A New Class Of Registers

Mark Peryer, David Aerne – Mentor Graphics

- ***Are you frustrated with the UVM Register Model?***
  - ***Are you using protocols like PCIe and AMBA® but can't use all their features?***
- ***Would you like the register model to work as advertised?***
  - ***When you make a burst API call, you actually get a burst transfer***
- ***Here is a simple way to support advanced protocol features***
  - ***But still be able to use the abstractions of the UVM Register Model***

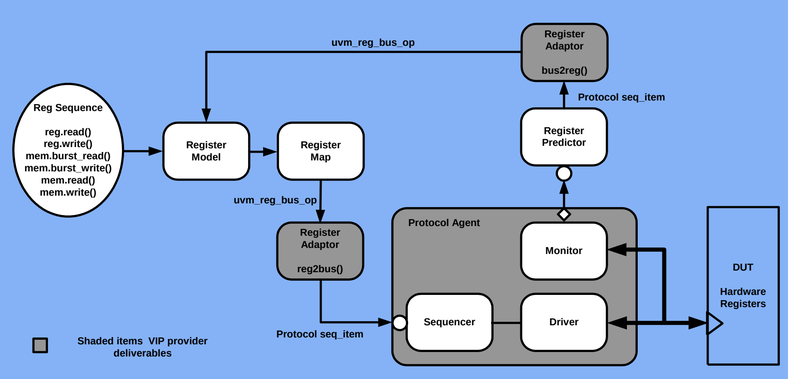
Overview and problem statement

The UVM register model is useful for abstracting stimulus and shadowing observed hardware register state, but its front door access use model is limited to simple blocking transfers. Each register or memory read or write is modelled as a single blocking transfer.

For most protocols, it is possible to use simple blocking transfers to provide register and memory transfers, but often there are other, more advanced, capabilities available within the protocol that would enable the testbench code to run more efficiently. If these capabilities were accessible from the register layer, then it could also improve the verification coverage.

The objective of this poster is to show how the register model can be extended to handle the capabilities of advanced bus protocols with a relatively small change to the code normally used to integrate a register model into a UVM testbench.

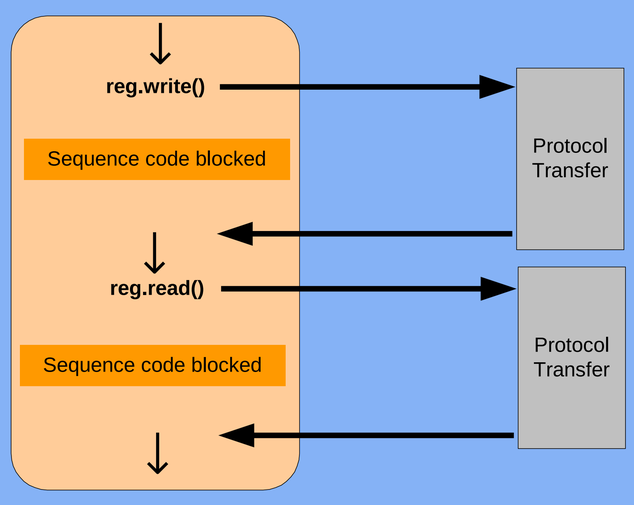
Integrating The Register Model In A UVM Testbench



The UVM register model is usually integrated into a UVM testbench using a register adaptor delivered by the VIP provider. The adaptor is responsible for converting a single uvm_reg_bus_op struct to/from a sequence_item for the target protocol agent. When a test sequence makes a register read() or write(), the address information for the register is looked up in the register map and then the access is made via the protocol agent. The register model is kept up to date via the predictor which uses monitored transfers to update the content of the register model.

Understanding The Register Model Limitations

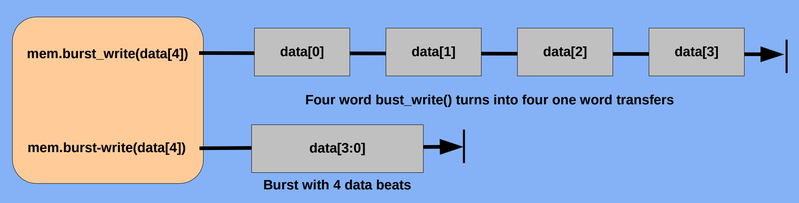
The UVM register layer only ever does a single blocking transfer.



In the sequence code, execution is blocked whilst a transfer is taking place. In an accelerated testbench, the sequence could be doing other work instead of being blocked, and in some protocols there are non-blocking transfers available.

