# UNIT - IV

## 6) Switch level modeling

### Introduction:-

- mos transistor is the basic element in VLSI
- Verilog has the provision to do the design description at the switch level using mos transistors.
- Switches level modeling forms the basic level of modeling digital circuits.
- The Switches are available as primitives in verilog.
- Basic gates can be defined in terms of such Switches
- By Repeated and Successive instantiation of these Switches, more ckts can be modeled.

### Basic Transistor Switches:-

(a) Consider Enhancement - mode Nmos transistor, which have 3 - modes of operation.

- It is off when $V_g \cong V_s$.
- It is moderately ON (or) Active when $V_G$ is Slightly greater than $V_s$. representing the Resistive mode of operation.
- when $V_g$ is Sufficiently greater than $V_s$, the transistor is in the ON - state representing the very low resistance ($\sim 0\Omega$)

(b) Consider depletion - mode Nmos transistor, which also have 3 - modes of operation,

- when $V_g < V_s$ w.r.to drain. The transistor is off and offers very high impedence (Z state)

• When $V_g \cong V_s$, the transistor will be in the active region. It presents resistance b/w D & S

• When $V_g > V_s$, the transistor is fully turned 'ON' which presents very low resistance (10Ω) b/w Source & drain.

→ Similarly the modes are present for pmos transistor also. All these are summarized in a table as follows.

| mode | | Nmos | | Pmos | |
|---|---|---|---|---|---|
| | | Depletion | Enhancement | Depletion | Enhancement |
| Applied voltage for normal operation (Range: 1.5V to 5V) | | positive | positive | Negative | Negative |
| Range of $V_g - V_s$ for | OFF(Z) state | Negative | $\cong 0$ | positive | positive |
| | Resistive state (pull1, pull0) | $\cong 0$ | rigidly positive | $\cong 0$ | rigidly negative |
| | ON(10Ω) state | positive | fully positive | negative | fully negative |

## Basic Switch primitives:-

Different Switch primitives are available in verilog

(a) Consider an Nmos Switch

It is instantiated as

```
nmos (out, in, control);
```

Here nmos is a keyword, where nmos acts as Switch. it has 3 terminals in, out and control.

→ When control=1 (high) the Switch is ON; with impedence and connects i/p to o/p

→ when control=0 (low) the switch is off, the output is left floating i.e, z state

– Now consider pmos switch, which also have 3 terminals

It is instantiated as

pmos (out, in, control);

The keyword pmos shows that pmos functioning as switch.

→ when control=1 (high), switch is off,

The o/p is left floating.

→ when control=0 (low), switch is ON, i/p is connected to the o/p is at same state as input.

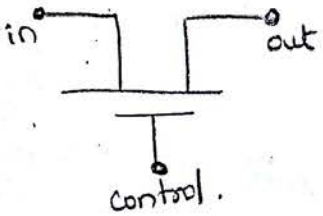



fig: nmos switch with terminals

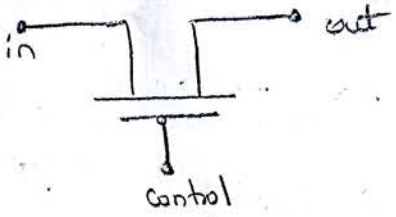



fig: pmos switch with terminals

Resistive Switches:–

nmos and pmos represent switches of low impedence in the ON-state.

–rnmos and rpmos represent the resistive counter-parts of these respectively.

— They are instantiated as follows.

rnmos (output1, input1, control1);

rpmos (output2, input2, control2);

- rnmos:-

When Control=1; the switch is ON and functions as a definite resistance. It connects input to output through resistance.

When Control=0, the switch is off and leaves output floating

rpmos:-

Here switch is ON when control=0, which inserts a definite resistance between the input and the o/p signals but retains the signal value.

Note

(1) The rpmos and rnmos are resistance switches which reduce the signal strength when 'ON'

(2) The rpmos and rnmos are unidirectional switches where signal flow is form input to output only.

pull-up and pull-down:-

A mos transistor functions as an resistive element when they are in active state.

