VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

HO CHI MINH UNIVERSITY OF TECHNOLOGY

ASSIGNMENT REPORT IC DESIGN – EE3165

LAB 4: ARITHMETRIC UNIT AND REGISTER FILE

CLASS L04 – GROUP 14

TUTOR : NGUYỄN PHAN THIÊN PHÚC

STUDENTS	ID	EMAIL	CONTRIBUTION
Bùi Khánh Hoàng	2113392	*hoang.bui060703@hcmut.edu.vn*	100%
Lê Đức Lân	2111634	*lan.le050203@hcmut.edu.vn*	100%
Lê Duy Thức	2110184	*hoang.nguyen16032003@hcmut.edu.vn*	100%

Ho Chi Minh City, May 2024

LABORATORY 4 : ARITHMETRIC UNIT AND REGISTER FILE

OBJECTIVES

No.	Objectives	Requirements
1	Implement a simple Arithmetic Logic Unit (ALU) performing eight functions with 4-bit Flags output (N, Z, C, V): ➢ Add and subtract. ➢ Logic shift right and Logic shift left. ➢ AND, OR, and NOT. ➢ Forwarding input.	▪ Present a block diagram illustrating the architecture of the ALU and include the HDL code implementing this design. Besides, provide waveforms to prove that the design functions correctly. ▪ Propose a transistor-level circuit for this ALU. ▪ Run transient simulations and show waveforms due to the existing testbench to prove your circuit works correctly. ▪ Define speed of your design.
2	Implement an 8 × 8 Register File.	▪ According to the existing block diagram, write HDL code to implement the system and show waveforms to prove that your design works correctly, using the testbench proposed by students. ▪ Propose a suitable transistor-level circuit for this system. ▪ Run transient simulations and display waveforms to demonstrate that the system operates correctly, utilizing the previous testbench for the HDL code. ▪ Define the clock frequency and maximum frequency of your design.

EXPERIMENT 1 : Implement a simple ALU performing 8 functions with 4 bits flags output

- Verification plan

Section	Item	Description	Testcase name	Owner	Status
1	Reset	Khi rst_n = 0 ouput bằng 0, khi rst_n =1, output sẽ được cập nhật bình thường theo chương trình	reset_test		PASS
2	Sum	Khi rst_n = 1 và op = 3'b000, result = a + b, carry = 1 khi a + b > 4'b1111	sum_test		PASS
3	Subtract	Khi rst_n = 1 và op = 3'b001, result = a - b, carry =1	sub_test		PASS
4	And	Khi rst_n = 1 và op = 3'b010, result = a and b, carry = 0.	and_test		PASS
5	Or	Khi rst_n = 1 và op = 3'b011, result = a or b, carry = 0.	or_test		PASS
6	Xor	Khi rst_n = 1 và op = 3'b100, result = a xor b, carry = 0.	xor_test		PASS
7	Not	Khi rst_n = 1 và op = 3'b101, result = ~a, carry = 0.	not_test		PASS
8	Shift right	Khi rst_n = 1 và op = 3'b110, result = a >> b, carry = 0.	shift_right_test		PASS
9	Shift left	Khi rst_n = 1 và op = 3'b111, result = a << b, carry = 0.	Shift_left_test		PASS

Signal	Width	Type	Desciption
clk	1	Input	Tín hiệu Clock
rst_n	1	Input	Mỗi cạnh xuống của tín hiệu, khi rst_n = 0 ouput bằng 0, khi rst_n =1, output sẽ được cập nhật bình thường theo chương trình.
a	4	Input	Đối số thứ nhất
b	4	Input	Đối số thứ hai
op	3	Input	Tùy chọn tính toán
result	4	Output	Kết quả tính toán
carry	1	Output	Cờ Carry

Table: Input/Output description

Operation (Op)	Description
3'b000	{carry, result} = a + b
3'b001	{carry, result} = a - b
3'b010	carry = 0; result = and(a, b)
3'b011	carry = 0; result = or(a, b)
3'b100	carry = 0; result = xor(a, b)
3'b101	carry = 0; result = not(a)
3'b110	carry = 0; result = a >> b
3'b111	carry = 0; result = a << b

