2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Introducing **IEEE 1800.2** The Next Step for UVM

Thomas Alsop - Intel Corp.
Srivatsa Vasudevan - Synopsys, Inc.
Mark Glasser - NVIDIA Corporation
Srinivasan Venkataramanan - CVC Pvt., Ltd.
Krishna Thottempudi - Qualcomm, Inc.

accellera SYSTEMS INITIATIVE

Tutorial Introductions

- Introduction to UVM – *Tom Alsop*
- 1800.2 UVM – *Srivatsa Vasudevan*
  - Changes from Accellera UVM, additions, clarifications
  - Details on compatibility issues for engineers updating VIP to 1800.2
- 1800.2 UVM – *Mark Glasser*
  - TLM clean-up, register package clean-up
- UVM for Designers – *Srini Venkataramanan and Krishna Thottempudi*

Agenda

- Brief history leading up to 1800.2
- Why standardize a verification methodology?
- UVM milestones
- UVM ownership
- What changed in UVM for the IEEE standard?
- Post IEEE 1800.2 efforts
- Tutorial intros
- Backward compatibility discussion
- Q&A

Why Standardization Was Needed

- Verification Intellectual Property (VIP) growing exponentially
- SoC development begins to dominate the industry
- IP-Reuse and sharing is an *absolute requirement*
- Sharing VIP requires interoperability of components

The industry could not afford huge amounts of time and resources consumed converting one form of VIP to another and in training on different methods used to create and use the VIP – i.e., major $$$ lost!!

- IEEE 1800 specification (SystemVerilog) is becoming the next language of choice for design and validation. It's standardized but there is still no standard for using its VIP.

Verification Libraries Evolve as Well

Pre-VM

SystemVerilog

AVM

eRM

OVM

VMM Central

UVM

Config	API Based	Path-Based Config	Resource Database
Comm	Direct Ref	TLM1	TLM2
Stimulus	API Based	Scenario/ Sequence	Sequence Efficiency
Phasing	Standard Phasing	Custom Phasing	Scheduler
RAL		Vendor RALs	Standard RAL
Report	Reporting API	Fast Report Macros	Report Object/Cbk

accellera SYSTEMS INITIATIVE

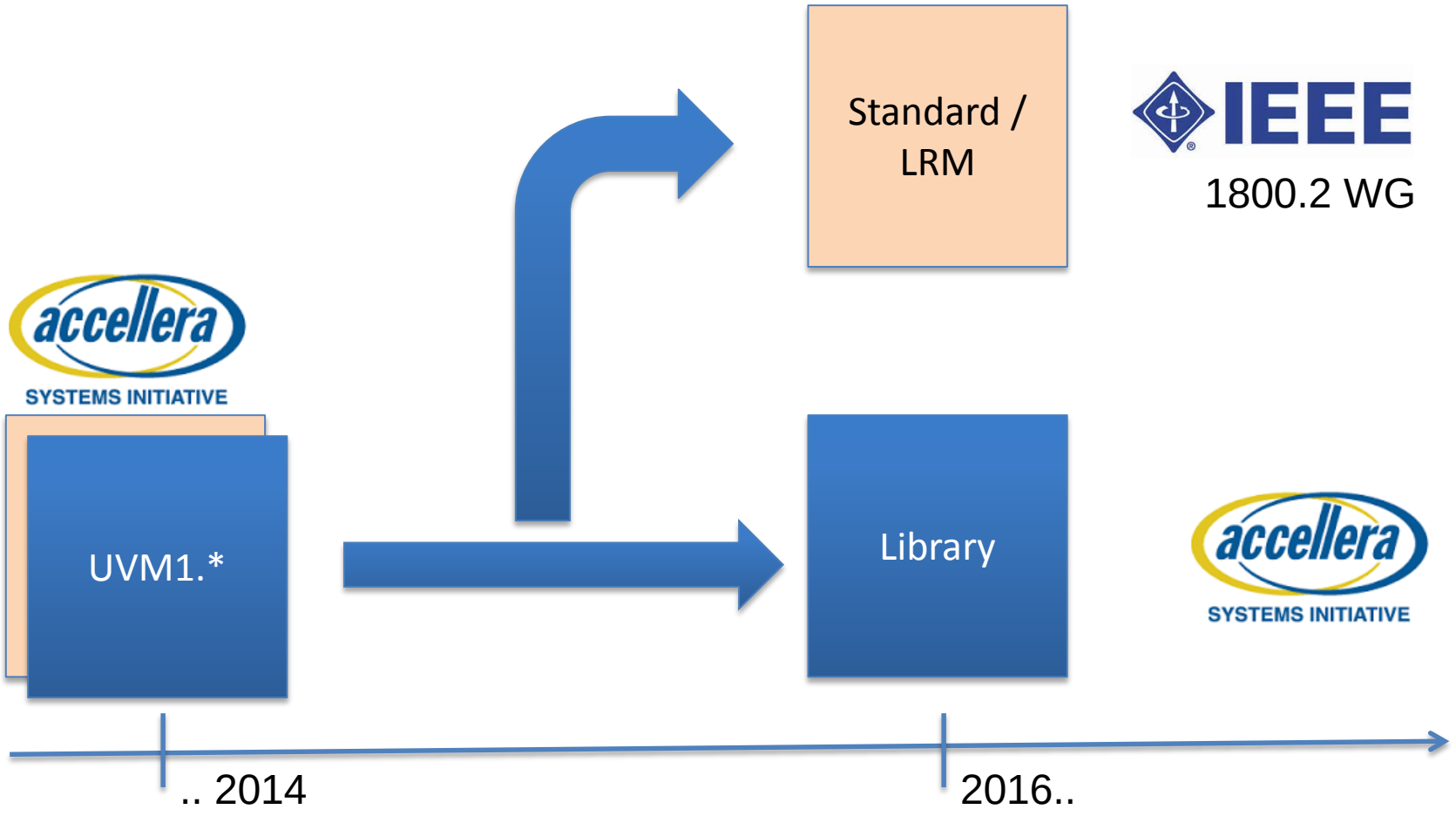
Standardization

- UVM had 3 major Accellera releases (1.0, 1.1, and 1.2)
- It has had many bug '**fix only**' releases
- It has been welcomed by JEITA (Asia/Japan) standards body
- IEEE PAR submitted and accepted by DASC in 2015
- UVM completed the IEEE standardization process in early 2017
  - Approved by IEEE SASB in February 2017
  - Will be published as IEEE 1800.2 Standard for UVM later in 2017
  - There is no other verification methodology in the IEEE
- **UVM is going to be around a very long time ☺**

UVM Milestones

Milestone	Date	Result	Key Points
OVM/VMM Interoperability	July 2009	Best Practices Document and Library for doing interoperability between OVM and VMM	• Simulators supporting multiple base classes • First joint library development on an Accellera TSC • Strong team (Collaboration, Speed, Quality, Best in Industry)
UVM 1.0 EA	May 2010	User & Reference Guide, BCL Library	• Starting point OVM 2.1.1 • Key features: callbacks, end-of-test and message catching • Documentation type selection • UVM/VMM interoperability library used for transitioning
UVM 1.0	February 2011	User & Reference Guide, BCL Library	• Registers, Phasing and TLM • Resources, Cmdline • Numerous other features • Infrastructure is being developed using the UVM 1.0 EA version
UVM 1.1 (a-d)	June 2011 to March 2013	User & Reference Guide, BCL Library	• Focus to stabilize and increase adoption • 60 high-priority bug fixes • Clean API spec with no errata • 1.1a – 1.1d all bug fixes to BCL over 18 month timeframe
UVM 1.2	June 2014	User & Reference Guide, BCL Library	• Phase-aware Sequences • Message/Debug System Revamp • New API and semantic changes
IEEE 1800.2-2017	February 2017	IEEE specification	• Non-standard worthy (non-user) APIs dropped • Many accessors added for property/data interface • TLM and Registers minor improvements • Overall cleanup and clarity of ambiguities

Who Owns UVM?


Standard / LRM
IEEE
1800.2 WG
accellera
SYSTEMS INITIATIVE
UVM1.*
Library
accellera
SYSTEMS INITIATIVE
.. 2014
2016..

UVM Ownership

- IEEE 1800.2 WG
  - 1800.2 = SV focused
  - Defines UVM functional API (not an implementation)
  - Produces 1800.2 LRM
  - Allows for multiple implementations
- Accellera WG
  - Delivers UVM Library (SV) Reference Implementation matching 1800.2 LRM
  - Provide bug fixes for UVM library

IEEE 1800.2 – What Changed?

- APIs were dropped
  - Non-user APIs – only used by the BCL, not by end users
  - Non-standard APIs – example or reference APIs, but not needed for standardization
  - Examples: DAP, uvm_comparator, uvm_utils, etc…
- New APIs were added
  - New accessors
  - Extendibility – things were added to allow debug to attach to them, or to extend functionality that was previously unavailable
  - Examples: uvm_init, uvm_runtest_callback, etc.
- Minor TLM cleanup – renamed as UVM to avoid scoping issues
- Register improvements
- “User Guide” material removed. It’s not standard-worthy.
- Overall upgrade to remove ambiguities and clarify the specification

Post IEEE 1800.2 Efforts

- Accellera UVM WG owns the implementation
- Restarted in Dec'16
- Discussions ongoing concerning effort
  - Very deep discussions WRT to backward compatibility
  - Migration options from previous versions
  - Scoping out the effort, detailing changes, timelines
- The Accellera effort will ultimately determine:
  - What types of changes do users need to make to ensure backwards compatibility?
  - What is the overall guidance? What needs to change?
  - Once you make the changes with the VIP, what are the implications?
  - Can it be used in the old ENV?
  - Does it have to be used with a new UVM IEEE/Accellera implementation release?

Contribution Options

- Accellera WG
  - For Accellera members – Thurs @ 9AM PST call
  - Tracking via accellera.mantishub.com → UVM
- IEEE 1800.2 WG (**Work completed late 2016**)
  - For IEEE members – Every other week @ 9AM PST call
  - Tracking via accellera.mantishub.com → P1800.2

IEEE and Backward Compatibility

- Many meetings discussing backward compatibility
- The issues were broken out into 5 categories consisting of 8 specific questions:
  - BCL compliance to the IEEE 1800.2 spec
  - Implementation artifacts & additive but non-IEEE APIs
  - Deprecation policy and roadmap
  - Removal of pre-1.2 deprecated code
  - APIs that changed from 1.2 to IEEE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Thank You!

