ECE 528

Final Project Report

Shivam Kundan

Anup Kumar Biswas

Design Description

The aim of this project was to design a simple 32-bit, 5-pipeline-stage MIPS CPU which can execute basic arithmetic and logical instructions. The design is implemented on a Basys-3 Board with Artix-7 FPGA.

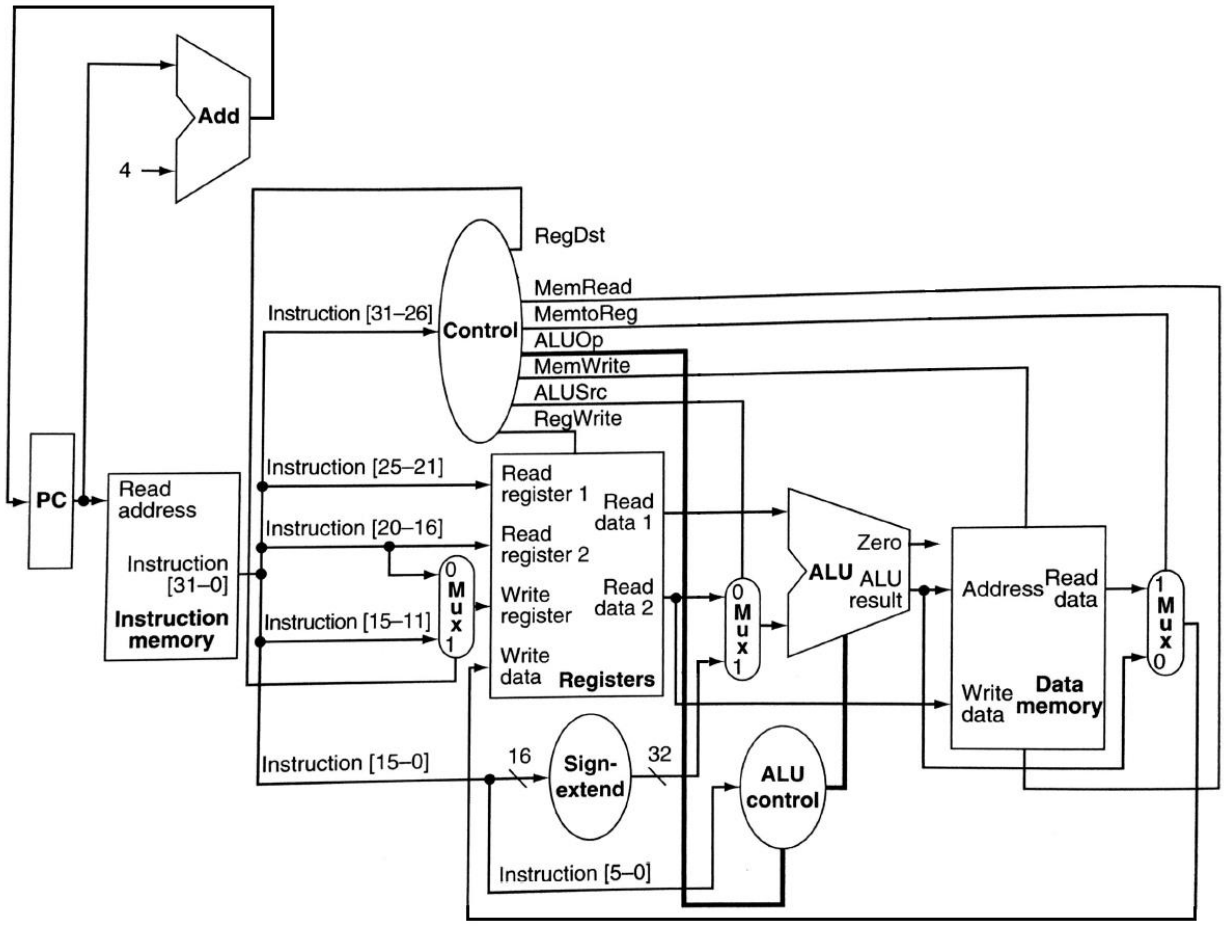

Fig 1: Basic Representation of CPU

The top-level design consists of five modules, corresponding to each stage of a 5-stage MIPS pipeline – IF, ID, EX, MEM, and WB.

