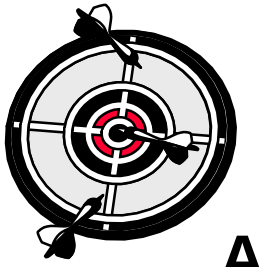
Agenda

DAY 1

1. The Device Under Test (DUT)
2. SystemVerilog Verification Environment
3. SystemVerilog Language Basics - 1
4. SystemVerilog Language Basics - 2

Unit Objectives



After completing this unit, you should be able to:

- **Describe the process of reaching verification goals**
- **Describe components of a SystemVerilog testbench environment**
- **Describe `program` and `interface` constructs**
- **Compile and simulate a SV testbench**
- **Drive and sample DUT signals**
- **Synchronize to known point in simulation**

Verification Goals (1/2)

- **Verify RTL design code**
  - Fully conforms with specifications
- **Must avoid false positives (untested functionalities)**

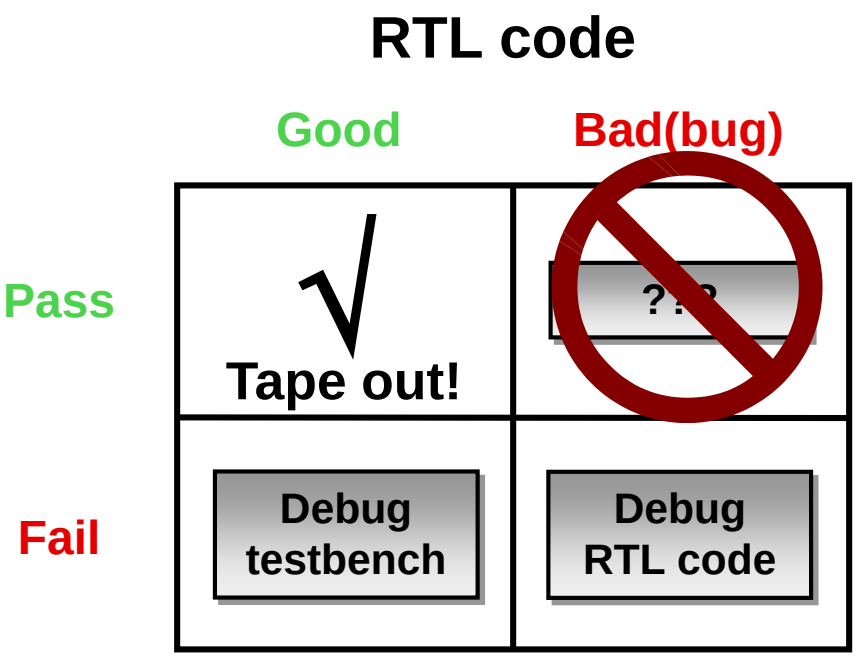
Testbench Simulation result	RTL code: Good	RTL code: Bad(bug)
Pass	√ Tape out!	???
Fail	Debug testbench	Debug RTL code

False positive results in shipping a bad design

How do we achieve this goal?

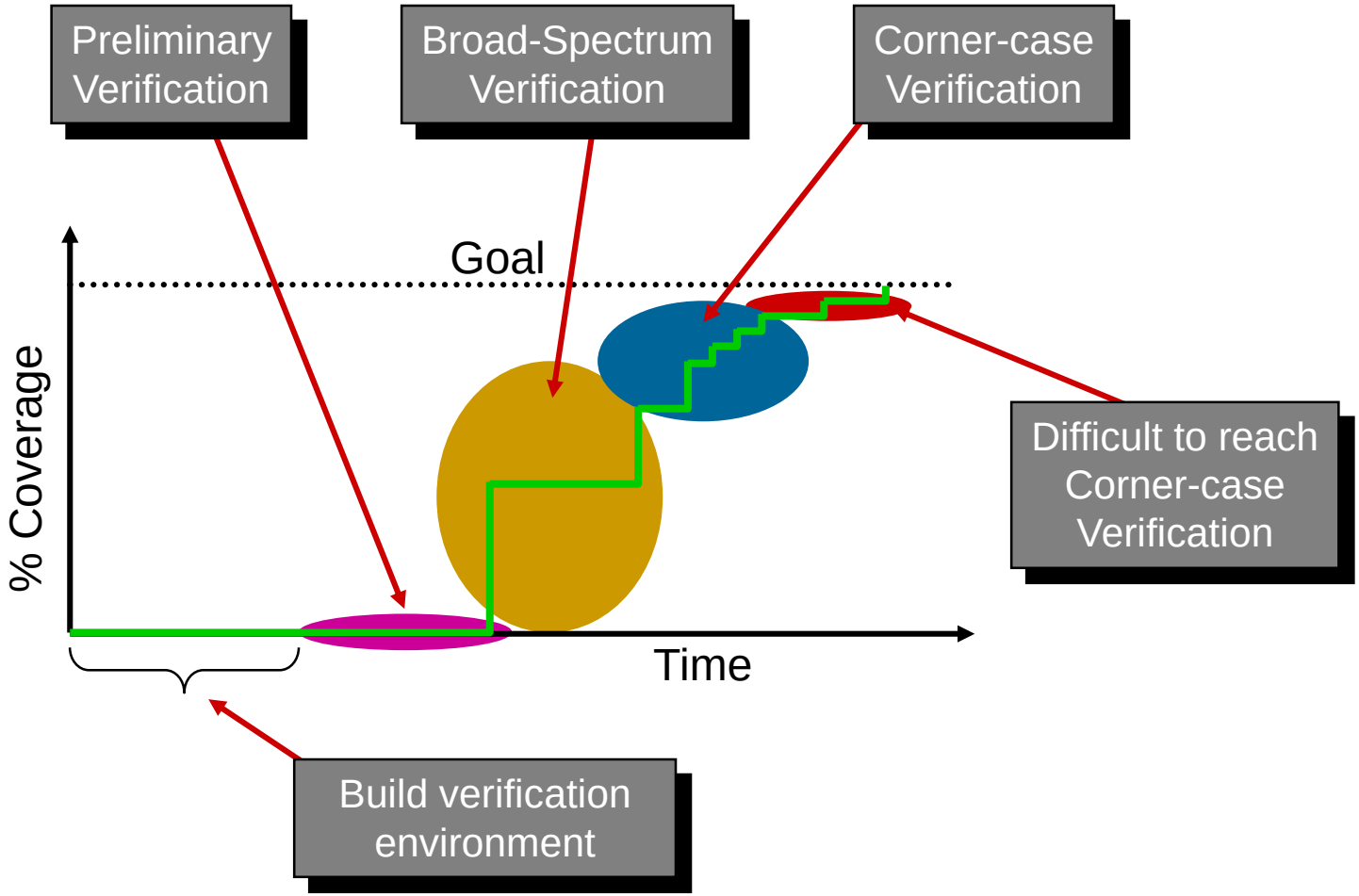
Verification Goals (2/2)

- **Test Environment must:**
  - Be structured for Debug
  - Avoid False Positives
- **Tests must:**
  - Achieve Functional Coverage
    - Prevent untested regions
  - Reach Corner Cases
    - Anticipated Cases
    - Error Injection
      - Environment Error
      - DUT Error
    - Unanticipated Cases
      - Random Tests
  - Be robust, reusable, scalable

