```
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    behavioral model of cordic
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.03.27
--
--  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 (
    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 (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 (
    din        : in  std_logic_vector (WD-1 downto 0);
    nbit       : in  std_logic_vector (SH-1 downto 0);
    dout       : out std_logic_vector (WD-1 downto 0) );
end component;


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

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


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

  port (
    clk  : in   std_logic := '0';
    rst  : in   std_logic := '0';
    en   : in   std_logic := '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;


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

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


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');

begin


  shftX : bshift
    port map (din  => xn, nbit => cnt, dout => xnS);
  shftY : bshift
    port map (din  => yn, nbit => cnt, dout => ynS);

  addsubX : addsub
    port map (an =>xn, bn => ynS,   s=>not z(WD-1), cn=>xr, co=>,  );
  addsubX : addsub
    port map (an =>yn, bn => xnS,   s=> z(WD-1),    cn=>yr, co=>,  );
  addsubX : addsub
    port map (an =>zn, bn => angle, s=>not z(WD-1), cn=>zr, co=>,  );




  -- if (zn(WD-1)='0') then
  --   xt := +xn  - ynS;
  --   yt := +xnS + yn;
  --   zt := +zn  - angle;
  -- else
  --   xt := -xn  + ynS;
  --   yt := -xnS + yn;
  --   zt := +zn  + angle;
  -- end if;
  RegX: dff port map (clk=>clk, rst=>rst, din=>xr, dq=>xn);
  RegY: dff port map (clk=>clk, rst=>rst, din=>yr, dq=>yn);
  RegZ: dff port map (clk=>clk, rst=>rst, din=>zr, dq=>zn);


  enReg: dff port map (clk=>clk, rst=>rst, din=>load, dq=>preEn);
  en <= preEn or load;

  Cnt: counter port map (clk=>clk, rst=>rst, en=>en, dq=>nIter);

  AngRom: rom port map (addr=>cnt, cs=>, data=>angle);



   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), WD) ;

     xn <= xi;
     yn <= yi;
     zn <= zi;
     wait for 1 ns;
     DispReg(xn, yn, zn, 2);
     DispAng(angle);

     LFOR: for j in 1 to n loop

       if (zn(WD-1) = '0')  then
       else

       wait until clk='1';

       if (angle_length < j + 1) then
         angle <= std_logic_vector(shift_right(signed(angle), 1));
```

```
        else
          angle <= Conv2fixedPt(angles(j), WD) ;
        end if;

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

      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, WD);
        --yo <= Conv2fixedPt(ry, WD);
        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;
```