# Incremental Trace-Buffer Insertion for FPGA Debug

Eddie Hung and Steven J. E. Wilton

***Abstract*—As integrated circuits encapsulate more functionality and complexity, verifying that these devices operate correctly under all scenarios is an increasingly difficult task. Rather than use traditional verification techniques such as software simulation, more and more designers are taking advantage of the significantly higher clock speeds that can be achieved by using Field-Programmable Gate Array based prototypes. A key challenge to these prototypes is the lack of on-chip observability during debug; one popular solution is to insert trace-buffers into the design to record a limited set of internal signals, but modifying this trace configuration often requires the entire circuit to be recompiled. In this work, we propose that the original circuit mapping is fully preserved and incremental techniques are used to eliminate the need for a full recompilation, thereby accelerating the debug process. By exploiting two opportunities available during trace-insertion: the ability to connect from any point of a signal to any trace-pin, and the internal symmetry of the FPGA architecture, we find that incremental trace-insertion can be 98X faster than a full recompilation, return a routing solution with a shorter wirelength, and have a negligible effect on the critical-path delay of the original circuit when reclaiming 75% of the leftover memory capacity for tracing.**

***Index Terms*—Design verification, Field-Programmable Gate Array (FPGA) debug, Trace-Buffer, Incremental Compilation**

## I. Introduction

With current state-of-the-art integrated circuits now reaching multi-billion transistor counts, designing these complex devices is far from trivial. In many cases, verifying that a circuit functions correctly under all expected (and unexpected) operating conditions is often even harder than the initial design phase. To combat this, many designers have turned to Field-Programmable Gate Array, or FPGA, -based prototyping to increase their verification coverage beyond that achievable using traditional software simulations. A study by IBM found that a full chip-level testing using a multi-FPGA prototype was 100,000x faster than software simulations, and 400x slower than the fabricated ASIC; in context, for every one second required to boot Linux on the ASIC, almost 5 years were required in simulation, but only 7 minutes on their FPGA system [1].

Circuits can be prototyped on FPGA platforms in a fraction of the time and cost required for a fabrication spin, with the result able to reach clock frequencies many orders-of-magnitude higher than in simulation. This allows designers to explore circuit behaviour that would otherwise be beyond reach. Despite these advantages, the primary challenge for debugging on these devices lies with on-chip observability. That is, whilst simulators can provide full visibility into all the intermediate signals of a circuit, for physical prototypes, normally only the signals that interact with the outside world would be observable using external equipment. One common solution to this problem is to instrument the prototype using trace-buffers to record a subset of internal signals into on-chip memory for subsequent analysis; tools such as ChipScope Pro, SignalTap II and Certus can be used for this task [2], [3], [4], [5].

By embedding trace-buffers into a circuit, designers can transparently, and in real-time, sample a small window of the internal circuit values into on-chip memory during regular device operation. Once recording is complete, designers can then extract this trace data onto a computer and view them as waveforms, just as with a simulation flow; the key difference being that these traces can be captured from much further into the device operation than would be feasible to achieve in a simulator. This concept is illustrated by Figure 1. Crucially, the choice of which signals to connect to a trace-buffer must be made before the device is tested. Observing a different set of signals requires a different set of connections to be made, which often requires a new circuit mapping to be constructed.

In this work, we explore the effect of using incremental synthesis techniques in order to reduce both the time needed to perform the initial instrumentation of a circuit, and the turnaround time between debugging iterations where designers wish to modify the signals connected to these trace-buffers. Rather than discarding the original circuit mapping, incremental techniques aim to make the minimum set of modifications on this preserved mapping in order to implement the requested changes. The advantages to this are many: besides being able to achieve significant CAD runtime savings, the modified circuit will retain most of the original mapping result and allow designers to debug something as close to the uninstrumented circuit as possible, as well as the ability to better preserve any low-level optimizations and timing closure.

Specifically, we propose that trace instrumentation is inserted directly into the post place-and-routed FPGA circuit mapping — without moving any existing logic blocks nor ripping up any of the existing routing — instead, using only the spare resources that were originally left unused. To make this feasible, we present novel techniques to increase the incremental routing

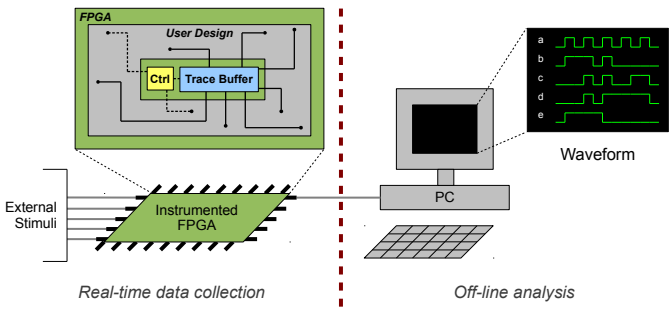



Fig. 1. Trace-based debugging

E. Hung and S. Wilton are both with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Canada; e-mail: {eddieh,stevew}@ece.ubc.ca

flexibility of trace-buffer insertion. We show the value of our techniques by comparing the effect of instrumenting before (pre-map insertion) and during the FPGA mapping procedure (mid-map) with doing so incrementally (post-map).

The key contributions of this paper is that it presents novel techniques to rapidly accelerate the turnaround time between FPGA debugging iterations by employing a form of incremental compilation specific to trace-buffer insertion. Unlike existing, general-purpose incremental compilation methods, our techniques are optimized for observing (as opposed to modifying) an existing circuit, which allows designers to add or change the signals connected to their trace-buffers up to 98X faster than a full recompilation when utilizing 75% of the available trace-buffers, assuming 20% routing slack exists.

To evaluate this work, we have made a complete and detailed comparison between our proposed post-mapping insertion technique and pre-map trace insertion, which requires the entire circuit to be recompiled, and mid-map insertion, which recompiles only part of the circuit. When targeting an architecture based on the Altera Stratix IV 40nm family of devices supporting heterogeneous RAM and DSP blocks, along with realistic wire delays, we find that our proposed technique returns a circuit solution with a shorter wirelength than the pre-map and mid-map strategies, with only a small (average at 0.6%) but stable impact on critical-path delay. Furthermore, we find that our work can outperform the general-purpose incremental-compilation feature present in Altera Quartus II.

An earlier version of this work investigating just the post-map technique was first presented in [6]. Since that paper, we have further improved our CAD optimizations and re-evaluated our work on a more realistic FPGA architecture using a bigger set of benchmarks and compared against pre-map and mid-map trace insertion methods, as well as with a commercial tool.

## II. Background

### A. 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 cost (84% area and 20% delay overhead [7]). However, some modern FPGAs typically have hardware support (device readback) to read not only the value of its configuration bits, but also all the state bits in a design. These techniques can typically offer simulator-like visibility into all 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 1, 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 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 recompiled. Choosing which signals to observe can be a time-consuming task, though a number of automated techniques have attempted to combat this [9], [10].

### B. Incremental Synthesis

The key idea behind incremental synthesis is to allow the functionality of a fully place-and-routed circuit implementation to be modified whilst preserving as much of the original solution as possible. Within the context of FPGAs, the aim is to move the fewest number of placed blocks, and rip-up and re-route the fewest number of existing nets to achieve this result. Owing to the general-purpose nature of FPGAs, this is a much more feasible task than in custom ASICs, as FPGA CAD tools have the flexibility to use any of the prefabricated logic or routing resources that were not employed in the original circuit. The motivations behind this technique are many: to minimize recompilation effort during the design phase, to preserve timing closure when undertaking Engineering Change Orders (ECOs), or for improved fault and defect resilience [11]. However, the penalty for incremental synthesis is often regarding to be a small loss in circuit performance.

Incremental synthesis techniques for FPGAs are not new however [12], [13], nor is their application in design prototyping and debug [14]. In this work, we refer to the use of incremental techniques for transparently inserting trace-buffers as incremental-tracing. Many of the techniques cited are targeted at modifying the user circuit for functional purposes and go beyond the requirements for observe-only incremental-tracing (which are explained in detail in a following section) and include provision 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 supported 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]. We benchmark our work against this in Section VI.

### C. Multi-FPGA ASIC prototyping

The main application that this work is most applicable, and one where functional debug is crucial due to simulation being

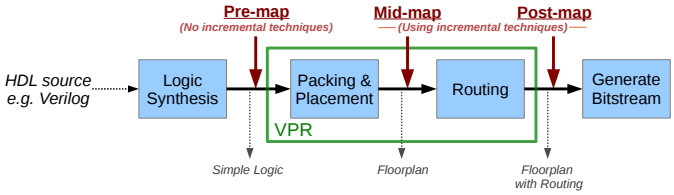



Fig. 2. Pre-, mid-, post-map stages of the FPGA compilation flow

impractical, is ASIC prototyping. In this use case, large ASIC circuits will often not fit onto a single FPGA device and will typically have to be partitioned (manually) over multiple boards each containing multiple FPGAs. The key issue during partitioning is the number of inter-FPGA connections that can be made — and given that FPGA devices have a limited number of I/O pins (Altera's largest Stratix IV device supports 1104 user I/Os) — this is often the constraining factor which prevents devices from being fully utilized, nor maximum clock frequency being reached [17]. This slack — both in terms of memory and routing, as well as with regards to timing, represents a unique opportunity for a non-intrusive debug technique.

