VELLORE INSTITUTE OF TECHNOLOGY, VELLORE-632014
(Deemed to be University under section 3 of UGC Act, 1956)

J Component Makethon Report on

"Design of Simple Processor"

Batch No : 3

MASTER OF TECHNOLOGY
in
VLSI DESIGN
FALL SEMESTER 2021-2022

Submitted by

Panchagnula Krishna Vamsidhar	21MVD0085
Shreyas S Bagi	21MVD0086
Pragya	21MVD0093
Suresh Bhati	21MVD0111
K Naga Sai Swaroop	21MVD0135

Under the guidance of

Dr. Nitish Kumar V
Associate Professor
School of Electronics Engineering(SENSE)
VIT, Vellore - 632014

SCHOOL OF ELECTRONICS ENGINEERING (SENSE)
2021-22

Chapter 1

Problem Statement and Problem Description

Design of Simple Processor Figure below shows 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.

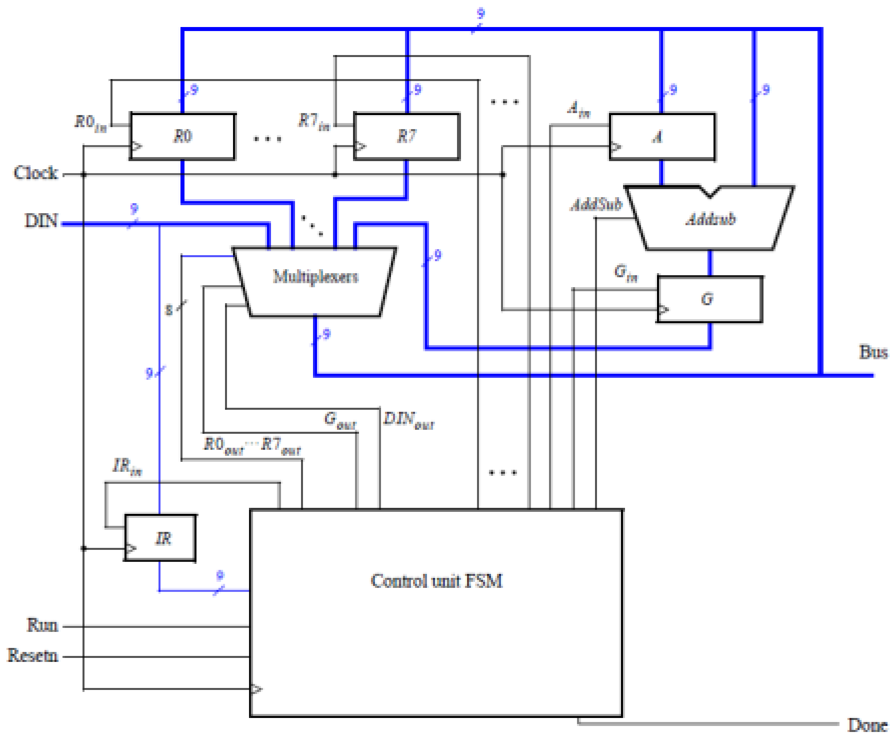


Figure 1.1 The Processor Block Diagram which includes Datapath and Control block separately.

The IR Register is of 9bit value and is loaded from DIN ate T0 state only.

IR[8:6]	IR[5:3]	IR[2:0]

The IR[8:6] is encode with 000 has MOVE operation, 001 has ADD operation, 010 Subtraction operation and 011 Move Immediate operation.

The IR[5:3] and IR[2:0] is having encoding scheme like

000	R0 register
001	R1 register
010	R2 register
011	R3 register
100	R4 register
101	R5 register
110	R6 register
111	R7 register

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	$Rx \leftarrow [Rx] - [Ry]$

Figure 1.2 Operation of the Instruction

	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*

Figure 1.3 Control Signals asserted in each Instruction/time step

Chapter 2

Block Diagram

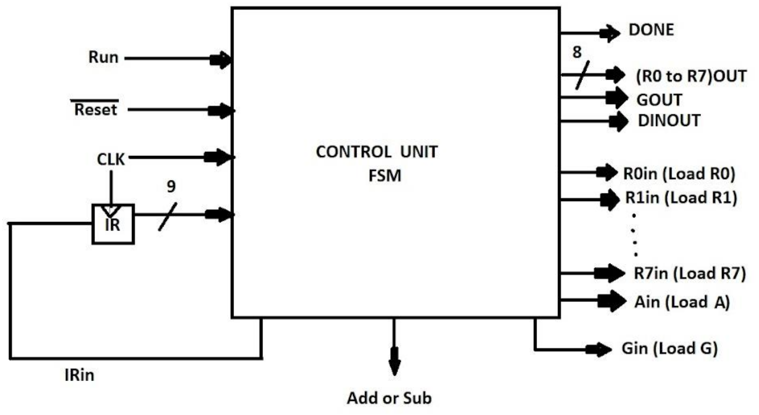


Figure 2.1 Operation of the Instruction : Block Diagram with respect to Control Unit FSM.

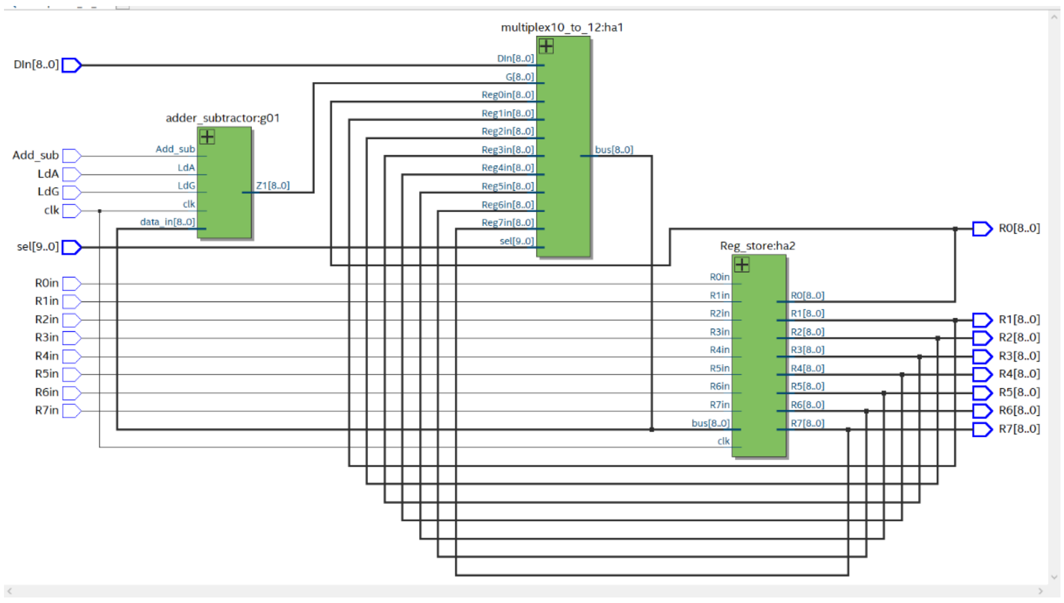
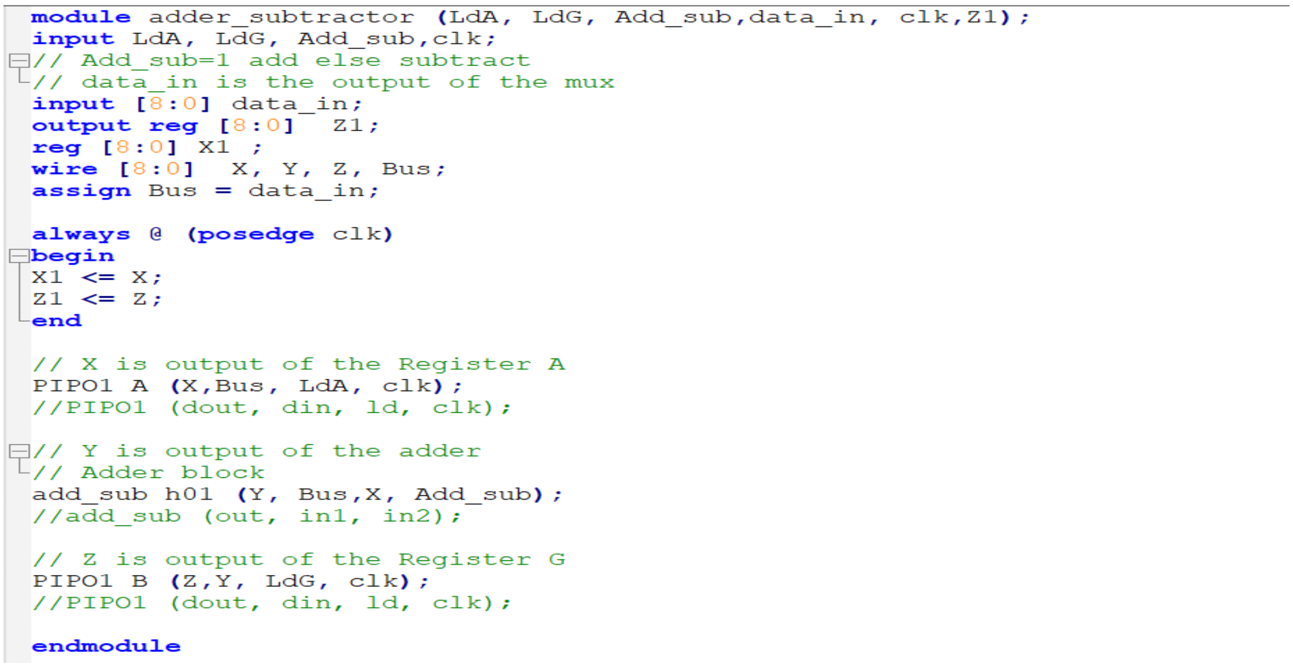


Figure 2.2 Block Diagram with respect to Datapath Unit.

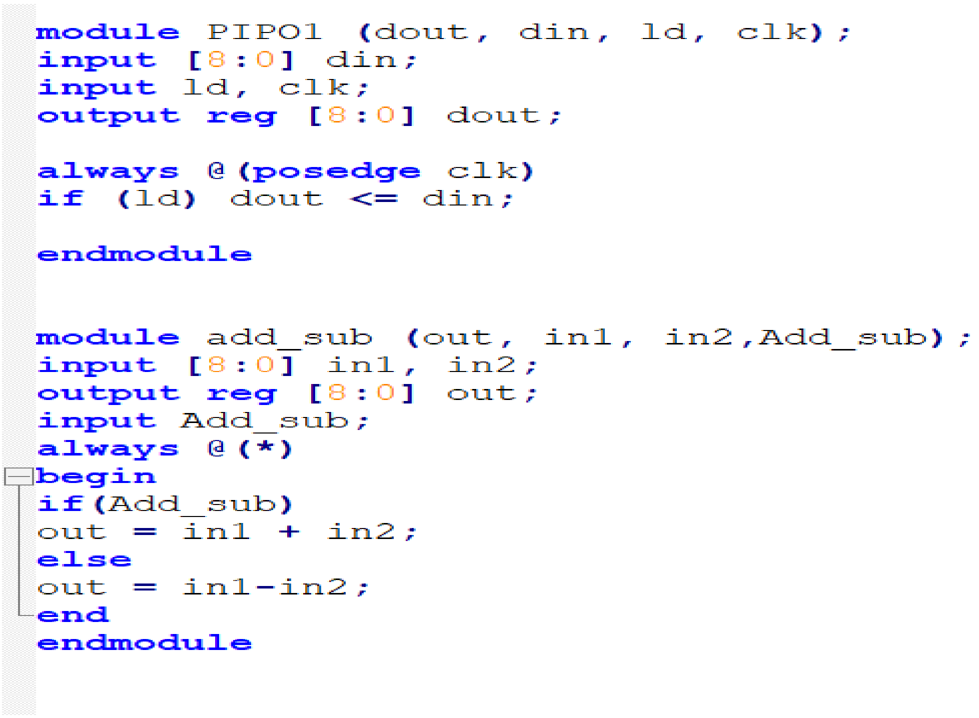
The Datapath path consists of Mux, Adder or Subtractor and Control signal (like enable signals) from the Control Block FSM.

