

```
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
```



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

```
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;
```

```
  constant ND : natural := WD/BD;
```

```
  -- an2d, bn2d, cn2d : array(ND, BD) <= an, bn, cn
  -- cild, cold       : array(ND)      <= ci, co
  -- gld, pld        : array(ND)      -- Generate, Propagate
  -- qild, qold      : 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 cild, cold : array1d := (others=> '0');
  signal qild, qold : array1d := (others=> '0');
  signal gld, pld  : array1d := (others=> '0');
```

```
begin
```

```
-----
-- ND Adders of BD-bit
-----
-- cild(i)   : cin's of the i-th BD-bit adder
-- cold(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 => qild(i),
                 cn => cn2d(i),
                 co => cold(i) );
end generate ILOOP;

```

```

-----
-- an2d <= an
-- bn2d <= bn
-- cn <= cn2d
-----

```

```

process (an)
    variable tmp2d: array2d := ((others=> (others=> '0')));
    variable tmpv : std_logic_vector (WD-1 downto 0) := (others=>'0');
begin
    tmpv := an;

    for i in ND-1 downto 0 loop
        tmp2d(i) := tmpv((i+1)*BD-1 downto i*BD);
        an2d(i) <= tmp2d(i);
    end loop;
end process;

```

```

process (bn)
    variable tmp2d: array2d := ((others=> (others=> '0')));
    variable tmpv : std_logic_vector (WD-1 downto 0) := (others=>'0');
begin
    tmpv := bn;

    for i in ND-1 downto 0 loop
        tmp2d(i) := tmpv((i+1)*BD-1 downto i*BD);
        bn2d(i) <= tmp2d(i);
    end loop;
end process;

```

```

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

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

    cn <= tmpv;
end process;

```

```

-----
-- Computing Carry Chain GP Logic
-- i-th BD-bit adder
-- gld(i) : carry generation : an2d(i) + bn2d(i) > BD-1
-- pld(i) : carry propagation : an2d(i) + bn2d(i) = BD-1
-----

```

```

-- LUT implementation : TBD --> Hauck, Hosler, Fry Paper
-----

```

```

process (an, bn)
    variable tmpv_a : std_logic_vector (WD-1 downto 0) := (others=>'0');
    variable tmpv_b : std_logic_vector (WD-1 downto 0) := (others=>'0');
    variable tmp_a  : std_logic_vector (BD-1 downto 0) := (others=>'0');
    variable tmp_b  : std_logic_vector (BD-1 downto 0) := (others=>'0');
    variable tmp1d_g : array1d := (others=> '0');
    variable tmp1d_p : array1d := (others=> '0');
    variable a, b : integer := 0;

```

```

begin
  tmpv_a := an;
  tmpv_b := bn;

  for i in ND-1 downto 0 loop
    tmp_a := tmpv_a((i+1)*BD-1 downto i*BD);
    tmp_b := tmpv_b((i+1)*BD-1 downto i*BD);
    a := to_integer(unsigned(tmp_a));
    b := to_integer(unsigned(tmp_b));

    if ((a+b) > (B -1)) then
      tmpld_g(i) := '1';
    else
      tmpld_g(i) := '0';
    end if;

    if ((a+b) = (B -1)) then
      tmpld_p(i) := '1';
    else
      tmpld_p(i) := '0';
    end if;
  end loop;

  gld <= tmpld_g;
  pld <= tmpld_p;

```

```
end process;
```

```

-----
-- co, qold <= Carry Chain Cell <= qild, ci
-- qild(i) : input of a carry chain cell
-- qold(i) : output of a carry chain cell
-----

```

```

process (ci, qold)
  variable tmpld : arrayld := (others=> '0');
  variable tmp : std_logic := '0';
begin
  tmp := ci;
  tmpld := qold;

  for i in ND-1 downto 1 loop
    qild(i) <= qold(i-1);
  end loop;

  qild(0) <= tmp;
  co <= qold(ND-1);
end process;

```

```

process (pld, gld, qild)
  variable tmpld_p, tmpld_g, tmpld_qi : arrayld := (others=> '0');
begin

  tmpld_p := pld;
  tmpld_g := gld;
  tmpld_qi := qild;

  for i in ND-1 downto 0 loop
    if (tmpld_p(i) = '1') then
      qold(i) <= tmpld_qi(i);
    else
      qold(i) <= tmpld_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:  
--  
-- 2012.10.05  
--  
-- 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;
```

```

for DUT: adder use configuration work.adder_cca_conf;

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_cca_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_cca_conf of adder is
  for cca
    for ILOOP
      for U0:subadder
        use entity work.adder(rca)
          port map (an, bn, ci, cn, co);
        end for;
      end for;
    end for;
  end adder_cca_conf;
:::
makefile
:::
```

```
anal : 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_conf.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_conf.vhdl
```

```
ghdl -a adder_tb.vhdl
```

```
ghdl -e adder_tb
```

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

```
ghdl -r adder_tb --disp-tree=inst --vcd=adder.vcd
```

```
# gtkwave adder.vcd &
```

```
clean :
```

```
\rm -f *.o *~ *# *.cf
```

```
\rm -f *_tb
```

```
\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
```