SIMPLE PROCESSOR IN VHDL

Anisha Sharma

B20029@students.iitmandi.ac.in

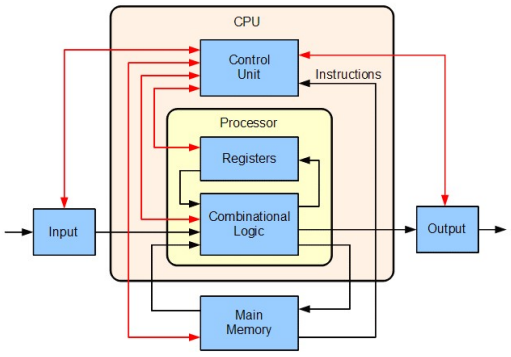
Aim

The main objective of this lab is to understand and implement a simple processor in VHDL with the following functionalities and instructions

Operation	Function
mv Rx, Ry	Rx←[Ry]
mvi Rx, #D	Rx←D
add Rx, Ry	Rx←[Rx]+[Ry]
sub Rx, Ry	Rx←[Rx]-[Ry]
and Rx, Ry	Rx←[Rx]Λ[Ry]
or Rx, Ry	Rx←[Rx]V[Ry]
xor Rx, Ry	Rx←[Rx]⊕[Ry]
not Rx	Rx←¬[Rx]

Theory

The main theory behind this working of the simple processor is as below.



We can see that there are registers the controlling unit and ALU unit in the CPU. Registers just simply store the data and output the data when the inputs are triggered. Control unit controls the state of a register whether to load data or dump the data to bus. It takes use of multiplexer to control the switching of required register among the rest that is which register to switch. ALU simply performs the operation.

Registers and control unit need the clock to trigger i.e. they are synchronous circuits, ALU and demultiplexer doesn't require any clock (they are combinational circuits.)

For implementing this simple processor given in the lab, these are the list of components required

1. Register (16bit) – R0 to R7,A,G
2. Decoder/rev Mux (1:16 with 4 select lines)
3. Control Unit (FSM)
4. ALU (Combinational logic) The

   mechanism is as below:

Register: This can store the data when the clock transition from 0 to 1 triggers and the load is high, and gives the output when the enable is high. There we can connect a not gate to load pin and connect these 2 combinedly naming the mode pin now the TT is as below

Mode	Function
1	Dump to output
0	Load from input

This mode pin is controlled by the decoder.

Also let us assign some addresses to these registers

Register	Address
R0	0000
R1	0001
R2	0010
R3	0011
R4	0100
R5	0101
R6	0110
R7	0111
DIN	1000
A	1001
G	1010
IR	1011

Decode:

The decode activates the required register and send the value from CU to Reg.

The table for register selection with the 4 select lines is as below

Address	16 bit output
0000	ZZZZZZZZZZZZZZZ"&val”
0001	ZZZZZZZZZZZZZZ"&val&"Z
0010	ZZZZZZZZZZZZZ"&val&"ZZ
0011	ZZZZZZZZZZZZ"&val&"ZZZ
0100	ZZZZZZZZZZZ"&val&"ZZZZ
0101	ZZZZZZZZZZ"&val&"ZZZZZ
0110	ZZZZZZZZZ"&val&"ZZZZZZ
0111	ZZZZZZZZ"&val&"ZZZZZZZ
1000	ZZZZZZZ"&val&"ZZZZZZZZ
1001	ZZZZZZ"&val&"ZZZZZZZZZ
1010	ZZZZZ"&val&"ZZZZZZZZZZ
1011	ZZZZ"&val&"ZZZZZZZZZZZ

Here the “val” is the mode of the register i.e. which controls the register to load in or dump out the data.

Control Unit:

For the given operations we can tabulate the steps as below (Collectively for all the instructions)

State	Process
0	DIN -> BUS
1	BUS -> IR
2	DIN -> BUS
3	BUS -> Rx
4	Rx -> BUS
5	BUS -> A
6	Ry -> BUS
7	f(A, BUS) -> G
8	G -> BUS
9	BUS -> Rx

Now the instructions can be realised as below

mv – 0,1,6,9 mvi – 0,1,2,3

all the rest – 0,1,4,5,6,7,8,9

Now for every state we can perform the required operation accordingly by just giving the signal to addr and val to de-mux. Defining a simple FSM for this control unit serves the purpose.

The format of instructions are as given in the question. “IIIXXXYYY” – Lower 9 bits

ALU:

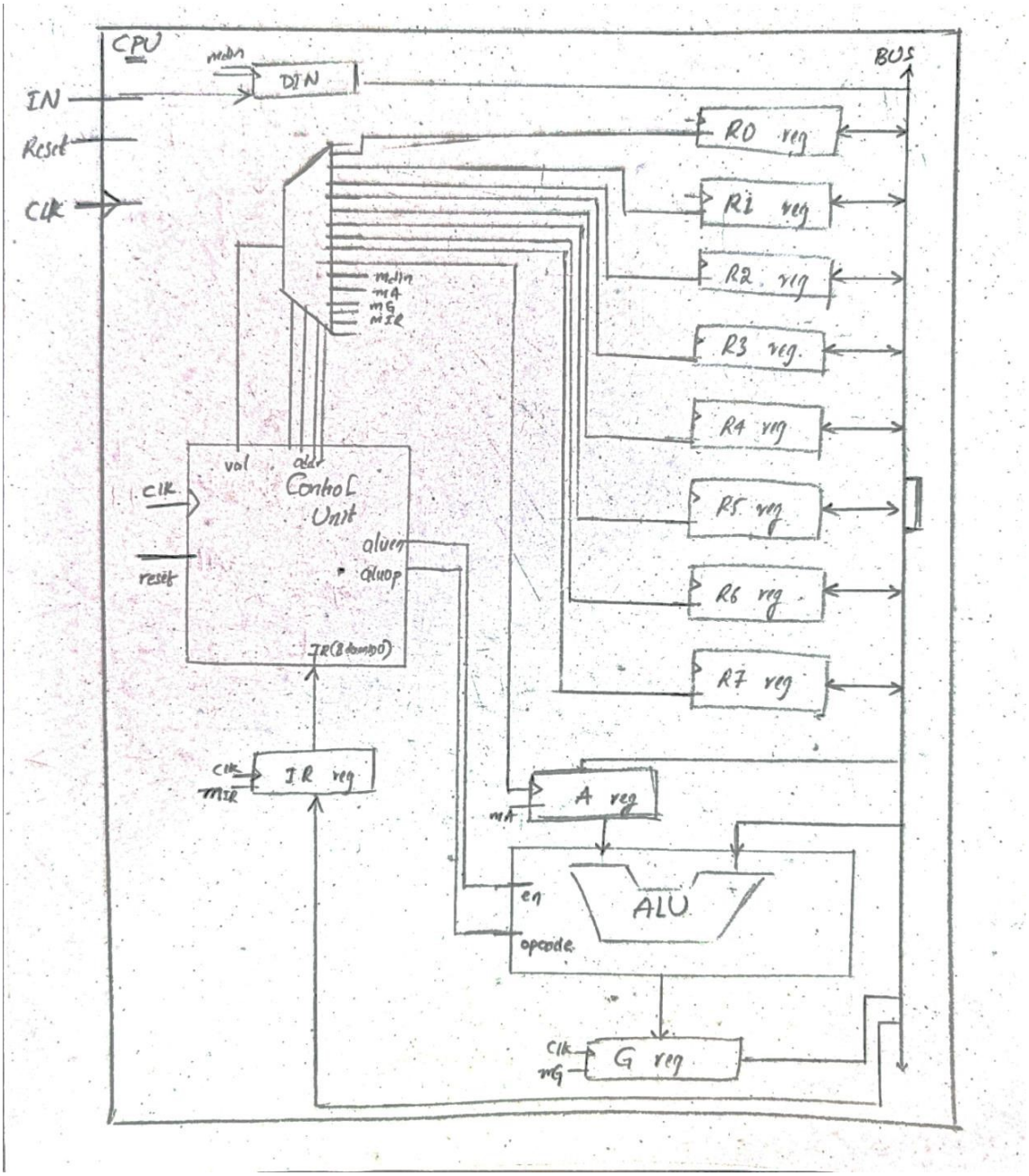
This takes the inputs, opcode. According to the opcode it performs the operation and dumps out the result.

