Institut Supérieur
d'Informatique de
Modélisation et de
leurs Applications

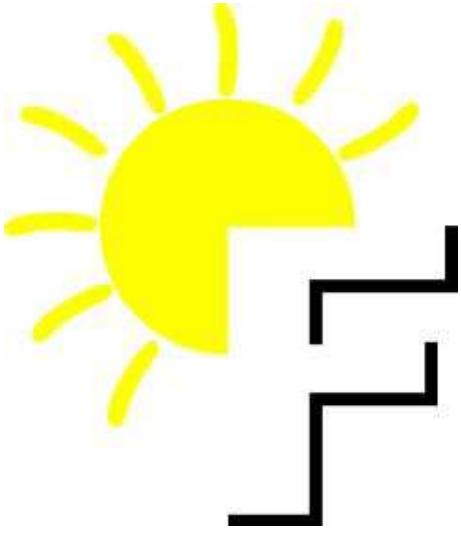
Complexe des Cézeaux
Campus de Clermont-Ferrand
BP 125 - 63173 AUBIERE CEDEX

The Freedom CPU Project
`http://f-cpu.seul.org/`

Rapport de projet de 2ème année

Conception de l'Additionneur de l'Unité de Calcul Flottant pour le projet Freedom-CPU

TOME II. – ANNEXES



Présenté par SEMET Gaëtan
Responsable ISIMA : WODEY Pierre
Responsable du projet F-CPU : GUIDON Yann
Année 2003-2004

Institut Supérieur
d'Informatique de
Modélisation et de
leurs Applications

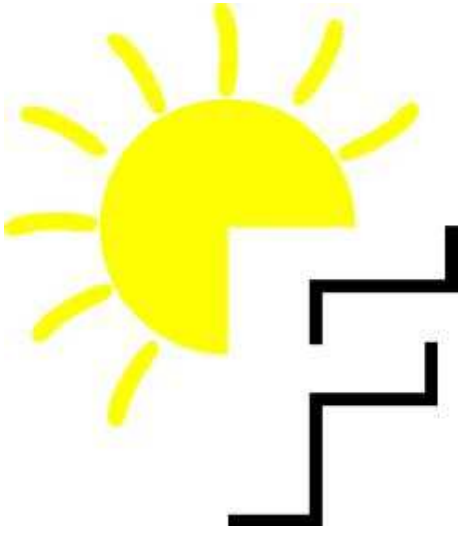
Complexe des Cézeaux
Campus de Clermont-Ferrand
BP 125 - 63173 AUBIERE CEDEX

The Freedom CPU Project
`http://f-cpu.seul.org/`

Rapport de projet de 2ème année

Conception de l'Additionneur de l'Unité de Calcul Flottant pour le projet Freedom-CPU

TOME II. – ANNEXES



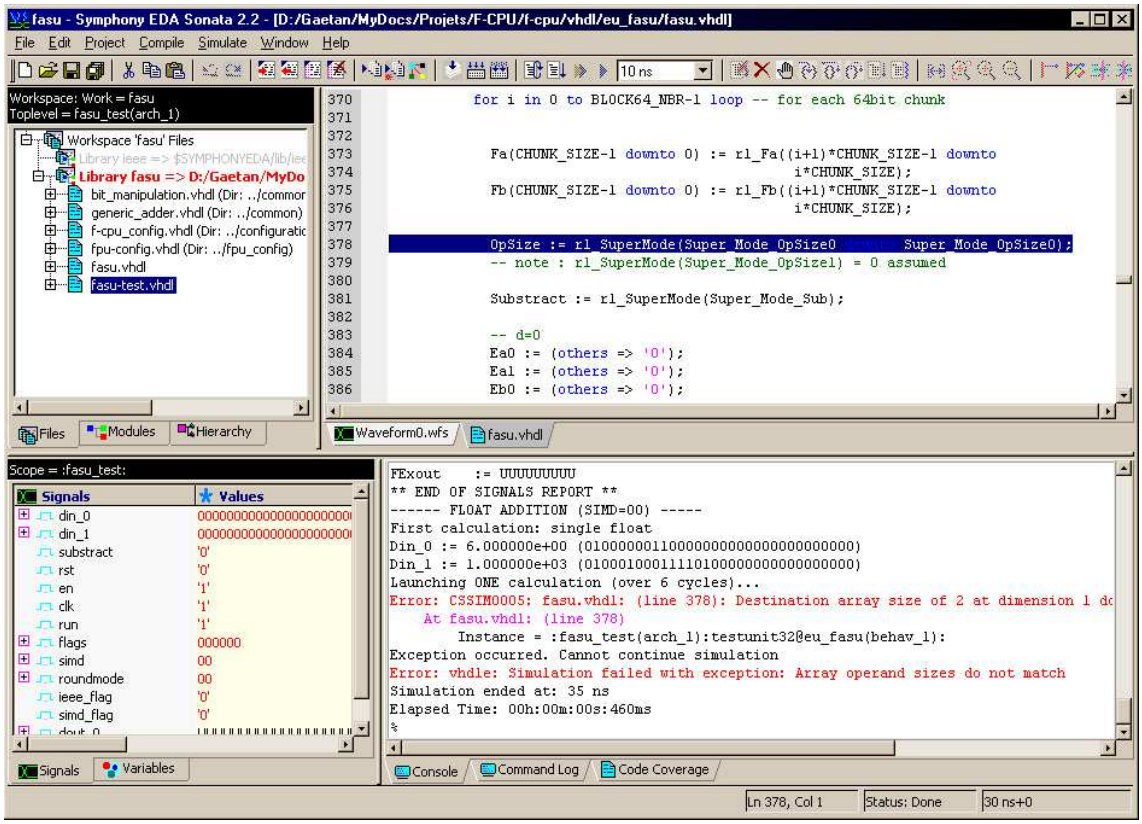
Présenté par SEMET Gaëtan
Responsable ISIMA : WODEY Pierre
Responsable du projet F-CPU : GUIDON Yann
Année 2003-2004

