SIEMENS EDA

SystemVerilog UVM Labs Student Workbook

2023.1

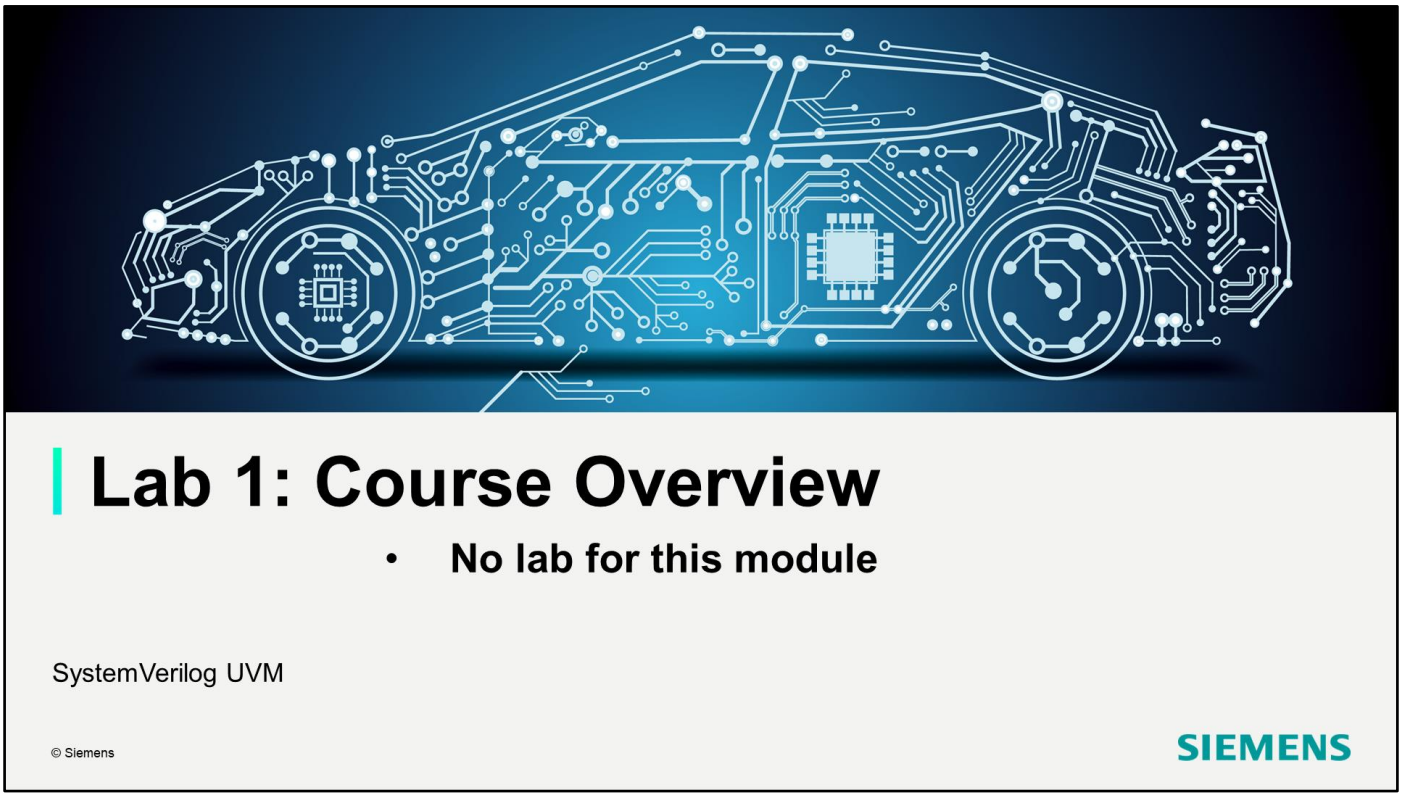
SIEMENS

Support Center: support.sw.siemens.com
Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Part Number: 074328

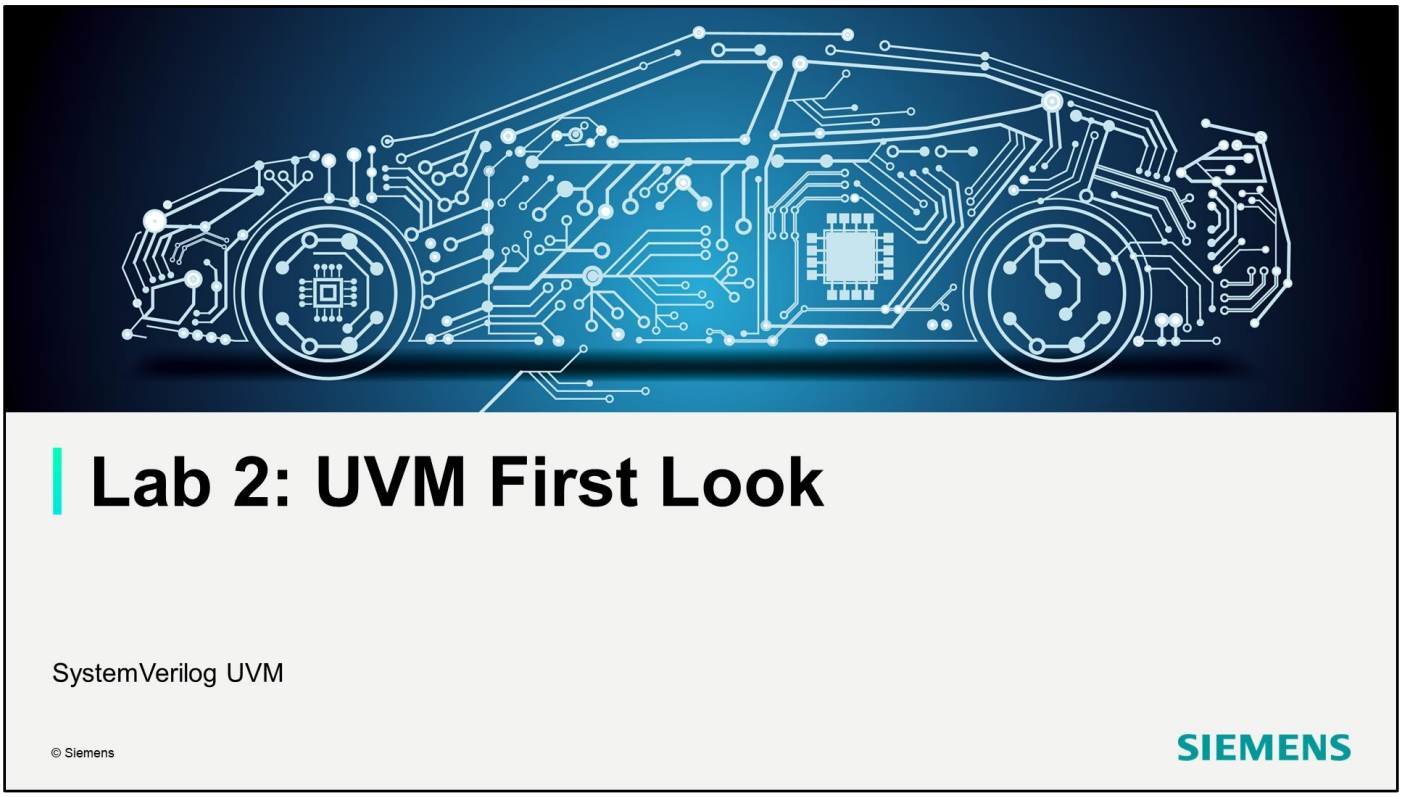
Table of Contents

Table of Contents



Notes:

Version 2023.1



Notes:

Version 2023.1

Objectives

- Learn the UVM testbench topology and file organization
- Learn how to write and filter UVM messages
- Peek at the UVM source code

SIEMENS

Notes:

Before You Begin

1. Right-click on the Linux desktop and select Open Terminal. In the Terminal window, type the command "**ls**" to list the current directory. The data for this exercise is in the sub-directory ***uvm_labs***. Does it exist?
2. If yes, your lab environment is setup. Proceed to the next page.
3. If no, double-click the **Download_lab_data** icon on the desktop. This opens the Firefox browser showing the Functional Verification Training Lab Data page.
4. Click ***uvm_labs_2023.1.tar*** and then 

5. Select the **Save File** option in the next dialog box and click **OK**. Wait for the download to complete and close the browser.
6. Extract the lab files with this command in your home directory.

```
% tar -xvf Downloads/uvm_labs_2023.1.tar
```

8. You should now see the sub-directory ***uvm_labs.***

Notes:

Lab2A UVM First Look

During this lab, you will…

- Learn the UVM testbench topology and file organization

All labs build on a common platform

- **This labs explores the platform**
- **In later labs you replace existing code with your solution**
- **Then your solution will be used in the following labs**
- **Or you can use the provided solution (last resort!)**

- Naming:
  - The SystemVerilog classes for the course start with "`lab_`"
    - `lab_test_reset`, `lab_test_write6`, …
  - UVM classes start with "`uvm_`"
    - `uvm_test`, `uvm_env`, …

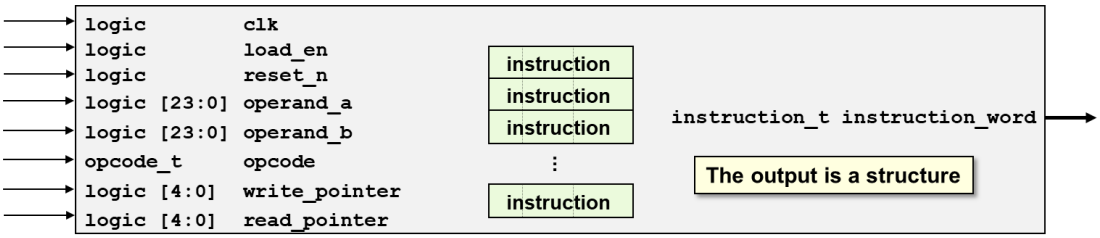
Page 2-3

A

Notes:

Lab2A Instructions

1. Start in the **uvm_labs** directory which has labs, examples, and documentation
   - Change directory to **lab_dut** with the Design Under Test
   - Open **instr_reg.sv** and **instr_reg_pkg.sv** to review the design

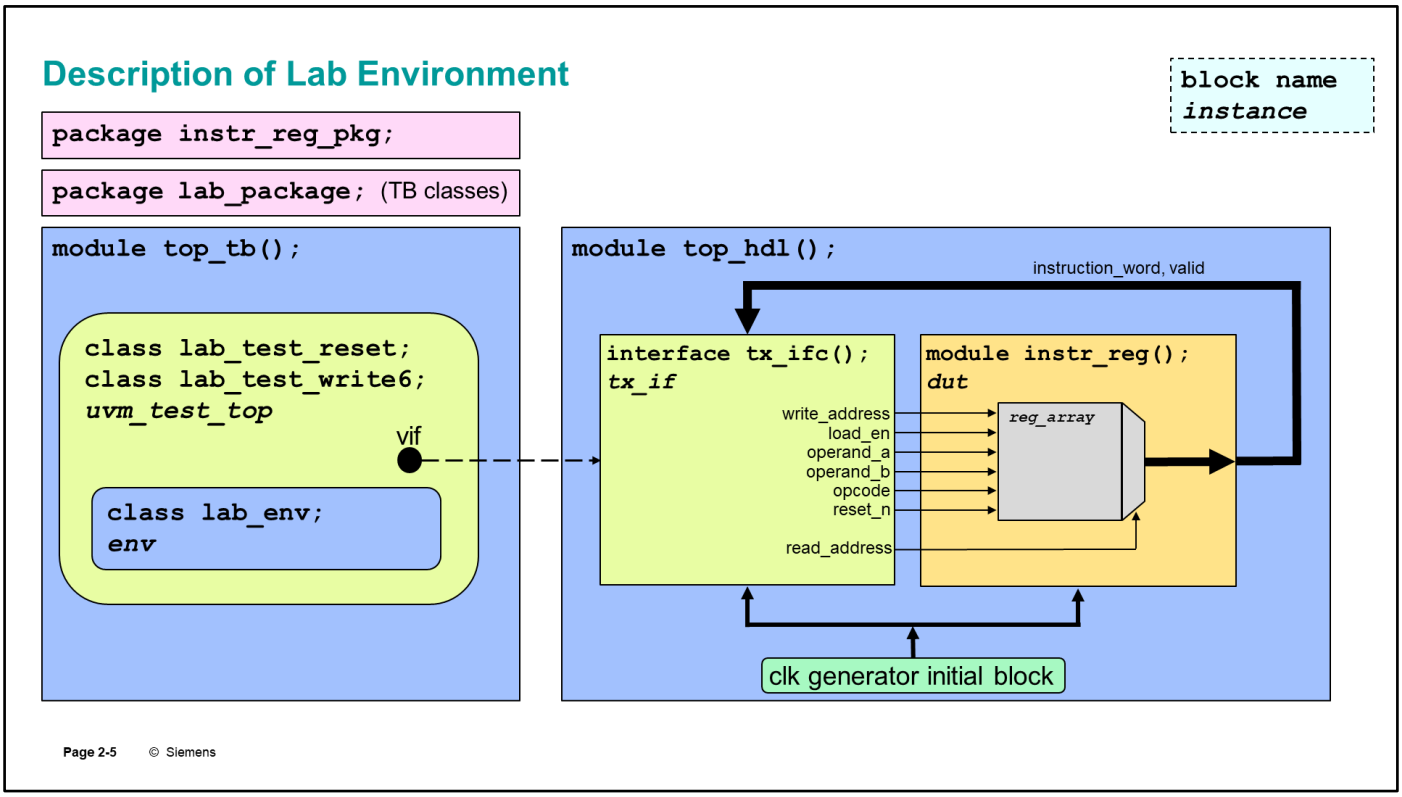


- **The DUT contains an unpacked array of 256 instruction_t structures, beginning with address 0**
- **A new instruction is written into the register on a positive edge of clk if load_en is true, at the address of the write_pointer**
- **An instruction_word is read continuously from the register, at the address of the read_pointer**

Page 2-4 © Siemens

Notes:

This is a simple design, but one that allows this course to focus on a wide variety of aspects with UVM without having to first understand the intricacies of a complex hardware model



Notes:

This is a dual-top system. All testbench code is under top_tb, while all design code and the interface are under top_hdl (Hardware Design Language)
The package instr_reg_pkg has the definitions needed for the design, while lab_package has the lab testbench classes

Lab 2A Instructions (Cont.)

1. Simulate the complete UVM testbench and UART
   - Change to the **lab02_complete_tb** directory
   - Compile and run the reset test class

```
% vlog -f ../run.f
% vopt +acc top_hdl top_tb -o top_opt
% vsim -c +UVM_TESTNAME=lab_test_reset -do "run -all" top_opt
```

   - The file **../run.f** has a list of simulator options and SystemVerilog source files

♦ Examine the simulation output, on the screen and log file: **transcript**
   - What time does Questa print the message "******* TEST RESET *******"? _____

Page 2-6 © Siemens

Notes:

Notice in the vopt command that this is a dual-top simulation: top_hdl has the DUT hierarchy and top_tb has the testbench. Siemens recommends this organization so that you can easily switch to emulation.
See the README.txt for hints on running Questa.
The vsim –c switch runs simulation in command mode and produces the log file: transcript

The reset test has input transactions (TX_IN) but no output transactions (TX_OUT). This is because most inputs assert reset, and the rest do not assert load. So there is nothing in the DUT to check. This is why the scoreboard reports that no transactions passed or failed.

Following UVM guidelines, almost all the stimulus is randomly generated.

Lab 2A Instructions (Cont.)

2. Simulate a different test without recompiling

```
% vsim -c +UVM_TESTNAME=lab_test_write6 -do "run -all" top_opt
```

- Examine the simulation, either on the screen or in the `transcript` log file
  - What is the message for the test being run? _____
  - In the "Test Score" message, how many transactions passed and failed? _____
  - In "UVM Report Summary", how many UVM_INFO ____, UVM_WARNING _____, UVM_ERROR _____?

NOTE: The write6 test generates 6 transactions with randomized values that are sent into the DUT
- **On DUT inputs (the stimulus), the write_pointer has a random value**
- **On DUT outputs, the read_pointer is always set to the value of the previous write_pointer**
- **The first transaction resets the DUT and does not produce an output transaction**
- **An extra transaction is always generated at the end to check the last location written**

3. Run Questa interactively and look at the commands that support UVM debug

```
% vsim -gui -classdebug +UVM_TESTNAME=lab_test_write6 top_opt
```

```
VSIM> uvm help
```

Page 2-7 © Siemens

Notes:

In UVM, all test classes are compiled into the simulation image. This allows you to "compile once, simulate many times"

Lab 2A Instructions (Cont.)

4. Review - all testbench code included in **uvm_labs/lab_testbench/lab_package.sv**

```
package lab_package;
   `include "uvm_macros.svh"
   import uvm_pkg::*;
   import instr_reg_pkg::*;
   typedef class lab_tx_base;
   typedef class lab_tx_in;
   typedef class lab_tx_out;
   …
//   `include "../lab02_complete_tb/lab02_test_message.svh"
//   `include "../lab03_transactions/lab03_tx_base.svh"
//   `include "../lab03_transactions/lab03_tx_in.svh"
//   `include "../lab03_transactions/lab03_tx_out.svh"
…
   `include "../lab_solution/lab02_test_message_solution.svh"
   `include "../lab_solution/lab03_tx_base_solution.svh"
   `include "../lab_solution/lab03_tx_in_solution.svh"
   `include "../lab_solution/lab03_tx_out_solution.svh"
    …
endpackage
```

UVM base class library and macros, and DUT definitions

typedefs allow your code to be included in any order

When you complete your code for lab2, remove the comment here…

… and the solution class will be automatically disabled

Page 2-8 © Siemens A

Notes:

This is a shortened version of the file.
The list of typedef definitions near the top of the file:
These define the class names to be completed later in the package
By first defining all the class names, the actual classes can be in any order, even if one class “has-a” name of another class that has not yet been defined
The list of `include file inclusion directives that are commented out
These are the files that you will create in later labs
As you complete each lab, you will need to uncomment the corresponding `include line in this section of the package
This is a common coding style in UVM. Each class is in its own .svh file, and all files are collected into one or more packages with `include.

The second list of `include directives that are not commented out
These are the solution versions for each of the labs
The solution files use conditional compilation so that if your version of the lab file has been included, the solution is not be compiled

Lab 2A Instructions (Cont.)

5. Change directories to the `uvm_labs/lab_solution` directory
   - This directory contains all the UVM components that make up the Big Picture
     - **As you go through the course, you will replace most pieces of the complete UVM testbench with your own version**
     - **To simulate your version of each UVM component, you will use the solution for any components you have not yet written**
   - Examine some of these files to see how the solutions are coded
     - You are encouraged to compare each lab that you write to these example solution to get ideas of alternate ways to code UVM components
     - You can also refer to these example solutions as you work on each lab if you are not certain how proceed with the lab
   - As you examine these solution files, jot down any questions you have so that we can be sure to answer them as we go through the various UVM components in detail

♦ You are now done with Part A of the lab

Page 2-9 © Siemens

Notes:

Just give the files a quick look. The UVM overview still has a few more topics to cover!

