Digital Design with Verilog

Verilog

Lecture 15: Useful Modeling Techniques

Learning Objectives

- force, and release procedural assignment statements
- Understand how to override parameters at the time of module instantiation.
- Explain conditional compilation and execution of parts of the Verilog description.
- Identify system tasks for file output, displaying hierarchy, strobing, random number generation, memory initialization, and value change dump.

Procedural Continuous Assignments

- **force** and **release**
  - They can be used to override assignments on both registers and nets.
  - force and release statements are typically used in the interactive debugging process, where certain registers or nets are forced to a value and the effect on other registers and nets is noted.
  - It is recommended that force and release statements not be used inside design blocks.
  - They should appear only in stimulus or as debug statements.

force and release on registers

- The register variables will continue to store the forced value after being released but can then be changed by a future procedural assignment.

```
module stimulus;
...
...
//instantiate the d-flipflop
edge_dff dff(Q, Qbar, D, CLK, RESET);
...
...
initial
begin
//these statements force value of 1 on dff.q between time 50 and
//100, regardless of the actual output of the edge_dff.
#50 force dff.q = 1'b1; //force value of q to 1 at time 50.
#50 release dff.q; //release the value of q at time 100.
end
...
...
endmodule
```

force and release on nets

- force on nets overrides any continuous assignments until the net is released.
- The net will immediately return to its normal driven value when it is released.
- A net can be forced to an expression or a value.

```
module top;
...
...
assign out = a & b & c; //continuous assignment on net out
...
initial
#50 force out = a | b & c;
#50 release out;
end
...
...
endmodule
```

Overriding Parameters

- Parameter values can be overridden when a module is instantiated.

```
//define module with delays
module bus_master;
parameter delay1 = 2;
parameter delay2 = 3;
parameter delay3 = 7;
...
<module internals>
...
endmodule
//top-level module; instantiates two bus_master modules
module top;
//Instantiate the modules with new delay values
//Parameter value assignment by ordered list
bus_master #(4, 5, 6) b1(); //b1: delay1 = 4, delay2 = 5, delay3 = 6
bus_master #(9, 4) b2(); //b2: delay1 = 9, delay2 = 4, delay3 =
7(default)
//Parameter value assignment by name
bus_master #(.delay2(4), delay3(7)) b3(); //b2: delay2 = 4, delay3 = 7
//delay1=2 (default)
// It is recommended to use the parameter value assignment by name
// This minimizes the chance of error and parameters can be added
// or deleted without worrying about the order.
endmodule
```

Conditional Compilation and Execution

- Conditional compilation can be accomplished by using compiler directives `ifdef, `ifndef, `else, `elsif, and `endif.

```
//Conditional Compilation
//Example 1
'ifdef TEST
//compile module test only if text macro TEST is defined
module test;
...
...
endmodule
'else //compile the module stimulus as default
module stimulus;
...
...
endmodule
'endif //completion of 'ifdef directive
```

Conditional Compilation Cont'd.

```
//Example 2
module top;
bus_master b1(); //instantiate module unconditionally
'ifdef ADD_B2
bus_master b2(); //b2 is instantiated conditionally if text macro
//ADD_B2 is defined
'elsif ADD_B3
bus_master b3(); //b3 is instantiated conditionally if text macro
//ADD_B3 is defined
'else
bus_master b4(); //b4 is instantiate by default
'endif
'ifndef IGNORE_B5
bus_master b5(); //b5 is instantiated conditionally if text macro
//IGNORE_B5 is not defined
'endif
endmodule
```

Conditional Execution

- Conditional execution flags allow the designer to control statement execution flow at run time.
- All statements are compiled but executed conditionally.
- Conditional execution flags can be used only for behavioral statements.
- The system task keyword $test$plusargs is used for conditional execution.

Conditional Execution Cont'd

```
//Conditional execution
module test;
...
...
initial
begin
if($test$plusargs("DISPLAY_VAR"))
$display("Display = %b ", {a,b,c} ); //display only if flag is
set
else
//Conditional execution
$display("No Display"); //otherwise no display
end
endmodule
```

Conditional Execution Cont'd

```
//Conditional execution with $value$plusargs
module test;
reg [8*128-1:0] test_string;
integer clk_period;
...
...
initial
begin
if($value$plusargs("testname=%s", test_string))
$readmemh(test_string, vectors); //Read test vectors
else
//otherwise display error message
$display("Test name option not specified");
if($value$plusargs("clk_t=%d", clk_period))
forever #(clk_period/2) clk = ~clk; //Set up clock
else
//otherwise display error message
$display("Clock period option name not specified");
end
//For example, to invoke the above options invoke simulator with
//+testname=test1.vec +clk_t=10
//Test name = "test1.vec" and clk_period = 10
endmodule
```

Time Scales

- Verilog HDL allows the reference time unit for modules to be specified with the `timescale compiler directive.

