LIMITATIONS OF INCREMENTAL SIGNAL-TRACING FOR FPGA DEBUG

Eddie Hung, Steven J. E. Wilton

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

ABSTRACT

Developing state-of-the-art custom silicon can be a prohibitively expensive and risky undertaking, due in no small part to the need to perform thorough design verification. Field-Programmable Gate-Arrays offer a flexible platform for constructing prototypes to aid in their verification, but unlike software simulation, observability into these prototypes is a major challenge. Designers can choose to insert trace-instrumentation to enhance on-chip observability, but doing so often requires re-compiling the entire design for each new trace configuration. This work presents two contributions: 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. We find that 99.4% of all used cluster outputs (driving both combinational and sequential circuit signals) can be incrementally-traced to 75% of the free memory-capacity on an FPGA, an order of magnitude quicker than the original compilation and with a nominal impact on circuit delay, for a 20% minimum channel width (10% area) increase.

1. 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 behaviour (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 [1].

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. 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) are 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 signals over time.

To enhance observability, tools such as SignalTap II [3], ChipScope [4], and Certus [5] instrument a user design by instantiating *trace-buffers* on the chip. 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 [6]. In addition to trace-buffers, these tools instantiate connections (or networks) that connect 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.

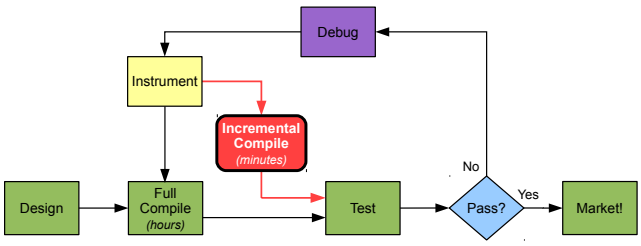


Fig. 1: Incremental-trace flow

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. This flow is illustrated in Figure 1. These additional connections could be created using routing tracks that were not used during the original circuit mapping. If this is possible, a full re-compilation 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.

In this paper, we *evaluate the potential and the limitations of incremental-tracing*. In particular, we evaluate under what conditions a set of signals can be connected to trace-buffers using incremental techniques. Of particular interest is the number of signals that can be simultaneously traced in this way, since this will dictate when this technique is effective in practice. Intuitively, the ability to make these incremental connections will also depend on the architecture of the FPGA; we examine these interactions. Finally, we present a number of CAD techniques which can increase the effectiveness of incremental-tracing.

2. BACKGROUND

2.1. Enhancing Observability

Methods to enhance device visibility during hardware validation can be divided into two broad categories: scan-based and trace-based. Scan-based techniques rely on the serial connection of all (or just a subset) of the flip-flops in a circuit so that their contents can be shifted out for observation, and new values shifted in for controllability. Direct implementation of complete scan chains using FPGA soft logic has a prohibitively high overhead (84% area and 20% delay overhead [7]). However, some modern FPGAs typically have hardware support (device readback) to read not only the value of configuration bits, but also all the state bits in a design. These techniques can typically offer simulator-like visibility into all combinational and sequential signals of the circuit, but only at a cycle-by-cycle basis due to the need for the circuit to be halted before the scan-out procedure can take place. Reference [8] reported that viewing any register in this manner can take between 2 to 8 seconds, precluding its use for real-time debugging.

With trace-based approaches, like that illustrated by Figure 2, a designer pre-inserts a set of trace-buffers into their circuit at compilation, so that during debugging, a history of signal values can be recorded *without* interrupting circuit

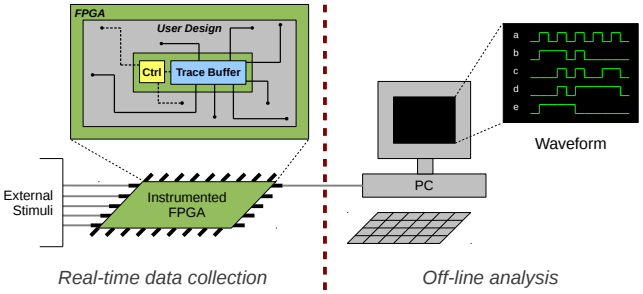


Fig. 2: Trace-based debugging

operation. This capability allows the circuit to be tested using real-world, real-time stimulus, and increases the likelihood of reproducing difficult-to-catch errors, such as those that cross multiple clock-domains. Due to limited on-chip resources however, only a small subset of internal circuit signals can be pre-selected for tracing, and this selection remains fixed until the circuit is re-compiled. Choosing which signals to observe can be a time-consuming task, though a number of automated techniques have attempted to combat this [6, 9].

