EC 551
Advanced Digital Design with Verilog and FPGAs

Lab 1 - Report

3-Stage 32-bit Pipelined Data Path

By
Aastha Anand
Runal Nair

TABLE OF CONTENTS

INTRODUCTION

OBJECTIVES

- The goal of this lab is to create a 3-stage 32-bit pipelined data path as shown below.

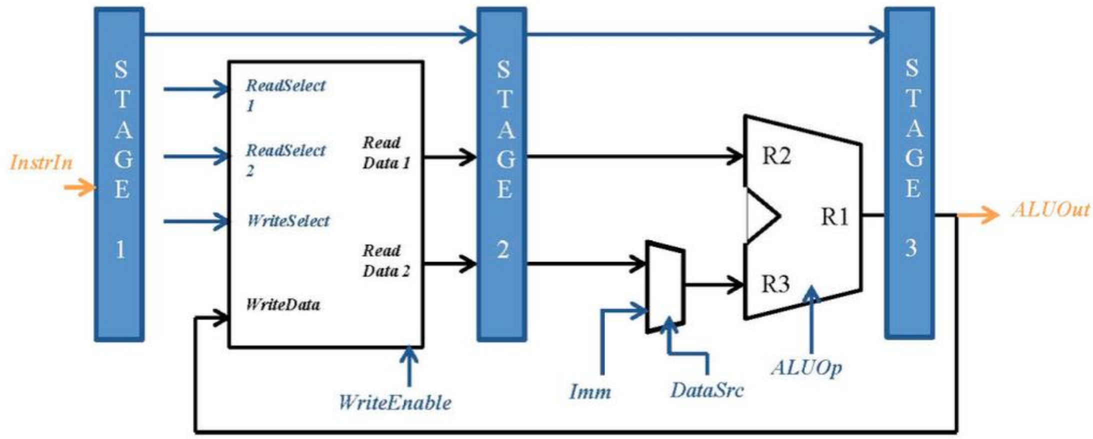


Fig1: The 3 stage 32-bit pipelined data path

- The interfaces to the top module, i.e., the module "call_modules" in my design are shown in orange in figure 1.
- There are 3 stages incorporated in the design of the data path and the design is hierarchical.
- In addition to the above diagram, control logic for routing of signals is incorporated in the design.
- For simplicity meaningful names and comments are added to the codes for each of the modules.
- There are 3 important parts to the design namely Arithmetic and Logical Unit (ALU), Multiplexer (MUX) and the read-write block (for memory).
- The instructions are I-type and R-type and as per the format the instructions are provided as input in the test bench "t_call_modules".

IMPLEMENTATION

- According to Figure 1, the 3-stage 32-bit pipelined data path in my design has the following modules in it: -
  a) call_modules
  b) stage1
  c) rd_wt_block
  d) stage2_3
  c) MUX
  d) ALU

- The test benches provided in the design are named as follows: -
  a) t_call_modules
  b) t_MUX
  c) t_ALU

- There are 3 important modes of operation namely fetching and decoding, execution and finally writing back.

- In the fetch mode, the instruction is fetched which maybe an I-type or R-type instruction with the following format: -

Type	**31**		**format (bits)**		**0**
R	opcode(6)	r1(5)	r2(5)	r3(5)	
I	opcode(6)	r1(5)	r2(5)	imm(16)	

- The most significant bit (31) is unused.
- The second most significant bit (30) is used to determine whether the instruction is Logical/Arithmetic (1) or other (0). This should always be set for this lab.
- The third most significant bit (29) is used to determine whether the instruction uses an immediate operand (1) or not (0), i.e., if this bit is set, the instruction is an I-type instruction.
- The least significant 3 bit of the opcode (28-26) represent the ALU opcode.

