Veritrace: A Tool To Generate Trace Buffers For Post-Silicon Debug

Danilo D. Almeida
Instituto de Ciências Exatas e Tecnológicas,Campus UFV-Florestal
Florestal,Brasil
danilo.damiao@ufv.br

Fredy A. M. Alves
Instituto de Ciências Exatas e Tecnológicas,Campus UFV-Florestal
Florestal,Brasil
fredy.alves@ufv.br

José Augusto M. Nacif
Instituto de Ciências Exatas e Tecnológicas,Campus UFV-Florestal
Florestal,Brasil
jnacif@ufv.br

***Abstract*—In the semiconductor industry, the integrated circuit verification process complexity is proportional to the circuit complexity. For complex designs, pre-silicion verification techniques cannot detect all errors. Post-silicon techniques use specific modules connected to the design under debug to detect functional and structural failures by monitoring the circuit internal signals while executing real applications, under real clock speed. The trace-buffer technique stores signals over time from the design under debug. This data is extracted and treated by an external software. The objective of this paper is to present a tool to help in the post-silicon debug process by automating the generation of trace-buffers using the Verilog hardware description language. We also analyze trace-buffer area overhead in different complexity scenarios.**

I. Introduction

The verification process in the semiconductor industry is the longest phase in the development cycle of an integrated circuit. This stage is responsible for 35% of the total development time [1]. The use of techniques capable of detecting and informing when an error occurs in an acceptable time limit is a very important factor for a semiconductor industry to stay on the market [4].

The verification of an integrated circuit is divided into two phases: pre and post-silicon. In the pre-silicon verification the hardware the developer has a large coverage of the internal circuit behavior using techniques based on simulation, formal verification, and emulation. Unfortunately, pre-silicon verification is too slow when compared to the circuit running at real clock speed [3].

Post-silicon verification is performed after the circuit manufacturing process. The objective is to detect design functional and/or structural failures not identified in the pre-silicon and manufacturing test phases. This verification phase uses a design for debug (DFD) architecture connected to the circuit under debug (CUD), allowing the detection and registration of error scenarios by storing internal signals values while executing real applications at full clock speed.

The circuit area overhead caused by the DFD grows with the number of internal signals being monitored. The use of hybrid techniques presents better results for in-chip observability. An example of this scenario is the use of the scan-chains with the trace-buffer technique. The Scan-chain method makes possible to observe a big set of internal circuit signals. The main drawback of this technique is the need to halt the system in order to dump the scan values [7].

The trace-buffer method stores data from a limited number of internal signals during in-system debug in order to monitor their behavior over time at real clock speed [9]. It is possible to extract and treat this data through an external software.

In this paper we preset Veritrace (Verilog Trace Buffer), a tool to generate trace-buffer modules in Verilog hardware description language. The generated modules are ready to be integrated in the design under debug in order to automate the post-silicon verification process. This paper is organized as follows. The post-silicon architecture and the trace buffer module are presented in Section II. Related work are presented in Section III. The Veritrace architecture is presented in section IV. We present results in Section V. We finally conclude and discuss future work in Section VI.

II. Background

In this Section we present a general overview of the Design for Debug used for the post-silicon validation and its internal components. Moreover, we present a more detailed explanation about how the trace-buffer works.

1) Design for Debug(DFD): The DFD is a module used for the post-silicon debug embedded to the design under debug (DUD). It is responsible for capturing groups of signals from the DUD and storing those signals in real-time at full clock speed. The main limitation in the post-silicon verification phase is the restricted observability of the circuit internal signals behavior. This occurs due to the area overhead in the integrated circuit for the DFD. Figure 1 shows the DUD and the DFD embedded to it. The DFD is divided into three parts: Interconnection Network, Trigger Logic and Trace Buffer. The

interconnection Network is responsible for tapping a group of signals from the DUD and connecting a subset of those signals to the trace buffer. The Trigger Logic is responsible for detecting an error in the DUD and routing the Interconnection Network to connect the group of signals to the trace buffer related to the error. The trace buffer is a module responsible for storing the signal samples received from the interconnection network, it is limited by two parameters: the buffer width and depth. The first one limit the number of signals which can be stored at each clock cycle, the second one limits the number of times a signal sample can be stored [7]. The multiplication of those parameters define the buffer size.