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Changes from Accellera UVM, Additions & Clarifications

Srivatsa Vasudevan

on behalf of UVM Working Group

accellera SYSTEMS INITIATIVE

Agenda

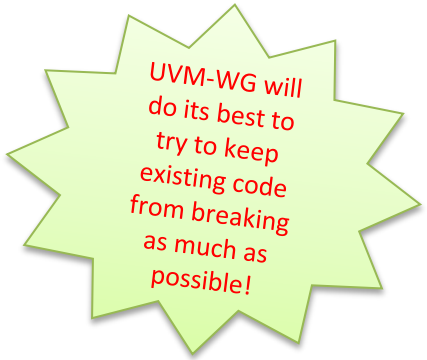
- Highlighting P1800.2 enhancements over UVM 1.2
- IEEE UVM effort overview
- Policy for Copy/Compare/Print/Record
- Factory enhancements
- Component enhancements
- Callback enhancements
- Other minor enhancements/changes

IEEE UVM Effort

- Effort made by UVM WG to ensure backward compatibility
- Focus on making the library easier to use/maintain/enhance
  - Accessor methods instead of fields
  - Removing implementation artifacts to not over-constraint future implementations
  - Provide more flexibility
    - No need to use macros for some functionality
    - Factory enablement of classes
  - Making API more uniform
- Documented API for classes and methods
  - Which were present in 1.2 but not in the documentation
  - Allow anyone to build an UVM library
  - There are no “secret sauce” methods. Everything uses documented API.
- Debug hooks are NOT mentioned in the IEEE LRM
  - More efficient methods can be provided by vendors/others

UVM WG Reference Implementation

- Reference implementation from UVM WG
  - Contain the UVM IEEE API with classes & methods
  - Contain additional useful API not in P1800.2
  - Will remove deprecated code that is pre- UVM 1.2
- Users may use linting or other solutions to ensure that their code is compliant to UVM 1800.2 API



Icons Used in this Presentation

Method was present in UVM 1.2
Was not documented earlier, now documented in P1800.2
Usually an internal function

Approach taken is different from UVM 1.2

Make a note of this versus UVM 1.2 for compatibility

New in IEEE 1800.2

4 ⇨ 2

4 state variable in UVM 1.2 → 2 state variable in IEEE

Policy Classes

UVM Policy Classes

- Every operation (Copy/Compare/Print/Pack/Record) now has a policy class
- Uniformity across all operations
- virtual class uvm_policy – base class for all policies
- UVM 1.2 had the following policy classes
  - Printer – virtual class uvm_printer
  - Comparer – virtual class uvm_comparer
  - Packer – virtual class uvm_packer
  - Recorder – virtual class uvm_recorder
- P 1800.2 changes them a little
  - Printer – virtual class uvm_printer extends uvm_policy
  - Comparer – virtual class uvm_comparer extends uvm_policy
  - Packer – virtual class uvm_packer extends uvm_policy
  - Recorder – virtual class uvm_recorder extends uvm_policy
- Policy Classes have a state. You can flush the state of the policy in the middle of the simulation.

Extensible & factory enabled

NEW

uvm_policy

virtual class uvm_policy extends from uvm_object

- Generic extension mechanism allows the user to pass additional information
- Objects may use these extensions to alter their interactions with the policy
  - An object may use extensions to selectively filter some of its fields/methods
- User can simply apply a different printer or compare “policy” to change how an object is printed or compared
  - Factory enabled
  - Callback mechanism provided via do_execute_op() method
- A number of accessor methods provided
  - virtual function bit extension_exists(uvm_object_wrapper ext_type)
  - virtual function int unsigned get_active_object_depth()
  - virtual function uvm_object get_active_object()
  - virtual function uvm_object get_extension(uvm_object_wrapper ext_type)
  - virtual function uvm_object pop_active_object()
  - virtual function uvm_object set_extension(uvm_object extension)
  - virtual function void clear_extension(uvm_object_wrapper ext_type)
  - virtual function void clear_extensions()
  - virtual function void push_active_object(uvm_object obj)

UVM PACKER

- Class uvm_packer extends uvm_policy
  - uvm_packer used to add a null terminating byte `0` to bitstream after packing strings
  - Incompatible with streaming operators
- To support streaming requires the uvm_packer to pack/unpack strings in the same manner as bitstream operators

Example: You couldn't do this in UVM 1.2

Pack:

```
bits = {>>{ size, my_string with [0 +: size]}};
```

Unpack:

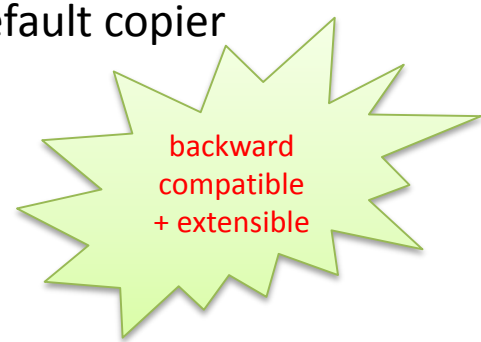
```
{>>{ size, my_string with [0 +: size]}} = bits;
```

- use_metadata is not mentioned in LRM. You can have a pack/unpack operation as long as the packer/unpacker understand each other.

uvm_copier

- Copy() operation did not have a policy before
- Signature of copy() has changed
  - function void copy (uvm_object rhs, **uvm_copier copier = null**)
- Class uvm_copier extends uvm_policy
  - Provides recursion state within the copier
  - Allows you to apply different recursion policies by setting default copier

static function uvm_copier get_default()
static function void set_default (uvm_copier copier)
virtual function uvm_recursion_policy_enum get_recursion_policy()
virtual function void copy_object (uvm_object lhs, uvm_object rhs)
virtual function void set_recursion_policy (uvm_recursion_policy_enum policy)



Example Copying in UVM P1800.2

What the policy classes buy you

```
class  class_B extends uvm_object;

  // basic datatypes
  rand int par_int;
  rand byte par_address;
  string par_string;

  // Some objects to demonstrate the copy recursion policy

  class_A cl1; // UVM_SHALLOW
  class_A cl3; // UVM_DEEP

…

function void do_copy(uvm_object rhs);
    class_B rhs_;
    uvm_copier copier;
    super.do_copy(rhs);
    $cast(rhs_,rhs);
    par_int = rhs_.par_int;
    par_address = rhs_.par_address;
    par_string = rhs_.par_string;
    $cast(copier, get_active_policy())
   copier.copy_object(this.cl1 _rhs.cl1);
endfunction

endclass
```

Method provided to not break backward compatibility of do_copy

Recursion policy is available

Capability /flexibility boost

1) You can add extensions
2) The copier class is also factory enabled
3) You could NOT do this before

uvm_comparer

- Virtual class comparer extends uvm_policy
- UVM 1.2 allowed you to set comparer variables directly
- P1800.2 provides you accessor methods
- Backward compatible

uvm_comparer

UVM 1.2

```
class custom_comparer extends uvm_comparer;
      int unsigned show_max = 1;
...
endclass
```

```
custom_comparer cc = new;
…
cc.show_max = 0;
```

P1800.2

```
custom_comparer cc = new;

…

 cc.set_show_max(0);
```

Accessor Methods

virtual function int unsigned get_show_max()
virtual function int unsigned get_threshold()
virtual function int unsigned get_verbosity()
virtual function string get_miscompares()
virtual function uvm_recursion_policy_enum get_recursion_policy()
virtual function uvm_severity get_severity()
virtual function void flush()
virtual function void set_check_type (bit enabled)
virtual function void set_recursion_policy (uvm_recursion_policy_enum policy)

virtual function void set_severity (uvm_severity severity)
virtual function void set_show_max (int unsigned show_max)
virtual function void set_threshold (int unsigned threshold)
virtual function void set_verbosity (int unsigned verbosity)

uvm_printer

- Virtual class uvm_printer extends uvm_policy
- Base class for other printer types
- Allows you to set the default printer
  - static function void set_default (uvm_printer printer)
  - static function uvm_printer get_default()
- Note that the default Printer Cannot be NULL in P1800.2
  - Change from UVM 1.2
- UVM Printer knobs is modified from UVM 1.2
  - Accessor methods are provided
  - Functionality is provided in the base class
  - Will be backward compatible
  - Relevant settings moved into each specific printer

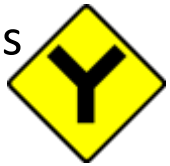
Replacement for UVM Printer Knobs

- **Knobs are broken up to have settings as per relevant printers**
- **A number of methods have been made a part of the uvm_printer class**
- **These are accessor methods for settings in UVM printer knobs**

virtual function void set_begin_elements (int elements = 5)
virtual function void set_default_radix (uvm_radix_enum radix)
virtual function void set_end_elements (int elements = 5)
virtual function void set_file (UVM_FILE fl)
virtual function void set_id_enabled (bit enabled)
virtual function void set_line_prefix (string prefix)
virtual function void set_max_depth (int depth)
virtual function void set_name_enabled (bit enabled)
virtual function void set_radix_enabled (bit enabled)
virtual function void set_radix_string (uvm_radix_enum radix, string prefix)
virtual function void set_recursion_policy (uvm_recursion_policy_enum policy)
virtual function void set_root_enabled (bit enabled)
virtual function void set_size_enabled (bit enabled)
virtual function void set_type_name_enabled (bit enabled)

uvm_line_printer

- Portion of the printer knobs relevant to the line printer are in this class
  - static function uvm_line_printer get_default()
  - static function void set_default (uvm_line_printer printer)
  - virtual function string get_separators()
  - virtual function void set_separators (string separators)

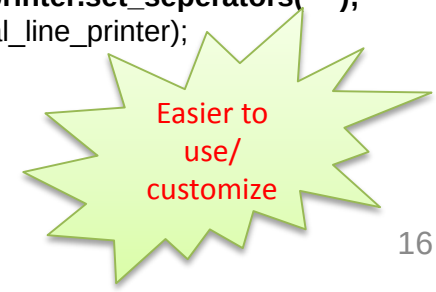


UVM 1.2

```
uvm_line_printer local_line_printer = new;

simple_packet pkt = new("simple_packet_live");

initial begin
pkt.randomize();
        pkt.print();
        uvm_default_printer = uvm_default_line_printer;
        pkt.print();
        local_line_printer.knobs.seperators = " ";
         pkt.print(local_line_printer);
    end
```

