Problem 1

Design and simulate **8-bit ripple carry adder** circuit using 8-full adders. Full adders can be behavioral or structural, but clearly state what you used.

Code:

Rippleadder.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;



entity rippleadder is

port ( a : in std_logic_vector(7 downto 0);

 b : in std_logic_vector(7 downto 0);

 cin : in std_logic;

 sum : out std_logic_vector(7 downto 0);

 cout : out std_logic);

end rippleadder;



architecture structural of rippleadder is



component full_adder_con is

  port (

    a, b, c : in  std_logic;            -- inputs

    sum, ca : out std_logic);           -- sum & carry

end component;

signal carry : std_logic_vector(6 downto 0);
```

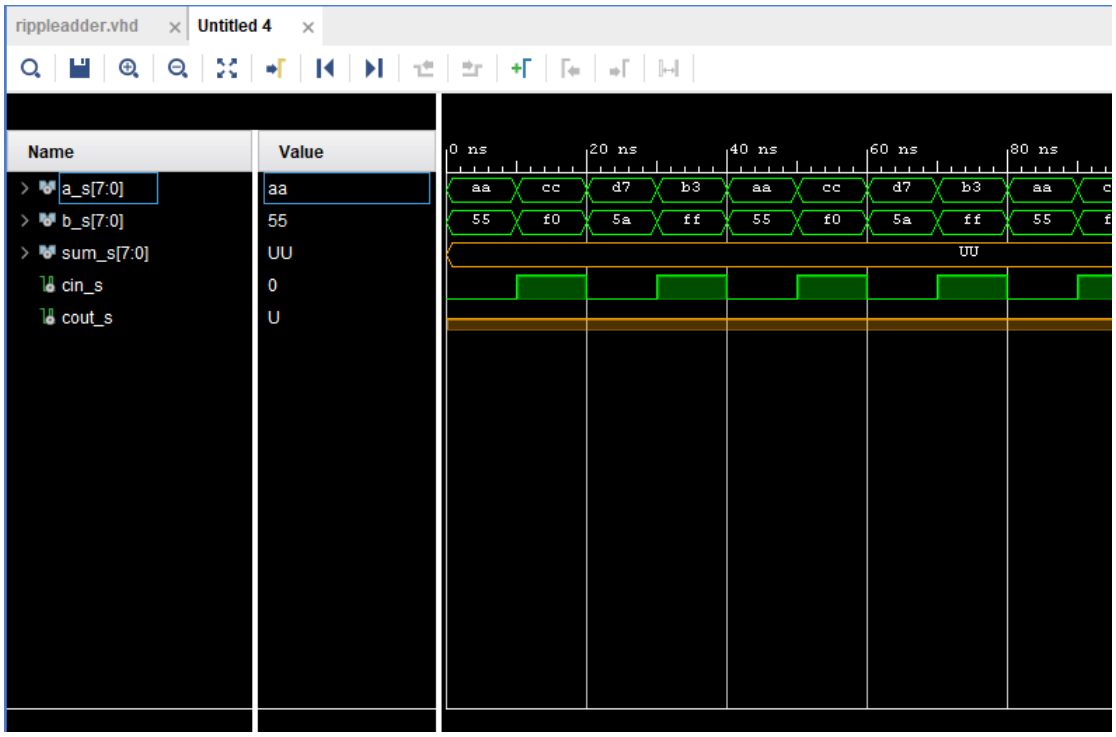
```
begin


U1 : full_adder_con port map (a(0),b(0),cin,sum(0),carry(0));

U2 : for i in 1 to 6 generate

U3 : full_adder_con port map (a(i),b(i),carry(i-1),sum(i),carry(i));

end generate;

