CS1050 Computer Organization and Digital Design

Lab 9-10 – Nanoprocessor Design Competition

Name	Index Number
Ranathunga N. D.	**200517U**
Keerthichandra H. M. P. M.	**200304N**
Marasinghe M. M. R. S.	**200382A**
Vihidun D. P. T.	**200682T**
Marium M. S. S.	**200384G**

Lab Task:

In this Lab, we were assigned to design and simulate a Nanoprocessor to execute several Assembly operations. Nanoprocessor was developed using components mentioned below.

Program Counter

- Specifically designed for this Lab

Program ROM

- designed by modifying the LUT in Lab 07

Register Bank

- Specifically designed for this Lab
- Register bank consists of seven 4-bit registers
- The Assembly Language code was converted to a 12-bit signal and stored in the Register Bank.

4-bit Add/Sub Unit

- designed by modifying 4-bit RCA in Lab 03
- The Add/Sub unit is capable of adding and subtracting two 4-bit numbers, using Two's complement method.

3-bit Adder -

- An optimized version was specifically designed for this Lab

8-way 4-bit Mux

- An optimized version was specifically designed for this Lab

2-way 3-bit Mux

- An optimized version was specifically designed for this Lab

Load Select Mux

- An optimized version was specifically designed for this Lab

Several buses were used to connect components together, including Instruction Bus(12-bit), Data bus(4-bit), Address to Jump(3-bit).

Vivado Simulator tool was used to verify the functionality of the Nanoprocessor, before implementing into BASYS3 board.

Assembly Code and its Machine Code:

- **Assembly Program**

Binary instructions stored in the program counter consist of 12 bits. Those instructions have the following structure.

Instruction	Description	Format (12-bit instruction)
MOVI R, *d*	Move immediate value *d* to register R, i.e., *R* ← *d* R ∈ [0, 7], *d* ∈ [0, 15]	1 0 R R R 0 0 0 d d d d
ADD Ra, Rb	Add values in registers Ra and Rb and store the result in Ra, i.e., Ra ← Ra + Rb Ra, Rb ∈ [0, 7]	0 0 Ra Ra Ra Rb Rb Rb 0 0 0 0
NEG R	2's complement of registers R, i.e., R ← – R R ∈ [0, 7]	0 1 R R R 0 0 0 0 0 0 0
JZR R, d	Jump if value in register R is 0, i.e., If R == 0 PC ← d; Else PC ← PC + 1; R ∈ [0, 7], *d* ∈ [0, 7]	1 1 R R R 0 0 0 0 d d d

- **Assembly Code**

The assembly code stored in the program counter is given below.

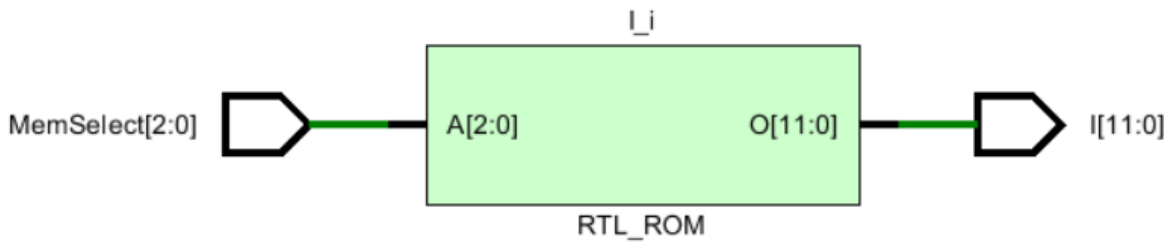
```
------------------ Operations -----------------------------
"100010001010", --- MOVI R1, 10 ; R1 <= 10
"100100000001", --- MOVI R2, 1 ; R2 <= 1
"010100000000", --- NEG R2 ; R2 <= -R2
"000010100000", --- ADD R1, R2 ; R1 <= R1 + R2
"110010000000", --- JZR R1, 7 ; If R1 = 0 jump to line 7
"110000000011" --- JZR R0, 3 ; If R0 = 0 jump to line 3
```

Program ROM:

VHDL Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity ProgramROM is
    Port ( MemSelect : in STD_LOGIC_VECTOR (2 downto 0);
           I : out STD_LOGIC_VECTOR (11 downto 0));
end ProgramROM;

architecture Behavioral of ProgramROM is
type rom_type is array(0 to 5) of std_logic_vector(11 downto 0);
signal ProROM : rom_type := (
------------------ Operations -----------------------------
"100010001010", --- MOVI R1, 10 ; R1 <= 10
"100100000001", --- MOVI R2, 1 ; R2 <= 1
"010100000000", --- NEG R2 ; R2 <= -R2
"000010100000", --- ADD R1, R2 ; R1 <= R1 + R2
"110010000000", --- JZR R1, 7 ; If R1 = 0 jump to line 7
"110000000011" --- JZR R0, 3 ; If R0 = 0 jump to line 3
);

begin
I <= ProROM(to_integer(unsigned(MemSelect)));

end Behavioral;
```

Schematic Diagram



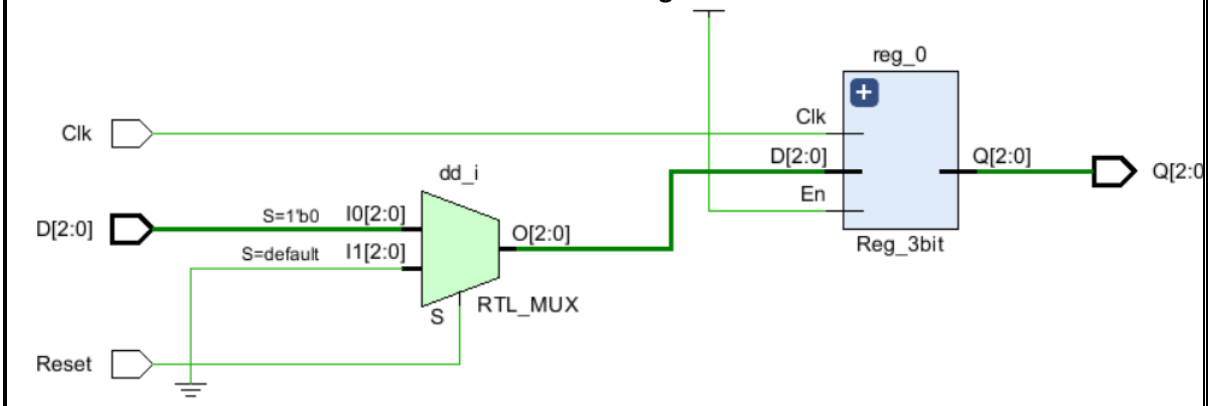
Program Counter:

VHDL Code

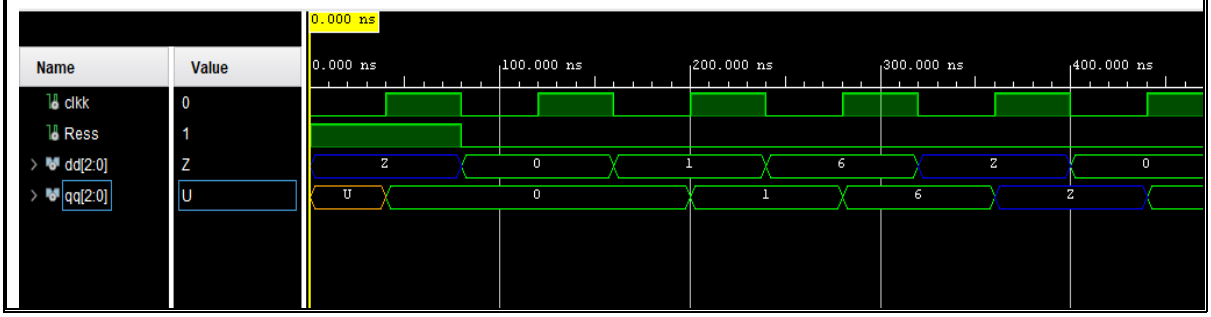
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Counter is
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           D: in STD_LOGIC_VECTOR(2 downto 0);
           Q : out STD_LOGIC_VECTOR(2 downto 0));
end Counter;
```

```
architecture Behavioral of Counter is

component Reg_3bit
    Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
           En : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (2 downto 0));
end component;

signal dd: std_logic_vector (2 downto 0);
signal qq: std_logic_vector (2 downto 0);

begin

reg_0: Reg_3bit
    PORT MAP(
        Clk => Clk,
        En => '1',
        Q => Q,
        D => dd
    );

with Reset select dd <= d when '0',
                        d when 'U',
                        "000" when others;

end Behavioral;
```

