Universidad de Buenos Aires
Facultad de Ingeniería

86.41 - Sistemas Digitales

Trabajo Práctico Nº2: Aritmética de Punto Flotante

Batallan, David Leonardo, *Padrón 97529*
dbatallan@fi.uba.ar

7 de junio de 2025

Índice

1. Introducción

El presente trabajo tiene como objetivo implementar las funciones de una unidad aritmética de punto flotante, en particular: multiplicación, suma y resta. Dichas funciones serán descritas en lenguaje VHDL, simuladas y sintetizadas sobre el dispositivo `xc7a15tftg256-1`. Para la simulación se utilizarán archivos de prueba provistos por la cátedra.

2. Desarrollo

2.1. Especificaciones de diseño

2.2. Especificaciones de diseño

Se llevaron a cabo dos implementaciones de unidades de cálculo en punto flotante, orientadas a realizar operaciones de multiplicación y suma/resta. Para ambas se definieron las siguientes características:

- Se aplicó un esquema de redondeo por truncamiento, es decir, redondeo hacia cero.
- Los tamaños del significando (N_F) y del exponente (N_E) fueron definidos como parámetros genéricos para facilitar su configuración.
- No se contemplaron números denormalizados ni situaciones especiales como NaN o infinitos. En caso de que el resultado supere los límites del rango representable, se optó por una saturación del valor, retornando el mayor o menor número que pueda expresarse.
- Las simulaciones de ambas unidades se realizaron de manera automática a partir de los archivos de prueba proporcionados por la cátedra.

2.3. Diseño propuesto

En esta implementación se optó por una arquitectura puramente combinacional para las unidades aritméticas, es decir, sin el uso de señales de reloj.

Esta elección implica que la lógica no es secuencial: la salida se genera únicamente en función de las entradas actuales. Este tipo de descripción es completamente válida para síntesis, ya que representa circuitos lógicos sin elementos de memoria.

3. Calculo de Impedancia Característica

Se implementó el diseño propuesto en VHDL. A continuación, se presentan los códigos desarrollados.

3.1. Multiplicador

MULTIPLICADOR.VHD

```
1  -- TP 2
2  -- Materia: Sistemas digitales
3  -- Alumno: Batallan David Leonardo
4  -- Padron: 97529
5
6  library IEEE;
7  use IEEE.std_logic_1164.all;
8  use IEEE.numeric_std.all;
9
10 entity fp_mul is
11     generic(N : natural := 20;
12             NE : natural := 6
13     );
14     port (
15         X   : in std_logic_vector(N-1 downto 0); -- Primer operando
16         Y   : in std_logic_vector(N-1 downto 0); -- Segundo operando
17         Z   : out std_logic_vector(N-1 downto 0) -- Resultado
18     );
19 end fp_mul;
20
21 architecture behavioral of fp_mul is
22
23     -- Constantes auxiliares para manejo de exponentes y formatos
24     constant EXC        : natural := 2**(NE-1)-1; -- Valor base del exponente en exceso
25     constant NF         : natural := N-NE-1;      -- Cantidad de bits de la mantisa
26     constant EXCESO     : unsigned(NE+1 downto 0) := to_unsigned(EXC, NE+2); -- Exceso extendido
```

```
    constant E_MIN      : unsigned(NE-1 downto 0) := to_unsigned(0, NE);      -- Exponente m nimo
    constant E_MAX      : unsigned(NE-1 downto 0) := to_unsigned(2**(NE)-2, NE); -- Exponente m ximo
        permitido
    constant E_CEROS    : unsigned(NE+1 downto 0) := to_unsigned(0, NE+2); -- Verificaci n de exponente
        nulo
    constant F_CEROS    : unsigned(NF-1 downto 0) := to_unsigned(0, NF);    -- Verificaci n de mantisa
        nula
    constant RES_CERO   : unsigned(N-2 downto 0) := to_unsigned(0, N-1);    -- Parte no signo de un
        resultado cero

    -- Se ales internas: separaci n de campos
    signal sx       : std_logic; -- Bit de signo de X
    signal sy       : std_logic; -- Bit de signo de Y

    signal ex       : unsigned(NE+1 downto 0) := (others => '0'); -- Exponente de X extendido para control
         de overflow/underflow
    signal ey       : unsigned(NE+1 downto 0) := (others => '0'); -- Exponente de Y extendido
    signal cero_op  : std_logic := '0';                           -- Indicador de si alguno de los
        operandos es cero

    signal fx       : unsigned(NF-1 downto 0) := (others => '0'); -- Mantisa de X
    signal fy       : unsigned(NF-1 downto 0) := (others => '0'); -- Mantisa de Y

    signal mx       : unsigned(NF downto 0) := (others => '0');    -- Significand de X (1.F)
    signal my       : unsigned(NF downto 0) := (others => '0');    -- Significand de Y (1.F)

    signal sz           : std_logic;                                    -- Signo del resultado
    signal ez           : unsigned(NE-1 downto 0) := (others => '0');    -- Exponente final en exceso
    signal ez_aux       : unsigned(NE+1 downto 0) := (others => '0');    -- Exponente auxiliar intermedio
    signal ez_aux_p     : unsigned(NE+1 downto 0) := (others => '0');    -- Exponente auxiliar ajustado
    signal mz           : unsigned(2*NF+1 downto 0) := (others => '0'); -- Producto de los significands
    signal fz           : unsigned(NF-1 downto 0) := (others => '0');    -- Mantisa resultante
    signal fz_aux       : unsigned(NF-1 downto 0) := (others => '0');    -- Mantisa ajustada (previa al
        redondeo final)

begin

    -- Separaci n de campos: signo, exponente y mantisa
    sx <= X(NE+NF);
    sy <= Y(NE+NF);
    ex <= '0' & '0' & unsigned(X(NF+NE-1 downto NF)); -- Agrega ceros al frente para prevenir overflow/
        underflow
    ey <= '0' & '0' & unsigned(Y(NF+NE-1 downto NF));
    fx <= unsigned(X(NF-1 downto 0));
    fy <= unsigned(Y(NF-1 downto 0));

    -- Detecci n de operando cero
    cero_op <= '1' when ( (ex = E_CEROS) and (fx = F_CEROS) ) else
               '1' when ( (ey = E_CEROS) and (fy = F_CEROS) ) else
               '0';

    -- C lculo del signo del resultado
    sz <= sx xor sy;

    -- Construcci n de los significands (1.F)
    mx <= '1' & fx;
    my <= '1' & fy;

    -- Producto de los significands
    mz <= mx * my;

    -- C lculo del nuevo exponente en exceso
    ez_aux <= ex + ey - EXCESO;

    -- Ajuste y redondeo de la mantisa
    fz_aux <=
        mz(2*NF downto NF+1) when mz(2*NF+1) = '1' else
        mz(2*NF-1 downto NF);

    ez_aux_p <=
        (ez_aux + 1) when mz(2*NF+1) = '1' else ez_aux;

    -- L gica de saturaci n del exponente y asignaci n de mantisa
    ez <= E_MAX when ( (ez_aux_p(NE+1) = '0') and (ez_aux_p(NE) = '1') ) else
          E_MIN when ( (ez_aux_p(NE+1) = '1') and (ez_aux_p(NE) = '1') ) else
          ez_aux_p(NE-1 downto 0);