- pullup and pull-down represent such resistance elements.
- Typically they are instantiated as follows.

pull-up (x);

here the net x is pulled up to supply1. pull-down (y);

pull-down (y);

pulls down y to supply0 level through resistance pullup (strong1) rs (x);

Specifies the resistance pullup of net x to supply1 pullup (strong1) rs (x)

represent an instantiation of pull up designated as ~~...~~ having strength strong1

Note:-

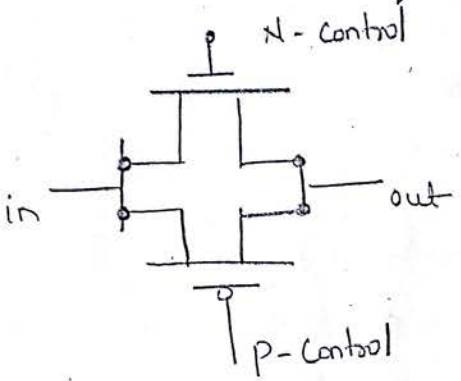
* pull-up and pull-down are functional Elements.
* 1'b0 and 1'b1 are nets. in the absence of assignments

## Cmos Switch:-

A cmos Switch is formed by connecting and an nmos Switch in parallel.

i.e, input leads are connected together and o/p leads are also connected together

- It has two control inputs




→ N- Control turns 'ON' the nmos transistor and keeps it ON when it is in the '1' state.

→ p- Control turns 'ON' the pmos transistor and keeps it ON when it is in the '0' state.

- The cmos Switch is instantiated as shown below

```
cmos csw (out, in, N-control, p-control);
```

Here

- cmos: The keyword for the Switch instantiation.
- csw: name assigned to the Switch
- out: Name assigned to the output variable
- in: Name assigned to the input variable
- N- Control: Name assigned to the Control variable of nmos transistor.

Bidirectional Gates:-

All the

- p-control: Name assigned to the control variable of pmos transis

## Bidirectional Gates:-

All the gates discussed previously are unidirectional gates (nmos, pmos, rnmos, rpmos, rcmos).

i.e when turned 'ON', these establish a connection and makes the signal at input side available at output side.

- Verilog has a set of primitives for bidirectional gates as well

They connect the net on either side when ON and isolate them when off.

i.e, the signal flow may be either directions.

There are 6-types of bidirectional gates.

### ① tran and rtran:-

The tran is a bi-directional gate of two ports. when instantiated, it connects the two ports directly. The instantiation is

tran (S1, S2);

Connects the signal lines S1 & S2. either line can be input inout or output.

- The rtran is the resistive tran

### ② tranif1 and rtranif1:-

Tranif1 is a bi-directional switch turned ON/off through a control line. It is in the ON state if Control = 1 and off state if control = 0.

Tranif1 (S1, S2, C);

Here C is the control line. S1 & S2 are connected and signal transmission can be either directions.

- rtranif1 is the resistive of tranif1

③ tranif0 and rtranif0:-

both are again bi-directional switches. The if control line is in '0' state switch is off if control line is in '1' state and ON if control line is in '0' state.

— The instantiation is

tranif0 (S1, S2, C);

• if C=0, S1 and S2 are connected and signal transmission can be in either directions.

• if C=0, S1 and S2 are isolated from each other and switch is OFF.

- rtranif0 is the resitive counterpart of tranif0.

