ISSN 2349-7815

International Journal of Recent Research in Electrical and Electronics Engineering (IJRREEE)
Vol. 2, Issue 2, pp: (173-180), Month: April 2015 - June 2015, Available at: www.paperpublications.org

FPGA Debug Using Incremental Trace Buffer

[1]T.Divya, [2]P.Rajamurthy

[1]M.E Applied Electronics, Final Year, Department of ECE, Jaya Engineering College, Tamilnadu, India
[2]Assistant Professor, Department of ECE, Jaya Engineering College, Tamilnadu, India

***Abstract*: Design verification is an essential step in the development of any product. It ensures that the product as designed is the same as the product as intended. Software simulation is the common approach for validating hardware design unfortunately, it will take hours together to execute. Difficulties in validation arise due to the complexity of the design and also due to the lack of on chip observability. One common solution to this problem is to instrument the prototype using trace-buffers to record a subset of internal signals into on-chip memory for subsequent analysis. In the proposed system, an example circuit is implemented to perform the tracing operation and various trace buffers are designed to record the different stages of internal signal states. The resulting signal states are to be stored, like a error outputs. Low power methodologies are also implemented to achieve low power consumption. Thus the errors are separately stored in the memory for analyzing the signals. This might be used for changes in the logic wherever needed. Thus this tracing is performed to monitor signal states of an FPGA.**

***Keywords:* Design verification, field-programmable gate-array (FPGA)debug, incremental compilation, trace-buffer.**

I. INTRODUCTION

As the capacities of Field-Programmable Gate Arrays (FPGAs) grow, ensuring that a design is functionally correct (verification and validation) and finding the sources of any observed incorrect behavior (debugging) has become increasingly difficult. This is due both to increasing device capacity (and the corresponding increase in system complexity) as well as limited on-chip observability. Support for effective debugging has been identified as a key technological requirement as device densities increase.

Verification and debugging both make extensive use of software simulators. Simulation provides full-chip visibility (any signal can be examined in simulation) and fast turnarounds between design changes. However, the simulation of large designs can be extremely slow. [1]As an extreme example, Intel reported that software simulations of their Core i7 chip ran one billion times slower than on real silicon, with the sum of all their simulation efforts on a large server farm culminating in no more than a few minutes of actual chip operation [2]. Clearly, the simulation of common tests (such as booting an operating system) is not practical. Because of this, designers regularly rely on hardware validation to complete the verification and debugging of their designs. By mapping designs to an actual FPGA, the design can be run at-speed (or close to at-speed) meaning much deeper traces are possible.

In addition, testing the FPGA in-situ may allow realistic input stimuli, since the device can be connected to the other chips in the target system. A primary challenge with hardware validation is the limited observability of the signals on-chip. Only signals connected to output pins can be directly observed, which may make it difficult to understand the behaviour of a system and isolate a design error. Some FPGAs allow a “snapshot” of all state bits to be taken, which can later be read-out using a JTAG interface, however, this does not easily allow for tracing signal over time.

To enhance observability, tools such as signaltap II ,ChipScope , and Certus instrument a user design by instantiating *trace-buffers* on the chip[1]. These trace-buffers are memories that record the activities of selected signals over a number of cycles. By carefully selecting which signals are recorded, on-chip observability can be enhanced, simplifying the debugging task. In addition to trace buffers, these tools instantiate connections (or networks) that connects the traced signals to these buffers. In most cases, after these tools add debug instrumentation, the user design must be recompiled from scratch. This has a number of drawbacks: (1) recompilation can be slow, especially in prototyping systems consisting of multiple FPGAs, (2) a recompilation may cause timing differences which may obscure (or even eliminate) the bug that was being sought, and (3) additional routing stress may cause a previously routable design to become unroutable. Rather than recompiling an entire design, it is sometimes possible to leave the user design mapping unmodified, and incrementally add connections between the trace-buffers and the signals that the user wishes to observe.

These additional connections could be created using routing tracks that were not used during the original circuit mapping. If this is possible, a full recompilation is not necessary, leading to significantly faster debug cycles and increased debug effectiveness. We refer to this technique as incremental-tracing. Intuitively, it should often be possible to make such connections. FPGA vendors typically over-provision their routing architecture, so there should often be sufficient unused routing tracks. However, for some parts of the design in which congestion is high, or if there are a large number of signals that need to be observed, this incremental-tracing may not be possible for all signals.

