Lab 9-10 – Nanoprocessor Design Competition

CS1050 Computer Organization and Digital Design
Dept. of Computer Science and Engineering
University of Moratuwa

Group members:

- Name :- Gunasena H.A.S.I
  Reg. No. :-210195L
- Name :- Madusanka G.I.D.L
  Reg. No. :- 210353V

Lab Task:

1. Our objective is to run a simple programme and get its result by using a 4-bit processor capable of executing 8 instructions.
2. To do this, we developed a 4-bit arithmetic unit capable of performing both addition and subtraction operations for signed integers.
3. To execute the desired operations, we need to decode instructions that will activate the specific components within the processor.
4. We will design and develop multiplexers or tri-state buses with k-way and b-bit configurations to facilitate data routing and selection.
5. After completing the design and development phase, it is crucial to verify the functionality of these components by conducting simulations and testing them on the development board.
6. Enhance team-working skills such as communication, coordination, sharing responsibilities, and integrating components developed by different team members.

Assembly program and its machine code representation:

1. **Start from 10 and decrease it to 0 by decrement one by 1**

```
MOVI        R7, 10   ;        R7 ⟵ 10
Machine Code: 10 111 000 1010

MOVI        R2, 1    ;        R2 ⟵ 1
Machine Code: 10 010 000 0001

NEG   R2             ;        R2 ⟵ -R2
Machine Code: 01 010 000 0000

ADD   R7, R2         ;        R7 ⟵ R7 + R2
Machine Code: 00 111 010 0000

JZR     R7, 7        ; If R7 = 0 jump to line 4 again ant stop executing programme
Machine Code: 11 111 000 0100

JZR     R0, 3        ; If R0 = 0 jump to line 3
Machine Code: 11 000 000 0011
```

2. **the total of all integers between 1 and 3**

MOVI R7,0 ; R7 ← 0
Machine Code: 10 111 000 0000

MOVI R1, 1 ; R1 ← 1
Machine Code: 10 001 000 0001

MOVI R2, 2 ; R2 ← 2
Machine Code: 10 010 000 0010

MOVI R3, 3 ; R3 ← 3
Machine Code: 10 110 000 0011

ADD R7, R1 ; R7 ← R7 + R1
Machine Code: 00 111 001 0000

ADD R7, R2 ; R7 ← R7 + R2
Machine Code: 00 111 010 0000

ADD R7, R3 ; R7 ← R7 + R3
Machine Code: 00 111 011 0000

JZR R0, 0 ; If R0 = 0 jump to line 6
Machine Code: 11 000 000 0111

VHDL codes:

SOURCE CODES:

HA

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

entity HA is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C : out STD_LOGIC;
           S : out STD_LOGIC);
end HA;

architecture Behavioral of HA is
signal NOT_A_out_sig : std_logic;
signal NOT_B_out_sig : std_logic;
begin
    NOT_A_out_sig <= NOT(A);
    NOT_B_out_sig <= NOT(B);
    C <= A AND B;
    S <= ( NOT_A_out_sig AND B ) OR ( A AND NOT_B_out_sig);
end Behavioral;
```

FA

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

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

architecture Behavioral of FA is
component HA
    port (
    A: in std_logic;
    B: in std_logic;
    S: out std_logic;
    C: out std_logic);
end component;

SIGNAL HA0_C, HA0_S, HA1_S, HA1_C : std_logic;
signal NOT_A_out_sig : std_logic;
signal NOT_B_out_sig : std_logic;
signal NOT_Cin_out_sig : std_logic;

begin
HA_0 : HA
    port map (
    A => A,
    B => B,
    S => HA0_S,
    C => HA0_C);

HA_1 : HA
        port map (
        A => A,
        B => B,
        S => HA1_S,
        C => HA1_C);

NOT_A_out_sig <= NOT(A);
NOT_B_out_sig <= NOT(B);
NOT_Cin_out_sig <= NOT(C_in);
S <= (NOT_A_out_sig AND NOT_B_out_sig AND C_in) OR
     (NOT_A_out_sig AND B AND NOT_Cin_out_sig ) OR
```

```
     (A AND NOT_B_out_sig AND NOT_Cin_out_sig ) OR
     (A AND B AND C_in);
C_out <= (NOT_A_out_sig AND B AND C_in) OR
          (A AND NOT_B_out_sig AND C_in ) OR
          (A AND B AND NOT_Cin_out_sig ) OR
          (A AND B AND C_in);

end Behavioral;
```

RCA_4

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

entity RCA_4 is
    Port ( A0 : in STD_LOGIC;
           A1 : in STD_LOGIC;
           A2 : in STD_LOGIC;
           A3 : in STD_LOGIC;
           B0 : in STD_LOGIC;
           B1 : in STD_LOGIC;
           B2 : in STD_LOGIC;
           B3 : in STD_LOGIC;
           C_in : in STD_LOGIC;
           S1 : out STD_LOGIC;
           S2 : out STD_LOGIC;
           S3 : out STD_LOGIC;
           C_out : out STD_LOGIC;
           S0 : out STD_LOGIC);
end RCA_4;

architecture Behavioral of RCA_4 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 FA0_S, FA0_C, FA1_S, FA1_C, FA2_S, FA2_C, FA3_S, FA3_C
      : std_logic;
```

```
begin
        FA_0 : FA
            port map (
            A => A0,
            B => B0,
            C_in => C_in,
            S => S0,
            C_Out => FA0_C);

        FA_1 : FA
             port map (
             A => A1,
             B => B1,
             C_in => FA0_C,
             S => S1,
             C_Out => FA1_C);

        FA_2 : FA
              port map (
              A => A2,
              B => B2,
              C_in => FA1_C,
              S => S2,
              C_Out => FA2_C);

        FA_3 : FA
              port map (
              A => A3,
              B => B3,
              C_in => FA2_C,
              S => S3,
              C_Out => C_out);

end Behavioral;
```

4-bit Add/Subtract unit

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

entity Add_Sub_4bit is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           S : inout STD_LOGIC_VECTOR (3 downto 0);
           AddSubSelect : in STD_LOGIC;
           Zero : out STD_LOGIC;
           Overflow : out STD_LOGIC);
end Add_Sub_4bit;

architecture Behavioral of Add_Sub_4bit is

    component RCA_4 is
        Port ( A0 : in STD_LOGIC;
               A1 : in STD_LOGIC;
               A2 : in STD_LOGIC;
               A3 : in STD_LOGIC;
               B0 : in STD_LOGIC;
               B1 : in STD_LOGIC;
               B2 : in STD_LOGIC;
               B3 : in STD_LOGIC;
               C_in : in STD_LOGIC;
               S1 : out STD_LOGIC;
               S2 : out STD_LOGIC;
               S3 : out STD_LOGIC;
               C_out : out STD_LOGIC;
               S0 : out STD_LOGIC);
    end component;

    signal Q0,Q1,Q2,Q3 :STD_LOGIC;

begin
    Q0 <= B(0) XOR AddSubSelect;
    Q1 <= B(1) XOR AddSubSelect;
    Q2 <= B(2) XOR AddSubSelect;
    Q3 <= B(3) XOR AddSubSelect;

    Add_Sub : RCA_4
        port map(
            A0 => A(0),
            A1 => A(1),
            A2 => A(2),
            A3 => A(3),
```

```
            B0 => Q0,
            B1 => Q1,
            B2 => Q2,
            B3 => Q3,
            S0 => S(0),
            S1 => S(1),
            S2 => S(2),
            S3 => S(3),
            C_in => AddSubSelect,
            C_out => Overflow);
    Zero <= NOT(S(0) OR S(1) OR S(2) OR S(3));
end Behavioral;
```

D_FF

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

entity D_FF is
    Port ( D : in STD_LOGIC;
           Res : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC;
           Qbar : out STD_LOGIC);
end D_FF;

architecture Behavioral of D_FF is
begin
    process (Clk) begin
        if (rising_edge(Clk)) then
            if Res = '1' then
                Q <= '0';
                Qbar <= '1';
            else
                Q <= D;
                Qbar <= not D;
            end if;
        end if;
    end process;
end Behavioral;
```

RCA_3

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

entity RCA_3 is
    Port ( --A : in STD_LOGIC_VECTOR (2 downto 0);      -- Always this is
equals to 001
           --C_in : in STD_LOGIC;                       -- Since we have to
add, this shold be GROUND. i.e C_in = '0'
           B : in STD_LOGIC_VECTOR (2 downto 0);      -- 3-bit memory address
input
           S : out STD_LOGIC_VECTOR (2 downto 0);     -- Next memory address
will out
           C_out : out STD_LOGIC);                    --  This will not we
care becuase our programme only have 8 instructions.

end RCA_3;

architecture Behavioral of RCA_3 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 C :std_logic_vector(1 downto 0);
begin

FA_0 : FA port map(
    A => '1', --A(0),
    B => B(0),
    C_in =>'0',
    S => S(0),
    C_out => C(0)
);

FA_1 : FA port map(
    A => '0', --A(1),
    B => B(1),
    C_in => C(0),
    S => S(1),
    C_out => C(1)
```

```
);

FA_2 : FA port map(
    A => '0', --A(2),
    B => B(2),
    C_in => C(1),
    S => S(2),
    C_out => C_out
);

end Behavioral;
```

Mux_2_to_1

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

entity Mux_2_to_1 is
    Port ( D : in STD_LOGIC_VECTOR (1 downto 0);
           S : in STD_LOGIC;
           Y : out STD_LOGIC);
end Mux_2_to_1;

architecture Behavioral of Mux_2_to_1 is
Signal NOT_S : std_logic;

begin
    NOT_S <= NOT(S);

    Y <= (D(1) AND NOT_S) OR (D(0) AND S);
end Behavioral;
```

Mux_2_way_3_bit

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

entity Mux_2way_3bit is
    Port ( jmp_adrs : in STD_LOGIC_VECTOR (2 downto 0);
           adder_3bit : in STD_LOGIC_VECTOR (2 downto 0);
           jmp_flag : in STD_LOGIC;
           output : out STD_LOGIC_VECTOR (2 downto 0));
end Mux_2way_3bit;

architecture Behavioral of Mux_2way_3bit is
    COMPONENT Mux_2_to_1
    PORT ( D : in STD_LOGIC_VECTOR (1 downto 0);
           S : in STD_LOGIC;
           Y : out STD_LOGIC);
    END COMPONENT;

begin
    Mux_0: Mux_2_to_1 port map(
        D(0) => jmp_adrs(0),
        D(1) => adder_3bit(0),
        S => jmp_flag,
        Y => output(0)
    );

    Mux_1: Mux_2_to_1 port map(
        D(0) => jmp_adrs(1),
        D(1) => adder_3bit(1),
        S => jmp_flag,
        Y => output(1)
    );

    Mux_2: Mux_2_to_1 port map(
        D(0) => jmp_adrs(2),
        D(1) => adder_3bit(2),
        S => jmp_flag,
        Y => output(2)
    );


end Behavioral;
```

