```
::::::::::::::
makefile
::::::::::::::
#-------------------------------------------------------------------------
# 1st pass: make import_files
# 2nd pass: make run ...
#-------------------------------------------------------------------------

.SUFFIXES : .o .vhdl

.vhdl.o :
        ghdl -a  $<



#-------------------------------------------------------------------------
# adder (CONF = rca / cca)
#-------------------------------------------------------------------------
Dadder = /home/young/MyWork/VLSI/c.Arithmetic/adder/rtl
Fadder = adder.flist
adder :
        cd ${Dadder};      \
        make export_files CONF=cca;

        mv filelist ${Fadder};



#-------------------------------------------------------------------------
# bshift (CONF = mux / rtl)
#-------------------------------------------------------------------------
Dbshift = /home/young/MyWork/VLSI/c.Arithmetic/bshift/rtl
Fbshift = bshift.flist
bshift :
        cd ${Dbshift};      \
        make export_files CONF=mux;

        mv filelist ${Fbshift};



#-------------------------------------------------------------------------
# rom (CONF = fint / fint_file / real / real_file)
#-------------------------------------------------------------------------
Drom = /home/young/MyWork/VLSI/d.FPGA/rom/rtl
From = rom.flist
rom :
        cd ${Drom};      \
        make export_files CONF=fint;
```

```
        mv filelist ${From};



#-----------------------------------------------------------------------
SRC_angle = angle_comp.c
EXE_angle = angle_comp

angle_comp : ${SRC_angle}
        gcc -o angle_comp angle_comp.c -lm
        ./angle_comp | more

angle_files :
        more angle_comp.c > file/angle.files
        tar cvf file/angle.tar ${SRC_angle} makefile

import_files : adder bshift rom



#-----------------------------------------------------------------------
SRC_adder  =$(shell cat ${Fadder})
SRC_bshift =$(shell cat ${Fbshift})
SRC_rom    =$(shell cat ${From})


OBJ = cordic_pkg.o                              \
      ${SRC_adder:.vhdl=.o}                     \
      c2.addsub.o                               \
      ${SRC_bshift:.vhdl=.o}                    \
      c4.dff.o                                  \
      c5.counter.o                              \
      ${SRC_rom:.vhdl=.o}                       \
      c7.mux.o                                  \
      m1.disp.o                                 \
      cordic_rtl.o                              \
      cordic_tb.o                               \

EXT = cordic_tb



cordic_tb : ${OBJ}
        ghdl -e cordic_tb

anal : ${OBJ}

elab : cordic_tb

run : cordic_tb
```

```
	ghdl -r cordic_tb --vcd=cordic.vcd --stop-time=2000ns

all : anal elab run

wave :
	gtkwave cordic.vcd &


#-----------------------------------------------------------------------------
clean :
	\rm -f *.o *~ *# *.cf *.tar *.print
	\rm -f *_tb
	\rm -f *_conf
	\rm -f *.vcd
	\rm -f ${EXE}

clean_exe :
	\rm -f ${EXE}


#-----------------------------------------------------------------------------
SRC	=  ${OBJ:.o=.vhdl}

print :
	more makefile ${SRC}   > cordic.print

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

::::::::::::::
cordic_pkg.vhdl
::::::::::::::
-------------------------------------------------------------------------------
--
--  Purpose:
--
--    utility package of cordic
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
```

```
--    2012.03.22
--
--  Author:
--
--    Young W. Lim
--
--  Functions:
--  Conv2fixedPt (x : real; n : integer) return std_logic_vector;
--  Conv2real (s : std_logic_vector (31 downto 0) ) return real;
--
--
--------------------------------------------------------------------------------

library STD;
use STD.textio.all;

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


package cordic_pkg is

  function Conv2fixedPt (x : real; n : integer) return std_logic_vector;
  function Conv2real (s : std_logic_vector (31 downto 0) ) return real;

  procedure DispReg (x, y, z : in std_logic_vector (31 downto 0);
                     flag : in integer );
  procedure DispAng (angle : in std_logic_vector (31 downto 0)) ;

  constant clk_period : time := 20 ns;
  constant half_period : time := clk_period / 2.0;

  constant pi : real := 3.141592653589793;
  constant K : real := 1.646760258121;

end cordic_pkg;



package body cordic_pkg is

  --------------------------------------------------------------------------------
  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;
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
procedure DispReg (x, y, z : in std_logic_vector (31 downto 0);
                   flag : in integer ) is
--------------------------------------------------------------------------------
  variable l : line;
begin
  if (flag = 0) then
    write(l, String'("---------------------------------------- "));
    writeline(output, l);
    write(l, String'("  xi = ")); write(l, real'(Conv2real(x)));