Schematic Diagram



Timing Diagram



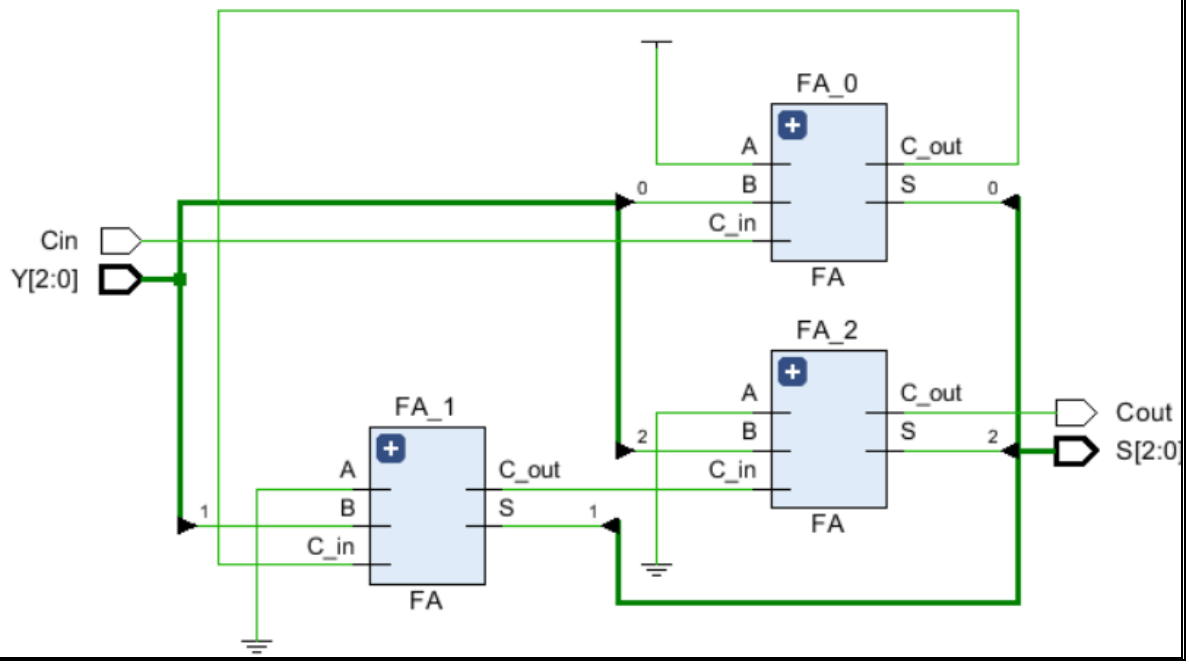
3-bit Adder:

VHDL Code

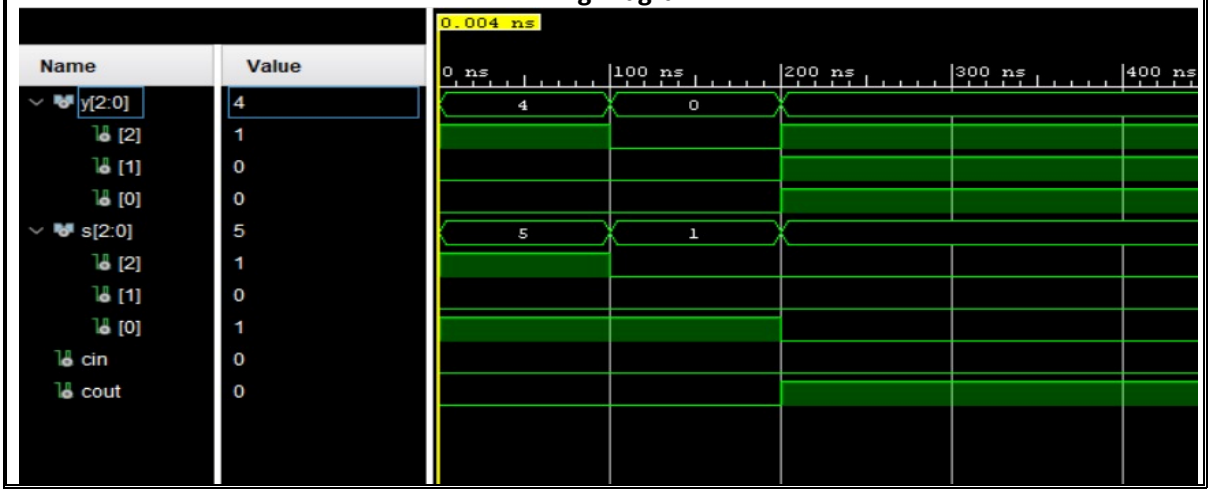
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity RCA is
    Port (Y : in STD_LOGIC_VECTOR (2 downto 0);
          S : out STD_LOGIC_VECTOR (2 downto 0);
          Cin : in STD_LOGIC;
          Cout : out STD_LOGIC);
end RCA;

architecture Behavioral of RCA is

component FA
Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C_in : in STD_LOGIC;
           S : out STD_LOGIC;
           C_out : out STD_LOGIC);
end component;

signal  FA_cout : STD_LOGIC_VECTOR (1 downto 0);



begin


FA_0: FA

PORT MAP(
A=>'1',
B=>Y(0),
C_in=>Cin,
S=>S(0),
C_out=>FA_cout(0)
);

FA_1: FA

PORT MAP(
A=>'0',
B=>Y(1),
C_in=>FA_cout(0),
S=>S(1),
C_out=>FA_cout(1)
);

FA_2: FA

PORT MAP(
A=>'0',
B=>Y(2),
```

```
C_in=>FA_cout(1),
S=>S(2),
C_out=>Cout
);