Verilog Code for Datapath Unit :

1. Adder or Subtractor Submodule

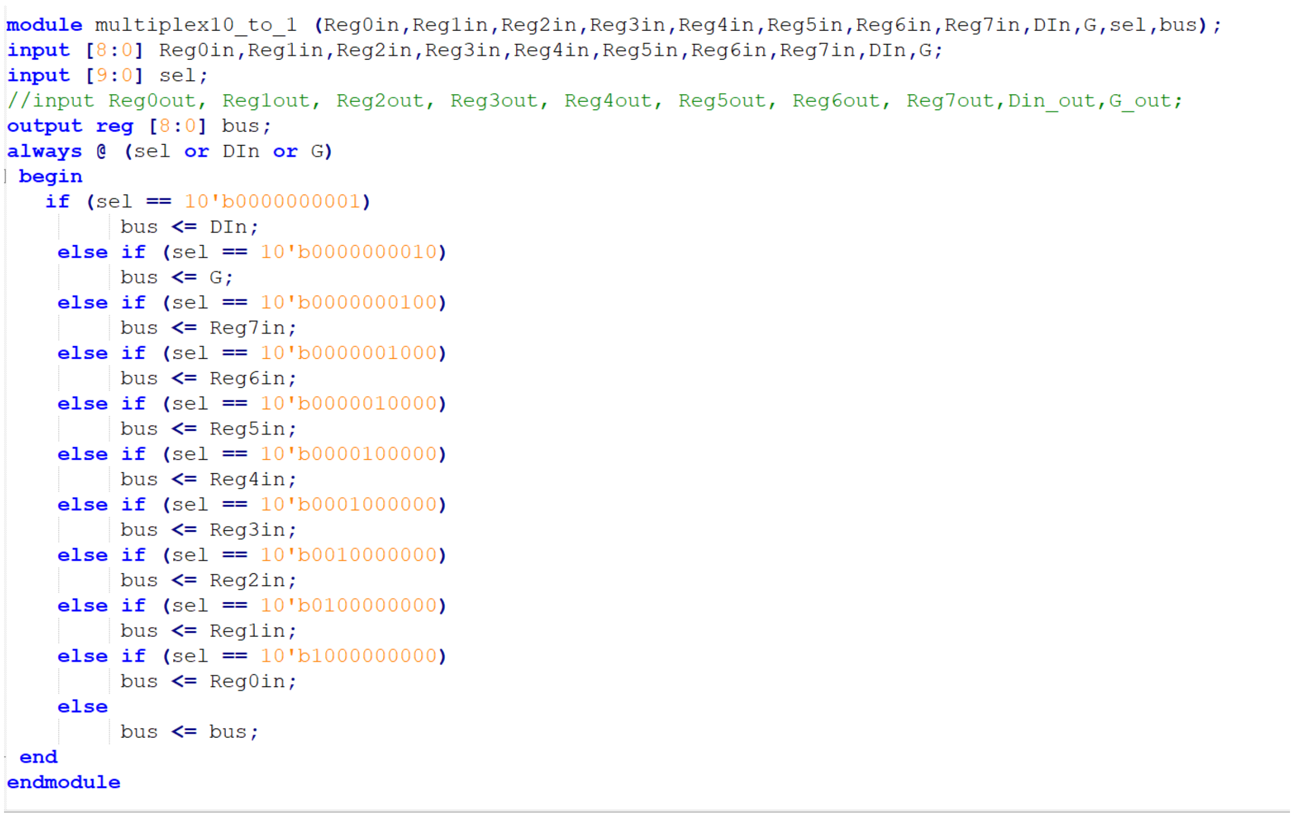


```
module adder_subtractor (LdA, LdG, Add_sub,data_in, clk,Z1);
input LdA, LdG, Add_sub,clk;
// Add_sub=1 add else subtract
// data_in is the output of the mux
input [8:0] data_in;
output reg [8:0]  Z1;
reg [8:0] X1  ;
wire [8:0]  X, Y, Z, Bus;
assign Bus = data_in;

always @ (posedge clk)
begin
X1 <= X;
Z1 <= Z;
end

// X is output of the Register A
PIPO1 A (X,Bus, LdA, clk);
//PIPO1 (dout, din, ld, clk);

// Y is output of the adder
// Adder block
add_sub h01 (Y, Bus,X, Add_sub);
//add_sub (out, in1, in2);

// Z is output of the Register G
PIPO1 B (Z,Y, LdG, clk);
//PIPO1 (dout, din, ld, clk);

endmodule
```



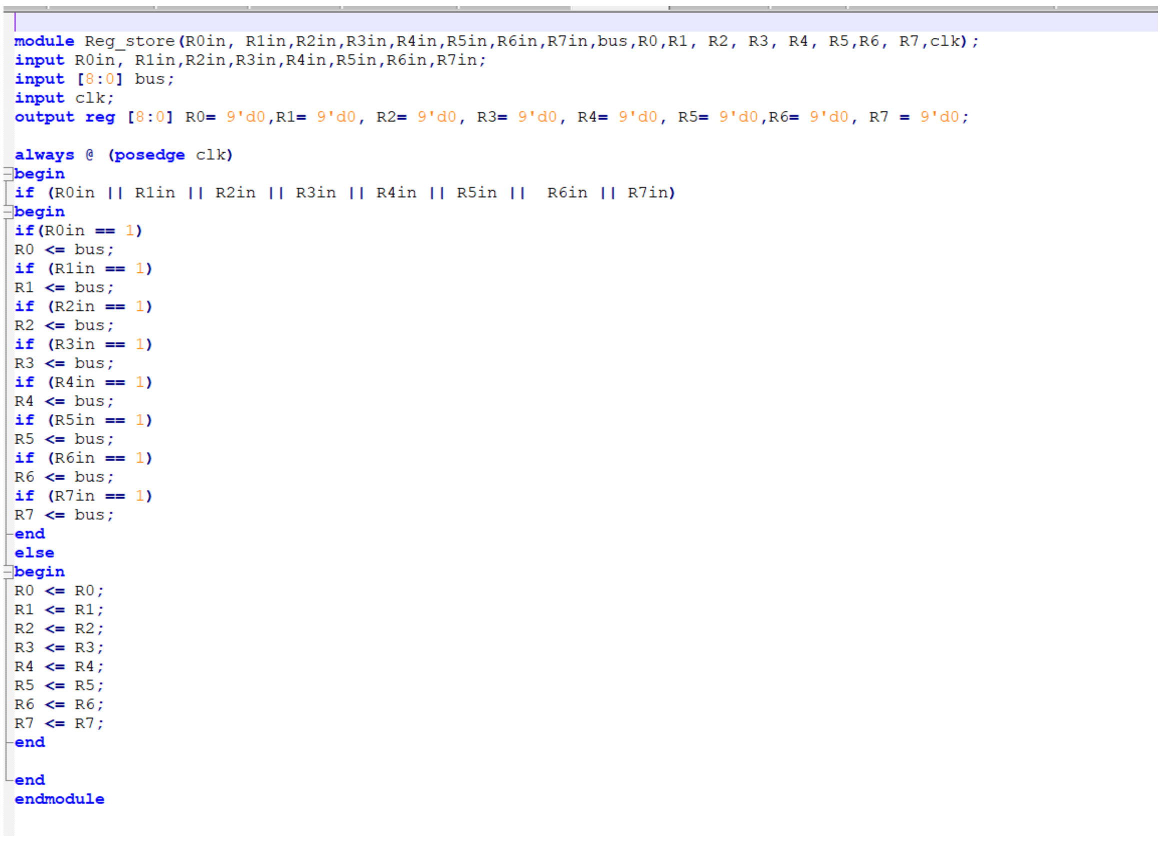
```
module PIPO1 (dout, din, ld, clk);
input [8:0] din;
input ld, clk;
output reg [8:0] dout;

always @(posedge clk)
if (ld) dout <= din;

endmodule


module add_sub (out, in1, in2,Add_sub);
input [8:0] in1, in2;
output reg [8:0] out;
input Add_sub;
always @(*)
begin
if(Add_sub)
out = in1 + in2;
else
out = in1-in2;
end
endmodule
```

Figure 2.3 Adder or Subtractor Submodule



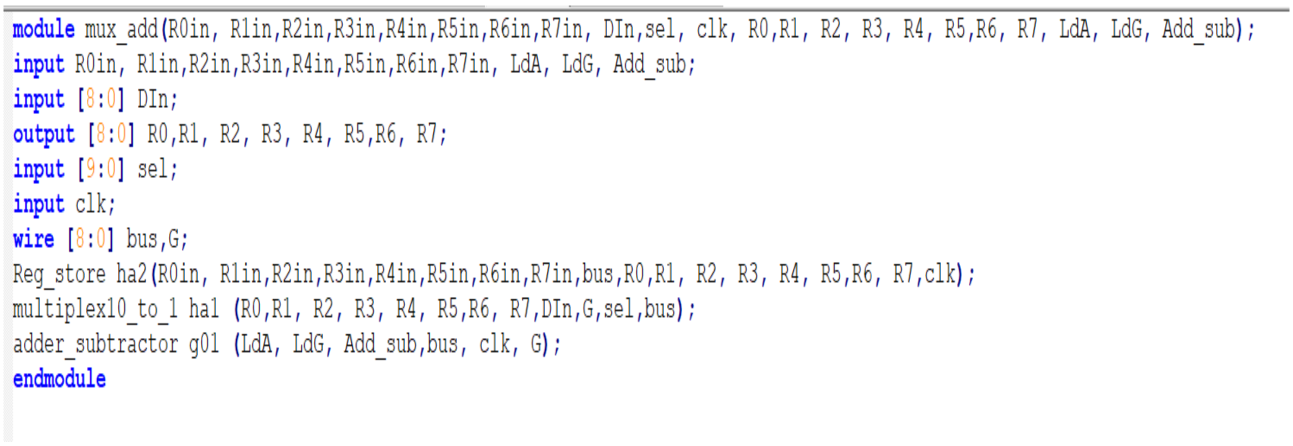
```verilog
module multiplex10_to_1 (Reg0in,Reg1in,Reg2in,Reg3in,Reg4in,Reg5in,Reg6in,Reg7in,DIn,G,sel,bus);
input [8:0] Reg0in,Reg1in,Reg2in,Reg3in,Reg4in,Reg5in,Reg6in,Reg7in,DIn,G;
input [9:0] sel;
//input Reg0out, Reg1out, Reg2out, Reg3out, Reg4out, Reg5out, Reg6out, Reg7out,Din_out,G_out;
output reg [8:0] bus;
always @ (sel or DIn or G)
 begin
   if (sel == 10'b0000000001)
        bus <= DIn;
    else if (sel == 10'b0000000010)
        bus <= G;
    else if (sel == 10'b0000000100)
        bus <= Reg7in;
    else if (sel == 10'b0000001000)
        bus <= Reg6in;
    else if (sel == 10'b0000010000)
        bus <= Reg5in;
    else if (sel == 10'b0000100000)
        bus <= Reg4in;
    else if (sel == 10'b0001000000)
        bus <= Reg3in;
    else if (sel == 10'b0010000000)
        bus <= Reg2in;
    else if (sel == 10'b0100000000)
        bus <= Reg1in;
    else if (sel == 10'b1000000000)
        bus <= Reg0in;
    else
        bus <= bus;
 end
endmodule
```

Figure 2.4 Multiplexer Module



```verilog
module Reg_store(R0in, R1in,R2in,R3in,R4in,R5in,R6in,R7in,bus,R0,R1, R2, R3, R4, R5,R6, R7,clk);
input R0in, R1in,R2in,R3in,R4in,R5in,R6in,R7in;
input [8:0] bus;
input clk;
output reg [8:0] R0= 9'd0,R1= 9'd0, R2= 9'd0, R3= 9'd0, R4= 9'd0, R5= 9'd0,R6= 9'd0, R7 = 9'd0;

always @ (posedge clk)
begin
if (R0in || R1in || R2in || R3in || R4in || R5in ||  R6in || R7in)
begin
if(R0in == 1)
R0 <= bus;
if (R1in == 1)
R1 <= bus;
if (R2in == 1)
R2 <= bus;
if (R3in == 1)
R3 <= bus;
if (R4in == 1)
R4 <= bus;
if (R5in == 1)
R5 <= bus;
if (R6in == 1)
R6 <= bus;
if (R7in == 1)
R7 <= bus;
end
else
begin
R0 <= R0;
R1 <= R1;
R2 <= R2;
R3 <= R3;
R4 <= R4;
R5 <= R5;
R6 <= R6;
R7 <= R7;
end

end
endmodule
```

Figure 2.5 Register Module



```
module mux_add(R0in, R1in,R2in,R3in,R4in,R5in,R6in,R7in, DIn,sel, clk, R0,R1, R2, R3, R4, R5,R6, R7, LdA, LdG, Add_sub);
input R0in, R1in,R2in,R3in,R4in,R5in,R6in,R7in, LdA, LdG, Add_sub;
input [8:0] DIn;
output [8:0] R0,R1, R2, R3, R4, R5,R6, R7;
input [9:0] sel;
input clk;
wire [8:0] bus,G;
Reg_store ha2(R0in, R1in,R2in,R3in,R4in,R5in,R6in,R7in,bus,R0,R1, R2, R3, R4, R5,R6, R7,clk);
multiplex10_to_1 ha1 (R0,R1, R2, R3, R4, R5,R6, R7,DIn,G,sel,bus);
adder_subtractor g01 (LdA, LdG, Add_sub,bus, clk, G);
endmodule
```

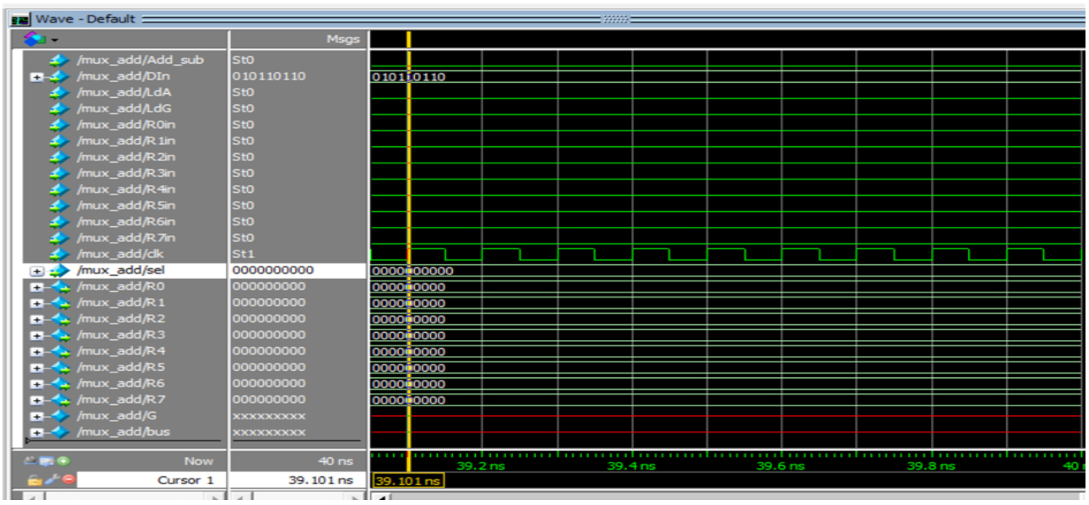
Figure 2.6 Datapath Module with Control signals and Input for verifying the data path.

