Math Doesn't Have to be Hard: Logic Block Architectures to Enhance Low-Precision Multiply-Accumulate on FPGAs

Andrew Boutros[1,2,*], Mohamed Eldafrawy[1,*], Sadegh Yazdanshenas[1] and Vaughn Betz[1,2]
[1]Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada
[2]Vector Institute, Toronto, ON, Canada
{firstname.lastname}@mail.utoronto.ca,vaughn@eecg.utoronto.ca

ABSTRACT

Recent work has shown that using low-precision arithmetic in Deep Neural Network (DNN) inference acceleration can yield large efficiency gains with little or no accuracy degradation compared to half or single precision floating-point by enabling more MAC operations per unit area. The most efficient precision is a complex function of the DNN application, structure and required accuracy, which makes the variable precision capabilities of FPGAs very valuable. We propose three logic block architecture enhancements to increase the density and reduce the delay of multiply-accumulate (MAC) operations implemented in the soft fabric. Adding another level of carry chain to the ALM (extra carry chain architecture) leads to a 1.5× increase in MAC density, while ensuring a small impact on general designs as it adds only 2.6% FPGA tile area and a representative critical path delay increase of 0.8%. On the other hand, our highest impact option, which combines our 4-bit Adder architecture with a 9-bit Shadow Multiplier, increases MAC density by 6.1×, at the cost of larger tile area and representative critical path delay overheads of 16.7% and 9.8%, respectively.

KEYWORDS

Logic block architecture; Soft multipliers; Low-Precision; Deep learning;

ACM Reference Format:
Andrew Boutros, Mohamed Eldafrawy, Sadegh Yazdanshenas and Vaughn Betz. 2019. Math Doesn't Have to be Hard: Logic Block Architectures to Enhance Low-Precision Multiply-Accumulate on FPGAs. In *The 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '19), February 24–26, 2019, Seaside, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3289602.3293912

1 INTRODUCTION

Since the demonstration of their huge potential in the 2012 ImageNet large-scale visual recognition challenge [13], deep neural networks (DNNs) have resulted in numerous breakthroughs in the machine intelligence field. They have rapidly replaced classical machine learning approaches that are based on hand-crafted domain-specific features in various areas such as computer vision and natural language processing. However, the state-of-the-art accuracy achieved by DNNs comes at the cost of increased computational complexity as DNN models become bigger and deeper to achieve higher accuracy. As a result, high-performance and energy-efficient hardware acceleration is necessary to deploy DNN models both in mobile devices and large-scale datacenter services that have tight power budget and latency constraints. From this perspective, FPGAs offer an attractive solution for accelerating DNNs due to their higher energy efficiency and lower latency compared to GPUs, as well as their flexibility and lower NRE cost compared to ASICs.

Two of the main classes of DNNs are convolutional neural networks (CNNs) and long short-term memories (LSTMs) which are considered the state-of-the-art models for visual recognition and natural language processing, respectively. CNNs are typically composed of several convolution layers in which an input feature tensor is convolved with a set of kernels to produce the output feature tensor consumed by the subsequent layer [13]. Recent CNN models consist of up to a thousand convolutional layers, followed by at least one fully-connected layer [22]. On the other hand, LSTMs consist of a series of matrix-vector multiplications followed by non-linear activation functions to compute the values of input, output and forget *gates* over several time steps [10]. Thus, the fundamental operation in both CNNs and LSTMs is multiply-accumulate (MAC) of weights and input features. With the introduction of high-bandwidth memory [24] and persistent DNNs that can store all model parameters in on-chip memory [9], off-chip communication has become less problematic and the primary bottleneck is the number of MACs that can be performed on-chip per cycle.

Recently, research efforts have shown that huge gains can be achieved by using low-precision arithmetic in DNN inference acceleration with little or no accuracy degradation compared to half or single precision floating-point [19]. The flexibility of FPGAs in implementing custom bitwidth datapaths gives them an additional advantage compared to GPUs in accelerating low-precision DNNs, particularly as there is no one precision that is always optimal. Mishra et al showed that a wide range of fixed-point precisions occur on the pareto optimal curve of CNN inference accuracy vs. hardware cost, with precisions from 2- to 8-bits all having a role to play [19]. Rybalkin et al showed similar promise for low-precision fixed point LSTM inference, with pareto-optimal points for high accuracy networks ranging from 3 to 8 bits of precision [21]. Microsoft's Project BrainWave achieves very high performance on LSTMs and gated recurrent units (GRUs) using non-standard block floating-point representations with mantissa precisions as low as

* Both authors contributed equally to this work.

FPGA '19, February 24–26, 2019, Seaside, CA, USA

ACM ISBN 978-1-4503-6137-8/19/02…$15.00
https://doi.org/10.1145/3289602.3293912

2 to 5 bits (implying 3 to 6 bit multiplies) mapped to soft logic multipliers and adders [9].

The desire for even higher performance low-precision multiply-accumulate on FPGAs motivated recent work to enhance FPGA DSP blocks to natively support low-precision 9-bit and 4-bit multiplication and MAC operations at minimal block area overhead [4]. However, DSP blocks represent only 5% of the FPGA core area in DSP-rich devices [15] which significantly dilutes the overall gains of these enhancements. Logic blocks are the most abundant resource in an FPGA, typically constituting about two thirds of its core area [6], and thus efficiently exploiting them can have more impact on the overall performance of an application than DSP block enhancements. In this work, we investigate architectural changes to the FPGA logic blocks that can significantly increase the density of on-chip low-precision MAC operations at minimal area and delay cost. Our contributions in this paper are the following:

- We propose three different architectural enhancements to the logic fabric of current commercial FPGAs to improve the density of on-chip MAC operations.
- We extend the COFFE automatic transistor sizing tool [25] to support more sophisticated logic block architectures similar to those in Intel Stratix-10 FPGAs, as well as our proposed enhancements.
- We evaluate the density and speed gains of these enhancements when implementing 4- to 9-bit multiply and MAC operations.
- We quantify the impact of our proposed changes on logic block area and on the delay of key paths that impact the speed of the FPGA logic for general use.

2 BACKGROUND

2.1 The Evolution of the FPGA Soft Logic

While various basic building blocks have been used for FPGA logic [20], current commercial FPGAs all use various forms of look-up tables (LUTs) as the basis of their logic elements [23]. A K-LUT can implement any K-input logic function by storing its truth table in SRAM cells and using the K inputs as multiplexer selection lines to choose between the stored values. Classical FPGA architecture exploration studies showed that LUTs with 4 to 6 inputs provide the best area-delay product for a wide range of benchmark circuits [2], with 6 LUTs being faster but less area-efficient than 4-LUTs.

The Stratix II architecture [16] introduced *fracturable LUTs*; these LUTs add a small amount of circuitry so that they can not only implement a single K-input function, but can alternately also implement two $K-1$-input functions by dividing the K-LUTs' truth table and output selection multiplexer. Fracturable LUTs can implement circuit critical paths with large LUTs to reduce the number of LUTs in series and help speed, while still packing two smaller logic functions into a single fracturable LUT when it is more area-efficient.

The latest FPGAs from both Xilinx and Intel use fracturable 6-LUT logic fabrics, but make different design choices on how flexible the fracturing is. The Stratix-10 Adapative Logic Module (ALM) [17] is a 6-LUT that can be fractured to implement any two 5-input functions that use no more than 8 distinct inputs, while the Virtex Ultrascale+ 6-LUT can be fractured to implement two 5-input functions with no more than 5 distinct inputs [7]. Both of these architectures also have dedicated structures to implement arithmetic (sum and carry) functions; in Stratix-10 a fracturable 6-LUT is paired with two bits of hard arithmetic, while Xilinx Ultrascale+ pairs each 6-LUT with one bit of arithmetic.

