On Hard Adders and Carry Chains in FPGAs

Jason Luu*, 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
† Dept. of Electrical and Computer Engineering, 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. There has been very little study, however, on these choices and hence we explore a number of possibilities for hard adder design. We also highlight optimizations during front-end elaboration that help ameliorate the restrictions placed on logic synthesis by hardened arithmetic. We show that hard adders and carry chains, when used for simple adders, increase performance by a factor of four or more, but on larger benchmark designs that contain arithmetic, improve overall performance by roughly 15%. We measure an average area increase of 5% for architectures with carry chains but believe that better logic synthesis should reduce this penalty. Interestingly, we show that adding dedicated inter-logic-block carry links or fast carry look-ahead hardened adders result in only minor delay improvements for complete designs.**

I. Introduction

One of the central questions in FPGA architecture is the determination of which functions to harden and which to leave for implementation in the soft logic [1]. A function should be hardened if it appears often in the set of used applications, and if there is a large advantage when it is implemented in hard logic rather than soft. This argument has held sway in the case of adder-type arithmetic functions – they appear often and hardened adders are much faster than soft adders. Consequently, commercial devices commonly have hardened adder and/or carry logic and routing [2] [3] [4] [5]. Indeed, hardened arithmetic structures have been a longstanding feature of commercial FPGAs, yet there has been no comprehensive published study of the performance benefits they offer on complete designs or their cost in terms of area; this paper aims to fill that gap.

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. There has been little published work that sheds light on the set of such choices, nor the impact they have on the resulting implementations of complete designs in FPGAs. We study a number of these choices and determine their impact on performance, area and CAD complexity. Some examples include: First, the determination of how an adder interacts with nearby LUTs and flip-flops. Second, the trade-off of performance and area between larger, faster, multi-bit adders and more flexible, slower, smaller single bit adders. Third, the design of the connection between the carry bits of adjacent hard adder units; for example, should there be dedicated links for the carry signal across soft logic block boundaries so that wide additions may be done at high speed but with a more constrained placement problem? Or should those connections cross soft logic boundaries using the general-purpose interconnect of the FPGA? These are important implementation details that an architect must decide on when embedding hard adders in with soft logic. We present quantitative measurements of the impact of these decisions.

Previous work in this area began in the early 90'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 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 [10] proposed adding hardened compressors to soft logic blocks to speed up multi-input addition with a focus on DSP and video applications. The benchmarks used in this study appear to be on the order of a few hundred 6-LUTs [11].

FPGA vendors currently choose different hard arithmetic architectures inside their soft logic blocks. The Xilinx Virtex-7 FPGA family [5] contains a basic ripple carry adder architecture where addition can only start on every 4^{th} adder bit. 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 with shared inputs. The Altera 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. Each adder input is driven by a 4-LUT with input sharing constraints [12]. 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

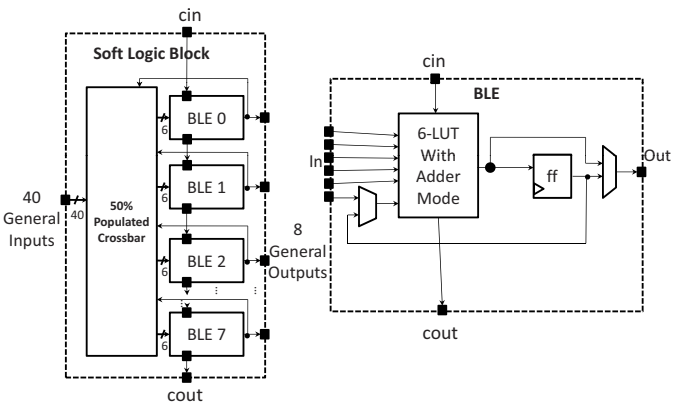

Fig. 1. The base soft logic block.

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.

This paper is organized as follows: Section II describes the FPGA architecture and circuit design that serve as the basis for the exploration. Section III describes the variations of the hard arithmetic structures and their interaction with the soft logic. Section IV describes CAD flow issues that must be dealt with to handle the regular arithmetic structure, and several important optimizations. Section V presents measurements of the various architectures on pure-adder *microbenchmarks*. Section VI describes results for full application circuits, and Section VII concludes and suggests future work.

