EECS 31L: Introduction to Digital Design Lab Lecture 8

Pooria M.Yaghini

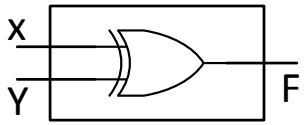
The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 4: Overview

- Testbench description
- Randomization
- Task, function,
- Module, initial
- Forever, repeat, fork/join
- Wait, #n
- File
- System Calls
- Readmemb, readmemh
- Force/release
- DPI

Lecture 4: Testbench

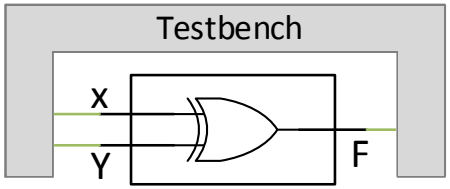
- How to verify the correctness of your design?




```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY xor2 IS
   PORT (x: IN std_logic;
         y: IN std_logic;
         F: OUT std_logic);
END xor2;

ARCHITECTURE xor2_beh OF xor2 IS
BEGIN
   F <= x XOR y;
END xor2_beh;
```

Lecture 4: Testbench

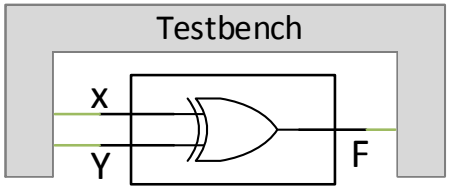
- How to verify the correctness of your design?
  - Simulation



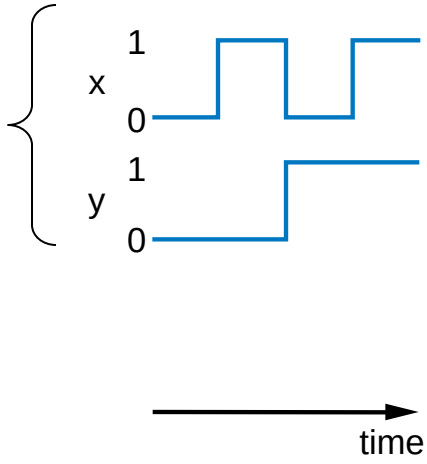

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY xor2 IS
   PORT (x: IN std_logic;
         y: IN std_logic;
         F: OUT std_logic);
END xor2;

ARCHITECTURE xor2_beh OF xor2 IS
BEGIN
   F <= x XOR y;
END xor2_beh;
```

Lecture 4: Testbench

- How to verify the correctness of your design?
  - Simulation
  - User provides input values,
    - Test vectors – sequence of input values



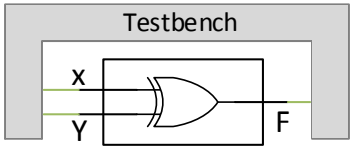
User provides test vectors




```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY xor2 IS
   PORT (x: IN std_logic;
         y: IN std_logic;
         F: OUT std_logic);
END xor2;

ARCHITECTURE xor2_beh OF xor2 IS
BEGIN
   F <= x XOR y;
END xor2_beh;
```

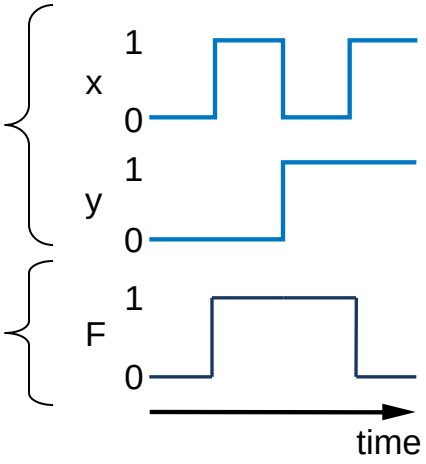
Lecture 4: Testbench

- How to verify the correctness of your design?
  - Simulation
  - User provides input values, simulator generates output values
    - Test vectors – sequence of input values
    - Waveform – graphical depiction of sequence



User provides test vectors

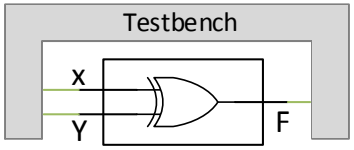
Simulator generates output values based on HDL description