— The Bidirectional switches are tabulated as follows

| Type of Bidirectional Switch | Typical Instantiation | Condition to be ON | Remarks |
|---|---|---|---|
| 2 port | tran (a, b); | Always ON (if instantiated) | Act essentially as a buffer |
| | rtran (a, b); | | Act essentially as a buffer with reduction in signal strength |
| 3 port | tranif1(a,b,c); | ON if c=1 | Act as a buffer 'ON' otherwise provides isolation |
| | tranif0(a,b,c); | ON if c=0 | -do- |
| | rtranif1(a,b,c); | ON if c=1 | Acts as buffer if 'ON'. otherwise provide isolation signal strength on o/p side is lower than that |
| | rtranif0(a,b,c); | ON if c=0 | — |

## Time delays with Switch primitives

Propagation delays can be defined / specified by switch primitives as like gate primitives.

eg:- Nmos Switch instantiation is

- nmos g1 (out, in, ctl); has no delay in it
- nmos (delay1) g2 (out, in, ctl); has a delay for output to rise, fall, and turn off.
- nmos (delay_r, delay_f) g3 (out, in, ctl); has delay-r as the rise time of the output delay-f is the fall-time for the o/p.

The turn-off time is zero.

- nmos (delay_r, delay_f, delay_0) g4 (out, in, ctl);

has delay_r as rise time for output, delay_f as fall time for the output, delay_0 as the turnoff time when the control signal goes from 0 to 1.

- Bidirectional switches do not delay transmission their rise time and fall times are zero.
- They can have only turn-on and turn-off delays associated with them.
- Tran has no delay associated with it.

tranif1 (delay_r, delay_f) g5 (out, in, ctl);

tranif1 (delay0) g2 (out, in, ctl);

## Instantiation with 'strengths' and delays:-

In the most general form of instantiation, strength values and delay values can be combined.

- for Eg:- The instantiation

nmos (strong1, strong0) (delay-r, delay-f, delay-o) gg ($S_1$, $S_2$, ctrl)

means

- It has strength strong0 when in the low state and strength strong1 when in the high state.
- when o/p changes from low to high, it has a delay time of delay-r
- when o/p changes state from high to low it has a delay time of delay-f.
- when o/p turn-off it has a turn-off delay time as delay-o

- rnmos, pmos and rpmos switches too can be instantiated in the general form in the same manner.

— The general instantiated for the bi-directional gates too can be done similarly.

## Strength Contention with trireg Nets:-

Net declared as trireg can have capacitive storage - such storage can be assigned one of three strengths — large, medium or small.

- Driving such a net from different sources can lead to contention.
- The relative strength levels of the sources also have signal level taken by the net.

r — The contention resolution is explained by an example as follows

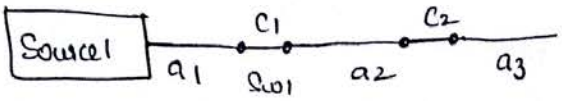



fig: ckt demonstrating contention resolution using tring net

## (b) System tasks, functions and compiler Directives

### Introduction:-

A no. of facilities manage the simulation in verilog; starting and stopping, selective monitoring, testing the design for timing constraints etc among them.

### parameters:-

often designers keep debugged modules for reuse. Such modules call for flexibility on two factors.

(a) They should be adaptable to design conforming to different technologies. Timing parameters used for testing should be flexible

(b) They should have a scalable feature; that is bus width, register size etc should be flexible.

The parameter constructs facilitate such flexibility.

def: constructs specifying timing values, ranges of variables, wires etc can be specified in terms of assigned names are called parameters

- The parameter values can be specified and changed to suit the design environment or test environment. But the assigned values cannot change during testing or synthesis.

∴ parameter is different from a net or a variable

- Two types of parameters are used in modules

i) parameters related to timing, time delays, rise time, fall time etc, are technology specific and used during simulation

- parameter values can be assigned or overridden with the keyword "specparam" preceding the assignments.

ii) parameters related to design, bus width and register size are of a different category.

- They are related to the size or dimension of specific design.
- they are technology independent.
- for this assignment (or) overridding is possible with keyword "defparam".

These two parameters are treated differently in verilog.

