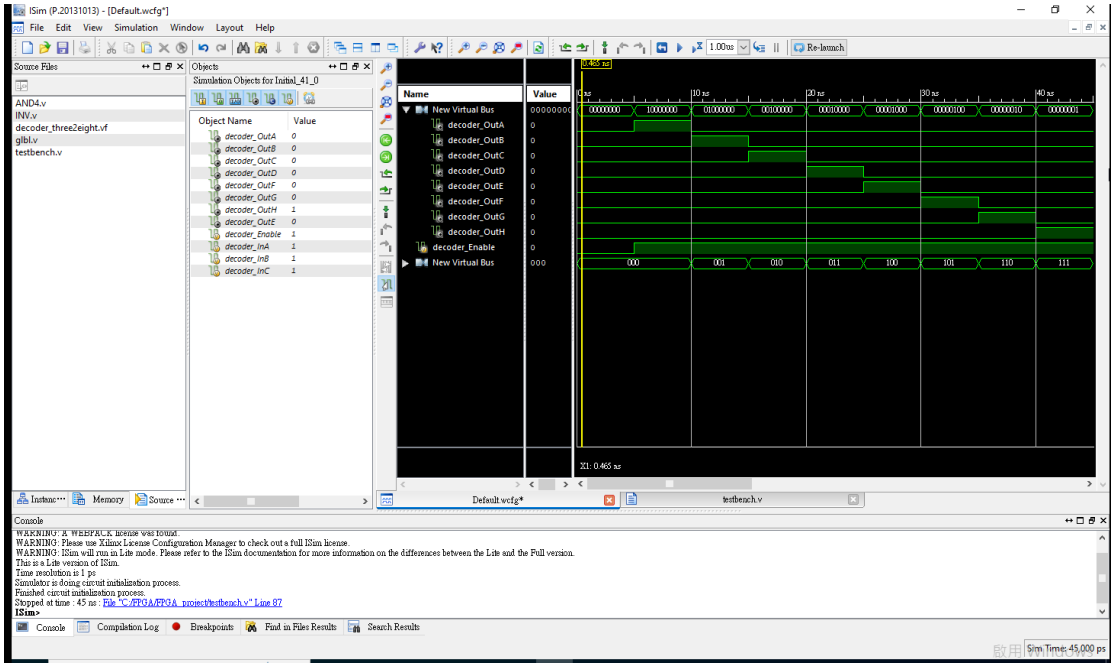
```
	wire decoder_Out1;
	wire decoder_Out2;
	wire decoder_Out4;

// Bidirs

// Instantiate the UUT
	decoder_328 UUT (
		.decoder_Out0(decoder_Out0),
		.decoder_Out3(decoder_Out3),
		.decoder_Out5(decoder_Out5),
		.decoder_Out6(decoder_Out6),
		.decoder_Out7(decoder_Out7),
		.decoder_Out1(decoder_Out1),
		.decoder_Out2(decoder_Out2),
		.decoder_Out4(decoder_Out4),
		.decoder_Enable(decoder_Enable),
		.decoder_In0(decoder_In0),
		.decoder_In1(decoder_In1),
		.decoder_In2(decoder_In2)
	);
// Initialize Inputs
	initial begin
		decoder_Enable = 0;
		decoder_In2 = 0;
		decoder_In1 = 0;
		decoder_In0 = 0;
		#5;
		decoder_Enable = 1;
		decoder_In2 = 0;
		decoder_In1 = 0;
		decoder_In0 = 0;
		#5;
		decoder_Enable = 1;
		decoder_In2 = 0;
		decoder_In1 = 0;
		decoder_In0 = 1;
		#5;
```

```
        decoder_Enable = 1;
        decoder_In2 = 0;
        decoder_In1 = 1;
        decoder_In0 = 0;
        #5;
        decoder_Enable = 1;
        decoder_In2 = 0;
        decoder_In1 = 1;
        decoder_In0 = 1;
        #5;
        decoder_Enable = 1;
        decoder_In2 = 1;
        decoder_In1 = 0;
        decoder_In0 = 0;
        #5;
        decoder_Enable = 1;
        decoder_In2 = 1;
        decoder_In1 = 0;
        decoder_In0 = 1;
        #5;decoder_Enable = 1;
        decoder_In2 = 1;
        decoder_In1 = 1;
        decoder_In0 = 0;
        #5;decoder_Enable = 1;
        decoder_In2 = 1;
        decoder_In1 = 1;
        decoder_In0 = 1;
        #5;
        $finish;
    end
endmodule
```

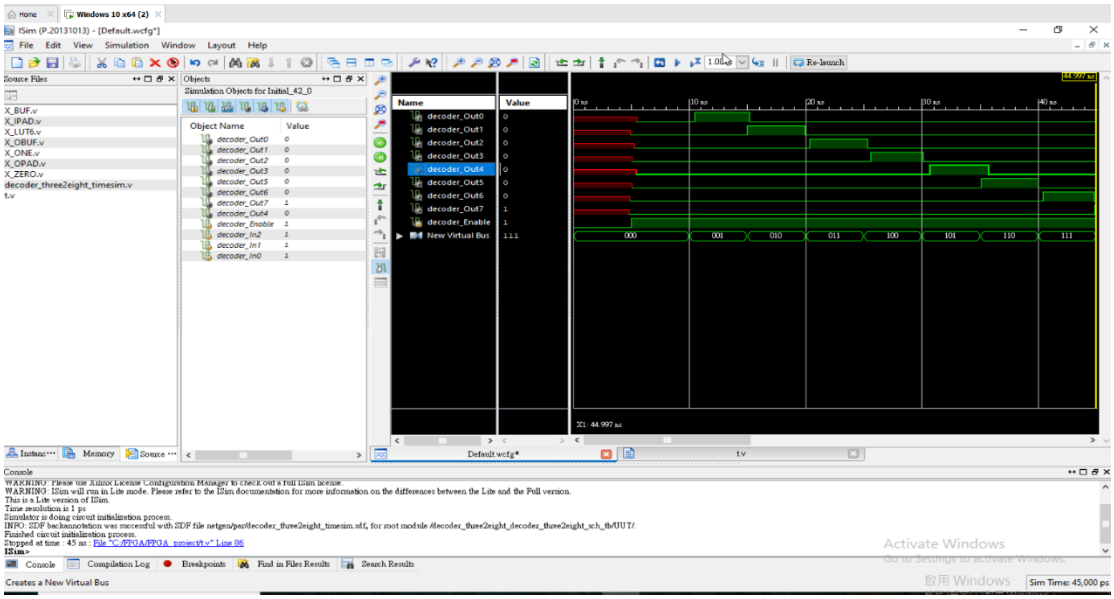
Simulation:

根據 3 bit 選擇的位置輸出的 out 改為 0。

Behavior:

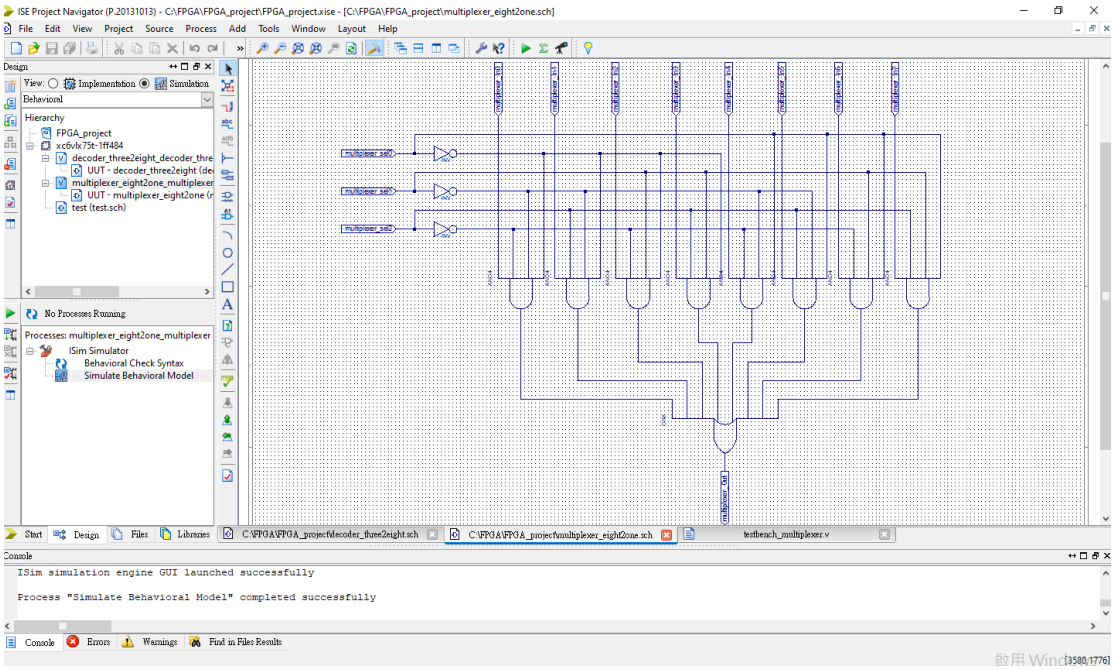

ISim (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
Source Files
AND4.v
INV.v
decoder_three2eight.vf
glbl.v
testbench.v
Objects
Simulation Objects for Initial_41_0
Object Name
Value
decoder_OutA
decoder_OutB
decoder_OutC
decoder_OutD
decoder_OutE
decoder_OutF
decoder_OutG
decoder_OutH
decoder_Enable
decoder_inA
decoder_inB
decoder_inC
Name
Value
New Virtual Bus
decoder_Enable
New Virtual Bus
X1: 0.465 ns
Default.wcfg*
testbench.v
Console
WARNING: Please use Xilinx License Configuration Manager to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
Stopped at time : 45 ns : File "C:/FPGA/FPGA_project/testbench.v" Line 87
ISim>
Console
Compilation Log
Breakpoints
Find in Files Results
Search Results
Sim Time: 45,000 ps

Post-Route:


ISim (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
Source Files
X_BUF.v
X_IPAD.v
X_LUT6.v
X_OBUF.v
X_ONE.v
X_OPAD.v
X_ZERO.v
decoder_three2eight_timesim.v
t.v
Objects
Simulation Objects for Initial_42_0
Object Name
Value
decoder_Out0
decoder_Out1
decoder_Out2
decoder_Out3
decoder_Out4
decoder_Out5
decoder_Out6
decoder_Out7
decoder_Enable
decoder_In2
decoder_In1
decoder_In0
Name
Value
New Virtual Bus
X1: 44.997 ns
Default.wcfg*
t.v
Console
WARNING: Please use Xilinx License Configuration Manager to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
INFO: SDF backannotation was successful with SDF file netgen/par/decoder_three2eight_timesim.sdf, for root module /decoder_three2eight_decoder_three2eight_sch_tb/UUT/.
Finished circuit initialization process.
Stopped at time : 45 ns : File "C:/FPGA/FPGA_project/t.v" Line 86
ISim>
Console
Compilation Log
Breakpoints
Find in Files Results
Search Results
Creates a New Virtual Bus
Sim Time: 45,000 ps

2. 8-to-1 MUX

Schematic:



Testbench:

// Verilog test fixture created from schematic C:\FPGA\FPGA_project\multiplexer_eight2one.sch - Wed Mar 12 15:28:15 2025

```
`timescale 1ns / 1ps

module multiplexer_eight2one_multiplexer_eight2one_sch_tb();

// Inputs
    reg multiplexer_sel0;
    reg multiplexer_sel1;
    reg multiplexer_sel2;
    reg multiplexer_In0;
    reg multiplexer_In1;
    reg multiplexer_In2;
    reg multiplexer_In3;
    reg multiplexer_In4;
    reg multiplexer_In5;
    reg multiplexer_In6;
    reg multiplexer_In7;

// Output
```

```
    wire multiplexer_Out;

// Bidirs

// Instantiate the UUT
    multiplexer_eight2one UUT (
        .multiplexer_sel0(multiplexer_sel0),
        .multiplexer_sel1(multiplexer_sel1),
        .multiplexer_sel2(multiplexer_sel2),
        .multiplexer_In0(multiplexer_In0),
        .multiplexer_In1(multiplexer_In1),
        .multiplexer_In2(multiplexer_In2),
        .multiplexer_In3(multiplexer_In3),
        .multiplexer_In4(multiplexer_In4),
        .multiplexer_In5(multiplexer_In5),
        .multiplexer_In6(multiplexer_In6),
        .multiplexer_In7(multiplexer_In7),
        .multiplexer_Out(multiplexer_Out)
    );
// Initialize Inputs
    initial begin
        multiplexer_sel0 = 0;
        multiplexer_sel1 = 0;
        multiplexer_sel2 = 0;
        multiplexer_In0 = 1;
        multiplexer_In1 = 0;
        multiplexer_In2 = 0;
        multiplexer_In3 = 0;
        multiplexer_In4 = 0;
        multiplexer_In5 = 0;
        multiplexer_In6 = 0;
        multiplexer_In7 = 0;
        #5;
        multiplexer_sel0 = 0;
        multiplexer_sel1 = 0;
        multiplexer_sel2 = 1;
        multiplexer_In0 = 0;
        multiplexer_In1 = 1;
```

```
multiplexer_In2 = 0;
multiplexer_In3 = 0;
multiplexer_In4 = 0;
multiplexer_In5 = 0;
multiplexer_In6 = 0;
multiplexer_In7 = 0;
#5;
multiplexer_sel0 = 0;
multiplexer_sel1 = 1;
multiplexer_sel2 = 0;
multiplexer_In0 = 0;
multiplexer_In1 = 0;
multiplexer_In2 = 1;
multiplexer_In3 = 0;
multiplexer_In4 = 0;
multiplexer_In5 = 0;
multiplexer_In6 = 0;
multiplexer_In7 = 0;
#5;
multiplexer_sel0 = 0;
multiplexer_sel1 = 1;
multiplexer_sel2 = 1;
multiplexer_In0 = 0;
multiplexer_In1 = 0;
multiplexer_In2 = 0;
multiplexer_In3 = 1;
multiplexer_In4 = 0;
multiplexer_In5 = 0;
multiplexer_In6 = 0;
multiplexer_In7 = 0;
#5;
multiplexer_sel0 = 1;
multiplexer_sel1 = 0;
multiplexer_sel2 = 0;
multiplexer_In0 = 0;
multiplexer_In1 = 0;
multiplexer_In2 = 0;
multiplexer_In3 = 0;
```

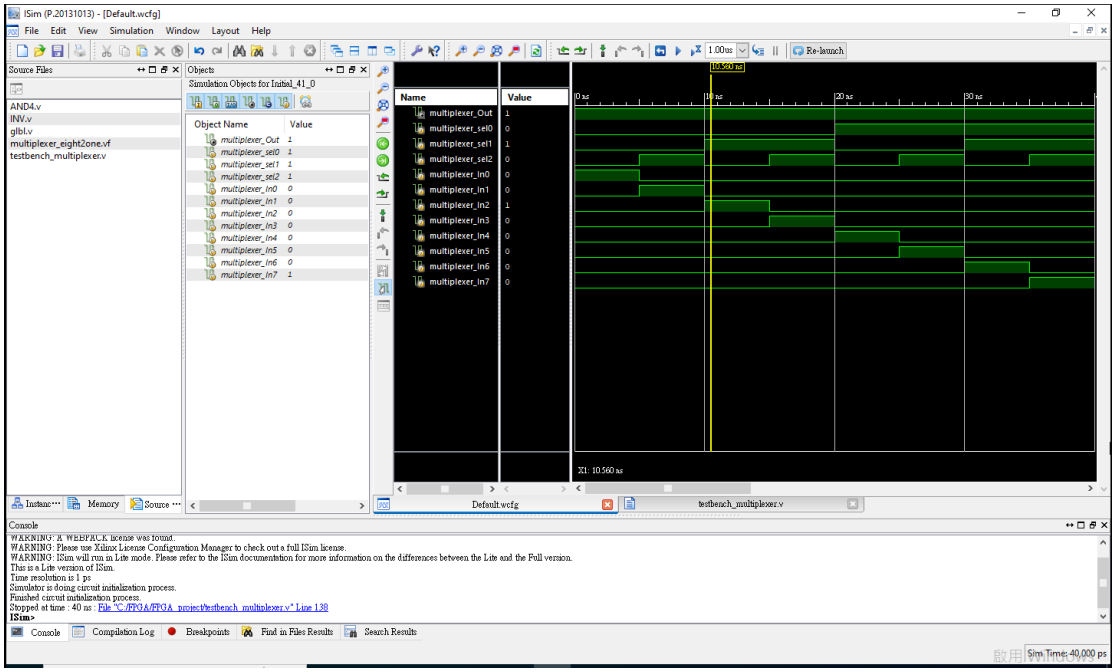
```
multiplexer_In4 = 1;
multiplexer_In5 = 0;
multiplexer_In6 = 0;
multiplexer_In7 = 0;
#5;
multiplexer_sel0 = 1;
multiplexer_sel1 = 0;
multiplexer_sel2 = 1;
multiplexer_In0 = 0;
multiplexer_In1 = 0;
multiplexer_In2 = 0;
multiplexer_In3 = 0;
multiplexer_In4 = 0;
multiplexer_In5 = 1;
multiplexer_In6 = 0;
multiplexer_In7 = 0;
#5;
multiplexer_sel0 = 1;
multiplexer_sel1 = 1;
multiplexer_sel2 = 0;
multiplexer_In0 = 0;
multiplexer_In1 = 0;
multiplexer_In2 = 0;
multiplexer_In3 = 0;
multiplexer_In4 = 0;
multiplexer_In5 = 0;
multiplexer_In6 = 1;
multiplexer_In7 = 0;
#5;
multiplexer_sel0 = 1;
multiplexer_sel1 = 1;
multiplexer_sel2 = 1;
multiplexer_In0 = 0;
multiplexer_In1 = 0;
multiplexer_In2 = 0;
multiplexer_In3 = 0;
multiplexer_In4 = 0;
multiplexer_In5 = 0;
```

```
        multiplexer_In6 = 0;
        multiplexer_In7 = 1;
        #5;
        $finish;
    end
endmodule
```

