```
::::::::::::::
c1.adder.rca.vhdl
::::::::::::::
-------------------------------------------------------------------------------
--
--  Purpose:
--
--    Ripple Carry Adder
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  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;


entity adder is
  generic (
    WD     : in natural := 32);

  port (
    an     : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    bn     : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    ci     : in   std_logic := '0';
    cn     : out  std_logic_vector (WD-1 downto 0) := (others=>'0');
    co     : out  std_logic := '0');

end adder;

architecture rca of adder is
begin
  process (an, bn, ci)
    variable sn : std_logic_vector (WD-1 downto 0) := (others=>'0');
    variable c  : std_logic := '0';
  begin  -- process
    c := ci;
    for i in 0 to WD-1 loop
      sn(i) := an(i) xor bn(i) xor c;
      c := (an(i) and bn(i)) or (an(i) and c) or (bn(i) and c);
    end loop;  -- i

    cn <= sn;
    co <= c;
  end process;

end rca;
```

```
::::::::::::::
c2.addsub.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    Add / Sub
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  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;


entity addsub is
  generic (
    WD     : in natural := 32);

  port (
    an    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    bn    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    s     : in   std_logic := '0';
    cn    : out  std_logic_vector (WD-1 downto 0) := (others=>'0');
    co    : out  std_logic := '0');
end addsub;


architecture rtl of addsub is

  component adder
    generic (
      WD     : in natural );
    port (
      an     : in   std_logic_vector (WD-1 downto 0);
      bn     : in   std_logic_vector (WD-1 downto 0);
      ci     : in   std_logic := '0';
      cn     : out  std_logic_vector (WD-1 downto 0);
      co     : out  std_logic := '0');
  end component;

  signal un : std_logic_vector (WD-1 downto 0) := (others=>'0');
begin
```

```
  process (bn, s)
  begin  -- process
    if (s='1') then
      un <= not bn;
    else
      un <= bn;
    end if;
  end process;

  A0 : adder generic map (WD => WD)
      port map (an => an, bn => un, ci => s, cn => cn, co => co);

end rtl;




::::::::::::::
c3.bshift.vhdl
::::::::::::::
-------------------------------------------------------------------------------
--
--  Purpose:
--
--    Barrel Shifter
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  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;


entity bshift is
  generic (
    WD     : in natural := 32;
    SH     : in natural := 5 );

  port (
    di    : in  std_logic_vector (WD-1 downto 0) := (others=>'0');
    nbit   : in  std_logic_vector (SH-1 downto 0) := (others=>'0');
    dq   : out std_logic_vector (WD-1 downto 0) := (others=>'0'));

end bshift;

architecture rtl of bshift is
```

```
begin

  bshft: process (di, nbit)
    variable diX    : std_logic_vector (2*WD-1 downto 0) := (others=>'0');
    variable offset : natural := 0;
    variable result : std_logic_vector (WD-1 downto 0) := (others=>'0');
  begin  -- process bshft

    for i in 2*WD-1 downto 0 loop
      if i < WD then
        diX(i) := di(i);
      else
        diX(i) := di(WD-1);
      end if;
    end loop;  -- i

    offset := to_integer(unsigned(nbit));

    result := diX(WD-1+offset  downto offset);

    dq <= result;

  end process;


end rtl;
::::::::::::::
c4.dff.vhdl
::::::::::::::
-------------------------------------------------------------------------------
--
--  Purpose:
--
--    D FlipFlop
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  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;





entity dff is
  generic (
    WD     : in natural := 32);

  port (
    clk    : in   std_logic := '0';
    rst    : in   std_logic := '0';
    en     : in   std_logic := '0';
