SESSION #4

COMPILER DIRECTIVES

Compiler Directives

Verilog has compiler directives which affect the processing of the input files. The directives start with a grave accent (`) followed by some keyword. A directive takes effect from the point that it appears in the file until either the end of all files or until another directive that cancels the effect of the first one is encountered.

- Compiler Directives are tool specific.
- A list of standard Compiler Directives are shown below:

Example:

`resetall - resets all compiler directives to default values

`define - text macro substitution

`timescale 1ns / 10ps - specifies time unit/precision

`ifdef, `else, `endif - conditional compilation

`include - file inclusion

Macro Definitions
A macro is an identifier that represents a string of text. Macros are defined with the directive `define, and are invoked with the quoted macro name as shown in the example.

Syntax

`define macro_name text_string;

. . . `macro_name . . .

Include Directive

Include is used to include the contents of a text file at the point in the current file where the include directive is.

Syntax

`include file_name;

Time Scale

`timescale specifies the time unit and time precision. A time unit of 10 ns means a time expressed as say #2.3 will have a delay of 23.0 ns. Time precision specifies how delay values are to be rounded off during simulation. Valid time units include s, ms, µs, ns, ps. Only 1, 10 or 100 are valid integers for specifying time units or precision. It also determines the displayed time units in display commands like $display

Syntax

`timescale time_unit / time_precision;

Conditional compilers

- The `ifdef compiler directive checks for the definition of a text_macro_name. If the text_macro_name is defined, then the lines following the `ifdef directive are included. If the text_macro_name is not defined and an `else directive exists, then this source is compiled.
- The `ifndef compiler directive checks for the definition of a text_macro_name. If the text_macro_name is not defined, then the lines following the `ifndef directive are included. If the text_macro_name is defined and an `else directive exists, then this source is compiled.
- If the `elsif directive exists (instead of the `else) the compiler checks for the definition of the text_macro_name. If the name exists the lines following the `elsif directive are included. The `elsif directive is equivalent to the compiler directive sequence `else `ifdef ... `endif. This directive does not need a corresponding `endif directive. This directive must be preceded by an `ifdef or `ifndef directive.

Conditional Execution

- Conditional execution flags allow the designers to control statement execution flow at runtime.
- All statements are compiled but executed conditionally.
- It can only be used for behavioral statements.
- $test$plusargs
- $value$plusargs

`define macros

- Guideline: only use macro definitions for identifiers that clearly require global definition of an identifier that will not be modified elsewhere in the design.
- Alternate Guideline: place all macro definitions in the top-level testbench module and read this module first when compiling the design.
- Guideline: do not use macro definitions to define constants that are local to a module
- Guideline: make clock cycle definitions using the `define compiler directive
- Reason: Clock cycles are a fundamental constant of a design and testbench. The cycle of a common clock signal should not change from one module to another; the cycle should be constant!
- vlog +define+WIDTH=8 adder_testbench.v
- vsim -64 -voptargs=+acc -c tb -do "log -r /*;run -all"

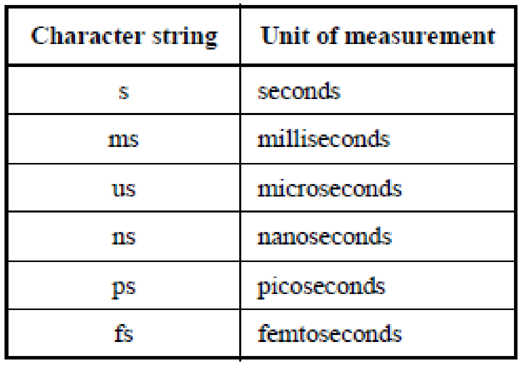
```
`define CYCLE 10
module tb_cycle;
// ...
initial begin
clk = 1'b0;
forever #(`CYCLE/2) clk
= ~clk;
end
// ...
endmodule
```