3-bit Program Counter (PC)

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

entity PC_3bit is
    Port ( Reset : in STD_LOGIC;
           Clk : in STD_LOGIC;
           MemorySelect : inout STD_LOGIC_VECTOR (2 downto 0);
           JumpAddr : in STD_LOGIC_VECTOR (2 downto 0);                -- If
there is a jump instruction, PC set MemorrySelect as the
           JumpFlag : in STD_LOGIC
           );
end PC_3bit;

architecture Behavioral of PC_3bit is
    COMPONENT D_FF
    PORT (
        D : in STD_LOGIC;
        Res : in STD_LOGIC;
        Clk : in STD_LOGIC;
        Q : out STD_LOGIC;
        Qbar : out STD_LOGIC );
    END COMPONENT;

    --COMPONENT Slow_Clk
    --PORT (
        --Clk_in : in STD_LOGIC;
        --Clk_out : out STD_LOGIC );
    --END COMPONENT;

    COMPONENT RCA_3
    PORT (
        B : in STD_LOGIC_VECTOR (2 downto 0);      -- Always this is equals to
001
        S : out STD_LOGIC_VECTOR (2 downto 0);     -- Next memory address will
out
        C_out : out STD_LOGIC);
    END COMPONENT;

    COMPONENT Mux_2way_3bit
    PORT ( jmp_adrs : in STD_LOGIC_VECTOR (2 downto 0);
           adder_3bit : in STD_LOGIC_VECTOR (2 downto 0);
           jmp_flag : in STD_LOGIC;
           output : out STD_LOGIC_VECTOR (2 downto 0));
    END COMPONENT;
```

```
    signal D, A: std_logic_vector (2 downto 0);
    --signal Clk_slow : std_logic;

    begin
        --Slow_Clk0 : Slow_Clk
            --port map (
                --Clk_in => Clk,
                --Clk_out => Clk_slow );

        D_FF0 : D_FF
            port map (
                D => D(0),
                Q => MemorySelect(0),
                Res => Reset,
                Clk => Clk);--Clk_slow );

        D_FF1 : D_FF
            port map (
                D => D(1),
                Q => MemorySelect(1),
                Res => Reset,
                Clk => Clk);--Clk_slow );

        D_FF2 : D_FF
            port map (
                 D => D(2),
                 Q => MemorySelect(2),
                 Res => Reset,
                 Clk => Clk);--Clk_slow );

        RCA3 : RCA_3
            port map (
                 B => MemorySelect,
                 S => A );

        Mux_2way_3bit0 : Mux_2way_3bit
            port map (
                 jmp_adrs => JumpAddr,
                 adder_3bit => A,
                 jmp_flag => JumpFlag,
                 output => D  );

    end Behavioral;
```

Mux_8_way_4_bit

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

entity Mux_8way_4bit is
    Port (  reg0 : in STD_LOGIC_VECTOR (3 downto 0);
            reg1 : in STD_LOGIC_VECTOR (3 downto 0);
            reg2 : in STD_LOGIC_VECTOR (3 downto 0);
            reg3 : in STD_LOGIC_VECTOR (3 downto 0);
            reg4 : in STD_LOGIC_VECTOR (3 downto 0);
            reg5 : in STD_LOGIC_VECTOR (3 downto 0);
            reg6 : in STD_LOGIC_VECTOR (3 downto 0);
            reg7 : in STD_LOGIC_VECTOR (3 downto 0);
            Reg_select : in STD_LOGIC_VECTOR (2 downto 0);
            output : out STD_LOGIC_VECTOR (3 downto 0));
end Mux_8way_4bit;

architecture Behavioral of Mux_8way_4bit is

begin
    process(reg0, reg1, reg2, reg3, reg4, reg5, reg6, reg7, Reg_select)
    begin
        case (Reg_select) is
            when "000" =>
                output <= reg0;
            when "001" =>
               output <= reg1;
            when "010" =>
               output <= reg2;
            when "011" =>
               output <= reg3;
            when "100" =>
               output <= reg4;
            when "101" =>
               output <= reg5;
            when "110" =>
               output <= reg6;
            when "111" =>
               output <= reg7;
            when others =>
               output <= "0000";  -- For handle any undefined cases
        end case;
    end process;
end Behavioral;
```

Mux_2_way_4_bit

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

 Mux_2way_4bit is
    Port ( AddSub : in STD_LOGIC_VECTOR (3 downto 0);
           Immediate_Val : in STD_LOGIC_VECTOR (3 downto 0);
           Load_Select : in STD_LOGIC;
           output : out STD_LOGIC_VECTOR (3 downto 0));
end Mux_2way_4bit;

architecture Behavioral of Mux_2way_4bit is

COMPONENT Mux_2_to_1
PORT ( D : in STD_LOGIC_VECTOR (1 downto 0);
       S : in STD_LOGIC;
       Y : out STD_LOGIC);
END COMPONENT;

begin
Mux_0: Mux_2_to_1 port map(
    D(1) => AddSub(0),
    D(0) => Immediate_Val(0),
    S => Load_Select,
    Y => output(0) );

Mux_1: Mux_2_to_1 port map(
    D(1) => AddSub(1),
    D(0) => Immediate_Val(1),
    S => Load_Select,
    Y => output(1) );

Mux_2: Mux_2_to_1 port map(
    D(1) => AddSub(2),
    D(0) => Immediate_Val(2),
    S => Load_Select,
    Y => output(2) );

Mux_3: Mux_2_to_1 port map(
    D(1) => AddSub(3),
    D(0) => Immediate_Val(3),
    S => Load_Select,
    Y => output(3) );

end Behavioral;
```

Slow_Clock

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

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

architecture Behavioral of Slow_Clk is

signal count : integer := 1;
signal clk_status : std_logic := '0';

begin
    process (Clk_in) begin
        if (rising_edge(Clk_in)) then
            count <= count + 1;
            if(count = 10) then --50000000 for project (10 for test)
                clk_status <= not clk_status;
                Clk_out <= clk_status;
                count <= 1;
            end if;
        end if;
    end process;

end Behavioral;
```

LUT_16_7

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

entity LUT_16_7 is
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (6 downto 0));
end LUT_16_7;

architecture Behavioral of LUT_16_7 is

type rom_type is array (0 to 15) of std_logic_vector(6 downto 0);
    signal sevenSegment_ROM : rom_type := (
        "1000000", -- 0     0= on 1=off    g f e d c b a
        "1111001", -- 1
        "0100100", -- 2
        "0110000", -- 3
        "0011001", -- 4
        "0010010", -- 5
        "0000010", -- 6
        "1111000", -- 7
        "0000000", -- 8
        "0010000", -- 9
        "0001000", -- a
        "0000011", -- b
        "1000110", -- c
        "0100001", -- d
        "0000110", -- e
        "0001110"  -- f
 );

begin
    data <= sevenSegment_ROM(to_integer(unsigned(address)));

end Behavioral;
```

Instruction decoder

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

entity Instruction_Dec is
    Port ( Reg_En : out STD_LOGIC_VECTOR (2 downto 0);
           Load_Select : out STD_LOGIC;
           Immediate_Val : out STD_LOGIC_VECTOR (3 downto 0);
           MuxA_Select : out STD_LOGIC_VECTOR (2 downto 0);
           MuxB_Select : out STD_LOGIC_VECTOR (2 downto 0);
           AddSub_Select : out STD_LOGIC;
           JmpFlag : out STD_LOGIC;
           JmpAddr : out STD_LOGIC_VECTOR (2 downto 0);
           Instruct_Bus : in STD_LOGIC_VECTOR (11 downto 0);
           JumpCheck : in STD_LOGIC_VECTOR (3 downto 0));
end Instruction_Dec;

architecture Behavioral of Instruction_Dec is

begin
   process(Instruct_Bus, JumpCheck)
   begin

       -- for ADD intruction
       if Instruct_Bus (11 downto 10) = "00" then
           JmpFlag <= '0';

           MuxA_Select <= Instruct_Bus (9 downto 7);
           MuxB_Select <= Instruct_Bus (6 downto 4);
           AddSub_Select <= '0'; -- Here we perform only addition.
           Load_Select <= '0';
           Reg_En <= Instruct_Bus (9 downto 7);  -- Store the result on
Register A

       -- for NEG intruction
       elsif Instruct_Bus (11 downto 10) = "01" then
           JmpFlag <= '0';

           MuxA_Select <= "000";  -- For get -B we have to set A to zero(i.e.
set to Register 0) because adder perform only A+B.
           MuxB_Select <= Instruct_Bus (9 downto 7);  -- For get a negative
value of a particular number, it can be only done from Mux B
           AddSub_Select <= '1';  -- Here we perform only subtraction.
           Load_Select <= '0';
```

```
            Reg_En <= Instruct_Bus (9 downto 7);  -- Store the result on
Register A

        -- for MOV intruction
        elsif Instruct_Bus (11 downto 10) = "10" then
            JmpFlag <= '0';
            MuxA_Select <= "000";
            MuxB_Select <= "000";

            Immediate_Val <= Instruct_Bus (3 downto 0);
            Reg_En <= Instruct_Bus (9 downto 7);
            Load_Select <= '1';  -- Select IMMEDIATE VALUE port via

        -- for JMP intruction
        else
            MuxB_Select <= "000";
            Immediate_Val <= "0000";
            Load_Select <= '1';
            Reg_En <= "000";

            MuxA_Select <= Instruct_Bus (9 downto 7);
            if JumpCheck = "0000" then
                JmpAddr <= Instruct_Bus (2 downto 0);
                JmpFlag <= '1';
            else
                JmpFlag <= '0';
            end if;
        end if;
    end process;

end Behavioral;
```

