```
anal : c1.adder.vhdl c1.adder.rca.vhdl c2.addsub.vhdl c3.bshift.vhdl \
        c4.dff.vhdl c5.counter.vhdl c6.rom.vhdl c7.mux.vhdl m1.disp.vhdl \
        cordic_pkg.vhdl cordic_rtl.vhdl cordic_tb.vhdl
        ghdl -a cordic_pkg.vhdl
        ghdl -a c1.adder.vhdl
        ghdl -a c1.adder.rca.vhdl
        ghdl -a c2.addsub.vhdl
        ghdl -a c3.bshift.vhdl
        ghdl -a c4.dff.vhdl
        ghdl -a c5.counter.vhdl
        ghdl -a c6.rom.vhdl
        ghdl -a c7.mux.vhdl
        ghdl -a m1.disp.vhdl
        ghdl -a cordic_rtl.vhdl
        ghdl -a cordic_tb.vhdl

elab : cordic_pkg.o \
        c1.adder.rca.o c2.addsub.o c3.bshift.o c4.dff.o \
        c5.counter.o c6.rom.o c7.mux.o m1.disp.o \
        cordic_rtl.o cordic_tb.o
        ghdl -e cordic_tb

run : cordic_pkg.o cordic_rtl.o cordic_tb.o
        ghdl -r cordic_tb --vcd=cordic.vcd

all : anal elab run

wave :
        gtkwave cordic.vcd &

cordic_files :
        more c1.adder.rca.vhdl  \
             c2.addsub.vhdl     \
             c3.bshift.vhdl     \
             c4.dff.vhdl        \
             c5.counter.vhdl    \
             c6.rom.vhdl        \
             c7.mux.vhdl        \
             m1.disp.vhdl       \
             cordic_pkg.vhdl    \
             cordic_rtl.vhdl    \
             cordic_tb.vhdl     > file/cordic.rtl.files
        tar cvf file/cordic.rtl.tar *.vhdl makefile



bshift : c3.bshift.mux.vhdl bshift_tb.vhdl cordic_pkg.vhdl
        ghdl -a cordic_pkg.vhdl
        ghdl -a c7.mux.vhdl
        ghdl -a c3.bshift.mux.vhdl
        ghdl -a bshift_tb.vhdl
        ghdl -e bshift_tb
        ghdl -r bshift_tb --vcd=bshift.vcd
        # gtkwave bshift.vcd &

#-------------------------------------------------------------------------
SRC_adder = cordic_pkg.vhdl c1.adder.vhdl c1.adder.rca.vhdl \
            c1.adder.cca.vhdl c1.adder.cca.gprom.vhdl \
            c1.adder_tb.vhdl c1.adder_tb_cca.vhdl c1.adder_tb_rca.vhdl
EXE_adder = adder_tb adder_tb_cca adder_tb_rca
adder : ${SRC_adder}
        ghdl -a cordic_pkg.vhdl
        ghdl -a c1.adder.vhdl
        ghdl -a c1.adder.rca.vhdl
        ghdl -a c1.adder.cca.gprom.vhdl
```

```
        ghdl -a c1.adder.cca.vhdl
        ghdl -a c1.adder_tb.vhdl

        \rm -f adder_tb adder_tb_cca adder_tb_rca


adder_tb:
        ghdl -e adder_tb
        ghdl -r adder_tb --disp-tree=inst --vcd=adder.vcd --stop-time=1us

        gtkwave adder.vcd &


adder_tb_cca:
        ghdl -a c1.adder_tb_cca.vhdl
        ghdl -e adder_tb_cca
        ghdl -r adder_tb_cca --disp-tree=inst --vcd=adder.vcd --stop-time=1us

        gtkwave adder.vcd &

adder_tb_rca:
        ghdl -a c1.adder_tb_rca.vhdl
        ghdl -e adder_tb_rca
        ghdl -r adder_tb_rca --disp-tree=inst --vcd=adder.vcd --stop-time=1us

        gtkwave adder.vcd &

adder_files :
        more ${SRC_adder} makefile > file/adder.files
        tar cvf file/adder.rtl.tar ${SRC_adder} makefile

#-------------------------------------------------------------------------
SRC_rom = cordic_pkg.vhdl c6.rom.vhdl c6.rom.rfile.vhdl \
            c6.rom_tb.vhdl c6.rom_tb_rfile.vhdl
EXE_rom = rom_tb rom_tb_rfile
rom : ${SRC_rom}
        ghdl -a cordic_pkg.vhdl
        ghdl -a c6.rom.vhdl
        ghdl -a c6.rom.rfile.vhdl
        ghdl -a c6.rom_tb.vhdl

        \rm -f EXE_rom

rom_tb_rfile :
        ghdl -a c6.rom_tb_rfile.vhdl
        ghdl -e rom_tb_rfile
        ghdl -r rom_tb_rfile --disp-tree=inst --vcd=rom.vcd --stop-time=1us

        gtkwave rom.vcd &

rom_files :
        more ${SRC_rom} makefile > file/rom.files
        tar cvf file/rom.rtl.tar ${SRC_rom} makefile


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
```

```
#-------------------------------------------------------------------------
clean :
        \rm -f *.o *~ *# *.cf
        \rm -f *_tb
        \rm -f *_conf
        \rm -f *.vcd
        \rm -f ${EXE_adder} ${EXE_rom} ${EXE_angle}


allfiles :
        more makefile *.vhdl > file/all.rtl.files
        tar cvf file/all.rtl.tar *.vhdl makefile




print_all :
        more makefile  >  print.all
        more *.vhdl    >> print.all
        more *.c *.dat >> print.all


tar :
        mkdir src
        cp makefile *.vhdl *.c *.dat src
        tar cvf 5.cordic_vhdl_rtl.tar src
        \rm -fr src
::::::::::::::
bshift_tb.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    testbench of bshfit
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.07.27
--
--  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;
```

```
use WORK.all;


entity bshift_tb is
end bshift_tb;




architecture beh of bshift_tb is

  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);
      dq       : out std_logic_vector (WD-1 downto 0) );
  end component;


  for bshift_0: bshift use entity work.bshift(mux);



  constant nBit : integer := 32;

  signal clk, rst: std_logic := '0';
  signal di      : std_logic_vector(31 downto 0) := X"7FFF_FFFF";
  signal dq      : std_logic_vector(31 downto 0) := X"0000_0000";
  signal cnt     : std_logic_vector( 4 downto 0) := "00000";

 begin

   bshift_0 : bshift generic map (WD=>32, SH=>5)
     port map (di  => di, nbit => cnt, dq => dq);


   clk <= not clk after half_period;

   rst <= '0', '1' after 2* half_period;


   process
   begin

      wait until rst = '1';

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

      -- di <= X"7FFF_FFFF";
      di <= X"8000_0000";
      wait for 0 ns;

      for i in 0 to 31  loop
        wait until (clk'event and clk='1');

        cnt <= std_logic_vector(to_unsigned(i, 5));

        wait for 0 ns;

      end loop;
```