`timescale Definition

- The `timescale directive gives meaning to delays that may appear in a Verilog model. The timescale is placed above the module header and takes the form:`timescale time_unit / time_precision

- The time_unit argument specifies the unit of measurement for times and delays.
- The time_precision argument specifies how delay values are rounded before being used in simulation.
- The time_precision argument shall be at least as precise as the time_unit argument; it cannot specify a longer unit of time than time_unit

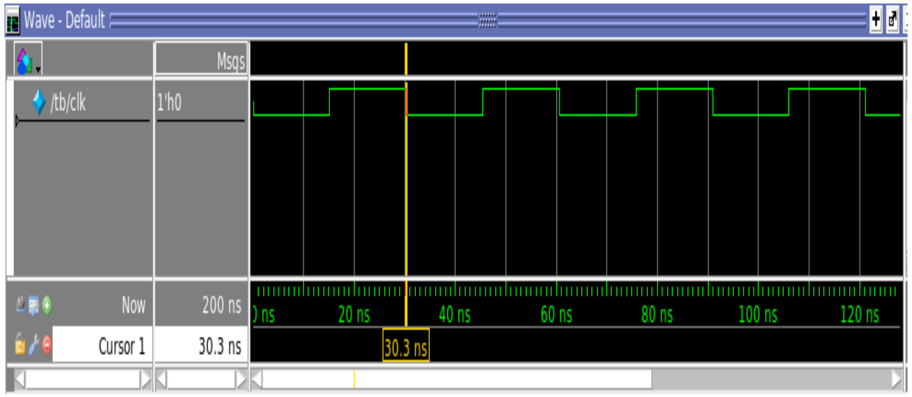
```
`timescale 10 ns / 1 ns
module test;
reg set;
parameter d = 1.55;
initial begin
#d set = 0;
#d set = 1;
end
endmodule
```

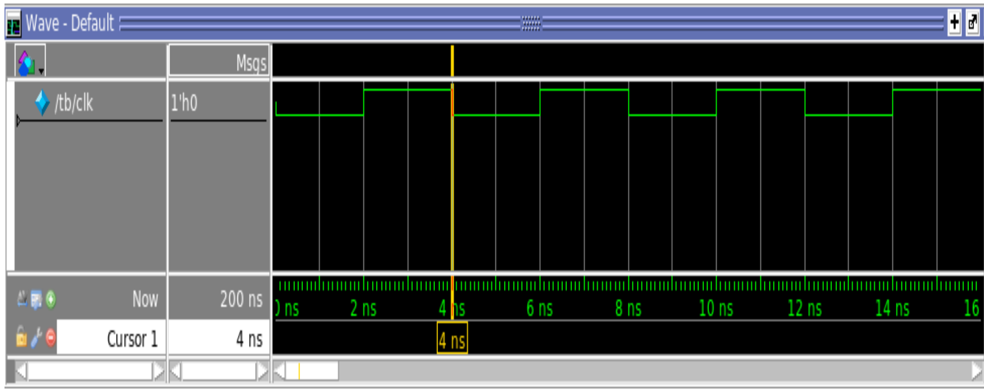


Character string	Unit of measurement
s	seconds
ms	milliseconds
us	microseconds
ns	nanoseconds
ps	picoseconds
fs	femtoseconds

```
//33MHZ Clock Generation
`timescale 1ns/1ps
module tb();
reg clk;

initial clk=0;

always #15.15 clk=~clk;

initial
begin
#200;
$finish;
end

endmodule
```

```
//250 MHZ Clock Generation
`timescale 1ns/1ps
module tb();
reg clk;

initial clk=0;

always #2 clk=~clk;

initial
begin
#200;
$finish;
end

endmodule
```