Timing Related parameters :-

The constructs associated with parameters are discussed here through specific design or test modules.

```
module ha_1 (s, ca, a, b);
  input a, b;
  output s, ca;
xor #(1,2) (s, a, b);
and #(3,4) (ca, a, b);
Endmodule
```

- The test bench for this is written as

```
module tstha_1();
reg a, b; wire s, ca;
ha_1 hh(s, ca, a, b);
 initial
   begin
    a=0; b=0;
   End
 always
    begin
  #5  a=1; b=0;
  #5  a=0; b=0;
  #5  a=1; b=1;
  #5  a=0; b=0;
   End
Initial $monitor ($time, "a=%b, b=%b, outsum=%b
                  a, b, s, ca);
```

```
initial
    #30 $stop;
    Endmodule
```

—Here

- a=0, b=0 at state of Simulation. Because of the transition times, the outputs remain indecisive at the 'x' state.
- The Sum bit falls down to '0' state with the specified delay of 2nsec. The carry bit falls down to 0 state with delay of 4nsec
- a=1 and b=0 at 5nsec. The Sum bit rises to 1 state at 6nsec
- a=0 and b=1 at 10nsec. But the Sum and carry bits remain unchanged.
- a=b=1 at 15nsec. The Sum bit falls to '0' at 17nsec and carry bit rises to '1' state at 18nsec.

parameter Declarations and Assignments:-

Declarations of parameters in a design as well as assignments to them can be effected using the keyword "parameter". The declaration is done as

```
parameter alpha=a, beta=b;
```

Where

- parameter is a Keyword
- alpha and beta are names of two parameters
- a, b are values assigned to alpha & beta.

— In general a and b can be contant expressions.

* The parameter value can be overridden during instantiation but cannot be changed during the run-time.

— If a parameter assignment is made through the keyword "localparam" its value cannot be overridden

Note:-

* parameters are constants which can be altered during Compila... but not during runtime.

* A parameter can be signed or unsigned; it can be an integer or a real number.

* It nature - signed or not, real or integer type as well as range can be specified at the time of declaration (or) decided by default based on assignment.

Examples

→ parameter a=3; // a is a positive integer

→ parameter b=-3; // b is a negative integer

→ parameter c=3.0, d=3.0e2; // c and d are unsigned real number

In all the above cases the parameter range an type are decided by default

→ parameter integer e=3; // e is declared as integer type which assigned a +ve value

Here the parameter type is declared explicitly.

* When ever a parameter value is overridden during instantiation type, signed/unsigned etc., will remain unchanged

Type declarations for parameters:-

for the before program there is no type declaration statements, and parameter statement.

- So, the before program is modified using parameters as shown below.

```
module ha_2(S, Ca, a, b);
  input a, b;
  output S, Ca;
parameter dir=1, d2f = dir+1, d3r=3, d4f = d2f*2;
  xor #(dir, d2f) (S, Ca, b);
  and #(d3r, d4f) (Ca, a, b);
  Endmodule
```

- Here parameters are defined to assign delays to risetime and fall time.
- The test bench for before program is

```
module tstha_2();
  reg a, b;
  wire S, Ca;
ha_2 hh(S, Ca, a, b);
  initial
    begin
    a=0; b=0;
    end
  always
    begin
  #5 a=1; b=0;
  #5 a=0; b=1;
  #5 a=1; b=1;
  #5 a=0; b=0;
  end
initial
  $monitor ($time, "a=%b, b=%b, outcarry=%b,
            outsum=%b", a, b, Ca, S);

initial
  #30 $stop;
  Endmodule
```

- The above program have parameter d1r, d2r, d3r, d4r. but - donot have type declaration.
- However default initial value is assigned to them.
- Consider parameter assignment statement.

→ parameter d1r=1, d2f = d1r+1, d3r=3, d4f = d2f*2 all these 4 are treated as integers by Simulator but if it is modified as

→ parameter d1r=1, d2f = d1r+1.0, d3r=3, d4f = d2f*2

Here d1r and d3r are of integer type but d2f and d4f are treated as real.

- However the numerical values assigned will remain constant.

## path Delays:-

The time delays discussed so far are the delays associated with individual operations or activities in a module.