Regardless, these techniques all require inserting additional logic into the design early in the compilation flow, which will affect the final place-and-routed solution due to the inherently noisy and heuristic nature of CAD algorithms. Rubin and DeHon [10] found that even small perturbations in the router inside the VPR CAD tool can cause the critical path delay to vary between 17–110%.

2.2. Incremental Synthesis

The key idea behind incremental synthesis is to allow a final, place-and-routed circuit implementation to be modified whilst preserving as much of the original solution as possible. The motivations behind this technique are many: to minimize re-compilation effort during the design phase, to preserve timing closure when undertaking Engineering Change Orders (ECOs), or for improved fault and defect resilience [11].

Incremental synthesis techniques for FPGAs are not new however [12, 13], nor is their application in design prototyping and debug [14]. Many of the techniques cited are targeted at modifying the user circuit for functional purposes and go beyond the requirements for incremental-tracing (which are explained in detail in a following section) including provisions for incremental re-packing and re-placement.

An approach much more similar to what we are investigating is taken by Graham et al. [15], who pre-insert unconnected embedded logic analyzers (trace-buffers) into their Xilinx FPGA ahead of time, and subsequently perform low-level bitstream-modification using incremental techniques [16] to connect them to the desired signals. However, this technique still requires some pre-reservation of FPGA resources, preventing their use by the original design. More importantly, what we are interested in are the limitations and scalability

of incremental-tracing — this work examined a trace set of only 128 signals, and only supports existing Xilinx devices, precluding the open investigation of FPGA architectures.

Incremental-compilation design is also supported by commercial vendor tools. Specifically, both Altera's SignalTap II trace IP solution, and SignalProbe, which multiplexes and routes signals directly to the I/O pins for connection to an external analyzer, is supported under this flow [3]. Again, these products can only target real FPGA devices using proprietary software from which we cannot acquire detailed performance metrics.

3. CONTEXT

The first contribution of this paper is to investigate the limitations of using incremental-synthesis techniques for trace-based debug, which we have termed incremental-tracing. The key challenge during incremental-tracing is to connect all of the signals that a designer wishes to observe to a trace-buffer, whilst minimizing the number of changes that are made to the original circuit mapping. As stated in the Introduction, there are many reasons for why this is desirable: reduced re-compilation runtime leading to improved debug productivity, and preserving the original timing of the circuit under debug.

Specifically, in this paper we look at how the feasibility of applying incremental-tracing to a circuit that has not been previously instrumented (i.e. where trace-buffers have not been pre-inserted [15]). We investigate where and why this technique fails, and how this is affected by the FPGA architecture, as well as the number of signals traced simultaneously. In addition, we compare the CAD run-time of incremental-tracing versus a full re-compilation, and measure the effect that these techniques have on the circuit's critical-path delay.

3.1. Assumptions

To support our interpretation of incremental-tracing, a number of simplifying, but realistic, assumptions have been made. Firstly, we have assumed that the embedded memory blocks on an FPGA can be configured as a trace-buffer without requiring any additional control circuitry, as this would need to be implemented using soft-logic resources located in its vicinity. At a minimum, each memory block would need to support a ring-buffer configuration, whereby any signals connected to it would be continuously recorded until the chip is halted, and to also support a method for the trace contents to be efficiently extracted. We believe that these two requirements will incur a negligible overhead with respect to the size of each memory block — a wrap-around incrementing address counter can be efficiently implemented using T-type flip-flops, and existing IP solutions already exist to allow a memory's contents to be accessed using the JTAG interface [3], or through device readback techniques as in [15].

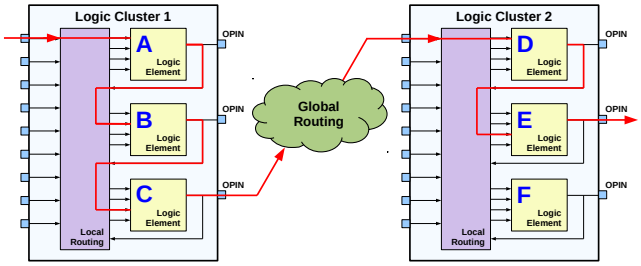


Fig. 3: Example signal path; logic elements A & B and D & F can be incrementally swapped for greater trace flexibility

Secondly, we have assumed that external triggering is used to determine when the trace-buffer stops (and possibly even starts) recording. This trigger may be driven by an off-chip signal that halts the clock, or manually inserted by the user on-chip, perhaps using other incremental-synthesis techniques. Alternatively, it may also be possible to implement simple triggering logic using fixed-circuitry that can be amortized over several memory blocks.

