# Practica 4

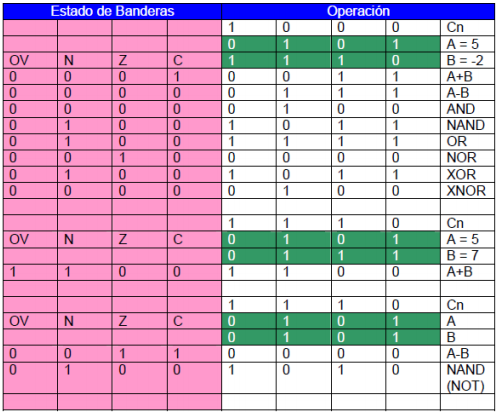
# Unidad aritmética/lógica de 4 bits

1. Implementar el circuito para la ALU diseñado en clase.

2. Generar la simulación con los siguientes estímulos y verificar que, tanto el vector de

resultado como los bits de las banderas correspondan a los datos que se muestran en la

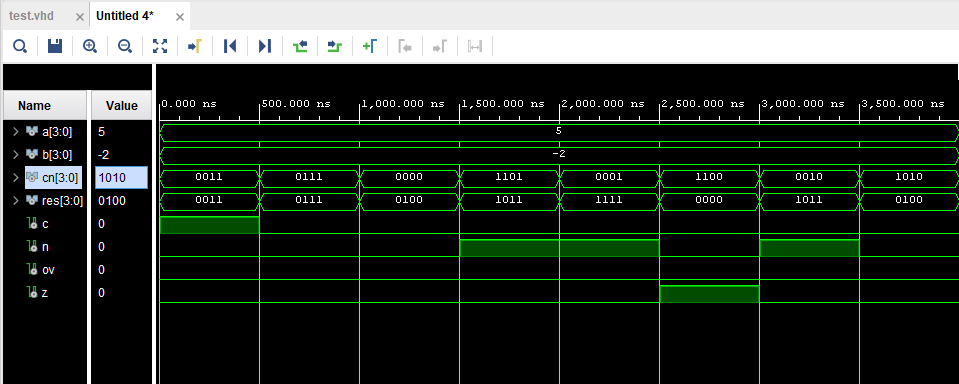
tabla para TODOS los casos.



## Simulaciones.

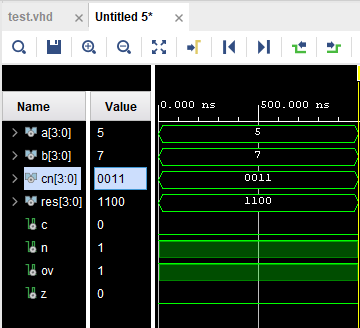
## Primer bloque

Operandos A = 5 y B = -2



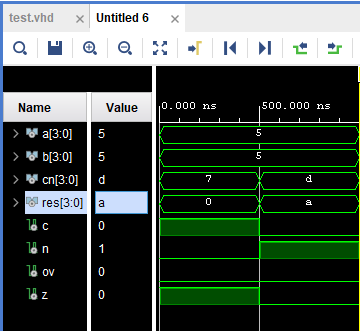
### Bloque 2

Operandos A = 5 y B = 7



### Bloque 3

Operandos A = 5 y B = 5



## Códigos de implementación.

### Archivo Suma1 (Sumador completo de un bit)

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY sum1 IS

    PORT (

        a, b, cin : IN STD\_LOGIC;

        s, cout : OUT STD\_LOGIC);

END sum1;

ARCHITECTURE Behavioral OF sum1 IS

BEGIN

    s <= a XOR b XOR cin;

    cout <= (a AND b) OR (b AND cin) OR (a AND cin);

END Behavioral;

### Archivo alu1 (Unidad aritmética lógica de 1 bit)

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY alu1 IS

    PORT (

        a, b, cin, sela, selb : IN STD\_LOGIC;

        op : IN STD\_LOGIC\_VECTOR (1 DOWNTO 0);

        res, cout : OUT STD\_LOGIC);

END alu1;

ARCHITECTURE Behavioral OF alu1 IS

    COMPONENT sum1 IS

        PORT (

            a, b, cin : IN STD\_LOGIC;

            s, cout : OUT STD\_LOGIC);

    END COMPONENT;

    SIGNAL aux\_a, aux\_b, res\_and, res\_or, res\_xor, res\_suma : std\_logic;

BEGIN

    aux\_a <= a XOR sela;

    aux\_b <= b XOR selb;

    res\_and <= aux\_a AND aux\_b;

    res\_or <= aux\_a OR aux\_b;

    res\_xor <= aux\_a XOR aux\_b;

    sumador : sum1 PORT MAP(

        a => aux\_a,

        b => aux\_b,

        cin => cin,

        s => res\_suma,

        cout => cout

    );

    res <= res\_and WHEN op = "00" ELSE

        res\_or WHEN op = "01" ELSE

        res\_xor WHEN op = "10" ELSE

        res\_suma;

END Behavioral;

### Archivo alun (Unidad Aritmética Lógica de 4 bits)

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY alun IS

    GENERIC (

        no : INTEGER := 4

    );