    write(l, String'("  yi = ")); write(l, real'(Conv2real(y)));
    write(l, String'("  zi = ")); write(l, real'(Conv2real(z)));
  elsif (flag = 1) then
    write(l, String'("  xo = ")); write(l, real'(Conv2real(x)));
    write(l, String'("  yo = ")); write(l, real'(Conv2real(y)));
    write(l, String'("  zo = ")); write(l, real'(Conv2real(z)));
  else
```

```
      write(l, String'("  xn = ")); write(l, real'(Conv2real(x)));
      write(l, String'("  yn = ")); write(l, real'(Conv2real(y)));
      write(l, String'("  zn = ")); write(l, real'(Conv2real(z)));
    end if;
    writeline(output, l);
  end DispReg;
  ------------------------------------------------------------------------------

  ------------------------------------------------------------------------------
  procedure DispAng (angle : in std_logic_vector (31 downto 0)) is
  ------------------------------------------------------------------------------
    variable l : line;
  begin
    write(l, String'("  angle = ")); write(l, real'(Conv2real(angle)));
    writeline(output, l);
    write(l, String'("..................................... "));
    writeline(output, l);
  end DispAng;


end cordic_pkg;
::::::::::::::
c1.adder.vhdl
::::::::::::::
------------------------------------------------------------------------------
--
--  Purpose:
--
--    Ripple Carry Adder
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
--
--    Output:
```

```
--------------------------------------------------------------------------------

library STD;
use STD.textio.all;

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


entity adder is
  generic (
    WD     : in natural := 32;
    BD     : in natural := 4 );

  port (
    an     : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    bn     : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    ci     : in   std_logic := '0';
    cn     : out  std_logic_vector (WD-1 downto 0) := (others=>'0');
    co     : out  std_logic := '0');

end adder;


::::::::::::::
c1.adder.rca.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    Ripple Carry Adder
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
```

```
--
--    Input:
--
--    Output:
-------------------------------------------------------------------------------

library STD;
use STD.textio.all;

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




architecture rca of adder is
begin
  process (an, bn, ci)
    variable sn : std_logic_vector (WD-1 downto 0) := (others=>'0');
    variable c  : std_logic := '0';
  begin  -- process
    c := ci;
    for i in 0 to WD-1 loop
      sn(i) := an(i) xor bn(i) xor c;
      c := (an(i) and bn(i)) or (an(i) and c) or (bn(i) and c);
    end loop;  -- i

    cn <= sn;
    co <= c;
  end process;

end rca;



::::::::::::::
c1.adder.cca.gprom.vhdl
::::::::::::::
-------------------------------------------------------------------------------
--
--  Purpose:
--
--    GP Logic of a Carry Chain Adder
--
--  Discussion:
--
--
--  Licensing:
--
```

```
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.11.06
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input: an, bn : BD-bits
--
--    Output: g, p : 1-bit
--------------------------------------------------------------------------------

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 gprom is
  generic (
    BD     : in natural := 4);

  port (
    an    : in   std_logic_vector (BD-1 downto 0) := (others=>'0');
    bn    : in   std_logic_vector (BD-1 downto 0) := (others=>'0');
    en    : in   std_logic := '0';
    g     : out  std_logic := '0';
    p     : out  std_logic := '0');
end gprom;




--------------------------------------------------------------------
architecture rtl of gprom is

begin
  ------------------------------------------------------------------------
  -- Computing Carry Chain GP Logic
  -- i-th BD-bit adder
```

```
  -- g : carry generation  : an + bn > BD-1
  -- p : carry propagation : an + bn = BD-1
  ------------------------------------------------------------------------------
  -- TBD:  LUT implementation --> Study Hauck, Hosler, Fry Paper
  ------------------------------------------------------------------------------

  ------------------------------------------------------------------------
  process (an, bn)
    constant max_addr : integer := 2**(2*BD) -1;
    constant max_half : integer := 2**BD -1;
    type rom_type is array (0 to max_addr) of std_logic;

    ----------------------------------------
    function init_g return rom_type is
    ----------------------------------------
      variable g : rom_type;
    begin
      for i in 0 to max_half loop
        for j in 0 to max_half loop
          if ((i+j) > (2**BD -1)) then
            g(i*(2**BD) + j) := '1';
          else
            g(i*(2**BD) + j) := '0';
          end if;
        end loop;  -- j
      end loop;  -- i
      return g;
