Logic Simulation

Prof. Chien-Nan Liu
TEL: 03-4227151 ext:34457
Email: jimmy@ee.ncu.edu.tw

Simulation

- Simulation replace the prototype with a software model
- Simulate the circuit behavior before realization
Applications of Simulation

- Design verification:
  - Function
  - Timing
  - Independent of initial states
  - Insensitive to parameter variation
  - Race and hazard free

- Other purposes:
  - Evaluate alternative designs
  - Evaluate design changes
  - Documentation
  - ........

Types of Simulations

- Primary approaches for simulations:
  - Compiler-Driven
  - Table-Driven
  - Event-Driven
  - Cycle-Based

- Compiler-Driven: *(Compiled Code Simulation)*
  - The circuit behavior is compiled into a executable program
  - The response can be obtained by calling the program with inputs as the argument
  - Faster, but lack of observability for internal nodes
Compiled Code Simulation

<table>
<thead>
<tr>
<th>Load</th>
<th>AND</th>
<th>AND</th>
<th>Store</th>
<th>Load</th>
<th>OR</th>
<th>INV</th>
<th>OR</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>F</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>H</td>
</tr>
</tbody>
</table>

\[ H = \text{Simulation} (1,0,1,0,1) \]

Table-Driven Simulation

- An interpreter is used to translate the circuit into internal data structure (table)
- Simulation is done by manipulating such data structure in a predetermined order
  - Called the "topological order"
- Cannot be applied to sequential circuit simulation
  - Feedback cannot be handled
Table-Driven Simulation

- Gates are indexed by the signal dependency graph
  - Topological sort
- Gates are evaluated in the predetermined order
- Fanout may not be needed

For \( (i=0 \text{ to } n) \)
Evaluate Gate \( i \)

<table>
<thead>
<tr>
<th>Node</th>
<th>Type</th>
<th>Fanin</th>
<th>Fanout</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>PI</td>
<td>-</td>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>PI</td>
<td>-</td>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>PI</td>
<td>-</td>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>PI</td>
<td>-</td>
<td>G</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>PI</td>
<td>-</td>
<td>G</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>AND</td>
<td>A,B,C</td>
<td>H</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>OR</td>
<td>D,E</td>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>H</td>
<td>OR</td>
<td>F,G</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>

Event-Driven Simulation

- Based on table-driven simulation
- Event: a change of value in a signal
  - A value change may affect several processes
  - All processes affected are executed in zero time
  - All resulting signal changes are scheduled in the \textit{time wheel}
  - Advance to the time next event is scheduled
  - Process that event
- Only process the activated part of the circuit
Algorithms for EDS

- Process a series of ordered events
- event-list : pending events
- time-wheel: simulation time

```
While (event-list != 0)
    advance time wheel
    determine current events
    update values
    propagate events
    evaluate active events
    schedule new events
```

<table>
<thead>
<tr>
<th>Time</th>
<th>R</th>
<th>S</th>
<th>Q</th>
<th>-Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2.1</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>2.2</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4.1</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>4.2</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

Delta Delay

- Infinitesimally small delay
- Models hardware where a minimal amount of time is needed for a change to occur
  - Even in zero delay simulation
- Allows for ordering of events that occur at the same simulation time
Simulation Example

reg A, B;
always @(B)
begin
  A = 1;
  A = #5 0;
end
always @(A)
begin
  if (A==1)
    B = #10 ~B;
end

Gate-Level Event Driven Simulation

- **Global Routine**
  
  While (event-list not empty)
  
  - advance time wheel to the next time \( t \)
  - Process event entries for time \( t \)

- **Event Process**
  
  **Activated** = 0
  
  for (every event \( E_i \) pending at \( t \))
  
  - Assign \( Ni \) value \( Vi \)
  
  - Put Fanout \( j \) into **Activated**
  
  for (every \( j \) in **Activated**)
  
  - Evaluate \( V_j \)
  
  - if (\( V'_j \neq V_j \)) Schedule \( E_j \) at time \( t + D_j \)
Gate-Level Event Driven Simulation