The Dini Group, who manufacture multi-FPGA boards for ASIC prototypes, currently assume that (and suggest to) their customers that each FPGA is filled only to 60% utilization [18], whilst Intel reported that they were able to build a highly-tuned prototype of their Nehalem (Core i7/i5) processor over 5 FPGAs, with a maximum logic utilization of 89% (average 84%) and memory utilization of 83% (average 55%). Importantly, they reported that due to I/O limitations, time-division multiplexing of connections was required which limited their user clock frequency to 520 KHz [19]. A study by an industrial and academic collaboration [20] states that very high (>85%) logic utilization is rare due to pin limitations, and that the average post-partitioning utilization they have observed in the field is less than 50%. Despite this, we do not believe that our techniques are exclusive to the ASIC prototyping space; in Section VI, we show that our methods degrade gracefully when the amount routing slack is reduced.

## III. Trace-Insertion

Current FPGA trace solutions such as Xilinx ChipScope Pro, Altera SignalTap II, Synopsys Identify and Tektronix Certus [2], [3], [4], [5] all operate primarily on the pre-mapped circuit. That is, these tools will instrument the original user circuit with trace-buffers and their connections before place-and-routing the combined design, although several of these tools can also support a limited amount of post-mapping reconfiguration. A simplified illustration of what we have defined to be the pre-map, mid-map, and post-map stages of the FPGA compilation flow is shown in Figure 2. In this section, we will first describe the assumptions that were made, before elaborating on the differences between performing FPGA trace-insertion at each of these stages in order, before arriving at the main focus of this paper: post-map trace insertion.

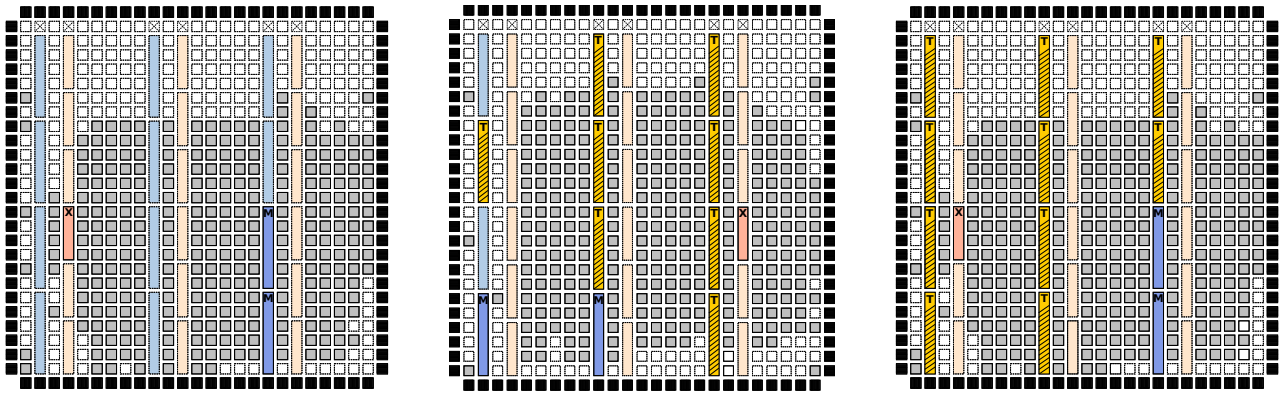
### A. Assumptions

In this work, we have made a number of simplifying assumptions for trace-based debugging. Firstly, we do not consider any overheads incurred by triggering logic. Because only a limited subset of signals can be connected to trace-buffers, and a limited window for which their signal values can be recorded, triggering logic allows designers to control when to start and stop tracing (for example, only in the clock cycles immediately surrounding the occurrence of the error) in order to make the most effective use of this finite memory capacity. One scenario where this assumption would be realistic is if this trigger event was driven by an external source (perhaps off-chip) that is used to halt the clock signal. Alternatively, this trigger logic may be inserted manually into the circuit using soft-logic, perhaps using more general-purpose incremental synthesis techniques. We believe that any trigger-logic would require far fewer routing wires (for example, triggering on a status flag, a state machine or a bus address) than for the trace-buffer connections, and would hence have a small impact on the circuit. Another option would be for such trigger logic to be implemented using fixed-circuitry as opposed to soft-logic; doing so would make it transparent to the user-circuit. The area overhead of this hard-logic can be reduced by amortizing it over several trace-buffers (for example, one trigger block per memory column).

A second simplifying assumption that we have made is the ability for free memory resources to convert into trace-buffers without the need for any additional control circuitry. We believe this is also realistic, as commercial devices allow memory blocks to be operated as wide shift-registers which can then be used to record a sliding window of signal data. The second requirement to enable this feature is the ability to unload the signal data once tracing was complete: we believe that this can be achieved by using existing IP solutions for low-bandwidth access over the JTAG interface [3], or through using device readback techniques [15].

The third assumption is that, due to our CAD tools, we are only able to synthesize (and hence instrument) circuits operating in a single clock-domain. For trace-based debug to support multiple clock-domains, each observed signal must be sampled by a trace-buffer operating in its clock-domain. We believe that our methods can be extended to support this by adding these requirements as additional synthesis constraints.

We note that adding trace-instrumentation to the circuit after logic synthesis, where the circuit has already been transformed from a high-level description (such as Verilog) into low-level FPGA primitives (lookup-tables) means that the designer is restricted to only observing *gate-level* signals. These gate-level signals, through logic optimizations and technology-mapping, may not have a direct correspondence to the original HDL signals. We believe several approaches exist to alleviate this mismatch: firstly, unless register re-timing is performed, both commercial and academic CAD tools are able to preserve the HDL-to-gate mapping for flip-flops in the circuit. Designers can therefore use these elements as fixed points of reference into their Verilog code, or to use the data collected for off-line simulation to compute all intermediate, combinational signals. Secondly, designers are able to manually specify additional points of reference by using synthesis attributes to force the CAD tool to maintain this HDL-to-gate correspondence: `(* syn_keep *)` is supported by Synplify and Quartus II



(a) Baseline uninstrumented circuit

(b) Instrumentation *without* incremental techniques (pre-map; utilizing 8/10 free memories)

(c) Instrumentation *with* incremental-techniques (mid-/post-map; utilizing 10/10 free memories)

Fig. 3. Placement results when instrumenting the or1200 benchmark with identical signals: `X` = multiplier, `M` = memory, `T` = trace-buffer (shaded)

tools, whilst ISE users can apply the `S` (`SAVE_NET`) attribute to do so. Implicitly, existing trace IP such as ChipScope Pro, SignalTap II and Certus already do this when instrumenting a circuit pre-synthesis.

Lastly, during ASIC prototyping, where typically I/O is the limiting factor and spare logic resources may be abundant, it may be feasible to optimize the circuit less aggressively so that more combinational signals can be preserved for instrumentation without requiring a larger FPGA or impacting circuit delay. This approach is not dissimilar to debug for software applications, where performance is often traded for visibility in debug binaries; in fact, the latest version of GCC 4.8 supports a new "`-Og`" optimization level which addresses the need for fast compilation and a superior debugging experience [21].

### B. Pre-Map Trace Insertion

Performing trace-insertion at the pre-map stage involves instrumenting the user circuit with trace IP early in the implementation flow, whilst the design is still described at a high-level of abstraction, and is the primary mode of operation in many of the existing trace solutions. For example, Xilinx ChipScope Pro allows instruments to be instantiated manually into the HDL source, or inserted directly into the synthesized circuit (but still prior to the place-and-route mapping procedure) using the Xilinx PlanAhead and ChipScope Pro Core Inserter tools [2]; whilst the Tektronix Certus product automatically modifies the HDL source to add all necessary trace infrastructure [5]. There are many advantages to this approach: by operating on the circuit early in the implementation flow, any trace IP can be described at an equally high-level which allows for increased portability across device families (and even FPGA vendors). Furthermore, because the instrumented circuit is treated as a single entity by the subsequent CAD stages, theoretically, the circuit can be globally optimized as a unit to create a more efficient result.

However, there are also several downfalls to instrumenting the circuit prior to physical mapping. Firstly, because this method inserts additional logic into the original user circuit, the CAD tools will need to work harder in order to place-and-route the circuit — not only because there exists more objects to solve for, but also because of the additional constraints that they impose. For example, each user net that the designer wishes to observe introduces at least one additional fan-out (the trace-buffer input) for the placement and routing stages to consider. This increased complexity can manifest as increased compilation runtime. Although prior work found that tracing 10% of the signals in a large design using SignalTap II incurred only a 10% increase to runtime over the uninstrumented baseline [22], this problem is compounded by the need to perform a full recompilation *every time* the designer wishes to modify the observed signal set.

Secondly, due to the chaotic and unpredictable nature of the heuristic algorithms used in CAD tools, the very act of modifying the circuit (even by a little) may alter, or even hide, the bug under investigation. Rubin and DeHon found that small perturbations just in the routing stage of the VPR CAD tool caused the critical-path delay to vary between 17–110% [23]; hence, it is not implausible to imagine a scenario whereby instrumenting the erroneous circuit would cause the faulty path to be implemented entirely differently, one in which the bug was much more difficult (if not impossible) to reproduce. Whilst this point may not apply to strictly-functional bugs (i.e. those that are caused solely by designer errors in the HDL source) this may be of critical importance when attempting to locate non-deterministic timing faults such as those introduced by underspecified timing constraints or multiple clock-domains.

Figure 3 illustrates the difference in the placement results between the original, uninstrumented circuit in Fig. 3a and the circuit instrumented pre-mapping in Fig. 3b, when using the VPR tool [24]. Square blocks on the peripheries indicate I/O blocks, whilst the square blocks in the centre of the diagram represent logic clusters: a dark shading means that the block is occupied. The columns of rectangular blocks interspersed in the logic fabric represent heterogeneous resources: each `X` indicates a used multiplier, whilst `M` represent a used memory block. In the instrumented circuit, `T` indicates a free memory block that has been converted into a trace-buffer. A clear difference exists between the placement results before and

after pre-map instrumentation: the act of instrumentation has significantly affected circuit placement, with the original user memory (annotated with M) have now been pushed away from the centre of the circuit, with trace-buffers (T) taking its place. This effect is due to the CAD algorithms being unable to differentiate between the existing memory blocks and any of the newly inserted trace-buffers, and so optimizes for them equally.