    end;
    ----------------------------------------

    constant rom_g : rom_type := init_g;

    variable addr : std_logic_vector (2*BD -1 downto 0) := (others=>'0');
  begin

    if (en = '1') then
      addr := an & bn;
      g <= rom_g(to_integer(unsigned(addr)));
    end if;

  end process;


  ------------------------------------------------------------------------
  process (an, bn)
    constant max_addr : integer := 2**(2*BD) -1;
    constant max_half : integer := 2**BD -1;
    type rom_type is array (0 to max_addr) of std_logic;

    ----------------------------------------
```

```
    function init_p return rom_type is
    ----------------------------------------
      variable p : rom_type;
    begin
      for i in 0 to max_half loop
        for j in 0 to max_half loop
          if ((i+j) = (2**BD -1)) then
            p(i*(2**BD) + j) := '1';
          else
            p(i*(2**BD) + j) := '0';
          end if;
        end loop;  -- j
      end loop;  -- i
      return p;
    end;
    ----------------------------------------

    constant rom_p : rom_type := init_p;

    variable addr : std_logic_vector (2*BD -1 downto 0) := (others=>'0');
  begin

    if (en = '1') then
      addr := an & bn;
      p <= rom_p(to_integer(unsigned(addr)));
    end if;
  end process;


end rtl;



::::::::::::::
c1.adder.cca.subadd.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    Ripple Carry Adder
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
```

```
--
--    2013.11.13
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
--
--    Output:
--------------------------------------------------------------------------------

library STD;
use STD.textio.all;

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


entity subadder is
  generic (
    WD     : in natural := 32;
    BD     : in natural := 4 );

  port (
    an     : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    bn     : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    ci     : in   std_logic := '0';
    cn     : out  std_logic_vector (WD-1 downto 0) := (others=>'0');
    co     : out  std_logic := '0');

end subadder;


architecture rca of subadder is

  component add is
  generic (
    WD     : in natural := 32;
    BD     : in natural := 4 );

    port (
      an    : in   std_logic_vector (WD-1 downto 0);
      bn    : in   std_logic_vector (WD-1 downto 0);
      ci    : in   std_logic := '0';
      cn    : out  std_logic_vector (WD-1 downto 0);
      co    : out  std_logic := '0');
```

```
  end component;

  for U0: add use entity work.adder(rca);

begin

        U0: add
                    generic map (WD => BD, BD => BD)
                    port map (an => an,
                              bn => bn,
                              ci => ci,
                              cn => cn,
                              co => co );



end rca;




::::::::::::::
c1.adder.cca.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    Carry Chain Adder
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.11.06
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input: an, bn : WD-bits,  ci : 1-bit
--
--    Output: cn : WD-bits, co : 1-bit
```

```
--------------------------------------------------------------------------------

library STD;
use STD.textio.all;

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

use WORK.cordic_pkg.all;




--------------------------------------------------------------------------
-- an : 1st operand (WD-bit)
-- bn : 2nd operand (WD-bit)
-- ci : carry in (1-bit)
-- cn : result (WD-bit)
-- co : carry out (1-bit)
--------------------------------------------------------------------------


architecture cca of adder is

  component subadder is
  generic (
    WD     : in natural := 32;
    BD     : in natural := 4 );

    port (
      an    : in   std_logic_vector (WD-1 downto 0);
      bn    : in   std_logic_vector (WD-1 downto 0);
      ci    : in   std_logic := '0';
      cn    : out  std_logic_vector (WD-1 downto 0);
      co    : out  std_logic := '0');
  end component;


  component gprom is
  generic (
    BD     : in natural := 4 );

    port (
      an    : in   std_logic_vector (BD-1 downto 0);
      bn    : in   std_logic_vector (BD-1 downto 0);
      en    : in   std_logic := '0';
      g     : out  std_logic := '0';
      p     : out  std_logic := '0');
```

```
  end component;


  constant ND : natural := WD/BD;

  ----------------------------------------------------------------------
  -- an2d, bn2d, cn2d : array(ND, BD) <= an, bn, cn
  -- ci1d, co1d       : array(ND)     <= ci, co
  -- g1d, p1d         : array(ND)     -- Generate, Propagate
  -- qi1d, qo1d       : array(ND)     -- Carry ChainIn, CarryChainOut
  ----------------------------------------------------------------------
  type array2d is array (ND-1 downto 0) of std_logic_vector (BD-1 downto 0);