Disadvantage
- A node may be scheduled more than once at one time slot if there are more than one input change
- A node may be scheduled more than once within the gate inertial delay

Solution 1
- Sort the gates/nodes according to signal dependency
- Give each gate a index to indicate its logic depth, the number of gates from primary inputs
- For each time stamp, process the event with smaller logic_depth index first

Solution 2
- Remove the previous events if they are within the gate inertial delay

Time Wheel Issue
- The delay can be any real number
- Sorting the events according is time consuming \(\log_2 N\)

Solutions
- Delay is discretized by using a “resolution”
  - e.g. 10ps
- Delay is an integer multiple of the resolution
- Each entry in the timing wheel represents the list of events happening in that time slot
- The insertion of event is trivial
Cycle-Based Simulation

- Event-Driven:
  - Timing accurate
  - Better debug environment
  - Simulation speed is slower

- Cycle-Based:
  - Perform evaluations just before the triggering clock edge
    - Repeatedly triggered events are evaluated only once in a clock cycle
    - Applicable to synchronous designs only
  - Faster simulation time (5x – 100x)
  - Only cycle-accurate
  - Require other tools (ex: STA) to check timing problems

Logic Values

- Two-Valued Logic
  - 0 (Logic Zero) and 1 (Logic One)

- Three-Valued Logic
  - 0 (Logic Zero), 1 (Logic 1), X (Unknown)

- Multi-Valued Logic
  - Multiple values to represent multiple bits
  - Number of bits = \( \log_2 \) (number of values)
  - None binary operations
  - Used in I/O and high density memories
**Logic Values in Verilog**

- Four-Valued Logic is used
  - 0 (Logic Zero), 1 (Logic 1), X (Unknown), Z (High Impedance)
- 3-State Logic (high impedance)
  - Report conflicts or potential conflicts if two or more drivers are activated
  - Require an uncertain for the case when the bus is high impedance

**Wired Logic**

- Wired-AND, Wired-OR
- Use an equivalent AND or OR gate to replace the wired net
- EX:
  - `wand D;`
  - `assign D = A1 | A2;`
  - `assign D = B1 | B2;`
  - `assign D = C1 | C2;`
Other Logic Values

MOS Logic / Switching Level
- Boolean logic has uni-directional signal flow
- MOS Circuit / Switching Level has bi-directional signal flow
- Gates can store charge to make it a dynamic latch
- Charge decay is another issue
- Charge sharing is another issue

Possible Solutions
- Use logic strength to represent the amount of charge
- Use device strength to represent the ability to charge or discharge

Logic Values in VHDL

Standard Logic:
- ‘U’ -- Uninitialized
- ‘X’ -- Forcing unknown
- ‘0’ -- Forcing 0
- ‘1’ -- Forcing 1
- ‘Z’ -- High impedance
- ‘W’ -- Weak unknown
- ‘L’ -- Weak 0
- ‘H’ -- Weak 1
- ‘-’ -- Don’t care
Resolution Function

