Código de implementación

Clase ALU4BITS

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE IEEE.std_logic_arith.ALL;
use work.PAQUETE_ALU.ALL;

entity ALU4BITS is
    GENERIC ( SIZE: INTEGER := 4);
    Port ( A,B: in STD_LOGIC_VECTOR (SIZE-1 downto 0);
           ALUOP: IN std_logic_vector(3 DOWNTO 0);
           RES : inout STD_LOGIC_VECTOR (SIZE-1 downto 0);
           C,N,Z,OV : OUT std_logic
           );
end ALU4BITS;

architecture Behavioral of ALU4BITS is
SIGNAL CARRAY: std_logic_vector(SIZE DOWNTO 0);
begin

    CARRAY(0) <= ALUOP(2);

    CICLO: FOR I IN 0 TO SIZE-1 GENERATE
        ELEMENTO: ALU1BIT PORT MAP
            (
                A       => A(I),
                B       => B(I),
                CIN     => CARRAY(I),
                AINVERT => ALUOP(3),
                BINVERT => ALUOP(2),
                OP      => ALUOP(1 DOWNTO 0),
                RES     => RES(I),
                COUT    => CARRAY(I+1)
            );
    END GENERATE;
    -- signo y zero siempre
    N <= RES(SIZE-1);
    Z <= '1' when res=0 else '0';

    -- carry y overflow se calcular para operaciones aritmeticas
    C <= CARRAY(SIZE) when ALUOP="0011" OR ALUOP="0111" ELSE '0';
```

```
    OV <= (CARRAY(SIZE) XOR CARRAY(SIZE-
1)) when ALUOP="0011" OR ALUOP="0111" ELSE '0';


--    OTRAS FORMAS DE CALCULAR Z
--    Z <= ZAUX;
--    Z <= not (RES(3) OR RES(2) OR RES(1) OR RES(0));
--    Z <= '1' when (res=(others => '0')) else '0';

