Universidad de Buenos Aires
Facultad de Ingeniería

Sistemas Digitales (86.41)

Trabajo practico Nº2:

Aritmética de Punto flotante

Integrantes:

Puy Gonzalo 99784 gpuy@fi.uba.ar

1er cuatrimestre 2022

1. Introducción

El presente trabajo tiene como objetivo implementar funciones de una unidad aritmética de punto flotante, en especial, multiplicación, suma y resta. Dichas funciones seras descriptas en lenguaje VHDL, simulado y sintetizado sobre el dispositivo `xc7a15tftg256-1`. Para el caso de la simulación, se tendrán en cuenta archivos de prueba provistos por la cátedra.

2. Desarrollo

2.1. Especificaciones de diseño

Se diseñaron 2 unidades de punto flotante para realizar la multiplicación y suma/resta. Para esta implementación se tuvieron en cuenta las siguientes especificaciones

- En ambos casos se utilizó el método de redondeo a 0 (truncado).
- El tamaño de los campos significando (N_F) y exponente (N_E) son de tamaño genérico.
- No se consideraron números denormales ni tampoco casos especiales. Al no considerar estos últimos, si el resultado excede el rango de operación entonces se aplicara saturación, es decir, se devolverá a la salida el máximo (o mínimo) representable.
- Se simuló todas las unidades aritméticas de forma automatizada con los archivos de prueba provistos por la cátedra.

2.2. Diseño propuesto

En el caso particular de este trabajo se decidió implementar las unidades aritméticas de punto flotante de forma no sincrónica. Esto es, sin ningún tipo de dependencia de reloj.

De esta forma, la implementación no es secuencial, ya que la salida solo depende de los valores actuales de las entradas. Esto es totalmente sintetizable, ya que básicamente se esta describiendo una función booleana.

3. Implementación en VHDL

Se implemento el diseño propuesto en VHDL, a continuación se presentan los códigos realizados

3.1. Multiplicador

```
1  1library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  entity fp_mul is
6      generic(N : natural := 20;
7              NE : natural := 6
8      );
9      port (
10         X   : in std_logic_vector(N-1 downto 0); -- Operando 1
11         Y   : in std_logic_vector(N-1 downto 0); -- Operando 2
12         Z   : out std_logic_vector(N-1 downto 0) -- Resultado
13     );
14 end fp_mul;
15
16 architecture behavioral of fp_mul is
```

```
17
18      constant EXC         : natural := 2**(NE-1)-1;
19      constant NF          : natural := N-NE-1;
20      constant EXCESO      : unsigned(NE+1 downto 0) := to_unsigned(EXC, NE+2);
21      constant E_MIN       : unsigned(NE-1 downto 0) := to_unsigned(0, NE);
22      constant E_MAX       : unsigned(NE-1 downto 0) := to_unsigned(2**(NE)-2, NE);
23      constant E_CEROS     : unsigned(NE+1 downto 0) := to_unsigned(0, NE+2); -- usada para chequear campo
24      constant F_CEROS     : unsigned(NF-1 downto 0) := to_unsigned(0, NF); -- usada para chequear campo F
25      constant RES_CERO    : unsigned(N-2 downto 0)  := to_unsigned(0,N-1);
26
27      signal sx        : std_logic; -- Signo operando X
28      signal sy        : std_logic; -- Signo operando Y
29
30      signal ex        : unsigned(NE+1 downto 0) := (others => '0'); -- Exponente operando X, representado
31      signal ey        : unsigned(NE+1 downto 0) := (others => '0'); -- Exponente operando Y, representado
32      signal cero_op   : std_logic := '0';                           -- "Flag" usada para saber si uno de
33
34      signal fx        : unsigned(NF-1 downto 0) := (others => '0'); -- Mantisa operando X
35      signal fy        : unsigned(NF-1 downto 0) := (others => '0'); -- Mantisa operando Y
36
37      signal mx        : unsigned(NF downto 0) := (others => '0'); -- Significand operandos X
38      signal my        : unsigned(NF downto 0) := (others => '0'); -- significand operando Y
39
40      signal sz            : std_logic;                                      -- Signo del resultado
41      signal ez            : unsigned(NE-1 downto 0) := (others => '0');     -- Exponente del resultado, rep
42      signal ez_aux        : unsigned(NE+1 downto 0) := (others => '0');     -- Exponente auxiliar para cuen
43      signal ez_aux_p      : unsigned(NE+1 downto 0) := (others => '0');     -- Exponente auxiliar 'prima' p
44      signal mz            : unsigned(2*NF+1 downto 0) := (others => '0'); -- Significand del resultado.
45      signal fz            : unsigned(NF-1 downto 0) := (others => '0');     -- Mantisa del resultado.
46      signal fz_aux        : unsigned(NF-1 downto 0) := (others => '0');     -- Mantisa auxiliar para cuenta
47
48  begin
49
50      -- Separacion de los campos de los operandos
51      sx <= X(NE+NF);
52      sy <= Y(NE+NF);
53      ex <= '0' & '0' & unsigned(X(NF+NE-1 downto NF)); -- Le agrego 2 '0' al principio para poder cheque
54      ey <= '0' & '0' & unsigned(Y(NF+NE-1 downto NF)); -- Por eso ex, ey, ez y EXCESO tienen NE+2 bits.
55      fx <= unsigned(X(NF-1 downto 0));
56      fy <= unsigned(Y(NF-1 downto 0));
57
58      -- Chequeo si alguno de los operandos es cero (Resultado trivial)
59      cero_op <= '1' when ( (ex = E_CEROS) and (fx = F_CEROS) ) else
60                 '1' when ( (ey = E_CEROS) and (fy = F_CEROS) ) else
61                 '0';
62
63      -- Calculo del signo del resultado
64      sz <= sx xor sy;
65
```

```
66      -- Multiplicacion de los Significand
67      mx <= '1' & fx; -- Le agrego un '1' al principio a F para calcular el Significand
68      my <= '1' & fy; -- por eso mx, my tienen NF+1 bits.
69
70      mz <= mx * my; -- mz tiene 2*NF+2 bits.
71
72      -- Calculo del exponente representado en exceso
73      ez_aux <= ex + ey - EXCESO;
74
75      -- Redondeo de la mantisa
76      fz_aux <=
77            mz(2*NF downto NF+1) when mz(2*NF+1) = '1' else
78            mz(2*NF-1 downto NF);
79
80      ez_aux_p <=
81               (ez_aux + 1) when mz(2*NF+1) = '1' else ez_aux;
82
83      -- Logica de saturacion:
84      ez <= E_MAX when ( (ez_aux_p(NE+ 1) = '0') and (ez_aux_p(NE) = '1') ) else
85            E_MIN when ( (ez_aux_p(NE+ 1) = '1') and (ez_aux_p(NE) = '1') )  else
86            ez_aux_p(NE-1 downto 0);
87
88      fz <= (others => '1') when ( (ez_aux_p(NE+ 1) = '0') and (ez_aux_p(NE) = '1') ) else
89            (others => '0') when ( ( (ez_aux_p(NE+ 1) = '1') and (ez_aux_p(NE) = '1') ) ) else
90            fz_aux;
91
92      Z <= std_logic_vector(sz & RES_CERO) when ( cero_op = '1' ) else
93           std_logic_vector(sz & ez & fz);
94
95  end architecture behavioral;
```

