Date:25/4/2022

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Fall Semester 2021-2022

ECE5014 – ASIC Design

M.Tech VLSI Design

School of Electronics Engineering

Vellore Institute of Technology

Name: Shreyas S Bagi

Register Number: 21MVD0086

Slot: L3+L4

Date:25/4/2022

Lab Task 01

Design Specification and Architecture Design of Simple Processor

Section 1 Block Diagram

Aim: Write the Architecture Design of Simple Processor.

Specification:

A digital system that contains a number of 9-bit registers, a multiplexer, an adder/subtractor unit, and a control unit (finite state machine). Data is input to this system via the 9-bit DIN input. This data can be loaded through the 9-bit wide multiplexer into the various registers, such as R0,….R7 and A. The multiplexer also allows data to be transferred from one register to another. The multiplexer's output wires are called a bus in the figure because this term is often used for wiring that allows data to be transferred from one location in a system to another. Addition or subtraction is performed by using the multiplexer to first place one 9-bit number onto the bus wires and loading this number into register A. Once this is done, a second 9-bit number is placed onto the bus, the adder/subtractor unit performs the required operation, and the result is loaded into register G. The data in G can then be transferred to one of the other registers as required.

The system can perform different operations in each clock cycle, as governed by the control unit. This unit determines when particular data is placed onto the bus wires and it controls which of the registers is to be loaded with this data? For example, if the control unit asserts the signals R0out and Ain, then the multiplexer will place the contents of register R0 onto the bus and this data will be loaded by the next active clock edge into register A. A system like this is often called a processor. It executes operations specified in the form of instructions.

Table below lists the instructions that the processor has to support for this exercise. The left column shows the name of an instruction and its operand. The meaning of the syntax Rx [Ry] is that the contents of register Ry are loaded into register Rx. The mv (move) instruction allows data to be copied from one register to another. For the mvi (move immediate) instruction the expression Rx D indicates that the 9-bit constant D is loaded into register Rx.

Operation	Function performed
mv Rx,Ry	$Rx \leftarrow [Ry]$
mvi Rx,#D	$Rx \leftarrow D$
add Rx, Ry	$Rx \leftarrow [Rx] + [Ry]$
sub Rx, Ry	$Rx \leftarrow [Rx] - [Ry]$

Table 1: Instructions performed in the processor.

Each instruction can be encoded and stored in the IR register using the 9-bit format IIIXXXYYY, where III represents the instruction, XXX gives the Rx register, and YYY gives the Ry register. Although only two bits are needed to encode our four instructions, we are using three bits because other instructions will be added to the processor in later parts of this exercise. Instructions are loaded from an external source; hence IR has to be connected to the nine bits of the DIN input, as indicated in Figure 1. For the mvi instruction the YYY field has no meaning, and the immediate data #D has to be supplied on the 9-bit DIN input after the mvi instruction word is stored into IR. Some instructions, such as an addition or subtraction, take more than one clock cycle to complete, because multiple transfers have to be performed across the bus. The finite state machine in the control unit "steps through" such instructions, asserting the control signals needed in successive clock cycles until the instruction has completed. The processor starts executing the instruction on the DIN input when the Run signal is asserted and the processor asserts the Done output when the instruction is finished.

Table 2 indicates the control signals that can be asserted in each time

	T_1	T_2	T_3
(mv): I_0	RY_{out}, RX_{in}, *Done*		
(mvi): I_1	DIN_{out}, RX_{in}, *Done*		
(add): I_2	RX_{out}, A_{in}	RY_{out}, G_{in}	G_{out}, RX_{in}, *Done*
(sub): I_3	RX_{out}, A_{in}	RY_{out}, G_{in}, *AddSub*	G_{out}, RX_{in}, *Done*

Table 2: Control signals asserted in each instruction/time step.