1. Instruction Fetch Module

```
module IF_top(input clk,
              input en,
              input reset,
              output [31:0] IF_out
    );

    wire [5:0] PC_out;
    PC PC0 (clk,reset,en,PC_out);
    wire [31:0] instruction;

    Instruction_Memory_BROM Instruction_Memory(PC_out,clk,instruction);

    Pipeline_Reg_Template IF_ID(clk,reset,instruction,IF_out);

endmodule
```

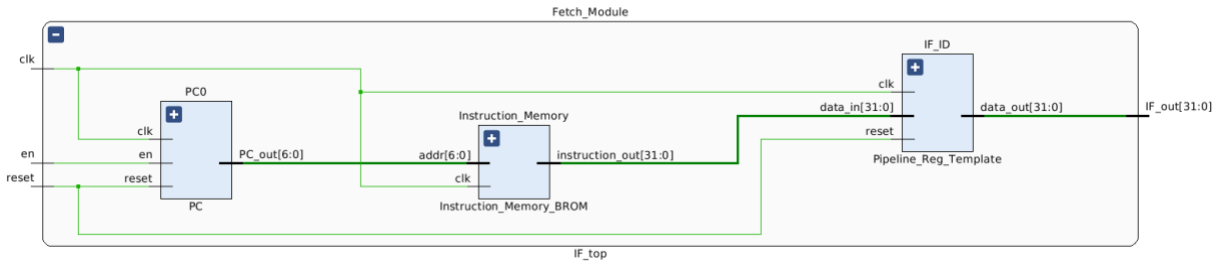

Fig 2: IF Top Module

i. Program Counter

```
module PC(
    input clk,
    input reset,
    input en,
    output reg[6:0] PC_out); //Because we need 0 to 63

    always @(posedge clk) begin
       if(reset)
           PC_out=63;
       else begin
            if(en) begin
                if (PC_out==63)
                    PC_out=0;
                else
                    PC_out = PC_out + 1;
            end
            else
```

```
                    PC_out=PC_out;
        end
        end
endmodule
```

ii. Instruction Memory (Block ROM IP)

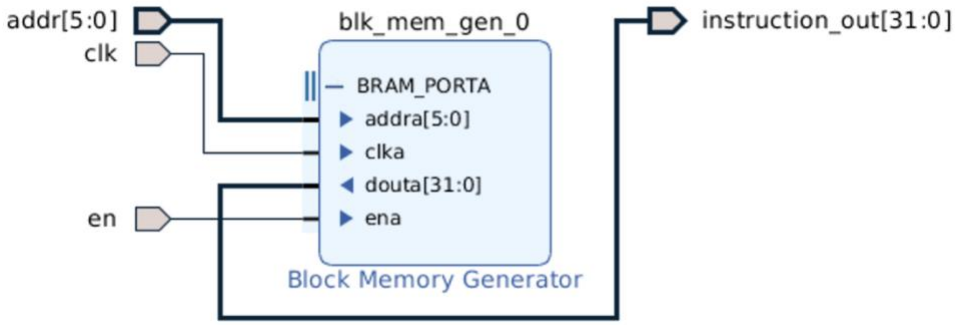

Fig 3: IP Block Memory Design

iii. IF/ID Pipeline Register

```
module Pipeline_Reg_Template(
    input clk,
    input reset,
    input [31:0] data_in,
    output reg [31:0] data_out
    );

    always@(posedge clk) begin
    if (reset)
        data_out=0;
    else
            data_out=data_in;
    end

endmodule
```

