GROUP 31 **LAB 09/10**

NANO PROCESSOR DESIGN COMPETTITION

LAB REPORT

- **A.A.A. Jasmin - 200238N**
- **K.G.T.R. Kumaratunga - 200327L**
- **K.V.R. Kurukulasooriya - 200332X**
- **G.S.D. Kuruppu - 200333C**
- **G.M.P. Silva - 200605M**

TASK ASSIGNED:

1. Design and develop a 4-bit arithmetic unit that can add and subtract signed integers
2. Modify 4-bit RCA developed to a 3-bit adder
3. Design and develop 3-bit program counter which needs to reset to 0 when required
4. Design and develop *k*-way *b*-bit multiplexers or tri-state busses
5. Design and develop register bank with the required functionality
6. Extend ROM based LUT developed to store the assembly program to run
7. Decode instructions to activate necessary components on the processor
8. Connect all built and modified components to the design the nano processor.
9. Write a constraint file to connect inputs and outputs.
10. Verify their functionality via simulation and on the development board

ASSEMBLY PROGRAM AND MACHINE CODE REPRESENTATION

MOVI R1,1	**10 001 000 0001**
MOVI R2,2	**10 010 000 0010**
MOVI R7,3	**10 111 000 0011**
ADD R7,R2	**00 111 010 0000**
ADD R7,R1	**00 111 001 0000**
JZR R0,7	**11 000 0000 111**
ADD R0,R0	**00 000 000 0000**
JZR R0,5	**11 000 0000 101**

Synthesis Report Summary

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	52	0	20800	0.25
LUT as Logic	52	0	20800	0.25
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	37	Flop & Latch
LUT5	25	LUT
OBUF	17	IO
LUT4	17	LUT
FDCE	16	Flop & Latch
LUT6	13	LUT
CARRY4	8	CarryLogic
LUT3	5	LUT
IBUF	2	IO
LUT2	1	LUT
LUT1	1	LUT
BUFG	1	Clock

How we achieved the above results

- Instead of using previously built components, we built 3 to 8 decoders, multiplexers, and 4 bit adder subtractors from ground up that resulted in 52 slice LUTs and 53 slice registers as flip flops.

VHDL FILES

SLOW CLOCK

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity Slow_Clk is
    Port ( Clk_in : in STD_LOGIC;
           Clk_out : out STD_LOGIC := '1');
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 = 1) then
--          if (count = 100000000) then
             clk_status <= not clk_status;
             Clk_out <= clk_status;
             count <= 1;
          end if;
       end if;
    end process;
```

INSTRUCTION DECODER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Instruction_Decoder is
    Port ( Ins : in STD_LOGIC_VECTOR (11 downto 0);
           RegJmp : in STD_LOGIC_VECTOR (3 downto 0);
           RegEn : out STD_LOGIC_VECTOR (2 downto 0);
           LoadSel : out STD_LOGIC;
           ImVal : out STD_LOGIC_VECTOR (3 downto 0);
           RegSel1 : out STD_LOGIC_VECTOR (2 downto 0);
           RegSel2 : out STD_LOGIC_VECTOR (2 downto 0);
           AddSubSel : out STD_LOGIC;
           Jmp : out STD_LOGIC;
           AddressJmp : out STD_LOGIC_VECTOR (2 downto 0));
end Instruction_Decoder;

architecture Behavioral of Instruction_Decoder is
begin
RegEn <= Ins(9 downto 7);
LoadSel <=Ins(11) AND NOT(Ins(10));
ImVal <= Ins(3 downto 0);
RegSel1 <= Ins(9 downto 7);
RegSel2 <= Ins(6 downto 4);
AddSubSel <= NOT(Ins(11)) AND Ins(10);
AddressJmp <= Ins(2 downto 0);
Jmp <= Ins(11) AND Ins(10) AND NOT (RegJmp(3) OR RegJmp(2) OR RegJmp(1) OR RegJmp(0));

end Behavioral;
```

REGISTER BANK

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Reg_Bank is
    Port ( En : in STD_LOGIC_VECTOR (2 downto 0);
           Clk : in STD_LOGIC;
           Res : in STD_LOGIC;
           D : in STD_LOGIC_VECTOR (3 downto 0);
           Q0 : out STD_LOGIC_VECTOR (3 downto 0) := "0000";
           Q1 : out STD_LOGIC_VECTOR (3 downto 0) := "0000";
           Q2 : out STD_LOGIC_VECTOR (3 downto 0) := "0000";
           Q3 : out STD_LOGIC_VECTOR (3 downto 0) := "0000";
           Q4 : out STD_LOGIC_VECTOR (3 downto 0) := "0000";
           Q5 : out STD_LOGIC_VECTOR (3 downto 0) := "0000";
           Q6 : out STD_LOGIC_VECTOR (3 downto 0) := "0000";
           Q7 : out STD_LOGIC_VECTOR (3 downto 0) := "0000");
end Reg_Bank;
architecture Behavioral of Reg_Bank is
    Component Reg_4
        Port ( En : in STD_LOGIC;
               Clk : in STD_LOGIC;
               D : in STD_LOGIC_VECTOR (3 downto 0);
               Res : in STD_LOGIC;
               Q : out STD_LOGIC_VECTOR (3 downto 0):= "0000");
    End Component;
    Component Decoder_3_to_8
        Port ( I : in STD_LOGIC_VECTOR (2 downto 0);
               Y0 : out STD_LOGIC;
               Y1 : out STD_LOGIC;
               Y2 : out STD_LOGIC;
               Y3 : out STD_LOGIC;
               Y4 : out STD  LOGIC;
               Y5 : out STD_LOGIC;
               Y6 : out STD_LOGIC;
               Y7 : out STD_LOGIC);
    End Component;
    Signal EN0, EN1, EN2, EN3, EN4, EN5, EN6, EN7 : STD_LOGIC;
begin
    Decoder : Decoder_3_to_8
        Port Map(
            I => En,
            Y0 => EN0,
            Y1 => EN1,
            Y2 => EN2,
            Y3 => EN3,
            Y4 => EN4,
            Y5 => EN5,
            Y6 => EN6,
            Y7 => EN7);
     Reg_4_0 : Reg_4
        Port Map(
            En => EN0,
            Clk => Clk,
            D => "0000",
            Res => Res,
            Q => Q0);
    Reg_4_1 : Reg_4
        Port Map(
            En => EN1,
            Clk => Clk,
            D => D,
            Res => Res,
            Q => Q1);
```

```
Reg_4_2 : Reg_4
   Port Map(
      En => EN2,
      Clk => Clk,
      D => D,
      Res => Res,
      Q => Q2);
Reg_4_3 : Reg_4
   Port Map(
      En => EN3,
      Clk => Clk,
      D => D,
      Res => Res,
      Q => Q3);
 Reg_4_4 : Reg_4
   Port Map(
      En => EN4,
      Clk => Clk,
      D => D,
      Res => Res,
      Q => Q4);
Reg_4_5 : Reg_4
   Port Map(
      En => EN5,
      Clk => Clk,
      D => D,
      Res => Res,
      Q => Q5);
Reg_4_6 : Reg_4
   Port Map(
      En => EN6,
      Clk => Clk,
      D => D,
      Res => Res,
      Q => Q6);
   Reg_4_7 : Reg_4
      Port Map(
      En => EN7,
      Clk => Clk,
      D => D,
      Res => Res,
      Q => Q7);

end Behavioral;
```

MUX 8_1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Mux_4_8_to_1 is
   Port ( I0 : in STD_LOGIC_VECTOR (3 downto 0);
          I1 : in STD_LOGIC_VECTOR (3 downto 0);
          I2 : in STD_LOGIC_VECTOR (3 downto 0);
          I3 : in STD_LOGIC_VECTOR (3 downto 0);
          I4 : in STD_LOGIC_VECTOR (3 downto 0);
          I5 : in STD_LOGIC_VECTOR (3 downto 0);
          I6 : in STD_LOGIC_VECTOR (3 downto 0);
          I7 : in STD_LOGIC_VECTOR (3 downto 0);
          S : in STD_LOGIC_VECTOR (2 downto 0);
          Y : out STD_LOGIC_VECTOR (3 downto 0));
end Mux_4_8_to_1;
architecture Behavioral of Mux_4_8_to_1 is
Component Mux_4_2_to_1
   Port ( I0 : in STD_LOGIC_VECTOR (3 DOWNTO 0);
          I1 : in STD_LOGIC_VECTOR  (3 DOWNTO 0);
          S : in STD_LOGIC;
          Y : out STD_LOGIC_VECTOR (3 DOWNTO 0));
End Component;
Signal Y0, Y1, Y2, Y3, Y4, Y5 : STD_LOGIC_VECTOR  (3
DOWNTO 0);
begin
Mux_4_2_to_1_0 : Mux_4_2_to_1
   Port Map(
      I0 => I0,
      I1 => I1,
      S => S(0),
      Y => Y0
   );
Mux_4_2_to_1_1 : Mux_4_2_to_1
   Port Map(
      I0 => I2,
      I1 => I3,
      S => S(0),
      Y => Y1
   );
Mux_4_2_to_1_2 : Mux_4_2_to_1
   Port Map(
      I0 => I4,
      I1 => I5,
      S => S(0),
      Y => Y2
   );
Mux_4_2_to_1_3 : Mux_4_2_to_1
   Port Map(
      I0 => I6,
      I1 => I7,
      S => S(0),
      Y => Y3
   );
Mux_4_2_to_1_4 : Mux_4_2_to_1
   Port Map(
      I0 => Y0,
      I1 => Y1,
      S => S(1),
      Y => Y4
   );
Mux_4_2_to_1_5 : Mux_4_2_to_1
   Port Map(
      I0 => Y2,
      I1 => Y3,
```

```
      S => S(1),
      Y => Y5
   );

Mux_4_2_to_1_6 : Mux_4_2_to_1
   Port Map(
      I0 => Y4,
      I1 => Y5,
      S => S(2),
      Y => Y
   );

end Behavioral;
```

