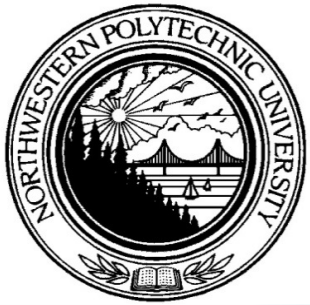



EE461 Verilog-HDL

Week 8 System Task & Function



Alex Yang, Engineering School, NPU

Week 8 Outlines

- System Task & Function
  - Display Task: $strobe
  - Simulation Control Task:
    - $reset, $stop, $finish
  - Stochastic Probabilistic Tasks:
    - $random
  - Waveform Dumping:
    - VCD file: $dumpfile, $dumpvar, $dumpon, $dumpoff, $dumpall
    - VCD+ file: $vcdplusfile (“file_name”); $vcdpluson;
  - File I/O Task:
    - $fopen, $fclose, $fdisplay, $fstrobe $fmonitor & $fwrite

Display Task

- $display & $strobe
  - $display is to print current variable's value and executes before the nonblocking statements update LHS. Therefore if $display contains LHS variable of nonblocking assignment, the results are not proper.
  - $strobe is like $monitor and displays the parameters at the very end of the current simulation time unit rather than exactly when it is executed. It shows updated values at the end of the time step after all other commands, including nonblocking assignments, have completed.

Display Task

- $display & $strobe

```
module case1;
  reg [31:0] data;
  initial begin
    #20 data=50;
    $strobe("Strobe",$time,data);
    $display("display",$time,data);
    data=30;
  end
endmodule
```

Note: Time=20, Display:50
Time=20, Strobe:30

```
module case2;
  reg [31:0] data;
  initial begin
    #20 data<=50;
    $strobe("Strobe",$time,data);
    $display("display",$time,data);
    data<=30;
end
endmodule
```

Note: Time:20,Display=x
Time:20,Strobe=30

```
module case3;
  reg [31:0] data;
  initial begin
    #20 data=50;
    $strobe("Strobe",$time,data);
    $display("display",$time,data);
    data<=30;
  end
endmodule
```

Note: Time=20, Display:50
Time=20, Strobe:30

```
module case4;
  reg [31:0] data;
  initial begin
    #20 data<=50;
    $strobe("Strobe",$time,data);
    $display("display",$time,data);
    data=30;
  end
endmodule
```

Note: Time=20, Display:x
Time=20, Strobe:50

Simulation Control Task

- $finish, $stop, $reset
  - $finish: ends the simulation, exits the simulator and passes control back to the operating system.

    $finish(n) => 1. n=0, No message 2. n=1 (default) simulation time and location, CPU time used in and data structure size

```
module testFinish;
        integer a;
        initial begin
                a = 100;
                $display("a is %0d",a);
                #10 $finish(0);
        end
endmodule
a is 100
        V C S   S i m u l a t i o n   R e p o r t
Time: 10
```

```
module testFinish;
        integer a;
        initial begin
                a = 100;
                $display("a is %0d",a);
                #10 $finish;  //Equivalent to $finish(1);
        end
endmodule
a is 100
$finish called from file "testFinish.v", line 6.
$finish at simulation time                   10
        V C S   S i m u l a t i o n   R e p o r t
Time: 10
CPU Time:      0.460 seconds;       Data structure size:   0.0Mb
Sat Feb 22 17:41:09 2014
```

Simulation Control Task

- $reset, $stop, $finish
  - $stop: suspends the simulation and display an interactive command line.

```
module testStop;
      integer a;
      initial begin
            a = 100;
            $display("a is %0d",a);
            #10 $stop; // Stop running here
            #10 a=200;
            $display("a is %0d",a);
            #10 $finish;
      end
endmodule

>> vcs   -debug_all   testStop.v
>> ./simv
```

ucli Commands	Description
restart	Restarts simulations.
next	Execute next one line.
run	Advances simulation to a point specified by time or edge of signal
finish	Terminates a simulation
stop	Sets and displays breakpoints
other	Check vcs manual

```
npu29.npu.edu - PuTTY
[yangjh@npu29 Test_stop]$ ./simv
Chronologic VCS simulator copyright 1991-2011
Contains Synopsys proprietary information.
Compiler version E-2011.03; Runtime version E-2011.03;  Feb 23 17:39 2014
a is 100
$stop at time 10 Scope: testStop File: testStop.v Line: 6
ucli%
```