### C. Mid-Map Trace Insertion

An opportunity for trace-insertion also exists mid-mapping, in which trace instrumentation is inserted part-way through the FPGA mapping procedure; specifically, as illustrated in Figure 2, between the placement and routing stages of the compilation flow. This approach offers a compromise between pre-map and post-map trace-insertion — the original packing and placement of the circuit is left untouched (which, as revealed in Section VI, comprises the majority of the compilation runtime).

Between the placement and routing stages, we propose that the trace-buffers are incrementally-placed into the unoccupied memory resources of the FPGA — initially similar to the post-map approach — but with the difference that both the original and instrumented net connections are subsequently routed in a combined fashion. Hence, mid-map trace-insertion can be expected to offer a trade-off between the quality-of-results gained by pre-map insertion and the fast runtime achievable using post-map insertion. Because the existing routing is ripped-up and re-routed, there still remains the possibility that any timing-bugs may also be obscured with this approach.

### D. Proposed Technique: Post-Map Trace Insertion

The final opportunity for applying trace instrumentation, and forming the main focus of this paper, is for trace support to be added at the end of a normal FPGA compilation flow, an approach that we have termed post-map insertion. Here, only at the very end of the compile flow would incremental techniques be used to make the minimal set of changes required to accommodate the trace instruments. This allows designers to preserve as much of their circuit as possible, in order to improve CAD runtime for a small loss in circuit quality.

Besides pre-map trace-insertion, Altera SignalTap II also supports a post-map flow in which the entire instrumentation procedure can be completed using the general incremental-compilation features available in the Quartus II tool [3]. Through experimentation, we have discovered that Quartus II appears to take a best-effort approach to inserting trace IP — in many cases, it can preserve over 99% of the circuit's original placement and routing (though this is not guaranteed as for our techniques) for moderate runtime savings — this is quantified in Section VI-F. The Synopsys Identify product, however, provides an incremental flow specifically for allowing designers to quickly modify and re-route the observed signal set but only once the design has been successfully instrumented [4].

In this work, we make the guarantee that during post-map trace-insertion, *all* placement and routing of the original circuit will be preserved: all new trace instruments must be incrementally inserted using only the fully-buffered routing resources that were previously unoccupied. We believe this is an important requirement for FPGA debug, and one that is necessary in order to minimize the possibility that timing-bugs will be altered, or even obscured, by the act of instrumentation. Due to trace-instrumentation being overlaid on top of and without affecting the existing user-circuit, an interesting side-effect is that as soon as the debug infrastructure is no longer required (for example, in a production bitstream) it can simply be ignored and the circuit run back at its original clock frequency prior to any instrumentation. This can help preserve timing-closure in a sign-off circuit.

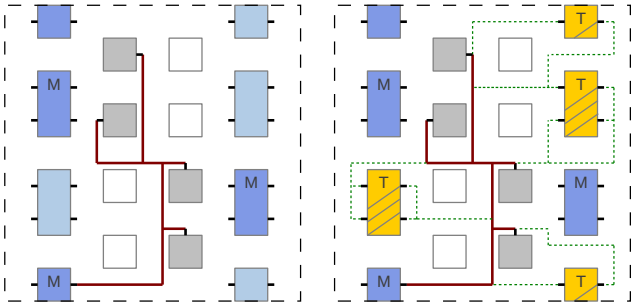
Of course, imposing the strict constraint that none of the user circuit can be modified can cause the newly instrumented circuit to no longer be routable, or require a larger FPGA; this overhead is quantified in Section VI. Figure 3 shows the difference between no instrumentation (Fig. 3a) and post-map instrumentation (Fig. 3c). Unlike pre-map previously, instrumenting the circuit after placement means that the original circuit placement is unaffected; additional blocks (and routing, which isn't shown) are mutually exclusive to the resources used in the original result.

### E. Framework

This work applies our techniques to the open-source VPR 6.0 FPGA mapping tool, which is part of the academic Verilog-To-Routing (VTR) project, version 1.0 [24]. We employ the VTR suite to synthesize our Verilog benchmarks into a technology-mapped BLIF netlist, which is fed into VPR for timing-driven packing, placement and routing for mapping onto a custom FPGA architecture. For routing, VPR employs PathFinder: a popular FPGA routing algorithm which allows nets to temporarily over-use routing resources, which is then iteratively resolved to allow access to only the most timing-critical nets in a process termed negotiated congestion [25].

Using this tool, circuits are mapped to an FPGA architecture based on the Altera Stratix IV device, with a cluster-size N=10, look-up table size K=6 (fracturable into two K=5 LUTs) and channel segment length L=4. The cluster input flexibility Fc_in=0.15, and the cluster output flexibility Fc_out=0.1. The targeted architecture is heterogeneous and based on the Stratix IV 40nm family with support for RAM and DSP blocks, as well as realistic wire delays; advanced architectural features such as inferring carry-chains and shift-registers for the user circuit are not currently supported. However, given that these optimizations are used primarily to improve area and delay-efficiency, we believe that integrating these in the future would not affect the conclusions presented. For example, PLL/DCM clock synthesis and distribution often use their own dedicated set of FPGA resources and hence will not affect the user circuit.

The routing architecture employed is a modern, unidirectional fully-buffered network, which means that adding extra fan-out loads to existing nets will not affect the original circuit timing. The dedicated memory resources of this architecture can be configured either as a 72 bits wide by 2048 entries deep shift-register — that is, each memory block can be configured as a trace-buffer capable of recording a sliding window of 72 signals for the last 2048 cycles — or as 36x4096, 18x9182




(a) Original FPGA layout (b) Layout with potential trace-connections

Fig. 4. Illustration of the many-to-many routing flexibility available in post-map insertion: solid red lines indicate user-routing, green dashed lines indicate potential trace-connections of which only one is sufficient

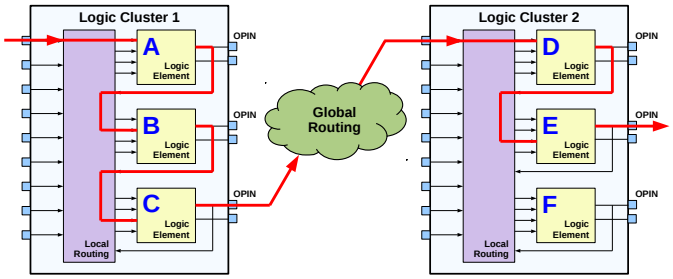



Fig. 5. Example signal path; logic elements A & B and D & F can be incrementally swapped for greater trace flexibility

or 9x18194. In this work, we have chosen the widest memory configuration of 72x2048 to investigate the limits of applying the maximum amount of pressure onto the routing interconnect.

## IV. Incremental CAD for Post-Map Tracing

In order to effectively implement post-map incremental trace insertion, we have developed a number of CAD optimizations to take advantage of the unique nature of this problem.

### A. Many-to-Many Trace Flexibility

During post-map trace insertion, one rather unique opportunity not previously available when mapping the original user-logic exists: a selected signal need only be incrementally-routed to *any* free trace-buffer input pin for its signal values to be observed. This differs from user-logic in that signals do not need to be connected to all of its sinks in order to create a valid routing solution; for incremental-tracing, by treating all inputs-pin of all trace-buffers of the FPGA as potential sinks, a connection to any pin is sufficient to allow observability.

Figure 4 illustrates this concept — any trace-pin connection along the dashed routes (or along any other combination of routes not shown) will be sufficient. Recall that, due to our assumptions, we can convert every unused memory block into a trace-buffer. In addition, for nets which utilize the fully-buffered global interconnect such as that shown, any point of the net can be tapped to make this connection. This many-to-many capability provides two advantages: significantly improved routing flexibility and CAD runtime, both of which are especially important given the self-imposed constraint that during post-map insertion, we prevent any existing user-routing from being ripped up. Routing algorithms can then be modified to search for any trace-pin and finish as soon as one is found. These algorithms are commonly coupled with PathFinder with which our techniques are also compatible.

### B. Logic Element Symmetry

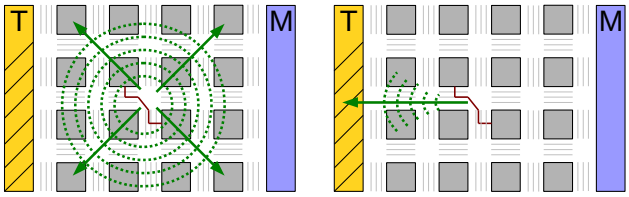
For local nets, which are entirely absorbed within a logic cluster and do not venture out onto the global interconnect, an additional optimization can be applied. Figure 5 illustrates an example FPGA signal path: the nets connecting Logic Element A to B, B to C, and D to E, are all local as they do not exit the cluster. On the other hand, the connection from C to D is global. Tracing local nets is possible by tapping its cluster output pin (OPIN) which would otherwise be unused, and forming a new global connection to a trace-buffer. Because the local routing inside logic clusters is formed of a fully-populated crossbar in which any cluster input can be switched to any logic element input, logic elements within FPGA logic clusters can be considered symmetric. This observation allows an incremental CAD tool to reorder certain logic elements inside a cluster without affecting the functional or timing behaviour of the original user circuit, but only those driving local nets. Logic elements driving global nets cannot be swapped, as doing so would cause the global net to be driven by a different OPIN for which a new global-routing solution would also be necessary. In Figure 5, logic elements A & B, and D & F, can be swapped as they both drive local nets, whilst logic elements C or E cannot be moved. However, extra care must be taken if those logic elements contain fractured LUTs.

