FUNCTION VERIFICATION OF A USB HOST CONTROLLER

Renata Garcia Oliveira, Edna Barros da Silva Natividade

UFPE, Universidade Federal de Pernambuco, Av. Professor Luis Freire s/n,
Cidade Universitária, Caixa Postal 7851, Recife-PE, Brasil
rgo, ensb@cin.ufpe.br

ABSTRACT

As NRE cost (Non-Recurring Engineering) to production of a single chip is very high, it is highly costly to fix bugs after the manufacturing. However by the end of 80s was standard practice to design a chip and then verify it. Later, a shift took place towards pre-silicon verification, where verification was performed prior to tape-out and in parallel with design [2].

This study aims at an instantiation of verification discipline of the development process ipPROCESS. The case study will be a USB Host Controller. Some tools will also be presented to automate design steps of the case study considerably reducing the chances of occurrence of human error. The USB IP Core was chosen because it has a standard interface design and is consolidated in the market.

Key Words - Functional Verification, USB, Functional Verification Automation, Self-Check Verification.

1. INTRODUCTION

More and more devices are based on CMOS (*Complementary Metal–Oxide–Semiconductor*) technology. The reducing scale of this technology has enabled more features to be integrated in a single chip as Moore's Law predicted. For an example, SoC (*system-on-chip*) offers a whole system integrated in a single chip resulting in a more complex and more required system by the market. SoCs provide high performance, less area, less memory requirement, greater system reliability and lower consumption [1].

As NRE cost (*Non-Recurring Engineering*) to production of a single chip is very high, it is highly costly to fix bugs after the manufacturing. However by the end of 80s was standard practice to design a chip and then verify it. Later, a shift took place towards pre-silicon verification, where verification was performed prior to tape-out and in parallel with design [2].

Verification is a process to demonstrate the design intention was preserved in implementation so Functional Verification must ensure the design desired function and it does not do anything unexpected, but do not consider: power, maximum speed and area. With all these responsibilities Verification spent 70% of total project effort [3].

The functional features that can fill a single chip have increased. This implies a more complex and expensive functional verification system. Chip-respin (chips that fail in the first layout) occurs because of errors and failures during function verification. This work reports the guidelines for functional verification.

The reasons to industrial failured devices or chips-respin were analyzed on 2002 and on 2004. The first reason was the functional errors which were not catch during verification. It is estimated that 60% of the verification effort are now in the debugging process. Verification engineers struggle to find the bugs and then fix them without inserting a new bug [5]. It is more rare find an first-silicon ASIC. The debug time can not be determined, that's mean it can not schedule it [4].

2. VERIFICATION FLOW AND DESIGN

In this section, it will be presented the build verification flow of the IPprocess [6]. The verification flow will be exampled through a Universal Serial Bus (USB) host controller verification.

The USB specification defines different types of transaction and transmission capabilities [7]. The USB verified it is Low and Full Speed Controller and it has just the Control and Bulk Transaction been capable of communicating (setting device and doing mass data transfer) with most devices on the market.

The USB Host Controller design was done in SystemC and was synthesized on Forte tool of Cynthesizer [12]. First, it was development the behavior model and then it was refined to verilog rtl.

The USB Host communicates with the Avalon Bus and the UTMI Low Pin Interface Bus (ULPI) [11]. USB interfaces with the host controller driver (HCD) of the µClinux operational system and the system memory through the Avalon bus and has to transmit/receive data to/from an USB PHY according to the ULPI interface at 60MHz. To achieve this requirement in design, was initially used a 1024x8 bits register bank that worked as a buffer to store

the transmitted/received packet.

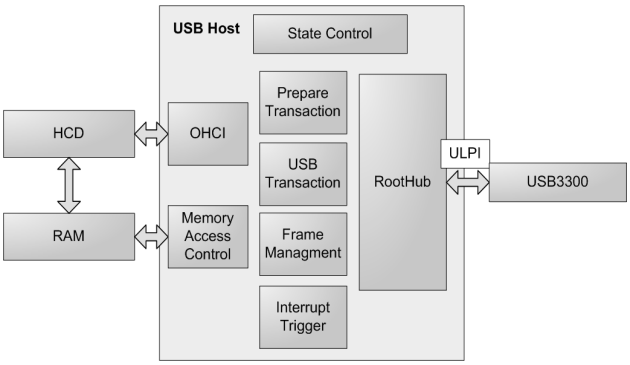

Figure 1. USB host controller architecture.

The USB Transaction is responsible for looking control registers, getting data memory through MAC, adding CRC16 [7], assembling the packets and sending to RootHub module. State Control manages the USB state and lets visible for all design. The Frame Management counts the frame time and sends frame packet to usb device through USB Transaction and the Interrupt Trigger module manages internal interruptions.