Simulation Control Task

- $reset, $stop, $finish
  - $reset: resets the simulation back to time 0

```
module testReset;
      integer a;
      initial begin
            a = 100;
            $display("a is %0d",a);
            #10 a=200;
            #10 $reset;
            $display("a is %0d",a);
            #10 $finish;
      end
endmodule
>> vcs   -debug_all   testStop.v
>> ./simv  -ucli
```

```
npu29.npu.edu - PuTTY
[yangjh@npu29 Test_stop]$ ./simv -ucli
Chronologic VCS simulator copyright 1991-2011
Contains Synopsys proprietary information.
Compiler version E-2011.03; Runtime version E-2011.03;  Feb 23 18:06 2014
ucli% next
testStop.v, 4 :                 a = 100;
ucli% next
testStop.v, 5 :                 $display("a is %0d",a);
ucli% next
a is 100
testStop.v, 7 :                 #10 a=200;
ucli% next
testStop.v, 8 :                 #10 $reset;
ucli% next
Chronologic VCS simulator copyright 1991-2011
Contains Synopsys proprietary information.
Compiler version E-2011.03; Runtime version E-2011.03;  Feb 23 18:06 2014
ucli%
```

Stochastic Task

- Stochastic Task: $random

  random # generation is very helpful in functional verification.
  - return value: 32-bits signed #.
  - random # generation within ranges
    - 0 ~ #; min ~ max; min1~max1 or min2~max2;
  - any numbers of bits random # generation
  - random # generation of multiple some numbers
  - floating point random number
  - random # within a range w/o repetition
  - random # generation by seed
  - the same random # could be created by the same seed.
  - seed value will be changed after calling $random(seed).

Stochastic Task

Stochastic Task: $random

```
module memTB;
  reg readWrite, clock;
  reg [31:0] data, address;
 initial begin
   clock = 0;
   forever
      #10 clock = ~clock;
 end

 initial begin
   repeat(5)@(negedge clock) begin
      readWrite = $random ;
      data = $random;
      address = $random;
   end
   $finish;
 end
initial
   $monitor($time,"readWrite = %d ; data =
%d ; address = %d;",readWrite,data,address);
endmodule
```

Note: random # in decimal radix 10

```
module TB;
integer address;

initial begin
  repeat(5)
    #1 address = $random;
end

initial
 $monitor("address = %d;",address);

endmodule
```

Note: assigning random # to integer variable, signed # may be like follows

```
address =          x;
address =  303379748;
address = -1064739199;
address = -2071669239;
address = -1309649309;
address =  112818957;
```

```
module TB;
reg [31:0] add1;
integer add2;
integer add3;
initial begin
 repeat(5) begin
   #1;
   add1 = $random % 10;
   add2 = {$random} %10 ; // range 0-10
   add3 = $unsigned($random) %10 ;
  end
end
initial
  $monitor("add3 = %d;add2 = %d;
  add1 =%d",add3,add2,add1);
endmodule
```

Note: if $random is negative, negative # will be out of range. {$random} = $unsigned($random)

```
add3 = 7;add2 = 7;add1 = 8
add3 = 7;add2 = 7;add1 = 4294967287
add3 = 1;add2 = 2;add1 = 4294967295
add3 = 7;add2 = 8;add1 = 9
add3 = 9;add2 = 2;add1 = 9
```

Stochastic Task

- Stochastic Task: $random

```
module TB;
  integer add;
 initial begin
  repeat(5) begin
    #1;
   if($random % 2)
       add = 40 + {$random} % (50 - 40) ;
   else
       add = 90 + {$random} % (100 - 90) ;
   $display("add = %d",add);
  end
 end
endmodule
```

Note: random # = (40~50) or (90~100)

```
module TB;
  reg [4:0] add1;
  reg [44:0] add2;

  initial begin
  repeat(5) begin
     add1 = $random ;
    add2 = {$random,$random};
$display("add1 = %b,add2 = %b", add1,add2);
  end
 end
endmodule
```

Note: any number of bits random #

```
module TB;
  integer num1,num2,tmp;

 initial begin
  repeat(5) begin
    #1;
    tmp = {$random} / 3;
    num1 = (tmp) * 3;
    tmp = {$random} / 3;
    num2 = (tmp) * 5;
  $display("num1 = %d,num2 = %d",num1,num2);
  end
 end
endmodule
```