2. Decode Module

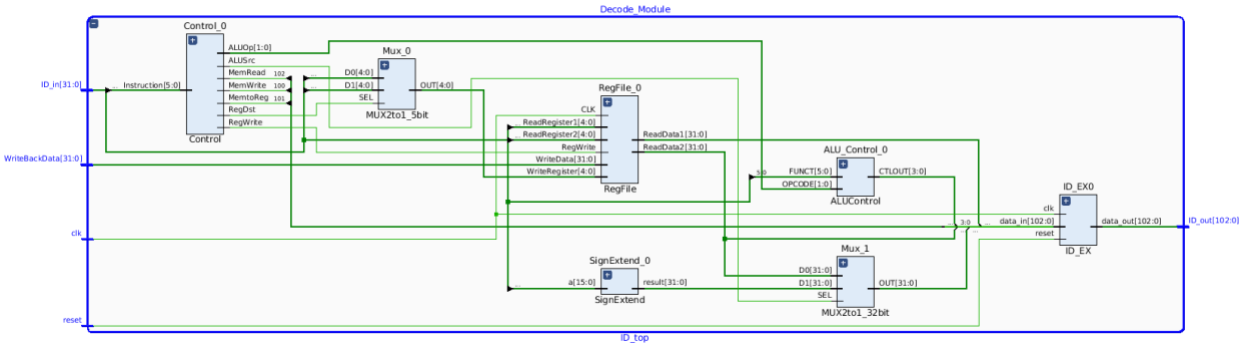

Fig 4: Decode Top Module

i. Control Module

```
1.  //MIPS decoder circuit plus ID_EX pipeline register
2.
3.  module ID_top(input clk,
4.                input reset,
5.                input [31:0] ID_in,
6.                input [31:0] WriteBackData,
7.                output [102:0] ID_out
8.      );
9.
10.
11.     //Internal connections
12.     wire MemRead, MemtoReg, MemWrite, RegDst, ALUSrc, RegWrite;
13.     wire [1:0] ALUOp;
14.     wire [3:0] ALUctlOUT;
15.     wire [4:0] WriteRegister;
16.     wire [31:0] ReadData1,ReadData2, RegsWriteData,SignEOut, DataMemRead, R
    egFile_Out;
17.     wire [7:0] PCOutWire;
18.     reg en;
19.
20.     //Instantiate Control Unit
21.     Control Control_0(ID_in[31:26],
22.                   ALUOp,
23.                   MemRead,
24.                   MemWrite,
25.                   MemtoReg,
26.                   RegDst,
27.                   RegWrite,
28.                   ALUSrc
29.                   );
30.
31.     //Instantiate Register File
32.     RegFile RegFile_0(clk,
33.               RegWrite,
34.               ID_in[25:21],
35.               ID_in[20:16],
36.               WriteRegister,
37.               WriteBackData,
```

```
            ReadData1,
            ReadData2
            );

    //Instantiate ALU Control
    ALUControl ALU_Control_0(ID_in[5:0],
                             ALUOp,
                             ALUctlOUT
                             );
    //Instantiate Sign Extend Module
    SignExtend SignExtend_0(ID_in[15:0],
                               SignEOut
                            );
    //Instantiate mux0
    MUX2to1_5bit Mux_0(ID_in[20:16],
                             ID_in[15:11],
                             WriteRegister,
                             RegDst
                             );

    //Instantiate mux1
    MUX2to1_32bit Mux_1(ReadData2,
                         SignEOut,
                         RegFile_Out,
                         ALUSrc
                         );

    //Assign output values
    wire [102:0] dout_temp;
    assign dout_temp[102]=MemRead;
    assign dout_temp[101]=MemtoReg;
    assign dout_temp[100]=MemWrite;
    assign dout_temp[99:68]=ReadData1;
    assign dout_temp[67:36]=RegFile_Out;
    assign dout_temp[35:4]=ReadData2;
    assign dout_temp[3:0]=ALUctlOUT;

    //Instantiate ID_EX pipeline register
    ID_EX ID_EX0(clk,reset,dout_temp,ID_out);

endmodule
```

