How to Create Reusable Portable Testing and Stimulus Standard (PSS) Verification IP

Sharon Rosenberg

The System and Methodology Library is a proven PSS methodology that provides value to users for years on some of the most complex system projects in the world. But wait! How can that be?! PSS was just released last June.
The reality is that beyond the input format and syntax, the base-layer, attributes and modeling style is used by users for years and remain the same. The same design patterns can be applied with the PSS DSL, PSS C++, or the input format from which the previous two were derived from. There are hundreds of users world-wide that may or may not read the PSS LRM but have deep understanding of the PSS core-concepts and vast modeling experience using them in real-life projects.

Motivation

The Portable Stimulus and testing Standard (PSS) was released in June 2018. The PSS language adopted powerful concepts to describe scenario space rules in terms of a behavioral model, and use actions and activities to specify desired scenarios. Both the legality rules and activities can be consumed by a PSS tool to automate legal test variations, visualize scenarios, measure coverage and steer randomization to accomplish more of it. PSS concepts such as components, actions with flow objects and constraints, activities, resources, states machines, exec blocks were initially adopted by Cadence in 2011 and have been used for verification and validation projects ever since. Much knowledge, expertise and patterns were accumulated on how to accurately frame the scenario space and scenarios. The System Methodology Library (SML) provides more than seven years of accumulated experience in how to use PSS to accurately frame the scenario space and needed scenarios.

Note that SML is fully applicable for both PSS C++ and Domain Specific Language (DSL) input formats and can be used in mixed-models.

Who is the SML library for?

The guidelines and library can be used by novice PSS users before they start writing PSS code, and for both novice and experts as a productivity tool and as a framework for reuse.

TIP: While the SML guide is built to be self-explanatory, Cadence highly recommends reading the official intro of the PSS Language Reference Model (LRM). The language reference model can be downloaded free of charge from the Accellera website at:
http://accellera.org/downloads/standards/portable-stimulus

What's in SML?

The Accellera PSS provides native support to many traditional verification challenges. For example:

- Stimulus and traffic – PSS actions and activities provide strong solving, auto-completion of partial specification, self-checking by the model, and portability to all platforms.
- Self-checking strategies – While PSS can feed scoreboards and compare the results, the behavioral model itself can be used as the reference model.
- Extensibility – The PSS language enables Aspect-Oriented Programming (AOP) and the traditional Object-Oriented Programming (OOP) to enhance the model for any purpose without the need for factories. This is a significant improvement in power and ease of use.
- Constructs for specifying structure – Those who are familiar with the SystemVerilog Universal Verification Methodology (UVM) know that UVM defines UVM_components on top of the native SV language. PSS defines components natively and leverages the built-in structure to bind resource pools down the hierarchy.
- Debug/visualization – While PSS does not define standard graphical terms for visualization, the PSS Language Reference Manual (LRM) adopted all its fundamental concepts from the Unified Modeling Language (UML) and lends itself naturally to activity diagram definitions.
- Resources with attributes – the ability to describe pools of resources and resolve legal distributions of these in the needed use-cases
- State-machines features – enables declaring low-power state-machines and other system operation modes, associating their dependencies to other actions and delegate the entire scenario creation to PSS tools

So what's in the library?

SML provides canonical base-classes, actions, and types to enable very high degree of reuse without planning in advance across both projects and teams. The SML layer is generic enough and enables applicability and reuse in all platforms. Here are two examples to illustrate the power of the library.

Seamless integration of PSS models

Consider a camera team that defines a `capture` action that outputs a video to memory, and the multimedia team that develops a `decode_to_display` action that inputs a video from memory and sends it to the display.

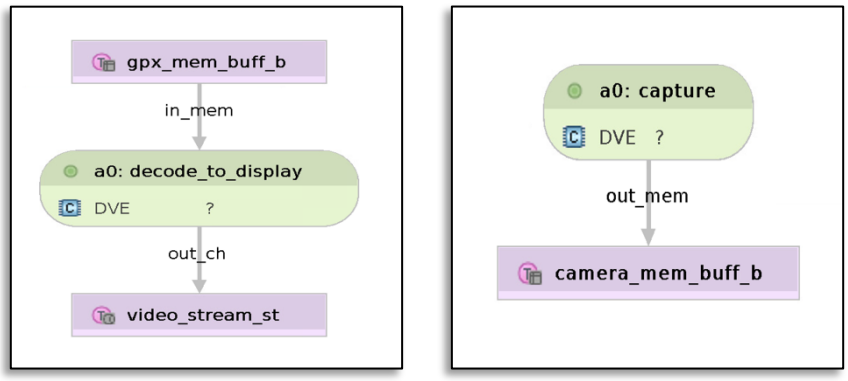

Figure 1: Two actions developed by different teams defining their own buffer types