4. INCREMENTAL-TRACING OPTIMIZATIONS

The second contribution of this paper is the proposal of several CAD techniques that can be used to improve incremental-tracing. Because incremental-tracing differs from incremental-synthesis where the application is functional circuit modification, a number of new opportunities exist for CAD optimizations to improve runtime and routability.

A key property of this incremental-tracing problem is that all trace-buffer inputs on the FPGA can be considered to be logically equivalent. Unlike user designs, making ECO changes, or increasing fault tolerance, there is no hard requirement that nets must reach a specific sink for it to be considered a valid routing solution — reaching *any* free trace-pin regardless is sufficient for the signal to be observable. To take advantage of this, an incremental-tracing algorithm can be modified to search for any unoccupied trace-pin, and exit as soon as one is found. Besides having implications for routing runtime, this modification enables tracing flexibility to be significantly increased.

An orthogonal opportunity also exists from taking advantage of the logical symmetry of elements within an FPGA logic cluster. At a high level, the soft-logic fabric within an FPGA consists of a 2D grid of logic clusters, each of which contains N logic elements connected through a local routing network, as illustrated in Figure 3. Here, a circuit path is shown passing through three logic elements in logic cluster 1, and then two elements of logic cluster 2. In this path, there are three *local* nets emanating from logic elements A, B and D, and one *global* net from C. Should a designer wish to observe net C, the algorithm would then connect any part of the global route between its source and sink to a trace-buffer.

Circuit	6LUTs	FFs	FPGA Size	I/O	Logic Clusters	Multipliers	Memories	Nets
mkSMAdapter4B	1977	983	18x18	400/576	**199/234**	0/8	5/9	2329
or1200	2963	691	25x25	**779/800**	298/475	1/18	2/12	3483
mkDelayWorker32B	5580	2491	42x42	1064/1344	560/1302	0/50	**41/42**	7439
LU8PEEng	21954	6630	59x59	216/1888	**2583/2596**	8/98	45/72	27657
mcml	99700	53736	119x119	69/3808	**10436/10591**	30/435	38/285	106555

Table 1: Benchmark summary (values in bold indicate the constraining resource)

However, should a designer wish to observe any of the local nets A, B or D, then the problem becomes significantly harder; here, the incremental tool can only connect to the dedicated cluster output pins corresponding to that particular net. By taking advantage of the fact all logic elements are fully symmetric within a logic cluster — due to the local routing being a fully-populated crossbar (i.e. any input can be multiplexed onto any of its outputs) — the algorithm can change the ordering of the logic elements within a cluster without affecting circuit functionality. Hence, elements A and B can be re-ordered, as can D and F. The benefit of this technique is that the incremental tool now has the flexibility of forming a trace-buffer connection with any of the local cluster outputs. This technique is only applicable to logic elements that generate local nets used exclusively inside a logic cluster — elements which connect to the global network cannot be re-ordered otherwise its global route will no longer be valid. For incrementally-tracing local nets, we have modified our algorithm to treat all local OPINs as logically equivalent sources, and allow Pathfinder to arbitrate their access.

5. METHODOLOGY

Whilst it is possible to move existing placed-blocks or routes in order to increase the feasibility of incremental-tracing (this option is taken by Altera's SignalTap II), in this work, we explore the more constrained case where this is not allowed. Specifically, we incrementally route signals in a placed-and-routed circuit solution to a spare trace-buffer input, without affecting *any* part of the original circuit — without moving any placed blocks, nor ripping up any existing routes. This constraint is especially relevant during debug, where it is not likely that the original circuit will change between debugging iterations — only the additional instrumentation circuitry will — until the design error has been identified and a fix is to be applied. Furthermore, designers may not wish to modify any part of their original design for fear of changing the very behaviour of the bug they are trying to hunt down, such as those of an non-deterministic nature.

In order to explore our incremental-tracing techniques, the VPR 6.0 FPGA CAD tool, made available as part of the VTR project [17] was extended to support an incremental-tracing flow. This new, non-intrusive flow makes *absolutely* no modifications to the initial packing-placement-routing algorithms, instead only operating on the final circuit solution produced by VPR (in fact, our incremental-tracing flow can read in and start from a pre-routed solution).

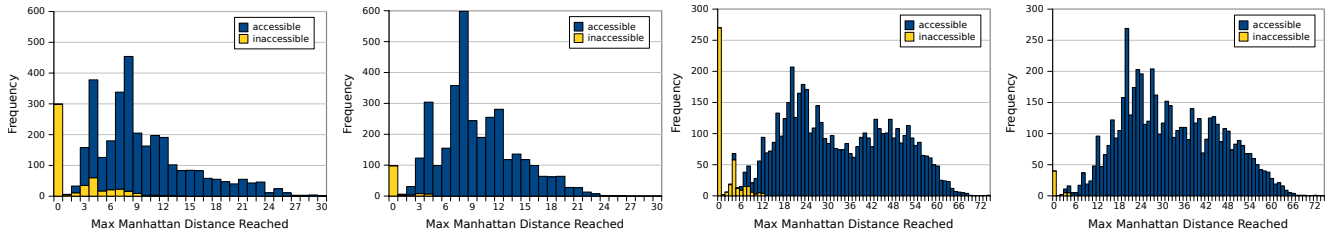
From this final solution, breadth-first search routing is performed from each of the nets selected for tracing (expanding from all points on the existing net) until any free memory pin is reached. Unlike the original routing stage, a directed search algorithm is not used due to this flexibility of needing to reach only one-of-many sinks. Similarly, the search window is not constrained within any sort of bounding box to maximize routability. Although we do not allow any of the existing user-routes to be ripped-up and re-routed, we do however allow incremental routes to negotiate for the most critical resources using the standard Pathfinder algorithm. By recognizing that incremental-routes need not negotiate with any of the user routing, a loss-less optimization can be applied: during breadth-first expansion, only routing-resource nodes that are not already fully occupied by the original solution is added to the routing priority-queue. In this work, we have used the heterogeneous benchmark circuits included in the VTR suite, which are summarized in Table 1.

6. RESULTS

6.1. OPIN Accessibility: Individual Signals

Due to the existence of symmetry between the logic elements inside a cluster that was described in Section 4, we can run experiments to determine the fraction of all used cluster output pins (OPINs) in a circuit that can be individually connected to a trace-buffer using incremental techniques. We term this metric its "accessibility". Figure 5 shows the accessibility values for four different benchmark circuits when mapped to identical FPGAs with varying channel widths (normalized to the minimum value possible: W_min), averaged over 10 different placement seeds. Channel width (W) is a measure of an FPGA's routing capacity: larger W indicate a wider and more flexible network for connecting logic, but occupies more silicon area. Each circuit was mapped to the default VTR heterogeneous FPGA architecture with representative parameters of logic cluster size N=10, look-up table size K=6 (non-fracturable), channel segment length L=4, and cluster input and output flexibilities of Fc_in=0.15 and Fc_out=0.1.

First, VPR was run through the entire (timing-based) packing-placement-routing flow to find its minimum channel width via binary-search, and for each subsequent channel width, the minimum value was inflated by the specified



(a) `or1200`: W_min (left), W_min +20% (right)

(b) `mkDelayWorker32B`: W_min (left), W_min +20% (right)

Fig. 4: Histogram of the maximum manhattan distance reached during breadth-first search

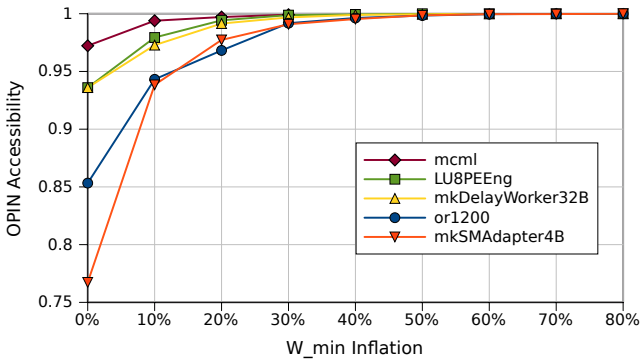


Fig. 5: Net accessibility with channel width

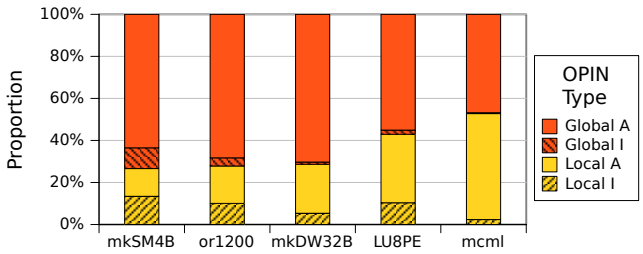


Fig. 6: Distribution of local and global OPINs, accessible (A) and inaccessible (I)

amount and the circuit re-routed using the identical circuit placement. Interestingly, it can be seen that even at a circuit's minimum channel width, over 75% of all OPINs can still be incrementally-traced, with this fraction rising with channel width and circuit size. The nets (and their OPINs) that exist within a placed-and-routed design can be divided into two categories: local and global. The proportion of each type is shown in Figure 6, sub-divided into those that are accessible (A) and inaccessible (I). From this chart, it can be seen that local OPINs dominate the majority of failing cases. The reasoning for this trend is intuitive: global OPINs connect onto the global interconnect, making it possible to tap *any* point on the existing route to branch-off to a trace-pin. No such luxury exists for local OPINs, however, as there is far less flexibility for a trace connection to be made.

This is reinforced by Figure 4, which shows a histogram of the maximum manhattan distance reached during breadth-

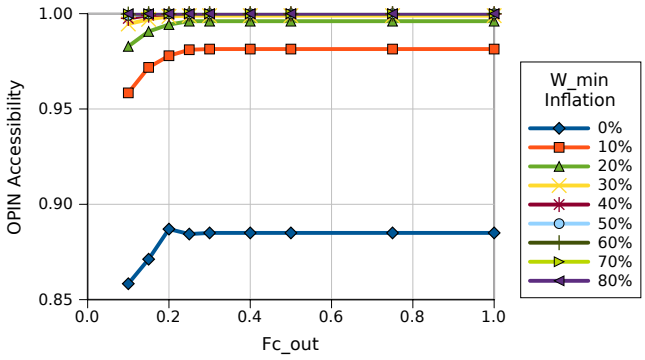
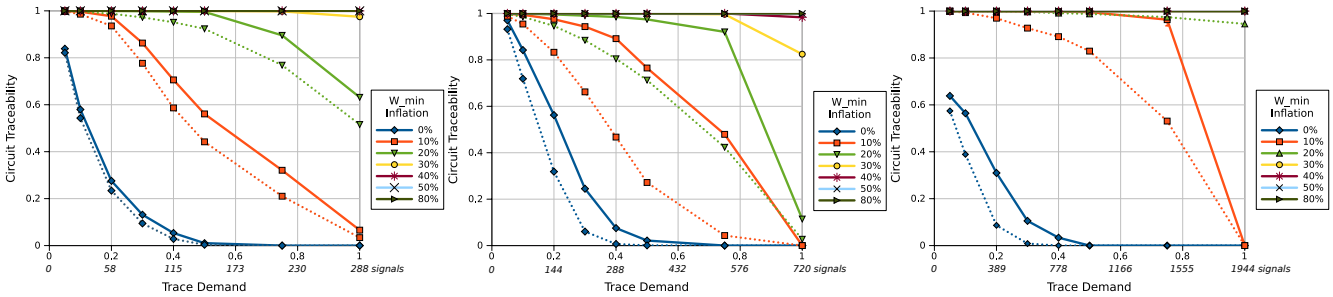


Fig. 7: Net accessibility: effect of Fc_out and channel width

first search routing, distinguishing between accessible (indicating the distance of the successful route) and inaccessible OPINs (the maximum distance reached before giving up, due to an empty priority queue). As seen, the vast majority of inaccessible OPINs possess a distance of zero which indicates they are unable to routed out onto the global network. Even though we have only shown the two circuits in which this finding is most prominent, this trend holds for all the other circuits we investigated.

This conclusion explains our previous observation in Figure 5 that increasing channel width increases a design's accessibility: by increasing the routing flexibility in exiting the cluster, as well as in easing any routing congestion around the cluster. Given this to be the problem, another key architecture parameter that determines the cluster output flexibility is Fc_out: the fraction of tracks that an OPIN can connect to. The results of varying this parameter, in combination with the channel width, again averaged over 10 placement seeds for each of the four circuits `mkSMAdapter4B`, `or1200`, `mkDelayWorker32B` and `LU8PEEng` is shown in Figure 7. As expected, increasing either parameter improves accessibility. In the case of the Fc_out parameter, this gain flattens out in the region of Fc_out=0.3 — the rationale here is that it doesn't matter how many tracks each cluster output pin can reach, if all of those tracks are occupied then the trace will remain unrouteable.

