Laboratory 4:

A SIMPLE 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.
- 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

Laboratory 4:

A SIMPLE PROCESSOR

EXPERIMENT 1

Objective: Design and implement a simple processor.

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

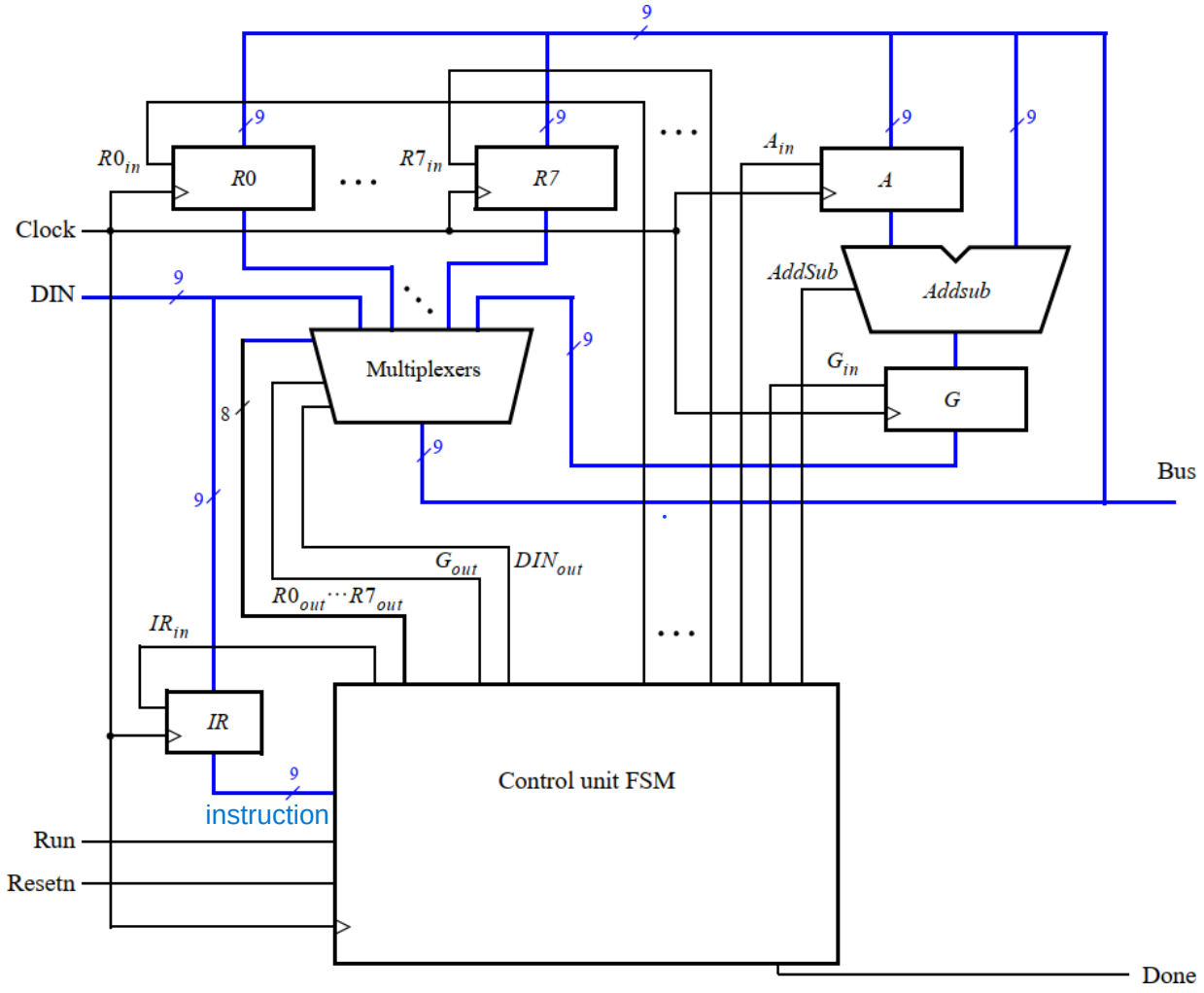


Figure 1: A simple processor.

Instruction:

- 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.