For this project the opcodes are considered as below because they will directly correspond to the instruction at required state.

Opcode	Operation
010	Add
011	Subtract
100	AND
101	OR
110	XOR
111	Not

Circuit Diagram

The circuit / architecture can be realised as follows



Note: The schematics and implementation designs are attached at the Discussion section.

VHDL

1.The VHDL code for the register is as below

```vhdl
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_unsigned.all;

entity reg is
    port(
        din : in std_logic ;
        clk: in std_logic ;
        rst: in std_logic ;
        mode : in std_logic ;
        input : in std_logic_vector (15 downto 0);
        output : out std_logic_vector (15 downto 0);
        output_alu : out std_logic_vector (15 downto 0)
    );
end entity;

architecture  behav of reg is
    signal t1 : std_logic ;
    signal stored : std_logic_vector (15 downto 0) := (others=>'Z');
begin
    process(clk,rst,mode)
    begin
        if (din = '1' and input /= "ZZZZZZZZZZZZZZZ") then
            stored <= input;
        end if;
        if rst = '1' then
            stored <= (others=>'Z');
        elsif rising_edge (clk) and mode='0' and din='0' then
            stored <= input ;
        end if;
    end process;
    output <= stored when mode ='1' else (others=>'Z');
    output_alu  <= stored;
end behav;
```

2.The VHDL code for the decoder/reverse multiplexer is as below

```vhdl
library ieee;
use ieee.std_logic_1164.ALL;

entity  mux is
port (
        val : in std_logic;
        sel : in std_logic_vector (3 downto 0);
        lines : out std_logic_vector (15 downto 0)
        );
end entity;

```

```
architecture behav of mux is
begin
process (sel, val )
begin
case sel is
    when "0000" =>
        lines <=  "ZZZZZZZZZZZZZZZ"&val ;
    when "0001" =>
        lines <= "ZZZZZZZZZZZZZZ"&val&"Z";
    when "0010" =>
        lines <= "ZZZZZZZZZZZZZ"&val&"ZZ";
    when "0011" =>
        lines <= "ZZZZZZZZZZZZ"&val&"ZZZ";
    when "0100" =>
        lines <= "ZZZZZZZZZZZ"&val&"ZZZZ";
    when "0101" =>
        lines <= "ZZZZZZZZZZ"&val&"ZZZZZ";
    when "0110" =>
        lines <= "ZZZZZZZZZ"&val&"ZZZZZZ";
    when "0111" =>
        lines <= "ZZZZZZZZ"&val&"ZZZZZZZ";
    when "1000" =>
        lines <= "ZZZZZZZ"&val&"ZZZZZZZZ";
    when "1001" =>
        lines <= "ZZZZZZ"&val&"ZZZZZZZZZ";
    when "1010" =>
        lines <= "ZZZZZ"&val&"ZZZZZZZZZZ";
    when "1011" =>
        lines <= "ZZZZ"&val&"ZZZZZZZZZZZ";
    when others =>
        lines(15) <= val;
end case;
end process ;
end behav;
```

3.The VHDL code for the ALU is as below

```vhdl
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_unsigned.ALL ;

entity  alu is
port(
        en: in std_logic ;
        a: in std_logic_vector (15 downto 0);
        b: in std_logic_vector (15 downto 0);
        op : in std_logic_vector (2 downto 0);
        output : out std_logic_vector (15 downto 0)
        );
end entity;

architecture behav of alu is
signal t16 : std_logic_vector (15 downto 0);
begin
process (a,b,op,en)
begin
if en = '1' then
case op is
    when "010" =>
        t16 <= a+b;
    when "011" =>
        t16 <= a-b;
    when "100" =>
        t16 <= a and b;
    when "101" =>
        t16 <= a or b;
    when "110" =>
        t16 <= a xor b;
    when "111" =>
        t16 <= not a;
    when others=>
        t16 <= (others =>'Z');
end case;
end if;
end process ;
output <= t16;
end behav ;
```

4.The VHDL code for the control unit FSM is as below

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity cu is
  Port (
        reset : in std_logic ;
        clk : in std_logic ;
        ir : in std_logic_vector (8 downto 0);
        muxval: out std_logic ;
        muxaddr : out std_logic_vector (3 downto 0);
        aluop : out std_logic_vector (2 downto 0);
        aluen : out std_logic
    );
end entity ;

architecture Behavioral of cu is
signal addrx,addry : std_logic_vector (3 downto 0);
signal instruction : std_logic_vector (2 downto 0);
signal tmuxaddr : std_logic_vector (3 downto 0);
signal tmuxval : std_logic ;
signal oo : std_logic := '0';
type statetype is (s00,s0,s1,s2,s3,s4,s5,s6,s7,s8,s9);
signal st,nst : statetype := s00;
begin
addry <= "0"&ir(2 downto 0);
addrx <= "0"&ir(5 downto 3);
instruction <= ir(8 downto 6);
process(instruction ,addrx , addry, st, tmuxaddr , tmuxval, nst )
begin
case st is
    when s00 =>
        nst <= s0;
    when s0 =>
        tmuxval <= '1';
        tmuxaddr  <= "1000";
        nst <= s1;
    when s1 =>
        tmuxval <= '0';
        tmuxaddr <= "1011";
        case instruction is
            when "000" =>
                nst <= s6;
            when "001" =>
                nst <= s2;
            when "010" =>
                nst <= s4;
            when "011" =>
                nst <= s4;
            when "100" =>
                nst <= s4;
            when "101" =>
                nst <= s4;
            when "110" =>
                nst <= s4;
            when "111" =>
                nst <= s4;
            when others =>
                nst <= s0;
        end case;
    when s2 =>
        tmuxval <= '1';
        tmuxaddr <= "1000";
        nst <= s3;
    when s3 =>
        tmuxval <= '0';
        tmuxaddr <= addrx ;
        nst <= s0;
    when s4 =>
        tmuxval <= '1';
        tmuxaddr <= addrx;
        nst <= s5;
    when s5 =>
        tmuxval <= '0';
        tmuxaddr <= "1001";
        nst <= s6;
    when s6 =>
        tmuxval <= '1';
        tmuxaddr <= addry;
        case instruction is
            when "000" =>
                nst <= s9;
            when others =>
                nst <= s7;
        end case;
    when s7 =>
        aluen <= '1';
        aluop <= instruction ;
        tmuxval <= '1';
        tmuxaddr <= "1010";
        tmuxval <= 'Z';
        nst <= s8;
    when s8 =>
        aluen <='0';
        tmuxval <= '1';
        tmuxaddr <= "1010";
        nst <= s9;
    when s9 =>
        tmuxval <= '0';
        tmuxaddr <= addrx;
        nst <= s0;
    when others =>
        nst <= s0;
end case;
end process;
 muxaddr <= tmuxaddr;
clk_proc : process
begin
wait until (clk'event and clk='1');
    if reset  = '1' then
        st <= s00;
    elsif oo = '0' then
        st <= nst;
    end if;
    oo <= not oo;
end process ;
end Behavioral;
```

5.The final VHDL code four our processing connecting all these components is as below