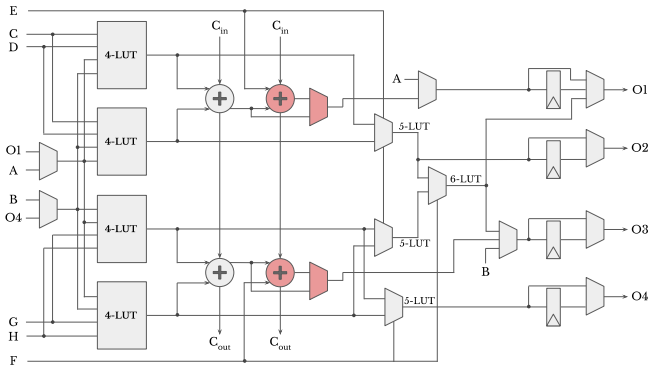

Figure 1: ALM architecture of Stratix-10 with the proposed Extra Carry Chain architecture modifications (highlighted in red).

2.2 Stratix-10 ALM Architecture

In later portions of this paper we will show that a key factor in the efficiency of a logic block's implementation of a multiply operation is how many arithmetic functions it can perform per LUT/logic element. This leads us to choose Stratix-10, which has 2-bits of hardened arithmetic per ALM, as the baseline against which we will compare our enhancements. Fig. 1 shows the ALM architecture in Intel Stratix-10 FPGAs [11]. Each ALM is a 6-LUT that can be fractured into two 5-LUTs. It has 2 bits of arithmetic (i.e. two full adders) with dedicated routing wires for the carries, 8 distinct inputs (A-H), and 4 outputs (O1-O4). An ALM can operate in two major different modes as follows:

(1) The normal mode in which the ALM can implement either a 6-input logic function, or two 5-input or smaller functions that together use no more than 8 distinct inputs. This allows implementation of two independent 4-LUTs, or two 5-LUTs that share two inputs and so on.
(2) The arithmetic mode in which four 4-LUTs feed the four inputs of the two full adders. In this mode, the 4-LUTs can implement identity functions to simply pass inputs to the adders or they can implement simple pre-addition logic if all four 4-LUTs use six or fewer distinct inputs.

A logic array block (LAB) contains 10 ALMs along with a local routing crossbar that allows connections from ALM outputs to inputs within the LAB and connections from the general (inter-logic-block) routing wires to the ALM inputs. There are also direct-connect wires which let ALM outputs in one LAB directly drive the local interconnect in the LABs to its immediate left and right, thereby allowing high speed nearest neighbor connections without use of the general routing. Each ALM can drive four outputs to the general routing, for a total of 40 outputs per LAB. Dedicated carry chain routing wires run between the ALMs of the same LAB to form multi-bit adders, and dedicated routing between vertically adjacent LABs allows wider adders to be efficiently constructed.

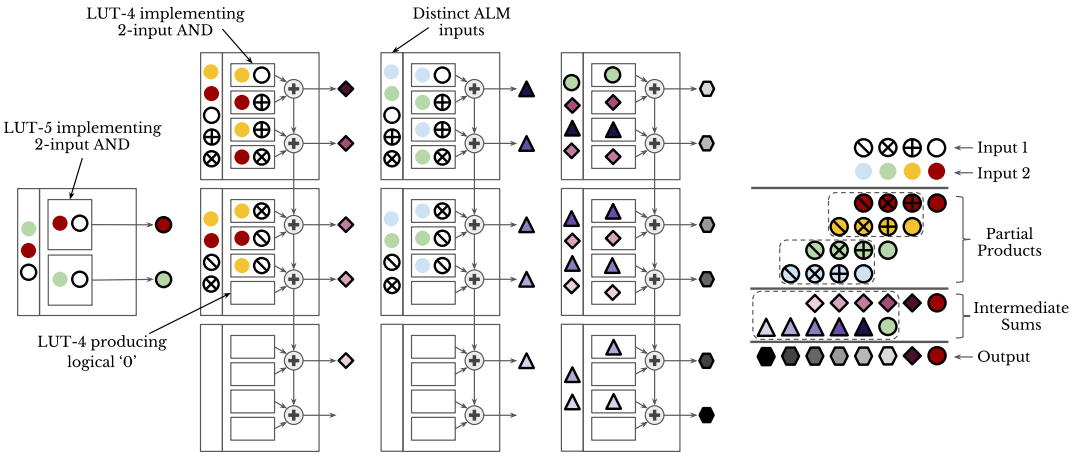

Figure 2: The mapping of a 4-bit unsigned multiplier to the current Stratix-10 ALM architecture.

2.3 Low-Precision Multipliers on FPGAs

Recent work has proposed DSP block architectural changes to support 9-bit and 4-bit multiplication and MAC operations for low-precision deep learning on FPGAs [4]. Despite doubling and quadrupling the capacity of DSP blocks in implementing 9-bit and 4-bit multipliers respectively, the overall performance of CNN inference improved by only a factor of 1.3× and 1.6×. The reason is twofold; Firstly, DSP blocks consume only 5% of the FPGA core area in current DSP-rich architectures which significantly dilutes the gain from making them capable of performing more low-precision multiplications. Secondly, as the multiplication bitwidth decreases, the multiplier array size shrinks quadratically and thus implementing small multiplications in the abundantly available logic blocks becomes a viable option. These reasons motivate the investigation of architectural changes at both the ALM and LAB levels to increase the efficiency of the FPGA's soft fabric in implementing multipliers in general, and more specifically smaller ones of size 4- to 9-bits.

In order to understand how multipliers are mapped to the Stratix-10 ALM architecture, we experimented with different sizes of multipliers ranging from 4-bit up to 9-bit on Quartus Prime Pro 17.1. For simplicity, we will explain the mapping of an unsigned 4-bit multiplier but we also found in our experiments that a signed or unsigned multiplier of any size from 4-bit to 9-bit is mapped using the same approach. In Fig. 2, the two multiplicands are represented by either patterned or coloured circles. The combination of a colour and a pattern represent an AND operation between the two corresponding bits while the reduction of two patterned coloured circles (i.e. two partial product bits) results in other shapes.

Most of the ALMs are used in the arithmetic mode such that the 4-LUTs implement the AND gates to produce the partial product bits which are then added using the hard carry chains. In a 4-bit multiplier, three and a half ALMs are needed to generate and add each of the two pairs of partial products in the first reduction stage as shown in Fig. 2. Then the second stage of reduction requires three additional ALMs to produce the final result. This results in a total of 10 ALMs for a 4-bit multiplier.

To validate our choice of the Stratix-10 ALM as a baseline, we also performed similar experiments in mapping 4- to 9-bit multiply-accumulate units to a Virtex Ultrascale+ device using Vivado 2018.1. On average 70% more fracturable LUTs were required than when targeting Stratix-10 ALMs. For both the Virtex Ultrascale+ and Stratix-10 architectures we iteratively shrank the floorplan until compilation failed in order to ensure we had found the densest mapping possible in each device.

