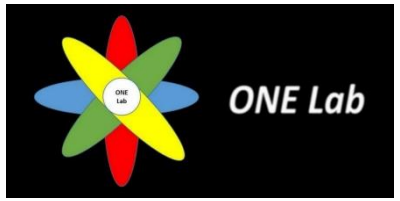


Faculty of Engineering
Cairo University

Project title:

Generic UVM for Soft Processors

Follow-up Thread

Agenda

- ❑ Introduction & History
- ❑ Problem Definition
- ❑ Objective
- ❑ Related Work
- ❑ Contribution
- ❑ Proposed Solution
- ❑ Results
- ❑ Comparison with related work
- ❑ Demo
- ❑ Conclusion & Future work

Introduction & History

- Leading up to the past decade, Digital IC development and verification suffered severely, due to lack of efficiency and testing constraints on manually-established test benches. Older environments included poorly-written VHDL/Verilog models, and almost 60% of the total project time involved an extreme hassle to adequately verily or even achieve basic functional testing.
- Although System Verilog quickly took the mantle as the industry's first *Hardware Description and Verification Language (HDVL)*, it wasn't long until proper verification plans and *Constrained-Random Verification (CRV)* were introduced as main pillars of the **Universal Verification Methodology (UVM)**; implementing a real-world test-case generator in a pseudo-random controlled and constrained randomness.
- **UVM** blocks can tackle complex multiple-pipeline CPU designs/ISA's, including multiple-precision operations; while being absolutely re-usable, encapsulated, and accommodates wide range of design sizes and types, in addition to providing functional coverage methodologies.

Problem Definition

The time spent by the verification engineers to verify the functionality of any complex design using ordinary verification plans consumes more than half of the total project time.

Even after using OVM and UVM with the verification process nowadays, we still need something as ***generic*** and reusable as possible; so that we can reach the golden dream: only one verification methodology for everything; to save effort, time and cost.

Objective

The objective of our project is: only one ***generic*** UVM used to verify the functionality of three different open-source cores (soft processors), based on three different ISAs and having different number of pipeline stages (Ri5cy, Leon 2.4 and Amber a23/a25).

Related Work

We didn’t find any related works with the same idea/concept of a generic UVM (the same UVM test bench without any tweaking) to verify different IPs or soft processors (cores); all the related works are based on the idea of implementing a UVM test bench to verify only one DUT (can be reused but after tweaking the source code itself).

Some of the references we toke as guidelines during this milestone:

1. “The UVM Primer: An Introduction to the Universal Verification Methodology” by Ray Salemi.
2. “Five Stage Pipelined MIPS Processor Verification using UVM” from GitHub + the source code.
3. “UVM Cookbook” from Mentor Graphics Verification Academy.

Contribution

The team members: Kholoud Mahmoud, Randa Ahmed, Karim Ayman, Mostafa Ayman, Waleed Samy and Yasser Ibrahim.

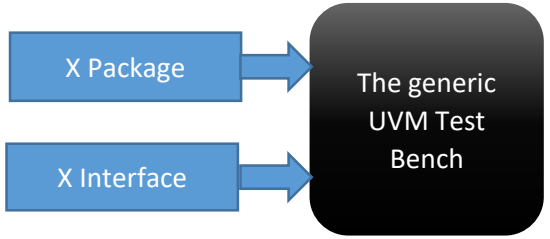
The contributions (till this point of the time plan):

Task	Description	Contributors
Task #1	Survey	All the team members
Task #2	Ri5cy core verification plan	All the team members
Task #3	Leon 2.4 core verification plan	Randa, Karim and Mostafa
	Amber a23/a25 core verification plan	Kholoud, Waleed and Yasser
	The general verification plan	All the team members
Task #4 (1st milestone in the implementation)	Implementing the Sequence, the Sequence items and the Packages	Karim and Mostafa
	Implementing the Driver and the Interface	Waleed and Yasser
	Implementing the Monitor	Kholoud
	Implementing the Scoreboard	Randa
	Editing, connecting the components of the test bench together, debugging and testing	All the team members

Proposed Solution

The user can use our generic UVM with any core (soft processor) of our three cores (***or any available core with the same instructions***) after attaching only 2 things to the test bench:

- **The core package**: includes all the core instructions and its format mapping.
- **The core interface**: **(1)** includes the ports of the top-level module (core interface), **(2)** each interface has functions that the driver use to drive instructions or data to the DUT and **(3)** deals with clk and timing to deal with each processor regarding the timing constraints.