II. Base Architecture Model

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. Fig. 1 illustrates the base soft logic block used in this study, which contains eight Basic Logic Elements (BLEs, to be described later), 40 general inputs, eight general outputs, one cin pin and one cout pin. The BLE consists of a 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 one architecture that does not contain hardened arithmetic, and hence has neither cin nor cout pins.

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 in [13]; 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.

For the logic block of Fig. 1 but without hard carry links, which BLE performs which function can be changed by the routing stage of the CAD flow to allow different functions to access different output pins – the outputs are *logically equivalent*. When the carry links of the BLEs are used however, the order of those BLEs are fixed and cannot be exchanged, so the outputs of BLEs using their carry function are not logically equivalent. The CAD tool that we are using, 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 at all, thus removing all output logical equivalence. To compensate for this restriction, each ouptut pin can directly access two sides of the logic block, and hence both a vertical and a horizontal channel. 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, but we believe the impact is small.

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

Table I gives the routing architecture parameters of the base architecture, which are chosen to be in line with the recommendations of prior research [14]. The hard memory logic block can implement memories of different aspect ratios ranging from 32Kx1 down to 1Kx32 for both dual-port and single-port modes. The multiplier logic block can implement a 36x36 multiplier that can optionally fracture to two 18x18 multipliers. Each 18x18 multiplier can further fracture down to two 9x9 multipliers.

Area and Delay Models

The transistor-level design of the base soft logic blocks and routing architecture are done using the COFFE tool [13] and a 22nm CMOS technology. The architecture uses pass gates; statically controlled pass gates are gate-boosted by 0.2V. 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 Design

The goal of this paper is to explore various hard adder and carry chain architectures, and to do so in the context of careful electrical design of the key circuits. The two hard adder primitives in this study are hand-optimized 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 II 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 of [13]. 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 III shows the properties of the 4-bit carry lookahead adder used in this study. The

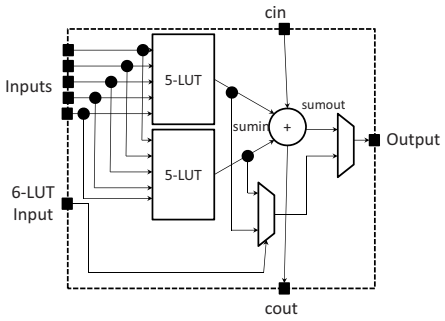

Fig. 2. A *balanced* 6-LUT and adder interaction where both adder inputs are driven by 5-LUTs.

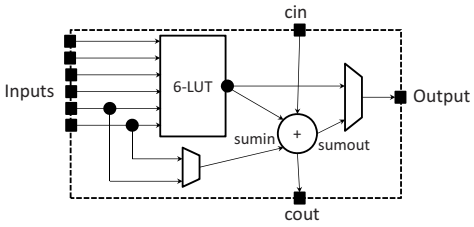

Fig. 3. An *unbalanced* 6-LUT and adder interaction where the 6-LUT drives only one adder input.

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. 2 shows one of the ways that we explore interaction between the adder and LUT. 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 is the most appropriate architecture. Example circuits that may benefit from this architecture would be applications where multiplexers select the inputs to an adder.

Fig. 3 shows another LUT-adder interaction architecture that we will explore. Here, the 6-LUT output drives one of the

TABLE II. PROPERTIES OF THE 1-BIT HARD ADDER USED IN THIS STUDY.

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 III. PROPERTIES OF THE 4-BIT CARRY LOOKAHEAD ADDER USED IN THIS STUDY.

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

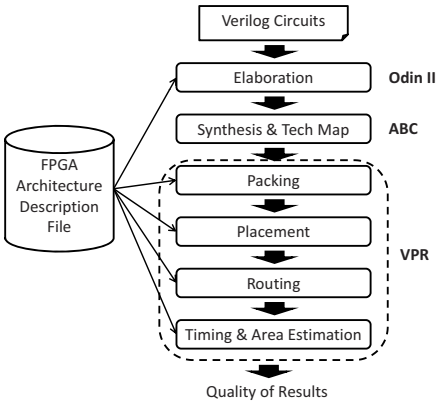

Fig. 4. The VTR CAD flow

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 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.

