

## Register package over simple bus example

### Introduction

This example shows how to connect the register package over a very simple bus, called `simple_bus`. Although it is very simple, the simple bus has everything you would expect to find in a standard OVC – a driver, monitor, sequencer, agent, environment, transaction, and so on. Therefore, extending this example to any other standard OVC should be quite easy.

The register package is used both to generate reads and writes to registers on the simple bus, and to monitor and check the values that are read back from the registers. The sections below describe the “generator” direction and the “monitor” direction in detail.

### Files

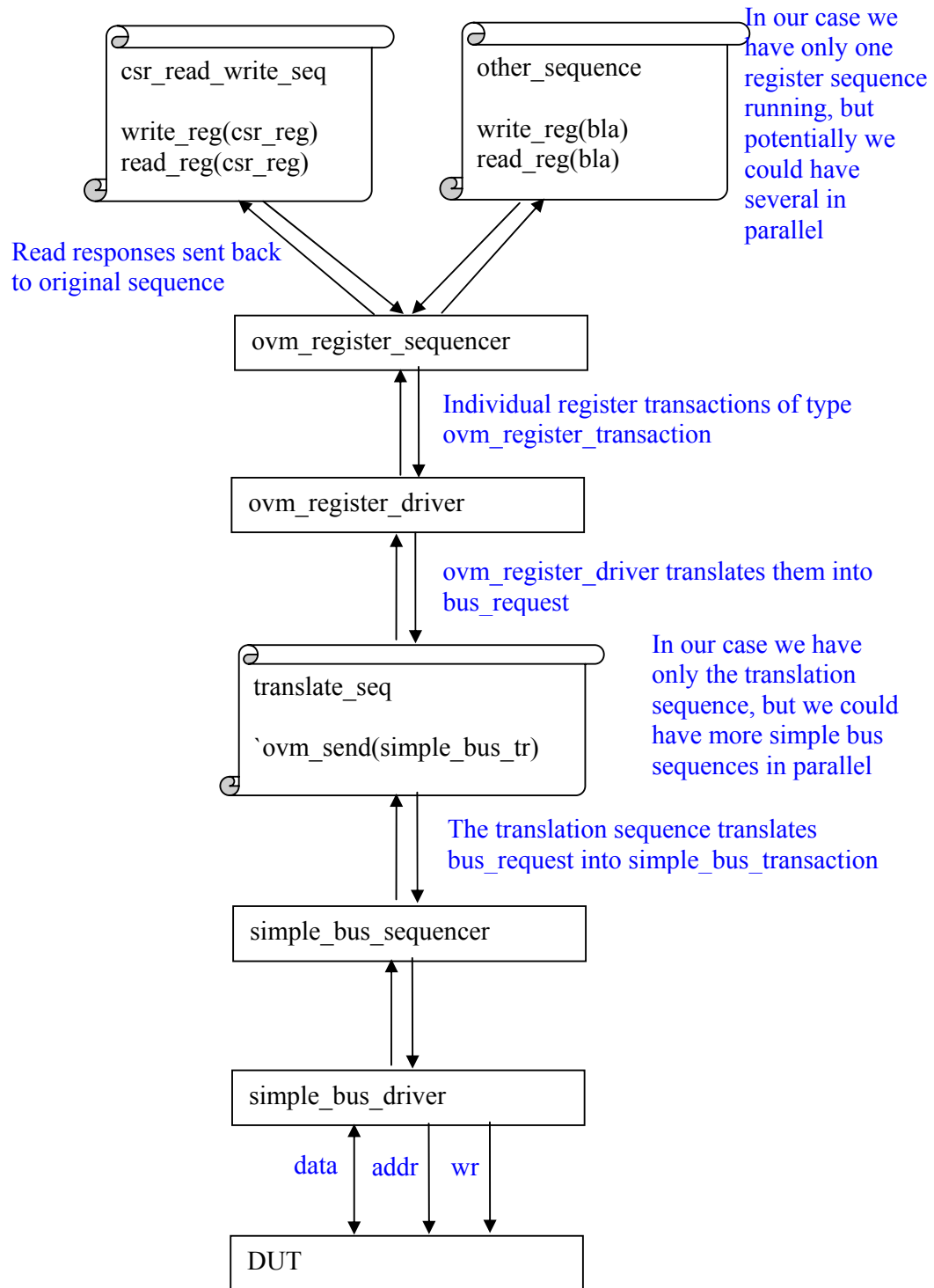
File name	Description
<code>simple_bus.sv</code>	Contains all the code for the simple bus OVC
<code>t.sv</code>	Defines the top level environment, the test, and the glue logic that is required to connect the register package over the simple bus OVC. Also defines the top level module and an interface container that is used to pass an interface pointer into the class based world.
<code>regdef.sv</code>	Register definitions. This file is identical to the one in the stopwatch example
<code>dut.sv</code>	Defines a simple DUT that can be accessed using the simple bus interface