II. BACKGROUND

ICs have revolutionized the world of electronics. It is critical that IC's function correctly or there can be costly or even deadly consequences. Observability is key to verifying behavior and tracking down the cause of bugs. Simulators can provide full observability into all signals of a circuit .[1]Eddie Hung, Steven J. E. Wilton (2012) this author says as the capacities of FPGA increases its verification and validation is difficult so, on chip observability is used in order to enhance on-chip observability, but doing so often requires re-compiling the entire design for each new trace configuration. To explore the limitations of incremental-synthesis for trace-buffer insertion, and to propose CAD optimizations exclusive to this application for improving runtime and routability is presented. During incremental tracing, there exists far more flexibility than with functional design changes-a traced signal is not constrained to reaching one particular sink.[2] Ho Fai Ko, Nicola Nicolici(2009) this author says silicon debug can be divided into two main steps data acquisition and analysis. An accelerated algorithm for restoring circuit state elements from the traces collected during a debug session, by exploiting bitwise parallelism is presented. New metrics that guide the automated selection of trace signals, which can enhance the real-time observability during in-system debug, was also introduced. To locate & correct design errors that escape pre-silicon verification here, the accelerated algorithms for restoring circuit state elements. state restoration, the algorithm only needs to check if data can be reconstructed at a circuit node and no branching & backtracking will be done if unsuccessful, undefined values will be concluded.[4] S Raman, c.l liu (2011) these author says a timing-constrained routing algorithm for symmetrical FPGAs which embodies a novel incremental routing strategy Experimental results confirm that the algorithm reduces delay along the longest path in the circuit, uses routing resources efficiently, and requires low CPU time . [3] Min Li and Azadeh Davoodi (2013) these authors says that the post silicon debug is the last step of debug process for faster and accurate debug an hybrid approach is employed. Here, the State Restoration Ratio(SRR) is used to measure the quality of a set of selected trace signal. Metric-based algorithms utilize metrics which allow approximating the ability of a candidate trace signal to restore the untraced state elements while taking into account the restoration that can be made from a subset of already-selected trace signals.

Thus it's concluded that the objective is to eliminate the recompilation so that the turnaround time will be high therefore the design accuracy is increased by using trace buffer and incremental technique.

III. SYSTEM ANALYSIS

A. Trace buffer:

Trace buffers are formed from a memory resource on the FPGA. Trace buffers record a limited-size history of the signals connected to them during regular device operation. Designers verify functionality or hunt bugs by properly adjusting the trigger conditions and examining trace buffer data. multiple trace buffers are distributed to observe and record nearby user signals. Distributing the trace buffers reduces the distance signals must be routed and improves circuit timing. Including more trace buffers will allow more signals to be observed. A centralized trigger unit controls the operation of all the trace buffers with a single output signal that halts their recording when necessary. The trigger unit requires a region of logic to

ISSN 2349-7815

International Journal of Recent Research in Electrical and Electronics Engineering (IJRREEE)
Vol. 2, Issue 2, pp: (173-180), Month: April 2015 - June 2015, Available at: www.paperpublications.org

detect conditions. Incremental distributed trigger insertion, we distribute the logic elements that make up the trigger function across logic elements that are not used by the user circuit. These logic elements may not be contiguous. Intuitively, this will allow the trigger logic to take better advantage of any left-over space after mapping the user circuit, adapting to changes in the size or make-up of the trigger function. After placing the logic cells in unused locations, the logic cells must be connected to each other and to the user circuit using incremental routing techniques. Sample a subset of signals into on-chip memories.

Capturing a sequence of states, at full speed. It does not cost and extra silicon area occupied. FPGAs commonly not filled to capacity. Functional definition errors can be doubly difficult to find since the designer has misunderstood a particular requirement, so the error can be overlooked even when looking carefully at the details of the design. An example of a common functional definition error would be where a state machine transition doesn't end up in the right state. Errors can also show up in system interfaces as an interaction problem. Interface latency, for example, might be incorrectly specified resulting in an unexpected buffer overflow or underflow condition. System level timing issues are another very common source of design errors. Asynchronous events, in particular, are a common source of errors when synchronization or crossing timing domain effects are not carefully considered. When operating at speed these types of errors can be very problematic and may show up very infrequently, perhaps only when specific data patterns manifest themselves. Many common timing violations fall into this category and are usually very difficult, if not impossible to simulate.

