Faculty of Automation and Computer Science
Digital Systems Design Project
Darius-Eric Săsărman, group 30415, Year 2024-2025

UNIVERSITATEA TEHNICĂ CLUJ-NAPOCA

Signed 8-bit Pocket Calculator Project with fundamental operations

Faculty of Automation and Computer Science
Digital Systems Design Project
Darius-Eric Săsărman, group 30415, Year 2024-2025

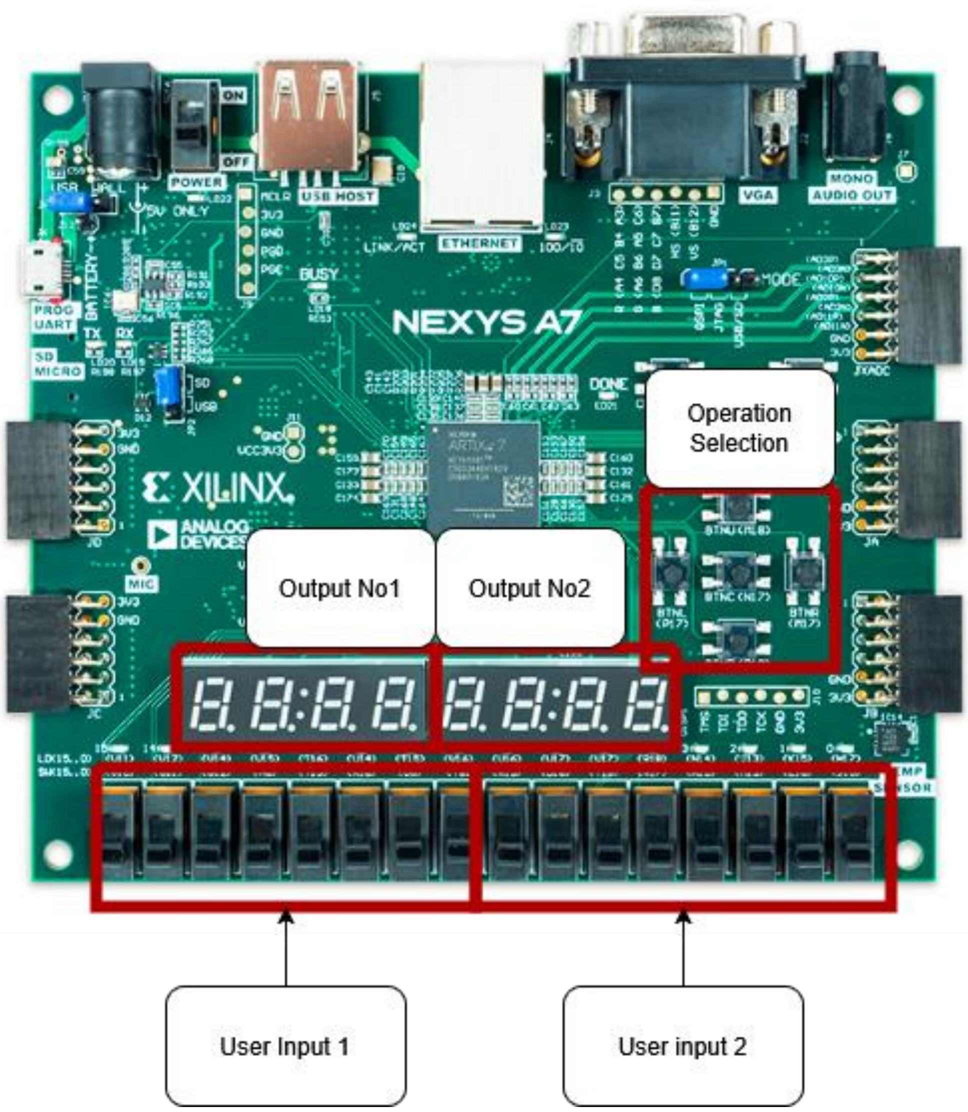
Table of Contents:

Faculty of Automation and Computer Science
Digital Systems Design Project
Darius-Eric Săsărman, group 30415, Year 2024-2025

Statement:

A10) Să se proiecteze un **calculator de buzunar cu operaţii aritmetice fundamentale** (adunare, scădere, înmulţire, împărţire). Operaţiile de înmulţire şi împărţire se vor implementa folosind algoritmi specifici, nu operatorii limbajului. Operanzii sunt reprezentaţi pe 8 biţi cu semn. Operanzii şi operatorii vor fi introduşi secvenţial în formă zecimală. Se vor folosi afişajele cu 7 segmente de pe plăcuţele cu FPGA. Proiectul va fi realizat de **1 student.**

Basic Functioning Procedure:



Considering each half of the switch array as an 8-bit signed number, the user can input two numbers in two's complement on eight bits. The input numbers, before an operation is selected, are shown on each half of the seven segment display.

The syntax is as follows :

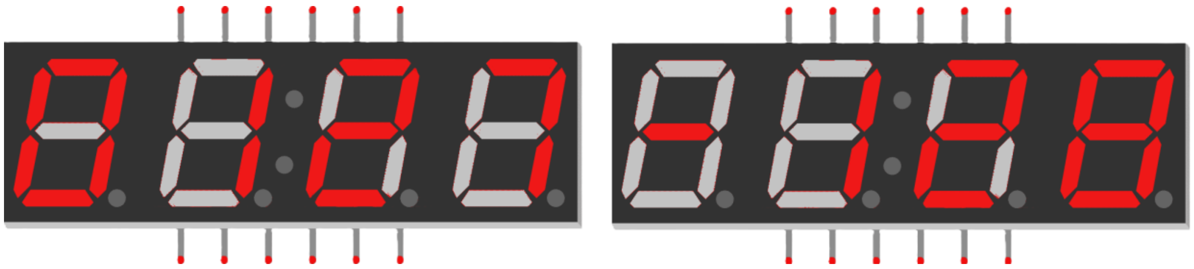
No1 OP No2 = Result

The buttons' meaning is as follows :

	Addition	
Multiplication	Reset	Division
	Subtraction	

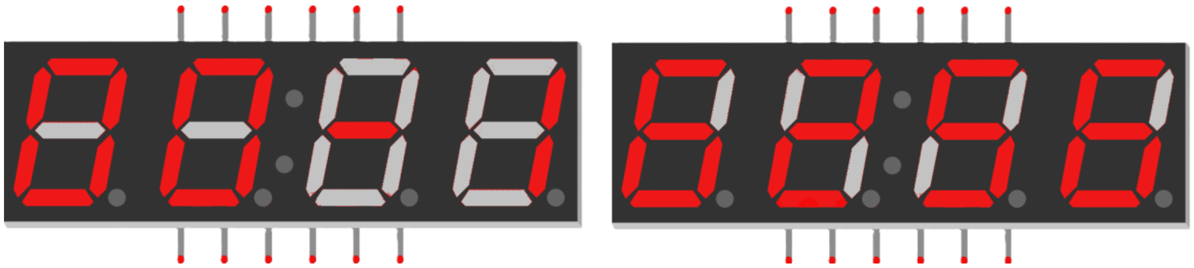
Top = Addition, Bottom = Subtraction, Left = Multiplication, Right = Subtraction, Middle = Reset (goes back to the two-number displaying and selecting).
After pressing any operation button, the calculator will display the selected operation's result on the seven-segment display, as a single number.

Example usage:
Say the user inputs these two numbers:



(intended max ranges showcase for 8-bit 2's complement representation)

And then decides to call the "multiply" operation. After the execution of the operation the seven-segment display will look like this:

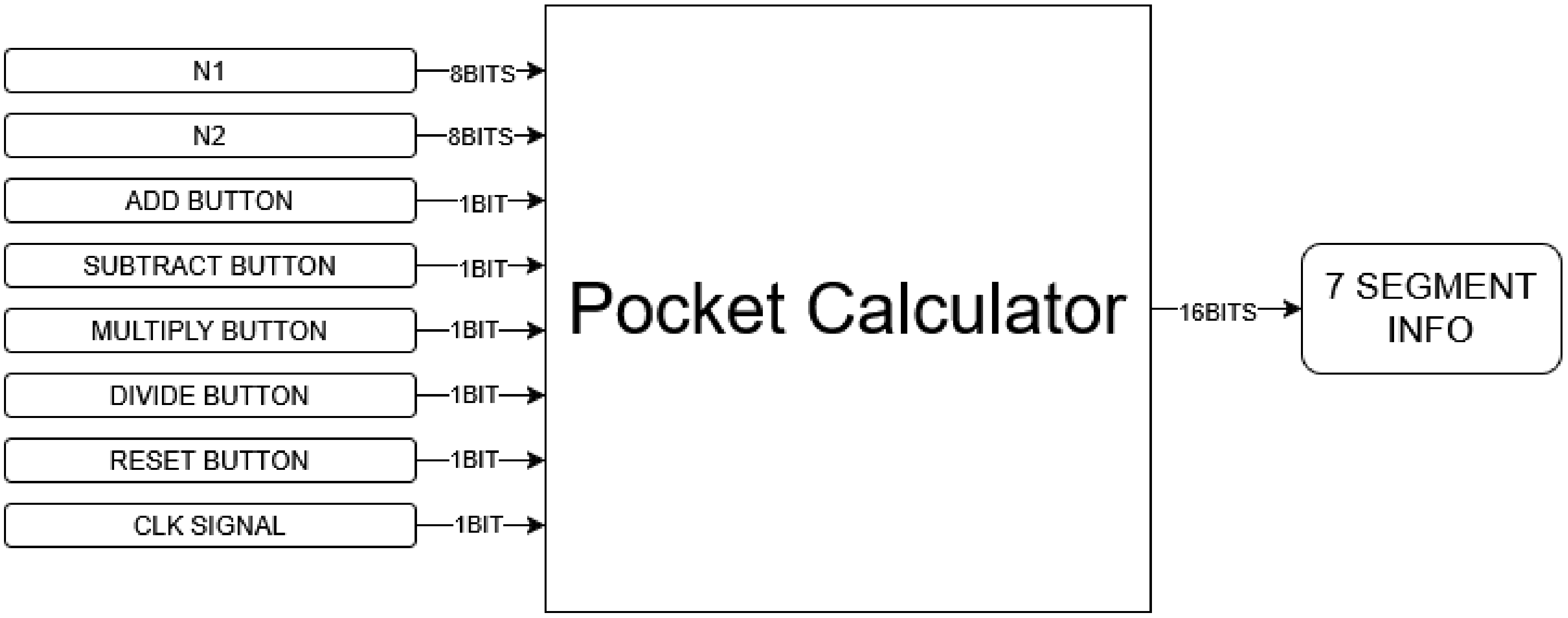


After this number is displayed, the calculator will await the 'reset' input from the user. After that selection, the calculator goes back to the number selecting scheme.

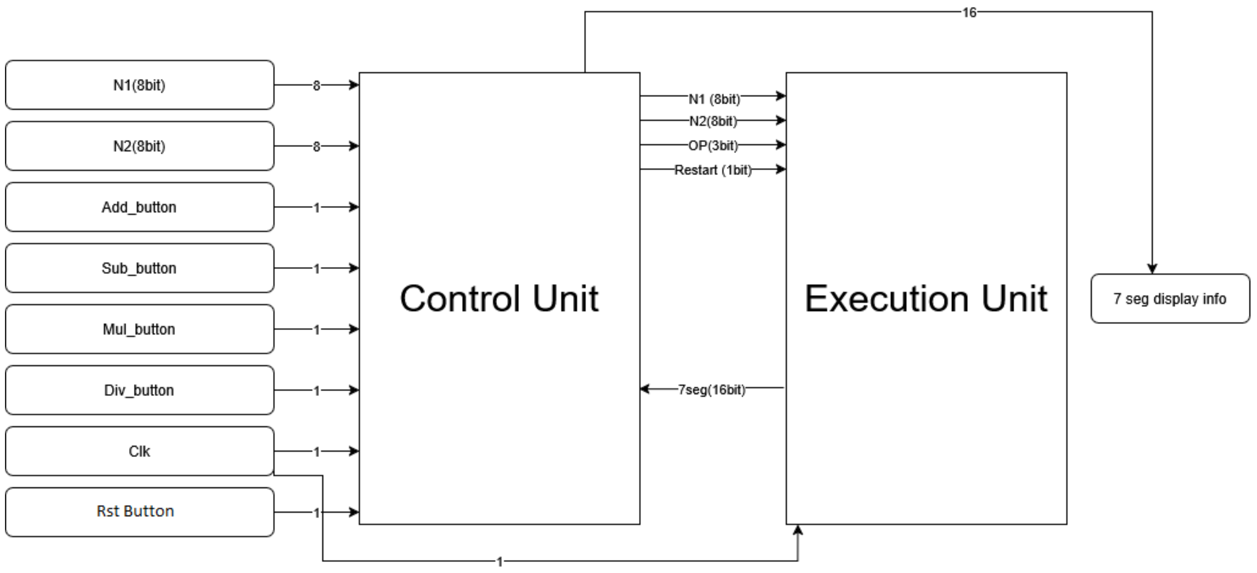
Faculty of Automation and Computer Science
Digital Systems Design Project
Darius-Eric Săsărman, group 30415, Year 2024-2025

UNIVERSITATEA TEHNICĂ CLUJ-NAPOCA

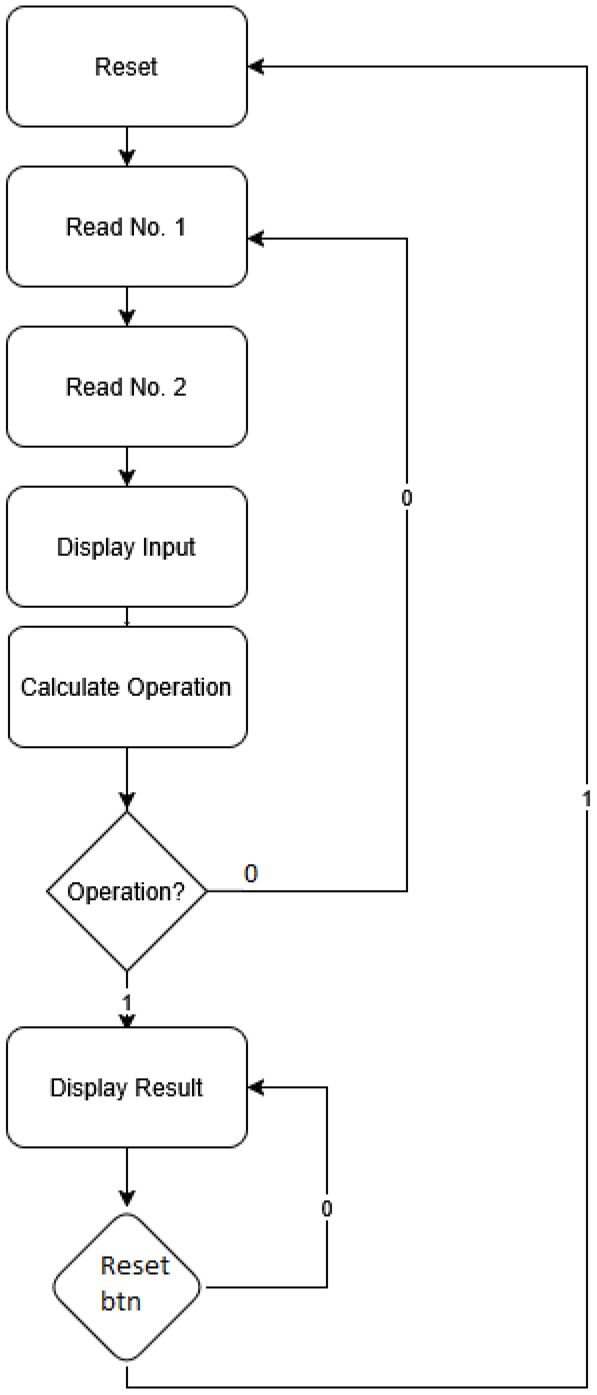
General block diagram:



Component block diagram:



State Diagram:



Execution Unit

In the following pages I will proceed to describe the following components:

1. Data register 8-bit
2. Arithmetic logic unit
    a. Mux 2 to 1 on 64bits
    b. Mux 4 to 1 on 64bits
    c. Mux 8 to 1 on 64bits
    d. Adder unit
        i. Full Adder 1-bit
        ii. Full Adder 8-bit
        iii. General structure
    e. Subtracter unit
        i. Full Subtracter 1-bit
        ii. Full Adder 8-bit
        iii. General structure
    f. Multiplier Unit
        i. State Diagram
    g. Division Unit
        i. State Diagram

1. Data register 8 bit

Purpose :

1. To secure no feedback loops are asynchronous (VHDL Bitstream Error)
2. To hold data for the other components in the Execution Unit

Inputs :

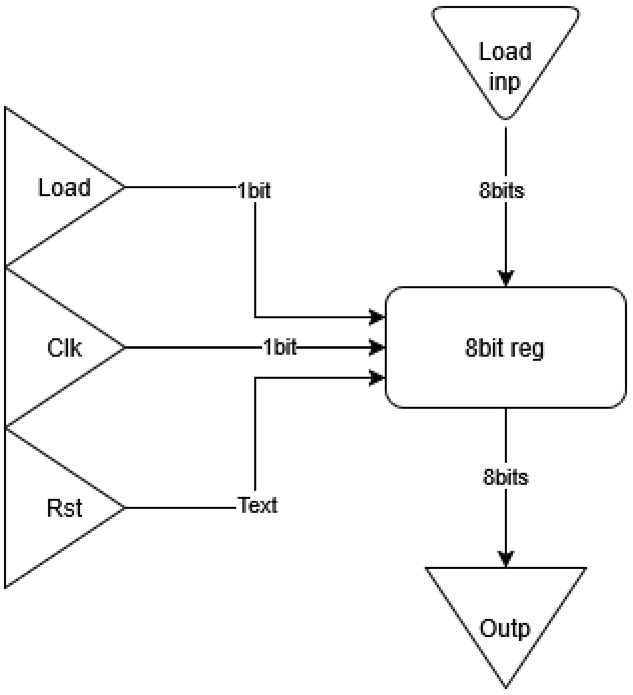
1. Load_inp : the 8 bits that are loaded onto the register
2. Load : the decision bit, active on '1'
3. Clk : Clock
4. Rst : clears the information stored

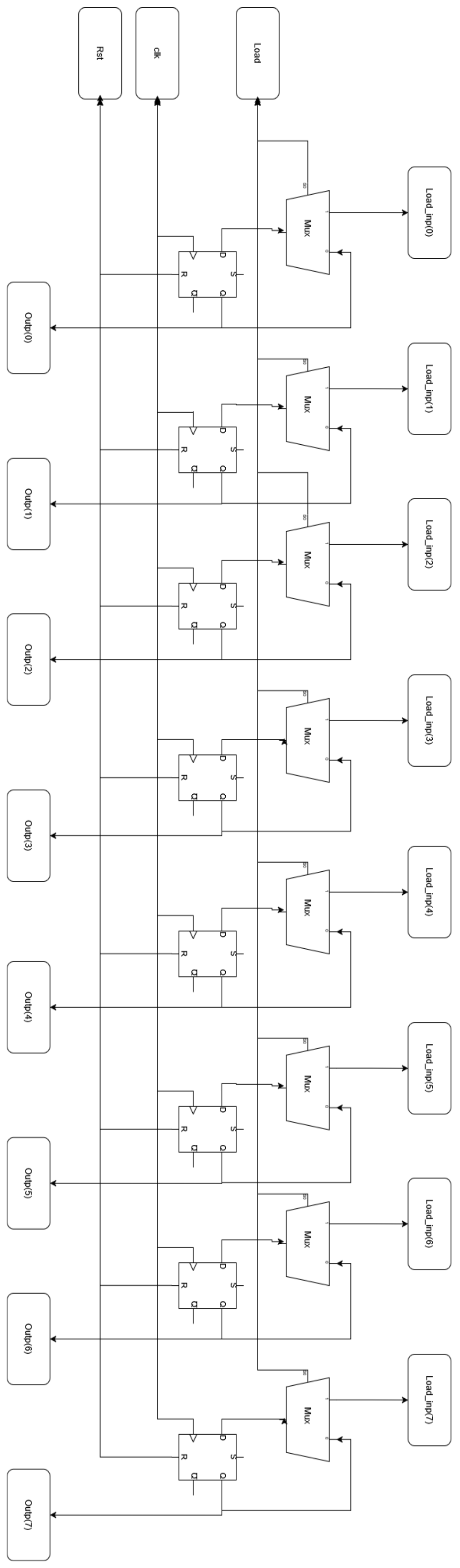
Outputs:

1. Outp: The information stored inside the register

Logic Scheme (on next page)

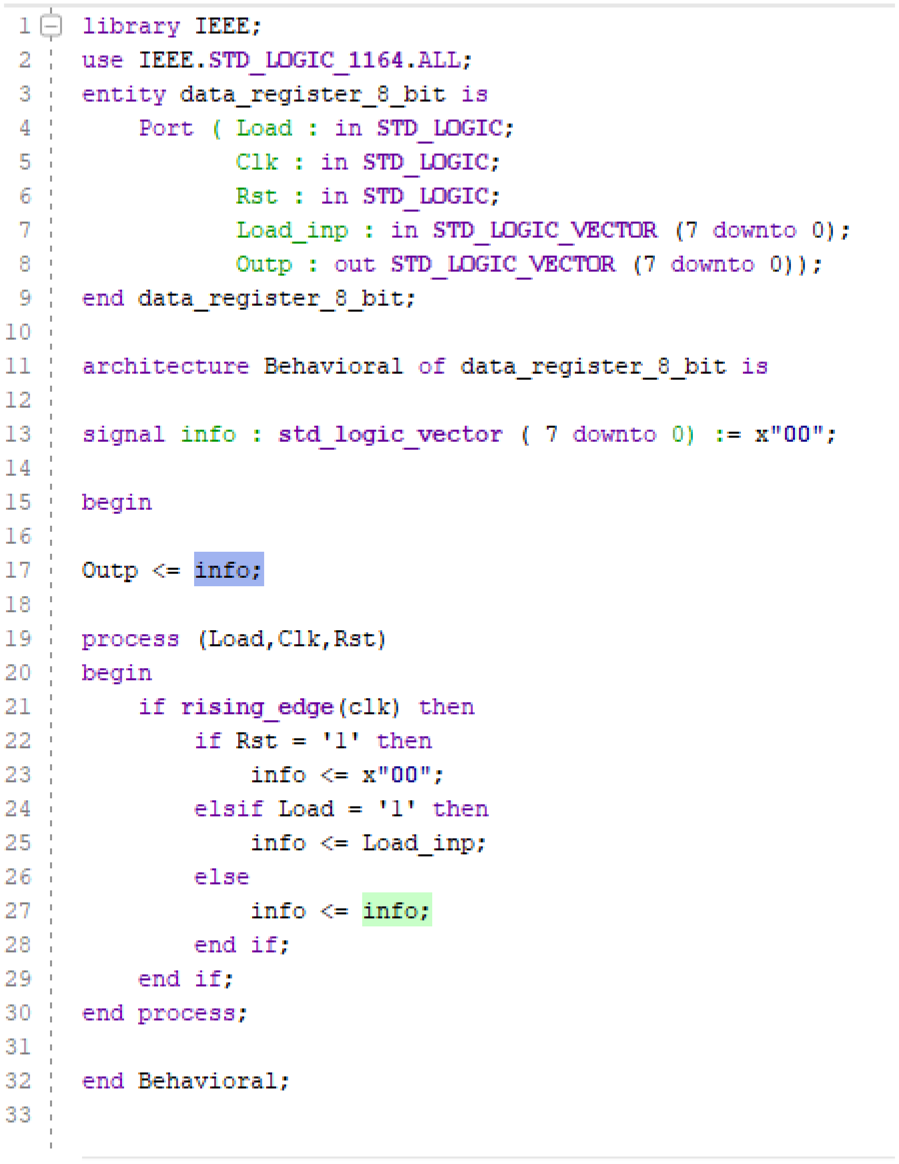
Black box:




Rst
clk
Load
Mux
D
S
R
Q
Load_inp(0)
Load_inp(1)
Load_inp(2)
Load_inp(3)
Load_inp(4)
Load_inp(5)
Load_inp(6)
Load_inp(7)
Outp(0)
Outp(1)
Outp(2)
Outp(3)
Outp(4)
Outp(5)
Outp(6)
Outp(7)

Faculty of Automation and Computer Science
Digital Systems Design Project
Darius-Eric Săsărman, group 30415, Year 2024-2025

Vhdl Code:



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity data_register_8_bit is
    Port ( Load : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Rst : in STD_LOGIC;
           Load_inp : in STD_LOGIC_VECTOR (7 downto 0);
           Outp : out STD_LOGIC_VECTOR (7 downto 0));
end data_register_8_bit;

architecture Behavioral of data_register_8_bit is

signal info : std_logic_vector ( 7 downto 0) := x"00";

begin

Outp <= info;

process (Load,Clk,Rst)
begin
    if rising_edge(clk) then
        if Rst = '1' then
            info <= x"00";
        elsif Load = '1' then
            info <= Load_inp;
        else
            info <= info;
        end if;
    end if;
end process;

end Behavioral;
```

2.Arithmetic Logic Unit

a. 2 to 1 Mux on 64 bits

Purpose : To pick between two 64bit options

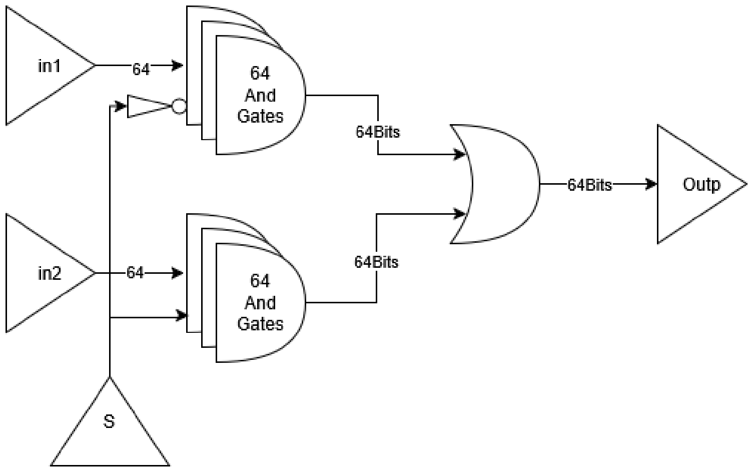
Inputs :

1. In1 : 64 bits, first option
2. In2 : 64 bits, second option
3. S : selection bit

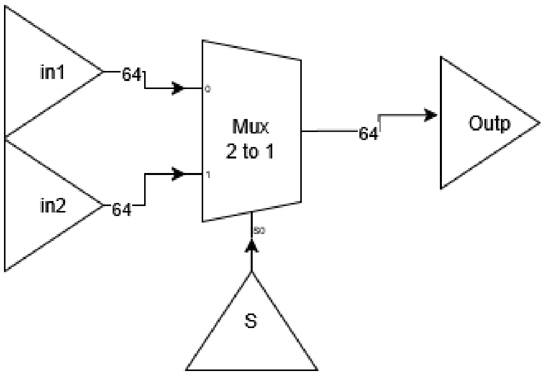
Outputs:

1. Outp : 64 bits, selected option

Logic scheme:

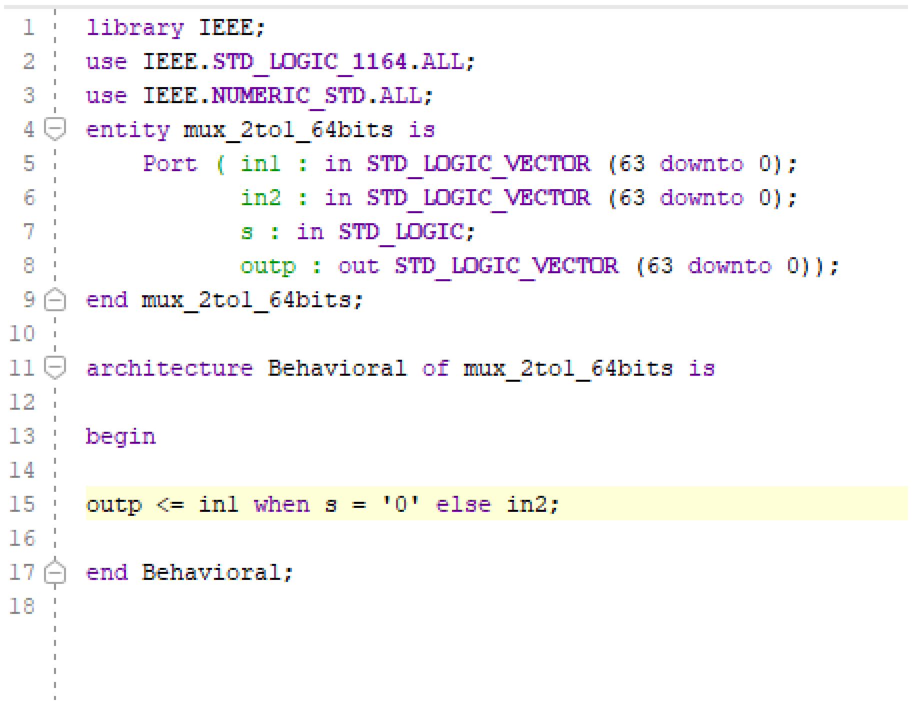


Black Box:



Faculty of Automation and Computer Science
Digital Systems Design Project
Darius-Eric Săsărman, group 30415, Year 2024-2025

VHDL Code:



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity mux_2to1_64bits is
    Port ( in1 : in STD_LOGIC_VECTOR (63 downto 0);
           in2 : in STD_LOGIC_VECTOR (63 downto 0);
           s : in STD_LOGIC;
           outp : out STD_LOGIC_VECTOR (63 downto 0));
end mux_2to1_64bits;

architecture Behavioral of mux_2to1_64bits is

begin

outp <= in1 when s = '0' else in2;

end Behavioral;
```

b.4 to 1 mux on 64 bits

Purpose: To pick between 4 options on 64 bits

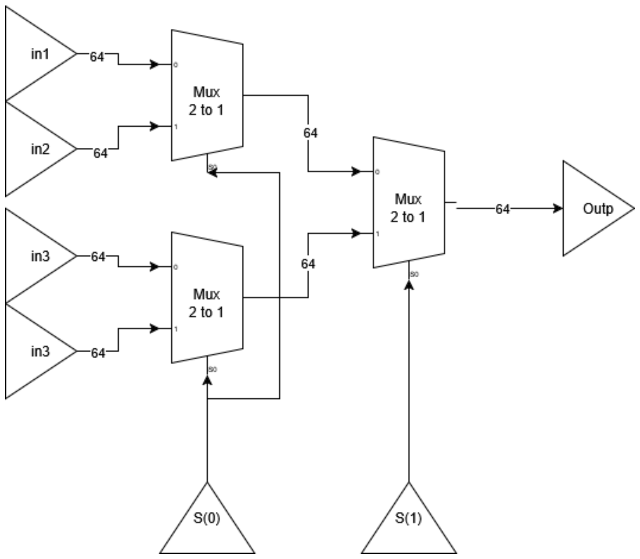
Inputs:

1. In1, in2 , in3, in4 : 64 bits option bitstrings
2. S : 2 bit selection bitstring

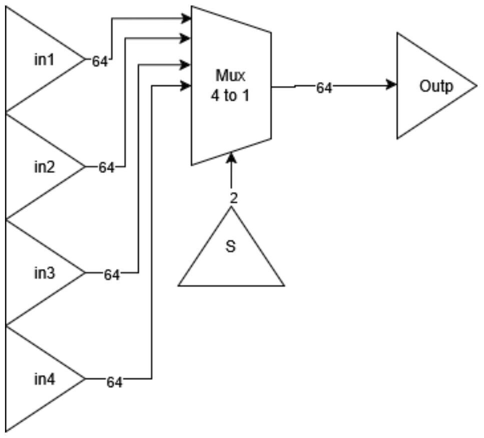
Outputs

1. Outp : 64 bit selected bitstring

Logic scheme:

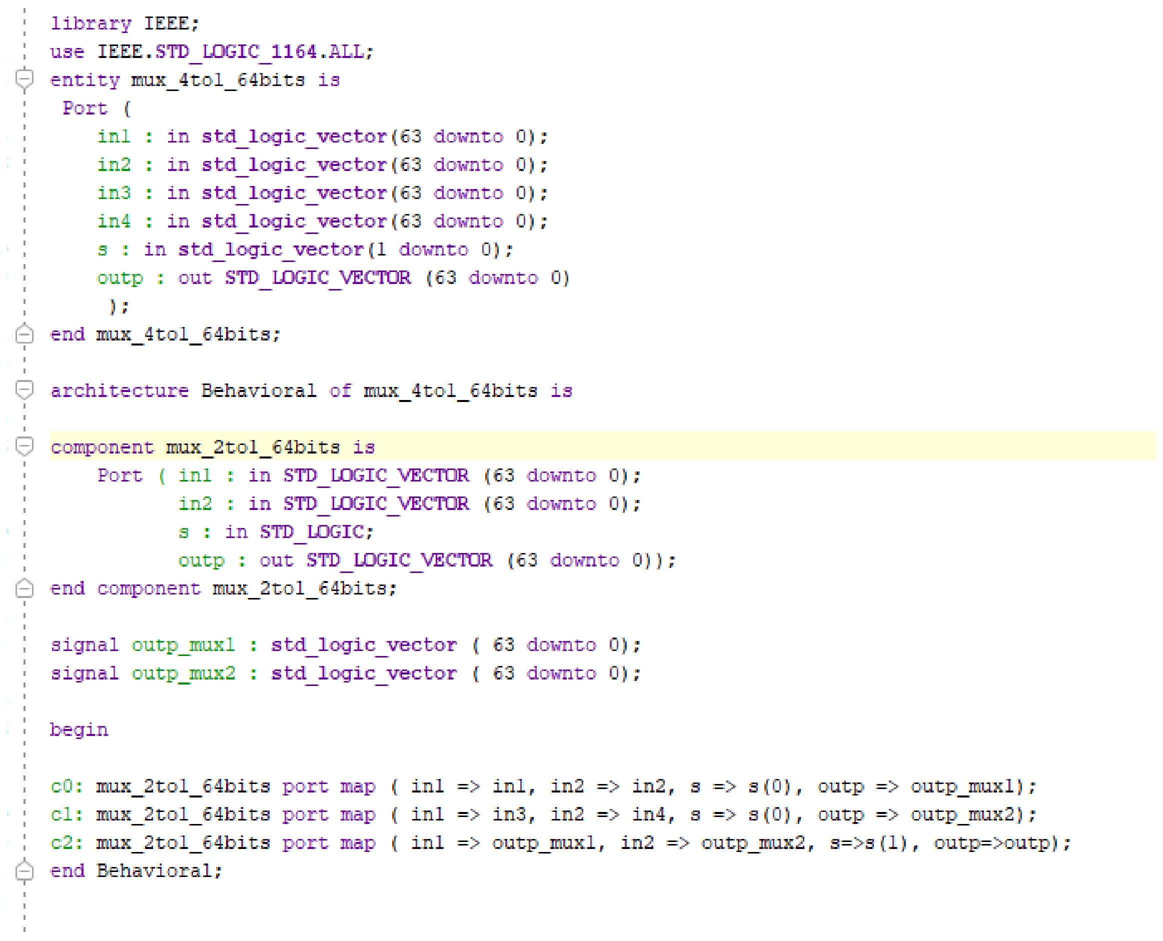


Black Box:



Faculty of Automation and Computer Science
Digital Systems Design Project
Darius-Eric Săsărman, group 30415, Year 2024-2025

Vhdl Code:



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux_4to1_64bits is
 Port (
    in1 : in std_logic_vector(63 downto 0);
    in2 : in std_logic_vector(63 downto 0);
    in3 : in std_logic_vector(63 downto 0);
    in4 : in std_logic_vector(63 downto 0);
    s : in std_logic_vector(1 downto 0);
    outp : out STD_LOGIC_VECTOR (63 downto 0)
     );
end mux_4to1_64bits;

architecture Behavioral of mux_4to1_64bits is

component mux_2to1_64bits is
    Port ( in1 : in STD_LOGIC_VECTOR (63 downto 0);
           in2 : in STD_LOGIC_VECTOR (63 downto 0);
           s : in STD_LOGIC;
           outp : out STD_LOGIC_VECTOR (63 downto 0));
end component mux_2to1_64bits;

signal outp_mux1 : std_logic_vector ( 63 downto 0);
signal outp_mux2 : std_logic_vector ( 63 downto 0);

begin

c0: mux_2to1_64bits port map ( in1 => in1, in2 => in2, s => s(0), outp => outp_mux1);
c1: mux_2to1_64bits port map ( in1 => in3, in2 => in4, s => s(0), outp => outp_mux2);
c2: mux_2to1_64bits port map ( in1 => outp_mux1, in2 => outp_mux2, s=>s(1), outp=>outp);
end Behavioral;
```

c.8 to 1 mux on 64 bits

Purpose:

To pick between 8 selections

Inputs :

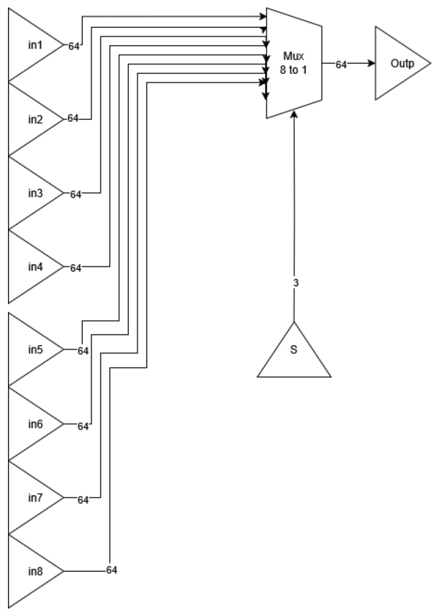
1. In1,in2, in3,in4,in5,in6,in7,in8 : selection 64-bit bitstrings
2. S : 3 bit bitstring

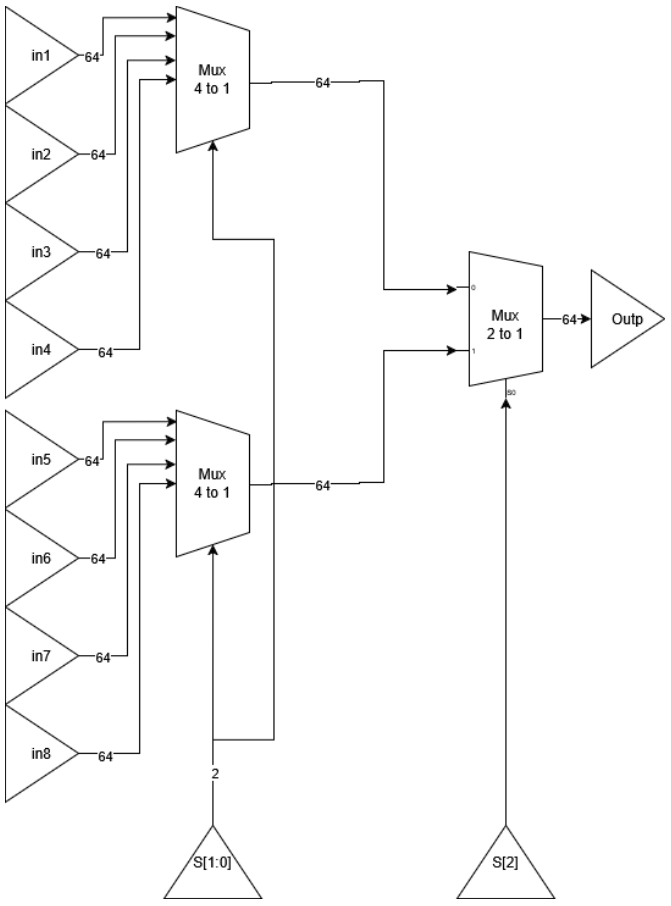
Outputs:

1. Outp : 64 -bit bitstring that has been selected by mux

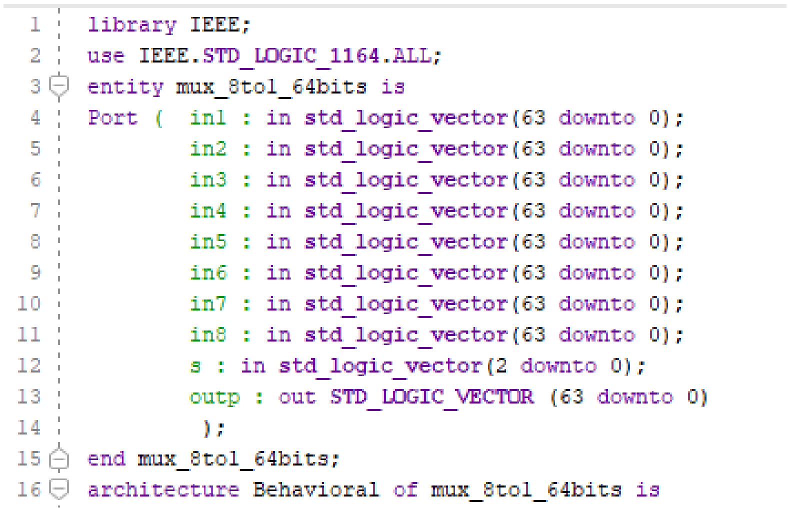
Logic Scheme(on next page):

Black box:

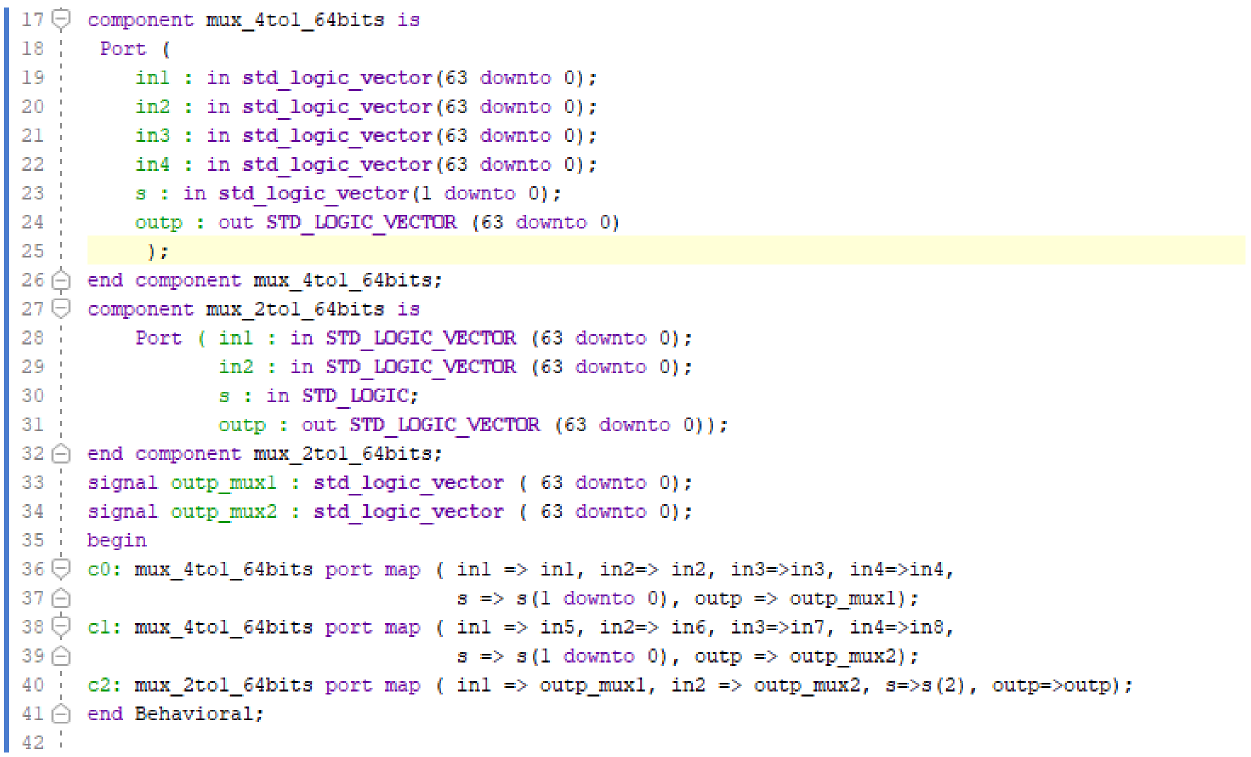




Vhdl Code:



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux_8to1_64bits is
Port (  in1 : in std_logic_vector(63 downto 0);
        in2 : in std_logic_vector(63 downto 0);
        in3 : in std_logic_vector(63 downto 0);
        in4 : in std_logic_vector(63 downto 0);
        in5 : in std_logic_vector(63 downto 0);
        in6 : in std_logic_vector(63 downto 0);
        in7 : in std_logic_vector(63 downto 0);
        in8 : in std_logic_vector(63 downto 0);
        s : in std_logic_vector(2 downto 0);
        outp : out STD_LOGIC_VECTOR (63 downto 0)
         );
end mux_8to1_64bits;
architecture Behavioral of mux_8to1_64bits is
```



```vhdl
component mux_4to1_64bits is
 Port (
    in1 : in std_logic_vector(63 downto 0);
    in2 : in std_logic_vector(63 downto 0);
    in3 : in std_logic_vector(63 downto 0);
    in4 : in std_logic_vector(63 downto 0);
    s : in std_logic_vector(1 downto 0);
    outp : out STD_LOGIC_VECTOR (63 downto 0)
     );
end component mux_4to1_64bits;
component mux_2to1_64bits is
    Port ( in1 : in STD_LOGIC_VECTOR (63 downto 0);
           in2 : in STD_LOGIC_VECTOR (63 downto 0);
           s : in STD_LOGIC;
           outp : out STD_LOGIC_VECTOR (63 downto 0));
end component mux_2to1_64bits;
signal outp_mux1 : std_logic_vector ( 63 downto 0);
signal outp_mux2 : std_logic_vector ( 63 downto 0);
begin
c0: mux_4to1_64bits port map ( in1 => in1, in2=> in2, in3=>in3, in4=>in4,
                               s => s(1 downto 0), outp => outp_mux1);
c1: mux_4to1_64bits port map ( in1 => in5, in2=> in6, in3=>in7, in4=>in8,
                               s => s(1 downto 0), outp => outp_mux2);
c2: mux_2to1_64bits port map ( in1 => outp_mux1, in2 => outp_mux2, s=>s(2), outp=>outp);
end Behavioral;
```

d.Adder Unit

i.1-bit full Adder

Purpose:

To calculate the sum and carry-out of two bits and a carry-in.

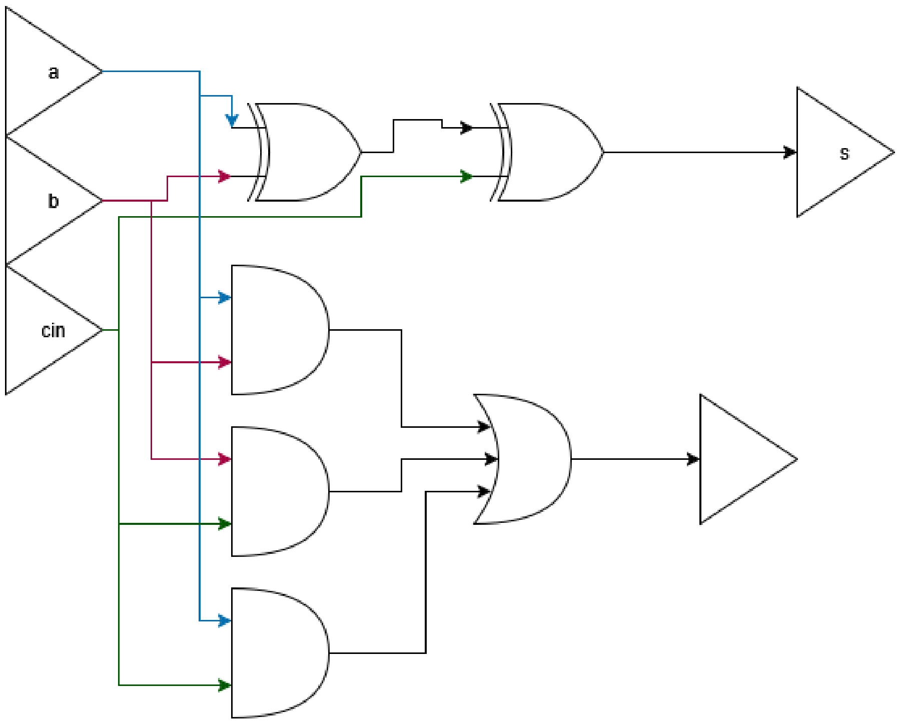
Inputs:

1. a: first input bit
2. b: second input bit
3. Cin: carry-in bit

Outputs:

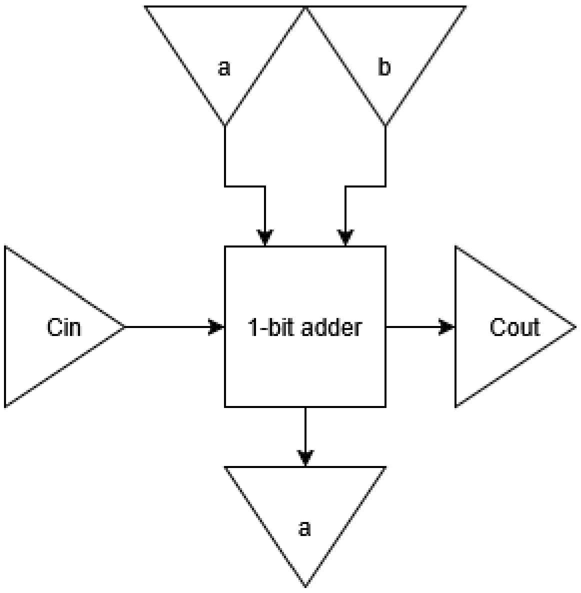
1. Cout : carry-out bit
2. S : sum bit

Logic Scheme:

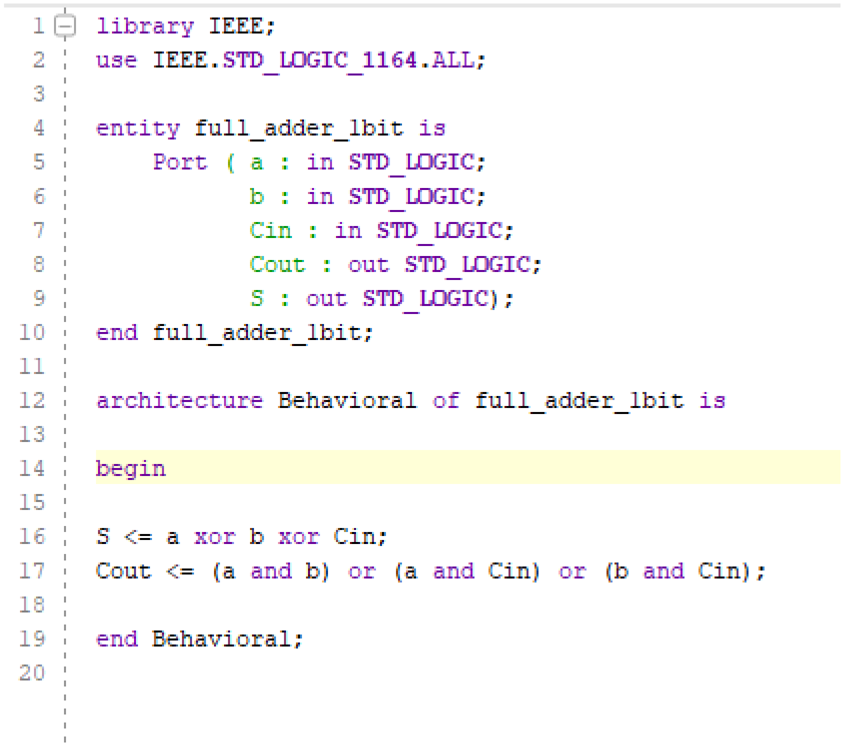


Faculty of Automation and Computer Science
Digital Systems Design Project
Darius-Eric Săsărman, group 30415, Year 2024-2025

Black Box:



Vhdl Code:



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

entity full_adder_1bit is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           Cin : in STD_LOGIC;
           Cout : out STD_LOGIC;
           S : out STD_LOGIC);
end full_adder_1bit;

architecture Behavioral of full_adder_1bit is

begin

S <= a xor b xor Cin;
Cout <= (a and b) or (a and Cin) or (b and Cin);

end Behavioral;
```

ii.8-bit full adder

Purpose:
To calculate the sum of two 8bit numbers

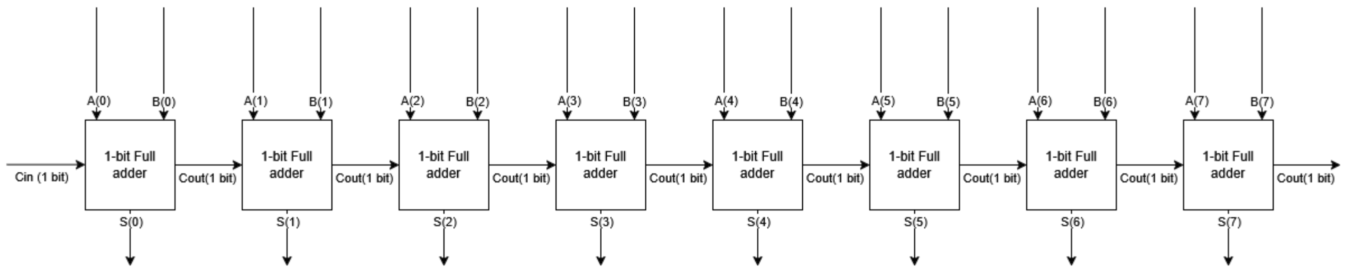
Inputs:

1. A: 8bit input
2. B: 8bit input
3. Cin: carry-in input

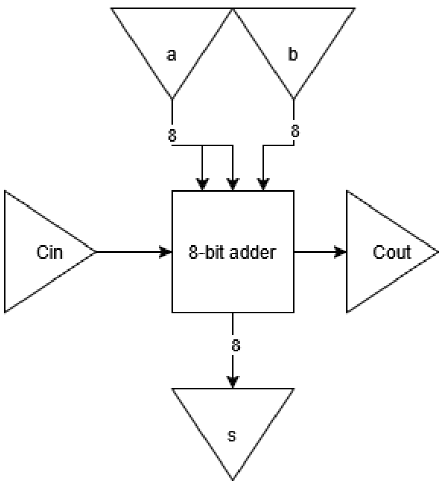
Outputs:

1. Cout: carry-out output
2. S: The sum of A and B and Carry-in on 8 bits

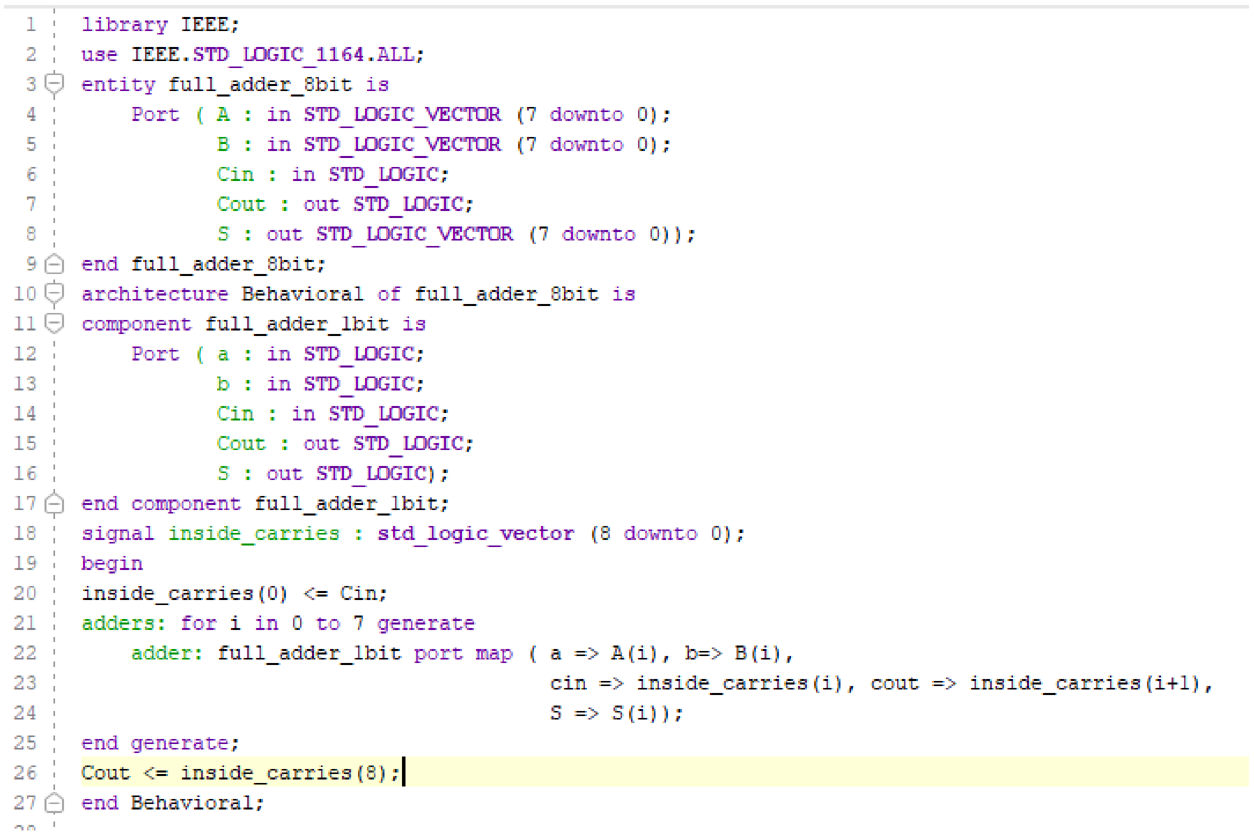
Logic Scheme:



Black box:



Vhdl Code:



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity full_adder_8bit is
    Port ( A : in STD_LOGIC_VECTOR (7 downto 0);
           B : in STD_LOGIC_VECTOR (7 downto 0);
           Cin : in STD_LOGIC;
           Cout : out STD_LOGIC;
           S : out STD_LOGIC_VECTOR (7 downto 0));
end full_adder_8bit;
architecture Behavioral of full_adder_8bit is
component full_adder_1bit is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           Cin : in STD_LOGIC;
           Cout : out STD_LOGIC;
           S : out STD_LOGIC);
end component full_adder_1bit;
signal inside_carries : std_logic_vector (8 downto 0);
begin
inside_carries(0) <= Cin;
adders: for i in 0 to 7 generate
    adder: full_adder_1bit port map ( a => A(i), b=> B(i),
                                      cin => inside_carries(i), cout => inside_carries(i+1),
                                      S => S(i));
end generate;
Cout <= inside_carries(8);
end Behavioral;
```

iii.Adder-unit general structure

Purpose:

To calculate the sum of two eight bit numbers. Because the result can be a 2’s complement number on 9 bits, the two inputs have to be resized before calculating their sum.

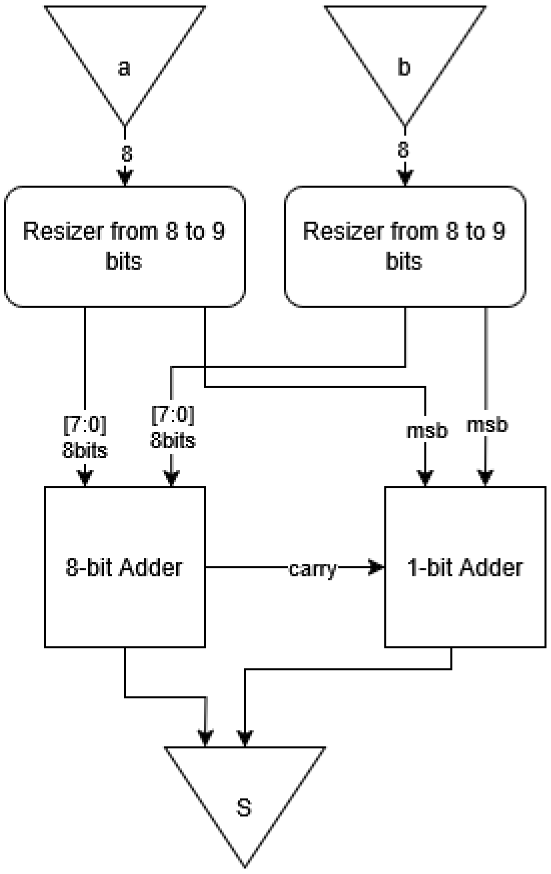
Inputs :

1. A : 8 bit number in 2’s complement
2. B : 8bit number in 2's complement

Outputs:

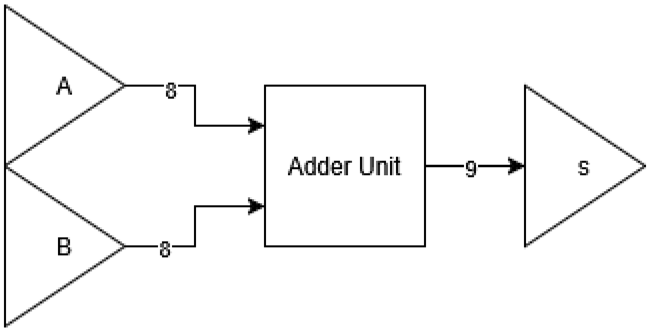
1. S : 9 bit number in 2's complement

Logic Scheme:

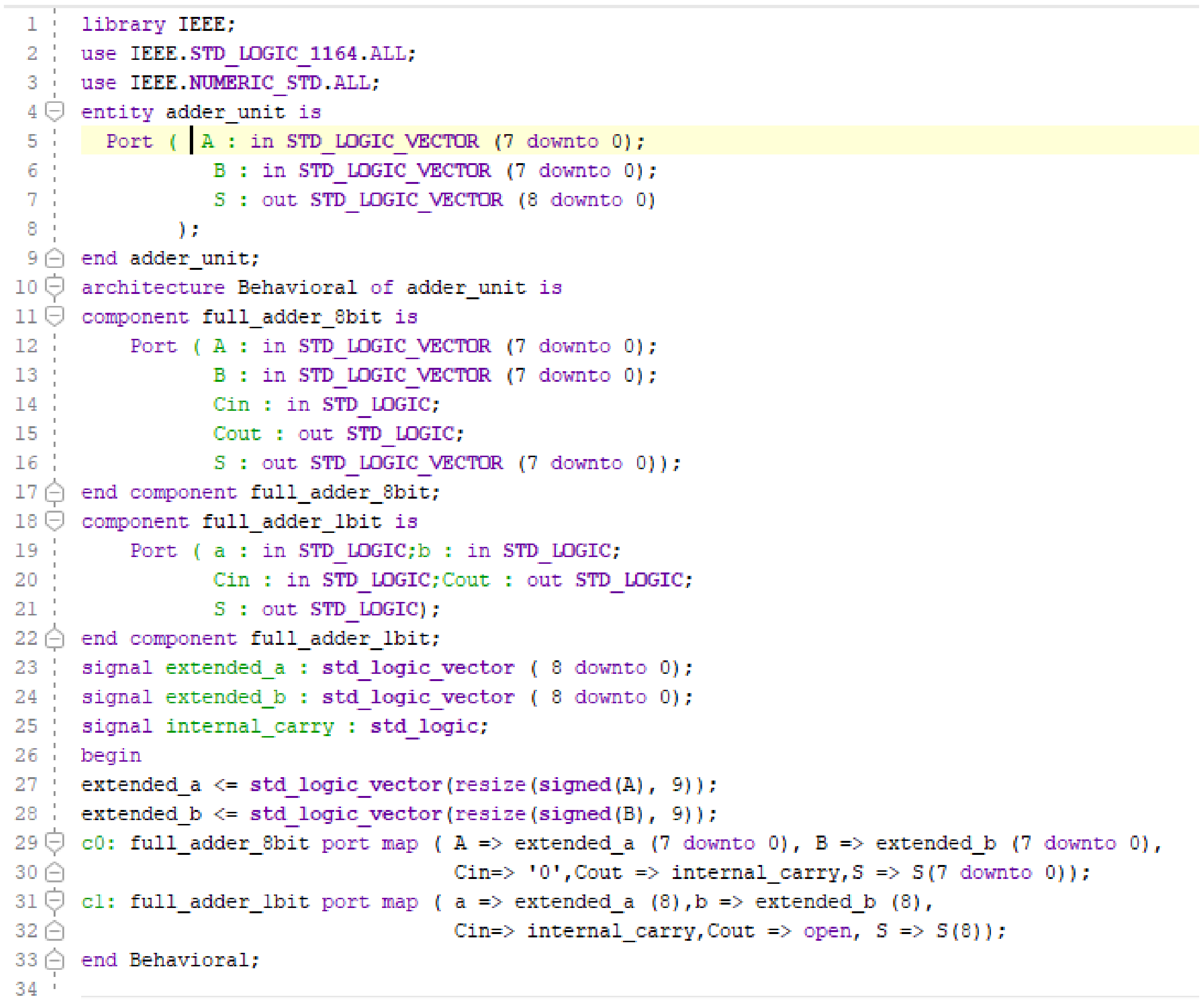


Faculty of Automation and Computer Science
Digital Systems Design Project
Darius-Eric Săsărman, group 30415, Year 2024-2025

Black Box:



Vhdl Code:



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity adder_unit is
  Port ( A : in STD_LOGIC_VECTOR (7 downto 0);
          B : in STD_LOGIC_VECTOR (7 downto 0);
          S : out STD_LOGIC_VECTOR (8 downto 0)
       );
end adder_unit;
architecture Behavioral of adder_unit is
component full_adder_8bit is
    Port ( A : in STD_LOGIC_VECTOR (7 downto 0);
           B : in STD_LOGIC_VECTOR (7 downto 0);
           Cin : in STD_LOGIC;
           Cout : out STD_LOGIC;
           S : out STD_LOGIC_VECTOR (7 downto 0));
end component full_adder_8bit;
component full_adder_1bit is
    Port ( a : in STD_LOGIC;b : in STD_LOGIC;
           Cin : in STD_LOGIC;Cout : out STD_LOGIC;
           S : out STD_LOGIC);
end component full_adder_1bit;
signal extended_a : std_logic_vector ( 8 downto 0);
signal extended_b : std_logic_vector ( 8 downto 0);
signal internal_carry : std_logic;
begin
extended_a <= std_logic_vector(resize(signed(A), 9));
extended_b <= std_logic_vector(resize(signed(B), 9));
c0: full_adder_8bit port map ( A => extended_a (7 downto 0), B => extended_b (7 downto 0),
                              Cin=> '0',Cout => internal_carry,S => S(7 downto 0));
c1: full_adder_1bit port map ( a => extended_a (8),b => extended_b (8),
                              Cin=> internal_carry,Cout => open, S => S(8));
end Behavioral;
```

Didn’t describe the logic behind the resizer because it’s just a command in VHDL.

e.Subtracter Unit

i. 1-bit full Subtracter

Purpose:

To calculate the difference between two one-bit numbers

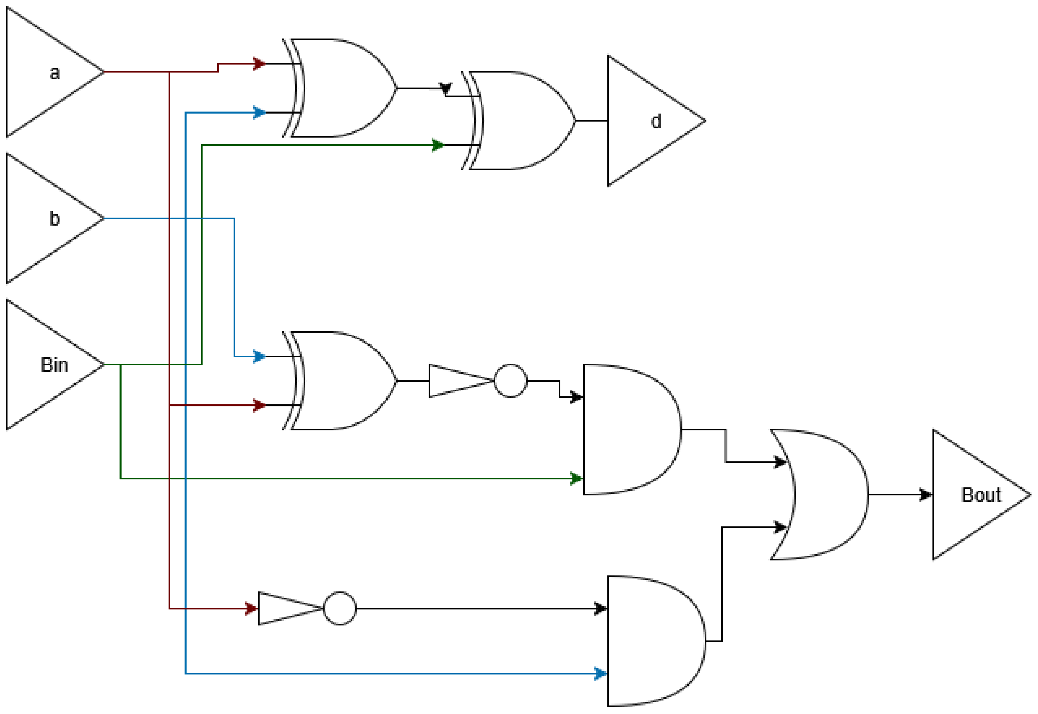
Inputs:

1. A : first 1bit number
2. B: second 1bit number
3. Bin : borrow-in bit

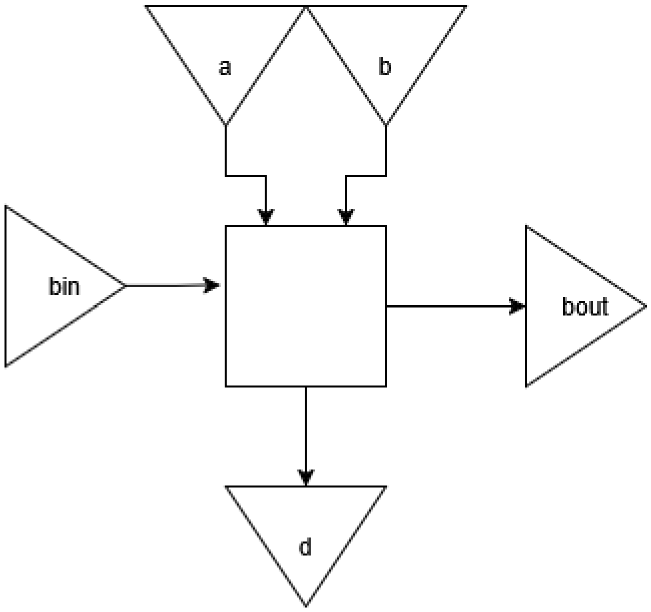
Outputs:

1. Bout: borrow-out bit
2. D: difference bit

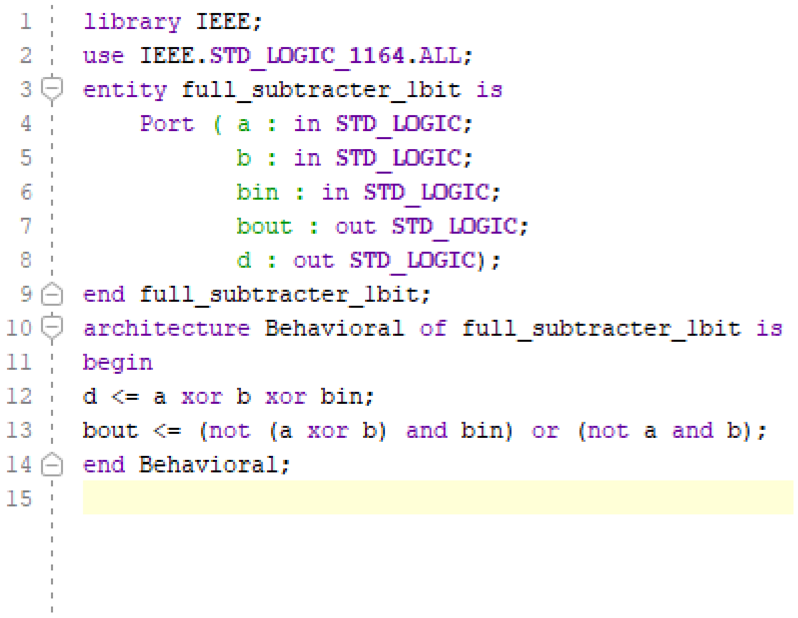
Logic scheme:



Black box:



Code:



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity full_subtracter_1bit is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           bin : in STD_LOGIC;
           bout : out STD_LOGIC;
           d : out STD_LOGIC);
end full_subtracter_1bit;
architecture Behavioral of full_subtracter_1bit is
begin
d <= a xor b xor bin;
bout <= (not (a xor b) and bin) or (not a and b);
end Behavioral;

```

ii. 8bit Full subtracter

Purpose:

To calculate the difference between two eight-bit numbers.

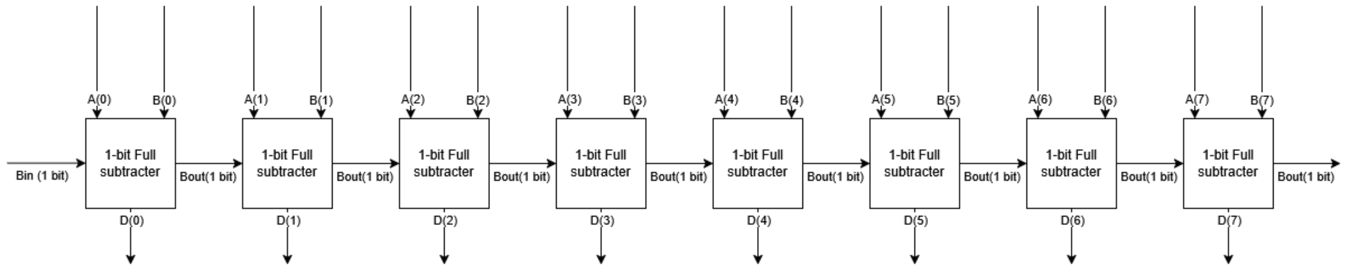
Inputs:

4. A : first 8bit number
5. B: second 8bit number
6. Bin : borrow-in bit

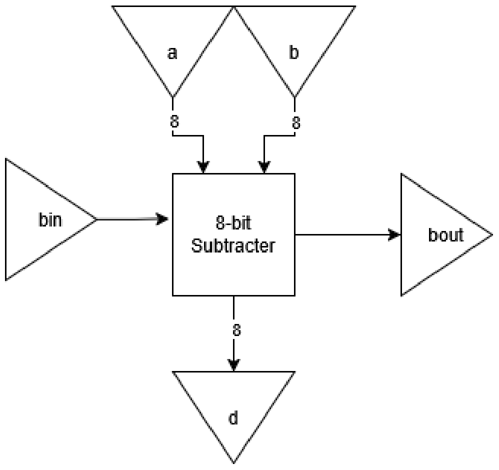
Outputs:

3. Bout: borrow-out bit
4. D: difference 8-bit number

Logic scheme:



Black box:



iii.Subtraction Unit General Structure

Purpose:

To calculate the difference of two eight bit numbers. Because the result can be a 2's complement number on 9 bits, the two inputs have to be resized before calculating their difference.