2.4 CAD for FPGA Circuit Design

To evaluate our proposed architectures, we must determine their impact on the logic tile (logic block plus its associated routing) area, as well as the delay of many paths within the tile. We use the COFFE FPGA transistor sizing and modeling tool for this purpose [8]. Given a description of the tile architecture, COFFE builds the relevant subcircuits for SPICE simulation, estimates layout area, adds wire loads, and iteratively optimizes the sizing of each type of transistor. By directly using SPICE for delay estimation, COFFE can accurately model delay in recent process nodes, and its full custom circuitry and transistor sizing approach matches the design style of commercial FPGA logic tiles. We use the latest release of COFFE, which more accurately models wire loads by creating and optimizing a floorplan of the logic tile, and whose area and delay estimates have been shown to correlate well with published commercial FPGA values [25]. This version of COFFE can also implement logic in standard cells for heterogeneous blocks, which allows us to incorporate standard cell logic within the logic tile for one of our architecture variants (shadow multipliers in Section 3.3).

The latest release of COFFE can model fracturable LUTs and either one or two bits of arithmetic per fracturable LUT [25]. However, it still requires considerable modifications to enable modeling more sophisticated ALM architectures similar to those of state-of-the-art commercial FPGAs as shown in Fig. 1. Our enhancements to COFFE will be discussed later in Section 4.2.

3 LOGIC ARCHITECTURE ENHANCEMENTS

In this section, we present three different architectural enhancements to FPGA logic blocks, two of which are on the ALM level while the third is on the logic cluster (i.e. LAB) level. For each architectural enhancement, we describe the circuitry added to the ALM or the LAB and show, as a simple example, how a 4-bit unsigned multiplier would map to the soft logic of the enhanced FPGA.

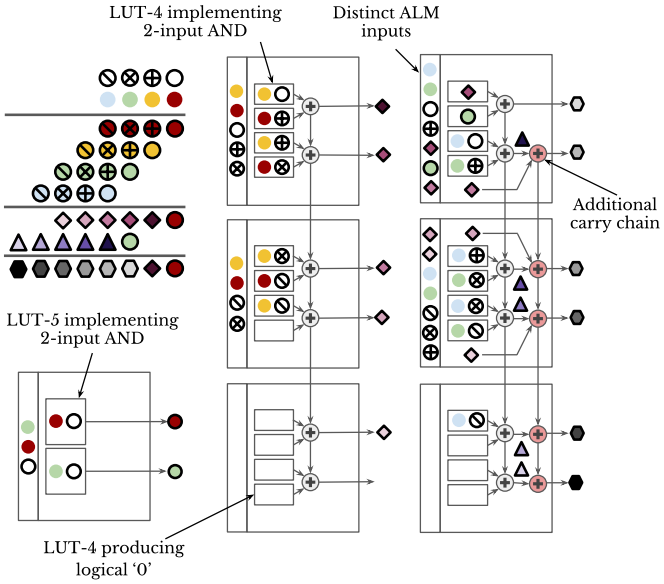

Figure 3: Mapping a 4-bit unsigned multiplier to the Extra Carry Chain architecture.

3.1 Extra Carry Chain

When examining the mapping of multipliers to the Stratix-10 soft logic as described in Section 2.3, we can observe that the LUTs inside the ALMs are used to implement the partial product generation 2-input AND gates in the first reduction stage. After that, only the carry chains are used to implement the subsequent reduction stages while the LUTs are used as identity functions to feed the hard adders with inputs as shown in the rightmost ALM column in Fig. 2. This results in an inefficient ALM utilization and emphasizes the need to implement more efficient adder reduction trees in order to increase the on-chip MAC density.

To address this, we propose adding another carry chain to the ALM such that the two new adders get their inputs from the two sum outputs of the first carry chain and the two ALM inputs (E and F) that are left unused in the arithmetic ALM mode as shown in Fig. 1. By doing this, we can use the second level of adders to perform another stage of reduction within the same ALMs instead of using the adders of another set of ALMs and leaving their LUTs unusable, as is the case in the current Stratix-10 architecture. Fig. 3 shows the mapping of a 4-bit multiplier to the proposed ALM architecture with an additional carry chain. The first pair of partial products are generated and added in 3 ALMs and then reduced with the second pair in another 3 ALMs using the added carry chain. This results in a mapping that consumes only 7 ALMs instead of the 10 ALMs required in the baseline Stratix-10 ALM architecture. This architectural enhancement does not require additional inputs or outputs to or from the ALM and thus the area overhead of the two adders and the 2:1 multiplexers allowing them to be bypassed (highlighted in red in Fig. 1) is minimal. The additional carry chain decreases the number of ALM levels not only for multipliers but also for adder reduction trees in general. For example, a 3-to-1 adder would require only one ALM level in the Extra Carry Chain architecture compared to two levels with the current Stratix-10 architecture.

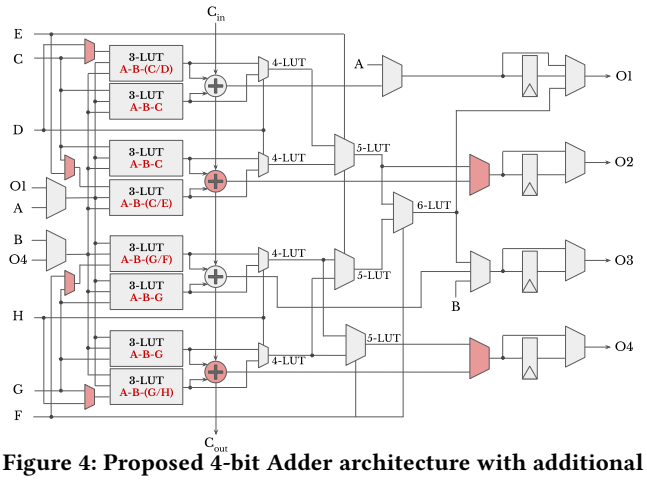

Figure 4: Proposed 4-bit Adder architecture with additional full adders and multiplexing (highlighted in red).

3.2 Deeper Fracturability: 4-bit Adder

Another approach to increase the density of MAC operations is to harden more adders per ALM to create a single but wider carry chain. Since multipliers use the 4-LUTs before adders to implement only two-input AND gates as shown in Fig. 2, the 4-LUTs are extremely underutilized. Therefore, we propose going one level deeper with ALM fracturability by splitting each 4-LUT into two 3-LUTs, followed by four bits of arithmetic instead of two as shown in Fig. 4. To compute and sum eight partial products in this ALM we must configure each of the eight 3-LUTs as a 2-input AND gate. This requires a total of 16 inputs. However, since there are many shared signals, the number of distinct inputs is only eight matching the ALM's input ports count. The reason is the input sharing nature of multiplier arrays which produce N^2 partial product bits from only $2N$ input bits for any multiplier bitwidth N. Fig. 5 shows the mapping of an unsigned 4-bit multiplier to ALMs with 4 bits of arithmetic. It shows that none of the used ALMs has more than 8 distinct inputs. This observation holds for signed and unsigned multipliers of all sizes.

Although we do not need additional input ports, we need to ensure that we can deliver the correct inputs to the 3-LUTs both to implement multiplier arrays and also to use the adders to implement a standalone 4-bit adder per ALM for general design use. For this reason, we add the small 2:1 multiplexers (highlighted in red in Fig. 4) in front of four out of the eight 3-LUTs. These multiplexers provide us with enough flexibility to deliver the 8 ALM inputs to the 3-LUTs to implement equations (1) and (2) in case of multiplier arrays and standalone adders respectively. Table 1 shows the assignment of ALM inputs (A-H in Fig. 4) as well as the function that each of the eight 3-LUTs implements for both scenarios. Also, to be able to output the result of all four adders, we add two 2:1 output select multiplexers before the FFs for outputs O2 and O4 as shown in Fig. 4.

$$\begin{array}{cccc} a_0b_4 & a_0b_3 & a_0b_2 & a_0b_1 \\ a_1b_3 & a_1b_2 & a_1b_1 & a_1b_0 \; + \end{array} \tag{1}$$

$$\begin{array}{cccc} a_3 & a_2 & a_1 & a_0 \\ b_3 & b_2 & b_1 & b_0 \; + \end{array} \tag{2}$$

Table 1: ALM input assignment and LUT masks for implementing multiplier arrays and standalone adders (Eq. (1) and (2), receptively) using the 4-bit Adder architecture.

