SPECULATIVE DEBUG INSERTION FOR FPGAS

Eddie Hung, Steven J. E. Wilton

Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, B.C., Canada
{eddieh,stevew}@ece.ubc.ca

ABSTRACT

FPGA prototypes have become an increasingly important part of the overall integrated circuit design and verification flow, providing the ability to test an integrated circuit running at (near) speed with realistic inputs and outputs. When unexpected behaviour is observed in the prototype, it is necessary to determine the source of this behaviour; this usually requires observing signals that are internal to one of the devices in the prototype. Tools currently exist to enable FPGAs to be instrumented, but these are normally used in a reactive manner; that is, instrumentation is only added after incorrect behaviour has been observed. In this paper, we propose *speculative debug insertion*, in which a tool automatically predicts what signals will be useful during debug, and instruments the design during the first compilation. If done correctly, this can significantly accelerate the debug process, especially for large prototypes containing many FPGAs. However, it is important that this does not negatively affect the performance, capacity, power, or compilation time. We show that speculative debug insertion is possible, and experimentally evaluate the limits to speculative insertion.

1. INTRODUCTION

Designing integrated circuits that function correctly has become increasingly challenging. Not only does increasing device capacities and faster speeds lead to more complex chips, but the prevalence of SoC design methodologies in which pre-designed IP blocks are combined together onto a single chip often leads to unexpected interactions. Verification of these complex chips has become extremely difficult, and often dominates the time to create working silicon.

Although simulation is an essential verification tool, many design errors will not be uncovered by even the most comprehensive simulation plan. Simulation can run a billion times slower than actual silicon [1], meaning that only an extremely small subset of all operations can be simulated, even when the simulation is accelerated by dedicated hardware [2]. In addition, simulation models can not normally interact with off-chip stimuli, and rely on models that may or may not accurately reflect the environment. As a result, designers regularly resort to FPGA-based prototyping of their designs. Large prototyping boards, often containing multiple FPGAs, are available [3]. Designers can use these boards to test their designs much more thoroughly than is possible in simulation, and to observe their chips running in-situ with real-world stimulus.

When unexpected behaviour is observed in a prototype system, it is necessary to determine its cause. Incorrect behaviour may have many causes including design errors, incorrect software or firmware settings, or invalid assumptions about the environment in which the design operates. This debugging task is difficult, primarily due to a lack of *observability* in the prototype. Unlike simulation, in which any signal in the design can be observed, in a prototype system, only signals which appear at the pins of an FPGA can be observed. This lack of observability makes it difficult for a debugging engineer to deduce the cause of any unexpected behaviour.

Observability can be enhanced in several ways. FPGA vendors provide tools (e.g. SignalTap II, Signal Probe, and ChipScope Pro [4, 5, 6]) which allow the designer to instrument their design and bring important signals to the output pins or to on-chip trace buffers for observation during debug. Third-party tools are also available [7, 8]. Although some of these tools (such as SignalTap II) allow a limited amount of reconfiguration after the circuit has been compiled, in most debugging scenarios, the designer must use the tools to insert the debug logic *before the circuit is compiled*. In many cases, designers will compile the "first cut" of their design

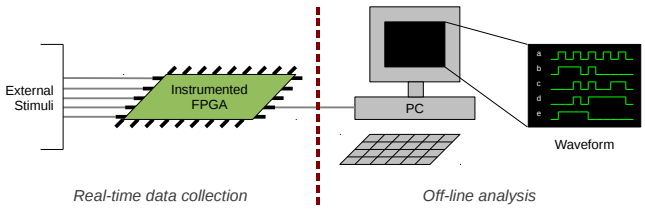


Fig. 1: FPGA Prototyping using a Trace-Based Approach

without debug instrumentation, and then when unexpected behaviour is encountered, determine which signals would be desirable to observe, instrument their design, and recompile. With compile times of large FPGAs steadily increasing [9], this can severely limit debugging productivity, especially for prototypes spread across multiple devices.

In this paper, we propose an alternative. Rather than waiting until unexpected behaviour is observed and then instrumenting and recompiling, we propose that the FPGA CAD tool *speculatively* insert debugging instrumentation every time a circuit is compiled. Before compilation, the CAD tool could determine which signals are *likely* to be important, and insert debug instrumentation to make those signals observable. Ideally, this would be entirely automated and transparent to the user; the user only needs to become aware of the instrumentation (and benefit from the extra visibility available) during debugging.

Although the added observability will be valuable during debug, inserting logic speculatively can have a number of drawbacks. First, it is important to limit the amount of insertion as to not increase the number of FPGAs required. Since prototypes often contain only partially-filled FPGAs (due to limitations in partitioning of large designs), there will often be extra room for significant debug logic. It is also important that the extra debug logic does not place additional stress on the routing fabric, so as to not increase the critical path, nor increase the compile time due to the added routing pressure. Finally, it is important that the CAD tool intelligently choose which signals will be instrumented; recently, several algorithms for automatic signal selection for on-chip circuit debug have been described [10, 11], and those algorithms are appropriate here.