Laboratory 4:

A SIMPLE PROCESSOR

- 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;
```

```
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
```

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

```
module dec3to8(W, En, Y);
input [2:0] W;
input En;
output [0:7] Y;
reg [0:7] Y;
always @(W or En)
begin
```

```
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
```

- 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,

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.

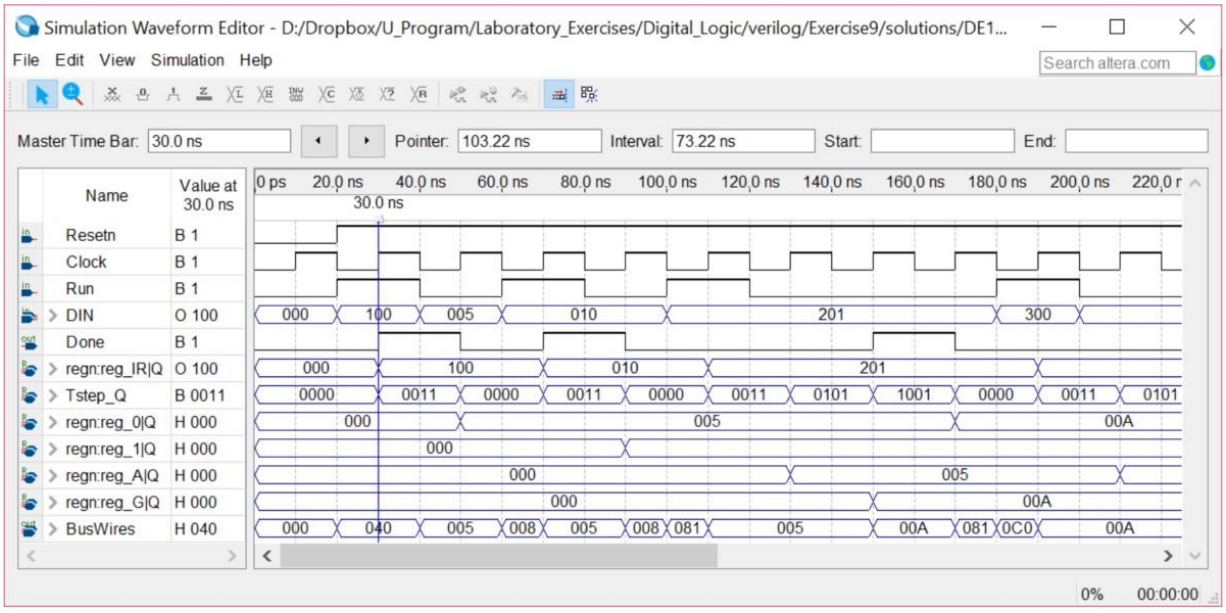


Figure 2: Simulation result for the processor.

Check: Your report has to show two results:

The code:

```
module lab4_bai1 (
        input logic  Resetn,Clock,Run,
        output reg  Done,
        output reg [1:0] Tstep_Q,
        output reg [8:0] IR,A,G,
        output reg [8:0] R0,R1,R2,R3,R4,R5,R6,R7,
        output logic [8:0] BusWires
);

logic [0:0] Ain,Gin,AddSub,CLKm;
logic [7:0] select,Rin;
logic [8:0] result,DIN;

regn reg_IR (DIN,IRin,Clock,IR);
fsm controller(
        .clk (Clock),
        .run (Run),
        .rst (Resetn),
        .IR (IR),
```

```
        .clkm (CLKm),
        .IRin (IRin),
        .Ain (Ain),
        .Gin (Gin),
        .AddSub (AddSub),
        .Done (Done),
        .Rin (Rin),
        .sel (select),
        .state (Tstep_Q)
);

regn reg_0 (BusWires, Rin[7], Clock, R0);
regn reg_1 (BusWires, Rin[6], Clock, R1);
regn reg_2 (BusWires, Rin[5], Clock, R2);
regn reg_3 (BusWires, Rin[4], Clock, R3);
regn reg_4 (BusWires, Rin[3], Clock, R4);
regn reg_5 (BusWires, Rin[2], Clock, R5);
regn reg_6 (BusWires, Rin[1], Clock, R6);
regn reg_7 (BusWires, Rin[0], Clock, R7);

