DVCon 2012

Design & Verification Conference & Exhibition

Sponsored By:

accellera SYSTEMS INITIATIVE

February 28 – March 1, 2012

e/eRM to SystemVerilog/UVM: Mind the Gap, But Don't Miss the Train

by

Michael Horn

Principal Verification Architect

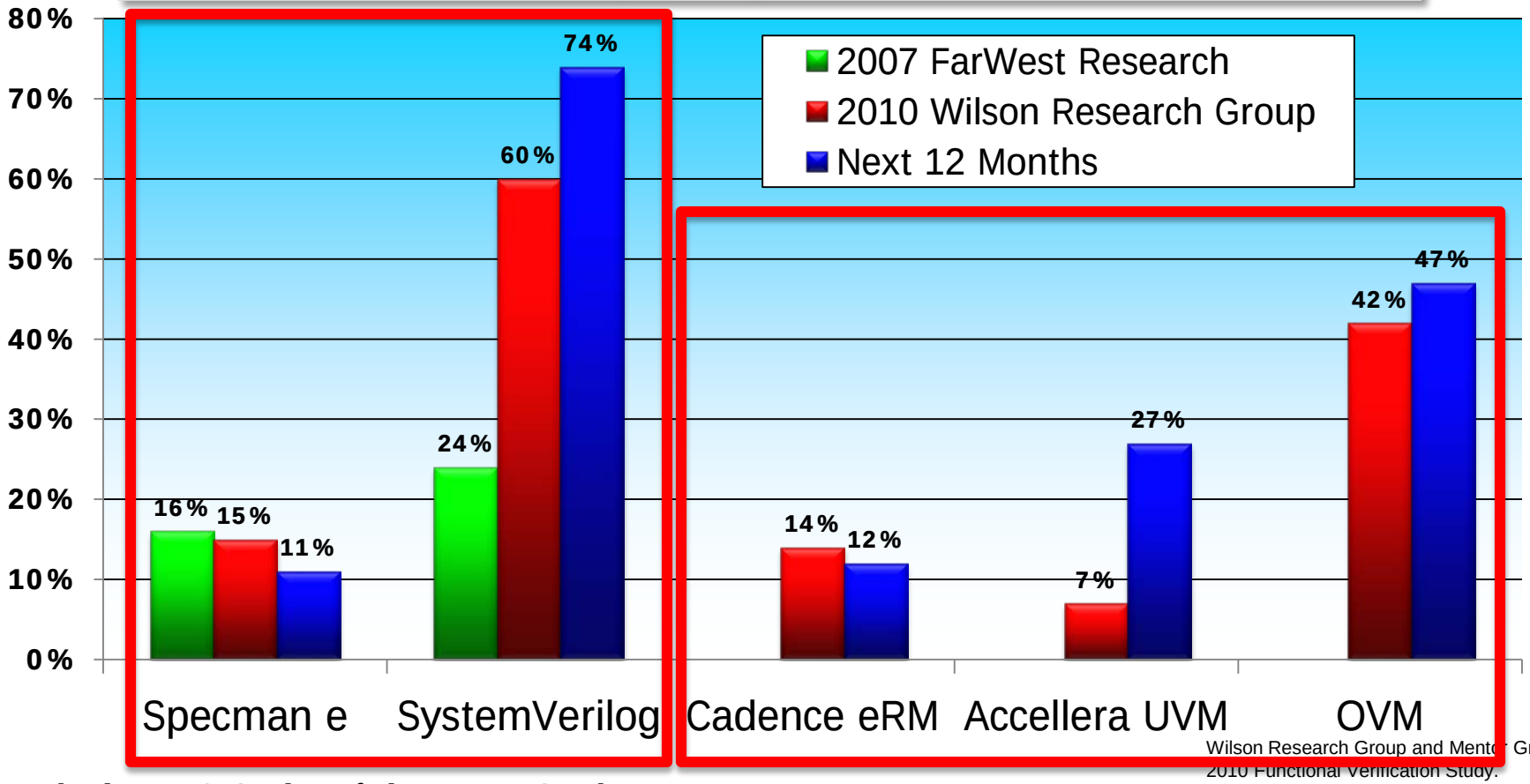
Mentor Graphics

Mentor Graphics®

Verification is Moving to SV/UVM

SystemVerilog adoption has increased by 233% in the past three years

UVM is expected to grow by 286% in the next 12 months



Wilson Research Group and Mentor Graphics
2010 Functional Verification Study.

