ES204 Digital Systems
Lab Exam

Indian Institute of Technology, Gandhinagar

220 marks

Dt. 13.04.2024

Final Demo:

FPGA implementation of the Complete “Tiny” Processor Design which runs one program.

Submit:
- Verilog Codes
- Testbench
- XDC Files
- Block diagram showing all the Verilog modules and how then interact
- Simulation results
- FPGA implementation Video any one program with at least 4 instructions.

(Max 30 marks for report.)

The following processor has a register file consisting of 16 registers each of 8 bit. The processor can execute the following instructions. The instructions that need 2 operands will take one of the operand from the Register file and another from the accumulator. The result will be transferred to Accumulator. There is an 8-bit extended (EXT) register used only during multiplication and division operation. This register stores the higher order bits during multiplication and quotient during division. The C/B register holds the carry and borrow during addition and subtraction, respectively.

Note:
- Each instruction takes 1 clock cycle.
- Division operation can never have the 0 as divisor.
- Branch instruction can only branch within the program.

Instruction format:

Direct instruction

Operation code	Register address

Branch Instruction

Operation code	4-bit address (label)

Instruction set:

Instruction Opcode	Operation	Explanation
0000 0000	NOP	No operation
0001 xxxx	ADD Ri	Add ACC with Register contents and store the result in ACC. Updates C/B
0010 xxxx	SUB Ri	Subtract ACC with Register contents and store the result in ACC. Updates C/B
0011 xxxx	MUL Ri	Multiple ACC with Register contents and store the result in ACC. Updates EXT
0100 xxxx	DIV Ri	Divides ACC with Register contents and store the Quotient in ACC. Updates EXT with remainder.

0000 0001	LSL ACC	Left shift left logical the contents of ACC. Does not update C/B
0000 0010	LSR ACC	Left shift right logical the contents of ACC. Does not update C/B
0000 0011	CIR ACC	Circuit right shift ACC contents. Does not update C/B
0000 0100	CIL ACC	Circuit left shift ACC contents. Does not update C/B
0000 0101	ASR ACC	Arithmetic Shift Right ACC contents
0101 xxxx	AND Ri	AND ACC with Register contents (bitwise) and store the result in ACC. C/B is not updated
0110 xxxx	XRA Ri	XRA ACC with Register contents (bitwise) and store the result in ACC. C/B is not updated
0111 xxxx	CMP Ri	CMP ACC with Register contents (ACC-Reg) and update C/B. If ACC>=Reg, C/B=0, else C/B=1
0000 0110	INC ACC	Increments ACC, updates C/B when overflows
0000 0111	DEC ACC	Decrements ACC, updates C/B when underflows
1000 xxxx	Br <4-bit address>	PC is updated and the program Branches to 4-bit address if C/B=1
1001 xxxx	MOV ACC, Ri	Moves the contents of Ri to ACC
1010 xxxx	MOV Ri, ACC	Moves the contents of ACC to Ri
1011 xxxx	Ret <4-bit address>	PC is updated, and the program returns to the called program.
1111 1111	HLT	Stop the program (last instruction)

Note:

- Avoid trying to design one large monolithic code. See how you can partition the design into smaller modules and then implement them, test them using detailed test bench codes, and synthesize them.
- Combine them together after detailed testing and synthesis is done.
- Finally write a meaningful program based on this set and show its running using FPGA.

The final implementation should be shown on FPGA.

