CSE112

Computer Organization and Architecture

Submitted to

Dr. Tamer Mostafa

Dr. Karim Emara

Eng. Yasmin Shaaban

Major Task Report

Team 8 Members

Name	ID
Mohamed Hany Mohamed Fadel	21P0299
Mazen Ahmed Galal	21P0103
Hams Assem Ahmed	20P3713
Bassel Ashraf	22P0122
Ezzeldine Ahmed Ali	21P0078

Table of Contents

Top module 7

1.Code 7

2.RTL 8

3. Explanation 9

MIPS 10

1.Code 10

2.RTL 11

3.Explaination 12

IMEM 13

1.Code 13

2.RTL 15

3.Explaination 16

DMEM 17

1.Code 17

2.RTL 18

3.Explaination 18

Controller 19

1.Code 19

2.RTL 20

3. Explaination 21

Main Decoder 22

1.Code 22

2.RTL 24

3.Explaination 25

ALU Decoder 26

1.Code 26

2.RTL 27
3.Explaination 28
Data Path 29
1.Code 29
2.RTL 33
3.Explaination 33
Register File 34
1.Code 34
2.RTL 40
3.Explaination 43
ALU 44
1.Code 44
2.RTL 45
3.Explaination 48
Mux 2*1 49
1.Code 49
Mux 32*1 50
1.Code 50
2.RTL 53
Flopr 54
1.Code 54
2.RTL 55
Decoder 56
1.Code 56
2.RTL 58
Adder 59
1.Code 59

2.RTL 60
Sign Extender 61
1.Code 61
2.RTL 61
Shift left 2 62
1.Code 62
2.RTL 62
Test Files 63
Main Module Test 63
1.Code 63
2.Test Results 65

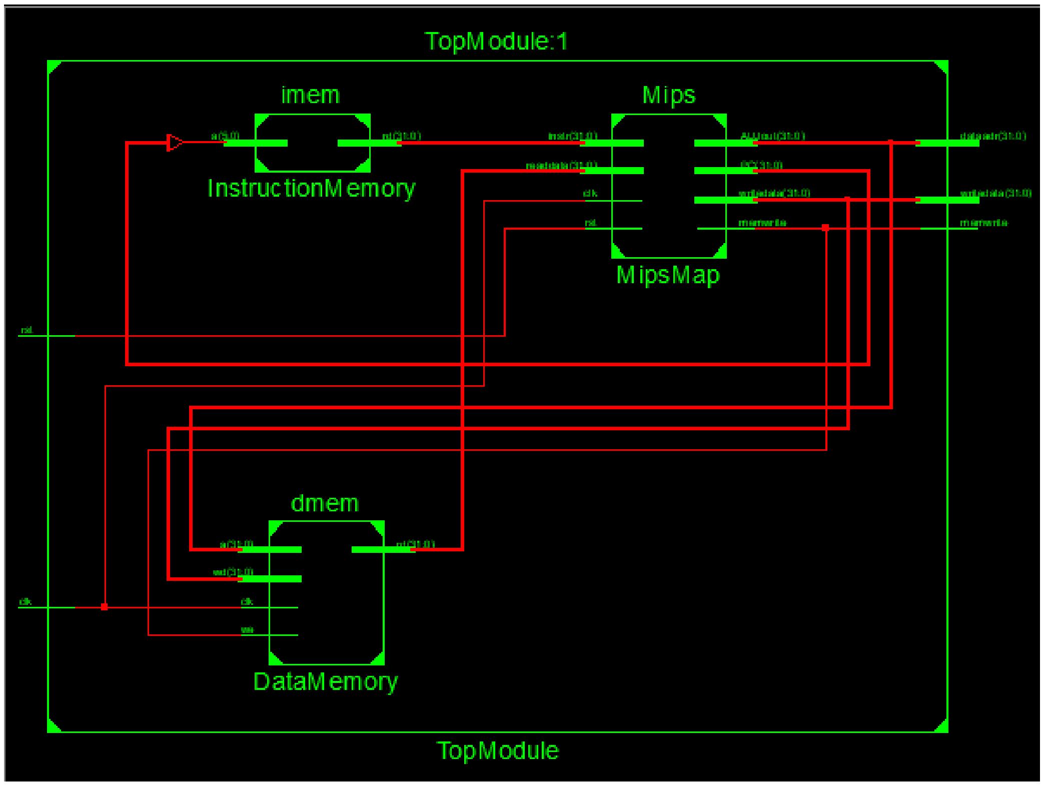
Top module

1.Code

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use work.Mips_Package.all;

use work.Dmem_Package.all;

use work.Imem_Package.all;


entity TopModule is

   Port ( clk : in  STD_LOGIC;

          rst : in  STD_LOGIC;

          writedata : out  STD_LOGIC_VECTOR (31 downto 0);

          dataadr : out  STD_LOGIC_VECTOR (31 downto 0);

          memwrite : out  STD_LOGIC);

end TopModule;


architecture Behavioral of TopModule is


signal memwritet: STD_LOGIC;

signal PC,instr,readdata,dataadrt,writedatat: STD_LOGIC_VECTOR (31 downto 0);
```

```
begin

InstructionMemory: imem port map(pc(7 downto 2),instr);

MipsMap: Mips port
map(clk,rst,pc,instr,memwritet,dataadrt,writedatat,readdata);

DataMemory: dmem port map(clk,memwritet,dataadrt,writedatat,readdata);

writedata<= writedatat;


memwrite<=memwritet;


dataadr<=dataadrt;

end Behavioral;
```

2.RTL



3. Explanation

Entity called Top Module which connect between mips, dmem and imem entities in order to test mips processor The imem module is instantiated with inputs pc(7 downto 2) and instr, The mips module is instantiated with inputs clk,reset, pc, and instr, and outputs memwritet, dataadrt, writedatat, and readdata and The dmem module is instantiated with inputs clk, memwritet, dataadrt, andwritedatat, and outputs readdata.

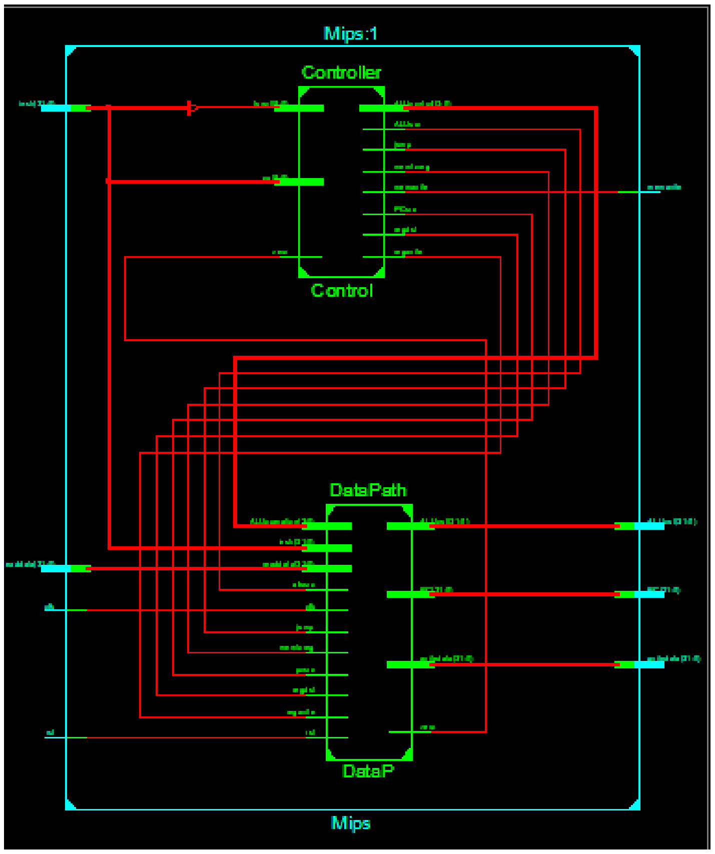
MIPS

1.Code

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use work.Controller_Package.all;

use work.DataPath_Package.all;


entity Mips is

   Port ( clk : in  STD_LOGIC;

          rst : in  STD_LOGIC;

          PC : out  STD_LOGIC_VECTOR (31 downto 0);

          instr : in  STD_LOGIC_VECTOR (31 downto 0);

          memwrite : out  STD_LOGIC;

          ALUout : out  STD_LOGIC_VECTOR (31 downto 0);

          writedata : out  STD_LOGIC_VECTOR (31 downto 0);

          readdata : in  STD_LOGIC_VECTOR (31 downto 0));

end Mips;


architecture Behavioral of Mips is

signal memtoreg, alusrc, regdst, regwrite, jump, pcsrc: STD_LOGIC;

signal zero: STD_LOGIC;

signal alucontrol: STD_LOGIC_VECTOR (3 downto 0);

signal pctemp,writedatatemp,aluouttemp: STD_LOGIC_VECTOR (31 downto 0);
```

```
begin

Control: controller port map(instr(31 downto 26),instr(5 downto
0),zero,memtoreg,memwrite,pcsrc,alusrc,regdst,regwrite,jump,alucontrol);


DataP : datapath port map(clk,rst,readdata,instr,memtoreg, pcsrc,alusrc,regwrite,
regdst,jump,alucontrol,zero,pctemp,writedatatemp,aluouttemp);


PC  <=pctemp;

writedata <= writedatatemp;

ALUout <= aluouttemp;


end Behavioral;
```

2.RTL



3.Explaination

The code provided is for an entity called "mips" which is the main module that connects the controller and datapath of a MIPS processor. The module has inputs for clock, reset, and instruction, and outputs for program counter, memory write, ALU output, write data, and read data. The control signals required for the processor are passed from the controller to the datapath usingsignals such as memtoreg, alusrc, regdst, regwrite, jump, pcsrc, zero, and alucontrol. This entity can be used to test the MIPS processor by instantiating itwith imem and dmem entities, where imem is connected to pc(7 downto 2) andinstr inputs, and dmem is connected to clk, memwrite, dataadrt, and writedatat inputs, and readdata output.

IMEM

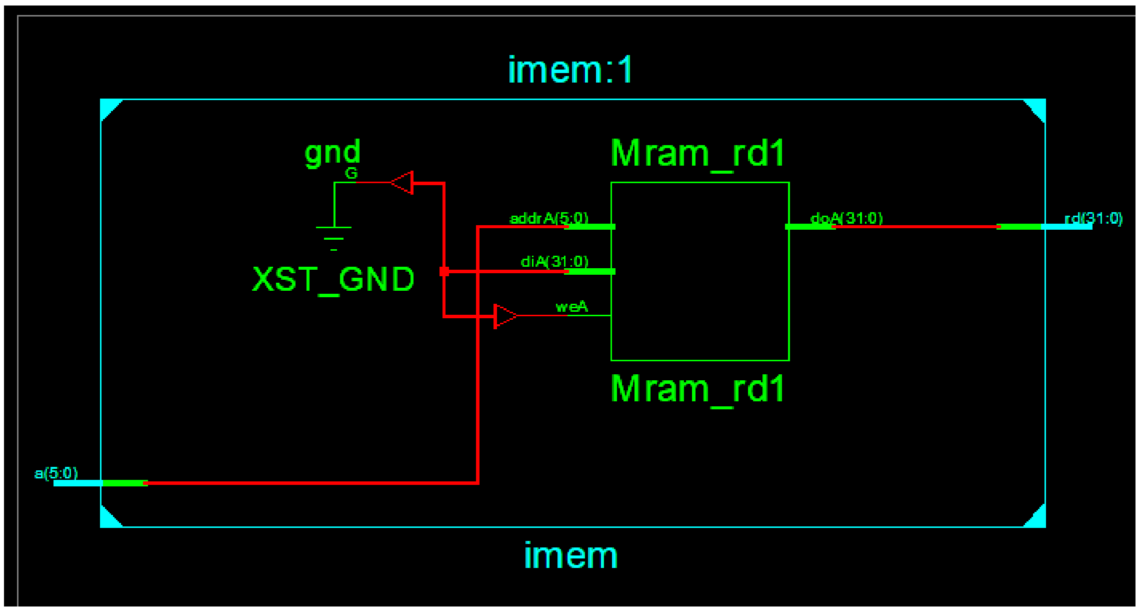
1.Code

```
library IEEE;

use IEEE.STD_LOGIC_1164.all;

use IEEE.STD_LOGIC_1164.all;

use IEEE.STD_LOGIC_SIGNED.all;

use IEEE.STD_LOGIC_ARITH.all;

use ieee.numeric_std.all;

use IEEE.std_logic_textio.all;

library STD;

use STD.textio.all;

entity imem is -- instruction memory

port(a: in STD_LOGIC_VECTOR(5 downto 0);

rd: out STD_LOGIC_VECTOR(31 downto 0));

end;


architecture behave of imem is

begin

process is

file mem_file: TEXT;

variable L: line;

variable ch: character;

variable i, index, result: integer;

type ramtype is array (63 downto 0) of STD_LOGIC_VECTOR(31 downto 0);

variable mem: ramtype;
```

```
begin
-- initialize memory from file
for i in 0 to 63 loop -- set all contents low
mem(i) := (others => '0');
end loop;
index := 0;
FILE_OPEN (mem_file, "E:/memfile.dat", READ_MODE);
while not endfile(mem_file) loop
readline(mem_file, L);
result := 0;
for i in 1 to 8 loop
read (L, ch);
if '0' <= ch and ch <= '9' then
result := character'pos(ch) - character'pos('0');
elsif 'a' <= ch and ch <= 'f' then
result := character'pos(ch) - character'pos('a')+10;
else report "Format error on line" & integer'
image(index) severity error;
end if;
mem(index)(35-i*4 downto 32-i*4):= std_logic_vector(to_unsigned(result,4));
end loop;
index := index + 1;
end loop;
-- read memory
```

```
for i in 1 to 1000 loop
rd <= mem(CONV_INTEGER(a));
wait on a;
end loop;
end process;


end;
```

