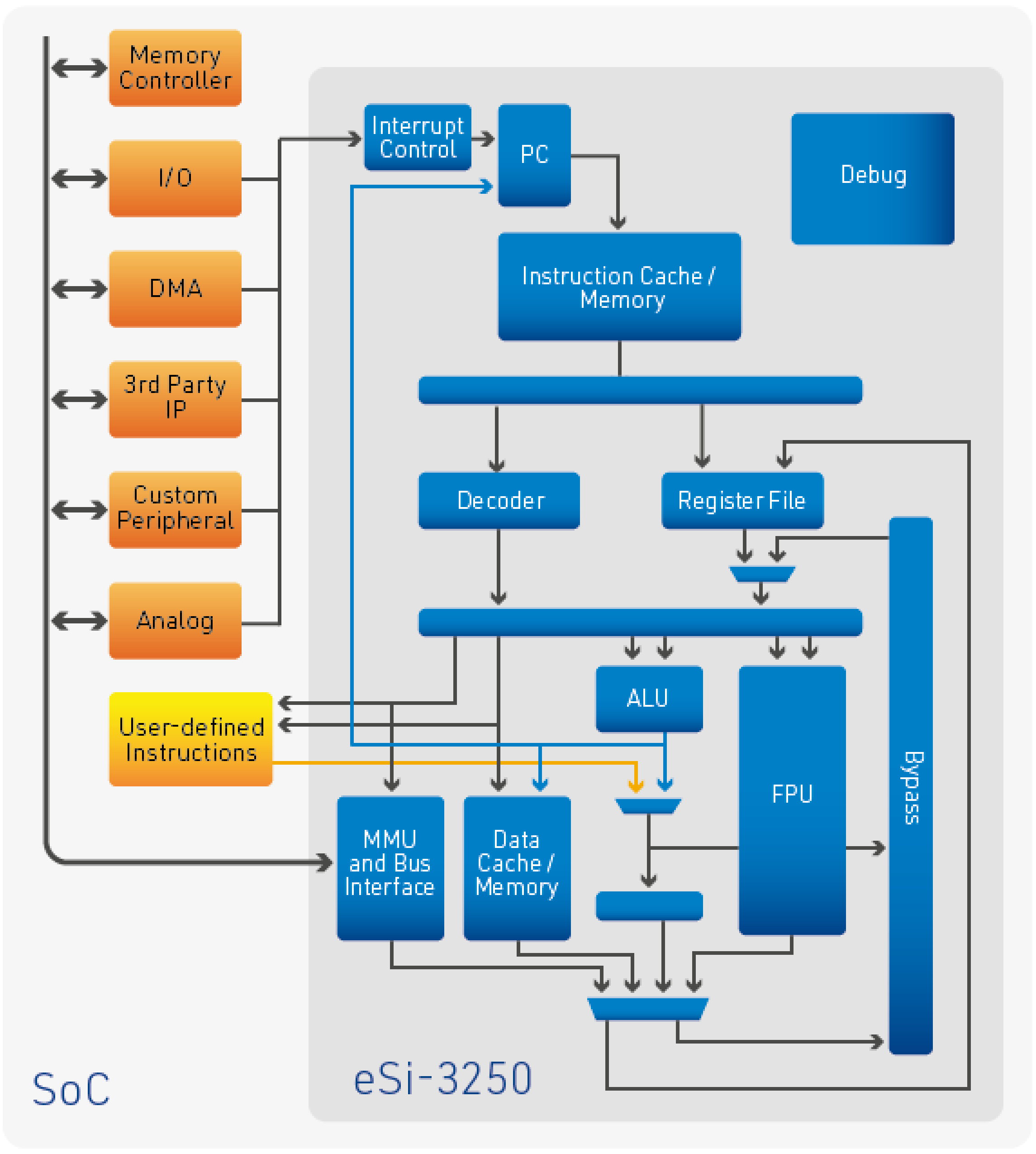
RISC -V 32I

VLSI Design Internship

Project name: RISC V 32I

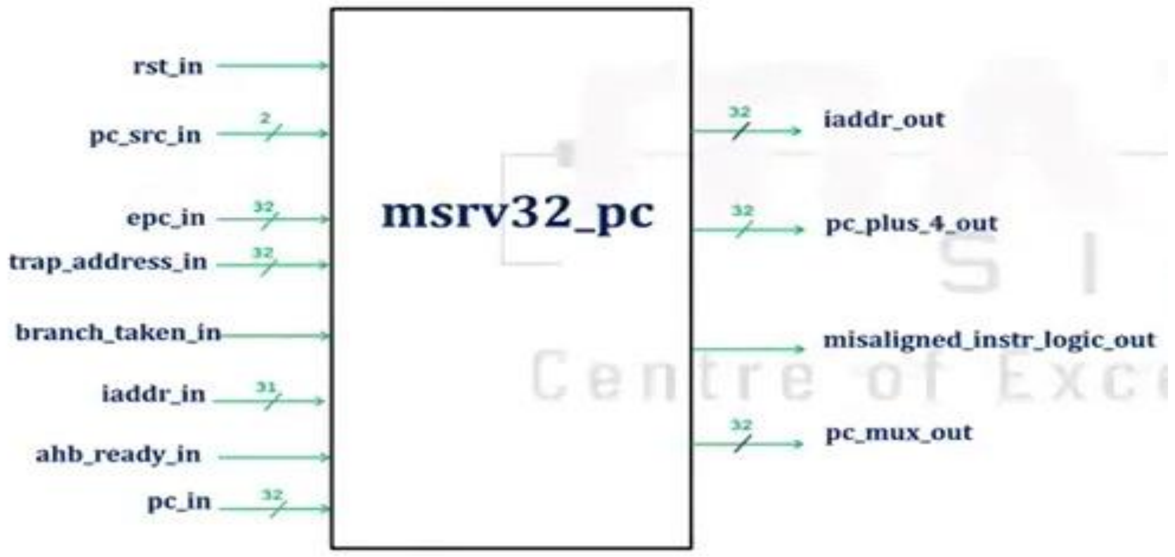


INDEX

SL NO	Model name	Module name	Page no
1)	Pc mux	msrv32_pc	03 - 11
2)	Reg block 1	msrv32_reg_block_1	12 - 18
3)	Immediate Generator	msrv32_imm_generator	19 - 26
4)	Immediate adder	msrv32_immediate_adder	27 - 33
5)	Write Enable Generator	msrv2_wr_en_generator	34 - 40
6)	Instruction Mux	msrv32_instruction_mux	41 - 47
7)	Integer File	msrv32_integer_file	**48 - 60**
8)	Branch Unit	msrv32_branch_unit	61 - 67
9)	Store unit	msrv32_store_unit	68 - 76
10)	Load Unit	msrv32_load_unit	77 - 84
11)	Reg block 2	msrv32_reg_block_2	85 - 95
12)	ALU	msrv32_alu	96 - 103
13)	Wb Mux Selection out	msrv32_wb_mux_sel_unit	104 – 110
14)	Decoder	msrv32_decoder	111 - 121
15)	RISC V 32I - TOP MODULE	msrv32_top	122 - 132

PC MUX

Block Diagram



Block Pins

1. rst_in : in this code used as Power on Reset pin.
2. pc_src_in : is used for current state of the core pin.
3. ahb_ready_in : is used active high signal used to update the i_addr_out.
4. epic_in : used to trap_return address from csr file.
5. trap_address_in : is use Address when interrupt occurs (from csr file).
6. branch_taken_in : used to Indication when branch instruction occurs (from branch unit).
7. iaddr_in : Adress with addition of immediate value (from immediate adder).
8. pc_plus_4_out : Registered for stage 2 operation (to Reg block 2).
9. pc_mux_out : similar to pc_out but not registered (to Reg block 1).

Explanation – PC Mux - msrv32_pc

The msrv32_pc module is a program counter unit for a RISC-V processor. It takes several inputs, including reset, program counter source, exception program counter, trap address, branch signal, instruction address, AHB ready signal, and current program counter value. Based on the inputs and the state of control signals, it calculates the next program counter value (next_pc) and outputs the updated instruction address, the program counter incremented by 4 (pc_plus_4_out), and a misaligned instruction logic signal. The module uses a multiplexer to select the correct program counter source (pc_mux_out), which could be a boot address, exception address, trap address, or the next program counter value. It also handles misaligned instruction detection by checking the least significant bit of the next program counter value.

Verilog Code

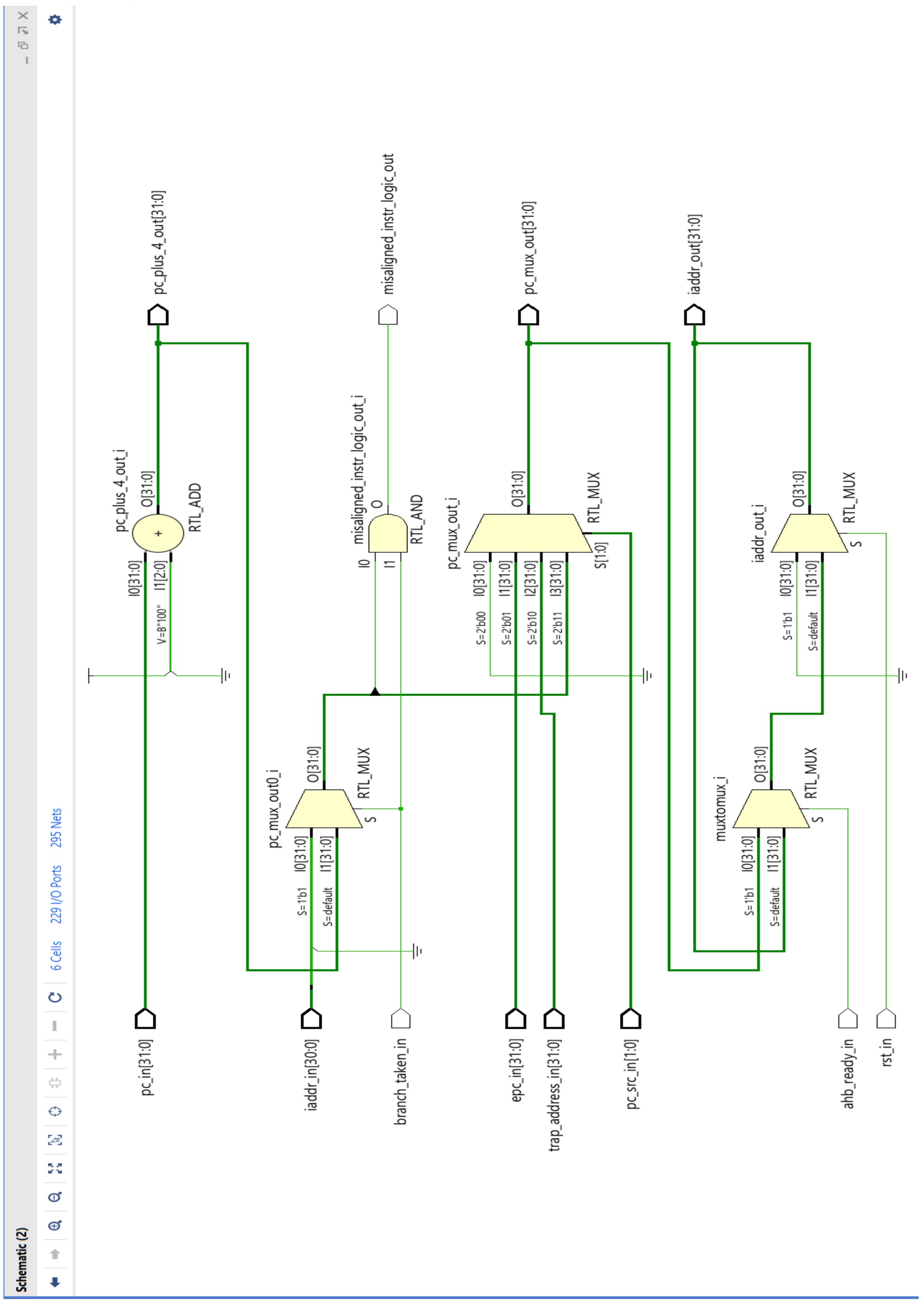
Design code (RTL) – PC Mux - msrv32_pc

```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////
/////////////////////////////
// Company: maven silicon - VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 21.06.2024 17:06:31
// Design Name: PC Mux - msrv32_pc
// Module Name: msrv32_pc
// Project Name: Achyut RISC - V 32I
// Target Devices:
// Tool Versions: Achyut's Vivado
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////
/////////////////////////////


module msrv32_pc(rst_in, pc_src_in, epc_in,
trap_address_in,branch_taken_in,iaddr_in,
ahb_ready_in,pc_in,iaddr_out,pc_plus_4_out,misaligned
_instr_logic_out,pc_mux_out);
input rst_in;
input [1:0] pc_src_in;
input [31:0] epc_in;
input [31:0] trap_address_in;
input branch_taken_in;
input [30:0] iaddr_in;
input ahb_ready_in;
input [31:0] pc_in;


output [31:0] iaddr_out;
output [31:0] pc_plus_4_out;
output misaligned_instr_logic_out;
output reg [31:0] pc_mux_out;
//wire iaddr_in_concat;
reg [31:0] next_pc;
wire [31:0] muxtomux;
reg [31:0] boot_address;
//assign iaddr_in_concat = {iaddr_in[30:0],1'b0};
assign pc_plus_4_out = pc_in[31:0] +4;
//next_pc = branch_taken_in ? iaddr_in_concat :
pc_plus_4_out;
always @(*)
begin
    next_pc = branch_taken_in ? {iaddr_in[30:0],1'b0}
: pc_plus_4_out;
    case(pc_src_in)
        2'b00 : pc_mux_out <= boot_address;
        2'b01: pc_mux_out <= epc_in;
        2'b10 : pc_mux_out <= trap_address_in;
        2'b11 : pc_mux_out <= next_pc;
    endcase
end
//assign pc_mux_out =(pc_src_in==2'b00)
?boot_address:
//     (pc_src_in==2'b01)? epc_in:
//    (pc_src_in==2'b10)? trap_address_in: next_pc;
assign muxtomux = (ahb_ready_in)? pc_mux_out
:iaddr_out;
assign iaddr_out = (rst_in)? boot_address:muxtomux;

assign misaligned_instr_logic_out = next_pc[0] &
branch_taken_in;
endmodule
```

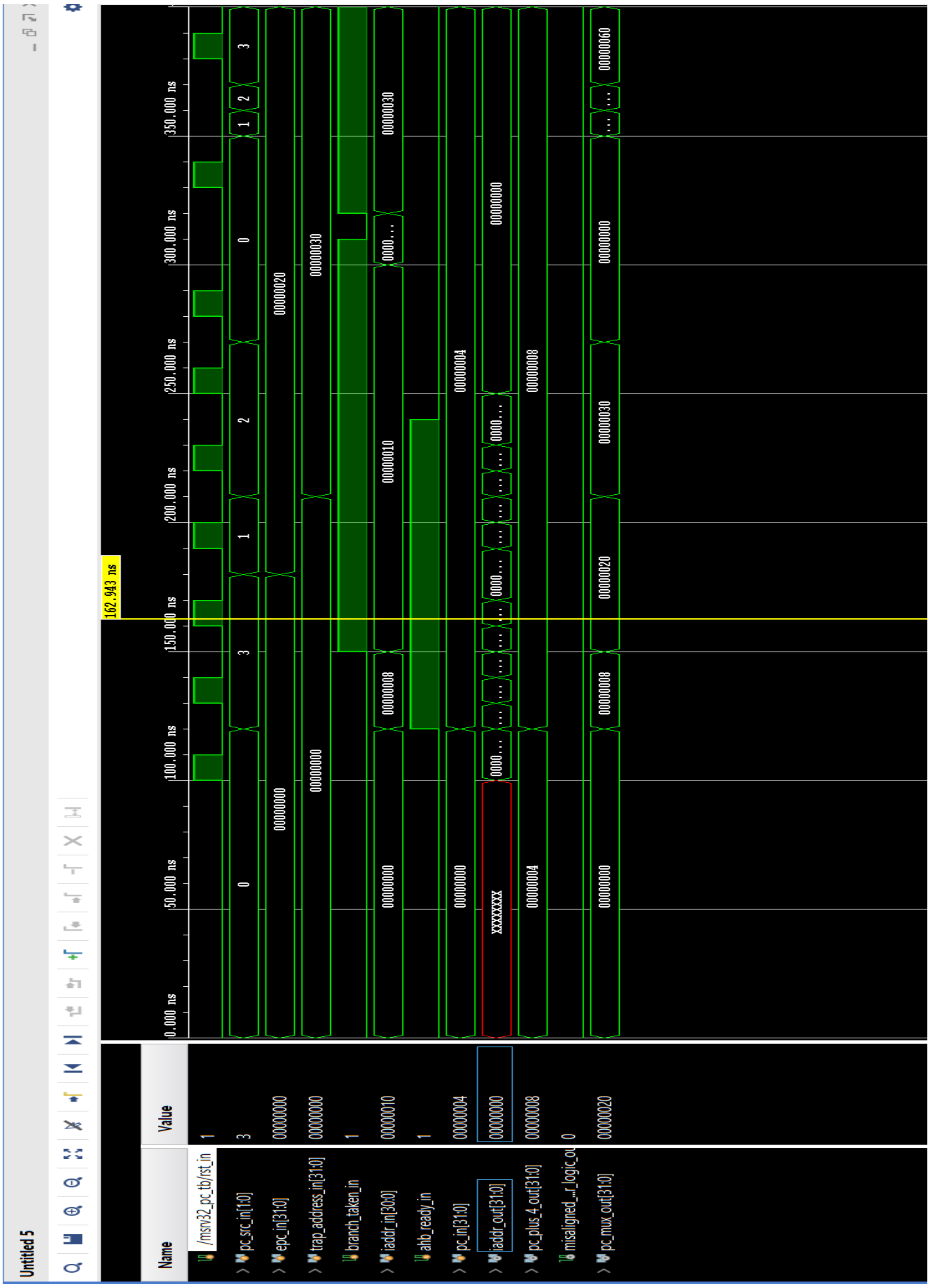
Testbench / Simulation code - PC Mux - msrv32_pc_tb

```
`timescale 1ns / 1ps

module msrv32_pc_tb;

  // Inputs
  reg rst_in;
  reg [1:0] pc_src_in;
  reg [31:0] epc_in;
  reg [31:0] trap_address_in;
  reg branch_taken_in;
  reg [30:0] iaddr_in;
  reg ahb_ready_in;
  reg [31:0] pc_in;

  // Outputs
  wire [31:0] iaddr_out;
  wire [31:0] pc_plus_4_out;
  wire misaligned_instr_logic_out;
  wire [31:0] pc_mux_out;

  // Instantiate the Unit Under Test (UUT)
  msrv32_pc uut (
    .rst_in(rst_in),
    .pc_src_in(pc_src_in),
    .epc_in(epc_in),
    .trap_address_in(trap_address_in),
    .branch_taken_in(branch_taken_in),
    .iaddr_in(iaddr_in),
    .ahb_ready_in(ahb_ready_in),
    .pc_in(pc_in),
    .iaddr_out(iaddr_out),
    .pc_plus_4_out(pc_plus_4_out),

.misaligned_instr_logic_out(misaligned_instr_logic_ou
t),
    .pc_mux_out(pc_mux_out)
  );

  initial begin
    // Initialize Inputs
    rst_in = 0;
    pc_src_in = 2'b00;
    epc_in = 32'h00000000;
    trap_address_in = 32'h00000000;
    branch_taken_in = 0;
    iaddr_in = 31'h00000000;
    ahb_ready_in = 0;
    pc_in = 32'h00000000;

    // Wait for global reset to finish
    #100;

    // Test Case 1: Reset active
    rst_in = 1;
    #10;
    rst_in = 0;
    #10;

    // Test Case 2: Normal operation with no branch
    pc_in = 32'h00000004;
    pc_src_in = 2'b11; // Select next_pc
    branch_taken_in = 0;
    iaddr_in = 31'h00000008;
    ahb_ready_in = 1;
    #10;
    rst_in = 1;
    #10;
    rst_in = 0;
    #10;

    // Test Case 3: Branch taken
    branch_taken_in = 1;
    iaddr_in = 31'h00000010;
    #10;
    rst_in = 1;
    #10;
    rst_in = 0;
    #10;

    // Test Case 4: Select epc
    pc_src_in = 2'b01;
    epc_in = 32'h00000020;
    #10;
    rst_in = 1;
    #10;
    rst_in = 0;
    #10;

    // Test Case 5: Select trap address
    pc_src_in = 2'b10;
    trap_address_in = 32'h00000030;
    #10;
    rst_in = 1;
    #10;
    rst_in = 0;
    #10;

    // Test Case 6: ahb_ready_in is low
    ahb_ready_in = 0;
    #10;
    rst_in = 1;
    #10;
    rst_in = 0;
    #10;

    // Additional Test Cases
    // Test Case 7: Boot address initialization
    $display("Test Case 7: Boot address
initialization");
    pc_src_in = 2'b00; // Select boot address
    #10;
    rst_in = 1;
    #10;
    rst_in = 0;
    #10;

    // Test Case 8: Multiple branches
    $display("Test Case 8: Multiple branches");
    branch_taken_in = 1;
    iaddr_in = 31'h00000020;
    #10;
    branch_taken_in = 0;
    #10;
    branch_taken_in = 1;
    iaddr_in = 31'h00000030;
    #10;
    rst_in = 1;
    #10;
    rst_in = 0;
    #10;
    // Test Case 9: Switching between sources
    $display("Test Case 9: Switching between 
sources");
    pc_src_in = 2'b01; // Select epc
    #10;
    pc_src_in = 2'b10; // Select trap address
    #10;
    pc_src_in = 2'b11; // Select next_pc
    #10;
    rst_in = 1;
    #10;
    rst_in = 0;
    #10;

    $finish;
  end

endmodule
```

RTL View / Micro Architecture – PC Mux – msrv32_pc



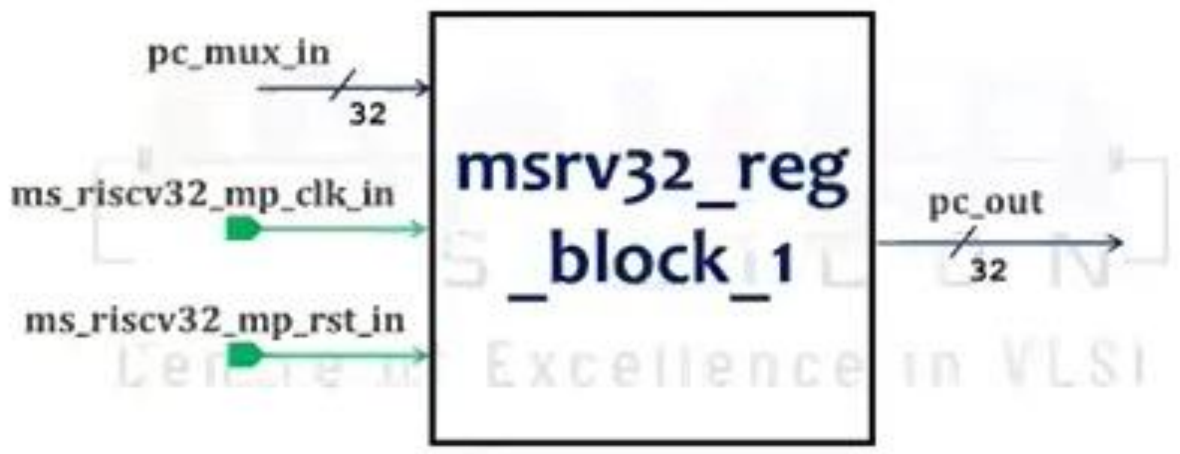
Wave form – PC Mux - msrv32_pc



Name	Value
/msrv32_pc_tb/rst_in	1
pc_src_in[1:0]	3
epc_in[31:0]	00000000
trap_address_in[31:0]	00000000
branch_taken_in	1
iaddr_in[30:0]	00000010
ahb_ready_in	1
pc_in[31:0]	00000004
iaddr_out[31:0]	00000000
pc_plus_4_out[31:0]	00000008
misaligned_...r_logic_ou	0
pc_mux_out[31:0]	00000020

Reg Block 1

Block Diagram



Explanation – Reg Block 1 – msrv32_reg_block_1

The block receives three inputs: pc_mux_in (32-bit data for the PC), ms_riscv32_mp_clk_in (clock signal), and ms_riscv32_mp_rst_in (reset signal).The clock input ms_riscv32_mp_clk_in ensures that the program counter is updated in sync with the processor's clock cycles. The reset input ms_riscv32_mp_rst_in initializes the program counter to a known state, typically zero, to ensure proper start-up and reset behavior. The output pc_out is the current value of the program counter, providing the address of the next instruction to be executed by the processor.

Block Pins

1. pc_mux_in - 32-bit input for the program counter value.
2. ms_riscv32_mp_clk_in - Clock input for synchronization.
3. ms_riscv32_mp_rst_in - Reset input for initializing the register.
4. pc_out - 32-bit output representing the current value of the program counter.

Verilog Code

Design Source – Reg Block 1 - msrv32_reg_block_1

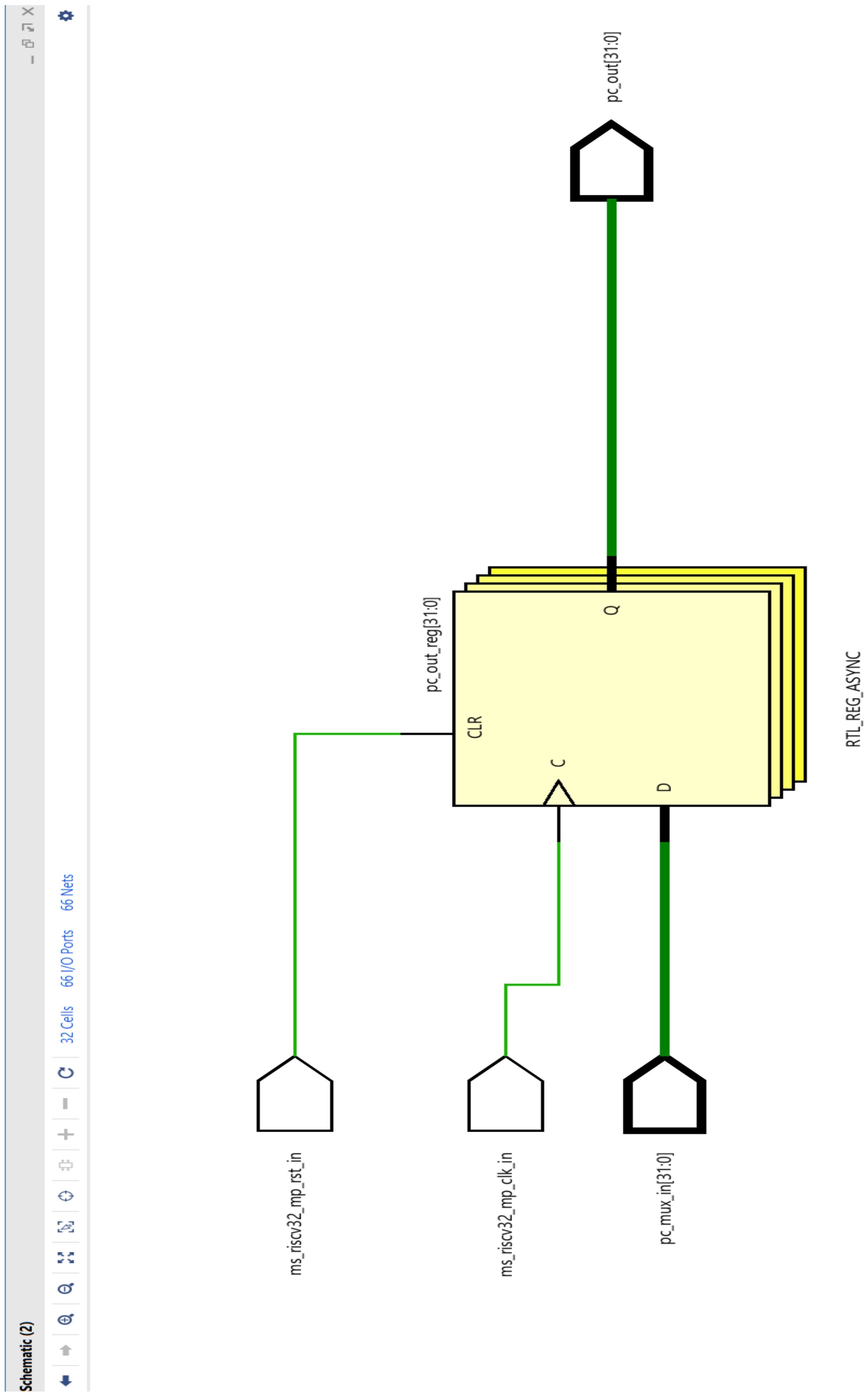
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////
/////////////////////////
// Company: Maven Silicon - VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 25.06.2024 20:00:04
// Design Name: Reg Block 1
// Module Name: msrv32_reg_block_1
// Project Name:  RISC -V 32I
// Target Devices:
// Tool Versions: Achyut's vivado
// Description:
// Design file
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////
/////////////////////////


module
msrv32_reg_block_1(pc_mux_in,ms_riscv32_mp_clk_in,ms_risc
v32_mp_rst_in,pc_out);
input [31:0] pc_mux_in;
input ms_riscv32_mp_clk_in;
input ms_riscv32_mp_rst_in;
output reg [31:0] pc_out;
always @(posedge ms_riscv32_mp_clk_in or posedge
ms_riscv32_mp_rst_in)
begin
        if(ms_riscv32_mp_rst_in)
            pc_out <=32'h00000000;
        else
            pc_out <= pc_mux_in;
end

endmodule
```

Source / Testbench code– Reg Block 1 - msrv32_reg_block_1

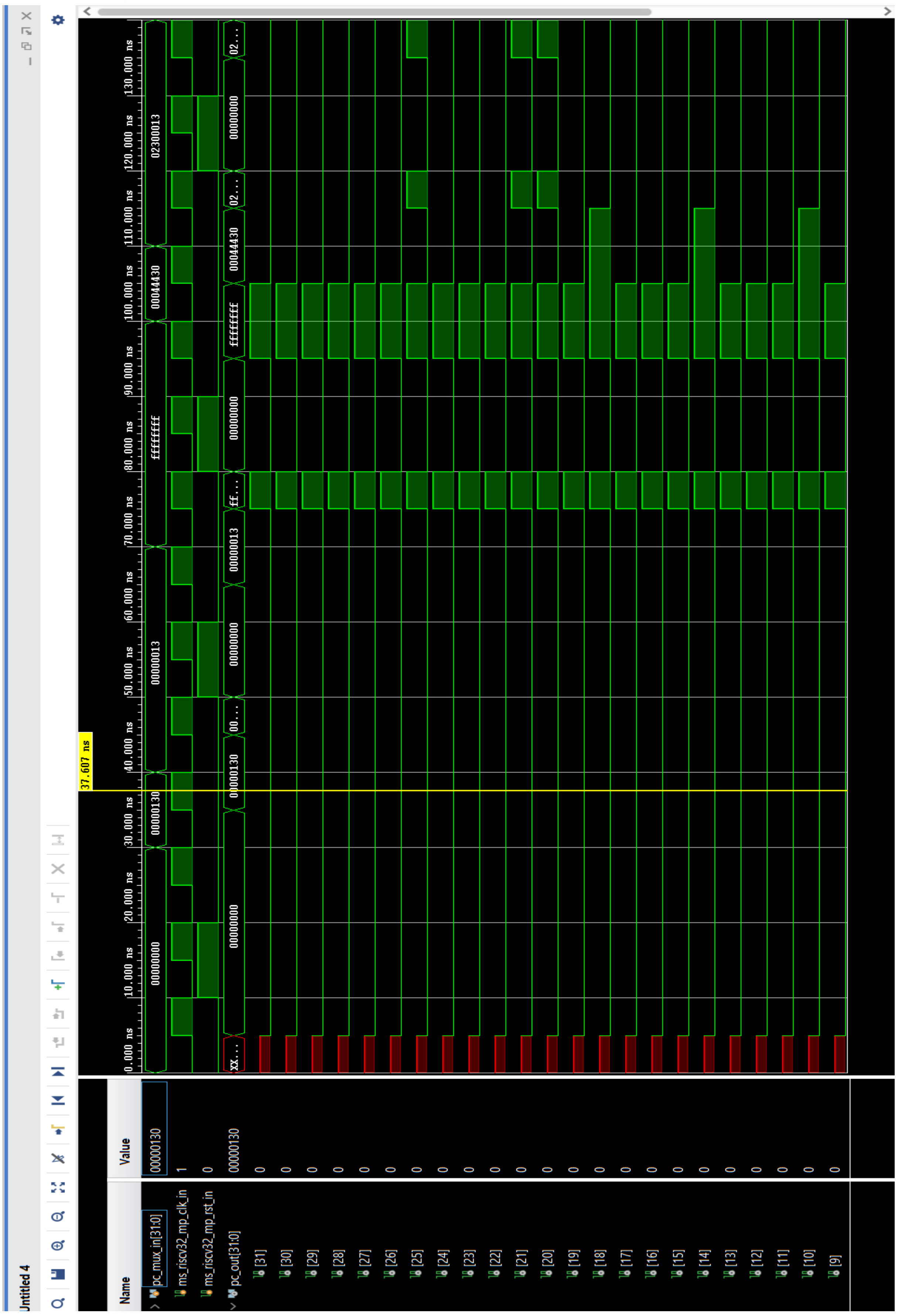
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////
/////////////////////////////
// Company: Maven Silicon - VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 27.06.2024 14:17:43
// Design Name: Reg Block 1
// Module Name: msrv32_reg_block_1_tb
// Project Name: RISC- V 32I
// Target Devices:
// Tool Versions: Achyut's vivado
// Description:
// Testbench code
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////
/////////////////////////////


module msrv32_reg_block_1_tb;
// Inputs
    reg [31:0] pc_mux_in;
    reg ms_riscv32_mp_clk_in;
    reg ms_riscv32_mp_rst_in;

    // Outputs
    wire [31:0] pc_out;

msrv32_reg_block_1 uut (
        .pc_mux_in(pc_mux_in),
        .ms_riscv32_mp_clk_in(ms_riscv32_mp_clk_in),
        .ms_riscv32_mp_rst_in(ms_riscv32_mp_rst_in),
        .pc_out(pc_out)
    );
    always #5 ms_riscv32_mp_clk_in =
~ms_riscv32_mp_clk_in;
    initial begin
```

```
        // Initialize Inputs
        pc_mux_in = 0;
        ms_riscv32_mp_clk_in = 0;
        ms_riscv32_mp_rst_in = 0;

        // Wait for global reset to finish
        #10;

        // Apply reset
        ms_riscv32_mp_rst_in = 1;
        #10;
        ms_riscv32_mp_rst_in = 0;
        #10;

        // Test Case 1: Apply a value to pc_mux_in
        pc_mux_in = 32'h00000130;
        #10;
        pc_mux_in = 32'h00000013;
        #10;

        // Test Case 2: Apply reset while pc_mux_in
is non-zero
        ms_riscv32_mp_rst_in = 1;
        #10;
        ms_riscv32_mp_rst_in = 0;
        #10;

        // Test Case 3: Apply another value to
pc_mux_in
        pc_mux_in = 32'hFFFFFFFF;
        #10;

        // Test Case 4: Apply another reset
        ms_riscv32_mp_rst_in = 1;
        #10;
        ms_riscv32_mp_rst_in = 0;
        #10;
        pc_mux_in = 32'h00044430;
        #10;
        pc_mux_in = 32'h02300013;
        #10;
        ms_riscv32_mp_rst_in = 1;
        #10;
        ms_riscv32_mp_rst_in = 0;
        #10;

        // Finish the simulation
        $stop;
    end

endmodule
```

RTL View / Micro Architecture - Reg Block 1 - msrv32_reg_block_1



Wave form - Reg Block 1 - msrv32_reg_block_1

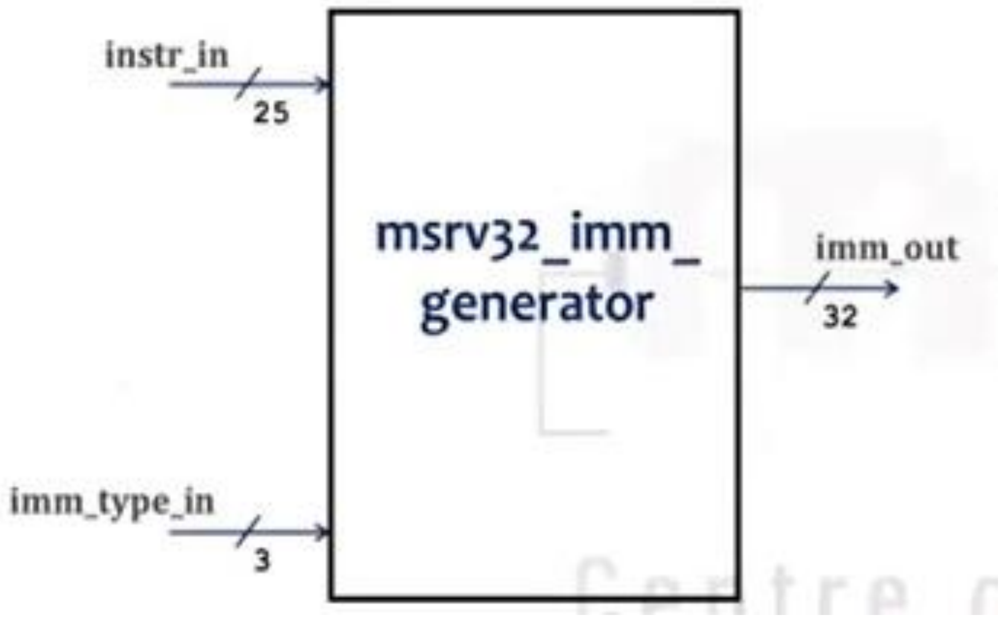


Name	Value
pc_mux_in[31:0]	00000130
ms_riscv32_mp_clk_in	1
ms_riscv32_mp_rst_in	0
pc_out[31:0]	00000130
[31]	0
[30]	0
[29]	0
[28]	0
[27]	0
[26]	0
[25]	0
[24]	0
[23]	0
[22]	0
[21]	0
[20]	0
[19]	0
[18]	0
[17]	0
[16]	0
[15]	0
[14]	0
[13]	0
[12]	0
[11]	0
[10]	0
[9]	0

```
pc_mux_in: 00000000 | 00000130 | 00000013 | ffffffff | 00044430 | 02300013
pc_out:    XX... | 00000000 | 00000130 | 00... | 00000000 | 00000013 | ff... | 00000000 | ffffffff | 00044430 | 02... | 00000000 | 02...
```

Immediate Generator

Block Diagram



Explanation – Immediate Generator – msrv32_imm_generator

Block receives two input signals named instr_in (25 bits) and imm_type_in(3 bits).