Table: Functional description

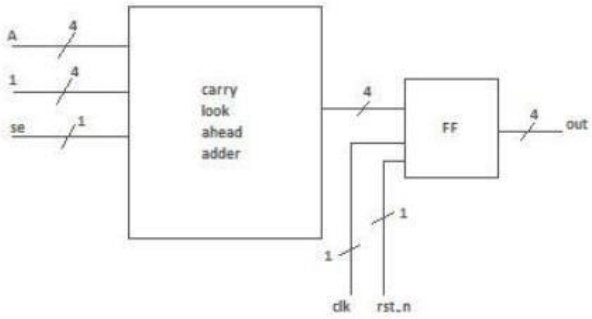
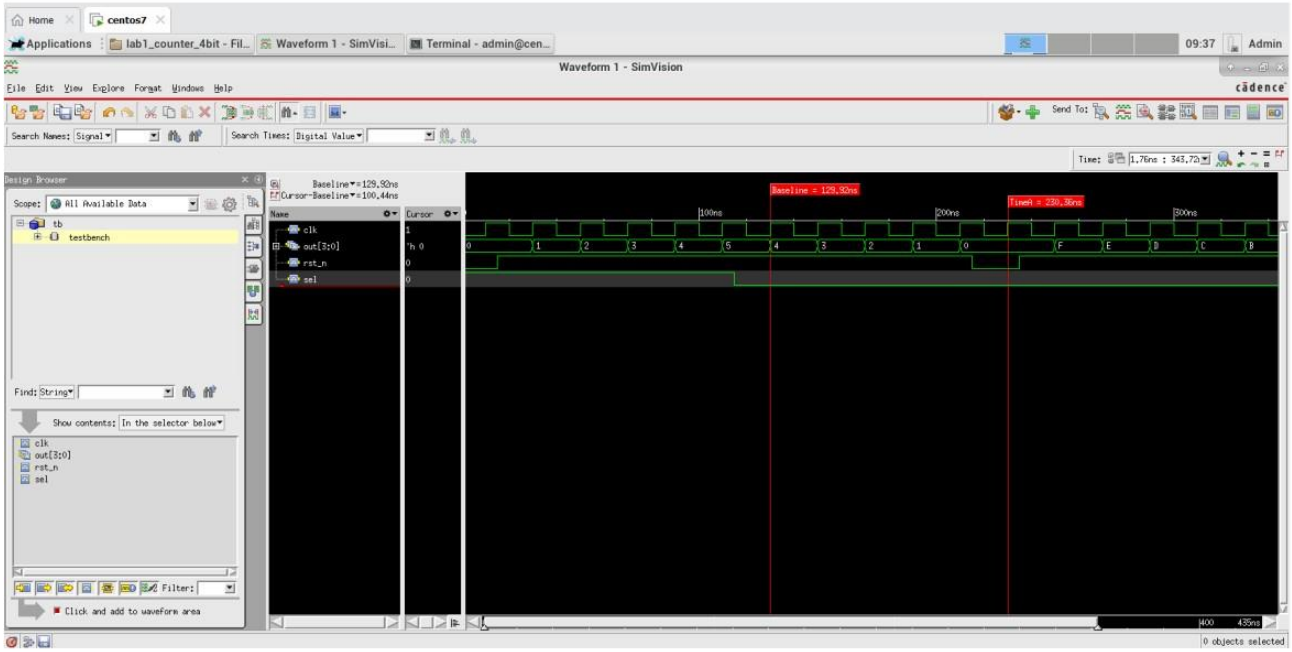
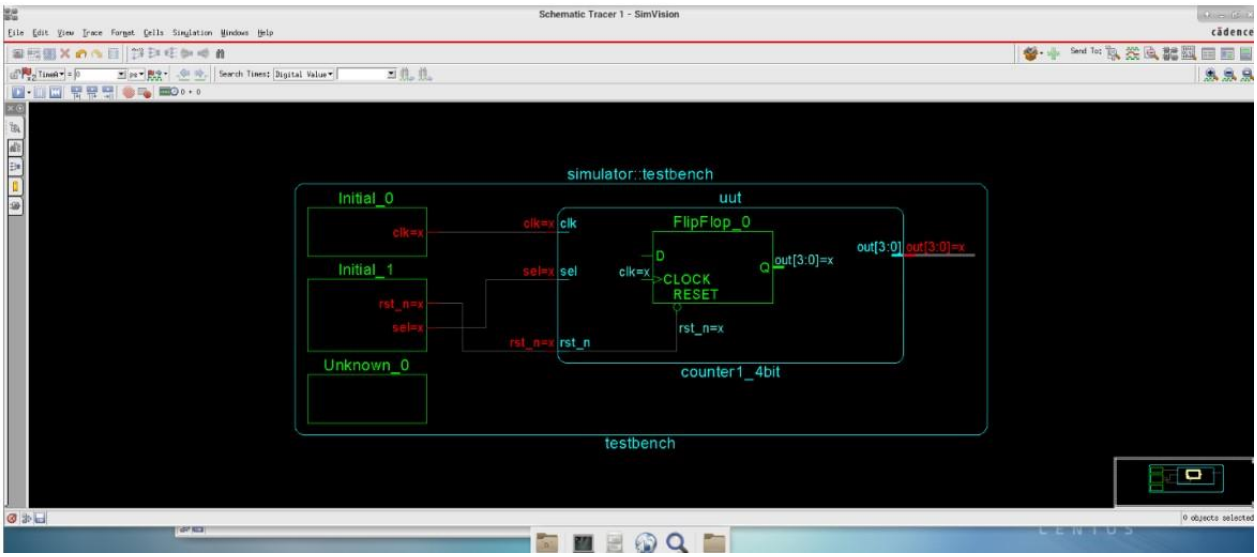


