

```
-----
--
-- Purpose:
--   behavioral model of cordic
--
-- Discussion:
--
-- Licensing:
--   This code is distributed under the GNU LGPL license.
--
-- Modified:
--   2012.03.13
--
-- 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;
```

```
entity cordic is
```

```
  generic (
    n      : integer := 10);
```

```
  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 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 :=
    ( 7.8539816339744830962E-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 );
```

```
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 );
```

```
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;
```

```
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 load='1';
```

```
  xn <= xi;  
  yn <= yi;  
  zn <= zi;
```

```
  LFOR: for j in 1 to n loop
```

```
    wait until clk='1';
```

```
    if (Conv2real(zn) >= 0.0) then  
      xt := std_logic_vector(signed(xn) - shift_right(signed(yn), 1));  
      yt := std_logic_vector(shift_right(signed(xn), 1) + signed(yn));  
      zt := std_logic_vector(signed(zn) - signed(angle));  
    else  
      xt := std_logic_vector(signed(xn) + shift_right(signed(yn), 1));  
      yt := std_logic_vector(-shift_right(signed(xn), 1) + signed(yn));  
      zt := std_logic_vector(signed(zn) + signed(angle));  
    end if;
```

```
    xn <= xt;  
    yn <= yt;  
    zn <= zt;
```

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

```

    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);
    zo <= zn;
    ready <= '1';

    wait until clk='1';

end if;

wait;

end process main;

XXXXXXX XXXXXX XXXXXX XXXXXX XXXXXXX XXXXXX XXXXX

end beh;

```