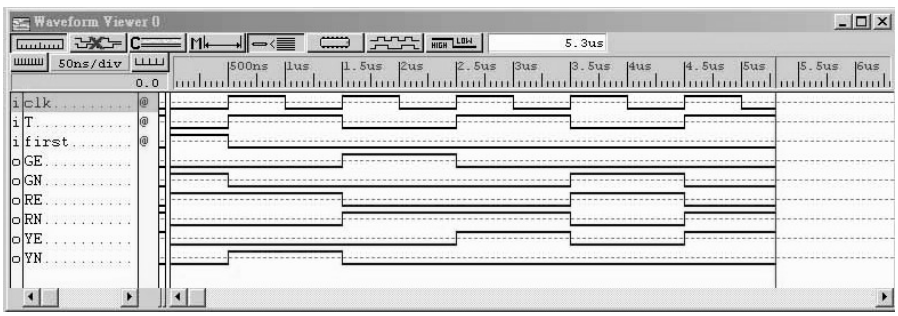
Logic Simulation

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

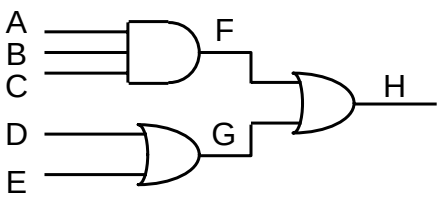
- Design verification:
  - Function
  - Timing
  - Independent of initial states
  - Insensitive to parameter variation
  - Race and hazard free
- Other purposes:
  - Evaluate alternative designs
  - Evaluate design changes
  - Documentation
  - ……

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

```
Load    A
AND     B
AND     C
Store   F
Load    D
OR      E
INV
OR      F
Store   H
```



H = Simulation (1,0,1,0,1)

3-5

Table-Driven Simulation

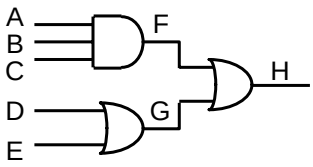
- An interpreter is used to translate the circuit into internal data structure (table)
- Simulation is done by manipulating such data structure in a predetermined order
  - Called the *"**topological order**"*
- Cannot be applied to sequential circuit simulation
  - Feedback cannot be handled

3-6

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 to n)
  Evaluate Gate i



Node	Type	Fanin	Fanout	Value
A	PI	-	F	1
B	PI	-	F	0
C	PI	-	F	1
D	PI	-	G	0
E	PI	-	G	1
F	AND	A,B,C	H	**0**
G	OR	D,E	H	**1**
H	OR	F,G	-	**1**

3-7

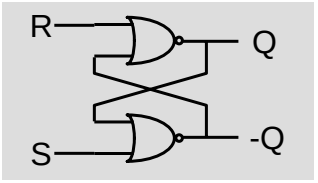
Event-Driven Simulation

- 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 ***time wheel***
  - Advance to the time next event is scheduled
  - Process that event
- Only process the activated part of the circuit

3-8

Algorithms for EDS

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



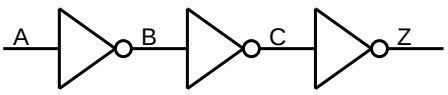
While (event-list != 0)
- advance time wheel
- determine current events
- update values
- propagate events
- evaluate active events
- schedule new events

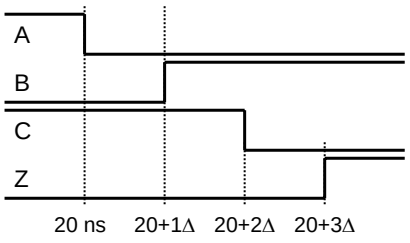
Time	R	S	Q	-Q
0	0	1	1	0
1	-	0	-	-
2	1	-	-	-
2.1	-	-	0	-
2.2	-	-	-	1
3	0	-	-	-
4	-	1	-	-
4.1	-	-	-	0
4.2	-	-	1	-