Lab2B Instructions

During this lab, you will…

- Learn how to write and filter UVM messages
- Peek at the UVM source code

- Every class definition is in a separate file
  - All are included in the package `uvm_labs/lab_testbench/lab_package.sv`
  - **IMPORTANT!** As you complete the file for this lab, you must uncomment the corresponding `` `include `` line in the package
  - But do not comment out the solution `` `include `` line
  - If your code compiles the first time with no errors, you may be still using the solution files

Page 2-10 © Siemens

A

Notes:

Lab 2B Instructions (Cont.)

1. Move to the `uvm_labs/lab02_complete_tb` directory
   a. Edit a simple test `lab02_test_message.svh` and add messages as instructed in the comments starting with: `// LAB 2 ASSIGNMENT`
   b. Include this class and messages in your simulation
      – Edit `lab_testbench/lab_package.sv`
      – Take a moment to read the comment blocks and become familiar with the contents of this file
      – Uncomment the `` `include `` line that inserts the file you just completed
   c. Run the script `3step.csh` to compile, optimize, and run the message test

```
% cd lab02_complete_tb
% ../3step.csh lab_test_message
```

   How many UVM_INFO messages were printed? _____
   d. Run the script `rerun.csh` to rerun the test with different verbosity levels. You should see a different set of messages for each run.

```
% ../rerun.csh lab_test_message +UVM_VERBOSITY=UVM_NONE
% ../rerun.csh lab_test_message +UVM_VERBOSITY=UVM_LOW
% ../rerun.csh lab_test_message +UVM_VERBOSITY=UVM_FULL
% ../rerun.csh lab_test_message +UVM_VERBOSITY=UVM_DEBUG
```


Notes:

If you are running on Windows, run ..\3step.cmd and ..\rerun.cmd
If you are running in Questa, use the files ../3step.do and ../rerun.do

Follow the message examples from the lecture and in the sample code.

With UVM_DEBUG, you will see many UVM internal messages. This is why you don't want to use this verbosity level, just UVM_FULL.

Lab 2B Instructions (Cont.)

If you have extra time at the end of this lab…

2. Unpack the documentation and open the UVM Reference Manual in a browser

```
% cd ~/uvm_labs
% tar -zxf docs.tgz
% firefox docs/UVM_Reference.html &
```

   – Look for **+UVM_TESTNAME** in the search box located at the bottom of the list on the left side of the window.

   What happens if you give this switch more than once? _____

- Open **docs/uvm-1.1_class-reference-manual.pdf** with **acroread** or **evince**
  – This can help you apply UVM constructs in your testbench
  – Find the description of **+UVM_TESTNAME** in the Command Line Debug section. What happens if you give this switch multiple times?

  _____

Refer to these documents during the labs

These documents are your primary resource during the labs for the full syntax and correct usage of the UVM base class library and macros

Page 2-12 © Siemens

Notes:

You can also download the UVM 1.1 and 1.2 class reference manuals and user guides from http://accellera.org/downloads/standards/uvm
The IEEE 1800.2 standard is available from http://accellera.org/downloads/ieee

Questa SIM Quick Guide - Interactive Commands

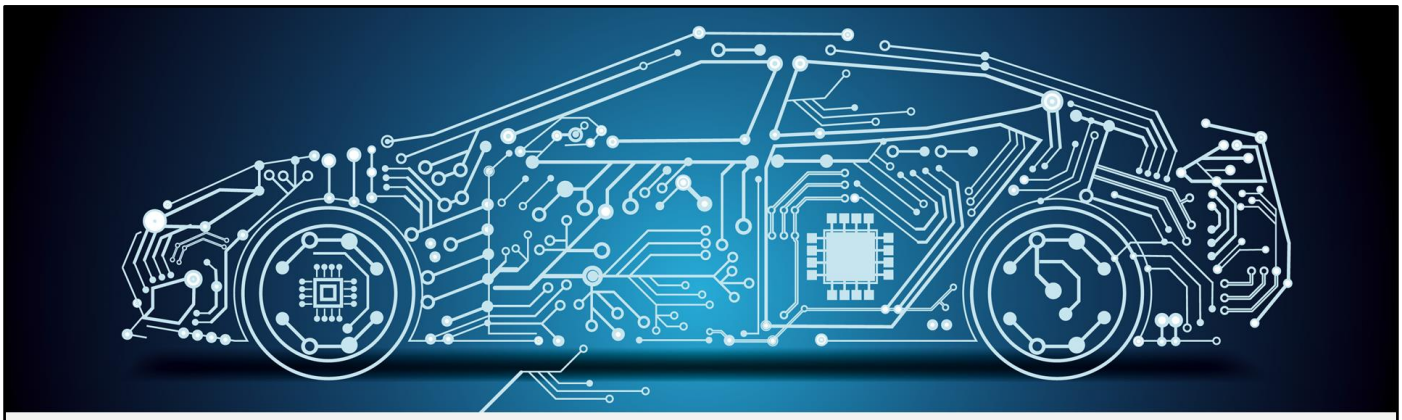
Key Simulator Commands

add memory opens the specified memory in the MDI frame
add watch adds signals or variables to the Watch window
add wave add Verilog nets and registers to the Wave window
alias creates a Tcl procedure that evaluates the specified commands
change modifies the value of a Verilog register variable
checkpoint saves the state of your simulation
delete removes objects from either the List or Wave window
do executes commands contained in a macro file
drivers displays the current value and scheduled values for all the drivers of a Verilog net
echo displays a specified message in the Main window
edit invokes the editor specified by the EDITOR environment variable
environment displays or changes the current dataset and region
environment examine examines one or more objects
find displays the full pathnames of all objects in the design whose names match the specification
force applies stimulus to Verilog nets
history lists the commands executed during the current session
next continues a search;
noforce removes the effect of any active force commands on the selected object
notepad opens a simple text editor
printenv echoes the names and values of all environment variables

profile on enables runtime profiling of simulation time
pwd displays the current directory path in the Main window
radix specifies the default radix
report displays the value of all simulator control variables
restart reloads the design elements and resets the sim time to zero
restore restores the state of a simulation that was saved with a checkpoint command during the current invocation of vsim
resume resumes execution of a macro file after a pause command or a breakpoint
right searches right (next) for signal transitions or values in the specified Wave window
run advances the simulation by the specified number of timesteps
search searches the specified window for one or more objects matching the specified pattern(s)
verror prints a detailed description of a message number
view opens a QuestaSim window
vlog compiles Verilog design units and SystemVerilog extensions
vmap defines a mapping between a logical library name and a directory
vopt produces an optimized version of your design
vsim loads a new design into the simulator
when perform actions when the specified conditions are met
where displays information about the system environment

Supplemental Information

Notes:



Lab 3: Define Transactions and Sequences

SystemVerilog UVM

SIEMENS

Notes:

Version 2023.1

Objectives

- Write UVM sequence items and define the protocol-specific behavior
- Write sequences that create and send these transactions

IMPORTANT!

Lab 3A includes instructions for the Siemens recommended `do_*()` methods and sequences.

Lab 3B is if your company uses field macros and `` `uvm_do() `` macros.

SIEMENS

Notes:

Lab 3A Instructions

- This style is for the Siemens recommended `do_*()` methods
- Change to the directory `uvm_labs/lab03_transactions`

- You will complete these lab files:
  - `lab03_tx_base.svh`, `lab03_tx_out.svh`, `lab03_tx_in.svh` – sequence item definitions
  - `lab03_sequence_reset.svh` – sequence to reset the DUT
  - `lab03_sequence_write6.svh` – sequence to send 6 write transactions into the DUT

- Every class definition is in a separate file
  - All are included in the package `uvm_labs/lab_testbench/lab_package.sv`
  - If your code complies the first time, you probably forgot to include your code in the package

Page: 2

Notes:

Additional files that are part of this lab are:
uvm_labs/lab_dut/instr_reg.sv – the Design Under Test (DUT)
uvm_labs/lab_dut/instr_reg_pkg.sv – typedefs used by the DUT
uvm_labs/lab_testbench/tb_top.sv – top-level module of the testbench
uvm_labs/lab_testbench/tx_ifc.sv – a SystemVerilog interface to connect the testbench & DUT
run.f – Verilog file with file names and switches for compiling the testbench and DUT

Lab 3A Instructions (Cont.)

