FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

Sistemas Digitales

Trabajo Práctico N2

Aritmética de punto flotante

Denise Gayet 100828 dgayet@fi.uba.ar

10 de noviembre de 2023

Sistemas Digitales

Trabajo Práctico N2: Aritmética de punto flotante

Denise Gayet 100828 dgayet@fi.uba.ar

Índice

1. Enunciado

El presente Trabajo Práctico tiene como objetivo aprender a especificar, diseñar, describir una arquitectura, simular, sintetizar e implementar en FPGA algunas funciones de una unidad aritmética de punto flotante.

2. Introducción

En este trabajo se implementaron dos unidades aritméticas de punto flotante: la operación suma/resta y la multiplicación, sobre vectores de largo arbitrario menor a 32 bits (con un campo de exponente y significando que no siguen la norma IEEE 754). En ambos casos, se utilizó como método de redondeo el truncamiento y, en caso de obtener una resultado fuera del rango de operación se aplicó la saturación (llevar el resultado al valor máximo o mínimo posible).

Un número representado mediante punto flotante en sistema binario consta de 3 vectores concatenados: 1 vector de 1 bit que representa el signo (Sx), 1 vector de Ne bits que representa el exponente (Ex), y 1 vector de Nf bits que representa el significando (Fx). Con estos vectores se puede armar el número (X) a representar, como se muestra a continuación:

$$X = (-1)^{Sx} \cdot 2^{(Ex-EXC)} \cdot \left(1 + \frac{Fx}{2^{Nf}}\right) \tag{1}$$

donde tanto Ex como Fx se interpretan sin signo.

EXC representa el excedente y vale: $2^{Ne-1} - 1$, con lo cual, el exponente puede tomar valores dentro del rango: $[-2^{Ne-1} - 1\ ;\ 2^{Ne-1}]$.

2.1. Diagrama esquemático del multiplicador

La multiplicación de dos números representados con punto flotante se puede escribir de la siguiente manera:

$$\begin{aligned} z = &\quad x \cdot y \\ z = &\quad (-1)^{Sx} \cdot 2^{(Ex-EXC)} \cdot \left(1 + \tfrac{Fx}{2^{Nf}}\right) \cdot (-1)^{Sy} \cdot 2^{(Ey-EXC)} \cdot \left(1 + \tfrac{Fy}{2^{Nf}}\right) \\ z = &\quad \left[(-1)^{Sx} \cdot (-1)^{Sy}\right] \cdot \left[2^{(Ex+Ey-EXC)-EXC}\right] \cdot \left[\left(1 + \tfrac{Fx}{2^{Nf}}\right)\left(1 + \tfrac{Fy}{2^{Nf}}\right)\right] \end{aligned} \tag{2}$$

El primer término representa el signo y se pude expresar como: $Sz = Sx$ XOR Sy.

El segundo término se corresponde con el exponente: $Ez^* = Ex + Ey - EXC$, y debe encontrarse dentro del rango definido anteriormente. De caso contrario, se debe llevar el número a un valor de saturación.

Por último, el tercer término se corresponde con la mantisa de Z. Como se puede observar en la ecuación 1, cualquier mantisa de un número representado mediante punto flotante puede tomar valores entre 1 y 2. Sin embargo, la mantisa del resultado tal como está expresada puede tomar valores entre 1 y 4. Se muestra a continuación los posibles escenarios:

$$\begin{aligned} Mz = &\quad AB, \overbrace{ZZZZZZZ...ZZZZZ}^{\text{2Nf bits}} \\ Mz = &\quad 0\ 1, ZZZZZZZ...ZZZZZ \\ Mz = &\quad 1\ 0, ZZZZZZZ...ZZZZZ \\ Mz = &\quad 1\ 1, ZZZZZZZ...ZZZZZ \end{aligned} \tag{3}$$

En caso de el bit A valga '0', el resultado de la mantisa es < 2, por lo tanto, para formar el significando de Z se toman Nf bits desde el bit siguiente al bit B.

En caso de que el bit A valga '1', el resultado de la mantisa es > 2. Para llevar el significando al rango correcto, se **incrementa el exponente**, y se toma el significando de Z desde el bit B.

A continuación, se muestra en la figura 1, el diagrama en bloques del circuito:

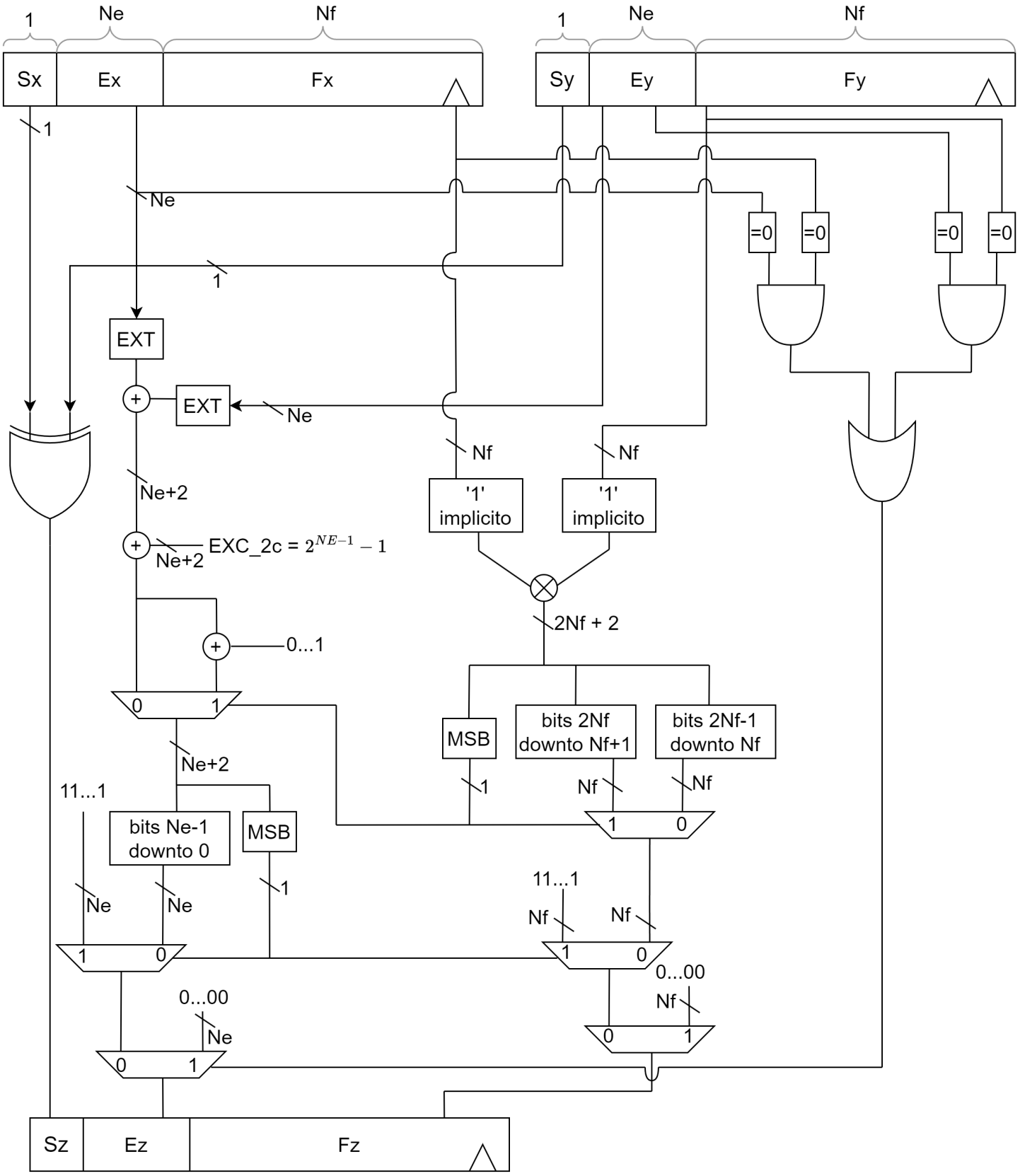

Figura 1: Diagrama en bloques del multiplicador

Este bloque recibe dos entradas X e Y, dos vectores de largo $Nf + Ne + 1$ que se organizan como se muestra en la figura. La salida se calcula como se explicó previamente y se empaqueta en el vector de salida z.

2.2. Diagrama esquemático del sumador

La suma o resta de dos números representados con punto flotante se puede escribir de la siguiente manera:

$$\begin{aligned} z = &\quad x \pm y \\ z = &\quad (-1)^{Sx} \cdot 2^{(Ex-EXC)} \cdot \left(1 + \tfrac{Fx}{2^{Nf}}\right) \pm (-1)^{Sy} \cdot 2^{(Ey-EXC)} \cdot \left(1 + \tfrac{Fy}{2^{Nf}}\right) \\ z = &\quad \left[(-1)^{Sx} \cdot 2^{Ex} \cdot \left(1 + \tfrac{Fx}{2^{Nf}}\right) \pm (-1)^{Sy} \cdot 2^{Ey} \cdot \left(1 + \tfrac{Fy}{2^{Nf}}\right)\right] 2^{-EXC} \end{aligned} \tag{4}$$

Suponiendo que $Ex \geq Ey$, se multiplica y divide la expresión por 2^{Ex} y se obtiene:

$$z = \quad \left[(-1)^{Sx} \cdot \left(1 + \tfrac{Fx}{2^{Nf}}\right) \pm (-1)^{Sy} \cdot \left(1 + \tfrac{Fy}{2^{Nf}}\right) \cdot 2^{Ex-Ey}\right] \cdot 2^{Ex-EXC} \tag{5}$$

A partir del primer término se puede calcular la mantisa del resultado.

Como se supuso que $Ex \geq Ey$, el exponente que multiplica a la mantisa de y está desplazado $Ex - Ey$ hacia la **derecha**. De la misma forma, para realizar el cálculo, se puede desplazar la mantisa de x, una cantidad $Ex-Ey$ bits hacia la **izquierda**, como se muestra en la figura 2.

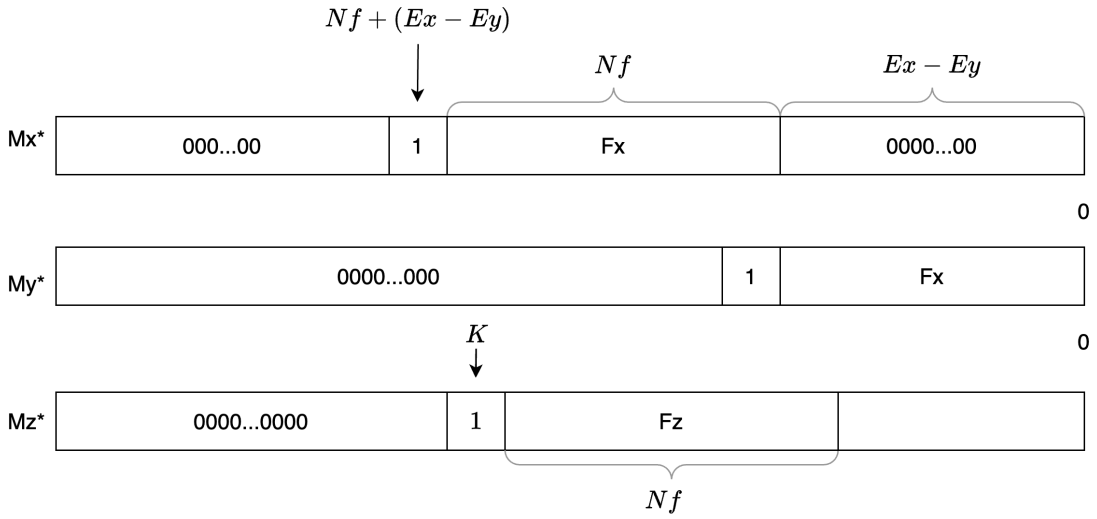

Figura 2: Cálculo de la mantisa del resultado

Luego, basta con hacer la suma binaria de ambos operandos para obtener la mantisa del resultado, teniendo en cuenta que previamente se debe complementar los operandos según fuera necesario para reflejar el signo de cada uno y la operación a realizar. Por lo tanto, el resultado tiene el signo correspondiente y para obtener el significando Fz se debe complementar el vector en caso de ser negativo.

A partir de Mz^* (según se observa en la figura 2), se obtiene el significando de la salida tomando los Nf bits siguiente al primer '1' del vector (este bit será el '1' implícito de la mantisa).

Por último, teniendo en cuenta que en la ecuación 5 se tiene como exponente provisorio el exponente de x Ex, se puede calcular mediante el resultado de la mantisa cuánto se debe desplazar dicho exponente para obtener Ez: el '1' implícito de x se encuentra en el bit $Nf + (Ex - Ey)$, con lo cual, la diferencia entre dicho bit y el bit del '1' implícito de Mz^* (señalizado en la figura 2 como K) es el desplazamiento que se debe aplicar. En conclusión:

$$Ez = Ex + (K - (Nf + (Ex - Ey))) \tag{6}$$

Cabe aclarar que como se supuso que $Ex \geq Ey$, se debe buscar previamente a realizar el cálculo el operando con mayor exponente e intercambiar los operandos si fuera el caso en que $Ey > Ex$.