3-9

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

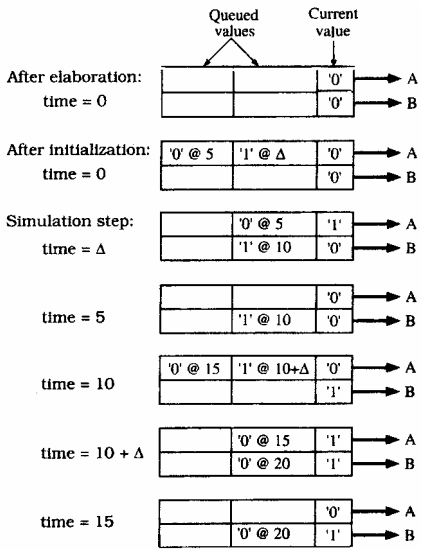




3-10

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 ***Ei*** pending at ***t***)
    - Assign ***Ni*** value ***Vi***
    - Put ***Fanout j*** into ***Activated***
    - for (every ***j*** in ***Activated***)
      - Evaluate ***Vj***
      - if (***Vj' != Vj***) Schedule ***Ej*** at time ***t + Dj***

Gate-Level Event Driven Simulation

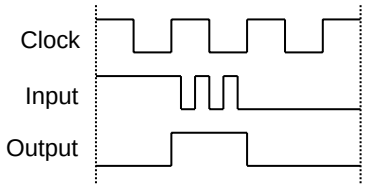
- Disadvantage
  - A node may be schedule more than once at one time slot if there are more than one input change
  - A node may be scheduled more than once within the gate inertial delay
- Solution 1
  - Sort the gates/nodes according to signal dependency
  - Give each gate a index to indicate its ***logic 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

Gate-Level Event Driven Simulation

- 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 happing 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

3-15

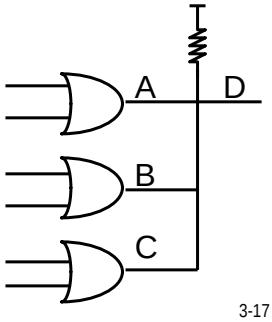
Logic Values

- 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

3-16

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

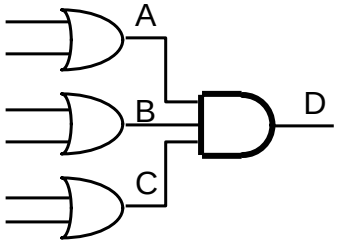


3-17

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;
```



3-18

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

3-19

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

3-20

Resolution Function

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

```
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', 'X' ), -- | 0 |
         ( 'U', 'X', 'X', '1', '1', '1', '1', '1', '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:

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

4-Valued Logic

- Truth table for 4-valued logic:

AND	'X'	'0'	'1'	'Z'
'X'	'X'	'0'	'X'	'X'
'0'	'0'	'0'	'0'	'0'
'1'	'X'	'0'	'1'	'X'
'Z'	'X'	'0'	'X'	'X'

OR	'X'	'0'	'1'	'Z'
'X'	'X'	'X'	'1'	'X'
'0'	'X'	'0'	'1'	'X'
'1'	'1'	'1'	'1'	'1'
'Z'	'X'	'X'	'1'	'X'

3-23

Control Values

- Control value C will dominate the output value
  - Use ***input scan*** first can determine the output values more quickly
  - Example

*NAND (***Ai***)*
for every input **Ai**
if **Ai = C** *then retuen* **Output=1**
return **Output = 0**

