ES204 Digital Systems
Lab Exam

Indian Institute of Technology, Gandhinagar

- Lovekesh Mahale - 22110131
- Omkar Rajeev Prabhu - 22110175

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

Processor Module Code for FPGA Implementation-

```
`timescale 1ns / 1ps

module Clock_Divider(Clk, Slow_Clk);
input Clk ;
output Slow_Clk ;
reg [31:0] Counter;
always @(posedge Clk)
begin
Counter <= Counter + 1;
end
assign Slow_Clk = Counter[27];
endmodule


//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
module Processor(Clk ,  Out );
input Clk ;
output reg [15:0]Out ;
reg [7:0]Inst ;
reg [7:0]Acc ;
reg [7:0]Inst_Mem[19:0] ;
reg [7:0]PC ;
reg C_B ;
reg [7:0]Ext ;
```

```
reg [7:0]Register[15:0] ;
reg [3:0]Oper ;
reg [3:0]Address ;
reg [7:0] Dividend , Divisor,  Quotient , Remainder ;
reg [15:0] Conc ;
wire Slow_Clk ;

integer i,k;

initial
begin
PC = 8'b00000000;
Ext = 8'b00000000;
Acc = 8'b01000100 ;
C_B = 0 ;

Register[0] = 8'b00001010; // Decimal 10
Register[1] = 8'b00010100; // Decimal 20
Register[2] = 8'b00011110; // Decimal 30
Register[3] = 8'b00101000; // Decimal 40
Register[4] = 8'b00110010; // Decimal 50
Register[5] = 8'b00111100; // Decimal 60
Register[6] = 8'b01000110; // Decimal 70
Register[7] = 8'b01010000; // Decimal 80
Register[8] = 8'b01011010; // Decimal 90
Register[9] = 8'b01100100; // Decimal 100
Register[10] = 8'b01101110; // Decimal 110
Register[11] = 8'b01111000; // Decimal 120
Register[12] = 8'b10000010; // Decimal 130
Register[13] = 8'b10001100; // Decimal 140
Register[14] = 8'b10010110; // Decimal 150
Register[15] = 8'b10100000; // Decimal 160

Inst_Mem[0] =  8'b00000000;
Inst_Mem[1] =  8'b00000001;
Inst_Mem[2] =  8'b00000010;
Inst_Mem[3] =  8'b00000011;
Inst_Mem[4] =  8'b00000100;
Inst_Mem[5] =  8'b00000101;
Inst_Mem[6] =  8'b00000110;
Inst_Mem[7] =  8'b00000111;
Inst_Mem[8] =  8'b00010010;
Inst_Mem[9] =  8'b00100011;
Inst_Mem[10] = 8'b00110101;
```

```
Inst_Mem[11] = 8'b01000011;
Inst_Mem[12] = 8'b01010100;
Inst_Mem[13] = 8'b01101111;
Inst_Mem[14] = 8'b01110100;
Inst_Mem[15] = 8'b10000001;
Inst_Mem[16] = 8'b10010001;
Inst_Mem[17] = 8'b10100001;
Inst_Mem[18] = 8'b10110001;
Inst_Mem[19] = 8'b11110001;
end

Clock_Divider uut(Clk, Slow_Clk) ;

always@(posedge Slow_Clk)
begin
   Inst = Inst_Mem[PC] ;
   Oper = Inst[7:4] ;
   Address = Inst[3:0] ;
   case(Oper)
      4'b1111 : $stop() ; // HALT
      4'b0000 :
         case(Address)
            4'b0001 : begin Acc <= Acc << 1 ; Out <= Acc ; end // Left Shift Left Logical
            4'b0010 : begin Acc <= Acc >> 1  ; Out <= Acc ; end // Left Shift Right Logical
            4'b0011 : begin Acc <= {Acc[0] , Acc[7:1]} ; Out <= Acc ; end // Circular Right Shift
            4'b0100 : begin Acc <= {Acc[6:0] , Acc[7]} ; Out <= Acc ; end  // Circular Left Shift
            4'b0101 : begin Acc <= {Acc[7] , Acc[7:1]} ; Out <= Acc ; end  // Arithematic Right shift
            4'b0110 : begin {C_B , Acc} <= Acc + 1 ; Out <= {C_B ,Acc} ; end  // Increments Acc ;
            4'b0111 : begin {C_B , Acc} <= Acc - 1 ; Out <= {C_B , Acc} ; end  // Decrements Acc ;
         endcase
      4'b0001 : begin {C_B , Acc} <= Acc + Register[Address] ; Out <= {C_B , Acc} ; end  // Add
Register[Address]
      4'b0010 : begin {C_B , Acc} <= Acc - Register[Address] ; Out <= {C_B , Acc} ; end  // Subtract
Register[Address]
      4'b0011 : begin {Ext , Acc} <= Acc * Register[Address]; Out <= {Ext , Acc} ; end  // Multiply
Register[Address]
      4'b0100 : begin // Divide Register[Address]
                Dividend = Acc;
                Divisor = Register[Address];
                Quotient = 8'b0;
                Remainder = 8'b0;

                for (k = 0; k < 8; k = k + 1)
                begin
```

```
                Remainder = Remainder << 1;
                Remainder[0] = Dividend[7];
                Dividend = {Dividend[6:0], 1'b0};

                if (Remainder >= Divisor)
                begin
                   Remainder = Remainder - Divisor;
                   Quotient[7-k] = 1'b1;
                end
                else
                begin
                   Quotient[7-k] = 1'b0;
                end
              end
              Acc <= Quotient ;
              Ext <= Remainder ;
              Out <= {Remainder, Quotient};
            end
   4'b0101 : begin Acc <= (Acc & Register[Address]); Out <= Acc ; end  // AND
   4'b0110 : begin Acc <= (Acc ^ Register[Address]); Out <= Acc ; end  // XRA
   4'b0111 : if (Acc < Register[Address])  // CMP
              begin
              C_B <= 1;
              Out <= {C_B , Acc} ;
              end
             else
              begin
              C_B <= 0;
              Out <= {C_B , Acc} ;
              end

   4'b1000 : if (C_B == 1) // Branch
              begin
              PC <= Address - 1;
              Out <= PC ;
              end
   4'b1001 : begin Acc <= Register[Address] ; Out <= Acc ; end // MOV ACC , Ri
   4'b1010 : begin Register[Address] <= Acc ; Out <= Register[Address] ; end // MOV Ri , ACC
   4'b1011 : begin PC <= Address - 1; Out <= PC ; end // Return
   default : begin Acc <= Acc; Out <= Acc ; end
endcase

PC = PC + 1 ;
```

