

```

*****
* Based on Technical Report 94-9 Monash University
*
* Being Modified By Young W. Lim
*
* Modified code is to be distributed under the GNU LGPL license.
*****

```

```

addsub : PROCESS(a,b,sel)
    VARIABLE res : VLBIT_VECTOR(n DOWNTO 0);
BEGIN
    result := zero(n DOWNTO 0);    -- needs to be initialised

    IF sel = '1' THEN
        result := add2c(a,b);
    ELSE
        result := sub2c(a,b);
    END IF;

    s <= result(n-1 downto 0);    -- discard cout
END PROCESS;

```

IEEE Standard Packages

```

add2c()
sub2c()
function add2c (v1, v2: vlbit_ld) return vlbit_ld;
function sub2c (v1, v2: vlbit_ld) return vlbit_ld;
-----

```

adder/subtractor structure

```

c(0) <= sel; -- carry in
connect: FOR i IN 0 TO n-1 GENERATE
    invert:      invf101 PORT MAP( b(i), b_bar(i) );
    mux_b_b_bar: muxf201 PORT MAP( b_bar(i), b(i), sel, b_hat(i) );
    addsub:     faf001  PORT MAP( a(i), b_hat(i), c(i), s(i), c(i+1) );
END GENERATE

```

Standard Cell library

Assume the library contains the following 6 components

```

nandf201: 2 input nand with 1x output drive
norf201:  2 input nor  with 1x output drive
invf101:  1 input not  gate with 1x output drive
xorf201:  2 input xor  gate with 1x output drive
xnof201:  2 input xnor gate with 1x output drive
dffb311:  D-Flip Flop with D, Reset, Set, Q, QN, Clk

```

```

invf101

```

muxf201
faf001

addsub

```
SUM <= A1 xor B1 xor CIN2;  
CO <= (A1 and B1) or (A1 and CIN2) or (B1 and CIN2);
```

ARCHITECTURE behaviour OF **adder** IS

begin

```
cell_i : process (xi,xs,yi,ys,zi,ai)
```

```
  VARIABLE x_res: vlbit_vector(n downto 0); -- temporary results
```

```
  VARIABLE y_res: vlbit_vector(n downto 0);
```

```
  VARIABLE z_res: vlbit_vector(k downto 0);
```

```
  begin
```

```
    x_res := zero(n downto 0); -- initialise, unless comp complains
```

```
    y_res := zero(n downto 0);
```

```
    z_res := zero(k downto 0);
```

```
    if zi(k-1) = '0' then -- z_i is positive
```

```
      x_res := add2c (xi, ys);
```

```
      y_res := sub2c (yi, xs);
```

```
      z_res := sub2c (zi, ai);
```

```
    else -- z_i is negative
```

```
      x_res := sub2c (xi, ys);
```

```
      y_res := add2c (yi, xs);
```

```
      z_res := add2c (zi, ai);
```

```
    end if;
```

```
    xipl <= x_res (n-1 downto 0);
```

```
    yipl <= y_res (n-1 downto 0);
```

```
    zipl <= z_res (e-1 downto 0);
```

```
  end process;
```

```
END behavior;
```

form a half adder. The interconnection of the inc001 components is:

```
c(0) <= cin; -- first carry
```

```
connect: for i in 0 to n-1 generate
```

```
  addsub: inc001 port map( a(i), c(i), s(i), c(i+1) );
```

```
end generate;
```

inc001

rounder : process (a,cin)

```
  VARIABLE res: vlbit_vector(n downto 0); -- temporary results
```

```
begin
```

```

    res := zero(n downto 0); -- initialise, unless comp complains
    res := addum(a,cin); -- use addum instead of add2c as it sign
                        -- extends the cin input making it -1 not +1
    s <= res (n-1 downto 0);
end process;

```

```

-----
addum
-----

```

shift value is not recognizable inside the generate statement.

```

-- Scaled a_i * 2^i values are decimal 45 53 56 57 57 57 57 57
ai <= X"39_39_39_39_39_38_35_2D";

