ES204 Digital Systems

Lab Exam

Indian Institute of Technology, Gandhinagar

Tamizhanban A G – 22110269

Yash Bachwana – 22110295

1. Verilog code:

```
`timescale 1ns / 1ps

module division (input enable, input [7:0] divisor,input [7:0] dividend,output
reg [7:0] quotient, output reg [7:0] remainder);

// Variables
integer i;
reg [7:0] divisor_reg, dividend_reg;
reg [7:0] storage;

always @(divisor or dividend or enable)
begin
    if (~enable)
    begin
        quotient = 8'b00000000;
        remainder = 8'b00000000;
    end
    else
    begin
        divisor_reg = divisor;
        dividend_reg = dividend;
        storage = 8'b00000000;
        for (i = 0; i < 8; i = i + 1) begin
            storage = {storage[6:0], dividend_reg[7]};
            dividend_reg[7:1] = dividend_reg[6:0];
            storage = storage - divisor_reg;
            if (storage[7] == 1) begin
                dividend_reg[0] = 0;
                storage = storage + divisor_reg;
            end
            else begin
                dividend_reg[0] = 1;
            end
        end

    quotient = dividend_reg;
    remainder = dividend - (divisor_reg * dividend_reg);
```

```
    end
end

endmodule


module Register_File(input mode,input [3:0] index_write,[3:0] index_read,input
[7:0] write_data,output reg [7:0] Data);
    reg [7:0] register_file [15:0]; // R0 to R15
    integer i;

    initial
    begin
        for (i = 0; i < 16; i = i + 1) begin
                register_file[i] = $random;
            end
    end

    always @(*)
    begin
            if (mode == 0) begin
                Data = register_file[index_read]; // Read
            end else begin
                register_file[index_write] = write_data; // Write
            end
        end

endmodule



module Instruction_File(input [3:0] index , output [7:0] Instruction) ;
    reg [7:0] Instruction_Set[5:0] ;

    initial
    begin
        Instruction_Set[0] = 8'b10010001;
        Instruction_Set[1] = 8'b01100001;
        Instruction_Set[2] = 8'b00010101;
        Instruction_Set[3] = 8'b01000010;
//        Instruction_Set[3] = 8'b00010110;
        Instruction_Set[4] = 8'b10100111;
        Instruction_Set[5] = 8'b11111111;
    end

    assign Instruction = Instruction_Set[index] ;

endmodule
```

```
module Processor (input [7:0] instruction , input [7:0] operand , input [7:0]
accumulator , output reg [15:0] result);
reg enable ;
wire [7:0]r, q;
initial begin
    enable = 0 ;
end
division div (enable, operand, accumulator,q, r);
    always @(*)
    begin
        case(instruction[7:4])
            4'b0001: result = operand + accumulator; // ADD
            4'b0010: result = operand - accumulator; // SUB
            4'b0011: result = operand * accumulator; // MUL
            //4'b0100: enable = 1; // DIV | need to change
            4'b0000: begin   if (instruction[3:0] == 4'b0001) result =
accumulator << 1; // LSL
                        else if (instruction[3:0] == 4'b0010) result =
accumulator >> 1; // LSR
                        else if (instruction[3:0] == 4'b0011) result =
{accumulator[0], accumulator[7:1]}; // CIR
                        else if (instruction[3:0] == 4'b0100) result =
{accumulator[6:0], accumulator[7]}; // CIL
                        else if (instruction[3:0] == 4'b0101) result =
{accumulator[7], accumulator[7:1]}; // ASR
                        else if (instruction[3:0] == 4'b0110) result =
accumulator + 1; // INCREMENT
                        else if (instruction[3:0] == 4'b0111) result =
accumulator -1 ; // DECREMENT
                    end
            4'b0101: result = operand & accumulator; // AND
            4'b0110: result = operand ^ accumulator; // XOR
            4'b0111: begin // CMP
                        if (accumulator >= operand) result = 0;
                        else result = 1;
                     end
            4'b1001: result = operand; //MOV ACC Ri
            4'b1010: result = instruction[3:0]; // MOV Ri ACC
            default: result = 0; // Default behavior when no instruction
matches
        endcase

        if (instruction[7:4] == 4'b0100)
        begin
            enable = 1 ;
            result[15:8] = r ;
            result[7:0] = q ;
```

```
        end
        else enable = 0 ;
    end
endmodule


module Control_Unit(input clk);
    reg [7:0] accumulator ;
    reg [7:0] extra_register ;
    reg cb ;
    reg [3:0] current_address ;
    wire [7:0] instruction ;
    wire [7:0] operand;
    wire [15:0] result;
    reg mode ;
    reg [3:0]index_write ;

    initial begin
        current_address = 0 ;
        mode = 0 ;
    end

    Processor
Processor_inst(.instruction(instruction),.operand(operand),.accumulator(accumu
lator),.result(result));
    Instruction_File Instruction_File_inst(current_address,instruction) ;
    Register_File
Register_File_inst(mode,index_write,instruction[3:0],accumulator,operand) ;

     always @(posedge clk) begin
        if (instruction == 8'b11111111) begin end // HALT
        else begin
            //might update branch address
            //ins = instructs[curr_address];
            case(instruction[7:4])
                4'b0001: {cb, accumulator} = result; // ADD
                4'b0010: {cb, accumulator} = result; // SUB
                4'b0011: {extra_register, accumulator} = result; // MUL
                4'b0100: {extra_register, accumulator} = result; // DIV | need
to change
                4'b0000: begin  if (instruction[3:0] == 4'b0001) accumulator =
result; // LSL
                                else if (instruction[3:0] == 4'b0010)
extra_register = result; // LSR
                                else if (instruction[3:0] == 4'b0011)
accumulator = result; // CIR
                                else if (instruction[3:0] == 4'b0100)
accumulator = result; // CIL
```

```
                                else if (instruction[3:0] == 4'b0101)
accumulator = result; // ASR
                                else if (instruction[3:0] == 4'b0110) {cb,
accumulator} = result; // INCREMENT
                                else if (instruction[3:0] == 4'b0111) {cb,
accumulator} = result; // DECREMENT
                           end
                4'b0101: accumulator = result; // AND
                4'b0110: accumulator = result; // XOR
                4'b0111: begin // CMP
                            if (accumulator >= operand) cb = result;
                            else cb = result;
                         end
                4'b1001: accumulator = result; //MOV ACC Ri
                4'b1010: index_write = result ;
                4'b1000: if (cb == 1) begin
                            //branch_address = curr_address;
                            current_address = operand;  //BRANCH
                            //branch_flag = 1;
                            end
                4'b1011: current_address = operand; //RETURN
                default: begin end
               endcase

               if (instruction[7:4] == 4'b1010) mode = 1 ;
               else mode = 0 ;

               current_address = current_address + 1 ;
            end

    end
endmodule
```