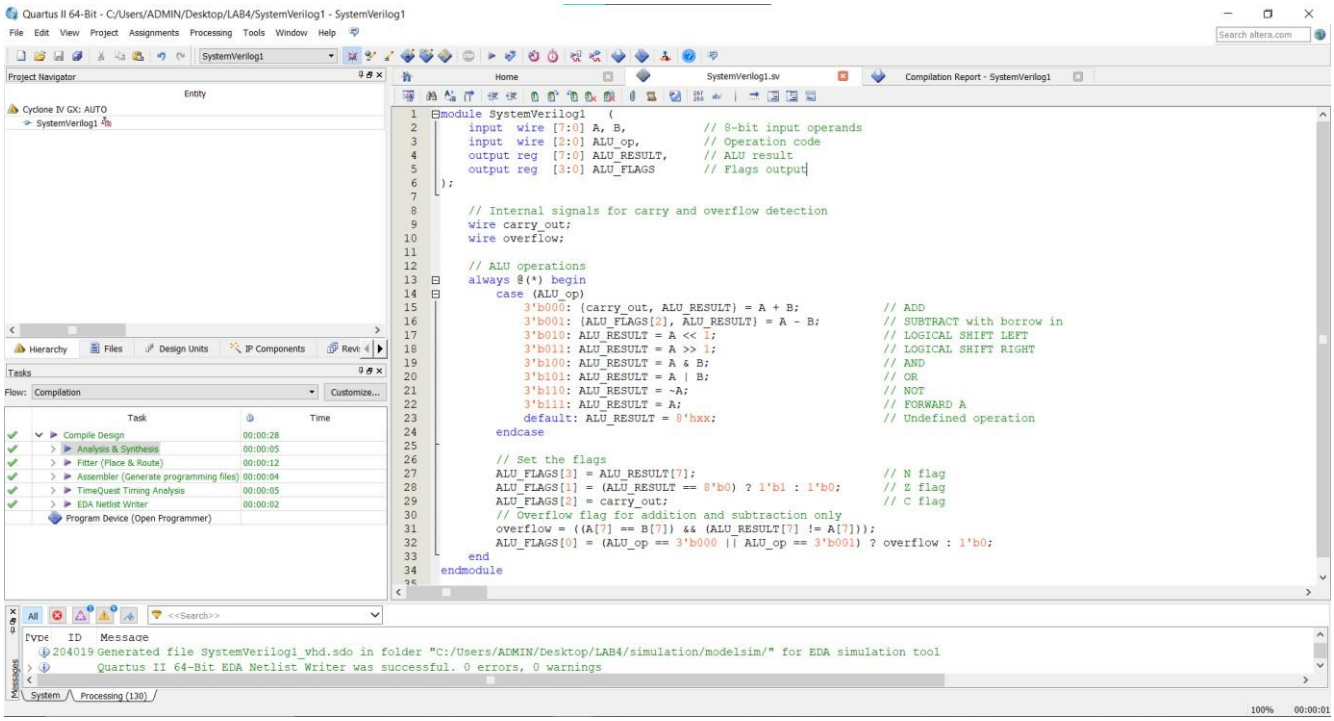
Figure: Block diagram of ALU



Waveform when type comand make sim GUI="-gui", it show signal count up when sec in high and count down when sec in low. Plan operated right.