2.RTL



3.Explaination

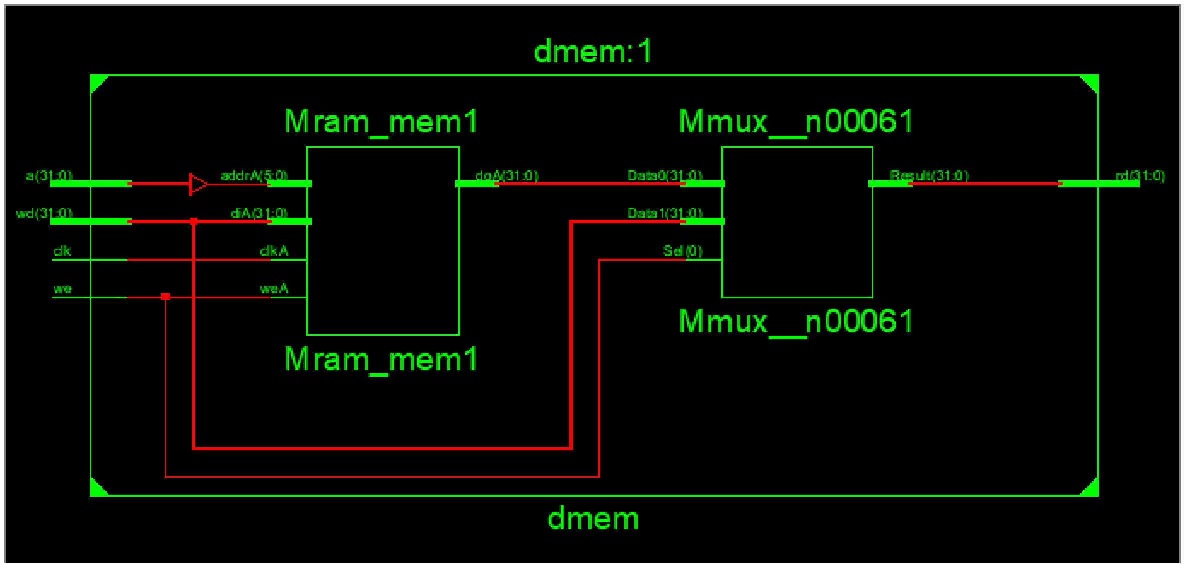
The provided code is for an entity called "imem" which represents the instruction memory of a MIPS processor. It has one input port "a" for the addressand one output port "rd" for the instruction data. The code reads the instructionmemory data from a file named "memfile.dat" and stores it in a RAM-type array"mem". The entity "imem" can be used in a larger design, such as in an entity called "Main_module" which connects between "mips", "dmem", and "imem" entities for testing the MIPS processor. The "imem" module is instantiated withinputs "pc(7 downto 2)" as the difference between instructionsis 1 so we divideby 4 by skipping the first two bits as it acts like a shift right and "instr", while the"mips" module is instantiated with inputs "clk", "reset", "pc", and "instr", and outputs "memwritet", "dataadrt", "writedatat", and "readdata". Finally, the "dmem" module is instantiated with inputs "clk", "memwritet", "dataadrt", and "writedatat", and outputs "readdata".

DMEM

1.Code

```
library IEEE;

use IEEE.STD_LOGIC_1164.all;

use IEEE.STD_LOGIC_SIGNED.all;

use IEEE.STD_LOGIC_ARITH.all;

use ieee.numeric_std.all;


entity dmem is -- data memory

port(clk, we: in STD_LOGIC;

a, wd: in STD_LOGIC_VECTOR (31 downto 0);

rd: out STD_LOGIC_VECTOR (31 downto 0));

end;

architecture behave of dmem is

begin

process  is

type ramtype is array (63 downto 0) of STD_LOGIC_VECTOR(31 downto 0);

variable mem: ramtype;

begin

-- read or write memory

for i in 1 to 1000 loop

if rising_edge(clk) then

if (we='1') then

mem (CONV_INTEGER('0'&a(7 downto 2))):= wd;

end if;
end if;

rd <= mem (CONV_INTEGER('0'&a (7 downto 2)));

wait on clk, a;

end loop;

end process;

end;
```

2.RTL



3.Explaination

The given code describes an entity for a data memory block that is implementedusing an array of 64 words, each 32 bits wide. The memory read and write operations are synchronized using a “clk” input, with write operations controlledby the we input and data and address inputs wd and a, respectively. The outputdata read from the memory is provided through the “rd” output port. The memory operation is implemented in a process block, which is triggered by a rising edge of the “clk” signal. Overall, this entity describes a simple synchronous RAM block implementation for reading and writing 32-bit data values.

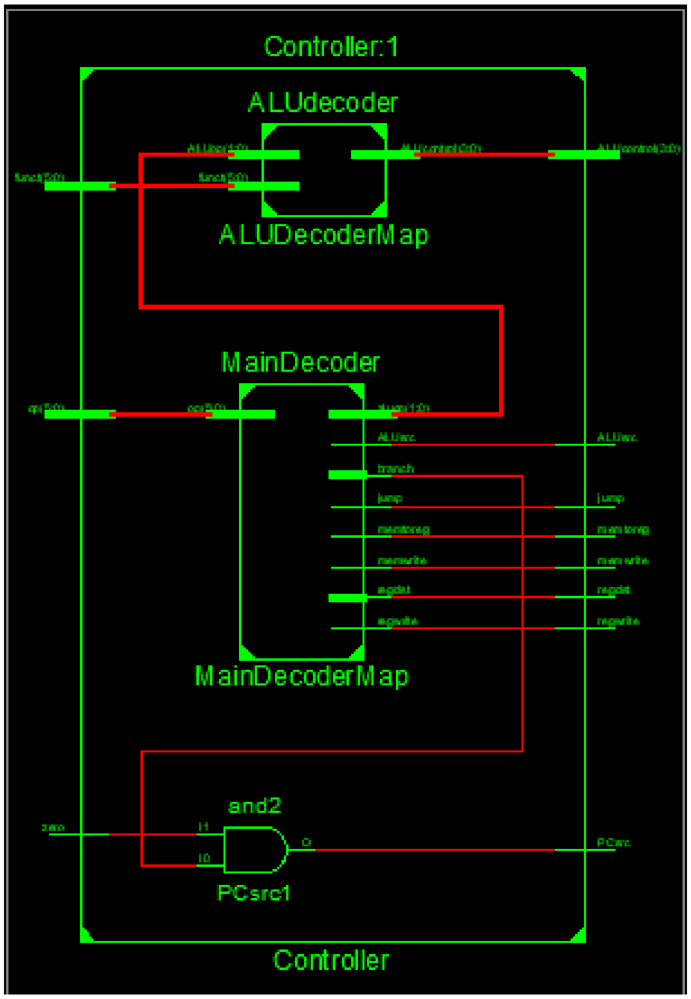
Controller

1.Code

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use work.ALUdecoder_Package.ALL;

use work.MainDecoder_Package.ALL;


entity Controller is

   Port ( op : in  STD_LOGIC_VECTOR (5 downto 0);

        funct : in  STD_LOGIC_VECTOR (5 downto 0);

        zero : in  STD_LOGIC;

        memtoreg : out  STD_LOGIC;

        memwrite : out  STD_LOGIC;

        PCsrc : out  STD_LOGIC;

        ALUsrc : out  STD_LOGIC;

        regdst : out  STD_LOGIC;

        regwrite : out  STD_LOGIC;

        jump : out  STD_LOGIC;

        ALUcontrol : out  STD_LOGIC_VECTOR (3 downto 0));

end Controller;


architecture Behavioral of Controller is


Signal ALUop :  STD_LOGIC_VECTOR (1 downto 0);

Signal branch : STD_LOGIC;
```

```
begin

MainDecoderMap : MainDecoder port map
(op,memtoreg,memwrite,branch,ALUsrc,regdst,regwrite,jump,ALUop);

ALUDecoderMap : ALUdecoder port map (funct, ALUop, ALUcontrol);

pcsrc <= branch and zero;

end Behavioral;
```

2.RTL



3. Explaination

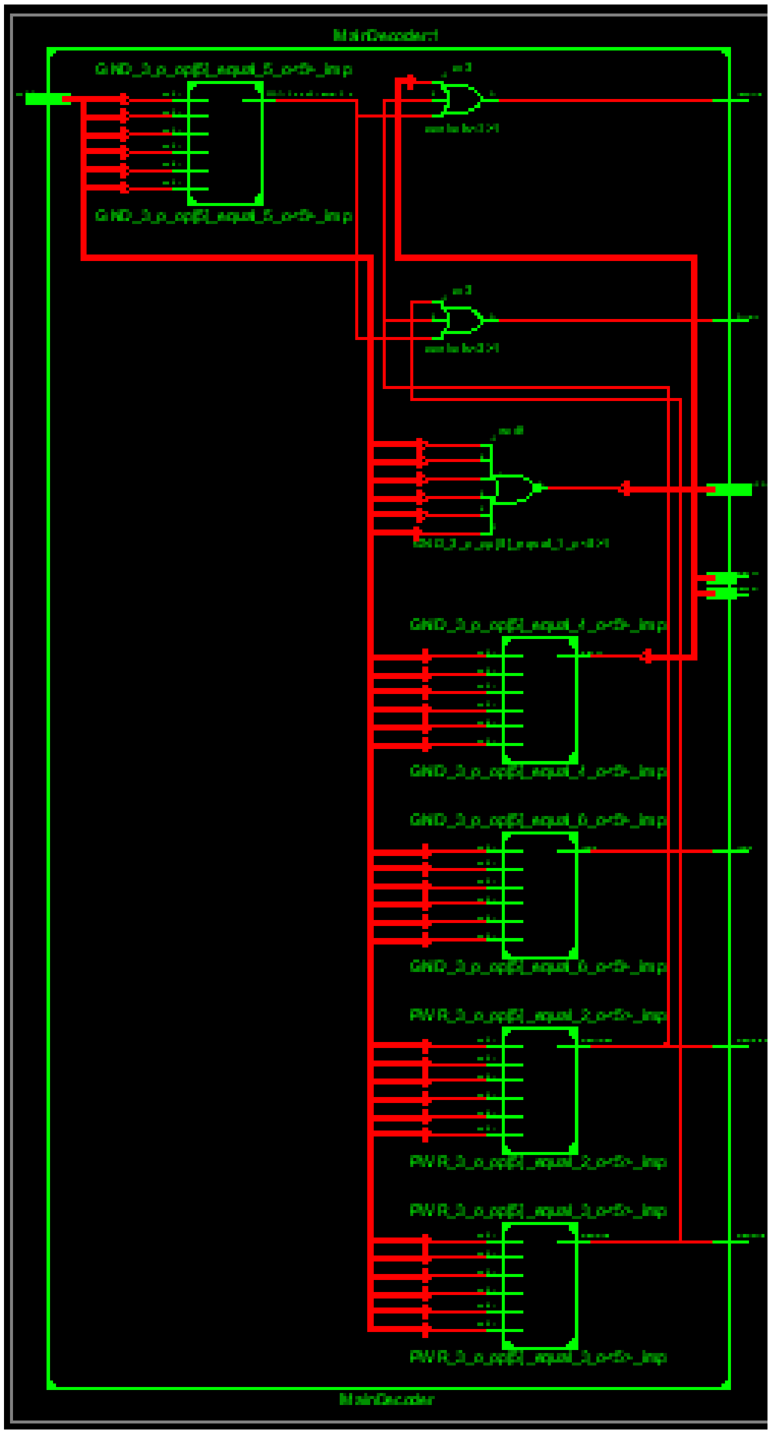
The provided code is an implementation of a behavioral model for a controller entity in a digital circuit. The controller entity has several inputs and outputs, including op, funct, zero, memtoreg, memwrite, pcsrc, alusrc, regdst, regwrite, jump, and alucontrol. The architecture of the controller entity includes a main decoder and an ALU decoder, which decode the input signals and set the output signals. Additionally, there are signals for aluop and branch, which are used in the decoding process. The pcsrc is the branch signal **AND** Zero.

Main Decoder

1.Code

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


entity MainDecoder is

   Port ( op : in  STD_LOGIC_VECTOR (5 downto 0);

          memtoreg : out  STD_LOGIC;

          memwrite : out  STD_LOGIC;

          branch : out  STD_LOGIC;

          ALUsrc : out  STD_LOGIC;

          regdst : out  STD_LOGIC;

          regwrite : out  STD_LOGIC;

          jump : out  STD_LOGIC;

          aluop : out  STD_LOGIC_VECTOR (1 downto 0));

end MainDecoder;


architecture Behavioral of MainDecoder is

signal controls: STD_LOGIC_VECTOR (8 downto 0);

begin

process (op)

begin

case op is

when "000000" => controls <="110000010"; --R type

when "100011" => controls <="101001000"; -- lw
when "101011" => controls <= "001010000"; -- sw

when "000100" => controls <= "000100001"; -- beg

when "001000" => controls <= "101000000"; -- addi

when "000010" => controls <= "000000100"; -- j

when others => controls <= "---------"; --illegal op

end case;

end process;

(regwrite, regdst, alusrc, branch, memwrite, memtoreg, jump, aluop (1), aluop (0))
<= controls;



end Behavioral;
```

