FPGA Logic Block Architectures for Efficient Deep Learning Inference

MOHAMED ELDAFRAWY, ANDREW BOUTROS, SADEGH YAZDANSHENAS, and VAUGHN BETZ, University of Toronto, Canada

Reducing the precision of deep neural network (DNN) inference accelerators can yield large efficiency gains with little or no accuracy degradation compared to half or single precision floating-point by enabling more multiplication operations per unit area. A wide range of precisions fall on the pareto-optimal curve of hardware efficiency vs. accuracy with no single precision dominating, making the variable precision capabilities of FPGAs very valuable. We propose three types of logic block architectural enhancements and fully evaluate a total of six architectures that improve the area efficiency of multiplications and additions implemented in the soft fabric. Increasing the LUT fracturability and adding two adders to the ALM (4-bit Adder Double Chain architecture) leads to a 1.5× area reduction for arithmetic heavy machine learning (ML) kernels, while increasing their speed. In addition, this architecture also reduces the logic area of general applications by 6%, while increasing the critical path delay by only 1%. However, our highest impact option, which adds a 9-bit shadow multiplier to the logic clusters, reduces the area and critical path delay of ML kernels by 2.4× and 1.2×, respectively. These large gains come at a cost of 15% logic area increase for general applications.

CCS Concepts: • **Hardware → Reconfigurable logic and FPGAs**; *Hardware accelerators*; *Reconfigurable logic applications*;

Additional Key Words and Phrases: Deep neural networks, FPGA, CAD tools

ACM Reference format:
Mohamed Eldafrawy, Andrew Boutros, Sadegh Yazdanshenas, and Vaughn Betz. 2020. FPGA Logic Block Architectures for Efficient Deep Learning Inference. *ACM Trans. Reconfigurable Technol. Syst.* 13, 3, Article 12 (June 2020), 34 pages.
https://doi.org/10.1145/3393668

1 INTRODUCTION

Deep neural networks (DNNs) now solve complex problems such as image classification, speech recognition, and natural language processing with much higher accuracies than prior algorithms [21]. A major driver of this recent breakthrough in DNN performance has been the significant improvements in the computational capability of the available platforms, making it possible to train large networks on huge datasets in a tolerable time. However, the continuing need to achieve or

The authors thank David Lewis and Tim Vanderhoek for helpful discussions, as well as Huawei, Intel, NSERC, CFI, and the Vector Institute for funding support.
Authors' addresses: M. Eldafrawy, A. Boutros, S. Yazdanshenas, and V. Betz, Department of Electrical and Computer Engineering, University of Toronto, 10 King's College Road, Toronto, Ontario, Canada M5S 3G4; emails: {mohamed.eldafrawy, andrew.boutros, sadegh.yazdanshenas}@mail.utoronto.ca, vaughn@ece.utoronto.ca.

1936-7406/2020/06-ART12 $15.00
https://doi.org/10.1145/3393668

even surpass human-level accuracy on deep learning (DL) tasks strongly motivates further network improvements. State-of-the-art DNN inference accuracy is largely achieved by increasing the computational complexity of DNNs, both by increasing the depth (number of layers) and the breadth (layer sizes) of the network. This commonly requires billions of operations to be carried out for each task [18, 35]. While these accuracy improvements are very valuable, increasing computational complexity also increases the power consumption and cost of the devices implementing the network. In datacenters, this raises the cost of deploying more advanced DNNs, and in embedded or edge devices it reduces battery life, increases cost, and can even preclude deployment of more complex DNNs if their power and cost requirements are excessive [40].

Network compression using precision reduction of the network weights and/or activations is one way of decreasing the computational complexity of DNNs [17]. Low-precision means fewer bits per computation, which reduces not only the area footprint of computational units but also on-chip storage requirements and memory transfers. However, precision reduction of weights and/or activations can degrade network accuracy compared to that achieved with a half- or single-precision floating-point representation.

Recent work shows that network re-engineering can partially or even fully recover this accuracy loss. Mishra et al. [30] showed that widening the network by a factor of 2× to 3× allows a wide range of fixed-point precisions to occur on the pareto-optimal curve of convolutional neural network (CNN) inference accuracy vs. hardware cost, with 2- to 8-bit precisions all having a role to play. Doubling the width of AlexNet layers allows a network with 2-bit wide weights and 4-bit wide activations to reach the same Top-1 accuracy as the full-precision network with only 39% of the compute cost. Rybalkin et al. [37] showed similar promise for low-precision fixed-point long short-term memory (LSTM) inference, with pareto-optimal points for high accuracy networks ranging from 3 to 8 bits of precision. Microsoft's Project Brainwave [16] achieves very high performance on LSTMs and gated recurrent units (GRUs) using non-standard block floating-point representations with mantissa precisions as low as 2 to 5 bits (implying 3- to 6-bit multiplies) mapped to soft logic multipliers and adders.