Date:25/4/2022

Architecture Design of Simple Processor:

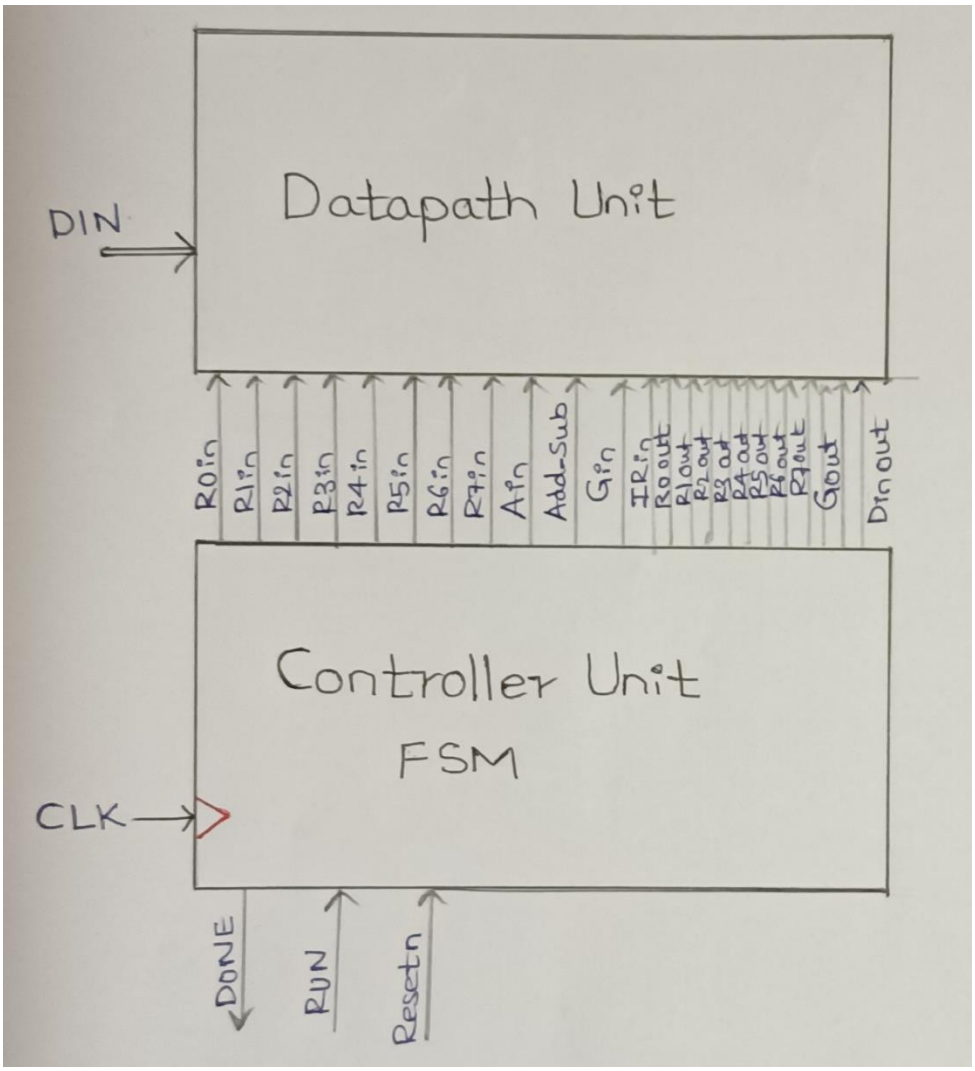

Figure 1.1 The Block diagram of Simple processor showing Datapath and Control path block.