U4 : full_adder_con port map (a(7),b(7),carry(6),sum(7),cout);



end structural;
```

tb_rippleadder

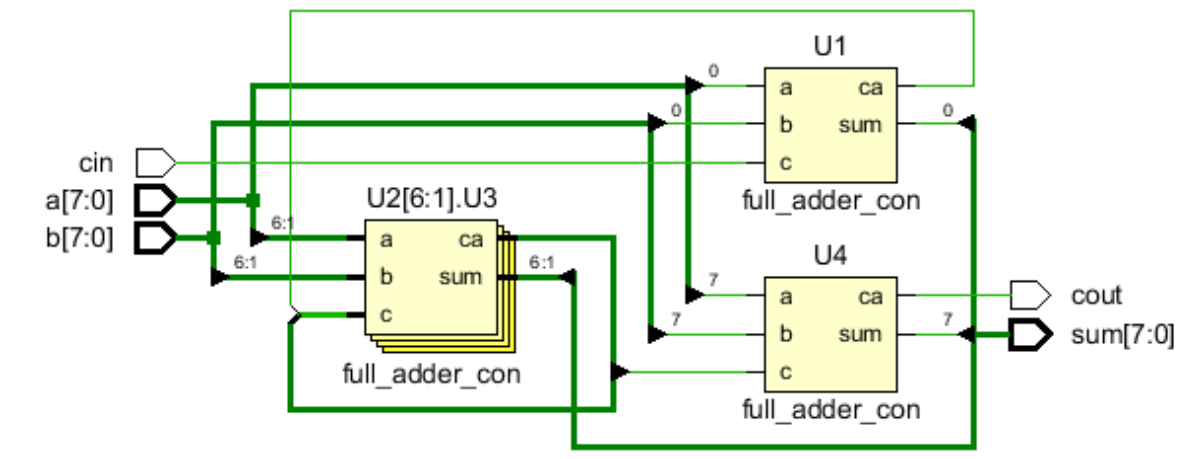
```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity tb_rippleadder is

end tb_rippleadder;

architecture beh of tb_rippleadder is

component rippleadder is

port ( a : in std_logic_vector(7 downto 0);

 b : in std_logic_vector(7 downto 0);

 cin : in std_logic;

 sum : out std_logic_vector(7 downto 0);

 cout : out std_logic);

end component;

signal a_s,b_s,sum_s : std_logic_vector (7 downto 0);

signal cin_s,cout_s : std_logic;

begin

DUT : rippleadder port map (a_s,b_s,cin_s,sum_s,cout_s);

process

 begin

  a_s <= "10101010";
```

```
  b_s <= "01010101";
  cin_s <= '0';
  wait for 10 ns;

  a_s <= "11001100";
  b_s <= "11110000";
  cin_s <= '1';
  wait for 10 ns;

  a_s <= "11010111";
  b_s <= "01011010";
  cin_s <= '0';
  wait for 10 ns;

  a_s <= "10110011";
  b_s <= "11111111";
  cin_s <= '1';
  wait for 10 ns;
 end process;

end beh;
```

Simulation:



Schematic:



Problem 2

Design and simulate **8-bit carry look-ahead adder**.

- How does the utilization changed? (vs. rca)
- How does the timing changed? (vs. rca)

Code:

Lookahead.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;



entity lookahead is

Port ( A : in STD_LOGIC_VECTOR (7 downto 0);

B : in STD_LOGIC_VECTOR (7 downto 0);

Cin : in STD_LOGIC;

S : out STD_LOGIC_VECTOR (7 downto 0);

Cout : out STD_LOGIC);

end lookahead;



architecture Behavioral of lookahead is



component full_adder

Port ( A : in STD_LOGIC;
```

```
B : in STD_LOGIC;
Cin : in STD_LOGIC;
S : out STD_LOGIC;
P : out STD_LOGIC;
G : out STD_LOGIC);
end component;

signal c1,c2,c3,c4,c5,c6,c7: STD_LOGIC;
signal P,G: STD_LOGIC_VECTOR(7 downto 0);
begin

PFA1: full_adder port map( A(0), B(0), Cin, S(0), P(0), G(0));
PFA2: full_adder port map( A(1), B(1), c1, S(1), P(1), G(1));
PFA3: full_adder port map( A(2), B(2), c2, S(2), P(2), G(2));
PFA4: full_adder port map( A(3), B(3), c3, S(3), P(3), G(3));
PFA5: full_adder port map( A(4), B(4), c4, S(4), P(4), G(4));
PFA6: full_adder port map( A(5), B(5), c5, S(5), P(5), G(5));
PFA7: full_adder port map( A(6), B(6), c6, S(6), P(6), G(6));
PFA8: full_adder port map( A(7), B(7), c7, S(7), P(7), G(7));

c1 <= G(0) OR (P(0) AND Cin);
c2 <= G(1) OR (P(1) AND G(0)) OR (P(1) AND P(0) AND Cin);
```

```
c3 <= G(2) OR (P(2) AND G(1)) OR (P(2) AND P(1) AND G(0)) OR (P(2) AND P(1) AND P(0) AND Cin);

c4 <= G(3) OR (P(3) AND G(2)) OR (P(3) AND P(2) AND G(1)) OR (P(3) AND P(2) AND P(1) AND P(0) AND
Cin);

c5 <= G(4) OR (P(4) AND G(3)) OR (P(4) AND P(2) AND G(2)) OR (P(4) AND P(3) AND P(2) AND P(1) AND P(0)
AND Cin);

c6 <= G(5) OR (P(5) AND G(4)) OR (P(5) AND P(3) AND G(3)) OR (P(5) AND P(4) AND P(3) AND P(2) AND P(1)
AND P(0) AND Cin);

c7 <= G(6) OR (P(6) AND G(5)) OR (P(6) AND P(4) AND G(4)) OR (P(6) AND P(5) AND P(4) AND P(3) AND P(2)
AND P(1) AND P(0) AND Cin);

Cout <= G(7) OR (P(7) AND G(7)) OR (P(7) AND P(6) AND G(5)) OR (P(7) AND P(6) AND P(5) AND G(4)) OR
(P(7) AND P(6) AND P(5) AND P(4) AND P(3) AND P(2) AND P(1) AND P(0) AND Cin);

end Behavioral;
```