A continuación, en la figura 3 se muestra el diagrama en bloques de esta unidad aritmética:

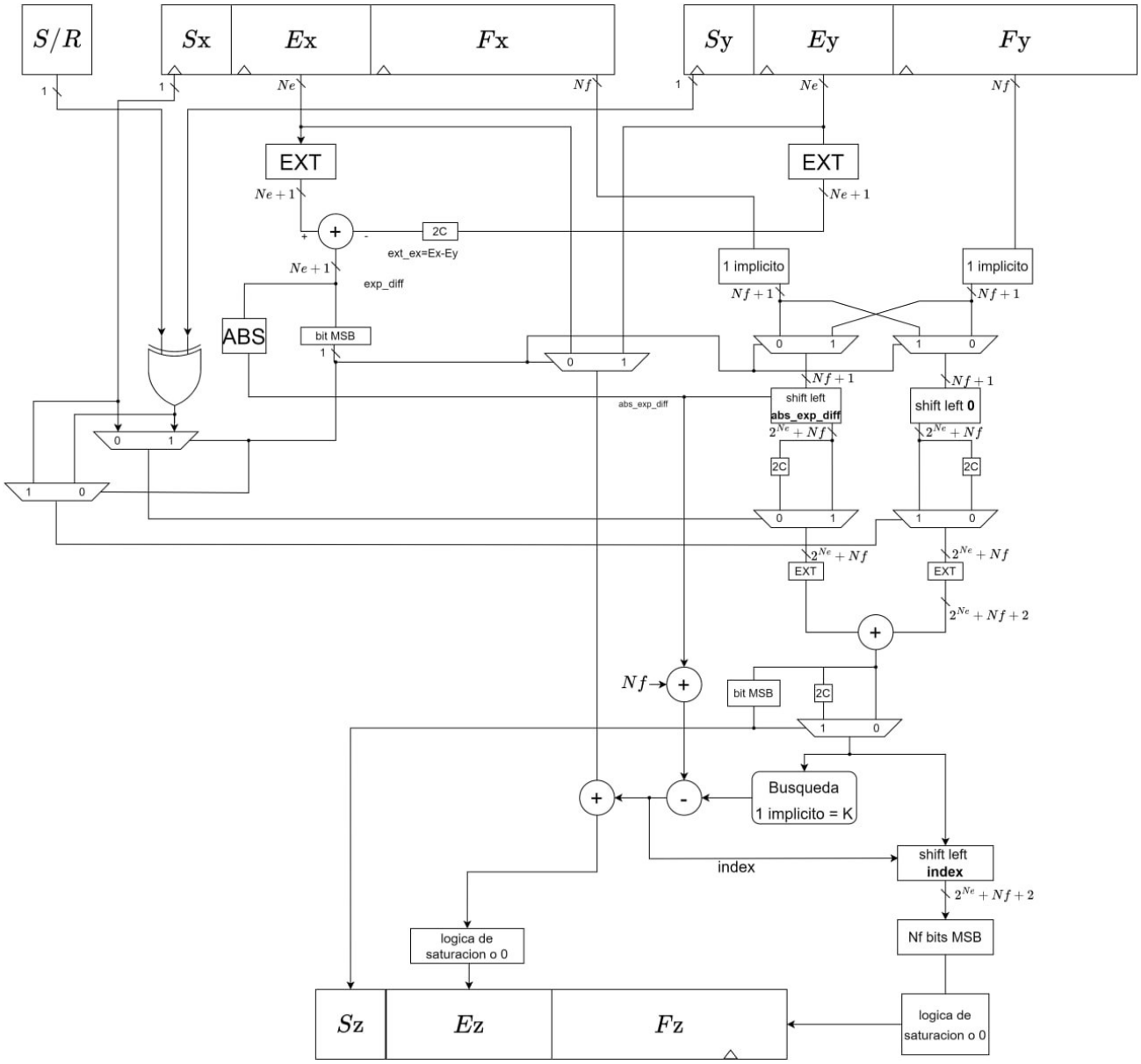

Figura 3: Diagrama en bloques del sumador/restar

En este caso se obvió la lógica particular para cuando algún operando es 0 y la lógica de saturación para no aumentar innecesariamente el tamaño del diagrama.

3. Descripción en VHDL

3.1. Generalidades

La implementación de ambas unidades aritiméticas es puramente combinacional. Sin embargo, se registraron tanto las entradas como la salida para evitar posibles desincronizaciones. Estos módulos que están sincronizados con el clock, reciben como entrada la señal de clock, y transicionan en el flanco **ascendente** del mismo.

Además, dichos módulos tienen de entrada la señal de reset, que se activa al estar en alto y pone a todos los módulos en su estado inicial. El reset es **asincrónico**, es decir, funciona de manera independiente al flanco del clock. Ambas señales son globales.

Para ambas unidades aritméticas se utilizaron los siguientes valores de saturaciones:

- Para el exponente:

- Saturación por valor máximo: $2^{Ne} - 1$
- Saturación por valor mínimo: 0

- Para el significando:
  - Saturación por valor máximo: $2^{Nf} - 1$
  - Saturación por valor mínimo: 0

Cabe destacar que la convención tomada difiere de la utilizada en los archivos de *testbench* (En el cual la saturación del exponente se toma como $2^{Ne} - 2$), por lo que se tuvo que modificar la misma para poder evaluar el correcto funcionamiento de la implimentación.

3.2. Multiplicador

Como se puede observar en el diagrama en bloques de la mulitpilicación, la implementación es fácilmente separable en distintos módulos, siendo necesario utilizar unas pocas señales para comunicarlos. Por lo tanto, se dividió la descripción en tres módulos distintos: el cálculo del signo, el cálculo del exponente y el cálculo del significando. Luego, se combinaron en un único modulo superior que empaqueta el resultado.

3.2.a. Exponente

En primer lugar, se muestra la descripción en VHDL para el cálculo del exponente:

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  entity exponent is
6      generic (
7          NE : natural := 7
8      );
9      port (
10         Ex : in std_logic_vector(NE-1 downto 0);
11         Ey : in std_logic_vector(NE-1 downto 0);
12         is_0 : in std_logic;
13         add1 : in std_logic;
14         Ez : out std_logic_vector(NE-1 downto 0);
15         saturation : out std_logic_vector (1 downto 0)
16     );
17 end;
18
19 architecture behavioral of exponent is
20     signal Ex_ext : unsigned(NE+1 downto 0);
21     signal Ey_ext : unsigned(NE+1 downto 0);
22     signal Ez_pre : unsigned(NE+1 downto 0);
23     signal aux_add1 : unsigned(NE+1 downto 0);
24     signal aux_Ez : unsigned(NE-1 downto 0);
25
26     signal EXC_2C :  unsigned(NE+1 downto 0) := not to_unsigned(2**(NE-1)-1, NE+2) + 1;
27     signal OVERFLOW_VALUE : unsigned(NE-1 downto 0) := to_unsigned(2**(NE)-2, NE);
28     signal ZERO_VALUE : unsigned(NE-1 downto 0) := to_unsigned(0, NE);
29
30 begin
31     Ex_ext <= unsigned("00" & Ex);
32     Ey_ext <= unsigned("00" & Ey);
33     aux_add1 <= to_unsigned(1, NE+2) when add1='1' else to_unsigned(0, NE+2);
34     Ez_pre <= Ex_ext + Ey_ext + EXC_2C + aux_add1;
35     saturation <= std_logic_vector(Ez_pre(NE+1 downto NE));
36     aux_Ez <= Ez_pre(NE-1 downto 0) when Ez_pre(NE+1 downto NE)="00" else
37               OVERFLOW_VALUE when Ez_pre(NE+1 downto NE)="01" else ZERO_VALUE;
38     Ez <= std_logic_vector(aux_Ez) when is_0='0' else std_logic_vector(ZERO_VALUE);
39 end;
```

3.2.b. Significando

Luego, el código para el cálculo del significando.

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

entity significand is
    generic (
        NF : natural := 21
    );
    port (
        Fx : in std_logic_vector(NF-1 downto 0);
        Fy : in std_logic_vector(NF-1 downto 0);
        is_0 : in std_logic;
        saturation : in std_logic_vector(1 downto 0);
        Fz : out std_logic_vector(NF-1 downto 0);
        add1 : out std_logic
    );
end;


architecture behavioral of significand is
    signal Fx_ext : unsigned(NF downto 0);
    signal Fy_ext : unsigned(NF downto 0);
    signal Fz_ext : unsigned(2*NF+1 downto 0);
    signal Fz_aux : unsigned(NF-1 downto 0);
    signal Fz_pre : unsigned(NF-1 downto 0);
    signal Fz_MSB : std_logic;
    signal aux_sat : std_logic;

    constant ZERO_VALUE : unsigned(NF-1 downto 0) := to_unsigned(0, NF);
    constant SAT_VALUE : unsigned(NF-1 downto 0) := to_unsigned(2**NF -1, NF);

begin

    Fx_ext <= unsigned('1' & Fx);
    Fy_ext <= unsigned('1' & Fy);
    Fz_ext <= Fx_ext * Fy_ext;
    Fz_MSB <= Fz_ext(2*NF+1);
    Fz_aux <= Fz_ext(2*NF downto NF+1) when Fz_MSB='1' else Fz_ext(2*NF-1 downto NF);
    Fz_pre <= Fz_aux when (saturation="10" or saturation="00") else
              SAT_VALUE when saturation="01" else ZERO_VALUE;

    aux_sat <='0' when (is_0='0') else '1';
    Fz <= std_logic_vector(Fz_pre) when is_0='0' else std_logic_vector(ZERO_VALUE);
    add1 <= Fz_MSB;
end;
```