Sample code:

Add the contents of R5 and R6, and store the result in R7.

```
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
```

Lab Assignment 5 ES 204

Birudugadda Srivibhav (22110050)
Reddybathuni Venkat (22110220)

Register file swap

1. Code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 20.04.2024 15:11:14
// Design Name:
// Module Name: Final_lab
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module main_module #(parameter n = 8, k = 16)(
    input clk,en,reset,
    output reg [n-1:0] OUT1, OUT2
    );
    reg [n - 1:0] registers [0:k - 1];
    reg [n-1:0] commands [0 : k-1];
    reg C_B;
    reg [n-1:0] Acc, EXT;
    reg [4:0]PC = 0;
    reg [n-1:0] b;
    integer i;
    always @(posedge clk) begin
```

```
if (en & reset == 0) begin
registers[0] <= 8'b000;
registers[1] <= 8'b001;
registers[2] <= 8'b010;
registers[3] <= 8'b011;
registers[4] <= 8'b100;
registers[5] <= 8'b101;
registers[6] <= 8'b110;
registers[7] <= 8'b111;
registers[8] <= 8'b1000;
registers[9] <= 8'b1001;
registers[10] <= 8'b1010;
registers[11] <= 8'b1011;
registers[12] <= 8'b1100;
registers[13] <= 8'b1101;
registers[14] <= 8'b1110;
registers[15] <= 8'b1111;
commands[0] <= 8'b10010001;
commands[1] <= 8'b01100001;
commands[2] <= 8'b00010101;
commands[3] <= 8'b00010110;
commands[4] <= 8'b10100111;
commands[5] <= 8'b00000111;
commands[6] <= 8'b11111111;

OUT1 <= 0; // Reset outputs
OUT2 <= 0;
C_B <= 0;
end
else if(en & reset) begin
if(PC < k) begin
   if(commands[PC][n-1:n/2] == 4'b0001)
   begin
      {C_B, Acc} <= Acc + registers[commands[PC][n/2-1:0]];
      OUT1 <= Acc;
      OUT2 <= registers[commands[PC][n/2-1:0]];
   end
   else if(commands[PC][n-1:n/2] == 4'b1001)
   begin
      Acc <= registers[commands[PC][n/2-1:0]];
      OUT1 <= Acc;
      OUT2 <= registers[commands[PC][n/2-1:0]];
   end
   else if(commands[PC][n-1:n/2] == 4'b1010)
```

```
        begin
           registers[commands[PC][n/2-1:0]] <= Acc;
           OUT1 <= Acc;
           OUT2 <= registers[commands[PC][n/2-1:0]];
        end
        else if(commands[PC][n-1:n/2] == 4'b0110)
        begin
           Acc <= Acc ^ registers[commands[PC][n/2-1:0]];
           OUT1 <= Acc;
           OUT2 <= registers[commands[PC][n/2-1:0]];
        end
        else if(commands[PC] == 8'b11111111)
        begin
           PC <= k;
        end
        else if(commands[PC][n-1:n/2] == 4'b0010)
        begin
           C_B <= (Acc < registers[commands[PC][n/2-1:0]]);
           if(Acc < registers[commands[PC][n/2-1:0]]) begin
           Acc <= registers[commands[PC][n/2-1:0]] - Acc;
           end
           else begin
           Acc <= Acc - registers[commands[PC][n/2-1:0]];
           end
           OUT1 <= Acc;
           OUT2 <= registers[commands[PC][n/2-1:0]];
        end
        else if(commands[PC][n-1:n/2] == 4'b0011)
        begin
           {EXT, Acc} = Acc * registers[commands[PC][n/2-1:0]];
           OUT1 <= Acc;
           OUT2 <= registers[commands[PC][n/2-1:0]];
        end
        else if(commands[PC][n-1:n/2] == 4'b0100)
        begin
//            write it
           b = registers[commands[PC][(n/2)-1:0]];
           EXT = 0;
           for (i = 0; i < n; i = i + 1) begin
           EXT = {EXT[(n-2):0], Acc[(n-1)]}; // Shifting EXT to the left by one and adding the most
significant bit of Acc
           Acc[(n-1):1] = Acc[(n-2):0]; // Shifting Acc to the left by one
           EXT = EXT - b; // Subtracting b from EXT
           if (EXT[(n-1)] == 1) begin // Checking if the most significant bit of EXT is 1
```

