Long-term on-chip verification of systems with logical events scattered in time

J. Viejo*, J.I. Villar, J. Juan, A. Millan, E. Ostua, J. Quiros

University of Seville, Electronic Technology Department, Seville, Spain

KEYWORDS:

FPGA

Long-term verification

On-chip verification

PicoBlaze

A B S T R A C T

Traditional on-chip and off-chip logic analyzers present important shortcomings when used for the long-term verification of industrial embedded systems, forcing the designer to implement ad hoc verification solutions. This paper introduces a suitable solution for long-term verification of FPGA-based designs consisting of a verification core that uses the PicoBlaze microcontroller, dedicated logic and a serial port communication in order to monitor the internal signals of the system in a continuous way. The core design focuses on low resource requirements and has been successfully applied to the verification of a real industrial synchronization platform showing remarkable advantages over commercial on-chip solutions like Xilinx's ChipScope Pro. Moreover, in order to improve the reusability of this core a software tool has been developed to automatically include the verification core in any specific system.

1. Introduction

Over the past decade, FPGAs have become the major implementation technology for industrial embedded digital systems due to their fast prototyping, short time-to-market and increasing capabilities. The always reducing cost of FPGAs make them suitable not only during the prototype phase but also for low and medium volume production. The re-programmability of FPGA chips is also very valuable during the design and production phases of the system allowing for easier verification, debugging and support of the systems.

FPGA verification can be done by simulation and hardware execution [1–4]. Simulation is especially useful during the first stages of the design flow to assure the correct operation of the systems. A variety of verification techniques are applied at the model and RTL levels like functional verification or regression testing. But there are scenarios where simulation is not a feasible approach, like the verification of a whole complex system, because the simulation time would be huge and/or computational resources may be exhausted. In these cases, hardware execution is an interesting alternative derived from the re-programmable nature of FPGAs that permits the observation of the design under study in real time during its operation. Such observations are carried out in several ways including external (off-chip) and internal (on-chip) logic analyzers. On-chip logic analyzers like ChipScope Pro [5] are able to acquire data at a very high frequency and store the results in memory for later processing, so data acquisition is limited by the storage resources available in the chip [6,7]. While this limitation is not a big problem for the verification of many types of systems, it is important to note that on-chip logic analyzers share the resources with the system under test and may occupy an important part of the available area, requiring significantly more resources during verification than in the final production system.

A typical case is the verification of the correct long-term operation of a system. It is particularly useful to test the robustness of the implementation in industrial aggressive environments where systems are supposed to operate continuously for months or years. In this case, the gathering of data from internal signals over a long period of time should be possible. These data should be transferred off-chip in order to avoid the use of internal storage resources (that would be exhausted over time) and to allow the continuous monitoring of the system.

Neither standard on-chip or off-chip solutions fit well to do this kind of long-term verification so designers typically have to develop ad hoc solutions (test logic and tools) for each design. A good example is a network synchronization system for remote terminal units previously developed by the authors [8] where internal data need be collected every few seconds for a period of days or even months.

This contribution introduces a more general solution for long-term verification of digital systems implemented on FPGA that can be adapted to several specific problems to avoid the cost of designing custom verification logic. The proposed solution takes the form of a test core based on the PicoBlaze microcontroller and an associated software tool that can greatly facilitate long term verification of complex systems with minimal resources or external equipment requirements when compared to traditional on-chip or off-chip logic analyzers.

The paper is organized as follows: in Section 2 an outline of common verification tools is presented from the point of view of their applicability to scattered event acquisition and analysis, Section 3 describes the architecture of the proposed long-term on-chip data acquisition core. In Section 4 the software tool that automates the generation and inclusion of the core in a specific design and performs the analysis of the captured data will be presented. In Section 5 the proposed solution and the commercial ChipScope Pro test system are compared against a real application. Finally, Section 6 summarizes the most relevant conclusions.

2. Current solutions for system verification

Typically, there are three main types of solutions when approaching digital system verification: standalone logic analyzers, on-chip logic analyzers and custom cores for specific purposes.