Block has one output signal named as imm_out (32 bit). The msrv32_imm_generator module extracts and assembles immediate values based on the instruction format specified by the imm_type_in input. In RISC-V 32I architecture, different instruction types have different ways of encoding immediate values, which the immediate generator handles. For example, I-type instructions use bits [31:20] of the instruction for immediate values, whereas S-type instructions combine bits [31:25] and [11:7]. The instr_in input provides the necessary bits from the instruction, and the imm_type_in input directs the module on how to assemble these bits into a 32-bit immediate value. The output imm_out is then used by other parts of the processor for executing immediate-related operations such as arithmetic operations, memory access, and branch offsets.

Block Signals – Immediate Generator– msrv32_imm_generator

1) instr_in : Connected to instruction bits (31 to 7).
2) imm_type_in : control signal generated by control unit that indicates the type of immediate that must be generated.
3) Imm_out : 32-bit generated immediate.

Verilog Code

Design code - Immediate Generator– msrv32_imm_generator

```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////
/////////////////////////////
// Company: Maven Silicon - VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 25.06.2024 21:06:29
// Design Name: Immediate Generator
// Module Name: msrv32_imm_generator
// Project Name: RISC -V 32I
// Target Devices:
// Tool Versions: Achyut's Vivado
// Description:
// Design Verilog code
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////
/////////////////////////////


module msrv32_imm_generator(instr_in, imm_type_in,
imm_out);
input [31:0] instr_in;
input [2:0] imm_type_in;
output reg [31:0] imm_out;
```

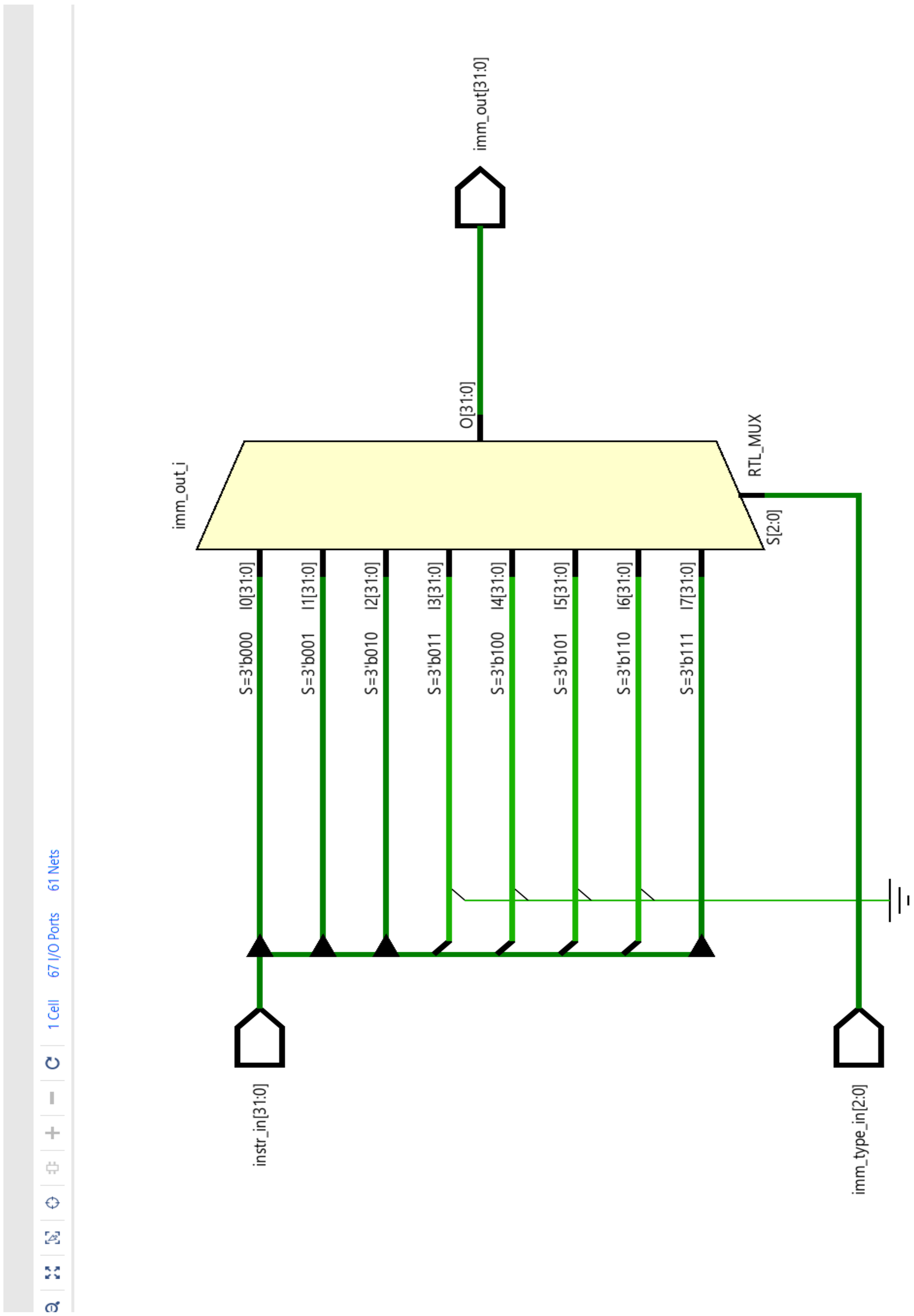
```
reg [31:0] i_type;
reg [31:0] s_type;
reg [31:0] b_type;
reg [31:0] u_type;
reg [31:0] j_type;
reg [31:0] csr_type;

always @(*)
begin
    i_type <= {{20{instr_in[31]}}, instr_in[31:20]};
    s_type <={{20{instr_in[31]}}, instr_in[31:25],
instr_in[11:7]};
    b_type <={{20{instr_in[31]}}, instr_in[7],
instr_in[30 :25], instr_in[11:8],1'b0};
    u_type <={instr_in[31 :12],12'h000};
    j_type <= {{12{instr_in[31]}}, instr_in[19:12],
instr_in[20], instr_in[30:21], 1'b0};
    csr_type <={27'b0,instr_in[19:15]};
end
always @(*)
begin
    case(imm_type_in)
        3'b000 : imm_out <= i_type;
        3'b001 : imm_out <= i_type;
        3'b010 : imm_out <= s_type;
        3'b011 : imm_out <= b_type;
        3'b100 : imm_out <= u_type;
        3'b101 : imm_out <= j_type;
        3'b110 : imm_out <= csr_type;
        3'b111 : imm_out <= i_type;
        default :imm_out <= i_type;
    endcase
end
endmodule
```

Simulation / Testbench code - Immediate Generator– msrv32_imm_generator

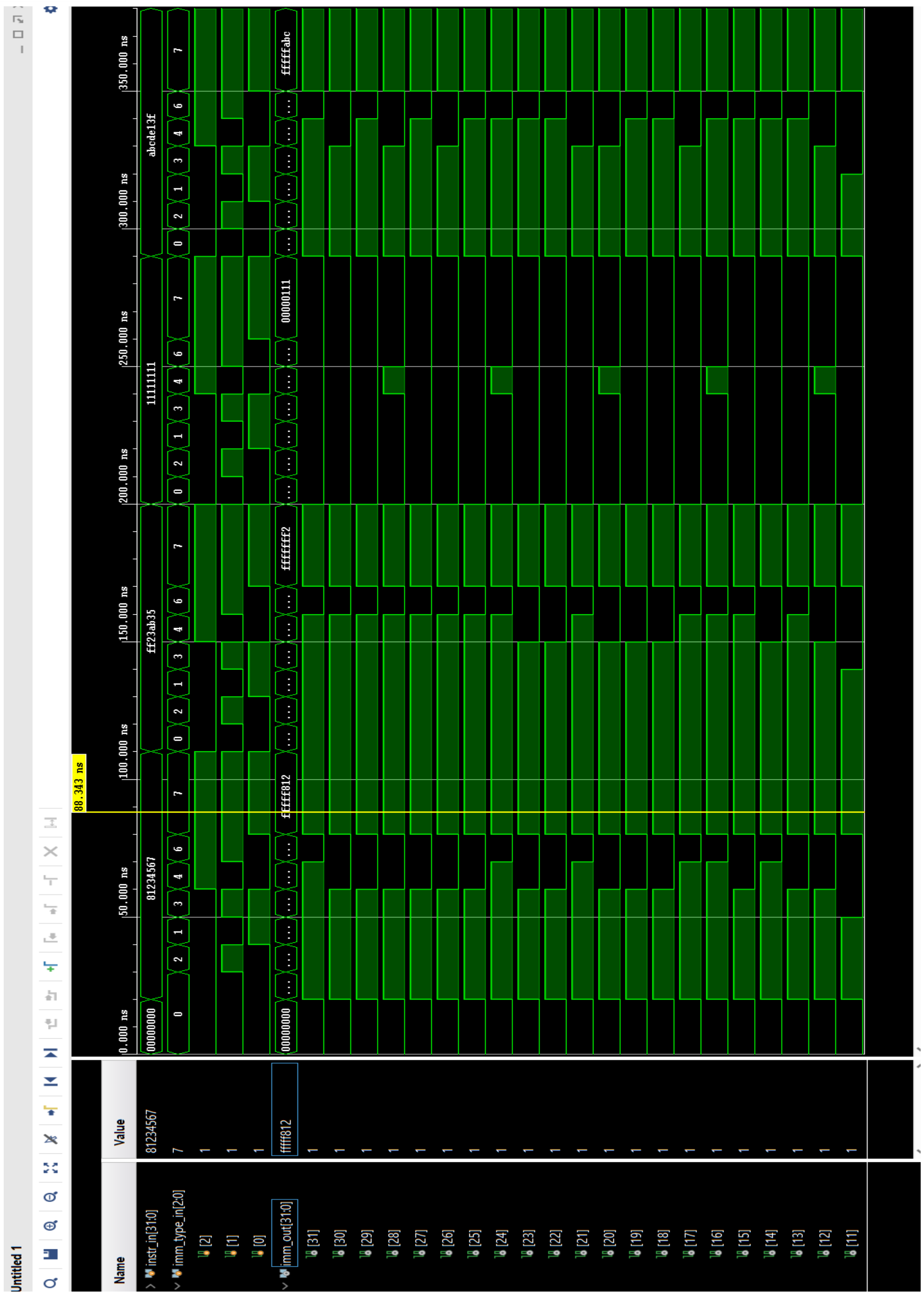
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////
//////////////////////////////
// Company: Maven Silicon - VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 27.06.2024 18:56:29
// Design Name: Immediate generator
// Module Name: msrv32_imm_generator_tb
// Project Name: RISC -V 32I
// Target Devices:
// Tool Versions: Achyut's vivado
// Description:
// Simulation testbench verilog code
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////
//////////////////////////////


module msrv32_imm_generator_tb;
reg [31:0] instr_in;
reg [2:0] imm_type_in;

// Outputs
wire [31:0] imm_out;

// Instantiate the Unit Under Test (UUT)
msrv32_imm_generator uut (
    .instr_in(instr_in),
    .imm_type_in(imm_type_in),
    .imm_out(imm_out)
);
initial
begin
      instr_in = 32'b0;
```

```
        imm_type_in = 3'b0;
        #20;
        instr_in = 32'h81234567;
        imm_type_in = 3'b000;
        #10;
        imm_type_in = 3'b010;
        #10;
        imm_type_in = 3'b001;
        #10;
        imm_type_in = 3'b011;
        #10;
        imm_type_in = 3'b100;
        #10;
        imm_type_in = 3'b110;
        #10;
        imm_type_in = 3'b111;
        #30;
        instr_in = 32'hFF23AB35;
        imm_type_in = 3'b000;
        #10;
        imm_type_in = 3'b010;
        #10;
        imm_type_in = 3'b001;
        #10;
        imm_type_in = 3'b011;
        #10;
        imm_type_in = 3'b100;
        #10;
        imm_type_in = 3'b110;
        #10;
        imm_type_in = 3'b111;
        #30;
        instr_in = 32'h11111111;
        imm_type_in = 3'b000;
        #10;
        imm_type_in = 3'b010;
        #10;
        imm_type_in = 3'b001;
        #10;
        imm_type_in = 3'b011;
        #10;
        imm_type_in = 3'b100;
        #10;
        imm_type_in = 3'b110;
        #10;
        imm_type_in = 3'b111;
        #30;
        instr_in = 32'hABCDE13F;
        imm_type_in = 3'b000;
        #10;
        imm_type_in = 3'b010;
        #10;
        imm_type_in = 3'b001;
        #10;
        imm_type_in = 3'b011;
        #10;
        imm_type_in = 3'b100;
        #10;
        imm_type_in = 3'b110;
        #10;
        imm_type_in = 3'b111;
        #30;
        $finish;
end
endmodule
```

RTL /Micro Architecture - Immediate Generator– msrv32_imm_generator

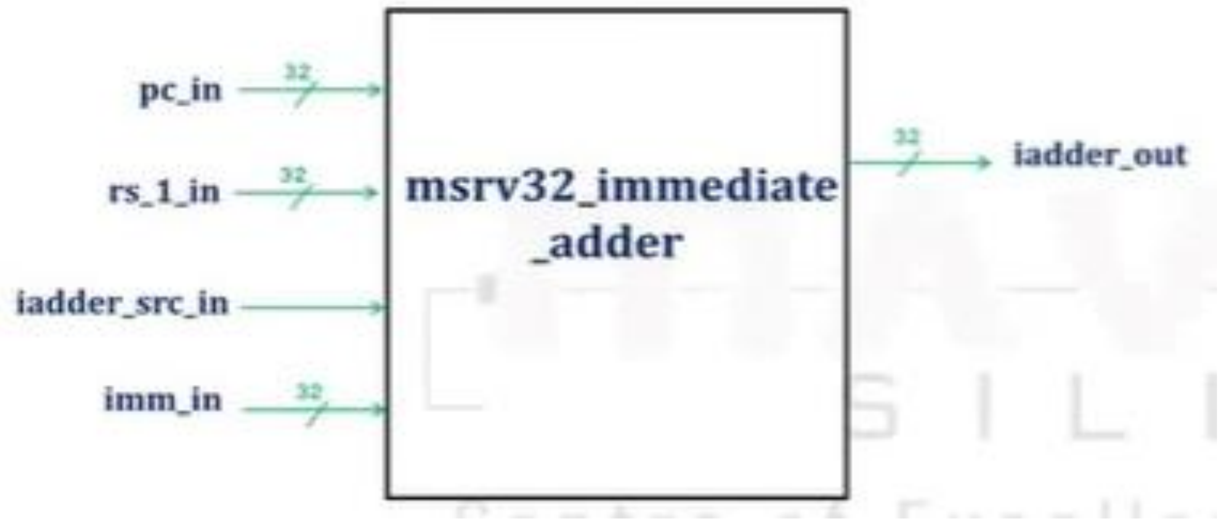


Wave form - Immediate Generator– msrv32_imm_generator



Immediate adder

Block Diagram



Block pins – Immediate adder – msrv32_immediate_adder

1) pc_in : program counter.
2) rs_1_in : Value of source 1 register.
3) imm_in : Immediate value used for instruction.
4) iadder_src_in : Gives indication of which type of instructions.
5) iadder_out : Resultant address with addition of immediate value with pc_in or rs_1_in.

Explanation – Immediate adder – msrv32_immediate_adder

The output **iadder_out** provides the resultant address, which is the sum of the immediate value and either the program counter or the source register value, depending on the iadder_src_in signal. This block is crucial for handling address calculations in instructions such as branches, jumps, and immediate arithmetic operations.

Verilog code

Design code - Immediate adder – msrv32_immediate_adder

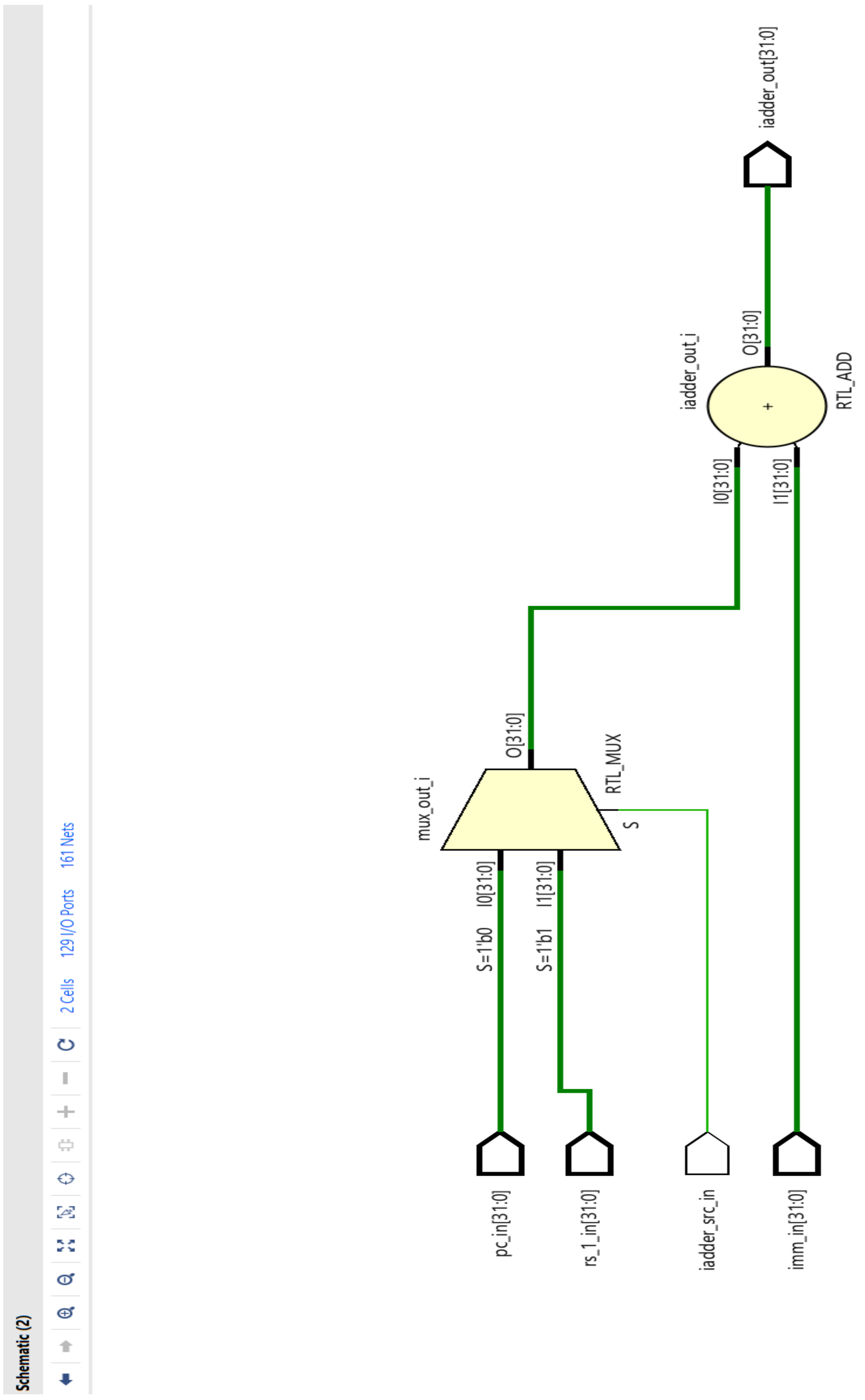
```
`timescale 1ns / 1ps
////////////////////////////////////////////////////
//////////////////////////////
// Company: Maven Silicon
// Engineer: VLSI Design
//
// Create Date: 26.06.2024 21:39:59
// Design Name: Immediate adder
// Module Name: msrv32_immediate_adder_tb
// Project Name: RISC -V 32I
// Target Devices:
// Tool Versions: Achyut's Vivado
// Description:
// Dependencies:
// RTL Design Code
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////
//////////////////////////////


module msrv32_immediate_adder(pc_in,rs_1_in,
iadder_src_in, imm_in,iadder_out);
input [31:0] pc_in;
input [31:0] rs_1_in;
input iadder_src_in;
input [31:0] imm_in;
output [31:0] iadder_out;
reg [31:0] mux_out;
always @(*) begin
    case(iadder_src_in)
        1'b0 : mux_out = pc_in;
        1'b1 : mux_out = rs_1_in;
        default: mux_out = 32'b0; // Add default case
to avoid latches
    endcase
end
```

```
assign iadder_out = mux_out + imm_in;
endmodule
```

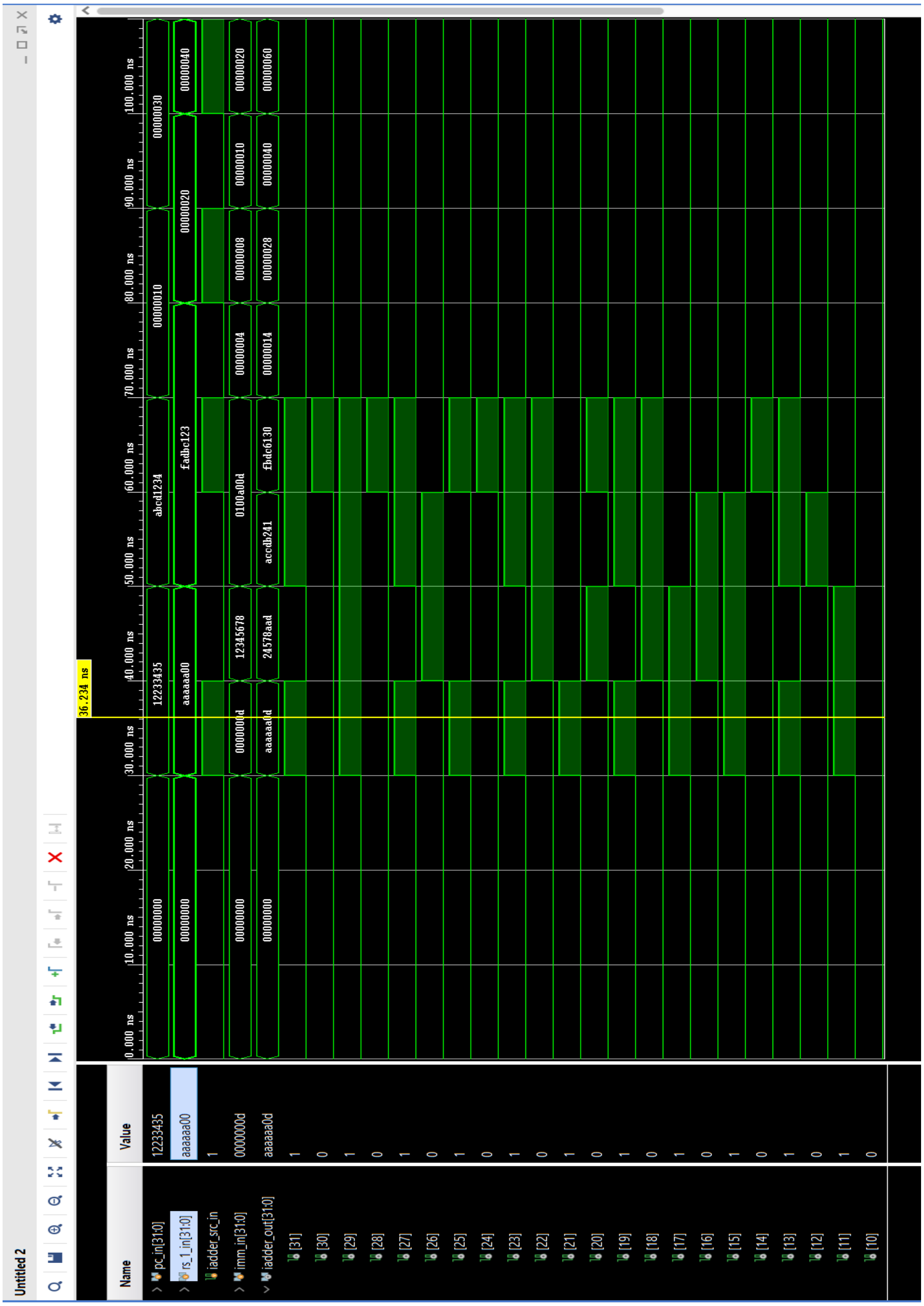
Simulation /Testbench - Immediate adder – msrv32_immediate_adder

```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////
//////////////////////////////
// Company: Maven Silicon
// Engineer: VLSI Design
//
// Create Date: 27.06.2024 20:38:59
// Design Name: Immediate adder
// Module Name: msrv32_immediate_adder_tb
// Project Name: RISC -V 32I
// Target Devices:
// Tool Versions: Achyut's Vivado
// Description:
// Simulation code
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////
//////////////////////////////


module msrv32_immediate_adder_tb;
reg [31:0] pc_in;
reg [31:0] rs_1_in;
reg iadder_src_in;
reg [31:0] imm_in;

// Outputs
wire [31:0] iadder_out;

// Instantiate the Unit Under Test (UUT)
msrv32_immediate_adder uut (
    .pc_in(pc_in),
    .rs_1_in(rs_1_in),
    .iadder_src_in(iadder_src_in),
    .imm_in(imm_in),
    .iadder_out(iadder_out)
);
initial begin
    // Initialize Inputs
    pc_in = 32'h00000000;
    rs_1_in = 32'h00000000;
    iadder_src_in = 0;
    imm_in = 32'h00000000;
    // Wait for global reset to finish
    #30;

    pc_in = 32'h12233435;
    rs_1_in = 32'hAAAAAA00;
    iadder_src_in = 1;
    imm_in = 32'hD;
    #10;
    pc_in = 32'h12233435;
    rs_1_in = 32'hAAAAAA00;
    iadder_src_in = 0;
    imm_in = 32'h12345678;
    #10;
    pc_in = 32'hABCD1234;
    rs_1_in = 32'hFADBC123;
    iadder_src_in = 0;
    imm_in = 32'h100A00D;
    #10;
    pc_in = 32'hABCD1234;
    rs_1_in = 32'hFADBC123;
    iadder_src_in = 1;
    imm_in = 32'h100A00D;
    #10;

    pc_in = 32'h00000010;
    imm_in = 32'h00000004;
    iadder_src_in = 0; // Select pc_in
    #10;

    rs_1_in = 32'h00000020;
    imm_in = 32'h00000008;
    iadder_src_in = 1; // Select rs_1_in
    #10;
    // Test Case 3: Add imm_in to pc_in with
different values
    pc_in = 32'h00000030;
    imm_in = 32'h00000010;
    iadder_src_in = 0; // Select pc_in
    #10;

    // Test Case 4: Add imm_in to rs_1_in with
different values
    rs_1_in = 32'h00000040;
    imm_in = 32'h00000020;
    iadder_src_in = 1; // Select rs_1_in
    #10;