The concept of the embedded logic analyzer was a direct result of the ad-hoc in-circuit debugging capabilities that designers implemented when FPGAs were first used. [2]Embedded logic analyzers added new capabilities and eliminated the requirement for the designer to develop their own analyzer. Most FPGAs offer these capabilities and third parties offer standard analyzers that can easily interface with higher level tools to further improve productivity.[2] The logic analyzer functionality is inserted into the design, using FPGA fabric and embedded memory blocks as trace buffers. Triggering resources are also created so that complex signal interactions can easily be selected and captured. In particular, if trace buffers are used this will reduce the number of block memories available If a wide buffer is needed this will also be a trade-off against memory depth (since the use of a wider memory results in shallower memory depth) a big disadvantage when using smaller devices. Perhaps the largest drawback to this technique is that every time an adjustment to probe placement is made, it is necessary to recompile and reprogram the design. When using a large device this process can take a significant amount of time. Due to the way the signal probes are placed in the design it can be difficult to correlate signal timing relationships.

Additionally, the delays between signal probes are not consistent and thus timing relationships are difficult to compare. This is a particular difficulty when comparing asynchronous signals or signals from different time domains. The use of in-circuit debug code in conjunction with external test equipment was a natural development when an external logic analyzer [2] was already available for system testing. By creating some simple debug code to identify and select internal test signals and apply them to FPGA I/Os, it was possible to leverage the analyzers advanced capabilities (such as large trace buffers, complex triggering sequences, and multiple viewing options) to create simple yet powerful debug environments.

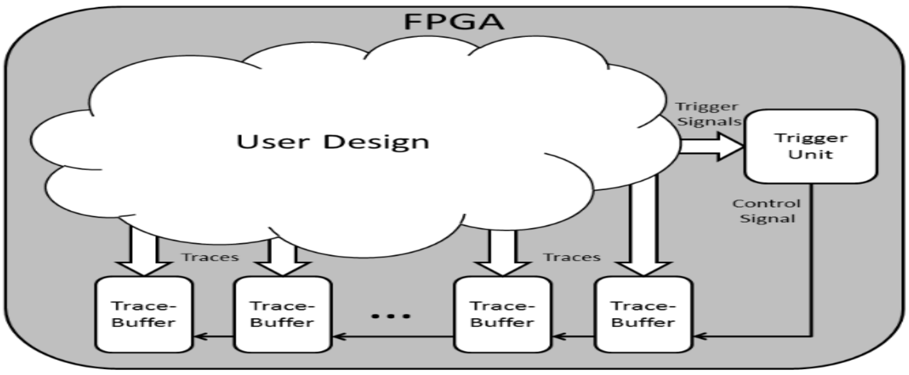


Fig 1 Block diagram with many trace buffers

The fig 1 illustrates our proposed approach. In our approach, multiple trace buffers are distributed to observe and record nearby user signals. Distributing the trace buffers reduces the distance signals must be routed and improves circuit timing.

Including more trace buffers will allow more signals to be observed. A centralized trigger unit controls the operation of all the trace buffers with a single output signal that halts their recording when necessary.

The trigger unit requires a region of logic to detect conditions. We try to find a large enough region of logic as close to the center of the design as possible to improving timing performance. However, for the trigger unit size we tested, the trigger unit typically had to be placed on an edge of the user design to find enough unused logic resources. The resources that the user design requires are known because this entire system is inserted incrementally after the user design has already been placed and routed.

B. Incremental trace insertion:

To increase the observability of FPGA circuits, we propose trace buffers and a trigger unit be inserted incrementally in an already placed-and-routed design. We shall refer to the already placed-and-routed design as the original circuit or user circuit and the trace buffers and trigger that are incrementally inserted as the debug system. Nets from the original circuit shall be incrementally connected to the trace buffers to be observed and recorded. The trigger unit shall control the trace buffers and allow them to record until trigger conditions specified by the designer are met.

We propose incrementally inserting the debug system because it will reduce the impact on the original circuit's area, placement, routing, and timing. From the perspective of the original circuit, the debug system has no area overhead. The original circuit will already be placed-and-routed and thus will already have claimed whatever area of the FPGA it needs. The debug system is inserted into whatever FPGA area has been left unused by the original circuit.