What Now?

- Training?
- Pilot Project?
- Wholesale conversion?
- Peaceful Coexistance?

e/eRM versus SystemVerilog/UVM

- At a high level, an e/eRM testbench is the same as a SystemVerilog/UVM testbench
- At the low level, there are differences
  - Some are minor such as syntax
  - Others require a different way of thinking
- Our paper discusses both minor and major issues

≈

eRM Logo from

Similarities

=

- Hierarchy of units
- e/eRM ports connect units
- Transactions are basic data items
- Sequences create transactions and control what happens in the TB
- Virtual Sequences synchronize stimuli across interfaces

- Hierarchy of uvm_components
- TLM ports connect components
- Transactions are basic data items
- Sequences create transactions and control what happens in the TB
- Virtual Sequences synchronize stimuli across interfaces

Similarities

=

- Objections are used to control test termination
- compare() is used to compare transactions
- Flexible control of message output
- Tests are separated from the testbench

- Objections are used to control test termination
- compare() is used to compare transactions.
- Flexible control of message output
- Tests are separated from the testbench

Technical Differences

eRM Compliant ≈ UVM

- e_path() is used to get a unique path to each e unit
- deep_compare() is built-in
- get_enclosing_unit(), get_all_units(), etc are used to find units in hierarchy
- Randomization is automatic and pervasive

- get_full_name() is used to get the UVM defined path to each component
- Comparisons must be done recursively
- find() and find_all() can be used to find components in hierarchy
- Randomization is explicit and requires user action

More Thought Required

≠

- Aspect Oriented Programming (AOP)
- "when" inheritance
- Reflection
- Memory allocation during randomization/generation
- Testbench connection is via signals

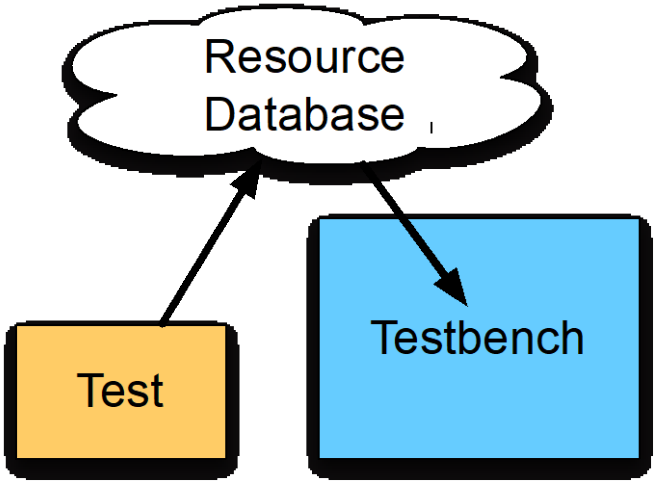
- Object Oriented Programming (OOP)
- Parent/child inheritance
- Field automation macros
- Separate memory allocation from randomization
- Testbench connection is via virtual interfaces

Aspect Oriented Programming

- Built-in to e language
  - Allows for extension or override of just about anything in an e based testbench
  - Used to keep test separate from testbench
- SystemVerilog doesn't support AOP
- UVM adds AOP-like features
  - UVM Resource Database (Configuration Space)
  - UVM Factory