```
library ieee;
use ieee.std_logic_1164.all;

entity cpu is
    Port(
         reset : in std_logic ;
         cpuclk : in std_logic ;
         mdin : in std_logic_vector (15 downto 0);
         outputtt : out std_logic_vector(15 downto 0)
            );
end entity ;

architecture structural of cpu is
component   mux is
port (
        val : in std_logic;
        sel : in std_logic_vector (3 downto 0);
        lines : out std_logic_vector (15 downto 0)
        );
end component ;
component  alu is
port(
        en: in std_logic ;
        a: in std_logic_vector (15 downto 0);
        b: in std_logic_vector (15 downto 0);
        op : in std_logic_vector (2 downto 0);
        output : out std_logic_vector (15 downto 0)
        );
end component ;
component  cu is
  Port (
        reset : in std_logic ;
        clk : in std_logic ;
        ir : in std_logic_vector (8 downto 0);
        muxval: out std_logic ;
        muxaddr : out std_logic_vector (3 downto 0);
        aluop : out std_logic_vector (2 downto 0);
        aluen : out std_logic
     );
end component ;
component  reg is
port(
    din : in std_logic ;
    clk: in std_logic ;
    rst: in std_logic ;
    mode : in std_logic ;
    input : in std_logic_vector (15 downto 0);
    output : out std_logic_vector (15 downto 0);
    output_alu : out std_logic_vector (15 downto 0)
    );
end component ;
--Signals
signal sbusin,sbusout,sir,slines : std_logic_vector (15 downto 0);
signal sclk : std_logic ;
signal srst, smuxval : std_logic ;
signal alua,aluo : std_logic_vector (15 downto 0);
signal smuxaddr  : std_logic_vector (3 downto 0);
signal opcode : std_logic_vector (2 downto 0);
signal saluen : std_logic ;

begin
Buss: reg port map (din ='1',clk => sclk,rst => srst,mode ='U',input => sbusout, output_alu => sbusin);
R0: reg port map (din => '0',clk => sclk,rst => srst,mode =>slines(0),input => sbusin, output => sbusout);
R1: reg port map (din => '0',clk => sclk,rst => srst,mode =>slines(1),input => sbusin, output => sbusout);
R2: reg port map (din => '0',clk => sclk,rst => srst,mode =>slines(2),input => sbusin, output => sbusout);
R3: reg port map (din => '0',clk => sclk,rst => srst,mode =>slines(3),input => sbusin, output => sbusout);
R4: reg port map (din => '0',clk => sclk,rst => srst,mode =>slines(4),input => sbusin, output => sbusout);
R5: reg port map (din => '0',clk => sclk,rst => srst,mode =>slines(5),input => sbusin, output => sbusout);
R6: reg port map (din => '0',clk => sclk,rst => srst,mode =>slines (6),input => sbusin, output => sbusout);
R7: reg port map (din => '0',clk => sclk,rst => srst,mode =>slines(7),input => sbusin, output => sbusout);
DINN: reg port map (din => '1',clk => sclk,rst => srst,mode =>slines(8),input => mdin, output => sbusout);
A : reg port map (din => '0',clk => sclk,rst => srst,mode =>slines(9),input => sbusin, output_alu => alua);
G : reg port map (din => '1',clk => sclk,rst => srst,mode =>slines(10),input => aluo, output => sbusout);
IR : reg port map (din => '0',clk => sclk,rst => srst,mode =>slines(11),input => sbusin, output_alu => sir);
demux : mux port map (val =>smuxval ,sel =>smuxaddr , lines => slines );
control : cu port map (reset => srst , clk => sclk  , ir => sir(8 downto 0), muxval => smuxval , muxaddr => smuxaddr , aluop => opcode, aluen => saluen);
aluu : alu  port map(en => saluen,a => alua,b => sbusin ,op => opcode,output=>aluo);
sclk <= opuclk;
srst <= reset ;
outputtt <= sbusout  ;
end structural;
```

Results of Testbench

The common testbench code is as below

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity cputb is
end entity;

architecture behav of cputb is
component cpu is
    Port(
         reset : in std_logic ;
         cpuclk : in std_logic ;
         mdin : in std_logic_vector (15 downto 0);
         outputtt : out std_logic_vector(15 downto 0)
            );
end component ;
signal clksignal,reset : std_logic ;
signal input : std_logic_vector (15 downto 0);
constant clk_period : time := 10 ns;
begin
Processor : cpu port map (reset => reset, cpuclk => clksignal, mdin => input);
clk_process :process
    begin
        clksignal  <= '0';
        wait for clk_period/2;
        clksignal  <= '1';
        wait for clk_period/2;
end process;
stim_proc: process
    begin
        reset <= '0';
        --inputs are given here.

        reset <= '1';
        input <= (others => 'U');
        wait ;
end process;
end architecture;
```

We considered R0 and R1 registers for the task.

Let us first load 2 values into the registers and perform the different operations.

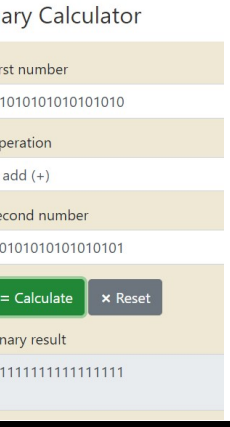
mv, mvi can be tested by loading into a register moving it to another, again loading to the same register. At this point we will be having the values loaded in 2 registers. By sending the instruction now we can test the remaining all operations.

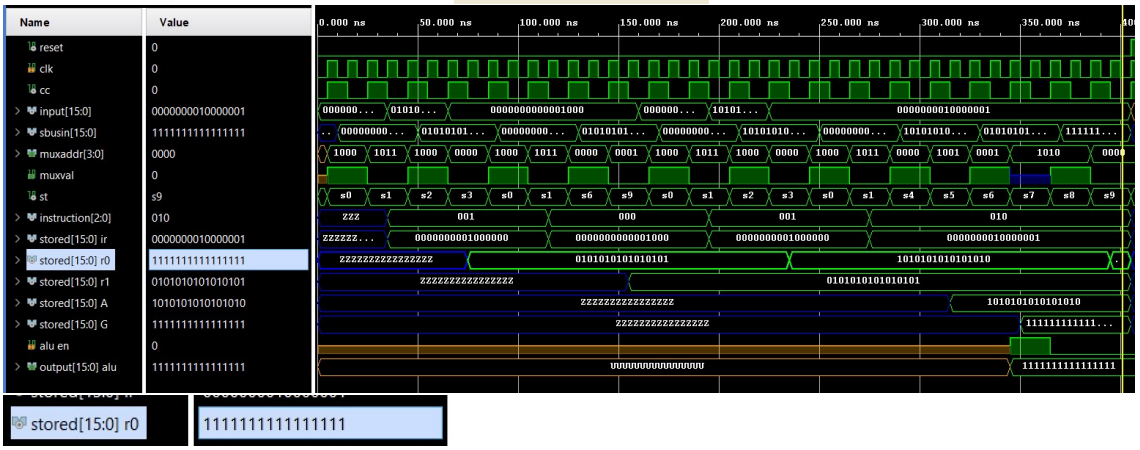
0. Loading the data – Testing “mv” and “mvi”

```vhdl
--Input mvi
input <= "0000000001000000" ;
wait for 35ns;
input <= "0101010101010101";
wait for 30ns;
--Input mv
input <= "0000000000001000";
wait for 95ns;
--Input mvi
input <= "0000000001000000";
wait for 35ns;
input <= "1010101010101010";
wait for 30ns;
```

1. Now performing the operation on the data in R0 and R1
   a.Add

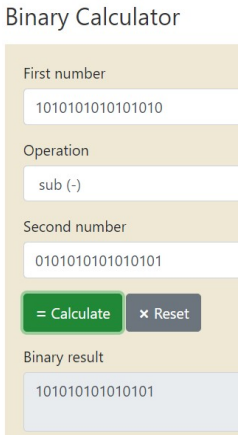
```
--Input mvi
input <= "0000000001000000" ;
wait for 35ns;
input <= "0101010101010101";
wait for 30ns;
--Input mv
input <= "0000000000001000";
wait for 95ns;
--Input mvi
input <= "0000000001000000";
wait for 35ns;
input <= "1010101010101010";
wait for 30ns;
--adding these and storing it to R0
input <= "0000000010000001";
wait for 180ns;
```