```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY xor2 IS
   PORT (x: IN std_logic;
         y: IN std_logic;
         F: OUT std_logic);
END xor2;

ARCHITECTURE xor2_beh OF xor2 IS
BEGIN
   F <= x XOR y;
END xor2_beh;
```

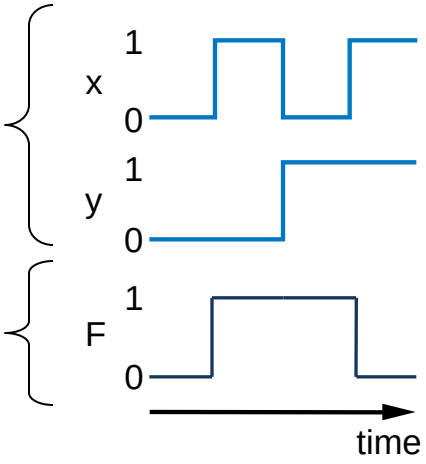
Lecture 4: Testbench

- How to verify the correctness of your design?
  - Simulation
  - User provides input values, simulator generates output values
    - Test vectors – sequence of input values
    - Waveform – graphical depiction of sequence
    - Testbench does not have any input/output port



User provides test vectors

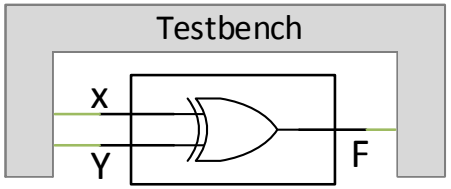
Simulator generates output values based on HDL description

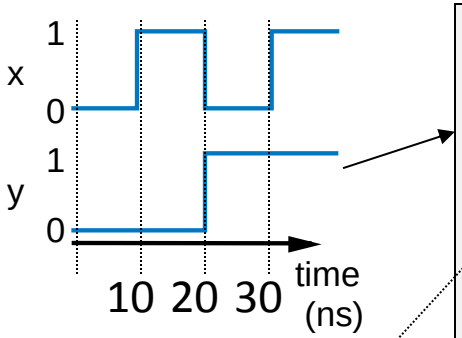



```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY xor2 IS
   PORT (x: IN std_logic;
         y: IN std_logic;
         F: OUT std_logic);
END xor2;

ARCHITECTURE xor2_beh OF xor2 IS
BEGIN
   F <= x XOR y;
END xor2_beh;
```

Lecture 4: Testbench

- Instead of drawing test vectors, user can describe them with HDL



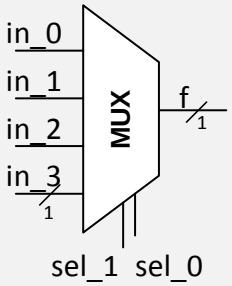



```
...
y = '0'; x = '0';
#10ns;
y = '0'; x = '1';
#10ns;
y = '1'; x = '0';
#10ns;
y = '1'; x = '1';
...
```

"#10ns;" –
Tells simulator to keep present values for 10 ns, before executing the next statement

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY xor2 IS
   PORT (x: IN std_logic;
         y: IN std_logic;
         F: OUT std_logic);
END xor2;

ARCHITECTURE xor2_beh OF xor2 IS
BEGIN
   F <= x XOR y;
END xor2_beh;
```

