Sunburst Design - Universal Verification Methodology

by Recognized Verilog & SystemVerilog Guru, Cliff Cummings of Sunburst Design, Inc.

4 Days
70% Lecture, 30% Lab

Prerequisites (mandatory) - *This is an advanced verification course that assumes engineers already have a good working knowledge of both Verilog & SystemVerilog.*

Course Syllabus

(~10 minute breaks near the top of each hour)
(Lab time is scheduled for "Lunch & Lab" and again near the end of the day)

This course may be customized by client companies on a WebEx conference call with Cliff Cummings.

Day One

UVM Resources & Introduction

Classes & Class Variables

- Intro to OOP using classes
- Class handles & null handles
- Garbage collection
- Built-in & user-defined constructors
- Classes & class extension (inheritance)
- this & super keywords & usage
- Methods & method extension
- Up-casting and down-casting (used extensively in verification environments)
- Local & protected (hiding)
- Static class methods
- Extern methods
- Singleton pattern and usage

UVM Overview First Pass & uvmtb_template files

Each of the concepts in this section will be taught a second time with greater detail in later sections.

- Overview of UVM component & transaction classes
- Basic top module with DUT, dut_if and UVM testbench classes
- Overview of UVM tests & testbench components
- Driver converts transactions into timed "pin-wiggles" and drives them to the dut_if
- Monitor samples dut_if "pin-wiggles" and converts them back into a transaction
- Scoreboard predict, compare and report
- Basic UVM display commands
- Running UVM tests

- UVM phases - top-down & bottom-up - first pass
- Phase names match method names
- UVM starting & stopping tests - raised & dropped objections
- uvmtb_template files used as a starting point for all UVM testbench development
- LAB - UVM Common Errors
- LAB - UVM First Testbench - Testing a Counter *(Full UVM self-checking testbench #1)*

Virtual Classes, Virtual Methods and Virtual Interfaces

Includes materials from Cliff's SNUG 2018 & SNUG 2009 award-winning papers on virtual classes, methods, interfaces.

- Virtual classes (abstract classes)
- Virtual methods
- Overriding virtual methods - polymorphism
- Pure virtual methods
- Virtual interfaces
- 3 requirements for virtual interface usage
- Tying module ports to virtual interfaces
- Passing virtual interface handles through class constructors
- (UVM only) storing virtual interface handles into a database
- Static interfaces -vs- virtual interfaces

Introduction to Constrained Random Testing

- Directed -vs- random testing
- rand & randc class variables
- pre_randomize(), randomize() & post_randomize() class methods
- randomize … with
- rand_mode()
- randcase usage for classes and procedural code
- Constraints
- Using inside & dist keywords

Introduction to Functional Coverage

- Covergroups
- Coverpoints
- Coverpoint bins & auto-bins
- Cross-coverage
- Covergroup sampling techniques
- LABS - Random Variables & Randomization *(Full UVM self-checking testbenches #2-3)*
- LAB - Constrained Random Stimulus *(Full UVM self-checking testbench #4)*

UVM Base Classes & Reporting (UVM print/display commands)

Includes materials from Cliff's SNUG 2014 award-winning paper on UVM messaging.

- UVM messaging methods
- UVM messaging macros
- convert2string()
- UVM verbosities

- UVM verbosity usage guidelines
- LAB - UVM Messaging

Day Two

UVM Transaction Base Classes, Sequences & Tests

Includes materials from Cliff's SNUG 2014 award-winning paper on UVM transactions.

- Dynamic arrays
- Associative arrays
- Why classes -vs- structs?
- `uvm_object_utils macros
- Standard transaction methods
- do_copy, do_compare and other do_methods
- Field macros
- Randomizable data members
- Randomizable knobs
- Randomization constraints
- UVM sequence body task
- start_item(tx) - finish_item(tx)
- `uvm_do macros
- randomize() the transaction
- randomize() the transaction with inline constraints
- UVM sequences of uvm_sequence_item and uvm_sequence
- Running UVM tests

Top Module, DUT and Config Storage Techniques

- Top module
- DUT (Design Under Test)
- DUT Interface
- Connecting DUT to DUT interface
- DUT interface handle
- uvm_config_db#(type) set/get (newer database commands)
- uvm_resource_db#(type) set/read_by_name/read_by_type (newer database commands)
- set_config_* / get_config_* (older storage commands)
- Virtual interfaces for verification
- LAB - UVM Agent (Sqr-Drv-Mon) *(Full UVM self-checking testbench #5)*

UVM Testbench Environment / Agent / Sequencer / Driver / Monitor

Includes materials from Cliff's SNUG 2018 award-winning paper on UVM analysis ports.

