Pontificia Universidad JAVERIANA Bogotá

Departamento de Ingeniería Electrónica
Talleres Organización de computadores
Eduardo A. Gerlein R

Informe Taller 1 - Diseño de circuitos Aritmeticos

Juan Pablo Conrado Molina
Febrero 2025

I. INTRODUCCIÓN

El presente informe se realizará la implementación de circuitos digitales por medio del software Quartus en conjunto con el lenguaje VHDL*(Very High Speed Integrated Circuits) Hardware Description Language)* y la introducción al concepto de ALU (Unidad Lógica Aritmética) que gestiona las operaciones aritméticas.

II. MARCO TEÓRICO

II-A. Compuertas Lógicas

Las compuertas lógicas son los bloques fundamentales de los circuitos digitales. Se utilizan para realizar operaciones booleanas básicas como AND, OR, NOT, XOR, entre otras. Estas compuertas permiten construir circuitos más complejos como sumadores y unidades de procesamiento.

II-B. Sumador Completo (Full Adder)

Un sumador completo es un circuito digital que suma tres bits de entrada: dos operandos y un bit de acarreo (carry-in). Sus ecuaciones de salida son:

$$S = A \oplus B \oplus C_{in} \tag{1}$$

$$C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B)) \tag{2}$$

Este módulo se utiliza como bloque básico para construir sumadores de múltiples bits.

II-C. Sumador de N Bits

Un sumador de N bits se construye encadenando múltiples sumadores completos, propagando el acarreo de un bit al siguiente. Esto permite sumar números binarios de tamaño arbitrario.

II-D. Complemento a 2

El complemento a 2 es una técnica utilizada en la representación de números negativos en binario. Se obtiene invirtiendo todos los bits de un número y sumando 1:

$$-X = \overline{X} + 1 \tag{3}$$

Esta operación permite realizar restas mediante sumas, facilitando el diseño de circuitos aritméticos.

II-E. Unidad de Procesamiento

La unidad de procesamiento es un componente de hardware que ejecuta diversas operaciones aritméticas y lógicas sobre datos de entrada. En este taller, se diseñó una unidad capaz de realizar suma, resta, negación en complemento a 2 e incremento de valores de entrada, controlada por una señal de selección.

III. DESARROLLO E IMPLEMENTACIÓN

III-A. Descripción del Diseño en VHDL

Se implementaron los siguientes módulos en VHDL:

- **Sumador Completo:** Implementado a nivel de compuertas.
- **Sumador de N Bits:** Basado en la interconexión de sumadores completos.
- **Sumador-Restador:** Utiliza el complemento a 2 para realizar sumas y restas con una señal de control.
- **Unidad de Procesamiento:** Incluye cuatro módulos de suma/resta y un multiplexor para seleccionar la operación.
- **Testbench:** Verifica la correcta operación de cada módulo con diferentes casos de prueba.

IV. Códigos implementados y RTL viewer

IV-A. Códigos del sistema

Código del fulladder c

El sumador completo es un circuito digital que permite sumar dos bits de entrada y un bit de acarreo de entrada, produciendo una suma y un acarreo de salida. Su implementación se basa en compuertas lógicas XOR y AND. El resultado de la suma se obtiene aplicando la operación XOR a las entradas, mientras que el acarreo de salida se calcula mediante una combinación de operaciones AND y OR.

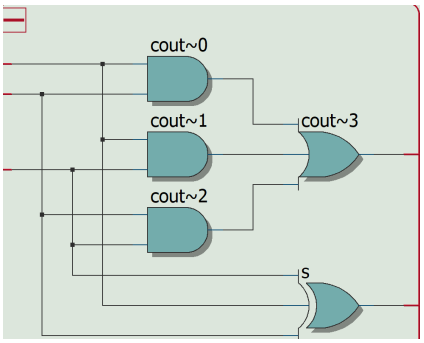

Figura 1. Rtl del full adder

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY full_adder IS
    PORT (
        x    : IN STD_LOGIC;
        y    : IN STD_LOGIC;
        cin  : IN STD_LOGIC;
        s    : OUT STD_LOGIC;
        cout : OUT STD_LOGIC
    );
END ENTITY full_adder;