This paper shows that speculative debug insertion is feasible and investigates to what extent each of the concerns in the previous paragraph limits how aggressive an automated tool should be. The paper is organized as follows: Section 2 describes background work and Section 3 explains our speculative insertion flow in detail. In Section 4, we review several algorithms for signal selection and describe the new algorithm used in this work. Section 5 investigates what limits exist for our speculative flow, and how they would affect the aggressiveness of any debug solution. Using these limits, Section 6 presents our speculative insertion tool and shows an example application.

2. BACKGROUND

The primary challenge with debugging a prototype system is the lack of observability. Due to the limited number of I/O resources that can be used to access internal signals, it is very difficult for designers to deduce what is happening inside each chip. When a circuit does not function correctly, it is important to understand the behaviour of the chip in order to identify its cause.

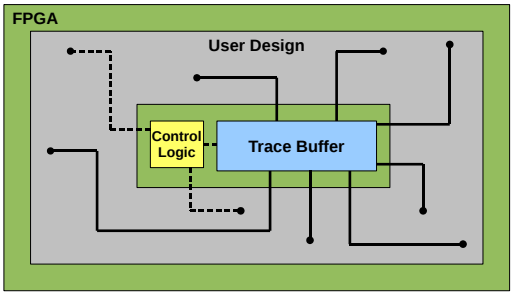
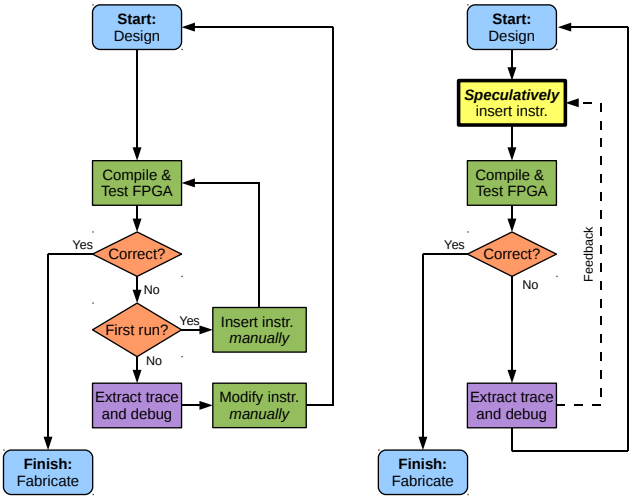


Fig. 2: Trace-Based Debug Instrumentation

As with ASICs, there are two main approaches to FPGA debug: scan-based and trace-based. The advantage of a scan-based approach are that all flip-flop values on a circuit can be observed; but in order to do so, requires the circuit to be precisely halted (for example, by disabling its clock) to allow its node values to be shifted out. In an FPGA, scan functionality can be emulated using general-purpose logic or by using bitstream readback techniques. Reference [12] describes one method of implementing scan-chains by using the general purpose logic resources on FPGAs to serially connect all flip-flops and embedded RAM blocks in the user circuit. This incurs an 84% increase in area and a 20% increase in delay. An alternative technique for viewing the current state of an FPGA is by reading back its configuration bitstream. BoardScope [13] is one such tool, compatible with Xilinx devices, which allows a user to observe or modify the value on any resource in their FPGA. However, BoardScope operates only at the resource level (as identified by its row/column location) and is unaware of the user circuit, leaving it up to the designer to translate between any signal at the RTL level, and its physical location on-chip.

Trace-based approaches require inserting trace buffers and control logic in order to record a small subset of predetermined signal values, as illustrated in Figures 1 and 2. By storing these values on-chip, a sequence of states surrounding an error can be captured as the device continues to operate at full-speed, removing the need to halt the device every time an observation is to be made [4, 5, 6, 7, 8]. Importantly, the signals to be observed must be selected at design-time, before the nature of any bugs is known. In most cases, modifying this set of traced signals requires a lengthy recompile, although its impact may be reduced by using incremental compilation techniques. Due to the high cost of storing signals values into on-chip memory, it is necessary for the designer to select only the most interesting and relevant signals for observation, which can be a difficult and tedious process. The area overhead of trace-based designs depends on how many signals are observed, and how long they are observed for.