```

```
    fz <= (others => '1') when ( (ez_aux_p(NE+1) = '0') and (ez_aux_p(NE) = '1') ) else
          (others => '0') when ( (ez_aux_p(NE+1) = '1') and (ez_aux_p(NE) = '1') ) else
          fz_aux;

    -- Resultado final: cero si corresponde, o n mero normalizado
    Z <= std_logic_vector(sz & RES_CERO) when (cero_op = '1') else
         std_logic_vector(sz & ez & fz);

end architecture behavioral;
```

Listing 1: Código VHDL del multiplicador

3.2. Sumador/Restador

SUMADOR_RESTADOR.VHD

```
-- TP 2
-- Materia: Sistemas digitales
-- Alumno: Batallan David Leonardo
-- Padron: 97529

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity fp_subadd is
    generic(N : natural := 20;
            NE : natural := 6
    );
    port (
        X          : in std_logic_vector(N-1 downto 0); -- Operando de entrada 1 (formato flotante)
        Y          : in std_logic_vector(N-1 downto 0); -- Operando de entrada 2 (formato flotante)
        ctrl       : in std_logic; -- Control de operaci n: '0' para suma, '1' para resta
        Z          : out std_logic_vector(N-1 downto 0) -- Resultado de la operaci n (formato flotante)
    );
end fp_subadd;

architecture behavioral of fp_subadd is

    -- Constantes relacionadas al formato
    constant EXC    : natural := 2**(NE-1)-1; -- Valor del sesgo (bias) para el exponente
    constant NF     : natural := N-NE-1; -- Ancho de la mantisa
    constant EXCESO : signed(NE-1 downto 0) := to_signed(EXC, NE);
    constant E_MAX  : signed(NE downto 0) := to_signed(2**(NE)-2, NE+1); -- M ximo valor permitido del
        exponente
    constant E_MIN  : signed(NE downto 0) := to_signed(0, NE+1);         -- M nimo valor permitido del
        exponente

    -- Se ales auxiliares
    signal Y_aux    : std_logic_vector(N-1 downto 0) := (others => '0');

    -- Campos de exponente extendidos a NE+1 bits para manejar signos en la resta
    signal ex_aux   : unsigned(NE downto 0) := (others => '0');
    signal ey_aux   : unsigned(NE downto 0) := (others => '0');
    signal resta_E  : unsigned(NE downto 0) := (others => '0'); -- Diferencia entre exponentes

    -- Operandos alineados para la operaci n
    signal X_p      : unsigned(N-1 downto 0) := (others => '0');
    signal Y_p      : unsigned(N-1 downto 0) := (others => '0');

    -- Campos descompuestos de los operandos
    signal sx_p     : std_logic; -- Bit de signo de X_p
    signal sy_p     : std_logic; -- Bit de signo de Y_p
    signal ex_p     : unsigned(NE-1 downto 0) := (others => '0'); -- Exponente de X_p
    signal ey_p     : unsigned(NE-1 downto 0) := (others => '0'); -- Exponente de Y_p
    signal fx_p     : unsigned(NF-1 downto 0) := (others => '0'); -- Mantisa de X_p
    signal fy_p     : unsigned(NF-1 downto 0) := (others => '0'); -- Mantisa de Y_p
    signal mx_p     : unsigned(NF downto 0) := (others => '0'); -- Significando (mantisa normalizada) de
        X_p
    signal my_p     : unsigned(NF downto 0) := (others => '0'); -- Significando (mantisa normalizada) de
        Y_p

    -- Preparaci n para el alineamiento y suma de los significandos
    -- Se reserva espacio suficiente para manejar el peor caso de desplazamiento entre exponentes
    signal mx_2p    : unsigned(NF+2**(NE)-1 downto 0) := (others => '0');
```

```
    signal my_2p    : unsigned(NF+2**(NE)-1 downto 0) := (others => '0');
    signal mx_3p    : unsigned(NF+2**(NE)+1 downto 0) := (others => '0');
    signal my_3p    : unsigned(NF+2**(NE)+1 downto 0) := (others => '0');
    signal mx_4p    : unsigned(NF+2**(NE)+1 downto 0) := (others => '0');
    signal my_4p    : unsigned(NF+2**(NE)+1 downto 0) := (others => '0');

    signal suma     : unsigned(NF+2**(NE)+1 downto 0) := (others => '0'); -- Resultado crudo
    signal suma_p   : unsigned(NF+2**(NE)+1 downto 0) := (others => '0'); -- Resultado positivo (
        complementado si fue negativo)

    signal fz_aux   : unsigned(NF+2**(NE)+1 downto 0) := (others => '0');