Usage: `timescale <reference_time_unit> / <time_precision>

- The <reference_time_unit> specifies the unit of measurement for times and delays.

- The <time_precision> specifies the precision to which the delays are rounded off during simulation.

- Only 1, 10, and 100 are valid integers for specifying time unit and time precision.

Time Scales Cont'd.

```
//Define a time scale for the module dummyl
//Reference time unit is 100 nanoseconds and precision is 1 ns
`timescale 100 ns / 1 ns
module dummyl;
reg toggle;
//initialize toggle
initial
toggle = 1'b0;
//Flip the toggle register every 5 time units
//In this module 5 time units = 500 ns = .5 µs
always #5
begin
toggle = ~toggle;
$display("%d , In %m toggle = %b ", $time, toggle);
end
endmodule
```

Useful System Tasks

- A file can be opened with the system task **$fopen**

Usage: <file_handle> = $fopen("<name_of_file>");

```
//Multichannel descriptor
integer handle1, handle2, handle3; //integers are 32-bit values
//standard output is open; descriptor = 32'h0000_0001 (bit 0 set)
initial
begin
handle1 = $fopen("file1.out"); //handle1 = 32'h0000_0002 (bit 1 set)
handle2 = $fopen("file2.out"); //handle2 = 32'h0000_0004 (bit 2 set)
handle3 = $fopen("file3.out"); //handle3 = 32'h0000_0008 (bit 3 set)
end
```

Useful System Tasks Cont'd

- Writing to files
  - The system tasks $fdisplay, $fmonitor, $fwrite, and $fstrobe are used to write to files.

```
//All handles defined in Example 9-9
//Writing to files
integer desc1, desc2, desc3; //three file descriptors
initial
begin
desc1 = handle1 | 1; //bitwise or; desc1 = 32'h0000_0003
$fdisplay(desc1, "Display 1");//write to files file1.out & stdout
desc2 = handle2 | handle1; //desc2 = 32'h0000_0006
$fdisplay(desc2, "Display 2");//write to files file1.out &
file2.out
desc3 = handle3 ; //desc3 = 32'h0000_0008
$fdisplay(desc3, "Display 3");//write to file file3.out only
end
```

Useful System Tasks Cont'd

- Closing files

  Files can be closed with the system task $fclose.

Usage: $fclose(<file_handle>);

```
//Closing Files
$fclose(handle1);
```

Useful System Tasks Cont'd

- Displaying Hierarchy
  - Hierarchy at any level can be displayed by means of the %m option in any of the display tasks, $display, $write task, $monitor, or $strobe task.

```
//Displaying hierarchy information
module M;
...
initial
$display("Displaying in %m");
endmodule
//instantiate module M
module top;
...
M m1();
M m2();
//Displaying hierarchy information
M m3();
endmodule
```

Useful System Tasks Cont'd

- Strobing
  - Strobing is done with the system task keyword $strobe.
  - If $strobe is used, it is always executed after all other assignment statements in the same time unit have executed.

```
//Strobing
always @(posedge clock)
begin
a = b;
c = d;
end
always @(posedge clock)
$strobe("Displaying a = %b, c = %b", a, c);
// display values at posedge
```

Useful System Tasks Cont'd

- Random Number Generation
  - The system task $random is used for generating a random number.

Usage: $random;

$random(<seed>);

- The value of <seed> is optional and is used to ensure the same random number sequence each time the test is run.

