

# Online UVM/OVM Methodology Cookbook: Registers/Overview

by Mark Peryer, Verification Methodologist, Mentor Graphics Corporation

## INTRODUCTION

The UVM register model provides a way of tracking the register content of a DUT and a convenience layer for accessing register and memory locations within the DUT.

The register model abstraction reflects the structure of a hardware-software register specification, since that is the common reference specification for hardware design and verification engineers, and it is also used by software engineers developing firmware layer software. It is very important that all three groups reference a common specification and it is crucial that the design is verified against an accurate model.

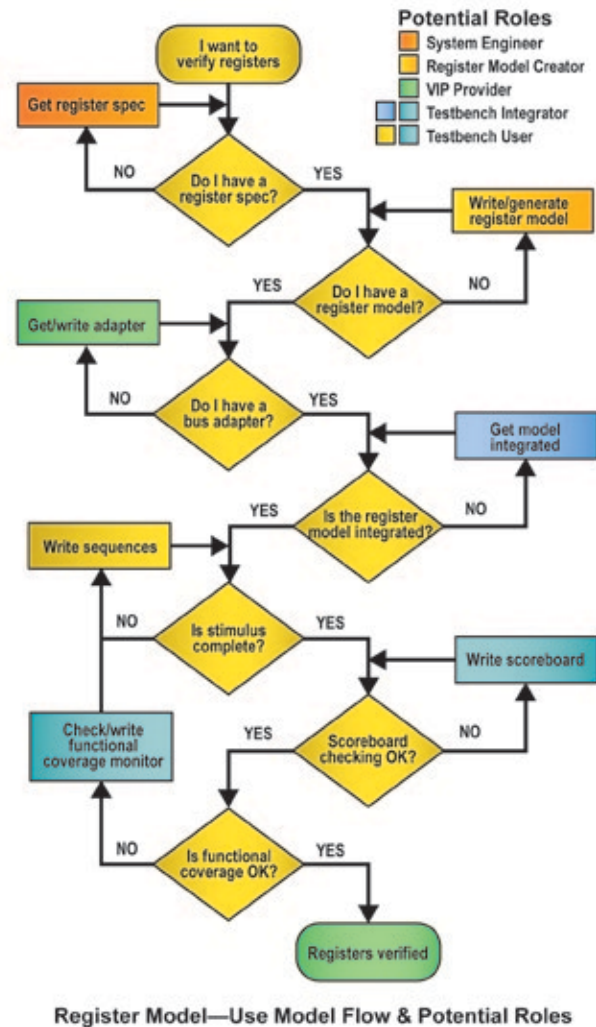
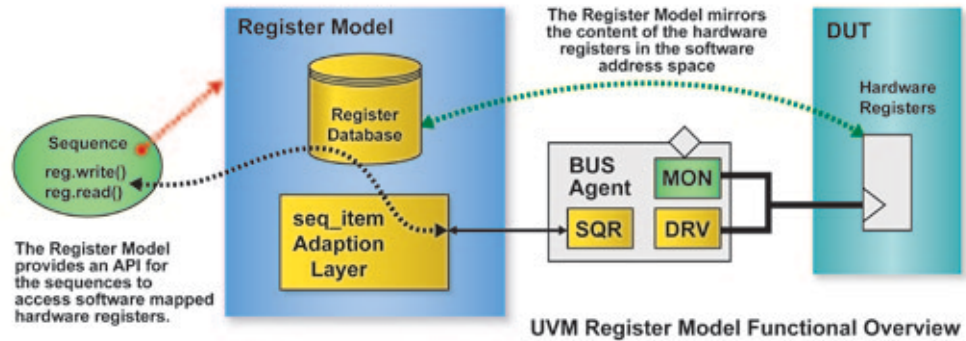
The UVM register model is designed to facilitate productive verification of programmable hardware. When used effectively, it raises the level of stimulus abstraction and makes the resultant stimulus code straight-forward to reuse, either when there is a change in the DUT register address map, or when the DUT block is reused as a sub-component.

## HOW THE UVM REGISTER MATERIAL IS ORGANIZED

The UVM register model can be considered from several different viewpoints and this page is separated into different sections so that you can quickly navigate to the material that concerns you most. The diagram to the right summarizes the various steps in the flow for using the register model and outlines the different categories of users.

Therefore, the different register viewpoints are:

- The VIP developer
- The Register Model writer
- The Testbench Integrator
- The Testbench User



## VIP DEVELOPER VIEWPOINT

In order to support the use of the UVM register package, the developer of an On Chip Bus verification component needs to develop an adapter class. This adapter class is responsible for translating between the UVM register packages generic register sequence\_items and the VIP specific sequence\_items. Developing the adapter requires knowledge of the target bus protocol and how the different fields in the VIP sequence\_item relate to that protocol.