Fig2: Instruction format

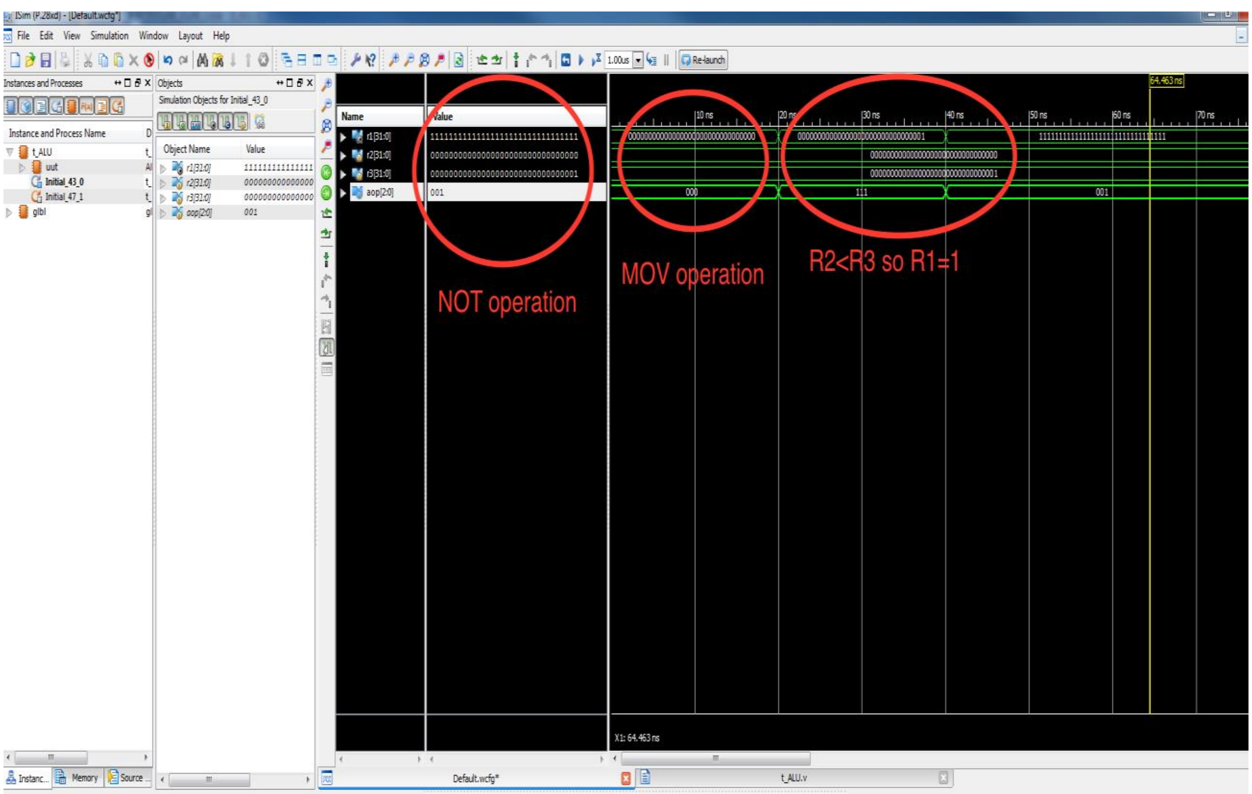
- The 32-bit instruction is decoded and is executed as per the instruction.

- The Write Enable provides the capability to write the data to the memory and this involves the writing back operation.

ARITHMETIC AND LOGICAL UNIT (ALU): -

```
module ALU(r2, r3, aop, r1);
input [31:0] r2; // 32 bit input to ALU
input [31:0] r3; // 32 bit input to ALU
input [2:0] aop; // ALU Operation
output reg[31:0] r1; //ALU Output
always @ (aop or r2 or r3) // Enter the block if either of the inputs are
provided
begin
case (aop) //type of ALU operation required
0: r1 = r2; //MOV operation
1: r1 = ~r2; // NOT operation
2: r1 = r2+r3; //ADD operation
3: r1 = r2-r3; //SUB operation
4: r1 = r2|r3; //OR operation
5: r1 = r2&r3; // AND operation
6: r1 = r2^r3; // XOR operation
7: r1 = (r2<r3) ? 1:0; //SLT Operation
endcase
end
endmodule
```