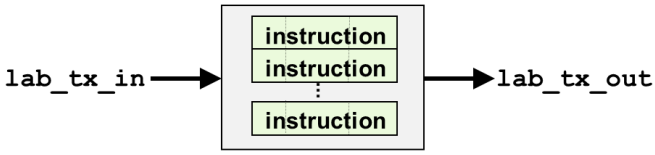
1. Define 3 transaction classes and include them in the testbench

a. Create the base transaction definition
- Edit `lab03_tx_base.svh` and complete the base transaction class

b. Create transactions for the DUT inputs and outputs
- Edit `lab03_tx_in.svh` and complete the DUT input transaction
- Edit `lab03_tx_out.svh` and complete the DUT output transaction

c. Include these files in your simulation
- Edit `lab_testbench/lab_package.sv`
  - Uncomment the `` `include `` lines for the above three files
  - Don't modify the lines for the solution files

IMPORTANT!
The files contain detailed directions:
`// LAB 3 instructions…`



Page: 3 © Siemens

Notes:

To reduce lab time spent typing, partial code for each of these classes is provided

TIP: If you get stuck trying to complete a lab step, peek at the same class in solution directory

Write the do_compare() method based on the example in the slides

The convert2string() methods are called in the lab_monitor class
The compare() method is called in the lab_scoreboard class
Make sure you use the correct SystemVerilog 4-state equality operator, not the 2-state one

Lab 3A Instructions (Cont.)

d. In the `lab03_transactions` directory, simulate the `lab_test_reset` and check that it works correctly

- Run the script `3step.csh` to compile, optimize, and run the reset test

```
% cd lab03_transactions
% ../3step.csh lab_test_reset
```

- The lab includes a `convert2string()` method so you can focus on the sequence item methods
- The field macros print transactions with `sprint()`

♦ TIP: You can plug or unplug your UVM classes from any lab

- Uncomment a `` `include `` line to use your version of a class
  - The solution version will automatically be ignored
- Comment out a `` `include `` line to exclude your version of a class
  - The solution class definition will automatically be used instead
- Don't modify the `` `include `` line for any solution file

Page: 4

Notes:

Questa commands to compile and run:
% vlog -f ../run.f
% vopt +acc top_hdl top_tb -o top_opt
% vsim -batch +UVM_TESTNAME=lab_test_reset -do "run -all; quit" top_opt

Following UVM guidelines, almost all the stimulus is randomly generated.

On Windows, the scripts are 3step.cmd, gui.cmd, and rerun.cmd
You can run them from the lab directory with ..\3step lab_test_reset

Lab 3A Instructions

Short on time?
Copy these sequences from the `lab_solution` directory

This style is for the Siemens recommended `start_item()`/`finish_item()` style

2. Define sequence classes and add them to the testbench
   a) Edit `lab03_sequence_reset.svh` and modify the sequence class
   b) Edit `lab03_sequence_write6.svh` and modify the sequence class
   c) Edit `lab_package.sv` and uncomment the `` `include `` lines for these two sequence files
      – Don't include the `lab03_sequence_*_do.svh` files, and don't modify the lines for the solution files
   d) Run the `lab_test_reset` and `lab_test_write6` tests, and check that the inputs are sent to the DUT
      – You can run without compilation with the `../rerun.csh` script
      – If you get unexpected mismatches, check your `do_compare()` methods and the `===` operator

```
% ../3step.csh lab_test_reset
% ../rerun.csh lab_test_write6
```

```
% ../vis.csh lab_test_write6  -- run with Visualizer GUI
% ../gui.csh lab_test_write6  -- run with Classic GUI
```

- You are now done with Lab 3A

Page: 5

Notes:

The first transaction from the design is generated during reset and is not valid.
The write6 sequence generates an extra transaction at the end to read the result.

Lab 3B Instructions

- The following instructions are for companies that use alternative styles for sequence items (field macros) and sequences (`` `uvm_do*() `` macros)

Page: 6

Notes:

Lab 3B Instructions

- This style is for companies that use the UVM field macros, not the `do_*()` methods
- Change to the directory `uvm_labs/lab03_transactions`

- You will complete these lab files:
  - `lab03_tx_base_field_macros.svh`, `lab03_tx_out_field_macros.svh`, `lab03_tx_in.svh` – sequence item definitions
  - `lab03_sequence_reset.svh` – sequence to reset the DUT
  - `lab03_sequence_write6.svh` – sequence to send 6 write transactions into the DUT

- Every class definition is in a separate file
  - All are included in the package `uvm_labs/lab_testbench/lab_package.sv`

Page: 7

Notes:

Some additional files that are part of this lab are:
uvm_labs/lab_dut/instr_reg.sv – the Design Under Test (DUT)
uvm_labs/lab_dut/instr_reg_pkg.sv – typedefs used by the DUT
uvm_labs/lab_testbench/tb_top.sv – top-level module of the testbench
uvm_labs/lab_testbench/tx_ifc.sv – a SystemVerilog interface that connects the testbench to the DUT
run.f – Verilog file with file names and switches for compiling the testbench and DUT

Lab 3B Instructions (Cont.)

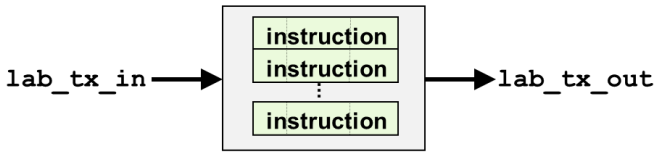
1. Define 3 transaction classes and include them in the testbench

a. Create the base transaction definition
   - Edit `lab03_tx_base_field_macros.svh` and complete the class

b. Create transactions for the DUT inputs and outputs
   - Edit `lab03_tx_in.svh` and complete the DUT input transaction
   - Edit `lab03_tx_out_field_macros.svh` and complete the DUT output transaction

c. Include these files in your simulation
   - Edit `lab_testbench/lab_package.sv`
     - Uncomment the `` `include `` lines for the above three files
     - Don't modify the lines for the solution files

IMPORTANT!
The files contain detailed directions:
`// LAB 3 instructions…`



Page: 8 © Siemens

Notes:

To reduce lab time spent typing, partial code for each of these classes is provided

TIP: If you get stuck trying to complete a lab step, peek at the same class in solution directory

Write the do_compare() method based on the example in the slides

The convert2string() methods are called in the lab_monitor class
The message has a UVM_FULL verbosity, and will only be displayed when simulation is run with +UVM_VERBOSITY=UVM_FULL
The compare() method is called in the lab_scoreboard class

Lab 3B Instructions (Cont.)

d. In the `lab03_transactions` directory, simulate the `lab_test_reset` and check that it works correctly

- Run the script `3step.csh` to compile, optimize, and run the reset test

```
% cd lab03_transactions
% ../3step.csh lab_test_reset
```

- The lab includes a `convert2string()` method so you can focus on the sequence item methods
- The field macros print transactions with `sprint()`

♦ TIP: You can plug or unplug your UVM classes from any lab

- Uncomment a `` `include `` line to use your version of a class
  - The solution version will automatically be ignored
- Comment out a `` `include `` line to exclude your version of a class
  - The solution class definition will automatically be used instead
- Don't modify the `` `include `` line for any solution file

Page: 9

Notes:

Questa commands to compile and run:
% vlog -f ../run.f
% vopt +acc top_hdl top_tb -o top_opt
% vsim -batch +UVM_TESTNAME=lab_test_reset -do "run -all; quit" top_opt

Following UVM guidelines, almost all the stimulus is randomly generated.

On Windows, the scripts are 3step.cmd, gui.cmd, and rerun.cmd
You can run them from the lab directory with ..\3step lab_test_reset

Lab 3B Instructions (Cont.)

Short on time?
Copy these sequences from the `lab_solution` directory

This style is for companies that use the `uvm_do*() macro style

2. Define sequence classes and add them to the testbench
   a) Edit `lab03_sequence_reset_do.svh` and modify the sequence class
   b) Edit `lab03_sequence_write6_do.svh` and modify the sequence class
   c) Edit `lab_package.sv` and uncomment the `include lines for these two sequence files
      – Don't include the `lab03_sequence` files without "`_do`", and don't modify the lines for the solution files
   d) Run the `lab_test_reset` and `lab_test_write6` tests, and lab scoreboard check that the inputs are sent to the DUT
      – `rerun.csh` script

```
% ../3step.csh lab_test_reset
% ../rerun.csh lab_test_write6
```

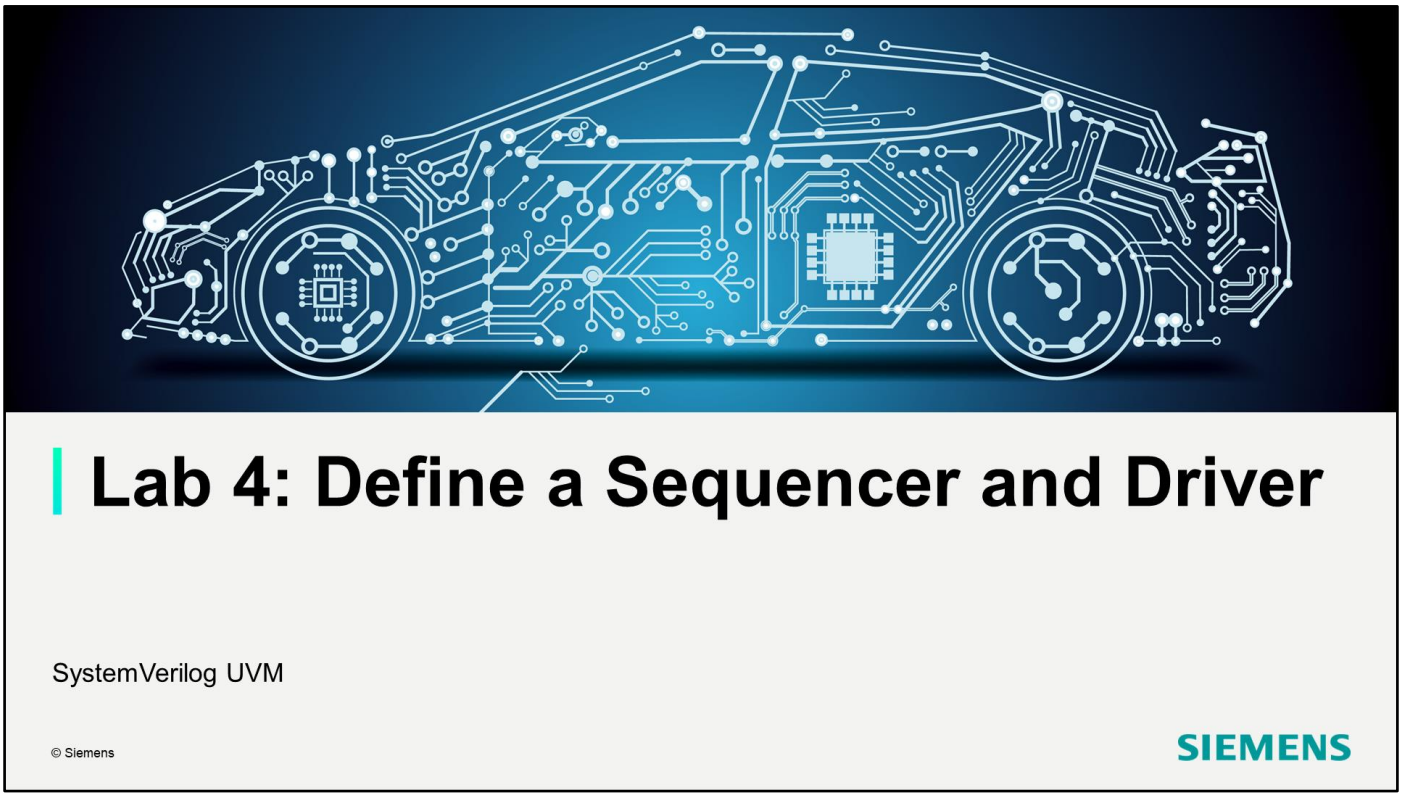
```
% ../vis.csh lab_test_write6   -- run with Visualizer GUI
% ../gui.csh lab_test_write6   -- run with Classic GUI
```

