भारतीय प्रौद्योगिकी संस्थान गांधीनगर
पालज, गांधीनगर, गुजरात 382 055
INDIAN INSTITUTE OF TECHNOLOGY GANDHINAGAR
PALAJ, GANDHINAGAR, GUJARAT 382 055

Name : Bhavik Patel
Roll no. : 22110047

IITGN

Digital Systems

Mini-Processor Design and Implementation on FPGA that runs a single Program

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

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

1. PROGRAM MEMORY is a 16x8 array to represent at max 16 instructions of 8 bits each

2. The Program Counter PC is a 4 bit register to store the address of the current instruction being executed

3. The Instruction Register IR is an 8 bit register to store the current instruction being executed

4. The Register File is a 16x8 array to store the values of 16 registers of 8 bits each

5. At each clock cycle, the instruction pointed by the PC is fetched from the PROGRAM MEMORY and stored in IR and exexuted and the PC is incremented by 1

6. All ALU operations are performed on the ACC register

7. RSTN is the reset signal. When RSTN is low, the processor is reset

8. PAUSE is a control signal to pause the processor

Instruction format:

Direct instruction

Operation code	Register address

Branch Instruction

Operation code	4-bit address (label)

Instructions

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)

CODE

```
`timescale 1ns / 1ps

module Processor(
    clk, rstn, pause,mode,
    CB,
    Output
    ,slow_clk
    ,PC
);
output wire slow_clk;
input [4:0] mode;
input clk, rstn, pause;
reg [7:0] PROGRAM [15:0];
reg [7:0] RegFile [15:0];
reg [7:0] ACC;
reg [7:0] EXT;
output reg [7:0] Output;
//reg [7:0] Register;
output reg CB;
reg [8:0] SUMDIFF;
reg [15:0] MULTDIV;
wire [7:0] Div;
reg [7:0] IR;
output reg [3:0] PC;
wire [7:0] Rem;
integer i;
//ClockDevide inst1(.clk(clk), .slow_clk(slow_clk));
division inst2(
    .A(ACC),
    .B({4'b0, RegFile[IR[3:0]]}),
    .Res(Div),
    .Rem(Rem)
```

```
division inst2(
    .A(ACC),
    .B({4'b0, RegFile[IR[3:0]]}),
    .Res(Div),
    .Rem(Rem)
  );
assign slow_clk = clk;
/*

1. PROGRAM MEMORY is a 16x8 array to represent
at max 16 instructions of 8 bits each

2. The Program Counter PC is a 4 bit register to store the
address of the current instruction being executed

3. The Instruction Register IR is an 8 bit register to store
the current instruction being executed

4. The Register File is a 16x8 array to store the values of
16 registers of 8 bits each

5. At each clock cycle, the instruction pointed by the PC is
fetched from the PROGRAM MEMORY and stored in IR and exexuted and
the PC is incremented by 1

6. All ALU operations are performed on the ACC register

7. RSTN is the reset signal. When RSTN is low, the processor is reset

8. PAUSE is a control signal to pause the processor
```

```
9. The instructions are of the format:
    1.  0000 0000 : NOP

    2.  0001 xxxx : ADD Ri
    3.  0010 xxxx : SUB Ri
    4.  0011 xxxx : MUL Ri
    5.  0100 xxxx : DIV Ri

    6.  0000 0001 : LSL ACC (Logical Shift Left the contents of ACC. Does not update CB)
    7.  0000 0010 : LSR ACC (Logical Shift Right the contents of ACC. Does not update CB)
    8.  0000 0011 : CIR ACC (Circular Shift Right the contents of ACC. Does not update CB)
    9.  0000 0100 : CIL ACC (Circular Shift Left the contents of ACC. Does not update CB)
