PORTABLE TEST AND STIMULUS STANDARD PROPERTIES AND USE

Author: Kalle Haapalainen

Supervisor: Jukka Lahti

Haapalainen K. (2019) Portable Test and Stimulus Standard: ominaisuudet ja käyttö. Oulun yliopisto, Elektroniikan ja tietoliikennetekniikan tutkinto-ohjelma. Kandidaatintyö 18 s.

TIIVISTELMÄ

Verifiointi on tärkeä osa järjestelmäpiirien suunnitteluprosessissa. Järjestelmäpiireistä tulee jatkuvasti suurempia ja monimutkaisempia kokonaisuuksia, mikä myös tekee niiden verifioimisesta vaikeampaa ja aikaa vievän prosessin. Uusia käytänteitä ja metodeja kehitetään jatkuvasti, jotta piirien verifiointi prosessia saataisiin tehokkaammaksi ja nopeammaksi. Portable test and stimulus standardin avulla testien generointia voidaan automatisoida ja testiympäristöjä voidaan uudelleen käyttää eri alustoilla.

Avainsanat: Verifiointi, Järjestelmäpiirit, Portable test and stimulus standardi, PSS, Abstraktio, Constrained random

Haapalainen K. (2019) Portable test and stimulus standard properties and use. University of Oulu, Degree Program in Electronics and Communications Engineering, Bachelor's thesis, 18 p.

ABSTRACT

Verification is a key part in the System-on-chip (SoC) design process. SoCs are constantly getting more and more complex, which makes verifying them harder and more time consuming. New standards and methods are constantly being developed to try and make the verification process easier and more effective. The portable test and stimulus standard allows automated generation of tests and reusing test environments across multiple different platforms.

Key words: Verification, System-on-a-chip, Portable test and stimulus standard, PSS, Abstraction, Constrained random

SISÄLLYS

Tiivistelmä 2
Abstract 3
Sisällys 4
Abbreviations 5
1. Introduction 6
2. Soc circuit design and verification flow 7
2.1 Design flow 7
2.2 Verification flow 8
2.2.1 Partitioning 8
2.3 Challenges in verification 9
2.4 Randomized stimulus 10
3. Portable test and stimulus standard 11
3.1 What is PSS? 11
3.2 Example scenario 11
3.3 Modelling concepts in PSS 12
3.4 How will it work 14
3.5 Tool flow 15
4. Discussion 16
5. Summary 17
6. References 18

ABBREVIATIONS

SoC	**System-on-a-chip**
PSS	**Portable test and stimulus standard**
UVM	**Universal verification methodology**
TLM	**Transaction level model**
RTL	**Register-transfer level**
HW	**Hardware**
SW	**Software**
FPGA	**Field-programmable gate array**
SV	**SystemVerilog**
VHDL	**VHSIC Hardware Description Language**
DUV	**Design Under Verification**
DSL	**Domain-specific language**
IP	**Intellectual Property**
EDA	**Electronic design automation**

1. INTRODUCTION

This thesis is about the verification methodologies used to verify system-on-a-chip (SoC) designs. The main focus is on the Accellera Portable test and stimulus standard (PSS) and on the properties and benefits of it compared to other verification methodologies. I will also introduce different stages for SoC design and verification flow and then show where and how PSS is applicable.

SoCs nowadays can contain more than 10 million gates with extremely large and complex systems, which makes it very hard and time-consuming process to verify tehse designs. Proper and thorough verification is required, because if there is a bug in the design and it is identified at a lower level of abstraction, the process to fix it may require a complete redesign of the system. Verifying a design's functionality and looking for bugs may take up to 80 percent or more of the overall time spent with it. This process is indispensable however, especially at earlier stages, since fixing the bugs at a later stage can be very expensive. [1] The optimization and development for verification methods and tools is an ongoing process. More complex designs require more sophisticated verification tools and methods. One of the recent released tools is Accellera's Portable test and stimulus standard. [2]

The goal of this thesis is to study how the Portable test and stimulus standard works and what are the benefits of it. The focus is comparing PSS to current methodologies and looking at the reusability, portability and stimulus generation across different platforms. Currently there is no single way to specify intent and behaviour that is reusable across target platforms (e.g emulation, simulation, post-silicon etc.). With PSS you can specify a behaviour once and multiple implementations can be derived. Generally, in functional verification, depending on whether a block, subsystem, SoC or a system is being verified, multiple different languages and techniques are used for stimulus generation. [5] Using different languages and techniques for block- and subsystem-level verification can result in challenges, such as making it difficult to utilize and reuse the test-scenarios at SoC and system level.

