Laboratory 5:

AN ENHANCED PROCESSOR

OBJECTIVES

- The purpose of this lab is to learn how to connect simple input (switches) and output devices (LEDs and 7-segment) to an FPGA chip and implement a circuit that uses these devices.
- Design an enhanced processor based on the simple processor in lab 4.

PREPARATION FOR LAB 5

- Finish Pre Lab 5 at home.
- Students have to simulate all the exercises in Pre Lab 5 at home. All results (codes, waveform, RTL viewer, …) have to be captured and submitted to instructors prior to the lab session. *If not, students will not participate in the lab and be considered absent this session.*

REFERENCE

1. Intel FPGA training

Laboratory 5:

AN ENHANCED PROCESSOR

EXPERIMENT 1

Objective: Design and implement a simple processor.

Requirement: Design and implement the processor which is shown in Figure 1. Then connect the enhanced processors to a memory and output register (Figure 2).

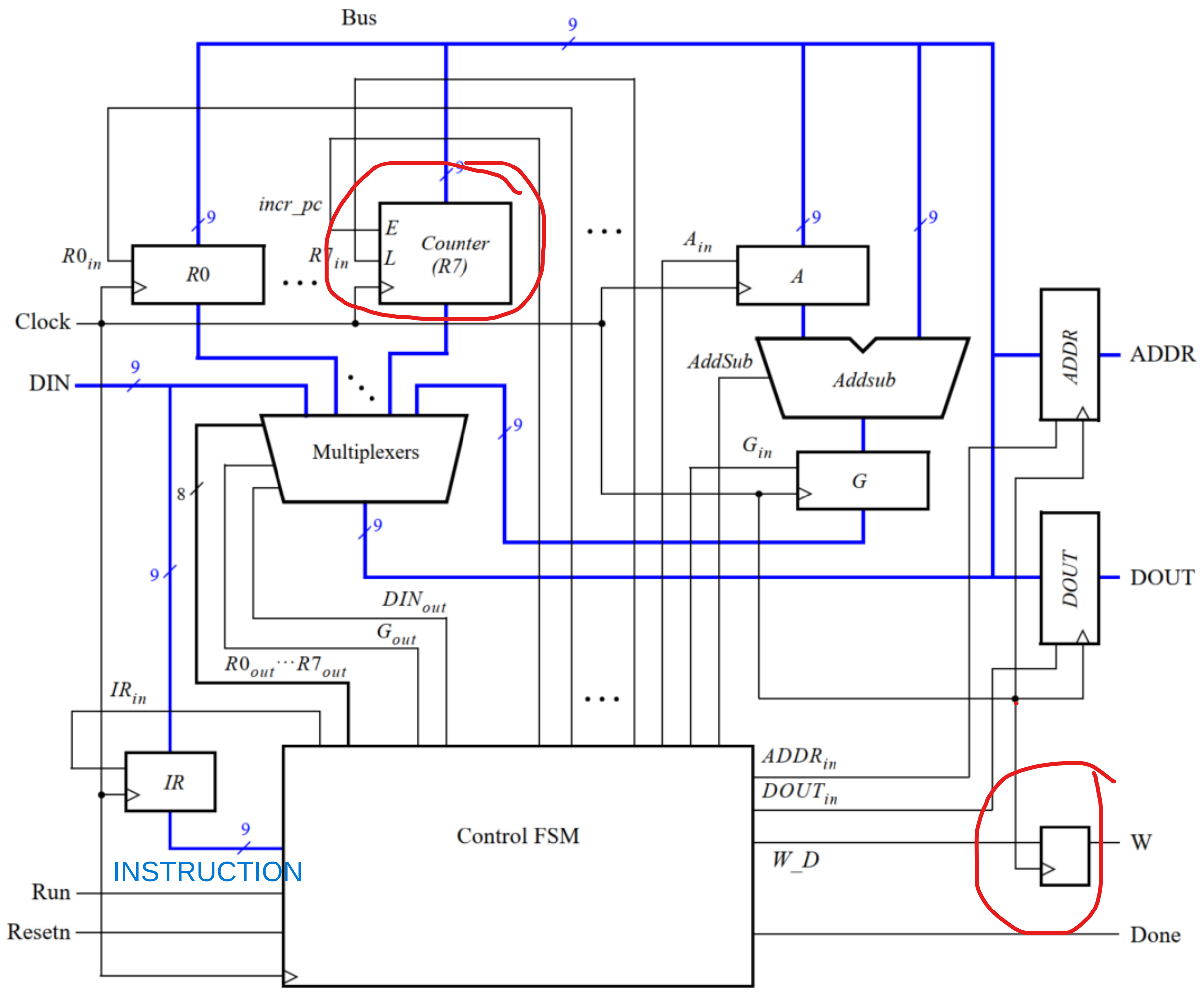


Figure 1: A reference schematic of the enhanced processor.

