International Journal of Engineering & Technology, 7 (1.5) (2018) 7-12

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Integrated and effective in-system debugging approach in FPGA circuits

A Murali [1*], K Hari Kishore [2]

[1]Research Scholar, Department of ECE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur, Andhra Pradesh, India-522502.
[2]Department of ECE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur, Andhra Pradesh, India-522502
**Corresponding author E-mail: amurali3@gmail.com*

Abstract

This paper explores the new approaches for trace-based and In-system debugging of high level synthesis-generated hardware. The presented approach also exhibits the use of the Event Observability Ports (EOP) which provides the source level event observability in the hardware. To trace the events, the use of the independent trace buffers also known as Event Observability Buffers (EOB) is also discussed. The EOB has the signal that specifies the enabling the data storage and allows the storage decisions cycle-by-cycle which are to be made on the basis of EOB-by-EOB. The proposed approach also reveals the timing relationships of the captured events in the trace buffers. To recover the lost relationships among these events, we have introduced two methods in this paper. We also presented the demonstration and effectiveness of the trace strategy of an EOB.

1. Introduction

The use of the High-Level Synthesis (HLS) tools have gained the interest in the Industrial sectors which are also has the improved technology that became the popular in the academics [1]. These tools have shown the 5X boost of the design productivity [2]. In some cases, it is observed that the boost in productivity comes from the ability to verify the design specification in software and avoid time-consuming simulation runs [3]. The quality in the productivity can be lost if the user encounters a bug in their HLS design after the design is operating at full speeds. The productivity loss stems from the fact that the user is required to build an intimate understanding of the final circuit before the user can instrument the circuit for debugging. Since the current HLS tools lack an effective, integrated approach to in-system, source-level debugging. To meet the effectiveness and integration, debug the entire system from the source-level as it can be fixed with Embedded Logic Analyzers (ELA)s where the debugging process of the FPGA is more popular in the final stage of the processes.

ELAs are usually configured by selecting signals from the net-list to connect to the inputs of a trace buffer. These signals are then recorded on a cycle-by-cycle basis during the active execution of the design. The relevant information is captured by the on-chip memory allocation. These analyzers are used to perform the in-system debugging of HLS designs on FPGAs as mentioned earlier. The cycle-by-cycle capturing method has no flexibility for the use of high-level knowledge from HLS tools, in optimizing the trace buffer efficiency. We introduced a new method for the in-system debugging of FPGA designs specifically optimized for debugging HLS-generated FPGA circuits. It provides enough flexibility to allow a trace buffer configuration to be optimized in capturing as much relevant information as possible. The instrumentation process will utilize the high-level information available within the HLS tool to optimize the trace buffer configuration for memory efficiency and also the resource usage. This also allows fully automated and debugging for which the circuitry will be compiled into the design.

2. Reviewers point

The goal of the HLS tools is to take a sequential software input specification written in a high-level language and “generate efficient hardware. The sequential software specification is a series of control flow and program state update events. Debugging the software by analyzing the sequence of these events includes the determines the reason for the failure of the circuit. The design flow for a typical HLS compiler proceeds as follows.

At First, the sequential input specification is compiled to an intermediate representation (IR) by a standard front-end compiler. IR is then represented in an assembly language-like format and contains the necessary operations to completely implement the input specification in software. As in [4], the IR is in the form of a two-level control flow diagram (CFD) consisting of basic blocks (higher-level) and operations (lower level). Basic blocks are straight lines of code that terminate in a control flow decision operation (such as a