- You are now done with Lab 3B

Page: 10 © Siemens

Notes:

The first transaction from the design is generated during reset and is not valid.
The write6 sequence generates an extra transaction at the end to read the result.



Notes:

Version 2023.1

Objectives

- Write a UVM driver that receives items from the sequencer and calls methods in an interface
- Replace the example solution and simulate

SIEMENS

Notes:

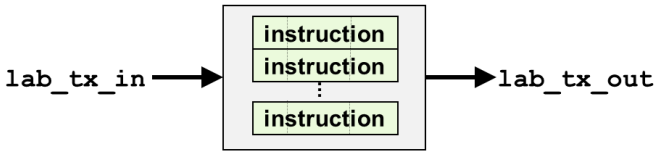
Lab 4 Instructions

- The files for this lab are in `uvm_labs/lab04_sequencer_driver`
- The driver for this lab is:
  - `lab04_driver.svh` – Drive transactions into the DUT
  - The sequencer is a specialization of uvm_sequencer and does not require a new type

IMPORTANT!

This lab uses the transactions and sequences from lab 3

- The `lab_package.sv` contains `` `include `` directives for your lab 4 files
- If you did not complete lab 3, comment out the corresponding `` `include `` lines in the package, and the solutions for those files will automatically be used



Page: 2

Notes:

Lab 4 Instructions (Cont.)

IMPORTANT!
The files contain directions for each part of the lab:
`// LAB 4 instructions`

1. Define your driver class and substitute it into the testbench
   - Edit **`lab04_driver.svh`**, and complete the driver definition by:
     - Creating the **`run_phase()`** method that sends transaction to the DUT
       - The driver's **`run_phase()`** contains a **`forever`** loop
       - This is a typical approach in UVM – the driver never stops asking for new transactions
       - The sequence controls the stream of transactions
     - The driver reads the virtual interface from the UVM configuration DB (shown in module 8)
2. Include this file in your simulation
   - Edit **`uvm_labs/lab_testbench/lab_package.sv`** and uncomment the line for your **`lab04_driver.svh`** file, but don't touch the line for the solution file
3. Run simulations with the reset and write6 test cases (see Lab 3 for tips)
   - Check the log file to verify that the driver is passing values to the DUT
   - You should see the same results as the previous lab

Page: 3

Notes:

This driver does not access any design signals, such as clock, reset, or the inputs or outputs

Notes:

Version 2023.1

Objectives

- Model a UVM monitor
- Model a UVM agent
- Connect them with TLM analysis ports

SIEMENS

Notes:

Lab Instructions

1. Move to the lab directory `uvm_labs/lab05_monitor_agent`
2. Create a UVM monitor class
   - Edit `lab05_monitor.svh` and complete the class definition. Only follow the "`LAB 5 ASSIGNMENT, STEP 1`", "`2`", and "`3`" . Do not follow step "`4`".
3. Create a UVM agent definition
   - Edit `lab05_agent.svh`, follow both "`LAB 5 ASSIGNMENT`" steps. Ignore other labs.
4. Uncomment the `` `include `` lines for these files in `lab_package.sv`
5. Simulate the UVM testbench and design
   a) Compile and run a simulation with `lab_test_reset`

```
% ../3step.csh lab_test_reset
```

   b) Monitor transactions don't show as they are printed with `UVM_FULL`, and you ran with the default verbosity of `UVM_MEDIUM`. Rerun with verbosity set to `UVM_FULL`.

```
% ../rerun.csh lab_test_reset +UVM_VERBOSITY=UVM_FULL
```

   The monitor messages print, and ones from other components

Notes:

The messages from the monitor have the verbosity UVM_FULL so that they don't print during a typical simulation
If you run the whole simulation with UVM_FULL, messages with this verbosity all over the testbench will print.
This course focuses on this one interface, so you want to see the messages, but don't want to change the monitor code

Lab Instructions (Cont.)

c) Set the verbosity of ALL messages in just the monitor to `UVM_FULL` during the run phase

```
% ../rerun.csh lab_test_reset \
   +uvm_set_verbosity=uvm_test_top.env.agt.mon,_ALL_,UVM_FULL,run
```

That displays the messages, but is a long, error-prone switch to type for every run. See the Notes below for a description of this switch.

6. In `lab05_monitor.svh`, make the edit for "`LAB 5 ASSIGNMENT, STEP 4`" to set the monitor's verbosity to the value in the agent configuration object, set to UVM_FULL. Now verbosity is set as part of configuration.
7. Simulate
   a) Compile and run the reset test and observe the monitor transactions

```
% ../3step.csh lab_test_reset
```

   b) Rerun with the write6 sequence and look at the transactions

```
% ../rerun.csh lab_test_write6
```

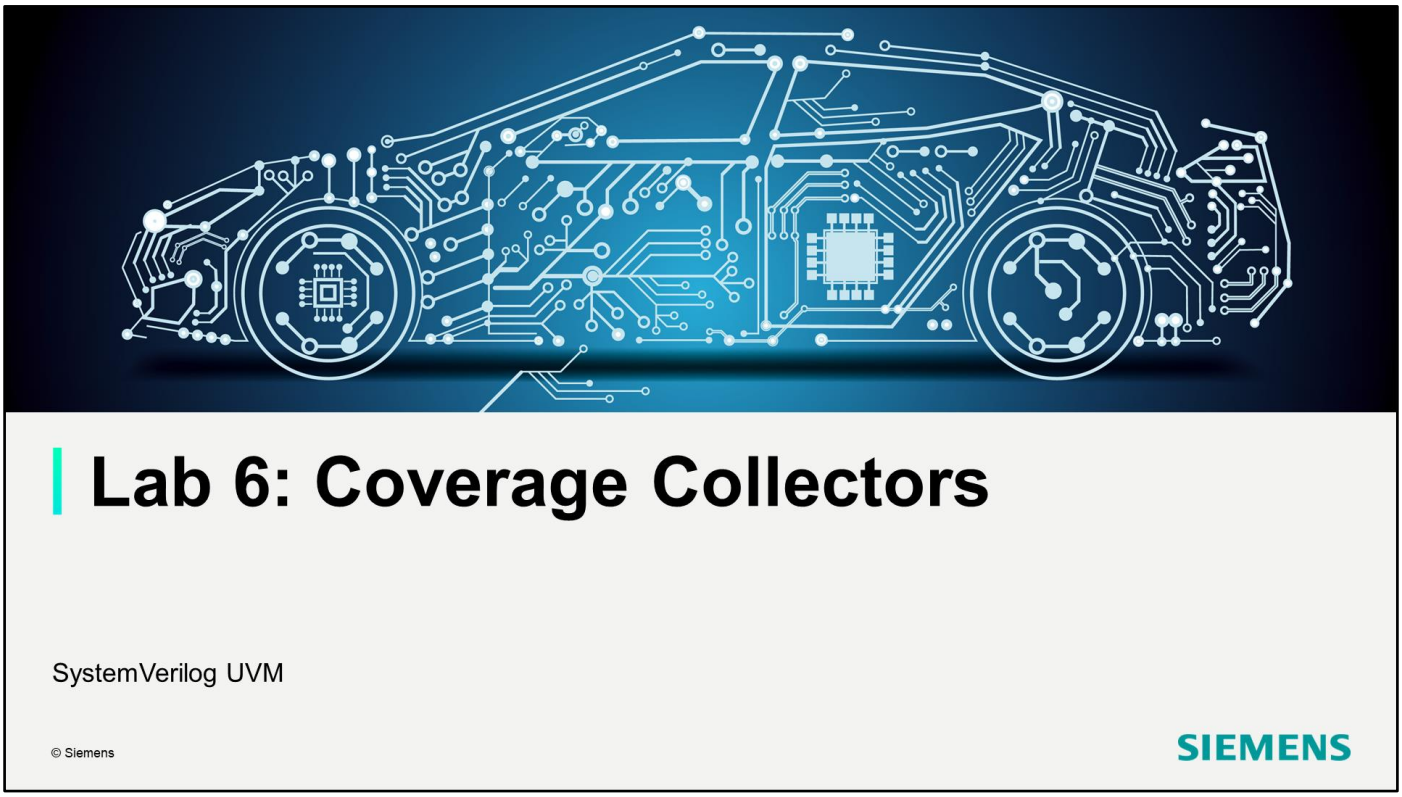
Page 5-3 © Siemens

Notes:

The +uvm_set_verbosity allows you to change the verbosity of messages for specific components at specific phases.
The first argument is the UVM path to the component. Remember that uvm_test_top is the instance name of your test.
The second argument is the ID of the message that you want to change. "_ALL_" tells UVM to change all messages in the monitor.
The third argument is the new verbosity level. Since this is equal or greater than your monitor message ID, the messages print.
Lastly is the phase when the verbosity changes, in this case the run_phase.