2. SOC CIRCUIT DESIGN AND VERIFICATION FLOW

2.1 Design flow

Soc design begins with defining the functional specifications for the product. After this the product and system engineers specify the architecture that is most appropriate for the required functionality. [4] Some of the functionalities can be partitioned for either hardware or software, which means sharing responsibility in the functionality of the design. After this, the design can be separated between the hardware and software design paths and the development can continue independently. [3]

After partitioning the hardware and software design are happening parallel to each other, which allows starting the designing of software without any ready physical hardware. Both hardware and software are being verified constantly during the design process to avoid and fix possible bugs as early as possible. After a certain point has been reached for the designs, they can be tested together with methods such as co-simulation. Next in hardware development there will be a netlist synthesized from the RTL and software is compiled to low-level computer language. These processes will be simulated together, which will help the design process of getting RTL models out faster. Once the processes are done the RTL models and low-level software are verified using HW/SW co-emulation process. [3]

At a later stage the physical design and application software development will be put to a hold and no big changes will be coming out, but only corrections and bug seeking. After this there will be a sample silicon manufactured for testing purposes. This silicon sample will work as a candidate for functional product and final verification takes place. If the results are satisfactory the prototype is sufficient for finalization and the final phase called tape-out can begin. Tape-out means sending the design plans to manufacturer for production. [3].

2.2 Verification flow

Verification is extremely important since a design that cannot be verified has no value. [2] Verification accounts for 80 percent or more of the whole design process and as stated before it is indispensable process. If bugs and mistakes in design go through to as far as the tape-out process unnoticed and unfixed, it can and will be very expensive to correct. This applies especially for hardware part since it cannot be patched like software can. Verification should start as early as possible, even at the same time as the design is being made. Finding bugs and fixing them early is important, because the earlier these bugs are fixed the less the cost to fix. If some mistakes are found later, it can require redesigning some of the parts in the SoC. [3]

2.2.1 Partitioning

Software and hardware are being verified separately in the beginning, but later are brought together when the designs are more ready. In software verification, in addiction to bugs in software, some bugs in hardware can also be revealed. Some repartitioning between hardware and software can also happen at this stage. If something is found difficult to implement on software, it can be implemented on hardware instead. [3]

Software verification can be categorized into static analysis and dynamic analysis. Static analysis means analysing the code without executing it and dynamic analysis means executing the code and analysing it. Static analysis is typically done when no hardware model is available. To make sure that software will work properly with hardware, dynamic analysis is required. This can be challenging if no physical implementation of hardware is ready. Some models can be ready at an early stage, but they might not work like the actual hardware. Emulation and FPGA prototyping can be used to test software in a later phase. [3]

Verification for hardware is different from software and it must be done more carefully, since hardware cannot be patched after release. A thorough verification plan is needed to make sure the final product works as intended. If the final design does not meet the specification, a lot of time and money is wasted. Hardware

verification can be divided into system-level verification, functional verification and physical verification.

