|  |  |
| --- | --- |
| Name | ID number |
|  |  |
|  |  |
|  |  |

# GROUP 1

# 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.
* Examine a simple processor.

# PREPARATION FOR LAB 4

* Finish Pre Lab 4 at home.
* Students have to simulate all the exercises in Pre Lab 4 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.*

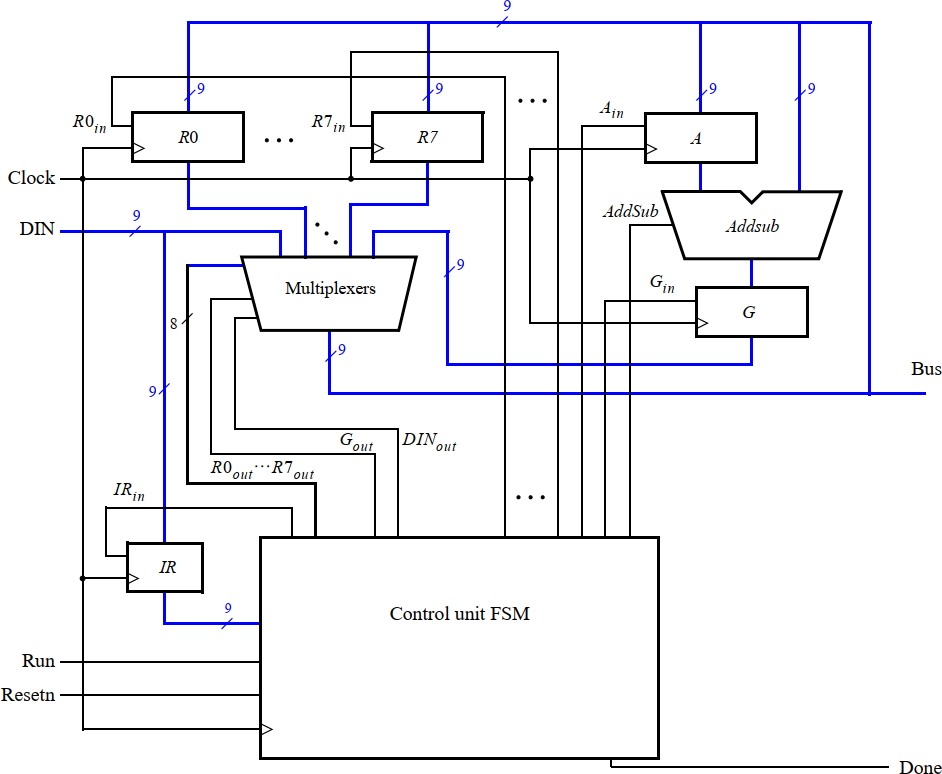
# REFERENCE

1. Intel FPGA training

# EXPERIMENT 1

***Objective:*** Design and implement a simple processor.

***Requirement:*** Design and implement a simple processor which is shown in Figure 1.



## Instruction:

*Figure 1:* A simple processor.

* + The *Registers* block and *Addsub* subsystem is written in Lab 3, *Multiplexer* block is written in Lab 1. Modify these subsystems to satisfy the parameters of the processor.
  + The FSM control unit is prepared in your Pre Lab 5. Write the code for this block.
  + To describe the circuit given in Figure 1, write a top-level VHDL entity to connect all the subsystems above*.* A suggested skeleton of the VHDL code is shown.

module proc (DIN, Resetn, Clock, Run, Done, BusWires); input [8:0] DIN;

input Resetn, Clock, Run; output Done;

output [8:0] BusWires;

typedef enum {T0 = 2’b00, T1 = 2’b01, T2 = 2’b10, T3 = 2’b11} states ;

…

… declare variables assign I = IR[1:3];

dec3to8 decX (IR[4:6], 1’b1, Xreg); dec3to8 decY (IR[7:9], 1’b1, Yreg);

// Control FSM state table always @(Tstep\_Q, Run, Done) begin

case (Tstep\_Q)

T0: // data is loaded into IR in this time step if (!Run) Tstep\_D = T0;

else Tstep\_D = T1; T1: . . .

endcase end

// Control FSM outputs

always @(Tstep\_Q or I or Xreg or Yreg) begin

. . . specify initial values case (Tstep\_Q)

T0: // store DIN in IR in time step 0 Begin

IRin = 1’b1;

