Documento trabajo práctico final : IP Sumador punto flotante

1. Implementación

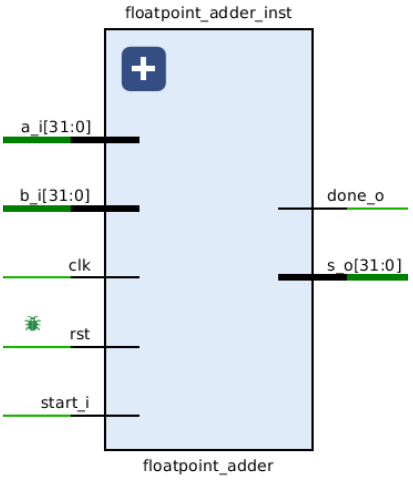
Para el trabajo final, se implementó un bloque sumador punto flotante. Este sumador realiza la suma en precisión simple (32 bits) o precisión doble (64 bits) de dos números. La siguiente imagen muestra un ejemplo de la instancia del componente para una suma de 32 bits.

```
component floatpoint_adder is
  generic (
  PRECISION_BITS: natural := 32
  );

  port(
    a_i: in std_logic_vector(PRECISION_BITS-1 downto 0);
    b_i: in std_logic_vector(PRECISION_BITS-1 downto 0);
    start_i: in std_logic;
    s_o: out std_logic_vector(PRECISION_BITS-1 downto 0);
    done_o: out std_logic;
    rst: in std_logic;
    clk: in std_logic
  );
end component;
```

El bloque tiene las siguientes entradas:

- a y b: números a sumar.
- clk.
- rst.
- start_i : start_i en 1 indica que se quiere realizar la suma de a con b.



El bloque tiene las siguientes salidas:

- done_o : indica que se realizó la suma y el resultado en s_o es válido.
- s_o: resultado de la suma a + b.

Para realizar la suma, se deben introducir los números a sumar en formato punto flotante en las entradas *a_i* y *b_i* y luego dar un pulso en *start_i*. El resultado estará en *s_o* y solo será válido cuando la salida *done_o* esté en alto. Se

recomienda utilizar un pulso de un período de reloj en *start_i* para evitar la repetición del cálculo al finalizar.

Algoritmo para resolver la suma en punto flotante

1. Determinamos el tipo de número según el exponente y la mantissa (NaN, infinito, normalizado, normalizado). Si es normalizado agregamos 01 a la mantissa, si es no normalizado agregamos 00 a la mantissa y exponente 1.
2. Desplazamos la mantissa del número con exponente menor para igualar el exponente mayor.
3. Sumamos mantissa. Si tienen signos diferente, realizamos mantissa mayor - mantissa menor. El resultado tomará el signo del número con mayor matissa.
4. Normalizamos la mantissa
5. Redondeamos la mantissa
6. Normalizamos la mantissa
7. Agrupamos signo, exponente y mantissa.

Ejemplo

Sumar los números a = 0.17128097 y b = - 50.75134.

1. sg = 0, exp = 124, mantissa = 01011110110010001000111
   sg = 1, exp = 132, mantissa = 10010110000000101011111

2. sg = 0, exp = 124, mantissa = 01,01011110110010001000111
   sg = 1, exp = 132, mantissa = 01,10010110000000101011111

3. sg = 0, exp = 132, mantissa = 00,00000001010111101100100010001111
   sg = 1, exp = 132, mantissa = 01,10010110000000101011111 (mayor mantissa)

4. 01,10010110000000101011111 (mayor mantissa) => sg_suma = 1
   - 00,00000001010111101100100010001111

   01,10010100101000111111010101110001

5. sg = 1, exp = 132, 01,10010100101000111111010101110001 (ya normalizado)

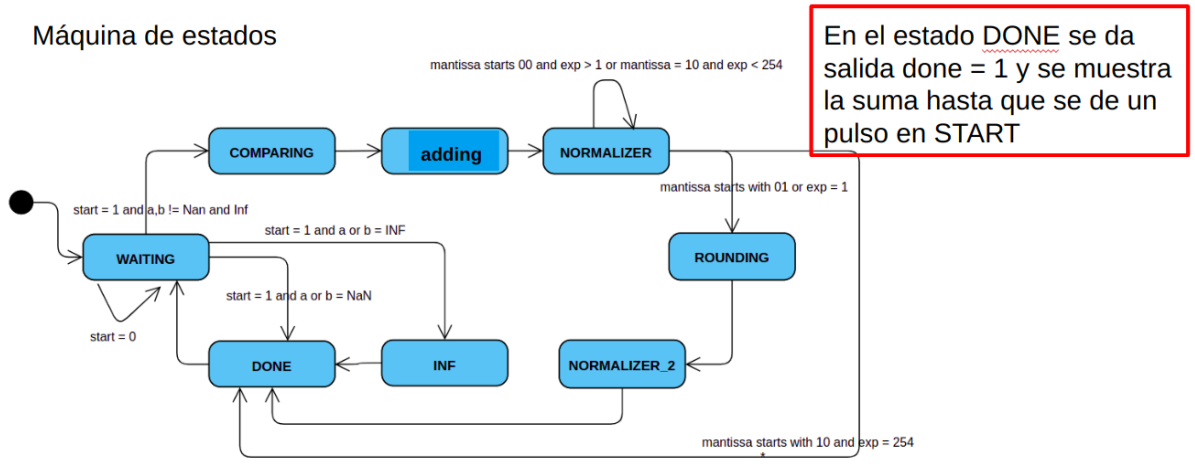
6. sg = 1, exp = 132, 01,10010100101000111111010101110001
   sg = 1, exp = 132, 01,10010100101000111111011 (redondeado)