Lecture 4: Testbench for 4x1 Multiplexer

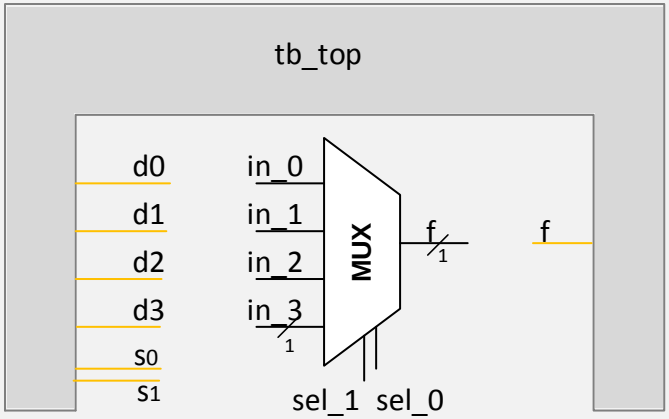


Lecture 4: Testbench for 4x1 Multiplexer

```
module tb_top;




endmodule;
```

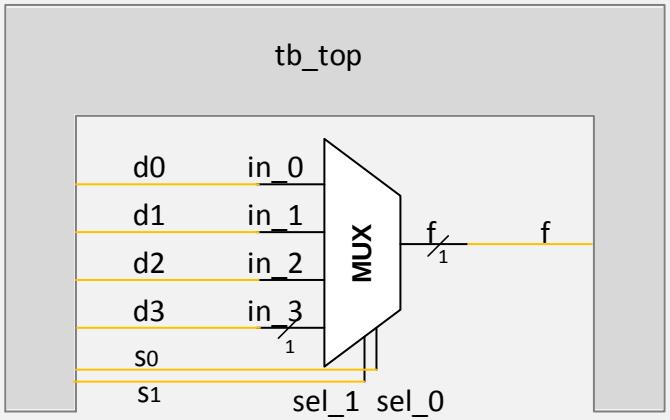
Lecture 4: Testbench for 4x1 Multiplexer

```
module tb_top;

logic d0,d1,d2,d3 ,s0, s1;
wire  f;









endmodule;
```



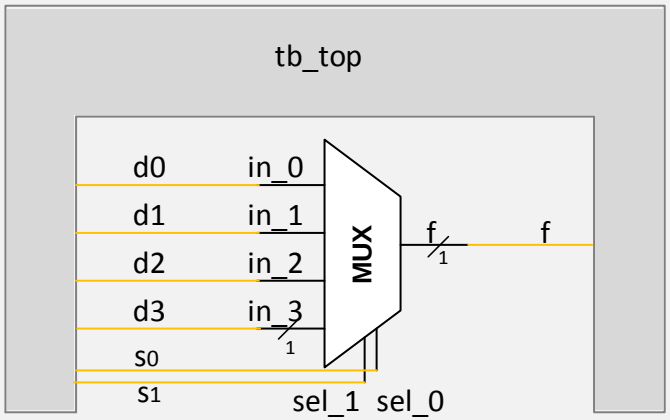
Lecture 4: Testbench for 4x1 Multiplexer Cont.

```
module tb_top;

logic d0,d1,d2,d3 ,s0, s1;
wire  f;

mux4to1 mux4to1_inst(
                        .in_0(d0),
                        .in_1(d1),
                        .in_2(d2),
                        .in_3(d3),
                        .sel_0(s0),
                        .sel_1(s1),
                        .f(f)
                    );



endmodule;
```



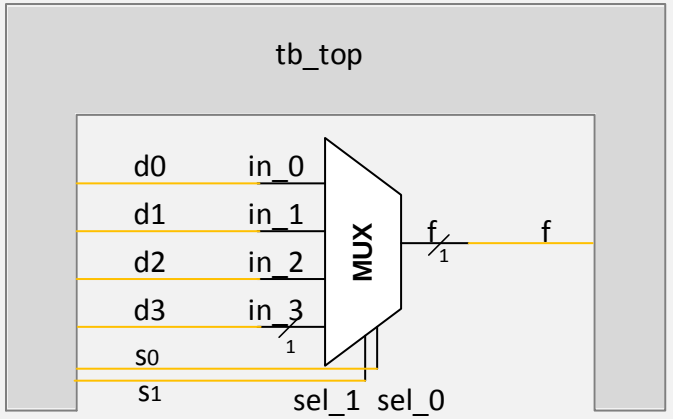
Lecture 4: Testbench for 4x1 Multiplexer Cont.