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


   process
   begin
     wait for 100* clk_period;
     assert false report "end of simulation" severity failure;
   end process;

  --   XXXXXXX XXXXXX XXXXXX XXXXXX XXXXXXX XXXXXX XXXXX

end beh;
::::::::::::::
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.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;



::::::::::::::
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_tb_cca.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    configuration of cca adder testbench
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
```

```
--  Modified:
--
--    2012.10.26
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
--
--
--    Output:
--------------------------------------------------------------------------------

use WORK.all;


configuration adder_tb_cca of adder_tb is
  for beh
    for DUT: adder
      use entity work.adder(cca) ;
      for cca
        for ILOOP
          for U0:subadder
            use entity work.adder(rca);
          end for;
        end for;
        for JLOOP
          for U1:gprom
            use entity work.gprom(rtl);
          end for;
        end for;
      end for;
    end for;
  end for;
end adder_tb_cca;


::::::::::::::
c1.adder_tb_rca.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    configuration of rca adder testbench
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.10.26
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
```

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

use WORK.all;


configuration adder_tb_rca of adder_tb is
  for beh
    for DUT: adder
      use entity work.adder(rca) ;
    end for;
  end for;
end adder_tb_rca;
::::::::::::::
c1.adder_tb.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    testbench of adder
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.10.25
--
--  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;
use WORK.all;


entity adder_tb is
end adder_tb;



architecture beh of adder_tb is

  component adder
```

```
   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 component;

 -- for DUT: adder use configuration work.adder_cca;
 -- for DUT: adder use entity work.adder(rca);

 signal clk, rst: std_logic := '0';
 signal an      : std_logic_vector(31 downto 0) := X"0000_0000";
 signal bn      : std_logic_vector(31 downto 0) := X"0000_0001";
 signal ci      : std_logic := '0';
 signal cn      : std_logic_vector(31 downto 0) := X"0000_0000";
 signal co      : std_logic := '0';



begin

  DUT: adder generic map (WD=>32, BD=>4)
    port map (an, bn, ci, cn, co);


  clk <= not clk after half_period;

  rst <= '0', '1' after 2* half_period;


  process
  begin
     wait until rst = '1';

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


     bn <= X"0000_00FF";
     wait for 0 ns;

     for i in 0 to 31  loop
       wait until (clk'event and clk='1');

       an <= std_logic_vector(to_unsigned(i, 32));

       -- wait for 0 ns;

     end loop;

  end process;


  process
  begin
    wait for 100* clk_period;
    assert false report "end of simulation" severity failure;
  end process;

 --   XXXXXXX XXXXXX XXXXXX XXXXXX XXXXXXX XXXXXX XXXXX
```

```
end beh;


::::::::::::::
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;


::::::::::::::
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.mux.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    Barrel Shifter Based on Mux
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.07.27
--
--  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');
    dq    : out std_logic_vector (WD-1 downto 0) := (others=>'0'));

end bshift;


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);
    end loop;

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

  end process;



end mux;
::::::::::::::
c3.bshift.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    Barrel Shifter
--
--  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 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');
    dq   : out std_logic_vector (WD-1 downto 0) := (others=>'0'));

end bshift;
```

```
architecture rtl of bshift is

begin

  bshft: process (di, nbit)
    variable diX   : std_logic_vector (2*WD-1 downto 0) := (others=>'0');
    variable offset : natural := 0;
    variable result : std_logic_vector (WD-1 downto 0) := (others=>'0');
  begin  -- process bshft

    for i in 2*WD-1 downto 0 loop
      if i < WD then
        diX(i) := di(i);
      else
        diX(i) := di(WD-1);
      end if;
    end loop;  -- i

    offset := to_integer(unsigned(nbit));

    result := diX(WD-1+offset  downto offset);

    dq <= result;

  end process;


end rtl;
::::::::::::::
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:
--------------------------------------------------------------------------------