In system-level verification a test environment is made, which is typically a behavioural testbench in a language like VHDL, SystemVerilog (SV), C or C++. Functional models that perform tasks in simulation are also designed according to the specification. These models are then simulated, and their output is compared to the specifications. [3]

Functional Verification is used for RTL design verification. RTL is simulated to ensure that the architecture of the design works on a register level. After RTL is synthesized and a netlist is generated, they have to be verified. Verifying in this case means checking that the netlist and RTL are logically equivalent. [3]

Before tape-out there is physical verification process, which means matching the chip's layout with schematic and checking for design rule violations, analysing for antenna effect, crosstalk and other physical effects. Any violations in physical layout should be fixed in this phase. [3]

2.3 Challenges in verification

Usually in functional verification multiple different languages are used for stimulus generation, depending on whether a block, subsystem or system is being verified. In RTL block and subsystem verification SystemVerilog is used frequently, but also "e", SystemC and VHDL are used. The use of multiple different languages can result in challenges, such as making it difficult to utilize block- or subsystem verification at SoC or system levels. Embedded software is used often to exercise the design at SoC and system level, but this software doesn't provide support for automated stimulus generation like languages such as SystemVerilog do (e.g. constrained random stimulus generation) in block- and subsystem level environments. [7]

One of the challenges in verification is when moving from block to SoC testing and from simulation to emulation, is having to rewrite many of the tests. The reuse of verification environments and tests is still limited to specific code based on functional protocols, making verification less productive. [6]

2.4 Randomized stimulus

Randomized stimulus is a very effective way to reach unpredictable corner cases fast. It can sound inefficient and chaotic, but with constraints and boundaries applied it is a very powerful tool for good coverage gain. Constraintd are used so the design under verification (DUV) gets fed with legal values. A reference model is normally created using TLMs (transaction level models), assertions or testbench. The results from feeding the DUV with randomized stimulus is compared to the reference model to see if the outputs are correct. [4]

The testbench has to be modelled for all different behaviours and error conditions that are randomly generated and also manage the test execution towards scenarios that are not yet executed. Randomization needs and environment that can support it and due to its complexity, the implementation can be laborious compared to directed approach. [4]

3. PORTABLE TEST AND STIMULUS STANDARD

3.1 What is PSS?

The portable test and stimulus standard is an Accellera standard created by the Portable stimulus working group. PSS defines a specification for creating a single representation of stimulus and test scenarios. This representation can be run on multiple platforms across different levels of integration with different configurations, which in turn enables generation of different implementations of a scenario. These implementations can be then run on different execution platforms such as simulation, emulation, FPGA prototyping and post-silicon. [5]

The portable test and stimulus standard describes a domain-specific language (DSL) that is used for modelling scenario spaces of system, generating test cases and analysing test runs. It also defines C++ library that is semantically equivalent to the DSL. Both of the DSL and C++ formats are designed so that declarations in either format can be referenced in the other. The portable stimulus specification captured in either DSL or C++ is referred to as PSS. [5] It's important to note that PSS is not two different standards with the DSL and C++ formats used. The tools used will be able to consume both formats. [7]

When creating the tests in PSS the idea is to focus on the test intent and an abstract specification that is generally applicable and not specified for a particular case. With abstract models the test writer can focus on the intent, PSS will handle the details and build the model. The PSS tool also has the ability to infer behaviour. If the tool is told that something has to happen, but that something has specific requirements, the tool can figure out what is needed to complete the requirements. An example scenario below. [7]

3.2 Example scenario

A short example would be if the test writer wanted for "A" to happen which has a buffer input. The tool will figure out that something has to happen before "A" so that it can be fed with a buffer. There could be something called "B" that has a buffer output, but a data stream input, so something would need to feed that "B". There

could also be “C” that feeds “A” with the buffer, but “C” also has a buffer input so something would need to feed it. Or simply there could just be a “D” that has a buffer output but no input. Just specifying this one thing that something has to happen, the tool can figure out all these and more different scenarios that will complete the given requirement. Visualization shown in figure 1.