Thus, the debug system has an area but it is area that is of no consequence to the original circuit because it did not need it. If the debug system were not incrementally inserted then it would increase the area of the original circuit. Incremental insertion allows us to adjust the size of the debug system to fit into whatever the original circuit does not use.

The amount of trace buffers and trigger logic we can insert will be influenced by the area of the original circuit. If the original circuit used a lot of BRAMs then there will be less available for trace buffers. Likewise, if the original circuit utilizes a high percentage of slices then we may be limited in the type of trigger unit we can insert. The debug system does not influence the area of the original circuit but the reverse is not true; the original circuit influences the area of the debug system.

The fig 2 shows how the time taken for the execution as been reduced after using incremental technique. The goal of incremental synthesis is generally to modify the functionality of an existing circuit with minimal changes to its current placement and routing. Our goal is different than this "general-purpose incremental synthesis" because we only desire to observe signals. General purpose incremental synthesis does not guarantee that the placement and routing of the original circuit will be preserved. However, we do make that guarantee for our method.

The placement and routing of the trace buffers and trigger unit is restricted to resources unused by the existing circuit. The original circuit will be left completely intact. When the debug system is removed the circuit will be exactly the same as it was originally. This allows designers to instrument a circuit without influencing the placing or routing of the circuit.

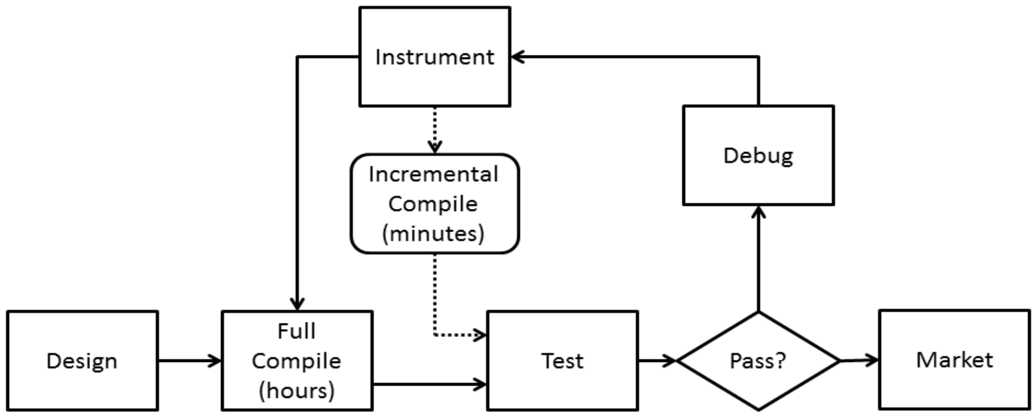


Fig 2 Design and debug flow demonstrating how incremental compile reduces the time

Restricting our method to unused FPGA resources also reduces impact on timing. All the paths and timing of the original circuit are preserved. However, the minimum period may temporarily change while the debug circuitry is included. The minimum period of the circuit will change if any of the paths we add have delays greater than those of the critical paths in the original circuit. The paths we added would then become the critical paths and the minimum period would increase.

The chance of these occurring increases as the minimum period of the original circuit decreases. This change in the minimum period would only apply while the debug system is included in the circuit. After the debug system is removed the circuit's minimum period will return to its original value if inserting the system caused a change.

C. Block diagram and its description:

From the figure 3.4 is the block diagram of the proposed system, from the clock generator, the input clock pulse gets generated. These clock pulses are then divided using a clock divider. The input data is given to circuit and this produces the corresponding output. This circuit's internal signal states are stored using a trace buffer. Various trace buffers are used for storing the various signal states. If any error occurs, it will be stored in a separate memory and monitored. The power supply is the input supply given to the hardware. The system clock is the basic clock pulse given to the hardware and the hardware reset is the initial condition for resetting of the hardware.