7. sg = 1, exp = 132, 01,10010100101000111111011 (ya normalizado)

8. Resultado = **01000010010010100101000111111011**

Implementación en VHDL

La implementación en VHDL del sumador se basa en una máquina de estados.



Descripción de los estados

WAITING: Se espera por start y se determina el tipo de número ingresado según el exponente y la mantissa (si es NaN, infinito, normalizado, normalizado). Se guardan los signos, mantissas y exponentes ingresados. Si es normalizado agregamos 01 a la mantissa, si es no normalizado agregamos 00 a la mantissa y el exponente en 1. Si los números ingresados son diferentes de NaN o infinito se procede a seguir el algoritmo para la suma. Si alguno de los números ingresados es NaN ya se tiene el resultado, si alguno es infinito se pasa al estado INF.

COMPARING: Se desplaza la mantissa del número con exponente menor para igualar el exponente mayor.

ADD: Se suman las mantissa. Si tienen signo diferente se realiza mantissa mayor - mantissa menor. El signo del resultado será el signo del número de mayor matissa.

NORMALIZER : se desplaza hasta que la mantissa tenga la forma 01,xxxxx en caso que el exponente lo permita. Si no se puede es un número denormalizado y tendrá exponente 0.

ROUNDING: se redondea la suma del número en la cantidad de bits de la mantissa según la precisión (23 simple, 52 doble).

NORMALIZER_2: se normaliza la mantissa luego del redondeo.

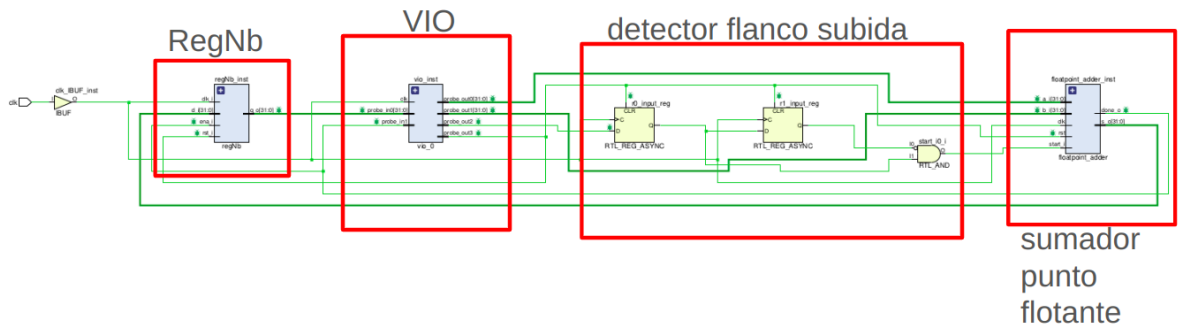
INF: Sí los dos números son infinitos hay que ver el signo, si son del mismo signo el resultado es infinito con el signo, si son de diferente signo el resultado es NaN. Si es

inf + número finito el resultado es infinito con su signo.

DONE: se agrupan signo, exponente y mantissa y se muestra la suma resultante. Se pone la salida done en alto.

2. Diagrama de bloques

Debido a que el diagrama de bloques del sumador de punto flotante es extenso, se lo adjunta en el github del proyecto. Para la prueba en la placa se utilizó un VIO, un registro de 32bits para guardar el resultado y un detector de flanco para iniciar la cuenta.



