FPGA 系統設計實務

FPGA System Design

Mid-term Project

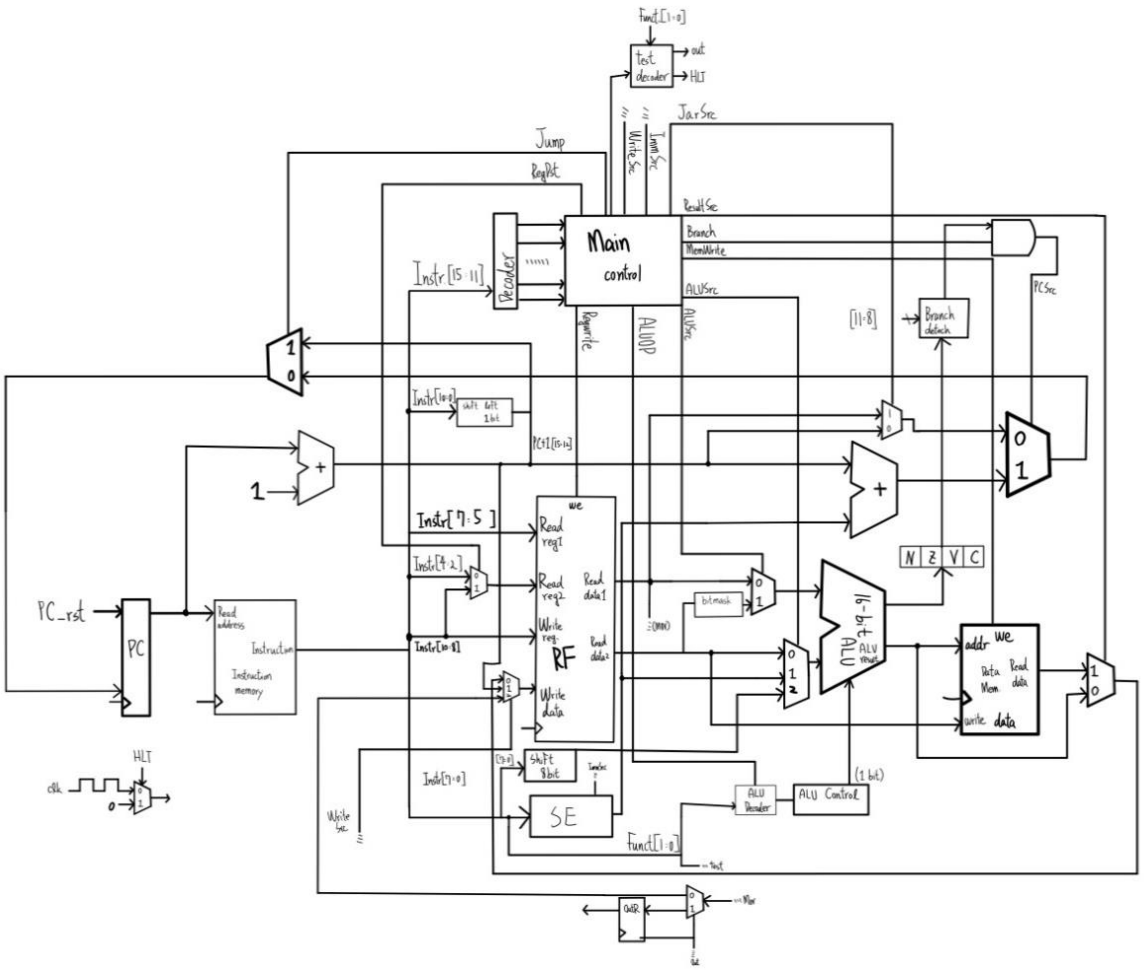
學號:M11215075

姓名:胡劭

1. Design:

我選擇的是 Single-cycle architecture，以下是我的設計圖，以及設計步驟明:

圖 1:完整設計圖



流程:

1.先從 FPGAHw(2K23f)文件中的 Table 1 來定義 instruction set

以下是我定義的 instruction format:

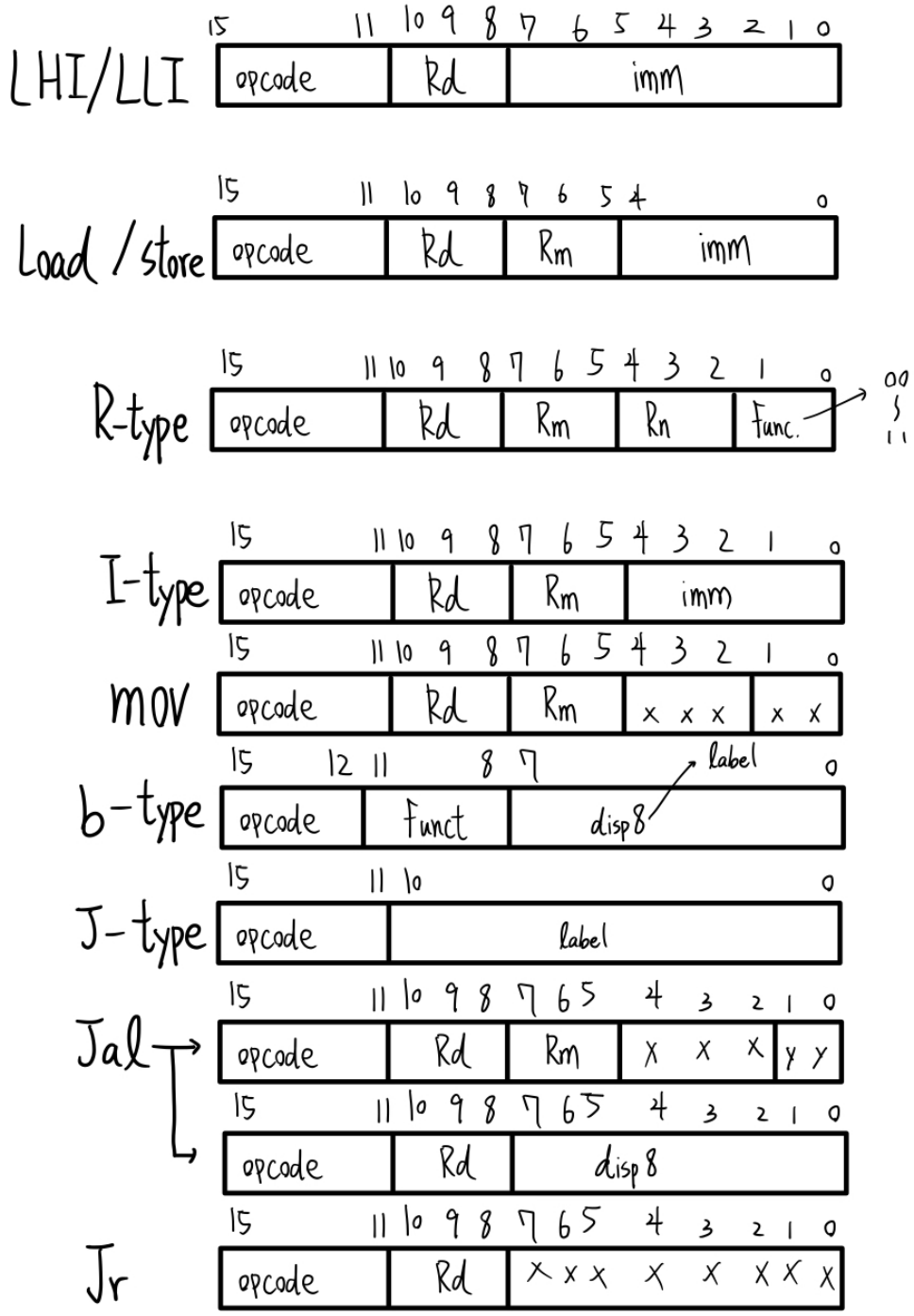


圖 2:指令格式

2.基本元件與各指令的製作:用 FPGAHw(2K23f)文件中給的 component 直接來做各個指令的串接:給的元件有 PC、Control signals generator、Memory、ALU、Register File

