A New Synthesizable Architecture Approach for Verification Environments Applying Transaction-based Methodology

Raimund Brackebusch*, Steffen Müller, Gabriel Sok-Ye Sokomak, Thorsten Wermke, Frank Grassert*, Dirk Timmermann*, Philips Semiconductors GmbH; *University of Rostock , Germany

Abstract

This paper describes the architecture for a generic functional verification (FV) environment that can be applied to different FV platforms (HDL simulation, simulation acceleration, ASIC emulation, and FPGA prototyping). The approach benefits from the reduction of setup time of the FV environment and minimization of transfer time to another FV platform. This can be achieved by extensively employing reuse methodologies for the components in FV environments. Furthermore, the transaction-based methodology supports the creation of stimuli on a higher abstraction level. Stimuli can be re-applied at different FV platforms. The design of a synthesizable FV environment speeds up the FV run itself. This novel approach combines three test strategies: synthesizable testbenches, transaction-based verification (TBV), and a dedicated architecture for FV environments independent from the FV platform. It has been successful qualified in several industrial applications.

1 Introduction

Continuously increasing chip design complexity requires new ways for the setup of functional verification (FV) in order to reduce effort and time because a typical chip development requires more than one FV platform to perform all necessary checks. Interactively providing complex stimuli to the device-under-test (DUT) in various verification runs on different FV platforms challenges design teams. The overall effort for the setup and performance of FV already takes up to 70% of the total development time of the chip. This results in the main question: what functionality is necessary to be checked and what can we afford, given limited development time and resources.

Some publications dealt with a FV outlined in [1] resulting in the fact that the more carefully FV is planned, the more effectively FV can be performed. This is certainly right. However, introduction of a *generic architecture* for a FV environment is the key to apply reuse, to raise the transparency level of stimuli, and to be able to follow the same architectural approach for different FV platforms. Different FV platforms (simulation acceleration, ASIC emulation or FPGA prototyping) are employed to achieve speed-up and certify the ability to see all signals within the chip design. A *synthesizable approach* allows mapping the architecture onto the different FV platforms and verification time decreases with the speed-up of the used FV platform.

Cohen presented in [2] a *transaction-based verification (TBV) methodology*. This approach is tool-independent and defines exactly the boundary between the reusable architecture approach and the application specific part (mostly interfaces). However, the application specific part of the FV environment reflects back to the used FV platform, e.g. reuse can only be applied between FV environments which base on the same FV platform.

This paper presents an architecture of a generic FV environment which is applicable in different FV platforms. First, we briefly review the state-of-the-art of FV methodologies. Next, we present our strategy and the concept of a FV architecture with mainly synthesizable components. The next section describes two different FV environments which show the realization of the presented approach. After evaluation, we finish with the conclusions and outlook.

2 Functional Verification Today

The current chip (hardware) design process can be roughly described with the following stages that start with the specification[1]. This is coded in an RTL model that needs to be simulated with a HDL simulator requiring a testbench. The process continues with the logic synthesis that ends up with a netlist and needs to be functionally verified, as well. Facing current chip design complexity, the designer switches to another FV platform (Sim Accelerator, ASIC emulation, FPGA prototyping) that requires a further FV environment which usually needs to be built up anew. The delivery of engineering samples finishes the FV part of the chip-design process. This is when the “silicon validation” phase starts.

[1] To short cut the process description, it has been assumed that feasibility study, HW/SW partitioning, and the system-reference modelling has already been done and ended up in the specification.

