Enabling Low Impact, Rapid Debug for Highly Utilized FPGA Designs

Robert Hale
robert.hale@byu.edu

Brad Hutchings
brad_hutchings@byu.edu

Abstract— **Inserting soft logic analyzers into FPGA circuits is a common way to provide signal visibility at run-time, helping users locate bugs in their designs. However, this can become infeasible for highly (70-90+%) utilized designs, which leave few logic resources or block RAMs available for internal logic analyzers. This paper presents a fast, low-impact method of enabling signal visibility in these situations using LUT-based distributed memory. Trace-buffers are inserted post-PAR allowing users to quickly change the set of observed nets. Results from routing-based experiments are presented which demonstrate that, even in highly utilized designs, many design signals can be observed with this technique.**

I. Introduction

It is widely understood that debugging FPGA circuits in-system is extremely challenging because observing the behavior of signals typically requires the insertion of invasive debugging circuitry. In-system test and debug is necessary because simulation is often unable to test the user circuit with rich data sets in timely fashion. In the simplest case, inserted debugging circuitry may consist solely of wires that route internal signals to external pads where they can be captured and observed with a logic analyzer. More commonly, complex debug circuitry that resembles the data-recorder of a logic analyzer (referred to as an Integrated Logic Analyzer, or ILA) is inserted into, and placed and routed with the original user circuit.

It can be difficult or impossible to insert an ILA in industrial designs as they are finalized because engineers typically attempt to exhaust as much of an FPGA device as possible in order to achieve lower cost. For example, once engineers implement circuits that utilize FPGA devices at 90%, or if their designs completely consume all available block RAMs (BRAM), it may become difficult or impossible to insert and use an ILA.

The goal of this project is to provide an alternate debugging tool that can be used when ILA insertion is not possible or feasible. This debugging tool is implemented post-place/route to minimize impact on the user's design and reduce implementation time. BRAMs aren't used for memory, leaving these resources fully available to the user design.

This tool achieves observability by scavenging for unused shift-register LUTs (SRLs) in a design and using them, paired with a 2-to-1 MUX, to capture and upload user-signal values at run-time. In Xilinx devices, SRLs are 16- or 32-bit shift registers that can be implemented on 50% of the device's LUTs. Thus, unused SRLs are commonly available even in highly utilized devices. For each signal the user wishes to probe, the tool finds the closest available SRL, routes it to its respective signal, and then connects all thusly used SRLs together into a single shift-register. During circuit operation, user signal values are captured in these SRLs. Once the defined trigger signal occurs, signal capture stops, and the captured values are shifted out to the JTAG port via inserted BSCAN primitives. Though this SRL-based approach may not provide as deep of a trace of user signals as an ILA might, it does provide observability when ILAs are infeasible or when it would take too long to place and route them into the user design.

This SRL-based debugging approach uses the recently introduced RapidWright [13] from Xilinx to add SRL primitives after place and route. RapidWright is an open source platform that provides an interface to Xilinx's Vivado back-end implementation tools. This SRL-based approach queries the Vivado circuit database via RapidWright to do the following: (1) determine the location of available SRL primitives, and (2) place the SRL primitives near their corresponding user signal. Routing of the circuit to the SRLs is completed via the Vivado back-end routing tool.

II. Related Work

A. Commercial Debug Tools

FPGA vendors provide internal debug tools to work with their hardware, such as Xilinx's Integrated Logic Analyzer [3] and Altera's SignalTap [2]. These tools offer a powerful method of viewing design signals in real time. However, they also require substantial resources for their implementation. In addition, these tools typically cannot be added to a user circuit post-map and require the entire user circuit to be re-placed and re-routed. If the engineer wants to change the set of nets being probed this entire compilation process needs to be repeated. For complex and highly utilized designs such iterating can consume an unacceptable amount of time. Some commercially available design tools include incremental functionality that attempts to reuse placed or routed elements to reduce compilation time, however, our tests of these features showed only trivial improvement. Finally, these tools require the presence of unused BRAMs that are relatively limited on FPGA devices.