2.RTL


MainDecoder:1
GND_3_o_op[5]_equal_5_o<5>_imp
GND_3_o_op[5]_equal_5_o<5>_imp
GND_3_o_op[5]_equal_4_o<5>_imp
GND_3_o_op[5]_equal_4_o<5>_imp
GND_3_o_op[5]_equal_6_o<5>_imp
GND_3_o_op[5]_equal_6_o<5>_imp
PWR_3_o_op[5]_equal_2_o<5>_imp
PWR_3_o_op[5]_equal_2_o<5>_imp
PWR_3_o_op[5]_equal_3_o<5>_imp
PWR_3_o_op[5]_equal_3_o<5>_imp
MainDecoder

3.Explaination

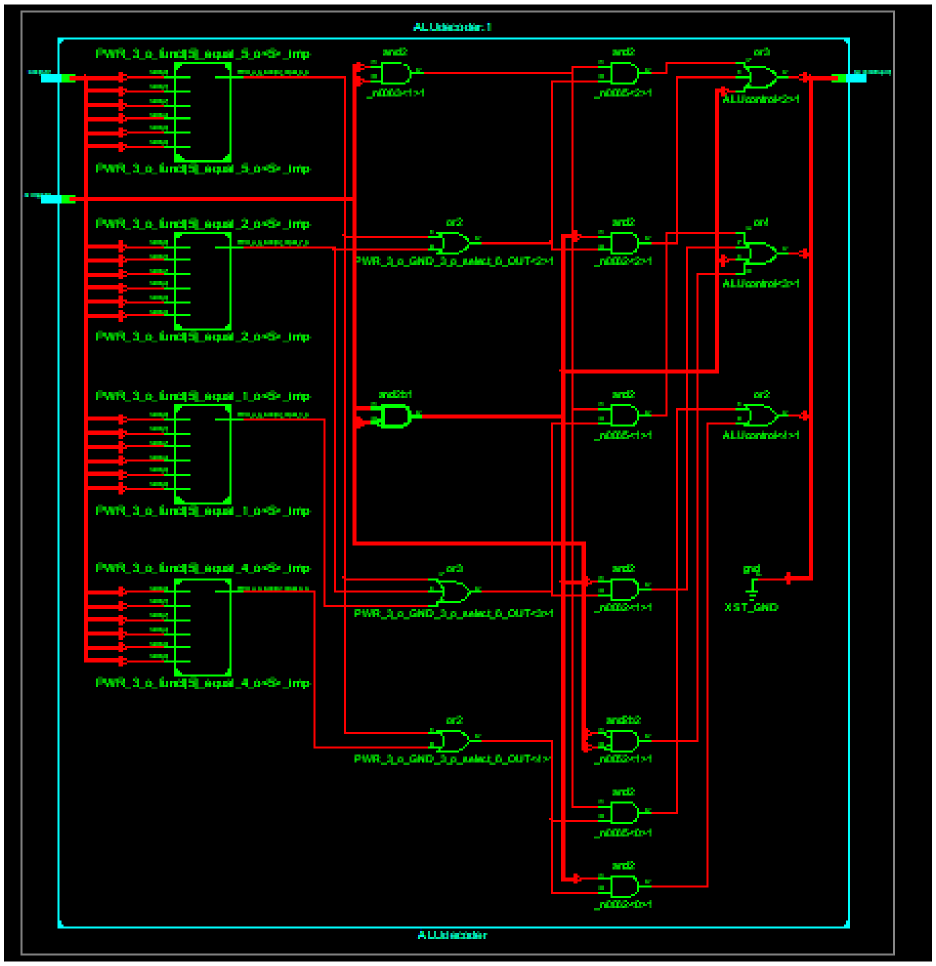
An entity called "maindec" which takes a 6-bit input signal "op" and generates 9control signals for a MIPS processor. The control signals include "memtoreg", "memwrite", "branch", "alusrc", "regdst", "regwrite", "jump", and "aluop" (a 2-bit signal). The entity maps the opcode of the instruction to the corresponding control signals using a case statement. The code can be used in conjunction withthe imem, mips, and dmem modules to test the MIPS processor. The imem module is instantiated with inputs "pc(7 downto 2)" and "instr", the mipsmodule is instantiated with inputs "clk", "reset", "pc", and "instr", and outputs "memwritet", "dataadrt", "writedatat", and "readdata", and the dmem moduleis instantiated with inputs "clk", "memwritet", "dataadrt", and "writedatat", andoutputs "readdata".

ALU Decoder

1.Code

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


entity ALUdecoder is

   Port ( funct : in  STD_LOGIC_VECTOR (5 downto 0);

        ALUop : in  STD_LOGIC_VECTOR (1 downto 0);

        ALUcontrol : out  STD_LOGIC_VECTOR (3 downto 0));

end ALUdecoder;


architecture Behavioral of ALUdecoder is


begin


process(ALUop,funct)

begin

        case ALUop is

        when "00" => ALUcontrol <= "0010"; -- Add for lw/sw or addi

        when "01" => ALUcontrol <= "0110"; -- Sub for beq

        when others =>

                case funct is

                when "100000" => ALUcontrol <= "0010";  -- Add

                when "100010" => ALUcontrol <= "0110";  --Sub
                when "100100" => ALUcontrol <= "0000";  --And
                when "100101" => ALUcontrol <= "0001";  --OT
                when "101010" => ALUcontrol <= "0111";  --slt
                when others => ALUcontrol <= "----"; --unknown
                end case;
        end case;
end process;


end Behavioral;
```

2.RTL



ALUdecoder:1
PWR_3_o_funct[5]_equal_5_o<5>_imp
PWR_3_o_funct[5]_equal_2_o<5>_imp
PWR_3_o_funct[5]_equal_1_o<5>_imp
PWR_3_o_funct[5]_equal_4_o<5>_imp
and2
or2
or3
and2b1
and3b2
ALUcontrol<2>1
ALUcontrol<3>1
ALUcontrol<1>1
PWR_3_o_GND_3_o_select_6_OUT<2>1
PWR_3_o_GND_3_o_select_6_OUT<3>1
PWR_3_o_GND_3_o_select_6_OUT<1>1
gnd
XST_GND
ALUdecoder

3.Explaination

The code provided is for an entity called "aludec" which is a behavioral implementation of an ALU decoder. It has three inputs - "funct", "aluop" and one output "alucontrol". Based on the input values of "funct" and "aluop", the "alucontrol" output is determined. If “aluop” is "00" or "01", the process block sets “alucontrol” to "0010" or "0110", respectively, indicating that the ALU should perform an addition or subtraction operation. If ”aluop” is neither "00" nor "01", the process block checks the “funct” signal to determine the appropriate “alucontrol” value for R-type instructions, such as add, sub, and, or,or slt. If “funct” does not match any of the R-type instructions, “alucontrol” is set to " ", indicating an unknown or unsupported operation.

Data Path

1.Code

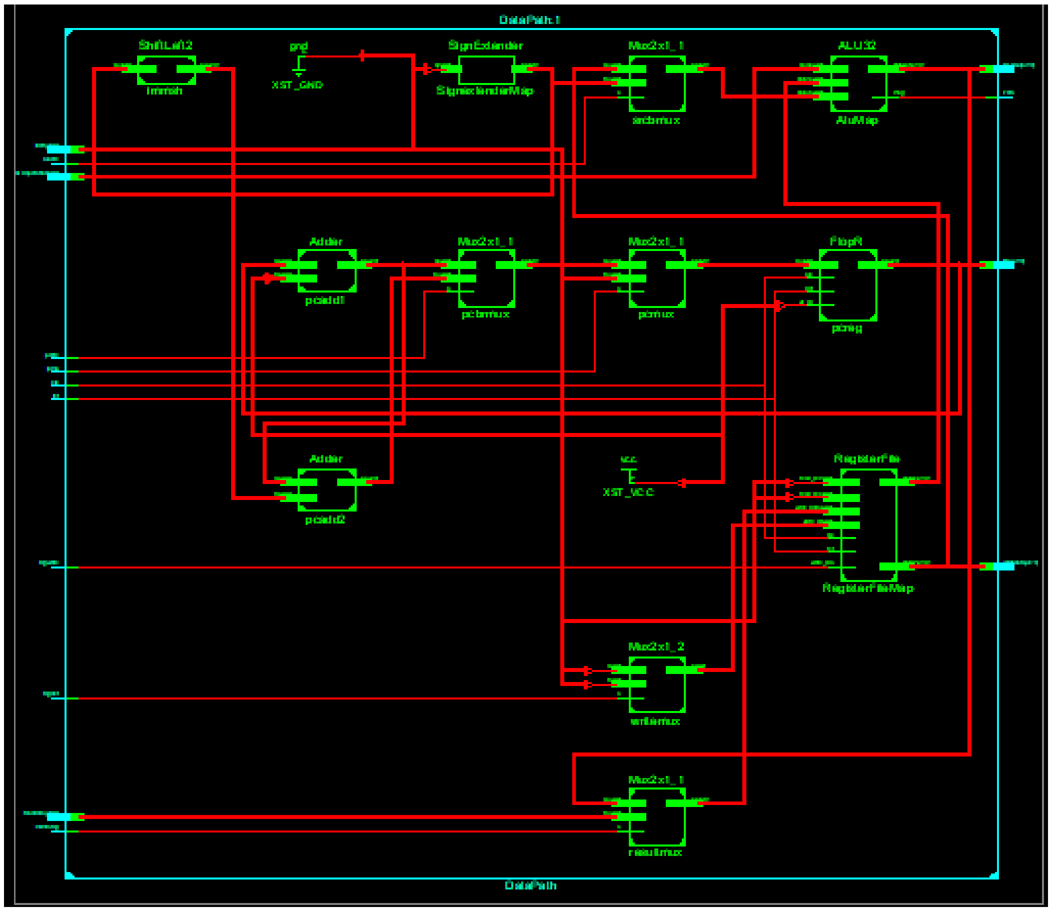
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.ShiftLeft2_Package.all;
use work.Adder_Package.all;
use work.Mux2x1_Package.all;
use work.SignExtender_Package.all;
use work.ALU32_Package.all;
use work.FlopR_PKG.all;
use work.Mux32x1_Package.all;
use work.RegisterFile_Package.all;
use work.Decoder4x1_Package.all;

entity DataPath is
   Port ( clk : in  STD_LOGIC;
        rst : in  STD_LOGIC;
        readdata : in  STD_LOGIC_VECTOR (31 downto 0);
        instr : in  STD_LOGIC_VECTOR (31 downto 0);
        memtoreg, pcsrc,alusrc,regwrite, regdst,jump: in STD_LOGIC;
        ALUoperation : in  STD_LOGIC_VECTOR (3 downto 0);
        zero : out  STD_LOGIC;
        PC : buffer  STD_LOGIC_VECTOR (31 downto 0);
        writedata : buffer  STD_LOGIC_VECTOR (31 downto 0);
        ALUout : buffer  STD_LOGIC_VECTOR (31 downto 0));
```

```
end DataPath;

architecture Behavioral of DataPath is


signal wdata : STD_LOGIC_VECTOR (31 downto 0);
signal writereg: STD_LOGIC_VECTOR(4 downto 0);
signal pcjump, pcnext, pcnextbr, pcplus4, pcbranch: STD_LOGIC_VECTOR(31
downto 0);
signal signimm, signimmsh: STD_LOGIC_VECTOR(31 downto 0);
signal srca, srcb, result: STD_LOGIC_VECTOR(31 downto 0);
signal pctemp,writedatatemp,aluouttemp: STD_LOGIC_VECTOR (31 downto 0);


Begin


--PC

pcjump <= pcplus4(31 downto 28) & instr(25 downto 0) & "00";

pcreg: FlopR port map(pcnext, pctemp,clk,rst,'1');

pcadd1: Adder port map(pctemp, X"00000004", pcplus4);
```

```
immsh: ShiftLeft2 port map(signimm, signimmsh);

pcadd2: Adder port map(pcplus4, signimmsh, pcbranch);

pcbrmux: Mux2x1 generic map(32) port map(pcplus4, pcbranch, pcsrc, pcnextbr);

pcmux: Mux2x1 generic map(32) port map(pcnextbr, pcjump, jump, pcnext);


--Register File

--                                read_sel1
 --       read_sel2
  --      write_sel
   --     write_ena
    --    clk
                          -- rst
--        write_data
 --       data1
  --      data2


writemux: Mux2x1 generic map(5) port map(instr(20 downto 16), instr(15 downto
11), regdst, writereg);
```

```
RegisterFileMap : RegisterFile port map(instr(25 downto 21),instr(20 downto 16),
writereg  ,regwrite ,clk,rst,wdata,srca,writedatatemp);

resultmux: Mux2x1 generic map(32) port
map(ALUout,readdata,memtoreg,wdata);

SignextenderMap : SignExtender port map(instr(15 downto 0),signimm);

--ALU

srcbmux: Mux2x1 generic map(32) port map(writedatatemp,signimm,alusrc,srcb);

AluMap : ALU32 port map (srca,srcb,ALUoperation,aluouttemp,zero);

pc <= pctemp;

writedata <= writedatatemp;

ALUout <= aluouttemp;

end Behavioral;
```