end
endmodule

Processor Module XDC Constraints File-

```
set_property PACKAGE_PIN L1 [get_ports {Out[15]}]
set_property PACKAGE_PIN P1 [get_ports {Out[14]}]
set_property PACKAGE_PIN N3 [get_ports {Out[13]}]
set_property PACKAGE_PIN P3 [get_ports {Out[12]}]
set_property PACKAGE_PIN W3 [get_ports {Out[10]}]
set_property PACKAGE_PIN U3 [get_ports {Out[11]}]
set_property PACKAGE_PIN V3 [get_ports {Out[9]}]
set_property PACKAGE_PIN V13 [get_ports {Out[8]}]
set_property PACKAGE_PIN V14 [get_ports {Out[7]}]
set_property PACKAGE_PIN U14 [get_ports {Out[6]}]
set_property PACKAGE_PIN U15 [get_ports {Out[5]}]
set_property PACKAGE_PIN W18 [get_ports {Out[4]}]
set_property PACKAGE_PIN E19 [get_ports {Out[1]}]
set_property PACKAGE_PIN V19 [get_ports {Out[3]}]
set_property PACKAGE_PIN U19 [get_ports {Out[2]}]
set_property PACKAGE_PIN U16 [get_ports {Out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Out[15]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Out[14]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Out[13]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Out[12]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Out[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Out[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Out[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Out[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Out[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Out[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Out[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Out[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Out[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports Clk]
set_property PACKAGE_PIN W5 [get_ports Clk]
```