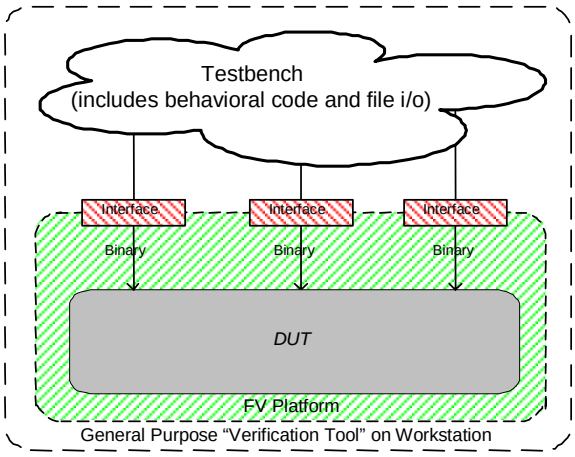

Figure 1 Tool-related verification approach

The FV platforms base on two groups of FV methodologies: either testbench (TB) or a target system (TS). Transferring the chip design to a new FV platform implies an entirely new FV environment. Typically, project management argues rather to avoid the usage of the further FV platform because of setup time and effort reasons than using it even if it would bring a significant value to the development process.
Consequently, the challenge is to reduce overall effort for setup and performance of FV. Also, the effort to transfer the design to another FV platform must be reduced because chip complexities require having multiple, heterogeneous FV platforms.
There are several approaches known to set up a FV environment [3]. Some directly relate to a commercial general-purpose verification tool (see **Figure 1**), other approaches relate to a certain FV methodology. The usage of a verification tool usually implies a tool-related FV environment which offers possible speed-up compared with slow conventional HDL simulation. Others use partly synthesizable FV modules where the synthesized part can be downloaded onto the dedicated FV hardware or use transaction-based verification which is today still tool dependent. Here, the advantage is raising the abstraction level for the stimuli

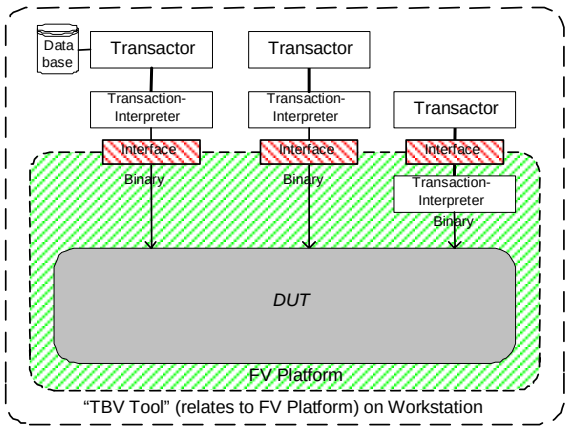

Figure 2 Transaction-based verification approach

[4], see **Figure 2**, and use tool dependent speed-up technologies[2].
Typically, reuse takes place in design teams between different generations of the chip or between projects that are similar with respect to chip application up to a certain level. Switching from one to another FV platform[3] does necessarily result in the setup of a new FV environment where reuse can seldom be applied between different FV environments.

3 Concept

Four items have been identified to improve the productivity, e.g. shorten time-to-market, of the FV phase in a chip development as mentioned in the previous section:

- Reusability
- High-level test creation
- FV platform independence
- Speed of the verification run

Reuse in chip design itself is today the key to a higher productivity. We can learn from this and apply reuse methodologies to the components of the verification environment. The first step is then to unitize the FV environment. In the next step, the interfaces between these modules are defined and standardized. FV modules can be cut between different abstraction levels of stimuli in general or between different types of stimuli e.g. data or control signals. The idea of transaction-based verification (TBV) generally supports both modularization approaches, e.g. is independent from chosen architecture.
A FV module in TBV methodology is divided into high transaction-level and low signal-level. At signal-level, these transactions or tasks are executed, interpreted, and instantly a corresponding sequence of binary values is created and propagated to the device under test (DUT). The modules that create transactions on the high level are called *transactors*, modules that interpret the transactions are called *transaction interpreters* [2]. For each FV module which provides an interface of the DUT a single transactor as well as transaction interpreter is to be modeled.
Transactors are used to initiate data transfers that are executed by the corresponding transaction-interpreters. Often, but not necessarily, transactors are modeled in high-level verification languages [7], [8]. The approach described in this paper avoids the usage of a verification language for the first moment in order

[2] Commercial suppliers drive most TBV solutions. That results in a need to use a certain tool and/or FV platform.
[3] The need for multiple FV environments in chip developments results from the continuously increasing complexity of the design as mentioned above in this paper.