    10. 0000 0101 : ASR ACC (Arithmetic Shift Right the contents of ACC. Does not update CB)

    11. 0101 xxxx : AND Ri
    12. 0110 xxxx : XOR Ri
    13. 0111 xxxx : CMP Ri (Compare ACC with Ri. If ACC >= Ri, CB = 0, else CB = 1)

    14. 0000 0110 : INC ACC (Increment ACC by 1. Updates CB if overflow)
    15. 0000 0111 : DEC ACC (Decrement ACC by 1. Updates CB if underflow)

    16. 1000 xxxx : BR <4-bit address> (PC is updated and the program branches to 4-bit addres
    17. 1001 xxxx : MOV Ri (Move the contents of Ri to ACC)
    18. 1010 xxxx : MOV ACC Ri (Move the contents of ACC to Ri)
    19. 1011 xxxx : RET <4-bit address> (PC is updated and program returns to the called progr

    20. 1111 1111 : HLT (Halt the program)
*/
```

```
always @(posedge slow_clk or negedge rstn) begin
    if(!rstn)begin

        RegFile[0] <= 8'd0;
        RegFile[1] <= 8'd1;
        RegFile[2] <= 8'd2;
        RegFile[3] <= 8'd3;
        RegFile[4] <= 8'd4;
        RegFile[5] <= 8'd5;
        RegFile[6] <= 8'd6;
        RegFile[7] <= 8'd7;
        RegFile[8] <= 8'd8;
        RegFile[9] <= 8'd9;
        RegFile[10] <= 8'd10;
        RegFile[11] <= 8'd11;
        RegFile[12] <= 8'd12;
        RegFile[13] <= 8'd13;
        RegFile[14] <= 8'd14;
        RegFile[15] <= 8'd15;


        PROGRAM[0] <= 8'b10010011;
        PROGRAM[1] <= 8'b01100011;
        PROGRAM[2] <= 8'b00010101;
        PROGRAM[3] <= 8'b00010110;
        PROGRAM[4] <= 8'b10100111;
        PROGRAM[5] <= 8'b00000110;
        PROGRAM[6] <= 8'b00000111;
        PROGRAM[7] <= 8'b00101010;
        PROGRAM[8] <= 8'b00110100;
        PROGRAM[9] <= 8'b01100101;
```

```
        PROGRAM[9] <= 8'b01100101;
        PROGRAM[10] <= 8'b00110100;
        PROGRAM[11] <= 8'b01000100;
        PROGRAM[12] <= 8'b01110101;
        PROGRAM[13] <= 8'b10000101;
        PROGRAM[14] <= 8'b11111111;


        PC <= 4'b0;
        IR <= 8'b0;
        ACC <= 8'b11111111;
        EXT <= 8'b0;
        CB <= 1'b0;
        SUMDIFF <= 9'b0;
        MULTDIV <= 16'b0;
        Output <= 8'b0;
    end

    else begin
        if(mode == 5'b00001)begin
            Output <= RegFile[0];
        end
        else if(mode == 5'b11111)begin
            Output <= ACC;
        end
        else if(mode == 5'b00010)begin
            Output <= RegFile[1];
        end
        else if(mode == 5'b00011)begin
            Output <= RegFile[2];
        end
```

```
end
else if(mode == 5'b00101)begin
    Output <= RegFile[4];
end
else if(mode == 5'b00110)begin
    Output <= RegFile[5];
end
else if(mode == 5'b00111)begin
    Output <= RegFile[6];
end
else if(mode == 5'b01000)begin
    Output <= RegFile[7];
end
else if(mode == 5'b01001)begin
    Output <= RegFile[8];
end
else if(mode == 5'b01010)begin
    Output <= RegFile[9];
end
else if(mode == 5'b01011)begin
    Output = RegFile[10];
end
else if(mode == 5'b01100)begin
    Output = RegFile[11];
end
else if(mode == 5'b01101)begin
    Output = RegFile[12];
end
else if(mode == 5'b01110)begin
    Output = RegFile[13];
end
```

```
else if(mode == 5'b01111)begin
    Output = RegFile[14];
end
else if(mode == 5'b10000)begin
    Output = RegFile[15];
end
else if(mode == 5'b10001)begin
    Output <= IR;
end
else if(mode == 5'b10010)begin
    Output <= {3'b0,PC};
end