* + In your design, you may need to use a *decoder 3 – 8*. The code for it is shown.

end

T1: //define signals in time step 1 case (I)

. . .

endcase

T2: //define signals in time step 2 case (I)

. . .

endcase

T3: //define signals in time step 3 case (I)

. . .

endcase endcase end

// Control FSM flip-flops

always @(posedge Clock, negedge Resetn) if (!Resetn)

. . .

regn reg\_0 (BusWires, Rin[0], Clock, R0);

. . . instantiate other registers and the adder/subtractor unit

. . . define the bus endmodule

module dec3to8(W, En, Y); input [2:0] W;

input En; output [0:7] Y;

reg [0:7] Y;

always @(W or En) begin

* + Use functional simulation to verify that your code is correct. An example of the output produced by a functional simulation for a correctly-designed circuit is given in Figure 2. It shows the value (010)8 being loaded into *IR* from *DIN* at time 30 ns. This pattern represents the instruction **mvi** R0,#D, where the value *D* = 5 is loaded into *R*0 on the clock edge at 50 ns. The simulation then shows the instruction **mv** R1,R0 at 90 ns, **add** R0,R1 at 110 ns,

if (En == 1) case (W)

3’b000: Y = 8’b10000000;

3’b001: Y = 8’b01000000;

3’b010: Y = 8’b00100000;

3’b011: Y = 8’b00010000;

3’b100: Y = 8’b00001000;

3’b101: Y = 8’b00000100;

3’b110: Y = 8’b00000010;

3’b111: Y = 8’b00000001;

endcase

else Y = 8’b00000000;

end endmodule

module regn(R, Rin, Clock, Q); parameter n = 9;

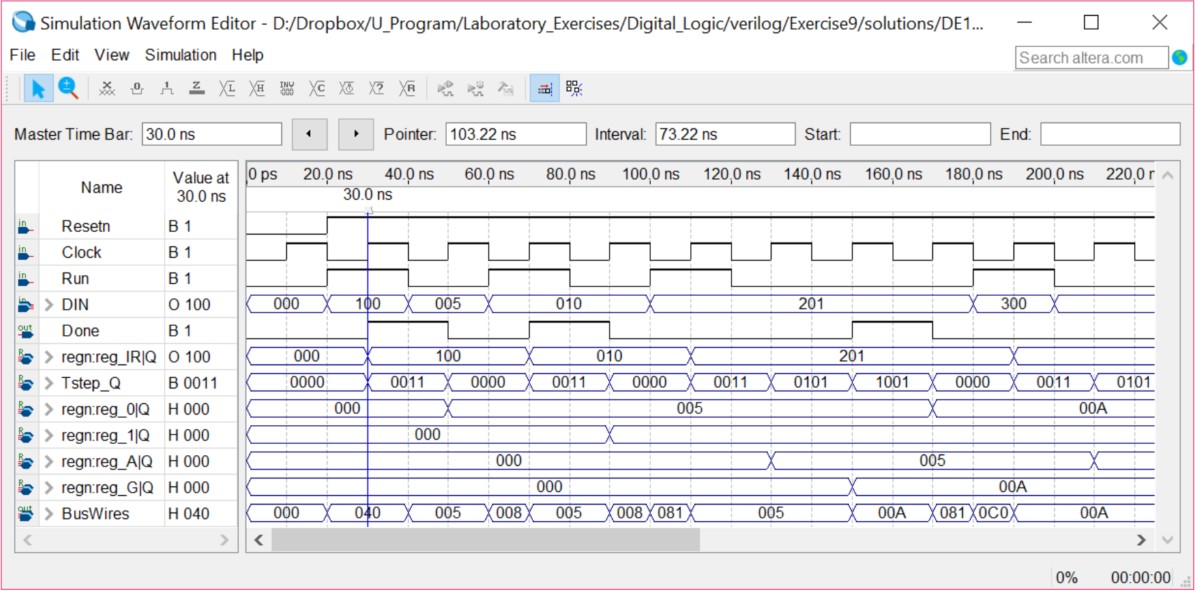
input [n-1:0] R; input Rin, Clock; output [n-1:0] Q;

reg [n-1:0] Q;

always @(posedge Clock) if (Rin) Q <= R;

endmodule

and **sub** R0,R0 at 190 ns. Note that the simulation output shows *DIN* and *IR* in octal, and it shows the contents of other registers in hexadecimal.



