Delay-Bounded Routing for Shadow Registers

Eddie Hung[⋆†] e.hung@imperial.ac.uk

Joshua M. Levine[⋆] josh.levine@imperial.ac.uk

Edward Stott[⋆] ed.stott@imperial.ac.uk

George A. Constantinides[⋆]

[⋆]Department of Electrical & Electronic Engineering
Imperial College London, England
g.constantinides@imperial.ac.uk

Wayne Luk[†]

[†]Department of Computing
Imperial College London, England
w.luk@imperial.ac.uk

ABSTRACT

The on-chip timing behaviour of synchronous circuits can be quantified at run-time by adding shadow registers, which allow designers to sample the most critical paths of a circuit at a different point in time than the user register would normally. In order to sample these paths precisely, the path skew between the user and the shadow register must be tightly controlled and consistent across all paths that are shadowed. Unlike a custom IC, FPGAs contain prefabricated resources from which composing an arbitrary routing delay is not trivial. This paper presents a method for inserting shadow registers with a minimum skew bound, whilst also reducing the maximum skew. To preserve circuit timing, we apply this to FPGA circuits post place-and-route, using only the spare resources left behind. We find that our techniques can achieve an average STA reported delay bound of ±200ps on a Xilinx device despite incomplete timing information, and achieve <1ps accuracy against our own delay model.

Categories and Subject Descriptors

B.8.2 [**Hardware**]: Performance Analysis and Design Aids

Keywords

Shadow Register; FPGA; Constrained Routing; Timing Measurement

1. INTRODUCTION

Aggressive process scaling has been the cornerstone behind the digital revolution that we live and breathe today. However, as the size of transistors shrink further and further into the nanometre spectrum, the ability to control any process variation has declined. This variation has required vendors to provision for the worse case, for example, by guaranteeing the performance of all of their devices only to the lowest common denominator.

By recouping some of this lost opportunity at runtime through per-device adaptation, ideally, we would expect reduced power consumption, increased throughput, and extended device lifetime. One method for enabling this, that has been gaining popularity, is to augment a design with shadow registers [2, 4, 6]. Conceptually, shadow registers exploit temporal redundancy inside a synchronous digital circuit in order to infer live, on-chip, timing information that can be used to 'personalise' each circuit to its host device, for example, by reducing its supply voltage, or by increasing its clock frequency. In particular, the reconfigurable and prefabricated nature of field-programmable gate arrays (FPGAs) offer a compelling platform for pursuing this goal.

To ensure accurate timing measurements, shadow registers must be located at a precise delay away from the register it duplicates. However, whilst in custom layout silicon it is possible for arbitrary delays to be added to the data or clock signals of a shadow register, this same flexibility does not exist in an FPGA, where circuits must be constructed out of a predetermined set of resources. In this paper, we present a method for inserting shadow registers, that: *i*) can attach to the most critical paths of a circuit with a bounded routing delay; *ii*) only operates post place-and-route, using just the spare and unused resources on an FPGA such that the original timing behaviour of the circuit remains unaffected.

Our paper is organised as follows: Section 2 provides a background on shadow registers and related work, and Section 3 provides the motivation the pursuit of post place-and-route shadowing, and the necessity of delay-bounded routing. Section 4 describes how we modify Dijkstra's algorithm with a rollback feature to enable minimum delay bounds, Section 5 describes our experimental application. We present and analyse our approach in Section 6, before describing our future work and conclusion in Sections 7 and 8.

FPGA'15, February 22–24, 2015, Monterey, California, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3315-3/15/02 ...$15.00.
http://dx.doi.org/10.1145/2684746.2689075.

2. BACKGROUND AND RELATED WORK

The idea of on chip timing measurement and shadow registers is not new, having already been presented by [2, 4, 6].

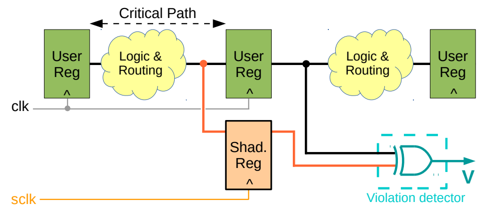
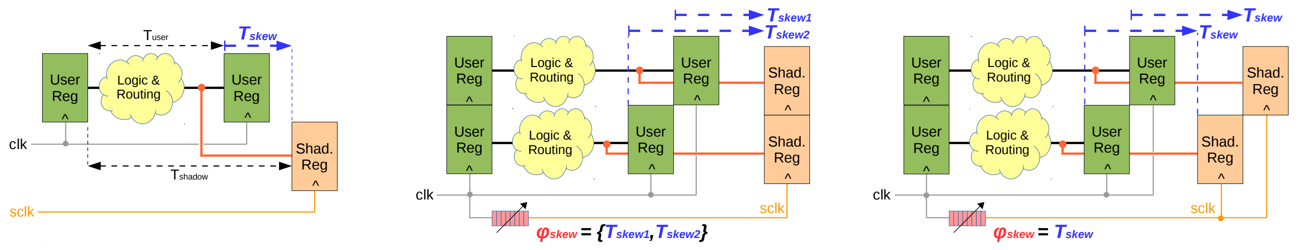


Figure 1: Illustration of a shadow register fed by a different clock, along with a violation detector circuit.



(a) Concept of path skew (T_{skew})

(b) Variance of T_{skew} between user paths requires phase compensation (ϕ_{skew}).

(c) Bounded T_{skew} requires single ϕ_{skew} value.

Figure 3: Utility of bounding the skew between user path and shadow path.

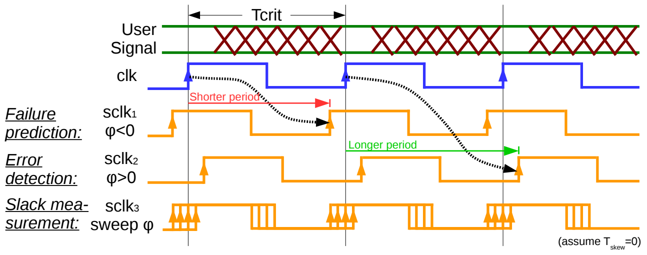


Figure 2: Timing diagram showing the three modes of operation for shadow registers.

The concept of shadow registers is shown in Figure 1. In particular, both the input and output of the user register must be accessible; the input is fed into a shadow register, and the outputs of both the user and the shadow register are compared using an XOR gate. Should the two latched values be different, the signal V would go high, thus indicating that (depending on the mode of operation) either the user or the shadow register experienced a timing violation in the previous clock cycle. The most challenging part of this method lies with inserting these shadow registers, which act as the (asynchronous) interface between the user circuit and any subsequent detection (XOR) or recovery logic — essentially, shadow registers decouple the latter from the former. For this reason, in this paper we focus only on the placement and routing of these register, and not on any downstream logic.

Three main modes of operating shadow registers exist, depending on the phase offset ϕ_{offset} between the user and shadow clock. Take for example Fig. 1, and let us assume that the critical net arrives at both the user and shadow register simultaneously. A negative ϕ_{offset} indicates a shadow clock that leads the user clock, thus the critical net is sampled into the shadow register earlier than being sampled by the user circuit, effectively shortening the circuit's clock period. This mode is commonly used for failure prediction, such as detecting when a device has aged sufficiently that the safety guard-band is breached. Bounding the skew of all shadow registers in this mode allows this guard-band to be positioned, and adjusted, more accurately and consistently across all registers.

A positive ϕ_{offset} corresponds to a lagging shadow clock, and effectively provides a longer clock period for the user circuit to return the correct result. This is the key technique for achieving Razor [6], a method that speculatively overclocks a circuit (most commonly, CPUs) beyond its rated limits, with the provision that any timing errors are infrequent, detectable, and recoverable. The final mode is when ϕ_{offset} is swept through a range of values in order to perform on-chip slack measurement by determining the exact point at which each shadowed path starts to fail, and has been used for device characterisation [13]. Bounding the skew for these modes would allow more precise Razor detection, as well as reduce calibration effort during slack measurement. All three modes are illustrated in Figure 2.

2.1 Negating Path Skew

In reality, it is unlikely that a critical net will arrive at the user register at the same time as to its shadow register. Thus, for accurate timing measurement using shadow registers, we must also consider their path skew.

The data skew of a single shadow path is the difference in routing delay between a signal arriving at the user register (T_{user}) and the shadow register (T_{shadow}) where $T_{skew} = T_{shadow} - T_{user}$. Although it may be expected that $T_{shadow} > T_{user}$, this is not always the case given that it is possible to shadow a net into a register located closer to its source than the user register. For registers driven by the same global clock buffer and balanced clock tree, let us first assume that the clock skew is zero (though in practice, as seen later, vendor tools report that even registers that are located relatively close to each other can have clock skews up to several hundreds of picoseconds). This is illustrated in Figure 3a.

In order for the signal to be latched into a shadow register at the same time as it is latched into the user register (i.e. for ϕ_{offset}=0) the shadow clock must lag the user clock by the absolute offset ϕ_{skew}=T_{skew}. Conveniently, on FPGAs, dedicated clock resources exist to accurately tailor the total phase offset, $\phi_{skew} + \phi_{offset}$ (by deriving a phase-shifted version of the user clock) at run-time. This contrasts with data signals, which must use general-purpose routing resources that must be determined during compilation (thus establishing T_{skew}) and which remains fixed whilst the device is operating. When shadowing multiple paths in the circuit, however, it is likely that the data skew of each path will be different, and hence, the phase offset requirements of each shadow register will also be different. Unfortunately, limited clock resources on FPGAs means that it is infeasible to supply individually shifted clocks for more than a handful of paths (as an example, the device used in this work supports a maximum of 32 global clock nets).

This situation, where two shadow registers share one phase-adjustable global clock net, is illustrated in Fig. 3b. Here, either one of the two shadow registers can be clocked with respect to its user register, but not both simultaneously; thus, when it may not be possible to customise ϕ_{skew} for each path, it would be valuable to control T_{skew} instead. Bounding the routing delay, and hence the data skew, to be identical across multiple shadow registers would allow them all to be shadowed simultaneously from the same clock net, as shown in Figure 3c.

2.2 Delay-Bounded Routing

Given that FPGA routing tools are required to find routes that meet both hold (minimum delay bound) and setup (maximum) time constraints, in some ways, existings tools already perform an element of delay-bounded routing. Thus, the allowed arrival window of all routed paths should be greater than T_{hold} (in the order of 50–300ps) and less than $T_{crit} - T_{setup}$ (where T_{setup} is in the order of 0–700ps). For T_{crit}=10ns (100MHz), this gives an arrival window of 9ns. In this work, we seek to make this window as small as possible.

The research community has primarily focused on improving circuit performance by minimising T_{crit}, the worse case delay of all paths through the circuit; algorithms like PathFinder [15] have been employed for this task. However, less attention has been paid to methods for elongating the best-case delay of paths in order to meet any minimum delay constraints; as an example, even the venerable VPR CAD suite [16] does not support hold time constraints.

Whilst the Altera routing tool supports both minimum delay (MINDELAY) and maximum delay (MAXDELAY) constraints for all nets, the Xilinx ISE router only supports minimum delay (OFFSET) constraints for nets that terminate at an I/O interface, to meet external hold requirements. In both tools, different worse case timing models are used — fast corner for MINDELAY (hold) and slow corner for MAXDELAY (setup) analysis — which makes precise routing difficult. For shadow registers, we care about the critical-path at the slow corner only. Fung et al. [7] describe a slack reallocation method to optimise both short paths (MINDELAY) and long paths (MAXDELAY) simultaneously on a regular routing algorithm, in order to meet hold and setup time constraints — this is the method reportedly used by Altera tools.

On the more general graph routing problem, K-shortest path algorithms exist (e.g. Yen's algorithm [18]) to find not just the shortest path from a single source to a single sink, but the set of K-1 next-shortest paths. By finding a sufficiently large value for K, it would be possible to find the shortest path which fulfils the minimum cost bound. Similar to Yen's, our algorithm removes edges from the graph to find longer paths, but unlike Yen's, we give up the ability to search the graph optimally (and exhaustively) for improved runtime by removing edges permanently, and not restarting Dijkstra on each removal.

Prior work on shadow registers [13] relies on standard incremental compilation techniques to insert shadow registers. MINDELAY and MAXDELAY constraints can be applied to (coarsely) bound T_{skew} across all nets, though this can be difficult when each constraint is applied to a different timing corner. For certain applications though (e.g. timing slack measurement) variations in T_{skew} can be calibrated out during post-processing.

2.3 Post P&R Instrumentation

Inserting timing instruments only after placing and routing the user circuit, and using only those resources left over, is a key part of being able to shadow critical registers without disrupting their timing characteristics. This statement is supported by the results of the following section, but prior research [12, 10] have also taken this approach for adding debug instrumentation.

Whilst, on the surface, it would appear that constraining the insertion process to use only spare resources that were left behind may be overly restrictive, significant flexibility is recouped from exploiting the convenient property that debugging signals can be connected to any trace-buffer input for it to be observable. Thus, unlike the user circuit, where nets need to be routed exactly from a single source pin to a predetermined set of sink pins, debug nets need only be routed from any point along the existing user net to *any* one of the many trace-buffer sinks that are available.

A keen similarity exists between shadow registers and trace-buffers, given that when shadowing a user net, tools will also have the freedom to connect to any one of the many spare register resources available; if anything, there are likely to be even more spare registers than trace-buffer inputs.

3. MOTIVATION

In this section, we present both a case for why shadow registers should be inserted post place-and-route, as well as a case for why delay-bounded routing is important for shadow register insertion. The delay-bounded experiments that follow targets shadow register insertion that minimises T_{skew} subject to $T_{skew} \geq +1$ns after the user register.

3.1 Case for Post P&R Shadowing

Table 1 presents a comparison between inserting shadow registers before, and after, the place-and-route procedure (defined to be the `map` and `par` tools in ISE) of three benchmarks. On each, three experiments are performed: the 'Base' column represents the baseline compilation run of this benchmark (without shadow registers). In order to allow for shadow registers to be inserted prior to place-and-route, this baseline is generated over two stages. The first stage involves synthesising (but not placing or routing) the benchmark as normal, using ISE's `xst` tool. The second stage converts this synthesis result into a flattened, technology-mapped Verilog file using ISE's `netgen` tool, creating a functionally-equivalent structural description that is typically used for verification in a simulator. This Verilog file is then compiled again from scratch, as before.

By first flattening the benchmark, we preserve the exact logical structure of the circuit (prior to packing, placement or routing) given that it explicitly instantiates all LUT, FF, *etc.* primitives of the design, to allow for shadow register insertion. This allows us to maintain the one-to-one mapping between a critical register (only identifiable after place-and-route) with a `reg` in the original source code. For this same reason, we were unable to perform any pre-synthesis shadowing experiments. This flattening step would not be necessary if it was possible to directly modify the post-synthesis netlist; a more elegant approach that we would like to adopt in future work would be to directly instrument a synthesised EDIF netlist.

The 'Source' and 'Post P&R' columns show the results for inserting shadow registers at the source-level (by appending to the structural Verilog description) and after the place-and-route procedure. A number of statistics are shown: the first three rows list the logic utilisation figures for each of the experiments, whilst the fourth row shows the critical-path delay. The following row shows the number of unique endpoints (defined as register, IOB or RAM input pins) that are within the most critical percentile of the critical-path delay, as reported by the Xilinx STA tool. For the LEON3, we are interested in all critical-paths that are within 10% of its critical-path delay (at 13.33ns, this results in 1436 paths with a slack less than or equal to 1.333ns), and for the AES x3 and JPEG-x2 benchmarks, we extend this margin to 40%. The next two rows indicate the proportion of these endpoints

Table 1: Comparison between inserting shadow registers at the source, and post place-and-route, with the latter preserving significantly more critical nets. ('Source' recompiles the entire circuit which can unpredictably lead to better, or worse, results).

		Shadow Method	
	Base	Source	**Post P&R (this work)**
Slice util.	93.5%	88.9%	**94.5%**
LUT util.	63.5%	56.6%	**64.0%**
Register util.	20.1%	20.6%	**20.6%**
Tcrit (ns)	13.329	13.331	**13.333**
Num. critical nets	1436	1222	**1436**
Common to Base	100%	41.0%	**100%**
Shad. coverage	-	41.0%	**98.7%**
Pack & Place time (s)	1875	1974	**-**
Routing runtime (s)	1281	1241	**183**

(a) LEON3 (all critical nets within 10% of Tcrit)

		Shadow Method	
	Base	Source	**Post P&R (this work)**
Slice util.	87.0%	86.9%	**89.8%**
LUT util.	71.4%	71.7%	**72.7%**
Register util.	10.6%	12.4%	**12.3%**
Tcrit (ns)	6.055	5.865	**6.055**
Num. critical nets	5362	7400	**5362**
Common to Base	100%	53.1%	**100%**
Shad. coverage	-	53.1%	**94.2%**
Pack & Place time (s)	1014	1408	**-**
Routing runtime (s)	479	486	**224**

(b) AES-x3 (within 40% of Tcrit)

		Shadow Method	
	Base	Source	**Post P&R (this work)**
Slice util.	86.8%	88.8%	**90.1%**
LUT util.	60.9%	61.3%	**61.8%**
Register util.	41.7%	42.7%	**42.6%**
Tcrit (ns)	13.936	15.611	**13.939**
Num. critical nets	3290	3335	**3295**
Common to Base	100%	46.7%	**99.8%**
Shad. coverage	-	46.7%	**84.0%**
Pack & Place time (s)	871	911	**-**
Routing runtime (s)	748	3658	**276**

(c) JPEG-x2 (within 40% of Tcrit)

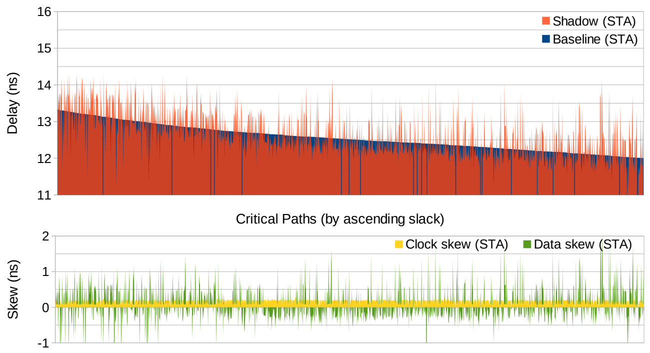
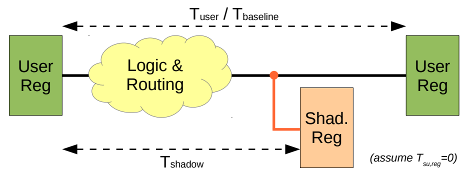
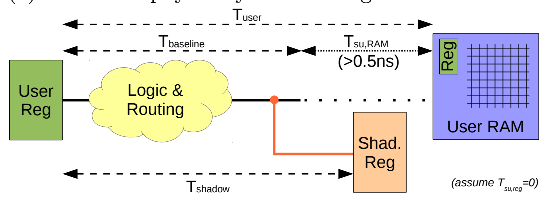


Figure 4: Unbounded — shadowing LEON3 critical endpoints into closest register, using Xilinx `par` (runtime: 810s).



(a) Shadow is physically closer to signal source.



(b) Shadow path occurs within setup window of endpoint.

Figure 5: Two possible scenarios for negative data skew.

that are identical to the baseline experiment, and the number that were shadowed successfully, and the additional runtime required to do so.

Our experiments here show that, when inserting shadow registers at the source-level, the paths that were worse case originally do not remain so — in fact, approximately 50% of all worse case endpoints from the baseline remained critical after recompilation. In contrast, inserting shadow registers post place-and-route resulted in exactly the same endpoints remaining critical, of which over 80–90% of them were successfully shadowed. We analyse why not all endpoints can be shadowed in a following section.

Unsurprisingly, inserting shadow registers post place-and-route can still have a small effect on critical-path delay of a circuit due to the extra loading induced by adding an extra fanout to the critical net. For LEON3 and JPEG-x2, this amounts to 3–4ps, whilst no effect was observed on the AES-x3 circuit because this extra fanout occurred inside the logic slice. In contrast, inserting at source-level can be quite chaotic: for LEON3, utilisation of slice and LUT resources was less than the baseline circuit, and for AES-x3, inserting shadow registers turned out to improve its T_{crit}.

3.2 Case for Delay-Bounded Shadowing

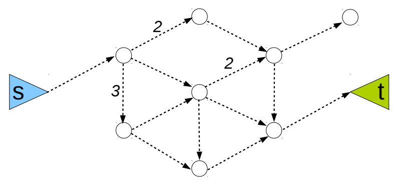
Figure 4 compares the total path delay (from state element to state element) in the baseline LEON3 circuit, and the new path to the shadow register, when inserted post place-and-route, but without any delay bounding. We achieve this by placing a register at the closest possible site to the end of each critical net (mimicking an incremental placement algorithm that seeks to minimise net wirelength), and then invoking Xilinx `par` to route to this new sink. We constrain all paths from the original circuit to this new set of shadow registers as being a multi-cycle path with a maximum delay of +1ns greater than the clock period, but no support exists for minimum delay constraints.

The result is that some shadow paths have a delay that is shorter than the baseline path, whilst in some other cases, this shadow path is longer, as shown in the upper graph. Highlighting this more clearly is the lower graph showing the path skew between the shadow and the baseline registers. A negative skew may exist because it has been possible to place and route a new shadow register: *a*) physically closer to the signal source than in the original baseline circuit; *b*) within the setup time window of the endpoint, which for RAM inputs as an example, can exceed 0.5ns. These scenarios are illustrated in Fig. 5.

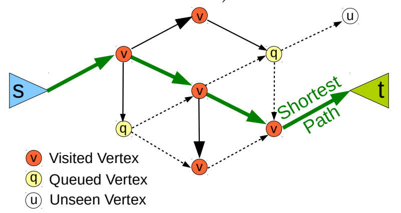
Clearly, such a large range of data skews is undesirable as it requires an equal number of skewed clocks if all shadow registers are to be operated simultaneously.

4. DIJKSTRA WITH ROLLBACK

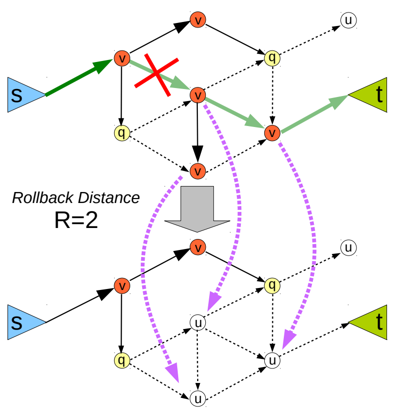
The FPGA routing problem can be described using a directed graph $G(V, E)$, where V is the set of vertices in the graph, and E describes the set of directed edges between two vertices. Each routing wire in the FPGA can be represented using a vertex, and each edge represents a programmable connection between two wires. A cost is associated with every edge, representing the routing delay of this wiring connection.



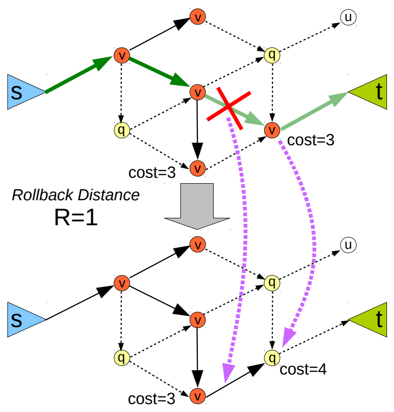
(a) Example routing graph (edge costs as 1, unless otherwise labelled).



(b) Standard invocation of Dijkstra.



(c) Rollback part 1: invalidate all children that were visited, or queued to visit.



(d) Rollback part 2: repair priority queue.

Figure 6: Illustration of Dijkstra with rollback.

During FPGA routing, the primary concern is typically to maximise performance, and that involves minimising the critical-path delay, which is equivalent to minimising the worse case cost of a path from some source vertex s to a sink vertex t. Once all setup constraints have been met (or as many as possible), then the tool can return to fix any short paths that violate hold time constraints by adding additional delays. Based on its output messages, we hypothesise this is how the Xilinx `par` router operates.

Finding the shortest path through G is a well-studied problem, with Dijkstra's algorithm being the de-facto choice for solving this optimally. This algorithm operates as a breadth-first search, and at a very high level, it works as follows: expand outward from s, explore the next closest unvisited vertex, expand outward from this vertex and repeat, until t is reached. Guaranteeing that every vertex is visited using the shortest path is accomplished using a priority queue, which sorts all unvisited vertices by ascending cost, so that at each step, the lowest cost vertex is always processed next.

While it is possible to find the shortest path between s and t in this way, it is not trivial to find paths with any other constraints, such as a path with cost exactly equal (or as close as possible) to T. The reason for this is that Dijkstra's algorithm only records the shortest path to every vertex in the graph, rather than *all* possible paths, the number of which can be expected to grow exponentially with $|E|$.

In many ways, this is a problem that bears many similarities with the subset sum problem, which is NP-complete. Within the context of this work, we seek to solve the problem: given a set of wire delays, how can we compose a target delay T_{skew} (exactly, or as close as possible)? Further constraints also exist on selecting wire delays since each wire may only connect to an adjacent wire, and multiple, non-overlapping, solutions are sought across all critical nets.

4.1 Differences from Regular FPGA Routing

Unlike regular FPGA routing, there are a number of key differences that set our problem apart: *i*) in order to preserve circuit timing, we are restricted to using only the leftover resources in the FPGA; *ii*) in order to allow the shadow register path to share as much of the original user path as possible, we branch from the original routing as close to the user register as possible; *iii*) we wish to keep the arrival window of all shadow registers as small as possible, by finding the shortest path that is at least our skew target; and *iv*) each critical net can reclaim any spare register in the FPGA as a shadow register.

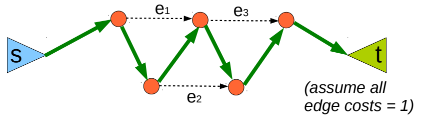


Figure 7: Example of a path with cost=6 that would never be found unless e_1, e_2 and e_3 were eliminated.

Regarding that last point, unlike regular FPGA routing where each net must be routed to all of its sinks exactly, a critical net is free to connect to any spare register. This element of freedom was exploited in prior work [11], which applied a single-commodity minimum-cost flow algorithm to route signals into pipelining registers. However, a key limitation of the minimum-cost flow algorithm is that it is only capable of minimising for the total delay of all nets (equivalently, their average case delay), and is not capable of optimising for the worse case net delay (MAXDELAY), nor the best-case (MINDELAY). For this reason, a direct application of the minimum cost flow algorithm is unsuitable for inserting shadow registers.

4.2 Rollback

Rather than terminating when the shortest path to t is found, we propose that the graph search continues in the hope of finding a longer path to the sink t. However, because t has already been visited, it will never be visited by the algorithm ever again. To rectify this, we rollback the state of the algorithm as if a previous edge to on the shortest path had not existed.

Consider the graph shown in Figure 6a, and an unmodified invocation of Dijkstra as in Fig. 6b. Rollback consists of two parts, with a free choice of how many vertices to undo; let us define the number of vertices to rollback as R and the last vertex to rollback as v_R. Part one is, if $R>0$, to eliminate all vertices on the heap that have already been visited, or in the priority queue, that are downstream from v_R. This is illustrated in Fig. 6c, where $R=2$. We perform this task by recursively checking the fanout vertices of v_R and removing

```
# Place a shadow register for each net at its closest
      spare location
net2closest = placeClosest(criticalNets,allRegs)
foreach net in criticalNets:
    closestReg = net2closest[net]
    # Find the closest spare register from this net
    path = searchShortestPath(net,closestReg)
    # Keep searching for (longer) paths to only that
          chosen register until length meets target
    while not path.empty and path.length < Tskew:
        rollbackSearch(path, R)
        path = continueSearch(closestReg)
    # Remove path from future graph searches
    markAsUsed(path)
    # Reset Dijkstra state (i.e. priority queue)
    resetDijkstra()
```

Listing 1: One-Closest Algorithm

```
foreach net in criticalNets:
    # Find the closest spare register from this net
    path = searchShortestPath(net,allRegs)
    # Keep searching for (longer) paths to any spare
          register until length meets target
    while not path.empty and path.length < Tskew:
        rollbackSearch(path, R)
        path = continueSearch(allRegs)
    # Remove path from future graph searches
    markAsUsed(path)
    # Reset Dijkstra state (i.e. priority queue)
    resetDijkstra()
```

Listing 2: One-All Algorithm

```
do:
    # Find the closest spare register from any net
    path = searchShortestPath(criticalNets,allRegs)
    # Keep searching for (longer) paths from any net to
          any spare register until length meets target
    while not path.empty and path.length < Tskew:
        rollbackSearch(path, R)
        path = continueSearch(allRegs)
    # Remove path from future graph searches
    markAsUsed(path)
    # Reset Dijkstra state (i.e. priority queue)
    resetDijkstra()
while not path.empty
```

Listing 3: All-All Algorithm

any item with a predecessor (shortest path) edge that leads back to it.

Part two is to revisit all of the fan-ins of v_R to consider if any of the previously visited vertices would have inserted it into the priority queue at the smallest cost that is higher (but not equal) to the previous smallest. This repair procedure is illustrated in Fig. 6d, for R=1.

A larger value for R would result in backtracking further in the current shortest path, and to restrict the algorithm to finding a higher cost path from before that last vertex. Given that the number of vertices reachable can be expected to increase exponentially with distance from the s, larger R values can also be expected to prune a greater number of paths from the solution space.

Performing rollback in this manner does not guarantee that we will find the shortest path no less than an arbitrary delay T, should one exist. Consider the example set out in Fig. 7, where unless all three edges $e_{1...3}$ were eliminated during rollback, it is impossible to find a path which is cost$\geq$6, for any fixed value of R. Incidentally, the longest path through this graph is 6, and finding the longest path through a cyclic graph (such as those in FPGAs) is known to be an NP-hard problem.

4.3 Proposed Algorithms

We propose three different heuristic algorithms for finding the shortest-path, subject to a MINDELAY constraint, between a set of critical nets and a set of shadow registers. *One-Closest:* is intended to emulate a standard incremental compilation flow. First, it places a shadow register onto all critical nets, at the closest spare location, before attempting to find the shortest path to this register that is greater than or equal to the target T_{skew} value. After each net is routed, the Dijkstra algorithm state is reset for the next search. This is shown in Listing 1.

One-All: improves on One-Closest by exploiting the freedom that each critical net may connect to any spare register, rather than a specific one chosen ahead of time. The pseudocode for this algorithm is shown in Listing 2.

All-All: goes further and considers routing all nets to any spare register simultaneously, as shown in Listing 3. In each iteration, this algorithm finds the shortest path (that is greater than or equal to the target T_{skew}) between any critical net and any spare register, and marks that path as being used. This approach is very similar to the successive shortest paths algorithm that is used to find the minimum-cost flow, with the difference that we employ rollback to ensure that each path meets a minimum delay requirement. Given that our rollback method does not preserve optimality, the All-All algorithm is also not guaranteed to find the minimum-cost solution in which all paths meet the T_{skew} target.

A point of note is that all three algorithms are greedy; for the One-Closest and One-All approaches, each net is operated on in turn (as in PathFinder [15]) and for each the first path found that meets the minimum T_{skew} target is claimed. Similarly, for the All-All algorithm, all nets are considered simultaneously, but for each iteration the first solution that meets the target is also claimed. Unlike PathFinder, however, no negotiated congestion is performed. As we see in the following sections, our proposed approach, combined with the sheer flexibility of the FPGA fabric, produces acceptable results even without negotiation.

5. EXPERIMENTAL APPLICATION

Although we believe that our techniques are valid for any FPGA vendor, we evaluate our techniques on the Xilinx platform, due to the fine-grained access that is available through the Xilinx design language (XDL) format. The XDL format is a text-based representation that allows designers to read and write to all aspects of the Xilinx netlists, and can be used to change LUT contents, how they are packed, where they are placed, and how they are routed. This level of access is sufficient to construct an entire CAD toolchain as evidenced by the VTR-to-Bitstream project [9]. The only information that is missing from this format is the delay of each individual wire on the FPGA. For this reason, we use wire delay and setup time values estimated (to picosecond accuracy, ranging from 10ps to 707ps) using a linear regression model.

In this work, we use Xilinx ISE 13.3 to place-and-route our benchmarks, and target the xc6vlx240t Virtex6 FPGA found on the ML605 evaluation board. We also use Torc [17] for manipulating the XDL format (such as extracting the set of all spare register and routing resources, including LUT route-throughs) and LEMON [5] for graph search operations. Experiments were performed on an Intel Core i7-3770 CPU workstation, with 16GB RAM, running Xubuntu 14.04. Runtime was measured using the `/usr/bin/time` utility.

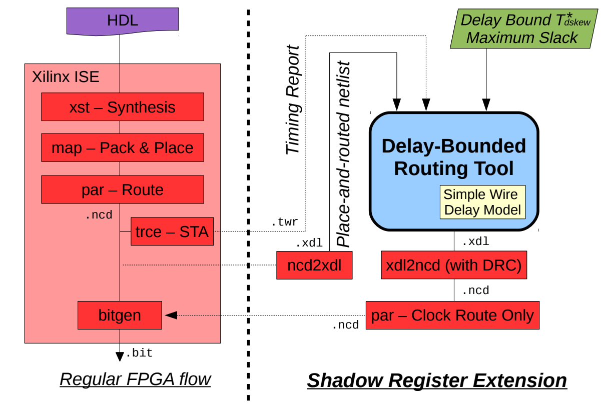


Figure 8: Proposed shadow register extension to design flow.

Table 2: Benchmark summary.

	LEON3	AES-x3	JPEG-x2	xc6vlx240t capacity
Slices	35217	32809	32733	37680
LUTs	95766	107685	91741	150720
Registers	60562	32025	125642	301440
RAMs	688	-	124	832
DSPs	32	-	172	768
Tcrit (ns)	13.329	6.055	13.936	-
Wirelength	2.1M	1.1M	1.6M	6.3M

5.1 Proposed Flow

Our proposed extension to the regular FPGA design flow is shown in Figure 8. On the left is a typical Xilinx flow: the HDL user circuit is first synthesised into FPGA resources, and then packed and placed onto physical locations on the target FPGA. The `par` tool is then called to find a viable routing configuration that connects all the necessary resources. Static timing analysis (STA) can then be performed using the `trce` tool, before generating a bitstream that can be programmed onto the device.

On the right is our proposed extension to this flow for inserting shadow registers. The inputs from the regular flow are a verbose STA report which details the exact resources that make up all critical and near-critical paths, as well as the place and routed netlist, which is converted from the binary format (`.ncd`) to the text-based `.xdl` format. On top of this, the user specifies the minimum T_{skew} (e.g. +1ns from user register) as well as the maximum slack for paths to be shadowed.

The output of our delay-bounded shadowing tool is a modified XDL netlist with as many of the most-critical endpoints requested as possible routed to newly-placed shadow registers, using only the leftover resources on the FPGA. This XDL format is converted back into an `.ncd` (which includes passing the Xilinx design rule check, DRC) for use in downstream Xilinx tools, necessary for routing the newly-inserted shadow register clock, before returning into the regular flow for bitstream generation.

5.2 Benchmark Analysis

We evaluate our proposed techniques on three benchmarks: the LEON3 system-on-chip, an AES encoder/decoder chain, and a JPEG decompression circuit. The LEON3 [1] is a functioning, open-source, multi-core SPARC SoC that can boot Linux; we configure it with 8 cores, each with 64kB of instruction and data cache, as well as a DDR3 memory controller,

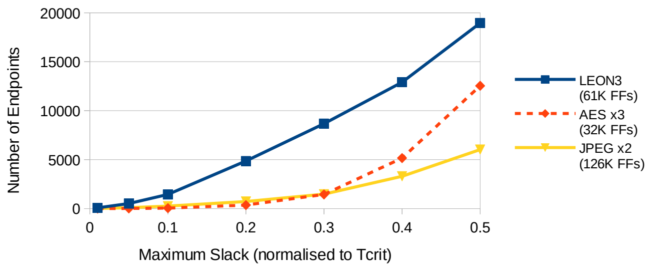


Figure 9: Number of endpoints at various slack values (normalised against Tcrit).

and Ethernet, CompactFlash and other peripherals. The arrangement of the DDR3 and other SoC components requires that the main CPU clock be constrained to 75MHz (13.33ns). We also constructed an AES encoder/decoder based on [3] consisting of three 128-bit AES encoders followed by three decoders. The encoder plaintext, and cipher-key, inputs are fed by LFSRs which are subsequently checked against the decoded output.

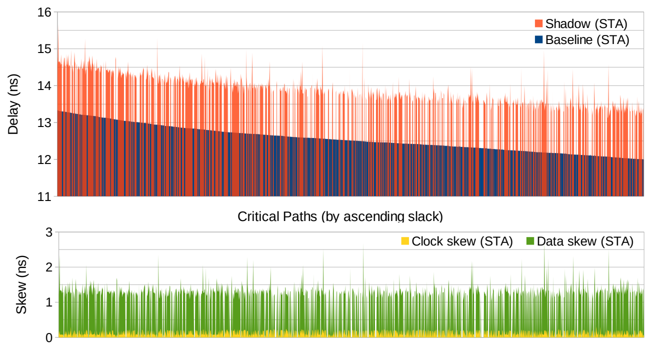
Our third benchmark is made up of two parallel instances of a C-based JPEG decoder [8], synthesised into Verilog using Vivado HLS 2013.4 with an aggressive timing constraint of 1ns in order to encourage pipelining and higher register utilisation. For the latter two benchmarks, no timing constraints are explicitly given for implementation and so we operate ISE in its performance evaluation mode, which aims to minimise the clock period. Table 2 summarises their resource utilisation and critical-path delays; we estimate the wirelength of each circuit by counting all occupied length 1, 2, 4 and 16 wires in the XDL netlist, and divide this by the total wirelength derived from the XDLRC device database.

Figure 9 plots the number of endpoints that have a slack value less than or equal to a fraction of the critical-path delay. This is an important metric as it shows how many paths must be shadowed in order to achieve a certain amount of timing coverage. For example, shadowing all endpoints within a normalised slack of 0.1 would allow, ideally, the circuit to be safely overclocked by up to 10% and still be able to detect all timing errors. Of the three benchmarks, LEON3 has the highest proportion of registers that are near-critical — we believe that this is due to it being the only one of the three benchmarks that has a timing constraint. With a timing constraint, the CAD tool will only attempt to optimise the circuit just enough to meet requirements.

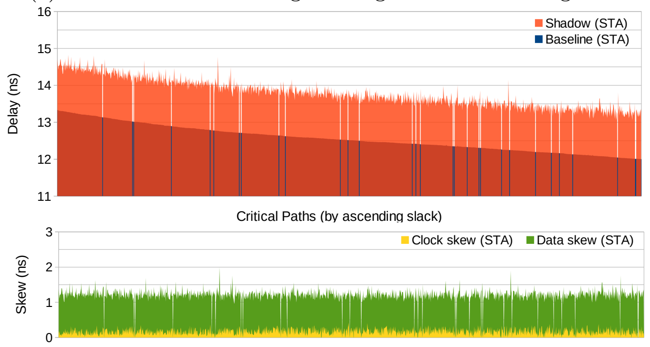
6. RESULTS

Figure 10 shows a comparison of the path delay and skew values, as reported by the Xilinx STA tool, for the three proposed bounded routing methods. The data is presented similarly to Fig. 4, where the total path delay to the baseline critical endpoint and the shadow registers is shown on the upper graph, and the skew between these two registers shown on the lower graph, broken down into data skew and clock skew elements.

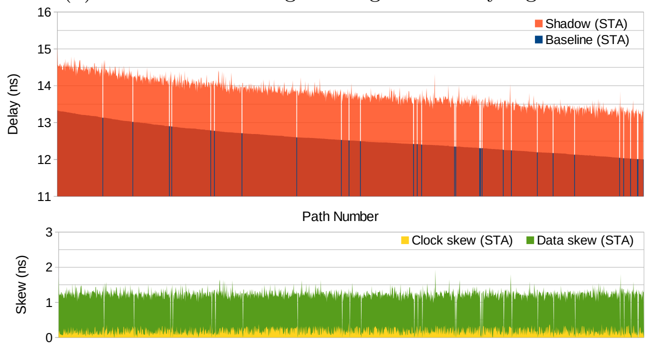
The effectiveness of the One-Closest approach — shadowing each signal into its closest register — with rollback distance R=4 is shown by Fig 10a. Although it is clear that applying a minimum delay bound does have an effect, not all signals were shadowed successfully (as indicated by a missing delay and skew value), and the skew variation is large, where the shortest path that was greater than T_{skew} turned out to be much larger.



(a) One-Closest: shadowing each signal into closest register.



(b) One-All: shadowing each signal into any register.



(c) All-All: shadowing all signals into any register.

Figure 10: Delay and skew comparisons (as reported by Xilinx `trce` STA) between the proposed delay-bounded routing methods, on LEON3, for $T_{skew} \geq +1$ns.

The One-All approach is shown in Fig. 10b, and the All-All approach in Fig. 10c, both with R=0. Both show an improvement in the delay variation, as well as the number of signals that were successfully shadowed. Rather interestingly, the results show that the One-All algorithm performs almost as well as the more complex All-All algorithm, which considers all signals simultaneously.

The results for Figure 10 show that, despite targeting a shadow skew of +1ns, the final timing results by the Xilinx STA reveals that deviations from this value exist. These errors are captured in Table 3, which shows the maximum and mean absolute error, and the standard deviation and range of the error from the desired +1ns value. The range is a key metric which measures the phase offset over which an engineer must align in order to detect violations; ideally, a value of zero means that only one phase offset exists, and that all shadow registers can perform detection simultaneously.

Table 3: Summary of delay-bounded error for nets shadowed, on LEON3, with 1436 critical nets and $T_{skew} \geq +1$ns.

	Unbounded	**Delay-bounded**		
(ns)	One-Closest	One-Closest	One-All	All-All
Nets shad.	1424	1051	1417	1418
Max. |Err|	1.942	1.779	1.002	0.928
Mean |Err|	0.867	0.329	0.223	0.249
StdDev Err	0.367	0.187	0.111	0.107
Range Err	2.771	1.744	1.075	0.956

(a) From Xilinx `trce` STA.

	Unbounded	**Delay-bounded**		
(ns)	One-Closest	One-Closest	One-All	All-All
Max. |Err|	2.011	1.586	0.788	0.704
Mean |Err|	1.012	0.201	0.114	0.114
StdDev Err	0.431	0.183	0.090	0.092
Range Err	2.763	1.754	0.948	0.860

(b) From Xilinx `trce` STA, with clock skew omitted.

	Unbounded	**Delay-bounded**		
(ns)	One-Closest	One-Closest	One-All	All-All
Max. |Err|	-	1.636	0.114	0.114
Mean |Err|	-	0.089	<0.001	0.001
StdDev Err	-	0.165	0.004	0.004
Range Err	-	1.636	0.114	0.114

(c) From our delay model, omitting clock skew.

Results are divided into 3 categories, corresponding to each of the three subtables (a)–(c), which presents results as reported by the Xilinx STA, results from Xilinx STA but with the effects of clock skew omitted, and results as viewed from our proposed router when using a simplified delay model, respectively. The first column shows the results corresponding to the One-Closest experiment from Fig. 4, which uses the Xilinx `par` router. Given that the Xilinx router does not support MINDELAY constraints, it seeks to optimise for MAXDELAY only and reports the highest mean error.

The remaining three columns show the results of our delay-bounded approach. The second 'One-Closest' column corresponds to routing each critical net, one at a time to its closest spare register, as shown in Fig. 10a. The third column 'One-All' considers each critical net to the closest of all registers (Fig. 10b) and fourth column 'All-All' considers all nets to all registers (Fig. 10c).

Table 3c shows the lowest mean error given that our tool optimises for and analyses against a simplified wire delay model (also without considering clock skew) as opposed to the accurate delay database to which the Xilinx STA tool has access to. When measured against our own model, we find a significant reduction in the mean and standard deviation of the bound error. Regardless, we find that the One-All algorithm performs just as well as the more complex All-All algorithm.

We believe that there are two main sources of error: *i*) neglecting the effects of clock skew; and *ii*) mismatch between our wire delay model and Xilinx's timing database. However, to reduce these errors would require proprietary Xilinx device information.

6.1 Effect of Rollback Distance

The choice of rollback distance R can have an effect on the performance of the algorithm. Table 4 shows the effect of varying R for each of the three algorithms, on the LEON3 benchmark. A value of R=0 represents that only the last edge into the target is rolled backed, which is sufficient for the

Table 4: Varying rollback distance R for LEON3 (T_{skew} $\geq$+1ns); default in bold.

$R \rightarrow$	0	1	2	3	4
Nets shadowed	13	323	533	824	**1051**
Model |*Error*|:					
Mean (ps)	79	38	44	81	**89**
Max (ps)	292	1252	1656	2373	**1636**
Runtime (s)	1705	1299	1067	757	**509**

(a) One-Closest

$R \rightarrow$	**0**	1	2	3	4
Nets shadowed	**1417**	1415	1386	1090	318
Model |*Error*|:					
Mean (ps)	**<1**	<1	<1	4	61
Max (ps)	**114**	171	829	262	549
Runtime (s)	**183**	188	214	180	130

(b) One-All

$R \rightarrow$	**0**	1	2	3	4
Nets shadowed	**1418**	1417	1398	993	139
Model |*Error*|:					
Mean (ps)	**<1**	<1	<1	<1	11
Max (ps)	**114**	171	34	184	140
Runtime (s)	**1281**	1456	2358	2708	262

(c) All-All

Table 5: Varying T_{skew} target for LEON3, using One-All algorithm; default in bold.

T_{skew} target→	+0.0ns	0.5ns	**+1ns**	+1.5ns	+2.0ns
Nets shadowed	1417	1419	**1417**	1416	1418
Model |*Error*|:					
Mean (ps)	110	3	**<1**	<1	<1
Max. (ps)	914	281	**114**	59	19
Runtime (s)	122	137	**183**	263	389

One-All and All-All algorithms, given that it can always route to another shadow register (that is further away). However, this is not the case for the One-Closest algorithm, which has no such freedom, and thus requires R >0 in order to route more than a small fraction of all signals. In the latter case, the larger the value of R, the fewer paths to that same register are eliminated and hence the greater the number of nets that can be shadowed (with greater error).

Increasing R appears to be detrimental for the One-All and All-All algorithms, matching intuition, as it eliminates that register from consideration to target another that is further away. The runtime of each of the three algorithms is also reported in Table 4, showing that One-All is faster than the more complex All-All algorithm (which considers all signals simultaneously) with comparable quality. The One-Closest algorithm is the slowest as it has the most failing nets, with each failing net requiring the entire routing graph to be exhausted before a signal is deemed unroutable.

6.2 Effect of Tskew Target Size

Table 5 shows the effect of varying the target T_{skew} size, when applying the One-All algorithm. A smaller minimum T_{skew} target allows more signals to be routed, but increases error given that not all signals can be routed to a shadow register within this delay budget. On the other hand, a larger target value provides more routing flexibility to assemble a net with a delay of at least T_{skew}, As can be expected, the runtime can also be seen to increase as the minimum target increases.

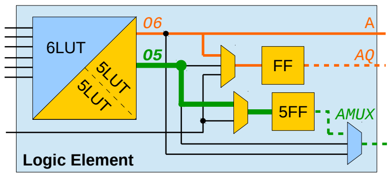


Figure 11: Virtex6 logic element where LUT output signal "O5" cannot reach any output pin.

Table 6: Reason for signals failing to shadow (for T_{skew} $\geq$+1ns using One-All algorithm)

	LEON3	AES-x3	JPEG-x2
Num. critical nets	1436	5362	3290
Nets shadowed	1417	5049	2769
i) Blocked in LE	10	313	468
ii) Blocked out LE	4	-	23
iii) Path search failed	5	-	30

6.3 Unroutable Signals

Three possible reasons exist for why user signals fail to be routed to a shadow register, either: *i)* it was not possible to connect the critical net to an output pin; *ii)* it was not possible to connect the critical net to any shadow register, even without a delay bound; or *iii)* a path meeting the minimum T_{skew} delay could not be found.

In the first scenario, the signal to be shadowed — i.e. the input signal of the user register — may be blocked from accessing the global routing network. Figure 11 shows the structure of a Virtex6 logic element, which contains 7 input pins, 6 of which feed a 6-input lookup table (which can be fractured into two 5-input tables) and a bypass input, as well as 3 outputs pins, fed by two registers. For scenario *i)*, consider when both fractured LUTs are used in this logic element, and the output of the second LUT (labelled "O5") is latched into the "5FF" register. In this case, it becomes impossible to connect the combinational signal to a free output pin so that it can be shadowed without modifying the user circuit.

For scenario *ii)*, where critical nets can already access the global network, we find that a small number of signals can never be connected to any shadow register, even without any delay bounds. We find this result by computing a maximum flow between all net sources and all net sinks. Scenario *iii)* represents the case when a signal that is equal or greater than the T_{skew} target could not be found. Either this could be because such a path doesn't exist under any situation, or because our heuristic rollback mechanism we employ was unable to find one of the paths that do.

The frequency of each of these four scenarios in our experiments is shown in Table 6. Across all benchmarks, the majority of failing signals came from the first scenario, where signals could not be extracted from inside of their logic element, whilst scenario *iii)* represents only a small fraction of all signals. In future, this could be alleviated by modifying the FPGA architecture so that such signals can always be accessed, by modifying the baseline CAD tools to prevent such dense packing for critical nets, or by allowing the post place-and-route circuit to be modified during shadow register insertion. The latter solutions could also be used to combat the three remaining scenarios *i)* and *ii)* by ripping up and yielding non-critical resources to our delay-bounded shadow nets.

7. FUTURE WORK

As part of future work, we would like to reduce the number of critical nets that fail to shadow, possibly by relaxing the self-imposed constraint that we must preserve all aspects of the existing circuit; instead of using just leftover resources, it may be possible to move/duplicate some non-critical parts of the design in order to free up shadowing resources. Furthermore, this approach may also be necessary to enable access to both the input and output signals of each critical-path endpoint for detecting errors, which is particularly challenging for RAM and DSP hard-blocks, for example.

Another area that we would like to pursue is to explore ways to effectively insert the violation detector (XOR) logic which compares the value in the user register with the shadow register. In particular, this detection logic (and any downstream infrastructure) has the freedom to be placed anywhere on the device, as long as it does not extend T_{crit}; however, because this operation is a pairwise reduction (i.e. each user/shadow register pair must converge into the same XOR) a new approach will be necessary. Eventually, our target is to achieve 100% shadow register coverage, and be able to detect/measure all timing errors occurring on a live device.

Recently, Altera has introduced a number of architectural features into their FPGAs that we would like to explore for this shadow register application. For example, Altera provides logic clusters that support multiple clocks (whereas Xilinx only supports one per cluster) as well as fine-grained time borrowing for flip-flops [14]. Lastly, we would like to investigate how to apply negotiated congestion into our routing algorithms. Although our results show that a greedy approach works well for the vast majority of critical nets, routing negotiation may enable those last few signals to be shadowed. An open question, however, is whether incorporating congestion into edge weights would disrupt the ability for Dijkstra to find paths of monotonically increasing delay.

8. CONCLUSIONS

Shadow registers are an essential tool for detecting, measuring, and reacting to physical imperfections in silicon technology. In this work, we have presented a method for precisely inserting shadow registers into FPGA circuits. Given that FPGAs contain a prefabricated set of configurable resources, we focus on the challenge of how to attach shadow registers onto existing critical and near-critical paths at a fixed delay skew away from their original endpoint. The main contributions of this work are:

- A proposal for inserting shadow registers into an FPGA circuit post place-and-route, using only spare, leftover, resources in order to preserve circuit timing.
- A modification to the Dijkstra algorithm to achieve minimum delay bounds when routing to one, or many, potential shadow registers.
- Experimental evaluation of our techniques on a commercial Xilinx architecture.

Results show that the flexibility of the FPGA fabric, even when using leftover resources only, supports the majority of shadow registers to within $\pm$200ps of the target skew bound, as measured using the Xilinx static timing analyser. The source of this error was found to be the lack of accurate wire delay and clock skew information; when measured against our own delay model, we find that the average error is <1ps.

9. ACKNOWLEDGMENTS

The authors would like to thank Jason Anderson for his timely suggestions, and Wenwei Zha for sharing his insight on how to model wire delays. This work was supported in part by Xilinx, the U.K. EPSRC (Grants EP/I020357/1 and EP/I012036/1), Imagination Technologies, the Royal Academy of Engineering, the European Union Seventh Framework Programme under Grant Agreements 287804 and 318521, and the HiPEAC NoE.

10. REFERENCES

[1] Aeroflex Gaisler. GRLIB IP Core User's Manual. http://www.gaisler.com/products/grlib/grip.pdf, Jan. 2013.

[2] M. Agarwal, V. Balakrishnan, A. Bhuyan, B. Paul, and S. Mitra. Optimized Circuit Failure Prediction for Aging: Practicality and Promise. In *2008 IEEE International Test Conference*, pages 1–10. IEEE, Oct. 2008.

[3] Altera. Advanced Synthesis Cookbook. http://www.altera.co.uk/literature/manual/stx_cookbook.pdf, July 2011.

[4] A. Amouri and M. Tahoori. A Low-Cost Sensor for Aging and Late Transitions Detection in Modern FPGAs. In *2011 21st International Conference on Field Programmable Logic and Applications*, pages 329–335. IEEE, Sept. 2011.

[5] B. Dezs, A. Jüttner, and P. Kovács. LEMON - an Open Source C++ Graph Template Library. *Electron. Notes Theor. Comput. Sci.*, 264(5):23–45, July 2011.

[6] D. Ernst, S. Das, S. Pant, R. Rao, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. In *22nd Digital Avionics Systems Conference. Proceedings (Cat. No.03CH37449)*, pages 7–18. IEEE Comput. Soc, 2003.

[7] R. Fung, V. Betz, and W. Chow. Slack Allocation and Routing to Improve FPGA Timing While Repairing Short-Path Violations. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(4):686–697, April 2008.

[8] Y. Hara, H. Tomiyama, S. Honda, and H. Takada. Proposal and quantitative analysis of the CHStone benchmark program suite for practical C-based high-level synthesis. *Journal of Information Processing*, 17:242–254, 2009.

[9] E. Hung, F. Eslami, and S. J. E. Wilton. Escaping the Academic Sandbox: Realizing VPR Circuits on Xilinx Devices. In *Proceedings of the 21st IEEE International Symposium on Field-Programmable Custom Computing Machines*, pages 45–52, April 2013.

[10] E. Hung, A.-S. Jamal, and S. J. E. Wilton. Maximum Flow Algorithms for Maximum Observability during FPGA Debug. In *2013 International Conference on Field-Programmable Technology (FPT)*, pages 20–27, Dec 2013.

[11] E. Hung, T. Todman, and W. Luk. Transparent Insertion of Latency-Oblivious Logic onto FPGAs. In *FPL 2014, International Conference on Field-Programmable Logic and Applications*, Sept. 2014.

[12] E. Hung and S. J. E. Wilton. Towards Simulator-like Observability for FPGAs: A Virtual Overlay Network for Trace-Buffers. In *Proceedings of the 21st ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 19–28, February 2013.

[13] J. Levine, E. Stott, G. Constantinides, and P. Cheung. SMI: Slack Measurement Insertion for Online Timing Monitoring in FPGAs. In *Field Programmable Logic and Applications (FPL), 2013 23rd Int'l Conference on*, pages 1–4, Sept 2013.

[14] D. Lewis, D. Cashman, M. Chan, J. Chromczak, G. Lai, A. Lee, T. Vanderhoek, and H. Yu. Architectural Enhancements in Stratix V^{TM}. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '13, pages 147–156, 2013.

[15] L. McMurchie and C. Ebeling. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs. In *Field-Programmable Gate Arrays, 1995. Proceedings of the Third International ACM Symposium on*, pages 111–117, 1995.

[16] 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*, pages 77–86, February 2012.

[17] N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas, and M. French. Torc: Towards an Open-Source Tool Flow. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA'11, pages 41–44, February 2011.

[18] J. Y. Yen. Finding the K Shortest Loopless Paths in a Network. *Management Science*, 17(11):712–716, 1971.