Simulation Output in Modelsim dor the above operation

Note:

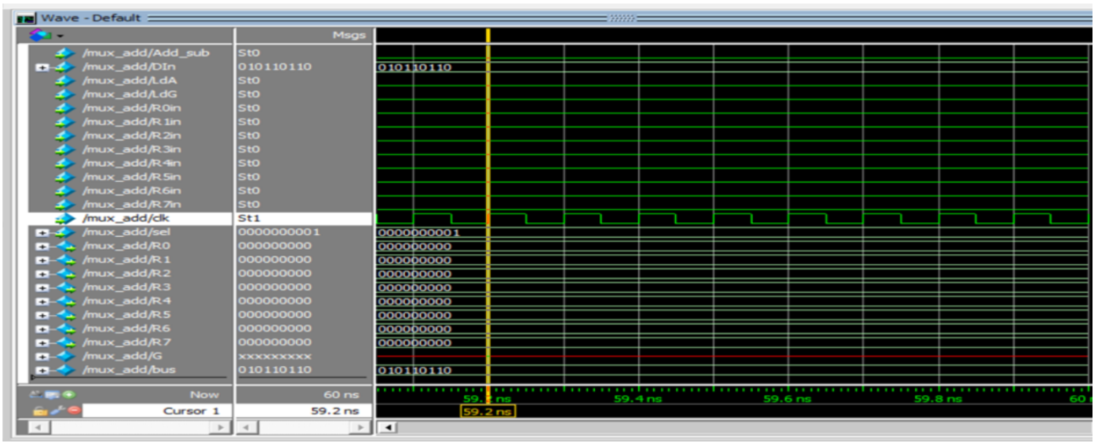
1. R0in to R7in are enable signals for registers Reg0 to Reg7;
2. Sel is a 10-bit input which represents selection inputs for the Multiplexer which represents control signals Sel[9:0]={Regout0, Regout1, Regout2, Regout3, Regout4, Regout5, Regout6, Regout7, Gout, Din(Data in) } respectively ;
3. R0 to R7 represents registers Reg0 to Reg7 & G represents Output Register of Alu, Bus register represents Data Bus;
4. LDA & LDG are enable control signals for A, G registers respectively, and clk represents clock signal;
5. Add_Sub is the control signal for adder/subtractor unit:
   a. For Addition ⇒Add_Sub=1
   b. For Subtraction ⇒Add_Sub=0

Datapath Verification:



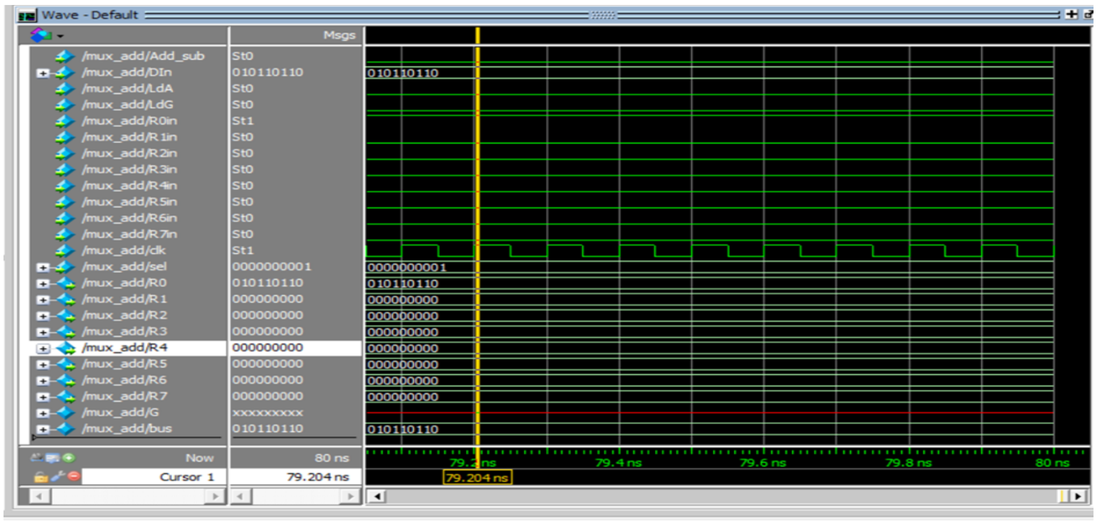
```
                         Msgs
/mux_add/Add_sub         St0
/mux_add/DIn             010110110
/mux_add/LdA             St0
/mux_add/LdG             St0
/mux_add/R0in            St0
/mux_add/R1in            St0
/mux_add/R2in            St0
/mux_add/R3in            St0
/mux_add/R4in            St0
/mux_add/R5in            St0
/mux_add/R6in            St0
/mux_add/R7in            St0
/mux_add/clk             St1
/mux_add/sel             0000000000
/mux_add/R0              000000000
/mux_add/R1              000000000
/mux_add/R2              000000000
/mux_add/R3              000000000
/mux_add/R4              000000000
/mux_add/R5              000000000
/mux_add/R6              000000000
/mux_add/R7              000000000
/mux_add/G               xxxxxxxxx
/mux_add/bus             xxxxxxxxx
Now                      40 ns
Cursor 1                 39.101 ns
```

Figure 2.7 Data taking



```
                         Msgs
/mux_add/Add_sub         St0
/mux_add/DIn             010110110
/mux_add/LdA             St0
/mux_add/LdG             St0
/mux_add/R0in            St0
/mux_add/R1in            St0
/mux_add/R2in            St0
/mux_add/R3in            St0
/mux_add/R4in            St0
/mux_add/R5in            St0
/mux_add/R6in            St0
/mux_add/R7in            St0
/mux_add/clk             St1
/mux_add/sel             0000000001
/mux_add/R0              000000000
/mux_add/R1              000000000
/mux_add/R2              000000000
/mux_add/R3              000000000
/mux_add/R4              000000000
/mux_add/R5              000000000
/mux_add/R6              000000000
/mux_add/R7              000000000
/mux_add/G               xxxxxxxxx
/mux_add/bus             010110110
Now                      60 ns
Cursor 1                 59.2 ns
```

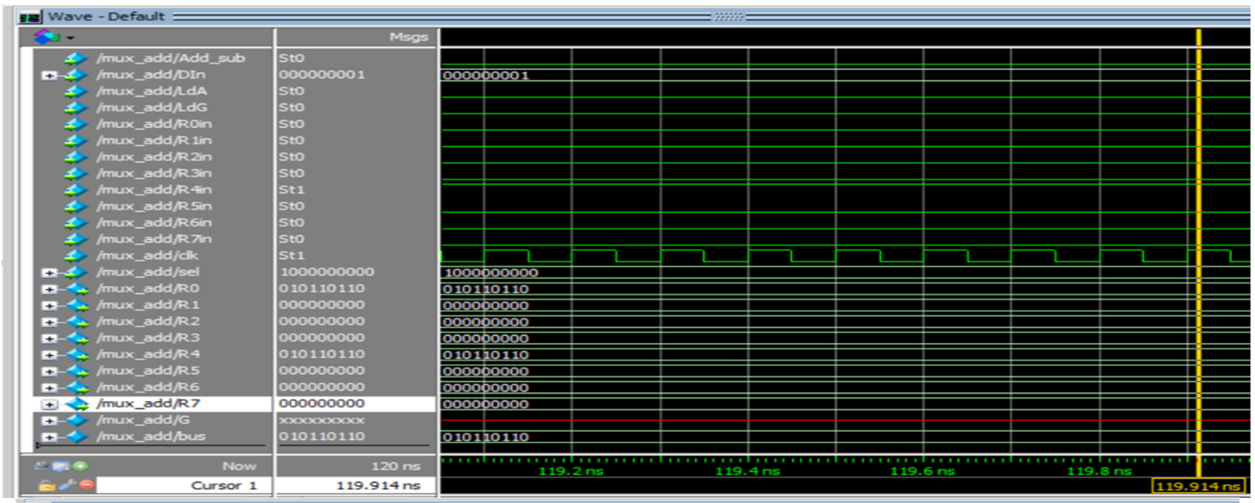
Figure 2.8 Assigning to the Bus



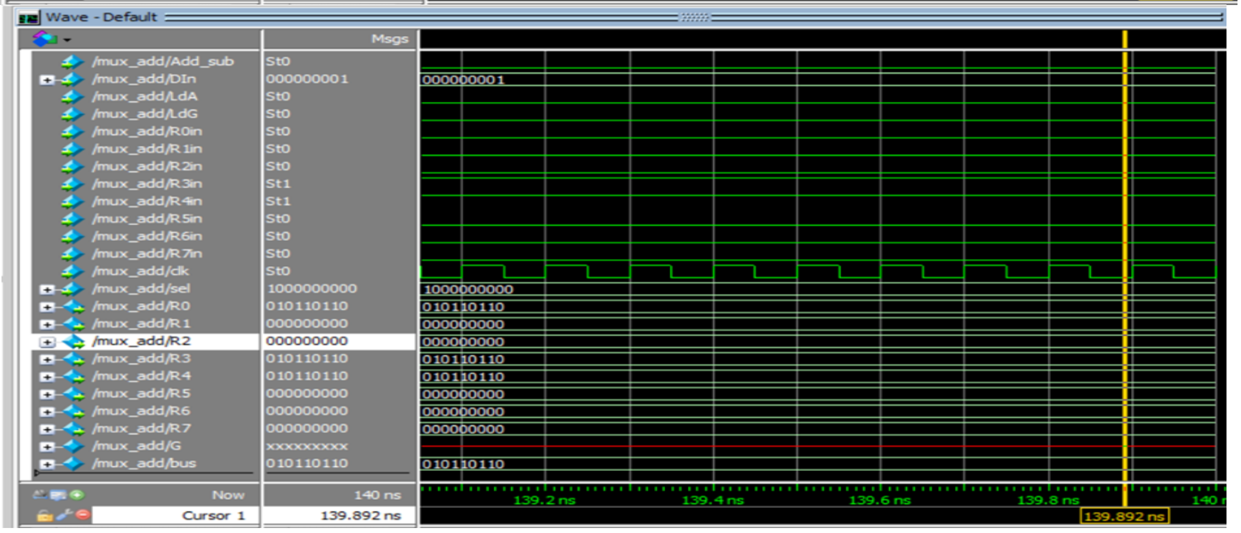
```
                         Msgs
/mux_add/Add_sub         St0
/mux_add/DIn             010110110
/mux_add/LdA             St0
/mux_add/LdG             St0
/mux_add/R0in            St1
/mux_add/R1in            St0
/mux_add/R2in            St0
/mux_add/R3in            St0
/mux_add/R4in            St0
/mux_add/R5in            St0
/mux_add/R6in            St0
/mux_add/R7in            St0
/mux_add/clk             St1
/mux_add/sel             0000000001
/mux_add/R0              010110110
/mux_add/R1              000000000
/mux_add/R2              000000000
/mux_add/R3              000000000
/mux_add/R4              000000000
/mux_add/R5              000000000
/mux_add/R6              000000000
/mux_add/R7              000000000
/mux_add/G               xxxxxxxxx
/mux_add/bus             010110110
Now                      80 ns
Cursor 1                 79.204 ns
```

Figure 2.9 MVI Register

The IR Register is of 9bit value and is loaded from DIN ate T0 state only.

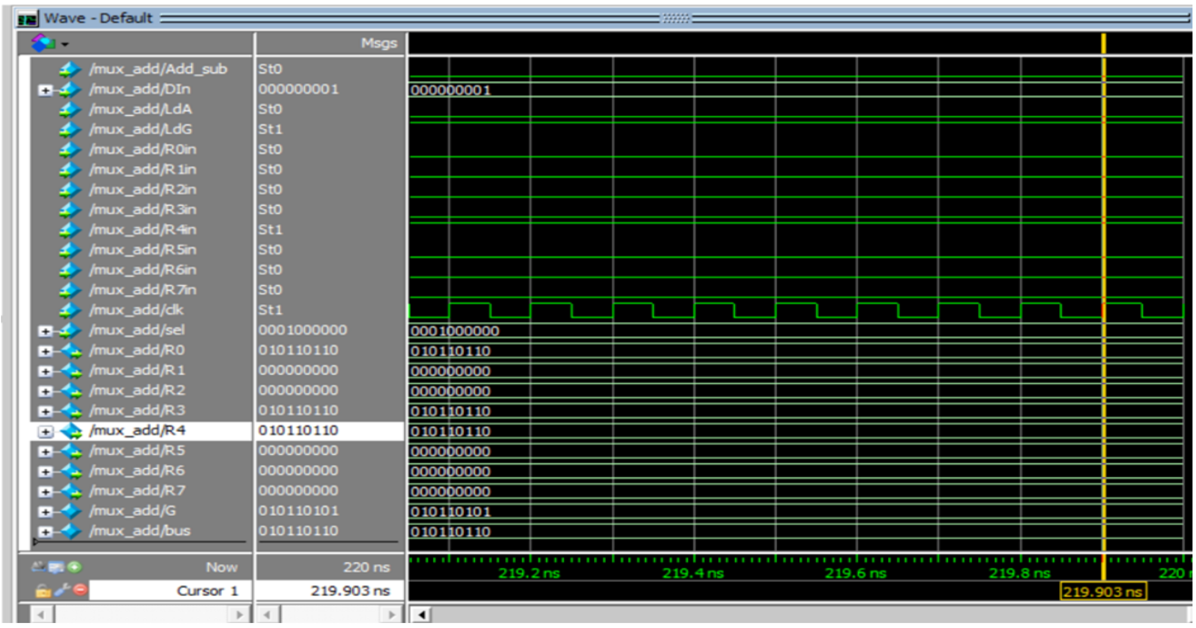


```
Wave - Default
                    Msgs
/mux_add/Add_sub    St0
/mux_add/DIn        000000001
/mux_add/LdA        St0
/mux_add/LdG        St0
/mux_add/R0in       St0
/mux_add/R1in       St0
/mux_add/R2in       St0
/mux_add/R3in       St0
/mux_add/R4in       St1
/mux_add/R5in       St0
/mux_add/R6in       St0
/mux_add/R7in       St0
/mux_add/clk        St1
/mux_add/sel        1000000000
/mux_add/R0         010110110
/mux_add/R1         000000000
/mux_add/R2         000000000
/mux_add/R3         000000000
/mux_add/R4         010110110
/mux_add/R5         000000000
/mux_add/R6         000000000
/mux_add/R7         000000000
/mux_add/G          xxxxxxxxx
/mux_add/bus        010110110
Now                 120 ns
Cursor 1            119.914 ns
119.2 ns   119.4 ns   119.6 ns   119.8 ns   119.914 ns
```



