UVMF Mathworks® Integration Users Guide

Version 2020.3_1

Contents

Chapter 1

Introduction to the UVMF Integration with Mathworks®

1.1 Overview

The UVM Framework provides automation for integrating Mathworks® generated DPI-C and RTL into a UVM based simulation environment. A shell script, `mtlb2uvmf.csh`, can be used to generate a block level environment that uses the Mathworks® generated DPI-C and RTL. By itself, a UVM based block level simulation provides little value over the validation that can be done within MATLAB® or Simulink®. Significant value is achieved when this block level environment is simulated within the context of a subsystem or full chip environment. The block level environment and interfaces generated using `mtlb2uvmf.csh` can be automatically instantiated as a sub-environment within subsystem or chip level environments using the UVMF code generator, `yaml2uvmf.py`.

1.2 Development Flow From Block to Top

When beginning from scratch it is recommended to take the following steps when developing an initial UVMF bench with the generator script:

1. Generate the DPI-C and RTL code using MATLAB® or Simulink®. Two DPI-C packages should be generated. The first package should model the design created in MATLAB® or Simulink®. This will be used for prediction and will provide cycle by cycle data for checking by the scoreboard against RTL output. The second package should model

input stimulus provided by MATLAB® or Simulink® to the design. This is used to verify correctness of the block level prediction, RTL, and environment. Each of these packages will be in their own directory whose name ends with "_build".

2. Execute `mtlb2uvmf.csh` providing a reference to the prediction package directory and stimulus generation package directory created by MATLAB® or Simulink®.

3. Run the block level simulation in either command line or GUI mode as described in `$UVMF_HOME/docs/UVM_Framework_Users_Guide.pdf`.

4. Instantiate the block level environment in a parent environment using YAML. Execution of `mtlb2uvmf.csh` generates a YAML file that characterizes the block level environment. This environment can be added as a sub-environment to a parent environment using YAML. Details of the UVMF code generator can be found in `$UVMF_HOME/docs/UVMF_Code_Generator_YAML_Reference.pdf`.

5. Generate the parent environment using the UVMF code generator, `yaml2uvmf.py` as described in `$UVMF_HOME/docs/UVMF_Code_Generator_YAML_Reference`.

6. Run the top level simulation environment which contains the block level environment as described in `$UVMF_HOME/docs/UVM_Framework_Users_Guide.pdf`.

Chapter 2

Input Data and Format for Generating UVMF Simulations from Mathworks Output

2.1 Overview

Once a design has been created using MATLAB® or Simulink®, the HDL Verifier™ feature can be used to generate a DPI-C model of the design. The generated souce will be placed in a directory with the following name: <designName>_build. The HDL Verifier™ feature can also be used to generate a DPI-C model of the block that provides input stimulus to the design within MATLAB® or Simulink®. The generated souce will be placed in a directory with the following name: <blockName>_build. These two _build directories are all that `mtlb2uvmf.csh` needs in order to generate a complete UVM simulation environment for the design created in MATLAB® or Simulink®. As generated, this UVM environment is ready for automatic integration into higher level simulations including subsytem or full-chip.

As an example: When HDL Verifier™ is used to create DPI-C for a design block named adder, the resulting directory will be named adder_build. When HDL Verifier™ is used to create DPI-C for a stimulus block named adder_stimgen the resulting directory will be named adder_stimgen_build. The command for generating a UVM environment from these two directories is: `mtlb2uvmf.csh adder adder_stimgen`. Note that the _build has been left off of the arguments to `mtlb2uvmf.csh`.

The HDL Coder™ feature can be used to generate RTL for the design characterized in MATLAB® or Simulink®. This code will be referenced by the

generated simulation environment using an environment variable. It is not required for generation of the simulation environment. However, the simulation environment will instantiate the design and connect its ports to SystemVerilog interfaces that will provide stimulus to inputs/inouts and capture values from outputs/inouts.

Chapter 3

Output from Generating UVMF Simulations from Mathworks Output

3.1 Overview

Running `mtlb2uvmf.csh` generates the following output.:

- Fully functional and reusable simulation environment located under `uvmf_template_output`
- A text file that contains the YAML which characterizes the generated simulation environment. This yaml is used to generate the environment located under `uvmf_template_output`. It is also used when executing `yaml2uvmf.py` to create a parent environment that contains the generated environment located under `uvmf_template_output`.

These outputs are described in more detail in the following sections.

3.2 Generated Simulation Environment

The generated UVM code located in uvmf_template_otput is a fully functional simulation. It contains the following featues which are described in more detail in the following subsections:

- Top level module located in `uvmf_template_output/project_benches/<designName>/tb/testbench/hdl_top.sv`.