Input	L1	L2	L3	L4	L5	L6	L7	L8
Mult (Eq. 1)	B&D (a_1b_0)	A&C (a_0b_1)	B&C (a_1b_1)	A&E (a_0b_2)	B&F (a_1b_2)	A&G (a_0b_3)	B&G (a_1b_3)	A&H (a_0b_4)
Add (Eq. 2)	D (a_0)	C (b_0)	A (b_1)	E (a_1)	F (a_2)	B (b_2)	G (a_3)	H (b_3)

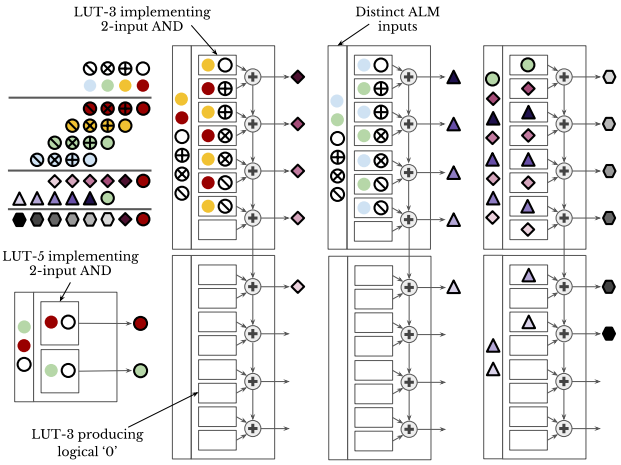

Figure 5: Mapping a 4-bit unsigned multiplier to the 4-bit Adder architecture.

3.3 Shadow Multipliers

In [12], Jamieson and Rose proposed *shadow clusters*. These are normal FPGA logic clusters added to hard blocks such as BRAMs or DSP blocks which can be used only when the hard blocks are not used. The motive was to increase the area efficiency for applications that do not fully utilize the hard blocks on an FPGA. However, for DL applications, DSP blocks are more valuable and are usually the bottleneck when implementing DL accelerators on current FPGAs [5]. Therefore, a more effective way to increase the density of on-chip MAC operations is to add a *shadow* hard multiplier within each logic cluster. In order to avoid adding any extra local or inter-tile routing area, this shadow multiplier borrows the input and output ports of some of the ALMs in the cluster; no extra ALM input muxes are built and no cluster inputs or outputs are added. As shown in Fig. 6a this makes some ALMs unusable when the shadow multiplier is in use.

We evaluate hardening various shadow multiplier sizes. As the hard multiplier bitwidth increases, the logic cluster area overhead increases and it leaves more ALMs unusable since it needs more input and output ports. On the other hand, a larger hard multiplier results in more efficient implementations of larger multiplies in the soft logic. Therefore, this design choice relies heavily on the multiplication bitwidths used by the target application. We experiment with shadow multiplier sizes ranging from 4- to 9-bits and show the trade-off between ALM savings and cluster area overhead in the results section of this paper.

FPGA applications have diverse needs for multiplication: there will be a variety of multiplication precisions, and a mix of signed and unsigned multiplication. To ensure our shadow multiplier is as flexible as possible, we design a special multiplier array, shown in Fig. 6b, that enables efficient implementation of two's complement signed multiplications of bigger sizes. The invertible cells (Fig. 6d) and the signed/unsigned cells are added to implement a Baugh-Wooley signed multiplication [3], and are marked with crossed circles and 'S' symbols respectively. The 'S' bits are 1 for a signed multiply and 0 otherwise, and the invertible cells are controlled using three different sign control signals ($C1, C2$ and $C3$) in our multiplier array instead of a single control signal in conventional ones.

Fig. 6e illustrates how a 6-bit signed multiplier can be implemented using a 4-bit shadow multiplier built using a conventional multiplier array. The hard multiplier is forced to implement the lower left corner of the multiplier array to align the invertible cells in the hard multiplier with the positions where they are needed in a 6-bit multiplier. This results in 5 partial results to be reduced in the soft logic in addition to extra logic to correct the contamination of the misplaced 'S' bit from the hard multiplier.

However, when using a multiplier array with three distinct sign control signals, we can invert the corner cell, disable the inversion of the bottom boundary cells and set the green S to 0, as shown in Fig. 6f. This enables the hard multiplier to implement the top left corner of the multiplier array resulting in no contamination bits and only 3 partial results to reduce in the soft logic. Larger multiplications can also be mapped to this architecture by combining several hard multipliers. For instance, an 8-bit multiplication can be implemented using four 4-bit multipliers and one level of carry chain to combine their outputs.

4 EVALUATION

4.1 ALM Savings for Multiplies and MACs

To quantify the increase in multiply and MAC operation density achieved by each of the three architectural changes presented in Section 3, we hand-map multipliers and MAC units of sizes ranging from 4- to 9-bit to the logic blocks of a Stratix-10-like architecture as well as the three proposed ones. We verify our mapping for the baseline architecture with synthesis results from Quartus 17.1. For MAC units, Quartus does not perform any cross-boundary optimizations between the multiplier array and the accumulator and therefore we follow the same approach in our hand-mapping to maintain a fair comparison across architectures. We present results where the accumulator is the same width as the multiplier output; however any other assumption leads to the same trend in results.

Fig. 7 shows the ALMs required for 4- to 9-bit multiplier and MAC units with a baseline ALM, as well as our Extra Carry Chain and 4-bit Adder architectures. Both proposed architectures outperform the baseline across all multiplier and MAC precisions. For standalone multipliers, the Extra Carry Chain and the 4-bit Adder architectures reduce ALM usage by 29% and 36% on average across this range of multiplier sizes, respectively. For MAC operations both proposed architectures perform similarly, with the Extra Carry Chain and 4-bit Adder architectures reducing average ALMs per MAC operation

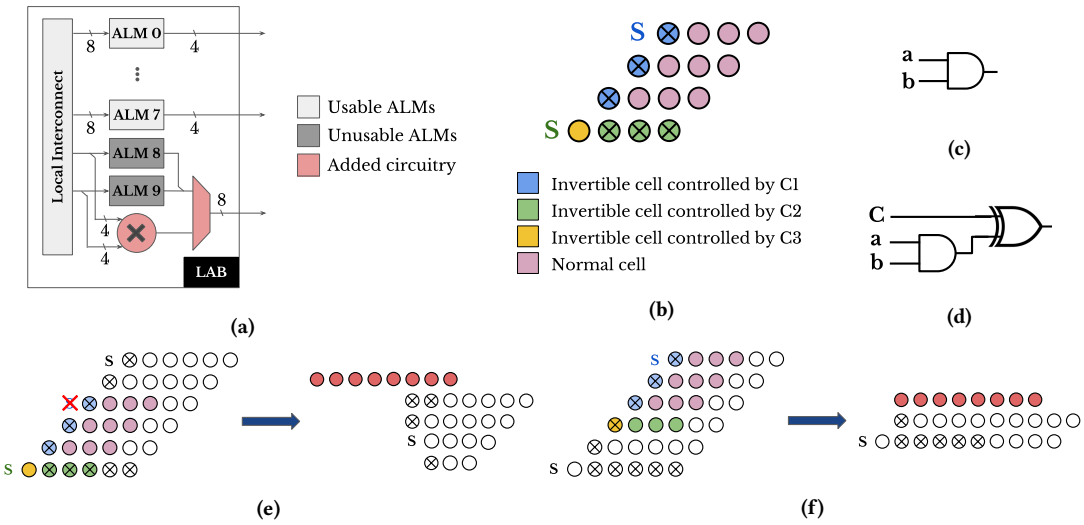

Figure 6: Shadow multiplier: (a) Hard multiplier placement in logic block, (b) Hard multiplier design, (c) Normal multiplier cell, (d) Invertible multiplier cell, and 6-bit multiplication mapped to 4-bit hard multiplier (e) using conventional multiplier array and (f) using enhanced multiplier array with three distinct sign control signals.