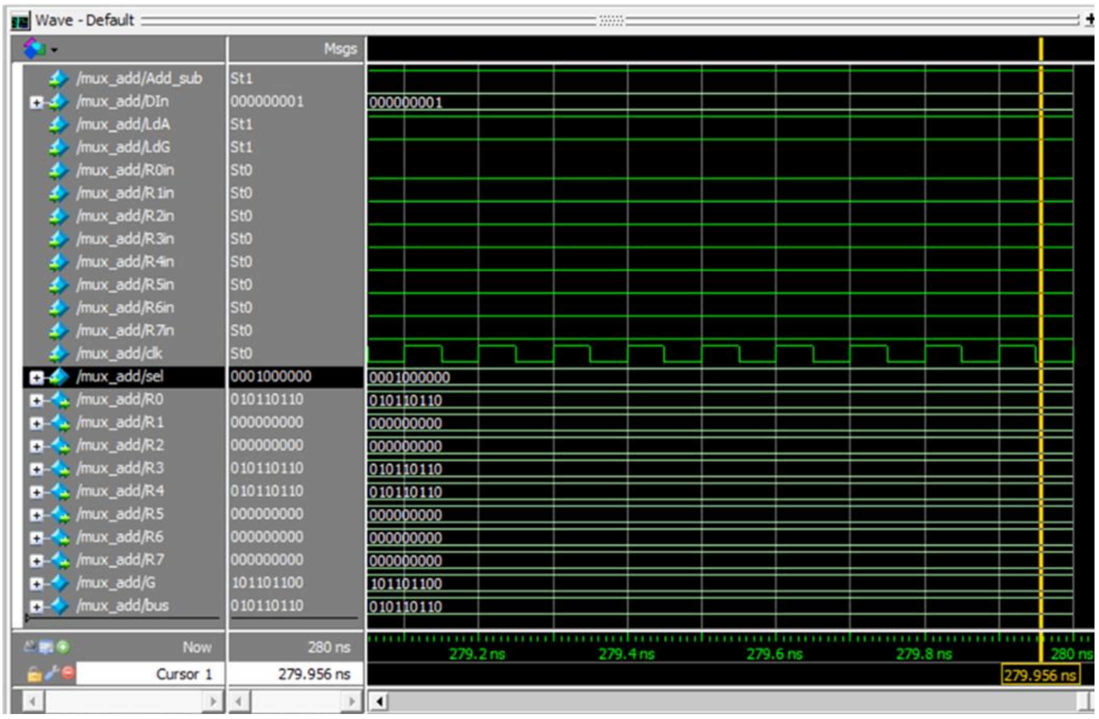
```
Wave - Default
                    Msgs
/mux_add/Add_sub    St0
/mux_add/DIn        000000001
/mux_add/LdA        St0
/mux_add/LdG        St0
/mux_add/R0in       St0
/mux_add/R1in       St0
/mux_add/R2in       St0
/mux_add/R3in       St1
/mux_add/R4in       St1
/mux_add/R5in       St0
/mux_add/R6in       St0
/mux_add/R7in       St0
/mux_add/clk        St0
/mux_add/sel        1000000000
/mux_add/R0         010110110
/mux_add/R1         000000000
/mux_add/R2         000000000
/mux_add/R3         010110110
/mux_add/R4         010110110
/mux_add/R5         000000000
/mux_add/R6         000000000
/mux_add/R7         000000000
/mux_add/G          xxxxxxxxx
/mux_add/bus        010110110
Now                 140 ns
Cursor 1            139.892 ns
139.2 ns   139.4 ns   139.6 ns   139.8 ns   140   139.892 ns
```

Figure 2.10 Moving between R0 and R3



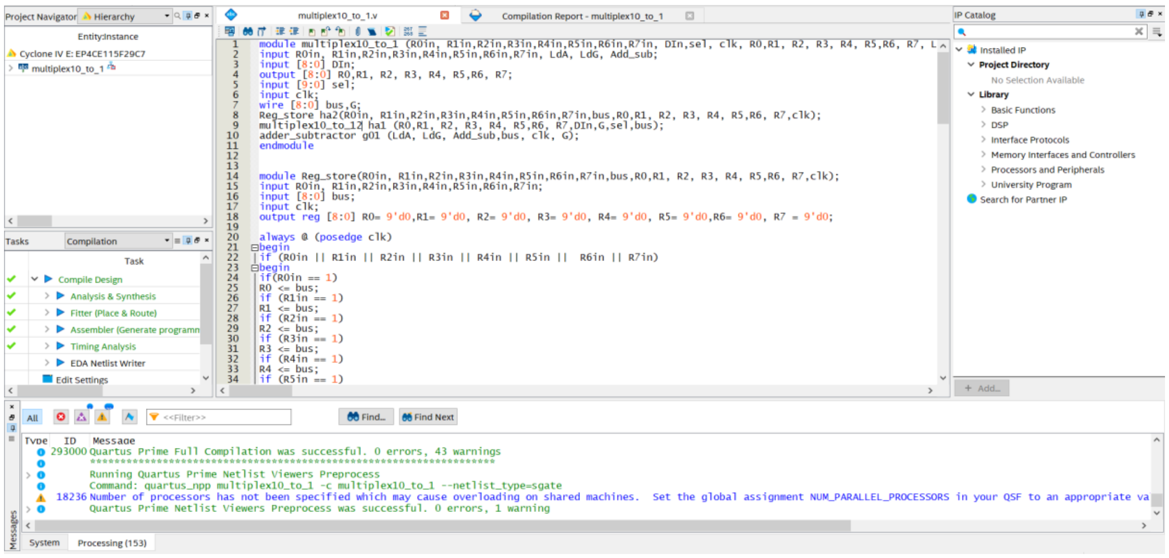
```
Wave - Default
                    Msgs
/mux_add/Add_sub    St0
/mux_add/DIn        000000001
/mux_add/LdA        St0
/mux_add/LdG        St1
/mux_add/R0in       St0
/mux_add/R1in       St0
/mux_add/R2in       St0
/mux_add/R3in       St0
/mux_add/R4in       St1
/mux_add/R5in       St0
/mux_add/R6in       St0
/mux_add/R7in       St0
/mux_add/clk        St1
/mux_add/sel        0001000000
/mux_add/R0         010110110
/mux_add/R1         000000000
/mux_add/R2         000000000
/mux_add/R3         010110110
/mux_add/R4         010110110
/mux_add/R5         000000000
/mux_add/R6         000000000
/mux_add/R7         000000000
/mux_add/G          010110101
/mux_add/bus        010110110
Now                 220 ns
Cursor 1            219.903 ns
219.2 ns   219.4 ns   219.6 ns   219.8 ns   220   219.903 ns
```

Figure2.11 Subtracting R3 and R4 and storing in R4



```
Wave - Default
                      Msgs
/mux_add/Add_sub      St1
/mux_add/DIn          000000001
/mux_add/LdA          St1
/mux_add/LdG          St1
/mux_add/R0in         St0
/mux_add/R1in         St0
/mux_add/R2in         St0
/mux_add/R3in         St0
/mux_add/R4in         St0
/mux_add/R5in         St0
/mux_add/R6in         St0
/mux_add/R7in         St0
/mux_add/clk          St0
/mux_add/sel          0001000000
/mux_add/R0           010110110
/mux_add/R1           000000000
/mux_add/R2           000000000
/mux_add/R3           010110110
/mux_add/R4           010110110
/mux_add/R5           000000000
/mux_add/R6           000000000
/mux_add/R7           000000000
/mux_add/G            101101100
/mux_add/bus          010110110
Now                   280 ns
Cursor 1              279.956 ns
```

Figure 2.12 Adding R3 and R4

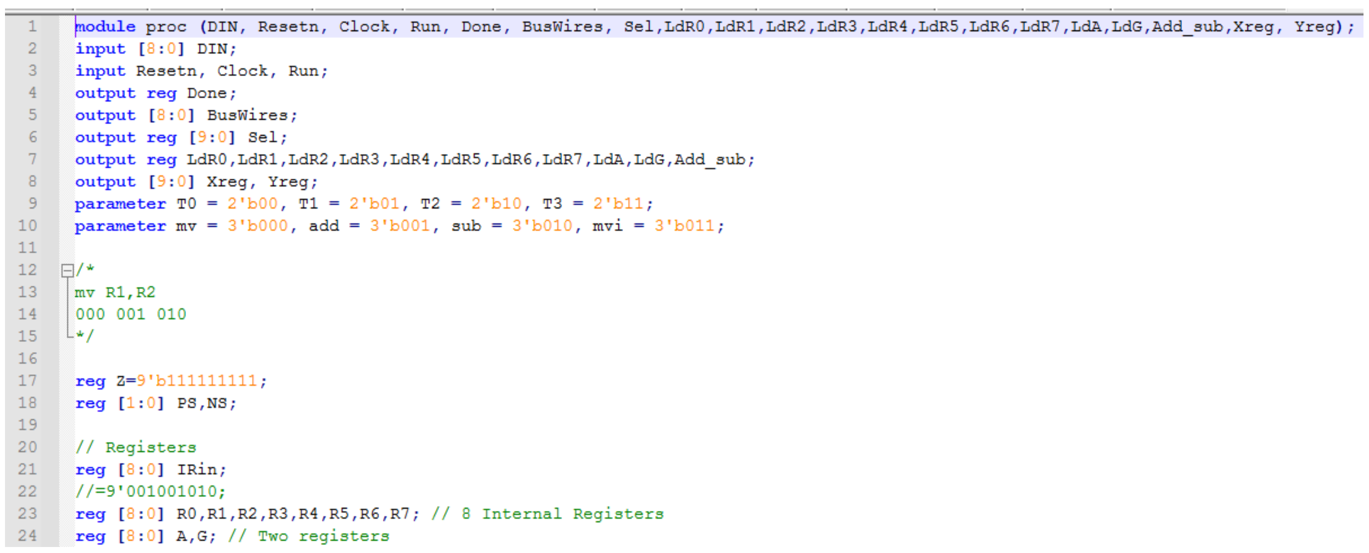


```
module multiplex10_to_1 (R0in, R1in,R2in,R3in,R4in,R5in,R6in,R7in, DIn,sel, clk, R0,R1, R2, R3, R4, R5,R6, R7, L
input R0in, R1in,R2in,R3in,R4in,R5in,R6in,R7in, LdA, LdG, Add_sub;
input [8:0] DIn;
output [8:0] R0,R1, R2, R3, R4, R5,R6, R7;
input [9:0] sel;
input clk;
wire [8:0] bus,G;
Reg_store ha2(R0in, R1in,R2in,R3in,R4in,R5in,R6in,R7in,bus,R0,R1, R2, R3, R4, R5,R6, R7,clk);
multiplex10_to_12 ha1 (R0,R1, R2, R3, R4, R5,R6, R7,DIn,G,sel,bus);
adder_subtractor g01 (LdA, LdG, Add_sub,bus, clk, G);
endmodule


module Reg_store(R0in, R1in,R2in,R3in,R4in,R5in,R6in,R7in,bus,R0,R1, R2, R3, R4, R5,R6, R7,clk);
input R0in, R1in,R2in,R3in,R4in,R5in,R6in,R7in;
input [8:0] bus;
input clk;
output reg [8:0] R0= 9'd0,R1= 9'd0, R2= 9'd0, R3= 9'd0, R4= 9'd0, R5= 9'd0,R6= 9'd0, R7 = 9'd0;

always @ (posedge clk)
begin
if (R0in || R1in || R2in || R3in || R4in || R5in ||  R6in || R7in)
begin
if(R0in == 1)
R0 <= bus;
if (R1in == 1)
R1 <= bus;
if (R2in == 1)
R2 <= bus;
if (R3in == 1)
R3 <= bus;
if (R4in == 1)
R4 <= bus;
if (R5in == 1)
```

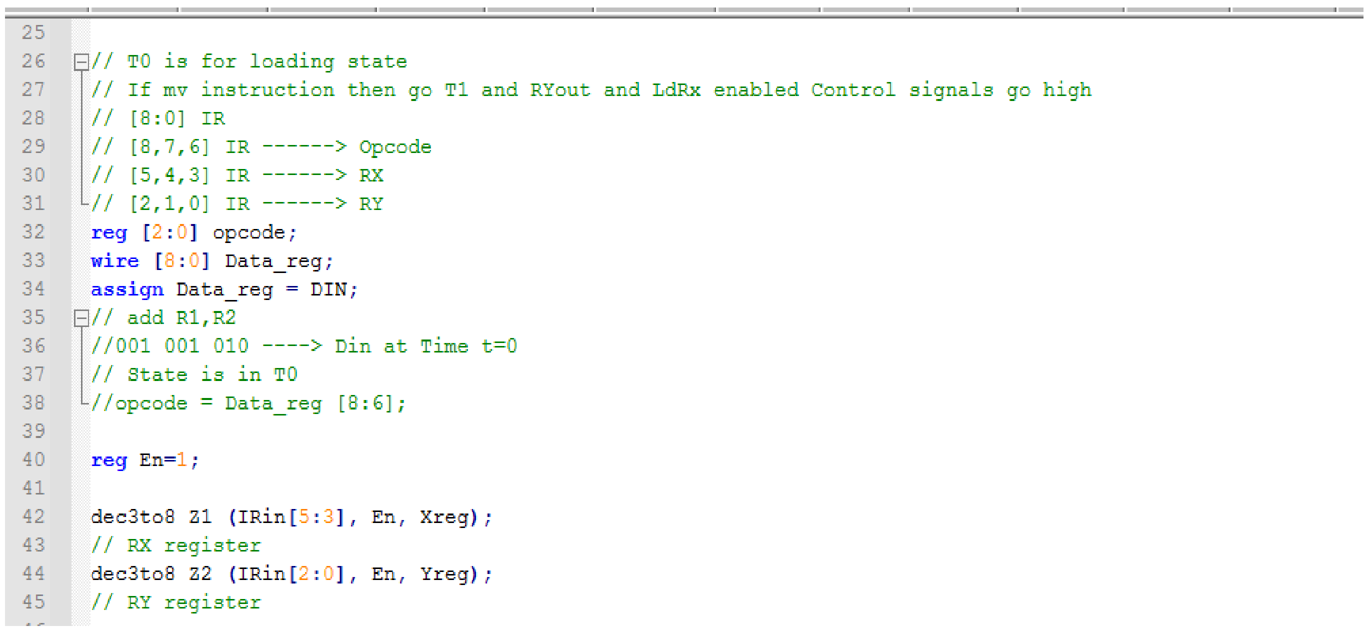
```
Type   ID     Message
       293000 Quartus Prime Full Compilation was successful. 0 errors, 43 warnings
              ******************************************************************
              Running Quartus Prime Netlist Viewers Preprocess
              Command: quartus_npp multiplex10_to_1 -c multiplex10_to_1 --netlist_type=sgate
       18236  Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an appropriate va
              Quartus Prime Netlist Viewers Preprocess was successful. 0 errors, 1 warning
```

Figure 2.13 Quartus Prime Synthesis of the Datapath Module.