The core computations in DNNs are multiplication and addition, and therefore being able to perform these operations efficiently will directly lead to a more efficient network implementation. However, as described above there is a wide diversity of numeric precisions used in the most efficient DNNs. This makes field-programmable gate arrays (FPGAs) a very promising candidate for efficient DNN inference. Unlike CPUs and GPUs, FPGAs have custom bit-width data paths, allowing them to naturally adapt to precision variation while achieving high degree of parallel computations. In addition, FPGAs are re-programmable and have low non-recurring engineering cost compared to ASICs; therefore, they can adapt to the rapid evolution in the state-of-the-art DNN algorithms. FPGAs are also generally more energy-efficient than state-of-the-art CPUs and GPUs, which is important for both datacenter and edge DNN implementations [32–34]. Finally, FPGAs can implement custom sensor interfaces and logic in the same device as a DNN, making them attractive in edge devices that often have diverse sensor and control needs.

While current FPGAs are a good fit for DNN inference, a key question is whether their architecture can adapt to make them still more capable in this domain. FPGAs have a history of evolving to become more efficient for important new application areas, thereby expanding their capabilities and applicability. The high-performance filtering required by the wireless market drove the inclusion and enhancement of DSP blocks in FPGAs, which then in turn led to increased FPGA adoption in wireless infrastructure [13, 28]. The very high data transfer rate requirements of wireline switches and network routers led to the incorporation of ever-faster and more capable serial transceivers in FPGAs, which again expanded FPGA use in these industries [12, 14]. Fast off-chip and in-package memory interfaces, and larger on-chip block RAMs have both helped networking

and memory-intensive datacenter applications, such as memcache-D [15, 39]. Given the myriad current uses of DNNs and the rapid pace of new application development, we believe DNN inference will be a key part of many datacenter and embedded systems in the future. Therefore, efficient DNN inference should be a major driver of FPGA architecture enhancements.

The search for more efficient DNN inference motivated the work in Reference [6], which introduced an enhanced FPGA DSP block that natively supports low-precision 9-bit and 4-bit multiplication and multiply-accumulate (MAC) operations at minimal block area overhead. However, DSPs represent at most 5% of the FPGA area [24], which significantly limits the overall gains of these enhancements. However, logic blocks are the most ubiquitous resource in the FPGA, typically constituting about two thirds of its core area [9]. Hence, exploiting the logic blocks efficiently can have more impact on the overall performance of an application compared to DSP block enhancements.

This article is a continuation of the work introduced in Reference [8], where we discuss logic block architectural changes that can increase the density of low-precision 4- to 9-bit fixed-point MAC operations implemented in the soft fabric at minimal area and delay cost. In this work, we extend the previous study by integrating the logic block enhancements into complete FPGA architectures and evaluating their utility using the VTR (Verilog-to-routing) [31] CAD flow to gather area and delay results for both arithmetic-intensive and general benchmarks. Our contributions include:

- We propose three logic block architectural changes and a total of six architectures that can reduce the area footprint of multiplications/additions implemented using soft logic.
- We extend the Circuit Optimization For FPGA Exploration (COFFE) automated transistor sizing tool [41] to support more sophisticated logic block architectures similarly to those in Intel Stratix 10 FPGAs, as well as our proposed enhancements.
- We extend the Verilog-to-Routing (VTR) CAD flow to support the more complex carry chain structures present in our proposed architectures and to produce area-efficient packing for arithmetic operations.
- We evaluate the proposed architectures in terms of post-routing area and speed using multiplication micro-kernels, as well as arithmetic heavy and general benchmark circuits.

2 BACKGROUND

2.1 The Evolution of the FPGA Soft Logic

2.1.1 Basic Logic Element (BLE). While various basic building blocks have been used for FPGA logic [36], current commercial FPGAs all use various forms of look-up tables (LUTs) as the basis of their BLEs [38]. 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 four to six 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 [26] introduced *fracturable LUTs*; these LUTs add a small amount of circuitry so that they can not only implement a single K-input function but also can alternately implement two $K-1$-input functions by dividing the K-LUTs' truth table and output selection multiplexer (Figure 2 shows a six-input fracturable LUT). 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 employ fracturable 6-LUTs as their BLEs but make different design choices on how flexible the fracturing is—Intel calls the resulting structure an adaptive logic module (ALM). The Stratix 10 ALM [27] has a 6-LUT that can be fractured to

implement any two five-input functions that use no more than eight distinct inputs, while the Virtex Ultrascale+ 6-LUT can be fractured to implement two five-input functions with no more than five distinct inputs [10]. In addition, both architectures have dedicated structures to implement additions as well as registers for implementing sequential logic.