3.2. Sumador/Restador

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   use IEEE.numeric_std.all;
4
5   entity fp_subadd is
6       generic(N : natural := 20;
7               NE : natural := 6
8       );
9       port (
10          X          : in std_logic_vector(N-1 downto 0); -- Operando 1
11          Y          : in std_logic_vector(N-1 downto 0); -- Operando 2
12          ctrl       : in std_logic; -- Indicador de suma o resta. Si es 0 se suma, si es 1 se resta.
13          Z          : out std_logic_vector(N-1 downto 0) -- Resultado
14      );
```

```
15  end fp_subadd;
16
17  architecture behavioral of fp_subadd is
18
19      constant EXC    : natural := 2**(NE-1)-1;
20      constant NF     : natural := N-NE-1;
21      constant EXCESO : signed(NE-1 downto 0) := to_signed(EXC, NE);
22      constant E_MAX  : signed(NE downto 0) := to_signed(2**(NE)-2, NE+1);
23      constant E_MIN  : signed(NE downto 0) := to_signed(0, NE+1);
24
25      signal Y_aux    : std_logic_vector(N-1 downto 0) := (others => '0');
26
27      signal ex_aux   : unsigned(NE downto 0) := (others => '0'); -- Tomo el campo E de la entrada X con
28      signal ey_aux   : unsigned(NE downto 0) := (others => '0'); -- Tomo el campo E de la entrada Y con
29      signal resta_E  : unsigned(NE downto 0) := (others => '0'); -- Resta de los campos E de X e Y.
30
31      signal X_p      : unsigned(N-1 downto 0) := (others => '0'); -- X_prima con el que voy a hacer efec
32      signal Y_p      : unsigned(N-1 downto 0) := (others => '0'); -- Y_prima con el que voy a hacer efec
33
34      signal sx_p     : std_logic; -- Signo operando X_prima
35      signal sy_p     : std_logic; -- Signo operando Y_prima
36      signal ex_p     : unsigned(NE-1 downto 0) := (others => '0');  -- Campo E de X_prima
37      signal ey_p     : unsigned(NE-1 downto 0) := (others => '0');  -- Campo E de Y_prima
38      signal fx_p     : unsigned(NF-1 downto 0) := (others => '0'); -- Mantisa operando X_prima
39      signal fy_p     : unsigned(NF-1 downto 0) := (others => '0'); -- Mantisa operando Y_prima
40      signal mx_p     : unsigned(NF downto 0) := (others => '0'); -- Significand operando X_prima
41      signal my_p     : unsigned(NF downto 0) := (others => '0'); -- significand operando Y_prima
42
43      -- Señales para cuentas auxiliares con los significands
44      --  Como tengo que agregar |Ex-Ey| '0's a la derecha de mx_p y |Ex - Ey| '0's a la izquierda de my_
45      -- creo un unsigned mx_2prima con el peor caso posible de |Ex-Ey| (Ex todo '1' e Ey todos '0'),
46      -- con esto me aseguro de tener siempre bien la cantidad de ceros.
47      signal mx_2p    : unsigned(NF+2**(NE)-1 downto 0) := (others => '0');
48      signal my_2p    : unsigned(NF+2**(NE)-1 downto 0) := (others => '0');
49      signal mx_3p    : unsigned(NF+2**(NE)+1 downto 0) := (others => '0');
50      signal my_3p    : unsigned(NF+2**(NE)+1 downto 0) := (others => '0');
51      signal mx_4p    : unsigned(NF+2**(NE)+1 downto 0) := (others => '0');
52      signal my_4p    : unsigned(NF+2**(NE)+1 downto 0) := (others => '0');
53
54      signal suma     : unsigned(NF+2**(NE)+1 downto 0) := (others => '0');
55      signal suma_p   : unsigned(NF+2**(NE)+1 downto 0) := (others => '0');
56
57      signal fz_aux   : unsigned(NF+2**(NE)+1 downto 0) := (others => '0');
58      signal fz_aux_p : unsigned(NF-1 downto 0) := (others => '0');
59      signal fz       : unsigned(NF-1 downto 0) := (others => '0');
60      signal ez_aux   : signed(NE downto 0) := (others => '0');
61      signal ez       : signed(NE-1 downto 0) := (others => '0');
62      signal sz       : std_logic := '0'; -- Signo Resultado Z
63
```

```
64      signal comp_x   : std_logic := '0'; -- Bit que indica si debo complementar X_p: 0 -> no compemento;
65      signal comp_y   : std_logic := '0'; -- Bit que indica si debo complentar Y_p: 0 -> no compemento; 1
66
67      -- Funcion para encontrar el primer uno en un std_logic_vector
68      function find_one (x0: std_logic_vector) return integer is
69
70          variable found  : boolean;
71          variable index  : integer;
72
73      begin
74          found := False;
75
76          for i in x0'length-1 downto 0 loop
77              if x0(i) = '1' and found = False then
78                  found := True;
79                  index := i;
80              end if;
81          end loop;
82
83          if index < 0 then
84              index := 0;
85          end if;
86
87          return index;
88      end function;
89
90  begin
91
92      Y_aux <= not(Y(N-1)) & Y(N-2 downto 0) when ctrl = '1' else
93               Y;
94
95
96      --  Verifico cual de los operandos tiene mayor exponente.
97      -- para esto, agrego un 0 a la izquierda y chequeo el signo (MSB) de la resta entre los campos E.
98      ex_aux <= '0' & unsigned(X(NF+NE-1 downto NF));
99      ey_aux <= '0' & unsigned(Y_aux(NF+NE-1 downto NF));
100
101     resta_E <= ex_aux - ey_aux;
102
103     -- El campo Ex debe ser mayor (en valor absoluto) que el campo Ey.
104     -- Si el resultado de la resta es negativo => Ex < Ey. Por lo tanto invierto los operandos,
105     -- de lo contrario, los dejo como estaban.
106     X_p <= unsigned(Y_aux) when ( resta_E(NE) = '1' ) else
107            unsigned(X);
108
109     Y_P <= unsigned(X) when ( resta_E(NE) = '1' ) else
110            unsigned(Y_aux);
111
112     -- Tomo lo necesario de los nuevos valores de X_prima e Y_prima
```