Standalone Logic Analyzers (SLAs) are very powerful tools for debugging an already implemented design. This type of equipment is able to acquire data at a very high frequency from any signal that can be accessed at the pins of the chip. Moreover, they may have a large number of channels (100 or more) that makes them a very useful tool for debugging high speed buses and signals between components. The main disadvantage of SLAs is that they cannot reach signals inside the chip. To overcome this issue, designs are modified in order to route the desired signals to external pins accessible by the SLA thus modifying the characteristics and timing parameters of the original design.

An evolution of SLAs are On-chip Logic Analyzers (OLAs) like Xilinx's ChipScope Pro, that have become very popular in the field of programmable logic. This type of logic analyzers are hardware modules that connect to the desired signals inside the chip and communicate over a standard bus (usually RS232 or JTAG) with a computer that executes software for data analysis. These modules are an intrusive solution since the verified design is different from the production design where the analysis components have been removed.

Both SLAs and OLAs find the size of the captured data to be limited by the size of their storage resources. When these resources are exhausted, the monitoring process need be stopped and the captured data is transmitted to a processing system.

On the other hand, the testing of some applications requires data to be captured and processed in a continuous way for a long time (from days to months). To perform this type of testing, developers usually create custom debugging cores (CDCs) to overcome the limitations of both SLAs and OLAs [9,10].

3. Logical event analyzer

To overcome the cost of designing a CDC for every particular application, we propose a general purpose device, the Logical Event Analyzer (LEA), that can fill the gap between SLAs and OLAs and substitute CDCs in several practical cases. As it can be observed in Fig. 1, SLAs and OLAs are best suited to verify high-speed systems where the number of samples is not a critical aspect. However, LEA can be used for debugging systems where it is necessary to capture a large number of samples scattered in time. The LEA also features a much lower footprint than an OLA.

The architecture of the proposed analyzer is based on the PicoBlaze microprocessor from Xilinx [11], but it could be replaced by any other equivalent soft microcontroller like Mico8 [12] from Lattice. Fig. 2 shows the block diagram of the analyzer. As it can be observed from the diagram, a set of input ports of PicoBlaze are reserved for trigger, clock and communication control signals. The remaining ports are dedicated to capture data signals. PicoBlaze allows the addressing of 256 8-bit ports. Thus, using a single PicoBlaze module and dedicating N ports to trigger, clock and control signals the analyzer can capture (256-N) $*$ 8 binary signals. However, as it will be discussed later, the number of data signals that can be acquired is also limited by the rate at which captured data can be transmitted out of chip.

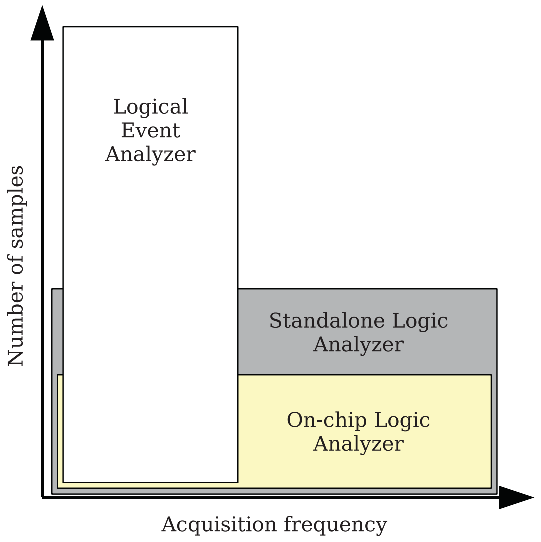


Fig. 1. Field of applicability of the analyzed verification tools.

PicoBlaze uses one of its output ports to communicate with an UART that is in charge of transmitting the captured data through a serial line. The UART has a *half_full_buffer* signal which indicates that the FIFO is half full, and its baud rate can be set as needed. PicoBlaze will use the *half_full_buffer* signal to control the data transmission to the UART. Assuming an UART configuration of fixed data format "8N1" (8 data bits, 1 stop bit and no parity), communication through the RX and TX signals (no other signals needed), and flow control disabled; the maximum bit rate (bps) that can be obtained is calculated according to (1).