regn reg_A (BusWires, Ain, Clock, A);
alu add_sub(
        .A (A),
        .BusWires (BusWires),
        .AddSub (AddSub),
        .result (result)
);
regn reg_G (result, Gin,Clock,G);

multiplexer mux10(
        .R0(R0), .R1(R1), .R2(R2), .R3(R3),
        .R4(R4), .R5(R5), .R6(R6), .R7(R7),
        .G(G),
        .DIN(DIN),
        .sel(select),
        .BusWires(BusWires)
);

endmodule
```

module alu(

```
        input logic [8:0] A,BusWires,
        input logic  AddSub,
```

```
        output logic [8:0] result
);
always_comb begin
        if (AddSub == 1'b0)
                result = A + BusWires;
        else
                result = A - BusWires;
end
endmodule :alu
```

module dec3to8(

```
                input logic [2:0] W,
                input logic En,
                output logic [7:0] Y
);

always @(W or En)
begin
        if (En == 1'b1)
                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'b000000000;
end
endmodule
```

module multiplexer(

```
        input logic [7:0] sel,
        input logic [8:0] R0,R1,R2,R3,R4,R5,R6,R7,G,DIN,
        output logic [8:0] BusWires
);
always_comb begin
        case (sel)
                8'b10000000: BusWires = R0;
                8'b01000000: BusWires = R1;
                8'b00100000: BusWires = R2;
                8'b00010000: BusWires = R3;
```

```
                8'b00001000: BusWires = R4;
                8'b00000100: BusWires = R5;
                8'b00000010: BusWires = R6;
                8'b00000001: BusWires = R7;
                8'b11111110: BusWires = G;
                default:        BusWires = DIN;
        endcase
end

endmodule :multiplexer
```

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
```

module fsm(

```
        input logic [0:0] clk,run,rst,
        input logic [8:0] IR,
        output logic [0:0] IRin, Ain, Gin, AddSub, Done, clkm,
        output logic [7:0] Rin,sel,
        output logic [1:0] state
);
enum logic [1:0] {
        T0 = 2'b00,
        T1 = 2'b01,
        T2 = 2'b10,
        T3 = 2'b11,
        DEFAULT = 2'bxx} Tstep_D,Tstep_Q;

logic [0:0] high;
logic [2:0] I;
logic [7:0] Xreg,Yreg;

assign I = IR[8:6];
```

```
assign high = 1'b1;
dec3to8 decX(
        .W (IR[5:3]),
        .En (high),
        .Y (Xreg),
);
dec3to8 decY(
        .W (IR[2:0]),
        .En (high),
        .Y (Yreg),
);

assign state = Tstep_Q;

//control fsm flip flop
always @(posedge clk, negedge rst) begin
        if (!rst)
                Tstep_Q <= T0;
        else
                Tstep_Q <= Tstep_D;
end

//control fsm state table
always @(Tstep_Q, run, Done) begin
        case(Tstep_Q)
                T0: begin //data is loaded into IR
                        if (!run)
                                Tstep_D = T0;
                        else
                                Tstep_D = T1;
                        end
                T1: begin
                        if (I == 3'b001)
                                Tstep_D = T2;
                         else if (Done == 1'b0)
                                Tstep_D = T2;
                         else
                                Tstep_D = T0;
                        end
                T2: Tstep_D = T3;
                T3: Tstep_D = T0;
                default: Tstep_D = DEFAULT;
        endcase
end
```

```
//control fsm outputs
always @(Tstep_Q or I or Xreg or Yreg) begin
        Done = 1'b0; Ain = 1'b0; Gin = 1'b0; AddSub = 1'b0;
        IRin = 1'b0; Rin = 8'b0; clkm = 1'b0;

        case (Tstep_Q)
                T0: begin
                        IRin = 1'b1;
                        sel = 8'b11000000;
                  end
                T1: case (I)
                        3'b000: begin
                                                sel = Yreg;
                                                Rin = Xreg;
                                                Done = 1'b1;
                                                clkm = 1'b1;
                                        end
                        3'b001: begin
                                                sel = 8'b11000000;
                                                Rin = Xreg;
                                                Done = 1'b0;
                                                clkm = 1'b1;
                                        end
                        3'b010: begin
                                                sel = Xreg;
                                                Ain = 1'b1;
                                                Done = 1'b0;
                                                clkm = 1'b0;
                                        end
                        3'b011: begin
                                                sel = Xreg;
                                                Ain = 1'b1;
                                                Done = 1'b0;
                                                clkm = 1'b0;
                                        end
                        endcase
                T2: case (I)
                        3'b001: begin
                                                Done = 1'b0;
                                                clkm = 1'b0;
                                        end
                        3'b010: begin
                                                sel = Yreg;
```