2.1.2 Logic Block (LB). Modern FPGAs from all major vendors use cluster-based logic blocks where N BLEs are grouped together into a cluster with a specific number of inputs I and an input crossbar connecting cluster inputs to BLE inputs. Logic clusters not only increase the area-efficiency of the FPGA but also reduce the complexity of the placement process, since the number of blocks to be placed is reduced by a factor of N [4]. Ahmed et al. showed that cluster sizes between 3 and 10 appear to give the best area-delay product among all cluster sizes from 1 to 10 [2]. In addition, Betz et al. [5] found that for 98% logic utilization, the total number of distinct cluster inputs needed is around $2N + 2$ for a 4-input BLE. Ahmed et al. [2] then generalized this equation to $I = K/2 \times (N + 1)$ for an arbitrary number of BLE inputs K, which is considerably lower than the maximum (there are $K \times N$ inputs) reducing routing multiplexer sizes; therefore, reducing the FPGA area.

Lemieux et al. [25] showed that FPGA area can be reduced further by 10–18% without affecting the delay by depopulating the LB input crossbar by 50%—each crossbar output is reachable from only 50% of the crossbar inputs. Since depopulating the crossbar stresses routing, Lemieux et al. showed that adding up to 15 spare inputs to the cluster can decrease the minimum channel width of the architecture by around 10%, improving routability. The Stratix 10 architecture has a logic array block (LAB) of size 10 with 60 inputs and a 50% depopulated crossbar [27]. The Virtex Ultrascale+ architecture has eight BLEs per configurable logic block (CLB) with 64 inputs all directly connected to the switch matrix, which provides access to both the CLB and local routing [10].

2.1.3 Hard Arithmetic Logic. Addition operations are widely used in hardware applications. Therefore, modern FPGA architectures have dedicated circuitry to enable fast and compact addition. The Stratix 10 ALM contains two bits of arithmetic (20 bits/LB), while the Ultrascale+ architecture has only one bit of arithmetic per BLE (8 bits/LB). Adders within the cluster are chained together by dedicated routing to build multi-bit additions. Adder chains also have an inter-cluster dedicated routing connection that bypasses the general routing, allowing faster long additions.

To implement the start of an addition, a multiplexer is included before the Cin pin of the adder as shown in Figure 1(a). This multiplexer chooses between 0 (first bit of addition), 1 (first bit of subtraction) and the Cout pin of the preceding adder for chaining multiple LBs together to implement long additions. Therefore, the number of these multiplexers within the LB defines that maximum number of independent additions that can be implemented in one LB. Ideally an architecture would support flexible addition starting points within the same logic block to enable dense low-precision arithmetic. However, adding multiplexers on the Cin-to-Cout logic path is undesirable, since this delay occurs N times on the critical path delay of an N-bit addition. Therefore, both Stratix 10 and Ultrascale+ have only two of those multiplexers, at the start and the middle of their LBs.

Another way of starting an addition is to precede the first bit of addition with a dummy adder primitive; both inputs connected to 0. This will force the Cout pin of this dummy adder to be 0 and therefore starting a new addition in the following adder. The same technique can be used for subtraction by connecting the dummy adder inputs to 1 instead. Although this synthesis technique consumes one more adder primitive per addition, it removes the limitation on the number of additions that can start in a LAB, thereby increasing packing flexibility. Throughout this article, we use this synthesis technique to achieve high area efficiency per LAB when implementing additions.

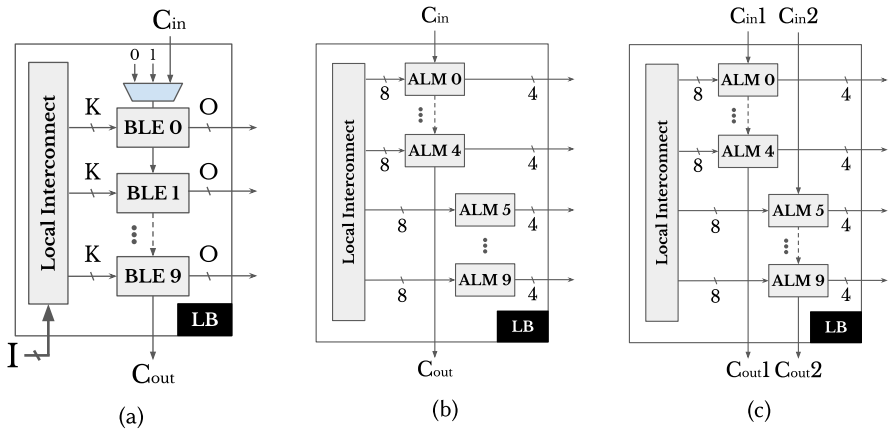

Fig. 1. Logic Clusters with various carry chain structures: (a) A single carry chain with addition starting multiplexer before the first BLE, (b) a single carry chain spanning only half the ALMs, and (c) two independent carry chains each connected to half the ALMs.