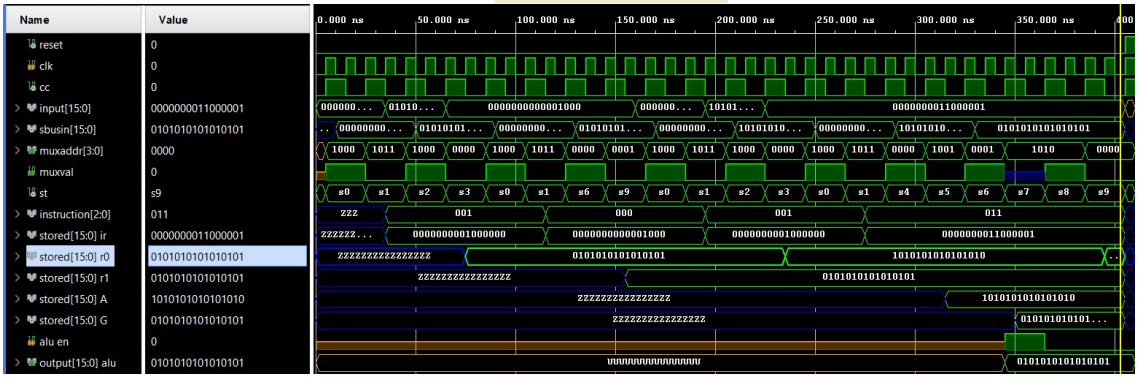




b.Subtract

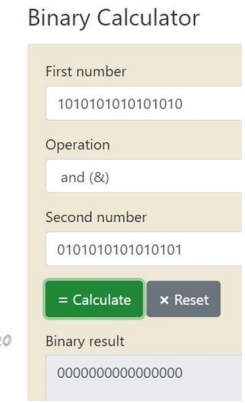
```
--Input mvi
input <= "0000000001000000" ;
wait for 35ns;
input <= "0101010101010101";
wait for 30ns;
--Input mv
input <= "0000000000001000";
wait for 95ns;
--Input mvi
input <= "0000000001000000";
wait for 35ns;
input <= "1010101010101010";
wait for 30ns;
--subtracting these and storing it to R0
input <= "0000000011000001";
wait for 180ns;
```

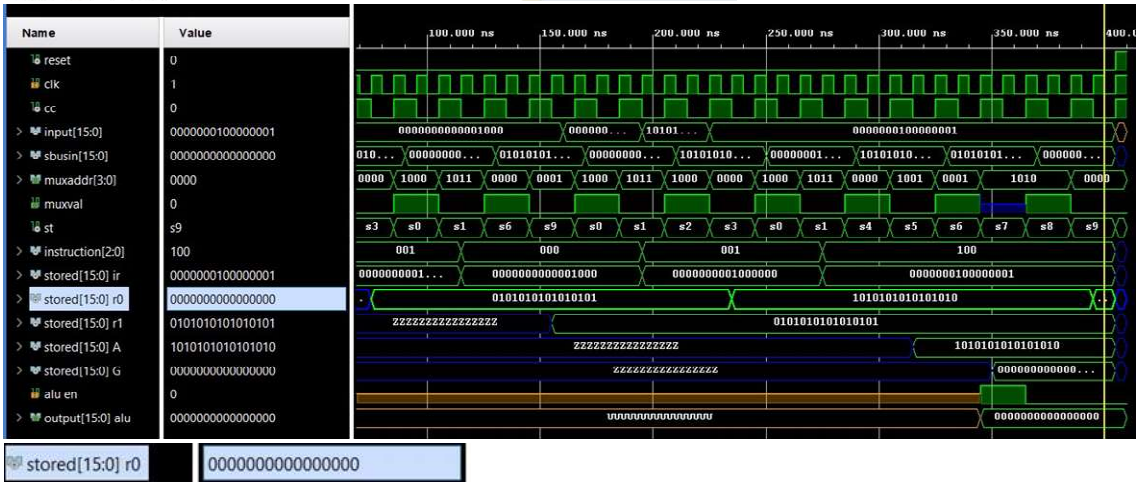




c.AND

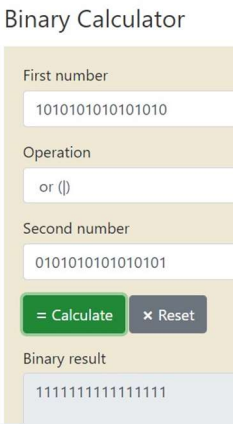
```
--Input mvi
input <= "0000000001000000" ;
wait for 35ns;
input <= "0101010101010101";
wait for 30ns;
--Input mv
input <= "0000000000001000";
wait for 95ns;
--Input mvi
input <= "0000000001000000";
wait for 35ns;
input <= "1010101010101010";
wait for 30ns;
--performing "and" on these and storing it to R0
input <= "0000000100000001";
wait for 180ns;
```

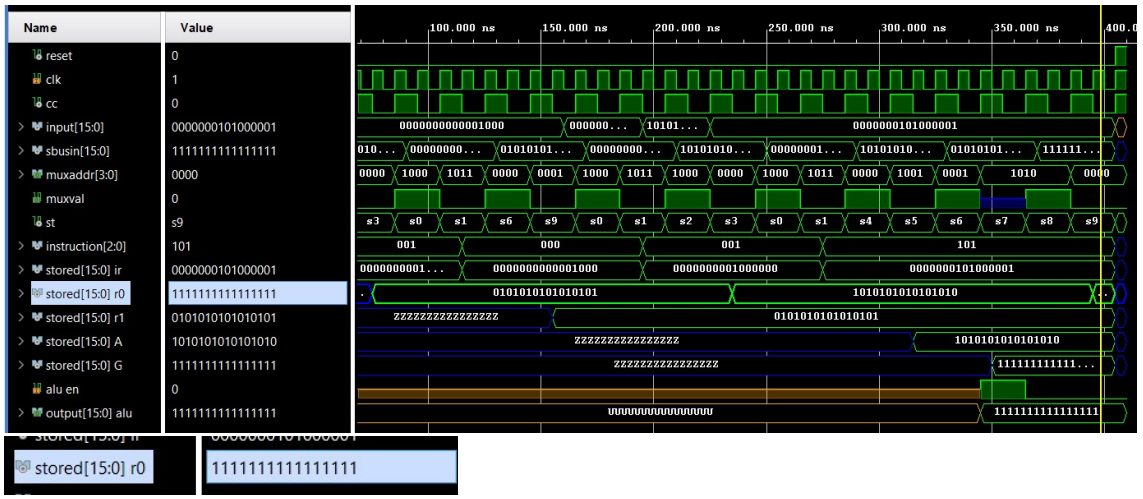




d.OR

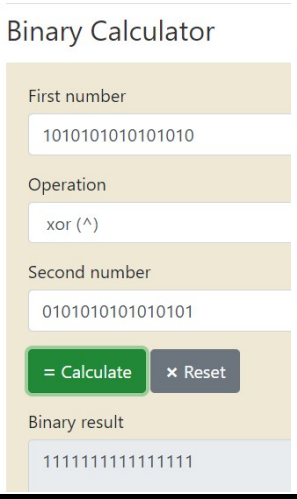
```
--Input mvi
input <= "0000000001000000" ;
wait for 35ns;
input <= "0101010101010101";
wait for 30ns;
--Input mv
input <= "0000000000001000";
wait for 95ns;
--Input mvi
input <= "0000000001000000";
wait for 35ns;
input <= "1010101010101010";
wait for 30ns;
--performing "or" on these and storing it to R0
input <= "0000000101000001";
wait for 180ns;
```