MUX 2_1

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

-- Uncomment the following library declaration if

entity Mux_3_2_to_1 is
   Port ( I0 : in STD_LOGIC_VECTOR (2 DOWNTO 0);
          I1 : in STD_LOGIC_VECTOR  (2 DOWNTO 0);
          S : in STD_LOGIC;
          Y : out STD_LOGIC_VECTOR (2 DOWNTO 0));
end Mux_3_2_to_1;

architecture Behavioral of Mux_3_2_to_1 is

Component Mux_2_to_1
   Port ( I0 : in STD_LOGIC;
          I1 : in STD_LOGIC;
          S : in STD_LOGIC;
          Y : out STD_LOGIC);
End Component;

begin

Mux_2_to_1_0: Mux_2_to_1
   Port Map(
      I0 => I0(0),
      I1 => I1(0),
      S => S,
      Y => Y(0)
   );
```

```
Mux_2_to_1_1: Mux_2_to_1
   Port Map(
      I0 => I0(1),
      I1 => I1(1),
      S => S,
      Y => Y(1)
   );

Mux_2_to_1_2: Mux_2_to_1
   Port Map(
      I0 => I0(2),
      I1 => I1(2),
      S => S,
      Y => Y(2)
   );

end Behavioral;
```

PROGRAM COUNTER

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

entity Program_Counter is
   Port ( D : in STD_LOGIC_VECTOR (2 downto 0);
          Clk : in STD_LOGIC;
          Res : in STD_LOGIC;
          Q : out STD_LOGIC_VECTOR (2 downto 0));
end Program_Counter;

architecture Behavioral of Program_Counter is

   Component D_FF
      Port ( D : in STD_LOGIC;
             Clk : in STD_LOGIC;
             Res : in STD_LOGIC;
             Q : out STD_LOGIC);
   End Component;

begin

   D_FF_0 : D_FF
      Port Map(
         D => D(0),
         Clk => Clk,
         Res => Res,
         Q => Q(0)
      );

   D_FF_1 : D_FF
      Port Map(
```

```
        D => D(1),
        Clk => Clk,
        Res => Res,
        Q => Q(1)
    );

  D_FF_2 : D_FF
    Port Map(
        D => D(2),
        Clk => Clk,
        Res => Res,
        Q => Q(2)
    );

end Behavioral;
```

4 BIT ADD SUB UNIT

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Add_Sub_4 is
   Port ( Control : in STD_LOGIC;
          A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          S : out STD_LOGIC_VECTOR (3 downto 0);
          Overflow : out STD_LOGIC;
          Zero : out STD_LOGIC);
end Add_Sub_4;
architecture Behavioral of Add_Sub_4 is
Component RCA_4
   Port (
      A, B : in STD_LOGIC_VECTOR (3 downto 0);
      C_in : in STD_LOGIC;
      S : out STD_LOGIC_VECTOR (3 downto 0);
      C_out : out STD_LOGIC;
      Overflow : out STD_LOGIC
   );
End Component;

   signal B_new, S_out : STD_LOGIC_VECTOR (3
downto 0);
   signal C_out : STD_LOGIC;

begin
   B_new(0) <= Control XOR B(0);
   B_new(1) <= Control XOR B(1);
   B_new(2) <= Control XOR B(2);
   B_new(3) <= Control XOR B(3);
```

```
   RCA : RCA_4
      Port Map(
         A => A,
         B => B_new,
         C_in => Control,
         S => S_out,
         C_out => C_out,
         Overflow => Overflow
      );

   Zero <= NOT((C_out) OR S_out(0) OR S_out(1) OR
S_out(2) OR S_out(3));
   S <= S_out;

end Behavioral;
```

3 BIT ADDER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Adder_3 is
   Port (
      A : in STD_LOGIC_VECTOR (2 downto 0);
      S : out STD_LOGIC_VECTOR (2 downto 0)
   );
end Adder_3;
architecture Behavioral of Adder_3 is
Component FA
   Port (
      A, B, C_in : in std_logic;
      S, C_out : out std_logic
   );
End Component;
Signal FA0_C, FA1_C, FA2_C : std_logic;
 begin
    FA_0 : FA
      Port Map(
         A => A(0),
         B => '0',
         C_in => '1',
         S => S(0),
         C_out => FA0_C
      );

   FA_1 : FA
      Port Map(
         A => A(1),
         B => '0',
         C  in => FA0  C,
```

```
        S => S(1),
        C_out => FA1_C
    );

  FA_2 : FA
    Port Map(
        A => A(2),
        B => '0',
        C_in => FA1_C,
        S => S(2),
        C_out => FA2_C
    );

end Behavioral;
```

PROGRAM ROM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
entity Program_Rom is
   Port ( memory_select : in STD_LOGIC_VECTOR (2 downto 0);
          instruction_bus : 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 program_rom : rom_type := (

"100010000001",    --10 001 000 0001   MOVI R1,1
"100100000010",    --10 010 000 0010   MOVI R2,2
"101110000011",    --10 111 000 0011   MOVI R7,3
"001110100000",    --00 111 010 0000   ADD R7,R2
"001110010000",    --00 111 001 0000   ADD R7,R1
"110000000111",    --11 000 0000 111   JZR R0,7
"000000000000",    --00 000 000 0000   ADD R0,R0
"110000000101"     --11 000 0000 101   JZR R0,5

);
 begin

instruction_bus <= program_rom(to_integer(unsigned(memory_select)));
end Behavioral;
```

LUT_16 7 SEG

```
library IEEE;
use IEEE.STD_LOGIC_1164.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
 "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;
```

REGISTER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Reg is
    Port ( En : in STD_LOGIC;
           Clk : in STD_LOGIC;
           D : in STD_LOGIC;
           Res : in STD_LOGIC;
           Q : out STD_LOGIC := '0');
end Reg;

architecture Behavioral of Reg is
begin

    process (Clk) begin
    if (Res = '1') then
        Q <= '0';
    end if;
    if (En = '1') then
        if (rising_edge(Clk)) then
            Q <= D;
        end if;
    end if;
    end process;

end Behavioral;
```

REG_4

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Reg_4 is
   Port ( En : in STD_LOGIC;
          Clk : in STD_LOGIC;
          D : in STD_LOGIC_VECTOR (3 downto 0);
          Res : in STD_LOGIC;
          Q : out STD_LOGIC_VECTOR (3 downto 0):=
"0000");
end Reg_4;

architecture Behavioral of Reg_4 is
Component Reg
   Port ( En : in STD_LOGIC;
          Clk : in STD_LOGIC;
          D : in STD_LOGIC;
          Res : in STD_LOGIC;
          Q : out STD_LOGIC := '0');
End Component;
begin

   Reg_0 : Reg
      Port Map(
         En => En,
         Clk => Clk,
         D => D(0),
         Res => Res,
         Q => Q(0)
      );

   Reg_1 : Reg
      Port Map(
         En => En,
         Clk => Clk,
         D => D(1),
         Res => Res,
         Q => Q(1)
      );

   Reg_2 : Reg
      Port Map(
         En => En,
         Clk => Clk,
         D => D(2),
         Res => Res,
         Q => Q(2)
      );

   Reg_3 : Reg
      Port Map(
         En => En,
         Clk => Clk,
         D => D(3),
         Res => Res,
         Q => Q(3)
      );

end Behavioral;
```

DECODER 3_8

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Decoder_3_to_8 is
   Port ( I : in STD_LOGIC_VECTOR (2 downto 0);
          Y0 : out STD_LOGIC;
          Y1 : out STD_LOGIC;
          Y2 : out STD_LOGIC;
          Y3 : out STD_LOGIC;
          Y4 : out STD_LOGIC;
          Y5 : out STD_LOGIC;
          Y6 : out STD_LOGIC;
          Y7 : out STD_LOGIC);
end Decoder_3_to_8;

architecture Behavioral of Decoder_3_to_8 is

Signal I0P, I1P, I2P : STD_LOGIC;
Signal A0, A1, A2, A3 : STD_LOGIC;

begin

I0P <= NOT(I(0));
I1P <= NOT(I(1));
I2P <= NOT(I(2));

A0 <= I1P AND I2P;
A1 <= I(1) AND I2P;
A2 <= I1P AND I(2);
A3 <= I(1) AND I(2);

Y0 <= A0 AND I0P;
Y1 <= A0 AND I(0);
Y2 <= A1 AND I0P;
Y3 <= A1 AND I(0);
Y4 <= A2 AND I0P;
Y5 <= A2 AND I(0);
Y6 <= A3 AND I0P;
Y7 <= A3 AND I(0);

end Behavioral;
```