P1800.2

```
uvm_line_printer local_line_printer = new;

simple_packet pkt = new("simple_packet_live");

initial begin
pkt.randomize();
        pkt.print();
            local_line_printer.set_default();
            pkt.print();
           local_line_printer.set_seperators(" ");
           pkt.print(local_line_printer);
    end
```



uvm_table_printer

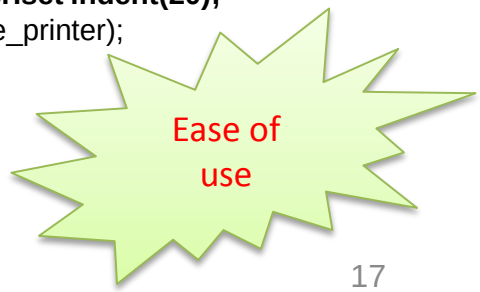
- Portion of the printer knobs relevant to the table printer are in this class
  - static function uvm_table_printer get_default()
    - static function void set_default (uvm_table_printer printer)
    - virtual function int get_indent()
    - virtual function void set_indent (int indent)

UVM 1.2

```
uvm_table_printer local_table_printer = new;

simple_packet pkt = new("simple_packet_live");

initial begin
pkt.randomize();
        pkt.print();
        uvm_default_printer = uvm_default_table_printer;
        pkt.print();
        local_table_printer.knobs.indent = 20;
        pkt.print(local_table_printer);
end
```

P1800.2


```
uvm_table_printer local_table_printer = new;

simple_packet pkt = new("simple_packet_live");

initial begin
        pkt.randomize();
        local_table_printer.set_default();
        pkt.print();
        local_table_printer.set indent(20);
        pkt.print(local_table_printer);
    end
```



uvm_tree_printer

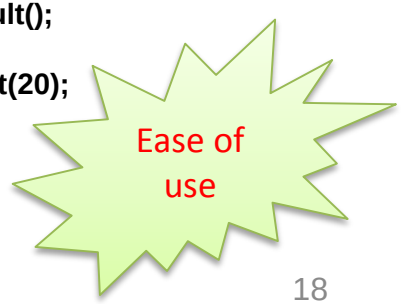
- Portion of the printer knobs relevant to the tree printer are in this class
  - static function void set_default (uvm_tree_printer printer)
  - virtual function int get_indent()
  - virtual function string get_separators()
  - virtual function void set_indent (int indent)
  - virtual function void set_separators (string separators)

UVM 1.2

```
uvm_tree_printer local_tree_printer = new;

simple_packet pkt = new("simple_packet_live");

initial begin
pkt.randomize();
        pkt.print();
        uvm_default_printer = uvm_default_tree_printer;
        pkt.print();
        local_tree_printer.knobs.indent = 20;
        pkt.print(local_tree_printer);
end
```

P1800.2

```
uvm_table_printer local_tree_printer = new;

simple_packet pkt = new("simple_packet_live");

initial begin
        pkt.randomize();
        local_tree_printer.set_default();
        pkt.print();
        local_tree_printer.set indent(20);
        pkt.print(local_tree_printer);
    end
```



uvm_recorder

- **P1800.2**
  - function int get_handle()
  - static function uvm_recorder get_recorder_from_handle(**int** id)
- **UVM 1.2**
  - function integer get_handle()
  - static function uvm_recorder get_recorder_from_handle(integer id)
- New Methods added
  - virtual function bit get_id_enabled()
  - virtual function int get_record_attribute_handle()
  - virtual function uvm_policy::recursion_state_e object_recorded(uvm_object value, uvm_recursion_policy_enum recursion)
  - virtual function uvm_radix_enum get_default_radix()
  - virtual function uvm_recursion_policy_enum get_recursion_policy()
  - virtual function void flush()
  - virtual function void set_default_radix (uvm_radix_enum radix)
  - virtual function void set_id_enabled (bit enabled)
  - virtual function void set_recursion_policy (uvm_recursion_policy_enum policy)

4 ⇨ 2 Changes

NEW

Essential Operations Summary

- UVM policy classes have gone through a revamp
  - Backward compatibility has been considered extensively
  - Accessors provided for uniformity
  - Classes use a policy
  - You can **not** set policy classes to null – Change from earlier behavior
  - There is a default to ensure backward compatibility
  - You can make changes in flight in a simulation. Things aren't hard-coded.
  - uvm_packer has reset
  - uvm_printer has flush
  - uvm_comparer has a flush
  - uvm_recorder has a free() method
- Field macros have documented methods
  - Most of the methods were "secret sauce" in earlier UVM versions
  - They now use Methods now documented in LRM
- UVM flags are documented
  - UVM_FLAGS is a type

UVM Factory

UVM Factory

- UVM additionally supports registration of abstract objects and components with the factory
- Two new classes to support this functionality
  - class uvm_abstract_object_registry #(type T=uvm_object, string Tname="unknown>") extends uvm_object_wrapper
  - class uvm_abstract_component_registry #(type T=uvm_component, string Tname="unknown>")

UVM Factory

Added

- pure virtual function void set_type_alias(string alias_type_name, uvm_object_wrapper original_type)
- pure virtual function void set_inst_alias(string alias_type_name, uvm_object_wrapper original_type, string full_inst_path);

Documented

- pure virtual function uvm_object_wrapper find_wrapper_by_name (string type_name)
- virtual function bit is_type_name_registered (string type_name);
- virtual function bit is_type_registered (uvm_object_wrapper obj);
- pure virtual function uvm_object_wrapper find_wrapper_by_name (string type_name)

UVM Factory

- Many classes in UVM were not factory enabled in 1.2
- Shortcoming addressed in P1800.2
  - All classes derived from uvm_object are factory enabled unless otherwise explicitly mentioned in LRM.
    - Remember that since many more objects are registered with the factory, you may see more class types/overrides than UVM 1.2

- Ability to have abstract classes
- Additional API

UVM Component

uvm_component

function apply_config_settings() implicitly called during the build phase

No way to turn it off

P1800.2 introduces a new method

virtual function bit use_automatic_config()

- Default implementation returns 1
- If you wish to disable the automatic call to **apply_config_settings()** this method needs to be overloaded to return a 0 in derived classes.

UVM Object

uvm_object

function bit compare (uvm_object rhs, **uvm_comparer comparer=null**)

function void copy (uvm_object rhs, **uvm_copier copier = null**)

function bit compare (uvm_object rhs, **uvm_comparer comparer = null**)

Will get default policies In P1800.2 are not NULL

New Methods

static function bit get_uvm_seeding()

static function void set_uvm_seeding (bit enable)

All objects extended from uvm_object are factory enabled by default.

Base Classes

uvm_transaction

4 ➡ 2 Changes

1800.2

function int begin_child_tr (time begin_time = 0, int parent_handle = 0)
function int begin_tr (time begin_time = 0)
function int get_transaction_id()
function int get_tr_handle()

UVM 1.2

function **integer** begin_child_tr (time begin_time = 0, **integer** parent_handle = 0)
function **integer** begin_tr (time begin_time = 0)
function **integer** get_transaction_id()
function **integer** get_tr_handle()

UVM Comparator

- Not mentioned in P1800.2
  - uvm_comparator
  - uvm_algorithmic comparator
  - uvm_in_order_comparator

UVM Reporting

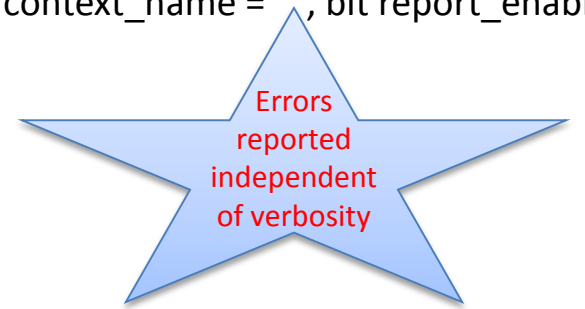
uvm_report_object

UVM 1.2

virtual function void uvm_report_error(string id, string message, int verbosity = UVM_LOW, string filename = "", int line = 0, string context_name = "", bit report_enabled_checked = 0)

P1800.2

virtual function void uvm_report_error(string id, string message**, int verbosity = UVM_NONE,** s filename = "", int line = 0, string context_name = "", bit report_enabled_checked = 0)



Changed:

virtual function void uvm_report(uvm_severity severity, string id, string message, int verbosity = (severity == uvm_severity'(UVM_ERROR)) ? **UVM_NONE** : (severity == uvm_severity'(UVM_FATAL)) ? **UVM_NONE** : UVM_MEDIUM,string filename = "", int line = 0, string context_name = "", bit report_enabled_checked = 0)

uvm_report_object

uvm_action is of a specific type – was defined as int in 1.2

Methods impacted:

function void set_report_severity_action (uvm_severity severity, uvm_action action)

function void set_report_id_action (string id, uvm_action action)

function void set_report_severity_id_action (uvm_severity severity, string id, uvm_action action)

uvm_report_server

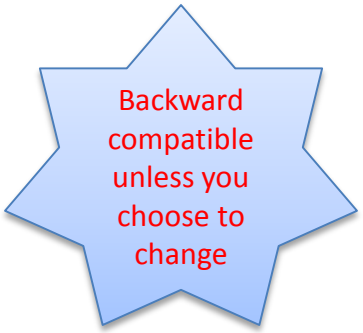
- UVM 1.2

pure virtual function void report_summarize(UVM_FILE file = 0)

- P1800.2

UVM_FILE is a type in P1800.2

pure virtual function void report_summarize(UVM_FILE file = UVM_STDOUT)



uvm_report_catcher

pure virtual function action_e catch() returns an enum, not an int in P1800.2

- Debug functionality is not mentioned in P1800.2
- UVM WG will provide an compatible implementation
- Users/Vendors can provide their own extensions for debug if needed

UVM Callbacks

Callbacks

- Callback classes in P1800 now extend from uvm_callback
  - uvm_callback extends from **uvm_object**.
- **UVM 1.2**
  - virtual class uvm_event_callback#(type T = uvm_object) extends uvm_object
- **P1800.2**
  - virtual class uvm_event_callback#(type T = uvm_object) extends **uvm_callback**
- Not documented in UVM 1.2; now documented in P1800.2
  - class uvm_callback_iter#(type T = uvm_object, type CB = uvm_callback)
    - function CB first()
    - function CB get_cb()
    - function CB last()
    - function CB next()
    - function CB prev()
    - function new(T obj)

Sequence Macros

- Many `uvm_do macros in UVM 1.2
- Macro is redefined in P1800.2

  `uvm_do(SEQ_OR_ITEM, SEQR=get_sequencer(), PRIORITY=-1, CONSTRAINTS={})

UVM WG Reference Implementation

- Reference implementation from UVM WG
  - Contain the UVM IEEE API with classes & methods
  - Contain additional useful API not in P1800.2
  - Will remove deprecated code that is pre-UVM 1.2
  - Code that is semantically incompatible with UVM 1.2
- Users can use linting or other solutions to ensure that their code is compliant to UVM 1800.2 API

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Thank You!

