UVM Template Generator (uvmgen) User Guide

G-2012.09
September 2012

Comments?
E-mail your comments about this manual to:
vcs_support@synopsys.com.

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

> "This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____."

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AEON, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, CHIPit, CoMET, Confirma, CODE V, Design Compiler, DesignWare, EMBED-IT!, Formality, Galaxy Custom Designer, Global Synthesis, HAPS, HapsTrak, HDL Analyst, HSIM, HSPICE, Identify, Leda, LightTools, MAST, METeor, ModelTools, NanoSim, NOVeA, OpenVera, ORA, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Sonic Focus, STAR Memory System, Syndicated, Synplicity, the Synplicity logo, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, UMRBus, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, ARC, ASAP, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomExplorer, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Eclypse, Encore, EPIC, Galaxy, HANEX, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM^{plus}, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Intelli, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Macro-PLUS, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, ORAengineering, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, RippledMixer, Saturn, Scirocco, Scirocco-i, SiWare, Star-RCXT, Star-SimXT, StarRC, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCSi, VHDL Compiler, VMC, and Worksheet Buffer are trademarks of Synopsys, Inc.

Service Marks (^{sm})

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
ARM and AMBA are registered trademarks of ARM Limited.
Saber is a registered trademark of SabreMark Limited Partnership and is used under license.
All other product or company names may be trademarks of their respective owners.

Contents

1

UVM Template Generator (uvmgen)

The chapter explains about `uvmgen` for accelerated UVM testbench development in the following major sections:

Introduction

The adoption of base class library can be easily accelerated if there is a powerful set of templates for you to start with. This helps in the adoption process and ensures that the common adoption challenges are mitigated. Also, if the base class itself is evolving, the templates

help to abstract out the changing APIs and recommendations. You can concentrate on adding functionality rather than having to worry about the syntax.

`uvmgen` is a template generator for creating robust, extensible UVM compliant environments. Though the primary functionality of `uvmgen` is to help minimize VIP and environment development cycle by providing the detailed templates for developing UVM compliant verification environments. In addition, it can also be used to quickly understand how different UVM base classes can be used in different contexts. This is possible because, the templates uses a rich set of the latest UVM features to ensure the appropriate base classes and their features are picked up optimally.

Features

The wide user interaction mechanism available with `uvmgen` provides features and options that allows you to choose the modes that are relevant to your requirement. In addition, it provides you the option to have your own templates, thus offering a rich layer of customization. Based on the requirements, you can generate the individual templates of different verification components or you can generate a complete verification environment.

The other salient and key features of `uvmgen` and the templates generated by it are:

- Mechanism to Enable / Disable UVM Field Automation Macros (Default is enabled)
- Mechanism to Enable / Disable UVM REG Hookup
- Helps Generate,

    - Agents (master/slave)
    - Different types of drivers, monitors
    - Sequences
    - Sequence library
    - Multiplexed domain support in UVM REG
- Multi-Driver generation support (For various different types of transactions)
- Default or user defined naming
- Support for various types of unidirectional and bi-directional TLM connections between generator and driver
- Analysis ports/exports coverage
- Usage of resource/config_db for virtual interfaces connection
- Support for Quick (minimum queries) and Verbose Mode
- Command-line support and support for file based entry formats
- Organized directory structure creation, with Makefile using appropriate options to compile/simulate the verification environment

uvmgen Options

`uvmgen` is a part of the VCS installation. It is available in `$VCS_HOME/bin`.

It can be invoked as:

```
uvmgen  [-L libdir] [-X] [-d] [-o fname] [-O][-q]
```

To use `uvmgen`, you need to set the `$VCS_HOME` environment variable.

Table 1-1 lists the options and their functionalities supported by uvmgen:

Table 1-1 Options supported by uvmgen

Option	Functionality
-L liblist	Pass the user-defined template directories. This is used in conjunction with -X option
-d	Display the standard template location
-o filename	Provide the user-defined name to the generated output file for individual templates
-O :	Enable overwriting of the output file
-X (user-template-path) :	Enables the use of custom templates
-q	Enable the quick mode of uvmgen (minimum questions asked for complete environment generation)
-RAL [y/n]	To enable/disable Hooking up UVM REG models in the environment in quick mode
-ENV (name)	Provide environment name in quick mode
-TR (name) :	Provide the transaction name in quick mode
-BU (name)	Provide an Intermediate BU layer in quick mode

uvmgen Customization

`uvmgen` allows customization where you have the choice to exclude the default template library and specify your template library. This is done with the following command line:

```
uvmgen -X -L <User template library path>
```

Here `-X` is used to exclude the default library and you can provide your own template library path by using `-L` option. The multiple template directories can also be provided and in this scenario, whenever there is a template in both the libraries, the one from the second template library overrides the first one.