3.2.c. Módulo superior

Por último, se muestra el módulo superior que combina el resto de los módulos. El cálculo del signo es muy sencillo, por lo que no amerita un archivo separado para implementarlo.

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

entity fp_multiplication is
    generic (
        NE : natural := 7;
        NF : natural := 21
    );
    port (
```

```
        clk : in std_logic;
        rst : in std_logic;
        x : in std_logic_vector(NF+NE downto 0);
        y : in std_logic_vector(NF+NE downto 0);
        z : out std_logic_vector(NF+NE downto 0)
    );
end;

architecture behavioral of fp_multiplication is
    signal is_0 : std_logic;
    signal sign : std_logic;
    signal add_1 : std_logic;
    signal saturation : std_logic_vector(1 downto 0);

    signal x_reg : std_logic_vector(NF+NE downto 0);
    signal y_reg : std_logic_vector(NF+NE downto 0);
    signal z_reg : std_logic_vector(NF+NE downto 0);

begin

    regX: entity work.reg(behavioral)
        generic map(NF+NE+1)
        port map(
            clk => clk,
            rst => rst,
            ena => '1',
            D => x,
            Q => x_reg
        );

    regY: entity work.reg(behavioral)
        generic map (NF+NE+1)
        port map (
            clk => clk,
            rst => rst,
            ena => '1',
            D => y,
            Q => y_reg
        );

    regZ: entity work.reg(behavioral)
        generic map (NF+NE+1)
        port map (
            clk => clk,
            rst => rst,
            ena => '1',
            D => z_reg,
            Q => z
        );

    IS_0_INST : is_0 <= '1' when (unsigned(x_reg(NF+NE-1 downto 0))=0 OR unsigned(y_reg(NF+NE-1 down
                        else '0';

    SIGN_INST : sign <= x_reg(NF+NE) XOR y_reg(NF+NE);

    EXP_INST : entity work.exponent(behavioral)
                generic map (
                    NE => NE
                )
                port map (
                    Ex => x_reg(NF+NE-1 downto NF),
                    Ey => y_reg(NF+NE-1 downto NF),
                    is_0 => is_0,
                    add1 => add_1,
```

```
                    Ez => z_reg(NF+NE-1 downto NF),
                    saturation => saturation
                );

    SIG_INST : entity work.significand(behavioral)
                generic map (
                    NF => NF
                )
                port map (
                    Fx => x_reg(NF-1 downto 0),
                    Fy => y_reg(NF-1 downto 0),
                    is_0 => is_0,
                    saturation => saturation,
                    Fz => z_reg(NF-1 downto 0),
                    add1 => add_1
                );

    Z_sign_inst: z_reg(NF+NE) <= sign;
end;
```

3.3. Sumador

Dado que el sumador es sustancialmente más complejo que el multiplicador, se optó por implementarlo en un único archivo, para evitar las múltiples señales que se deberían comunicar entre el cálculo del exponente, el significando y la suma.

A continuación se muestra la descripción en VHDL del sumador/restador:

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

entity fp_add is
    generic (
        NE : natural := 7;
        NF : natural := 21
    );
    port (
        clk : in std_logic;
        rst : in std_logic;
        add_sub : in std_logic;
        x : in std_logic_vector(NF+NE downto 0);
        y : in std_logic_vector(NF+NE downto 0);
        z : out std_logic_vector(NF+NE downto 0)
    );
end;

architecture behavioral of fp_add is
    function search_implicit_1(value: unsigned(2**NE+NF downto 0)) return integer is
        variable found : boolean := false;
        variable ind : integer range 0 to 2**NE+NF := 0;

    begin
        for i in (2**NE + NF) downto 0 loop
            if not found then
                ind := i;
            end if;
            if value(i) = '1' then
                found := true;
            end if;
        end loop;

        return ind;
    end;