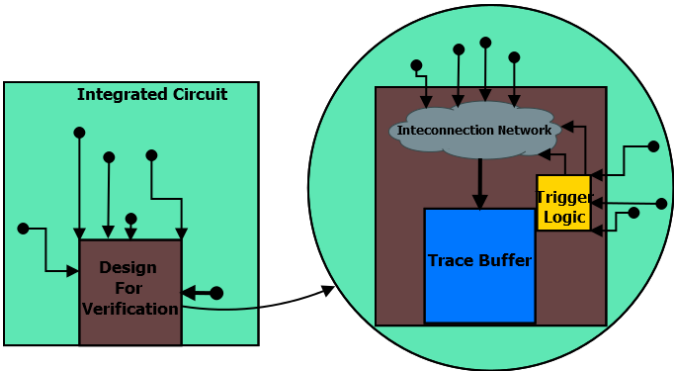

Fig. 1. Internal structure of a DFD module

2) Trace-Buffer Module: The trace buffer has two states: the write and the read state. The initial state is the write state. Figure **??** shows the write pointer (WP) and read pointer (RP) for one signal connected to the trace buffer. At every clock cycle, a signal sample is stored on the memory position indicated by the WP and it moves to the next position. When the WP reaches the buffer depth and a sample is received, it returns to the first position and the buffer changes to a read state. On this state, all of the memory positions for that signal have to be consumed. At each read, the RP moves one memory position ahead and when it reaches the buffer depth and one read happens, it returns to the beginning and the buffer goes back to the write state, closing the buffer write/read cyclic behavior. On Figure 2, the coloured squares represent stored signal samples and the white ones, consumed or free memory positions.

When the buffer is full, the trace-buffer send a signal to the serialize module to start the read process. The data read from trace buffer is transferred from the buffer to a computer using the RS-232 protocol where a software could be able to receive an treat this data. After the read cycle is complete, the trace buffer restart all pointers to the initial position and it return to the write state to restart the signal sample capture process. Figure 3 represents the communication between the trace-buffer module and the PC through the Serialize module.

III. Related Work

A hybrid memory that works like a data-cache reducing the final overhead generated by the trace-buffer is proposed

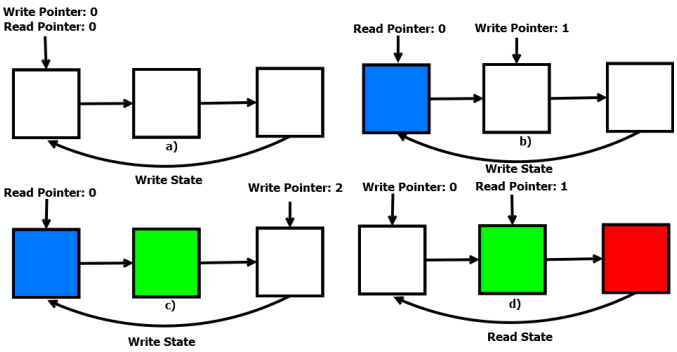

Fig. 2. Example of a FIFO buffer:
a) Memory in Initial write state.
b) Memory after one write cycle.
c) Memory after two write cycles.
d) Memory after three write cycles
and one read cycle, transition from write to read state.

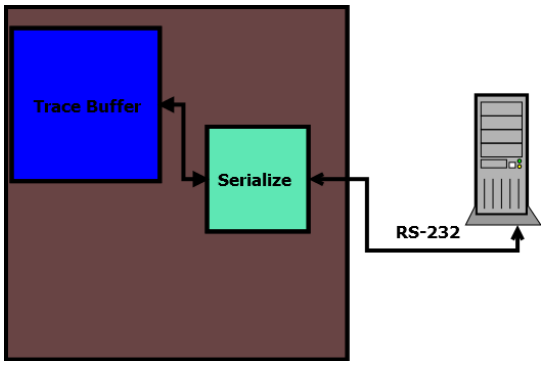

Fig. 3. The block diagram of the communication module

in [6]. The use of a hybrid method using scan-chains in order to expand the circuit observability and trace-buffers to store this signals to avoid chip halting in the execution process is proposed in [5]. A study of selective signal capture in suspect clock cycles with the objective of expanding the observation window is proposed in [9].

