```
::::::::::::::
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
```

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

```
--
--    2012.10.05
--
--  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;




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


  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');



begin
```

```vhdl
-- ND Adders of BD-bit
-- ci1d(i) : carry in of the i-th BD-bit adder
-- co1d(i) : carry out 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 => ci1d(i),
                            cn => cn2d(i),
                            co => co1d(i) );
end generate ILOOP;


-- Carry Chain GP Logic
-- g1d(i) : carry generation of the i-th BD-bit adder
--          cn2d(i) > BD-1
-- p1d(i) : carry propagation of the i-th BD-bit adder
--          cn2d(i) = BD-1

process (co1d)
  variable tmp1d : array1d := (others=> '0');
begin
  tmp1d := co1d;

  for i in ND-1 downto 0 loop
      g1d(i) <= co1d(i);
  end loop;
end process;


process (cn2d)
  variable tmp2d: array2d  := ((others=> (others=> '0')));
  variable tmpv : std_logic_vector (BD-1 downto 0) := (others=>'0');
  variable tmp : std_logic := '0';
begin
  tmp2d := cn2d;

  for i in ND-1 downto 0 loop
      tmpv := tmp2d(i);
      tmp  := '1';
      for j in BD-1 downto 0 loop
          tmp := tmp and tmpv(j);
      end loop;
      p1d(i) <= tmp;
  end loop;
end process;

-- Carry Chain Cell
-- 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;
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;



::::::::::::::
adder_tb.vhdl
::::::::::::::
-------------------------------------------------------------------------------
--
--  Purpose:
--
--    testbench of adder
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.10.05
--
--  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;


  signal clk, rst: std_logic := '0';
  signal an      : std_logic_vector(31 downto 0) := X"7FFF_FFFF";
  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;
::::::::::::::
adder_tb_conf.vhdl
::::::::::::::
-------------------------------------------------------------------------------
--
--  Purpose:
--
--    configuration of testbench of adder
--
```

```
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.10.05
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
--
--
--    Output:
-------------------------------------------------------------------------------

use WORK.all;


configuration adder_tb_conf 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;
      end for;
    end for;
  end for;
end adder_tb_conf;
```