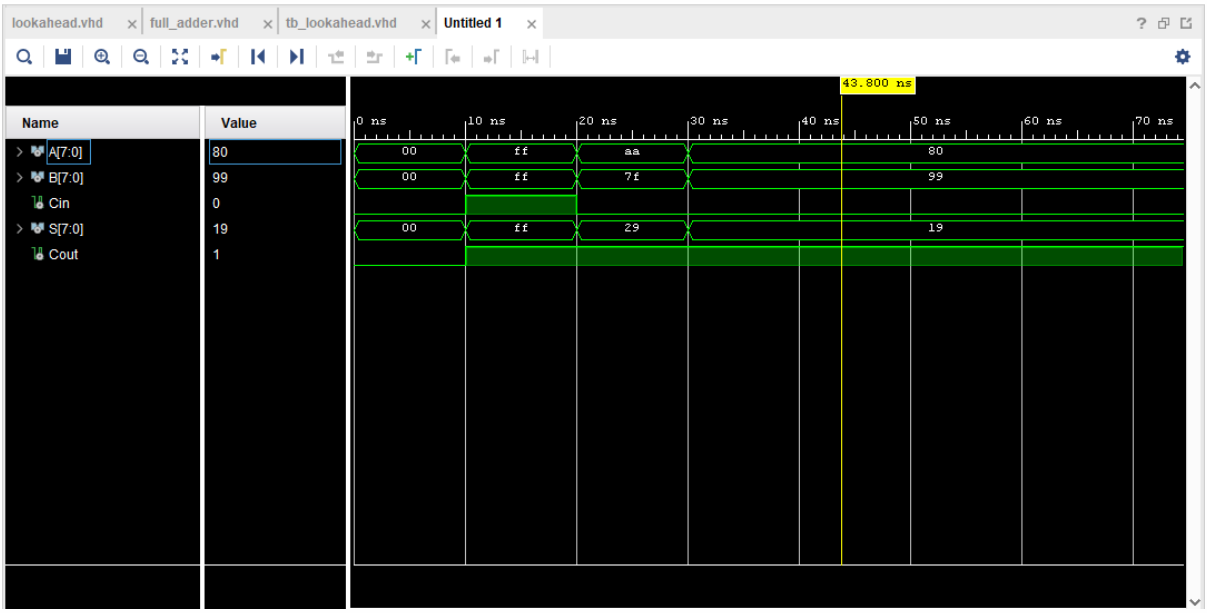
tb_lookahead

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL;

ENTITY tb_lookahead IS

END tb_lookahead;

ARCHITECTURE behavior OF tb_lookahead IS

COMPONENT lookahead

PORT(

A : IN std_logic_vector(7 downto 0);

B : IN std_logic_vector(7 downto 0);

Cin : IN std_logic;