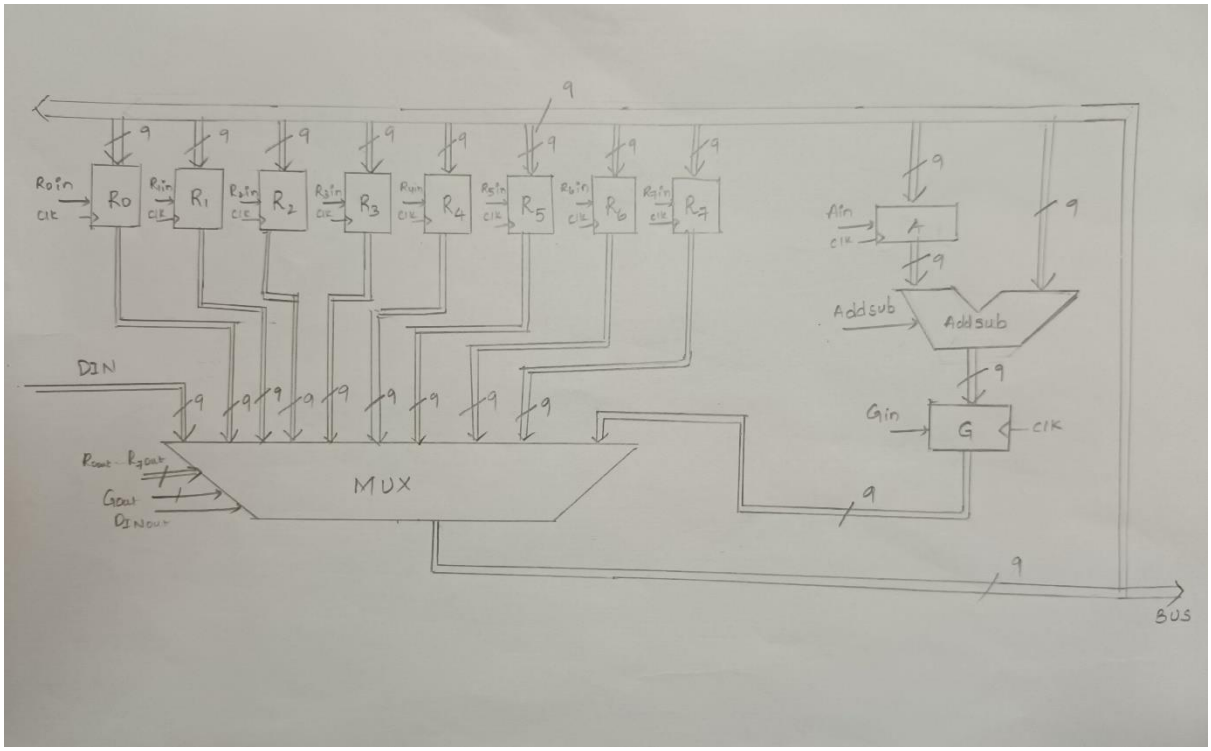

Figure 1.2 The Block diagram of Simple processor showing Datapath block.

