functions/procedures (also recursion)

```
FUNCTION bit_wise (a, b : std_logic_vector) RETURN std_logic_vector IS
BEGIN
  ASSERT a'LENGTH=b'LENGTH REPORT "operand length not equal“
                                  SEVERITY failure;
  FOR i IN a'RANGE LOOP
    res(i) := operation( a(i), b(i) );
  END LOOP;
  RETURN res;
END bit_wise;
```

often works ..

But ..

```
FUNCTION bit_wise (a, b : std_logic_vector) RETURN std_logic_vector IS
  CONSTANT ai  : std_logic_vector(a'LENGTH-1 DOWNTO 0)  := a;
  CONSTANT bi  : std_logic_vector(b'LENGTH-1 DOWNTO 0) := b;
  VARIABLE res : std_logic_vector(a'LENGTH-1 DOWNTO 0) ;
BEGIN
  ASSERT ai'LENGTH=bi'LENGTH REPORT "operand length not equal“
                                  SEVERITY failure;
  FOR i IN ai'RANGE LOOP
    res(i) := operation(ai(i),bi(i));
  END LOOP;
  RETURN res;
END bit_wise;
```

Why is vector not aligned automatically?

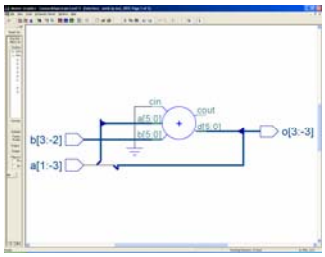
can be used for (un)signed fixed point

- **type** ufix **is array** (integer **range <>**) **of** std_logic;
- variable v : ufix (2 downto -1);
  - V:=“1011” ➔ $1\times2^{2} + 0\times2^{1} + 1\times2^{0} + 1\times2^{-1}$
- variable v1 : ufix(2 downto -4);
  variable v2 : ufix(3 downto -3);
  variable r : ufix(3 downto -4);
- Overloading “+” operator with ‘ufix’ interpretation:
  r := v1 + v2;

Example

```
use work.fixed_point.all;
entity fp is
 port (a : in  sfix(1 downto -3);
       b : in  sfix(3 downto -2);
       o : out sfix(3 downto -3)
       );
end fp;

architecture test of fp is
begin
  o <= a + b;
end test;
```



math

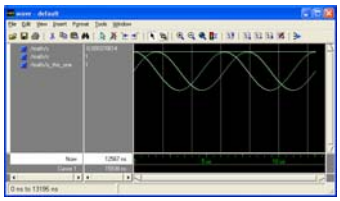
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.math_real.all;
entity math is
end math;

architecture demo of math is
  signal s, c, is_this_one : real := 0.0;
Begin

  process
    constant delta : real := 10.0E-4;
    variable x : real := 0.0;
  begin
    x:=0.0;
    while x < 4.0*MATH_PI loop
      s <= sin(x);
      c <= cos(x);
      wait for 1 ns;
      x:=x+delta;
    end loop;
    report "finished";
    wait;
  end process;

  is_this_one <= s**2 + c**2;

end demo;
```



or is it a complex problem ?

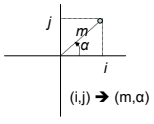
```
entity complex_demo is
end complex_demo;

library ieee;
use ieee.math_complex.all;
architecture test of complex_demo is
  signal i, square_root : complex := (0.0,0.0);
  signal o : complex_polar := (0.0,0.0);

begin
  stimuli:process
  begin
    i <= ( 3.0, 4.0);
    wait for 10 ns;
    i <= (-3.0, 4.0);
    wait for 10 ns;
    i <= (-4.0, 0.0);
    wait for 10 ns;
    i <= (-1.0, 0.0);  -- sqrt(-1)!
    wait for 10 ns;
    wait;
  end process;

  o <= complex_to_polar(i);
  square_root <= sqrt (i);

end test;
```



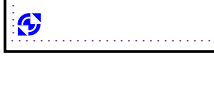
(i,j) ➔ (m,α)

ns	i	square_root	o
0	{3 4}	{2 1}	{5 0.927295}
10	{-3 4}	{1 2}	{5 2.2143}
20	{-4 0}	{1.22461e-016 2}	{4 3.14159}
30	{-1 0}	{6.12303e-017 1}	{1 3.14159}

assertion based verification

```
maximum:PROCESS(a,b,c)
  VARIABLE res : integer;
 BEGIN
  IF a > b THEN
    res := a;
  ELSE
    res := b;
  END IF;
  IF res > c THEN
    res := c;
  ELSE
    res := res;
  END IF;
  o <= res;
 END PROCESS maximum;

POSTPONED ASSERT ((o>=a) AND (o>=b) AND (o>=c))
  REPORT "maximum is not "& integer'image(o) SEVERITY error;
```



assertion based verification

PSL: Property Specification Language
(included in VHDL IEEE std. 1076-2004 (fast track) or 2006)

--PSL **default** clock **is** (clk'**event and** clk='0');

--PSL **always** ((state=S1) -> **next** ((state=S2) **until** (state=S3)));
State transitions from S1 to S2, and stays in S2 until S3

--PSL **always** ((state=S1) -> **eventually!** (state=S2) **until** (state=S3));
State eventually will go to S2, either directly of through some others states first.
Once it reaches S2, state will stay in S2 until state goes to S3