else if(mode == 5'b10011)begin
    Output <= EXT;
end
if (!pause) begin
    IR <= PROGRAM[PC];
    PC <= PC + 1;
end

else begin
    IR <= IR;
    PC <= PC;
end
```

```
        case(IR[7:4] )
            4'b0000:begin
                if (IR[3:0] == 4'b0000)begin
                    //NOP
                end
                else if (IR[3:0] == 4'b0001) begin
                    // LSL ACC
                    ACC <= ACC << 1;
                end
                else if (IR[3:0] == 4'b0010) begin
                    // LSR ACC
                    ACC <= ACC >> 1;
                end
                else if (IR[3:0] == 4'b0011) begin
                    // CIR ACC
                    ACC <= {ACC[0], ACC[7:1]};
                end
                else if (IR[3:0] == 4'b0100) begin
                    // CIL ACC
                    ACC <= {ACC[6:0], ACC[7]};
                end
                else if (IR[3:0] == 4'b0101) begin
                    // ASR ACC
                    ACC <= {ACC[7], ACC[7:1]};
                end
                else if (IR[3:0] == 4'b0110) begin
                    // INC ACC
                    SUMDIFF = ACC + 1;
                    CB = SUMDIFF[8];
                    ACC = SUMDIFF[7:0];
                end
```

```
        end
        else if (IR[3:0] == 4'b0111) begin
            // DEC ACC
            SUMDIFF = ACC - 1;
            CB = SUMDIFF[8];
            ACC = SUMDIFF[7:0];
        end
    end

    4'b0001:begin
        //ADD Ri
        SUMDIFF <= ACC + RegFile[IR[3:0]];
        CB <= SUMDIFF[8];
        ACC <= SUMDIFF[7:0];
    end
    4'b0010:begin
        //SUB Ri
        SUMDIFF = ACC - RegFile[IR[3:0]];
        CB = SUMDIFF[8];
        ACC = SUMDIFF[7:0];
    end
    4'b0011:begin
        //MUL Ri
        MULTDIV = ACC * RegFile[IR[3:0]];
        ACC = MULTDIV[7:0];
        EXT = MULTDIV[15:8];
    end
    4'b0100:begin
        //DIV Ri
        // Need a synthesizable way to implement division
        ACC = Div;
```

```
                ACC = Div;
                EXT = Rem;
            end
            4'b0101:begin
                //AND Ri
                ACC <= ACC & RegFile[IR[3:0]];
            end
            4'b0110:begin
                //XOR Ri
                ACC <= ACC ^ RegFile[IR[3:0]];
            end
            4'b0111:begin
                //CMP Ri
                SUMDIFF = ACC - RegFile[IR[3:0]];
                CB = SUMDIFF[8];
            end
            4'b1000:begin
                //BR <4-bit address>
                if(CB == 1)begin
                    PC <= IR[3:0];
                end
            end
            4'b1001:begin
                //MOV Ri
                ACC <= RegFile[IR[3:0]];
            end
            4'b1010:begin
                //MOV ACC Ri
                RegFile[IR[3:0]] <= ACC;
            end
                end
                4'b1010:begin
                    //MOV ACC Ri
                    RegFile[IR[3:0]] <= ACC;
                end
                4'b1011:begin
                    //RET <4-bit address>
                    PC <= IR[3:0];
                end
                4'b1111:begin
                    //HLT
                    $finish;
                end
            endcase
        end
    end
    //assign Output = Register;
    endmodule
```