At this point, the most straightforward solution for improving accessibility appears to be to increase the FPGA channel width. Using an area model developed from the



(a) `mkSMAdapter4B` (1977 LUTs) (b) `or1200` (2963 LUTs) (c) `LU8PEEng` (21954 LUTs)

Fig. 8: Circuit traceability: effect of trace demand and channel width; *dotted lines show no LE re-ordering*

detailed transistor-level delay model described in [18], we have been able to estimate that for the four circuits explored, tolerating a 10% area increase will result in an average accessibility of 0.9907, whilst sacrificing 20% will return a 0.9997 payoff. For full accessibility, the area penalty is almost 30%. It is worth noting, at this point, that the minimum channel width is only an artificial metric used to measure the routing demands of a circuit. Physical FPGAs can only be manufactured with a fixed channel width, and in this and previous work [19], it has been observed that in commercial architectures the routing resources and flexibility are sufficiently over-provisioned to ensure routability even with the most stubborn circuits.

6.2. Circuit Traceability: Multiple Signals

In the previous subsection, we looked at the feasibility of incrementally-tracing one signal at a time. Let us now consider the more realistic case where multiple signals are connected to trace-buffers at the same time. We term the number of signals that are observed simultaneously to be the trace demand, which is normalized to the maximum number of free memory pins available in the FPGA. For each of the 10 placement seeds, 1000 signal selections were randomly chosen for each trace demand to achieve a sufficiently large sample size to draw conclusions from. In order to observe the effect of congestion-related failures only, we have restricted the OPINs eligible for selection to those that are known to be independently accessible from the previous subsection. This, however, does not restrict the designer to only being able to select the fraction of signals corresponding to its "accessibility" value as in Figure 5; this metric actually represents a lower-bound of signals that can be selected due to the LE re-ordering technique described previously. Should a particular combination of signals be un-traceable, a designer can fallback onto a lengthy re-compile of the whole design.

We term the fraction of successfully traced routes (where the *entire* signal selection is simultaneously connected to a trace-pin each) over the 1000 random selections across the 10 placement seeds, to be its "traceability". If a circuit fails to route, this is due solely to congestion that cannot be re-

Trace Dmnd (FFs)	W_min Inflation +0%	+10%	+20%	+30%
0.05 (889)	887 (6)	888 (5)	889 (8)	889 (7)
0.10 (1778)	-	1777 (8)	1778 (10)	1778 (9)
0.20 (3557)	-	3555 (15)	3557 (18)	3557 (17)
0.30 (5335)	-	5332 (23)	5335 (33)	5335 (32)
0.40 (7114)	-	-	7114 (49)	7114 (50)
0.50 (8892)	-	-	8892 (152)	8892 (133)
0.75 (13338)	-	-	-	13338 (1864)
1.00 (17784)	-	-	-	-

Table 2: `mcml` (53736 FF) signals incrementally-traced (runtime in seconds; c.f. placement runtime = 19507s)

solved within 50 Pathfinder iterations, if at all. The results in Figure 8 show this traceability value for selection sizes indicated on the x-axis. The scale of the x-axis is normalized to the trace capacity of each circuit, which varies for each circuit due to the number of free memory-blocks that exist in each physical implementation, which is specified in Table 1. We have omitted the traceability results for the memory-constrained circuit `mkDelayWorker32B`, which have high traceability values exceeding 0.98 for all trace demands, even at minimum channel width. In this architecture, we have elected to configure each memory-block at its widest configuration which enables each to sample up to 72 signals simultaneously to fully stress the routing fabric. For these results, Fc_out has also been fixed at a moderate value of 0.2.

Our results show that incremental-tracing is very much feasible, even when channel width is only inflated by a small amount. For a 20% channel width increase (approximately 1.1X area) there is a greater than 90% probability that a signal selection utilizing 75% of the available memory capacity of the FPGA can be fully traced. Further increasing the channel width to 40% above minimum allows *all* of the free memory on the FPGA to be utilized for debug. The difference between the dotted and solid-lines represent the improvements gained by our LE re-ordering optimization.

To further validate this conclusion, we have successfully traced an intelligently selected signal set (using the method described in [19]) from the largest circuit at our disposal:

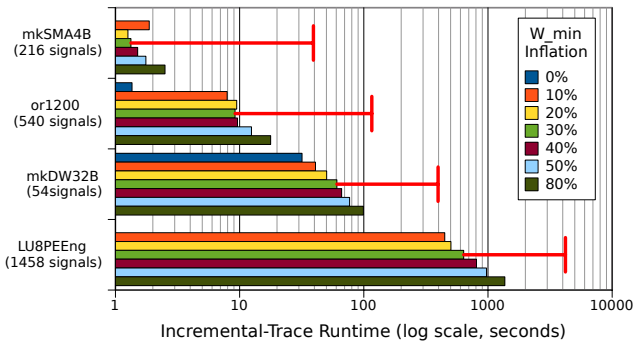


Fig. 9: Incremental-trace runtime for trace demand = 0.75 (red error bar indicates original placement runtime)