2. Testbench code:

```
`timescale 1ns / 1ps

module tb();
    reg clk;
    reg [3:0] Indexing ;
    wire [3:0]Accum ;
    wire [7:0]regis ;
    Control_Unit uut(clk,Indexing,Accum,regis);

    initial begin
    clk = 0;
        forever #5 clk = ~clk;
    end
```

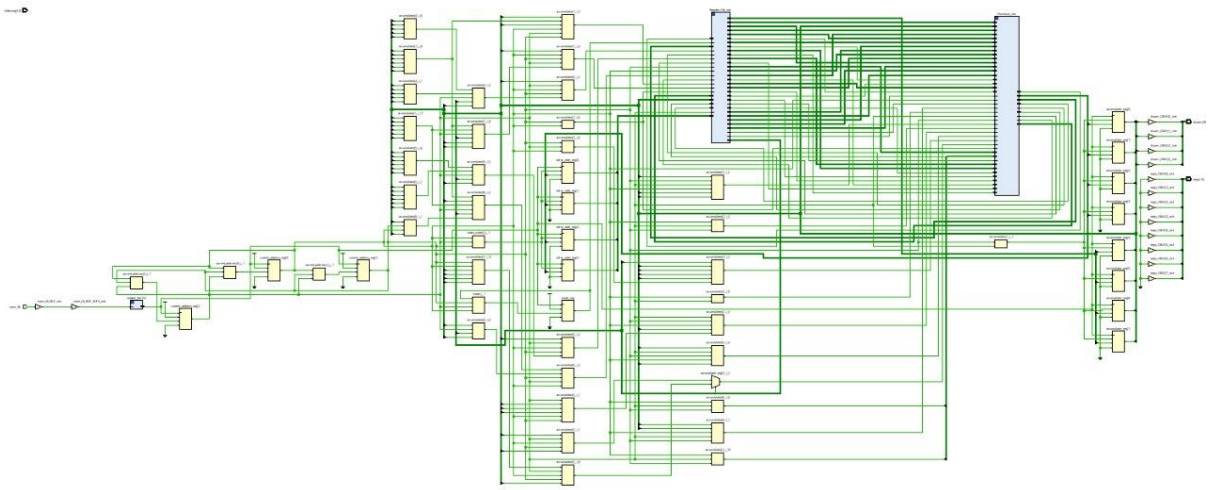
```
    initial begin
    Indexing = 0 ;
    #20 ;
    Indexing = 3 ;
    #20 ;
    Indexing = 5 ;
    #20 ;
    $finish;
    end

endmodule
```