Generating Templates: uvmgen Modes

`uvmgen` provides the options to either generate the complete environment or generate specific individual templates.

Environment Generation Mode

In this mode, you need to answer a set of questions based on the different UVM components that are required in the desired verification environment. At the end of this process, a complete environment is generated with an appropriately organized directory structure, `Makefile` for compiling and simulating the same and a `README` which explains the directory structure and the different options for the Makefile.

```
Ex: % uvmgen
```

You will see the following two options:

1) Enter 1 to Create Complete Environment

2) Enter 2 to Generate Individual Template

Choose 1 to generate complete environment.

Example 1-1 Complete Environment Generation:

```
% uvmgen

1) Enter 1 to Create Complete Environment
2) Enter 2 to Generate Individual Template

Select [1-2]: 1

Do you want to create your own methods [Instead of UVM shorthand macros] ?
Select [ y/Y/n/N ][Default: n]:n

Would you be associating UVM REG models in your environment class? enter (y/
n): y
Enter Name of RAL Adapter: ral_ad1
Enter the environment name: top_env
Do you want to create Agents?
Select(y/Y/n/N) [Default: n]:y
Enter Master agent data
Enter name of master agent: mas
Enter name of sequencer: seq1
Enter name of driver: drv1
Enter name of monitor: mon1
Enter name of interface: intf1
Enter name of the transaction: tr1
Enter Slave agent data
Enter name of slave agent: slv
Enter name of sequencer: seq2
Enter name of driver: drv2
Enter name of monitor: mon2
Enter name of interface: if2

Choose one of the following RAL adpaters:
1) RAL sequence adapter, single domain
2) RAL sequence adapter, multiplexed domains
select [1-2]:2

Enter Driver information for the slave agent slv ::
Choose one of following driver available
1) Driver, PUSH DRIVER (uvm_push_driver)
2) Driver, PULL DRIVER (uvm_driver)
Select [1- 2] [Default: 2]:

Enter Driver information for the master agent mas ::
Choose one of following driver available
1) Driver, PUSH DRIVER (uvm_push_driver)
2) Driver, PULL DRIVER (uvm_driver)
Select [1- 2] [Default: 2]:

Would you like to implement scoreboard?
Select(y/Y/n/N) [Default: y]:y
```

```
Enter Name of Scoreboard Class:scbd

This will result in the following:

The testcase generated is env1_test
------------------------------------------------------------
            Template generation completed.
------------------------------------------------------------
Usage notes :
1) Find the generated files in "proj/env1" directory.

2) Makefile has been placed in run "proj/env1/run" directory.

3) Edit files and look for comments marked "ToDo:" and fill
   in the application-specific behavior for your function.

------------------------------------------------------------
```