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

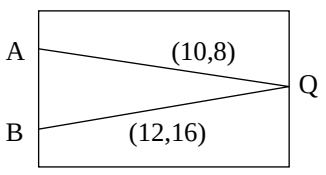
3-24

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

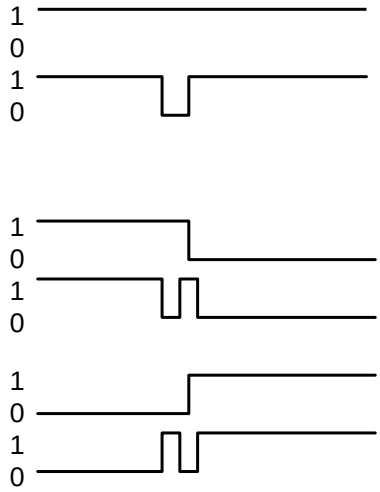
Delay Assignments

- 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

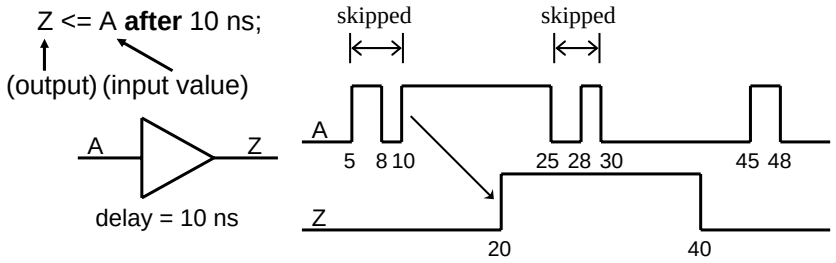
- Static Hazard -
  - Unnecessary transition when there is no signal change
- Dynamic Hazard -
  - Unnecessary extra transition when signal changes from 0 to 1 or from 1 to 0



3-27

Inertial Delay

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



3-28

Inertial Delay

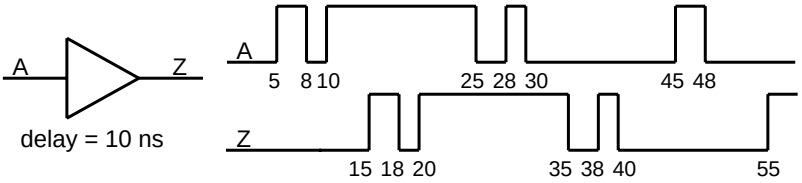
- It is often found in digital circuits
  - Results of setup time and hold time
- The default delay model in both Verilog and VHDL
- Used to filter unwanted spikes and transients on signals

3-29

Transport Delay

- Models hardware that does not exhibit inertial delay
- Represents pure propagation delay
- Example:

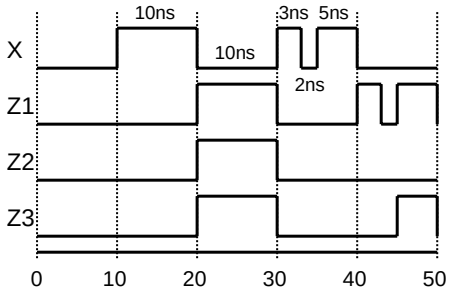
  Z <= **transport** A **after** 10 ns;



3-30

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;



3-31

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

```
# time1
  <new_value><variable1>
  <new_value><variable2>
        ⋮
# time2
  <new_value><variable1>
  <new_value><variable2>
        ⋮
```

3-32

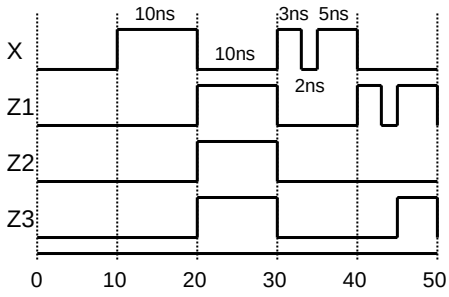
An Example of Dumpfiles

```
$date
   March 3, 2002  10:08:08
$end
$version
   VERILOG-XL 3.10.p001
$end
$timescale
    1ns
$end
```

no. of bits

```
$scope module test $end
$var reg   1  !   X  $end
$var reg   1  "   Z1  $end
$var reg   1  #   Z2  $end
$var reg   1  $   Z3  $end
$upscope $end
$enddefinitions $end
```

```
$dumpvars   #10   #30   #35   #45
0!          1!    1!    1!    1"
0"          #20   0"    #40   1$
0#          0!    0#    0!    #50
0$          1"    0$    1"    0"
$end        1#    #33   #43   0$
            1$    0!    0"
```