The switch +uvm_set_verbosity is lower case, which means you can give it multiple times on one command line. +UVM_VERBOSITY is upper case, so it can only be given once.

UVM 1.1d has a bug where only one +uvm_set_verbosity works. Multiple copies of the switch are ignored.



Notes:

Version 2023.1

Objectives

- Model a UVM coverage collector
- Connect a coverage collector to a monitor
- Simulate and review the functional coverage reports

SIEMENS

Notes:

Lab Instructions

1. Review the design in **uvm_labs/lab_dut**
   - Open **instr_reg.sv** and **instr_reg_pkg.sv**
2. Move to the lab directory **uvm_labs/lab06_coverage**
3. Define a UVM coverage collector
   - Edit **lab06_coverage.svh** and complete the class definition, following the comments in the file. There are **TWO** sections that you need to complete.
4. Modify the agent component to create a coverage collector
   - Edit **lab05_monitor-agent/lab05_agent.svh** and add the code to build and connect the coverage collector, following the comments in the file. There are **TWO** sections that you need to complete.
   - This is an agent-level coverage collector, so it is controlled at the agent level
   - NOTE: Leave the file in the **lab05** directory; that is where the **`include** directive in the package looks for it
5. Uncommon the **`include "lab06_coverage.svh"** in **lab_package.sv**

Notes:

Hint: Search for "// LAB 6" in the files to locate the specific tasks

Lab Instructions (Cont.)

6. Verify the UVM testbench
   - Compile and simulate the write6 test

```
% cd lab06_coverage
% ../3step.csh lab_test_write6
```

   - What is the functional coverage percentage reported by the coverage collector? _____
   - NOTE: You will not get high coverage in this lab!
     - Only a small number of transactions are being generated by the tests
     - Not all the functionality is being tested at this point
     - Sequence item constraints prevent some input values from occurring

Page: 3

Notes:

The exact coverage percentage will be different on different Questa versions because the stimulus transaction values are random values

The coverage percentage is not expected to be very high in this lab
The lab_test_write6 only generates 6 random stimulus transactions – not enough to fully exercise the design being tested
A later lab will make the number of transactions that are generated configurable
This will make it easy to increase the number of stimulus transactions to obtain better total coverage

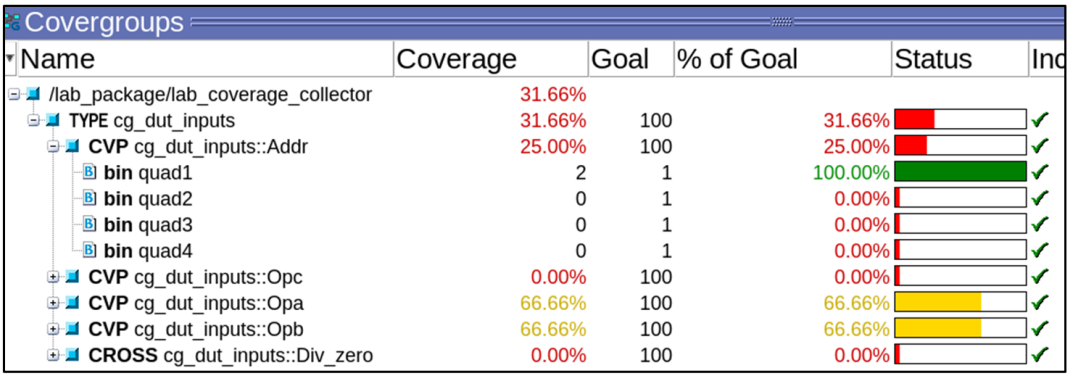
Run the simulation interactively with the gui script

Lab Instructions (Cont.)

7. Review the coverage results in the QuestaSim GUI
   - Run the simulation in the GUI mode

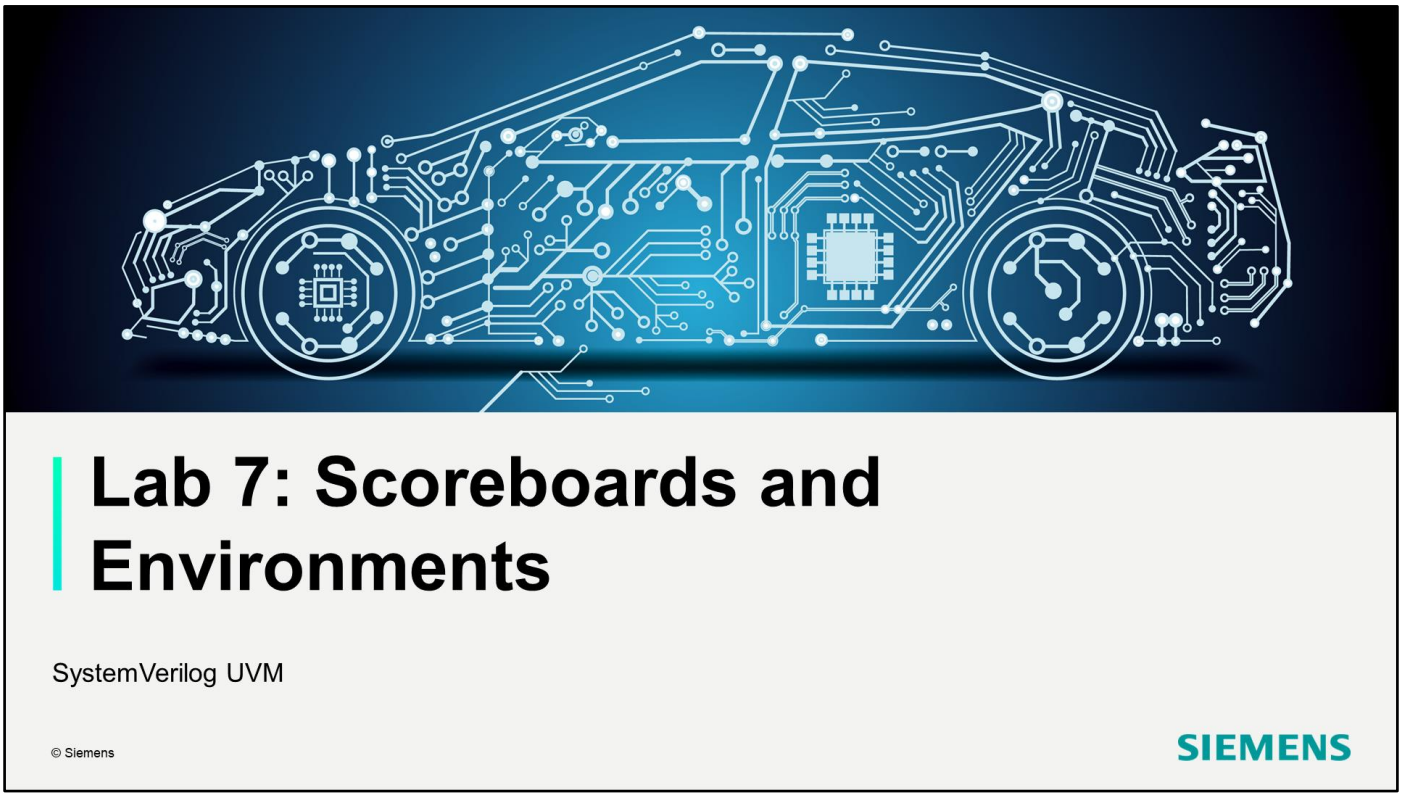
```
% ../gui.csh lab_test_write6
vsim> run -all
```

   - Open the Covergroups window: **View > Coverage > Covergroups**
   - Identify what coverpoint values are not being generated by the test sequences (the stimulus)



Name	Coverage	Goal	% of Goal	Status	Inc
/lab_package/lab_coverage_collector	31.66%				
TYPE cg_dut_inputs	31.66%	100	31.66%		
CVP cg_dut_inputs::Addr	25.00%	100	25.00%		
bin quad1	2	1	100.00%		
bin quad2	0	1	0.00%		
bin quad3	0	1	0.00%		
bin quad4	0	1	0.00%		
CVP cg_dut_inputs::Opc	0.00%	100	0.00%		
CVP cg_dut_inputs::Opa	66.66%	100	66.66%		
CVP cg_dut_inputs::Opb	66.66%	100	66.66%		
CROSS cg_dut_inputs::Div_zero	0.00%	100	0.00%		

Notes:



Notes:

Version 2023.1

Objectives

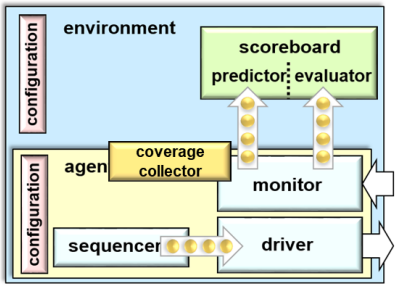
- Model a UVM Scoreboard, with a predictor and evaluator
- Model a UVM Environment
- Connect them with TLM analysis ports