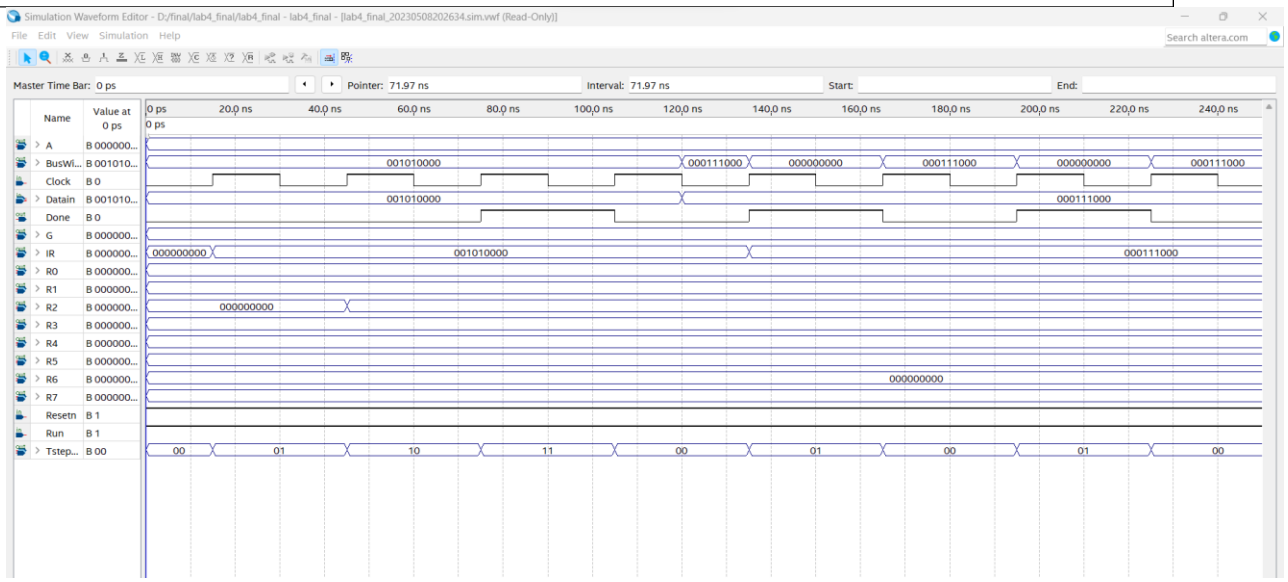
```
                                        AddSub = 1'b1;
                                        Gin = 1'b1;
                                        Done = 1'b0;
                                        clkm = 1'b1;
                                end
                        3'b011: begin
                                        sel = Yreg;
                                        AddSub = 1'b1;
                                        Gin = 1'b1;
                                        Done = 1'b0;
                                        clkm = 1'b1;
                                end
                        endcase
                T3: case (I)
                        3'b001: begin
                                        Done = 1'b1;
                                        clkm = 1'b1;
                                end
                        3'b010: begin
                                        sel = 8'b11111110;
                                        Rin = Xreg;
                                        Done = 1'b1;
                                end
                        3'b011: begin
                                        sel = 8'b11111110;
                                        Rin = Xreg;
                                        Done = 1'b1;
                                end
                        endcase
                endcase
end

endmodule
```

- The waveform to prove the circuit works correctly.