- They refer to basic ckt Elements in the design i.e, at microlevel itself.
- These are called "distributed delays"
- Verilog has a provision to specify and check delays associated with total paths - from any input to any output of a module
- Such path delays are at chiplevel or system level. They are refered to as "module path delays"
- In verilog, constructs are available to specify paths and assigning delay values to them.

## Specific Blocks:-

module paths are specified and values assigned to their delays through "specify" blocks.

- They are used to specify rise time, fall time, path del... pulse widths and the like.

- The "specify" block has the format

```
specify
    specparam rise_time = 5, fall_time = 6;
    (a => b) = (rise_time, fall_time);
    (c => d) = (6, 7);
endspecify
```

The block starts with the keyword "specify" and ends with keyword "endspecify"

- specify blocks can appear anywhere in the module.
- The block can have two types of statements.
  - One type starts with keyword "specparam" and assigns numerical values to timing parameters declared elsewhere.
  - The specparam statements can be appear anywhere with in a module or within a specific block.
  - The second type specifies paths and assigns values to time delays to them

"A specify block can have only the above type assignments".

i.e; ckt function assignments, assignments to module parameters etc are not permitted.

## module paths:-

module paths can be specified in different ways inside a specify block. This has the form

A *→ B

here A is source and B is destination. The source can be an input or inout port. The destination can be an output or inout port.

The symbol *→ specifies the path from the source to destination.

- It comprises all the possible paths from A to B.

→ If A & B are scalars, it signifies a single path.

→ If A is a vectors and B is a scalar, it signifies all the paths from every bit of A to the scalar B. thus if A is 4-bit wide vector, 4 paths are specified.

→ If A is a scalar and B is a vector, all the paths from A to every bit of the vector B.

→ If both A & B are vectors, it signifies all the possible paths from every bit of A to every bit of B.

Note:

⇒ Signifies only all the parallel paths

*> Signifies all the possible paths from source to destination.

## Conditional pin-to-pin delay:-

The pin-to-pin path of a signal may change depending on the value of another signal; in turn the no. of ckt elements in the alternate path may be different

Pulse P

— Conditional selection and assignment of path delays facili- simulation in such cases.

—A simple condition was used to illustrate conditional assignment to delay values. In a general case a conditional expression can be more involved with different logical operations performed in random, with the following restrictions.

- The expression can involve any logical reduction operation
- All bit wise logical operation can be used
- If the conditional expression evaluates to multiply bits, the LSB decides the delay.

Edge - Sensitive paths:-

Behaviour level modules can have signal paths activated following an edge in a different signal.

—Verilog has the provision to specify such delays during simula- -tion. They can be specified in a variety of ways.

—The path may get activated following a positive edge or a negative edge in a signal.

—The path delay may be specified for rise or fall in the o/p or for positive or negative polarity transitions separately.

— The delay assignment can be made conditional on an expression; such a path specification is an "Edge Sensitive state dependent path".

## Pulse filtering and its control:-

All transitions on an input pin with less than a specified module path delay are termed "pulses".

- Normally, when a module path delay is specified, all pulses are ignored; that is, the simulator does not take cognizance of such narrow transitions.
- However, response to such narrow pulses can be specified through "specparam" in a specify block.
- A statement

```
specparam PATHPULSE$(x,y) = (a,b);
```

implies the following concerning the module pulse path from x to y.

→ Ignore all pulses of width less than a nsec 'a' is refered to as the "rejection limit" for the pulse path.

→ Take cognizance of all the pulses wider than b nsec. note that the specification has relevance only if the delay value for the pulse path is larger than b.

→ for all pulses of width value between a and b, the output is in error and in 'x' state.

- The path pulse $ specification is governed by the following
  - It has to appear with in a specify block as a "specparam" assignment.
  - It signifies the limits for the path pulse – error limit as well as reject limit for the specified path.