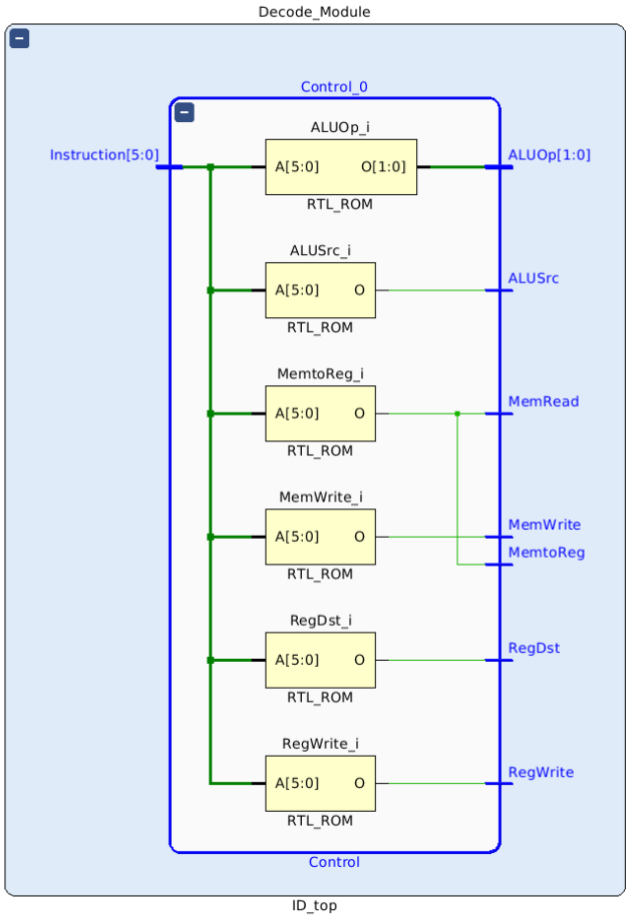

Fig 5: Control Signals Module

ii. Register File (32-bit*32-bit)

```
/*This module implements a 32x32-bit MIPS register file.*/

module RegFile(input CLK,
                    input RegWrite,
                    input [4:0] ReadRegister1,
                    input [4:0] ReadRegister2,
                    input [4:0] WriteRegister,
                    input [31:0] WriteData,
                    output [31:0] ReadData1,
                    output [31:0]ReadData2
                    )
reg [31:0] Regist [31:0];   //Internal rgisters
integer i;                      //Used for initializing

//All registers contain 0 initially (Only for Simulation)
initial
    begin
        for(i = 0; i < 32; i = i + 1)
            Regist[i] = 32'b0;
    end

always @(posedge CLK)
    if (RegWrite) Regist[WriteRegister] = WriteData;                //Write
 operation
    assign ReadData1 = ReadRegister1 ? Regist[ReadRegister1]:0; //Read oper
ation
    assign ReadData2 = ReadRegister2 ? Regist[ReadRegister2]:0; //Read oper
ation

endmodule
```

iii. ALU Control Module

```
/*This file implements a MIPS ALU control module.*/

module ALUControl(input wire [5:0] FUNCT,
                        input wire [1:0] OPCODE,
                        output reg [3:0] CTLOUT
                        );
reg [3:0] Function;

always @(*)
    begin
        Function=0;
        case(FUNCT[3:0])
            4'b0000: Function = 4'b0010;    //Add
            4'b0010: Function = 4'b0110;    //Sub
            4'b0100: Function = 4'b0000;    //And
            4'b0101: Function = 4'b0001;    //Or
            4'b0110: Function = 4'b1101;    //Nand
            4'b0111: Function = 4'b1100;    //Nor
            4'b1010: Function = 4'b0111;    //slt
            default: Function = 4'b0000;
        endcase
```

```
    end

always@(*)
    begin
        case(OPCODE)
            4'b0000: CTLOUT = 4'b0010;//Add
            4'b0010: CTLOUT = Function;
            default: CTLOUT = 0;
        endcase
    end
endmodule
```