B. Academic Tools

Several research projects have attempted to address the issues with FPGA debug visibility. Ideas have included guessing what nets should be probed beforehand [5] [17] or attempting to probe nearly every net in the design [1]. Other techniques involve scanning the entirety of the FPGA's current status [16] [15], referred to as readback. This gives

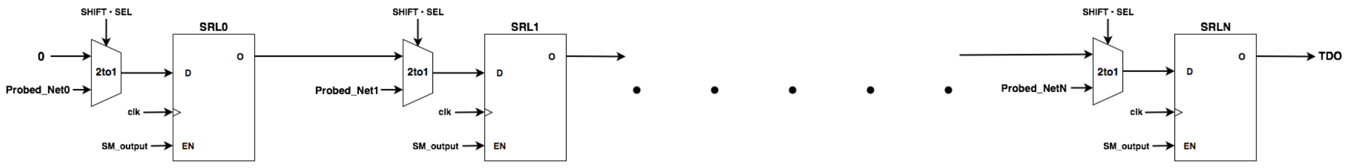

Fig. 1. MUX-SRL Pair Chain

full visibility and requires no added logic or use of FPGA resources, but requires stopping the DUT and offers no signal history.

One research effort has sought to imitate software debugging as closely as possible for FPGAs [11], including full breakpoints and live variable visibility. This comes at the expense of high FPGA fabric overhead. One similarity this method shares with our tool is avoidance of BRAM use, favoring distributed LUT memory.

Other researchers have proposed ways to reduce the time it takes to insert debug circuitry by instrumenting the circuit post place/route, or by partitioning the debug circuitry into a small area on the chip [7] [6] [4] [8]. Overlays have also been proposed to enhance debugging of FPGAs[9].

The most similar work to that presented here is that of Keeley and Hutchings [10]. Their debug tool is similarly instrumented post-implementation with RapidSmith [12]. However, the produced trace buffers are more similar to other research methods in their use of BRAM trace buffers.

Our SRL-based debug trace buffer tool differs from previous work because it focuses on designs that highly utilize the FPGA device (70% or higher). Alongside the benefit of reducing debug time with incremental insertion, our tool finds trace amounts of unused logic resources and utilizes them to enable debugging. Where other internal debug methods would fail due to high resource needs or exhausted BRAMs, our SRL trace buffers can squeeze into crowded designs and provide at least some amount of signal visibility.

III. SRL-Based Debugging

Instrumentation of our debug tool is executed using a software suite of Xilinx FPGA design editing tools called RapidWright [13]. RapidWright allows post-PAR modification of design files. RapidWright is similar to its parent tool, RapidSmith [12]. The instrumentation steps are listed below, starting with the user design.

a) User Design: The user creates a design and completes the Xilinx Vivado design implementation process. Nets to be debugged are marked for debug in the typical Vivado flow and the design is exported to RapidWright.

b) Insert Probes: SRL-based trace buffers, consisting of a 16-bit SRL coupled with a 2-to-1 MUX, are inserted into the design. One is created for each net requested for probing. The source tile of the debug net is identified, and the trace buffer is placed as close as possible to minimize timing issues. However, if needed, the trace buffer can extend to any location on the chip with unused LUTs. The trace buffer is then linked to the chain of other trace buffers in the design (see Figure 1).

SRL-based trace buffers operate in two modes, operation mode and debug mode. In operation mode the MUX passes the value on the debugged net into the SRL. The SRL connects to the same clock that drives that net, recording each subsequent value. Data is stored in first-in, first-out fashion.

Debug mode is used after triggering has occurred and debug data is requested. In this mode, the MUXs of all MUX-SRL pairs will chain together, passing data to the next MUX-SRL pair. The last pair passes data to a BSCAN primitive that interfaces with the host. The correct number of data bits is collected by knowing beforehand how many MUX-SRL pairs have been instrumented into the design.

c) Route, BitGen: After instrumentation, RapidWright produces a new, modified design checkpoint. Xilinx tools are then used to ensure the design is without error, route the design, and generate a bitstream. Route is performed incrementally, meaning that the original user design will be left undisturbed where possible and run-time is minimal. Xilinx bitgen then generates the bitstream.

d) Debug: The bitstream is downloaded to the FPGA and the hardware target is closed from program mode and reopened in JTAG mode. Once the debug system has been triggered, the engineer can send appropriate commands over JTAG to request data from the MUX-SRL chain. This data becomes visible at the user terminal. Scripting is then used to format this data and view it in a waveform. Since the order of the chained SRLs is known, data can be identified by net name automatically. The entire tool-chain was tested and found to correctly capture and display data captured from a simple counter. For the larger benchmark used in this paper, only routing experiments were performed.