ARCHITECTURE gate_level OF full_adder IS
BEGIN
    -- Implementaci n a nivel de
    compuertas
    s    <= x XOR y XOR cin;
    cout <= (x AND y) OR (x AND cin) OR (y
    AND cin);
END ARCHITECTURE gate_level;
```

Código nbit adder

El sumador de N bits se construye encadenando múltiples sumadores completos. Este diseño permite sumar dos números binarios de N bits propagando el acarreo a través de las diferentes posiciones. Se emplea un generador de instancias que permite iterar sobre cada bit del vector de entrada, conectando los sumadores completos de manera secuencial.

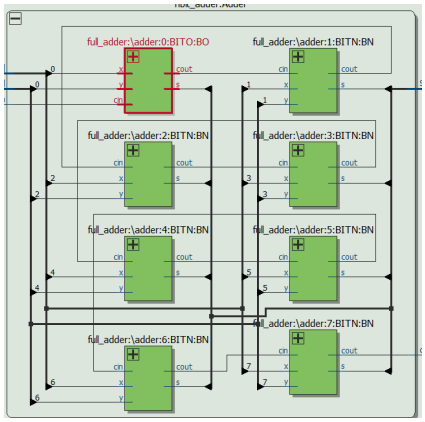

Figura 2. N bit adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
ENTITY nbit_adder IS
  GENERIC (N : INTEGER :=8);
  PORT(
    a, b  : IN STD_LOGIC_VECTOR (N-1 DOWNTO
     0);
    cin   : IN STD_LOGIC;
    S   : OUT  STD_LOGIC_VECTOR (N-1 DOWNTO
     0);
    cout  : OUT STD_LOGIC);

END ENTITY nbit_adder;

ARCHITECTURE rt1 OF nbit_adder IS

  SIGNAL carry: STD_LOGIC_VECTOR (N-1
    DOWNTO 0);
BEGIN
  adder: FOR i in N-1 DOWNTO 0 GENERATE
    BITO: IF i=0 GENERATE
        BO: ENTITY work.full_adder PORT MAP
     (a(i), b(i),cin,s(i), carry(i));
      END GENERATE;
    BITN: IF i/=0 GENERATE
        BN: ENTITY work.full_adder PORT MAP
     (a(i),b(i),carry(i-1),s(i),carry(i));
      END GENERATE;
  END GENERATE;
cout <= carry (carry'LEFT);

END ARCHITECTURE;
```

Código adder substractor

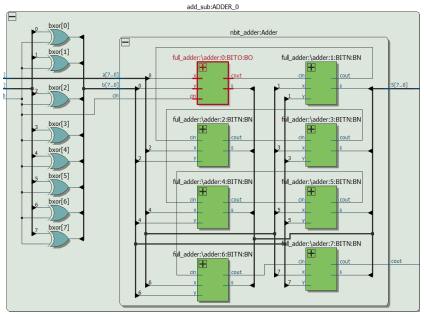

Figura 3. Adder substractor

```vhdl

-- add_sub.vhd (component):
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY add_sub IS
    GENERIC (N          : INTEGER :=8);
    PORT (
        a          : IN  STD_LOGIC_VECTOR (N-1 DOWNTO 0);
        b          : IN  STD_LOGIC_VECTOR (N-1 DOWNTO 0);
        addn_sub   : IN  STD_LOGIC;
        s          : OUT STD_LOGIC_VECTOR (N-1 DOWNTO 0);
        cout       : OUT STD_LOGIC
    );
END ENTITY add_sub;

ARCHITECTURE rtl OF add_sub IS
    SIGNAL bxor               :
    STD_LOGIC_VECTOR (N-1 DOWNTO 0);
    SIGNAL addn_sub_vector    :
    STD_LOGIC_VECTOR (N-1 DOWNTO 0);
BEGIN
    -- Create a vector with the value of addn_sub input
    vector_generation: FOR i IN N-1 DOWNTO 0 GENERATE
        addn_sub_vector(i) <= addn_sub;
    END GENERATE;

    bxor <= b XOR addn_sub_vector;

    -- Adder instantiation
    Adder: ENTITY work.nbit_adder
        GENERIC MAP (N => N)
        PORT MAP (
            a    => a,
            b    => bxor,
            cin  => addn_sub,
            s    => s,
            cout => cout
        );

END ARCHITECTURE;
```

CÓDIGO DE LA UNIDAD DE PROCESAMIENTO

La unidad de procesamiento es un módulo que permite realizar múltiples operaciones aritméticas, como suma, resta, negación en complemento a 2 e incremento. Se compone de cuatro instancias del módulo sumador/restador y un multiplexor que selecciona la operación a ejecutar en función de una señal de control.