    PORT (

        a, b : IN STD\_LOGIC\_VECTOR (no - 1 DOWNTO 0);

        cn : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

        res : OUT STD\_LOGIC\_VECTOR (no - 1 DOWNTO 0);

        c : OUT STD\_LOGIC;

        n : OUT STD\_LOGIC;

        ov : OUT STD\_LOGIC;

        z : OUT STD\_LOGIC

    );

END alun;

ARCHITECTURE Behavioral OF alun IS

    COMPONENT alu1 IS

        PORT (

            a, b, cin, sela, selb : IN STD\_LOGIC;

            op : IN STD\_LOGIC\_VECTOR (1 DOWNTO 0);

            res, cout : OUT STD\_LOGIC);

    END COMPONENT;

    SIGNAL ca : std\_logic\_vector(no DOWNTO 0);

    SIGNAL re : std\_logic\_vector(no - 1 DOWNTO 0);

    SIGNAL q : std\_logic;

BEGIN

    ca(0) <= cn(2);

    ciclo : FOR i IN 0 TO no - 1 GENERATE

        objeto1 : alu1 PORT MAP

        (

            a => a(i),

            b => b(i),

            cin => ca(i),

            sela => cn(3),

            selb => cn(2),

            op(1) => cn(1),

            op(0) => cn(0),

            res => re(i),

            cout => ca(i + 1)

        );

*--q<=re(i) or q;*

    END GENERATE;

    c <= ca(no) WHEN (cn AND "0011") = "0011" ELSE

        '0';

    ov <= ca(no) XOR ca(no - 1) WHEN (cn AND "0011") = "0011" ELSE

        '0';

    n <= re(no - 1);

*--ov <= ca(no) xor ca(no-1);*

    res <= re;

*--sq <= '0';*

    z <= NOT(re(0) OR re(1) OR re(2) OR re(3));

## Código de simulación

library ieee;

use ieee.std\_logic\_1164.all;

entity tb\_alun is

end tb\_alun;

architecture tb of tb\_alun is

    component alun

        port (a   : in std\_logic\_vector (3 downto 0);

              b   : in std\_logic\_vector (3 downto 0);

              cn  : in std\_logic\_vector (3 downto 0);

              res : out std\_logic\_vector (3 downto 0);

              c   : out std\_logic;

              n   : out std\_logic;

              ov  : out std\_logic;

              z   : out std\_logic);

    end component;

    signal a   : std\_logic\_vector (3 downto 0);

    signal b   : std\_logic\_vector (3 downto 0);

    signal cn  : std\_logic\_vector (3 downto 0);

    signal res : std\_logic\_vector (3 downto 0);

    signal c   : std\_logic;

    signal n   : std\_logic;

    signal ov  : std\_logic;

    signal z   : std\_logic;

begin

    dut : alun

    port map (a   => a,

              b   => b,

              cn  => cn,

              res => res,

              c   => c,

              n   => n,

              ov  => ov,

              z   => z);

    stimuli : process

    begin

*-- Bloque 1*

        a <= "0101";

        b <= "1110";

        cn <= "0011";

        wait for 500ns;

        cn <= "0111";

        wait for 500ns;

        cn <= "0000";

        wait for 500ns;

        cn <= "1101";

        wait for 500ns;

        cn <= "0001";

        wait for 500ns;

        cn <= "1100";

        wait for 500ns;

        cn <= "0010";

        wait for 500ns;

        cn <= "1010";

        wait for 500ns;

*-- Bloque 2*

        a <= "0101";

        b <= "0111";

        cn <= "0011";

        wait for 500ns;

*-- Bloque 3*

        a <= "0101";

        b <= "0101";

        cn <= "0111";

        wait for 500ns;

        cn <= "1101";

        wait;

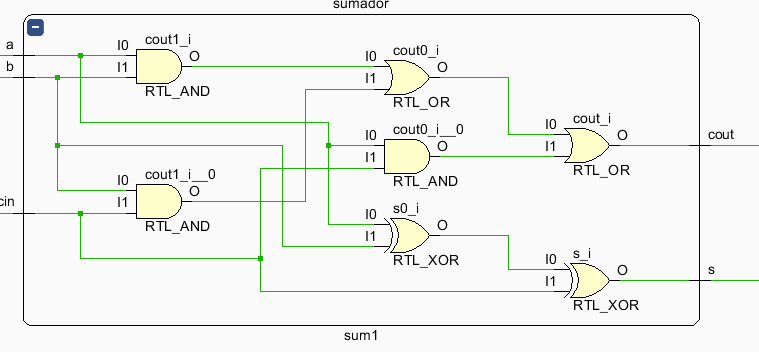
        wait;

    end process;

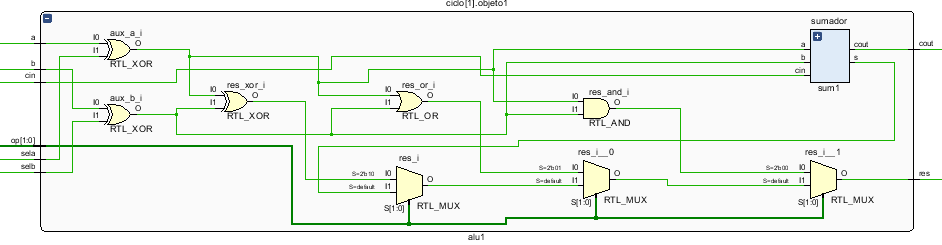
end tb;

## Diagramas RTL

### Suma1



### Alu1



### Alun