Table 8-4 Resolution Function Table for IEEE 9-valued Logic

```python
CONSTANT resolution_table : stdlogic_table := (
    --  U  X  0  1  Z  W  L  H  --
    ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
    ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
    ( 'U', 'X', '0', 'X', '0', '0', '0', '0', '0' ), -- | 0 |
    ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | 1 |
    ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
    ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
    ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
    ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
    ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ) -- | - |
);```

Logic Evaluation

- Based on look-up table for each gate
- Truth tables for 2-valued logic:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
### 4-Valued Logic

Truth table for 4-valued logic:

<table>
<thead>
<tr>
<th>AND</th>
<th>X</th>
<th>0</th>
<th>1</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>Z</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OR</th>
<th>X</th>
<th>0</th>
<th>1</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Z</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>X</td>
</tr>
</tbody>
</table>

---

### Control Values

Control value C will dominate the output value

- Use *input scan* first can determine the output values more quickly
- Example

  \[
  \text{NAND}(A_i)
  \]

  *for every input Ai*

  *if Ai = C then return Output=1*

  *return Output = 0*

- Control Values:
  - AND : C=0;  OR : C=1.
Delay Model

- Zero delay: no delay in gate
- Unit delay: all the gates have an delay of 1 unit
- Distributed delay: assign circuit dependent delay
  - Transition-independent delay
  - Rise/Fall delay
  - Ambiguous delay model (minimal, nominal, maximal)
- Module path delay: assign delay values from inputs to outputs

Delay Assignments

- Transition-independent delay
  - assign #5 b = a;
- Rise/Fall delay
  - assign #(5,20) c = d;  // (rise_time, fall_time)
- Ambiguous delay model
  - assign #(95:100:105) clk = 1;  // (min: typ: max)
- Module path delay
  - specify
    (A => Q) = (10,8);
    (B => Q) = (12,16);
  end specify
Hazard Detection

- **Static Hazard** -
  Unnecessary transition when there is no signal change

- **Dynamic Hazard** -
  Unnecessary extra transition when signal changes from 0 to 1 or from 1 to 0

Inertial Delay

- **Minimum pulse width**
  - Input value must be stable for specified delay before value is allowed to propagate to the output

- **Example:**
  
  \[ Z \leq A \text{ after } 10 \text{ ns}; \]
  
  (output) (input value)

  \[ Z \]
  
  \[ A \]
  
  delay = 10 ns
Inertial Delay

- It is often found in digital circuits
  - Results of setup time and hold time
- The default delay model in both Verilog and VHDL
- Used to filter unwanted spikes and transients on signals

Transport Delay

- Models hardware that does not exhibit inertial delay
- Represents pure propagation delay
- Example:
  
  \[ Z \leftarrow \text{transport} \ A \text{ after} \ 10 \text{ ns}; \]
Transport and Inertial Delays

Signal name <= reject pulse-width expression after delay-time;
Z1 <= transport X after 10 ns; -- transport delay
Z2 <= X after 10 ns; -- inertial delay
Z3 <= reject 4ns X after 10 ns;

Value Change Dump (VCD) Files

- VCD files can record every value change during simulation
  - Changing at which time
  - Changing to what value
- A simple ASCII format
- Widely used for post-processing after simulation
  - Waveform viewer
An Example of Dumpfiles

Sdate
March 3, 2002 10:08:08
Send
Sversion
VERILOG-XL 3.10.p001
Send
Stimescale
1ns
Send no. of bits
Scope module test Send
Var reg 1! X Send
Var reg 1" Z1 Send
Var reg 1# Z2 Send
Var reg 1$ Z3 Send
Supscope Send
Senddefinitions Send

What is PLI?

PLI = Programming Language Interface

PLI is an interface mechanism for users to

- Link their applications to the simulator
- Share the internal data structure of the simulator
Verilog PLI Routines

In Verilog, there are three kinds of PLI routines:

- TF (task/function) routines
- ACC (access) routines
- VPI (Verilog procedural interface) routines

TF and ACC routines are supported by almost all simulators
- Often classified as PLI 1.0

VPI routines have better capabilities for users but not supported by all simulators
- Often classified as PLI 2.0

How to Use the PLI?

1. Create your C routine
2. Make table entries in veriuser.c
3. Use vconfig to compile and link the simulator, your C routine, and veriuser.c
4. New Verilog executable
5. Run the new Verilog with the HDL description

/* in file hdl.v */
module test ;
......
initial begin
......
   $hello ;
end
endmodule
Your C Routine

```c
#include <stdio.h>
#include "veriuser.h"

int hello()
{
    io_printf("hello from the task!!\n") ;
    return 0;
}
```

It works the same way as the C `printf` routine except that it writes output to both standard output and log file output.

The `veriuser.c` File