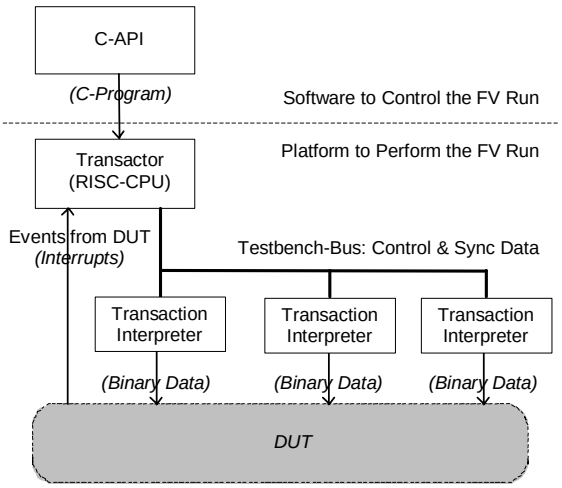

Figure 3 The new architectural approach

to keep tool and platform independence. Verification language may come in when the platform-dependent FV module is going to be implemented.

To verify complex designs, the synchronization of tasks / transactions has to be supported. This is difficult since multiple interfaces/transactors to the DUT need to be modeled. The described approach solves this by using a single, central transactor connected to all transaction interpreters via a standardized communication structure called testbench bus (TB-bus) (see **Figure 3**). All transaction and transfer operations are initiated from this central transactor, so that the synchronization of multiple transaction interpreters is feasible. The central transactor is called TB-processor and implemented as a small RISC processor. Transactor, transactor commands, transaction interpreters, and TB-bus are fully reusable.

For each platform a platform-specific environment is necessary. To reuse a FV environment on different platforms with a minimum effort, the FV environment has to be as much as possible platform-independent. All *synthesizable components* of a verification environment are *FV platform independent*. For that reason, our approach is to build up an environment that is mostly synthesizable. Since synthesizable components can be integrated into FV platforms, less communication between tool interfaces (example: simulation accelerator hardware and connected HDL simulator) is required. In conjunction with hardware-based FV platforms a significant *speed up of the FV environment* can be achieved. Nevertheless, the implementations of memory and I/O from and into the FV platform are specific to the FV platform.

4 Strategy

To realize the idea of a synthesizable TBV-based FV environment we have to modularize [3] and to standardize [5] the environment.

We use a RISC-CPU as single transactor for all transaction-interpreters. In conjunction with a C-Compiler, we are capable to achieve the same abstraction level as a traditional transaction-based environment. There are commercial testbench languages which offer higher flexibility for a description than the C-language but we pursue a tool and platform independent approach. In the future, with an existing methodology, an optimal testbench language can be applied. Moreover, the CPU offers a flexibility that is requested by the functional complexity of the DUT. The CPU initiates all transactions and is responsible for the control flow inside the FV environment, i.e. the verification run is controlled by the C-program that is fed into the CPU. The transactions are transferred from the CPU to the transaction-interpreters via the testbench bus (TB-bus), see **Figure 3**. Such interpreters can also implement the interfaces to the required data for DUT which have to be provided from the verification platform. The CPU realizes a central unit to control and synchronize the interfaces if needed. The main advantage is that the total verification flow is described and documented in a single program. This again supports the reusability on different platforms with the same architecture of the verification environment.