Table des annexes

Annexe A

Environnements de développement

1 Symphony Sonata EDA (Simili)



Annexe B

Codes sources VHDL de l'additionneur en virgule flottante

1 Package commun aux unités de calcul flottant

Le listing suivant correspond au code qui sera commun à toutes les unités de calcul flottant.

```
-------------------------------BEGIN_VHDL_LICENSE-----------------------------------------
-- fpu-config.vhdl - Floating Point Execution Unit Configuration File
--                   for the F-CPU
-- Copyright (C) 2003, 2004 -- SEMET Gaetan <gaetan@xeberon.net>
-- ISIMA Engineering School -- http://www.isima.fr

-- This program is free software; you can redistribute it and/or modify
-- it under the terms of the GNU General Public License as published by
-- the Free Software Foundation; either version 2 of the License, or
-- (at your option) any later version.
--
-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.
--
-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
-------------------------------END_VHDL_LICENSE-------------------------------------------
-- version Thu Dec 30 2003
-- version Mon Jan  5 2004
-- version Mon Feb  2 2004

library ieee;
  use ieee.std_logic_1164.ALL;
  use IEEE.std_logic_arith.ALL;
  use IEEE.std_logic_unsigned.ALL;

        use work.generic_adder.all;
        use work.Bit_Manipulation.all;

package FCPU_FPU_Config is
```

```
35  -- CONSTANTS


    -- single/double mantissa and exponent sizes
    constant  SGL_SIZE    : natural := 32;
40  constant  SGL_M_SIZE  : natural := 23;
    constant  SGL_E_SIZE  : natural :=  8;
    constant  DBL_SIZE    : natural := 64;
    constant  DBL_M_SIZE  : natural := 52;
    constant  DBL_E_SIZE  : natural := 11;
45  -- signs are always 1 bit long...

    -- single/double mantissa and exponent positions in the different SIMD modes
    -- SIMD 32 bit mode
    constant SGL_M_Start : natural :=  0;
50  constant SGL_M_End   : natural := 22;
    constant SGL_E_Start : natural := 23;
    constant SGL_E_End   : natural := 30;
    constant SGL_S_Pos   : natural := 31;
    -- SIMD 64 bit mode :
55  constant DBL_M_Start : natural :=  0;
    constant DBL_M_End   : natural := 51;
    constant DBL_E_Start : natural := 52;
    constant DBL_E_End   : natural := 62;
    constant DBL_S_Pos   : natural := 63;
60  -- TODO: don't know VHDL syntax for having Fa(SGL2_M_POS) with
    -- SGL2_M_POS = "SGL2_M_End downto SGL2_M_Start"

    -- indices for Super Mode and Mode vectors
    constant SUPER_MODE_VECTOR_SIZE : natural := 7; -- Super Mode vector size
65
    constant Super_Mode_IEEE        : natural :=  0;
            -- 0: IEEE exception flag not set
            -- 1: IEEE exception flag not set
    constant Super_Mode_Sub         : natural :=  1; -- 0: add, 1: subtraction
70
    constant Super_Mode_SIMD        : natural :=  2; -- 0: add, 1: subtraction
    constant Super_Mode_OpSize0     : natural :=  3; -- 0 : 32 bit (single)
                                                     -- 1 : 64 bit (double)
    constant Super_Mode_OpSize1     : natural :=  4; -- reserved for further
75                                                   -- floating formats
    constant Super_Mode_RoundMode0  : natural :=  5; -- Rounding Mode lowest bit
    constant Super_Mode_RoundMode1  : natural :=  6; -- Rounding Mode highest bit



80


    constant MODE_VECTOR_SIZE       : natural := 7;  -- Mode vector size

    constant Mode_EX0  : natural :=  0; -- exceptions indicator bit 0
85  constant Mode_EX1  : natural :=  1; -- exceptions indicator bit 1
    constant Mode_EX2  : natural :=  2; -- exceptions indicator bit 2
              -- 000 : no exception occured
              -- 100 : DBZ
              -- 101 : IOD
90            -- 110 : EUD
              -- 111 : EOD
```

```
    constant Mode_SO0   : natural :=  3; -- Special Output indicator bit 0
    constant Mode_SO1   : natural :=  4; -- Special Output indicator bit 1
    constant Mode_SO2   : natural :=  5; -- Special Output indicator bit 2
95              -- 000 : no special output
                -- 001 : output is directly in Fout vector
                -- 010 : SNaN
                -- 011 : QNaN
                -- 100 : +inf
100             -- 101 : -inf
                -- 110 : +0

                -- IMPORTANT: if Mode_SpecialOutput is set to Fb (Mode_SO_Fb),
                -- the result sign will be not the sign of Fb if we are
105             -- in substration instruction => the last stage will have
                -- to invert the sign bit of Fb

    constant Mode_DONE : natural := 6;  -- finished calculation flag
                                        -- if Mode_EX != "000" then result is
110                                     -- an exception (NaN, inf,...)
                                        -- or the result is in Fout vector
                                        -- (in the current stage)
    -- Mode sub-vectors values
    constant Mode_OpSize_Single : std_ulogic_vector(1 downto 0) := "00";
115 constant Mode_OpSize_Double : std_ulogic_vector(1 downto 0) := "01";

    constant Mode_RoundMode_Nearest : std_ulogic_vector(1 downto 0) := "00";
    constant Mode_RoundMode_Zero    : std_ulogic_vector(1 downto 0) := "01";
    constant Mode_RoundMode_mInfty  : std_ulogic_vector(1 downto 0) := "10";
120 constant Mode_RoundMode_Infty   : std_ulogic_vector(1 downto 0) := "11";


    constant Mode_Ex_None     : std_ulogic_vector(2 downto 0) := "000";
    constant Mode_Ex_DBZ      : std_ulogic_vector(2 downto 0) := "100";
125 constant Mode_Ex_IOD      : std_ulogic_vector(2 downto 0) := "101";
    constant Mode_Ex_EUD      : std_ulogic_vector(2 downto 0) := "110";
    constant Mode_Ex_EOD      : std_ulogic_vector(2 downto 0) := "111";

    constant Mode_SO_None     : std_ulogic_vector(2 downto 0) := "000";
130 constant Mode_SO_Fout     : std_ulogic_vector(2 downto 0) := "001";
    constant Mode_SO_SNaN     : std_ulogic_vector(2 downto 0) := "010";
    constant Mode_SO_QNaN     : std_ulogic_vector(2 downto 0) := "011";
    constant Mode_SO_PINFTY   : std_ulogic_vector(2 downto 0) := "100";
    constant Mode_SO_MINFTY   : std_ulogic_vector(2 downto 0) := "101";
135 constant Mode_SO_ZERO     : std_ulogic_vector(2 downto 0) := "110";


    -- indicies of Repres representation vector
    -- (for easy recognition of input type through the pipeline)
140 constant REPRES_VECTOR_SIZE         : natural := 3;

    constant REPRES_NORMALISED    : std_ulogic_vector(2 downto 0) := "000";
    constant REPRES_DENORMALISED  : std_ulogic_vector(2 downto 0) := "001";
    constant REPRES_SNAN          : std_ulogic_vector(2 downto 0) := "010";
145 constant REPRES_QNAN          : std_ulogic_vector(2 downto 0) := "011";
    constant REPRES_PINFTY        : std_ulogic_vector(2 downto 0) := "100";
    constant REPRES_MINFTY        : std_ulogic_vector(2 downto 0) := "101";
    constant REPRES_ZERO          : std_ulogic_vector(2 downto 0) := "110";
```

```
  constant REPRES_MZERO         : std_ulogic_vector(2 downto 0) := "111";

  -- indices for FExout exception output vector
  constant Ex_DBZ  : natural := 0;  --  Divide By Zero Detected
  constant Ex_IOD  : natural := 1;  --  Illegal Operand Detected
  constant Ex_EUD  : natural := 2;  --  Exponent Underflow Detected
  constant Ex_EOD  : natural := 3;  --  Exponent Overflow Detected
  constant Ex_SNAN : natural := 4;  --  Signaling Not A Number output
  constant Ex_QNAN : natural := 5;  --  Quiet Not A Number output
  constant Ex_INF  : natural := 6;  --  Infinite output
  constant Ex_INE  : natural := 7;  --  Inexact output
  constant Ex_ZERO : natural := 8;  --  Zero output


  -- special types
  type t_float is (t_single, t_double);

  -- special Floating point values for outputs
  -- single floats (32 bit)
  constant const32_infty   : std_ulogic_vector
                           := "01111111100000000000000000000000"; --  +infinity
  constant const32_m_infty : std_ulogic_vector
                           := "11111111100000000000000000000000"; --  -infinity
  constant const32_SNaN    : std_ulogic_vector
                           := "01111111100000000000000000000001"; --    SNaN
  constant const32_QNaN    : std_ulogic_vector
                           := "01111111110000000000000000000001"; --    QNaN
  constant const32_zero    : std_ulogic_vector
                           := "00000000000000000000000000000000"; --       0
  constant const32_m_zero  : std_ulogic_vector
                           := "10000000000000000000000000000000"; --      -0

  -- double floats (64 bit)
  constant const64_infty   : std_ulogic_vector                                 --  +infinity
        := "0111111111110000000000000000000000000000000000000000000000000000";
  constant const64_m_infty : std_ulogic_vector                                 --  -infinity
        := "1111111111110000000000000000000000000000000000000000000000000000";
  constant const64_SNaN    : std_ulogic_vector                                 --    SNaN
        := "0111111111110000000000000000000000000000000000000000000000000001";
  constant const64_QNaN    : std_ulogic_vector                                 --    QNaN