```

```
38
39      signal is_0 : std_logic_vector(1 downto 0) := "00";
40      signal saturation : std_logic_vector(1 downto 0) := "00";
41
42      signal x_reg : std_logic_vector(NF+NE downto 0) := (others => '0');
43      signal y_reg : std_logic_vector(NF+NE downto 0) := (others => '0');
44      signal z_reg : std_logic_vector(NF+NE downto 0) := (others => '0');
45
46      signal Sx : std_logic := '0';
47      signal Fx : std_logic_vector(NF-1 downto 0) := (others => '0');
48      signal Ex : std_logic_vector(NE-1 downto 0) := (others => '0');
49
50      signal Sy : std_logic := '0';
51      signal Fy : std_logic_vector(NF-1 downto 0) := (others => '0');
52      signal Ey : std_logic_vector(NE-1 downto 0) := (others => '0');
53
54      signal Sz : std_logic := '0';
55      signal Fz : std_logic_vector(NF-1 downto 0) := (others => '0');
56      signal Ez : std_logic_vector(NE-1 downto 0) := (others => '0');
57
58      -- constantes y seniales del calculo del significante
59      constant F_SAT_VALUE : std_logic_vector(NF-1 downto 0) := std_logic_vector(to_unsigned(2**NF-1,
60      constant F_ZERO_VALUE : std_logic_vector(NF-1 downto 0) := std_logic_vector(to_unsigned(0, NF));
61      constant E_SAT_VALUE : std_logic_vector(NE-1 downto 0) := std_logic_vector(to_unsigned(2**NE-1,
62      constant E_ZERO_VALUE : std_logic_vector(NE-1 downto 0) := std_logic_vector(to_unsigned(0, NE));
63
64
65      signal Fx_ext : unsigned(NF downto 0) := (others => '0');
66      signal Fy_ext : unsigned(NF downto 0) := (others => '0');
67
68      signal aligned_Fx : unsigned(NF+2**NE-1 downto 0) := (others => '0');
69      signal aligned_Fy : unsigned(NF+2**NE-1 downto 0) := (others => '0');
70
71      signal Mx : unsigned(NF+2**NE+1 downto 0) := (others => '0');
72      signal My : unsigned(NF+2**NE+1 downto 0) := (others => '0');
73
74      signal Sx_ext : std_logic := '0';
75      signal Sy_ext : std_logic := '0';
76
77      signal Mz_pre : unsigned(NF+2**NE+1 downto 0) := (others => '0');
78      signal Mz : unsigned(NF+2**NE+1 downto 0) := (others => '0');
79      signal Mz_shift : unsigned(NF+2**NE+1 downto 0) := (others => '0');
80      signal Fz_pre : unsigned(NF-1 downto 0) := (others => '0');
81      signal Fz_aux : std_logic_vector(NF-1 downto 0) := (others => '0');
82
83      signal abs_exp_diff : unsigned(NE downto 0) := (others => '0');
84      signal aux_index : integer range 0 to 2**NE+NF := 0;
85      signal shift : integer := 0;
86      signal lower_index : integer range 0 to 2**NE+NF := 20;
87      signal upper_index : integer range 0 to 2**NE+NF := 0;
88
89
90      -- constantes y seniales del calculo del exponente
91
92      signal Ex_ext : unsigned(NE+1 downto 0);
93      signal Ey_ext : unsigned(NE+1 downto 0);
94      signal Ez_pre : unsigned(NE+1 downto 0);
95      signal aux_Ez : std_logic_vector(NE-1 downto 0);
96
97      signal exp_diff : unsigned(NE+1 downto 0);
98      signal index : unsigned(NE+1 downto 0);
99
100 begin
101
```

```
102     regX: entity work.reg(behavioral)
103             generic map(NF+NE+1)
104             port map(
105                 clk => clk,
106                 rst => rst,
107                 ena => '1',
108                 D => x,
109                 Q => x_reg
110             );
111 
112     regY: entity work.reg(behavioral)
113             generic map (NF+NE+1)
114             port map (
115                 clk => clk,
116                 rst => rst,
117                 ena => '1',
118                 D => y,
119                 Q => y_reg
120             );
121 
122     regZ: entity work.reg(behavioral)
123             generic map (NF+NE+1)
124             port map (
125                 clk => clk,
126                 rst => rst,
127                 ena => '1',
128                 D => z_reg,
129                 Q => z
130             );
131 
132     ZERO : is_0 <= "10" when (unsigned(x_reg(NF+NE-1 downto 0))=0 AND unsigned(y_reg(NF+NE-1 downto 
133                    "10" when (unsigned(x_reg(NF+NE-1 downto 0))/=0 AND unsigned(y_reg(NF+NE-1 downto
134                    "11" when (unsigned(x_reg(NF+NE-1 downto 0))=0 AND unsigned(y_reg(NF+NE-1 downto 
135                    "00";
136 
137     EXPONENT :
138         Ex <= x_reg(NF+NE-1 downto NF);
139         Ey <= y_reg(NF+NE-1 downto NF);
140 
141         Ex_ext <= "00" & unsigned(Ex);
142         EY_ext <= "00" & unsigned(Ey);
143 
144         exp_diff <= Ex_ext + (not Ey_ext + 1);
145         index <= unsigned(std_logic_vector(to_signed(shift, NE+2)));
146 
147         Ez_pre <= Ex_ext+index when exp_diff(NE)='0' else Ey_ext+index;
148 
149         -- chequeo por algun operando = 0;
150         aux_Ez <= std_logic_vector(Ez_pre(NE-1 downto 0)) when is_0="00" else
151                   Ex when is_0="01" else
152                   Ey when is_0="10" else
153                   E_ZERO_VALUE;
154 
155         -- saturacion
156         saturation <= std_logic_vector(Ez_pre(NE+1 downto NE));
157         Ez <= aux_Ez when (saturation="00" or saturation="10") else
158               E_SAT_VALUE when (saturation="01") else
159               E_ZERO_VALUE when (saturation="10");
160 
161     SIGNIFICAND :
162 
163         -- tomo el significando de X y de Y
164         Fx <= x_reg(NF-1 downto 0);
165         Fy <= y_reg(NF-1 downto 0);
```

```
166
167         -- elijo Fx* y Fy* segun el MSB de la resta de los exponentes y agrego el 1 implicito
168         Fx_ext <= ('1' & unsigned(Fx)) when exp_diff(NE) = '0' else ('1' & unsigned(Fy));
169         Fy_ext <= ('1' & unsigned(Fy)) when exp_diff(NE) = '0' else ('1' & unsigned(Fx));
170
171         -- en caso de que la operacion sea una resta debo cambiar el signo del operando Y:
172         Sx_ext <= Sx when exp_diff(NE) = '0' else (add_sub XOR Sy);
173         Sy_ext <= add_sub XOR Sy when exp_diff(NE) = '0' else Sx;
174
175         -- armo las alineaciones en funcion de la abs(Ex - Ey)
176         abs_exp_diff <= exp_diff(NE downto 0) when exp_diff(NE) = '0' else (not exp_diff(NE downto 0)
177         aligned_Fy <= to_unsigned(0, 2**NE-1) & Fy_ext;
178         aligned_Fx <= shift_left(to_unsigned(0, 2**NE-1) & Fx_ext, to_integer(abs_exp_diff));
179
180         -- Complemento a 2 correspondiente a cada mantisa en funcion de su signo y si es una resta o
181         Mx <= "00" & unsigned(aligned_Fx) when Sx_ext='0' else ("11" & unsigned(not aligned_Fx + 1));
182         My <= "00" & unsigned(aligned_Fy) when Sy_ext='0' else unsigned( "11" & not aligned_Fy + 1);
183
184         -- Sumo ambas mantisas y si resulta negativa complemento.
185         Mz_pre <=  Mx + My;
186         Mz <= Mz_pre when (Mz_pre(2**NE+NF+1)='0') else (not Mz_pre+1);
187
188         -- Busco el 1 implicito de la mantisa resultante
189         aux_index <= search_implicit_1(Mz(2**NE+NF downto 0));
190         shift <= aux_index - (NF+to_integer(abs_exp_diff));
191
192         -- Defino los indices que corresponden al significando de Z (Fz),
193         -- en caso de que el largo del vector resultante sea < NF, se paddea con 0 a la derecha.
194         upper_index <= aux_index-1 when (aux_index>0) else 0;
195         lower_index <= aux_index-NF when (aux_index-NF > 0) else 0;
196         Mz_shift <= shift_left(Mz, Mz'length - aux_index);
197
198         Fz_pre <= Mz_shift(Mz'length-1 downto Mz'length - NF);
199
200         -- chequeo por algun operando = 0;
201         Fz_aux <= std_logic_vector(Fz_pre) when (is_0="00") else
202                   Fx when (is_0="01") else
203                   Fy when (is_0="10") else
204                   F_ZERO_VALUE;
205
206         -- saturacion
207         Fz <= Fz_aux when (saturation="00" or saturation="10") else
208         F_SAT_VALUE when (saturation="01") else
209         F_ZERO_VALUE when (saturation="10");
210
211     SIGN :
212         Sx <= x_reg(NF+NE);
213         Sy <= y_reg(NF+NE);
214         -- el signo de la salida es el signo de la mantisa resultante
215         -- (dado que complemente cada operando segun fuera necesario)
216         Sz <= Mz_pre(2**NE+NF) when is_0="00" else
217               Sx when is_0="01" else
218               add_sub XOR Sy when is_0="10" else
219               '0';
220
221     z_reg <= Sz & Ez & Fz;
222 end;
```