The TB-bus is a synchronous, parallel, and standardized bus that transfers commands, result data from an operation, and synchronization events to the FV modules. The TB-Bus consists of four signals. These are the address of the transaction interpreter, the dedicated command id of FV module, read/write signals and data. The width of the address, command id, and data is adjustable. The CPU is always the master of the bus and it is connected to the TB-bus via the CPU-adapter. All transaction interpreters are connected to the TB-bus. The CPU as master receives data from the interpreters by polling. For higher efficiency, the use of the interrupt port of the CPU instead is possible with modified instruction set.

This structure allows the synchronization of any num-

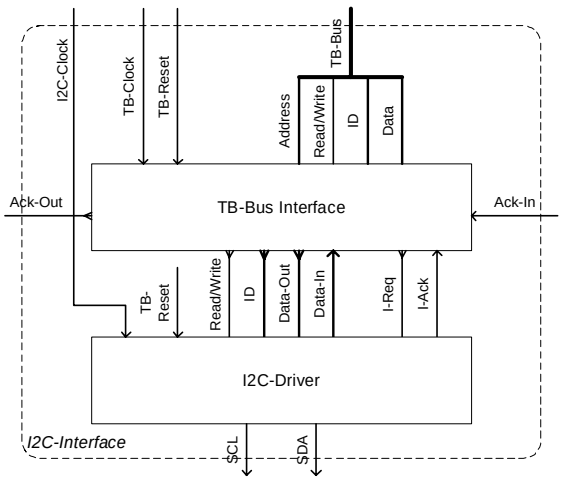

Figure 4 I²C interpreter modeled using TBV

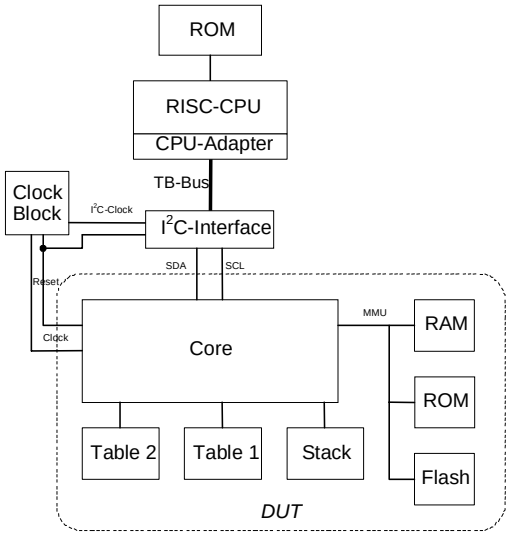

Figure 5 Schematic of the FV environment for a Java processor

ber of transaction interpreters, e.g. the approach is scalable with the number of attached FV modules. The synchronization procedure is similar to an event-based synchronization. The CPU handles requests of the FV modules and drives the synchronization mechanism.
As described in **Figure 4**, the transaction interpreter includes a standard TB-bus interface. The TB-bus interface needs to provide asynchronous communication with the TB-bus since the TB-bus and the transaction-interpreters does not necessarily run with the same clock speed. The second part of transaction-interpreters can be called DUT driver. The driver part in the transactor interpreter is connected to the part "TB-bus Interface" (see **Figure 4**) and implements the specific protocol which is provided by the transaction interpreter. Here, we use I²C. The designer or verification engineer implements this protocol-specific part because intimate knowledge about the DUT-interfaces is required. Transaction interpreters need to be synthesizable.
The loop is closed between CPU, FV module, DUT, and CPU, again, by having the interrupts processed by the CPU itself, e.g. the DUT triggers interrupts which are processed by interrupt service routines of the CPU.

5 Example

We used our approach to verify two designs: a Java processor and a TV sound block for high-end TV solutions.