```
    Acc[0] = 0;
    EXT = EXT + b; // Restoring EXT if subtraction resulted in a negative value
    end
    else Acc[0] = 1;
    end
    OUT1 = Acc;
    OUT2 = registers[commands[PC][n/2-1:0]];
end
else if(commands[PC] == 8'b0000001)
begin
    Acc = {Acc[n-2:0], 1'b0};
    OUT1 <= Acc;
    OUT2 <= registers[commands[PC]];
end
else if(commands[PC] == 8'b0000010)
begin
    Acc <= {1'b0, Acc[n-1:1]};
    OUT1 <= Acc;
    OUT2 <= registers[commands[PC]];
end
else if(commands[PC] == 8'b0000011)
begin
    Acc <= {Acc[0], Acc[n-1:1]};
    OUT1 <= Acc;
    OUT2 <= registers[commands[PC]];
end
else if(commands[PC] == 8'b00000100)
begin
    Acc <= {Acc[n-2:0], Acc[n-1]};
    OUT1 <= Acc;
    OUT2 <= registers[commands[PC]];
end
else if(commands[PC] == 8'b00000101)
begin
    Acc <= {Acc[n-1], Acc[n-1:1]};
    OUT1 <= Acc;
    OUT2 <= registers[commands[PC]];
end
else if(commands[PC][n-1:n/2] == 4'b0101)
begin
    Acc = Acc & registers[commands[PC][n/2-1:0]];
    OUT1 <= Acc;
    OUT2 <= registers[commands[PC][n/2-1:0]];
end
```

```
    else if(commands[PC][n-1:n/2] == 4'b0111)
    begin
        if(Acc >= registers[commands[PC][n/2-1:0]]) begin
        C_B <= 0;
        end
        else C_B <= 1;
        OUT1 <= Acc;
        OUT2 <= registers[commands[PC][n/2-1:0]];
    end
    else if(commands[PC] == 8'b00000110)
    begin
        {C_B, Acc} = Acc + 1;
        OUT1 <= Acc;
        OUT2 <= registers[commands[PC]];
    end
    else if(commands[PC] == 8'b00000111)
    begin
        if (Acc == 0) C_B <= 1;
        else C_B <= 0;
        Acc <= Acc - 1;
        OUT1 <= Acc;
        OUT2 <= registers[commands[PC]];
    end
    else if(commands[PC][n-1:n/2] == 4'b1000)
    begin
        if(C_B == 1) begin
        PC <= commands[PC][n/2-1:0] - 1;
        end
        OUT1 <= C_B;
        OUT2 <= commands[PC][n/2-1:0];
    end
    else if(commands[PC][n-1:n/2] == 4'b1011)
    begin
        PC <= commands[PC][n/2-1:0] - 1;
        OUT1 <= Acc;
        OUT2 <= registers[commands[PC][n/2-1:0]];
    end
    PC <= PC+1;
end
else begin
OUT1 <= 0; // Reset outputs
OUT2 <= 0;
Acc <= 8'bX; // Reset accumulator
C_B <= 0;
```