    di     : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    dq     : out  std_logic_vector (WD-1 downto 0) := (others=>'0') );

end dff;



architecture rtl of dff is
begin

  Reg: process (clk, rst)
  begin  -- process Reg
    if rst = '0' then                   -- asynchronous reset (active low)
      dq <= (others=>'0');
    elsif clk'event and clk = '1' then  -- rising clock edge
      if (en='1') then
        dq <= di;
      end if;
    end if;
  end process Reg;

end rtl;



-- entity dff1 is
--   generic (
--     WD     : in natural := 32);
--
--   port (
--     clk    : in   std_logic := '0';
--     rst    : in   std_logic := '0';
--     di     : in   std_logic;
--     dq     : out  std_logic);

-- end dff1;



-- architecture rtl of dff1 is
-- begin

--   Reg: process (clk, rst)
--   begin  -- process Reg
--     if rst = '0' then                   -- asynchronous reset (active low)
--       dq <= (others=>'0');
--     elsif clk'event and clk = '1' then  -- rising clock edge
--       dq <= di;
--     end if;
--   end process Reg;

-- end rtl;


::::::::::::::
c5.counter.vhdl
::::::::::::::
-------------------------------------------------------------------------------
--
--  Purpose:
--
--    Synchronous Counter
--
--  Discussion:
```

```vhdl
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  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;


entity counter is
  generic (
    SH     : in natural :=  5 );

  port (
    clk    : in   std_logic := '0';
    rst    : in   std_logic := '0';
    en     : in   std_logic := '0';
    ld     : in   std_logic := '0';
    di     : in   std_logic_vector (SH-1 downto 0) := (others=>'0');
    dq     : out  std_logic_vector (SH-1 downto 0) := (others=>'0') );

end counter;


architecture rtl of counter is

  component adder
    generic (
      WD     : in natural := 32 );

    port (
      an    : in   std_logic_vector (SH-1 downto 0);
      bn    : in   std_logic_vector (SH-1 downto 0);
      ci    : in   std_logic;
      cn    : out  std_logic_vector (SH-1 downto 0);
      co    : out  std_logic );
  end component;

  signal cnt    : std_logic_vector (SH-1 downto 0);
  signal cntInc : std_logic_vector (SH-1 downto 0);
  signal tbd : std_logic;

begin

  inc : adder generic map (WD => SH)
    port map (an=>cnt, bn=>(others=>'0'), ci=>'1', cn=>cntInc, co=>tbd);

  Reg: process (clk, rst)
  begin  -- process Reg
    if rst = '0' then                   -- asynchronous reset (active low)
```

```
      cnt <= (others=>'0');
    elsif clk'event and clk = '1' then  -- rising clock edge
      if (ld='1') then
        cnt <= di;
      elsif (en='1') then
        cnt <= cntInc;
      end if;
    end if;
  end process Reg;

  dq <= cnt;

end rtl;
::::::::::::::
c6.rom.vhdl
::::::::::::::
-------------------------------------------------------------------------------
--
--  Purpose:
--
--    ROM Model
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  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 rom is
  generic (
    WD     : in natural := 32;
    SH     : in natural :=  5;
    PWR    : in natural := 64);

  port (
    addr   : in   std_logic_vector (SH-1 downto 0) := (others=>'0');
    cs     : in   std_logic := '0';
    data   : out  std_logic_vector (WD-1 downto 0) := (others=>'0') );

end rom;

architecture rtl of rom is
   type rarray is array (natural range <>) of real;
```

```vhdl
 constant angles : rarray :=
                  (  7.8539816339744830962E-01,  -- pi/4 rad
                     4.6364760900080611621E-01,
                     2.4497866312686415417E-01,
                     1.2435499454676143503E-01,
                     6.2418809995957348474E-02,
                     3.1239833430268276254E-02,
                     1.5623728620476830803E-02,
                     7.8123410601011112965E-03,
                     3.9062301319669718276E-03,
                     1.9531225164788186851E-03,
                     9.7656218955931943040E-04,
                     4.8828121119489827547E-04,
