

```
-----
--
-- Purpose:
--
--   Carry Chain Adder
--
-- Discussion:
--
--
-- Licensing:
--
--   This code is distributed under the GNU LGPL license.
--
-- Modified:
--
--   2012.09.08
--
-- 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;
```

```
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;
```

```
-----
-- 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 Badder is
    generic (
      WD      : 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) : carry in of the i-th BD-bit adder
-- cold(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: Badder generic map (WD => BD)
    port map (an => an2d(i),
              bn => bn2d(i),
              ci => cild(i),
              cn => cn2d(i),
              co => cold(i) );
```

```
end generate ILOOP;
```

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

```
process (cold)
  variable tmp1d : array1d := (others=> '0');
```

```
begin
  tmp1d := cold;
```

```
  for i in ND-1 downto 0 loop
    gld(i) <= cold(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;
```

```

        pld(i) <= tmp;
    end loop;
end process;

-- Carry Chain Cell
-- 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;
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;

```