Structural Description

* Highlights of structural Description :-

- structural description simulates the system by describing it's logical components.

  The components can be gate level such as And gates, OR gates or NOT gates or the components can be in a higher logical level such as register transfer level (RTL) or processor level.

- It is more convenient to use structural description rather than behavioral description for systems that required a specific design.
- All the statements in structural descriptⁿ are concurrent
- A major difference betⁿ VHDL & verilog structural description is the availability of components to the user.

* Organization of The structural description :-

- In the VHDL description the structural code has two parts. declaration and instantiation.
- In declaration all of the different types of components are declared. for eg.

```
component xor2
Port ( I1, I2 : in std_logic ; O1 : out
        std_logic);
end component;
```

- To specify the type of component (eg. AND, OR, XOR etc). additional information should be given to the simulator.
- If system has two or more identical components only one declaration is needed.
- The instantiation part of the code maps the generic inputs/outputs to the actual inputs/outputs of the system.

for eg.

```
X1 : xor2 portmap (a, b, sum);
```

maps input a to i/p I_1 of xor2, input b to input I_2 of xor2 & o/p sum to o/p O_1 of xor2.

- This mapping means that the logic relationship between a, b & sum is same as I1, I2 & O1.
- structural description statements are concurrent & are driven by events. This means that their execution depends on events not on the order that the statements are placed in the module.

* VHDL structural Description :-

```
Library ieee;
use ieee.std_logic_1164.all;
entity system is
    Port (a, b: in std_logic;
    sum, cout : out std_logic);
    end system;
architecture stru_arch of system is
  component xor2
    Port( I1, I2 : in std_logic ;
          O1 : out std_logic);
    end component;
    component and2
          Port ( I1, I2 : in std_logic;
            O1 : out std_logic);
    end component;
    begin
```

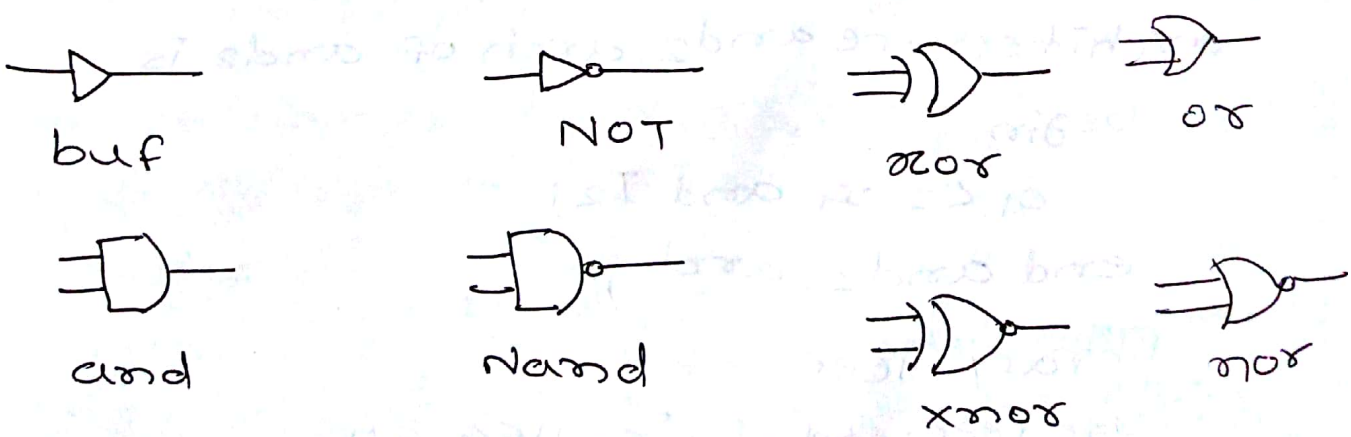
-- start of instantiation statements.

```
x1 : xor2 port map (a, b, sum);
A1 : and2 port map (a, b, cout);
end stru-arch;
```

* Verilog structural Description :-

```
module system (a, b, sum, cout);
 input a, b;
 output sum, cout;
 xor x1 (sum, a, b);
/* x1 is an optional identifier ; it can
   be omitted.*/
 and a1 (cout, a, b);
/* a1 is optional identifier ; it can be
   omitted. */
 endmodule
```