(a) Existing *reactive* flow (b) Proposed *speculative* flow

Fig. 3: Existing and Proposed Design Flows

3. SPECULATIVE INSERTION

In this section, we describe our speculative debug insertion flow, and the framework within which it operates. Figure 3 shows a typical prototyping flow along with our speculative flow. In the typical flow, a designer first maps his or her design to the FPGA prototype using standard compilation tools (this often requires maniplating the original design to make it more amenable to FPGA implementation). The implementation is then tested, often *in-situ*. If incorrect behaviour is observed, the designer can then use tools such as SignalTap II or ChipScope Pro to add instrumentation to the circuit. Typically, at this point, the designer will use his or her understanding of the nature of the failure to carefully determine the number of signals that will be observed and the size of the trace buffers. The circuit is then recompiled (sometimes using incremental techniques) and the error reproduced. The designer can then use the data in the trace buffer to help narrow down the cause of the failure, possibly using formal techniques such as [14]. We refer to this typical flow as *reactive* or *post-mortem*, since instrumentation is only done after a failure is observed. FPGA compile times are significant, often taking as long as a day for a large FPGA. For a board with many large FPGAs, each compilation "spin" takes significant time, leading to a time-consuming debug flow.

Our proposed *speculative flow* differs in that, before compilation, the tool will automatically determine a set of signals that it predicts may be useful during debug and instrument the design accordingly, without intervention from the user. In FPGA prototypes, pin constraints often dictate that each FPGA is not filled to capacity (often far below capacity). Our speculative flow uses these unused resources to implement debug circuitry. Figure 3b shows that the design is instrumented before the first compilation, meaning that when the implementation is tested, it is possible to immediately obtain debug data without recompilation, providing a "head start" in the debugging process.

It is important to note that the speculative flow can coexist with the reactive flow. Each iteration, the designer may add instrumentation as in the reactive flow to provide visibility into signals that he or she deems important. During the recompilation, the tool can supplement these selected signals with signals that it predicts may also be useful, hopefully leading to more debugging information. The number of extra signals that the tool can select depends on the amount of spare resources in the FPGA. In this work, we assume that the trace buffer is operated by an external trigger signal and select only the signals that are to be sampled.

The success of this flow depends critically on two factors. First, the tool must be able to effectively and automatically select signals for observation. This will be discussed in Section 4. Second, the tool must be careful not to add so much instrumentation that the designs targeted for each FPGA no longer fits in that device. In addition, it must ensure that the extra logic does not lead to congestion that may slow down the circuit, increase power significantly, or increase place and route run-time. These issues are the focus of Section 5.

4. SIGNAL SELECTION

A key component to the success of this speculative flow lies in the ability to automatically predict which signals a designer would likely want to look at in their circuit. By hand, signal selection can be a lengthy and laborious process requiring expert knowledge of the design, which in many cases may not be readily available — for example, with IP blocks designed by third-parties, and even with IP designed internally by different teams, it is unlikely that a single expert exists to determine how best to construct a global debug solution.

Recent work [10, 11, 15] have proposed several solutions to this trace signal selection problem. These techniques aim to enhance the visibility into the integrated circuits. The intuition is that some signals will provide more insight than other signals into the internal state of the chip. In [11], the focus is on selecting signals from which the value of as many *additional* signals can be deduced. For example, if the input of an AND gate is observed to be a 0, the output can immediately be deduced to be 0. The approach in [10] attempts to select signals that reduce the size of the state space as much as possible. For example, selecting two highly correlated signals provides little additional information over selecting just one of these signals on its own. Rather than selecting two correlated signals, [10] argues that it is better to select uncorrelated signals since together, they can better prune the possible state space of the system. Commercial signal selection algorithms also exist [7], however the details of these algorithms have not been disclosed.

Based on these previous techniques, we created a simpler, but more scalable, signal selection method for use in this work. First, a connectivity graph is built in which vertices are used to represent each register in the circuit, and edges indicate the existence of a connection (possibly through combinational logic) between pairs of registers. From this connectivity graph, the *centrality* of each vertex is computed and used to rank the importance of all registers in the design. Using this method, we have observed that arguably important signals — state machine bits, enable and bus control signals — are ranked highly across several benchmarks.

It is important to note that these signal selection techniques are not intended to produce *better* signal selections than an expert designer having observed a malfunctioning prototype. Our signal selections are used to automatically predict which signals may be useful during debug in an attempt to make effective use of spare resources on the FPGA. As described in the previous section, these automated techniques can be combined with manual signal selections to provide even more observability into the internal operation of the prototype.

5. LIMITS TO SPECULATIVE INSERTION

Inserting debug instrumentation into any FPGA design will consume logic and memory resources. Although utilizing these otherwise-spare resources may be thought of as being 'free' from an area perspective, it would not be desirable for circuit performance to be affected. This section investigates the limitations of speculative debug insertion by quantifying the trade-off between how aggressively the circuit can be instrumented and the overhead that will be incurred.