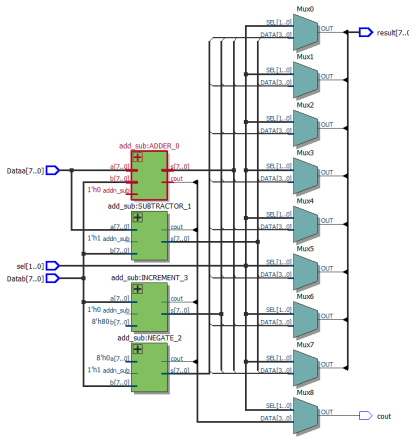

Figura 4. Unidad de Procesamiento

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY processing_unit IS
    GENERIC ( N : INTEGER := 8 );
    PORT (
        Dataa     : IN  STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        Datab     : IN  STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        sel       : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
        result    : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        cout      : OUT STD_LOGIC
    );
END ENTITY processing_unit;

ARCHITECTURE behavior OF processing_unit IS
    SIGNAL res_0, res_1, res_2, res_3 : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
    SIGNAL cout_0, cout_1, cout_2, cout_3 : STD_LOGIC;
BEGIN
    -- Instancia del sumador (Dataa + Datab)
    ADDER_0: ENTITY work.add_sub
        GENERIC MAP (N => N)
        PORT MAP (
            a        => Dataa,
            b        => Datab,
            addn_sub => '0',
```

```
            s         => res_0,
            cout      => cout_0
        );

    -- Instancia del restador (Dataa -
    Datab)
    SUBTRACTOR_1: ENTITY work.add_sub
        GENERIC MAP (N => N)
        PORT MAP (
            a         => Dataa,
            b         => Datab,
            addn_sub => '1',
            s         => res_1,
            cout      => cout_1
        );

    -- Instancia de negaci n en
    complemento a 2 (-Datab)
    NEGATE_2: ENTITY work.add_sub
        GENERIC MAP (N => N)
        PORT MAP (
            a         => (OTHERS => '0'), --
     Se usa 0 como entrada A
            b         => Datab,
            addn_sub => '1',
            s         => res_2,
            cout      => cout_2
        );

    -- Instancia del incremento (Datab + 1)
    INCREMENT_3: ENTITY work.add_sub
    GENERIC MAP (N => N)
    PORT MAP (
        a         => Datab,          --
    Usamos Datab como entrada A
        b         => "00000001",  -- Vector
    con solo el bit menos significativo en
    '1'
        addn_sub => '0',           -- Modo
    suma
        s         => res_3,
        cout      => cout_3
    );

    -- Multiplexor para seleccionar la
    salida
    WITH sel SELECT
        result <= res_0 WHEN "00",
                  res_1 WHEN "01",
                  res_2 WHEN "10",
                  res_3 WHEN "11",
                  (OTHERS => '0') WHEN
    OTHERS;

    WITH sel SELECT
        cout <= cout_0 WHEN "00",
                cout_1 WHEN "01",
                cout_2 WHEN "10",
                cout_3 WHEN "11",
                '0' WHEN OTHERS;