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

TLM and Register Update for IEEE 1800.2

Mark Glasser, NVIDIA

on behalf of UVM Working Group

accellera SYSTEMS INITIATIVE

TLM

- Transaction-level Modeling Library in UVM
- Collection of interfaces and channels for communicating transactions
- Based on IEEE-1666 SystemC/TLM standard

TLM Mantis Items

ID	Summary
5391	uvm_tlm_time::decr() missing default value for parameter
5613	D3 section 12.1 Overview needs to be rewritten
5591	Explicit methods definition instead of using uvm_*_port/export/imp
5619	uvm_tlm_*_socket needs clarification on attributes and allowable connectivity
5590	Missing APIs for UVM TLM1
5588	Class uvm_tlm_if_base #(T1,T2) missing
5496	get_peek_export gives reference to get more info, but no info is there
5392	Master Mantis for TLM issues
5592	TLM naming in LRM
5589	Name inconsistencies between UVM TLM1 and SystemC TLM-1
5586	Move uvm_sqr_if_base #(REQ,RSP) from TLM to sequencer section

TLM Consistency

- Make method names consistent with SystemC TLM
  - Maintain consistency?
  - Or… break backward compatibility
- Decision: not break backward compatibility

TLM Naming

- “UVM TLM”
- Disconnect UVM implementation from SystemC implementation
- Cannot have two things called “TLM” as IEEE standards
- Maintain provenance

Registers

Focus of UVMREG WG

- Review and address current limitations and use model gaps
  - System level and/or dynamic address map applications
  - Limitations or missing capabilities with current use models
- Align uvm_reg with other standards (e.g., IP-XACT)
- Review and improve documentation
- Backward compatibility

IEEE 1800.2 Changes

- No major change affecting backward compatibility
- LRM/Documentation enhanced and clarified
- Addressed system level/dynamic use models
- Further alignment with IP-XACT deferred for backward compatibility reasons

Reg Model Unlock

- Allows to unlock and re-lock a ‘locked’ register model
- Allows to restructure and rebuild addressing hierarchy
- Minimal infrastructure to accomplish system level/dynamic scenarios
  - Changing addressing/visibility at runtime
  - Additional and compatible to current infrastructure

Registers in the Factory

	Class Name	P1800.2 Virtual	UVM 1.2
12	uvm_reg_block	No	Yes
13	uvm_reg	No	Yes
14	uvm_reg_file	No	Yes
16	uvm_vreg_field_cbs	Yes	No
17	uvm_reg_backdoor	Yes	No
21	uvm_reg_read_only_cbs	Yes	No
23	uvm_reg_write_only_cbs	Yes	No

- Classes that were formerly virtual are no longer
- `uvm_object_utils added to these classes
- These classes are now “factory ready”

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Thank You!

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

UVM for RTL Designers

Srinivasan Venkataramanan, VerifWorks

Krishna Thottempudi, Qualcomm

QUALCOMM®

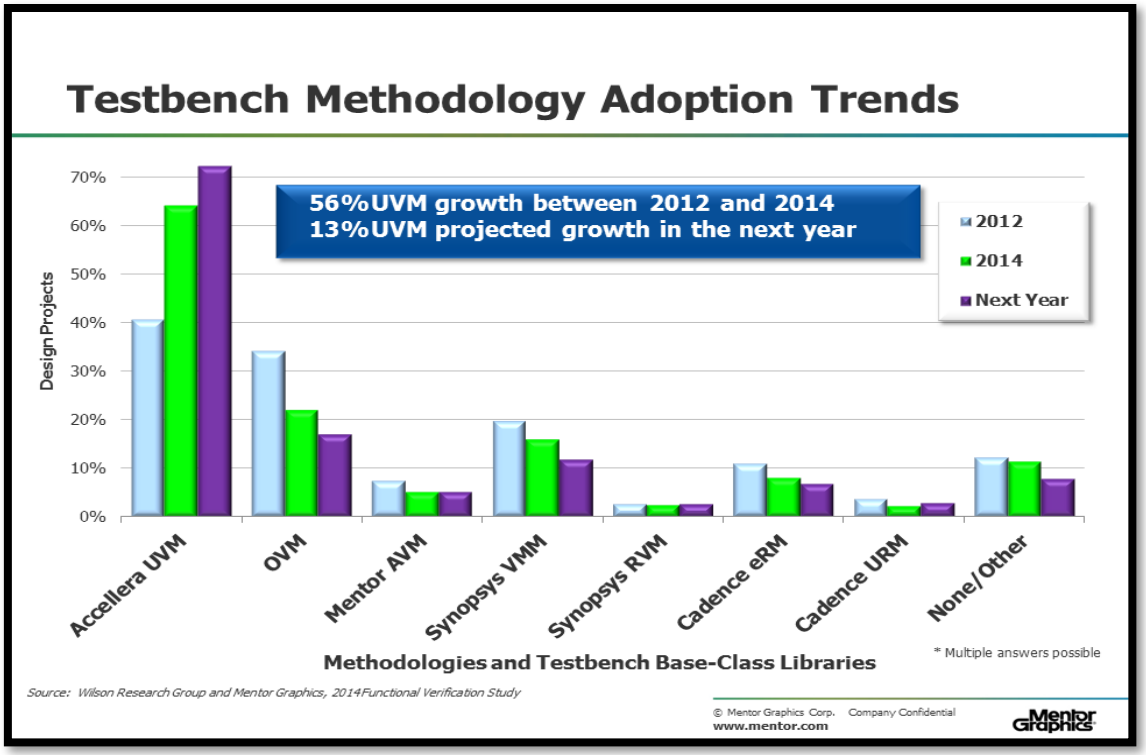
accellera SYSTEMS INITIATIVE

Agenda

- UVM Journey so far
  - Typical RTL designer's verification requirements
- Go2UVM test layer
- Signal Access API(Force/Deposit/Release)
- Register verification
  - Simple approach
  - VIP approach
- Testing Reset propagation & clock controller
- Waves2UVM
- Bridging RTL Designers' UVM to Verification team's UVM

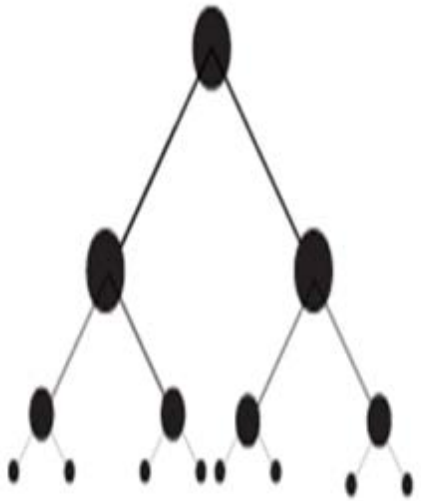
UVM – Fastest Growing Methodology

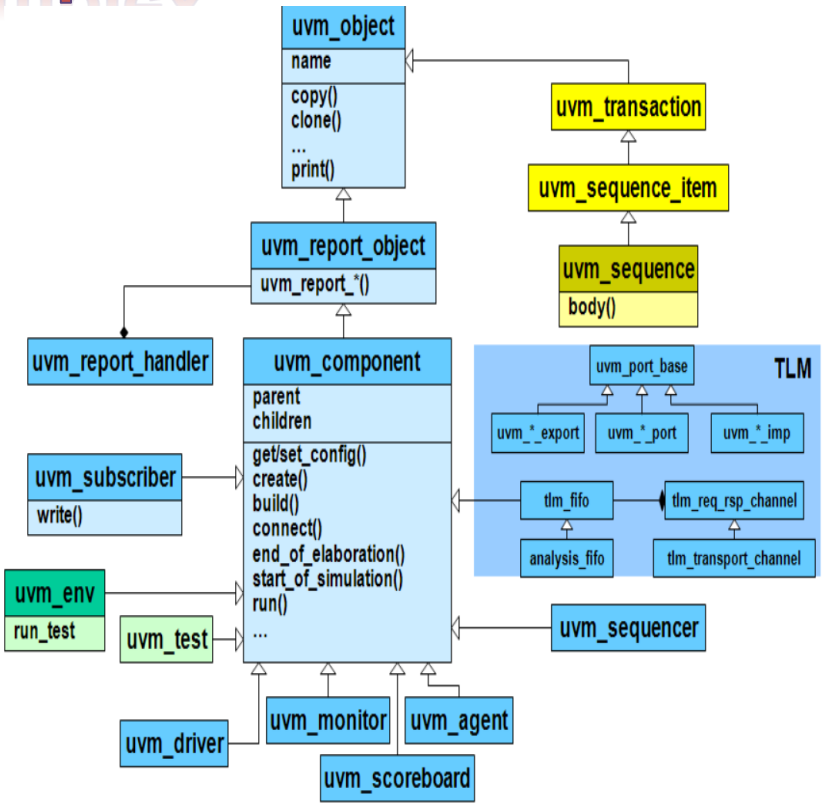
- Source: Independent survey by Wilson group
  - Sponsored by Mentor Graphics