A third type of architecture we are interested in 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. We create two physically equivalent pins at the right and bottom sides of the logic block for both carry-in and carry-out (i.e. 4 pins in total). For architectures that have a dedicated carry link, the carry link has a delay of 20 ps.

Finally, 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 LSB that generates a 0 or a 1 cin for addition and subtraction, respectively. We employ this approach in our research.

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 employ and modify the open-source VTR 7.0 [15] CAD flow, which is illustrated in Fig. 4. 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 [16] 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. Several of these steps had to be changed or augmented to enable hardened adders and carry chains, as described below.

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 [17]; 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.

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 VI 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 IV shows the impact of these optimizations on the benchmark circuits described in Section VI. Note that certain circuits are more heavily impacted by duplicate hard block reduction, and others by unused logic removal, so it is clear that both are necessary for efficient optimization. Enabling both duplicate hard block reduction and unused logic removal reduces logic blocks used by 12% on average.

B. Threshold of When to Use Hard Adders

While hardened adder and carry logic is clearly good to use 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.

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

Circuit	DHR CLB	ULR CLB	All CLB	All 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

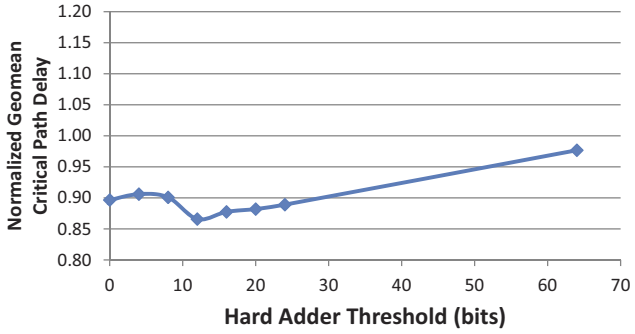

Fig. 5. Circuit speed vs. hard adder threshold. Results are the average across 14 benchmarks and normalized to the soft implementation.

Fig. 5 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 IV. There is a general trend towards achieving a minimum mean delay at a threshold of around 12 bits.

Fig. 6 shows the area impact of different hard adder thresholds. The x-axis is again the hard adder threshold, while the y-axis shows 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

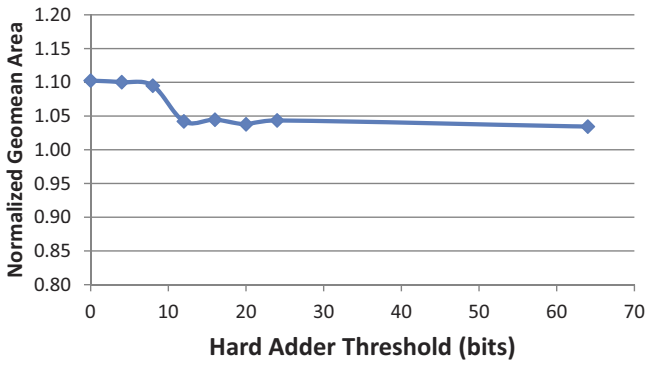

Fig. 6. Average total area of different hard adder thresholds normalized to the soft architecture.

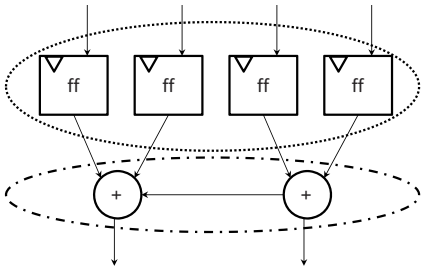

Fig. 7. Example of transitive connections.

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.

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. 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.

The packer inside VPR 7.0 is an interconnect-aware packing algorithm [18] 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. Fig. 7 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. The packer was modified 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 scores them higher than other unconnected logic. With this modification, circuits such as that illustrated in Fig. 7 were packed correctly.

