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.

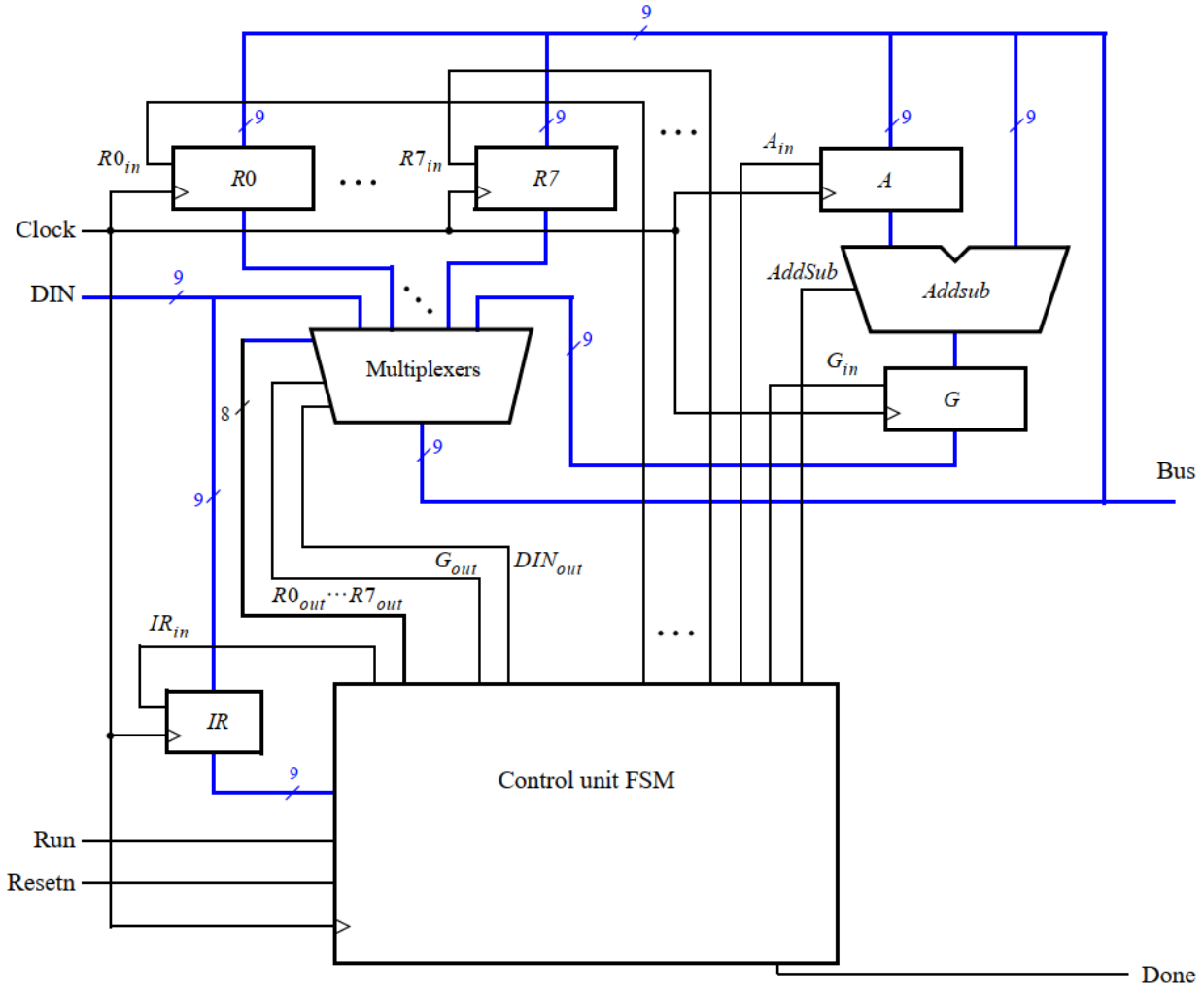

Figure 1: A simple processor.

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.

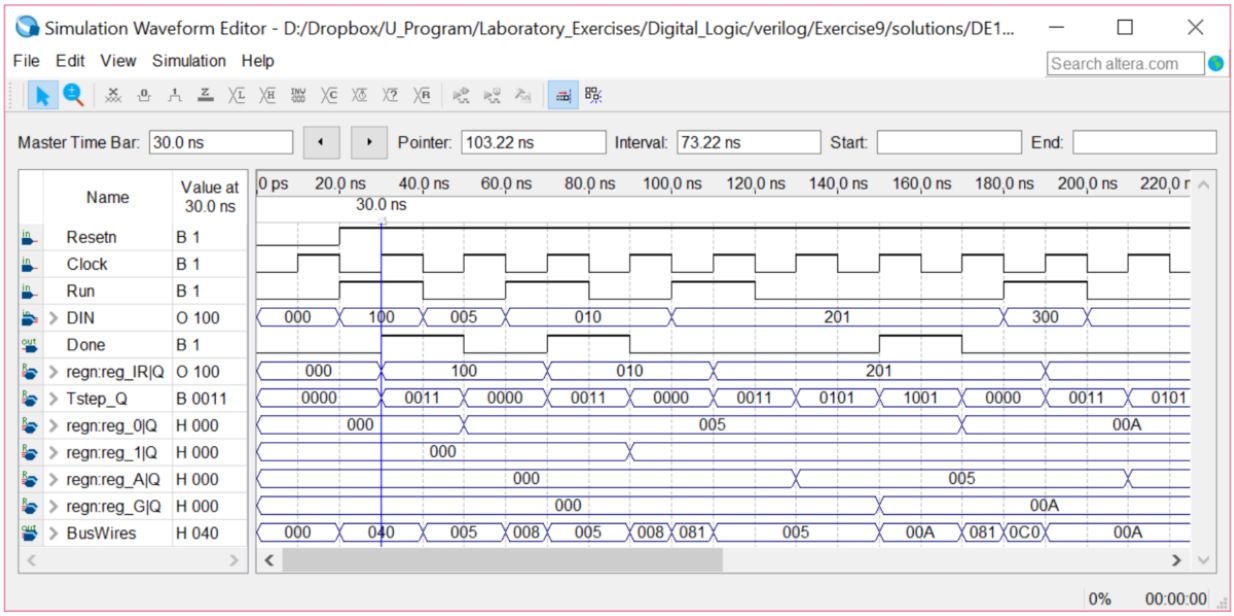

Figure 2: Simulation result for the processor.

Check: Your report has to show two results:

- The waveform to prove the circuit works correctly.
- The result of RTL viewer.

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.

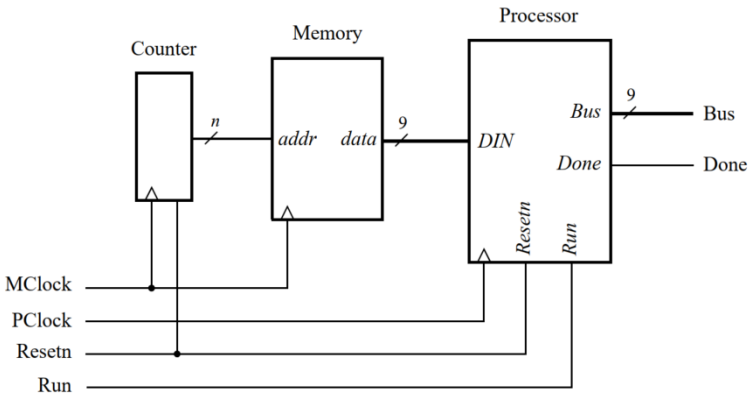

Figure 3: Connecting the processor to a memory and counter.

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.

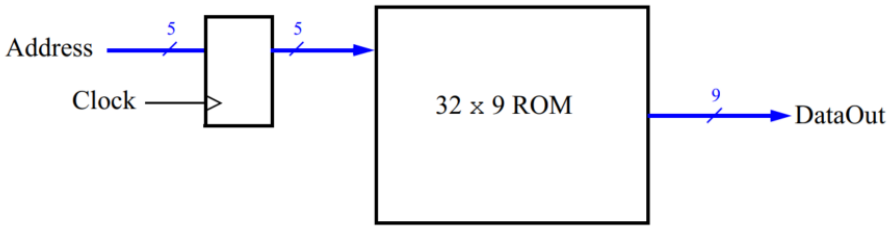

Figure 4: The 32 x 9 ROM with address register.

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

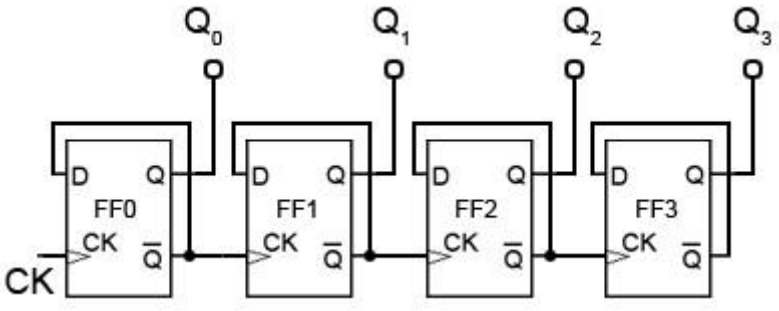

Figure 5: The 5 bit serial counter.

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

Check: Your report has to show two results:

- The waveform to prove the circuit works correctly.
- The result of RTL viewer.