The results are given in terms of a hybrid data cache miss rate, the data acquired during the post-silicon debug process and the expansion in the observation window. In this work, we propose to present the results in terms of area overhead, in order to evaluate the trade-off between the quantity of monitored signals and the extra area cost caused by the DFD.

IV. Veritrace Structure

The software was developed in the C Language, it supports any size of trace buffer. This tool provides for the hardware developer a way to accelerate and automatize the process of choosing the best size of trace buffer for its design.

A. Trace-Buffer Generate Process

The process of generating a trace-buffer module requires two arguments: the buffer width and size. The first argument consist in the number of traced signals from

the interconnection network which connects to the trace buffer. The second one is the number of traced signals samples stored after many clock cycles. Veritrace generates all design files used by the trace buffer in the verilog hardware description language and reports the total size of the memory in bytes. Figure 4 presents the trace-buffer generation process using Veritrace.

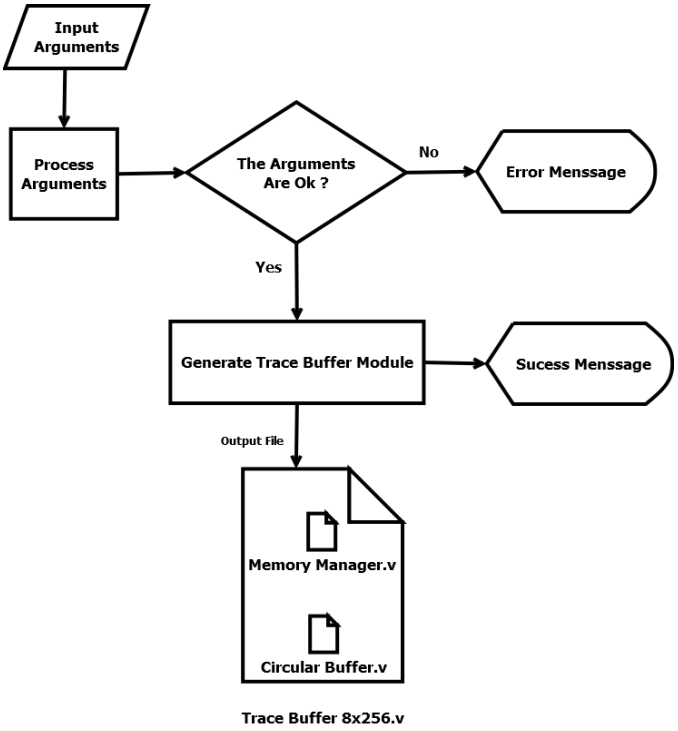

Fig. 4. Fluxogram of a trace-buffer 8x256 generation

B. Trace-Buffer Interface

The trace-buffer Verilog interface consists in a synchronizer input signal named **clk**, a input signal to set all pointers to the buffer initial position named **reset**, a data input signal to receive the data from the interconnection network named **datain**, a data output signal to send the stored data to the serialize module named **dataout** and a signal to initialize the transfer of all stored data named **pause**. Figure 5 presents a simulation of a trace-buffer module of 8 width and 8 depth.

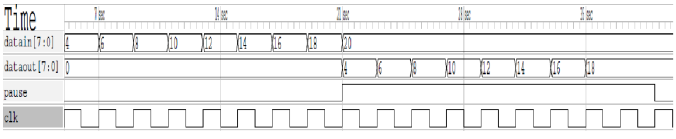

Fig. 5. The buffer write and read in the post-edge of **clk**. When the trace buffer is full, all data stored is transferred using the **dataout** pin by setting **pause**.

V. Results

We conducted the experiments by generating four trace buffers with fixed width 25 and with depths 8, 16, 64 and 128. We inserted these trace buffers into a open source Verilog project. MIPS 32 Release 1 [2]. The synthesis has been made using the free synthesis tool Yosys [8] and the area for the designs is presented in number of ports. Table 1 presents the number of ports for different trace buffer sizes separated from MIPS.