3-33

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

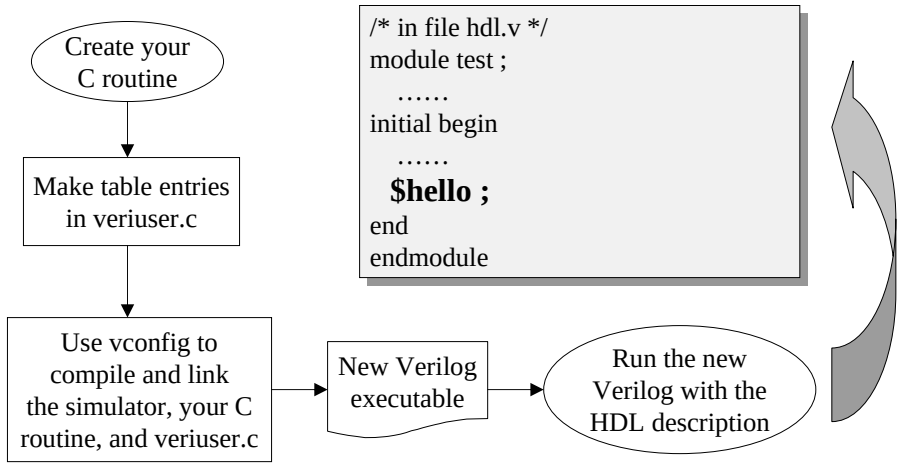
3-34

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

3-35

How to Use the PLI ?



3-36

Your C Routine

```
#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.

3-37

The *veriuser.c* File

```
#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 ***/
};
```

function declaration

always set to 1

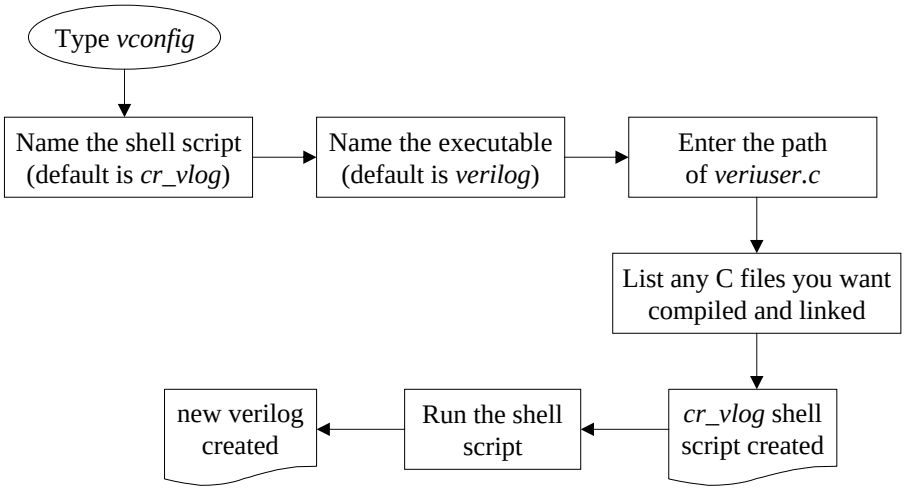
task type

task name

function name

3-38

Compiling and Linking with *vconfig*



3-39

Running Results

```
/* in file hdl.v */
module test ;
   ……
initial begin
   ……
  $hello ;
end
endmodule
```

task invocation

```
Compiling source file "hdl.v"
Highest level modules:
test

hello from the task
```

output results

3-40

Passing Data

veriuser.c

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

task invocation

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

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

user-supplied routine

3-41

Special Events

veriuser.c

```
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 ***/
};
```

data field — type transformation — misctf function — task name

3-42

Special Events (cont'd)

```
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;
}
```

user-supplied routine

3-43

Running Results

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

task invocation