        := "0111111111111000000000000000000000000000000000000000000000000001";
  constant const64_zero    : std_ulogic_vector                                 --       0
        := "0000000000000000000000000000000000000000000000000000000000000000";
  constant const64_m_zero  : std_ulogic_vector                                 --      -0
        := "1000000000000000000000000000000000000000000000000000000000000000";


-- procedures

  -- right shift F by N (coded digit), fill with G bit
  procedure fpu_rshift(F : in  std_ulogic_vector; N: in std_ulogic_vector;
                       G : in  std_ulogic_vector;
                       Y : out std_ulogic_vector; B : out std_ulogic_vector
                       );
  -- right shift like fpu_rshift but producing a Sticky bit
  procedure fpu_rshift(F : in  std_ulogic_vector; N: in std_ulogic_vector;
                        G : in  std_ulogic_vector;
```

```
                        Y : out std_ulogic_vector; B : out std_ulogic_vector;
                        s : out std_ulogic
                       );

  -- integer incrementer
  function fpu_incr(F: std_ulogic_vector) return std_ulogic_vector;

  -- integer decrementer
  function fpu_decr(F: std_ulogic_vector) return std_ulogic_vector;

  -- integer negation
  function fpu_neg(F: std_ulogic_vector) return std_ulogic_vector;

  -- integer compound adder (computes both A+B and A+B+1)
  procedure fpu_add (A, B : in std_ulogic_vector;
                     Y, Z : out std_ulogic_vector;
                     G, P : out std_ulogic);

  -- integer substraction
  -- A and B are unsigned integer representation
  -- Y is UNSIGNED integer representation
  -- C represents the sign of result
  procedure fpu_sub (A, B : in std_ulogic_vector;  Y : out std_ulogic_vector;
                     C : out std_ulogic);

  -- return true if F only have 0
  function fpu_isnull(F: std_ulogic_vector) return boolean;

        -- carry select vectors
        procedure fpu_csv (G, P : in std_ulogic_vector;
                     S, T : out std_ulogic_vector);

        -- carry look-ahead
        procedure fpu_cla (Gi, Pi : in std_ulogic_vector;
                                          Go, Po : out std_ulogic_vector);

        procedure fpu_half_adder(A, B : in  std_ulogic_vector;
                                 C, S : out std_ulogic_vector);

end FCPU_FPU_Config;


package body FCPU_FPU_Config is


  -- this function right shift F by N bits
  -- != from rshift from bit_manipulation.vhdl where N is a natural
  -- here N is unknown (it's a bit vector)
  -- mantissa shifting goes from 0 to 52 so we can hardwire it directly
  -- (N will be only coded with 6 bits)
  -- G : what bits to insert ?
  -- S : sticky bit ("or" on every droped bits out of F & B)
  procedure fpu_rshift(F : in  std_ulogic_vector; N: in std_ulogic_vector;
                       G : in  std_ulogic_vector;
                       Y : out std_ulogic_vector; B : out std_ulogic_vector;
                       s : out std_ulogic
                      ) is
```

```
    constant LF : natural := F'length;
    constant LN : natural := N'length;
    constant LY : natural := Y'length;
    constant LB : natural := B'length;
    constant LG : natural := G'length;
    constant L  : natural := LG+LF+LB;
    variable gg : std_ulogic_vector(LG-1 downto 0) := G;
    variable ff : std_ulogic_vector(LF-1 downto 0) := F;
    variable yy : std_ulogic_vector(L-1 downto 0);
    variable ss : std_ulogic;
    constant default_bit : std_ulogic := '0';
  begin
        yy(L-1 downto LB) := gg(LG-1 downto 0) & ff(LF-1 downto 0);
        yy(LB-1 downto 0) := (others => default_bit);
        ss := '0';
    for i in 0 to LN-1 loop
          if (N(i) = '1') then
            -- reduce_or need a vector with size 4.n
            if i=0 then
              ss := ss or yy(0);
            elsif i=1 then
              ss := ss or yy(0) or yy(1);
            else
                    ss := ss or reduce_or(yy(2**i-1 downto 0));
                end if;
            yy(L-2**i-1 downto 0) := yy(L-1 downto 2**i);
            yy(L-1 downto L-2**i) := (others => default_bit);
          end if;
    end loop;

    Y := yy(LF+LB-1 downto LB);
    B := yy(LB-1    downto 0);

    -- i've read somewhere than the unrolled form of this shifter generates
    -- better hardware...

--    if (N(0) = '1') then
--      yy(L-1) := G;
--      yy(L-2 downto 0) := F(L-1 downto 1);
--     else
--       yy := F;
--     end if;
--    if (N(1) = '1') then
--      yy(L-1) := G;
--      yy(L-2) := G;
--      yy(L-2-1 downto 0) := yy(L-1 downto 2);
--    end if;
--
--    if (N(2) = '1') then
--      yy(L-1 downto L-3-1) := (others => G);
--      yy(L-4-1 downto 0) := yy(L-1 downto 4);
--    end if;
--
--    if (N(3) = '1') then
--      yy(L-1 downto L-7-1) := (others => G);
--      yy(L-8-1 downto 0) := yy(L-1 downto 8);
--    end if;
```

```
--
--    if (N(4) = '1') then
--      yy(L-1 downto L-15-1) := (others => G);
--      yy(L-16-1 downto 0) := yy(L-1 downto 16);
--    end if;
--
--    if (L >= DBL_M_SIZE) then
--      if (N(5) = '1') then
--        yy(L-1 downto L-31-1) := (others => G);
--        yy(L-32-1 downto 0) := yy(L-1 downto 32);
--      end if;
--    end if;
--
  end procedure;
  -- hope this is not too deep...
  -- TODO: delay estimation

  procedure fpu_rshift(F : in  std_ulogic_vector; N: in std_ulogic_vector;
                       G : in  std_ulogic_vector;
                       Y : out std_ulogic_vector; B : out std_ulogic_vector
                      ) is
      variable s : std_ulogic;
      begin
              fpu_rshift(F, N, G, Y, B, s);
      end procedure;


  -- integer incrementer
  function fpu_incr(F: std_ulogic_vector) return std_ulogic_vector is
  begin
    return F xor lshift(cascade_and(F), 1, '1');
  end function;
  -- TODO: delay estimation


  -- integer decrementer
  function fpu_decr(F: std_ulogic_vector) return std_ulogic_vector is
  begin
    return not fpu_incr(not F);
  end function;
  -- TODO: delay estimation

  -- integer negation
  function fpu_neg(F: std_ulogic_vector) return std_ulogic_vector is
  begin
    return fpu_incr(not F);
  end function;
  -- TODO: delay estimation

  -- 8 bit exponent substraction
  -- A and B are unsigned integer representation
  -- Y is unsigned integer representation
  -- C represents the sign of result
  procedure fpu_sub(A, B : in std_ulogic_vector;  Y : out std_ulogic_vector;
                    C : out std_ulogic) is
  constant L : natural := A'length;
  variable a1, b1: std_ulogic_vector(L downto 0);
```

```
    variable a2, b2: std_ulogic_vector(L downto 0);
    variable r1,ri1 : std_ulogic_vector(L downto 0);
    variable r2,ri2: std_ulogic_vector(L downto 0);
380 variable g1, p1 : std_ulogic;
    variable g2, p2 : std_ulogic;
    begin -- A - B = A + not(B) + 1
          -- use generic adder Carry select adder to have directly A + B~ +1

385   -- computing A - B
      a1(L) := '0';
      a1(L-1 downto 0) := A;
      b1(L) := '0';
      b1(L-1 downto 0) := B;
390   CSAdd(a1, not b1, r1, ri1, g1, p1);

      -- computing B - A
      a2(L) := '0';
      a2(L-1 downto 0) := B;
395   b2(L) := '0';
      b2(L-1 downto 0) := A;
      CSAdd(a2, not b2, r2, ri2, g2, p2);

      -- r1  is A + not(B)
400   -- ri1 is A + not(B) + 1
      -- r2  is B + not(A)
      -- ri2 is B + not(A) + 1

      if (r1(L) = '1') then -- result < 0
405     -- if A-B is < 0, then B-A is > 0...
        Y := ri2(L-1 downto 0);
      else
        Y := ri1(L-1 downto 0);
      end if;
410   C := r1(L);
    end procedure;
    -- note: does it fit into 1 cycle???
    -- TODO: delay estimation

415 procedure fpu_add(A, B : in std_ulogic_vector;
                      Y, Z : out std_ulogic_vector;
                      G, P : out std_ulogic) is

    begin
420   CSAdd(A, B, Y, Z, G, P);
    end procedure;

    -- return true if F only have 0
    function fpu_isnull(F: std_ulogic_vector) return boolean is
425   variable m   : std_logic_vector(1 to F'length);
    begin
      m := To_StdLogicVector(F);
      return (m=0); -- better use of cascaded_or????
    end;
430
        -- carry select vector
        procedure fpu_csv (G, P : in std_ulogic_vector;
                      S, T : out std_ulogic_vector) is
```

```
  begin
    CSV(G,P,S,T);
  end;

        -- carry look-ahead
        procedure fpu_cla (Gi, Pi : in std_ulogic_vector;
                                    Go, Po : out std_ulogic_vector) is
        begin
          CLA(Gi,Pi,Go,Po);
        end;

        procedure fpu_half_adder(A, B : in  std_ulogic_vector;
                                 C, S : out std_ulogic_vector) is
    constant LA : natural := A'Length;
    constant LB : natural  := A'Length;
    constant LC : natural  := C'Length;
    constant LS : natural  := S'Length;
    variable aa : std_ulogic_vector(A'Length-1 downto 0) := A;
    variable bb : std_ulogic_vector(B'Length-1 downto 0) := B;
  begin
--pragma synthesis_off
    assert LB = LA;
                assert LC = LA;
                assert LS = LA;
--pragma synthesis_on

     for i in 0 to LA-1 loop
       C(i) := A(i) and B(i);
       S(i) := A(i) xor B(i);
     end loop;

   end procedure;

end FCPU_FPU_Config;
```