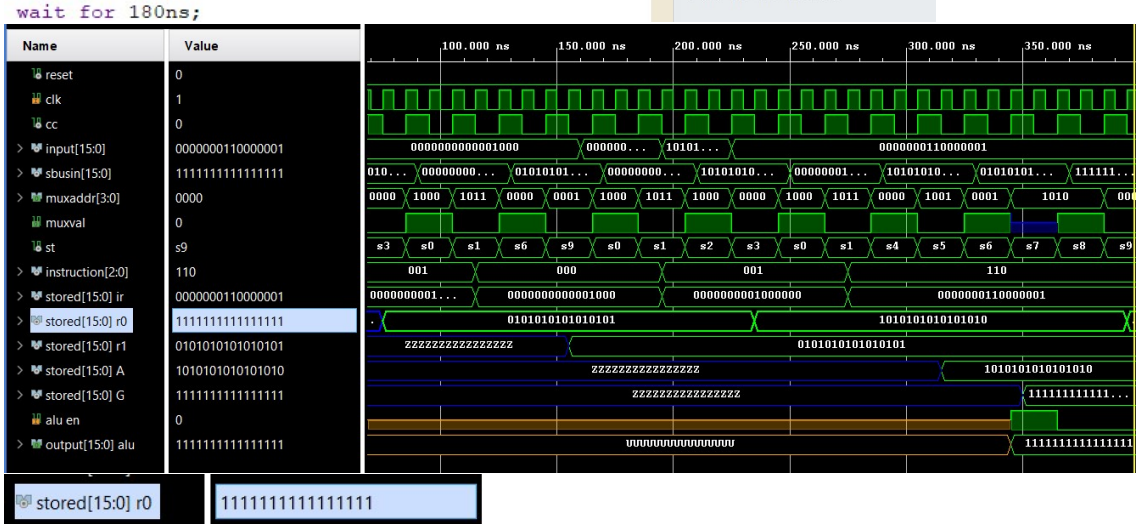




e.XOR

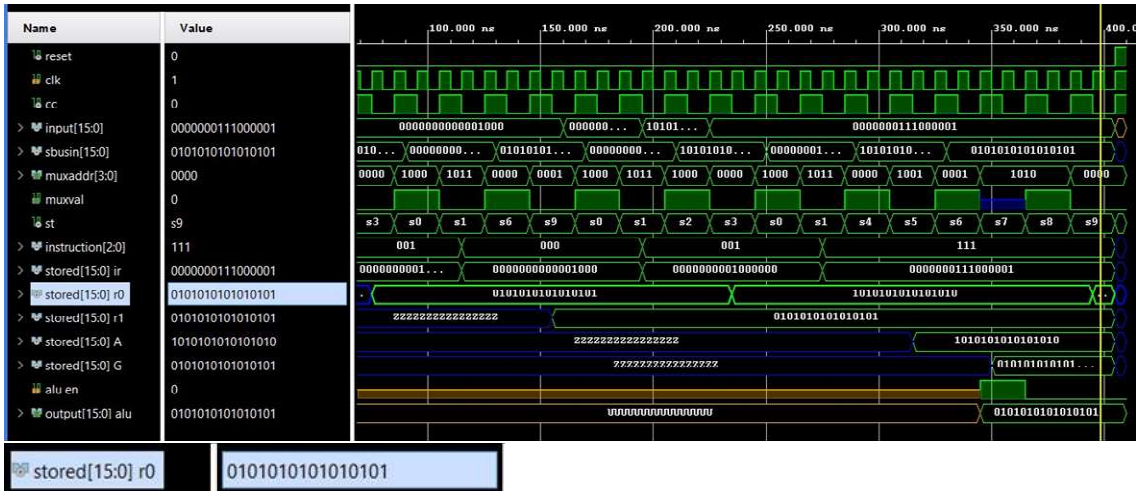
```
--Input mvi
input <= "0000000001000000" ;
wait for 35ns;
input <= "0101010101010101";
wait for 30ns;
--Input mv
input <= "0000000000001000";
wait for 95ns;
--Input mvi
input <= "0000000001000000";
wait for 35ns;
input <= "1010101010101010";
wait for 30ns;
--performing "xor" on these and storing it to R0
input <= "0000000110000001";
wait for 180ns;
```





f.NOT

```
--Input mvi
input <= "0000000001000000" ;
wait for 35ns;
input <= "0101010101010101";
wait for 30ns;
--Input mv
input <= "0000000000001000";
wait for 95ns;
--Input mvi
input <= "0000000001000000";
wait for 35ns;
input <= "1010101010101010";
wait for 30ns;
--performing "not" on R0 and storing it to R0
input <= "0000000111000001";
wait for 180ns;
```



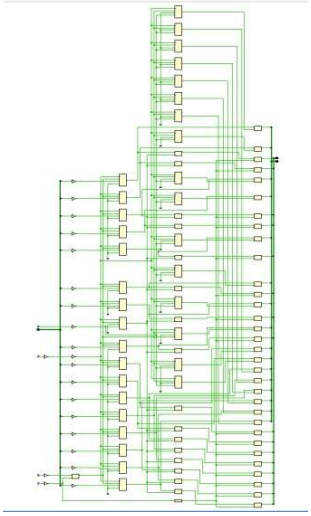
Hence we got the required results (correct results).