branch) that determines the next basic block that will be executed. Next, the IR is passed to a scheduler that assigns the operations from the CFD to the control steps of a state transition graph (STG). Once the scheduling is completed, the operations in the STG are bound to allocated functional units and registers. RTL code is then generated based on the results of scheduling and binding. The RTL code is then instanced into a larger design and passed through the vendor tool flow to create a bit-stream to program the FPGA. The control flow and state update events are represented in the IR by the completion of operations. The operations corresponding to control flow events are generally the branching operations found at the end of basic blocks. State update events generally correspond to the completion of operations that store data in memory or in static single assignment (SSA) registers. To instrument the hardware design for debugging, software needs to track the correspondence between source-level events and the operations. This correspondence can be maintained using the -g flag to instruct the compiler front-end to annotate the IR representation with debugging information. Maintaining the correspondence through the rest of the HLS flow is straight forward as the operations from the IR are assigned to specific functional units in the RTL. Maintaining the correspondence is complicated by compiler and HLS optimizations to the design flow. In the previous reviews, [5] [6] has addressed many aspects of HLS correspondence problem. In this paper, we move forward with the assumption that the appropriate correspondences have been maintained and leave further investigation into the correspondence problem to be addressed as future work. Incrementing HLS circuits for debug has also been previously proposed [7][8][9][10]. However, most previously proposed methods are scan-chain [8][9] or bit stream read back [10] based, scan-based [11]

3. Observability ports

We propose to instrument the HLS designs with Event Observability Ports (EOP) to provide visibility of the state update and control flow events. The ports consists of (a) an event signal which is a one-bit signal that is asserted when the operation corresponding to the event has completed and also validates the data signal which provides the result of the corresponding operation and (b) a data signal. EOP provides an abstract which links the source-level correspondence information to the final hardware implementation of the system. EOP benefits in adding the ports to the design after it has been scheduled and bounded. By this, no modification in the results of these processes and potentially obscures the bug where the user is trying the find. The incremental debug insertion provides both better performance and higher productivity compared to recompiling and inserting instrumentation into the net-list [12] [13]. Delaying EOP insertion until after RTL-code generation permits the possibility those EOPs could be incrementally inserted at these late points in the design flow.

The relevant event and signals has to be created to instrument a design with an EOP. An important source of information to complete this task is the STG. Using the definition of STG provided by[4], an STG is a directed graph with a set of control states with a set of transitions between them. Each transition is associated with a transition condition. Each control state also contains a set of operations. Each operation is associated with a guard condition to control its execution. The RTL code generation process implements the STG according to the structured circuit model. The guards conditions from the STG are used to identify the operations correspond to the control flow and the state update events. The guarding conditions are often implemented as register clock enables for the hardware registers. This is beneficial because registers survive logic synthesis more often than combinational logic while preserving name correspondence to the RTL. Hence, the register inputs use the data signal and the register clock enable is used as the event signal. This process is advantageous because it utilizes the circuitry that already exists in final circuit.

An important property of EOPs is the relative assertion rate (RAR) of the event signal. This assertion rate is calculated by dividing the number of occurrences of the event signal by the execution latency of the hardware. The RAR of an event signal A is then calculated as the ratio of the assertion rate of A to the lowest assertion rate in the design. These assertion rates are not always statically determined. In these cases, a user-guided approach may be useful. For example, when a for-loop has a variable loop bound, Vivado HLS allows the user to specify the expected number of loop iterations using the TRIPCOUNT directive [14]. The TRIPCOUNT directive provides a number to Vivado HLS so the tool can calculate an estimate of the latency of the design. A similar, directive-based approach could be used to assist instrumentation software in estimating RAR. The RAR data would then be used to efficiently instrument the EOPs for trace.

4. Traceability event ports

The primary purpose of developing EOPs is to enable the efficient trace of source-level control-low and state update events. We propose the use of Event Observability Buffers (EOB) for tracing EOPs. An EOB is a small, independent trace buffer with a data input and data storage enable input [15]. There are advantages of using EOBs over the monolithic trace buffer of an ELA. The small size of EOBs often allows an EOB to be allocated on an EOP-by-EOP basis. This allows the storage depth of the EOB to be configured according to the RAR of the EOP's event signal. For example, an EOB could be configured with a greater depth when the RAR of an EOP's event signal is high. Additionally, by connecting the EOPs event signal to the data storage enable input of the EOB the event data signal is only stored when an event has actually occurred. Small, independent trace buffers also have advantages for incremental insertion after place and route [7][8]. On the other hand, the centralized, monolithic nature of the ELA prevents it from being able to make storage decisions on an EOP-by-EOP basis. This means that the ELA must trace both the event and data signals and will likely store event data signals when they are invalid thereby wasting the limited trace buffer memory.