Division code

```
module division(A, B, Res, Rem);
  parameter WIDTH = 8;
  input [WIDTH-1:0] A;
  input [WIDTH-1:0] B;
  output [WIDTH-1:0] Res;
  output reg [WIDTH-1:0] Rem;
  reg [WIDTH-1:0] Res;
  reg [WIDTH-1:0] a1, b1;
  reg [WIDTH:0] p1;
  integer i;

  always @(A or B)
    begin
      a1 = A;
      b1 = B;
      p1 = 0;

      for (i = 0; i < WIDTH; i = i + 1) begin
        p1 = {p1[WIDTH-2:0], a1[WIDTH-1]};
        a1[WIDTH-1:1] = a1[WIDTH-2:0];
        p1 = p1 - b1;

        if (p1[WIDTH-1] == 1'b1) begin
          a1[0] = 0;
          p1 = p1 + b1;
        end else begin
          a1[0] = 1;
        end
      end
      Res = a1;
      Rem = p1;
    end
```

Slow Clock(Clock Divider)

```
module ClockDivide(
input clk,
output slow_clk
    );

reg [31:0] counter = 0;

always@(posedge clk)
begin
    counter <= counter + 1;
end
assign slow_clk = counter[27];
endmodule
```

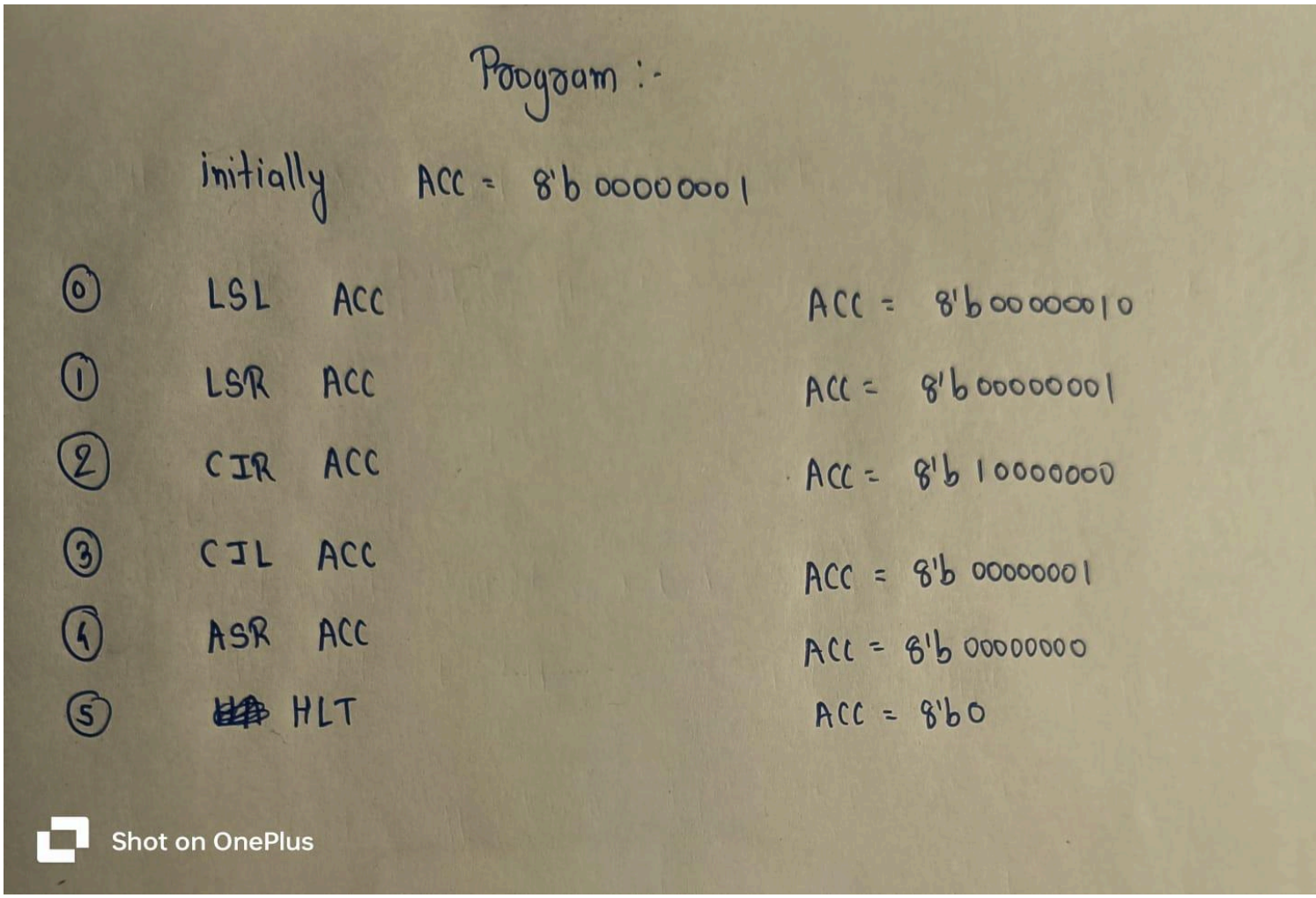
Test Bench

```
`timescale 1ns / 1ps
module Processor_tb;
reg clk, rstn, pause;
reg [4:0] mode;
wire CB;
wire [7:0] Output;
wire [3:0] PC;
wire slow_clk;
Processor dut(
    .clk(clk),
    .rstn(rstn),
    .pause(pause),
    .mode(mode),
    .CB(CB),
    .Output(Output),
    .slow_clk(slow_clk),
    .PC(PC)
);
initial begin
    clk = 0;
    forever #5 clk = ~clk;
end
initial begin
    rstn = 0;
    pause = 0;
    mode = 5'b11111;
    #15 rstn = 1;
    #500 $finish;
end
endmodule
```