```

```

sh_x: xis <= shift_all(xi); -- xis, xi width?
sh_y: yis <= shift_all(yi); -- yis, yi width?
sh_z: zis <= shift_z(zi);

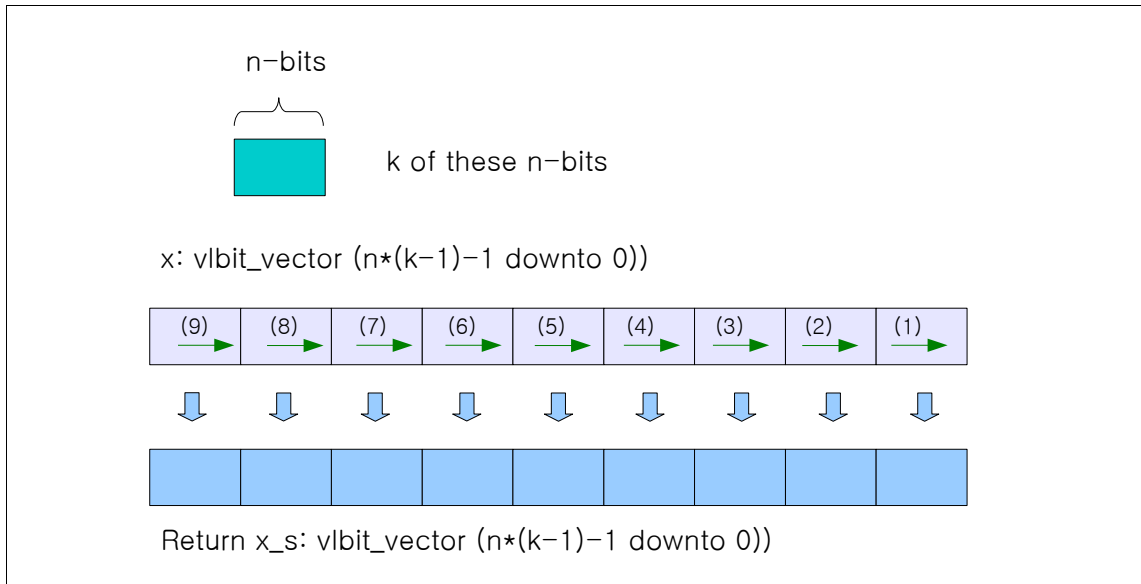
```

```

FUNCTION shift_all (x : vlbit_vector (n*(k-1)-1 downto 0))
RETURN vlbit_vector IS
    VARIABLE x_s : vlbit_vector(n*(k-1)-1 downto 0)
                        := zero(n*(k-1)-1 downto 0);
BEGIN
    x_s(1*n-1 downto 0)      := shiftr2c(x( 1*n-1 downto 0) ,1); -- 2 stage
    x_s(2*n-1 downto 1*n)   := shiftr2c(x( 2*n-1 downto 1*n) ,2); -- 3 stage
    x_s(3*n-1 downto 2*n)   := shiftr2c(x( 3*n-1 downto 2*n) ,3); -- 4 stage
    x_s(4*n-1 downto 3*n)   := shiftr2c(x( 4*n-1 downto 3*n) ,4); -- 5 stage
    x_s(5*n-1 downto 4*n)   := shiftr2c(x( 5*n-1 downto 4*n) ,5); -- 6 stage
    x_s(6*n-1 downto 5*n)   := shiftr2c(x( 6*n-1 downto 5*n) ,6); -- 7 stage
    x_s(7*n-1 downto 6*n)   := shiftr2c(x( 7*n-1 downto 6*n) ,7); -- 8 stage
    x_s(8*n-1 downto 7*n)   := shiftr2c(x( 8*n-1 downto 7*n) ,8); -- 9 stage
    x_s(9*n-1 downto 8*n)   := shiftr2c(x( 9*n-1 downto 8*n) ,9); -- 10 stage

    return x_s;
END shift_all;

```



Standard Cell library

function shiftr2c (v: vlbit_1d; i: integer) **return** vlbit_1d;

```

initial: init port map( xi <= X"00",
                        xs <= x_in,
                        yi <= X"00",
                        ys <= y_in,
                        zi <= z_in,
                        ai <= B"0_0101_1010", -- add/sub 90 degrees
                        xipl <= xinit, -- xinit = 0 +- yin
                        yipl <= yinit, -- yinit = 0 -+ xin
                        zipl <= zinit );

```

connect: for i in 0 to k-1 generate -- k stages

```

ls_unit: if i=0 generate
  first_unit: adder port map( ... );
end generate ls_unit;

```

```

i_unit: if i>0 and i<k-1 generate
  x_round: round port map ( ... );
  y_round: round port map ( ... );
  middle_units: adder port map( ... );
end generate ls_unit;

```

```
ms_unit: if i=k-1 generate
  x_round_last: round port map ( ... );
  y_round_last: round port map ( ... );
  last_unit: adder port map( ... );
end generate ms_unit;
```

```
end generate connect;
```