In the following experiments, we have used Altera Quartus II v10.1 software (64-bit, single-processor mode) to implement a LEON3 system-on-chip [16] design onto an Altera Startix III FPGA. This design was configured as an eight-core multi-processor with instruction and data caches, full floating-point and memory management support, as well as DDR memory controller, Ethernet and UART IP blocks all attached to an internal AMBA 2 bus. Without any debug instrumentation, this circuit consumes almost 100,000 ALM (Adaptive Logic Modules, which are the basic logic unit in Stratix FPGAs) resources, using up close to 70% of the largest Stratix III device (EP3SL340). The design contains 47,000 user registers, which were ranked using the algorithm described in Section 4. For each experiment, the highest ranked signals selected for instrumentation. To limit experimental noise, each configuration was compiled using ten different random seeds, and the geometric average reported.

5.1. Impact on Area: Logic

To understand how much debug logic can be speculatively added to our benchmark circuit, we first constructed a variety

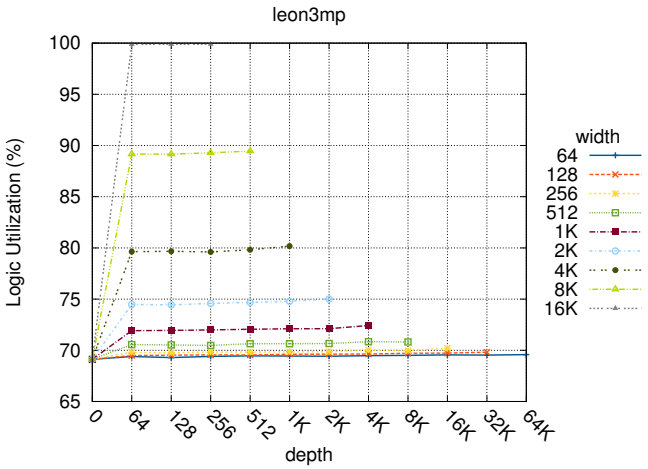


Fig. 4: Area: Trace Depth vs Logic Utilization

of instrumentation scenarios and measured their utilization on the FPGA. Figure 4 shows the total logic utilization of the instrumented circuit, after place and route, over a variety of different trace configurations. The number of signal samples stored by the on-chip trace buffer (its depth) is shown on the X-axis, and the number of signals observed simultaneously is displayed as different data series. The utilization value displayed on the Y-axis represents the number of ALM resources partially or fully used in the implemented design. We found that this was the most appropriate metric for logic utilization as the other measures reported by Quartus II did not account for blocked resources, making it impossible to achieve 100% utilization.

To a first order, the logic area required for debug instrumentation is proportional to the number of signals observed, but roughly independent of the depth of the trace buffer. This suggests that a speculative debug tool can use the number of available logic elements to determine how many signals can be observed without increasing the size beyond the capacity of the FPGA. Where this breaks down, though, is for the set of debug configurations with a trace width of 16K signals which consumes all the logic resources present on the FPGA. Recognizing that it costs approximately 20% of the logic resources to trace 8K signals, it would be assumed that, following from the existing trend, that to observe 8K more signals would cost an additional 20%; this would exceed the capacity of the chip. Yet, Quartus II is still able to implement this configuration by packing the logic more densely, but as explored in a following subsection, this comes at a detrimental cost to delay.

We have observed that almost all of the instrumentation logic is formed of register resources. Although we cannot examine the source code behind the proprietary SignalTap IP to determine exactly how these registers are used, we believe that they are pipelining registers for routing between observed signals and their trace buffers, in order to minimize