    signal fz_aux_p : unsigned(NF-1 downto 0) := (others => '0');
    signal fz       : unsigned(NF-1 downto 0) := (others => '0');
    signal ez_aux   : signed(NE downto 0) := (others => '0');
    signal ez       : signed(NE-1 downto 0) := (others => '0');
    signal sz       : std_logic := '0'; -- Bit de signo del resultado final

    -- Indicadores de si es necesario complementar los operandos
    signal comp_x   : std_logic := '0';
    signal comp_y   : std_logic := '0';

    -- Funci n que retorna la posici n del primer '1' m s significativo
    function find_one (x0: std_logic_vector) return integer is
        variable found  : boolean;
        variable index  : integer;
    begin
        found := False;

        for i in x0'length-1 downto 0 loop
            if x0(i) = '1' and not found then
                found := True;
                index := i;
            end if;
        end loop;

        if index < 0 then
            index := 0;
        end if;

        return index;
    end function;

begin

    -- Negaci n condicional del operando Y si se desea realizar una resta
    Y_aux <= not(Y(N-1)) & Y(N-2 downto 0) when ctrl = '1' else Y;

    -- C lculo de exponentes con bit adicional para comparaci n
    ex_aux <= '0' & unsigned(X(NF+NE-1 downto NF));
    ey_aux <= '0' & unsigned(Y_aux(NF+NE-1 downto NF));

    resta_E <= ex_aux - ey_aux;

    -- Determinaci n del operando con mayor exponente. Se reordenan si es necesario.
    X_p <= unsigned(Y_aux) when resta_E(NE) = '1' else unsigned(X);
    Y_p <= unsigned(X)     when resta_E(NE) = '1' else unsigned(Y_aux);

    -- Extracci n de campos
    sx_p <= X_p(NE+NF);
    sy_p <= Y_p(NE+NF);
    ex_p <= X_p(NF+NE-1 downto NF);
    ey_p <= Y_p(NF+NE-1 downto NF);
    fx_p <= X_p(NF-1 downto 0);
    fy_p <= Y_p(NF-1 downto 0);
    mx_p <= '1' & fx_p;
    my_p <= '1' & fy_p;

    -- Determinaci n de si se requiere complemento a 2
    comp_x <= '1' when sx_p = '1' else '0';
    comp_y <= '1' when sy_p = '1' else '0';

    -- Preparaci n de significandos para el desplazamiento
    mx_2p(NF downto 0) <= mx_p;
    my_2p(NF downto 0) <= my_p;

    -- Alineaci n de los significandos de acuerdo a los exponentes
    mx_3p <= '0' & '0' & (mx_2p sll to_integer(ex_p));
```

```
    my_3p <= '0' & '0' & (my_2p sll to_integer(ey_p));

    -- Aplicaci n de complemento si es necesario
    mx_4p <= (not(mx_3p) + 1) when comp_x = '1' else mx_3p;
    my_4p <= (not(my_3p) + 1) when comp_y = '1' else my_3p;

    -- Suma efectiva de los significandos
    suma <= mx_4p + my_4p;

    -- Determinaci n del signo del resultado final
    sz <= suma(NF+2**(NE)+1);

    -- Aplicaci n de complemento si el resultado fue negativo
    suma_p <= suma when sz = '0' else (not(suma) + 1);

    -- Normalizaci n del resultado
    fz_aux   <= suma_p sll (suma_p'length - find_one(std_logic_vector(suma_p)));
    fz_aux_p <= fz_aux(fz_aux'length-1 downto fz_aux'length-NF);

    -- C lculo del nuevo exponente
    ez_aux <= to_signed(find_one(std_logic_vector(suma_p)) - NF, NE+1);

    -- Saturaci n del exponente y ajuste de la mantisa
    ez <= E_MAX(NE-1 downto 0) when to_integer(ez_aux) > to_integer(E_MAX) else
          E_MIN(NE-1 downto 0) when to_integer(ez_aux) < to_integer(E_MIN) else
          ez_aux(NE-1 downto 0);

    fz <= (others => '1') when to_integer(ez_aux) > to_integer(E_MAX) else
          (others => '0') when to_integer(ez_aux) < to_integer(E_MIN) else
          fz_aux_p;

    -- Construcci n del resultado final
    Z <= sz & std_logic_vector(ez) & std_logic_vector(fz);

end architecture behavioral;
```

Listing 2: Código VHDL del multiplicador

4. Análisis de los Resultados

Para validar los resultados de los códigos, se implementaron diversos bancos de prueba que permitieron realizar simulaciones utilizando GHDL y GTKWave. Dado que se cuenta con múltiples archivos de prueba, a continuación se presenta únicamente una corrida correspondiente a un archivo de prueba representativo.