end Behavioral;
```

Schematic Diagram



Timing Diagram



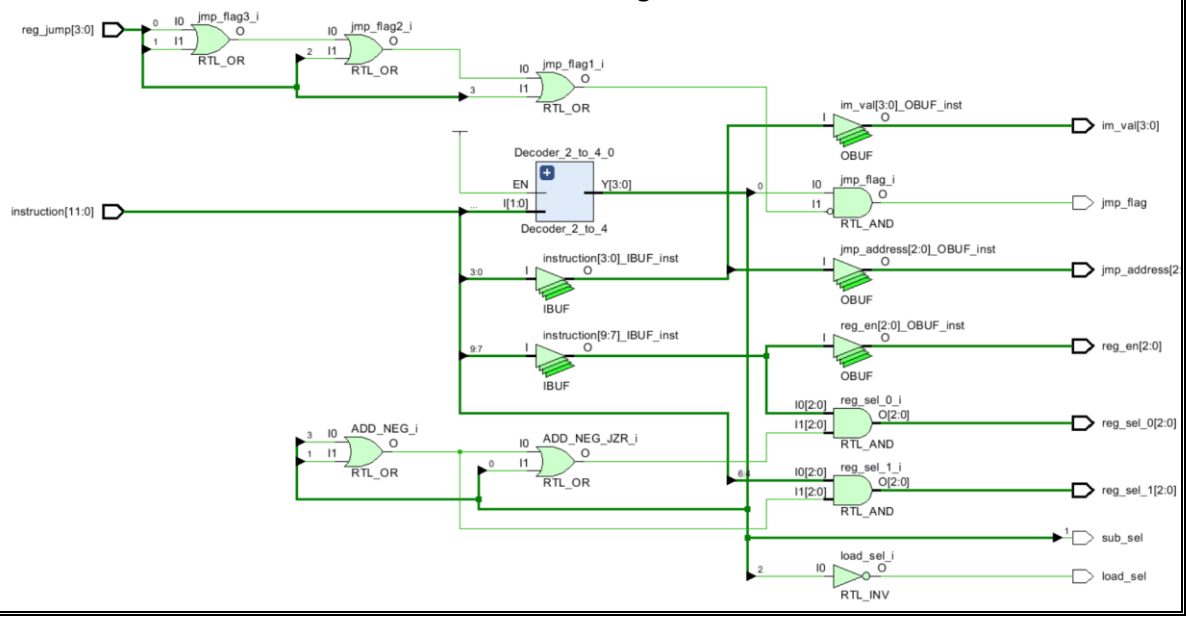
Instruction Decoder:

VHDL Code

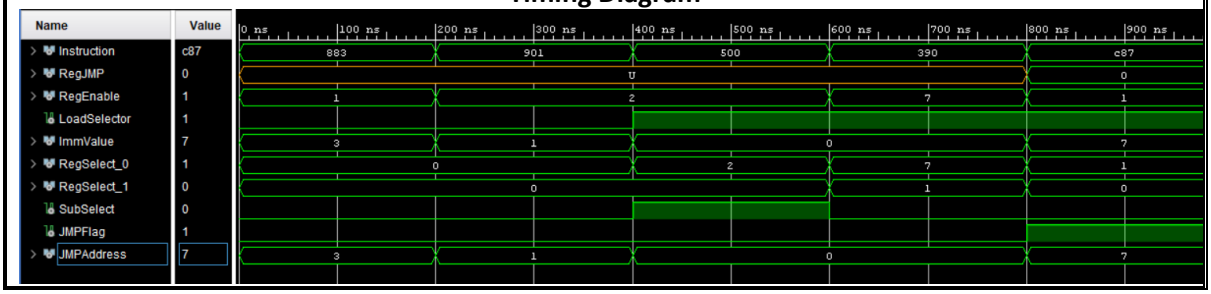
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Instruction_Decoder is
    Port ( instruction : in STD_LOGIC_VECTOR (11 downto 0);
           reg_jump : in STD_LOGIC_VECTOR (3 downto 0);
           reg_en : out STD_LOGIC_VECTOR (2 downto 0);
           load_sel : out STD_LOGIC;
           im_val : out STD_LOGIC_VECTOR (3 downto 0);
           reg_sel_0 : out STD_LOGIC_VECTOR (2 downto 0);
           reg_sel_1 : out STD_LOGIC_VECTOR (2 downto 0);
           sub_sel : out STD_LOGIC;
           jmp_flag : out STD_LOGIC;
           jmp_address : out STD_LOGIC_VECTOR (2 downto 0));
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is

component Decoder_2_to_4
    Port ( I : in STD_LOGIC_VECTOR (1 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0);
           EN : in STD_LOGIC);
end component;

component Tri_state_buffer3
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           Y : out STD_LOGIC_VECTOR (2 downto 0);
           EN : in STD_LOGIC);
end component;

signal ADD, MOVI, NEG, JZR: STD_LOGIC;
signal ADD_NEG, ADD_NEG_JZR: STD_LOGIC;
signal reg_sel_0_activate, reg_sel_1_activate: STD_LOGIC_VECTOR (2
downto 0);

begin

Decoder_2_to_4_0: Decoder_2_to_4
    PORT MAP(
        I => instruction(11 downto 10),
        EN => '1',
        Y(3) => ADD,
        Y(2) => MOVI,
        Y(1) => NEG,
        Y(0) => JZR
    );

reg_en <= instruction(9 downto 7);
load_sel <= not MOVI;
im_val <= instruction(3 downto 0);

ADD_NEG <= ADD or NEG;
```

```
ADD_NEG_JZR <= ADD_NEG or JZR;

reg_sel_1_activate <= (ADD_NEG, ADD_NEG, ADD_NEG);
reg_sel_0_activate <= (ADD_NEG_JZR, ADD_NEG_JZR, ADD_NEG_JZR);

reg_sel_0 <= instruction(9 downto 7) and reg_sel_0_activate;
reg_sel_1 <= instruction(6 downto 4) and reg_sel_1_activate;

sub_sel <= NEG;

jmp_flag <= JZR and not(reg_jump(0) or reg_jump(1) or reg_jump(2)
or reg_jump(3));
jmp_address <= instruction(2 downto 0);

end Behavioral;
```

Schematic Diagram



Timing Diagram

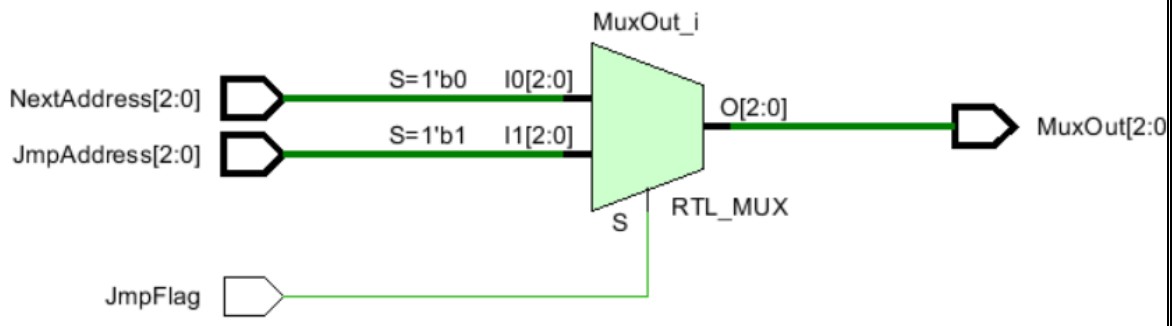


2-way 3-bit Mux:

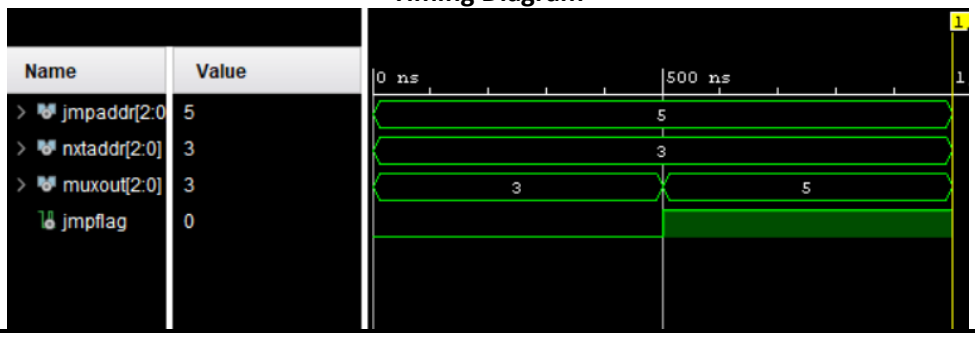
VHDL Code

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

entity mux_2way_3bit is
    Port ( JmpAddress : in STD_LOGIC_VECTOR (2 downto 0);
           NextAddress : in STD_LOGIC_VECTOR (2 downto 0);
           JmpFlag : in STD_LOGIC;
           MuxOut : out STD_LOGIC_VECTOR (2 downto 0));
end mux_2way_3bit;

architecture Behavioral of mux_2way_3bit is

begin

with JmpFlag select MuxOut <= NextAddress when '0',
                              JmpAddress when '1',
                              nextaddress when others;