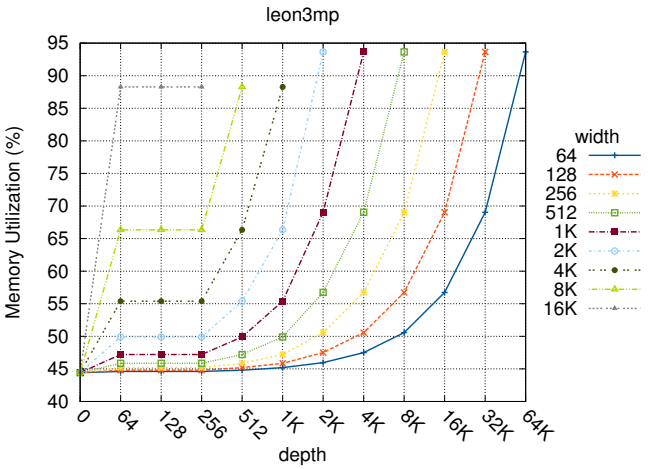


Fig. 5: Area: Trace Depth vs Memory Utilization

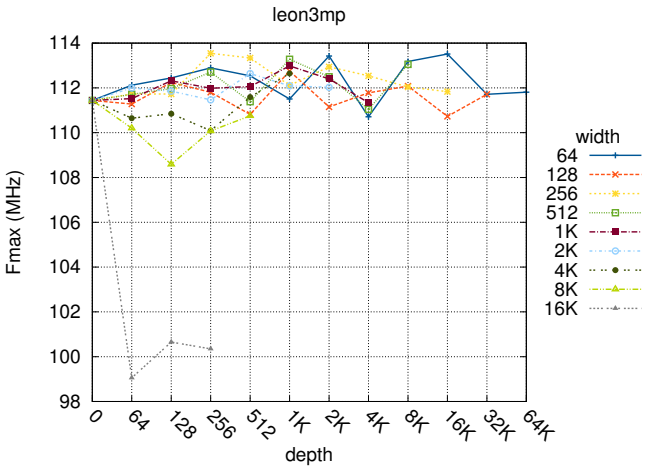


Fig. 6: Delay: Trace Depth vs Fmax

their impact on timing. This matches with the previous observation that total logic utilization is unaffected by trace buffer depth, because once the signals have been routed to their memory resource, which are located in groups on the FPGA, it becomes trivial to increase the number of samples captured by connecting it to any of the other memory blocks situated nearby.

5.2. Impact on Area: Memory

Figure 5 shows the corresponding plot for the memory utilization. This utilization value captures only how many of the M9K memory resources available on the Stratix III device are being either partially, or fully, occupied. In our experiments, we observed that Quartus II would resort to using the larger M144K memory resources (which accounts for approximately 40% of the total on-chip memory) only after it had exhausted all M9K resources. As expected, the number of memory resources required increases with both the width and depth of the trace configuration. Because our metric treats partially utilized memory resources as fully utilized, the memory usage results are flat for configurations with 256 or fewer samples (the widest configuration of a M9K block corresponds to a minimum depth of 256 words).

These results suggest that any speculative debug insertion tool must also consider the amount of available memory when determining how many signals can be observed, as well as how deep each trace buffer can be.

5.3. Impact on Delay

Although making use of the unoccupied resources on an FPGA is 'free' from an area perspective, the speculative insertion of debug logic can be expected to impact the circuit's performance. In our experiments, we left the main clock of the LEON3 circuit constrained to its default value of

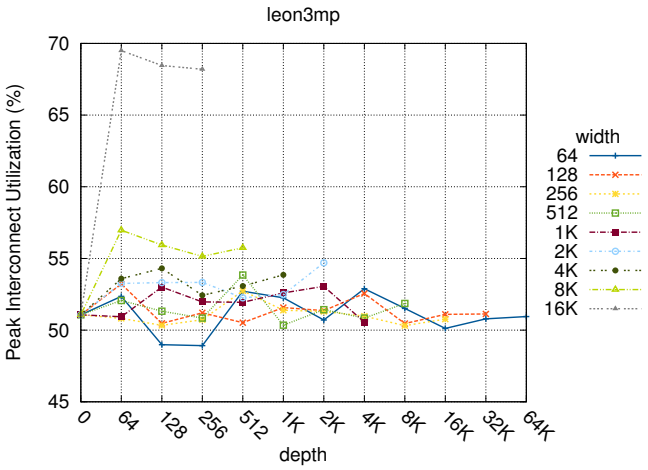


Fig. 7: Delay: Trace Depth vs Peak Interconnect Utilization

150 MHz, which is not met either before or after speculative debug insertion. Although over-constraining is not realistic of industrial designs, this was necessary for us to accurately quantify the impact of speculative insertion on circuit delay.

Figure 6 plots the estimated maximum clock frequency of the implemented circuit, as determined by the TimeQuest Timing Analyzer tool in Quartus II. Even though ten different compilation seeds were employed for each debug configuration, the results still exhibit experimental noise due to the nature of the heuristic CAD algorithms. Nonetheless, the results suggest that there is little impact on circuit speed for all widths less than or equal to 8K (difference of less than 3%). For a trace width of 16K signals, however, Fmax is reduced by over 10%. At this configuration, as discussed earlier, it is necessary for the CAD tool to optimize for area rather than speed in order to fit the entire instrumented circuit onto the device.