```
113     sx_p <= X_p(NE+NF);
114     sy_p <= Y_p(NE+NF);
115     ex_p <= X_p(NF+NE-1 downto NF); -- Los ceros se los agrego para luego ver saturacion.
116     ey_p <= Y_p(NF+NE-1 downto NF); -- Los ceros se los agrego xq no puedo operar con vectores de disti
117     fx_p <= X_p(NF-1 downto 0);
118     fy_p <= Y_p(NF-1 downto 0);
119     mx_p <= '1' & fx_p; -- Le agrego un '1' al principio a F para calcular m
120     my_p <= '1' & fy_p; -- por eso mx, my tienen NF+1 bits.
121
122     -- Logica para complementar los operandos
123     comp_x <= '0' when sx_p = '0' else
124               '1';
125
126     comp_y <= '0' when ( (sy_p = '0') ) else
127               '1';
128
129     -- Alineacion de los significands
130     mx_2p(NF downto 0) <= mx_p;
131     my_2p(NF downto 0) <= my_p;
132
133     -- Shifteo a izquierda |Ex| posiciones el significand del operando X_prima
134     -- Shifteo a izquierda |Ey| posiciones el significand del operando Y_prima
135     -- De esta forma me quedan bien alineados.
136     mx_3p <= '0' & '0' & (mx_2p sll to_integer(ex_p));
137     my_3p <= '0' & '0' & (my_2p sll to_integer(ey_p));
138
139
140     -- En caso de que tenga que complementar, lo hago
141     mx_4p <= (not(mx_3p) + 1) when (comp_x = '1') else
142              mx_3p;
143     my_4p <= (not(my_3p) + 1) when (comp_y = '1') else
144              my_3p;
145
146     -- Suma/Resta efectiva
147     suma <= mx_4p + my_4p;
148
149     -- Signo del resultado Z.
150     sz <= suma(NF+2**(NE)+1);
151
152     --  Si el resultado de la suma/resta da negativo, tengo que complementar
153     suma_p <= suma when suma(NF+2**(NE)+1) = '0' else
154               (not(suma)+1);
155
156     -- Normalizacion de la mantisa de Z
157     fz_aux   <= suma_p sll (suma_p'length - find_one(x0 => std_logic_vector(suma_p)));
158     fz_aux_p <= fz_aux(fz_aux'length-1 downto fz_aux'length-NF );
159
160
161     -- Campo Ez
```