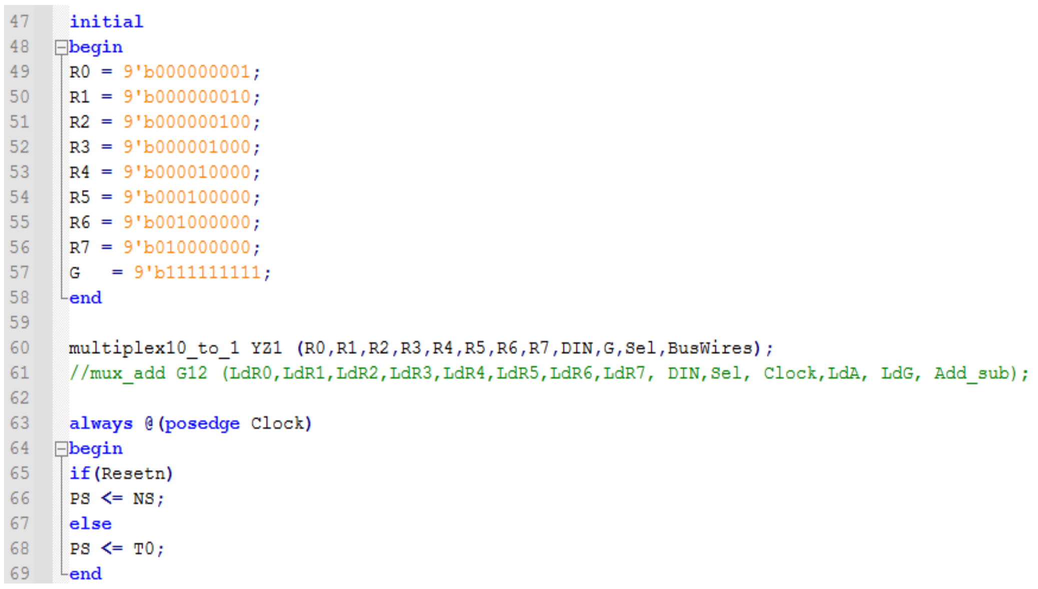
The Controller Path and Processor Integration Part:



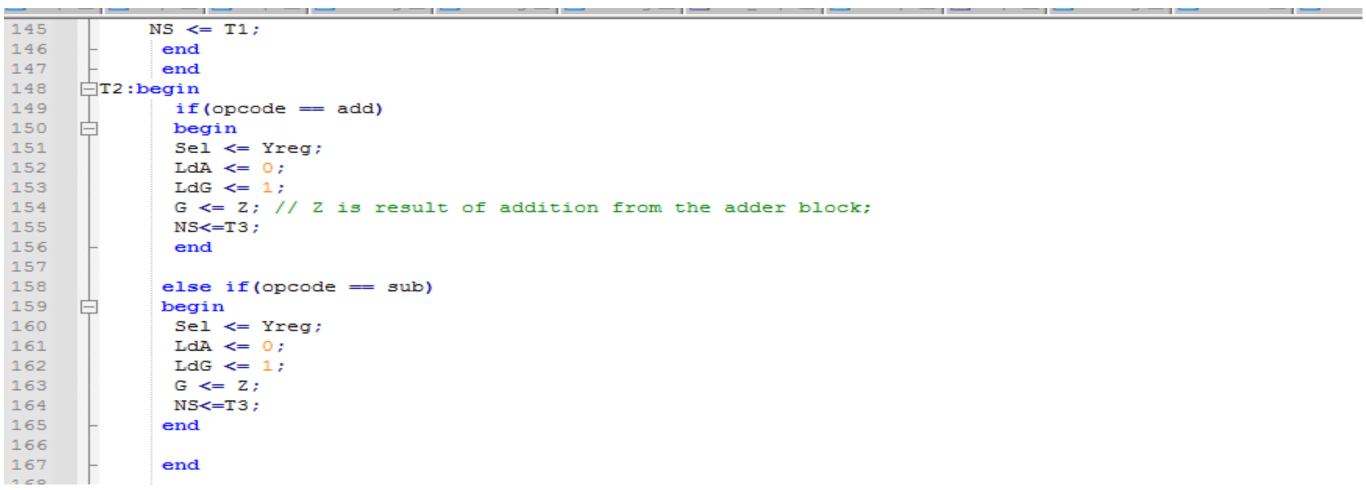
```verilog
module proc (DIN, Resetn, Clock, Run, Done, BusWires, Sel,LdR0,LdR1,LdR2,LdR3,LdR4,LdR5,LdR6,LdR7,LdA,LdG,Add_sub,Xreg, Yreg);
input [8:0] DIN;
input Resetn, Clock, Run;
output reg Done;
output [8:0] BusWires;
output reg [9:0] Sel;
output reg LdR0,LdR1,LdR2,LdR3,LdR4,LdR5,LdR6,LdR7,LdA,LdG,Add_sub;
output [9:0] Xreg, Yreg;
parameter T0 = 2'b00, T1 = 2'b01, T2 = 2'b10, T3 = 2'b11;
parameter mv = 3'b000, add = 3'b001, sub = 3'b010, mvi = 3'b011;

/*
mv R1,R2
000 001 010
*/

reg Z=9'b111111111;
reg [1:0] PS,NS;

// Registers
reg [8:0] IRin;
//=9'001001010;
reg [8:0] R0,R1,R2,R3,R4,R5,R6,R7; // 8 Internal Registers
reg [8:0] A,G; // Two registers
```



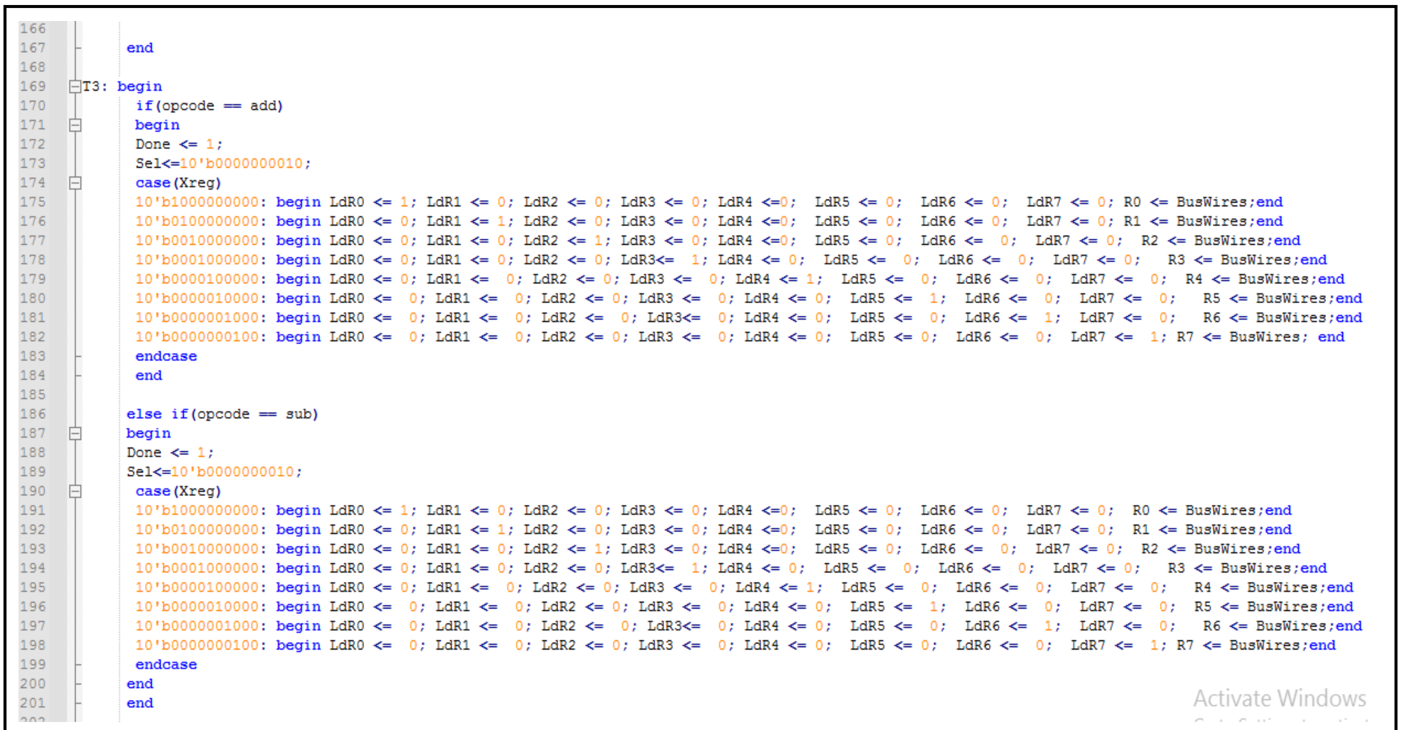
```verilog

// T0 is for loading state
// If mv instruction then go T1 and RYout and LdRx enabled Control signals go high
// [8:0] IR
// [8,7,6] IR ------> Opcode
// [5,4,3] IR ------> RX
// [2,1,0] IR ------> RY
reg [2:0] opcode;
wire [8:0] Data_reg;
assign Data_reg = DIN;
// add R1,R2
//001 001 010 ----> Din at Time t=0
// State is in T0
//opcode = Data_reg [8:6];

reg En=1;

dec3to8 Z1 (IRin[5:3], En, Xreg);
// RX register
dec3to8 Z2 (IRin[2:0], En, Yreg);
// RY register
```



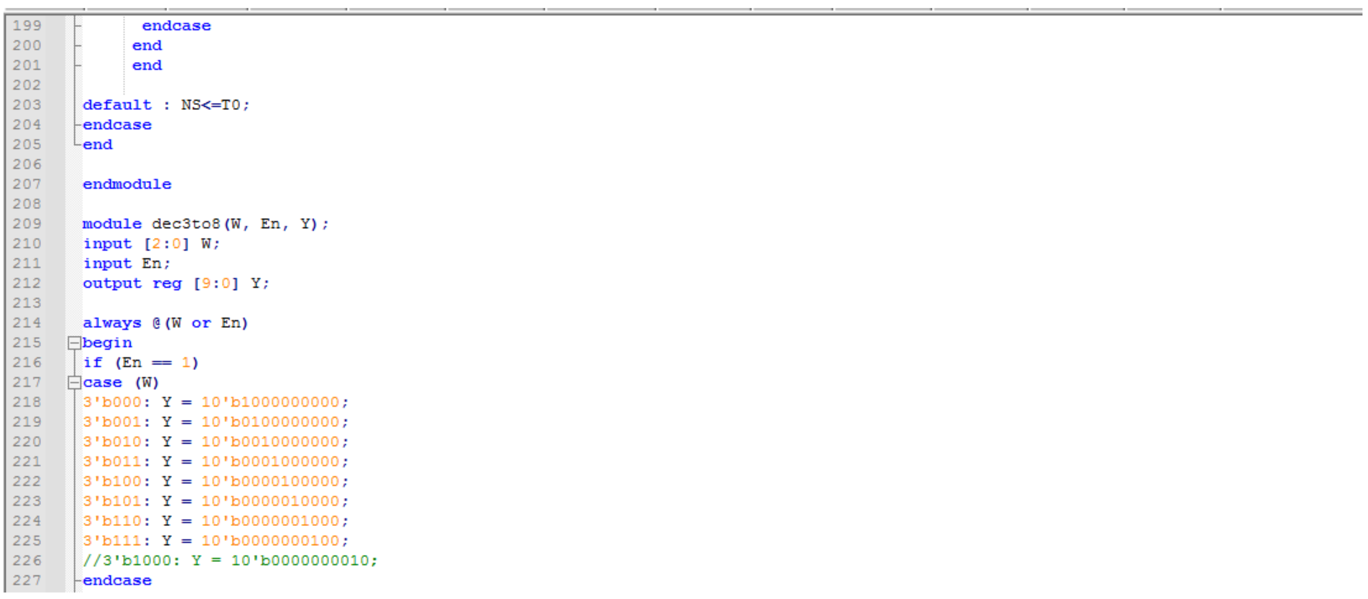
```
initial
begin
R0 = 9'b000000001;
R1 = 9'b000000010;
R2 = 9'b000000100;
R3 = 9'b000001000;
R4 = 9'b000010000;
R5 = 9'b000100000;
R6 = 9'b001000000;
R7 = 9'b010000000;
G   = 9'b111111111;
end

multiplex10_to_1 YZ1 (R0,R1,R2,R3,R4,R5,R6,R7,DIN,G,Sel,BusWires);
//mux_add G12 (LdR0,LdR1,LdR2,LdR3,LdR4,LdR5,LdR6,LdR7, DIN,Sel, Clock,LdA, LdG, Add_sub);

always @(posedge Clock)
begin
if(Resetn)
PS <= NS;
else
PS <= T0;
end
```