TABLE I
Necessary ports for build a trace buffer

Width	Depth	Ports
32	8	898
32	16	1704
32	64	6413
32	128	12652
6	32	744
16	128	6489

To calculate the area overhead caused by the DFD, we developed 4 different designs, each one has a mux-tree interconnection network which outputs 25 signals connected to a trace buffer, this DFD is embedded on the MIPS 32, each design has a buffer with different depths, those being 8, 16, 64 and 128. The interconnection network used in the synthesis presents a fixed separated value of 726 ports. The trace buffers utilized in this experiment present a fixed width equal to the interconnection network output. Figure 6 presents the number of used ports by the MIPS 32 design without the DFD and with the interconnection network and the trace buffer with different depths.

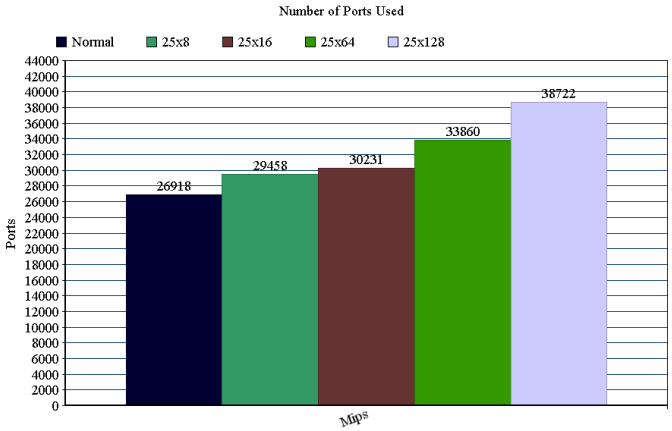

Fig. 6. Number of used ports with and without DFD

Table II presents the overhead when the DFD with different buffer sizes is embedded to the MIPS project.

The overhead in the MIPS project had an almost linear growth, the design with the 25x8 buffer size presents the lowest overhead with 9,4%. We cause that depths above 128 cause a final overhead greater than 50%, which may be a too

TABLE II
OVERHEAD IN THE ORIGINAL PROJECT

Project/Size	25x8	25x16	25x64	25x128
Mips 32	9.4%	12.3%	25.78%	43.85%

high cost for the circuit design.

This result demonstrates that the choice of the trace buffer size impact significantly on the final project cost. It is important to be able to study the tradeoff of different sizes of buffers, the trade off will be acceptable or not depending on the need for observability.

VI. CONCLUSION AND FUTURE WORK

We have presented a tool to assist hardware developers on the post-silicon verification phase by automating the process of trace buffer generation of any size. Our results show the importance of being able to try different sizes of buffers to find the optimal one. For future work we intent to add the serialize module in the final trace-buffer structure, with this module, the extraction and treatment of failures occurred in the process of post-silicon verification becomes possible with an external software.

ACKNOWLEDGEMENT

We would like to thank CAPES, CNPq, FAPEMIG and UFV for the financial support.

REFERENCES

[1] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller. A reconfigurable design-for-debug infrastructure for socs. In *Design Automation Conference, 2006 43rd ACM/IEEE*, pages 7–12, 2006.

[2] G. Ayers. Mips32 release 1, 2014.

[3] A. Gomes, F. Alves, R. Ferreira, and J. Augusto M.Nacif. Vericonn: a tool to generate efficient interconnection networks for post-silicon debug. In *Test Symposium (LATS), 2015 16th Latin-American*, pages 1–6, March 2015.

[4] Y.-C. Hsu, F. Tsai, W. Jong, and Y.-T. Chang. Visibility enhancement for silicon debug. In *Proceedings of the 43rd annual Design Automation Conference*, pages 13–18. ACM, 2006.

[5] H. Ko and N. Nicolici. Combining scan and trace buffers for enhancing real-time observability in post-silicon debugging. In *Test Symposium (ETS), 2010 15th IEEE European*, pages 62–67, May 2010.

[6] C.-H. Lai, Y.-C. Yang, and I.-J. Huang. A versatile data cache for trace buffer support. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 61(11):3145–3154, Nov 2014.

[7] S. Prabhakar and M. Hsiao. Using non-trivial logic implications for trace buffer-based silicon debug. In *Asian Test Symposium, 2009. ATS '09.*, pages 131–136, Nov 2009.

[8] C. Wolf. Yosys open synthesis suite, 2014.

[9] J.-S. Yang and N. Touba. Expanding trace buffer observation window for in-system silicon debug through selective capture. In *VLSI Test Symposium, 2008. VTS 2008. 26th IEEE*, pages 345–351, April 2008.