*Figure 2:* Simulation result for the processor.

***Check:*** Your report has to show two results:

Code:

module ***LAB4\_EXer1***(

input [8:0] DIN,

input Clock, Run, Resetn,

output [8:0] Bus,

output Done

);

logic [7:0] Rout, Rin;

logic Gout, DINout, IRin, Ain, Gin, AddSub;

logic [8:0] IR\_Q;

logic [8:0] G\_Q, A\_Q;

logic [8:0] R0\_Q, R1\_Q, R2\_Q, R3\_Q, R4\_Q, R5\_Q, R6\_Q, R7\_Q;

logic [8:0] AddSub\_Result;

Control\_unit FSM(.Clock(Clock),.Run(Run),.Resetn(Resetn),.IR(IR\_Q),.IRin(IRin),

.Rout(Rout),.Gout(Gout),.DINout(DINout),.Rin(Rin),.Ain(Ain),.Gin(Gin),

.AddSub(AddSub),.Done(Done));

NineBitRegister IR(.clk(Clock),.enable(IRin),.R(DIN),.Q(IR\_Q));

NineBitRegister Q (.clk(Clock),.enable(Gin),.R(AddSub\_Result),.Q(G\_Q));

NineBitRegister A (.clk(Clock),.enable(Ain),.R(Bus),.Q(A\_Q));

NineBitRegister R0(.clk(Clock),.enable(Rin[0]),.R(Bus),.Q(R0\_Q));

NineBitRegister R1(.clk(Clock),.enable(Rin[1]),.R(Bus),.Q(R1\_Q));

NineBitRegister R2(.clk(Clock),.enable(Rin[2]),.R(Bus),.Q(R2\_Q));

NineBitRegister R3(.clk(Clock),.enable(Rin[3]),.R(Bus),.Q(R3\_Q));

NineBitRegister R4(.clk(Clock),.enable(Rin[4]),.R(Bus),.Q(R4\_Q));

NineBitRegister R5(.clk(Clock),.enable(Rin[5]),.R(Bus),.Q(R5\_Q));

NineBitRegister R6(.clk(Clock),.enable(Rin[6]),.R(Bus),.Q(R6\_Q));

NineBitRegister R7(.clk(Clock),.enable(Rin[7]),.R(Bus),.Q(R7\_Q));

AddSub ALU(.A(A\_Q),.B(Bus),.AddSub(AddSub),.Result(AddSub\_Result));

Mux10\_1 select(.DIN(DIN),.R0(R0\_Q),.R1(R1\_Q),.R2(R2\_Q),.R3(R3\_Q),.R4(R4\_Q),.R5(R5\_Q),

.R6(R6\_Q),.R7(R7\_Q),.G(G\_Q),.Rout(Rout),.DINout(DINout),.Gout(Gout),.Bus(Bus));

endmodule

module ***Control\_unit***(

input Resetn, Clock, Run,

input [8:0] IR,

output [7:0] Rout,

output DINout, Gout,

output IRin, Ain, Gin, AddSub,

output [7:0] Rin,

output Done

);

logic [1:0] 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];

Dec3\_8 rxin(.W(XXX),.Y(RXin\_temp));

Dec3\_8 rxout(.W(XXX),.Y(RXout\_temp));

Dec3\_8 ryout(.W(YYY),.Y(RYout\_temp));

parameter T0 = 2'b00, T1 = 2'b01, T2 = 2'b10, T3 = 2'b11;

// Control FSM state table

always @(Tstep\_Q, Run, Done)

case (Tstep\_Q)

T0: // data is loaded into IR in this time step

if (~Run) Tstep\_D = T0;

else Tstep\_D = T1;

T1: // some instructions end after this time step

if (Done) Tstep\_D = T0;

else Tstep\_D = T2;

T2: // always go to T3 after this

Tstep\_D = T3;

T3: // instructions end after this time step

Tstep\_D = T0;

default: Tstep\_D = 2'bxx;

endcase

parameter mv = 3'b000, mvi = 3'b001, add = 3'b010, sub = 3'b011;

// Control FSM outputs

always @(Tstep\_Q) 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;

case (Tstep\_Q)

T0: // store instruction on DIN in IR

IRin = 1'b1;

T1: // define signals in T1

case (III)

mv: begin

Rout = RYout\_temp;

Rin = RXin\_temp;

Done = 1'b1;

end

mvi: begin

DINout = 1'b1;

