```
::::::::::::::
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;
::::::::::::::
```

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.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    Carry Chain Adder
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.10.25
--
--  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 cca_rom 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);
      p     : out  std_logic := '0';
      g     : 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;


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

  FromA2d(cn2d, cn);


  ------------------------------------------------------------------------
  -- Computing Carry Chain GP Logic
```

```
-- i-th BD-bit adder
-- g1d(i) : carry generation  : an2d(i) + bn2d(i) > BD-1
-- p1d(i) : carry propagation : an2d(i) + bn2d(i) = BD-1
--------------------------------------------------------------------------------
-- TBD:  LUT implementation --> Hauck, Hosler, Fry Paper
--------------------------------------------------------------------------------
process (an2d, bn2d)
  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*BD + j) := '1';
        else
          g(i*BD + j) := '0';
        end if;
      end loop;  -- j
    end loop;  -- i
    return g;
  end;


  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*BD + j) := '1';
        else
          p(i*BD + j) := '0';
        end if;
      end loop;  -- j
    end loop;  -- i
    return p;
  end;

  constant rom_g : rom_type := init_g;
  constant rom_p : rom_type := init_p;


  variable a, b, addr : integer := 0;
  variable i : line;

  variable tmp1d_p, tmp1d_g : array1d := (others=> '0');
begin

  for i in ND-1 downto 0 loop
      a := to_integer(unsigned(an2d(i)));
      b := to_integer(unsigned(bn2d(i)));
      addr := a * (2**BD) + b;

      tmp1d_g(i) := rom_g(addr);
      tmp1d_p(i) := rom_p(addr);
  end loop;

  g1d <= tmp1d_g;
  p1d <= tmp1d_p;

end process;


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



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


::::::::::::::
adder_cca.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    configuration of cca adder
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.10.22
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
--
--
--    Output:
--------------------------------------------------------------------------------

use WORK.all;


configuration adder_cca of adder is
   for cca
     for ILOOP
        for U0:subadder
          use entity work.adder(rca);
        end for;
      end for;
    end for;
end adder_cca;
::::::::::::::
adder_tb_cca.vhdl
```

```
::::::::::::::
-------------------------------------------------------------------------------
--
--  Purpose:
--
--    configuration of testbench of cca adder
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.10.20
--
--  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;
      end for;
    end for;
  end for;
end adder_tb_cca;


--configuration adder_cca_conf of adder is
--   for cca
--     for ILOOP
--      for U0:subadder
--          use entity work.adder(rca);
--        end for;
--      end for;
--    end for;
--end adder_cca_conf;
::::::::::::::
adder_tb_rca.vhdl
::::::::::::::
-------------------------------------------------------------------------------
--
--  Purpose:
--
--    configuration of testbench of rca adder
--
--  Discussion:
--
--
```

```
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.10.20
--
--  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;
:::::::::::::::
makefile
:::::::::::::::


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.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 &

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\
            adder_tb.vhdl adder_cca.vhdl adder_tb_cca.vhdl adder_tb_rca.vhdl
adder : ${SRC_adder}
        ghdl -a cordic_pkg.vhdl
        ghdl -a c1.adder.vhdl
        ghdl -a c1.adder.rca.vhdl
        ghdl -a c1.adder.cca.vhdl
        ghdl -a adder_cca.vhdl
        ghdl -a adder_tb.vhdl

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


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


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


        more ${SRC_adder} makefile > adder.cca.files

        gtkwave adder.vcd &


clean :
        \rm -f *.o *~ *# *.cf
        \rm -f *_tb
        \rm -f *_conf
        \rm -f *.vcd

file :
        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     > print.file
```