S : OUT std_logic_vector(7 downto 0);

Cout : OUT std_logic

);
```

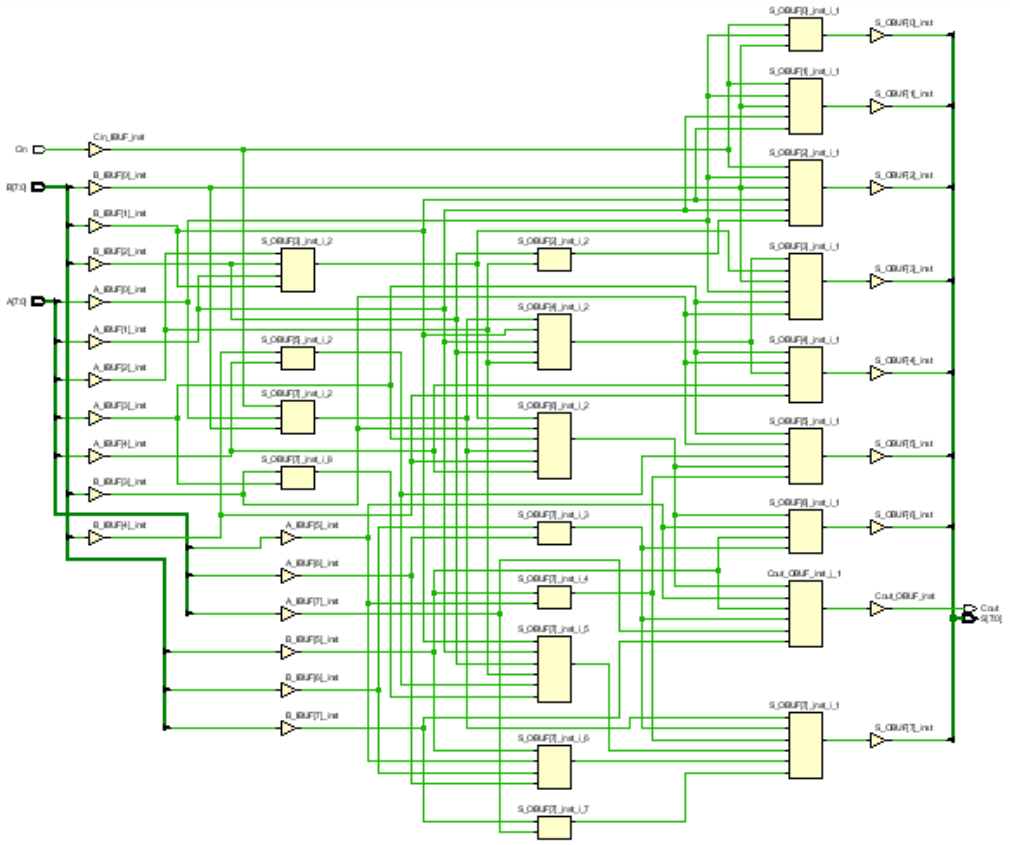
```
END COMPONENT;

 --Inputs

signal A : std_logic_vector(7 downto 0) := (others => '0');

signal B : std_logic_vector(7 downto 0) := (others => '0');

signal Cin : std_logic := '0';

--Outputs

signal S : std_logic_vector(7 downto 0);

signal Cout : std_logic;

BEGIN

uut: lookahead PORT MAP (

A => A,

B => B,

Cin => Cin,

S => S,

Cout => Cout

);



stim_proc: process

begin

wait for 10 ns;

A <= "11111111";

B <= "11111111";
```

```
Cin <= '1';

wait for 10 ns;

A <= "10101010";

B <= "01111111";

Cin <= '0';

wait for 10 ns;

A <= "00000010";

B <= "00000011";

Cin <= '0';

wait;

end process;