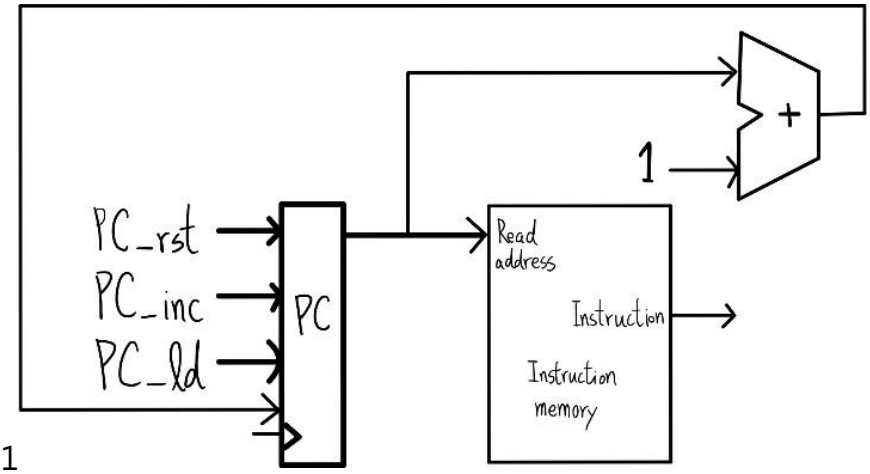


圖 3:PC 與+1

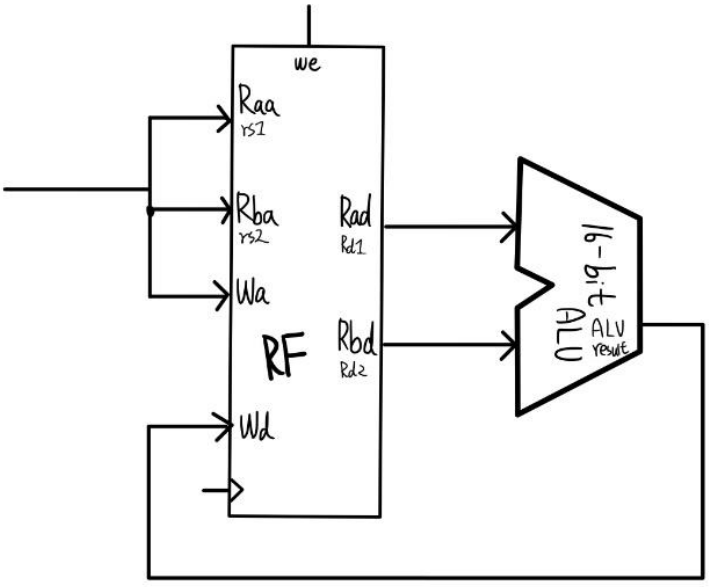


圖 4:R-type

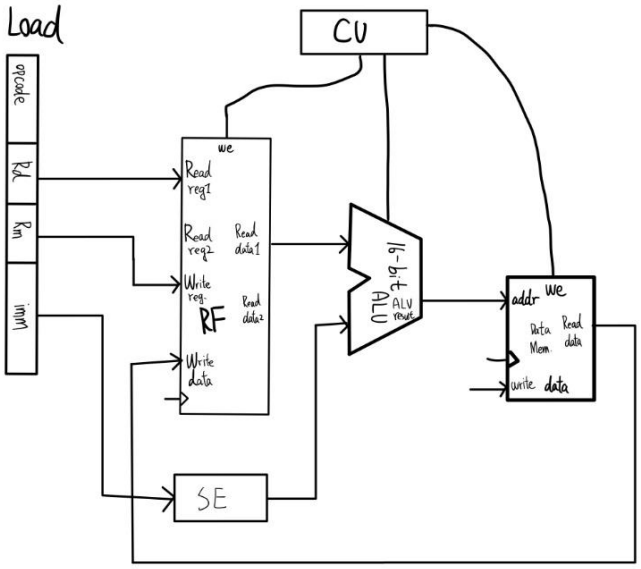


圖 5:Load

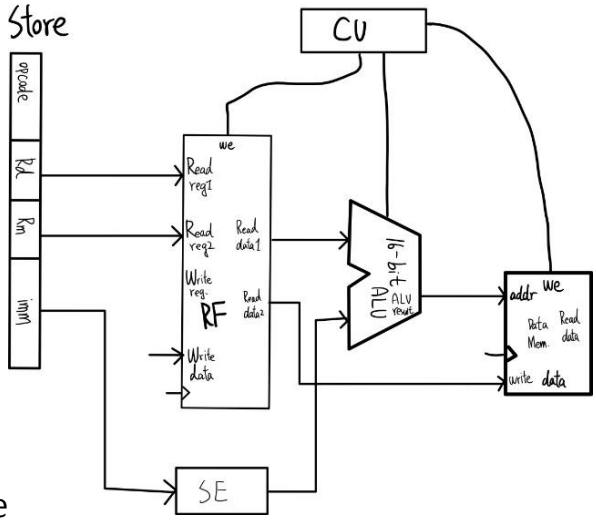


圖 6:Store

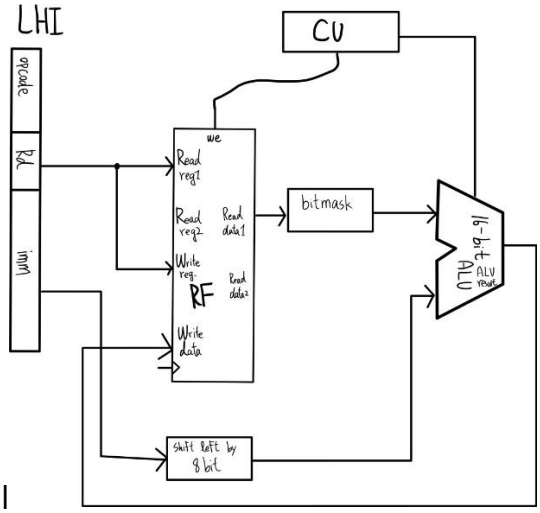


圖 7:LHI

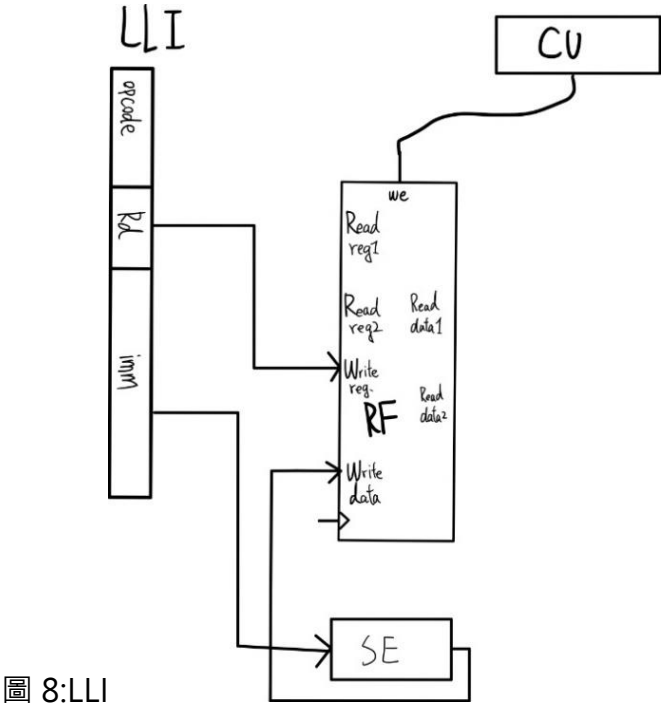


圖 8:LLI

圖 9:Branch & Jump

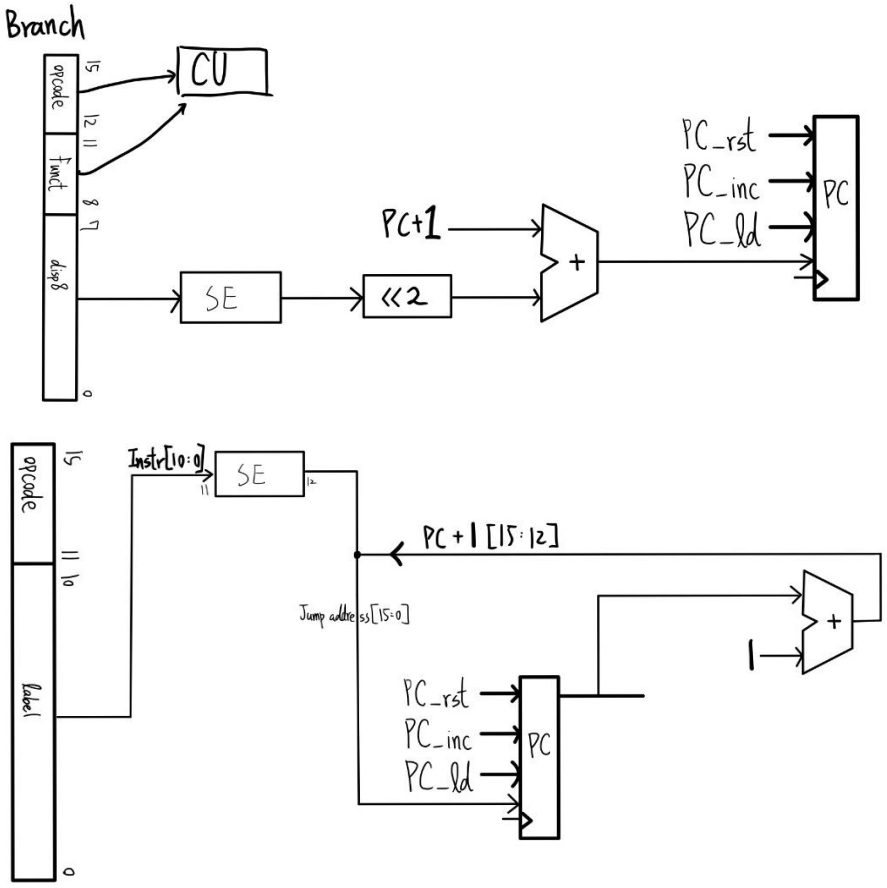
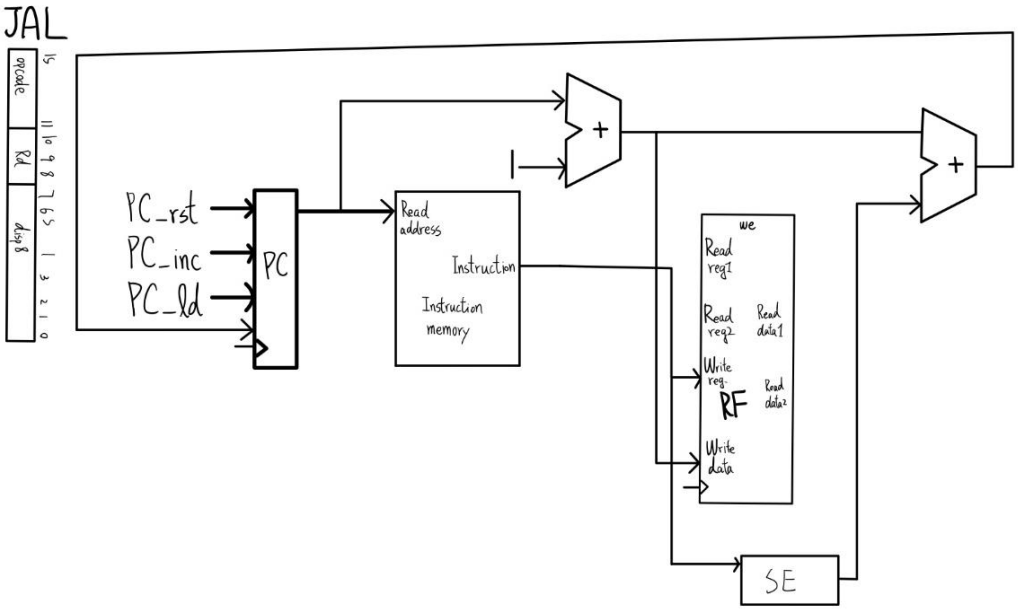


圖 10:JAL(兩種)


JAL
opcode
Rd
disp
15 11 10 9 8 7 6 5 4 3 2 1 0
PC_rst
PC_inc
PC_ld
PC
Read address
Instruction
Instruction memory
we
Read reg1
Read reg2
Read data1
Write reg.
RF
Read data2
Write data
SE

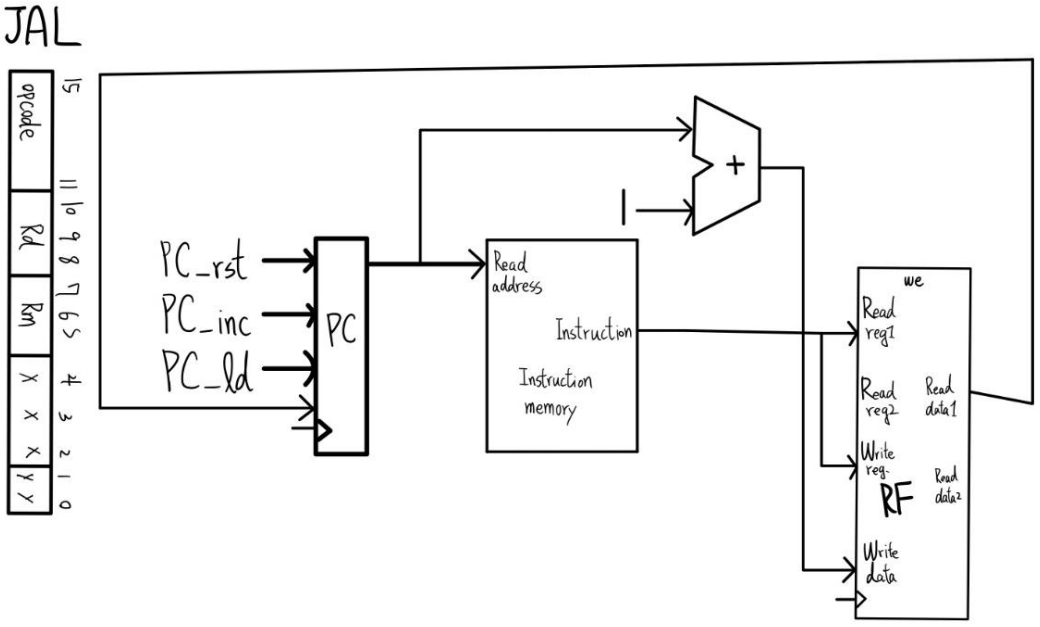
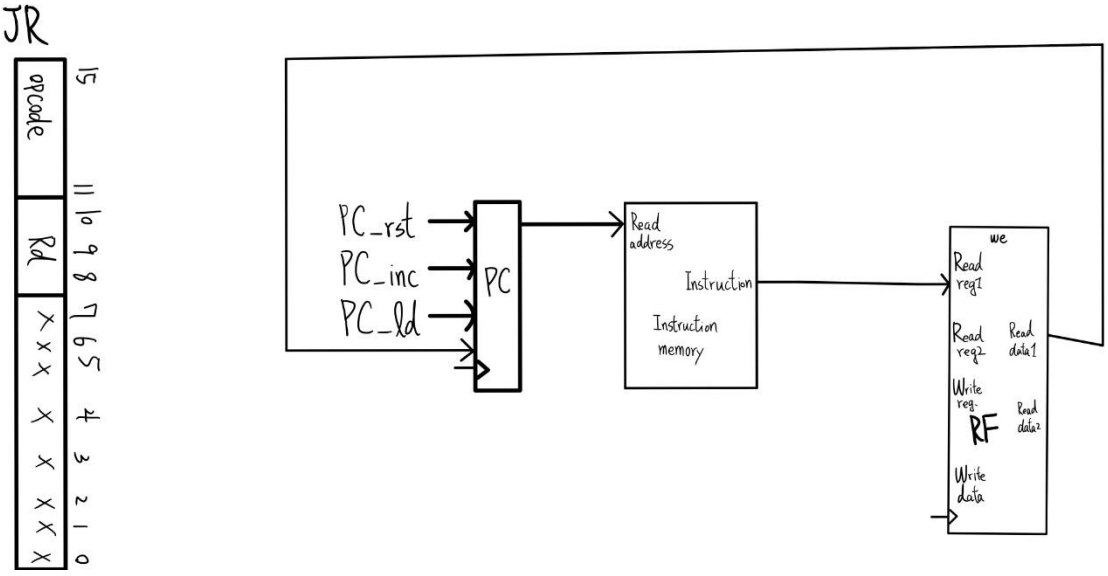

JAL
opcode
Rd
Rm
X X X Y Y
15 11 10 9 8 7 6 5 4 3 2 1 0
PC_rst
PC_inc
PC_ld
PC
Read address
Instruction
Instruction memory
we
Read reg1
Read reg2
Read data1
Write reg.
RF
Read data2
Write data

圖 11:JR



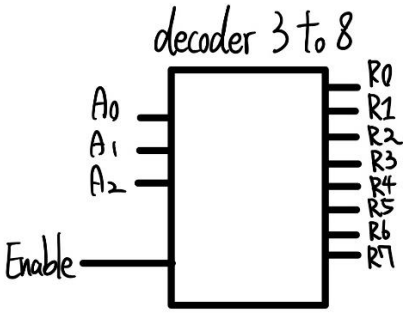
3.依序把各個元件組起來，以及增加信號控制以及 Mux 來做控制，順序為:

Store+Load 先合在一起->合成 R-type->合成 LLI 與 LHI->合成 Branch 與 Jump 與 JAL 與 JR 即完成。

其他模組設計圖與做法:

- **A 16-Bit Eight-Register Register File**

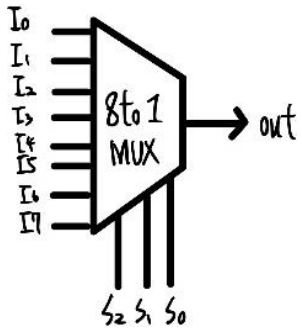
  1. enabled-controlled 3-to-8 noninverting output decoder



  作法:用 3 個信號線與 3 個 not gate 以及 8 顆 4and gate，其中用一條線接到每一顆 and gate 作為 enableed-controlled 作為輸出。

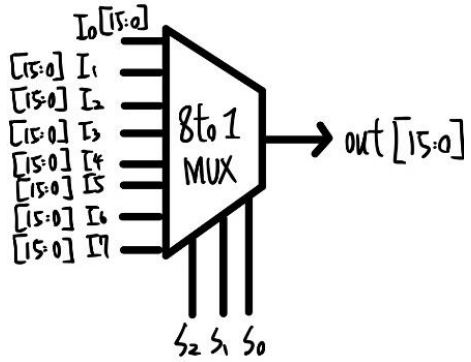
2. 8-to-1 multiplexer

作法:用兩顆 4 對 1 多工器再接上一個 2 對一多工器



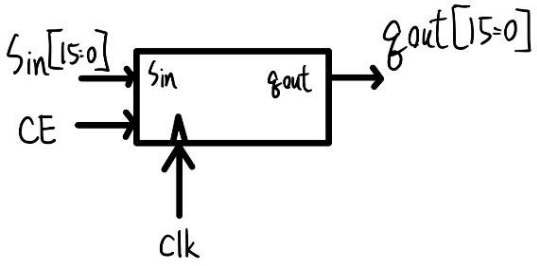
3. 16-bit 8-to-1 multiplexer

作法:把 16 顆 8 對 1 多工器疊加裡用 bus 來整合輸入輸出線



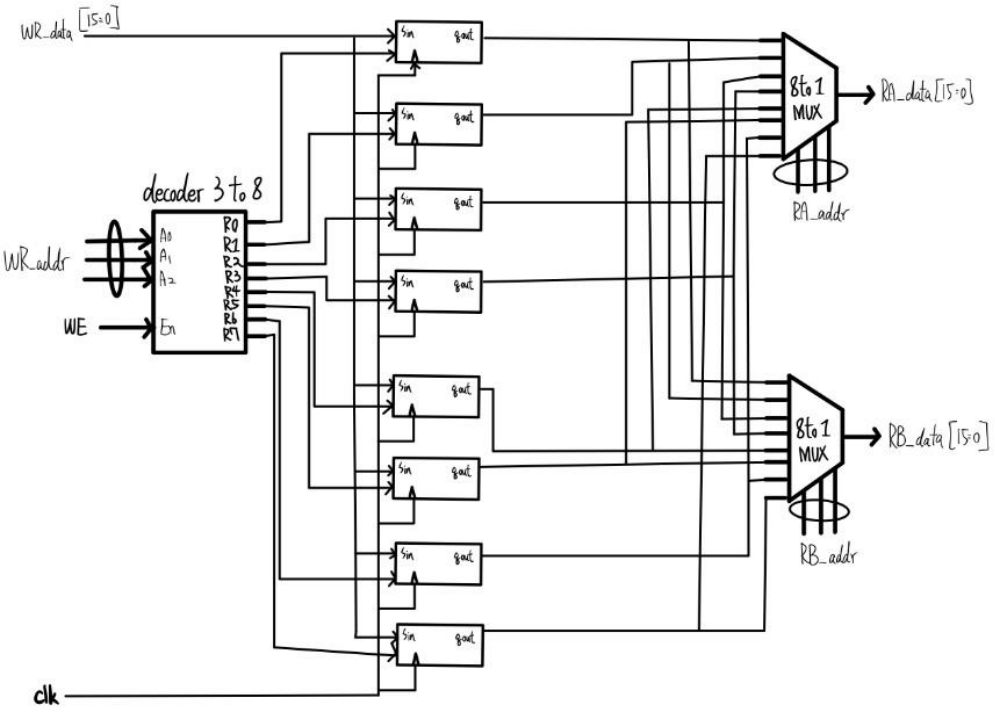
4. a 16-bit D-flip-flop register with clock-enable

作法:把 16 顆 D-flip-flop 用 bus 串接輸入與輸出，以及同步 clk 與 CE 線來控制 clock-enable



5. register file

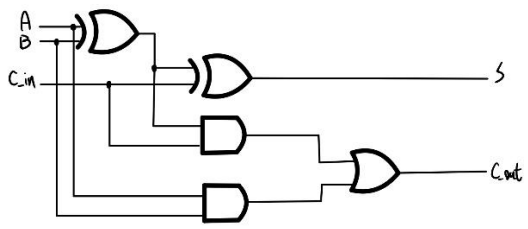
作法:把 3-to-8 與 8 顆 reg 做串接，八顆 reg 再分別跟兩個 8 對 1 多工器做串接，再去串接其他信號線，來做出一個 RF



- **A 16-Bit ALU**

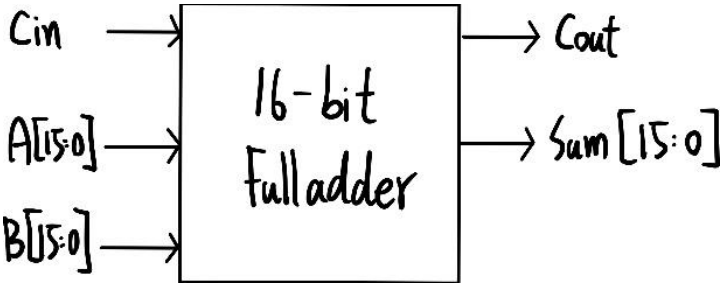
1. full adder

作法:用兩個 XOR gate 與兩個 and gate 與一個 or gate 來做



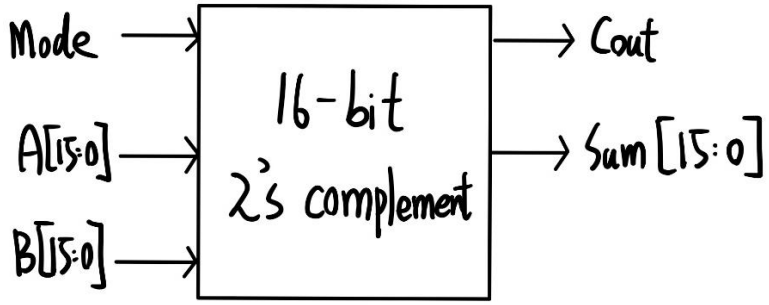
2. 16bit-adder

作法:把 16 顆 full adder 做疊加，用 bus 串聯輸入以及輸出



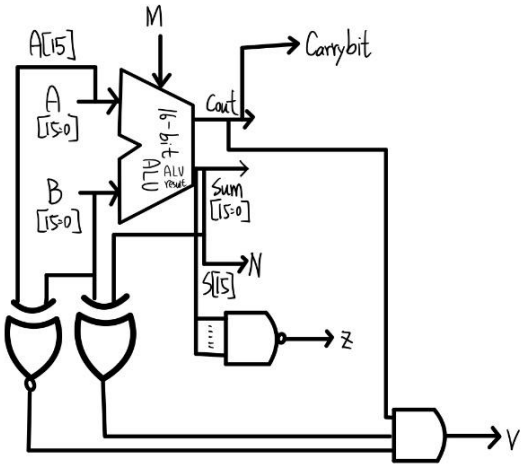
3. 16-bit 2’s complement adder

作法:把 16bit-adder 的 cin 與輸入 B 與 16 個 XOR 做串接，這樣就可以做出 2 補數加減法



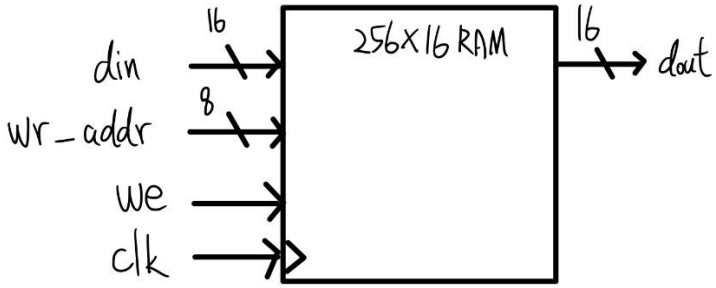
4. ALU

作法:要做 flag，cout 就是 flag C，sum 的 MSB 為 flag N，sum0~15bit 做 nand 為 flag Z，A 的 MSB 與 B 的 MSB 做 XNOR，B 的 MSB 與 Sum 的 MSB 做 XOR，最後再與 cout 一起做 and 為 flag V