How can the system or cluster integrator specify a scenario that combines actions for both?
A PSS tool will issue the following error:

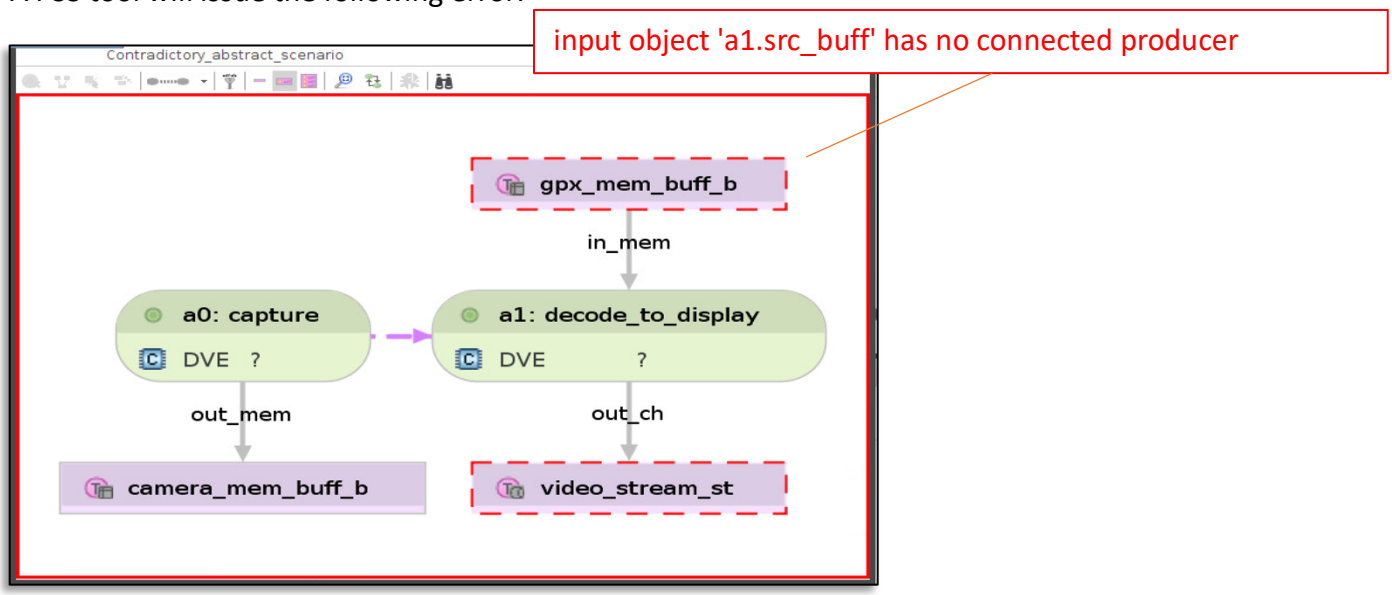

Figure 2: Tool error:no legal producer is found to `gpx_mem_buff_b`

Using the standard memory buffer definition solves the problem.

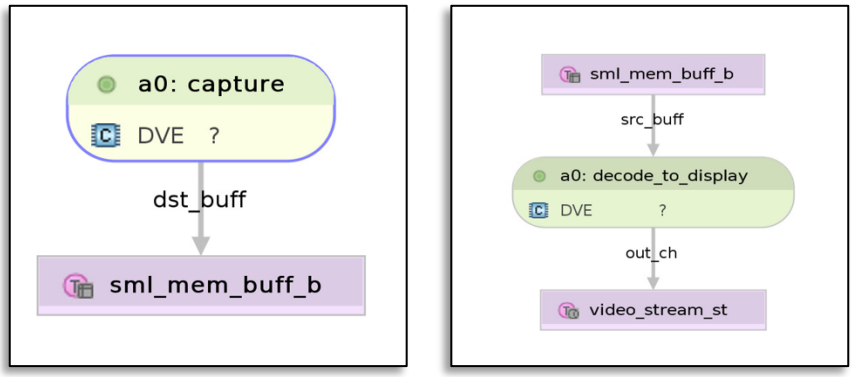

Figure 3: User actions using the `sml_mem_buff_b`

An example of the integrated solution is illustrated below:

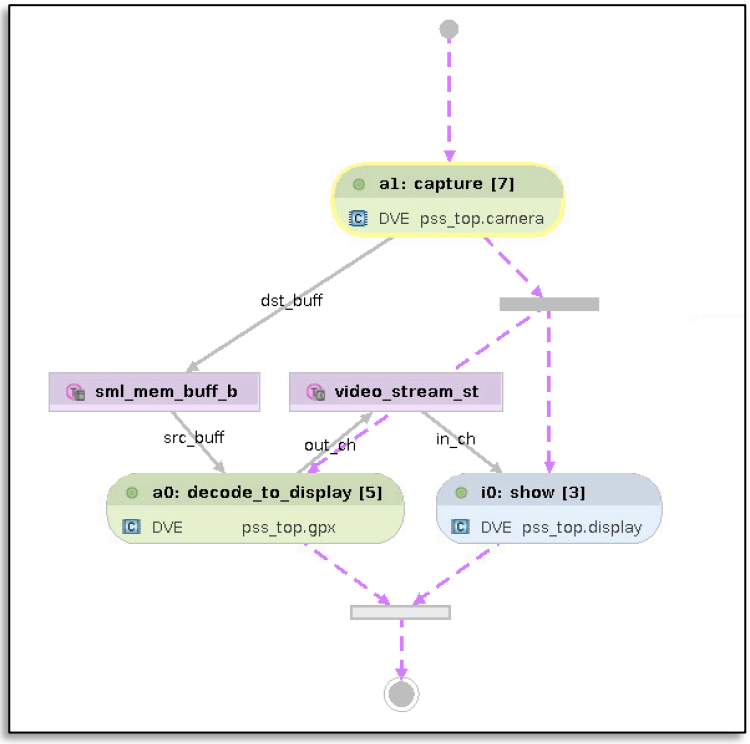

Figure 4: A solution combining user-actions using `sml_mem_buff_b`
Note that the `show` action is an inferred solution, since decode to display has an output stream.

Creation of abstract reusable PSS VIP

Assume that you want to create a PSS library for stress scenarios, and you start with this simple example.

```
extend component pss_top {
    import my_mem_ops_c::*;

    action stress_xfer {
        activity {
            schedule {
                do my_mem_ops_c::write_data;
                do my_mem_ops_c::write_data;
            }
            repeat (5) {
                do my_mem_ops_c::copy_data;
            }
            schedule {
                do my_mem_ops_c::read_check_data;
                do my_mem_ops_c::read_check_data;
            }
        }
```

```
    }
}
```
Example 1: A building block for integration/stress scenarios by copying information throughout the system.

On one hand, the scenario above is highly-reusable and demonstrates the power of PSS:

- It is topology-independent and relevant to a changing number of bus masters or CPU cores.
- You can execute multiple instances of it in parallel to create legal stress.
- It will generate resource-aware random ordering (for example, in UVM this would be a randomly-created virtual sequence).

On the other hand, what will happen if different teams model their own version of `read_check_data`, `write_data`, and `copy_data` actions and corresponding activities? While it is possible to create a mapping between atomic actions, activities like `stress_xfer` above that leverage specific action kinds will *not* be reusable.
In contrast, using common basic memory operations allows multiple teams to create reusable scenario libraries for coherency, performance, and more. The generic actions allow memory and register operations that can be used in both SW-driven verification (where programming is done with actual CPU cores) and in transactional testbenches (where verification IP agents program the various IPs).

PSS Golden Examples

The best way to learn the PSS and SML is by observing and learning complete examples. SML provides a set of system- and IP-level models that can be studied. The examples are PSS-compliant, open-source, and can be used with any PSS-compliant tool. The following is an example of a simple DMA model. It is not the entire golden example but demonstrates integration of user-defined IP model with the generic memory operations, actions and activities.

Basic DMA Device

This example represents devices that need to be programmed using simple C firmware or register sequence activation.