4.1. Test Bench: Multiplicador

TB_MULTIPLICADOR.VHD

```
-- TP 2
-- Materia: Sistemas digitales
-- Alumno: Batallan David Leonardo
-- Padron: 97529

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use std.textio.all;

entity tb_fpmul is
end entity tb_fpmul;

architecture tb_arch of tb_fpmul is

    constant FILE_PATH  : string := "../../Archivos_de_Prueba/fmul_21_7.txt";
    constant TCK        : time := 20 ns; -- Ciclo de reloj
    constant F_SIZE     : natural := 21; -- Bits de la mantisa
    constant EXP_SIZE   : natural := 7;  -- Bits del exponente
    constant WORD_SIZE  : natural := EXP_SIZE + F_SIZE + 1; -- Total de bits (incluye bit de signo)

    signal clk      : std_logic := '0';
    signal x_file   : std_logic_vector(WORD_SIZE-1 downto 0) := (others => '0');
    signal y_file   : std_logic_vector(WORD_SIZE-1 downto 0) := (others => '0');
    signal z_file   : std_logic_vector(WORD_SIZE-1 downto 0) := (others => '0');
```

```
    signal z_dut    : std_logic_vector(WORD_SIZE-1 downto 0) := (others => '0');

    signal ciclos   : integer := 0;
    signal errores  : integer := 0;

    file datos : text open read_mode is FILE_PATH;

begin

    clk <= not(clk) after TCK/2; -- Generador de reloj

    Test_Sequence: process
        variable l   : line;
        variable ch  : character := ' ';
        variable aux : integer;
    begin
        while not(endfile(datos)) loop
            wait until rising_edge(clk);
            -- Incremento de contador de ciclos (uso opcional para depuraci n)
            ciclos <= ciclos + 1;
            -- Lectura de una l nea del archivo de pruebas
            readline(datos, l);
            -- Lectura del primer operando (X) desde la l nea
            read(l, aux);
            x_file <= std_logic_vector(to_unsigned(aux, WORD_SIZE));
            -- Lectura del car cter separador
            read(l, ch);
            -- Lectura del segundo operando (Y)
            read(l, aux);
            y_file <= std_logic_vector(to_unsigned(aux, WORD_SIZE));
            -- Lectura del siguiente separador
            read(l, ch);
            -- Lectura del valor esperado de salida (Z)
            read(l, aux);
            z_file <= std_logic_vector(to_unsigned(aux, WORD_SIZE));
        end loop;

        file_close(datos); -- Cierre del archivo de entrada

        -- Finalizaci n expl cita de la simulaci n
        assert false report
            "Fin de la simulacion" severity failure;

    end process Test_Sequence;

    -- Instancia del m dulo a probar (DUT)
    DUT: entity work.fp_mul(behavioral)
    generic map(
        N => WORD_SIZE,
        NE => EXP_SIZE
    )
    port map(
        X => x_file,
        Y => y_file,
        Z => z_dut
    );

    -- Proceso de verificaci n: compara la salida del DUT con la referencia
    verificacion: process(clk)
    begin
        if rising_edge(clk) then
            assert to_integer(unsigned(z_file)) = to_integer(unsigned(z_dut)) report
                "Error: Salida del DUT no coincide con referencia (salida del DUT = " &
                integer'image(to_integer(unsigned(z_dut))) &
                ", salida del archivo = " &
                integer'image(to_integer(unsigned(z_file))) & ")"
                severity warning;

            -- Contador de errores si hay discrepancia
            if to_integer(unsigned(z_file)) /= to_integer(unsigned(z_dut)) then
                errores <= errores + 1;
            end if;
        end if;
    end process;

end architecture tb_arch;
```

Listing 3: Código VHDL del multiplicador

4.2. Test Bench: Sumador Restador

TB_SUMADOR_RESTADOR.VHD

```
-- TP 2
-- Materia: Sistemas digitales
-- Alumno: Batallan David Leonardo
-- Padron: 97529

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use std.textio.all;

entity tb_fpsubadd is
end entity tb_fpsubadd;

architecture tb_arch of tb_fpsubadd is

    -- Ruta al archivo de prueba con los casos de test
    constant FILE_PATH  : string := "../../Archivos_de_Prueba/fsub_12_6.txt";

    -- Par metros de reloj y formato flotante
    constant TCK        : time := 20 ns; -- Per odo del reloj
    constant F_SIZE     : natural := 12; -- Bits para la mantisa
    constant EXP_SIZE   : natural := 6;  -- Bits para el exponente
    constant WORD_SIZE  : natural := EXP_SIZE + F_SIZE + 1; -- Tama o total de palabra (signo + exponente
        + mantisa)

    -- Se ales internas
    signal clk      : std_logic := '0';
    signal ctrl_tb  : std_logic := '1'; -- Control para la operaci n ('1' = resta)

    -- Operandos y resultado esperados desde archivo
    signal x_file   : std_logic_vector(WORD_SIZE-1 downto 0) := (others => '0');
    signal y_file   : std_logic_vector(WORD_SIZE-1 downto 0) := (others => '0');
    signal z_file   : std_logic_vector(WORD_SIZE-1 downto 0) := (others => '0');

    -- Resultado producido por el DUT
    signal z_dut    : std_logic_vector(WORD_SIZE-1 downto 0) := (others => '0');

    -- Contadores de ciclos y errores
    signal ciclos   : integer := 0;
    signal errores  : integer := 0;

    -- Archivo de texto con los vectores de prueba
    file datos : text open read_mode is FILE_PATH;

begin

    -- Generador de reloj
    clk <= not(clk) after TCK / 2;

    -- Proceso de lectura y aplicaci n de vectores de prueba
    Test_Sequence: process
        variable l   : line;
        variable ch  : character := ' ';
        variable aux : integer;
    begin
        while not(endfile(datos)) loop
            wait until rising_edge(clk);

            -- Incrementa el ciclo ( til  para debugging)
            ciclos <= ciclos + 1;

            -- Lee una l nea del archivo
            readline(datos, l);

            -- Lee los valores en el orden: X (entero), espacio, Y (entero), espacio, Z esperado (entero)
            read(l, aux);
            x_file <= std_logic_vector(to_unsigned(aux, WORD_SIZE));

            read(l, ch); -- Espacio

            read(l, aux);
            y_file <= std_logic_vector(to_unsigned(aux, WORD_SIZE));