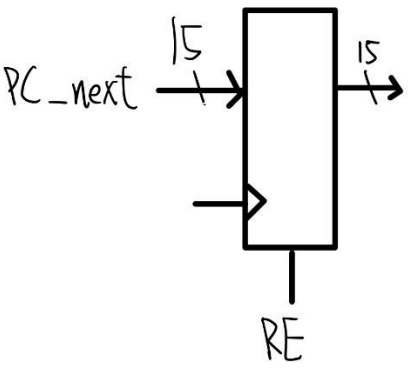
- **A 256×16 Memory Module**

  作法:把內建的 32X8RAM，先並排成 32X16RAM，接著再把 3to8 decoder，與一顆 16 對 1 多工器與 8 顆 32X16RAM 連結而成



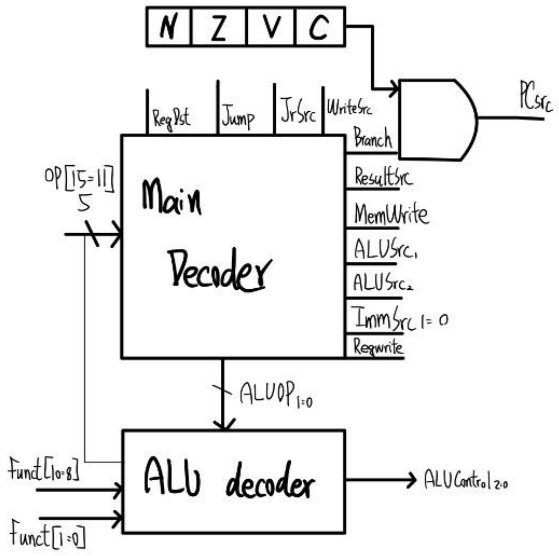
- **PC Circuitry**

  作法:用 D-flip-flop 作為 counter，每個 D-flip-flop 要有 reset，把 16 顆疊加之後就可以做成 16bit 的 counter



- **Instruction Decoder**

  作法:由 Main decoder 與 ALU decoder 一起組成



- **Complete Controller**

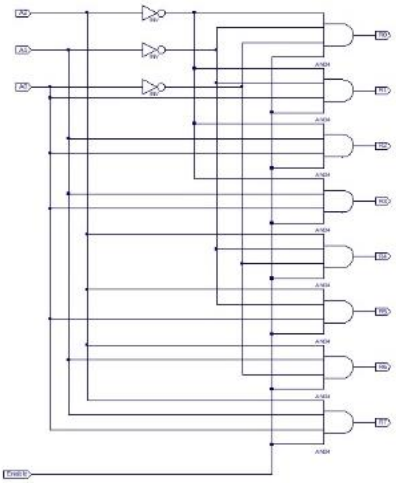
  以下是控制信號表:

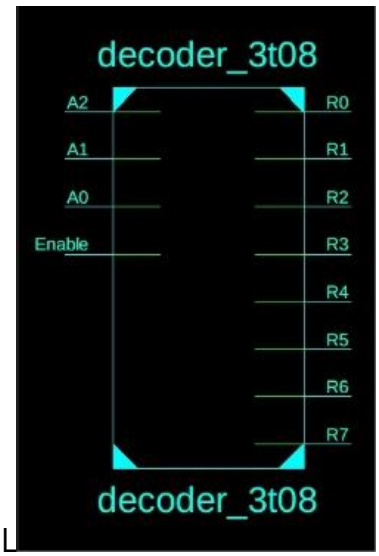
opcode	instr.	Funct.	RegDst	ALUSrc1	ALUSrc2	ResultSrc	MemWrit	RegWrite	Branch	ALUOp	WriteSrc	ImmSrc	Jump	JarSrc	Test
00000	ADD	00	0	0	0	0	0	1	0	0	0	0	0	0	0
00000	ADC	01	0	0	0	0	0	1	0	0	0	0	0	0	0
00000	SUB	10	0	0	0	0	0	1	0	1	0	0	0	0	0
00000	SBB	11	0	0	0	0	0	1	0	1	0	0	0	0	0
00001	LHI	xx	0	1	10	0	0	1	0	0	0	1	0	0	0
00010	LLI	xx	0	0	01	0	0	1	0	x	10	1	0	0	0
00011	LDR	xx	0	0	01	1	0	1	0	0	0	0	0	0	0
00101	STR	xx	1	0	01	0	1	0	0	0	0	0	0	0	0
00110	CMP	01	0	0	0	0	0	1	0	1	0	0	0	0	0
00111	ADDI	xx	0	0	01	0	0	1	0	0	0	0	0	0	0
01000	SUBI	xx	0	0	01	0	0	1	0	1	0	0	0	0	0
01011	MOV	xx	0	0	x	0	0	1	0	x	10	0	0	0	0
10000	JMP	xx	0	0	x	0	0	0	0	x	0	0	1	0	0
10001	JAL	xx	0	0	x	0	0	1	0	x	1	1	0	1	0
10010	JAL	xx	0	0	x	0	0	1	0	x	1	0	0	0	0
10011	JR	xx	0	0	x	0	0	0	0	x	0	0	0	1	0
11000	BCC	011	0	0	0	0	0	0	1	x	0	1	0	0	0
11000	BCS	010	0	0	0	0	0	0	1	x	0	1	0	0	0
11000	BNE	001	0	0	0	0	0	0	1	x	0	1	0	0	0
11000	BEQ	000	0	0	0	0	0	0	1	x	0	1	0	0	0
11001	B[AL]	110	0	0	0	0	0	0	1	x	0	1	0	0	0
11100	OutR	00	0	0	0	0	0	0	0	x	0	0	0	0	1
11100	HLT	01	0	0	0	0	0	0	0	x	0	0	0	0	1

2. Schematic Entry:

- **A 16-Bit Eight-Register Register File**

1. enabled-controlled 3-to-8 noninverting output decoder

Schematic 

RTL 

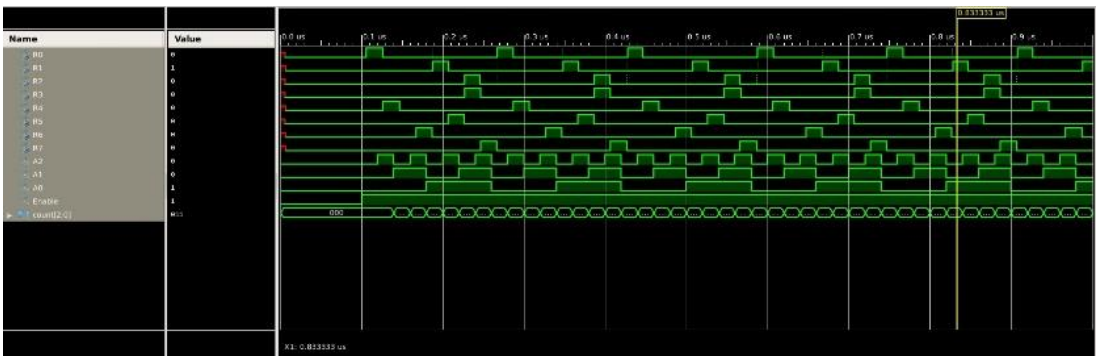
Testbench

```
// Verilog test fixture created from

`timescale 1ns / 1ps

module decoder_3t08_tb();

// Inputs
    reg A2;
    reg A1;
    reg A0;
    reg Enable;

// Output
    wire R0;
    wire R1;
    wire R2;
    wire R3;
    wire R4;
    wire R5;
    wire R6;
    wire R7;

//temporary variable

    reg [2:0] count = 3'd0;
// Instantiate the UUT
    decoder_3t08 uut (
        .A2(A2),
        .A1(A1),
        .A0(A0),
        .Enable(Enable),
        .R0(R0),
        .R1(R1),
        .R2(R2),
        .R3(R3),
        .R4(R4),
        .R5(R5),
```

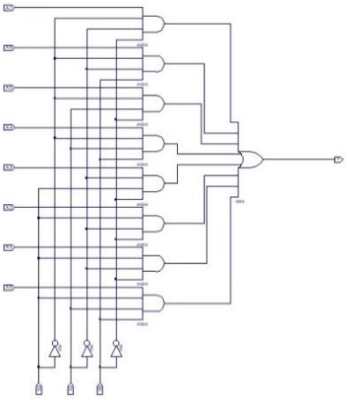
```
38          .R6(R6),
39          .R7(R7)
40      );
41  // Initialize Inputs
42
43
44      initial begin
45        A2 = 1'b0;
46        A1 = 1'b0;
47        A0 = 1'b0;
48        Enable = 1'b0;
49  // Wait 100 ns for global reset to finish
50
51        #100;
52
53        Enable = 1'b1;
54
55        #20;
56        for (count = 0; count < 8; count = count + 1'b1)
57
58        begin
59
60         {A0,A1,A2} = {A0,A1,A2} + 1'b1;
61
62         #20;
63
64        end
65
66        Enable = 1'b0;
67      end
68  endmodule
```

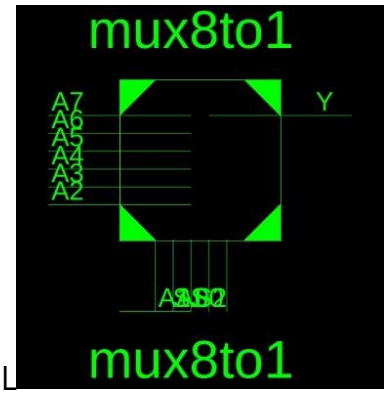
Behavioral

Post-Route



2. 8-to-1 multiplexer

Schematic 

RTL 

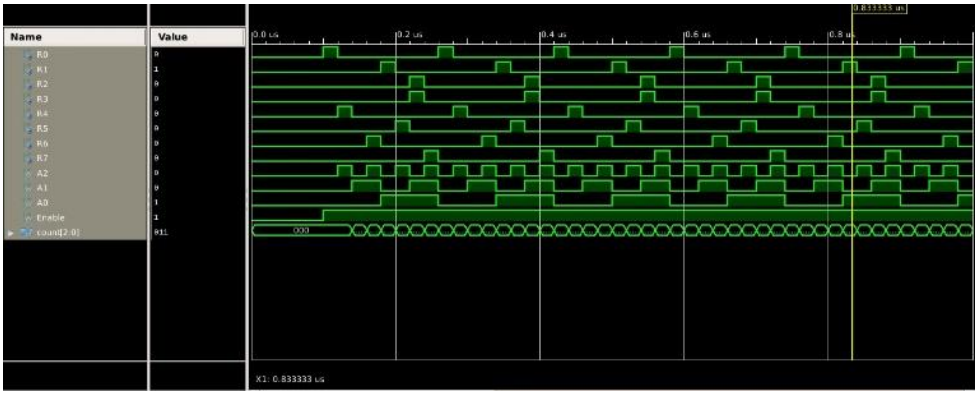
Testbench

```
`timescale 1ns / 1ps

module mux8to1_tb();

// Inputs
   reg [7:0] A; //8 input signals
   reg [2:0] S; //select signals

// Output
   wire Y;

// Instantiate the UUT
   mux8to1 UUT (
       .Y(Y),
       .A7(A[7]),
       .A6(A[6]),
       .A5(A[5]),
       .A4(A[4]),
       .A3(A[3]),
       .A2(A[2]),
       .A1(A[1]),
       .A0(A[0]),
       .S0(S[0]),
       .S1(S[1]),
       .S2(S[2])
   );
```

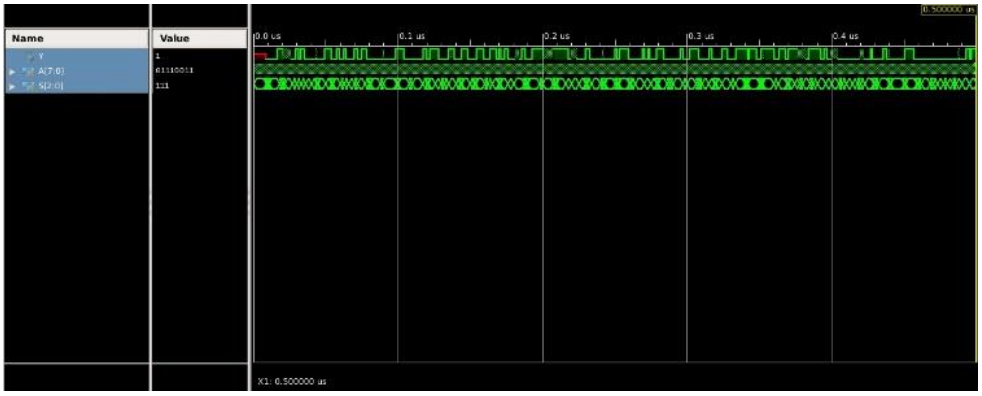
```
// Initialize Inputs
   initial begin
     $dumpfile("mux8to1.vcd");
     $dumpvars(0, mux8to1_tb);
      //A[0] = 1'b0;
      //A[1] = 1'b0;
      //A[2] = 1'b0;
      //A[3] = 1'b0;
      //A[4] = 1'b0;
      //A[5] = 1'b0;
      //A[6] = 1'b0;
      //A[7] = 1'b0;
      A = 8'b00000000;
      S[0] = 1'b0;
      S[1] = 1'b0;
      S[2] = 1'b0;

     #500 $finish;
   end
   always #1 A[0]=~A[0];
   always #2 A[1]=~A[1];
   always #3 A[2]=~A[2];
   always #4 A[3]=~A[3];
   always #5 A[4]=~A[4];
   always #6 A[5]=~A[5];
   always #7 A[6]=~A[6];
   always #8 A[7]=~A[7];
   always #9 S[0]=~S[0];
   always #10 S[1]=~S[1];
```

```
      //A[2] = 1'b0;
      //A[3] = 1'b0;
      //A[4] = 1'b0;
      //A[5] = 1'b0;
      //A[6] = 1'b0;
      //A[7] = 1'b0;
      A = 8'b00000000;
      S[0] = 1'b0;
      S[1] = 1'b0;
      S[2] = 1'b0;

     #500 $finish;
   end
   always #1 A[0]=~A[0];
   always #2 A[1]=~A[1];
   always #3 A[2]=~A[2];
   always #4 A[3]=~A[3];
   always #5 A[4]=~A[4];
   always #6 A[5]=~A[5];
   always #7 A[6]=~A[6];
   always #8 A[7]=~A[7];
   always #9 S[0]=~S[0];
   always #10 S[1]=~S[1];
   always #11 S[2]=~S[2];
   //always@ (A[0] or A[1] or A[2] or A[3] or A[4] or A[5] or A[6] or A[7] or S[0] or S[1] or S[2])
   always@ (Y)
    $monitor("At time = %t, Output = %d", $time, Y);
endmodule
```