Laboratory 5:

AN ENHANCED PROCESSOR

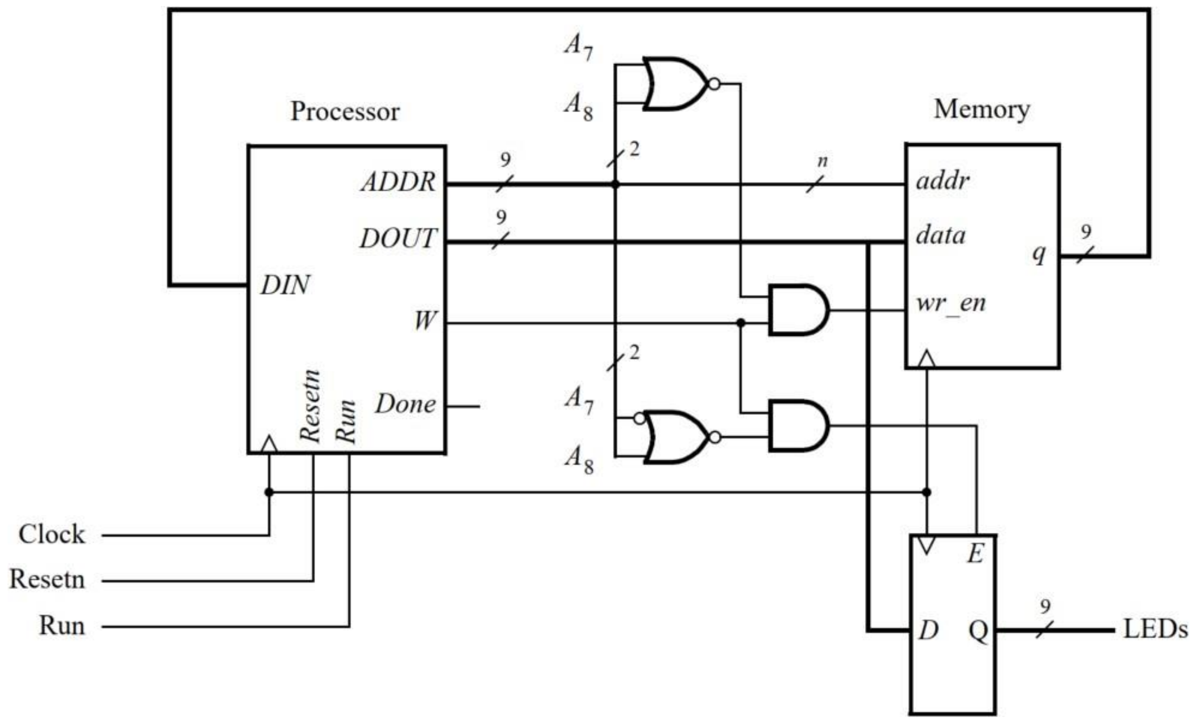
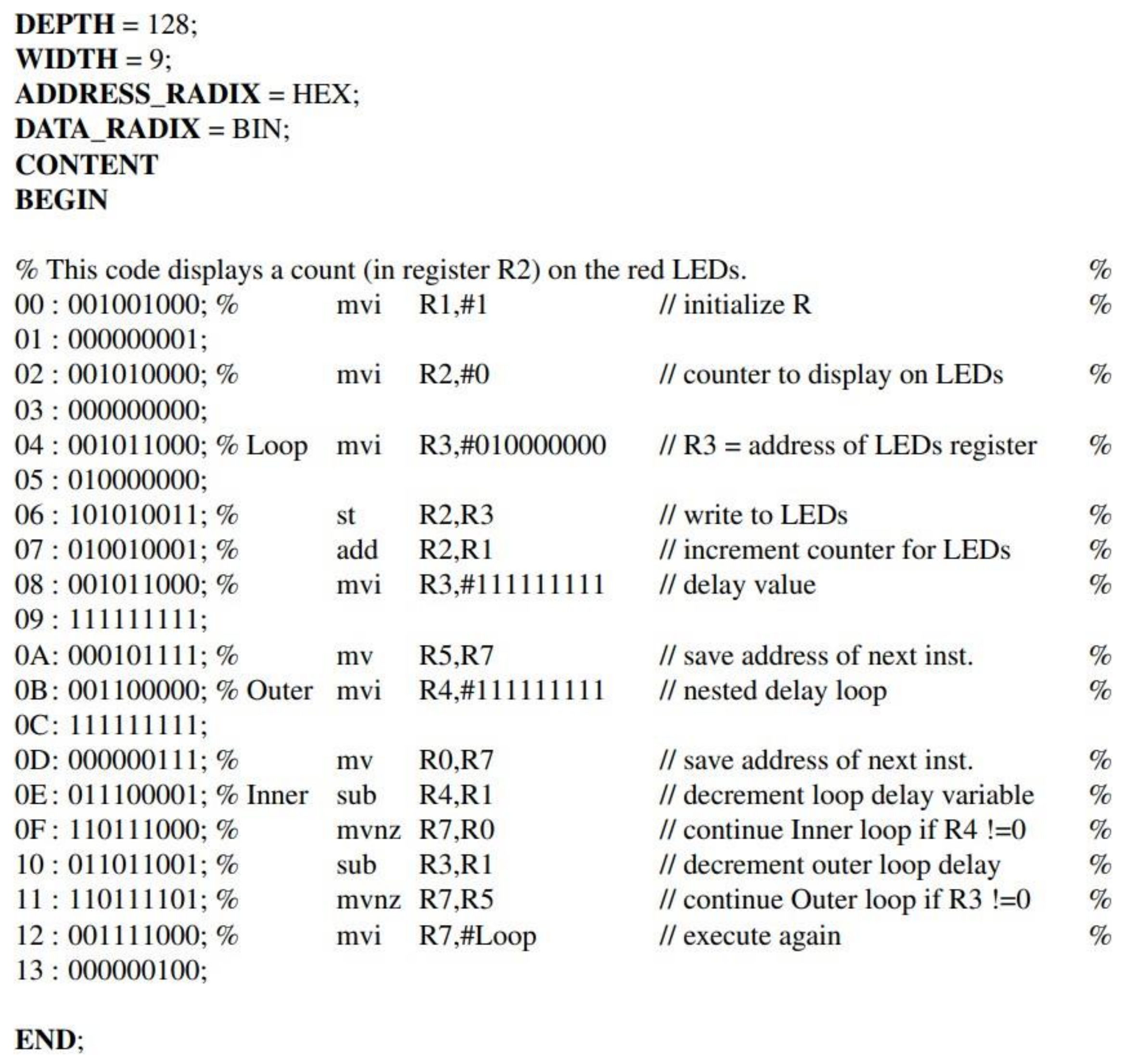