$$bitrate_{max} = \frac{baudrate \times 8}{10} \tag{1}$$

Bit rates calculated for some typical baud rates are shown in Table 1. With these numbers, if the LEA is configured to acquire the largest possible number of signals, that is, only one port is used for trigger, clock and control signals, the maximum number of signals that can be captured is $255 \times 8 = 2040$ binary signals. According to Table 1, a rate of 4800 baud is enough to transmit this number of bits if the interval between events is at least one second. In a general way, the baud rate required to transmit a variable number of signals (*numsignals*) connected to the data ports in t seconds can be calculated according to (2).

$$baudrate = \frac{numsignals}{t} \times \frac{10}{8} \tag{2}$$

Nevertheless, if it is necessary to capture more data signals, a simple solution is to add an external n bits register to extend the port selection signal *port_id* as shown in Fig. 2. By using this alternative, the maximum number of signals ($numsignals_{max}$) that can be sampled is calculated according to (3). It must be considered that $N < 256 \times 2^n$ in order to dedicate at least one port for data signals.

$$numsignals_{max} = (256 \times 2^n - N) \times 8 \tag{3}$$

From (2) and (3), the minimum baud rate required to transmit the information connected to the data ports can be calculated according to (4).

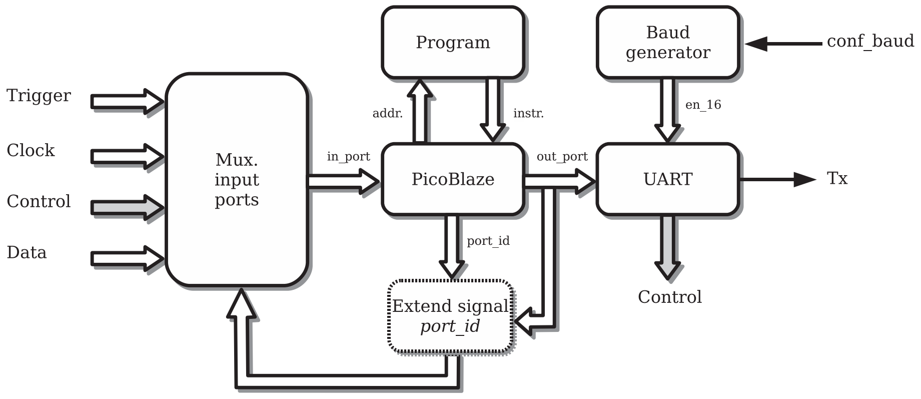


Fig. 2. Architecture of the logical event analyzer.

Table 1
Bit rates calculated for some typical baud rates.

Baud rate	Bit rate
4800	3840
9600	7680
19,200	15,360
38,400	30,720
57,600	46,080
115,200	92,160

$$baudrate_{min} = \frac{(256 \times 2^n - N) \times 10}{t} \tag{4}$$

From the above analysis, it can be easily seen that the sampling frequency is clearly limited by the UART baud rate. For example, if 32-bit data is to be monitored, the LEA can sample above 2800 data values per second transmitting at a 115,200 baud rate. This frequency should be enough for a system which events are scattered in time (events occurring in the range of 1 ms). When events are not scattered in time, the system would require a more sophisticated transmission system (USB, Ethernet, etc.) which, on the other hand, would greatly increase the complexity of the final system.

Finally, the program module corresponds to the program that will be run by PicoBlaze. This program performs the following tasks:

1. Trigger condition verification. PicoBlaze reads the ports assigned to the trigger signals and applies the configured boolean function. If the condition is verified, data acquisition starts.
2. Data acquisition according to the clock signal. The clock signal corresponds to an event of the designed system, so that whenever this event occurs, PicoBlaze starts to read the data connected to the input ports.
3. Data processing and transmission. This processing consists of calculating a checksum of the transmitted data so that the receiver can verify the correct reception of information. Data will be sent to the UART.
4. Communication control. Periodically, the microprocessor will check the status of *half_full_buffer* signal. If it is active PicoBlaze will wait for the FIFO to become half empty before sending more data.

4. Software tool

To automate the verification process using the proposed analyzer, a graphical and multi-platform tool has been developed in Java. The two main tasks performed by this software are depicted in Fig. 3: (1) automatic generation of the hardware core, and (2) data acquisition and processing.

