PSS: The Promises and Pitfalls of Early Adoption

Mike Bartley, CEO, Test and Verification Solutions, Bristol, UK (mike@testandverification.com)

Abstract- **The new Accellera PSS (Portable Stimulus Standard) recently released at DAC suggests an exciting revolution, combining the best of the existing approaches and allowing for different implementations. There are already multiple portable stimuli technologies available which differ in many ways so the obvious question is now which horse to bet on?**

EDA history tends to show that the standard solutions won the industry while others evaporated while locking their users into proprietary technology. Adopting a technology early allows you to enjoy the automation sooner, build-up local expertise and even steer the standard to your own needs. However, there are risks that you lock into a solution that ultimately does not best match your requirements or that cannot be easily migrated or inter-operate with the future standard.

This paper is based on a deep analysis of the early adopter releases, and the Accellera PSS working group public announcements to provide a check-list on how to select a technology that will allow you to easily migrate to the recent PSS and maximize the capabilities of PSS. We will avoid discussing specific solutions but will instead give the audience a set of criteria for evaluating how to best to adopt PSS and a process to measure the existing industry solutions.

Keywords—Portable Stimulus Standard; PSS; Accellera;

I. Introduction

On June 26th 2018 the Accellera System Initiatives announced the release of the Portable Test and Stimulus Standard (PSS). The standard provides a language for capturing test scenarios and verification logic in an abstract, implementation-agnostic way, which can then be applied on multiple platforms and testbench implementations. At this point in time, there are several proprietary commercial solutions that try to address the portability challenge with different approaches but there is a huge gap between a portable stimulus tool and a PSS compliant tool. Whilst early adoption of a solution has many benefits, adopting the right solution is key. Consider, for example, adoption of the Accellera UVM standard where users went through verifying migration paths based on the previous methods or tools they had adopted (OVM, VMM, eRM, etc.). To date the three leading portable stimulus tool vendors have all announced full support of the standard, which should mitigate this concern and eliminate proprietary tool options. This paper introduces the PSS concepts and allows readers to make a knowledgeable, safe technology decision with the biggest return on investment and ability to migrate to the upcoming PSS releases once available. Beyond suggesting solutions criteria, we are also going to suggest an effective process for evaluating and adopting a PSS tool.

II. Motivation

A. What is PSS?

PSS is a standard input format and innovative paradigm for describing portable test scenarios. Similar to other languages, PSS allows the description of activities with loops, parallelism, conditional statements etc. but unlike other languages PSS also provides powerful semantics to capture the scenarios legality rules. Concepts like resources, data-flow objects (such as buffers and streams), state-machines and more allows for both the checking of the correctness of a provided scenario and the automated creation of a complete scenario from a user partial scenario request. While this paper is not a PSS tutorial, an example (using DMA allocation for peripherals) is introduced for the benefit of those who are not familiar with the standard. The DMA example is one of the formal usage examples that Accellera used to define the scope and demonstrate the standard capabilities.

1) *The DMA allocation for peripheral usage example*

a) *The challenge:*

Figure 1 shows a system with a four channel DMA engine that can move data between memory blocks, three UART devices that are DMA enabled (i.e. the DMA can copy information in and out of their queue), and a testbench that can initialize memory buffers using a backdoor access mechanism.

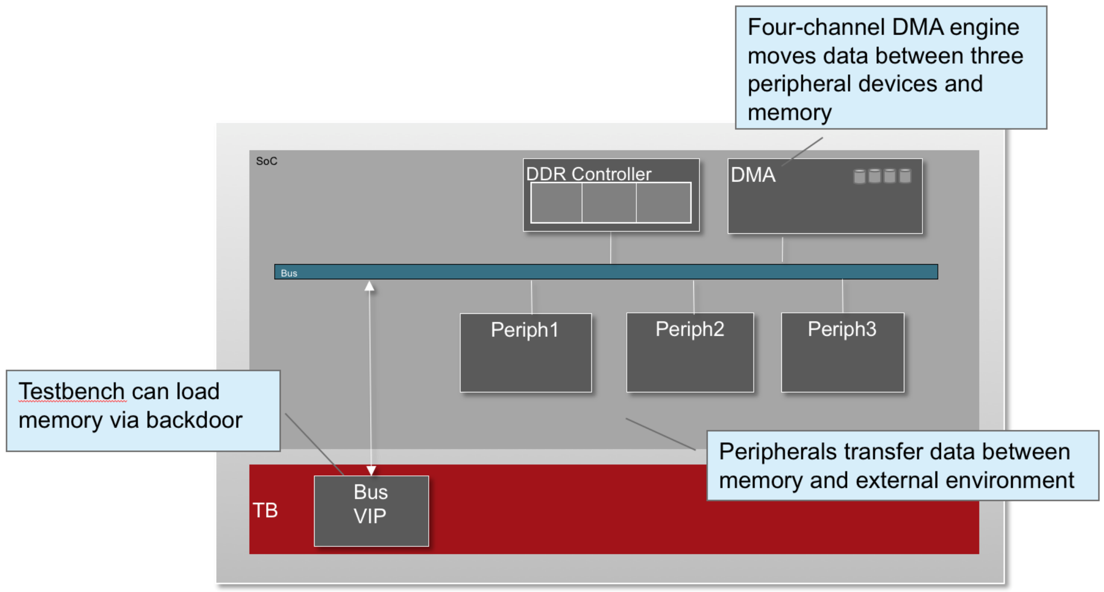

Four-channel DMA engine moves data between three peripheral devices and memory
SoC
DDR Controller
DMA
Bus
Periph1
Periph2
Periph3
Testbench can load memory via backdoor
Peripherals transfer data between memory and external environment
TB
Bus VIP

Figure 1: DMA allocation for Peripheral example

The challenge is to create scenarios to stress this system in multiple ways. While this might seem straight forward, we cannot just randomly configure the DMA engine to copy buffers between memory locations or activate the UART devices without taking the following considerations into account:

- A DMA channel must be available for the task
- If one of the UART devices is involved, then we need to ensure that it is not busy and the write or read starts in coordination with the loading of the specific UART queue.
- The DMA is copying from the queue in case of a UART read and copying into the queue in case of a write.
- The DMA is not copying uninitialized memory (to enable checking).
- The testbench backdoor initialization writes data to a location that is accessible to the DMA and in the right alignment.

All of these are just examples (and we are not suggesting the above list is exhaustive) of why random SOC scenarios are challenging and often coded in a directed approach in a UVM virtual sequence.

b) *The PSS solution*

PSS allows the definition of behaviors including their dependencies or legality rules. We are going to model each one of the behaviors in a PSS action and use flow objects for the dependencies. For example, Figure 2 shows how the UART component (device) is going to have a read and a write actions (note that for simplification we have provided illustration and not code):

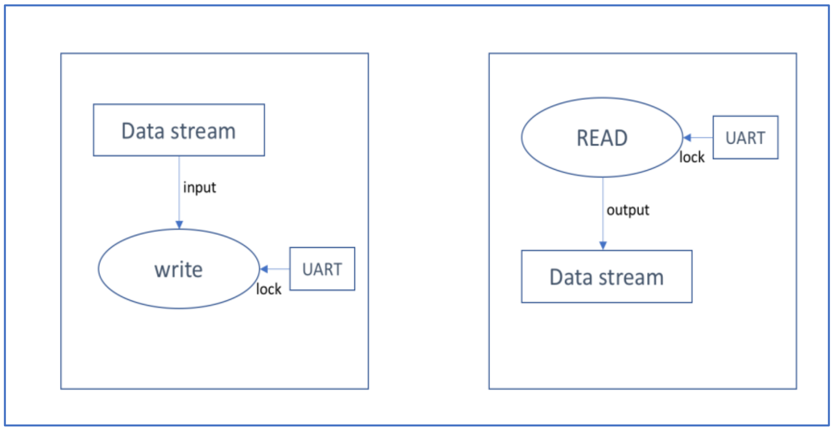


Figure 2: UART read and write actions

Note that the write action reads a data stream (input) and locks the UART device using a PSS resource. The action outputs a data stream and locks the UART as well. Figure 3 shows how the DMA component actions are modelled as follows:

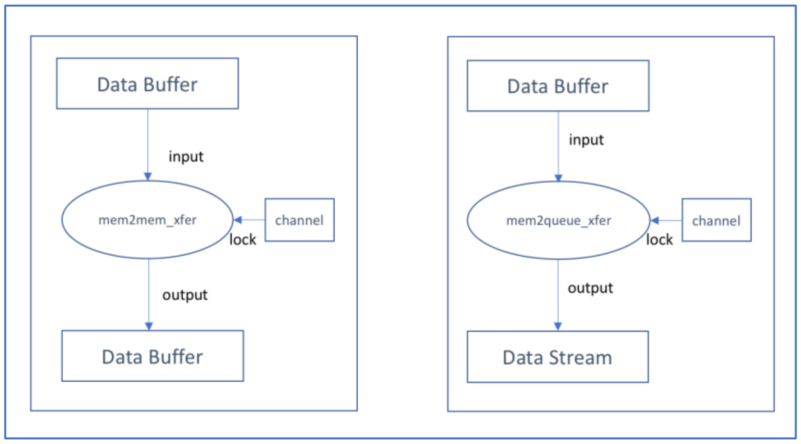
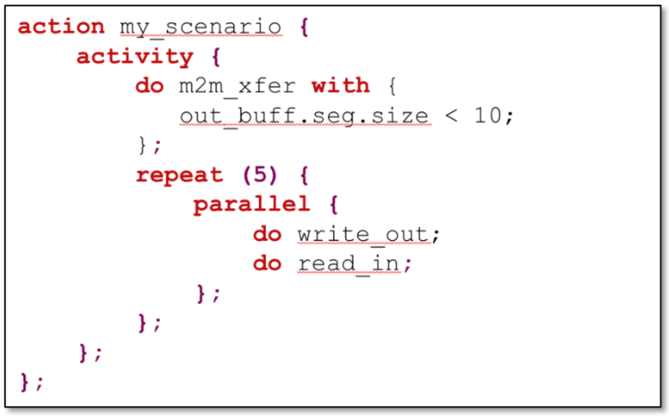


Figure 3: DMA actions

Note that the DMA mem2mem_xfer action is locking one of the DMA channels and not the entire device. Also, the mem2queue action is reading from the memory and streaming the information into the UART queue. Inputs and output have randomized attributes that allows defining legal chains of producers and consumers in the system. Given the above model then Figure 4 shows how the user can create the following abstract scenario:



```
action my_scenario {
    activity {
        do m2m_xfer with {
            out_buff.seg.size < 10;
        };
        repeat (5) {
            parallel {
                do write_out;
                do read_in;
            };
        };
    };
};
```

Figure 4: scenario request example in a compound action

The example above is a partial specification of a scenario in PSS. Scenarios in PSS are formed of compound actions that includes an activity block that describes a desired execution of sub-actions and scenario constraints. Using the dependencies defined in the model above, a PSS compliant tool can complete this scenario request by:

- Inserting appropriate memory initialization.
- Select two UART devices to handle UART write and UART read in parallel.
- Select available DMA channels to feed the write action and copy data from the queue to memory in case of a UART read
- Both DMA constraints and any scenario constraints (for example, in this case we may ask for a scenario with a buffer size smaller than 10) are resolved to provide a legal consistent scenario

Similarly, a user could have captured the DMA low-power or a configuration dependency using a PSS state-machine. Although this is only a simplified example, it is already enough to illustrate how PSS components and actions define the legal space, and how compound actions with activities leverage the rules to come-up with legal complete scenarios to stress any system.

Notice that the PSS model is not connected to a specific implementation and thus can be targeted or ported to any concrete testbench or execution platform using PSS exec blocks. Figure 5 gives an example:

```
extend uart_c::read_in {// aspect oriented
   output data_stream_s  data;

   exec run_start C = “””
       init_uart( {{uart_id}});
   “””
   exec body C = """
       uart_read({{uart_id}} );
   """
}
```

Figure 5: Connecting abstract environment to specific C routines or SV sequences

Figure 5 shows how a user specifies the implementation specific calls. The user specifies that the uart device needs to be initialized at the beginning of the test (using the “init_uart” routine at the “run_start” phase of the test) and that whenever the “read_in” action need to be executed the “uart_read” C runtime needs to be executed. The text within the triple-quotes is templatized code that will be embedded into the test. A “mustache” notation allows embedding randomized attributes (“uart_id” in this example) into the generated test. This example of code with two read_in actions of uart with id 0 can generate the code shown in Figure 6.

```
int main_core3()
{
   init_uart(0); // run_start of read_in
   load_mem(0x1000);
   mem2queue(0x1000,0);
   uart_read(0); // body of action read_in #1
   uart_read(0); // body of action read_in #2
   done(1);
}
```

Figure 6: Generated code sample using exec blocks

Once a PSS solution determines the scheduling of the actions it places the right exec in the right location in the file and replaces all the randomized attributes with their randomized value. The user can decide to call C, SV or any implementation language of choice. This simple exec block API allows users to own and maintain the connectivity layer and not to get locked to a PSS vendor specific implementation.

The standard input format is provided in two styles: a domain specific language (DSL); and codified in a C++ library. These input formats are consistent by design and are equally capable in supporting the standard concepts.

PSS also provides support for coverage and checking. DSL and C++ provide the "coverspec" construct to specify coverage targets related to the test scenario being created. While using the SV style of coverage, PSS allows capturing use-cases that are not practically to capture in SV and allows both gen-time and run-time coverage.

There are no specific language constructs built into PSS to facilitate checking but the abstract model can be used as a reference model and prediction mechanism for complex systems. On top of that a PSS model can interact with external foreign-language code, such as reference models and checkers, to help compute expected results during stimulus generation or in run-time.

Regarding re-use, traditionally users tried to take a piece of code that was created for one platform and apply it to another platform using code translators or data convertors. Beyond marginal success, this approach does not work as arbitrary SV code cannot be migrated to SW driven testbenches that have different level of control and different objectives. PSS suggests an upfront analysis of both the test interfaces and the required scenarios on multiple platforms, and identifies a level of abstraction that is valid for both. Once this analysis is achieved and implemented within the action models, it can be safely applied on multiple platforms. PSS provide a couple of mechanisms to implement the abstract actions on top of users testbenches using import functions or templatized code with the mustache notation. A few guidelines on reuse:

1. Ensure that your actions inputs and outputs are well encapsulated for reuse and can be implemented on all platforms.
2. It is possible to have functions call UVM sequences, but make sure that your reusable scenarios are not locking you into UVM which will make them too slow for Acceleration and not-reusable to other integration levels.
3. Having the PSS constraint solver dependent on the SV solver can make your PSS solution platform dependent.

Some users wish to write abstract firmware that can run on top of transactional master/slave testbench or on top of software driven setup. While it was discussed in Accellera the past, this is not part of PSS. There are multiple home-grown and proprietary solutions to achieve a C layer that can run or both. Such extra reuse can be useful but also introduces a style that is foreign for both the SV and the C teams. We highly recommend exploring this on real-life firmware and getting the blessing of both C and SV teams before committing to such a solution.

To summarize, PSS provides a powerful run-time semantic using state-machines, resources and flow objects to provide both automation and portability.

III. What is the value of adopting a standard (as opposed to a proprietary) tool?

According to [2], standards enable competition, lead to economies of scale, and allow innovation. Consider, for example, light bulbs. You can go to any store and buy any brand of light bulb and be sure it will fit in your lamp because the threads at the base are standardized. This allows many companies to make bulbs in competition with each other. They can make large numbers of bulbs because anyone can use them. They can innovate by coming up with more energy efficient bulbs, colored light, etc., and can rest assured their bulbs will work with any lamp. To the above advantages I would also add the fact that consumers investment decisions also become easier. Following the light bulb example, users of the light bulb can invest in lamps and fittings that support the standard in confidence that there will be a plentiful supply of light bulbs at competitive pricing and with innovative options. In this section we see how enabling this applies to the EDA and semiconductor industries.

In EDA, adopting a standard holds many merits:

- While not always perfect, standards are typically multi-perspective thought out solutions combining the experience of many participating users and vendors.
- Multi-vendor support: since a standard is open, it is a great opportunity for vendors to address a large community of users. This in turn allows users to negotiate better pricing. The multi-vendor competition typically feeds progress, automation and a full eco-system around it.

- Staffing implications: standards also tend to grow industry-wide knowledge and expertise, making it easier to attract talented engineers with minimal ramp-up needs.
- Industry guaranteed solution: history shows (especially in the EDA industry) that proprietary solutions typically disappear in the presence of a standard or eventually evolve into a standard. Many times, we experienced the "island" syndrome where companies attempted to build home-grown solutions or adopt a proprietary solution and found themselves isolated from the industry and future development. For example, consider e and System Verilog. Although e has now been standardized, it was seen for many years as proprietary and System Verilog now has the market share (according to [4] System Verilog accounts for about 75% of ASIC/IC Languages Used for Verification (Testbenches)).

It should be recognised that there can be disadvantages to adopting a standard. For example,

- The implementation of standard removes creativity and innovation. With respect to PSS, the standard has required users and tool vendors to agree compromises and it could be argued that a single user could work with a single vendor to agree a set of innovative features for their own PSS solution.
- Standards often force people to change their methods in order to adopt the standard.
- Some argue that standards reduce productivity by forcing unnecessary syntax or actions for compliance.

However, it should be remembered that adoption of a standard usually accelerates adoption (as described below).

1) *Consider the impact of standardization: UVM*

UVM now dominates the constrained random verification market. Accellera approved version 1.0 of UVM on February 21, 2011. The biennial Wilson Research Group Functional Verification Study tracks the adoption of UVM from about 7% in 2010 to about 85% in 2016 (the figures from the most recent 2016 survey can be found in 4). Before SV UVM standard, in the experience of the author, there were multiple, competing, incompatible methodologies (and languages) such as eRM (which was the first re-use methodology), VMM, RMM, etc. This had a number of impacts in the experience of the author

- For VIP developers it was not clear which methodology to develop against and meant that any particular methodology/language decision would limit the target market and hence the ROI. There were of course techniques to develop VIP that could address multiple methodologies (e.g. suitable wrappers, mixed language simulations) but all increased development costs and reduced customer satisfaction. This made VIP development commercial decisions more complex and impacted pricing accordingly, impacting VIP buying decisions.
- Recruitment decisions were more complex as the pool of engineers had a wider variety diversity of skills and so cross-training had to be considered. The recruitment process was also more complex as differences in background had to be considered. It also often meant that there was a longer ramp-up for engineers as they had to learn new methodologies and languages.
- Hiring contract resources was often harder too. The author personally remembers recruiting a team of 10 contract engineers for a Specman project where a significant training and ramp-up period had to be planned into the project to ensure all the verification contractors had the skills in the languages, methodologies and tools selected for the project.
- Difficulty in making investment decisions (e.g. training decisions, VIP development) because it is not always clear how long a particular solution will last in the market. For example, anybody making VMM VIP development decision pre-UVM would have to concern themselves with the potential lock-in with VMM and consider a potential transition path.

Post UVM standardisation a number of these issues become resolved, for example

- There is a competitive market for tools and VIP
- Investment decisions (e.g. training, VIP) are easier as companies can see the longer-term viability of UVM
- There are many more UVM-ready verification engineers available for both recruitment and contracting making faster ramp up, reduced costs, easier interviewing, etc.

The author would postulate that the standardization of PSS will have a similar impact in time. However, the question now becomes one of when to adopt the technology.

B. *Should I adopt a PSS technology early?*

Obviously, the answer is context and circumstances dependent. Teams that adopt the right solution early typically enjoy the automation early, create verification assets that can be leveraged over longer life-times or migrated later on, learn the PSS principles and concepts sooner and are able to drive the standard and industry solution in their preferred direction. The standard early adopters thus became a major asset to their companies.

An interesting example of this is the ‘e’ language (developed by Verisity who were acquired by Cadence) and Superlog/Vera (developed by Co-Design Automation and System Science respectively who were acquired by Synopsys). Superlog and Vera were based on Verilog. These two languages were donated by Synopsys to Accellera and were developed into SystemVerilog standard which was supported by multiple vendors. The ‘e’ language was standardized later (IEEE P1647) and the only publicly available implementation is Specman from Cadence. According to [4], SV and Specman have 75% and 6% respectively of the market in verification languages. Ignoring the debate on which is the better language which is outside the remit of this paper, I believe this has 2 lessons: the first is that the earlier standardization of SV helped it to gain traction in the market; the second is that the single implementations of “e” gave the perception of proprietary even after standardization. I believe this leads to two lessons for PSS: the standardization will drive adoption; the perception of proprietary features in a tool will have a negative impact on its adoption. Figure 7 shows the migration efforts when the UVM standard was released. The motivation is clear: how can I select a tool that is PSS compatible and that serves my needs best?

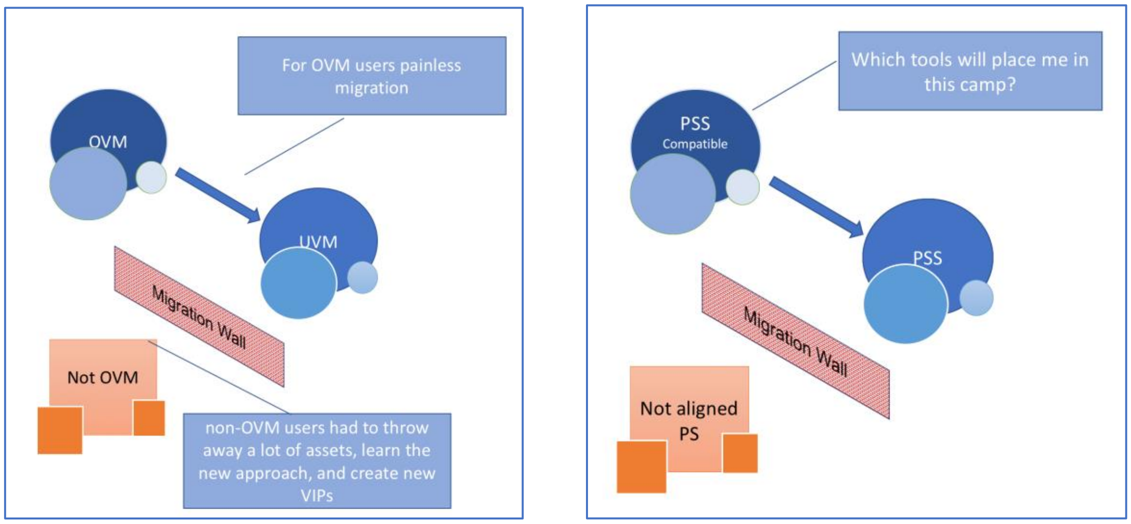

For OVM users painless migration
OVM
UVM
Migration Wall
Not OVM
non-OVM users had to throw away a lot of assets, learn the new approach, and create new VIPs
Which tools will place me in this camp?
PSS
Compatible
PSS
Migration Wall
Not aligned PS

Figure 7: UVM vs. PSS migration example

C. *Selecting an appropriate PSS tool*

There are at least 3 PSS tools on the market at the time of writing. In alphabetical order:

- **Breker**: Breker Verification Systems has been working in this area for many years and their website lists a number of application case studies. At DAC 2018 they announced that the latest version of their Trek product suite and apps provides complete support for both the DSL and C++ versions of the standard. Breker’s product synthesizes test cases that may be deployed into block level UVM environments, complete multicore SoC platforms, and for Silicon bring-up. The product includes libraries such as the HSI that provide system services, memory management, and UVM virtual sequencers. The GUI includes debuggers that enable visualization of the multi-threaded tests themselves, as well as graphical editing. The tool includes abstract path coverage and constraints, profiling and coverage closure technology. They also provide automated apps for various applications including Cache Coherency.
- **Cadence**: On 2011, Cadence introduced a PSS tool called Perspec to inject automation and portability to sub-system and full system testbench environments. Perspec suggested innovative concepts such as actions and components, flow objects with input/output and constraints, compound actions with control flow and constraints, exec blocks, resource deceleration and resource aware solving, object oriented and aspect-oriented programming and many more constructs that became the core PSS concepts. Perspec also suggested UML based GUI for simplified scenario creation and abstract debug using the same diagrams. Quoting from deepchip, their solution was #1 "Best of 2017" with Portable Stimulus (see deepchip survey [3] for voting criteria) and, according to the success stories on their website, is already proven by hundreds of users on production projects. At DAC 2018 Cadence also announced support of PSS C++ and DSL input formats and even allowed users on the floor to model and create scenario using the official PSS LRM format. Their solution comes with a rich set of ready-made content of coherency, DVM, low-power, HSI support, memory management and more for both pre- and post-silicon environments.

- **Mentor**, a Siemens Business: The inFact portable stimulus tool from Mentor, a Siemens Business, creates tests targeting block, subsystem, and SoC verification running on simulation, emulation, and FPGA prototypes. inFact accelerates coverage closure, reducing the time to functional and coverage at block level, and ensuring more meaningful tests run at subsystem and SoC level. The product provides an integrated development environment for developing portable stimulus models, and the ability to reuse existing SystemVerilog classes and constraints by importing them into PSS. Integrated analysis and debug capabilities help users develop, analyze, and debug constraints and coverage. inFact is a part of Mentor's functional verification platform.

D. Adopting "future" specification enhancements?

The PSWG worked for a few years collecting user requirements and following well-organized and prioritised process to deliver PSS 1.0. It should be noted that while PSS 1.0 has been released, there are a number of requirements identified, which were not included in the first version of the standard, for reasons of timing. Like all standards, the committee must make a call as to when to release the first version that have the critical concepts to get useful feedback, versus trying to provide every possible feature up front. Vendors may claim to support these unfulfilled requirements. Obviously, there are risks that these proprietary solutions will not be adopted by Accellera or adapted before adoption. An example of this is the Hardware Software Interface (HSI). The PSWG discussed another level of TB control in which the user specified in abstract register layer and use a PSS tool to generate the firmware routines in any implementation language. Referring back to the example in Figure 5, this means that instead of using exec body to call the uart_read function or sequence a user will specify the full implementation of the firmware in PSS primitives. This future standard layer will be called HSI (HW/SW interface). The up-side of this layer is the single source generation of firmware. The down-side is that you may already have the C routines and UVM versions, and that SV users need sequences with constraints and HW access, while the C guys may prefer regular C and have no backdoor access and or HW timing control. HSI and Scenario Constraints are not part of the PSS 1.0 release.
A few suggestions regarding vendor specific future enhancements:

1. Conforming to the standard 1.0 LRM is the best advice to avoid future refactoring and even re-write.
2. While PSS will progress, users should not expect radical changes. One of the key arguments for including a feature in version1.0 was the fact that it accepted by multiple users as useful.
3. Talking to more than a single vendor or an independent PSS expert will allow an evaluation to get a more rounded view if a proprietary enhancement is likely to be adopted or if there is a standard way to accomplish the same.
4. While not always possible, joining Accellera will allow you to know in advanced what is coming and even help steer the standard direction.

E. Tool selection process

We recommend a three-stage selection approach

1. Review the PSS LRM, examples, Accellera tutorials, and recommended usage
2. Define the evaluation criteria, taking into account the PSS LRM
3. Then perform an initial investigation of all the tools against the evaluation criteria to identify a short list. We recommend that the short list contain at list two tools.
4. Perform a detailed evaluation of the tools on the short list against the evaluation criteria

These 3 stages of selection are discussed later in the section but we first turn our attention to preparing for selection.

F. Preparing for selection

In order to perform an informed selection then it is necessary to first have some knowledge and understanding of PSS. We covered some of the PSS concepts at the beginning of this paper, but you will have to conduct your own research.

1) How can I identify the PSS concepts?

We recommend the following for getting up to speed on PSS

- The first step is to read the official PSS LRM and tutorial or just the first few pages (which are quite informative) if you are time limited. There are multiple tutorials that explains how PSS is used in practice to support various industry challenges.
- It is important to identify examples that combined cover the full extent of the PSS specification. components, actions with inputs and outputs, compound actions with control flow, it might be a great proprietary solution but it is not PSS.
- Also, check the various available case studies which show how the tool has been used in practice to date. They are the best mechanism to understand the PSS scope and intended usage. As was mentioned above, the DMA example is part of set of examples that will be provided by Accellera and so it might be useful review this set of examples.
- It might be useful to have access to at least one of the tools at this point so that the various examples and available case studies can actually be run.
- You might also consider the use of external consultants who have PSS knowledge and PSS (or just technology) adoption experience. This can have the advantage of accelerating ramp-up and adoption and can usually help to establish best-practice more quickly. However, there is a risk that those external consultants might not fully appreciate the objectives, company culture, etc. and a further risk that you could become dependent on such consultants (although this can usually be avoided through suitable planning).

It should be noted that while adherence to the standard with the core model is useful, the addition of configuration and library capabilities is necessary to make the tests work. Much like the use of TCL in various tools, the configuration capabilities will be applied to PSS models for particular verification runs. End users should consider the value of the basic models generating basic tests, which will require adaption for their environments, and facilities created by the tool vendors to make this adaption seamless.

2) Selecting the right PSS input format

PSS allows different input formats along 2 main axes:

- Text vs GUI scenario specification: One of the key motivations of PSS is becoming a bridge technology between teams that uses different terminology and testbenches. Users may not understand specific language, methodology and or verification IP, but we all can understand flow-charts. PSS borrowed most of its concepts from model-based approaches and the unified modelling language (UML). This provides a breakthrough in both visualizing and expressing the desired scenario. While many may think right now that a GUI is for test-creations for beginners (which is true), PSS allows questions the boundaries of the actions and legality rules. Model creators can check their PSS models and advanced test-writers can debug the reason for a resource conflict. The fact that PSS randomizes the activity order and timing is an opportunity for a PSS tool to visualize these. Some advices about GUI:
    - Check that a technology can create activities with loops, parallelism, data-flow and more
    - Check that flow-objects, resources, state-variables, etc' are represented in the diagrams, if possible before execution.
    - Check that users can debug numeric or resource contradictions, again ideally statically, before running the simulation
- DSL vs C++: PSS supports two equally powerful standard input formats: a Domain Specific Language (DSL) with a custom parser and a C++ library using a C++ parser.

In general, PSS DSL is viewed as shorter and more readable with better error messages whereas PSS C++ has the advantage of a C++ look and feel. It is tempting to go for C++ based purely on familiarity. However, there are both good and bad reasons (in the view of the author) for such a decision.

- Firstly, the wrong reasons
    - “I will never adopt a new language for religious reasons”: There is still a learning curve with respect to the PSS/C++ class library and design pattern
    - “I have legacy C++ code that runs on my target platform”: Good. But this has nothing to do with PSS/C++, which executes before gen-time on the host. The C++ execution will simply build the data structures that the tool will use at gen-time to create the test(s). Executing the C++ is equivalent to compiling the DSL.
    - “I have gen-time C++ libraries, such as reference models, to bind to”: PSS/DSL can bind to gen-time libraries pretty much like PSS/C++
- And, the right reasons

- “I do a lot of computation as part of my solving/code generation”: This should be easier and more seamless in PSS/C++. This is the same as binding to gen-time libraries. The mechanism is pretty much the same.
- “I construct parts of my model / scenarios programmatically”: Only possible on top of PSS/C++
- “I expand parts of my model from external sources (spreadsheets, XMLs)”: Would require PSS/DSL code-generation or could be handled by imported procedures in DSL, but can be integrated in PSS/C++

G. Defining the evaluation criteria

1) What are my needs?

The scope for PSS is potentially very wide, touching multiple teams with different focus and platforms (e.g. architects, designers, verification and validation engineers). It is key to ensure that the company needs as a whole is being discussed and addressed.

Also, different organizations will have different target applications for PSS, which will drive the evaluation criteria. Identifying these target applications early will help to prioritize criteria based on short, medium, and long-term applications. For example: Is PSS to be used to bring automated tests to the SoC verification and validation process? Will PSS be used to reuse some test scenarios across IP, subsystem, and SoC? Will it be applied to create more-comprehensive scenarios at subsystem level?

Our recommendation is to organize a company-wide committee that can review the needs of the different teams and balance short and long-term requirements. This should result in a list of agreed, prioritised, company requirements. For example

- how to achieve re-configuration? Low-power verification? Interrupt verification?
- Can I capture requirements in PSS? And trace them through the design flow?

Again, you might also consider the use of external consultants who have PSS knowledge who can independently collect and analyse the company-wide PSS usage requirements.

2) What level of alignments exist?

We need to consider the level of completeness we require to the PSS standard. The areas of completeness to be considered might cover

- Standard Input format: LRM compliant means running the DSL or C++ standard LRM code. When evaluating a specific tool, try to run the Accellera usage examples.as a minimum and check for both the parsing and the correct semantics. For example, considering SV UVM, we have seen quite a few times that different simulation run the same SV code in a different way.
- Concepts: The next level of alignment is the conceptual alignment. Does it use the core concepts of activities, actions, input and output with random attributes, constraints, Usage of these concepts is critical as it will maximize the intended value, key for multi-vendor or migration later on, essential in connecting other commercial or home-grown PSS models and also allow you to grow your expertise using the standard PSS paradigm.
- Alignment with the standard should also be considered. For example, does the tool correctly implement the intended semantics of the standard? It would be useful to independently assess PSS compliance by running obtaining an independent parser for example[1] or by comparing two tools it is possible to compare how they handle the basic Accellera examples.

3) What extensions to consider?

Some technologies might offer extensions to the PSS. We can classify the extensions into 2 broad categories:

1. Utilities on top of the standard (e.g. a utility that reads SV TB and generates legal PSS code, or PSS VIP)
2. Adding features that are not part of the LRM standard.

We already discussed the proprietary features a few words on utilities:

- They can enhance the adoption of PSS and improve efficiency but you make sure that standard PSS is generated and part of the flow.
- However, they can also lock the user into a particular technology and run against adoption of a standard.
- It is important to ensure that the extensions do not alter the alignment considered above (e.g. the application that reads the SV code generate standard PSS code). This will allow you to use the standard code later on.

[1] While it may not be 100% complete at the time of writing, the Amiq DVT tool has an independent PSS parser. The authors do not make any recommendations in respect of this tool and others may exist.

We therefore recommend that if extensions are to be considered then a future migration path away from such extensions must also be considered.

4) Vendor selection criteria

It is also important to evaluate the tool vendor as well as the tool

- PSS deployment capabilities for specific environments such as UVM, SoC, etc.
- Coverage modelling and closure; How do you collect coverage and ensure convergence? Can you perform ranking?
- Debug: Most estimates suggest engineers spend about one third of their time in debug (39% according to [4] System) and so the debug capabilities should be a key consideration. This should encompass PSS model debug, test generation debug, test failure analysis (gen and run time) and debug?
- Availability of extensions: we have already discussed the availability of libraries and the issue around vendor lock-in, but there might also be other extensions such as pre-prepared models for popular designs (in the same way that there is UVM VIP for USB, PCIe, etc). The evaluation should consider the target applications identified above and identify what infrastructure will be required to support those target applications. What elements are readily-available in the organization or the industry, and what elements does your PSS tool need to provide?
- License model: for example, consider the ownership of the tests generated by the tool. Can the tests generated be run free of any PSS licensing? Can they be shared with others who do not have the PSS tools and technologies?
- Environment support: What verification environments and verification engines are supported by the vendor tool? How do these align with the environments and engines used by the target applications identified above? For example, is there support for verification tools such as simulators and emulators from different companies? It is important to understand a PSS tool selection impacts other tooling decisions: such as limiting choices on simulators or emulators to run the PSS-generated tests. For example, many of the testbench tools from the early 2000s (e.g. Vera, Specman) supported all simulators but had enhanced support in certain environments. The critical thing here is to identify and understand the trade-offs in functionality, performance, and automation.
- PSS offers more automation to the user than ever before. While beneficial for users, this also puts a burden on vendors to deliver the needed technology such as a constraint solver, coverage engine and other engines to serve your needs moving forward.
- Reuse of existing verification infrastructure: Although PSS is aimed at improving reuse through abstraction, an evaluation should also consider existing verification infrastructure. For example, does the vendor tool enable reuse of existing infrastructure that are relevant based on your target applications? For example, in an IP to SoC reuse application, does the vendor enable automated reuse of existing SystemVerilog code? The tool should allow calling system Verilog tasks and sequences but not rely on the SystemVerilog to be there (which will block reuse in cluster or system).
- Tool ecosystem: Consider how the PSS tool relates to the vendor's tool ecosystem and your tool ecosystem. For example, how does coverage captured by the PSS tool align with the coverage database used in your organization?
- Automation: What automation does the vendor tool enable within the standard syntax? For example, analysis tools, accelerated coverage closure, boilerplate code generation, etc.
- Training resources, online/in-person
- Field support: Availability, knowledge and experience of the vendor's local support team. It might also be useful to consider wider support models through consultancies. Does your vendor have a local team with the right experience to support your project? For example, when the author was an early-stage adopter of e and Specman at a previous employer, Verisity had already enabled an ecosystem that provided consultants, training and VIP.

H. What tests or questions can be asked while evaluating the short-listed technologies?

As different teams may have different needs, challenges and jargon, we recommend that teams meet separately with different teams for both identifying the challenges and also for demonstrations of specific technology.

There are quite a few tests that can help identify a PSS tool:

1. The practical deployment test: Can the tool be easily applied into the various environments in use? Is there a lot of extra code to be written by the user to make this happen? For example, how does it work with

existing UVM environments. Can it generate software-based tests that make use of system services and memory management for SoC platforms?

2. The migration test – there are two migration routes to consider

   a) Migration of existing infrastructure: As mentioned above, we are likely to want to re-use existing verification infrastructure (SV UVM VIP, directed SoC C-based tests, ICE environments, validation benches) and so we have to consider how they can be incorporated into any agreed PSS strategy.

   b) Migration of PSS infrastructure: how easy it is to migrate the vendors legacy format to the PSS input format? Can it co-exist with the standard input format models? For example, what would it take to integrate the code with the Accellera provided user examples?

3. The automation test – Check that the automation provided in the tool is on top of the standard and not a bypass on the standard. For example, if a technology automatically connects the PSS model to your test bench rather than using the standard exec blocks for connectivity as such automation might lock you to a specific vendor.

4. The "real design" test. Can the tool be used on a realistic SoC platform with multiple interlocking processors, or a UVM testbench with complex multi-threaded sequences, for example? Can the tool generate tests powerful enough to verify these designs? This might require ramping up on PSS as discussed in section F.1) "How can I identify the PSS concepts?".

IV. Conclusion

The recent release by Accellera of the standardization of The Portable Test and Stimulus Standard (PSS) is likely to accelerate the adoption of PSS. Potential adopters of the standard currently have three vendors to select from: Breker, Cadence and Mentor Graphics. However, adopters will want to ensure that whilst they gain the benefits of early PSS adoption, they are aware of the potential pitfalls of getting trapped with a particular vendor by, for example, reliance on proprietary features. This paper has identified both a process and criteria for valuation for selecting the right technology.

V. Acknowledgments

The author would like to acknowledge the contributions made by Cadence, Breker and Mentor Graphics.

VI. References

1. PSS Accellera Standard, http://www.accellera.org/downloads/standards/portable-stimulus
2. The importance of standards, http://bpastudio.csudh.edu/fac/lpress/471/hout/netech/standards.htm
3. http://www.deepchip.com/items/dac17-01a.html
4. The 2016 Wilson Research Group Functional Verification Study, https://blogs.mentor.com/verificationhorizons/blog/2016/10/31/part-10-the-2016-wilson-research-group-functional-verification-study/