                     2.4414062014936176402E-04,
                     1.2207031189367020424E-04,
                     6.1035156174208775022E-05,
                     3.0517578115526096862E-05,
                     1.5258789061315762107E-05,
                     7.6293945311019702634E-06,
                     3.8146972656064962829E-06,
                     1.9073486328101870354E-06,
                     9.5367431640596087942E-07,
                     4.7683715820308885993E-07,
                     2.3841857910155798249E-07,
                     1.1920928955078068531E-07,
                     5.9604644775390554414E-08,
                     2.9802322387695303677E-08,
                     1.4901161193847655147E-08,
                     7.4505805969238279871E-09,
                     3.7252902984619140453E-09,
                     1.8626451492309570291E-09,
                     9.3132257461547851536E-10,
                     4.6566128730773925778E-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,
                     1.7347234759768070944E-18,
                     1.7347234759768070944E-18,
                     1.7347234759768070944E-18,
                     1.7347234759768070944E-18  );


signal dinInc : std_logic_vector (SH-1 downto 0);
```

```
begin

  ROM: process (addr, cs)
    type darray is array (0 to PWR) of std_logic_vector (WD-1 downto 0);
    variable romData : darray;
    variable initRom : boolean := false;
    variable cntInt : integer := 0;
    variable angleMin : std_logic_vector (WD-1 downto 0);
  begin  -- process Reg
    if (initRom=false) then
      for i in 0 to PWR-1 loop
        romData(i) := Conv2fixedPt(angles(i), WD);
      end loop;  -- i
      initRom := true;
    end if;

    if cs = '1' then                    -- asynchronous reset (active low)
      data <= romData(to_integer(unsigned(addr)));
    else
      data <= (others=>'1');
    end if;
  end process ROM;

end rtl;
::::::::::::::
c7.mux.vhdl
::::::::::::::
-------------------------------------------------------------------------------
--
--  Purpose:
--
--    Mux
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  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;


entity mux is
  generic (
    WD     : in natural := 32);

  port (
    an    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    bn    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    s     : in   std_logic := '0';
    cn    : out  std_logic_vector (WD-1 downto 0) := (others=>'0') );
end mux;


architecture rtl of mux is
begin

  process (an, bn, s)
  begin  -- process
    if (s='1') then
      cn <= bn;
    else
      cn <= an;
    end if;
  end process;

end rtl;




::::::::::::::
m1.disp.vhdl
::::::::::::::
-------------------------------------------------------------------------------
--
--  Purpose:
--
--    Monitors signals and writes their values
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  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 disp is
  generic (
    WD     : in natural := 32;
    SH     : in natural :=  5;
    PWR    : in natural := 64);
  port (
    clk   : in   std_logic := '0';
    load  : in   std_logic := '0';
    cntEn : in   std_logic := '0';
    ready : in   std_logic := '0';
    xi    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    yi    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    zi    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    xn    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    yn    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    zn    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    angle : in   std_logic_vector (WD-1 downto 0) := (others=>'0') );

end disp;

architecture beh of disp is

  signal dinInc : std_logic_vector (SH-1 downto 0);

begin

  monitor: process
  begin  -- process Reg

    wait until (clk'event and clk = '0');

    if (load='1') then
      DispReg(xn, yn, zn, 0);
      DispAng(angle);
    elsif (ready='1') then