Once the adapter is in place it can be used by the testbench developer to integrate the register model into the UVM testbench.

To understand how to create an adapter the suggested route through the register material is:

Step	Page	Description	Relevance
1	<a href="#">Integrating</a>	Describes how the adaptor fits into the overall testbench architecture	Background
2	<a href="#">integration</a>	Describes in detail how the adaptor is used	Background
3	<a href="#">Adapter</a>	How to implement a register adaptor, with an example	Essential

## CREATING A REGISTER MODEL

A register model can be created using a register generator application or it can be written by hand. In both cases, the starting point is the hardware-software register specification and this is transformed into the model.

If you are using a generator or writing a register model based on a register specification then these topics should be followed in this order:

Step	Page	Description	Relevance	
			Using Generator	Writing Model
1	<a href="#">Specification</a>	Overview of Register Specification	Background	Background
2	<a href="#">RegisterModelOverview</a>	Register Model Hierarchy Overview	Useful background	Essential
3	<a href="#">ModelStructure</a>	Implementing a register model	Background	Essential
4	<a href="#">QuirkyRegisters</a>	Implementing 'Quirky' registers	Essential	Essential
5	<a href="#">ModelCoverage</a>	Adding coverage models	Background	Essential
6	<a href="#">BackdoorAccess</a>	Using and specifying back door accesses	Background	Essential
7	<a href="#">Generation</a>	Generating a register model	Essential	Unecessary

## INTEGRATING A REGISTER MODEL

### Integration Pre-requisites

If you are integrating a register model into a testbench, then the pre-requisites are that a register model has been written and that there is an adaptor class available for the bus agent that is going to be used to interact with the DUT bus interface.

### Integration Process

In the testbench, the register model object needs to be constructed and a handle needs to be passed around the testbench environment using either the configuration and/or the resource mechanism.

In order to drive an agent from the register model an association needs to be made between it and the target sequencer so that when a sequence calls one of the register model methods a bus level sequence\_item is sent to the target bus driver. The register model is kept updated with the current hardware register state via the bus agent monitor, and a predictor component is used to convert bus agent analysis transactions into updates of the register model, pictured at the top of the next page.

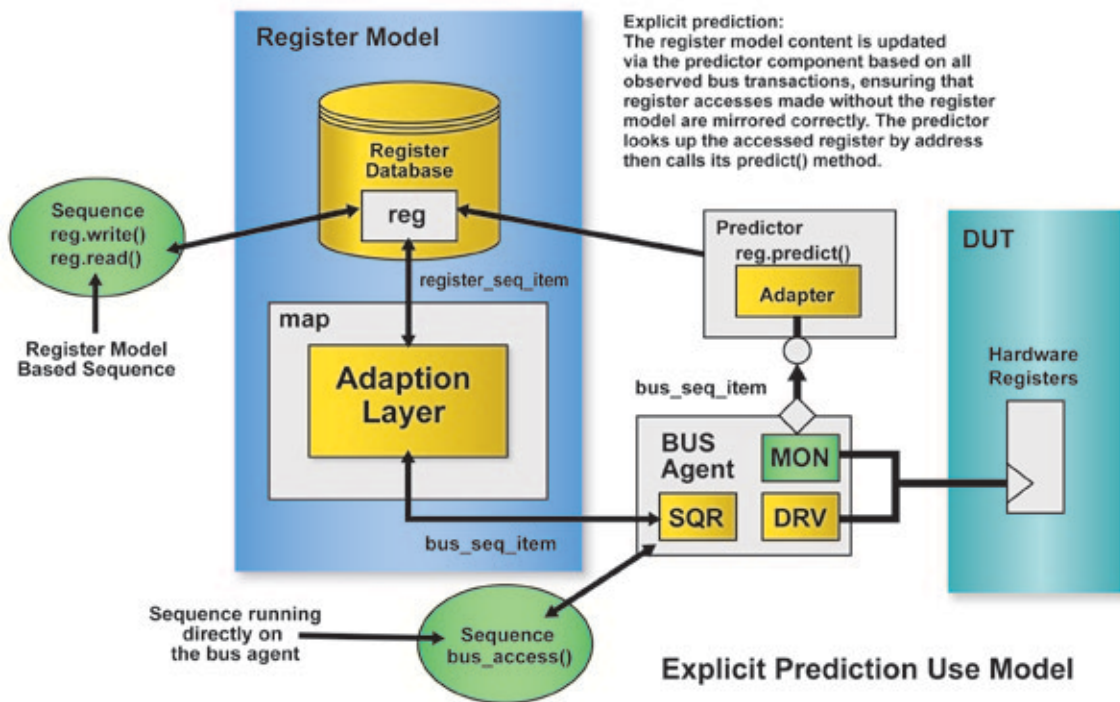
The testbench integrator might also be involved with implementing other analysis components which reference the register model, and these would include a scoreboard and a functional coverage monitor.