```
162     ez_aux <= to_signed(find_one(x0 => std_logic_vector(suma_p)) - NF, NE+1);
163 
164     --  Logica de saturacion para el resultado
165     ez <= E_MAX(NE-1 downto 0) when ( to_integer(ez_aux) > to_integer(E_MAX) ) else
166           E_MIN(NE-1 downto 0) when ( to_integer(ez_aux) < to_integer(E_MIN) ) else
167           ez_aux(NE-1 downto 0);
168 
169     fz <= (others => '1') when ( to_integer(ez_aux) > to_integer(E_MAX) ) else
170           (others => '0') when ( to_integer(ez_aux) < to_integer(E_MIN) ) else
171           fz_aux_p;
172 
173     -- Resultado final
174     Z <= sz & std_logic_vector(ez) & std_logic_vector(fz);
175 
176 end architecture behavioral;
```

4. Simulación

Para simular los resultados de los códigos se implementaron los siguientes bancos de prueba para luego hacer una simulación mediante `GHDL` y `GTKWave`. Dado que los archivos de prueba son muchos, solo se mostraran una corrida para un solo archivo de prueba.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use std.textio.all;
5 
6  entity tb_fpmul is
7  end entity tb_fpmul;
8 
9  architecture tb_arch of tb_fpmul is
10 
11     constant FILE_PATH  : string := "..\..\Archivos_de_Testeo\fmul_21_7.txt";
12     constant TCK        : time := 20 ns; -- Periodo de reloj
13     constant F_SIZE     : natural := 21; -- Tamaño de mantisa
14     constant EXP_SIZE   : natural := 7;  -- Tamaño del exponente
15     constant WORD_SIZE  : natural := EXP_SIZE + F_SIZE + 1; -- Tamaño de datos
16 
17 
18     signal clk      : std_logic := '0';
19     signal x_file   : std_logic_vector(WORD_SIZE-1 downto 0) := (others => '0');
20     signal y_file   : std_logic_vector(WORD_SIZE-1 downto 0) := (others => '0');
21     signal z_file   : std_logic_vector(WORD_SIZE-1 downto 0) := (others => '0');
22     signal z_dut    : std_logic_vector(WORD_SIZE-1 downto 0) := (others => '0');
23 
24     signal ciclos   : integer := 0;
25     signal errores  : integer := 0;
```

```
26
27      file datos : text open read_mode is FILE_PATH;
28
29  begin
30
31      clk <= not(clk) after TCK/2; -- Reloj
32
33      Test_Sequence: process
34          variable l   : line;
35          variable ch  : character := ' ';
36          variable aux : integer;
37      begin
38          while not(endfile(datos)) loop
39              wait until rising_edge(clk);
40              -- Solo para debugging
41              ciclos <= ciclos + 1;
42              -- Se lee una linea del archivo de valores de prueba
43              readline(datos, l);
44              -- Se extrae un entero de la linea
45              read(l, aux);
46              -- Se carga el valor del operando X
47              x_file <= std_logic_vector(to_unsigned(aux, WORD_SIZE));
48              -- Se lee un caracter (el espacio)
49              read(l, ch);
50              -- Se lee otro entero de la linea
51              read(l, aux);
52              -- Se carga el valor del operando Y
53              y_file <= std_logic_vector(to_unsigned(aux, WORD_SIZE));
54              -- Se lee otro caracter (el espacio)
55              read(l, ch);
56              -- Se lee otro entero
57              read(l, aux);
58              -- Se carga el valor de la salida (resultado)
59              z_file <= std_logic_vector(to_unsigned(aux, WORD_SIZE));
60          end loop;
61
62          file_close(datos); -- Se cierra el archivo
63
64
65          -- Se aborta la simulacion (fin del archivo)
66          assert false report
67              "Fin de la simulacion" severity failure;
68
69      end process Test_Sequence;
70
71      -- Instanciacion del DUT
72      DUT: entity work.fp_mul(behavioral)
73      generic map(
74          N => WORD_SIZE,
```

```
        NE => EXP_SIZE
    )
    port map(
        X => x_file,
        Y => y_file,
        Z => z_dut
    );


    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;

            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;
```

```
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

    constant FILE_PATH  : string := "..\..\Archivos_de_Testeo\fsub_12_6.txt";