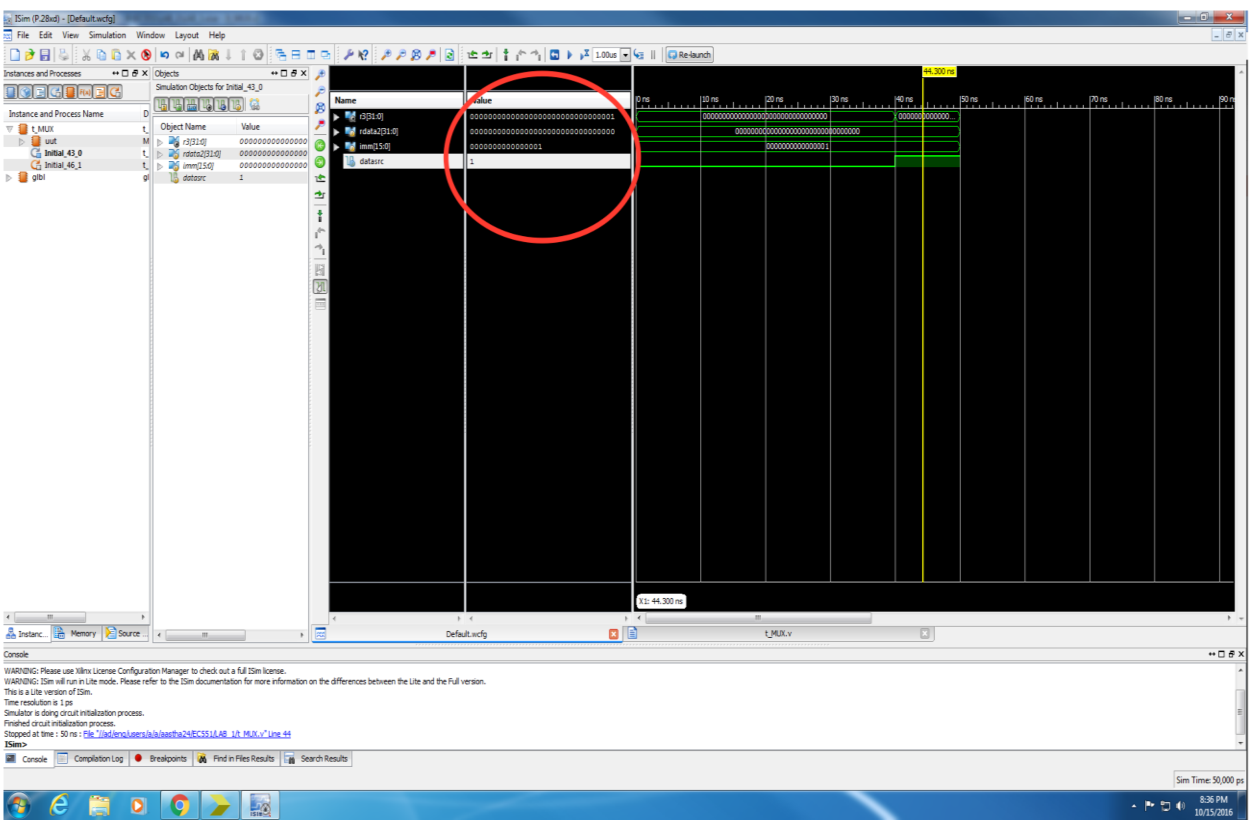
```
module t_ALU;

    // Inputs
    reg [31:0] r2;
    reg [31:0] r3;
    reg [2:0] aop;

    // Outputs
    wire [31:0] r1;

    // Instantiate the Unit Under Test (UUT)
    ALU uut (
        .r2(r2),
        .r3(r3),
        .aop(aop),
        .r1(r1)
    );

    initial begin
    #100 $finish;
    end

    initial begin
    r2 = 32'd0;
    r3 = 32'd1;
    aop =0;
    #20 aop=3'd7;
    #20 aop=3'd1;
    end

endmodule
```