Figure 2: Connecting the enhanced processor to a memory and output register.

Instruction:

- Create a new Quartus project for the enhanced version of the processor.
- Write System Verilog code for the processor and test your circuit by using functional simulation: apply instructions to the DIN port and observe the internal processor signals as the instructions are executed. Pay careful attention to the timing of signals between your processor and external memory; account for the fact that the memory has registered input ports, as we discussed for Figure 2.
- Create another Quartus project that instantiates the processor, memory module, and register shown in Figure 2. Use the Quartus IP Catalog to create the RAM: 1-PORT memory module. Follow the instructions provided by the wizard to create a memory that has one 9-bit wide read/write data port and is 128 words deep. Ensure that the output is not registered. Use a MIF file to store instructions in the memory that are to be executed by your processor. An example program in the form of a MIF file is shown in Figure 3. This program display an 8-bit counter value on the

LEDs output port. Loops are used in the program to create delays so that the counter values are not changed too quickly to be observed. Comments are included in the MIF file in Figure 3 to describe the program’s code.



```
DEPTH = 128;
WIDTH = 9;
ADDRESS_RADIX = HEX;
DATA_RADIX = BIN;
CONTENT
BEGIN

% This code displays a count (in register R2) on the red LEDs.                              %
00 : 001001000; %         mvi    R1,#1            // initialize R                           %
01 : 000000001;
02 : 001010000; %         mvi    R2,#0            // counter to display on LEDs             %
03 : 000000000;
04 : 001011000; % Loop    mvi    R3,#010000000    // R3 = address of LEDs register          %
05 : 010000000;
06 : 101010011; %         st     R2,R3            // write to LEDs                          %
07 : 010010001; %         add    R2,R1            // increment counter for LEDs             %
08 : 001011000; %         mvi    R3,#111111111    // delay value                            %
09 : 111111111;
0A: 000101111; %          mv     R5,R7            // save address of next inst.             %
0B: 001100000; % Outer    mvi    R4,#111111111    // nested delay loop                      %
0C: 111111111;
0D: 000000111; %          mv     R0,R7            // save address of next inst.             %
0E: 011100001; % Inner    sub    R4,R1            // decrement loop delay variable          %
0F : 110111000; %         mvnz   R7,R0            // continue Inner loop if R4 !=0          %
10 : 011011001; %         sub    R3,R1            // decrement outer loop delay             %
11 : 110111101; %         mvnz   R7,R5            // continue Outer loop if R3 !=0          %
12 : 001111000; %         mvi    R7,#Loop         // execute again                          %
13 : 000000100;

END;
```

➢ *Figure 3:* An example program in a memory initialization file (MIF).