  signal an2d, bn2d, cn2d: array2d := ((others=> (others=> '0')));


  type array1d is array (ND-1 downto 0) of  std_logic;
  signal ci1d, co1d : array1d := (others=> '0');
  signal qi1d, qo1d : array1d := (others=> '0');
  signal g1d,  p1d : array1d := (others=> '0');


  procedure ToA2d
    (signal a : in std_logic_vector (WD-1 downto 0);
     signal a2d : out array2d ) is
    variable tmp2d: array2d  := ((others=> (others=> '0')));
    variable tmpv : std_logic_vector (WD-1 downto 0) := (others=>'0');
  begin
    tmpv := a;

    for i in ND-1 downto 0 loop
        tmp2d(i) := tmpv((i+1)*BD-1 downto i*BD);
        a2d(i) <= tmp2d(i);
    end loop;

  end ToA2d;


  procedure FromA2d
    (signal a2d : in  array2d;
     signal a : out std_logic_vector (WD-1 downto 0) ) is
    variable tmp2d: array2d  := ((others=> (others=> '0')));
    variable tmpv : std_logic_vector (WD-1 downto 0) := (others=>'0');
  begin
    tmp2d := a2d;

    for i in ND-1 downto 0 loop
        tmpv((i+1)*BD-1 downto i*BD) := tmp2d(i);
    end loop;

    a <= tmpv;
```

```
  end FromA2d;


begin

  ------------------------------------------------------------------------
  -- ND Adders of BD-bit
  ------------------------------------------------------------------------
  -- ci1d(i)    : cin's of the i-th BD-bit adder
  -- co1d(i)    : cout's of the i-th BD-bit adder
  -- cn2d(i, j) : j-th bit of the result of the i-th BD-bit adder
  ------------------------------------------------------------------------
  ILOOP: for i in ND-1 downto 0 generate
        U0:subadder generic map (WD => BD, BD => BD)
                    port map (an => an2d(i),
                              bn => bn2d(i),
                              ci => qi1d(i),
                              cn => cn2d(i),
                              co => co1d(i) );
  end generate ILOOP;


  ------------------------------------------------------------------------
  -- ND gproms
  ------------------------------------------------------------------------
  -- Carry Chain GP Logic
  -- j-th gprom with inputs of BD-bit an and bn
  -- g1d(j) : carry generation  : an + bn > BD-1
  -- p1d(j) : carry propagation : an + bn = BD-1
  ------------------------------------------------------------------------
  -- TBD:  LUT implementation --> Study Hauck, Hosler, Fry Paper
  ------------------------------------------------------------------------
  JLOOP: for j in ND-1 downto 0 generate
        U1:gprom generic map (BD => BD)
                    port map (an => an2d(j),
                              bn => bn2d(j),
                              en => '1',
                              g  => g1d(j),
                              p  => p1d(j) );
  end generate JLOOP;


  ------------------------------------------------------------------------
  -- an2d <= an
  -- bn2d <= bn
  -- cn <= cn2d
  ------------------------------------------------------------------------
  ToA2d(an, an2d);
  ToA2d(bn, bn2d);
```

```
  FromA2d(cn2d, cn);



  ------------------------------------------------------------------------
  -- co, qo1d <= Carry Chain Cell <= qi1d, ci
  -- qi1d(i) : input of a carry chain cell
  -- qo1d(i) : output of a carry chain cell
  ------------------------------------------------------------------------
  process (ci, qo1d)
    variable tmp1d : array1d := (others=> '0');
    variable tmp : std_logic := '0';
  begin
    tmp   := ci;
    tmp1d := qo1d;

    for i in ND-1 downto 1 loop
        qi1d(i) <= qo1d(i-1);
    end loop;

    qi1d(0) <= tmp;
    co <= qo1d(ND-1);
  end process;


  process (p1d, g1d, qi1d)
    variable tmp1d_p, tmp1d_g, tmp1d_qi : array1d := (others=> '0');
  begin

    tmp1d_p  := p1d;
    tmp1d_g  := g1d;
    tmp1d_qi := qi1d;

    for i in ND-1 downto 0 loop
        if (tmp1d_p(i) = '1') then
           qo1d(i) <= tmp1d_qi(i);
        else
           qo1d(i) <= tmp1d_g(i);
        end if;
    end loop;

  end process;



end cca;
```

```
::::::::::::::
c2.addsub.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    Add / Sub
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
--
--    Output:
--------------------------------------------------------------------------------

library STD;
use STD.textio.all;

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


entity addsub is
  generic (
    WD     : in natural := 32);

  port (
    an    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    bn    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    s     : in   std_logic := '0';
    cn    : out  std_logic_vector (WD-1 downto 0) := (others=>'0');
    co    : out  std_logic := '0');
end addsub;
```

```
architecture rtl of addsub is

  component adder
    generic (
      WD     : in natural );
    port (
      an     : in   std_logic_vector (WD-1 downto 0);
      bn     : in   std_logic_vector (WD-1 downto 0);
      ci     : in   std_logic := '0';
      cn     : out  std_logic_vector (WD-1 downto 0);