2 Unité Fasu

Voici le code source de l'unité d'exécution fasu (Floating Point Adder Execution Unit).

```
-------------------------------BEGIN_VHDL_LICENSE----------------------------------------
-- fasu.vhdl - Floating Point Add/Sub Execution Unit for the F-CPU
-- Copyright (C) 2003, 2004 -- SEMET Gaetan <gaetan@xeberon.net>
-- ISIMA Engineering School -- http://www.isima.fr
-- use some integer adder-related Michael Riepe code:
-- Copyright (C) 2000, 2001, 2003 Michael Riepe <michael@stud.uni-hannover.de>

-- This program is free software; you can redistribute it and/or modify
-- it under the terms of the GNU General Public License as published by
-- the Free Software Foundation; either version 2 of the License, or
-- (at your option) any later version.
--
-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.
--
-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
-------------------------------END_VHDL_LICENSE------------------------------------------
-- version Thu Dec 30 2003
-- version Mon Jan  5 2004
--
-- huge thanks to Michael Riepe for his help !

library IEEE;
use IEEE.std_logic_1164.all;

use work.FCPU_config.all;
use work.Bit_Manipulation.all;
use work.Generic_Adder.all;


use work.FCPU_FPU_Config.all;

entity EU_FASU is
  generic (
    WIDTH : natural := 64
  );

  port(
  -- inputs :
    -- operands
    Din_0     : in std_ulogic_vector(WIDTH-1 downto 0);
    Din_1     : in std_ulogic_vector(WIDTH-1 downto 0);
    -- subtract flag (should be derived from opcode)
    -- 0 for fadd, 1 for fsub
    Substract : in std_ulogic;
    ieee_flag: in std_ulogic; -- 0 : ieee flag not set, 1: ieee flag set
    simd_flag: in std_ulogic; -- 0 : not SIMD operation, 1: SIMD operation
    -- SIMD mode bits (not decoded)
```

```
    SIMD      : in std_ulogic_vector(1 downto 0); -- 00 32 bit FP, 01 64 bit FP
    -- Rounding Mode selection vector:
    RoundMode: in std_ulogic_vector(1 downto 0);
               -- 00 : nearest, 01: zero, 10: +inf, 11: -inf
    -- clock/reset/enable inputs
    Clk       : in std_ulogic;
    Rst       : in std_ulogic;
    En        : in std_ulogic;

  -- outputs:
    -- Result (8/16/32)
    Dout_0    : out std_ulogic_vector(WIDTH-1 downto 0);
    -- FPU Exception Flags
    FExout    : out std_ulogic_vector(8 downto 0);
    -- FExout(0) : DBZ  : Divide By Zero Detected
    -- FExout(1) : IOD  : Illegal Operand Detected
    -- FExout(2) : EUD  : Exponent Underflow Detected
    -- FExout(3) : EOD  : Exponent Overflow Detected
    -- FExout(4) : SNAN : Signaling Not A Number output
    -- FExout(5) : QNAN : Quiet Not A Number output
    -- FExout(6) : INF  : Infinite output
    -- FExout(7) : INE  : Inexact output
    -- FExout(8) : ZERO : Zero output
    excep_trigged : out std_ulogic
    -- "exceptions can be trigged" flag
    -- (when ieee_flag is clear, the pipeline is stopped until
    --  the FP confirms that exceptions should be trigged)
    -- TODO
  );
--pragma synthesis_off
begin
  assert (WIDTH >= 64) and (WIDTH mod 64 = 0)
    report "width of FASU must be an integer multiple of 64"
    severity failure;
--pragma synthesis_on
end EU_FASU;


-- Outputs:
--                +-------------------------------------------------------+
--   Addition     |                          B                            |
--    A + B       +-----------+-----------+-----------+-----------+-------+
--                |     F     |  -infty   |  +infty   |   QNaN    | SNaN  |
--+---+-----------+-----------+-----------+-----------+-----------+-------+
--|   | F         |     F     |  -infty   |  +infty   |   QNaN    | SNaN  |
--|   | -infty    |  -infty   |  -infty   |   QNaN    |   QNaN    | SNaN  |
--| A | +infty    |  +infty   |   QNaN    |  +infty   |   QNaN    | SNaN  |
--|   | QNaN      |   QNaN    |   QNaN    |   QNaN    |   QNaN    | SNaN  |
--|   | SNaN      |   SNaN    |   SNaN    |   SNaN    |   SNaN    | SNaN  |
--+---+-----------+-----------+-----------+-----------+-----------+-------+
--
--                +-------------------------------------------------------+
--   Substraction |                          B                            |
--    A - B       +-----------+-----------+-----------+-----------+-------+
--                |     F     |  -infty   |  +infty   |   QNaN    | SNaN  |
--+---+-----------+-----------+-----------+-----------+-----------+-------+
--|   | F         |     F     |  +infty   |  -infty   |   QNaN    | SNaN  |
```

```
--| A  | -infty   | -infty   |   QNaN   | -infty   | QNaN   | SNaN   |
--|    | +infty   | +infty   | +infty   |   QNaN   | QNaN   | SNaN   |
--|    | QNaN     | QNaN     | QNaN     |   QNaN   | QNaN   | SNaN   |
--|    | SNaN     | SNaN     | SNaN     |   SNaN   | SNaN   | SNaN   |
--+----+----------+----------+----------+----------+--------+--------+
--
-- F : any finite floating value, ie except {S,Q}NaN, +/-infinity

-- Note FPU doesn't generate Signaling Not A Number (SNaN) except if
-- a SNaN was given in one operand.
-- Rem: IEEE wants a SNAN to be generated when overflow occurs, in place of
-- a QNAN
-- Rem : QNaN must be propagated AS it is (different representation of QNaN possible)
-- QNAN isgenerated when an illegal operation occurs
-- Rem : FPU will not ouput -0 but +0, even if two -0 were given.


-- Operating mode:
--   Single: Float    : 32 bits
--           Mantissa : 23 bits
--           Exponent :  8 bits
--           Sign     :  1 bit
--   Double: Float    : 64 bits
--           Mantissa : 52 bits
--           Exponent : 11 bits
--           Sign     :  1 bit
--
-- SIMD Modes:
--            00 : 32 bit mode (float)
--            01 : 64 bit mode (double)
--            any other combinaison is invalid
--
-- Rounding Modes:
--            00 : Rounding to nearest
--            01 : Rounding to zero
--            10 : Rounding to +infinity
--            11 : Rounding to -infinity


architecture Behav_1 of EU_FASU is

  -- value classes (thx to M.R. for this)
  subtype FP_CLASS is std_ulogic_vector(2 downto 0);
  type FP_CLASS_VECTOR is array ( NATURAL RANGE <> ) of FP_CLASS;

  -- possible values
  -- Note: "10x" is impossible
  constant CLASS_ZERO      : FP_CLASS := "000";
  constant CLASS_DENORMAL  : FP_CLASS := "001";
  constant CLASS_NORMAL_1  : FP_CLASS := "010";
  constant CLASS_NORMAL_2  : FP_CLASS := "011";
  constant CLASS_INF       : FP_CLASS := "110";
  constant CLASS_NAN       : FP_CLASS := "111";


  -- number of 64 bit blocks by word
  constant  CHUNK_SIZE   : natural := 64;
```

```
    constant  BLOCK64_NBR : natural := WIDTH/CHUNK_SIZE;


170 -- SIGNALS

    -- generals signals
    signal Fout           : std_ulogic_vector (BLOCK64_NBR*WIDTH-1 downto 0);

175 -- stage 1 inputs registers
    signal r1_Enable      : std_logic;
    signal r1_SuperMode   : std_ulogic_vector(SUPER_MODE_VECTOR_SIZE-1 downto 0);
    --signal r1_Modes       : std_ulogic_vector(2*BLOCK64_NBR*MODE_VECTOR_SIZE-1
    signal r1_Fa, r1_Fb   : std_ulogic_vector(BLOCK64_NBR*CHUNK_SIZE-1 downto 0);
180
--  -- stage 1 output, stage 2 input registers
--  signal r2_Enable      : std_logic;
--  signal r2_SuperMode   : std_ulogic_vector(SUPER_MODE_VECTOR_SIZE-1 downto 0);
--  signal r2_gm00, r2_gm01,
185 --          r2_pm00, r2_pm01 : std_ulogic_vector(BLOCK64_NBR*SGL_E_SIZE-1 downto 0);
--  signal r2_sv00, r2_sv01,
--          r2_tv00, r2_tv01 : std_ulogic_vector(BLOCK64_NBR*SGL_E_SIZE-1 downto 0);
--
--  signal r2_gm10, r2_gm11,
190 --          r2_pm10, r2_pm11 : std_ulogic_vector(BLOCK64_NBR*DBL_E_SIZE-1 downto 0);
--  signal r2_sv10, r2_sv11,
--          r2_tv10, r2_tv11 : std_ulogic_vector(BLOCK64_NBR*DBL_E_SIZE-1 downto 0);
--
--  signal r2_Class_a_DBL, r2_Class_a_hSGL,
195 --          r2_Class_a_lSGL : FP_CLASS_VECTOR(BLOCK64_NBR-1 downto 0);
--  signal r2_Class_b_DBL, r2_Class_b_hSGL,
--          r2_Class_b_lSGL : FP_CLASS_VECTOR(BLOCK64_NBR-1 downto 0);
--
--  signal r2_Fa, r2_Fb : std_ulogic_vector(BLOCK64_NBR*CHUNK_SIZE-1 downto 0);
200 --
--  signal r2_EffSubDBL, r2_EffSubSGL : std_ulogic_vector(BLOCK64_NBR-1 downto 0);
--
--
--  -- stage 2 output and stage 3 input registers
205 --  signal r3_Enable      : std_logic;
--  signal r3_SuperMode : std_ulogic_vector (SUPER_MODE_VECTOR_SIZE-1 downto 0);
--  signal r3_Modes     : std_ulogic_vector (2*BLOCK64_NBR*MODE_VECTOR_SIZE-1
--                                           downto 0);
--
210




-- INTERNAL FUNCTIONS
215
  -- number of exponent bits in <format>
  function expt_bits (format : natural) return natural is
  begin
    case format is
220   when 32 => return  8;  -- IEEE single
      when 64 => return 11;  -- IEEE double
      when others => null;
    end case;
```

```
  --pragma synthesis_off
    assert false report "unsupported FP format" severity failure;
  --pragma synthesis_on
  end expt_bits;

  -- return a value's class
  function classify (X : std_ulogic_vector) return FP_CLASS is
    constant L : natural := X'length;
    constant E : natural := expt_bits(L);
    constant E_high : natural := L-2;
    constant E_low  : natural := L-E-1;
    constant M_high : natural := L-E-2;
    constant M_low  : natural := 0;
    variable xx : std_ulogic_vector(L-1 downto 0) := to_X01(X);
    variable yy : FP_CLASS;
  begin
    yy(0) := reduce_or(xx(M_high downto M_low));
    yy(1) := reduce_or(xx(E_high downto E_low));
    yy(2) := reduce_and(xx(E_high downto E_low));
    return yy;
  end classify;




--
begin



-- FLOATING POINT ADD/SUB UNIT STAGES :


  r1_Fa <= Din_0(WIDTH-1 downto 0);
  r1_Fb <= Din_1(WIDTH-1 downto 0);
  Dout_0 <= Fout;

  r1_Enable <= En;

  r1_SuperMode(Super_Mode_IEEE) <= ieee_flag;
  r1_SuperMode(Super_Mode_Sub)  <= Substract;
  r1_SuperMode(Super_Mode_SIMD) <= simd_flag;
  r1_SuperMode(Super_Mode_OpSize0)<= SIMD(0);
  r1_SuperMode(Super_Mode_OpSize1)<= SIMD(1);