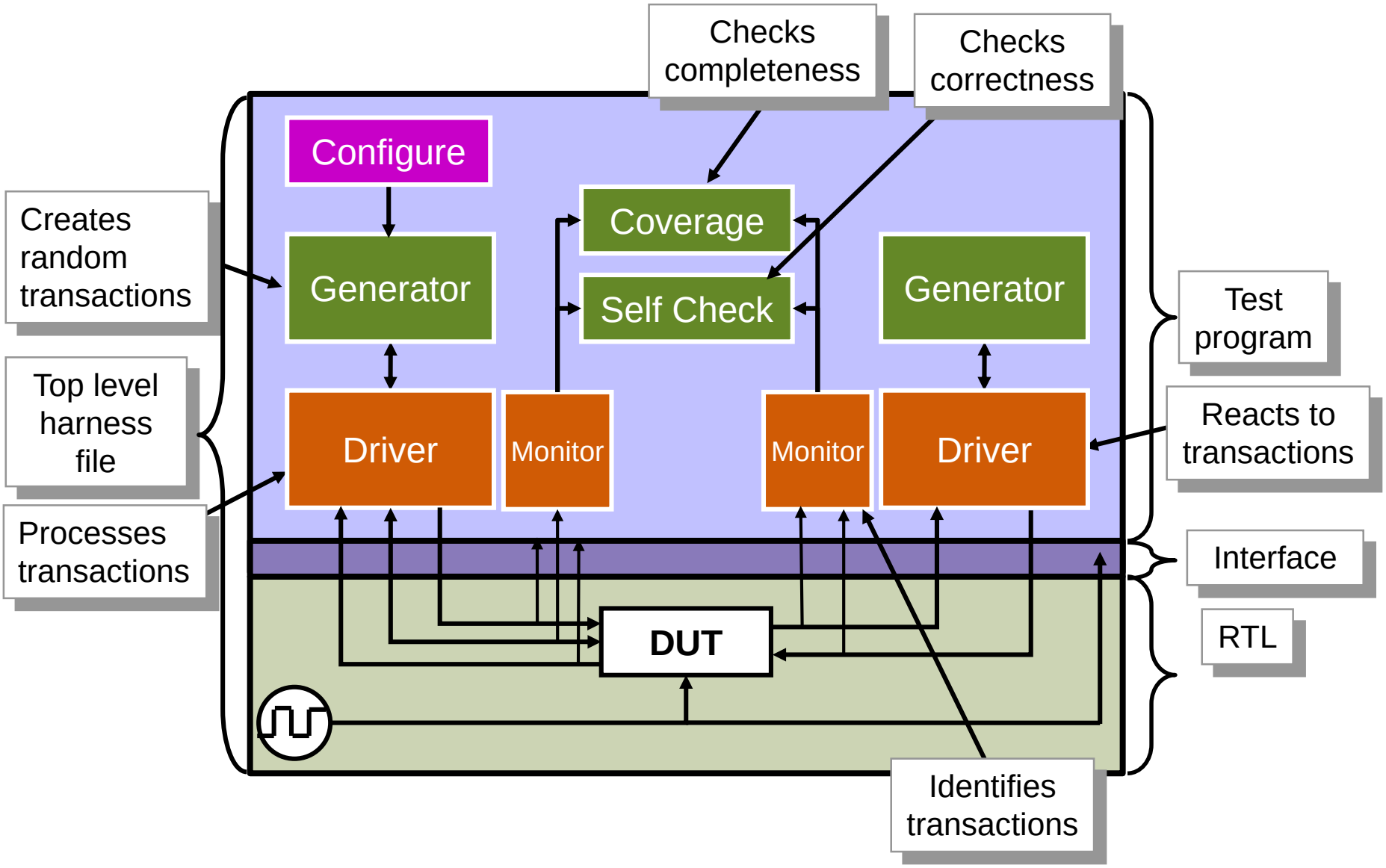


Process of Reaching Verification Goals

Phases of verification

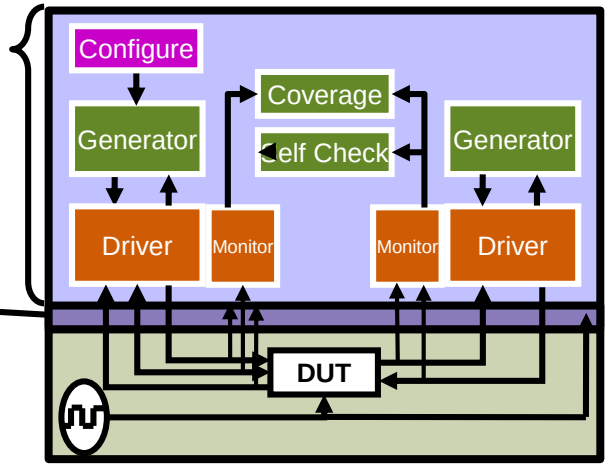


The SystemVerilog Test Environment



SystemVerilog – Key Features

- **SystemVerilog introduces two new design units**
  - The **`program`** block
    - Is where you develop testbench code
    - Is entry point for testbench execution
  - The **`interface`**
    - Is mechanism to connect testbench to DUT
    - Is a named bundle of wires
    - Can be passed just like a port in a port list



- **SystemVerilog programs use Object Oriented Programming (OOP)**
  - Uses **`class`** definitions
    - discussed later in this workshop

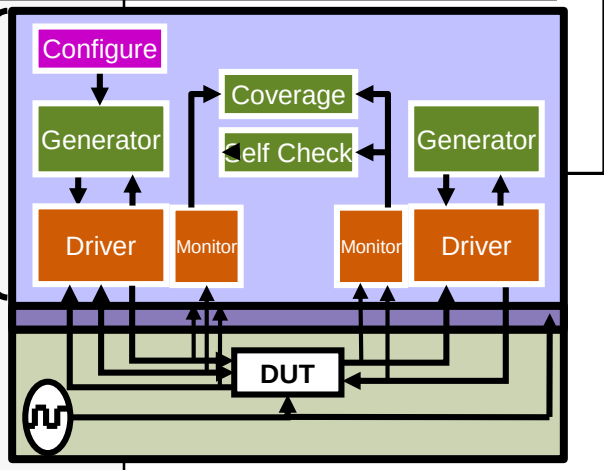
Program Block – Encapsulate Test Code

- **The `program` block provides**
  - Entry point to test execution
  - Scope for program-wide data and routines
  - Race-free interaction between testbench and design
- **Develop test code in `program` block**

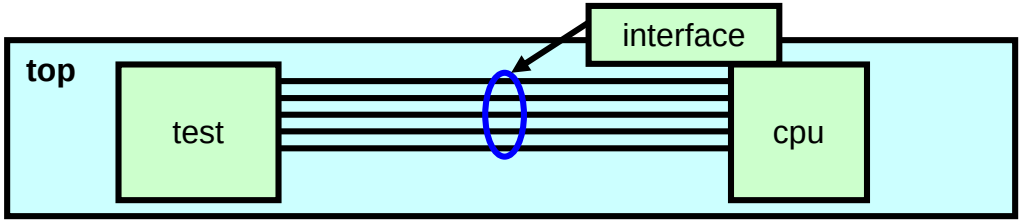
```
program automatic test(router_io.TB rtr_io);
  //testbench code in initial block:
  initial begin
    run();
  end
  task run();
    …
  endtask: run
endprogram: test
```

Optionally pass interface in port list

Only initial blocks allowed in programs



SystemVerilog Key Features – Interface



- **An interface encapsulates the communication between DUT and Testbench including**
  - Connectivity (signals) – named bundle of wires
    - One or more bundles to connect modules and tests
    - Can be reused for different tests and devices
  - Directional information (modports)
  - Timing (clocking blocks)
  - Functionality (routines, assertions, initial/always blocks)
- **Solves many problems with traditional connections**
  - Port lists for the connections are compact
  - No missed connections
  - Easy to add new connections

Comparing SystemVerilog Containers

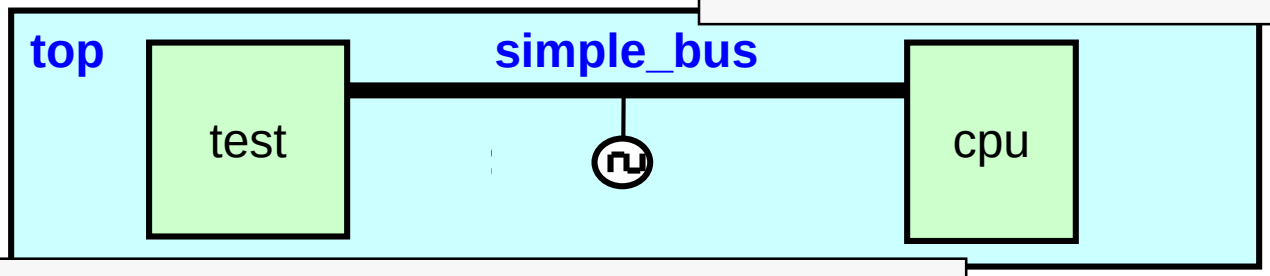
← Hardware (DUT)		Testbench →	
← static			dynamic →
`module`	**`interface`**	**`program`**	**`class`**
module instance			
interface instance	interface instance		
`class`	**`class`**	**`class`**	**`class`**
object	object	object	object
`reg (logic)`	**`reg (logic)`**	**`reg (logic)`**	**`reg (logic)`**
variable	variable	variable	variable
`wire`	**`wire`**	**`wire`**	
`assign`	**`assign`**	**`assign`**	
`initial`	**`initial`**	**`initial`**	
`always`	**`always`**		
`task`	**`task`**	**`task`**	**`task`**
`function`	**`function`**	**`function`**	**`function`**

Interface – An Example

- The RTL code is connected with bundled signals

```
program automatic test(simple_bus sb);
…
endprogram
```

```
module cpu(simple_bus sb);
…
endmodule
```




```
interface simple_bus(input bit clk);
  logic req, gnt;
  logic [7:0] addr;
  wire  [7:0] data;
  logic [1:0] mode;
  logic start, rdy;
endinterface
```