SIEMENS

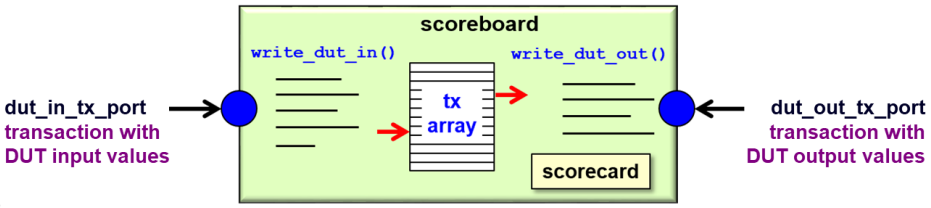
Notes:

Lab Diagram

- The environment contains an agent that sends transactions to the scoreboard



- The scoreboard component includes the predictor and evaluator methods



Page: 2 © Siemens

Notes:

Lab Instructions

1. Move to the lab directory **uvm_labs/lab07_scoreboard_env**
2. Create a UVM scoreboard definition
   - Edit **lab07_scoreboard.svh** and complete the class definition
     - The comments in these files explain what needs to be added
3. Create a UVM environment definition
   - Edit **lab07_environment.svh** and complete the class definition
     - The comments in the file explain what needs to be added
4. Uncomment the **`include** lines for these files in **lab_package.sv**
5. Simulate the UVM testbench and design
   - Compile and run simulations with the **lab_test_write6** test
   - How many transactions passed and failed? _____
   - Re-simulate with **+FORCE_LOAD_ERROR** and run **lab_test_write6** test again
     Did your scoreboard detect errors? _____

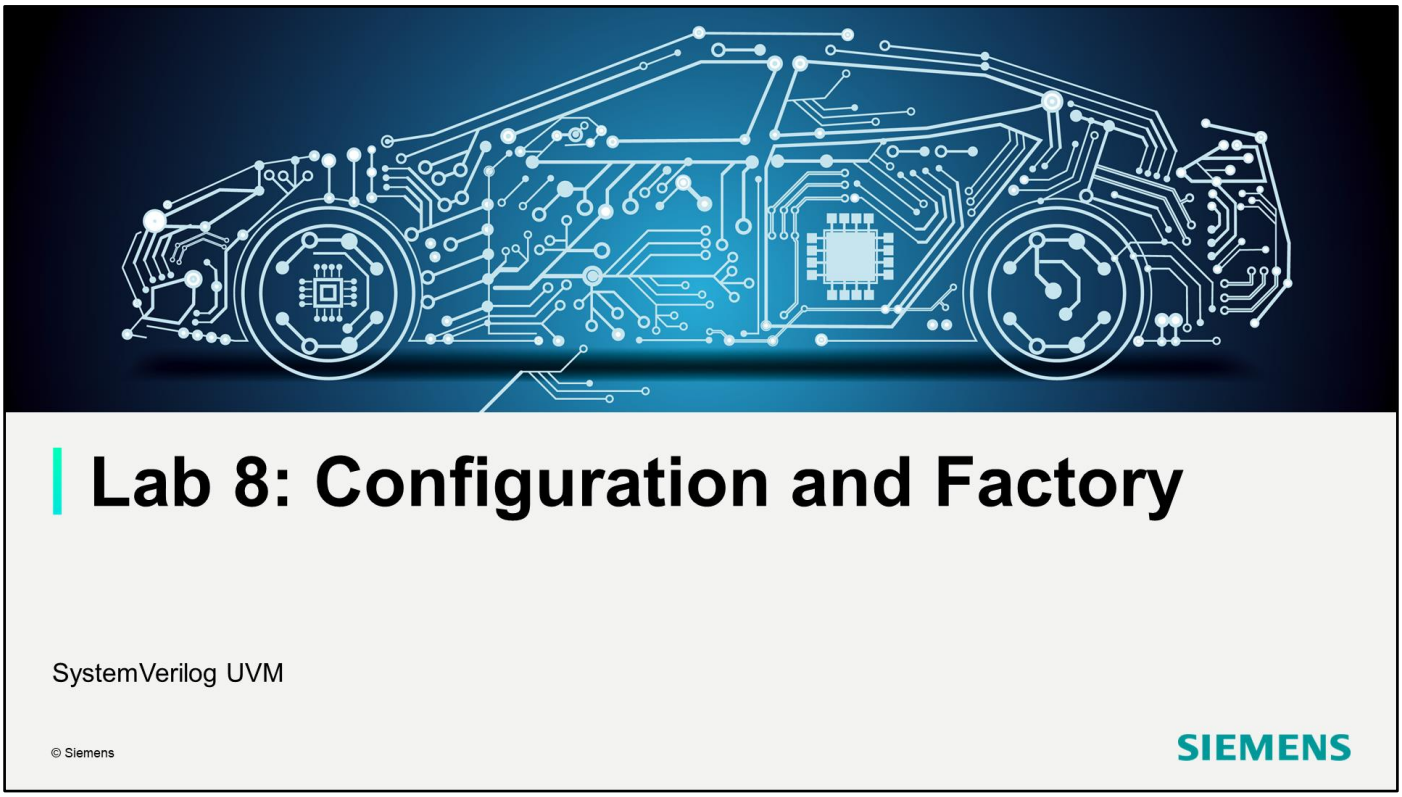
```
% ../rerun.csh lab_test_write6 +FORCE_LOAD_ERROR
```

Page: 3

Notes:

The lab instructions for the predictor portion of the scoreboard add predicting results for write operations
All stimulus should pass during the lab_test_write6 test
Compiling with +define+FORCE_LOAD_ERROR injects an error into the DUT
Your scoreboard should detect several errors during the lab_test_write6 test

To reduce the time required for this lab, this scoreboard does not fully predict the result of every possible input combination
No result is expected for if a write occurs, and load enable is off (the DUT register should retain is previous storage)
When the DUT is being reset, the only expected results are for the locations accessed by the write_pointer (a more complete scoreboard would verify that all register locations were reset, because reset affects all register addresses)
A good way to exercise and test register resets is with the UVM Register Layer, discussed later in this course



Notes:

Version 2023.1

Objectives

- Model a UVM sequence that can be configured without being recompiled
- Use the UVM configuration database to control components in the design
- Use the UVM factory to override sequence items in the design

SIEMENS

Notes:

Lab Instructions

1. Move to the lab directory **`uvm_labs/lab08_configuration`**
2. Create configuration classes for the environment and agent
   - Edit **`lab08_env_config.svh`** and **`lab08_tx_agent_config.svh`**, and complete the classes
   - The comments in the file explain what needs to be added
3. Create a configurable UVM sequence
   - Edit **`lab08_sequence_writeN.svh`** and complete the class
   - The comments in the file explain what needs to be added
4. Create a UVM test that dynamically configures a sequence
   - Edit **`lab08_test_configuration.svh`** and complete the class definition
   - The comments in the file explain what needs to be added
   - Update **`lab_testbench/lab_package.sv`** to include these **4** files

Page: 2

Notes:

This is a simple design, but one that allows this course to focus on a wide variety of aspects with UVM without having to first understand the intricacies of a complex hardware model

Lab Instructions (Cont.)

5. Simulate the configurable UVM testbench with the test **lab_test_configuration**

```
% ../3step.csh lab_test_configuration \
              +uvm_set_config_int=uvm_test_top,count,1
```

- Look for your
- What is the overall functional coverage? _____

```
% ../rerun.csh lab_test_configuration \
              +uvm_set_config_int=uvm_test_top,count,5
```

- Why does the coverage stop increasing? (see the Notes below)

Base lab_tx_in	
count	%Coverage
1	
5	
10	
50	
100	
200	
500	

Page: 3

Notes:

Why can't you get to 100% coverage by running longer sequences?
The lab_tx_in transaction has constraints that prevent generating the full range of values for the write_pointer, operand_a and operand_b
The next step of this lab fixes this by overriding lab_tx_in with a transaction that has different constraints

Lab Instructions (Cont.)