    constant TCK        : time := 20 ns; -- Periodo de reloj
    constant F_SIZE     : natural := 12; -- Tamaño de mantisa
    constant EXP_SIZE   : natural := 6;  -- Tamaño del exponente
    constant WORD_SIZE  : natural := EXP_SIZE + F_SIZE + 1; -- Tamaño de datos


    signal clk          : std_logic := '0';
    signal ctrl_tb      : std_logic := '1';
    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');
```

```
22      signal z_file       : std_logic_vector(WORD_SIZE-1 downto 0) := (others => '0');
23      signal z_dut        : std_logic_vector(WORD_SIZE-1 downto 0) := (others => '0');
24
25      signal ciclos   : integer := 0;
26      signal errores  : integer := 0;
27
28      file datos : text open read_mode is FILE_PATH;
29
30  begin
31
32      clk <= not(clk) after TCK/2; -- Reloj
33
34      Test_Sequence: process
35          variable l   : line;
36          variable ch  : character := ' ';
37          variable aux : integer;
38      begin
39          while not(endfile(datos)) loop
40              wait until rising_edge(clk);
41              -- Solo para debugging
42              ciclos <= ciclos + 1;
43              -- Se lee una linea del archivo de valores de prueba
44              readline(datos, l);
45              -- Se extrae un entero de la linea
46              read(l, aux);
47              -- Se carga el valor del operando X
48              x_file <= std_logic_vector(to_unsigned(aux, WORD_SIZE));
49              -- Se lee un caracter (el espacio)
50              read(l, ch);
51              -- Se lee otro entero de la linea
52              read(l, aux);
53              -- Se carga el valor del operando Y
54              y_file <= std_logic_vector(to_unsigned(aux, WORD_SIZE));
55              -- Se lee otro caracter (el espacio)
56              read(l, ch);
57              -- Se lee otro entero
58              read(l, aux);
59              -- Se carga el valor de la salida (resultado)
60              z_file <= std_logic_vector(to_unsigned(aux, WORD_SIZE));
61          end loop;
62
63          file_close(datos); -- Se cierra el archivo
64
65
66          -- Se aborta la simulacion (fin del archivo)
67          assert false report
68              "Fin de la simulacion" severity failure;
69
70      end process Test_Sequence;
```

```
71
72      -- Instanciacion del DUT
73      DUT: entity work.fp_subadd(behavioral)
74      generic map(
75          N => WORD_SIZE,
76          NE => EXP_SIZE
77      )
78      port map(
79          X       => x_file,
80          Y       => y_file,
81          ctrl    => ctrl_tb,
82          Z       => z_dut
83      );
84
85
86      verificacion: process(clk)
87      begin
88          if rising_edge(clk) then
89              assert to_integer(unsigned(z_file)) = to_integer(unsigned(z_dut)) report
90                  "Error: Salida del DUT no coincide con referencia (salida del DUT = " &
91                  integer'image(to_integer(unsigned(z_dut))) &
92                  ", salida del archivo = " &
93                  integer'image(to_integer(unsigned(z_file))) & ")"
94                  severity warning;
95
96              if to_integer(unsigned(z_file)) /= to_integer(unsigned(z_dut)) then
97                  errores <= errores + 1;
98              end if;
99          end if;
100     end process;
101
102 end architecture tb_arch;
```

4.1. Multiplicador

4.1.1. Resultados de simulación y problemas encontrados

Para el caso del multiplicador, el resultado final arroja varios errores en varios de los archivos de prueba. Sin embargo, estos errores son debidos a los archivos de prueba y al diseño propuesto por la cátedra. En dicho diseño se plantea que no se utilizan números denormales, lo cual implica que el campo E que representa el exponente del numero medido en exceso, puede ir desde 0 hasta $2^{N_E} - 1$, donde N_E es la cantidad de bits de dicho campo. Esto difiere de la norma `IEEE 754` (Chequear), en donde el campo E con valor $2^{N_E} - 1$ esta reservado para un caso especial (Buscar cual es). Lo que hace que el valor máximo que este puede tomar sea el de $2^{N_E} - 2$.

