Shift Operators :-

Variable A : bit_vector := "101001";

A sll 2 results in "100100"
A srl 2 results in "001010"
A sla 2 results in "100111"
A sra 2 results in "111010"
A rol 2 results in "100110"
A ror 2 results in "011010"

4. Addition Operators :-

- the addition operators are used to perform arithmetic operation (addition and subtraction) on operands of any numeric type.
- the concatenation (&) operator is used to concatenate two vectors together to make a new one.
- In order to use these operators one has to specify the ieee.std_logic_unsigned.all
(or)
std_logic_arith.all
in addition to the ieee.std_logic_1164 package

Example for concatenation operator :-

'0' & '1' will result in "01".

'C' & 'A' & 'T' will result in "CAT"

"BA" & "LL" will result in "BALL"

⇒ '+' and '-' will have simple addition & subtraction operations.

5. Multiplying Operators :-

" * / rem mod "

* → gives multiplication output

10 * 5 = 50

/ → gives ~~multiplication~~ division output as quotient

10/5 = 2.

rem → gives division output as remainder.

A rem B

eg:- +5 rem +3 = +2

where sign of output depends on operand 'A'

eg:- -5 rem +3 = -2

mod → modulo operator

→ Here sign of output depends on operand 'B'

A B

eg:- +9 mod +4

it can be calculated as:-

~~A mod B = A - B~~

~~A B → A * B = +ve~~

So,

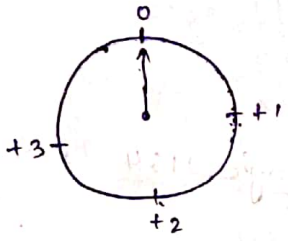
1. Draw a clock and clock has digits 0 to one less in magnitude than B.

i.e B = 4 . so go from 0 to 3.

2. Sign of no.s on dial determined by sign B.

3. Start counting around clock from 0 for A steps.

- if a × b = +ve then rotate clockwise
- if a × b = -ve the rotate anticlockwise.



Here A×B = 9×4 = +36 = +ve

So, clockwise from 0 to 9 steps:-

So, we get +1.

∴ +9 mod +4 = +1

6. Miscellaneous Operators :-

"**" → exponentiation.

eg:- $2^{**}8 = 2^8 = 256$.

"abs" → absolute value.

eg:-
```
X <= 2;
Y <= 3;
abs (X-Y); will result in +1.
```

"not" is simply ~~xx~~logical ~~negicatio~~ negation or inversion.

eg:- not '1' gives '0' as result.

Behavioural Modelling:-

- In this section, we will discuss different constructs for describing the behaviour of components and circuits in terms of sequential statements.
- The basis of sequential modeling is the process construct.
- The process construct allows us to model complex digital systems, in particular sequential statements.

1. Process :-

Syntax for process statement is:-

```
[process_label : process (sensitivity_list)
        [process_item_declarations]
begin
        sequential_statements : these are ->
        variable_assignment_statement
        signal_assignment_statement
        wait_statement
        if_statement
        case_statement
        loop_statement
        null_statement
        exit_statement
        next_statement
        assertion_statement
        procedure_call_statement
        return_statement
end process process_label
```

Before describing sequential statements, let us know that not all objects in VHDL description are created using object declarations. These other objects are declared as:-

1. ports of an entity. All ports are signal objects.
2. generics of an entity. These are constant objects.
3. formal parameters of function and procedures. function parameters are constant objects or signal objects. while procedure parameters can belong to any object class.
4. A file declared by file declaration.

- There are two other types of objects that are implicitly declared. These are indices of a for...loop statement and the generate statement

→ Now, lets take an example of sequential system:-

eg:- A positive edge triggered D-flip flop with asynchronous clear input

Program:-

```
library ieee;
use ieee.std_logic_1164.all;
entity DFF_CLEAR is
    port( CLK, clear, D : in std_logic;
          Q : out std_logic);
end DFF_CLEAR;
architecture BEHAV_DFF of DFF_CLEAR is
begin
DFF_PROCESS : process (CLK, clear)
    begin
        if (clear = '1') then
            Q<='0';
        else if ( CLK'event and CLK='1') then
            Q<= D;
```

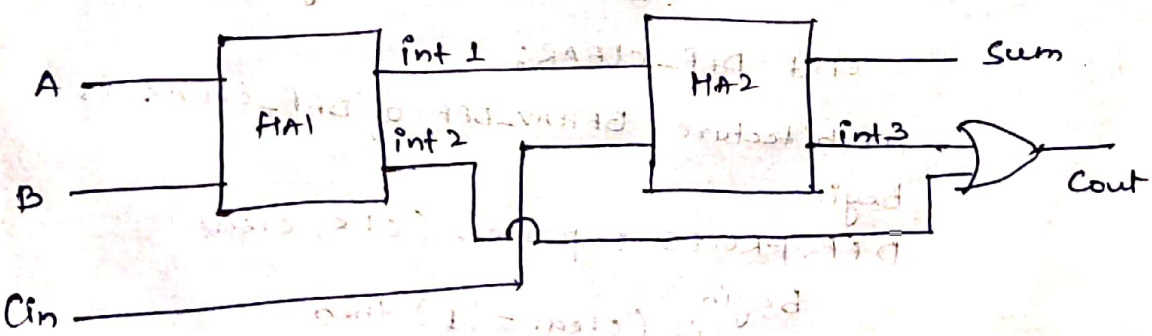
```
    end if;
end process;
end BEHAV_DFF;
```

Explaination :-

- Here, expression clk'event and clk='1' checks for "positive clock edge".
- The sensitivity list here "clk and clear" are the set of signals to which the process is sensitive.
  If the sensitivity list is not specified, one has to include a wait statement to make sure the process will halt.
  One cannot include both sensitivity list and wait statement.
- Variables and constants that are used inside a process have to be defined in the "process_declaration" part before keyword begin.
  Variables assignment inside a process are denoted by ":="
  Signal assignment inside a process are denoted by "<=".

eg:-2 Combinational circuit using process statement.

1. Program for full Adder using HALF ADDER



Program : (using two process)

```
library ieee;
use ieee.std_logic_1164.all;
entity FULL_ADDER is
  port( A, B, Cin : in std_logic
        Sum, Cout : out std_logic);

end FULL_ADDER;

architecture BEHAV_FA of FULL_ADDER is
signal int1, int2, int3 : std_logic;
begin
-- Process P1 that defines the first half adder--
P1 : process (A, B)
     begin
        int1 <= A xor B;
        int2 <= A and B;

     end process;
-- Process P2 that defines the second half adder and
     OR gate--
P2: process (int1, int2, Cin)
     begin
        Sum <= int1 xor Cin
        int3 <= int1 and Cin
        Cout <= int2 or int3;

     end process;

end BEHAV_FA;
```