- The statement as

specparam pathpulse$ = (a, b);

implies that a and b are the Error and reject limit for the pulse widths for all the paths specified within the specify block; the Simulator checks for the pulse width and if it is between a and b values, the o/p goes to 'x' state

- A set of statements

specparam path pulse$(x, y) = (a, b);

specparam path PATHPULSE$ = (c, d);

Implies that for the path from x to y a and b are the Error and Reject limit for the Concerned paths.

Module parameters:-

module parameters are associated with size of bus, Register, memory, ALU and Soon. They can be specified within the Concerned module but their value can be altered during instantiation.

- The alterations can be brought about through assignments made with 'defparam'. Such defparam assignments can appear anywhere in a module.

- The rules of assigning values for the module parameters, deciding their size, type etc., are all Similar to those of specify parameters.

program:- ALU module with its size declared as parameter: 4

```
module alu(d, co, a, b, cci, f);
  parameter msb=3;
output [msb:0]d; output co; wire [msb:0]d;
input cci; input [msb:0] a, b; input [1:0] f;
specify
  (a, b=>d) = (1,2);
  (a, b, cci *> co) = 1;
Endspecify
assign {co, d} = (f == 2'b00) ? (a+b+cci) : ((f == 2'b01) ?
                 (a-b) : ((f == 2'b10) ?
                 {1'bz, a^b} : {1'bz, ~a}));
Endmodule
```

System tasks and functions:-

Verilog has a no. of system tasks and functions defined in the LRM. They are for tasking output from simulation, control simulation, debugging design modules, testing modules for specifications etc.

— The $ sign preceding a word or a word group signifies a system task or a system function.

— Some of the system tasks and functions have been extensively used in the previous concepts.

— The complete list is available in the LRM

Output tasks:-

A no. of system tasks are available to output values of variables and selected messages etc., on the monitor.

— out of these $monitor and $display tasks have been extensively used.

## Display Tasks:-

The $display task, whenever Encountered, displays the arguments in the desired format, and the display advances to the new line.

- The $write task carries out the desired display but doesnot advance to the newline.
- for both, the format is identical to that of scanf and printf in 'c' language.
- The features are given as follows.
  - The arguments are displayed in the same order as they appear in the display statement.
  - The strings are output as such Except the escape sequence, an escape sequence starts with the character '\' or the character '%'
  - '\' signifies on of a set of special character. shown in the table

| Sequence | Implication |
|---|---|
| \n | -Display to advance to a newline |
| \t | Insert to tab space in the display. |
| \\ | Insert a '\' character in the display. |
| \" | Insert a " in the display. |
| \aaa | Insert an ASCII character, specified by the octal number "aaa" in the display. |
| %% | Insert a % in the string displayed |

- "%m" signifies that the hierarchical name of the particular argument is to be displayed.
- % followed by a character - as shown in below table - signifies the format for display of the following argument or an aspect of following argument.

| character combination | Implication |
|---|---|
| %h or %H | Display in Hex format |
| %d or %D | Display in decimal format |
| %o or %O | Display in octal format |
| %b or %B | Display in Binary format. |
| %c or %C | Display in ASCII character format |
| %l or %L | Display library binding information |
| %v or %V | Display net signal strength |
| %s or %S | Display as a string |
| %t or %T | Display in current time format |
| %u or %U | unformatted 2-value data |
| %z or %Z | unformatted 4-value data |
| %f or %F | Display real in decimal format |
| %g or %G | Display real in Exponential format. |

- If the format for the display of an argument is not specified, a default format is assumed. i.e, It is binary for $display b and $writeb,
  octal for $displayo and $writeo,
  decimal for $displayd and $writed,
  hex for $displayh and $writeh.

• If any argument is in the form of an expression, it is Eoo-ed and the result displayed or written; it is sized automatically

$strobe task:-

When a variable or a set of variables is sampled and its value displayed, the $strobe task can be used; It senses the value of the specified variables and displays them.