    // Add more test cases as needed

    $finish;
end


endmodule
```

RTL / Micro Architecture - Immediate adder – msrv32_immediate_adder



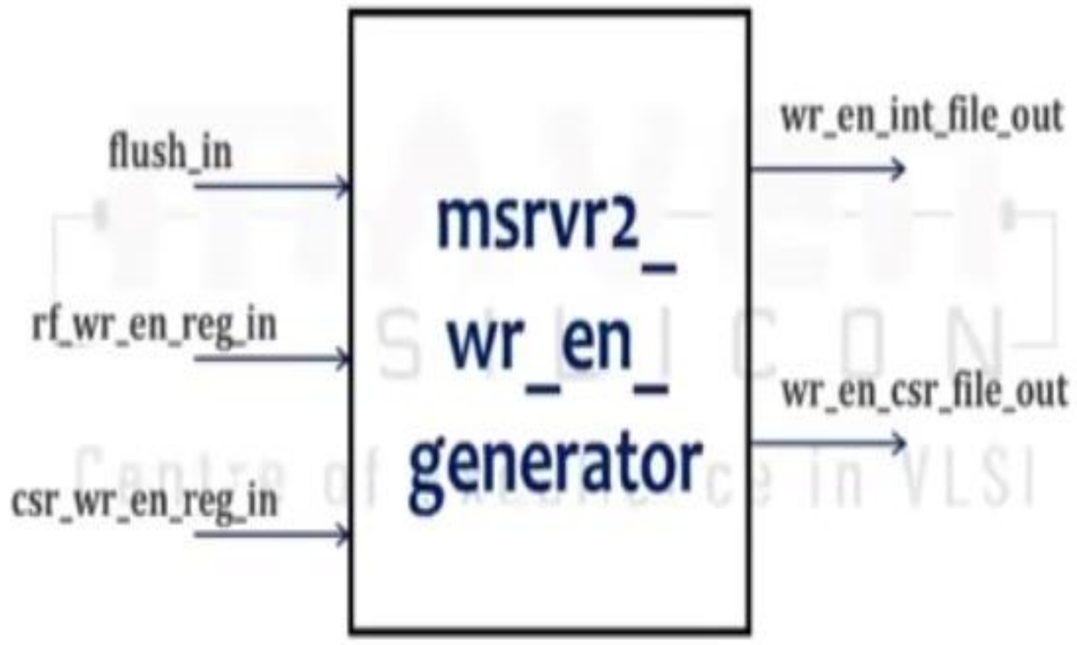
Wave Form - Immediate adder – msrv32_immediate_adder



Name	Value
pc_in[31:0]	12233435
rs_1_in[31:0]	aaaaaa00
iadder_src_in	1
imm_in[31:0]	0000000d
iadder_out[31:0]	aaaaaa0d
[31]	1
[30]	0
[29]	1
[28]	0
[27]	1
[26]	0
[25]	1
[24]	0
[23]	1
[22]	0
[21]	1
[20]	0
[19]	1
[18]	0
[17]	1
[16]	0
[15]	1
[14]	0
[13]	1
[12]	0
[11]	1
[10]	0

Write Enable Generator

Block Diagram



Block pins – Write Enable Generator - msrv32_wr_en_generator

1) flush_in : Signal to flush the pipeline during exceptions or branch mispredictions.
2) rf_wr_en_reg_in : Write enable signal for the register file to allow data writing.
3) csr_wr_en_reg_in : Write enable signal for the Control and Status Registers to allow data writing.
4) wr_en_int_file_out : Generated write enable signal for internal file operations.
5) wr_en_csr_file_out : Generated write enable signal for CSR file operations.

Explanation– Write Enable Generator - msrv32_wr_en_generator

It gets signals that tell it when to clear the pipeline and when to write data to the main registers and special control registers. Based on these inputs, it sends out signals to allow writing to these areas. This ensures that data is written correctly and the pipeline is managed properly during certain events. Overall, it helps keep the processor running smoothly and efficiently.

Verilog code

RTL/design code– Write Enable Generator -msrv32_wr_en_generator

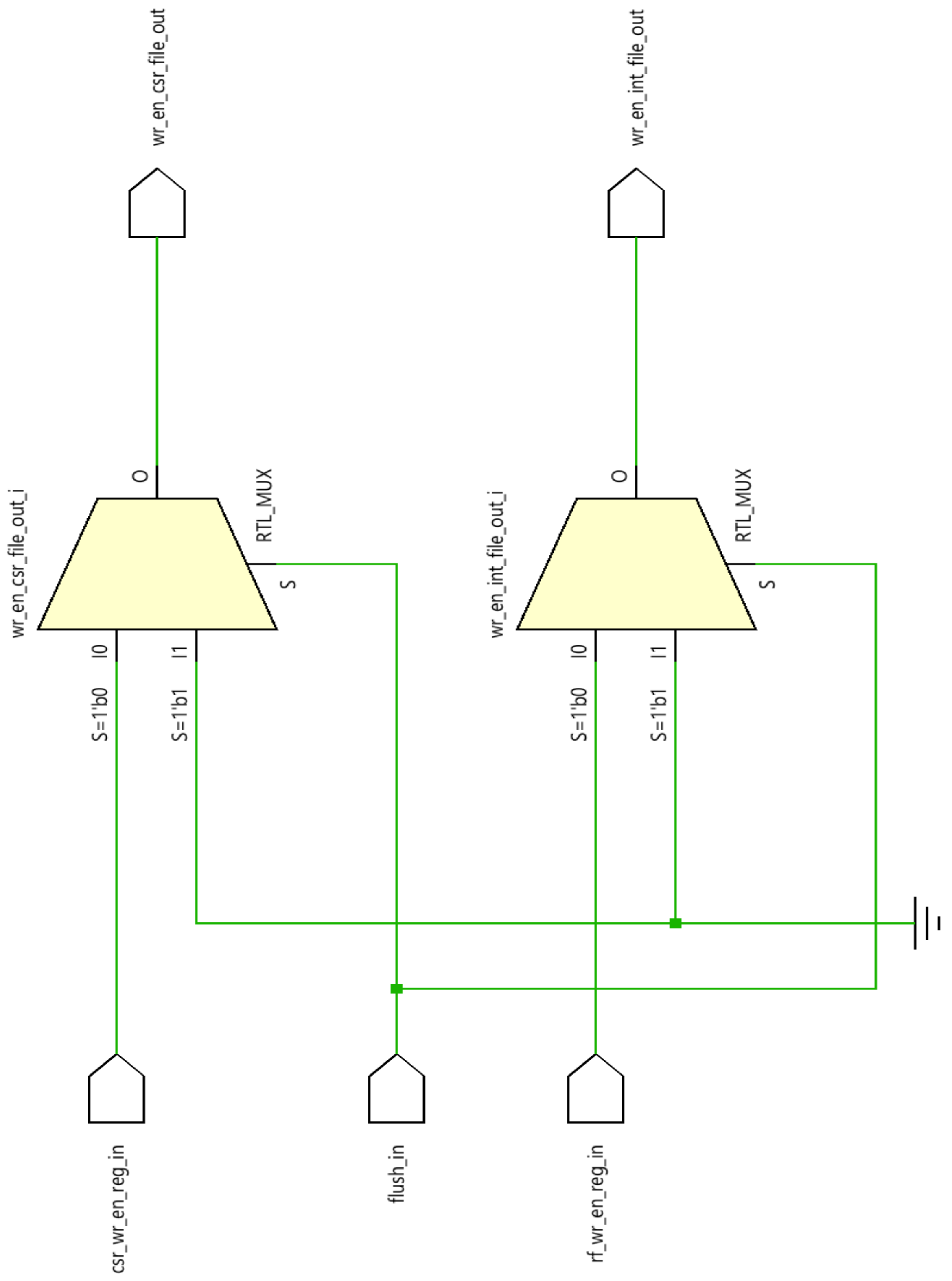
```
`timescale 1ns / 1ps
////////////////////////////////////////////////////
///////////////////////////////
// Company: Maven Silicon - VIT vellore
// Engineer: VLSI Design
//
// Create Date: 26.06.2024 16:06:47
// Design Name: Write Enable Generator
// Module Name: msrv2_wr_en_generator
// Project Name: RISC -V 32I
// Target Devices:
// Tool Versions: Achyut's Vivado
// Description:
// RTL Design code`
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////
///////////////////////////////


module msrv2_wr_en_generator(flush_in,
rf_wr_en_reg_in, csr_wr_en_reg_in,
wr_en_int_file_out, wr_en_csr_file_out);
input flush_in;
input rf_wr_en_reg_in;
input csr_wr_en_reg_in;

output reg wr_en_int_file_out;
output reg wr_en_csr_file_out;

always @(flush_in)
    case(flush_in)
        1'b0 : begin
            wr_en_int_file_out <= rf_wr_en_reg_in;
            wr_en_csr_file_out <= csr_wr_en_reg_in;
            end
        1'b1: begin
            wr_en_int_file_out <= 1'b0;
            wr_en_csr_file_out <= 1'b0;
            end
    endcase
endmodule
```

Testbench /simulation code – Write Enable Generator - msrv32_wr_en_generator

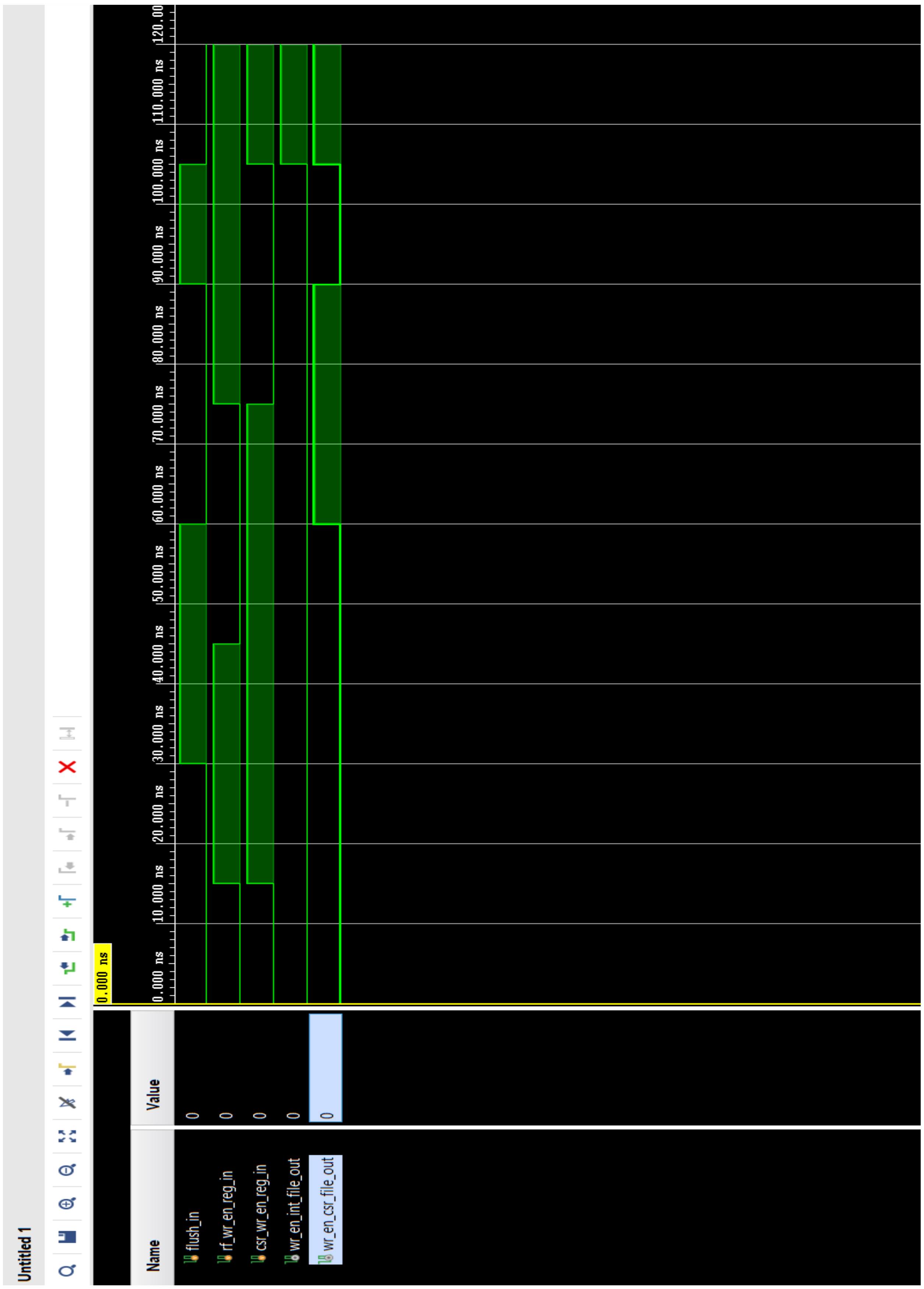
```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////
/////////////////////////////
// Company: Maven Silicon -VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 30.06.2024 17:46:30
// Design Name: Write enable generator
// Module Name: msrv2_wr_en_generator_tb
// Project Name: RISC -V 32I
// Target Devices:
// Tool Versions: Achyut's Vivado
// Description:
// Testbench code
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////
/////////////////////////////


module msrv2_wr_en_generator_tb;
reg flush_in;
reg rf_wr_en_reg_in;
reg csr_wr_en_reg_in;

// Outputs
```

```
wire wr_en_int_file_out;
wire wr_en_csr_file_out;

// Instantiate the Unit Under Test (UUT)
msrv2_wr_en_generator uut (
    .flush_in(flush_in),
    .rf_wr_en_reg_in(rf_wr_en_reg_in),
    .csr_wr_en_reg_in(csr_wr_en_reg_in),
    .wr_en_int_file_out(wr_en_int_file_out),
    .wr_en_csr_file_out(wr_en_csr_file_out)
);

initial
begin
    // Initialize Inputs
    flush_in = 0;
    rf_wr_en_reg_in = 0;
    csr_wr_en_reg_in = 0;
    #15;
    flush_in = 0;
    rf_wr_en_reg_in = 1;
    csr_wr_en_reg_in = 1;
    #15;
    flush_in = 1;
    rf_wr_en_reg_in = 1;
    csr_wr_en_reg_in = 1;
    #15;
    flush_in = 1;
    rf_wr_en_reg_in = 0;
    csr_wr_en_reg_in = 1;
    #15;
    flush_in = 0;
    rf_wr_en_reg_in = 0;
    csr_wr_en_reg_in = 1;
    #15;
    flush_in = 0;
    rf_wr_en_reg_in = 1;
    csr_wr_en_reg_in = 0;
    #15;
    flush_in = 1;
    rf_wr_en_reg_in = 1;
    csr_wr_en_reg_in = 0;
    #15;
    flush_in = 0;
    rf_wr_en_reg_in = 1;
    csr_wr_en_reg_in = 1;
    #15
    $finish;
end
endmodule
```

RTL Design / Micro Architecture – Write Enable Generator - msrv32_wr_en_generator

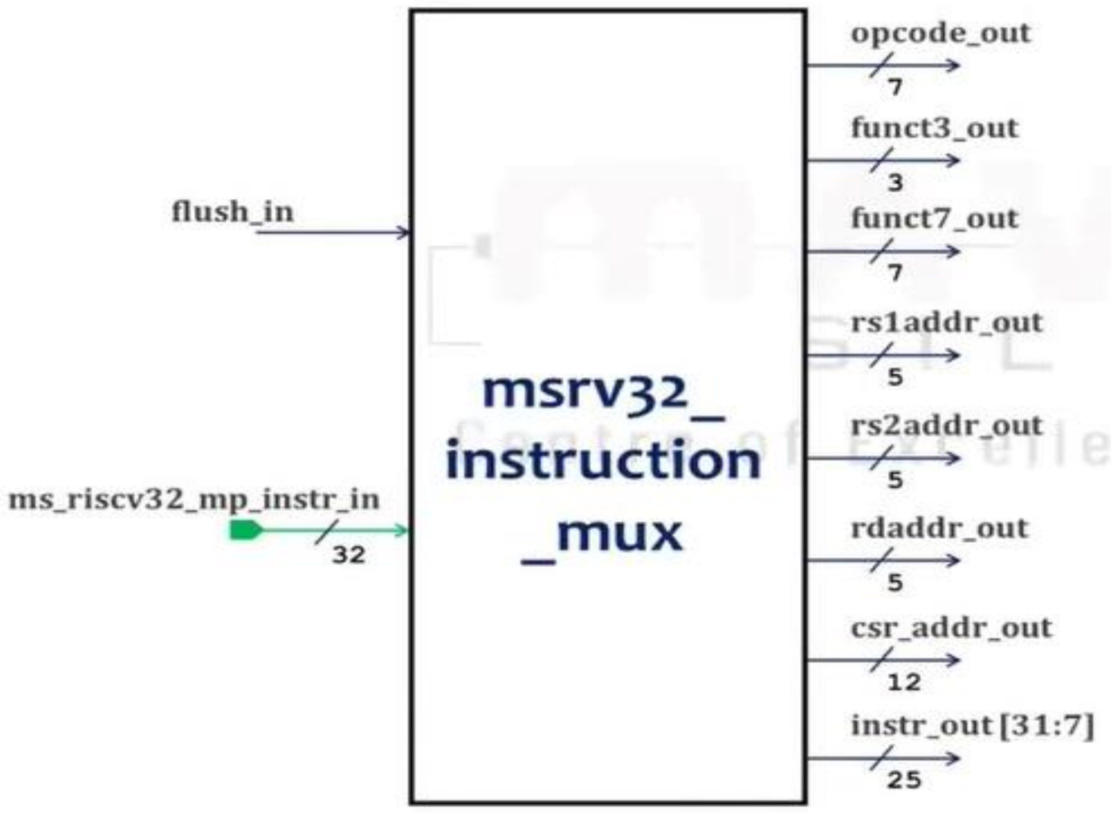


Waveform – Write Enable Generator -msrv32_wr_en_generator



INSTRUCTION MUX

Block Diagram



Block pins -Instruction mux - msrv32_instruction_mux

1) flush_in : Flushes the unit(with 32’h00000013) when set.
2) instr_in : Contains the instruction fetched from memory.
3) opcode_out : Opcode of each instructions.
4) funct3_out : funct3 field of instruction.
5) funct7_out : funct7 field of instruction.
6) rs1_addr_out : Contains the address of the source 1 register.
7) rs2_addr_out : Contains the address of the source 2 register.
8) rd_addr_out : Destination register address.
9) csr_addr_out : Adress of the CSR to read/ write/modify.
10) Instr_31_7_out : Connected to immediate generator.

Explanation -Instruction mux - msrv32_instruction_mux

The msrv32_instruction_mux block takes an instruction input (ms_riscv32_mp_instr_in) and a flush signal (flush_in). If flush_in is active, the block outputs a default instruction (32'h00000013). The block extracts and outputs specific fields from the input instruction: opcode_out, funct3_out, funct7_out, rs1addr_out, rs2addr_out, rdaddr_out, and csr_addr_out. It also outputs the remaining part of the instruction as instr_out. This module is used to process and decode the RISC-V instruction, preparing it for further stages in the processor pipeline.

Verilog code

RTL / Design code - Instruction mux - msrv32_instruction_mux

```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////
//////////////////////////////
// Company: Maven Silicon -VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 26.06.2024 16:36:41
// Design Name: Instruction mux
// Module Name: msrv32_instruction_mux
// Project Name: RSIC -V 32I
// Target Devices:
// Tool Versions: Achyut's Vivado
// Description:
// RTL / design code
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////
//////////////////////////////
```

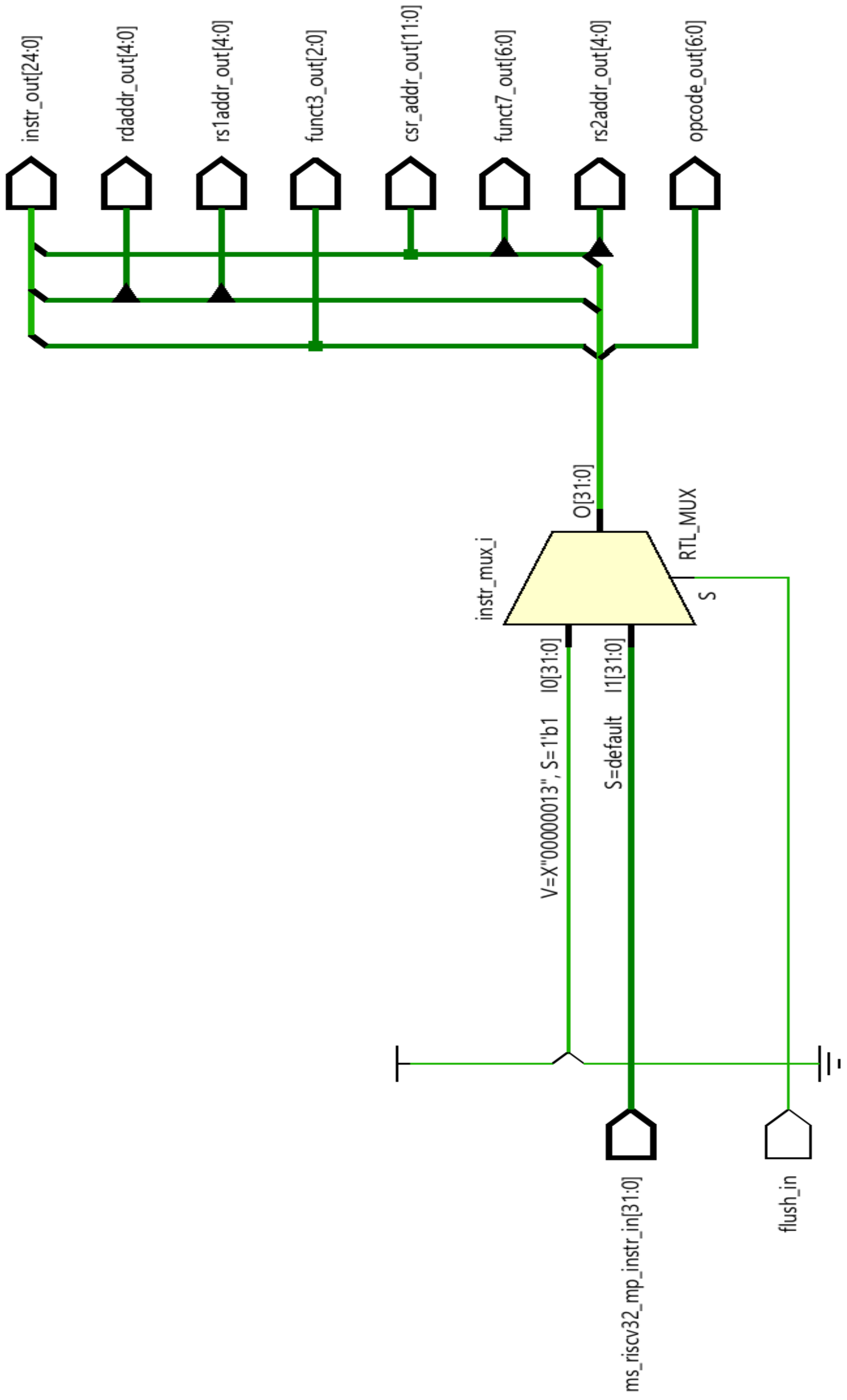
```
module msrv32_instruction_mux(flush_in,
ms_riscv32_mp_instr_in, opcode_out, funct3_out,
funct7_out, rs1addr_out, rs2addr_out, rdaddr_out,
csr_addr_out, instr_out);
input flush_in;
input [31:0] ms_riscv32_mp_instr_in;

output [6:0] opcode_out;
output [2:0] funct3_out;
output [6:0] funct7_out;
output [4:0] rs1addr_out;
output [4:0] rs2addr_out;
output [4:0] rdaddr_out;
output [11:0] csr_addr_out;
output [24:0]  instr_out;
wire [31:0] instr_mux;
assign  instr_mux = (flush_in)? 32'h00000013 :
ms_riscv32_mp_instr_in;
assign opcode_out = instr_mux[6:0];
assign funct3_out = instr_mux[14:12];
assign funct7_out = instr_mux[31:25];
assign rs1addr_out = instr_mux[19:15];
assign rs2addr_out = instr_mux[24:20];
assign rdaddr_out = instr_mux[11:7];
assign csr_addr_out = instr_mux[31:20];
assign instr_out = instr_mux[31:7];
endmodule
```

Simulation / Testbench - Instruction mux - msrv32_instruction_mux

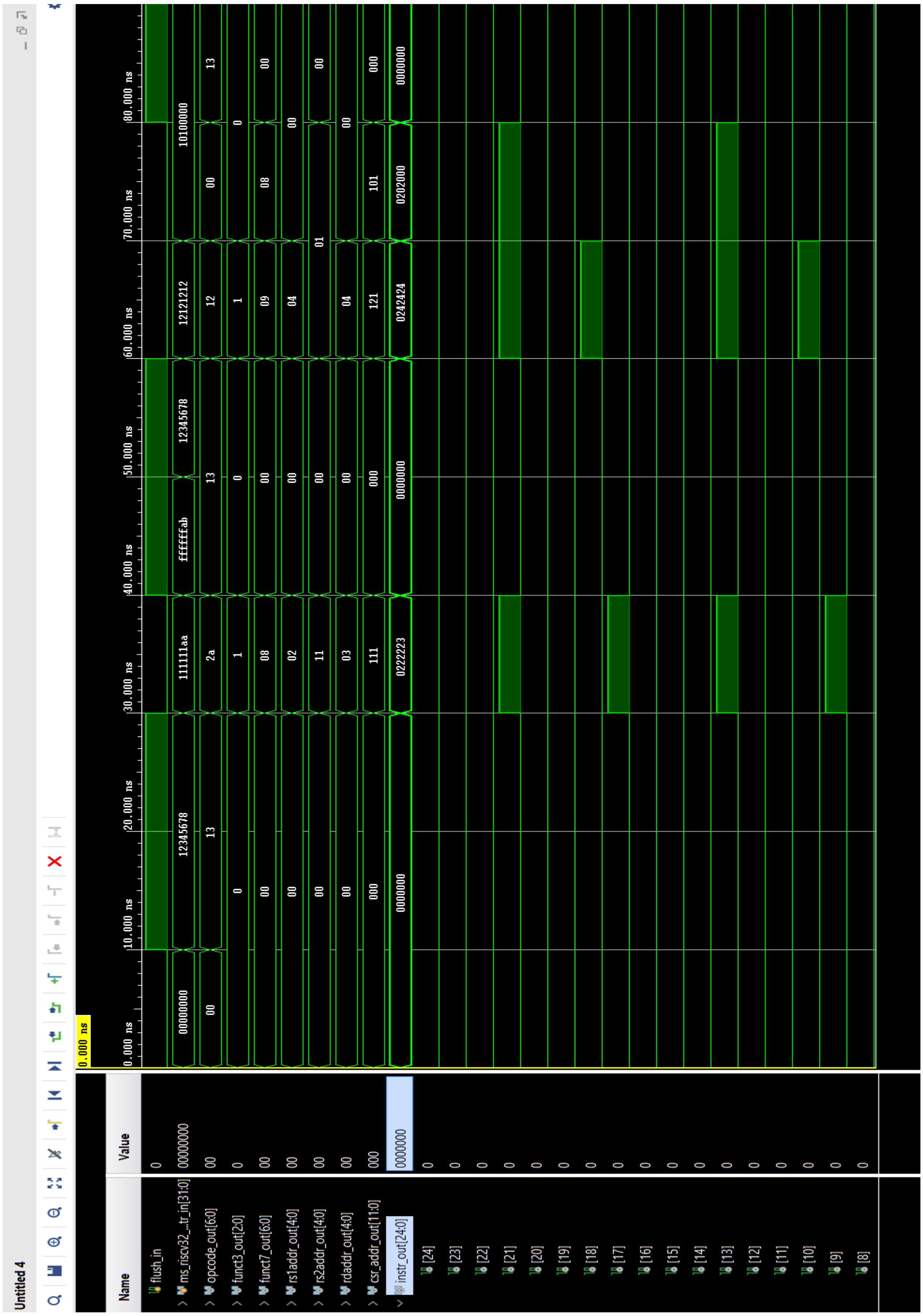
```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////
/////////////////////////////
// Company: Maven Silicon - VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 30.06.2024 19:33:29
// Design Name: Instuction mux
// Module Name: msrv32_instruction_mux_tb
// Project Name: RISC -V 32I
// Target Devices:
// Tool Versions: Achyut's vivado
// Description:
// Testbench code
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////
//////////////////////////////


module msrv32_instruction_mux_tb;
    // Inputs
    reg flush_in;
    reg [31:0] ms_riscv32_mp_instr_in;

    // Outputs
    wire [6:0] opcode_out;
    wire [2:0] funct3_out;
    wire [6:0] funct7_out;
    wire [4:0] rs1addr_out;
    wire [4:0] rs2addr_out;
    wire [4:0] rdaddr_out;
    wire [11:0] csr_addr_out;
    wire [24:0] instr_out;

    // Instantiate the Unit Under Test (UUT)
    msrv32_instruction_mux uut (
        .flush_in(flush_in),

.ms_riscv32_mp_instr_in(ms_riscv32_mp_instr_in),
        .opcode_out(opcode_out),
        .funct3_out(funct3_out),
        .funct7_out(funct7_out),
        .rs1addr_out(rs1addr_out),
        .rs2addr_out(rs2addr_out),
        .rdaddr_out(rdaddr_out),
        .csr_addr_out(csr_addr_out),
        .instr_out(instr_out)
    );
```

```
initial
begin
        // Initialize Inputs
        flush_in = 0;
        ms_riscv32_mp_instr_in = 32'h00000000;
        #10;
        flush_in = 1;
        ms_riscv32_mp_instr_in = 32'h12345678;
        #10;
        flush_in = 1;
        ms_riscv32_mp_instr_in = 32'h12345678;
        #10;
        flush_in = 0;
        ms_riscv32_mp_instr_in = 32'h111111AA;
        #10;
        flush_in = 1;
        ms_riscv32_mp_instr_in = 32'hFFFFFFAB;
        #10;
        flush_in = 1;
        ms_riscv32_mp_instr_in = 32'h12345678;
        #10;
        flush_in = 0;
        ms_riscv32_mp_instr_in = 32'h12121212;
        #10;
        flush_in = 0;
        ms_riscv32_mp_instr_in = 32'h10100000;
        #10;
        flush_in = 1;
        #10;
        flush_in = 0;

        $finish;
end


endmodule
```

RTL / Micro Architecture - Instruction mux - msrv32_instruction_mux

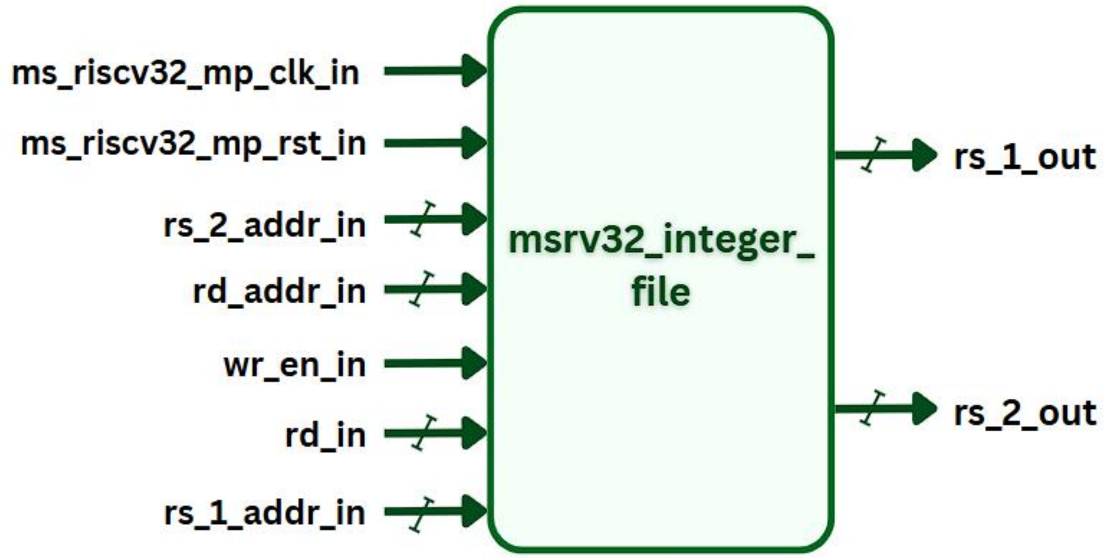


Waveform - Instruction mux - msrv32_instruction_mux



INTEGER FILE

Block Diagram



Block pins:

1) ms_riscv32_mp_clk_in: Input clock signal for the module to synchronize operations.
2) ms_riscv32_mp_rst_in : Input reset signal to initialize or reset the module.
3) rs_2_addr_in : Input address for the second source register.
4) rd_addr_in : Input address for the destination register where results are written.
5) wr_en_in : Input signal to enable writing data to the destination register.
6) rd_in : Input data to be written to the destination register.
7) rs_1_out : Output data from the first source register.
8) rs_1_out : Output data from the second source register.

Explanation – Integer File - msrv32_integer_file

The msrv32_integer_file module operates as a part of the RISC-V 32I processor architecture. It utilizes the input clock (ms_riscv32_mp_clk_in) and reset (ms_riscv32_mp_rst_in) signals to synchronize and initialize its operations. The module takes addresses for source registers (rs_2_addr_in) and the destination register (rd_addr_in). When the write enable signal (wr_en_in) is active, the input data (rd_in) is written to the specified destination register. The module reads data from the specified source registers and provides the output through rs_1_out (and possibly rs_2_out). This data can then be used by other components of the processor for various arithmetic and logical operations. Overall, the module facilitates the storage, retrieval, and updating of integer data in the processor's register file, enabling efficient instruction execution.

Verilog Code

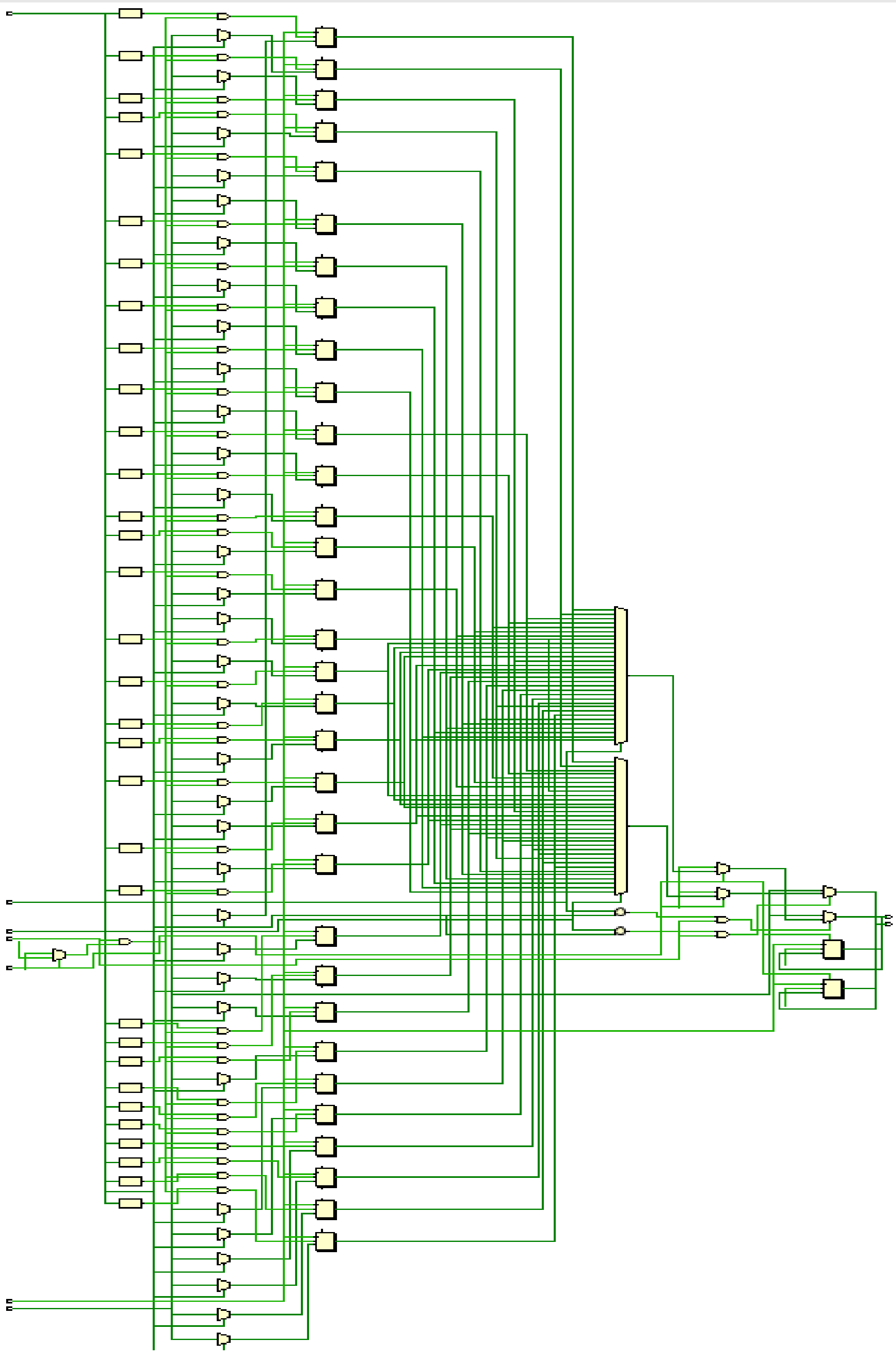
RTL / Design Code - Integer File -msrv32_integer_file

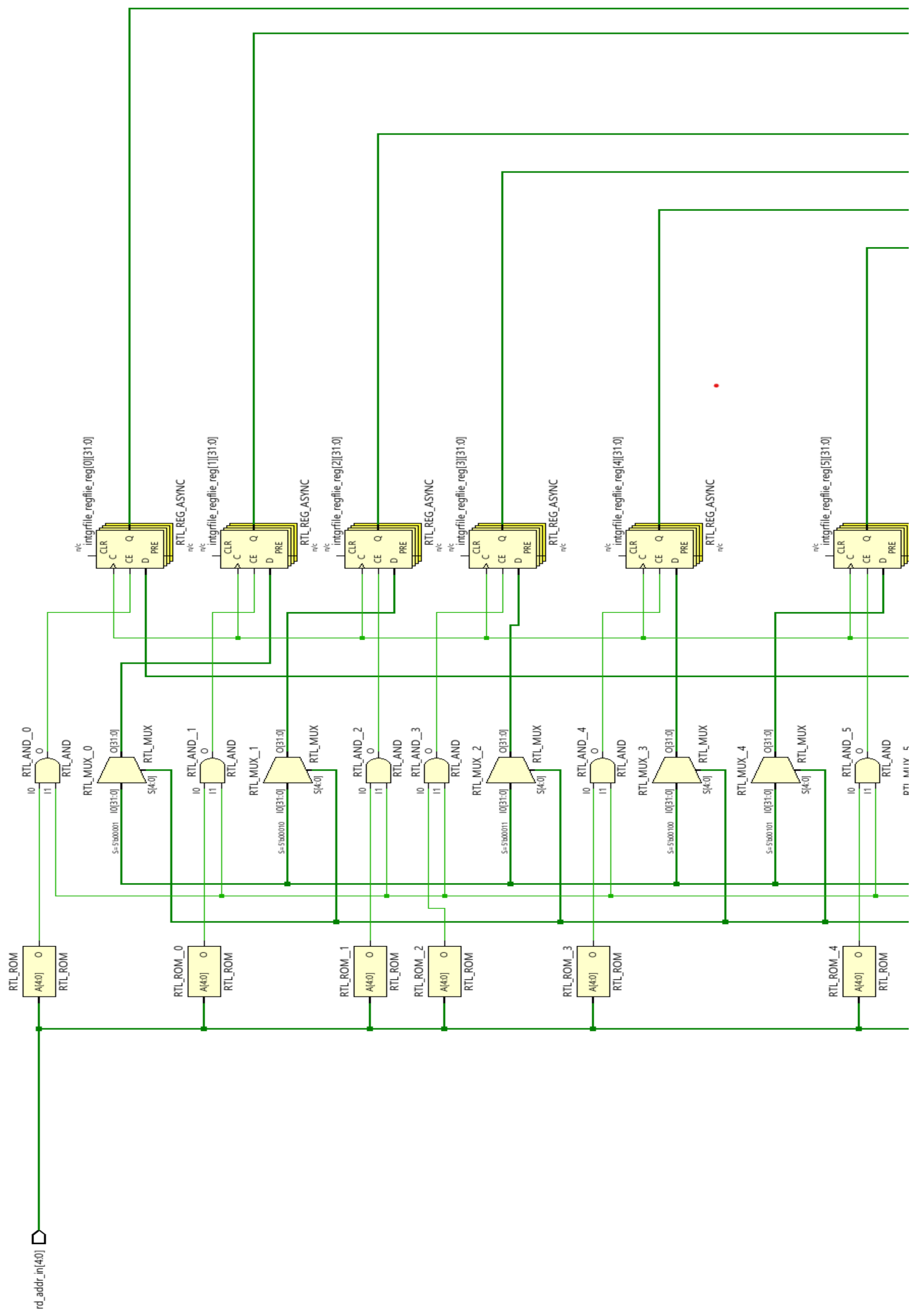
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////
//////////////////////////////
// Company: Maven Silicon -VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 29.06.2024 09:00:42
// Design Name: Integer file
// Module Name: msrv32_integer_file
// Project Name: RISC -V 32I
// Target Devices:
// Tool Versions: Achyut's Vivado
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////
//////////////////////////////


module msrv32_integer_file(ms_riscv32_mp_clk_in,
ms_riscv32_mp_rst_in, rs_2_addr_in, rd_addr_in,
wr_en_in, rd_in, rs_1_addr_in, rs_1_out, rs_2_out);
input ms_riscv32_mp_clk_in ;
input ms_riscv32_mp_rst_in;
input [4:0]rs_2_addr_in;
input [4:0]rd_addr_in;
input wr_en_in;
input [31:0]rd_in;
input [4:0]rs_1_addr_in;
reg [31:0] rs1_out;
reg [31:0] rs2_out;
output reg [31:0] rs_1_out;
output reg [31:0] rs_2_out;
wire mux_rs1_en;
wire mux_rs2_en;
reg [31:0] intgrfile_regflie [31:0];
always @(posedge ms_riscv32_mp_clk_in or posedge
ms_riscv32_mp_rst_in)
begin
    if(ms_riscv32_mp_rst_in)
    begin
        rs_1_out  <=1'b0;
        rs_2_out <= 1'b0;
    end
    else
    begin
        if(wr_en_in)
        begin
            intgrfile_regflie[rd_addr_in]  = rd_in;
        end
        end
end
 always @(*) begin
        if (ms_riscv32_mp_rst_in) begin
            rs1_out = 32'b0;
            rs2_out = 32'b0;
        end else begin
            rs1_out =
intgrfile_regflie[rs_1_addr_in];
            rs2_out =
intgrfile_regflie[rs_2_addr_in];
        end
    end
assign mux_rs1_en = (rs_1_addr_in == rd_addr_in &&
wr_en_in == 1'b1) ? 1'b1 : 1'b0;
assign mux_rs2_en = (rs_2_addr_in == rd_addr_in &&
wr_en_in == 1'b1) ? 1'b1 : 1'b0;
always @(rs1_out or rs2_out)
begin
        rs_1_out = (mux_rs1_en)? rd_in : rs1_out;
        rs_2_out = (mux_rs2_en)? rd_in : rs2_out;
    end



endmodule
```