```

```
            read(l, ch); -- Espacio

            read(l, aux);
            z_file <= std_logic_vector(to_unsigned(aux, WORD_SIZE));
        end loop;

        file_close(datos); -- Cierre del archivo al finalizar

        -- Finaliza la simulaci n cuando se termina el archivo
        assert false report
            "Fin de la simulaci n" severity failure;

    end process Test_Sequence;

    -- Instanciaci n del DUT (Device Under Test)
    DUT: entity work.fp_subadd(behavioral)
    generic map(
        N  => WORD_SIZE,
        NE => EXP_SIZE
    )
    port map(
        X    => x_file,
        Y    => y_file,
        ctrl => ctrl_tb,
        Z    => z_dut
    );

    -- Proceso de verificaci n autom tica
    verificacion: process(clk)
    begin
        if rising_edge(clk) then
            assert to_integer(unsigned(z_file)) = to_integer(unsigned(z_dut)) report
                "Error: la salida del DUT no coincide con el valor esperado. DUT = " &
                integer'image(to_integer(unsigned(z_dut))) &
                ", esperado = " &
                integer'image(to_integer(unsigned(z_file))) & "."
                severity warning;

            -- Acumula errores para evaluaci n posterior
            if to_integer(unsigned(z_file)) /= to_integer(unsigned(z_dut)) then
                errores <= errores + 1;
            end if;
        end if;
    end process;

end architecture tb_arch;
```

Listing 4: Código VHDL del multiplicador

4.3. Multiplicador

En el caso del multiplicador, el resultado final presenta varios errores en distintos archivos de prueba. No obstante, estos errores se atribuyen tanto a los archivos de prueba como al diseño planteado por la cátedra. En dicho diseño se especifica que no se consideran números denormales, lo cual implica que el campo E, que representa el exponente en formato exceso, puede tomar valores desde 0 hasta $2^{N_E} - 1$, donde N_E es el número de bits de dicho campo. Esta convención difiere de la norma IEEE 754[1], donde el valor $2^{N_E} - 1$ está reservado para casos especiales[2], por lo que el valor máximo representable para E es $2^{N_E} - 2$.

Considerando lo anterior y los resultados obtenidos en la simulación, se observa que el método utilizado para generar los archivos de prueba sí tiene en cuenta esta particularidad. Por ello, los errores detectados se originan al truncar el máximo representable al valor $E = 2^{N_E} - 1$, mientras que el archivo indica un resultado con $E = 2^{N_E} - 2$.

Además, sólo se detectó un error relacionado con las mantisas, en el cual la diferencia radica en los valores de estas. En este caso particular, se asume que el error proviene del archivo de prueba.

A continuación, se presenta una imagen de la simulación de la multiplicación para el archivo $fmul\_21\_7.txt$ (Figura 1), así como un ejemplo de los errores mencionados (Figura 2).

[1] Verificar esta afirmación
[2] Investigar cuál es exactamente el caso reservado