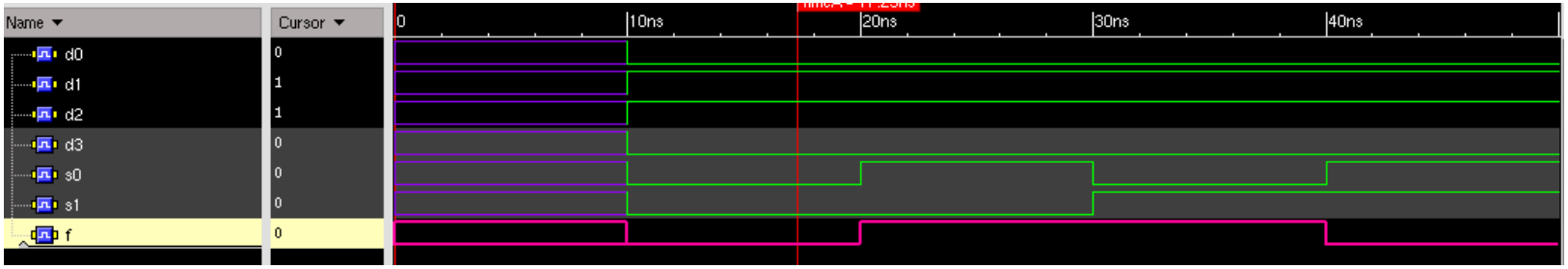
```
module tb_top;

logic d0,d1,d2,d3 ,s0, s1;
wire  f;

mux4to1 mux4to1_inst(.in_0(d0),  .in_1(d1),  .in_2(d2), .in_3(d3),
                     .sel_0(s0), .sel_1(s1), .f(f)
                     );

   initial begin
      #10ns;
      d0 = 0; d1 = 1; d2 = 1; d3 = 0;
      s0 = 0; s1 = 0;

      #10ns;
      d0 = 0; d1 = 1; d2 = 1; d3 = 0;
      s0 = 1'b1; s1 = 1'b0;

      #10ns;

   end //initial
endmodule;
```



Lecture 4: Testbench for 4x1 Multiplexer Cont.

```
module tb_top;

logic d0,d1,d2,d3 ,s0, s1;
wire  f;

mux4to1 mux4to1_inst(.in_0(d0),  .in_1(d1),  .in_2(d2), .in_3(d3),
                          .sel_0(s0), .sel_1(s1), .f(f)
                          );

   initial begin
      #10ns;
      d0 = 0; d1 = 1; d2 = 1; d3 = 0;
      s0 = 0; s1 = 0;

      #10ns;
      d0 = 0; d1 = 1; d2 = 1; d3 = 0;
      s0 = 1'b1; s1 = 1'b0;

      #10ns;

   end //initial
endmodule;
```





Lecture 4: Testbench

- Is the fed data bit patterns enough !!!?
- Debugging using only waveform !!!?