- UVM test case located in
`uvmf_template_output/project_benches`
`/<designName>/tb/tests/src/DPI_stimgen_test.svh`
- UVM sequence located in
`uvmf_template_output/project_benches`
`/<designName>/tb/sequences/src/DPI_stimgen_sequence.svh`
- UVM Environment located in
`uvmf_template_output/verification_ip`
`/environment_packages/<designName>_env_pkg/src`
- UVM Agent to provide input stimulus located in
`uvmf_template_output/verification_ip`
`/interface_packages/<designName>_input_agent_pkg/src`
- UVM Agent to capture output activity located in
`uvmf_template_output/verification_ip`
`/interface_packages/<designName>_output_agent_pkg/src`
- UVM Predictor located in
`uvmf_template_output/verification_ip`
`/environment_packages/<designName>_env_pkg/src`
- UVM Scoreboard located in
`$UVMF_HOME/uvmf_base_pkg/src`
- UVM Environment configuration located in
`uvmf_template_output/verification_ip`
`/environment_packages/<designName>_env_pkg/src`
- Makefile based compile and run flow

3.2.1 Top level module

The top level module instantiates the DUT and SystemVerilog interface based bus functional models, BFM's, that connect to the ports of the DUT. The top level module also generates a clock and reset. The virtual interface handles of the BFM's are placed into the uvm_config_db by the top level module for retrieval by UVM objects.

3.2.2 UVM Test Case

The generated test package contains a test, named DPI_stimgen_test, that sets a factory override to run the
`DPI_stimgen_sequence`.

3.2.3 UVM Test Sequence

The generated test sequence package contains a sequence, named `DPI_stimgen_sequence`, that uses the DPI-C functions generated by Mathworks® which provides input stimulus for the design. This sequence can be configured to run a fixed number of cycles. It can also be configured to run continuously for the duration of the simulation. This is useful for cases where data path stimulus needs to be provided throughout a simulation whose length is determined by other operations in the test.

3.2.4 UVM Environment

The generated environment contains an input agent, output agent, predictor, and scoreboard. DUT operations are scoreboarded and checked on a cycle by cycle basis throughout the test. Coverage components can be easily added to this environment based on the coverage model of the project.

3.2.5 UVM Input Agent

An agent package is generated to provide input stimulus to the DUT. This package contains all class definitions for the agent, driver, monitor, configuration, transaction, sequence, coverage, etc. The SystemVerilog interface based driver BFM and monitor BFM are also part of the generated agent source. The agent can be configured as active or passive to support block to top reuse of the generated environment.

3.2.6 UVM Output Agent

An agent package is generated to capture output from the DUT. This package contains all class definitions for the agent, driver, monitor, configuration, transaction, sequence, coverage, etc. The SystemVerilog interface based driver BFM and monitor BFM are also part of the generated agent source. The agent can be configured as active or passive to support block to top reuse of the generated environment.

3.2.7 UVM Predictor

The generated environment package includes a predictor class that uses the DPI-C functions generated by Mathworks® which models the DUT behavior. Values observed on the DUT inputs are submitted to the DPI-C functions. Values returned from the C model are placed in a transaction and sent to the scoreboard for comparison.

3.2.8 UVM Scoreboard

The UVM Framework provides a set of scoreboards that compare expected transactions with DUT output transactions. These scoreboards store expected transactions until DUT output transactions arrive for comparison. Once the comparison is made, both transactions are discarded. Additional details on the scoreboards is available in
`$UVMF_HOME/docs/UVM_Framework_Users_Guide.pdf`

3.2.9 UVM Environment configuration

Each UVMF environment has a configuration object that contains configuration variables as well as configuration objects for each agent within the environment.

3.3 Generated YAML File

The generated YAML file characterizes the generated environment, input agent, and output agent. It is used by the UVMF code generator `yaml2uvmf.py` to generate parent environments, other agent packages, integrate QVIP agents for standard protocols, and benches for running simulations. Details of how to generate environments that instantiate sub-environments can be found in `$UVMF_HOME/docs/UVMF_Code_Generator_YAML_Reference.pdf`

3.3.1 Makefile based compile and run flow

The UVMF generates a Makefile based compile and run flow. It can run simulations in command line or GUI mode. Technologies like VRM, VM, Visualizer, inFact, Questa VIP, Covercheck, etc. are built into the flow.

Chapter 4

Flow for Generating UVMF Simulations from Mathworks Output

4.1 Overview

These are the steps required to generate a UVM environment for designs created using Mathworks®. The steps shown are desribed in detail in the following sections of this chapter.

1. Generate the DPI-C code for prediction and block level stimulus using HDL Verifier™. Generate the RTL code using HDL Coder™.
2. Generate the UVM environment using `mtlb2uvmf.csh`.
3. Set environment variables for the simulation which reference DPI-C code, header files, and the design RTL.
4. Run the simulation.

4.2 Generating DPI-C and RTL using Mathworks®

The procedure for generating DPI-C and RTL from HDL Verifier™ is beyond the scope of this document. Please reference HDL Verifier™ documentation. Instructional videos are also available online. When generating the DPI-C, be sure to select the following options: systemverilog dpi, grt.

