Lab 8: Register File and ALU in Verilog

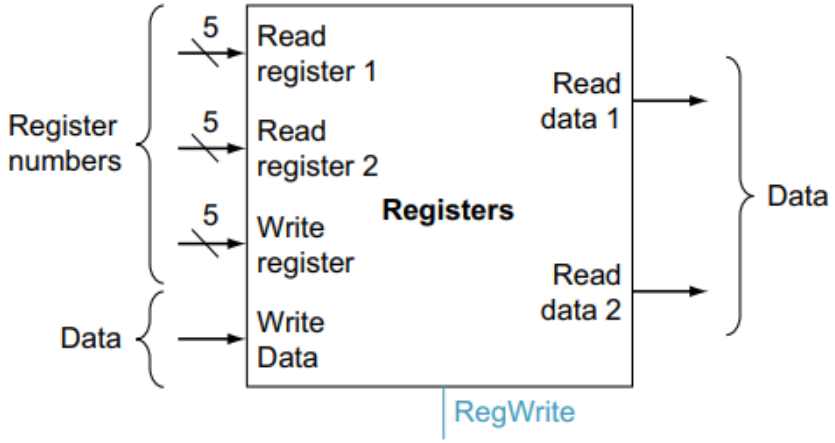
Lab Objectives:

- Develop Verilog coding proficiency by implementing a 32x32-register file.
- Understand the architecture of a register file, including registers, read/write operations, and control signals.
- Design a simple ALU in Verilog and test its functionality.
- Learn verification techniques by creating a test bench for comprehensive functional validation.
- Develop troubleshooting and debugging skills for identifying and addressing coding issues.

Register file: In RISC-V, the register file is a small, fast memory component that is used to hold the processor's general-purpose registers. The register file is typically implemented as a set of flip-flops or static random-access memory (SRAM) cells, which can be quickly read and written by the processor.

The RISC-V register file contains a set of 32 or 64-bit registers, depending on the implementation. The number of registers can vary depending on the implementation, but the standard RISC-V integer register file contains 32 registers labeled x0 to x31. Register x0 is hardwired to the value zero and cannot be written by any instruction.

The register file is used by the processor to hold intermediate results of computations, store function arguments and return values, and hold temporary values during program execution. Registers can be read by instructions that require their values, and they can be written by instructions that produce new values.



INPUT PORTS:

ReadRegister1: 5-Bit address to select a register to be read through 32-Bit output port 'ReadRegister1'.

ReadRegister2: 5-Bit address to select a register to be read through 32-Bit output port 'ReadRegister2'.

WriteRegister: 5-Bit address to select a register to be written through 32-Bit input port 'WriteRegister'.

WriteData: 32-Bit write input port.

RegWrite: 1-Bit control input signal. (0: Read and 1:Write)

OUTPUT Ports:

ReadData1: 32-Bit registered output.

ReadData2: 32-Bit registered output.

Functionality:

'ReadRegister1' and 'ReadRegister2' are two 5-bit addresses to read two registers simultaneously. The two 32-bit data sets are available on ports 'ReadData1' and 'ReadData2', respectively. 'ReadData1' and 'ReadData2' are registered outputs. You can view it as if outputs of registers specified by ReadRegister1 and ReadRegister2 are written into output registers ReadData1 and ReadData2. 'RegWrite' signal is high during the rising edge of the clock if the input data is to be written into the register file. The contents of the register specified by address 'WriteRegister' in the register file are modified at the rising edge of the clock if 'RegWrite' signal is high.

Example: 32-bit register file having 32-bit data input, 1-bit clock, 1-bit reset, 1-bit enable input and 32-bit data output. The data of the output will be updated by input data on every positive edge of the clock only if the enable input is high. The data will reset on every positive edge of reset.

Code:

```
module register_file(
input [31:0]in,
output reg [31:0]out,
input clk, reset, en
);

always @(posedge clk , posedge reset)
begin
    if(reset)
        out <= 32'h0;
    else if (en)
        out <= in;
end
```

```
endmodule
```

Test Bench:

```
module test12;

	// Inputs
	reg [31:0] in;
	reg clk;
	reg reset;
	reg en;

	// Outputs
	wire [31:0] out;

	// Instantiate the Unit Under Test (UUT)
	register_file uut (
		.in(in),
		.out(out),
		.clk(clk),
		.reset(reset),
		.en(en)
	);

	always #5 clk = ~clk;

	initial begin
		// Initialize Inputs
		in = 0;
		clk = 0;
		reset = 1;
		en = 0;

		#10 en = 1; reset = 0; in = 10;
		#10 in = 20;
		#10 in = 30;
		#10 en = 0; in = 40;
		#10 in = 50;
		#10 in = 50;
		#100 $finish;

	end

endmodule
```

ALU: An Arithmetic Logic Unit (ALU) is a fundamental component of a computer processor responsible for performing arithmetic (addition, subtraction, multiplication, division) and logical (AND, OR, NOT, XOR) operations on binary data. It takes input from memory or registers, processes the data according to the instructions provided by the control unit, and produces output results. ALUs are critical for executing mathematical calculations and logical operations necessary for computing tasks.

Example: Two bit ALU

```
module alu_2bit (
   input [1:0] A,       // First 2-bit input
   input [1:0] B,       // Second 2-bit input
   input [1:0] ALU_Sel, // ALU select signal
   output reg [2:0] Result // Output result (3 bits to account for carry/borrow)
);

always @(*) begin
   case(ALU_Sel)
      2'b00: Result = A & B;      // AND
      2'b01: Result = A | B;      // OR
      2'b10: Result = A + B;      // ADD
      2'b11: Result = A - B;      // SUB
      default: Result = 3'b000;
   endcase
end

endmodule
```

Lab Tasks:

1) Design an Arithmetic Logic Unit (ALU) using behavioral modeling of Verilog HDL and verify its functionality through a test bench. The ALU should perform various arithmetic and logical operations based on the opcode given in the table.

Opcode	Outputs
0	0
1	A+B
2	A-B
3	A&B
4	A/B
5	~A
6	NoP

2) Design a register file with the specifications shown in the figure below. The write data and read data ports are 32bits. The data (register values) will be updated on every positive edge and negedge of the clock. A reset input is used to reset every register value to zero.
Positive Edge: write data to a register. **Negative Edge:** read data from a register