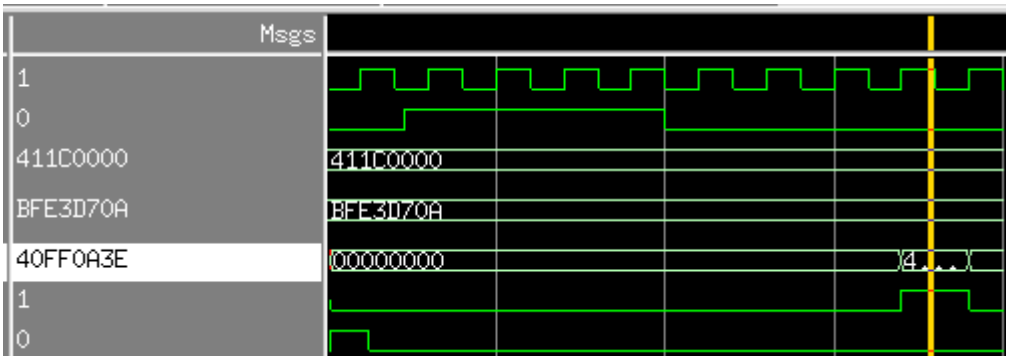
Se utilizó un detector de flanco para evitar que cuando termine la cuenta se vuelva comenzar. Se utilizó un registro de 32 bits para capturar el dato ya que solo se muestra cuando done_o = 1 y dura solo 1 período de reloj.

3. Simulaciones

Para verificar los resultados de la simulación se utilizó el siguiente link http://weitz.de/ieee/.

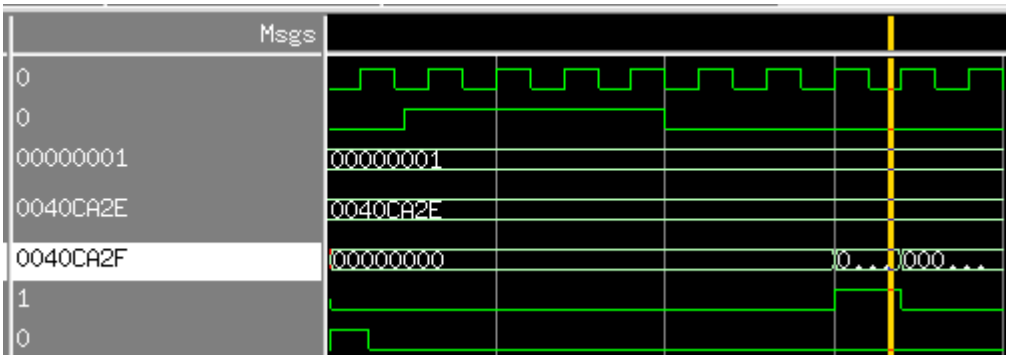
Primer simulación: suma de dos números normalizados

a = 9.75, b = -1.78, res = 7.9700003
a = 0 10000010 00111000000000000000000 (0x411c0000)
b = 1 01111111 11000111101011100001010 (0xbfe3d70a)
res = 0 10000001 11111110000101000111110 (0x40ff0a3e)



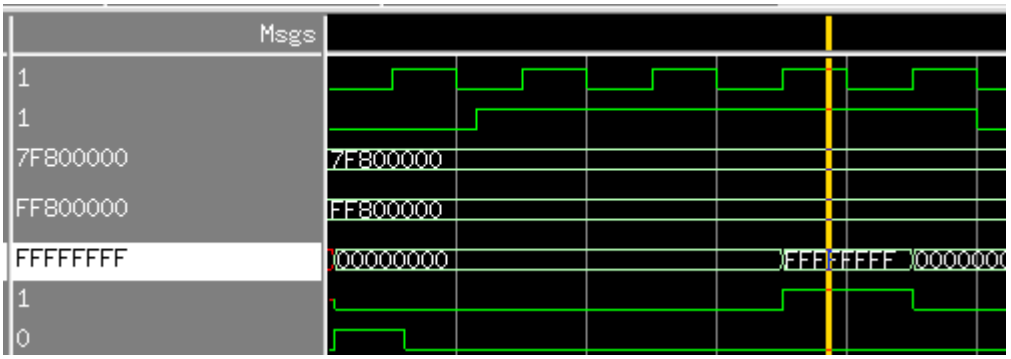
Segunda simulación: suma de dos números desnormalizados

a = 1e-45, b = 5.95e-39, res = 5.950002e-39
a = 00000000000000000000000000000001 (0x00000001)
b = 00000000100000011001010001011110 (0x0040CA2E)
res = 00000000100000011001010001011111 (0x0040CA2F)



Tercera simulación: suma de dos números infinitos

a = +Inf, b =-Inf, res = NaN
a = 01111111100000000000000000000000 (0x7F800000)
b = 11111111100000000000000000000000 (0xFF800000)
res = 11111111111111111111111111111111 (0xFFFFFFFF)