UVM is Complex, But…

Complex

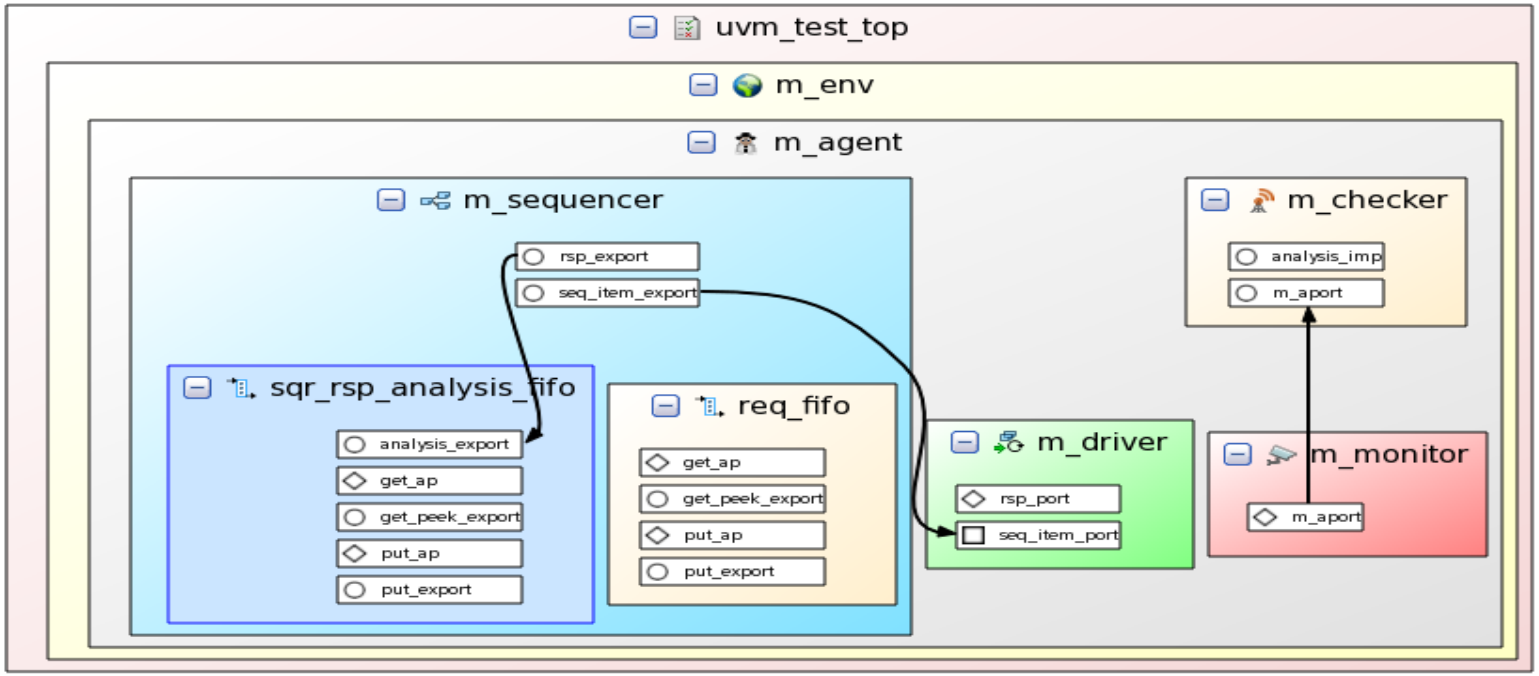


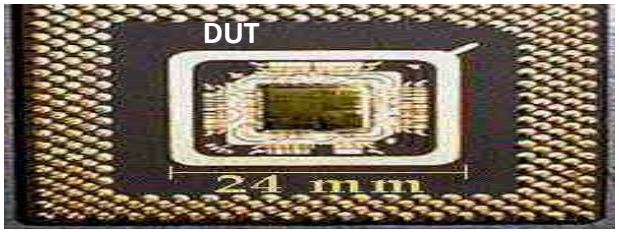


Need Not Be Complicated

Need a way to simplify for wider adoption

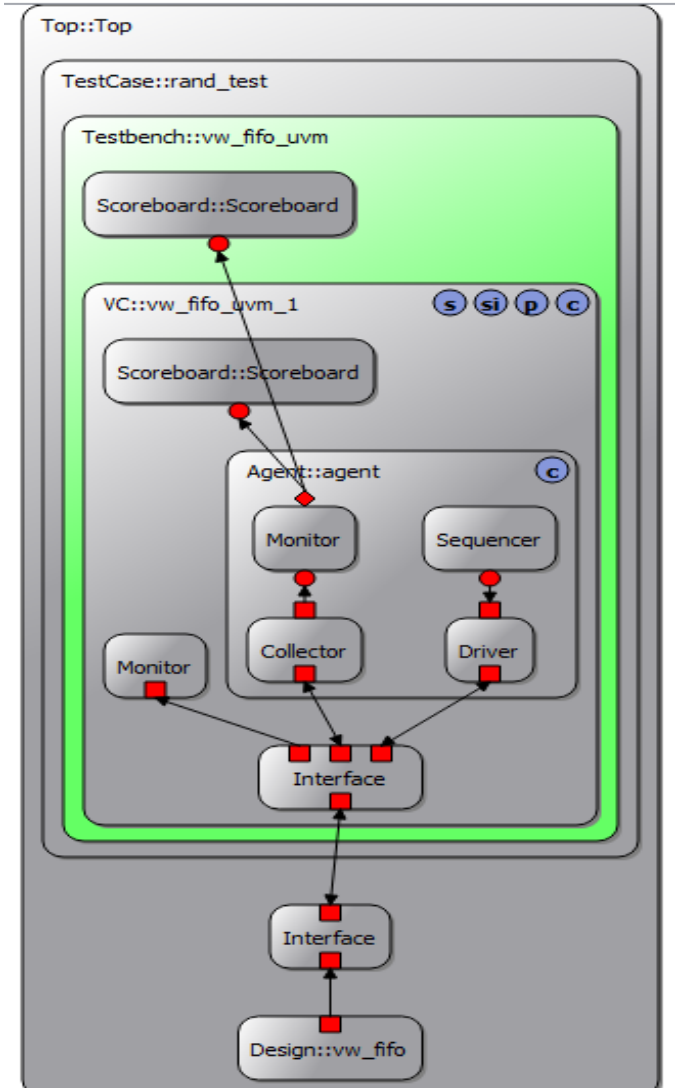
Typical UVM Architecture





UVM Mechanics

- Multiple layers of components
- Hierarchical component hook-ups
  - uvm_component::new (string name, uvm_component parent)
- UVM macros to automate lot of features
- Phasing (reset_phase, main_phase, run_phase, etc.)
- Objection mechanism (a must in UVM to get even a simple stimulus through to the DUT)



Need to Simplify UVM

- Not all engineers are familiar with full UVM
- RTL designers need simple tests
  - Simple wire wiggling
  - Force/Deposit/Release for corner cases
  - Register access tests using UVM reg-model
- Several auto-generated tests need a simple testing framework
  - Direct wire toggling
  - Waves2UVM, etc., DFT

Verilog's $display

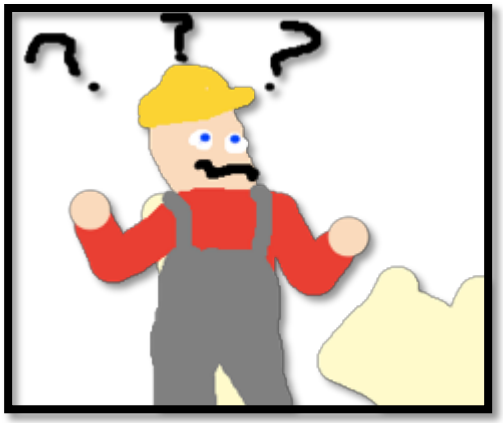
- Standard messaging in Verilog
- Pros:
  - Most used debug technique
  - Simple usage, familiar to all
- Cons:
  - No built-in $time
  - No file/line information
  - No control over verbosity
  - No severity information

```
`$display ("Hello World!")
```

```
$display ("%t %s %0d
  Driven addr: 0x%0h
       data: 0x%0h",
   $time,
   `__FILE__,
   `__LINE__,
   x0.adddr, x0.data)
```

Standard UVM Messaging

- UVM provides functions & macros
  - `uvm_info (ID, MSG, VERBOSITY)
  - `uvm_error (ID, MSG)
- Pros:
  - Versatile, powerful
  - Flexible formatting
- Cons:
  - First timers don't appreciate ID usage
  - Seasoned users ID → get_name()
  - VERBOSITY → Default value would be handy!



Simplified UVM Display

- `g2u_display → Simplified macros around standard UVM messaging API
- `g2u_display(MSG, VERBOSITY=UVM_MEDIUM) == `uvm_info (get_name(), MSG, VERBOSITY)

```
`g2u_display ("Hello World!")
```

ID = <default>, VERBOSITY = UVM_MEDIUM

```
`g2u_display ("Found a match!",
              UVM_HIGH)YY
```

ID = <default>, VERBOSITY = UVM_HIGH

```
`g2u_display ($sformatf("Driven
        addr: 0x%0h data: 0x%0h",
   x0.adddr, x0.data)
```

ID = <default>, VERBOSITY = UVM_MEDIUM

What is *Go2UVM*?

