Assignment Report

Bratin Mondal (21CS10016)
Somya Kumar (21CS30050)

October 18, 2023

1 Introduction

The assignment involves the implementation of a Register Bank and its integration with the ALU module, as designed earlier. The primary objective is to create a top-level module for testing the modules by implementing operations like $R_x = R_y op R_z$, where R_x, R_y, R_z, and op can be specified from outside. The designed system is also expected to be downloaded on an FPGA for testing the correctness of the operations.

2 Description of Register Bank

The Register Bank module plays a crucial role in the assignment. It manages a set of registers and allows for reading from and writing to these registers. The registers are initialized with specific values during reset, and they can be accessed by specifying the register address.

The Verilog code for the Register Bank module includes logic to read and write data to registers, as well as the initialization process during a reset. The module interfaces with the top-level module to provide access to the registers.

3 Description of TopModule

The Top Module is the core of the assignment, as it connects the Register Bank with the ALU module and controls the data flow between them. It takes inputs such as read and write control signals, register addresses, ALU operation codes, and display selection.

The Top Module instantiates the Register Bank and ALU modules, manages the data flow, and controls the output display. It also includes logic for selecting which bits to display.

4 Mapping to FPGA Pins

In this section, we describe how various Verilog signals from the 'topModule' are mapped to specific FPGA pins using the Xilinx Constraints Language (XDC). These pin assignments are crucial for the correct operation of the design on the FPGA board.

4.1 Clock Signal

The clock signal (`clk`) is essential for synchronizing the entire design. It is mapped to FPGA pin E3 with an IOSTANDARD of LVCMOS33. The corresponding constraint is as follows:

```
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }];
```

The clock signal has a specified period of 10.00 ns with a waveform of 0 to 5, ensuring proper clocking of the design.

4.2 Switches

The assignment utilizes switches (`opcode[0:3]`, `rs[0:2]`, `rt[0:2]`, `rd[0:2]`, `write`, `read`, `dispBit`) for various control and data inputs. Here's how these switches are mapped to FPGA pins:

- `opcode[0:3]` switches are mapped to FPGA pins J15, L16, M13, and R15 with an IOSTANDARD of LVCMOS33. These switches determine the ALU operation code.
- `rt[0:2]` switches are mapped to FPGA pins R17, T18, and U18 with an IOSTANDARD of LVCMOS33. These switches specify the target register address.
- `rs[0:2]` switches are mapped to FPGA pins R13, T8, and U8 with IOSTANDARD LVCMOS18. These switches define the source register address.
- `rd[0:2]` switches are mapped to FPGA pins R16, T13, and H6 with an IOSTANDARD of LVCMOS33. These switches represent the destination register address.
- `write` switch is mapped to FPGA pin U12 with an IOSTANDARD of LVCMOS33. This switch controls register write operations.
- `read` switch is mapped to FPGA pin U11 with an IOSTANDARD of LVCMOS33. This switch is used for reading data from registers.
- `dispBit` switch is mapped to FPGA pin V10 with an IOSTANDARD of LVCMOS33. This switch selects between displaying the lower or upper 16 bits of the ALU result.

4.3 LEDs

The LED outputs (`disp[0:15]`) are used to display the results. Here's how these LED outputs are mapped to FPGA pins:

- `disp[0:15]` LEDs are mapped to FPGA pins H17, K15, J13, N14, R18, V17, U17, U16, V16, T15, U14, T16, V15, V14, V12, and V11 with an IOSTANDARD of LVCMOS33. These LEDs display various bits of the result.