Note: random # is mutiple of 3 & 5

Stochastic Task

- Stochastic Task: $random

```
module TB;

  integer num1,num2,num3;
  real rNum;

  initial begin
    repeat(5) begin
      #1;
      num1 = $random;
      num2 = $random;
      num3 = $random;
      rNum = num1 + ((10)**(-(num2)))*(num3);
     $display("rNum = %e",rNum);
    end
  end
endmodule
```

Note: real # = rand1+$10^{-\text{rand2}}$* rand3

```
module TB;

   real rNum;

  initial begin
    repeat(5) begin
      #1;
      rNum=$bitstoreal({$random,$random});
      $display("rNum = %e",rNum);
    end
  end
endmodule
```

Note: real # -> 64 bits

```
module TB;
    reg sgn;
    reg [10:0] exp;
    reg [51:0] man;
    real rNum;

    initial begin
      repeat(5) begin
        sgn = $random;
        exp = $random;
        man = $random;
    rNum = $bitstoreal({sgn,exp,man});
        $display("rNum = %e",rNum);
      end
    end
endmodule
```

Note: scientific notation random #

Stochastic Task

Stochastic Task: $random

```
module TB;
    integer num,i,j,index;
    integer arr[9:0];
    reg ind[9:0];
    reg got;
    initial begin
      index=0;
      for(i=0;i<10;i=i+1) begin
                    arr[i] = i;
                    ind[i] = 1;
      end
      for(j = 0;j<10 ;j=j+1) begin
         got = 0;
         while(got == 0) begin
            index = { $random() } % 10;
            if(ind[index] == 1) begin
                    ind[index] = 0;
                    got = 1;
                    num = arr[index];
           end
          end
        $write("| num=%2d |",num);
      end
    end
endmodule
```

Note: random # =(0~9) without repetiation

```
module TB;
    integer num,seed,i,j;

    initial begin
       for(j = 0;j<4 ;j=j+1) begin
          seed = j;
          $display(" seed is %d",seed);
          for(i = 0;i < 10; i=i+1) begin
              num = { $random(seed) } % 10;
             $write("| num=%2d |",num);
          end
         $display(" ");
      end
   end
endmodule
```

Note: variable as seed must be reg, real, integer or time.

```
module TB;
    integer num,seed,i,j;

   initial begin
      seed = 0;
      for(j = 0;j<2 ;j=j+1) begin
        if(j ==0) $display(" seed is %d",seed);
       else $display(" No seed is given ");
       for(i = 0;i < 10; i=i+1) begin
         if( j == 0) num = { $random(seed) } % 10;
         else num = { $random() } % 10;
         $write("| num=%2d |",num);
       end

       $display(" ");
      end
   end
endmodule
```

Note: by default, value of random task seed is 0. And return value is the same when seed = 0

Stochastic Task

Stochastic Task: $random

```
module TB;
   integer num,seed,i,j;

  initial begin
    for(j = 0;j<4 ;j=j+1) begin
      seed = 2;
      $display(" seed is %d",seed);
      for(i = 0;i < 10; i=i+1) begin
        num = { $random(seed) } % 10;
        $write("| num=%2d |",num);
      end
    $display(" ");
  end
 end
endmodule
```

Note: the same starting seed will generate the same group of random #

```
module TB;
  integer num,seed,i,j;

  initial begin
    seed = 0;
    for(j = 0;j<10 ;j=j+1) begin
      num = { $random(seed) } % 10;
      $write("| num=%2d |",num);
      $display(" seed is %d ",seed);
    end
  end
endmodule
```

Note: after calling $random w/ seed, seed value will be changed. If calling $random(), the seed could be fixed as 0. But the value of $random() is different.

```
module TB;
  integer num,seed,i,j;

  initial begin
    seed = 837833973;
    for(j = 0;j<10 ;j=j+1) begin
      num = { $random(seed) } % 10;
      $write("| num=%2d |",num);
      $display(" seed is %d ",seed);
    end
  end
endmodule
```

Note: in each iteration of for loop, seed value in random task is different.