  r1_SuperMode(Super_Mode_RoundMode0) <= RoundMode(0);
  r1_SuperMode(Super_Mode_RoundMode1) <= RoundMode(1);


  -- stage 1 : start DE calculation and // trivial case checks
  stage_1 : process (Clk, Rst)

    -- first part of a integer adder
    -- deliberately inspirated by M. Riepe's CSAdd function
    -- estimated delay : d=4, t=5
    procedure fasu_ExpAdder_PartOne(a, b : in std_ulogic_vector;
```

```
                                        p, s, t, y, z : out std_ulogic_vector) is
    constant L : natural := a'length;
    variable aa, bb : std_ulogic_vector(L-1 downto 0);
    variable sv, tv : std_ulogic_vector(L-1 downto 0);
    variable gm, pm : std_ulogic_vector(L-1 downto 0);
    variable gt, pt : std_ulogic_vector(L-1 downto 0);
    variable ym, zm : std_ulogic_vector(L-1 downto 0);
  begin
    --pragma synthesis_off
    assert a'length = L;
    assert b'length = L;
    assert p'length = L;
    assert s'length = L;
    assert t'length = L;
    --pragma synthesis_on

    -- normalize inputs
    aa := A;
    bb := B;

    -- a row of 4-bit adders
    -- (d=0)
    gm := aa and bb; -- d=1, t=1
    pm := aa xor bb; -- d=1, t=2
    -- ( d=1 t=2 )
    CSV(gm, pm, sv, tv); -- d=2, t=2
    -- (d=3, t=4)

    -- (d=1)
    gt := gm; pt := pm;
    CLA(gt, pt, gm((L-1)/4 downto 0), pm((L-1)/4 downto 0));
    -- (d=3)

    -- (d=3)
    ym := pm xor sv;
    zm := pm xor tv;
    -- (d=4)

    p := pm;
    s := sv;
    t := tv;
    y := ym;
    z := zm;
  end;


  variable Fa       : std_ulogic_vector(CHUNK_SIZE-1 downto 0);
  variable Fb       : std_ulogic_vector(CHUNK_SIZE-1 downto 0);

  variable gm0, pm0, sv0, tv0 : std_ulogic_vector(SGL_E_SIZE-1 downto 0);

  variable gm1, pm1, sv1, tv1 : std_ulogic_vector(DBL_E_SIZE-1 downto 0);

  variable Ea0, Eb0 : std_ulogic_vector(SGL_E_SIZE-1 downto 0);
  variable Ea1, Eb1 : std_ulogic_vector(DBL_E_SIZE-1 downto 0);

  variable y0, z0 : std_ulogic_vector(SGL_E_SIZE-1 downto 0);
```

```
    variable y1, z1 : std_ulogic_vector(DBL_E_SIZE-1 downto 0);

    variable EffSubDBL, EffSubSGL : std_ulogic;

    variable Substract : std_ulogic;

    -- only for representation detection, there is 3 datapaths:
    -- see operands as DOUBLE
    --               as 2 SINGLES
--    variable Fa_raE_DBL, Fa_roM_DBL, Fa_roE_DBL,
--             Fb_raE_DBL, Fb_roM_DBL, Fb_roE_DBL  : std_ulogic;
--    variable Fa_raE_hSGL, Fa_roM_hSGL, Fa_roE_hSGL,
--             Fb_raE_hSGL, Fb_roM_hSGL, Fb_roE_hSGL  : std_ulogic;
--    variable Fa_raE_lSGL, Fa_roM_lSGL, Fa_roE_lSGL,
--             Fb_raE_lSGL, Fb_roM_lSGL, Fb_roE_lSGL  : std_ulogic;
    variable Class_a_DBL, Class_a_hSGL, Class_a_lSGL : FP_CLASS;
    variable Class_b_DBL, Class_b_hSGL, Class_b_lSGL : FP_CLASS;

    variable s_dbl, s_sgl: std_ulogic;
    variable r_dbl, ri_dbl, DE_dbl: std_ulogic_vector(DBL_E_SIZE-1 downto 0);
    variable r_sgl, ri_sgl, DE_sgl: std_ulogic_vector(SGL_E_SIZE-1 downto 0);


    variable OpSize : std_ulogic_vector(1 downto 0);

  begin
    if (Rst = '1') then
--      r2_Enable <= '0';
    else
       if (rising_edge(Clk)) then
        if (r1_Enable = '1') then


          for i in 0 to BLOCK64_NBR-1 loop -- for each 64 bit chunk


            Fa(CHUNK_SIZE-1 downto 0) := r1_Fa((i+1)*CHUNK_SIZE-1 downto
                                               i*CHUNK_SIZE);
            Fb(CHUNK_SIZE-1 downto 0) := r1_Fb((i+1)*CHUNK_SIZE-1 downto
                                               i*CHUNK_SIZE);

            OpSize := r1_SuperMode(Super_Mode_OpSize0 downto Super_Mode_OpSize0);
            -- note : r1_SuperMode(Super_Mode_OpSize1) = 0 assumed

            Substract := r1_SuperMode(Super_Mode_Sub);

            -- d=0
            Ea0 := (others => '0');
            Ea1 := (others => '0');
            Eb0 := (others => '0');
            Eb1 := (others => '1'); -- fill with '1' for carry propagation

            Ea1(SGL_E_SIZE-1 downto 0) := Fa(SGL_E_END + SGL_SIZE downto
                                  SGL_E_START + SGL_SIZE); -- d=1
            Eb1(SGL_E_SIZE-1 downto 0) := not (Fb(SGL_E_END + SGL_SIZE downto
                                               SGL_E_START + SGL_SIZE)); -- d=2
```

```
395             if (OpSize = "00") then -- SIMD mode = SINGLE
                  -- First part Exponent Substraction
                  -- (d=1)
                  Ea0(SGL_E_SIZE-1 downto 0) := Fa(SGL_E_END downto
                                                   SGL_E_START); -- d=1
400               Eb0(SGL_E_SIZE-1 downto 0) := not(Fb(SGL_E_END downto
                                                   SGL_E_START)); -- d=2

                else -- SIMD mode = double
                  -- (d=0)
405               Ea0(SGL_E_SIZE-1 downto 0) := (others => 'X'); -- d=0
                  Eb0(SGL_E_SIZE-1 downto 0) := (others => 'X'); -- d=0

                  -- (d=1)
                  Ea1(DBL_E_SIZE-SGL_E_SIZE-1 downto 0) :=
410                               Fa(DBL_E_END-SGL_E_SIZE downto
                                    DBL_E_START); -- d=1
                  Eb1(DBL_E_SIZE-SGL_E_SIZE-1 downto 0) :=
                                 not (Fb(DBL_E_END-SGL_E_SIZE downto
                                    DBL_E_START)); -- d=2
415
                end if;
                -- (d=2)

                -- First part 11-bits adders ('DOUBLE or highest single' datapath)
420             -- (d=2)
                -- computing Ea-Eb
                fasu_ExpAdder_PartOne(Ea1, Eb1, pm1, sv1, tv1, y1, z1); -- d=4
                -- (d=5)

425

                -- First part 8-bits adders ('lowest single' datapath)
                -- (d=3)
                -- computing Ea-Eb
                fasu_ExpAdder_PartOne(Ea0, Eb0, pm0, sv0, tv0, y0, z0); -- d=4
430             -- (d=6)


                -- Special value detection, part one: DOUBLE datapath
                -- (!!ONLY DOUBLE, NOT 'DOUBLE or highest SINGLE datapath !!)
435             --(d=1)
--                Fa_roE_DBL := reduce_or (Fa(DBL_E_End downto DBL_E_Start)); -- d=3
--                Fa_roM_DBL := reduce_or (Fa(DBL_M_End downto DBL_M_Start)); -- d=3
--                Fa_raE_DBL := reduce_and(Fa(DBL_E_End downto DBL_E_Start)); -- d=3
--                Fb_roE_DBL := reduce_or (Fb(DBL_E_End downto DBL_E_Start)); -- d=3
440 --                Fb_roM_DBL := reduce_or (Fb(DBL_M_End downto DBL_M_Start)); -- d=3
--                Fb_raE_DBL := reduce_and(Fb(DBL_E_End downto DBL_E_Start)); -- d=3
--                -- (d=4)
--
--                -- Special value detection, part one: highest SINGLE datapath
445 --                --(d=1)
--                Fa_roE_hSGL := reduce_or (Fa(32+SGL_E_End downto 32+SGL_E_Start)); -- d=3
--                Fa_roM_hSGL := reduce_or (Fa(32+SGL_M_End downto 32+SGL_M_Start)); -- d=3
--                Fa_raE_hSGL := reduce_and(Fa(32+SGL_E_End downto 32+SGL_E_Start)); -- d=3
--                Fb_roE_hSGL := reduce_or (Fb(32+SGL_E_End downto 32+SGL_E_Start)); -- d=3
450 --                Fb_roM_hSGL := reduce_or (Fb(32+SGL_M_End downto 32+SGL_M_Start)); -- d=3
--                Fb_raE_hSGL := reduce_and(Fb(32+SGL_E_End downto 32+SGL_E_Start)); -- d=3
```

```
--          -- (d=4)
--
--          -- Special value detection, part one: lowest SINGLE datapath
--          -- (d=1)
--          Fa_roE_lSGL := reduce_or (Fa(SGL_E_End downto SGL_E_Start)); -- d=3
--          Fa_roM_lSGL := reduce_or (Fa(SGL_M_End downto SGL_M_Start)); -- d=3
--          Fa_raE_lSGL := reduce_and(Fa(SGL_E_End downto SGL_E_Start)); -- d=3
--          Fb_roE_lSGL := reduce_or (Fb(SGL_E_End downto SGL_E_Start)); -- d=3
--          Fb_roM_lSGL := reduce_or (Fb(SGL_M_End downto SGL_M_Start)); -- d=3
--          Fb_raE_lSGL := reduce_and(Fb(SGL_E_End downto SGL_E_Start)); -- d=3
--          -- (d=4)

          Class_a_DBL  := classify(Fa(63 downto  0));
          Class_b_DBL  := classify(Fb(63 downto  0));

          Class_a_hSGL := classify(Fa(63 downto 32));
          Class_b_hSGL := classify(Fb(63 downto 32));

          Class_a_lSGL := classify(Fa(31 downto  0));
          Class_b_lSGL := classify(Fb(31 downto  0));


          -- Effective Subtraction detection for 'DOUBLE and highest SINGLE'
          -- datapath
          -- (d=1)
          EffSubDBL := Fa(DBL_S_POS) xor Fb(DBL_S_POS) xor Substract; -- d=2
          -- rem: DBL_S_POS = SGL_S_POS + SGL_SIZE = 63
          -- (d=3)

          -- Effective Subtraction detection for 'lowest SINGLE' datapath
          -- (d=1)
          EffSubSGL := Fa(SGL_S_POS) xor Fb(SGL_S_POS) xor Substract; -- d=2
          -- (d=3)


          -- (((((((( END OF STAGE 1 ))))))))


          -- (d=6)

          -- TODO: Second part 11-bit Compound Adder, d=2
          -- outputs: sign : s_dbl
          --          result : r_dbl
          --          incremented result : ri_dbl
          -- (d=2)
          ri_dbl := not ri_dbl; -- d=1
          -- (d=9)
          if (s_dbl = '1') then -- MUX (d=1)
            DE_dbl := ri_dbl;
          else
            DE_dbl := r_dbl;
          end if;
          --(d=10)

          -- TODO: Second part 8-bit Compound Adder, d=1
          -- outputs: sign : s_sgl
          --          result : r_sgl
```

```
                --              incremented result : ri_sgl
510             -- (d=7)
                ri_sgl := not ri_sgl; -- d=1
                -- (d=8)
                if (s_sgl = '1') then -- MUX (d=1)
                  DE_sgl := ri_sgl;
515             else
                  DE_sgl := r_sgl;
                end if;
                --(d=9)

520             -- OPERAND SWAPPING


                end loop;

525           -- enable stage ?
--              r?_Enable <= '1';
            else
--              r?_Enable <= '0';
            end if;
530       end if;
      end if;
    end process;

end Behav_1;
```