Random Number Generation

```
//Generate random numbers and apply them to a simple ROM
module test;
integer r_seed;
reg [31:0] addr;//input to ROM
wire [31:0] data;//output from ROM
...
...
ROM roml(data, addr);
initial
r_seed = 2; //arbitrarily define the seed as 2.
always @(posedge clock)
addr = $random(r_seed); //generates random numbers
...
<check output of ROM against expected results>
...
...
endmodule
```

Useful System Tasks Cont'd

- Initializing Memory from File

```
module test;
reg [7:0] memory[0:7]; //declare an 8-byte memory
integer i;
initial
begin
//read memory file init.dat. address locations given in memory
$readmemb("init.dat", memory);
module test;
//display contents of initialized memory
for(i=0; i < 8; i = i + 1)
$display("Memory [%0d] = %b", i, memory[i]);
end
endmodule
```

Initializing memory from File

Init.dat

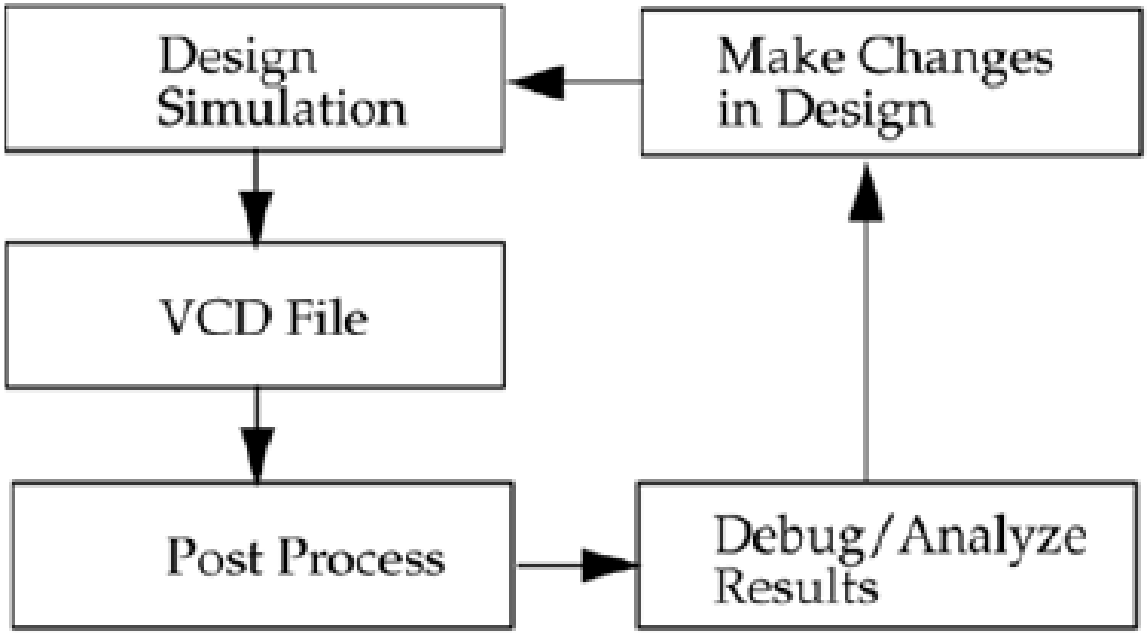
```
@002
11111111 01010101
00000000 10101010
@006
1111zzzz 00001111
```

Output

```
Memory [0] = xxxxxxxx
Memory [1] = xxxxxxxx
Memory [2] = 11111111
Memory [3] = 01010101
Memory [4] = 00000000
Memory [5] = 10101010
Memory [6] = 1111zzzz
Memory [7] = 00001111
```

Value Change Dump File

- A value change dump (VCD) is an ASCII file that contains information about simulation time, scope and signal definitions, and signal value changes in the simulation run.
- All signals or a selected set of signals in a design can be written to a VCD file during simulation.



Value Change Dump File

```
//specify name of VCD file. Otherwise,default name is
//assigned by the simulator.
initial
$dumpfile("myfile.dmp"); //Simulation info dumped to myfile.dmp
//Dump signals in a module
initial
$dumpvars; //no arguments, dump all signals in the design
initial
$dumpvars(1, top); //dump variables in module instance top.
//Number 1 indicates levels of hierarchy. Dump one
//hierarchy level below top, i.e., dump variables in
top,
//but not signals in modules instantiated by top.
initial
$dumpvars(2, top.m1);//dump up to 2 levels of hierarchy below
top.m1
initial
$dumpvars(0, top.m1);//Number 0 means dump the entire hierarchy
// below top.m1
//Start and stop dump process
initial
begin
$dumpon; //start the dump process.
#100000 $dumpoff; //stop the dump process after 100,000 time units
end
//Create a checkpoint. Dump current value of all VCD variables
initial
$dumpall;
```

Thank you