END;
```

Simulation:



Schematic:



Utizilation Report

Hierarchy

Name	Slice LUTs (53200)	Slice (13300)	LUT as Logic (53200)	Bonded IPADs (2)
lookahead	17	5	17	26

Problem 3

Design and simulate a **1-bit Arithmetic Logic Unit**. The ALU should support `a+b`, `a-b`, `a and b`, `a or b`, `a nor b`, `a nand b`, `a < b`, `not a` operations with carry in support. You can use gate-level adder subtractor operations.

one_alu Design Source

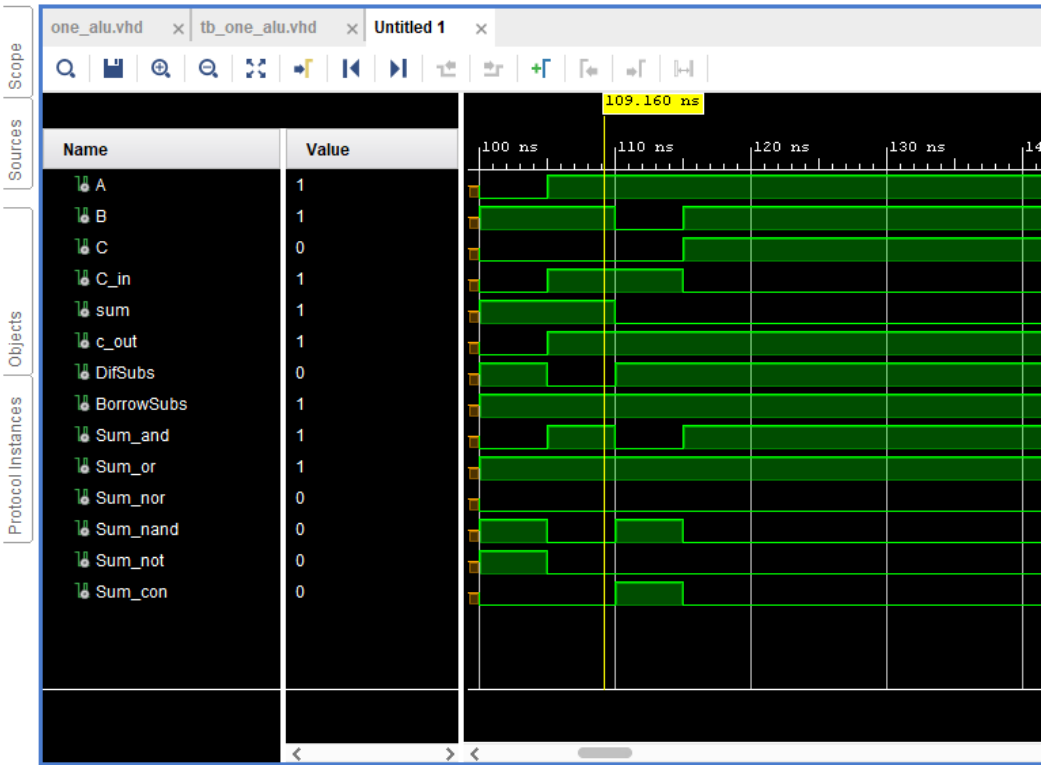
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity one_alu is
 Port (
 A : in std_logic;
 B : in std_logic;
 C : in std_logic; --substractor C
 C_in : in std_logic; -- adder C_in
 sum : out std_logic; -- adder sum
 c_out : out std_logic; --adder c_out
 DifSubs : out std_logic; --substractor
 BorrowSubs : out std_logic; -- substractor
 Sum_and : out std_logic;
 Sum_or : out std_logic;
 Sum_nor : out std_logic;
 Sum_nand : out std_logic;
 Sum_con : out std_logic;
 Sum_not : out std_logic
  );
end one_alu;
architecture Behavioral of one_alu is
begin
DifSubs <= A xor B xor C;
BorrowSubs <= (A and B) or (B and (not C)) or (A and (not C));
sum <= A xor B xor C_in;
c_out <= (A and B) or (A and C_in) or (B and C_in);
Sum_and <= A and B;
Sum_or <= A or B;
Sum_nor <= A nor B;
Sum_nand <= A nand B;
Sum_not <= (not A);
Sum_con <= A and (not B);

end Behavioral;
```