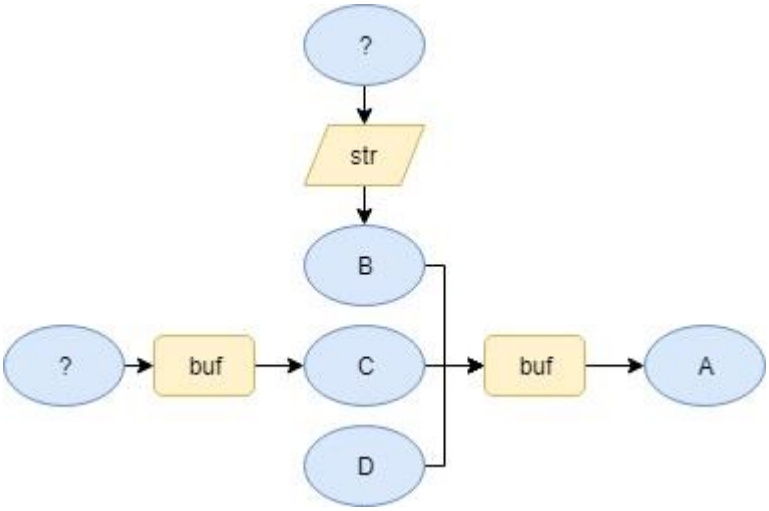

Figure 1. Visualization of the example.

3.3 Modelling concepts in PSS

Actions in PSS define behaviours and they handle data flow and require resources. Data flow can be data streams or buffers. Data streams require parallel objects to produce and consume data, because the data isn’t stored anywhere. Buffer objects have their own storage, so the produced data doesn’t have to be consumed immediately. *Activity* is an abstract, partial specification of a scenario. Activities also allow scheduling behaviours. [7]

Actions run sequentially in PSS tool by default, but they can be grouped under *sequence* to ensure that they are executed sequentially. Actions under *parallel* will be run at the same time. Actions can also be grouped *select* which allows the tool to choose any actions for execution that will support the test and is within constraints. The if-else, do-while, repeat and foreach statements can also be used. *Schedule* lets

the tool choose any execution order for the actions that is legal. Elements are grouped into *components* for reuse and composition. [7]

Resources can also be defined. Resources are grouped in *pools* to define which actions have access to specific resources. Those resources can be *locked* or *shared* to put a limit on how many can be in use. If a pool with 2 resources with both of them locked, nothing else can run concurrently that would use the same resources. Sharing resource means that those resources can run concurrently, and they can also be using the same resource if possible. [7]

Below is simple example of an activity in PSS. [7]

```
activity {
   that;
   do an_a;
   parallel {a1, a2};
   sequence {a3, a3};
   select {a5, a6};
   schedule {a7, a9};
   if (i == 0) {a9};
   else {a10};
   repeat (2) {a11, a12};
   foreach (arr[j]) {
      a13 with {a13.val == arr[j];};
```