Discussion

The synthesis, implementation and RTL schematics are as below

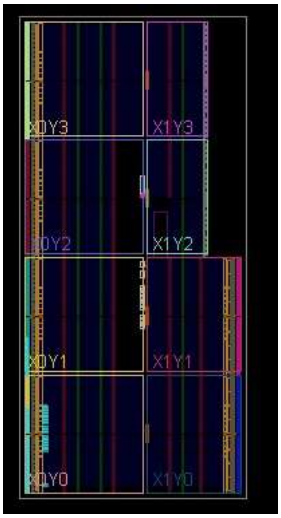
i. Registers

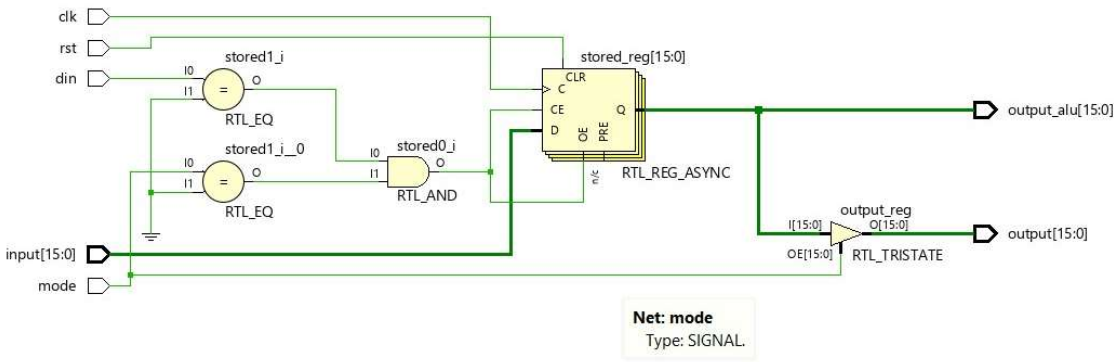
Implementation schematic



RTL schematic

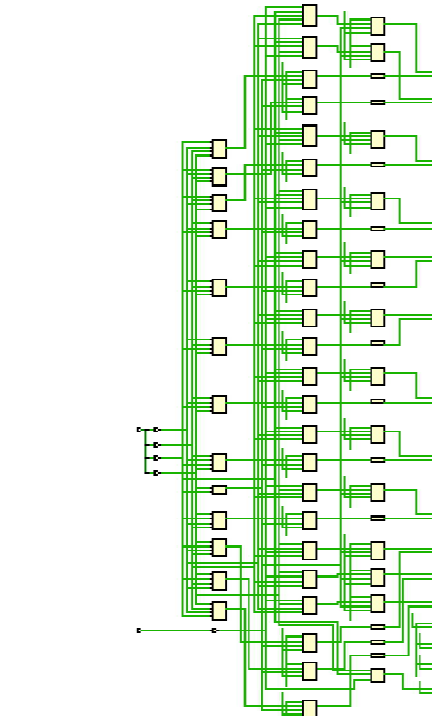
Device





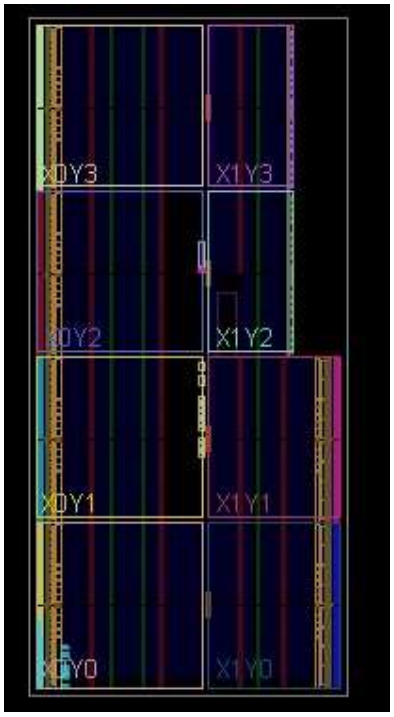
ii. Decoder/ rev Mux

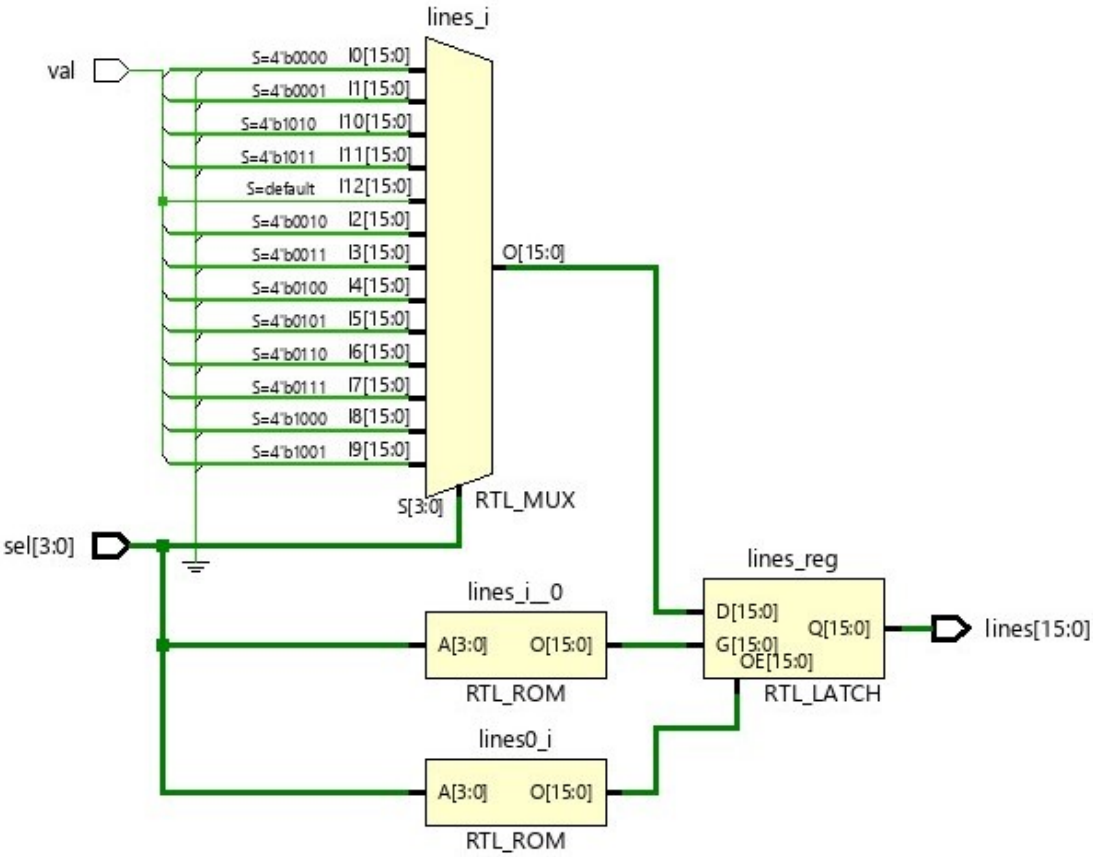
Implementation Schematic



RTL schematic

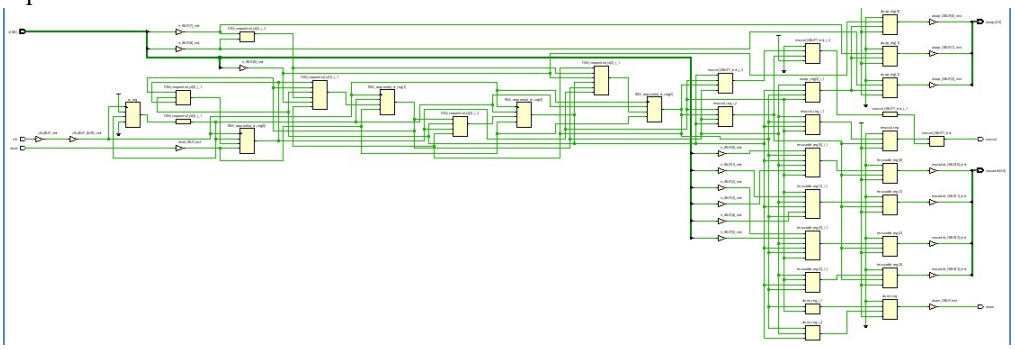
Device



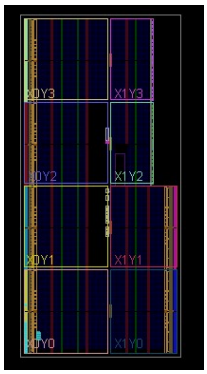


iii. Control Unit

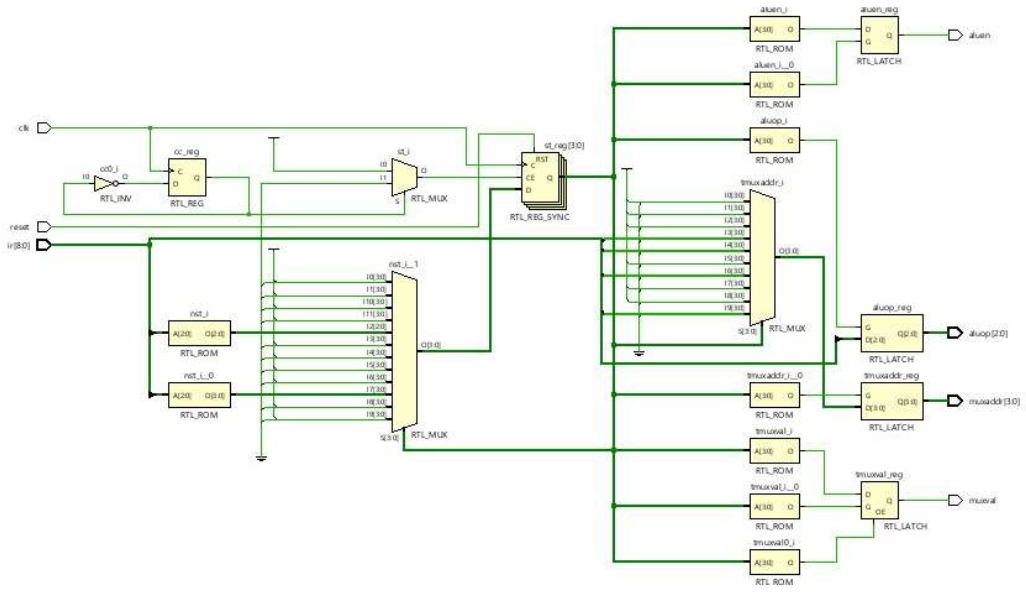
Implementation Schematic



Device

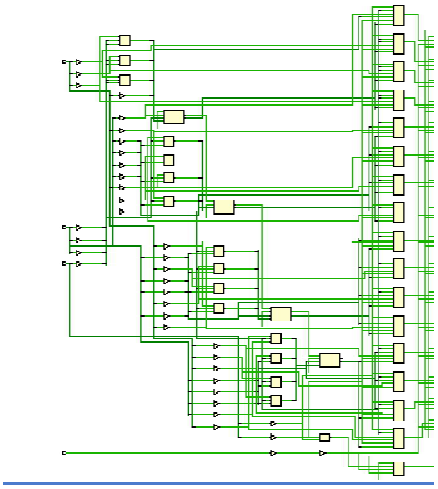


RTL schematic

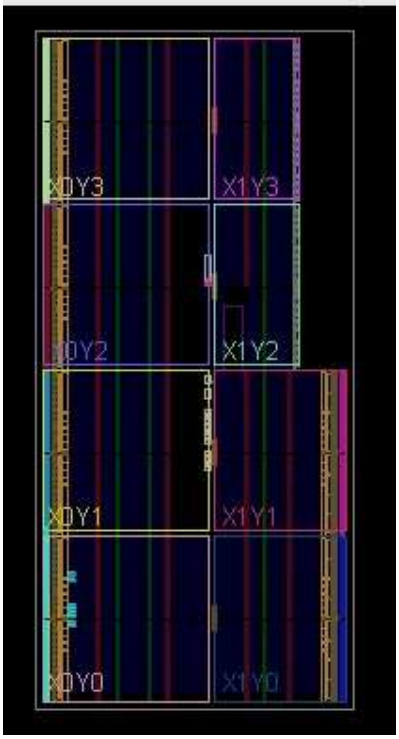


iv. ALU

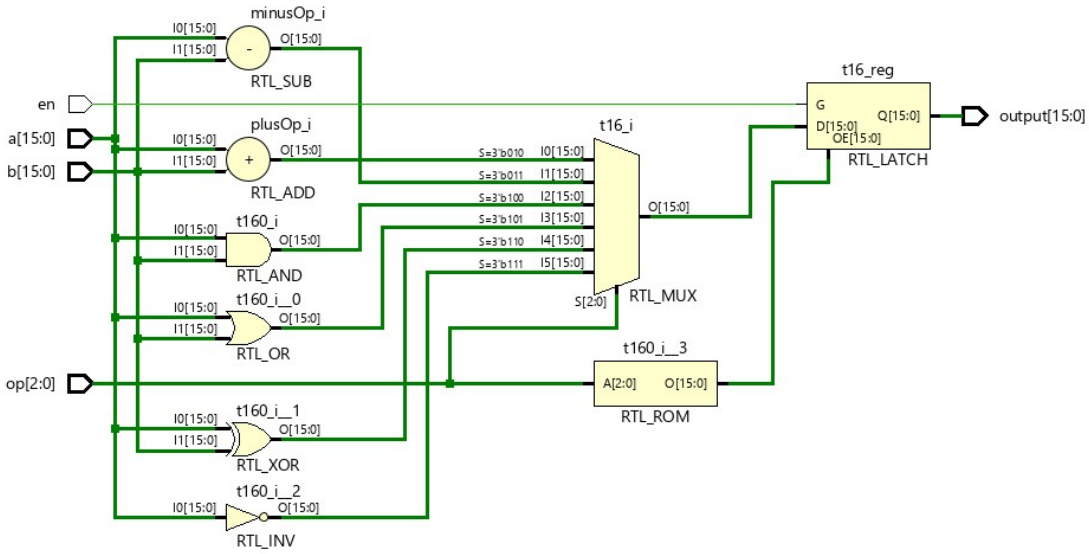
Implementation Schematic



Device

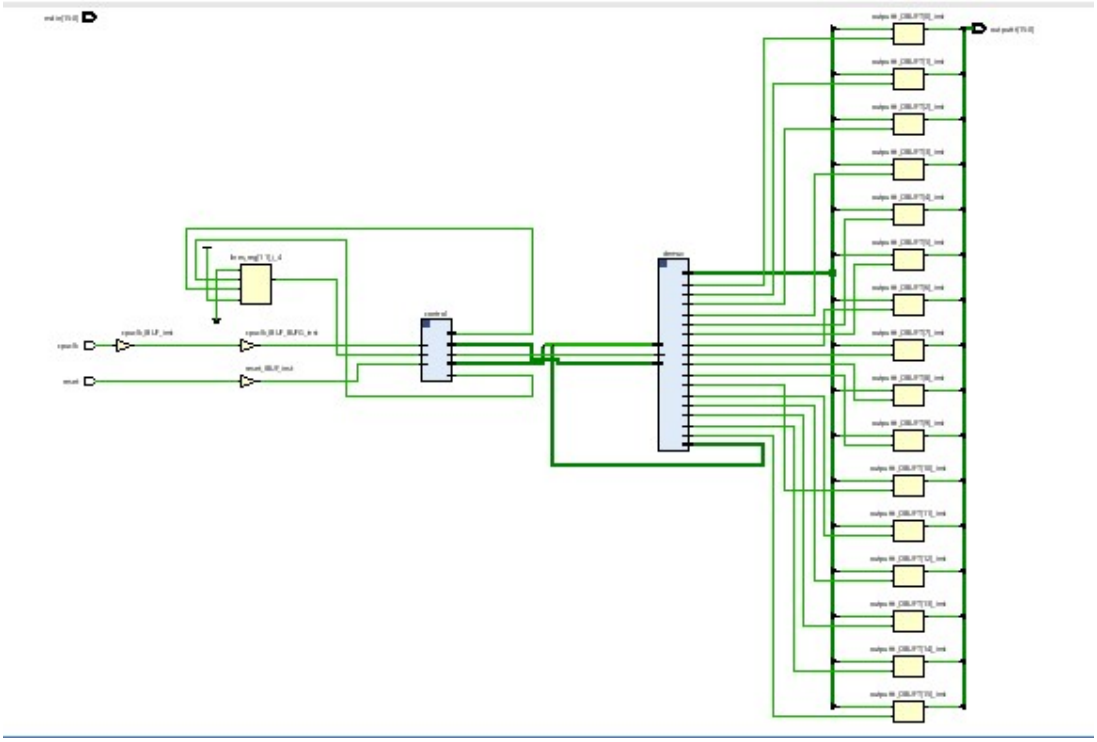