Testbench tb_one_alu

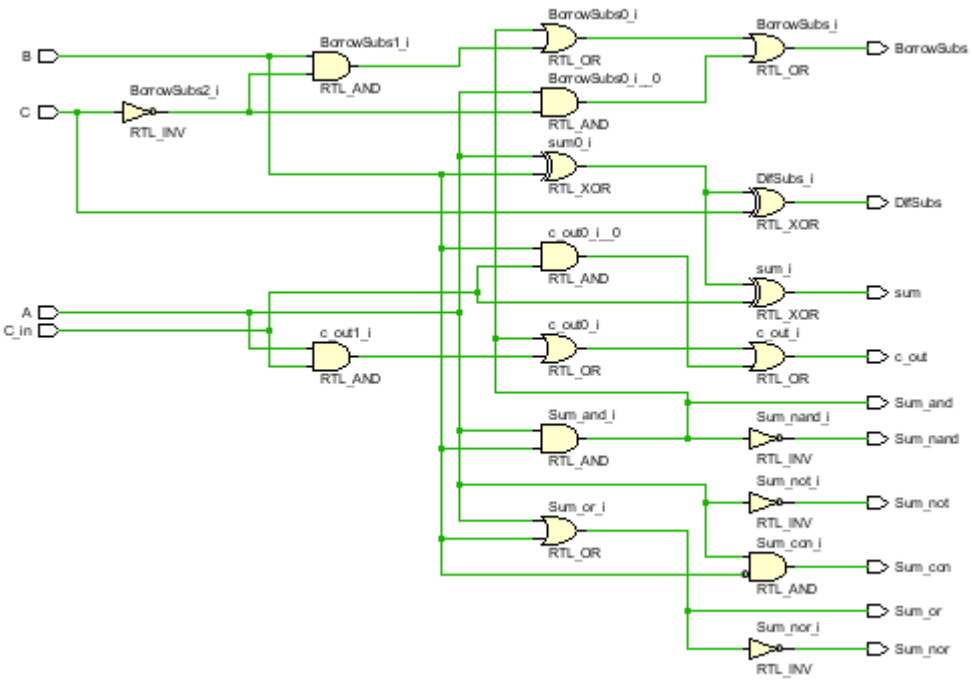
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tb_one_alu is
--  Port ( );
end tb_one_alu;
architecture Behavioral of tb_one_alu is
component one_alu is
port( A,B,C, C_in : in std_logic;
sum : out std_logic; -- adder sum
 c_out : out std_logic; --adder c_out
 DifSubs : out std_logic; --substractor
 BorrowSubs : out std_logic; -- substractor
 Sum_and : out std_logic;
 Sum_or : out std_logic;
 Sum_nor : out std_logic;
 Sum_nand : out std_logic;
 Sum_con : out std_logic;
 Sum_not : out std_logic
 );
 end component;


signal A,B,C,C_in, sum, c_out, DifSubs, BorrowSubs,Sum_and, Sum_or,
Sum_nor,Sum_nand,Sum_not,Sum_con : std_logic;


begin
    h0: one_alu port map(
    A => A, B => B, C => C,C_in => C_in, sum => sum, c_out => c_out, DifSubs =>
DifSubs, BorrowSubs => BorrowSubs, Sum_and=>Sum_and, Sum_or => Sum_or,
Sum_nor=>Sum_nor,Sum_nand=>Sum_nand,Sum_not=>Sum_not,Sum_con=>Sum_con);


process
begin
    wait for 100ns;
    A <= '0';
    B <= '1';
    C <= '0';
```

```
    C_in <= '0';
    wait for 5ns;
    A <= '1';
    B <= '1';
    C <= '0';
    C_in <= '1';
    wait for 5ns;
    A <= '1';
    B <= '0';
    C <= '0';
    C_in <= '1';
    wait for 5ns;
    A <= '1';
    B <= '1';
    C <= '1';
    C_in <= '0';
    wait;
end process;

end Behavioral;

//////End code
```

Simulation:



Schematic:



Utizilation Report:

Name	Slice LUTs (53200)	Slice (13300)	LUT as Logic (53200)	Bonded IPADs (2)
one_alu	5	2	5	14

Problem 4

Design and simulate an **32-bit Arithmetic Logic Unit** *using the one you designed in Problem 3*. For simulation, just pick 4 random input pairs for each possible operation. (32 different cases total)

alu_32 design code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity alu_32 is
 Port (
 A : in std_logic_vector(31 downto 0);
 B : in std_logic_vector(31 downto 0);
 C : in std_logic_vector(31 downto 0); --substractor C
 C_in : in std_logic_vector(31 downto 0); -- adder C_in
 sum : out std_logic_vector(31 downto 0); -- adder sum
 c_out : out std_logic_vector(31 downto 0); --adder c_out
 DifSubs : out std_logic_vector(31 downto 0); --substractor
 BorrowSubs : out std_logic_vector(31 downto 0); -- substractor
 Sum_and : out std_logic_vector(31 downto 0);
 Sum_or : out std_logic_vector(31 downto 0);
 Sum_nor : out std_logic_vector(31 downto 0);
 Sum_nand : out std_logic_vector(31 downto 0);
 Sum_con : out std_logic_vector(31 downto 0);
 Sum_not : out std_logic_vector(31 downto 0)
  );
end alu_32;

architecture Behavioral of alu_32 is



begin
DifSubs <= A xor B xor C;
```