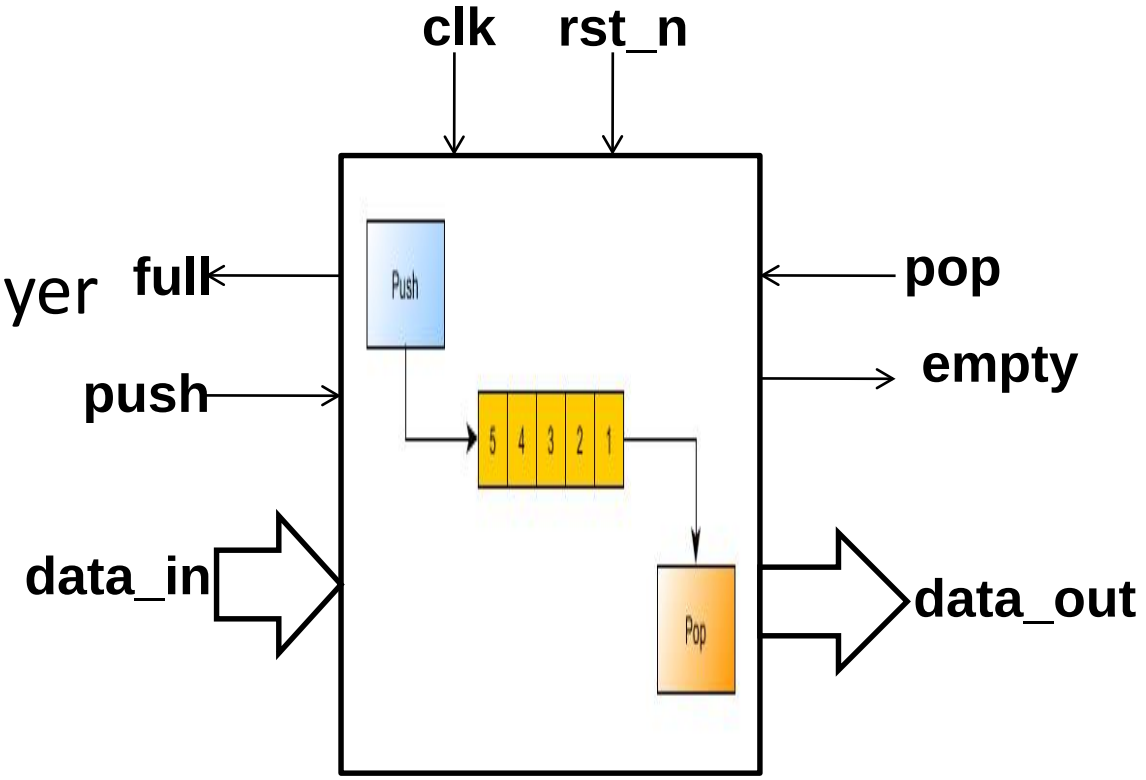
- SystemVerilog package
- TCL “apps” to auto-create Go2UVM files
- Package on top of standard UVM framework
  - A minimal subset
- Messaging same as `uvm_info
  - Simplified versions available as well
- Test from *uvm_test* base class
- Phasing (Active)
  - Main Phase – mandatory
  - Reset Phase – optional, recommended
- Objections mechanism
  - Automated, users don’t have to bother
- Test PASS/FAIL declaration
  - Automated

What's Inside VW_Go2UVM Package?

- Wrapper around UVM test layer
- Hides phasing completely from users
  - Uses ***main_phase*** – mandatory (User fills ***main()***)
  - ***reset_phase*** optional (User fills ***reset()***)
- Leverages on "pure virtual" to guard against misuse
- Internally handles objections
  - A must in UVM
  - Big time saver for first time UVM users

Simple FIFO Design

- Sanity checks by RTL engineers
- Provide a UVM test layer with task-based APIs
  - ***task reset()***
  - ***task main()***
- Go2UVM test layer
  - Hides complexity
  - Skips several layers



Go2UVM Test for a FIFO

```
import uvm_pkg::*;
`include "vw_go2uvm_macros.svh"
// Import Go2UVM Package
   import vw_go2uvm_pkg::*;
// Use the base class provided by the
   vw_go2uvm_pkg
   `G2U_TEST_BEGIN(fifo_test)
// Create a handle to the actual interface
 `G2U_GET_VIF(fifo_if)
  task reset();
    `g2u_display ("Start of reset")
     this.vif.cb.rst_n <= 1'b0;
     repeat (5)@(this.vif.cb);
     this.vif.cb.rst_n <= 1'b1;
     repeat (1)@(this.vif.cb);
    `g2u_display ("End of reset")
  endtask : reset
```

Test

SV Interface

reset task

main task

```
  task main ();
    `g2u_display ("Start of main")
     this.vif.cb.push<= 1'b1;
     this.vif.cb.data_in <= 22;
     @(vif.cb);
     vif.cb.push <= 1'b0;
     repeat(5)@(this.vif.cb);
    `g2u_display ("End of main")
  endtask : main
`G2U_TEST_END
```

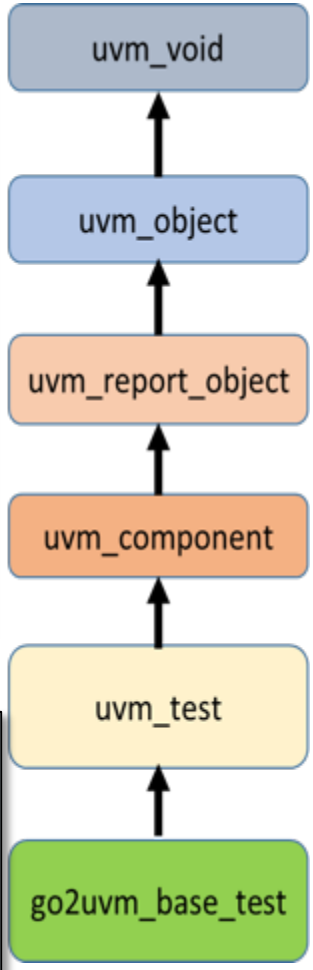
Inside Go2UVM

```
virtual class go2uvm_base_test extends uvm_test;
  extern virtual function void end_of_elaboration_phase(uvm_phase
phase);
  extern virtual task reset_phase (uvm_phase phase);
  extern virtual task reset();
  extern virtual task main_phase(uvm_phase phase);
  pure virtual task main();
  extern virtual function void report_header(UVM_FILE file= 0);
  extern function void report_phase(uvm_phase phase);

  string vw_run_log_fname = "vw_go2uvm_run.log“;
  UVM_FILE vw_log_f;
```

Go2UVM base test

```
task go2uvm_base_test::main_phase(uvm_phase phase);
  phase.raise_objection(this);
  `vw_uvm_info(log_id,"Driving stimulus via UVM",UVM_MEDIUM)
   this.main();
  `vw_uvm_info(log_id,"End of stimulus",UVM_MEDIUM)
  phase.drop_objection(this);
endtask:main_phase
```

Go2UVM main phase



Hookup – DUT, Interface & UVM

- Very similar to Verilog flow
  - Create module tb_top
  - Instantiate DUT
  - Instantiate interface
  - Generate clock
- Go2UVM
  - Instantiate *UVM test*
  - Call run_test ()

```
//Interface instance
fifo_if fifo_if_0 (.*);

  initial begin : g2u_test
// Connect virtual interface to
// physical interface
`G2U_SET_VIF(fifo_if,fifo_if_0)

// Kick start standard UVM
//phasing
   run_test ();
 end : g2u_test
endmodule : go2uvm_fifo
```

- All of the above is automated via TCL “apps”

VW_Go2UVM_pkg – Verilog vs. UVM Stimulus

Verilog

```
task drive();
  int i;
  $display("start of Test");
  @(posedge clk);
  load <= 1'b1;

  repeat(2) @(posedge clk);
  load <= 1'b1;
  data <= 8'h78;

  repeat(2) @(posedge clk);
  load <= 1'b1;
  cen  <= 1'b1;
  up_dn<= 1'b1;

  repeat(3) @(posedge clk);
  load <= 1'b1;
  cen  <= 1'b1;
  up_dn<= 1'b0;
```

UVM

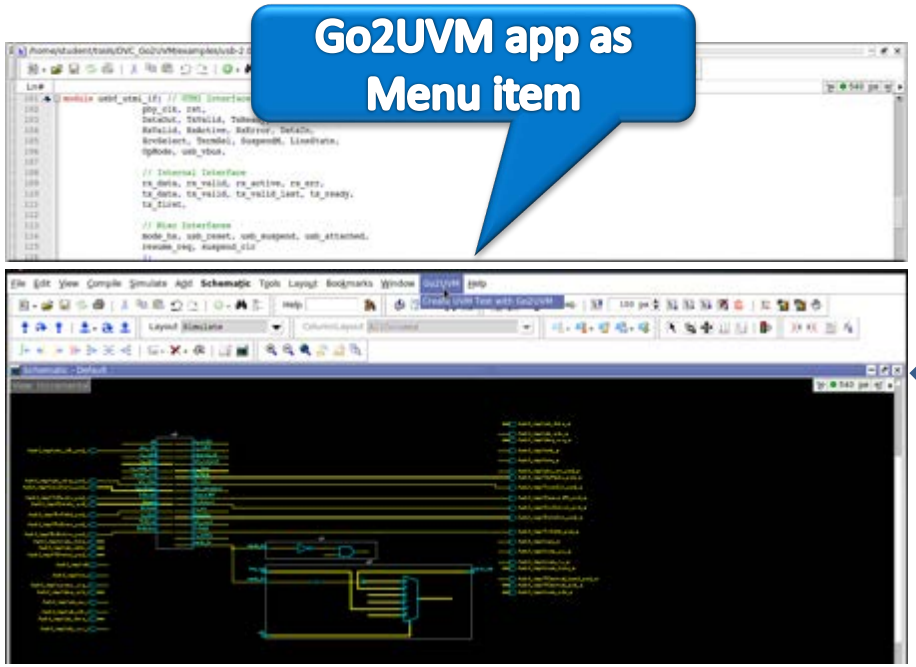
```
`g2u_display("End of reset”)
endtask:reset

task main();
  int i;
  `g2u_display(Start of test”)
   @(vif.cb);
   vif.cb.load  <=1'b1;
  repeat(2) @ (vif.cb);
   vif.cb.load  <= 1'b0;
   vif.cb.data  <= 8'h78;
  repeat(2) @ (vif.cb);
   vif.cb.load  <= 1'b1;
   vif.cb.cen   <= 1'b1;
   vif.cb.up_dn <= 1'b1;
  repeat(3) @ (vif.cb);
   vif.cb.load  <= 1'b1;
   vif.cb.cen   <= 1'b1;
   vif.cb.up_dn <= 1'b0;
```

TCL Apps – DVC_Go2UVM

- Open-source “apps” to generate Go2UVM files for a given RTL
- Works with all major EDA tools:
  - Aldec, Riviera-PRO
  - Cadence, IUS
  - Mentor Graphics, Questa
  - Synopsys, Verdi
- Given RTL top, the app generates:
  - SystemVerilog interface
  - Full go2uvm package (source code)
  - A template test that extends go2uvm_base_test
  - Scripts to compile and run on ALL major EDA tools