```c
#include "veriuser.h"
#include "vxl_veriuser.h"

extern int hello();

s_tfcell veriusertfs[TF_MAXARRAY] =
{
    /*** Template for an entry:
        { usertask|userfunction, data, checktf( ), sizetf( ), calltf( ),
          misctf( ), "$tfname", forwref?, Vtool?, ErrMsg? }, ***/
    /*** add user entries here ***/
    { usertask, 0, 0, 0, hello, 0, "$hello", 1},
    {0} /*** final entry must be 0 ***/
};
```

always set to 1
function declaration
task type
task name
function name

3-37

3-38
Compiling and Linking with \textit{vconfig}

Type \textit{vconfig}

Name the shell script (default is \textit{cr_vlog})

Name the executable (default is \textit{verilog})

Enter the path of \textit{veriuser.c}

List any C files you want compiled and linked

new verilog created

Run the shell script

\textit{cr_vlog} shell script created

Running Results

/* in file hdl.v */
module test ;
......
initial begin
......
\textbf{hello} ;
end
endmodule

task invocation

Compiling source file “hdl.v”
Highest level modules: test

\textbf{hello from the task}

output results
Passing Data

```haskell
/* in file hdl.v */
module test;
    ....
    initial begin
        $hello(3);
    end
endmodule

extern int hello();
s_tfcell veriusertfs[TF_MAXARRAY] = {
    { usertask, 0, 0, hello, 0, "Hello", 1},
    {0} /*** final entry must be 0 ****/
};

int hello() {
    int i;
    int count = acc_fetch_tfarg_int(1);
    for (i=0; i<count; i++)
        io_printf("Hello from the task!
");
    return 0;
}
```

Special Events

```haskell
extern int my_misctf ( int data, int reason );
s_tfcell veriusertfs[TF_MAXARRAY] = {
    { usertask, 1, 0, 0, 0, (int (*)(void)) my_misctf, "Task1", 1},
    { usertask, 2, 0, 0, 0, (int (*)(void)) my_misctf, "Task2", 1},
    {0} /*** final entry must be 0 ****/
};
```
Special Events (cont’d)

```c
int my_misctf ( int data, int reason ) {
    char *task_name;
    switch (data) {
        case 1: task_name = "$task1"; break;
        case 2: task_name = "$task2"; break;
    }
    switch (reason) {
        case REASON_ENDOFCOMPILE:
            io_printf("End compile from %s\n",task_name);
            break;
        case REASON_FINISH:
            io_printf("Finish for %s\n",task_name);
            break;
    }
    return 0;
}
```

Running Results