* Verilog built in gates :-



Structural description of Half adder:-

VHDL Half adder description :-

```
Library ieee;
use ieee.std_logic_1164.all;
entity xor2 is
port (I1, I2: in std_logic;
      O1: out std_logic);
end xor2;
architecture xor2_arch of xor2 is
  begin
    O1 <= I1 xor I2;
  end xor2;
library ieee;
use ieee.std_logic_1164.all;
entity and2 is
  Port (I1, I2: in std_logic;
        O1; out std_logic);
end and2;
architecture and2_arch of and2 is
  begin
    O1 <= I1 and I2;
  end and2_arch;
library ieee;
use ieee.std_logic_1164.all;
```

```
entity half-adder is
   Port (a, b : in std-logic;
         S, C : out std-logic);
   end half-adder;
architecture HA-arch of half-adder is
   component xor2
    Port (I1, I2 : in std-logic;
          O1 : out std-logic);
   end component;
Component and2
    Port (I1, I2 : in std-logic;
          O1 : out std-logic);
 end component;
 begin
  X1 : xor2 port map (a,b,s);
  A1 : and2 port map (a, b, c);
  end HA-arch;
```

* Verilog Half Adder Description :-

```
module half-add (a,b,S,C);
 input a, b;
 output S,C;
 xor (S, a, b);
 and ( C, a, b);
 endmodule
```

* write verilog codes for xor & and gates.

* Binding :-

- Binding in HDL is common practice. Binding (linking) segment1 in HDL code to segment2 makes all information in segment2 visible to segment1.

* Binding betⁿ Entity and Architecture in VHDL :-

```
entity one is
port( I1, I2 : in std_logic ;
      O1 : out std_logic);
end one;
architecture A of one is
signal s : std_logic;
-----
end A;
architecture B of one is
signal x : std_logic;
---
end B;
```

- architecture A is bound to entity one through the predefined coord of. Also architecture B is bound to entity one through predefined word of.

- Accordingly I1, I2 & O1 can be used in both architecture A & B.
- Architecture A is not bound to B hence signal s is not recognized in architecture B. likewise signal x is not recognized in architecture A.

* Binding betn Entity & component in VHDL :-

```
entity orgate is
Port ( I1, I2 : in std_logic;
       O1 : out std_logic);
end orgate;
architecture or_dataflow of orgate is
 begin
   O1 <= I1 or I2;
 end or_dataflow;
entity System is
Port ( x, y, z : in std_logic;
  r : out std_logic_vector (3 downto 0);
 end system;
architecture System_str of System is
 component orgate
  Port ( I1, I2 : in std_logic;
         O1 : std_logic);
  end component;
```

```
begin
orgate port map (x, y, z<co>);
.....
end system_str;
```

- The component orgate is bound to the entity orgate bcoz it has same name. architecture or-dataflow is bound to entity orgate by 'of'.
- All the information in the entity is now visible to the component.

* **Binding between Library & Module in VHDL:-**

```
library ieee;
use ieee.std_logic_1164.all;
entity system is
Port( I1, I2 : in std_logic;
      O1 : out std_logic_vector (3 downto 0));
end system;
architecture li_bound OF system is
signal s : std_logic;
-------
end li_bound;
```

* Library is a predefined word,
IEEE is the name of library,
use is a predefined word & ieee.std_logic_1164.all refers to part of library.

- If we don't write 1st two statements in code the standard_logic cannot be used.
- Libraries can also be generated by the user.

* Binding Between two modules in Verilog

```
module one (o1, o2, a, b);
    input [1:0] a;
    input [1:0] b;
    output [1:0] o1, o2;
two M0 (o1[0], o2[0], a[0], b[0]);
two M1 (o1[1], o2[1], a[1], b[1]);
endmodule

module two (s1, s2, a1, b1);
    input a1;
    input b1;
    output s1, s2;
    xor (s1, a1, b1);
    and (s2, a1, b1);
endmodule
```

- The statement two m0(o1[0], o2[0], a[0], b[0]) written in module one binds module two to module one.
- o1[0] is the o/p of two i/p xor gate with a[0] & b[0] as the inputs
- o2[1] is the output of a two input AND gate with a[1] & b1[1] as inputs