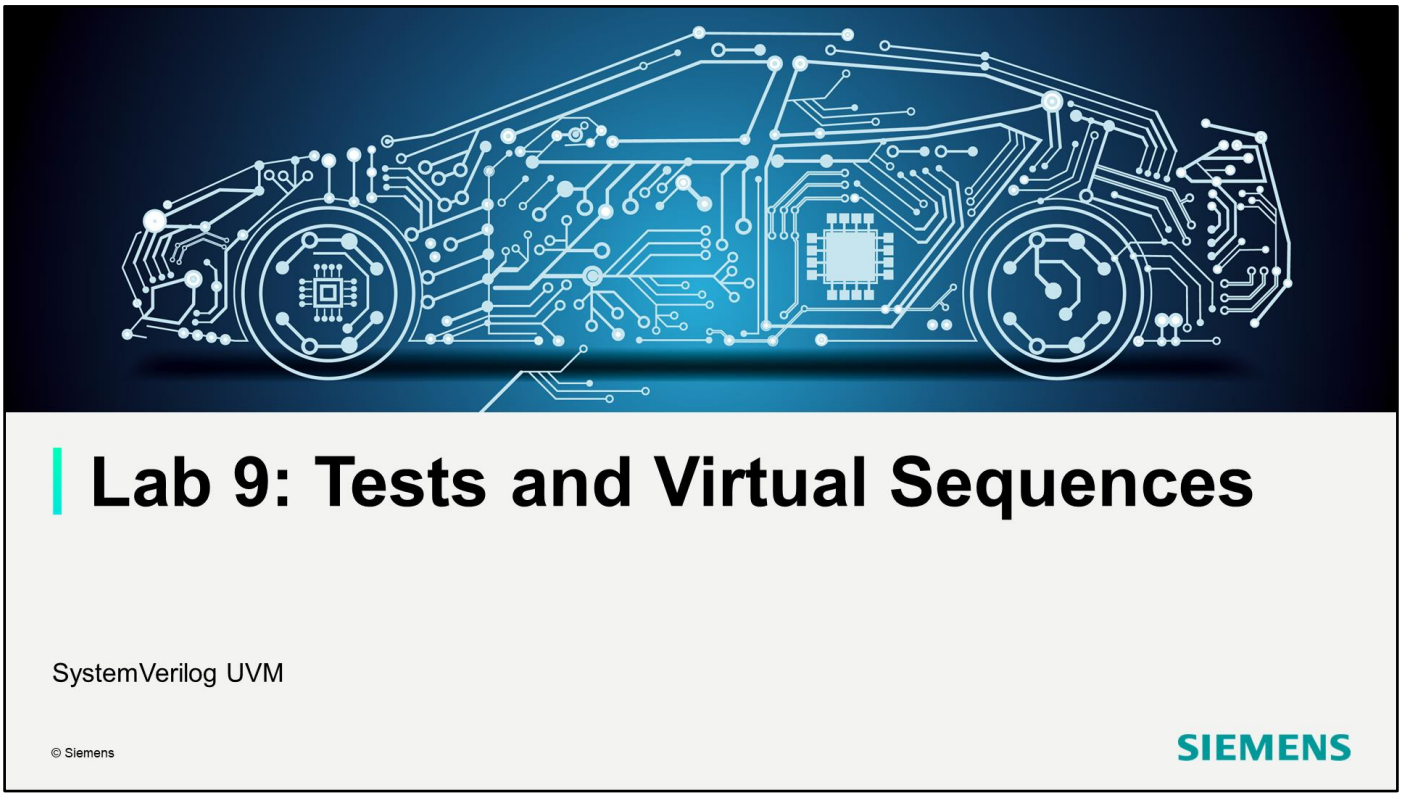
6. Create a custom transaction to override the `lab_tx_in` transaction
   - Edit `lab08_test_override.svh` and override the `lab_tx_in` class with the `lab_tx_in_override` class
     - The comments in the file explains what you need to do
   - Review the transaction defined in `lab_solution/lab08_tx_in_override_solution.svh`
   - Edit `lab_package.sv` and uncomment the `` `include `` line for the `lab_test_override` file
7. Compile and simulate the `lab_test_override` test
   - Compile and run simulations with more and more sequence items and write the overall coverage in the table on the right
   - Did you get a UVM message with ID=SUCCESS?
     - Where did this come from?
     - Look at that file and compare the constraints to `lab_tx_in`

Extended lab_tx_in_override	
count	%Coverage
1	
5	
10	
50	
100	
200	
500	

Page: 4

Notes:

Increasing the number of transactions should help increase the total coverage, but additional changes might be required in order to obtain 100% coverage
A special “directed test” might be needed that overrides some of the constraints built into the transaction class randomization
Examining the simulator-specific detailed coverage reports help determine what stimulus values never occur, and what directed tests are required to force values in those ranges to be generated
Factory overrides can be used to allow a UVM test to run a directed test with special randomization constraints



Notes:

Version 2023.1

Objectives

- Define a virtual sequence and pass it a sequencer handle
- Define a UVM test class
- Put common test code into a base test class
- Start a virtual sequence in a test

SIEMENS

Notes:

Lab Instructions

1. Move to the lab directory **`uvm_labs/lab09_test_vseq`**
2. Create a virtual sequence that runs two actual sequences
   - Edit **`lab09_virtual_sequence.svh`** and complete the class
3. Create a common base test class
   - Edit **`lab09_test_base.svh`** and complete the class
   - Move shared code from the test in **`lab09_test_vseq.svh`** into this class
4. Create a UVM test to run a virtual sequence
   - Edit **`lab09_test_vseq.svh`** and complete the class
   - The test class is simplified by moving common code to the base class
5. Modify the agent component to retrieve values from the configuration object
   - Edit **`lab05_agent.svh`** in the **`lab05_monitor_agent`** directory, look for the one place with the "**`LAB 9`**" comment and add the code to set a sequencer handle in the agent config
   - Leave the file in the **`lab05_monitor_agent`** directory

Notes:

This is a simple design, but one that allows this course to focus on a wide variety of aspects with UVM without having to first understand the intricacies of a complex hardware model

Lab Instructions (Cont.)

6. Uncommon the `include lines for this lab in lab_package.sv

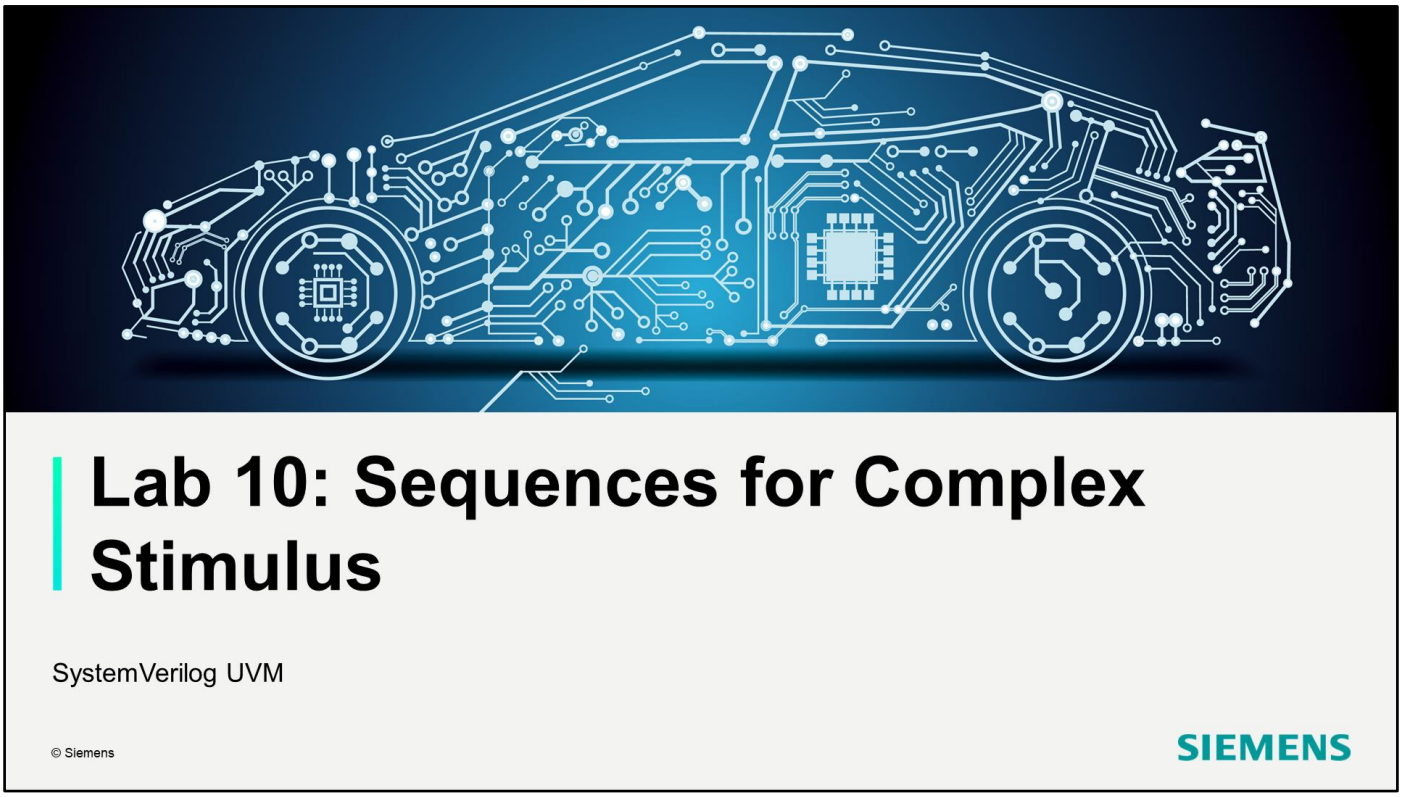
7. Simulate the UVM testbench
   - Compile and run a simulation that runs the virtual sequence test

     ```
     % ../3step.csh lab_test_vseq
     ```
   - Do you see the results from both the reset and write6 sequences? You should!

Page: 3 © Siemens

Notes:

Additional info:
These tests do not fully test the DUT reset
Reset affects all memory locations
The scoreboard developed in an earlier lab only verifies one register location, rather than all locations
A good “extra credit” exercise would be to add a test and scoreboard prediction algorithm specific for reset
This is not as simple as it might appear, however, because reset effects all register locations, which means the predictor must somehow store multiple predicted results
An easier way to handle tests and verification of a register reset is using the UVM Register Abstraction Layer



Notes:

Version 2023.1

Module 10 Lab: Sequences for Complex Stimulus

There is no lab exercise for this section.

- This section has discussed
  - Using multiple agents to connect to the DUT
  - Several ways to generate complex stimulus
- The goal of this section has been to introduce the various UVM coding techniques to verify complex designs.
  - These techniques are tools in your UVM toolbox
  - Each technique might be just the right tool for one project and not needed on another project
  - More information on these techniques is covered in advanced UVM courses

Notes: