DOULOS KnowHow WEBINARS

Delivering KnowHow www.doulos.com

Improving your VHDL FPGA verification using OSVMM and UVVM

Verification Futures 2025

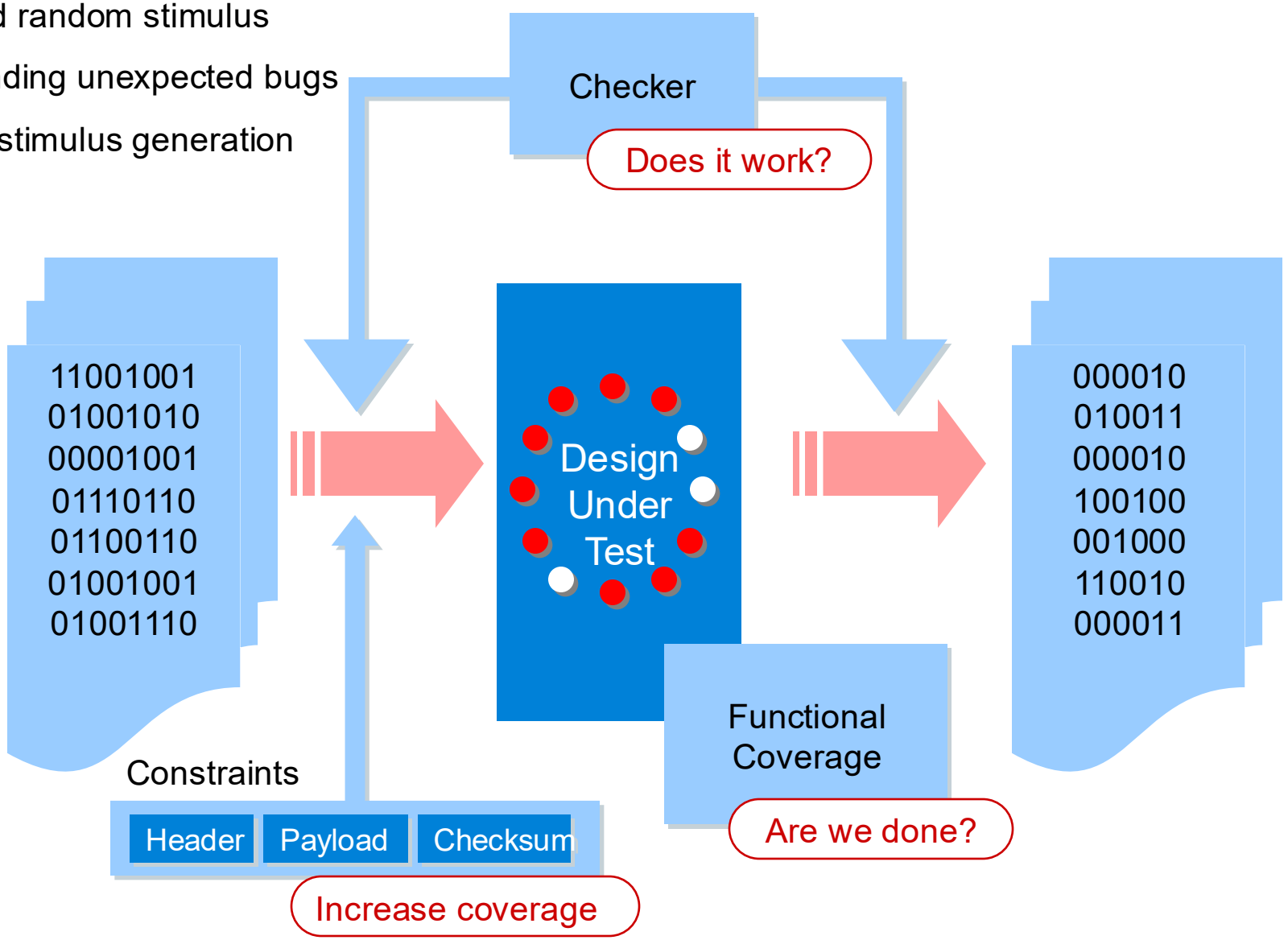
Presenter: Matt Bridle
Certified Training Instructor