END ARCHITECTURE behavior;
```

IV-B. Testbench de cada código y explicación

TestBench del n_bit_adder

El código de testeo del n bit adder utiliza 10 vectores para evaluar los casos de overflow y de suma normal

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

entity tb_nbit_adder is
end tb_nbit_adder;

architecture testbench of tb_nbit_adder is
    constant N : integer := 4;
    signal a_tb, b_tb, s_tb :
    STD_LOGIC_VECTOR (N-1 downto 0) := (
    others => '0');
    signal cin_tb, cout_tb : STD_LOGIC :=
    '0';

    component nbit_adder
        generic (N : integer := 4);
        port (
            a, b  : in  STD_LOGIC_VECTOR (N
    -1 downto 0);
            cin   : in  STD_LOGIC;
            S     : out STD_LOGIC_VECTOR (N
    -1 downto 0);
            cout  : out STD_LOGIC
        );
    end component;

begin
    UUT: nbit_adder generic map (N => 4)
        port map (a => a_tb, b => b_tb, cin
     => cin_tb, s => s_tb, cout => cout_tb)
    ;

    process
    begin
        -- Test vector 1: 0 + 0
        a_tb <= "0000"; b_tb <= "0000";
    cin_tb <= '0'; wait for 10 ns;
        -- Test vector 2: 1 + 1
        a_tb <= "0001"; b_tb <= "0001";
    cin_tb <= '0'; wait for 10 ns;
        -- Test vector 3: 4 + 13 (overflow)
        a_tb <= "0100"; b_tb <= "1101";
    cin_tb <= '0'; wait for 10 ns;
        -- Test vector 4: 7 + 8(overflow)
        a_tb <= "0111"; b_tb <= "1000";
    cin_tb <= '0'; wait for 10 ns;
        -- Test vector 5: 15 + 1 (overflow)
        a_tb <= "1111"; b_tb <= "0001";
    cin_tb <= '0'; wait for 10 ns;
        -- Test vector 6: 10 + 10
        a_tb <= "1010"; b_tb <= "1010";
    cin_tb <= '0'; wait for 10 ns;
        -- Test vector 7: 6 + 9
        a_tb <= "0110"; b_tb <= "1001";
    cin_tb <= '0'; wait for 10 ns;
        -- Test vector 8: 14 + 2
```

```
        a_tb <= "1110"; b_tb <= "0010";
    cin_tb <= '0'; wait for 10 ns;
        -- Test vector 9: 5 + 5
        a_tb <= "0101"; b_tb <= "0101";
    cin_tb <= '0'; wait for 10 ns;
        -- Test vector 10: 12 + 3
        a_tb <= "1100"; b_tb <= "0011";
    cin_tb <= '0'; wait for 10 ns;
        wait;
    end process;
end testbench;
```

Código Unidad de Procesamiento

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY processing_unit_tb IS
END ENTITY processing_unit_tb;

ARCHITECTURE behavior OF processing_unit_tb IS
    -- Par metro de tama o de datos
    CONSTANT N : INTEGER := 8;

    -- Se ales de prueba con sufijo _tb
    SIGNAL dataa_tb, datab_tb, result_tb : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
    SIGNAL sel_tb : STD_LOGIC_VECTOR(1 DOWNTO 0);
    SIGNAL cout_tb : STD_LOGIC;

    -- Instancia del DUT (Device Under Test)
BEGIN
    DUT: ENTITY work.processing_unit
        GENERIC MAP (N => N)
        PORT MAP (
            Dataa  => dataa_tb,
            Datab  => datab_tb,
            sel    => sel_tb,
            result => result_tb,
            cout   => cout_tb
        );

    -- Proceso de est mulo
    STIMULUS: PROCESS
    BEGIN
        -- Caso 1: Suma (5 + 3)
        dataa_tb <= "00000101"; -- 5
        datab_tb <= "00000011"; -- 3
        sel_tb <= "00";
        WAIT FOR 20 ns;

        -- Caso 2: Suma (-4 + 7)
        dataa_tb <= "11111100"; -- -4 (Comp. A2)
        datab_tb <= "00000111"; -- 7
        sel_tb <= "00";
        WAIT FOR 20 ns;

        -- Caso 3: Resta (10 - 6)
        dataa_tb <= "00001010"; -- 10
        datab_tb <= "00000110"; -- 6
        sel_tb <= "01";
        WAIT FOR 20 ns;

        -- Caso 4: Resta (-5 - 3)
        dataa_tb <= "11111011"; -- -5
        datab_tb <= "00000011"; -- 3
        sel_tb <= "01";
        WAIT FOR 20 ns;