5 Verilog Code

Below are the relevant portions of Verilog code for the Register Bank, Top Module, and associated modules. The code has been formatted for clarity:

```verilog
`timescale 1ns/1ps

module REG_BANK (
    input wire read,              // Input wire to control read operation
    input wire write,             // Input wire to control write operation
    input wire [2:0] portIN,      // Input port for specifying the register to write
        to
    input wire [2:0] portOUT1,    // Input port for specifying the first register to
        read from
    input wire [2:0] portOUT2,    // Input port for specifying the second register to
        read from
    input wire [31:0] writePort,  // Data to be written to the selected register
    output reg [31:0] readPort1,  // Output port for the data read from the first
        register
    output reg [31:0] readPort2,  // Output port for the data read from the second
        register
    input reset,
    input clk
);
```

```verilog

    reg [31:0] registers [0:15]; // Array of registers to store data

    always @(posedge clk) begin
        if(reset == 1)begin
            registers[0] = 32'h00000001;  // Set register 0 to 2^0 = 1
            registers[1] = 32'h00000002;  // Set register 1 to 2^1 = 2
            registers[2] = 32'h00000004;  // Set register 2 to 2^2 = 4
            registers[3] = 32'h00000008;  // Set register 3 to 2^3 = 8
            registers[4] = 32'h00000010;  // Set register 4 to 2^4 = 16
            registers[5] = 32'h00000020;  // Set register 5 to 2^5 = 32
            registers[6] = 32'h00000040;  // Set register 6 to 2^6 = 64
            registers[7] = 32'h00000080;  // Set register 7 to 2^7 = 128
            registers[8] = 32'h00000100;  // Set register 8 to 2^8 = 256
            registers[9] = 32'h00000200;  // Set register 9 to 2^9 = 512
            registers[10] = 32'h00000400; // Set register 10 to 2^10 = 1024
            registers[11] = 32'h00000800; // Set register 11 to 2^11 = 2048
            registers[12] = 32'h00001000; // Set register 12 to 2^12 = 4096
            registers[13] = 32'h00002000; // Set register 13 to 2^13 = 8192
            registers[14] = 32'h00004000; // Set register 14 to 2^14 = 16384
            registers[15] = 32'h00008000; // Set register 15 to 2^15 = 32768
        end
        else begin
            if(read == 1)
                registers[portIN] = writePort; // Write data to the selected register
            if (write == 1) begin
                readPort1 = registers[portOUT1]; // Read data from the first selected
                    register
                readPort2 = registers[portOUT2]; // Read data from the second selected
                     register
            end
        end
    end

endmodule
```

Listing 1: Verilog Code for Register Bank

```verilog
module topModule(
    input read,                  // Control signal for reading from registers
    input write,                 // Control signal for writing to registers
    input [2:0] rd,              // Destination register address
    input [2:0] rs,              // Source register address
    input [2:0] rt,              // Target register address
    input [3:0] opcode,          // ALU operation code
    input dispBit,               // Display bit selection
    input rst,                   // Reset Signal
    output reg [15:0] disp,      // Output display value
    input clk                    // Clock Signal
);
    wire [31:0] result;          // ALU result
    wire [31:0] read1;           // Data read from source register
    wire [31:0] read2;           // Data read from target register

    // Instantiate the Register Bank (REG_BANK)
    REG_BANK reg_bank(
        .read(read),           // Connect the read control signal
        .write(write),         // Connect the write control signal
        .portIN(rd),           // Connect the destination register address
        .portOUT1(rs),           // Connect the source register address 1
        .portOUT2(rt),           // Connect the source register address 2
        .writePort(result),      // Connect the write data to the result
        .readPort1(read1),       // Connect data read from source register 1
        .readPort2(read2),       // Connect data read from target register 2
        .reset(rst),             // Connect the reset signal
        .clk(clk)                // Connect the clock signal
    );

    reg shamt;                     // Shift amount for ALU
    ALU alu(
        .opcode(opcode),             // Connect ALU operation code
        .A(read1),                   // Connect source operand A
        .B(read2),                   // Connect source operand B
        .shift_amount(shamt),        // Connect shift amount
        .ALU_result(result)          // Connect ALU result
    );

    always @(*)begin
        if(rst == 1)begin
            shamt = 1;
        end
    end
    always @(*) begin
        if (dispBit == 0) begin
            disp = result[15:0];     // Display lower 16 bits of the ALU result
        end else begin
            disp = result[31:16];    // Display upper 16 bits of the ALU result
        end
    end
endmodule
```

Listing 2: Verilog Code for Top Module

6 Conclusion

In conclusion, this assignment involves the design and integration of a Register Bank and ALU module in Verilog. The assignment report provides an overview of the problem statement, detailed descriptions of the Register Bank and Top Module, and information about the mapping of signals to FPGA pins. The Verilog code segments demonstrate the implementation of these modules, and the assignment is ready for further testing on an FPGA.