Con lo dicho anteriormente y con los resultados de la simulación, se puede ver que el método usado para crear los archivos de prueba si tiene en cuenta este caso, por lo que los errores encontrados se dan al truncar el máximo representable al valor de E : $2^{N_E} - 1$, cuando el archivo tiene como resultado el valor $2^{N_E} - 2$.

Solo se pudo encontrar un error, en donde lo que difiere son las mantisas. Por lo tanto, se asume en este caso que es un error dado en el archivo de prueba.

A continuación se muestra una imagen de la simulación de la multiplicación, para el caso del archivo '`fmul_21_7.txt`' (Figura 1) y también para este archivo un ejemplo de los errores descritos anteriormente (Figura 2).

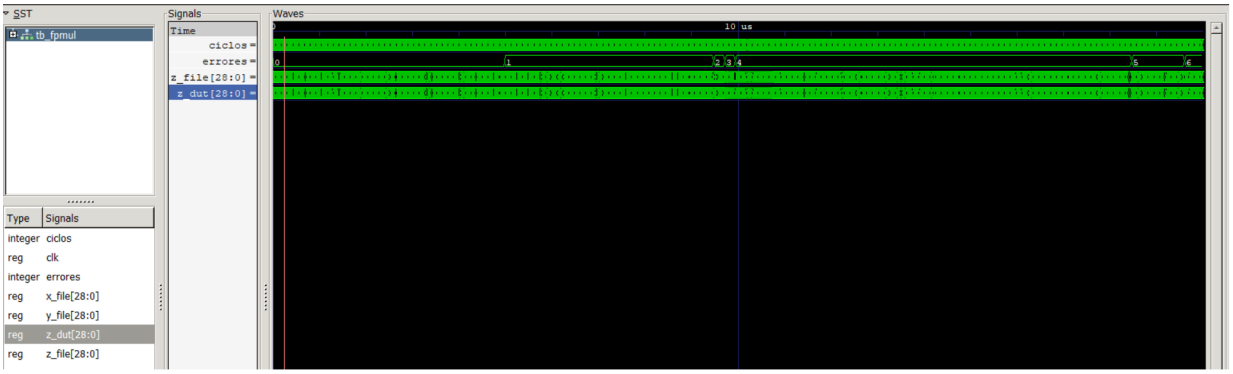

Figura 1

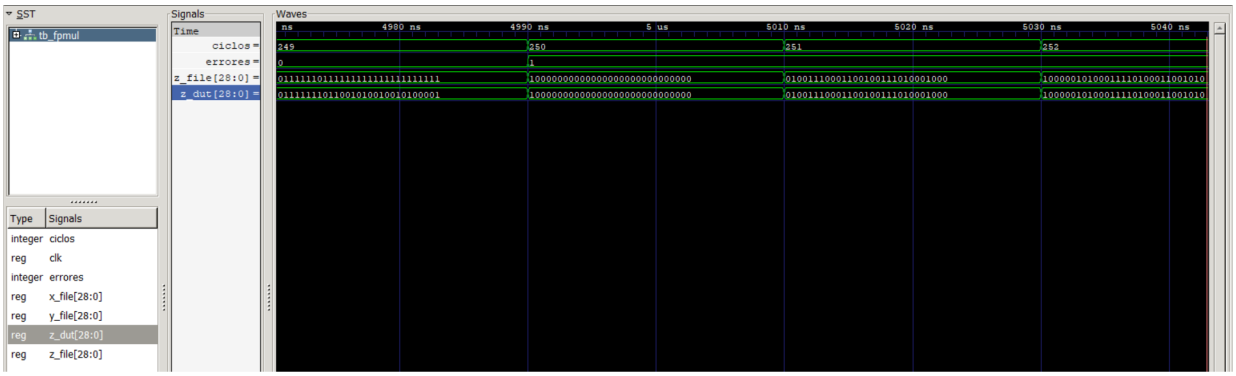

Figura 2

4.2. Sumador/Restador

4.2.1. Resultados de simulación

Para el caso del sumado/restador, no se apreciaron errores notorios, salvo por alguno en particular donde lo que difiere son las mantisas. Como en el caso del multiplicador, se adjudica el error al archivo de pruebas. Se adjunta una muestra de la simulacion para los archivos '`fsub_12_6.txt`' y '`fadd_12_6.txt`' en las figuras 3 y 4 respectivamente.