The ability to swap logic elements allows greater routing flexibility in tracing local nets, which can compensate somewhat for the lack of any existing presence on the global interconnect from which a trace connection can tap off. Experimental data on this phenomenon can be found in [6]. Any routing algorithm can therefore be modified to treat all local logic elements as sources, and perform routing expansion from any of them. PathFinder can then be used to iteratively arbitrate between multiple trace-connections until a valid solution (where each OPIN is used only once) is found.

### C. Timing-Driven Directed Search

In previous work [6], we pursued a breadth-first search routing strategy (with the two optimizations described previously) during post-map incremental-tracing in order to maximize circuit routability. Even though breadth-first search is an exhaustive algorithm, we found that incremental-tracing was still an order of magnitude faster than the original circuit placement. In this work, we improve on this result with a timing-aware directed search technique, which is able to route slightly fewer signals, but provide better timing, all for much lower computational effort.

With the optimizations described in the previous subsection, the nature of the problem is now a one- or many-to-many routing search. For this reason, it is now unclear what the target of any directed-search algorithm should be; previously,




(a) Breadth-first trace search [6] (b) Directed trace search

Fig. 6. Illustration of breadth-first and directed search routing strategy

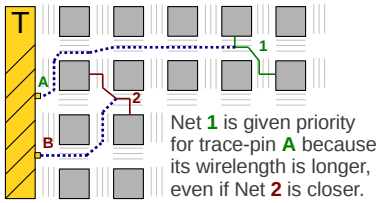



Fig. 7. Heuristic for pre-assigning *suggested* targets during directed search

with breadth-first search, the algorithm can expand outwards from any part of the existing net and end as soon as any trace sink is reached, as illustrated in Fig. 6a. However, this can require high computational effort, as the CAD tool will need to exhaustively search all routing resources in an expanding wavefront — even if they are unlikely to lead to a trace-buffer.

A more efficient routing algorithm would first explore the routing resources that are most likely to lead to a valid solution, yet not forget about the less likely resources in case the preferred resources are congested. By default, the VPR CAD tool adopts a directed search approach, such as that shown in Fig. 6b. However, a key difficulty in adopting the directed strategy used in routing user-logic during tracing is that there is not one or more unique targets that any route must connect to. Instead, during tracing, we only require that any free trace-buffer input is reached in order for the signal to be observed. Conveniently, because incremental-tracing works on top of a complete and legally routed circuit, the timing slack of each signal to be traced is fully known in advance — this value can be used to adjust its priority over any congested trace-buffer and routing resources to minimize its timing impact.

To achieve this, we developed a heuristic to preassign a *suggested* trace-buffer input for each selected net. The heuristic works by first sorting all nets by their decreasing manhattan distance to the nearest available trace-buffer and weighting this by its timing slack, after which the nets furthest away are allocated their first-choice trace-buffer, as in Figure 7. When any trace-buffer is full, all nets are re-sorted according to the remaining buffers and this procedure repeats. The objective of this algorithm is to suggest trace-buffers that minimize the post-placement wirelength of all trace-signals, whilst considering their timing criticality; importantly though, signals do not have to connect to their suggested input — due to the many-to-many trace flexibility optimization, signals can connect to any input that can be more easily reached.

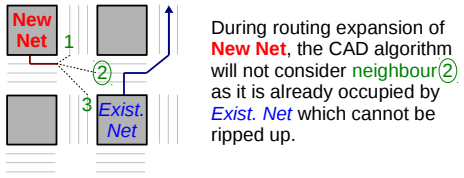



Fig. 8. Incremental-tracing neighbour expansion: consider free routing resources only (1 and 3) as existing user-net cannot be ripped up

### D. Neighbour Expansion

A final, but small, optimization that we make is to reduce the search space during incremental-routing. During post-map insertion, because we do not allow any existing user-routes to be ripped up, we can improve incremental-routing efficiency by preventing resources that are already fully utilized from ever being considered. This is achieved by modifying the neighbour-expansion routine of PathFinder, which is responsible for adding new routing resources to the priority queue of candidates to search, to only add those resources that have free capacity as illustrated in Fig. 8. In addition, we keep track of the utilization from user-routing and incremental-tracing separately, thereby allowing the routing algorithm to assign capacity only to the traces that need it the most. Essentially, we have subtracted all existing routes of the user circuit from the routing resource graph, and treat the graph that remains as an entirely new routing problem.

## V. Methodology

In this work, we have looked at the effect of tracing 100 random signal selections, which consume between 5% and 100% of the leftover memory capacity in eight different heterogeneous circuits, each placed using 5 different seeds across six different channel widths, for a total of 192,000 data points per insertion strategy. The details of these circuits are shown in Table I, where the $W_{min}$ column represents the minimum channel width of the baseline, uninstrumented circuit (explained in the following subsection) and the traceable nets column represents the number of gate-level nets, both combinational and sequential (including all RAM and DSP outputs) that can be connected to a trace-buffer. For DSPs that employ inputs/output registers, we can observe their values by tracing the input net before it enters the DSP, and its output net after exiting. These benchmark circuits are supplied with the VTR flow [24] and represent realistic, sizable, designs which include an open-source processor core, or1200, a matrix decomposition core, LU8PEEng, and the largest circuit at over 100,000 LUTs, a Monte Carlo hardware simulator, mcml. The number of heterogeneous DSP and RAM resources used by each benchmark can also be found in Table I. Although we could apply automated signal selection techniques to these circuits, such as those described in [26], [10] to trace only the most influential signals in the circuit, we decided instead to take multiple random samples of signals to gain an understanding of our techniques when applied to any signal that a designer may wish to observe.

| Circuit | 6-input LUTs | FFs | FPGA Size | $W_{min}$ | I/O | Logic Clusters | DSP Blocks | RAM Blocks | Traceable Nets | Max Trace-Buffer Inputs |
|---|---|---|---|---|---|---|---|---|---|---|
| or1200 | 3054 | 691 | 25x25 | 90 | **779/800** | 258/475 | 1/18 | 2/12 | 3807 | 720 |
| mkDelayWorker32B | 5590 | 2491 | 42x42 | 94 | 1064/1344 | 468/1302 | 0/50 | **41/42** | 7918 | 72 |
| stereovision1 | 10290 | 11789 | 36x36 | 118 | 278/1152 | **866/936** | 38/45 | 0/30 | 16653 | 2160 |
| LU8PEEng | 22634 | 6630 | 54x54 | 136 | 216/1728 | **2175/2255** | 8/91 | 45/63 | 29001 | 1296 |
| stereovision2 | 29943 | 18416 | 84x84 | 184 | 331/2688 | 2338/5208 | **213/231** | 0/154 | 47882 | 11088 |
| bgm | 32884 | 5362 | 64x64 | 150 | 289/2048 | **2987/3072** | 11/128 | 0/80 | 37639 | 5760 |
| LU32PEEng | 76211 | 20898 | 101x101 | 200 | 216/3232 | **7470/7575** | 32/325 | 150/208 | 97563 | 4176 |
| mcml | 101858 | 53736 | 95x95 | 164 | 69/3040 | **6680/6745** | 30/276 | 38/180 | 113994 | 10224 |

TABLE I
HETEROGENEOUS BENCHMARK SUMMARY, UNINSTRUMENTED (VALUES IN BOLD INDICATE THE CONSTRAINING RESOURCE)

*1) Pre-Map Insertion:* To implement pre-map trace insertion, we directly modified the synthesized BLIF netlist to add one additional sink — a trace-buffer pin — for each signal selected for observation. A unique single-port RAM slice was instantiated for each selected signal using the `.subckt` construct, with the responsibility for packing each of these 1-bit slices left to VPR. Currently, VPR packs to minimize resource utilization, and hence returns the result seen in Fig. 3b where only the minimum number of memory blocks are used.

*2) Mid-Map Insertion:* In mid-map insertion, the packing and placement of the circuit has been computed, but not any of its routing. At this stage, the number, and location, of all free memory blocks (which can be transformed into trace-buffers at no cost) are also known. The challenge here is to ensure that each of the selected signals is allocated to one available trace-pin, before routing can commence. This is achieved using the same heuristic as described in Section IV-C, where nets are iteratively assigned their nearest trace-buffer input based on their manhattan distance, but without timing information. The key difference here, besides mid-map insertion requiring the entire circuit to be re-routed from scratch, is that these signals must be connected exactly to their assigned pins for the circuit to be deemed legal.