Figure 1-1 Generated Directory Structure for Complete Environment Generation

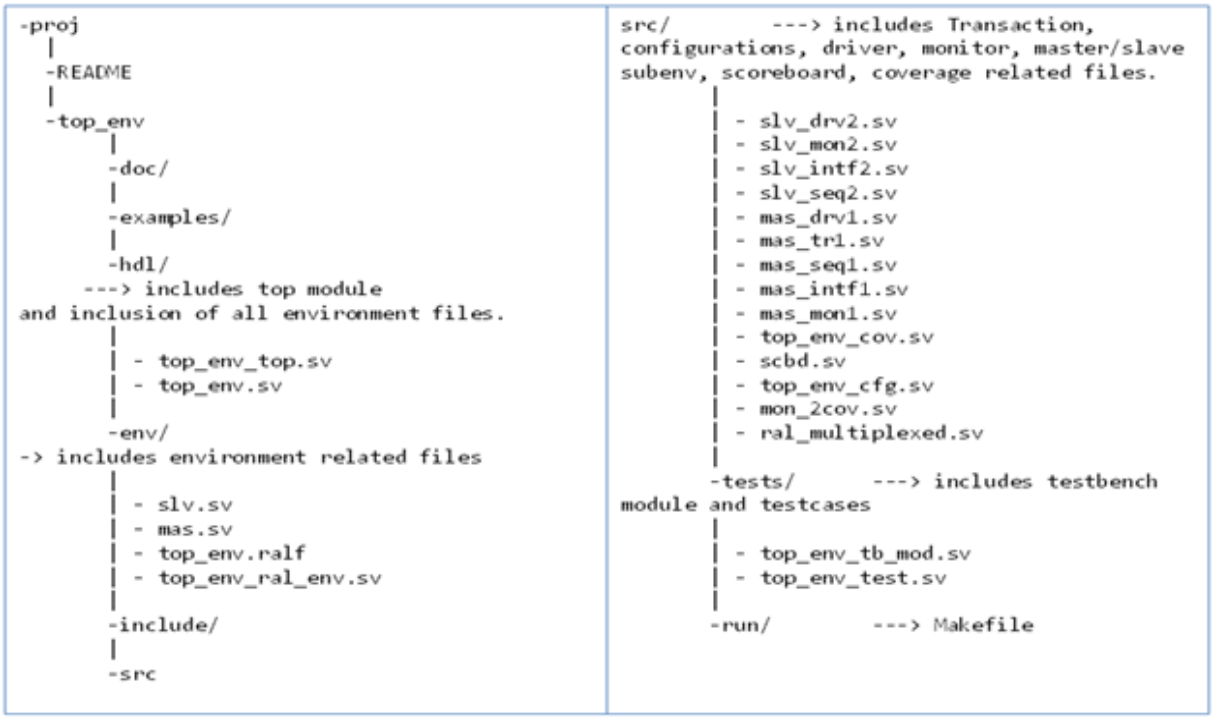
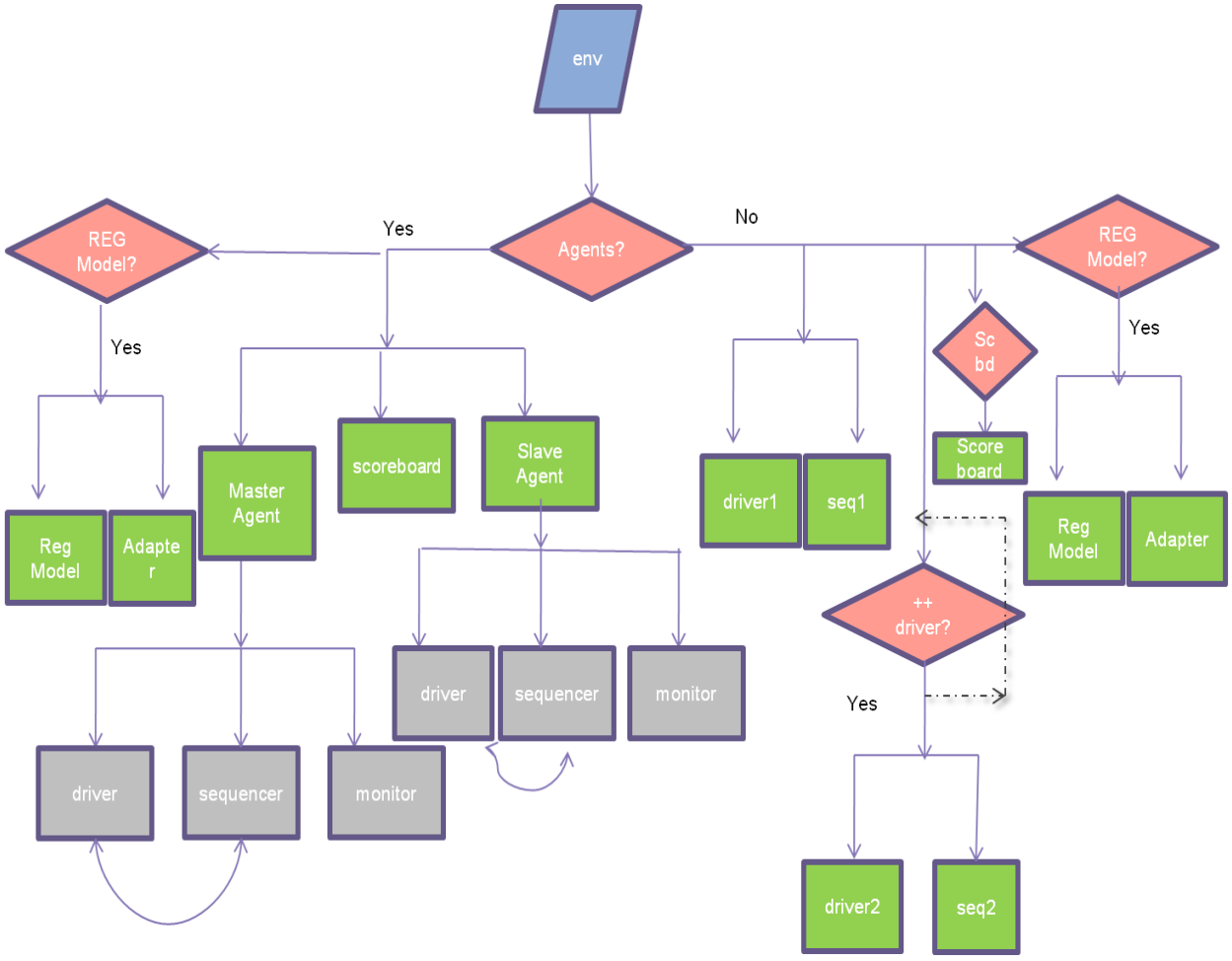