3 Code de test de l'unité Fasu

Voici le code source du jeu de test de l'unité d'addition en virgule flottante

```
-------------------------------BEGIN_VHDL_LICENSE-----------------------------------
-- fasu-test.vhdl - Testbench for Floating Point Add/Sub Execution Unit
-- Copyright (C) 2003-2004 -- SEMET Gaetan <gaetan@xeberon.net>
-- ISIMA Engineering School -- http://www.isima.fr
-- Copyright (C) 2000, 2001, 2003 Michael Riepe <michael@stud.uni-hannover.de>

-- This program is free software; you can redistribute it and/or modify
-- it under the terms of the GNU General Public License as published by
-- the Free Software Foundation; either version 2 of the License, or
-- (at your option) any later version.
--
-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.

-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
-------------------------------END_VHDL_LICENSE-------------------------------------

--pragma synthesis_off

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

use work.FCPU_config.all;
use work.FCPU_FPU_Config.all;
use work.FCPU_FPU_Random.all;

entity fasu_test is
  generic (WIDTH : natural := 64);
end fasu_test;

architecture Arch_1 of fasu_test is

  component EU_FASU is
    generic (
      WIDTH : positive := 64
    );

    port(
    -- inputs :
      Din_0     : in std_ulogic_vector(WIDTH-1 downto 0);
      Din_1     : in std_ulogic_vector(WIDTH-1 downto 0);
      Substract : in std_ulogic;
      ieee_flag: in std_ulogic; -- 0 : ieee flag not set, 1: ieee flag set
      simd_flag: in std_ulogic; -- 0 : not SIMD operation, 1: SIMD operation
      SIMD      : in std_ulogic_vector(1 downto 0);
```

```
      RoundMode: in std_ulogic_vector(1 downto 0);
                 -- 00 : nearest, 01: zero, 10: -inf, 11: +inf
      Clk      : in std_ulogic;
      Rst      : in std_ulogic;
      En       : in std_ulogic;

    -- outputs:
      -- Result (32+32/64)
      Dout_0   : out std_ulogic_vector(WIDTH-1 downto 0);
      FExout   : out std_ulogic_vector(8 downto 0);
      excep_trigged : out std_ulogic
    );
  end component;

  -- constants used in tests
  constant SGL_SIZE            : natural := 32;
  constant DBL_SIZE            : natural := 64;

  -- numbers:
  constant const32_0d3568      :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "00111110101101101010111001111101"; --  0.3568
  constant const32_0d3569      :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "00111110101101101011101110011001"; --  0.3569
  constant const32_0d35692215 :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "00111110101101101011111010000000";
                                                                  -- ~ 0.35692215
  constant const32_0d35692212 :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "00111110101101101011111001111111";
                                                                  -- ~ 0.35692212
  constant const32_0           :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "00000000000000000000000000000000"; --  0
  constant const32_m_0         :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "10000000000000000000000000000000"; -- -0
  constant const32_1           :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "00111111100000000000000000000000"; --  0
  constant const32_m_1         :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "10111111100000000000000000000000"; -- -0
  constant const32_PI          :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "01000000010010010000111111010000"; -- 3.14159
  constant const32_6           :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "01000000110000000000000000000000"; --  6
  constant const32_m6          :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "11000000110000000000000000000000"; -- -6
  constant const32_9           :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "01000001000100000000000000000000"; --  9
  constant const32_m9          :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "11000001000100000000000000000000"; --  9
  constant const32_9d58448     :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "01000001000110010101101000001000"; -- 9.58448
  constant const32_15          :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "01000001011100000000000000000000"; --  15
  constant const32_m15         :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "11000001011100000000000000000000"; -- -15
  constant const32_15d58448    :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "01000001011110010101101000001000";-- 15.58448
  constant const32_18d7868     :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "01000001100101100100101101011110";-- 18.7868
```

```
  constant const32_21          :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "01000001101010000000000000000000"; --  21
  constant const32_m21         :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "11000001101010000000000000000000"; --  -21
  constant const32_28d37128    :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "01000001111000101111100001100010"; --28.37128
  constant const32_m994        :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "11000100011110001000000000000000"; --  -994
  constant const32_1000        :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "01000100011110100000000000000000"; --  1000
  constant const32_m1000       :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "11000100011110100000000000000000"; --  -1000
  constant const32_1006        :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "01000100011110111000000000000000"; --  1006
  constant const32_1e21        :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "01100010010110001101011100100111"; --  10^21
  constant const32_1em21       :std_ulogic_vector(SGL_SIZE-1 downto 0)
                               := "00011100100101110001110110100000"; --  10^-21


  constant CYCLEDURATION : time := 10 ns;
  constant CYCLEDURATION_DIV2 : time := 5 ns;

  signal Din_0, Din_1 : F_VECTOR;
  signal Substract, Rst, En : std_ulogic;
  signal Clk, Run : std_ulogic := '0';
  signal Flags : std_ulogic_vector(23 downto 18);
  signal SIMD : std_ulogic_vector(1 downto 0);
  signal RoundMode : std_ulogic_vector(1 downto 0);
  signal ieee_flag : std_ulogic;
  signal simd_flag : std_ulogic;
  signal Dout_0 : std_ulogic_vector(WIDTH-1 downto 0);
  signal FExout : std_ulogic_vector(8 downto 0);

  -- CONVERSION PROCEDURES


  function real2single(A: real) return std_ulogic_vector is
    variable sign: std_ulogic;
    variable exp: std_ulogic_vector(SGL_E_SIZE-1 downto 0);
    variable man: std_ulogic_vector(SGL_M_SIZE+1-1 downto 0);
    variable aa: real;
    variable ee: real;
    variable eee: integer;
    variable bb: real;
    variable cc: real;
    variable lout : line;
  begin

    -- Sign
    if (A > 0.0) then
      sign := '0';
    else
      sign := '1';
    end if;

    aa := abs(A);
```

```
    if (aa = 0.0) then
      return const32_zero;
    end if;

    if (aa > 1.0) then
      ee := 0.0;
      bb := aa;
      while (bb > 1.0) loop
        bb := bb / 2.0;
        ee := ee + 1.0;
      end loop;
      eee := integer(ee + 126.0);
      exp := To_StdULogicVector(CONV_STD_LOGIC_VECTOR(eee, SGL_E_SIZE));
    else
      cc := aa;
      ee := 0.0;
      while (cc < 1.0) loop
        cc := cc * 2.0;
        ee := ee + 1.0;
      end loop;
      eee := integer(-ee + 127.0);
      exp := To_StdULogicVector(CONV_STD_LOGIC_VECTOR(eee, SGL_E_SIZE ) );
    end if;

    if (eee > 126) then
      for i in 0 to eee-126-1 loop
          aa := aa / 2.0;
        end loop;
     else
      for i in 0 to 126-eee-1 loop
          aa := aa * 2.0;
        end loop;
    end if;

    for i in 0 to SGL_M_SIZE+1-1 loop
      aa := aa * 2.0;
      if (aa > 1.0) then
        man(SGL_M_SIZE - i) := '1';
        aa := aa - 1.0;
      else
        man(SGL_M_SIZE -   i) := '0';
      end if;
    end loop;
    -- Rounding to nearest
    -- TODO : check if correct, add rounding to +/-infinity, to 0
    if (aa > 0.5) then
      man := fpu_incr(man);
    end if;
    return (sign & exp & man(SGL_M_SIZE-1 downto 0));
  end;

--TODO : for double and single2real/double2real;

  procedure print_single_vector (lbl : string;
              y: real;
              des : string := " := ") is
```

```
225     variable yy : std_ulogic_vector(SGL_SIZE-1 downto 0);
        variable lout : line;
    begin
      yy := real2single(y);
      write(lout, lbl & des);
230     write(lout, y);
      write(lout, " (");
      write(lout, yy);
      write(lout, ")");
      writeline(output, lout);
235 end print_single_vector;

    procedure print_double_vector (lbl : string;
                x : std_ulogic_vector;
                sf: string;
240             des : string := " := ") is
      variable lout : line;
    begin
      write(lout, lbl & des);
      write(lout, x);
245   write(lout," (");
      write(lout,sf);
      write(lout,")");
      writeline(output, lout);
    end print_double_vector;
250

    -- REPORT PROCEDURES

    procedure print_str (s : string) is
255   variable lout : line;
    begin
      write(lout, s);
      writeline(output, lout);
    end print_str;
260

    procedure print_stdval (lbl : string;
                x : std_ulogic;
                des : string := " := ") is
      variable lout : line;
265 begin
      write(lout, lbl & des); write(lout, x); writeline(output, lout);
    end print_stdval;

    procedure print_vector (lbl : string;
270             x : std_ulogic_vector;
                des : string := " := ") is
      variable lout : line;
    begin
      write(lout, lbl & des); write(lout, x); writeline(output, lout);
275 end print_vector;

    procedure do_error (lbl : string;
              a_x, a_y : std_ulogic_vector) is
    begin
280   print_str("WHOA THERE!!!");
      print_vector(lbl, a_x);
```

```
    print_vector(lbl, a_y, " /= ");
  end do_error;

  procedure check_vector (lbl : string;
                          x : std_ulogic_vector;
                          y : std_ulogic_vector) is
    variable lout : line;
    constant chk: string := "Check successfull: ";
  begin
    if x /= y then
      do_error(lbl, x, y);
    else
      write(lout, chk & lbl);
      write(lout, " := ");
      write(lout, x);
      writeline(output, lout);
    end if;
  end;


  procedure check_single (lbl : string;
                          x : std_ulogic_vector;
                          y : real) is
    variable lout : line;
    variable yy : std_ulogic_vector(x'length-1 downto 0);
    constant chk: string := "Check successfull: ";
  begin
    yy := real2single(y);
    if x /= yy then
      do_error(lbl, x, yy);
    else
      write(lout, chk & lbl);
      write(lout, " := ");
      write(lout, x);
      write(lout, " (");
      write(lout, y);
      write(lout, ")");
      writeline(output, lout);
    end if;
  end;

  procedure print_signals is
  begin
    print_str    ("** SIGNALS REPORT **");
    print_str    ("Inputs:");
    print_stdval("En        ", En);
    print_stdval("Rst       ", Rst);
    print_vector("Din_0     ", Din_0);
    print_vector("Din_1     ", Din_1);
    print_stdval("Substract", Substract);
    print_vector("Flags     ", Flags);
    print_vector("SIMD      ", SIMD);
    print_str    ("Outputs:");
    print_vector("Dout_0    ", Dout_0);
    print_vector("FExout    ", FExout);
    print_str    ("** END OF SIGNALS REPORT **");
  end;
```

```
340
    -- estimated delay
    --    2 < L <=  4   : d=4/t=5
    --    5 < L <=  8   : d=5/t=6
    --    9 < L <= 16   : d=6/t=7
345 --   17 < L <= 32   : d=7/t=8
    --   33 < L <= 64   : d=8/t=9
    function compare_vector(a, b : std_ulogic_vector) return std_ulogic is
      constant L : natural := a'length;
      variable aa : std_ulogic_vector(L-1 downto 0);
350   variable bb : std_ulogic_vector(L-1 downto 0);
      variable pp, vv: std_ulogic_vector(L-1 downto 0);
      variable p, v, swap : std_ulogic;
      variable step, level, left : natural;
      variable lout: line;
355 begin

      aa := a;
      bb := b;

360   write(lout, "a=");
      write(lout, aa);
      write(lout, "  b=");
      write(lout, bb);
      writeline(output, lout);
365
      -- (d=0/t=0)
      for i in  L-1 downto 0 loop