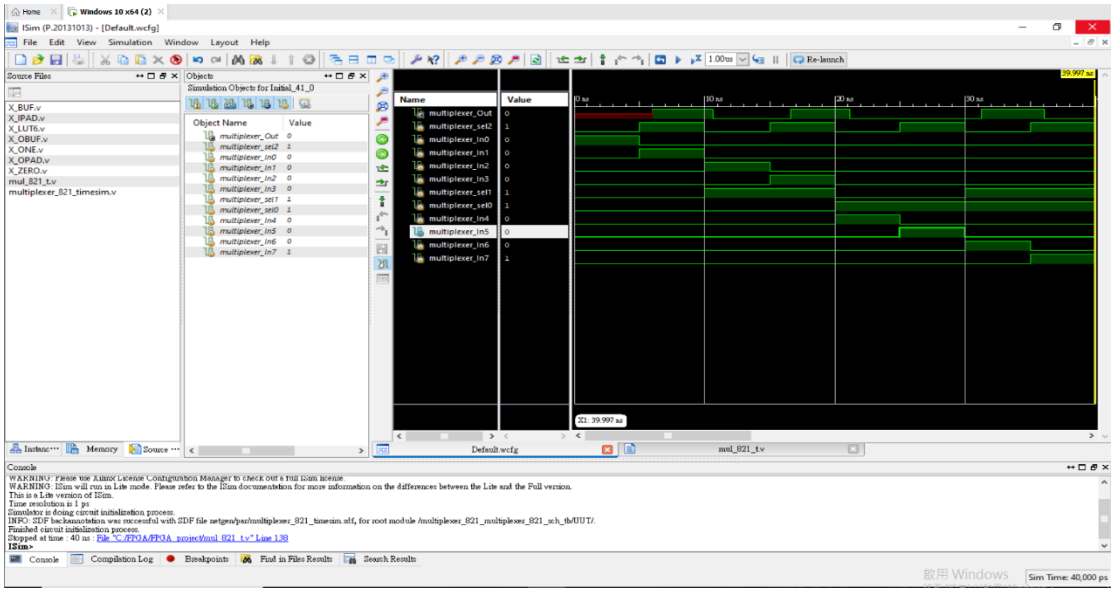
Simulation:

根據所選的 sel 線去判斷要輸出哪一個 out。

Behavior:

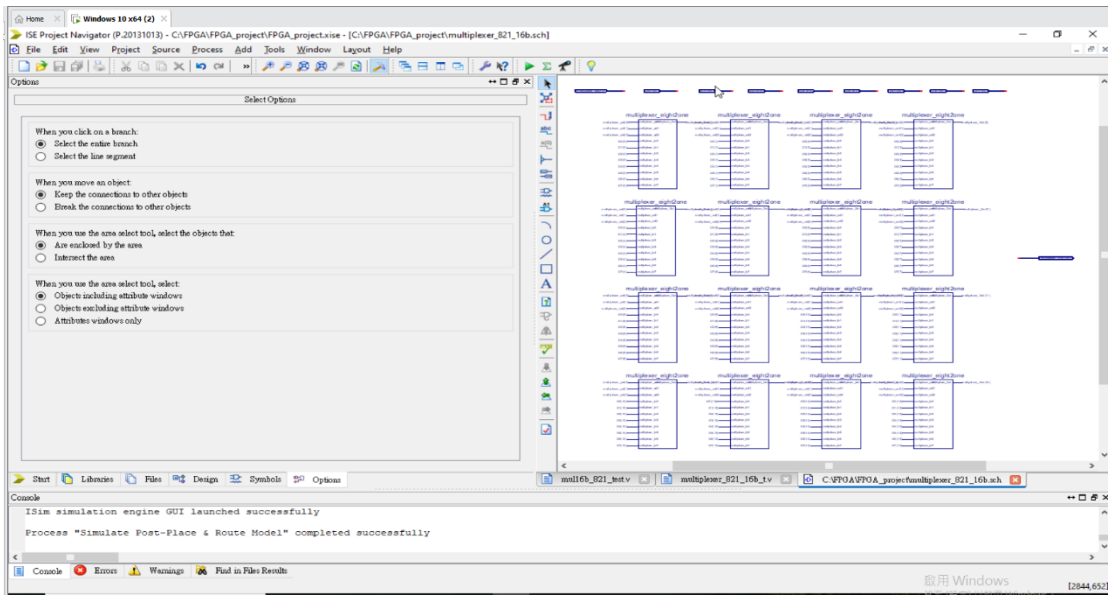


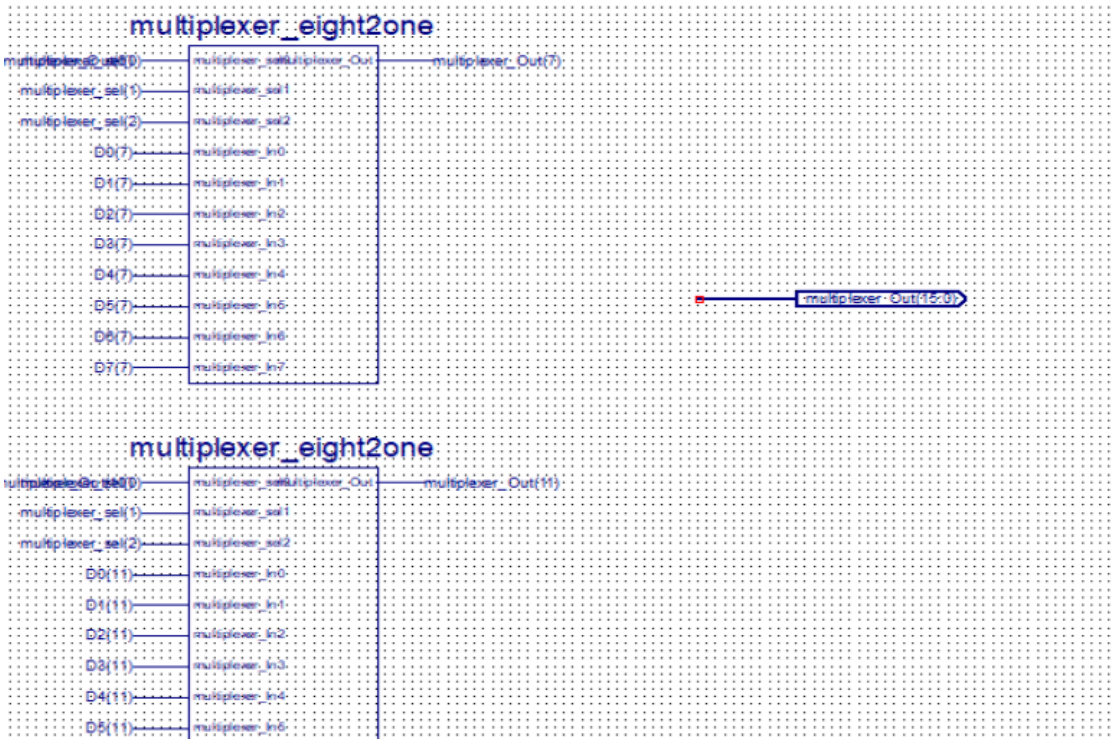
Post-Route:



3. 16 bit 8-to-1 mux

Schematic:





Testbench:

```
// Verilog test fixture created from schematic
C:\FPGA\FPGA_project\multiplexer_821_16b.sch - Fri Mar 14 12:48:55 2025

`timescale 1ns / 1ps

module multiplexer_821_16b_multiplexer_821_16b_sch_tb();
```

```
// Inputs
   reg [2:0] multiplexer_sel;
   reg [15:0] D0;
   reg [15:0] D1;
   reg [15:0] D2;
   reg [15:0] D3;
   reg [15:0] D4;
   reg [15:0] D5;
   reg [15:0] D6;
   reg [15:0] D7;

// Output
   wire [15:0] multiplexer_Out;

// Bidirs

// Instantiate the UUT
   multiplexer_821_16b UUT (
         .multiplexer_sel(multiplexer_sel),
         .D0(D0),
         .D1(D1),
         .D2(D2),
         .D3(D3),
         .D4(D4),
         .D5(D5),
         .D6(D6),
         .D7(D7),
         .multiplexer_Out(multiplexer_Out)
   );
// Initialize Inputs
    initial begin
         multiplexer_sel = 3'b000;
         D0 = 1;
         D1 = 2;
         D2 = 3;
         D3 = 4;
         D4 = 5;
```

```
D5 = 6;
D6 = 7;
D7 = 8;
#5;
multiplexer_sel = 3'b001;
D0 = 1;
D1 = 2;
D2 = 3;
D3 = 4;
D4 = 5;
D5 = 6;
D6 = 7;
D7 = 8;
#5;
multiplexer_sel = 2;
D0 = 1;
D1 = 2;
D2 = 3;
D3 = 4;
D4 = 5;
D5 = 6;
D6 = 7;
D7 = 8;
#5;
multiplexer_sel = 3;
D0 = 1;
D1 = 2;
D2 = 3;
D3 = 4;
D4 = 5;
D5 = 6;
D6 = 7;
D7 = 8;
#5;
multiplexer_sel = 4;
D0 = 1;
D1 = 2;
D2 = 3;
```

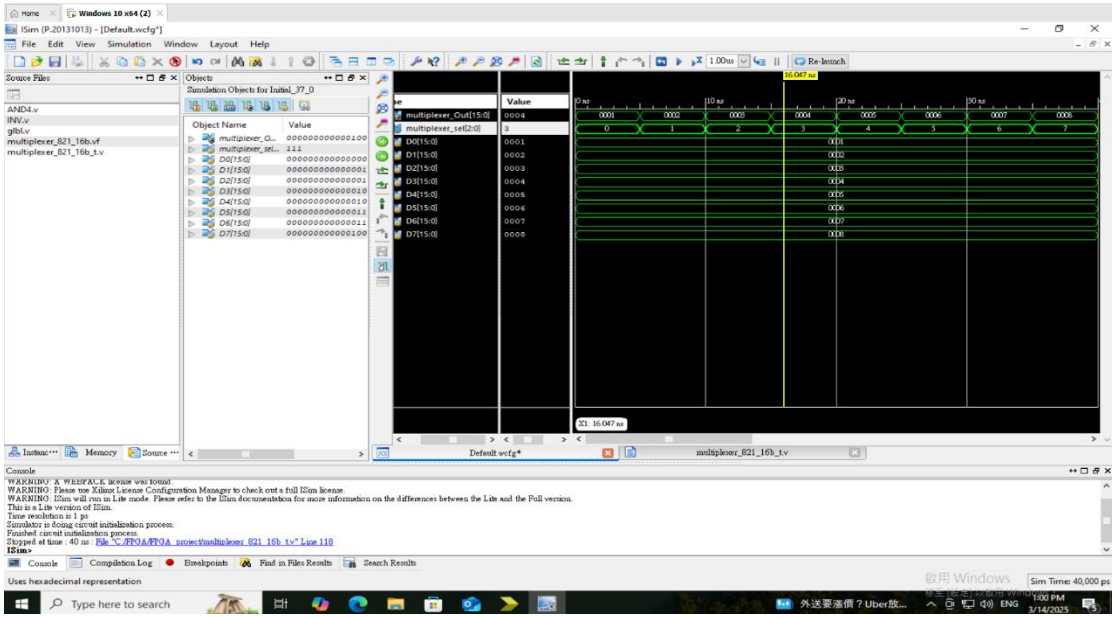
```
D3 = 4;
D4 = 5;
D5 = 6;
D6 = 7;
D7 = 8;
#5;
multiplexer_sel = 5;
D0 = 1;
D1 = 2;
D2 = 3;
D3 = 4;
D4 = 5;
D5 = 6;
D6 = 7;
D7 = 8;
#5;
multiplexer_sel = 6;
D0 = 1;
D1 = 2;
D2 = 3;
D3 = 4;
D4 = 5;
D5 = 6;
D6 = 7;
D7 = 8;
#5;
multiplexer_sel = 7;
D0 = 1;
D1 = 2;
D2 = 3;
D3 = 4;
D4 = 5;
D5 = 6;
D6 = 7;
D7 = 8;
#5;
$finish;
```

```
    end
endmodule
```

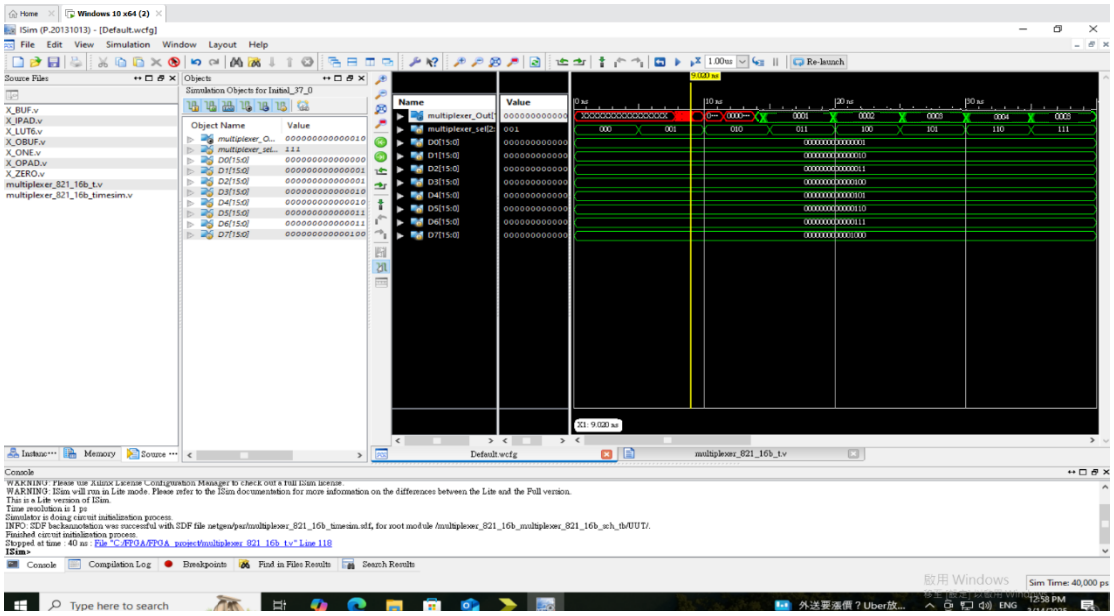
Simulation:

改成 16bit 的方式

Behavior:

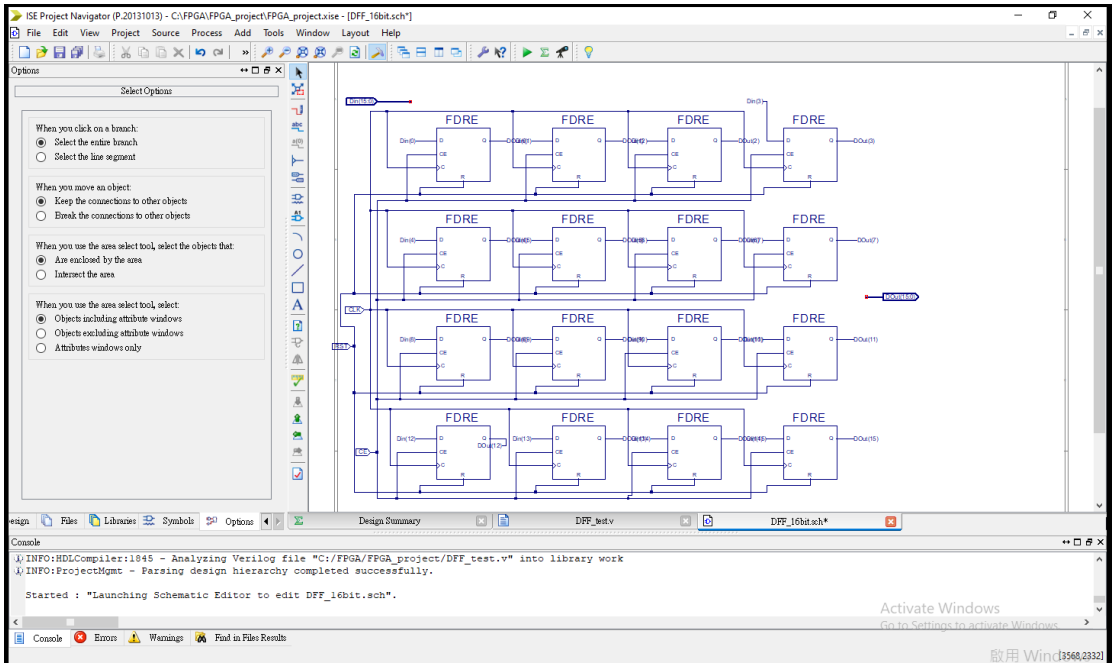


Post-Route:



4. 16 bit DFF

Schematic:



Testbench:

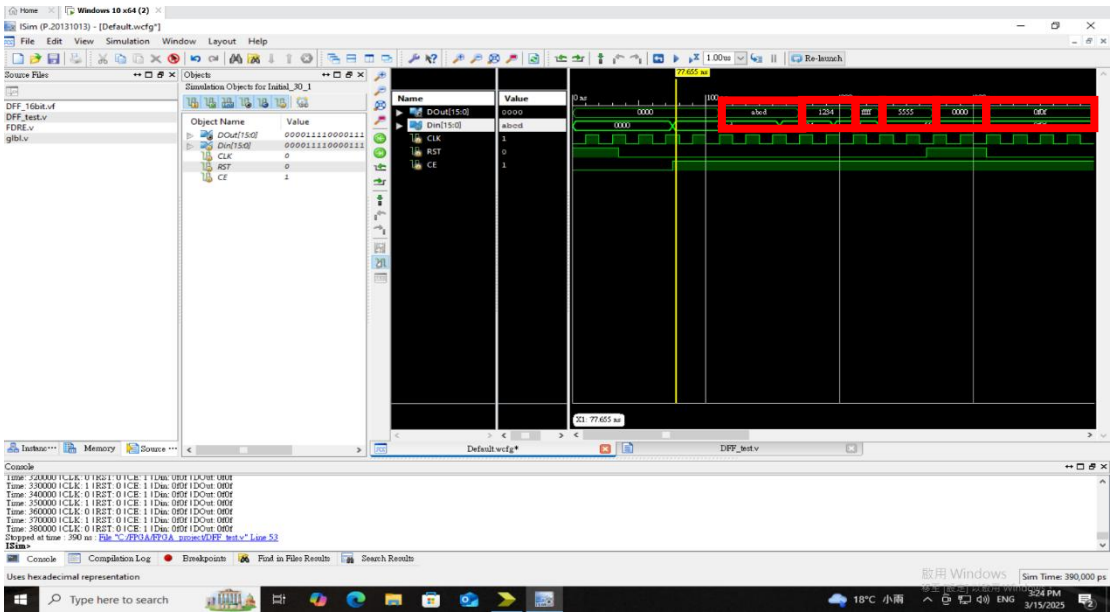
// Verilog test fixture created from schematic C:\FPGA\FPGA_project\DFF_16bit.sch - Thu Mar 13 15:23:26 2025

```
`timescale 1ns / 1ps

module DFF_16bit_DFF_16bit_sch_tb();

// Inputs
    reg [15:0] Din;
    reg CLK;
    reg RST;
    reg CE;

// Output
    wire [15:0] DOut;

// Bidirs

// Instantiate the UUT
    DFF_16bit UUT (
            .Din(Din),
```

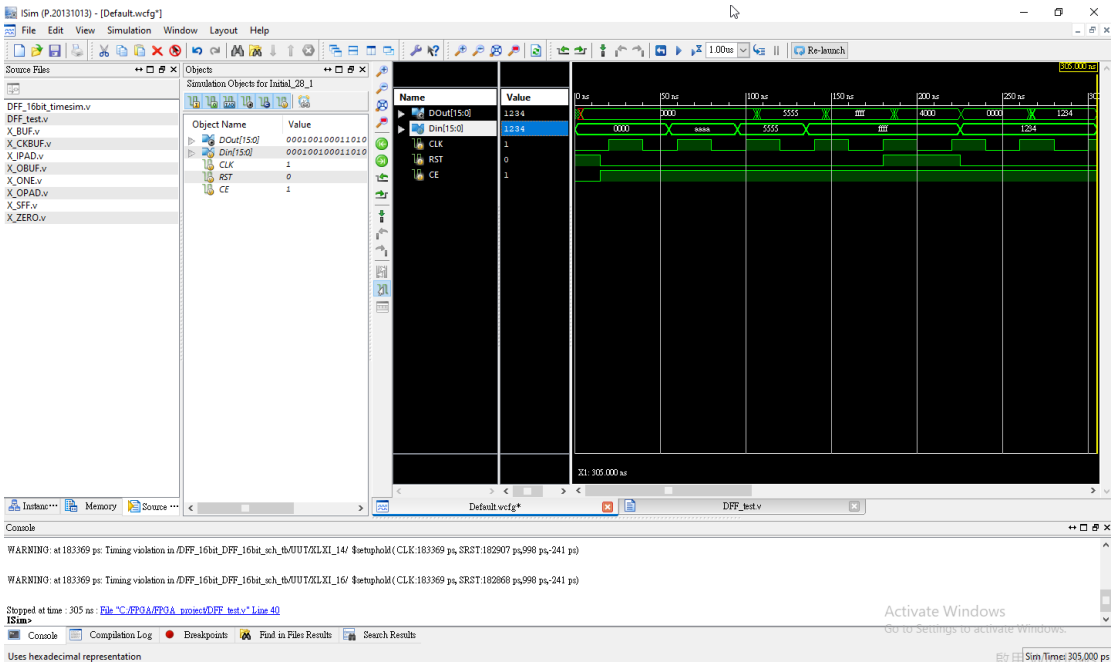
```
            .CLK(CLK),
            .RST(RST),
            .DOut(DOut),
            .CE(CE)
    );
// Initialize Inputs
        always #10 CLK = ~CLK;

    // 測試流程
    initial begin
        // 初始化
        CLK = 0;
        RST = 1;    // 先重置
        CE = 0;
        Din = 16'h0000;

        #35 RST = 0; // 取消 Reset

        // 測試 1: 確認 Reset 是否清零
        #40 Din = 16'hABCD; CE = 1;   // 設定輸入值
            #40 CE = 1;
        #40 Din = 16'h1234; CE = 1;   // 重新開啟 CE，並改變輸入

        // 測試 2: 觀察是否保持數值
        #40 Din = 16'hFFFF;    // Clock Enable 關閉，輸出應保持不變
        #35   Din = 16'h5555;   // 再次開啟 Clock Enable，看看數據是否正確變化

        // 測試 3: Reset 影響
        #35 RST = 1; // 施加 Reset，應清除輸出
        #45 RST = 0; Din = 16'h0F0F; CE = 1; // 取消 Reset，設定新值

        // 測試結束
        #80 $finish;
    end

    // 監視訊號變化
    initial begin
        $monitor("Time: %0t | CLK: %b | RST: %b | CE: %b | Din: %h | DOut: %h",
```

```
                    $time, CLK, RST, CE, Din, DOut);
    end
endmodule
```

Behavior:

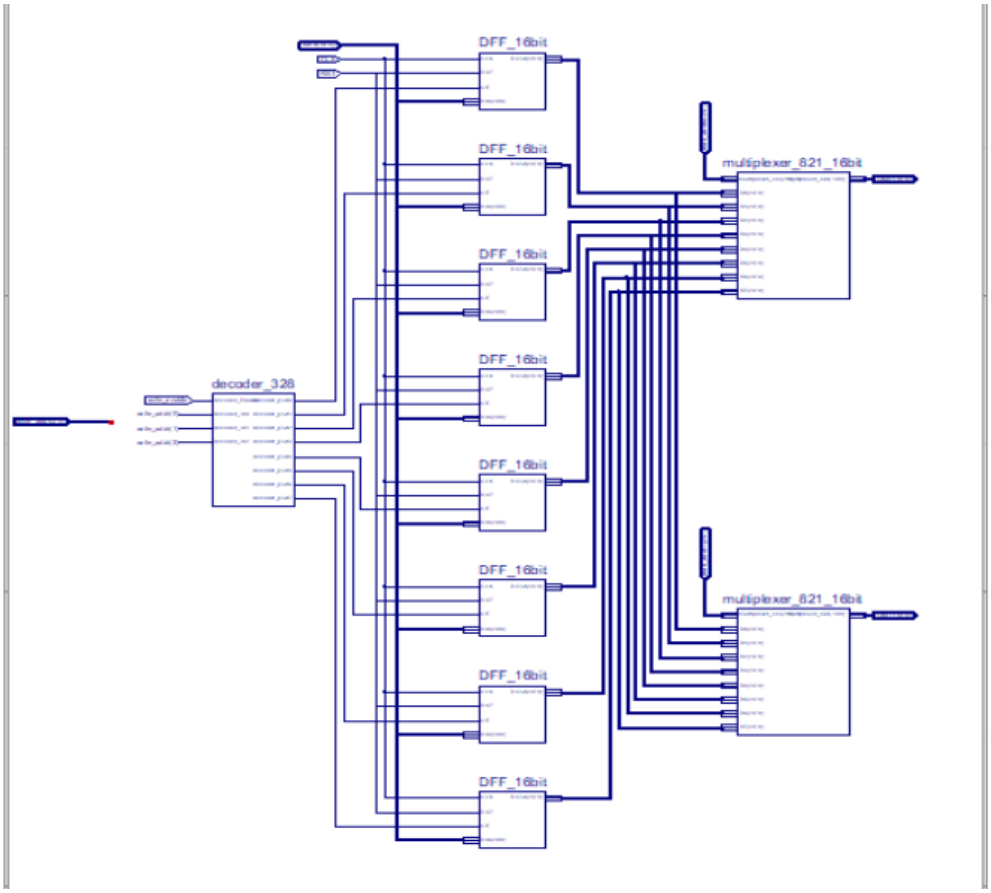


Post-Route:



5. 16 bit Eight-Register Register File

Schematic:



Testbench:

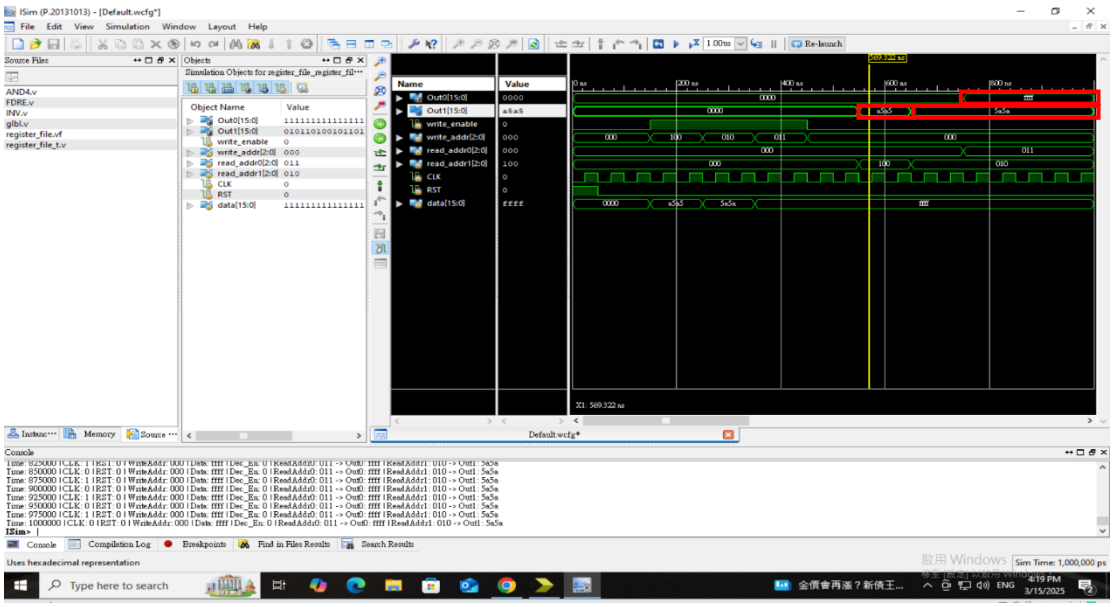
```
// Verilog test fixture created from schematic C:\FPGA\FPGA_project\register_file.sch - Fri Mar 14
15:10:04 2025

`timescale 1ns / 1ps

module register_file_register_file_sch_tb();

// Inputs
    reg write_enable;
    reg [2:0] write_addr;
    reg [2:0] read_addr0;
    reg [2:0] read_addr1;
    reg CLK;
    reg RST;
    reg [15:0] data;
```

```
// Output
    wire [15:0] Out0;
    wire [15:0] Out1;

// Bidirs

// Instantiate the UUT
    register_file UUT (
            .write_enable(write_enable),
            .write_addr(write_addr),
            .read_addr0(read_addr0),
            .read_addr1(read_addr1),
            .Out0(Out0),
            .Out1(Out1),
            .CLK(CLK),
            .RST(RST),
            .data(data)
    );
// Initialize Inputs
   always #25 CLK = ~CLK;   // 每 5ns 反轉，產生 10ns 週期的時鐘

     // 測試流程
     initial begin
          // 初始化
          CLK = 0;
          RST = 1;
          write_enable = 0;
          write_addr = 3'b000;
          read_addr0 = 3'b000;
          read_addr1 = 3'b000;
          data = 16'h0000;

          #50 RST = 0;   // 取消 Reset

          // 測試寫入資料
          #100 write_enable = 1; write_addr = 3'b100; data = 16'hA5A5; // 寫入 0xA5A5 到寄存器
1
          #100 write_enable = 1; write_addr = 3'b010; data = 16'h5A5A; // 寫入 0x5A5A 到寄存器
```

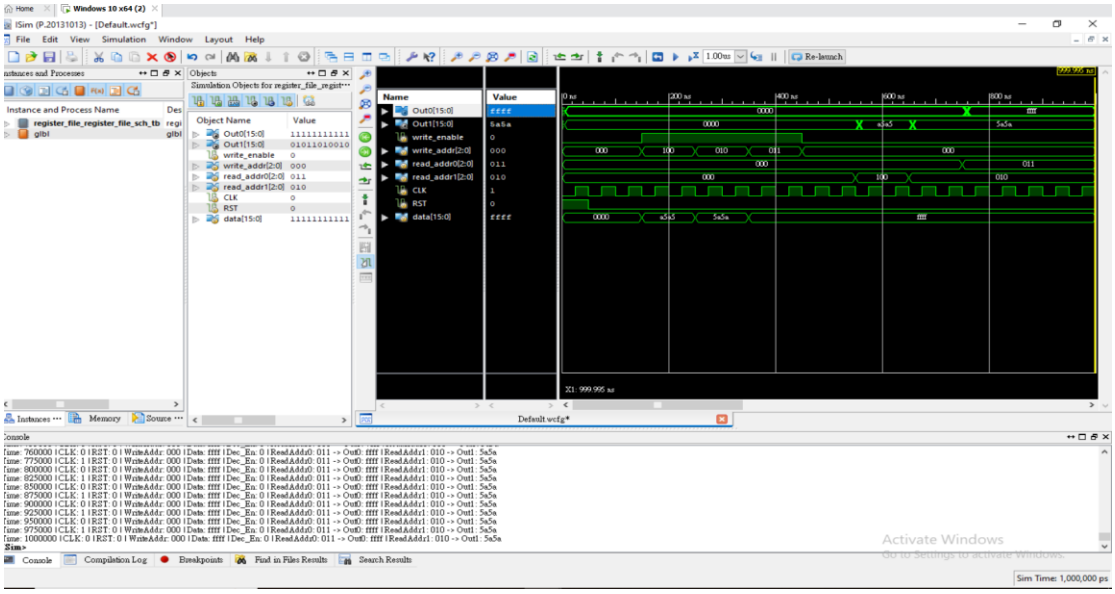
2

```
        #100 write_enable = 1; write_addr = 3'b011; data = 16'hFFFF; //
            #100 write_enable = 0; write_addr = 3'b000;
        // 測試讀取
        #100 read_addr1 = 3'b100; // 讀取寄存器 1，應該是 0xA5A5
        #100 read_addr1 = 3'b010; // 讀取寄存器 2，應該是 0x5A5A
        #100 read_addr0 = 3'b011; // 讀取寄存器 3，應該是 0xFFFF

        // 測試完成
        #1000 $finish;
    end

    // 監視輸出變化
    initial begin
        $monitor("Time: %0t | CLK: %b | RST: %b | WriteAddr: %b | Data: %h | Dec_En: %b |
ReadAddr0: %b -> Out0: %h | ReadAddr1: %b -> Out1: %h",
                $time, CLK, RST, write_addr, data, write_enable, read_addr0, Out0, read_addr1,
Out1);
    end
endmodule
```

Behavior:

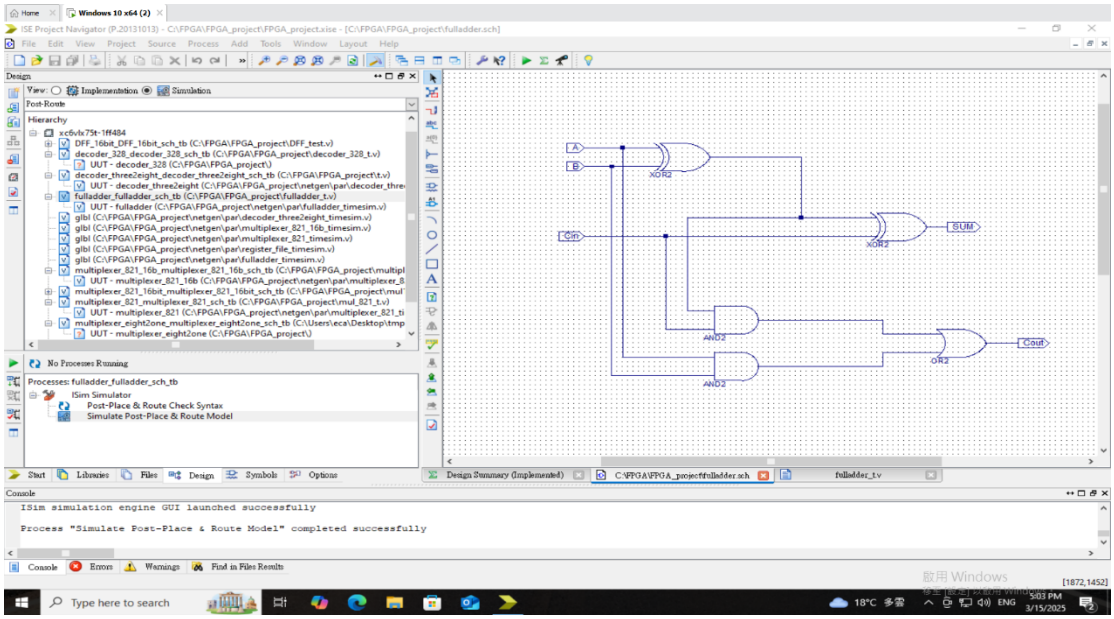


Post-Route:



6. Full adder

Schematic:



Testbench:

// Verilog test fixture created from schematic C:\FPGA\FPGA_project\fulladder.sch - Sat Mar 15 16:55:54 2025

`timescale 1ns / 1ps

module fulladder_fulladder_sch_tb();

// Inputs

reg B;