      co     : out  std_logic := '0');
  end component;

  signal un : std_logic_vector (WD-1 downto 0) := (others=>'0');
begin

  process (bn, s)
  begin  -- process
    if (s='1') then
      un <= not bn;
    else
      un <= bn;
    end if;
  end process;

  A0 : adder generic map (WD => WD)
      port map (an => an, bn => un, ci => s, cn => cn, co => co);

end rtl;




::::::::::::::
c3.bshift.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    Barrel Shifter
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
```

```
--
--  Modified:
--
--    2013.10.23
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
--
--    Output:
--------------------------------------------------------------------------------

library STD;
use STD.textio.all;

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


entity bshift is
  generic (
    WD     : in natural := 32;
    SH     : in natural := 5 );

  port (
    di   : in  std_logic_vector (WD-1 downto 0) := (others=>'0');
    nbit : in  std_logic_vector (SH-1 downto 0) := (others=>'0');
    cs   : in  std_logic ;
    dq   : out std_logic_vector (WD-1 downto 0) := (others=>'0'));

end bshift;


::::::::::::::
c3.bshift.mux.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    Barrel Shifter Based on Mux
--
--  Discussion:
--
--
```

```
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2013.10.23
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
--
--    Output:
--------------------------------------------------------------------------------

library STD;
use STD.textio.all;

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



architecture mux of bshift is

  component mux is
    generic (
      WD     : in natural := 1);

    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) );
  end component;


  type array2d is array (WD-1 downto 0, SH-1 downto 0) of std_logic;
  signal mout: array2d  := ((others=> (others=> '0')));
  signal m_in: array2d  := ((others=> (others=> '0')));

begin

  -- j = SH-1 : top level mux row
  ILOOP: for i in WD-1 downto 0 generate
```

```
    -- Sign bit extension
    ZERO: if ((WD-1-i) < 2**(SH-1)) generate
      U0: mux generic map (WD => 1)
              port map (an(0) => di(i),
                        bn(0) => di(WD-1), -- '0', -- di(WD-1),
                        s  => nbit(SH-1),
                        cn(0) => mout(i, SH-1));
    end generate ZERO;


    -- stride 2**(SH-1)
    REST: if ((WD-1-i) >= 2**(SH-1)) generate
      U1 : mux generic map (WD => 1)
               port map (an(0) => di(i),
                         bn(0) => di(i+2**(SH-1)),
                         s  => nbit(SH-1),
                         cn(0) => mout(i, SH-1));
    end generate REST;


  end generate ILOOP;


  -- rest of mux rows
  JLOOP: for j in SH-2 downto 0 generate
    ILOOP: for i in WD-1 downto 0 generate
      -- Sign bit extension
      ZERO: if ((WD-1-i) < 2**j)  generate
        U0: mux generic map (WD => 1)
                port map (an(0) => m_in(i, j+1),
                          bn(0) => m_in(WD-1, j+1), -- '0', -- m_in(WD-1, j+1),
                          s  => nbit(j),
                          cn(0) => mout(i, j));
      end generate ZERO;


      -- Stride 2**j
      REST: if ((WD-1-i) >= 2**j) generate
        U1: mux port map (an(0) => m_in(i, j+1),
                          bn(0) => m_in(i+2**j, j+1),
                          s  => nbit(j),
                          cn(0) => mout(i, j));
      end generate REST;


    end generate ILOOP;
  end generate JLOOP;
```

```
  process (mout)
    variable tmp: array2d  := ((others=> (others=> '0')));
  begin

    tmp := mout;

    m_in <= tmp;

    for i in WD-1 downto 0 loop
        -- dq(i) <= tmp(i, 0);
      dq(i) <= tmp(i, 0) and cs;
    end loop;

    -- wait on mout; --, di, nbit;

  end process;



end mux;
::::::::::::::
c4.dff.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    D FlipFlop
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
--
--    Output:
--------------------------------------------------------------------------------
```

```vhdl
library STD;
use STD.textio.all;

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




entity dff is
  generic (
    WD     : in natural := 32);

  port (
    clk    : in   std_logic := '0';
    rst    : in   std_logic := '0';
    en     : in   std_logic := '0';
    di     : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    dq     : out  std_logic_vector (WD-1 downto 0) := (others=>'0') );

end dff;



architecture rtl of dff is
begin

  Reg: process (clk, rst)
  begin  -- process Reg
    if rst = '0' then                   -- asynchronous reset (active low)
      dq <= (others=>'0');
    elsif clk'event and clk = '1' then  -- rising clock edge
      if (en='1') then
        dq <= di;
      end if;
    end if;
  end process Reg;

end rtl;



-- entity dff1 is
--   generic (
--     WD     : in natural := 32);
