HDL - Hardware Descreptive Language

- describe struct & behaviour of electronic ckts
- can execute parallel blocks
- .v file extension (verilog)
- case sensitive (verilog)

→ **VERILOG :**

- levels of Abstraction : various layers of complexity in a logical device

eg. logic cells → instruction set architecture → machine code → assembly prog

① **switch level :**

module can be implemented in terms of switches.

syntax:

mos_name instance_name (o/p, data, control)

```
module inverter (Q,A);
input A;
 output Q;
supply1 vdd;
supply0 vss;

pmos p (Q, vdd, A);
nmos n (Q, vss, A);
endmodule
```

② Gate level

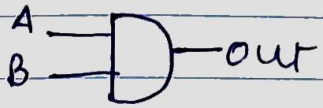
- module is implemented in terms of logic gates
- gate level - lowest level of abstraction
- Basic logic gates - available as predefined primitives

Syntax:

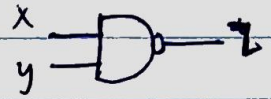
primitive_name instance_name (output, inputs)

// primitive_name: name of logic gate

Eg. and G1 (out, A, B);
nand G (z, x, y);
Or G_2 (C, A, B);



③ Data flow level



- "assign" keyword is used

④ Behavioral Level

- Highest level of abstraction
- function, task, blocks can be used
- 2 imp. constructs : initial and always.

→ MODULES

- Basic Building block

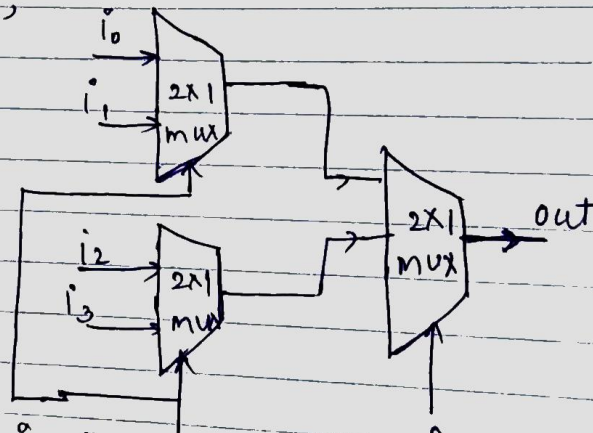
- Verilog framework :

```
module module_name (x, y, z); // port declaration
 input x, y; // assigning direct^n to ports.
 output z;
 statements; // explain functionality of ckt at
                any level
end module.
```

- Module instantiation -
- creating copies of module
- similar to creating functions and C++ and calling them in main ftn.
- can't define ~~one~~ another module inside preexisting mod in verilog.

Eg- 4x1 mux using 2x1 mux

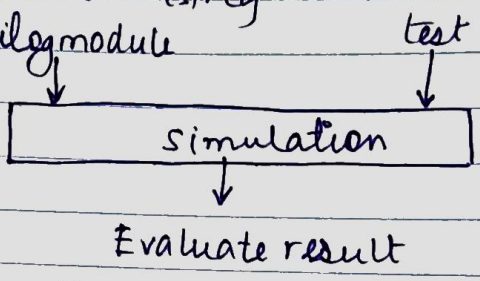
```
module mux_2to1 (i0, i1, select, out); // define module for 2x1 mux
 input i0, i1, select;
 output out;
 always @ (i0, i1, select);
 begin
   if (select)
     out = i1;
   else
     out = i0;
 end
 end module
```




```
module mux_4to1 (i0, i1, i2, i3, s1
                 s0, s1, out);
 Input i0, i1, i2, i3, s1, s0;
 output out;
  wire x1, x2;
```

```
// instantiating mux_2to1
// somewhat like function overloading
mux_2to1 m1 (i0, i1, s1, x1);
mux_2to1 m2 (i2, i3, s1, x2);
mux_2to1 m3 (x1, x2, s0, out);
// use same order of ports as defined mod
end module
```

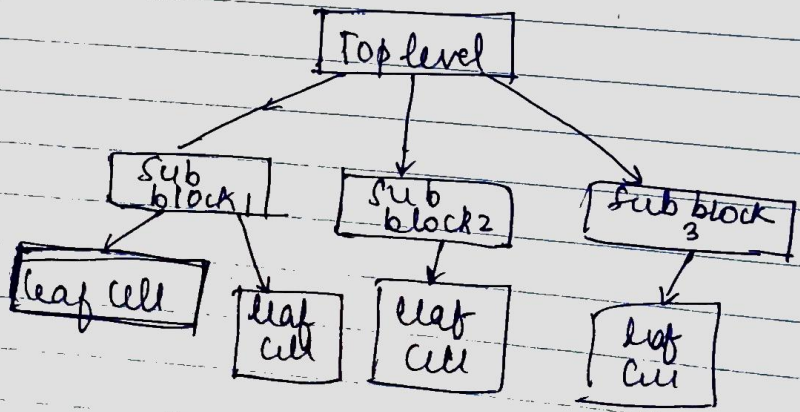
→ Simulation : verify functionality of digital design that is modelled using HDL



→ Synthesis : digital design modeled using HDL is translated into an implementation consisting logic gates.

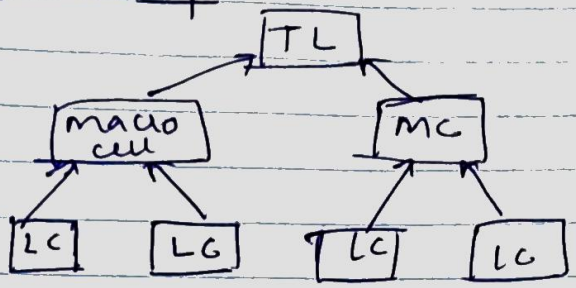
→ Design Methodology

① Top down



- Input & output declarations declare a wire unless specified.

② Bottom up



→ Data Types

① Register data types
- holds value
- Keyword: reg
- reg, integer, real, time etc.

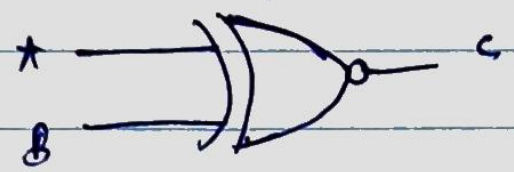
② Net data type

- represents connections b/w hardware elements
- does not store const. values i.e. continuosly driven.
- Keyword: wire
- default value : z
- wire, wand, wor, tri, triand, trior, trireg etc.

→ Values & Signal Strength

• unconnected wire → high impedance → z

- negation : ~ {assign out = ~a;}
- OR : | {eg. assign out = a | b;}
- AND : & { assign out = a & b; }
- XOR : ^ ; XNOR : ~^
- XNOR gate ;



XOR $C = A \oplus B = (A+B).(\overline{A}+\overline{B})$

A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

XNOR $\Rightarrow C = \sim(A \oplus B)$