Program Rom

1. Start from 10 and decrease it to 0 by decrement one by 1

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


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

architecture Behavioral of Program_ROM is

    type rom_type is array (0 to 7) of std_logic_vector(11 downto 0);
        signal Assembly_Code : rom_type := (
                    "101110000000", -- movi R7,0
                    "100010000001", -- movi R1,1
                    "100100000010", -- movi R2,2
                    "101100000110", -- movi R3,3
                    "001110010000", -- add  R7,R1
                    "001110100000", -- add  R7,R2
                    "110000000110", -- add  R7,R3
                );
begin
    InstructBus <= Assembly_Code(to_integer(unsigned(MemSelect)));
end Behavioral;
```

2. The total of all integers between 1 and 3

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

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

architecture Behavioral of Program_ROM is

    type rom_type is array (0 to 7) of std_logic_vector(11 downto 0);
        signal Assembly_Code : rom_type := (
                    "101110000000", -- MOVI R2, 1  ; R2 <--- 1
                    "100010000001", -- MOVI R2, 1  ; R2 <--- 1
                    "100100000010", -- NEG  R2     ; R2 <--- -R2
                    "100110000011", -- ADD  R7, R2 ; R7 <--- R7 + R2
                    "001110010000", -- JZR  R7, 7  ; If R7 = 0 jump to line 7
                    "001110100000", -- JZR R0, 3   ; If R0 = 0 jump to line 3
                    "001110110000", -- 6
                    "110000000111"  -- 7
         );
begin
    InstructBus <= Assembly_Code(to_integer(unsigned(MemSelect)));
end Behavioral;
```

Decoder_2_to_4

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

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

architecture Behavioral of Decoder_2_to_4 is
Signal NOT_I0_sig : std_logic;
Signal NOT_I1_sig : std_logic;


begin
    NOT_I0_sig <= NOT(I(0));
    NOT_I1_sig <= NOT(I(1));

    Y(0) <= NOT_I0_sig AND NOT_I1_sig AND EN;
    Y(1) <= I(0) AND NOT_I1_sig AND EN;
    Y(2) <= NOT_I0_sig AND I(1) AND EN;
    Y(3) <= I(0) AND I(1) AND EN;

end Behavioral;
```

Decoder_3_to_8

```
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;
    EN: in STD_LOGIC;
    Y: out STD_LOGIC_VECTOR );
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
 Decode_2_to_4_0 : Decoder_2_to_4
 port map(
    I => I0,
    EN => en0,
    Y => Y0 );

 Decode_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;
```

Register

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

entity Reg is
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
           En : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Reset : in STD_LOGIC;   -- add reset pin for reset output
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end Reg;

architecture Behavioral of Reg is

begin
    process(Clk) begin
    if (Reset = '1') then
        Q <= (others => '0');
    elsif (rising_edge(Clk)) then
        if En = '1' then
            Q <= D;
        end if;
    end if;
    end process;

end Behavioral;
```

Register Bank

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

entity Register_Bank is
    Port ( D_input : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_En : in STD_LOGIC_VECTOR (2 downto 0);
           Clock : in STD_LOGIC;
           Reset : in STD_LOGIC;
           D_out0 : out STD_LOGIC_VECTOR (3 downto 0):= “0000”;
           D_out1 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out2 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out3 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out4 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out5 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out6 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out7 : out STD_LOGIC_VECTOR (3 downto 0));
end Register_Bank;

architecture Behavioral of Register_Bank is

    component 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 component;
   --####################THIS REG D FLIP FLOP MUST HAVE a reset PIN. ADD THIS
FOR COMPLETION
    component Reg is
        Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
               En : in STD_LOGIC;
               Clk : in STD_LOGIC;
               Reset : in STD_LOGIC;
               Q : out STD_LOGIC_VECTOR (3 downto 0));
    end component;

signal y : STD_LOGIC_VECTOR (7 downto 0);

begin
    decoder : Decoder_3_to_8
    port map (
        I => Reg_En,
        En => '1',
        Y => y );

    Reg0 : Reg
```

```
    port map (
        D => D_input,
        En => '0',
        Clk => Clock,
        Reset => Reset,
        Q => D_out0 );

    Reg1 : Reg
    port map (
        D => D_input,
        En => y(1),
        Clk => Clock,
        Reset => Reset,
        Q => D_out1 );

     Reg2 : Reg
     port map (
        D => D_input,
        En => y(2),
        Clk => Clock,
        Reset => Reset,
        Q => D_out2 );

     Reg3 : Reg
     port map (
        D => D_input,
        En => y(3),
        Clk => Clock,
        Reset => Reset,
        Q => D_out3 );

     Reg4 : Reg
     port map (
        D => D_input,
        En => y(4),
        Clk => Clock,
        Reset => Reset,
        Q => D_out4 );

     Reg5 : Reg
     port map (
        D => D_input,
        En => y(5),
        Clk => Clock,
        Reset => Reset,
        Q => D_out5 );

     Reg6 : Reg
```

```
    port map (
       D => D_input,
       En => y(6),
       Clk => Clock,
       Reset => Reset,
       Q => D_out6 );

    Reg7 : Reg
    port map (
       D => D_input,
       En => y(7),
       Clk => Clock,
       Reset => Reset,
       Q => D_out7 );

end Behavioral
```

Nano Processor

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

entity Nanoprocessor is
    Port ( Reset : in STD_LOGIC;
           clk : in STD_LOGIC;
           Zero_LED : out STD_LOGIC;
           Overflow_LED : out STD_LOGIC;
           LED0 : out STD_LOGIC;
           LED1 : out STD_LOGIC;
           LED2 : out STD_LOGIC;
           LED3 : out STD_LOGIC;
           Anode : out STD_LOGIC_VECTOR (3 downto 0);
           S_7Seg : out STD_LOGIC_VECTOR (6 downto 0)
    );
end Nanoprocessor;

architecture Behavioral of Nanoprocessor is
component Register_Bank is
    Port ( D_input : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_En : in STD_LOGIC_VECTOR (2 downto 0);
           Clock : in STD_LOGIC;
           Reset : in STD_LOGIC;
           D_out0 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out1 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out2 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out3 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out4 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out5 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out6 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out7 : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component Mux_8way_4bit is
    Port (  reg0 : in STD_LOGIC_VECTOR (3 downto 0);
            reg1 : in STD_LOGIC_VECTOR (3 downto 0);
            reg2 : in STD_LOGIC_VECTOR (3 downto 0);
            reg3 : in STD_LOGIC_VECTOR (3 downto 0);
            reg4 : in STD_LOGIC_VECTOR (3 downto 0);
            reg5 : in STD_LOGIC_VECTOR (3 downto 0);
            reg6 : in STD_LOGIC_VECTOR (3 downto 0);
            reg7 : in STD_LOGIC_VECTOR (3 downto 0);
```

```
            Reg_select : in STD_LOGIC_VECTOR (2 downto 0);
            output : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component Add_Sub_4bit is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           S : inout STD_LOGIC_VECTOR (3 downto 0);
           AddSubSelect : in STD_LOGIC;
           Zero : out STD_LOGIC;
           Overflow : out STD_LOGIC);
end component;

component Instruction_Dec is
    Port ( Reg_En : out STD_LOGIC_VECTOR (2 downto 0);
           Load_Select : out STD_LOGIC;
           Immediate_Val : out STD_LOGIC_VECTOR (3 downto 0);
           MuxA_Select : out STD_LOGIC_VECTOR (2 downto 0);
           MuxB_Select : out STD_LOGIC_VECTOR (2 downto 0);
           AddSub_Select : out STD_LOGIC;
           JmpFlag : out STD_LOGIC;
           JmpAddr : out STD_LOGIC_VECTOR (2 downto 0);
           Instruct_Bus : in STD_LOGIC_VECTOR (11 downto 0);
           JumpCheck : in STD_LOGIC_VECTOR (3 downto 0));
end component;

component Program_ROM is
    Port ( MemSelect : in STD_LOGIC_VECTOR (2 downto 0);
           InstructBus : out STD_LOGIC_VECTOR (11 downto 0));
end component;

component PC_3bit is
    Port ( Reset : in STD_LOGIC;
           Clk : in STD_LOGIC;
           MemorySelect : inout STD_LOGIC_VECTOR (2 downto 0);
           JumpAddr : in STD_LOGIC_VECTOR (2 downto 0);                -- If
there is a jump instruction, PC set MemorrySelect as the
           JumpFlag : in STD_LOGIC);
end component;

component Mux_2way_4bit is
    Port ( AddSub : in STD_LOGIC_VECTOR (3 downto 0);
           Immediate_Val : in STD_LOGIC_VECTOR (3 downto 0);
           Load_Select : in STD_LOGIC;
           output : 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;


component LUT_16_7 is
    Port ( address : in STD_LOGIC_VECTOR (3 downto 0);
           data : out STD_LOGIC_VECTOR (6 downto 0));
end component;


signal Clk_slow : std_logic;
signal Memory_Select : std_logic_vector (2 downto 0);  -- from program counter
to program ROM
signal Jump_Address : std_logic_vector (2 downto 0);  -- from instruction
decoder to program counter
signal Jump_Flag : std_logic; -- from instruction decoder to program counter
signal I : std_logic_vector (11 downto 0);  -- from program ROM to instruction
decoder
signal Register_Enable : std_logic_vector (2 downto 0);  -- from instruction
decoder to Register Bank
signal Load_Select : std_logic; -- from instruction decoder to Mux_2way_4bit
signal Immediate_Value : std_logic_vector (3 downto 0);  -- from instruction
decoder to Mux_2way_4bit
signal Mux_A_Select : std_logic_vector (2 downto 0);  -- from instruction
decoder to Mux_8way_4bit(A)
signal Mux_B_Select : std_logic_vector (2 downto 0);  -- from instruction
decoder to Mux_8way_4bit(B)
signal ADD_SUB_Select : std_logic; -- from instruction decoder to Add_Sub_4bit
signal Mux_A : std_logic_vector (3 downto 0);  -- from Mux_8way_4bit(A) to
Add_Sub_4bit / from instruction decoder to Mux_8way_4bit(A) output
signal Mux_B : std_logic_vector (3 downto 0);  -- from Mux_8way_4bit(B) to
Add_Sub_4bit
signal Add_Sub_Result : std_logic_vector (3 downto 0);  -- from Add_Sub_4bit
to Mux_2way_4bit
signal Register_Input : std_logic_vector (3 downto 0);  -- from Mux_2way_4bit
to Register Bank
signal R0 : std_logic_vector (3 downto 0);  -- from R0 to Mux_8way_4bit
signal R1 : std_logic_vector (3 downto 0);  -- from R1 to Mux_8way_4bit
signal R2 : std_logic_vector (3 downto 0);  -- from R2 to Mux_8way_4bit
signal R3 : std_logic_vector (3 downto 0);  -- from R3 to Mux_8way_4bit
signal R4 : std_logic_vector (3 downto 0);  -- from R4 to Mux_8way_4bit
signal R5 : std_logic_vector (3 downto 0);  -- from R5 to Mux_8way_4bit
signal R6 : std_logic_vector (3 downto 0);  -- from R6 to Mux_8way_4bit
signal R7 : std_logic_vector (3 downto 0);  -- from R7 to Mux_8way_4bit


signal S_7s : STD_LOGIC_VECTOR (6 downto 0) := "0000000";
```

```
signal clk_7seg : STD_LOGIC;
signal selected_7seg : integer := 0;




begin

Slow_Clock : Slow_Clk
port map (
    Clk_in => clk,
    Clk_out => Clk_slow
);

LUT_16_7_0 : LUT_16_7
port map (
    address => R7,
    data => S_7s
 );

Programming_Counter : PC_3bit
port map (
    Reset => Reset,
    Clk => Clk_slow,
    MemorySelect => Memory_Select,
    JumpAddr => Jump_Address,
    JumpFlag => Jump_Flag
);

Program_ROM0 : Program_ROM
port map (
    MemSelect => Memory_Select,
    InstructBus => I
);

Instruction_Decoder : Instruction_Dec
port map (
    Reg_En => Register_Enable,
    Load_Select => Load_Select,
    Immediate_Val => Immediate_Value,
    MuxA_Select => Mux_A_Select,
    MuxB_Select => Mux_B_Select,
    AddSub_Select => ADD_SUB_Select,
    JmpFlag => Jump_Flag,
    JmpAddr => Jump_Address,
    Instruct_Bus => I,
    JumpCheck => Mux_A
);
```

```
Mux_2way_4bit0 : Mux_2way_4bit
port map (
    AddSub =>  Add_Sub_Result,
    Immediate_Val => Immediate_Value,
    Load_Select => Load_Select,
    output => Register_Input
);

ADD_SUB_Unit : Add_Sub_4bit
port map (
    A => Mux_A,
    B => Mux_B,
    S => Add_Sub_Result,
    AddSubSelect => ADD_SUB_Select,
    Zero => Zero_LED,
    Overflow => Overflow_LED
);

MuxA : Mux_8way_4bit
port map (
    reg0 => R0,
    reg1 => R1,
    reg2 => R2,
    reg3 => R3,
    reg4 => R4,
    reg5 => R5,
    reg6 => R6,
    reg7 => R7,
    Reg_select => Mux_A_Select,
    output => Mux_A
);

MuxB : Mux_8way_4bit
port map (
    reg0 => R0,
    reg1 => R1,
    reg2 => R2,
    reg3 => R3,
    reg4 => R4,
    reg5 => R5,
    reg6 => R6,
    reg7 => R7,
    Reg_select => Mux_B_Select,
    output => Mux_B
);

RegisterBank : Register_Bank
```

```
port map (
    D_input => Register_Input,
    Reg_En => Register_Enable,
    Clock => Clk_slow,
    Reset => Reset,
    D_out0 => R0,
    D_out1 => R1,
    D_out2 => R2,
    D_out3 => R3,
    D_out4 => R4,
    D_out5 => R5,
    D_out6 => R6,
    D_out7 => R7
);

LED0 <= R7(0);
LED1 <= R7(1);
LED2 <= R7(2);
LED3 <= R7(3);

process(clk_7seg)
begin
    --if(selected_7seg = 0) then
        Anode <= "1110";
        S_7Seg <= S_7s;
    --elsif ( selected_7seg = 1 ) then
        --Anode <= "1101";
        --if(c = '0') then
            --S_7seg <= "0000001";
        --else
            --S_7seg <= "1001111";
        --end if;
    --end if;
end process;

end Behavioral;
```