Behavioral

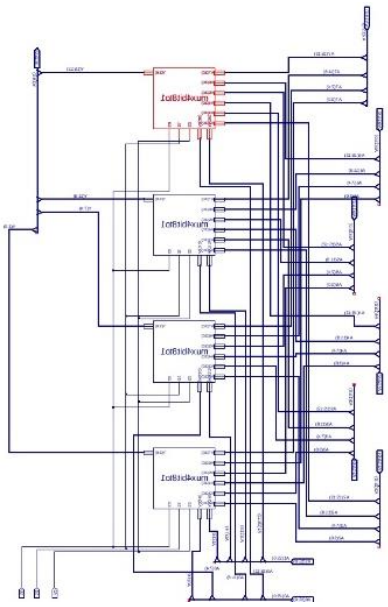


Post-Route

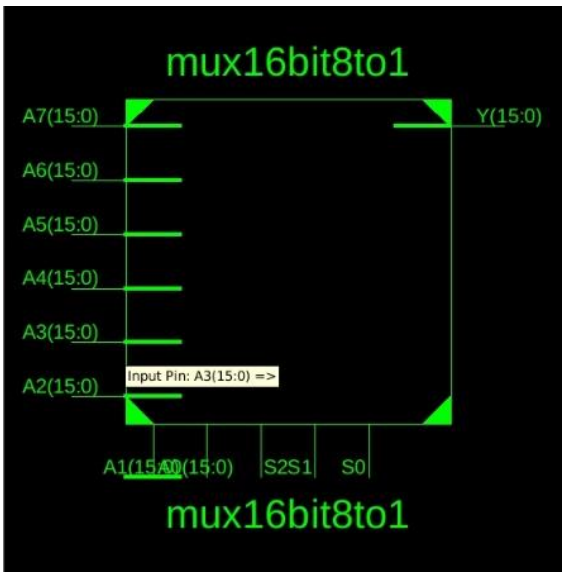


3. 16-bit 8-to-1 multiplexer

Schematic



RTL

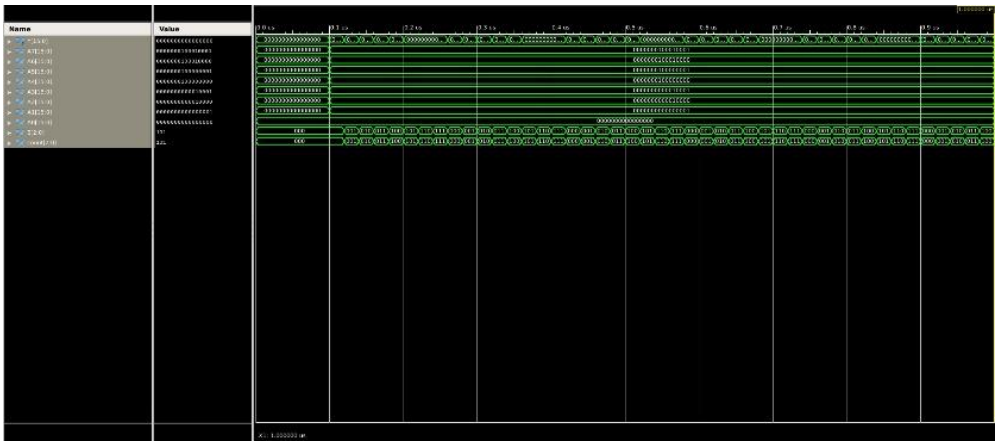


Testbench

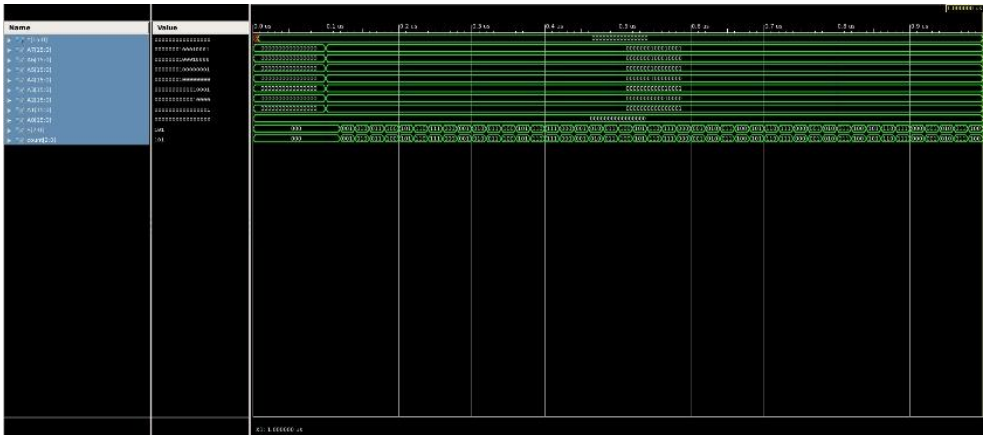
```
`timescale 1ns / 1ps

module mux16bit8to1_tb();

// Inputs
   reg [15:0] A7; //16 input signals
   reg [15:0] A6;
   reg [15:0] A5;
   reg [15:0] A4;
   reg [15:0] A3;
   reg [15:0] A2;
   reg [15:0] A1;
   reg [15:0] A0;
   reg [2:0] S; //select signals

// Output
   wire [15:0] Y;

reg [2:0] count = 3'b0;
// Instantiate the UUT
   mux16bit8to1 UUT (
       .Y(Y),
       .A7(A7),
       .A6(A6),
       .A5(A5),
       .A4(A4),
       .A3(A3),
       .A2(A2),
       .A1(A1),
       .A0(A0),
       .S0(S[0]),
       .S1(S[1]),
       .S2(S[2])
   );
// Initialize Inputs
   initial begin
     S = 0;
     A0 = 0;
     A1 = 0;
     A2 = 0;
     A3 = 0;
     A4 = 0;
     A5 = 0;
     A6 = 0;
     A7 = 0;

     #100;
     S = 3'b0;
     A0 = 16'h0000;
     A1 = 16'h0001;
     A2 = 16'h0010;
     A3 = 16'h0011;
     A4 = 16'h0100;
     A5 = 16'h0101;
     A6 = 16'h0110;
     A7 = 16'h0111;
```

```
     A3 = 0;
     A4 = 0;
     A5 = 0;
     A6 = 0;
     A7 = 0;

     #100;
     S = 3'b0;
     A0 = 16'h0000;
     A1 = 16'h0001;
     A2 = 16'h0010;
     A3 = 16'h0011;
     A4 = 16'h0100;
     A5 = 16'h0101;
     A6 = 16'h0110;
     A7 = 16'h0111;

     for (count = 0; count < 111; count = count + 1'b1) begin
       S = count;

       #20;
     end


   end
endmodule
```

Behavioral

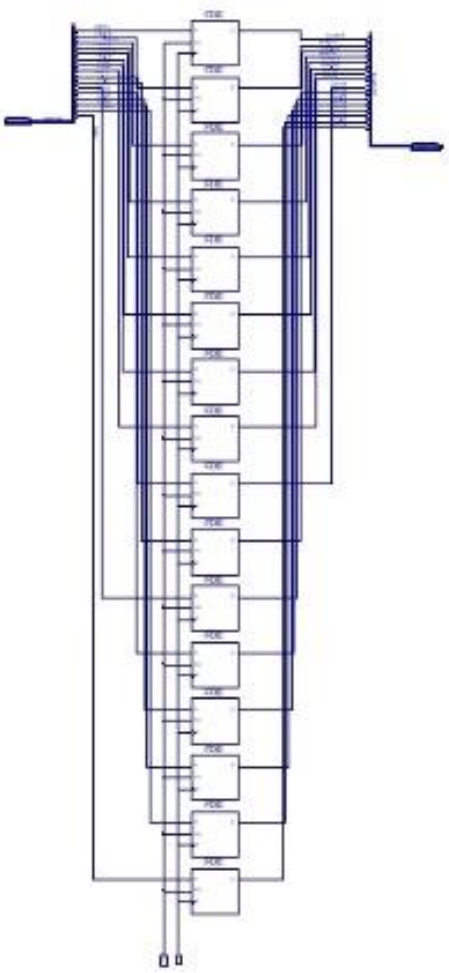


Post-Route

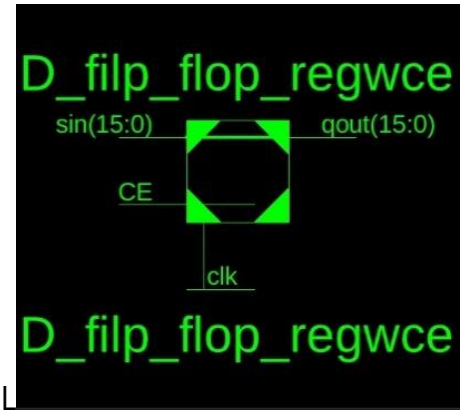


4. a 16-bit D-flip-flop register with clock-enable

Schematic



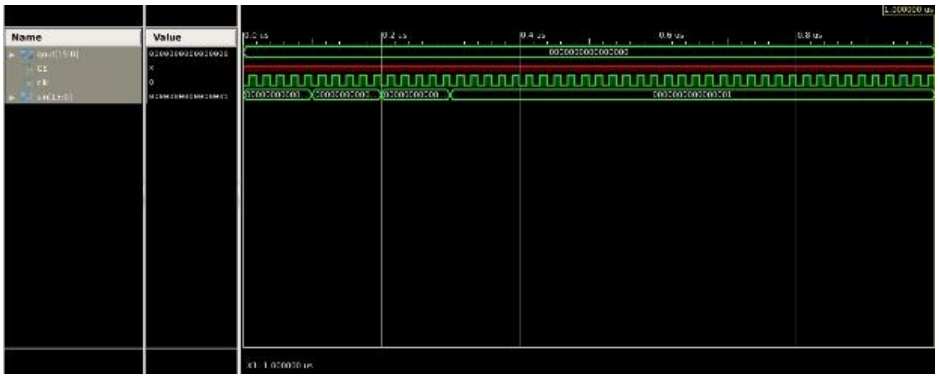
RTL



Testbench

```
module D_filp_flop_regwce_tb();

// Inputs

   reg CE;
   reg clk;
   reg [15:0] sin;
// Output
   wire [15:0] qout;

// Instantiate the UUT
   D_filp_flop_regwce UUT (
       .sin(sin),
       .CE(CE),
       .clk(clk),
       .qout(qout)
   );
// Initialize Inputs
   initial begin
     clk = 0;
         forever #10 clk = ~clk;
   end
   initial begin
    sin <= 16'h0000;
    #100
    sin <= 16'h0001;
   reg CE;
   reg clk;
   reg [15:0] sin;
// Output
   wire [15:0] qout;

// Instantiate the UUT
   D_filp_flop_regwce UUT (
       .sin(sin),
       .CE(CE),
       .clk(clk),
       .qout(qout)
   );
// Initialize Inputs
   initial begin
     clk = 0;
         forever #10 clk = ~clk;
   end
   initial begin
    sin <= 16'h0000;
    #100
    sin <= 16'h0001;
    #100
    sin <= 16'h0000;
    #100
    sin <= 16'h0001;
   end
endmodule
```

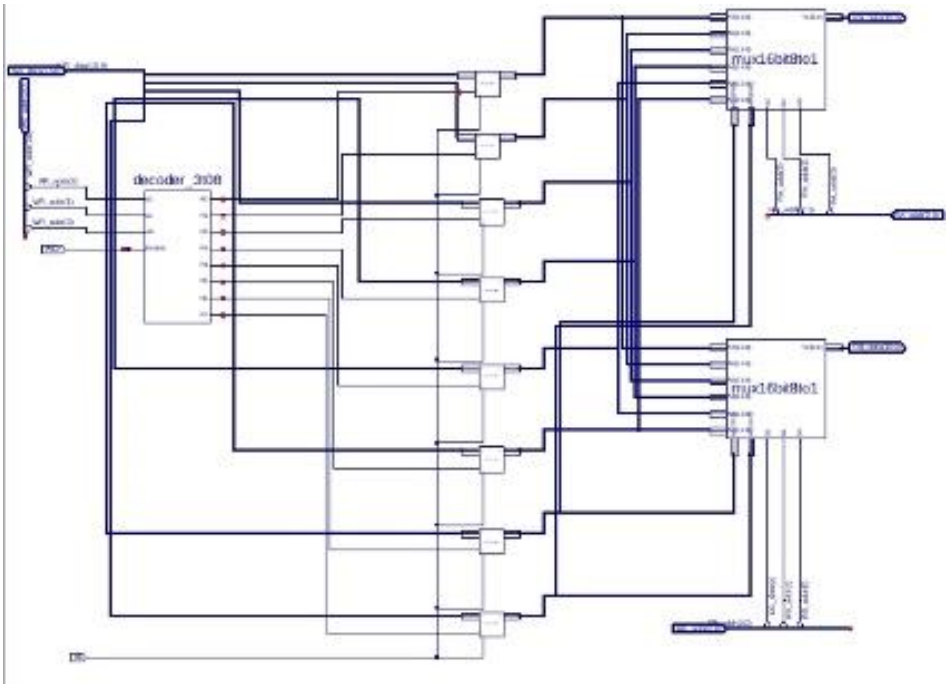
Behavioral



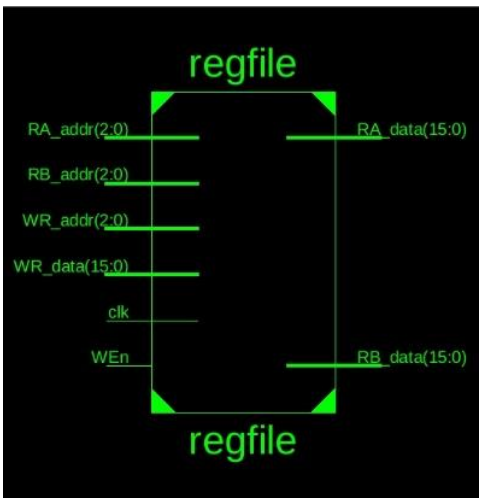
Post-Route

5. register file

Schematic



RTL



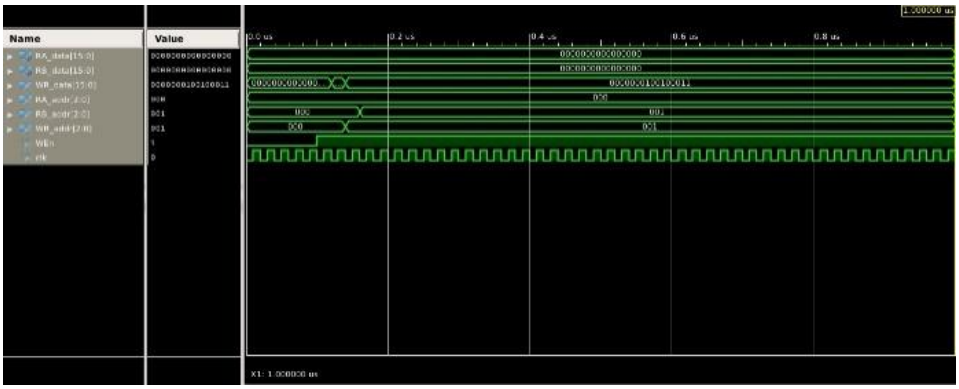
Testbench

```
`timescale 1ns / 1ps

module regfile_tb();

 // Inputs
   reg [15:0] WR_data;

   reg [2:0]RA_addr;

   reg [2:0]RB_addr;

   reg [2:0]WR_addr;

   reg WEn;

   reg clk;

 // Outputs

   wire [15:0] RA_data;

   wire [15:0] RB_data;

 // Instantiate the Unit Under Test (UUT)
```

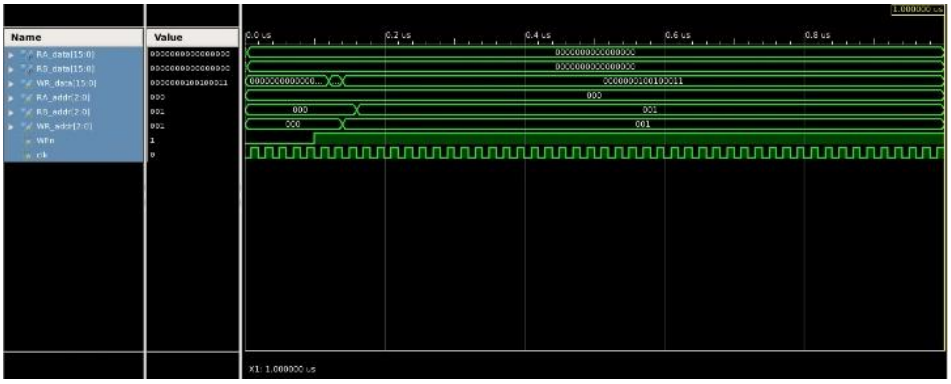
```
regfile uut (
        .WR_data(WR_data),
        .RA_addr(RA_addr),
        .RB_addr(RB_addr),
        .WR_addr(WR_addr),
        .WEn(WEn),
        .clk(clk),
        .RA_data(RA_data),
        .RB_data(RB_data)
        );
initial begin

 // Initialize Inputs

 WR_data  = 16'b0;

 WR_addr  = 3'b000;

 RA_addr  = 3'b000;

 RB_addr  = 3'b000;

 WEn  = 1'b0;

 clk  = 1'b0;
 // Wait 100 ns for global reset to finish

 #100;
```

```
  // Add stimulus here

   WEn  = 1'b1;

   #20;

   WR_data  = 16'habcd;

   WR_addr  = 3'h000;

   #20;

   WR_data  = 16'h0123;

   WR_addr  = 3'h001;

   #20;

   RA_addr  = 3'h000;

   RB_addr  = 3'h001;

 end
 always #10 clk = ~clk;


endmodule
```

