Bulletin of the *Transilvania* University of Braşov • Vol. 11 (60) No. 1 - 2018
Series I: Engineering Sciences

SPECMAN-E TESTBENCH

Al. GROSU[1] M. CARP[2]

Abstract: *The scope of this document is to present a Verification Environment (VE) and how useful is the functional verification of a specific Device Under Test (DUT) in the process of developing and pouring into silicon a digital integrated circuit. The Hardware Verification Language (HVL) used for implementation is Specman. This paper will show the interconnection of the DUT with our VE, coverage results and the bugs found. Also it will point out that verifying the integrated circuit in an early stage will bring savings to the project even though the release date of the chip is delayed.*

Key words: *Specman, Functional Verification, electronics.*

1. Introduction

In electronic design automation, functional verification is the task of verifying that the logic design conforms to specification. In everyday terms, functional verification attempts to answer the question "Does this proposed design do what is intended?" This is a complex task, and takes the majority of time and effort in most large electronic system design projects [3].

Each digital integrated circuit must be functional verified before entering production because today's projects are very complex and these are exposed to many bugs. Most of the time of a project (70-80%) is spent for functional verification and 20-30% for design developing. Also for functional verification are needed more resources than on design side, such as: people who work (2-3 verification engineers to a design engineer), simulator licences (licences for running regressions) etc. Increasing the complexity of the integrated circuits is done in a very fast way. In 1965, Gordon Moore, co-founder of Intel, issued the law that today bears his name. Moore's law can be formulated as follows: "The number of transistors in a chip will double at every 24 months" [2], because of complexity growing of the integrated circuits, the number of existing cases to be verified increase exponential and a specific verification language is needed.

Collett International Research did a study of the errors behind the re-spin process of integrated circuits. The study shows that all of these iterations of the technological process, more than 60% contain errors of logic or functionality.

These kinds of errors can be divided into 3 major categories Figure 1:

- Design errors (Register Transfer Level (RTL) development process), the cause of these

[1] NoBug Consulting SRL, Braşov.
[2] Dept. of Electronics and Computers, *Transilvania* University of Braşov.

errors incorrect or incomplete verification at design level, that's why coverage implementation is needed;

- Specification errors - through a rigorous design check, these types of errors can be corrected even before synthesis phase;
- Reuse of modules and importing Intellectual Properties (IP), some of the logic or functionality errors that led to the resumption of the manufacturing process were generated by the errors already in the reused modules from the oldest generation of the chip or in the imported IPs.

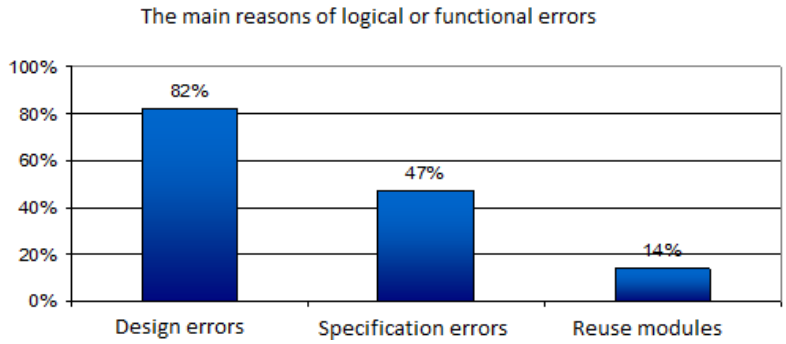

Fig. 1. *Reasons of logical or functional errors* [1]

Further, in this article will be described the developed components of our Test-Bench (TB) (Figure 4) and a short description of the verified circuit, in this case the verified circuit is System Pack Interface - Data Process(SPI_DP).

2. Scope

The scope of this paper is to demonstrate an easy way to implement a verification environment for a small digital integrated circuit where each developed component can be reused in another verification environment. The verification environment will follow a basic and efficient structure which is presented in Figure 2. A basic verification environment has the following components: driver, monitor, checker, predictor and scoreboard.

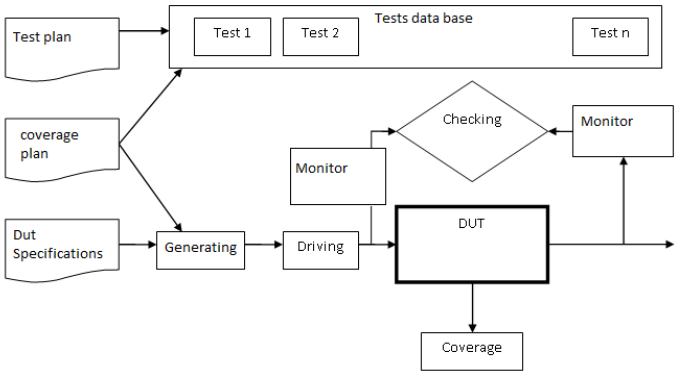

Fig. 2. *Basic verification flow* [1]

For the feature, this basic verification environment structure can be easy modified to a specific methodology as e Reuse Methodology (eRM)[5] or Universal Verification Methodology (UVM) [4], methodologies which are used currently in the industry.