Stochastic Task

- Stochastic Task: $random

```
module TB;
    integer num,seed,j;
   reg [0:31] i;

  initial begin
    i = 0;
   seed = 1;
   while (seed != 0) begin
                  if(i == 0)
                  seed = 0;
                  i = i + 1;
                  num = $random(seed);
                  if(seed < 20 && seed > 0)
                  $display(" seed is %d after values %d ",seed,i);
   end
   $display(" seed is one after this number of random numbers
%0d total numbers available are %d",i,{32'hffff_ffff});
 end
endmodule
```

Note: check seed changes after calling $random(seed)

```
module TB;
   integer num,seed,j;
   reg [0:31] i;

   initial begin
     i = 0;
    seed = 0;
    while (seed != 1) begin
       if(i == 0)  seed = 1;
       i = i + 1;
      num = $random(seed);
      if(seed < 20 && seed > 0)
         $display(" seed is %d after values %d ",seed,i);
     end
     $display(" seed is one after this number of  random
numbers %0d total numbers available are
%d",i,{32'hffff_ffff});
   end
endmodule
```

Note: Create random #

Waveform Dumping

- VCD(Value Change Dump) File Generation:
  - Waveform viewer (Synopsys dve) needs VCD file to display signal waveforms for debugging.
  - VCD file could be used for power analysis
  - A value change dump (VCD) file contains information about value changes on selected variables in the design
  - A VCD file is an ASCII file which contains header information, variable definitions, and the value changes for all variables

The $dumpfile task shall be used to specify the name of the VCD file.

Waveform Dumping

- VCD(Value Change Dump) File Generation:

```
module cnt ( out, clk, reset );
    input clk, reset;
   output [3:0] out;
   reg [3:0] out;
   wire [3:0] next;

// Implements reset and increment

assign next = reset ? 4'b0 :
                    (out + 4'b1);

// Implements the flip-flops

always@(posedge clk)begin
   out <= #1 next;
end
endmodule
```

```
module cntTB;
    wire [3:0] out;
    reg clk, reset;
    cnt uCnt ( .out (out[3:0]),
               .reset (reset),
               .clk (clk)
     );
    always #10 clk = ~clk;
    initial begin
      $monitor ("time=%5,
       clk=%b,reset=%b,out=%b",
       $time,clk, reset, out[3:0]);
       clk = 1'b0;
       reset = 1'b0;
      @(posedge clk);
       #1; reset = 1'b1;
      @(posedge clk);
       #1; reset = 1'b0;
      @(posedge clk);#1;
      @(posedge clk);#1;
```

```
    @(posedge clk);#1;
    @(posedge clk);#1;
    @(posedge clk);#1;
     if (out != 4'b0110) begin
         $display("ERROR 1: Out
          is not equal to 4'b0110");
          $finish;
    end
    $display("All tests completed
                sucessfully\n\n");
    $finish;
  end
  initial begin
    $dumpfile ("cnt.vcd");
    $dumpvars (0, cntTB);
  end
endmodule
```

Note:

a. 0 - dump all variables in top module and below submodules

b. 1 - dump all variables in top module only

c. 2- dump all variables in top module and one level below submodule

Waveform Dumping

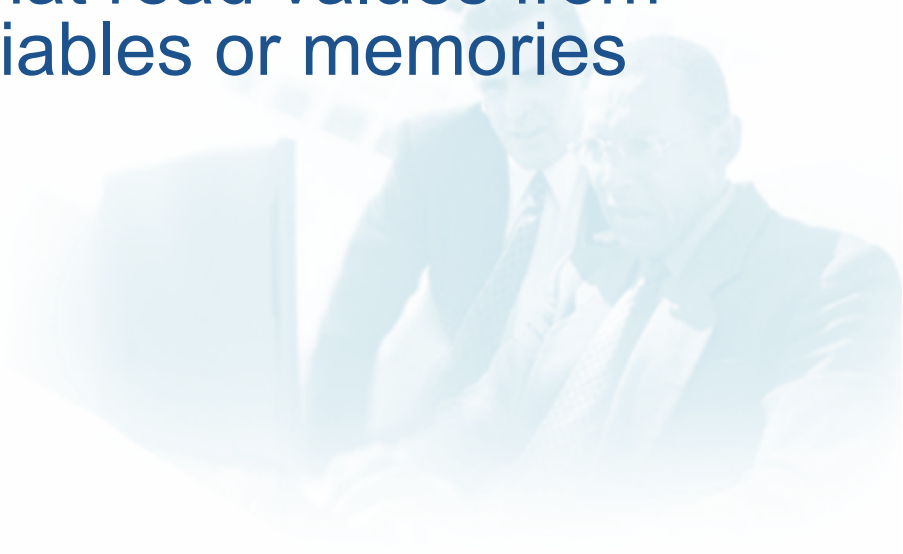
- VPD(VDC Plus Dumping-VCD+) File Generation:
  - VCD+ provides a compressed binary format that dramatically reduces file size as compared to VCD and other proprietary file formats.
  - The VCD+ compressed binary format dramatically reduces signal load time.
  - VPD file generation:

    $vcdplusfile (“file_name”);

    $vcdpluson; //turn on simulation

File I/O Task

- File-based operations:
  - Functions and tasks that open and close files
    - $fopen, $fclose
  - Tasks that output values into variables & files
    - $fdisplay, $fmonitor, $fwrite
  - Tasks and functions that read values from files and load into variables or memories
    - $readmemb, $readmemh



File I/O Task

- Open and Close Files: $fopen, $fclose
  - e.g. handleA = $fopen("fileName"); $fclose(handleA);

```
module testFopen;
//an integer has 32 bits so is perfect for this usage
    integer handleA, handleB;
initial begin
      handleA = $fopen("myfile.out");
 // handleA = 0000_0000_0000_0000_0000_0000_0000_0010
   handleA = handleA | 1; // Print on monitor too
   handleB = $fopen("anotherfile.out");
 // handleB = 0000_0000_0000_0000_0000_0000_0000_0100
     $fdisplay(handleA,"Hello!!!");
     $fclose(handleA);
   end
endmodule
```

$fopen:

1. Data type of return value: integer
2. 1st handle value = 32'h0002;
   2nd handle value = 32'h0004; so on
3. print on monitor:
   monitor file handle value=32'h0001
4. print on both monitor & file
   handleA = handleA | 1;
5. "fileName":
   a. non existing file (new file)
   b. existing file
   c. the old contents in the file will be overridden every time after running

File I/O Task

- Output Values into Variables & Files
  - $fdisplay, $fmonitor, $fwrite

  e.g. $fdisplay(handleA, "value a: %0b", a);

  $fmonitor/$fwrite is the same as $monitor/$write except file printing
  - $fdisplay - > $fdisplayb, $fdisplayo, $fdisplayh, no need to specify %b, %o, %h.

```
module testFdisplay;
        integer mcd,number;

        initial begin
          mcd = $fopen("temp.txt"); // mcd = multi_channel_descriptor
          repeat(7) begin
                  number = $random;
                  $fdisplayb(mcd, "Number is ", number);
          end
          $fclose(mcd);
        end
endmodule
```

File I/O Task

- Load file into memories or variables
  - $readmemb, $readmemh

```
module  memory;
  integer i;
  reg [7:0] myMemory [0:255];

  initial begin
 //$readmemb("memory.list", myMemory);
 $readmemh("memory1.list", myMemory);
  end
  initial begin
      #10 $display("Data in the memory");
      for(i=0;i<10; i=i+1)begin
$display("myMemory[%0d]:%b",i,myMemory[i]);
      end
  end
endmodule
```

```
//Comments are allowed in memory.list
//1100_1100   // This is first address i.e 8'h00
//1010_1010   // This is second address i.e 8'h01
//@05       // Jump to new address 8'h05
//0101_1010   // This is address 8'h55
//0110_1001   // This is address 8'h56
ab
cd
@ 06
ef
12
```

```
Data in the memory          // Display on monitor
myMemory[0]:10101011
myMemory[1]:11001101
myMemory[2]:xxxxxxxx
myMemory[3]:xxxxxxxx
myMemory[4]:xxxxxxxx
myMemory[5]:xxxxxxxx
myMemory[6]:11101111
myMemory[7]:00010010
myMemory[8]:xxxxxxxx
myMemory[9]:xxxxxxxx
```

File I/O Task

- Example:

```
module fileDemo;
        integer   handle, channels, index, rand;
        reg [7:0] memory [15:0];
        initial begin
          handle = $fopen("mem.dat");
          channels = handle | 1;
          $display("Generating contents of file mem.dat");
          $fdisplay(channels, "@2");              // Jump to 8'h02 address
          for(index = 0; index < 14; index = index + 1) begin
              rand = $random;
              $fdisplay(channels, "%b", rand[12:5]);
          end
          $fclose(handle);
          $readmemb("mem.dat", memory);
          $display("\nContents of memory array");
          for(index = 0; index < 16; index = index + 1)
              $displayb(memory[index]);
        end
      endmodule // fileDemo
```