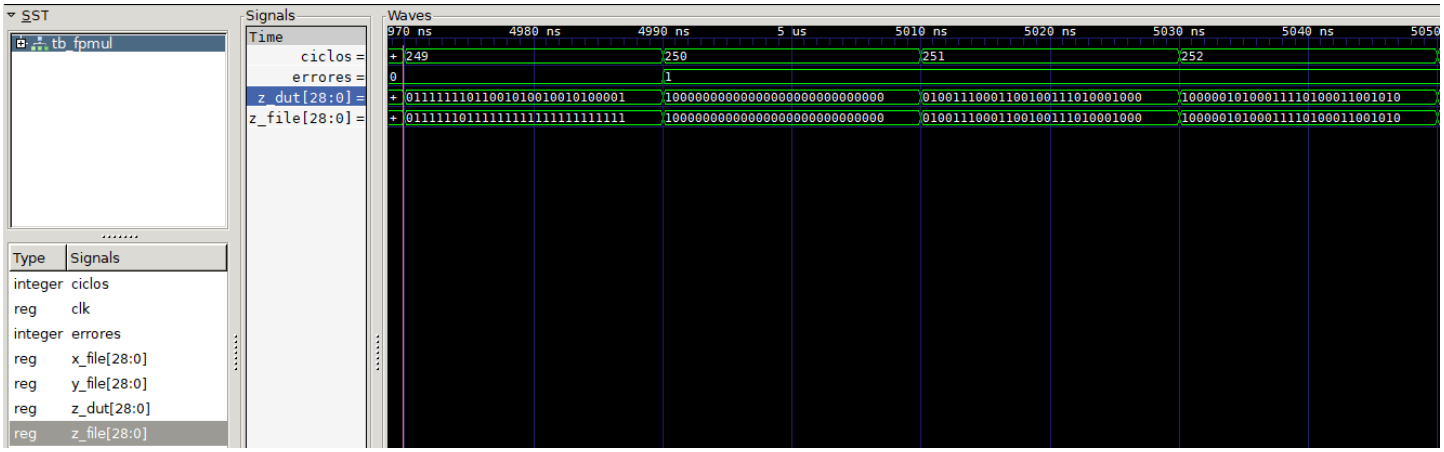

Figura 1: Simulacion multiplicación fmul_21_7.txt

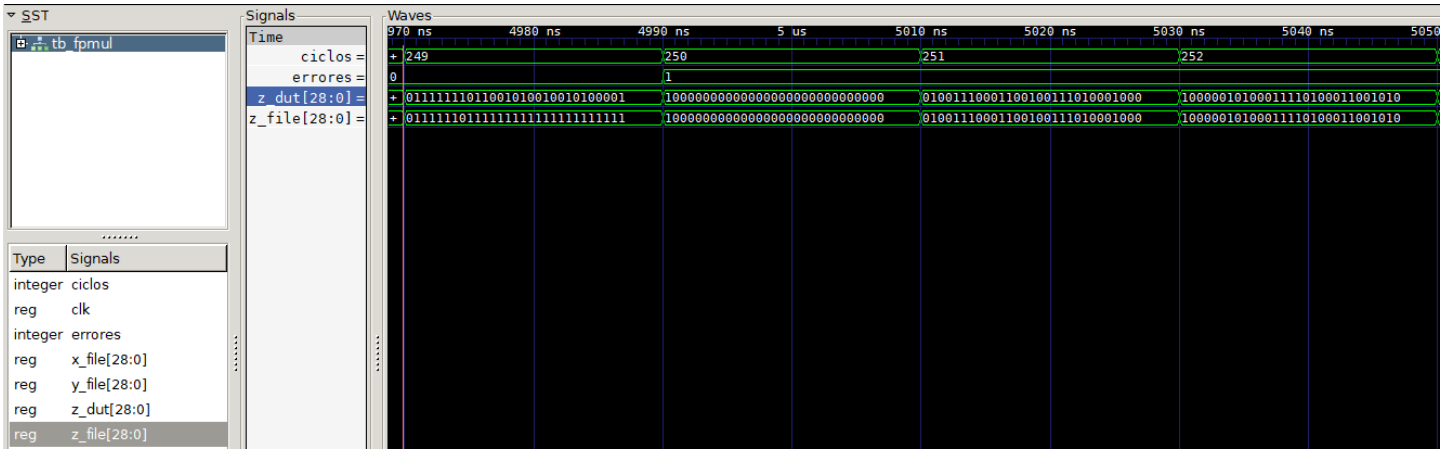

Figura 2: Error

4.3.1. Sumador/Restador

En el caso del sumador/restador, no se observaron errores significativos, salvo por algunos casos puntuales en los que las diferencias se presentan en las mantisas. Al igual que en el caso del multiplicador, estos errores se atribuyen a inconsistencias en los archivos de prueba.

Se incluyen muestras de la simulación para los archivos $fsub\_12\_6.txt$ y $fadd\_12\_6.txt$, que se ilustran en las figuras 3 y 4, respectivamente.

Es importante destacar que en las simulaciones de la suma se detecta un error que ocurre en el tiempo cero, antes del primer flanco de reloj, cuando todas las señales están en '0'. Para el archivo seleccionado como ejemplo, se registran dos errores similares a los mencionados anteriormente, tal como se muestra en la figura 5.