The Universal Verification Methodology (UVM) is a standard being developed by Accellera for the expressed purpose of fostering universal verification IP (VIP) interoperability. Championed and supported by electronics companies throughout the verification ecosystem, the UVM will increase productivity by eliminating the expensive interfacing that typically slows VIP reuse [6].

3. DUT Functional Description

The SPI_DP is an interface for packets and cells transfer between a physical layer (PHY) device and a link layer device. Data is received from the PHY layer on a maximum of 10 ports. The structure of the circuit is presented in Figure 3.

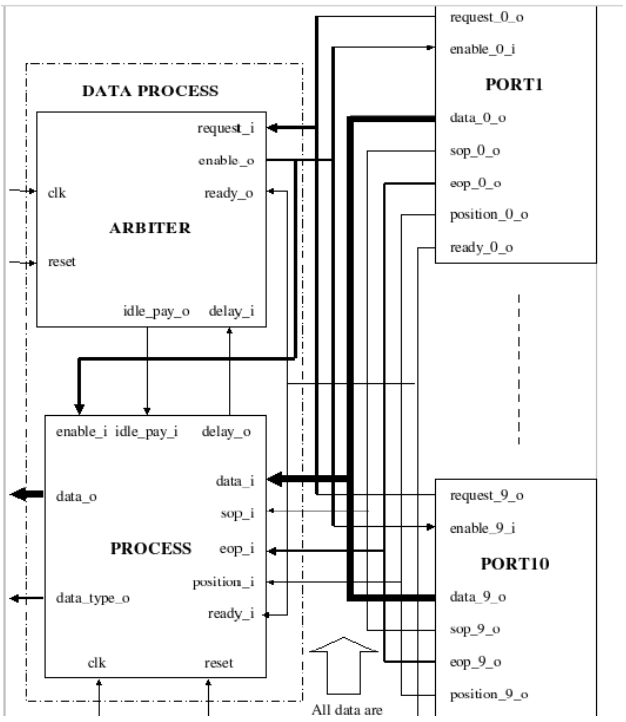

Fig. 3. *SPI_DP architecture*

It has the following features:

- Point to point connection(between single PHY and single Link Layer device);
- Support for 10 ports;
- In band port address, start/end of packet indication, error-control code;
- *Source-synchronous double-edge clocking, 311 MHz minimum;*
- Only a single port is able to transfer data on a specific time, which receives an enable signal from ARBITER;

• The Port's data words are organized into data packets. The packets can be sent to SPI_DP during a single transfer or in bursts. The EOP (End Of Packet) and SOP (Start Of Packet) may be in the same burst or in different bursts. The last word in a burst is signalled with a high level on the “ready_i”;

• The payload data words are transferred to the link layer in the same order as they are received from the Port.

4. Proposed Verification Environment

The first step for developing a verification environment is to have a good understanding of the design, and after that, the next step is to write a verification plan, test plan and a coverage plan

The proposed structure of the verification environment is presented in Figure 4.

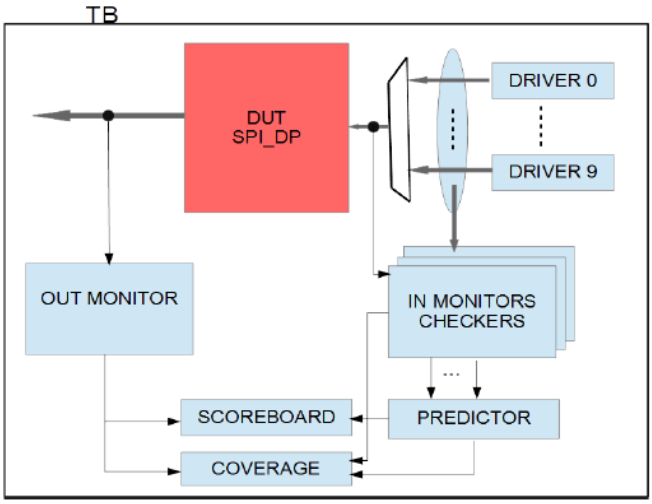

Fig. 4. *SPI_DP verification architecture*

The following components are implemented:

• Driver: drive all the input stimuli of DUT according to the input protocol interface. The port which has packets t send, does a request and puts the data on bus when it is enabled. When the last data from burst is sent, then it will de-assert the request and assert the ready signals. A single driver is implemented for driving the input signals of our device to be verified. This driver is instanciated for each input port.