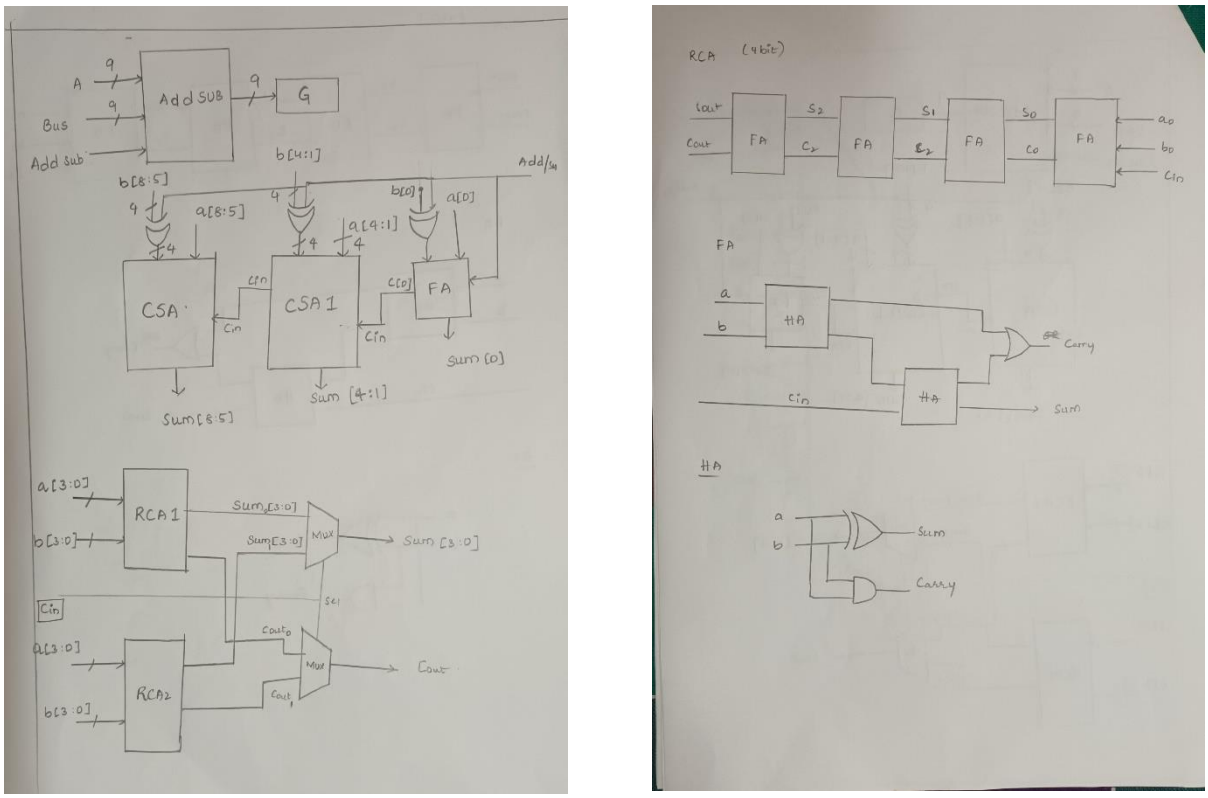

Figure 1.3 The Block diagram of Adder and Subtractor block.