Additionally, the ELA model provides equal storage depth for all trace buffer inputs. Therefore, ELAs cannot adjust storage depths on a per input basis. EOBs are most useful when the RAR of different EOPs is vastly different. For example, consider an HLS design that computes an 8-bit unsigned char value to determine the loop-bound of a for-loop that computes a 16-bit short int on each iteration. To enable the trace of these events EOPs are added for each event. The maximum RAR of the 16-bit event (compared to the 8-bit event) is 256. This means that the 16-bit event will occur a maximum of 256 times for each of the 8-bit events. The user could also provide an expected maximum number of occurrences of the 16-bit event via a directive. For the purposes of this example let us assume the user has provided a maximum RAR of 32. The individual configurability of EOBs allows the instrumentation software to allocate 32 16-bit EOB entries for each 8-bit EOB entry. Sizing EOBs in this manner helps them fill at approximately the same rate. Another advantage of the EOB approach is the ability for EOPs to share EOBs when their event signals are asserted during mutually exclusive clock cycles. EOB sharing is accomplished by multiplexing the EOB data inputs and using the event signals to appropriately select each input. The data storage enable signal is then created by logically ORing both event signals together.

In the previous example, assume that the 8-bit loop bound event occurs a cycle before the first 16-bit event ensuring that the event signals of the EOPs is asserted during mutually exclusive clock cycles. Having these two EOPs share the same EOB avoids the situation of wasting an entire BRAM to capture the 8-bit event or relying on the distributed memory within the FPGA that might not be available. Adding a multiplexor in front of the EOB increases the

potential that the EOP data signal will become a critical path in the design. One approach to mitigate this problem is to apply multi-cycle path constraints to the EOP data input paths.

5. Order and timing recover

The primary challenge of EOB method is the timing relationship between events is lost. This relationship is inherently preserved by the ELA approach because of the use of a monolithic trace buffer. This problem can be solved by using an event reference trace. It uses a single EOB to trace the event signals of the EOPs. This is accomplished by connecting the event signals of the EOPs to the data input of an EOBs allocated specifically. The same event signals are logically ORed to establish the input for the data storage enabling signal of EOB.

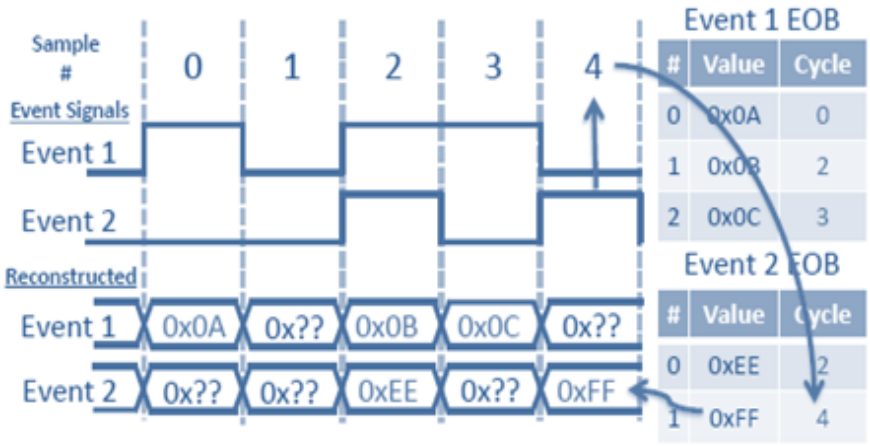

Fig.1: Event order and timing recovery by event reference trace.