Constrained Random Verification

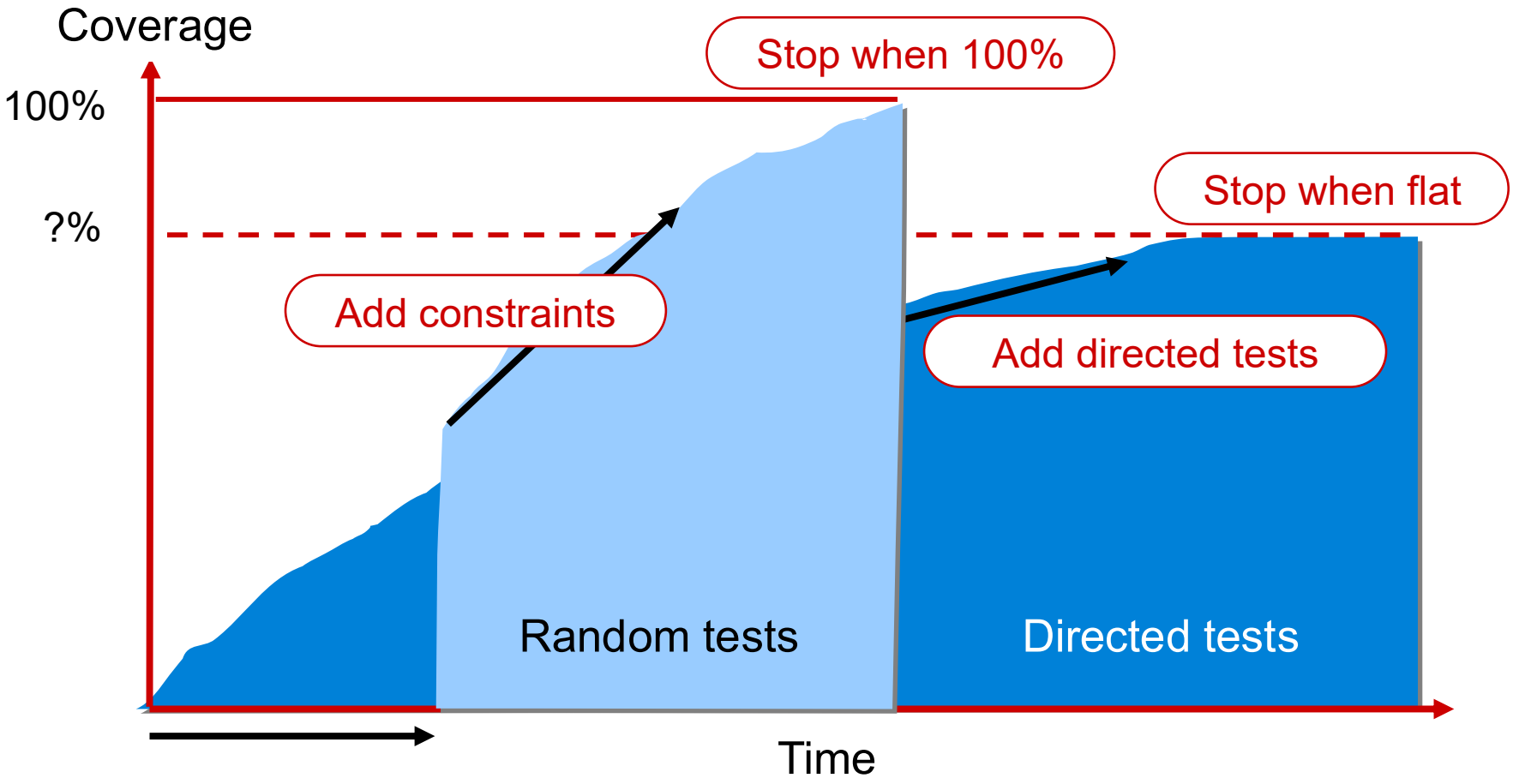
Constrained random stimulus

Good for finding unexpected bugs

Automates stimulus generation



Coverage Driven Verification



Define coverage goals and minimal constraints

What is OSVVM?

OSVVM, the Open Source VHDL Verification Methodology

- Developed by SynthWorks Design Inc and Aldec Inc
- Free and Open Source

- VHDL verification framework
- Verification utility library
  - Randomization, functional coverage, error logging etc
- Verification component library
  - AXI4, Ethernet, SPI and Wishbone etc
- Scripting API
- Co-simulation capability

What is UVVM?

UVVM, the Universal VHDL Verification Methodology

- Developed by EmLogic AS (Norway)
  - Released under MIT License
- Two parts:
  - VVC Framework
  - Utility Library

OSVMM Verification Utility Library

- Collection of useful testbench utility functions and procedures

Category	Example
Constrained Random testing	`i := RV.RandInt(0, 255)`
Functional Coverage	`ICover(cp_mode, mode);`
Error logging and reporting	`Log("OSVVM message 3", DEBUG);`
Useful testbench utilities	`WaitForBarrier(OsvvmTestDone)`
Clock and Reset control	`CreateClock(Clk, Tperiod_Clk)`
Common transcript file	`TranscriptOpen("osvvm.log")`
Resolved types	`AddressBusRecType`

and more.

- VHDL context:

```
library osvvm ;
context osvvm.OsvvmContext ;
```

UVVM Utility Library

- Collection of useful testbench utility functions and procedures

Category	Example
Logging and Verbosity Control	`disable_log_msg(msg_id, …)`
Alert Handling	`set_alert_file_name(…)`
Checks and Awaits	`await_change(…)`
String Handling	`to_string(…)`
Randomization	`randomize(…)`
Signal Generators	`gen_pulse(…)`
Synchronization	`await_barrier(…)`
BFM Common Package	`wait_num_rising_edge(…)`

and more.

- VHDL context:

```
library uvvm_util;
  context uvvm_util.uvvm_util_context;
```

Randomization

OSVVM

```
variable RV: RandomPType;
…
RV.InitSeed(RV'instance_name);
i := RV.RandInt(0, 255);
i := RV.RandInt(0, 255, Exclude => (1, 4, 16, 64));
v := RV.RandSlv( (1, 4, 16, 64), Size => 8);
s := RV.RandSigned(-128, 127, Size => 8);
i := RV.DistValInt( ((1, 1), (2, 10), (3, 100)) );
```

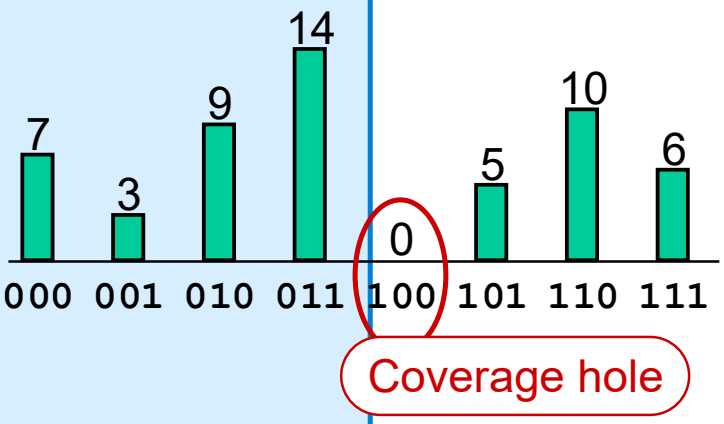
UVVM

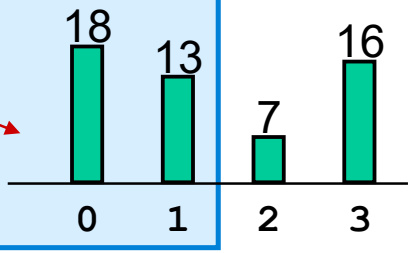
```
variable RV : t_rand;
…
RV.InitSeed(RV'instance_name);
i := RV.rand(0, 255);
i := RV.rand(0, 255, ADD, (511), EXCL, (1, 4, 16, 64));
v := RV.rand(8, ONLY, (1, 4, 16, 64));
s := RV.rand(8, -128, 127);
i := RV.rand_range_weight( ((0,1,1),(2,2,10),(3,5,100)) );
```