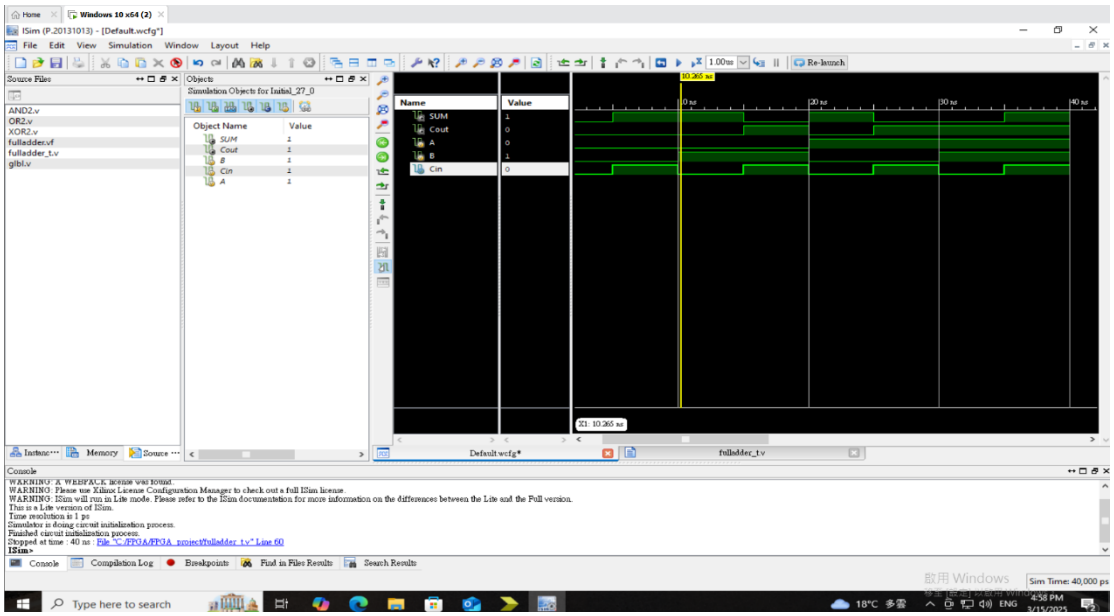
```
    reg Cin;
    reg A;

// Output
    wire SUM;
    wire Cout;

// Bidirs

// Instantiate the UUT
    fulladder UUT (
            .B(B),
            .Cin(Cin),
            .A(A),
            .SUM(SUM),
            .Cout(Cout)
    );
// Initialize Inputs
      initial begin
      A = 0;
      B = 0;
      Cin = 0;
      #5;
      A = 0;
      B = 0;
      Cin = 1;
      #5;
      A = 0;
      B = 1;
      Cin = 0;
      #5;
      A = 0;
      B = 1;
      Cin = 1;
      #5;
      A = 1;
      B = 0;
      Cin = 0;
```

```
        #5;
        A = 1;
        B = 0;
        Cin = 1;
        #5;
        A = 1;
        B = 1;
        Cin = 0;
        #5;
        A = 1;
        B = 1;
        Cin = 1;
        #5;
        $finish;
        end
endmodule
```

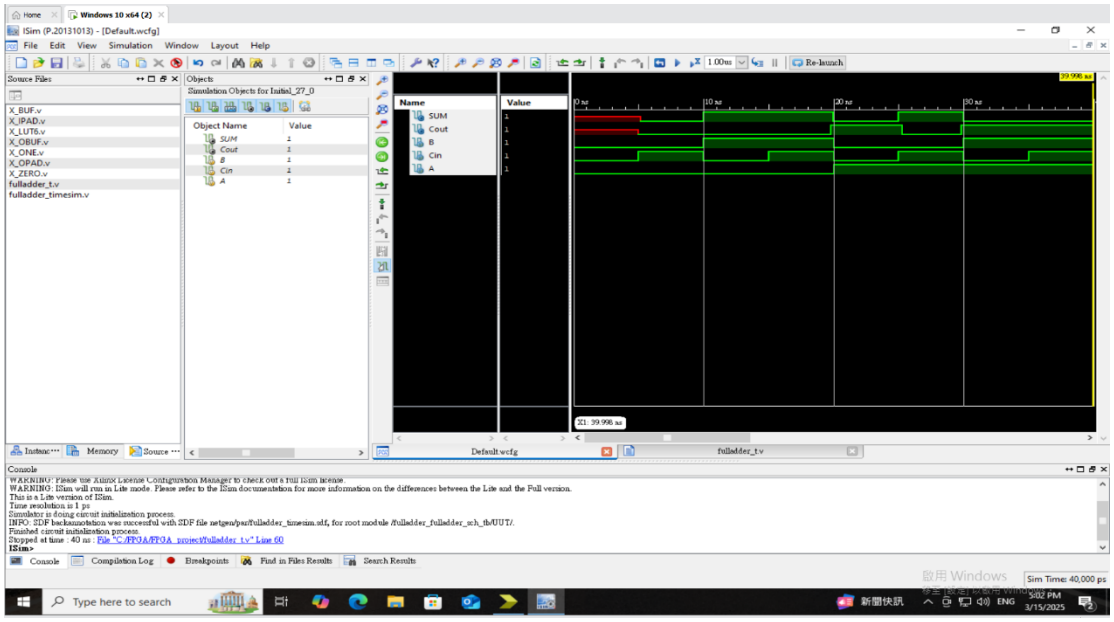
Simulation:

基本加法

Behavior:

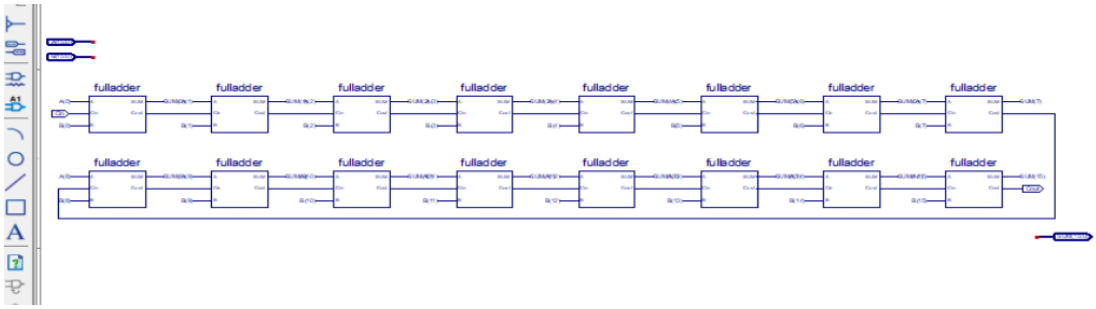


Post-Route:



7. 16 bit full adder

Schematic:



Testbench:

// Verilog test fixture created from schematic C:\FPGA\FPGA_project\fulladder_16b.sch - Sat Mar 15 17:52:51 2025

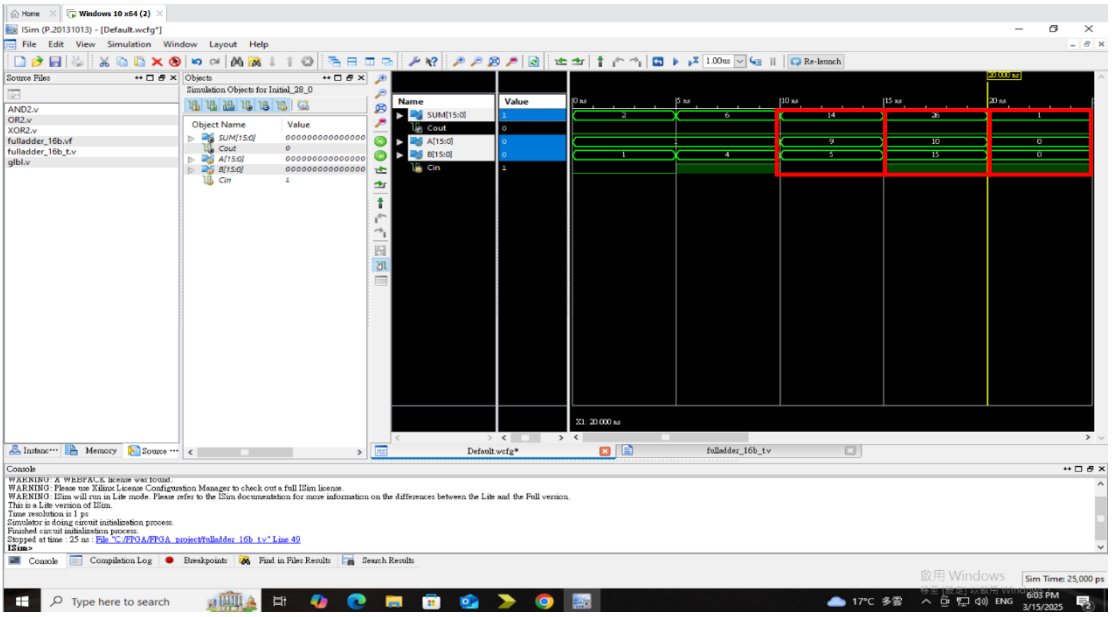
```
`timescale 1ns / 1ps

module fulladder_16b_fulladder_16b_sch_tb();

// Inputs
    reg [15:0] A;
    reg [15:0] B;
    reg Cin;

// Output
    wire [15:0] SUM;
```

```
    wire Cout;

// Bidirs

// Instantiate the UUT
    fulladder_16b UUT (
            .SUM(SUM),
            .A(A),
            .B(B),
            .Cin(Cin),
            .Cout(Cout)
    );
// Initialize Inputs

      initial begin
            A = 1;
            B = 1;
            Cin = 0;
            #5;
            A = 1;
            B = 4;
            Cin = 1;
            #5;
            A = 9;
            B = 5;
            Cin = 0;
            #5;
            A = 10;
            B = 15;
            Cin = 1;
            #5;
            A = 0;
            B = 0;
            Cin = 1;
            #5;
            $finish;
    end
endmodule
```

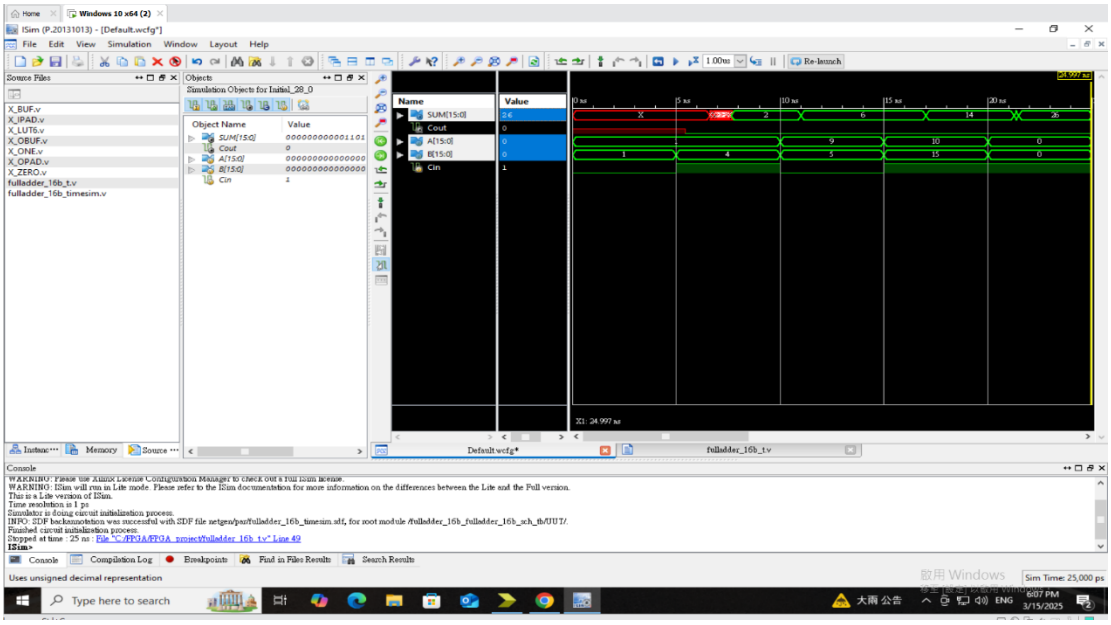
Simulation:

改為 16bit 的加法

Behavior:

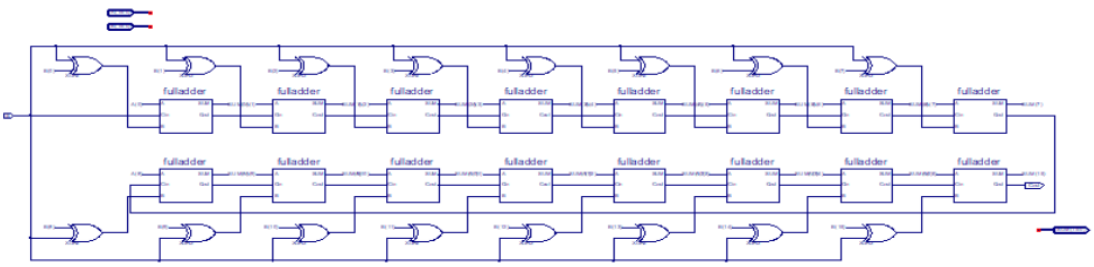


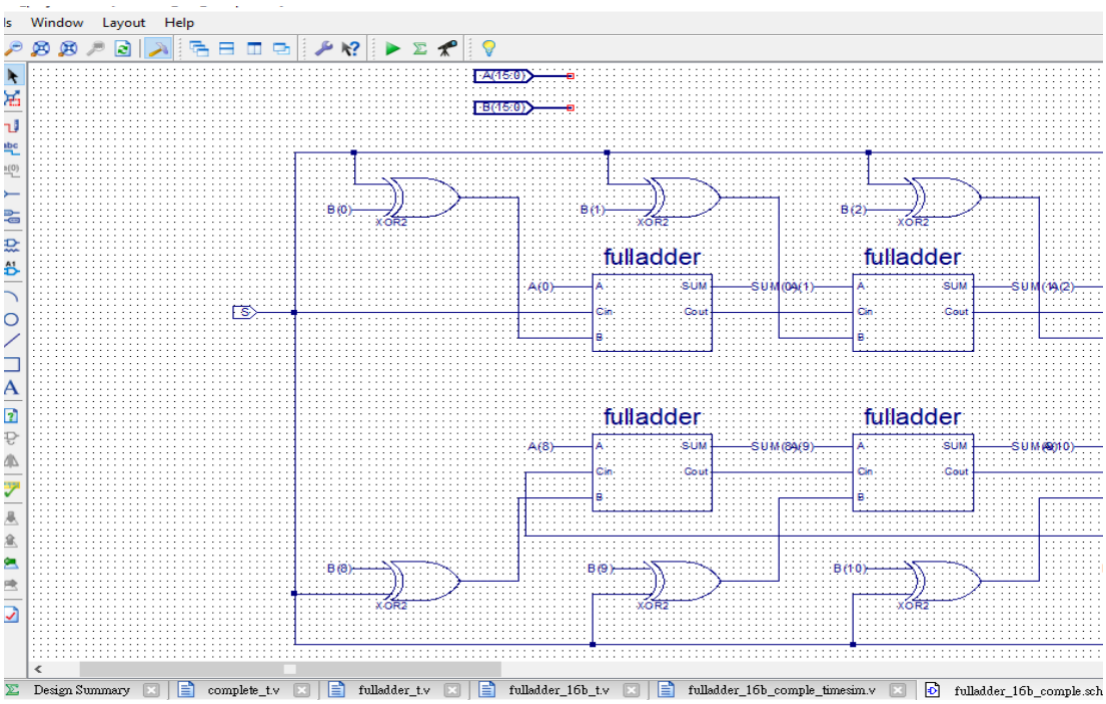
Post-Route:



8. 16 bit full adder(complement)

Schematic:





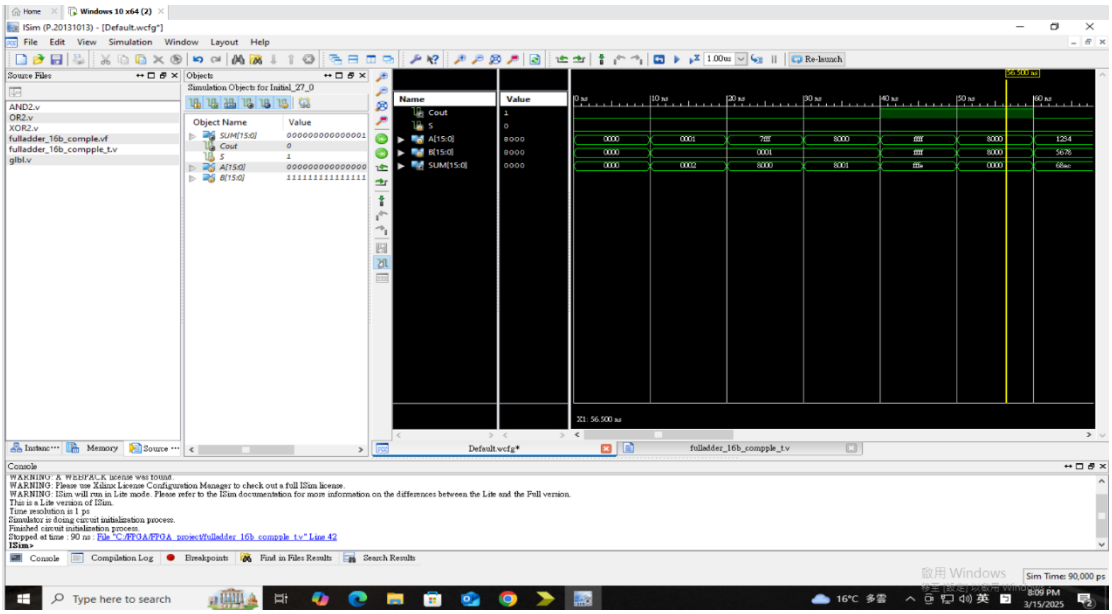
Testbench:

// Verilog test fixture created from schematic C:\FPGA\FPGA_project\fulladder_16b_comple.sch - Sat Mar 15 20:00:39 2025

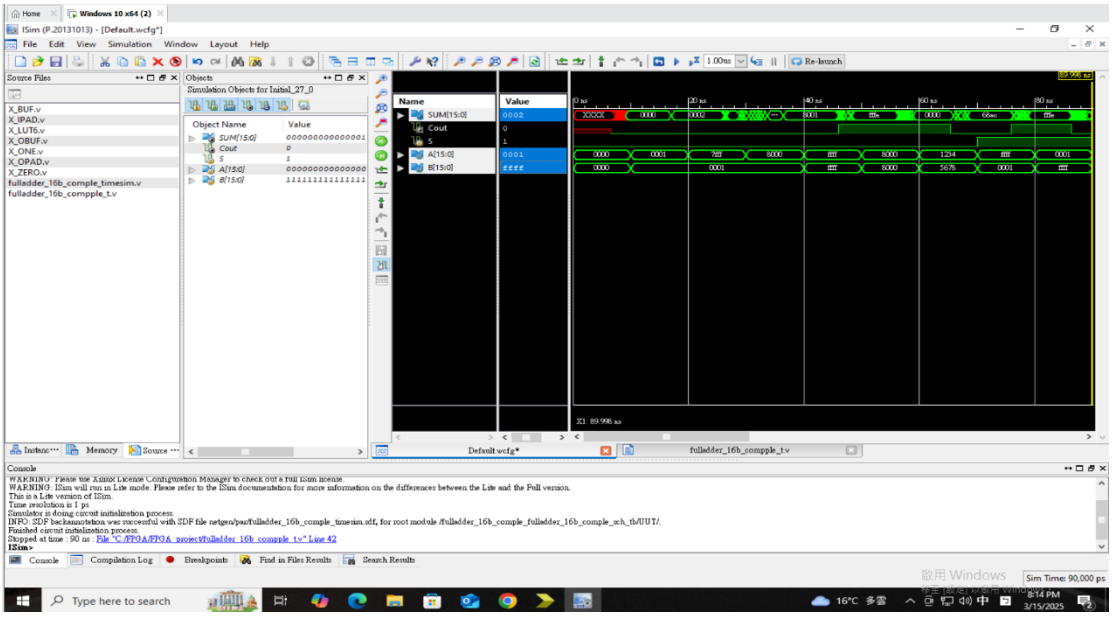
```
`timescale 1ns / 1ps

module fulladder_16b_comple_fulladder_16b_comple_sch_tb();

// Inputs
    reg S;
    reg [15:0] A;
    reg [15:0] B;

// Output
    wire [15:0] SUM;
    wire Cout;

// Bidirs

// Instantiate the UUT
    fulladder_16b_comple UUT (
        .SUM(SUM),
        .S(S),
```

```
            .Cout(Cout),
            .A(A),
            .B(B)
    );
// Initialize Inputs
   initial begin
            // 初始化
            A = 16'h0000;
            B = 16'h0000;
            S = 0;

            #10 A = 16'h0001; B = 16'h0001; S = 0;   // 測試 1 + 1
            #10 A = 16'h7FFF; B = 16'h0001; S = 0;   // 測試最大正數 + 1
            #10 A = 16'h8000; B = 16'h0001; S = 0;   // 測試最小負數 + 1
            #10 A = 16'hFFFF; B = 16'hFFFF; S = 0;   // 測試 -1 + (-1)
            #10 A = 16'h8000; B = 16'h8000; S = 0;   // 測試溢出情況 (-32768 + -32768)
            #10 A = 16'h1234; B = 16'h5678; S = 0;   // 測試一般加法
            #10 A = 16'hFFFF; B = 16'h0001; S = 1;   // 測試 -1 + 1 + 進位
            #10 A = 16'h0001; B = 16'hFFFF; S = 1;   // 測試 1 + (-1) + 進位

            #10 $finish;   // 測試結束
      end
endmodule
```

Behavior:

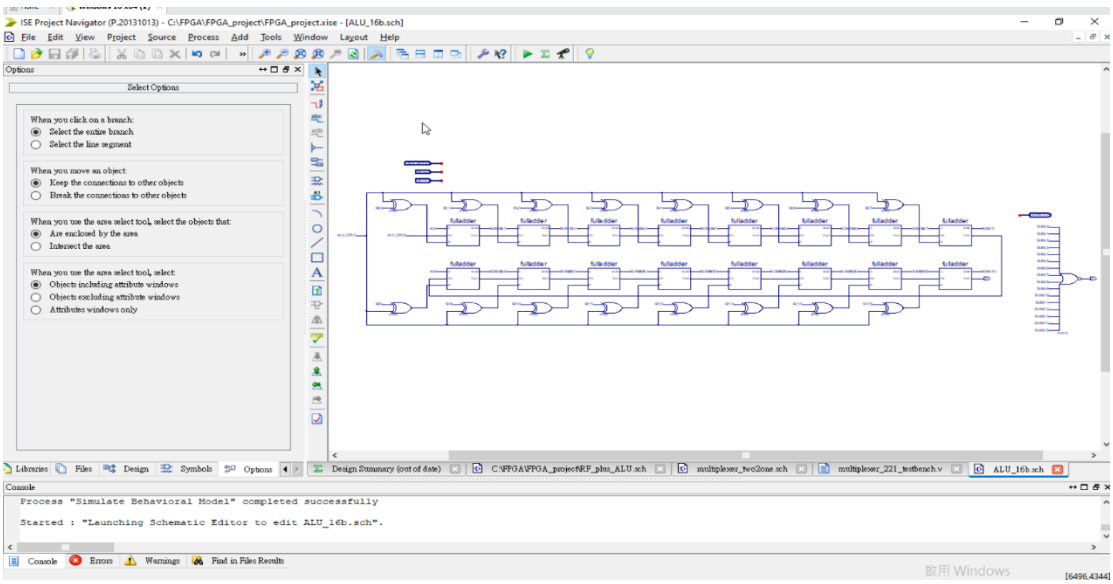


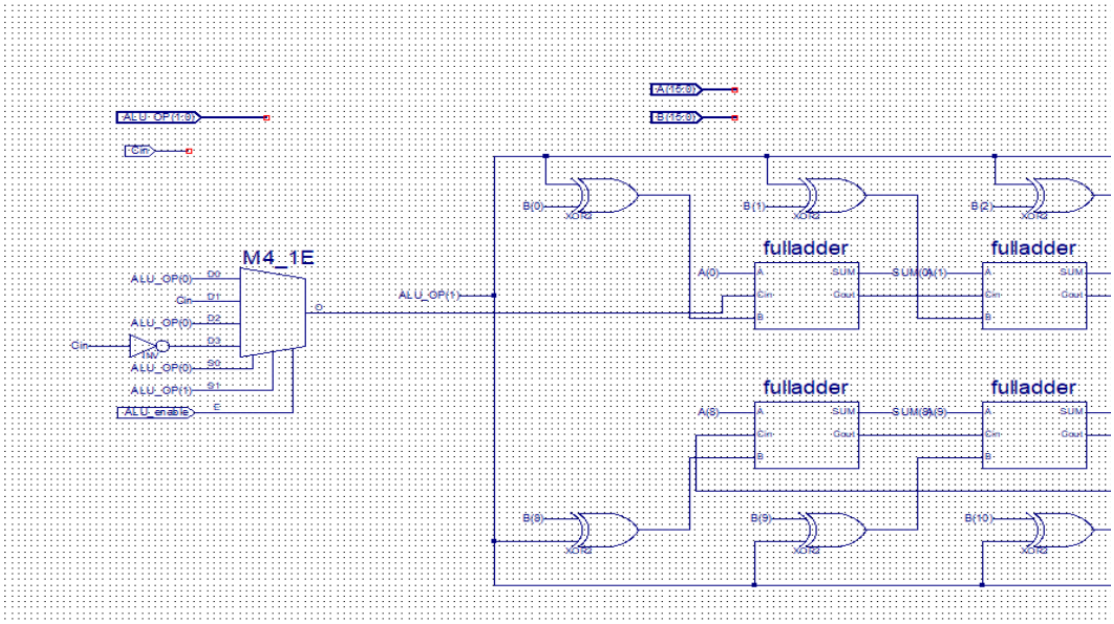
Post-Route:



9. 16 bit ALU

Schematic:





Testbench:

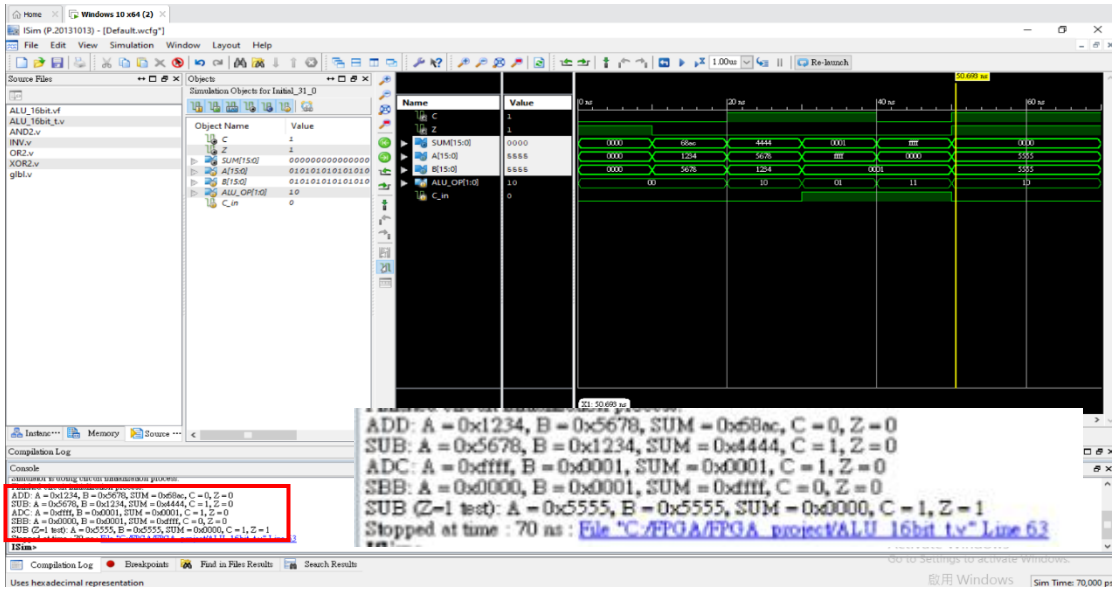
// Verilog test fixture created from schematic C:\FPGA\FPGA_project\ALU_16b.sch - Mon Mar 17 13:35:55 2025

```
`timescale 1ns / 1ps

module ALU_16b_ALU_16b_sch_tb();

// Inputs
    reg [15:0] A;
    reg [15:0] B;
    reg [1:0] ALU_OP;

// Output
    wire C;
    wire Z;
    wire [15:0] SUM;

// Bidirs

// Instantiate the UUT
    ALU_16b UUT (
        .C(C),
        .A(A),
        .B(B),
```

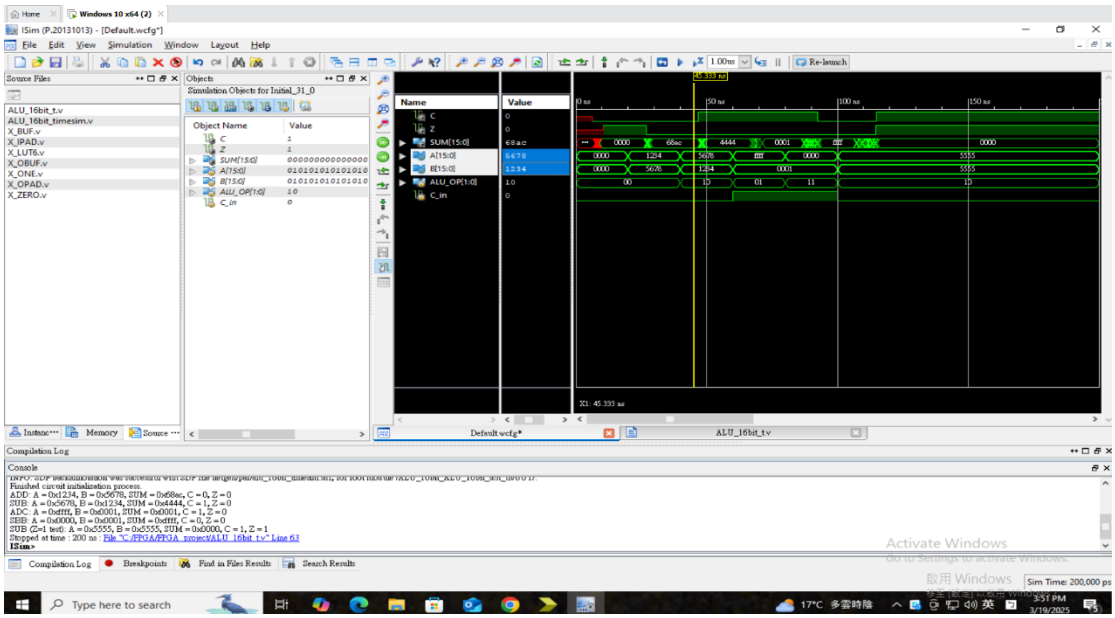
```
            .ALU_OP(ALU_OP),
            .Z(Z),
            .SUM(SUM)
    );
// Initialize Inputs

        initial begin
            // 測試加法
            A = 16'h0003; B = 16'h0005; ALU_OP = 2'b00;
            #10;
            $display("ADD: %h + %h = %h, Carry = %b, Zero = %b", A, B, SUM, C, Z);
                    // 測試加法
            A = 16'h0003; B = 16'h0005; ALU_OP = 2'b01;
            #10;
                $display("ADD: %h + %h = %h, Carry = %b, Zero = %b", A, B, SUM, C, Z);
            // 測試加法 (進位)
            A = 16'hFFFF; B = 16'h0001; ALU_OP = 2'b00;
            #10;
            $display("ADD: %h + %h = %h, Carry = %b, Zero = %b", A, B, SUM, C, Z);
                    // 測試加法 (進位)
            A = 16'hFFFF; B = 16'h0001; ALU_OP = 2'b01;
            #10;
            $display("ADD: %h + %h = %h, Carry = %b, Zero = %b", A, B, SUM, C, Z);
                // 測試減法
            A = 16'h0005; B = 16'h0003; ALU_OP = 2'b10;
            #10;
            $display("SUB: %h - %h = %h, Carry = %b, Zero = %b", A, B, SUM, C, Z);

            // 測試減法
            A = 16'h0005; B = 16'h0003; ALU_OP = 2'b11;
            #10;
            $display("SUB: %h - %h = %h, Carry = %b, Zero = %b", A, B, SUM, C, Z);

            // 測試減法 (結果為零)
            A = 16'h0004; B = 16'h0004; ALU_OP = 2'b11;
            #10;
            $display("SUB: %h - %h = %h, Carry = %b, Zero = %b", A, B, SUM, C, Z);
```

```
        // 測試減法 (借位)
        A = 16'h0003; B = 16'h0005; ALU_OP = 2'b11;
        #10;
        $display("SUB: %h - %h = %h, Carry = %b, Zero = %b", A, B, SUM, C, Z);

        // 測試完成
        $finish;
    end
endmodule
```

Behavior:

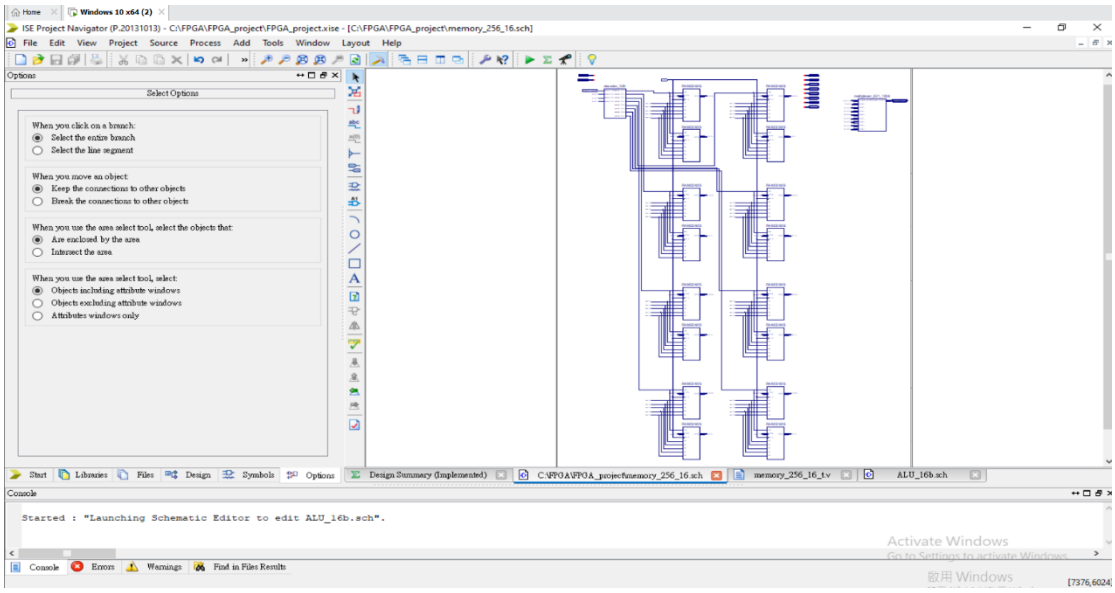


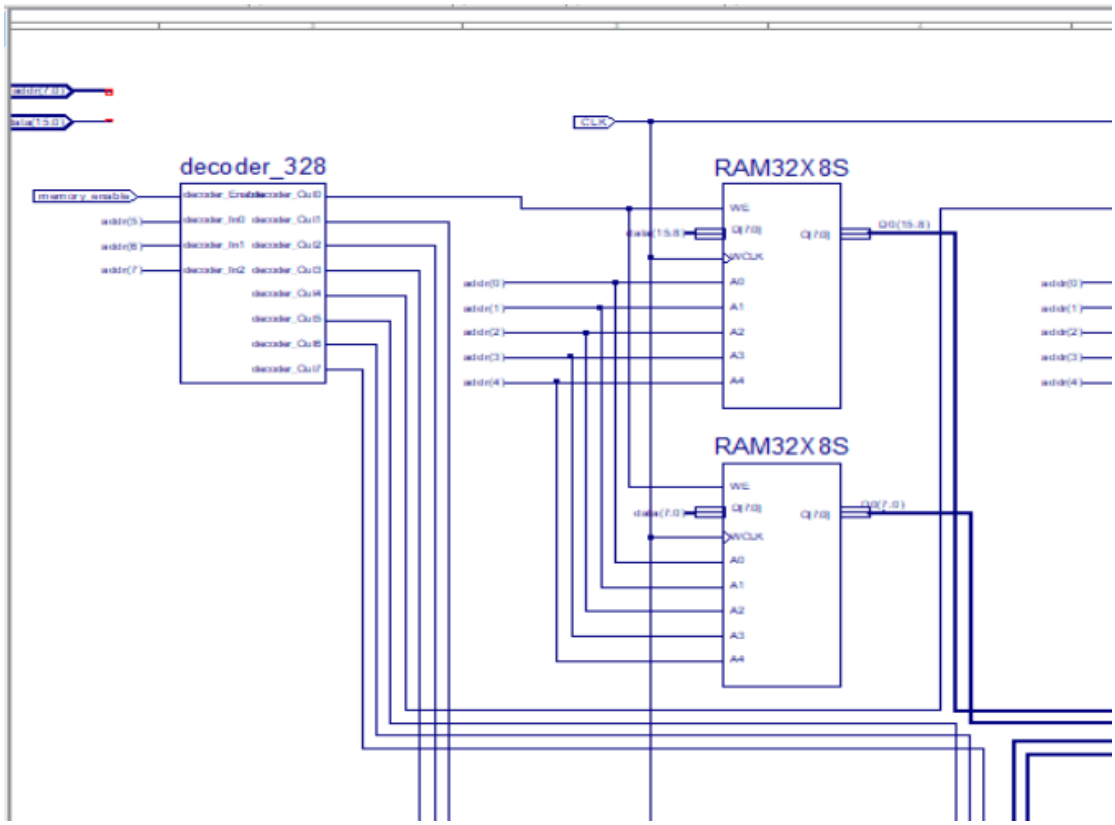
Post-Route:

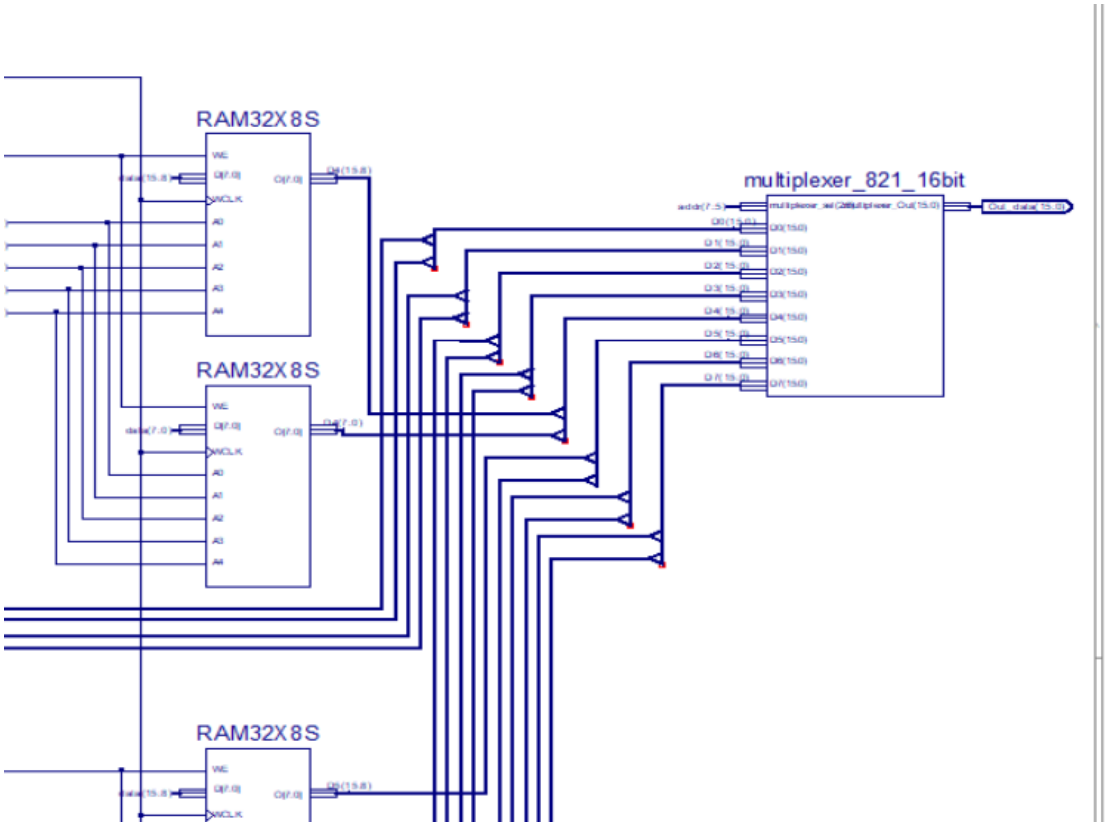


10. 256x16 memory

Schematic:







Testbench:

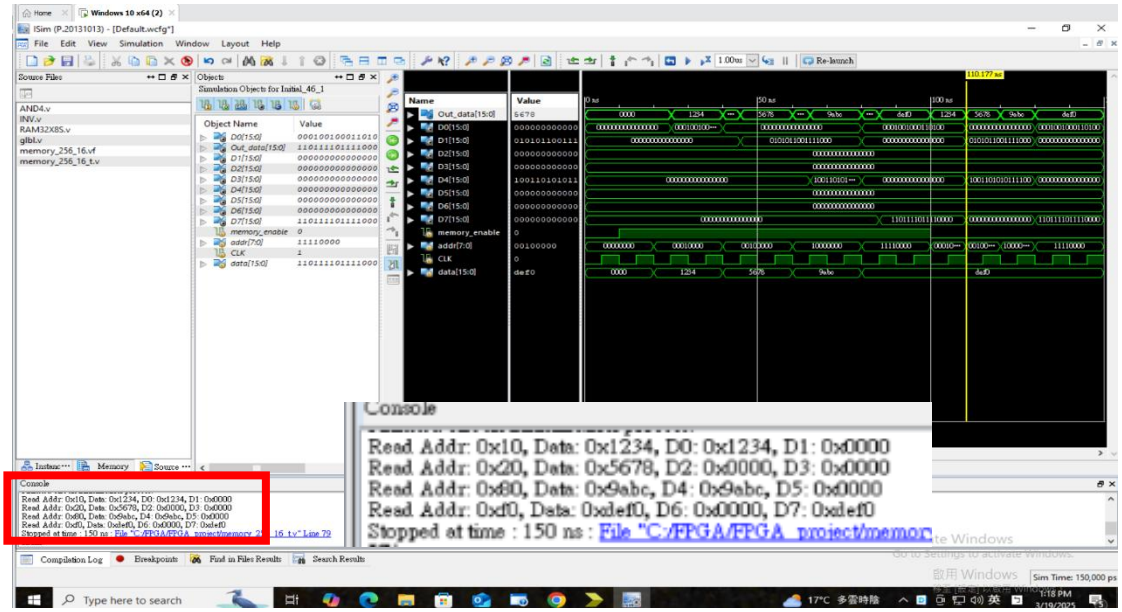
// Verilog test fixture created from schematic C:\FPGA\FPGA_project\memory_256_16.sch - Fri Mar 21 19:43:04 2025

```
`timescale 1ns / 1ps

module memory_256_16_memory_256_16_sch_tb();

// Inputs
    reg memory_enable;
    reg [7:0] addr;
    reg CLK;
    reg [15:0] data;

// Output
    wire [15:0] Out_data;

// Bidirs

// Instantiate the UUT
```

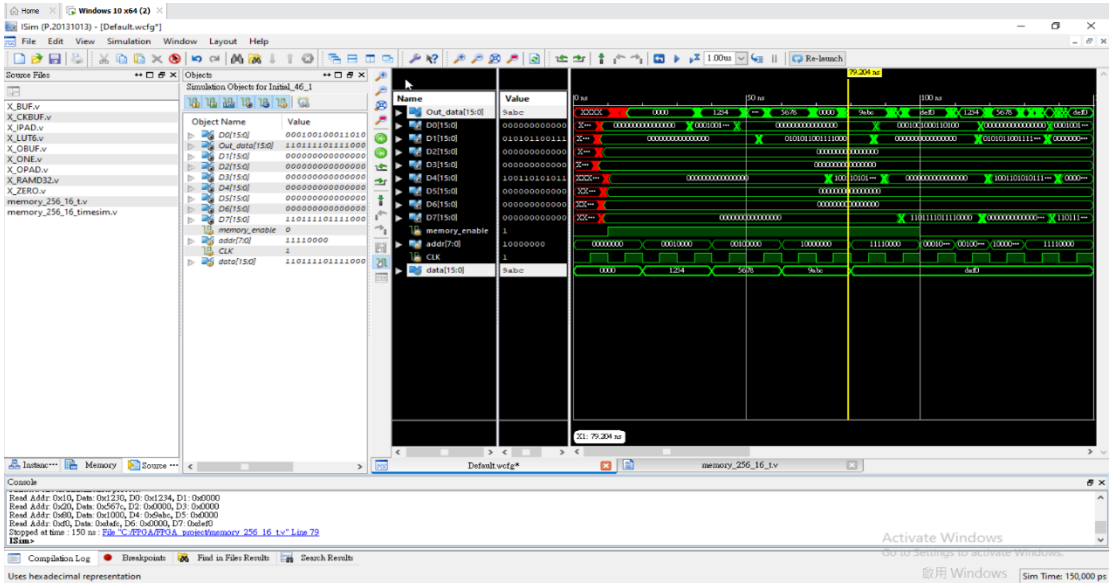
```
    memory_256_16 UUT (
            .memory_enable(memory_enable),
            .addr(addr),
            .CLK(CLK),
            .data(data),
            .Out_data(Out_data)
    );
// Initialize Inputs
always #5 CLK = ~CLK;

// **初始化測試**
initial begin
    // 設定初始值
    CLK = 0;
    memory_enable = 0;
    addr = 8'h00;
    data = 16'h0000;

    // **Step 1: 使能記憶體**
    #10;
    memory_enable = 1; // 啟用寫入

    // **Step 2: 測試寫入**
    #10; addr = 8'h10; data = 16'h1234; #10; // 寫入 0x1234 到地址 0x10
    #10; addr = 8'h20; data = 16'h5678; #10; // 寫入 0x5678 到地址 0x20
    #10; addr = 8'h80; data = 16'h9ABC; #10; // 寫入 0x9ABC 到地址 0x80
    #10; addr = 8'hF0; data = 16'hDEF0; #10; // 寫入 0xDEF0 到地址 0xF0

    // **Step 3: 測試讀取**
    #10; memory_enable = 0; // 關閉寫入，進入讀取模式
    addr = 8'h10; #10;
    $display("Read Addr: 0x%02X, Data: 0x%04X", addr, Out_data);

    addr = 8'h20; #10;
    $display("Read Addr: 0x%02X, Data: 0x%04X", addr, Out_data);

    addr = 8'h80; #10;
    $display("Read Addr: 0x%02X, Data: 0x%04X", addr, Out_data);
```

```
        addr = 8'hF0; #10;
        $display("Read Addr: 0x%02X, Data: 0x%04X", addr, Out_data);

        // **Step 4:  測試結束**
        #10;
        $finish;
end
endmodule
```

Behavior:

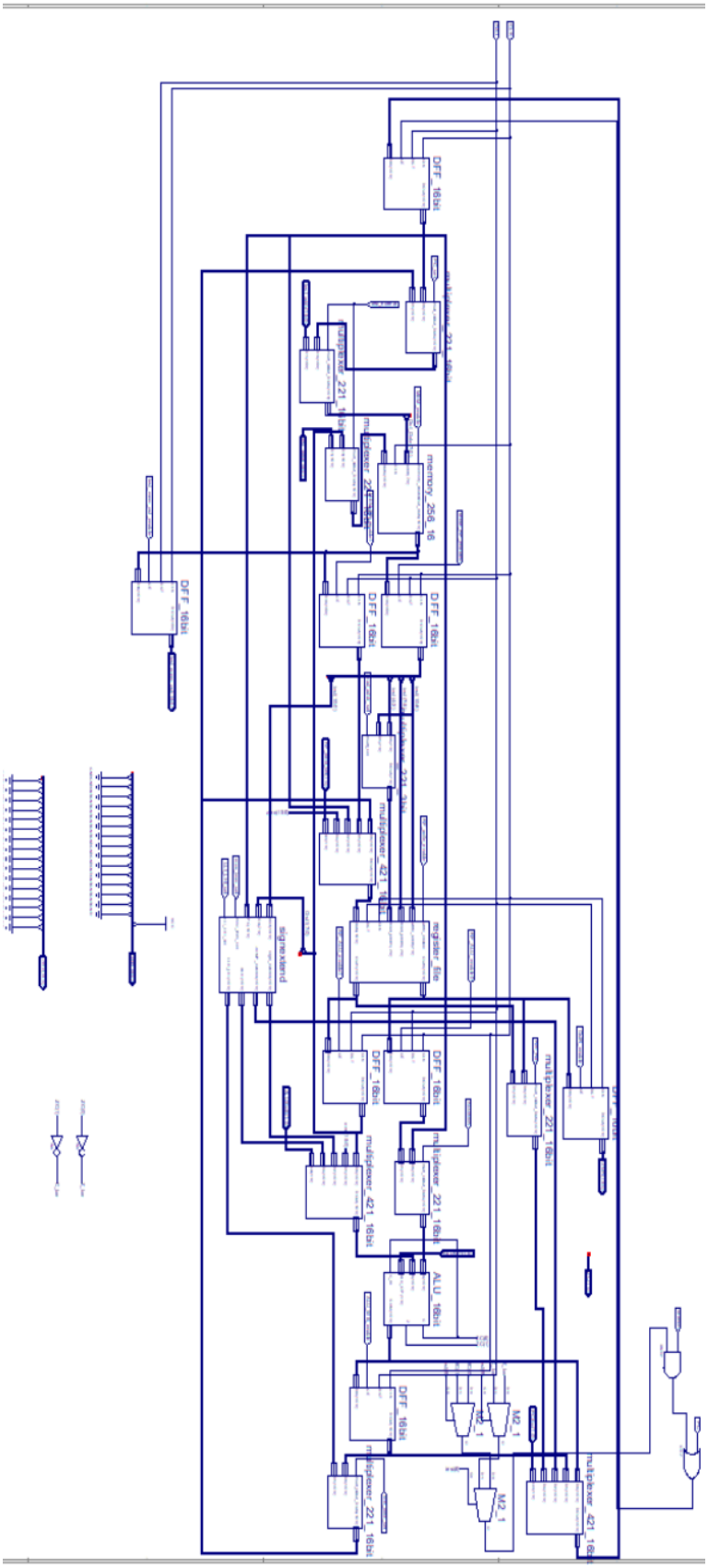


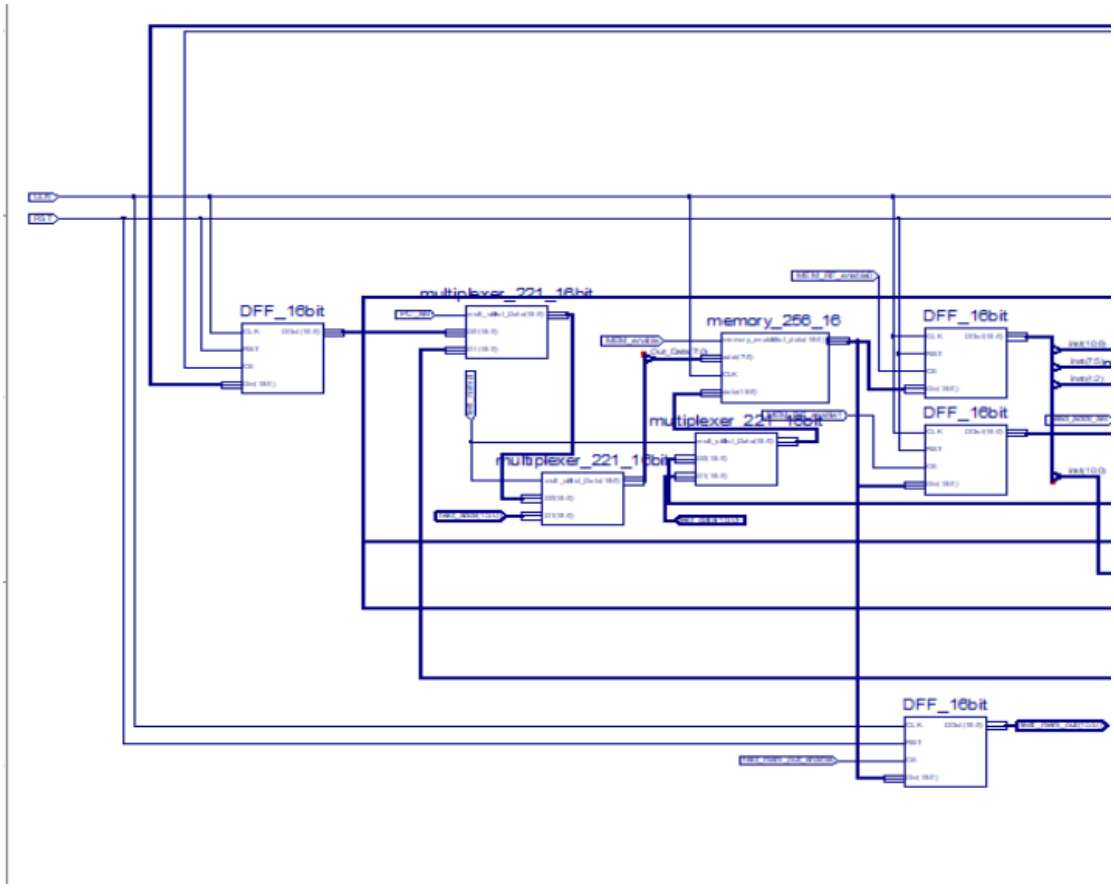
Post-Route:

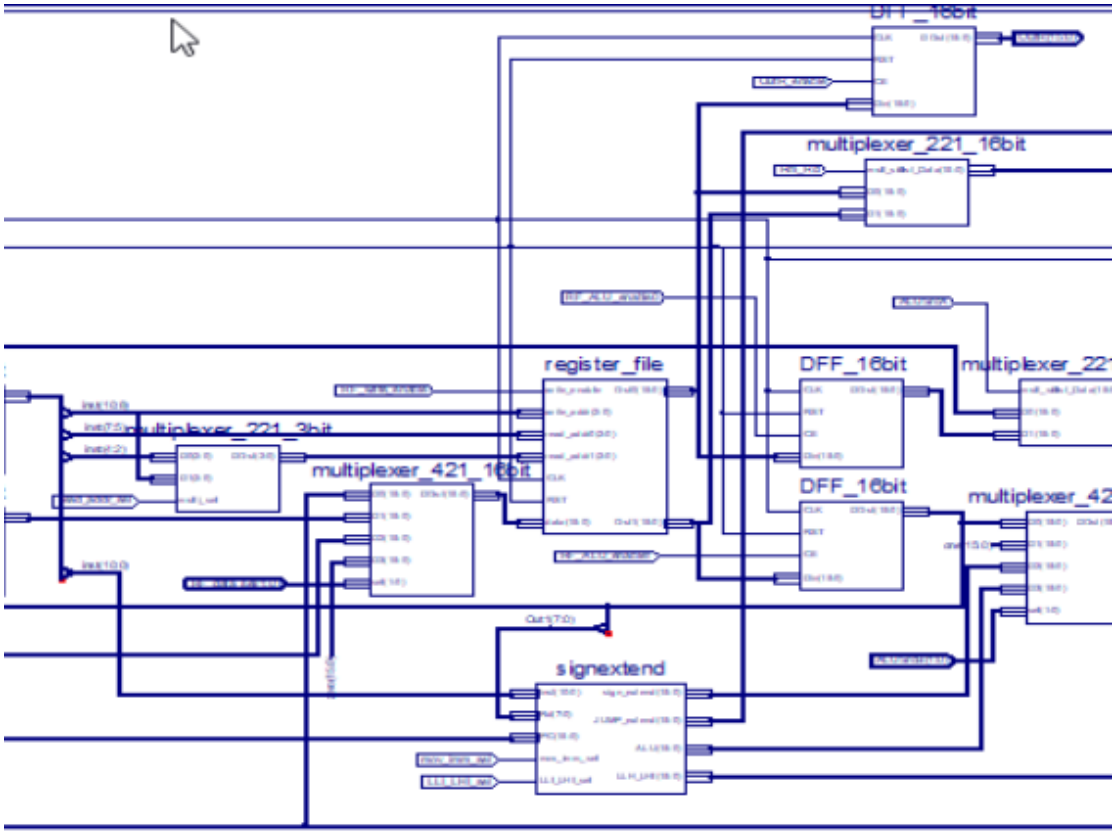


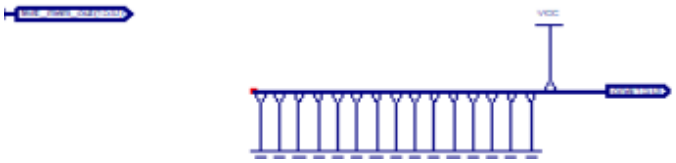
11. Complete Data path

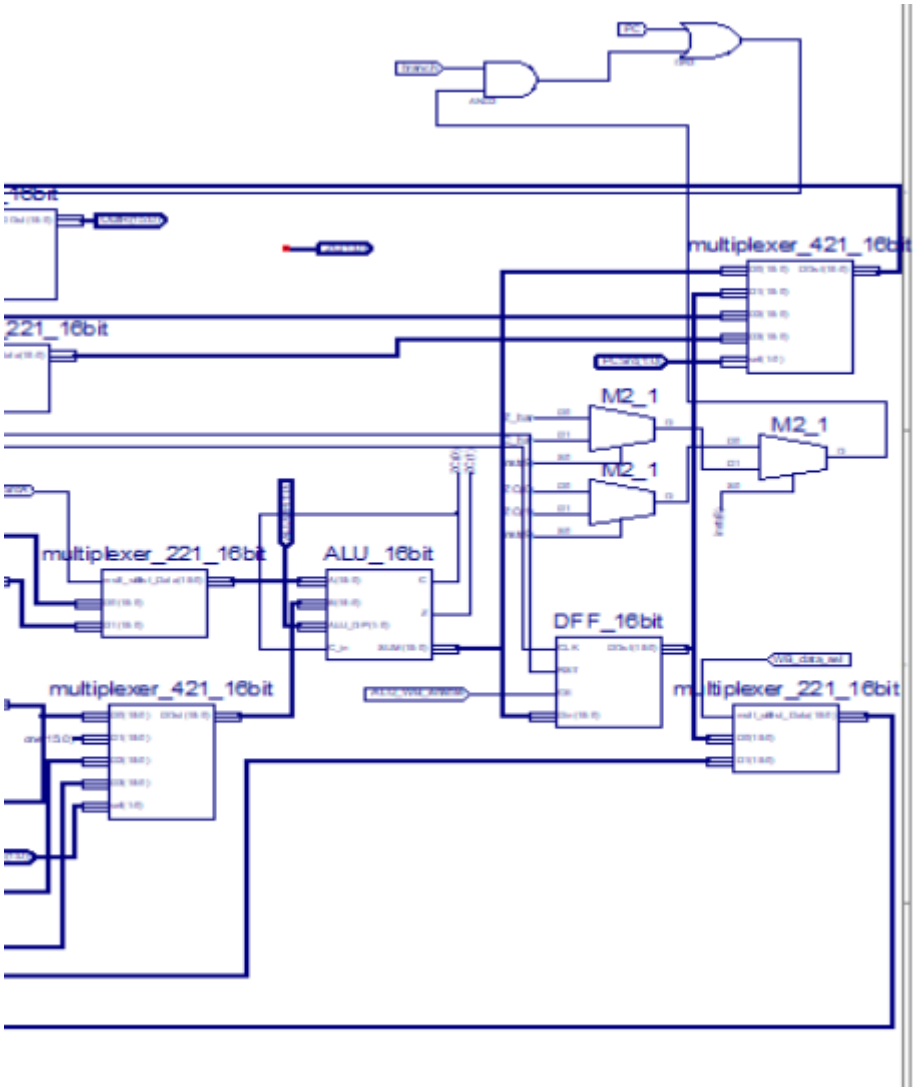
Schematic:




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



Testbench:

// Verilog test fixture created from schematic C:\FPGA\FPGA_project\complete.sch - Wed Mar 26 19:23:56 2025

```
`timescale 1ns / 1ps

module complete_complete_sch_tb();

// Inputs
        parameter clk_period = 20;
        parameter delay_factor = 2;
    reg PC;
    reg PC_sel;
    reg MEM_enable;
    reg MEM_RF_enable0;
```

```
    reg MEM_RF_enable1;
    reg read_addr_sel;
    reg [1:0] RF_data_sel;
    reg RF_write_enable;
    reg mov_imm_sel;
    reg LLI_LHI_sel;
    reg RF_ALU_enable0;
    reg RF_ALU_enable1;
    reg ALUSrcA;
    reg [1:0] ALUSrcB;
    reg ALU_WB_enable;
    reg WB_data_sel;
    reg [1:0] PCSrc;
    reg Rm_Rd;
    reg OutR_enable;
    reg CLK;
    reg RST;
    reg [15:0] test_addr;
    reg [15:0] test_data;
    reg test_mem_out_enable;
    reg [1:0] ALUOP;
    reg test_normal;

// Output
    wire [15:0] one;
    wire [1:0] ZC;
    wire [15:0] OutR;
    wire [15:0] zero;
    wire [15:0] test_mem_out;
        wire done;
        integer i;
// Bidirs

// Instantiate the UUT
    complete UUT (
            .one(one),
            .ZC(ZC),
            .PC(PC),
```

```
                .PC_sel(PC_sel),
                .MEM_enable(MEM_enable),
                .MEM_RF_enable0(MEM_RF_enable0),
                .MEM_RF_enable1(MEM_RF_enable1),
                .read_addr_sel(read_addr_sel),
                .RF_data_sel(RF_data_sel),
                .RF_write_enable(RF_write_enable),
                .mov_imm_sel(mov_imm_sel),
                .LLI_LHI_sel(LLI_LHI_sel),
                .RF_ALU_enable0(RF_ALU_enable0),
                .RF_ALU_enable1(RF_ALU_enable1),
                .ALUSrcA(ALUSrcA),
                .ALUSrcB(ALUSrcB),
                .ALU_WB_enable(ALU_WB_enable),
                .WB_data_sel(WB_data_sel),
                .PCSrc(PCSrc),
                .Rm_Rd(Rm_Rd),
                .OutR(OutR),
                .OutR_enable(OutR_enable),
                .zero(zero),
                .CLK(CLK),
                .RST(RST),
                .test_addr(test_addr),
                .test_data(test_data),
                .test_mem_out(test_mem_out),
                .test_mem_out_enable(test_mem_out_enable),
                .ALUOP(ALUOP),
                .test_normal(test_normal)
    );
// Initialize Inputs
      always begin
        #(clk_period/2)CLK <= 1'b0;
        #(clk_period/2)CLK <= 1'b1;
        end
        initial begin
                PC = 0;
                MEM_enable = 0;
                MEM_RF_enable0 = 0;
```

```
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;


RST <= 1'b1; test_normal = 1'b0;
repeat (2) @(posedge CLK)
                    #(clk_period/delay_factor) RST <= 1'b1;
RST <= 1'b0; test_normal = 1'b1;

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'b1110_0000_1000_0000); // OUT R4 (36H)
//write_mem(16'h10,16'b1110_0000_0110_0000); // OUT R3 (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] pc = pc + 3 (to pc:16)
//write_mem(16'h16,16'b1110_0000_0100_0000); // OUT R2 (89H)

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)
@(posedge CLK)#(clk_period/delay_factor) MEM_enable = 1'b0;
  test_normal = 1'b1;
for(i = 0 ; i < 25 ; i = i + 1)
        @(posedge CLK) #(clk_period/delay_factor) test_addr = i;
```

```
  test_normal = 1'b0;
RST = 1'b0;
repeat (2) @(posedge CLK)
                                #(clk_period/delay_factor) RST = 1'b1;
RST = 1'b0;
test_mem_out_enable = 1;

//////////LLI R0 25//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;


PC_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
ALUOP = 2'b00;
@(posedge CLK)#10

PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable = 0;
RF_ALU_enable0 = 1;
RF_ALU_enable1 = 1;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
ALUSrcB = 2'b00;
```

```
@(posedge CLK)#10
LLI_LHI_sel = 0;
WB_data_sel = 1;
RF_data_sel = 2'b00;
RF_write_enable=1;

//////////LHI R0 63//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;

PC_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
read_addr_sel = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_ALU_enable0=1;
RF_ALU_enable1=1;
ALUSrcB = 2'b00;
@(posedge CLK)#10
RF_data_sel = 2'b00;
LLI_LHI_sel = 1;
WB_data_sel = 1;
RF_write_enable=1;
//////////OUT R0//////////
@(posedge CLK)#10
```

```
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
LLI_LHI_sel = 1;
PC_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
ALUOP = 2'b00;
RF_write_enable=0;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
ALUSrcB = 2'b00;
@(posedge CLK)#10
OutR_enable = 1;
//////////LDR R1 R0 #0//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;
```

```
RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=1;
RF_ALU_enable1=1;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
```

```
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=1;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 1;
LLI_LHI_sel = 0;
ALUSrcA = 1;
ALUSrcB = 2'b11;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 1;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
```

```
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=1;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b01;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
//////////LDR R2 R0 #1//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
```

```
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=1;
RF_ALU_enable1=1;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
```

```
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=1;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 1;
LLI_LHI_sel = 0;
ALUSrcA = 1;
ALUSrcB = 2'b11;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 1;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
```

```
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=1;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b01;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
//////////OUT R1 47//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
PC_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
ALUOP = 2'b00;
```

```
RF_write_enable=0;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
ALUSrcB = 2'b00;
@(posedge CLK)#10
OutR_enable = 1;
//////////OUT R2 89//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
PC_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
ALUOP = 2'b00;
RF_write_enable=0;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
ALUSrcB = 2'b00;
```

```
@(posedge CLK)#10
OutR_enable = 1;
//////////ADD R3 R1 R2//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=1;
RF_ALU_enable1=1;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;
```

```
RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=1;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 1;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
```

```
MEM_RF_enable1 = 0;
RF_write_enable=1;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 1;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
//////////OUT R3 D0//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
PC_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
ALUOP = 2'b00;
RF_write_enable=0;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
```

```
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
ALUSrcB = 2'b00;
@(posedge CLK)#10
OutR_enable = 1;
//////////SUB R3 R1 R2//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=1;
RF_ALU_enable1=1;
```

```
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=1;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 1;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b10;
```

```
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=1;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 1;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
//////////OUT R3 D0//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
PC_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
ALUOP = 2'b00;
RF_write_enable=0;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
```

```
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
ALUSrcB = 2'b00;
@(posedge CLK)#10
OutR_enable = 1;



//////////LDR R4 R1 R2//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
```

```
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=1;
RF_ALU_enable1=1;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=1;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
```

```
mov_imm_sel = 1;
LLI_LHI_sel = 0;
ALUSrcA = 1;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 1;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=1;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
```

```
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b01;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
//////////OUT R4 39H//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
PC_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
ALUOP = 2'b00;
RF_write_enable=0;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
ALUSrcB = 2'b00;
```

```
@(posedge CLK)#10
OutR_enable = 1;
//////////STR R4 R0 3//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=1;
RF_ALU_enable1=1;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;
```

```
RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=1;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 1;
LLI_LHI_sel = 0;
ALUSrcA = 1;
ALUSrcB = 2'b11;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 1;
MEM_RF_enable0 = 0;
```

```
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 1;
PC_sel = 1;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
//////////STR R3 R1 R2//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
```

```
ALUSrcA = 0;
ALUSrcB = 2'b01;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=1;
RF_ALU_enable1=1;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=1;
Rm_Rd=0;
```

```
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 1;
LLI_LHI_sel = 0;
ALUSrcA = 1;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 1;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 1;
PC_sel = 1;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
//////////OUT R1 47H//////////
/*@(posedge CLK)
```

```
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
PC_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
ALUOP = 2'b00;
@(posedge CLK)
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
ALUSrcB = 2'b00;
@(posedge CLK)
OutR_enable = 1;*/
//////////OUT R2 89H//////////
/*@(posedge CLK)
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
PC_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
ALUOP = 2'b00;
@(posedge CLK)
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
```

```
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
ALUSrcB = 2'b00;
@(posedge CLK)
OutR_enable = 1;*/
//////////CMP R1 R2//////////
@(posedge CLK)
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
```

```
RF_ALU_enable0=1;
RF_ALU_enable1=1;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 1;
ALUSrcB = 2'b00;
WB_data_sel = 0;
```

```
PCSrc = 2'b00;
ALUOP = 2'b10;
//////////ADC R3 R1 R2//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=1;
RF_ALU_enable1=1;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;
```

```
RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=1;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 1;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b01;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
```

```
MEM_RF_enable1 = 0;
RF_write_enable=1;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 1;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
//////////OUT R3 (D0H)//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
PC_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
RF_write_enable=0;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
```

```
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
ALUSrcB = 2'b00;
@(posedge CLK)#10
OutR_enable = 1;
//////////ADDI R2,R1 #3//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=1;
RF_ALU_enable1=1;
```

```
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=1;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 1;
LLI_LHI_sel = 0;
ALUSrcA = 1;
ALUSrcB = 2'b11;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
```

```
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=1;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
//////////OUT R2 (4AH) //////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
PC_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
ALUOP = 2'b00;
RF_write_enable=0;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
```

```
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
ALUSrcB = 2'b00;
@(posedge CLK)#10
OutR_enable = 1;
//////////SUBI R2,R1 #9//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
```

```
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=1;
RF_ALU_enable1=1;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=1;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 1;
LLI_LHI_sel = 0;
ALUSrcA = 1;
```

```
ALUSrcB = 2'b11;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b10;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=1;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
//////////OUT R2 (80H)//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
PC_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
ALUOP = 2'b00;
RF_write_enable=0;
```

```
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
ALUSrcB = 2'b00;
@(posedge CLK)#10
OutR_enable = 1;
//////////MOV R2 R1//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
```

```
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=1;
RF_ALU_enable1=1;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=1;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
```

```
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 1;
ALUSrcB = 2'b11;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=1;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
//////////OUT R2 (80H)//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
PC_sel = 0;
```

```
ALUSrcA = 0;
ALUSrcB = 2'b01;
ALUOP = 2'b00;
RF_write_enable=0;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
ALUSrcB = 2'b00;
@(posedge CLK)#10
OutR_enable = 1;
//////////JMP pc = 16'h35//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
```

```
ALUSrcB = 2'b01;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
```

```
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b10;
ALUOP = 2'b00;
//////////OUT R1 (8CH)//////////

@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
PC_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
ALUOP = 2'b00;
RF_write_enable=0;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
ALUSrcB = 2'b00;
@(posedge CLK)#10
OutR_enable = 1;
```

```
//////////B[AL] pc = pc + 3 (to pc:16)//////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b01;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 0;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=1;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
```

```
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b10;
WB_data_sel = 0;
PCSrc = 2'b00;
ALUOP = 2'b00;
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 0;
MEM_RF_enable1 = 0;
RF_write_enable=0;
RF_ALU_enable0=0;
RF_ALU_enable1=0;
ALU_WB_enable=0;
Rm_Rd=0;
OutR_enable=0;
test_mem_out_enable=1;

RF_data_sel = 2'b00;
read_addr_sel = 0;
PC_sel = 0;
mov_imm_sel = 0;
LLI_LHI_sel = 0;
ALUSrcA = 0;
ALUSrcB = 2'b00;
WB_data_sel = 0;
PCSrc = 2'b01;
ALUOP = 2'b00;
//////////OUT R2 (89H) //////////
@(posedge CLK)#10
PC = 1;
MEM_enable = 0;
MEM_RF_enable0 = 1;
MEM_RF_enable1 = 1;
```

```
		PC_sel = 0;
		ALUSrcA = 0;
		ALUSrcB = 2'b01;
		ALUOP = 2'b00;
		RF_write_enable=0;
		@(posedge CLK)#10
		PC = 0;
		MEM_enable = 0;
		MEM_RF_enable0 = 0;
		MEM_RF_enable1 = 0;
		RF_write_enable=0;
		RF_ALU_enable0=0;
		RF_ALU_enable1=0;
		ALU_WB_enable=0;
		Rm_Rd=0;
		OutR_enable=0;
		ALUSrcB = 2'b00;
		@(posedge CLK)#10
		OutR_enable = 1;
		//wait(done);
	end
		task write_mem;
		input [15:0] addr,data;
		begin
			@(posedge CLK)#(clk_period/delay_factor)begin
				test_normal = 1'b1;
				MEM_enable = 1;
				test_mem_out_enable = 1'b1;test_addr = addr;
				test_data = data;
			end
		end
		endtask
	initial #10000 $finish;
	initial
		$monitor($realtime ,"ns %h %h %h %h %h %h %h %h %h \n
",CLK,RST,test_mem_out_enable,test_normal,test_addr,test_data,test_mem_out,OutR,done);
endmodule
```