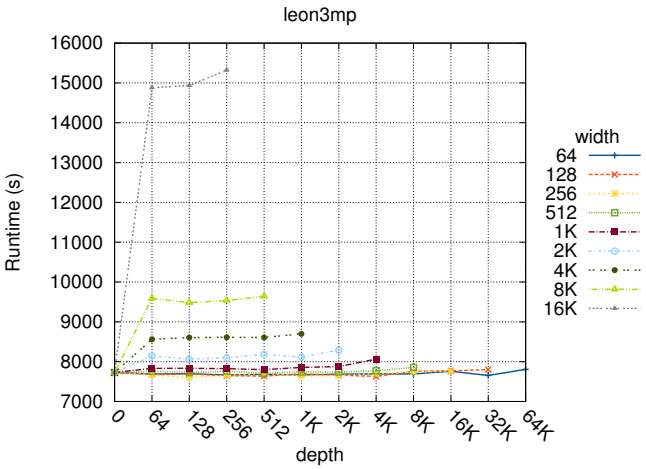


Fig. 8: Runtime: Trace Depth vs CAD Runtime

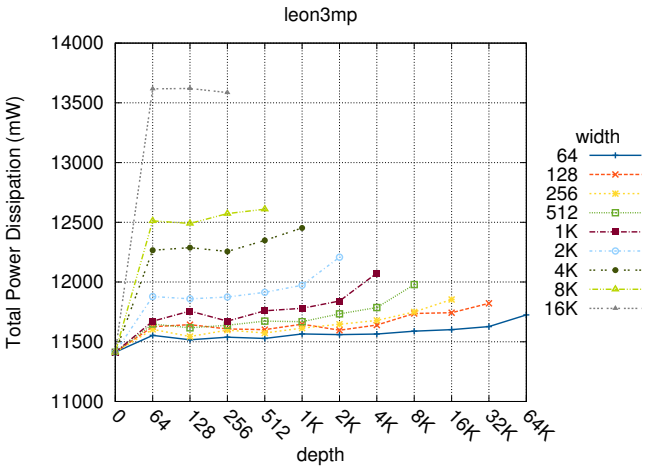


Fig. 9: Power: Trace Depth vs Power Dissipation

A related concern that designers may have is the effect that speculative insertion may have on circuit routability. Figure 7 shows the *peak* interconnect utilization for all trace configurations. Circuits instrumented with a width of less than 16K signals only incur a slight increase in peak utilization over the original value of 51% (for which the corresponding *average* interconnect utilization is 25%). This appears to indicate that sufficient routing resources have been provisioned for this FPGA device so that routability is only minimally affected by debug insertion.

From these results, it is clear that the speculative debug tool can use an estimate of the available resources on the FPGA to determine the amount of instrumentation to add. As long as the tool ensures that the capacity of the instrumented FPGA does not approach 100%, no impact on delay should be expected.

5.4. Impact on Run-time and Power

Figure 8 shows the total CAD runtime for the mapping (synthesis) and fitting (place and route) stages of Quartus II. When adding sufficient instrumentation to the LEON3 benchmark to observe 4K debug signals simultaneously, a 10% penalty to runtime is observed. The run-time goes up dramatically when instrumenting 16K signals; this is because the CAD tool has to work much harder to pack the circuit more tightly into the device. In contrast, runtime appears to be only mildly affected by the depth of the trace buffer, which follows on from the previous observation that very little logic is added as the trace depth increases.

We expect that in most implementations, far fewer than 4K signals will be observed; this implies that only a small impact on run-time is to be expected when using speculative debug insertion.

Inserting additional logic into an FPGA will also affect its power consumption, and this is shown in Figure 9. These values are for the design running at a 150 MHz clock frequency, and were obtained using Quartus II's PowerPlay Power Analyzer tool operating in its vector-less estimation mode. In common with the previous performance metrics, increasing trace depth has a smaller effect on power compared to increasing its width. For this LEON3 design, it can be seen that up to 4K signals can be traced simultaneously for less than 10% power overhead.

5.5. Debug Aggressiveness

While it is possible to fill all of the free FPGA resources with speculative debug logic, this is not desirable as doing so will have a considerable effect on its performance. With the overheads investigated in this section, the limiting factors can be classified into either hard or soft limits. Hard limits are those posed by resource availability, which cannot be exceeded (the exception being a small amount of flexibility with how tightly the logic is packed); whilst soft limits are those that can have an unbounded effect on circuit performance, such as with delay, runtime or power. With these soft limits, it would be up to the designer to determine how much of a penalty he or she can accept.

For speculative insertion, if it was assumed that a designer would be unwilling to tolerate any decrease in the maximum frequency of a circuit, but could accept a 10% increase to its compilation runtime and power consumption, this would correspond to a trace configuration of 4096 signals by 1024 samples, which would cover almost 10% of the user registers in the design. For the LEON3 design, this equates to a trace solution which utilizes 10% of the FPGA's logic resources, and 40% of its memory resources. Any further increase in the width of the trace solution (and hence logic) observed will increase runtime and power beyond these acceptable overheads. Although increasing memory utilization has a minimal effect on these metrics, the SignalTap debug

IP requires that the depth of the trace buffer is a power of two between 64 and 128K samples. This restriction prevents a deeper trace solution from being realized in this design: doubling the depth to 2K samples would exceed the available memory capacity.

From this analysis, we propose that a reasonable trade-off for debug overhead and aggressiveness lies with inserting a trace solution which consumes 10% of the FPGA's total logic resources, and as many memory resources as possible.