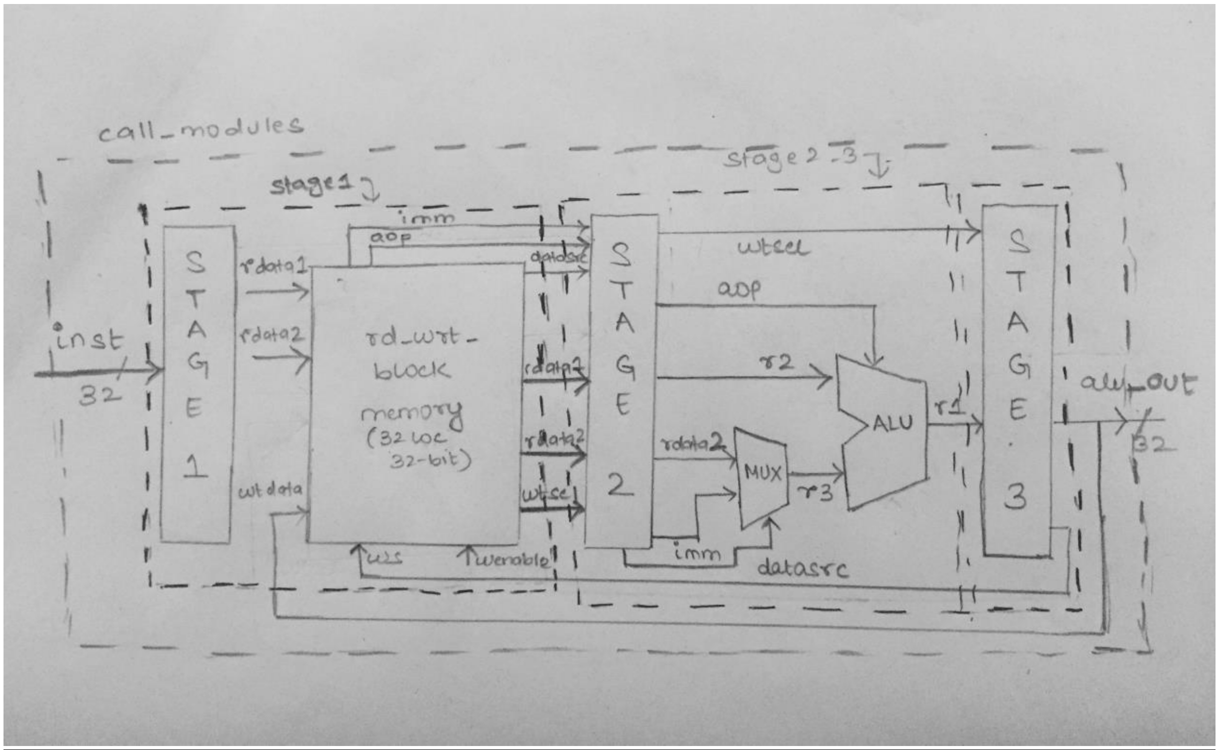
MULTIPLEXER (MUX): -

```
module MUX (rdata2,imm,datasrc,r3);
input [31:0] rdata2; // Read data 2 input to MUX
input [15:0] imm; // Immediate data to MUX
input datasrc; // Select line to MUX
output reg [31:0] r3; //Output of MUX

always @ (datasrc or rdata2 or imm)
begin
if (datasrc == 0) begin
r3 = rdata2;
end
else
begin
r3 = {16'd0,imm};
end
end
endmodule
```

```
module t_MUX;

    // Inputs
    reg [31:0] rdata2;
    reg [15:0] imm;
    reg datasrc;

    // Outputs
    wire [31:0] r3;

    // Instantiate the Unit Under Test (UUT)
    MUX uut (
        .rdata2(rdata2),
        .imm(imm),
        .datasrc(datasrc),
        .r3(r3)
    );

    initial begin
        #50 $finish;
        end
        initial begin
        rdata2 = 0;
        imm = 16'd1;
        datasrc = 0;
        #20 datasrc = 0;
        #20 datasrc = 1;
      end

endmodule
```



3 Stage 32-bit data path




```
module call_modules(clk,rst,inst,alu_out);
input clk,rst;
input [31:0] inst;
output [31:0] alu_out;

wire [31:0] rd1;
wire [31:0] rd2;
wire [15:0] i;
wire ds;
wire [4:0] ws_prev;
wire [4:0] wsel;
wire [2:0] ao;

stage1 sg1(.clk(clk),.rst(rst),.wtdata(alu_out),.wenable(1'd1),.inst(
inst),.wsprev(ws_prev),.rdata1(rd1),.rdata2(rd2),.wtsel(wsel),.imm(
i),.datasrc(ds),.aop(ao));
stage2_3 s23(.clk(clk),.rst(rst),.rdata1(rd1),.rdata2(rd2),.imm(
i),.datasrc(ds),.aop(ao),.alu_out(alu_out),.ws(wsel),.wtsel(ws_prev));

endmodule
```

Points of importance in the data path

- In the stage1 module, during the fetching and decoding the write select is provided as output to the next stage since in the next clock cycle a new write select is provided for the ALU output (alu_out) to be written to, hence in order to avoid data to be written to a wrong memory location, the wsprev has the data from the previous stage and wtsel has current instruction wtsel. This is an important point which is difficult to implement in the creation of the 3 stage 32-bit pipelined data path.

- The instantiations and the wire connections should be proper to avoid confusion and incorrect passage of data between the 3 stages.

- Timing is a very important part of the entire design process. In order to ensure correct result from the provided input instruction, the test bench must incorporate proper delay and timing.

RESULTS

```
module t_call_modules;

    // Inputs
    reg clk;
    reg rst;
    reg [31:0] inst;

    // Outputs
    wire [31:0] alu_out;
    integer i;

    // Instantiate the Unit Under Test (UUT)
    call_modules uut (
        .clk(clk),
        .rst(rst),
        .inst(inst),
        .alu_out(alu_out)
    );

always begin
        #1 clk = ~clk;
    end

    initial begin
        // Initialize Inputs
    clk = 0;
    rst = 1;
    #1 rst = 0;
    #1 rst = 1;
    inst = {2'b01,1'b1,3'b010,5'b00000,5'b00001,16'h9525};
    #4 inst = {2'b01,1'b1,3'b010,5'b00001,5'b00001,16'h9635};
    for(i = 0;i < 7;i = i + 1)
    begin
        #3 inst = {2'b01,1'b0,i,5'b00010,5'b00000,5'b00001,11'd0};
    end
    #2 inst = 32'h00000000;
    end
endmodule
```

Check for all the ALU operations