NANO PROCESSOR

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Nano_Processor is
   Port ( Clk : in STD_LOGIC;
          Reset : in STD_LOGIC;
          Overflow : out STD_LOGIC;
          Zero : out STD_LOGIC;
          Answer: out STD_LOGIC_VECTOR(3 downto 0));
end Nano_Processor;
architecture Behavioral of Nano_Processor is
component Instruction_Decoder
   Port ( Ins : in STD_LOGIC_VECTOR (11 downto 0);
          RegJmp : in STD_LOGIC_VECTOR (3 downto 0);
          RegEn : out STD_LOGIC_VECTOR (2 downto 0);
          LoadSel : out STD_LOGIC;
          ImVal : out STD_LOGIC_VECTOR (3 downto 0);
          RegSel1 : out STD_LOGIC_VECTOR (2 downto
0);
          RegSel2 : out STD_LOGIC_VECTOR (2 downto
0);
          AddSubSel : out STD_LOGIC;
          Jmp : out STD_LOGIC;
          AddressJmp : out STD_LOGIC_VECTOR (2
downto 0));
end component;
component Reg_Bank
   Port ( En : in STD_LOGIC_VECTOR (2 downto 0);
          Clk : in STD_LOGIC;
          Res : in STD_LOGIC;
          D : in STD_LOGIC_VECTOR (3 downto 0);
          Q0 : out STD_LOGIC_VECTOR (3 downto 0);
          Q1 : out STD_LOGIC_VECTOR (3 downto 0);
          Q2 : out STD_LOGIC_VECTOR (3 downto 0);
          Q3 : out STD_LOGIC_VECTOR (3 downto 0);
          Q4 : out STD_LOGIC_VECTOR (3 downto 0);
          Q5 : out STD_LOGIC_VECTOR (3 downto 0);
          Q6 : out STD_LOGIC_VECTOR (3 downto 0);
          Q7 : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component Mux_4_8_to_1
   Port ( I0 : in STD_LOGIC_VECTOR (3 downto 0);
          I1 : in STD_LOGIC_VECTOR (3 downto 0);
          I2 : in STD_LOGIC_VECTOR (3 downto 0);
          I3 : in STD_LOGIC_VECTOR (3 downto 0);
          I4 : in STD_LOGIC_VECTOR (3 downto 0);
          I5 : in STD_LOGIC_VECTOR (3 downto 0);
          I6 : in STD_LOGIC_VECTOR (3 downto 0);
          I7 : in STD_LOGIC_VECTOR (3 downto 0);
          S : in STD_LOGIC_VECTOR (2 downto 0);
          Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component Mux_3_2_to_1
   Port ( I0 : in STD_LOGIC_VECTOR (2 DOWNTO 0);
          I1 : in STD_LOGIC_VECTOR  (2 DOWNTO 0);
          S : in STD_LOGIC;
          Y : out STD_LOGIC_VECTOR (2 DOWNTO 0));
end component;

component Mux_4_2_to_1
   Port ( I0 : in STD_LOGIC_VECTOR (3 DOWNTO 0);
          I1 : in STD_LOGIC_VECTOR  (3 DOWNTO 0);
          S : in STD_LOGIC;
```

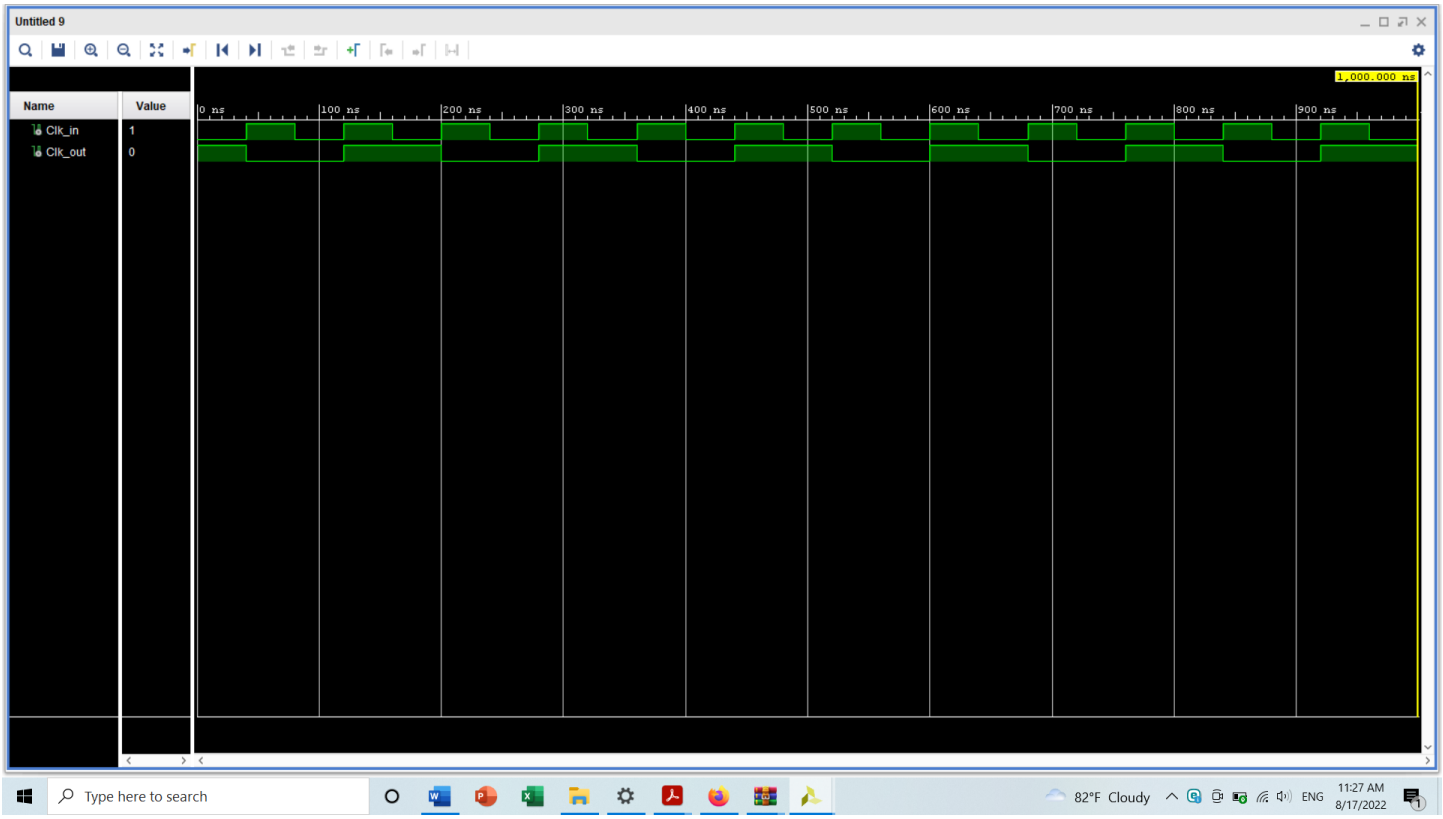
```
        Y : out STD_LOGIC_VECTOR (3 DOWNTO 0));
end component;

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

component Add_Sub_4
    Port ( Control : in STD_LOGIC;
        A : in STD_LOGIC_VECTOR (3 downto 0);
        B : in STD_LOGIC_VECTOR (3 downto 0);
        S : out STD_LOGIC_VECTOR (3 downto 0);
        Overflow : out STD_LOGIC;
        Zero : out STD_LOGIC);
end component;

component Adder_3
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
        S : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component Program_Rom
    Port ( memory_select : in STD_LOGIC_VECTOR (2
downto 0);
        instruction_bus : out STD_LOGIC_VECTOR (11
downto 0));
end component;

component Slow_Clk
    Port ( Clk  in : in STD  LOGIC;
        Clk_out : out STD_LOGIC);
end component;

signal I : STD_LOGIC_VECTOR (11 downto 0);
signal M, mux1_out, mux2_out, add_sub_out,
immediate_value, D0,D1, D2,D3,D4,D5,D6,D7  :
STD_LOGIC_VECTOR (3 downto 0);
signal register_enable, register_select_1, register_select_2,
address_to_jump, adder_out : STD_LOGIC_VECTOR (2
downto 0);
signal clock_out, load_select, jump_flag, add_sub_select :
std_logic;
signal memory_select, pc_in : STD_LOGIC_VECTOR (2
downto 0):= "000";

begin

--    pc_in <= "000";
--    memory_select <= "000";
--    adder_out <= "000";

    slow_clock_0 : Slow_Clk
        port map ( Clk_in => Clk,
                Clk_out => clock_out);

    Instruction_Decoder_0 :Instruction_Decoder
        Port map ( Ins => I,
                RegJmp => mux1_out,
                RegEn => register_enable,
                LoadSel => load_select,
                ImVal => immediate_value,
                RegSel1 => register_select_1,
                RegSel2 => register_select_2,
```

```
        Jmp => jump_flag,
        AddSubSel => add_sub_select,
        AddressJmp => address_to_jump);

Reg_Bank_0 :Reg_Bank
  Port map ( En => register_enable,
        Clk => clock_out,
        Res => reset,
        D => M,
        Q0 => D0,
        Q1 => D1,
        Q2 => D2,
        Q3 => D3,
        Q4 => D4,
        Q5 => D5,
        Q6 => D6,
        Q7 => D7);

Mux_4_8_to_1_1 :Mux_4_8_to_1
  Port map ( I0 => D0,
        I1 => D1,
        I2 => D2,
        I3 => D3,
        I4 => D4,
        I5 => D5,
        I6 => D6,
        I7 => D7,
        S => register_select_1,
        Y => mux1_out);

Mux_4_8_to_1_2 :Mux_4_8_to_1
  Port map ( I0 => D0,
        I1 => D1,
        I2 => D2,
        I3 => D3,
        I4 => D4,
        I5 => D5,
        I6 => D6,
        I7 => D7,
        S => register_select_2,
        Y => mux2_out);

Mux_3_2_to_1_0 :Mux_3_2_to_1
  Port map ( I0 => adder_out,
        I1 => address_to_jump,
        S => jump_flag,
        Y => pc_in);

Mux_4_2_to_1_0 :Mux_4_2_to_1
  Port map ( I0 => add_sub_out,
        I1 => immediate_value,
        S => load_select,
        Y => M);

Program_Counter_0 :Program_Counter
  Port map ( D => pc_in,
        Clk => clock_out,
        Res => reset,
        Q => memory_select);