Behavioral



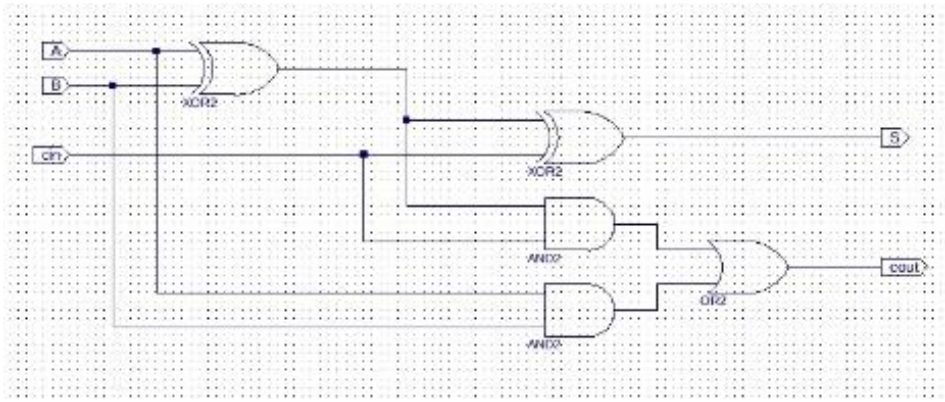
Post-Route



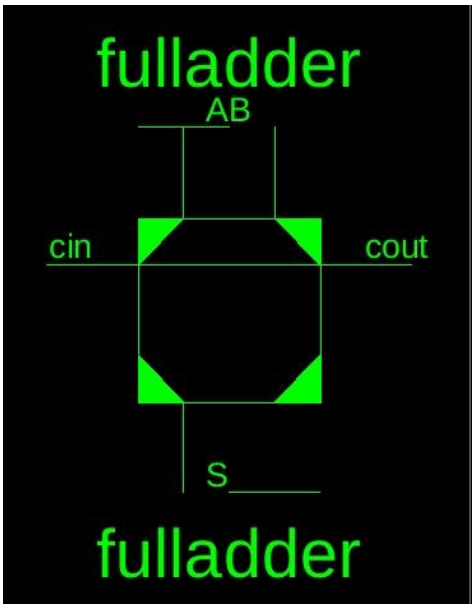
- **A 16-Bit ALU**

1. full adder

Schematic



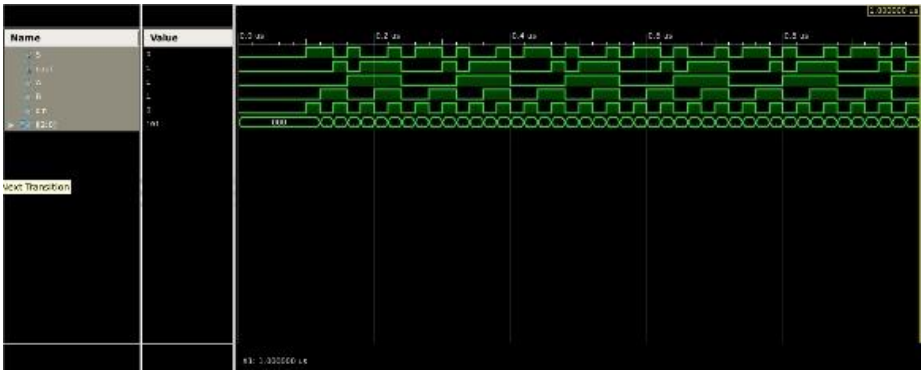
RTL



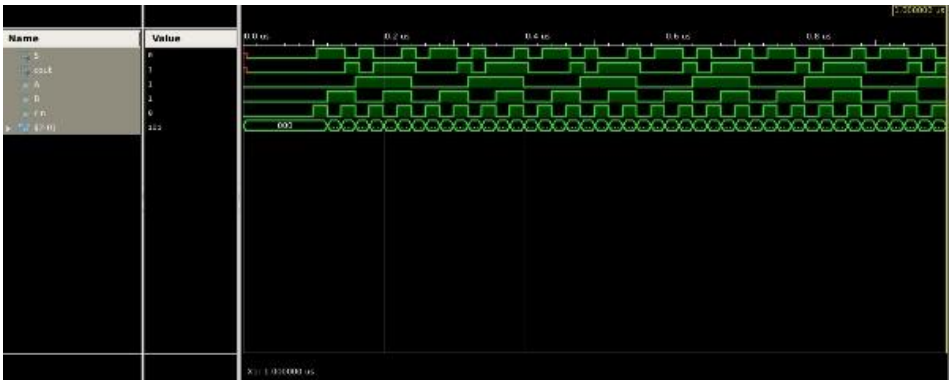
Testbench

```
`timescale 1ns / 1ps

module fulladder_tb();

// Inputs
 reg A;

 reg B;

 reg cin;

// Output
 wire S;
 wire cout;
//Temporary looping variable

 reg [2:0] i = 3'd0;

// Instantiate the UUT
   fulladder UUT (
       .A(A),
       .B(B),
       .cin(cin),
       .S(S),
       .cout(cout)
   );
// Initialize Inputs
initial begin
  A = 1'b0;

  B = 1'b0;

  cin = 1'b0;


  // Wait 100 ns for global reset to finish

  #100;



  // Add stimulus here

  for ( i = 0; i < 8; i = i + 1'b1)begin

   {A, B, cin} = {A, B, cin} + 1'b1;

   #20;

  end

 end

endmodule
```

Behavioral

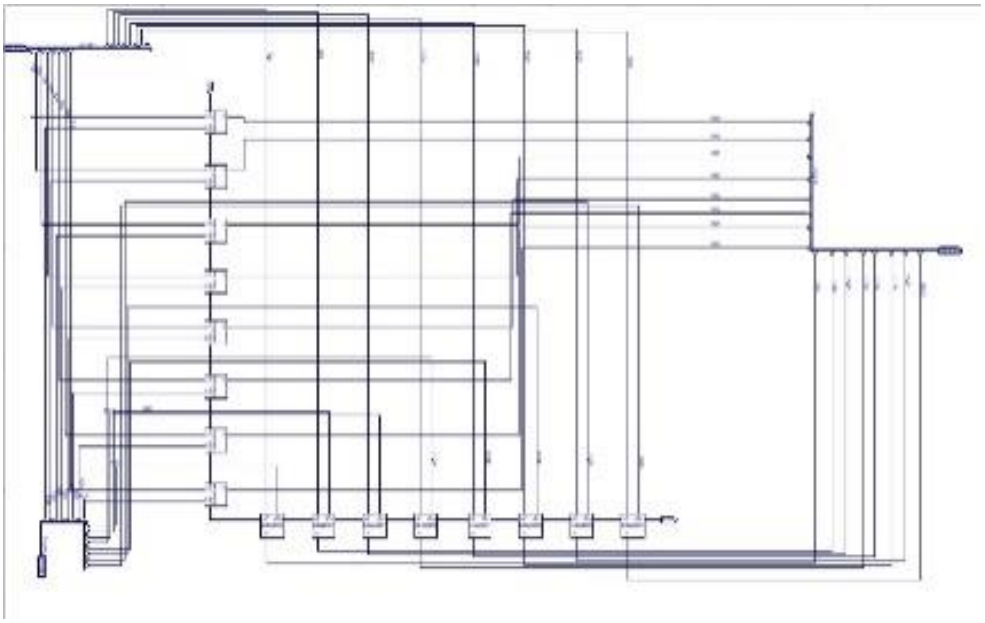


Post-Route

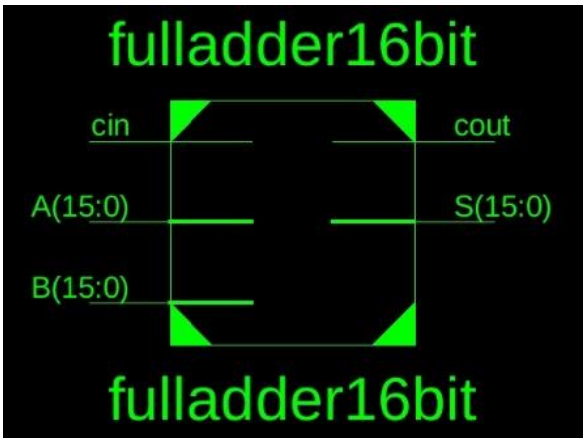


2. 16bit-adder

Schematic



RTL

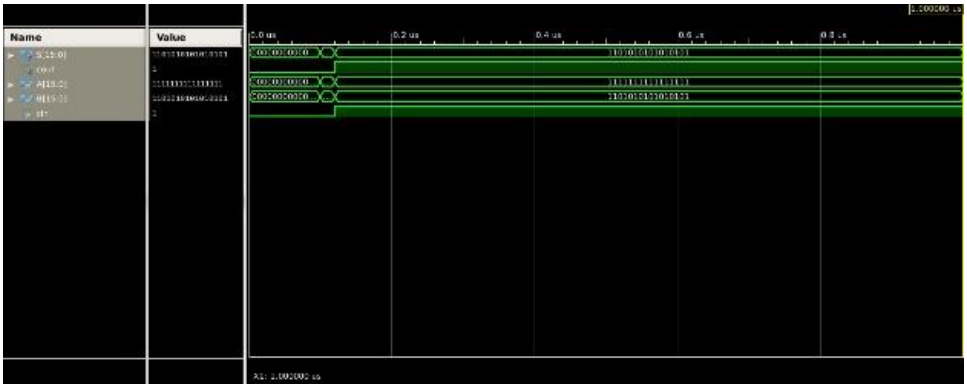


Testbench

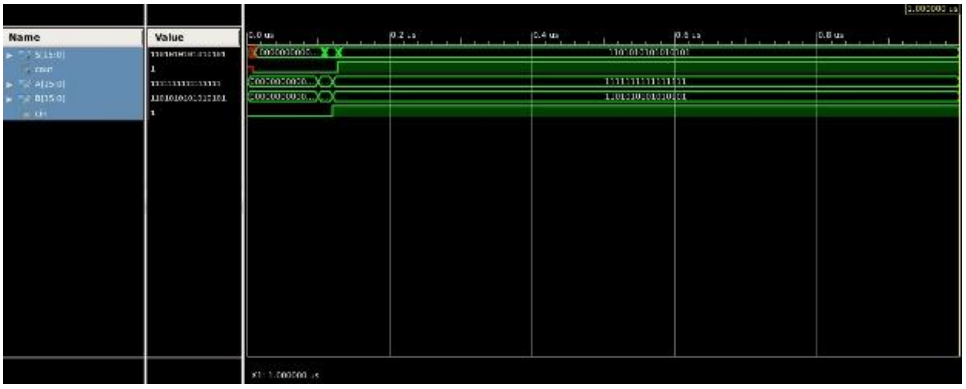
```
3  `timescale 1ns / 1ps
4
5  module fulladder16bit_tb();
6
7  // Inputs
8   reg [15:0] A;
9
10  reg [15:0] B;
11  reg cin;
12 // Output
13  wire [15:0] S;
14
15  wire cout;
16
17 // Bidirs
18
19 // Instantiate the UUT
20    fulladder16bit UUT (
21              .A(A),
22              .B(B),
23
24              .cin(cin),
25              .S(S),
26              .cout(cout)
27    );
```

```
28 // Initialize Inputs
29  initial begin
30    A = 16'b0;
31
32    B = 16'b0;
33
34    cin = 1'b0;
35   // Wait 100 ns for global reset to finish
36
37    #100;
38
39   // Add stimulus here
40
41    A = 16'b1011001100110011;
42
43    B = 16'b0100010001000100;
44
45    cin  = 1'b0;
46
47    #20;
48
49    A = 16'd65535;
50
51    B = 16'd54613;
52
53    cin = 1'b1;
54
55  end
56 endmodule
```

Behavioral

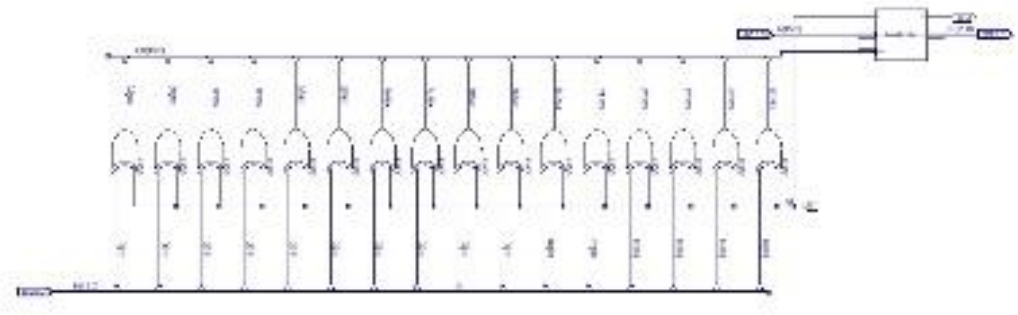


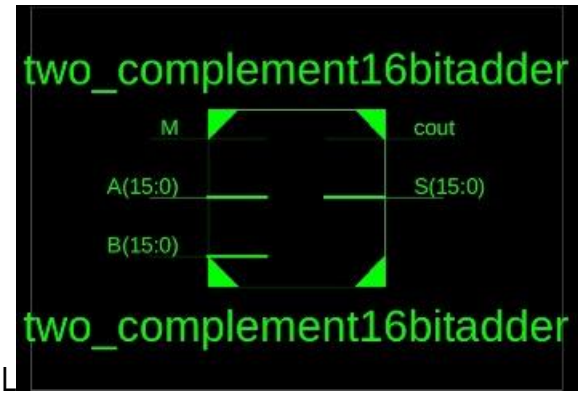
Post-Route



3. 16-bit 2’ s complement adder

Schematic



RTL 

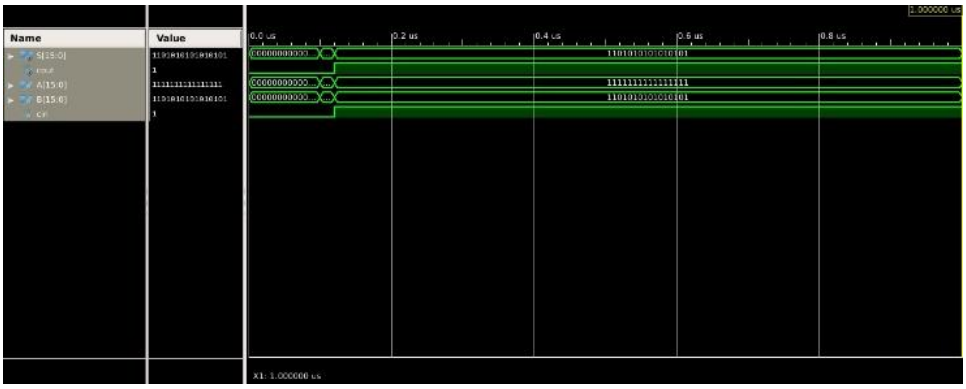
Testbench

```
3  `timescale 1ns / 1ps
4
5  module twos_complementadder_tb();
6
7  // Inputs
8   reg [15:0] A;
9   reg [15:0] B;
10  reg cin;
11  reg M; // 0 for addition, 1 for subtraction
12
13 // Output
14  wire [15:0] S;
15  wire cout;
16
17  // Instantiate the UUT
18  twos_complementadder UUT (
19              .A(A),
20              .B(B),
21              .cin(cin),
22              .S(S),
23              .cout(cout),
24              .M(M)
25    );
26
```

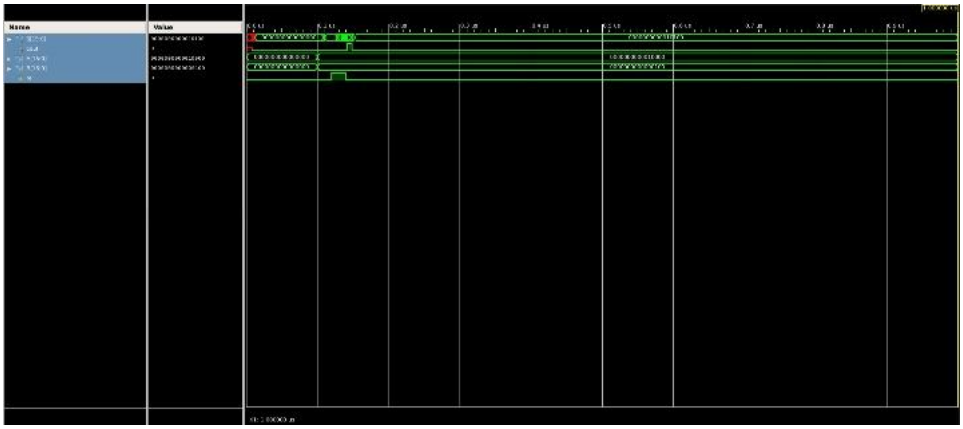
```
27 // Initialize Inputs
28 initial begin
29   A = 16'b0;
30   B = 16'b0;
31   cin = 1'b0;
32   M = 1'b0; // Start with addition mode
33
34  // Wait 100 ns for global reset to finish
35   #100;
36
37 // Test case 1: Addition (A + B)
38  A = 16'b0000000000010000;
39  B = 16'b0000000000000100;
40  cin = 1'b0;
41  M = 1'b0; // Addition mode
42  #20;
43
44   // Test case 2: Subtraction (A - B)
45  A = 16'b0000000000010000;
46  B = 16'b0000000000000100;
47  cin = 1'b0;
48  M = 1'b1; // Subtraction mode
49  #20;
50   // Test case 3: Addition with carry-in (A + B + Cin)
51  A = 16'b0000000000010000;
52  B = 16'b0000000000000100;
53  cin = 1'b1;
54  M = 1'b0; // Addition mode
```

```
43
44     // Test case 2: Subtraction (A - B)
45     A = 16'b0000000000010000;
46     B = 16'b0000000000000100;
47     cin = 1'b0;
48     M = 1'b1; // Subtraction mode
49     #20;
50     // Test case 3: Addition with carry-in (A + B + Cin)
51     A = 16'b0000000000010000;
52     B = 16'b0000000000000100;
53     cin = 1'b1;
54     M = 1'b0; // Addition mode
55     #20;
56
57   end
58 endmodule
59
60
```