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.rfile.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    ROM Model (Data is read from a file)
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.11.12
--
--  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');
    cs     : in   std_logic := '0';
    data   : out  std_logic_vector (WD-1 downto 0) := (others=>'0') );

end rom;


architecture rfile of rom is
  type rarray is array (0 to PWR-1) of real;

  procedure ReadData (variable angles : out rarray) is
    file DataFile  : text open read_mode is "angle_real.dat";
    variable lbuf  : line;
    variable i     : integer := 0;
    variable rdata : real;

  begin
    while not endfile(DataFile) loop
      readline(DataFile, lbuf);
      read(lbuf, rdata);
      angles(i) := rdata;
      i := i + 1;
    end loop;
   end procedure;

begin

  ROM: process (addr, cs)
    variable angles : rarray;
    type darray is array (0 to PWR-1) of std_logic_vector (WD-1 downto 0);
    variable romData : darray;
    variable initRom : boolean := false;
  begin
    if (initRom=false) then
      ReadData(angles);
      for i in 0 to PWR-1 loop
        romData(i) := Conv2fixedPt(angles(i), WD);
      end loop;  -- i
      initRom := true;
    end if;

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

end rfile;
::::::::::::::
c6.rom_tb_rfile.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
```

```
--    configuration of cca adder testbench
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.11.12
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
--
--
--    Output:
--------------------------------------------------------------------------------

use WORK.all;


configuration rom_tb_rfile of rom_tb is
  for beh
    for rom_0: rom
      use entity work.rom(rfile) ;
    end for;
  end for;
end rom_tb_rfile;


::::::::::::::
c6.rom_tb.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    general testbench of c6.rom
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.11.14
--
--  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;
use WORK.all;


entity rom_tb is
end rom_tb;


architecture beh of rom_tb is

  component rom
    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');
      cs     : in   std_logic := '0';
      data   : out  std_logic_vector (WD-1 downto 0) := (others=>'0') );
  end component;

  -- for rom_0: rom use entity work.rom(rfile);

  constant WD : integer := 32;
  constant SH : integer := 6;
  constant PWR : integer := 64;

  signal clk, rst: std_logic := '0';
  signal addr    : std_logic_vector(SH-1 downto 0) := (others=>'0');
  signal data    : std_logic_vector(WD-1 downto 0) := (others=>'0');

 begin

   rom_0 : rom generic map (WD=>32, SH=>6, PWR=>64)
     port map (addr => addr, cs => '1', data => data);


   clk <= not clk after half_period;

   rst <= '0', '1' after 2* half_period;


   process
    file DataFile  : text open write_mode is "angle_real_out.dat";
    variable lbuf  : line;
    variable wdata : integer;
   begin

      wait until rst = '1';

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

       wait for 0 ns;

      for i in 0 to PWR-1  loop
```

```
        wait until (clk'event and clk='1');

        addr <= std_logic_vector(to_unsigned(i, SH));

        wait for 0 ns;
        wait until (clk'event and clk='0');
        wdata := to_integer(signed(data));
        write(lbuf, wdata, right, 10);
        writeline(DataFile, lbuf);


        wait for 0 ns;

      end loop;


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


   process
   begin
     wait for 100* clk_period;
     assert false report "end of simulation" severity failure;
   end process;

  --   XXXXXXX XXXXXX XXXXXX XXXXXX XXXXXXX XXXXXX XXXXX

end beh;
::::::::::::::
c6.rom.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    ROM Model (Data is embedded in this file as a constant)
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.11.12
--
--  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');
    cs     : in   std_logic := '0';
    data   : out  std_logic_vector (WD-1 downto 0) := (others=>'0') );

end rom;


architecture rtl of rom is
   type rarray is array (natural range <>) of real;

   constant angles : rarray :=
                     (  7.8539816339744830962E-01,
                        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,
                        1.7347234759768070944E-18,
                        1.7347234759768070944E-18,
                        1.7347234759768070944E-18,
                        1.7347234759768070944E-18  );


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) := Conv2fixedPt(angles(i), WD);
      end loop;  -- i
      initRom := true;
    end if;

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

end rtl;
::::::::::::::
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;




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