Add_Sub_4_0 :Add_Sub_4
  Port map ( Control => add_sub_select,
        A => mux2_out,
        B => mux1_out,
```

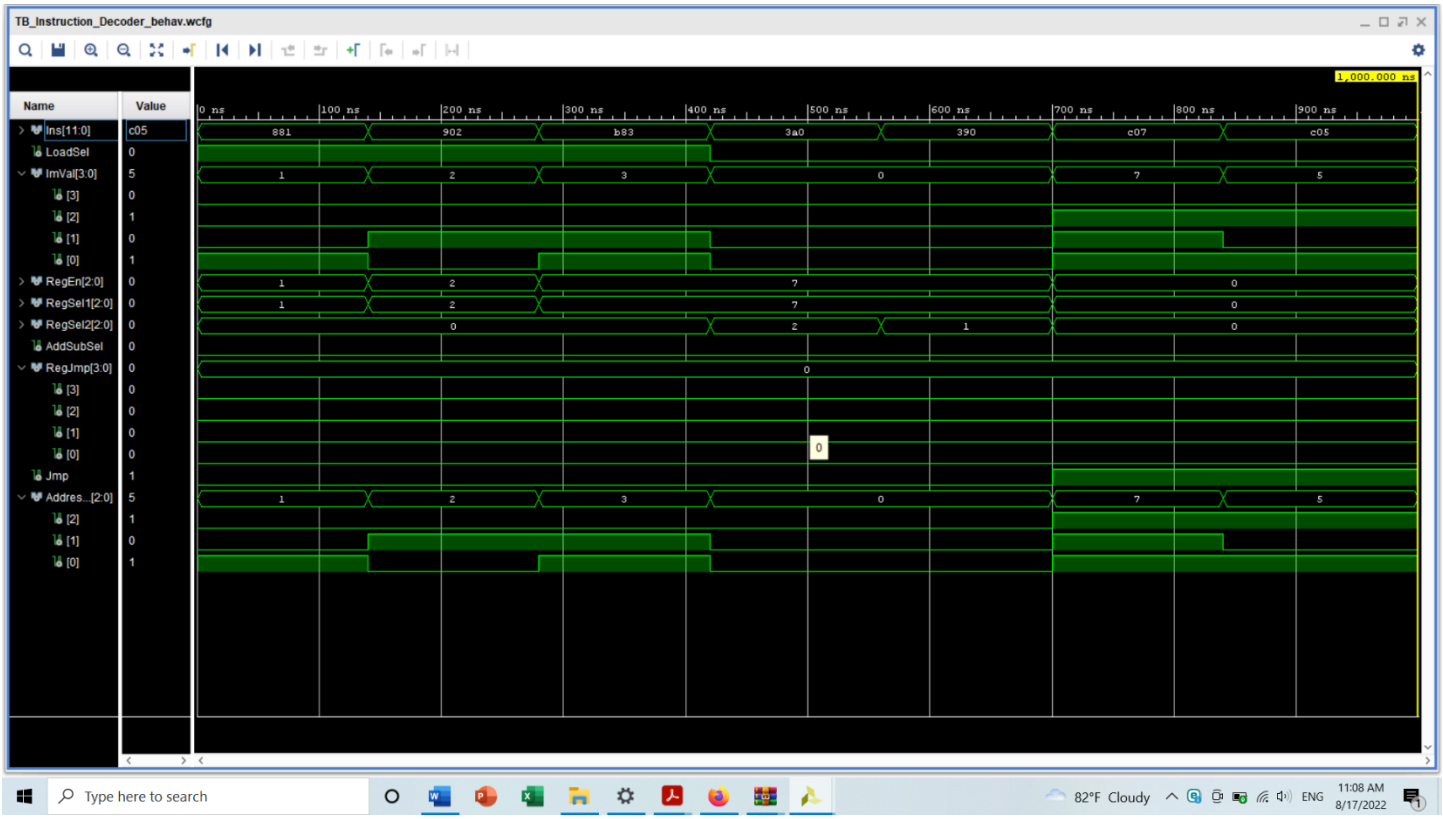
```
            S => add_sub_out,
            Overflow => Overflow,
            Zero => Zero);

    Adder_3_0 :Adder_3
        Port map ( A => memory_select,
            S => adder_out);

    ProgramROM_0 :Program_Rom
        Port map ( memory_select => memory_select,
            instruction_bus => I);

    Answer <= D7;