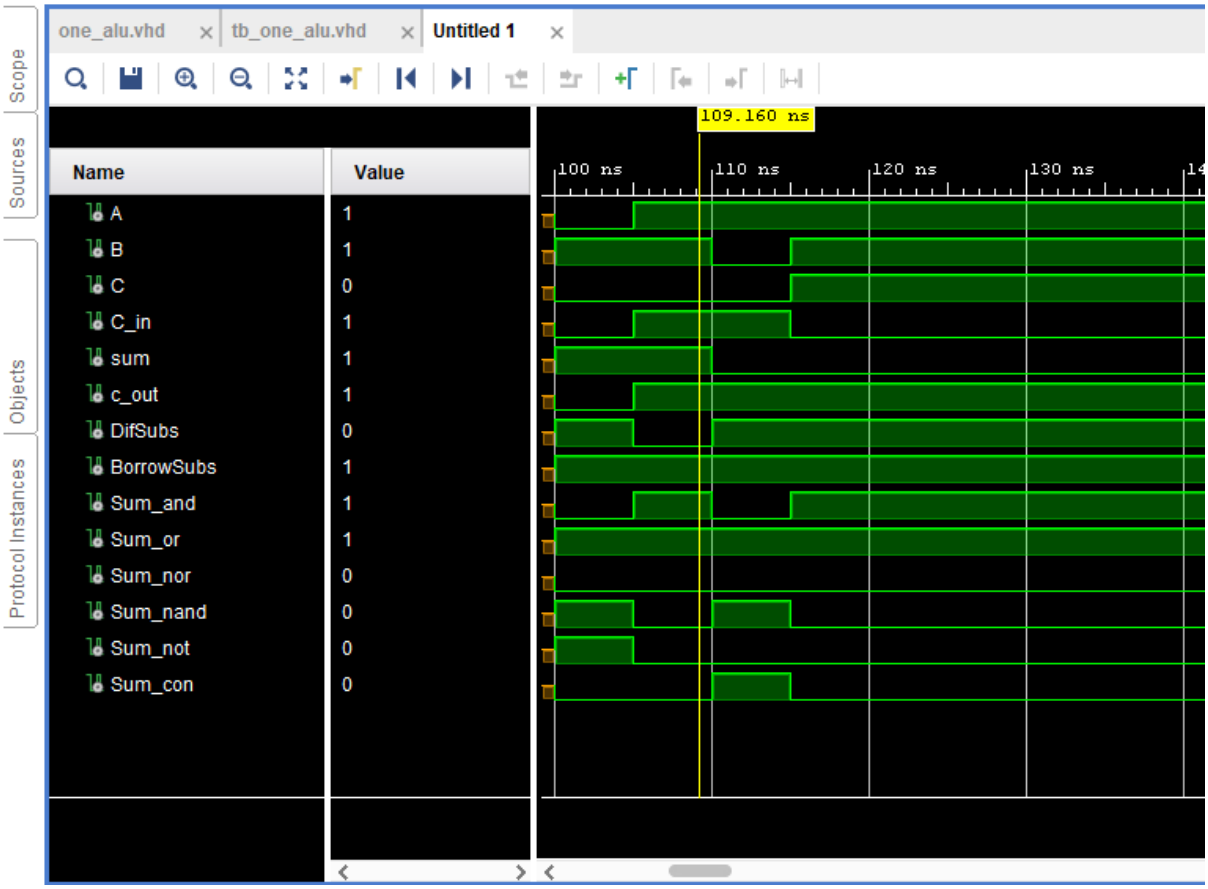
```
BorrowSubs <= (A and B) or (B and (not C)) or (A and (not C));
sum <= A xor B xor C_in;
c_out <= (A and B) or (A and C_in) or (B and C_in);
Sum_and <= A and B;
Sum_or <= A or B;
Sum_nor <= A nor B;
Sum_nand <= A nand B;
Sum_not <= (not A);
Sum_con <= A and (not B);

end Behavioral;
```