3.4. Registro

Como se explicó previamente, se registraron las entradas y las salidas en ambas unidades aritméticas. Esto se realizó mediante la utilización de un flip-flop D.

```
1 library IEEE;
```

```
use IEEE.std_logic_1164.all;

entity reg is
    generic(N: integer := 4);
    port (
        clk : in std_logic;
        rst : in std_logic;
        ena : in std_logic;
        D : in std_logic_vector(N-1 downto 0);
        Q : out std_logic_vector(N-1 downto 0)
    );
end;

architecture behavioral of reg is
begin
    process(clk, rst)
    begin
        if rst='1' then
            Q <= (others => '0');
        elsif rising_edge(clk) then
            if ena='1' then
                Q <= D;
            end if;
        end if;
    end process;
end;
```

3.5. Generador de delay

Como se registraron entradas y salidas, al circuito le toma 2 ciclos de reloj para obtener una salida. Sin embargo, el archivo de *testbench* desde el cual se leen los operandos de entrada y salida lo realiza cada un ciclo de reloj. Por lo tanto, se deben alinear las salidas de ambos bloques para poder evaluar el funcionamiento. Para logar esto, se propuso generar un *delay* de 2 ciclos de reloj en el resultado leido del archivo.

```
library ieee;
use ieee.std_logic_1164.all;

entity delay_gen is
    generic(
            N: natural:= 26;
            DELAY: natural:= 0
        );
    port(
        clk: in std_logic;
        rst: in std_logic;
        A: in std_logic_vector(N-1 downto 0);
        B: out std_logic_vector(N-1 downto 0)
    );
end;


architecture behavioral of delay_gen is
    type std_logic_matrix is array(DELAY-1 downto 0) of std_logic_vector(N-1 downto 0);
    signal sr : std_logic_matrix;
begin
    process(clk,rst)
    begin
        if rst='1' then
            sr <= (others=>(others=>'0'));
        elsif clk = '1' and clk'event then
            sr(DELAY-1 downto 1) <= sr(DELAY-2 downto 0);
            sr(0) <= A;
        end if;
    end process;
```

```
31      B <= sr(DELAY-1);
32  end;
```

3.6. Testbench

Estos módulos se usan a modo de banco de pruebas, su utilidad es únicamente durante la simulación, no forma parte de la sintetización del diseño.

Se implementó un testbench para la suma/resta y un testbench para la multiplicación. Ambos leen un archivo provisto por la cátedra, el cual indica línea a línea las entradas y la salida de la operación. Además, se indica en el nombre del archivo el largo de cada campo y la operación a realizar (ej: `mul_NF_NE.txt`).

Para levantar un archivo, se deben cambiar los valores de las constantes NE y NF de modo tal que reflejen los tamaños de los campos de exponente y significando.

A continuación se muestran ambos *testbenchs*.

3.6.a. Multiplicación

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4   use std.textio.all;
5
6
7   entity pf_testbench is
8   end entity pf_testbench;
9
10
11  architecture pf_testbench_arq of pf_testbench is
12      constant TCK: time:= 20 ns; -- periodo de reloj
13      constant DELAY: natural:= 2; -- retardo de procesamiento del DUV
14      constant NF : natural := 21;
15      constant EXP_SIZE_T: natural:= 7; -- tamanio exponente
16      constant WORD_SIZE_T: natural:= NF+EXP_SIZE_T+1; -- tamanio de datos
17      signal clk: std_logic:= '0';
18      signal rst: std_logic:= '1';
19      signal a_file: unsigned(WORD_SIZE_T-1 downto 0):= (others => '0');
20      signal b_file: unsigned(WORD_SIZE_T-1 downto 0):= (others => '0');
21      signal z_file: unsigned(WORD_SIZE_T-1 downto 0):= (others => '0');
22      signal z_del: unsigned(WORD_SIZE_T-1 downto 0):= (others => '0');
23      signal z_duv: std_logic_vector(WORD_SIZE_T-1 downto 0):= (others => '0');
24      signal ciclos: integer := 0; signal errores: integer := 0;
25      -- La senal z_del_aux se define por un problema de conversion
26      signal z_del_aux: std_logic_vector(WORD_SIZE_T-1 downto 0):= (others => '0');
27
28      file datos: text open read_mode is "../testbench/fmul_21_7.txt";
29      -- Declaracion del componente a probar
30      component fp_multiplication is
31          generic(
32                  NE: natural := 7;
33                  NF: natural := 21
34              );
35              port(
36                  clk : in std_logic;
37                  rst : in std_logic;
38                  x : in std_logic_vector(NF+NE downto 0);
39                  y : in std_logic_vector(NF+NE downto 0);
40                  z : out std_logic_vector(NF+NE downto 0)
41                  );
42      end component fp_multiplication ;
43      -- Declaracion de la linea de retardo
```

```
44      component delay_gen is
45          generic(
46              N: natural:= 26;
47              DELAY: natural:= 0
48          );
49          port(
50              clk: in std_logic;
51              rst: in std_logic;
52              A: in std_logic_vector(N-1 downto 0);
53              B: out std_logic_vector(N-1 downto 0)
54          );
55      end component;
56  begin
57      -- Generacion del clock del sistema
58      clk <= not(clk) after TCK/ 2; -- reloj
59      rst <= '0' after 1 ns;
60
61      Test_Sequence: process
62          variable l: line;
63
64          variable ch: character:= ' ';
65          variable aux: integer;
66      begin
67          while not(endfile(datos)) loop
68              wait until rising_edge(clk);
69              -- solo para debugging
70              ciclos <= ciclos + 1;
71              -- se lee una linea del archivo de valores de prueba
72              readline(datos, l);
73              -- se extrae un entero de la linea
74              read(l, aux);
75              -- se carga el valor del operando A
76              a_file <= to_unsigned(aux, WORD_SIZE_T);
77              -- se lee un caracter (es el espacio)
78              read(l, ch);
79              -- se lee otro entero de la linea
80              read(l, aux);
81              -- se carga el valor del operando B
82              b_file <= to_unsigned(aux, WORD_SIZE_T);
83              -- se lee otro caracter (es el espacio)
84              read(l, ch);
85              -- se lee otro entero
86              read(l, aux);
87              -- se carga el valor de salida (resultado)
88              z_file <= to_unsigned(aux, WORD_SIZE_T);
89          end loop;
90          -- se cierra del archivo
91          file_close(datos);
92          wait for TCK*(DELAY+1);
93          -- se aborta la simulacion (fin del archivo)
94          assert false report
95          "Fin de la simulacion" severity failure;
96      end process Test_Sequence;
97  -- Instanciacion del DUV
98  DUV: fp_multiplication
99          generic map (
100                 NE => EXP_SIZE_T,
101                 NF => NF
102         )
103         port map(
104             clk => clk,
105             rst => rst,
106             x => std_logic_vector(a_file),
107             y => std_logic_vector(b_file),
```