D. Placement and Routing

VPR 7.0 has place-and-route functionality for carry chain exploration. The architecture description file specifies the dedicated carry chain links between soft logic blocks. VPR 7.0 automatically generates an FPGA architecture with those links and places logic blocks that use those links in the right order. However, in our initial experiments, we found that the routing process was simply too slow, particularly in determining the minimum channel width of the biggest VTR benchmarks. The reason was that the router didn't always detect impossible-to-route situations, and spent too long trying to route them. We modified the routing stage of VPR to perform the minimum channel width search faster, by adding a heuristic that used linear extrapolation of the overused routing resource node count on a window of routing iterations to predict the iteration at which routing may succeed. If the predicted final iteration is above the maximum number of routing iterations plus a threshold, then routing is likely impossible so VPR exits early. This heuristic sped up the minimum channel width search by 70% without any loss in quality of result.

TABLE V. ARCHITECTURE ACRONYMS.

Acronym	Architecture
Soft	Soft logic only
Ripple	1-bit ripple carry, balanced LUT
U-Ripple	1-bit ripple carry, unbalanced LUT
CLA	4-bit CLA, balanced LUT
U-CLA	4-bit CLA, unbalanced LUT

V. MICROBENCHMARK RESULTS

Table V lists the 5 different ways of supporting arithmetic in an FPGA architecture that we investigate. Before exploring the effect of each architecture on full designs, it is instructive to measure their effect on various sizes of simple adders – 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 that critical path delay measurement is a direct function of the adder combinational logic delay. These registered adders are implemented using the flow described in Section IV.

Fig. 8 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 across soft logic blocks, rather than using dedicated carry links. 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. The *no CLB carry* circuits 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 of 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) noise. For hard adders, the lack of CAD flexibility forces a predictable physical design thus greatly reducing CAD noise for these microbenchmarks.

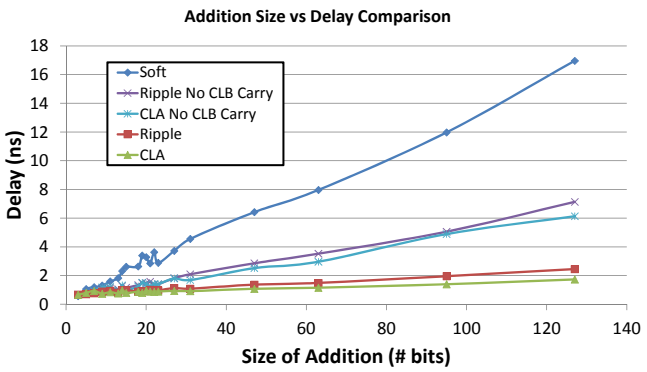

Fig. 8. Delay vs adder length for various architectures.

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.

VI. Application Circuit Results

The goal of this work is to study the impact of hard adder architectural decisions on the performance and area of complete designs implemented in FPGAs. This includes the study of adder granularity (one-bit ripple vs. four-bit CLA), the symmetry of the LUT structure feeding the adder, and the utility of high-speed inter-CLB carry links. The complete design benchmarks we use are from the VTR 7.0 release, specifically, all circuits larger than 1000 6-LUTs[1]. We will refer to these as the VTR+ benchmarks. The geometric average *atom* count across all 14 circuits is 11,700.

Table VI 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 number of adder bits after elaboration, 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. On average, the ratio of adder bits divided by the number of 6-LUTs is 0.21, 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). For $blob\_merge$, the longest chain has just 13 bits. The geometric mean of the longest addition/subtraction lengths is 31.2 bits. The most adder-intensive circuit is $stereovision2$ with 1.26 adders per LUT, while the least adder-heavy circuit is arm_core at 0.04. These measurements correspond well with other modern benchmarks. For the TITAN benchmarks (with the SPARC cores excluded because these cores have almost no adders at all) [19], the geometric average of the fraction of LUTs in arithmetic mode and the maximum of length of addition/subtraction is 0.22 and 35.8, respectively.

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

TABLE VI. Benchmark Statistics when mapped to Ripple architecture.