ifdef, `else, `endif

```
`define WIDTH 3
module tb();
reg [`WIDTH : 0]a,b;
reg sel;
reg [`WIDTH : 0] d;

wire [`WIDTH : 0] c;

`ifdef cont_stat
        begin
        assign  c=sel ? a:b;
        end

`elsif proc_stat
always@(*)
   if(sel!=1)
        begin
                d=a;
        end
  else
        begin
                d=b;
        end

 `else
        initial $display("NOTHING TO DO WITH THIS
CONDTION");

`endif
```

```
initial
                $monitor("%t \t a=%d \t b=%d \t c=%d \t d=%d \t sel=%d",
$time,a,b,c,d,sel);
        initial
        begin
                sel=1;a=3;b=4;
                #10;
                sel=0;a=2;b=5;
                #10;
                sel=1;a=7;b=4;
                #10;
                sel=0;a=6;b=3;
        end
endmodule
```

```
Loading work.tb(fast)
# log -r /*
# run -all
#              0        a=1     b=0     c=1     d=x     sel=1
#             10        a=0     b=1     c=1     d=x     sel=0
#             20        a=1     b=0     c=1     d=x     sel=1
#             30        a=0     b=1     c=1     d=x     sel=0
VSIM 3>
```

```
vlog  +define+cont_stat   +define+WIDTH=0   if_def_mux.v
```

SYSTEM TASKS & FUNCTIONS

System Tasks and Functions

- Operations such as displaying on screen monitoring values of nets,stopping ,finishing simulations etc, their names begin with a “dollar sign ($)” “($)<keyword>”.

- The Verilog HDL Compiler/Design Compiler and many other synthesis tools parse and ignore system functions, and hence can be included even in synthesizable models.

$display, $strobe, $monitor, $write:

$display, $write - utility to display information

$display ("format_string", par_1, par_2, ...); handle1=$fopen("filenam1.suffix");

$write(“register values”,reg1,reg2,reg3); handle2=$fopen("filenam2.suffix");

$fdisplay, $fwrite - write to file

$fdisplay(handle2, format, variable list) //write data into filenam2.suffix

$fwrite(handle2, format, variable list) //write data into filenam2.suffix all on one line. Put in the format string where a new line is desired.

$strobe, $fstrobe - display, write simulation data

$strobe ("format_string", par_1, par_2, ...);