6. IN-SPEC: SPECULATIVE INSERTION TOOL

We have developed a tool, called In-Spec, which can automatically and speculatively instrument any design described by a Quartus II project. In-Spec depends only on Quartus II software to function, the intention being to simplifying the number of tools required and to enable its robust HDL support to be leveraged. The current implementation of this speculative flow consists of two stages. In the first stage, In-Spec fully compiles the original design to obtain the connectivity graph required for signal selection. This is necessary because extracting this data utilizes Quartus Tcl commands available only after place and route. Using this Tcl interface, all register-register paths in the circuit are dumped into a text file, which is then parsed by a script used to build the connectivity graph for signal selection. Once the design is compiled, accurate values for logic and memory resource utilization can also be obtained.

6.1. Estimating Instrumentation Area

In the second stage, the tool calculates the maximum trace configuration that can be inserted, without exceeded the utilization values determined in Section 5. To realize this, an area model was developed to estimate, in advance, how many resources would be required to implement any trace configuration. Using the data gathered in the previous section, an area model of the following format can be constructed:

$$L_{ALM}(w,d) = Aw + B \quad (1)$$

$$M_{M9K}(w,d) = Cwd + Dw + Ed \quad (2)$$

where w and d represent the trace width and depth.

Equation 1 estimates that the logic usage (in ALMs) is a function of the trace width only. Using surface-fitting techniques, it can be determined that: $A = 3.39$ and $B = 123$ meaning that the variable cost for each signal observed simultaneously is approximately 3.4 ALM logic resources, and that the fixed cost $B = 123$ resources. Similarly, for the memory resources modelled by Equation 2: $C = 1/9216$, $D = 0.00118$ and $E = 0.0107$. Particularly interesting is that for the coefficient C, which represents the total number of memory bits required, a value corresponding to the inverse capacity of each M9K resource can be used.

	No Dbg	Spec Dbg	Δ
Logic (% ALM)	74.7	83.1	+8pp
Memory (% M9K)	10.0	52.2	+42pp
Fmax (MHz)	74.0	71.9	-3%
Peak Interconnect (%)	59	57	-2pp
Map Runtime (s)	1441	1620	+12%
Fit Runtime (s)	3384	3980	+18%
Power @100MHz (mW)	5716	6318	+11%

Table 1: Example Application: OpenSPARCT1 (3951x1024)

Using Equation 1, the maximum trace width that can be supported without exceeding the logic utilization limit is calculated. This value can then be substituted into Equation 2 to compute the maximum trace depth, rounded down to the nearest power of two as required by SignalTap. Finally, this trace configuration is instantiated, the highest ranked signals corresponding to the trace width selected for observation, and the instrumented design recompiled.

6.2. Example Application

We have validated In-Spec against a different, large open-source benchmark: the OpenSPARC T1 Processor Core [17], which implements a 64-bit SPARC core supporting 4 concurrent threads, I- and D-caches, along with an MMU. Two processor cores were combined into a single circuit of 100,000 registers, and mapped onto the same EP3SL340 Stratix III device as used in the previous section. The circuit was constrained for a clock frequency of 100 MHz.

A trace configuration of 3951 signals wide by 1024 samples deep was inserted automatically, corresponding to a 4% coverage of the design. A comparison between the two implementations is shown in Table 1. These results show that although the actual logic inserted was only 8% (rather than the 10% requested by our tool) further investigation indicated that all 4000 signals were correctly traced, and that the savings had come from Quartus II recouping some unnecessary logic from the user design. Importantly, this did not significantly affect the circuit's Fmax.

6.3. Future Work

We expect that the information required to build the register connectivity graph can be obtained after the HDL design is elaborated early on in the synthesis stage of compilation, eliminating the current requirement to compile the circuit twice. Similarly, work exists to provide early predictions of circuit area [18]. As part of our future work, we intend to incorporate these techniques, and/or Quartus II's incremental compilation feature, into our speculative insertion tool.

7. CONCLUSION

In this paper, we have presented the concept of speculative debug insertion for FPGAs used in a large prototyping system. In our flow, trace buffers and signal observation circuitry are automatically and transparently inserted before compilation, allowing key internal signals to be recorded during normal circuit operation. The primary difference between our speculative flow and the more traditional flow is that in the latter, debug instrumentation is only added after a bug is found, often necessitating a long recompile cycle. Our speculative flow can give designers a head-start when debugging newly observed errors, possibly reducing the number of debugging iterations. The automated nature of our flow may also be advantageous for engineers that are using third-party IP and may not have a intimate understanding of the internals of the circuit, and hence find it difficult to select important signals.

We have identified two important considerations. First, our flow must be automatic, so it is necessary to have a CAD tool that can automatically determine which signals may be valuable for debug, before the circuit is compiled. This capability is enabled by recent work in signal selection algorithms. Second, it is important that the tool have an understanding of the overhead implications of inserting additional logic, to ensure that the inserted logic does not result in a reduction in speed, power, or an increase in run-time. We have experimentally investigated the limits of automatic insertion to determine how aggressive such a tool can be. Finally, we have encapsulated our ideas into an automated tool and showed a successful example application.