Circuit	Num 6LUTs	Num Add Bits	Max Add Len	Avg Add Len	Add/LUT Ratio
arm_core	13812	537	35	9.16	0.04
bgm	32337	5438	25	9.34	0.17
blob_merge	7843	3754	13	11.96	0.48
boundtop	2846	309	19	7.22	0.11
LU8PEEng	21668	3251	47	11.04	0.15
LU32PEEng	73828	8249	47	11.9	0.11
mcml	94048	24302	65	47.52	0.26
mkSMAdapter4B	1819	431	33	6.87	0.24
or1200	2813	534	65	23.85	0.19
raygentop	1778	580	32	11.8	0.33
sha	1994	309	33	23.95	0.15
stereovision0	8282	2920	18	11.17	0.35
stereovision1	7845	2388	19	6.38	0.30
stereovision2	11006	13843	32	23.89	1.26
geomean	8606	1807	31.2	12.5	0.21

TABLE VII. Delay for different hard adder architectures, normalized to the soft logic architecture.

Arch	32-bit Add Delay	Application Circuits Delay
Ripple	0.239	0.866
U-Ripple	0.231	0.878
CLA	0.201	0.871
U-CLA	0.195	0.850

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 yielded the best area-delay results in subsection IV-B.

Each of the circuits was mapped to one of the four architectures described in Table V, which correspond to the two granularities and the balanced and unbalanced architectures described in Section III. In addition, each of these architectures was modified to remove the hard inter-CLB carry links, creating four more architectures, for a total of eight.

A. Microbenchmarks vs. Application Circuits

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 VII shows the geometric average critical path delay for each of the architectures normalized to the soft logic architecture. An isolated 32-bit adder sees a compelling delay reduction of 76% to 80% 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. This is a common outcome in the hardening of any kind of circuit – the final impact on critical path delay 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,

TABLE VIII. 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	Area-Delay Product	Min W	Num CLB
Ripple	1.042	0.902	0.960	1.029
U-Ripple	1.038	0.911	0.927	1.032
CLA	1.060	0.923	0.961	1.037
U-CLA	1.044	0.888	0.918	1.035

however, that the other hardened adder architectures benefit circuit speed almost as much.

B. Simple vs. High Performance Adder Logic

An FPGA architect much choose between smaller, more flexible, slower adders vs. larger, less flexible, faster adders. The second column of Table VII shows that, on average, the two ripple architectures have 19% more delay than the two carry-lookahead architectures for a 32-bit addition. For the application circuits, however, the ripple architectures average only 1.3% more delay than the CLA architectures. 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 VIII shows the quality of results (QoR) for each of the architectures 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, area-delay product, minimum channel width, and number of used soft logic blocks. The CLA architecture increases area slightly (by between 1% and 2%) but cuts delay by roughly the same amount, leading to an area-delay product that is very close to that of the ripple architectures.

On these complete circuits, the results reaffirm the importance of hard adders but show that different hard adder granularities (1-bit ripple or 4-bit CLA) remain reasonable architectural choices. This is an unexpected result, as Table II and Table III show markedly different area and delay characteristics between 1-bit and 4-bit hard adders, respectively. 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.

C. Balanced vs. Unbalanced

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: Altera's Stratix V FPGAs [12] support a balanced style, while Xilinx's Virtex7 FPGAs [5] allow both unbalanced and balanced styles.

The third column of Table VII shows the normalized delay values for each of the different architectures. The delay of the U-Ripple architecture is approximately the same as that of the Ripple architecture. The delay of the U-CLA architecture is 2.5% faster than the CLA architecture. From these results, we conclude that balanced and unbalanced architectures achieve approximately the same overall delay.

TABLE IX. 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	1.92	1.03	1.003	1.03
U-Ripple	1.77	1.01	1.003	1.02
CLA	1.60	0.99	1.003	0.99
U-CLA	1.85	1.02	1.000	1.02

Table VIII shows the QoR for each of the architectures normalized to the soft logic architecture. 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.

D. Utility of Inter-CLB Carry

Dedicated carry links between logic blocks improve the speed of long adders significantly, as shown in Fig. 8, but their use constrains the placement engine to keep long adders in a fixed relative placement, which may de-optimize the wiring between other blocks. Table IX 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 78% on average across the hard adder architectures, but increases the delay of the VTR+ designs by only 1.3%. 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.