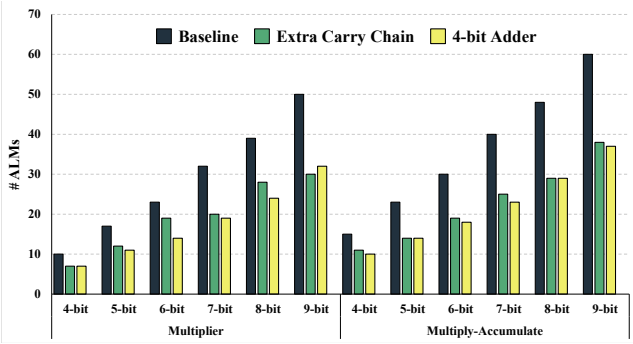

Figure 7: ALM utilization for 4- to 9-bit multipliers and MACs on Stratix-10, Extra Carry Chain and 4-bit Adder architectures.

vs. the baseline by 36% and 38%, respectively. The Extra Carry Chain architecture can implement the accumulation adder in the second carry chain of the last reduction stage of the multiplier array so its advantage over the baseline architecture is larger on MAC operations than on multiplies.

We map multipliers and MACs to the Shadow Multiplier architecture discussed in Section 3.3 to achieve the highest reduction in ALM count rather than the shortest critical path delay. The average ALM savings increase with the hard multiplier size; for instance, when implementing a 9-bit multiplier using the 9-bit Shadow Multiplier architecture only 5 ALMs are consumed since they were made unusable by selecting the output of the multiplier (see Fig. 6a). On average for 4-bit to 9-bit MAC operations, the reduction in ALMs used or unusable vs. the baseline varies from 46% for a 4-bit Shadow Multiplier to 88% for a 9-bit Shadow Multiplier architectures. These are larger reductions than those achieved by the Extra Carry Chain or 4-bit Adder architectures, but the Shadow Multiplier architecture also adds more area to the FPGA tile.

In Section 3 we ensured that our logic enhancements respect the architectural constraint that each ALM can use no more than 8 inputs. There is also a constraint on the number of inputs to a logic block – in Stratix-10 the number of LAB local routing wires is 60, and therefore the sum of the number of distinct input signals to all ALMs within a LAB must be at most 60 [17]. While our architecture enhancements increase MAC density, all the LABs produced for all 3 enhanced architectures for MAC units from 4- to 9-bits still fit comfortably within this local routing limit. The 4-bit Adder architecture has the highest LAB local routing demand, but even in this case, the MAC mappings we choose balance partial product (low-input-demand) and adder tree (higher-input-demand) operations in LABs and the worst-case LAB local routing demand is only 46 (out of 60) wires.

4.2 COFFE Flow: Extensions and Technology

Section 4.1 showed that the three proposed architectural changes will lead to considerable savings in the total number of ALMs needed to implement multipliers on the soft fabric. However, the architecture changes will also increase the size of the logic block and will impact the delay of some paths within the logic block as well; for a complete evaluation we need to compute these overheads. As mentioned previously, we extend the COFFE CAD tool in several ways so that it can model and evaluate these new architectures. First, we add flexible control over the way inputs connect to fracturable LUTs; this enables us to better capture the functionality of a Stratix-10 ALM which is our baseline architecture. Second, we add new options for carry chain architectures. COFFE could model 1-bit and 2-bits of carry chain per ALM/fracturable LUT; we added support for 4 bits of arithmetic per fracturable LUT. We also extended COFFE to support two cascaded carry chains per ALM. Next, we modified COFFE to support deeper fracturing of LUTs; this was particularly important to enable efficient use of the large number of arithmetic bits per fracturable LUT in some of our proposed architectures. COFFE previously supported fracturing a 6-LUT into two 5-LUTs; we extended COFFE so it can now also fracture a 6-LUT into four 4-LUTs or eight 3-LUTs. These deeper levels of fracturing are needed

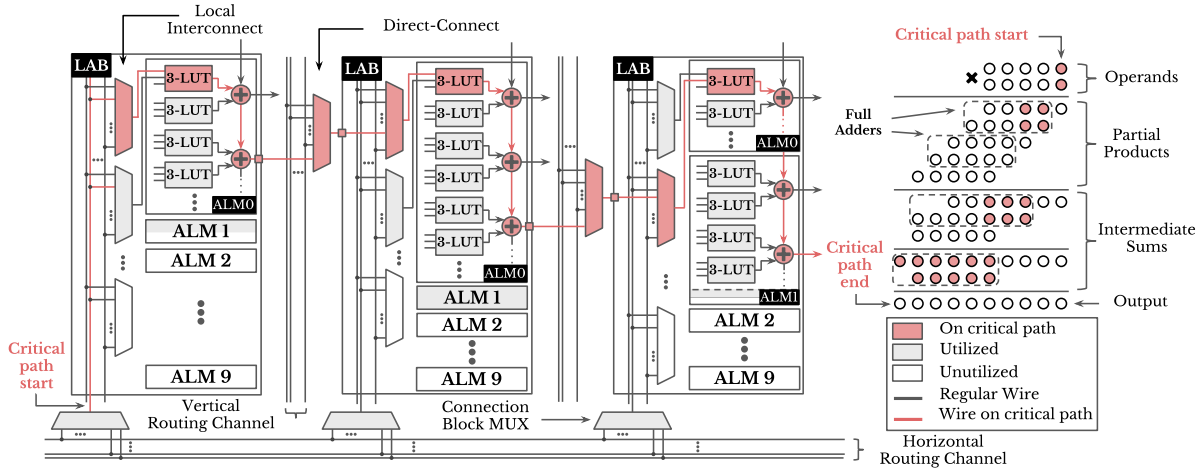

Figure 8: Critical path of a 5-bit multiplier mapped to the 4-bit Adder architecture.

Table 2: Routing and tile architecture parameters.

Parameter	Value	Parameter	Value
N (ALMs/LAB)	10	Fs	3
W (Channel width)	320	Fcin	0.2
L (Wire segment length)	4	Fcout	0.025
I (Inputs/LAB)	40	Fclocal	0.5

to model the LUTs feeding the carry chain in Stratix-10 and in one of our proposed architectures, respectively. We also added finer control over COFFE's intra-ALM routing so we could add multiplexers where necessary to make these new features as useful as possible. Since wire load affects both transistor sizing and delay significantly, we also created a floorplan for the proposed ALM that is used within COFFE to estimate all the intra-ALM wire lengths.

With these modifications, COFFE now has three additional modes of operation which allow us to perform area and delay measurements for three new architectures: the Stratix-10-like architecture, Stratix-10 with a second level of carry chain adders (both shown in Fig. 1) and finally the deeper fracturability design with 4-adders per ALM (Fig. 4). We believe these new features open up new areas for exploration and will be helpful to future FPGA logic block architecture research as well[1].