end Behavioral;
```

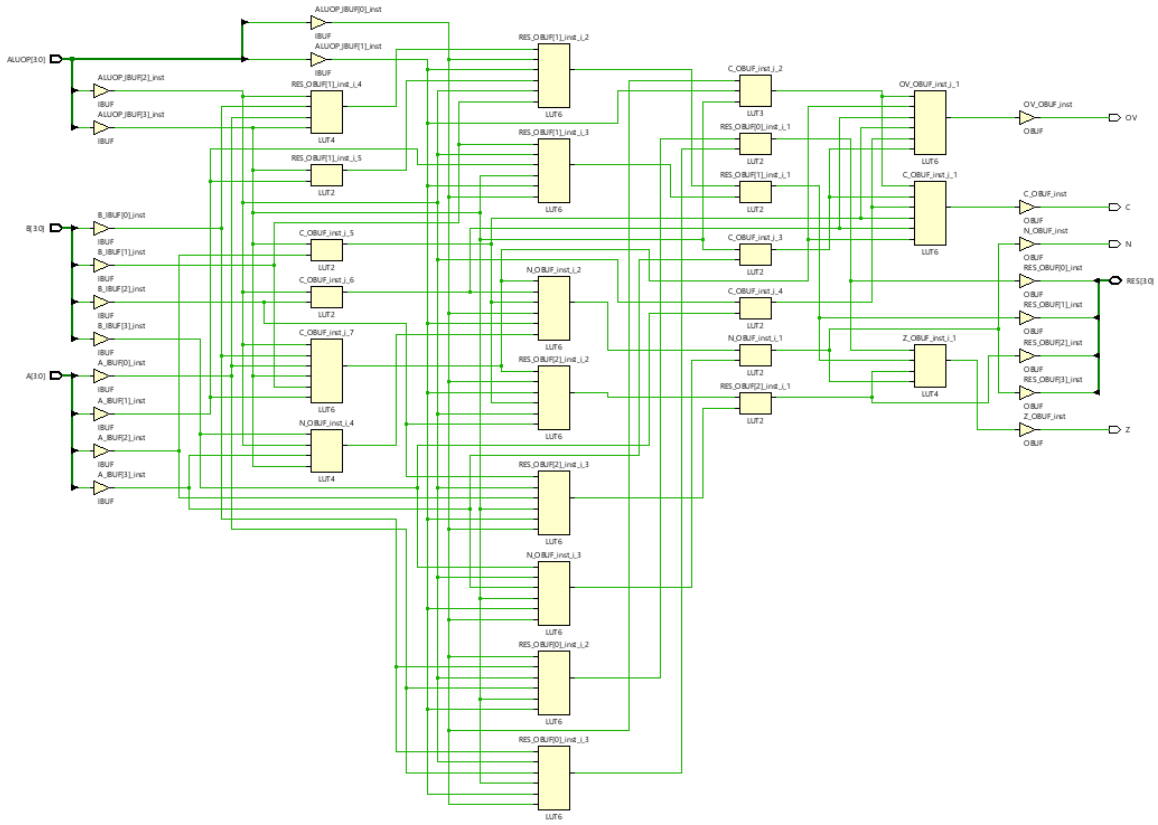
CLASE ALU1BIT

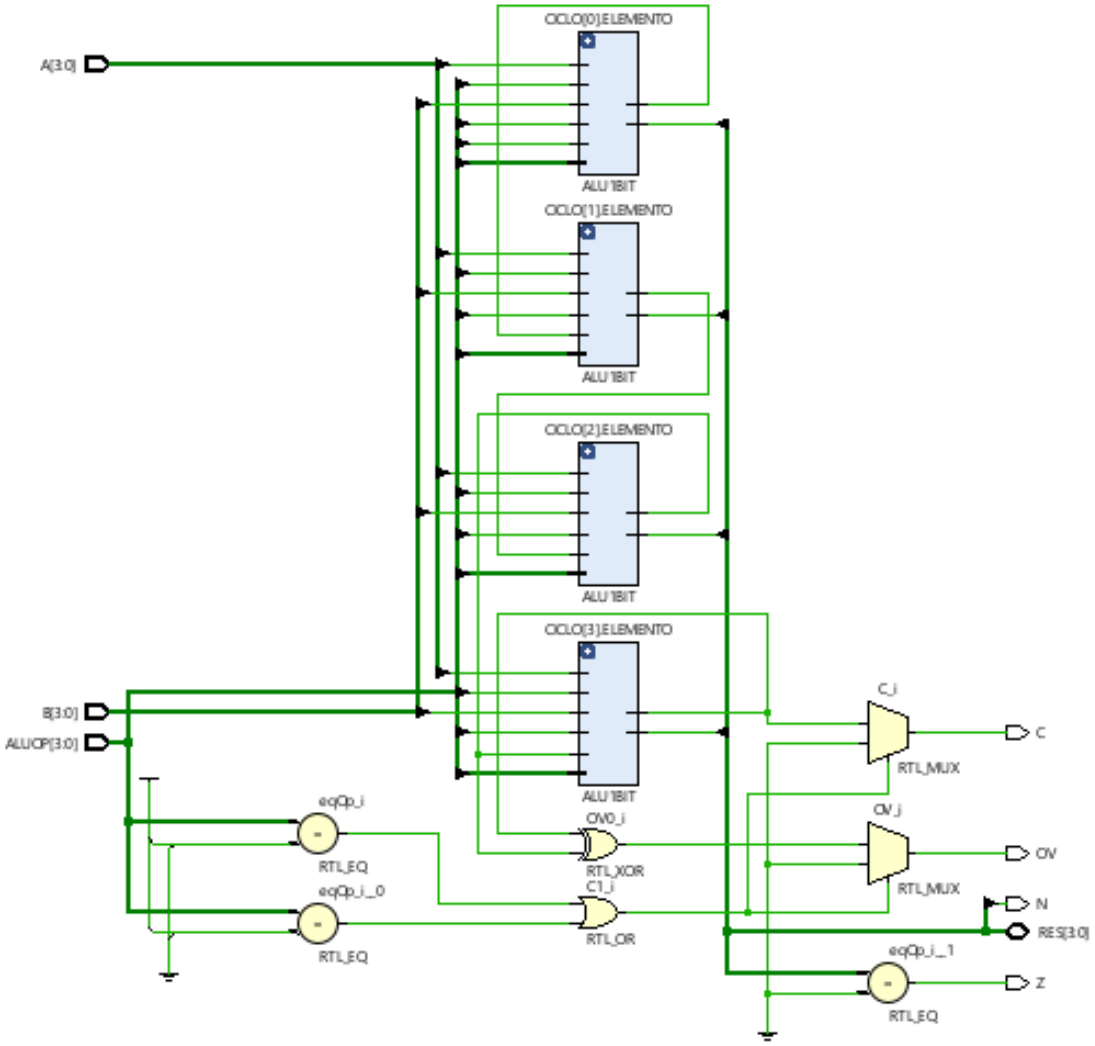
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ALU1BIT is
    Port ( A,B,AINVERT,BINVERT, CIN : in STD_LOGIC;
           OP : in STD_LOGIC_VECTOR (1 downto 0);
           RES, COUT : out STD_LOGIC);
end ALU1BIT;

architecture Behavioral of ALU1BIT is
SIGNAL AUXA, AUXB, AUXAND, AUXOR, AUXXOR, AUXSUMA: std_logic;
begin

    AUXA <= A XOR AINVERT;
    AUXB <= B XOR BINVERT;

    AUXAND <= AUXA AND AUXB;
    AUXOR <= AUXA OR AUXB;
    AUXXOR <= AUXA XOR AUXB;

    AUXSUMA <= AUXA XOR AUXB XOR CIN;
    COUT <= (AUXA AND CIN) OR (AUXA AND AUXB) OR (AUXB AND CIN);

    RES <=  AUXAND WHEN OP="00" ELSE
            AUXOR WHEN OP="01" ELSE
            AUXXOR WHEN OP="10" ELSE
            AUXSUMA;

end Behavioral;
```