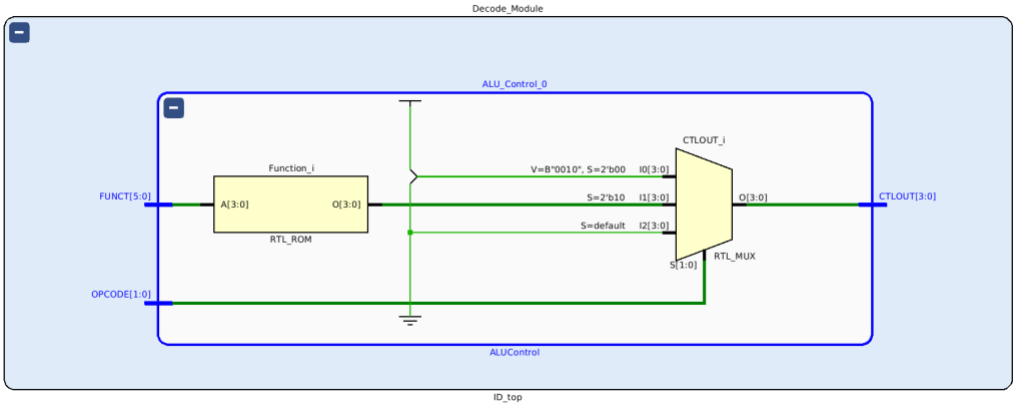

Fig 6: ALU Control Module

iv. Sign Extend Module

```
module SignExtend(input [15:0] a,
                          output [31:0] result);

assign result = {{16{a[15]}}, {a[15:0]}};

endmodule
```

v. 2-to-1 Multiplexer (5-bit)

```
module MUX2to1_5bit(D0, D1, OUT, SEL);
    input [4:0] D0, D1;
    input SEL;
    output [4:0] OUT;
    assign OUT = (SEL == 0) ? D0:D1;
endmodule
```

vi. 2-to-1 Multiplexer (32-bit)

```
module MUX2to1_32bit(D0, D1, OUT, SEL);
    input [31:0] D0, D1;
```

```
3.      input SEL;
4.      output [31:0] OUT;
5.      assign OUT = (SEL == 0) ? D0:D1;
6.  endmodule
```

vii. ID/EX Pipeline Register

```
1.  module ID_EX(
2.      input clk,
3.      input reset,
4.      input [102:0] data_in,
5.      output reg [102:0] data_out
6.      );
7.
8.      always@(posedge clk) begin
9.      if (reset)
10.         data_out=0;
11.     else
12.         data_out=data_in;
13.     end
14.
15. endmodule
```