TEST BENCH FILES:

4-bit Add/Subtract unit

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

entity Add_Sub_Sim is
--  Port ( );
end Add_Sub_Sim;

architecture Behavioral of Add_Sub_Sim is

    -- Component declaration
    component Add_Sub_4bit is
         Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
                B : in STD_LOGIC_VECTOR (3 downto 0);
                S : inout STD_LOGIC_VECTOR (3 downto 0);
                AddSubSelect : in STD_LOGIC;
                Zero : out STD_LOGIC;
                Overflow : out STD_LOGIC
            );
    end component;

    -- Signal declarations
    signal A_tb, B_tb, S_tb: STD_LOGIC_VECTOR (3 downto 0);
    signal subtract_tb, C_out_tb, zero: STD_LOGIC;

begin

    -- Instantiate the DUT (Design Under Test)
    uut: Add_Sub_4bit
        port map (
            A => A_tb,
            B => B_tb,
            S => S_tb,
            AddSubSelect => subtract_tb,
            Overflow => C_out_tb,
            Zero => zero
        );

    -- Stimulus process
    process
    begin
        --The binary value of 210353 is 110011100100110001
        --The binary Value of 210195 is 110011100000100011
```

```
        -- Initialize inputs
        A_tb <= "0001";
        B_tb <= "0011";
        subtract_tb <= '0';

        -- Apply inputs
        wait for 100 ns;
        A_tb <= "0011";
        B_tb <= "0010";
        subtract_tb <= '0';

        wait for 100 ns;
        A_tb <= "1001";
        B_tb <= "1000";
        subtract_tb <= '0';

        wait for 100 ns;
        A_tb <= "0011";
        B_tb <= "0011";
        subtract_tb <= '1';

        wait for 100 ns;
        A_tb <= "1111";
        B_tb <= "0110";
        subtract_tb <= '1';

        wait for 100 ns;
        A_tb <= "1100";
        B_tb <= "1001";
        subtract_tb <= '1';

        wait;
    end process;

end Behavioral;
```

RCA_3

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

entity RCA_3_Sim is
--  Port ( );
end RCA_3_Sim;

architecture Behavioral of RCA_3_Sim is
    component RCA_3 is
         Port (
               --A : in STD_LOGIC_VECTOR (2 downto 0);      -- 3-bit memory
address input
               --C_in : in STD_LOGIC;                       -- Since we have
to add, this shold be GROUND. i.e C_in = '0'
               B : in STD_LOGIC_VECTOR (2 downto 0);      -- Always this is
equals to 001
               S : out STD_LOGIC_VECTOR (2 downto 0);     -- Next memory
address will out
               C_out : out STD_LOGIC);                    --  This will not we
care becuase our programme only have 8 instructions.
    end component;

     -- Signal declarations
    signal B_tb, S_tb: STD_LOGIC_VECTOR (2 downto 0);
    signal C_out_tb: STD_LOGIC;

begin
    -- Instantiate the DUT (Design Under Test)
    uut: RCA_3
        port map (
            B => B_tb,
            S => S_tb,
            C_out => C_out_tb
        );

    process
    begin
        --The binary value of 210353 is 110011100100110001
        --The binary Value of 210195 is 110011100000100011
        -- Apply inputs
        B_tb <= "011";
        wait for 100 ns;

        B_tb <= "001";
        wait for 100 ns;
```

```
        B_tb <= "100";
        wait for 100 ns;

        B_tb <= "110";
        wait for 100 ns;

        B_tb <= "000";
        wait for 100 ns;

        B_tb <= "101";
        wait for 100 ns;

        B_tb <= "010";
        wait for 100 ns;

        B_tb <= "111";
        wait for 100 ns;

        wait;
    end process;

end Behavioral;
```

3-bit Program Counter (PC)

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

entity PC_3bit_Sim is
--  Port ( );
end PC_3bit_Sim;

architecture Behavioral of PC_3bit_Sim is
    -- Component declaration for the PC_3bit module
    component PC_3bit is
        Port (
            Reset : in STD_LOGIC;
            Clk : in STD_LOGIC;
            MemorySelect : inout STD_LOGIC_VECTOR (2 downto 0);
            JumpAddr : in STD_LOGIC_VECTOR (2 downto 0);
            JumpFlag : in STD_LOGIC
        );
    end component;
```

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

    -- Inputs
    signal Reset_tb : STD_LOGIC := '0';
    signal Clk_tb : STD_LOGIC := '0';
    signal Slow_Clk_tb : STD_LOGIC;
    signal JumpAddr_tb : STD_LOGIC_VECTOR (2 downto 0) := "000";
    signal JumpFlag_tb : STD_LOGIC := '0';

    -- Outputs
    signal MemorySelect_tb : STD_LOGIC_VECTOR (2 downto 0);

    -- Clock period definition
    constant Clk_period : time := 10 ns;
    constant T : time := Clk_period * 20;    -- This is the actual clock period
which feel the PC.

begin
    -- Instantiate the PC_3bit module
    UUT: PC_3bit
        port map (
            Reset => Reset_tb,
            Clk => Clk_tb,
            MemorySelect => MemorySelect_tb,
            JumpAddr => JumpAddr_tb,
            JumpFlag => JumpFlag_tb
        );

    UUT_Slow_Clk: Slow_Clk
         port map (
            Clk_in => Clk_tb,
            Clk_out => Slow_Clk_tb
        );


    -- Clock process
   Clk_process: process
    begin
        Clk_tb <= '0';
        wait for Clk_period/2;
        Clk_tb <= '1';
        wait for Clk_period/2;
    end process;
```

```
    -- Stimulus process
    process
    begin
        -- Reset assertion
        Reset_tb <= '1';
        wait for T;

        Reset_tb <= '0';

        -- Wait for 5 clock cycles
        wait for T * 5;

        -- Set JumpAddr_tb to test jump instruction
        JumpAddr_tb <= "001";
        JumpFlag_tb <= '1';
        wait for T;
        JumpFlag_tb <= '0';
        -- Wait for 3 clock cycles
        wait for T * 3;

        -- Set JumpAddr_tb to test jump instruction
        JumpAddr_tb <= "110";
        JumpFlag_tb <= '1';
        wait for T;
        JumpFlag_tb <= '0';
        -- Wait for 3 clock cycles
        wait for T * 3;

        -- Set JumpAddr_tb to test jump instruction
        JumpAddr_tb <= "100";
        JumpFlag_tb <= '1';
        wait for T;
        JumpFlag_tb <= '0';
        -- Wait for 3 clock cycles
        wait for T * 3;

        -- Set JumpAddr_tb to test jump instruction
        JumpAddr_tb <= "011";
        JumpFlag_tb <= '1';
        wait for T;
        JumpFlag_tb <= '0';
        -- Wait for 3 clock cycles
        wait for T * 3;

        -- Set JumpAddr_tb to test jump instruction
        JumpAddr_tb <= "000";
        JumpFlag_tb <= '1';
        wait for T;
```

```
        JumpFlag_tb <= '0';
        -- Wait for 3 clock cycles
        wait for T * 3;

       -- End simulation
        wait;
    end process;
end Behavioral;
```

D Flip Flop

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

entity D_FF_Sim is
end D_FF_Sim;

architecture Behavioral of D_FF_Sim is
    -- Component declaration for the D_FF module
    component D_FF is
        Port (
            D : in STD_LOGIC;
            Res : in STD_LOGIC;
            Clk : in STD_LOGIC;
            Q : out STD_LOGIC;
            Qbar : out STD_LOGIC
        );
    end component;