The primary application of this flow is in an FPGA prototyping environment where (a) design errors are expected, (b) there are typically many large FPGAs, so recompile times are long, and (c) many FPGAs are not fully utilized. It is conceivable that such an approach could also apply in a more general FPGA implementation scenario. In this case, speed and area overhead are likely more of a concern, so the automated tool would have to be less aggressive in the amount of debug logic inserted. The results in this paper show, however, that as long as there are spare resources available, significant observability can be added without negatively affecting the quality of results.

8. REFERENCES

[1] R. Kaivola, R. Ghughal, N. Narasimhan, A. Telfer, J. Whittemore, S. Pandav, A. Slobodová, C. Taylor, V. Frolov, E. Reeber, and A. Naik, "Replacing Testing with Formal Verification in Intel(R) Core(TM) i7 Processor Execution Engine Validation," in *CAV '09: Proc. of the 21st International Conference on Computer Aided Verification*, 2009, pp. 414–429.

[2] Mentor Graphics, "A Closer Look at Veloce Technology: Taking Hardware-Assisted Verification to the Next Level," http://www.mentor.com/products/fv/techpubs/a-closer-look-at-veloce-technology-taking-hardware-assisted-verification-to-the-next-level-49128, May 2009.

[3] Synopsys, "HAPS-64 Virtex-6 Motherboard," http://www.synopsys.com/cgi-bin/sbg/fpga_dsdla/pdfr1.cgi?file=HAPS64-Virtex6-Motherboard-ds.pdf, November 2010.

[4] Altera, "Quartus II Handbook Version 10.1 Volume 3: Verification; 15. Design Debugging Using the SignalTap II Embedded Logic Analyzer," http://www.altera.com/literature/hb/qts/qts_qii5v3_05.pdf, December 2010.

[5] ——, "Quartus II Handbook Version 10.1 Volume 3: Verification; 14. Quick Design Debugging Using SignalProbe," http://www.altera.com/literature/hb/qts/qts_qii53008.pdf, November 2010.

[6] Xilinx, "ChipScope Pro 12.3, Software and Cores, User Guide," http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/chipscope_pro_sw_cores_ug029.pdf, September 2010.

[7] Veridae, "Clarus Prototyper: Multi-FPGA Debug Suite," http://www.veridae.com/index.php/fpga-prototyping/clarus-prototyper, 2011.

[8] Synopsys, "Identify: Simulator-like Visibility into Hardware Debug," http://www.synopsys.com/Tools/Implementation/FPGAImplementation/CapsuleModule/identify_ds.pdf, August 2010.

[9] H. Bian, A. C. Ling, A. Choong, and J. Zhu, "Towards scalable placement for fpgas," in *FPGA '10: Proceedings of the 18th annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, February 2010, pp. 147–156.

[10] E. Hung and S. J. E. Wilton, "On Evaluating Signal Selection Algorithms for Post-Silicon Debug," in *ISQED 2011, International Symposium on Quality Electronic Design; Santa Clara, USA*, March 2011, pp. 290–296.

[11] H. F. Ko and N. Nicolici, "Algorithms for State Restoration and Trace-Signal Selection for Data Acquisition in Silicon Debug," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 2, pp. 285–297, Feb. 2009.

[12] T. Wheeler, P. Graham, B. E. Nelson, and B. Hutchings, "Using Design-Level Scan to Improve FPGA Design Observability and Controllability for Functional Verification," in *FPL '01: Proceedings of the 11th International Conference on Field-Programmable Logic and Applications*, 2001, pp. 483–492.

[13] D. Levi and S. A. Guccione, "BoardScope: A Debug Tool for Reconfigurable Systems," in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, vol. 3526, Oct. 1998, pp. 239–246.

[14] F. M. De Paula, M. Gort, A. J. Hu, S. J. E. Wilton, and J. Yang, "BackSpace: Formal Analysis for Post-Silicon Debug," in *FMCAD '08: Proceedings of the 2008 International Conference on Formal Methods in Computer-Aided Design*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–10.

[15] S. Prabhakar and M. Hsiao, "Using non-trivial logic implications for trace buffer-based silicon debug," in *Asian Test Symposium, 2009. ATS '09.*, November 2009, pp. 131–136.

[16] Aeroflex Gaisler, "LEON3/GRLIB SOC IP Library," http://www.gaisler.com/doc/Leon3\%20Grlib\%20folder.pdf, September 2008.

[17] Sun Microsystems, "OpenSPARC T1 Processor Design and Verification User's Guide," http://www.opensparc.net/offers/OpenSPARCT1.1.7.tar.bz2, March 2009.

[18] J. Das, A. Lam, S. J. E. Wilton, P. Leong, and W. Luk, "An Analytical Model Relating FPGA Architecture to Logic Density and Depth," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*.