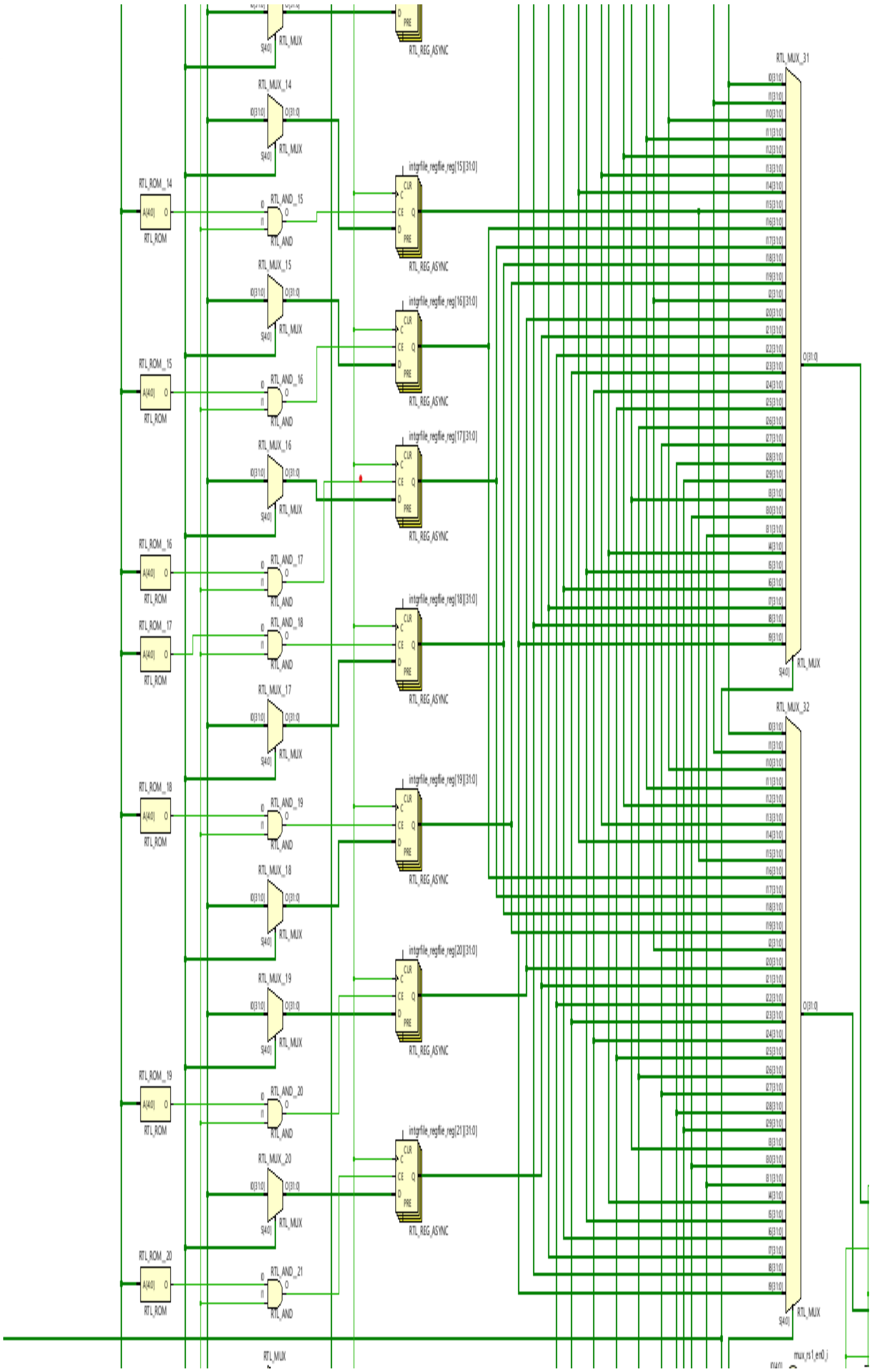
Testbench Code - Integer File -msrv32_integer_file

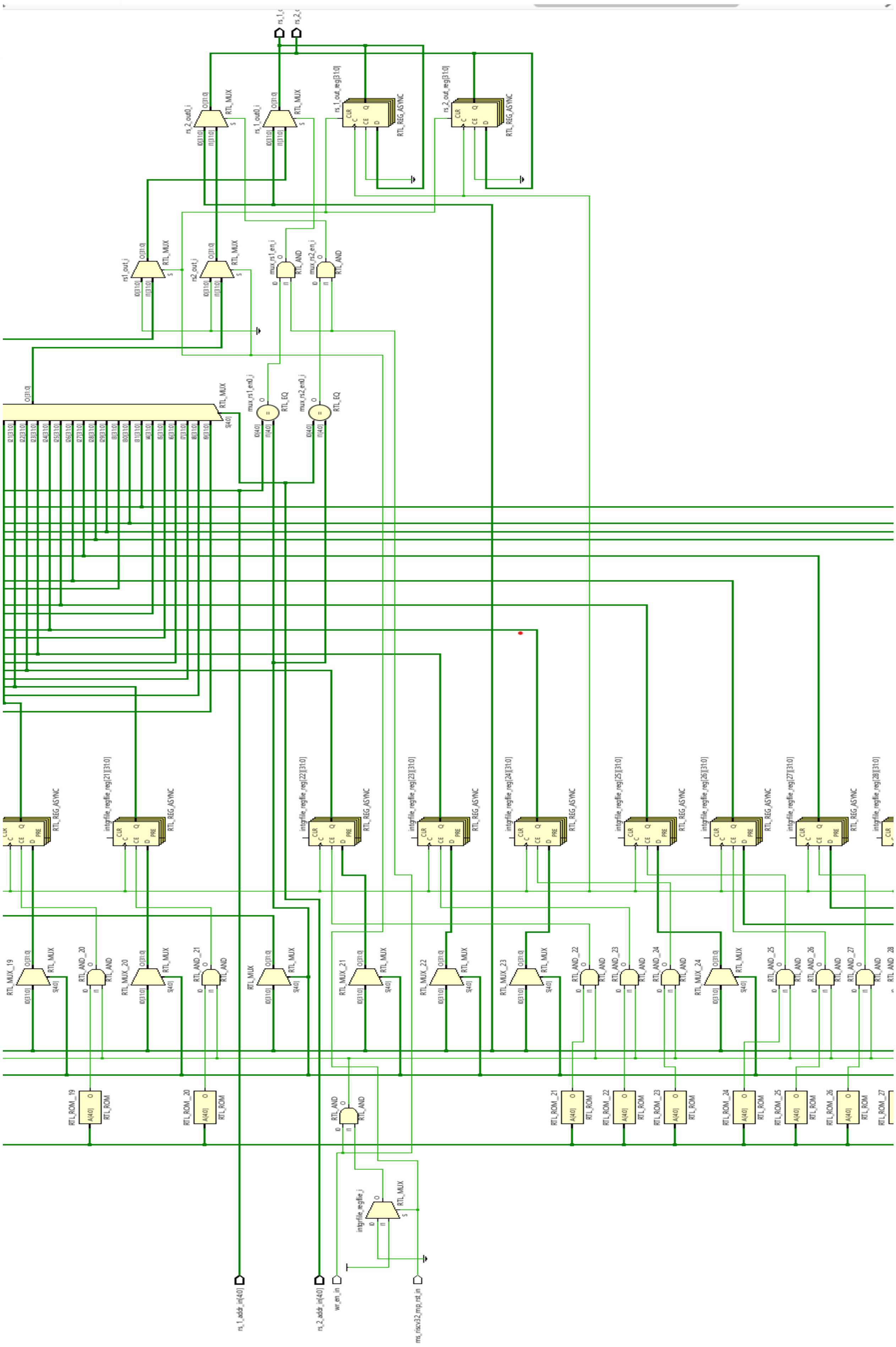
```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////
//////////////////////////////
// Company: Maven Silicon - VIT vellore
// Engineer: VLSI Design
//
// Create Date: 01.07.2024 20:54:02
// Design Name: Integer file
// Module Name: msrv32_integer_file_tb
// Project Name: RISC -V 32I
// Target Devices:
// Tool Versions: Achyut's Vivado
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////
//////////////////////////////

module msrv32_integer_file_tb;
reg ms_riscv32_mp_clk_in;
reg ms_riscv32_mp_rst_in;
reg [4:0] rs_2_addr_in;
reg [4:0] rd_addr_in;
reg wr_en_in;
reg [31:0] rd_in;
reg [4:0] rs_1_addr_in;

// Outputs
wire [31:0] rs_1_out;
wire [31:0] rs_2_out;

// Instantiate the Unit Under Test (UUT)
msrv32_integer_file uut (
    .ms_riscv32_mp_clk_in(ms_riscv32_mp_clk_in),
    .ms_riscv32_mp_rst_in(ms_riscv32_mp_rst_in),
    .rs_2_addr_in(rs_2_addr_in),
    .rd_addr_in(rd_addr_in),
    .wr_en_in(wr_en_in),
    .rd_in(rd_in),
    .rs_1_addr_in(rs_1_addr_in),
    .rs_1_out(rs_1_out),
    .rs_2_out(rs_2_out)
);

// Clock generation
initial begin
    ms_riscv32_mp_clk_in = 0;
    forever #5 ms_riscv32_mp_clk_in =
~ms_riscv32_mp_clk_in; // 10ns period
end
initial  begin
    ms_riscv32_mp_rst_in = 1;
    rs_2_addr_in = 0;
    rd_addr_in = 0;
    wr_en_in = 0;
    rd_in = 0;
    rs_1_addr_in = 0;
    #10;
    ms_riscv32_mp_rst_in = 0;
    wr_en_in = 1;
    // Test Case 1: Write to register 1 and read it
back
    #10;
    wr_en_in = 1;
    rd_addr_in = 5'd1;
    rd_in = 32'hA5A5A5A5;

    #10;
    wr_en_in = 1;
    rs_1_addr_in = 5'b10111;
    rs_2_addr_in = 5'b00001;

    #10;
    wr_en_in = 1;
    rd_addr_in = 5'd2;
    rd_in = 32'h5A5A5A5A;

    #10;
    wr_en_in = 0;
    rs_1_addr_in = 5'd2;
    rs_2_addr_in = 5'd2;

    #10;
    wr_en_in = 1;
    rd_addr_in = 5'd3;
    rd_in = 32'h12345678;

    #10;
    wr_en_in = 0;
    rs_1_addr_in = 5'd3;
    rs_2_addr_in = 5'd3;
    #10;
    wr_en_in = 1;
    rd_addr_in = 5'd6;
    rd_in = 32'hF0F0F0F0;

    #10;
    wr_en_in = 0;
    rs_1_addr_in = 5'd6;
    rs_2_addr_in = 5'd6;
        #10;
    wr_en_in = 1;
    rd_addr_in = 5'd7;
    rd_in = 32'hAAAAAAAA;

    #10;
    wr_en_in = 0;
    rs_1_addr_in = 5'd7;
    rs_2_addr_in = 5'd7;

        #10;
    wr_en_in = 1;
    rd_addr_in = 5'd8;
    rd_in = 32'h55555555;

    #10;
    wr_en_in = 0;
    rs_1_addr_in = 5'd8;
    rs_2_addr_in = 5'd8;

    #10;
    wr_en_in = 1;
    rd_addr_in = 5'd9;
    rd_in = 32'hFFFFFFFF;

    #10;
    wr_en_in = 0;
    rs_1_addr_in = 5'd9;
    rs_2_addr_in = 5'd9;

    #10;
    wr_en_in = 1;
    rd_addr_in = 5'd10;
    rd_in = 32'h00000000;

    #10;
    wr_en_in = 0;
    rs_1_addr_in = 5'd10;
    rs_2_addr_in = 5'd10;
    #10;
    $stop;

end
endmodule
```

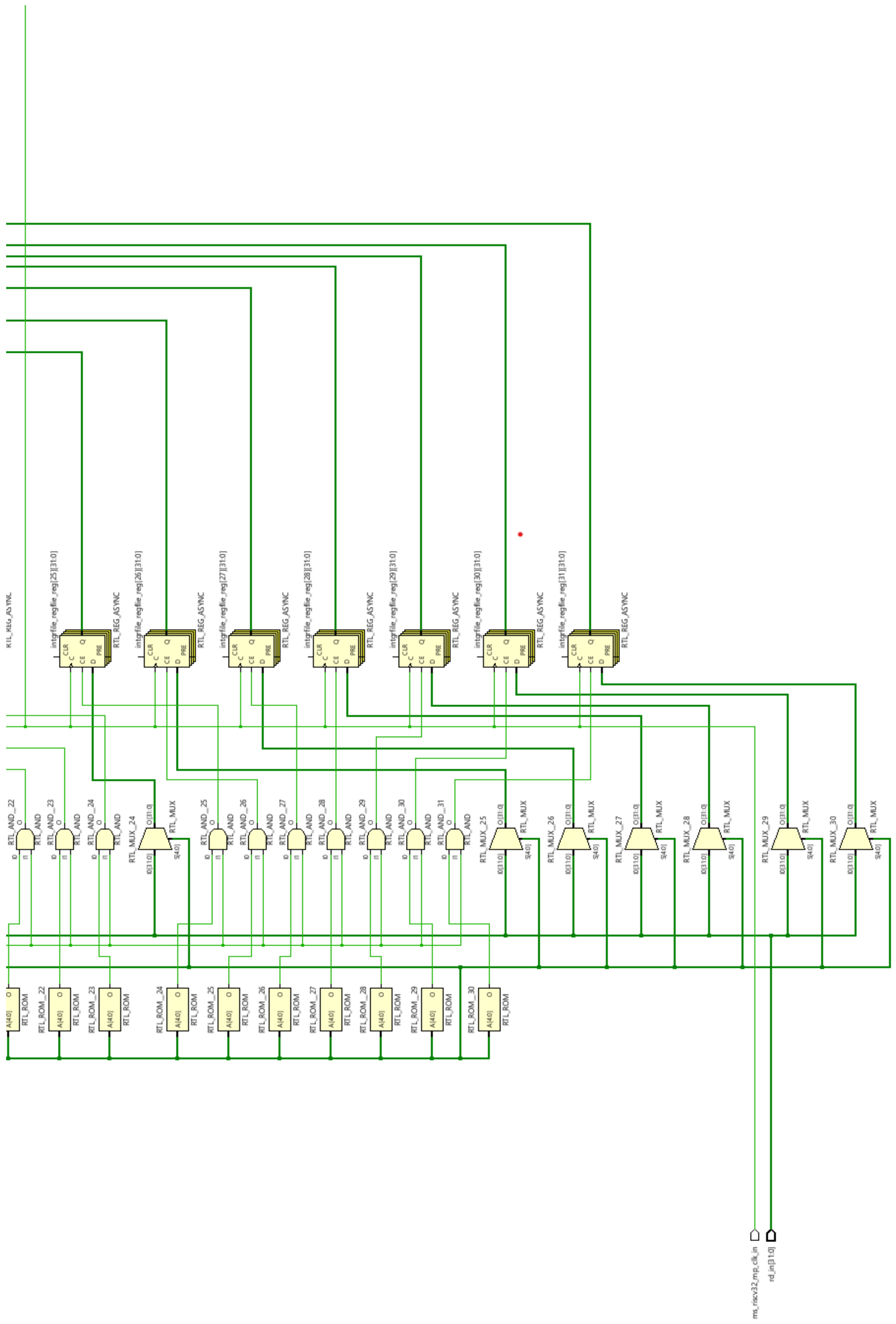
RTL / Micro Architecture - Integer File - msrv32_integer_file




rd_addr_in[4:0]
RTL_ROM
RTL_ROM_0
RTL_ROM_1
RTL_ROM_2
RTL_ROM_3
RTL_ROM_4
A[4:0]
O
RTL_AND_0
RTL_AND_1
RTL_AND_2
RTL_AND_3
RTL_AND_4
RTL_AND_5
RTL_AND
I0
I1
RTL_MUX_0
RTL_MUX_1
RTL_MUX_2
RTL_MUX_3
RTL_MUX_4
RTL_MUX
O[31:0]
I0[31:0]
S[4:0]
S=5'b00001
S=5'b00010
S=5'b00011
S=5'b00100
S=5'b00101
intgrfile_regfile_reg[0][31:0]
intgrfile_regfile_reg[1][31:0]
intgrfile_regfile_reg[2][31:0]
intgrfile_regfile_reg[3][31:0]
intgrfile_regfile_reg[4][31:0]
intgrfile_regfile_reg[5][31:0]
RTL_REG_ASYNC
CLR
C
CE
D
Q
PRE
n/c

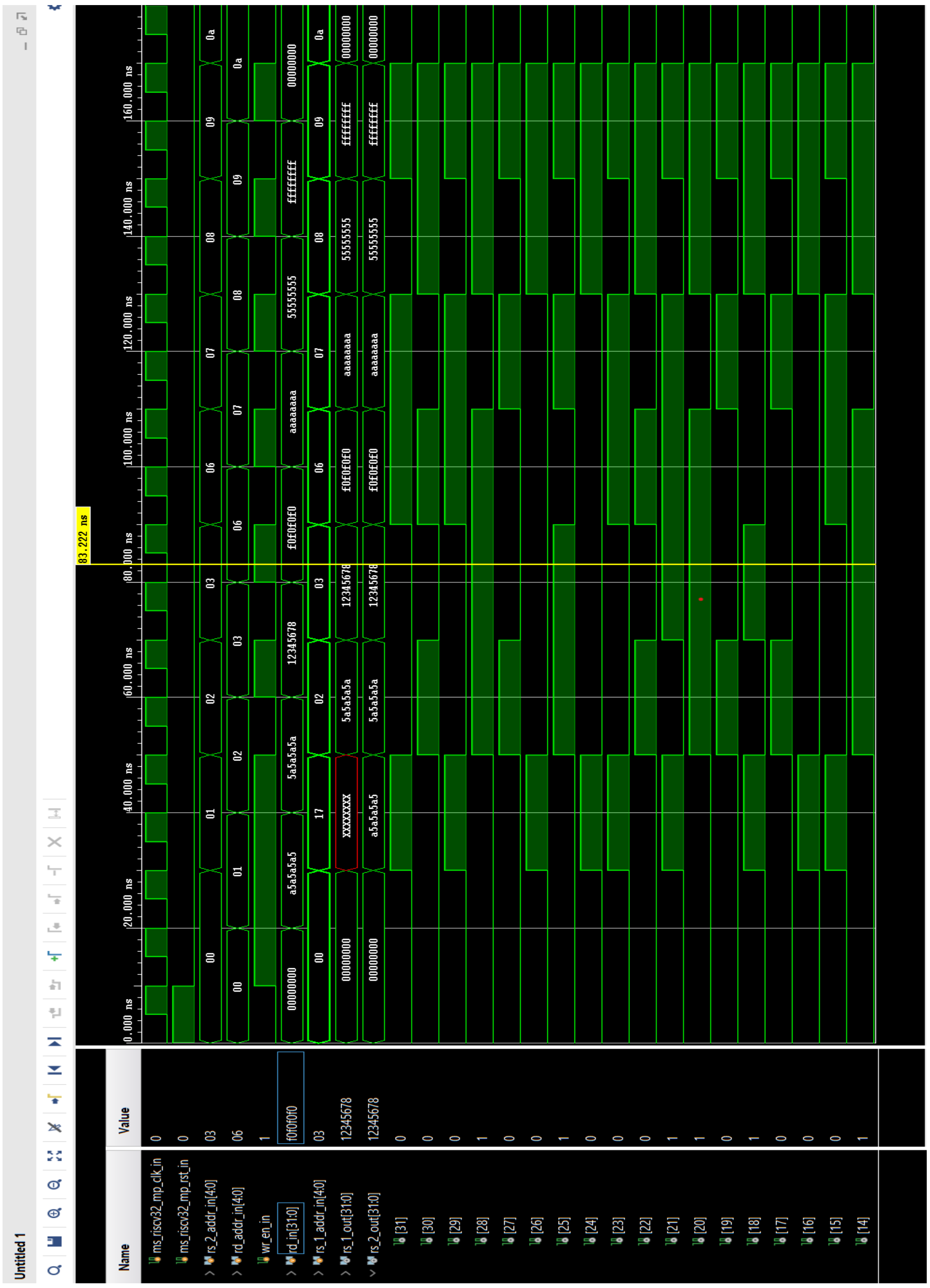

RTL_MUX
RTL_REG_ASYNC
RTL_MUX_14
RTL_ROM_14
RTL_AND_15
intgrfile_regfile_reg[15][31:0]
RTL_MUX_15
RTL_ROM_15
RTL_AND_16
intgrfile_regfile_reg[16][31:0]
RTL_MUX_16
intgrfile_regfile_reg[17][31:0]
RTL_ROM_16
RTL_AND_17
RTL_ROM_17
RTL_AND_18
intgrfile_regfile_reg[18][31:0]
RTL_MUX_17
RTL_ROM_18
RTL_AND_19
intgrfile_regfile_reg[19][31:0]
RTL_MUX_18
intgrfile_regfile_reg[20][31:0]
RTL_MUX_19
RTL_ROM_19
RTL_AND_20
RTL_MUX_20
intgrfile_regfile_reg[21][31:0]
RTL_ROM_20
RTL_AND_21
RTL_MUX_31
RTL_MUX_32
mux_rs1_en0_i
RTL_ROM
RTL_AND
O[31:0]
S[4:0]


rs_1_addr_in[4:0]
rs_2_addr_in[4:0]
wr_en_in
ms_riscv32_mp_rst_in
rs_1_out_reg[31:0]
rs_2_out_reg[31:0]
RTL_REG_ASYNC
RTL_MUX
RTL_AND
RTL_EQ
RTL_ROM
mux_rs1_en_i
mux_rs2_en_i
mux_rs1_en0_i
mux_rs2_en0_i
rs1_out_i
rs2_out_i
rs_1_out0_i
rs_2_out0_i
intgrfile_regfile_i
intgrfile_regfile_reg[21][31:0]
intgrfile_regfile_reg[22][31:0]
intgrfile_regfile_reg[23][31:0]
intgrfile_regfile_reg[24][31:0]
intgrfile_regfile_reg[25][31:0]
intgrfile_regfile_reg[26][31:0]
intgrfile_regfile_reg[27][31:0]
intgrfile_regfile_reg[28][31:0]



RTL_REG_ASYNC
intgrfile_regfile_reg[25][31:0]
intgrfile_regfile_reg[26][31:0]
intgrfile_regfile_reg[27][31:0]
intgrfile_regfile_reg[28][31:0]
intgrfile_regfile_reg[29][31:0]
intgrfile_regfile_reg[30][31:0]
intgrfile_regfile_reg[31][31:0]
RTL_AND_22
RTL_AND_23
RTL_AND_24
RTL_AND_25
RTL_AND_26
RTL_AND_27
RTL_AND_28
RTL_AND_29
RTL_AND_30
RTL_AND_31
RTL_MUX_24
RTL_MUX_25
RTL_MUX_26
RTL_MUX_27
RTL_MUX_28
RTL_MUX_29
RTL_MUX_30
RTL_ROM_22
RTL_ROM_23
RTL_ROM_24
RTL_ROM_25
RTL_ROM_26
RTL_ROM_27
RTL_ROM_28
RTL_ROM_29
RTL_ROM_30
ms_riscv32_mp_clk_in
rd_in[31:0]

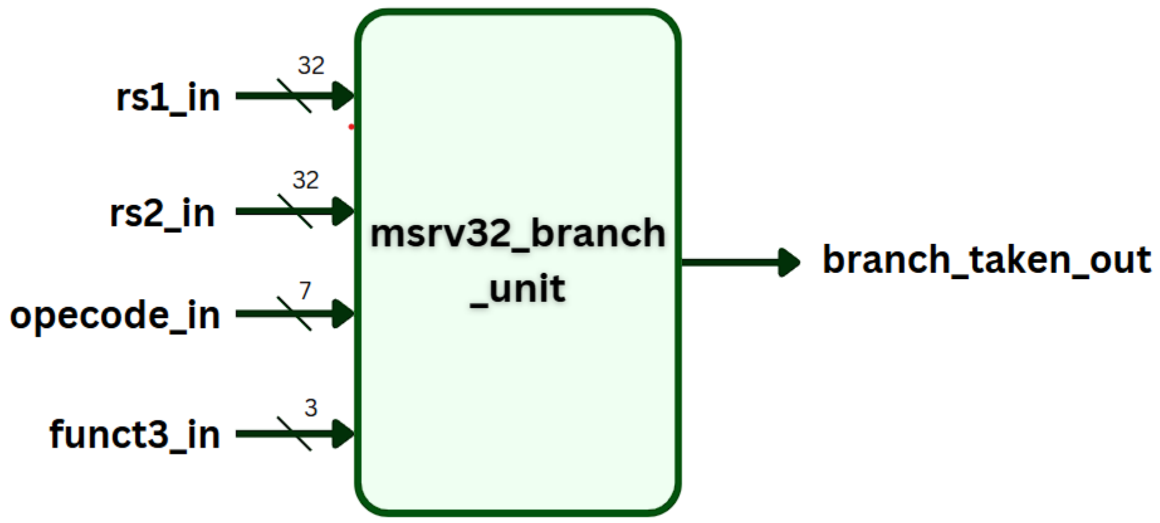
Wave Form - Integer File -msrv32_integer_file



Name	Value
ms_riscv32_mp_clk_in	0
ms_riscv32_mp_rst_in	0
rs_2_addr_in[4:0]	03
rd_addr_in[4:0]	06
wr_en_in	1
rd_in[31:0]	f0f0f0f0
rs_1_addr_in[4:0]	03
rs_1_out[31:0]	12345678
rs_2_out[31:0]	12345678
[31]	0
[30]	0
[29]	0
[28]	1
[27]	0
[26]	0
[25]	1
[24]	0
[23]	0
[22]	0
[21]	1
[20]	1
[19]	0
[18]	1
[17]	0
[16]	0
[15]	0
[14]	1

BRACH UNIT

Block Diagram



Block Pin

1) rs1_in (32) : 32-bit input data from the first source register.
2) rs2_in (32) : 32-bit input data from the second source register.
3) opcode_in (7) : 7-bit input opcode to specify the type of operation to be performed.
4) funct3_in (3) : 3-bit input function code to further define the operation specified by the opcode.
5) branch_taken_out : Output signal indicating whether a branch condition has been met and the branch is taken.

Examination – Branch Unit - msrv32_branch_unit

The msrv32_branch_unit module is responsible for handling branch instructions in the RISC-V 32I processor. It receives two 32-bit input values (rs1_in and rs2_in) from the source registers. The module also takes a 7-bit opcode (opcode_in) that specifies the type of branch instruction. Additionally, a 3-bit function code (funct3_in) is provided to further define the branch condition. The module compares the two input values based on the provided opcode and function code. If the specified branch condition is met, the module outputs a signal (branch_taken_out) indicating that the branch should be taken. This output signal is used by the processor to determine the next instruction to execute. Overall, the msrv32_branch_unit ensures that branch instructions are correctly processed, enabling the processor to change the flow of execution based on specific conditions.