Understanding The Register Model Limitations

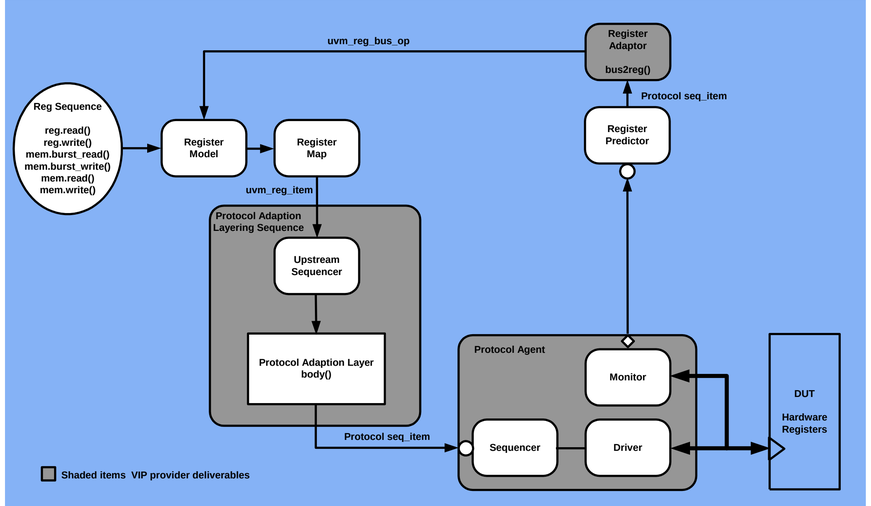
The register layer breaks down burst transfers into a series of single transfers. For large block transfers of data, this quickly becomes inefficient.



There are other aspects of protocol behavior that the UVM register model does not handle since it assumes a simple request-response model. For example, in PCIe a single request may result in several responses before the transfer is complete.

However, the UVM register model can easily be extended to handle more advanced protocol semantics by using a protocol layering sequence in place of the uvm_reg_adapter.

Using A Protocol Adaption Layering Sequence



The protocol layering sequence contains a handle to the sequencer that the register model sends uvm_reg_items to. The body() method of the sequence gets uvm_reg_items and converts them to the target protocol sequence_items. This approach has several advantages over the standard adaptor based approach:

- The protocol adaptor sequence is long-lived and is able to keep track of context
- It receives uvm_reg_items which are objects that contain more information than the uvm_reg_bus_op structs

The useful fields from the uvm_reg_item

Field	Purpose
element_kind	REG, MEM or FIELD
element	Handle for the originating register or memory in the register model
kind	READ or WRITE
value	The data field, expressed as an array allowing burst support
parent	Handle for the originating sequence, allowing its methods to be called to return data
offset	Valid for memory transfers
extension	Handle to extension object that can be used to pass protocol specific information between the parent sequence and the adaption layer

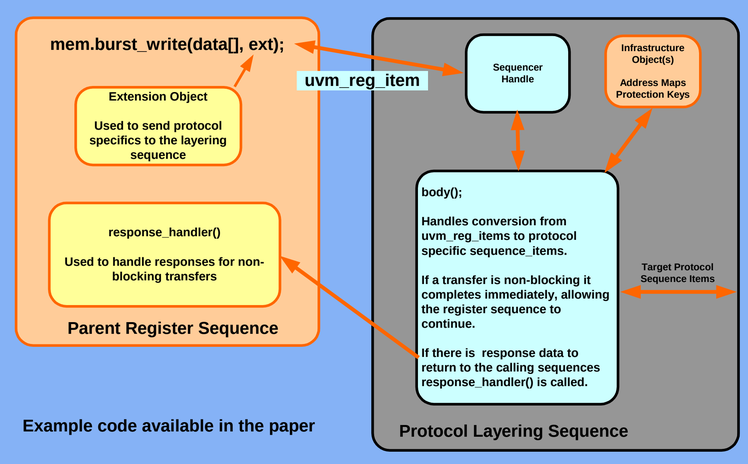
Application Examples – AMBA® AXI™

The protocol adaption layering sequence has been used with AXI bus VIP to enable support of the following features of the protocol from the register layer:

- True burst support
- Non-blocking transfers
- Multiple concurrent transfers
- Out of order completion
- Barrier transactions
- Meaningful support of extended bus fields
  - Protection
  - Cache access
  - QoS

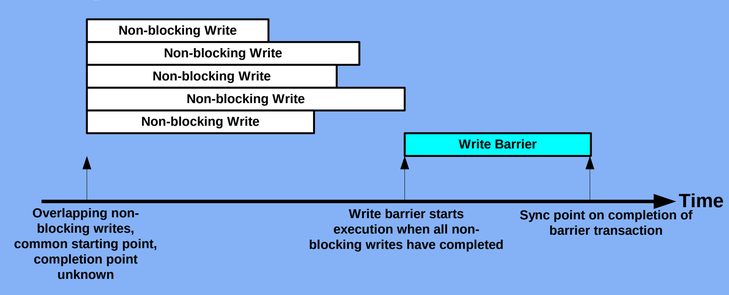
The AXI protocol adaption layering sequence can access infrastructure objects such as system address maps to determine the correct settings for fields such as AxPROT and AxCACHE.

Exactly which completion semantic is required can be coded into the extension object. If a non-blocking transfer is used, then the response can be routed back to a response handling routine in the parent sequence using the parent handle in the uvm_reg_item.



Example code available in the paper

Several different completion semantics can be supported, allowing full exploration of concurrent transfers on both the AXI read and write channels with out of order completion via the register model memory API. The barrier semantic is a useful extension for synchronization between blocks of non-blocking transfers.



Application Examples - PCIe

Although the UVM register layer can be used with PCIe, it is typically fairly inefficient. Using a protocol adaption layering sequence allows the following aspects of the protocol to be exploited:

- Bulk data transfers
- Handling of multiple concurrent requests
- Handling and collation of multiple completions
  - Returning complete transport packet to parent
- Using additional PCIe transaction fields – e.g:
  - ATTR – Affecting completion ordering
  - TC – Traffic Class
- Credit Management

mark_peryer@mentor.com, david_aerne@mentor.com

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Mentor Graphics®