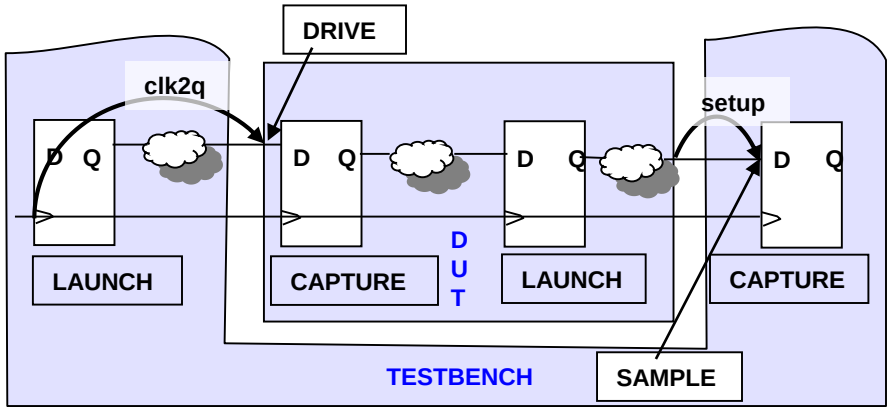
```
module top;
  logic clk = 0;
  always #10 clk = !clk;
  simple_bus sb(clk);
  test t1(sb);
  cpu c1(sb);
endmodule
```

Synchronous Timing: Clocking Blocks

- **Just for testbench**
  - Emulates the launch and capture flops at IO of DUT.
- **Creates explicit synchronous timing domains**
  - All signals driven or sampled at clocking event - by default all interface signals are asynchronous
  - Interaction between testbench and DUT ideally happens only at clock edges (cycle-based)
- **Specifies signal direction**
  - Outputs cannot be sampled
  - Input signals cannot be driven
  - Typically 3 clocking blocks per interface
    - active driver
    - reactive driver
    - monitor

setup Clock to q

```
clocking cb @(posedge clk);
 default input #1ns output #1ns;
 output reset_n;
 output din;
 output frame_n;
 output valid_n;
 input  dout;
 input  busy_n;
 input  valido_n;
 input  frameo_n;
endclocking: cb
```



Signal Direction Using Modport

- Enforce signal access & direction with `modport`

```
interface router_io(input bit clock);
  logic  reset_n;
  …
  clocking cb @(posedge clock);
    default input #1ns output #1ns;
    output reset_n;
    output valid_n;
    …
  endclocking
   modport DUT(input reset_n, input din, output dout,...);
   modport TB(clocking cb, output reset_n);
endinterface: router_io
```

```
program automatic test(router_io.TB rtr_io);
  initial begin
    rtr_io.reset_n = 1'b0;
    rtr_io.cb.reset_n <= 1'b1'
    rtr_io.cb.valid_n <= ~('b0);
  end
endprogram: test
```

```
module router(
 router_io.DUT dut_io,
 input logic clk);
…
endmodule: router
```

A Complete Interface

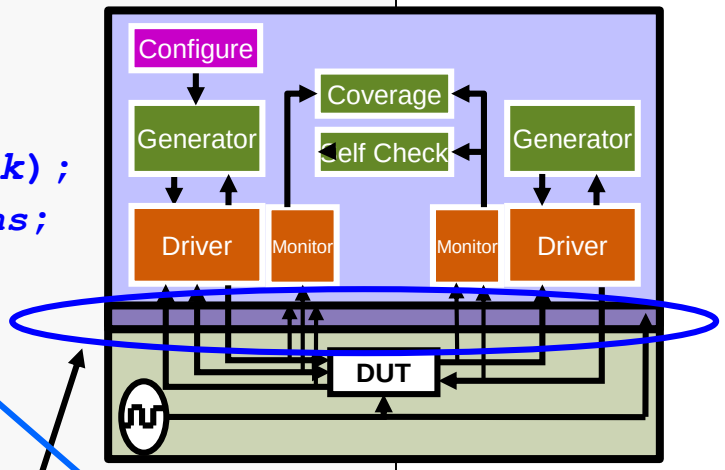
Named bundle of asynchronous signals

Create synchronous behavior by placing into `clocking` block

Define access and direction with `modport`

router_io.sv

```
interface router_io(input bit clock);
  logic  reset_n;
  logic  [15:0]  din;
  logic  [15:0]  frame_n;
  logic  [15:0]  valid_n;
  logic  [15:0]  dout;
  logic  [15:0]  busy_n;
  logic  [15:0]  valido_n;
  logic  [15:0]  frameo_n;
  clocking cb @(posedge clock);
   default input #1ns output #1ns;
    output reset_n;
    output din;
    output frame_n;
    output valid_n;
    input  dout;
    input  busy_n;
    input  valido_n;
    input  frameo_n;
  endclocking
  modport TB(clocking cb, output reset_n);
endinterface
```



