

```
-----  
--  
-- Purpose:  
--  
-- behavioral model of cordic  
--  
-- Discussion:  
--  
--  
-- Licensing:  
--  
-- This code is distributed under the GNU LGPL license.  
--  
-- Modified:  
--  
-- 2012.03.26  
--  
-- 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 (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;
```

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

```
  component bshift
```

```
    port (  
      din      : in  std_logic_vector (31 downto 0) := X"0000_0000";
```

```

nbit      : in  std_logic_vector (4 downto 0) := X"00";
dout      : out std_logic_vector (31 downto 0) := X"0000_0000";
end component;

component addsub is
  port (an, bn,      : in  std_logic_vector (31 downto 0) := X"0000_0000";
        s,          : in  std_logic := '0';
        cn,          : out std_logic_vector (31 downto 0) := X"0000_0000";
        co           : out std_logic := '0');
end addsub;

```

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(31), cn=>xr, co=>, );
addsubX : addsub
  port map (an =>yn, bn => xnS, s=> z(31), cn=>yr, co=>, );
addsubX : addsub
  port map (an =>zn, bn => angle, s=>not z(31), cn=>zr, co=>, );

-- if (zn(31)='0') then
--   xt := +xn - ynS;
--   yt := +xnS + yn;
--   zt := +zn - angle;
-- else
--   xt := -xn + ynS;
--   yt := -xnS + yn;
--   zt := +zn + angle;
-- end if;

-- purpose: Register X
-- inputs : clk, rst
-- outputs: xn
RegX: process (clk, rst)
begin -- process RegX
  if (rst='0') then
    xn <= X"0000_0000";
  elsif rising_edge(clk) then
    xn <= xr;
  end if;
end process RegX;

-- purpose: Register Y
-- inputs : clk, rst
-- outputs: yn
RegY: process (clk, rst)
begin -- process RegY
  if (rst='0') then
    yn <= X"0000_0000";
  elsif rising_edge(clk) then
    yn <= yr;
  end if;
end process RegY;

-- purpose: Register Z
-- inputs : clk, rst
-- outputs: zn
RegZ: process (clk, rst)
begin -- process RegZ
  if (rst='0') then

```

```

    zn <= X"0000_0000";
elsif rising_edge(clk) then
    zn <= zr;
    end if;
end if;
end process RegZ;

```

```

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;
        DispReg(xn, yn, zn, 2);
        DispAng(angle);

        LFOR: for j in 1 to n loop

            if (zn(31) = '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), 32) ;
                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, 32);
                --yo <= Conv2fixedPt(ry, 32);
                xo <= xn;
            end if;
        end process main;

```

```
yo <= yn;
zo <= zn;
wait for 1 ns;

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

end if;

end loop;

wait;

end process main;

XXXXXXXX XXXXXX XXXXXX XXXXXX XXXXXXXX XXXXXX XXXXX

end beh;
```