PAQUETE PAQUETE_ALU

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

package PAQUETE_ALU is

    component ALU1BIT is
        Port ( A,B,AINVERT,BINVERT, CIN : in STD_LOGIC;
               OP : in STD_LOGIC_VECTOR (1 downto 0);
               RES, COUT : out STD_LOGIC);
    end component;

end package;
```

Diagrama RTL




A[3:0]
B[3:0]
ALUOP[3:0]
OCLO[0]ELEMENTO
ALU1BIT
OCLO[1]ELEMENTO
ALU1BIT
OCLO[2]ELEMENTO
ALU1BIT
OCLO[3]ELEMENTO
ALU1BIT
eqOp_i
RTL_EQ
eqOp_i_0
RTL_EQ
OV0_i
RTL_XOR
C1_i
RTL_OR
C_i
RTL_MUX
C
OV_i
RTL_MUX
OV
N
RES[3:0]
eqOp_i_1
RTL_EQ
Z

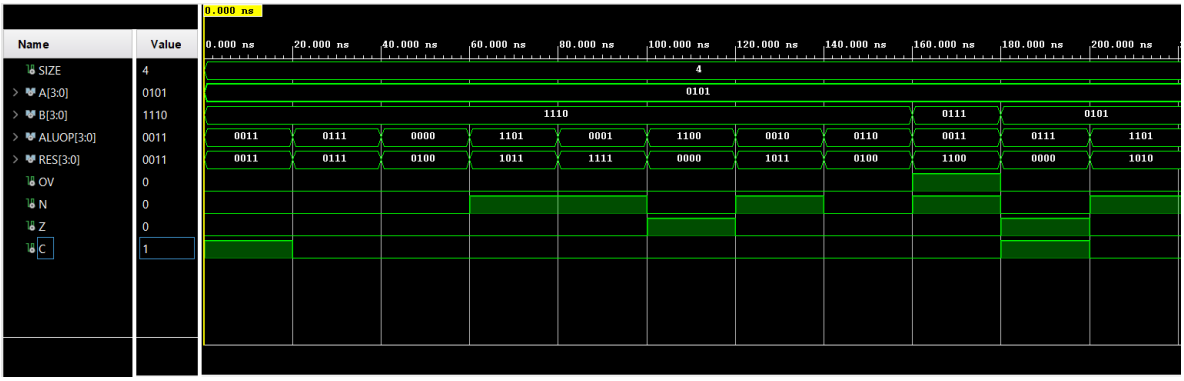
Código de Simulación

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity ALU4BITS_tb is
end;

architecture bench of ALU4BITS_tb is

  component ALU4BITS
      GENERIC ( SIZE: INTEGER := 4);
      Port ( A,B: in STD_LOGIC_VECTOR (SIZE-1 downto 0);
             ALUOP: IN std_logic_vector(3 DOWNTO 0);
             RES : inout STD_LOGIC_VECTOR (SIZE-1 downto 0);
             C,N,Z,OV : OUT std_logic
             );
  end component;
  SIGNAL SIZE: INTEGER:=4;
  signal A,B: STD_LOGIC_VECTOR (SIZE-1 downto 0);
  signal ALUOP: std_logic_vector(3 DOWNTO 0);
  signal RES: STD_LOGIC_VECTOR (SIZE-1 downto 0);
  signal C,N,Z,OV: std_logic ;

begin

  -- Insert values for generic parameters !!
  uut: ALU4BITS generic map ( SIZE  =>  SIZE)
                  port map ( A     => A,
                             B     => B,
                             ALUOP => ALUOP,
                             RES   => RES,
                             C     => C,
                             N     => N,
                             Z     => Z,
                             OV    => OV );

  stimulus: process
  begin

    -- Put initialisation code here
    A <= "0101";
    B <= "1110";
```

```
    -- Suma
    ALUOP <= "0011";
    WAIT for 20 ns;

    -- Resta
    ALUOP <= "0111";
    WAIT for 20 ns;

    -- AND
    ALUOP <= "0000";
    WAIT for 20 ns;

    -- NAND
    ALUOP <= "1101";
    WAIT for 20 ns;

    -- OR
    ALUOP <= "0001";
    WAIT for 20 ns;

    -- NOR
    ALUOP <= "1100";
    WAIT for 20 ns;

    -- XOR
    ALUOP <= "0010";
    WAIT for 20 ns;

    -- XNOR
    ALUOP <= "0110";
    WAIT for 20 ns;

    -- SUMA
    A <= "0101";
    B <= "0111";
    ALUOP <= "0011";
    WAIT for 20 ns;

    -- RESTA
    A <= "0101";
    B <= "0101";
    ALUOP <= "0111";
    WAIT for 20 ns;

    -- NAND(NOT)
```

```
    ALUOP <= "1101";
    -- Put test bench stimulus code here

    wait;
  end process;


end;
```

Simulación


Name
Value
SIZE
4
A[3:0]
0101
B[3:0]
1110
ALUOP[3:0]
0011
RES[3:0]
0011
OV
0
N
0
Z
0
C
1
0.000 ns
20.000 ns
40.000 ns
60.000 ns
80.000 ns
100.000 ns
120.000 ns
140.000 ns
160.000 ns
180.000 ns
200.000 ns
4
0101
1110
0111
0101
0011 0111 0000 1101 0001 1100 0010 0110 0011 0111 1101
0011 0111 0100 1011 1111 0000 1011 0100 1100 0000 1010

Resultados

Operación	*A*	*B*	*RES*	*BANDERAS*			
				OV	**N**	**Z**	**C**
Suma	5	-2	0011	0	0	0	1
Resta	5	-2	0111	0	0	0	0
AND	5	-2	0100	0	0	0	0
NAND	5	-2	1011	0	1	0	0
OR	5	-2	1111	0	1	0	0
NOR	5	-2	0000	0	0	1	0
XOR	5	-2	1011	0	1	0	0
XNOR	5	-2	0100	0	0	0	0
Suma	5	7	1100	1	1	0	0
Resta	5	5	0000	0	0	1	1
NAND(NOT)	5	5	1010	0	1	0	0