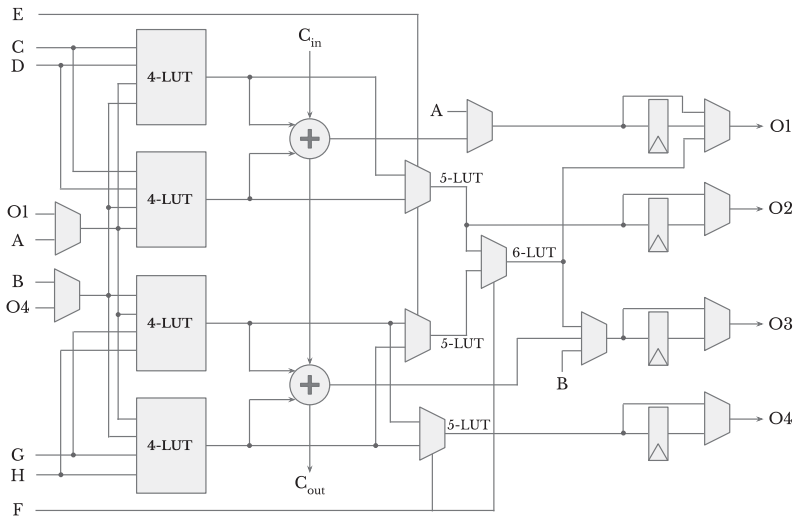

Fig. 2. ALM architecture of Intel Stratix-10-like FPGA.

2.2 Stratix 10 ALM Architecture

In later portions of this article, we will show that a key factor in the efficiency of a logic block's implementation of multiplication is how many arithmetic functions it can perform per LUT/BLE. This leads us to choose the Stratix 10 architecture, which has 2 bits of hardened arithmetic per ALM, as the baseline against which we will compare. Figure 2 shows the ALM architecture in Intel Stratix 10 FPGAs [19]. Each ALM is a 6-LUT that can be fractured into two 5-LUTs. It has 2 bits of arithmetic (i.e., two adders) with dedicated routing wires for the carries, eight distinct inputs (A-H), and four optionally registered outputs (O1-O4). An ALM can operate in two major modes as follows:

(1) The normal mode in which the ALM can implement either a six-input logic function, or two five-input or smaller functions that together use no more than eight distinct inputs.

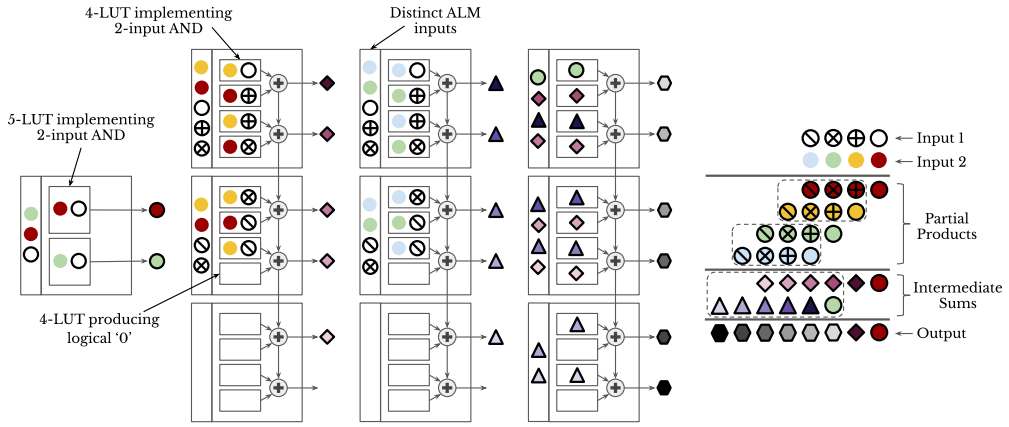

Fig. 3. The mapping of a 4-bit unsigned multiplication to the current Stratix 10 ALM architecture.

This allows implementing 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 adders. In this mode, the 4-LUTs can implement identity functions to simply pass inputs to the adders or simple pre-addition logic if all four 4-LUTs use six or fewer distinct inputs.

Stratix 10 LABs contain 10 ALMs along with a local routing crossbar that allows connections from the general (inter-logic-block) routing wires to the ALM inputs. 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.

2.3 Low-precision Multiplications on FPGAs

To understand how multiplications are mapped to the Stratix 10 ALM architecture, we experimented with different multiplication sizes ranging from 4- to 9-bit on Quartus Prime Pro 17.1. For simplicity, we will explain the mapping of an unsigned 4-bit multiplication but we also found in our experiments that a signed or unsigned multiplication of any size from 4 to 9 bits is mapped using the same approach. In Figure 3, 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 (addition) 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 multiplication, three and a half ALMs are used to generate and reduce two rows of partial products in the first reduction stage as shown in Figure 3. Then the second stage of reduction (adding intermediate sums together) requires three additional ALMs to produce the final result. This results in a total of 10 ALMs for a 4-bit multiplication.

To validate our choice of the Stratix 10 ALM as a baseline, we also performed similar experiments in mapping 4- to 9-bit MAC 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 to ensure we had found the densest mapping possible in each device.