The logical ORed signal creates a data storage enable input that filters the storage of data during cycles in which no events occur. A cycle-accurate reference trace can be created by replacing the previous data storage enable signal with a constantly asserted enable signal if needed. Then the event reference trace can be used to recover the results from the corresponding EOB by matching the last assertion in the event reference trace with the last value written to the corresponding EOB. This step is repeated moving backwards through each sample of the event reference trace. In Figure 1, it is shown the representation of a cycle-accurate event reference trace after it has been uploaded. The other approach to recover the event order is by using high-level knowledge of relationships between events. For example, consider the trace of a completely un-optimized version of a design in which events in the hardware occurred in the same order they do in the original specification. The scheduling and binding optimizations practically guarantee that events will not occur in the same order in the hardware as in the source. Recall that the operations that correspond to the source-level events are contained in the STG. Since hardware is generated from the STG we can expect that the sequential relationships of events are also preserved. In this method, firstly all EOBs start their traces at the beginning of the HLS design operation. Secondly, all EOPs representing control flow decisions must be included in the trace.

6. Experimental setup

We have evaluated the effect of preserving the required signals to create the EOPs and comparing the efficiency of capturing events using EOBs to standard ELA methods. For this purpose, the instrumentation of three small HLS benchmarks with EOPs is implemented. These designs include a fully-pipelined fir filter, a partially-pipelined sample mean and population variance estimator, and an un-pipelined floating-point accumulator.

The circuits are compiled to RTL using Vivado HLS 2013.2. This is configured to target a ZYNQ 7020 device with 10 ns for the fully-pipelined design and 25 ns for the partially-pipelined and un-pipelined designs. The lower clock rate target for the partially-pipelined and un-pipelined designs reduced the latency of the floating point operations and avoided inserting empty clock cycles which reduce the events count which are captured by the proposed method. Enabled pipelining in fully-pipelined and partially-pipelined designs needs to add the HLS PIPELINE directive to each design. To achieve an initiation interval (II) of one for the fully-pipelined design required that we apply the HLS PARTITION directive to two arrays in the design.

Next, the design was instrumented with EOPs corresponding to the control flow and state update events that has to be traced. The circuits are instrumented with the four step procedure. First, the C-code is tested to determine a set of events that provide a good picture of the execution of the design. Second, the mapping between events in the C-code and RTL was determined. Third, the design was synthesized using XST (14.6) and an EDIF net-list was written. Finally, the EOPs are added as top-level ports of the EDIF net-list. This is done by searching the EDIF net-list for the needed signals and routing them to the top-level ports. In addition to the EOP signals ports are also created for the clock and start signals. If a needed signal was not found in the design an attribute was added to the RTL source. The RTL was then re-synthesized and the EDIF file is then recreated. After the instrumentation with EOPs, an event-trace strategy is selected and trace buffers are configured according to that strategy. The EOP-instrumented EDIF file was then instantiated within a VHDL wrapper with trace instrumentation. This wrapper is run through the Xilinx Plan-Ahead design flow to generate a bit-stream. This bit-stream is downloaded to a Zed Board and then a trace experiment was run. The goal of each trace experiment was to determine the number of control flow and state update events captured by the currently implemented trace strategy. The test-bench circuitry is configured such that each experiment is triggered by the DUT start signal and ends when an EOB is full. The contents of the trace buffers are uploaded to the host machine and the events are compared.

7. Results

Two sets of experiments were performed on each test design. The first set demonstrates the effect of the keep attribute to preserve name correspondence from the RTL to the net-list. The remaining experiments test the efficiency of different trace strategies used to trace EOPs. Baseline design statistics are presented in Table 1. The experimental results in Figures 2 and 3 are normalized to the design statistics in Table 1.