4.3 Generating the UVM Environment using `mtlb2uvmf.csh`

The `mtlb2uvmf.csh` is located in `$UVMF_HOME/scripts`. You will need to either reference the command directly using `$UVMF_HOME/scripts` or place `$UVMF_HOME/scripts` in your PATH variable. `mtlb2uvmf.csh` requires two arguments. The first argument references the directory that contains the DPI-C model the design created in MATLAB® or Simulink®. This will be used for prediction and will provide cycle by cycle data for checking by the scoreboard against RTL output. The second argument references the directory that contains the input stimulus DPI-C model generated by MATLAB® or Simulink®. This is used to verify correctness of the block level prediction, RTL, and environment. Each of these packages will be in their own directory whose name ends with "_build".

As an example: When HDL Verifier™ is used to create DPI-C for a design block named adder, the resulting directory will be named adder_build. When HDL Verifier™ is used to create DPI-C for a stimulus block named adder_stimgen, the resulting directory will be named adder_stimgen_build. The command for generating a UVM environment from these two directories is: `mtlb2uvmf.csh adder adder_stimgen`. Note that the _build has been left off of the arguments to `mtlb2uvmf.csh`.

4.4 Environment Variables Required for Simulation

Environment variables are used to reference C code, C headers, and RTL code that are either generated by Mathworks® or is part of the Mathworks® installation. A list of environment variables required for simulation are provided by `mtlb2uvmf.csh` in a file named
`setup_<designName>_environment_variables.source`. This file is in the generated
`uvmf_template_output/project_benches/<designName>`. Use this file as a list of environment variables that need to be set. The following lists the environment variables needed.

- <designName>_VERILOG_DUT_SRC - Reference generated verilog RTL
- <designName>_VHDL_DUT_SRC - Reference generated vhdl RTL
- <designName>_ENV_DPI_SRC - Reference generated prediction DPI-C source
- <designName>_ENV_GCC_COMP_ARGUMENTS - Reference include directories for generated prediction DPI-C source
- <designName>_input_agent_IF_DPI_SRC - Reference generated stimulus generator DPI-C source
- <designName>_input_agent_IF_GCC_COMP_ARGUMENTS - Reference include directories for generated stimulus generator DPI-C source

4.5 Running the Simulation

Running the simulation can be done by executing one of the following commands from within the following generated directory:
`uvmf_template_output/project_benches/<designName>/sim`

- make cli TEST_NAME=DPI_stimgen_test
- make debug TEST_NAME=DPI_stimgen_test

Additional details on running simulations can be found in
`$UVMF_HOME/docs/UVM_Framework_Users_Guide.pdf`

Chapter 5

Reusing Generated UVMF Environments from Mathworks Output in Chip Level Simulations

5.1 Overview

One key characteristic of the UVM Framework and its generated code is reuse. Horizontal reuse of verification code from project to project. Vertical reuse from block to top. Platform reuse from simulation to emulation. By using the UVM Framework, a verification reuse library is established that can span projects, teams, sites, and even companies. Any generated UVMF environemnt can be a sub-environment within another environment. This includes environments generated by UVMF based on input from Mathworks® generated DPI-C source. This allows for the rapid generation of environments and simulation infrastructure within hours rather than weeks to months.

Once the environment package and both interface packages are generated using `mtlb2uvmf.csh` these packages are part of the verification reuse library and available for reuse in parent environments. This can include subsystem, full-chip, as well as system level simulations. Details of how to generate environments that instantiate sub-environments can be found in `$UVMF_HOME/docs/UVMF_Code_Generator_YAML_Reference.pdf`

Appendix A

Additional Information Regarding UVM Framework

A.1 Overview

The UVM Framework is an open-source package that provides a reusable UVM methodology and code generator that provides rapid testbench generation. As a reusable UVM methodology, it defines a UVM use model that guarantees horizontal reuse across projects, vertical reuse from block to top, and platform reuse from simulation to emulation.

Development of UVMF started in 2007. It utilizes best practices developed in the field starting with AVM through OVM to UVM. UVMF development continues through cooperation between customers, Mentor field application engineers, and Mentor product engineering. UVMF is used by companies worldwide.

A.2 No License Required

No license is required to use the UVMF base class library or the UVMF code generators. The UVMF is provided by Mentor in order to increase customer productivity. The UVMF source is licensed under the Apache License, Version 2.0, just as the UVM base class library.

A.3 How to Download UVMF

UVMF is available for download from Verification Academy and is available on this page:
https://verificationacademy.com/courses/UVM-Framework-One-Bite-at-a-Time
Automatic notification of new releases is available through this site.

A.4 Available Videos, Training, and Support

An instructional video series is available on Verification Academy on this page:
https://verificationacademy.com/courses/UVM-Framework-One-Bite-at-a-Time
This video series covers a wide range of topics relating to UVMF including its architecture, operation, and use.

Training on UVMF, UVM, and SystemVerilog is available through Mentor on-demand sessions that can be found on this page:
https://www.mentor.com/training/courses/functional-verification-training-library

Support is available through your local Mentor functional verification field application engineer.

Training on SystemVerilog/UVM/UVMF is available at North Carolina State University through Engineering Online. Additional information can be found on this page:
https://www.engineeringonline.ncsu.edu/course/ece-748-advanced-verification-with-uvm/