Here's something for the less experienced verifiers out there. I've been asked to help with this issue several times in the past so I guess some of you will find it useful.

If you're applying the eRM/OVM-e methodology in your verification project, even if you're not fully compliant, you probably take advantage of the vr_ad package. For those who are less familiar with it - it's a library of base classes and macros that help you define and access registers very easily. I've worked with this package for quite a while and I'm rather pleased with it although it has its flaws, as with any verification package out there I guess. A couple of years ago I even met with the guy from Verisity (Cadence) who developed the package and we held a long and interesting discussion about it.

One of the problems is the lack of inherent ability to activate register access methods and sequences from anywhere in the environment, and not just from other sequences. As a matter of fact, the ability to launch sequences only from other sequences is a fundamental feature of the eRM methodology (some see it as advantage, others don't). But while non-register sequences (packet generation sequences, for example) are likely to be activated from dedicated areas in the environment (top level sequences, virtual sequences), register access is kind of different. Because register operations (reads, writes, reading shadow value or updating shadow value) tend to be frequent and almost orthogonal to everything else that's going on during simulation, it seems reasonable NOT to limit register access by the sequence mechanism and allow users to launch read and write operations from wherever they want (well, maybe not from EVERYWHERE, but you know what I mean).

One quick way to get around this is to use the built in sequence mechanism in the not recommended way. You will still have to encapsulate the lower level read/write operations in a vr_ad_sequence but you will be able to do it from anywhere in your environment, and not only from top-level sequences:

The following example assumes you have already defined a nice little sequence called **SIMPLE_WRITE**
(of type vr_ad_sequence), and let's say you want to call it from a legacy non-eRM compliant environment that has a TCM with the inspiring name of

drive_packets()
:

{code}

drive_packets(num: uint)@mac_clock_e is {

// initialization

// driving a bunch of packets here

// here we want to activate a pre-defined vr_ad sequence
var reg_seq1: SIMPLE_WRITE vr_ad_sequence;
gen reg_seq1 keeping {
.driver == <pointer to vr_ad driver>
// i.e. sys.top_env.virt_seq.cpu_driver
};
reg_seq1.start_sequence();

// driving some more packets

};

{/code}

Note that the driver should be constrained properly to the location of the desired vr_ad driver instance.

Now this is only a quick solution, although it should work well in most cases. Robust solutions would include a set of new macros that will enable one-line register access from anywhere in the environment with no extra effort.