        -- Caso 5: Negaci n complemento a 2 (-7)
        datab_tb <= "00000111"; -- 7
        sel_tb <= "10";
        WAIT FOR 20 ns;

        -- Caso 6: Negaci n complemento a 2 (-(-10))
        datab_tb <= "11110110"; -- -10
        sel_tb <= "10";
        WAIT FOR 20 ns;

        -- Caso 7: Incremento (4 + 1)
        datab_tb <= "00000100"; -- 4
        sel_tb <= "11";
        WAIT FOR 20 ns;

        -- Caso 8: Incremento (-1 + 1)
        datab_tb <= "11111111"; -- -1
        sel_tb <= "11";
        WAIT FOR 20 ns;

        -- Caso 9: Suma con carry (127 + 1)
        dataa_tb <= "01111111"; -- 127
        datab_tb <= "00000001"; -- 1
        sel_tb <= "00";
        WAIT FOR 20 ns;

        -- Caso 10: Resta con carry (-128 - 1)
        dataa_tb <= "10000000"; -- -128
        datab_tb <= "00000001"; -- 1
        sel_tb <= "01";
        WAIT FOR 20 ns;

        -- Finaliza la simulaci n
        WAIT;
    END PROCESS STIMULUS;

END ARCHITECTURE behavior;
```

V. RESULTADOS DE SIMULACIÓN Y ANÁLISIS

Full Adder / N bit adder

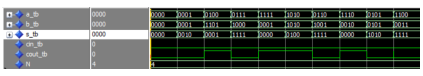

Figura 5. Full Adder en binario

Figura 8. Adder substractor fig 2

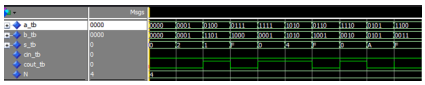

Figura 6. Full Adder Hex

Al analizar el comportamiento del N bit adder las sumas se realizan perfectamente mientras los números no tengan posibilidad de Overflow; Además, los números en quartus son representados en decimal como signed numbers en complemento a 2 por lo que se recomienda ver los números en Hexadecimal o en su defecto en binario.

Simulación Adder Substractor El adder substractor contiene 3 puntos importantes a la hora de simularlo

- ***Las restas de resultado negativo son representadas en su complemento a 2 pero se efectúan correctamente***
- ***Al igual que con las sumas, si se presenta overflow no se hará evidente ya sea un número negativo o positivo***

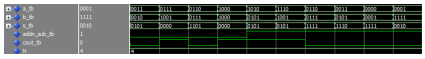

Figura 7. AdderSubstractor

Procesamiento

En las simulaciones de procesamiento se puede apreciar el trabajo del selector y como al tener 8 bits de valencia con signo se pueden tener representaciones más amplias. Al funcionar de base con los sumadores anteriores solo se debia verificar que la importación de los espacios de trabajo funcionaran correctamente.

Figura 9. Unidad de Procesamiento

VI. CONCLUSIONES

En este informe se ha detallado la implementación de distintos módulos aritméticos en VHDL, incluyendo un sumador completo, un sumador de N bits y una unidad de procesamiento. Estos circuitos forman la base del diseño de sistemas digitales más complejos y permiten la ejecución de operaciones fundamentales en arquitecturas computacionales.

El uso de la estructura 'WITH SELECT' en VHDL ha demostrado ser una herramienta clave en la simplificación de la selección de señales. En la unidad de procesamiento, esta estructura permitió implementar un multiplexor eficiente para elegir la operación aritmética correspondiente según la señal de selección 'sel'. Esto facilita la legibilidad del código y optimiza la ejecución de las operaciones, eliminando la necesidad de múltiples estructuras condicionales anidadas.

En general, la implementación de estos circuitos en VHDL ha permitido una mejor comprensión del diseño de hardware digital y su optimización, sentando las bases para desarrollos más avanzados en sistemas de procesamiento de datos.

- ***Si se tiene un Overflow no se hara evidente si este sobrepasa los bits usables del número***
- ***Todos los numeros son representados con signo y por consiguiente hay que interpretarlos como complemento a 2***
- ***El uso de señales para trasladar las señales de los buses entre modulos es fundamental para poder controlar cada uno.***