➢ Use functional simulation to test the circuit. Ensure that data is read properly from the memory and executed by the processor

➢ Include in your project the necessary pin assignments to implement your circuit on your DE-series board. Use switch SW9 to drive the processor’s Run input, use KEY0 for Resetn, and use the board’s 50 MHz clock signal as the Clock input. Since the circuit needs to run properly at 50 MHz, make sure that a timing constraint is set in Quartus to constrain the circuit’s clock to this frequency. Read the Report produced by the Quartus Timing Analyzer to ensure that your circuit operates at this speed; if not, use the Quartus tools to analyze your circuit and modify your Verilog code to make a more efficient design that meets the 50-MHz speed requirement. Also note that the Run input is asynchronous to the clock signal, so make sure to synchronize this input using flip-flops. Connect the LEDs register in Figure 2 to LEDR8−0 so that you can observe the output produced by the processor.

➢ Compile the circuit, download it into the FPGA chip, and ensure that your program runs properly.

Check: Your report has to show two results:

The Code:

```
module Processor(
            input logic [8:0] DIN,
            input clk,
            input logic  Run, Resetn,
            output logic [8:0] ADDRout,DOUTout,
            output logic Done,Win,
            output [8:0] IR_test, BUS_test
);
reg [8:0] IRout,R0d,R1d,R2d,R3d,R4d,R5d,R6d,R7d,Ad,hehe,hehed;
logic incr_pc, ADDRin, DOUTin, W_D;
reg [7:0] Rout,Rin;
logic [8:0] Bus;
//assign Zero = G_Q == 0;
reg IRin,Ain,Addsub,Gin,Gout,DINout,incre_PC,Zero;
ControlUnitFSM ControlunitFSM(.Clk(clk),.Run(Run),.Resetn(Resetn),.Zero(Zero),
.IR(IR_Q),.IRin(IRin),.Rout(Rout),.Gout(Gout),.DINout(DINout),
.Rin(Rin),.Ain(Ain),.Gin(Gin),.AddSub(AddSub),.incr_pc(incr_pc),
.ADDRin(ADDRin),.DOUTin(DOUTin),.W(W_D),.Done(Done));

NineBitRegister IR(.clk(clk),.Enable(IRin),.D(DIN),.Q(IRout));
OneBitRegister ff_W(.clk(clk),.R(W_D),.Q(Win));
NineBitRegister R0(.clk(clk),.Enable(Rin[0]),.D(Bus),.Q(R0d));
NineBitRegister R1(.clk(clk),.Enable(Rin[1]),.D(Bus),.Q(R1d));
NineBitRegister R2(.clk(clk),.Enable(Rin[2]),.D(Bus),.Q(R2d));
NineBitRegister R3(.clk(clk),.Enable(Rin[3]),.D(Bus),.Q(R3d));
NineBitRegister R4(.clk(clk),.Enable(Rin[4]),.D(Bus),.Q(R4d));
NineBitRegister R5(.clk(clk),.Enable(Rin[5]),.D(Bus),.Q(R5d));
NineBitRegister R6(.clk(clk),.Enable(Rin[6]),.D(Bus),.Q(R6d));
```

```
NineBitRegister2 R7(.clk(clk),.enable(Rin[7]),.incr_pc(incr_pc),.Resetn(Resetn),.D(Bus),.Q(R7d));
NineBitRegister A(.clk(clk),.Enable(Ain),.D(Bus),.Q(Ad));
NineBitRegister G(.clk(clk),.Enable(Gin),.D(hehe),.Q(hehed),.Zero(Zero));
//NineBitRegister ADDR(.clk(clk),.Enable(ADDRin),.D(Bus),.Q(ADDRout));
//NineBitRegister DOUT(.clk(clk),.Enable(DOUTin),.D(Bus),.Q(DOUTout));
Multiplexer
Multiplexer(.DINout(DINout),.Gout(Gout),.Rout(Rout),.DIN(DIN),.R0(R0d),.R1(R1d),.R2(R2d),.R3(R3d),.
R4(R4d),.R5(R5d),.R6(R6d),.R7(R7d),.G(hehed),.Bus(Bus));
AddSub AddSub(.A(Ad),.B(Bus),.AddSub(Addsub),.G(hehe));
NineBitRegister2 ff_ADDR(.clk(clk),.enable(ADDRin),.D(Bus),.Q(ADDRout));
NineBitRegister2 ff_DOUT(.clk(clk),.enable(DOUTin),.D(Bus),.Q(DOUTout));

assign IR_test = IRout;
assign BUS_test = Bus;
endmodule
```