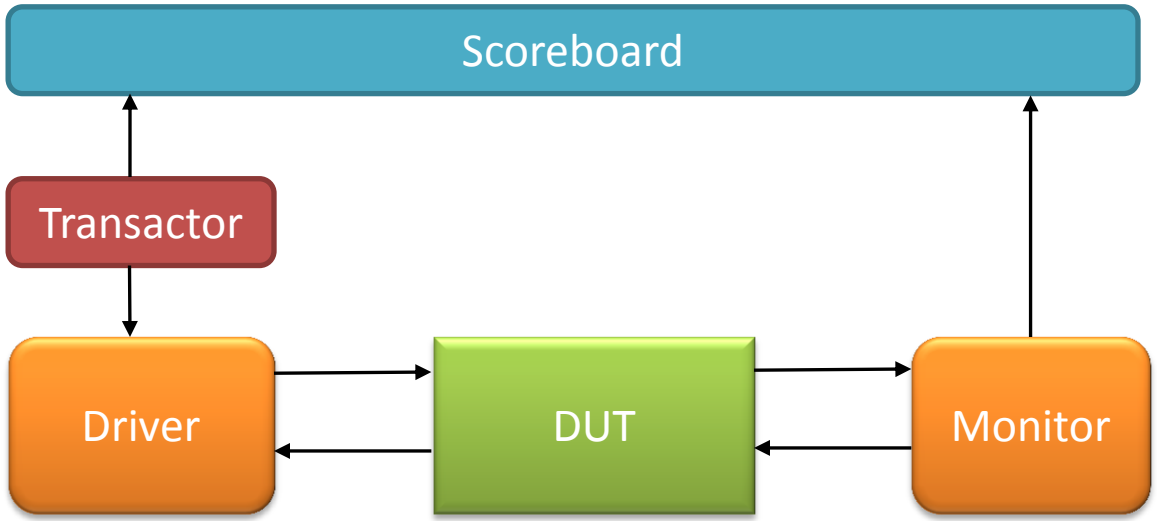
Lecture 4: Testbench

- Is the fed data bit patterns enough !!!?
- Debugging using only waveform !!!?

Absolutely NO

Lecture 4: Testbench

- **A sample testbench architecture**



Lecture 4: SystemVerilog for Verification

- Data types

The SystemVerilog value set consists of the following four basic values:

0—represents a logic zero or a false condition
1—represents a logic one or a true condition
x—represents an unknown logic value
z—represents a high-impedance state

✓ Types that can have unknown and high-impedance values are called *4-state types*. These are **logic**, **reg**, **integer**, and **time**. The other types do not have unknown values and are called *2-state types*, for example, **bit** and **int**.

Lecture 4: SystemVerilog for Verification

- Data types
  - 2 state value integer data types are
    - shortint : 16-bit signed integer.
    - int : 32-bit signed integer.
    - longint : 64-bit signed integer.
    - byte : 8-bit signed integer.
    - bit : user-defined vector size, unsigned.
  - 4-state value integers data types are
    - logic : User defined vector types, unsigned.
    - reg : User defined vector types, unsigned.
    - wire : User defined vector types, unsigned.
    - integer : 32-bit signed integer.
    - time : 64-bit unsigned integer.
  - string
  - enum
  - event

Lecture 4: SystemVerilog for Verification

- Aggregate data types
  - Structures
  - Packed/Unpacked arrays
  - Dynamic arrays
  - Queues
  - Classes

Lecture 4: SystemVerilog for Verification

- struct
  - Structure and union declarations types are same as that found in C language.

```
/* Example 8.0.0 */

typedef struct {
  byte               a;
  logic              b;
  shortint unsigned  c;
} myStruct;


myStruct inst;

inst.a = 10;
inst.b = 1'b1;
inst.c = 1000;
```

Lecture 4: SystemVerilog for Verification

- Associative arrays
  - Number of elements changes dynamically.
  - Size of the collection is unknown or the data space is sparse
  - Associative arrays do not have any storage allocated until it is used
  - The index expression is not restricted to integral expressions, but can be of any type.
  - An associative array implements a lookup table of the elements of its declared type.