IV. Routing Experiments

The goal of this work is to show that the SRL-based debug approach can successfully route to a high percentage of user signals even when the user circuit nearly exhausts all resources on the FPGA device. Successful routing of user signals will depend on the availability of SRL primitives near the desired signal, routing congestion, etc. The approach to the experiments is as follows. For example, assume that you want to determine the likelihood of routing a set of 10 signals in a circuit that utilizes 90% of an FPGA.

1) Randomly select 10 signals from the user net-list.
2) Connect the selected signals to available SRLs.
3) Attempt to route the design.

Repeat this process N times (selecting a different random set of signals each time). If, for example, N/2 of the attempts successfully routed, you may predict that, on average, 10

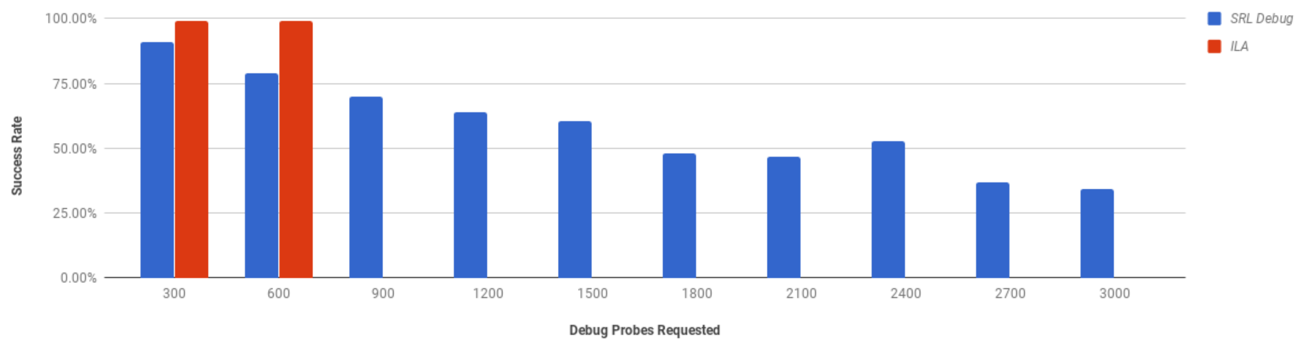

Fig. 2. Outcomes for Experiments on a 70% Utilized Design. At 70% utilization a very high number of probes can be instrumented. While the ILA held out for up to 600 probes, the SRL based approach was able to get a reasonable success rate of over 50% up through 1500 probes. Absence of a red or blue bar means all experiments failed for that value.

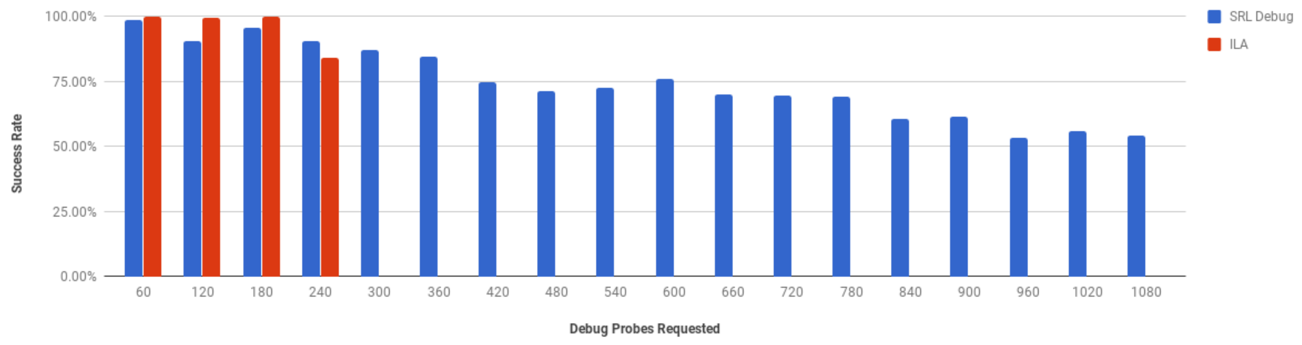

Fig. 3. Outcomes for Experiments on a 80% Utilized Design. When the FPGA's LUTs are 80% utilized, the ILA can only be inserted with up to 240 probes. This design didn't show an early hard cutoff for the SRL based tool, but success drops to 50% after about 900 probes. Absence of a red or blue bar means all experiments failed for that value.