        pp(i) := bb(i);
        vv(i) := aa(i) xor bb(i);
370   end loop;

      -- (d=1/t=2)

--    for level in 1 to 15 loop
375 --      step := 2**level;
--      left := L/step;
--      for i in 0 to left-1 loop
--        if (vv(2*i+1)='1') then
--          pp(i) := pp(2*i+1);
380 --        else
--          pp(i) := pp(2*i);
--        end if;                  -- d=1/t=1
--        vv(i) := vv(2*i+1) or vv(2*i); -- d=1/t=1
--      end loop;
385 --      exit when step >= L;
--      -- cost for each loop : d=1/t=1
--    end loop;

      for level in 1 to 15 loop
390     step := 4**level;
        left := L / step;
        if (left = 0) then
          if vv(1) = '1' then
            pp(0) := pp(1);
395       else
```

```
          pp(0) := pp(0);
        end if;
        vv(0) := vv(1) or vv(0);
      else
        for i in 0 to left-1 loop
          if vv(4*i+3) = '1' then
            pp(i) := pp(4*i+3);
          elsif vv(4*i+2) = '1' then
            pp(i) := pp(4*i+2);
          elsif vv(4*i+1) = '1' then
            pp(i) := pp(4*i+1);
          else
            pp(i) := pp(4*i+0);
          end if;
          vv(i) := vv(4*i+3) or vv(4*i+2)
                or vv(4*i+1) or vv(4*i+0);
        end loop;
      end if;
      exit when step >= L;
    end loop;
    swap := pp(0) and vv(0);      -- d=1/t=1

    print_stdval("swap?", swap);
    return swap;
  end;


-- TEST PROCESSES AND COMPONENTS
begin

  -- COMPONENT TO TEST

  testunit32 : EU_FASU
    generic map ( WIDTH => 64)
    port map (Din_0 => Din_0,
              Din_1 => Din_1,
              ieee_flag => ieee_flag,
              simd_flag => simd_flag,
              Substract => Substract,
              RoundMode => RoundMode,
              SIMD => SIMD,
              Clk => Clk,
              Rst => Rst,
              En => En,
              Dout_0 => Dout_0,
              FExout => FExout);


  -- Clock generator process
  ClkProcess: process(clk, Run)
  begin
    if (Run = '1') then
      Clk <= not clk after CYCLEDURATION_DIV2;
    end if;
  end process;

  -- driver process
```

```
  testproc : process
  variable lout : line;
455 begin

--  verification of conversion functions:
    print_str("*** Preliminary test ***");
    Run <= compare_vector("1000","1001");
460 Run <= compare_vector("1001","0100");
    Run <= compare_vector("1001","1100");
    Run <= compare_vector("0011","0011");
    Run <= compare_vector("1110","1111");
    Run <= compare_vector("1000","1000");
465 Run <= compare_vector("1001","1000");
    Run <= compare_vector("0010","1000");
    Run <= compare_vector("10010000", "01000000");
    Run <= compare_vector("00100000", "00110000");
    Run <= compare_vector("11100001", "11100000");
470 Run <= compare_vector("11100000", "11100001");
    Run <= compare_vector("00100001", "10000001");
    Run <= compare_vector("0010","1000");
    Run <= compare_vector("0000000000100001", "1000000000000001");
    Run <= compare_vector("0000000000100001", "0000000000100010");
475 Run <= compare_vector("0000000000100001", "0000000000100000");
    Run <= compare_vector("1000000000100001", "1000000000000000");
    Run <= compare_vector("0010","0011");
    Run <= compare_vector("00100000","00110000");
    Run <= compare_vector("01000000000001000000000", "00000000000010000000000");
480 Run <= compare_vector("01000000000010000000000", "00000000000100000000000");
    Run <= compare_vector("01001111110001001111110", "00000000000100000000000");

    print_str("*** Conversion routine checks ***");
    check_vector("Single float:    0.0       ",
485                 const32_0, real2single(0.0));
    check_vector("Single float:    0.3568    ",
                 const32_0d3568, real2single(0.3568));
    check_vector("Single float:    0.3569    ",
                 const32_0d3569, real2single(0.3569));
490 check_vector("Single float:    0.35692215",
                 const32_0d35692215, real2single(0.35692215));
    check_vector("Single float:    0.35692212",
                 const32_0d35692212, real2single(0.35692212));
    check_vector("Single float:    1.0       ",
495                 const32_1, real2single(1.0));
    check_vector("Single float:    3.14159   ",
                 const32_PI, real2single(3.14159));
    check_vector("Single float:    6.0       ",
                 const32_6, real2single(6.0));
500 check_vector("Single float:   -9.0       ",
                 const32_m9, real2single(-9.0));
    check_vector("Single float:    9.58448   ",
                 const32_9d58448, real2single(9.58448));
    check_vector("Single float:   15.58449   ",
505                 const32_15d58448, real2single(15.58448));
    check_vector("Single float:   18.7868    ",
                 const32_18d7868, real2single(18.7868));
    check_vector("Single float:   28.37128   ",
                 const32_28d37128, real2single(28.37128));
```

```
510     check_vector("Single float: 1000.0       ",
                     const32_1000, real2single(1000.0));
        check_vector("Single float: 1006.0       ",
                     const32_1006, real2single(1006.0));
        check_vector("Single float: 1e 21        ",
515                  const32_1e21, real2single(1.0e21));
        check_vector("Single float: 1e-21        ",
                     const32_1em21, real2single(1.0e-21));



520



        print_str("*** testing FPU add/sub unit (fadd/fsub instruction) ***");
        print_str("FASU size : 64 bit (2x32-bit SIMD)");
525
        write(lout, "Clock period: ");
        write(lout, CYCLEDURATION);
        writeline(output, lout);

530     Din_0 <= (others => '0');
        Din_1 <= (others => '0');
        Substract <= '0';
        Flags <= (others => '0');
        SIMD <= (others => '0');
535     En <= '0';
        Rst <= '0';
        ieee_flag <= '0';
        simd_flag <= '0';
        RoundMode <= Mode_RoundMode_Nearest;
540     Run <= '1';


        print_str("Reseting unit...");
        Rst <= '1';
545
        wait for CYCLEDURATION/2 + CYCLEDURATION - CYCLEDURATION/5;

        Rst <= '0';
        print_str("End of reset.");
550
        wait for 15 ns;

        print_str("Default values:");
        print_signals;
555     print_str("------FLOAT ADDITION (SIMD=00) -----");
        print_str("First calculation: single float");
        print_single_vector("Din_0", 6.0);
        print_single_vector("Din_1", 1000.0);
        Din_0(SGL_SIZE-1 downto 0)      <= const32_6(SGL_SIZE-1 downto 0);
560     Din_0(WIDTH-1 downto SGL_SIZE) <= (others => '0');
        Din_1(SGL_SIZE-1 downto 0)      <= real2single(1000.0);
        Din_1(WIDTH-1 downto SGL_SIZE) <= (others => '0');
        En <= '1';
        print_str("Launching ONE calculation (over 6 cycles)...");
565
        wait for 6*CYCLEDURATION;
```

```
    check_single("Dout_0", Dout_0(SGL_SIZE-1 downto 0), 21.0);

    print_str("Using SIMD: +inf on Din_0(high) + -15 on Din_1(hight bits)...");
    Din_0(WIDTH-1 downto SGL_SIZE) <=  const32_infty;
    Din_1(WIDTH-1 downto SGL_SIZE) <=  const32_15;
    print_str("Launching ONE calculation (over 6 cycles)...");

    wait for 6*CYCLEDURATION;

    check_vector("Dout_0", Dout_0(WIDTH-1 downto SGL_SIZE),
                 const32_infty);

    En <= '0';

    wait for 2*CYCLEDURATION;


    En <= '1';

    print_str("Launching cascading calculations (6 calculations)");
    print_str("1st cycle");
    print_single_vector("Din_0",  6.0);
    print_single_vector("Din_1", -15.0);
    Din_0(SGL_SIZE-1 downto 0) <=  const32_6(SGL_SIZE-1 downto 0);
    Din_0(WIDTH-1 downto SGL_SIZE) <= (others => '0');
    Din_1(SGL_SIZE-1 downto 0) <=  const32_m15(SGL_SIZE-1 downto 0);
    Din_1(WIDTH-1 downto SGL_SIZE) <= (others => '0');

    wait for CYCLEDURATION;

    print_str("2nd cycle");
    print_single_vector("Din_0", -6.0);
    print_single_vector("Din_1", -15.0);
    Din_0(SGL_SIZE-1 downto 0) <=  const32_m6(SGL_SIZE-1 downto 0);
    Din_0(WIDTH-1 downto SGL_SIZE) <= (others => '0');
    Din_1(SGL_SIZE-1 downto 0) <=  const32_m15(SGL_SIZE-1 downto 0);
    Din_1(WIDTH-1 downto SGL_SIZE) <= (others => '0');

    wait for CYCLEDURATION;

    print_str("3rd cycle");
    print_single_vector("Din_0", -6.0);
    print_single_vector("Din_1", 15.58448);
    Din_0(SGL_SIZE-1 downto 0) <=  const32_m6(SGL_SIZE-1 downto 0);
    Din_0(WIDTH-1 downto SGL_SIZE) <= (others => '0');
    Din_1(SGL_SIZE-1 downto 0) <=  const32_15d58448(SGL_SIZE-1 downto 0);
    Din_1(WIDTH-1 downto SGL_SIZE) <= (others => '0');

    wait for CYCLEDURATION;

    print_str("4th cycle");
    print_single_vector("Din_0", 6.0);
    print_single_vector("Din_1", 1000.0);
    Din_0(SGL_SIZE-1 downto 0) <=  const32_6(SGL_SIZE-1 downto 0);
    Din_0(WIDTH-1 downto SGL_SIZE) <= (others => '0');
    Din_1(SGL_SIZE-1 downto 0) <=  const32_1000(SGL_SIZE-1 downto 0);
```

```
    Din_1(WIDTH-1 downto SGL_SIZE) <= (others => '0');

    wait for CYCLEDURATION;

    print_str("5th cycle");
    print_single_vector("Din_0", 6.0);
    print_single_vector("Din_1", -1000.0);
    Din_0(SGL_SIZE-1 downto 0) <=  const32_6(SGL_SIZE-1 downto 0);
    Din_0(WIDTH-1 downto SGL_SIZE) <= (others => '0');
    Din_1(SGL_SIZE-1 downto 0) <=  const32_m1000(SGL_SIZE-1 downto 0);
    Din_1(WIDTH-1 downto SGL_SIZE) <= (others => '0');

    wait for CYCLEDURATION;

    print_str("6th cycle");
    print_single_vector("Din_0", 1000.0);
    print_single_vector("Din_1", 1.0e21);
    Din_0(SGL_SIZE-1 downto 0) <=  const32_1000(SGL_SIZE-1 downto 0);
    Din_0(WIDTH-1 downto SGL_SIZE) <= (others => '0');
    Din_1(SGL_SIZE-1 downto 0) <=  const32_1e21(SGL_SIZE-1 downto 0);
    Din_1(WIDTH-1 downto SGL_SIZE) <= (others => '0');

    print_str("Result first calculation (6.0f + -15.0f)");
    check_single("Dout_0", Dout_0(SGL_SIZE-1 downto 0), -9.0);
    wait for CYCLEDURATION;

    print_str("7th cycle");
    print_single_vector("Din_0", 9.58448);
    print_single_vector("Din_1", 18.7848);
    Din_0(SGL_SIZE-1 downto 0) <=  const32_9d58448(SGL_SIZE-1 downto 0);
    Din_0(WIDTH-1 downto SGL_SIZE) <= (others => '0');
    Din_1(SGL_SIZE-1 downto 0) <=  const32_18d7868(SGL_SIZE-1 downto 0);
    Din_1(WIDTH-1 downto SGL_SIZE) <= (others => '0');

    print_str("Result second calculation (-6.0f + -15.0f)");
    check_single("Dout_0", Dout_0(SGL_SIZE-1 downto 0), -21.0);

    wait for CYCLEDURATION;

    print_str("8th cycle");
    print_str("Result third calculation (-6.0f + 15.58448f)");
    check_single("Dout_0", Dout_0(SGL_SIZE-1 downto 0), 9.58448);

    wait for CYCLEDURATION;

    print_str("9th cycle");
    print_str("Result fourth calculation (6.0f + 1000.0f)");
    check_single("Dout_0", Dout_0(SGL_SIZE-1 downto 0), 1006.0);

    wait for CYCLEDURATION;

    print_str("10th cycle");
    print_str("Result fifth calculation (6.0f + -1000.0f)");
    check_single("Dout_0", Dout_0(SGL_SIZE-1 downto 0), -994.0);

    wait for CYCLEDURATION;
```

```
    print_str("11th cycle");
    print_str("Result sixth calculation (1000.0f + 1e21f)");
    check_single("Dout_0", Dout_0(SGL_SIZE-1 downto 0),1.0e21);

    wait for CYCLEDURATION;

    print_str("12th cycle");
    print_str("Result seventh calculation (9.58448f + 18.7868)");
    check_single("Dout_0", Dout_0(SGL_SIZE-1 downto 0), 28.37128);

    wait for 2*CYCLEDURATION;
    print_str("Start new tests : 32 bit SIMD mode");
    print_str("              Din_0     Din_1");
    print_str("high SIMD : +infty + 0");
    print_str("low  SIMD : -infty + +infty");

    Din_0(WIDTH-1 downto SGL_SIZE) <= const32_infty;
    Din_1(WIDTH-1 downto SGL_SIZE) <= const32_zero;
    Din_0(SGL_SIZE-1 downto 0) <=  const32_m_infty;
    Din_1(SGL_SIZE-1 downto 0) <=  const32_infty;
    print_double_vector("Din_0",(const32_infty & const32_m_infty),
                        "(+infty, -infty)");
    print_double_vector("Din_1", (const32_zero & const32_infty),
                        "( 0, +infty)");
    Run <= '0';
    wait;
  end process;

end Arch_1 ;

-- simili 2.2 specific:
-- recreate waveform:
-- source "Waveform0-wave.tcl"

-- custom convert program command:
-- convert from hexadecimal binary representation to float:
-- floatrepres.exe -h 4c00000

-- convert from binary representation to float:
-- floatrepres.exe -b 100101100010...

-- convert from float to bin:
-- floatrepres.exe -f 10.5
```