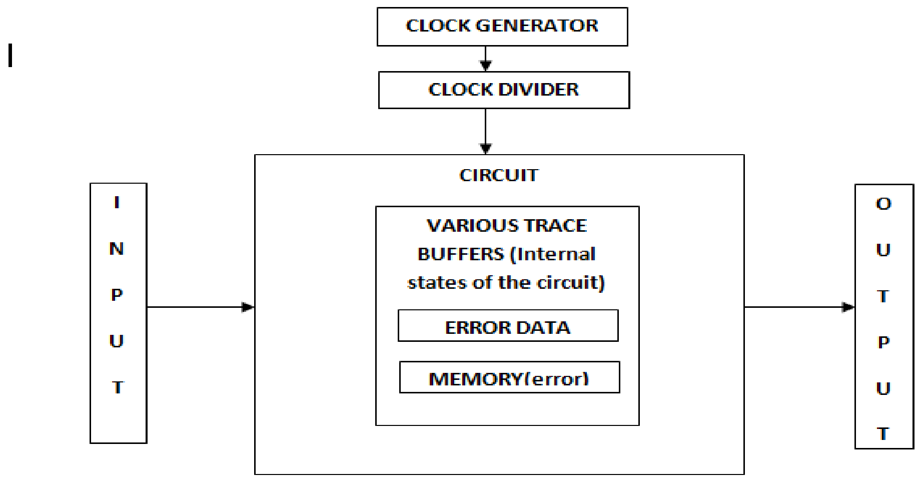


Fig 3 Block diagram

D. Circuit diagram:

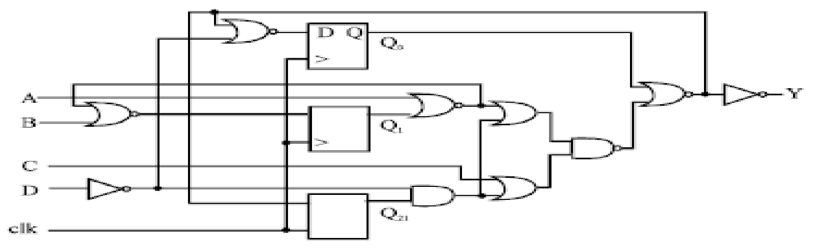


Fig 4 Circuit diagram

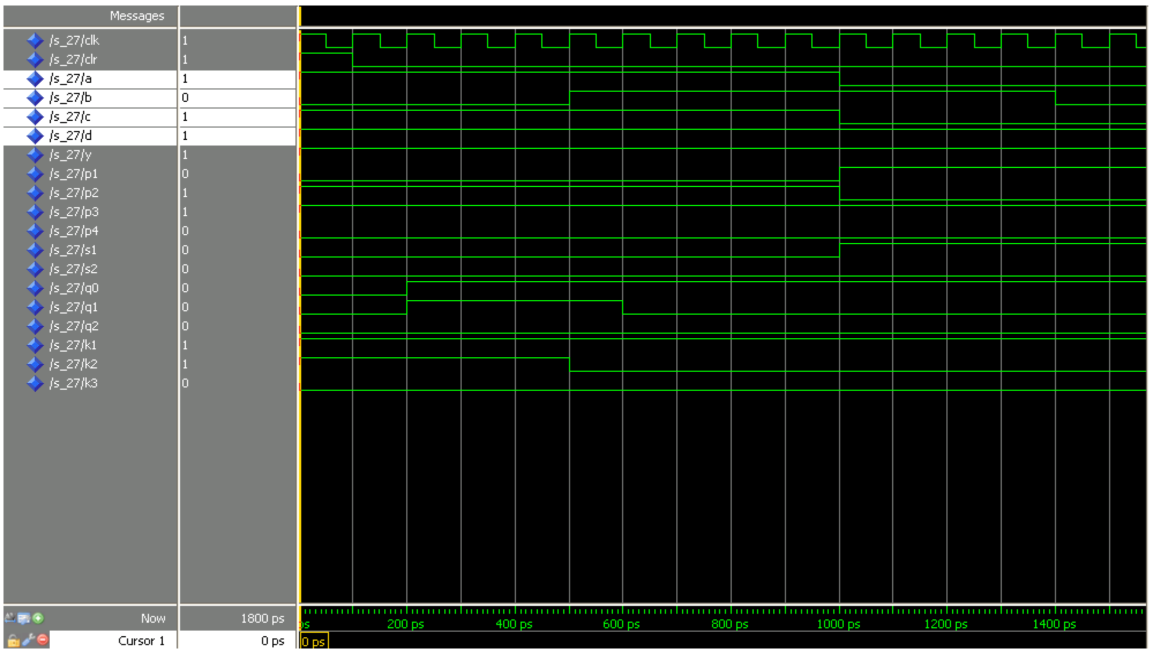
The fig 4 is the circuit diagram that is used in this project; here the circuit used is sequential circuit. Sequential logic is a type of logic circuit whose output depends not only on the present value of its input signals but on the past history of its inputs. This is in contrast to combinational logic, whose output is a function of only the present input. That is, sequential logic has state (memory) while combinational logic does not. Or, in other words, sequential logic is combinational logic with memory.