Behavioral

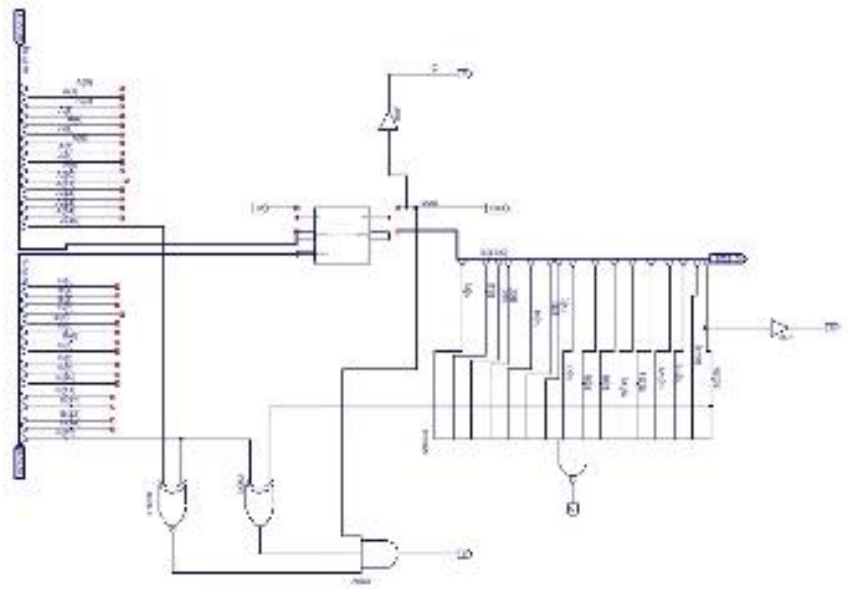


Post-Route

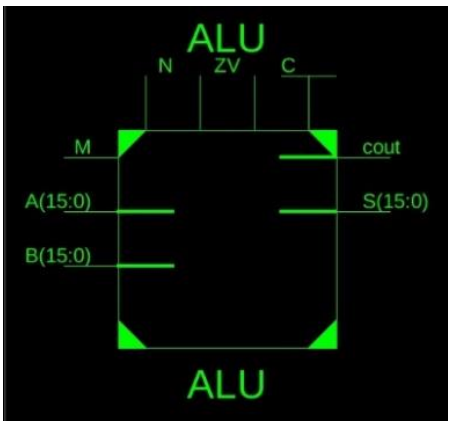


4. ALU

Schematic



RTL

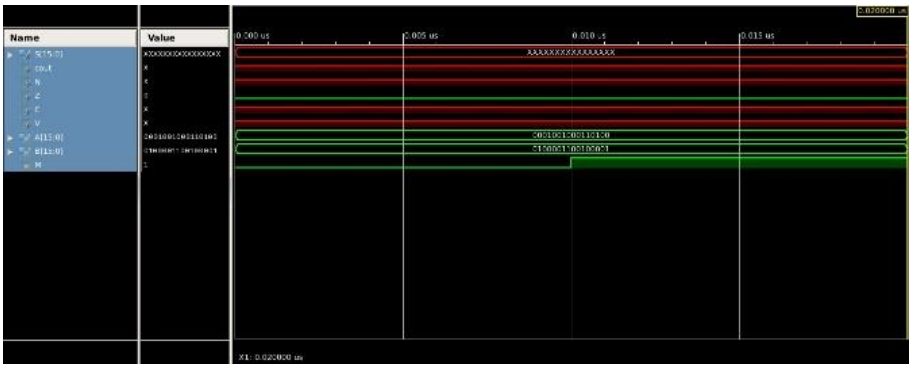


Testbench

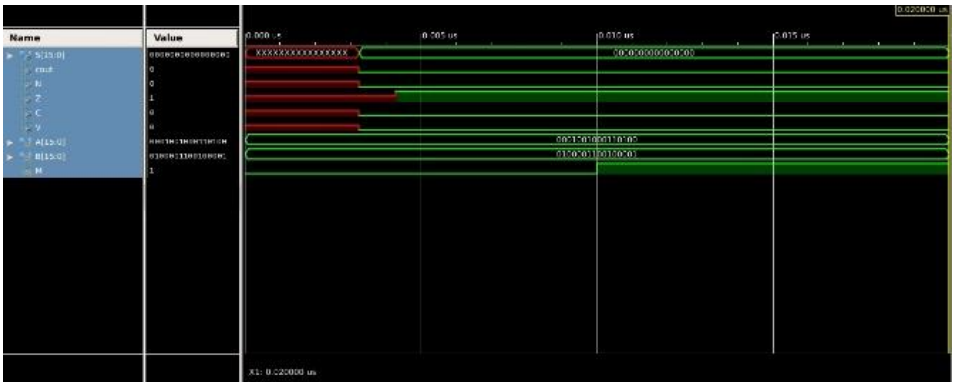
```
module ALU_tb();

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

    // Output
  wire [15:0] S;
  wire cout;
  wire N;  // Negative flag
  wire Z;  // Zero flag
  wire C;  // Carry flag
  wire V;  // Overflow flag
  //Temporary variable;
    // Instantiate the UUT
  ALU UUT (
    .A(A),
    .B(B),
    .S(S),
    .cout(cout),
    .M(M),
    .N(N),
    .Z(Z),
    .C(C),
    .V(V)
  );
```

```
  // Initialize Inputs
  initial begin
    A = 16'h1234;
    B = 16'h4321;
    M = 1'b0;  // Set the mode (0 for addition, 1 for subtraction)

    #10;
    M = 1'b1;
    #10;
    $finish;
  end
  initial begin
    $monitor("At time %d, Sum = %d, Zero = %b, Negative = %b, Carry = %b, Overflow = %b", $time, S, Z, N, C, V);
  end

endmodule
```

Behavioral

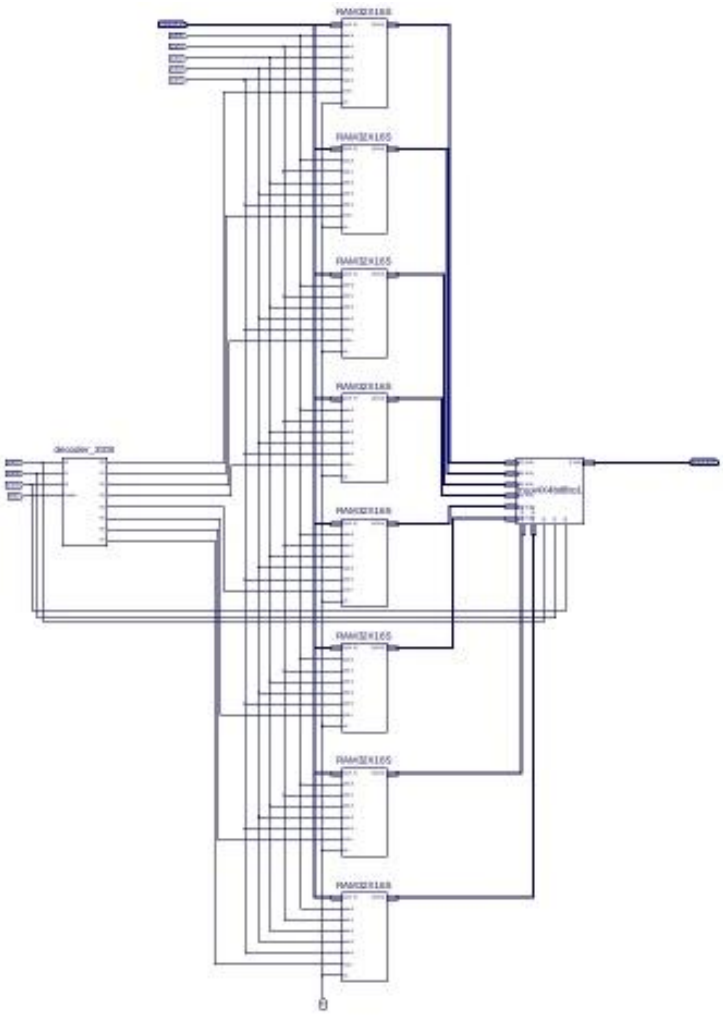


Post-Route

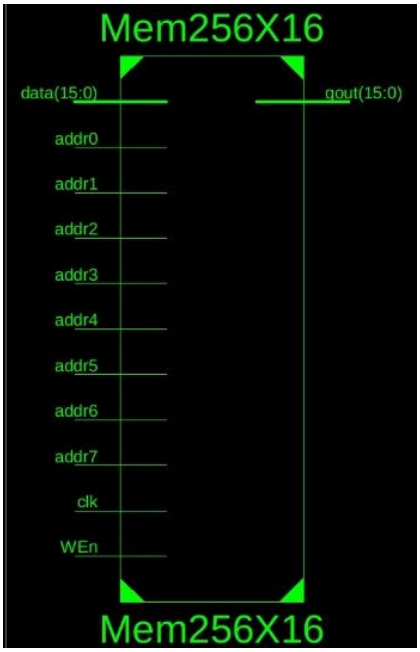


- **A 256×16 Memory Module**

Schematic



RTL



Testbench

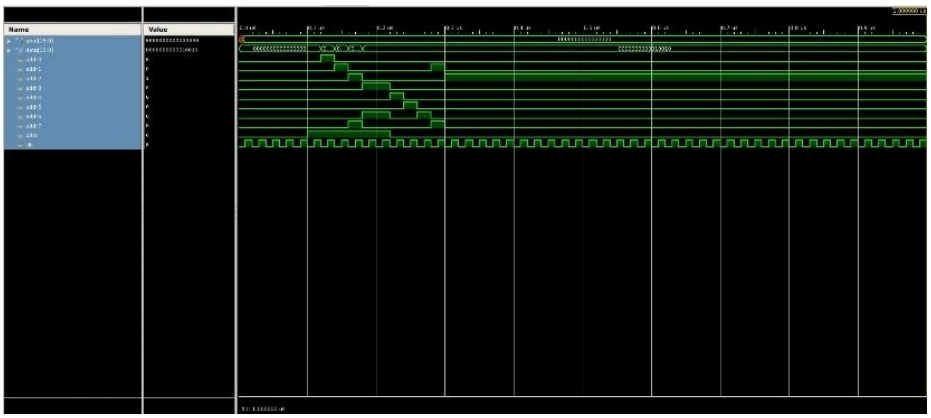
```
`timescale 1ns / 1ps

module Mem256X16_tb();

// Inputs
   reg [15:0] data;
   reg addr0;
   reg addr1;
   reg addr2;
   reg addr3;
   reg addr4;
   reg addr5;
   reg addr6;
   reg addr7;
   reg WEn;
   reg clk;

// Output
   wire [15:0] qout;

// Instantiate the UUT
   Mem256X16 uut (
      .data(data),
      .qout(qout),
      .addr0(addr0),
      .addr1(addr1),
      .addr2(addr2),
      .addr3(addr3),
      .addr4(addr4),
      .addr5(addr5),
      .addr6(addr6),
      .addr7(addr7),
      .WEn(WEn),
      .clk(clk)
   );
// Initialize Inputs
   initial begin
      data = 16'h0;
      addr0 = 1'b0;
      addr1 = 1'b0;
      addr2 = 1'b0;
      addr3 = 1'b0;
      addr4 = 1'b0;
      addr5 = 1'b0;
      addr6 = 1'b0;
      addr7 = 1'b0;
      WEn = 1'b0;
      clk = 1'b0;
   // Wait 100 ns for global reset to finish

      #100;
   // Add stimulus here
      data  = 16'h0;
      addr0 = 1'b0;
      addr1 = 1'b0;
      addr2 = 1'b0;
      addr3 = 1'b0;
      addr4 = 1'b0;
      addr5 = 1'b0;
      addr6 = 1'b0;
      addr7 = 1'b0;
      WEn  = 1'b1;
      #20;
      data  = 16'h1;
      addr0 = 1'b1;
      addr1 = 1'b0;
      addr2 = 1'b0;
      addr3 = 1'b0;
      addr4 = 1'b0;
      addr5 = 1'b0;
      addr6 = 1'b0;
      addr7 = 1'b0;
      #20;
      data  = 16'h10;
      addr0 = 1'b0;
      addr1 = 1'b1;
      addr2 = 1'b0;
      addr3 = 1'b0;
      addr4 = 1'b0;
      addr5 = 1'b0;
      addr6 = 1'b0;
      addr7 = 1'b0;
      #20;
      data  = 16'h6;
      addr0 = 1'b0;
      addr1 = 1'b0;
      addr2 = 1'b1;
      addr3 = 1'b0;
```

```
91          addr4 = 1'b0;
92          addr5 = 1'b0;
93          addr6 = 1'b0;
94          addr7 = 1'b1;
95          #20;
96          data  = 16'h12;
97          addr0 = 1'b0;
98          addr1 = 1'b0;
99          addr2 = 1'b0;
100         addr3 = 1'b1;
101         addr4 = 1'b0;
102         addr5 = 1'b0;
103         addr6 = 1'b1;
104         addr7 = 1'b0;
105         #40;
106         addr0 = 1'b0;
107         addr1 = 1'b0;
108         addr2 = 1'b0;
109         addr3 = 1'b0;
110         addr4 = 1'b1;
111         addr5 = 1'b0;
112         addr6 = 1'b0;
113         addr7 = 1'b0;
114         WEn   = 1'b0;
115         #20;
116         addr0 = 1'b0;
117         addr1 = 1'b0;
118         addr2 = 1'b0;
119         addr3 = 1'b0;
120         addr4 = 1'b0;
121         addr5 = 1'b1;
122         addr6 = 1'b0;
123         addr7 = 1'b0;
124         #20;
125         addr0 = 1'b0;
126         addr1 = 1'b0;
127         addr2 = 1'b0;
128         addr3 = 1'b0;
129         addr4 = 1'b0;
130         addr5 = 1'b0;
131         addr6 = 1'b1;
132         addr7 = 1'b0;
```

```
109         addr3 = 1'b0;
110         addr4 = 1'b1;
111         addr5 = 1'b0;
112         addr6 = 1'b0;
113         addr7 = 1'b0;
114         WEn   = 1'b0;
115         #20;
116         addr0 = 1'b0;
117         addr1 = 1'b0;
118         addr2 = 1'b0;
119         addr3 = 1'b0;
120         addr4 = 1'b0;
121         addr5 = 1'b1;
122         addr6 = 1'b0;
123         addr7 = 1'b0;
124         #20;
125         addr0 = 1'b0;
126         addr1 = 1'b0;
127         addr2 = 1'b0;
128         addr3 = 1'b0;
129         addr4 = 1'b0;
130         addr5 = 1'b0;
131         addr6 = 1'b1;
132         addr7 = 1'b0;
133         #20;
134         addr0 = 1'b0;
135         addr1 = 1'b1;
136         addr2 = 1'b0;
137         addr3 = 1'b0;
138         addr4 = 1'b0;
139         addr5 = 1'b0;
140         addr6 = 1'b0;
141         addr7 = 1'b1;
142         #20;
143         addr0 = 1'b0;
144         addr1 = 1'b0;
145         addr2 = 1'b1;
146         addr3 = 1'b0;
147         addr4 = 1'b0;
148         addr5 = 1'b0;
149         addr6 = 1'b0;
150         addr7 = 1'b0;
151     end
152     always #10 clk = ~clk;
153 endmodule
```