Table 1. Number of ALMs Needed to Implement 4- to 9-bit Multiplications and MACs on Stratix 10 Using the Default Synthesis Settings and with the Fractal Synthesis Option Enabled in Quartus Prime Pro 19.1

Multiplication Size	Default	Fractal Synthesis	MAC Size	Default	Fractal Synthesis
4-bit × 4-bit	10	8	4-bit × 4-bit + 8-bit	15	13
5-bit × 5-bit	17	13	5-bit × 5-bit + 10-bit	23	19
6-bit × 6-bit	23	21	6-bit × 6-bit + 12-bit	30	28
7-bit × 7-bit	31	28	7-bit × 7-bit + 14-bit	39	36
8-bit × 8-bit	39	35	8-bit × 8-bit + 16-bit	48	44
9-bit × 9-bit	51	47	9-bit × 9-bit + 18-bit	61	57
Geomean	**25.0**	**21.6**	**Geomean**	**32.5**	**29.2**

Intel recently developed fractal synthesis [23], which uses multiplication regularization to balance the LUT and adder logic for each multiplication size, resulting in efficient ALM packing. The multiplication regularization is done by extracting some of the addition logic into auxiliary cells that are implemented using LUTs. Using this technique one of the additions in the multiplication logic could be completely remove and therefore reduce the total number of ALMs needed to implement the multiplication. Table 1 shows the number of ALMs needed to implement 4- to 9-bit multiplications and MACs on Stratix 10 for two cases: with the default synthesis settings and with fractal synthesis enabled (both in Quartus 19.1). We found that using fractal synthesis reduces the geometric mean ALM count by 14% for multiplies and 10% for MACs and had no impact on LAB count. Although fractal synthesis clearly reduces ALM count, it is beyond the scope of this article to adapt its complex algorithms to each architecture we investigate. Therefore, we use traditional multiplication synthesis instead so that the optimization quality is consistent across architecture variants. Since this technique relies on extracting auxiliary cells from the multiplication logic and implementing it using LUTs, we believe that the 4-bit Adder architectures and the Extra Carry Chain architectures could definitely benefit from this synthesis technique to a large extent. However, the Shadow Multiplier architectures will not benefit from it, since the partial product generation and addition is mostly done inside the Shadow Multipliers as we will show in Section 3.4. However, as we will show in the result section of this article, the area gains of the Shadow Multiplier architectures are much higher than what Fractal Synthesis can offer; therefore, we do not expect fractal synthesis to significantly impact any of our architecture conclusions in this article.

2.4 COFFE: 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 computer-aided design (CAD) tool for this purpose [11]. 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 (v2.0), 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 [42]. 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 Multiplier in Section 3.4). The latest release of COFFE can model fracturable LUTs and either one or two bits of arithmetic per fracturable LUT. However, it still requires considerable modifications to enable modeling more sophisticated ALM architectures similarly to those of state-of-the-art commercial FPGAs as shown in Figure 2. Our enhancements to COFFE will be discussed later in Section 4.2.

2.5 VTR: Application-level Speed and Area Evaluation

After evaluating the architectures from circuit-level area and delay perspective, another level of evaluation is also required to quantify the impact of the introduced architectural changes on the speed and area of FPGA applications. For this purpose, we use the Verilog-to-Routing (VTR) flow [31]. VTR takes a digital design described in Verilog, implements it on an FPGA architecture defined by the user and produces post-routing area and speed results. VTR enables exploration of a wide range of FPGA architecture ideas by taking an architecture description file in a human-readable xml format. We use the tile-level area and delay values produced by COFFE to create an appropriate VTR architecture file for each of our proposed logic block enhancements.

There are three underlying software tools in VTR responsible for different stages in the flow: (1) ODIN II for elaboration and synthesis, (2) ABC for logic reduction and technology mapping, and (3) Versatile Placement and Routing (VPR) for packing, placement, routing, and timing analysis. To evaluate the proposed architectures using VTR, we made some modifications to ODIN II and VPR; some of these modifications were necessary to support the new architectures, while others were needed to ensure high-quality optimization for all architectures. Our modifications are applied to the latest release of VTR (v8.0) [31] and they will be discussed in Section 5.2.

3 LOGIC ARCHITECTURE ENHANCEMENTS

3.1 Room for Improvement and Design Constraints

In Section 2.3, we showed how a 4-bit unsigned multiplication is implemented on a Stratix 10 FPGA. By analysing this implementation, it is clear that the available resources are underutilized when implementing stand-alone additions and multiplications. First, when generating partial products, the 4-input LUTs feeding the adders are used to implement 2-input AND gates, thereby making use of only 50% of the 4-LUT capabilities. Second, when implementing stand-alone additions (the third column of ALMs in Figure 3), the LUTs are used only as buffers to pass the ALM inputs to the adder inputs. Finally, we use at most five of the eight ALM inputs and two of the four ALM outputs (top ALM in the first and second columns in Figure 3) to implement multiplication, which indicates that we are underutilizing the ALM's routing interfaces as well.