### Generator

In the “generator” direction, register transactions, originally coming from one or more register sequences, are sent via the register sequencer to a simple bus translation sequence. This sequence pulls the register transactions from the register sequencer and translates them into simple bus transactions. The simple bus transactions are then executed on the simple bus, and the responses are sent back to the original register sequence.

The diagram below shows all the stages in this path, and the classes participating in each.

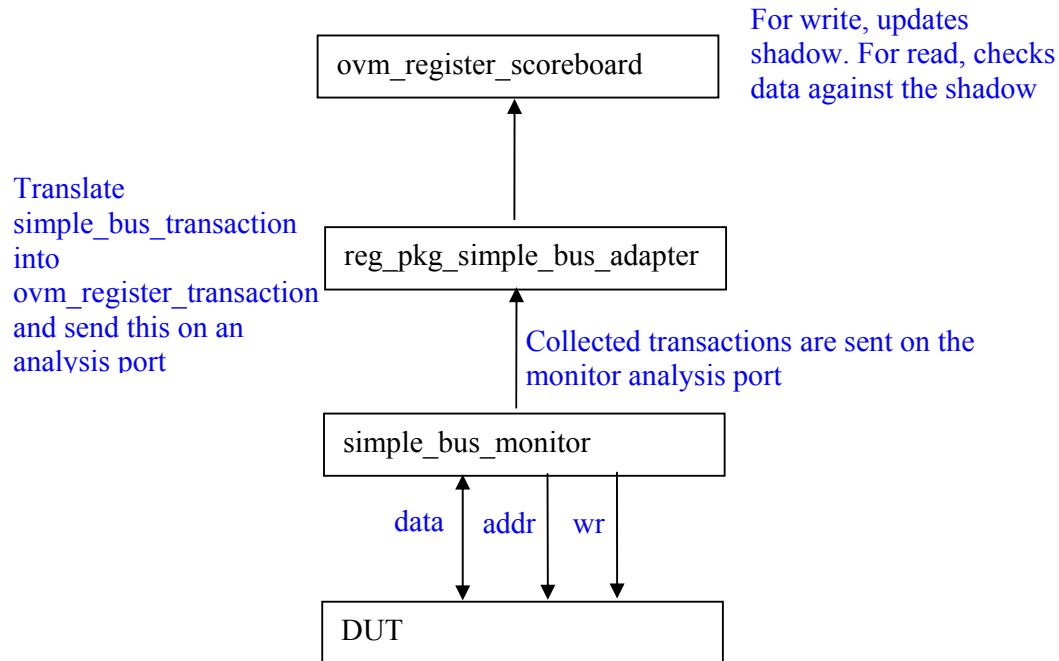
Note: In parallel to the sequence “`csr_write_read_seq`”, which is defined in the top level file `t.sv`, the register package will automatically start the sequence “`register_sequence_all_registers`”, which does a write-read-compare check for all the defined registers.



## Monitor

In the “monitor” direction, simple bus transactions are collected by the simple bus monitor and are sent out on an analysis port. This analysis port is connected to an adapter that translates the simple bus transactions into register transactions. The register transactions are then sent to the register package, in order to update the shadow registers (in case of write) or in order to be checked against the expected value (in case of read).

The diagram below shows this path and the classes participating in each stage:



## Simple bus protocol

The simple bus protocol is a memory-like protocol. Writes are executed in a single cycle, reads have an address cycle which is followed by a data cycle. When the pin “wr” is high then we have a write cycle, when it is low we have a read cycle, when it is ‘z’ nothing happens.

The simple bus is connected on one side to the simple bus OVC defined in the testbench, and on the other side to a DUT which implements a memory (dut.sv) file.

## Running the example

To run the example: (1) start questasim (2) type “do run\_questa” in command line (3) type “run -all”

In the wave you’ll see a write followed by a read to the CSR register. The read and the write come from the sequence **csr\_read\_write\_reg** which is defined in the **top\_level** package.

**Questions/Feedback:**

Please email:

[avidan\\_efody@mentor.com](mailto:avidan_efody@mentor.com)