Sequential logic is used to construct finite state machines, a basic building block in all digital circuitry, as well as memory circuits and other devices. Virtually all circuits in practical digital devices are a mixture of combinational and sequential logic. Digital sequential logic circuits are divided into synchronous and asynchronous types. In synchronous sequential circuits, the state of the device changes only at discrete times in response to a clock signal. In asynchronous circuits the state of the device can change at any time in response to changing inputs. The combinational circuit does not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit. But sequential circuit has memory so output can vary based on input. This type of circuits uses previous input, output, clock and a memory element.

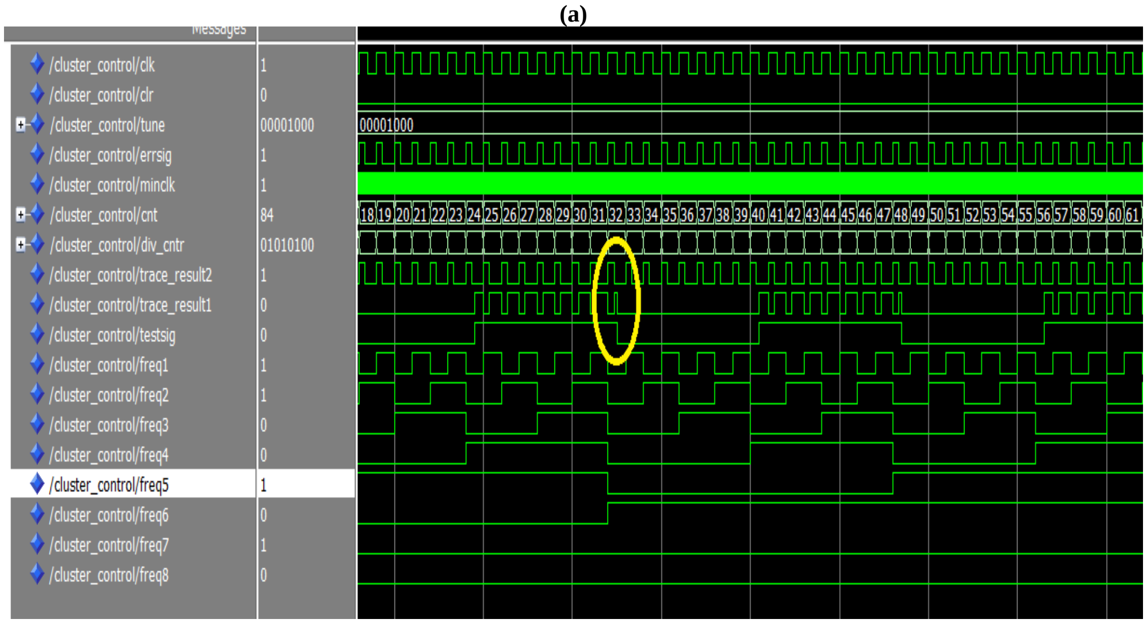
In the fig 4, the trace buffers are implemented which caches the errors when the logic produces the error. The error may be of frequency, timing. These are all found out and we are clearing that error. The trace buffers are also designed which acts like a memory.

IV. SIMULATION RESULTS

The fig 5(a) is the simulation output of a sequential circuit We are analyzing this circuit's internal signal states. we are designing a circuit and debugging it and locating the errors by means of signal tracing. This is performed by implementing trace buffers. The signal tracing concept is applied here to analyze the internal signals.