end Behavioral;
```

TIMING DIAGRAMS

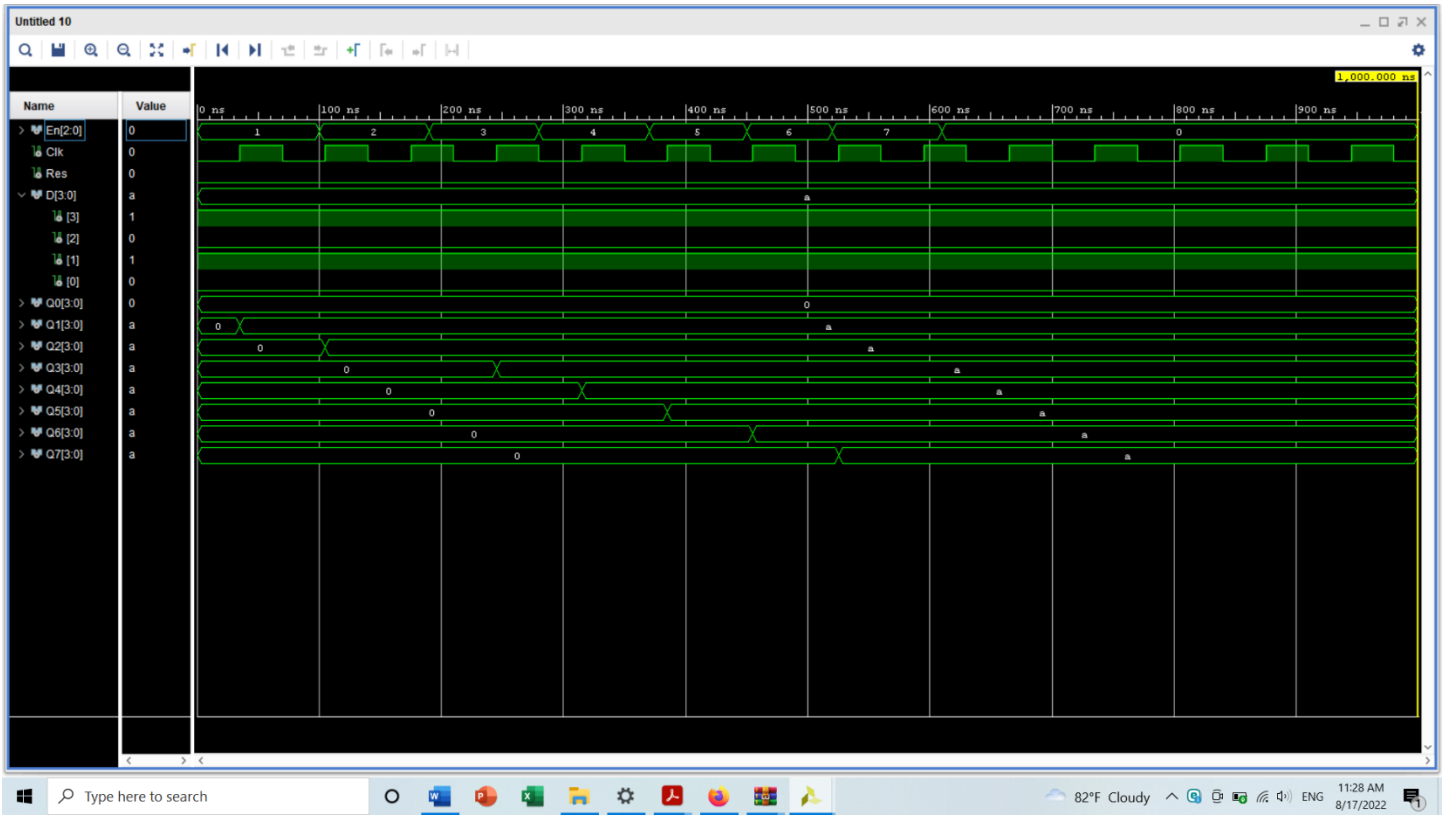
1. SLOW CLOCK



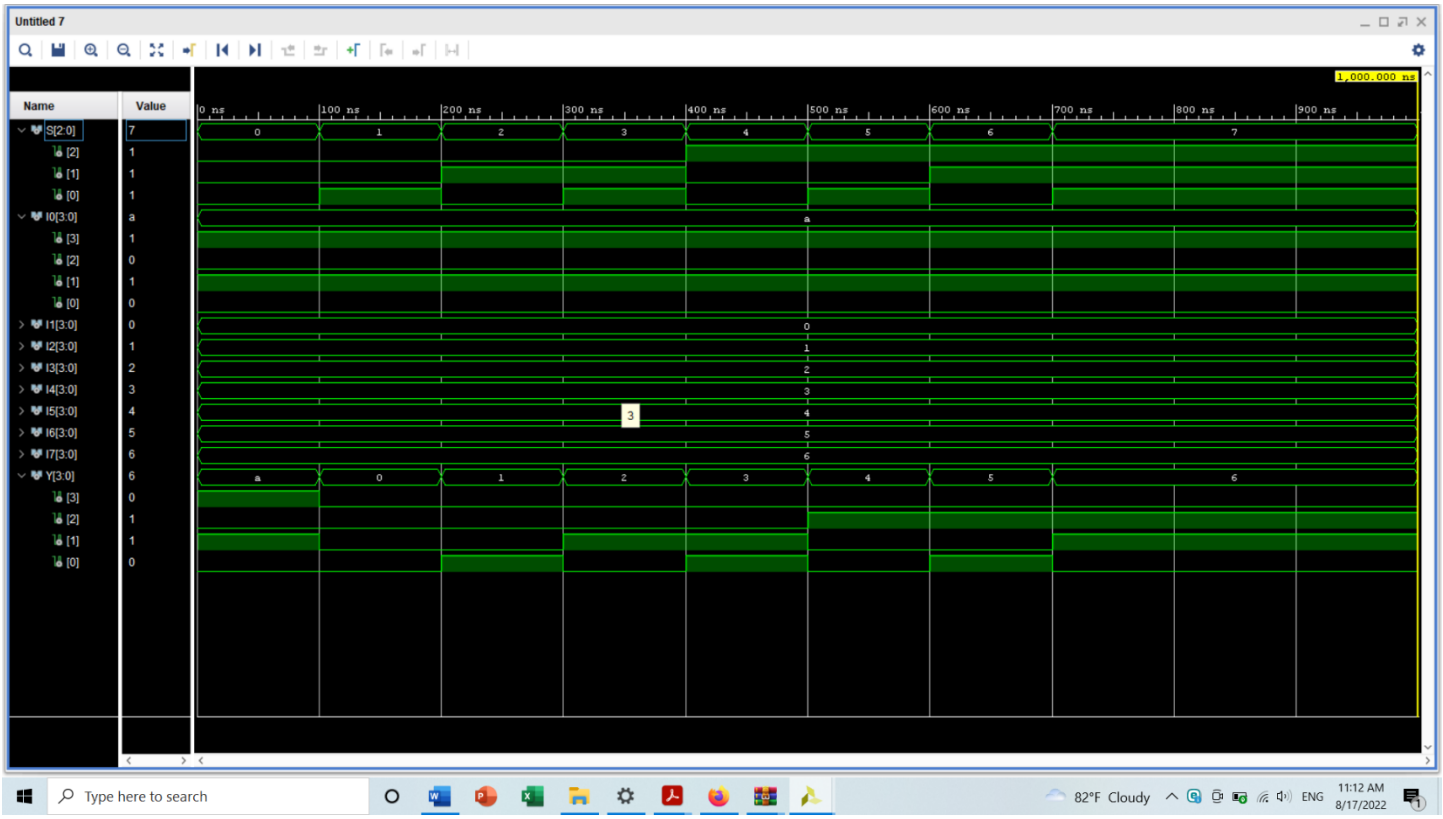
2. INSTRUCTION DECODER



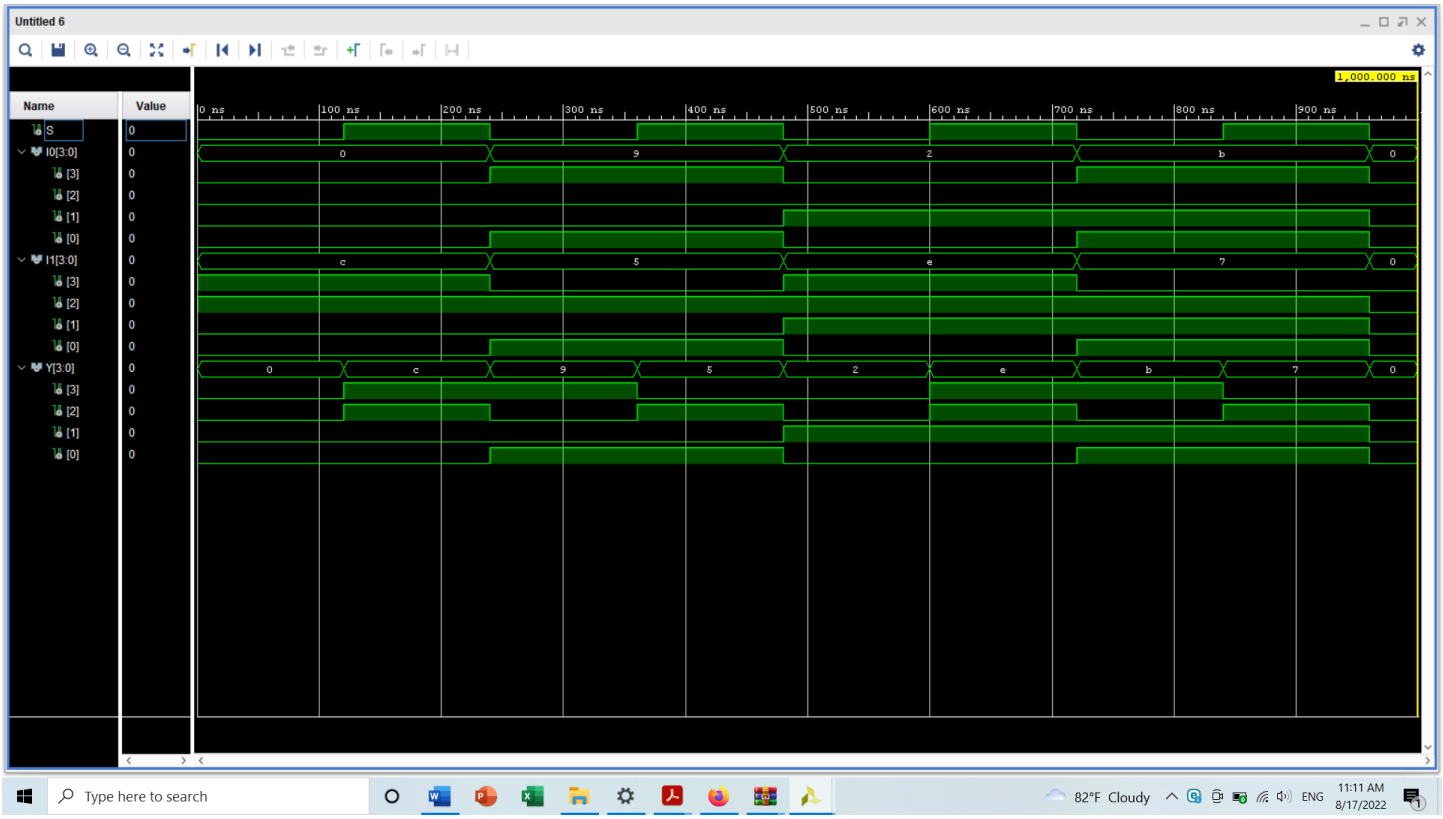
3. REGISTER BANK



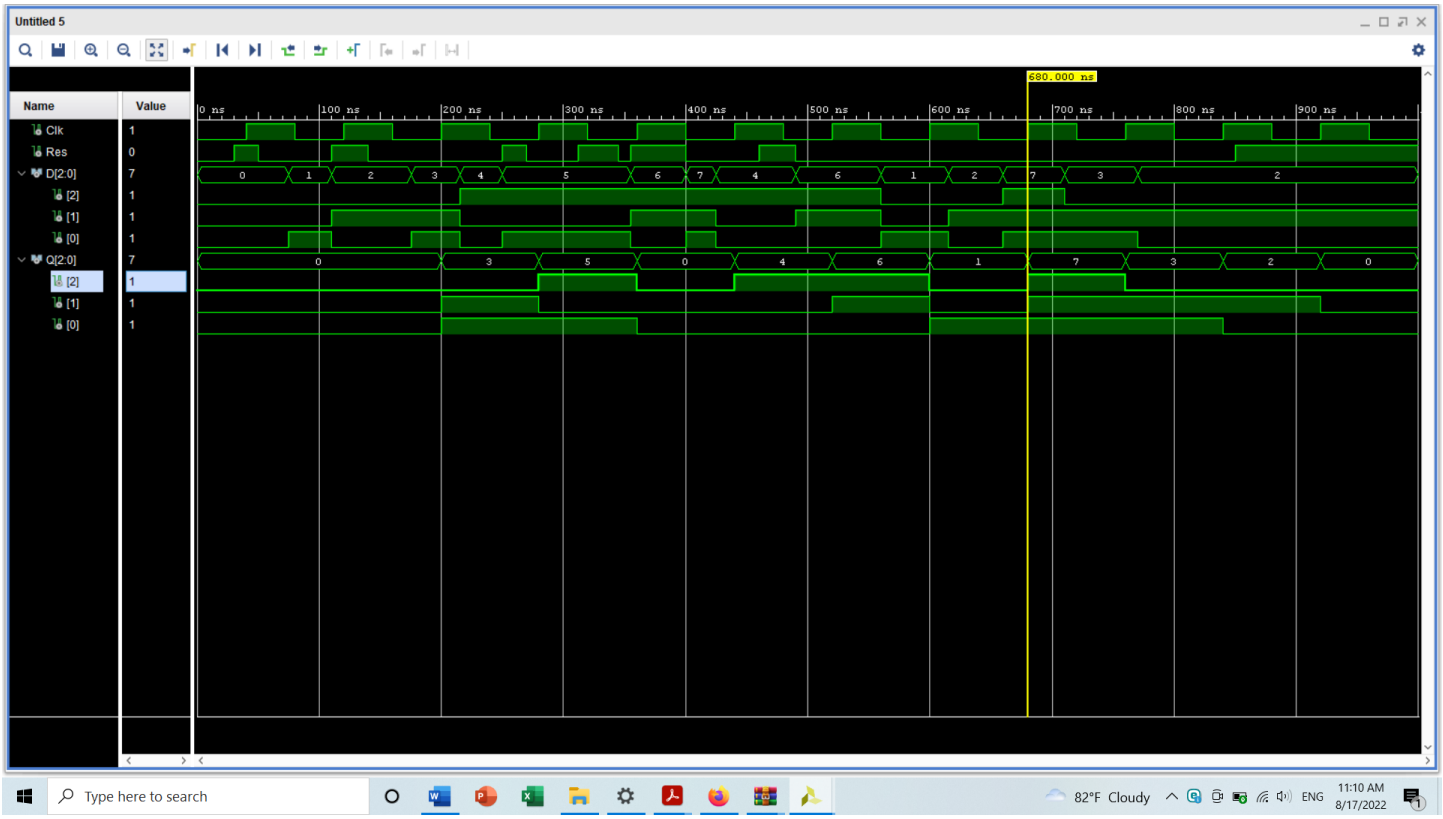
4. MUX 8_1



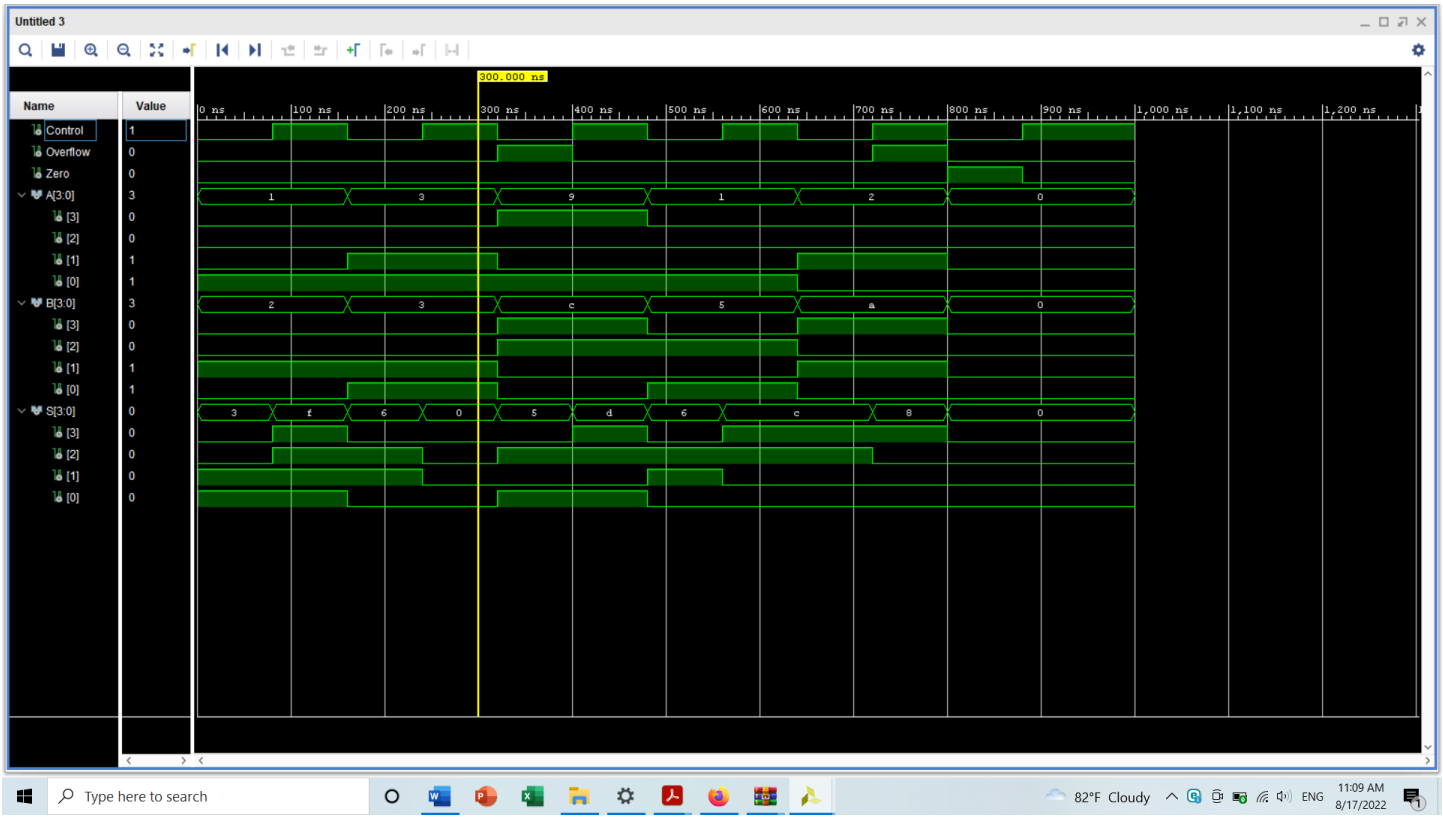
5. MUX 2_1



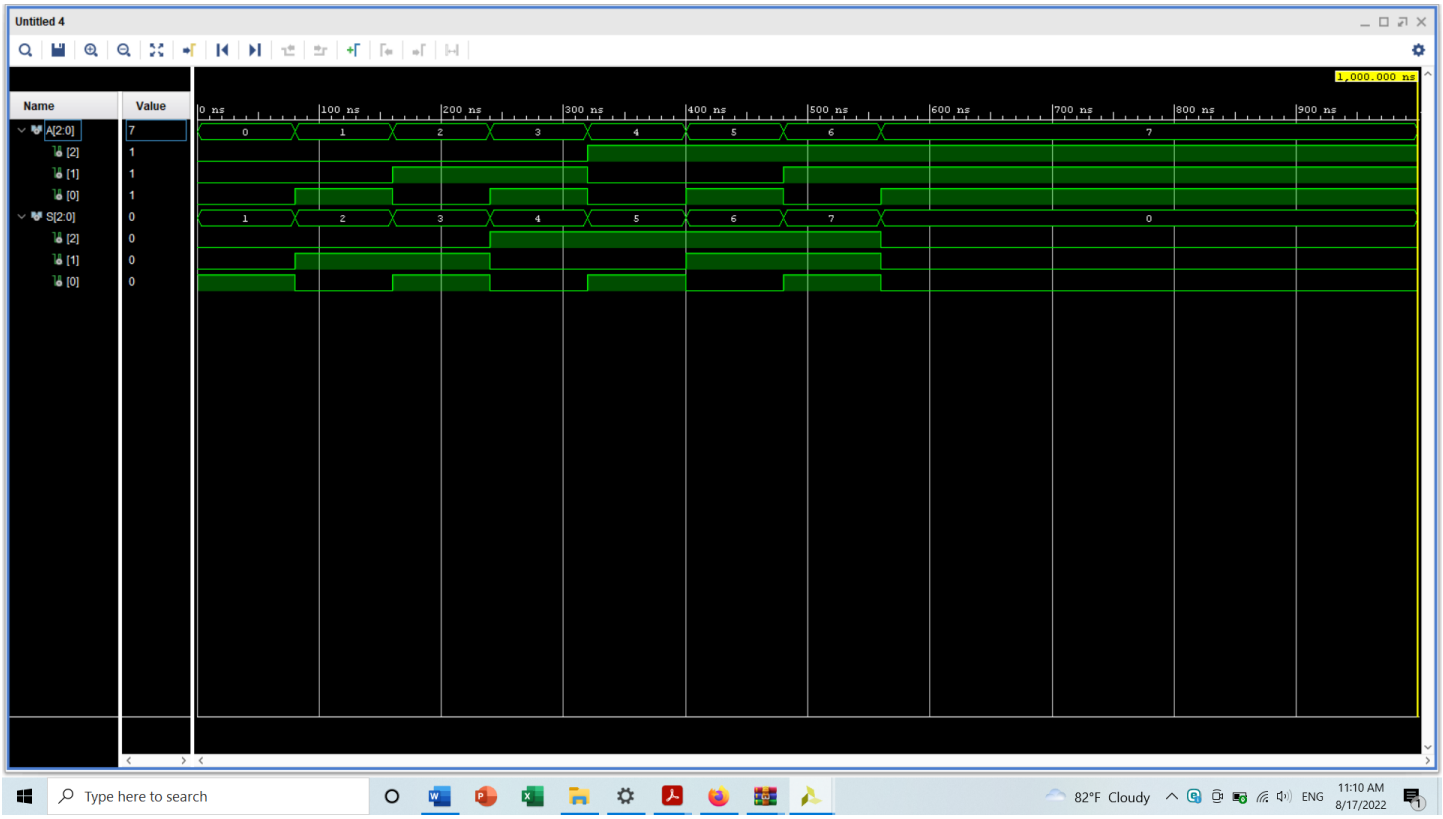
6. PROGRAM COUNTER



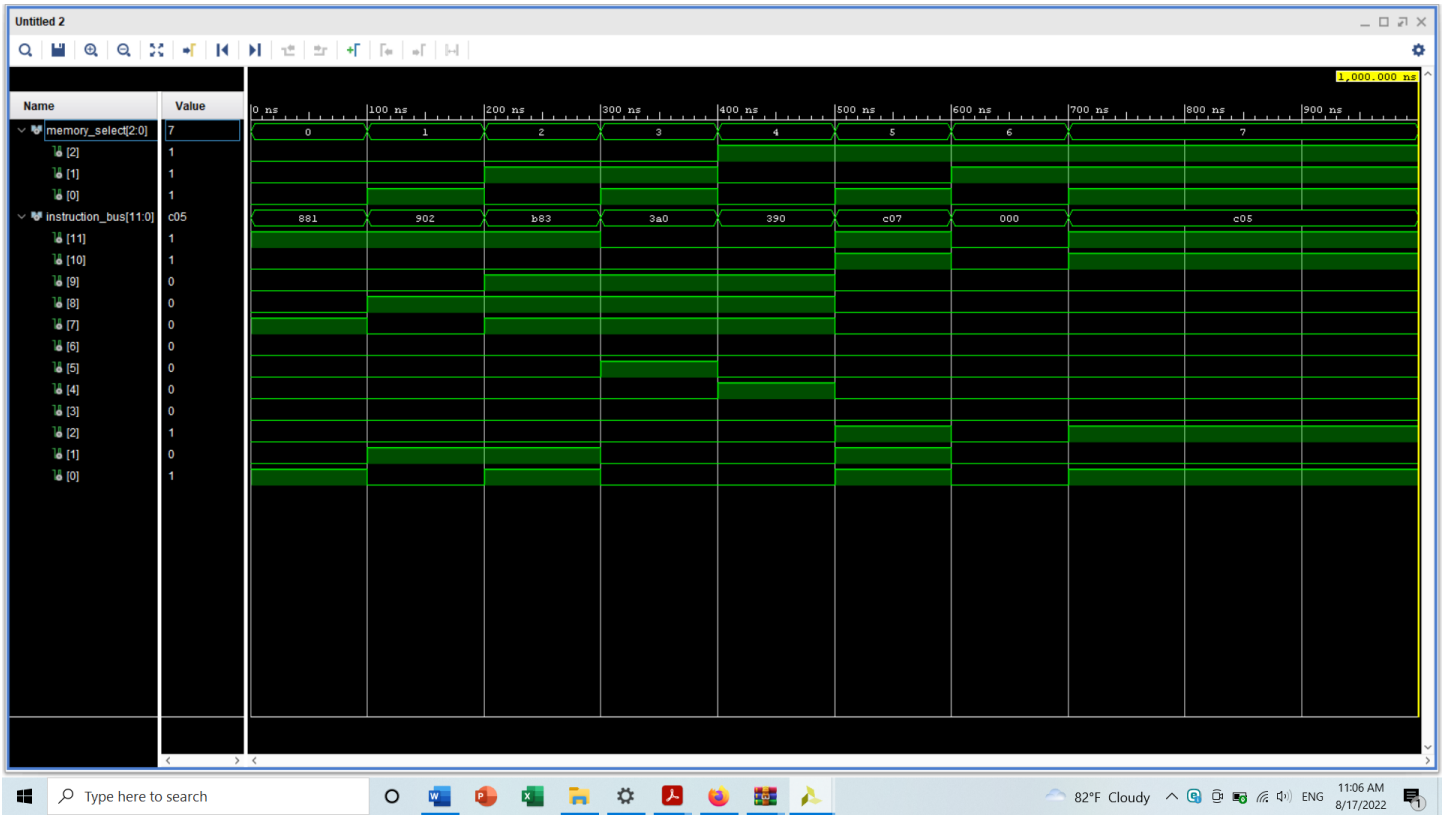
7. 4 BIT ADD AND SUB UNIT



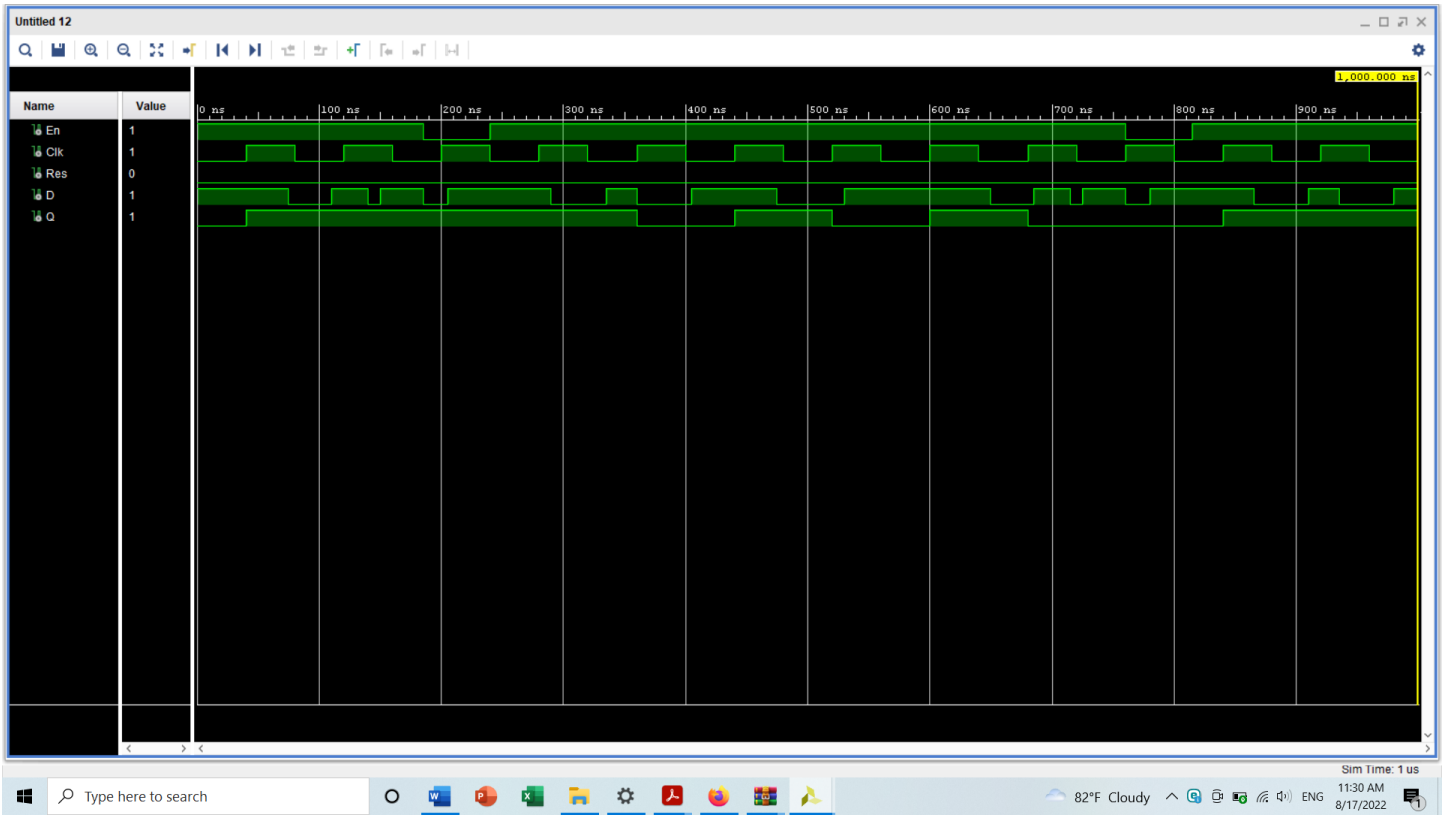
8. 3 BIT ADDER



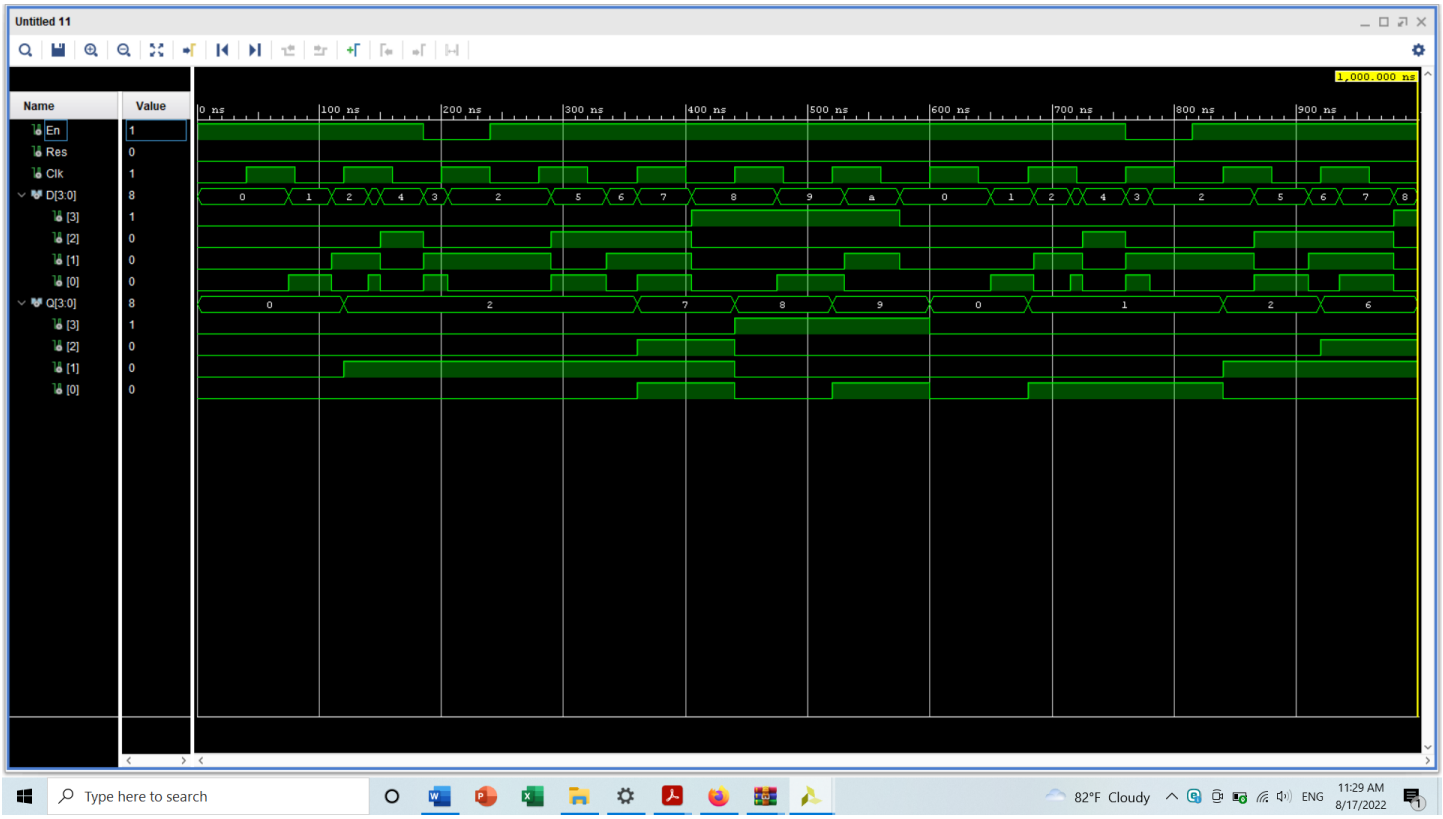
9. PROGRAM ROM



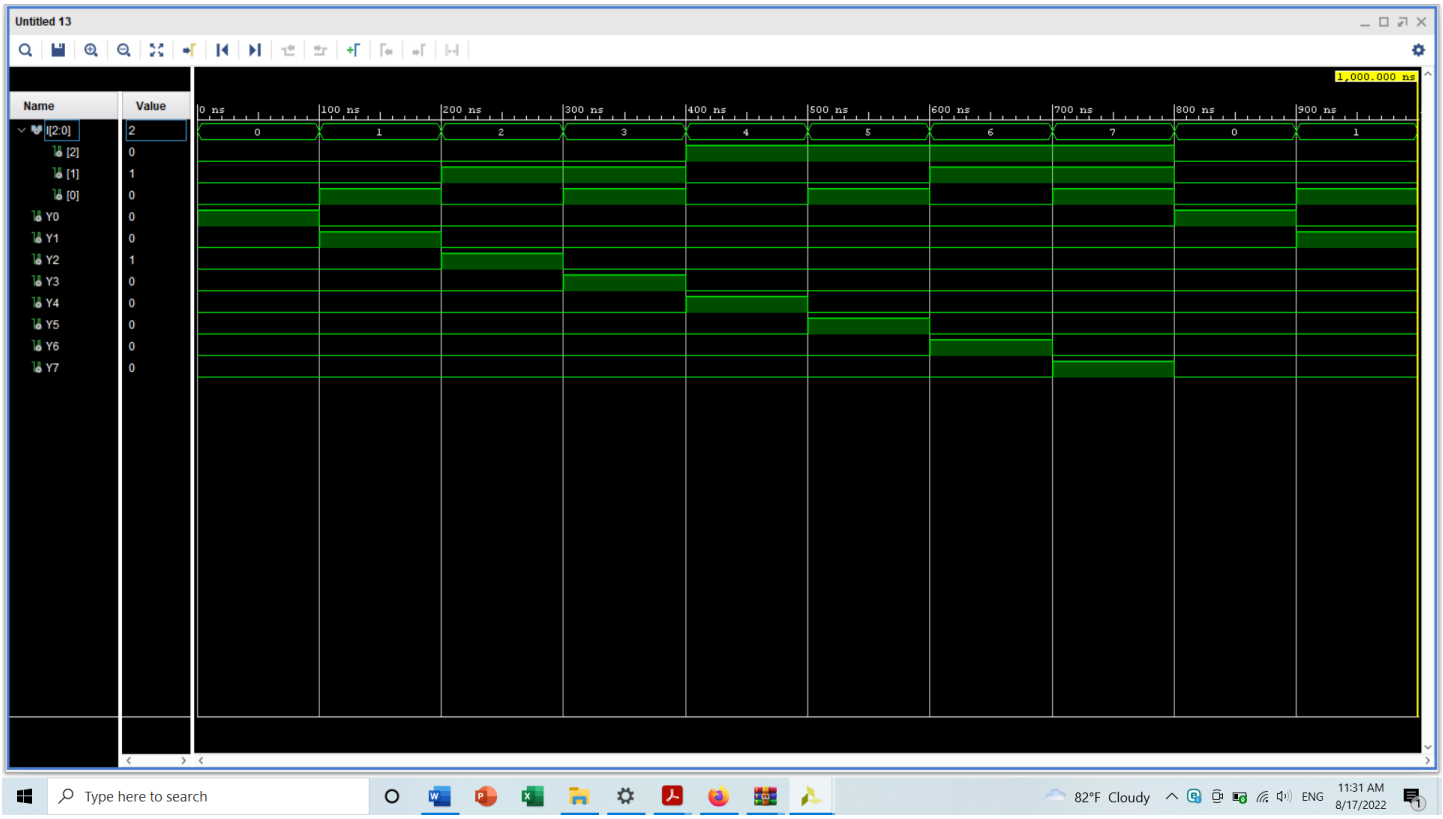
10. REGISTER



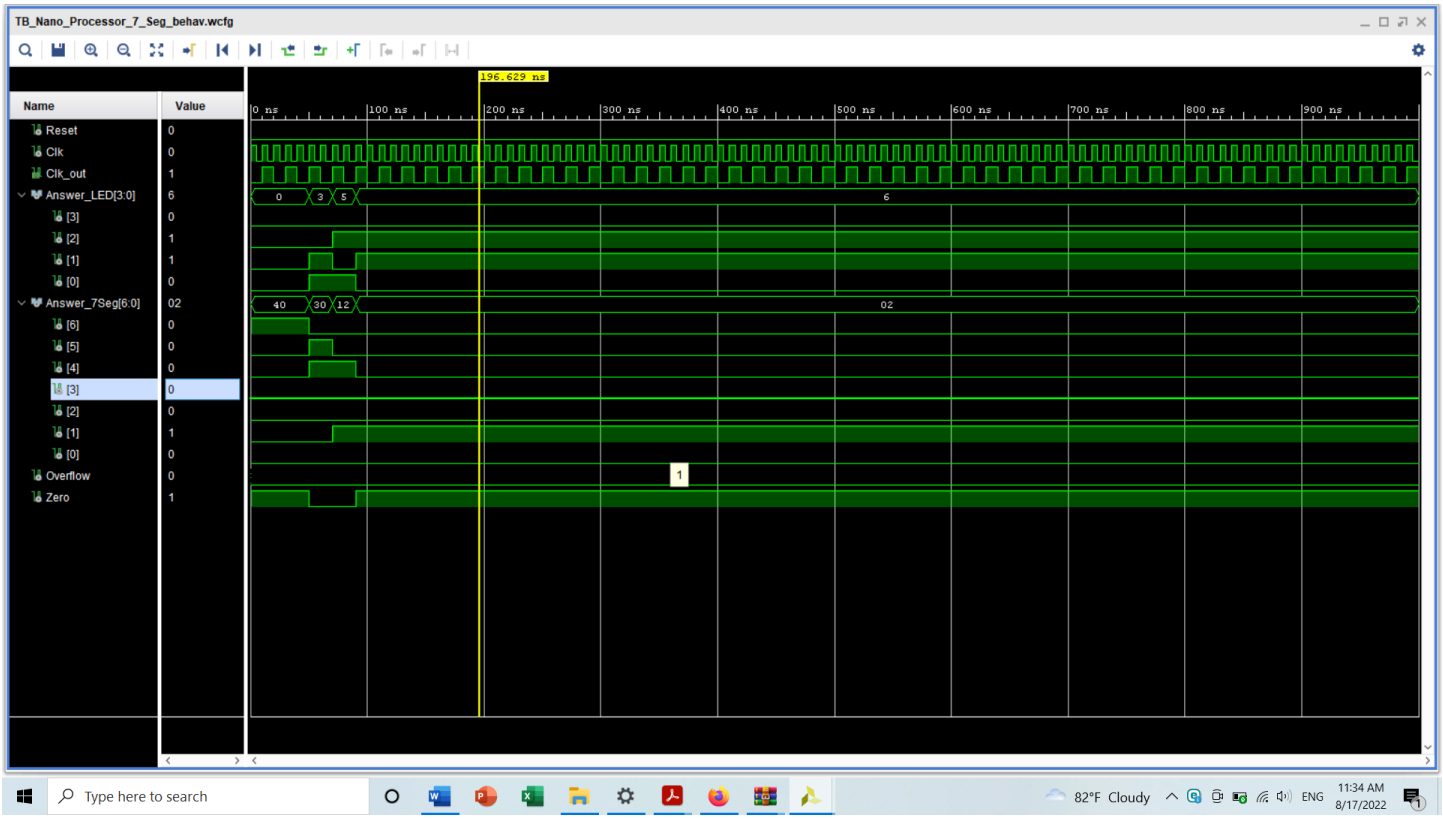
11. REG_4



12. DECODER 3_8