module ControlUnitFSM(

```
input Resetn, Clk, Run,
input [8:0] IR,
input Zero,
output [7:0] Rout,
output DINout, Gout,
output IRin, Ain, Gin, AddSub,
output incr_pc, ADDRin, DOUTin, W,
output [7:0] Rin,
output Done
);

logic [2:0] Tstep_Q, Tstep_D;
logic [2:0] III, XXX, YYY;
logic [7:0] RXin_temp, RXout_temp, RYout_temp;

assign III = IR[8:6];
assign XXX = IR[5:3];
assign YYY = IR[2:0];

decoder rxin(.in(XXX),.register(RXin_temp));
decoder rxout(.in(XXX),.register(RXout_temp));
decoder ryout(.in(YYY),.register(RYout_temp));

parameter T0 = 3'b000, T1 = 3'b001, T2 = 3'b010, T3 = 3'b011, T4 = 3'b100, T5 = 3'b101;

  // Control FSM state table
  always @(Tstep_Q, Run, Done)
    case (Tstep_Q)
      T0: // instruction fetch
        if (~Run) Tstep_D = T0;
```

```
            else Tstep_D = T1;
        T1: // wait cycle for synchronous memory
            Tstep_D = T2;
        T2: // this time step stores the instruction word in IR
            Tstep_D = T3;
        T3: // some instructions end after this time step
            if (Done) Tstep_D = T0;
            else Tstep_D = T4;
        T4: // always go to T5 after this
            Tstep_D = T5;
        T5: // instructions end after this time step
            Tstep_D = T0;
        default: Tstep_D = 3'bxxx;
    endcase

parameter mv = 3'b000, mvi = 3'b001, add = 3'b010, sub = 3'b011, ld = 3'b100, st = 3'b101, mvnz = 3'b110;

// Control FSM outputs
always @(*) begin
Done = 1'b0; Ain = 1'b0; Gin = 1'b0; AddSub = 1'b0; IRin = 1'b0; Rin = 8'b0;
DINout = 1'b0; Gout = 1'b0; Rout = 8'b0;
incr_pc = 1'b0; ADDRin = 1'b0; DOUTin = 1'b0; W = 1'b0;
                    case (Tstep_Q)
        T0: begin // fetch the instruction
            Rout = 8'b10000000; // put pc onto the internal bus
            ADDRin = 1'b1;
            incr_pc = Run; // to increment pc
        end
        T1: // wait cycle for synchronous memory
            ;
                                    T2: // store instruction on DIN in IR
            IRin = 1'b1;
        T3: // define signals in T3
            case (III)
                mv: begin
                    Rout = RYout_temp;
                                                            Rin = RXin_temp;
                                                            Done = 1'b1;
                end
                mvi: begin
                                                            Rout = 8'b10000000;
                                                            ADDRin = 1'b1;
                                                            incr_pc = 1'b1;
                end
                add, sub: begin
                    Rout = RXout_temp;
                                                            Ain = 1'b1;
                    end
                                                            ld, st: begin
Rout = RYout_temp;
                                                                        ADDRin = 1'b1;
                                                            end
                    mvnz: begin
                                                                        if (Zero) begin
                                                                              Rout = RYout_temp;
                                                                              Rin = RXin_temp;
                                                                              Done = 1'b1;
                                                                              end
                                                                        else
                                                                              Done = 1'b1;
                                                            end
                                                            default: begin
                                                                  Done = 1'b0; Ain = 1'b0; Gin = 1'b0;
                                                           AddSub = 1'b0; IRin = 1'b0; Rin = 8'b0;
                                                     DINout = 1'b0; Gout = 1'b0; Rout = 8'b0;
                                         incr_pc = 1'b0; ADDRin = 1'b0; DOUTin = 1'b0; W = 1'b0;
                                                           end
                  endcase
              T4: // define signals T4
                  case (III)
                    add: begin
                      Rout = RYout_temp;
                                                                        Gin = 1'b1;
                    end
                    sub: begin
                      Rout = RYout_temp;
                                                                        Gin = 1'b1;
                                                                        AddSub = 1'b1;
                    end

                                                            mvi: // wait cycle for synchronous memory
                                                                        ;
                                                            ld: // wait cycle for synchronous memory
                      ;
                    st: begin
                      Rout = RXout_temp;
                                                                        DOUTin = 1'b1;
                                                                        W = 1'b1;
                    end
                    default: begin
                                                                                    Done = 1'b0; Ain = 1'b0;
Gin = 1'b0;
                                                                                    AddSub = 1'b0; IRin =
1'b0; Rin = 8'b0;
```

```
                                                                 DINout = 1'b0; Gout =
1'b0; Rout = 8'b0;
                                                                 incr_pc = 1'b0; ADDRin =
1'b0; DOUTin = 1'b0; W = 1'b0;
                                                                 end
        endcase
      T5: // define T5
        case (III)
          add, sub: begin
            Gout = 1'b1;
                                                      Rin = RXin_temp;
                                                      Done = 1'b1;
          end
                                          mvi: begin
                                                      DINout = 1'b1;
                                                      Rin = RXin_temp;
                                                      Done = 1'b1;
                                          end
                                          ld: begin
            DINout = 1'b1;
                                                      Rin = RXin_temp;
                                                      Done = 1'b1;
          end
          st: // wait cycle for synhronous memory
            Done = 1'b1;
          default: begin
                                                                 Done = 1'b0; Ain = 1'b0;
Gin = 1'b0;
                                                                 AddSub = 1'b0; IRin =
1'b0; Rin = 8'b0;
                                                                 DINout = 1'b0; Gout =
1'b0; Rout = 8'b0;
                                                                 incr_pc = 1'b0; ADDRin =
1'b0; DOUTin = 1'b0; W = 1'b0;
                                                                 end
        endcase
    endcase
  end

// Control FSM flip-flops
always @(posedge Clk, negedge Resetn)
             if (!Resetn)
                   Tstep_Q <= T0;
             else
    Tstep_Q <= Tstep_D;
endmodule
```