Rin = RXin\_temp;

Done = 1'b1;

end

add, sub: begin

Rout = RXout\_temp;

Ain = 1'b1;

Done = 1'b0;

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;

end

endcase

T2: // define signals T2

case (III)

add: begin

Rout = RYout\_temp;

Gin = 1'b1;

AddSub = 1'b0;

end

sub: begin

Rout = RYout\_temp;

Gin = 1'b1;

AddSub = 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;

end

endcase

T3: // define T3

case (III)

add, sub: begin

Gout = 1'b1;

Rin = RXin\_temp;

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;

end

endcase

endcase

end

// Control FSM flip-flops

always @(posedge Clock, negedge Resetn)

if (!Resetn)

Tstep\_Q <= T0;

else

Tstep\_Q <= Tstep\_D;

endmodule

module ***Dec3\_8***(

input [2:0] W,

output [7:0] Y

);

always\_comb begin

case (W)

3'b000: Y = 8'b00000001;

3'b001: Y = 8'b00000010;

3'b010: Y = 8'b00000100;

3'b011: Y = 8'b00001000;

3'b100: Y = 8'b00010000;

3'b101: Y = 8'b00100000;

3'b110: Y = 8'b01000000;

3'b111: Y = 8'b10000000;

endcase

end

endmodule

module ***Mux10\_1***(

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 ***AddSub***(

input [8:0] A, B,

input AddSub,

output [8:0] Result

);

always\_comb begin

if(!AddSub) Result <= A + B;

else Result <= A - B;

end

endmodule

module ***NineBitRegister***(

input clk, enable,

input [8:0] R,

output [8:0] Q

);

always@(posedge clk) begin

if (enable)

Q <= R;

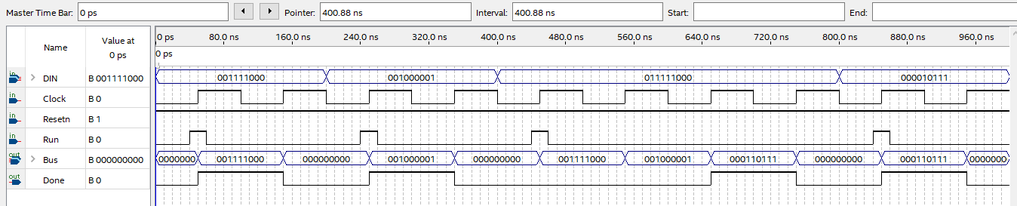
else

;