2.RTL



3.Explaination

This is code for a datapath module that includes a PC, register file, and ALU. Themodule has several input and output ports such as clk, reset, readdata, instr, memtoreg, pcsrc, alusrc, regwrite, regdst, jump, aluoperation, zero, pc, writedata, and aluout. The code includes signals for data manipulation, and several sub-components like a flopr, adder, mux, and signextender. The PC section is responsible for updating the program counter based on different conditions, while the register file reads and writes data to registers. The ALU performs arithmetic and logic operations on data coming from the register file or the signextender. The outputs of the datapath module are the updated PC value, data to be written to memory or register file, and the output of the ALU.

Register File

1.Code

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;

use IEEE.STD_LOGIC_unsigned.ALL;


use work.Decoder4x1_Package.all;

use work.FlopR_PKG.all;

use work.ALU32_Package.all;

use work.Mux32x1_Package.all;



entity RegisterFile is

   Port ( read_sel1 : in  STD_LOGIC_VECTOR (4 downto 0);

        read_sel2 : in  STD_LOGIC_VECTOR (4 downto 0);

        write_sel : in  STD_LOGIC_VECTOR (4 downto 0);

        write_ena : in  STD_LOGIC;

        clk : in  STD_LOGIC;

                          rst : in  STD_LOGIC;

        write_data : in  STD_LOGIC_VECTOR (31 downto 0);

        data1 : out  STD_LOGIC_VECTOR (31 downto 0);

        data2 : out  STD_LOGIC_VECTOR (31 downto 0));

end RegisterFile;
```

```
architecture Behavioral of RegisterFile is

--Decoder signal
signal Decoderout: STD_LOGIC_VECTOR (31 downto 0);

--Register signals
signal sig1 : STD_LOGIC_VECTOR(31 downto 0);
signal sig2 : STD_LOGIC_VECTOR(31 downto 0);
signal sig3 : STD_LOGIC_VECTOR(31 downto 0);
signal sig4 : STD_LOGIC_VECTOR(31 downto 0);
signal sig5 : STD_LOGIC_VECTOR(31 downto 0);
signal sig6 : STD_LOGIC_VECTOR(31 downto 0);
signal sig7 : STD_LOGIC_VECTOR(31 downto 0);
signal sig8 : STD_LOGIC_VECTOR(31 downto 0);
signal sig9 : STD_LOGIC_VECTOR(31 downto 0);
signal sig10 : STD_LOGIC_VECTOR(31 downto 0);
signal sig11 : STD_LOGIC_VECTOR(31 downto 0);
signal sig12 : STD_LOGIC_VECTOR(31 downto 0);
signal sig13 : STD_LOGIC_VECTOR(31 downto 0);
signal sig14 : STD_LOGIC_VECTOR(31 downto 0);
signal sig15 : STD_LOGIC_VECTOR(31 downto 0);
signal sig16 : STD_LOGIC_VECTOR(31 downto 0);
signal sig17 : STD_LOGIC_VECTOR(31 downto 0);
signal sig18 : STD_LOGIC_VECTOR(31 downto 0);
```

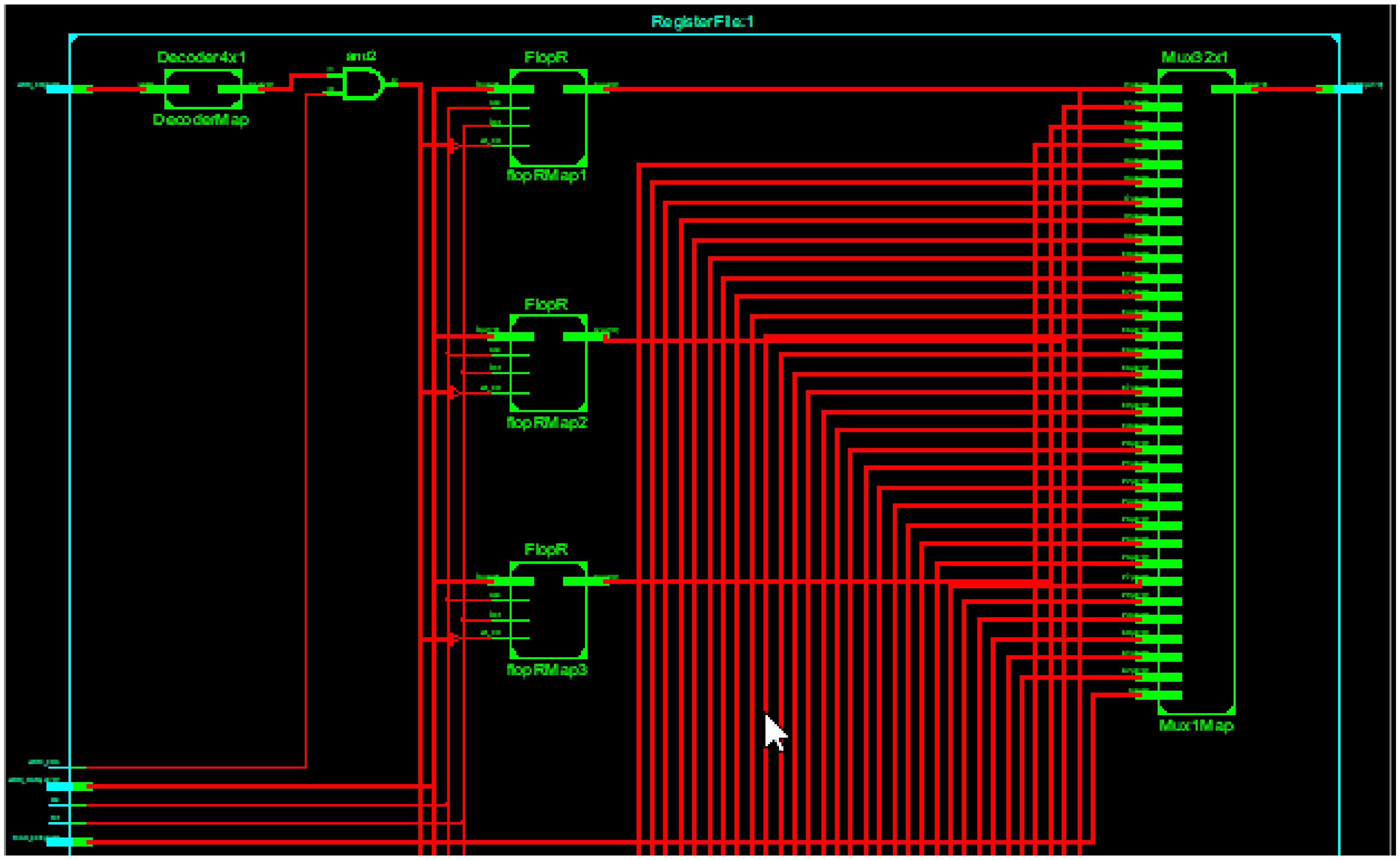
```vhdl
signal sig19 : STD_LOGIC_VECTOR(31 downto 0);
signal sig20 : STD_LOGIC_VECTOR(31 downto 0);
signal sig21 : STD_LOGIC_VECTOR(31 downto 0);
signal sig22 : STD_LOGIC_VECTOR(31 downto 0);
signal sig23 : STD_LOGIC_VECTOR(31 downto 0);
signal sig24 : STD_LOGIC_VECTOR(31 downto 0);
signal sig25 : STD_LOGIC_VECTOR(31 downto 0);
signal sig26 : STD_LOGIC_VECTOR(31 downto 0);
signal sig27 : STD_LOGIC_VECTOR(31 downto 0);
signal sig28 : STD_LOGIC_VECTOR(31 downto 0);
signal sig29 : STD_LOGIC_VECTOR(31 downto 0);
signal sig30 : STD_LOGIC_VECTOR(31 downto 0);
signal sig31 : STD_LOGIC_VECTOR(31 downto 0);
signal sig32 : STD_LOGIC_VECTOR(31 downto 0);


begin


--Decoder Implement
--I,O
DecoderMap : Decoder4x1 port map(write_sel,DecoderOut);