    -- Inputs/Outputs
    signal D_tb : STD_LOGIC := '0';
    signal Res_tb : STD_LOGIC := '0';
    signal Clk_tb : STD_LOGIC := '0';
    signal Q_tb : STD_LOGIC;
    signal Qbar_tb : STD_LOGIC;

    -- Clock period definition
    constant Clk_period : time := 10 ns;
    constant D_period : time := 12ns;  -- Oscillating period for D_tb

begin
    -- Instantiate the D_FF module
    UUT: D_FF
        port map (
            D => D_tb,
            Res => Res_tb,
```

```
            Clk => Clk_tb,
            Q => Q_tb,
            Qbar => Qbar_tb
        );

    -- Clock process
    Clock_process: process
    begin
        Clk_tb <= '0';
        wait for Clk_period/2;

        while (true) loop
            Clk_tb <= not Clk_tb;
            wait for Clk_period/2;
        end loop;
    end process;

    -- D_tb oscillating process
    D_process: process
    begin
        D_tb <= '0';
        wait for D_period/2;

        while (true) loop
            D_tb <= not D_tb;
            wait for D_period/2;
        end loop;
    end process;

    -- Stimulus process
    Stimulus_process: process
    begin
        -- Reset assertion
        Res_tb <= '1';
        wait for 100 ns;

        Res_tb <= '0';

        -- Wait for some time
        wait for Clk_period * 10;

        -- End simulation
        wait;
    end process;

end Behavioral;
```

Instruction Decoder

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

entity Instruction_Dec_Sim is
--  Port ( );
end Instruction_Dec_Sim;

architecture Behavioral of Instruction_Dec_Sim is
-- Component declaration
    component Instruction_Dec is
        Port (
            Reg_En : out STD_LOGIC_VECTOR (2 downto 0);
            Load_Select : out STD_LOGIC;
            Immediate_Val : out STD_LOGIC_VECTOR (3 downto 0);
            MuxA_Select : out STD_LOGIC_VECTOR (2 downto 0);
            MuxB_Select : out STD_LOGIC_VECTOR (2 downto 0);
            AddSub_Select : out STD_LOGIC;
            JmpFlag : out STD_LOGIC;
            JmpAddr : out STD_LOGIC_VECTOR (2 downto 0);
            Instruct_Bus : in STD_LOGIC_VECTOR (11 downto 0);
            JumpCheck : in STD_LOGIC_VECTOR (3 downto 0)
        );
    end component;

    -- Signals declaration
    signal Instruct_Bus_TB : STD_LOGIC_VECTOR (11 downto 0);
    signal JumpCheck_TB : STD_LOGIC_VECTOR (3 downto 0);
    signal Reg_En_TB : STD_LOGIC_VECTOR (2 downto 0);
    signal Load_Select_TB : STD_LOGIC;
    signal Immediate_Val_TB : STD_LOGIC_VECTOR (3 downto 0);
    signal MuxA_Select_TB : STD_LOGIC_VECTOR (2 downto 0);
    signal MuxB_Select_TB : STD_LOGIC_VECTOR (2 downto 0);
    signal AddSub_Select_TB : STD_LOGIC;
    signal JmpFlag_TB : STD_LOGIC;
    signal JmpAddr_TB : STD_LOGIC_VECTOR (2 downto 0);

begin
    -- Component instantiation
    UUT: Instruction_Dec
        Port map (
            Reg_En => Reg_En_TB,
            Load_Select => Load_Select_TB,
            Immediate_Val => Immediate_Val_TB,
            MuxA_Select => MuxA_Select_TB,
            MuxB_Select => MuxB_Select_TB,
```

```
            AddSub_Select => AddSub_Select_TB,
            JmpFlag => JmpFlag_TB,
            JmpAddr => JmpAddr_TB,
            Instruct_Bus => Instruct_Bus_TB,
            JumpCheck => JumpCheck_TB
        );

    -- Stimulus process
    stimulus_proc: process
    begin
        -- Test case 1: ADD instruction
        Instruct_Bus_TB <= "111010000110";
        JumpCheck_TB <= "0100";
        wait for 10 ns;

        -- Test case 2: SUB instruction
        Instruct_Bus_TB <= "101010000110";
        JumpCheck_TB <= "0100";
        wait for 10 ns;

        -- Test case 3: LOAD instruction with immediate value 5
         Instruct_Bus_TB <= "010001000101";
         JumpCheck_TB <= "0100";
         wait for 10 ns;

        Instruct_Bus_TB <= "100100110001";
        JumpCheck_TB <= "1111";
        wait for 10 ns;

        Instruct_Bus_TB <= "100000100011";
        JumpCheck_TB <= "0011";
        wait for 10 ns;

        wait;
    end process stimulus_proc;

end Behavioral;
```

Mux_2_to_1

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

entity Mux_2_to_1_Sim is
--  Port ( );
end Mux_2_to_1_Sim;

architecture Behavioral of Mux_2_to_1_Sim is
    component Mux_2_to_1 is
        Port ( D : in STD_LOGIC_VECTOR (1 downto 0);
               S : in STD_LOGIC;
               Y : out STD_LOGIC);
    end component;

    -- Signals for the testbench
    signal D_tb : STD_LOGIC_VECTOR (1 downto 0);
    signal S_tb : STD_LOGIC;
    signal Y_tb : STD_LOGIC;
begin
    uut: Mux_2_to_1
        port map (D => D_tb, S => S_tb, Y => Y_tb);

    -- Stimulus process
    stimulus: process
    begin
        -- Test case 1 for check whether D(0) is the outhput
        D_tb <= "00";
        S_tb <= '0';
        wait for 100 ns;

        D_tb <= "11";
        S_tb <= '0';
        wait for 100 ns;

        D_tb <= "01";
        S_tb <= '0';
        wait for 100 ns;

        D_tb <= "10";
        S_tb <= '0';
        wait for 100 ns;

        -- Test case 2 for check whether D(1) is the outhput
        D_tb <= "00";
        S_tb <= '1';
```

```
        wait for 100 ns;

        D_tb <= "11";
        S_tb <= '1';
        wait for 100 ns;

        D_tb <= "01";
        S_tb <= '1';
        wait for 100 ns;

        D_tb <= "10";
        S_tb <= '1';
        --wait for 100 ns;

        -- End simulation
        wait;
    end process;

end Behavioral;
```

Mux_2_way_3_bit

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

entity Mux_2way_3bit_Sim is
--  Port ( );
end Mux_2way_3bit_Sim;

architecture Behavioral of Mux_2way_3bit_Sim is
    component Mux_2_to_1 is
        Port ( D : in STD_LOGIC_VECTOR (1 downto 0);
               S : in STD_LOGIC;
               Y : out STD_LOGIC);
    end component;

    component Mux_2way_3bit is
            Port (
               jmp_adrs : in STD_LOGIC_VECTOR (2 downto 0);
               adder_3bit : in STD_LOGIC_VECTOR (2 downto 0);
               jmp_flag : in STD_LOGIC;
               output : out STD_LOGIC_VECTOR (2 downto 0));
        end component;
```

```
    -- Signals for the testbench
    signal jmp_adrs_tb : STD_LOGIC_VECTOR (2 downto 0);
    signal adder_3bit_tb : STD_LOGIC_VECTOR (2 downto 0);
    signal jmp_flag_tb : STD_LOGIC;
    signal output_tb : STD_LOGIC_VECTOR (2 downto 0);

begin
    uut: Mux_2way_3bit
        port map (jmp_adrs => jmp_adrs_tb, adder_3bit => adder_3bit_tb,
jmp_flag => jmp_flag_tb, output => output_tb);

    -- Stimulus process
    stimulus: process
    begin
    --The binary value of 210353 is 110011100100110001
    --The binary Value of 210195 is 110011100000100011
    -- Test case 1 ( activating jump_address pin)
        jmp_adrs_tb <= "001";
        adder_3bit_tb <= "011";
        jmp_flag_tb <= '0';
        wait for 100 ns;

        jmp_adrs_tb <= "110";
        adder_3bit_tb <= "100";
        jmp_flag_tb <= '0';
        wait for 100 ns;

        jmp_adrs_tb <= "100";
        adder_3bit_tb <= "000";
        jmp_flag_tb <= '0';
        wait for 100 ns;

        jmp_adrs_tb <= "100";
        adder_3bit_tb <= "100";
        jmp_flag_tb <= '0';
        wait for 100 ns;

        jmp_adrs_tb <= "011";
        adder_3bit_tb <= "011";
        jmp_flag_tb <= '0';
        wait for 100 ns;

        jmp_adrs_tb <= "110";
        adder_3bit_tb <= "110";
        jmp_flag_tb <= '0';
        wait for 100 ns;

  -- Test case 2 ( activating 3bit adder pin)
```

```
        jmp_adrs_tb <= "001";
        adder_3bit_tb <= "011";
        jmp_flag_tb <= '1';
        wait for 100 ns;

        jmp_adrs_tb <= "110";
        adder_3bit_tb <= "100";
        jmp_flag_tb <= '1';
        wait for 100 ns;

        jmp_adrs_tb <= "100";
        adder_3bit_tb <= "000";
        jmp_flag_tb <= '1';
        wait for 100 ns;

        jmp_adrs_tb <= "100";
        adder_3bit_tb <= "100";
        jmp_flag_tb <= '1';
        wait for 100 ns;

        jmp_adrs_tb <= "011";
        adder_3bit_tb <= "011";
        jmp_flag_tb <= '1';
        wait for 100 ns;

        jmp_adrs_tb <= "110";
        adder_3bit_tb <= "110";
        jmp_flag_tb <= '1';
        wait for 100 ns;

  -- End simulation
        wait;
    end process;

end Behavioral;
```