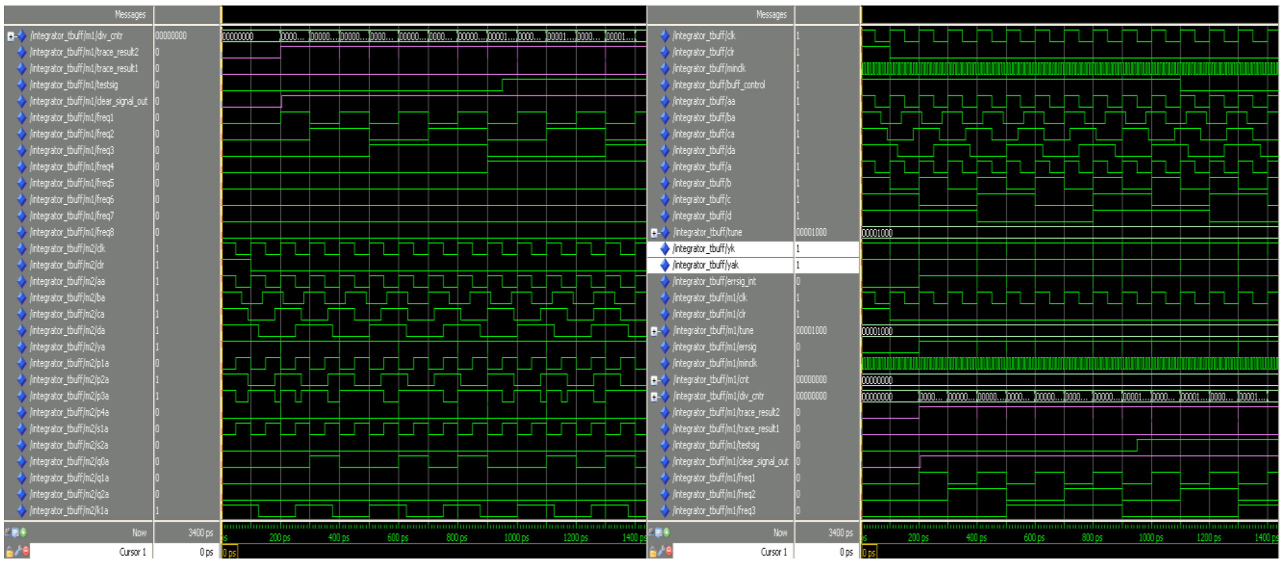


(a)

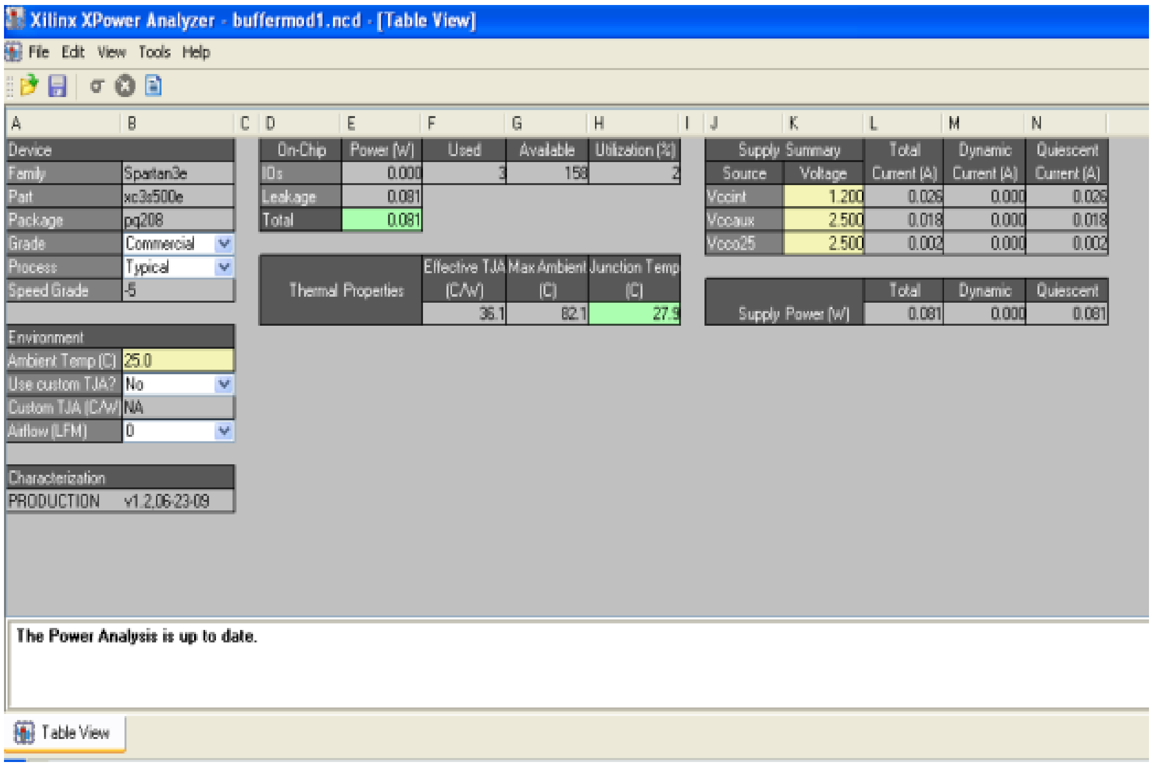


(b)