--FlopR Implement
--D,Q,Clk,Rst, wr_en
```

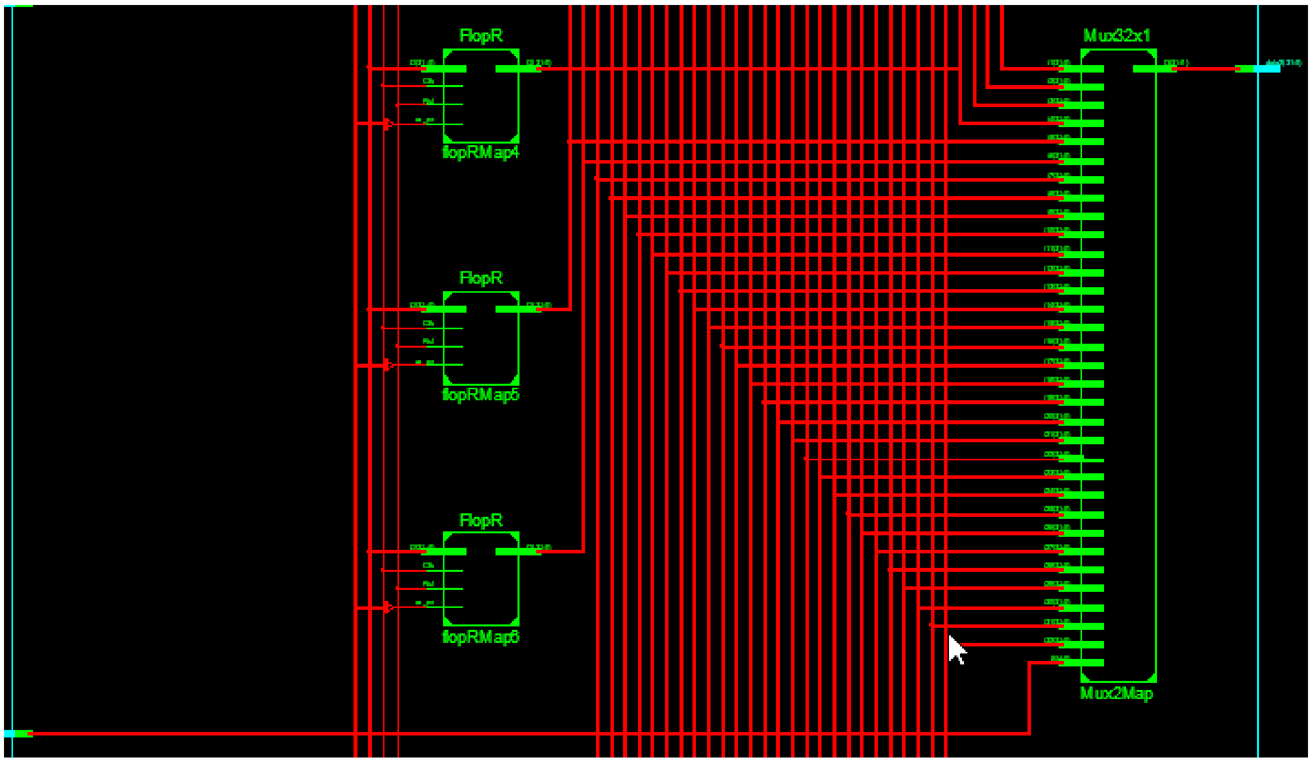
```
flopRMap1: FlopR port map(write_data, sig1, Clk, rst, write_ena and
Decoderout(0));

flopRMap2: FlopR port map(write_data, sig2, Clk, rst, write_ena and
Decoderout(1));

flopRMap3: FlopR port map(write_data, sig3, Clk, rst, write_ena and
Decoderout(2));

flopRMap4: FlopR port map(write_data, sig4, Clk, rst, write_ena and
Decoderout(3));

flopRMap5: FlopR port map(write_data, sig5, Clk, rst, write_ena and
Decoderout(4));

flopRMap6: FlopR port map(write_data, sig6, Clk, rst, write_ena and
Decoderout(5));

flopRMap7: FlopR port map(write_data, sig7, Clk, rst, write_ena and
Decoderout(6));

flopRMap8: FlopR port map(write_data, sig8, Clk, rst, write_ena and
Decoderout(7));

flopRMap9: FlopR port map(write_data, sig9, Clk, rst, write_ena and
Decoderout(8));

flopRMap10: FlopR port map(write_data, sig10, Clk, rst, write_ena and
Decoderout(9));

flopRMap11: FlopR port map(write_data, sig11, Clk, rst, write_ena and
Decoderout(10));

flopRMap12: FlopR port map(write_data, sig12, Clk, rst, write_ena and
Decoderout(11));

flopRMap13: FlopR port map(write_data, sig13, Clk, rst, write_ena and
Decoderout(12));

flopRMap14: FlopR port map(write_data, sig14, Clk, rst, write_ena and
Decoderout(13));
```

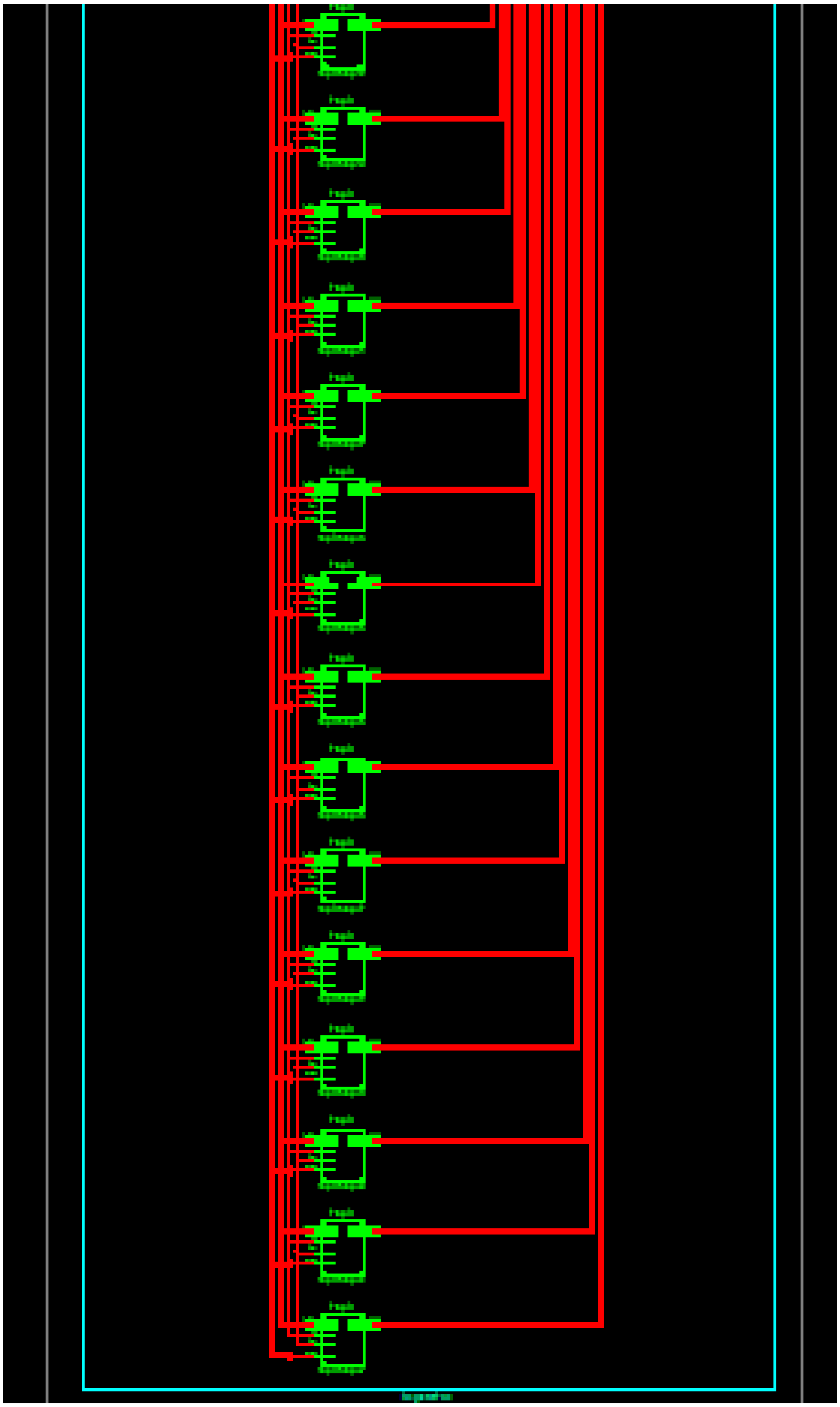
```
flopRMap15: FlopR port map(write_data, sig15, Clk, rst, write_ena and
Decoderout(14));

flopRMap16: FlopR port map(write_data, sig16, Clk, rst, write_ena and
Decoderout(15));

flopRMap17: FlopR port map(write_data, sig17, Clk, rst, write_ena and
Decoderout(16));

flopRMap18: FlopR port map(write_data, sig18, Clk, rst, write_ena and
Decoderout(17));

flopRMap19: FlopR port map(write_data, sig19, Clk, rst, write_ena and
Decoderout(18));

flopRMap20: FlopR port map(write_data, sig20, Clk, rst, write_ena and
Decoderout(19));

flopRMap21: FlopR port map(write_data, sig21, Clk, rst, write_ena and
Decoderout(20));

flopRMap22: FlopR port map(write_data, sig22, Clk, rst, write_ena and
Decoderout(21));

flopRMap23: FlopR port map(write_data, sig23, Clk, rst, write_ena and
Decoderout(22));

flopRMap24: FlopR port map(write_data, sig24, Clk, rst, write_ena and
Decoderout(23));

flopRMap25: FlopR port map(write_data, sig25, Clk, rst, write_ena and
Decoderout(24));

flopRMap26: FlopR port map(write_data, sig26, Clk, rst, write_ena and
Decoderout(25));

flopRMap27: FlopR port map(write_data, sig27, Clk, rst, write_ena and
Decoderout(26));

flopRMap28: FlopR port map(write_data, sig28, Clk, rst, write_ena and
Decoderout(27));
```

```
flopRMap29: FlopR port map(write_data, sig29, Clk, rst, write_ena and
Decoderout(28));

flopRMap30: FlopR port map(write_data, sig30, Clk, rst, write_ena and
Decoderout(29));

flopRMap31: FlopR port map(write_data, sig31, Clk, rst, write_ena and
Decoderout(30));

flopRMap32: FlopR port map(write_data, sig32, Clk, rst, write_ena and
Decoderout(31));


--Mux 1

--signal 1-32,Selector,Output

Mux1Map : Mux32x1 port map(

   sig1, sig2, sig3, sig4, sig5, sig6, sig7, sig8,

   sig9, sig10, sig11, sig12, sig13, sig14, sig15, sig16,

   sig17, sig18, sig19, sig20, sig21, sig22, sig23, sig24,

   sig25, sig26, sig27, sig28, sig29, sig30, sig31, sig32,

   read_sel1, data1);

Mux2Map : Mux32x1 port map(

   sig1, sig2, sig3, sig4, sig5, sig6, sig7, sig8,

   sig9, sig10, sig11, sig12, sig13, sig14, sig15, sig16,

   sig17, sig18, sig19, sig20, sig21, sig22, sig23, sig24,

   sig25, sig26, sig27, sig28, sig29, sig30, sig31, sig32,

   read_sel2, data2);