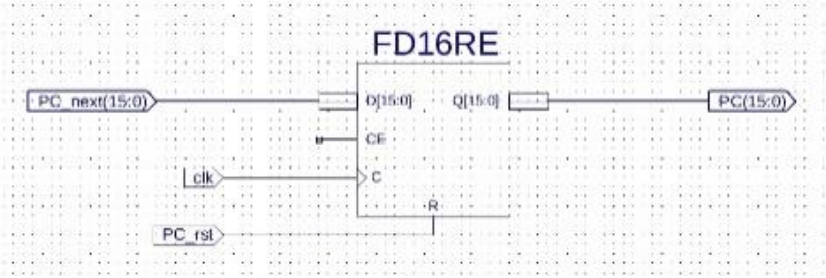
Behavioral

Post-Route

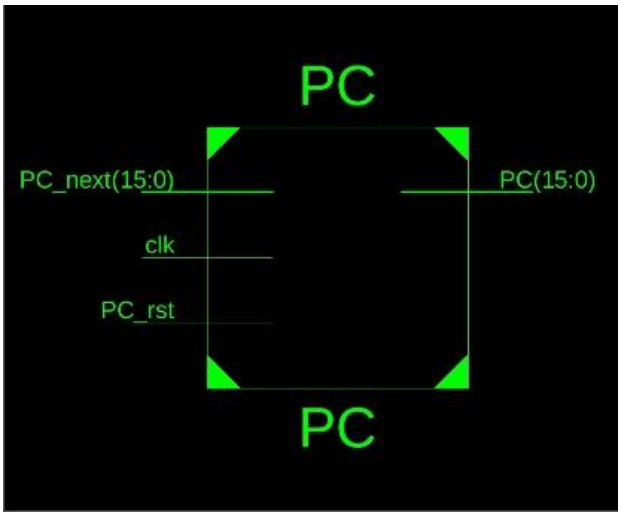


- **PC Circuitry**

Schematic



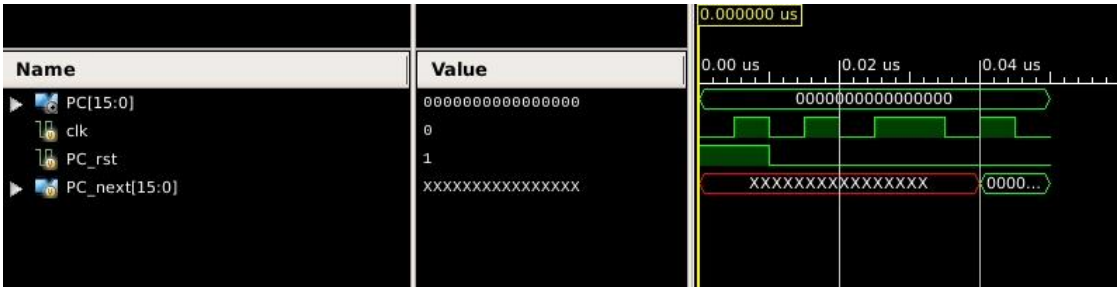
RTL



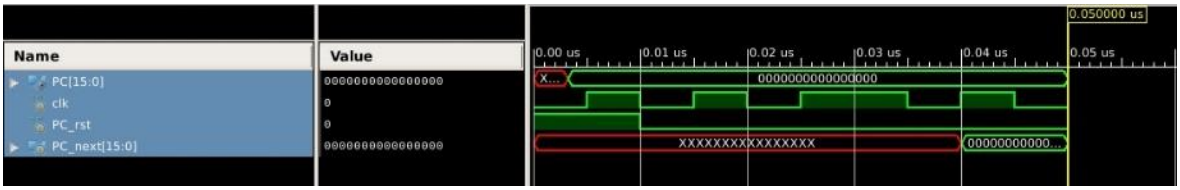
Testbench

```
`timescale 1ns / 1ps

module PC_tb();

// Inputs
   reg clk;
   reg PC_rst;
   reg [15:0] PC_next;

// Output
   wire [15:0] PC;

// Instantiate the UUT
   PC UUT (
      .clk(clk),
      .PC_rst(PC_rst),
      .PC_next(PC_next),
      .PC(PC)
   );
// Initialize Inputs
   initial begin
      // Initialize Inputs
      PC_rst = 0;
      clk = 0;

      // Apply a reset pulse
      PC_rst = 1;
      #10;
      PC_rst = 0;
      #10;

      // Clock the design
      clk = 1;
      #10;
      clk = 0;
      #10;
      $display("Initial PC: %h", PC);
      case (PC_next)
         16'h0000: PC_next = 16'h0001;  // Increment by 1
         16'h0001: PC_next = 16'h0010;  // Increment by 2
         16'h0010: PC_next = 16'h0000;  // Wrap around to 0
         default: PC_next = 16'h0000;  // Default case
      endcase
      #10;
      $display("New PC: %h", PC);
         // Finish the simulation
         $finish;
      end
      always begin
         #5 clk = ~clk;  // Toggle clock every 5 time units
      end
endmodule
```

Behavioral

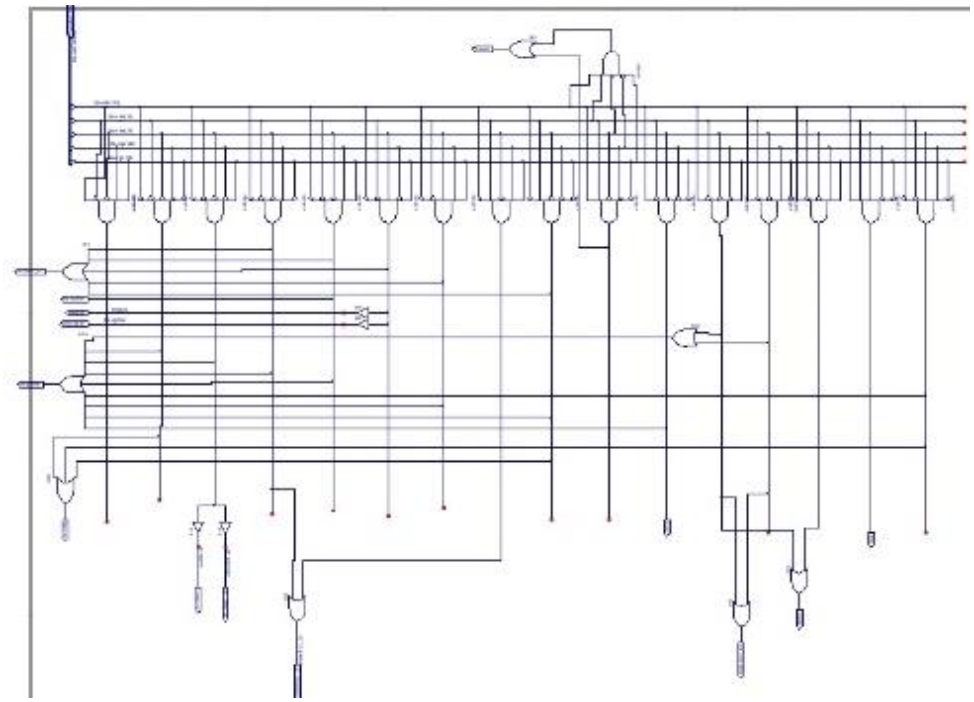


Post-Route

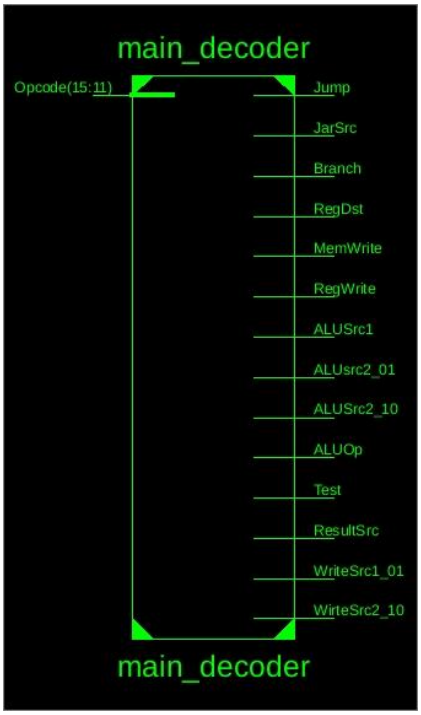


- **Instruction Decoder**

Schematic



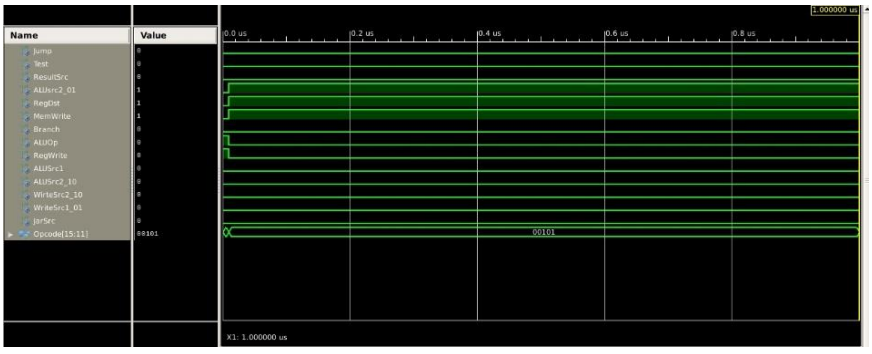
RTL



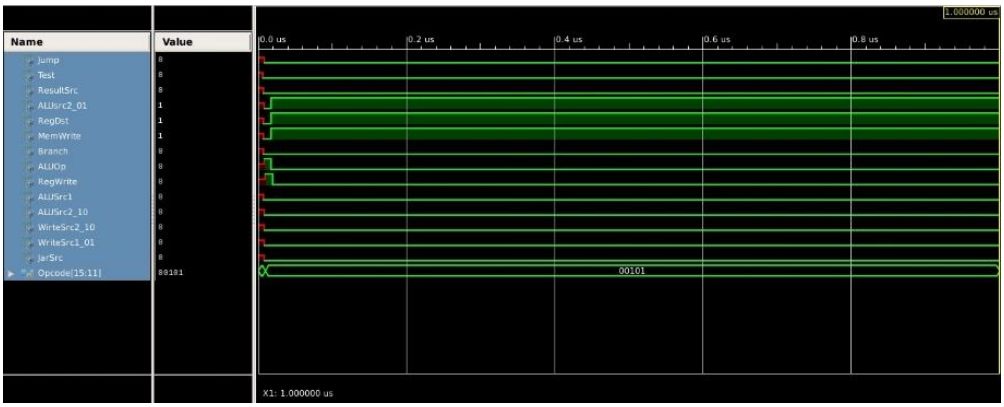
Testbench

```verilog
// Verilog test fixture cr

`timescale 1ns / 1ps

module main_decoder_tb();

// Inputs
   reg [15:11] Opcode;

// Output
   wire Jump;
   wire Test;
   wire ResultSrc;
   wire ALUsrc2_01;
   wire RegDst;
   wire MemWrite;
   wire Branch;
   wire ALUOp;
   wire RegWrite;
   wire ALUSrc1;
   wire ALUSrc2_10;
   wire WirteSrc2_10;
   wire WriteSrc1_01;
   wire JarSrc;

// Instantiate the UUT
   main_decoder UUT (
      .Opcode(Opcode),
      .Jump(Jump),
      .Test(Test),
      .ResultSrc(ResultSrc),
      .ALUsrc2_01(ALUsrc2_01),
      .RegDst(RegDst),
      .MemWrite(MemWrite),
      .Branch(Branch),
      .ALUOp(ALUOp),
      .RegWrite(RegWrite),
      .ALUSrc1(ALUSrc1),
      .ALUSrc2_10(ALUSrc2_10),
      .WirteSrc2_10(WirteSrc2_10),
      .WriteSrc1_01(WriteSrc1_01),
      .JarSrc(JarSrc)
   );
// Initialize Inputs
   initial begin
      // Test with different opcodes
      Opcode = 5'b00000; // You can change the opcode for testing
      #10; // Wait for some time
      Opcode = 5'b00101; // Another opcode
      #10; // Wait for some time
      // Add more tests with different opcodes as needed
   end
endmodule
```

Behavioral

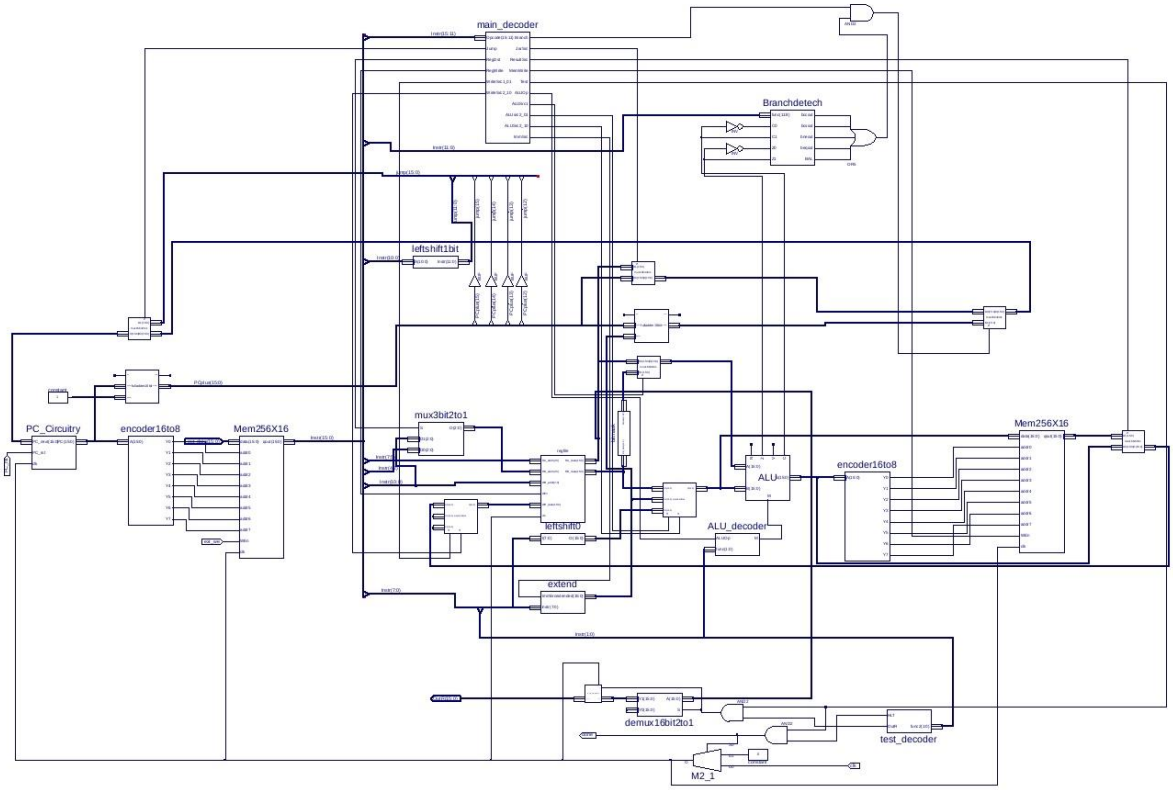


Post-Route



- **Complete Computer**

Schematic



Testbench

```
`timescale 1ns / 1ps

module RISCV16bit_tb();

// Inputs
    parameter clk_period = 20;
    parameter delay_factor = 2;
    reg clk;
    reg [15:0] ext_data;
    reg PC_rst;
    reg ext_we;

// Output
    wire [15:0] OutR;
    wire done;
// Instantiate the UUT
    RISCV16bit UUT (
        .clk(clk),
        .ext_data(ext_data),
        .ext_we(ext_we),
        .PC_rst(PC_rst),
        .OutR(OutR),
        .done(done)
    );

// Clock generation
    always begin
       #(clk_period / 2) clk <= 1'b0;
       #(clk_period / 2) clk <= 1'b1;
    end
// Initialize Inputs
    initial begin
    // Find max&min
        PC_rst <= 1'b0;
        repeat (9) @(posedge clk)
                   #(clk_period / delay_factor) PC_rst <= 1'b0;
        PC_rst <= 1'b1;

        write_mem1(16 'b0000_1000_0111_1111 ) ; // LHI R0,#127
        write_mem1(16 'b0000_1001_0000_0000 ) ; // LHI R1,#0

        write_mem1(16 'b0001_1010_1010_0000 ) ; // LDR R2,R5,#0
        loop1:
        write_mem1(16 'b0001_1011_1010_0001 ) ; // LDR R3,R5,#1

        write_mem1(16 'b0000_0100_0100_1110 ) ; // SUB R4,R2,R3
        write_mem1(16 'b1100_0010_0000_0010 ) ; // BCS next_min
        write_mem1(16 'b0101_1000_0110_0000 ) ; // MOV R0,R3
        next_min:
        write_mem1(16 'b0000_0100_0010_0010 ) ; // SUB R4,R1,R3
        write_mem1(16 'b1100_0011_0000_0010 ) ; // BCC next_max
        write_mem1(16 'b0101_1001_0110_0000 ) ; // MOV R1,R3
```