3. XDC File:

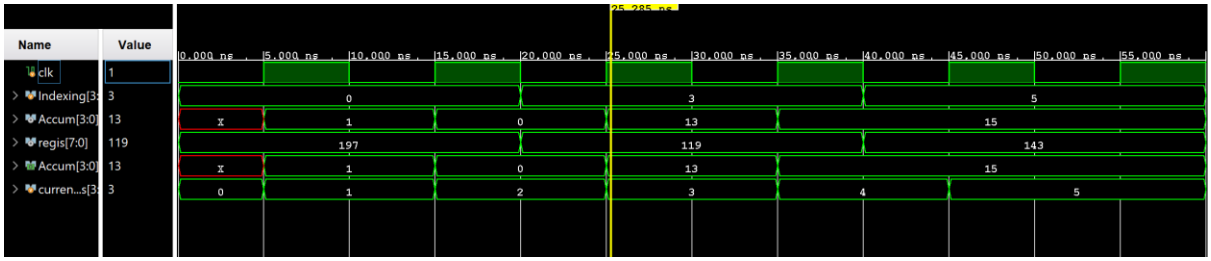
```
set_property PACKAGE_PIN L1 [get_ports {Accum[3]}]
set_property PACKAGE_PIN P1 [get_ports {Accum[2]}]
set_property PACKAGE_PIN N3 [get_ports {Accum[1]}]
set_property PACKAGE_PIN P3 [get_ports {Accum[0]}]
set_property PACKAGE_PIN R2 [get_ports {Indexing[3]}]
set_property PACKAGE_PIN T1 [get_ports {Indexing[2]}]
set_property PACKAGE_PIN U1 [get_ports {Indexing[1]}]
set_property PACKAGE_PIN W2 [get_ports {Indexing[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Accum[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Accum[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Accum[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Accum[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Indexing[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Indexing[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Indexing[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Indexing[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {regis[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {regis[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {regis[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {regis[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {regis[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {regis[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {regis[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {regis[0]}]
set_property PACKAGE_PIN W5 [get_ports main_clk]
set_property IOSTANDARD LVCMOS33 [get_ports main_clk]
set_property PACKAGE_PIN W3 [get_ports {regis[7]}]
set_property PACKAGE_PIN V3 [get_ports {regis[6]}]
set_property PACKAGE_PIN V13 [get_ports {regis[5]}]
set_property PACKAGE_PIN V14 [get_ports {regis[4]}]
set_property PACKAGE_PIN U14 [get_ports {regis[3]}]
set_property PACKAGE_PIN U15 [get_ports {regis[2]}]
set_property PACKAGE_PIN W18 [get_ports {regis[1]}]
```

```
set_property PACKAGE_PIN V19 [get_ports {regis[0]}]
```

4. Block Diagram (Schematic):



5. Simulation Files:



We used the sample code for the simulation waveform:

```
MOV ACC, R1 ; Load R1 in ACC
XRA R1 ; clears ACC
ADD R5 ; ACC+R5
ADD R6 ; ACC + R6 (which is R5+R6)
MOV R7, ACC
HLT
```

6. Explanation of the processor design:

We have used 4 modules:

- Control unit
- Register file
- Instruction file
- Division

The control unit is the main or the top module under which we have instantiated the above modules. The instruction file module contains all the instructions to be performed as the program counter increments from one to six. The register file contains all the 15 registers in which operands are present and the output or the result is sent to the accumulator.