The testbench of this IP is illustrated in the following diagram:

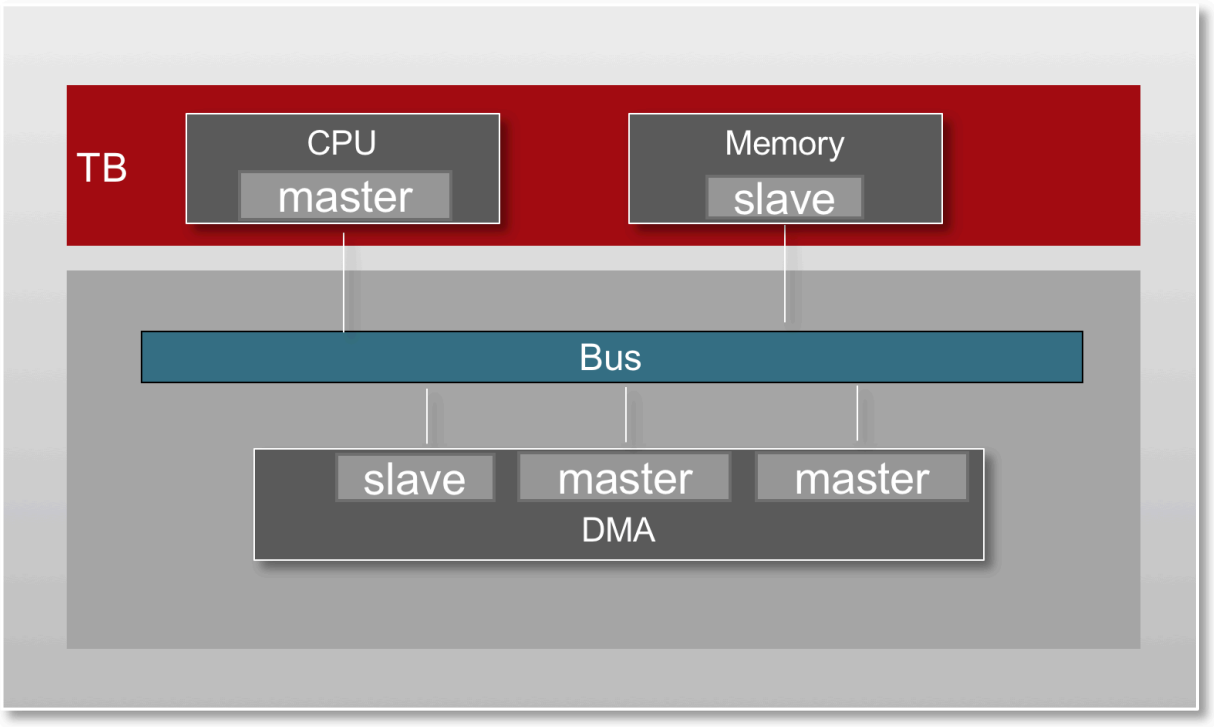

Figure 5: Sample DMA Device and Testbench

PSS model

The DMA model demonstrates resource-aware solving, as the same DMA channel will never be assigned to perform two tasks at the same time. Here is a quick description of the PSS actions that are needed for this setup:

- DMA `transfer` action – The main operation of the DMA is to copy data from one location to the other. This translates into an input buffer and an output buffer (for simplicity we focus on memory-to-memory transfers – but similar data flow applies to other forms of DMA transfer). For resource distribution, the transfer action also locks one of the channels.
- Master/CPU actions – Masters can perform `write_data`, `read_data`, or `copy` actions that involve accessing memory buffers.
- Modeling the memory – The memory can be instantiated but can also be emulated via a UVM slave with a sparse memory model. Beyond initial programming, the memory does not need to be programmed, and is thus abstracted away from the PSS model.
- Testbench backdoor load – In UVM we might want to consider backdoor loading the memory with data to save initialization cycles. This could be a backdoor load action. It is also possible to implement a backdoor option for the read, write, and copy actions.

The objects in all the actions are buffers. A master or backdoor operation can initialize memories that might be fetched (or not) multiple times later on. Using the sml_`mem_buff_b` object is the right choice to leverage the built-in attributes and to allow seamless connection between the standard SML actions to connect to the DMA `transfer` action.

The following figure illustrates the PSS model:

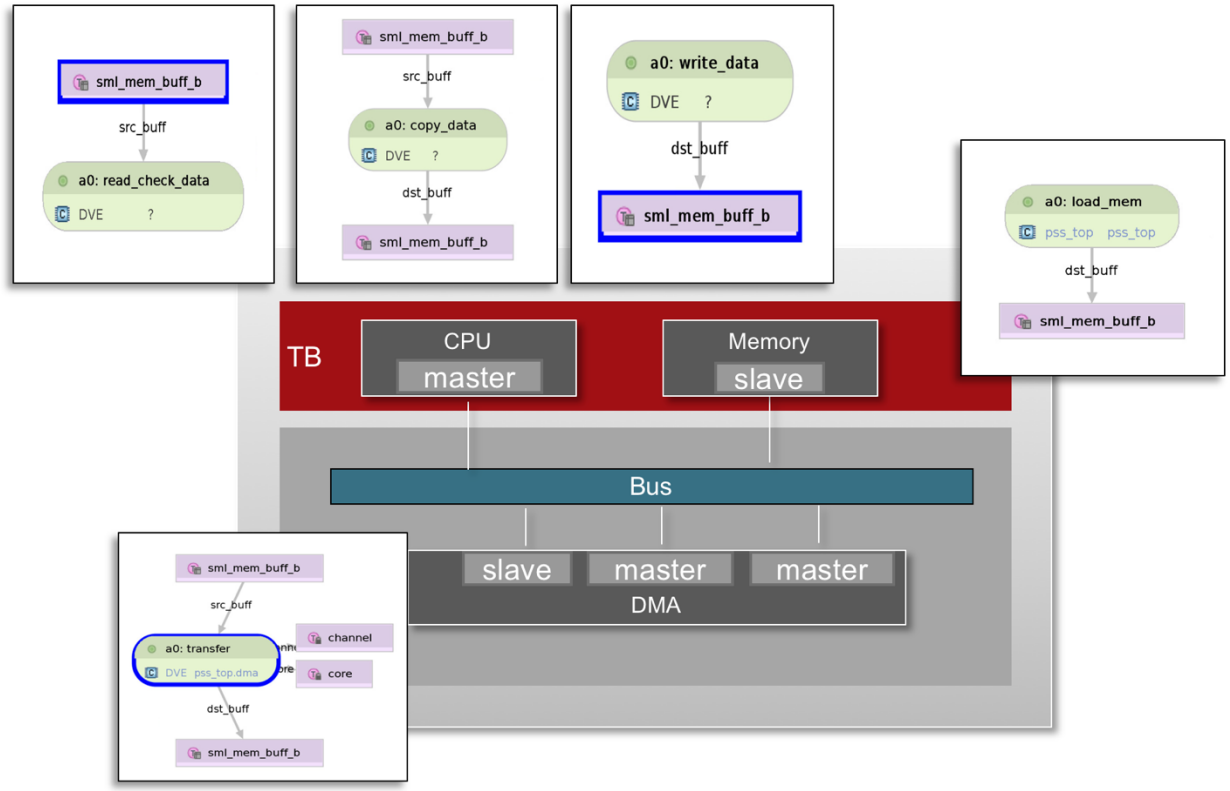

Figure 6: DMA and TB PSS model

Note the usage of the generic `write_data`, `copy_data`, and `read_check_data` that are used to verify the DMA device.
Leveraging the SML actions, you need to model only the DMA and instantiate it within PSS top.

```
component dma_c {
    resource channel_s {};
    pool[2] channel_s channels_p;
    bind channels_p *;

    action tranfer {
        input sml_mem_buff_b src_buff;
        output sml_mem_buff_b dst_buff;
        lock channel_s channel;

        constraint propagate_data_attrs {
            src_buff.data.size == dst_buff.data.size;
            src_buff.data.kind == dst_buff.data.kind;
        };
    };
};
```

Example 2: Leverage SML to focus on DMA actions model and instantiate within PSS top

The following code shows an example of the user-defined DMA model mixed with the pre-provided SML actions. In this case, we enhance the stress scenario to optionally use the DMA transfer action.

```
action my_stress_xfer {
    activity {
        schedule {
            do sml_mem_ops_c::write_data;
            do sml_mem_ops_c::write_data;
        }
        repeat (5) {
            select {
                do sml_mem_ops_c::copy_data;
                do dma_c::transfer;
            }
        }
        schedule {
            do sml_mem_ops_c::read_check_data;
            do sml_mem_ops_c::read_check_data;
        }
    }
```

Example 3: Bulding block for stress scenarios with DMA transfer action

SML Adoption and Experience

The reuse aspects of the base-layer and guidelines involve:

- Model reuse across teams and geographical places
- Scenario and regression reuse across multiple device generations
- Home-grown VIP and commercial VIP development

Since 2011, a long list of users such as Infineon, Qualcomm, Saumsung, ST micro-electronics, ARM, and more reported successful usage of the solution on multiple platforms and applications. These user case-studdies already use the PSS1.0 terminology and core-concepts as well as the mentioned base-layer to achieve automation. The same diagrams that you can find in the PSS LRM and the Accellera official tutorials, are used by these users to visualize the achieved use-cases for massive production projects. The same library and guidelines were adopted for different kinds of reuse.

If you are curious about PSS applications, wish to learn PSS modeling, and maximize its value and reuse, please contact Cadence or send an email to pss_info@cadence.com