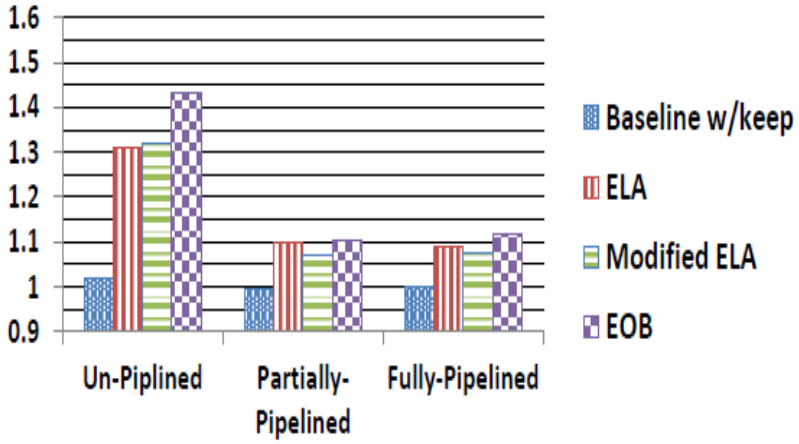

Fig. 2: LUT usage normalized to the baseline.

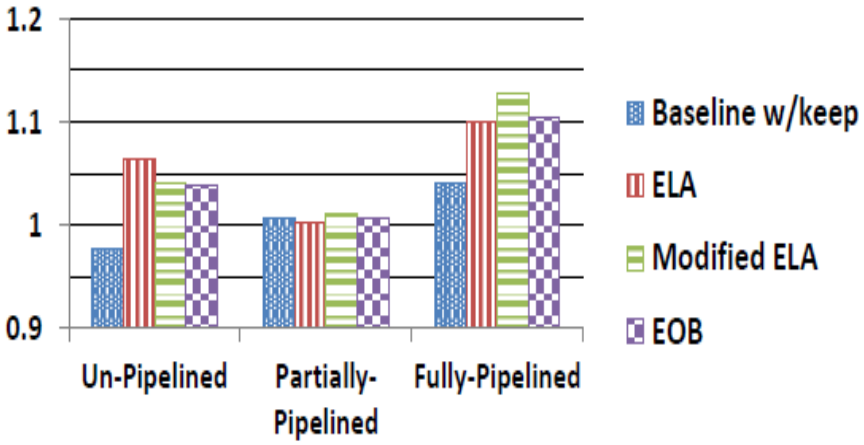

Fig. 3: Minimum clock period normalized to baseline.

Table 1: Maximum Resource Utilization of Baseline Designs

Design	REGs	LUTs	Min. Clk. Period
Un-Pipelined	358	332	18.738 ns
Partially-Pipelined	817	1908	19.886 ns
Fully-Pipelined	922	696	8.726 ns

The first set of experiments was performed to determine the effect of using the keep attribute to preserve name correspondence between the RTL and net-list. Name correspondence Was preserved to ensure that the correct signals were used to instrument the circuit with EOPs. Figures 2 and 3 show that the keep attribute had very little effect on LUT usage or the minimum clock period. This is because many of the signals to which the keep attribute was applied already existed under a different name. The worst effect of the keep attributes was seen in the fully-pipelined design which saw a 4% increase in LUT usage. In all cases, the EOB strategy increased the number of source-level events captured shown in Table 2.

Table 2: Captured Events list

Design	ELA	Mod. ELA	EOB
Un-pipelined	388	752	1505
Partially-Pipelined	750	1424	1507
Fully-Pipelined	1533	1533	1538

The EOB strategy was most successful on the un-pipelined design where it captured 3.88 times more events than the ELA strategy and 2 times more events than the modified ELA strategy. The partially-pipelined design is particularly interesting because it represents a more realistic use case as it alternates between dense and sparse regions of events. The sparse nature of the event's in the un-pipelined and partially-pipelined designs created a lot of opportunities for the EOB strategy to optimize the trace. On the other hand, the fully-pipelined design contained fewer opportunities for trace optimization. For example, there were no opportunities for buffer sharing. Implementing the EOB strategy in these instance only results in the increase of five events over the ELA strategies. [15]" Suggested a scheme in which the inputs of an ELA style trace can also be shared. Figures 2 and 3 show a great deal of resource and minimum clock period parity between the trace strategies. That is a fine result. However, the designs provided in this case look at are small. Hence, greater paintings may be needed to decide how the EOB strategy scales to large designs.