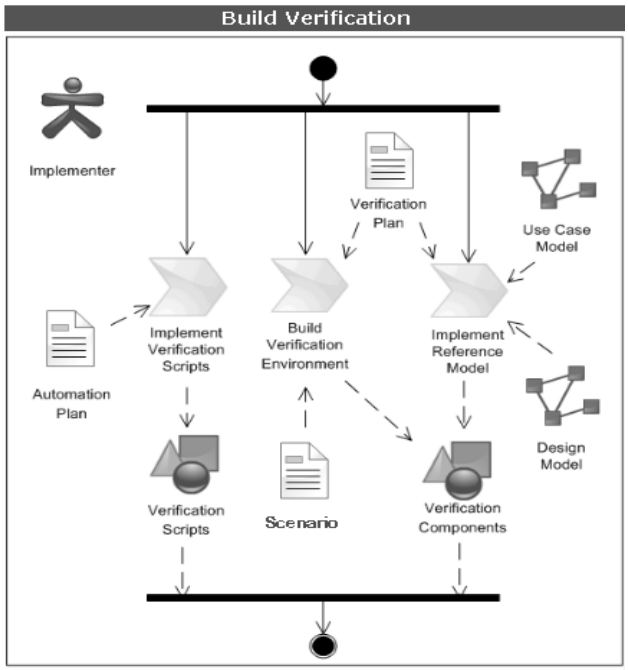

Figure 2. IPprocess Verification Build Flow [6].

The RootHub module is responsible for managing device status and for assembling packets to ULPILink module. This module interfaces with the ULPI (UTMI Low Pins Interface) to communicate with USB3300 board [8].

RootHub has a special importance for verification flow, because it interfaces an external board and the USB was prototype in FPGA, so timing is critical. Then IPprocess verification flow is instantiated to verify the RootHub.

Figure 2 shows the IPprocess Build Verification Flow. There are three steps to be followed and each steps define the input required and output generated. All steps were followed on verification of USB host controller design. From now on, each step will be numbered to facility reference only:

1. Implement Verification Scripts
2. Build Verification Environment
3. Implement Reference Model

The IPprocess verification flow on Roothub will be showed in the next sub-section. The USB instatiation will be showed after.

2.1. Roothub Testbench

Figure 3. Roothub verification environment.

Roothub needs a simulated testbench and a prototyping testbench. In implementation was used SystemC and Forte tools, so it could create just one testbench, synthesizable SistemC. Figure 3 has the result of step 2 in verification flow. The Driver Simulator is responsible to send the stimuli to DUV (Roothub). These stimuli was described as a scenario, the intention is validated the communication with an external board (USB3300) and not to do a stressful stimulus verification.

The Roothub reports the reset device and the status attach. It is also responsible for sending and for receiving packets through ULPI interface to device. This functional environment uses a pin accurate interface.

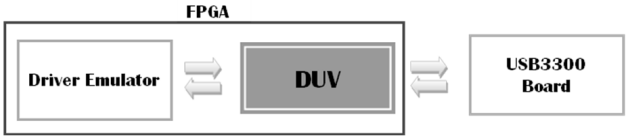

Figure 4. Roothub testbench.

Once the testbench was simulated, the roothub testbench need to be prototype, so it will be validate the correct timing with the external board as showed in Figure 4. The Driver Simulator and the DUV are synthesized by Quartus 7.2 [10] to Stratix II FPGA [10]. If there is any error, it is used the Logic Analyzer [10] to find the error faster, because we know the exact failure point at that moment.

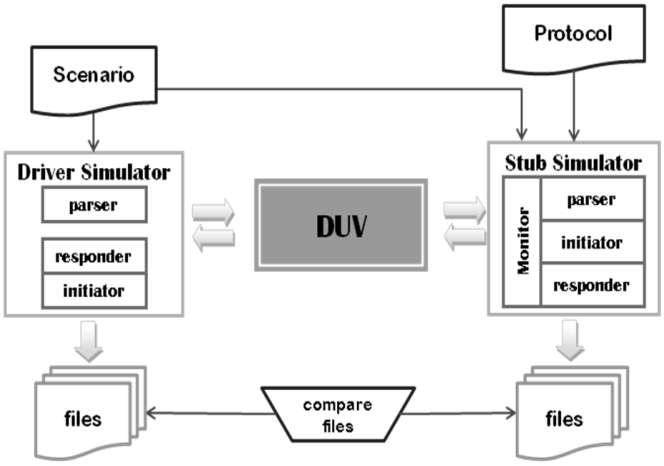

Figure 5. USB Host Testbench

Step 1 of functional verification flow results in platform automation of Figure 4. This improves the time spent on the build and verification process. Below it has the explanation of USB host controller functional verification environment.