tb_alu_32

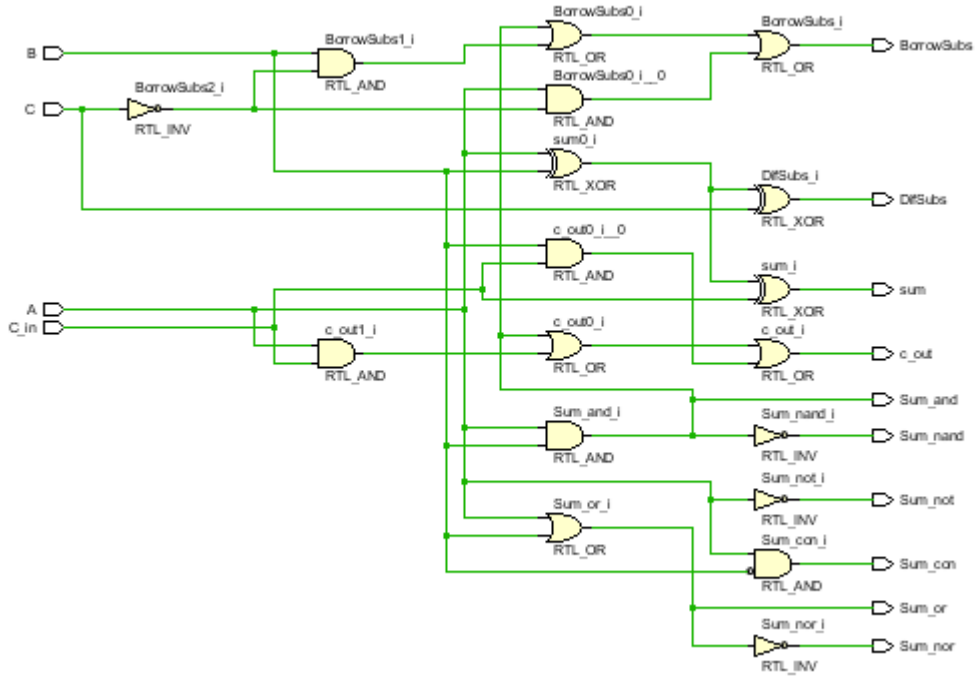
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity alu_32 is
 Port (
 A : in std_logic_vector(31 downto 0);
 B : in std_logic_vector(31 downto 0);
 C : in std_logic_vector(31 downto 0); --substractor C
 C_in : in std_logic_vector(31 downto 0); -- adder C_in
 sum : out std_logic_vector(31 downto 0); -- adder sum
 c_out : out std_logic_vector(31 downto 0); --adder c_out
 DifSubs : out std_logic_vector(31 downto 0); --substractor
 BorrowSubs : out std_logic_vector(31 downto 0); -- substractor
 Sum_and : out std_logic_vector(31 downto 0);
 Sum_or : out std_logic_vector(31 downto 0);
 Sum_nor : out std_logic_vector(31 downto 0);
 Sum_nand : out std_logic_vector(31 downto 0);
 Sum_con : out std_logic_vector(31 downto 0);
 Sum_not : out std_logic_vector(31 downto 0)
  );
end alu_32;

architecture Behavioral of alu_32 is


begin
DifSubs <= A xor B xor C;
```

```
BorrowSubs <= (A and B) or (B and (not C)) or (A and (not C));

sum <= A xor B xor C_in;

c_out <= (A and B) or (A and C_in) or (B and C_in);

Sum_and <= A and B;

Sum_or <= A or B;

Sum_nor <= A nor B;

Sum_nand <= A nand B;

Sum_not <= (not A);

Sum_con <= A and (not B);


end Behavioral;
```

Simulation:



Schematic:



Utizilation Report:

I got some errors when i implementation the Project. I couldn’t fix the problem.