Optimizing FPGA Logic Block Architectures for Arithmetic

Kevin E. Murray*, Jason Luu*, Matthew J. P. Walker*, Conor McCullough†, Sen Wang†, Safeen Huda*, Bo Yan†, Charles Chiasson*, Kenneth B. Kent†, Jason Anderson*, Jonathan Rose*, Vaughn Betz*
* Dept. of Electrical and Computer Engineering, University of Toronto
† Faculty of Computer Science, University of New Brunswick

***Abstract*—Hardened adder and carry logic is widely used in commercial FPGAs to improve the efficiency of arithmetic functions. There are many design choices and complexities associated with such hardening including: circuit design, FPGA architectural choices, and the CAD flow. However these choices have seen little study, and hence we explore a number of possibilities. We also highlight front-end elaboration optimizations that help ameliorate the restrictions placed on logic synthesis by hardened arithmetic. We show that hard adders and carry chains increase performance of simple adders by a factor of four or more, but on larger benchmark designs that contain arithmetic improve overall performance by 15%. Our results also show that for complete application circuits simple hardened ripple-carry adders perform as well as more complex carry-lookahead adders. Our best non-fracturable LUT architecture with hardened arithmetic yields 12% better area-delay product than architectures without hardened arithmetic. We also investigate the impact of fracturable LUTs and their interaction with hardened arithmetic. We find that fracturable LUTs offer significant (12-15%) area reductions, which are complementary to the delay reductions of hardened arithmetic. Therefore our best fracturable LUT architectures which use two bits of hardened arithmetic achieve 25% better area-delay product than non-fracturable LUT architectures without hardened arithmetic.**

***Index Terms*—Field programmable gate arrays, Digital arithmetic, Logic design, Design automation**

I. Introduction

A key FPGA architecture question is which functions should be *hardened* and which should be left for implementation in the soft logic [1]. Hardening a function makes an FPGA more efficient if the function occurs often in applications, and there is a large advantage when it is implemented in hard, rather than soft, logic. As adder-type arithmetic functions appear often and hardened adders are much faster than soft adders, commercial devices commonly have hardened adder and/or carry logic and routing [2]–[5].

There are many degrees of freedom in the electrical and architectural design of hard adder logic, and in the software used to map a complete application to such structures. While commercial devices use a wide variety of hardened adder circuits and architectures (indicating there is no general agreement on the best options), there has been little published work that explores the trade-offs of different hardening choices, or on the software flow used to map arithmetic to these structures. We study a number of these choices and determine their impact on the performance and area of both micro-benchmarks and complete designs. Some examples include: First, how should adders and LUTs interact? For instance, should there be fast (but less flexible) adder inputs, or are flexible (but slower) inputs coming from LUTs preferable? Second, what are the trade-offs in terms of performance and area between large, fast, multi-bit adders, and smaller, slower, but more flexible, single-bit adders? Third, should adjacent hard adder units use dedicated links for carry signals crossing soft logic block boundaries (which constrains the placement problem) or use the more flexible regular routing fabric? Fourth, how should hard adders be integrated with a fracturable LUT (a large LUT that can be split into two smaller LUTs)? Does this effect how many bits of arithmetic should be associated with each LUT? These are important questions an architect must answer when embedding hard adders with soft logic, and we present quantitative measurements of the impact of each of these decisions.

Previous work in this area began in the early 1990's, when Hsieh et al. [6] described the Xilinx 4000 FPGA that had soft logic blocks that were capable of implementing two independent adder bits per block. They employed dedicated carry logic and routing from adjacent logic blocks for the carry signals. Woo [7] proposed adding additional flexibility to the fast carry links between logic blocks to enable flexible tree-based mappings of addition/subtraction/comparison functions. Both Hseih and Woo targeted older FPGAs that had relatively fewer and smaller lookup tables in the logic block compared to the latest FPGAs.

Xing proposed implementing carry lookahead adders (in an FPGA architecture that contains just ripple adders) by using soft logic to do the carry lookahead operation [8]. His case study on the Xilinx 4000 series FPGAs show that this approach is limiting because of the large area and delay penalty that results when soft logic is involved in carry lookahead computations. Hauck [9] evaluated different implementations for FPGA adders including ripple carry, carry-skip, and tree-based adders. He showed that a Brent-Kung adder achieves a 3.8 times speedup vs. the basic ripple carry adder for 32-bit addition, at the expense of between 5 to 9.5 times more area for the adder. Parandeh-Afshar has studied the implementation of compressor trees in commercial architectures [10], and proposed adding hardened compressors to soft logic blocks to speed up multi-input addition with a focus on DSP and video applications [11]. The benchmarks used in this study appear to be on the order of a few hundred 6-LUTs [12].

FPGA vendors currently choose different hard arithmetic architectures inside their soft logic blocks. The Xilinx Ultrascale

FPGA family [13] contains a basic ripple carry architecture where addition can only start on every 8^{th} adder bit (up from every 4^{th} bit in Virtex 7 [5]). The interaction between the soft logic and the adder is flexible; the adder can either be driven by a 6-LUT and a logic block input pin or be driven by two 5-LUTs (fractured from the 6-LUT) with shared inputs. Each fracturable 6-LUT drives one bit of arithmetic. The Intel Stratix V architecture uses a two-level carry-skip adder architecture [2]. Each soft logic block contains ten 2-bit carry-skip adders that can be cascaded with dedicated links. Between two logic blocks, there is an additonal carry-skip stage that can skip 20 bits of addition. Lewis claims that this adder results in both a delay improvement and an area reduction compared to the basic ripple carry adder, as the increase in logic gates necessary for the carry-skip feature is more than offset by the area reduction made possible via transistor size optimization. Each fracturable LUT in Stratix V drives two bits of arithmetic, with each adder input driven by a 4-LUT with input sharing constraints [14]. The recent Stratix 10 family has a similar arithmetic structure but has removed the 20-bit carry skip hardware [15]. Outside of microbenchmarks, neither vendor has published, in depth, the impact of the major design decisions for their hard adder and carry chain architectures.

Prior published work on hardened arithmetic focused on the implementation of arithmetic structures, and evaluated results on microbenchmarks like adders and adder trees or very small designs. A full design, on the other hand, imposes many other demands on the FPGA and its CAD flow. We seek to measure the impact of different hard adder choices not only on microbenchmarks, but also on complete designs with a full CAD flow.

We published an earlier version of this work in [16]. This paper extends that work in two important ways by including 1) a new study on fracturable LUT architectures and 2) a new investigation of arithmetic-heavy benchmarks that exhibit characteristics in between adder microbenchmarks and general benchmarks. We also perform additional analysis of how hard adders change a circuit's timing path delay distribution.

This paper begins with a description of the base FPGA architecture (Section II), hard arithmetic structures (Section III), and CAD (Section IV) to handle the unique properties of carry chains. Afterwards, we discuss the effects of hardened arithmetic starting with the pure-adder *microbenchmarks* (Section V), arithmetic-heavy kernels (Section VI), and then full application benchmarks (Section VII).

II. Baseline Architectures

The base FPGA architecture used in this study is designed in a 22nm CMOS process, and is a heterogeneous architecture with soft logic blocks, simple I/Os, configurable memories and fracturable multipliers.

The internal connectivity of the blocks is provided by a 50% depopulated crossbar that connects block inputs and BLE outputs to the BLE inputs. We have chosen a depopulated crossbar as this is common in most commercial devices [2], [5]. The depopulated crossbar is composed of four, smaller, fully populated crossbars as designed by Chiasson [17]; this depopulation results in the soft logic block inputs being divided into four groups of ten logically equivalent pins. The input pins are evenly distributed on the bottom and the right sides of the logic block, as this simplifies the layout of the FPGA.

TABLE I
Routing architecture parameters.

Parameter	Value
Cluster input flexibility (Fc_{in})	0.2
Cluster output flexibility (Fc_{out})	0.1
Switch block flexibility (Fs)	3
Wire segment length (L)	4
Switch Block Type	Wilton
Interconnect Style	Single-driver

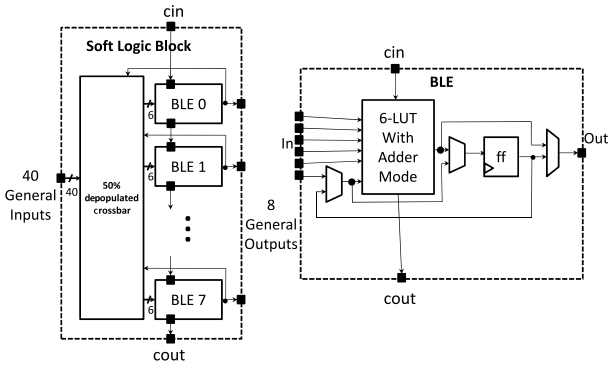


Fig. 1. The base soft logic block consists of 8 BLEs connected by a 50% depopulated crossbar. Each BLE consist of a LUT and a flip-flop with fast feedforward and feedback paths reflecting what is commonly found in state-of-the-art FPGAs.

Table I gives the routing architecture parameters of the base architecture. In addition to logic blocks the architecture includes hard 32K-bit RAM blocks (with configurable width/depth) and DSP blocks (36x36 bit multipliers which can be fractured down to two 18x18 or four 9x9 multipliers). These values are chosen to be in line with the recommendations of prior research [17], [18].

A. Non-Fracturable LUTs

Figure 1 illustrates the baseline non-fracturable soft logic block used in this study, which contains eight Basic Logic Elements (BLEs), 40 general inputs, eight general outputs, one `cin` pin and one `cout` pin. The BLE consists of a non-fracturable 6-input LUT with an optionally registered output pin. There are `cin` and `cout` pins into and out of the BLE, respectively, to drive a hard adder. The specific details are described in Section III below. There is also a fast path from the flip-flop output to the LUT input. We also consider architectures that do not contain hardened arithmetic, and hence have neither `cin` nor `cout` pins.

Our choice of following industry trends on using larger LUTs has interesting implications in terms of the efficiency of addition. When implementing arithmetic using only 4-LUTs, every bit of addition requires one LUT for the sum and another LUT for the carry. With 5-LUTs and larger, a soft implementation of arithmetic can be more efficient. Figure 2 shows how three LUTs can implement 2-bits of addition. With fracturable 6-LUTs, this benefit grows even larger as fracturing into the two 5-LUTs mode allows implementation of both the 2-bit carry and a sum operation in a single fracturable 6-LUT.

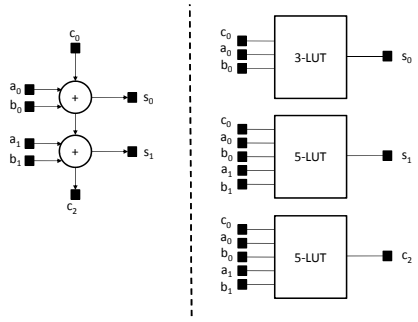


Fig. 2. 5-LUTs and larger allow for more flexibility in technology-mapping addition.

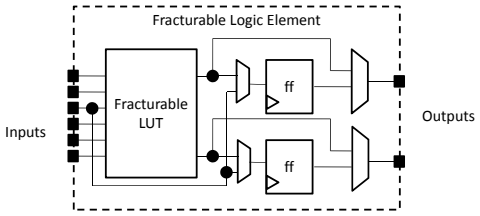


Fig. 3. A baseline fracturable Basic Logic Element (fBLE) which contains one fracturable LUT (fLUT) with optionally registered outputs.

B. Fracturable LUTs

FPGAs have traditionally used non-fracturable LUTs as described above. However many modern commercial FPGA soft logic blocks now employ fracturable LUTs (fLUTs) to obtain the performance advantages of 6-LUTs with the area advantages of 4-LUTs [19]. Some academic work has questioned whether the additional flexibility of fLUTs is worth their cost [20]. It is notable however, that [20] did not consider the impact of hardened arithmetic, which we find to be significant. Fracturable LUTs change the interaction of soft logic with hard adders and carry chains, so it is important to evaluate their combined effect.

As much as possible, our fLUT architectures reuses the same architecture as the non-fracturable case so that we may compare between these architectures. Instead of BLEs, our baseline fLUT architecture uses fracturable Basic Logic Elements (fBLEs) which, as shown in Figure 3, contains one fracturable LUT (fLUT) with optionally registered outputs. Unlike the baseline non-fracturable architecture (Figure 1), which had only one output per BLE, each fBLE has two independent outputs. Therefore the internal crossbar inside the soft logic block has an additional eight inputs (local feedbacks) compared to the non-fracturable case, and the soft logic block has 16 outputs instead of 8.

Figure 4 shows the baseline fLUT. This fLUT can operate either as one 6-LUT or two 5-LUTs with four shared inputs.

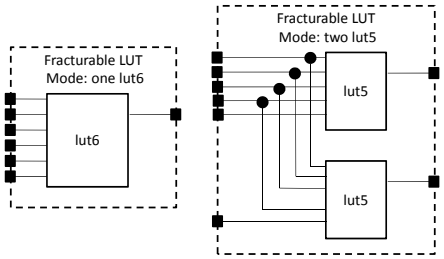


Fig. 4. Baseline fLUT which operates as either one 6-LUT or two 5-LUTs with four shared inputs.

TABLE II
DESIGN CHOICES EXPLORED

Architecture Name	Adder Prim.	Balanced	Chain Flex.	LUT Style	Bits per LUT
Soft	N/A	N/A	N/A	LUT	0
Ripple No CLB Carry	Ripple	Yes	Soft	LUT	1
CLA No CLB Carry	CLA	Yes	Soft	LUT	1
Ripple	Ripple	Yes	Hard	LUT	1
CLA	CLA	Yes	Hard	LUT	1
UCLA	CLA	No	Hard	LUT	1
URipple	Ripple	No	Hard	LUT	1
frac_soft	N/A	N/A	N/A	fLUT	0
frac_ripple	Ripple	Yes	Hard	fLUT	1
frac_2ripple	Ripple	Yes	Hard	fLUT	2
frac_uripple	Ripple	No	Hard	fLUT	1
frac_2uripple	Ripple	No	Hard	fLUT	2

This choice of shared inputs is between that of a Virtex-style fracturable LUT [13], where all 5 inputs of the 5-LUTS are shared, and an Intel-style fracturable LUT [14], where 2-inputs of the two 5-LUTs are shared.

C. Area and Delay Models

Transistor-level design of the base soft logic blocks and routing architecture are performed with the COFFE tool [17] and a 22nm CMOS technology. The architecture uses pass gates; statically controlled pass gates are gate-boosted by 0.2V [21]. The architecture, area, and delay models for the memories and multipliers are scaled to 22nm from the comprehensive 40nm architecture in the VTR 7.0 release.

III. HARD ADDER AND CARRY CHAIN ARCHITECTURES

To evaluate the impact of including hard adders in FPGA logic blocks we explore different design choices relating to adder implementation and interaction with the rest of the logic block. Table II summarizes the different design choices, which are described in detail below.

A. Adder Primitive

To ensure we fairly compare various hard adder and carry chain architectures, we carefully electrically designed two hard adder primitives and hand optimized them at the transistor level. The first adder primitive is a basic 1-bit full adder. In a soft logic block, eight of these full adders are linearly chained together to form a ripple carry chain. Table III shows the properties of the 1-bit hard full adder used in this study. Area is measured as minimum width transistor areas (MWTAs), using the transistor drive to area conversion equations from Chiasson [17]. The adder circuitry, LUTs and routing are all designed with a similar goal of minimizing the area-delay product of the FPGA, and the `cin` to `cout` path of the adder is particularly optimized for delay as it occurs n-1 times on an n-bit adder.

The second adder primitive is a 4-bit carry-lookahead adder (CLA). Each logic block contains two of these 4-bit adders chained in a ripple carry fashion. Table IV shows the properties of the 4-bit carry lookahead adder used in this study. The carry lookahead optimization allows for a faster carry path (20 ps) compared to a ripple of four 1-bit adders (44 ps) when performing a 4-bit addition. The CLA design trades off flexibility (as some bits are wasted if the desired adder length is not divisible by 4) and area in exchange for speed.

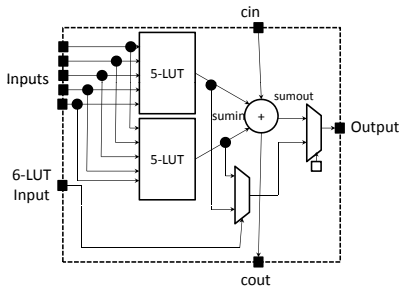


Fig. 5. A LUT with *balanced* adder interaction where both adder inputs are driven by 5-LUTs.

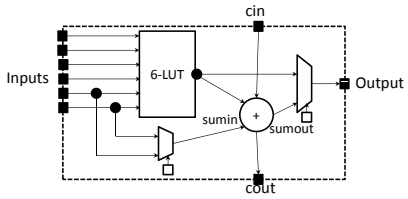


Fig. 6. A LUT with *unbalanced* adder interaction where the 6-LUT drives only one adder input.

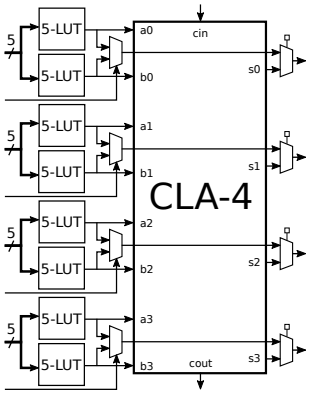


Fig. 7. 4-bit CLA with *balanced* LUT interaction.

TABLE III
PROPERTIES OF THE 1-BIT HARD ADDER.

Property	Value
Area	47.7 MWTAs
Delay cin to cout	11 ps
Delay sumin to cout	56 ps
Delay cin to sumout	30 ps
Delay sumin to sumout	83 ps

TABLE IV
PROPERTIES OF THE 4-BIT CARRY LOOKAHEAD ADDER.

Property	Value
Area	257 MWTAs
Delay cin to cout	20 ps
Delay sumin to cout	80 ps
Delay cin to sumout LSB	25 ps
Delay cin to sumout MSB	30 ps
Delay sumin to sumout LSB	65 ps
Delay sumin to sumout MSB	82 ps

B. Adder Input Balancing

Figure 5 shows one way the LUT and adder (within a BLE) may interact. Here, we make use of the observation that a 6-LUT is constructed with two 5-LUTs and a mux. If that mux is dropped, then the adder can be driven by two 5-LUTs, where the LUTs share inputs. If the adder is not used, then another mux can be used to produce the 6-LUT output. We call this the *balanced* LUT interaction, and its underlying rationale is that a symmetric amount of prior logic for each adder input may be the most appropriate architecture. Example circuits that may benefit from this architecture would be applications where multiplexers select the inputs to an adder. Similar interaction for the CLA is shown in Figure 7.

Figure 6 shows another LUT-adder interaction architecture that we will explore. Here, the 6-LUT output drives one of the adder inputs and the other adder input is driven by one of the 6-LUT *inputs*. As with the previous case, if the adder is not used, then another mux can be used to select the 6-LUT output. We call this the *unbalanced* LUT interaction. We model each additional SRAM-controlled 2-to-1 mux (one per BLE for the balanced LUT interaction, two per BLE for the unbalanced LUT interaction) as having 22 ps of delay and occupying 15 minimum width transistor areas (including the SRAM configuration bit). The underlying rationale for this architecture is that there might be an advantage to allowing a faster input into one side of the adder, which would be appropriate when speed was an issue.

C. Carry Chain Flexibility

Another class of interesting architectures are those with hardened adders but no dedicated carry link between logic blocks. Here, both the `cin` and `cout` pin are treated as though they are regular input and output pins, respectively, in the inter-block routing architecture. Within the logic block, the carry signals maintain the same restricted connections. For architectures that have a dedicated carry link, the carry link has a delay of 20 ps. For those without a dedicated `cin`/`cout` we add the usual circuitry to allow them to access the right and bottom side channels of the logic block.

There are a few different ways to implement the starting location of a multi-bit addition. One can place a mux at every carry link that can select from logic-0, logic-1, or a carry signal of a previous stage, but this can incur a significant delay penalty because every carry link must now go through a mux. Alternatively, one can place these muxes only on selected carry links, thus minimizing the overhead of excessive muxing, but at the cost of having fewer locations where an addition may begin. This latter approach is typical in commercial devices. Alternatively, the responsibility for starting an addition can be implemented in a front-end CAD tool – the tool can pad the addition with a dummy adder before the LSB (whose addends are fed by constants) which generates a `0` or a `1` `cin` for addition and subtraction, respectively. We employ this approach in this work.

D. Bits of Addition per LUT

For fLUT architectures each fLUT can produce two independent outputs. This makes it possible to include a second bit of addition in the fLUT with minimal cost. Figure 8

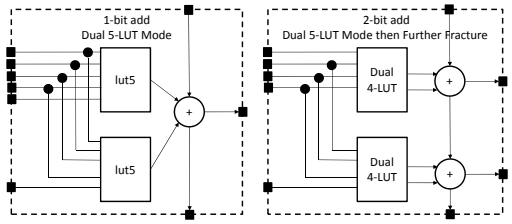


Fig. 8. fLUT with adder 1-bit vs. 2-bit balanced.

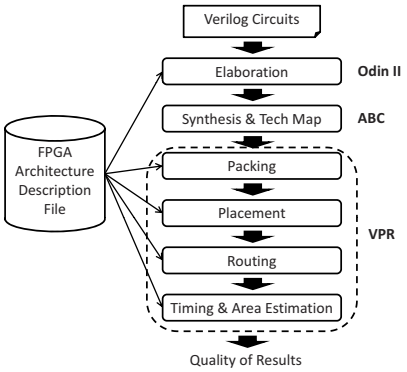


Fig. 9. The VTR CAD flow

contrasts the balanced 1-bit adder fLUT on the left with the balanced 2-bit adder fLUT on the right. The balanced 2-bit adder requires that each 5-LUT further fracture down to two 4-LUTs, with most inputs shared, in order to supply the requisite four addends to the adder.

IV. CAD

In this section, we describe the CAD tools we use and the significant enhancements they required to explore the architectures described in the previous sections. We can not use commercial FPGA CAD tools to evaluate our proposed architectures, as they are closed-source (and hence can not be modified), and do not support re-targeting to proposed FPGA architectures. Instead, we employ the VTR 7.0 [22] CAD flow, which can target a wide range of user-described FPGA architectures. The VTR CAD flow is open-source which allows us to modify and enhance it to ensure it optimizes well for the architectures we will evaluate.

The VTR CAD flow is illustrated in Figure 9. The two key inputs are a circuit described in Verilog and a description of the FPGA architecture in a human-readable text file. The circuit is elaborated by Odin II and ABC [23] performs logic synthesis to produce a technology-mapped netlist of device atoms such as LUTs, FFs and basic multipliers. VPR then packs these atoms into logic, RAM and DSP blocks, places those blocks, and routes connections between them. Finally, VPR computes the area and delay of that final, physical mapping. Below we detail the modifications and enhancements required to enable hardened adders and carry chains.

A. Elaboration and Logic Optimization

In our initial experiments targeting hardened adders, we discovered a surprising and unexpected downside: when front-end elaboration inserts hardened adders into the circuit, it creates a boundary in the elaborated circuit that cannot be crossed by ABC's logic synthesis. Furthermore, the hardened logic is a "black box" and hence invisible to ABC and cannot be optimized. This boundary reduces the effectiveness of basic logic synthesis optimizations such as common sub-expression elimination. We observed that ABC was able to reduce the number of soft adders when these boundaries were not in place, and that multiple copies of adders with the same inputs were left intact when hardened adders were used. We also attempted to use the "white box" feature of ABC [24]; while this made the functionality of the hard logic visible, it also led to ABC converting it into regular soft logic and hence was not suitable.

TABLE V
EFFECT OF ODIN II OPTIMIZATIONS. ALL VALUES ARE NORMALIZED TO THE BASE CASE WITH NO OPTIMIZATIONS.

Circuit	DHR CLB	ULR CLB	Both CLB	Both Delay
arm_core	0.97	0.95	0.94	0.92
bgm	1.00	0.80	0.80	0.87
blob_merge	0.91	0.99	0.91	0.55
boundtop	0.92	0.99	0.90	1.00
LU8PEEng	0.84	0.99	0.83	1.01
LU32PEEng	0.83	0.99	0.82	0.98
mcml	1.00	0.91	0.91	0.94
mkSMAdapter4B	1.00	0.89	0.89	0.92
or1200	0.90	0.92	0.86	1.09
raygentop	0.93	0.94	0.87	0.92
sha	1.00	0.99	0.99	1.03
stereovision0	1.00	0.80	0.80	0.95
stereovision1	0.99	0.79	0.79	0.98
stereovision2	1.00	0.97	0.97	1.01
geomean	0.95	0.92	0.88	0.93
stdev	0.06	0.08	0.06	0.13

To compensate for reduced down-stream optimization, we implemented two new optimizations in Odin II: the removal of duplicate hard adders and unused logic removal. Both these optimizations are generalized to all hard blocks and are not exclusive to hard adders. Duplicate hard block reduction is a simplified version of common sub-expression elimination. If all of the input pins of any two hard blocks anywhere in the circuit are found to be the same, the duplicate hard block can be removed and its fanout attached to the other hard block.

In a typical CAD flow, logic synthesis is responsible for sweeping away unused logic because synthesis optimizations can sometimes reveal unused logic. ABC is unable to do this for hard blocks as it optimizes exclusively based on logic expressions, and views hard blocks as black boxes. Hence we augmented Odin II to sweep away unused hard and soft logic based purely on circuit connectivity.

We quantified the impact of the new optimizations, using the experimental methodology described subsequently in Section VII but with adders always hardened. These experiments covered four cases for the optimization settings in Odin II: *None*, *DHR* (duplicate hard logic removal), *ULR* (unused logic removal), and *All* (both DHR and ULR enabled).

Table V shows the impact of these optimizations on the benchmark circuits described in Section VII; most circuits benefit from both optimizations. In this work we use the geometric average to summarize results, as it equally weights each benchmark. On average, duplicate hard logic removal and unused logic removal reduce the logic blocks required by a circuit by 5% and 8%, respectively, while their combination reduces logic blocks required by 12% and the critical path delay by 7%.

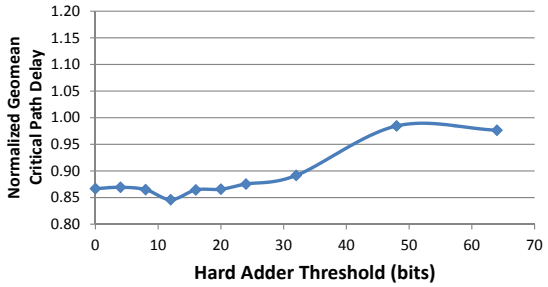


Fig. 10. Circuit speed vs. hard adder threshold. Results are the average across 14 benchmarks and normalized to the soft implementation.

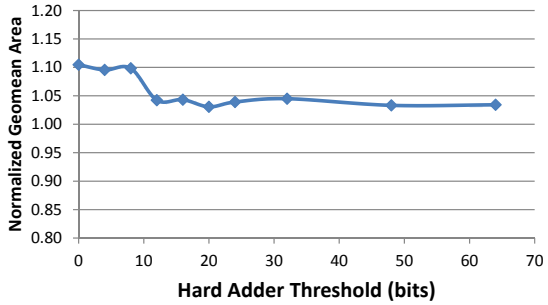


Fig. 11. Average total area of different hard adder thresholds normalized to the soft architecture.

B. Threshold of When to Use Hard Adders

While using hardened adder and carry logic is clearly beneficial for wide arithmetic structures, for small adders the flexibility provided by soft logic might actually prove superior as hard adders impose a boundary across which it is difficult for logic synthesis to optimize. We define the *hard adder threshold* as the size, in bits, of addition/subtraction above which the CAD flow will implement it with hard adders and below/equal to which the function is implemented in soft logic.

Figure 10 shows the impact on delay of different hard adder thresholds when we target the ripple carry architecture. The x-axis shows the hard adder threshold in bits. The y-axis shows the geometric mean of the delay over the 14 circuits of Table V. There is a general trend towards achieving a minimum mean delay at a threshold of around 12 bits.

Figure 11 shows the area impact of different hard adder thresholds. The x-axis is again the hard adder threshold, while the y-axis shows geometric mean of the total area for all benchmarks. The area consumed using an architecture with hard adders is on average more than that of an equivalent architecture without carry chains. We see a gradual drop in area with an increasing hard adder threshold; area drops from 10% above the soft adder architecture with a hard adder threshold of 0, to 3% above with a threshold of 12. Interestingly, preliminary measurements we made on commercial FPGAs showed using carry chains in the CAD flow reduced area; we therefore suspect that with further improvements in logic synthesis the remaining 3% area penalty could be eliminated.

Considering area and delay, the best hard adder threshold is approximately 12 bits. This threshold is used for all subsequent results unless otherwise noted.

C. Packing

The packing stage of the CAD flow is responsible for grouping technology-mapped atoms such as LUTs, hard adder bits, flip-flops, and memory slices, into complex logic blocks.

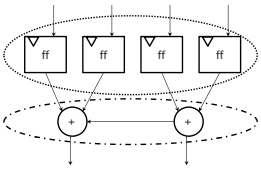


Fig. 12. Example of transitive connections.

When a logic block contains carry chains, adder atoms must be placed inside the logic block in an order that respects the restrictive carry links. The packer should also make use of the architecture-specific features that allow the LUTs and flip-flops to interact with the adder.

AAPack, the packer inside VPR 7.0 is an interconnect-aware packing algorithm [25] that recognizes and respects the various pin constraints that arise with different LUT and adder interactions. The carry chain itself is specified using the "molecule" feature in AAPack that allows the architect to specify how certain atoms *must* be packed together.

In our initial experiments with microbenchmarks (mostly pure adders fed by, and feeding into registers), we discovered that the packing algorithm in VPR 7.0 was imperfect in a number of cases. Figure 12 shows an example of the simple input circuits that caused a problem. The adders in this figure form a carry chain so they will be packed together into a logic block. If the flip-flops cannot be packed into the same logic block as the adders, then the packer will see these flip-flops as completely unrelated to each other because they do not share common nets. These flip-flops may then be separated and packed with other logic, which is undesirable as it makes it impossible to place all the logic clusters containing these registers close to the adder. We modified the packer to consider atoms that have *transitive connectivity* with the current logic block being packed. In this particular example, the flip-flops that drive the adder are transitively connected via the carry chain so the packer gives them higher attraction to each other than to other unconnected logic. With this modification, circuits such as that illustrated in Figure 12 were packed well.

D. Placement and Routing

VPR 7.0 has place-and-route functionality for carry chain exploration. The architecture description file allows any logic block pin to connect to the general inter-block routing and/or to use a special dedicated connection to a specific pin on a specific other logic block. In this work, when we explore dedicated carry chain links they run vertically – the cout pin of one logic block can only connect to the cin pin of the logic block immediately below it. When such dedicated connections are specified, VPR 7.0 automatically not only generates the appropriate edges in the routing resource graph to represent this direct routing possibility, but also constrains the placement algorithm to keep any blocks that are part of a hardened carry chain in the correct relative position (in our case vertically adjacent) throughout placement.

For logic blocks without hard carry chains, it is possible to change which BLE performs which function during the routing stage of the CAD flow – making the outputs *logically equivalent*. However if hard carry chains are used the order of BLEs is fixed, and the outputs of BLEs using their carry

function are not logically equivalent. VPR 7.0, does not allow us to selectively switch off output pin logical equivalence in cases when the carry links are used by the BLEs. Hence, for correctness, we do not allow any BLE swaps, thus removing all output logical equivalence. Turning off logical equivalence for all outputs will lead to a slight pessimism on the routability of the soft logic only architecture vs. that of the hard adder architectures. To quantify this we evaluated the soft logic only architecture both with and without output equivalence enabled, and found the impact on delay and area was $< 3\%$ on the VTR benchmarks. Since the effect is small (compared to the impact of architectural modifications) this restriction will not change the architecture conclusions. Furthermore, to compensate for the restriction, each output pin can directly access two sides of the logic block, and hence both a vertical and a horizontal channel, ensuring the pins still have access to a diverse set of routing wires.

V. Microbenchmark Results

In this section we explore the impact of the different ways of supporting arithmetic in an FPGA architecture when evaluated on simple adder microbenchmarks. Here, each circuit is an adder of N bits, where N ranges from one to 127. Both the inputs and outputs of the adder are registered, so the critical path delay measured is a direct function of the adder combinational logic delay. These registered adders are implemented using the flow described in Section IV. In this section the hard adder threshold (Section IV-B) is disabled in order to measure the impact of the different architectures at small bit widths.

Figure 13 shows the impact on critical path delay vs. width of addition, for the Soft, Ripple and CLA architectures, where the critical path delay is averaged over three placement seeds. In addition, two variants of the Ripple and CLA architectures are included, labelled *no CLB carry*, in which the general-purpose interconnect is used to implement carry links between soft logic blocks, rather than using dedicated carry links. Figure 14 shows the results for the fracturable LUT architectures. The unbalanced architectures are not included here as their performance difference vs. balanced is negligible on the microbenchmarks.

These results show trends that we generally expect, in that delay grows linearly with adder size, and that the more hardened architectures are faster.

In the extreme case, for 127 bit addition, it is interesting to note that a pure soft adder is ten times slower than the fastest (CLA) adder. For 32-bit addition, the hard adders provide a 3.4-fold speedup over the soft adder. The *no CLB carry* architectures have delay values in between fully hard and fully soft adder architectures. While the CLA architecture is the fastest of all, ripple carry is only 19% slower for 32 bit adders, and 42% slower even for 127 bit addition. A ripple architecture can sustain 400 MHz operation for even a 96-bit addition.

When adders are implemented in soft logic, CAD noise can have a significant impact on delay. The effect of this noise is evident in the figure when observing the delay of additions ranging from 17 bits to 25 bits for the soft logic architecture, where delays for additions of similar size can vary significantly as a result of CAD (in this case packer)

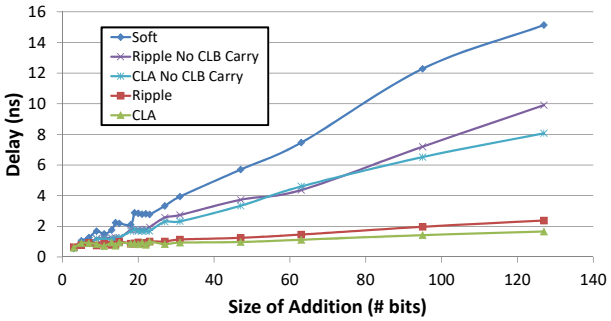


Fig. 13. Delay vs. adder length for various non-fracturable architectures.

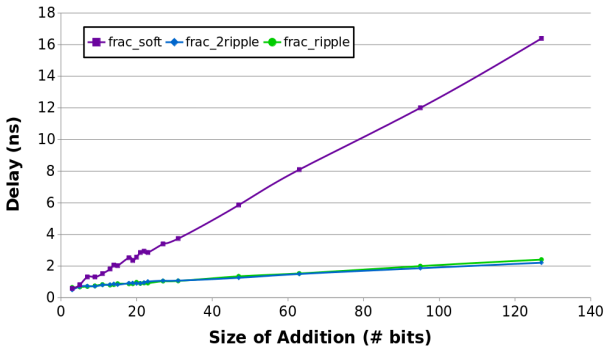


Fig. 14. Delay vs. adder length for various architectures with fracturable LUTs.

noise. For hard adders, the lack of CAD flexibility forces a predictable physical design, thus greatly reducing CAD noise for these microbenchmarks. The combination of higher and more predictable performance provided by hard adders, especially those with hard inter-CLB links, is very desirable.

The data from this experiment also shows that a 3-bit addition implemented in soft logic is actually slightly faster than any of the hard-logic adders, further motivating the CAD hard adder thresholds described in Section IV-B.

Table VI shows the tile area for a logic block (including both inter and intra-block routing) in each architecture. Here we observe that the inclusion of hard adders increase tile area by $< 2.5\%$ over their respective baseline architectures. The delay, logic block count and area for implementing a 32-bit adder are also shown in Table VI. The architectures with hard adders are all substantially faster than the soft architectures, they also use fewer CLBs than the baseline *Soft* architecture and hence (with the exception of *frac_ripple* and *frac_uripple*) are more area efficient.

TABLE VI
32-bit Adder Delay & Area.

Architecture	Tile Area (10^3 MTWA)	32-bit Addition		
		Delay (ns)	CLBs	Total Area (10^3 MTWA)
soft	19.84	3.94	18	357.06
cla	20.25	0.93	16	324.05
ucla	19.92	0.83	16	318.77
ripple	20.14	1.14	16	322.25
uripple	19.90	1.03	16	318.34
frac_soft	23.06	3.74	14	322.77
frac_ripple	23.06	1.06	16	368.92
frac_uripple	23.37	1.15	16	373.86
frac_2ripple	23.40	1.08	14	327.64
frac_2uripple	23.62	1.01	14	339.71

Tile Area is the geomean across application benchmarks (Section VII) and so includes both logic and realistic routing area.

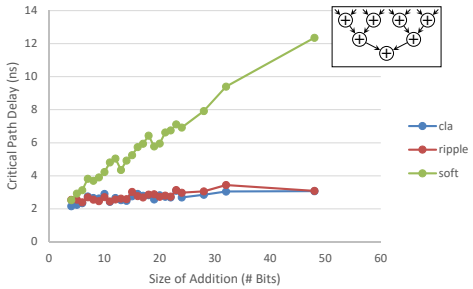


Fig. 15. Delays across the architectures on depth 3 adder trees.

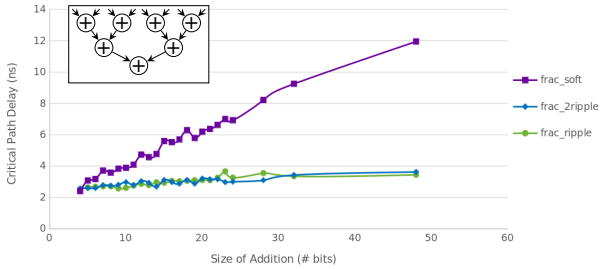


Fig. 16. Delays across the architectures on depth 3 adder trees.

VI. Arithmetic Kernel Results

This section explores the impact of hardened adders and carry chains on arithmetic kernels that are more complex than a pure adder, but would still only be part of a full user application. Intuitively, we expect that when a circuit becomes more complex, with interactions between different types of logic, then the differences between the different architectures may be reduced. The hard adder threshold (Section IV-B) is disabled in this section to enable evaluation of the different architectures at small bit widths.

A. Adder Trees

Adder trees are frequently used to implement functions like filters and dot products. Figures 15 and 16 shows the performance of a depth three adder tree (which can add 8 numbers) for non-fracturable and fracturable architectures respectively. The critical path now consists of the concatenation of several ripple carry stages and 2 or 3 input to sumout stages. Each LUT before an adder input must be configured as a wire, so the LUT is wasted. The performance of the balanced and unbalanced architectures are the same; hence only balanced is shown. The hard carry lookahead adder is no faster than the ripple adder for these adder trees, as now the sum-generation logic has a larger impact. All the hard implementations are faster than the soft implementation for adders of 4 or more bits, with the gain widening to 4x at 48 bits for depth-3 adder trees. The soft implementation uses essentially the same number of logic blocks as the hard versions for adder trees with 8-bit or smaller additions, as the adder trees provide more opportunities for a synthesis tool to make use of the 6-LUTs than a simple adder does. Again this highlights that using soft logic for small adders is a very reasonable choice. However, for large additions of 48-bits, soft logic requires 67% more logic blocks than the hard architectures.

B. Multiplication by a Constant

Another important arithmetic kernel is multiplication by a constant, which can be efficiently implemented using addition, subtraction and wired-shifts. These microbenchmarks implement 16-bit multiplication using different constants; the inputs and outputs to the combinational multiplier are registered.

TABLE VII
Geometric mean of delay and number of CLBs for constant multiplication (128 randomly selected constants).

Arch	Crit Delay (ns)	Num CLB
soft	6.68	21.9
cla	3.07	17.8
ucla	2.95	17.8
ripple	3.28	17.5
uripple	3.14	17.5
frac_soft	6.62	15.4
frac_ripple	3.39	18.3
frac_uripple	3.70	20.0
frac_2ripple	3.54	10.1
frac_2uripple	3.79	12.5

Table VII shows the geometric average of performance and number of CLBs for different architectures across 128 multiply-by-constant circuits. The soft adder architecture is over twice as slow as the hard adder architectures, the same ratio as that of the 16-bit pure adder benchmark. The carry-lookahead architectures are 7% faster than the single-bit adder ones, again matching the 16-bit pure adder benchmark. The unbalanced hard adder architectures are slightly faster than the balanced architectures, as the constant multiplier circuits have some critical paths that are longer than others and the unbalanced architecture can give slightly faster resources to these paths. Hard adders reduce CLB count versus soft adders by approximately 24%.

Both the fracturable and non-fracturable architectures with hard adders improve critical path delay. However the results for logic block counts are quite different. The soft fracturable architecture uses fewer logic blocks than the non-fracturable soft architecture and less than the non-fracturable architectures with hardend adders. Interestingly, the 1-bit fracturable LUT architectures use more logic blocks. This is due to the complexity of the logic block causing more registered input opportunities to be missed by the CAD tool. However, the 2-bit fracturable LUT architectures are substantially more efficient, using significantly fewer logic blocks than all other architectures (up to 42% compared to non-fracturable hardened architectures and 34% compared to the baseline fracturable architecture) while achieving similar delay.

C. FIR Filters

FIR filters are widely used in DSP applications, are rich with arithmetic, and are well suited for implementation on FPGAs. We study the impact of hardened adders and carry chains on direct-form FIR filters with 18-bit word widths that consist of shift registers of input data feeding DSP blocks that in turn feed adder trees. We vary the number of taps on the filter and compare the results of pipelining vs. no pipelining in the adder tree.

Figures 17 and 18 show the effect on critical path delay as we increase the number of taps of a combinational FIR filter for non-fracturable and fracturable architectures respectively. Only the balanced hard architectures are shown as the unbalanced variants have essentially the same performance. Figures 19 and 20 shows the same effect with FIR filters with deeply pipelined adder trees. For the non-pipelined filters, hard adders are substantially faster than soft adders, but the various hard adder architectures show almost identical performance.

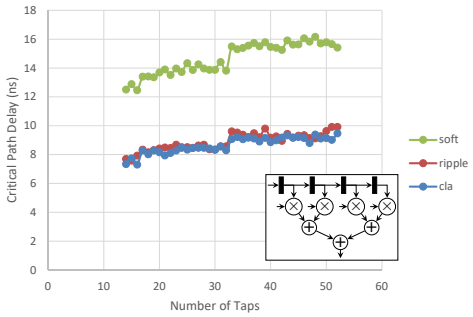


Fig. 17. Delays across the architectures on non-pipelined FIR filters.

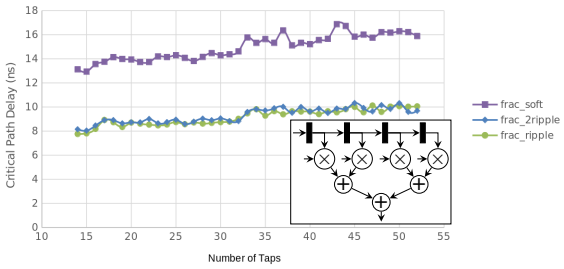


Fig. 18. Delays across the architectures on FIR filters with no pipelining.

However the gap between soft and hard adders is not as large for the fracturable architectures. With the pipelined filters, hardened carry chains do not improve speed over soft arithmetic as the critical path has moved into the multipliers.

Figures 21 and 22 shows the effect on logic block count as we increase the number of taps of non-pipelined FIR filters for the non-fracturable and fracturable architectures respectively. Figures 23 and 24 shows the same effect with pipelining. Architectures with hard adders show lower CLB counts than soft adders, but the absolute CLB counts are quite different between the fracturable and non-fracturable architectures. Interestingly both *frac_soft* and *frac_ripple* are quite close in CLB count, but both are substantially lower than the non-fracturable architectures. This underscores our

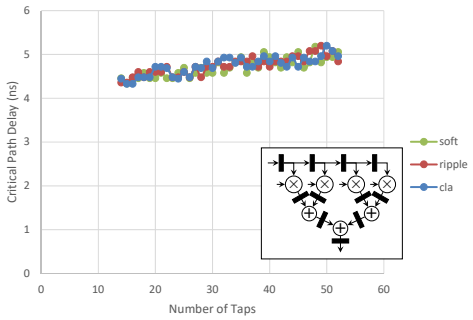


Fig. 19. Delays across the architectures on FIR filters with pipelined adder trees.

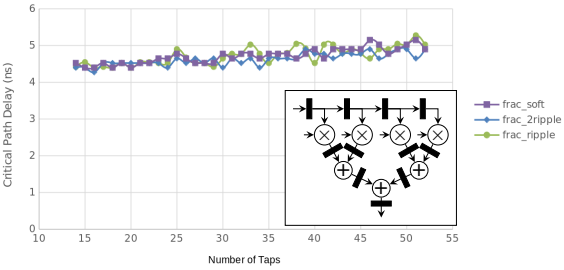


Fig. 20. Delays across the architectures on FIR filters with pipelining.

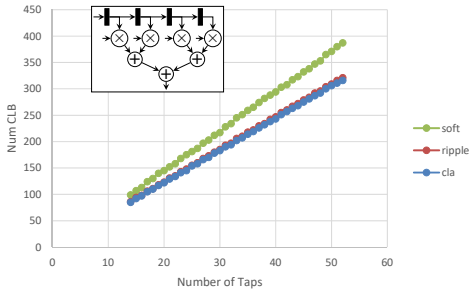


Fig. 21. Logic block counts across the architectures on non-pipelined FIR filters.

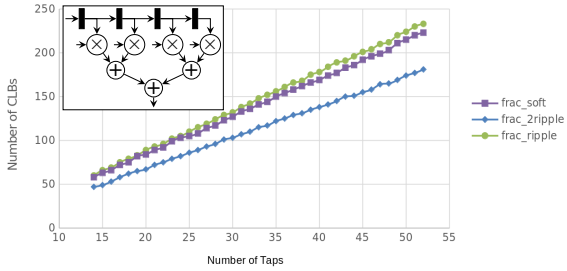


Fig. 22. Logic block counts across the architectures on FIR filters with no pipelining.

earlier observation that fracturable LUTs are much more area efficient at implementing soft adders than non-fracturable, thus lowering the area curve. Notably, using 2-bits of addition per fLUT is again the most efficient. This indicates that two bits of arithmetic achieves better utilization of the fLUT in front of the adders than the one bit case.

VII. Application Circuit Results

We now focus on evaluating the different hard adder variants using full application benchmarks. These allow us to evaluate the overall quality of the different implementation approaches, considering overall delay and area. Architectures which minimize the area-delay product are the most efficient. The complete design benchmarks we use are from the VTR 7.0

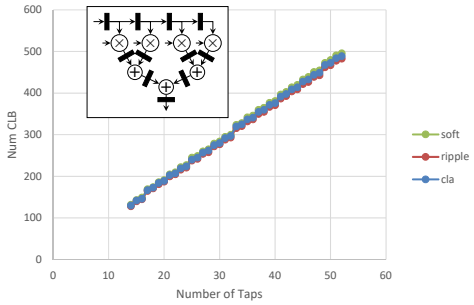


Fig. 23. Logic block counts across the architectures on pipelined FIR filters.

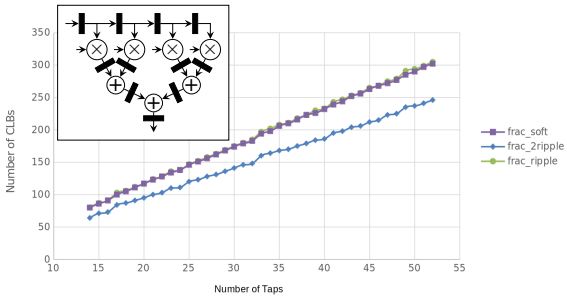


Fig. 24. Logic block counts across the architectures on FIR filters with pipelining.

TABLE VIII
BENCHMARK STATISTICS WHEN MAPPED TO RIPPLE ARCHITECTURE.

Circuit	Num 6LUTs	Max Add Len	Avg Add Len	Add/LUT Ratio
bgm	5438	25	9.3	0.17
blob_merge	3754	13	12.0	0.48
boundtop	309	19	7.2	0.11
LU8PEEng	3241	47	11.1	0.15
LU32PEEng	8235	47	11.9	0.11
mcml	24302	65	47.5	0.26
mkSMAdapter4B	431	33	6.9	0.24
or1200	534	65	23.9	0.19
raygentop	580	32	11.8	0.33
sha	309	33	24.0	0.15
stereovision0	2920	18	11.2	0.35
stereovision1	2388	19	6.4	0.30
stereovision2	13843	32	23.9	1.26
geomean	2109	31	13	0.25

release, specifically, all circuits larger than 1000 6-LUTs[1]. We will refer to these as the VTR+ benchmarks. The geometric average primitive count across the 13 circuits is 11,700. The benchmarks are full application circuits, which include a variety of arithmetic operations including addition, subtraction, and multiplication which can make use of hardened adders.

Table VIII provides statistics on these benchmarks, and includes the number of addition/subtraction functions found in the benchmarks on the *Ripple* architecture. The table columns list the number of 6-LUTs, the length of the longest adder chain in bits, the average adder chain length, and the ratio of adder bits to LUTs. The benchmarks exhibit a wide range in the number and length of addition/subtraction functions. The geometric mean of the ratio of adder bits divided by the number of 6-LUTs is 0.25, indicating arithmetic is plentiful and hence it is reasonable to include hard adder circuitry in *every* CLB. The widest addition/subtraction generated in these benchmarks is 65 bits, which corresponds to a 64-bit operation (as the first bit must always be used to generate the carry-in signal). The geometric mean of the longest addition/subtraction lengths is 31 bits. The most adder-intensive circuit is $stereovision2$ with 1.26 adders per LUT. These measurements correspond well with other modern benchmarks. For the Titan benchmarks (with the SPARC cores excluded because they have almost no adders at all) [26], the geometric average of the fraction of LUTs in arithmetic mode and the maximum of length of addition/subtraction are 0.22 and 35.8, respectively.

We use the standard VTR 7.0 CAD flow, augmented as described in Section IV, to determine the minimum routable channel width (W_{min}) for each circuit. The router is then invoked again with a channel width of $1.3 \times W_{min}$ to measure critical path delay and area. Area measurements are in minimum-width-transistor-area units. Area is computed as the total number of soft logic blocks (CLBs) multiplied by the area of a soft logic tile, where this tile includes both the logic cluster and inter-cluster interconnect area. The hard adder threshold is set to 12, as this yields the best area-delay results in Section IV-B. Each of the circuits was mapped to the soft logic and hard carry chain architectures described in Table II.

[1]A large ARM processor core is also included, and the mkDelayWorker32B benchmark is excluded as it caused ABC to crash.

TABLE IX
DELAY FOR DIFFERENT HARD ADDER ARCHITECTURES, NORMALIZED TO THE SOFT LOGIC ARCHITECTURE.

Arch	32-bit Add Delay	Application Circuits Delay
Ripple	0.29	0.85
U-Ripple	0.26	0.87
CLA	0.24	0.85
U-CLA	0.21	0.85

TABLE X
QOR OF THE VTR+ BENCHMARKS ON DIFFERENT CARRY CHAIN ARCHITECTURES. VALUES ARE THE GEOMETRIC MEAN OF VTR+ CIRCUITS NORMALIZED TO THE SOFT ADDER ARCHITECTURE.

Arch	Area	Min W	Num CLB	Delay	Area-Delay Product
Ripple	1.04	0.97	1.03	0.85	0.88
URipple	1.03	0.92	1.03	0.87	0.90
CLA	1.06	0.96	1.04	0.85	0.90
UCLA	1.04	0.91	1.03	0.85	0.88

A. Contrasts between Microbenchmarks, Kernels & Applications

An interesting first comparison is to assess the impact of hard adders on application circuits vs. microbenchmarks. We use a 32-bit adder as a representative microbenchmark, as this is close to the average size of the longest adders in the application circuits. Table IX shows the geometric average critical path delay for each of the non-fracturable architectures normalized to the non-fracturable soft logic architecture. An isolated 32-bit adder sees a compelling delay reduction of 71% to 79% with hard carry architectures, while application circuits see much smaller (but still very significant) delay reductions of 13% to 15%, depending on the hard carry architecture. The arithmetic kernels mostly had delay reductions between these extremes: from no delay reduction for deeply pipelined FIR filters to 50% for constant multiplication and non-pipelined FIR filters, to as much as 80% for adder trees. This is a common outcome in the hardening of any kind of circuit – the final impact on critical path delay in more complex applications is limited because other paths in the design quickly become more critical than the adder. On the application circuits, the best delay improvement achieved by hardening adders is 15%, for the U-CLA architecture. Observe, however, that the other simpler hardened adder architectures benefit circuit speed almost as much.

Table X shows the quality of results (QoR) of each architecture normalized to the soft logic architecture. All values are the geometric averages across all application benchmarks, normalized to the soft adder architecture. The columns from left to right are the architecture, the total soft logic area including routing, minimum channel width, number of used soft logic blocks, critical path delay, and area-delay product. The CLA architecture increases area slightly (by between 1% and 2%) compared to ripple, but cuts delay by roughly the same amount, leading to an area-delay product that is very close to that of the ripple architectures.

B. Circuit-by-Circuit Analysis

Table XI provides a circuit-by-circuit breakdown comparing the U-CLA and Soft architectures. The columns from left to right are the benchmark name followed by the ratio of the U-CLA/Soft values for critical path delay, the total soft logic area including routing, and the number of LUTs on the critical path.

TABLE XI
CIRCUIT-BY-CIRCUIT BREAKDOWN COMPARING THE U-CLA ARCHITECTURE TO THE SOFT ARCHITECTURE.

Circuit	Delay	Area	LUTs on Crit Path	CLA cout on Crit Path
LU32PEEng	0.76	1.05	0.62	26
mcml	0.55	1.05	0.25	144
or1200	0.65	1.11	0.15	19
sha	0.57	1.04	0.25	10
stereovision2	0.85	0.77	0.07	1
blob_merge	0.97	1.04	0.89	0
boundtop	0.95	1.02	0.89	0
LU8PEEng	0.82	1.04	0.64	0
mkSMAdapter4B	1.01	1.11	0.86	0
bgm	1.17	1.13	1.00	0
raygentop	0.97	1.11	N/A	0
stereovision0	0.94	1.02	1.00	0
stereovision1	1.07	1.10	N/A	0
geomean	0.85	1.04	0.46	–
stdev	0.19	0.09	0.36	–

The last column is the number of hard adders on the critical path for the U-CLA architecture. On average, the delay of the circuits is reduced by 15% and the critical path LUT depth is cut by more than 50%, but there are 3 distinct classes of circuits that show markedly different behaviour.

For the top 5 circuits hard adders are on the critical path, and we obtain a large delay reduction of 33%. It is interesting to note that despite having the lowest adder to LUT ratio (11% in Table VIII), the LU32PEEng design still obtains a significant 24% delay reduction. Clearly, even when adders are a smaller portion of the design logic addition operations can still be timing-critical. This illustrates that even circuits with relatively low arithmetic intensity still benefit from hard adders.

The next 4 circuits (blob_merge, boundtop, and LU8PEEng, and mkSMAdapter4B) have reductions in the critical path LUT depth of 19% when targeting the U-CLA architecture, even though no hard adders occur on their critical paths. This indicates that adder logic was likely timing critical in the Soft architecture[2], but has sped up enough to move off the critical path in the U-CLA architecture. Interestingly, while these 4 circuits have an average LUT depth that is 19% lower when targeting U-CLA vs. Soft, the average delay reduction across the 3 designs is only 7%. We believe this illustrates an interesting trade-off when hard carry chains are added to an FPGA: by limiting the flexibility of the packer and placer, the carry chains have increased the average routing delay per LUT level on non-adder paths, and this costs some of the speed gain one would expect from reducing the logic on the critical path with hard adders.

Finally, there are four circuits where the LUT depth is not significantly reduced and where there is not a significant delay reduction, indicating adders were not very timing-critical in even the Soft architecture. Two of these circuits (raygentop and stereovision1) have hard multipliers as their critical paths so they show less variation in speed vs. carry architecture, as one would expect.

C. Impact of Hard Adders on Path Delay Distribution

As noted above, the introduction of fast hard adders can change the character and distribution of a circuit's timing path delays. While the foremost factor is the movement of addition related timing paths out of the slower soft logic and into the faster hard adders (which typically reduces LUT depth on the critical path), several other factors can also induce changes including: technology mapping (due to limited optimization across the adders), packing (due to differences in logic block structure and flexibility), and placement (due to restrictions keeping long carry chains in a fixed relative position).

[2]Ideally we would examine the Soft implementation of a design to directly determine if its critical path included addition, but as ABC does not preserve node names, we cannot trace LUTs back to specific HDL operations.

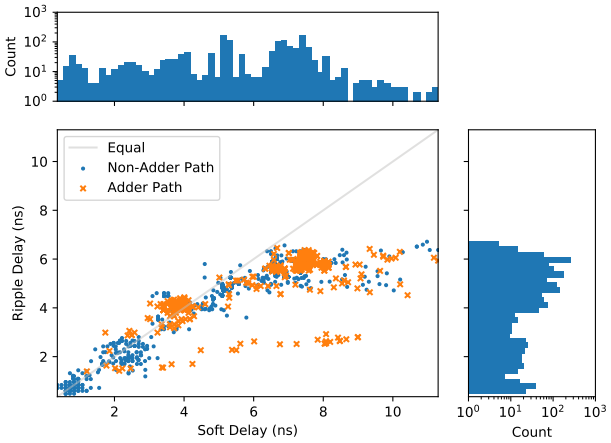


Fig. 25. Endpoint path delay comparison for or1200 benchmark on the *Soft* and *Ripple* architectures

Figure 25 shows how the timing path delay characteristics of the *or1200* design vary between the *Soft* and *Ripple* architectures. Here we see the path delays of *Ripple* have been shifted down compared to the *Soft* architecture. It is interesting to note that many timing paths not involving adders also see improvement. Another factor beyond those listed above is the impact of timing driven optimizations, which refocus optimization effort on these non-adder paths as they are now more timing critical (i.e. no longer dominated by addition related timing paths). It is also worth noting that while the critical timing paths on the *Ripple* architecture still include adders, the path delays are dominated by other non-adder primitives (e.g. LUTs) and routing. Finally, there are a number of other near critical timing paths not involving adders, which indicate the CAD flow has successfully balanced the competing requirements of the various timing paths.

D. Simple vs. High Performance Adder Architectures

An FPGA architect must choose between smaller, more flexible, slower adders vs. larger, less flexible, faster adders. On these complete circuits, the results reaffirm the importance of hard adders but show that different hard adder granularities (1-bit ripple or a more expensive 4-bit CLA) remain reasonable architectural choices. This is an unexpected result, as Table III and Table IV show markedly different area and delay characteristics between 1-bit and 4-bit hard adders, respectively – the ripple adder has 19% more delay for 32-bit addition. One would normally expect that architectures with 1-bit adders would result in smaller circuits that are also slower, yet the area and delay results on complete circuits exhibit this trend only very weakly. Clearly the benefit of a very fast adder for long word-length additions is greatly diluted by the presence of all the logic surrounding adders in complete designs.

TABLE XII
QOR FOR ARCHITECTURES WITH SOFT INTER-CLB LINKS. VALUES ARE THE GEOMETRIC MEAN OF VTR+ CIRCUITS NORMALIZED TO THE EQUIVALENT ARCHITECTURE WITH DEDICATED INTER-CLB LINKS.

Arch	32-bit Add Delay	VTR+ Delay	VTR+ Area	VTR+ Area-Delay Product
Ripple	2.41	1.07	0.99	1.06
U-Ripple	2.31	1.05	0.99	1.04
CLA	2.49	1.04	0.99	1.03
U-CLA	2.12	1.04	0.99	1.03

E. Adder Input Balancing

We now turn to consider how best to integrate the LUT and arithmetic circuitry. The balanced approach of splitting the 6-LUT into two 5-LUTs, where each 5-LUT drives a different adder input has good symmetry. The unbalanced approach of using the 6-LUT to drive one adder input and a small mux to select BLE input pins for the other adder input offers richer LUT functionality feeding the adder input (six pins compared to five for the balanced case) but worse symmetry. It is thus unclear which of these two approaches is better. Note also that commercial FPGAs differ in their approach: Intel's Stratix V [14] and Stratix 10 [15] FPGAs support a balanced style, while Xilinx's Virtex 7 [5] and Ultrascale [13] FPGAs allow both unbalanced and balanced styles.

The second column of Table IX shows the normalized delay values for each of the different architectures on the application circuits. The delays for all architectures are similar, achieving an approximately 15% delay reduction. We therefore conclude that both balanced and unbalanced architectures achieve approximately the same overall delay.

Table X shows the balanced and unbalanced architectures require virtually the same CLB count, indicating that the packer can fill both architectures with roughly the same amount of logic per CLB, despite the fact that the balanced architectures can use a LUT on each input of an adder instead of only one input. Interestingly, the unbalanced architectures require a channel width that is 4% lower, on average. This is due to the fact that the unbalanced architecture can use all 6 inputs of a BLE when in adder mode, while the balanced architectures can use only 5 – the packer has more freedom on what to pack with the adder in the unbalanced architecture and reduces the number of signals to route between clusters. The net impact is that while the unbalanced architectures require slightly more logic area due to their extra 2:1 mux per BLE, they reduce overall area by 1% by reducing the required amount of inter-cluster routing.

F. Utility of Inter-CLB Carry

Dedicated carry links between logic blocks improve the speed of long adders significantly, as shown in Figure 13, but their use constrains the placement engine to keep long adders in a fixed relative placement, which may lengthen the wiring between other blocks. Table XII compares the QoR of architectures with soft inter-CLB carry links (i.e, routed using the general-purpose interconnect) normalized to their corresponding architectures with hard inter-CLB carry links. The first column is the architecture. The second column shows normalized delays for the 32-bit addition micro benchmark. The next three columns show the normalized geometric mean of delay, area, and area-delay product over the VTR+ benchmarks. Using soft inter-CLB links increases the delay of a 32-bit adder by 2.3x on average across the hard adder architectures, but increases the delay of the VTR+ designs by only 5%. The area cost of hard inter-CLB carry is negligible, as little hardware needs to be added to support them, and as their use does not significantly increase the required inter-CLB channel width, despite the constraint they create on the placement engine.

TABLE XIII
FRACTURABLE LUT APPLICATION BENCHMARK RESULTS. VALUES ARE THE GEOMEAN OVER BENCHMARKS, NORMALIZED TO THE FRACTURABLE SOFT ARCHITECTURE.

Arch	Crit Path Delay	Total Used Soft Area	Num CLB	Area Delay Product
frac_ripple	0.85	1.10	1.10	0.94
frac_uripple	0.85	1.13	1.12	0.97
frac_2ripple	0.84	1.02	1.01	0.85
frac_2uripple	0.82	1.04	1.02	0.85

We expect that the impact of hard inter-CLB carry links is a strong function of the number of adder bits per logic block. Fewer bits per block means more inter-CLB links are required for an addition of a given size, which in turn may have a bigger impact on delay. Therefore, we believe that architectures with 8 adder bits per logic block (e.g. Ultrascale [13]) will benefit more from hard inter-CLB links than architectures with 20 bits per block (e.g. Stratix 10 [15]).

G. Bits of Addition per LUT

Table XIII compares the different fracturable LUT carry-chain architectures normalized to the fracturable soft architecture to study the impact of the number of addition bits per LUT. We see that critical path delay is fairly consistent across architectures with a reduction of 15% to 18% vs. the soft fracturable LUT architecture. The area numbers are quite different, with 2-bits of addition per fraturable LUT exhibiting a much lower area overhead (2% to 4% more area than soft), while 1-bit of addition per fracturable LUT uses 10% to 13% more area vs. soft. We therefore conclude that 2-bits of addition per fracturable LUT is a better architecture than 1-bit because it provides both slightly better delay and significantly better area, reducing area delay product by 15% vs. soft.

H. Per Circuit Breakdown for frac_2uripple

Table XIV shows a circuit-by-circuit breakdown of our results on for *frac_2uripple*, our best architecture. By providing absolute values, this table serves as a comparison point for future studies. The columns from left to right are: circuit name, minimum channel width, critical path delay in ns, number of soft logic blocks, and total soft logic area including routing in number of minimum width transistors.

I. Summary

Table XV compares our best hard adder architectures with soft LUT and fracturable LUT architectures. For the soft adder architectures critical path delay is similar, while both hard adder architectures reduce critical path delay by 15-16%. However the fracturable LUTs significantly reduce the number of CLBs required by 12-15%. This reduces area but is slightly tempered by a higher minimum channel width. As a result the combination of fracturable LUTs with 2-bits of hardened addition per LUT improves area-delay product by 25% compared to a soft (non-fracturable) LUT architecture.

TABLE XIV
CIRCUIT-BY-CIRCUIT BREAKDOWN OF THE BEST (FRAC_2URIPPLE) ARCHITECTURE.

Circuit	Min W	Crit Path Delay	Num CLB	Soft Area
bgm	104	22.89	3744	$8.84 \cdot 10^7$
blob_merge	96	8.35	620	$1.42 \cdot 10^7$
boundtop	68	5.51	262	$5.51 \cdot 10^6$
LU8PEEng	122	66.12	2440	$6.08 \cdot 10^7$
LU32PEEng	172	66.72	8225	$2.33 \cdot 10^8$
mcml	142	37.36	8211	$2.14 \cdot 10^8$
mkSMAdapter4B	78	4.89	192	$4.19 \cdot 10^6$
or1200	98	7.49	295	$6.92 \cdot 10^6$
raygentop	90	4.52	198	$4.56 \cdot 10^6$
sha	70	6.64	241	$5.15 \cdot 10^6$
stereovision0	62	3.57	1051	$2.14 \cdot 10^7$
stereovision1	120	5.27	1033	$2.57 \cdot 10^7$
stereovision2	146	11.28	2063	$5.51 \cdot 10^7$

TABLE XV
FRACTURABLE-LUT VS. NON-FRACTURABLE LUT ARCHITECTURE RESULTS. VALUES ARE THE GEOMEAN OVER THE APPLICATION CIRCUITS.

Arch	Crit Path Delay	Total Used Soft Area	Num CLB	Area Delay Product
Soft	1.00	1.00	1.00	1.00
Ripple	0.85	1.04	1.02	0.88
frac_soft	1.02	0.85	0.73	0.86
frac_2uripple	0.84	0.88	0.74	0.75

There is an important caveat that this comparison is not completely fair because the number of inputs to the CLBs are kept constant when in reality the ideal number of inputs to a CLB with non-fracturable LUTs should be lower than that for fracturable even if the underlying logic elements themselves have the same number of inputs. Hence our area results for the non-fracturable LUT architectures could likely be somewhat improved with cluster input count retuning. Overall however, the case for a fracturable LUT architecture with 2-bits of hard arithmetic per LUT is compelling.

VIII. CONCLUSIONS AND FUTURE WORK

This study covered a broad range of different implementations of hard adders, carry chains and fracturable LUTs within an FPGA soft logic block. Our results show that hardening adders and carry chains significantly improves the performance of arithmetic operations (69-79% delay reduction on microbenchmarks), and improves average application circuit delay by 13-16%. While more complex architectures hardening a carry-lookahead adder improve standalone adder speed, we found that simpler hardened ripple-carry adders perform just as well on complete application circuits. We found the additional flexibility of fracturable LUT based architectures enabled them to be more area efficient (12-15% compared to non-fracturable). Fracturable LUTs and hardened adders and carry chains are complementary and the combination can improve both area and delay. We found 2-bits of addition to be particularly well matched to fracturable LUT architectures, yielding an overall area-delay product improvement of 25% compared to a non-fracturable architecture without hardened arithmetic.

Our results show that hardened arithmetic is an important consideration when evaluating soft logic block architectures, and in particular fracturable LUTs. It is therefore important for future work studying these areas to consider the impact of hardened arithmetic. Fracturable LUTs in particular have a large design space, so it would be interesting future work to consider how different fracturable LUT architectures would interact with arithmetic. It would also be interesting to consider the impact of re-tuning the number of inputs to the logic block for these architectures. Another direction for future work is to focus on improving logic synthesis when hardened adders are used; ideally ABC would be upgraded to understand the logic within hard adders and optimize across their boundaries. Finally, we expect that using hardened adders will reduce power consumption for arithmetic operations, however it would be useful to quantify this impact, and determine whether any of the proposed hard adder architectures would be better suited to low power FPGAs.

ACKNOWLEDGMENT

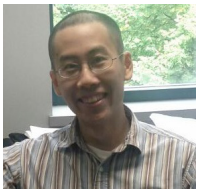
We gratefully acknowledge the funding support of NSERC, Intel, the New Brunswick Innovation Foundation, CMC Microsystems, the Semiconductor Research Corporation, and Texas Instruments.

REFERENCES

[1] J. Rose, "Hard vs. Soft: The Central Question of Pre-Fabricated Silicon," *IEEE ISMVL*, pp. 2–5, 2004.
[2] D. Lewis *et al.*, "Architectural Enhancements in Stratix V," in *ACM FPGA*, 2013, pp. 147–156.
[3] J. Greene *et al.*, "A 65nm Flash-Based FPGA Fabric Optimized for Low Cost and Power," in *ACM FPGA*, 2011, pp. 87–96.
[4] Lattice Semiconductor, "LatticeECP3 Family Handbook," http://d12lxohwf1zsq3.cloudfront.net/documents/HB1009.pdf, 2013.
[5] Xilinx Inc., "7 Series FPGAs Configurable Logic Block User Guide," http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf, 2013.
[6] H.-C. Hsieh *et al.*, "Third-Generation Architecture Boosts Speed and Density of Field-Programmable Gate Arrays," in *IEEE CICC*, 1990, pp. 31–2.
[7] N.-S. Woo, "Revisiting the Cascade Circuit in Logic Cells of Lookup Table Based FPGAs," in *ACM FPGA*, 1995, pp. 90–96.
[8] S. Xing and W. W. Yu, "FPGA Adders: Performance Evaluation and Optimal Design," *IEEE Design & Test of Computers*, vol. 15, no. 1, pp. 24–29, 1998.
[9] S. Hauck, M. Hosler, and T. Fry, "High-Performance Carry Chains for FPGA's," *IEEE TVLSI*, vol. 8, no. 2, pp. 138–147, 2000.
[10] H. Parandeh-Afshar, A. Neogy *et al.*, "Compressor tree synthesis on commercial high-performance fpgas," *ACM TRETS*, vol. 4, no. 4, pp. 39:1–39:19, Dec. 2011.
[11] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "A Novel FPGA Logic Block for Improved Arithmetic Performance," in *ACM FPGA*, 2008, pp. 171–180.
[12] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Efficient Synthesis of Compressor Trees on FPGAs," in *IEEE ASP-DAC*, 2008, pp. 138–143.
[13] Xilinx Inc., "UltraScale Architecture Configurable Logic Block," https://www.xilinx.com/support/documentation/user_guides/ug574-ultrascale-clb.pdf, 2017.
[14] Altera Corp., "Logic Array Blocks and Adaptive Logic Modules in Stratix V Devices," http://www.altera.com/literature/hb/stratix-v/stx5_51002.pdf, 2013.
[15] Intel Corp., "Intel Stratix 10 Logic Array Blocks and Adaptive Logic Modules User Guide," https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/ug-s10-lab.pdf, 2018.
[16] J. Luu, C. McCullough *et al.*, "On hard adders and carry chains in fpgas," in *IEEE FCCM*, 2014, pp. 52–59.
[17] C. Chiasson and V. Betz, "COFFE: Fully-Automated Transistor Sizing for FPGAs," in *IEEE FPT*, 2013, pp. 34–41.
[18] I. Kuon and J. Rose, "Area and Delay Trade-Offs in the Circuit and Architecture Design of FPGAs," in *ACM FPGA*, 2008, pp. 149–158.
[19] D. Lewis, E. Ahmed *et al.*, "The Stratix II logic and routing architecture," in *ACM FPGA*, 2005, pp. 14–20.
[20] G. Zgheib and P. Ienne, "Evaluating fpga clusters under wide ranges of design parameters," in *FPL*, Sep. 2017, pp. 1–8.
[21] C. Chiasson and V. Betz, "Should fpgas abandon the pass-gate?" in *Int. Conf. on Field programmable Logic and Applications*, 2013, pp. 1–8.

[22] J. Luu, J. Goeders *et al.*, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 2, pp. 6:1–6:30, Jul. 2014.
[23] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification," http://www.eecs.berkeley.edu/~alanmi/abc, 2009.
[24] S. Jang *et al.*, "SmartOpt: An Industrial Strength Framework for Logic Synthesis," in *ACM FPGA*, 2009, pp. 237–240.
[25] J. Luu, J. Rose, and J. Anderson, "Towards Interconnect-Adaptive Packing for FPGAs," in *ACM FPGA*, 2014.
[26] K. E. Murray, S. Whitty *et al.*, "Titan: Enabling large and complex benchmarks in academic cad," in *Int. Conf. on Field Programmable Logic and Applications*, 2013, pp. 1–8.

Kevin E. Murray (S'12) is a PhD candidate at the University of Toronto, where he received his BASc and MASc in 2012 and 2015. He has previously been a visiting Research Assistant at Imperial College London, and worked on digital design flows at Advanced Micro Devices (AMD). His research interests include FPGA CAD and Architecture, timing analysis, and modular design flows. He has contributed extensively to the VTR project since 2012.



Jason Luu is a senior software engineer at Altera. He received the BASc degree in CE from the University of Waterloo in 2007 and the MASc and PhD degrees at the University of Toronto in 2010 and 2014 respectively. He has contributed 8 years to the open source VTR project for FPGA CAD and architecture research.

Matthew J. P. Walker (S'16) received his BASc in Computer Engineering from the University of Toronto in 2017, and is presently a MASc candidate in Computer Engineering at the same university, where he is researching CGRA mapping. He has previously worked at Altera and Intel PSG, and worked as a summer researcher in 2014 on the VTR project.

Conor McCullough (S'14) received the B.Eng. degree in computer engineering from the University of New Brunswick, Fredericton, NB, Canada, in 2014. He worked as a summer researcher in 2014 on the VTR project under the supervision of Kenneth Kent.

Sen Wang (S'14) received the B.SC. degree in computer science from Nanjing University of Science & Technology, Nanjing, Jiangsu, China, in 2009, and the M.Sc. degree in computer science, Fredericton, New Brunswick, Canada, in 2014.

Safeen Huda (S'13) received the B.A.Sc. and M.A.Sc. degrees in electrical engineering from the University of Toronto, Toronto, ON, Canada in 2009 and 2012, respectively and the Ph.D. degree in computer engineering in 2017 His research interests include all the aspects of digital circuit and system design. Mr. Huda has held the Natural Sciences and Engineering Research Council of Canada Post-Graduate Scholarship and the University of Toronto Fellowship.

Bo Yan (S'14) received the B.Eng. degree in electronic and information engineering from Hubei University of Technology, Wuhan, Hubei, China, in 2010, and the M.Sc. degree in computer science, Fredericton, New Brunswick, Canada, in 2015.

Charles Chiasson (S'08–M'15) received the B.Eng. degree in electrical engineering from Universit de Moncton in 2011 and the M.A.Sc. degree in electrical and computer engineering from the University of Toronto in 2013. He is currently with the Altera Toronto Technology Center, Altera Corporation, Toronto, ON, Canada.

Kenneth B. Kent (S'96–M'02–SM'13) received his BSc degree from Memorial University of Newfoundland, Canada, and MSc and PhD degrees from the University of Victoria, Canada. He is a professor in the Faculty of Computer Science at the University of New Brunswick, and Director of the IBM Centre for Advanced Studies-Atlantic, Canada. His research interests are Hardware/Software Co- Design, Virtual Machines, Reconfigurable Computing, and Embedded Systems. His research groups are key contributors to widely used software such as the IBM J9 Java virtual machine and the VTR (Verilog-To-Routing) FPGA CAD flow.

Jason Anderson (S'96–M'05) received the B.Sc. degree in computer engineering from the University of Manitoba, Winnipeg, MB, Canada, and the Ph.D. and M.A.Sc. degrees in electrical and computer engineering (ECE) from the University of Toronto (U of T), Toronto, ON, Canada. In 1997, he joined the FPGA Implementation Tools Group, Xilinx, Inc., San Jose, CA, working on placement, routing, and synthesis. He is currently an Associate Professor with the ECE Department at U of T and holds the Jeffrey Skoll Chair in Software Engineering. He has authored over 60 papers published in refereed conference proceedings and journals, and is an inventor on 26 issued U.S. patents. His current research interests include computer-aided design, architecture and circuits for FPGAs.

Jonathan Rose (F'09) is a Professor with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada. He has worked in the area of FPGA CAD and architecture for over 20 years, including stints at the two major vendors, Xilinx and Altera, as well as a startup. Prof. Rose is a Fellow of the ACM and a Foreign Member of the American National Academy of Engineering.

Vaughn Betz (S'88–M'91–SM'17) Dr. Betz co-founded Right Track CAD to develop new FPGA architectures and CAD tools and was its VP of Engineering until its acquisition by Altera in 2000. He was at Altera from 2000 to 2011, ultimately as Senior Director of Software Engineering, and is one of the architects of both the Quartus II CAD system and the Stratix I - V and Cyclone I - V FPGAs. He is a Professor and the NSERC/Intel Industrial Research Chair in Programmable Silicon at the University of Toronto, where his research covers FPGA architecture, CAD, and acceleration of computation using FPGAs.