To evaluate the area and delay impact of shadow multipliers we created a structural system verilog implementation of our enhanced multiplier array with 3 distinct sign controls and parameterized precision. DSP block multiplies in FPGAs are typically implemented with standard cells, and accordingly we used a standard cell flow for the shadow multiplier: Synopsys Design Compiler 2013.03 for synthesis, Cadence Innovus for place and route, and 28 nm ST Microelectronics standard cell libraries. We used COFFE's full custom flow with 22 nm HP predictive technology model SPICE decks [1], so we scale the area of the resulting standard cell block to 22 nm and input it to COFFE, along with the area of the input drivers and output select multiplexers (see Fig. 6a). This allows COFFE to model the increased wire length due to the extra tile area, and to resize buffers where appropriate to cope with the larger loads.

[1]This enhanced version of COFFE is available at: https://github.com/vaughnbetz/COFFE/tree/lbChanges

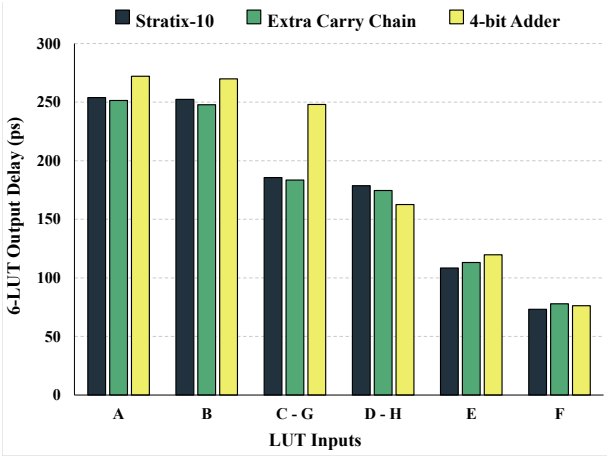

Figure 9: LUT input delays for the Stratix-10, Extra Carry Chain and 4-bit Adder architectures.

4.3 Tile Area and Delay Cost

COFFE sizes transistors to optimize a user-specified cost function of area and delay. For this study we have chosen a cost function of $area \cdot delay^2$ as it reflects the greater emphasis on delay compared to area typical in high-performance FPGAs like Stratix-10. The routing architecture and logic cluster parameters are chosen to match those in [8], as they are representative of recent Stratix series architectures; these parameters are summarized in Table 2.

Fig. 10 shows the area breakdown of the baseline, Extra Carry Chain, and 4-bit Adder architectures along with 4- and 9-bit Shadow Multiplier architectures using the baseline ALM. The Extra Carry Chain architecture and the 4-bit Adder architecture show small tile area increases over the baseline, of 2.6% and 2.9%, respectively. The major contributor to this increase is doubling the number of full adders per ALM for both architectures, which led to an approximately 2% area increase. The remainder of the area increase is primarily due to the extra multiplexing required by these architectures – as Fig. 1 and 4 show there are also two and six 2:1 multiplexers added in the Extra Carry Chain and 4-bit Adder architectures vs. the baseline, respectively.

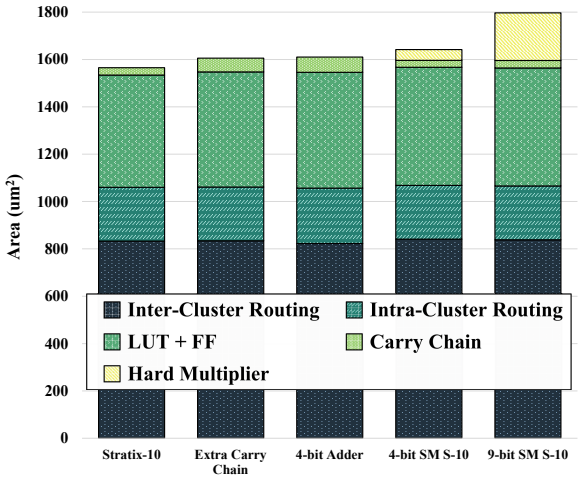

Figure 10: Tile area breakdown for various architectures.

As shown in Fig. 10, the 4- and 9-bit shadow multipliers increase the logic tile area more significantly, by 5.0% and 14.8% respectively. The main contributor to this increase in area is the added shadow multiplier which contributed a 2.9% and 12.8% area increase in case of 4- and 9-bit shadow multipliers, respectively. The remaining portion of the overall area increase is due to the increase in buffer sizes to drive the intra- and inter-cluster wires which have become longer due to the hard multiplier's area. The tile area increase also leads to some slowdown in the programmable routing due to longer wire-lengths. For the 4-bit shadow multiplier case, the delay of a direct (nearest neighbor) connection and a length 4 routing wire have increased by 4.4% and 1%, respectively. A 9-bit shadow multiplier increases routing delay more: by 9.5% for a direct connection and 6.5% a length 4 routing wire.

While the small area increases of the Extra Carry Chain and 4-bit Adder architectures mean that the routing wire delays are not significantly impacted, some LUT delays are. The delay of each LUT input from the local interconnect of the LAB to the 6-LUT output is shown in Fig. 9 for each ALM architecture. This figure shows the delays for the 8 inputs of the ALM; note that input C has exactly the same delay as input G and similarly inputs D and H have the same delay.

For the Extra Carry Chain architecture, inputs A, B, C, and D have a slightly lower delay compared to the baseline architecture; this is due to small variations in wire load and transistor sizing decisions made by COFFE. Inputs E and F have experienced a 10.4% and 4.2% increase in delay, respectively. In this architecture a connection is added from both these inputs to the new full adders in the ALM, increasing the capacitive loading of these inputs and hence increasing their delay. Overall the Extra Carry Chain architecture leads to very little change in the key delay paths in the tile.

The 4-bit Adder architecture has more impact on tile delay, and shows an increase in delay for all of its inputs except D and H. As Fig. 4 shows, this architecture contains eight 3-input LUTs feeding four adders. For the Stratix-10 baseline and Extra Carry Chain architectures there are instead four 4-LUTs in front of the adders, and COFFE speeds up these 4-LUTs by inserting buffers after the first two stages of pass transistors, evenly dividing the 4 cascaded pass gates in the 4-LUTs. In the 3-LUTs of the 4-bit Adder architecture

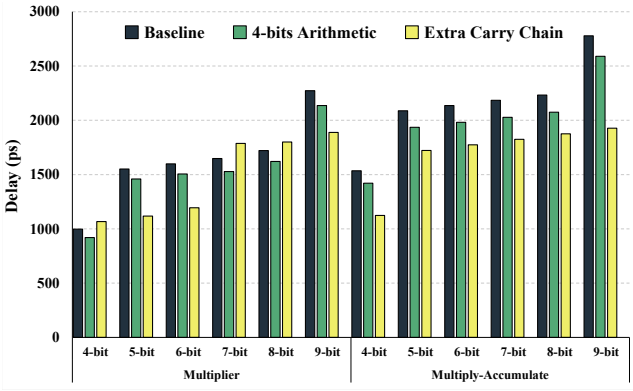

Figure 11: Multiplier and MAC critical path delays for Stratix-10, Extra Carry Chain and 4-bit Adder architectures.

such even buffering is no longer an option, and instead COFFE implements 3 stages of pass gate with no internal buffering to realize the 3-LUTs with a buffer only at the LUT output. This leads to a 7% delay increase for inputs A and B. Input C's delay increases by 34% as it is also impacted by the extra 2:1 multiplexers (see Fig. 4) added to ensure all the necessary signals can reach the 3-LUTs when in multiplier mode.