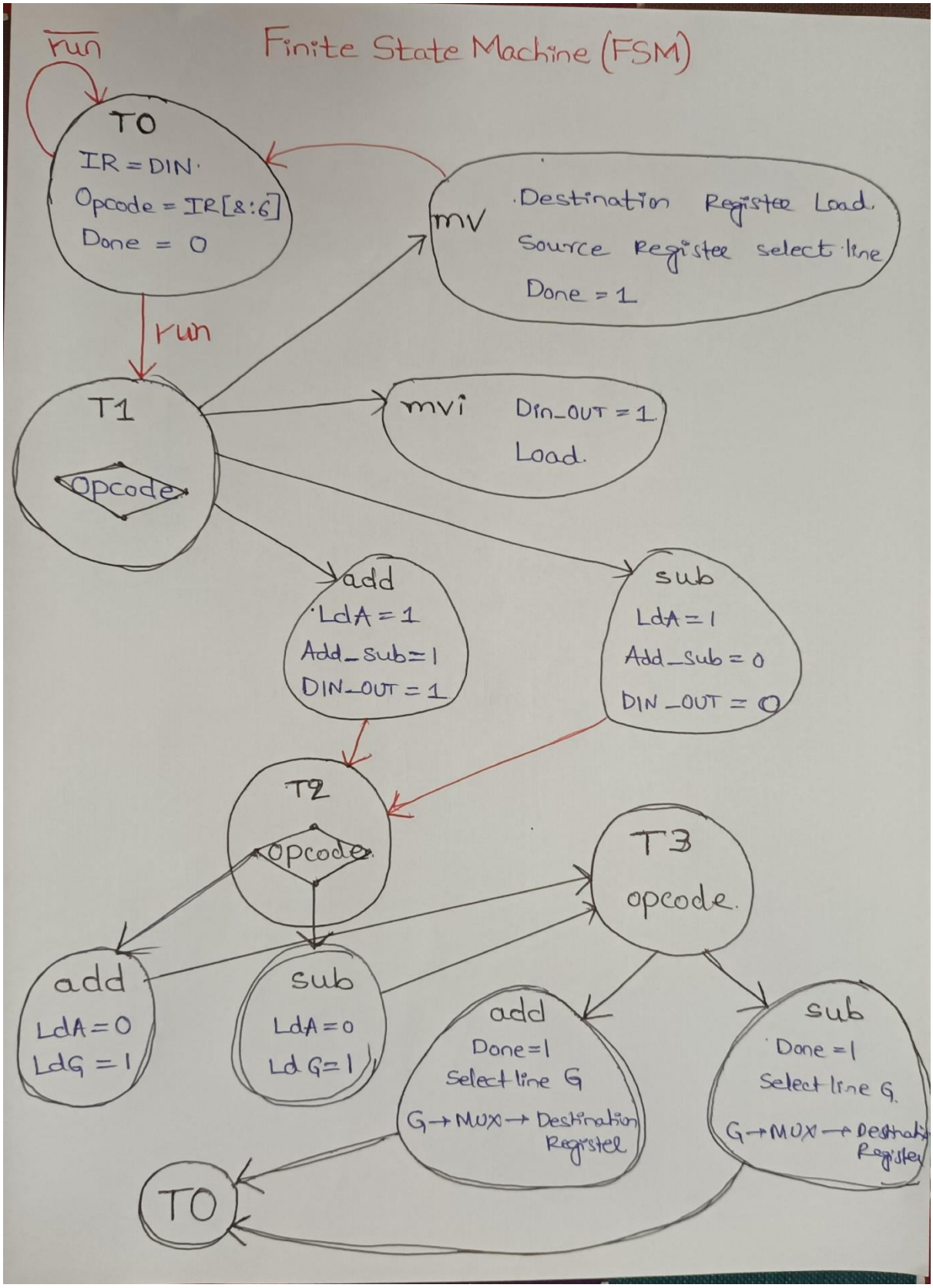

Figure 1.4 The Finite State Machine for Controller Block.

Date:25/4/2022

RTL Coding:

1. **Control path Verilog Code**

2. **Datapath Verilog Code**

3. **Top Module Code**

Testbench Codes based on Operations

1. **Move Immediate Operation**

```
`timescale 1ns/1ns
module simple_processor_testbench ();
reg Run, Resetn, Clock;
reg [8:0] DIN;
wire [8:0] Bus;
wire Done;

controller_new G1 (DIN, Run, Resetn, Clock, Done, R0out, R1out,
R2out, R3out, R4out, R5out,
R6out, R7out, Gout, DINout, LdR0, LdR1, LdR2, LdR3, LdR4, LdR5,
LdR6, LdR7, LdA, LdG, Add_sub);

datapath_register_array G2 (R0out, R1out, R2out, R3out, R4out, R5out,
R6out, R7out, Gout, DINout, Clock, Resetn, LdR0,
LdR1, LdR2, LdR3, LdR4, LdR5, LdR6, LdR7, LdA, Bus, DIN,
Add_sub, LdG);

initial
Clock = 1'b1;

always #5 Clock = ~Clock;
initial
begin

$set_toggle_region(simple_processor_testbench.G1,
simple_processor_testbench.G2);
$toggle_start();
// ...

Resetn = 1'b0;
#10 Resetn = 1'b1;
Run = 1'b1;

DIN = 9'b011000001; // mvi operation
#20 DIN = 9'b111001111; // immediate data
//R0 is loaded with 111001111;
#5;

#20 DIN = 9'b011010001; // mvi operation
#20 DIN = 9'b111111111; // immediate data after 50ns from 1st instuction
@t=10
// R2 is loaded with 111111111;
```

```
#30 DIN = 9'b011001001; // mvi operation
#20 DIN = 9'b101010101; // immediate data
//R1 is loaded with 101010101;
#5;

#30 DIN = 9'b011011001; // mvi operation
#20 DIN = 9'b101010111; // immediate data
//R3 is loaded with 101010101;
#5;

#30 DIN = 9'b011100001; // mvi operation
#20 DIN = 9'b111101111; // immediate data
//R4 is loaded with 101010101;
#5;

#30 DIN = 9'b011101001; // mvi operation
#20 DIN = 9'b110110110; // immediate data
//R5 is loaded with 101010101;
#5;

#30 DIN = 9'b011110001; // mvi operation
#20 DIN = 9'b111011011; // immediate data
//R6 is loaded with 101010101;
#5;

#30 DIN = 9'b011111001; // mvi operation
#20 DIN = 9'b111111111; // immediate data
//R7 is loaded with 101010101;

// ...
$toggle_stop();
$toggle_report("Simple_Processor_SAIF_Move_Immediate_Operation.
saif", 1.0e-12, "simple_processor_testbench");
#5; $stop;

Run = 1'b0;
end

initial
 begin

/*
// mvi R1,R5
#2 DIN = 9'b011001010; // mvi operation
#30 DIN = 9'b111001111; // immediate data
```

```
Run = 0;
#10 Run = 1;
DIN = 9'b000001010;
#1000 $stop;
*/

end
endmodule
```