• Monitor: the monitor block is looking on input signals of the port, sends the collected data to Predictor and also it does the protocol checking. We have an instance for each input port. The monitor block is independent from other worikng blocks and samples continously the signals from the interface.

• Predictor: it has the responsibility to predict the output data of the verified circuit depending of the collected data from monitor. The predicted data is send to Scoreboard block to be mached with the collected data from DUT. It is able to predict a Round Robin arbitration between the input ports.

• Scoreboard: it compares the predicted data with the result from DUT, if the collected data is different than predicted data, it will signal a DUT error. The scoreboard structure is presented in Figure 5.

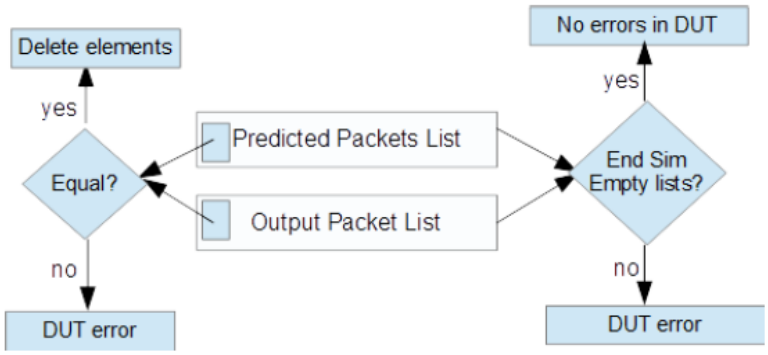

Fig. 5. *Scoreboard structure*

• Coverage: the coverage block is implemented by following a specific coverage plan document and it is used to have a metric of what we proposed to verify and what we verified;

• Output monitor: it collects the received data from DUT and the collected data is send to scoreboard block and also are implemented protocol checkers.

For a feature implementation, our drivers, monitors and coverage will be integrated in an Universal Verification Component (UVC), which is something specific to UVM. The basic structure of an UVC is presented in Figure 6.

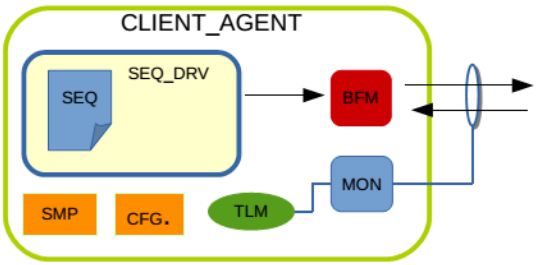

Fig. 6. *UVC*

Using this kind of implementation has a lot of benefits such as: makes it modular and simplifies maintenance, “virtual” sequences enable control at the system level, easy to use test writer interface, architecuture exactly the same across all languages. The UVC will be instanciated in VE and it will be connected with Predictor and Scoreboard modules.

UVM has also a specific logger which makes the debugging of tests more easier because messages are displayed with details of the agent kind which sends the message and there can be used a specific color codding of each agent.

5. Conclusions

The deliverables from a verification environment are status reports which provide information regarding the overall coverage and logs which tell us how each test generated stimulus during running and if there are any errors. When no simulation errors occur, the main results of the verification are provided by coverage. Coverage is a metric that shows how well the proposed testing goals were covered by the developed test base.

For this verification environment the coverage reached a 99% value and 6 bugs were caught in the design. This proves that the verification process is important in the production of digital integrated circuits because the device is subjected to stimulus in a virtual environment where it is cheap to bring any modifications to the RTL.

The way that this test-bench was designed makes it easy in the feature to bring modifications if new features are added to the RTL and also few modifications need to be made in order to align with a specific methodology such as eRM, UVM, the last one being the latest one used in the industry.

References

1. Balint, A.: *Arhitectura mediilor de verificare funcțională reutilizabile* (*Architecture of Reusable Functional Verification Environments*). In: Ph.D. Thesis, *Transilvania* University of Braşov, Braşov, Romania, 2008.
2. Ioniță, F.: *Specman-UVM Based Testbench*. In: Bulletin of the *Transilvania* University of Braşov (2017) Vol. 10 (59), No. 2, p. 175-180.
3. https://en.wikipedia.org/wiki/Functional_verification. Accessed: 02.04.2018.
4. https://en.wikipedia.org/wiki/Universal_Verification_Methodology.Accessed: 02.04. 2018.
5. https://en.wikipedia.org/wiki/ERM_(e_Reuse_Methodology). Accessed: 02.04.2018.
6. https://www.cadence.com/content/cadence-www/global/en_US/home/alliances/ standards-and-languages/universal-verification-methodology.html. Accessed: 23.03.2018.