These observations motivate the search for architectural changes that allow better utilization of the ALM resources. Since these changes are mainly targeting addition and multiplication operations, we add some design constraints to ensure they do not reduce the flexibility of the FPGA or degrade performance for applications that do not heavily depend on addition or multiplication operations. Therefore, when designing the proposed architectures, we ensure that they have:

(1) Backward compatibility: support all the operations that the baseline architecture supports,
(2) Low added cost: introduce minimal area and delay penalty for existing features to maintain the performance of existing applications, and
(3) Same routing interfaces: maintain the same number of LAB and ALM input/output pins and the same routing architecture to make it easier to measure the actual influence of our changes and ensure routability is not degraded.

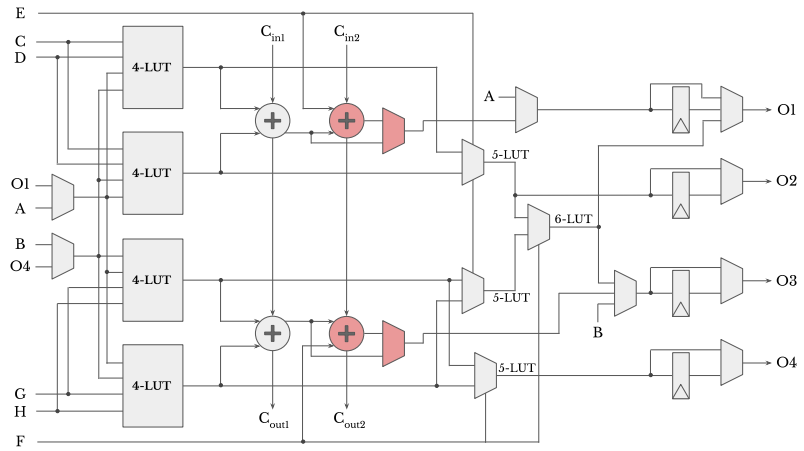

Fig. 4. ALM architecture of Stratix 10 with the proposed Extra Carry Chain architecture modifications (highlighted in red).

In the remainder of this section, we discuss three different architectural enhancements to FPGA logic blocks, two of which are on the ALM level while the third is on the LAB level. For each architectural enhancement, we describe the circuitry added to the ALM or the LAB to overcome one or more of the resource under-utilization issues discussed previously. As a simple example, we also show how a 4-bit unsigned multiplication would map to the soft logic of each proposed FPGA architecture and compare it to the baseline mapping discussed in Section 2.3.

3.2 Efficient Reduction: Extra Carry Chain

When mapping multiplies to ALMs, after adding the partial products together (the first addition stage in Figure 3), only the adder chains are used to implement the subsequent reduction stages. In this case, the LUTs are used as identity functions to feed the hard adders with inputs as shown in the rightmost ALM column in Figure 3. This results in inefficient ALM utilization and motivates more efficient adder reduction trees to increase the on-chip multiplication density.

To address this, we propose the ALM architecture shown in Figure 4. The ALM is modified by adding a second level of carry chain such that the two new adders (highlighted in red) receive their inputs from the two sum outputs of the first carry chain and the two ALM inputs (E and F) that are left unused when the ALM is in arithmetic mode. With this architecture change, 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. Figure 5 shows the mapping of a 4-bit unsigned multiplication to the proposed Extra Carry Chain ALM architecture. 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. Absorbing the final stage of addition into the second column of ALMs leads to a mapping that consumes only 7 ALMs instead of the 10 ALMs required in the Stratix 10 ALM architecture.

The additional carry chain decreases the number of ALM levels not only for multiplies 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. This architectural enhancement follows the constraints mentioned in Section 3.1 and does not require any modifications to the ALM input/output interface. When implementing

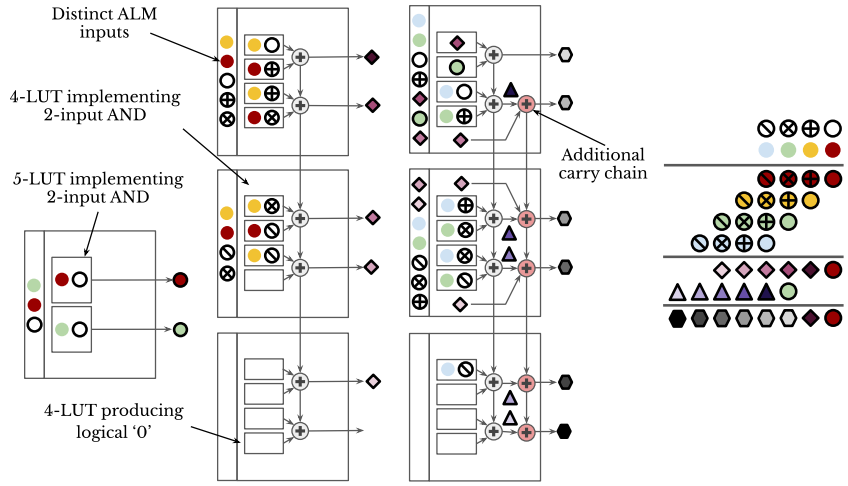

Fig. 5. Mapping a 4-bit unsigned multiplication to the Extra Carry Chain architecture.