module decoder(

```
                    input logic [2:0] in,
                    output logic [7:0] register
);
always_comb begin
case (in)
3'b000: register = 8'b00000001;
3'b001: register = 8'b00000010;
3'b010: register = 8'b00000100;
3'b011: register = 8'b00001000;
3'b100: register = 8'b00010000;
3'b101: register = 8'b00100000;
3'b110: register = 8'b01000000;
3'b111: register = 8'b10000000;
endcase
end
endmodule
```

module AddSub(

```
                    input logic [8:0] A,B,
                    input logic [0:0] AddSub,
                    output logic [8:0] G
);
always_comb begin
case (AddSub)
1: G=A+B;
default: G= A-B;
endcase
end
endmodule
```

module logicgates(

```
                    input logic [8:0] ADDRin,DOUTin,
                    input logic W,
                    output logic wr_en,Enable,
                    output logic [6:0] ADDR,
                    output logic [8:0] DOUT
);
assign ADDR = ADDRin[6:0];
reg NOR1,NNOR1;
assign NOR1 = ~(ADDRin[7]|ADDRin[8]);
assign wr_en = NOR1 & W;// if 1 => wr_en on
assign NNOR1 = (~(~ADDRin[7])|ADDRin[8]);
assign Enable = NNOR1&W;
assign DOUT =DOUTin;
endmodule
```

Laboratory 5:

AN ENHANCED PROCESSOR

module Multiplexer(

```
input [8:0] DIN,
input [8:0] R0, R1, R2, R3, R4, R5, R6, R7,
input [8:0] G,
input [7:0] Rout,
input DINout, Gout,
output [8:0] Bus
);

logic [9:0] Sel;

assign Sel = {Rout, DINout, Gout};

always_comb begin
                case (Sel)
                    10'b1000000000: Bus = R7;
                    10'b0100000000: Bus = R6;
                    10'b0010000000: Bus = R5;
                    10'b0001000000: Bus = R4;
                    10'b0000100000: Bus = R3;
                    10'b0000010000: Bus = R2;
                    10'b0000001000: Bus = R1;
                    10'b0000000100: Bus = R0;
                    10'b0000000010: Bus = DIN;
                    10'b0000000001: Bus = G;
                    default: Bus = '0;
                endcase
end
endmodule
```