In the fig 5(b)is the simulation output of with the occurrences of glitches, the trace buffers are used for storing the internal signals during debug. Because we can only view the output in normal case, if any error occurs we are unable to locate the errors. So, trace buffers are also used for locating errors. These trace buffers are designed such that, it stores the signals in an incremental way for easy compilation.The analysis of signals and errors is done in case if any error occurs it will be stored in a separate memory. This memory stores the corresponding errors in the circuit. The trace buffers are in an incremental logic, so that we can easily identify the errors. Also the outputs of the circuit are viewed using the trace buffers stage by stage. Thus the errors and the signal are analysed. The fig 5(c)is the integration of all the sub modules. This analysis is done using Modelsim Simulator.



(c)



Xilinx XPower Analyzer - buffermod1.ncd - [Table View]

Device	
Family	Spartan3e
Part	xc3s500e
Package	pq208
Grade	Commercial
Process	Typical
Speed Grade	-5

Environment	
Ambient Temp (C)	25.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	0

Characterization	
PRODUCTION	v1.2,06-23-09

On-Chip	Power (W)	Used	Available	Utilization (%)
IOs	0.000	3	158	2
Leakage	0.081			
Total	0.081			

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	36.1	82.1	27.9

Supply Summary		Total	Dynamic	Quiescent
Source	Voltage	Current (A)	Current (A)	Current (A)
Vccint	1.200	0.026	0.000	0.026
Vccaux	2.500	0.018	0.000	0.018
Vcco25	2.500	0.002	0.000	0.002

	Total	Dynamic	Quiescent
Supply Power (W)	0.081	0.000	0.081

The Power Analysis is up to date.

(d)

Fig 5 (a) simulation result of the circuit (b) simulation result with the occurrences of gliches (c)The output of integration of sub modules (d) power report

V. CONCULSION

FPGAs are an increasingly being used for IC verification. They run faster than simulation and cost less than fabricating ASIC prototypes. However, the major disadvantage of performing verification on FPGAs is a lack of signal observability.

Here we have investigated a new incremental trace-based method for increasing the observability of FPGAs. The method incrementally inserts trace buffers and a trigger unit in an already placed-and-routed FPGA circuit. A unique characteristic of the method is the centralized trigger unit that controls all the distributed trace buffers. This simplifies incremental insertion and allows the method to easily scale to any number of trace buffers. Advantages of this incremental method include not affecting the placing and routing of the user's circuit, taking full advantage of leftover. Here, we had designed a circuit which plays the role of an FPGA. For this design we applied trace buffer which are also designed. These trace buffers trace the signal errors which occur due to some disturbances. Also errors which act according to the different frequency and time periods are all determined and stored in these trace buffers. Thus we are monitoring the internal signal states. All these designs are successfully designed and verified using Model sim Simulator.

REFERENCES

[1] E. Hung and S. J. E. Wilton, "Limitations of incremental signal tracing for FPGA debug," in Proc. Int. Conf. Field Program. Logic Appl.,Aug. 2012.

[2] H. F. Ko and N. Nicolici, "Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 28, no. 2,, Feb. 2009

[3] Min Li and Azadeh Davoodi, "A Hybrid Approach for Fast and Accurate Trace Signal Selection for Post-Silicon Debug"2013

[4] S. Raman, c.l liu (2011), "A Timing-Constrained Incremental Routing Algorithm for Symmetrical FPGAs Symmetrical FPGAs"

[5] S. Asaad, R. Bellofatto, B. Brezzo, C. Haymes, M. Kapur, B. Parker, T. Roewer, P. Saha, T. Takken, and J. Tierno, "A cycle-accurate, cyclereproducible multi-FPGA system for accelerating multi-core processor simulation," in Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays, 2012

[6] Xilinx. (2012, Dec.). ChipScope Pro 12.3, Software and Cores, User Guide, San Jose, CA, USA[Online].Available: http:// www.xilinx.com/support/document.