Its miles cleared that the EOB approach is powerful for designs which have sparse occasion areas and no longer effective for the designs that comprise best densely-populated occasion regions. The reality that each hint approach became a hit beneath one-of-a-kind occasions indicates that achieving maximum occasion visibility calls for the application of the proper trace strategy. A right analysis of statistics available from HLS gear has the capacity to allow the creation of an automatic technique to deciding on trace techniques.

8. Conclusion

A new method for tracing source-level events during the in-machine execution of HLS designs is delivered in this paper. Our method gadgets HLS generated hardware with EOPs that provide observability of source-level activities. To capture the run-time execution of the HLS-generated hardware we have proposed using small, independent trace buffers known as EOBs. The most critical function of EOBs is that they can be configured to seize information simplest whilst an occasion occurs. We've got also proposed new strategies to recover the order and timing of source-stage occasions captured into specific EOBs. In addition, we have delivered the idea of RAR which may be used to assist configure EOB depth efficiently. We also observed that EOBs may be used to put in force exceptional techniques for tracing supply-degree occasions through EOPs. Our maximum successful strategy established the capability to seize extra supply-stage activities than the standard ELA-like configurations. But, the achievements of this method become restricted to HLS benchmarks that contained as a minimum some degree of manipulate go with the flow. We also observed that for fairly-pipelined designs EOBs also can be configured to put in force a more well known ELA-like strategy that eliminates the want for an event-reference trace and lowers the performance overhead.

References

[1] J. Cong, L. Bin, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhiru, "High-level synthesis for fpgas: From prototyping to deployment," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, vol. 30, no. 4, pp. 473–491, 2011.

[2] K. Rupnow, L. Yun, L. Yinan, and C. Deming, "A study of high-level synthesis: Promises and challenges," in ASIC (ASICON), 2011 IEEE 9th International Conference on, Conference Proceedings, pp. 1102–1105.

[3] J. Noguera, S. Neuendorffer, S. Haastregt, J. Barba, K. Vissers, and C. Dick, "Implementation of sphere decoder for mimo-ofdm on fpgas using high-level synthesis tools," Analog Integrated Circuits and Signal Processing, vol. 69, no. 2-3, pp. 119–129, 2011.

[4] J. Cong and Z. Zhang, "An efficient and versatile scheduling algorithm based on sdc formulation," pp. 433–438, 2006.

[5] K. S. Hemmert, "Source level debugging of circuits synthesized from high level language descriptions," 2004.

[6] K. S. Hemmert, J. L. Tripp, B. L. Hutchings, and P. A. Jackson, "Source level debugger for the sea cucumber synthesizing compiler," in Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on, Conference Proceedings, pp. 228–237.

[7] Y. S. Iskander, C. D. Patterson, and S. D. Craven, "Improved abstractions and turnaround time for fpga design validation and debug," in Field Programmable Logic and Applications (FPL), 2011 International Conference on, Conference Proceedings, pp. 518–523.

[8] C.-T. Chen, K. K #252, #231, #252, k #231, and akar, "A source-level dynamic analysis methodology and tool for high-level synthesis," pp. 134–140, 1997.

[9] G. Koch, U. Kebschull, and W. Rosenstiel, "Debugging of behavioral vhdl specifications by source level emulation," in Design Automation Conference, 1995, with EURO-VHDL, Proceedings EURO-DAC '95., European, Conference Proceedings, pp. 256–261.

[10] J. Curreri, G. Stitt, and A. D. George, "High-level synthesis techniques for in-circuit assertion-based verification," in Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on, Conference Proceedings, pp. 1–8.