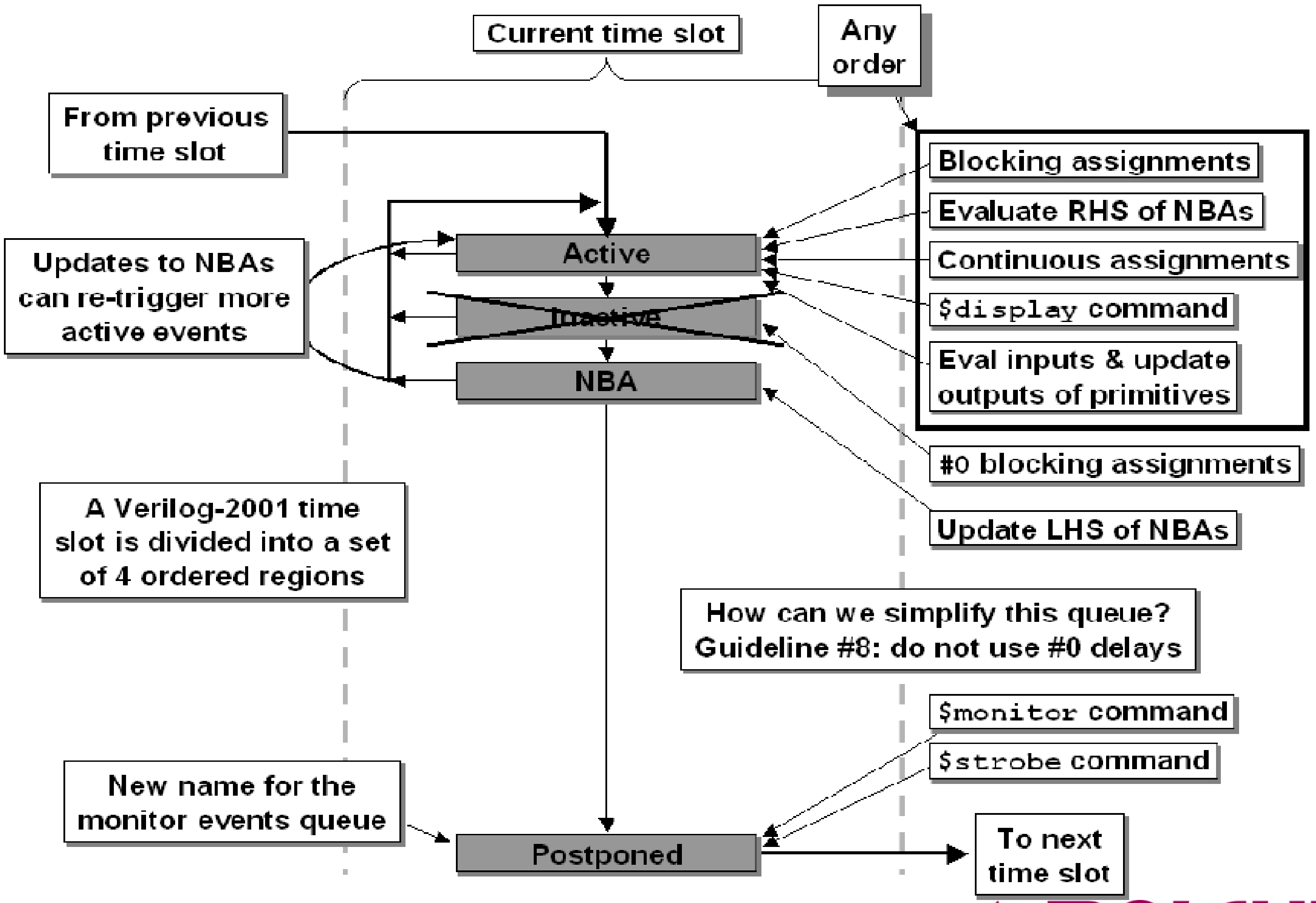
$fstrobe(handle1, format, variable list) //strobe data into filenam1.suffix

$monitor, $fmonitor - monitor, display/write information to file

$monitor ("format_string", par_1, par_2, ...);

$fmonitor(handle1, format, variable list) //monitor data into filenam1.suffix

Verilog Stratified Event Queue

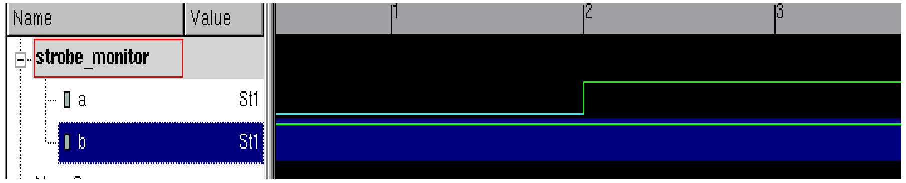

Current time slot
Any order
From previous time slot
Blocking assignments
Evaluate RHS of NBAs
Continuous assignments
$display command
Eval inputs & update outputs of primitives
Active
~~Inactive~~
NBA
Updates to NBAs can re-trigger more active events
#0 blocking assignments
Update LHS of NBAs
A Verilog-2001 time slot is divided into a set of 4 ordered regions
How can we simplify this queue? Guideline #8: do not use #0 delays
$monitor command
$strobe command
New name for the monitor events queue
Postponed
To next time slot