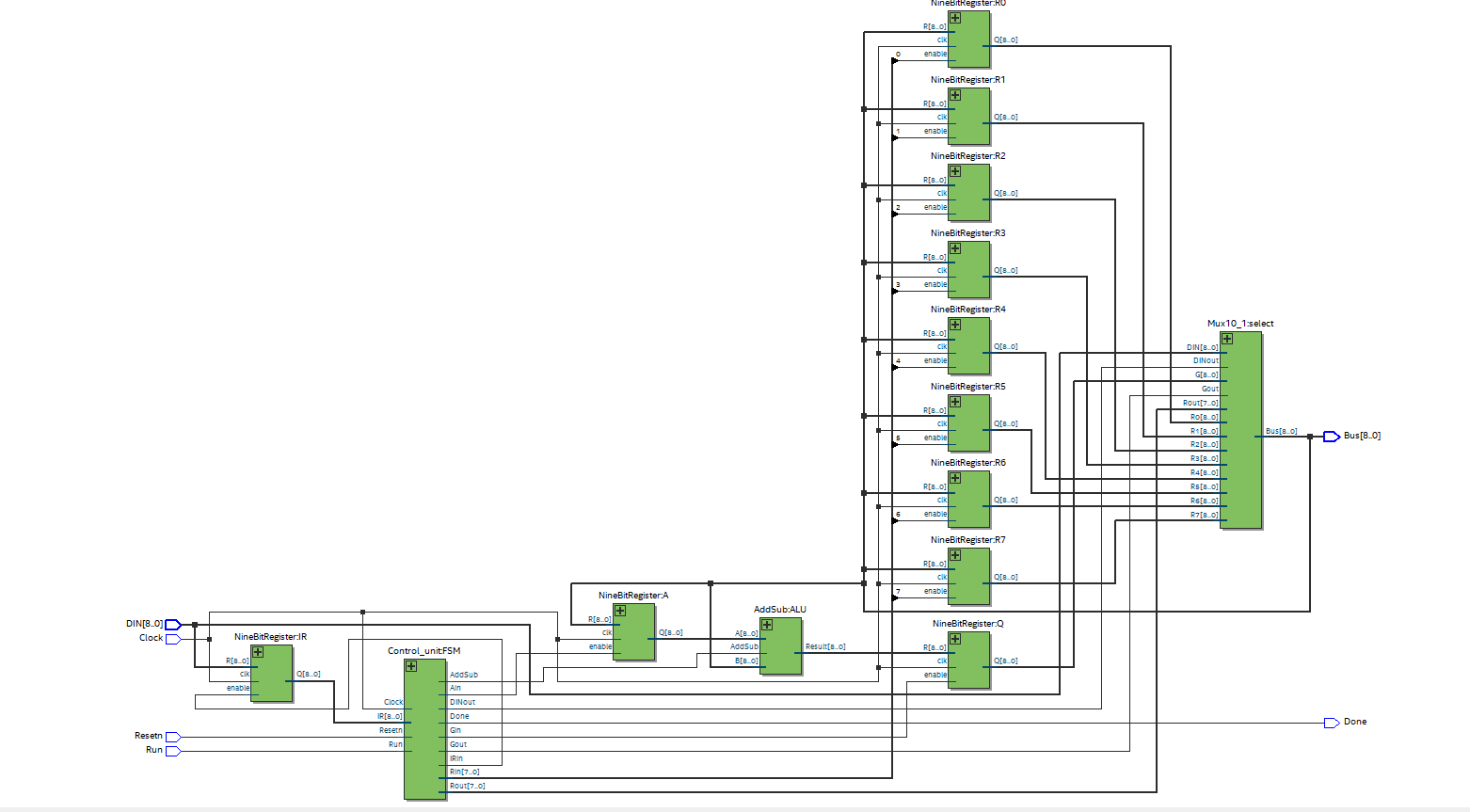
end

endmodule

* + The waveform to prove the circuit works correctly.



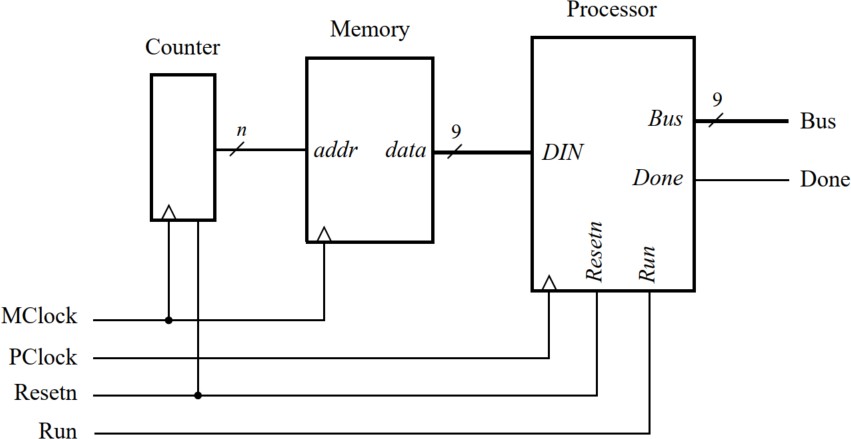
* + The result of RTL viewer.



# EXPERIMENT 2

***Objective:*** Design and implement a simple processor with memory.

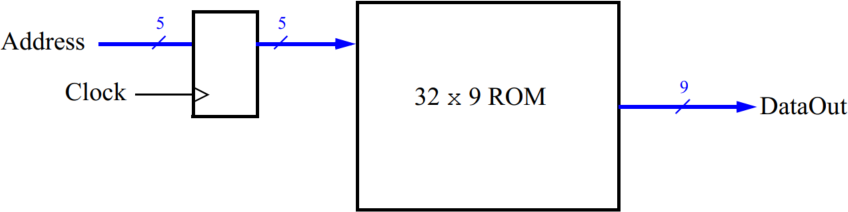
***Requirement:*** Extend the circuit from Experiment 1 to the circuit in Figure 3, in which a memory module and counter are connected to the processor. The counter is used to read the contents of successive addresses in the memory, and this data is provided to the processor as a stream of instructions. To simplify the design and testing of this circuit we have used separate clock signals, *PClock* and *MClock*, for the processor and memory.