Excecuting code



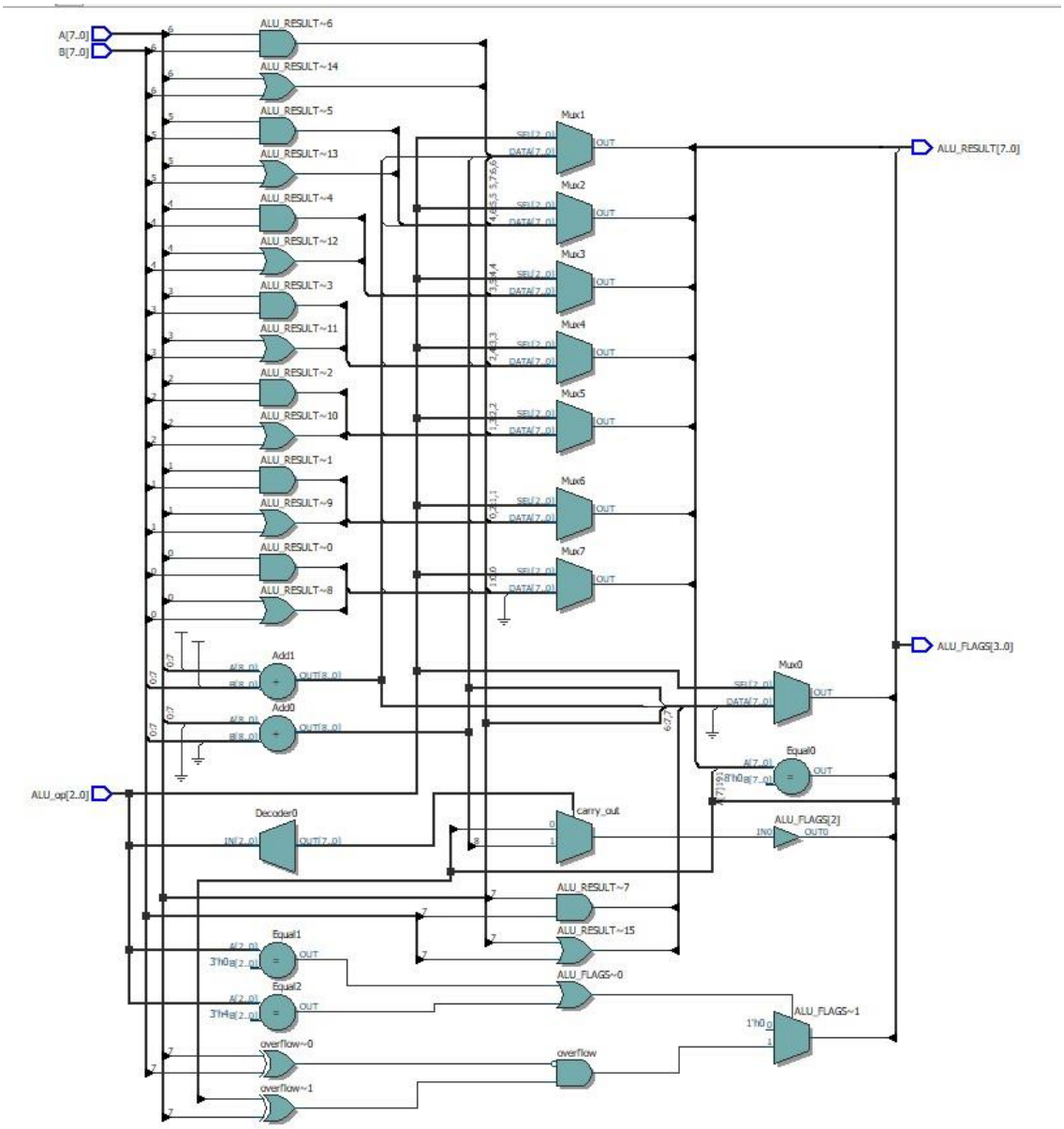
```
module SystemVerilog1   (
    input  wire [7:0] A, B,          // 8-bit input operands
    input  wire [2:0] ALU_op,        // Operation code
    output reg  [7:0] ALU_RESULT,    // ALU result
    output reg  [3:0] ALU_FLAGS      // Flags output
);

    // Internal signals for carry and overflow detection
    wire carry_out;
    wire overflow;

    // ALU operations
    always @(*) begin
        case (ALU_op)
            3'b000: {carry_out, ALU_RESULT} = A + B;            // ADD
            3'b001: {ALU_FLAGS[2], ALU_RESULT} = A - B;         // SUBTRACT with borrow in
            3'b010: ALU_RESULT = A << 1;                        // LOGICAL SHIFT LEFT
            3'b011: ALU_RESULT = A >> 1;                        // LOGICAL SHIFT RIGHT
            3'b100: ALU_RESULT = A & B;                         // AND
            3'b101: ALU_RESULT = A | B;                         // OR
            3'b110: ALU_RESULT = ~A;                            // NOT
            3'b111: ALU_RESULT = A;                             // FORWARD A
            default: ALU_RESULT = 8'hxx;                        // Undefined operation
        endcase

        // Set the flags
        ALU_FLAGS[3] = ALU_RESULT[7];                           // N flag
        ALU_FLAGS[1] = (ALU_RESULT == 8'b0) ? 1'b1 : 1'b0;      // Z flag
        ALU_FLAGS[2] = carry_out;                               // C flag
        // Overflow flag for addition and subtraction only
        overflow = ((A[7] == B[7]) && (ALU_RESULT[7] != A[7]));
        ALU_FLAGS[0] = (ALU_op == 3'b000 || ALU_op == 3'b001) ? overflow : 1'b0;
    end
endmodule
```