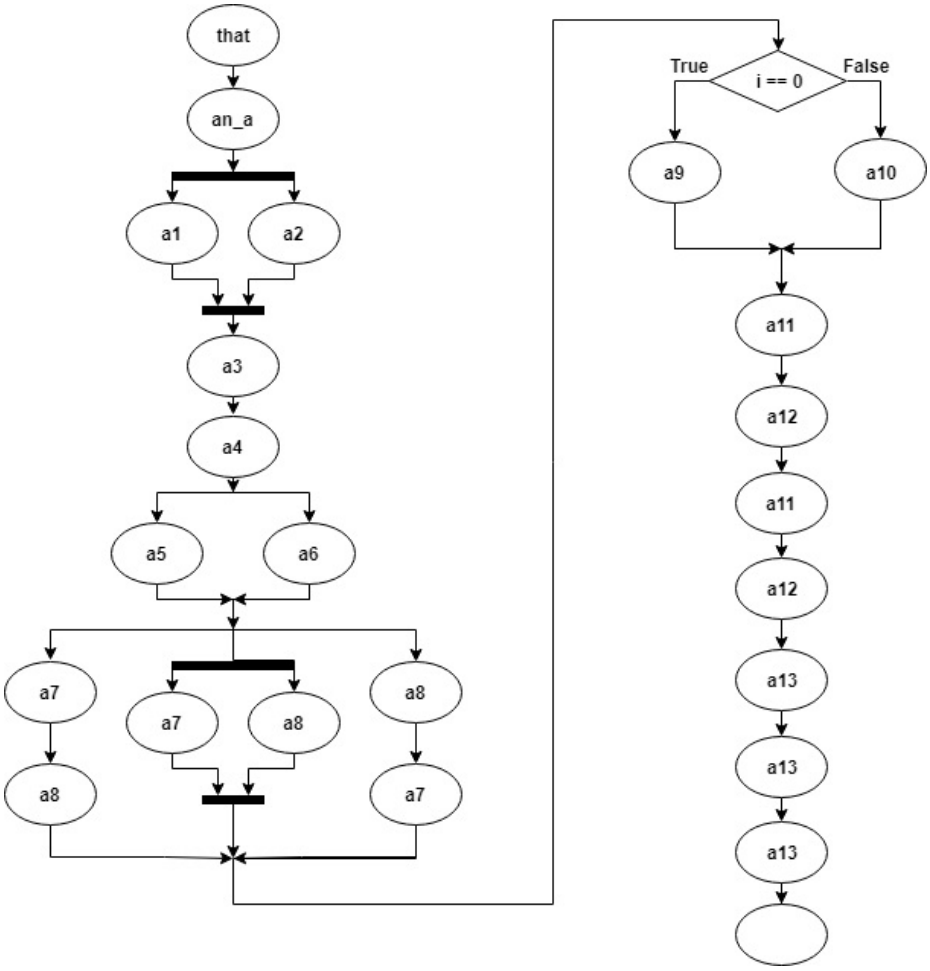

Figure 2. Flowchart of the example code.

3.4 How will it work

When using PSS for verification first you need to describe an abstract PSS model. This description goes into a PSS processing tool, in which a semantic data model and a static structure will be built. This static structure is not fully complete but has empty spaces where the tool can explore different scenarios and possible ways to complete the given critical intent. Scheduling and algebraic relationships for the built model are solved in the PSS solving platform. After this a scenario can be implemented for a target-platform where the tool can talk to the target-platform user code. The tool can then call functions in the user code for tests and make sure that the code that's being verified works as intended. [7]

If the tool is given a specification that is incomplete (partial), it will infer the execution of additional actions and other model elements to make the specification complete and valid. This way from a single partial specification, multiple different scenarios may be derived while the critical intent is still being satisfied. [5]

3.5 Tool flow

Tool flow for the PSS begins at having an abstract partial specification in either the DSL or C++ format which will be compiled. After compiling a scenario model with constraints is ready. The tool then solves the constraints and we will have a solved model, from which tests can be generated. Test will be generated for each possible solution of the specified scenario. [7] Projected tool flow shown below in figure 3.