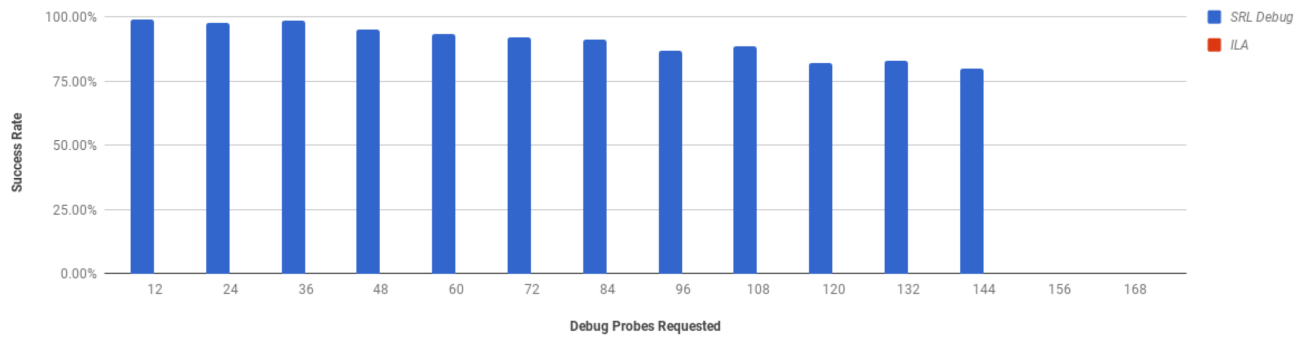

Fig. 4. Outcomes for Experiments on a 90% Utilized Design. Once the design reached 90% utilization, Xilinx tools were unable to probe even 12 nets. The SRL debug tool was able to probe up to 144 nets with a high success rate, never falling below 75%. At this point success falls to zero, as the LUTs on our FPGA reach 100% utilization, a hard cutoff point. Even so, on designs with as high as 90% design LUT utilization, our SRL-based tool was able to probe 2304 bits worth of debug data. Absence of a red or blue bar means all experiments failed for that value.

signals can be successfully routed in a highly-utilized device about 50% of the time. For these experiments, N is equal to 200. These experiments are repeated for varying numbers of signals (from 10s of signals to 1,000s of signals) and are applied to three benchmark circuits (70%, 80%, and 90% LUT utilization for a Kintex Ultrascale XCKU025 containing 145,440 CLB LUTs). The benchmark circuit is created by implementing an array of LC-3 [14] soft processors. The success rate for the SRL-based approach is compared against the success rate for Vivado's ILA by routing the ILA with the same sets of signals.

During the instrumentation phase, checkpoint files containing the benchmark designs are edited using RapidWright tools. These tools insert and place MUX-SRL pairs in the benchmark design for each net marked for debug. Next, Vivado route is run to connect the pins between probed nets and the MUX-SRL pairs. Placement, however, is completed entirely during the RapidWright stage and user design logic

Average Time Consumed (minutes)		
Design LUT density	SRL-based	ILA-based
70%	12	76
80%	15	42
90%	14.5	N/A

TABLE I
IMPLEMENTATION TIME FOR SRL AND ILA

placement remains unaltered. ILA-insertion experiments required a complete re-implementation of the design (place and route). We chose a standard level of optimization in an attempt to give both tools a roughly equal chance at success, as well as keep runtime moderate. For our tool, incremental route is allowed to rip-up and replace nets only where needed. All of the above steps are completed in an automated fashion on a remote supercomputer. Each combination of design and probe count was attempted 200 times to show trends. Scripts record any errors produced during the steps as well as completion time in the case of success. A total of 8,400 experiments were conducted.

V. EXPERIMENTAL RESULTS

Results from the three benchmark designs tested are summarized in Figures 2, 3, and 4. As shown, regardless of design size, the SRL-based debug tool was able to probe a significantly higher number of design nets than Xilinx's ILA tool. In addition to enabling debug at high design densities, insertion of the SRL-based debug logic was far faster than ILA insertion (see Table I). No results are shown for instrumentation time of the ILA in a 90% utilized design, since none of those experiments were successful.