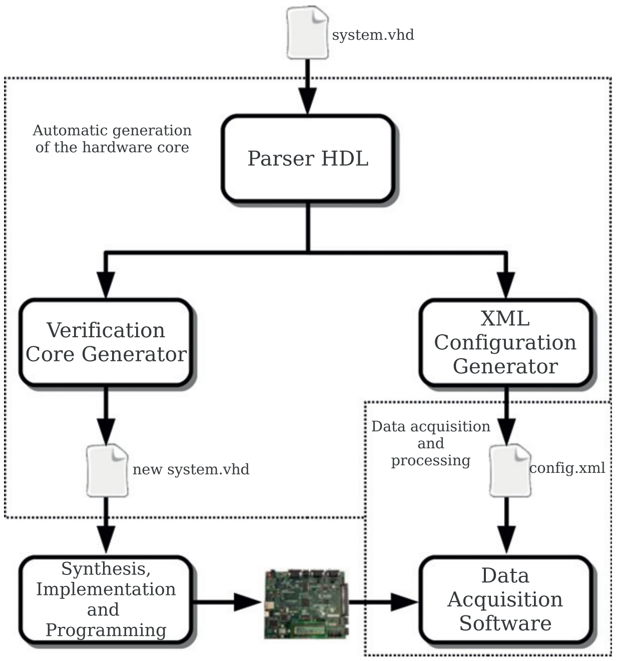

system.vhd
Automatic generation of the hardware core
Parser HDL
Verification Core Generator
XML Configuration Generator
Data acquisition and processing
new system.vhd
config.xml
Synthesis, Implementation and Programming
Data Acquisition Software

Fig. 3. Main tasks performed by the software tool.

Regarding the first task, the tool must parse the system's top level module described using a hardware description language (HDL). Next, the tool facilitates the selection of the trigger, clock and data signals and the configuration of the triggering condition. Finally, the software automatically generates the verification core according to the specified signals and provides a new system top level that includes the generated core. Moreover, this tool generates a configuration file in XML format that is used by the data acquisition software and contains the verification system information necessary to process the received data correctly.

In relation to the second task, once the new system is implemented on the programmable device, the tool is in charge of acquiring the data transmitted by the verification core. To do this, the software allows the user to: (1) configure the baud rate, (2) start and stop the acquisition process, and (3) format and depict the received data correctly using the configuration file. Finally, this information can be stored for later processing.

5. Application example and results

In this section we describe the application of the LEA to the on-chip verification of a SNTP client and server fully implemented in hardware. SNTP is a simplified version of the more general Network Time Protocol (NTP) [13] that is commonly used for synchronizing the clocks of computer systems over data networks such as the internet. The operation of this protocol is to send periodic time requests to a server synchronized with an accurate time source like a GPS receiver at request intervals that can vary from a few seconds to several minutes. When the server reply is received, the client uses a set of timestamps to calculate the round trip time and the time offset between the client's and server's clocks.

The client can then adjust its local clock based on these calculations. In a typical scenario, the client will be accurately synchronized to the server only after several request–response cycles. Since the time between requests can vary from a few seconds to several minutes the most important aspect in the testing analysis of these systems is not the speed at which samples are acquired but to capture a large number of system events, covering a wide time interval.

On-chip verification tools like ChipScope Pro feature high frequency sampling which allow the testing of high-speed buses and systems, but they face some limitations regarding the maximum number of samples that can be obtained from the system. This is mainly due to: (1) internal resources of the FPGA are used to store the samples, and (2) some of these resources, depending on the type of programmable device used, are often limited.

The number of LUTs, FFs and BRAMs used by ChipScope Pro depending on the number of signals and the number of samples are shown in Figs. 4a, 5a, and 6a, respectively. Figs. 4b, 5b, and 6b show the number of LUTs, FFs and BRAMs used by LEA depending on the number of signals and the number of samples.