end Behavioral;
```

Schematic Diagram



Timing Diagram

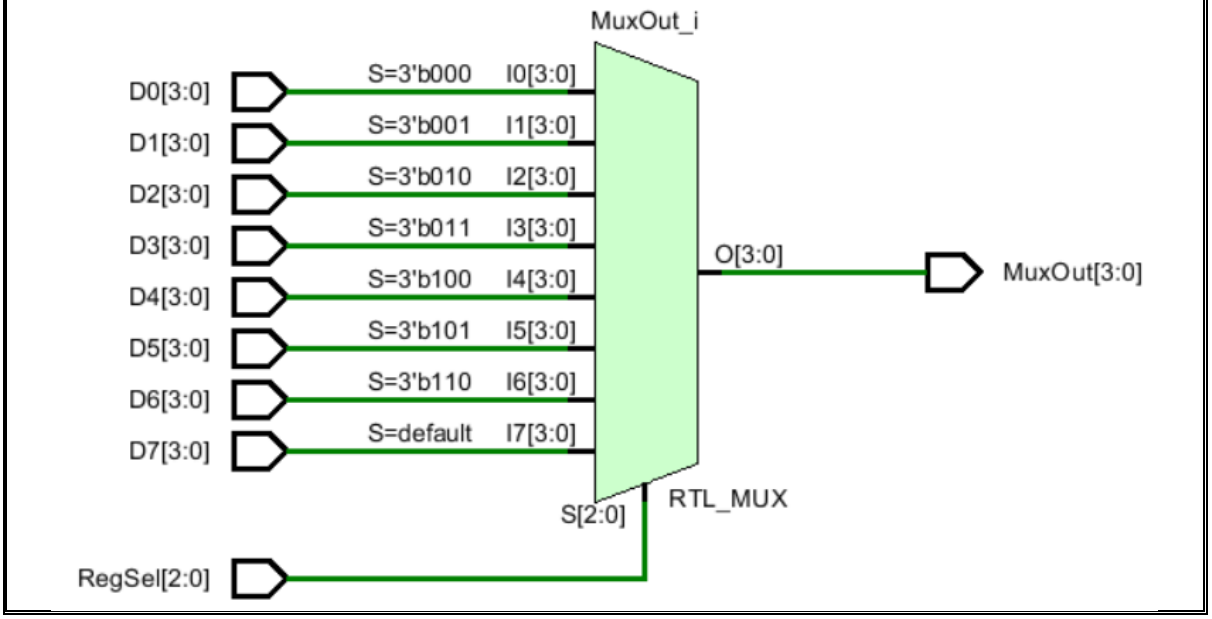


8-way 4-bit Mux:

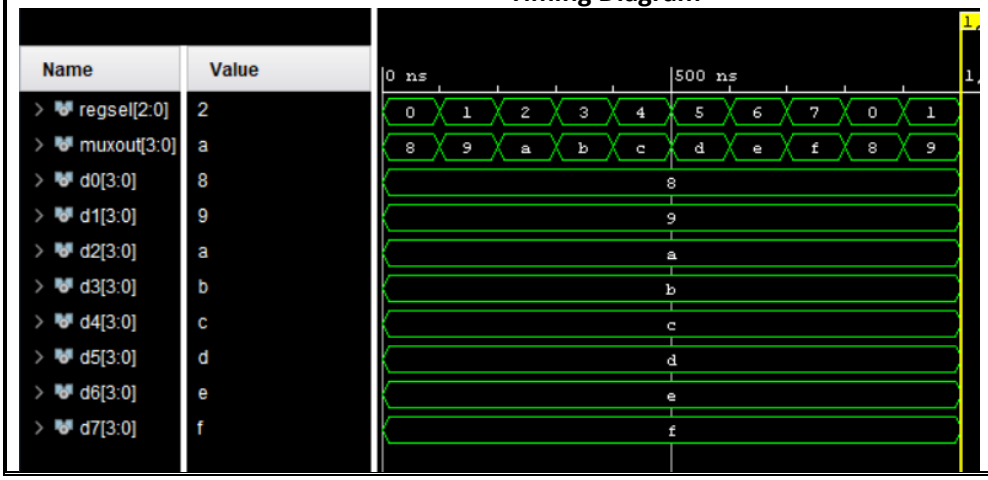
VHDL Code

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

entity mux_8way_4bit is
    Port ( RegSel : in STD_LOGIC_VECTOR (2 downto 0);
           MuxOut : out STD_LOGIC_VECTOR (3 downto 0);
           D0 : in STD_LOGIC_VECTOR (3 downto 0);
           D1 : in STD_LOGIC_VECTOR (3 downto 0);
           D2 : in STD_LOGIC_VECTOR (3 downto 0);
           D3 : in STD_LOGIC_VECTOR (3 downto 0);
           D4 : in STD_LOGIC_VECTOR (3 downto 0);
           D5 : in STD_LOGIC_VECTOR (3 downto 0);
           D6 : in STD_LOGIC_VECTOR (3 downto 0);
           D7 : in STD_LOGIC_VECTOR (3 downto 0));
end mux_8way_4bit;

architecture Behavioral of mux_8way_4bit is

begin

with RegSel select MuxOut <= D0 when "000",
                             D1 when "001",
                             D2 when "010",
                             D3 when "011",
                             D4 when "100",
                             D5 when "101",
                             D6 when "110",
                             D7 when others;

end Behavioral;
```

Schematic Diagram



Timing Diagram

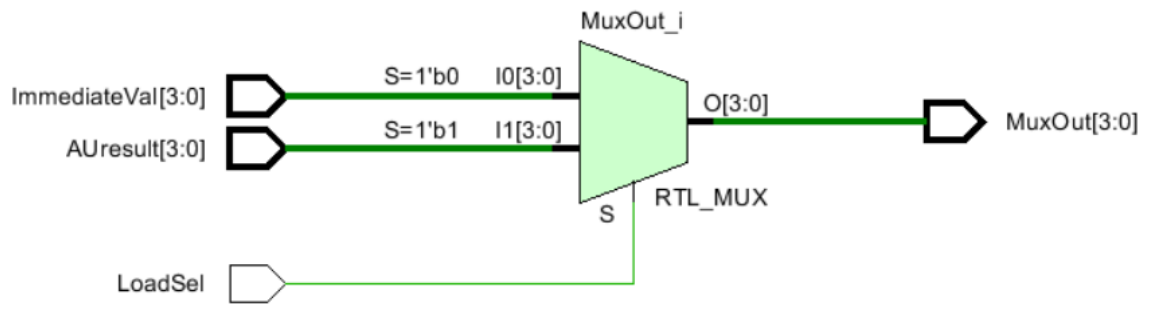


Load select Mux:

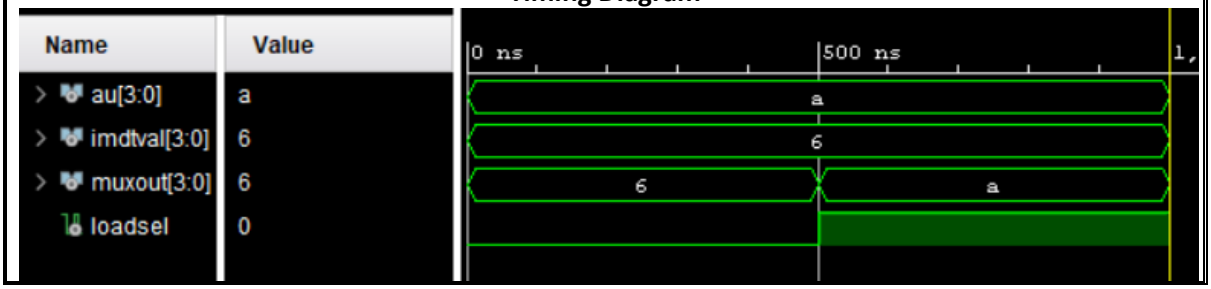
VHDL Code

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

entity mux_LoadSelect is
    Port ( LoadSel : in STD_LOGIC;
           AUresult : in STD_LOGIC_VECTOR (3 downto 0);
           ImmediateVal : in STD_LOGIC_VECTOR (3 downto 0);
           MuxOut : out STD_LOGIC_VECTOR (3 downto 0));
end mux_LoadSelect;

architecture Behavioral of mux_LoadSelect is

begin

with LoadSel select MuxOut <= ImmediateVal when '0',
                              AUresult when '1',
                              "ZZZZ" when others;

end Behavioral;
```

Schematic Diagram



Timing Diagram

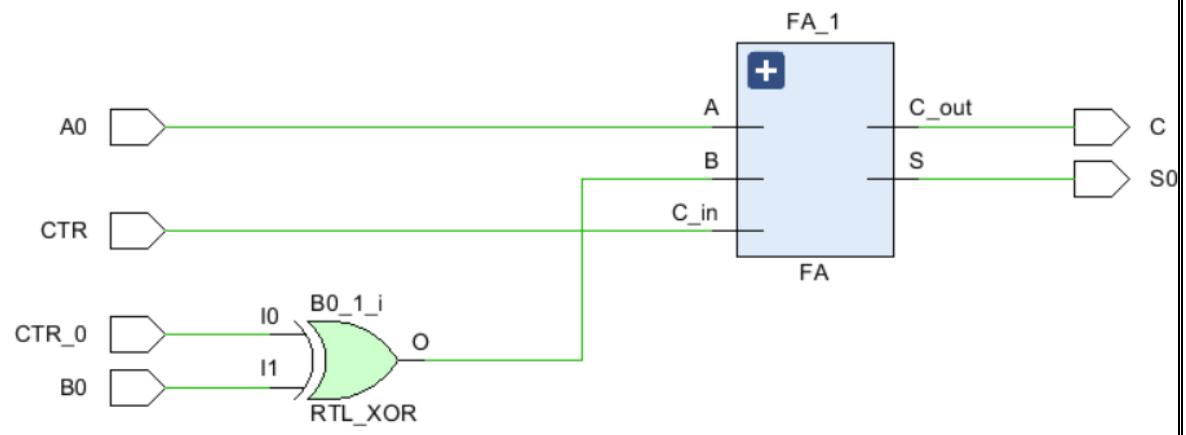


ADD_SUB Unit:

VHDL Code

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

entity ADD_SUB is
    Port ( CTR : in STD_LOGIC;
           A0 : in STD_LOGIC;
           B0 : in STD_LOGIC;
           CTR_0 : in STD_LOGIC;
           C : out STD_LOGIC;
           S0 : out STD_LOGIC);
end ADD_SUB;

architecture Behavioral of ADD_SUB is
component FA
    port (
     A : in STD_LOGIC;
     B : in STD_LOGIC;
     C_in : in STD_LOGIC;
     C_out : out STD_LOGIC;
     S : out STD_LOGIC
    );
end component;

signal B0_1: STD_LOGIC;

begin
FA_1 : FA
PORT MAP(
    A => A0,
    B => B0_1 ,
    C_in => CTR,
    C_out => C,
    S => S0
);

    B0_1 <= CTR_0 XOR B0;

end Behavioral;
```