end Behavioral;
```

2.RTL


RegisterFile1
Decoder4x1
DecoderMap
and2
FlopR
flopRMap1
FlopR
flopRMap2
FlopR
flopRMap3
Mux32x1
Mux1Map


FlopR
flopRMap4
FlopR
flopRMap5
FlopR
flopRMap6
Mux32x1
Mux2Map



3.Explaination

The register file has 32 registers each of 32-bit width. The register file has two read ports and one write port. The register file is asynchronous reset andpositive-edge clock-triggered.

The module RegFile has the following ports:

- read_sel1, read_sel2: 5-bit input ports that are used to select theregisters to be read from.
- write_sel: 5-bit input port that is used to select the register to bewritten to.
- write_ena: 1-bit input port that is used to enable/disable writing to theselected register.
- clk, reset: input ports for clock and reset signals respectively.
- write_data: 32-bit input port that provides data to be written to theselected register.
- data1, data2: 32-bit output ports that provide the data read from theselected registers

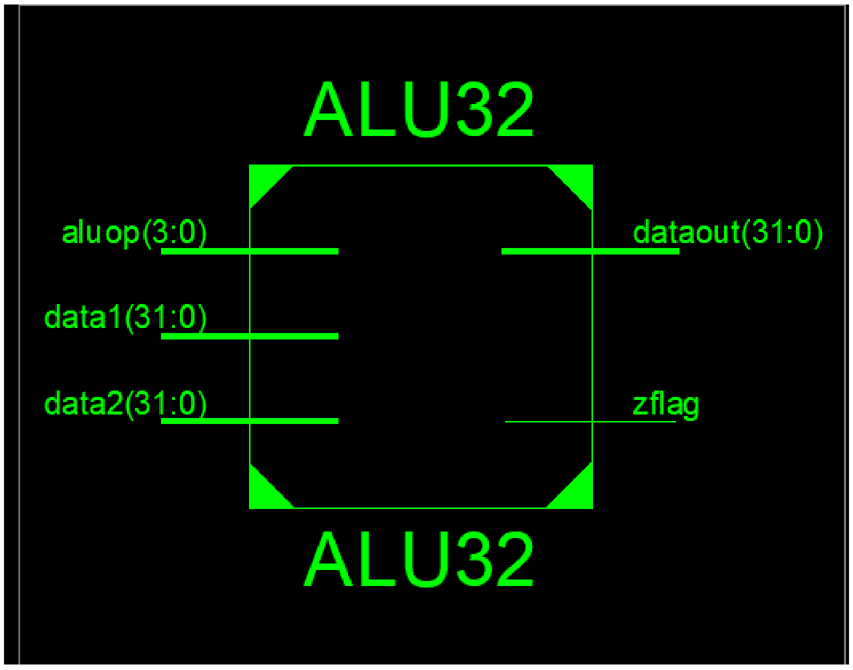
ALU

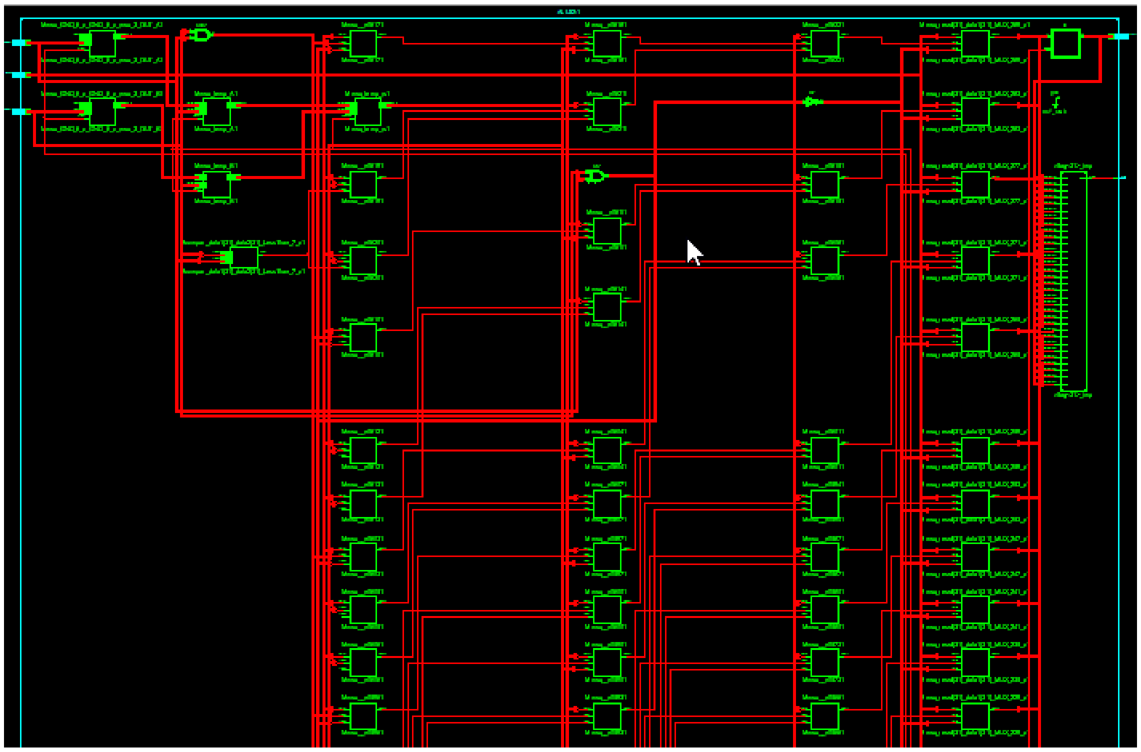
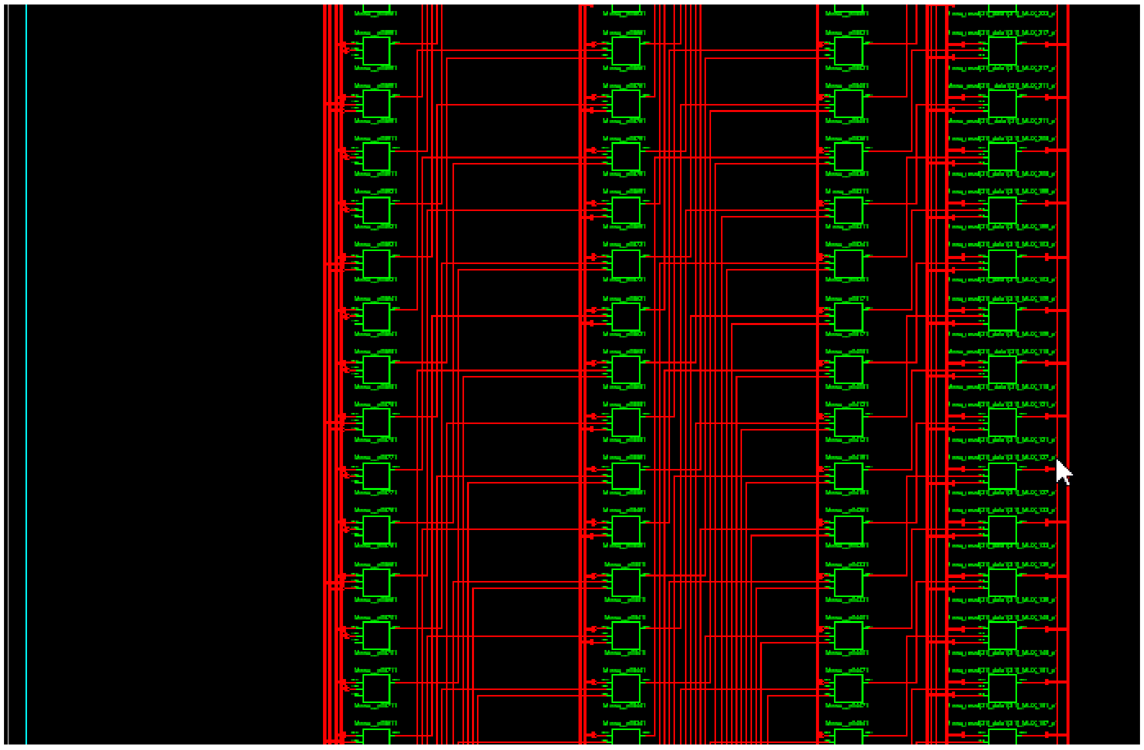
1.Code

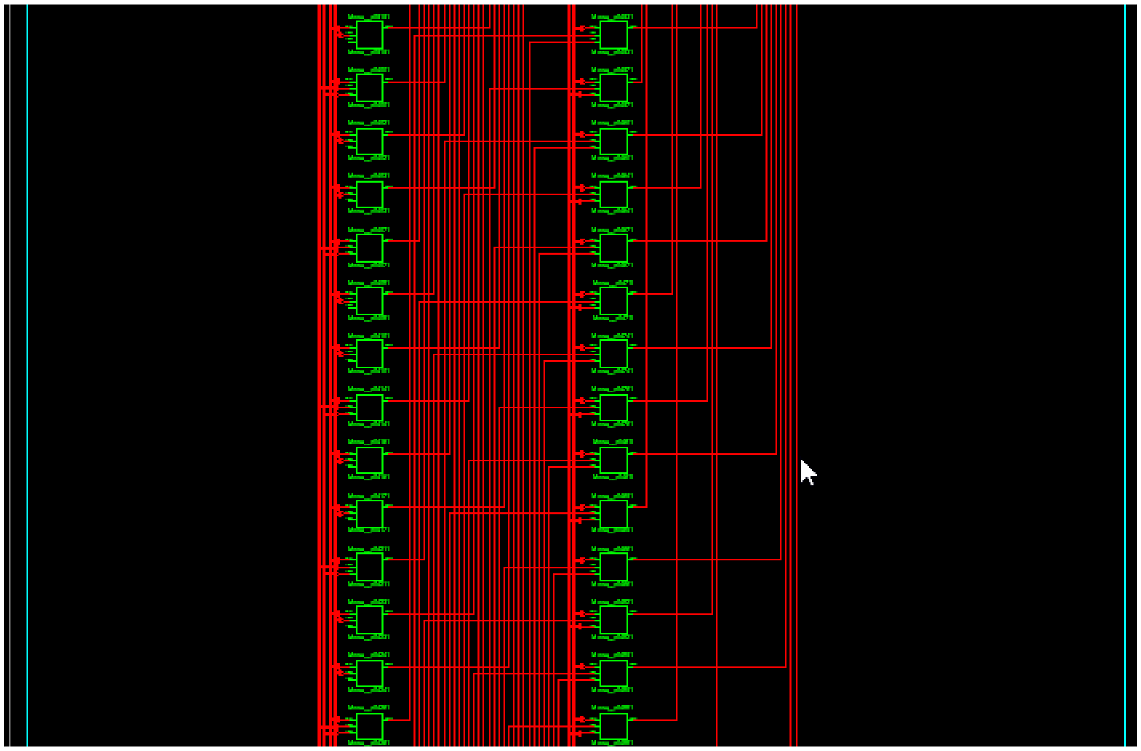
```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;

use IEEE.STD_LOGIC_unsigned.ALL;


entity ALU32 is

   Port ( data1 : in  STD_LOGIC_VECTOR (31 downto 0);

          data2 : in  STD_LOGIC_VECTOR (31 downto 0);

          aluop : in  STD_LOGIC_VECTOR (3 downto 0);

          dataout : out  STD_LOGIC_VECTOR (31 downto 0);

          zflag : out  STD_LOGIC);

end ALU32;


architecture Behavioral of ALU32 is


signal result: STD_LOGIC_VECTOR (31 downto 0);

signal slt: STD_LOGIC_VECTOR (31 downto 0);

signal temp: STD_LOGIC_VECTOR (32 downto 0);

begin


temp <= (('0' & data1) + ('0' & data2)) WHEN aluop = "0010" ELSE

(('0' & data1) - ('0' & data2)) WHEN aluop = "0110" ELSE

 (OTHERS=>'0');
```

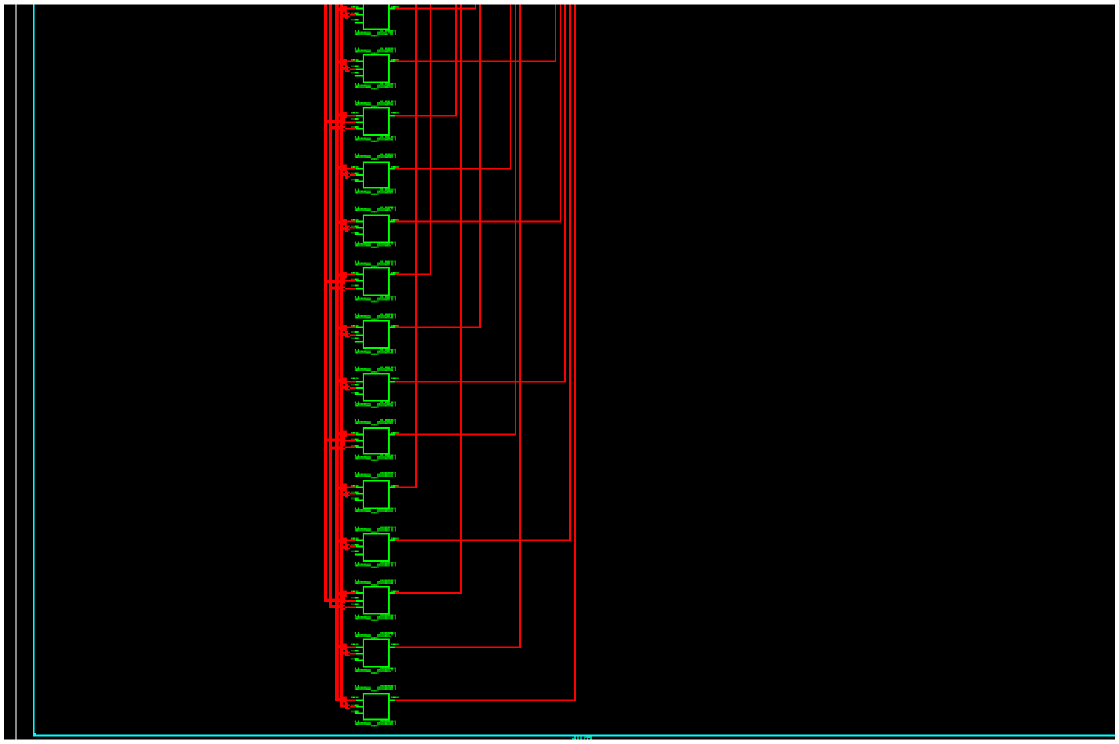
```
slt <= X"00000001" WHEN data1 < data2 ELSE
X"00000000";
result <= data1 AND data2 when aluop = "0000" Else
data1 OR data2 when aluop = "0001" ELSE
temp(31 downto 0) when aluop = "0010" ELSE
temp(31 downto 0) when aluop = "0110" ELSE
data1 NOR data2 when aluop = "1100" ELSE
slt when aluop = "0111";
zflag <= '1' when result = X"00000000" ELSE
'0';
dataout <= result;

end Behavioral;
```

2.RTL







3.Explaination

An ALU (Arithmetic Logic Unit) module that performs various arithmetic andlogic operations on two 32-bit input data (data1 and data2) based on a 4-bitoperation code (aluop) and produces a 32-bit output (dataout) and a zero flag (zflag).

Mux 2*1

1.Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity Mux2x1 is
            generic(width: integer);
    Port ( A,B : in  STD_LOGIC_VECTOR (width-1 downto 0);
           S : in  STD_LOGIC;
           O : out  STD_LOGIC_VECTOR (width-1 downto 0));
end Mux2x1;


architecture Behavioral of Mux2x1 is


begin


O <= B when S = '1' else A;


end Behavioral;
```