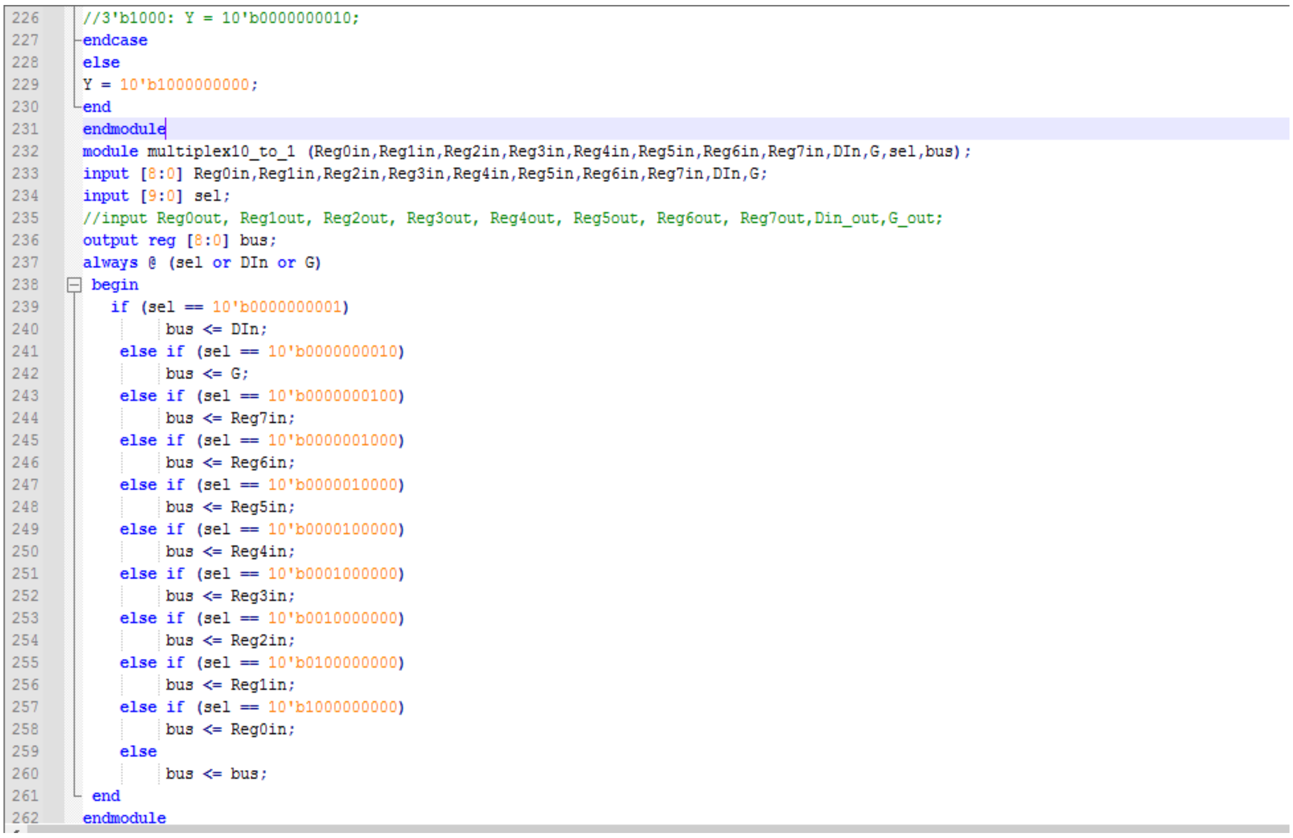
```
        NS <= T1;
         end
         end
T2:begin
          if(opcode == add)
          begin
          Sel <= Yreg;
          LdA <= 0;
          LdG <= 1;
          G <= Z; // Z is result of addition from the adder block;
          NS<=T3;
          end

         else if(opcode == sub)
         begin
          Sel <= Yreg;
          LdA <= 0;
          LdG <= 1;
          G <= Z;
          NS<=T3;
         end

         end
```



```
        end

T3: begin
         if(opcode == add)
         begin
         Done <= 1;
         Sel<=10'b0000000010;
         case(Xreg)
         10'b1000000000: begin LdR0 <= 1; LdR1 <= 0; LdR2 <= 0; LdR3 <= 0; LdR4 <=0;  LdR5 <= 0;  LdR6 <= 0;  LdR7 <= 0; R0 <= BusWires;end
         10'b0100000000: begin LdR0 <= 0; LdR1 <= 1; LdR2 <= 0; LdR3 <= 0; LdR4 <=0;  LdR5 <= 0;  LdR6 <= 0;  LdR7 <= 0; R1 <= BusWires;end
         10'b0010000000: begin LdR0 <= 0; LdR1 <= 0; LdR2 <= 1; LdR3 <= 0; LdR4 <=0;  LdR5 <= 0;  LdR6 <=  0;  LdR7 <= 0;  R2 <= BusWires;end
         10'b0001000000: begin LdR0 <= 0; LdR1 <= 0; LdR2 <= 0; LdR3<=  1; LdR4 <= 0;  LdR5 <=  0;  LdR6 <=  0;  LdR7 <= 0;   R3 <= BusWires;end
         10'b0000100000: begin LdR0 <= 0; LdR1 <=  0; LdR2 <= 0; LdR3 <=  0; LdR4 <= 1;  LdR5 <=  0;  LdR6 <=  0;  LdR7 <=  0;  R4 <= BusWires;end
         10'b0000010000: begin LdR0 <=  0; LdR1 <=  0; LdR2 <= 0; LdR3 <=  0; LdR4 <= 0;  LdR5 <=  1;  LdR6 <=  0;  LdR7 <=  0;   R5 <= BusWires;end
         10'b0000001000: begin LdR0 <=  0; LdR1 <=  0; LdR2 <=  0; LdR3<=  0; LdR4 <= 0;  LdR5 <=  0;  LdR6 <=  1;  LdR7 <=  0;   R6 <= BusWires;end
         10'b0000000100: begin LdR0 <=  0; LdR1 <=  0; LdR2 <= 0; LdR3 <=  0; LdR4 <= 0;  LdR5 <= 0;  LdR6 <=  0;  LdR7 <=  1; R7 <= BusWires; end
         endcase
         end

        else if(opcode == sub)
        begin
        Done <= 1;
        Sel<=10'b0000000010;
         case(Xreg)
         10'b1000000000: begin LdR0 <= 1; LdR1 <= 0; LdR2 <= 0; LdR3 <= 0; LdR4 <=0;  LdR5 <= 0;  LdR6 <= 0;  LdR7 <= 0;  R0 <= BusWires;end
         10'b0100000000: begin LdR0 <= 0; LdR1 <= 1; LdR2 <= 0; LdR3 <= 0; LdR4 <=0;  LdR5 <= 0;  LdR6 <= 0;  LdR7 <= 0;  R1 <= BusWires;end
         10'b0010000000: begin LdR0 <= 0; LdR1 <= 0; LdR2 <= 1; LdR3 <= 0; LdR4 <=0;  LdR5 <= 0;  LdR6 <=  0;  LdR7 <= 0;  R2 <= BusWires;end
         10'b0001000000: begin LdR0 <= 0; LdR1 <= 0; LdR2 <= 0; LdR3<=  1; LdR4 <= 0;  LdR5 <=  0;  LdR6 <=  0;  LdR7 <= 0;   R3 <= BusWires;end
         10'b0000100000: begin LdR0 <= 0; LdR1 <=  0; LdR2 <= 0; LdR3 <=  0; LdR4 <= 1;  LdR5 <=  0;  LdR6 <=  0;  LdR7 <=  0;   R4 <= BusWires;end
         10'b0000010000: begin LdR0 <=  0; LdR1 <=  0; LdR2 <= 0; LdR3 <=  0; LdR4 <= 0;  LdR5 <=  1;  LdR6 <=  0;  LdR7 <=  0;  R5 <= BusWires;end
         10'b0000001000: begin LdR0 <=  0; LdR1 <=  0; LdR2 <=  0; LdR3<=  0; LdR4 <= 0;  LdR5 <=  0;  LdR6 <=  1;  LdR7 <=  0;   R6 <= BusWires;end
         10'b0000000100: begin LdR0 <=  0; LdR1 <=  0; LdR2 <= 0; LdR3 <=  0; LdR4 <= 0;  LdR5 <= 0;  LdR6 <=  0;  LdR7 <=  1; R7 <= BusWires;end
         endcase
        end
        end
```

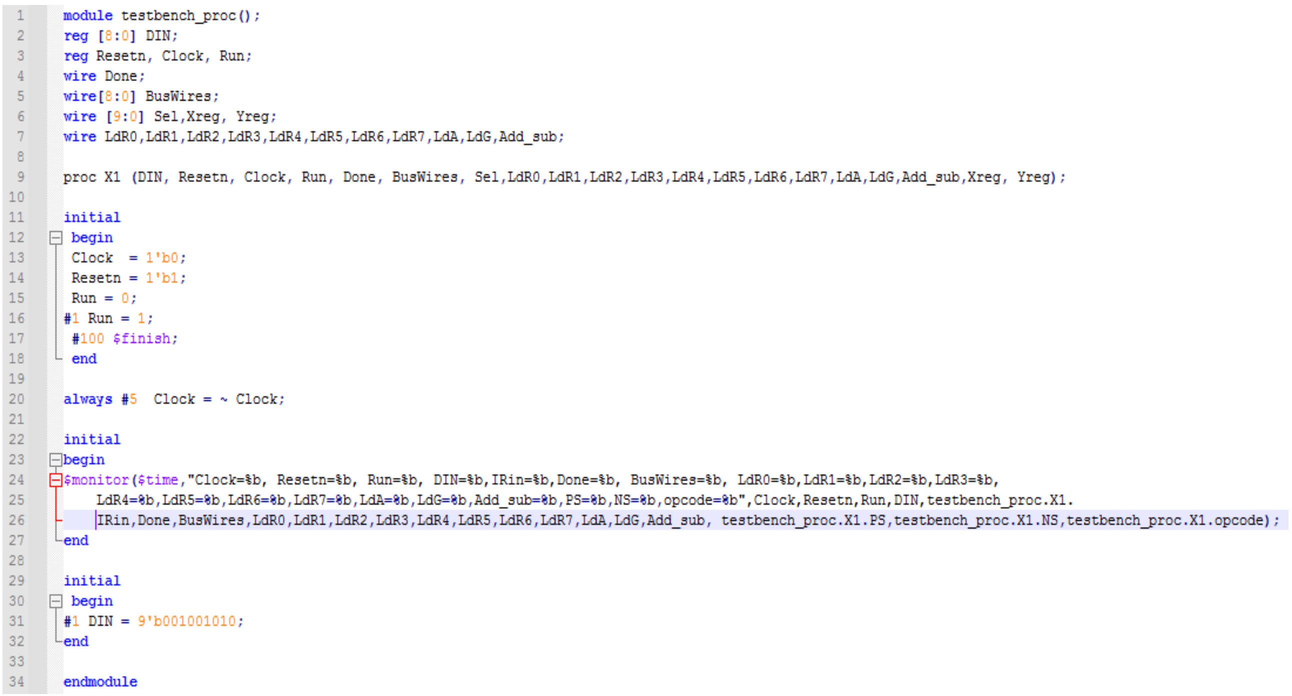


```
         endcase
        end
        end

default : NS<=T0;
endcase
end

endmodule

module dec3to8(W, En, Y);
input [2:0] W;
input En;
output reg [9:0] Y;

always @(W or En)
begin
if (En == 1)
case (W)
3'b000: Y = 10'b1000000000;
3'b001: Y = 10'b0100000000;
3'b010: Y = 10'b0010000000;
3'b011: Y = 10'b0001000000;
3'b100: Y = 10'b0000100000;
3'b101: Y = 10'b0000010000;
3'b110: Y = 10'b0000001000;
3'b111: Y = 10'b0000000100;
//3'b1000: Y = 10'b0000000010;
endcase
```



```verilog
    //3'b1000: Y = 10'b0000000010;
endcase
else
Y = 10'b1000000000;
end
endmodule
module multiplex10_to_1 (Reg0in,Reg1in,Reg2in,Reg3in,Reg4in,Reg5in,Reg6in,Reg7in,DIn,G,sel,bus);
input [8:0] Reg0in,Reg1in,Reg2in,Reg3in,Reg4in,Reg5in,Reg6in,Reg7in,DIn,G;
input [9:0] sel;
//input Reg0out, Reg1out, Reg2out, Reg3out, Reg4out, Reg5out, Reg6out, Reg7out,Din_out,G_out;
output reg [8:0] bus;
always @ (sel or DIn or G)
 begin
    if (sel == 10'b0000000001)
          bus <= DIn;
     else if (sel == 10'b0000000010)
          bus <= G;
     else if (sel == 10'b0000000100)
          bus <= Reg7in;
     else if (sel == 10'b0000001000)
          bus <= Reg6in;
     else if (sel == 10'b0000010000)
          bus <= Reg5in;
     else if (sel == 10'b0000100000)
          bus <= Reg4in;
     else if (sel == 10'b0001000000)
          bus <= Reg3in;
     else if (sel == 10'b0010000000)
          bus <= Reg2in;
     else if (sel == 10'b0100000000)
          bus <= Reg1in;
     else if (sel == 10'b1000000000)
          bus <= Reg0in;
     else
          bus <= bus;
 end
endmodule
```

Testbench Code



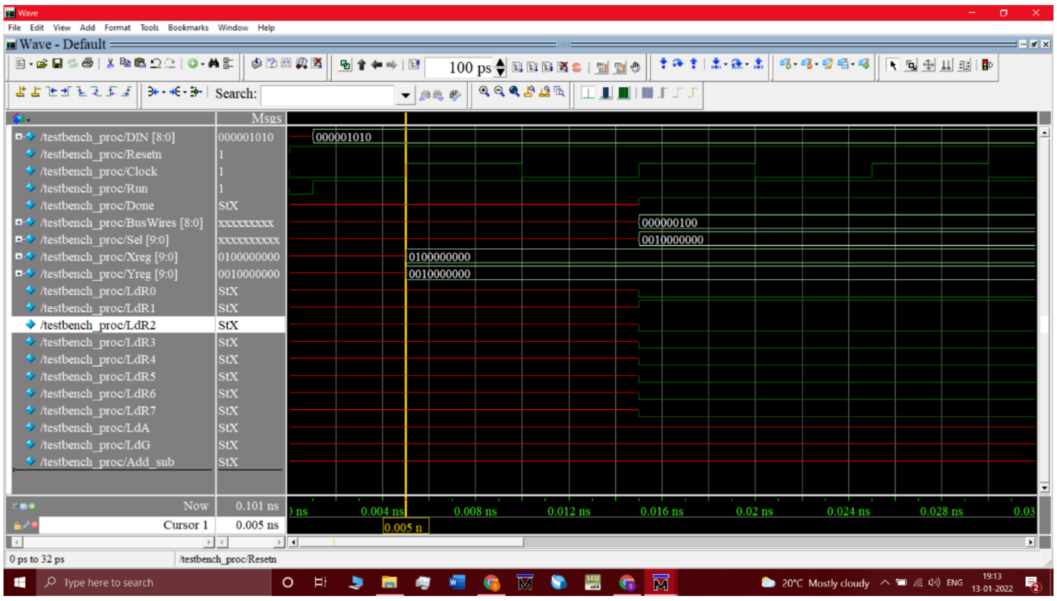
```verilog
module testbench_proc();
reg [8:0] DIN;
reg Resetn, Clock, Run;
wire Done;
wire[8:0] BusWires;
wire [9:0] Sel,Xreg, Yreg;
wire LdR0,LdR1,LdR2,LdR3,LdR4,LdR5,LdR6,LdR7,LdA,LdG,Add_sub;

proc X1 (DIN, Resetn, Clock, Run, Done, BusWires, Sel,LdR0,LdR1,LdR2,LdR3,LdR4,LdR5,LdR6,LdR7,LdA,LdG,Add_sub,Xreg, Yreg);

initial
 begin
 Clock  = 1'b0;
 Resetn = 1'b1;
 Run = 0;
#1 Run = 1;
 #100 $finish;
 end

always #5  Clock = ~ Clock;

initial
begin
$monitor($time,"Clock=%b, Resetn=%b, Run=%b, DIN=%b,IRin=%b,Done=%b, BusWires=%b, LdR0=%b,LdR1=%b,LdR2=%b,LdR3=%b,
    LdR4=%b,LdR5=%b,LdR6=%b,LdR7=%b,LdA=%b,LdG=%b,Add_sub=%b,PS=%b,NS=%b,opcode=%b",Clock,Resetn,Run,DIN,testbench_proc.X1.
    IRin,Done,BusWires,LdR0,LdR1,LdR2,LdR3,LdR4,LdR5,LdR6,LdR7,LdA,LdG,Add_sub, testbench_proc.X1.PS,testbench_proc.X1.NS,testbench_proc.X1.opcode);
end

initial
 begin
#1 DIN = 9'b001001010;
end

endmodule
```

The DIN is provided as Input and Loaded to IRin Register in T0 state only.