3. Execute Module

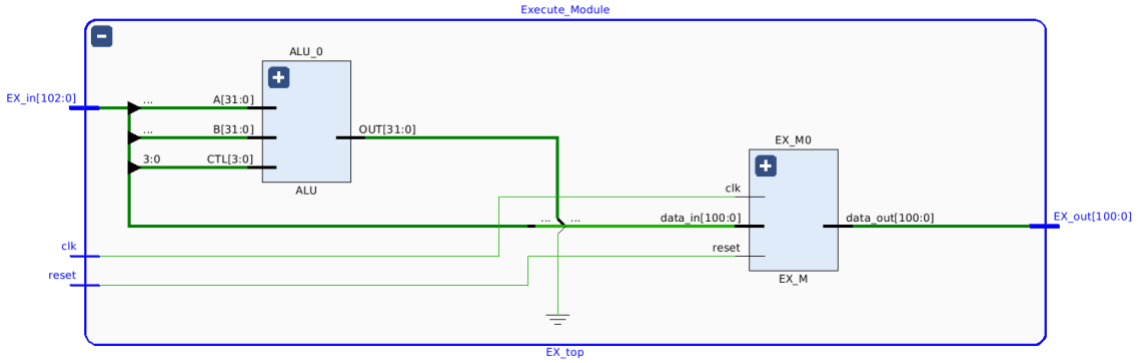

Fig 7: Execute Top Module

i. ALU

```
1.  /*This module implements an ALU.*/
2.
3.  module ALU(input [3:0] CTL,
4.                 input [31:0] A,
5.                 input [31:0] B,
6.                 output reg [31:0] OUT
7.                 );
8.
9.
10.     always @(*)
11.         begin
12.             case(CTL)
13.                 4'b0000: OUT = A & B;           //And
14.                 4'b0001: OUT = A | B;           //Or
15.                 4'b0010: OUT = A + B;           //Add
16.                 4'b0110: OUT = A - B;           //sub
17.                 4'b0111: OUT = (A < B) ? 1:0;    //Set less than
18.                 4'b1100: OUT = ~(A | B);        //Nor
19.                 4'b1101: OUT = ~(A & B);        //Nand
20.                 default: OUT = 0;
21.             endcase
22.         end
23. endmodule
```

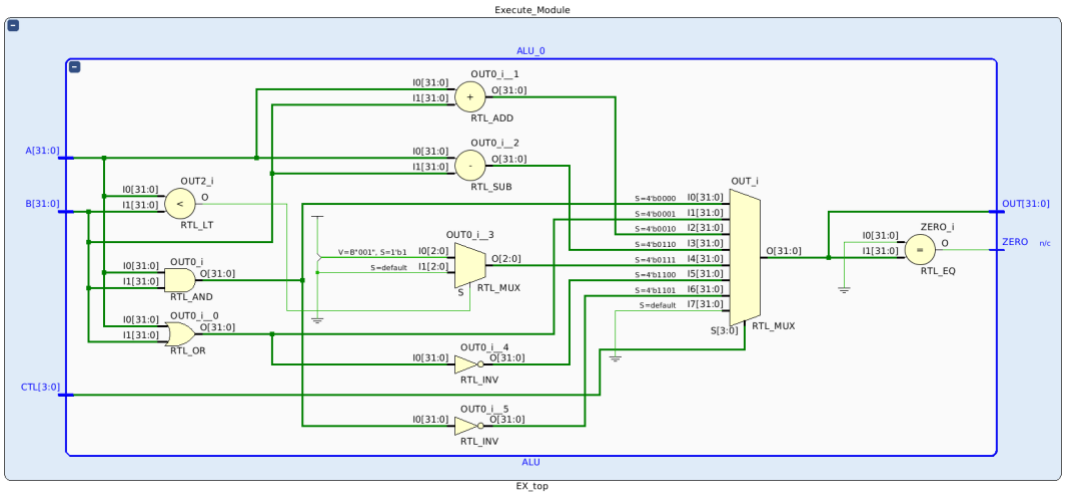

Fig 8: ALU Schematic

ii. EX/MEM Pipeline Register

```
1.  module EX_M(
2.      input clk,
3.      input reset,
4.      input [99:0] data_in,
5.      output reg [99:0] data_out
6.      );
7.
8.      always@(posedge clk) begin
9.      if (reset)
10.         data_out=0;
11.     else
12.         data_out=data_in;
13.     end
14.
15. endmodule
```