Date:25/4/2022

Waveforms:

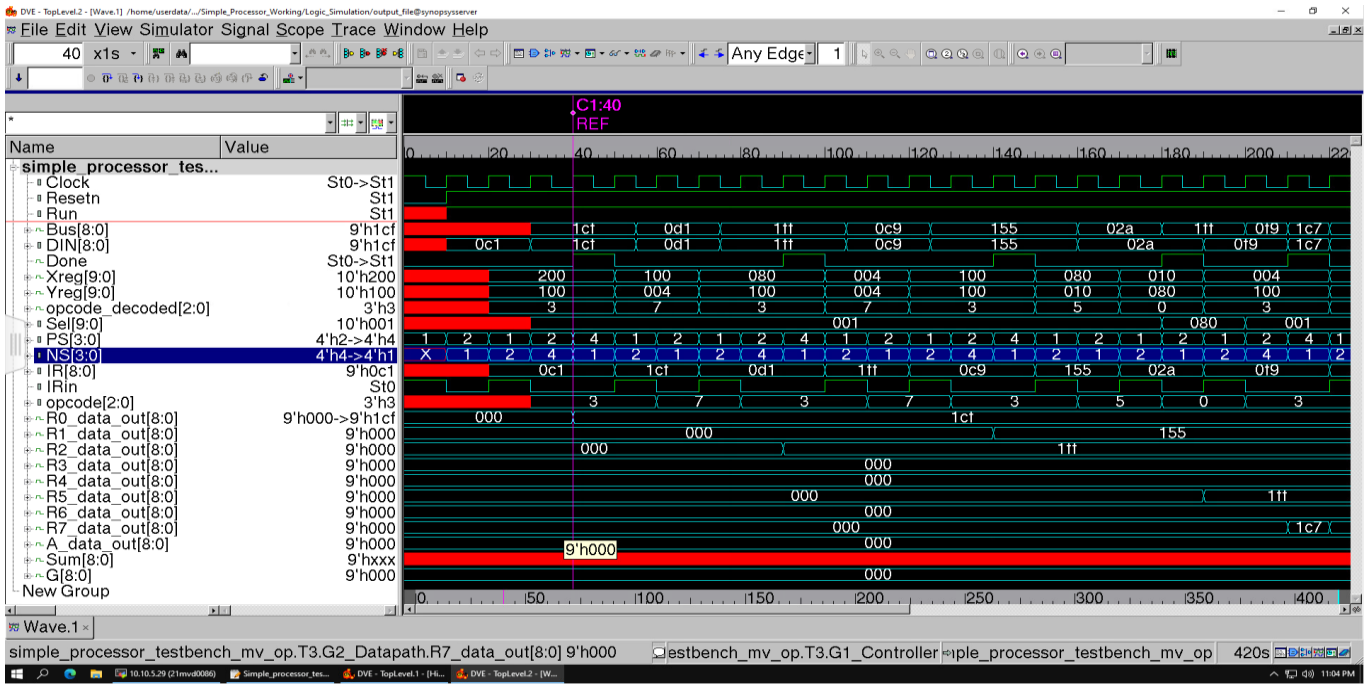

Figure 1.1: The data stored in Register and Move Immediate Operation.