*3) Post-Map Insertion:* For post-map insertion, we allow VPR to complete its entire packing-placement-routing process unmodified, before performing any incremental-tracing. Even though the constraint to prevent user-routing from being moved appears to create a more restrictive problem, this is countered by the additional flexibility enabled by the techniques described in Section IV. For this work, we use the timing-driven directed search algorithm described previously, and remove the bounding-box search window (i.e. the router considers resources that do not lie on a shortest-path) in order to maximize routability. In addition, we have increased the overuse penalty factors used by the PathFinder routing algorithm (experimentally, a reasonable set of values were found to be `--first_iter_pres_fac 10` and `--initial_pres_fac 15`) so that routing congestion is penalized more heavily, and also restrict the number of incremental-routing iterations to 5 rounds after which all conflicting trace-nets are sequentially discarded until a legal solution remains.

| Circuit | Average Pins/Net | Avg. Routing Utilization | Average Wirelength |
|---|---|---|---|
| or1200 | 3.7 | 39% | 21.1 |
| mkDelayWorker32B | 3.2 | 30% | 22.9 |
| stereovision1 | 2.4 | 45% | 15.6 |
| LU8PEEng | 4.6 | 47% | 28.9 |
| stereovision2 | 2.5 | 33% | 29.0 |
| bgm | 4.7 | 45% | 29.5 |
| LU32PEEng | 4.8 | 45% | 40.1 |
| mcml | 3.4 | 41% | 26.9 |

TABLE II
BENCHMARK SUMMARY, UNINSTRUMENTED (AT $W_{min}$+20%)

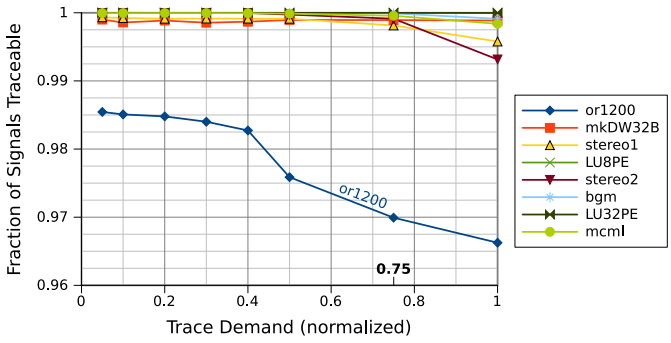
| | stereo2 | LU32PE | mcml |
|---|---|---|---|
| Device | EP4SGX110 | | |
| LAB (logic cluster) utilization | 33% | 89% | 99% |
| Average routing utilization | 8% | 35% | 35% |
| Peak routing utilization | 21% | 67% | 47% |

TABLE III
ROUTING UTILIZATION WHEN MAPPED ONTO ALTERA STRATIX IV.

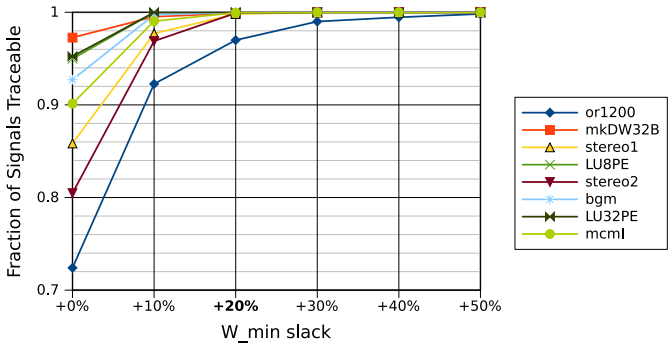
### A. Routing Slack of Minimum Channel Width $W_{min}$+20%

A result that has commonly been used as a metric for routability is the minimum number of FPGA tracks — or channel width — required to implement a circuit. A smaller channel width is desirable as it means a more optimized implementation which requires a smaller FPGA area (and hence cost) to realize. Whilst the minimum channel width ($W_{min}$ is an important metric for measuring circuit routability during FPGA architecture and CAD research, it is however, not realistic nor relevant when targeting real FPGAs, which contain a prefabricated channel width that the CAD tool must not exceed.

In order to eliminate channel width as an independent variable from the experiments that follow, we have opted to map each circuit to an FPGA architecture with $W_{min}$+20% routing tracks of the uninstrumented case. The average routing utilization at of these circuits are shown in Table II and average to 41%. Whilst routing at $W_{min}$ represents the absolute best-case of routing-efficiency possible, it would be expected that FPGA vendors would provision some additional slack in their devices to cope with stubborn circuits within a reasonable runtime. Due to the proprietary and closed nature of commercial CAD tools, we are unable to find the exact channel width that are employed on these FPGAs, nor to make an exact comparison with the VTR flow; however, we are able to




(a) Signals traceable as a function of trace-demand, at $W_{min}$+20%




(b) Signals traceable as a function of $W_{min}$ slack, at trace demand = 0.75

Fig. 9. Average fraction of signals traceable using post-map insertion

infer a relative amount of routing slack by mapping our same benchmark circuits to a set of similar parts. The results of these experiments are shown in Table III. Even in the worst case for a mid-sized circuit, the *peak* routing utilization by Quartus II v12.1 is only 67% — when not searching for the minimum utilization possible. The average routing utilization for all three circuits is lower on these Altera devices than in our theoretical architecture; hence, we believe that assuming a FPGA implementation with 20% routing slack above the best-case $W_{min}$ is reasonable.

## VI. Results

### A. Signals Traceable using Post-Map Insertion

The objective of post-map insertion is to add trace-buffer connections incrementally on top of an existing design — without re-placing or re-routing any of the user circuit. Under this constraint, it may not be possible to connect all selected signals to a trace-buffer due to unresolvable routing congestion, either between the existing user circuit and the new trace connection, or between two new connections, where no solution exists. To prevent the router from searching for such impossible solutions, we force post-map trace insertion to run for five PathFinder iterations to resolve any congestion, after which we iteratively discard illegal trace-nets (i.e. those which have routing resource conflicts) until a legal solution is found.

The fraction of signals that can be traced using post-map techniques, as a function of the trace-demand — the number of signals selected for tracing — is shown in Figure 9a. The results show an expected trend: requesting more signals for tracing means that a smaller proportion (but still a higher absolute number) of them can be successfully traced. However, even in the worst case when all left-over memory blocks from the smallest circuit are reclaimed as trace-buffers, over 96% of all requested signals can still be successfully connected. Figure 9b shows how the number of signals traced varies with the amount of channel width slack. Intuitively, the more slack that exists, the less the routing congestion and the more signals that can be connected. Importantly, our techniques are able to degrade gracefully even when there is little routing slack — this allows post-map insertion to operate on a circuit routed at minimum channel width, when pre-map and mid-map techniques would fail to route. From Fig. 9b, it appears that increasing the channel with slack produces diminishing returns. The "knee" of the curve appears to lie at either 10% or 20% slack, with further estimates using an FPGA area model [27] also indicating that the best traceability-per-area is achieved when 0%–20% routing slack exists.

### B. CAD Runtime

Figure 11 shows the CAD runtime (on a log scale) for each of the different stages of physical mapping, across all benchmark circuits. The results are broken down into the runtime for the individual packing, placement and routing stages of the flow at a fixed channel width of $W_{min}$+20%. The number of signals selected for each circuit is fixed at 0.75 of the leftover memory capacity. As expected, pre-insertion incurs a runtime penalty over the baseline case in which the circuit has not been instrumented, followed by mid-insertion, and post-insertion, both of which utilize past results and are faster than a full recompilation. Incidentally, the stereovision2, bgm and mcml circuits instrumented pre-map were unroutable at our assumed 20% routing slack. Excluding those, on average, post-map insertion was 98X faster than pre-map insertion, and 22X faster than mid-map insertion.

For pre-insertion, the circuit is instrumented prior to mapping and hence all three VPR stages — packing, placement, routing — must be re-run. Inherently, this instrumented circuit will be more complex than prior to instrumentation, rendering a more difficult (and a more constrained) problem for the CAD algorithms to solve. During mid-insertion, the original circuit's packing and placement results are re-utilized, on top of which the trace-buffers are incrementally placed and connected to the traced nets. Here, the two most computationally expensive parts of the mapping flow — packing and placement — do not need to be re-executed, leaving only the routing stage.

For the proposed technique, post-insertion, this is taken one step further: results from all three VPR stages are re-utilized (the bar shown for tpack represents the time required to load the pre-packed netlist into the CAD tool). This time, trace routes are incrementally connected to a trace-buffer without affecting this previous routing by using only the routing resources that were not used in the original circuit mapping. This has the effect of reducing the solution space as compared to a complete re-route, and coupled with the CAD techniques described in Section IV where any connection to any trace-pin will be sufficient, results in a more efficient algorithm.

Figure 10 compares the runtime (shown on a linear scale) of all three trace-insertion strategies when applied to the

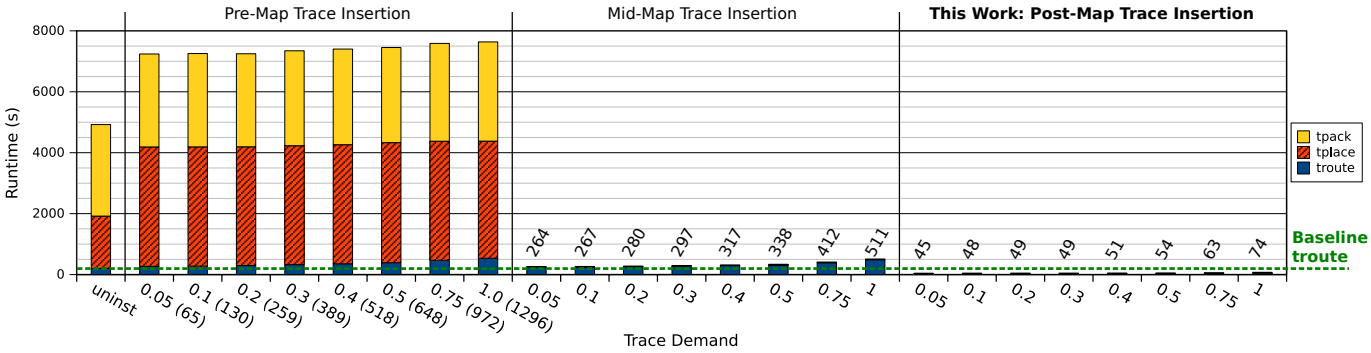



Fig. 10. Runtime breakdown, as a function of trace-demand, for benchmark LU8PEEng only

