Laboratory 5:

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 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:

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.

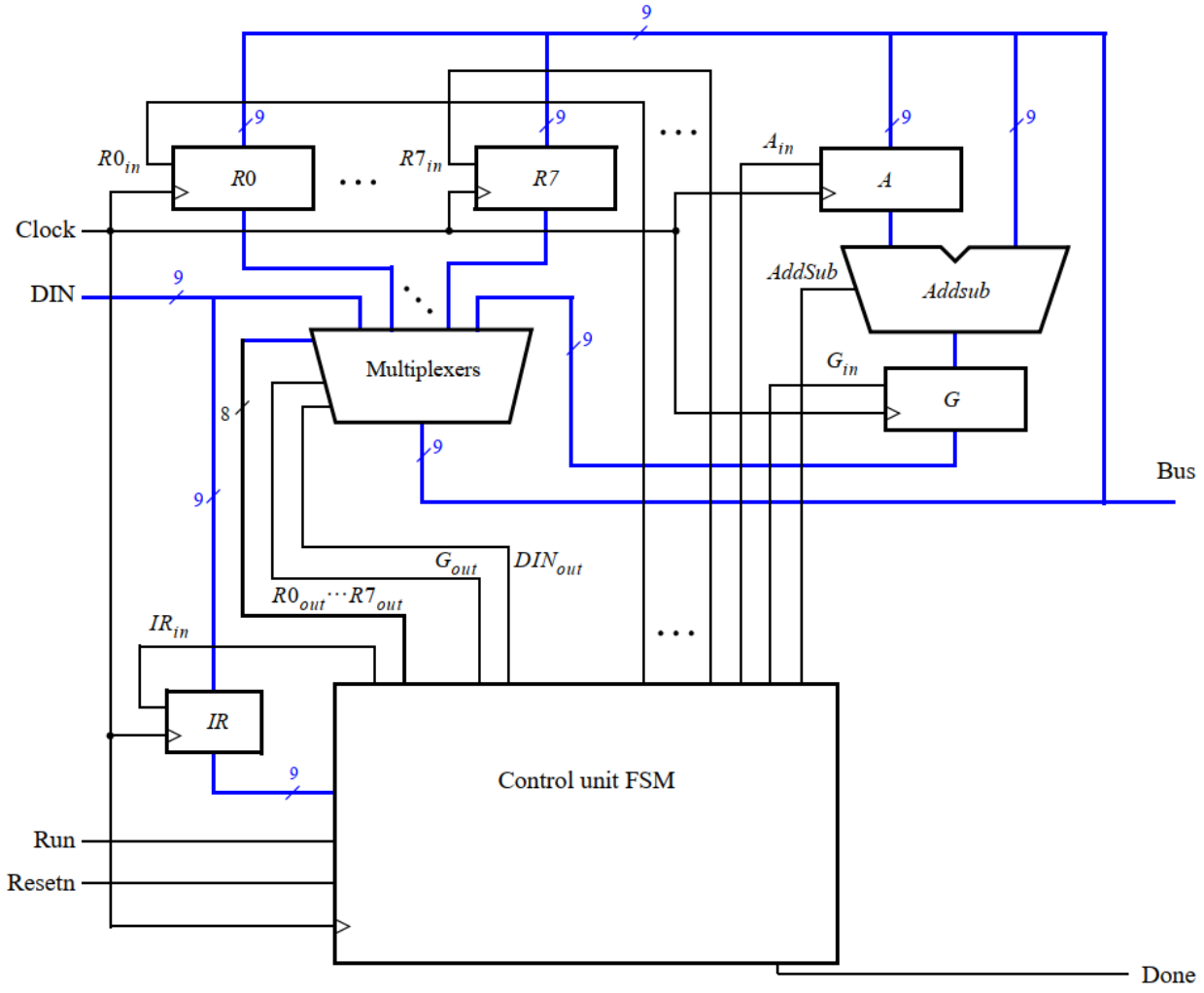

Figure 1: A simple processor.

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 5:

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.

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY proc IS
PORT ( DIN : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
Resetn, Clock, Run : IN STD_LOGIC;
Done : BUFFER STD_LOGIC;
BusWires : BUFFER STD_LOGIC_VECTOR(8 DOWNTO 0));
END proc;

ARCHITECTURE Behavior OF proc IS
… declare components
… declare signals
TYPE State_type IS (T0, T1, T2, T3);
SIGNAL Tstep_Q, Tstep_D: State_type;
…
BEGIN
statetable: PROCESS (Tstep_Q, Run, Done)
BEGIN
  CASE Tstep_Q IS
    WHEN T0 => IF(Run = '0') THEN Tstep_D <= T0;
                  ELSE Tstep_D <= T1;
                  END IF; - - data is loaded into IR in this time step
     ... other states
    END CASE;
END PROCESS;

controlsignals: PROCESS (Tstep_Q, I, Xreg, Yreg)
BEGIN
... specify initial values
  CASE Tstep_Q IS
    WHEN T0 => - - store DIN in IR as long as Tstep_Q = 0
```

```
        IRin <= '1';
    WHEN T1 => - - define signals in time step T1
        CASE I IS
        ...
        END CASE;
   WHEN T2 => - - define signals in time step T2
        CASE I IS
        ...
        END CASE;
    WHEN T3 => - - define signals in time step T3
        CASE I IS
        ...
        END CASE;
  END CASE;
END PROCESS;

fsmflipflops: PROCESS (Clock, Resetn, Tstep_D)
BEGIN
...
END PROCESS;

reg_0: regn PORT MAP (BusWires, Rin(0), Clock, R0);
... instantiate other registers and the adder/subtractor unit
... define the bus
END Behavior;
```

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

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dec3to8 IS
PORT ( W : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
En : IN STD_LOGIC;
Y : OUT STD_LOGIC_VECTOR(0 TO 7));
END dec3to8;
```

```
ARCHITECTURE Behavior OF dec3to8 IS
BEGIN
PROCESS (W, En)
BEGIN
IF En = '1' THEN
CASE W IS
WHEN "000" => Y <= "10000000";
WHEN "001" => Y <= "01000000";
WHEN "010" => Y <= "00100000";
WHEN "011" => Y <= "00010000";
WHEN "100" => Y <= "00001000";
WHEN "101" => Y <= "00000100";
WHEN "110" => Y <= "00000010";
WHEN "111" => Y <= "00000001";
END CASE;
ELSE
Y <= "00000000";
END IF;
END PROCESS;
END Behavior;
```

- 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.

Laboratory 5:

A SIMPLE PROCESSOR

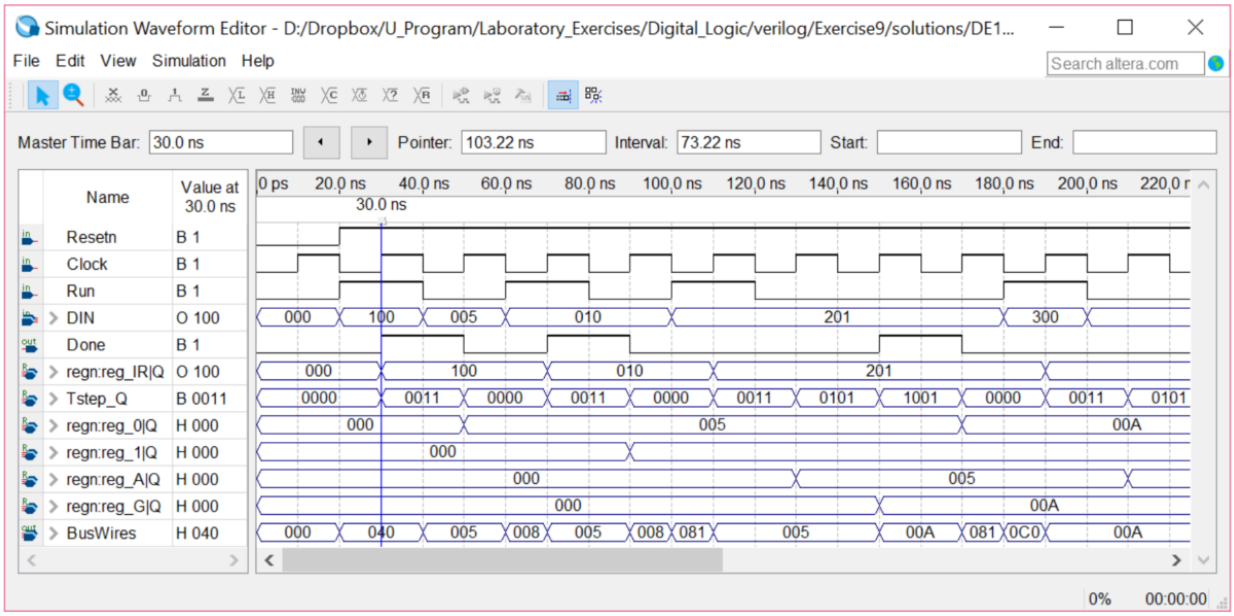

Figure 2: Simulation result for the processor.

Check: Your report has to show two results:

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

Laboratory 5:

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.

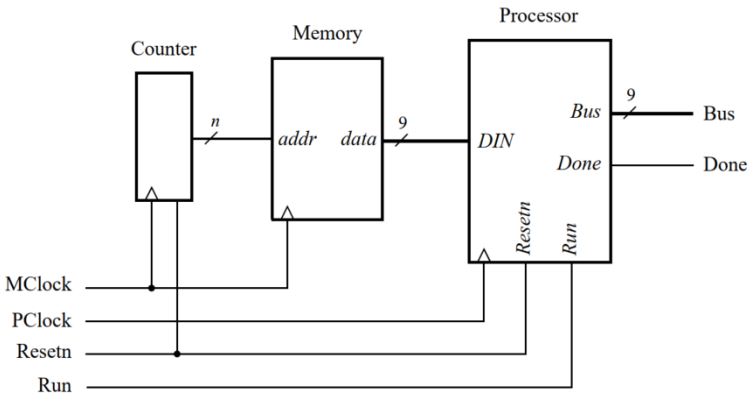

Figure 3: Connecting the processor to a memory and counter.

Instruction:

- A diagram of the memory module that we need to create is depicted in Figure 4. The VHDL code for this module is prepared in exercise 3, pre lab 5.

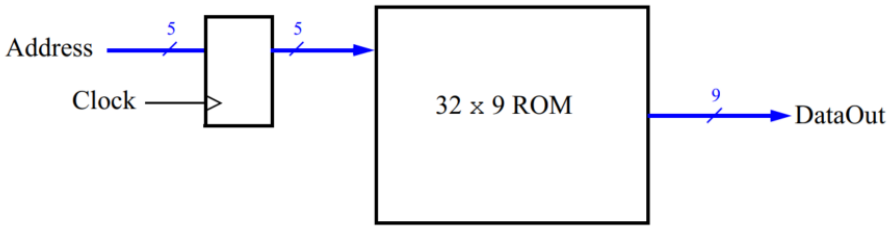

Figure 4: The 32 x 9 ROM with address register.

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

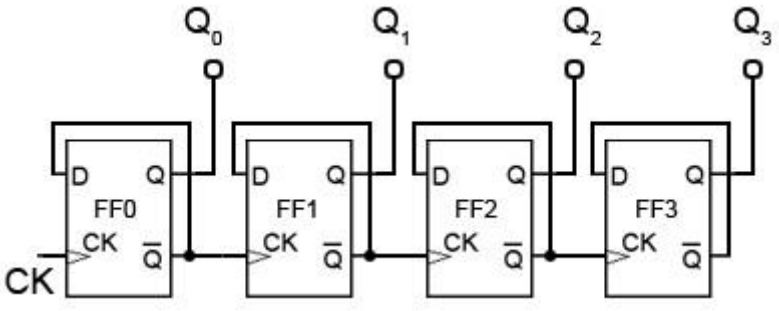

Figure 5: The 5 bit serial counter.

- 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.