Schematic Diagram



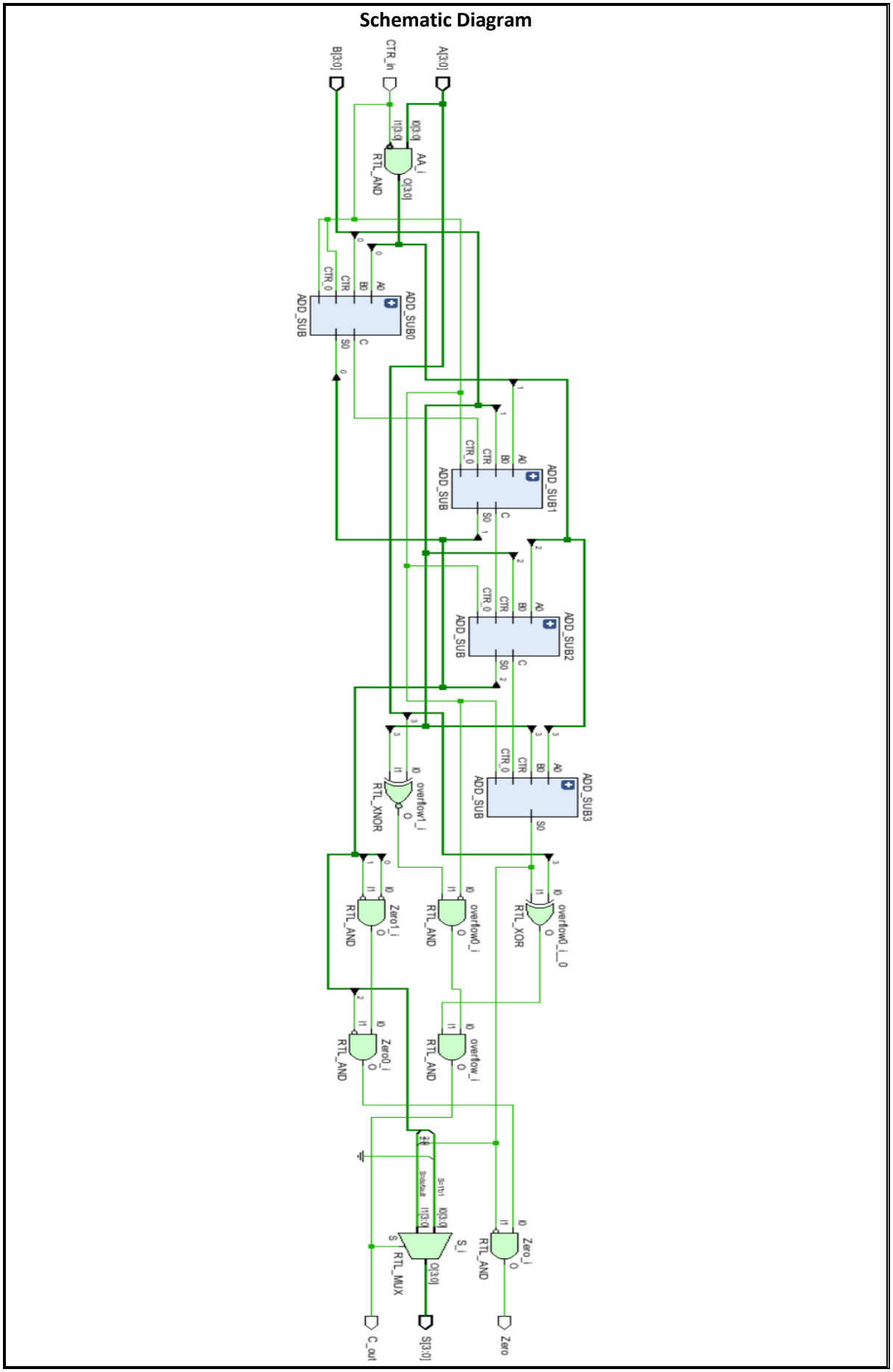
4-bit RC Add/Sub Unit:

VHDL Code

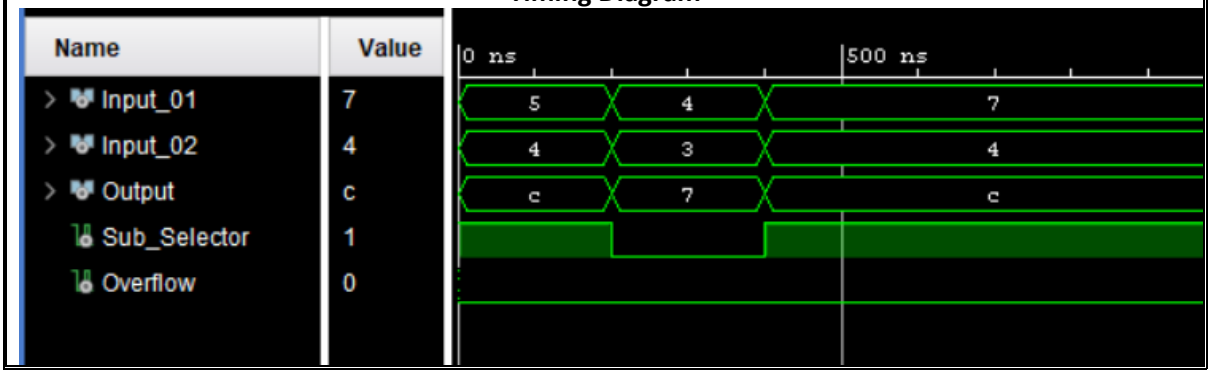
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity RCA_S is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           CTR_in : in STD_LOGIC;
           C_out : out STD_LOGIC;
           S : out STD_LOGIC_VECTOR (3 downto 0);
           Zero: out STD_LOGIC);
end RCA_S;

architecture Behavioral of RCA_S is

component ADD_SUB
     Port ( CTR : in STD_LOGIC;
           A0 : in STD_LOGIC;
           B0 : in STD_LOGIC;
           CTR_0 : in STD_LOGIC;
           C : out STD_LOGIC;
           S0 : out STD_LOGIC);
end component;

signal C0,C1,C2,C3, overflow : STD_LOGIC;
signal AA, NEG: STD_LOGIC_VECTOR (3 downto 0);
signal ss: STD_LOGIC_VECTOR (3 downto 0);
begin

NEG <= (CTR_in, CTR_in, CTR_in, CTR_in);
AA <= A and not NEG;

ADD_SUB0 : ADD_SUB
    PORT MAP(
        CTR => CTR_in,
        A0 => AA(0),
        B0 => B(0),
```

```
        CTR_0 => CTR_in,
        C => C0,
        S0 => Ss(0)
    );

ADD_SUB1 : ADD_SUB
    PORT MAP(
        CTR => C0,
        A0 => AA(1),
        B0 => B(1),
        CTR_0 => CTR_in,
        C => C1,
        S0 => Ss(1)
    );

ADD_SUB2 : ADD_SUB
    PORT MAP(
        CTR => C1,
        A0 => AA(2),
        B0 => B(2),
        CTR_0 => CTR_in,
        C => C2,
        S0 => Ss(2)
    );

ADD_SUB3 : ADD_SUB
    PORT MAP(
        CTR => C2,
        A0 => AA(3),
        B0 => B(3),
        CTR_0 => CTR_in,
        C => C3,
        S0 => Ss(3)
    );

zero <= not Ss(0) and not ss(1) and not ss(2) and not ss(3);
overflow <= not CTR_in and (A(3) XNOR B(3)) and (A(3) XOR ss(3));
with overflow select s <= ('0', ss(2), ss(1), ss(0)) when '1',
                          ss when others;
C_Out <= overflow;
end Behavioral;
```

Schematic Diagram



Timing Diagram

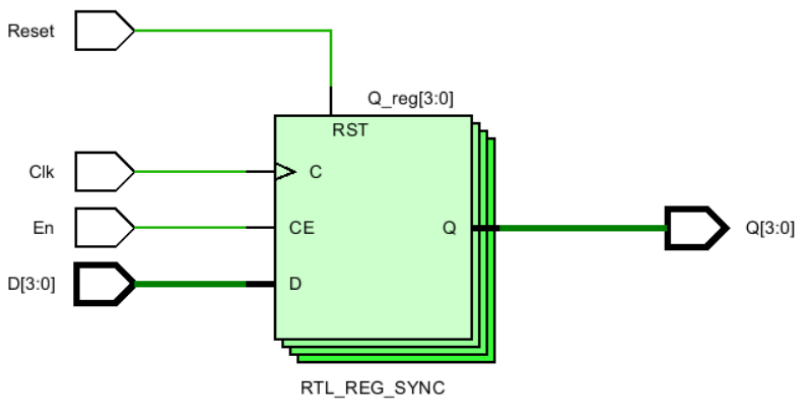


Register:

VHDL Code

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

entity Reg is
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
           En : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end Reg;

architecture Behavioral of Reg is

begin

process(Clk) begin

    if (rising_edge(Clk)) then --respond when clock rises
       if Reset = '1' then
            Q <= "0000";
       elsif En = '1' then          -- Enable should be set
            Q <= D;
       end if;
    end if;
end process;

end Behavioral;
```