2.2. USB Host Controller Testbench

Error! Reference source not found. shows step 2 of functional verification flow. The scenario contains stimuli that driver simulator will pass to DUV through the pin accurate interface and protocol contains protocol answers that the stub simulator can send to DUV. Both simulators have a parser to read these files, an initiator stimulus and a responder stimulus.

The checker was implemented through file compare scripts. Both simulators write the trace simulation so to check the simulation accuracy just compares the files. Step 3 of functional verification flow implements a reference model inside of the stub simulator.

The driver simulator configures the USB registers (OHCI – Open Host Controller Interface) and puts the data on memory system. The stub simulator pretends to be various devices and responds according the USB protocol and ULPI protocol, besides that the stub simulator must stress protocols communication and monitoring the bus looking for failures.

Next subsection will show tools built for automating environment and improving time spent on verification process.

2.2.1. Automation of Environment

There are many parameters to be set on Scenario file, so it was necessary to avoid errors on a recent file created, to do this a php script called Scenario Generator (SG) was created. All parameters can be set by a graphical interface; Figure 6 shows one's SG interface.

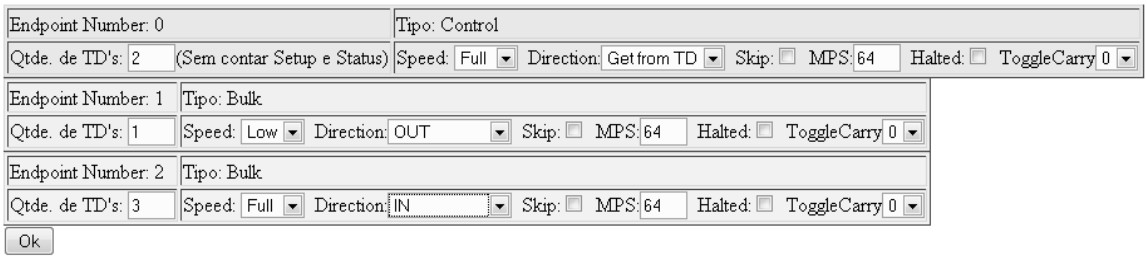

Figure 6. Build Verification Scenario.

2.2.2. Domain Specific Language creates bash script

The DSL Automation Scripts was created to improve the planning of bash script creation. This tool allows:

- To build quickly any bash script using graphical, so it is a flexible tool.
- To improve the script reuse, it aims to reduce the effort to build vital scripts to the verification environment.

Figure 7 shows the USB verification script. Toolbox shows the primary components and below it has the properties of selected components. The Sample.shl file contains model script, the ellipses represent the variable environment, the HC box and dark circle represents a target of the script, and it rests the light circles which represents parameters target. For example, target all does not need a parameter, because this target simulates all scenarios.

3. CONCLUSION AND FUTURE WORK

Table 1. Coverage Analysis.

	# of events specified	# of events found	Percent Coverage
USB Protocol	481	444*	100%
ULPI	444	370*	100%
Total	925	814*	100%

*Some events specified for the design were never found; it was made an analysis and discussed with the design team. It was concluded that these events were impossible.

This paper presented the IPprocess verification flow and it's instantiation on a USB host controller design. It was used files to set the simulation so further functionality can be added and verified any time.

During the USB verification was created thirty seven scenarios and twenty four protocols files, as one scenario and one protocol file is necessary to simulate the environment, 37x22=814 different environment configuration was simulated. Fifty four design errors were found. Table 1 shows more details.

When scenarios were simulated without any scripts, it took almost two days to verify a scenario. So it was showed the automation structure and how it helps to improve the environment. The complete environment permits to verify all different configurations in less than three days.

Future work will focus on improving coverage metrics and random stimulus generation towards full coverage.