Lecture 4: SystemVerilog for Verification

```
/* Example 8.0.1 */
integer as_mem [integer];
integer i;

as_mem[100] = 101;
as_mem[1]   = 100;
as_mem[50]   = 99;
as_mem[256] = 77;

$display ("size of array is %d", as_mem.num());       // Print the size of array
$display ("index 2 exists   %d", as_mem.exists(2));   // Check if index 2 exists
$display ("index 100 exists %d", as_mem.exists(100)); // Check if index 100 exists
// Value stored in first index
if (as_mem.first(i)) begin
  $display ("value at first index %d value %d", i, as_mem[i]);
end
// Value stored in last index
if (as_mem.last(i)) begin
  $display ("value at last index  %d value %d", i,  as_mem[i]);
End

as_mem.delete(100);       // Delete the first index
$display ("Deleted index 100");
// Value stored in first index
if (as_mem.first(i)) begin
  $display ("value at first index %d value %d", i, as_mem[i]);
end
```

Lecture 4: SystemVerilog for Verification

- Queues
  - A queue is a variable-size, ordered collection of homogeneous elements.
  - Each element in a queue is identified by an ordinal number that represents its position within the queue, with 0 representing the first, and $ representing the last.

Lecture 4: SystemVerilog for Verification

```
/* Example 8.0.2 */

// Queue is declated with $ in array size
integer queue[$] = { 0, 1, 2, 3, 4 };
integer i;

  // Insert new element at begin of queue
  queue = {5, queue};
  queue.push_front(6);
  queue.push_back(7);
  queue.insert(4,8);
  i = queue.pop_front();
  i = queue.pop_back();
  queue.delete(4);



task print_queue;
  integer i;
  $write("Queue contains ");
  for (i = 0; i < queue.size(); i ++) begin
    $write (" %g", queue[i]);
  end
  $write("\n");
endtask
```

Lecture 4: SystemVerilog for Verification

- Class
  - SystemVerilog introduces an object-oriented class data abstraction. Classes allow objects to be dynamically created, deleted, assigned, and accessed via object handles.
  - Features:

    Objects
    Object members
    Constructors
    Static class members
    Assignments
    Inheritance and subclasses
    this and super
    Data hiding and encapsulation
    Constant class properties
    Abstract classes and virtual methods
    Class scope resolution operator ::
    Parameterized classes

Lecture 4: SystemVerilog for Verification

```
/* Example 8.0.3 */

   class packet;

     integer size;
     integer payload [];
     integer i;

     function new (integer size); // Constructor
       begin
         this.size = size;
         payload = new[size];
         for (i=0; i < this.size; i ++) begin
           payload[i] = $random;
         end
       end
     endfunction

     task print ();
       begin
         $write("Payload : ");
         for (i=0; i < size; i ++) begin
           $write("%x ",payload[i]);
         end
         $write("\n");
       end
     endtask

   function integer get_size();
       begin
         get_size = this.size;
       end
     endfunction

   endclass
```

Lecture 4: SystemVerilog for Verification

```
/* Example 8.0.3 */

   packet pkt;

   initial begin
     pkt = new(5);
     pkt.print();
     $display ("Size of packet %0d", pkt.get_size());
   end
```

Lecture 4: SystemVerilog for Verification

Processes

- Structured procedures (initial procedures, always procedures, final procedures)
- Block statements (begin-end sequential blocks, fork-join parallel blocks)
- Timing control (delays, events, waits, intra-assignment)
- Process threads and process control

Lecture 4: SystemVerilog for Verification

Structured procedures

- **initial** procedure
- **always** procedure
  - always
  - always_comb
  - always_latch
  - always_ff
- **final** procedure
- **task**
- **function**

Lecture 4: SystemVerilog for Verification

- **initial**
  - is enabled **only once** at the beginning of a simulation
- **always**
  - is enabled at the beginning of a simulation
  - execute repeatedly until the simulation is terminated
- **final**
  - is enabled at the end of simulation time and execute only once