5.1 Java Processor

As a starting point to see the environment running, we used a Java processor [9] which is controlled by the I²C interface. **Figure 5** shows the FV environment for the Java processor.
Therefore, the FV environment contains only one transactor and proves the feasibility of the concept in general. I²C-transactions are initiated in the C-program of the CPU by using function calls as it could be „i2c_write". Function calls send the communication requests via the TB-bus to the I²C FV module that generates signals in order to stimulate the I²C data interface of the DUT. Following program code gives an example:

```
main()
{ unsigned char data[3], result;

init_ports();        //init CPU-Ports
data[0] = 0;                  //defines Operation: add
data[1] = 35;        //Operand 1
data[2] = 64;        //Operand 2

i2c_write(32, 3, data);    //execute operation
i2c_read (32, 1, &result); //read result
}
```

Figure 6 shows the waveform signal for the above shown C-program example. The first transaction to the I²C interpreter with tb_address=4, tb_id=1 and tb_data=32 brings the I²C interpreter to open the I²C bus in write mode with the I²C address 32. The following three transactions with tb_id=3 write three bytes to the Java processor. The fifth transaction closes the I²C bus. The last three transactions open the I²C bus in read mode again, read one byte and close the I²C bus.

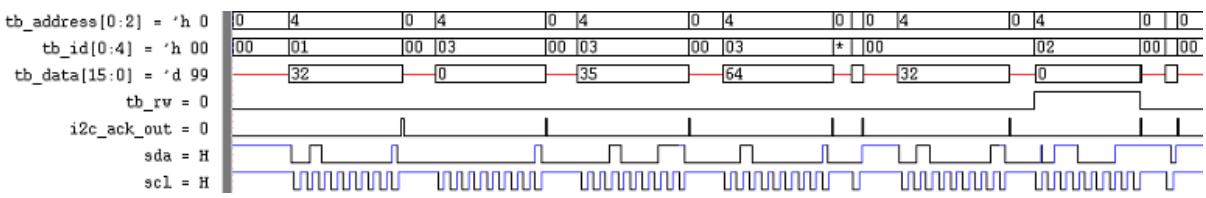

Figure 6 Signal Waveform of Program Example

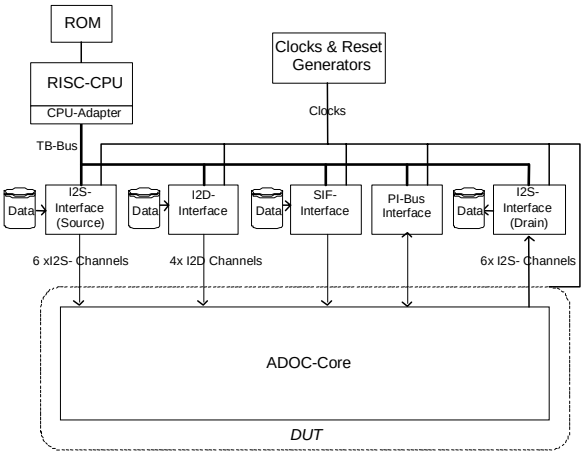

Figure 7 Schematic of the FV environment for a high-end TV sound block (ADOC)

5.2 High-end TV Sound Block

The 1-to-n transaction approach and the investigation of typical traffic on the TB-bus has been tested with the following example, see **Figure 7**. The sound block consists of the interfaces I²D, SIF, I²S source, and I²S drain. The PI-bus interface controls DUT, all needed data sources are controlled by other interpreters. Due to the complexity of the ADOC-core RTL level software verification requires 3 to 5 minutes for just 3 audio frames. This should be contrasted with the frame rate of 48 KHz. Therefore, DUT and the synthesized part of the verification environment have been transferred to a simulation accelerator, requiring less than one person week and yielding a simulation speedup of 40. This is about the same as has been achieved by a hardcoded VHDL testbench but with much greater flexibility. For example, now we were able to control the sound by software during verification.

6 Evaluation

We use a standard TB-bus that allows a scalable FV environment setup and leads to a point-to-multipoint TBV strategy (traditional definition for transactor sees in [6]). This can be achieved by separation of control flow (TB processor) and data flow (transaction-based interface) in the FV environment.