multiplications, this architecture uses at most seven of eight ALM inputs and two of four ALM outputs (top ALM in the second column of ALMs in Figure 5). On the LAB level, this generalizes to $2N + 3$ distinct inputs for an N-bit long multiplication. Since the widest adder that can fit in one LAB is 20-bit-wide, the maximum number of distinct inputs needed per LAB is 43 inputs for large multiplies (20 bits or larger). Similarly, for an N-bit-wide 3-to-1 adder reduction, $3N$ distinct inputs are needed, requiring at most 60 distinct inputs per LAB. Therefore, whether long multiplies or long 3-to-1 adder reductions are implemented, no more than 60 distinct inputs per LAB are needed. In addition, the area overhead is minimal, as we have added only two adders and two bypassing 2:1 multiplexers to the ALM, as highlighted in red in Figure 4.

3.3 Deeper Fracturability: 4-bit Adder

Another approach to increase the density of multiplication operations is to harden more adders per ALM to create a single-level but wider carry chain. To avoid under-utilizing 4-LUTs to implement 2-input AND gates (partial products), 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 Figure 6. 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, requiring a total of 16 inputs. However, since there are many shared signals, the number of distinct inputs is only 7 (see Figure 14(b)), which is less than the number of ALM inputs. The reason for that is the input sharing nature of multiplier arrays that produce $M \times N$ partial product bits from only $M + N$ input bits for any M-bit × N-bit multiplication. Figure 7 shows the mapping of a 4-bit unsigned multiplication to ALMs with 4 bits of arithmetic; none of the used ALMs need more than eight distinct inputs. This property that no more than eight inputs are required per ALM holds for signed and unsigned multiplications of all sizes.

Although we do not need additional ALM input pins, we must ensure that we can deliver the correct inputs to the 3-LUTs not only to implement multiplier arrays but also to use the adders to implement a stand-alone 4-bit addition per ALM. For this reason, we add the small 2:1 multiplexers (highlighted in red in Figure 6) in front of four out of the eight 3-LUTs. These multiplexers provide us with enough flexibility to deliver the eight ALM inputs to the 3-LUTs to implement Equations (1)

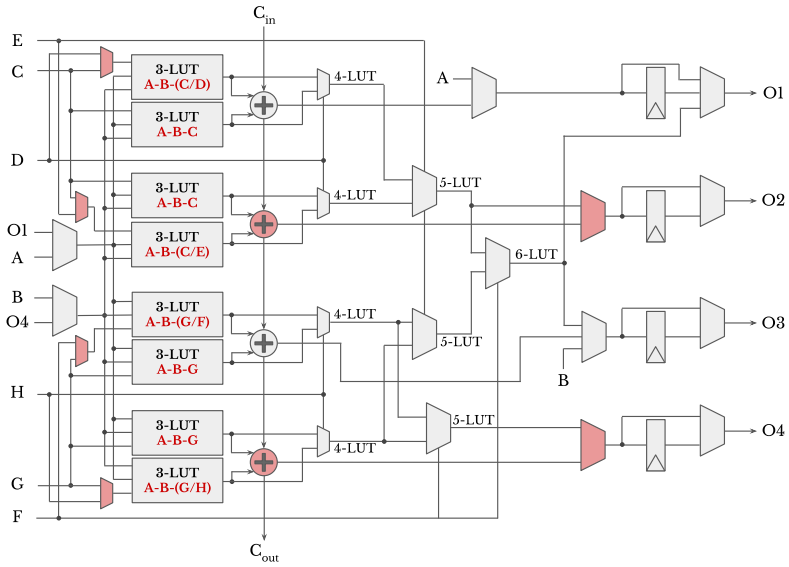

Fig. 6. Proposed 4-bit Adder architecture with additional adders and multiplexing (highlighted in red).