Schematic Diagram



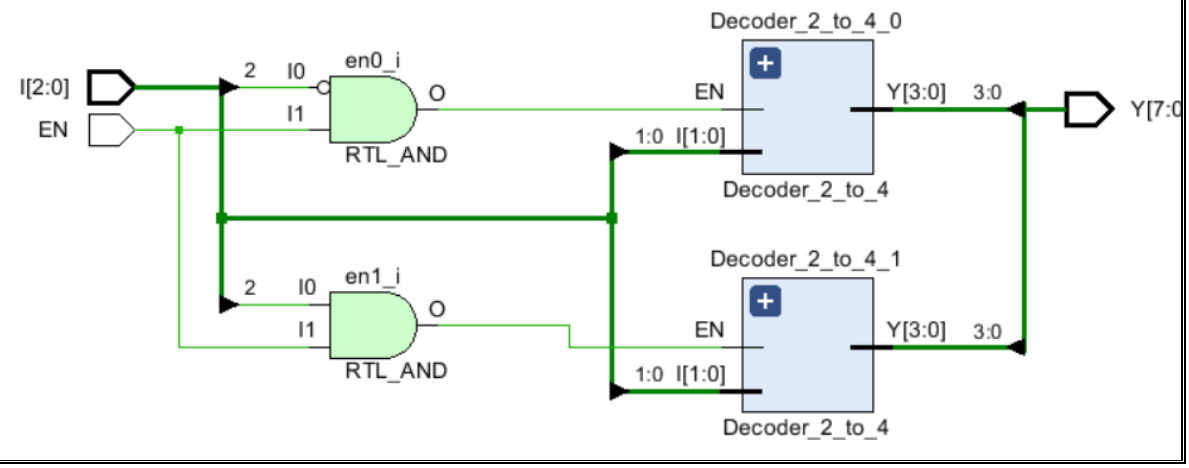
3-8 Decoder:

VHDL Code

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

entity Decoder_3_to_8 is
    Port ( I : in STD_LOGIC_VECTOR (2 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (7 downto 0));
end Decoder_3_to_8;

architecture Behavioral of Decoder_3_to_8 is
component decoder_2_to_4
    Port ( I : in STD_LOGIC_VECTOR (1 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0);
           EN : in STD_LOGIC);
end component;

signal I0,I1 : STD_LOGIC_VECTOR (1 downto 0);
signal Y0,Y1 : STD_LOGIC_VECTOR (3 downto 0);
signal en0,en1,I2 : STD_LOGIC;

begin

    Decoder_2_to_4_0 : Decoder_2_to_4
        port map(
            I => I0,
            EN => en0,
            Y => Y0
        );
    Decoder_2_to_4_1 : Decoder_2_to_4
        port map(
            I => I1,
            EN => en1,
            Y => Y1
        );
    en0 <= NOT(I(2)) AND EN;
    en1 <= I(2) AND EN;
    I0 <= I(1 downto 0);
```

```
    I1 <= I(1 downto 0);
    --I2 <= I(2);
    Y(3 downto 0) <= Y0;
    Y(7 downto 4) <= Y1;

end Behavioral;
```

Schematic Diagram



Register Bank:

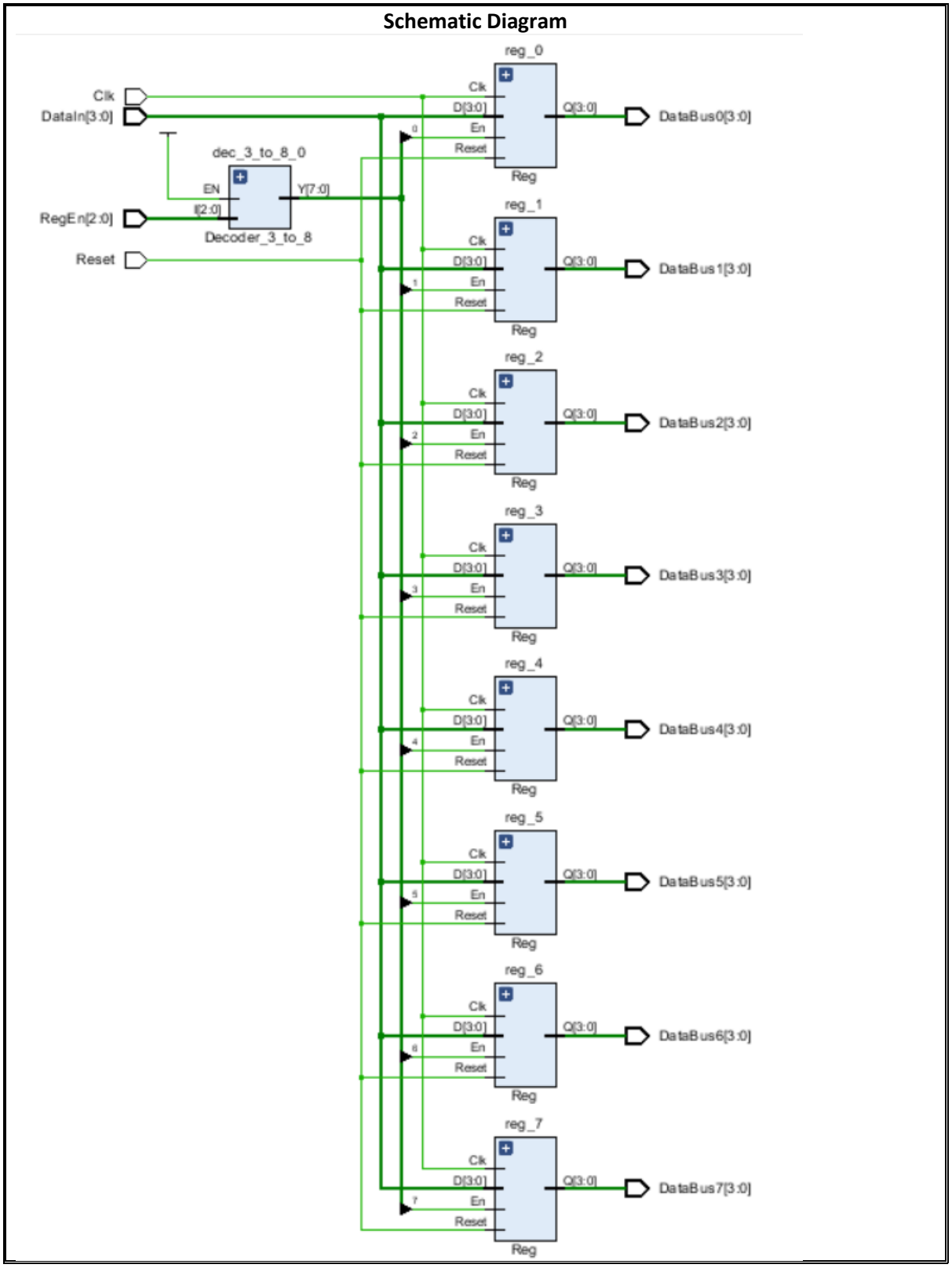
VHDL Code

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

entity RegBank is
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           RegEn : in STD_LOGIC_VECTOR (2 downto 0);
           DataIn : in STD_LOGIC_VECTOR (3 downto 0);
           DataBus0 : out STD_LOGIC_VECTOR (3 downto 0);
           DataBus1 : out STD_LOGIC_VECTOR (3 downto 0);
           DataBus2 : out STD_LOGIC_VECTOR (3 downto 0);
           DataBus3 : out STD_LOGIC_VECTOR (3 downto 0);
           DataBus4 : out STD_LOGIC_VECTOR (3 downto 0);
           DataBus5 : out STD_LOGIC_VECTOR (3 downto 0);
           DataBus6 : out STD_LOGIC_VECTOR (3 downto 0);
           DataBus7 : out STD_LOGIC_VECTOR (3 downto 0)
           );
end RegBank;

architecture Behavioral of RegBank is

component Reg
    Port(D : in STD_LOGIC_VECTOR (3 downto 0);
        En : in STD_LOGIC;
        Reset : in STD_LOGIC;
        Clk : in STD_LOGIC;
```

```
        Q : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component Decoder_3_to_8
    Port ( I : in STD_LOGIC_VECTOR (2 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (7 downto 0));
end component;

signal DecOut: STD_LOGIC_VECTOR (7 downto 0);

begin

dec_3_to_8_0: Decoder_3_to_8
    PORT MAP(
        EN => '1',
        I => RegEn,
        Y => DecOut
    );

reg_0: Reg
    PORT MAP(
        D => DataIn,
        Q => DataBus0,
        En => DecOut(0),
        Clk => Clk,
        reset => reset
    );

reg_1: Reg
    PORT MAP(
        D => DataIn,
        Q => DataBus1,
        En => DecOut(1),
        Clk => Clk,
        reset => reset
    );

reg_2: Reg
    PORT MAP(
        D => DataIn,
        Q => DataBus2,
        En => DecOut(2),
        Clk => Clk,
        reset => reset
    );

reg_3: Reg
    PORT MAP(
        D => DataIn,
        Q => DataBus3,
        En => DecOut(3),
        Clk => Clk,
        reset => reset
    );
```

```
reg_4: Reg
    PORT MAP(
        D => DataIn,
        Q => DataBus4,
        En => DecOut(4),
        Clk => Clk,
        reset => reset
    );

reg_5: Reg
    PORT MAP(
        D => DataIn,
        Q => DataBus5,
        En => DecOut(5),
        Clk => Clk,
        reset => reset
    );

reg_6: Reg
    PORT MAP(
        D => DataIn,
        Q => DataBus6,
        En => DecOut(6),
        Clk => Clk,
        reset => reset
    );

reg_7: Reg
    PORT MAP(
        D => DataIn,
        Q => DataBus7,
        En => DecOut(7),
        Clk => Clk,
        reset => reset
    );
end Behavioral;
```