4. Uso de recursos en FPGA

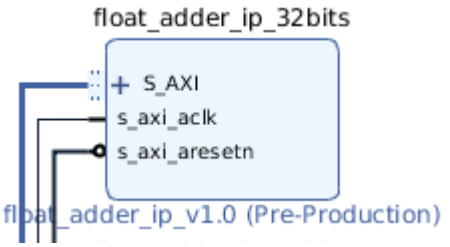
Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	LUT Flip Flop Pairs (20800)	Bonded IOB (210)	BUFGCTRL (32)	BSCANE2 (4)
float32VIO	3616	2344	6	1132	3592	24	1436	1	2	1
dbg_hub (dbg_hub)	463	723	0	224	439	24	299	0	1	1
floatpoint_adder_in...	2880	983	6	791	2880	0	915	0	0	0
regNb_inst (regNb)	0	32	0	9	0	0	0	0	0	0
vio_inst (vio_0)	272	604	0	154	272	0	202	0	0	0

El bloque sumador punto flotante, floatpoint_adder tiene el siguiente uso de recursos:

Name	Slice LUTs (20800)	Slice Register	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LuT as Memory (9600)	LUT Flip Flop Pairs (20800)
floatpoint_adder	2880 (13,8%)	983 (2,4%)	6 (0,04%)	791 (9,7%)	2880 (14%)	0	915 (4.4%)

5. IP

Se desarrolló el IP sumador punto flotante para que se comunique con el micro a través del bus AXI Lite. Este IP core es paramétrico en la precisión (32 bits o 64 bits).



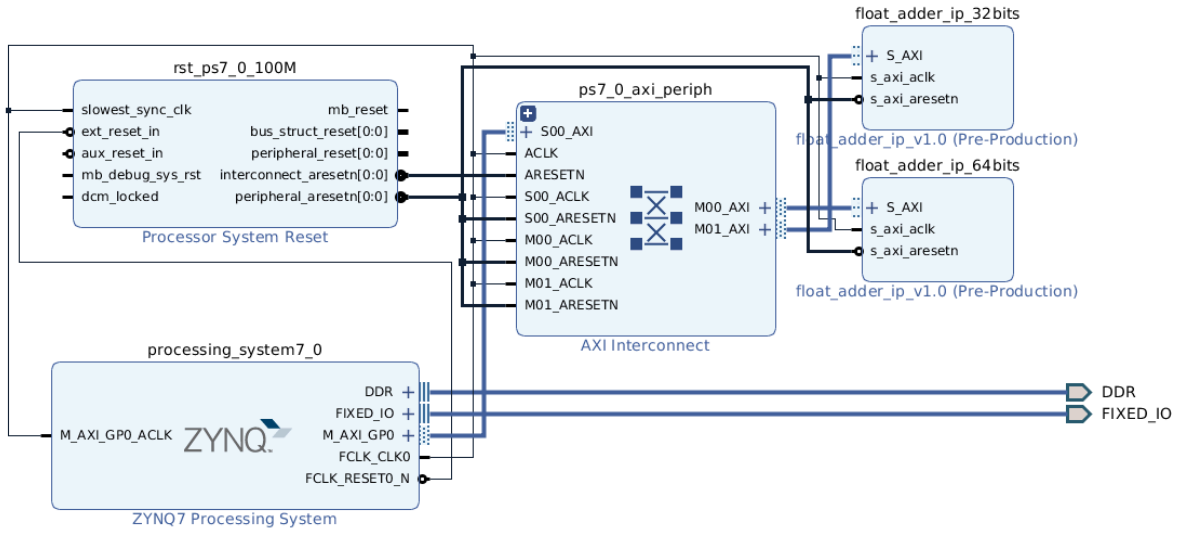
A través de 8 registros se da utilidad al bloque:

- Parámetro A
  - reg0 paramAlow
  - reg1 paramAhigh
- Parámetro B
  - reg2 paramBlow
  - reg3 paramBhigh
- Start
  - reg4(0)
- Output
  - reg5 outHigh
  - reg6 outLow
- Done
  - reg7(0)

Sí el IP float adder es de 32 bits se utilizan las partes bajas de los registros (parámetro A, parámetro B y output).

6. Block design

Se incorporan 2 IP float adder, uno de 32bits y otro de 64bits.



7. Código en C

El código cargado en el microcontrolador realiza lo siguiente:

- Se ingresa mediante la UART que IP core quiere utilizar, según si quiere realizar una suma en 32 o 64 bits. 64 bits no implementado en código

- Luego se ingresan los números A y B según la cantidad de bits correspondiente. Se cargan los valores en los registros reg0 y reg1 el parámetro A (si es de 32 bits solo se utiliza el registro reg0), los registros reg2 y reg3 el parámetro B (si es de 32 bits solo se utiliza el registro reg2).

- Se realiza la suma cargando al bit menos significativo del registro reg3 un 1.

- Se espera en loop a que esté pronta la cuenta a través de la lectura del bit menos significativo del registro reg7.

- Luego se muestra el resultado en consola leyendo los registros reg5 y reg6 (si es de 32 bits la operación solo se lee el registro reg5).