13. NANO PROCESSOR 7 SEG



CONCLUSIONS:

- As the nano processor only understands binary values, we need to hardcode assembly instructions as binary values.
- Using the 4-bit arithmetic unit we can do arithmetic operations with input signals of 4 bits.
- To run a nano processor a clock signal is essential.
- Usage of buses simplify the design.
- Slice logic and primitive tables in synthesis report can be used to observe the optimizations done to the nano processor.
- With the integration of previously created components and using them in hierarchical order made the development process less complicated and reusable.
- In a project like this, as the components are getting complicated, it is easier to distribute and delegate the workload.
- Program counter points to the next suitable instruction set.
- Register bank is used to store the 4-bit binary values in each register with help of d flipflops.
- Instruction decoder act as the main controlling hub in the nano processor which decode the binary instruction and send suitable input signals for suitable components.
- 4-bit Adder/Subtractor can add and subtract 2 4-bit binary inputs and give the output with overflow and zero signals.
- To select an input from a set of inputs, we mainly used different multiplexers.

OPTIMIZATIONS:

Instead of using previously built components, we built 3 to 8 decoders, multiplexers, and 4 bit adder subtractors from ground up that resulted in 52 slice LUTs and 53 slice registers as flip flops.

CONTRIBUTION FROM THE TEAM:

A.A.A. Jasmin **200238N**	• Instruction Decoder • Multiplexers • Nano processor assembly • Simulation of multiplexers, instruction decoder, nano processor
K.G.T.R. Kumaratunga **200327L**	• 4-bit adder subtractor • Simulation of 4-bit adder subtractor, register bank, 3-to-8 decoder
K.V.R. Kurukulasooriya **200332X**	• Program rom • Simulation of program rom • Documentation
G.S.D. Kuruppu **200333C**	• 3-bit program counter • 3-bit adder • Simulation of 3-bit program counter, 3-bit adder
G.M.P. Silva **200605M**	• Register bank • 3-to-8 decoder • Nano processor 7 segment integration