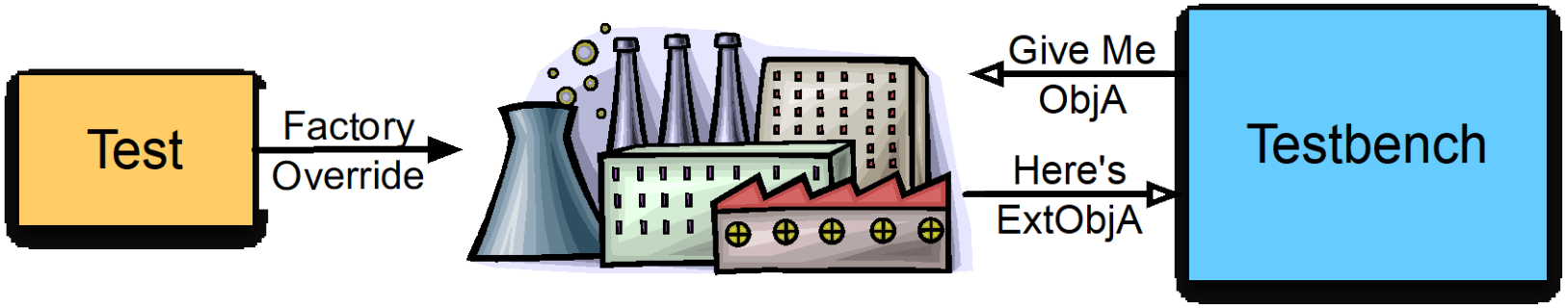
UVM Resource Database

- Allows tests to place information into a central repository to be pulled out by the testbench
- Allows test to control “What” happens in the testbench without changing any testbench code



UVM Factory

- Allows tests to override a class type in a testbench without changing any testbench code
- Must be polymorphically compatible
- Gives substantially the same flexibility as AOP in e



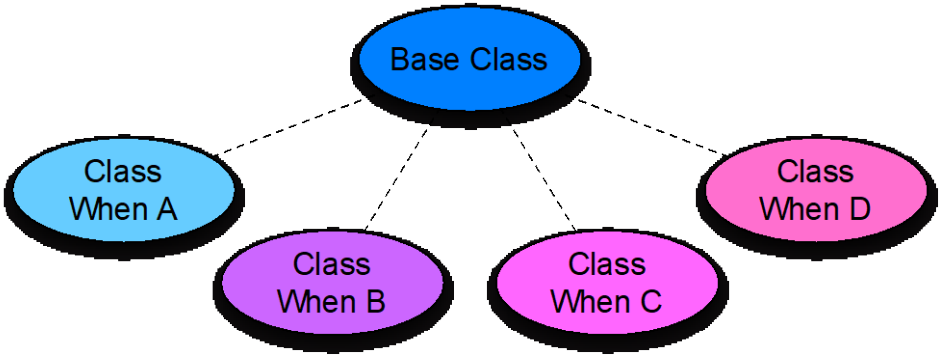
- Can add new data members, change method behavior, add constraints, etc.

"When" Inheritance

- Built-in to e language
  - Allows e users to add fields, constraints, etc. based on an enumerated value in a type
  - Used extensively in eRM testbenches
- SystemVerilog uses two techniques to perform the same function as "when" inheritance
  - Multi-class
    - Define a class for every "when" type
  - Single class
    - Define a class which is the aggregate of all the "when" types