```
--  Modified:
--
--    2012.04.03
--
--  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;
::::::::::::::
cordic_rtl.vhdl
::::::::::::::
------------------------------------------------------------------------------
--
--  Purpose:
--
--    behavioral model of cordic
--
--  Discussion:
--
```

```
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.04.03
--
--  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);
      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, dq => xnS);

   ShftY : bshift generic map (WD=>32, SH=>5)
     port map (di  => yn, nbit => cnt, 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=>'1', 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 2* 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;
::::::::::::::
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;
::::::::::::::
angle_comp.c
::::::::::::::
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX_ANGLE (1 << 10)
// #define STR_PR

int main (int argc, char *argv[])
{

   FILE *fin1, *fin2;
   int i, num;

   char str_real[MAX_ANGLE][30];
   char str_fixed[MAX_ANGLE][30];


   double A_real[MAX_ANGLE];
   double A_fixed[MAX_ANGLE];
   double A_freal[MAX_ANGLE];
   double A_delta[MAX_ANGLE];
   double cos_delta[MAX_ANGLE];
   double sin_delta[MAX_ANGLE];
```

```
   fin1 = fopen("angle_real.dat", "r");
   fin2 = fopen("angle_real_out.dat", "r");

   if (fin1 == NULL) {
     perror ("Unable to open file fin1 \n");
     exit( EXIT_FAILURE);
   }

   if (fin2 == NULL) {
     perror ("Unable to open file fin2 \n");
     exit( EXIT_FAILURE);
   }

   i = 0;

   while (fscanf(fin1, "%s", &str_real[i]) != EOF) {
     fscanf(fin2, "%s", &str_fixed[i]);