```
        end
        end
    end
endmodule
```

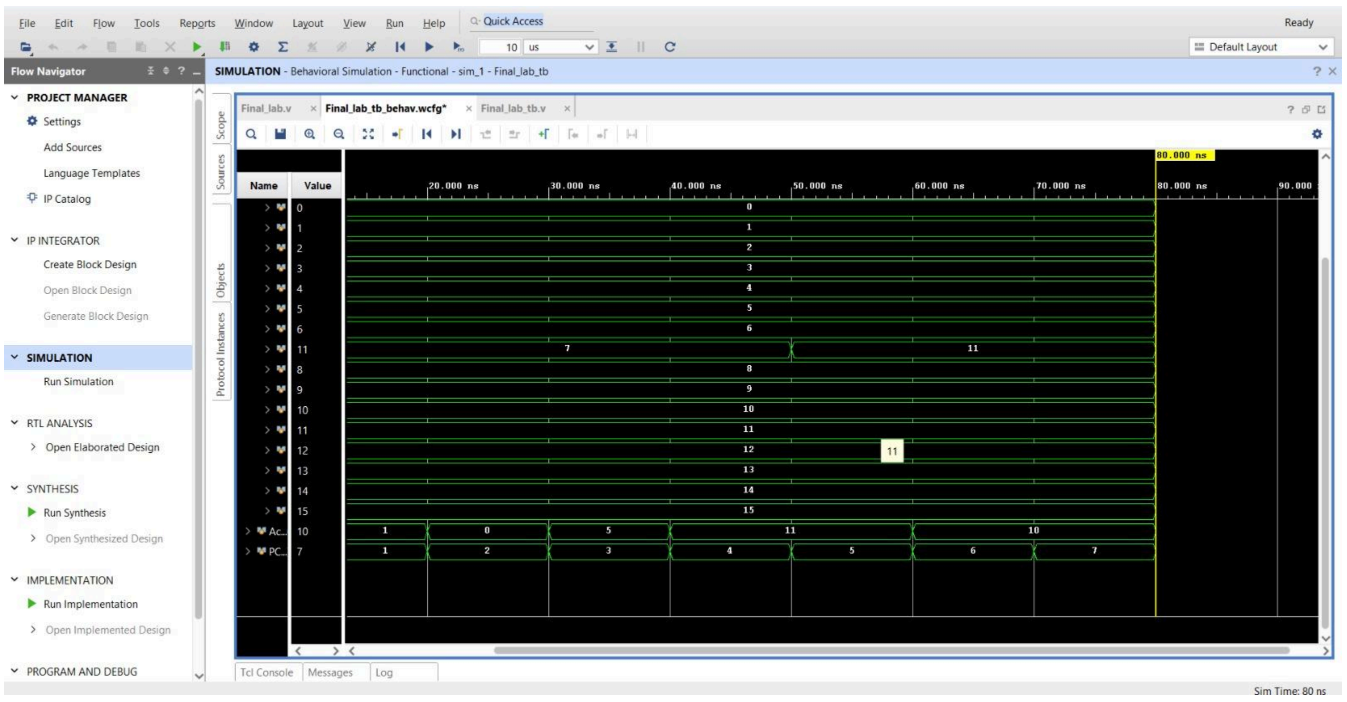
2. Test bench

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 20.04.2024 15:54:31
// Design Name:
// Module Name: Final_lab_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Final_lab_tb();
    reg clk,en,reset;
    wire [7:0] OUT1, OUT2;
    main_module uut(.clk(clk), .en(en), .reset(reset), .OUT1(OUT1), .OUT2(OUT2));
    initial
    begin
    clk = 1;
    forever #5 clk = ~clk;
    end
    initial
    begin
    en = 1; reset = 0;
    #10;
    en = 1; reset = 1;
    #70;
    $finish();
    end

endmodule
```

3. Simulation

The following instructions were executed:
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