```c
/* in file hdl.v */
module test ;
......
initial $task1;
initial $task2;
endmodule
```

Compiling source file “hdl.v”
Highest level modules:
test

end compile from $task1
finish for $task2

Task invocation
Output results
Case Study -- Coverage Analysis

- Insert PLI TF routines ≈ set break points in the program
  - Can do the jobs we want at the stops
  - Supported by almost all simulator

- Where to insert those TF routines is an important issue
  - Require a pre-processing tool

- Conduct the pre-processing and execute the simulation with the instrumented HDL code can obtain the required coverage data

Coverage Analysis Flow

1. Verilog Code
2. Instrumentation Tool
3. User-Requirement (metrics, region…)
4. modified HDL code
5. Instrumented Code
6. Simpler simulator
7. Simulator with special PLI
8. GUI or Text
9. Coverage Report
The veriuser.c File

#include "veriuser.h"
#include "vxl_veriuser.h"

extern int setup ( int data, int reason );
extern int count ( );
s_tfcell veriusertfs[TF_MAXARRAY] = {
    { usertask, 0, 0, 0, 0, (int (*)(void)) setup, "$setup_count", 1},
    { usertask, 1, 0, 0, count, 0, "$count_block", 1},
    {0} /*** final entry must be 0 ***/
};

The C Routines

#include <stdio.h>
#include <stdlib.h>
#include "veriuser.h"
#include "acc_user.h"

int blockno = 0;
int *binfo = NULL;

int count( )
{
    int cnt = acc_fetch_tfarg_int(1);
    if (!binfo) return -1;
    else binfo[cnt]++;
    return 0;
}
The C Routines (cont’d)

int setup(int data, int reason) {
    int i;  FILE *fp = NULL;
    switch (reason) {
    case REASON_ENDOFCompile:
        fp = fopen (“_jimmy_blockno.dat”,”r”);
        fscanf (fp,”%d”,&blockno);  fclose (fp);
        binfo = (int *) malloc (blockno*sizeof(int));
        for (i=0; i<blockno; i++) binfo[i]=0;
        break;
    case REASON_FINISH:
        if (binfo) {
            fp = fopen (“_jimmy_execount.dat”,”w”);
            for (i=0; i<blockno; i++) fprintf (fp,”b%d %d
”,i,binfo[i]);
            fclose (fp);  free (binfo);
        }
        break;
    }
    return 0;
}

A FSM Example

module fsm ( found, serial, clk, reset ) ;
output     found ;
input      serial, clk, reset ;
reg found ;
reg [1:0]   current_state, next_state ;
parameter [1:0]
S0 = 0,
S1 = 1,
S2 = 2 ;
always @ ( reset or serial or current_state )
begin
    if ( reset ) begin
        next_state = S0 ;
        found = 0 ;
    end
else begin
    next_state = current_state ;
    found = 0 ;
case ( current_state )
S0 : begin
    if ( serial == 1 ) next_state = S2 ;
end
S2 : begin
    if ( serial == 0 ) next_state = S1 ;
    else next_state = S2 ;
end
S1 : begin
    next_state = S0 ;
    if ( serial == 1 ) found = 1 ;
end
endcase
always @(posedge clk)
    current_state = next_state ;
endmodule
Instrumented Code

always @ ( reset or serial or current_state )
begin
  if ( reset ) begin
    next_state = S0 ;
    found = 0 ;
    $count_block(0);
  end
  else begin
    next_state = current_state ;
    found = 0 ;
    case ( current_state )
      S0 : begin
        if ( serial == 1 ) begin
          next_state = S2 ;
          $count_block(1);
        end
        $count_block(2);
      end
      S2 : begin
        if ( serial == 0 ) begin
          next_state = S1 ;
          $count_block(3);
        end
        else begin
          next_state = S2 ;
          $count_block(4);
        end
        $count_block(5);
      end
      S1 : begin
        next_state = S0 ;
        if ( serial == 1 ) begin
          found = 1 ;
          $count_block(6);
        end
        $count_block(7);
      end
    endcase
    $count_block(8);
  end
  $count_block(9);
end

always @ ( posedge clk ) begin
  current_state = next_state ;
  $count_block(10);
end

Test Patterns

module test;
  reg serial, clk, reset ;
  wire found ;
  fsm u1 ( found, serial, clk, reset ) ;
  always #5 clk=~clk;
  initial begin
    clk=0; reset=0; serial=0;
    #2  reset=1;