Mux 32*1

1.Code

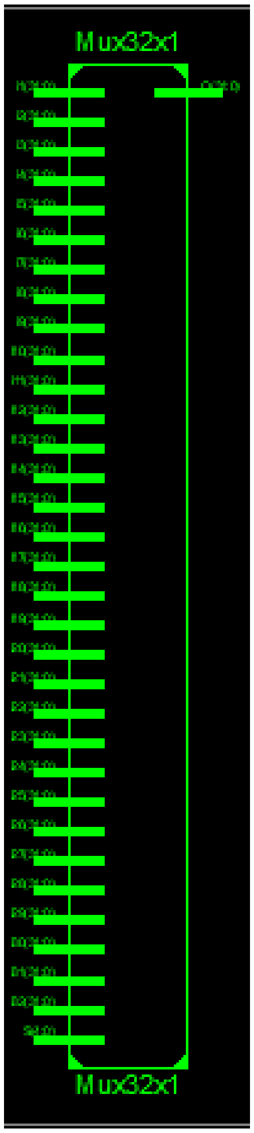
```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


entity Mux32x1 is

    Port ( i1 : in  STD_LOGIC_VECTOR (31 downto 0);

           i2 : in  STD_LOGIC_VECTOR (31 downto 0);

           i3 : in  STD_LOGIC_VECTOR (31 downto 0);

           i4 : in  STD_LOGIC_VECTOR (31 downto 0);

           i5 : in  STD_LOGIC_VECTOR (31 downto 0);

           i6 : in  STD_LOGIC_VECTOR (31 downto 0);

           i7 : in  STD_LOGIC_VECTOR (31 downto 0);

           i8 : in  STD_LOGIC_VECTOR (31 downto 0);

           i9 : in  STD_LOGIC_VECTOR (31 downto 0);

           i10 : in  STD_LOGIC_VECTOR (31 downto 0);

           i11 : in  STD_LOGIC_VECTOR (31 downto 0);

           i12 : in  STD_LOGIC_VECTOR (31 downto 0);

           i13 : in  STD_LOGIC_VECTOR (31 downto 0);

           i14 : in  STD_LOGIC_VECTOR (31 downto 0);

           i15 : in  STD_LOGIC_VECTOR (31 downto 0);

           i16 : in  STD_LOGIC_VECTOR (31 downto 0);

           i17 : in  STD_LOGIC_VECTOR (31 downto 0);

           i18 : in  STD_LOGIC_VECTOR (31 downto 0);
```

```
        i19 : in  STD_LOGIC_VECTOR (31 downto 0);
        i20 : in  STD_LOGIC_VECTOR (31 downto 0);
        i21 : in  STD_LOGIC_VECTOR (31 downto 0);
        i22 : in  STD_LOGIC_VECTOR (31 downto 0);
        i23 : in  STD_LOGIC_VECTOR (31 downto 0);
        i24 : in  STD_LOGIC_VECTOR (31 downto 0);
        i25 : in  STD_LOGIC_VECTOR (31 downto 0);
        i26 : in  STD_LOGIC_VECTOR (31 downto 0);
        i27 : in  STD_LOGIC_VECTOR (31 downto 0);
        i28 : in  STD_LOGIC_VECTOR (31 downto 0);
        i29 : in  STD_LOGIC_VECTOR (31 downto 0);
        i30 : in  STD_LOGIC_VECTOR (31 downto 0);
        i31 : in  STD_LOGIC_VECTOR (31 downto 0);
        i32 : in  STD_LOGIC_VECTOR (31 downto 0);
        S : in  STD_LOGIC_VECTOR (4 downto 0);
        O : out  STD_LOGIC_VECTOR (31 downto 0));
end Mux32x1;

architecture Behavioral of Mux32x1 is


begin
        o <=
        i1 when s="00000"else
        i2 when s="00001"else
```

```
i3 when s="00010"else
i4 when s="00011"else
i5 when s="00100"else
i6 when s="00101"else
i7 when s="00110"else
i8 when s="00111"else
i9 when s="01000"else
i10 when s="01001"else
i11 when s="01010"else
i12 when s="01011"else
i13 when s="01100"else
i14 when s="01101"else
i15 when s="01110"else
i16 when s="01111"else
i17 when s="10000"else
i18 when s="10001"else
i19 when s="10010"else
i20 when s="10011"else
i21 when s="10100"else
i22 when s="10101"else
i23 when s="10110"else
i24 when s="10111"else
i25 when s="11000"else
i26 when s="11001"else
```

```
        i27 when s="11010"else

        i28 when s="11011"else

        i29 when s="11100"else

        i30 when s="11101"else

        i31 when s="11110"else

        i32 when s="11111"ELSE

        "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";


end Behavioral;
```

2.RTL



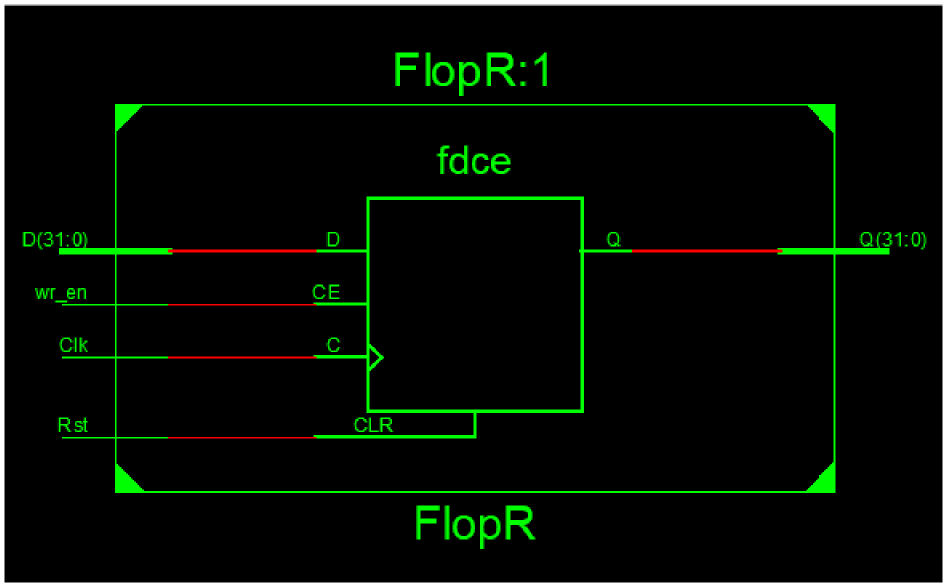
Flopr

1.Code

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use ieee.std_logic_unsigned.all;




entity FlopR is

   Port ( D : in  STD_LOGIC_VECTOR (31 downto 0);

          Q : out  STD_LOGIC_VECTOR (31 downto 0);

          Clk : in  STD_LOGIC;

          Rst : in  STD_LOGIC;

                              wr_en: in STD_LOGIC);

end FlopR;

--D,Q,Clk,Rst

architecture Behavioral of FlopR is

begin

PROCESS(Clk,rst)


Begin


IF(rst='1') THEN

          Q <=(others => '0');
```

```
ELSIF (Clk'EVENT and clk = '1' and wr_en = '1') THEN
          Q <= D;
          END IF;
END PROCESS;


end Behavioral;
```

2.RTL



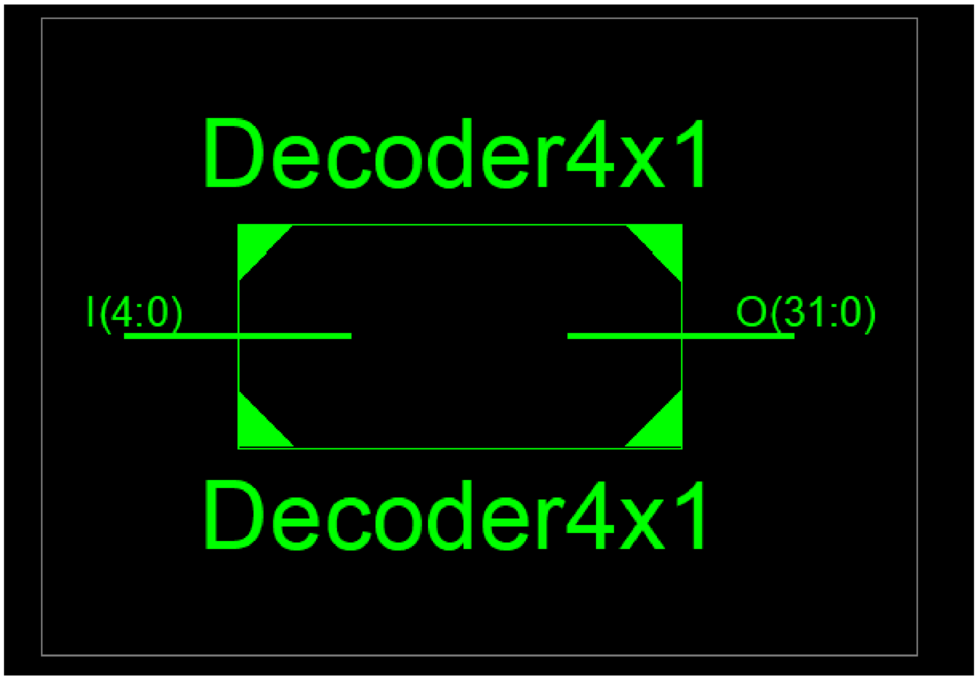
Decoder

1.Code

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;



entity Decoder4x1 is

    Port ( I : in  STD_LOGIC_VECTOR (4 downto 0);

           O : out  STD_LOGIC_VECTOR (31 downto 0));

end Decoder4x1;


architecture Behavioral of Decoder4x1 is


begin

O <=

"00000000000000000000000000000001" when I = "00000" else

"00000000000000000000000000000010" when I = "00001" else

"00000000000000000000000000000100" when I = "00010" else

"00000000000000000000000000001000" when I = "00011" else

"00000000000000000000000000010000" when I = "00100" else

"00000000000000000000000000100000" when I = "00101" else

"00000000000000000000000001000000" when I = "00110"else

"00000000000000000000000010000000" when I = "00111"else

"00000000000000000000000100000000" when I = "01000"else

"00000000000000000000001000000000" when I = "01001"else
```

```
"00000000000000000000010000000000" when I = "01010"else
"00000000000000000000100000000000" when I = "01011"else
"00000000000000000001000000000000" when I = "01100"else
"00000000000000000010000000000000" when I = "01101"else
"00000000000000000100000000000000" when I = "01110"else
"00000000000000001000000000000000" when I = "01111"else
"00000000000000010000000000000000" when I = "10000"else
"00000000000000100000000000000000" when I = "10001"else
"00000000000001000000000000000000" when I = "10010"else
"00000000000010000000000000000000" when I = "10011"else
"00000000000100000000000000000000" when I = "10100"else
"00000000001000000000000000000000" when I = "10101"else
"00000000010000000000000000000000" when I = "10110"else
"00000000100000000000000000000000" when I = "10111"else
"00000001000000000000000000000000" when I = "11000"else
"00000010000000000000000000000000" when I = "11001"else
"00000100000000000000000000000000" when I = "11010"else
"00001000000000000000000000000000" when I = "11011"else
"00010000000000000000000000000000" when I = "11100"else
"00100000000000000000000000000000" when I = "11101"else
"01000000000000000000000000000000" when I = "11110"else
"10000000000000000000000000000000" when I = "11111";

end Behavioral;
```