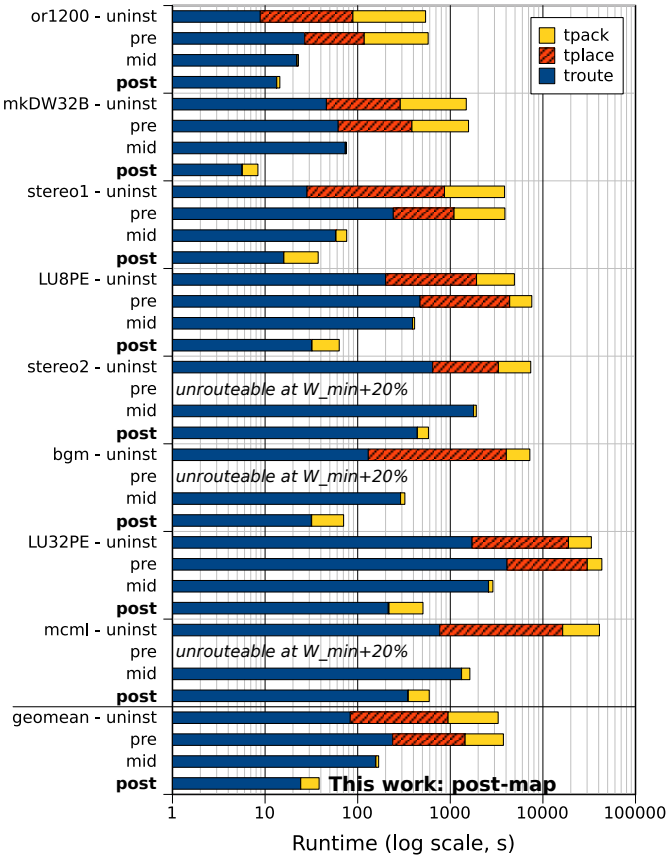



Fig. 11. Runtime breakdown (at trace-demand = 0.75, shown on a log scale) across all circuits

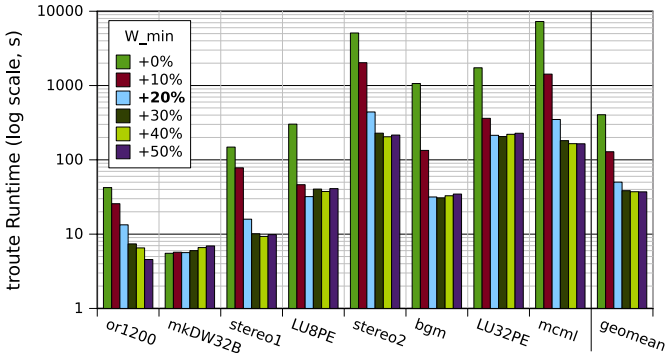



Fig. 12. Post-map routing runtime (log scale, at trace-demand = 0.75) as a function of channel width, across all circuits

LU8PEEng circuit, across all trace-demand values from 0.05 (65 signals) to 1.0 (1296 signals). For all three strategies, it can be seen that increasing the trace-demand (and hence the complexity of the mapped circuit) increases runtime for each of the mapping stages. In all cases, post-map trace insertion is faster than a full re-route of the circuit, which is required for pre-map and mid-map insertion strategies.

Lastly, the sensitivity of post-map insertion runtime, as a function of the channel width slack, is investigated in Figure 12 (shown on log scale). Here, it can be seen that routing runtime is highly dependent on the circuit channel width. Despite the existence of two conflicting factors: an increase in the size of the solution space balanced against a reduction in routing congestion, as the channel width of a circuit increases, the latter factor dominates. Due to the directed-search routing strategy, the algorithm will only explore the routing resources that lead towards the suggested sink, and hence will not explore all of the additional resources provided by an increased channel width. Increasing the channel width though, will reduce the amount of routing congestion that the algorithm will need to resolve, leading to a net gain in runtime which flattens off at $W_{min}$+30% and beyond.

### C. Wirelength

The total wirelength, averaged across 100 random signal selections and 5 placement seeds, normalized to the uninstrumented case, is shown in Figure 14. Again, we fix the channel width to $W_{min}$+20%. This metric represents the number of FPGA routing resources that are utilized to implement the circuit; lower values indicate a more efficient circuit mapping, which can also lead to smaller routing delays. At first glance, the results shown in these charts are surprising: it would be expected that by instrumenting the circuit as early in the CAD flow as possible, the implementation can be optimized with more degrees of freedom and return the best result. For example, by completely re-placing the circuit, the algorithm may be able to situate some trace-buffers more centrally, allowing shorter connections to be made, as illustrated in Figure 3.

This does not appear to be the case here, however, as these results shows that in all of the benchmark circuits, those instrumented pre-mapping have the highest wirelength, followed by mid- and post-mapping. One explanation may be that due to the compound and heuristic nature of the

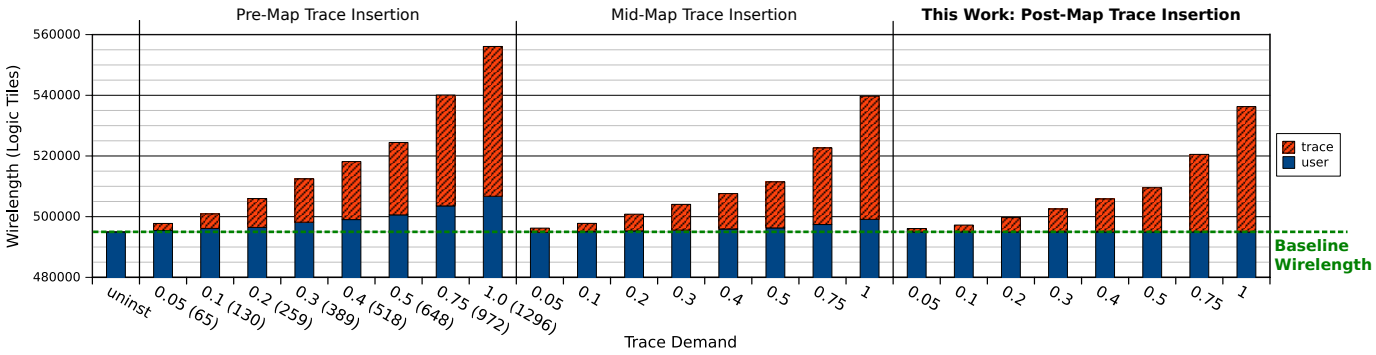



Fig. 13. Circuit wirelength breakdown, as a function of trace-demand, for benchmark LU8PEEng only

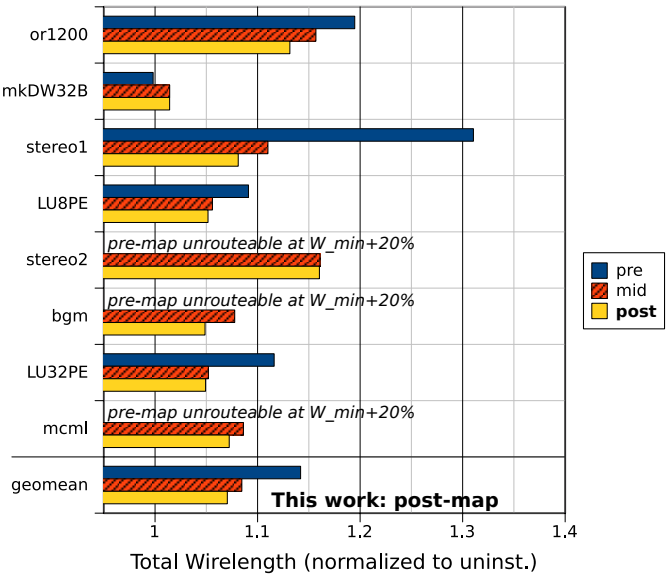



Fig. 14. Circuit wirelength, across all circuits at trace-demand = 0.75

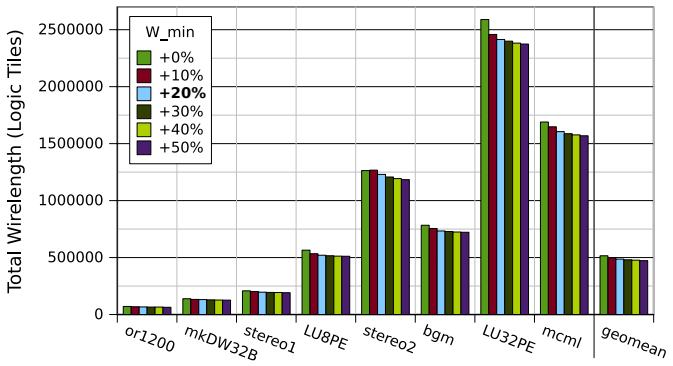


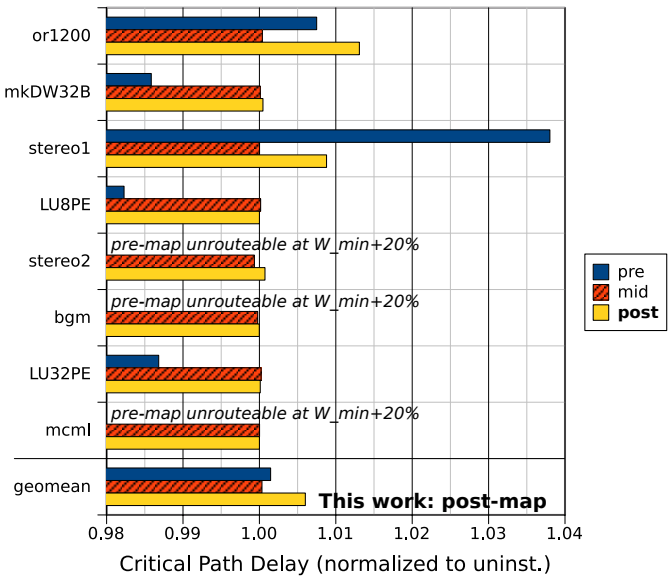

Fig. 15. Post-map circuit wirelength as a function of channel width, across all circuits at trace-demand = 0.75

CAD algorithms involved, more degrees of freedom may not necessarily translate into better results. An example of this is that during the earliest packing stage, VPR has very little information to decide which 1-bit trace-nets should be packed into the same 72-bit trace-buffer. In this way, in attempting to compromise between the user circuit and the new debug instrumentation (which, for the pre- and mid-insertion strategies, is indifferentiable by the CAD tool) it ends up making a poor global choice. Returning to the example in Figure 3, during placement, trace-buffers that are situated more centrally during pre-map insertion (Fig. 3b) have the inadvertent effect of moving the original memory blocks from the user circuit further away from the logic blocks that it must connect to. Whilst this can decrease the wirelength of any trace connections, it will also cause the length of the original user connections to increase, possibly in a way that causes the net total to increase.