4. Memory Access Module

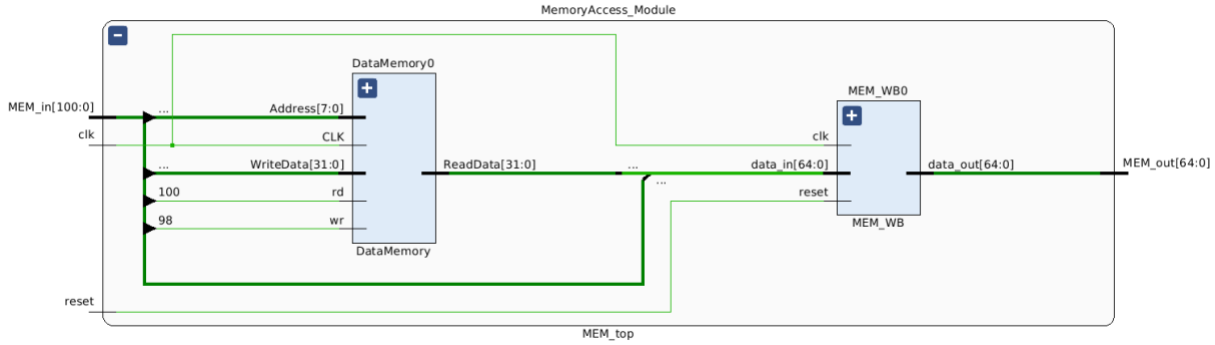

Fig 9: Memory Top Module

i. Data Memory (256 addresses, 32-bits each)

```
/*This file implements a MIPS data file.*/

module DataMemory(input wire CLK,
                          input wire [5:0] Address,
                          input wire rd,
                          input wire wr,
                          input wire [31:0] WriteData,
                          output wire [31:0] ReadData
                          );
reg [31:0] Mem [63:0]; //Internal registers
integer i;
    always @(posedge CLK)
        begin
            if(wr)
                begin
                    Mem[Address] = WriteData;
                end
        end

    assign ReadData = wr ? WriteData: Mem[Address][5:0];

endmodule
```

ii. MEM/WB Pipeline Register

```
module MEM_WB(
    input clk,
    input reset,
    input [64:0] data_in,
    output reg [64:0] data_out
    );

    always@(posedge clk) begin
```

```
    if (reset)
        data_out=0;
    else
        data_out=data_in;
    end

endmodule
```