As it can be observed from Figs. 4a and 5a, LUTs and FFs depend mainly on the number of signals. However, Fig. 6a shows that the main problem when performing on-chip verification using such tools is that the number of BRAMs used is directly proportional to the number of signals and the number of samples. Furthermore, an additional BRAM must be included for each added Integrated Logic Analyzer (ILA) since each ILA can capture a maximum of 256 signals only. Thus, considering that BRAMs are the most limited resource and fixing a number of signals to capture, this type of simulation is unfeasible if the goal is to capture a large number of samples. Comparing these results to data obtained using LEA,

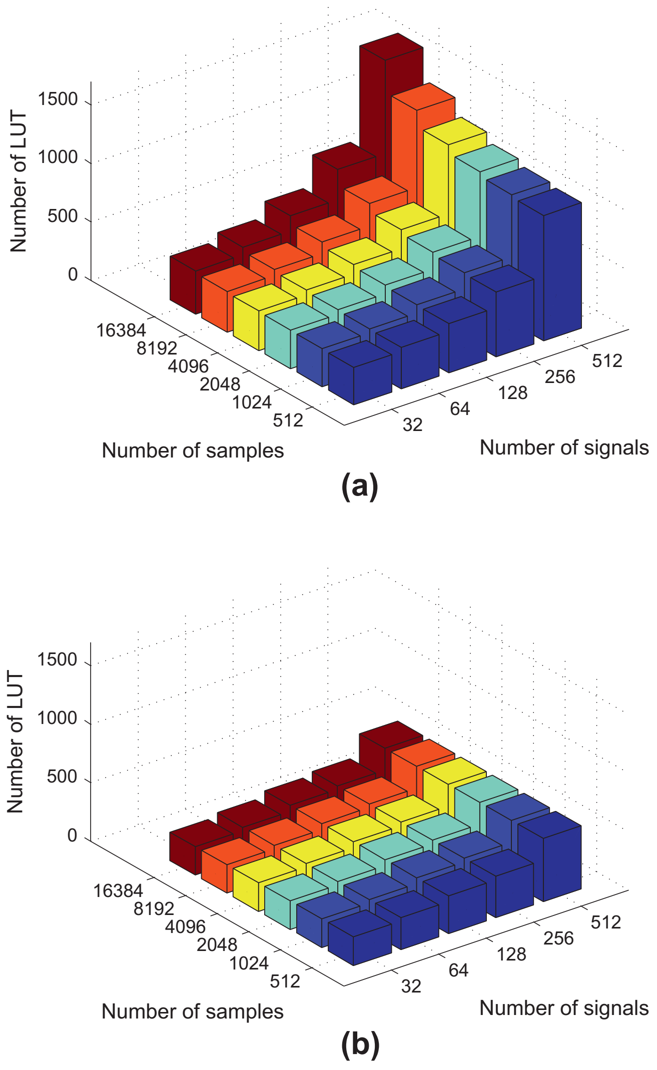


Fig. 4. Number of LUTs depending on the number of signal and samples using: (a) ChipScope Pro, and (b) LEA.

