```
::::::::::::::
makefile
::::::::::::::
INCD = /home/young/MyWork/inc
LIBD = /home/young/MyWork/lib


SRC = cordic_beh.vhdl  \
      cordic_vtb.vhdl  \
      sendrecv.c       \
      cordic_ghdl.c    \
      cordic_ghdl_tb.c \

OBJ = cordic_pkg.o     \
      cordic_beh.o     \
      cordic_vtb.o     \
      *cordic_vtb.o    \
      sendrecv.o       \
      cordic_ghdl.o    \
      cordic_ghdl_tb.o \

LIB = libcordic-ghdl.a \
      work-*.cf        \

EXE = cordic_vtb       \
      cordic_ghdl_tb   \
      ip_fifo          \
      op_fifo          \


sendrecv.o: sendrecv.c
        gcc -c -Wall sendrecv.c


cordic_vtb: sendrecv.o cordic_vtb.vhdl
        cp /home/young/MyWork/5.cordic_vhdl/a.beh/cordic_pkg.vhdl .
        cp /home/young/MyWork/5.cordic_vhdl/a.beh/cordic_beh.vhdl .
        ghdl -a cordic_pkg.vhdl
        ghdl -a cordic_beh.vhdl
        ghdl -a cordic_vtb.vhdl
        ghdl -e -Wl,sendrecv.o cordic_vtb

cordic_ghdl: cordic_ghdl.c
        gcc  -c -Wall cordic_ghdl.c

libcordic-ghdl.a: cordic_ghdl.o
#       ar -rcs libcordic-ghdl.a cordic_ghdl.o
        ar -cvq libcordic-ghdl.a cordic_ghdl.o

cordic_ghdl_tb: cordic_ghdl_tb.c libcordic-ghdl.a
```

```
#       gcc  -Wall cordic_ghdl_tb.c  libcordic-ghdl.a -o cordic_ghdl_tb
        gcc  -Wall cordic_ghdl_tb.c  -L. -lcordic-ghdl -o cordic_ghdl_tb

all: cordic_ghdl_tb cordic_vtb
        cp libcordic-ghdl.a ${LIBD}
        \rm -f ${OBJ}

run: all
        ./cordic_ghdl_tb


clean :
        \rm -f *.o *~ *# *.UA0
        \rm -f ${OBJ} ${LIB} ${EXE}
        \rm -f a.out work*.cf


tar :
        mkdir src
        cp makefile ${SRC}  src
        tar cvf ghdlIF.tar src
        \rm -fr src

print :
        /bin/more makefile ${SRC} > ghdlIF.print



::::::::::::::
cordic_beh.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    behavioral model of cordic
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.03.22
--
--  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 (
    vflag    : boolean := false;
    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;

   constant angles : real_array :=
                     (  7.853981633974830962E-01,  -- pi/4 rad
                        4.636476090008061162E-01,
                        2.449786631268641541E-01,
                        1.243549945467614350E-01,
                        6.241880999595734847E-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 );


 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 (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;

    if (vflag = true) then
      DispReg(xn, yn, zn, 2);
      DispAng(angle);
    end if;

    LFOR: for j in 1 to n loop

      if (zn(31) = '0')  then
        xt := std_logic_vector(signed(xn) - shift_right(signed(yn), j-1));
        yt := std_logic_vector(shift_right(signed(xn), j-1) + signed(yn));
        zt := std_logic_vector(signed(zn) - signed(angle));
      else
        xt := std_logic_vector(signed(xn) + shift_right(signed(yn), j-1));
        yt := std_logic_vector(-shift_right(signed(xn), j-1) + signed(yn));
        zt := std_logic_vector(signed(zn) + signed(angle));
      end if;

      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;
      if (vflag = true) then
        DispReg(xn, yn, zn, 2);
        DispAng(angle);
      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);
        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;
::::::::::::::
cordic_vtb.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    testbench of cordic interfacing with C
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
```

```
--  Modified:
--
--    2013.09.04
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
--
--
--    Output:
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
package ghdlif is
  function send_op(v : real) return real;
  attribute foreign of send_op : function is "VHPIDIRECT send_op";

  function recv_ip (v : real) return real;
  attribute foreign of recv_ip : function is "VHPIDIRECT recv_ip";
end ghdlif;

package body ghdlif is
  function send_op (v : real) return real is
  begin
    assert false severity failure;
  end send_op;

  function recv_ip (v : real) return real is
  begin
    assert false severity failure;
  end recv_ip;
end ghdlif;



library STD;
use STD.textio.all;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;


use WORK.cordic_pkg.all;
use WORK.ghdlif.all;
```

```
--------------------------------------------------------------------------------
entity cordic_vtb is
end cordic_vtb;




architecture behav of cordic_vtb 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
     variable x : real;
     variable y : real;
     variable z : real;
     variable l : line;
   begin
     ------------------------------------------------
```

```
  -- input x, y, z