Processor Module Code for Simulation-

```
`timescale 1ns / 1ps

module Processor_Sim(En , Clk ,  Out );
input En ;
input Clk ;
output reg [15:0]Out ;
reg [7:0]Inst ;
reg [7:0]Acc ;
reg [7:0]Inst_Mem[19:0] ;
reg [7:0]PC ;
reg C_B ;
reg [7:0]Ext ;
reg [7:0]Register[15:0] ;
reg [3:0]Oper ;
reg [3:0]Address ;
reg [7:0] Dividend , Divisor, Quotient , Remainder ;
reg [15:0] Conc ;

integer k;

initial
begin
PC = 8'b00000000;
Ext = 8'b00000000;
Acc = 8'b01000100 ;
Out = {8'b00000000 , Acc} ;
C_B = 0 ;

Register[0] = 8'b00001010; // Decimal 10
Register[1] = 8'b00010100; // Decimal 20
Register[2] = 8'b00011110; // Decimal 30
Register[3] = 8'b00101000; // Decimal 40
Register[4] = 8'b00110010; // Decimal 50
Register[5] = 8'b00111100; // Decimal 60
Register[6] = 8'b01000110; // Decimal 70
Register[7] = 8'b01010000; // Decimal 80
Register[8] = 8'b01011010; // Decimal 90
```

```
Register[9] = 8'b01100100; // Decimal 100
Register[10] = 8'b01101110; // Decimal 110
Register[11] = 8'b01111000; // Decimal 120
Register[12] = 8'b10000010; // Decimal 130
Register[13] = 8'b10001100; // Decimal 140
Register[14] = 8'b10010110; // Decimal 150
Register[15] = 8'b10100000; // Decimal 160

Inst_Mem[0] =  8'b00000000;
Inst_Mem[1] =  8'b00000001;
Inst_Mem[2] =  8'b00000010;
Inst_Mem[3] =  8'b00000011;
Inst_Mem[4] =  8'b00000100;
Inst_Mem[5] =  8'b00000101;
Inst_Mem[6] =  8'b00000110;
Inst_Mem[7] =  8'b00000111;
Inst_Mem[8] =  8'b00010010;
Inst_Mem[9] =  8'b00100011;
Inst_Mem[10] = 8'b00110101;
Inst_Mem[11] = 8'b01000011;
Inst_Mem[12] = 8'b01010100;
Inst_Mem[13] = 8'b01101111;
Inst_Mem[14] = 8'b01110100;
Inst_Mem[15] = 8'b10000001;
Inst_Mem[16] = 8'b10010001;
Inst_Mem[17] = 8'b10100001;
Inst_Mem[18] = 8'b10110001;
Inst_Mem[19] = 8'b11110001;
end

always@(posedge Clk)
begin
   Inst = Inst_Mem[PC] ;
   Oper = Inst[7:4] ;
   Address = Inst[3:0] ;
   case(Oper)
      4'b1111 : $stop() ; // HALT
      4'b0000 :
         case(Address)
            4'b0001 : begin Acc <= Acc << 1 ; Out <= Acc ; end // Left Shift Left Logical
            4'b0010 : begin Acc <= Acc >> 1  ; Out <= Acc ; end // Left Shift Right Logical
            4'b0011 : begin Acc <= {Acc[0] , Acc[7:1]} ; Out <= Acc ; end // Circular Right Shift
            4'b0100 : begin Acc <= {Acc[6:0] , Acc[7]} ; Out <= Acc ; end  // Circular Left Shift
            4'b0101 : begin Acc <= {Acc[7] , Acc[7:1]} ; Out <= Acc ; end  // Arithematic Right shift
```

```
            4'b0110 : begin {C_B , Acc} <= Acc + 1 ; Out <= {C_B ,Acc} ; end  // Increments Acc ;
            4'b0111 : begin {C_B , Acc} <= Acc - 1 ; Out <= {C_B , Acc} ; end  // Decrements Acc ;
          endcase
      4'b0001 : begin {C_B , Acc} <= Acc + Register[Address] ; Out <= {C_B , Acc} ; end  // Add
Register[Address]
      4'b0010 : begin {C_B , Acc} <= Acc - Register[Address] ; Out <= {C_B , Acc} ; end  // Subtract
Register[Address]
      4'b0011 : begin {Ext , Acc} <= Acc * Register[Address]; Out <= {Ext , Acc} ; end  // Multiply
Register[Address]
      4'b0100 : begin // Divide Register[Address]
              Dividend = Acc;
              Divisor = Register[Address];
              Quotient = 8'b0;
              Remainder = 8'b0;

              for (k = 0; k < 8; k = k + 1)
              begin
                 Remainder = Remainder << 1;
                 Remainder[0] = Dividend[7];
                 Dividend = {Dividend[6:0], 1'b0};

                 if (Remainder >= Divisor)
                 begin
                    Remainder = Remainder - Divisor;
                    Quotient[7-k] = 1'b1;
                 end
                 else
                 begin
                    Quotient[7-k] = 1'b0;
                 end
              end
              Acc <= Quotient ;
              Ext <= Remainder ;
              Out <= {Remainder, Quotient};
            end
      4'b0101 : begin Acc <= (Acc & Register[Address]); Out <= Acc ; end  // AND
      4'b0110 : begin Acc <= (Acc ^ Register[Address]); Out <= Acc ; end  // XRA
      4'b0111 : if (Acc < Register[Address])  // CMP
              begin
              C_B <= 1;
              Out <= {C_B , Acc} ;
              end
             else
              begin
```