```
Compiling source file "hdl.v"
Highest level modules:
test

end compile from $task1
finish for $task2
```

output results

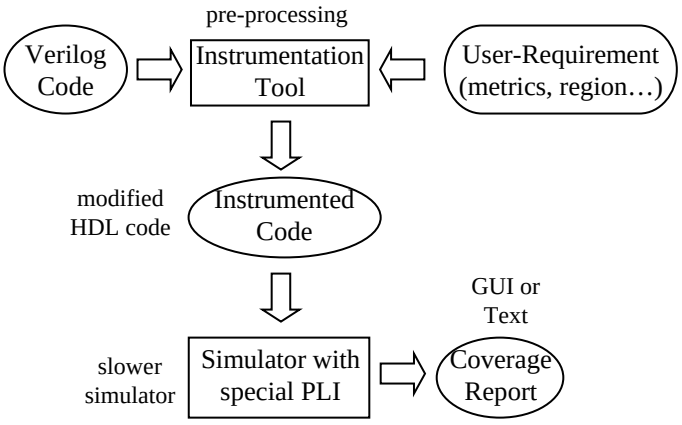
3-44

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

3-45

Coverage Analysis Flow



3-46

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\n",i,binfo[i]);
      fclose (fp);  free (binfo);
    }
    break;
  }
  return 0;
}
```

3-49

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
  end
end

always @(posedge clk)
  current_state = next_state ;

endmodule
```

3-50

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
```

3-51

Test Patterns

original

```
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
```

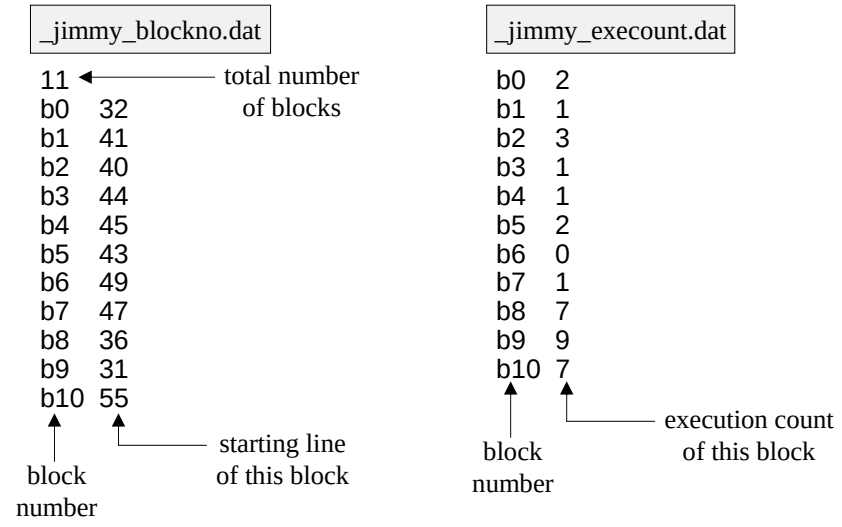
instrumented

```
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
initial $setup_count;
endmodule
```

Add this line

3-52

Temporary Files



3-53

Output Results

```
statement coverage information ...
count #line      text
------------------------------------------------------------
        21  module fsm ( found, serial, clk, reset );
            output found;
            input serial, cllk, reset;
            reg found;
            reg [1:0] current_state;
            reg [1:0] next_state;

            parameter S0 = 0 ;
            parameter S1 = 1 ;
            parameter S2 = 2 ;

        31  always
        31  @ ( reset or serial or current_state )
        31    begin
        32      if ( reset ) begin
    2   33          next_state = S0 ;
    2   34          found = 0 ;
        35      end
        36      else begin
    7   37          next_state = current_state ;
    7   38          found = 0 ;
        39          case (current_state)
        40          S0 : begin
        41            if ( serial == 1 )
    1   41              next_state = S2 ;
        42          end
        43          S2 : begin
        44            if ( serial == 0 )
    1   44              next_state = S1 ;
        45            else
    1   45              next_state = S2 ;
        46          end
        47          S1 : begin
    1   48            next_state = S0 ;
        49            if ( serial == 1 )
 **0**  49              found = 1 ;
        50          end
        51          endcase
        52        end
        53    end

        55  always
        55  @ ( posedge clk )
    7   55    current_state = next_state ;
        56  endmodule
------------------------------------------------------------
statement coverage = 9 / 10 = 90%
```

3-54