The more fixed components are in the FV environment, the more benefit can be gained from reuse. Minimum effort is the initial setup of the FV environment for an application/project. The number of dedicated transaction interpreters available will increase as the approach has been accepted and is used in practice. There could be the restriction that the transaction interpreter is dedicated to a certain FV platform which would limit the reuse benefit.

The CPU's C-programs controls the verification, e.g. extended programming structures (loops, subroutines, branches, pointer, etc.) can be used. Today, this is only possible with verification tools that execute the testbench activity on a workstation and cannot be downloaded to a FV platform and limits speed.

Table 1 compares the new approach against traditional TBV methodology and shows the advantages of the new approach.

	Traditional TBV	New approach
High level verification task description	Yes	Yes
Standard interface to transaction-interpreters	No	Yes
Communication structure	No	Yes
Synchronization mechanism	No	Yes
Synthesizable	No	Yes(*)
Reuse code and interpreters	Yes	Yes
Tool dependent	Yes	No
Portable to different FV Platforms	No	Yes

* restricted by data I/O of FV platform

Table 1 Comparison of a traditional TBV and the new approach

7 Conclusion

We have introduced a generic architecture for FV environments that merges the methodology of transaction-based verification and synthesizable FV environments. The benefits of this new approach are the extensive reusability of FV modules.

Using the transaction-based methodology in combination with the synthesizable testbench approach offers additional advantages: a small processor as transactor communicates with multiple transaction interpreters via a standardized TB-bus system. The use of a small RISC processor and its software development tools result in the advantage that programs in high-level language (C) control verification runs, e.g., transactions are initiated by a C-program that can be reused between FV environments. The FV environment is scalable since additional FV components can easily connect to the TB-bus and just a C-program needs to be adapted.

The synthesizable TB-bus system takes care about the communication in the FV environment. The attached protocol ensures the data communication between the transactor (processor) and transactor interpreters as well as synchronization between transactor interpreters. Giving the processor the ability to handle interrupts which return from DUT closes the loop between processor and DUT.

Processor and communication structure are synthesizable, e.g. their representation as a netlist can be easily

mapped onto different FV platforms. The limitation of the portability is given by the method how stimuli data is delivered to and results are fetched from the FV platform (ASIC emulation, FPGA prototyping, and simulation acceleration). Furthermore, the implementation of memory differs between the different FV platforms.

8 Outlook

Having such an approach, the next challenge is the deployment of the methodology and installation of the reuse database. Reuse is only applicable when engineers use the approach and contribute to the reuse database. Defining generic interfaces for the data stimuli (in the described approach we separated control and data flow) and identifying appropriate technologies in order to implement these interfaces for certain FV platforms will complete the approach. These interfaces have been described as “partly re-usable, e.g. for the same FV platform” in the description before.

9 References

[1] James, P.: The Five-Day Verification Plan. Synopsys User Group SNUG, Boston, 2000

[2] Cohen, B.: Transaction-Based Verification in HDL. www.vhdlcohen.com

[3] Bergeron, J.: Writing Testbenches – Functional Verification of HDL Models. Kluwer Academic Publishers

[4] Cadence: Transaction-Based Verification White Paper. www.cadence.com

[5] Rashinkar, P., Paterson, P., Singh, L.: System-on-a-chip Verification, Methodology and Techniques. Kluwer Academic Publishers

[6] Müller, S., Hasewinkel, O.: A Framework for the Reuse of Testbenches for Signal Path Designs. Design and Test Conference in Europe, DATE 2000, User Forum

[7] Cadence: A Verification Methodology That Enables Reuse. www.cadence.com

[8] Specman Elite: Verisity Inc, www.verisity.com

[9] Golatowski, F., Preuss, S., Ploog, H., Geithner, T., Cap, C., Timmermann, D.: Integration of Java Processor Core JSM into SmartDev(ices). 8th IEEE International Conference on Emerging Technologies and Factory Automation, Proceedings, ISBN: 0-7803-7241-7, pp. 699-702, Antibes Juan les Pins (France), Oct. 2001