Lecture 4: SystemVerilog for Verification

```
/* Example 8.1 */

initial begin
   a = 0; // initialize a
   for (int index = 0; index < size; index++)
   memory[index] = 0; // initialize memory word
end
```

```
/* Example 8.2 */

initial begin
   inputs = 'b000000; // initialize at time zero
   #10 inputs = 'b011001; // first pattern
   #10 inputs = 'b011011; // second pattern
   #10 inputs = 'b011000; // third pattern
   #10 inputs = 'b001000; // last pattern
end
```

Lecture 4: SystemVerilog for Verification

```
/* Example 8.3 */

always #half_period areg = ~areg;
```

```
/* Example 8.4 */

always @(*) begin
   a = b & c;
   A1:assert (a != e) else if (!disable_error) $error("failed");
end
```

Lecture 4: SystemVerilog for Verification

- Timing Control
  - #
  - @
  - wait

```
/* Example 8.5 */

initial begin
   #10;
   #10us;
   @(a or b or c);
   @(posedge clk);
   wait (!enable) a = b;
   #10 c = d;
end
```

Lecture 4: SystemVerilog for Verification

Assignment statements

- Continuous assignments
- Procedural blocking and nonblocking assignments
- Procedural continuous assignments (assign, deassign, force, release)
- Net aliasing

```
/* Example 8.6 */

   assign z = y; // non-blocking

   initial begin
      a = c;       // blocking
      c = a;
      d <= e;      // non-blocking
   end
```

Lecture 4: SystemVerilog for Verification

Assignment statements

```
/* Example 8.7 */

module nonblock1;
   logic a, b, c, d, e, f;

   // blocking assignments
   initial begin
      a = #10 1; // a will be assigned 1 at time 10
      b = #2 0;  // b will be assigned 0 at time 12
      c = #4 1;  // c will be assigned 1 at time 16
   end

   // nonblocking assignments
   initial begin
      d <= #10 1; // d will be assigned 1 at time 10
      e <= #2 0;  // e will be assigned 0 at time 2
      f <= #4 1;  // f will be assigned 1 at time 4
   end
endmodule
```

scheduled changes at time 2

e = 0

scheduled changes at time 4

f = 1

scheduled changes at time 10

d = 1

Lecture 4: SystemVerilog for Verification

Input/output system tasks and system functions

Display tasks

- $display
- $write
- $displayb
- $writeb
- $displayh
- $writeh
- $displayo
- $writeo
- $strobe
- $monitor
- $strobeb
- $monitorb
- $strobeh
- $monitorh
- $strobeo
- $monitoro
- $monitoroff
- $monitoron

File I/O tasks and functions

- $fclose $fopen
- $fdisplay $fwrite
- $fdisplayb $fwriteb
- $fdisplayh $fwriteh
- $fdisplayo $fwriteo
- $fstrobe $fmonitor
- $fstrobeb $fmonitorb
- $fstrobeh $fmonitorh
- $swriteh $fgetc
- $swriteo $ungetc
- $fscanf $fgets
- $fread $sscanf
- $fseek $rewind
- $fflush $ftell
- $feof $ferror

Lecture 4: SystemVerilog for Verification

Most useful system tasks

- ✓ $display
- ✓ $write
- ✓ $monitor
- ✓ $readmemh
- ✓ $fopen
- ✓ $fwrite
- ✓ $feof
- ✓ $fscanf
- ✓ $fclose
- ✓ $system
- ✓ $time
- ✓ $finish

Lecture 4: SystemVerilog for Verification

Most useful system tasks

```
/* Example 8.8 */

module tb_top;
   reg [8:0] a ; // a = 492 ;
   reg [7:0] b ; // b = 205 ;

   initial begin
      $display("The decimal value of a is: %d", a) ;
      $display("The octal value of a is: %o", a) ;
      $display("The binary value of a is: %b", a) ;
      $display("The hexadecimal value of a is: %h", a);
      $write("The decimal value of b is: %d\n", b) ;
      $write("The hexadecimal value of b is: %h\n", b);
      $write("The binary value of b is: %b ", b) ;
      $write("The octal value of b is: %o\n", b);
      $display("\t @TIME:%t", $time);
   end
endmodule
```