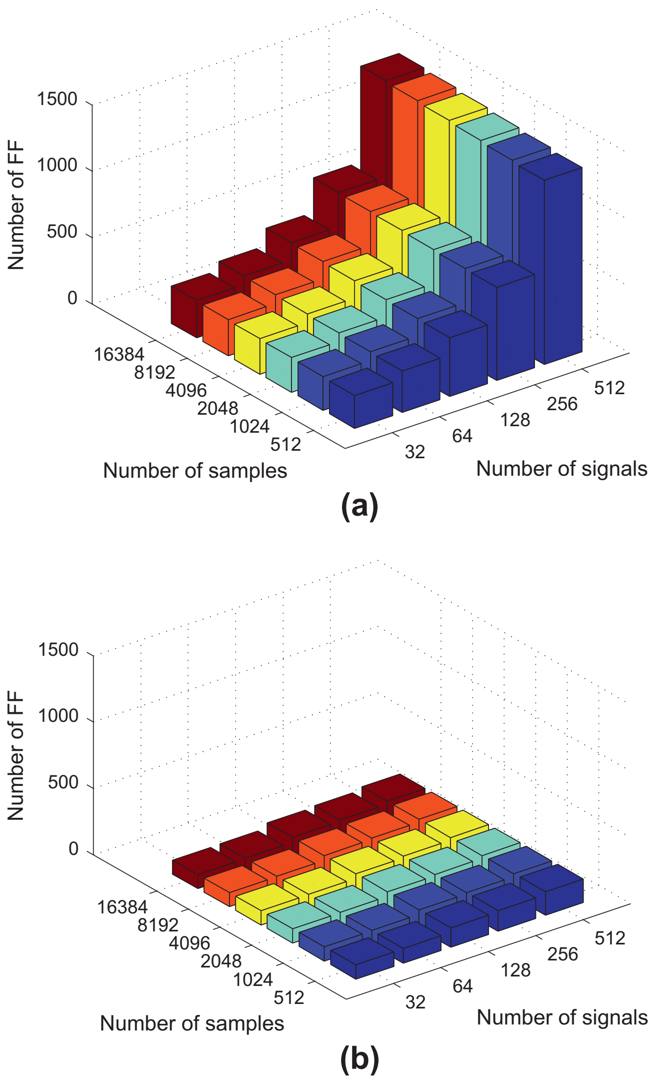
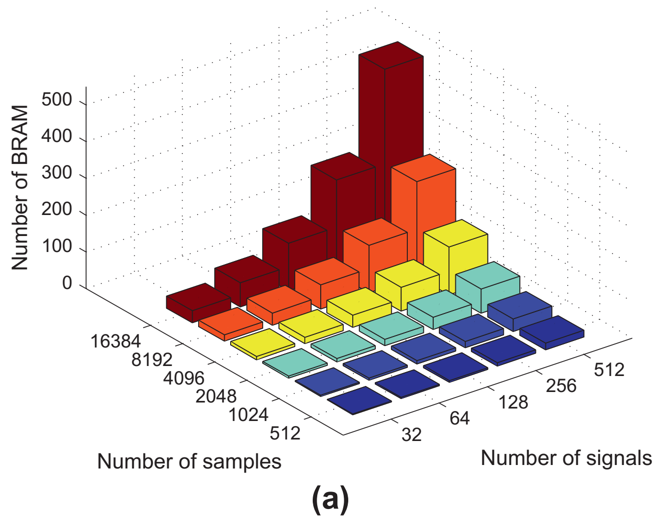


Fig. 5. Number of FFs depending on the number of signal and samples using: (a) ChipScope Pro, and (b) LEA.



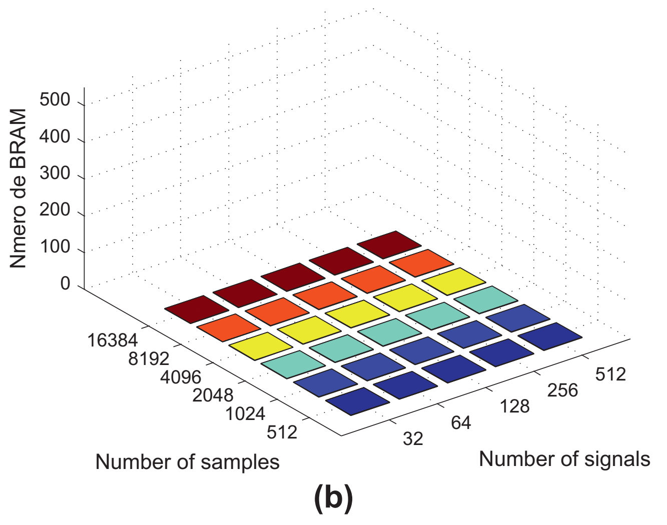


Fig. 6. Number of BRAMs depending on the number of signal and samples using: (a) ChipScope Pro, and (b) LEA.

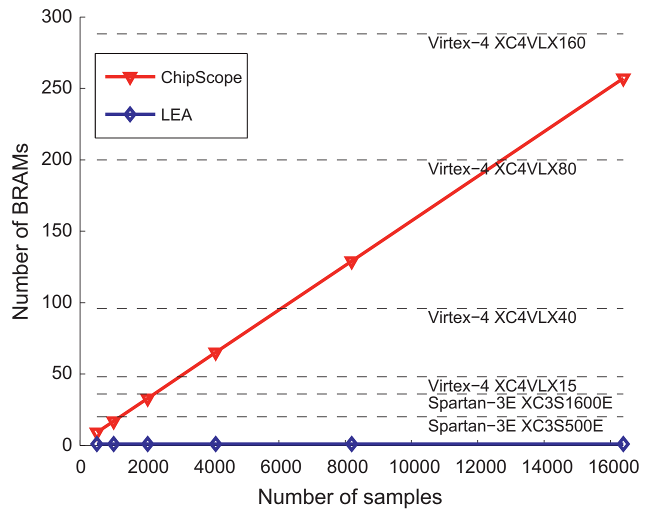


Fig. 7. Percentage of used BRAMs depending on the number of samples for 256 signals using ChipScope Pro and LEA.