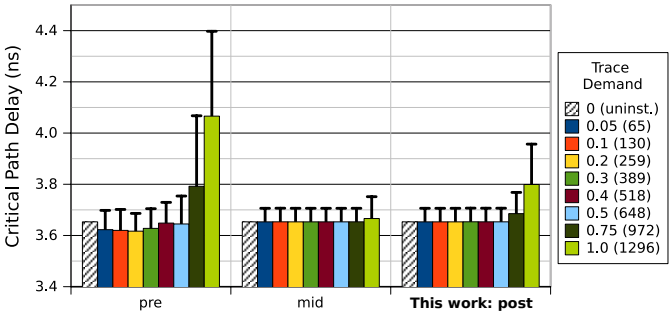
The wirelength for various trace-demands of the LU8PEEng circuit is shown in Figure 13. As expected, increasing the trace-demand also increases the total wirelength for all three insertion strategies. This chart shows that pre-map insertion returns the highest wirelength for both the user and incremental connections, regardless of the number of signals traced. These results also show that, in attempting to optimize for them both equally, the CAD tool is actually worse at both. Similarly, mid-map insertion also returns a larger result than at post-map; but this time, the user wirelength is affected by a much smaller amount. At trace demand 0.5 and below, post-map insertion returns shorter trace wires than mid-map insertion, and even at the maximum demand of 1.0, the post-map trace wirelength is still within 2% of its mid-map value. In all cases, post-map returns a shorter total wirelength. Lastly, the sensitivity of this metric to the amount of routing slack is presented in Figure 15, which shows that as the channel width increases, the instrumented wirelength decreases — due to the router being able to find more direct, uncongested, paths to each trace-buffer.

### D. Critical-Path Delay

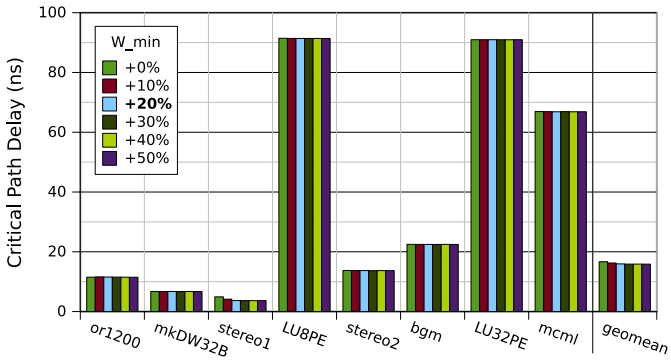
A metric that is important to many designers is the effect that instrumentation has on the critical-path delay of their circuit. Figure 16a measures the average critical-path delay of all three trace-insertion strategies, when normalized to the uninstrumented case, at a fixed channel width of $W_{min}$+20%. Despite the increase in wirelength observed in the previous subsection, for all three strategies, on average, there was a much smaller effect on the critical-path delay of each circuit. In the worst case, pre-map insertion increased the delay (and hence, decreased its maximum clock frequency) by 3.8%; however, even more interesting is that inserting trace-buffers actually improved the critical-path delay of three circuits:




(a) Critical-path delay, across all circuits at trace-demand = 0.75




(b) Critical-path delay for stereovision1 only (error bars indicate stdev)




(c) Post-map delay, as a function of channel width at trace-demand = 0.75

Fig. 16. Critical-path delay of instrumented circuits

mkDelayWorker32B, LU8PEEng and LU32PEEng, by 1.4%, 1.8% and 1.2% respectively. We believe this can be attributed to the chaotic nature of CAD algorithms, as reported by [23].

Whilst pre-map and mid-map insertion can both potentially return a smaller delay than the uninstrumented circuit, this is not possible for post-map insertion, where existing routing connections are never ripped up and re-routed for an even better solution. Our experiments found that mid-map trace-insertion returned a solution within 0.1% of the original delay. For post-map trace-insertion, this overhead was on average higher by 0.6%, which we believe is acceptable for our target application of ASIC prototyping.

| | Signals Traced | Runtime (s) | Wirelength | T_crit (ns) |
|---|---|---|---|---|
| **This work**: stereo0 | **1318** | **27** | **118569** | **3.61** |
| No dir-search [6] | 1322 | 38 | 117528 | 3.67 |
| No many-to-many | 1287 | 44 | 127175 | 4.48 |
| No LE symmetry | 1300 | 31 | 122172 | 3.73 |
| No n. expand opt. | 1318 | 28 | 118569 | 3.67 |
| No optimizations | 1311 | 97 | 129692 | 3.94 |
| **This work**: mcml | **7665** | **151** | **1603771** | **66.15** |
| No dir-search [6] | 7666 | 2802 | 1605644 | 66.15 |
| No many-to-many | 7666 | 256 | 1641781 | 66.15 |
| No LE symmetry | 7665 | 185 | 1621184 | 66.15 |
| No n. expand opt. | 7665 | 216 | 1611983 | 66.15 |
| No optimizations | 7664 | 3261 | 1650272 | 66.15 |

TABLE IV
POST-MAP CAD OPTIMIZATION BREAKDOWN

Figure 16b fixes the benchmark to the circuit with the smallest critical-path delay (and hence, the most sensitive to extra routing): stereovision1, and plots the effect that varying the number of signals traced has on its delay. Again, these results show that the effect is small for the mid-map and post-map techniques, up until the maximum trace-demand. This indicates that there is enough flexibility in the routing fabric to comfortably support incremental-tracing. Noticeably though, there are small perturbations in the delay when using pre-map trace-insertion, where the delay varies by a small and unpredictable amount with the number of signals traced. This effect is also supported by the error-bars which show a much higher variance for pre-map than for the other two strategies.

The impact of trace-insertion, as the circuit channel width varied, is shown in Figure 16c. Although the number of signals that can be successfully traced decreases with channel width (as explored in Section VI-A previously) the critical-path of the instrumented circuit remains stable.

### *E. CAD Optimizations for Post-Map Tracing*

Given the results thus far show that post-map trace-insertion is superior to doing so pre-map in terms of CAD runtime, circuit wirelength, and a more stable critical-path delay, and also superior to mid-map also in all aspects except for a slightly higher delay, an interesting question to explore is how each of the CAD optimizations described in Section IV contribute to these metrics. Table IV shows the contribution of each of the CAD optimizations for both a difficult-to-route circuit, stereovision0, and the largest circuit, mcml. These figures show that whilst using a undirected, breadth-first search routing approach can lead to more signals being traceable and a smaller wirelength due to the solution space being more thoroughly explored, this can have a huge impact on the runtime. Disabling the many-to-many flexibility, and the logic element symmetry optimizations both have the effect of reducing the number of signals that can connected to trace-buffers, whilst also increasing its critical-path delay and routing runtime. The neighbour expansion optimization gives a small increase in runtime without substantially affecting its other metrics.

| *mcml* | Trace Demand | Time (s) | Logic Elements | RAM Blocks | F_max (MHz) |
|---|---|---|---|---|---|
| Quartus II (Stratix IV EP4SGX180 device, peak routing utilization: 31%) | | | | | |
| Uninst. | - | 3514 | 52688 (75%) | 536 (56%) | 29.54 |
| Pre-map | 0.27 (4094) | 4131 | 65268 (93%) | 667 (70%) | 30.94 |
| | 0.52 (7676) | 4715 | 65859 (94%) | 767 (81%) | 29.91 |
| Post-map | 0.27 (4094) | 2451 | 64708 (92%) | 650 (68%) | 29.15 |
| | 0.52 (7676) | 2959 | 70251 (100%) | 750 (79%) | 29.50 |
| **This work**, at $W_{min}$+20% channel width | | | | | |
| Uninst. | - | 35526 | 66685 (99%) | 38 (21%) | 15.12 |
| Pre-map | 0.4 (4090) | 41908 | 66737 (99%) | 95 (53%) | 15.03 |
| | Above 0.4 | Circuit unrouteable at $W_{min}$+20% | | | |
| Post-map | 0.5 (5111) | 158 | 66685 (99%) | 180 (100%) | 15.12 |
| | 0.75 (7663) | 210 | 66685 (99%) | 180 (100%) | 15.12 |
| | 1.0 (10213) | 489 | 66585 (99%) | 180 (100%) | 15.12 |

TABLE V
COMPARISON BETWEEN QUARTUS II AND THIS WORK FOR MCML

### F. Comparison with Altera Quartus II

In this last subsection, we compare our techniques with Altera Quartus II's incremental-compilation feature, which can be achieved by designating the circuit as "post-fit" design partition. In this mode, the CAD tool will attempt to preserve all placement and routing. To instrument the circuit, we employed the Altera SignalTap II product which inserts the necessary trace-buffers, supporting logic, and connections to observe a set of user-specified signals. Key differences between our work, and that of SignalTap II, is that the latter will insert approximately 4 pipelining registers for each trace-connection, whilst we do not perform any pipelining. Additionally, when operating in post-fit mode, SignalTap II can only instrument flip-flops in the design, whilst in our work we allow any gate-level signal (combinational or sequential) to be traced.