Figure 1-2 shows explicitly how `uvmgen` sets up the environment creation:

Figure 1-2 Flowchart



Complete Environment Generation (Quick Mode)

`uvmgen` also provides a quick mode for complete environment generation. In this mode, the minimum questions are asked and you can also pass all the options directly on the command line.

For example,

```
uvmgen -q
```

`uvmgen -RAL=y -ENV=apb -TR=apb_data+ahb_data -SE=n -q:` will generate an environment without further questions.

Table 1-2 lists the various options which can be passed in quick mode (`uvmgen` will ask for that field if it is not specified through the command line, however, it is required for generating the complete environment).

Table 1-2 Options Supported in Quick Mode

-SE	Associate master and slave agents with environment [argument :y/n].
-RAL	Associate RAL environment [argument :y/n].
-ENV	Switch to provide name of environment [argument: env name].
-TR	Switch to provide name of transaction class, multiple transaction file can be generated using '+' operator. i.e. -TR apb+atm.
-BU	Switch to have the classes extending from a Business Unit Layer which extends from the UVM base class e.g -BU bu

Individual Templates Generation Mode

This option is used to generate individual templates corresponding to different UVM components.

For example,

```
% uvmgen
```

You will now see the following two options:

1) Enter 1 to Create Complete Environment

2) Enter 2 to Generate Individual Template

Choose 2 to generate individual templates.

The following are the various individual templates available with uvmgen:

- Physical interface declaration
- Transaction descriptor
- Driver
- Generic Master Agent
- Generic Slave Agent
- Monitor
- RAL Adapter Sequence, single domain
- RAL Adapter Sequence, multiplexed domains
- Verification Environment (RAL or non-RAL)
- Verification Environment with Agents (RAL or non-RAL)
- Test case
- Agent
- Program block
- UVM sequence library
- UVM Sequencer
- Transaction descriptor test case
- UVM Scoreboard
- Top Module
- Coverage Template

- Monitor To coverage Connector
- Configuration descriptor

The templates are located in $VCS_HOME/etc/uvm_template/shared/lib/templates/uvm-1.0.

Note:The templates are compliant for both UVM-1.0 and UVM-1.1.

As mentioned, all the best practices and UVM recommendations are catered to when the templates are created. Figure 1-3 is a snapshot of the generated template for a generic master agent.

Figure 1-3 Generated Template for a Generic Master Agent

Using Configuration File

At times, you might want to have a non-interactive interface so that dynamic generation of templates can be done through other scripts. In such scenario, a `configuration` file can be used to specify the various options.

Table 1-3 lists the various options which can be passed by using the `configuration` file.

(UVMGEN will ask for that field if it is not specified and is required)

For example,

```
uvmgen -cfg_file cfile
```

Table 1-3 Options Supported by Using Configuration File

-SE	Associate master and slave agents with environment [argument :y/n].
-RAL	Associate RAL environment [argument :y/n].
-ENV	Switch to provide name of environment [argument:env name].
-TR	Switch to provide name of transaction class, multiple transaction file can be generated using '+' operator. i.e. -TR apb+atm.
-BU	Switch to have the classes extending from a Business Unit Layer which extends from the UVM base class e.g -BU bu

Here is the template of the configuration file template

```
ENV = ENV

PI = Y

SE  = Y

RAL = N
```

```
TR   = TR1+TR3
```

Inserting User Code

When the templates are generated, you can walk through the code and start inserting 'user' code or functionality in the specified places. In the generated code, you need to check for comments starting with “ //ToDo:" and fill in the application specific functionality. When you go for the complete environment generation, `uvmgen` will generate the Makefile in the run directory. This would compile and simulate the environment. You can then customize by adding additional options to the compile and run commands.