--
--   port (
```

```
--     clk    : in   std_logic := '0';
--     rst    : in   std_logic := '0';
--     di     : in   std_logic;
--     dq     : out  std_logic);

-- end dff1;



-- architecture rtl of dff1 is
-- begin

--   Reg: process (clk, rst)
--   begin  -- process Reg
--     if rst = '0' then                   -- asynchronous reset (active low)
--       dq <= (others=>'0');
--     elsif clk'event and clk = '1' then  -- rising clock edge
--       dq <= di;
--     end if;
--   end process Reg;

-- end rtl;


::::::::::::::
c5.counter.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    Synchronous Counter
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
```

```
--    Input:
--
--    Output:
--------------------------------------------------------------------------------

library STD;
use STD.textio.all;

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


entity counter is
  generic (
    SH     : in natural :=  5 );

  port (
    clk    : in   std_logic := '0';
    rst    : in   std_logic := '0';
    en     : in   std_logic := '0';
    ld     : in   std_logic := '0';
    di     : in   std_logic_vector (SH-1 downto 0) := (others=>'0');
    dq     : out  std_logic_vector (SH-1 downto 0) := (others=>'0') );

end counter;


architecture rtl of counter is

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

    port (
      an    : in   std_logic_vector (SH-1 downto 0);
      bn    : in   std_logic_vector (SH-1 downto 0);
      ci    : in   std_logic;
      cn    : out  std_logic_vector (SH-1 downto 0);
      co    : out  std_logic );
  end component;

  signal cnt    : std_logic_vector (SH-1 downto 0);
  signal cntInc : std_logic_vector (SH-1 downto 0);
  signal tbd : std_logic;

begin

  inc : adder generic map (WD => SH)
    port map (an=>cnt, bn=>(others=>'0'), ci=>'1', cn=>cntInc, co=>tbd);
```

```
  Reg: process (clk, rst)
  begin  -- process Reg
    if rst = '0' then                   -- asynchronous reset (active low)
      cnt <= (others=>'0');
    elsif clk'event and clk = '1' then  -- rising clock edge
      if (ld='1') then
        cnt <= di;
      elsif (en='1') then
        cnt <= cntInc;
      end if;
    end if;
  end process Reg;

  dq <= cnt;

end rtl;
::::::::::::::
c6.rom.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    ROM Model Entity
--
--    Architectures:
--      fint_arch       ( c6.rom.fint.vhdl      )
--      fint_file_arch  ( c6.rom.fint_file.vhdl )
--      real_arch       ( c6.rom.real.vhdl      )
--      real_file_arch  ( c6.rom.real_file.vhdl )
--
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2013.10.14
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
```

```
--    Input: "angle_real.dat"
--               WD  := 32 -- data bus width
--               SH  :=  6 -- addr bus width
--               PWR := 64 -- address space (PWR = 2**SH)
--               addr(SH-1:0)
--               cs
--    Output:   data(WD-1:0)
-- ----------------------------------------------------------------------------

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 rom is
  generic (
    WD     : in natural := 32;  -- data bus width
    SH     : in natural :=  6;  -- addr bus width
    PWR    : in natural := 64); -- address space (PWR = 2**SH)

  port (
    addr   : in   std_logic_vector (SH-1 downto 0) := (others=>'0');
    data   : out  std_logic_vector (WD-1 downto 0) := (others=>'0');
    cs     : in   std_logic := '0' );

end rom;


::::::::::::::
c6.rom.fint.vhdl
::::::::::::::
-------------------------------------------------------------------------------
--
--  Purpose:
--
--    ROM Model Architecture
--    (integer type data is embedded in this file as a constant)
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
```

```
--  Modified:
--
--    2013.10.14
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--    MyWork/5.cordic_vhdl/f.c/lut_conv.c
--    lut_real.dat => lut_fint.dat (bits_frac=29)
--    lut_fint embedded in iarray
--
--    Input:
--
--    Output:   data(WD-1:0)
-- ------------------------------------------------------------------------

library STD;
use STD.textio.all;

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

use WORK.cordic_pkg.all;



------------------------------------------------------------------------
--   iarray (iarray valued array - iarray type)
--   type iarray is array (natural range <>) of iarray;
--        constant angles : iarray ------------------------------------------------------------------------------------
--   darray (discrete value array - WD-bit std_logic_vector)
--   type darray is array (0 to PWR-1) of std_logic_vector (WD-1 downto 0);
--        variable romData : darray
------------------------------------------------------------------------
--   angles(i) => romData(i)
--       romData(i) := Conv2fixedPt(angles(i), WD);
------------------------------------------------------------------------