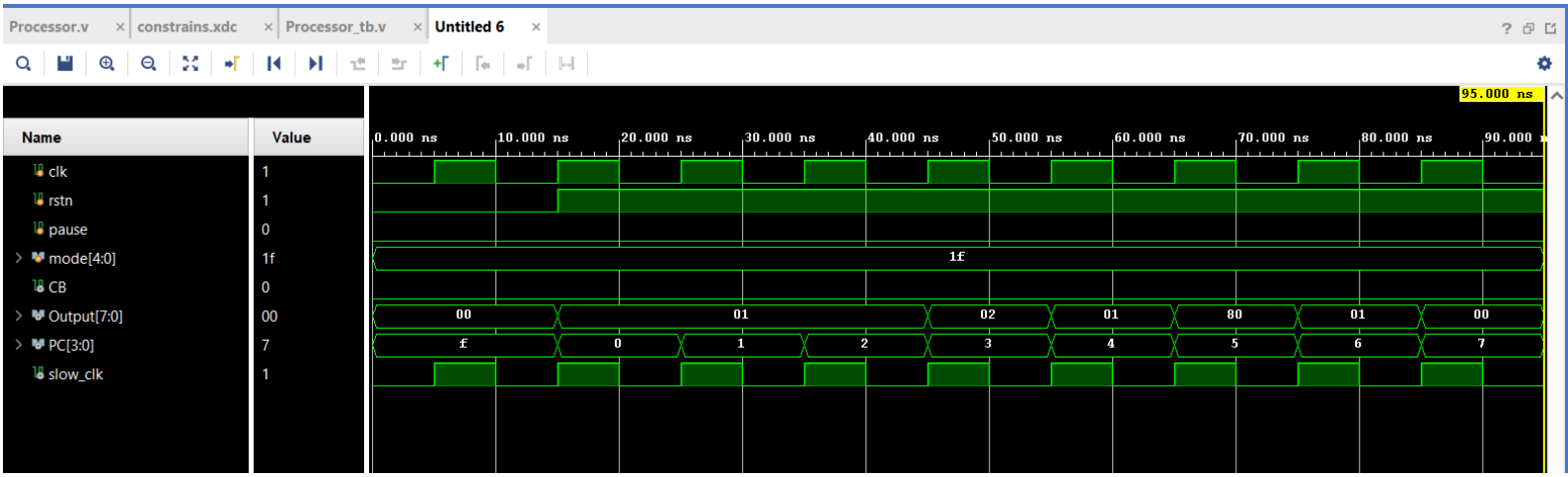
Constrain file

```
set_property IOSTANDARD LVCMOS33 [get_ports {mode[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {mode[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {mode[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {mode[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {mode[0]}]
set_property PACKAGE_PIN W17 [get_ports {mode[3]}]
set_property PACKAGE_PIN W16 [get_ports {mode[2]}]
set_property PACKAGE_PIN V16 [get_ports {mode[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Output[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Output[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Output[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Output[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Output[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Output[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Output[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Output[0]}]
set_property PACKAGE_PIN V19 [get_ports {Output[3]}]
set_property PACKAGE_PIN U19 [get_ports {Output[2]}]
set_property PACKAGE_PIN E19 [get_ports {Output[1]}]
set_property PACKAGE_PIN U16 [get_ports {Output[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {PC[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {PC[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {PC[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {PC[0]}]
set_property PACKAGE_PIN P3 [get_ports {PC[3]}]
set_property PACKAGE_PIN W3 [get_ports {PC[1]}]
set_property PACKAGE_PIN V3 [get_ports {PC[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports CB]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports pause]
set_property IOSTANDARD LVCMOS33 [get_ports rstn]
```