Laboratory 4:

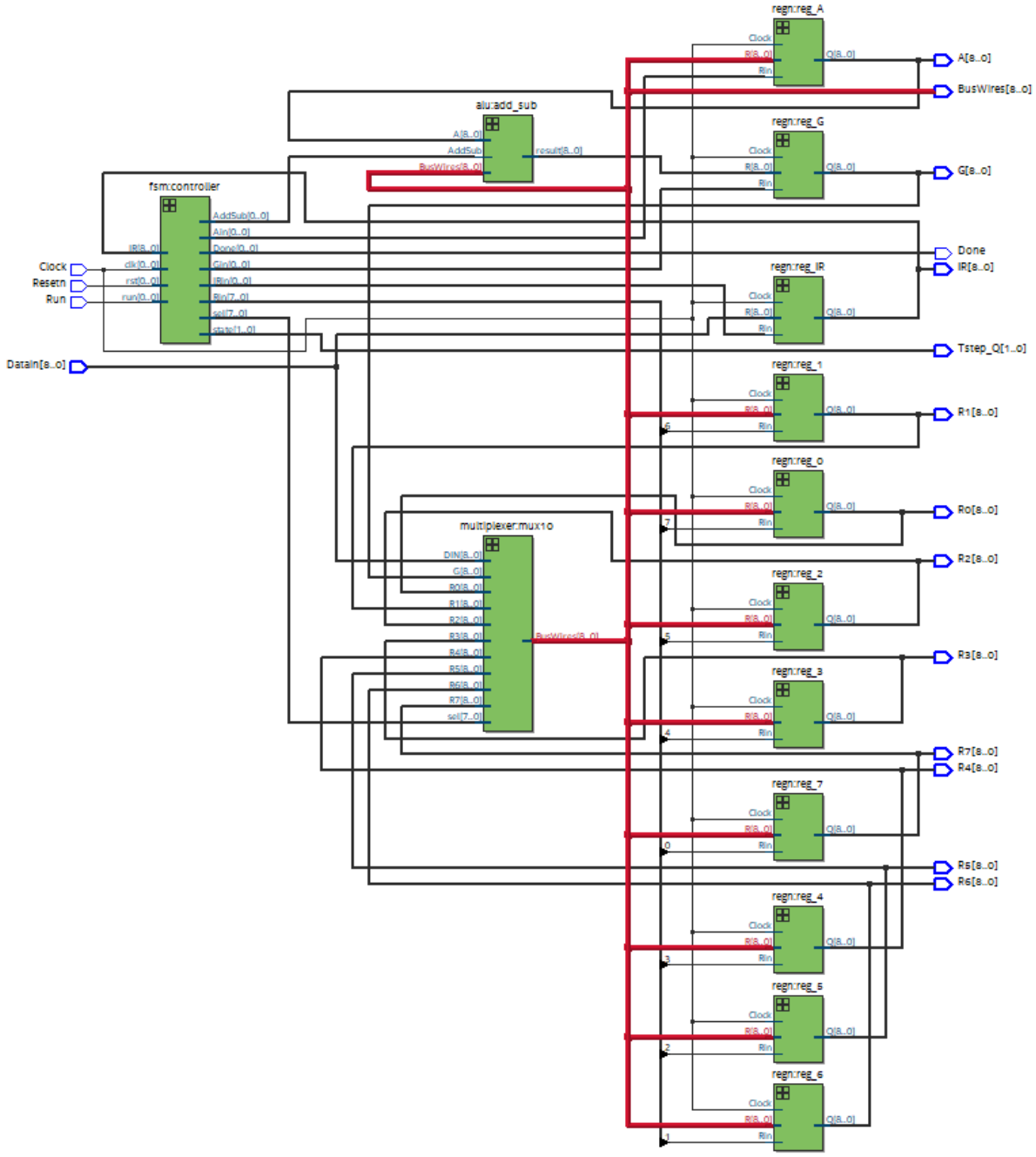
A SIMPLE PROCESSOR



- The result of RTL viewer.

Laboratory 4:

A SIMPLE PROCESSOR



Laboratory 4:

A SIMPLE PROCESSOR

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.

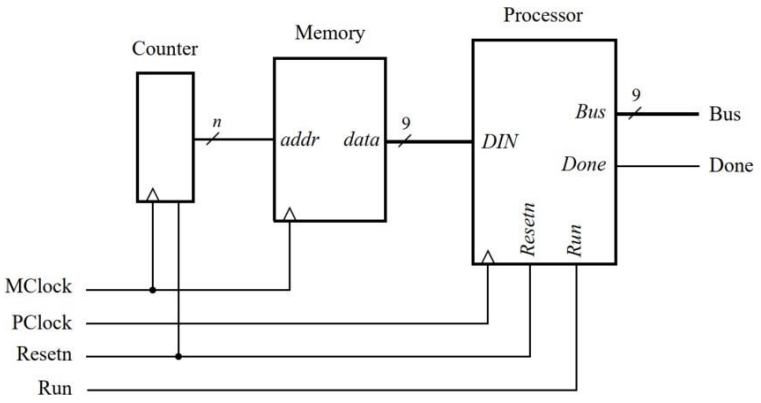


Figure 3: Connecting the processor to a memory and counter.