      DispReg(xn, yn, zn, 1);
      DispAng(angle);
    elsif (cntEn='1') then
      DispReg(xn, yn, zn, 2);
      DispAng(angle);
    end if;


  end process monitor;

end beh;
::::::::::::::
cordic_pkg.vhdl
::::::::::::::
-------------------------------------------------------------------------------
--
--  Purpose:
--
--    utility package of cordic
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  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_rtl.vhdl
::::::::::::::
----------------------------------------------------------------------------
--
--  Purpose:
--
--    behavioral model of cordic
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  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 (
    WD          : in natural := 32;
    SH          : in natural :=  5;
    nIter       : in std_logic_vector (4 downto 0) := "01010" );

  port (
    clk, rst    : in  std_logic;
    load        : in  std_logic;
    ready       : out std_logic := '0' ;
    xi, yi, zi  : in  std_logic_vector (WD-1 downto 0) := (others=>'0');
    xo, yo, zo  : out std_logic_vector (WD-1 downto 0) := (others=>'0'));

end cordic;


architecture beh of cordic is

  component adder
    generic (
      WD     : in natural := 32 );

    port (
      an    : in   std_logic_vector (WD-1 downto 0);
      bn    : in   std_logic_vector (WD-1 downto 0);
      ci    : in   std_logic;
      cn    : out  std_logic_vector (WD-1 downto 0);
      co    : out  std_logic  );
  end component;


  component addsub is
    generic (
      WD     : in natural := 32 );

    port (
      an    : in   std_logic_vector (WD-1 downto 0);
      bn    : in   std_logic_vector (WD-1 downto 0);
      s     : in   std_logic;
      cn    : out  std_logic_vector (WD-1 downto 0);
      co    : out  std_logic  );
  end component;


  component bshift
    generic (
      WD     : in natural := 32;
      SH     : in natural := 5 );

    port (
      di        : in  std_logic_vector (WD-1 downto 0);
      nbit       : in  std_logic_vector (SH-1 downto 0);
      dq       : out std_logic_vector (WD-1 downto 0) );
  end component;
```

```vhdl
  component dff
    generic (
      WD     : in natural := 32);

    port (
      clk  : in   std_logic ;
      rst  : in   std_logic ;
      en   : in   std_logic ;
      di  : in   std_logic_vector (WD-1 downto 0);
      dq   : out  std_logic_vector (WD-1 downto 0)  );
  end component;


--  component dff1
--    generic (
--      WD     : in natural := 32);
--
--    port (
--      clk  : in   std_logic ;
--      rst  : in   std_logic ;
--      di   : in   std_logic ;
--      dq   : out  std_logic  );
--  end component;


  component counter
    generic (
      SH     : in natural :=  5 );

    port (
      clk  : in   std_logic := '0';
      rst  : in   std_logic := '0';
      en   : in   std_logic := '0';
      ld   : in   std_logic := '0';
      di   : in   std_logic_vector (4 downto 0);
      dq   : out  std_logic_vector (4 downto 0) );
  end component;


  component rom is
    generic (
      WD     : in natural := 32;
      SH     : in natural :=  5;
      PWR    : in natural := 64);

    port (
      addr : in   std_logic_vector (SH-1 downto 0);
      cs   : in   std_logic ;
      data : out  std_logic_vector (WD-1 downto 0) );
  end component;

  component mux is
    generic (
      WD     : in natural := 32);

    port (
      an    : in   std_logic_vector (WD-1 downto 0);
      bn    : in   std_logic_vector (WD-1 downto 0);
      s     : in   std_logic;
      cn    : out  std_logic_vector (WD-1 downto 0) );
  end component;


  component disp is
    generic (
      WD     : in natural := 32;
      SH     : in natural :=  5 );

    port (
      clk   : in   std_logic := '0';
      load  : in   std_logic := '0';
      cntEn : in   std_logic := '0';
      ready : in   std_logic := '0';
      xi    : in   std_logic_vector (WD-1 downto 0);
      yi    : in   std_logic_vector (WD-1 downto 0);