Cabe mencionar que en las simulaciones de suma hay un error que se da en tiempo 0, antes del primer flanco de clock, cuando todas las señales valen '0'. Para el caso del archivo seleccionado como ejemplo, se dan dos errores como los mencionados arriba. Esto se muestra en la figura 5.

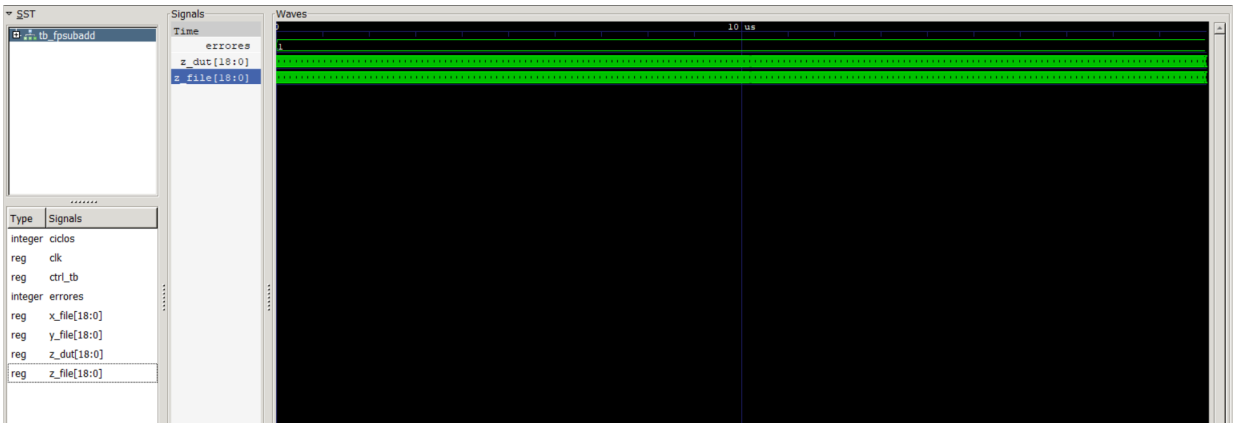

Figura 3

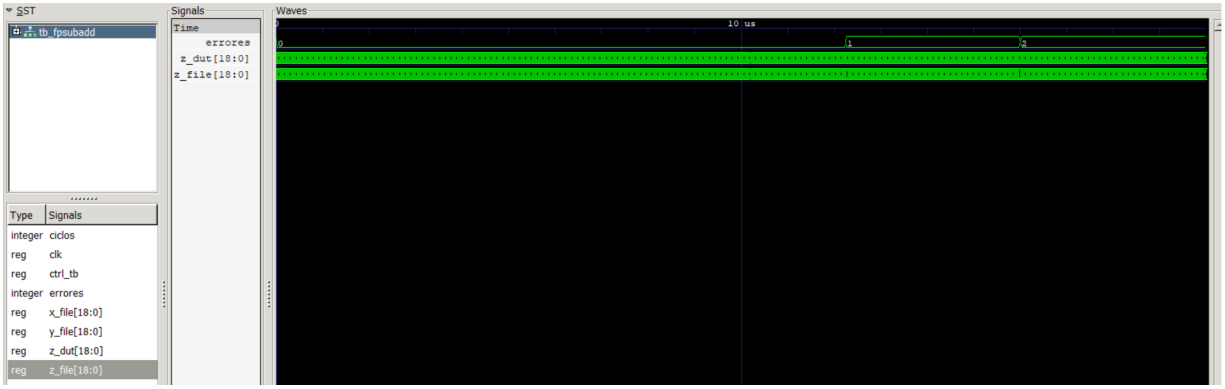

Figura 4

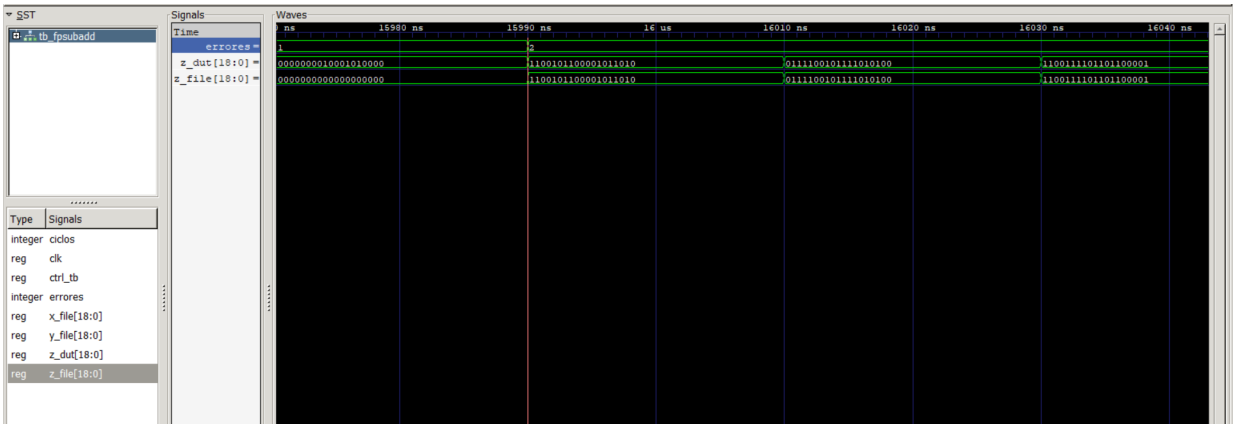

Figura 5

5. Síntesis

Para esta sección, se realizo una síntesis sobre el dispositivo FPGA `xc7a15tftg256-1` mediante el *software* Vivado. En esta herramienta, se muestra como resulta el circuito descripto mediante VHDL implementado en el dispositivo.

Para el caso de los esquemáticos, las imágenes que quedaron muy grandes para poder adjuntarlas en este documento, no se mostraran. Adjuntarlas en este informe no tendría sentido.

5.1. Multiplicación

El esquemático RTL se muestra en la figura 6. El de implementación resultó muy grande para el informe.

Otro de los resultados de interés, es el resumen de recursos utilizados en el diseño. El cual muestra la cantidad utilizada de Flip-Flops, LUT y puertos IO que se serian necesarios en la implementación en el dispositivo. Este resumen y el resumen de tiempos se muestra en las figuras 7 y 8 respectivamente.

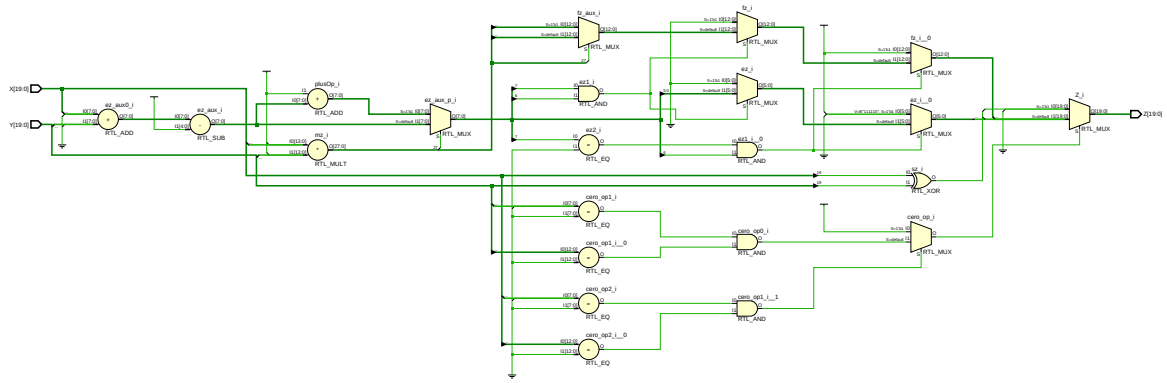

Figura 6

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

Figura 7: Resumen de los recursos utilizados en el dispositivo `xc7a15tftg256-1`

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, los esquemáticos RTL y los de implementación quedaron muy grandes para ser mostrados en el informe.