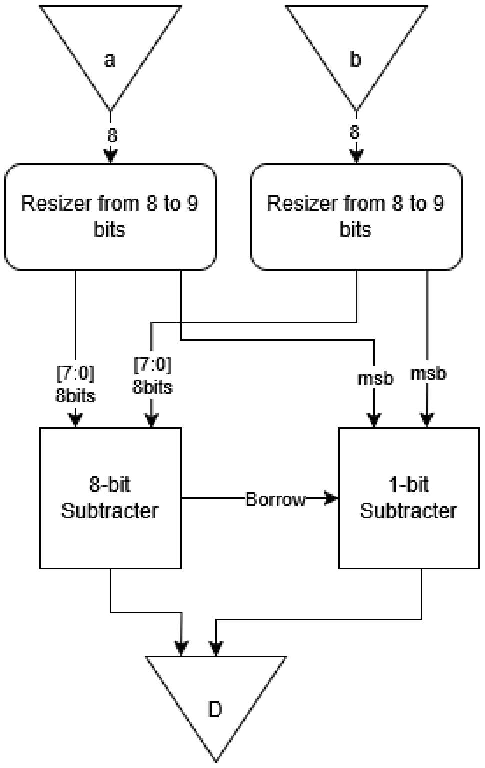
Inputs :

3. A : 8 bit number in 2's complement
4. B : 8bit number in 2's complement

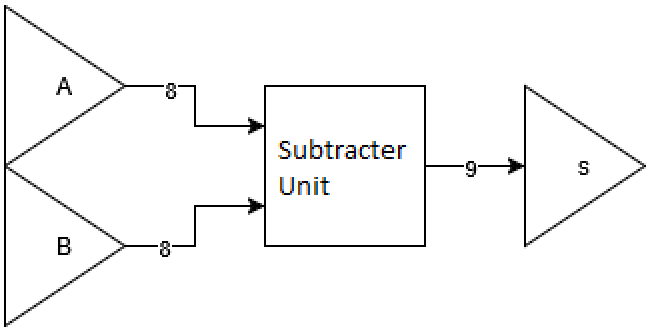
Outputs:

2. D : 9 bit number in 2's complement

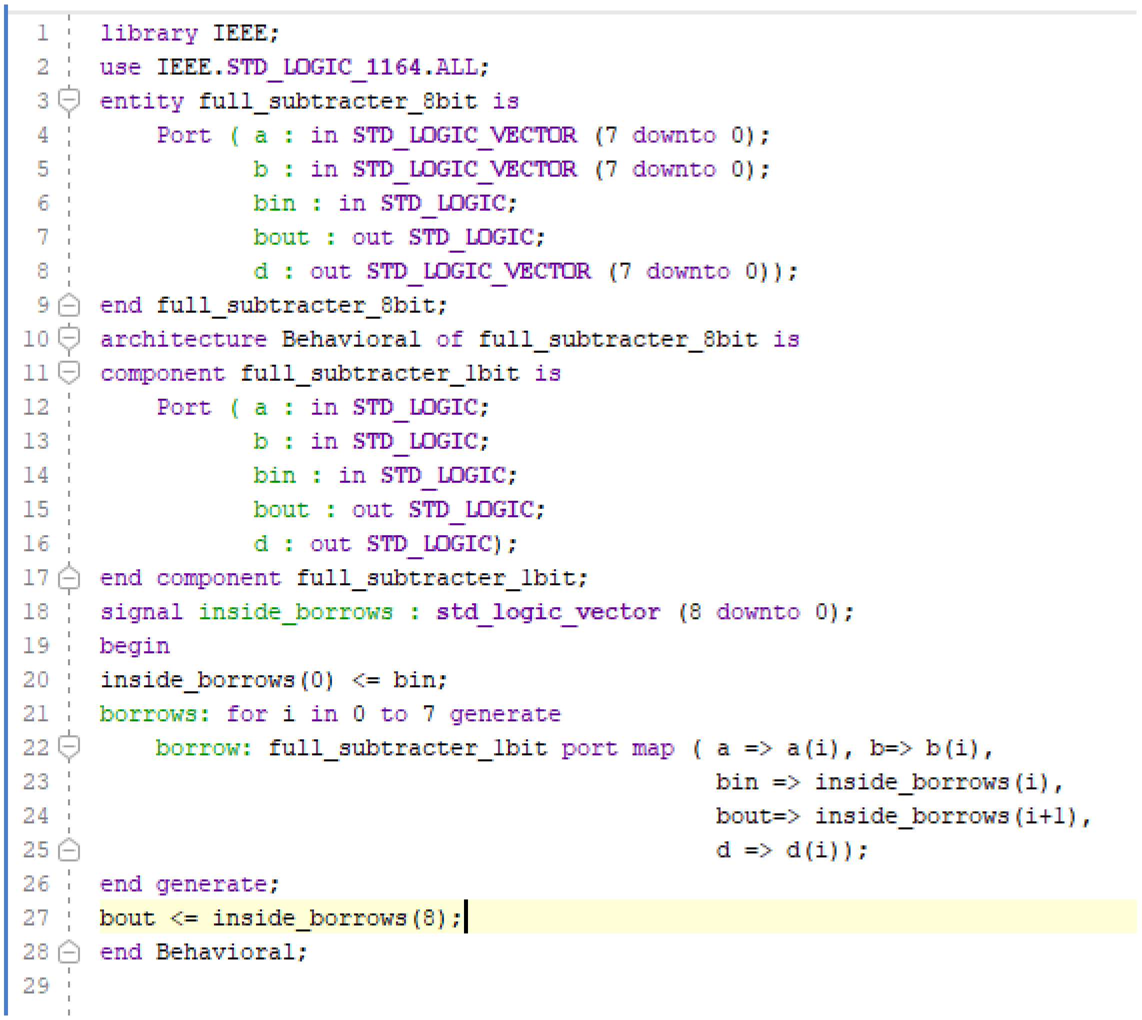
Logic Scheme:



Black Box:



Vhdl Code:



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity full_subtracter_8bit is
    Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
           b : in STD_LOGIC_VECTOR (7 downto 0);
           bin : in STD_LOGIC;
           bout : out STD_LOGIC;
           d : out STD_LOGIC_VECTOR (7 downto 0));
end full_subtracter_8bit;
architecture Behavioral of full_subtracter_8bit is
component full_subtracter_1bit is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           bin : in STD_LOGIC;
           bout : out STD_LOGIC;
           d : out STD_LOGIC);
end component full_subtracter_1bit;
signal inside_borrows : std_logic_vector (8 downto 0);
begin
inside_borrows(0) <= bin;
borrows: for i in 0 to 7 generate
    borrow: full_subtracter_1bit port map ( a => a(i), b=> b(i),
                                             bin => inside_borrows(i),
                                             bout=> inside_borrows(i+1),
                                             d => d(i));
end generate;
bout <= inside_borrows(8);
end Behavioral;

```

f.Multiplier Unit

i.State Diagram

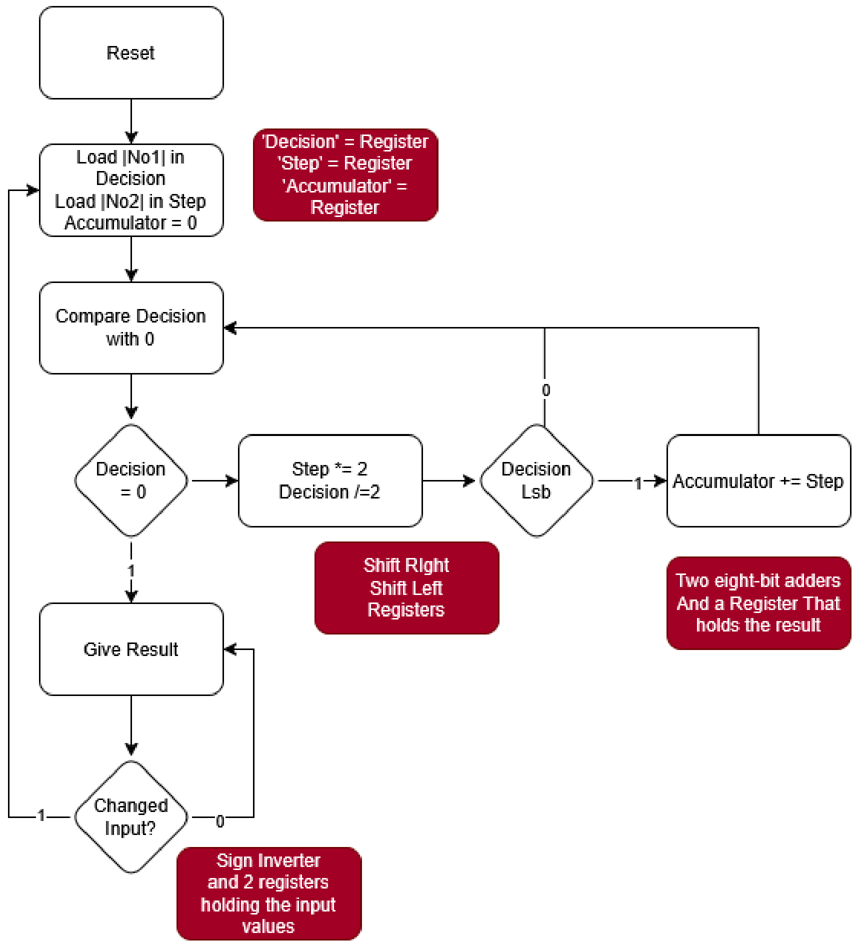
Main Reasoning behind the Multiplication algorithm:

Shift and add:

Example :1010 * 1100 = (1000 + 0010) * 1100 = 1000 * 1100 + 0010 * 1100 =
= 1100 << 3 + 1100 << 1 =
= 1100 << 3 * $N1_3$ + 1100 << 2 * $N1_2$ + 1100 <<1*$N1_1$+1100*$N1_0$

Based on the bits of the first number, we can determine what shifted versions of the second we can add to the sum. The pro of this algorithm is the fact that it executes at most 8 additions. The cons are that it works only for numbers that are not in two's complement. This issue is solved by saving the sign difference, storing the absolute value of the numbers, and at the end, negate the result, if there was any difference between the signs

State Diagram:



Faculty of Automation and Computer Science
Digital Systems Design Project
Darius-Eric Săsărman, group 30415, Year 2024-2025

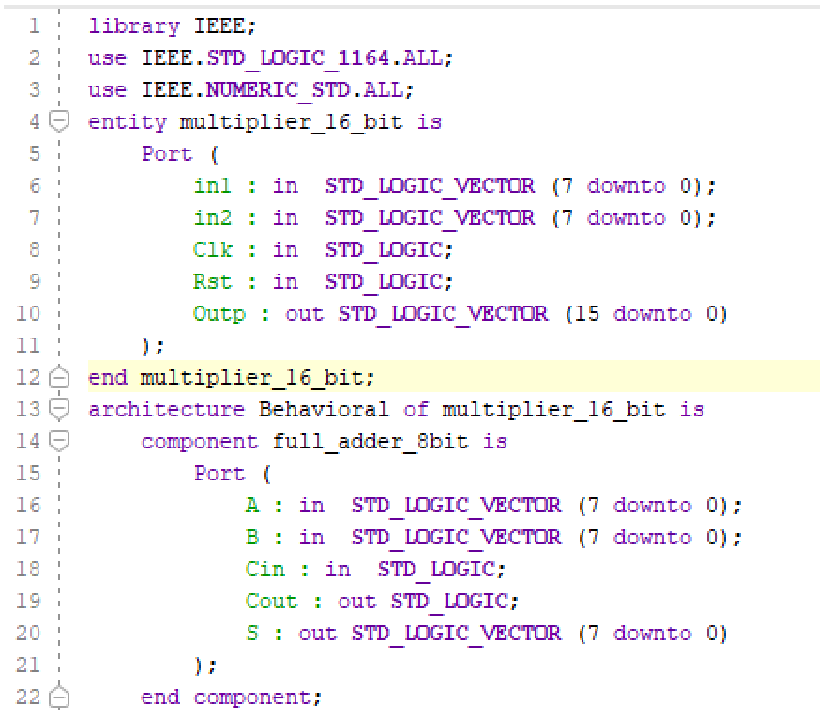
Inputs:

1. In1: first number on 8 bits
2. In2: second number on 8 bits
3. Clk : clock signal for sequential execution
4. Rst: Asynchronous reset

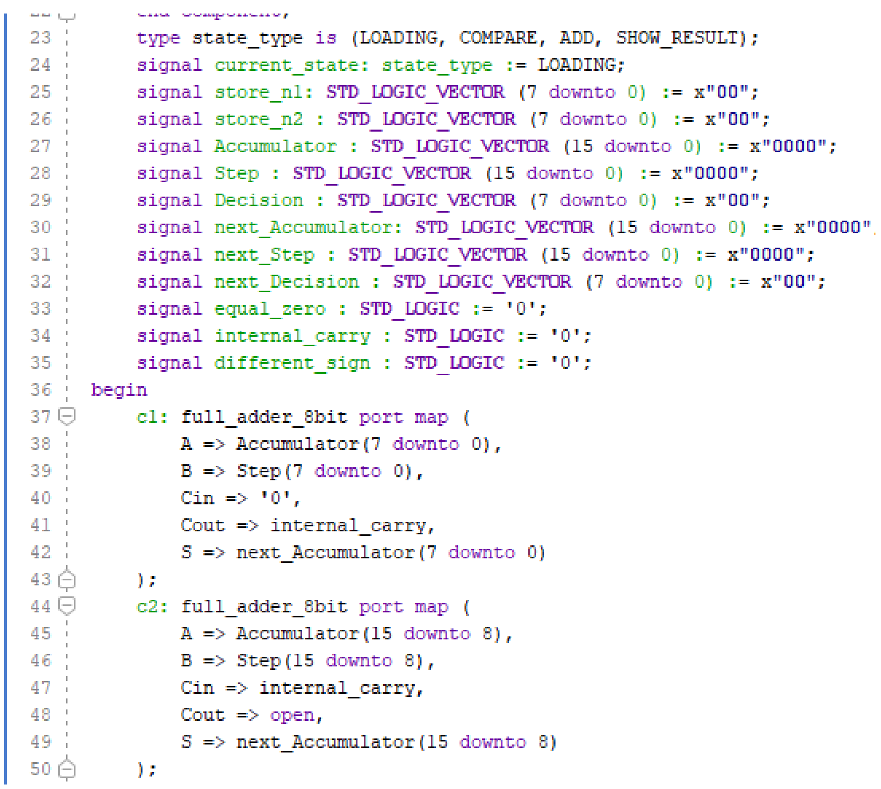
Outputs:

1. Multiplication result on 16 bits

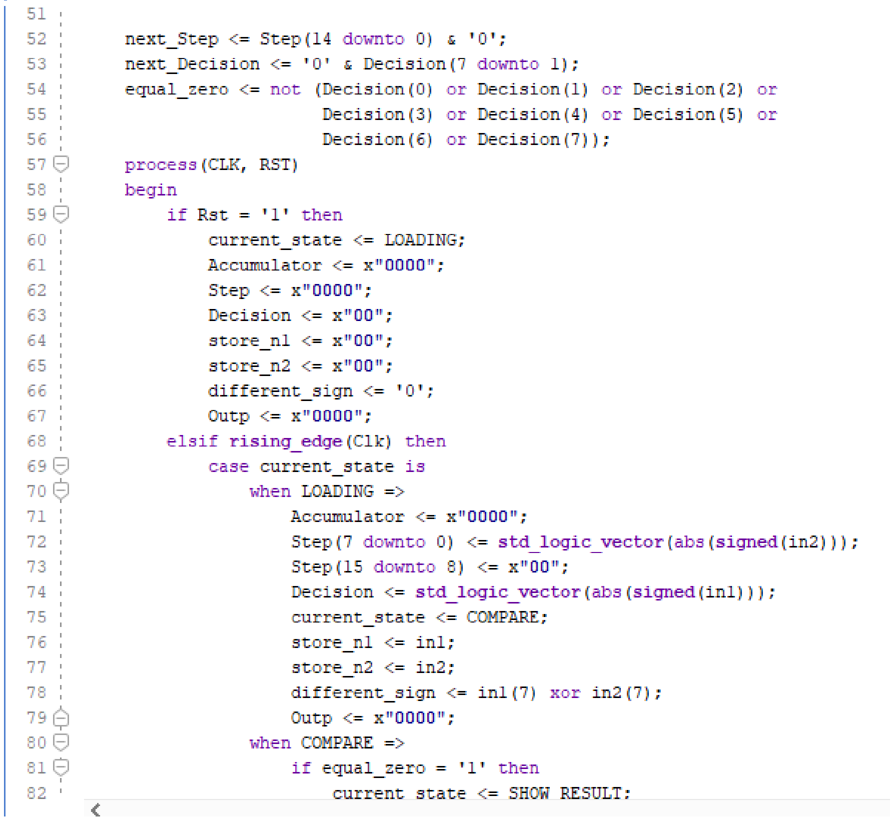
Vhdl Code:



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity multiplier_16_bit is
    Port (
        in1 : in  STD_LOGIC_VECTOR (7 downto 0);
        in2 : in  STD_LOGIC_VECTOR (7 downto 0);
        Clk : in  STD_LOGIC;
        Rst : in  STD_LOGIC;
        Outp : out STD_LOGIC_VECTOR (15 downto 0)
    );
end multiplier_16_bit;
architecture Behavioral of multiplier_16_bit is
    component full_adder_8bit is
        Port (
            A : in  STD_LOGIC_VECTOR (7 downto 0);
            B : in  STD_LOGIC_VECTOR (7 downto 0);
            Cin : in  STD_LOGIC;
            Cout : out STD_LOGIC;
            S : out STD_LOGIC_VECTOR (7 downto 0)
        );
    end component;
```

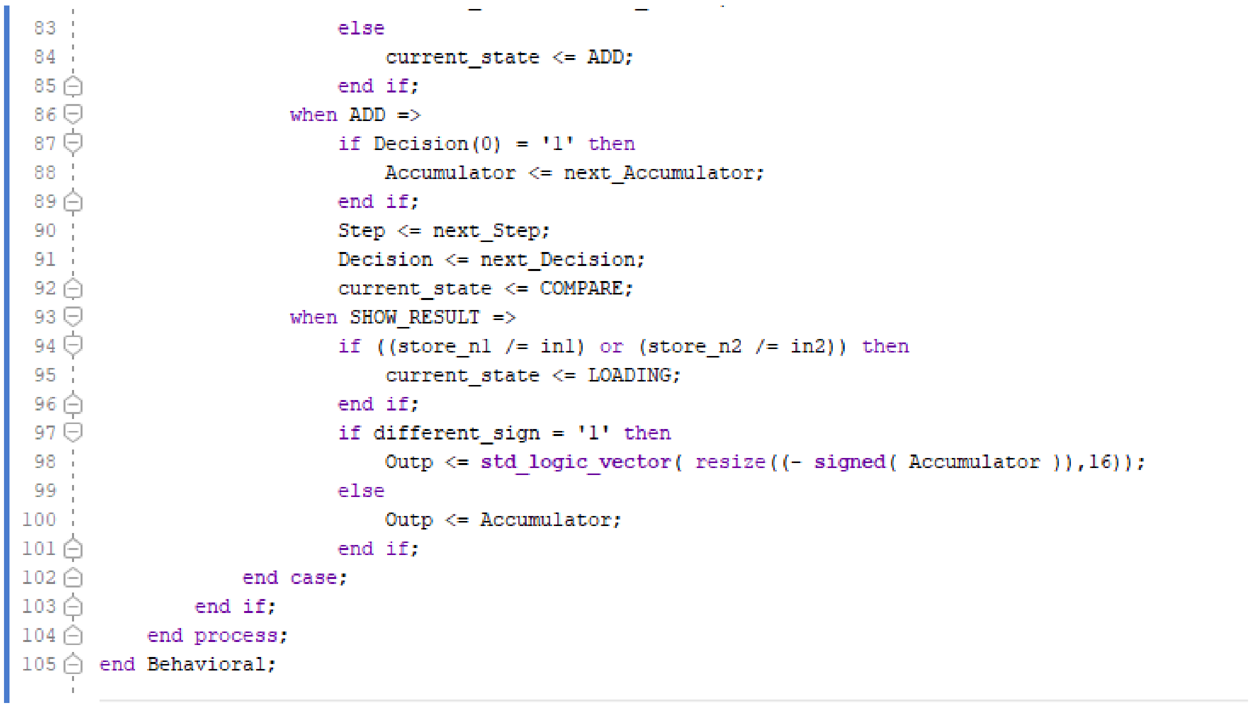


```vhdl
    type state_type is (LOADING, COMPARE, ADD, SHOW_RESULT);
    signal current_state: state_type := LOADING;
    signal store_n1: STD_LOGIC_VECTOR (7 downto 0) := x"00";
    signal store_n2 : STD_LOGIC_VECTOR (7 downto 0) := x"00";
    signal Accumulator : STD_LOGIC_VECTOR (15 downto 0) := x"0000";
    signal Step : STD_LOGIC_VECTOR (15 downto 0) := x"0000";
    signal Decision : STD_LOGIC_VECTOR (7 downto 0) := x"00";
    signal next_Accumulator: STD_LOGIC_VECTOR (15 downto 0) := x"0000"
    signal next_Step : STD_LOGIC_VECTOR (15 downto 0) := x"0000";
    signal next_Decision : STD_LOGIC_VECTOR (7 downto 0) := x"00";
    signal equal_zero : STD_LOGIC := '0';
    signal internal_carry : STD_LOGIC := '0';
    signal different_sign : STD_LOGIC := '0';
begin
    c1: full_adder_8bit port map (
        A => Accumulator(7 downto 0),
        B => Step(7 downto 0),
        Cin => '0',
        Cout => internal_carry,
        S => next_Accumulator(7 downto 0)
    );
    c2: full_adder_8bit port map (
        A => Accumulator(15 downto 8),
        B => Step(15 downto 8),
        Cin => internal_carry,
        Cout => open,
        S => next_Accumulator(15 downto 8)
    );
```



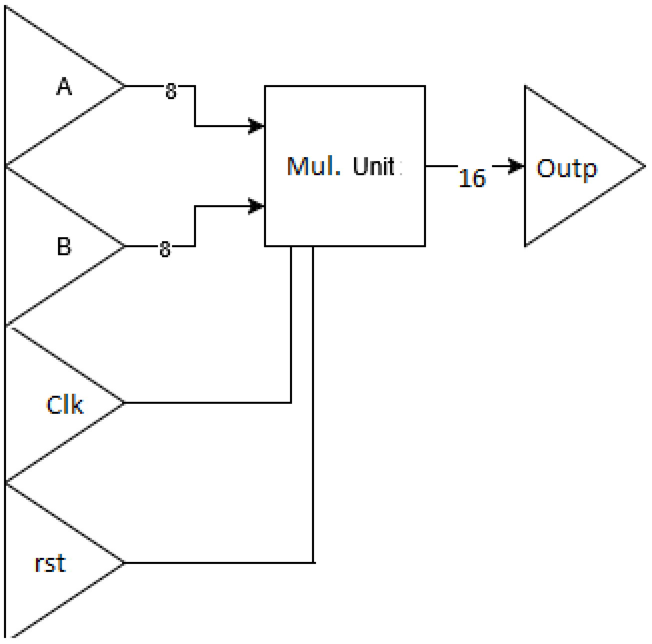
```vhdl

    next_Step <= Step(14 downto 0) & '0';
    next_Decision <= '0' & Decision(7 downto 1);
    equal_zero <= not (Decision(0) or Decision(1) or Decision(2) or
                       Decision(3) or Decision(4) or Decision(5) or
                       Decision(6) or Decision(7));
    process(CLK, RST)
    begin
        if Rst = '1' then
            current_state <= LOADING;
            Accumulator <= x"0000";
            Step <= x"0000";
            Decision <= x"00";
            store_n1 <= x"00";
            store_n2 <= x"00";
            different_sign <= '0';
            Outp <= x"0000";
        elsif rising_edge(Clk) then
            case current_state is
                when LOADING =>
                    Accumulator <= x"0000";
                    Step(7 downto 0) <= std_logic_vector(abs(signed(in2)));
                    Step(15 downto 8) <= x"00";
                    Decision <= std_logic_vector(abs(signed(in1)));
                    current_state <= COMPARE;
                    store_n1 <= in1;
                    store_n2 <= in2;
                    different_sign <= in1(7) xor in2(7);
                    Outp <= x"0000";
                when COMPARE =>
                    if equal_zero = '1' then
                        current_state <= SHOW_RESULT;
```



```
                    else
                        current_state <= ADD;
                    end if;
                when ADD =>
                    if Decision(0) = '1' then
                        Accumulator <= next_Accumulator;
                    end if;
                    Step <= next_Step;
                    Decision <= next_Decision;
                    current_state <= COMPARE;
                when SHOW_RESULT =>
                    if ((store_n1 /= in1) or (store_n2 /= in2)) then
                        current_state <= LOADING;
                    end if;
                    if different_sign = '1' then
                        Outp <= std_logic_vector( resize((- signed( Accumulator )),16));
                    else
                        Outp <= Accumulator;
                    end if;
            end case;
        end if;
    end process;
end Behavioral;
```

Black box representation:

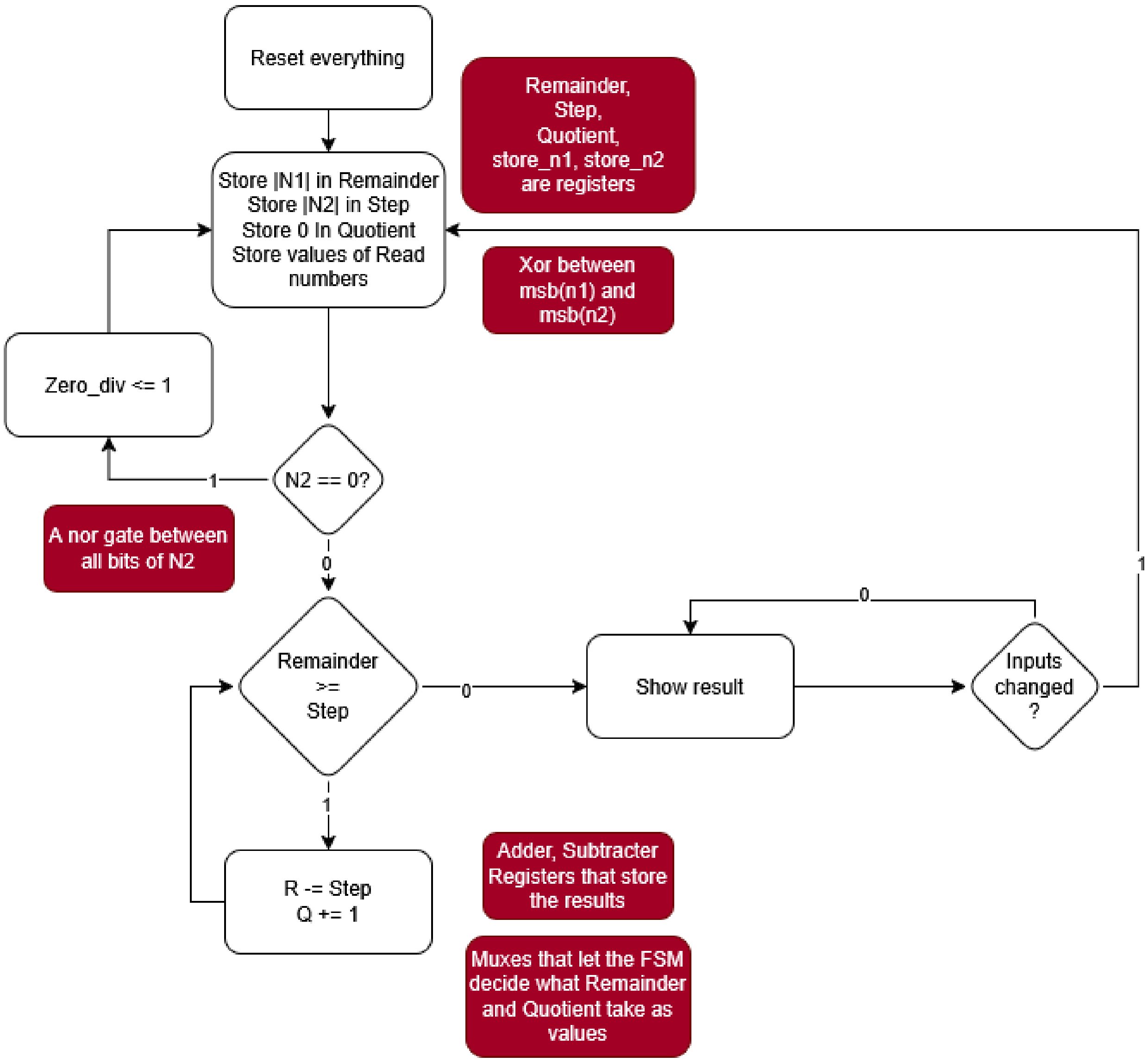


Logic representation hasn't been provided because the design is mostly behavioral, not structural, nor dataflow.

g.Division Unit

Algorithm:
Repeated subtraction. Works for unsigned numbers. To use it for number on two's complement we just have to extract absolute value, and after the result is give, add the signe to the representation.

State Diagram:


Reset everything
Remainder, Step, Quotient, store_n1, store_n2 are registers
Store |N1| in Remainder Store |N2| in Step Store 0 In Quotient Store values of Read numbers
Xor between msb(n1) and msb(n2)
Zero_div <= 1
1
N2 == 0?
A nor gate between all bits of N2
0
1
0
Remainder >= Step
0
Show result
Inputs changed ?
1
R -= Step Q += 1
Adder, Subtracter Registers that store the results
Muxes that let the FSM decide what Remainder and Quotient take as values

Inputs:

1. No1 : first 8bit number in 2's complement

2. No2 : second 8bit number in 2’s complement

3. Clk: clock signal

4. Rst: reset signal

Outputs:

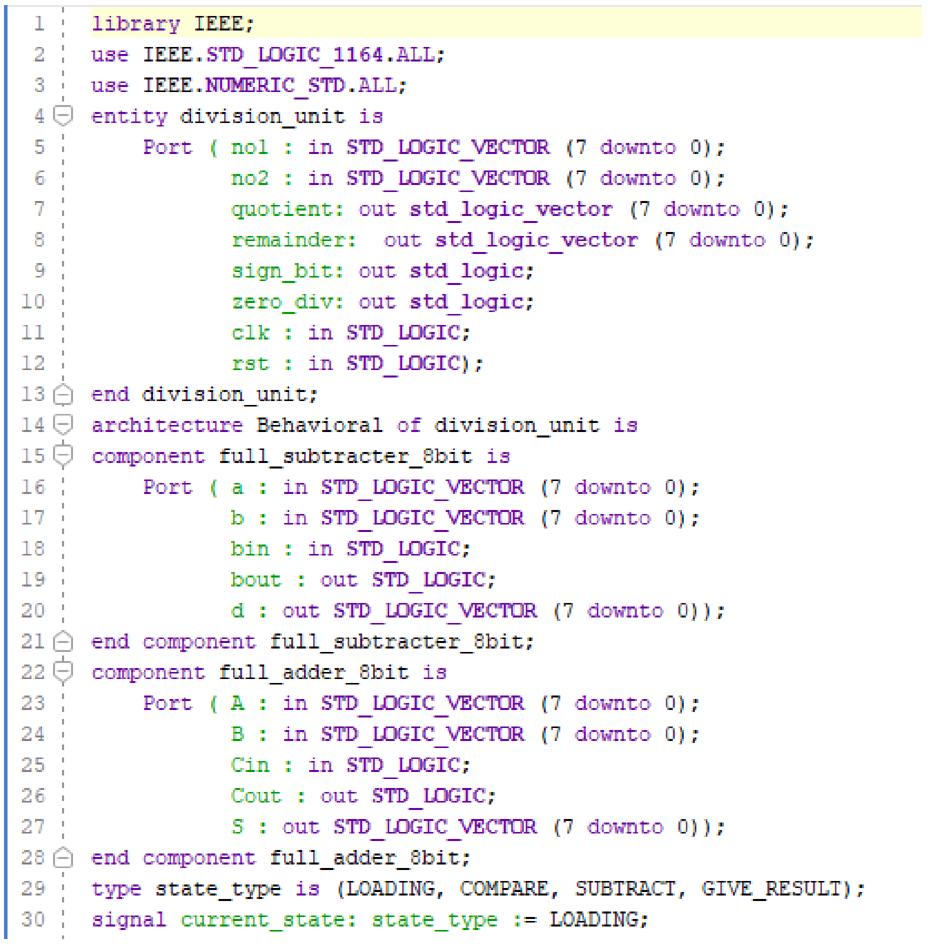
1. Quotient : modulus of quotient value

2. Remainder: modulus of remainder value

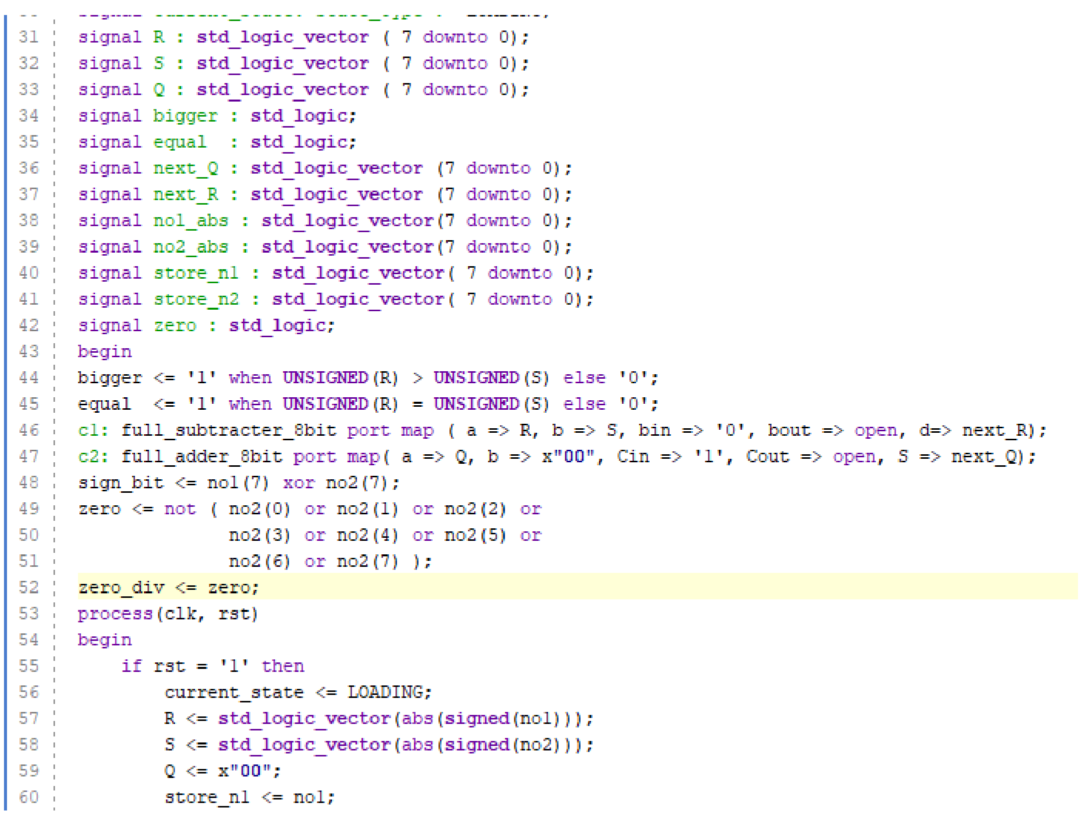
3. Sign bit : tells the output processor to put a minus sign or not in front of the quotient result

5. Zero div bit: is 1 when division by 0 takes place

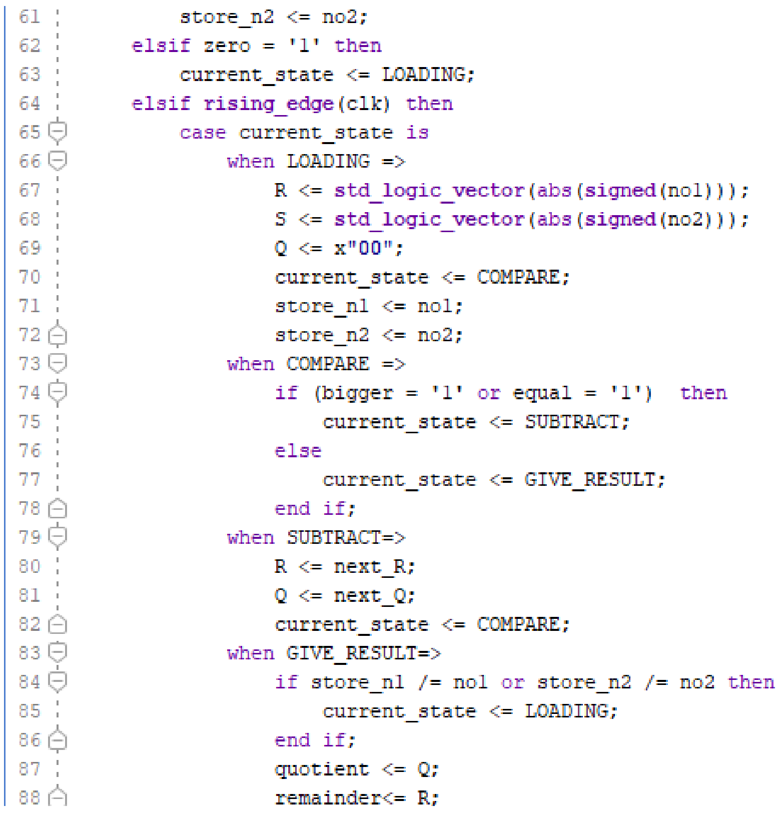
Vhdl Code:



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity division_unit is
    Port ( no1 : in STD_LOGIC_VECTOR (7 downto 0);
           no2 : in STD_LOGIC_VECTOR (7 downto 0);
           quotient: out std_logic_vector (7 downto 0);
           remainder:  out std_logic_vector (7 downto 0);
           sign_bit: out std_logic;
           zero_div: out std_logic;
           clk : in STD_LOGIC;
           rst : in STD_LOGIC);
end division_unit;
architecture Behavioral of division_unit is
component full_subtracter_8bit is
    Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
           b : in STD_LOGIC_VECTOR (7 downto 0);
           bin : in STD_LOGIC;
           bout : out STD_LOGIC;
           d : out STD_LOGIC_VECTOR (7 downto 0));
end component full_subtracter_8bit;
component full_adder_8bit is
    Port ( A : in STD_LOGIC_VECTOR (7 downto 0);
           B : in STD_LOGIC_VECTOR (7 downto 0);
           Cin : in STD_LOGIC;
           Cout : out STD_LOGIC;
           S : out STD_LOGIC_VECTOR (7 downto 0));
end component full_adder_8bit;
type state_type is (LOADING, COMPARE, SUBTRACT, GIVE_RESULT);
signal current_state: state_type := LOADING;
```



```
signal R : std_logic_vector ( 7 downto 0);
signal S : std_logic_vector ( 7 downto 0);
signal Q : std_logic_vector ( 7 downto 0);
signal bigger : std_logic;
signal equal  : std_logic;
signal next_Q : std_logic_vector (7 downto 0);
signal next_R : std_logic_vector (7 downto 0);
signal no1_abs : std_logic_vector(7 downto 0);
signal no2_abs : std_logic_vector(7 downto 0);
signal store_n1 : std_logic_vector( 7 downto 0);
signal store_n2 : std_logic_vector( 7 downto 0);
signal zero : std_logic;
begin
bigger <= '1' when UNSIGNED(R) > UNSIGNED(S) else '0';
equal  <= '1' when UNSIGNED(R) = UNSIGNED(S) else '0';
c1: full_subtracter_8bit port map ( a => R, b => S, bin => '0', bout => open, d=> next_R);
c2: full_adder_8bit port map( a => Q, b => x"00", Cin => '1', Cout => open, S => next_Q);
sign_bit <= no1(7) xor no2(7);
zero <= not ( no2(0) or no2(1) or no2(2) or
              no2(3) or no2(4) or no2(5) or
              no2(6) or no2(7) );
zero_div <= zero;
process(clk, rst)
begin
    if rst = '1' then
        current_state <= LOADING;
        R <= std_logic_vector(abs(signed(no1)));
        S <= std_logic_vector(abs(signed(no2)));
        Q <= x"00";
        store_n1 <= no1;
```

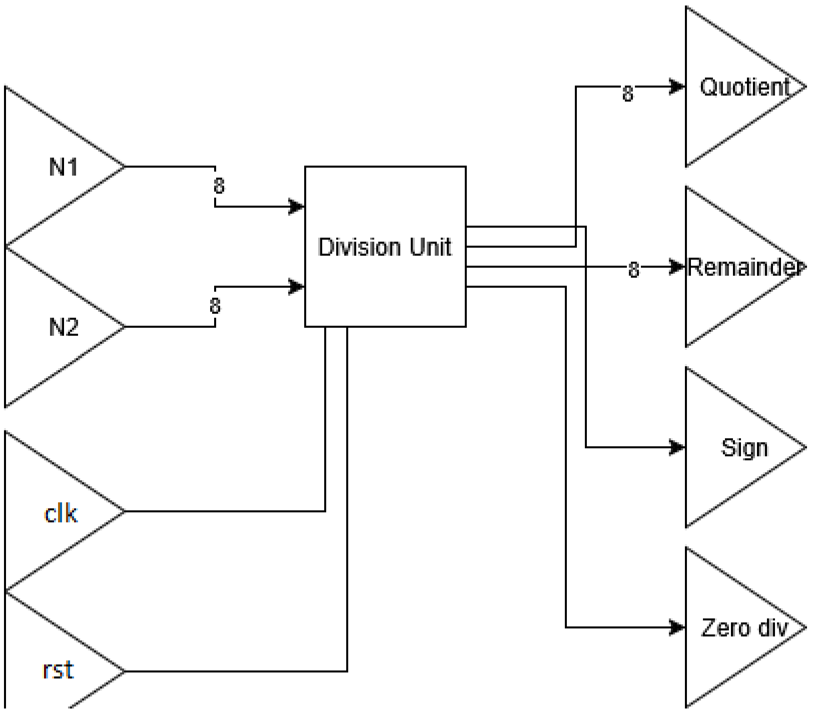


```
        store_n2 <= no2;
    elsif zero = '1' then
        current_state <= LOADING;
    elsif rising_edge(clk) then
        case current_state is
            when LOADING =>
                R <= std_logic_vector(abs(signed(no1)));
                S <= std_logic_vector(abs(signed(no2)));
                Q <= x"00";
                current_state <= COMPARE;
                store_n1 <= no1;
                store_n2 <= no2;
            when COMPARE =>
                if (bigger = '1' or equal = '1')  then
                    current_state <= SUBTRACT;
                else
                    current_state <= GIVE_RESULT;
                end if;
            when SUBTRACT=>
                R <= next_R;
                Q <= next_Q;
                current_state <= COMPARE;
            when GIVE_RESULT=>
                if store_n1 /= no1 or store_n2 /= no2 then
                    current_state <= LOADING;
                end if;
                quotient <= Q;
                remainder<= R;
            end case;
        end if;
    end process;

    end Behavioral;

```

Black box:



Logic representation hasn't been provided because the design is mostly behavioral, not structural, nor dataflow.

h.Output processors

It’s a collection of ROMs that have variable ranges.

Each digit of the number in base 10 has a corresponding seven-segment display value:

"C0"	0
"F9"	1
"A4"	2
"B0"	3
“99”	4
“92”	5
“82”	6
“F8”	7
“80”	8
“90”	9
"BF"	‘-‘
“AF”	‘r’

1. Addition output processor
2. Subtraction output processor

Each number is directly converted from signed to integer and then to ROM(integer_value).
Range is from -256 to 255.

```
outp(31 downto 0) <= ROM(to_integer(signed(inp)));
outp(63 downto 32)<= x"C0C0C0C0";
```

3. Multiplication output processor

Same thing as above but from -32768 to 32767.

```
outp <= ROM(to_integer(signed(inp)));
```

4. Division output processor

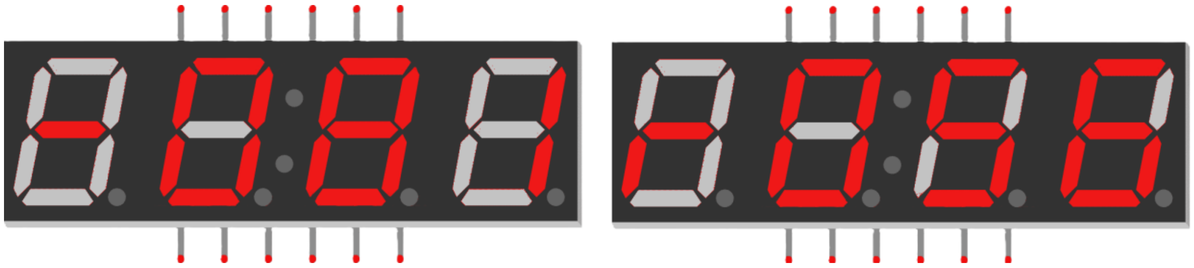
Output is as follows:

```
sign <= x"C0" when sign_bit = '0' else x"BF";