    #10 reset=0;
    #20 serial=1;
    #10 serial=0;
    #30 $finish;
  end
endmodule

module test;
  reg serial, clk, reset ;
  wire found ;
  fsm u1 ( found, serial, clk, reset ) ;
  always #5 clk=~clk;
  initial begin
    clk=0; reset=0; serial=0;
    #2  reset=1;
    #10 reset=0;
    #20 serial=1;
    #10 serial=0;
    #30 $finish;
  end
endmodule

original

Add this line
Temporary Files

<table>
<thead>
<tr>
<th><em>jimmy_blockno.dat</em></th>
<th><em>jimmy_execount.dat</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>b0 2</td>
</tr>
<tr>
<td>b0 32</td>
<td>b1 1</td>
</tr>
<tr>
<td>b1 41</td>
<td>b2 3</td>
</tr>
<tr>
<td>b2 40</td>
<td>b3 1</td>
</tr>
<tr>
<td>b3 44</td>
<td>b4 1</td>
</tr>
<tr>
<td>b4 45</td>
<td>b5 2</td>
</tr>
<tr>
<td>b5 43</td>
<td>b6 0</td>
</tr>
<tr>
<td>b6 49</td>
<td>b7 1</td>
</tr>
<tr>
<td>b7 47</td>
<td>b8 7</td>
</tr>
<tr>
<td>b8 36</td>
<td>b9 9</td>
</tr>
<tr>
<td>b9 31</td>
<td>b10 7</td>
</tr>
<tr>
<td>b10 55</td>
<td>block number</td>
</tr>
<tr>
<td></td>
<td>starting line</td>
</tr>
<tr>
<td></td>
<td>of this block</td>
</tr>
<tr>
<td></td>
<td>total number</td>
</tr>
<tr>
<td></td>
<td>of blocks</td>
</tr>
<tr>
<td></td>
<td>block number</td>
</tr>
<tr>
<td></td>
<td>execution count</td>
</tr>
<tr>
<td></td>
<td>of this block</td>
</tr>
</tbody>
</table>

Output Results

statement coverage information ...

<table>
<thead>
<tr>
<th>count</th>
<th>#line</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td></td>
<td>module fsm ( found, serial, clk, reset );</td>
</tr>
<tr>
<td></td>
<td></td>
<td>output found;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>input serial, clk, reset;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reg found;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reg [1:0] current_state;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reg [1:0] next_state;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameter S0 = 0;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameter S1 = 1;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameter S2 = 2;</td>
</tr>
<tr>
<td>31</td>
<td></td>
<td>always</td>
</tr>
<tr>
<td>31</td>
<td></td>
<td>@ ( reset or serial or current_state )</td>
</tr>
<tr>
<td>31</td>
<td></td>
<td>begin</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>if ( reset ) begin</td>
</tr>
<tr>
<td>33</td>
<td></td>
<td>next_state = S0 ;</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>found = 0 ;</td>
</tr>
<tr>
<td>34</td>
<td></td>
<td>end</td>
</tr>
<tr>
<td>35</td>
<td></td>
<td>else begin</td>
</tr>
<tr>
<td>36</td>
<td></td>
<td>next_state = current_state ;</td>
</tr>
<tr>
<td>7</td>
<td>37</td>
<td>found = 0 ;</td>
</tr>
<tr>
<td>37</td>
<td></td>
<td>case (current_state)</td>
</tr>
<tr>
<td>7</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td>S0 : begin</td>
</tr>
<tr>
<td>41</td>
<td></td>
<td>if ( serial == 1 )</td>
</tr>
<tr>
<td>41</td>
<td></td>
<td>next_state = S2 ;</td>
</tr>
<tr>
<td>42</td>
<td></td>
<td>end</td>
</tr>
<tr>
<td>43</td>
<td></td>
<td>S2 : begin</td>
</tr>
<tr>
<td>44</td>
<td></td>
<td>if ( serial == 0 )</td>
</tr>
<tr>
<td>44</td>
<td></td>
<td>next_state = S1 ;</td>
</tr>
<tr>
<td>45</td>
<td></td>
<td>else</td>
</tr>
<tr>
<td>46</td>
<td></td>
<td>end</td>
</tr>
<tr>
<td>47</td>
<td></td>
<td>S1 : begin</td>
</tr>
<tr>
<td>48</td>
<td></td>
<td>next_state = S0 ;</td>
</tr>
<tr>
<td>49</td>
<td></td>
<td>if ( serial == 1 )</td>
</tr>
<tr>
<td>49</td>
<td></td>
<td>found = 1 ;</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>end</td>
</tr>
<tr>
<td>51</td>
<td></td>
<td>endcase</td>
</tr>
<tr>
<td>52</td>
<td></td>
<td>end</td>
</tr>
<tr>
<td>53</td>
<td></td>
<td>end</td>
</tr>
<tr>
<td>55</td>
<td></td>
<td>always</td>
</tr>
<tr>
<td>55</td>
<td></td>
<td>@ ( posedge clk )</td>
</tr>
<tr>
<td>56</td>
<td></td>
<td>endmodule</td>
</tr>
</tbody>
</table>

statement coverage = 9 / 10 = 90%