--       data <= romData(to_integer(unsigned(addr)));
------------------------------------------------------------------------
architecture fint_arch of rom is
   type iarray is array (natural range <>) of integer;

   constant angles : iarray :=
                     (
                        421657428 ,
                        248918914 ,
                        131521918 ,
```

```
66762579 ,
33510843 ,
16771757 ,
8387925 ,
4194218 ,
2097141 ,
1048574 ,
524287 ,
262143 ,
131071 ,
65535 ,
32767 ,
16383 ,
8191 ,
4095 ,
2047 ,
1023 ,
511 ,
255 ,
127 ,
63 ,
31 ,
15 ,
7 ,
4 ,
2 ,
1 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
0 ,
```

```
                        0 ,
                        0 ,
                        0 ,
                        0 ,
                        0 ,
                        0 ,
                        0 ,
                        0 ,
                        0 ,
                        0 ,
                        0
                      );


begin

  ROM: process (addr, cs)
    type darray is array (0 to PWR-1) of std_logic_vector (WD-1 downto 0);
    variable romData : darray;
    variable initRom : boolean := false;

  begin  -- process Reg
    if (initRom=false) then
      for i in 0 to PWR-1 loop
        romData(i) := std_logic_vector(to_signed(angles(i), 32));
      end loop;  -- i
      initRom := true;
    end if;

   if (initRom=true) then
     if cs = '1' then
       data <= romData(to_integer(unsigned(addr)));
     else
       data <= (others=>'1');
     end if;
   end if;
 end process ROM;

end fint_arch;
::::::::::::::
c7.mux.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    Mux
--
--  Discussion:
--
```

```
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
--
--    Output:
--------------------------------------------------------------------------------

library STD;
use STD.textio.all;

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


entity mux is
  generic (
    WD     : in natural := 32);

  port (
    an    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    bn    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    s     : in   std_logic := '0';
    cn    : out  std_logic_vector (WD-1 downto 0) := (others=>'0') );
end mux;


architecture rtl of mux is
begin

  process (an, bn, s)
  begin  -- process
    if (s='1') then
      cn <= bn;
    else
      cn <= an;
    end if;
```

```
  end process;

end rtl;




::::::::::::::
m1.disp.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    Monitors signals and writes their values
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
--
--    Output:
--------------------------------------------------------------------------------

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 disp is
```

```
  generic (
    WD     : in natural := 32;
    SH     : in natural :=  5;
    PWR    : in natural := 64);

  port (
    clk   : in   std_logic := '0';
    load  : in   std_logic := '0';
    cntEn : in   std_logic := '0';
    ready : in   std_logic := '0';
    xi    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    yi    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    zi    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    xn    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    yn    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    zn    : in   std_logic_vector (WD-1 downto 0) := (others=>'0');
    angle : in   std_logic_vector (WD-1 downto 0) := (others=>'0') );

end disp;

architecture beh of disp is

  signal dinInc : std_logic_vector (SH-1 downto 0);

begin

  monitor: process
  begin  -- process Reg

    wait until (clk'event and clk = '0');

    if (load='1') then
      DispReg(xn, yn, zn, 0);
      DispAng(angle);
    elsif (ready='1') then
      DispReg(xn, yn, zn, 1);
      DispAng(angle);
    elsif (cntEn='1') then
      DispReg(xn, yn, zn, 2);
      DispAng(angle);
    end if;


  end process monitor;

end beh;
::::::::::::::
cordic_rtl.vhdl
::::::::::::::
--------------------------------------------------------------------------------
```

```
--
--  Purpose:
--
--    behavioral model of cordic
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2013.09.20
--
--  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;
    nIter       : 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);
      cs       : in   std_logic ;
      dq       : 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 ;
      en   : 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 (
      SH     : in natural :=  5 );

    port (
      clk  : in   std_logic := '0';
      rst  : in   std_logic := '0';
      en   : in   std_logic := '0';
      ld   : in   std_logic := '0';
      di   : in   std_logic_vector (4 downto 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 mux 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) );
end component;


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

  port (
    clk   : in   std_logic := '0';
    load  : in   std_logic := '0';
    cntEn : in   std_logic := '0';
    ready : in   std_logic := '0';
    xi    : in   std_logic_vector (WD-1 downto 0);
    yi    : in   std_logic_vector (WD-1 downto 0);
    zi    : in   std_logic_vector (WD-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 xt, yt, zt : std_logic_vector(WD-1 downto 0) := (others=>'0');
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');

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

signal cntEn : std_logic := '0';
signal endIter  : std_logic := '0';
signal dffEn : std_logic := '0';
```

```
begin


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

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

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


  MuxX: mux generic map (WD=>32)
    port map (an=>xr, bn=>xi, s=>load, cn=>xt);

  MuxY: mux generic map (WD=>32)
    port map (an=>yr, bn=>yi, s=>load, cn=>yt);

  Muxz: mux generic map (WD=>32)
    port map (an=>zr, bn=>zi, s=>load, cn=>zt);


  dffEn <= (cntEn and (not endIter)) or load;

  RegX: dff generic map (WD=>32)
    port map (clk=>clk, rst=>rst, en=>dffEn, di=>xt, dq=>xn);
```

```
  RegY: dff generic map (WD=>32)
    port map (clk=>clk, rst=>rst, en=>dffEn, di=>yt, dq=>yn);

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



  -- EnReg: dff generic map (WD=>1)
  --   port map (clk=>clk, rst=>rst, di(0)=>preEn, dq(0)=>cntEn);

  EnReg: process (clk, rst)
  begin  -- process EnReg
    if rst = '0' then
      cntEn <= '0';
    elsif clk'event and clk = '1' then
      if (load='1') then
        cntEn <= '1';
      elsif (endIter='1') then
        cntEn <= '0';
      end if;
    end if;
  end process EnReg;


  endIter <= '1' when cnt=nIter else '0';
  ready <= endIter;




  CntReg: counter generic map (SH=>5)
    port map (clk=>clk, rst=>rst, en=>cntEn, ld=>endIter, di=>"00000", dq=>cnt);


  AngRom: rom generic map (WD=>32, SH=>5, PWR=>64)
    port map (addr=>cnt, cs=> rst, data=>angle);


  Monitor: disp generic map (WD=>32, SH=>5)
    port map (clk=>clk, load=>load, cntEn=>cntEn, ready=>endIter,
              xi=>xi, yi=>yi, zi=>zi,
              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);

    --XXXXXXX XXXXXX XXXXXX XXXXXX XXXXXXX XXXXXX XXXXX

end beh;
::::::::::::::
cordic_tb.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    testbench of cordic
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
--
--
--    Output:
--------------------------------------------------------------------------------

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_tb is
end cordic_tb;




architecture beh of cordic_tb 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 4* half_period;


   process
   begin

      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(1.0/K, nBit);
   yi <= Conv2fixedPt(0.0, nBit);
   zi <= Conv2fixedPt(0.0, 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);

--------------------------------------------------------------------------
-- printf ("\nGrinding on [K, 0, pi/6] -> [0.86602540, 0.50000000, 0]\n");
-- Circular (X0C, 0L, HalfPi / 3L);
--------------------------------------------------------------------------
   for i in 0 to 4  loop
     wait until clk = '1';
   end loop;  -- i

   xi <= Conv2fixedPt(1.0/K, nBit);
   yi <= Conv2fixedPt(0.0, nBit);
   zi <= Conv2fixedPt(pi/6.0, nBit);
   wait for 1 ns;
   load <= '1', '0' after clk_period;
   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);

--------------------------------------------------------------------------
-- printf ("\nGrinding on [K, 0, pi/4] -> [0.70710678, 0.70710678, 0]\n");
-- Circular (X0C, 0L, HalfPi / 2L);
--------------------------------------------------------------------------
   for i in 0 to 4  loop
     wait until clk = '1';
   end loop;  -- i

   xi <= Conv2fixedPt(1.0/K, nBit);
   yi <= Conv2fixedPt(0.0, nBit);
   zi <= Conv2fixedPt(pi/4.0, nBit);
```

```
      wait for 1 ns;
      load <= '1', '0' after clk_period;
      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);

   ------------------------------------------------------------------------
   -- printf ("\nGrinding on [K, 0, pi/3] -> [0.50000000, 0.86602540, 0]\n");
   -- Circular (X0C, 0L, 2L * (HalfPi / 3L));
   ------------------------------------------------------------------------
      for i in 0 to 4  loop
        wait until clk = '1';
      end loop;  -- i

      xi <= Conv2fixedPt(1.0/K, nBit);
      yi <= Conv2fixedPt(0.0, nBit);
      zi <= Conv2fixedPt(pi/3.0, nBit);
      wait for 1 ns;
      load <= '1', '0' after clk_period;
      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);

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


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

  --   XXXXXXX XXXXXX XXXXXX XXXXXX XXXXXXX XXXXXX XXXXX

end beh;
```