5. Write Back Module

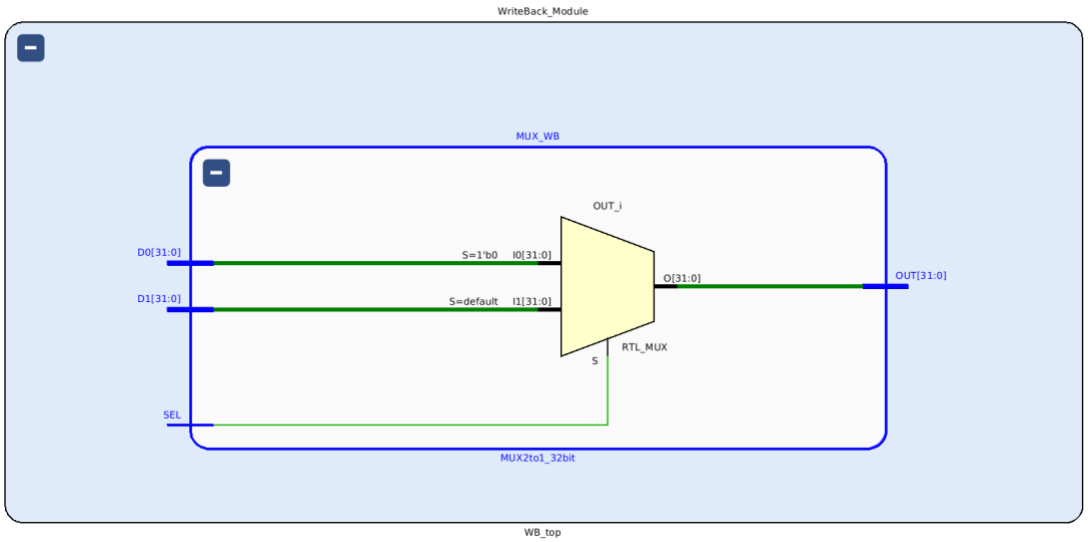

Fig 10: Write Back Module

User Constraints

```
## Clock signal
#set_property PACKAGE_PIN W5 [get_ports clk]
#set_property IOSTANDARD LVCMOS33 [get_ports clk]
#create_clock -add -name sys_clk_pin -period 1000000000.00 -
waveform {0 500000000} [get_ports clk]

## Switches
set_property PACKAGE_PIN V17 [get_ports clk]                  #Temporary clock
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property PACKAGE_PIN T1 [get_ports en]
set_property IOSTANDARD LVCMOS33 [get_ports en]
set_property PACKAGE_PIN R2 [get_ports reset]
set_property IOSTANDARD LVCMOS33 [get_ports reset]

## LEDs
set_property PACKAGE_PIN V19 [get_ports dout[0]]
set_property IOSTANDARD LVCMOS33 [get_ports dout[0]]
set_property PACKAGE_PIN W18 [get_ports dout[1]]
set_property IOSTANDARD LVCMOS33 [get_ports dout[1]]
set_property PACKAGE_PIN U15 [get_ports dout[2]]
set_property IOSTANDARD LVCMOS33 [get_ports dout[2]]
set_property PACKAGE_PIN U14 [get_ports dout[3]]
set_property IOSTANDARD LVCMOS33 [get_ports dout[3]]
set_property PACKAGE_PIN V14 [get_ports dout[4]]
set_property IOSTANDARD LVCMOS33 [get_ports dout[4]]
set_property PACKAGE_PIN V13 [get_ports dout[5]]
set_property IOSTANDARD LVCMOS33 [get_ports dout[5]]
set_property PACKAGE_PIN V3 [get_ports dout[6]]
set_property IOSTANDARD LVCMOS33 [get_ports dout[6]]
set_property PACKAGE_PIN W3 [get_ports dout[7]]
set_property IOSTANDARD LVCMOS33 [get_ports dout[7]]
```

Coefficient File

```
memory_initialization_radix=2;
memory_initialization_vector=00100000000010000000000000000101, 00100000000100100000000
00000110, 00000001000010010101100000100000, 10101100000010110000000000000000, 100011000
00011000000000000000000, 00000001000011000110100000100100, 00000001000011010111000000010
1010, 00000000000000000000000000000000, 00000000000000000000000000000000, 0000000000000
00000000000000000000, 00000000000000000000000000000000, 00000000000000000000000000000000
, 00000000000000000000000000000000, 00000000000000000000000000000000, 0000000000000000
00000000000000, 00000000000000000000000000000000, 00000000000000000000000000000000, 00
00000000000000000000000000000000, 00000000000000000000000000000000, 0000000000000000000
00000000, 00000000000000000000000000000000, 00000000000000000000000000000000, 000000
00000000000000000000000000, 00000000000000000000000000000000, 00000000000000000000000
0000000, 00000000000000000000000000000000, 00000000000000000000000000000000, 0000000000
00000000000000000000, 00000000000000000000000000000000, 00000000000000000000000000000
000, 00000000000000000000000000000000, 00000000000000000000000000000000, 0000000000000
00000000000000000, 00000000000000000000000000000000, 00000000000000000000000000000000,
 00000000000000000000000000000000, 00000000000000000000000000000000, 0000000000000000
0000000000000, 00000000000000000000000000000000, 00000000000000000000000000000000, 000
00000000000000000000000000000, 00000000000000000000000000000000, 00000000000000000000
0000000000, 00000000000000000000000000000000, 00000000000000000000000000000000, 000000
00000000000000000000000000, 00000000000000000000000000000000, 00000000000000000000000
000000, 00000000000000000000000000000000, 00000000000000000000000000000000, 0000000000
00000000000000000000, 00000000000000000000000000000000, 00000000000000000000000000000
00, 00000000000000000000000000000000, 00000000000000000000000000000000, 0000000000000
0000000000000000, 00000000000000000000000000000000, 00000000000000000000000000000000,
00000000000000000000000000000000, 00000000000000000000000000000000, 0000000000000000
0000000000000, 00000000000000000000000000000000, 00000000000000000000000000000000, 0000
00000000000000000000000000000;
```

Utilization Report