Y finalmente, se muestra el resumen de los recursos utilizados y el resumen 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: Resumen de los recursos utilizados en el dispositivo `xc7a15tftg256-1`

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

6. Conclusiones

Para concluir, se puede acotar la importancia de entender que este lenguaje es de descripción de *hardware*. Esto ya se había contemplado en el primer trabajo de la materia, pero en este se pudo apreciar de mejor manera, ya que se diseño un circuito mas complejo que en el trabajo anterior.

Al ver esto, uno puede entender que lo que realmente esta haciendo a la hora de escribir en VHDL es describir conexiones **reales**, y que es un cambio de 180 grados con respecto a pensar las cosas como se pensarían en otro lenguaje de programación como lo puede ser C, Python, etc.

Siguiendo esta idea, también es interesante notar como actúa el simulador a la hora de trabajar con este tipo de diseño de lógica combinacional. Este realizará toda la lógica combinacional (asignaciones, etc.) en tiempos δ_1, δ_2, ... , δ_n los cuales están todos contenidos en un tiempo Δ mas grande, que esta dado por el reloj del sistema.

Finalmente, también se puede aclarar que se pudo comprender el funcionamiento de una unidad de punto flotante de forma combinacional sin uso de sincronismo alguno. Y en cuanto a la síntesis, lo demandantes que son en cuanto a recursos de la fpga estas unidades implementadas en lógica combinacional sobre todo el sumador/restador.