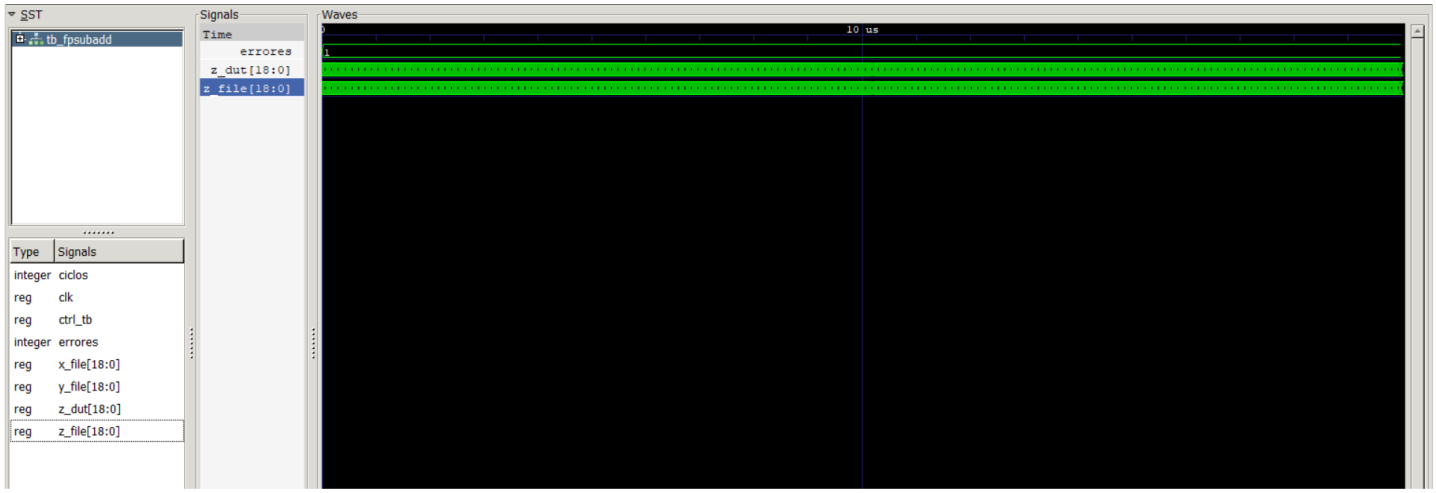

Figura 3: Resta

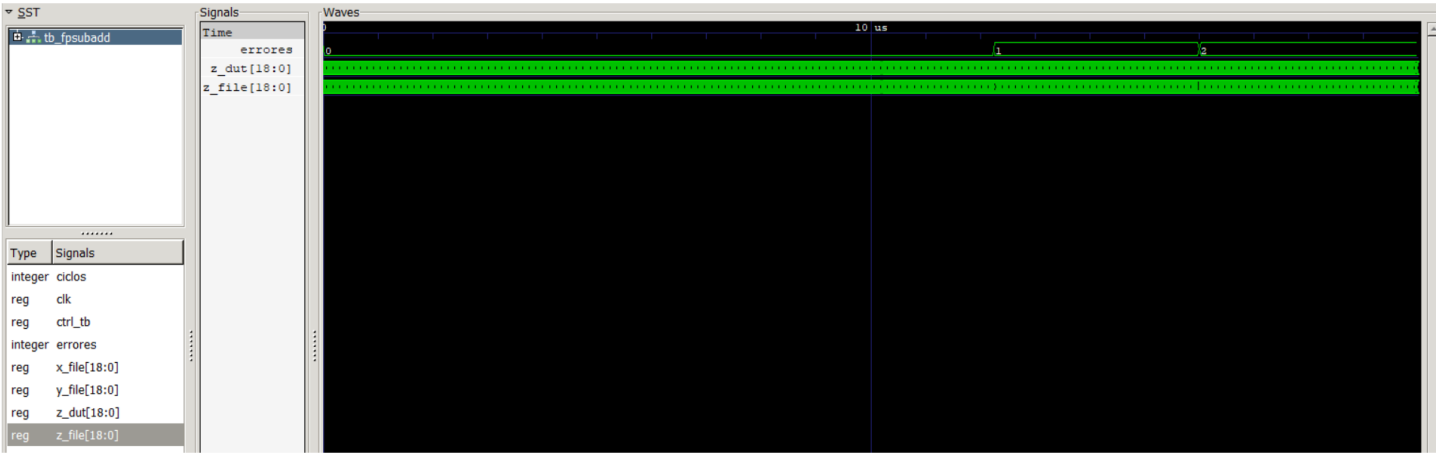

Figura 4: Suma

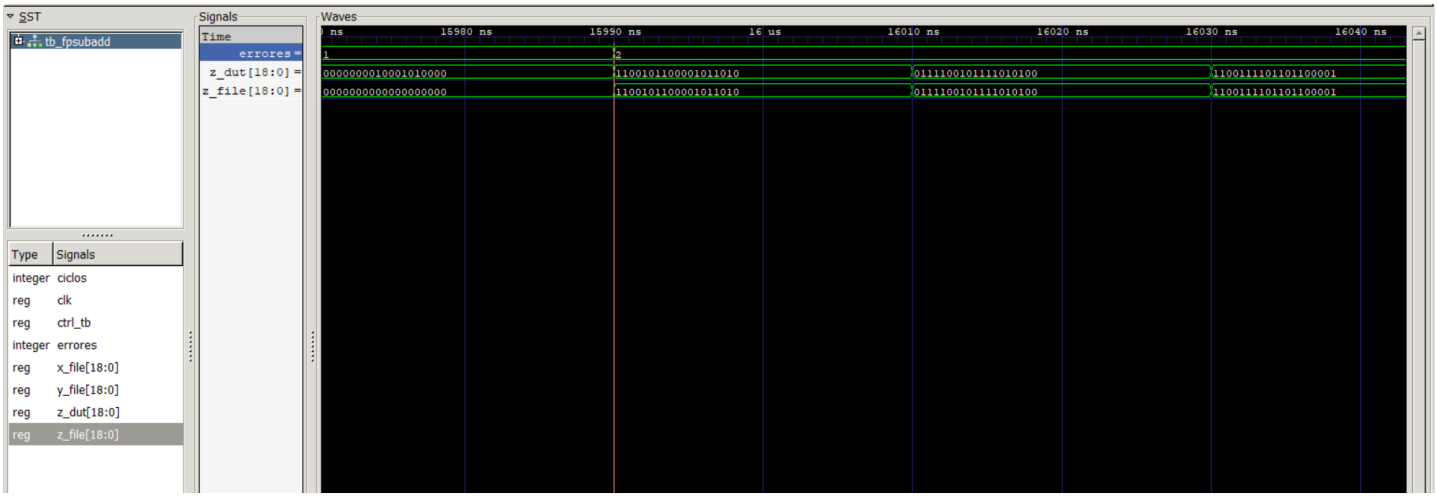

Figura 5: Suma error

5. Sintesis

Para esta sección, se llevó a cabo la síntesis del diseño sobre el dispositivo FPGA `xc7a15tftg256-1` utilizando la herramienta Vivado. Este software permite visualizar cómo se implementa el circuito descrito en VHDL dentro del dispositivo.

En cuanto a los esquemáticos generados, las imágenes resultaron ser de un tamaño considerablemente grande, lo cual dificulta su inclusión adecuada en este informe. Por este motivo, no se adjuntan en el presente documento, ya que su incorporación no aportaría valor significativo.

5.1. Multiplicación

El esquemático RTL se presenta en la Figura 6. En cambio, el esquemático correspondiente a la implementación física resultó ser demasiado extenso para incluirlo en este informe.

Otro resultado relevante es el resumen de recursos utilizados en el diseño, el cual detalla la cantidad de Flip-Flops, LUTs y puertos de entrada/salida (IO) necesarios para la implementación en el dispositivo. Este resumen, junto con el reporte de tiempos, se muestra en las Figuras7 y 8, respectivamente.