Instruction:

- 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.

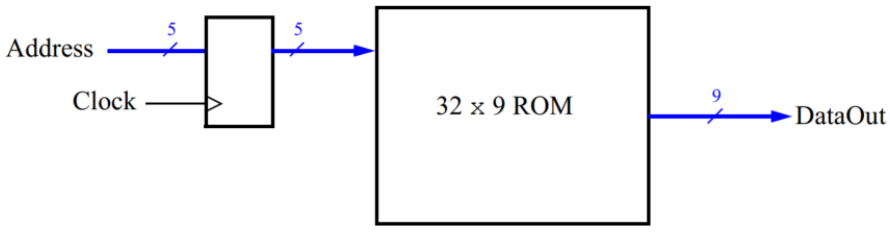


Figure 4: The 32 x 9 ROM with address register.

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

Laboratory 4:

A SIMPLE PROCESSOR

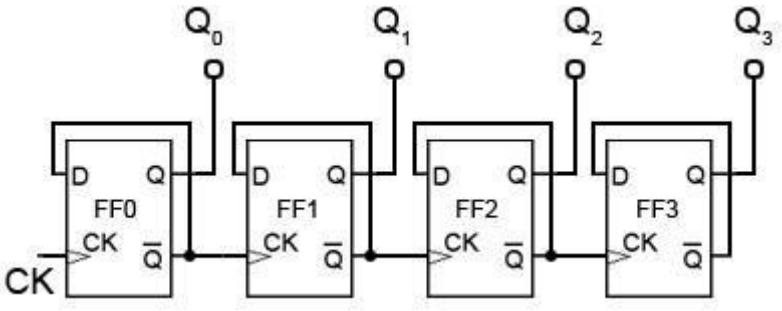


Figure 5: The 5 bit serial counter.

- Use functional simulation to verify that your code is correct.

Check: Your report has to show two results:

The code:

module MyROM

```
#(parameter int unsigned width = 9,
parameter int unsigned depth = 32,
parameter intFile = "inst_mem.mif",
parameter int unsigned addrBits = 5)
(
input logic CLKm,
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 CLKm)
begin
DATAOUT <= rom[ADDRESS];
end
endmodule
```

```
module top_mem(
        input logic [0:0] CLKm,
        output logic [8:0] DATA
);
logic [4:0] ADDRESS_out;
logic [4:0] ADDRESS_in;
int cnt = 1;
MyROM U0(
```

```
        .CLKm(CLKm),
        .ADDRESS (ADDRESS_out),
        .DATAOUT (DATA)
);
assign ADDRESS_in = cnt;
always_ff @(posedge CLKm) begin
        ADDRESS_out = ADDRESS_in;
        cnt = cnt + 1;
end
endmodule
```

