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

library STD;
use STD.textio.all;

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


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

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

```
end adder;


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

library STD;
use STD.textio.all;

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




architecture rca of adder is
begin
  process (an, bn, ci)
    variable sn : std_logic_vector (WD-1 downto 0) := (others=>'0');
    variable c  : std_logic := '0';
```

```
  begin  -- process
    c := ci;
    for i in 0 to WD-1 loop
      sn(i) := an(i) xor bn(i) xor c;
      c := (an(i) and bn(i)) or (an(i) and c) or (bn(i) and c);
    end loop;  -- i

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

end rca;



::::::::::::::
c1.adder.cca.gprom.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    GP Logic of a Carry Chain Adder
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2012.11.06
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input: an, bn : BD-bits
--
--    Output: g, p : 1-bit
--------------------------------------------------------------------------------

library STD;
use STD.textio.all;

library IEEE;
```

```
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

use WORK.cordic_pkg.all;



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

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




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

begin
  -----------------------------------------------------------------------
  -- Computing Carry Chain GP Logic
  -- i-th BD-bit adder
  -- g : carry generation  : an + bn > BD-1
  -- p : carry propagation : an + bn = BD-1
  -----------------------------------------------------------------------
  -- TBD:  LUT implementation --> Study Hauck, Hosler, Fry Paper
  -----------------------------------------------------------------------

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

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

```
        else
          g(i*(2**BD) + j) := '0';
        end if;
      end loop;  -- j
    end loop;  -- i
    return g;
  end;
  ----------------------------------------

  constant rom_g : rom_type := init_g;

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

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

end process;


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

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

  constant rom_p : rom_type := init_p;

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

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


end rtl;



::::::::::::::
c1.adder.cca.subadd.vhdl
::::::::::::::
--------------------------------------------------------------------------------
--
--  Purpose:
--
--    Ripple Carry Adder
--
--  Discussion:
--
--
--  Licensing:
--
--    This code is distributed under the GNU LGPL license.
--
--  Modified:
--
--    2013.11.13
--
--  Author:
--
--    Young W. Lim
--
--  Parameters:
--
--    Input:
--
--    Output:
--------------------------------------------------------------------------------

library STD;
use STD.textio.all;

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

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

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

end subadder;


architecture rca of subadder is

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

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

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

begin

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



end rca;
```

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

library STD;
use STD.textio.all;

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

use WORK.cordic_pkg.all;




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

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


architecture cca of adder is

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

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


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

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


  constant ND : natural := WD/BD;

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


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

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

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

  end ToA2d;


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

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

    a <= tmpv;
  end FromA2d;



begin

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

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


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

FromA2d(cn2d, cn);



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

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

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


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

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

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

  end process;



end cca;
```