Mux_8_way_4_bit

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

entity Mux_8way_4bit_Sim is
--  Port ( );
end Mux_8way_4bit_Sim;

architecture Behavioral of Mux_8way_4bit_Sim is
    component Mux_8way_4bit is
        Port (  reg0 : in STD_LOGIC_VECTOR (3 downto 0);
                reg1 : in STD_LOGIC_VECTOR (3 downto 0);
                reg2 : in STD_LOGIC_VECTOR (3 downto 0);
                reg3 : in STD_LOGIC_VECTOR (3 downto 0);
                reg4 : in STD_LOGIC_VECTOR (3 downto 0);
                reg5 : in STD_LOGIC_VECTOR (3 downto 0);
                reg6 : in STD_LOGIC_VECTOR (3 downto 0);
                reg7 : in STD_LOGIC_VECTOR (3 downto 0);
                Reg_select : in STD_LOGIC_VECTOR (2 downto 0);
                output : out STD_LOGIC_VECTOR (3 downto 0));
    end component;

    signal reg0_tb : STD_LOGIC_VECTOR (3 downto 0);
    signal reg1_tb : STD_LOGIC_VECTOR (3 downto 0);
    signal reg2_tb : STD_LOGIC_VECTOR (3 downto 0);
    signal reg3_tb : STD_LOGIC_VECTOR (3 downto 0);
    signal reg4_tb : STD_LOGIC_VECTOR (3 downto 0);
    signal reg5_tb : STD_LOGIC_VECTOR (3 downto 0);
    signal reg6_tb : STD_LOGIC_VECTOR (3 downto 0);
    signal reg7_tb : STD_LOGIC_VECTOR (3 downto 0);
    signal Reg_select_tb : STD_LOGIC_VECTOR (2 downto 0);
    signal output_tb : STD_LOGIC_VECTOR (3 downto 0);

begin
    uut: Mux_8way_4bit
        port map (  reg0 => reg0_tb,
                    reg1 => reg1_tb,
                    reg2 => reg2_tb,
                    reg3 => reg3_tb,
                    reg4 => reg4_tb,
                    reg5 => reg5_tb,
                    reg6 => reg6_tb,
                    reg7 => reg7_tb,
                    Reg_select => Reg_select_tb,
                    output => output_tb);
```

```
    process
    begin
    --The binary value of 210353 is 110011100100110001
    --The binary Value of 210195 is 110011100000100011
            reg0_tb <= "0000";
            reg1_tb <= "1111";
            reg2_tb <= "1010";
            reg3_tb <= "0101";
            reg4_tb <= "1100";
            reg5_tb <= "0011";
            reg6_tb <= "0110";
            reg7_tb <= "1001";

            Reg_select_tb <= "001";
            wait for 100 ns;

            Reg_select_tb <= "110";
            wait for 100 ns;

            Reg_select_tb <= "100";
            wait for 100 ns;

            Reg_select_tb <= "011";
            wait for 100 ns;

            Reg_select_tb <= "000";
            wait for 100 ns;

            Reg_select_tb <= "101";
            wait for 100 ns;

            Reg_select_tb <= "111";
            wait for 100 ns;


      end process;
end Behavioral;
```

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

entity Nanoprocessor_Sim is
--  Port ( );
end Nanoprocessor_Sim;

architecture Behavioral of Nanoprocessor_Sim is

    component Nanoprocessor is
    Port ( Reset : in STD_LOGIC;
           clk : in STD_LOGIC;
           Zero_LED : out STD_LOGIC;
           Overflow_LED : out STD_LOGIC;
           LED0 : out STD_LOGIC;
           LED1 : out STD_LOGIC;
           LED2 : out STD_LOGIC;
           LED3 : out STD_LOGIC
    );
    end component;

signal reset : STD_LOGIC;
signal clock : STD_LOGIC;
signal zero_LED : STD_LOGIC;
signal overflow_LED : STD_LOGIC;
signal led0 : STD_LOGIC;
signal led1 : STD_LOGIC;
signal led2 : STD_LOGIC;
signal led3 : STD_LOGIC;

constant Clk_period : time := 10 ns;
constant T : time := Clk_period * 20;

begin
    uut : Nanoprocessor
    port map (
        Reset => reset,
        clk => clock,
        Zero_LED => zero_LED,
        Overflow_LED => overflow_LED,
        LED0 => led0,
        LED1 => led1,
        LED2 => led2,
        LED3 => led3
    );
```

```
    Clock_process: process
      begin
        while now < 100 us loop
          clock <= '0';
          wait for Clk_period/2;
          clock <= '1';
          wait for Clk_period/2;
        end loop;
        wait;
      end process;

    Stimulus_process: process
    begin
        -- Initialize inputs
        reset <= '1';
        wait for T;

        reset <= '0';
        wait for T*7;

        reset <= '1';
        wait for T;

        reset <= '0';
        wait;
    end process;
end Behavioral;
```

Program Rom

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

entity Program_ROM_Sim is
--  Port ( );
end Program_ROM_Sim;

architecture Behavioral of Program_ROM_Sim is

component Program_ROM is
    Port ( MemSelect : in STD_LOGIC_VECTOR (2 downto 0);
           InstructBus : out STD_LOGIC_VECTOR (11 downto 0));
end component;

signal MemSelect : STD_LOGIC_VECTOR (2 downto 0);
signal InstructBus : STD_LOGIC_VECTOR (11 downto 0);

begin
    uut : Program_ROM
    port map (
        MemSelect => MemSelect,
        InstructBus => InstructBus
    );

    process
    begin
                MemSelect <= "000";
                wait for 100 ns;

                MemSelect <= "001";
                wait for 100 ns;

                MemSelect <= "010";
                wait for 100 ns;

                MemSelect <= "011";
                wait for 100 ns;

                MemSelect <= "100";
                wait for 100 ns;

                MemSelect <= "101";
                wait for 100 ns;

                MemSelect <= "110";
```

```
                wait for 100 ns;

                MemSelect <= "111";
                wait for 100 ns;

    end process;
end Behavioral;
```

Register Bank

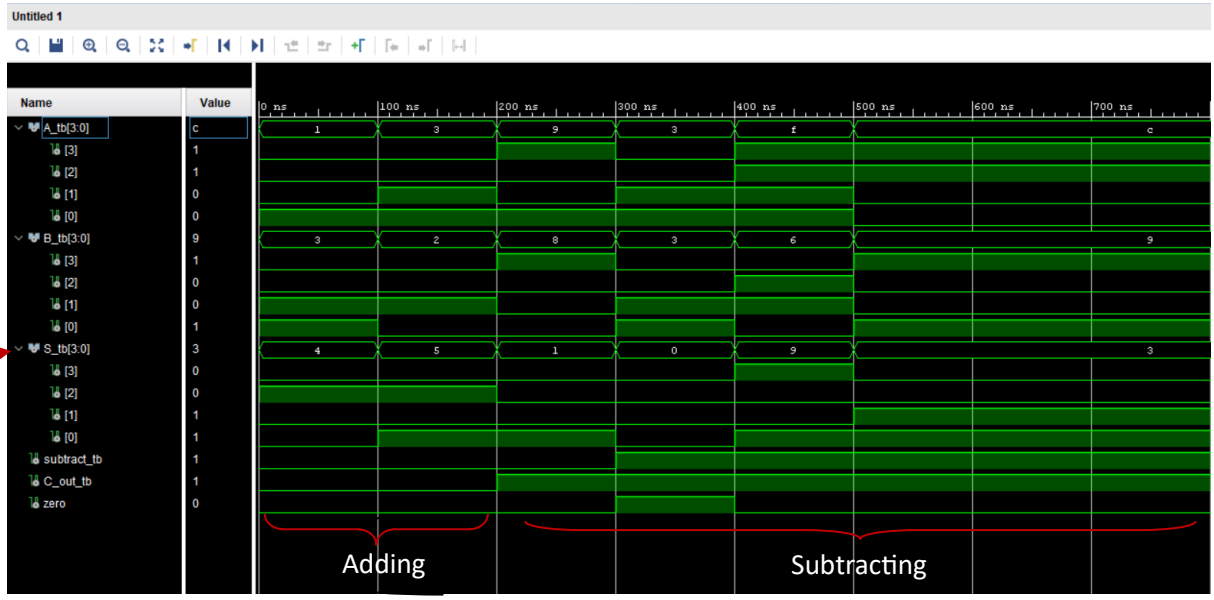
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Register_Bank_Sim is
--  Port ( );
end Register_Bank_Sim;

architecture Behavioral of Register_Bank_Sim is

  component Register_Bank is
    Port ( D_input : in STD_LOGIC_VECTOR (3 downto 0);
           Reg_En : in STD_LOGIC_VECTOR (2 downto 0);
           Clock : in STD_LOGIC;
           Reset : in STD_LOGIC;
           D_out0 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out1 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out2 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out3 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out4 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out5 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out6 : out STD_LOGIC_VECTOR (3 downto 0);
           D_out7 : out STD_LOGIC_VECTOR (3 downto 0));
  end component;

  signal D_input_tb : STD_LOGIC_VECTOR (3 downto 0);
  signal Reg_En_tb : STD_LOGIC_VECTOR (2 downto 0);
  signal Clock_tb : STD_LOGIC := '0';
  signal Rst : STD_LOGIC;
  signal D_out0_tb : STD_LOGIC_VECTOR (3 downto 0);
  signal D_out1_tb : STD_LOGIC_VECTOR (3 downto 0);
  signal D_out2_tb : STD_LOGIC_VECTOR (3 downto 0);
  signal D_out3_tb : STD_LOGIC_VECTOR (3 downto 0);
  signal D_out4_tb : STD_LOGIC_VECTOR (3 downto 0);
  signal D_out5_tb : STD_LOGIC_VECTOR (3 downto 0);
```

```
  signal D_out6_tb : STD_LOGIC_VECTOR (3 downto 0);
  signal D_out7_tb : STD_LOGIC_VECTOR (3 downto 0);

  constant Clk_period : time := 100 ns;

begin
  uut: Register_Bank
  port map (  D_input => D_input_tb,
              Reg_En => Reg_En_tb,
              Clock => Clock_tb,
              Reset => Rst,
              D_out0 => D_out0_tb,
              D_out1 => D_out1_tb,
              D_out2 => D_out2_tb,
              D_out3 => D_out3_tb,
              D_out4 => D_out4_tb,
              D_out5 => D_out5_tb,
              D_out6 => D_out6_tb,
              D_out7 => D_out7_tb);

  Clock_process: process
  begin
    while now < 4000 ns loop
      Clock_tb <= '0';
      wait for Clk_period/2;
      Clock_tb <= '1';
      wait for Clk_period/2;
    end loop;
    wait;
  end process;

  Stimulus_process: process
  begin
  --The binary value of 210353 is 110011100100110001
  --The binary Value of 210195 is 110011100000100011

    -- Initialize inputs
    Rst <= '1';
    wait for Clk_period;

    -- Apply inputs
    Rst <= '0';
    D_input_tb <= "0001";
    Reg_En_tb <= "011";
    wait for Clk_period;

    -- Apply inputs
    D_input_tb <= "0011";
```

```
    Reg_En_tb <= "100";
    wait for Clk_period;

    Rst <= '0';
    D_input_tb <= "1001";
    Reg_En_tb <= "000";
    wait for Clk_period;

    Rst <= '0';
    D_input_tb <= "0010";
    Reg_En_tb <= "110";
    wait for Clk_period;

    Rst <= '0';
    D_input_tb <= "1000";
    Reg_En_tb <= "001";
    wait for Clk_period;

    Rst <= '1';
    D_input_tb <= "0000";   --Remember reset D_input also to zero after reset
registers
    --Reg_En_tb <= "101";
    wait for Clk_period;
    Rst <= '0';

    wait;
  end process;

end Behavioral;
```