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 adder 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 4 adder bits per logic block (e.g. Virtex 7 [5]) will benefit more from hard inter-CLB links than architectures with 20 bits per block (e.g. Stratix V [12]).

TABLE X. 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
arm_core	0.959	1.083	0.720	2
LU8PEEng	0.837	1.047	0.646	2
mcml	0.520	1.101	0.238	20
or1200	0.698	1.052	0.154	5
sha	0.560	1.026	0.250	6
stereovision2	0.891	0.782	0.050	3
LU32PEEng	0.730	0.955	0.650	0
bgm	1.099	1.118	0.727	0
boundtop	0.998	1.032	0.778	0
blob_merge	0.954	1.081	1.000	0
mkSMAdapter4B	0.997	1.106	0.857	0
raygentop	0.972	1.100	N/A	0
stereovision0	0.936	1.051	1.200	0
stereovision1	1.024	1.138	N/A	0
geomean	0.850	1.044	0.456	–
stdev	0.177	1.041	0.358	–

E. Circuit-by-Circuit Breakdown

Table X 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, area-delay product, and the number of LUTs on the critical path. The last column is the number of (4-bit) 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 6 circuits hard adders are on the critical path, and we obtain a large delay reduction of 27% (a 38% speed-up). The next 3 circuits (bgm, boundtop and LU32PEEng) have reductions in the critical path LUT depth of more than 20% 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 3 circuits have an average LUT depth that is 28% lower when targeting U-CLA vs. Soft, only LU32PEEng speeds up significantly, and the average delay reduction across the 3 designs is only 7%. We believe this illustrates a key 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 five 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 very little variation in speed vs. carry architecture, as one would expect.

VII. CONCLUSIONS AND FUTURE WORK

This study covered a broad range of different implementations of hard adders and carry chains within a soft logic block. We show that different hard adder and carry chain architectures show very similar area and delay values on real applications despite significant differences on microbenchmarks. We conclude that hardened adders provide a speed up of approximately 15% for an area penalty of approximately 5% resulting in an overall area-delay product reduction of approximately 10%.

There is much future work in both CAD and architecture to explore. The interaction between fracturable LUTs and hard adders is interesting as it adds another dimension to the architecture space. In terms of CAD, the most pressing issue is the lack of good logic synthesis when adders are used; ideally ABC would be upgraded to understand the logic within hard adders.

[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.

VIII. ACKNOWLEDGEMENTS

We gratefully acknowledge the funding support of NSERC, Altera, the Semiconductor Research Corporation and Texas Instruments. We also thank Kevin Murray for providing data on carry usage in the Titan benchmarks.

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, P. Brisk, and P. Ienne, "A Novel FPGA Logic Block for Improved Arithmetic Performance," in *ACM FPGA*, 2008, pp. 171–180.

[11] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Efficient Synthesis of Compressor Trees on FPGAs," in *IEEE ASP-DAC*, 2008, pp. 138–143.

[12] Altera Co., "Logic Array Blocks and Adaptive Logic Modules in Stratix V Devices," http://www.altera.com/literature/hb/stratix-v/stx5_51002.pdf, 2013.

[13] C. Chiasson and V. Betz, "COFFE: Fully-Automated Transistor Sizing for FPGAs," in *IEEE FPT*, 2013, pp. 34–41.

[14] 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.

[15] J. Rose *et al.*, "The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing," in *ACM FPGA*, 2012, pp. 77–86.

[16] A. Mishchenko *et al.*, "ABC: A System for Sequential Synthesis and Verification," http://www.eecs.berkeley.edu/alanmi/abc, 2009.

[17] S. Jang *et al.*, "SmartOpt: An Industrial Strength Framework for Logic Synthesis," in *ACM FPGA*, 2009, pp. 237–240.

[18] J. Luu, J. Rose, and J. Anderson, "Towards Interconnect-Adaptive Packing for FPGAs," in *ACM FPGA*, 2014.

[19] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "TITAN: Enabling Large and Complex Benchmarks in Academic CAD," in *IEEE FPL*, 2013.