Table V shows the runtime, logic utilization (ALMs in Altera terminology), memory utilization (M9K) and maximum operating frequency (F_max) for the largest benchmark mcml, when targeting a Stratix IV architecture in Quartus II and a Stratix IV-like architecture using our flow. For the Altera post-map flow, inserting trace instrumentation incrementally is about 40% faster than performing pre-map instrumentation; we believe the primary reason why this is not accelerated any further is that trace logic is inserted using the general-purpose incremental-compilation flow. This general-purpose flow is designed to cope with functional modifications to the circuit (e.g. ECO changes, bug fixes) as opposed to a trace-specific, observe-only flow which seeks to instrument without modifying, and so still has to perform placement, routing (and their preparation) steps. Furthermore, no guarantee exists for the circuit to be fully preserved: for pre-map insertion at 0.27 trace demand, routing conflicts required the whole circuit to be re-routed from scratch (much like mid-map insertion would) leading a reduced maximum clock frequency. We have been unable to instrument any more than 7600 signals due to exhausting all logic resources.

By comparison, the trace-specific flow presented in this paper is capable of preserving the circuit entirely (and with that, its critical-path). Although it takes the academic CAD flow significantly longer to compile the uninstrumented circuit, the runtime required to perform trace insertion is almost two orders of magnitude faster — driven mainly by the ability to skip packing and placement stages of the mapping flow. Additionally, because our flow does not perform any pipelining, we are able to reclaim all 100% of leftover memory resources for tracing; in particular, taking advantage of the flexibility offered by all memory blocks even when the trace demand is less than 1.0.

## VII. CONCLUSION

Circuit verification is an increasingly difficult, but necessary, task within IC design. Traditional pre-silicon techniques, such as software simulation and formal verification, are being augmented with FPGA prototyping in order to maximize the verification coverage that can be achieved. Whilst FPGAs devices can be used to implement circuits that can run at orders-of-magnitude faster than in simulation, the main challenge of using this platform is the lack of internal observability: when something goes wrong, the designer cannot look at signal values inside the circuit to debug this error.

A common solution for improving on-chip observability is to embed trace-buffers into the design. This allows a limited number of signal values to be recorded into on-chip memory, during real-time device operation, for off-line analysis. However, the choice of which signals to connect to a trace-buffer must be made prior to running the prototype, and should the designer wish to observe a different set of signals, the circuit must be recompiled.

In this work, we propose that incremental mapping techniques are used to accelerate this trace-insertion (and subsequent modification) procedure. Rather than recompiling the circuit from scratch, we propose that the original circuit mapping is completely preserved, and that new trace connections are made using only the free resources that were not previously used. For this to be feasible, we made several optimizations to the incremental CAD algorithms in order to exploit the unique nature of incremental-tracing: by recognizing that circuit signals can be observed by connecting *any* point of its net to *any* available input pin of *any* available trace-buffer, and by taking advantage of the internal symmetry of the FPGA architecture, the flexibility for making incremental trace connections was vastly increased.

We evaluated our incremental-tracing technique, applied after the FPGA mapping procedure is complete, by comparing it with two other strategies for trace-insertion: inserting prior to the mapping procedure, and inserting part-way through this procedure. We found that, when targeting a realistic FPGA with a channel width 20% greater than its minimum, our proposed post-map insertion technique is on average 98X faster than pre-map trace-insertion, and 22X faster than mid-map insertion when reclaiming 75% of the leftover memory capacity as trace-buffers. Furthermore, we find that the post-map solution has a smaller wirelength than solutions returned by both the pre- and mid-map strategies, and also that post-mapping insertion has only a $<$1% effect on the critical-path delay of the circuit, with less variance.

We believe that the ideas presented in this paper are extremely applicable to commercial FPGAs. By their very nature, FPGA devices contain a prefabricated set of general-purpose logic and resources that are assembled in such a way to

support the implementation of ideally, any digital circuit. Due to this generality, not all resources will be utilized in all situations, thus, there exists a unique opportunity to reclaim any such spare resources for debug without any additional silicon area. We have shown the feasibility of our incremental-tracing idea on a realistic island-style FPGA architecture based on the Altera Stratix IV family of devices, and we see no clear reason why our techniques cannot be realized on those same commercial island-style devices. Researchers interested in applying or extending our techniques are invited to download our Inc-Trace patch for VTR 1.0 from http://ece.ubc.ca/~eddieh.

## ACKNOWLEDGMENT

The authors would like to thank Altera for their support.

## REFERENCES

[1] S. Asaad, R. Bellofatto, B. Brezzo, C. Haymes, M. Kapur, B. Parker, T. Roewer, P. Saha, T. Takken, and J. Tierno, "A Cycle-Accurate, Cycle-Reproducible Multi-FPGA System for Accelerating Multi-core Processor Simulation," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '12, 2012, pp. 153–162.

[2] 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.

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

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

[5] Tektronix, "Certus Debug Suite," http://www.tek.com/sites/tek.com/files/media/media/resources/Certus_Debug_Suite_Datasheet_54W-28030-1_4.pdf, July 2012.

[6] E. Hung and S. J. E. Wilton, "Limitations of Incremental Signal-Tracing for FPGA Debug," in *FPL 2012, International Conference on Field Programmable Logic and Applications*, August 2012, pp. 49–56.

[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] 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.

[10] 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, 2009.

[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] D. Lewis, D. Galloway, M. Van Ierssel, J. Rose, and P. Chow, "The Transmogrifier-2: a 1 million gate rapid-prototyping system," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 6, no. 2, pp. 188–198, June 1998.

[18] M. Dini, "The Multi-FPGA Prototyping Platform," http://www.dinigroup.com/files/The%20DINI%20Group%20Multi-FPGA%20Prototyping%20Platform_9-4-10.pdf, September 2010.

[19] Kaivola, Roope and Ghughal, Rajnish and Narasimhan, Naren and Telfer, Amber and Whittemore, Jesse and Pandav, Sudhindra and Slobodová, Anna and Taylor, Christopher and Frolov, Vladimir and Reeber, Erik and Naik, Armaghan, "Replacing Testing with Formal Verification in Intel(R) Core(TM) i7 Processor Execution Engine Validation," in *CAV '09: Proceedings of the 21st International Conference on Computer Aided Verification*, 2009, pp. 414–429.

[20] M. Khalid and J. Rose, "A Novel and Efficient Routing Architecture for Multi-FPGA Systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 8, no. 1, pp. 30–39, February 2000.

[21] Free Software Foundation, "GCC 4.8 Release Series: Changes, New Features, and Fixes," http://gcc.gnu.org/gcc-4.8/changes.html, December 2012.

[22] 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.

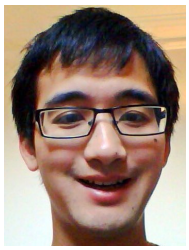
[23] R. Y. Rubin and A. M. DeHon, "Timing-Driven Pathfinder Pathology and Remediation: Quantifying and Reducing Delay Noise in VPR-Pathfinder," in *FPGA'11, Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2011, pp. 173–176.

[24] 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 *Proceedings of the 20th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2012, pp. 77–86.

[25] L. McMurchie and C. Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs," in *Proceedings of the 1995 ACM Third International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '95. New York, NY, USA: ACM, 1995, pp. 111–117.

[26] E. Hung and S. J. E. Wilton, "Scalable Signal Selection for Post-Silicon Debug," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (to appear, DOI: 10.1109/TVLSI.2012.2202409)*.

[27] 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: Proceedings of the 8th International Conference on Field-Programmable Technology; Sydney, Australia*, December 2009, pp. 96–103.



**Eddie Hung** received the M.Eng. degree in Electronic and Communications Engineering from the University of Bristol, Bristol, U.K. in 2008. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering at the University of British Columbia, Vancouver, Canada, where he is working on harnessing the reconfigurable and prefabricated nature of FPGA technology for rapid circuit debug. Along the way, he has also spent some time with Imperial College London, London, U.K.; Tektronix, Vancouver, Canada; The Chinese University of Hong Kong, Hong Kong; ARM, Cambridge, U.K.; Panasonic PSDCE, Bristol, U.K.; and Motorola, Swindon, U.K. His current research interests include novel ways of using FPGA architecture and CAD tools to achieve "simulator-like observability" during post-silicon debug.

**Steven J. E. Wilton** is a Professor in the Department of Electrical and Computer Engineering at the University of British Columbia. His research focuses on the architectures of next-generation Field-Programmable Gate Arrays and their associated Computer-Aided Design Tools. Along with his students, he has published over 100 papers in many areas related to Field-Programmable Technology, ranging from flexible memories, routing architectures, power-efficient architectures, packing, placement, and routing algorithms, analytical modeling and debugging techniques. He has spent time at Imperial College London and IMEC, and has been a consultant for Altera, Cypress, and Cadence. He was also a co-founder of Veridae Systems (acquired by Tektronix in 2011), which develops debug solutions for ASICs, FPGAs, and FPGA-based systems. He received best paper awards at the International Conference on Field-Programmable Technology in 2003, 2005, and 2007 and at the International Conference on Field-Programmable Logic and Applications in 2001, 2004, 2007, and 2008. In 1998, he won the Douglas Colton Medal for Research Excellence for his research into FPGA memory architectures.