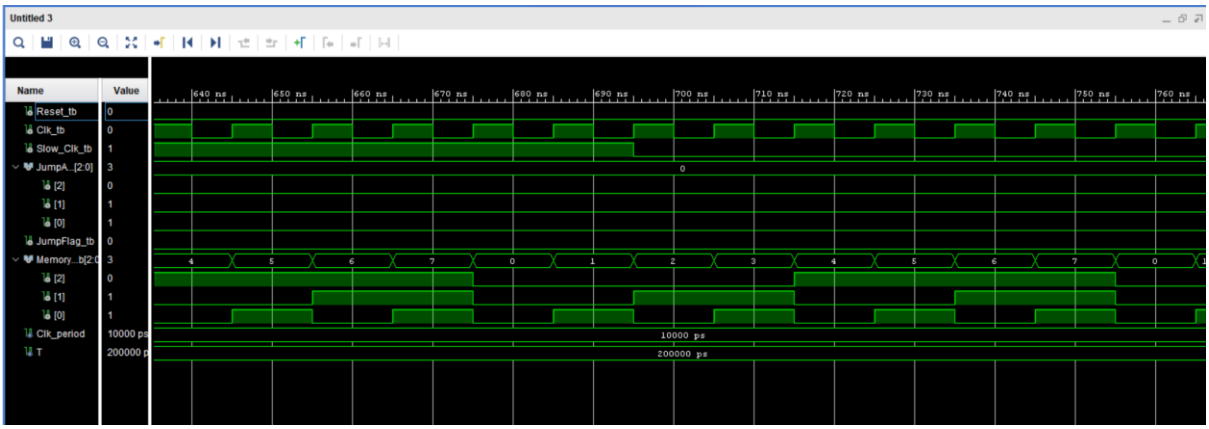
Timing Diagrams:

4-bit Add/Subtract unit

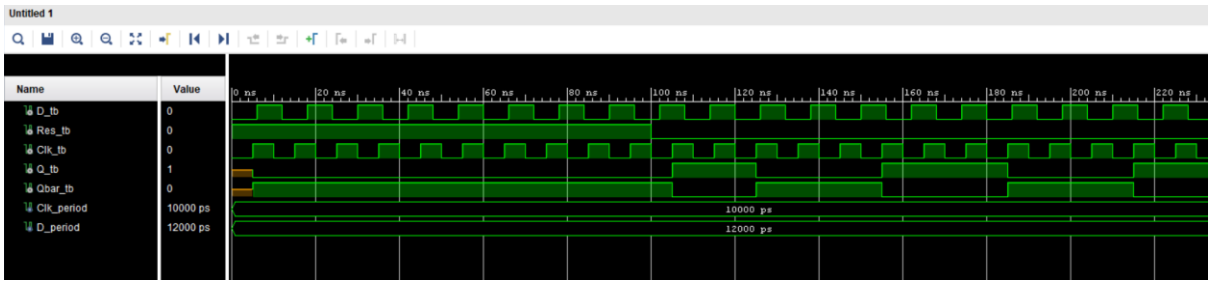


RCA_3

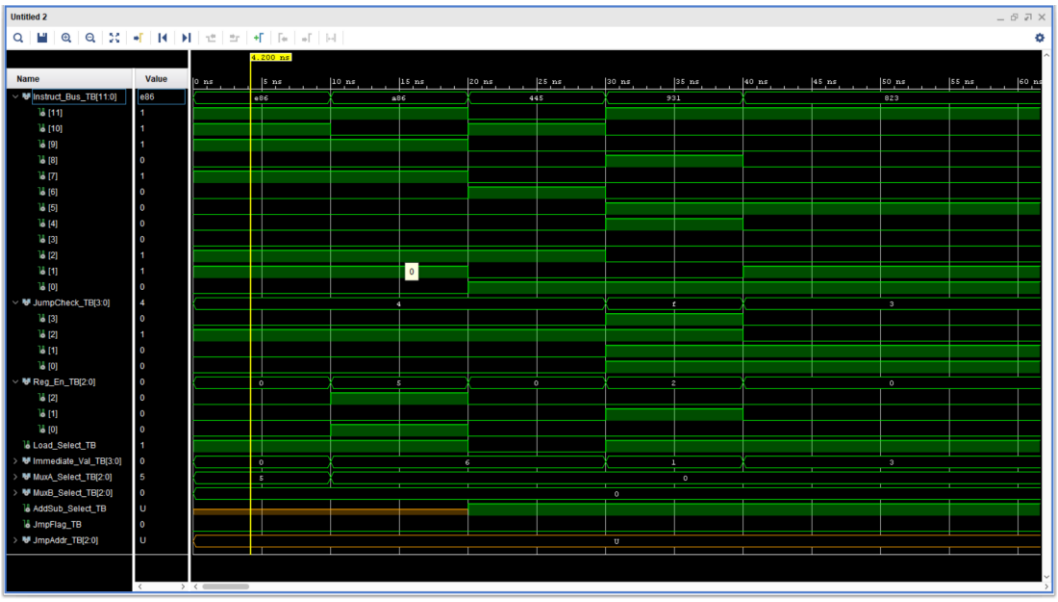
3-bit Program Counter (PC)