  ------------------------------------------------
  x := 0.0;
  y := 0.0;
  z := 0.0;

  x := recv_ip(x);
  y := recv_ip(y);
  z := recv_ip(z);
  ------------------------------------------------



   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(x, nBit);
   yi <= Conv2fixedPt(y, nBit);
   zi <= Conv2fixedPt(z, 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);

   x := real'(Conv2real(xo));
   y := real'(Conv2real(yo));
   z := real'(Conv2real(zo));

   for i in 0 to 4  loop
     wait until clk = '1';
   end loop;  -- i


  ------------------------------------------------
  -- output x, y, z
  ------------------------------------------------
  write (l, string'("send_op ("));
  write (l, x);
  write (l, string'(") = "));
```

```
     write (l, send_op (x));
     writeline (output, l);

     write (l, string'("send_op ("));
     write (l, y);
     write (l, string'(") = "));
     write (l, send_op (y));
     writeline (output, l);

     write (l, string'("send_op ("));
     write (l, z);
     write (l, string'(") = "));
     write (l, send_op (z));
     writeline (output, l);
     ------------------------------------------------
     wait;

   end process;


--   process
--   begin
--     wait for 2000* clk_period;
--     assert false report "end of simulation" severity failure;
--   end process;


end behav;



::::::::::::::
sendrecv.c
::::::::::::::
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>

#define MSGSIZ 63

char *ip_fifo = "ip_fifo";
char *op_fifo = "op_fifo";


double recv_ip(double val) {
```

```
  int fd;


  /*
  if (mkfifo(ip_fifo, 0666) == -1) {
    if(errno != EEXIST)
      perror("vhdl receiver: mkfifo");
  }
  */

  if ((fd = open(ip_fifo, O_RDWR )) < 0)
    perror("vhdl: ip_fifo open failed");

  if (read(fd, &val, sizeof(val)) < 0)
      perror("vhdl: ip message read failed");

  printf("vhdl ip message received: %f\n", val);

  return val;

}



double send_op (double val)
{
  int fd, nwrite;

  if ((fd = open(op_fifo, O_WRONLY )) < 0)
    perror("vhdl: op_fifo open failed");

  if ((nwrite = write(fd, &val, sizeof(val)) == -1))
    perror("vhdl: op message write failed");

  return val;
}


::::::::::::::
cordic_ghdl.c
::::::::::::::
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
#define MSGSIZ 63

char *ip_fifo = "ip_fifo";
char *op_fifo = "op_fifo";

/*---------------------------------------------------------------*/
double send_ip (double val) ;
double recv_op() ;
void  cordic_ghdl( double *x, double *y, double *z) ;
/*---------------------------------------------------------------*/



/*---------------------------------------------------------------*/
double send_ip (double val)
{
  int fd, nwrite;

  if ((fd = open(ip_fifo, O_WRONLY )) < 0)
    perror("C: ip_fifo open failed");

  if ((nwrite = write(fd, &val, sizeof(val)) == -1))
    perror("C: ip message write failed");

  return val;
}



/*---------------------------------------------------------------*/
double recv_op() {
  int fd;
  double val;

  /*
  if (mkfifo(op_fifo, 0666) == -1) {
    if(errno != EEXIST)
      perror("C: receiver: mkfifo");
  }
  */

  if ((fd = open(op_fifo, O_RDWR  )) < 0)
    perror("C: op_fifo open failed");

  if (read(fd, &val, sizeof(val)) < 0)
      perror("C: op message read failed");

  printf("C: op message received: %f\n", val);

  return val;
```

```
}


/*---------------------------------------------------------------*/
void  cordic_ghdl( double *x, double *y, double *z) {
  double vx, vy, vz;

  pid_t pid;

  if (mkfifo(ip_fifo, 0666) == -1) {
     if(errno != EEXIST)
       perror("mkfifo error");
  }

  if (mkfifo(op_fifo, 0666) == -1) {
     if(errno != EEXIST)
       perror("mkfifo error");
  }


  pid = fork();


  if (pid == 0) {
     execl("/usr/bin/ghdl", "ghdl", "-r", "cordic_vtb", (char *) 0);
  } else {
    send_ip(*x);
    send_ip(*y);
    send_ip(*z);
    vx=recv_op();
    vy=recv_op();
    vz=recv_op();
    *x = vx;
    *y = vy;
    *z = vz;
  }



}
::::::::::::::
cordic_ghdl_tb.c
::::::::::::::
#include <stdio.h>

void  cordic_ghdl( double *x, double *y, double *z) ;
```

```
/*--------------------------------------------------------------*/
int main(int argc, char *argv[]) {
  double x =.0;
  double y =.0;
  double z =.0;


  x = 6.072529349476099e-1;
  y = 0.0e0;
  z = 0.0e0;


  printf("---------------------------------\n");
  printf("* before cordic_ghdl \n");
  printf("  x =%f \n", x);
  printf("  y =%f \n", y);
  printf("  z =%f \n", z);
  printf("---------------------------------\n");

  cordic_ghdl(&x, &y, &z);

  printf("---------------------------------\n");
  printf("* after cordic_ghdl \n");
  printf("  x =%f \n", x);
  printf("  y =%f \n", y);
  printf("  z =%f \n", z);
  printf("---------------------------------\n");

  return 0;
}
```