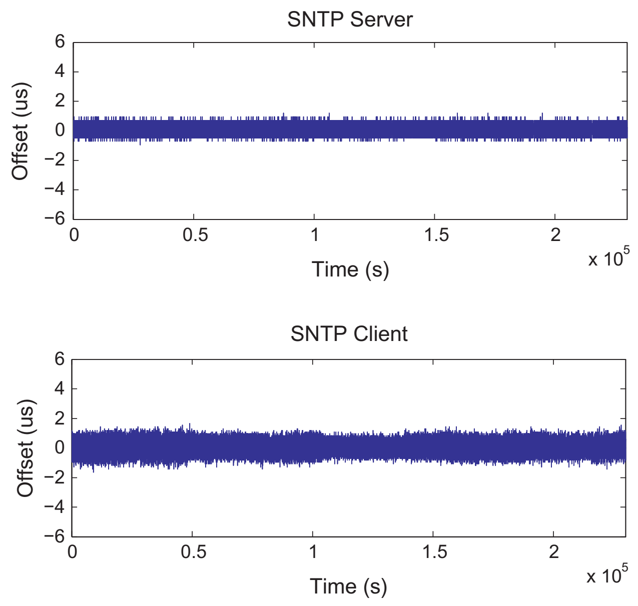


Fig. 8. Offset as a function of time for various days.

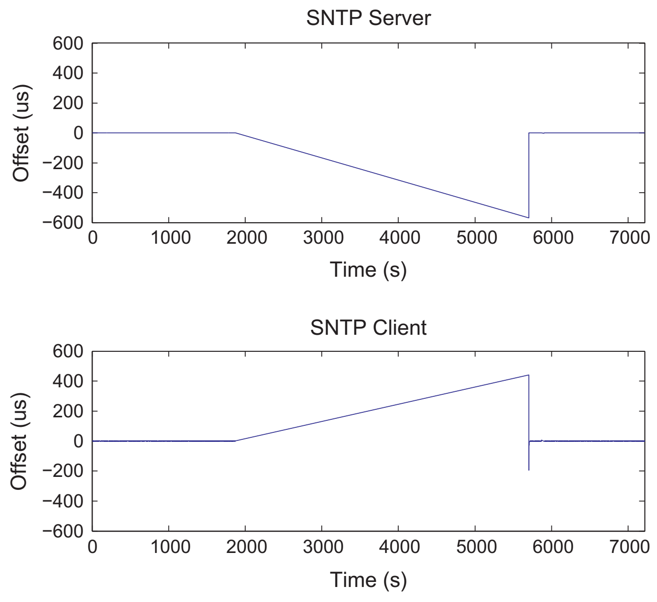


Fig. 9. Offset as a function of time when the system operates in holdover.

Figs. 4b and 5b show LUTs and FFs become solely dependent on the number of signals. As it can be observed from Fig. 6b, the LEA has the advantage that it does not store data in BRAM but transmits them via serial. Therefore the number of BRAMs used to verify the system does not depend on the number of signals or the number of samples, so the system can be tested indefinitely, only limited by external resources.

For the case under discussion, the SNTP client and server have been implemented on a Spartan-3E FPGA device (xc3s500e). These FPGAs have a total of 20 BRAMs and each block contains 18,432 bits of fast static RAM, with 16 Kbit allocated for data storage. For each design a total of 256 signals have been sampled: timestamps (least significant part), time offset, round trip time and local clock adjustment parameters. With this configuration, the LUT and FF usage is not critical, since the percentage of used LUTs and FFs to verify the system, in terms of number of samples, is 9% and 8%, respectively using ChipScope Pro, and 3.79% and 1.68% using the LEA. These results are obtained considering the worst case. However, as it can be observed in Fig. 7, when ChipScope Pro tool is used, there is a big usage of BRAMs even for a small number of samples. In this case, a maximum of 1024 samples can be captured with ChipScope Pro, which is insufficient for the type of system that is intended to be verified. In the case of the LEA, for the same scenario presented for ChipScope Pro, it is worth noting that only one BRAM is used (this memory stores the program that will be run by PicoBlaze), being this alternative feasible in any grade of FPGA chips.

