```
.SUFFIXES : .o .vhdl

.vhdl.o :
        ghdl -a  $<


cordic_tb : cordic_pkg.o cordic_beh.o cordic_tb.o
        ghdl -e cordic_tb


anal : cordic_pkg.o cordic_beh.o cordic_tb.o

elab : cordic_tb

run : cordic_tb
        ghdl -r cordic_tb --vcd=cordic.vcd --stop-time=2000ns

all : anal elab run

wave :
        gtkwave cordic.vcd &

clean :
        \rm -f *.o *~ *# *.cf *.vcd print.*
        \rm -f cordic_tb

print_all :
        more makefile  >  print.cordic_beh
        more *.vhdl    >> print.cordic_beh

tar :
        mkdir src
        cp makefile *.vhdl src
        tar cvf 5.cordic_beh.tar src
        \rm -fr src
::::::::::::::
cordic_beh.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    behavioral model of cordic
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.03.22
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input: clk, rst,
--           load, ready,
--           xi, yi, zi
--
--    Output: xo, yo, zo
--------------------------------------------------------------------------------
```

```
library STD;
use STD.textio.all;

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

use WORK.cordic_pkg.all;


entity cordic is

  generic (
    vflag    : boolean := false;
    n        : integer := 10);

  port (
    clk, rst    : in  std_logic;
    load        : in  std_logic;
    ready       : out std_logic := '0' ;
    xi, yi, zi  : in  std_logic_vector (31 downto 0) := X"0000_0000";
    xo, yo, zo  : out std_logic_vector (31 downto 0) := X"0000_0000");

end cordic;


architecture beh of cordic is

   constant angle_length : integer := 60;
   constant kprod_length : integer := 33;

   type real_array is array (natural range <>) of real;

   constant angles : real_array :=
                     (  7.853981633974830962E-01,  -- pi/4 rad
                        4.636476090008061162E-01,
                        2.449786631268641541E-01,
                        1.243549945467614350E-01,
                        6.241880999595734847E-02,
                        3.123983343026827625E-02,
                        1.562372862047683080E-02,
                        7.812341060101111296E-03,
                        3.906230131966971827E-03,
                        1.953122516478818685E-03,
                        9.765621895593194304E-04,
                        4.882812111948982754E-04,
                        2.441406201493617640E-04,
                        1.220703118936702042E-04,
                        6.103515617420877502E-05,
                        3.051757811552609686E-05,
                        1.525878906131576210E-05,
                        7.629394531101970263E-06,
                        3.814697265606496283E-06,
                        1.907348632810187035E-06,
                        9.536743164059608794E-07,
                        4.768371582030888599E-07,
                        2.384185791015579825E-07,
                        1.192092895507806853E-07,
                        5.960464477539055441E-08,
                        2.980232238769530368E-08,
                        1.490116119384765515E-08,
                        7.450580596923827987E-09,
                        3.725290298461914045E-09,
                        1.862645149230957029E-09,
                        9.313225746154785154E-10,
                        4.656612873077392578E-10,
```

```
                      2.3283064365386962890E-10,
                      1.1641532182693481445E-10,
                      5.8207660913467407226E-11,
                      2.9103830456733703613E-11,
                      1.4551915228366851807E-11,
                      7.2759576141834259033E-12,
                      3.6379788070917129517E-12,
                      1.8189894035458564758E-12,
                      9.0949470177292823792E-13,
                      4.5474735088646411896E-13,
                      2.2737367544323205948E-13,
                      1.1368683772161602974E-13,
                      5.6843418860808014870E-14,
                      2.8421709430404007435E-14,
                      1.4210854715202003717E-14,
                      7.1054273576010018587E-15,
                      3.5527136788005009294E-15,
                      1.7763568394002504647E-15,
                      8.8817841970012523234E-16,
                      4.4408920985006261617E-16,
                      2.2204460492503130808E-16,
                      1.1102230246251565404E-16,