Sample and drive skews

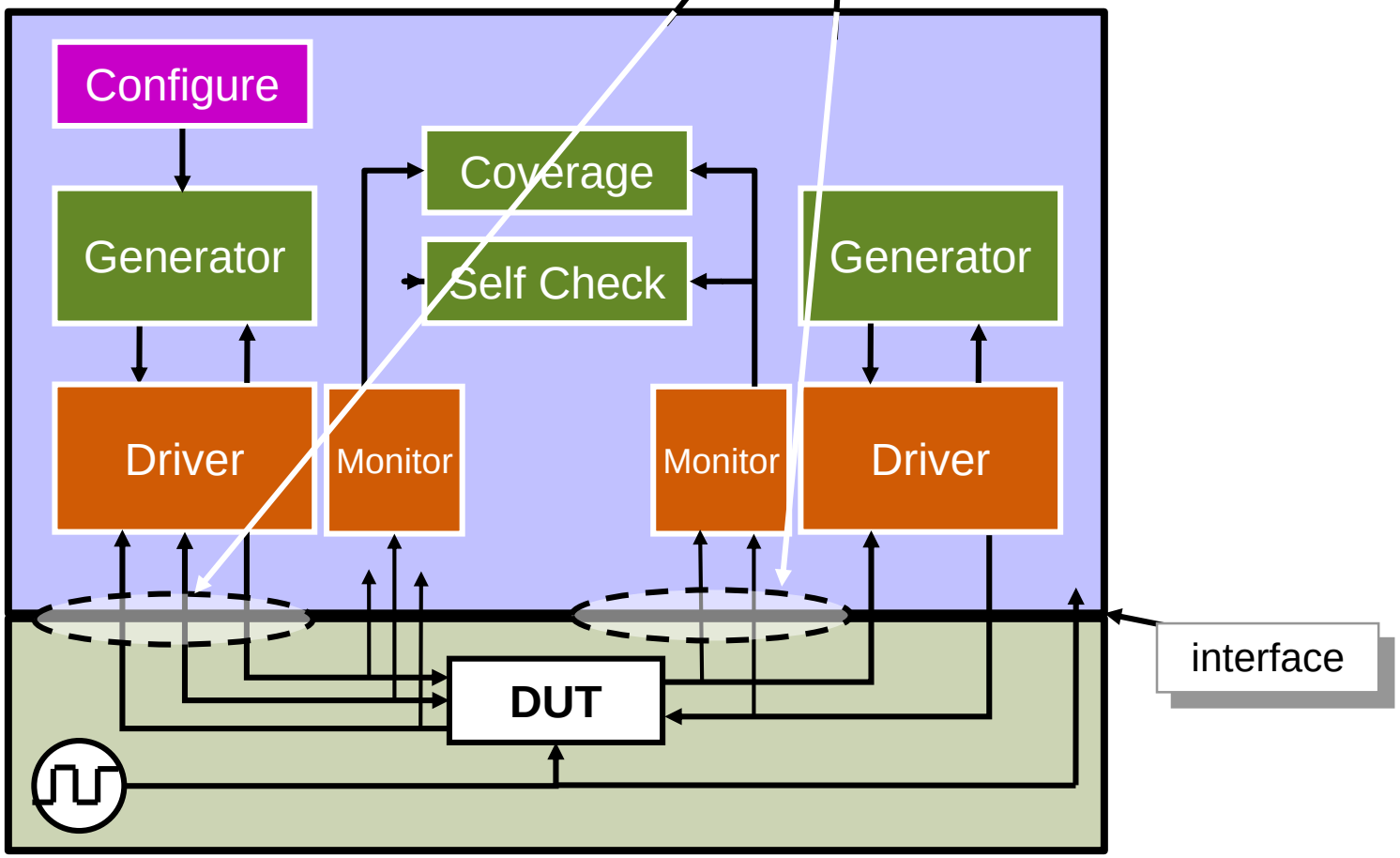
Direction w/respect to test

Synchronous

Asynchronous

Driving & Sampling DUT Signals

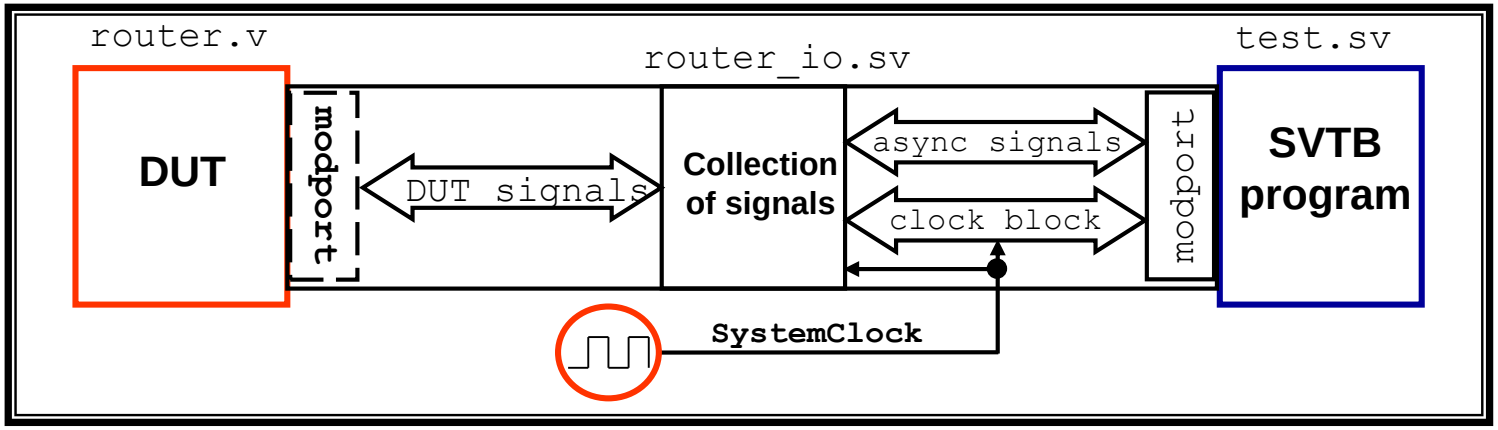
- DUT signals are driven in the device driver
- DUT signals are sampled in the device monitor



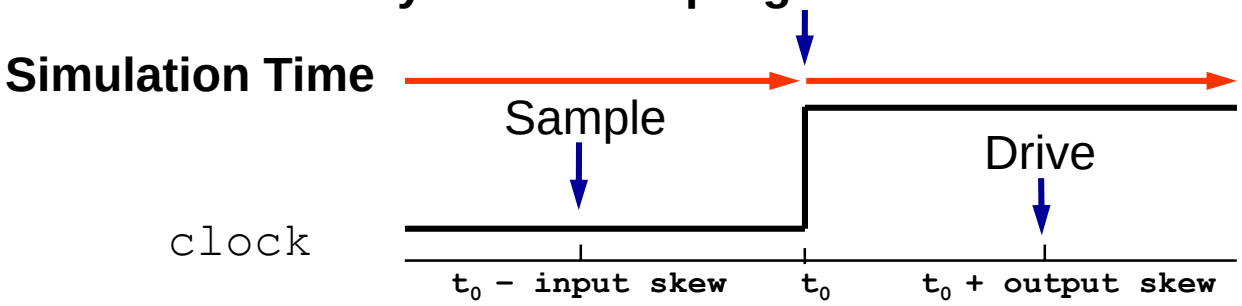
SystemVerilog Testbench Timing

- **Clocking block emulates synchronous drives and samples**
  - Driving and sampling events occur at clocking event

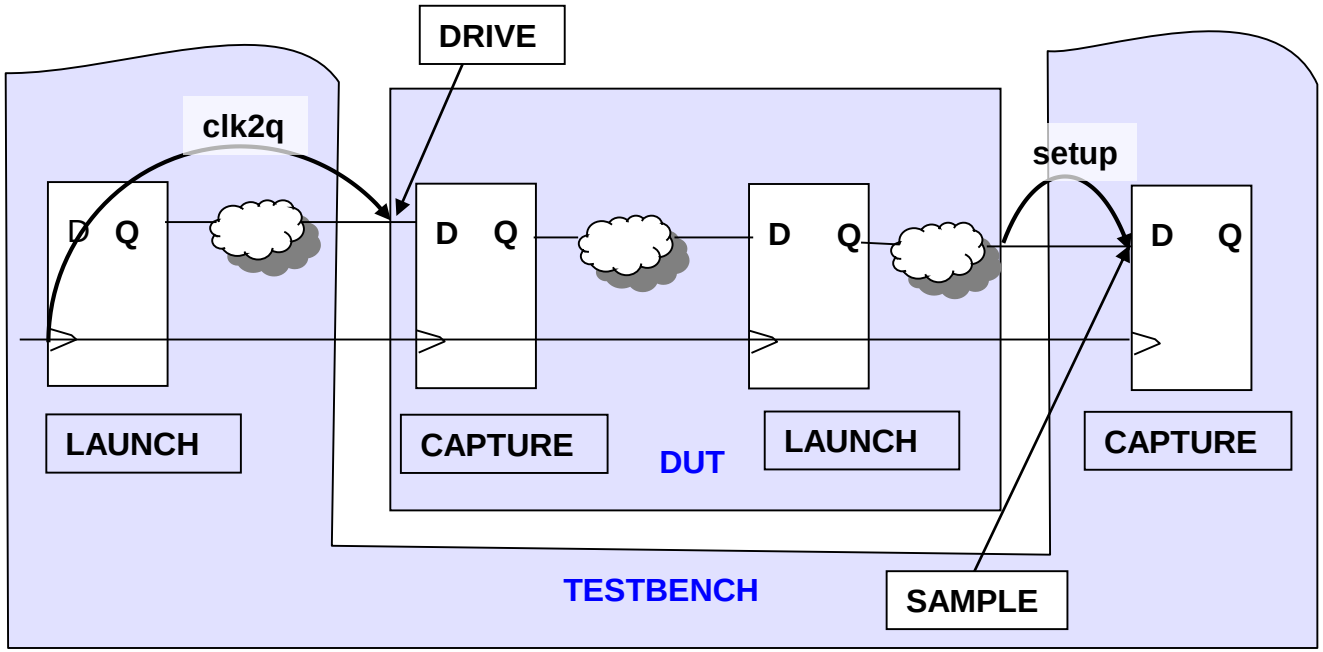
router_test_top.sv



SystemVerilog Testbench (SVTB)
Synchronous program code execution



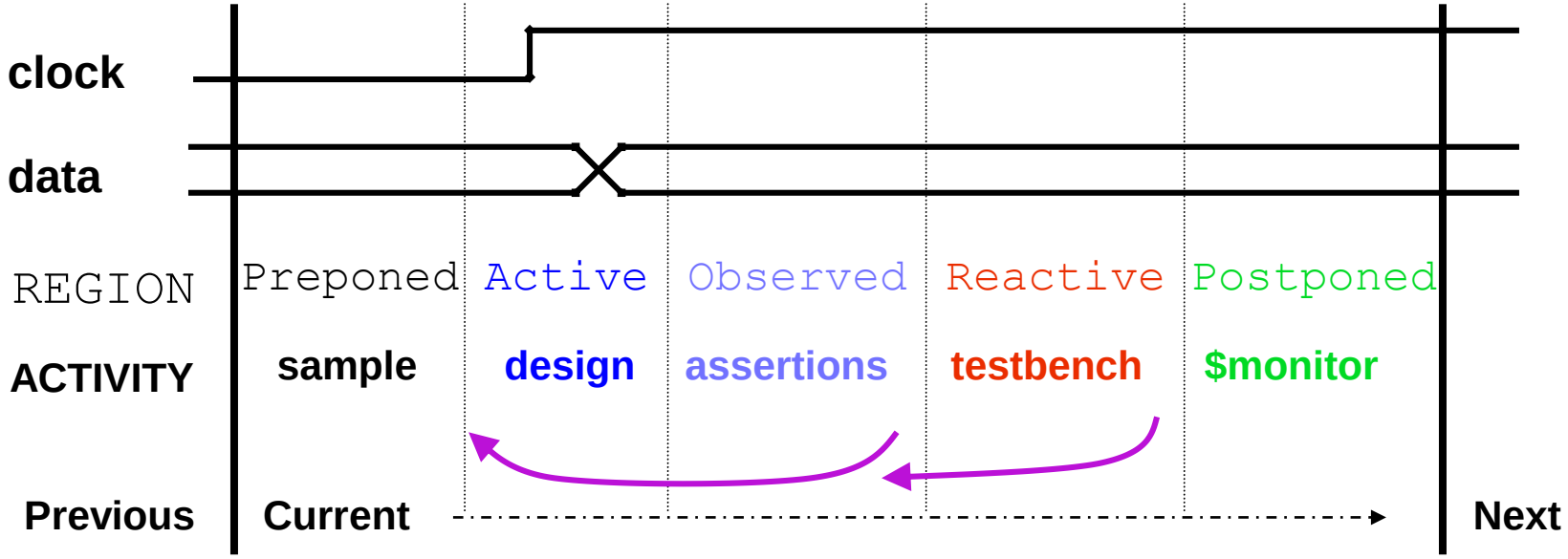
Input and Output Skews



- **Output Skew is the <u>clk2q</u> delay of the launch flop for the DUT input**
  - Defaults to #0
- **Input skew is the <u>setup</u> time of the capture flop for the DUT output**
  - Defaults to #1step – preponed region of simulation step

SystemVerilog Scheduling

- **Each time slot is divided into 5 major regions**
  - **`Preponed`** Sample signals before any changes (**`#1step`**)
  - **`Active`** Design simulation (**`module`**), including NBA
  - **`Observed`** Assertions evaluated after design executes
  - **`Reactive`** Testbench activity (**`program`**)
  - **`Postponed`** Read only phase



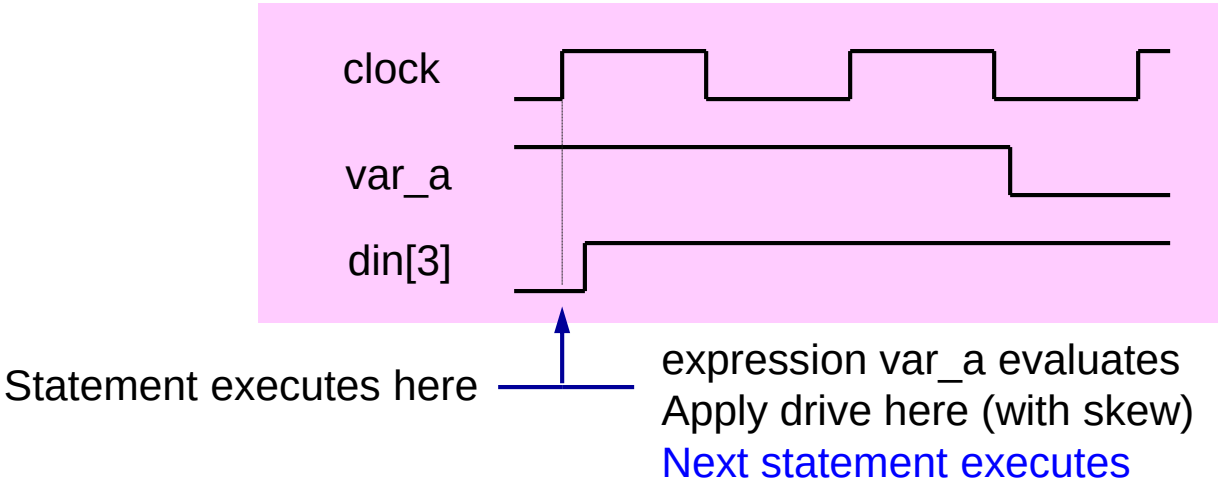
Assertion and testbench events can trigger more design evaluations in this time slot

Synchronous Drive Statements

interface.cb.signal <= <value | expression>;

- **Drive must be non-blocking**
- **Driving of input signal is not allowed**

```
rtr_io.cb.din[3] <= var_a;
```




```
rtr_io.cb.din[3]   = 1'b1;      //error
rtr_io.cb.dout[3] <= 1'b1;      //error
rtr_io.din[3]     <= 1'b1;      //error
```

Sampling Synchronous Signals

***variable* = *interface.cb.signal*;**

- **Variable is assigned the sampled value**
  - value that the clocking block sampled at the most recent clocking event
- **Avoid non-blocking assignment**
- **Sampling of output signal is not allowed**

```
data[i]   = rtr_io.cb.dout[7];
all_data = rtr_io.cb.dout;
frm_out = rtr_io.frameo_n[7];                    //error
$display("din = %b\n", rtr_io.cb.din);           //error
if(rtr_io.cb.din[3] == 1'b0) begin … end         //error
```

Signal Synchronization – Level Sensitive

- **Scheduling region of a clocking variable (signal) is not defined by the LRM**
  - Causes inconsistent behavior between programs and modules
  - Causes inconsistent behavior between simulators from different vendors
  - Solution: Always synchronize to clock first
- **E.g. To synchronize to the low level of a clocking variable *rtr_io.cb.frameo_n*[7]**
  - ALWAYS USE

```
if (rtr_io.cb.frameo_n[7] !== 1'b0)
@(rtr_io.cb iff(rtr_io.cb.frameo_n[7] === 1'b0));
```

  - NEVER USE

```
wait (rtr_io.cb.frameo_n[7] === 1'b0);
```

Signal Synchronization – Edge Sensitive

- **E.g. To synchronize to the negative edge of a clocking block signal *rtr_io.cb.frameo_n*[7]**
  - ALWAYS USE

```
wait (rtr_io.cb.frameo_n[7] !== 1'b0);
@(rtr_io.cb iff(rtr_io.cb.frameo_n[7] === 1'b0));
```

  - NEVER USE

```
@(negedge rtr_io.cb.frameo_n[7]);
```

    - The scheduling of this statement is not clearly defined by the SystemVerilog LRM
    - This may cause inconsistent and unexpected behavior

Using Interface in Program

Pass modport as port list

```
interface router_io(input bit clock);
            logic  reset_n;
            logic  [15:0] din;
                   ] frame_n;
                   ] valid_n;
```

Asynchronous signals are driven without reference to clocking block

```
program automatic test(router_io.TB rtr_io);
  //testbench code in initial block
  initial begin
    reset();
  end
  task reset();
    rtr_io.reset_n = 1'b0;
    rtr_io.cb.frame_n <= 16'hffff;
    rtr_io.cb.valid_n <= ~('b0);
// reset_n can be  both synchronous and asynchro
    repeat(2) @(rtr_io.cb);
    rtr_io.cb.reset_n <= 1'b1;
    repeat(15) @(rtr_io.cb);
  endtask: reset
endprogram:
```

Synchronous signals are referenced via clocking block

Advance clock cycles via clocking block

```
...
clocking cb @(posedge clock);
default input #1ns output #1ns;
  output reset_n;
  output din;
  output frame_n;
  output valid_n;
 ...
endclocking:  cb
modport TB (
clocking cb, output reset_n);
endinterface: router_io
```

Complete Top Level Harness

Instantiate test program and interface in harness file

Legacy DUT (wires)

Instantiate interface

Connect SystemClock to interface block

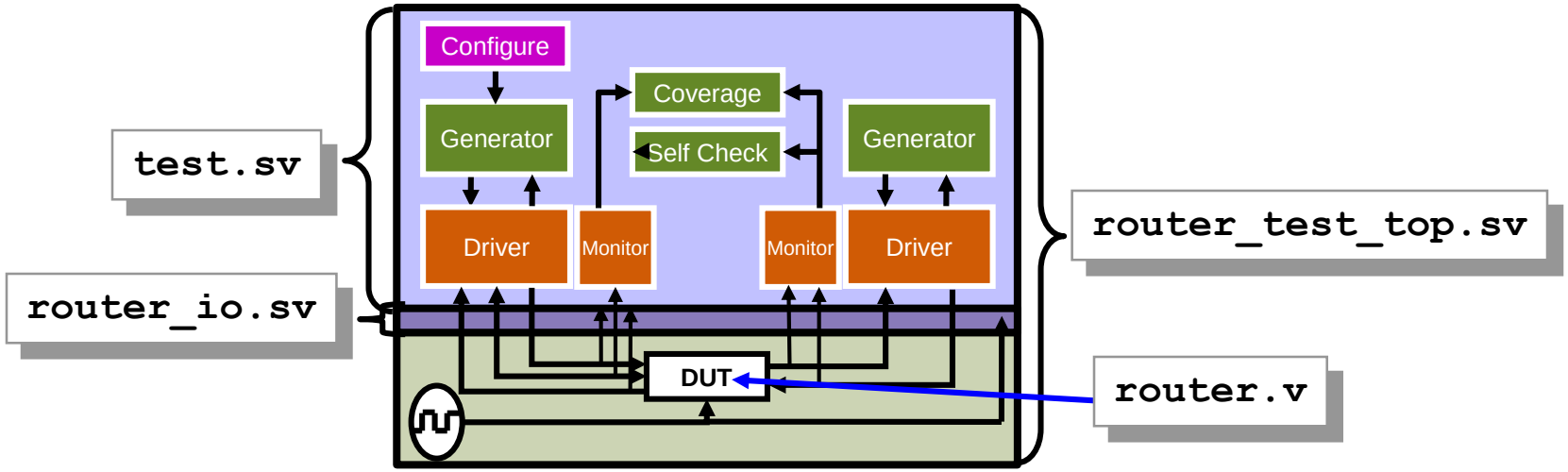
Instantiate test program

DUT instantiation using interface connection

```
module router(
     reset_n,
     clock,
     frame_n,
     valid_n,
     din,
     dout,
     busy_n,
     valido_n,
     frameo_n);
…
endmodule: router
```

```
module router_test_top;
parameter simulation_cycle =
bit  SystemClock;
router_io top_io(SystemClock
test router_test(top_io);
router dut(.reset_n (top_io.reset_n),
           .clock   (top_io.clock),
           .frame_n (top_io.frame_n),
           .valid_n (top_io.valid_n),
           .din     (top_io.din),
           .dout    (top_io.dout),
           .busy_n  (top_io.busy_n),
           .valido_n(top_io.valido_n),
           .frameo_n(top_io.frameo_n)
            );
…
endmodule:  router_test_top
```

Compile RTL & Simulate with VCS



- **Compile HDL code: (generate `simv` simulation binary)**

```
> vcs -sverilog [-debug] router_test_top.sv \
  test.sv router_io.sv router.v
```

- **Simulate DUT with SystemVerilog testbench**

```
> ./simv
```

SystemVerilog Run-Time Options

- **Pass values from simulation command line with +*arg***
- **Retrieve +*arg* value with `$value$plusargs()`**

```
initial begin: proc_user_args
  int user_seed;
  if ($value$plusargs("ntb_random_seed=%d", user_seed))
    $display("User seed is %d", user_seed);
  else
    $display("Using default seed");
end: proc_user_args
```

```
> ./simv +ntb_random_seed=100
```

User seed is 100

- **Create your own +*arg* user options for simulation control and debug**

Getting Help with VCS

- **Get vcs compiler switch summary:**

  > `vcs -help`

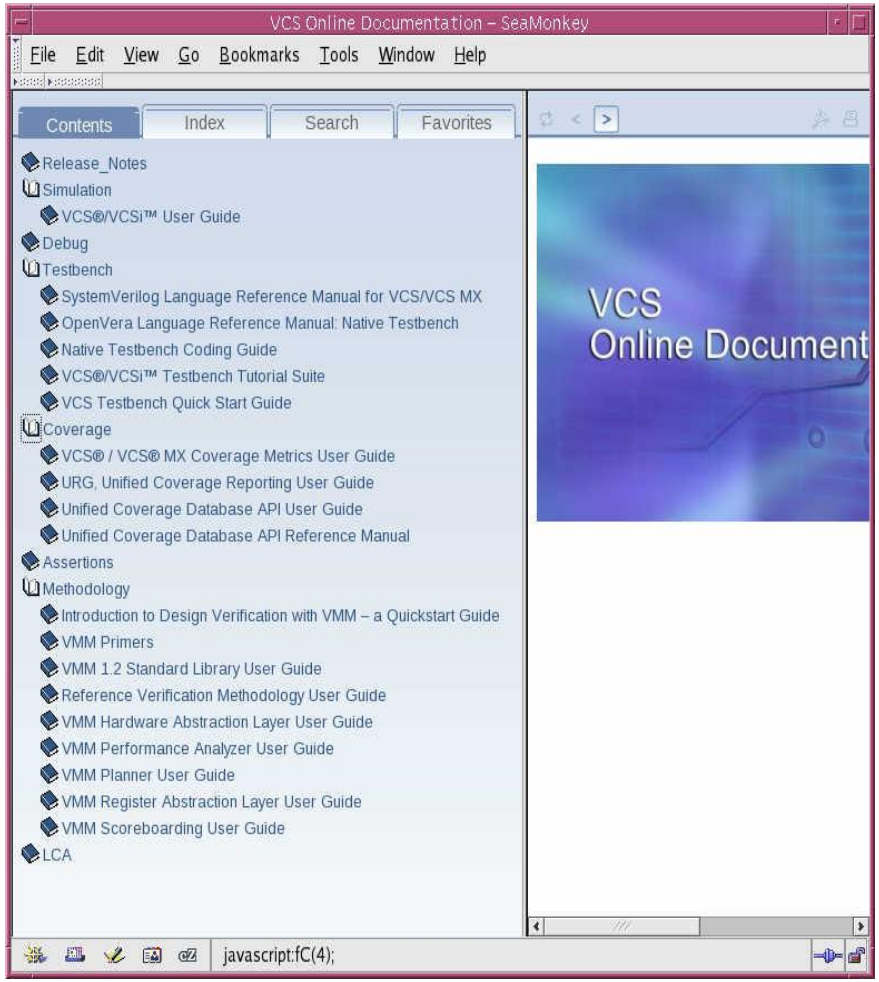
- **Read vcs manuals:**

  > `vcs -doc`

  - HTML hyperlinked docs
  - PDF documents available at:

  $VCS_HOME/doc/UserGuide/pdf/

- **Examples**
  - $VCS_HOME/doc/examples
- **Email Support:**
  - vcs_support@synopsys.com
- **On-line knowledge/forums**
  - http://solvnet.synopsys.com
  - http://verificationguild.com



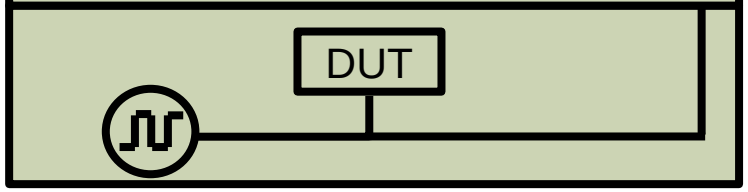
- **SystemVerilog LRM**
  - http://ieeexplore.ieee.org/Xplore/guesthome.jsp

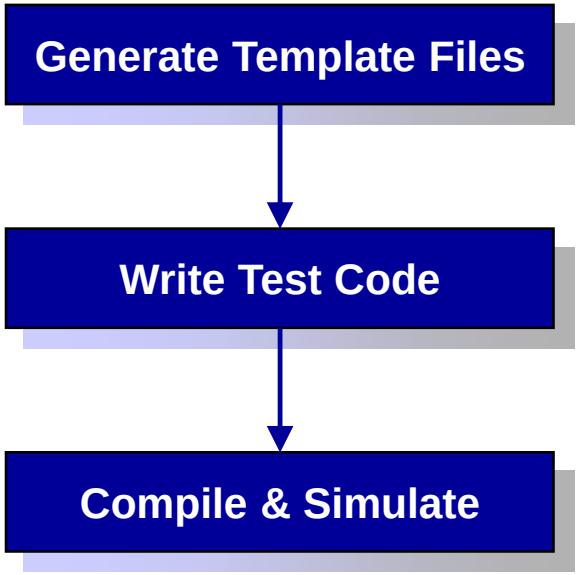
Lab 1 Introduction

30 min

Build Simulation Environment

```
program automatic test(...);
  initial begin
    $vcdpluson;
    $display("Hello World!");
    reset();
  end
  task reset();
    ...
  endtask
endprogram
```





Unit Objectives Review

Having completed this unit, you should be able to:

- **Describe the process of reaching verification goals**
- **Describe components of a SystemVerilog testbench environment**
- **Describe `program` and `interface` constructs**
- **Compile and simulate a SV testbench**
- **Drive and sample DUT signals**
- **Synchronize to known point in simulation**

Appendix

Cycle Delay, Default Clocking, Synchronous Drive

Useful VCS compile and run time switches

External binding of components

Debugging with DVE & Testbench Debugger

Cycle Delay, Default Clocking, Synchronous Drive

Cycle Delay and Default Clocking

- **Cycle Delay**
  - ## operator to indicate number of cycles to delay
- **One clocking block can be specified as the default within a given module, interface or program**

```
default clocking rtr_io.cb;
```

  - Not Recommended
    - Potential for errors when testbench has multiple clocks
    - Delay not obvious when debugging
  - Used by the cycle delay operator when used
    - Standalone

```
##4;  //wait 4 cycles using default clocking
##(k + 2);  //wait k+2 cycles of default clocking
```

    - In a synchronous drive (next slide)

Cycle Delay and Synchronous Drive

```
##num1 interface.cb.signal <= <value | expression>;
```

- **`##num1` specifies default clocking cycle delays**
  - A **`default clocking`** must be defined in the enclosing module, interface or program
  - Does not use the *interface.cb* clock
  - execution of statement is blocked (delayed)

```
interface.cb.signal <= ##num2 <value | expression>;
```

- **`##num2` specifies cycle delays of the `interface.cb` clocking block for the synchronous drive**
  - execution of statement is not blocked

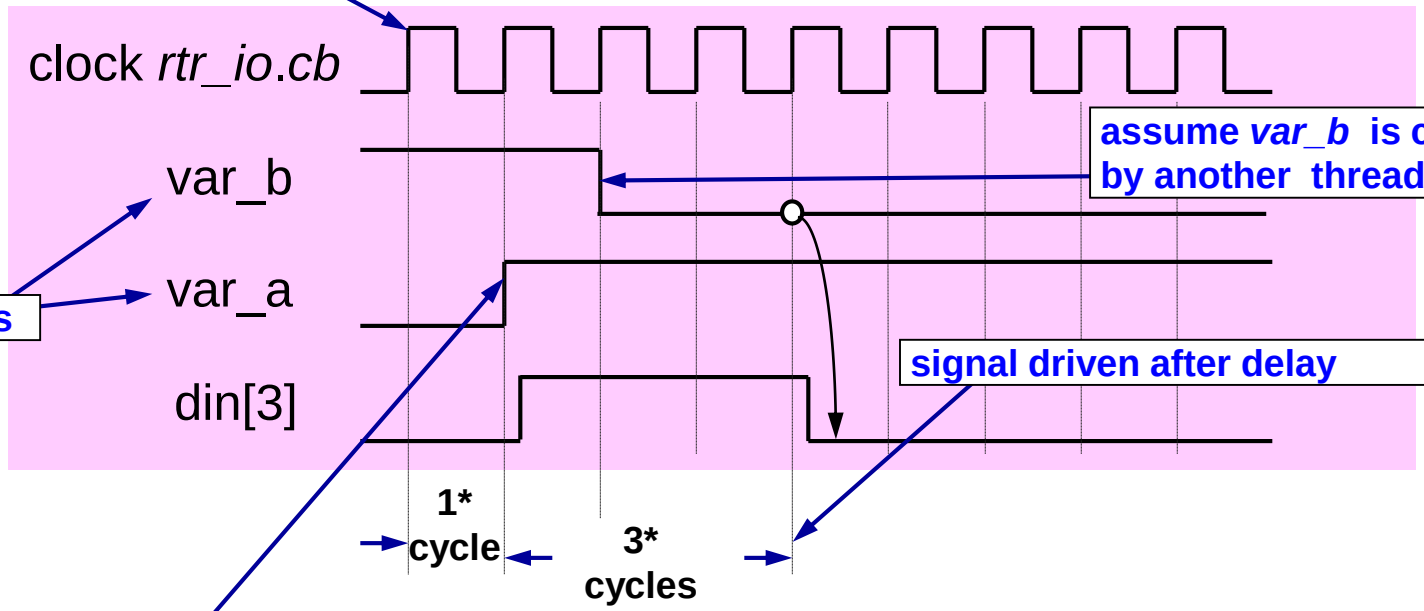
Cycle Delay Example

Example:

```
default clocking rtr_io.cb;
initial begin
…
##1; rtr_io.cb.din[3] <= 1'b1;
     var_a = var_b;
##3 rtr_io.cb.din[3] <= var_b;
…
end
```

These cycle delays use default clocking

Current clock edge



assume *var_b* is changed by another thread

variables

signal driven after delay

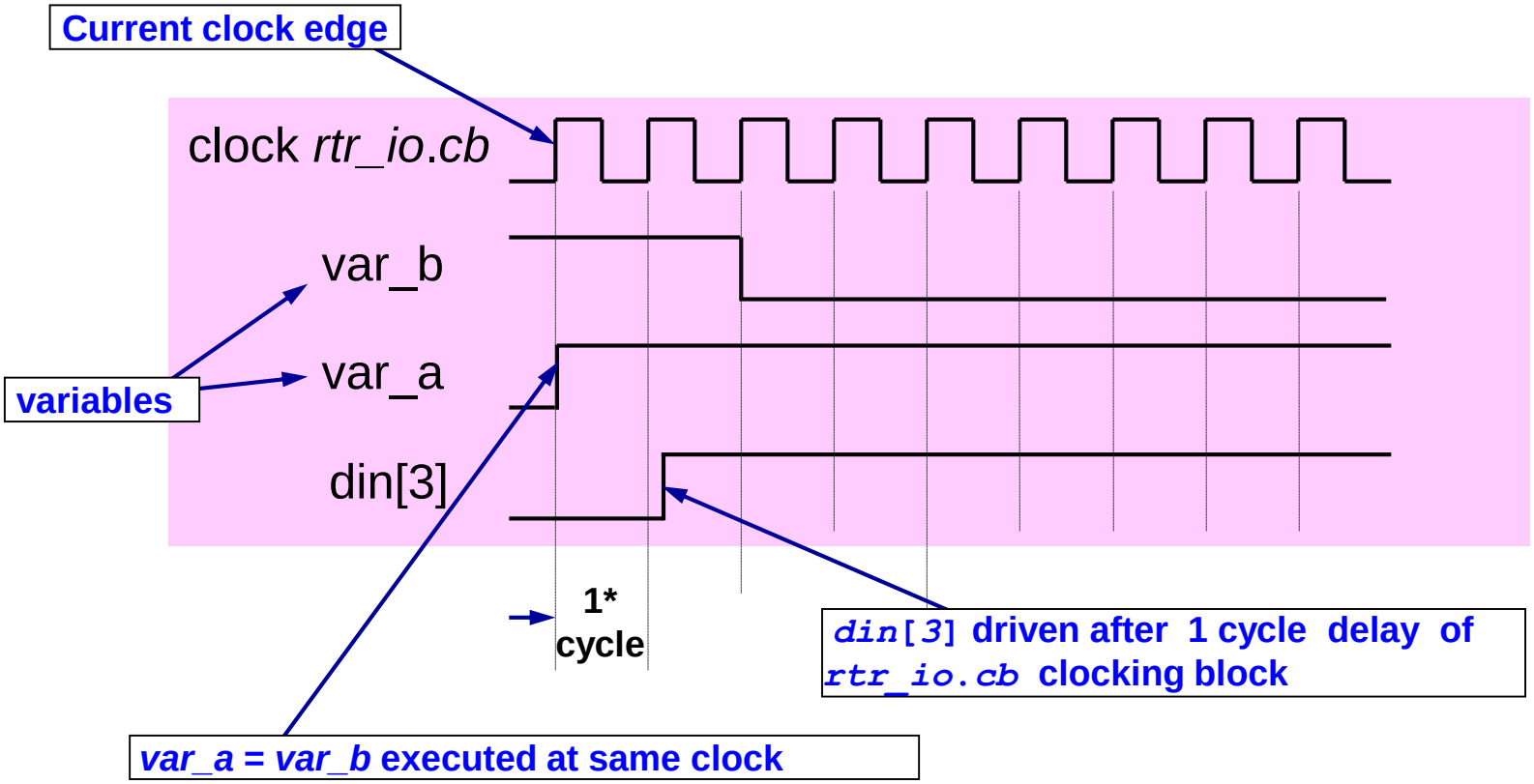
var_a = var_b executed after one clock cycle

* default clocking

Synchronous Drive Example

Example:

```
rtr_io.cb.din[3] <= ##1 1'b1;
var_a = var_b;
```



Useful VCS compile and run time switches

Compiling and Running with VCS

- **Compile:**

  `vcs -sverilog -debug` ***`top.sv test.sv dut.sv`***

  - **`-sverilog`** Enable SystemVerilog constructs
  - **`-debug`** Enable debug except line stepping
  - **`-debug_all`** Enable debug including line stepping
  - **`-lca`** Enable LCA features (see release note)

- **Run:**

  `simv` *`+user_tb_runtime_options`*

  - **`-l`** *`logfile`* Create log file
  - **`-gui`** Run GUI
  - **`-ucli`** Run with new command line debugger
  - **`-i`** *`cmd.key`* Execute UCLI commands

See the VCS User Guide for all options

Compiling with VCS - Legacy Code (1/2)

- SystemVerilog has dozens of new reserved keywords such as `bit`, `packed`, `logic` that might conflict with existing Verilog code
- Keep your Verilog-2001 code separate from SystemVerilog code and compile with:

```
vcs -sverilog new.v +verilog2001ext+.v2k old.v2k
```

- *or*

```
vcs +systemverilogext+.sv old.v new.sv
```

```
// Old Verilog-1995/2001 legacy code
integer bit, count;
initial begin
  count = 0;
  for (bit = 0; bit < 8; bit = bit + 1)
    if (adrs[bit] === 1'bx)
      count = count + 1;
end
```

Compiling with VCS – Legacy Code (2/2)

- **New keywords in new LRM revisions cause identifiers in legacy code to be invalid**
  - VCS 2014.12 provides command line control with new switch
    - **`-sv=<version>`**, where version is 2005, 2009, 2012 etc.
    - Use in Two-step flow e.g.
      - `% vcs -sv=2009 <all files>`
    - Use in Three-step flow e.g.
      - `% vlogan -sv=2005 <some files>`
        `% vlogan -sv=2009 <other files>`
        `% vcs -q <top_module>`
  - **`-sv`** option enables SystemVerilog mode
    - same as **`-sverilog`** option
    - Specifying both **`-sv`** and **`-sverilog`** together will be an error
    - Controls only keyword set, no modification of behavior