## Instruction:

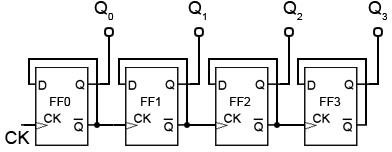
*Figure 3:* Connecting the processor to a memory and counter.

* + A diagram of the memory module that we need to create is depicted in Figure 4. The System Verilog code for this module is prepared in exercise 3, pre lab 4.



*Figure 4:* The 32 x 9 ROM with address register.

* + A diagram of the counter is shown in Figure 5. Write System Verilog code for the counter using the hint from the Figure.



*Figure 5:* The 5 bit serial counter.

* + Use functional simulation to verify that your code is correct.

***Check:*** Your report has to show two results:

Code:

module ***Lab4\_ex2***(

input MClock, PClock, Resetn, Run,

output Done,

output [8:0] Bus

);

logic [8:0] DATA;

top\_mem Getaddress(

.CLK(MClock),

.RST(Resetn),

.DATAOUT(DATA)

);

LAB4\_EXer1 Getdata(

.DIN(DATA),

.Clock(PClock),

.Resetn(Resetn),

.Run(Run),

.Bus(Bus),

.Done(Done)

);

endmodule

module ***top\_mem***(

input logic CLK, RST,

output logic [8:0] DATAOUT

);

logic [4:0] ADDRESS;

address\_counter getaddress(

.CLK (CLK),

.RST (RST),

.ADDRESS (ADDRESS)

);

MyROM getdata(

.CLK (CLK),

.ADDRESS (ADDRESS),

.DATAOUT (DATAOUT)

);

endmodule

module ***address\_counter***(

input CLK, RST,

output [4:0] ADDRESS

);

always @ (posedge CLK or negedge RST)

begin

if (!RST) ADDRESS <= '0;

else ADDRESS <= ADDRESS + 1;

end

endmodule

module ***MyROM***

#(parameter int unsigned width = 9,

parameter int unsigned depth = 32,

parameter intFile = "inst\_mem.txt",

parameter int unsigned addrBits = 5)

(

input logic CLK,

input logic [addrBits-1:0] ADDRESS,

output logic [width-1:0] DATAOUT

);

logic [width-1:0] rom [0:depth-1];

// initialise ROM contents

initial begin

$readmemb(intFile,rom);

end

always\_ff @ (posedge CLK)

begin

DATAOUT <= rom[ADDRESS];

end

endmodule

***inst\_mem.txt file:***

**001101110**

**001110000**

**011110101**

**101011001**

**111001001**

**010100111**

**110101000**

**001000111**

**100101010**

**011011001**

**101101110**

**111000101**

**010001011**

**110110010**

**001011101**

**100110001**

**011000111**

**101000101**

**111011110**

**010010001**

**110001011**

**001111100**

**100000001**

**011010010**

**101010111**

**111100111**

**010111001**

**110010110**

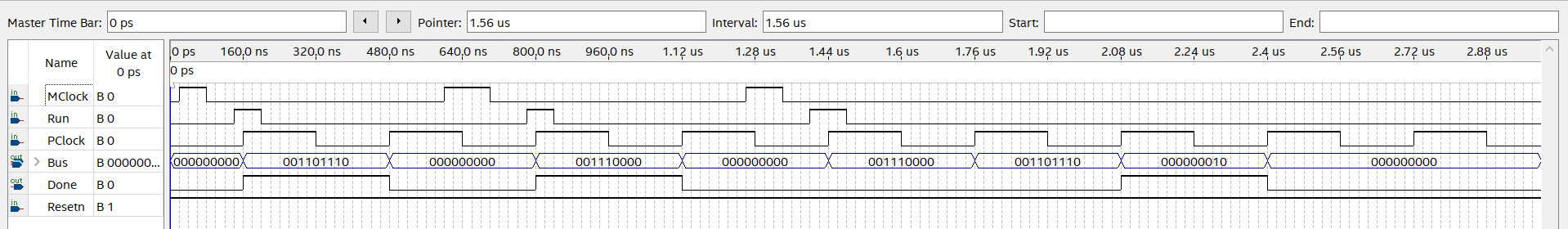
**001100101**

**100001011**

**011101000**

**101111010**

* + The waveform to prove the circuit works correctly.



* + The result of RTL viewer.