Simulation on quartus


A[7..0]
B[7..0]
ALU_op[2..0]
ALU_RESULT~6
ALU_RESULT~14
ALU_RESULT~5
ALU_RESULT~13
ALU_RESULT~4
ALU_RESULT~12
ALU_RESULT~3
ALU_RESULT~11
ALU_RESULT~2
ALU_RESULT~10
ALU_RESULT~1
ALU_RESULT~9
ALU_RESULT~0
ALU_RESULT~8
Mux1
Mux2
Mux3
Mux4
Mux5
Mux6
Mux7
Mux0
ALU_RESULT[7..0]
ALU_FLAGS[3..0]
Add1
Add0
Equal0
Equal1
Equal2
Decoder0
carry_out
ALU_FLAGS[2]
ALU_RESULT~7
ALU_RESULT~15
ALU_FLAGS~0
ALU_FLAGS~1
overflow~0
overflow~1
overflow

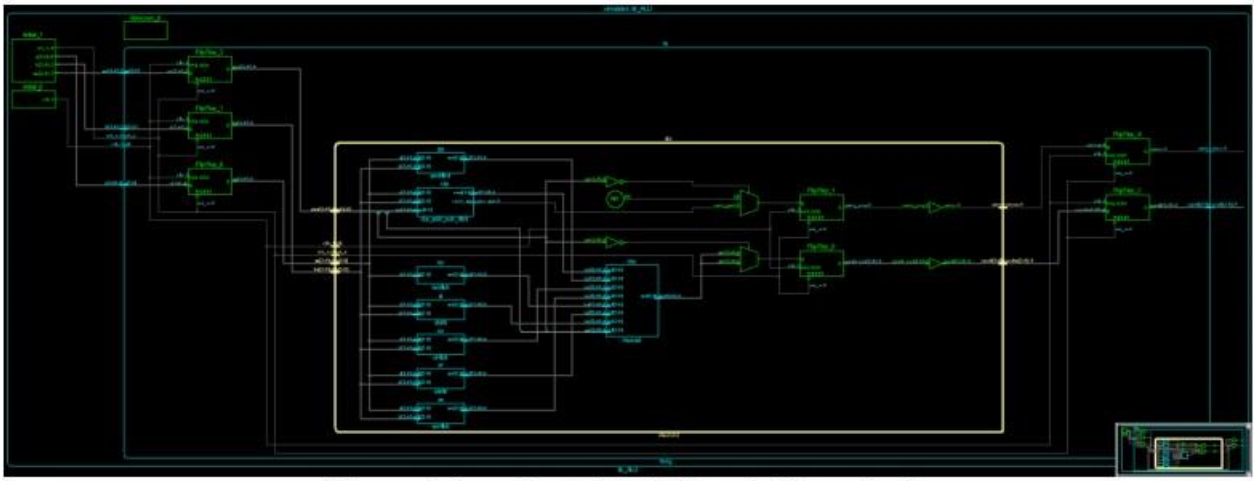


Figure: Schematic của bộ ALU trước khi synthesis

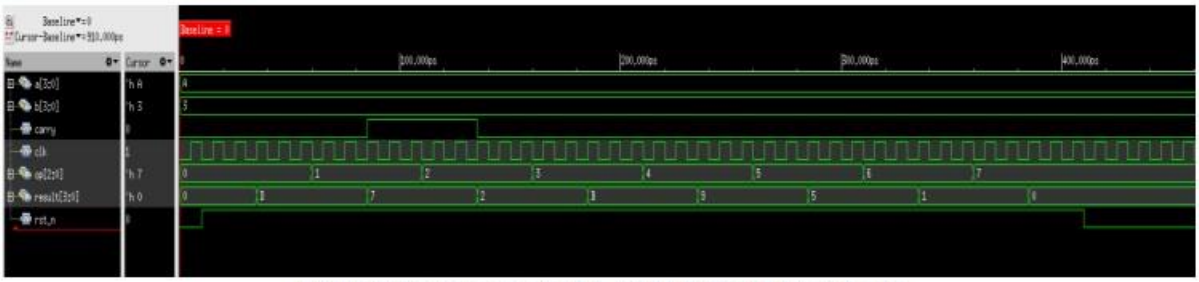


Figure: Waveform của bộ ALU trước khi synthesis

EXPERIMENT 2 : Implement 8x8 Register File

EXPERIMENT 2

Objective: Implement an 8 × 8 Register File.

Requirements:

- According to the existing block diagram, write HDL code to implement the system and show waveforms to prove that your design works correctly, using the testbench proposed by students.
- Propose a suitable transistor-level circuit for this Register File.
- Run transient simulations and display waveforms to demonstrate that the system operates correctly, utilizing the previous testbench for the HDL code.
- Define the clock frequency and maximum frequency of your design.

HDL CODE :

```
module register_file (
  input logic       clk,        // clock
  input logic       rst_n,      // reset
  input logic [2:0]  rd_addr1,    // first address
```

```
    input logic [2:0]   rd_addr2,      // second address
    input logic [2:0]   wrt_dest,      // Address to be written
    input logic [7:0]   wrt_data,      // Data to be written
    input logic         wrt_en,        // Write en signal
    output logic [7:0]  rd_data1,      // read data first address
    output logic [7:0]  rd_data2       // read data second address
);

    logic [7:0] registers [7:0];

    // Read process
    assign rd_data1 = registers[rd_addr1];
    assign rd_data2 = registers[rd_addr2];

    // Write process
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            for (int i = 0; i < 8; i++) begin