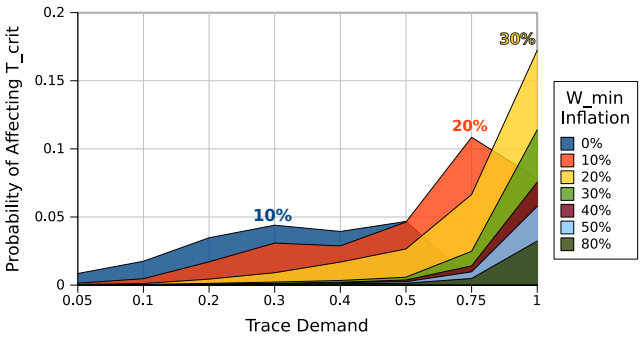


Fig. 10: Average probability of incremental-trace affecting critical path delay

mcml. Table 2 shows that at least 50%–75% of the trace-capacity can be fully utilized at W_min +20%–30%, when inaccessible global OPINs are ignored as previously. We have found these to be surprising results, and a testament to the flexibility of FPGAs devices.

6.3. Effect on Flow Runtime and Circuit Delay

Two additional performance metrics that designers care about during FPGA debug is the effect of instrumentation on the compilation flow runtime and the circuit delay. Figure 9 shows the incremental-tracing runtime for a successful route when utilizing 75% of the available trace capacity. To put this into perspective, the red error bars have been used to indicate the original VPR placement runtime, which was the lengthiest portion of the full compilation flow when the binary-search for the minimum channel width is omitted. As these results show, incremental-tracing can be much more computationally efficient than a full re-compilation, even for large circuits. Currently, we are using the default Pathfinder routing parameters, which are tuned to routing user-logic. We anticipate that, by tuning these parameters to penalize congestion sooner, the runtime can be further improved.

During prototyping, it is anticipated that the operating frequency would not be a primary concern given that the instrumented FPGAs will remain many orders of magnitude faster than software simulation; with incremental-tracing, designers can now sacrifice some circuit speed for accelerated debugging productivity. Figure 10 charts the average probability that a successful incremental-trace will increase the critical-path delay, across all 10 placement seeds for each circuit. As channel width increases, this fraction reduces due to the easing of any routing congestion in the circuit, allowing trace connections to take a more direct route. As trace demand increases, the number of routes affecting T_crit also increases due to a higher contention for trace-pins. We have found that when the critical path is affected, this increase is on average only 3.4%, 0.6% and 3.6% for mkSM, or1200 and mkDW respectively, with a maximum of 15%, 1.4% and 8% over all trace demand and signal selections. LU8, the largest and logic-constrained benchmark showed a 0.5% maximum increase.

6.4. Future Work

We believe that these results show that there is much promise in using incremental-synthesis techniques for debug. Even though these techniques are very much feasible even on current architectures, there remains a number of interesting avenues for future work to explore. Firstly, there is a huge scope for investigating architectural changes to improve routing flexibility beyond, and at a lower cost than, those described by the current set of parameters, such as W or Fc_out, and increase its accessibility and traceability.

Alternatively, there are also many opportunities for improving the CAD algorithms to achieve this. Routing using a breadth-first search strategy is known to be more computationally intensive than a directed approach, which was not used in this work due to the opportunity for any trace-pin to be a feasible sink. However, it may still be possible to accelerate this incremental-tracing procedure by prioritizing regions for the router to first explore. Looking even further, it would also be possible to integrate incremental-placement techniques into this trace flow: opening the door to the on-demand pipelining for trace signals to minimize its effect on timing (as the latency of a trace signal does not affect its observability), improving routing flexibility by passing through empty BLEs, or even allowing the insertion of complex trigger circuitry.

7. CONCLUSION

In this paper we have explored the limitations of applying incremental synthesis techniques to the insertion of trace-buffers for improving on-chip observability. During incremental-tracing, there exists far more flexibility (and opportunity) than with functional design changes — a traced signal is not constrained to reaching one particular sink for example, but

instead, it need only reach any free trace-pin to achieve a feasible routing solution.

By using the academic Verilog-To-Routing toolchain, we first observed that even at the most constrained case of minimum channel width, at least 75% of cluster output pins can be connected to a trace-buffer input using incremental techniques. Although less than 50% of all nets are locally absorbed inside a logic cluster, we discovered that this category made up the majority of the inaccessible cases, and that within those, most signals were unable to even route out from its cluster. By inflating the minimum channel width, W, and varying the cluster output flexibility, Fc_out, we found that our accessibility metric can be improved to 99.1% for a 10% area overhead, and 99.9% for 20% area.

