

```
-----
--
-- Purpose:
--
-- behavioral model of cordic
--
-- Discussion:
--
--
-- Licensing:
--
-- This code is distributed under the GNU LGPL license.
--
-- Modified:
--
-- 2012.03.31
--
-- 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;
  nBit    : 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);
    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 ;
    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 (
    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;

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

  port (
    load  : in  std_logic := '0';
    ready : in  std_logic := '0';
    cnt   : in  std_logic_vector (SH-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 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, nIter : std_logic_vector(SH-1 downto 0) := (others=>'0');

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

signal preEn, cntEn : std_logic := '0';
signal endIter, endIterD : std_logic := '0';
signal preReady, readySig : std_logic := '0';

begin

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

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

S <= zn(WD-1);
S <= 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;

```

```

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

```

```

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

```

```

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

```

```
preEn <= load and not preReady;
```

```

EnReg: dff1 generic map (WD=>1)
  port map (clk=>clk, rst=>rst, di=>preEn, dq=>cntEn);

```

```

endIter <= '1' when cnt=nBit else '0';
preReady <= endIter xor endIterD;

```

```

DisReg1: dff1 generic map (WD=>1)
  port map (clk=>clk, rst=>rst, di=>endIter, dq=>endIterD);

```

```

ready <= readySig;
DisReg2: dff1 generic map (WD=>1)
  port map (clk=>clk, rst=>rst, di=>preReady, dq=>readySig);

```

```

CntReg: counter generic map (SH=>5)
  port map (clk=>clk, rst=>rst, en=>cntEn, dq=>nIter);

```

```

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 (load=>load, ready=>readySig, cnt=>cnt,
    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);

```

```
XXXXXXXX XXXXXX XXXXXX XXXXXX XXXXXXXX XXXXXX XXXXX
```

```
end beh;
```