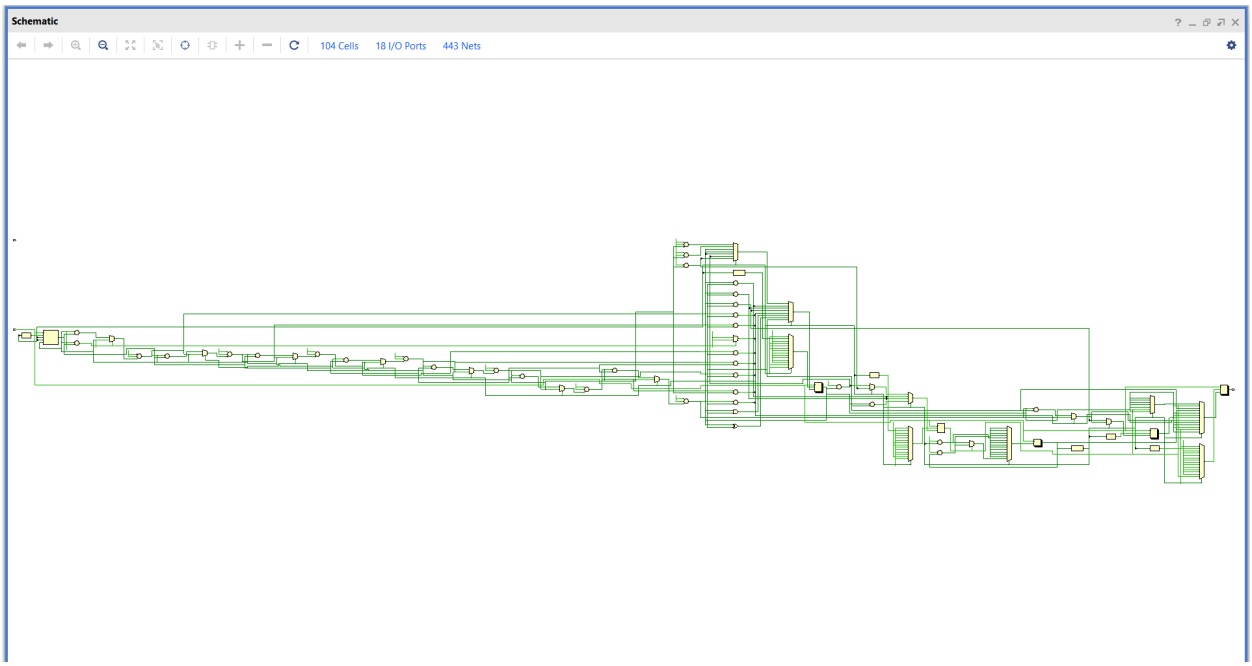
```
            C_B <= 0;
            Out <= {C_B , Acc} ;
            end

      4'b1000 : if (C_B == 1) // Branch
            begin
            PC <= Address - 1;
            Out <= PC ;
            end
      4'b1001 : begin Acc <= Register[Address] ; Out <= Acc ; end // MOV ACC , Ri
      4'b1010 : begin Register[Address] <= Acc ; Out <= Register[Address] ; end // MOV Ri , ACC
      4'b1011 : begin PC <= Address - 1; Out <= PC ; end // Return
      default : begin Acc <= Acc; Out <= Acc ; end
   endcase

   PC = PC + 1 ;

end
endmodule
```