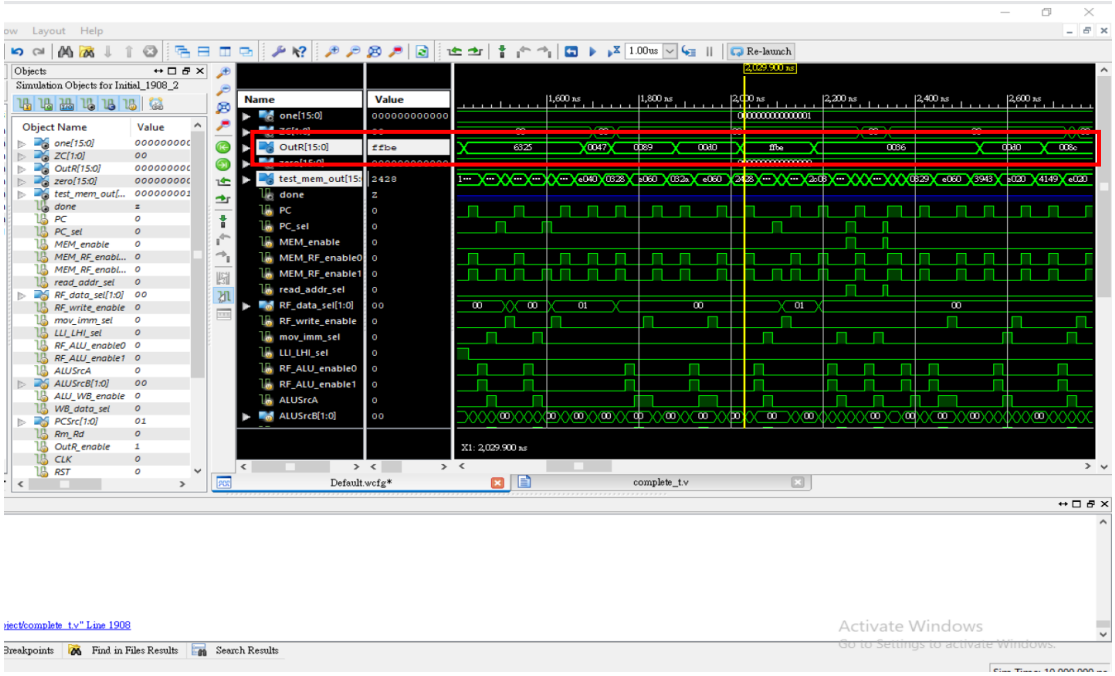
Simulation

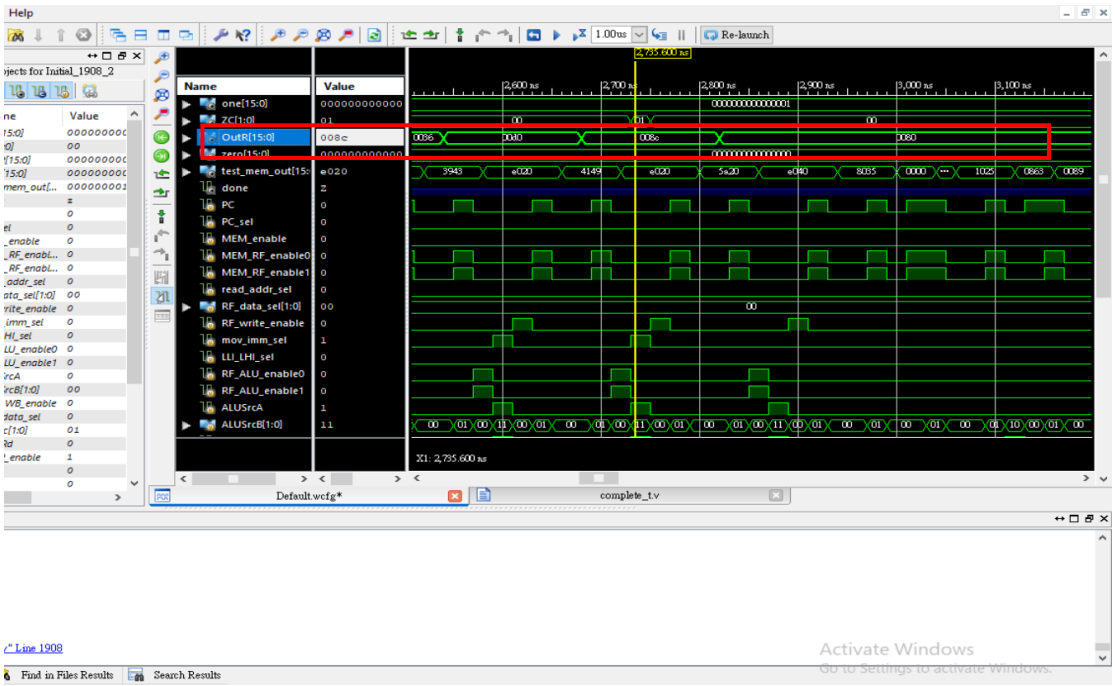
除了老師給的測試範例 我另外寫了其他的範例用 test 隔出來，也有將應該輸出的值標示。

```
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)
```

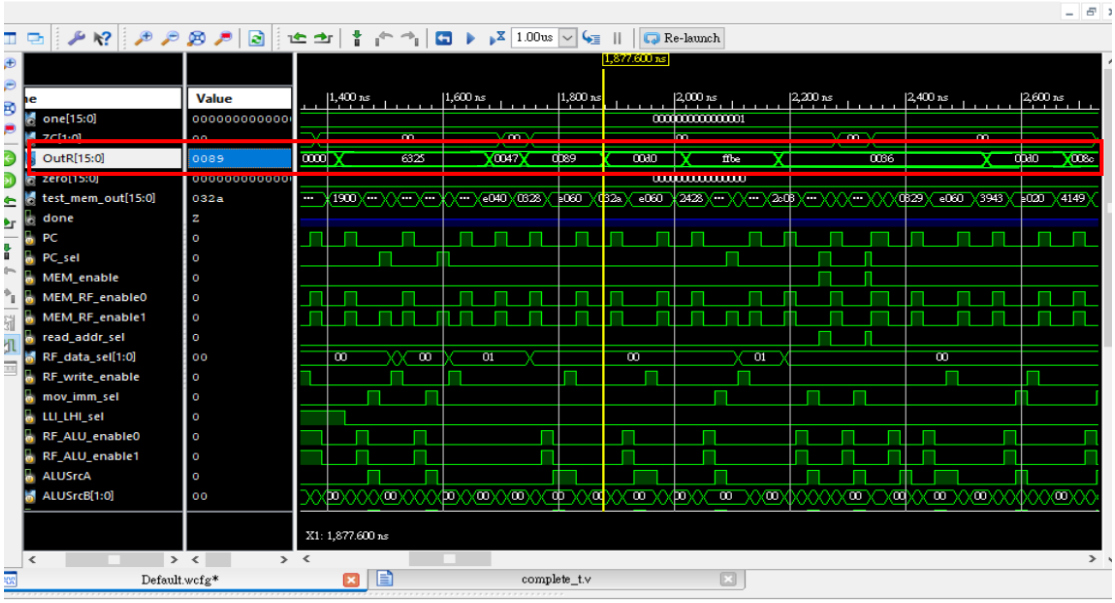
Behavior:

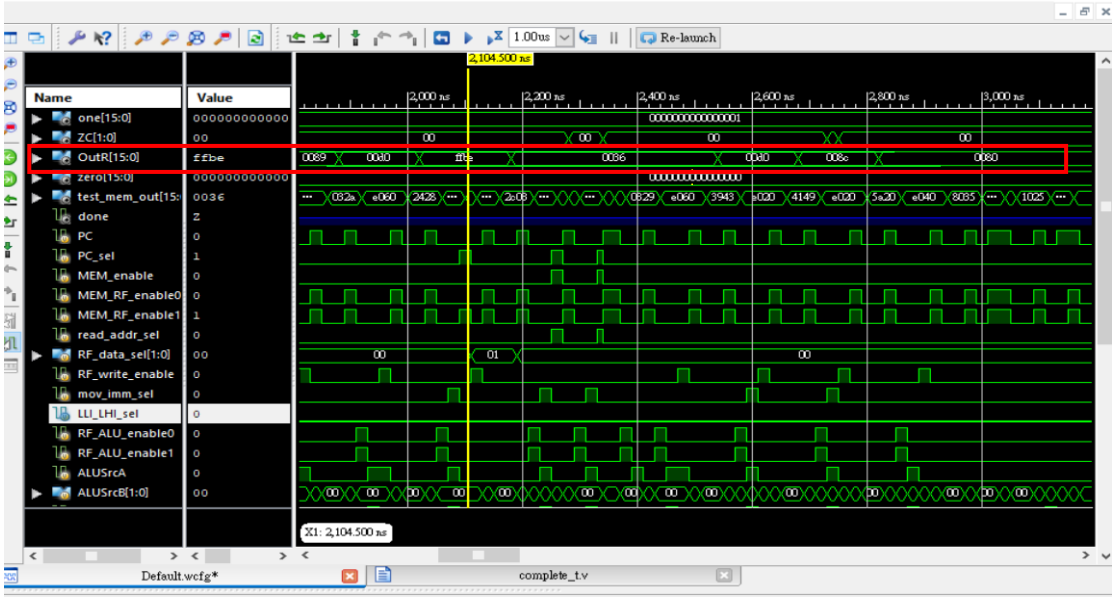
6325->0047->0089->00D0->FFBE->0036->00D0->008C->0080





Post-Route:


Re-launch
1.00us
1,877.600 ns
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


Re-launch
1.00us
2,104.500 ns
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: 2,104.500 ns
Default.wcfg*
complete_t.v

Hardware cost

Xilinx Design Summary Page 1 of 3

complete Project Status			
Project File:	FPGA_project.xise	**Parser Errors:**	No Errors
Module Name:	complete	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx75t-1ff484	• **Errors:**	No Errors
Product Version:	ISE 14.7	• **Warnings:**	49 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	251	93,120	1%	
Number used as Flip Flops	251			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	594	46,560	1%	
Number used as logic	528	46,560	1%	
Number using O6 output only	528			
Number using O5 output only	0			
Number using O5 and O6	0			
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	0			
Number using O5 output only	0			
Number using O5 and O6	64			
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	253	11,640	2%	
Number of LUT Flip Flop pairs used	660			
Number with an unused Flip Flop	413	660	62%	

file:///C:/FPGA/FPGA_project/complete_summary.html 4/7/2025

Number with an unused LUT	66	660	10%	
Number of fully used LUT-FF pairs	181	660	27%	
Number of unique control sets	24			
Number of slice register sites lost to control set restrictions	5	93,120	1%	
Number of bonded IOBs	119	240	49%	
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	3.66			

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

Timing Constraints:	All Constraints Met		

Detailed Reports					[-]
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Sat Apr 5 15:10:47 2025	0	44 Warnings (0 new)	0
Translation Report	Current	Sat Apr 5 15:10:54 2025	0	5 Warnings (0 new)	0
Map Report	Current	Sat Apr 5 15:11:23 2025	0	0	8 Infos (0 new)
Place and Route Report	Current	Sat Apr 5 15:11:40 2025	0	0	3 Infos (0 new)
Power Report					
Post-PAR Static Timing Report	Current	Sat Apr 5 15:11:50 2025	0	0	4 Infos (0 new)
Bitgen Report					

Secondary Reports			[-]
Report Name	**Status**	**Generated**	
ISIM Simulator Log	Current	Sat Apr 5 15:17:13 2025	
Post-Place and Route Simulation Model Report	Current	Sat Apr 5 15:12:03 2025	

Date Generated: 04/07/2025 - 10:57:58

Conclusion

這次的作業剛開始遇到的時候完全不知道從何下手，一步一步的慢慢查資料查閱教科書、pdf 講義以及聽老師的講解等等，才慢慢開始有一些方法做出來(花了大概一個多禮拜摸索)，雖然在大學的時候有用硬體描述語言做出同為 risc-v 的 CPU 架構，但是要自己將 schematic 畫出來完全是不一樣的狀況。這次的架構只能設計 multi-cycle 的架構讓難度增加非常多，因為要考慮每個指令在不同的週期上會有什麼樣的變化，信號該如何傳遞以及 data path 該如何跑，都非常重要，我非常理解教授要我們從 schematic 開始慢慢做出來就是要讓我們知道，平常寫硬體描述語言的時候應該在心中要有硬體實際的樣子，不管是簡單的加法器、MUX、decoder 等等，都要在心中用硬體的方式呈現出來，我認為這部分非常重要，我也了解到這些元件彼此之間共同的部分，是否可沿用，串起來再轉為一個新的 symbol 都十分有趣，也非常認同教授所說的學而不思則罔而不學則殆這句話，如果不實際動手做出來，就永遠得不到這之中的奧秘。

這次非常遺憾的沒有將 controller 做出來，我認為最大的問題在於對於硬體的架構不夠熟悉，以及許多基本功的部分還需要進一步雕磨，例如 data path、truth table 這種基本的呈現、PLA 的轉換等等，這次花太多時間在前面的基本電路上，以至於時間鄰近繳交期限才將 controller 的 truth table、FSM 做出來，我認為這部分是我需要再努力、加強的部分。

本次作業有幾個困擾我的地方，在這邊回顧一下。

1. 基本元件:這裡想說的是有些元件的模組化命名應該要統一清楚以及線的路徑該怎麼繞應該要事先想清楚才不會打結，還有像是 memory 該如何組出 256x16 需要先把 32x8 串接再組合就讓我想了一天左右，這都算是基本功不足的部分，但是親自嘗試過後其實並沒有想像中難，也讓我學到了很多。
2. Data path 的部分，雖然以前有學過一模一樣的東西，但是都不需要實際做出來，但是到了實際要我們一條線一條線畫的時候，就發現事情並沒有想的這麼簡單，每一個指令的流程該如何走，每條線該怎麼接，每個 enable 該如何處理都是一大難題，而且事前動作還要先將每個指令會參與 cycle 該要有甚麼信號，都讓第一次接觸的我算是一大挑戰，而且再測試的時候因為電路複雜，所以模擬的時候比較難直接看出來，花了很多時間再 debug。
3. 最後就是 controller 的部分，寫完所有信號之後，將 FSM 畫出來再把信號統整成 truth table 最後還要再做成電路，一開始完全沒有任何頭緒該如何處理，後來剩一些時間之後才好好想清楚整個流程，只要將狀態定義好，再來將狀態透過 AND/OR 轉換成對應的邏輯閘，基本上大部分就沒有問題了。

以上就是我在這次 project 學到的東西以及面對到的困難以及該如何解決，我覺得再面對比較複雜的電路上也比較有信心，學道非常多的東西，雖然第一次接觸腦袋一片空白，但是一步一步慢慢看就會發現眼前的電路活起來了，看錯誤也看得越來越快，希望未來在硬體的世界裡面可以更往上一層樓，也感謝教授開了這麼棒的一堂課，另外大學時的指導教授為嚴茂旭教授聽說也是師承林教授，非常幸運可以再兩位這麼棒的教授底下學習。