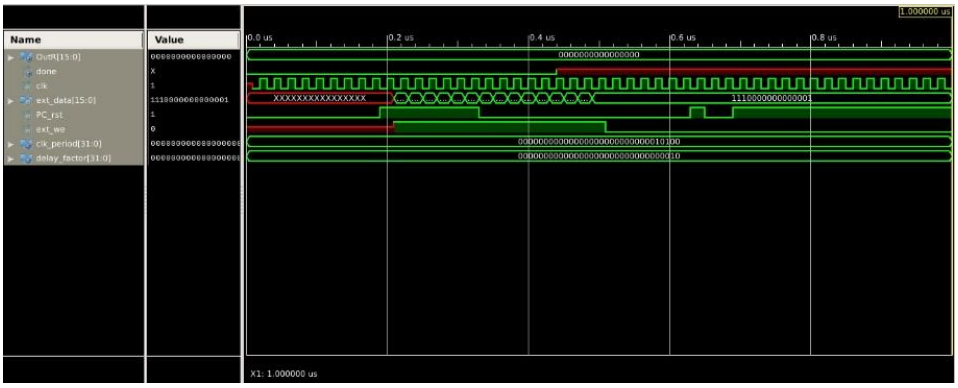
```
    next_max:
    write_mem1(16 'b0011_1101_1010_0001 ) ; // ADDI R5,R5,#1
    write_mem1(16 'b1100_0001_1111_0101 ) ; // BNE loop2

    done1:
    write_mem1(16 'b1110_0000_0000_0000 ) ; // OutR R0
    $display("Minimum Value (R0): %h", OutR);
    write_mem1(16 'b1110_0000_0010_0000 ) ; // OutR R1

    $display("Maximum Value (R1): %h", OutR);
    write_mem1(16 'b1110_0000_0000_0001 ) ; // HLT
    @(posedge clk) #(clk_period / delay_factor) ext_we = 1'b0;
    //read data from the dual-port memory

    repeat (9) @(posedge clk)
               #(clk_period / delay_factor) PC_rst = 1'b0;
    PC_rst = 1'b1;
    wait (done);
end
initial begin
// ADD 2 numbers in mem and store result in another location
    PC_rst <= 1'b0;
    repeat (9) @(posedge clk)
               #(clk_period / delay_factor) PC_rst <= 1'b0;
    PC_rst <= 1'b1;
    write_mem2(16 'b0001_1000_1010_0000 ) ; // LDR R0,R5,#0
    write_mem2(16 'b0001_1001_1100_0000 ) ; // LDR R1,R6,#0
    write_mem2(16 'b0000_0010_0000_0100 ) ; // ADD R2,R0,R1
    write_mem2(16 'b0010_1010_1110_0000 ) ; // STR R2,R7,#0
    write_mem2(16 'b1110_0000_0000_0001 ) ; // HLT
    @(posedge clk) #(clk_period / delay_factor) ext_we = 1'b0;
    //read data from the dual-port memory

    repeat (9) @(posedge clk)
               #(clk_period / delay_factor) PC_rst = 1'b0;
    PC_rst = 1'b1;
    wait (done);
end
initial begin
// ADD 10 numbers in consecutive in mem.
    PC_rst <= 1'b0;
    repeat (9) @(posedge clk)
               #(clk_period / delay_factor) PC_rst <= 1'b0;
    PC_rst <= 1'b1;
    write_mem3(16 'b0000_1000_0000_0000 ) ; // LHI R0,#0
    write_mem3(16 'b0000_1001_0000_0000 ) ; // LHI R1,#0
    write_mem3(16 'b0000_1011_0000_1010 ) ; // LHI R3,#10
    loop2:
    write_mem3(16 'b0001_1010_1010_0000 ) ; // LDR R2,R5,#0
    write_mem3(16 'b0000_0000_0000_1000 ) ; // ADD R0,R0,R2
    write_mem3(16 'b0011_1001_0010_0001 ) ; // ADDI R1,R1,#1
    write_mem3(16 'b0011_1101_1010_0001 ) ; // ADDI R5,R5,#1
    write_mem3(16 'b0011_0000_0010_1101 ) ; // CMP R1,R3
    write_mem3(16 'b1100_0001_1111_1010 ) ; // BNE loop2
    write_mem3(16 'b1110_0000_0000_0000 ) ; // OutR R0
    write_mem3(16 'b1110_0000_0000_0001 ) ; // HLT
```

```
    @(posedge clk) #(clk_period / delay_factor) ext_we = 1'b0;
    //read data from the dual-port memory

    repeat (9) @(posedge clk)
               #(clk_period / delay_factor) PC_rst = 1'b0;
    PC_rst = 1'b1;
    wait (done);
end
initial begin
// Mov a mem block of N words from one place to another.
    PC_rst <= 1'b0;
    repeat (9) @(posedge clk)
               #(clk_period / delay_factor) PC_rst <= 1'b0;
    PC_rst <= 1'b1;
    write_mem4(16 'b0000_1000_0000_0000 ) ; // LHI R0,#0
    write_mem4(16 'b0000_1001_0000_0010 ) ; // LHI R1,#2
    write_mem4(16 'b0000_1010_0010_0000 ) ; // LHI R2,#32
    write_mem4(16 'b0000_1011_0100_0000 ) ; // LHI R3,#64
    move_loop:
    write_mem4(16 'b0001_1100_0100_0000 ) ; // LDR R4,R2,#0
    write_mem4(16 'b0010_1100_0110_0000 ) ; // STR R4,R3,#0
    write_mem4(16 'b0011_1000_0000_0001 ) ; // ADDI R0,R0,#1
    write_mem4(16 'b0011_1010_0010_0010 ) ; // ADDI R2,R2,#2
    write_mem4(16 'b0011_1011_0011_0010 ) ; // ADDI R3,R3,#2
    write_mem4(16 'b0011_0000_0010_0001 ) ; // CMP R0,R1
    write_mem4(16 'b1100_0001_1000_0110 ) ; // BNE move_loop
    write_mem4(16 'b1110_0000_0000_0001 ) ; // HLT
    @(posedge clk) #(clk_period / delay_factor) ext_we = 1'b0;
```

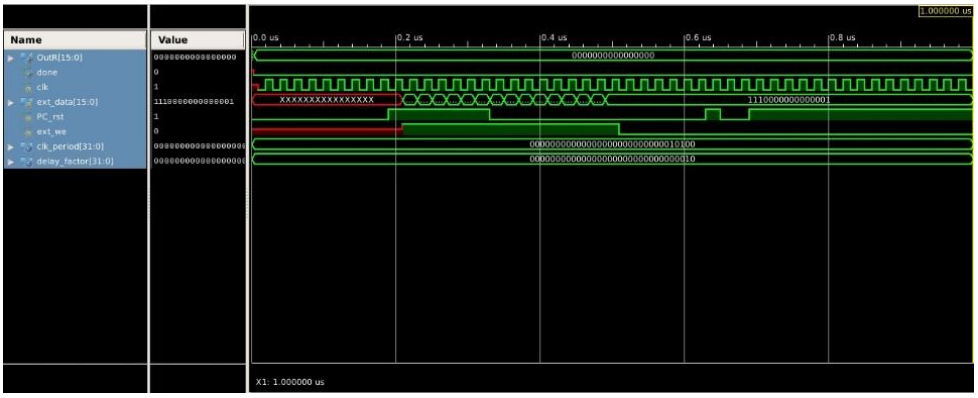
```
        repeat (9) @(posedge clk)
                   #(clk_period / delay_factor) PC_rst = 1'b0;
        PC_rst = 1'b1;
        wait (done);
    end
    task write_mem1;
    input [15:0] data;
    begin
       @(posedge clk) #(clk_period/delay_factor) begin
           ext_we = 1'b1;
           ext_data = data;
        end
    end

    endtask

    task write_mem2;
    input [15:0] data;
    begin
       @(posedge clk) #(clk_period/delay_factor) begin
           ext_we = 1'b1;
           ext_data = data;
        end
    end

    endtask
```

```
   task write_mem3;
   input [15:0] data;
   begin
      @(posedge clk) #(clk_period/delay_factor) begin
          ext_we = 1'b1;
          ext_data = data;
       end
   end

   endtask
   task write_mem4;
   input [15:0] data;
   begin
      @(posedge clk) #(clk_period/delay_factor) begin
          ext_we = 1'b1;
          ext_data = data;
       end
   end

   endtask

   initial #1000000000 $finish;
   initial
      $monitor ($realtime, "ns %h %h %h %h %h %h \n", clk, PC_rst, ext_we, ext_data, OutR, done);
endmodule
```

Behavioral



Post-Route



3. Programs

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

```
write_mem1(16 'b0000_1000_0111_1111 ) ; // LHI R0,#127
write_mem1(16 'b0000_1001_0000_0000 ) ; // LHI R1,#0

write_mem1(16 'b0001_1010_1010_0000 ) ; // LDR R2,R5,#0
loop1:
write_mem1(16 'b0001_1011_1010_0001 ) ; // LDR R3,R5,#1

write_mem1(16 'b0000_0100_0100_1110 ) ; // SUB R4,R2,R3
write_mem1(16 'b1100_0010_0000_0010 ) ; // BCS next_min
write_mem1(16 'b0101_1000_0110_0000 ) ; // MOV R0,R3
next_min:
write_mem1(16 'b0000_0100_0010_0010 ) ; // SUB R4,R1,R3
write_mem1(16 'b1100_0011_0000_0010 ) ; // BCC next_max
write_mem1(16 'b0101_1001_0110_0000 ) ; // MOV R1,R3
next_max:
write_mem1(16 'b0011_1101_1010_0001 ) ; // ADDI R5,R5,#1
write_mem1(16 'b1100_0001_1111_0101 ) ; // BNE loop2

done1:
write_mem1(16 'b1110_0000_0000_0000 ) ; // OutR R0
$display("Minimum Value (R0): %h", OutR);
write_mem1(16 'b1110_0000_0010_0000 ) ; // OutR R1

$display("Maximum Value (R1): %h", OutR);
write_mem1(16 'b1110_0000_0000_0001 ) ; // HLT
```

```
    // Initialization
    LHI     R0, #32767; R0 = 32767
    LHI     R1, #0; R1 = 0

    // Load the first number from memory into R2
    LDR     R2, [R5, #0]; R2 = Mem[R5]

loop:// Load the next number from memory into R3
    LDR     R3, [R5, #1]; R3 = Mem[R5 + 1]

    // Compare R2 and R3 to find the minimum
    SUB     R4, R2, R3; R4 = R2 - R3
    BCS     next_min; Jump to next_min if R4 is not positive
    MOV     R0, R3; R0 = R3

next_min:// Compare R3 and R1 to find the maximum
    SUB     R4, R1, R3; R4 = R1 - R3
    BCC     next_max; Jump to next_max if R4 is not positive
    MOV     R1, R3; R1 = R3

next_max:
    ADDI R5, R5, #1 ; R5 = R5 + 1
    BNE loop;

done:// R0 and R1 now contain the minimum and maximum values
    // End of program
    OutR R0;
    OutR R1;
    HLT
```

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

memory location.

```
//Load the first number from memory into R0
    LDR     R0, [R5, #0];
//Load the second number from memory into R1
    LDR     R1, [R6, #0];
//Add the numbers
    ADD     R2, R0, R1; R2 = R0 + R1
//Store the result back into memory at address R7
    STR     R2, [R7, #0];
// End of program
    HLT;
```

```
write_mem2(16 'b0001_1000_1010_0000 ) ; // LDR R0,R5,#0
write_mem2(16 'b0001_1001_1100_0000 ) ; // LDR R1,R6,#0
write_mem2(16 'b0000_0010_0000_0100 ) ; // ADD R2,R0,R1
write_mem2(16 'b0010_1010_1110_0000 ) ; // STR R2,R7,#0
write_mem2(16 'b1110_0000_0000_0001 ) ; // HLT
```

3. Add ten numbers in consecutive memory locations.

```
write_mem3(16 'b0000_1000_0000_0000 ) ; // LHI R0,#0
write_mem3(16 'b0000_1001_0000_0000 ) ; // LHI R1,#0
write_mem3(16 'b0000_1011_0000_1010 ) ; // LHI R3,#10
loop2:
write_mem3(16 'b0001_1010_1010_0000 ) ; // LDR R2,R5,#0
write_mem3(16 'b0000_0000_0000_1000 ) ; // ADD R0,R0,R2
write_mem3(16 'b0011_1001_0010_0001 ) ; // ADDI R1,R1,#1
write_mem3(16 'b0011_1101_1010_0001 ) ; // ADDI R5,R5,#1
write_mem3(16 'b0011_0000_0010_1101 ) ; // CMP R1,R3
write_mem3(16 'b1100_0001_1111_1010 ) ; // BNE loop2
write_mem3(16 'b1110_0000_0000_0000 ) ; // OutR R0
write_mem3(16 'b1110_0000_0000_0001 ) ; // HLT
```

```
// Initialization
LHI R0, #0; Initialize R0 to store the sum
LHI R1, #0; Initialize R1 as a loop counter
LHI R3, #10; Set R3 to 10, indicating we want to add ten numbers

// Loop
loop:
    LDR R2, [R5, #0]; Load the first number into R2
    ADD R0, R0, R2; Add the number in R2 to the sum in R0
    ADDI R1, R1, #1; Increment the loop counter in R1
    ADDI R5, R5, #1; Move to the next memory location
    CMP R1, R3; Compare the counter to 10
    BNE loop; Continue the loop if the counter is not yet 10

// End of program
OutR R0; Display the sum
HLT; End the program
```

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

```
write_mem4(16 'b0000_1000_0000_0000 ) ; // LHI R0,#0
write_mem4(16 'b0000_1001_0000_0010 ) ; // LHI R1,#2
write_mem4(16 'b0000_1010_0010_0000 ) ; // LHI R2,#32
write_mem4(16 'b0000_1011_0100_0000 ) ; // LHI R3,#64
move_loop:
write_mem4(16 'b0001_1100_0100_0000 ) ; // LDR R4,R2,#0
write_mem4(16 'b0010_1100_0110_0000 ) ; // STR R4,R3,#0
write_mem4(16 'b0011_1000_0000_0001 ) ; // ADDI R0,R0,#1
write_mem4(16 'b0011_1010_0010_0010 ) ; // ADDI R2,R2,#2
write_mem4(16 'b0011_1011_0011_0010 ) ; // ADDI R3,R3,#2
write_mem4(16 'b0011_0000_0010_0001 ) ; // CMP R0,R1
write_mem4(16 'b1100_0001_1000_0110 ) ; // BNE move_loop
write_mem4(16 'b1110_0000_0000_0001 ) ; // HLT
```

```
// Initialization
    LHI R0, #0;
    LHI R1, #2;
    LHI R2, #32;
    LHI R3, #64;
// Loop
move_loop:
    LDR R4, [R2, #0];
    STR R4, [R3, #0];
    ADDI R0, R0, #1;
    ADDI R2, R2, #2;
    ADDI R3, R3, #2;
    CMP RO, R1;
    BNE move_loop;

// End of program
    HLT;
```

4. Discussion

1. 以前只有在計算機組織課程只有修過 MIPS 指令集，在轉換到 RISCV 上的過程中遇到了很多的阻礙，但是透過老師給的那兩本 text book，這部分得到了解決，有一些指令集可以去參考 ARM 與 X86 會有類似的解方。
2. 指令部分一般的算術運算與記憶體搬運沒有太大問題，主要問題落在 LHI 與 LLI，過去只有遇過 LUI 與 ORI 指令，沒有遇過保留一部分暫存器的問題，後來思考許久使用 bitmask 的方式來保留需要的部分。

3. 遇到 Opcode 不一致的問題，像是 Branch 是 8bit，在這邊思考很久，決定透過固定分析前五個 bit(Opcode)，如果是 branch 指令在去偵測後面 3bit 是甚麼
4. 最後是遇到一些在設計上的小問題，像是怎麼串接 RAM 或是指令解碼器的製作，透過課本以及網路上的資料來慢慢解決問題，也了解到硬體好玩的地方，硬體很簡單但是變化無窮，最後在這部分收穫很大。

5.Conclusion

Hardware Cost:

RISCV16bit Project Status (10/29/2023 - 16:31:19)			
Project File:	RISCV16bit.xise	**Parser Errors:**	No Errors
Module Name:	RISCV16bit	**Implementation State:**	Placed and Routed
Target Device:	xc3s100e-4cp132	**• Errors:**	No Errors
Product Version:	ISE 14.7	**• Warnings:**	51 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 [-]				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	16	1,920	1%	
Number of 4 input LUTs	108	1,920	5%	
Number of occupied Slices	84	960	8%	
Number of Slices containing only related logic	84	84	100%	
Number of Slices containing unrelated logic	0	84	0%	
Total Number of 4 input LUTs	108	1,920	5%	
Number of bonded IOBs	18	83	21%	
Number of BUFGMUXs	1	24	4%	
Number of RPM macros	32			
Average Fanout of Non-Clock Nets	1.81			

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	Sun Oct 29 11:49:17 2023	0	33 Warnings (0 new)	0
Translation Report	Current	Sun Oct 29 11:49:26 2023	0	0	0
Map Report	Current	Sun Oct 29 11:49:33 2023	0	18 Warnings (0 new)	2 Infos (0 new)
Place and Route Report	Current	Sun Oct 29 11:49:41 2023	0	0	2 Infos (0 new)
Power Report					