[11] Murali, K. Hari Kishore, G. Vijaya Padma, L. Srikanth and R.H. Gopalkrishna, "Design of Scan cell for System on chip scan based Debugging Applications", 2016 Springer International Conference proceedings of IC3T, Lecture Notes in Networks and Systems, pg-577-586.

[12] J. Keeley, "An incremental trace-based debug system for field programmable gate-arrays," 2013.

[13] E. Hung and S. J. Wilton, "Incremental trace-buffer insertion for fpga debug."

[14] Murali, K. Hari Kishore, A. Trinadha, Saswat Tripathy and P. Dhanunjaya Rao, "Improved In-System Debugging of High Level Synthesis Generated FPGA Circuits", 2016 Springer International Conference proceedings of IC3T, Lecture Notes in Networks and Systems, pg-513-519.

[15] Dr. Seetaiah Kilaru, Hari Kishore K, Sravani T, Anvesh Chowdary L, Balaji T "Review and Analysis of Promising Technologies with Respect to fifth Generation Networks", 2014 First International Conference on Networks & Soft Computing, ISSN:978-1-4799-3486-7/14,pp.270-273,August2014.

[16] Meka Bharadwaj, Hari Kishore "Enhanced Launch-Off-Capture Testing Using BIST Designs" Journal of Engineering and Applied Sciences, ISSN No: 1816-949X, Vol No.12, Issue No.3, page: 636-643, April 2017.

[17] N Bala Dastagiri, Kakarla Hari Kishore "Reduction of Kickback Noise in Latched Comparators for Cardiac IMDs" Indian Journal of Science and Technology, ISSN No: 0974-6846, Vol No.9, Issue No.43, Page: 1-6, November 2016.

[18] Murali, K Hari Kishore, D Venkat Reddy "Integrating FPGAs with Trigger Circuitry Core System Insertions for Observability in Debugging Process" Journal of Engineering and Applied Sciences, ISSN No: 1816-949X, Vol No.11, Issue No.12, page: 2643-2650, December 2016.

[19] Mahesh Mudavath, K Hari Kishore "Design of CMOS RF Front-End of Low Noise Amplifier for LTE System Applications Integrating FPGAs" Asian Journal of Information Technology, ISSN No: 1682-3915, Vol No.15, Issue No.20, page: 4040-4047, December 2016.

[20] P Bala Gopal, K Hari Kishore, B.Praveen Kittu "An FPGA Implementation of On Chip UART Testing with BIST Techniques", International Journal of Applied Engineering Research, ISSN 0973-4562, Volume 10, Number 14 , pp. 34047-34051, August 2015

[21] S Nazeer Hussain, K Hari Kishore "Computational Optimization of Placement and Routing using Genetic Algorithm" Indian Journal of Science and Technology, ISSN No: 0974-6846, Vol No.9, Issue No.47, page: 1-4, December 2016.

[22] N Bala Gopal, K Hari Kishore "Analysis of Low Power Low Kickback Noise in Dynamic Comparators in Pacemakers" Indian Journal of Science and Technology, ISSN No: 0974-6846, Vol No.9, Issue No.44, page: 1-4, November 2016.

[23] S.V.Manikanthan and V.Rama“Optimal Performance Of Key Predistribution Protocol In Wireless Sensor Networks” International Innovative Research Journal of Engineering and Technology ,ISSN NO: 2456-1983,Vol-2,Issue –Special –March 2017.

[24] T. Padmapriya, V.Saminadan, “Performance Improvement in long term Evolution-advanced network using multiple imput multiple output technique”, Journal of Advanced Research in Dynamical and Control Systems, Vol. 9, Sp-6, pp: 990-1010, 2017.

[25] Rajesh, M., and J. M. Gnanasekar. "Path observation-based physical routing protocol for wireless ad hoc networks." International Journal of Wireless and Mobile Computing 11.3(2016): 244-257.