Multi-Class

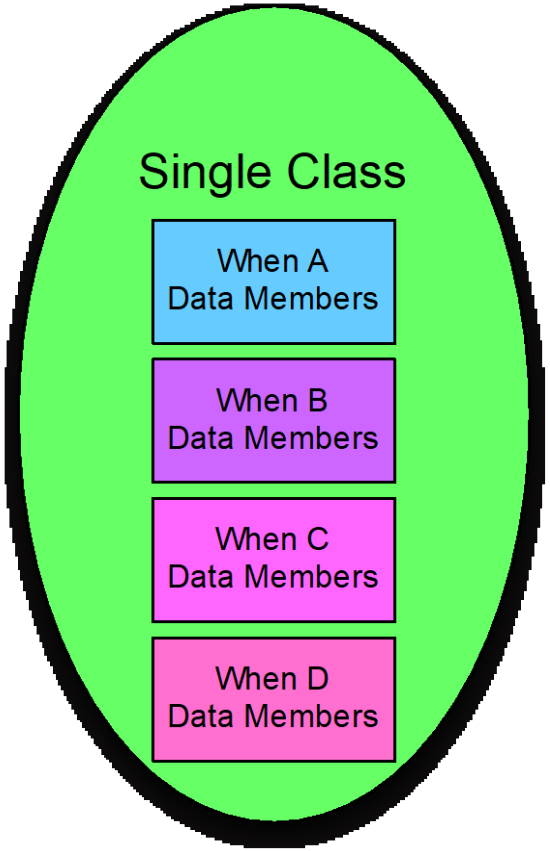
- Every “when” sub-type becomes a new class type



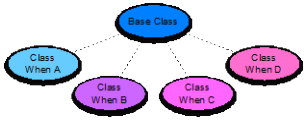
- Some limitations
  - Data types shared between a subset of subtypes will have to duplicated
  - Randomization must be broken down into two steps to generate a list of random subtypes
- Used in UVM for sequences and UVM Register Layer

Single Class

- All subtypes merged into one class
- Create sub-classes for each sub-type data members
  - Instantiate within single class
  - Null out irrelevant object handles

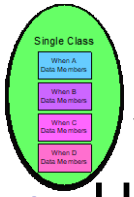


Multi-Class versus Single Class



Multi-Class

- Default choice
- Provides easier debug
- Better performance
- Becomes unworkable when multiple “when” subtypes are matrixed



Single Class

- Use when multiple “when” subtypes matrix with each other
- Can use more memory
- Requires more upfront thought

More Thought Required

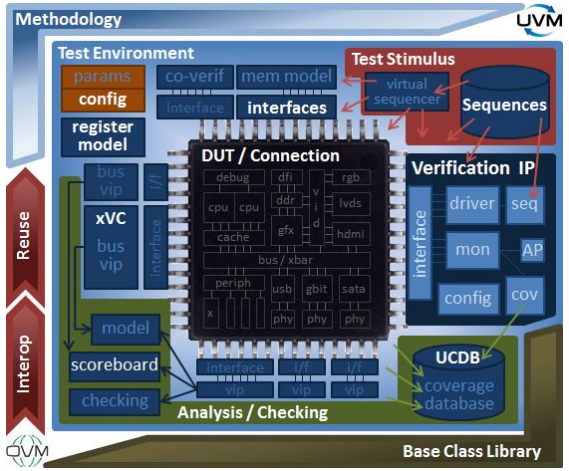
≠

- Aspect Oriented Programming (AOP)
- "when" inheritance
- Reflection
- Memory allocation during randomization/generation
- Testbench connection is via signals

- Object Oriented Programming (OOP)
- Parent/child inheritance
- Field automation macros
- Separate memory allocation from randomization
- Testbench connection is via virtual interfaces

Migration Kit

- Packages and examples to help ease a transition
  - AOP, “when” inheritance, randomization/generation, testbench connection, etc. examples
- Available for download
  - http://verificationacademy.com/forum/uvmovm-kit-downloads-and-user-contributions-forum/kit-downloads-and-user-contributions/24118-erm-uvm-migration-kit
  - http://tinyurl.com/eRM-To-UVM



Conclusion

- The industry is moving to SystemVerilog and UVM
- e/eRM have many similarities with SV/UVM
- Differences can be managed with the knowledge contained within our paper

Acknowledgments

- Avidan Efody – Verification Technologist - Mentor Graphics
- Geoff Koch – Technical Writer - Mentor Graphics
- Verification Methodology Team – Mentor Graphics