      zi    : in   std_logic_vector (WD-1 downto 0);
      xn    : in   std_logic_vector (WD-1 downto 0);
      yn    : in   std_logic_vector (WD-1 downto 0);
      zn    : in   std_logic_vector (WD-1 downto 0);
      angle : in   std_logic_vector (WD-1 downto 0) );
  end component;

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

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


  signal xt, yt, zt : std_logic_vector(WD-1 downto 0) := (others=>'0');
  signal xn, yn, zn : std_logic_vector(WD-1 downto 0) := (others=>'0');
  signal xr, yr, zr : std_logic_vector(WD-1 downto 0) := (others=>'0');
  signal xnS, ynS   : std_logic_vector(WD-1 downto 0) := (others=>'0');
  signal angle      : std_logic_vector(WD-1 downto 0) := (others=>'0');

  signal cnt        : std_logic_vector(SH-1 downto 0) := (others=>'0');

  signal S, invS : std_logic := '0';
  signal open_co : std_logic;

  signal cntEn : std_logic := '0';
  signal endIter  : std_logic := '0';
  signal dffEn : std_logic := '0';

begin



   ShftX : bshift generic map (WD=>32, SH=>5)
     port map (di  => xn, nbit => cnt, dq => xnS);

   ShftY : bshift generic map (WD=>32, SH=>5)
     port map (di  => yn, nbit => cnt, dq => ynS);

   S <= zn(WD-1);
   invS <= not zn(WD-1);


   AddsubX : addsub generic map (WD=>32)
     port map (an =>xn, bn => ynS,   s=>invS, cn=>xr, co=>open_co);

   AddsubY : addsub generic map (WD=>32)
     port map (an =>yn, bn => xnS,   s=>   S, cn=>yr, co=>open_co);

   AddsubZ : addsub generic map (WD=>32)
     port map (an =>zn, bn => angle, s=>invS, cn=>zr, co=>open_co);



   -- if (zn(WD-1)='0') then
   --   xr := +xn  - ynS;
   --   yr := +xnS + yn;
   --   zr := +zn  - angle;
   -- else
   --   xr := -xn  + ynS;
   --   yr := -xnS + yn;
   --   zr := +zn  + angle;
   -- end if;


   MuxX: mux generic map (WD=>32)
     port map (an=>xr, bn=>xi, s=>load, cn=>xt);

   MuxY: mux generic map (WD=>32)
     port map (an=>yr, bn=>yi, s=>load, cn=>yt);

   Muxz: mux generic map (WD=>32)
     port map (an=>zr, bn=>zi, s=>load, cn=>zt);


   dffEn <= (cntEn and (not endIter)) or load;

   RegX: dff generic map (WD=>32)
     port map (clk=>clk, rst=>rst, en=>dffEn, di=>xt, dq=>xn);

   RegY: dff generic map (WD=>32)
     port map (clk=>clk, rst=>rst, en=>dffEn, di=>yt, dq=>yn);

   RegZ: dff generic map (WD=>32)
     port map (clk=>clk, rst=>rst, en=>dffEn, di=>zt, dq=>zn);



   -- EnReg: dff generic map (WD=>1)
   --   port map (clk=>clk, rst=>rst, di(0)=>preEn, dq(0)=>cntEn);

   EnReg: process (clk, rst)
   begin  -- process EnReg
     if rst = '0' then
       cntEn <= '0';
     elsif clk'event and clk = '1' then
       if (load='1') then
         cntEn <= '1';
       elsif (endIter='1') then
         cntEn <= '0';
       end if;
     end if;
   end process EnReg;


   endIter <= '1' when cnt=nIter else '0';
   ready <= endIter;




   CntReg: counter generic map (SH=>5)
     port map (clk=>clk, rst=>rst, en=>cntEn, ld=>endIter, di=>"00000", dq=>cnt);


   AngRom: rom generic map (WD=>32, SH=>5, PWR=>64)
     port map (addr=>cnt, cs=>'1', data=>angle);


   Monitor: disp generic map (WD=>32, SH=>5)
     port map (clk=>clk, load=>load, cntEn=>cntEn, ready=>endIter,
               xi=>xi, yi=>yi, zi=>zi,
               xn=>xn, yn=>yn, zn=>zn, angle=>angle);




        -- 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, WD);
        --yo <= Conv2fixedPt(ry, WD);
    --XXXXXXX XXXXXX XXXXXX XXXXXX XXXXXXX XXXXXX XXXXX

end beh;
::::::::::::::
cordic_tb.vhdl
::::::::::::::
-------------------------------------------------------------------------------
--
--  Purpose:
--
--    testbench of cordic
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  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
    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 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
   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
   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;
```