2.RTL

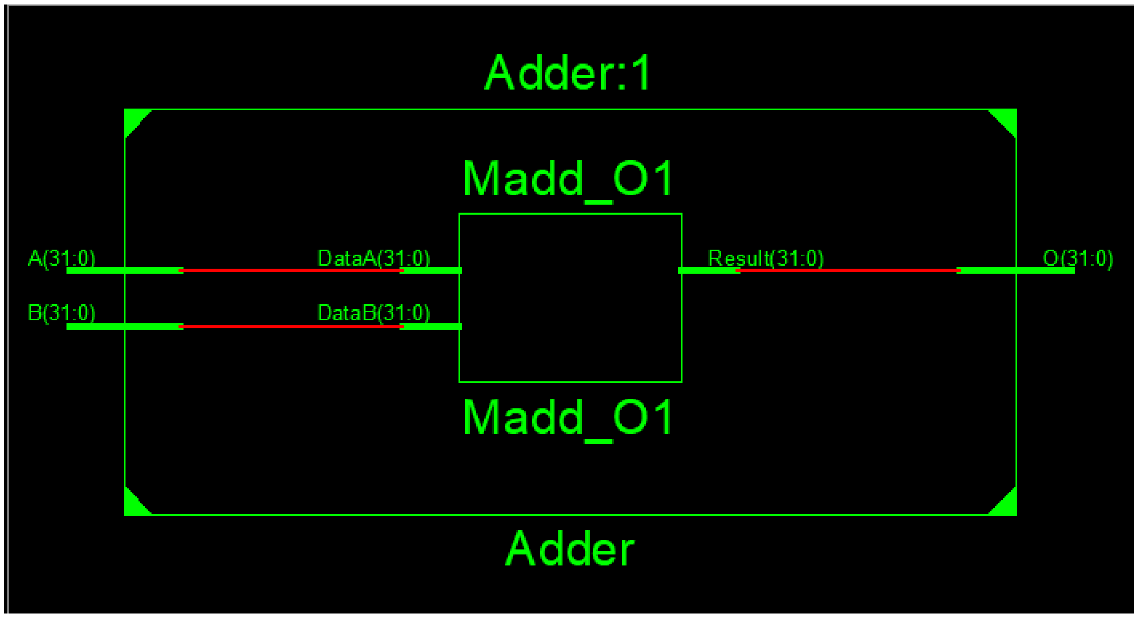


Adder

1.Code

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Adder is

   Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);

          B : in  STD_LOGIC_VECTOR (31 downto 0);

          O : out  STD_LOGIC_VECTOR (31 downto 0));

end Adder;


architecture Behavioral of Adder is


begin


O <= A+B;



end Behavioral;
```

2.RTL

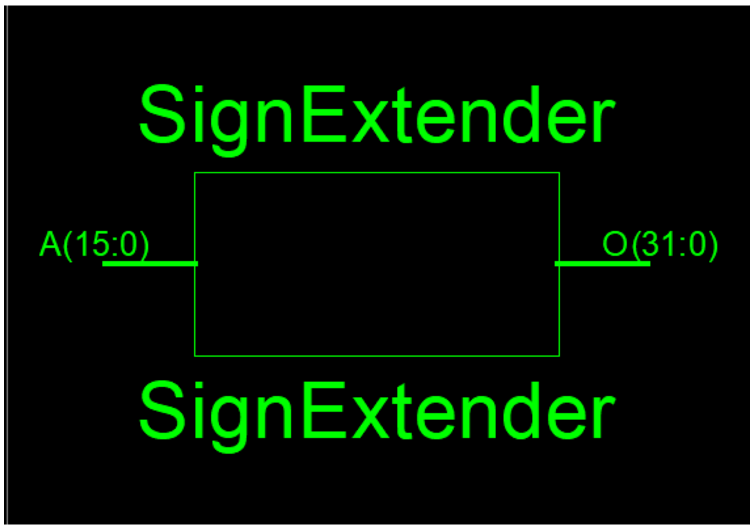

Adder:1
Madd_O1
A(31:0)
DataA(31:0)
B(31:0)
DataB(31:0)
Result(31:0)
O(31:0)
Madd_O1
Adder

Sign Extender

1.Code

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


entity SignExtender is

   Port ( A : in  STD_LOGIC_VECTOR (15 downto 0);

          O : out  STD_LOGIC_VECTOR (31 downto 0));

end SignExtender;


architecture Behavioral of SignExtender is


begin


O <= X"ffff" & A when A(15) = '1' else X"0000" & A;

end Behavioral;
```

2.RTL

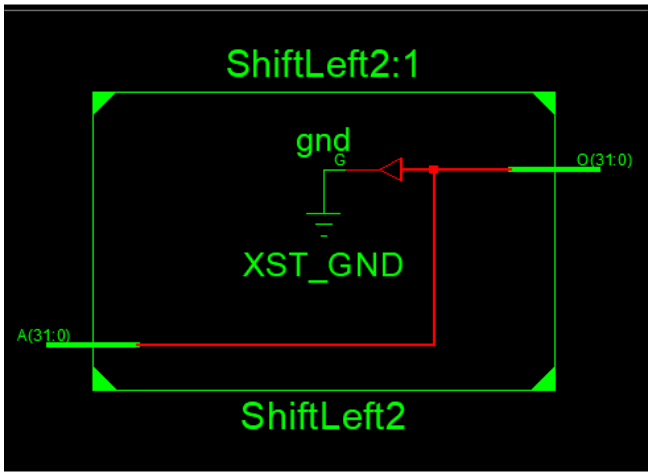


Shift left 2

1.Code

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


entity ShiftLeft2 is

   Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);

          O : out  STD_LOGIC_VECTOR (31 downto 0));

end ShiftLeft2;


architecture Behavioral of ShiftLeft2 is


begin


O <= A(29 downto 0) & "00";


end Behavioral;
```

2.RTL



Test Files

Main Module Test

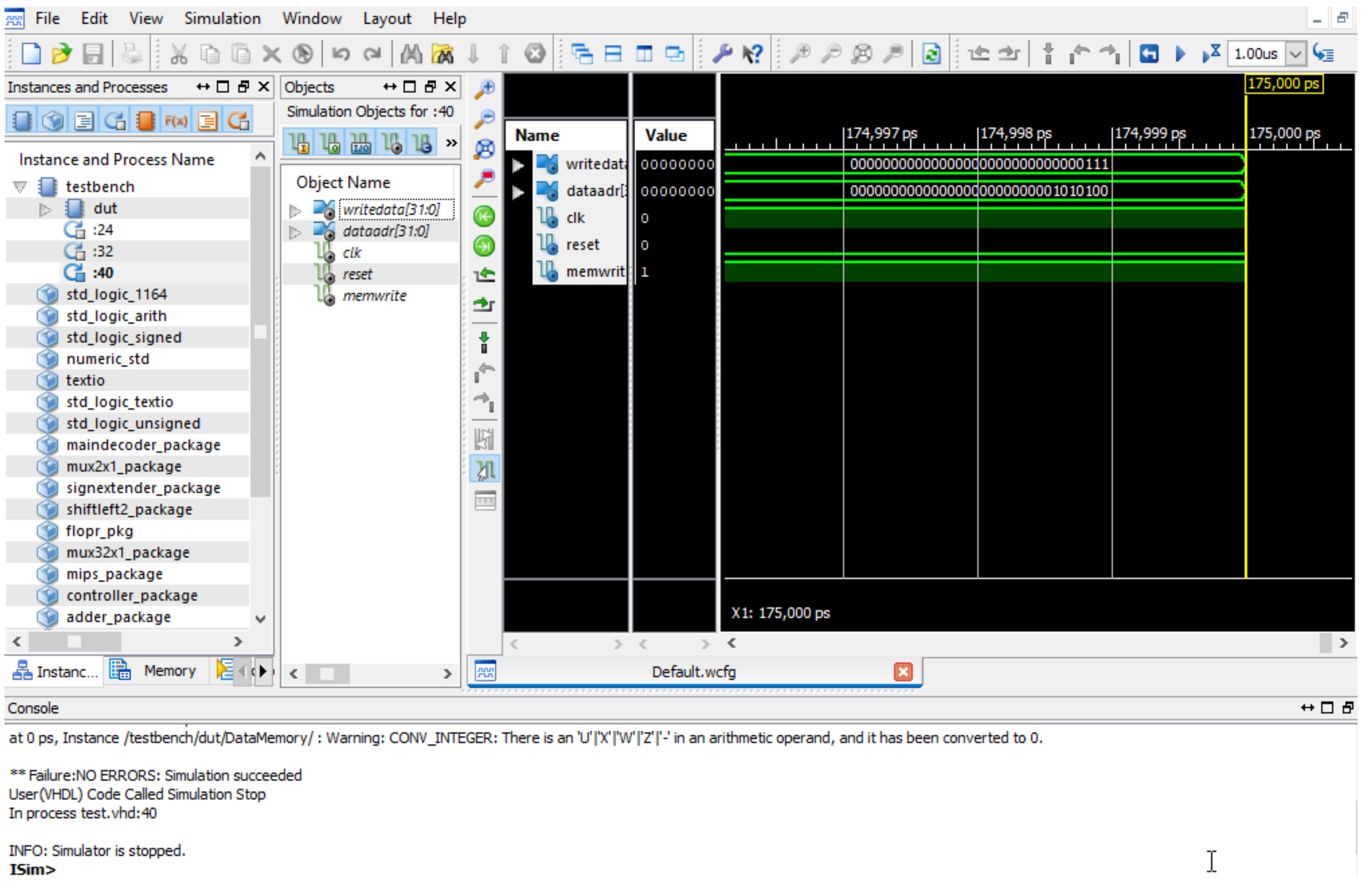
1. Code

```
library IEEE;

use IEEE.STD_LOGIC_1164.all;

use IEEE.STD_LOGIC_SIGNED.all;

use IEEE.STD_LOGIC_ARITH.all;

use ieee.numeric_std.all;

-- use IEEE.NUMERIC_STD_UNSIGNED.all;

entity testbench is

end;


architecture test of testbench is

component TopModule

port(clk, rst: in STD_LOGIC;

writedata, dataadr: out STD_LOGIC_VECTOR(31 downto 0);

memwrite: out STD_LOGIC);

end component;

signal writedata, dataadr: STD_LOGIC_VECTOR(31 downto 0);

signal clk, reset, memwrite: STD_LOGIC;


begin

-- instantiate device to be tested

dut: TopModule port map(clk, reset, writedata, dataadr, memwrite);
```

```
-- Generate clock with 10 ns period
process begin
clk <= '1';
wait for 5 ns;
clk <= '0';
wait for 5 ns;
end process;

-- Generate reset for first two clock cycles
process begin
reset <= '1';
wait for 22 ns;
reset <= '0';
wait;
end process;

-- check that 7 gets written to address 84 at end of program
process(clk) begin
if (clk'event and clk = '0' and memwrite = '1') then
if (CONV_INTEGER(dataadr) = 84 and CONV_INTEGER(writedata) = 7) then
report "NO ERRORS: Simulation succeeded" severity failure;
elsif (CONV_INTEGER(dataadr) = 84) then
report "Simulation failed" severity failure;
```

```
end if;

end if;


end process;

end;
```

2.Test Results



```
at 0 ps, Instance /testbench/dut/DataMemory/ : Warning: CONV_INTEGER: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, and it has been converted to 0.

** Failure:NO ERRORS: Simulation succeeded
User(VHDL) Code Called Simulation Stop
In process test.vhd:40

INFO: Simulator is stopped.
ISim>
```