     A_real[i]  = atof(str_real[i]);
     A_fixed[i] = atof(str_fixed[i]);

     A_freal[i] = A_fixed[i] / (1L << 29);
     A_delta[i] = A_real[i] - A_freal[i];

     printf("[%2d] ", i);
#ifdef STR_PR
     printf("str_real= %20s ", str_real[i]);
     printf("str_fixed= %12s ", str_fixed[i]);
#endif
     printf("A_real= %25.19e ", A_real[i]);
     printf("A_freal= %25.19e ", A_freal[i]);
     printf("A_delta= %25.19e ", A_delta[i]);
     printf("\n");
     i++;
   }
   num = i;

   for (i=0; i<num; ++i) {
     cos_delta[i] = cos(A_real[i]) - cos(A_freal[i]);

     printf("[%2d] ", i);
     printf("cos(A_real)= %25.19e ", cos(A_real[i]));
     printf("cos(A_freal)= %25.19e ", cos(A_freal[i]));
     printf("cos_delta= %25.19e ", cos_delta[i]);
     printf("\n");
   }

   for (i=0; i<num; ++i) {
     sin_delta[i] = sin(A_real[i]) - sin(A_freal[i]);

     printf("[%2d] ", i);
     printf("sin(A_real)= %25.19e ", sin(A_real[i]));
     printf("sin(A_freal)= %25.19e ", sin(A_freal[i]));
     printf("sin_delta= %25.19e ", sin_delta[i]);
     printf("\n");
   }

   double A_delta_avg   = 0.0;
   double cos_delta_avg = 0.0;
   double sin_delta_avg = 0.0;
   double A_delta_max   = 0.0;
   double cos_delta_max = 0.0;
   double sin_delta_max = 0.0;
   double A_delta_min   = 9999L;
   double cos_delta_min = 9999L;
```

```
   double sin_delta_min = 9999L;
   for (i=0; i<num; ++i) {
     A_delta[i] = fabs(A_delta[i]);
     cos_delta[i] = fabs(cos_delta[i]);
     sin_delta[i] = fabs(sin_delta[i]);
     A_delta_avg   += A_delta[i];
     cos_delta_avg += cos_delta[i];
     sin_delta_avg += sin_delta[i];
     if (A_delta_min > A_delta[i]) A_delta_min = A_delta[i];
     if (cos_delta_min > cos_delta[i]) cos_delta_min = cos_delta[i];
     if (sin_delta_min > sin_delta[i]) sin_delta_min = sin_delta[i];
     if (A_delta_max < A_delta[i]) A_delta_max = A_delta[i];
     if (cos_delta_max < cos_delta[i]) cos_delta_max = cos_delta[i];
     if (sin_delta_max < sin_delta[i]) sin_delta_max = sin_delta[i];
   }
   A_delta_avg   /= num;
   cos_delta_avg /= num;
   sin_delta_avg /= num;

   printf("|A_delta|   (min= %g avg= %g max= %g)\n",
               A_delta_min, A_delta_avg, A_delta_max);
   printf("|cos_delta| (min= %g avg= %g max= %g)\n",
               cos_delta_min, cos_delta_avg, cos_delta_max);
   printf("|sin_delta| (min= %g avg= %g max= %g)\n",
               sin_delta_min, sin_delta_avg, sin_delta_max);



   fclose(fin1);
   fclose(fin2);

   return 0;

}

::::::::::::::
angle_real.dat
::::::::::::::
                        7.8539816339744830962E-01
                        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
                    1.7347234759768070944E-18
                    1.7347234759768070944E-18
                    1.7347234759768070944E-18
                    1.7347234759768070944E-18
::::::::::::::
angle_real_out.dat
::::::::::::::
 421657428
 248918915
 131521918
  66762579
  33510843
  16771758
   8387925
   4194219
   2097141
   1048575
    524288
    262144
    131072
     65536
     32768
     16384
      8192
      4096
      2048
      1024
       512
       256
       128
        64
        32
        16
         8
         4
         2
```

```
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```