Verilog Code

RTL /Design code – Branch Unit - msrv32_branch_unit

```
`timescale 1ns / 1ps
///////////////////////////////////////////////////////
//////////////////////////////
// Company: Maven Silicon - VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 29.06.2024 12:15:46
// Design Name: Branch Unit
// Module Name: msrv32_branch_unit
// Project Name: RISC -V 32I
// Target Devices:
// Tool Versions: Achyut's Vivado
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
///////////////////////////////////////////////////////
//////////////////////////////


module msrv32_branch_unit(rs1_in,rs2_in,
opecode_in,funct3_in,branch_taken_out);
input [31:0] rs1_in;
input [31:0] rs2_in;
input [6:0]  opecode_in;
input [2:0] funct3_in;
output reg branch_taken_out;
wire signed[31:0] rs1,rs2;
assign rs1= rs1_in;
assign rs2= rs2_in;
always @(*)
```

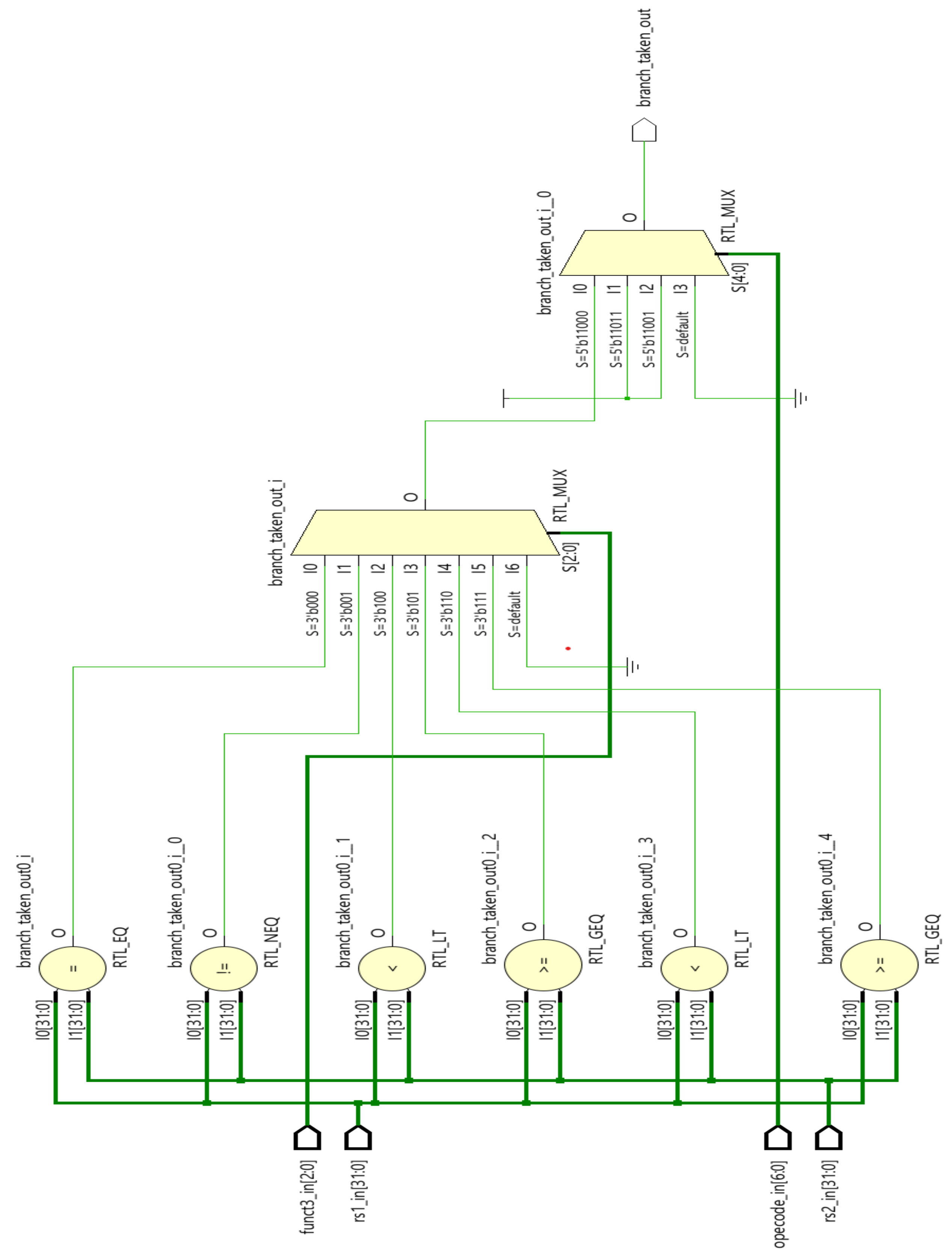
```
begin
    case(opecode_in[6:2])
    5'b11000: begin
        case(funct3_in)
            3'b000 : branch_taken_out = (rs1_in ==
rs2_in);
            3'b001 : branch_taken_out = (rs1_in !=
rs2_in);
            3'b100 : branch_taken_out = (rs1 <rs2);
            3'b101 : branch_taken_out = (rs1 >=rs2);
            3'b110 : branch_taken_out = (rs1_in <
rs2_in);
            3'b111 : branch_taken_out = (rs1_in >=
rs2_in);
            default :branch_taken_out =0;
         endcase
    end
    5'b11011: branch_taken_out=1; //jal
    5'b11001:branch_taken_out =1; //jalr
    default :branch_taken_out =0;
    endcase
end
endmodule
```

Simulation / Testbench code – Branch Unit - msrv32_branch_unit

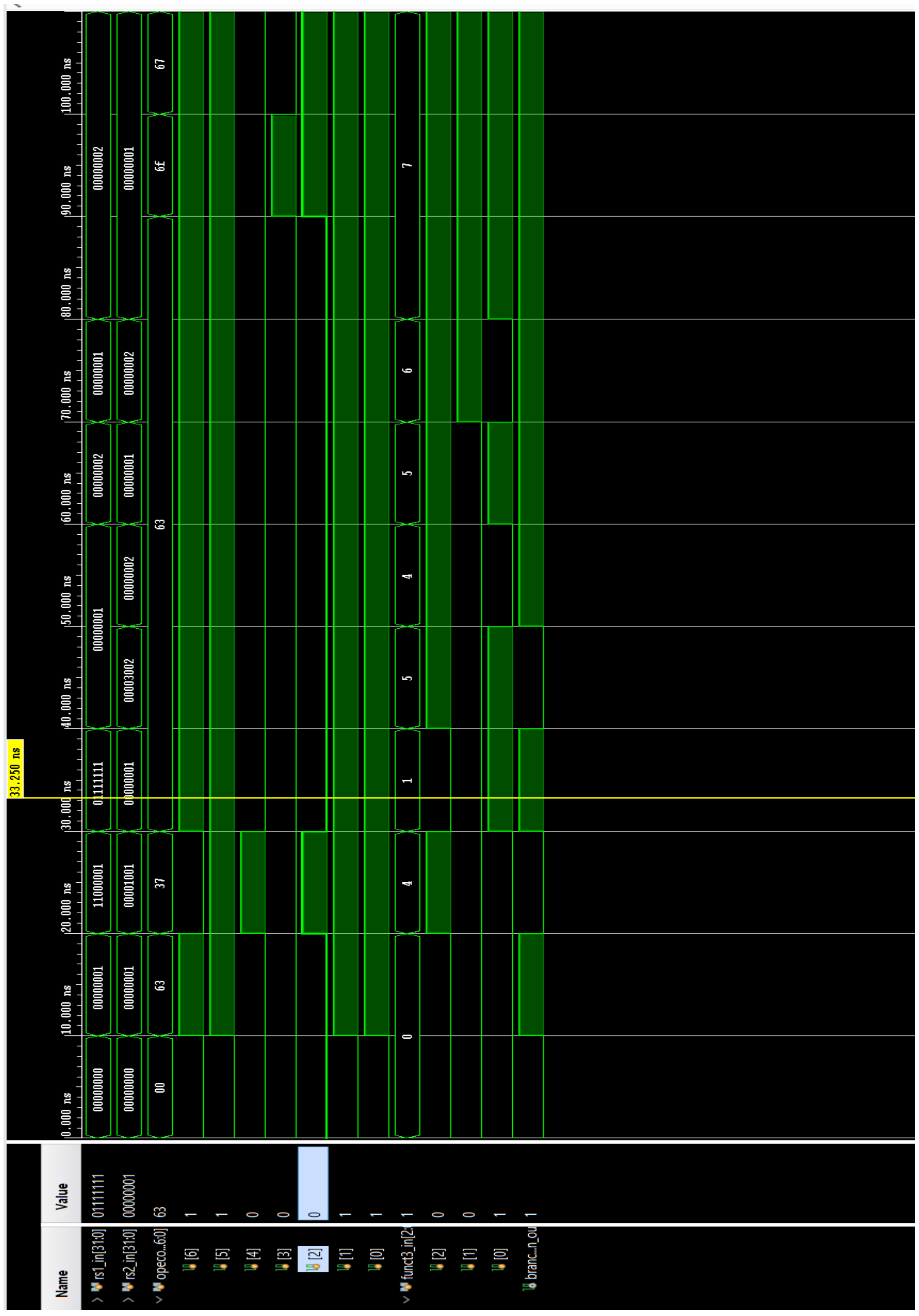
```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////
//////////////////////////////
// Company: Maven Silicon - VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 02.07.2024 20:44:37
// Design Name: Brach unit
// Module Name: msrv32_branch_unit_tb
// Project Name: RISC-V 32I
// Target Devices:
// Tool Versions: Achyut's Vivado
// Description:
//
// Dependencies:
```

```
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////
/////////////////////////////


module msrv32_branch_unit_tb;
reg [31:0] rs1_in;
reg [31:0] rs2_in;
reg [6:0] opecode_in;
reg [2:0] funct3_in;

// Outputs
wire branch_taken_out;

// Instantiate the Unit Under Test (UUT)
msrv32_branch_unit uut (
    .rs1_in(rs1_in),
    .rs2_in(rs2_in),
    .opecode_in(opecode_in),
    .funct3_in(funct3_in),
    .branch_taken_out(branch_taken_out)
);

initial begin
    // Initialize Inputs
    rs1_in = 0;
    rs2_in = 0;
    opecode_in = 0;
    funct3_in = 0;
    #10;

    rs1_in = 32'h00000001;
    rs2_in = 32'h00000001;
    opecode_in = 7'b1100011; // opcode for branch
instructions
    funct3_in = 3'b000; // BEQ
    #10;
    rs1_in = 32'h11000001;
    rs2_in = 32'h0001001;
    opecode_in = 7'b110111; // opcode for branch
instructions
    funct3_in = 3'b100; // BEQ
    #10;
    rs1_in = 32'h01111111;
    rs2_in = 32'h00000001;
    opecode_in = 7'b1100011; // opcode for branch
instructions
    funct3_in = 3'b001; // BEQ
    #10;
    rs1_in = 32'h00000001;
    rs2_in = 32'h00003002;
    opecode_in = 7'b1100011;
    funct3_in = 3'b101; // BNE
    #10;
    rs1_in = 32'h00000001;
    rs2_in = 32'h00000002;
    opecode_in = 7'b1100011;
    funct3_in = 3'b100; // BLT
    #10;
    rs1_in = 32'h00000002;
    rs2_in = 32'h00000001;
    opecode_in = 7'b1100011;
    funct3_in = 3'b101;
    #10;
    rs1_in = 32'h00000001;
    rs2_in = 32'h00000002;
    opecode_in = 7'b1100011; // opcode for branch
instructions
    funct3_in = 3'b110;
    #10;
    rs1_in = 32'h00000002;
    rs2_in = 32'h00000001;
    opecode_in = 7'b1100011; // opcode for branch
instructions
    funct3_in = 3'b111; // BGEU
    #10;
    opecode_in = 7'b1101111; // opcode for JAL
    #10;
    opecode_in = 7'b1100111; // opcode for JALR
    #10;
    $stop;
    end
endmodule
```

RTL / Micro Architecture – Branch Unit - msrv32_branch_unit

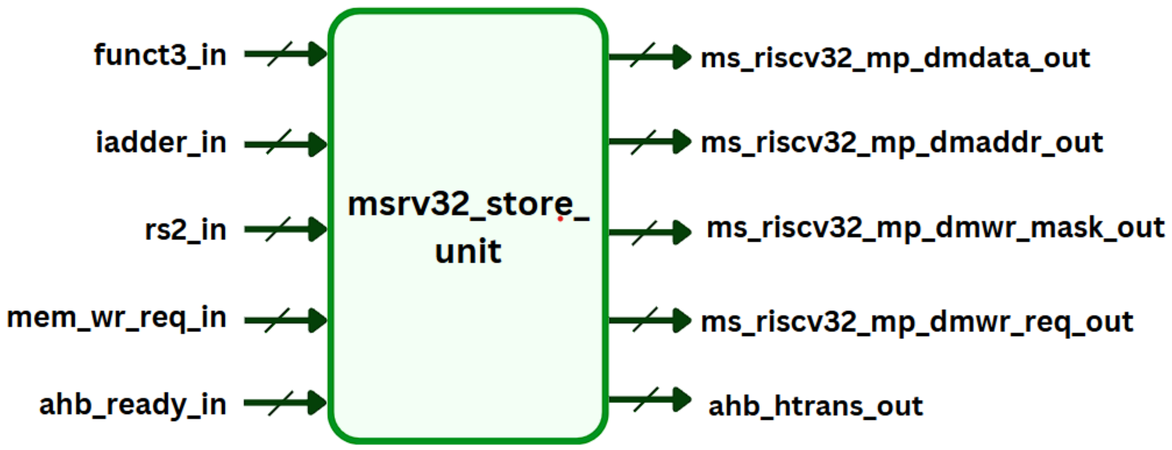


Wave Form – Branch Unit - msrv32_branch_unit



STORE UNIT

Block Unit



Block Pins

1) funct3_in : connected to the funct3 instruction field. Indicates the data size. (byte, half word or word).([1:0] bit from instruction decoder).
2) iadder_in : Contains the address (possibly unaligned) where the data must be written (from Immediate adder)
3) ahb_ready_in : Active high signal when asserted, updates the ms_riscv32_mp_data_out.
4) rs2_in : connected to Integer Register file source 2. Contains the data to be written (possibly in the right position).from integer file.
5) mem_wr_req_in : Connected to the memory in which data to be write (from decoder).
6) ms_riscv32_mp_dmdata_out : contains the data to be written in the right position.(output of core).
7) ms_riscv32_mp_dmaddr_out : contains the address (aligned) where the data must be written.
8) ms_riscv32_mp_dmwr_mask_out : A bitmask that indicates which bytes of data_out must be written.
9) ms_riscv32_mp_dmwr_req_out : write enable pin for external memory.
10) ahb_htrans_out : This signal signifies if the transfer is in progress or completed.

Explanation – Store unit – msrv32_store_unit

The msrv32_store_unit is a vital component in the RISC-V 32I processor, ensuring that data is correctly and efficiently written to memory. It handles various control signals, prepares data, calculates addresses, and coordinates with the AHB bus to perform store operations.

Verilog Code

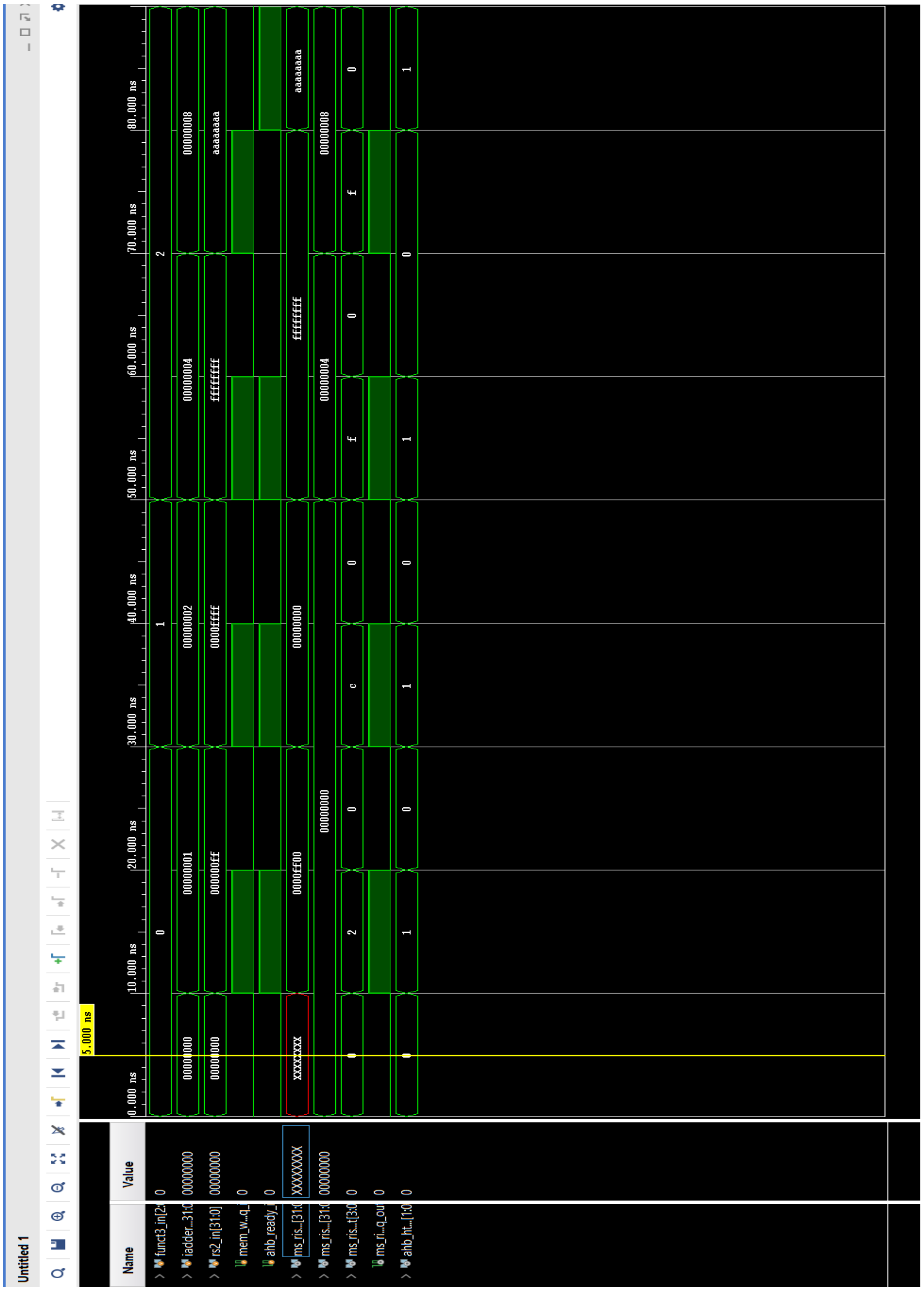
Design /RTL Code – Store unit – msrv32_store_unit

```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////
//////////////////////////////
// Company: Maven Silicon - VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 01.07.2024 17:39:49
// Design Name: Store Unit
// Module Name: msrv32_store_unit
// Project Name: RISC -V 32I
// Target Devices:
// Tool Versions: Achyut's Vivado
// Description:
// Store unit RTL / Design code
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////
//////////////////////////////

module msrv32_store_unit(funct3_in, iadder_in,
rs2_in, mem_wr_req_in, ahb_ready_in,
ms_riscv32_mp_dmdata_out, ms_riscv32_mp_dmaddr_out,
ms_riscv32_mp_dmwr_mask_out,
ms_riscv32_mp_dmwr_req_out,ahb_htrans_out);

input [2:0] funct3_in;
input [31:0] iadder_in;
input [31:0] rs2_in;
input mem_wr_req_in;
input ahb_ready_in;

output [31:0] ms_riscv32_mp_dmdata_out;
output [31:0] ms_riscv32_mp_dmaddr_out;
output reg [3:0] ms_riscv32_mp_dmwr_mask_out;
output ms_riscv32_mp_dmwr_req_out;
output [1:0] ahb_htrans_out;
reg [3:0] byt_wr_mask;
reg [31:0] byt_data_out;
reg [31:0] data_out;
wire [3:0] half_wr_mask;
assign ahb_htrans_out = (ahb_ready_in)? 2'b01 :2'b00;
assign half_wr_mask = (iadder_in[1])?
{{2{mem_wr_req_in}},2'b0} :
{2'b0,{2{mem_wr_req_in}}};
assign half_data_out = (iadder_in[1])? {rs2_in[15:0],
{16{1'b0}}} : {{16{1'b0}},rs2_in[15:0]};
always @(*)
begin
    case(iadder_in[1:0])
        2'b00 : begin
                byt_wr_mask <= {3'b0,mem_wr_req_in};
                byt_data_out <= {{24{1'b0}},
rs2_in[7:0]};
            end
        2'b01 : begin
                byt_wr_mask <=
{2'b0,mem_wr_req_in,1'b0};
                byt_data_out <= {{16{1'b0}},
rs2_in[7:0],{8{1'b0}}};
            end
        2'b10 : begin
                byt_wr_mask <=
{1'b0,mem_wr_req_in,2'b0};
                byt_data_out <= {{8{1'b0}},
rs2_in[7:0],{16{1'b0}}};
            end
        2'b11 : begin
                byt_wr_mask <= {mem_wr_req_in,3'b00};
                byt_data_out <=
{rs2_in[7:0],{24{1'b0}}};
            end
    endcase
    case(funct3_in[1:0])
        2'b00 : begin
                ms_riscv32_mp_dmwr_mask_out <=
byt_wr_mask ;
                data_out <= byt_data_out;
            end
        2'b01 : begin
            ms_riscv32_mp_dmwr_mask_out <=
half_wr_mask;
            data_out <= half_data_out;
        end
        2'b10 : begin
            ms_riscv32_mp_dmwr_mask_out <=
{4{mem_wr_req_in}};
            data_out <= rs2_in;
        end
        2'b11 : begin
            ms_riscv32_mp_dmwr_mask_out <=
{4{mem_wr_req_in}};
            data_out <= rs2_in;
        end
    endcase
end
assign ms_riscv32_mp_dmdata_out =(ahb_ready_in)?
data_out: ms_riscv32_mp_dmdata_out;
assign ms_riscv32_mp_dmaddr_out =
{iadder_in[31:2],2'b00};
assign ms_riscv32_mp_dmwr_req_out = mem_wr_req_in;

endmodule
```

Simulation Code – Store unit – msrv32_store_unit

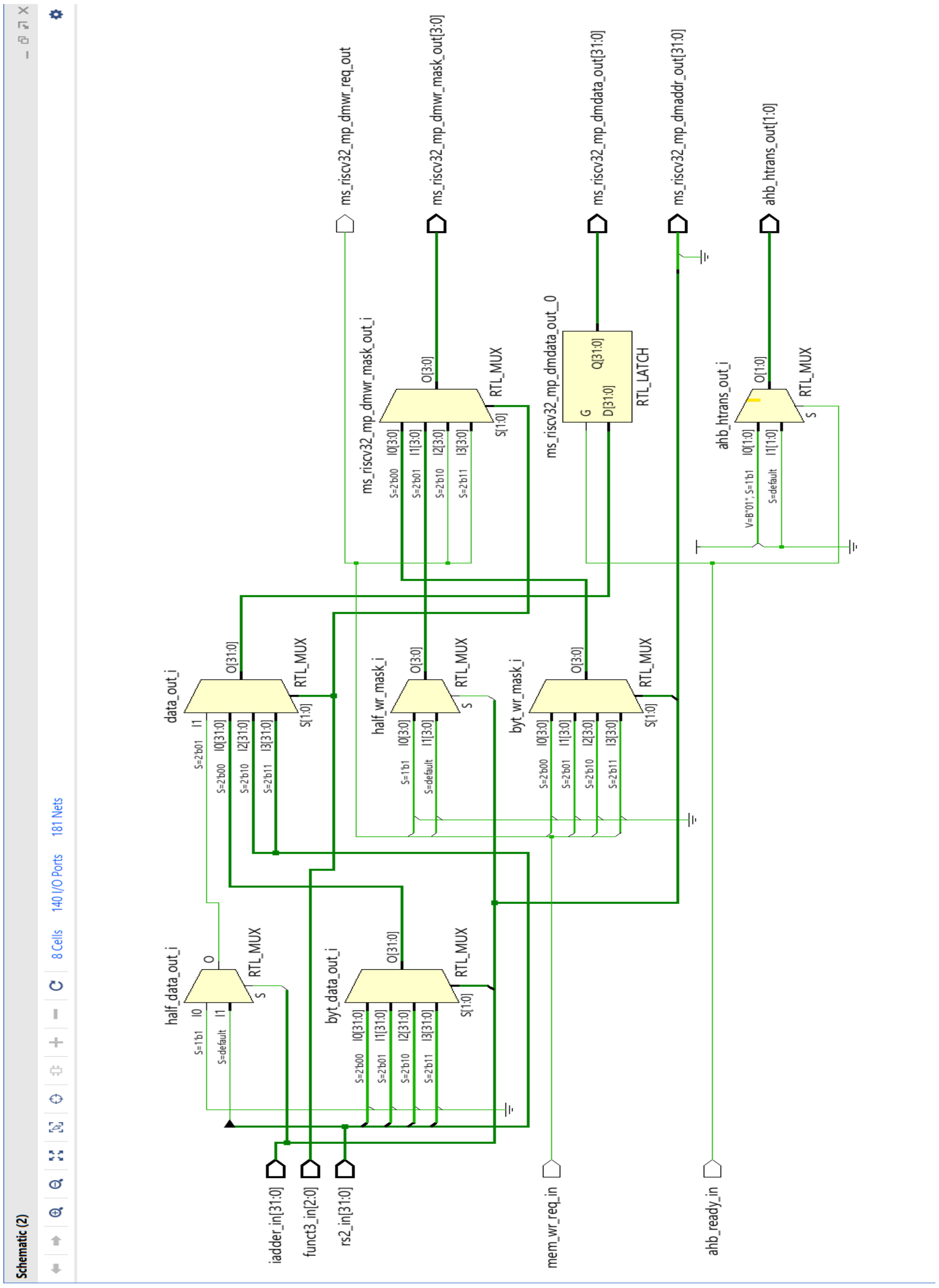
```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////
//////////////////////////////
// Company: Maven Silicon -VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 03.07.2024 09:50:02
// Design Name: Store Unit
// Module Name: msrv32_store_unit_tb
// Project Name: RISC V 32I
// Target Devices:
// Tool Versions: Achyut's Vovado
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////
//////////////////////////////


module msrv32_store_unit_tb;
reg [2:0] funct3_in;
reg [31:0] iadder_in;
reg [31:0] rs2_in;
reg mem_wr_req_in;
reg ahb_ready_in;

// Outputs
wire [31:0] ms_riscv32_mp_dmdata_out;
wire [31:0] ms_riscv32_mp_dmaddr_out;
wire [3:0] ms_riscv32_mp_dmwr_mask_out;
wire ms_riscv32_mp_dmwr_req_out;
wire [1:0] ahb_htrans_out;

// Instantiate the Unit Under Test (UUT)
msrv32_store_unit uut (
    .funct3_in(funct3_in),
    .iadder_in(iadder_in),
    .rs2_in(rs2_in),
    .mem_wr_req_in(mem_wr_req_in),
    .ahb_ready_in(ahb_ready_in),

.ms_riscv32_mp_dmdata_out(ms_riscv32_mp_dmdata_out),

.ms_riscv32_mp_dmaddr_out(ms_riscv32_mp_dmaddr_out),

.ms_riscv32_mp_dmwr_mask_out(ms_riscv32_mp_dmwr_mask_
out),
.ms_riscv32_mp_dmwr_req_out(ms_riscv32_mp_dmwr_req_ou
t),
    .ahb_htrans_out(ahb_htrans_out)
);

initial begin
    // Initialize Inputs
    funct3_in = 0;
    iadder_in = 0;
    rs2_in = 0;
    mem_wr_req_in = 0;
    ahb_ready_in = 0;
    #10;
     funct3_in = 3'b000; // SB
    iadder_in = 32'h00000001; // Address offset
    rs2_in = 32'h000000FF; // Data to be written
    mem_wr_req_in = 1;
    ahb_ready_in = 1;
    #10;
    mem_wr_req_in = 0;
    ahb_ready_in = 0;
    #10;
     funct3_in = 3'b001; // SH
    iadder_in = 32'h00000002; // Address offset
    rs2_in = 32'h0000FFFF; // Data to be written
    mem_wr_req_in = 1;
    ahb_ready_in = 1;
    #10;
    mem_wr_req_in = 0;
    ahb_ready_in = 0;
    #10;
    funct3_in = 3'b010; // SW
    iadder_in = 32'h00000004; // Address offset
    rs2_in = 32'hFFFFFFFF; // Data to be written
    mem_wr_req_in = 1;
    ahb_ready_in = 1;
    #10;
    mem_wr_req_in = 0;
    ahb_ready_in = 0;
    #10;
    funct3_in = 3'b010; // SW
    iadder_in = 32'h00000008; // Address offset
    rs2_in = 32'hAAAAAAAA; // Data to be written
    mem_wr_req_in = 1;
    ahb_ready_in = 0;
    #10;
    mem_wr_req_in = 0;
    ahb_ready_in = 1;
    #10;
    $stop;
end
endmodule
```

Wave form – Store unit – msrv32_store_unit



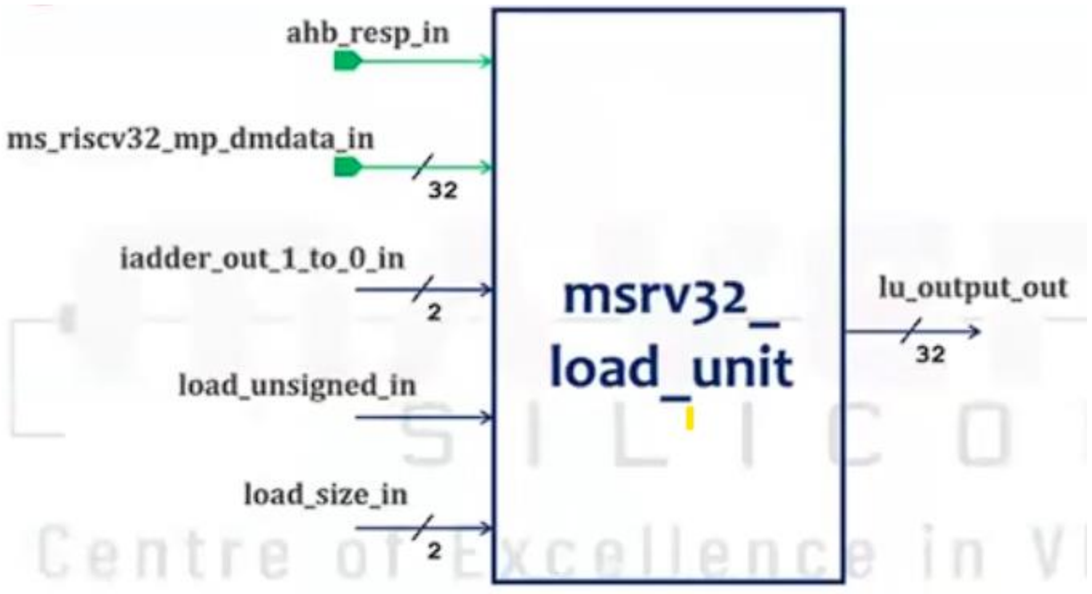
Name	Value
funct3_in[2:0]	0
iadder...31:0]	00000000
rs2_in[31:0]	00000000
mem_w...q_i	0
ahb_ready_i	0
ms_ris...[31:0]	XXXXXXXX
ms_ris...[31:0]	00000000
ms_ris...t[3:0]	0
ms_ri...q_out	0
ahb_ht...[1:0]	0

RTL Architecture – Store unit – msrv32_store_unit



LOAD UNIT

Block Diagram



Signals

1. load_size_in : Connected to the two least significant bits of the fuct3 instruction field from register block 2.
2. load_unsigned_in : Connected to the most significant bit of the funct3 from instruction field from register block 2.
3. Ahb_resp_in : Active Low signal used to load the external data from memory.
4. Ms_riscv32_mp_dmdata_in : 32 bit word read from memory. (from external memory).
5. Iadder_out_1_0_in : Indicates the byte half word position in data_in used only with load byte half word instructions.
6. Lu_output_out : 32 bit value to be written in the integer Register file.

Explanation

The msrv32_load_unit plays a crucial role in handling load operations within the RISC-V 32I processor. It efficiently processes data fetched from memory, ensuring that it is correctly loaded into the processor's registers. The unit supports various data sizes, including byte, halfword, and word, and can handle both signed and unsigned load operations. By processing data accurately based on the load type and size, it maintains the integrity of data transfers within the processor. This unit is vital for the overall functionality of the processor, as it ensures that data is available for subsequent operations and computations. Its efficient design contributes to the processor's performance and reliability, making it an essential component of the RISC-V 32I architecture.

