TP2 Aritmética de Punto Flotante

Sistemas Digitales FIUBA

2021

1 Objetivo

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

2 Especificaciones

- Implementar en lenguaje descriptor de hardware VHDL dos unidades de punto flotante para realizar la suma/resta y multiplicación.
- En ambos casos, el método de redondeo será a 0 (truncado). El tamaño de los campos significando y exponente serán genéricos (N_F y N_e respectivamente).
- No se considerarán números denormales como así tampoco casos especiales (NaN, $\pm\infty$).
- Al no considerar denormales, el caso de todos los campos '0' mientras que el bit de signo sea '0' o '1' se considerará como el número cero.
- Al no considerar NaN y $\pm\infty$, si el resultado excede el rando de operación entonces deberá aplicarse saturación, es decir se devolverá a la salida el máximo (o mínimo) representable.
- Simular de forma automatizada todas las unidades aritméticas con los vectores de prueba suministrados en el campus virtual de la materia (utilizar todos los tamaño de datos disponibles).

3 Consideraciones generales

3.1 Modalidad de trabajo

Se recomienda enfáticamente realizar un testbench (simulación) por cada descripción de hardware a implementar. Una vez que se haya comprobado fehacientemente que cada descripción funciona por separado, se procederá a integrarlas y realizar una simulación general, la cual abarcará todo el sistema.

3.2 Lógica en el camino del reloj

No disponer lógica en el camino del reloj. No utilizar más de un dominio de reloj. Utilizar sólo uno de los dos flancos de reloj (*falling_edge* o *rising_edge*) en todos los circuitos sincrónicos que se implementen.

3.3 Lógica en el camino del reset

No disponer lógica en el camino del reset. No utilizar más de un reset asincrónico global. Utilizar sólo uno de los dos niveles de reset (’1’ o ’0’) en todos los circuitos sincrónicos que se implementen.

3.4 Estilo de codificación

No utilizar señales de tipo `bit` o `bit_vector`. Tampoco utilizar señales de tipo inout o buffer para puertos de entidades. Indentar correctamente cada bloque. Utilizar mayúsulas sólo para las constantes.

3.5 Implementación en FPGA

El alumno deverá realizar una síntesis sobre el dispositivo xc7a15tftg256-1 demostrando la correcta implementación del diseño en el mismo.

3.6 Forma de evaluación

El desarrollo y evaluación del trabajo práctico es individual.

El informe del trabajo práctico se deberá entregar en formato PDF junto con el código fuente VHDL en la fecha límite dispuesta por la cátedra (ver en el campus de la materia).

El docente correrá una simulación y verificará la correcta operación del sistema. funciona correctamente. Se evaluará individualmente al alumno con el trabajo práctico funcionando y, en base a las preguntas realizadas por el docente, se fijará una nota.

4 Manejo de archivos en VHDL para testbench

4.1 Introducción

Se explicará a continuación el manejo de archivos en VHDL orientado al diseño de un testbench o banco de pruebas para simular y verificar el funcionamiento de arquitecturas de hardware. Las funciones que se detallarán forman parte de la biblioteca std.textio, por lo cual, ésta debe ser incluida en el encabezado del banco de pruebas.

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

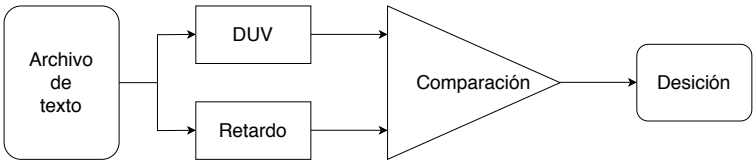
4.2 Diseño de un banco de pruebas

El objetivo al que apunta el diseño de un banco de pruebas es a la verificación exhaustiva del correcto funcionamiento de una descripción de hardware. Para ello se utilizará un archivo de texto, el cual contiene las muestras sin procesar y el resultado esperado de las muestras procesadas. Alimentando al dispositivo bajo verificación (DUV) con estas muestras se puede verificar a la salida si el resultado que arroja el procesamiento de las mismas es consistente con el resultado presente en el citado archivo de texto el cual a sido obtenido con un modelo de referencia de alto nivel (*golden model*.