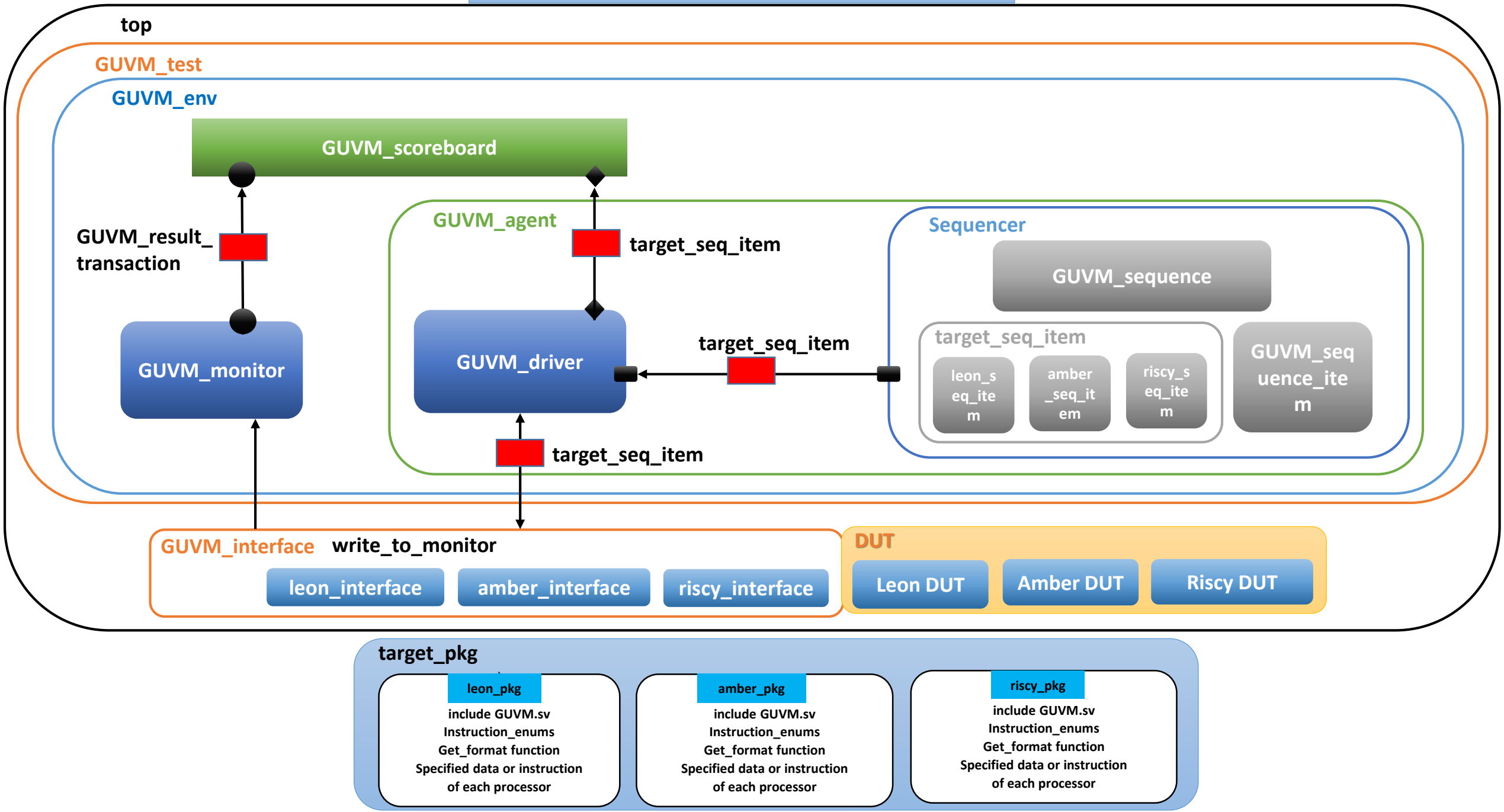
Results

The desired output of the project is: only one ***generic*** UVM used to verify the functionality of three different open-source cores (soft processors), based on three different ISAs and having different number of pipeline stages (Ri5cy, Leon 2.4 and Amber a23/a25).

Until now, we've successfully reached our first milestone of the implementation of our generic UVM: a fully functional generic UVM to test the functionality of one instruction with our three different DUTs, the rest of the instructions will need just some changes in the timing inside the interface functions (to reach the critical path).

The architecture of our generic UVM test bench is represented in the next slide.