Go2UVM Generator App

- EDA tools have several interfaces
  - VPI (and like)
  - TCL
- Go2UVM app is integrated with EDA tool's GUI



SV Interface

Go2UVM pkg

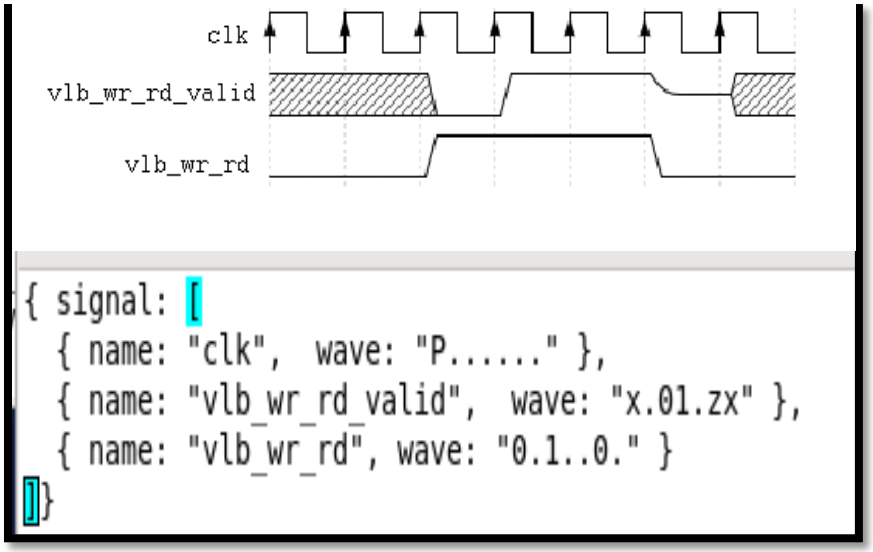
Go2UVM test template

Makefile

Waves2UVM – a Go2UVM App

- Specifications include timing diagrams
  - Formal, unambiguous way to capture timing
- How about converting spec timing diagrams to UVM tests?
  - Waves2UVM
- Works with popular waveform formats
- WaveDrom – an open-source timing diagram editor
  - JSON based description language
  - Rendering engine (SVG/PNG)

Waves2UVM




```
task main();
 `g2u_display("Start of main")
  fork
   drive_vlb_wr_rd_valid();
   drive_vlb_wr_rd();
  join
```

Go2UVM app

```
task drive_vlb_wr_rd();
  `g2u_display("Driving signal: vlb_wr_rd")
  vif.vlb_wr_rd <= 0;
  @(vif.cb);
  vif.vlb_wr_rd <= 0;
  @(vif.cb);
  vif.vlb_wr_rd <= 1;
  @(vif.cb);
  vif.vlb_wr_rd <= 1;
  @(vif.cb);
  vif.vlb_wr_rd <= 1;
  @(vif.cb);
  vif.vlb_wr_rd <= 0;
  @(vif.cb);
  vif.vlb_wr_rd <= 0;
  @(vif.cb);
  `g2u_display("End of stimulus for signal:
vlb_wr_rd")
endtask: drive_vlb_wr_rd
```

Signal Access API

- At times – arbitrary signal accesses are necessary in DV
  - Quick FIFO full test (or almost_full)
  - PLL outputs
  - Init sequence bypass, etc.
  - Detailed BFMs are not ready yet!
  - Gate level sims
- Force/Deposit/Release
  - A handy technique
  - Not a “recommended” approach, yet many find it useful occasionally

Signal Access Within UVM

- Recommended approach – add a virtual interface
  - Difficult for “sideband” signals
  - Across hierarchies, etc.
- May need to work across HDLs – Verilog/SV/VHDL
- EDA vendors have VPI-like system functions
  - Hard to use for first-timers
  - Code becomes tool dependent
- *go2uvm_sig_access* – A base class with APIs to wrap EDA vendor functions

Signal Access API in Go2UVM

```
class go2uvm_sig_access extends uvm_object;
  `uvm_object_utils(go2uvm_sig_access)

 extern static function void g2u_force (string sig_name,
   logic [`VW_G2U_SIG_MAX_W-1:0] sig_val,
   bit verbose = 1,
   bit is_vhdl_sig = 0);

 extern static function void g2u_deposit (string sig_name,
   logic [`VW_G2U_SIG_MAX_W-1:0] sig_val,
   bit verbose = 1,
   bit is_vhdl_sig = 0);

 extern static function void g2u_release(string sig_name,
   bit verbose = 1,
   bit is_vhdl_sig = 0);
endclass : go2uvm_sig_access
```

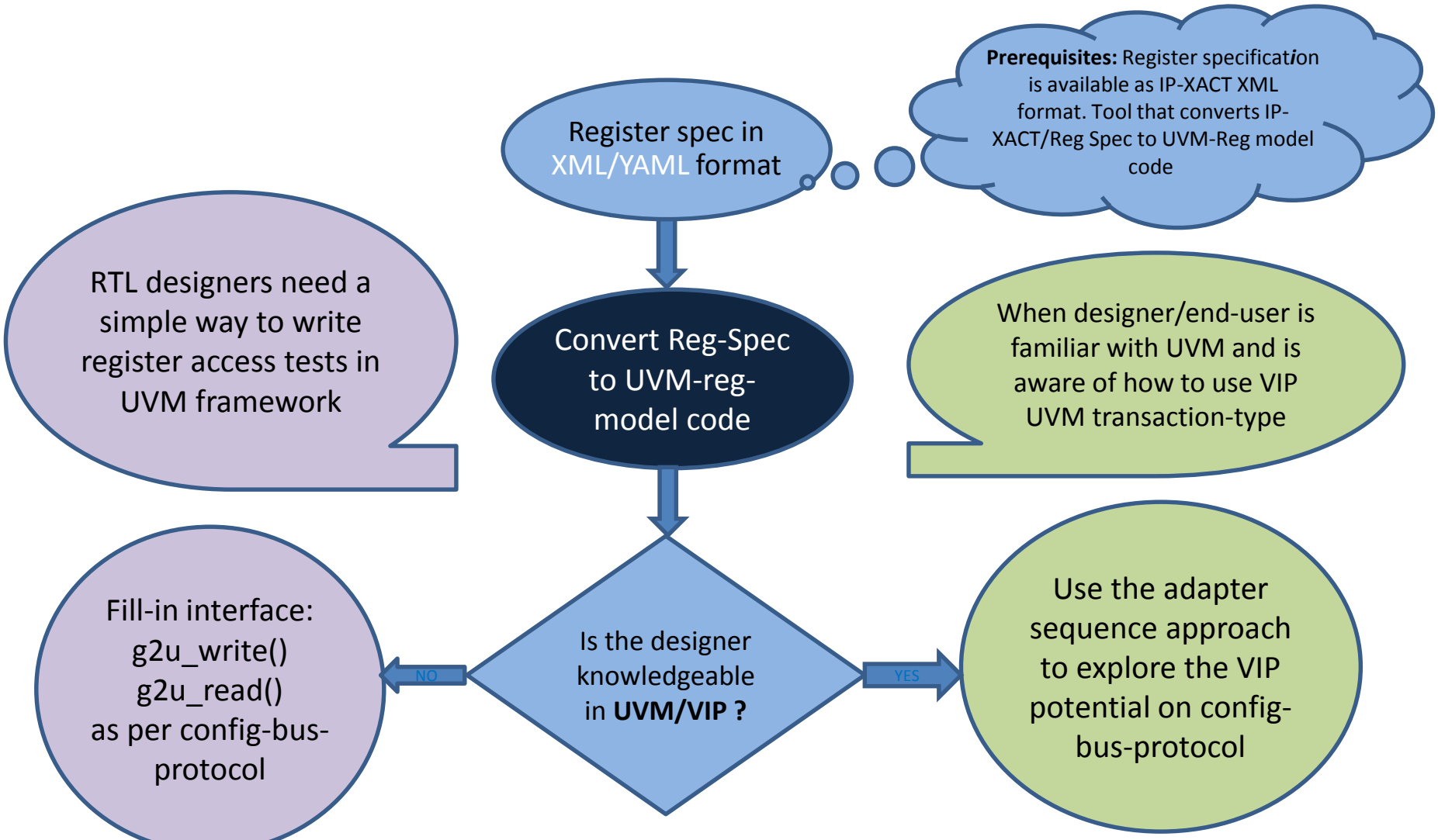
Using *g2u_force* API

```
//Import Go2UVM Package
import vw_go2uvm_pkg::*;

// Use the base class provided by the vw_go2uvm_pkg
`G2U_TEST_BEGIN(sprot_test)
  task main();
    `g2u_display("Starting force test");
     g2u_force("/sprot_go2uvm/sprot_0/byte_val", 22);
     #100;
     g2u_force("/sprot_go2uvm/sprot_0/byte_val", 'haa);
     #100;
    `g2u_display("End of main");
  endtask : main
`G2U_TEST_END
```

Signal Access – Under the Hood

- EDA vendors provide VPI-like system functions
  - Aldec: $force
  - Cadence: $nc_force
  - Mentor: $signal_force
  - Synopsys: $hdl_xmr (*)
- Go2UVM Signal access – wraps them in an easy to use manner
- Keeps user code portable across tools!