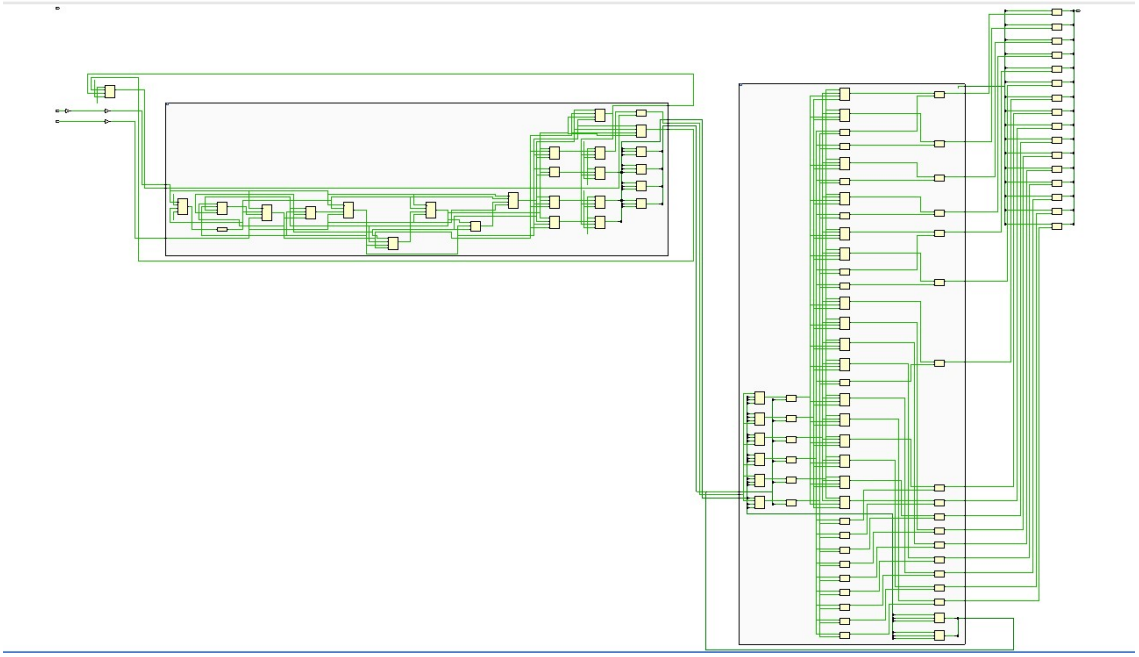
RTL schematic



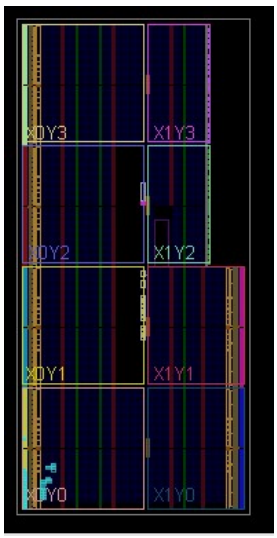
v. CPU

Implementation Schematic

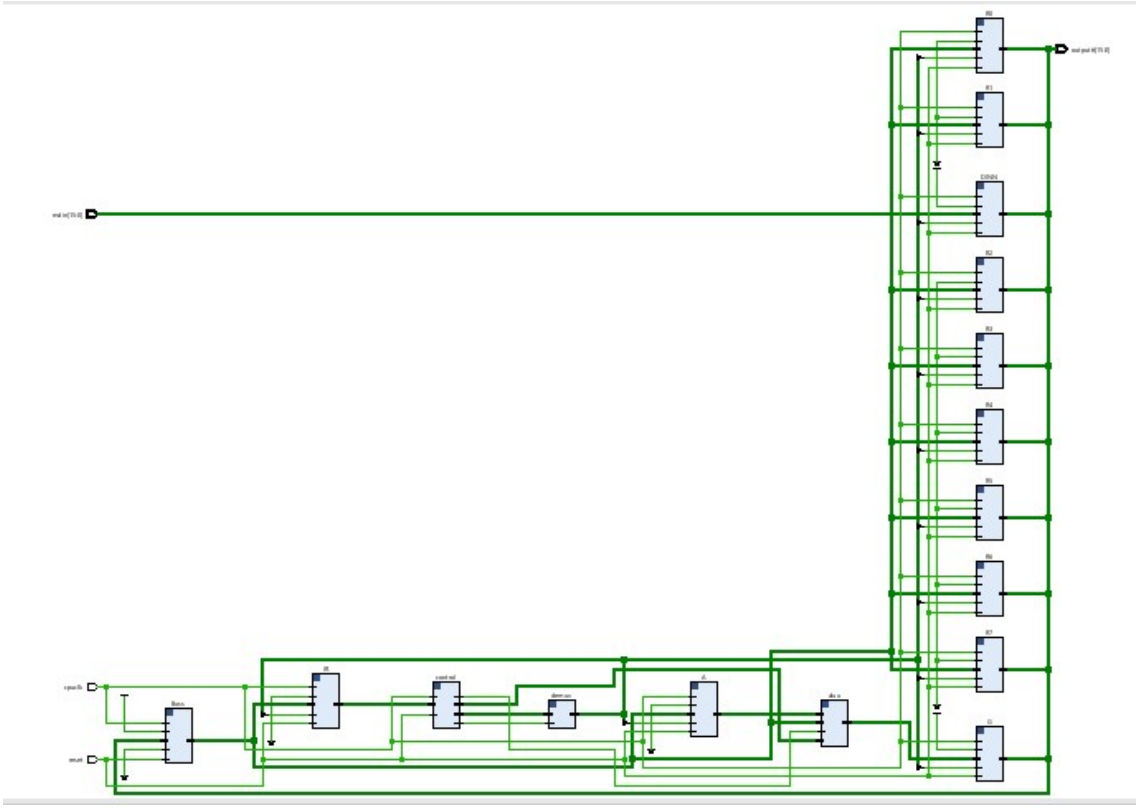


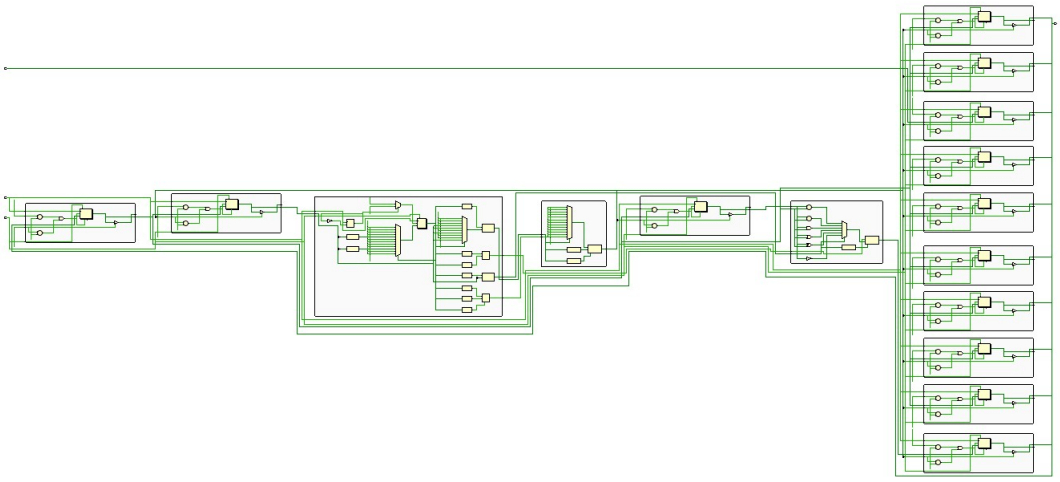


Device



RTL schematic of this structural code is





Note:

- For Simulations check the result waveforms in testbench, which contains working of all these components. Or open the project folder in to the submitted zip using VIVADO and run the simulation.
- The source code files can be found in “\Lab6_V2\Lab6_V2.srcs\sources_1\new” folder.

Inferences of Discussion

From the testbench results (inferences of clock cycles and correctness of processor)

- We can verify the correctness of each operation. The screenshot attached next to code is taken from the calculator. We can observe that our result matches.
- For the mv, mvi operations it took 8 clock cycles to complete. (As we used half frequency clock for FSM and normal frequency clock for registers. So, to complete 4 states in FSM it takes 8 clock cycles)
- For the rest of operation which require the ALU, it took 8 clock cycles of FSM i.e. 16 clock cycles to complete.

Conclusion

By this lab project we are able to understand the complete working of a simple processor and able to implement it in VHDL.

(And also

- Compared and verified our results of Operations
- Analysed the number of required clock cycles and triggering.)

References

[1] https://www.partitionwizard.com/partitionmanager/cpu-architecture.html

The screenshots of Binary calculator are taken from https://www.rapidtables.com/calc/math/binarycalculator.html

All the rest of the images are of my own simulations and drawings.

-Thank you-