                registers[i] <= 8'b0;
            end
        end else if (wrt_en) begin
            registers[wrt_dest] <= wrt_data;
        end
    end
Endmodule
```

```
module register_file (
    input logic         clk,            // clock
    input logic         rst_n,          // reset
    input logic [2:0]   rd_addr1,       // first address
    input logic [2:0]   rd_addr2,       // second address
    input logic [2:0]   wrt_dest,       // Address to be written
    input logic [7:0]   wrt_data,       // Data to be written
    input logic         wrt_en,         // Write en signal
    output logic [7:0]  rd_data1,       // read data first address
    output logic [7:0]  rd_data2        // read data second address
);

    logic [7:0] registers [7:0];

    // Read process
    assign rd_data1 = registers[rd_addr1];
    assign rd_data2 = registers[rd_addr2];

    // Write process
    always_ff @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            for (int i = 0; i < 8; i++) begin
                registers[i] <= 8'b0;
            end
        end else if (wrt_en) begin
            registers[wrt_dest] <= wrt_data;
        end
    end
endmodule
```

```
module top_level (
    input logic         clk,        // Clock
    input logic         rst_n,      // active low reset
    input logic [2:0]   wrt_dest,   // Write address
    input logic [7:0]   wrt_data,   // Data to write
    input logic         wrt_en,     // write enable
    input logic [2:0]   rd_addr1,   // Read address 1
    input logic [2:0]   rd_addr2,   // Read address 2
    output logic [7:0]  rd_data1,   // Read data 1
    output logic [7:0]  rd_data2    // Read data 2
);
    register_file u_register_file (
        .clk(clk),
        .rst_n(rst_n),
        .rd_addr1(rd_addr1),
        .rd_addr2(rd_addr2),
        .wrt_dest(wrt_dest),
        .wrt_data(wrt_data),
        .wrt_en(wrt_en),
        .rd_data1(rd_data1),
        .rd_data2(rd_data2)
    );

endmodule
```