outp(63 downto 56) <= sign;
outp(55 downto 32) <= ROM(to_integer(unsigned(quotient)));
outp(31 downto 24) <= x"AF";
outp(23 downto 0)    <= ROM(to_integer(unsigned(remainder)));
```

Total source code is provided in the attached source files.

Division display scheme on the seven segment display:



i. ALU general structure

Purpose:

To calculate the 7segment result of two given input numbers in 8bits two's complement and a operation.

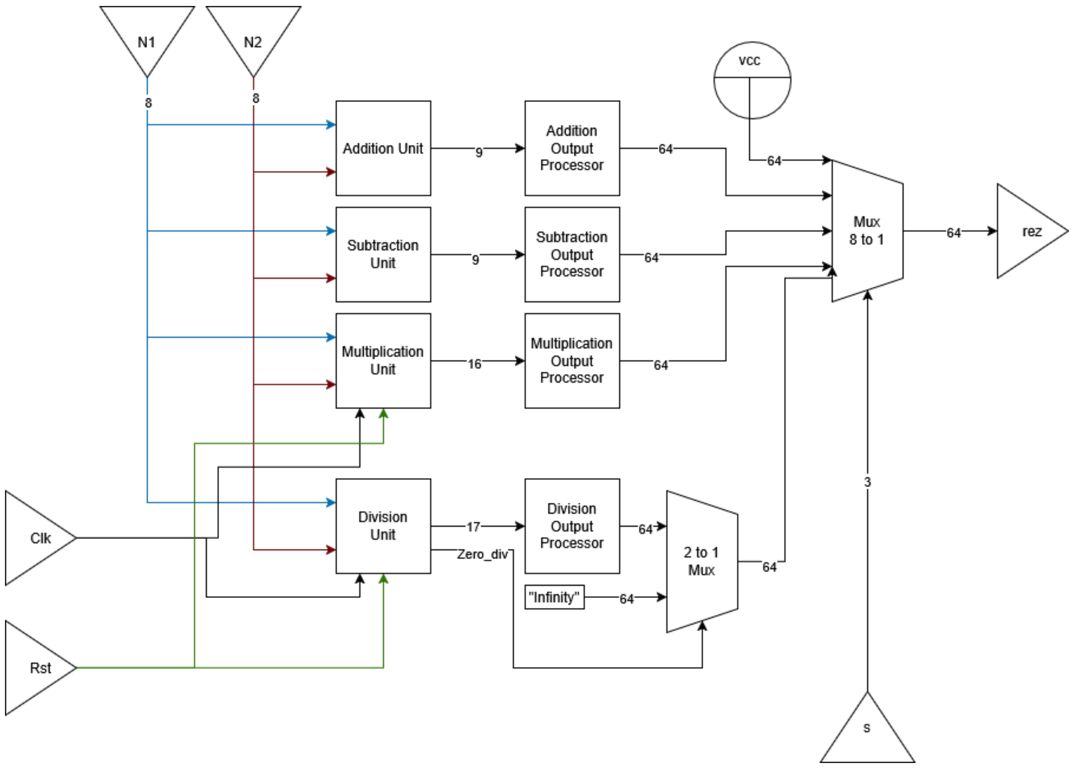
Inputs:

1. No1: first given number in two's complement
2. No2: second given number in two's complement
3. S: operation selection bits
4. Clk: clock signal
5. Rst : reset signal

Outputs:

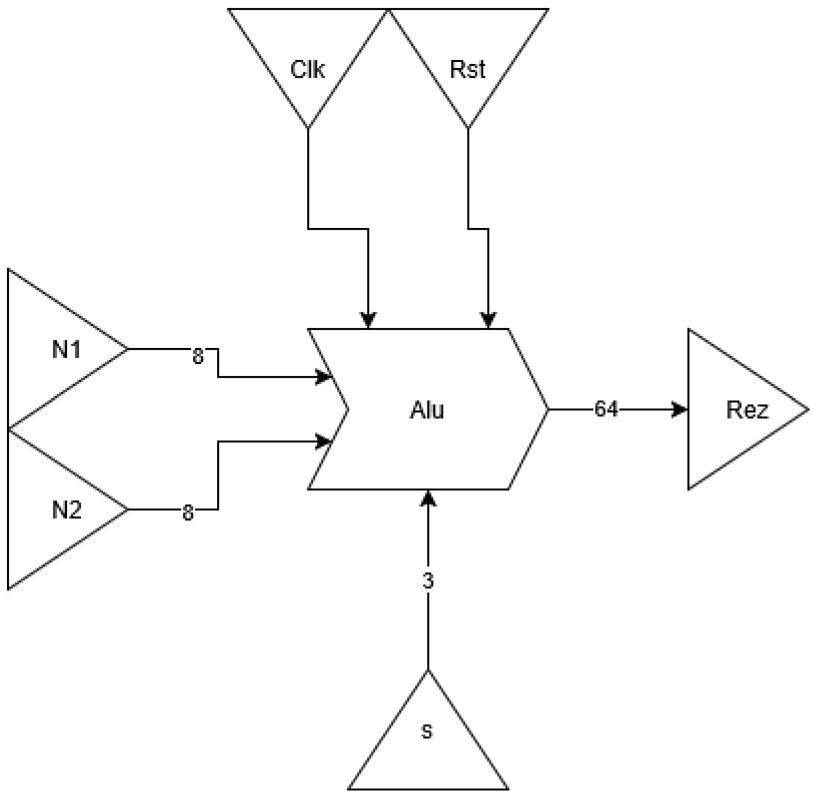
1. Rez: 64 bits which represent the operation and input's result in 7 segment encoding

Logic Scheme:

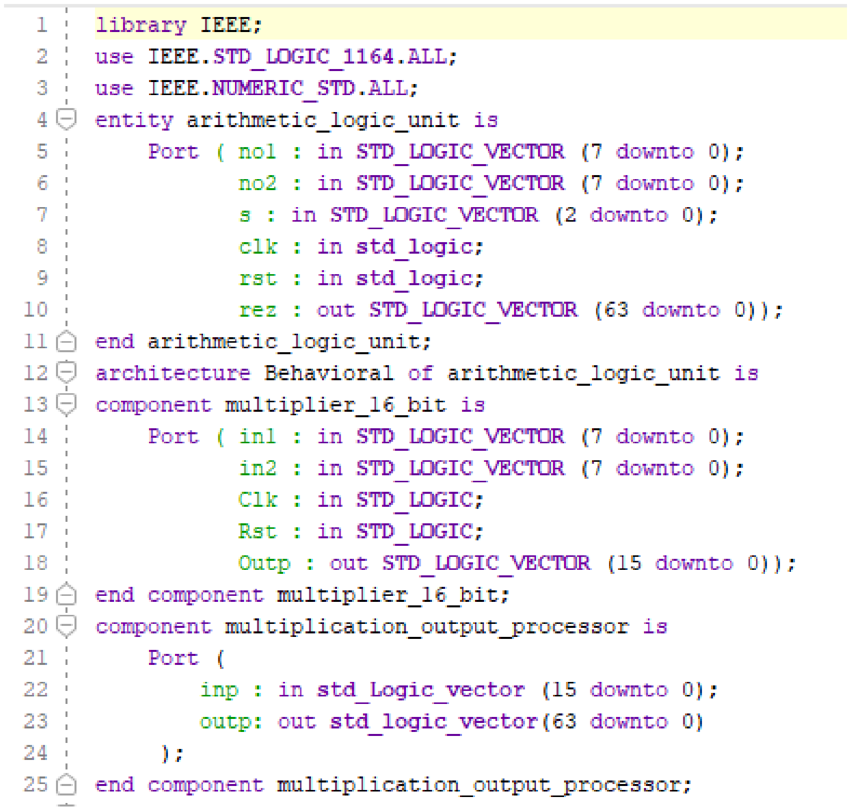


Faculty of Automation and Computer Science
Digital Systems Design Project
Darius-Eric Săsărman, group 30415, Year 2024-2025

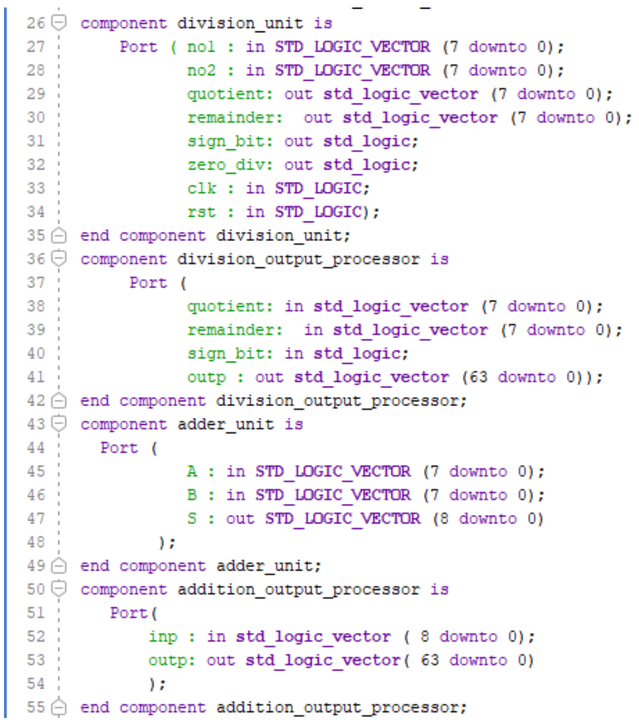
Black box:



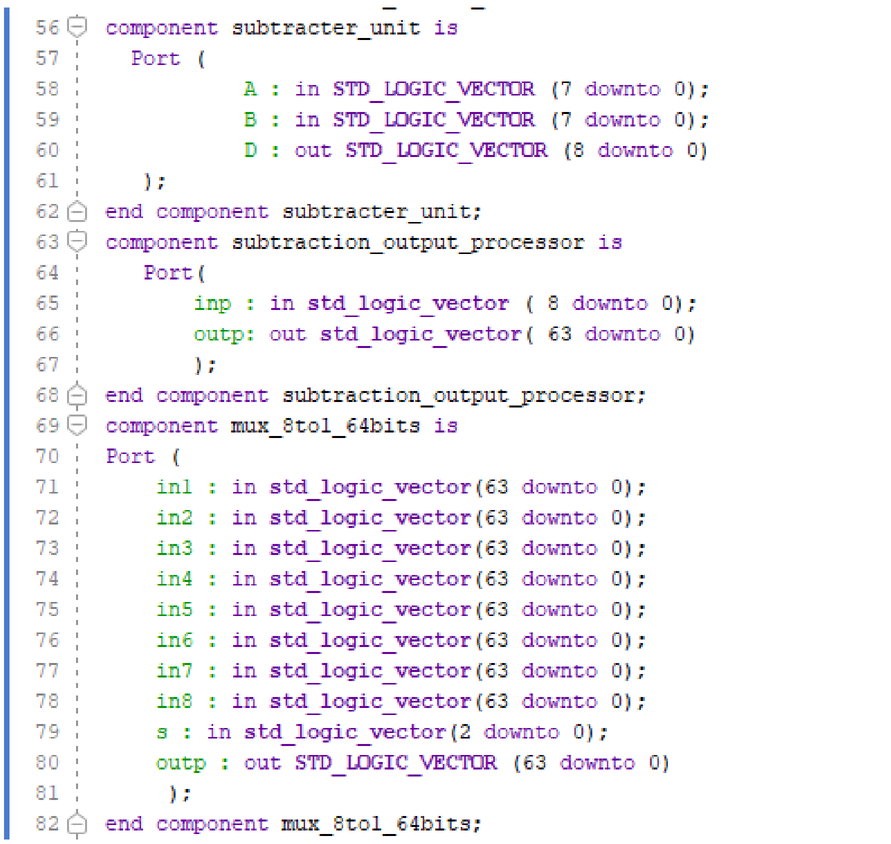
Vhdl Code:



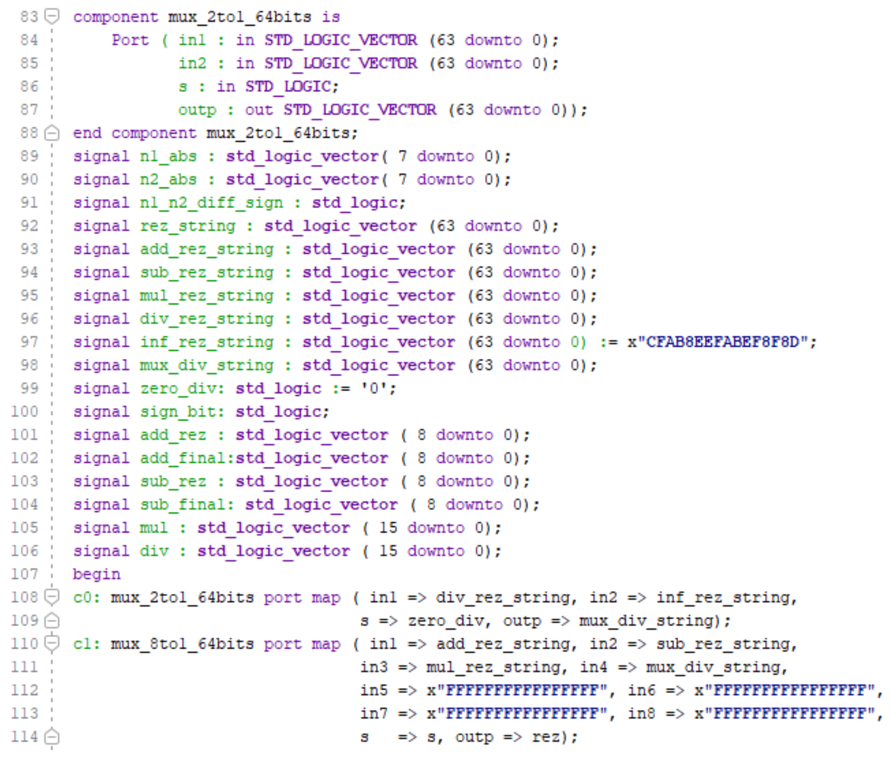
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity arithmetic_logic_unit is
    Port ( no1 : in STD_LOGIC_VECTOR (7 downto 0);
           no2 : in STD_LOGIC_VECTOR (7 downto 0);
           s : in STD_LOGIC_VECTOR (2 downto 0);
           clk : in std_logic;
           rst : in std_logic;
           rez : out STD_LOGIC_VECTOR (63 downto 0));
end arithmetic_logic_unit;
architecture Behavioral of arithmetic_logic_unit is
component multiplier_16_bit is
    Port ( in1 : in STD_LOGIC_VECTOR (7 downto 0);
           in2 : in STD_LOGIC_VECTOR (7 downto 0);
           Clk : in STD_LOGIC;
           Rst : in STD_LOGIC;
           Outp : out STD_LOGIC_VECTOR (15 downto 0));
end component multiplier_16_bit;
component multiplication_output_processor is
    Port (
        inp : in std_Logic_vector (15 downto 0);
        outp: out std_logic_vector(63 downto 0)
      );
end component multiplication_output_processor;
```



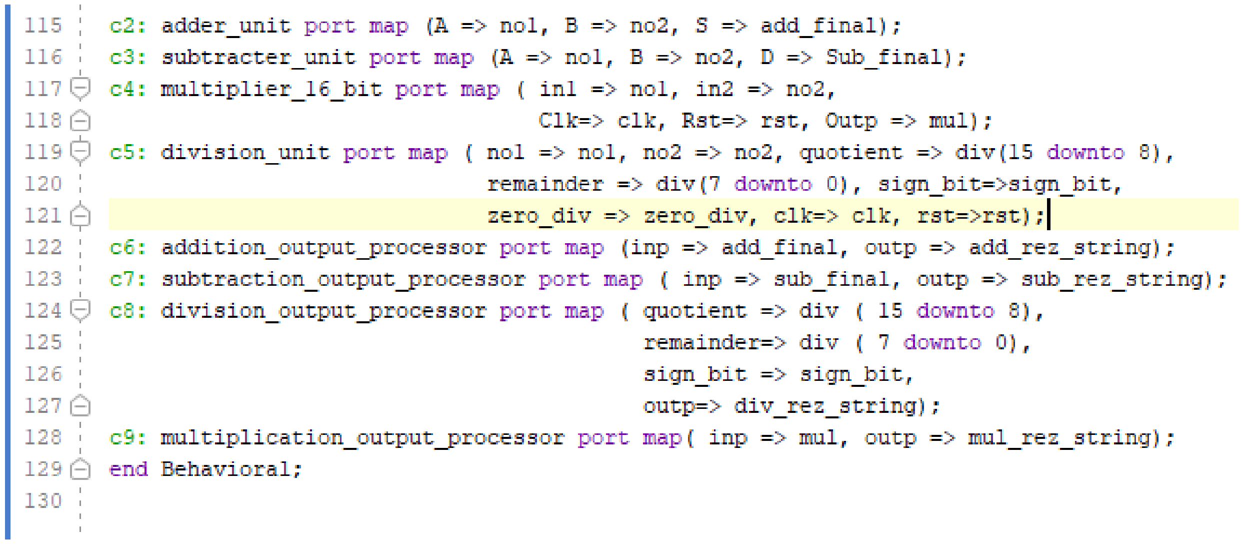
```vhdl
component division_unit is
    Port ( no1 : in STD_LOGIC_VECTOR (7 downto 0);
           no2 : in STD_LOGIC_VECTOR (7 downto 0);
           quotient: out std_logic_vector (7 downto 0);
           remainder:  out std_logic_vector (7 downto 0);
           sign_bit: out std_logic;
           zero_div: out std_logic;
           clk : in STD_LOGIC;
           rst : in STD_LOGIC);
end component division_unit;
component division_output_processor is
     Port (
           quotient: in std_logic_vector (7 downto 0);
           remainder:  in std_logic_vector (7 downto 0);
           sign_bit: in std_logic;
           outp : out std_logic_vector (63 downto 0));
end component division_output_processor;
component adder_unit is
  Port (
           A : in STD_LOGIC_VECTOR (7 downto 0);
           B : in STD_LOGIC_VECTOR (7 downto 0);
           S : out STD_LOGIC_VECTOR (8 downto 0)
        );
end component adder_unit;
component addition_output_processor is
   Port(
       inp : in std_logic_vector ( 8 downto 0);
       outp: out std_logic_vector( 63 downto 0)
       );
end component addition_output_processor;
```



```vhdl
component subtracter_unit is
  Port (
            A : in STD_LOGIC_VECTOR (7 downto 0);
            B : in STD_LOGIC_VECTOR (7 downto 0);
            D : out STD_LOGIC_VECTOR (8 downto 0)
   );
end component subtracter_unit;
component subtraction_output_processor is
   Port(
       inp : in std_logic_vector ( 8 downto 0);
       outp: out std_logic_vector( 63 downto 0)
       );
end component subtraction_output_processor;
component mux_8to1_64bits is
Port (
    in1 : in std_logic_vector(63 downto 0);
    in2 : in std_logic_vector(63 downto 0);
    in3 : in std_logic_vector(63 downto 0);
    in4 : in std_logic_vector(63 downto 0);
    in5 : in std_logic_vector(63 downto 0);
    in6 : in std_logic_vector(63 downto 0);
    in7 : in std_logic_vector(63 downto 0);
    in8 : in std_logic_vector(63 downto 0);
    s : in std_logic_vector(2 downto 0);
    outp : out STD_LOGIC_VECTOR (63 downto 0)
     );
end component mux_8to1_64bits;
```



```vhdl
component mux_2to1_64bits is
    Port ( in1 : in STD_LOGIC_VECTOR (63 downto 0);
           in2 : in STD_LOGIC_VECTOR (63 downto 0);
           s : in STD_LOGIC;
           outp : out STD_LOGIC_VECTOR (63 downto 0));
end component mux_2to1_64bits;
signal n1_abs : std_logic_vector( 7 downto 0);
signal n2_abs : std_logic_vector( 7 downto 0);
signal n1_n2_diff_sign : std_logic;
signal rez_string : std_logic_vector (63 downto 0);
signal add_rez_string : std_logic_vector (63 downto 0);
signal sub_rez_string : std_logic_vector (63 downto 0);
signal mul_rez_string : std_logic_vector (63 downto 0);
signal div_rez_string : std_logic_vector (63 downto 0);
signal inf_rez_string : std_logic_vector (63 downto 0) := x"CFAB8EEFABEF8F8D";
signal mux_div_string : std_logic_vector (63 downto 0);
signal zero_div: std_logic := '0';
signal sign_bit: std_logic;
signal add_rez : std_logic_vector ( 8 downto 0);
signal add_final:std_logic_vector ( 8 downto 0);
signal sub_rez : std_logic_vector ( 8 downto 0);
signal sub_final: std_logic_vector ( 8 downto 0);
signal mul : std_logic_vector ( 15 downto 0);
signal div : std_logic_vector ( 15 downto 0);
begin
c0: mux_2to1_64bits port map ( in1 => div_rez_string, in2 => inf_rez_string,
                               s => zero_div, outp => mux_div_string);
c1: mux_8to1_64bits port map ( in1 => add_rez_string, in2 => sub_rez_string,
                               in3 => mul_rez_string, in4 => mux_div_string,
                               in5 => x"FFFFFFFFFFFFFFFF", in6 => x"FFFFFFFFFFFFFFFF",
                               in7 => x"FFFFFFFFFFFFFFFF", in8 => x"FFFFFFFFFFFFFFFF",
                               s   => s, outp => rez);
```



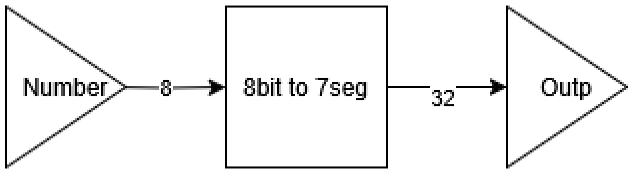
```
c2: adder_unit port map (A => no1, B => no2, S => add_final);
c3: subtracter_unit port map (A => no1, B => no2, D => Sub_final);
c4: multiplier_16_bit port map ( in1 => no1, in2 => no2,
                                 Clk=> clk, Rst=> rst, Outp => mul);
c5: division_unit port map ( no1 => no1, no2 => no2, quotient => div(15 downto 8),
                             remainder => div(7 downto 0), sign_bit=>sign_bit,
                             zero_div => zero_div, clk=> clk, rst=>rst);
c6: addition_output_processor port map (inp => add_final, outp => add_rez_string);
c7: subtraction_output_processor port map ( inp => sub_final, outp => sub_rez_string);
c8: division_output_processor port map ( quotient => div ( 15 downto 8),
                                         remainder=> div ( 7 downto 0),
                                         sign_bit => sign_bit,
                                         outp=> div_rez_string);
c9: multiplication_output_processor port map( inp => mul, outp => mul_rez_string);
end Behavioral;
```

3. Display Unit

a. Eight-bit two’s complement to seven segment display ROM

Same logic as the “Output Processor”. Takes in a number in 8-bit 2’s complement and outputs the corresponding bitstring.

Black box:



b. Three-bit counter with clock divider

Purpose:

To alter between digits and allow the seven-segment displayer. Takes in the clock signal and every 100000 clock cycles goes to the next state (0 to 7)

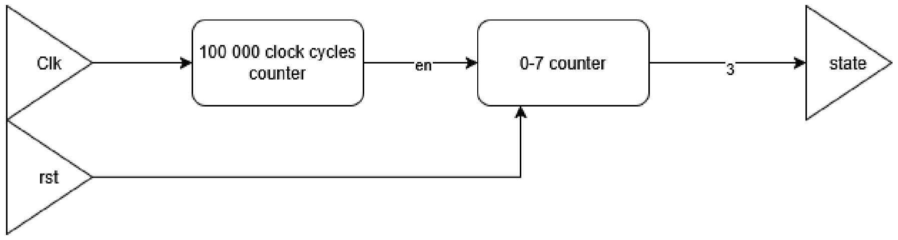
Inputs:

1. Clk: clock signal
2. Rst: reset signal

Outputs:

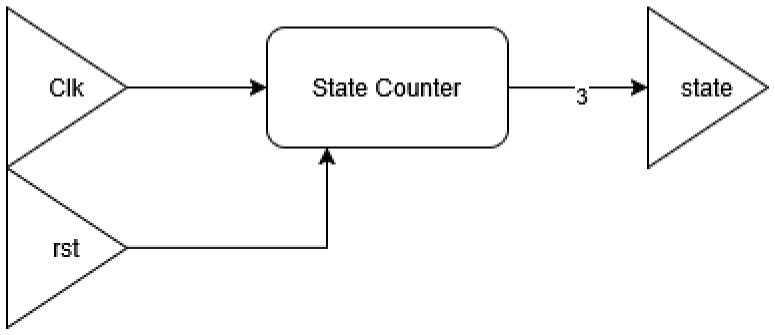
1. Outp : 3-bit signal which signals the state to the 7 segment displayer.

Logic Scheme:

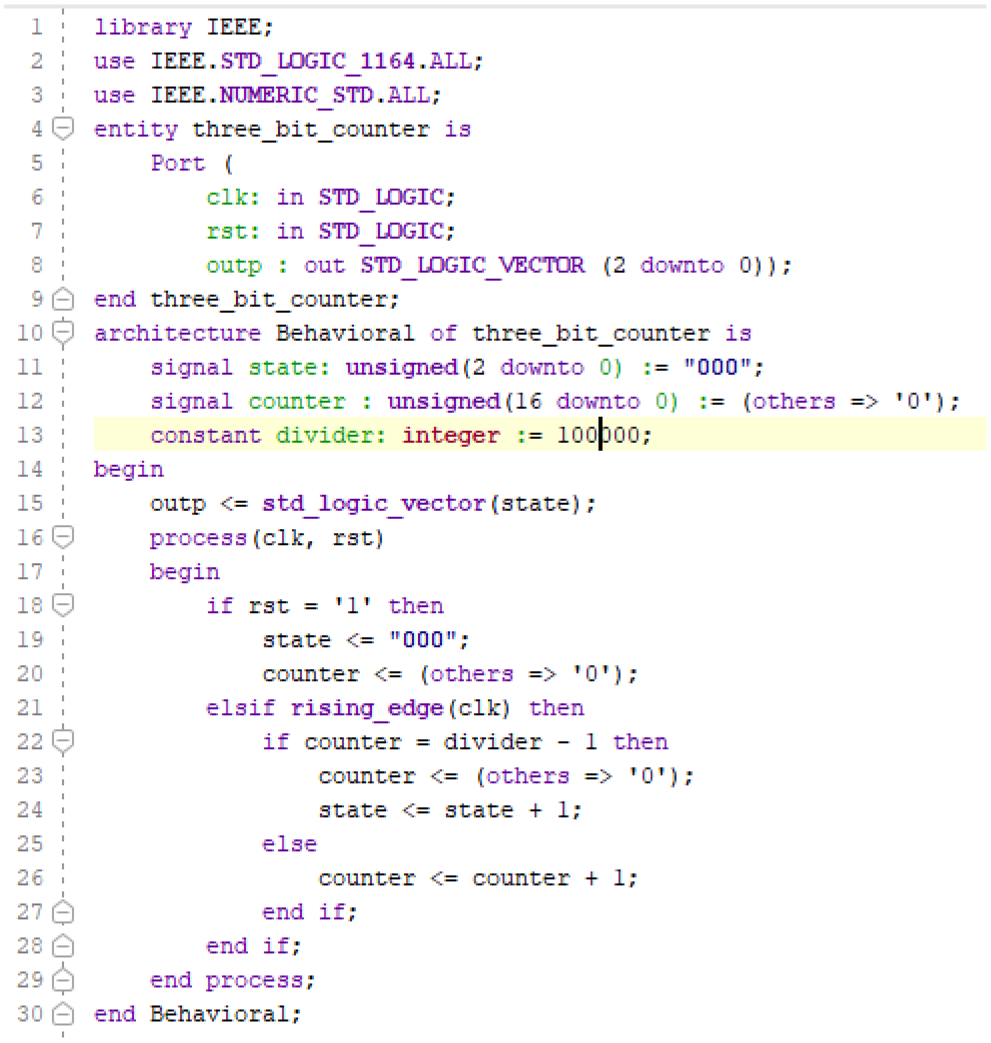


Faculty of Automation and Computer Science
Digital Systems Design Project
Darius-Eric Săsărman, group 30415, Year 2024-2025

Black-box:



VHDL Code:



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity three_bit_counter is
    Port (
        clk: in STD_LOGIC;
        rst: in STD_LOGIC;
        outp : out STD_LOGIC_VECTOR (2 downto 0));
end three_bit_counter;
architecture Behavioral of three_bit_counter is
    signal state: unsigned(2 downto 0) := "000";
    signal counter : unsigned(16 downto 0) := (others => '0');
    constant divider: integer := 100000;
begin
    outp <= std_logic_vector(state);
    process(clk, rst)
    begin
        if rst = '1' then
            state <= "000";
            counter <= (others => '0');
        elsif rising_edge(clk) then
            if counter = divider - 1 then
                counter <= (others => '0');
                state <= state + 1;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;
end Behavioral;
```

c.Seven segment displayer

i. Three-bit decoder

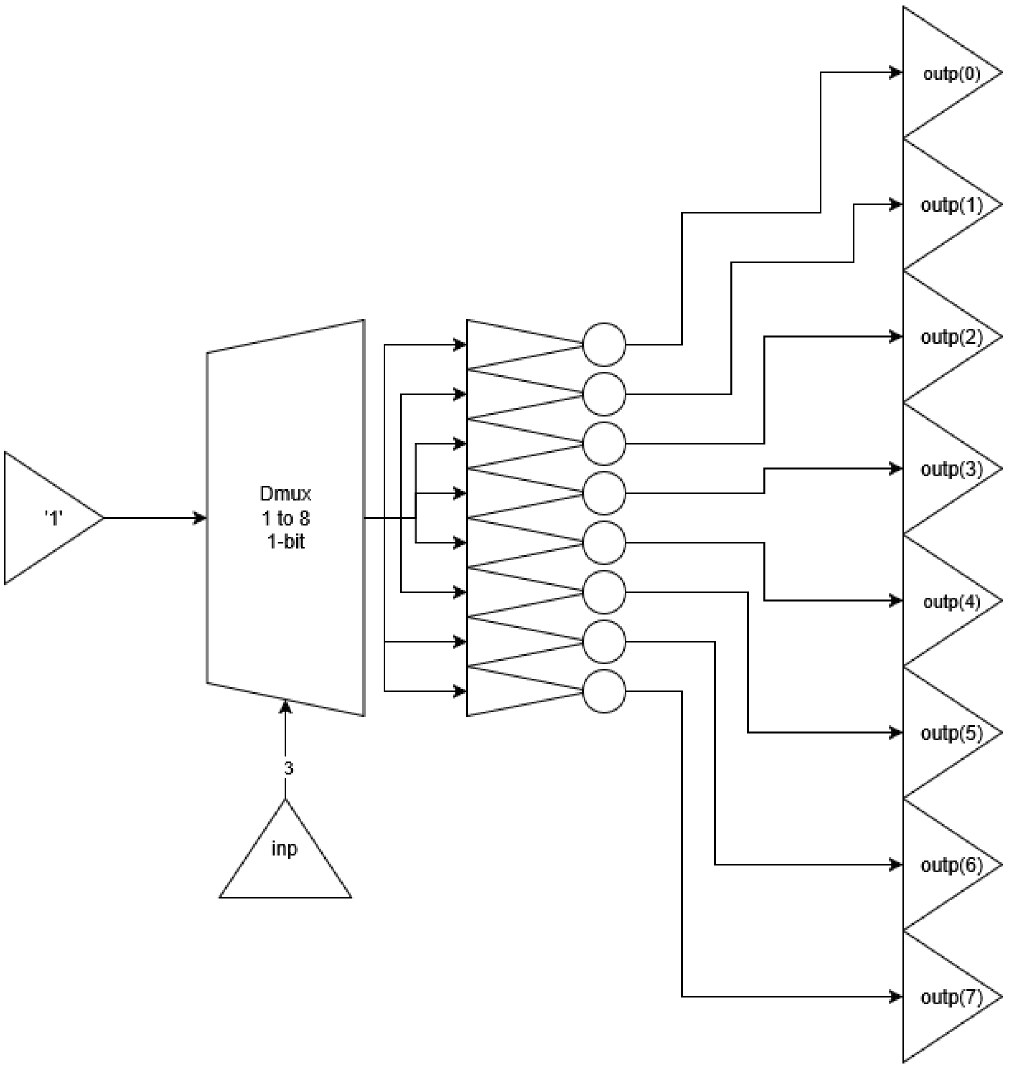
Purpose:

To tell the Seven-segment display what digit position to display the selected digit onto.

Inputs: Inp : 3 bits

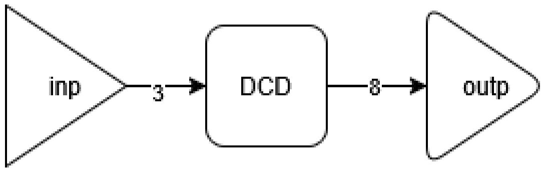
Outputs: Outp: 8 bits, active-low when corresponding combination is inserted.

Logic scheme:

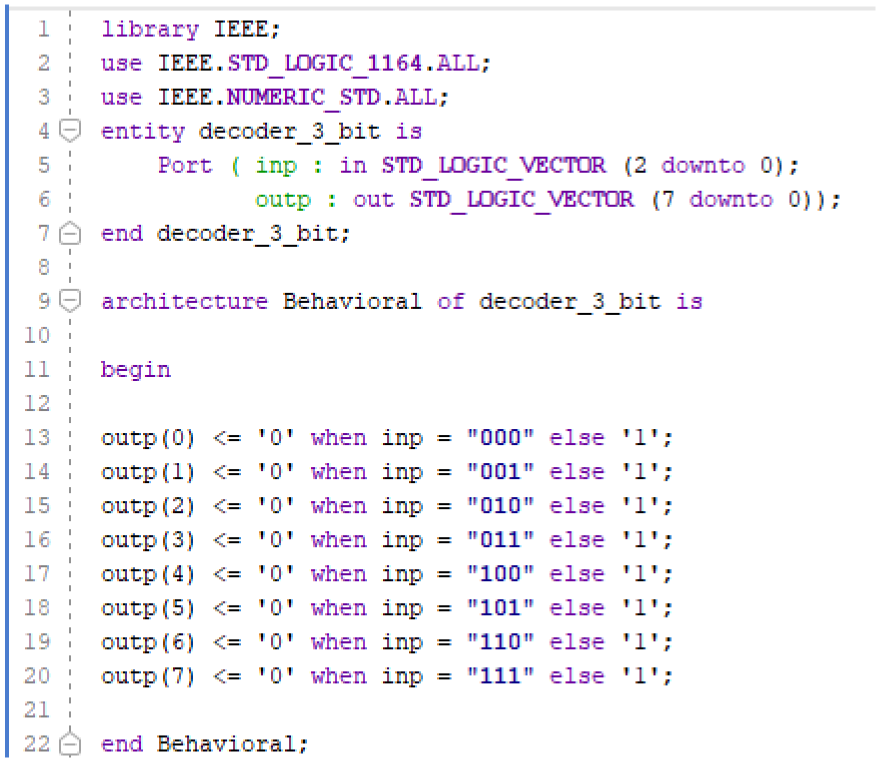


Faculty of Automation and Computer Science
Digital Systems Design Project
Darius-Eric Săsărman, group 30415, Year 2024-2025

Black box:



VHDL Code:



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity decoder_3_bit is
    Port ( inp : in STD_LOGIC_VECTOR (2 downto 0);
           outp : out STD_LOGIC_VECTOR (7 downto 0));
end decoder_3_bit;

architecture Behavioral of decoder_3_bit is

begin

outp(0) <= '0' when inp = "000" else '1';
outp(1) <= '0' when inp = "001" else '1';
outp(2) <= '0' when inp = "010" else '1';
outp(3) <= '0' when inp = "011" else '1';
outp(4) <= '0' when inp = "100" else '1';
outp(5) <= '0' when inp = "101" else '1';
outp(6) <= '0' when inp = "110" else '1';
outp(7) <= '0' when inp = "111" else '1';

end Behavioral;
```

ii. 8 to 1 Mux on 8 bits

Purpose :

Breaks down the 64-bit 7-segment info into separate digits for 'on-screen' writing

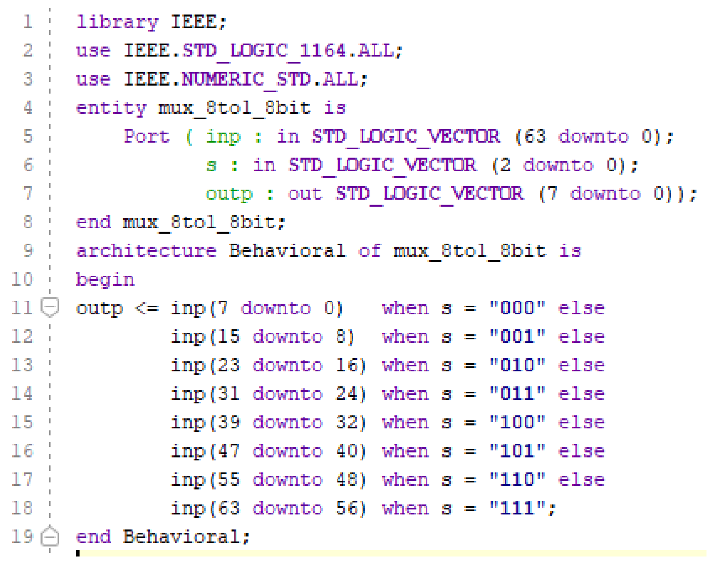
Inputs:

1. Inp : the 64 bits that represent the choices
2. S : 3 bits that represent the choice bits

Outputs:

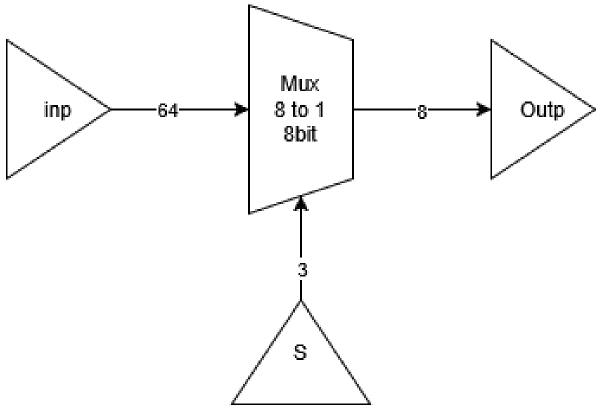
1. Outp: the selected 8 bits

VHDL code:



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity mux_8to1_8bit is
    Port ( inp : in STD_LOGIC_VECTOR (63 downto 0);
           s : in STD_LOGIC_VECTOR (2 downto 0);
           outp : out STD_LOGIC_VECTOR (7 downto 0));
end mux_8to1_8bit;
architecture Behavioral of mux_8to1_8bit is
begin
outp <= inp(7 downto 0)   when s = "000" else
        inp(15 downto 8)  when s = "001" else
        inp(23 downto 16) when s = "010" else
        inp(31 downto 24) when s = "011" else
        inp(39 downto 32) when s = "100" else
        inp(47 downto 40) when s = "101" else
        inp(55 downto 48) when s = "110" else
        inp(63 downto 56) when s = "111";
end Behavioral;
```

Black box:



iii.Seven Segment Displayer General Structure

Purpose:

To select and transfer the given 7-segment information required to be displayed.

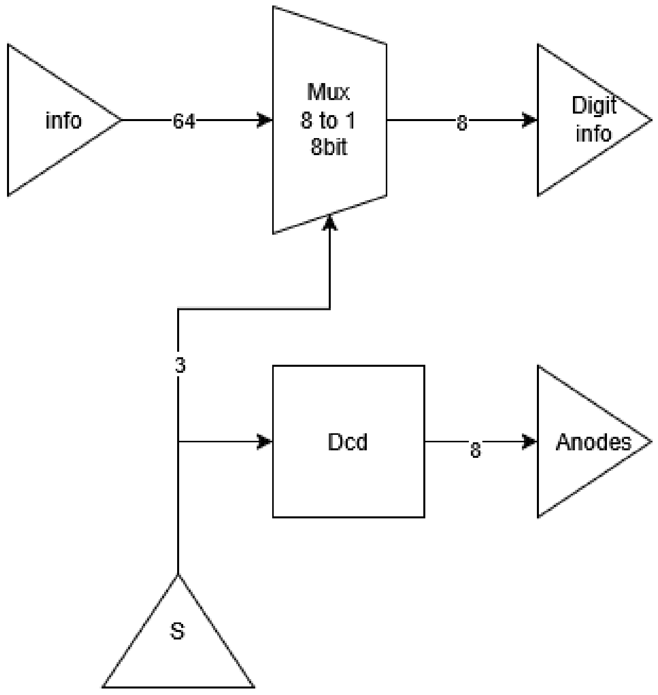
Inputs:

1. Info : the total 64-string to be displayed on the 7-segment display
2. State: 3-bits which decide the output

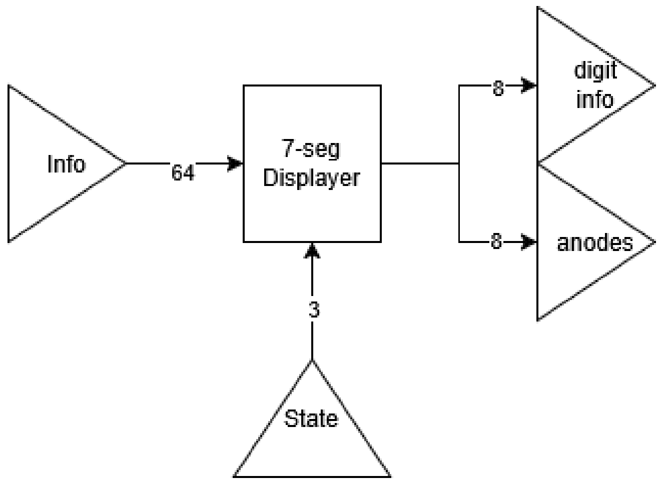
Outputs:

1. Digit_info : that's the multiplexed digit info
2. Anodes : what's the chosen digit to write to

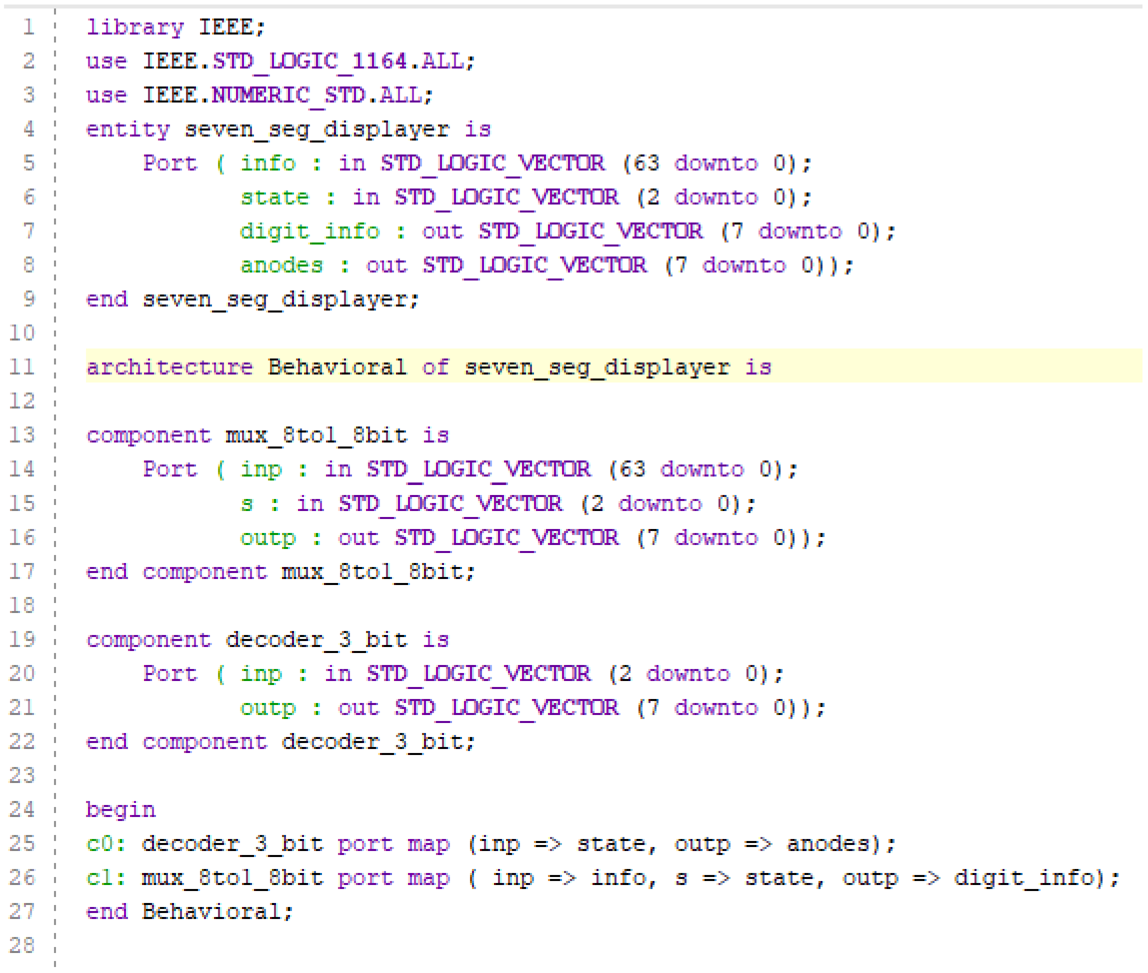
Logic Scheme:



Black box:



Vhdl Code:



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity seven_seg_displayer is
    Port ( info : in STD_LOGIC_VECTOR (63 downto 0);
           state : in STD_LOGIC_VECTOR (2 downto 0);
           digit_info : out STD_LOGIC_VECTOR (7 downto 0);
           anodes : out STD_LOGIC_VECTOR (7 downto 0));
end seven_seg_displayer;

architecture Behavioral of seven_seg_displayer is

component mux_8to1_8bit is
    Port ( inp : in STD_LOGIC_VECTOR (63 downto 0);
           s : in STD_LOGIC_VECTOR (2 downto 0);
           outp : out STD_LOGIC_VECTOR (7 downto 0));
end component mux_8to1_8bit;

component decoder_3_bit is
    Port ( inp : in STD_LOGIC_VECTOR (2 downto 0);
           outp : out STD_LOGIC_VECTOR (7 downto 0));
end component decoder_3_bit;

begin
c0: decoder_3_bit port map (inp => state, outp => anodes);
c1: mux_8to1_8bit port map ( inp => info, s => state, outp => digit_info);
end Behavioral;
```

d. Display Unit general Structure

Purpose:

To take the input numbers and the ALU Result and show those on the 7-segment display.

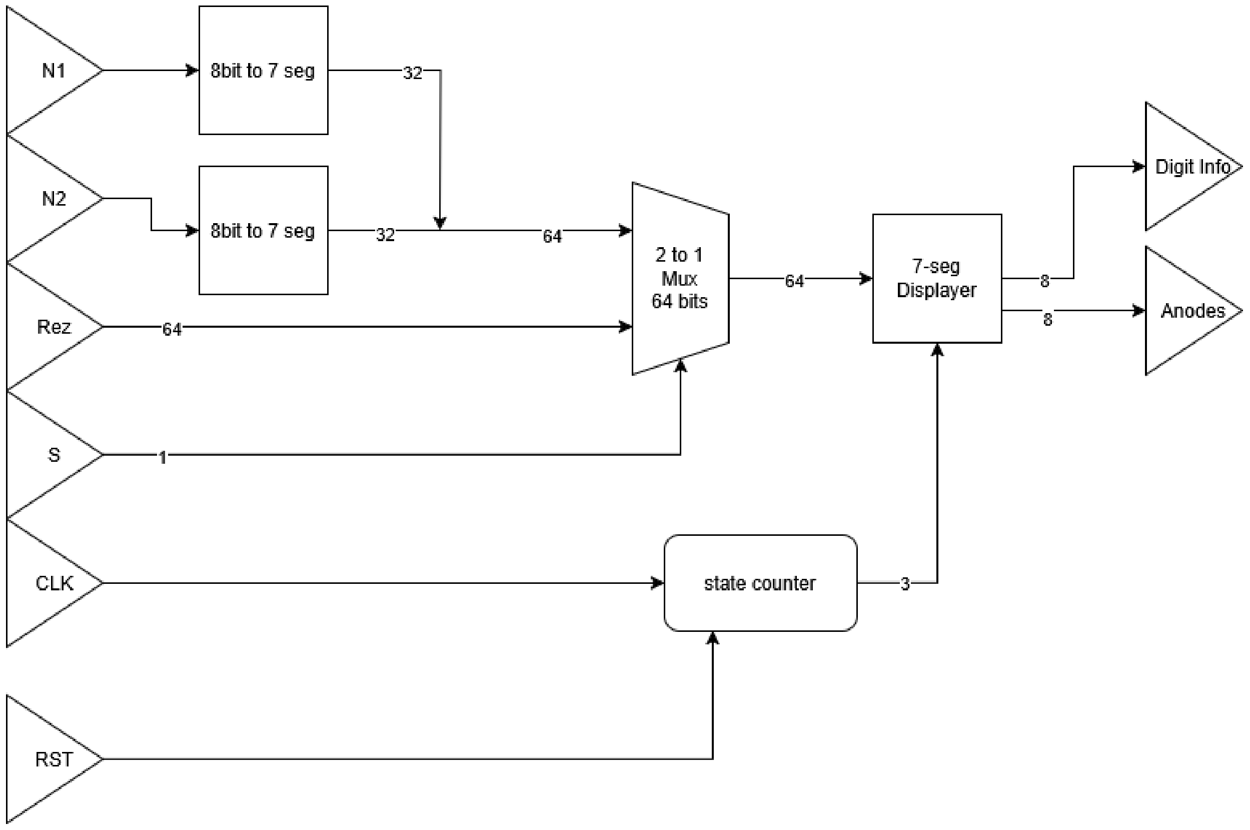
Inputs:

1. No1 : the first inserted 8-bit 2’s complement number
2. No2 : the second inserted 8-bit 2's complement number
3. Rez : the ALU result
4. Clk : clock signal
5. Rst: reset signal
6. S : picks between displaying input numbers and result

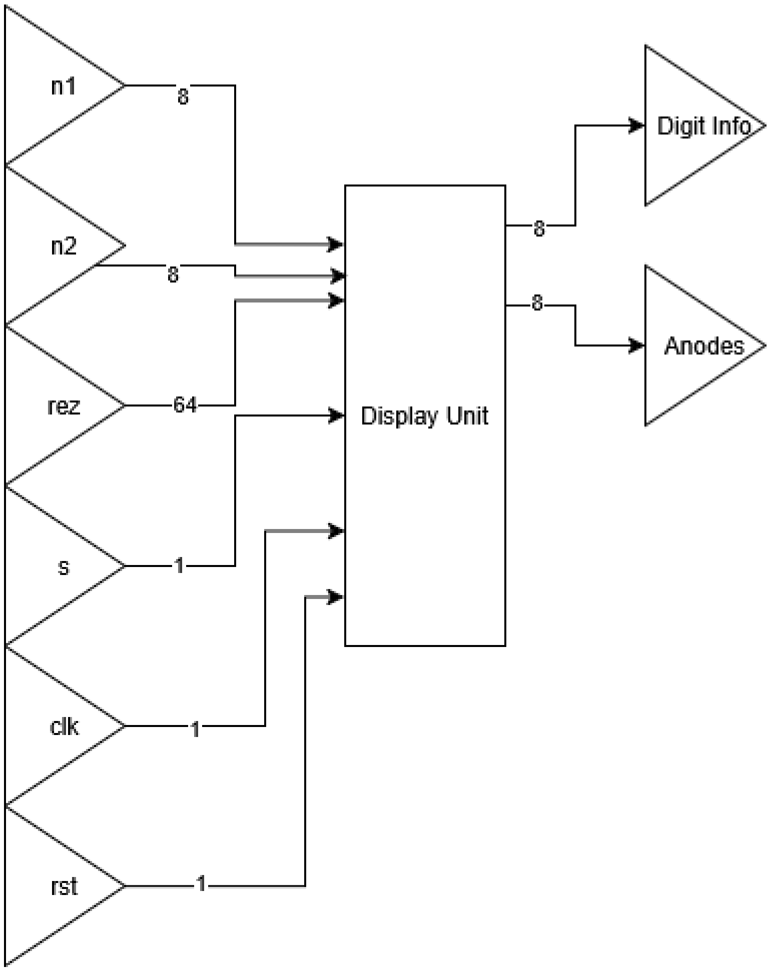
Outputs:

1. Digit_info : the 7-segment information which choses which segment lights up
2. Anode_info : 8-bits which pick the target 7-segment digit

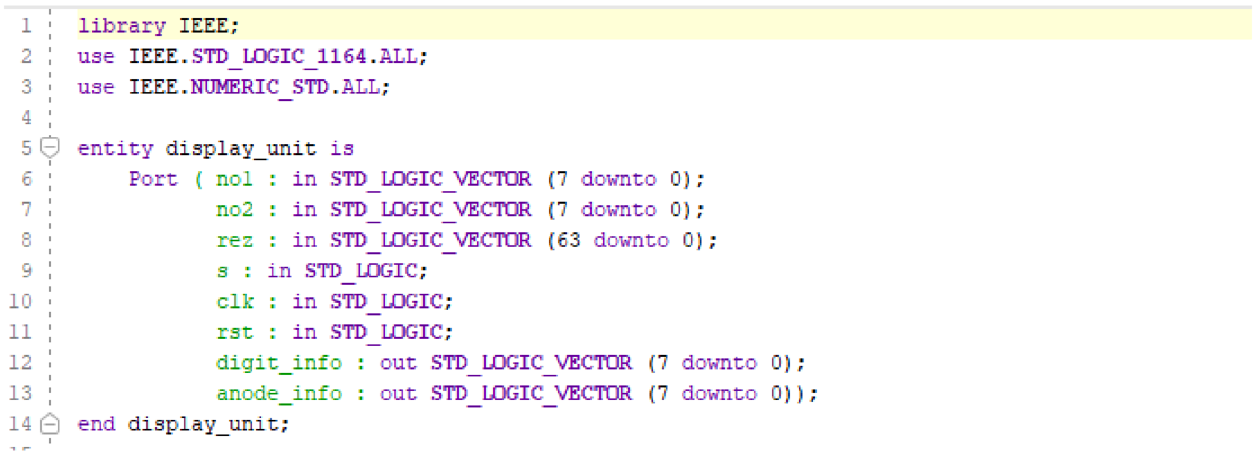
Logic scheme:



Black box:

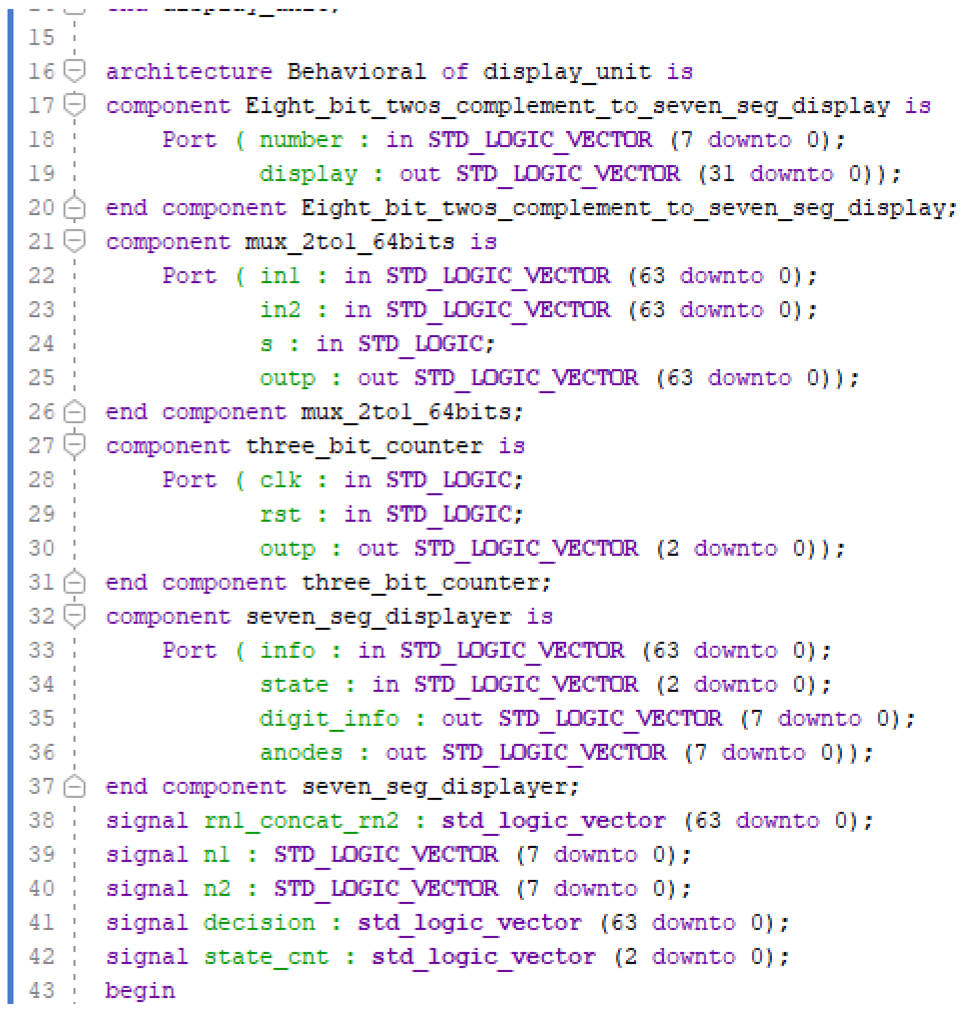


VHDL Code:

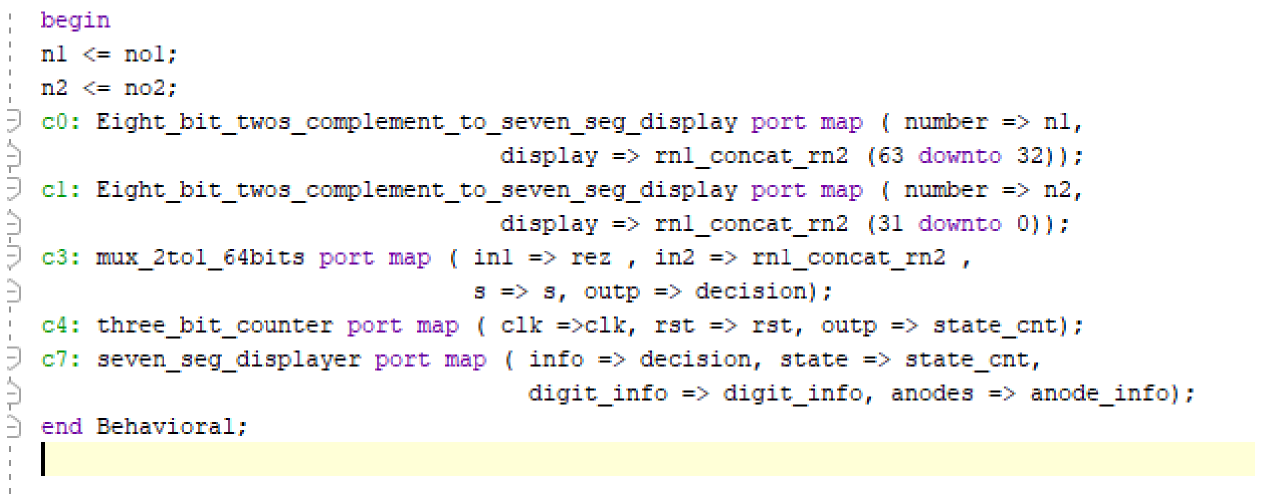


```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity display_unit is
    Port ( no1 : in STD_LOGIC_VECTOR (7 downto 0);
           no2 : in STD_LOGIC_VECTOR (7 downto 0);
           rez : in STD_LOGIC_VECTOR (63 downto 0);
           s : in STD_LOGIC;
           clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           digit_info : out STD_LOGIC_VECTOR (7 downto 0);
           anode_info : out STD_LOGIC_VECTOR (7 downto 0));
end display_unit;
```



```
architecture Behavioral of display_unit is
component Eight_bit_twos_complement_to_seven_seg_display is
    Port ( number : in STD_LOGIC_VECTOR (7 downto 0);
           display : out STD_LOGIC_VECTOR (31 downto 0));
end component Eight_bit_twos_complement_to_seven_seg_display;
component mux_2to1_64bits is
    Port ( in1 : in STD_LOGIC_VECTOR (63 downto 0);
           in2 : in STD_LOGIC_VECTOR (63 downto 0);
           s : in STD_LOGIC;
           outp : out STD_LOGIC_VECTOR (63 downto 0));
end component mux_2to1_64bits;
component three_bit_counter is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           outp : out STD_LOGIC_VECTOR (2 downto 0));
end component three_bit_counter;
component seven_seg_displayer is
    Port ( info : in STD_LOGIC_VECTOR (63 downto 0);
           state : in STD_LOGIC_VECTOR (2 downto 0);
           digit_info : out STD_LOGIC_VECTOR (7 downto 0);
           anodes : out STD_LOGIC_VECTOR (7 downto 0));
end component seven_seg_displayer;
signal rn1_concat_rn2 : std_logic_vector (63 downto 0);
signal n1 : STD_LOGIC_VECTOR (7 downto 0);
signal n2 : STD_LOGIC_VECTOR (7 downto 0);
signal decision : std_logic_vector (63 downto 0);
signal state_cnt : std_logic_vector (2 downto 0);
begin
```



```
begin
n1 <= no1;
n2 <= no2;
c0: Eight_bit_twos_complement_to_seven_seg_display port map ( number => n1,
                                display => rn1_concat_rn2 (63 downto 32));
c1: Eight_bit_twos_complement_to_seven_seg_display port map ( number => n2,
                                display => rn1_concat_rn2 (31 downto 0));
c3: mux_2to1_64bits port map ( in1 => rez , in2 => rn1_concat_rn2 ,
                              s => s, outp => decision);
c4: three_bit_counter port map ( clk =>clk, rst => rst, outp => state_cnt);
c7: seven_seg_displayer port map ( info => decision, state => state_cnt,
                                   digit_info => digit_info, anodes => anode_info);
end Behavioral;
```

4.Execution Unit General Structure

Purpose:
To execute all computation required by the control unit.

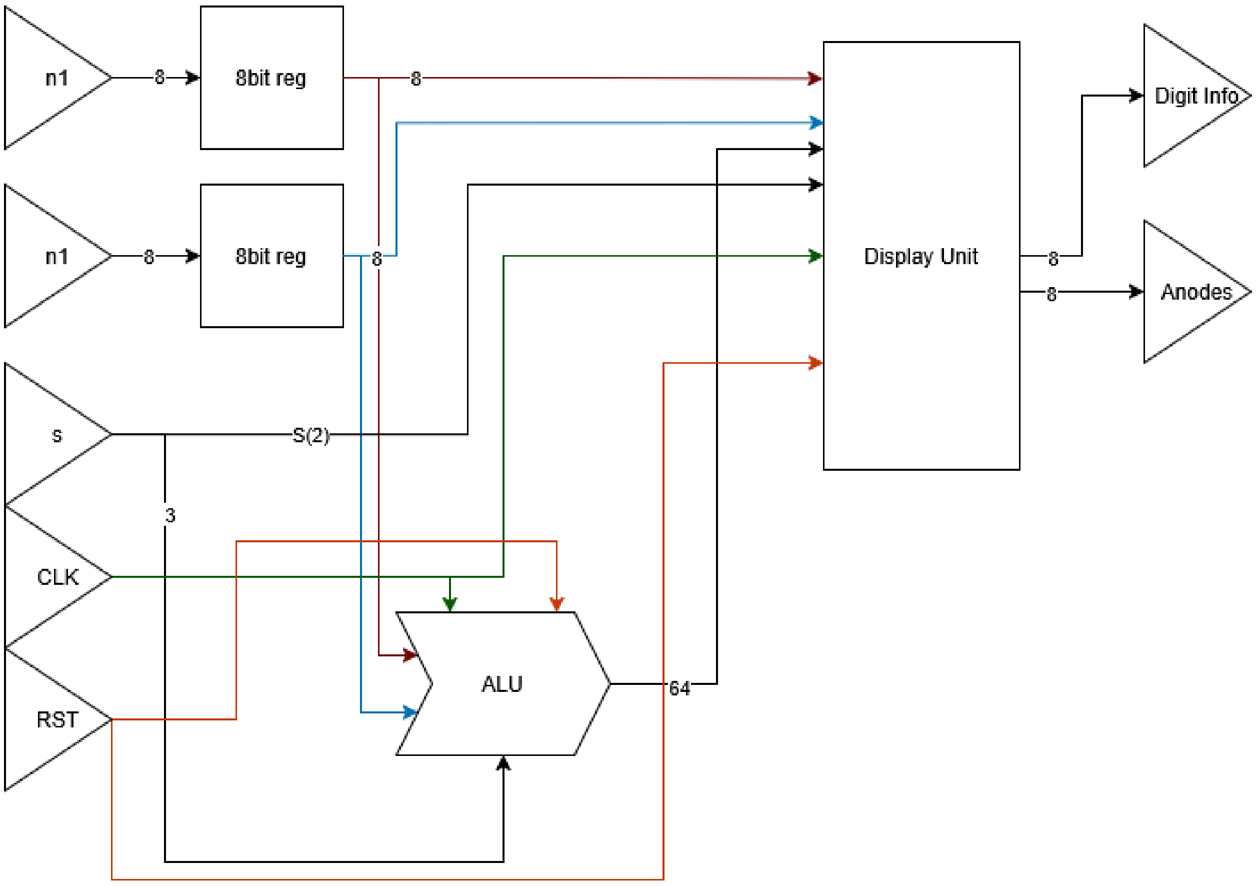
Inputs:

1. N1: first input number in 8bits two's complement
2. N2 : second input number in 8bits two's complement
3. S : three bit choice signal

Outputs:

1. Digit_info : same as Display Unit
2. Anode_info: same as Display Unit

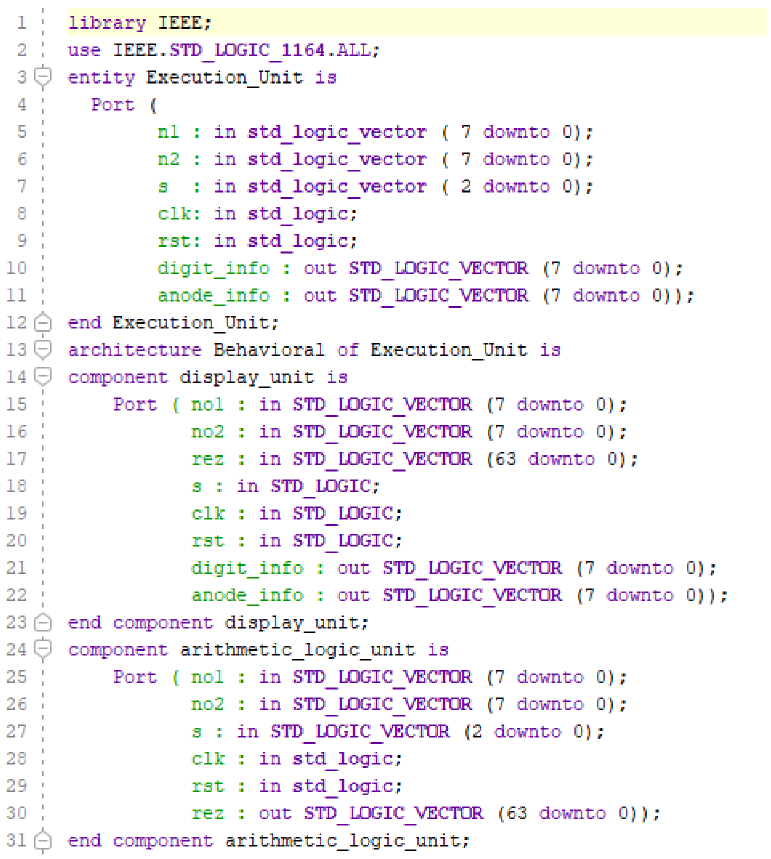
Logic
scheme:



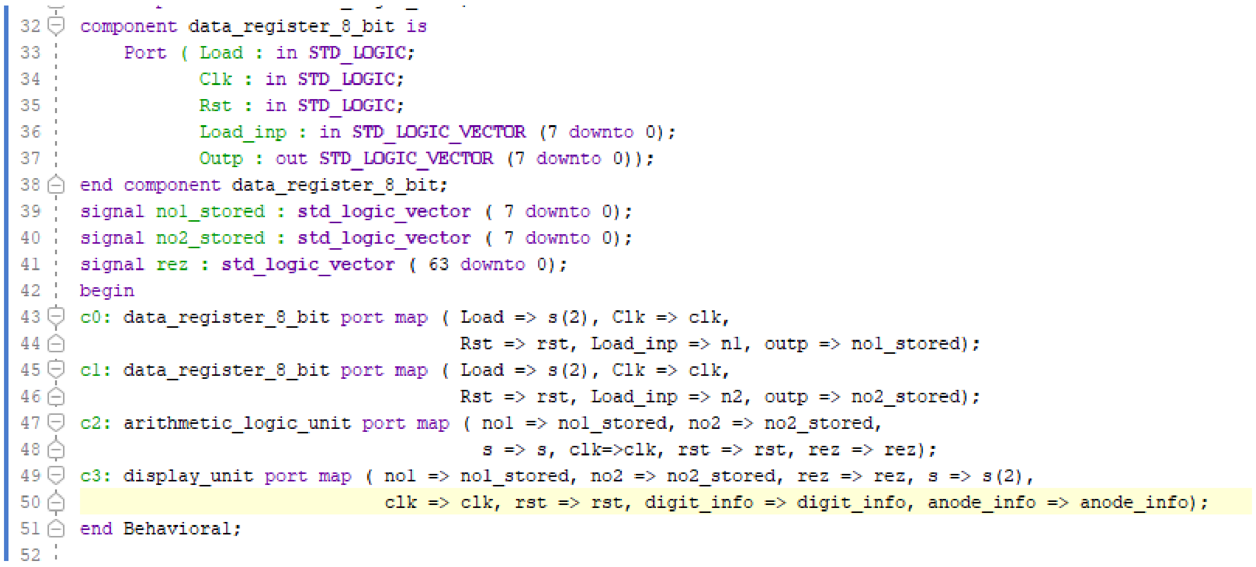
Faculty of Automation and Computer Science
Digital Systems Design Project
Darius-Eric Săsărman, group 30415, Year 2024-2025

Black Box : (Shown above in component block diagram)

Vhdl Code:



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Execution_Unit is
  Port (
        n1 : in std_logic_vector ( 7 downto 0);
        n2 : in std_logic_vector ( 7 downto 0);
        s  : in std_logic_vector ( 2 downto 0);
        clk: in std_logic;
        rst: in std_logic;
        digit_info : out STD_LOGIC_VECTOR (7 downto 0);
        anode_info : out STD_LOGIC_VECTOR (7 downto 0));
end Execution_Unit;
architecture Behavioral of Execution_Unit is
component display_unit is
    Port ( no1 : in STD_LOGIC_VECTOR (7 downto 0);
           no2 : in STD_LOGIC_VECTOR (7 downto 0);
           rez : in STD_LOGIC_VECTOR (63 downto 0);
           s : in STD_LOGIC;
           clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           digit_info : out STD_LOGIC_VECTOR (7 downto 0);
           anode_info : out STD_LOGIC_VECTOR (7 downto 0));
end component display_unit;
component arithmetic_logic_unit is
    Port ( no1 : in STD_LOGIC_VECTOR (7 downto 0);
           no2 : in STD_LOGIC_VECTOR (7 downto 0);
           s : in STD_LOGIC_VECTOR (2 downto 0);
           clk : in std_logic;
           rst : in std_logic;
           rez : out STD_LOGIC_VECTOR (63 downto 0));
end component arithmetic_logic_unit;
```



```vhdl
component data_register_8_bit is
    Port ( Load : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Rst : in STD_LOGIC;
           Load_inp : in STD_LOGIC_VECTOR (7 downto 0);
           Outp : out STD_LOGIC_VECTOR (7 downto 0));
end component data_register_8_bit;
signal no1_stored : std_logic_vector ( 7 downto 0);
signal no2_stored : std_logic_vector ( 7 downto 0);
signal rez : std_logic_vector ( 63 downto 0);
begin
c0: data_register_8_bit port map ( Load => s(2), Clk => clk,
                                   Rst => rst, Load_inp => n1, outp => no1_stored);
c1: data_register_8_bit port map ( Load => s(2), Clk => clk,
                                   Rst => rst, Load_inp => n2, outp => no2_stored);
c2: arithmetic_logic_unit port map ( no1 => no1_stored, no2 => no2_stored,
                                     s => s, clk=>clk, rst => rst, rez => rez);
c3: display_unit port map ( no1 => no1_stored, no2 => no2_stored, rez => rez, s => s(2),
                            clk => clk, rst => rst, digit_info => digit_info, anode_info => anode_info);
end Behavioral;
```

Control unit

Button Push Decoder Logic

Here is the decoder logic for the button pushes

Add Button	Subtraction Button	Multiplication Button	Division Button	Result
1	0	0	0	000
0	1	0	0	001
0	0	1	0	010
0	0	0	1	011
Other cases	Other cases	Other Cases	Other Cases	100

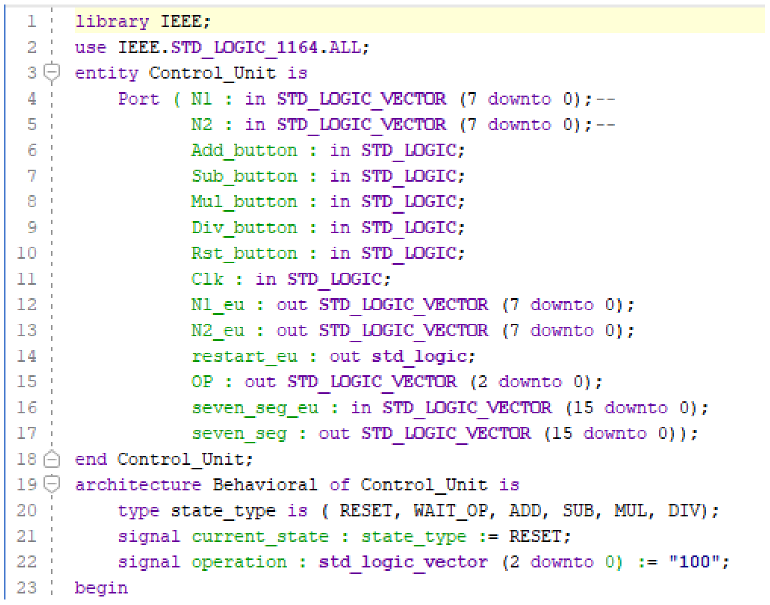
“Result” is directly sent as ‘s’ towards the execution unit.

Purpose:

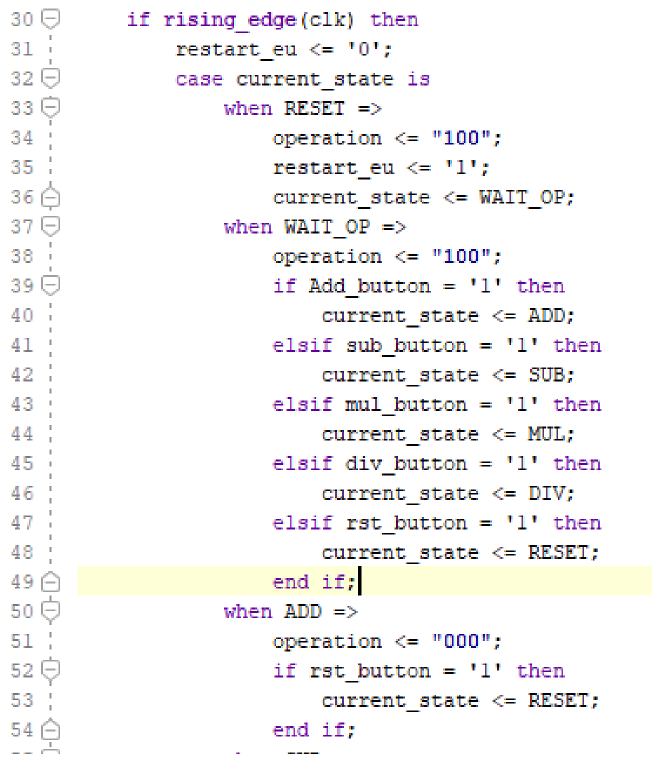
To control the operation of the whole project.

Inputs / Outputs / Black box: Exemplified in the Component block diagram

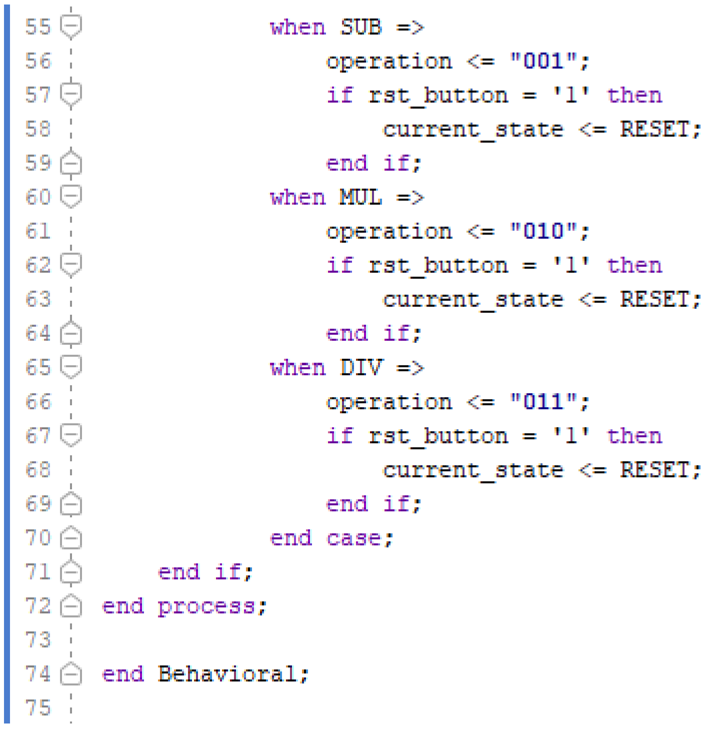
VHDL Code:



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Control_Unit is
    Port ( N1 : in STD_LOGIC_VECTOR (7 downto 0);--
           N2 : in STD_LOGIC_VECTOR (7 downto 0);--
           Add_button : in STD_LOGIC;
           Sub_button : in STD_LOGIC;
           Mul_button : in STD_LOGIC;
           Div_button : in STD_LOGIC;
           Rst_button : in STD_LOGIC;
           Clk : in STD_LOGIC;
           N1_eu : out STD_LOGIC_VECTOR (7 downto 0);
           N2_eu : out STD_LOGIC_VECTOR (7 downto 0);
           restart_eu : out std_logic;
           OP : out STD_LOGIC_VECTOR (2 downto 0);
           seven_seg_eu : in STD_LOGIC_VECTOR (15 downto 0);
           seven_seg : out STD_LOGIC_VECTOR (15 downto 0));
end Control_Unit;
architecture Behavioral of Control_Unit is
    type state_type is ( RESET, WAIT_OP, ADD, SUB, MUL, DIV);
    signal current_state : state_type := RESET;
    signal operation : std_logic_vector (2 downto 0) := "100";
begin
```



```
    if rising_edge(clk) then
        restart_eu <= '0';
        case current_state is
            when RESET =>
                operation <= "100";
                restart_eu <= '1';
                current_state <= WAIT_OP;
            when WAIT_OP =>
                operation <= "100";
                if Add_button = '1' then
                    current_state <= ADD;
                elsif sub_button = '1' then
                    current_state <= SUB;
                elsif mul_button = '1' then
                    current_state <= MUL;
                elsif div_button = '1' then
                    current_state <= DIV;
                elsif rst_button = '1' then
                    current_state <= RESET;
                end if;
            when ADD =>
                operation <= "000";
                if rst_button = '1' then
                    current_state <= RESET;
                end if;
```



```
            when SUB =>
                operation <= "001";
                if rst_button = '1' then
                    current_state <= RESET;
                end if;
            when MUL =>
                operation <= "010";
                if rst_button = '1' then
                    current_state <= RESET;
                end if;
            when DIV =>
                operation <= "011";
                if rst_button = '1' then
                    current_state <= RESET;
                end if;
        end case;
    end if;
end process;

end Behavioral;
```

State Diagram: Has been shown in the introduction.