En la figura siguiente se reconocen los siguientes elementos:

- **Archivo de texto:** Contiene las muestras procesadas (resultado) y sin procesar (datos de entrada). Se utilizarán las muestras procesadas como referencia para verificar el correcto funcionamiento del DUV.
- **DUV:** *Device Under Verification*. Es la entidad que se desea verificar.
- **Retardo:** En ocasiones es necesario introducir un retardo a fin de compensar la demora en el procesamiento de los datos, para que el resultado que procesó el DUV se encuentre sincronizado con el resultado tomado del archivo de texto. Esta línea de retardo se puede implementar fácilmente con un registro desplazamiento. Otra forma práctica es que el DUV posea una entrada comúnmente denominada *start* y una salida de tipo *strobe* denominada *done*. De esta forma el testbench enviará un pulso de duración de un período de reloj (*strobe*) cuando dispone de un dato nuevo para procesar y esperará hasta que el DUV devuelva un *strobe* sobre la señal *done* indicando que el resultado está listo.

- **Comparación:** Se comparará aquí el resultado del DUV y el resultado esperado, obtenido del archivo de texto. Para implementar la comparación y la decisión se usará un assertion statement.
- **Decisión:** De acuerdo al índice de severidad se decide qué hacer con la simulación. Se puede abortar o señalizar una advertencia o error.



4.3 Lectura de archivos

Se utilizarán las funciones readline y read de la biblioteca textio.La función *readline* levanta una línea de texto completa de un archivo. Dicha línea se alojará en una variable tipo *line*. Teniendo la línea de texto cargada, se procederá a leerla y recuperar los datos con la función *read*. Este procedimiento de levantar una línea de texto y leer de ella se repetirá indefinidamente hasta el final del archivo de texto y es por ello que se debe realizar un bucle.

4.4 Ejemplo

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

entity pf_testbench is
end entity pf_testbench;

architecture pf_testbench_arq of pf_testbench is

    constant TCK: time:= 20 ns;          -- periodo de reloj
    constant DELAY: natural:= 0;         -- retardo de procesamiento del DUV
    constant WORD_SIZE_T: natural:= 21; -- tamaño de datos
    constant EXP_SIZE_T: natural:= 7;    -- tamaño exponente

    signal clk: std_logic:= '0';
    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: unsigned(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 conversión
    signal z_del_aux: std_logic_vector(WORD_SIZE_T-1 downto 0):= (others => '0');

    file datos: text open read_mode is "test_mul_float_23_7.txt";

    -- Declaracion del componente a probar
    component xxx_pfis
        generic(
            WORD_SIZE: natural := 23
            EXP_SIZE: natural := 7
        );
        port(
            op_a: in std_logic_vector(WORD_SIZE-1 downto 0);   -- Operando A
            op_b: in std_logic_vector(WORD_SIZE-1 downto 0);   -- Operando B
            res: out std_logic_vector(WORD_SIZE-1 downto 0)    -- Resultado
        );
    end component xxx_pf;

    -- Declaracion de la linea de retardo
    component delay_gen is
        generic(
            N: natural:= 26;
            DELAY: natural:= 0
        );
        port(
            clk: 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

    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: xxx_pf
        generic map(
            WORD_SIZE => WORD_SIZE_T,
            EXP_SIZE => EXP_SIZE_T
        )
        port map(
            op_a => std_logic_vector(a_file),
```

```
                op_b => std_logic_vector(b_file),
                res  => z_duv
            );

    -- Instanciacion de la linea de retardo
    del: delay_gen
            generic map(WORD_SIZE_T, DELAY)
            port map(clk, 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(z_duv) report
                "Error: Salida del DUV no coincide con referencia (salida del duv = " &
                integer'image(to_integer(z_duv)) &
                ", salida del archivo = " &
                integer'image(to_integer(z_del)) & ")"
                severity warning;
            if to_integer(z_del) /= to_integer(z_duv) then
                errores <= errores + 1;
            end if;
        end if;
    end process;

end architecture pf_testbench_arq;
```