Annexe C

Codes sources VHDL de modules extérieurs

1 Generic Adder (CSAdder) de M. Reipe

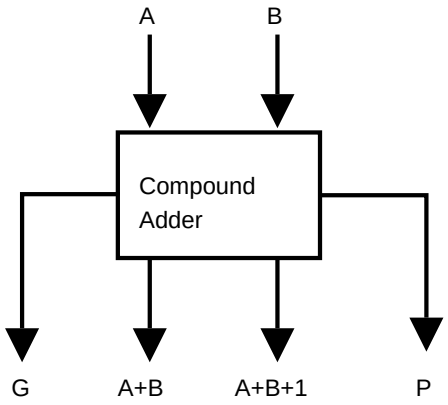


FIG. C.1 – L'additionneur « Compound Adder »

```
-- carry look-ahead
  -- d=1-2
  procedure CLA (Gi, Pi : in std_ulogic_vector;
                 Go, Po : out std_ulogic_vector) is
    constant L : natural := Gi'length;
    constant L4 : natural := (L + 3) / 4;
    variable gg, pp : std_ulogic_vector(4*L4-1 downto 0);
    variable og, op : std_ulogic_vector(L4-1 downto 0);
  begin
--pragma synthesis_off
    assert Gi'length = L;
    assert Pi'length = L;
    assert Go'length = L4;
    assert Po'length = L4;
```

```
--pragma synthesis_on
    gg := (others => '0');
    pp := (others => '1');
    gg(L-1 downto 0) := Gi;
    pp(L-1 downto 0) := Pi;
    for i in L4-1 downto 0 loop
      -- d=2
      og(i) := gg(4*i+3)
        or (pp(4*i+3) and gg(4*i+2))
        or (pp(4*i+3) and pp(4*i+2) and gg(4*i+1))
        or (pp(4*i+3) and pp(4*i+2) and pp(4*i+1)
            and gg(4*i+0));
      -- d=1
      op(i) := pp(4*i+3) and pp(4*i+2) and pp(4*i+1)
               and pp(4*i+0);
    end loop;
    Go := og;
    Po := op;
  end CLA;

  -- carry select vectors
  -- d=2
  procedure CSV (G, P : in std_ulogic_vector;
                 S, T : out std_ulogic_vector) is
    constant L : natural := G'length;
    constant L4 : natural := (L + 3) / 4;
    variable gg, pp, ss, tt : std_ulogic_vector(4*L4-1 downto 0);
  begin
--pragma synthesis_off
    assert G'length = L;
    assert P'length = L;
    assert S'length = L;
    assert T'length = L;
--pragma synthesis_on
    gg := (others => 'U');
    pp := (others => 'U');
    gg(L-1 downto 0) := G;
    pp(L-1 downto 0) := P;
    for i in L4-1 downto 0 loop
      -- d=2
      ss(4*i+0) := '0';
      ss(4*i+1) := gg(4*i+0);
      ss(4*i+2) := gg(4*i+1)
        or (pp(4*i+1) and gg(4*i+0));
      ss(4*i+3) := gg(4*i+2)
        or (pp(4*i+2) and gg(4*i+1))
        or (pp(4*i+2) and pp(4*i+1) and gg(4*i+0));
      -- d=2
      tt(4*i+0) := '1';
      tt(4*i+1) := gg(4*i+0)
        or pp(4*i+0);
      tt(4*i+2) := gg(4*i+1)
        or (pp(4*i+1) and gg(4*i+0))
        or (pp(4*i+1) and pp(4*i+0));
      tt(4*i+3) := gg(4*i+2)
        or (pp(4*i+2) and gg(4*i+1))
        or (pp(4*i+2) and pp(4*i+1) and gg(4*i+0))
```

```
        or (pp(4*i+2) and pp(4*i+1) and pp(4*i+0));
    end loop;
    S := ss(L-1 downto 0);
    T := tt(L-1 downto 0);
  end CSV;


  -- carry-select adder
  -- delay (without pipelining):
  -- L <=    4: d=4  t=6
  -- L <=    8: d=5  t=7
  -- L <=   16: d=6  t=7
  -- L <=   32: d=7  t=8
  -- L <=   64: d=8  t=9
  -- L <=  128: d=9  t=10
  -- L <=  256: d=10 t=11
  -- L <=  512: d=11 t=12
  -- L <= 1024: d=12 t=13
  procedure CSAdd (A, B : in std_ulogic_vector;
                   Y, Z : out std_ulogic_vector;
                   G, P : out std_ulogic) is
    constant L : natural := A'length;
    variable aa, bb : std_ulogic_vector(L-1 downto 0);
    variable ym, zm : std_ulogic_vector(L-1 downto 0);
    variable gm, pm : std_ulogic_vector(L-1 downto 0);
    variable yt, zt : std_ulogic_vector(L-1 downto 0);
    variable gt, pt : std_ulogic_vector(L-1 downto 0);
    variable sv, tv : std_ulogic_vector(L-1 downto 0);
    variable step, left, right : natural;
  begin
--pragma synthesis_off
    assert A'length = L;
    assert B'length = L;
    assert Y'length = L;
    assert Z'length = L;
--pragma synthesis_on

    -- normalize inputs
    aa := A;
    bb := B;

    -- a row of 4-bit adders
    -- d=1 t=2
    gm := aa and bb;
    pm := aa xor bb;
    -- d=3 t=4
    CSV(gm, pm, sv, tv);
    -- d=4 t=6
    ym := pm xor sv;
    zm := pm xor tv;
    -- d=3 t=4
    gt := gm; pt := pm;
    CLA(gt, pt, gm((L-1)/4 downto 0), pm((L-1)/4 downto 0));

    -- carry-select tree
    for level in 1 to 15 loop  -- should be enough...
      step := 4 ** level;
```

```
130         exit when step >= L;
            left := (L - 1) / step;

            -- single tree level
            -- d=5/7/9/11/...
135         -- t=6/8/10/12/...
            CSV(gm(left downto 0), pm(left downto 0),
              sv(left downto 0), tv(left downto 0));
            gt := gm; pt := pm;
            CLA(gt(left downto 0), pt(left downto 0),
140           gm(left/4 downto 0), pm(left/4 downto 0));

            -- level mux
            -- d=6/8/10/12/...
            -- t=7/9/11/13/...
145         yt := ym; zt := zm;
            for i in L/step-1 downto 0 loop
              if to_X01(sv(i)) = '1' then
                ym(step*(i+1)-1 downto step*i) :=
                  zt(step*(i+1)-1 downto step*i);
150           end if;
              if to_X01(tv(i)) /= '1' then
                zm(step*(i+1)-1 downto step*i) :=
                  yt(step*(i+1)-1 downto step*i);
              end if;
155         end loop;

            -- last (partial) chunk
            if L mod step /= 0 then
              if to_X01(sv(L/step)) = '1' then
160             ym(L-1 downto L - L mod step) :=
                  zt(L-1 downto L - L mod step);
              end if;
              if to_X01(tv(L/step)) /= '1' then
                zm(L-1 downto L - L mod step) :=
165               yt(L-1 downto L - L mod step);
              end if;
            end if;
          end loop;

170       -- outputs
          Y := ym;
          Z := zm;
          G := gm(0);
          P := pm(0);
175     end CSAdd;
```