4.4 Overall MAC Area and Delay

As mentioned in Section 4.1, we hand map multipliers and MAC units of 4-bit to 9-bit precisions for each architecture evaluated. This allows us to calculate the number of ALMs used (or rendered unusable) by the mapping and to determine the logic and routing on the critical path. By combining these ALM counts and critical paths with the tile areas, logic delays and routing component delays computed by COFFE and summarized in Section 4.3, we can determine the overall multiplier and MAC speed and density achieved by each architecture.

We implement all multipliers as AND gates (i.e a partial product using LUTs) followed by a sequence of carry chains to sum the partial products. Fig. 8 shows the critical path of a 5-bit multiplier implemented using the 4-bit Adder architecture. The right hand-side of the figure shows the partial products and how the addition reductions are performed. The solid circles show the operands and the partial products on the critical path. The rest of the figure shows how this multiplier is mapped to the soft logic. The addition operations occurring on the critical path need 6 ALMs which could fit in one LAB; however, for clarity we map them to different LABs in this figure. For the intra-ALM routing we assume that the critical path signal will use the direct-connect between (or within) LABs and have a delay of a directly-driven (no multiplexer) wire spanning one LAB, plus the delay of a connection block and the local cluster interconnect. This is not a best case placement and routing as the local cluster routing is slightly faster than direct connect, but it represents a high quality placement and routing and is applied consistently for all architectures.

Fig. 11 shows the delay of 4- to 9-bit multiply and MAC operations mapped to the baseline, Extra Carry Chain and 4-bit Adder architectures. The results show that for the multiply operations over all the multiplier sizes chosen in this study, the 4-bit Adder

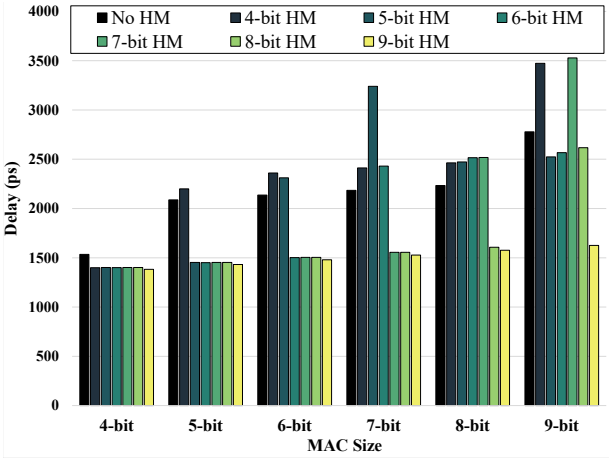

Figure 12: MAC critical path delays for Stratix-10 architectures with shadow multiplier sizes from 4- to 9-bit.

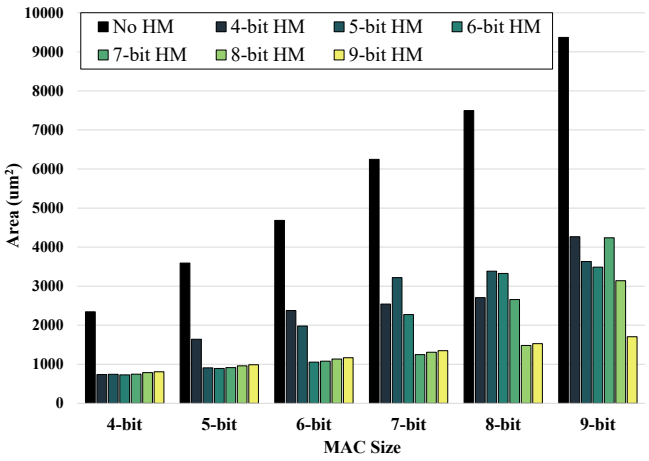

Figure 13: MAC areas for Stratix-10 architectures with shadow multiplier sizes from 4- to 9-bit.

and the Extra Carry Chain architectures show an average reduction of 3.9% and 8.4% in the critical path delay respectively over the baseline architecture. For MAC operations the delay reduction increases to 7.2% and 20% in the case of the 4-bit Adder and Extra Carry Chain architectures, respectively; the extra adders in these architecture benefit MAC even more than multiply. The speed gains are larger with the Extra Carry Chain architecture because it is able to pack two levels of an adder reduction tree in one LAB, reducing the number of (direct connect) routing hops.

For the Shadow Multiplier architecture, the way the multipliers are implemented depends on the hard multiplier size available in the logic block as well as the size of the multiplier to be synthesized. If the size of the multiplier to be synthesized is smaller than or equal to the size of the hard multiplier size, then the critical path delay will simply be the LAB input to LUT input delay, hard multiplier delay and the output select multiplexer delay. As discussed in Section 3.3 larger multiplies can be implemented either by combining a single shadow multiplier with ALM logic to complete the multiplier array, or by combining multiple shadow multipliers with ALM carry chains to sum their outputs. We evaluate both options and choose the one with smaller area (used plus unusable ALMs). The critical path delays and the areas required to implement 4- to 9-bit MACs using 4- to 9-bit Shadow Multiplier architectures are shown in Fig. 12 and Fig. 13. The critical path delay of a MAC operation increases with the multiplication size to the point were multiple hard multipliers can be combined without significantly impacting the area. For instance, Fig. 12 shows that the critical path delays of 6-bit to 8-bit MACs are almost the same when implemented on the 4-bit Shadow Multiplier architecture; this is because all 3 MAC sizes map to four 4-bit hard multipliers. For the same reason, Fig. 13 shows that there is almost no change in area for this 6- to 8-bit MAC precision range on a 4-bit Shadow Multiplier architecture.

Table 3 summarizes the performance of all the proposed architectures, along with a fifth architecture which is a combination of the Shadow Multiplier architecture and the 4-bit Adder architecture. We show multiple variations of the Shadow Multiplier architectures using three different hardened multiplier sizes (4, 6, and 9-bit). We are interested both in what gains we can achieve when the architecture is used for MACs and what cost it entails in terms of tile area and delay for other circuits. In order to evaluate the impact of our proposed architectures on the delay of a typical design which does not exploit our new features, we used the notion of the representative critical path delay [14]. From our experience the representative critical path delay is divided into 70% routing delay, 25% logic delay and 5% carry chain delay; using these ratios we can combine the COFFE delay numbers for each architecture into the single delay metric shown in the table. The average MAC critical path delay and average MAC area numbers are geometric averages over 4-bit to 9-bit MAC operations. The bold numbers in each row in Table 3 show the best value for each category.

The Shadow Multiplier architectures achieve higher MAC area reductions compared to the Extra Carry Chain and the 4-bit Adder architectures. MAC area is further reduced by combining the 4-bit Adder architecture with the Shadow Multiplier architecture – this combination achieves an 83.6% average MAC area reduction with a 9-bit shadow multiplier. Shadow multipliers of size 6-bit and larger are faster than the Stratix-10 baseline and have MAC delay reductions comparable to the Extra Carry Chain and the 4-bit Adder architectures. The 4-bit Shadow Multiplier architecture is the slowest architecture (8% slower than the baseline) so 6-bit or larger shadow multipliers are more desirable. The high MAC area reductions of 6-bit and larger shadow multipliers unfortunately come with a considerable increase in the FPGA tile area and the representative critical path delay, and this cost grows with shadow multiplier size. Accordingly the 6-bit shadow multiplier seems like the best size: it achieves a 68% MAC area reduction at a reasonable cost of 2.3% representative critical path delay increase and 3.8% tile area growth. Combining a 6-bit shadow multiplier with the 4-bit Adder architecture reduces MAC area, but increases the area and delay penalties for general logic.