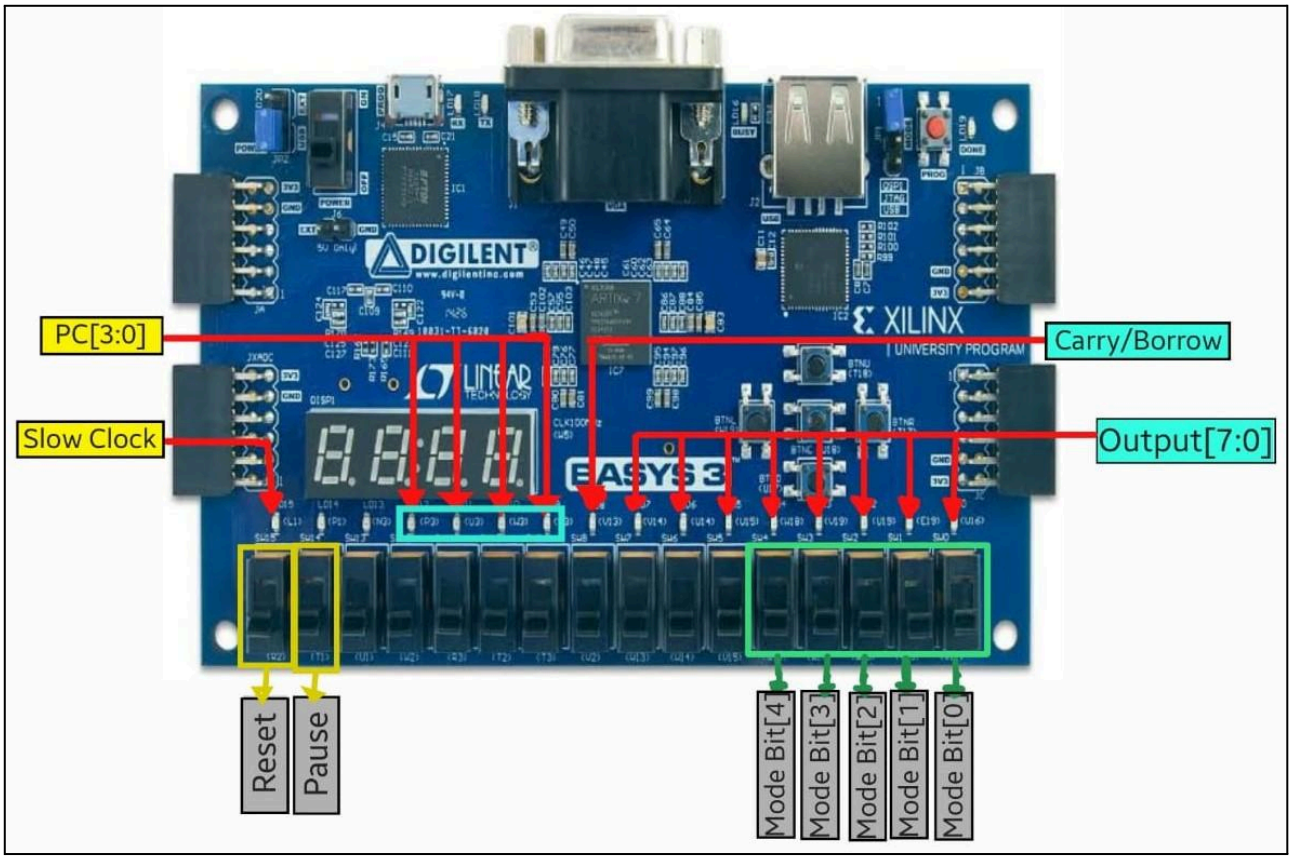
```
set_property IOSTANDARD LVCMOS33 [get_ports CB]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports pause]
set_property IOSTANDARD LVCMOS33 [get_ports rstn]
set_property PACKAGE_PIN W5 [get_ports clk]
set_property PACKAGE_PIN R2 [get_ports rstn]
set_property PACKAGE_PIN T1 [get_ports pause]

set_property PACKAGE_PIN N3 [get_ports CB]

set_property PACKAGE_PIN W15 [get_ports {mode[4]}]
set_property PACKAGE_PIN V17 [get_ports {mode[0]}]
set_property PACKAGE_PIN W18 [get_ports {Output[4]}]
set_property PACKAGE_PIN U15 [get_ports {Output[5]}]
set_property PACKAGE_PIN U14 [get_ports {Output[6]}]
set_property PACKAGE_PIN V14 [get_ports {Output[7]}]
set_property PACKAGE_PIN U3 [get_ports {PC[2]}]

set_property PACKAGE_PIN L1 [get_ports slow_clk]
set_property IOSTANDARD LVCMOS33 [get_ports slow_clk]
```



Simulation



FPGA Implementation Diagram



Video:-

We have successfully implemented the processor on FPGA. You can see a demo of implementation of Processor on FPGA by clicking on this **link**