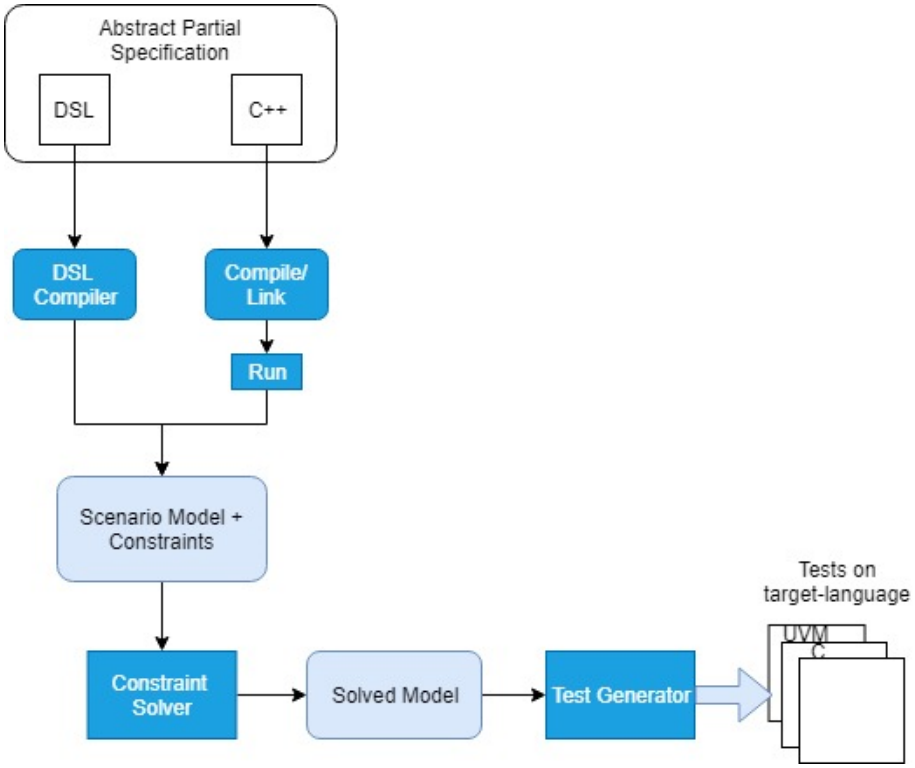

Figure 3. Projected tool flow

4. DISCUSSION

The portable test and stimulus standard provides helpful tools and features for verification. Stimulus and test generation are essential parts in verification and enabling automation for them would be a great way to increase the effectiveness and productivity of verification. With PSS test scenarios from lower-level can be reused at higher level to gain more confidence in the design. This way writing the same tests again in another language for higher-level scenarios or to another target-platform could be prevented.

The ability for the tool to infer behaviours and know what is required for something to happen while only specifying the critical intent is a good way to reduce the amount of code that has to be written. Filling the not specified cases with constrained random data should also give a fairly good coverage gain within a reasonable span of time. Constrained random stimulus generation is generally better for good and fast coverage gain compared to only directed tests.

Unfortunately, because PSS is a fairly new standard (officially released on June 2018), the materials and sources for its actual use were very limited. Instead I had to mostly rely on manuals and tutorials made by the developers of the standard. This might result in knowing the ideal situations from the publisher and how PSS is supposed to work, lacking the possible complexity and practical issues that might exist. No mention of support in EDA tools for the standard was mentioned, but this is probably due to the standard being new.

5. SUMMARY

In this thesis I introduced a simple design and verification flow, which was followed by some of the common challenges in verification. Then in the third chapter I introduced the Portable test and stimulus standard with some of the basic functionality and modelling concepts. Followed by few examples to demonstrate properties of PSS.

Verification is a major part of the whole designing process for SoCs. There are multiple challenges in verifying designs properly and getting a good enough coverage gain for them within a reasonable time period. Translating tests from one platform and one language to another is also a hard and time-consuming process. Solving this with writing abstract, high-level design models and letting the Portable stimulus tool create the tests for target-platforms could help ease and speed up the process. PSS would allow the use of same tests on multiple different platforms and the testing of lower-level cases in higher level scenarios such as blocks in a subsystem-level. This helps gaining confidence in the verification process even further.

With PSS the test writer should focus on high level abstract models rather than focusing on specific details. The main idea is to specify the critical intent and that what you want to verify. The PSS tool fills in the details and explores options to complete the given scenario in different ways to make sure if works as intended. The automation of test cases and test suites is a key purpose of PSS.

6. REFERENCES

[1] Fujita, M. (2008). Verification techniques for system-level design. Morgan kaufmann Publishers.

[2] Moretti, G. (2017). Accellera's support for ESL verification and stimulus reuse. 34(4), 69-75.

[3] Taipaleenmäki N. (2017) Hardware Emulation Environment Setup for Digital Front-End SoC. University of Oulu, Degree Programme in Electrical Engineering. Master's thesis, 47p.

[4] Orava J. (2013) Test case coding for a portable verification environment. University of Oulu, Degree programmet in Electrical engineering. Master's Thesis, 65p.

[5] Accellera, Systems Initiative. (2019) Portable test and stimulus standard manual version 1.0a. URL: https://accellera.org/activities/working-groups/portable-stimulus. Viitattu 22.4.2019.

[6] Breker Verification systems. (2018) Functional intent specification with Portable Stimulus. URL: https://brekersystems.com/products/portable-stimulus/. Viitattu 22.4.2019.

[7] Accellera, Systems Initiative. (2018) Technical Tutorial: "Portable test and stimulus: The next level of verification productivity is here". URL: http://videos.accellera.org/portablestimulus2018/ps84ht9ng3p/. Viitattu 10.4.2019