2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Portable Stimulus Standard: The Promises and Pitfalls of Early Adoption

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

accellera SYSTEMS INITIATIVE

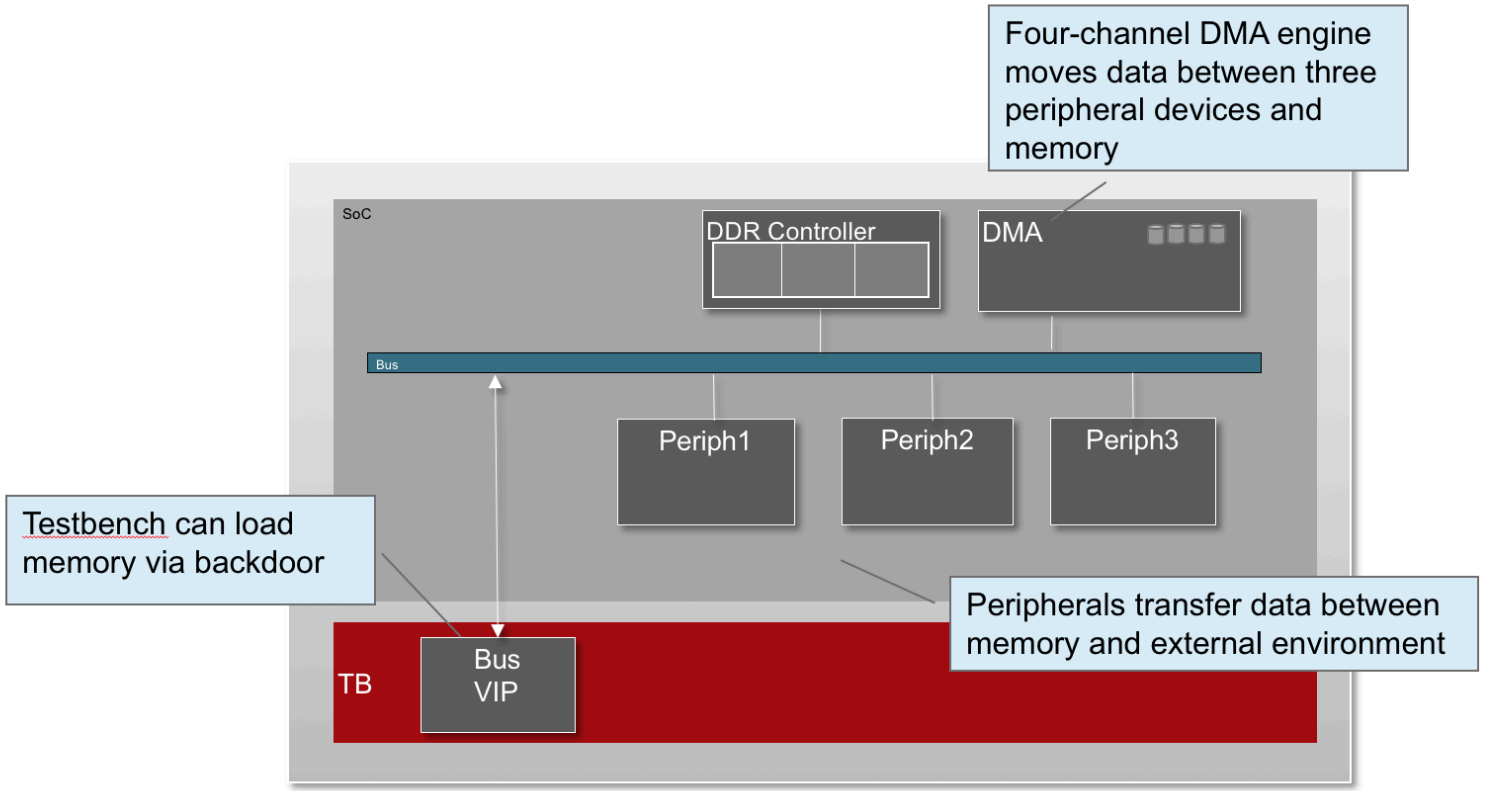
Agenda

- Introduction
- What is the value of adopting a standard (as opposed to a proprietary) tool?
- Should I adopt a PSS technology early?
- Selecting an appropriate PSS tool
- Adopting “future” specification enhancements?
- Tool selection process
- Conclusion
- Acknowledgments

Introduction

- Portable Test and Stimulus **Standard**
- What is it?
  - 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
- Tool Solutions
- Adoption strategies!

PSS: An Example



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

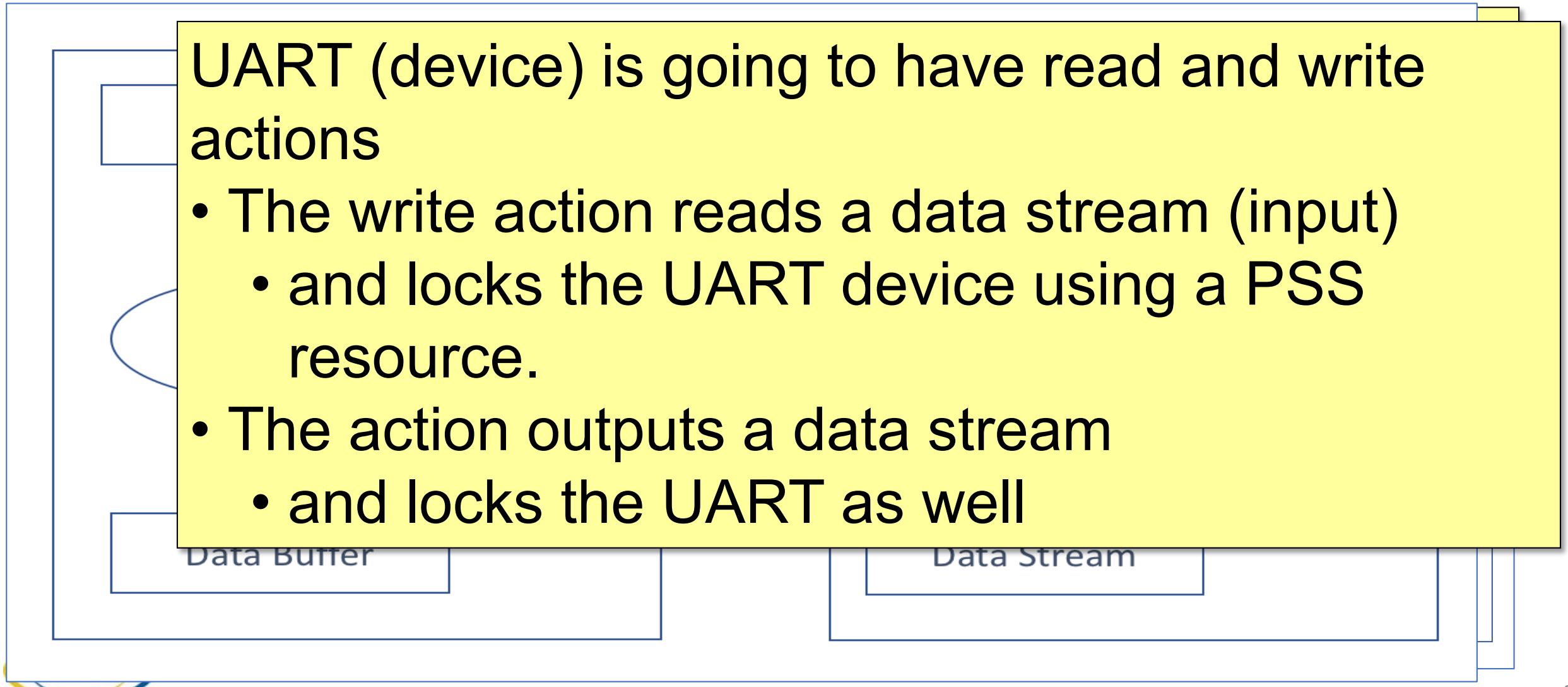
Note: there are 2 input formats
- a domain specific language (DSL);
- C++

Our example uses DSL

PSS Example ctd: Objective & Challenges

- Objective:
  - To create scenarios to stress this system in multiple ways
- Challenges (not exhaustive)
  - A DMA channel must be available for the task
  - Coordinate writes and reads with the loading of the specific UART queue
  - The DMA is not copying uninitialized memory (to enable checking)
- Solution today
  - Is often a directed approach

PSS Example ctd: Actions & Dependencies



UART (device) is going to have read and write actions

- 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

PSS Example ctd: An Example Scenario

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

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

NOTE: the PSS model is not connected to a specific implementation

PSS Example ctd: Concrete Implementation

```
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}} );
    """
}
```

Connecting abstract environment to specific C routines or SV sequences

- The user specifies the uart device needs to be initialized
- The “read_in” action is implemented by the “uart_read” C runtime
- 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.

PSS Example ctd: Generated Code

```
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);
}
```

- 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

Getting up to speed on PSS

- Read the official PSS LRM and tutorial(s)
- Check the various available case studies
- Identify examples that (combined) cover the full extent of PSS
- You might also consider the use of external consultants who have PSS knowledge and PSS (or just technology) adoption experience

Notes on coverage and checking

- Coverage
  - PSS provides support for coverage “coverspec”
  - Generate both gen-time and run-time coverage
- Checking
  - express properties that are checked throughout execution
  - There are no specific language constructs built into PSS to facilitate checking
  - external foreign-language code (reference models, checkers) to compute expected results during stimulus generation or in run-time

The Power of a Standard

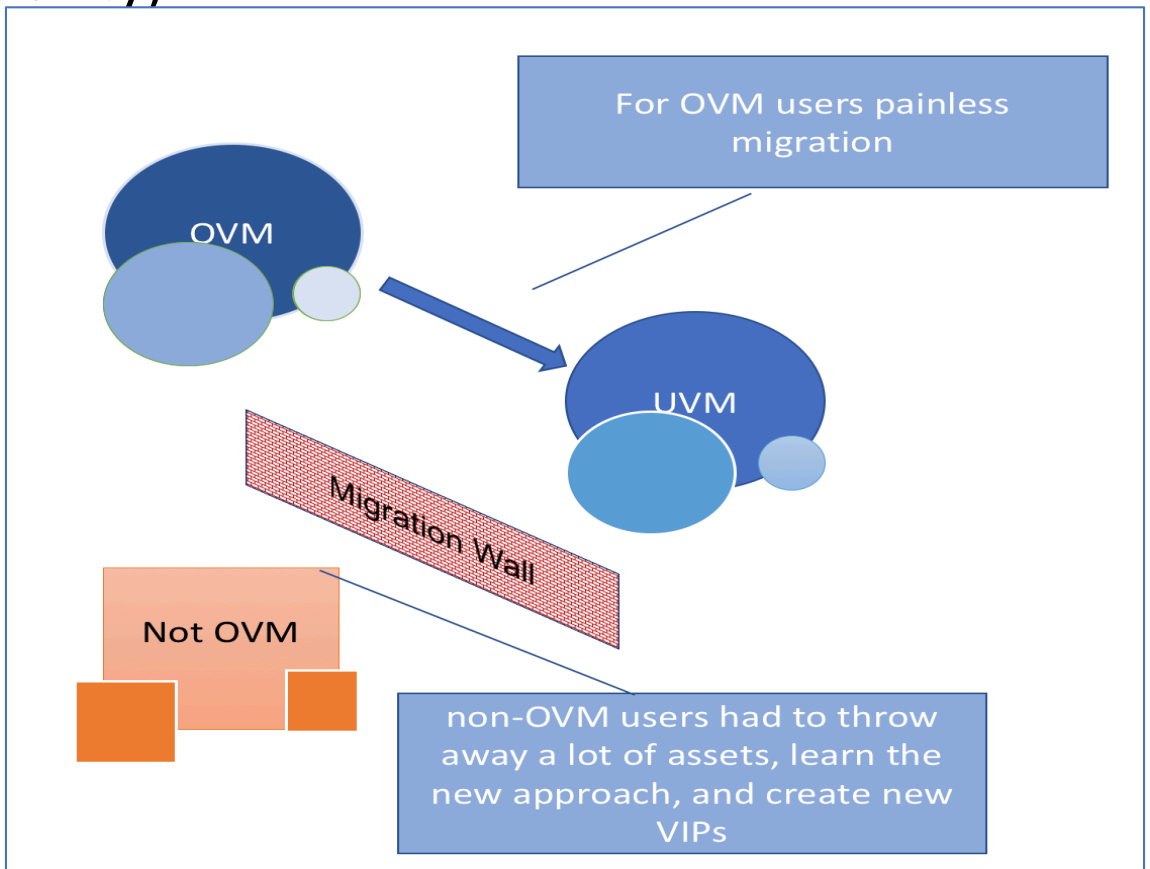
- Advantages
  - enable competition
  - lead to economies of scale
  - allow innovation (?)
  - User investment decisions

In EDA

Advantages	Disadvantages
multi-perspective	removes creativity and innovation • users and tool vendors compromise
Multi-vendor support	
Staffing implications	force people to change their methods
Industry solution	Compliance forces unnecessary syntax & actions impacts productivity

The Impact of a Standard: UVM

- Accellera approved version 1.0 of UVM on February 21, 2011
  - UVM adoption from about 7% in 2010 to about 85% in 2016
  - VIP & tools (development and availability)
  - Recruitment decisions
  - Hiring contract resources
  - Investment decisions
- Will PSS standardization have a similar impact?



Should I adopt a PSS technology early?

- Lessons from UVM
  - PSS standardization will drive adoption
  - Perception of proprietary features will impact adoption
- Benefits?
  - Learn the PSS principles and concepts sooner
  - Get the benefits earlier
  - Create verification assets that can be leveraged over longer life-times
  - Drive the standard and industry solution in their preferred direction
  - Early adopters thus became a major asset to their companies

Selecting an appropriate PSS tool

- 3 PSS tools on the market at the time of writing. In alphabetical order:
  - Breker; Cadence; Mentor, a Siemens Business
- Adopting “future” specification enhancements?
  - Future refactoring or re-write?
  - Vendors may claim to support “future” requirements in PSS v1.0
  - While PSS will progress, users should not expect radical changes.
  - Talk to multiple vendors &/or independent PSS experts for a more rounded view
  - Joining Accellera for advanced information (and steer the standard direction?)

Tool selection process

- We recommend a 4-stage selection approach
  1. Review the PSS LRM, examples, Accellera tutorials, and recommended usage
  2. Define the evaluation criteria, taking into account
     1. The PSS LRM
     2. Your expected use models
  3. Short list (via a quick eval) against the main evaluation criteria (2 tools?)
  4. Perform a detailed evaluation of the short list against the evaluation criteria

Defining the evaluation criteria

- What are my needs?
  - Can this need be supported and implemented by existing PSS technology?
- What extensions to consider?
- Vendor selection criteria
  - PSS deployment capabilities for specific environments such as UVM, SoC, etc.
  - Coverage modelling and closure
  - Debug
  - Availability of extensions
  - License model
  - Environment support
  - Reuse of existing verification infrastructure
  - Tool ecosystem
  - Automation
  - Training resources
  - Field support
  - Top down or bottom up methodology support

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

- The practical deployment test:
  - Can the tool be easily applied into the various environments in use?
- The migration test – there are two migration routes to consider
  - Migration of existing infrastructure
  - Migration of PSS infrastructure
- The automation test
  - Check that the automation provided in the tool is on top of the standard and not a bypass on the standard
- The “real design” test

Selecting the right PSS input format

- Text vs GUI scenario specification
- DSL vs C++
  - DSL should be more succinct than C++
  - There is still a learning curve for PSS/C++
  - PSS/DSL can bind to gen-time libraries pretty much like PSS/C++
  - DSL error messages are readable vs. gcc C++ error messages
  - External sources (spreadsheets, XMLs) should be easier to integrate in PSS/C++
  - Computation as part of solving/code generation should be easier in PSS/C++
  - In general, DSL is easier for C or SV users while PSS/C++ if better for C++ users

Conclusion

- Accellera standardization of PSS is likely to accelerate adoption
- Currently three vendors to select from:
- Early adopters can gain benefits
- But should be aware of the potential pitfalls
- This paper has identified both
  - a process
  - and criteria for valuation for selecting the right technology.

- Thanks to contributions from Cadence, Breker and Mentor Graphics