- The form of specifying arguments is identical to that of the $ display task.
- The $strobe task is executed as the last activity in the Concern-ed timestep.
- It is useful to check for specific activities and debug modules

Example program:- A module set to illustrate use of $ strobe task

```
module dff (do, di, clk);
    output do;
    input di, clk;
    specify
        (negedge clk *> (do:di) = 1;
    endspecify
    reg do;
    initial do = 1'bdo;
    always @ (negedge clk) do = di;
    endmodule
```

## $stop and $finish Tasks:-

The $stop task suspends simulation. The compiled design remains active; the simulation can be resumed through commands available in the simulator.

- In contrast $finish stops simulation, closes the simulation environment and reverts block back to operating system.

## $ Random function:-

A set of random number generator functions are available as system functions.

- one can start with a seed number (optional) and generate a random number repeatedly
- Such random number sequence is most used for testing
- If the seed is not changed with every simulation, the same sequence of random numbers are generated.
- If the seed is changed, the values in the random number sequence too change.
- If the seed is not specified, the $random function uses a default seed and generates the random number.
- only the least 8-bits of the random number generated are used to assign values to a 8-b here.

## File Based Tasks and functions:-

VLSI has the provision to accommodate and integrate design and test modules kept in different files.

- It makes room for structuring the design in an elegant manner and developing it with a "cross-functional" approach.
- To carry out any file-based task, the file has to be opened, reading, writing etc completed and the file closed. The keywords

The testbench for the above program is written as

```
module tst_diff ();
  reg di, clk;
   wire do;
 diff_ d1 (do, di, clk);
  initial
    begin
    clk=0; di = 1'b0; #35 $stop; end
always
  #3 clk = ~clk;
always
  #5 di = ~di;
initial $monitor ($time," clk = %b, di = %b, do = %b", clk, di, do);
initial #9 $strobe ("at time %t, di = %b, do = %b", $time, di, do);
Endmodule
```

$monitor Task:-

The $monitor task has been used Extensively.

- The form of specifying arguments is same to $display.
- only one $monitor task can be active at a time
- $monitor task is activated and displays the arguments specified whenever any of the arguments changes. This includes $time, $stime and $real-time tasks.
- $monitoroff and $monitoron are two additional tasks assigned to the $monitor task; they are useful to Enable and disable the monitoring activity. i.e, turns off and turnson the monitoring at specified time.

## Compiler Directives :-

A no. of compiler directives are available in verilog. They allow for macros, inclusion of files and time scale - related parameters for simulation.

- All compiler directives are preceded by the "`" (accent grave) character.
- Representative compiler directives are explained below.

### (a) `define directive :-

The `define directive is for macro substitution.

- It substitutes the macro by a defined text.
- Hence the macro name can be used in place of such a group of characters in the listing wherever the group is to appear.
- The `define directive is used to define and associate the desired text with the macro name.
- The `define compiler directive can also be used to substitute an number by a macro name. It allows for deciding bus-width, specific delay values, etc., at compilation time.

### Time_Related Tasks :-

A set of compiler directives and system tasks relate to the running time of simulation as well as the delays in the concerned modules.

- A wide range of timescales as well as precision level are available for selection during simulation.

1. timescale

for all file-based tasks start with the letter f, to distinguish them from the other tasks.

— A typical sequence of activities to write to a file can be shown in table as follows.

| Line in module listing | Significance |
|---|---|
| Integer fileno; | file no is declared as Integer |
| file no: $fopen ("info.txt"); | A file with a name "info.txt" is opened. The value of file no signifies the same. |
| $ display (file no, "string" argument); | The arguments are displayed as specified. |
| $fclose (fileno); | The file is closed |

Observations:—

- The listing lines used need not be contiguous but have to be in the same sequence.
- All the system tasks to output information can be used to output to a file.

$display, $strobe, $monitor etc are of this category. The respective keywords to output to a file are $fdisplay, $fstrobe, $fmonitor respectively.

- The 1st field of the task statement is an argument the file descriptor. The subsequent fields are identical to the corresponding non-file task.
- The specified file will be opened and sustained in the directory of the Executable file of th

- Verilog has the provision to access each such item in a unique and hierarchical manner.