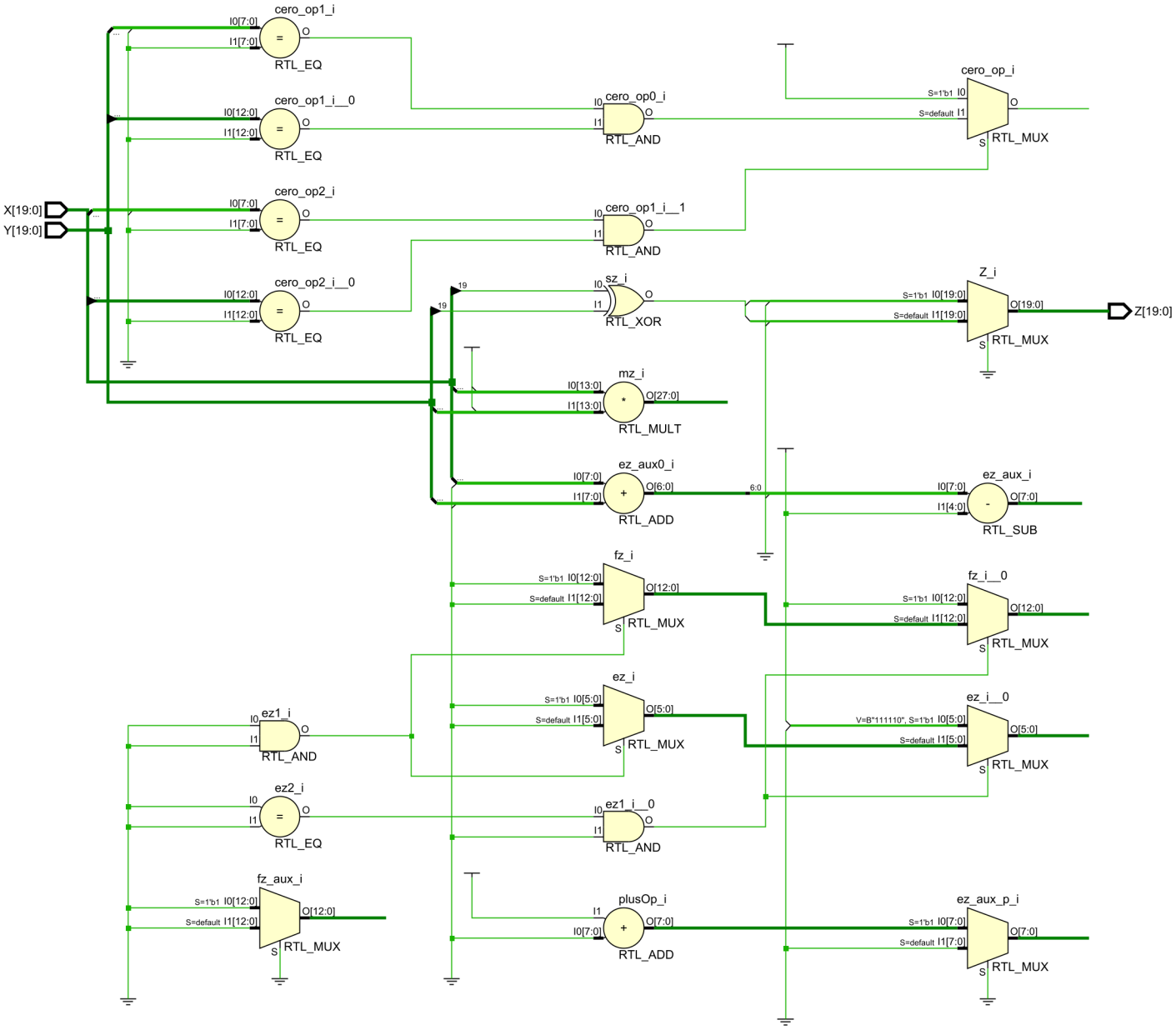

Figura 6: Síntesis Multiplicador

Resource	Utilization	Available	Utilization %
LUT	42	10400	0.40
DSP	1	45	2.22
IO	60	170	35.29

Figura 7: Recursos multiplicador

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	inf	Worst Hold Slack (WHS):	inf	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	NA
Total Number of Endpoints:	20	Total Number of Endpoints:	20	Total Number of Endpoints:	NA

There are no user specified timing constraints.

Figura 8: Resumen de tiempos

5.2. Suma/Resta

A diferencia del multiplicador, tanto los esquemáticos RTL como los de implementación resultaron demasiado extensos para incluirlos en este informe.

Finalmente, se presenta el resumen de recursos utilizados y el reporte de tiempos en las Figuras 9 y 10, respectivamente.

Resource	Utilization	Available	Utilization %
LUT	840	10400	8.08
IO	61	170	35.88

Figura 9: Recursos sumador

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	inf	Worst Hold Slack (WHS):	inf	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	NA
Total Number of Endpoints:	20	Total Number of Endpoints:	20	Total Number of Endpoints:	NA

There are no user specified timing constraints.

Figura 10: Resumen de tiempos sumador

6. Conclusiones

Para concluir, es importante destacar que **VHDL es un lenguaje de descripción de hardware**. Aunque esto ya se había mencionado en el primer trabajo de la materia, en esta oportunidad se pudo apreciar con mayor profundidad al diseñar un circuito más complejo que el anterior.

Al trabajar con VHDL, se comprende que lo que realmente se está haciendo es describir conexiones físicas y comportamientos reales del hardware, lo cual implica un cambio radical respecto a pensar en términos de lenguajes de programación tradicionales.

Además, resulta interesante observar cómo el simulador ejecuta la lógica combinacional: realiza todas las asignaciones y operaciones en una sucesión de tiempos infinitesimales $\delta_1, \delta_2, \ldots, \delta_n$, todos contenidos dentro de un intervalo Δ, que está determinado por el período del reloj del sistema.

Finalmente, se logró entender el funcionamiento de una unidad de punto flotante implementada completamente con lógica combinacional, sin necesidad de sincronismo. También se pudo comprobar que, en términos de síntesis, estas unidades consumen una cantidad significativa de recursos en la FPGA, especialmente el sumador/restador.

7. Repositorio del Proyecto

El código fuente y archivos relacionados están disponibles en el siguiente repositorio de GitHub: Repositorio Sistemas Digitales.