                      5.5511151231257827021E-17,
                      2.7755575615628913511E-17,
                      1.3877787807814456755E-17,
                      6.9388939039072283776E-18,
                      3.4694469519536141888E-18,
                      1.7347234759768070944E-18  );

 constant kprod : real_array :=
                    ( 0.70710678118654752440,
                      0.63245553203367586640,
                      0.61357199107789634961,
                      0.60883391251775242102,
                      0.60764825625616820093,
                      0.60735177014129595905,
                      0.60727764409352599905,
                      0.60725911229889273006,
                      0.60725447933256232972,
                      0.60725332108987516334,
                      0.60725303152913433540,
                      0.60725295913894481363,
                      0.60725294104139716351,
                      0.60725293651701023413,
                      0.60725293538591350073,
                      0.60725293510313931731,
                      0.60725293503244577146,
                      0.60725293501477238499,
                      0.60725293501035403837,
                      0.60725293500924945172,
                      0.60725293500897330506,
                      0.60725293500890426839,
                      0.60725293500888700922,
                      0.60725293500888269443,
                      0.60725293500888161574,
                      0.60725293500888134606,
                      0.60725293500888127864,
                      0.60725293500888126179,
                      0.60725293500888125757,
                      0.60725293500888125652,
                      0.60725293500888125626,
                      0.60725293500888125619,
                      0.60725293500888125617 );


signal xn, yn, zn : std_logic_vector(31 downto 0) := X"0000_0000";
signal angle      : std_logic_vector(31 downto 0) := X"0000_0000";
```

```
begin

 main: process
   variable xt, yt, zt : std_logic_vector(31 downto 0) := x"0000_0000";
   variable rx, ry : real := 0.0;
   variable idx : integer := 0;
 begin  -- process main

   wait until (rst'event and rst='1');

   loop
     while (load /= '1') loop
       wait until (clk'event and clk='1');
     end loop;

     angle <= Conv2fixedPt(angles(0), 32) ;

     xn <= xi;
     yn <= yi;
     zn <= zi;
     wait for 1 ns;

     if (vflag = true) then
       DispReg(xn, yn, zn, 2);
       DispAng(angle);
     end if;

     LFOR: for j in 1 to n loop

       if (zn(31) = '0')  then
         xt := std_logic_vector(signed(xn) - shift_right(signed(yn), j-1));
         yt := std_logic_vector(shift_right(signed(xn), j-1) + signed(yn));
         zt := std_logic_vector(signed(zn) - signed(angle));
       else
         xt := std_logic_vector(signed(xn) + shift_right(signed(yn), j-1));
         yt := std_logic_vector(-shift_right(signed(xn), j-1) + signed(yn));
         zt := std_logic_vector(signed(zn) + signed(angle));
       end if;

       wait until clk='1';

       if (angle_length < j + 1) then
         angle <= std_logic_vector(shift_right(signed(angle), 1));
       else
         angle <= Conv2fixedPt(angles(j), 32) ;
       end if;

       xn <= xt;
       yn <= yt;
       zn <= zt;
       wait for 1 ns;
       if (vflag = true) then
         DispReg(xn, yn, zn, 2);
         DispAng(angle);
       end if;

     end loop LFOR;


     if (0 < n) then
       if n > kprod_length then
         idx := kprod_length -1;
       else
         idx := n -1;
       end if;
```

```
        --rx := Conv2real(xn) * kprod(idx);
        --ry := Conv2real(yn) * kprod(idx);

        --xo <= Conv2fixedPt(rx, 32);
        --yo <= Conv2fixedPt(ry, 32);
        xo <= xn;
        yo <= yn;
        zo <= zn;
        wait for 1 ns;

        ready <= '1', '0' after clk_period;

      end if;

    end loop;

    wait;

  end process main;

  --  XXXXXXX XXXXXX XXXXXX XXXXXX XXXXXXX XXXXXX XXXXX

end beh;
::::::::::::::
cordic_pkg.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    utility package of cordic
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.03.22
--
--  Author:
--
--    Young W. Lim
--
--  Functions:
--  Conv2fixedPt (x : real; n : integer) return std_logic_vector;
--  Conv2real (s : std_logic_vector (31 downto 0) ) return real;
--
--
--------------------------------------------------------------------------------

library STD;
use STD.textio.all;

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


package cordic_pkg is

  function Conv2fixedPt (x : real; n : integer) return std_logic_vector;
  function Conv2real (s : std_logic_vector (31 downto 0) ) return real;
```

```
  procedure DispReg (x, y, z : in std_logic_vector (31 downto 0);
                     flag : in integer );
  procedure DispAng (angle : in std_logic_vector (31 downto 0)) ;

  constant clk_period : time := 20 ns;
  constant half_period : time := clk_period / 2.0;

  constant pi : real := 3.141592653589793;
  constant K : real := 1.646760258121;

end cordic_pkg;



package body cordic_pkg is

  ---------------------------------------------------------------------------
  function Conv2fixedPt (x : real; n : integer) return std_logic_vector is
  ---------------------------------------------------------------------------
    constant shft : std_logic_vector (n-1 downto 0) := X"2000_0000";
    variable s : std_logic_vector (n-1 downto 0) ;
    variable z : real := 0.0;
  ---------------------------------------------------------------------------
  begin
      -- shft = 2^29 = 536870912
      -- bit 31 : msb - sign bit
      -- bit 30,29 : integer part
      -- bit 28 ~ 0 : fractional part
      -- for the value of 0.5
      -- first 4 msb bits [0, 0, 0, 1] --> X"1000_0000"
      --
      -- To obtain binary number representation of x,
      -- where the implicit decimal point between bit 29 and bit 28,
      -- multiply "integer converted shft"
      --
      z := x * real(to_integer(unsigned(shft)));

      s := std_logic_vector(to_signed(integer(z), n));

      return s;

  end Conv2fixedPt;
  ---------------------------------------------------------------------------



  ---------------------------------------------------------------------------
  function Conv2real (s : std_logic_vector (31 downto 0) ) return real is
  ---------------------------------------------------------------------------
    constant shft : std_logic_vector (31 downto 0) := X"2000_0000";
    variable z : real := 0.0;
  ---------------------------------------------------------------------------
  begin
    z := real(to_integer(signed(s))) / real(to_integer(unsigned(shft)));
    return z;
  end Conv2real;
  ---------------------------------------------------------------------------

  ---------------------------------------------------------------------------
  procedure DispReg (x, y, z : in std_logic_vector (31 downto 0);
                     flag : in integer ) is
  ---------------------------------------------------------------------------
    variable l : line;
  begin
    if (flag = 0) then
      write(l, String'("------------------------------------ "));
      writeline(output, l);
      write(l, String'("  xi = ")); write(l, real'(Conv2real(x)));
```

```
      write(l, String'("  yi = ")); write(l, real'(Conv2real(y)));
      write(l, String'("  zi = ")); write(l, real'(Conv2real(z)));
    elsif (flag = 1) then
      write(l, String'("  xo = ")); write(l, real'(Conv2real(x)));
      write(l, String'("  yo = ")); write(l, real'(Conv2real(y)));
      write(l, String'("  zo = ")); write(l, real'(Conv2real(z)));
    else
      write(l, String'("  xn = ")); write(l, real'(Conv2real(x)));
      write(l, String'("  yn = ")); write(l, real'(Conv2real(y)));
      write(l, String'("  zn = ")); write(l, real'(Conv2real(z)));
    end if;
    writeline(output, l);
  end DispReg;
  ------------------------------------------------------------------------------

  ------------------------------------------------------------------------------
  procedure DispAng (angle : in std_logic_vector (31 downto 0)) is
  ------------------------------------------------------------------------------
    variable l : line;
  begin
    write(l, String'("  angle = ")); write(l, real'(Conv2real(angle)));
    writeline(output, l);
    write(l, String'("..................................... "));
    writeline(output, l);
  end DispAng;


end cordic_pkg;
::::::::::::::
cordic_tb.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    testbench of cordic
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.03.22
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
--
--
--    Output:
--------------------------------------------------------------------------------

library STD;
use STD.textio.all;

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