Random seeds can also be saved and restored

OSVVM Functional Coverage

```
architecture
 signal op_code : std_logic_vector(2 downto 0);
 signal mode    : std_logic_vector(1 downto 0);
 ...
 signal cp_opcode: CoverageIDType;
 signal cp_mode  : CoverageIDType;
begin
  process
  begin
    cp_opcode <= NewID(" cp_opcode ");
    cp_mode <= NewID(" cp_mode ");
    wait for 0 ns ;
    AddBins(cp_opcode, "opcode", GenBin(0,7));
    AddBins(cp_mode, "mode",  GenBin(0,3));
    ...
    wait;
  end process;
```

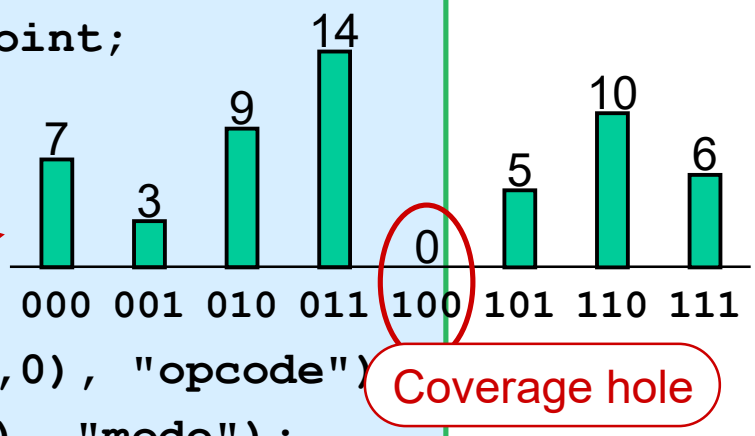


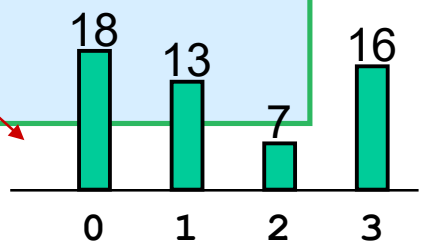


UVVM Functional Coverage

```
architecture
 signal op_code : std_logic_vector(2 downto 0);
 signal mode    : std_logic_vector(1 downto 0);
 ...
 shared variable cp_opcode: t_coverpoint;
 shared variable cp_mode  : t_coverpoint;

begin
  p_main : process
  begin
    cp_opcode.add_bins(bin_range(0,7,0), "opcode");
    cp_mode.add_bins(bin_range(0,3,0), "mode");
    ...
    wait;
  end process;
```





Sampling

OSVVM

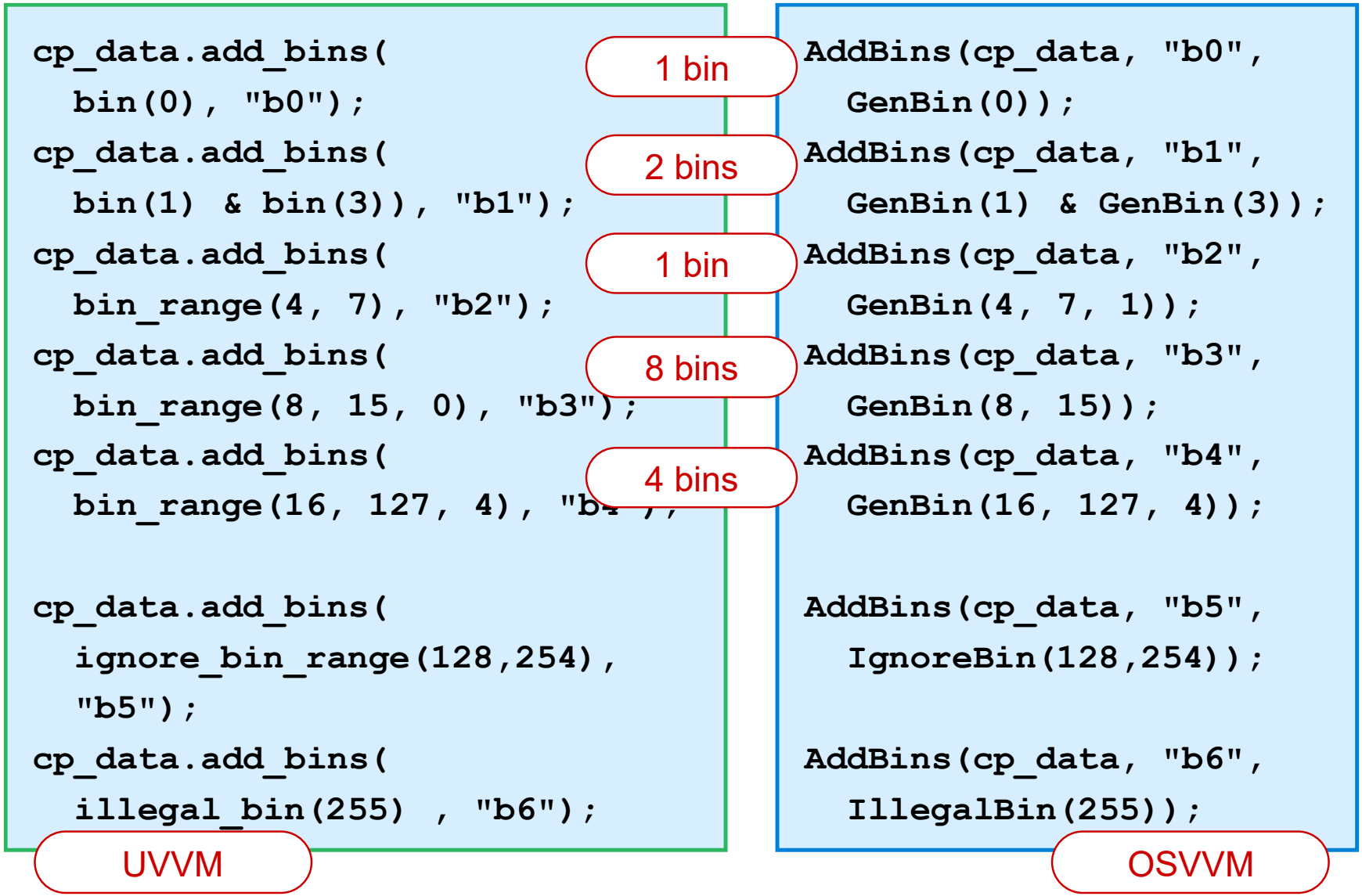
```
process (clock)
  begin
    if rising_edge(clock) then
      ICover(cp_opcode, to_integer(unsigned(op_code)));
      ICover(cp_mode, to_integer(unsigned(mode)));
      ICover(cp_enable, to_integer(unsigned'(0 => enable)));
      …
```