For the testbench integrator, the recommended route through the register material is outlined in the table to the right:

## USING A REGISTER MODEL

Once it has been integrated, the register model is used by the testbench user to create stimulus using sequences or through analysis components such as scoreboards and functional coverage monitors.

The register model is intended to make it easier to write reuseable sequences that access hardware



registers and areas of memory. The model data structure is organized to reflect the DUT hierarchy and this makes it easier to write abstract and reusable stimulus in terms of hardware blocks, memories, registers and fields rather than working at a lower bit pattern level of abstraction. The model contains a number of access methods which sequences use to read and write registers. These methods cause generic register transactions to be converted into transactions on the target bus.

The UVM package contains a library of built-in test sequences which can be used to do most of the basic register and memory tests, such as checking register reset values and checking the register and memory data paths. These tests can be disabled for those areas of the register

or memory map where they are not relevant using register attributes.

One common form of stimulus is referred to as configuration. This is when a programmable DUT has its registers set up to support a particular mode of operation. The register model can support auto-configuration, a process whereby the contents of the register model are forced into a state that represents a device configuration using constrained randomization and then transferred into the DUT.

The register model supports front door and back door access to the DUT registers. Front door access uses the bus agent in the testbench and register accesses use the normal bus transfer protocol. Back door access uses

simulator data base access routines to directly force or observe the register hardware bits in zero time, by-passing the normal bus interface logic.

As a verification environment evolves, users may well develop analysis components such as scoreboards and functional coverage monitors which refer to the contents of the register model in order to check DUT behaviour

Step	Page	Description	Relevance
1	<a href="#">RegisterModelOverview</a>	Overview of the register model hierarchy	Useful to understand terminology
2	<a href="#">Integrating</a>	Overview of the register model stimulus and prediction architecture	Essential
3	<a href="#">Adapter</a>	Adapter implementation detail	Useful background
4	<a href="#">integration</a>	Register model integration detail	Essential
5	<a href="#">Scoreboarding</a>	Scoreboard implementation	Useful background
6	<a href="#">FunctionalCoverage</a>	Coverage implementation	Useful background



or to ensure that it has been tested in all required configurations.

If you are a testbench consumer using the register model, then you should read the following topics in the recommended order:

Step	Page	Description	Relevance
1	<a href="#">Specification</a>	Register Specification	Background
2	<a href="#">RegisterModelOverview</a>	Register Model Hierarchy Overview	Essential to understand the terminology
3	<a href="#">Integrating</a>	Register Model Testbench architecture	Background
4	<a href="#">StimulusAbstraction</a>	Stimulus Abstraction for registers	Essential
5	<a href="#">MemoryStimulus</a>	Memory stimulus abstraction	Essential
6	<a href="#">BackdoorAccess</a>	Back door accesses	Relevant if you need to do backdoor accesses
7	<a href="#">SequenceExamples</a>	Example sequences	Essential
8	<a href="#">Configuration</a>	How to configure a programmable DUT	Essential
9	<a href="#">BuiltinSequences</a>	How to use the UVM built-in register sequences	May be relevant
10	<a href="#">Scoreboarding</a>	Implementing a register model based scoreboard	Important if you need to maintain a scoreboard.
11	<a href="#">FunctionalCoverage</a>	Implementing functional coverage using the register model	Important if you need to enhance a functional coverage model

## REGISTER MODEL EXAMPLES

The UVM register use model is illustrated by code excerpts which are taken from two example testbenches. The main example is a complete verification environment for a SPI master DUT, in addition to register model this includes a scoreboard and a functional coverage monitor, along with a number of test cases based on the use of register based sequences. The other example is designed to illustrate the use of memories and some of the built-in register sequences from the UVM library. Download links for these examples are provided in the table below:

Example	Download Link
SPI Master Testbench	 Download a complete working example (as used by <a href="#">these pages</a> ) (tarball: <a href="#">spi_bi_reg_tb.tgz</a> )
Memory Sub-System Testbench	 Download a complete working example (as used by <a href="#">these pages</a> ) (tarball: <a href="#">mem_example.tgz</a> )

**Editor's Note:** This article is an excerpt from the *Online UVM/OVM Methodology Cookbook*, available via Mentor Graphics' Verification Academy (<http://verificationacademy.com>)