Verilog code

RTL Design code - Load Unit - msrv32_load_unit

```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////
//////////////////////////////
// Company: Maven Silicon - VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 03.07.2024 19:13:20
// Design Name: Load Unit
// Module Name: msrv32_load_unit
// Project Name: RISC V 32I
// Target Devices:
// Tool Versions: Achyut’s Vivado
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////
//////////////////////////////


module msrv32_load_unit(ahb_resp_in,
ms_riscv32_mp_dmdata_in,
iadder_out_1_to_0_in,load_unsigned_in, load_size_in,
lu_output_out);
input ahb_resp_in;
input [31:0] ms_riscv32_mp_dmdata_in;
input [1:0] iadder_out_1_to_0_in;
input load_unsigned_in;
input [1:0] load_size_in;
```

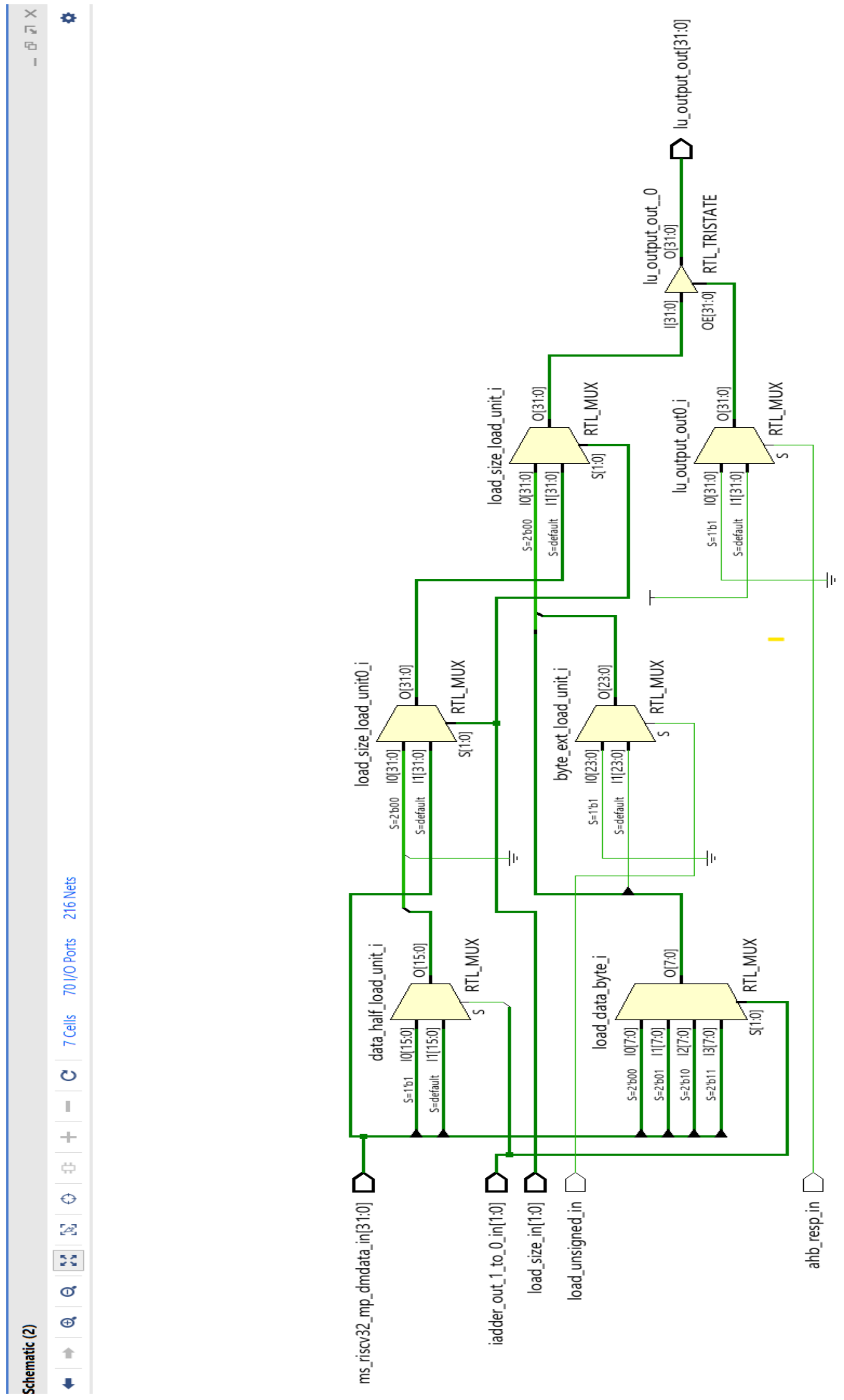
```
output [31:0] lu_output_out;
wire [15:0] data_half_load_unit;
wire [23:0] byte_ext_load_unit;
wire [15:0] half_ext_load_unit;
wire [31:0] load_size_load_unit;
reg [7:0] load_data_byte;
assign  data_half_load_unit =
(iadder_out_1_to_0_in[1])?
ms_riscv32_mp_dmdata_in[31:16] :
ms_riscv32_mp_dmdata_in[15:0];

always @(*)
    begin
        case(iadder_out_1_to_0_in)
            2'b00 :
load_data_byte<=ms_riscv32_mp_dmdata_in[7:0];
            2'b01 :
load_data_byte<=ms_riscv32_mp_dmdata_in[15:8];
            2'b10 :
load_data_byte<=ms_riscv32_mp_dmdata_in[23:16];
            2'b11 :
load_data_byte<=ms_riscv32_mp_dmdata_in[31:24];
        endcase
    end
assign byte_ext_load_unit =(load_unsigned_in) ? 24'b0
: {24{load_data_byte[7]}};
assign half_ext_load_unit =(load_unsigned_in) ? 16'b0
: {16{load_data_byte[15]}};
assign load_size_load_unit =(load_size_in==2'b00) ?
{byte_ext_load_unit,load_data_byte} :
(load_size_in==2'b00)? {half_ext_load_unit,
data_half_load_unit}: (load_size_in==2'b10)?
ms_riscv32_mp_dmdata_in :  ms_riscv32_mp_dmdata_in;
assign lu_output_out = (ahb_resp_in) ? 32'bz:
load_size_load_unit;



endmodule
```

Simulation code - Load Unit - msrv32_load_unit

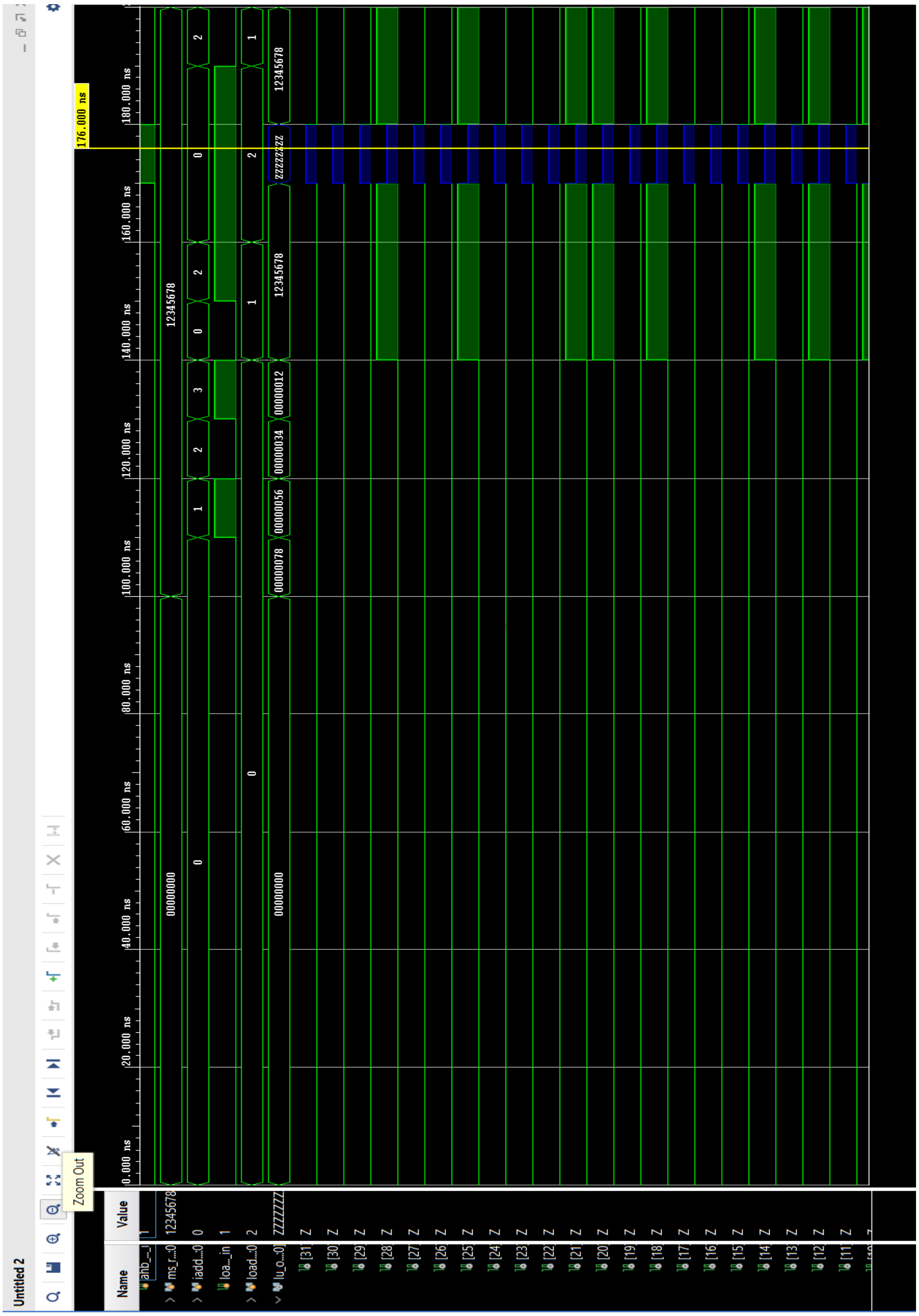
```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////
/////////////////////////////
// Company: Maven Silicon - VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 12.07.2024 09:42:33
// Design Name: Load Unit
// Module Name: msrv32_load_unit_tb
// Project Name: RISC V32I
// Target Devices:
// Tool Versions: Achyut viavdo
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////
/////////////////////////////


module msrv32_load_unit_tb;
reg ahb_resp_in;
reg [31:0] ms_riscv32_mp_dmdata_in;
reg [1:0] iadder_out_1_to_0_in;
reg load_unsigned_in;
reg [1:0] load_size_in;
wire [31:0] lu_output_out;

msrv32_load_unit uut (
    .ahb_resp_in(ahb_resp_in),

.ms_riscv32_mp_dmdata_in(ms_riscv32_mp_dmdata_in),
    .iadder_out_1_to_0_in(iadder_out_1_to_0_in),
    .load_unsigned_in(load_unsigned_in),
    .load_size_in(load_size_in),
    .lu_output_out(lu_output_out)
);
```

```
initial begin
    ahb_resp_in = 0;
    ms_riscv32_mp_dmdata_in = 0;
    iadder_out_1_to_0_in = 0;
    load_unsigned_in = 0;
    load_size_in = 0;
    #100;

    ms_riscv32_mp_dmdata_in = 32'h12345678;
    iadder_out_1_to_0_in = 2'b00;
    load_unsigned_in = 0;
    load_size_in = 2'b00;
    #10;

    iadder_out_1_to_0_in = 2'b01;
    load_unsigned_in = 1;
    #10;

    iadder_out_1_to_0_in = 2'b10;
    load_unsigned_in = 0;
    #10;

    iadder_out_1_to_0_in = 2'b11;
    load_unsigned_in = 1;
    #10;

    iadder_out_1_to_0_in = 2'b00;
    load_unsigned_in = 0;
    load_size_in = 2'b01;
    #10;

    iadder_out_1_to_0_in = 2'b10;
    load_unsigned_in = 1;
    #10;

    iadder_out_1_to_0_in = 2'b00;
    load_size_in = 2'b10;
    #10;

    ahb_resp_in = 1;
    #10;

    ahb_resp_in = 0;
    #10;

    iadder_out_1_to_0_in = 2'b10;
    load_unsigned_in = 0;
    load_size_in = 2'b01;
    #10;

    $stop;
end

endmodule
```

RTL Design- Load Unit - msrv32_load_unit

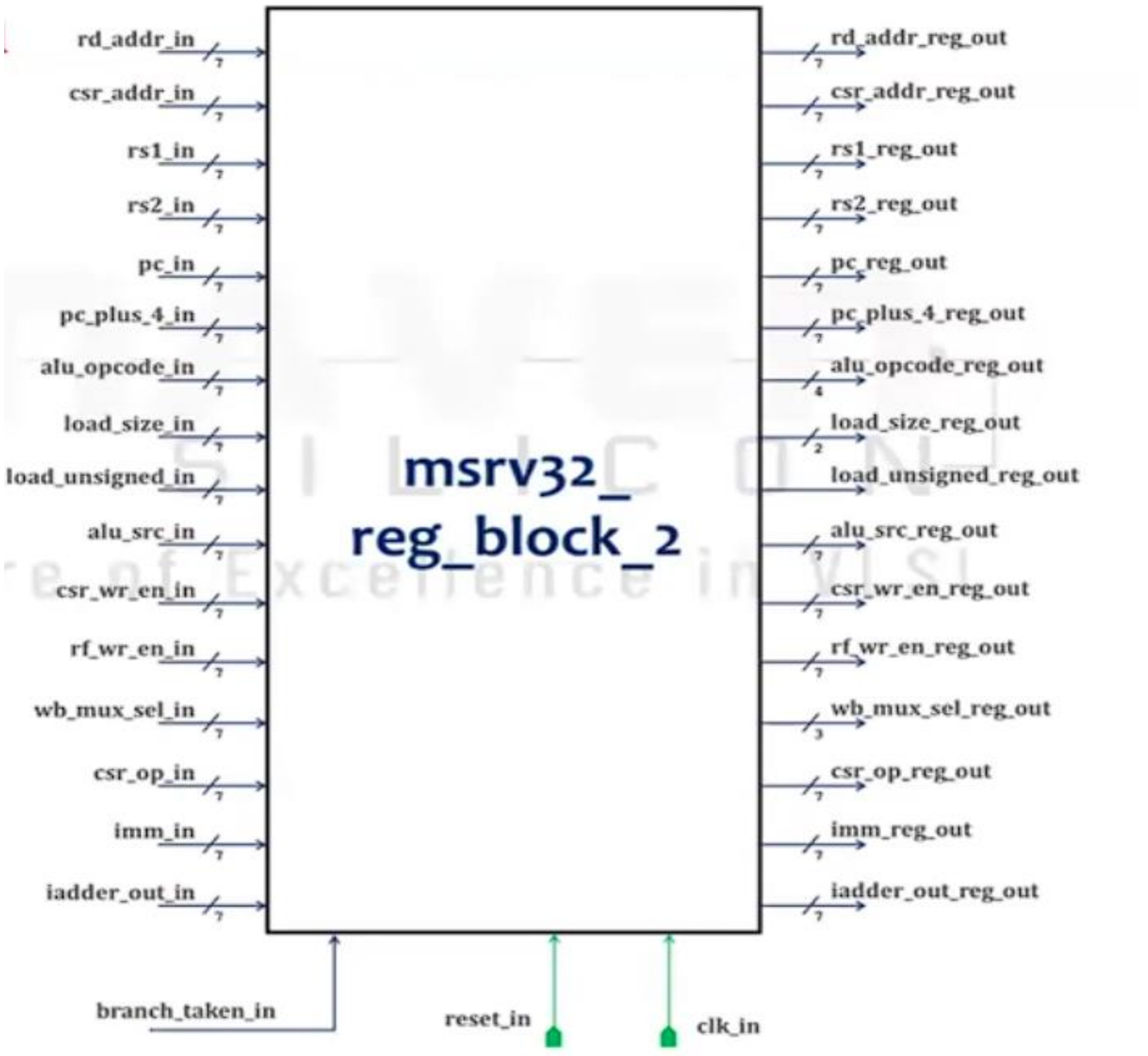


Wave form - Load Unit - msrv32_load_unit



REG BLOCK 2

Block Diagram



Explanation

All the input signals is store with respect to the clock

Reset_in : is to reset all the values

Clk_in : clock pulses to the registers inside the block.

Verilog code

Design Code – Reg block 2 - msrv32_reg_block_2

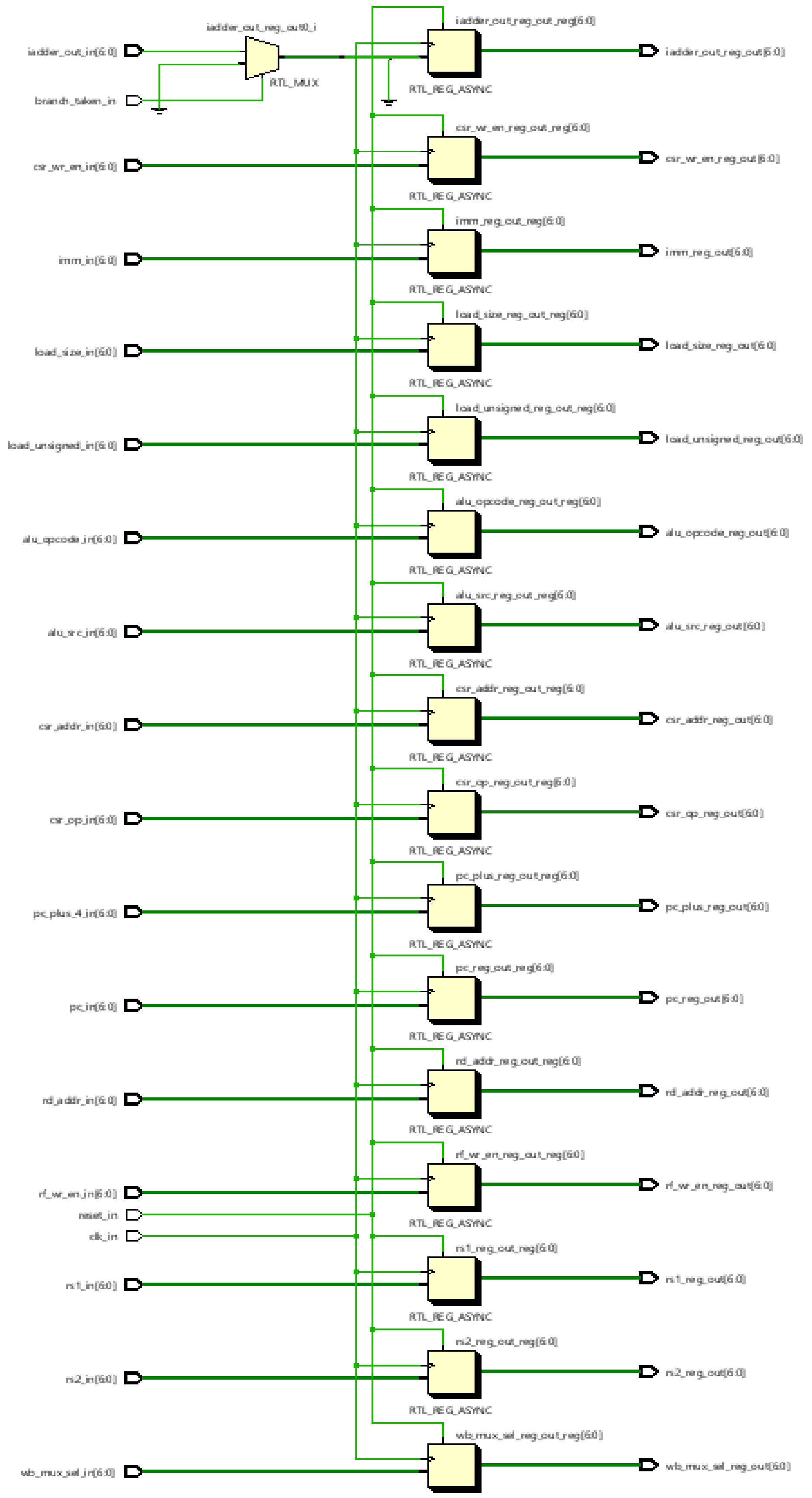
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////
//////////////////////////////
// Company: Maven Silicon - VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 03.07.2024 20:35:55
// Design Name: Reg block 2
// Module Name: msrv32_reg_block_2
// Project Name: RISC V 32I
// Target Devices:
// Tool Versions: Achyut's Vivado
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////
//////////////////////////////

module msrv32_reg_block_2 (
    input [6:0] rd_addr_in,
    input [6:0] csr_addr_in,
    input [6:0] rs1_in,
    input [6:0] rs2_in,
    input [6:0] pc_in,
    input [6:0] pc_plus_4_in,
    input [6:0] alu_opcode_in,
    input [6:0] load_size_in,
    input [6:0] load_unsigned_in,
    input [6:0] alu_src_in,
    input [6:0] csr_wr_en_in,
    input [6:0] rf_wr_en_in,
    input [6:0] wb_mux_sel_in,
    input [6:0] csr_op_in,
    input [6:0] imm_in,
    input [6:0] iadder_out_in,
    input branch_taken_in,
    input reset_in,
    input clk_in,
    output reg [6:0] rd_addr_reg_out,
    output reg [6:0] csr_addr_reg_out,
    output reg [6:0] rs1_reg_out,
    output reg [6:0] rs2_reg_out,
    output reg [6:0] pc_reg_out,
    output reg [6:0] pc_plus_reg_out,
    output reg [6:0] alu_opcode_reg_out,
    output reg [6:0] load_size_reg_out,
    output reg [6:0] load_unsigned_reg_out,
    output reg [6:0] alu_src_reg_out,
    output reg [6:0] csr_wr_en_reg_out,
    output reg [6:0] rf_wr_en_reg_out,
    output reg [6:0] wb_mux_sel_reg_out,
    output reg [6:0] csr_op_reg_out,
    output reg [6:0] imm_reg_out,
    output reg [6:0] iadder_out_reg_out
);

always @(posedge clk_in or posedge reset_in) begin
    if (reset_in) begin
        rd_addr_reg_out <= 7'b0;
        csr_addr_reg_out <= 7'b0;
        rs1_reg_out <= 7'b0;
        rs2_reg_out <= 7'b0;
        pc_reg_out <= 7'b0;
        pc_plus_reg_out <= 7'b0;
        alu_opcode_reg_out <= 7'b0;
        load_size_reg_out <= 7'b0;
        load_unsigned_reg_out <= 7'b0;
        alu_src_reg_out <= 7'b0;
        csr_wr_en_reg_out <= 7'b0;
        rf_wr_en_reg_out <= 7'b0;
        wb_mux_sel_reg_out <= 7'b0;
        csr_op_reg_out <= 7'b0;
        imm_reg_out <= 7'b0;
        iadder_out_reg_out <= 7'b0;
    end else begin
        rd_addr_reg_out <= rd_addr_in;
        csr_addr_reg_out <= csr_addr_in;
        rs1_reg_out <= rs1_in;
        rs2_reg_out <= rs2_in;
        pc_reg_out <= pc_in;
        pc_plus_reg_out <= pc_plus_4_in;
        alu_opcode_reg_out <= alu_opcode_in;
        load_size_reg_out <= load_size_in;
        load_unsigned_reg_out <= load_unsigned_in;
        alu_src_reg_out <= alu_src_in;
        csr_wr_en_reg_out <= csr_wr_en_in;
        rf_wr_en_reg_out <= rf_wr_en_in;
        wb_mux_sel_reg_out <= wb_mux_sel_in;
        csr_op_reg_out <= csr_op_in;
        imm_reg_out <= imm_in;
        iadder_out_reg_out <= (branch_taken_in) ?
1'b0 : iadder_out_in[0];
    end
end

endmodule
```

Simulation – Reg block 2 - msrv32_reg_block_2

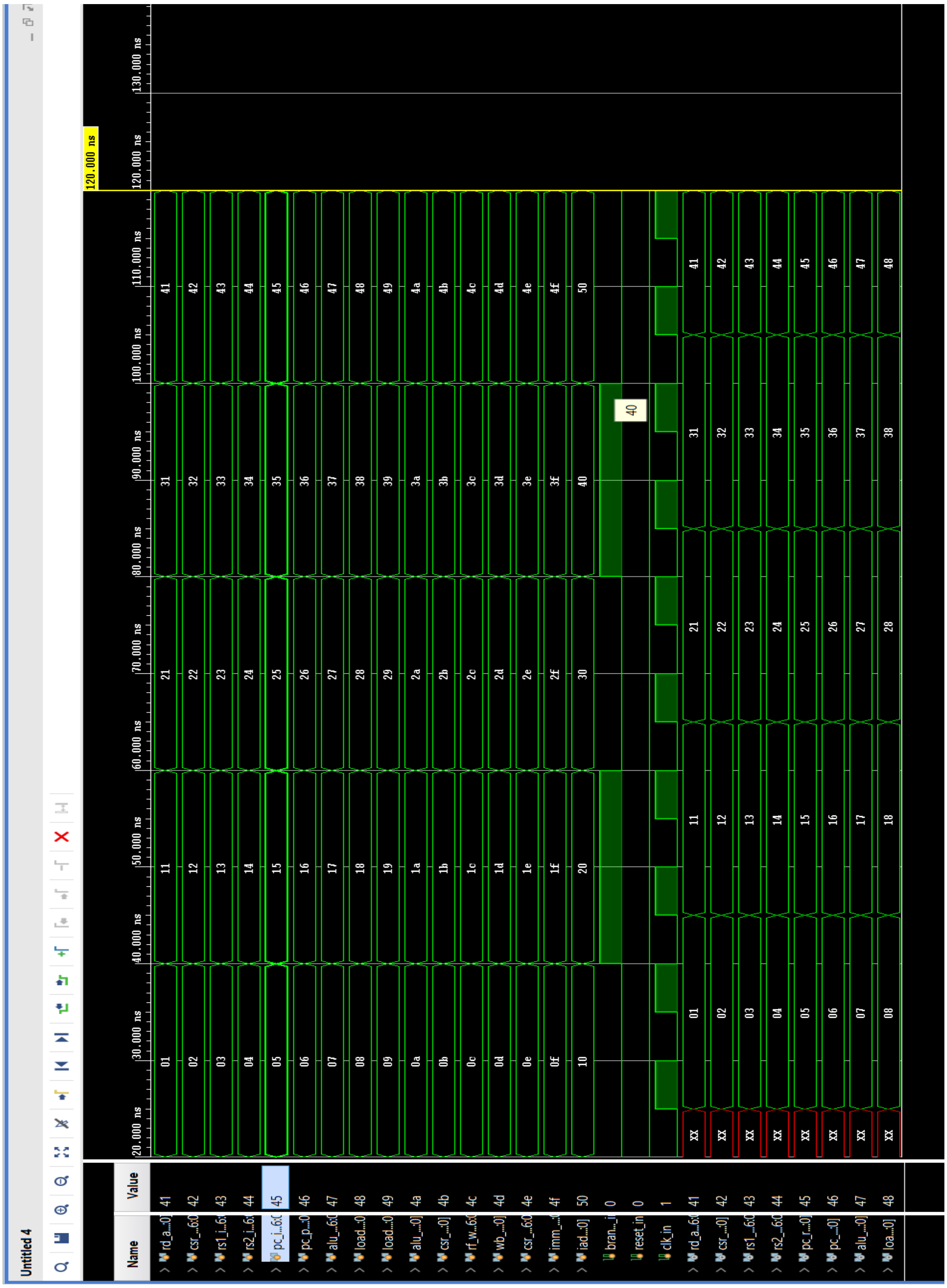
```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////
/////////////////////////////
// Company: maven silicon - VIT Vellore
// Engineer: VLSI Design
//
// Create Date: 12.07.2024 10:13:36
// Design Name: Regblock 2
// Module Name: msrv32_reg_block_2_tb
// Project Name: RISC V32I
// Target Devices:
// Tool Versions: Achyut's Vivado
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////
/////////////////////////////
```

```
module msrv32_reg_block_2_tb;
reg [6:0] rd_addr_in;
    reg [6:0] csr_addr_in;
    reg [6:0] rs1_in;
    reg [6:0] rs2_in;
    reg [6:0] pc_in;
    reg [6:0] pc_plus_4_in;
    reg [6:0] alu_opcode_in;
    reg [6:0] load_size_in;
    reg [6:0] load_unsigned_in;
    reg [6:0] alu_src_in;
    reg [6:0] csr_wr_en_in;
    reg [6:0] rf_wr_en_in;
    reg [6:0] wb_mux_sel_in;
    reg [6:0] csr_op_in;
    reg [6:0] imm_in;
    reg [6:0] iadder_out_in;
    reg branch_taken_in;
    reg reset_in;
    reg clk_in;
    wire [6:0] rd_addr_reg_out;
    wire [6:0] csr_addr_reg_out;
    wire [6:0] rs1_reg_out;
    wire [6:0] rs2_reg_out;
    wire [6:0] pc_reg_out;
    wire [6:0] pc_plus_reg_out;
    wire [6:0] alu_opcode_reg_out;
    wire [6:0] load_size_reg_out;
    wire [6:0] load_unsigned_reg_out;
    wire [6:0] alu_src_reg_out;
    wire [6:0] csr_wr_en_reg_out;
    wire [6:0] rf_wr_en_reg_out;
    wire [6:0] wb_mux_sel_reg_out;
    wire [6:0] csr_op_reg_out;
    wire [6:0] imm_reg_out;
    wire [6:0] iadder_out_reg_out;

    msrv32_reg_block_2 uut (
        .rd_addr_in(rd_addr_in),
        .csr_addr_in(csr_addr_in),
        .rs1_in(rs1_in),
        .rs2_in(rs2_in),
        .pc_in(pc_in),
        .pc_plus_4_in(pc_plus_4_in),
        .alu_opcode_in(alu_opcode_in),
        .load_size_in(load_size_in),
        .load_unsigned_in(load_unsigned_in),
        .alu_src_in(alu_src_in),
        .csr_wr_en_in(csr_wr_en_in),
        .rf_wr_en_in(rf_wr_en_in),
        .wb_mux_sel_in(wb_mux_sel_in),
        .csr_op_in(csr_op_in),
        .imm_in(imm_in),
        .iadder_out_in(iadder_out_in),
        .branch_taken_in(branch_taken_in),
        .reset_in(reset_in),
        .clk_in(clk_in),
        .rd_addr_reg_out(rd_addr_reg_out),
        .csr_addr_reg_out(csr_addr_reg_out),
        .rs1_reg_out(rs1_reg_out),
        .rs2_reg_out(rs2_reg_out),
        .pc_reg_out(pc_reg_out),
        .pc_plus_reg_out(pc_plus_reg_out),
        .alu_opcode_reg_out(alu_opcode_reg_out),
        .load_size_reg_out(load_size_reg_out),

.load_unsigned_reg_out(load_unsigned_reg_out),
        .alu_src_reg_out(alu_src_reg_out),
        .csr_wr_en_reg_out(csr_wr_en_reg_out),
        .rf_wr_en_reg_out(rf_wr_en_reg_out),
        .wb_mux_sel_reg_out(wb_mux_sel_reg_out),
        .csr_op_reg_out(csr_op_reg_out),
        .imm_reg_out(imm_reg_out),
        .iadder_out_reg_out(iadder_out_reg_out)
    );

    initial begin
        clk_in = 0;
        reset_in = 1;
        #10;
        reset_in = 0;
        #10;

        // Test case 1
        rd_addr_in = 7'b0000001;
        csr_addr_in = 7'b0000010;
        rs1_in = 7'b0000011;
        rs2_in = 7'b0000100;
        pc_in = 7'b0000101;
        pc_plus_4_in = 7'b0000110;
        alu_opcode_in = 7'b0000111;
        load_size_in = 7'b0001000;
        load_unsigned_in = 7'b0001001;
        alu_src_in = 7'b0001010;
        csr_wr_en_in = 7'b0001011;
        rf_wr_en_in = 7'b0001100;
        wb_mux_sel_in = 7'b0001101;
        csr_op_in = 7'b0001110;
        imm_in = 7'b0001111;
        iadder_out_in = 7'b0010000;
        branch_taken_in = 0;
        #20;

        // Test case 2
        rd_addr_in = 7'b0010001;
        csr_addr_in = 7'b0010010;
        rs1_in = 7'b0010011;
        rs2_in = 7'b0010100;
        pc_in = 7'b0010101;
        pc_plus_4_in = 7'b0010110;
        alu_opcode_in = 7'b0010111;
        load_size_in = 7'b0011000;
        load_unsigned_in = 7'b0011001;
        alu_src_in = 7'b0011010;
        csr_wr_en_in = 7'b0011011;
        rf_wr_en_in = 7'b0011100;
        wb_mux_sel_in = 7'b0011101;
        csr_op_in = 7'b0011110;
        imm_in = 7'b0011111;
        iadder_out_in = 7'b0100000;
        branch_taken_in = 1;
        #20;

        // Test case 3
        rd_addr_in = 7'b0100001;
        csr_addr_in = 7'b0100010;
        rs1_in = 7'b0100011;
        rs2_in = 7'b0100100;
        pc_in = 7'b0100101;
        pc_plus_4_in = 7'b0100110;
        alu_opcode_in = 7'b0100111;
        load_size_in = 7'b0101000;
        load_unsigned_in = 7'b0101001;
        alu_src_in = 7'b0101010;
        csr_wr_en_in = 7'b0101011;
        rf_wr_en_in = 7'b0101100;
        wb_mux_sel_in = 7'b0101101;
        csr_op_in = 7'b0101110;
        imm_in = 7'b0101111;
        iadder_out_in = 7'b0110000;
        branch_taken_in = 0;
        #20;

        // Test case 4
        rd_addr_in = 7'b0110001;
        csr_addr_in = 7'b0110010;
        rs1_in = 7'b0110011;
        rs2_in = 7'b0110100;
        pc_in = 7'b0110101;
        pc_plus_4_in = 7'b0110110;
        alu_opcode_in = 7'b0110111;
        load_size_in = 7'b0111000;
        load_unsigned_in = 7'b0111001;
        alu_src_in = 7'b0111010;
        csr_wr_en_in = 7'b0111011;
        rf_wr_en_in = 7'b0111100;
        wb_mux_sel_in = 7'b0111101;
        csr_op_in = 7'b0111110;
        imm_in = 7'b0111111;
        iadder_out_in = 7'b1000000;
        branch_taken_in = 1;
        #20;

        // Test case 5
        rd_addr_in = 7'b1000001;
        csr_addr_in = 7'b1000010;
        rs1_in = 7'b1000011;
        rs2_in = 7'b1000100;
        pc_in = 7'b1000101;
        pc_plus_4_in = 7'b1000110;
        alu_opcode_in = 7'b1000111;
        load_size_in = 7'b1001000;
        load_unsigned_in = 7'b1001001;
        alu_src_in = 7'b1001010;
        csr_wr_en_in = 7'b1001011;
        rf_wr_en_in = 7'b1001100;
        wb_mux_sel_in = 7'b1001101;
        csr_op_in = 7'b1001110;
        imm_in = 7'b1001111;
        iadder_out_in = 7'b1010000;
        branch_taken_in = 0;
        #20;

        $finish;
    end

    always #5 clk_in = ~clk_in;

endmodule
```

RTL Design – Reg block 2 - msrv32_reg_block_2

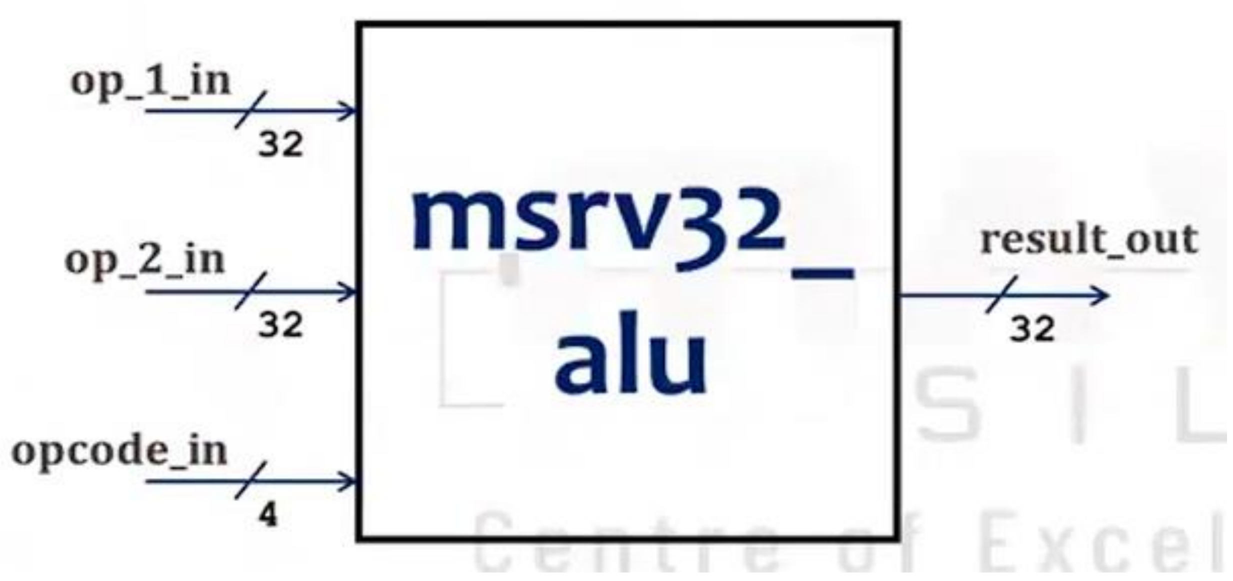


Waveform – Reg block 2 - msrv32_reg_block_2



ALU

Block Diagram



Signals -alu

1. op_1_in : operation first operand
2. op_2_in : operation second operand
3. opcode_in : operation code thes is signal is driven by function3 and function7 instruction fields
4. result_out : Result of the requested operation.

Explanation

In our RISC-V 32I project, the ALU (Arithmetic Logic Unit) serves as a crucial component for executing arithmetic and logical operations within the processor. It operates using four main signals: op_1_in and op_2_in, which represent the operands involved in the operation, opcode_in driven by function3 and function7 instruction fields, specifying the type of operation to be performed, and result_out, which outputs the computed result of the operation. These signals collectively manage the flow of data and control within the ALU, enabling it to perform tasks like addition, subtraction, bitwise operations, and comparisons as dictated by the processor's instructions.

Verilog code

Design code – ALU – msrv32_alu

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////
///////////////////////////////
// Company: maven silicon - VIT Vellore
// Engineer:
//
// Create Date: 08.07.2024 09:17:18
// Design Name: ALU
// Module Name: msrv32_alu
// Project Name: RSIC V 32I
// Target Devices:
// Tool Versions: Achyut's Vivado
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////
///////////////////////////////


module msrv32_alu(op_1_in, op_2_in, opcode_in,
result_out);
input [31:0] op_1_in;
input [31:0] op_2_in;
input [3:0] opcode_in;
output reg [31:0] result_out;
wire signed [31:0] op_1_in_signed;
wire signed [31:0] op_2_in_signed;
assign op_1_in_signed = op_1_in;
assign op_2_in_signed = op_2_in;

always @(*)
begin
case(opcode_in[3:0])
```