Schematic Diagram



Nanoprocessor:

VHDL Code

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

entity Nanoprocessor is
    Port ( Clock : in STD_LOGIC;
           Reset : in STD_LOGIC;
           Overflow : out STD_LOGIC;
           Zero : out STD_LOGIC);
end Nanoprocessor;

architecture Behavioral of Nanoprocessor is

component Instruction_Decoder
    Port ( instruction : in STD_LOGIC_VECTOR (11 downto 0);
           reg_jump : in STD_LOGIC_VECTOR (3 downto 0);
           reg_en : out STD_LOGIC_VECTOR (2 downto 0);
           load_sel : out STD_LOGIC;
           im_val : out STD_LOGIC_VECTOR (3 downto 0);
           reg_sel_0 : out STD_LOGIC_VECTOR (2 downto 0);
           reg_sel_1 : out STD_LOGIC_VECTOR (2 downto 0);
           sub_sel : out STD_LOGIC;
           jmp_flag : out STD_LOGIC;
           jmp_address : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component ProgramROM
    Port ( MemSelect : in STD_LOGIC_VECTOR (2 downto 0);
           I : out STD_LOGIC_VECTOR (11 downto 0));
end component;

component RegBank
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           RegEn : in STD_LOGIC_VECTOR (2 downto 0);
           DataIn : in STD_LOGIC_VECTOR (3 downto 0);
           DataBus0 : out STD_LOGIC_VECTOR (3 downto 0);
           DataBus1 : out STD_LOGIC_VECTOR (3 downto 0);
           DataBus2 : out STD_LOGIC_VECTOR (3 downto 0);
           DataBus3 : out STD_LOGIC_VECTOR (3 downto 0);
           DataBus4 : out STD_LOGIC_VECTOR (3 downto 0);
           DataBus5 : out STD_LOGIC_VECTOR (3 downto 0);
           DataBus6 : out STD_LOGIC_VECTOR (3 downto 0);
           DataBus7 : out STD_LOGIC_VECTOR (3 downto 0)
           );
end component;

component Counter
    Port ( Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           D: in STD_LOGIC_VECTOR(2 downto 0);
           Q : out STD_LOGIC_VECTOR(2 downto 0));
end component;
```

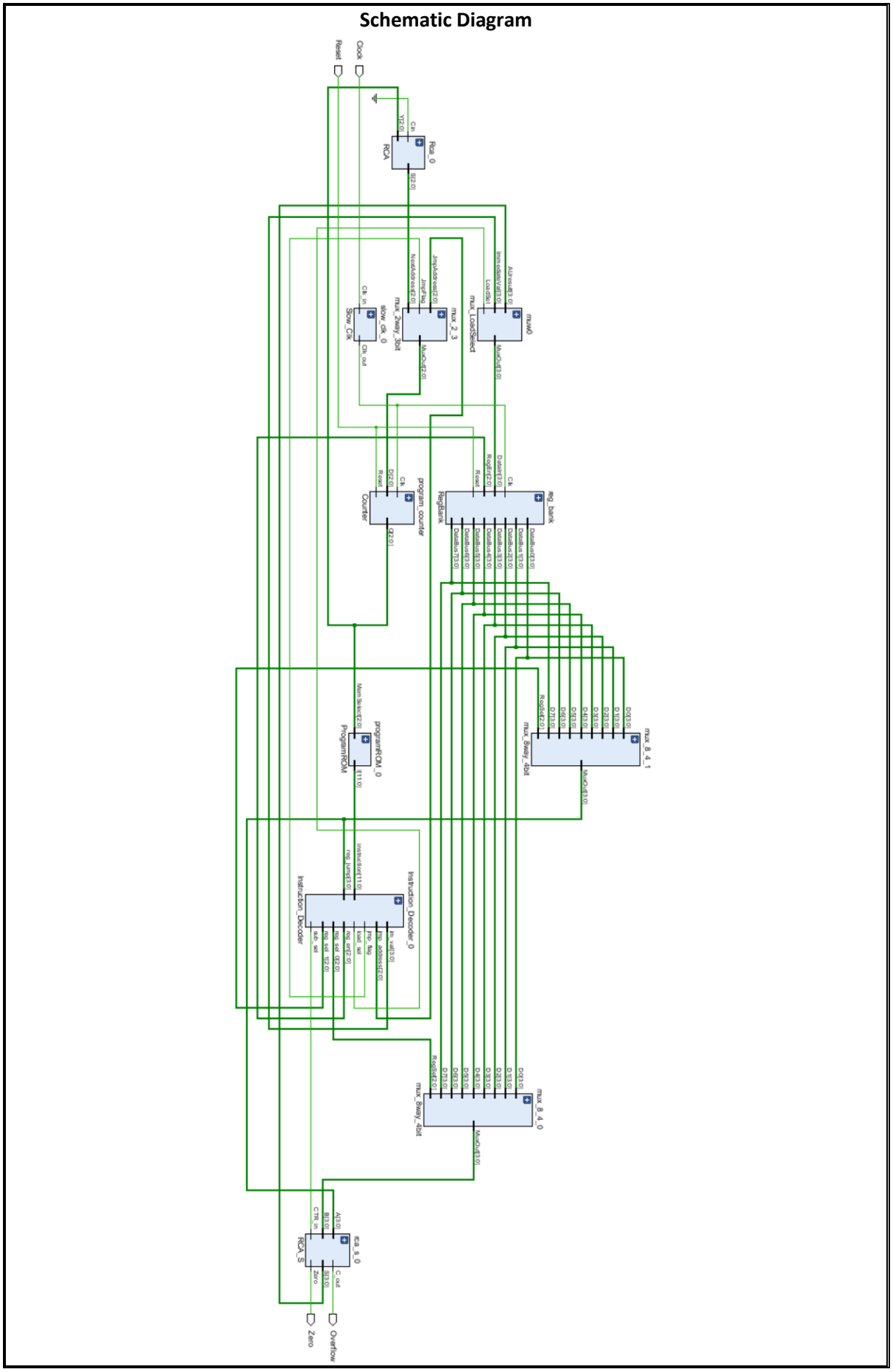
```
component RCA_S
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           CTR_in : in STD_LOGIC;
           C_out : out STD_LOGIC;
           S : out STD_LOGIC_VECTOR (3 downto 0);
           Zero : out STD_LOGIC);
end component;

component RCA
    Port (Y : in STD_LOGIC_VECTOR (2 downto 0);
          S : out STD_LOGIC_VECTOR (2 downto 0);
          Cin : in STD_LOGIC;
          Cout : out STD_LOGIC);
end component;

component mux_2way_3bit
    Port ( JmpAddress : in STD_LOGIC_VECTOR (2 downto 0);
           NextAddress : in STD_LOGIC_VECTOR (2 downto 0);
           JmpFlag : in STD_LOGIC;
           MuxOut : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component mux_8way_4bit
    Port ( RegSel : in STD_LOGIC_VECTOR (2 downto 0);
           MuxOut : out STD_LOGIC_VECTOR (3 downto 0);
           D0 : in STD_LOGIC_VECTOR (3 downto 0);
           D1 : in STD_LOGIC_VECTOR (3 downto 0);
           D2 : in STD_LOGIC_VECTOR (3 downto 0);
           D3 : in STD_LOGIC_VECTOR (3 downto 0);
           D4 : in STD_LOGIC_VECTOR (3 downto 0);
           D5 : in STD_LOGIC_VECTOR (3 downto 0);
           D6 : in STD_LOGIC_VECTOR (3 downto 0);
           D7 : in STD_LOGIC_VECTOR (3 downto 0));
end component;

component mux_LoadSelect
    Port ( LoadSel : in STD_LOGIC;
           AUresult : in STD_LOGIC_VECTOR (3 downto 0);
           ImmediateVal : in STD_LOGIC_VECTOR (3 downto 0);
           MuxOut : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component Slow_Clk
    Port ( Clk_in : in STD_LOGIC;
           Clk_out : out STD_LOGIC);
end component;