```
            z => z_duv
        );
-- Instanciacion de la linea de retardo
del: delay_gen
    generic map(WORD_SIZE_T, DELAY)
    port map(clk, '0', std_logic_vector(z_file), z_del_aux);
    z_del <= unsigned(z_del_aux);
-- Verificacion de la condicion
verificacion: process(clk)
        begin
            if rising_edge(clk) then
            assert to_integer(z_del) = to_integer(unsigned(z_duv)) report
            "Error: Salida del DUV no coincide con referencia (salida del duv = " &
            integer'image(to_integer(unsigned(z_duv))) & ", salida del archivo = " &
            integer'image(to_integer(z_del)) & ")" &
            "Errores= " & integer'image(errores+1) & " en la linea " & integer'image(ciclos-2)
            severity warning;
            if to_integer(z_del) /= to_integer(unsigned(z_duv)) then
            errores <= errores + 1;
            end if;
            end if;
        end process;
end architecture pf_testbench_arq;
```

3.6.b. Suma/Resta

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;


entity pf_testbench_add is
end entity pf_testbench_add;


architecture pf_testbench_add_arq of pf_testbench_add is
    constant TCK: time:= 20 ns; -- periodo de reloj
    constant DELAY: natural:= 2; -- retardo de procesamiento del DUV
    constant NF : natural := 21;
    constant EXP_SIZE_T: natural:= 7; -- tamanio exponente
    constant WORD_SIZE_T: natural:= NF+EXP_SIZE_T+1; -- tamanio de datos
    signal clk: std_logic := '0';
    signal rst: std_logic := '1';
    signal a_file: unsigned(WORD_SIZE_T-1 downto 0):= (others => '0');
    signal b_file: unsigned(WORD_SIZE_T-1 downto 0):= (others => '0');
    signal z_file: unsigned(WORD_SIZE_T-1 downto 0):= (others => '0');
    signal z_del: unsigned(WORD_SIZE_T-1 downto 0):= (others => '0');
    signal z_duv: std_logic_vector(WORD_SIZE_T-1 downto 0):= (others => '0');
    signal ciclos: integer := 0; signal errores: integer := 0;
    -- La senal z_del_aux se define por un problema de conversion
    signal z_del_aux: std_logic_vector(WORD_SIZE_T-1 downto 0):= (others => '0');

    signal add_sub : std_logic := '1';

    file datos: text open read_mode is "../testbench/fsub_21_7.txt";
    -- Declaracion del componente a probar
    component fp_add is
        generic(
                NE: natural := 8;
                NF: natural := 23
            );
            port(
                clk : in std_logic;
                rst : in std_logic;
```

```
                add_sub : in std_logic;
                x : in std_logic_vector(NF+NE downto 0);
                y : in std_logic_vector(NF+NE downto 0);
                z : out std_logic_vector(NF+NE downto 0)
                );
    end component fp_add ;
    -- Declaracion de la linea de retardo
    component delay_gen is
        generic(
            N: natural:= 26;
            DELAY: natural:= 0
        );
        port(
            clk: in std_logic;
            rst: in std_logic;
            A: in std_logic_vector(N-1 downto 0);
            B: out std_logic_vector(N-1 downto 0)
        );
    end component;
begin
    -- Generacion del clock del sistema
    clk <= not(clk) after TCK/ 2; -- reloj
    rst <= '0' after 1 ns;

    Test_Sequence: process
        variable l: line;

        variable ch: character:= ' ';
        variable aux: integer;
    begin
        while not(endfile(datos)) loop
            wait until rising_edge(clk);
            -- solo para debugging
            ciclos <= ciclos + 1;
            -- se lee una linea del archivo de valores de prueba
            readline(datos, l);
            -- se extrae un entero de la linea
            read(l, aux);
            -- se carga el valor del operando A
            a_file <= to_unsigned(aux, WORD_SIZE_T);
            -- se lee un caracter (es el espacio)
            read(l, ch);
            -- se lee otro entero de la linea
            read(l, aux);
            -- se carga el valor del operando B
            b_file <= to_unsigned(aux, WORD_SIZE_T);
            -- se lee otro caracter (es el espacio)
            read(l, ch);
            -- se lee otro entero
            read(l, aux);
            -- se carga el valor de salida (resultado)
            z_file <= to_unsigned(aux, WORD_SIZE_T);
        end loop;
        -- se cierra del archivo
        file_close(datos);
        wait for TCK*(DELAY+1);
        -- se aborta la simulacion (fin del archivo)
        assert false report
        "Fin de la simulacion" severity failure;
    end process Test_Sequence;
-- Instanciacion del DUV
DUV: fp_add
        generic map (
                NE => EXP_SIZE_T,
```

```
104                 NF => NF
105         )
106         port map(
107             clk => clk,
108             rst => rst,
109             add_sub => add_sub,
110             x => std_logic_vector(a_file),
111             y => std_logic_vector(b_file),
112             z => z_duv
113         );
114 -- Instanciacion de la linea de retardo
115 del: delay_gen
116     generic map(WORD_SIZE_T, DELAY)
117     port map(clk, '0', std_logic_vector(z_file), z_del_aux);
118     z_del <= unsigned(z_del_aux);
119 -- Verificacion de la condicion
120 verificacion: process(clk)
121         begin
122             if rising_edge(clk) then
123             assert to_integer(z_del) = to_integer(unsigned(z_duv)) report
124             "Error: Salida del DUV no coincide con referencia (salida del duv = " &
125             integer'image(to_integer(unsigned(z_duv))) & ", salida del archivo = " &
126             integer'image(to_integer(z_del)) & ")" &
127             "Errores= " & integer'image(errores+1) & " en la linea " & integer'image(ciclos-2)
128             severity warning;
129             if to_integer(z_del) /= to_integer(unsigned(z_duv)) then
130             errores <= errores + 1;
131             end if;
132             end if;
133         end process;
134 end architecture pf_testbench_add_arq;
```

En este caso, existe un archivo para la suma y un archivo para la resta, por lo que además de cambiar NF y NE cada vez que se levanta un archivo, se debe cambiar el valor de `add_sub` (0 si es suma, 1 si es resta).

4. Simulación

A continuación se muestran los resultados obtenidos de realizar la simulación de comportamiento sobre los *testbenchs*. Los resultados se muestran para los archivos en los cuales el campo de $Nf = 21$ y el campo $Ne = 7$, pero se verificaron los mismos para el resto de los archivos.

4.1. Multiplicador

En la figura 4 se muestra la simulación de la unidad aritmética de multiplicación con el archivo `fmul_21_7.txt`. En particular se muestran las señales `z_duv`, que se corresponde con la salida del DUV (*device under verification*) y `z_file`, la salida del archivo retrasada los ciclos de clock necesarios para que coincidan ambas salidas.