Example : strobe_monitor.v

```
module strobe_monitor;
reg a,b;
initial begin
b=1'b1;//Execute in the Active Region
a <=1'b0;//Execute in the Postpone region.
#2 a <=1'b1;
#4 $finish; end
initial $display($time,"DISPLAY:::a=%b \t b=
%b",a,b);
initial $monitor($time,"MONITOR:::a=%b \t b=
%b",a,b);
initial $strobe($time,"STROBE:::a=%b \t b=
%b",a,b);
endmodule
```

Output:

```
0DISPLAY:::a=x b=1 //Executes in
Active region,'a' will get the assigned
value in postponed region(which at the
end current simulation)
0MONITOR:::a=0 b=1
0STROBE::: a=0 b=1
2MONITOR:::a=1 b=1
```



$time, $stime, $realtime

These return the current simulation time as a 64-bit integer, a 32-bit integer, and a real number, respectively.

$time, $realtime -They return current simulation time where ever they are used.

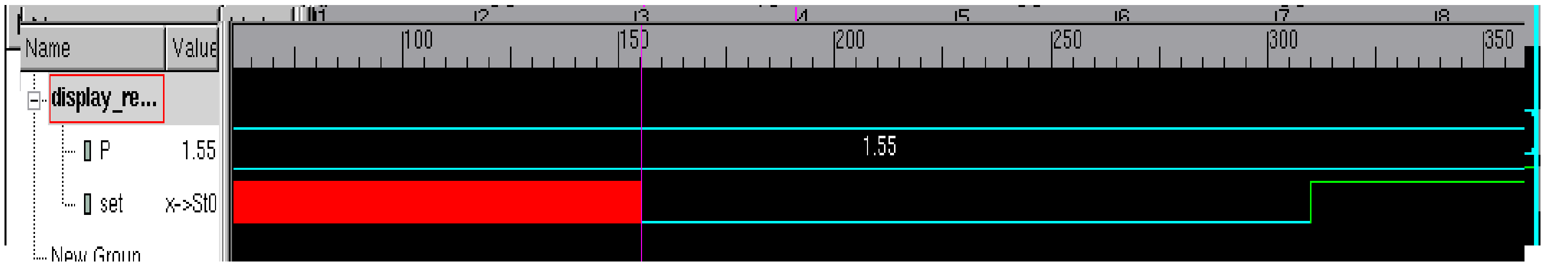
```
$time;

$display ($time, "...");

$realtime;

$display ($realtime, "...");

$stime;

$display ($stime, "...");
```

$time Example

```
Example: display_time.v
`timescale 10ns/1ns
module display_time();
        reg set;
        parameter P = 1.55;

        initial
        begin
                $monitor($time,"set = %b",set);
                #P set =0;
                #P set =1;
        end
endmodule
/*Output: with time scale 10ns/1ns
 0set = x
 2set = 0
 3set = 1
*/
```

```
module display_time();
        reg set;
        parameter P = 1.55;

        initial
        begin
                $monitor($time,"set = %b",set);
                #P set =0;
                #P set =1;
        end
endmodule
/* Output: Without timescale
 0set = x
 2set = 0
 4set = 1
*/
```

```
`timescale 10ns/100ps
module display_time();
        reg set;
        parameter P = 1.55;

        initial
        begin
                $monitor($time,"set = %b",set);
                #P set =0;
                #P set =1;
        end
endmodule
/*Output: with time scale 10ns/100ps
 0set = x
 2set = 0
 3set = 1
*/
```