module NineBitRegister(

```
                input logic[0:0] Enable,
                input clk,
                input logic [8:0] D,
                output logic [8:0] Q,
                output logic Zero
);
always@(posedge clk) begin
if ( Enable) begin
Q<=D;
//Q=D;
end
end
always@(posedge clk) begin
if (D==0) begin
```

```
Zero = 1;
end
else
begin
Zero = 0;
end
end
endmodule
```

module NineBitRegister2(

```
input clk, enable, incr_pc, Resetn,
input [8:0] D,
output [8:0] Q
);
always@(posedge clk) begin
                        if (!Resetn)
          Q <= 9'b0;
       else if (enable)
          Q <= D;
       else if (incr_pc)
          Q <= Q + 1'b1;
end
endmodule
```

module OneBitRegister(

```
input clk,
input R,
output Q
);
always@(posedge clk) begin
                                Q <= R;
end
endmodule
```

module counter(

```
                        input logic CLK,RST,
                        output logic [4:0] ADDRESS
);
always@(posedge CLK, negedge RST) begin
if(!RST) begin
ADDRESS = 0;
end
else begin
ADDRESS = ADDRESS +1;
end
```

```
end
endmodule
```

module W(

```
                    input logic D,
                    input logic clk,
                    output logic Q
);
always@(posedge clk) begin
Q=D;
end
endmodule
```

module MyROM(

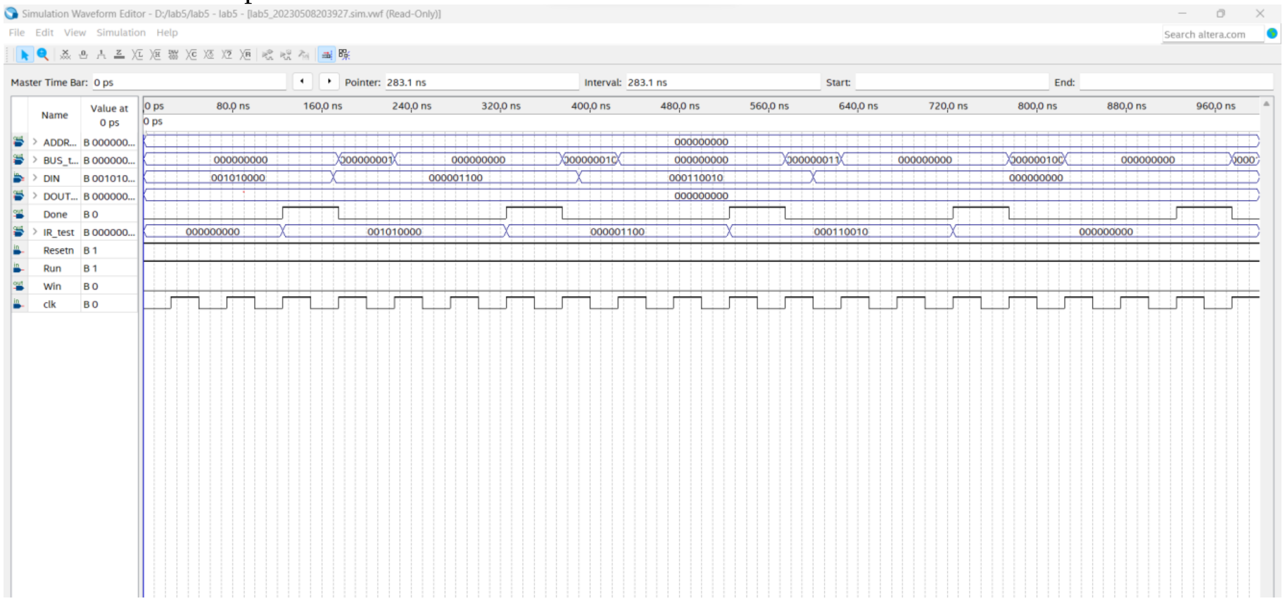
```
input logic [6:0] addr,
input logic [8:0] data,
input logic wr_en,
input logic clk,
output logic [8:0] q
);

logic [8:0] memory [0:127]; // 128 words of 9-bit memory

initial begin
$readmemb("hehe.txt",memory);
end


always_ff @ (posedge clk) begin
                    if (wr_en) begin
                          memory[addr] <= data;
  end
  q = memory[addr];
end

endmodule
```

file hehe.txt (file data ROM)