Processor Module Testbench for Simulation-

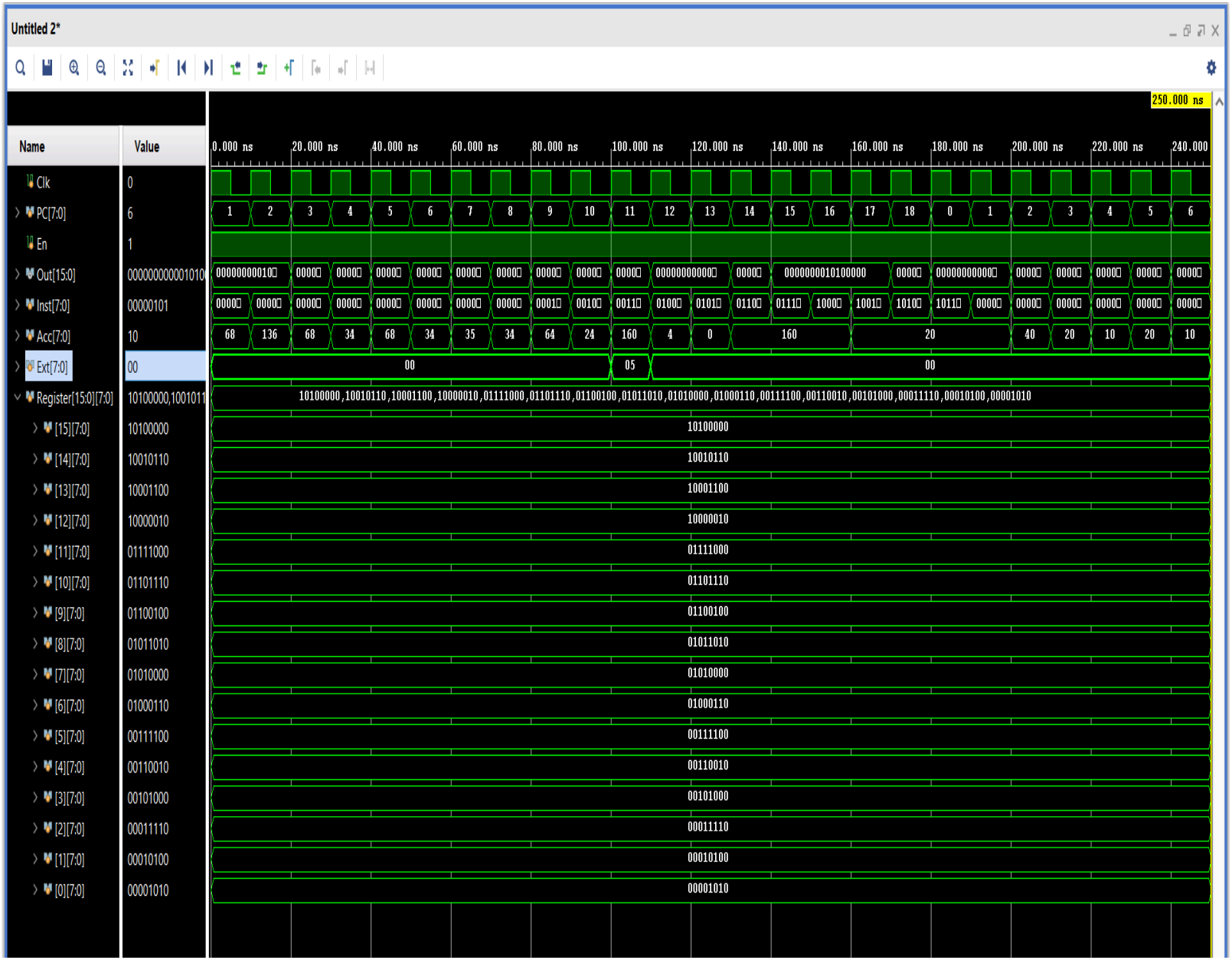
```
`timescale 1ns / 1ps

module Processor_Sim_tb();
reg Clk;
reg En;
wire [15:0] Out ;

Processor_Sim uut (En , Clk, Out );

initial
begin
   Clk = 1;
   forever #5 Clk = ~Clk;
end

initial
begin

En = 1 ;
#150;
```

```
$finish() ;
end
endmodule
```

Schematic -



Simulation Results -


Untitled 2*
250.000 ns
Name
Value
0.000 ns
20.000 ns
40.000 ns
60.000 ns
80.000 ns
100.000 ns
120.000 ns
140.000 ns
160.000 ns
180.000 ns
200.000 ns
220.000 ns
240.000
Clk
0
PC[7:0]
6
En
1
Out[15:0]
0000000000001010
Inst[7:0]
00000101
Acc[7:0]
10
Ext[7:0]
00
Register[15:0][7:0]
10100000,10010110,10001100,10000010,01111000,01101110,01100100,01011010,01010000,01000110,00111100,00110010,00101000,00011110,00010100,00001010
[15][7:0]
10100000
[14][7:0]
10010110
[13][7:0]
10001100
[12][7:0]
10000010
[11][7:0]
01111000
[10][7:0]
01101110
[9][7:0]
01100100
[8][7:0]
01011010
[7][7:0]
01010000
[6][7:0]
01000110
[5][7:0]
00111100
[4][7:0]
00110010
[3][7:0]
00101000
[2][7:0]
00011110
[1][7:0]
00010100
[0][7:0]
00001010

Drive Link for the Video of FPGA implementation -

https://drive.google.com/file/d/1zhTtOO0WKj3HNYMdQhGerITKS28QCut_/view?usp=sharing

---