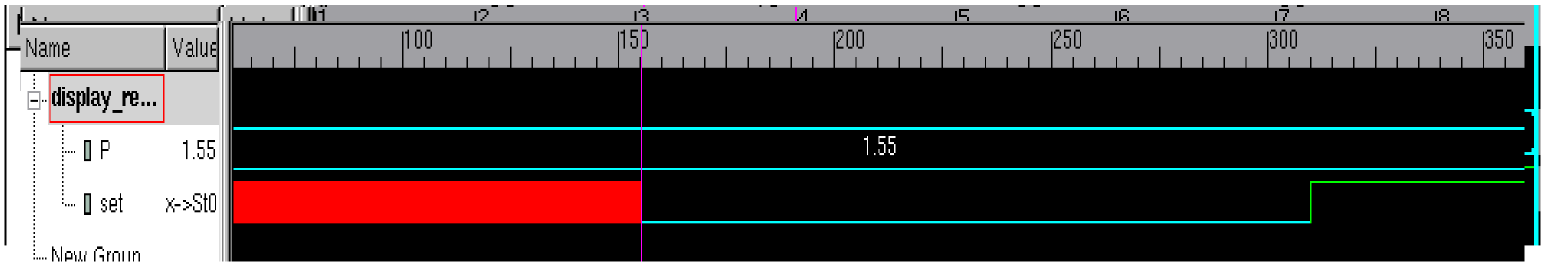


$realtime Example

```
Example: display_realtime.v
`timescale 10ns/1ns
module display_time();
        reg set;
        parameter P = 1.55;

        initial
        begin
                $monitor($realtime,"set =
%b",set);
                #P set =0;
                #P set =1;
        end
endmodule
/* Output: with timescale 10ns/1ns
0set = x
1.6set = 0
3.2set = 1
*/
```

```
module display_realtime();
        reg set;
        parameter P = 1.55;

        initial
        begin
                $monitor($realtime,"set =
%b",set);
                #P set =0;
                #P set =1;
        end
endmodule
/* Output: without any timescale
0set = x
2set = 0
4set = 1
*/
```

```
`timescale 10ns/100ps
module display_realtime();
        reg set;
        parameter P = 1.55;

        initial
        begin
                $monitor($realtime,"set =
%b",set);
                #P set =0;
                #P set =1;
        end
endmodule
/* Output: with timescale 10ns/100ps
0set = x
1.55set = 0
3.1set = 1
*/
```



Simulation control Tasks

$finish - exit the simulator

Delay $finish;

$stop - stop the simulator $stop;

$reset - resets the simulation back to time 0;$reset;

Timing scale Tasks

The **$printtimeformat** system task is used when information about time units and precision is needed.

When the $printtimescale system task is invoked without an argument, the time unit and precision of the current modules are displayed.

$printtimescale [(hierarchical_path)] ;

The **$timeformat** system task specifies the %t format specification until the next `timescale compiler directive occurs.

initial $timeformat(-9, 5, " ns", 10);

$timeformat [(unit_number, precision, suffix, min_width)] ;

$scale

scaling timeunits from another module. Sytax : $scale(hier_id) ;Scale "foreign" time value

$scope

$scope sets the current hierarchical scope to hierarchy_name.
Syntax:$scope(hierarchy_name);
example: $scope module top

$showscopes(n): lists all modules, tasks and block names in
(and below, if n is set to 1) the current scope.
Syntax:$showscopes (n);

$list : lists line-numbered source code of the named module, task, function or named
block.
Sytax:$list (hierarchical_name);

Miscellaneous System Tasks

$deposit

$deposit sets a net to a particular value, overwriting what was put there by the “circuit”. Good for test

benches.

Syntax:

$deposit (net_name, value);

$random

$random generates a random integer every time it is called. If the sequence is to be repeatable, the

first time one invokes random give it a numerical argument (a seed). Otherwise the seed is derived

from the computer clock.

Syntax:

```
reg [3:0] xyz;
initial begin
xyz= $random (7); // Seed the generator so number
end
```

Random number generation

```
module random_ex();
        reg [31:0] a,b,x;
        initial begin
                a = $random(7);
                b = $random(7);
                repeat(2)x = $random;
        end
        initial begin
                $monitor("value of a = %d \n value of b = %d \nvalue of x = %d",a,b,x);
        end
endmodule
/* Output:
value of a = 2147967488
 value of b = 2147967488
value of x =  760705370// how many times you simulate you will get constant result
*/
```

Thank You