For each successful experiment, regardless of design utilization, the SRL-based approach was able to place and route probes in the design without disturbing the placement of the user design in a fast, incremental fashion[1]. In addition, no BRAMs were consumed. If these dense designs required the use of all BRAMs available on the chip our SRL-based debug tool could still be used to capture and view signal values.

VI. FUTURE WORK

Future work may include: 1) studying timing impacts caused by the instrumentation process, 2) designing improved triggering methods, 3) applying this tool to additional benchmark circuits, and 4) providing deeper, variable-length SRL-based trace buffers.

VII. CONCLUSION

We have presented an SRL-based FPGA debug tool that is capable of leveraging very small amounts of leftover logic resources in highly utilized designs. This paper presents a basic proof-of-concept of its ability to act as a fully triggered debugging tool as well as the feasibility of probing a number of nets even in designs that utilized up to 90% of the FPGA. Though the SRL-based trace buffers are relatively small relative to an ILA, for example, they are able to provide observability when ILA-based techniques are infeasible.

[1]Use of incremental techniques with the ILA did not reduce place and route time.

REFERENCES

[1] Certus debug suite. https://www.tek.com/sites/default/files/media/media/resources/54W_28030_0.pdf. Accessed: 11.17.2017.

[2] Quartus ii handbook volume ii. https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/qts/archives/quartusii_handbook_archive_11.0.pdf. Accessed: 11.17.2017.

[3] Vivado design suite tutorial, programming and debugging. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug936-vivado-tutorial-programming-debugging.pdf. Accessed: 1.26.2018.

[4] F. Eslami and S. J. E. Wilton. Incremental distributed trigger insertion for efficient fpga debug. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4, Sept 2014.

[5] E. Hung and S. J. E. Wilton. Speculative debug insertion for fpgas. In *2011 21st International Conference on Field Programmable Logic and Applications*, pages 524–531, Sept 2011.

[6] E. Hung and S. J. E. Wilton. Limitations of incremental signal-tracing for fpga debug. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pages 49–56, Aug 2012.

[7] E. Hung and S. J. E. Wilton. Incremental trace-buffer insertion for fpga debug. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(4):850–863, April 2014.

[8] Eddie Hung, Jeffrey B. Goeders, and Steven J. E. Wilton. *Faster FPGA Debug: Efficiently Coupling Trace Instruments with User Circuitry*, pages 73–84. Springer International Publishing, Cham, 2014.

[9] Eddie Hung and Steven J.E. Wilton. Towards simulator-like observability for fpgas: A virtual overlay network for trace-buffers. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '13, pages 19–28, New York, NY, USA, 2013. ACM.

[10] B. L. Hutchings and J. Keeley. Rapid post-map insertion of embedded logic analyzers for xilinx fpgas. In *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 72–79, May 2014.

[11] Yousef Iskander, Cameron Patterson, and Stephen Craven. High-level abstractions and modular debugging for fpga design validation. *ACM Trans. Reconfigurable Technol. Syst.*, 7(1):2:1–2:22, February 2014.

[12] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings. Rapidsmith: Do-it-yourself cad tools for xilinx fpgas. In *2011 21st International Conference on Field Programmable Logic and Applications*, pages 349–355, Sept 2011.

[13] Chris Lavin and Alireza Kaviani. Rapidwright: Enabling custom crafted implementations for fpgas. In *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, page to appear, 2018.

[14] Yale N. Patt and Sanjay Patel. *Introduction to Computing Systems: From Bits and Gates to C and Beyond*. Osborne/McGraw-Hill, Berkeley, CA, USA, 1st edition, 2000.

[15] Pankaj Shanker. Spatial debug &#38; debug without re-programming in fpgas: On-chip debugging in fpgas. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pages 3–3, New York, NY, USA, 2016. ACM.

[16] Timothy Wheeler, Paul Graham, Brent Nelson, and Brad Hutchings. *Using Design-Level Scan to Improve FPGA Design Observability and Controllability for Functional Verification*, pages 483–492. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[17] S. J. E. Wilton, B. R. Quinton, and E. Hung. Rapid rtl-based signal ranking for fpga prototyping. In *2012 International Conference on Field-Programmable Technology*, pages 1–7, Dec 2012.