- `uvm_component_utils macros
- UVM components connected through ports & exports
- Testbench driver (get-port configuration)
- Managing the virtual interface - config table - required dynamic casting
- Testbench sequencer (get-export configuration)
- Testbench agent & environment
- Active and passive agents
- UVM analysis ports

- Analysis port broadcast command
- UVM monitors with analysis ports
- uvm_subscriber with analysis export
- Connecting a coverage collector using an analysis export
- LAB – FIFO Gray Code Pointer - *(Full UVM self-checking testbench #6)*

UVM Scoreboards - Part I

Includes materials from Cliff's SNUG 2013 paper on UVM scoreboard architectures.

- SystemVerilog queues
- SystemVerilog mailboxes
- uvm_tlm_fifo
- uvm_tlm_analysis_fifo
- What is the job of the scoreboard
- Scoreboard architecture style #1
- Pre-coded scoreboard wrapper and predictor
- Extern calc_exp function - requires user to complete this function
- Pre-coded comparator with 2 uvm_tlm_analysis_fifos
- LAB – UVM Scoreboard Style #1 - Barrel Shifter - *(Full UVM testbench lab #7)*
- LAB – UVM Scoreboard Style #1 - Pipeline Design - *(Full UVM testbench lab #8)*

Day Three

UVM Scoreboards - Part II

Includes more materials from Cliff's SNUG 2013 paper on UVM scoreboard architectures.

- Scoreboard architecture style #2
- Multiple analysis implementation ports
- `uvm_analysis_imp_decl macros
- LAB – UVM Scoreboard Style #2 - 2 Analysis Imp Ports - *(Full UVM testbench lab #9)*

Fork-Join Enhancements & Advanced UVM Sequence Generation

Includes materials from Cliff's DVCon 2016 paper on UVM virtual sequences.

- New SystemVerilog fork-join processes
- UVM virtual sequences
- Virtual sequencers & virtual sequences - requirements
- m_sequencer, p_sequencer, `uvm_declare_p_sequencer
- Virtual sequence base class details
- Common test_base
- Starting virtual sequences
- Multi-bus virtual sequencer example
- LAB - Virtual Sequencer & Sequences

Clocking Blocks & Verification Timing

Includes materials from Cliff's SNUG 2016 paper on UVM verification timing techniques.

- Testbench stimulus/verification vector timing strategies
- #1step sampling
- Clocking blocks
- Clocking skews

- Default clocking block cycles
- Clocking block scheduling
- UVM usage of clocking blocks in an interface
- UVM driver timing using clocking blocks
- UVM signal sampling using clocking blocks
- LABS - All of the self-checking labs use these clocking block techniques

Transaction Level Modeling (TLM) Basics

- TLM ports & exports
- Why "ports" and "exports"
- TLM put, get and transport configurations
- Transaction-level control flow / data flow / transaction type
- Put configurations
- Get configurations
- Transport configurations

UVM Factory & Constructors

Includes materials from Cliff's SNUG 2012 paper on the UVM factory and overrides.

- UVM factory basics
- Why is a factory used in UVM
- What is needed to use the factory
- new() -vs- type_id::create() construction
- Component and data lookup from the factory
- Running without re-compilation
- Tests can make substitutions without changing the testbench source code
- Introduction to factory overrides

Day Four

Constrained Random Testing and Functional Coverage Part II

- Code coverage -vs- functional coverage
- Covergroups & coverpoints - part 2
- Auto-bins & user-named bins
- Cross coverage
- Covergroup.sample() method
- Transition bins
- Coverage options & coverage capabilities
- Comparing cover to covergroup coverage
- LAB – UVM Functional Coverage - *(Full UVM testbench lab #10)*

UVM Register Abstraction Layer (RAL)

- Register package resources
- Why use the register package?
- Register models & memories
- Register model definition: fields / registers / blocks / maps
- Register adapter & methods - derivative of uvm_object

- Register model - derivative of uvm_component
- Setting the model sequencer handle
- Predictor - derivative of uvm_component
- The RAL API
- Built-In register sequences
- uvm_reg_mem_built_in_seq type & handle declaration
- Setting the seq.model & seq.tests
- Executing the seq.start(null)
- LABS – UVM 4-Register Design (no RAL) - *(Full UVM testbench lab #11)*
- LABS – UVMREG 4-Register Design (using RAL) - *(Full UVM testbench lab #12)*