UVVM

```
process (clock)
  begin
    if rising_edge(clock) then
      cp_opcode.sample_coverage(to_integer(unsigned(opcode)));
      cp_mode.sample_coverage(to_integer(unsigned(mode)));
      cp_opcode.sample_coverage(to_integer(unsigned'(0=>enable)));
      …
```

- 'X', 'Z' etc. treated as zero
- Integer types expected – must convert vectors to integers first
- Sampling a single bit requires creating a 1-bit vector, assigned by name

Specifying Bins



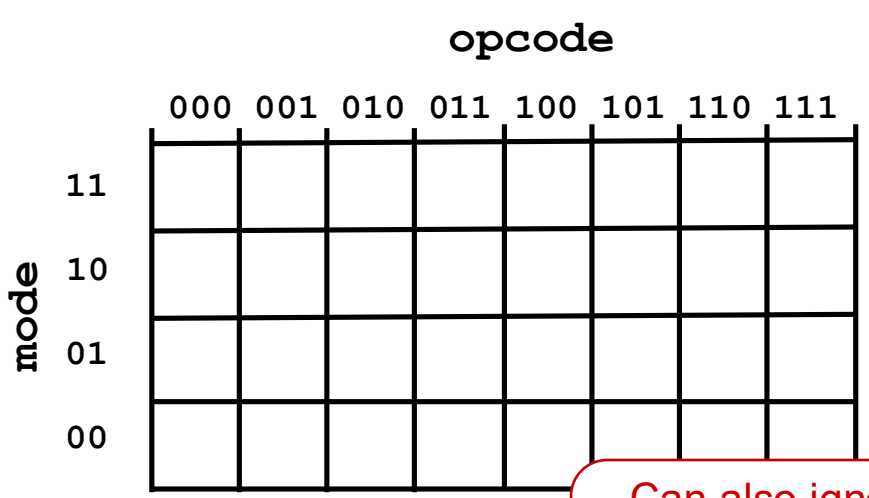
Specifying Cross Bins

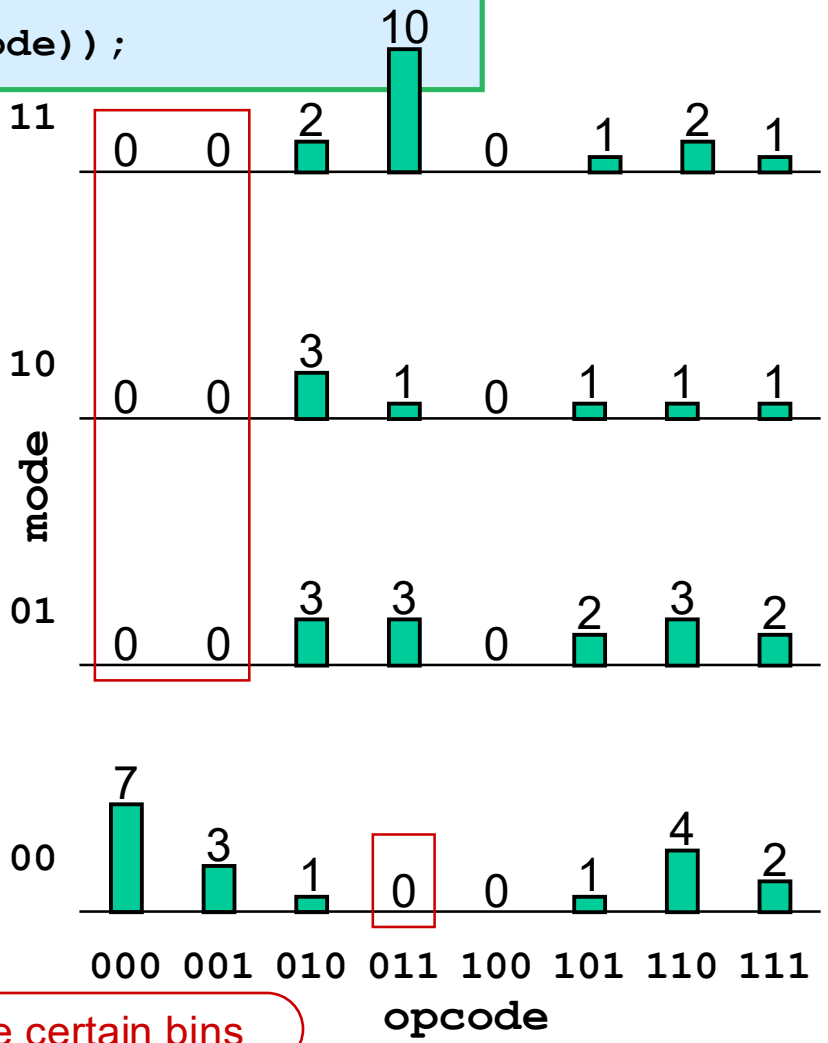
```
cp_cross.add_cross(bin_range(0, 7), bin_range(0, 3));
…
cp_cross.sample_coverage((opcode, mode));
```

UVVM

OSVVM

```
AddCross(cp_cross, GenBin(0,7),
                   GenBin(0, 3));
…
Icover(cp_cross, (opcode, mode));
```





Can also ignore certain bins

OSVVM - Reporting Coverage

```
WriteBin(cp_mode);
WriteCovHoles(cpmode);
```

OSVVM

UVVM

```
cp_mode.report_coverage(NON_VERBOSE);
cp_mode.report_coverage(HOLES_ONLY);
```

OSVVM Intelligent Coverage

```
architecture
  signal cov: CoverageIDType;
begin
  process
    variable  v: integer;
  begin
    cov <= NewID(" cov ") ;
    wait for 0 ns ;
    cov.AddBins(GenBin(0, 255));
    ...
    loop
      wait until rising_edge(clock);
      v := GetRandPoint(cov);
      report to_string( GetItemCount(cov) ) & " " & to_string(v);
      ...
      ICover(cov, v);
      exit when IsCovered(cov);
    end loop;
```

256 bins

Random value in range 0 to 255 not already covered

"Intelligent Coverage"

UVVM Optimized Randomization

```
architecture
  shared variable cov: t_coverpoint;
begin
  process
    variable  v: integer;
  begin
    cov.add_bins(bin_range(0, 255, 0));
    ...
    loop
      wait until rising_edge(clock);
      v := cov.rand(NO_SAMPLE_COV);
      ...
      cov.sample_coverage(v);
      exit when cov.coverage_completed(BINS_AND_HITS);
    end loop;
```

256 bins

Don't let this affect overall coverage

Random value in range 0 to 255 not already covered

Weight by Coverage Shortfall

```
AddBins(ID=>cov, CovBin=>GenBin( 0, 3), AtLeast=>8);
AddBins(ID=>cov, CovBin=>GenBin( 4, 7), AtLeast=>4);
AddBins(ID=>cov, CovBin=>GenBin( 8,11), AtLeast=>2);
AddBins(ID=>cov, CovBin=>GenBin(12,15), AtLeast=>1);
SetWeightMode(cov, REMAIN);
```

OSVVM

UVVM

```
cov.add_bins(bin_range( 0,  3, 0), min_hits =>8);
cov.add_bins(bin_range( 4,  7, 0), min_hits =>4);
cov.add_bins(bin_range( 8, 11, 0), min_hits =>2);
cov.add_bins(bin_range(12, 15, 0), min_hits =>1);
```

No explicit rand_weight argument

Can alternatively define explicit weights for randomization

OSVVM Logging

```
process
begin
  Log("OSVVM message 1");
  Log("OSVVM message 2", ALWAYS);
  Log("OSVVM message 3", DEBUG);

  wait until rising_edge(clock);
  SetLogEnable(DEBUG, TRUE);
  SetLogEnable(INFO, TRUE);

  Log("OSVVM message 4", DEBUG);
  Log("OSVVM message 5", INFO);
```

ALWAYS, DEBUG, FINAL, INFO, PASSED

ALWAYS is enabled by default

```
%% Log    ALWAYS   OSVVM Message 1 at 0 ns
%% Log    ALWAYS   OSVVM Message 2 at 0 ns
%% Log    DEBUG    OSVVM Message 4 at 5 ns
%% Log    INFO     OSVVM Message 5 at 5 ns
```

UVVM Log Messages

- output a message:

```
log(ID_LOG_HDR, "Starting simulation", C_SCOPE);
```

```
UVVM: ID_LOG_HDR                    3000.0 ns  TB seq.              Starting simulation
UVVM: ------------------------------------------------------------------------------------
```

- output a text block:

```
write(L, string'("The first sentence of the block. "));
write(L, string'("The second sentence of the block."));
log_text_block(ID_SEQUENCER, L, FORMATTED, "Header", C_SCOPE);
```

```
UVVM: ************************************************************************************
UVVM:
UVVM:
UVVM: ID_SEQUENCER                  3000.0 ns  TB seq.              Header
UVVM: ------------------------------------------------------------------------------------
UVVM: The first sentence of the block. The second sentence of the block.
UVVM: ************************************************************************************
```

OSVVM - Redirecting to a Log File

```
process
begin
  TranscriptOpen("osvvm.log");

  Log("OSVVM message 6", ALWAYS);

  SetTranscriptMirror;

  wait until rising_edge(clock);
  Log("OSVVM message 7");
```

Redirect log to file

Duplicate log in std output and file

File

```
%% Log    ALWAYS   OSVVM Message 6 at 0 ns
%% Log    ALWAYS   OSVVM Message 7 at 5 ns
```

Terminal

```
%% Log    ALWAYS   OSVVM Message 7 at 5 ns
```

UVVM - Redirecting Log Messages

- set the log file name:

```
set_log_file_name("new_log_file_name.txt");
```

- set the default destination for all log messages:

```
set_log_destination(CONSOLE_ONLY);
```

```
CONSOLE_AND_LOG
CONSOLE_ONLY
LOG_ONLY
```

- report which IDs are enabled or disabled:

```
report_msg_id_panel(VOID);
```

```
UVVM: ------------------------------------------------------------------------------
UVVM:      ***  REPORT OF MSG ID PANEL ***
UVVM: ------------------------------------------------------------------------------
UVVM:                ID                              Status
UVVM:                ------------------------        ------
UVVM:                ...
UVVM:                ID_LOG_HDR                   : ENABLED
UVVM:                ...
UVVM:                ID_BFM                       : DISABLED
UVVM:                ...
UVVM: ------------------------------------------------------------------------------
```

UVVM Log Message Control

- Enable a message with a specific ID in this scope:

```
enable_log_msg(ID_BFM);
```

- Disable all messages in this scope :

```
disable_log_msg(ALL_MESSAGES);
```

- Disable a message with a specific ID in this scope:

```
disable_log_msg(ID_BFM);
```

- Enable a message with a specific ID in a VVC:

```
enable_log_msg(COUNTER_VVCT, 1, ID_BFM);
```

- Disable all messages in a VVC:

```
disable_log_msg(COUNTER_VVCT, 1, ALL_MESSAGES);
```

Alerts

- Like fancy log messages
  - simulation can stop after a certain number of alerts
  - many severity levels
  - filtering by severity level
  - messages can be redirected to a file

```
Alert("Message 1", WARNING);
Alert("Message 2", ERROR);
Alert("Message 3", FAILURE);
```

OSVVM

UVVM

```
alert(TB_WARNING, "This is a TB_WARNING.");
warning( "This is a WARNING.");
tb_warning("This is another TB_WARNING.");
alert(NOTE, "This is a NOTE from the BFM.", "BFM");
```

OSVVM Stop Count

```
process
begin
  SetAlertStopCount(WARNING, 10);
  SetAlertStopCount(ERROR, 2);

  SetAlertStopCount(FAILURE, 2);

  Alert("Message 1", WARNING);
  Alert("Message 2", ERROR);
  Alert("Message 3", FAILURE);
  Alert("Message 4", ERROR);
```

default stop count is integer'right

default stop count is 0

```
%% Alert WARNING Message 1 at 0 ns
%% Alert ERROR   Message 2 at 0 ns
%% Alert FAILURE Message 3 at 0 ns
%% Alert ERROR   Message 4 at 0 ns

%% Alert Stop Count on ERROR   reached at 0 ns
%% DONE  FAILED  AlertLogTop  Total Error(s) = 4
Failures: 1  Errors: 2  Warnings: 1  at 0 ns
```

OSVVM Alert Control

```
process
begin
  SetAlertEnable(WARNING, FALSE);
  SetAlertEnable(ERROR,   FALSE);
  SetAlertEnable(FAILURE, FALSE);

  Alert("Message 1", WARNING);
  Alert("Message 2", ERROR);
  Alert("Message 3", FAILURE);

  SetGlobalAlertEnable(TRUE);

  Alert("Message 4", WARNING);
  Alert("Message 5", ERROR);
  Alert("Message 6", FAILURE);
```

```
%% Alert WARNING Message 4 at 0 ns
%% Alert ERROR   Message 5 at 0 ns
%% Alert FAILURE Message 6 at 0 ns
```

UVVM Alert Control

- Set stop limit:

```
set_alert_stop_limit(TB_WARNING, 100);
```

```
UVVM: Simulator has been paused as requested after 100 TB_WARNING
UVVM: *** To find the root cause of this alert, step out the HDL
calling stack in your simulator. ***
UVVM: *** For example, step out until you reach the call from the
test sequencer. ***
```

```
set_alert_file_name("new_alert_log_file.txt");
```

```
set_alert_attention(NOTE, IGNORE);
```

More Alerts and Checks

```
AlertIf(Condition, "Condition occurred!", ERROR);
AlertIfNot(Condition, "Condition did not occur!", ERROR);
AlertIfEqual(2, 2, "Some things are equal!", ERROR);
AlertIfNotEqual(2, 3, "Some things not are equal!", ERROR);
```

OSVVM

UVVM

```
check_value(the_answer, 42, WARNING, "what's the question?");
check_value_in_range(some_integer, 10, 100, ERROR, "Out of range");
check_stable(slv8, 9 ns, ERROR, "Checking if SLV is stable");
```

UVVM Message IDs

- There are various built-in IDs:

```
ID_FINISH_OR_STOP -- Terminating the complete simulation

ID_CLOCK_GEN,     -- Logging when clock generators are en/disabled

ID_LOG_HDR,       -- Log section headers
ID_LOG_HDR_LARGE, -- Large log section headers
ID_LOG_HDR_XL,    -- Extra large log section headers
ID_SEQUENCER,     -- Normal log (not log headers)

ID_BFM,           -- Used inside a BFM (to log BFM access)

ID_UVVM_SEND_CMD, -- shows commands being sent to queues

ALL_MESSAGES      -- Applies to ALL message ID apart from ID_NEVER
```

ONLY in test sequencer

- User message IDs can be defined by editing the **`t_msg_id`** enumerated type in the package **`adaptations_pkg`**

OSVVM Hierarchical Alerts

```
entity child is
  generic (TOP_ALERT_ID: AlertLogIDType);
  constant ALERT_ID: AlertLogIDType :=
                    GetAlertLogID(PathTail(child'PATH_NAME), TOP_ALERT_ID);
begin
  Alert(ALERT_ID, "Message", ERROR);
end entity;

entity top is
  constant TOP_ALERT_ID: AlertLogIDType :=
                   GetAlertLogID(PathTail(top'PATH_NAME), ALERTLOG_BASE_ID);
end entity;
architecture bench of top is
begin
  inst1: entity work.child generic map (TOP_ALERT_ID);
  inst2: entity work.child generic map (TOP_ALERT_ID);
  ReportAlerts;
end architecture;
```

```
AlertIf(ALERT_ID, ... );
AlertIfNotEqual(ALERT_ID, ... );
SetAlertEnable(ALERT_ID, ...);
SetAlertStopCount(ALERT_ID, ... );
Log(ALERT_ID, ... );
SetLogEnable(ALERT_ID, ...);
```

```
%% DONE  FAILED  AlertLogTop  Total Error(s) = 2  Failures: 0  Errors: 2
Warnings: 0  at 10 ns
%%    Default  Failures: 0  Errors: 0  Warnings: 0
%%    OSVVM    Failures: 0  Errors: 0  Warnings: 0
%%    Top      Failures: 0  Errors: 2  Warnings: 0
%%      inst1  Failures: 0  Errors: 1  Warnings: 0
%%      inst2  Failures: 0  Errors: 1  Warnings: 0
```

Generating signals

```
CreateClock (Clk => Clk, Period => Tperiod_Clk  );
```

OSVVM

Concurrent procedure calls

```
CreateReset (Reset => Reset, ResetActive => '1', Clk => Clk,
             Period => 7 * tperiod_Clk, tpd => tpd  ) ;
```

Concurrent procedure call

```
clock_generator(clk100M, clk100Men, 10 ns,
                "100MHz, 60% duty cycle", 6 ns);
```

UVVM

Sequential procedure call

```
gen_pulse(trig, x"AB", clk100M, 2, "2 pulses of 0xAB on trig");
```

OSVVM Barriers

- Barriers enable multiple processes to wait until they all reach a specified step:
  - waits until all processes have called `WaitForBarrier`

```
WaitForBarrier(OsvvmTestDone);
```

- Requires a signal of type BarrierType:

```
signal TestDone : BarrierType ;
```

- There are 6 predefined signals:

```
alias OsvvmResetStarted : BarrierType is Barrier.ResetStarted ;
alias OsvvmResetDone : BarrierType is Barrier.ResetDone ;
alias OsvvmTestInit : BarrierType is Barrier.TestInit ;
alias OsvvmTestDone : BarrierType is Barrier.TestDone ;
alias TestDone : BarrierType is Barrier.TestDone ;
alias OsvvmVcInit : BarrierType is Barrier.VcInit ;
```

UVVM Synchronization

- Block and unblock flags…

```
block_flag("my_flag", "blocking my flag");
unblock_flag("my_flag", "unblocking my flag", global_trigger);
```

Global signal – required for correct operation

…and then wait for them to be unblocked:

```
await_unblock_flag("my_flag", 10 us, "waiting for my_flag unblock",
  RETURN_TO_BLOCK, WARNING);
```

```
RETURN_TO_BLOCK
KEEP_UNBLOCKED
```

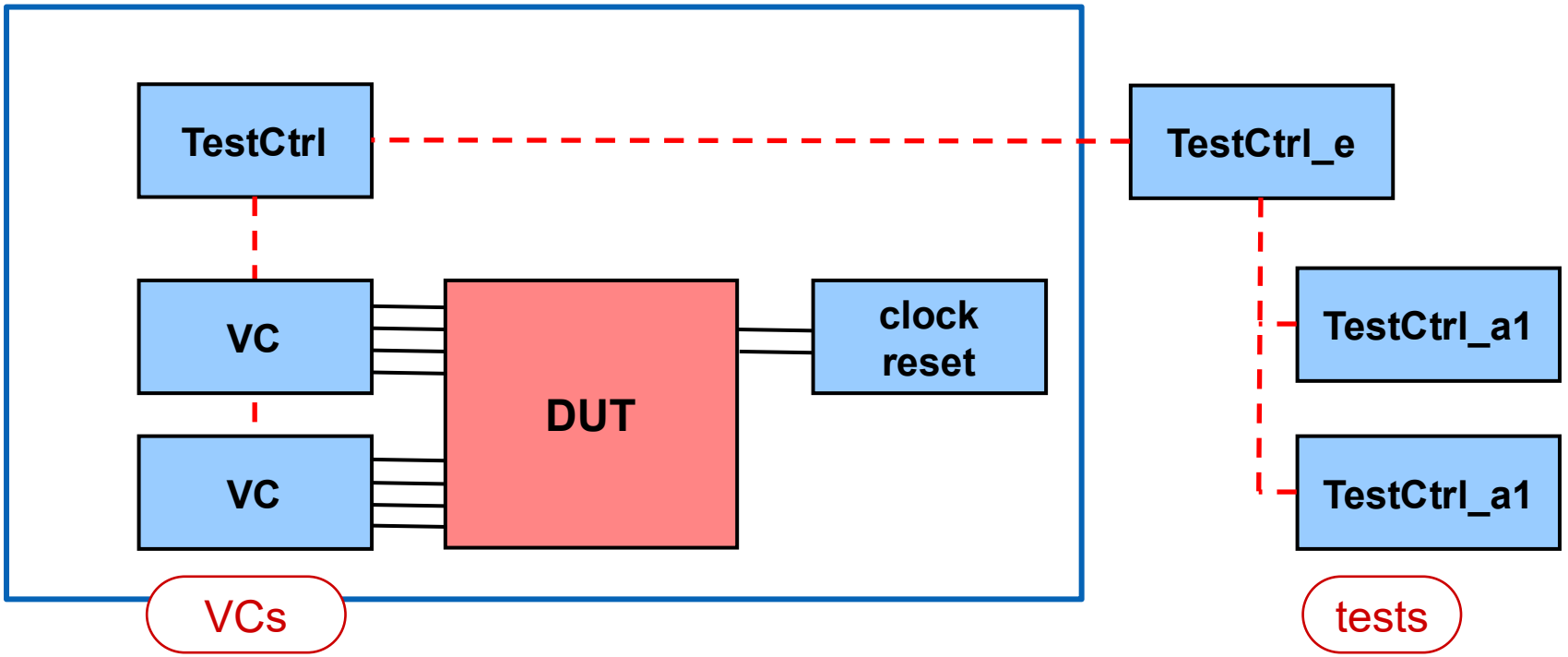
- Barriers enable multiple processes to wait until they all reach a specified step:
  - waits until all processes have called `await_barrier`

```
await_barrier(global_barrier, 100 us, "waiting for global barrier",
  ERROR);
```

Global signal – required for correct operation

OSVVM Structured Testbench Framework

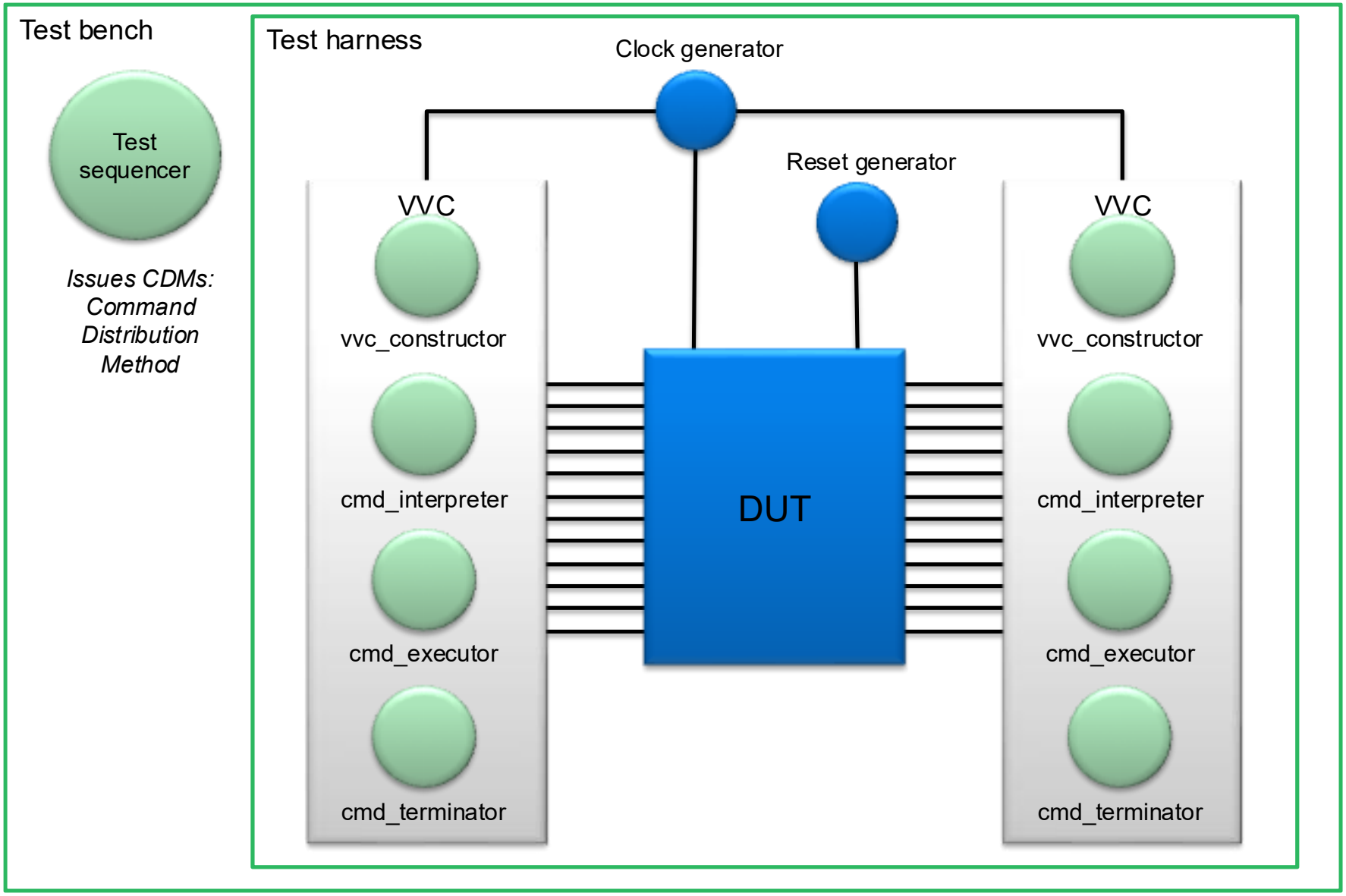
- Implements a structured testbench
- VC = Verification Component



- VHDL context:

```
library osvvm ;
context osvvm.OsvvmCommonContext ;
```

UVVM Structure



VIP

OSVVM

- AXI4
- Ethernet
- SPI
- UART
- Wishbone

UVVM

- Avalon-MM / Avalon-ST
- AXI4 / AXI4-Lite / AXI-Stream
- Clock Generator
- Error Injection
- GMII / RGMII
- GPIO
- I2C
- Scoreboard
- SPI
- UART

DOULOS

Delivering KnowHow www.doulos.com

SoC Design & Verification	» SystemVerilog » UVM » Formal » SystemC » TLM-2.0
FPGA & Hardware Design	» VHDL » Verilog » SystemVerilog » Tcl » AMD » Intel FPGA
Embedded Software & Arm	» Emb C/C++ » Emb Linux » Yocto » RTOS » Security » Android » Arm » Rust » Zephyr
Python, AI & Machine Learning	» Python » Edge AI » Deep Learning

SystemVerilog OUT OF THE BOX · UVM · SYSTEMC™ · yocto PROJECT PARTICIPANT · Zephyr · SECURE EMBEDDED · APPROVED TRAINING PARTNER arm · AI & MACHINE LEARNING