Register Verification



Register Testing Using Interface Approach (No Knowledge of VIP)

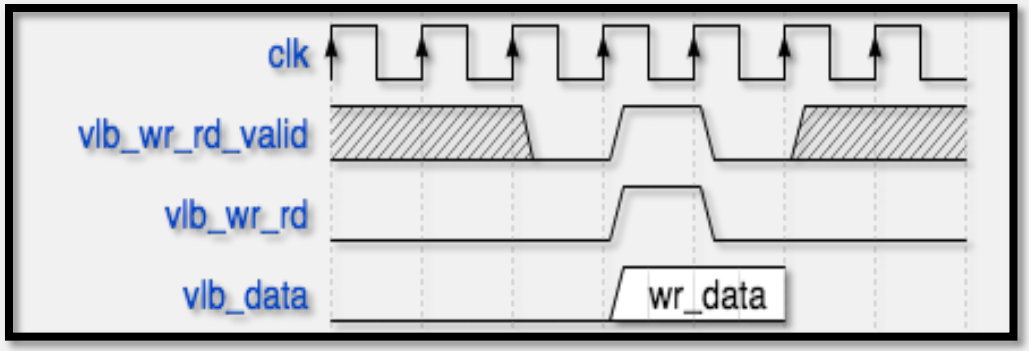
- Steps:
  - Generate UVM reg model from IP-XACT
  - Generate a template UVM TB/env through a tool/script (an open-source version may be made available soon for this via www.go2uvm.org)
  - The skeleton UVM interface structure has a dummy driver & monitor that creates g2u_write() and g2u_read() tasks
  - Fill in write() and read() inside interface(s) based on DUT's configuration bus protocol
  - Write a test in UVM framework using macros and APIs

Register Testing Using VIP BFM Model with Some UVM Knowledge

- Steps:
  - Generate UVM reg model from IP-XACT
  - Generate a template UVM TB/env through a tool/script (an open-source version may be made available soon for this via www.go2uvm.org)
  - The skeleton UVM structure has a adapter sequence driver that calls bus2reg() and reg2bus() functions
  - Create the VIP transaction item data type “txn”
  - Fill in bus2reg() and reg2bus() functions based on the configuration bus protocols
  - Write a test in UVM framework

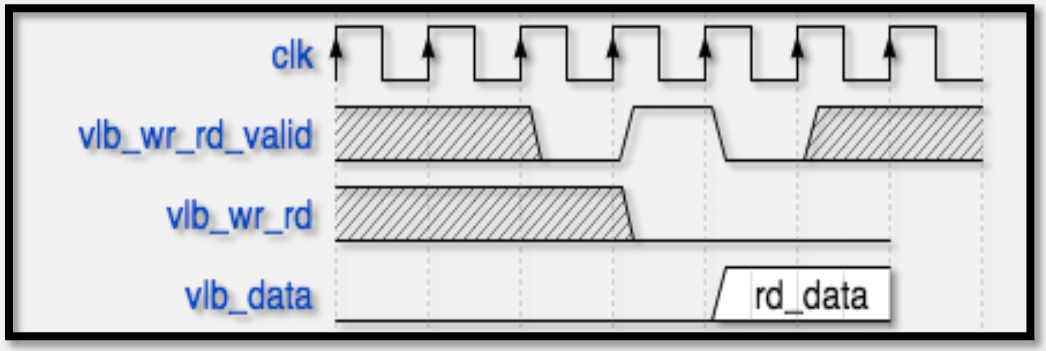
Sample *g2u_write* API in interface

```
//Import Go2UVM Package
import vw_go2uvm_pkg::*;
//change g2u_write()task according to your config-bus-protocol
task g2u_write(uvm_reg_addr_t wr_addr,
               uvm_reg_data_t wr_data);
    `g2u_display("Starting a new write")
     wr_rd_valid <= 1'b1;
     wr_rd <= 1'b1;
     addr <= wr_addr;
     data_in <= wr_data;
     @(posedge clk);
     wr_rd_valid <= 1'b0;
     wr_rd <= 1'b0;
     @(posedge clk);
endtask : g2u_write
```



Sample *g2u_read* API in interface

```
//Import Go2UVM Package
import vw_go2uvm_pkg::*;
//change g2u_read()task according to your config-bus-protocol
task g2u_read(uvm_reg_addr_t wr_addr,
                output uvm_reg_data_t rd_data);
    `g2u_display("Starting a new read")
     wr_rd_valid <= 1'b1;
     wr_rd <= 1'b0;
     addr <= rd_addr;
     @(posedge clk);
     wr_rd_valid <= 1'b0;
     wr_rd <= 1'b0;
     @(posedge clk);
     rd_data <= data_out;
endtask : g2u_read
```



Using *reg2bus()* function

```
virtual function uvm_sequence_item reg2bus(const ref uvm_reg_bus_op rw);
  int rand_ret;
    if(rw.kind == UVM_READ) begin
        rand_ret = txn.randomize() with {
          hwrite == 1'b0; cmd_type== AHB_SINGLE_RD;
          hsize == AHB_TWO; cmd_address[1:0]  == 2'h0;
          hlock == vip_enum_pkg::AHB_HLOCK_NORMAL;
          hprotns == vip_enum_pkg::AHB_HPROTNS_ZERO;};
    end
   txn.hwrite = ((rw.kind == UVM_WRITE) ? 1'b1 : 1'b0);
    if(txn.hwrite) begin // write txn
     txn.cmd_type = AHB_SINGLE_WR;
     txn.hwdata[0] = rw.data;
    end
    else begin // read txn
     txn.cmd_type = AHB_SINGLE_RD;
     txn.hrdata[0] = rw.data;
    end
    txn.cmd_byte_count = rw.n_bits/8;
    txn.cmd_address = rw.addr;

    return txn;
```

Using *bus2reg()* function

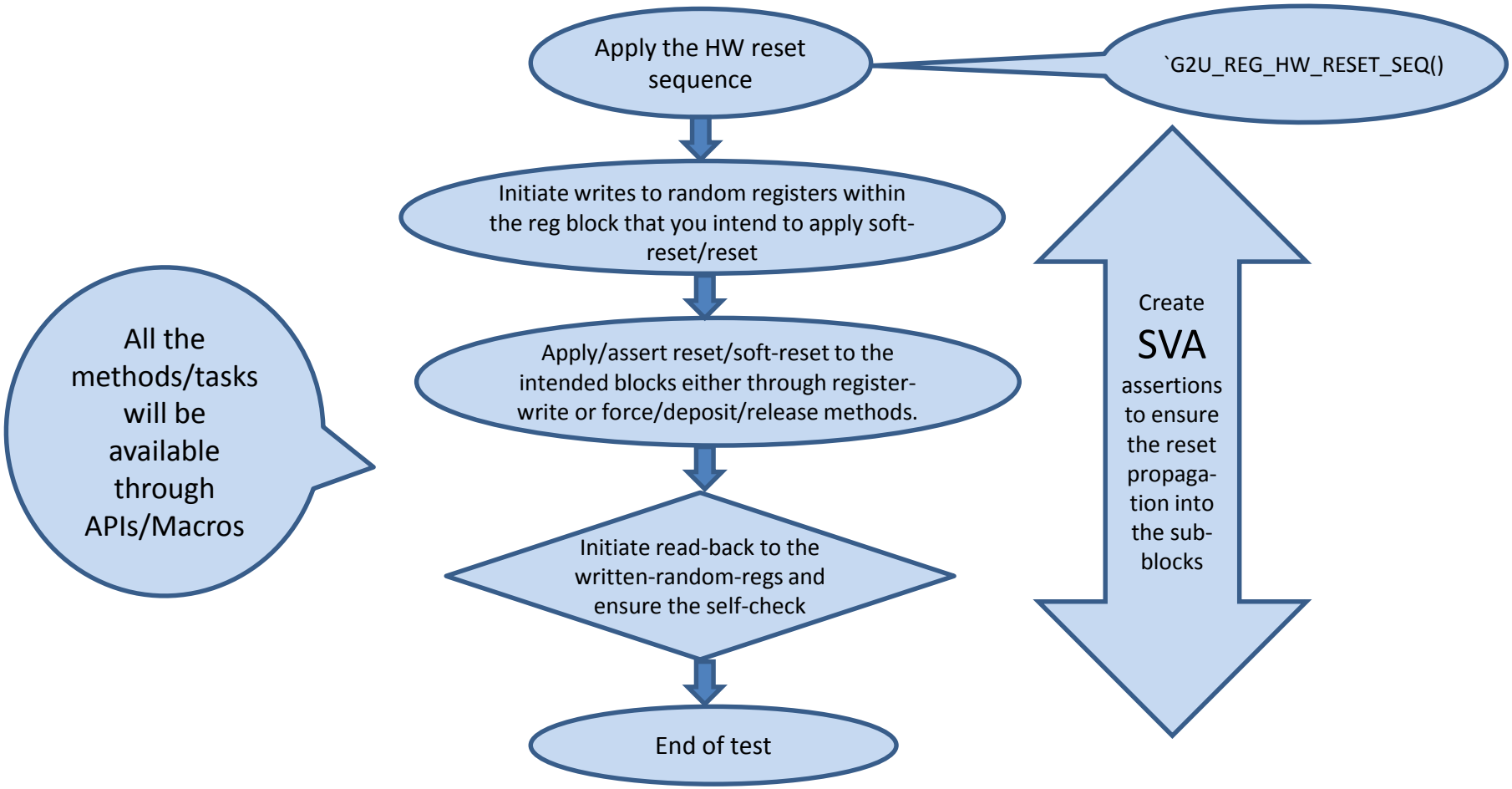
```
vip_trans#(32,32) txn;
vi  rw.addr = txn.haddr[0];
            rw.n_bits = txn.cmd_byte_count*8;
            rw.byte_en = 4'hf;
            rw.status = UVM_IS_OK;
            `g2u_display( "REG_ADAPTER: In bus2reg ")
          end
        end
        else if(txn.hresp[0] === AHB_HRESP_ERROR) begin
              rw.kind = UVM_READ;
              rw.data = txn.hrdata[0];
              rw.addr = txn.haddr[0];
              rw.n_bits = txn.cmd_byte_count*8;
              rw.byte_en = 4'hf;
              rw.status = UVM_IS_OK;
        end
      end
   endfunction
```

Reg Test with Go2UVM

```
`G2U_REG_TEST_BEGIN(i2c_g2u_bit_bash_test, i2c_g2u_env)
  task main();
    `G2U_REG_BIT_BASH_SEQ(`I2C_REG_BLOCK)
    `g2u_display("End of test ")
  endtask : main
`G2U_REG_TEST_END
```

```
`G2U_REG_TEST_BEGIN(i2c_g2u_access_policies_test,
                    i2c_g2u_env)
  task main();
    `G2U_REG_ACCESS_SEQ(`I2C_REG_BLOCK)
    `g2u_display("End of test ")
  endtask : main
`G2U_REG_TEST_END
```

Testing Reset Propagation



2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Thank You!

accellera SYSTEMS INITIATIVE