# HW2: Register File and ALU

Summer 2021

Thiha Myint

Demo

https://www.youtube.com/watch?v=4BjudBxTys0

## Description

In this lab, the RAM design and ALU machine was written on the SystemVerilog. That was part of the circuit required to produce a OTTER machine. From the lab manual, we create a 32x32 register file and write a simulation file for 8 different address location. The input is from the memory file that was connected prior to register circuit. Then, the addr1 and adr2 read from that address. The output rs2 and rs2 are linked into ALU circuit that can do calculations. The eleven instructions set for calculations are provided by lab manual.

## Simulation Waveform

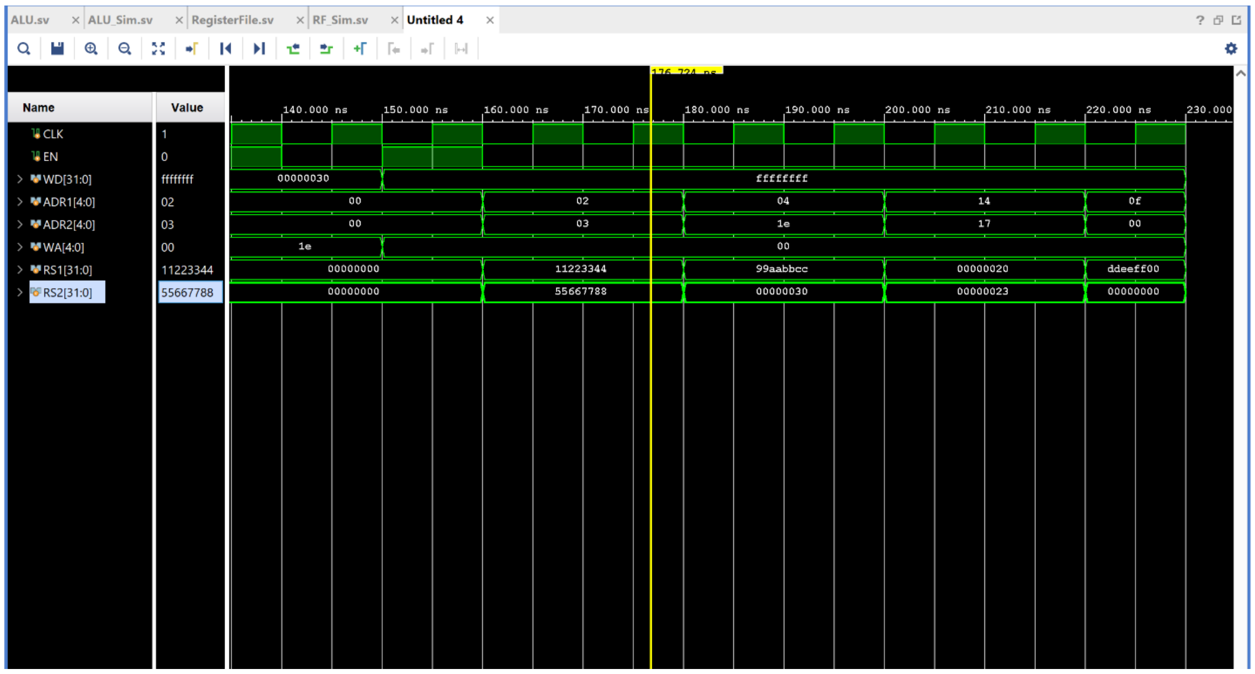

Fig 1: Simulation for RFSim

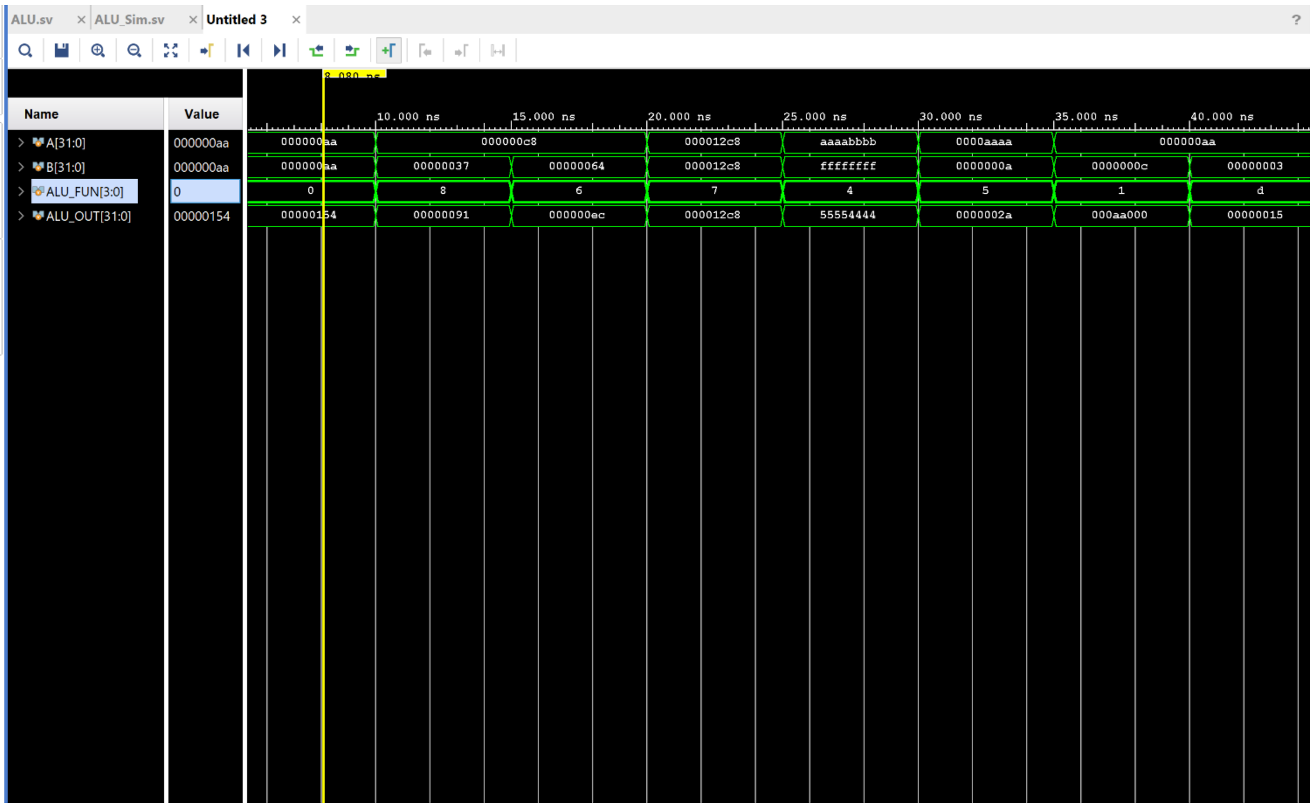
ALU.sv
ALU_Sim.sv
Untitled 3
8.080 ns
Name
Value
A[31:0] 000000aa
B[31:0] 000000aa
ALU_FUN[3:0] 0
ALU_OUT[31:0] 00000154
10.000 ns
15.000 ns
20.000 ns
25.000 ns
30.000 ns
35.000 ns
40.000 ns
000000aa 000000c8 000012c8 aaaabbbb 0000aaaa 000000aa
000000aa 00000037 00000064 000012c8 ffffffff 0000000a 0000000c 00000003
0 8 6 7 4 5 1 d
00000154 00000091 000000ec 000012c8 55554444 0000002a 000aa000 00000015

Fig 2 : Simulation for ALU_Sim

## Code

```
`timescale 1ns / 1ps
//create a 32x32 bits memory
//Address size = log2(32) = 5
//Data size = 32 bits
module RegisterFile(
    input CLK,
    input EN,
    input [31:0] WD,
    input [4:0] ADR1,
    input [4:0] ADR2,
    input [4:0] WA,
    output logic [31:0] RS1,
    output logic [31:0] RS2
    );

    logic [31:0] memory [0:31];

    //initialize memory
    initial
    begin
        int i;
        for (i=0; i<32; i++) //(initialization, check, action per loop)
            memory [i] <= 0;
    end

    //describe memory
    always_ff@ (posedge CLK)
    begin
        //sychronous write
        if(EN == 1 && WA !=0)
            memory[WA] <= WD;
         else
            memory <= memory;
    end

  //asychronous read
    assign RS1 = memory[ADR1];
    assign RS2 = memory [ADR2];


endmodule
```

---

```
`timescale 1ns / 1ps

module RFSim( );

    logic CLK;
    logic EN;
    logic [31:0] WD;
    logic [4:0] ADR1;
    logic [4:0] ADR2;
    logic [4:0] WA;
    logic [31:0] RS1;
    logic [31:0] RS2;

RegisterFile RF_test (.*);

always
    begin
        CLK = 0; #5; CLK = 1; #5;
    end

initial
    begin

        ADR1 <= 0;
        ADR2 <= 0;
        WD <= 0;
        WA <= 0;
        EN <= 0;

        // Write Test Number 1
        #10
        WD <= 32'h11223344;
        WA <= 2;
        EN <= 1;
        #10
        EN <= 0;
```

```
// Write Test Number 2
#10
WD <= 32'h55667788;
WA <= 3;
EN <= 1;
#10
EN <= 0;

// Write Test Number 3
#10
WD <= 32'h99AABBCC;
WA <= 4;
EN <= 1;
#10
EN <= 0;

// Write Test Number 4
#10
WD <= 32'hDDEEFF00;
WA <= 15;
EN <= 1;
#10
EN <= 0;

// Write Test Number 5
#10
WD <= 32'h00000020;
WA <= 20;
EN <= 1;
#10
EN <= 0;

// Write Test Number 6
#10
WD <= 32'h00000023;
WA <= 23;
EN <= 1;
#10
EN <= 0;

// Write Test Number 7
#10
WD <= 32'h00000030;
WA <= 30;
EN <= 1;
#10
EN <= 0;

// Write Test Number 8
#10
WD <= 32'hFFFFFFFF;
WA <= 0;
EN <= 1;
#10
EN <= 0;
```

```
        $display("Memory Address\t:\tWritten Value\t:\tRead Value");
        // Now read from memory
        ADR1 <= 2;
        ADR2 <= 3;
        #10
        $display("%d\t\t:\t0x%08x\t\t:\t0x%08x",2,32'h11223344,RS1);
        $display("%d\t\t:\t0x%08x\t\t:\t0x%08x",3,32'h55667788,RS2);
        #10

        ADR1 <= 4;
        ADR2 <= 30;
        #10
        $display("%d\t\t:\t0x%08x\t\t:\t0x%08x",4,32'h99AABBCC,RS1);
        $display("%d\t\t:\t0x%08x\t\t:\t0x%08x",30,32'h00000030,RS2);

        #10

        ADR1 <= 20;
        ADR2 <= 23;
        #10
        $display("%d\t\t:\t0x%08x\t\t:\t0x%08x",20,32'h00000020,RS1);
        $display("%d\t\t:\t0x%08x\t\t:\t0x%08x",23,32'h00000023,RS2);
        #10

        ADR1 <= 15;
        ADR2 <= 0;
        #10
        $display("%d\t\t:\t0x%08x\t\t:\t0x%08x",15,32'hDDEEFF00,RS1);
        $display("%d\t\t:\t0x%08x\t\t:\t0x%08x",0,32'hFFFFFFFF,RS2);
        $finish;

     end
endmodule
```

---

```
`timescale 1ns / 1ps

module ALU(
    input [31:0] A,
    input [31:0] B,
    input [3:0] ALU_FUN,
    output logic signed [31:0] ALU_OUT);


    bit [4:0]x;

    always @ (*)
    assign x=B[4:0];

    always_comb
    begin
    case (ALU_FUN)
       4'b0000 : ALU_OUT=A + B;
       4'b1000 : ALU_OUT=A - B;
       4'b0110 : ALU_OUT=A | B;
       4'b0111 : ALU_OUT=A & B;
```

```
        4'b0100 : ALU_OUT=A ^ B;
        4'b0101 : ALU_OUT=A >> x;
        4'b0001 : ALU_OUT=A << x;
        4'b1101 : ALU_OUT=A >>> x;
        4'b0010 : ALU_OUT=A < B? 1 : 0;
        4'b0011 : ALU_OUT=$unsigned(A) < $unsigned(B)? 1 : 0;
        4'b1001 : ALU_OUT= A;
        default : ALU_OUT= 0;

        endcase
        end
endmodule
```

---

```
`timescale 1ns / 1ps

module ALU_Sim();
    logic [31:0] A;
    logic [31:0] B;
    logic [3:0] ALU_FUN;
    logic signed [31:0] ALU_OUT;

    ALU ALU_test (.*);

    ALU DUT(A,B,ALU_FUN,ALU_OUT);

    initial
    begin
    $monitor($time," ADL_OUT=M,ALU_FUN=M,A=M,B=%b",ALU_OUT,ALU_FUN,A,B);
    #5 A=32'hAA; B=32'hAA; ALU_FUN=4'b0000;
    #5 A=32'hC8; B=32'h37; ALU_FUN=4'b1000;
    #5 A=32'hC8; B=32'h64; ALU_FUN=4'b0110;
    #5 A=32'h12C8; B=32'h12C8; ALU_FUN=4'b0111;
    #5 A=32'hAAAABBBB; B=32'hFFFFFFFF; ALU_FUN=4'b0100;
    #5 A=32'hAAAA; B=32'h0A; ALU_FUN=4'b0101;
    #5 A=32'hAA; B=32'h0C; ALU_FUN=4'b0001;
    #5 A=32'hAA; B=32'h03; ALU_FUN=4'b1101;
    #5 A=-20; B=02; ALU_FUN=4'b1101;
    #5 A=32'hAA; B=32'hAA; ALU_FUN=4'b0010;
    #5 A=-5; B=-10; ALU_FUN=4'b0010;
    #5 A=32'hAA; B=55; ALU_FUN=4'b0011;
    #5 A=32'hABC1231; B=32'h12345678; ALU_FUN=4'b1001;
    end

    initial
    #500 $finish;

endmodule
```