- Due to its importance, we has to understand the mode of deciding the hierarchical name and accessing each item.

- The hier access is explained in the following program along with the simulation results.

```
module hier_a;
integer aa, bb, cc, pp, qq, rr;
 initial
   begin: alpha
     aa = 2;  bb = 3;
     cc = fad (aa, bb);
   $ display ("fad.a = %0d, fad.b = %0d, fad.fad = %0d",
              fad.a, fad.b, fad.fad);

   End
   initial
     begin: beta
       pp = 4;  qq = 6;
       rr = fad (pp, qq);
   $ display ("fad.a = %0d, fad.b = %0d, fad.fad = %0d",
              fad.a , fad.b , fad.fad);

     End
 function integer fad;
    input [7:0] a, b;
     fad = a + b;
    End function
      End module
```

Simulation Results:-

```
# fad.a = 2, fad.b = 3, fad.fad = 5
  fad.a = 4, fad.b = 6, fad.fad = 10.
```

- Verilog has the provision to access each such item in a unique and hierachical manner.
- Due to its importance, we has to understand the mode of deciding the hierarchical name and accessing each item.
- The hier access is explained in the following program, along with the simulation results.

```
module hier_a;
integer aa, bb, cc, pp, qq, rr;
 initial
   begin: alpha
     aa = 2;  bb = 3;
     cc = fad (aa, bb);
   $display ("fad.a = %0d, fad.b = %0d, fad.fad = %0d",
              fad.a, fad.b, fad.fad);

  End
  initial
    begin: beta
      pp = 4;  qq = 6;
      rr = fad (pp, qq);
  $display ("fad.a = %0d, fad.b = %0d, fad.fad = %0d",
      fad.a , fad.b , fad.fad);

    End
function integer fad;
   input [7:0] a, b;
   fad = a + b;
   End function
     End module
```

Simulation Results:-

```
# fad.a = 2, fad.b = 3,  fad.fad = 5
# fad.a = 4, fad.b = 6,  fad.fad = 10.
```

Whenever "us" is to be specified for defining or changing time's scale, it is specified as "us" Remaining all are same (s, ms, ns, ps, fs etc.

Simulation time :-

Simulation time value can be obtained, displayed or used in specific expressions, a limited amount of flexibility is available here.

- $time returns the value of simulation time as integer
- $realtime returns the value of simulation as real number.

Default timescale :-

If the time scale values are not specified in the source file, simulation is carried out with the default values specified in the tool used for Simulation.

- The default value of time unit is taken as nano-second.

Hierarchical Access :-

A verilog design will normally have a module or two at the apex level.

- A number of modules and UDP's will be instantiated within it.
- They can have other instantiations with in them.
- They can also have functions and tasks defined in them and invoked Repeatedly.
- In addition, begin-end and fork-join blocks too may be present
- All these represent identified functional blocks in a design.
- The Despite the variety here, we should have access to every variable, net as well as named Identity in a design.
- The access can be to sample and display the values, to change specific parameters or disable selected blocks.

- The 'timescale compiler directive allows the time scale to be specified for the design, when a 'timescale directive is encountered in a file, the same is valid for all subsequent modules within the file.
- The 'timescale directive has two components
  1. 'timescale 1ms/100us → implies that in the following design all the time values specified are in msecs and they have precision of 100usec.
  2. 'timescale 10ms/100us → implies that in the following design all the timescales are specified as multiples of 10msec and with precision of 100usec.
  3. 'timescale 1ms/1ms → implies that in the following design all the time values specified are in msec and they have a precision of 1msec.

$ time format:-

The timescale and the format for display can be changed during simulation with the help of $timeformat task.

- The syntax for the task is shown below

```
$timeformat (-aa, bb, "cc", dd);
```

Where -aa; represent a negative number in the 0 to -15 range signifying time unit.

bb; represent an integer specifying precision,

"cc"; represents any convenient string to be displayed

dd; represents an integer specifying field width of display.