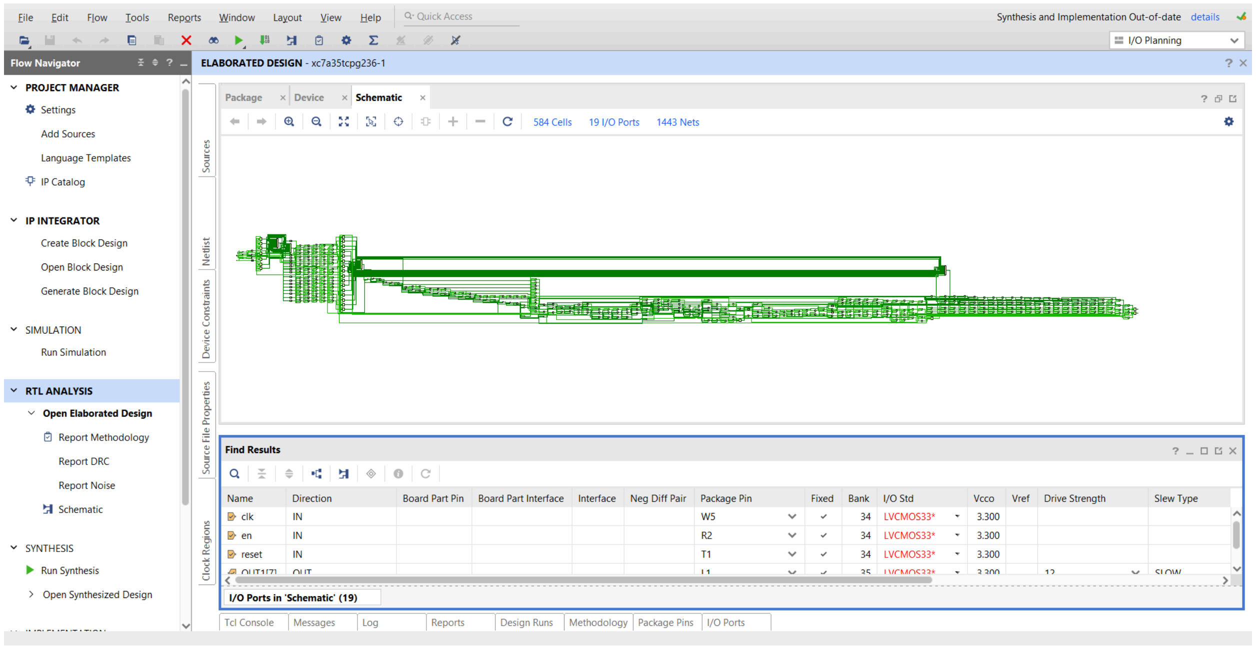
The implementation video can be found here: FPGA Videos



4. XDC

```
set_property IOSTANDARD LVCMOS33 [get_ports {OUT1[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {OUT1[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {OUT1[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {OUT1[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {OUT1[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {OUT1[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {OUT1[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {OUT1[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {OUT2[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {OUT2[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {OUT2[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {OUT2[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {OUT2[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {OUT2[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {OUT2[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {OUT2[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports en]
set_property IOSTANDARD LVCMOS33 [get_ports reset]
set_property PACKAGE_PIN W5 [get_ports clk]
set_property PACKAGE_PIN R2 [get_ports en]
set_property PACKAGE_PIN T1 [get_ports reset]
set_property PACKAGE_PIN L1 [get_ports {OUT1[7]}]
set_property PACKAGE_PIN P1 [get_ports {OUT1[6]}]
set_property PACKAGE_PIN N3 [get_ports {OUT1[5]}]
set_property PACKAGE_PIN P3 [get_ports {OUT1[4]}]
set_property PACKAGE_PIN U3 [get_ports {OUT1[3]}]
set_property PACKAGE_PIN W3 [get_ports {OUT1[2]}]
set_property PACKAGE_PIN V3 [get_ports {OUT1[1]}]
set_property PACKAGE_PIN V13 [get_ports {OUT1[0]}]
set_property PACKAGE_PIN V14 [get_ports {OUT2[7]}]
set_property PACKAGE_PIN U14 [get_ports {OUT2[6]}]
set_property PACKAGE_PIN U15 [get_ports {OUT2[5]}]
set_property PACKAGE_PIN W18 [get_ports {OUT2[4]}]
set_property PACKAGE_PIN V19 [get_ports {OUT2[3]}]
set_property PACKAGE_PIN U19 [get_ports {OUT2[2]}]
set_property PACKAGE_PIN E19 [get_ports {OUT2[1]}]
set_property PACKAGE_PIN U16 [get_ports {OUT2[0]}]
```

5. Block design



ELABORATED DESIGN - xc7a35tcpg236-1

Find Results

Name	Direction	Board Part Pin	Board Part Interface	Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type
clk	IN					W5	✓	34	LVCMOS33*	3.300			
en	IN					R2	✓	34	LVCMOS33*	3.300			
reset	IN					T1	✓	34	LVCMOS33*	3.300			
OUT1[7]	OUT					L1	✓	35	LVCMOS33*	3.300		12	SLOW

I/O Ports in 'Schematic' (19)