--Signals
signal Instruction_Bus: std_logic_vector (11 downto 0);
signal reg_check_jump, imm_val, from_asUnit, to_reg_bank:
std_logic_vector (3 downto 0);
signal addr_to_jump, reg_select0, reg_select1, reg_enable,
memory_select, to_pc, from_rca: std_logic_vector (2 downto 0);
signal jump_flag, add_sub_sel, load_selector, slw_clk,
```

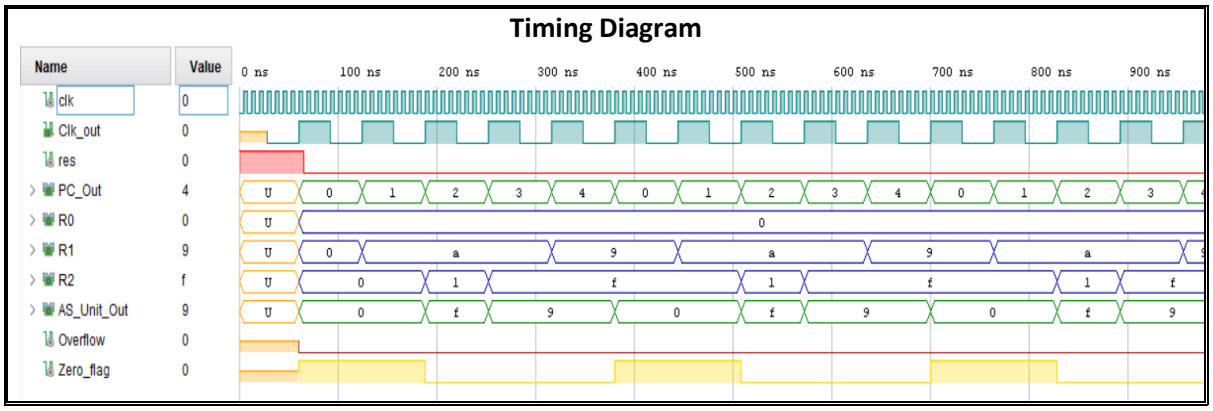
```
RCA_Overflow: std_logic;

signal dbus0, dbus1, dbus2, dbus3, dbus4, dbus5, dbus6, dbus7:
std_logic_vector (3 downto 0);
signal to_asUnit_0, to_asUnit_1: std_logic_vector (3 downto 0);

begin

Instruction_Decoder_0: Instruction_decoder
    PORT MAP(
        instruction => Instruction_Bus,
        jmp_flag => jump_flag,
        jmp_address => addr_to_jump,
        sub_sel => add_sub_sel,
        load_sel => load_selector,
        im_val => imm_val,
        reg_sel_0 => reg_select0,
        reg_sel_1 => reg_select1,
        reg_en => reg_enable,
        reg_jump => reg_check_jump
    );

programROM_0: ProgramROM
    PORT MAP(
        MemSelect => memory_select,
        I => Instruction_Bus
    );

program_counter: Counter
    PORT MAP(
        Clk => slw_clk,
        Reset => Reset,
        D => to_pc,
        Q => memory_select
    );

Rca_0: RCA
    PORT MAP(
        Y => memory_select,
        S => from_rca,
        Cin => '0',
        Cout => RCA_Overflow
    );

mux_2_3: mux_2way_3bit
    PORT MAP(
        JmpAddress => addr_to_jump,
        NextAddress => from_rca,
        JmpFlag => jump_flag,
        MuxOut => to_pc
    );

muw0: mux_LoadSelect
    PORT MAP(
        LoadSel => load_selector,
```

```
        AUresult => from_asUnit,
        ImmediateVal => imm_val,
        MuxOut => to_reg_bank
    );

reg_bank: RegBank
    PORT MAP(
        Clk => slw_clk,
        Reset => Reset,
        RegEn => reg_enable,
        DataIn => to_reg_bank,
        DataBus0 => dbus0,
        DataBus1 => dbus1,
        DataBus2 => dbus2,
        DataBus3 => dbus3,
        DataBus4 => dbus4,
        DataBus5 => dbus5,
        DataBus6 => dbus6,
        DataBus7 => dbus7
    );

mux_8_4_0: mux_8way_4bit
    PORT MAP(
        RegSel => reg_select0,
        MuxOut => to_asUnit_1,
        D0 => dbus0,
        D1 => dbus1,
        D2 => dbus2,
        D3 => dbus3,
        D4 => dbus4,
        D5 => dbus5,
        D6 => dbus6,
        D7 => dbus7
    );

mux_8_4_1: mux_8way_4bit
PORT MAP(
     RegSel => reg_select1,
     MuxOut => to_asUnit_0,
     D0 => dbus0,
     D1 => dbus1,
     D2 => dbus2,
     D3 => dbus3,
     D4 => dbus4,
     D5 => dbus5,
     D6 => dbus6,
     D7 => dbus7
 );

 rca_s_0: RCA_S
    PORT MAP(
        A => to_asUnit_0,
        B => to_asUnit_1,
        CTR_in => add_sub_sel,
        C_out => Overflow,
```

```
        S => from_asUnit,
        Zero => Zero
    );

slow_clk_0: Slow_Clk
    PORT MAP(
        Clk_in => Clock,
        Clk_out => slw_clk
    );

reg_check_jump <= to_asUnit_0;

end Behavioral;
```

Schematic Diagram





Conclusion:

- This nanoprocessor can be improved further by adding new components and improving the current components.
- We can increase the number range which we can execute mathematical operations using this nanoprocessor by increasing the width of the busses and other components.
- Instruction decoder plays a huge role in the nano processor because it is the component which activates the relevant components and busses to execute the instructions.
- We can optimize the nanoproceesor to give the best performance in the following way,
  - Improving the Add/Subtract unit to handle NEG instruction without the help of the R0 register so that we can convert R0 register to a read-write register and use for the calculation.
  - Use busses rather than using many wires
  - Developing the nanoprocessor in a way that each instruction uses a single clock cycle to execute.
  - Create Adder and Subtractor in the same unit
  - Designing the MUX using VHDL conditional statements rather than using four 8-to-1 multiplexers or basic logic gates. It is much easier and less complex.
  - This nanoprocessor can execute mathematical operations in the range of –8 to +7. Therefore, the nanoprocessor should be optimized in a way that any value larger than +7 or less than –8 considered as an overflow.
  - The nanoprocessor, zero flags will be raised only when an ADD instruction is executed.

Resource Consumption (LUT/FF counts):

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	45	0	20800	0.22
LUT as Logic	45	0	20800	0.22
LUT as Memory	0	0	9600	0.00
Slice Registers	53	0	41600	0.13
Register as Flip Flop	53	0	41600	0.13
Register as Latch	0	0	41600	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	53	Flop & Latch
LUT4	22	LUT
LUT5	18	LUT
OBUF	17	IO
LUT6	11	LUT
LUT3	10	LUT
CARRY4	8	CarryLogic
LUT2	2	LUT
IBUF	2	IO
LUT1	1	LUT
BUFG	1	Clock

Contribution:

Everyone	Nanoprocessor Assemble
Ranathunga N. D. (200517U)	Instruction Decoder NEG Instruction optimization Debugging
Keerthichandra H. M. P. M. (200304N)	Program Counter Program ROM
Marasinghe M. M. R. S. (200382A)	K-way b-bit MUX MUX optimization
Vihidun D. P. T. (200682T)	ADD/SUB Unit ADD/SUB Optimization
Marium M. S. S. (200384G)	3-bit adder Simulations for all components