```
use WORK.cordic_pkg.all;


entity cordic_tb is
end cordic_tb;




architecture beh of cordic_tb is

  component cordic
    generic (
      vflag    : boolean ;
      n        : integer );
    port (
      clk, rst    : in  std_logic;
      load        : in  std_logic;
      ready       : out std_logic;
      xi, yi, zi  : in  std_logic_vector (31 downto 0);
      xo, yo, zo  : out std_logic_vector (31 downto 0) );
  end component;

  for cordic_0: cordic use entity work.cordic;



  constant nBit : integer := 32;

  signal clk, rst, load, ready : std_logic := '0';
  signal xi, yi, zi : std_logic_vector(31 downto 0) := X"0000_0000";
  signal xo, yo, zo : std_logic_vector(31 downto 0) := X"0000_0000";

 begin

   cordic_0 : cordic
        generic map (vflag => false, n => 10 )

        port map ( clk => clk, rst => rst,
                   load => load, ready => ready,
                   xi  => xi, yi  => yi, zi  => zi,
                   xo  => xo, yo  => yo, zo  => zo  );


   clk <= not clk after half_period;

   rst <= '0', '1' after 2* half_period;


   process
     variable curr : time;
   begin

      wait until rst = '1';

   ------------------------------------------------------------------------
   -- printf ("\nGrinding on [K, 0, 0]\n");
   -- Circular (X0C, 0L, 0L);
   ------------------------------------------------------------------------
      for i in 0 to 4  loop
        wait until clk = '1';
      end loop;  -- i

      xi <= Conv2fixedPt(1.0/K, nBit);
      yi <= Conv2fixedPt(0.0, nBit);
      zi <= Conv2fixedPt(0.0, nBit);
      wait for 1 ns;
```

```
    load <= '1', '0' after clk_period;
    DispReg(xi, yi, zi, 0);

    while (ready /= '1') loop
      wait until (clk'event and clk='1');
    end loop;
    DispReg(xo, yo, zo, 1);

  ------------------------------------------------------------------------
  -- printf ("\nGrinding on [K, 0, pi/6] -> [0.86602540, 0.50000000, 0]\n");
  -- Circular (X0C, 0L, HalfPi / 3L);
  ------------------------------------------------------------------------
    for i in 0 to 4  loop
      wait until clk = '1';
    end loop;  -- i

    xi <= Conv2fixedPt(1.0/K, nBit);
    yi <= Conv2fixedPt(0.0, nBit);
    zi <= Conv2fixedPt(pi/6.0, nBit);
    wait for 1 ns;
    load <= '1', '0' after clk_period;
    load <= '1', '0' after clk_period;
    DispReg(xi, yi, zi, 0);

    while (ready /= '1') loop
      wait until (clk'event and clk='1');
    end loop;
    DispReg(xo, yo, zo, 1);

  ------------------------------------------------------------------------
  -- printf ("\nGrinding on [K, 0, pi/4] -> [0.70710678, 0.70710678, 0]\n");
  -- Circular (X0C, 0L, HalfPi / 2L);
  ------------------------------------------------------------------------
    for i in 0 to 4  loop
      wait until clk = '1';
    end loop;  -- i

    xi <= Conv2fixedPt(1.0/K, nBit);
    yi <= Conv2fixedPt(0.0, nBit);
    zi <= Conv2fixedPt(pi/4.0, nBit);
    wait for 1 ns;
    load <= '1', '0' after clk_period;
    load <= '1', '0' after clk_period;
    DispReg(xi, yi, zi, 0);

    while (ready /= '1') loop
      wait until (clk'event and clk='1');
    end loop;
    DispReg(xo, yo, zo, 1);

  ------------------------------------------------------------------------
  -- printf ("\nGrinding on [K, 0, pi/3] -> [0.50000000, 0.86602540, 0]\n");
  -- Circular (X0C, 0L, 2L * (HalfPi / 3L));
  ------------------------------------------------------------------------
    for i in 0 to 4  loop
      wait until clk = '1';
    end loop;  -- i

    xi <= Conv2fixedPt(1.0/K, nBit);
    yi <= Conv2fixedPt(0.0, nBit);
    zi <= Conv2fixedPt(pi/3.0, nBit);
    wait for 1 ns;
    load <= '1', '0' after clk_period;
    load <= '1', '0' after clk_period;
    DispReg(xi, yi, zi, 0);

    while (ready /= '1') loop
```

```
        wait until (clk'event and clk='1');
      end loop;
      DispReg(xo, yo, zo, 1);

      for i in 0 to 4  loop
        wait until clk = '1';
      end loop;  -- i

      report "End time is " & time'image(now);



      wait;
   end process;


   --process
   --begin
   --  wait for 2000* clk_period;
   --  assert false report "end of simulation" severity failure;
   --end process;

  --   XXXXXXX XXXXXX XXXXXX XXXXXX XXXXXXX XXXXXX XXXXX

end beh;
```