Finally, the collected data, using the LEA and the developed software tool, have permitted the correct long-term verification of the synchronization platform. In this way, Fig. 8 shows the offset as a function of time for various days, where one sample per second has been transmitted by the verification core. As it can be seen from the figure, the system always remains synchronized, and no inappropriate behavior has been observed. Moreover, captured data have been used in order to measure the clock drift when the system operates in holdover. Fig. 9 shows the clock drift when the SNTP server loses the GPS reference and how the synchronization is recovered once the reference source is reestablished.

6. Conclusion

In this contribution, a verification core based on PicoBlaze for long-term on-chip verification is presented. The proposed solution allows developers to avoid the implementation of custom verification cores in many cases, greatly improving the design and verification time.

The verification core has been successfully applied to the long term verification of a client–server synchronization system, and compared to ChipScope Pro logic analyzer. The results show that the ChipScope Pro tool is not suitable to verify this type of systems because this would need excessive internal resources to store the captured data even for a small number of samples to be acquired. The proposed core does not store data using internal resources but transmits them via serial port so the system can be verified indefinitely, only limited by external computer storage.

Acknowledgment

This work has been partially supported by the Ministry of Education and Science of the Spanish Government through the TEC2007-61802/MIC (HIPER) project, and the European Regional Development Fund (ERDF).

References

[1] B. Hutchings, B. Nelson, Unifying simulation and execution in a design environment for FPGA systems, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 9 (2001) 201–205.
[2] T. Wheeler, P. Graham, B. Nelson, B. Hutchings, Using design-level scan to improve design observability and controllability for functional verification of FPGAs, in: 2001 Proceedings International Conference on Field-Programmable Logic and Applications (FPL), Belfast, Northern Ireland (UK), pp. 483–492.
[3] P.S. Graham, Logical Hardware Debuggers for FPGA-Based Systems, PhD Dissertation, Bringham Young University, Department of Electrical and Computer Engineering, 2001.
[4] N. Ohba, K. Takano, Hardware debugging method based on signal transitions and transactions, in: IEEE Proceedings of Asia and South Pacific Conference on Design Automation, Yokohama (Japan), pp. 454–459.
[5] Xilinx, ChipScope Pro 11.1 Software and Cores User Guide, Xilinx, Inc., 2009.
[6] K. Arshak, E. Jafer, C. Ibala, Testing FPGA based digital system using XILINX ChipScopeTM logic analyzer, in: 29th International Spring Seminar on Electronics Technology (ISSE), St. Marienthal (Germany), pp. 355–360.
[7] L. Ehrenpreis, P. Ellervee, K. Tammemae, Open source on-chip logic analyzer for FPGAs, in: 2006 International Baltic Electronics Conference, Tallinn (Estonia), pp. 1–4.
[8] J. Viejo, J. Juan, M.J. Bellido, E. Ostua, A. Millan, P. Ruiz-de Clavijo, A. Muñoz, D. Guerrero, Design and implementation of a SNTP client on FPGA, in: Proceedings 2008 IEEE International Symposium on Industrial Electronics (ISIE), Cambridge (United Kingdom), pp. 1971–1975.
[9] T. Lee, Y. Fan, S. Yen, C. Tsai, R. Hsiao, An integrated functional verification tool for FPGA systems, in: International Conference on Innovative Computing, Information and Control, Kumamoto (Japan), p. 203.
[10] G. Knittel, S. Mayer, C. Rothlaender, Integrating logic analyzer functionality into VHDL designs, in: 2008 International Conference on Reconfigurable Computing and FPGAs, Cancun (Mexico), pp. 127–132.
[11] K. Chapman, PicoBlaze 8-Bit Embedded Microcontroller User Guide for Spartan-3, Virtex-II and Virtex-II PRO FPGAs, Xilinx, Inc., 2005.
[12] Lattice, LatticeMico8 ProcessorReference Manual, Lattice Corp., 2011.
[13] D.L. Mills, Network Time Protocol (Version 3) Specification, Implementation and Analysis, RFC 1305 (Draft Standard), 1992.