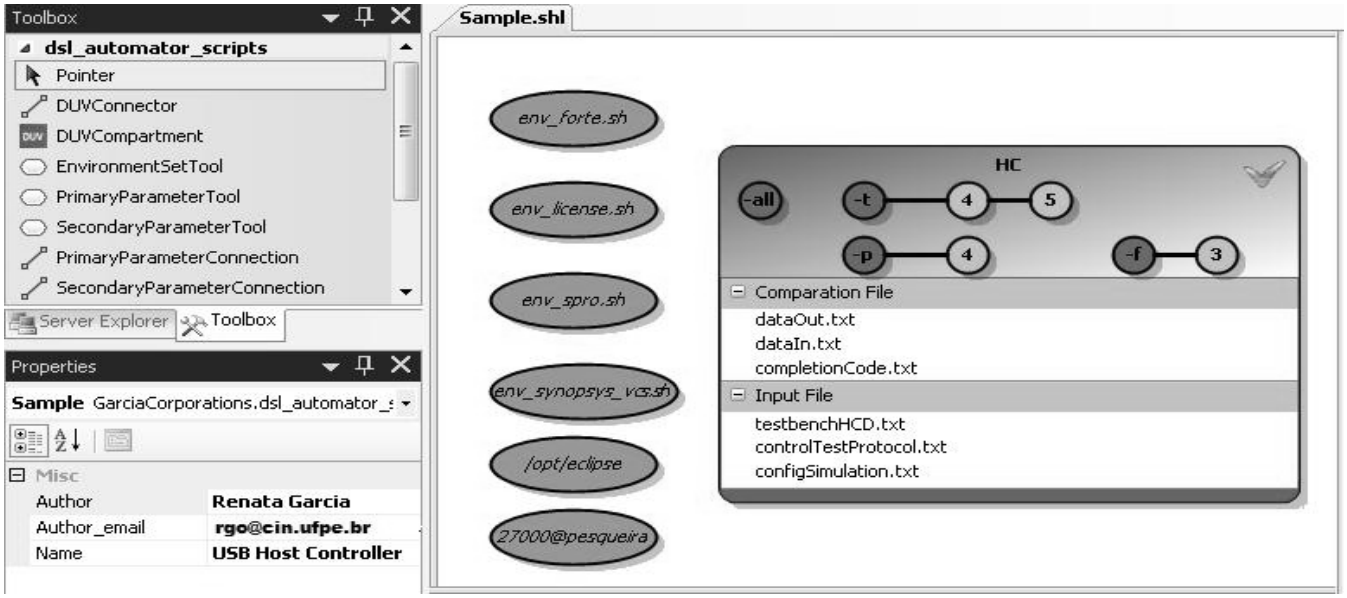

Figure 7. DSL Automation Script.

4. REFERENCES

[1] Silicon Far East. 2005. Site sobre Fabricaçõ de Semicondutores. [Online] 2005. [Citado em: 25 de ago de 2008.] http://www.siliconfareast.com/soc.htm.

[2] Wilson, Chris. 2008. Leveraging Design Insight for Intelligent Verification Methodologies. *EDA Design Line.* [Online] 2008. [Citado em: 25 de ago de 2008.] http://www.edadesignline.com/showArticle.jhtml;jsessionid=JSXAPDWQSBUSUQSNDLRSKH0CJUNN2JVN?articleID=208401374&queryText=verification.

[3] Bergeron, J. 2003. *Writing Testbenches: Functional Verification of HDL Models.* s.l. : Second Edition, Kluwer Academic Publishers, 2003.

[4] Lange, Michelle e Boer, TJ. 2007. Effective Functional verification Methodologies fo D0-254 Level A/B and other Safety-Critical Devices. s.l. : Mentor Graphics Corporation, 2007. Rev. 1.1. White Paper.

[5] Mentor Graphics Corporation, TNI-Software, D0-254 User's Group, HighRely Inc. 2007. D0-254: Understanding the Issues that Impact Business. 2007. Rev. 2.0. White Paper.

[6] ipPROCESS. 2007. Modelagem do processo ipPROCESS. [Online] 2007. [Citado em: 25 de ago de 2008.] http://www.lincs.org.br/ipprocess.

[7] USB.org. Universal Serial Bus. [Online] USB Implementers Forum, Inc., creators of USB technology. [Citado em: 24 de nov de 2008.] http://www.usb.org/.

[8] SMSC. 2008. USB3300 Hi-Speed USB Host, Device or OTG PHY with ULPI Low Pin Interface. [Online] SMSC, 2008. [Citado em: 24 de nov de 2008.] http://www.smsc.com/main/catalog/usb3300.html.

[9] BRAZIL-IP. 2005. [Online] 2005. [Citado em: 25 de ago de 2008.] http://www.brazilip.org.br.

[10] Altera Corporation. 2008. Altera Literature. *Design Debugging Using the SignalTap II Embedded Logic Analyzer.* [Online] Nov de 2008. [Citado em: 8 de nov de 2008.] http://www.altera.com/literature/hb/qts/qts_qii53009.pdf.

[11] ULPI. 2004. UTMI+ Low Pin Interface (ULPI) Specification. 2004.

[12]. Cynthesizer 3.6 Automated Design partitioning, interface Creation, Scalability Enhancements, & More [Citado em: 27 de julho de 2009.] http://www.forteds.com/.