%d or %D	Decimal format
%b or %B	Binary format
%h or %H	Hexadecimal format
%o or %O	Octal format
%c or %C	ASCII character format
%m or %M	Hierarchical name
%s or %S	As a string
%t or %T	Current time format

Lecture 4: SystemVerilog for Verification

Most useful system tasks

```
/* Example 8.9 */

module tb_top;

   initial begin
      $system("date");
      $system("uname -n");
      $system("whoami");
      $system("pwd");
      $finish;
   end
endmodule
```

Lecture 4: SystemVerilog for Verification

Working with files

```
/* Example 8.10 */

 task automatic read_file(input string inputfile);
        integer fileid ;
        string filename;
        integer status;
        bit [6:0] a, b;

        filename = {inputfile,".txt"};
        fileid = $fopen(filename,"r");
        if (fileid == 0) begin
               $display("ERROR : CAN NOT OPEN THE FILE %s",filename);
        end else begin
               while(!$feof(fileid))begin
                     status = $fscanf(fileid, "%h %d", a, b);
                     $display("%03d %02d", a, b);
               end
               $fclose(fileid);
        end

  endtask
```

Lecture 4: SystemVerilog for Verification

Task

```
/* Example 8.11 */

module task_example ();

   initial begin
      #1  doInit(4,5);
      #1  doInit(9,6);
      #1  $finish;
   end

   task automatic doInit (input bit [3:0] count, delay);
     if (count > 5) begin
       $display ("@%g Returning from task", $time);
       return;
     end
     #(delay) $display ("@%t Value passed is %d", $time, count);
   endtask

endmodule
```

Lecture 4: SystemVerilog for Verification

Direct Programming Interface (DPI)

DPI consists of two separate layers:

- SystemVerilog layer
- Foreign language layer (like C)

Lecture 4: SystemVerilog for Verification

Direct Programming Interface (DPI)

```
/* Example 8.12 – C layer*/

#include <stdio.h>
#include <stdlib.h>
#include "svdpi.h"
#include "vpi_user.h"


int  test(
                    int sel,
                    svOpenArrayHandle data_in,
                    svOpenArrayHandle data_out,
                    int * output
                    ){

        int *data_in_p = (int *)svGetArrayPtr(data_in);
        printf("C:: The value of element %d was %d\n", sel, data_in_p[sel]);
        data_in_p[sel]++;

        *(int *)svGetArrElemPtr1(data_out, sel) = data_in_p[sel];
        *output = data_in_p[sel]*2;  return 0;

}
```

Lecture 4: SystemVerilog for Verification

Direct Programming Interface (DPI)

```
/* Example 8.12 */

 import "DPI-C" context test = function int test(
                               input int sel,
                               input int data[],
                               output int outdata[],
                               output int double
                               );

module dpi_example ();

...
```

Lecture 4: SystemVerilog for Verification

Direct Programming Interface (DPI)

```
/* Example 8.12 */
...
module dpi_example ();
   parameter size = 5;
   int   dpi_in  [size-1:0];
   int   dpi_out [size-1:0];
   int   selector;
   int   doubleVal;

   initial begin

      for(int index=0; index<size; index++) begin
         dpi_in = index;
      end
      selector  = 1;
      doubleVal = 0;

      test(1, dpi_in, dpi_out, doubleVal);

      $display("SV:: OUT[%d] = %d is %d, 2X = %d\n", selector, dpi_out[selector], doubleVal);
      $finish;
end
endmodule
```

Lecture 4: SystemVerilog for Verification

Randomization

- $urandom
- $random
- rand
- randc
- constraint

```
/* Example 8.13 */

module system_random();

   initial begin
    $display ("Value is %0d",$urandom(10));
    $display ("Value is %0d",$random(10));
   end

endmodule
```