Move Operation

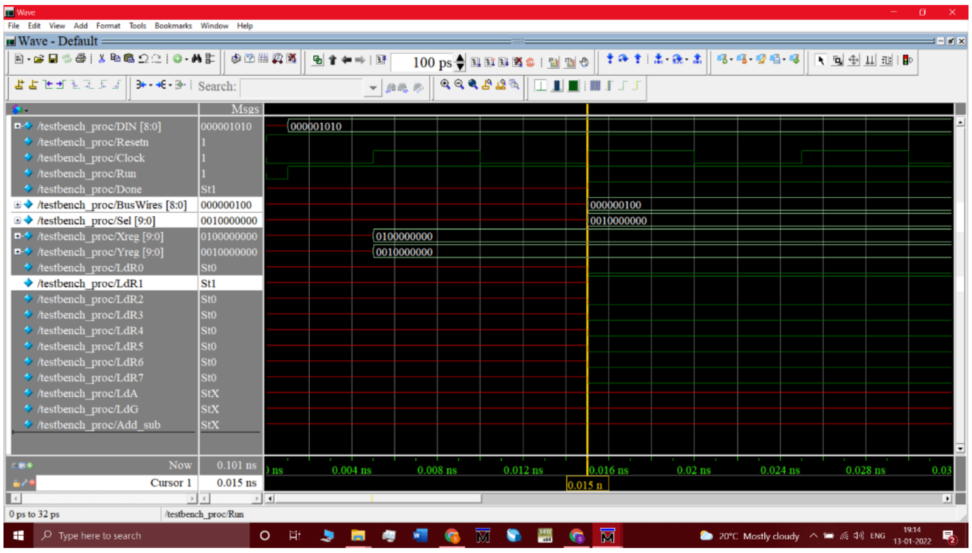
In Testbench we are Loading DIN=9'000001010. At T0 state only DIN is loaded to IRin Register and opcode = IRin[8:6], XReg = IRin[5:3] and YReg = IRin[2:0]. Here, XReg and YReg are used for Decoding the registers.

In this screenshot at t=0.005ns the T0 state is considered. So, IRin=9'b000001010 so, 000 --- Move, 001 --- R1 and 010 --- R2. Loading the data from R2 to R1. Here, we have predefined the values of R0 to R1 initial values. At T1 state according to the table here RYout corresponds to Select line for Mux and RXin is corresponding to the Load signal or Enable signal generated from the ControlFSM Block.



	Msgs
/testbench_proc/DIN [8:0]	000001010
/testbench_proc/Resetn	1
/testbench_proc/Clock	1
/testbench_proc/Run	1
/testbench_proc/Done	StX
/testbench_proc/BusWires [8:0]	xxxxxxxxx
/testbench_proc/Sel [9:0]	xxxxxxxxxx
/testbench_proc/Xreg [9:0]	0100000000
/testbench_proc/Yreg [9:0]	0010000000
/testbench_proc/LdR0	StX
/testbench_proc/LdR1	StX
/testbench_proc/LdR2	StX
/testbench_proc/LdR3	StX
/testbench_proc/LdR4	StX
/testbench_proc/LdR5	StX
/testbench_proc/LdR6	StX
/testbench_proc/LdR7	StX
/testbench_proc/LdA	StX
/testbench_proc/LdG	StX
/testbench_proc/Add_sub	StX
Now	0.101 ns
Cursor 1	0.005 ns

Figure 2.14 T0 state for Move operation.



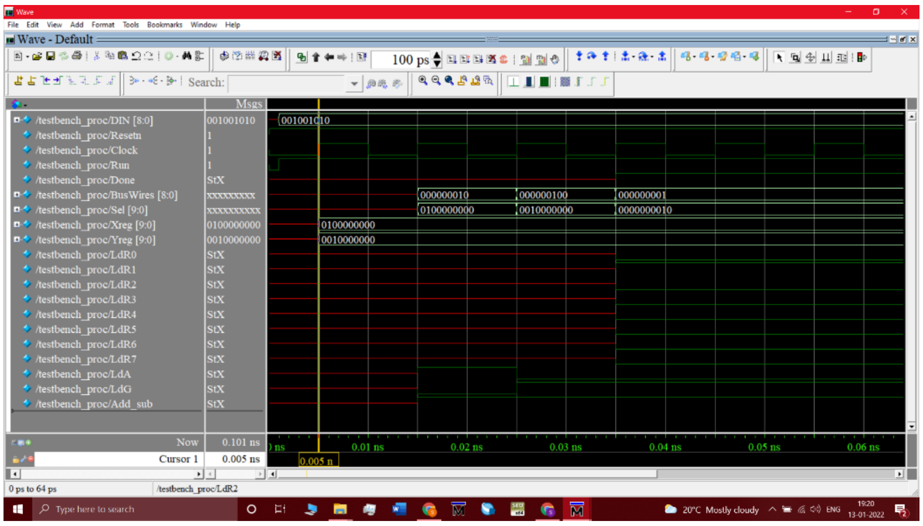
	Msgs
/testbench_proc/DIN [8:0]	000001010
/testbench_proc/Resetn	1
/testbench_proc/Clock	1
/testbench_proc/Run	1
/testbench_proc/Done	St1
/testbench_proc/BusWires [8:0]	000000100
/testbench_proc/Sel [9:0]	0010000000
/testbench_proc/Xreg [9:0]	0100000000
/testbench_proc/Yreg [9:0]	0010000000
/testbench_proc/LdR0	St0
/testbench_proc/LdR1	St1
/testbench_proc/LdR2	St0
/testbench_proc/LdR3	St0
/testbench_proc/LdR4	St0
/testbench_proc/LdR5	St0
/testbench_proc/LdR6	St0
/testbench_proc/LdR7	St0
/testbench_proc/LdA	StX
/testbench_proc/LdG	StX
/testbench_proc/Add_sub	StX
Now	0.101 ns
Cursor 1	0.015 ns

Figure 2.15 T1 state for Move operation. The BusWires lines is loaded with data stored in R2 and LdR1 signal is asserted high. So, the data is stored to R1 register.

Adder Operation

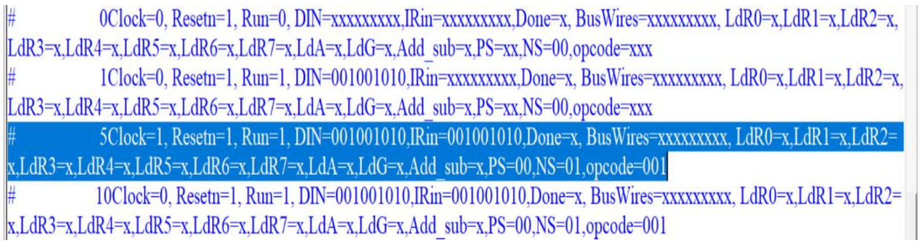
In Testbench we are Loading DIN=9’00100 010. At T0 state only DIN is loaded to IRin Register and opcode = IRin[8:6], XReg = IRin[5:3] and YReg = IRin[2:0]. Here, XReg and YReg are used for Decoding the registers.

In this screenshot at t=0.005ns the T0 state is considered. So, IRin=9’b001001010 so, 001 --- Add, 001 --- R1 and 010 --- R2. Add R1 and R2 and store back to R1. Here, we have predefined the values of R0 to R1 initial values.



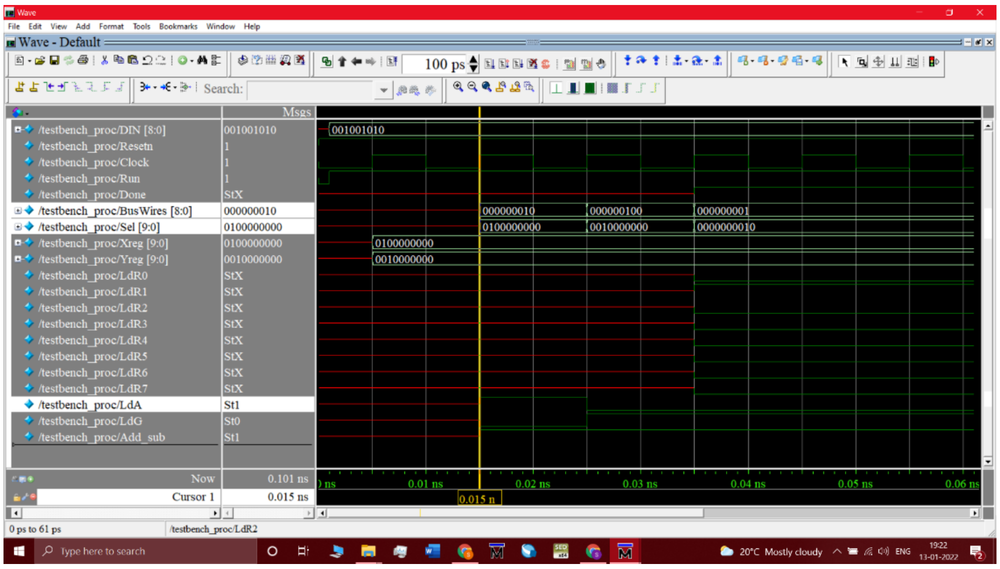
```
/testbench_proc/DIN [8:0]        001001010
/testbench_proc/Resetn           1
/testbench_proc/Clock            1
/testbench_proc/Run              1
/testbench_proc/Done             StX
/testbench_proc/BusWires [8:0]   xxxxxxxxx
/testbench_proc/Sel [9:0]        xxxxxxxxxx
/testbench_proc/Xreg [9:0]       0100000000
/testbench_proc/Yreg [9:0]       0010000000
/testbench_proc/LdR0             StX
/testbench_proc/LdR1             StX
/testbench_proc/LdR2             StX
/testbench_proc/LdR3             StX
/testbench_proc/LdR4             StX
/testbench_proc/LdR5             StX
/testbench_proc/LdR6             StX
/testbench_proc/LdR7             StX
/testbench_proc/LdA              StX
/testbench_proc/LdG              StX
/testbench_proc/Add_sub          StX
Now       0.101 ns
Cursor 1  0.005 ns
```

Figure 2.16 Adder operation at T0 state. IRin is loaded with DIN.



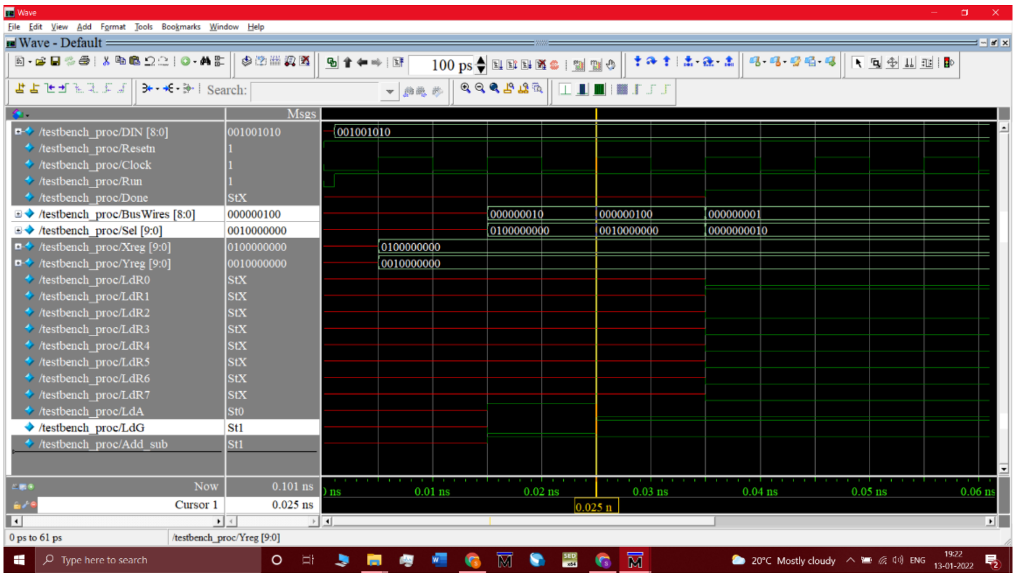
```
#            0Clock=0, Resetn=1, Run=0, DIN=xxxxxxxxx,IRin=xxxxxxxxx,Done=x, BusWires=xxxxxxxxx, LdR0=x,LdR1=x,LdR2=x,
LdR3=x,LdR4=x,LdR5=x,LdR6=x,LdR7=x,LdA=x,LdG=x,Add_sub=x,PS=xx,NS=00,opcode=xxx
#            1Clock=0, Resetn=1, Run=1, DIN=001001010,IRin=xxxxxxxxx,Done=x, BusWires=xxxxxxxxx, LdR0=x,LdR1=x,LdR2=x,
LdR3=x,LdR4=x,LdR5=x,LdR6=x,LdR7=x,LdA=x,LdG=x,Add_sub=x,PS=xx,NS=00,opcode=xxx
#            5Clock=1, Resetn=1, Run=1, DIN=001001010,IRin=001001010,Done=x, BusWires=xxxxxxxxx, LdR0=x,LdR1=x,LdR2=
x,LdR3=x,LdR4=x,LdR5=x,LdR6=x,LdR7=x,LdA=x,LdG=x,Add_sub=x,PS=00,NS=01,opcode=001
#           10Clock=0, Resetn=1, Run=1, DIN=001001010,IRin=001001010,Done=x, BusWires=xxxxxxxxx, LdR0=x,LdR1=x,LdR2=
x,LdR3=x,LdR4=x,LdR5=x,LdR6=x,LdR7=x,LdA=x,LdG=x,Add_sub=x,PS=00,NS=01,opcode=001
```

Figure 2.17 At 5ns we observe the loading the operation part.