Subsequently, we found equally promising results for tracing signals *concurrently*: 75% of the available trace-capacity can be utilized for a 10% area overhead, with the full capacity available for 20% area. Furthermore, we show that incremental-tracing can be significantly more efficient than a full circuit re-compilation, even for large circuits, and that under these same conditions for 10% area, the critical path will be increased on average 7% of the time, by 2%.

We believe that these results show incremental-tracing to be a highly feasible reality, especially given that we have stressed our techniques on the most extreme use-case. Real-world designs will likely not be implemented using minimum-sized FPGA dimensions, nor will its logic be packed for maximum density as VPR does. Such circuits will not be mapped to a minimum routing channel width either; we have seen some evidence that commercial FPGAs do over-provision their routing tracks, which provide sufficient slack for incremental-tracing. Furthermore, because incremental-tracing is used only to accelerate the debugging flow, we can see any tool that implements our techniques operating on a "best-effort" basis — one which may try and trace as many signals as it possibly can, and should that not be acceptable, a designer can always fallback onto a lengthy, full re-compilation run. We anticipate that, with the ability to trace the majority of all sequential or combinational nodes in the circuit and to combine these values with signal restoration techniques, this may be a very rare eventuality indeed.

8. REFERENCES

[1] ITRS, "International Technology Roadmap for Semiconductors, 2007 Edition: Design," http://www.itrs.net/Links/2007ITRS/2007_Chapters/2007_Design.pdf, February 2008.

[2] 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.

[3] Altera, "Quartus II Handbook Version 11.1 Vol. 3: Verification," http://www.altera.com/literature/hb/qts/qts_qii5v3.pdf, November 2011.

[4] 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.

[5] Tektronix, "Certus: ASIC Multi-FPGA Prototyping Debug Suite," http://www.veridae.com/index.php/products/clarus-prototyper.html, 2012.

[6] 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.

[7] 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.

[8] Y. S. Iskander, C. D. Patterson, and S. D. Craven, "Improved Abstractions and Turnaround Time for FPGA Design Validation and Debug," in *FPL'11, Proceedings of the 2011 21st International Conference on Field Programmable Logic and Applications*, 2011, pp. 518–523.

[9] 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 Trans. on*, vol. 28, no. 2, pp. 285–297, 2009.

[10] R. Y. Rubin and A. M. DeHon, "Timing-Driven Pathfinder Pathology and Remediation: Quantifying and Reducing Delay Noise in VPR-Pathfinder," in *FPGA'11, Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2011, pp. 173–176.

[11] S. Dutt, V. Shanmugavel, and S. Trimberger, "Efficient Incremental Rerouting for Fault Reconfiguration in Field Programmable Gate Arrays," in *Computer-Aided Design, 1999. Digest of Technical Papers. 1999 IEEE/ACM International Conference on*, 1999, pp. 173–176.

[12] O. Coudert, J. Cong, S. Malik, and M. Sarrafzadeh, "Incremental CAD," in *Computer Aided Design, 2000. ICCAD-2000. IEEE/ACM International Conference on*, 2000, pp. 236–243.

[13] J. Emmert and D. Bhatia, "Incremental Routing in FPGAs," in *ASIC Conference 1998. Proceedings. Eleventh Annual IEEE International*, Sep 1998, pp. 217–221.

[14] S. Trimberger, *Field-Programmable Gate Array Technology*. Norwell, MA, USA: Kluwer Academic Publishers, 1994.

[15] P. Graham, B. Nelson, and B. Hutchings, "Instrumenting Bitstreams for Debugging FPGA Circuits," in *Field-Programmable Custom Computing Machines, FCCM'01. The 9th Annual IEEE Symposium on*, March 2001, pp. 41–50.

[16] E. Keller, "JRoute: A Run-Time Routing API for FPGA Hardware," in *Parallel and Distributed Processing*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2000, vol. 1800, pp. 874–881.

[17] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson, "The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing," in *Proc. of the 20th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2012, pp. 77–86.

[18] E. Hung, S. J. E. Wilton, H. Yu, T. C. P. Chau, and P. H. W. Leong, "A Detailed Delay Path Model for FPGAs," in *FPT 2009, International Conference on Field-Programmable Technology; Sydney, Australia*, December 2009, pp. 96–103.

[19] E. Hung and S. J. E. Wilton, "Speculative Debug Insertion for FPGAs," in *FPL 2011, International Conference on Field Programmable Logic and Applications; Chania, Greece*, September 2011, pp. 524–531.