Generic UVM for Soft Processors Test Bench Architecture



target_pkg

leon_pkg

include GUVM.sv
Instruction_enums
Get_format function
Specified data or instruction
of each processor

amber_pkg

include GUVM.sv
Instruction_enums
Get_format function
Specified data or instruction
of each processor

riscy_pkg

include GUVM.sv
Instruction_enums
Get_format function
Specified data or instruction
of each processor

Comparison with the Related Work

We are building a ***generic*** UVM to verify the functionality of three different open-source soft processors (or any available core with the same instructions and specifications), based on three different ISAs and having different number of pipeline stages (Ri5cy, Leon 2.4 and Amber a23/a25).

Comparing to the related works, we found none of them hosting the same idea/concept of a generic UVM (the same UVM test bench without any tweaking) to verify different IPs or soft processors, but they are all based on a quite similar UVM components infrastructure.

Demo

The user can use our generic UVM test bench by doing the following steps:

1. download the source code from GitHub with this link: https://github.com/cufeolm/codeGP.git
2. Run the .bat file of the desired core to start the test bench: ***a.bat*** for Amber core, ***l.bat*** for Leon core and ***r.bat*** for Ri5cy core.
3. Check the transcript file for results.

The following figures in the next slide are screenshots from the transcript showing the result of the generic UVM.

```
# UVM_INFO GUVM/GUVM_scoreboard.sv(83) @ 13300: uvm_test_top.env_h.sb [ADDITION_PASS] Actual Calculation= 596050502
Expected Calculation= 596050502
# Scoreboard started
# There should be no rvalid when we the LSU is IDLE               13310
# driver has started
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1268) @ 13600: reporter [TEST_DONE] 'run' phase is ready
to proceed to the 'extract' phase
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   13
# UVM_WARNING :    0
# UVM_ERROR :    2
# UVM_FATAL :    0
# ** Report counts by id
# []     1
# [ADDITION_FAIL]     2
# [ADDITION_PASS]     8
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# ** Note: $finish    : C:/questasim64_10.4e/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 13600 ns  Iteration: 60  Instance: /top
# End time: 14:45:31 on Apr 05,2020, Elapsed time: 0:00:13
# Errors: 0, Warnings: 38
```

Ri5cy core

```
# UVM_INFO GUVM/GUVM_scoreboard.sv(83) @ 33800: uvm_test_top.env_h.sb [ADDITION_PASS] Actual Calculation= 596050502
Expected Calculation= 596050502
# Scoreboard started
# driver has started
# driver starting fetching
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1268) @ 34000: reporter [TEST_DONE] 'run' phase is ready
to proceed to the 'extract' phase
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   12
# UVM_WARNING :    0
# UVM_ERROR :    3
# UVM_FATAL :    0
# ** Report counts by id
# []     1
# [ADDITION_FAIL]     3
# [ADDITION_PASS]     7
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# ** Note: $finish    : C:/questasim64_10.6c/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 34 us  Iteration: 60  Instance: /top
# End time: 15:49:20 on Apr 06,2020, Elapsed time: 0:00:10
# Errors: 0, Warnings: 2
```

Amber core

```
# UVM_INFO GUVM/GUVM_scoreboard.sv(83) @ 10600: uvm_test_top.env_h.sb [ADDITION_PASS] Actual Calculation= 596050502
Expected Calculation= 596050502
# Scoreboard started
# driver has started
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1268) @ 10800: reporter [TEST_DONE] 'run' phase is ready
to proceed to the 'extract' phase
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   13
# UVM_WARNING :    0
# UVM_ERROR :    2
# UVM_FATAL :    0
# ** Report counts by id
# []     1
# [ADDITION_FAIL]     2
# [ADDITION_PASS]     8
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# ** Note: $finish    : C:/questasim64_10.6c/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 10800 ns  Iteration: 64  Instance: /top
# End time: 15:51:10 on Apr 06,2020, Elapsed time: 0:00:06
# Errors: 0, Warnings: 22
```

Leon core

Conclusion & Future work

As a conclusion: with our 1st milestone of the implementation of the generic UVM, we have successfully proven the possibility/idea of creating only one UVM to verify more than one core (soft processor).

The next milestones will cover the following:

1. Testing the functionality of the rest of the instructions of the three different cores.
2. Negative testing.

Future work:

1. A complete coverage for the three DUTs.
2. Applying the concept of the generic UVM on a complete SOC.
3. Applying different machine learning algorithms to reach the golden dream/goal: only one verification methodology to verify everything.