Signal	Msgs
/testbench_proc/DIN [8:0]	001001010
/testbench_proc/Resetn	1
/testbench_proc/Clock	1
/testbench_proc/Run	1
/testbench_proc/Done	StX
/testbench_proc/BusWires [8:0]	000000010
/testbench_proc/Sel [9:0]	0100000000
/testbench_proc/Xreg [9:0]	0100000000
/testbench_proc/Yreg [9:0]	0010000000
/testbench_proc/LdR0	StX
/testbench_proc/LdR1	StX
/testbench_proc/LdR2	StX
/testbench_proc/LdR3	StX
/testbench_proc/LdR4	StX
/testbench_proc/LdR5	StX
/testbench_proc/LdR6	StX
/testbench_proc/LdR7	StX
/testbench_proc/LdA	St1
/testbench_proc/LdG	St0
/testbench_proc/Add_sub	St1
Now	0.101 ns
Cursor 1	0.015 ns

Figure 2.18 Adder operation at T1 state. Here, The LdA signal asserted high, so now the data stored from R1 (The select line for R1 is 0100000000) is written to A register. We are setting Add Sub =1 for adder operation and Add-Sub = 0 for subtraction.

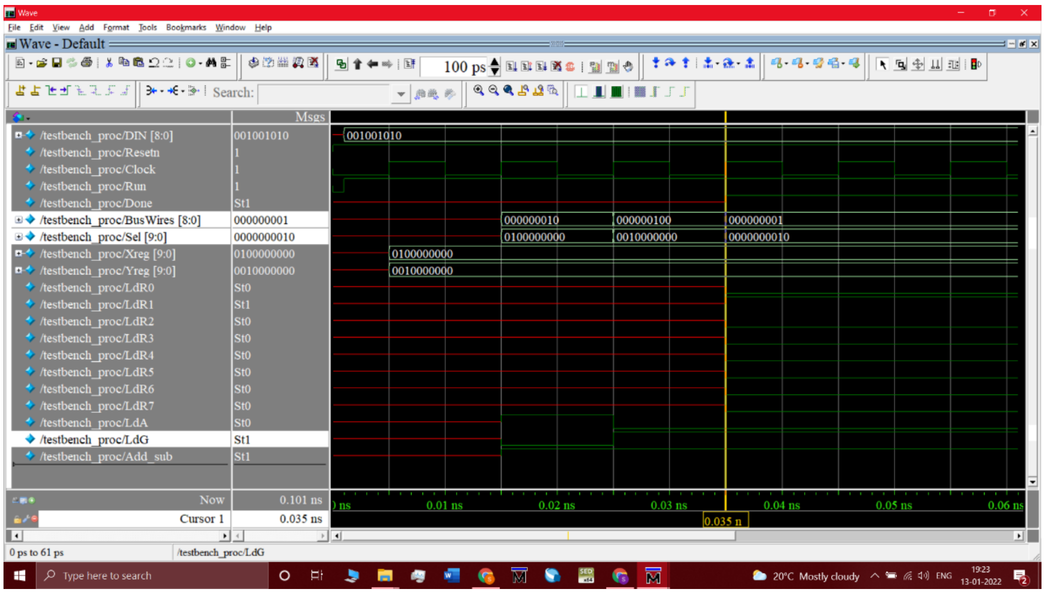


Signal	Msgs
/testbench_proc/DIN [8:0]	001001010
/testbench_proc/Resetn	1
/testbench_proc/Clock	1
/testbench_proc/Run	1
/testbench_proc/Done	StX
/testbench_proc/BusWires [8:0]	000000100
/testbench_proc/Sel [9:0]	0010000000
/testbench_proc/Xreg [9:0]	0100000000
/testbench_proc/Yreg [9:0]	0010000000
/testbench_proc/LdR0	StX
/testbench_proc/LdR1	StX
/testbench_proc/LdR2	StX
/testbench_proc/LdR3	StX
/testbench_proc/LdR4	StX
/testbench_proc/LdR5	StX
/testbench_proc/LdR6	StX
/testbench_proc/LdR7	StX
/testbench_proc/LdA	St0
/testbench_proc/LdG	St1
/testbench_proc/Add_sub	St1
Now	0.101 ns
Cursor 1	0.025 ns

Figure 2.19 Adder Operation at T2 state.

Since the Adder or subtractor block is combinational block, it will be continuously adding. Here, at 25ns the LdG is asserted High and the R2 value is loaded to Adder with A register output signal to adder and saved in G register.

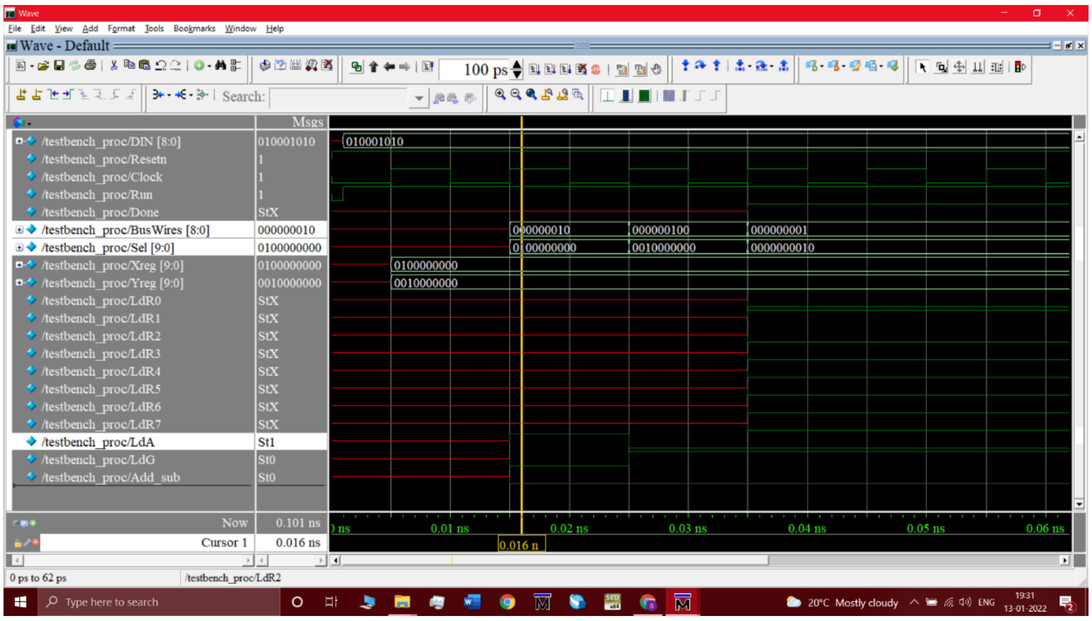
Here, we have implemented Control Path and Datapath separately integration part we have not done. So, In T3 sate the data stored in G is loaded back to R1 through Mux by selecting the Select line of R1 register.



	Msgs
/testbench_proc/DIN [8:0]	001001010
/testbench_proc/Resetn	1
/testbench_proc/Clock	1
/testbench_proc/Run	1
/testbench_proc/Done	St1
/testbench_proc/BusWires [8:0]	000000001
/testbench_proc/Sel [9:0]	0000000010
/testbench_proc/Xreg [9:0]	0100000000
/testbench_proc/Yreg [9:0]	0010000000
/testbench_proc/LdR0	St0
/testbench_proc/LdR1	St1
/testbench_proc/LdR2	St0
/testbench_proc/LdR3	St0
/testbench_proc/LdR4	St0
/testbench_proc/LdR5	St0
/testbench_proc/LdR6	St0
/testbench_proc/LdR7	St0
/testbench_proc/LdA	St0
/testbench_proc/LdG	St1
/testbench_proc/Add_sub	St1
Now	0.101 ns
Cursor 1	0.035 ns

Figure 2.20 Adder Operation at T3 state.

Since, as said we did not integrate Dataflow and Controller Block, so the G register is loaded with predefined value and LdR1 is made high. So, now the BusWires is output of Mux and Loaded to R1.Similarly, for Subtraction Operation In T1 state Add Sub control signal will be 0. The Figure 2.21 will show its working.



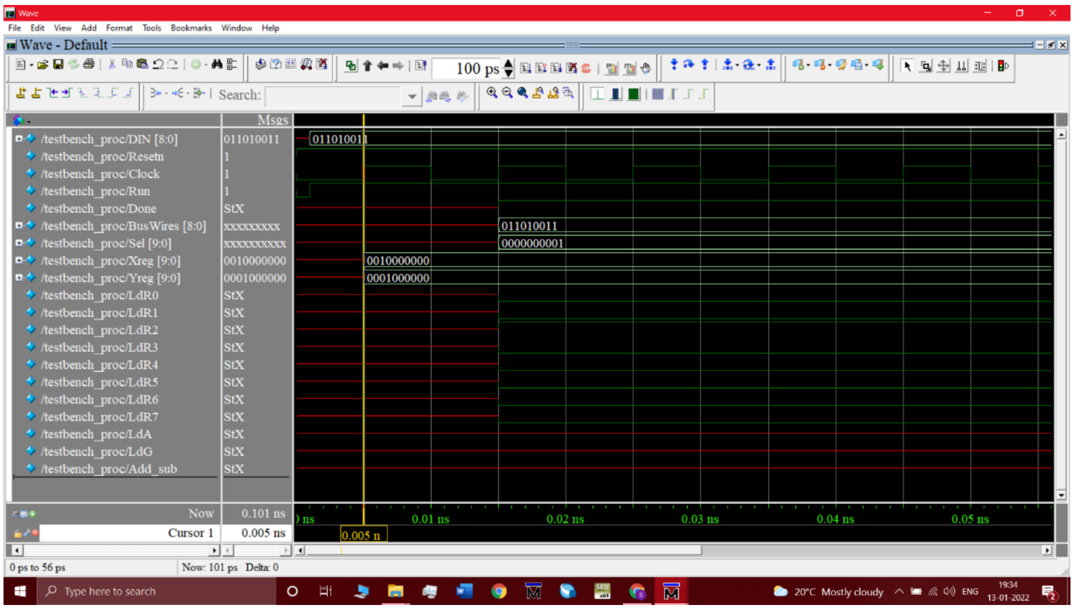
	Msgs
/testbench_proc/DIN [8:0]	010001010
/testbench_proc/Resetn	1
/testbench_proc/Clock	1
/testbench_proc/Run	1
/testbench_proc/Done	StX
/testbench_proc/BusWires [8:0]	000000010
/testbench_proc/Sel [9:0]	0100000000
/testbench_proc/Xreg [9:0]	0100000000
/testbench_proc/Yreg [9:0]	0010000000
/testbench_proc/LdR0	StX
/testbench_proc/LdR1	StX
/testbench_proc/LdR2	StX
/testbench_proc/LdR3	StX
/testbench_proc/LdR4	StX
/testbench_proc/LdR5	StX
/testbench_proc/LdR6	StX
/testbench_proc/LdR7	StX
/testbench_proc/LdA	St1
/testbench_proc/LdG	St0
/testbench_proc/Add_sub	St0
Now	0.101 ns
Cursor 1	0.016 ns

Figure 2.21 Subtraction Operation at T3 state. See the Add_sub control signal is 0.

Move Immediate Operation :

In Testbench we are Loading DIN=9’ b011010011. At T0 state only DIN is loaded to IRin Register and opcode = IRin[8:6], XReg = IRin[5:3] and YReg = IRin[2:0]. Here, XReg and YReg are used for Decoding the registers.

In this screenshot at t=0.005ns the T0 state is considered. So, IRin=9’b000001010 so, 011 --- Move Immediate, 010 --- R2 and YYY part of register has no importance. Loading the data from DIN to R2 only at the next clock cycle after decoding. Here, we have predefined the values of R0 to R1 initial values.



```
Msgs
/testbench_proc/DIN [8:0]       011010011
/testbench_proc/Resetn          1
/testbench_proc/Clock           1
/testbench_proc/Run             1
/testbench_proc/Done            StX
/testbench_proc/BusWires [8:0]  xxxxxxxxx
/testbench_proc/Sel [9:0]       xxxxxxxxxx
/testbench_proc/Xreg [9:0]      0010000000
/testbench_proc/Yreg [9:0]      0001000000
/testbench_proc/LdR0            StX
/testbench_proc/LdR1            StX
/testbench_proc/LdR2            StX
/testbench_proc/LdR3            StX
/testbench_proc/LdR4            StX
/testbench_proc/LdR5            StX
/testbench_proc/LdR6            StX
/testbench_proc/LdR7            StX
/testbench_proc/LdA             StX
/testbench_proc/LdG             StX
/testbench_proc/Add_sub         StX

Now       0.101 ns
Cursor 1  0.005 ns
0 ps to 56 ps    Now: 101 ps  Delta: 0
```

Figure 2.22 T0 state for Move Immediate operation. Instruction decoding takes place.

Result :

The Processor Controller Path and Datapath is implemented in Modelsim and Verified. The Control signal are generated from the Controller block and Datapath is able to load data based on Control signal