In the Figure 1.1 we are doing Move immediate operation. At time instant from t=0ns to 40ns we observe that Register R0 is stored with 0 value and after 40ns we store the data 9'h1CF data. The flow of data is like this at 10ns the (2nd Clock pulse), Resetn = 1 and Run = 1. These conditions are met then only start the operation of processor. The Move immediate operation has three states T0 to T1 to T2. In these states the operation occurs. At time t=40ns we observe R0 changes data. Similarly In Figure 1.2 we observe the move operation taking place for R2 register and R1 register.

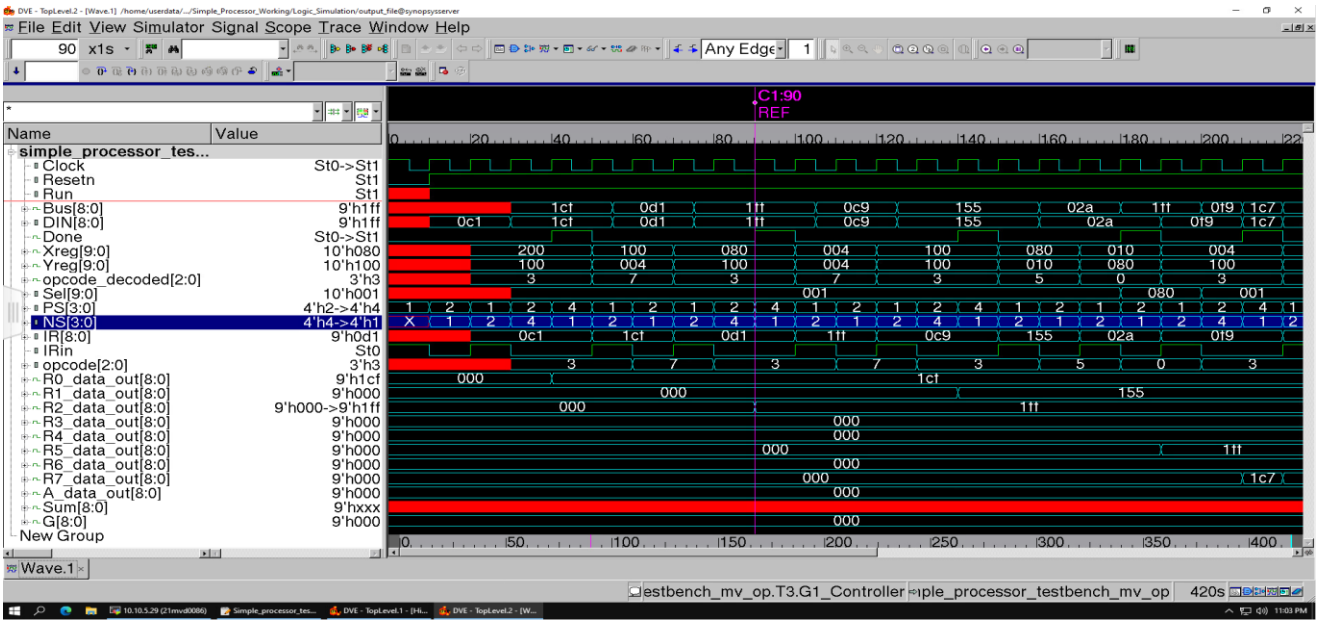

Figure 1.2: The data stored in Register and Move Immediate Operation.

Date:25/4/2022

Inference:

1. The Simple Processor 9 bit was designed using Verilog code. The simulation was done using Synopsys VCS tool.
2. The opcodes used were Move, Move immediate, Addition and Subtraction operation was done and verified.