Figura 4: Simulación de la multiplicación

4.1.a. Resolución de errores

Luego de verificar los valores de saturación tomados en el archivo de pruebas, se redujo la cantidad de errores a 6, como se muestra en la figura 5

```
../src/testbench_mult.vhd:119:13:@5030ns:(assertion warning): Error: Salida del DUV no coincide con referencia
   (salida del duv = 267166881, salida del archivo = 266338303)Errores= 1 en la linea 249
../src/testbench_mult.vhd:119:13:@9510ns:(assertion warning): Error: Salida del DUV no coincide con referencia
   (salida del duv = 530481, salida del archivo = 0)Errores= 2 en la linea 473
../src/testbench_mult.vhd:119:13:@9750ns:(assertion warning): Error: Salida del DUV no coincide con referencia
   (salida del duv = 267497596, salida del archivo = 266338303)Errores= 3 en la linea 485
../src/testbench_mult.vhd:119:13:@9970ns:(assertion warning): Error: Salida del DUV no coincide con referencia
   (salida del duv = 1758045, salida del archivo = 0)Errores= 4 en la linea 496
../src/testbench_mult.vhd:119:13:@18490ns:(assertion warning): Error: Salida del DUV no coincide con referencia
   (salida del duv = 268844440, salida del archivo = 268435456)Errores= 5 en la linea 922
../src/testbench_mult.vhd:119:13:@19630ns:(assertion warning): Error: Salida del DUV no coincide con referencia
   (salida del duv = 267574940, salida del archivo = 266338303)Errores= 6 en la linea 979
../src/testbench_mult.vhd:94:9:@20050ns:(assertion failure): Fin de la simulacion
```

Figura 5: Logfile de la salida del multiplicador

Como ejemplo para justificar estos errores se utilizó el primero, aunque la misma justificación aplica para el resto de los casos.

La línea 249 tiene como entradas: $X = 490862360$ e $Y = 445255520$, y la salida del archivo es $Z = 266338303$. En binario, se representan de la siguiente forma:

señal	representación binaria en 29 bits
X	$\overbrace{1}^{Sx}\ \overbrace{1101010}^{Ex}\ \overbrace{00001111011100011000}^{Fx}$
Y	$\overbrace{1}^{Sy}\ \overbrace{1010100}^{Ey}\ \overbrace{01010000111101100000}^{Fy}$
Z	$\overbrace{0}^{Sz}\ \overbrace{1111110}^{Ez}\ \overbrace{11111111111111111111}^{Fz}$

Tabla 1: Caption

Si se realiza el cálculo del exponente como se explicó previamente: se obtiene $Ez = 1111111$, el cual, según la convención tomada en este trabajo, es un exponente válido. Sin embargo, como en el archivo se toma como saturación $Ez = 1111110$, cualquier exponente mayor a ese llevará a salida al valor de saturación. Es por esto que en los errores obtenidos, según el archivo la salida corresponde a los valores de saturación, pero según el DUV es un valor de salida posible.

4.2. Sumador

En las figuras 6 y 7 se muestra la simulación de la unidad aritmética de la suma y resta con los archivos `fadd_21_7.txt` y `fsub_21_7.txt`, respectivamente. En particular se muestran las señales `z_duv`, que se corresponde con la salida del DUV

(*device under verification*) y `z_file`, la salida del archivo retrasada los ciclos de clock necesarios para que coincidan ambas salidas.

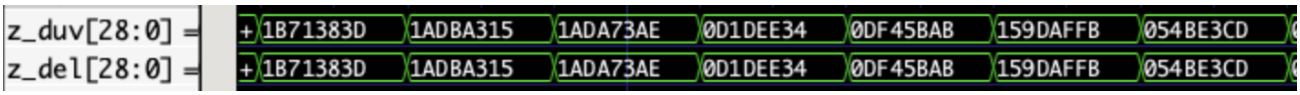

Figura 6: Simulación de la suma

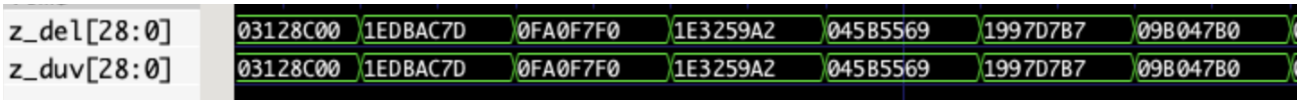

Figura 7: Simulación de la resta

A diferencia de la multiplicación, no se obtuvo ningún error en ninguno de los casos. Probablemente se deba a que los *testbenchs* no tienen ejemplos en donde se den valores de saturación.

5. Síntesis

La síntesis del diseño se hizo sobre la FPGA xc7a15tftg256-1, utilizando Vivado.

Solamente en el caso de la multiplicación tiene sentido mostrar el RTL (8), dado que se pueden visualizar los bloques descritos previamente y mostrados en la introducción.

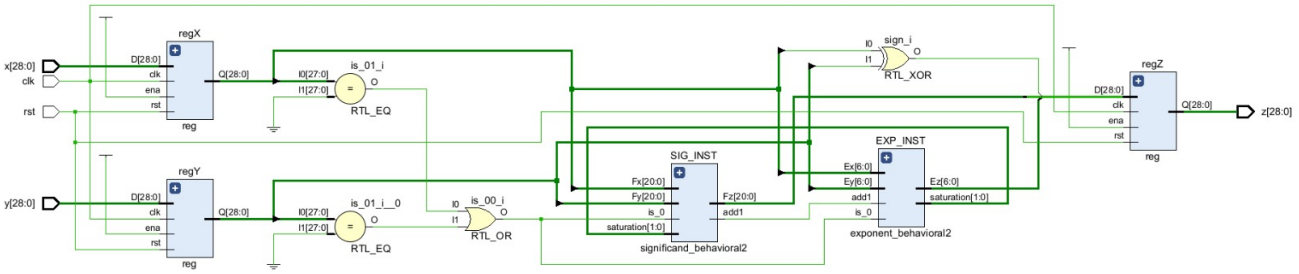

Figura 8: RTL de la multiplicación

Por el contrario, el RTL de la unidad de suma/resta no se muestra, dado que es considerablemente más extenso y no se puede identificar fácilmente el esquemático planteado.
A continuación se muestra el reporte de utilización de LUTs y FFs de ambas unidades aritméticas.

Name	Status	Failed Routes	LUT	FF	BRAM	URAM	DSP
synth_1	synth_design Complete!		49	87	0	0	2
impl_1	Not started						

Figura 9: Uso de recursos de la multiplicación

Name	Status	Failed Routes	LUT	FF	BRAM	URAM	DSP
synth_1	synth_design Complete!		1266	87	0	0	0
impl_1	Not started						

Figura 10: Uso de recursos de la suma

6. Conclusión

En este trabajo se cumplió el objetivo de describir, simular y sintetizar dos operaciones de punto flotante. Se logró un primer acercamiento a las unidades aritméticas de esta representación numérica, quedando pendientes ciertas optimizaciones de cálculo y sin ahondar en profundidad sobre los distintos números *denormales* y formas de redondeo (especificados en la norma IEEE 754).

Además, se pudo apreciar la diferencia en las prioridades y los abordamientos entre la descripción de hardware y el desarrollo de software, especialmente en el uso de funciones, que no describen únicamente un algoritmo, sino que en última instancia se transforman a LUTs. Esto permitió que se pueda observar la necesidad de mantener el código lo más simple posible para poder minimizar el uso de recursos de la FPGA.

Por último, en la unidad aritmética de suma/resta se pudo observar cómo se desdibuja el diseño original del esquemático del circuito, debido a la minimización que realiza la herramienta de síntesis.