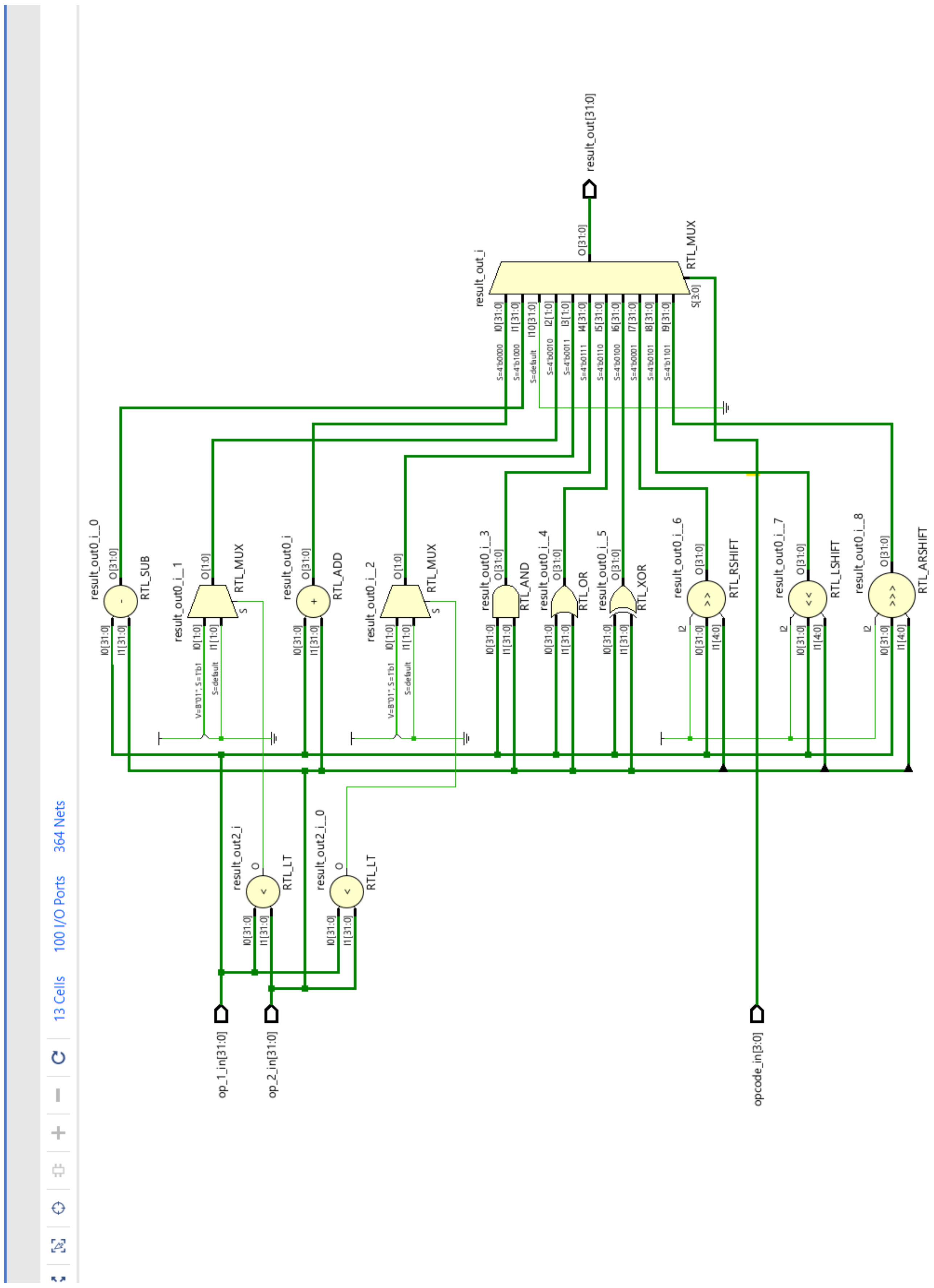
```
    4'b0000: result_out = op_1_in + op_2_in;
// Addition
        4'b1000: result_out = op_1_in - op_2_in;
// Subtraction
        4'b0010: result_out = (op_1_in < op_2_in) ?
32'b1 : 32'b0; // Unsigned less than
        4'b0011: result_out = (op_1_in_signed <
op_2_in_signed) ? 32'b1 : 32'b0; // Signed less than
        4'b0111: result_out = op_1_in & op_2_in;
// Bitwise AND
        4'b0110: result_out = op_1_in | op_2_in;
// Bitwise OR
        4'b0100: result_out = op_1_in ^ op_2_in;
// Bitwise XOR
        4'b0001: result_out = op_1_in >>
op_2_in[4:0];             // Logical right shift
        4'b0101: result_out = op_1_in <<
op_2_in[4:0];             // Logical left shift
        4'b1101: result_out = op_1_in_signed >>>
op_2_in[4:0];
     default  : result_out = 32'b0;

endcase

end

endmodule
```

Simulation code – ALU – msrv32_alu

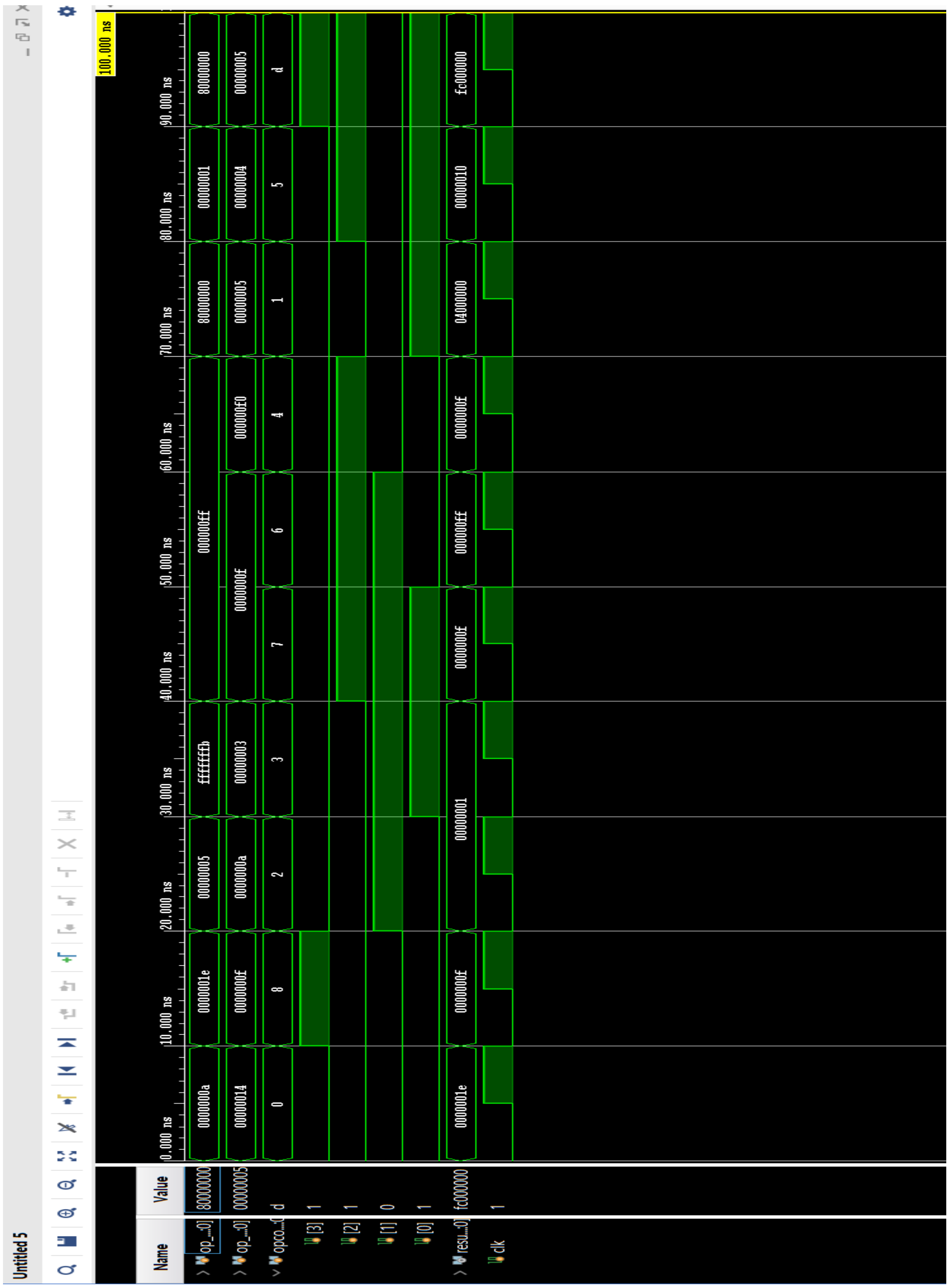
```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////
//////////////////////////////
// Company: maven silicon vit vellore
// Engineer: vlsi design
//
// Create Date: 12.07.2024 10:53:00
// Design Name: alu
// Module Name: msrv32_alu_tb
// Project Name: RISCV 32I
// Target Devices:
// Tool Versions: Achyut's vivado
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////
/////////////////////////////

module msrv32_alu_tb;
reg [31:0] op_1_in_tb;
  reg [31:0] op_2_in_tb;
  reg [3:0] opcode_in_tb;

  // Outputs
  wire [31:0] result_out_tb;

  // Instantiate the msrv32_alu module
  msrv32_alu dut (
    .op_1_in(op_1_in_tb),
    .op_2_in(op_2_in_tb),
    .opcode_in(opcode_in_tb),
    .result_out(result_out_tb)
  );

  // Clock generation
  reg clk = 0;
  always #5 clk = ~clk;

  // Test cases
  initial begin
    // Test case 1: Addition
    op_1_in_tb = 10;
    op_2_in_tb = 20;
    opcode_in_tb = 4'b0000;
    #10;

    // Test case 2: Subtraction
    op_1_in_tb = 30;
    op_2_in_tb = 15;
    opcode_in_tb = 4'b1000;
    #10;
    // Test case 3: Unsigned less than
    op_1_in_tb = 5;
    op_2_in_tb = 10;
    opcode_in_tb = 4'b0010;
    #10;

    // Test case 4: Signed less than
    op_1_in_tb = -5;
    op_2_in_tb = 3;
    opcode_in_tb = 4'b0011;
    #10;

    // Test case 5: Bitwise AND
    op_1_in_tb = 8'hFF;
    op_2_in_tb = 8'h0F;
    opcode_in_tb = 4'b0111;
    #10;

    // Test case 6: Bitwise OR
    op_1_in_tb = 8'hFF;
    op_2_in_tb = 8'h0F;
    opcode_in_tb = 4'b0110;
    #10;

    // Test case 7: Bitwise XOR
    op_1_in_tb = 8'hFF;
    op_2_in_tb = 8'hF0;
    opcode_in_tb = 4'b0100;
    #10;

    // Test case 8: Logical right shift
    op_1_in_tb = 32'h80000000;
    op_2_in_tb = 5;
    opcode_in_tb = 4'b0001;
    #10;

    // Test case 9: Logical left shift
    op_1_in_tb = 32'h00000001;
    op_2_in_tb = 4;
    opcode_in_tb = 4'b0101;
    #10;

    // Test case 10: Arithmetic right shift
    op_1_in_tb = -32'h80000000;
    op_2_in_tb = 5;
    opcode_in_tb = 4'b1101;
    #10;

    // End simulation
    $finish;
  end
endmodule
```

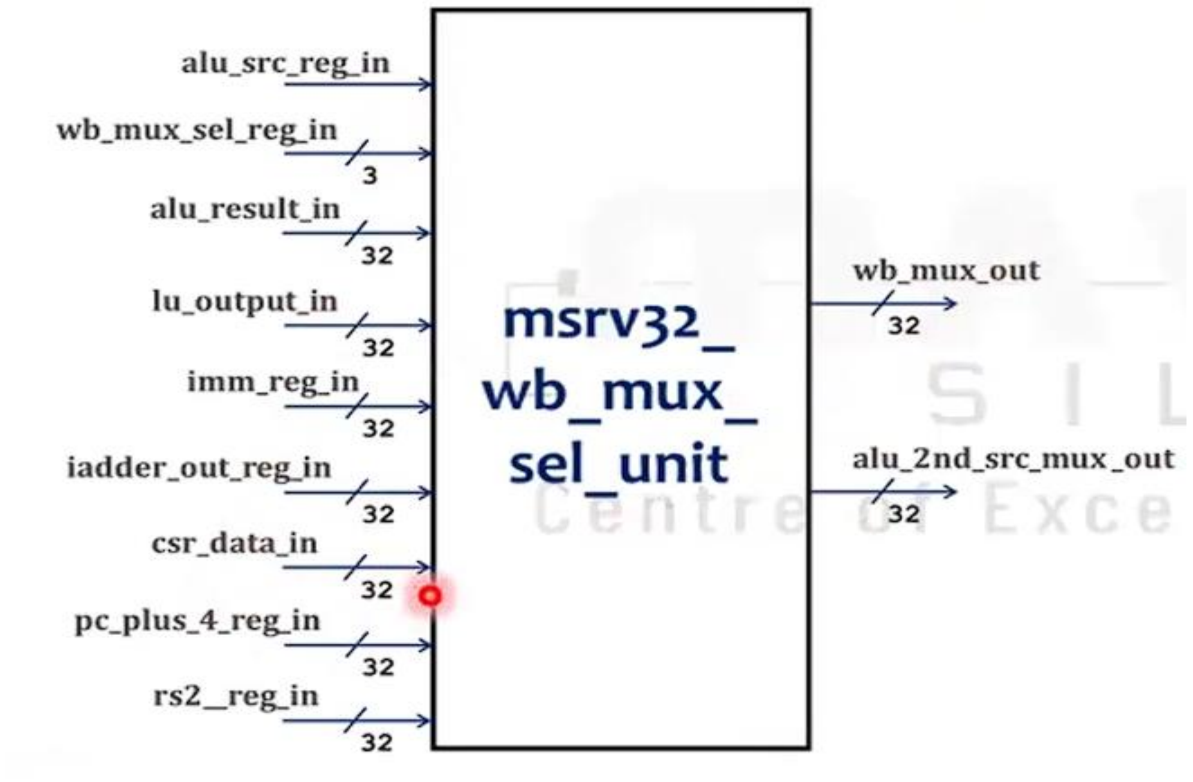
RTL VIEW– ALU – msrv32_alu



Waveform – ALU – msrv32_alu


Untitled 5
Name
Value
op_...:0] 80000000
op_...:0] 00000005
opco...:0] d
[3] 1
[2] 1
[1] 0
[0] 1
resu...:0] fc000000
clk 1
0.000 ns
10.000 ns
20.000 ns
30.000 ns
40.000 ns
50.000 ns
60.000 ns
70.000 ns
80.000 ns
90.000 ns
100.000 ns
0000000a 0000001e 00000005 fffffffb 000000ff 80000000 00000001 80000000
00000014 0000000f 0000000a 00000003 0000000f 000000f0 00000005 00000004 00000005
0 8 2 3 7 6 4 1 5 d
0000001e 0000000f 00000001 0000000f 000000ff 0000000f 04000000 00000010 fc000000

Wb Mux Selection out



Signals

1. wb_mux_sel_reg_in : Selects the data to be written in integer register file
2. alu_result_in :the result produced by alu.
3. Lu_output_in : output of load unit.
4. Imm_reg_in : Immediate data.
5. Iadder_out_reg_in : the sum of Immediate data and rs1.
6. Csr_data_in : data out port of CSR Module.
7. Pc_plus_4_reg_in: PC + 4
8. Rs2_reg_in : RS2 register output from Integer file.
9. Alu_src_reg_in : used to select rs2 register data immediate data.
10. Wb_mux_out : The output port of wb_mux.
11. Alu_2nd_src_mux_out: RS2 register output

Verilog code

Design code - Wb Mux Selection out

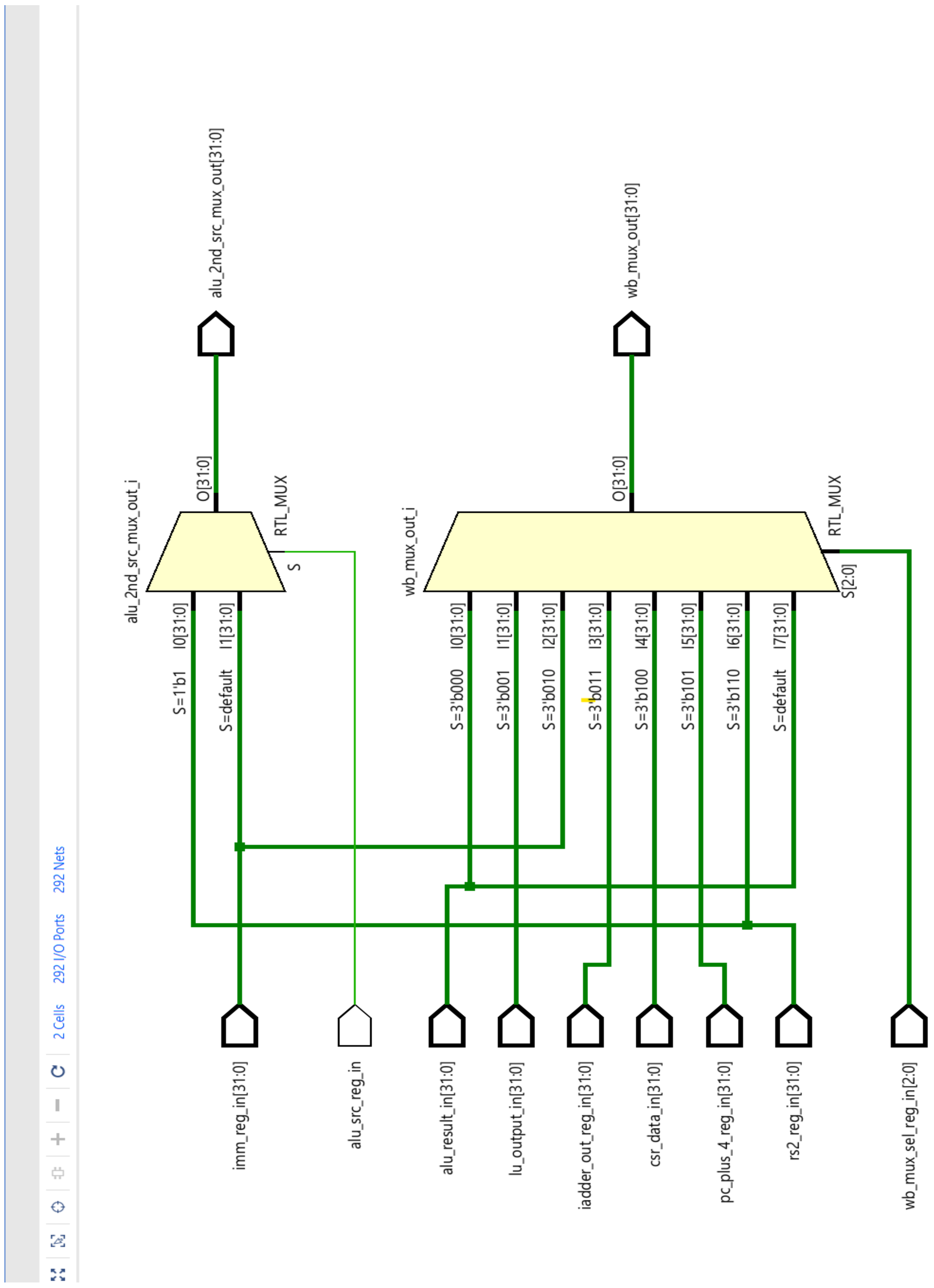
```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////
//////////////////////////////
// Company:
// Engineer:
//
// Create Date: 08.07.2024 10:26:08
// Design Name:
// Module Name: msrv32_wb_mux_sel_unit
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////
//////////////////////////////


module msrv32_wb_mux_sel_unit(alu_src_reg_in
,wb_mux_sel_reg_in , alu_result_in, lu_output_in ,
imm_reg_in, iadder_out_reg_in, csr_data_in,
pc_plus_4_reg_in, rs2_reg_in, wb_mux_out,
alu_2nd_src_mux_out);
input alu_src_reg_in;
input [2:0]wb_mux_sel_reg_in;
input [31:0] alu_result_in;
input [31:0] lu_output_in;
input [31:0] imm_reg_in;
input [31:0] iadder_out_reg_in;
input [31:0] csr_data_in;
input [31:0] pc_plus_4_reg_in;
input [31:0] rs2_reg_in;
output reg [31:0] wb_mux_out;
output [31:0] alu_2nd_src_mux_out;
assign alu_2nd_src_mux_out = (alu_src_reg_in) ?
rs2_reg_in : imm_reg_in;
always  @(*)
begin
    case(wb_mux_sel_reg_in)
        3'b000: wb_mux_out = alu_result_in;
        3'b001: wb_mux_out = lu_output_in;
        3'b010: wb_mux_out = imm_reg_in;
        3'b011: wb_mux_out = iadder_out_reg_in;
        3'b100: wb_mux_out = csr_data_in;
        3'b101: wb_mux_out = pc_plus_4_reg_in;
        3'b110: wb_mux_out = rs2_reg_in;
        default: wb_mux_out = alu_result_in;
    endcase
end



endmodule
```

Simulation code - Wb Mux Selection out

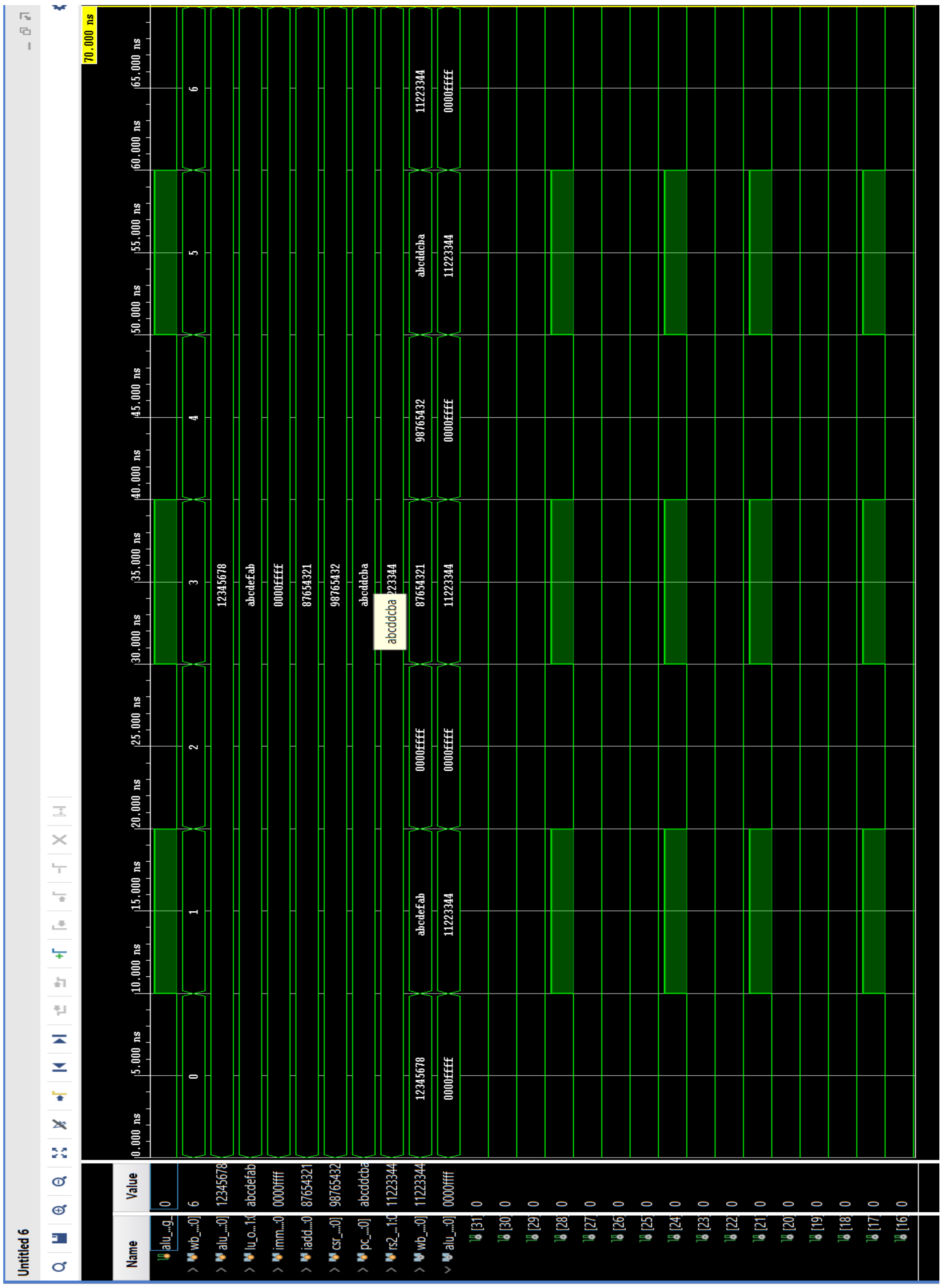
```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////
//////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12.07.2024 11:22:00
// Design Name:
// Module Name: msrv32_wb_mux_sel_unit_tb
// Project Name:
// Target Devices:
// Tool Versions: Achyut's vivado
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
```

```
///////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////

module msrv32_wb_mux_sel_unit_tb;
   reg alu_src_reg_in;
    reg [2:0] wb_mux_sel_reg_in;
    reg [31:0] alu_result_in;
    reg [31:0] lu_output_in;
    reg [31:0] imm_reg_in;
    reg [31:0] iadder_out_reg_in;
    reg [31:0] csr_data_in;
    reg [31:0] pc_plus_4_reg_in;
    reg [31:0] rs2_reg_in;

    // Outputs
    wire [31:0] wb_mux_out;
    wire [31:0] alu_2nd_src_mux_out;

    // Instantiate the unit under test
    msrv32_wb_mux_sel_unit dut (
        .alu_src_reg_in(alu_src_reg_in),
        .wb_mux_sel_reg_in(wb_mux_sel_reg_in),
        .alu_result_in(alu_result_in),
        .lu_output_in(lu_output_in),
        .imm_reg_in(imm_reg_in),
        .iadder_out_reg_in(iadder_out_reg_in),
        .csr_data_in(csr_data_in),
        .pc_plus_4_reg_in(pc_plus_4_reg_in),
        .rs2_reg_in(rs2_reg_in),
        .wb_mux_out(wb_mux_out),
        .alu_2nd_src_mux_out(alu_2nd_src_mux_out)
    );

    // Initial stimulus
    initial begin
        // Test Case 1
        alu_src_reg_in = 1'b0;
        wb_mux_sel_reg_in = 3'b000;
        alu_result_in = 32'h12345678;
        lu_output_in = 32'hABCDEFAB;
        imm_reg_in = 32'h0000FFFF;
        iadder_out_reg_in = 32'h87654321;
        csr_data_in = 32'h98765432;
        pc_plus_4_reg_in = 32'hABCDDCBA;
        rs2_reg_in = 32'h11223344;
        #10;

        // Test Case 2
        alu_src_reg_in = 1'b1;
        wb_mux_sel_reg_in = 3'b001;
        #10;

        // Test Case 3
        alu_src_reg_in = 1'b0;
        wb_mux_sel_reg_in = 3'b010;
        #10;

        // Test Case 4
        alu_src_reg_in = 1'b1;
        wb_mux_sel_reg_in = 3'b011;
        #10;

        // Test Case 5
        alu_src_reg_in = 1'b0;
        wb_mux_sel_reg_in = 3'b100;
        #10;

        // Test Case 6
        alu_src_reg_in = 1'b1;
        wb_mux_sel_reg_in = 3'b101;
        #10;

        // Test Case 7
        alu_src_reg_in = 1'b0;
        wb_mux_sel_reg_in = 3'b110;
        #10;

        $finish;
    end
endmodule
```

RTL Design - Wb Mux Selection out

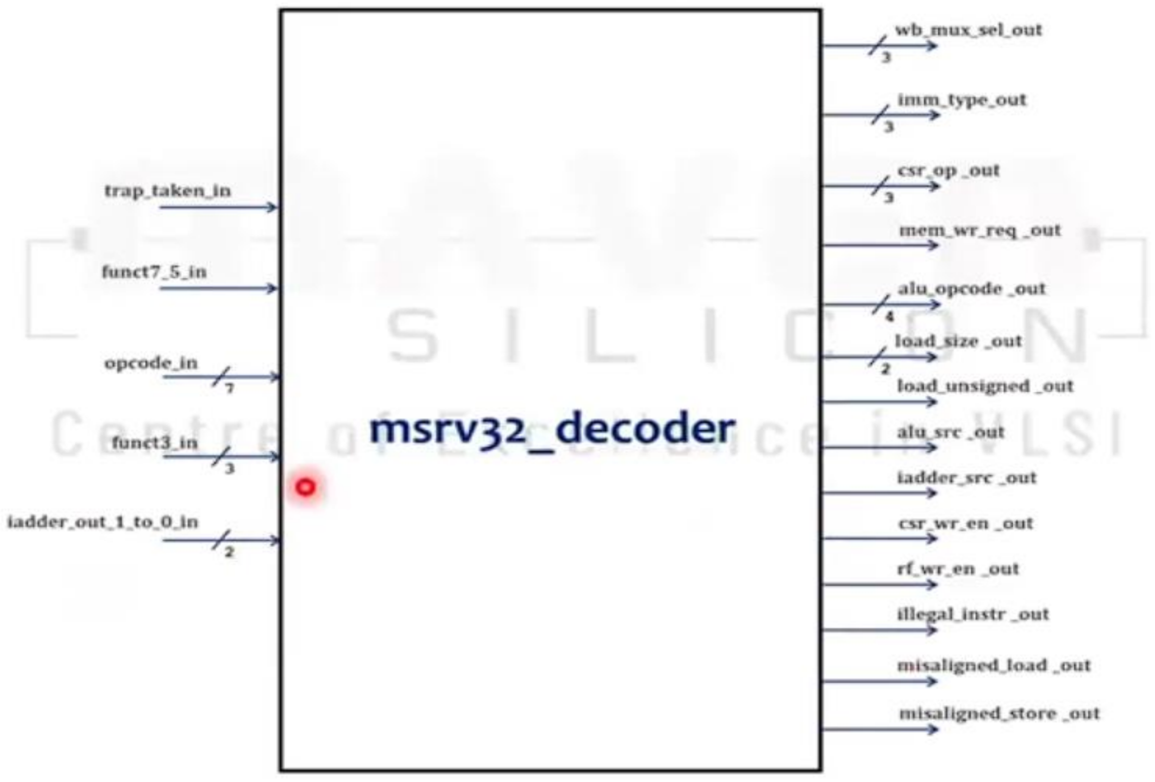


Waveform- Wb Mux Selection out



DECODER

Block Diagram



Signals – Decoder

1. Op_code_6_to_2_in : connected to the instruction opcode field.
2. Funct7_5_in : connected to instruction funct7 field.
3. Funct3_in : connected to function 3 field.
4. Iadder_out_1_0_in : used to verify the alignment of loads and stores.
5. Trap_taken_in : when set is high it indicates that trap will be taken next clock cycle. Connected to machine control module.
6. Alu_opcode_out : selects the operation to performed by alu.
7. Mem_wr_req_out : when set high indicates a request to write memory.
8. Load_size_out : Indicates the word size of load instruction.
9. Load_unsigned_out :Indicates the type of load instruction.
10. Alu_src-out : selects alu second operand.
11. Iadder_src_out : selects immediate adder 2nd operand.
12. Csr_wr_en_out : controls the wr_en input of CSR register file.

13. Rf_wr_en_out: controls the input wr_en of csr file.
14. Wb_mux_sel_out : Selects the data to be written in the integer register file.
15. Imm_type_out : selects the data to be written in the integer file.
16. Csr_op_out : selects the operation to be performed by CSR Register file.
17. Illeagle_instr_out: When set is high that indicates an invalid or not implemented instruction was fetched from memory.
18. Misaligned_load_out : When set is high that an attempt to read data_in disagreement with the memory.
19. Misaligned_store_out : when set is high indicates an attempt to write data in memory in disagreement with memory alignment.