module lab4_final (

```
        input logic  Resetn,Clock,Run,
        output reg  Done,
        output reg [1:0] Tstep_Q,
        output reg [8:0] IR,A,G,
        output reg [8:0] R0,R1,R2,R3,R4,R5,R6,R7,
        output logic [8:0] BusWires
);

logic [0:0] Ain,Gin,AddSub,CLKm;
logic [7:0] select,Rin;
logic [8:0] result,DIN;

top_mem ROM(
        .CLKm (CLKm),
        .DATA (DIN)
);

regn reg_IR (DIN,IRin,Clock,IR);
fsm controller(
        .clk (Clock),
        .run (Run),
        .rst (Resetn),
        .IR (IR),
        .clkm (CLKm),
        .IRin (IRin),
        .Ain (Ain),
        .Gin (Gin),
        .AddSub (AddSub),
        .Done (Done),
        .Rin (Rin),
        .sel (select),
        .state (Tstep_Q)
```

```
);

regn reg_0 (BusWires, Rin[7], Clock, R0);
regn reg_1 (BusWires, Rin[6], Clock, R1);
regn reg_2 (BusWires, Rin[5], Clock, R2);
regn reg_3 (BusWires, Rin[4], Clock, R3);
regn reg_4 (BusWires, Rin[3], Clock, R4);
regn reg_5 (BusWires, Rin[2], Clock, R5);
regn reg_6 (BusWires, Rin[1], Clock, R6);
regn reg_7 (BusWires, Rin[0], Clock, R7);

regn reg_A (BusWires, Ain, Clock, A);
alu add_sub(
        .A (A),
        .BusWires (BusWires),
        .AddSub (AddSub),
        .result (result)
);
regn reg_G (result, Gin,Clock,G);

multiplexer mux10(
        .R0(R0), .R1(R1), .R2(R2), .R3(R3),
        .R4(R4), .R5(R5), .R6(R6), .R7(R7),
        .G(G),
        .DIN(DIN),
        .sel(select),
        .BusWires(BusWires)
);

endmodule
```

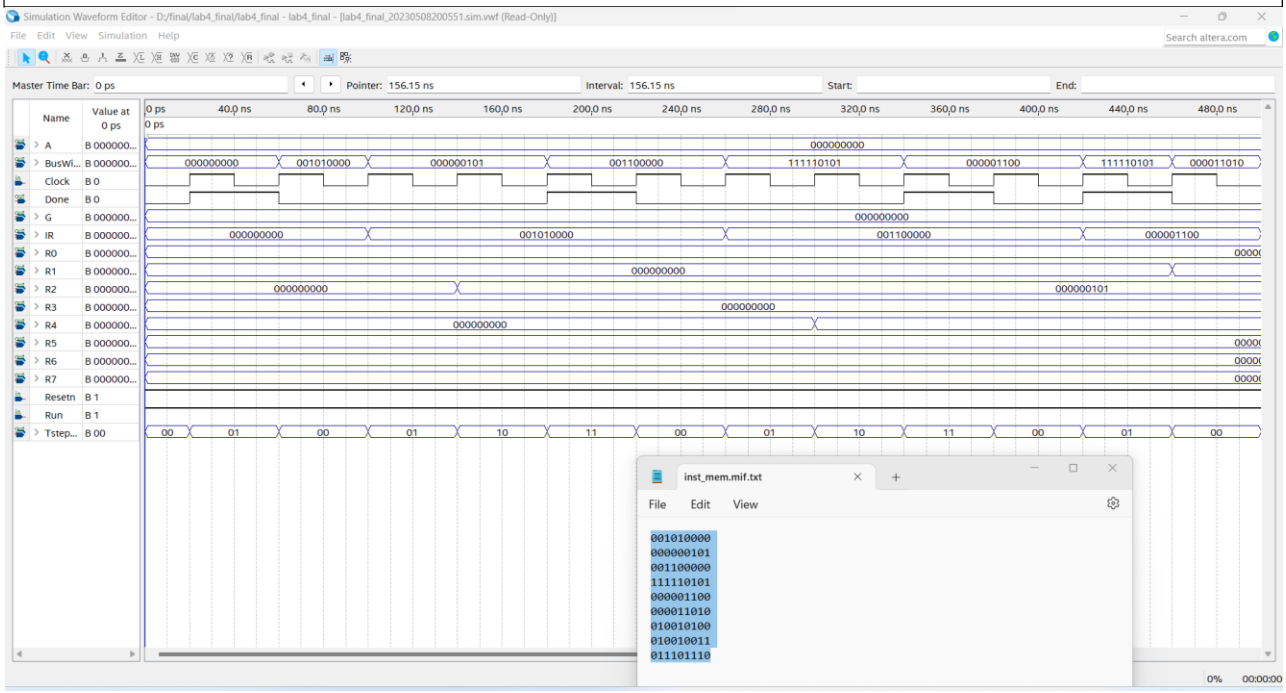
File inst_mem.mif

```
001010000
000000101
001100000
111110101
000001100
000011010
010010100
010010011
011101110
```

- The waveform to prove the circuit works correctly.

Laboratory 4:

A SIMPLE PROCESSOR



- The result of RTL viewer.

Laboratory 4:

A SIMPLE PROCESSOR