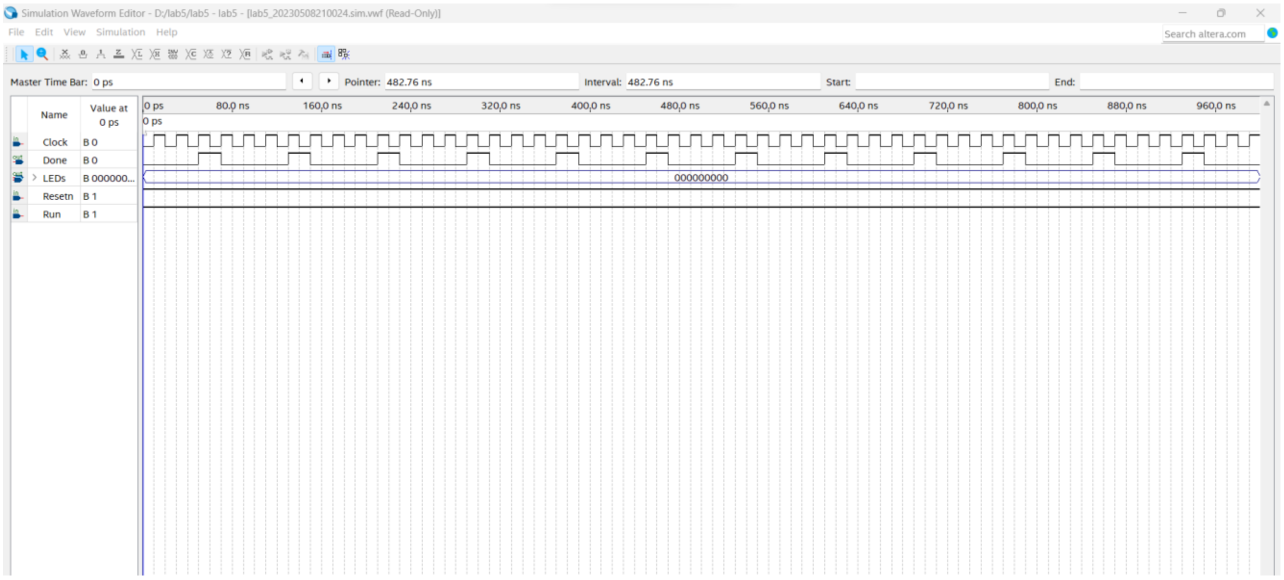
```
001010000
010011000
101010110
001000001
001000000
001010000
010011000
```

101110010
001000001
001011000
111111111
000101111

- ➢ The waveform to prove the circuit works correctly.
  - Waveform processor



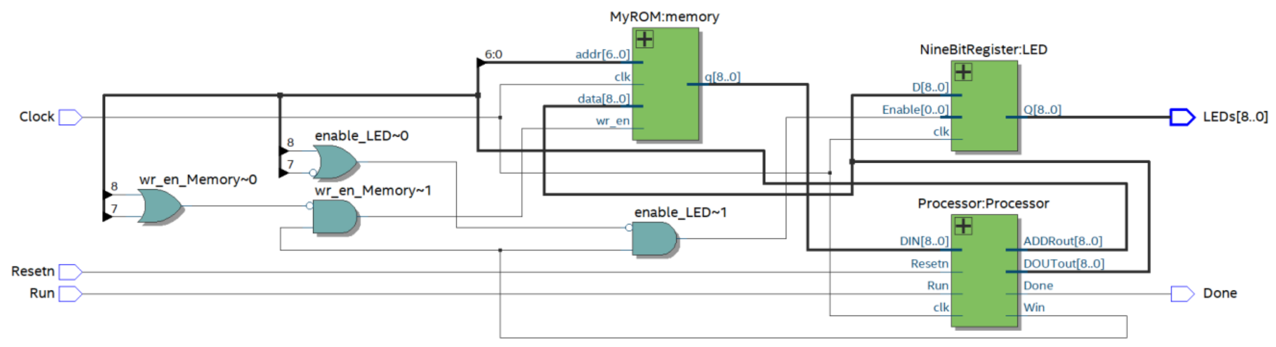
  - Waveform full:



- ➢ The result of RTL viewer.

Laboratory 5:

AN ENHANCED PROCESSOR



MyROM:memory
addr[6..0]
clk
data[8..0]
wr_en
q[8..0]
6:0
NineBitRegister:LED
D[8..0]
Enable[0..0]
clk
Q[8..0]
LEDs[8..0]
Clock
enable_LED~0
wr_en_Memory~0
wr_en_Memory~1
enable_LED~1
8
7
Processor:Processor
DIN[8..0]
Resetn
Run
clk
ADDRout[8..0]
DOUTout[8..0]
Done
Win
Resetn
Run
Done