Verilog code

Design code – Decoder

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////
///////////////////////////////
// Company: maven silicon - VIT Vellore
// Engineer: Maven Silicon
//
// Create Date: 08.07.2024 19:38:34
// Design Name: Decoder
// Module Name: msrv32_decoder
// Project Name: RISC V 32I
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////
///////////////////////////////


module msrv32_decoder(trap_taken_in, funct7_5_in,
opcode_in, funct3_in, iadder_out_1_to_0_in,
wb_mux_sel_out, imm_type_out, csr_op_out,
mem_wr_req_out, alu_opcode_out, load_size_out,
load_unsigned_out, alu_src_out,
iadder_src_out,csr_wr_en_out, rf_wr_en_out,
illegal_instr_out, misaligned_load_out,
misaligned_store_out);
input trap_taken_in;
input funct7_5_in;
input [6:0] opcode_in;
input [2:0] funct3_in;
input [1:0] iadder_out_1_to_0_in;

output [2:0] wb_mux_sel_out;
output [2:0] imm_type_out;
output [2:0] csr_op_out;
output mem_wr_req_out;
output [3:0] alu_opcode_out;
output [1:0] load_size_out;
output load_unsigned_out;
output alu_src_out;
output iadder_src_out;
output csr_wr_en_out;
output rf_wr_en_out;
output illegal_instr_out;
output misaligned_load_out;
output misaligned_store_out;
reg is_branch,is_jal, is_jalr , is_auipc, is_lui,
is_op, is_op_imm,is_load,is_store,is_system,
is_misc_mem;
reg is_addi,is_slti, is_sltiu, is_andi , is_ori,
is_xori;
wire alu_opcode_out_3_wire;
wire csr_or_wire;
wire is_csr;
wire is_implimented_instr;
wire mal_word;
wire mal_half;
wire misaligned;
always @(*)
begin
    case(opcode_in[6:0])
        5'b01100 :{is_op, is_op_imm, is_load,
is_store,  is_branch, is_jal, is_jalr, is_lui,
is_auipc, is_misc_mem, is_system} =11'b10000000000;
```

```
        5'b00100 :{is_op, is_op_imm, is_load, 
is_store,  is_branch, is_jal, is_jalr, is_lui, 
is_auipc, is_misc_mem, is_system} =11'b01000000000;
        5'b00000 :{is_op, is_op_imm, is_load, 
is_store,  is_branch, is_jal, is_jalr, is_lui, 
is_auipc, is_misc_mem, is_system} =11'b00100000000;
        5'b01000 :{is_op, is_op_imm, is_load, 
is_store,  is_branch, is_jal, is_jalr, is_lui, 
is_auipc, is_misc_mem, is_system} =11'b00010000000;
        5'b11000 :{is_op, is_op_imm, is_load, 
is_store,  is_branch, is_jal, is_jalr, is_lui, 
is_auipc, is_misc_mem, is_system} =11'b00001000000;
        5'b11011 :{is_op, is_op_imm, is_load, 
is_store,  is_branch, is_jal, is_jalr, is_lui, 
is_auipc, is_misc_mem, is_system} =11'b00000100000;
        5'b11001 :{is_op, is_op_imm, is_load, 
is_store,  is_branch, is_jal, is_jalr, is_lui, 
is_auipc, is_misc_mem, is_system} =11'b00000010000;
        5'b01101 :{is_op, is_op_imm, is_load, 
is_store,  is_branch, is_jal, is_jalr, is_lui, 
is_auipc, is_misc_mem, is_system} =11'b00000001000;
        5'b00101 :{is_op, is_op_imm, is_load, 
is_store,  is_branch, is_jal, is_jalr, is_lui, 
is_auipc, is_misc_mem, is_system} =11'b00000000100;
        5'b00011 :{is_op, is_op_imm, is_load, 
is_store,  is_branch, is_jal, is_jalr, is_lui, 
is_auipc, is_misc_mem, is_system} =11'b00000000010;
        5'b11100 :{is_op, is_op_imm, is_load, 
is_store,  is_branch, is_jal, is_jalr, is_lui, 
is_auipc, is_misc_mem, is_system} =11'b00000000001;
        default    :{is_op, is_op_imm, is_load, 
is_store,  is_branch, is_jal, is_jalr, is_lui, 
is_auipc, is_misc_mem, is_system} =11'b00000000000;
    endcase
end
always @*
begin
case(funct3_in)
    3'b000: {is_addi, is_slti, is_sltiu, is_andi, 
is_ori, is_xori} = {is_op_imm, 1'b0, 1'b0, 1'b0, 
1'b0, 1'b0, 1'b0};
    3'b010: {is_addi, is_slti, is_sltiu, is_andi, 
is_ori, is_xori} = {1'b0, is_op_imm, 1'b0, 1'b0, 
1'b0, 1'b0, 1'b0};
```

```
    3'b011: {is_addi, is_slti, is_sltiu, is_andi,
is_ori, is_xori} = {1'b0, 1'b0, is_op_imm, 1'b0,
1'b0, 1'b0, 1'b0};
    3'b111: {is_addi, is_slti, is_sltiu, is_andi,
is_ori, is_xori} = {1'b0, 1'b0, 1'b0, is_op_imm,
1'b0, 1'b0, 1'b0};
    3'b110: {is_addi, is_slti, is_sltiu, is_andi,
is_ori, is_xori} = {1'b0, 1'b0, 1'b0, 1'b0,
is_op_imm, 1'b0, 1'b0};
    3'b100: {is_addi, is_slti, is_sltiu, is_andi,
is_ori, is_xori} = {1'b0, 1'b0, 1'b0, 1'b0, 1'b0,
is_op_imm, 1'b0};
    default:{is_addi, is_slti, is_sltiu, is_andi,
is_ori, is_xori} =6'b000000;
endcase
end

assign load_size_out = funct3_in[1:0];
assign load_unsigned_out =funct3_in[2];
assign alu_src_out = opcode_in[5];

assign is_csr = is_system & (funct3_in[2]
|funct3_in[1]|funct3_in[0]);
assign csr_wr_en_out = is_csr;
assign csr_op_out =funct3_in;

assign iadder_src_out = is_load |is_store|is_jalr;
assign rf_wr_en_out = is_lui
|is_auipc|is_jalr|is_jal|is_op|is_load|is_csr|is_op_i
mm;
assign alu_opcode_out[2:0] = funct3_in;
assign alu_opcode_out[3] = funct7_5_in &~(is_addi
|is_slti|is_sltiu|is_andi|is_ori|is_xori);

assign wb_mux_sel_out[0] =
is_load|is_auipc|is_jal|is_jalr;
assign wb_mux_sel_out[1] =is_lui|is_auipc;
assign wb_mux_sel_out[2] =is_csr|is_jal|is_jalr;

assign imm_type_out[0] =
is_op_imm|is_load|is_jalr|is_branch|is_jal;
assign imm_type_out[1] = is_store|is_branch|is_csr;
assign imm_type_out[2] =
is_lui|is_auipc|is_jal|is_csr;
```

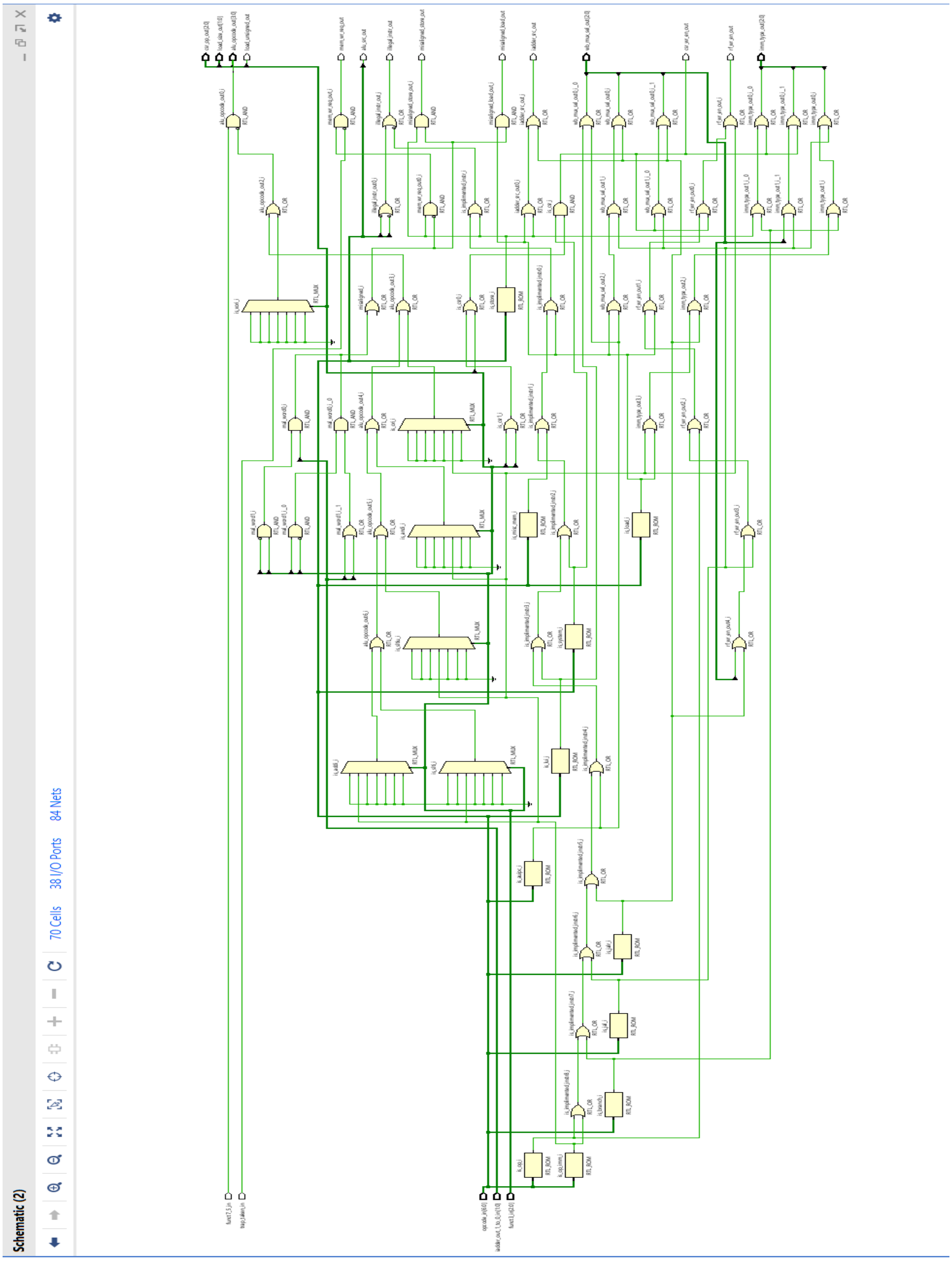
```
assign is_implimented_instr =
is_op|is_op_imm|is_branch|is_jal|is_jalr|is_auipc|is_
lui|is_system|is_misc_mem|is_load|is_store;
assign illegal_instr_out
=~opcode_in[1]|~opcode_in[0]|~is_implimented_instr;

assign mal_word =
funct3_in[1]&~funct3_in[0]&(iadder_out_1_to_0_in[1]|i
adder_out_1_to_0_in[0]);
assign mal_word =
~funct3_in[1]&funct3_in[0]&iadder_out_1_to_0_in[0];
assign misaligned = mal_word|mal_half;
assign misaligned_store_out = is_store&misaligned;
assign misaligned_load_out = is_load&misaligned;

assign mem_wr_req_out =
is_store&~misaligned&~trap_taken_in;

endmodule
```

Simulation code – Decoder

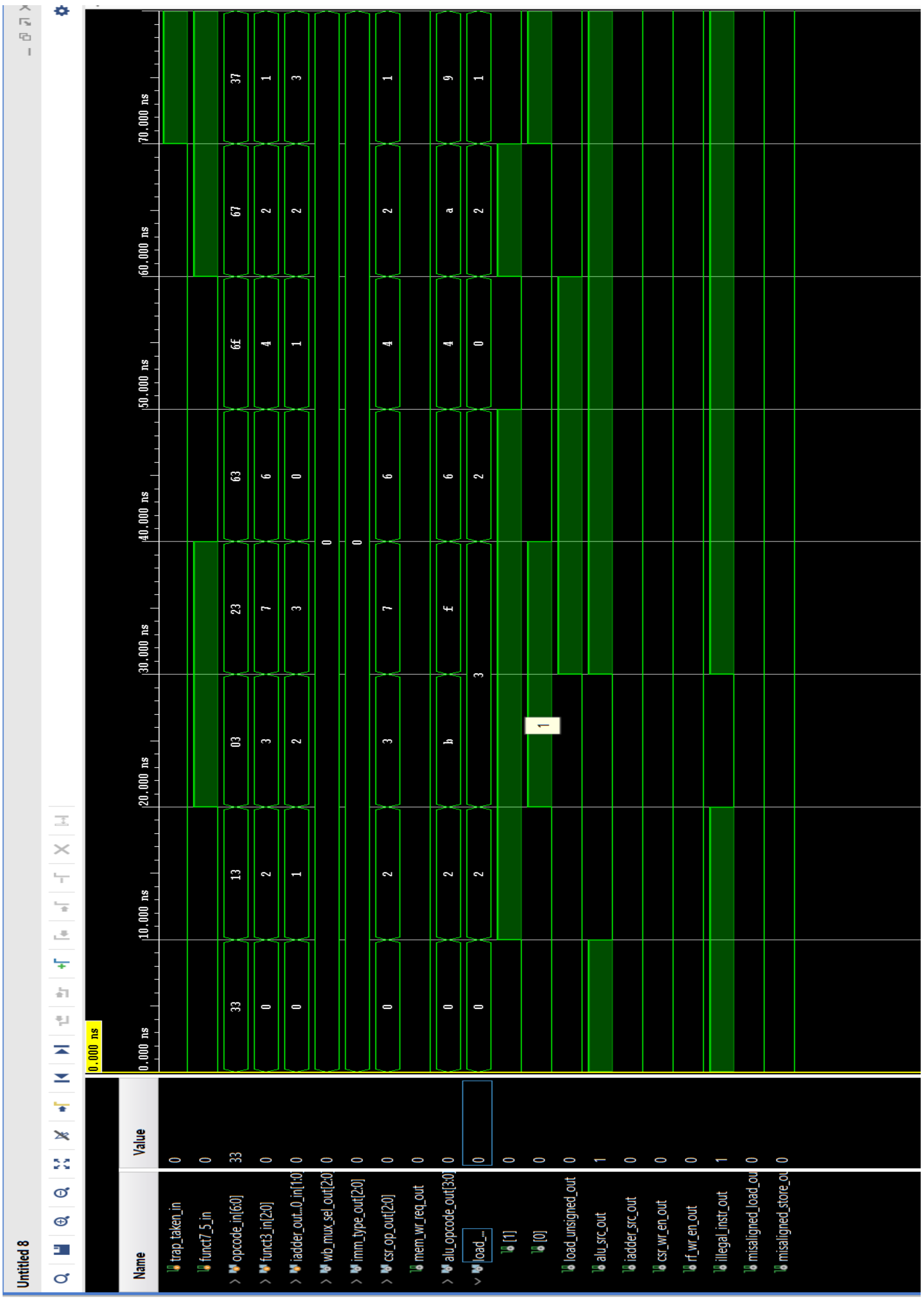
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////
//////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12.07.2024 12:36:49
// Design Name:
// Module Name: msrv32_decoder_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
```

```
//////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////

module msrv32_decoder_tb;
reg trap_taken_in;
    reg funct7_5_in;
    reg [6:0] opcode_in;
    reg [2:0] funct3_in;
    reg [1:0] iadder_out_1_to_0_in;

    wire [2:0] wb_mux_sel_out;
    wire [2:0] imm_type_out;
    wire [2:0] csr_op_out;
    wire mem_wr_req_out;
    wire [3:0] alu_opcode_out;
    wire [1:0] load_size_out;
    wire load_unsigned_out;
    wire alu_src_out;
    wire iadder_src_out;
    wire csr_wr_en_out;
    wire rf_wr_en_out;
    wire illegal_instr_out;
    wire misaligned_load_out;
    wire misaligned_store_out;

    msrv32_decoder uut (
        .trap_taken_in(trap_taken_in),
        .funct7_5_in(funct7_5_in),
        .opcode_in(opcode_in),
        .funct3_in(funct3_in),
        .iadder_out_1_to_0_in(iadder_out_1_to_0_in),
        .wb_mux_sel_out(wb_mux_sel_out),
        .imm_type_out(imm_type_out),
        .csr_op_out(csr_op_out),
        .mem_wr_req_out(mem_wr_req_out),
        .alu_opcode_out(alu_opcode_out),
        .load_size_out(load_size_out),
        .load_unsigned_out(load_unsigned_out),
        .alu_src_out(alu_src_out),
        .iadder_src_out(iadder_src_out),
        .csr_wr_en_out(csr_wr_en_out),
        .rf_wr_en_out(rf_wr_en_out),
        .illegal_instr_out(illegal_instr_out),
        .misaligned_load_out(misaligned_load_out),
        .misaligned_store_out(misaligned_store_out)
    );

    initial begin
        // Test case 1
        trap_taken_in = 0;
        funct7_5_in = 0;
        opcode_in = 7'b0110011;  // is_op
        funct3_in = 3'b000;
        iadder_out_1_to_0_in = 2'b00;
        #10;

        // Test case 2
        trap_taken_in = 0;
        funct7_5_in = 0;
        opcode_in = 7'b0010011;  // is_op_imm
        funct3_in = 3'b010;
        iadder_out_1_to_0_in = 2'b01;
        #10;

        // Test case 3
        trap_taken_in = 0;
        funct7_5_in = 1;
        opcode_in = 7'b0000011;  // is_load
        funct3_in = 3'b011;
        iadder_out_1_to_0_in = 2'b10;
        #10;

        // Test case 4
        trap_taken_in = 0;
        funct7_5_in = 1;
        opcode_in = 7'b0100011;  // is_store
        funct3_in = 3'b111;
        iadder_out_1_to_0_in = 2'b11;
        #10;

        // Test case 5
        trap_taken_in = 0;
        funct7_5_in = 0;
        opcode_in = 7'b1100011;  // is_branch
        funct3_in = 3'b110;
        iadder_out_1_to_0_in = 2'b00;
        #10;
        // Test case 6
        trap_taken_in = 0;
        funct7_5_in = 0;
        opcode_in = 7'b1101111;  // is_jal
        funct3_in = 3'b100;
        iadder_out_1_to_0_in = 2'b01;
        #10;

        // Test case 7
        trap_taken_in = 0;
        funct7_5_in = 1;
        opcode_in = 7'b1100111;  // is_jalr
        funct3_in = 3'b010;
        iadder_out_1_to_0_in = 2'b10;
        #10;

        // Test case 8
        trap_taken_in = 1;
        funct7_5_in = 1;
        opcode_in = 7'b0110111;  // is_lui
        funct3_in = 3'b001;
        iadder_out_1_to_0_in = 2'b11;
        #10;

        $finish;
    end
endmodule
```

RTL Design – Decoder

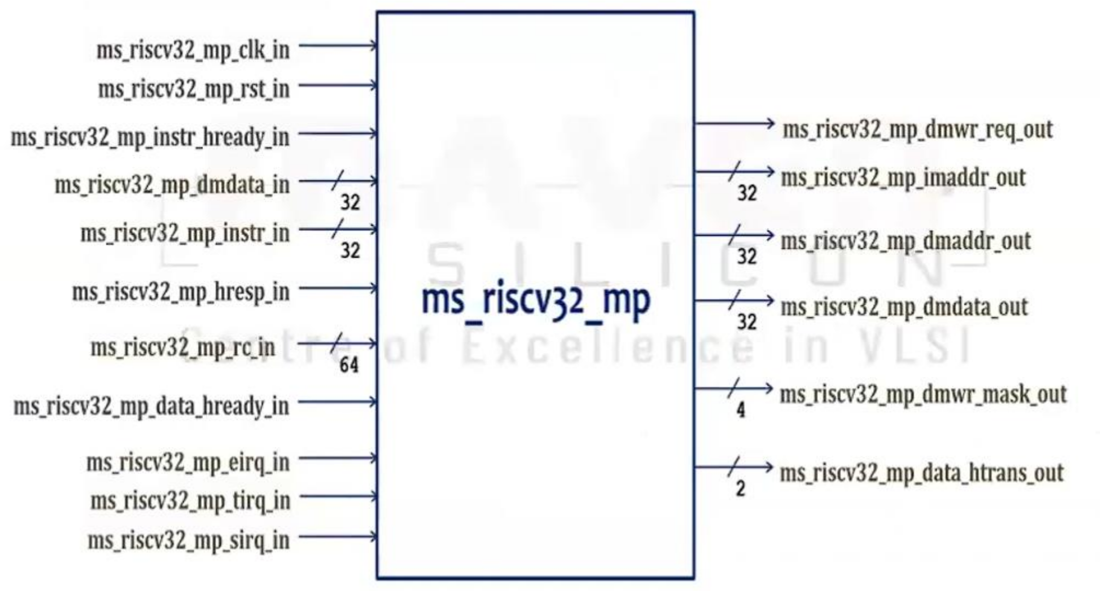


Waveform - Decoder



RISC V 32I - TOP MODULE

Block Diagram



Verilog code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////
////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12.07.2024 14:34:50
// Design Name:
// Module Name: msrv32_top
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
```

```
/////////////////////////////////////////////////////
/////////////////////////////


module msrv32_top(
// Connections with Instruction Memory
input ms_riscv32_mp_clk_in,
input ms_riscv32_mp_rst_in,
input ms_riscv32_mp_rc_in,
//input [31:0] ms_riscv32_mp_imaddr_in,
output [31:0] ms_riscv32_mp_imaddr_out,
input [31:0] ms_riscv32_mp_instr_in,
input ms_riscv32_mp_instr_hready_in,
output [31:0] ms_riscv32_mp_dmaddr_out,
output [31:0] ms_riscv32_mp_dmdata_out,
output ms_riscv32_mp_dmwr_req_out,
output [3:0 ] ms_riscv32_mp_dmwr_mask_out,
input [31:0] ms_riscv32_mp_data_in,
input ms_riscv32_mp_data_hready_in,
input ms_riscv32_mp_hresp_in,
output [1:0] ms_riscv32_mp_data_htrans_out,

input ms_riscv32_mp_eirq_in,
input ms_riscv32_mp_tirq_in,
input ms_riscv32_mp_sirq_in
);
// writeback selection
parameter WB_ALU = 3'b000;
parameter WB_LU = 3'b001;
parameter WB_IMM = 3'b010;
parameter WB_IADDER_OUT = 3'b011;
parameter WB_CSR = 3'b100;
parameter WB_PC_PLUS = 3'b101;

wire [31:0] iaddr;
wire [31:0] pc;
wire [31:0] pc_plus_4;
wire misaligned_instr;
wire [31:0] pc_mux;
wire [31:0] rs2;
wire mem_wr_req;
wire flush;
wire [6:0] opcode;
wire [2:0] funct3;
wire [6:0] funct7;
wire [4:0] rs1_addr;
wire [4:0] rs2_addr;
wire [4:0] rd_addr;
wire [11:0] csr_addr;
wire [31:7] instr_31_to_7;
wire [31:0] rs1;
wire [31:0] imm;
wire iadder_src;
wire wr_en_csr_file;
wire wr_en_integer_file;
wire [11:0] csr_addr_reg;
wire [2:0] csr_op_reg;
wire [31:0] imm_reg;
wire [31:0] rs1_reg;
wire [31:0] pc_reg2;
wire i_or_e;
wire set_cause;
wire [3:0] cause;
wire set_epc;
wire instret_inc;
wire mie_clear;
wire mie_set;
wire misaligned_exception;
wire mie;
wire meie_out;
wire mtie_out;
wire msie_out;
wire meip_out;
wire mtip_out;
wire msip_out;
wire rf_wr_en_reg;
wire csr_wr_en_reg;
wire csr_wr_en_reg_file;
wire integer_wr_en_reg_file;
wire [4:0] rd_addr_reg;
wire [2:0] wb_mux_sel;
wire [2:0] wb_mux_sel_reg;
wire [31:0] lu_output;
wire [31:0] alu_result;
wire [31:0] csr_data;
wire [31:0] pc_plus_4_reg;
wire [31:0] iadder_out_reg;
wire [31:0] rs2_reg;
wire alu_src_reg;
wire [31:0] wb_mux_out;
wire [31:0] alu_2nd_src_mux;
wire illegal_instr;
wire branch_taken;
wire [31:0] next_pc;
reg [31:0] pc_reg;
wire misaligned_load;
wire misaligned_store;
wire [3:0] cause_in;
wire [1:0] pc_src;
wire trap_taken;
wire [1:0] load_size_reg;
wire [3:0] alu_opcode_reg;
wire load_unsigned_reg;
wire [31:0] iadder_out;
wire [31:0] epc;
wire [31:0] trap_address;
wire [3:0] alu_opcode;
wire [3:0] mem_wr_mask;
wire [1:0] load_size;
wire load_unsigned;
wire alu_src;
wire csr_wr_en;
wire rf_wr_en;
wire [2:0] imm_type;
wire [2:0] csr_op;
wire [31:0] su_data_out;
wire [31:0] su_d_addr;
wire [3:0] su_wr_mask;
wire su_wr_req;
msrv32_pc PC(
    .rst_in(ms_riscv32_mp_rst_in),
    .pc_src_in(pc_src),
    .epc_in(epc),
    .trap_address_in(trap_address),
    .branch_taken_in(branch_taken),
    .iaddr_in(iaddr [31:1]),
    .ahb_ready_in(ms_riscv32_mp_instr_hready_in),
    .pc_in(pc),
    .iaddr_out(ms_riscv32_mp_imaddr_out),
    .pc_plus_4_out(pc_plus_4),
    .misaligned_instr_logic_out(misaligned_instr),
    .pc_mux_out(pc_mux)
```

```
);
msrv32_reg_block_1 REG1 (
     .pc_mux_in(pc_mux),
     .ms_riscv32_mp_clk_in(ms_riscv32_mp_clk_in),
     .ms_riscv32_mp_rst_in(ms_riscv32_mp_rst_in),
     .pc_out(pc)
);

msrv32_instruction_mux IM (
     .flush_in(flush),
     .ms_riscv32_mp_instr_in(ms_riscv32_mp_instr_in),
     .opcode_out(opcode),
     .funct3_out(funct3),
     .funct7_out(funct7),
     .rs1addr_out(rs1_addr),
     .rs2addr_out(rs2_addr),
     .rdaddr_out(rd_addr),
     .csr_addr_out(csr_addr),
     .instr_out(instr_31_to_7)

);

msrv32_store_unit su (
     .funct3_in(funct3[1:0]),
     .iadder_in(iaddr),
     .rs2_in(rs2),
     .mem_wr_req_in(mem_wr_req),
     .ahb_ready_in(ms_riscv32_mp_data_hready_in),

.ms_riscv32_mp_dmdata_out(ms_riscv32_mp_dmdata_out),

.ms_riscv32_mp_dmaddr_out(ms_riscv32_mp_dmaddr_out),

.ms_riscv32_mp_dmwr_mask_out(ms_riscv32_mp_dmwr_mask_
out),

.ms_riscv32_mp_dmwr_req_out(ms_riscv32_mp_dmwr_req_ou
t),
     .ahb_htrans_out(ms_riscv32_mp_data_htrans_out)
);

msrv32_decoder DEC (
     .trap_taken_in(trap_taken),
     .funct7_5_in(funct7[5]),
     .opcode_in(opcode),
     .funct3_in(funct3),
     .iadder_out_1_to_0_in(iaddr[1:0]),
     .wb_mux_sel_out(wb_mux_sel),
     .imm_type_out(imm_type),
     .csr_op_out(csr_op),
     .mem_wr_req_out(mem_wr_req),

     .alu_opcode_out(alu_opcode),

     .load_size_out(load_size),
     .load_unsigned_out(load_unsigned),
     .alu_src_out(alu_src),
     .iadder_src_out(iadder_src),
     .csr_wr_en_out(csr_wr_en),
     .rf_wr_en_out(rf_wr_en),
     .illegal_instr_out(illegal_instr),
     .misaligned_load_out(misaligned_load),
     .misaligned_store_out(misaligned_store)
);

msrv32_imm_generator IMG (
     .instr_in(instr_31_to_7),
     .imm_type_in(imm_type),
     .imm_out (imm)
);

msrv32_immediate_adder imm_adder(
     .pc_in(pc),
     .rs_1_in(rs1),
     .iadder_src_in(iadder_src),
     .imm_in(imm),
     .iadder_out(iaddr)
);

msrv32_branch_unit BU (
.rs1_in(rs1),
.rs2_in(rs2),
.opecode_in(opcode[6:0]),
.funct3_in(funct3),
.branch_taken_out(branch_taken)
);
```

```
msrv32_integer_file IRF(
    .ms_riscv32_mp_clk_in(ms_riscv32_mp_clk_in),
    .ms_riscv32_mp_rst_in(ms_riscv32_mp_rst_in),
    .rs_2_addr_in(rs2_addr),
    .rd_addr_in(rd_addr_reg),
    .wr_en_in(integer_wr_en_reg_file),
    .rd_in(wb_mux_out),
    .rs_1_addr_in(rs1_addr),
    .rs_1_out(rs1),
    .rs_2_out(rs2)
);

msrv2_wr_en_generator WREN (
    .flush_in(flush),
    .rf_wr_en_reg_in(rf_wr_en_reg),
    .csr_wr_en_reg_in(csr_wr_en_reg),
    .wr_en_int_file_out(integer_wr_en_reg_file),
    .wr_en_csr_file_out(csr_wr_en_reg_file)
);

msrv32_csr_file CSRF(
.clk_in(ms_riscv32_mp_clk_in),
.rst_in(ms_riscv32_mp_rst_in),
.wr_en_in(csr_wr_en_reg_file),
.csr_addr_in(csr_addr_reg),
.csr_op_in(csr_op_reg),
.csr_uimm_in(imm_reg[4:0]),
.csr_data_in(rs1_reg),
.csr_data_out(csr_data),
.pc_in(pc_reg2),
.iadder_in(iadder_out_reg),
.e_irq_in(ms_riscv32_mp_eirq_in),
.t_irq_in(ms_riscv32_mp_tirq_in),
.s_irq_in(ms_riscv32_mp_sirq_in),
.i_or_e_in(i_or_e),
.set_cause_in(set_cause),
.cause_in(cause),
.set_epc_in(set_epc),
.instret_inc_in(instret_inc),
.mie_clear_in(mie_clear),
.mie_set_in(mie_set),
.misaligned_exception_in(misaligned_exception),
.mie_out(mie),
.meie_out(meie),
.mtie_out(mtie),
.msie_out(msie),
.meip_out(meip),
.mtip_out(mtip),
.msip_out(msip),
.real_time_in(ms_riscv32_mp_rc_in),
.epc_out(epc),
.trap_address_out(trap_address)
);

msrv32_machine_control MC (
.clk_in(ms_riscv32_mp_clk_in),
.reset_in(ms_riscv32_mp_rst_in),
.illegal_instr_in(illegal_instr),
.misaligned_load_in(misaligned_load),
.misaligned_store_in(misaligned_store),
.misaligned_instr_in(misaligned_instr),
.opcode_6_to_2_in(opcode[6:2]),
.funct3_in(funct3),
.funct7_in(funct7),
.rs1_addr_in(rs1_addr),
.rs2_addr_in(rs2_addr),
.rd_addr_in(rd_addr),
.e_irq_in(ms_riscv32_mp_eirq_in),
.t_irq_in(ms_riscv32_mp_tirq_in),
.s_irq_in(ms_riscv32_mp_sirq_in),
.mie_in(mie), //mie use this
.meie_in(meie),
.mtie_in(mtie),
.msie_in(msie),
.meip_in(meip),
.mtip_in(mtip),
.msip_in(msip),
.i_or_e_out(i_or_e),
.set_epc_out(set_epc),
.set_cause_out(set_cause),
.cause_out(cause),
.instret_inc_out(instret_inc),
.mie_clear_out(mie_clear),
.mie_set_out(mie_set),
.misaligned_exception_out(misaligned_exception),
//misaligned_exception use this local wire
.pc_src_out(pc_src),
.flush_out (flush),
.trap_taken_out(trap_taken)
);

msrv32_reg_block_2 REG2 (
.rd_addr_in(rd_addr),
.csr_addr_in(csr_addr),
.rs1_in(rs1),
.rs2_in(rs2),
.pc_in(pc),
.pc_plus_4_in(pc_plus_4),
.alu_opcode_in(alu_opcode),
.load_size_in(load_size),
.load_unsigned_in(load_unsigned),
.alu_src_in(alu_src),
.csr_wr_en_in(csr_wr_en),
.rf_wr_en_in(rf_wr_en),
.wb_mux_sel_in(wb_mux_sel),
.csr_op_in(csr_op),
.imm_in(imm),
.iadder_out_in(iaddr),
.branch_taken_in(branch_taken),
.reset_in(ms_riscv32_mp_rst_in),
.clk_in(ms_riscv32_mp_clk_in),

.rd_addr_reg_out(rd_addr_reg),
.csr_addr_reg_out(csr_addr_reg),
.rs1_reg_out(rs1_reg),
.rs2_reg_out(rs2_reg),
.pc_reg_out(pc_reg2),
.pc_plus_reg_out(pc_plus_4_reg),
.alu_opcode_reg_out(alu_opcode_reg),
.load_size_reg_out(load_size_reg),
.load_unsigned_reg_out(load_unsigned_reg),
.alu_src_reg_out(alu_src_reg), //
.csr_wr_en_reg_out(csr_wr_en_reg),
.rf_wr_en_reg_out(rf_wr_en_reg),
.wb_mux_sel_reg_out(wb_mux_sel_reg),
.csr_op_reg_out(csr_op_reg),
.imm_reg_out(imm_reg),
.iadder_out_reg_out(iadder_out_reg)
);

msrv32_load_unit LU (
.ahb_resp_in(ms_riscv32_mp_hresp_in),
.ms_riscv32_mp_dmdata_in(ms_riscv32_mp_data_in),
.iadder_out_1_to_0_in(iadder_out_reg[1:0]),
.load_unsigned_in(load_unsigned_reg),
.load_size_in(load_size_reg),
.lu_output_out(lu_output)
);

msrv32_alu ALU (
.op_1_in(rs1_reg),
.op_2_in(alu_2nd_src_mux),
.opcode_in(alu_opcode_reg),
.result_out(alu_result)
);



msrv32_wb_mux_sel_unit WBMUX(
.alu_src_reg_in(alu_src_reg),
.wb_mux_sel_reg_in(wb_mux_sel_reg),
.alu_result_in(alu_result),
.lu_output_in(lu_output),
.imm_reg_in(imm_reg),
.iadder_out_reg_in(iadder_out_reg),
.csr_data_in(csr_data),
.pc_plus_4_reg_in(pc_plus_4_reg),
.rs2_reg_in(rs2_reg),

.wb_mux_out(wb_mux_out),
.alu_2nd_src_mux_out(alu_2nd_src_mux)
);

Endmodule
```

RTL Design