On the other hand, the Extra Carry Chain architecture shows the smallest increase in representative critical path delay (0.8%) and the smallest tile area increase (2.6%) over the Stratix-10 baseline so it is the least intrusive change in terms of its impact on general applications. Despite its small cost, the Extra Carry Chain architecture reduces the average MAC delay by 20.9% and the average MAC

Table 3: Area and Delay summaries for Stratix-10 architecture, Extra Carry Chain architecture, 4-bit Adder architecture, 4-, 6-, and 9-bit Shadow Multiplier architectures built on Stratix-10 and 4-bit Adder architectures.

Architectures	Stratix-10	Extra Carry Chain	4-bit Adder	SM on Stratix-10			SM on 4-bit Adder		
				4-bit	6-bit	9-bit	4-bit	6-bit	9-bit
Rep. Crit. Path Delay (ps)	119.2	**120.2 (+0.8%)**	122.5 (+2.7%)	121.1 (+1.6%)	122.0 (+2.3%)	126.6 (+6.2%)	125.1 (+4.9%)	126.5 (+6.1%)	130.9 (+9.8%)
FPGA Tile Area (um^2)	1565	**1605 (+2.6%)**	1610 (+2.9%)	1642 (+4.9%)	1624 (+3.8%)	1797 (+14.8%)	1629 (+4.1%)	1702 (+8.8%)	1826 (+16.7%)
Avg. MAC crit. Path Delay (ps)	2127	1682 (-20.9%)	1975 (-7.2%)	2304 (+8.3%)	1906 (-10.4%)	**1502 (-29.4%)**	2542 (+19.5%)	2042 (-4.0%)	1528 (-28.2%)
Avg. MAC Area (um^2)	5097	3337 (-34.9%)	3205 (-36.8%)	2095 (-58.8%)	1621 (-68.1%)	1218 (-76.0%)	1166 (-77.1%)	996 (-80.4%)	**835 (-83.6%)**

area by 34.9%. As this architecture benefits not only multiply and MAC operations but also any adder tree, it is also likely to result in area and delay reductions in other arithmetic applications.

Finally, the 4-bit Adder architecture achieves a slightly higher MAC area reduction (36.8%), with a slightly higher representative critical path delay increase (2.7%) and tile area increase (2.9%) than the Extra Carry Chain architecture; this makes it another fairly low-risk ALM change. This architecture will also benefit not just multiplication as it can implement all standalone adders and subtractors in half as many ALMs as in the Stratix-10 baseline. Given that arithmetic functions constitute over 20% of the logic primitives in recent benchmark sets [18] and most map to standalone adders or subtractors, this architecture is likely to yield significant ALM count reductions for a wide range of designs.

5 CONCLUSION AND FUTURE WORK

In this paper we proposed three different architectures which increase the MAC density in FPGAs, thereby improving FPGA performance on deep learning applications. The Extra Carry Chain architecture adds a second level of carry chains in the ALMs, making the reduction trees of the multiply operation more efficient. The 4-bit Adder architecture widens the carry chains in each ALM and increases the level of fracturability of their LUTs. The Shadow Multiplier architecture adds hard multipliers to the logic blocks without adding programmable routing, thereby keeping its cost lower.

We extended the COFFE transistor sizing and optimization tool to support these new architectures, and used it to generate detailed area and delay models. With a small impact on the tile area (+2.6%) and the representative critical path delay (+0.8%), the Extra Carry Chain architecture can achieve a 21% and a 35% reduction in average MAC delays and areas, respectively. The 4-bit Adder architecture achieves slightly better MAC area reductions, at the cost of slightly more tile area and representative critical path delay impact. MAC area reductions as high as 84% (representing a 6.1× increase in MAC density) can be achieved by combining the Shadow Multiplier and 4-bit Adder architectures. However, this comes with higher costs for general circuits: a tile area overhead of 16.7% and representative critical path delay increase of 9.8%.

ACKNOWLEDGMENTS

The authors would like to thank Huawei and the Vector Institute for funding support.

REFERENCES

[1] 2018. Predictive Technology Model (PTM). http://ptm.asu.edu/
[2] E. Ahmed and J. Rose. 2004. The effect of LUT and cluster size on deep-submicron FPGA performance and density. *TVLSI* 12, 3, 288–298.
[3] C. Baugh and B. Wooley. 1973. A two's complement parallel array multiplication algorithm. *TC* 100, 12.
[4] A. Boutros et al. 2018. Embracing diversity: Enhanced DSP blocks for low-precision deep learning on FPGAs. In *FPL*. 1–8.
[5] A. Boutros et al. 2018. You cannot improve what you do not measure: FPGA vs. ASIC efficiency gaps for convolutional neural network inference. *TRETS* 11, 3.
[6] M. Burich. 2012. Conference workshop: FPGAs in 2032, challenges and opportunities in the next 20 years, convergence of programmable solutions. In *FPGA*.
[7] S. Chandrakar et al. 2015. Enhancements in UltraScale CLB architecture. In *ISFPGA*. 108–116.
[8] C. Chiasson and V. Betz. 2013. COFFE: Fully-automated transistor sizing for FPGAs. In *FPT*. 34–41.
[9] J. Fowers et al. 2018. A configurable cloud-scale DNN processor for real-time AI. *ISCA*, 1–14.
[10] I. Goodfellow et al. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.
[11] Intel Corporation. 2017. Intel Stratix 10 logic array blocks and adaptive logic modules user guide (UG-S10LAB).
[12] P. Jamieson and J. Rose. 2006. Enhancing the area-efficiency of FPGAs with hard circuits using shadow clusters. In *FPT*. 1–8.
[13] A. Krizhevsky et al. 2012. ImageNet classification with deep convolutional neural networks. In *NIPS*. 1097–1105.
[14] I. Kuon and J. Rose. 2011. Exploring area and delay tradeoffs in FPGAs with architecture and automated transistor design. *TVLSI* 19, 1, 71–84.
[15] M. Langhammer and B. Pasca. 2015. Floating-point DSP block architecture for FPGAs. In *ISFPGA*. ACM, 117–125.
[16] D. Lewis et al. 2005. The Stratix II logic and routing architecture. In *ISFPGA*. 14–20.
[17] D. Lewis et al. 2016. The Stratix 10 highly pipelined FPGA architecture. In *ISFPGA*. ACM, 159–168.
[18] J. Luu et al. 2014. On hard adders and carry chains in FPGAs. In *FCCM*. 52–59.
[19] A. Mishra et al. 2017. WRPN: wide reduced-precision networks. *arXiv preprint arXiv:1709.01134*.
[20] J. Rose et al. 1993. Architecture of field-programmable gate arrays. *Proc. IEEE* 81, 7, 1013–1029.
[21] V. Rybalkin et al. 2018. FINN-L: Library extensions and design trade-off analysis for variable precision LSTM networks on FPGAs. In *FPL*. 1–8.
[22] V. Sze et al. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12, 2295–2329.
[23] S. M. Trimberger. 2015. Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology. *Proc. IEEE* 103, 3, 318–331.
[24] Xilinx Inc. 2017. Virtex UltraScale+ HBM FPGA: A revolutionary increase in memory performance.
[25] S. Yazdanshenas and V. Betz. 2017. Automatic circuit design and modelling for heterogeneous FPGAs. In *ICFPT*. 9–16.