External binding of components

External Binding of Components

- Components can be instantiated using `bind`

```
module router_test_top;
  parameter simulation_cycle = 100;
  reg  SystemClock;
  router_io top_io(SystemClock);
  test t(top_io);
  router dut( //device under test
      .reset_n  (top_io.reset_n),
      ...
      .frameo_n (top_io.frameo_n)
    );
  ...
endmodule
```

instance replaced by bind

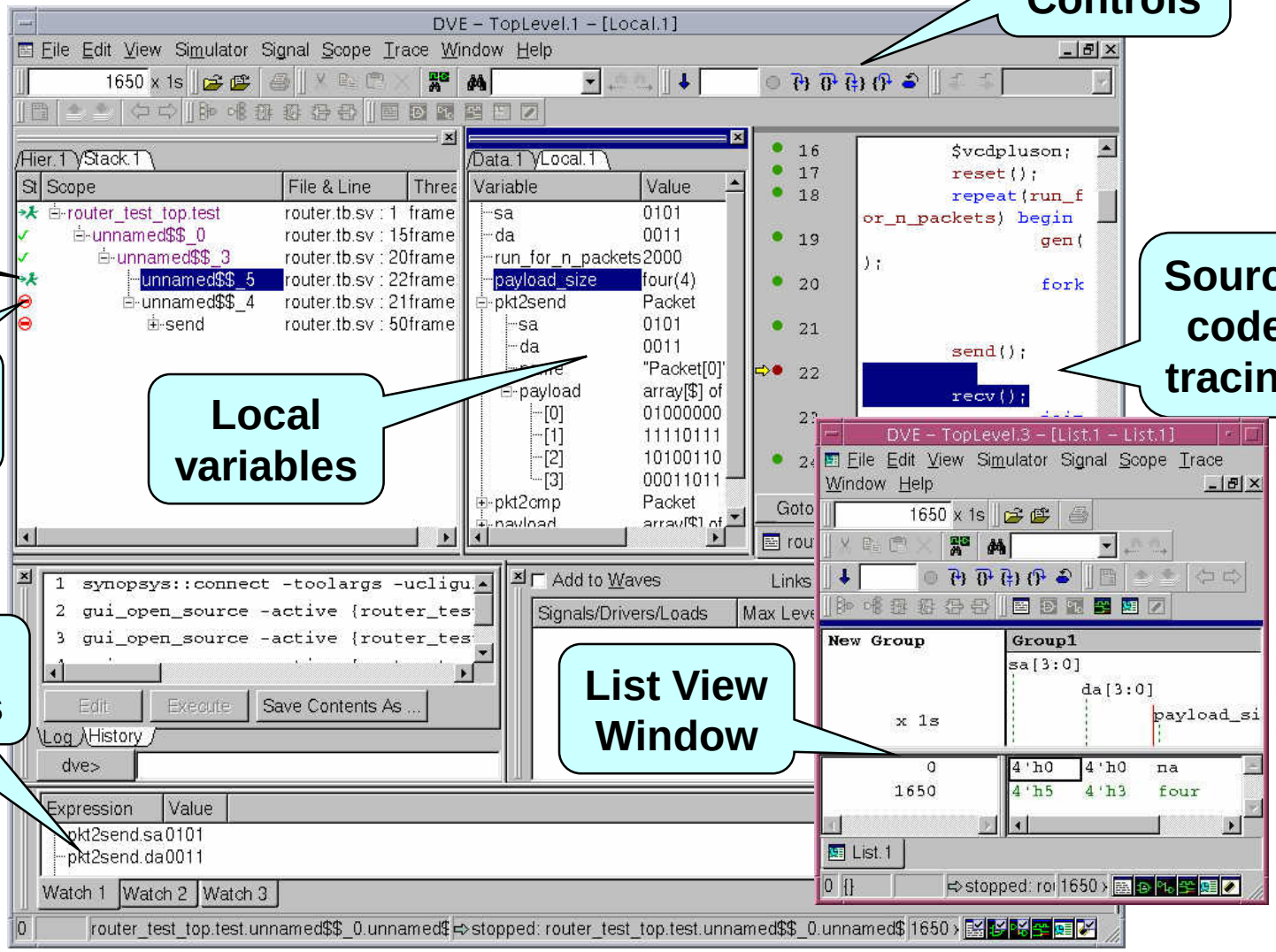
XMR to top module interface instance

```
bind router_test_top test t(router_test_top.top_io);
```

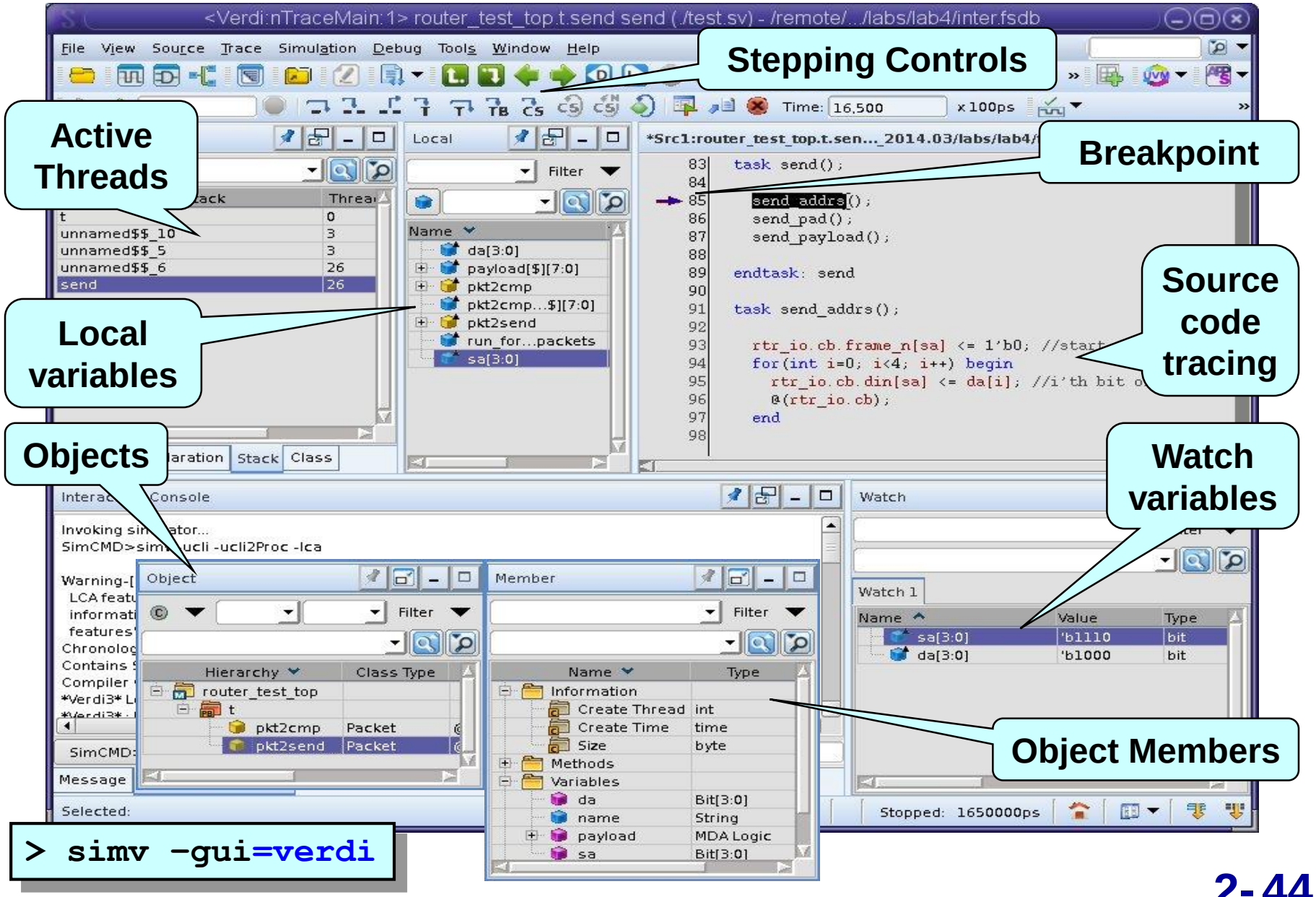
Debugging with DVE & Testbench Debugger

DVE Testbench Debug: Getting Started

```
> simv -gui=dve
```



Verdi Testbench Debug: Getting Started




```
> simv -gui=verdi
```