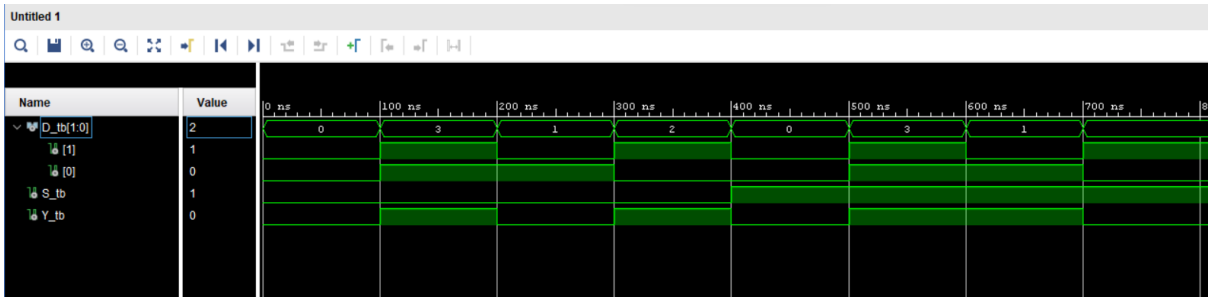
D Flip Flop



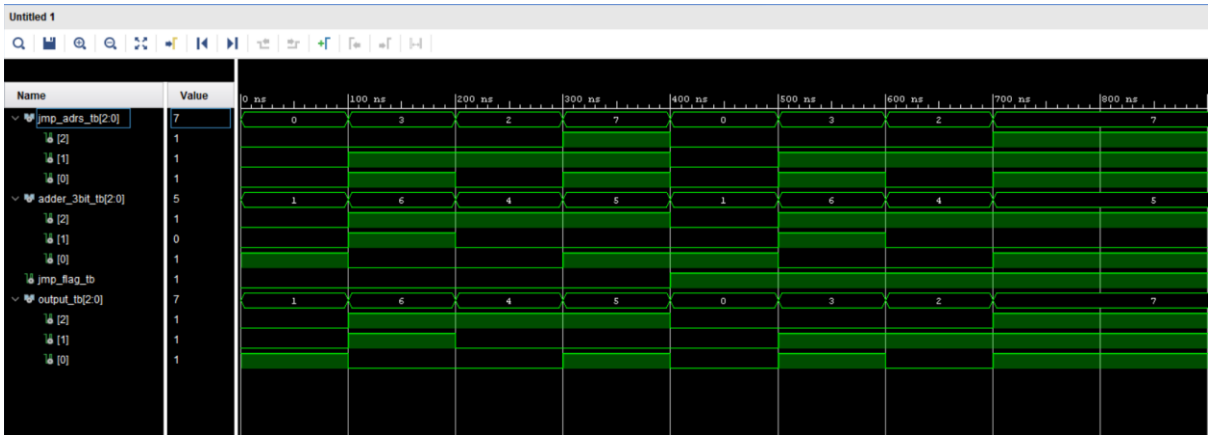
Instruction Decoder



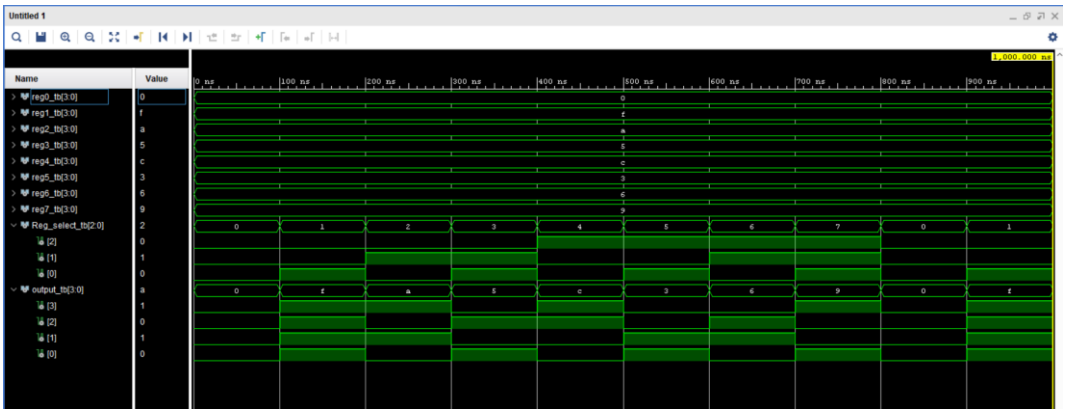
Mux_2_to_1



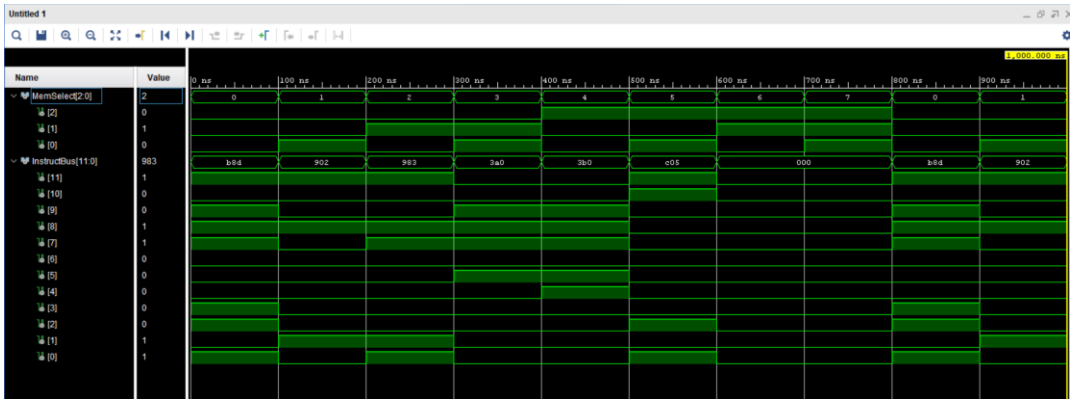
Mux_2_way_3_bit



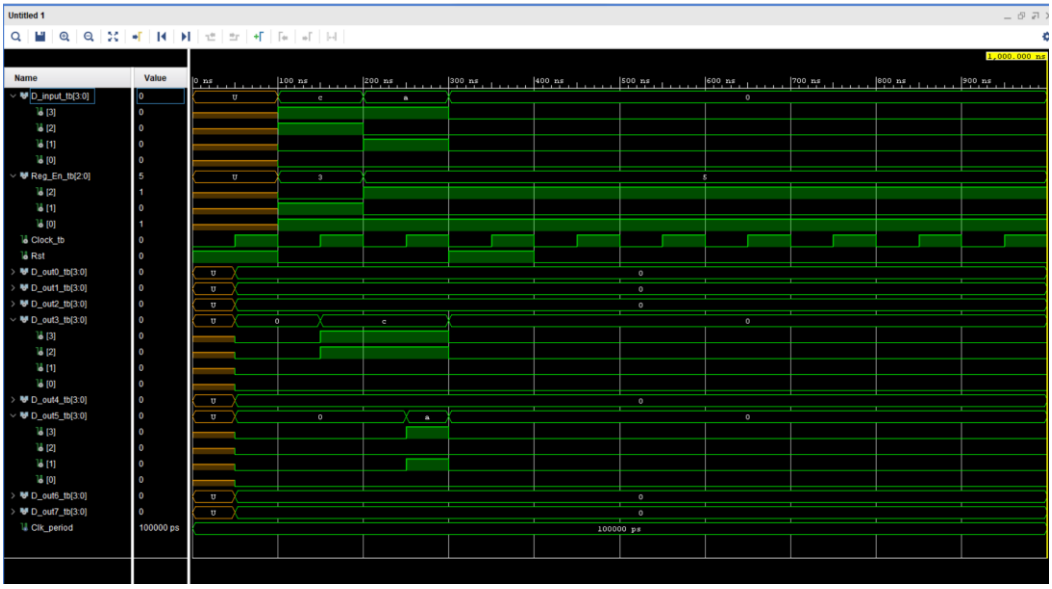
Mux_8_way_4_bit



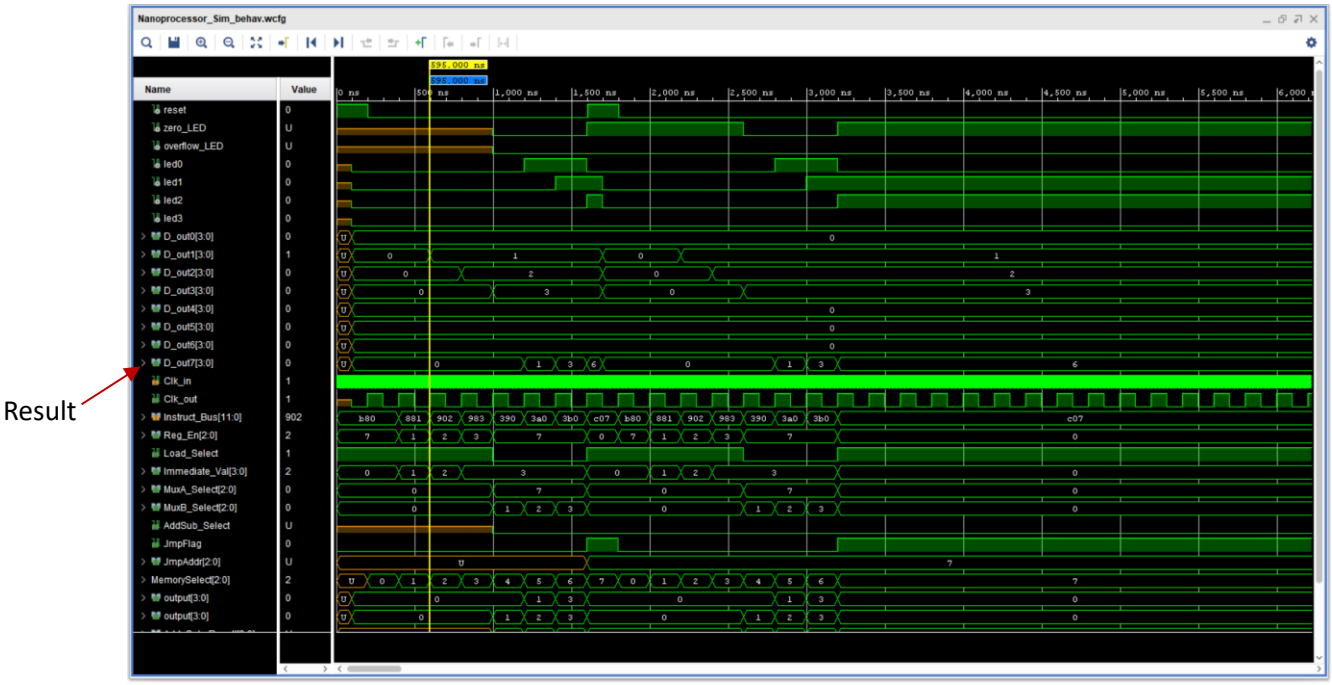
Program Rom



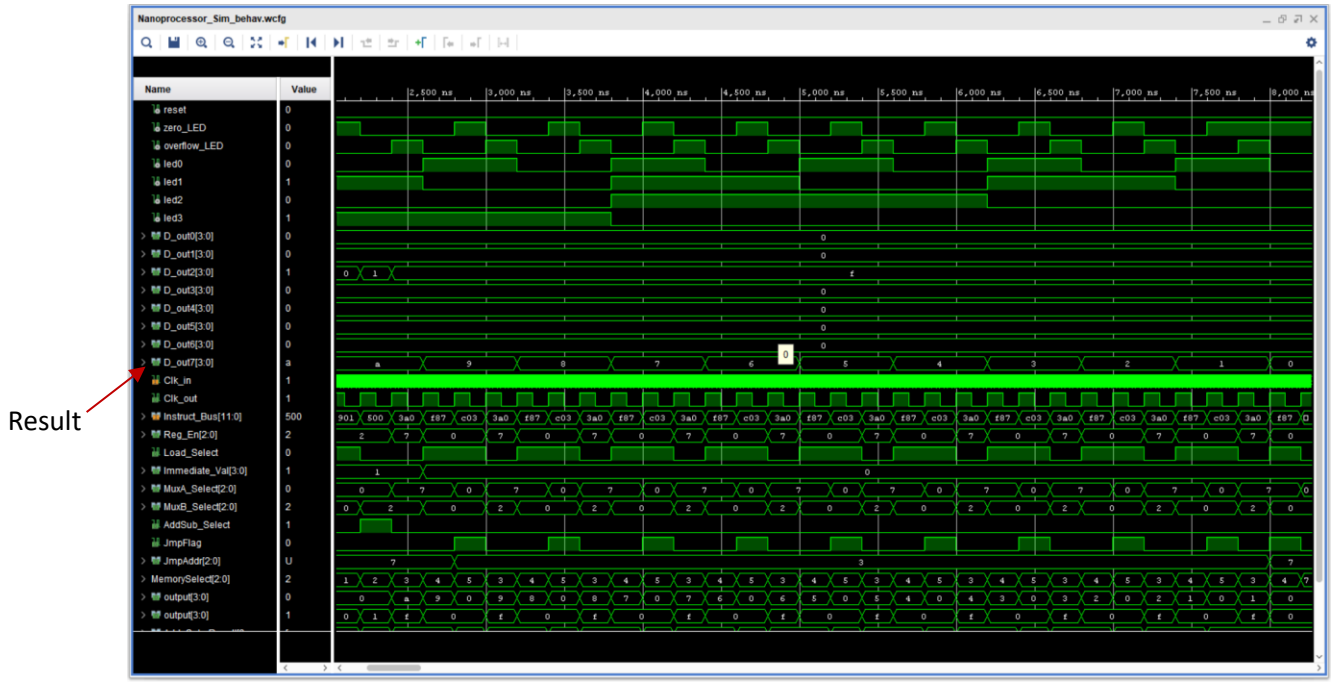
Register Bank



Nano Processor



For adding all the integers from 1 to 3 programme



For the programme of start from 10 and decrease it to 0 by decrement one by 1

Conclusions:

We successfully designed a 4-bit nanoprocessor capable of executing a simple set of instructions.

- Designed and developed a 4-bit arithmetic unit capable of adding and subtracting signed integers.
- Developed a 3-bit adder to increment the Program Counter (PC).
- Implemented a 3-bit Program Counter (PC) using D Flip-Flops with a clear/reset input. The PC was reset to 0 when required.
- Built k-way b-bit multiplexers (2-way, 8-way) to handle data and control signal selection.
- Designed and built the Instruction Decoder circuit to activate necessary components based on the instructions to be executed. We carefully activated only the required modules for each instruction, such as enabling specific registers, selecting inputs from multiplexers, and setting appropriate flags.

Throughout the lab, teamwork and communication skills were practiced among team members. We divided the workload and collaborated on different components simultaneously.

Overall, this lab provided valuable hands-on experience in designing and implementing a simple nanoprocessor, fostering an understanding of microprocessor architecture, teamwork, and digital design principles.

Contribution of member:

Gunasena H.A.S.I

- Register_Bank (time spent :- 2 days)
- Add_Sub_4bit (time spent :- 1 days)
- Instruction_Dec (time spent :- 3 days
- Program_ROM (time spent :- 1 days)
- PC_3bit (time spent :- 3 days)
- Mux_2way_4bit (time spent :- 1 days)
- Nanoprocessor (time spent :- 2 days)
- Lab sheet (time spent :- 1 days)

Madusanka G.I.D.L

- Mux_8way_4bit (time spent :- 1 days)
- Mux_2way_3bit (time spent :- 1 days)
- Mux_2way_4bit (time spent :- 1 days)
- Add_Sub_4bit (time spent :- 1 days)
- 3_bit_adder (time spent : - around 1 hour)
- Lookup table (time spent : - around 2 hours)
- Nanoprocessor (time spent : - 2 days)
- Lab sheet (time spent :- 1 days)

Total time spent :- about 13 days