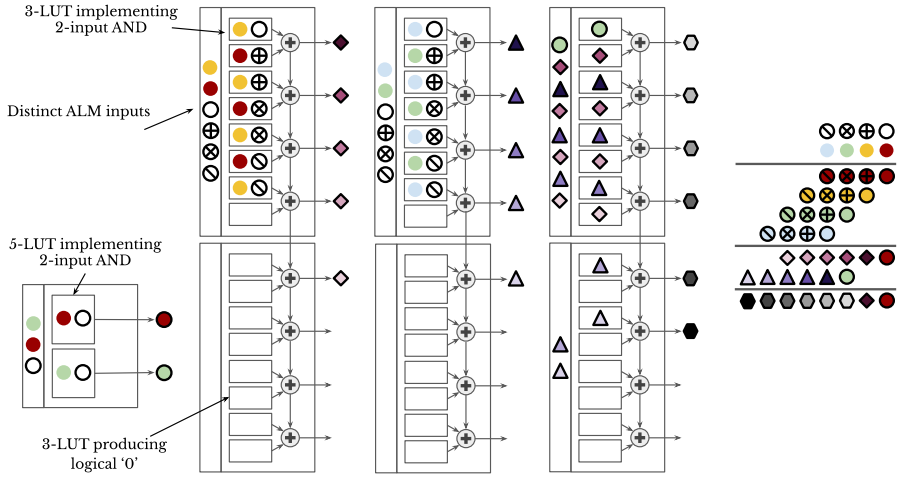

Fig. 7. Mapping a 4-bit unsigned multiplication to the 4-bit Adder architecture.

and (2) in the case of multiplications and stand-alone additions, respectively.

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

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

Table 2 shows the assignment of ALM inputs (A-H in Figure 6) as well as the function that each of the eight 3-LUTs implements for both scenarios. To be able to output the result of all four adders,

Table 2. ALM Input Assignment and LUT Masks for Implementing Multiplications and Stand-alone Additions (Equations (1) and (2), Receptively) Using the 4-bit Adder Architecture

Input	L1	L2	L3	L4	L5	L6	L7	L8
Mult (Equation (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 (Equation (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)

we also add two 2:1 output select multiplexers before the ALM registers for outputs O2 and O4 as shown in Figure 6. In total, this architecture adds six 2:1 multiplexers and two adders to the ALM.

This architecture uses all the ALM inputs and outputs when implementing pure additions as shown in the top rightmost ALM in Figure 7. Although this is not a problem on the ALM level, at the LAB level we will use more distinct input signals than the routing interfaces can bring in when implementing long additions. The Stratix 10 LAB has 60 distinct inputs; therefore only a 30-bit-wide addition that uses 7.5 ALMs (instead of a 40-bit addition spanning all ten ALMs) would fit in the LAB. Increasing the number of LAB routing interfaces (inputs) is expensive [4]; hence, as mentioned in Section 3.1, we have a constraint that restricts us from adding more LAB inputs.

Instead, we propose two variants of the 4-bit Adder Architecture to solve this problem: a single-chain architecture and a double-chain architecture. Figure 1(b) shows the 4-bit Adder single-chain architecture where there is a single adder chain that spans only half of the ALMs in the LAB (i.e., 5 ALMs). The ALMs in the other half are similar to a Stratix 10 ALM (see Figure 2), but have no hard adders. The LAB of this architecture can implement a total of 20 bits of addition, requiring a maximum of 40 distinct inputs to fully utilize the adders, which is the same as Stratix 10. However, a 20-bit-wide addition will be implemented in only half a LAB, leaving the other half for non-arithmetic logic (LUTs and registers), facilitating more efficient LAB resource utilization.

Figure 1(c) shows the second variant of the 4-bit Adder Architecture, which is the 4-bit Adder double-chain architecture. Unlike the single-chain variant, all the ALMs of this architecture contain four adders. However, the LAB contains two adder chains that are completely independent from one another, with separate LAB-level Cin and Cout pins for each. Each adder chain can implement 20 bits of addition per LAB. Although we still cannot use all the adders of the LAB with completely distinct inputs, since the 60-input limitation still exists, fully utilizing one of the two chains with distinct inputs is possible without exceeding the LAB limit of 60. The double-chain architecture still has an advantage over the single-chain architecture as it can implement up-to 40 bits of addition per LAB when implementing additions with low input demand. For instance, an N-bit incrementer/decrementer requires only $N + 1$ distinct inputs, and the addition of two N-bit partial products in a multiplication only requires $N + 3$ independent inputs (in Equation (1), adding two 4-bit partial products require seven distinct inputs a_{0-1} and b_{0-4}). Therefore, we expect this architecture to achieve higher LAB packing density when implementing designs containing a mix of adder usages with different input demands.

3.4 Re-Purposing Routing Interfaces: Shadow Multipliers

In Reference [20], Jamieson and Rose proposed *shadow clusters*. These are FPGA logic clusters added to hard blocks such as BRAMs or DSP blocks, which use their routing interfaces; hence, they can be used only when the hard blocks are not used. The goal was to increase area efficiency for applications that do not fully utilize the hard blocks on an FPGA. However, for DL applications, DSP blocks are valuable and are usually the bottleneck when implementing DL accelerators on current FPGAs [7]. Therefore, a more effective way to increase the density of on-chip multiplication operations is to add a *shadow* hard multiplier to each logic cluster. Since adding any extra input/output pins to the LAB would add area and violate the constraint of not changing the

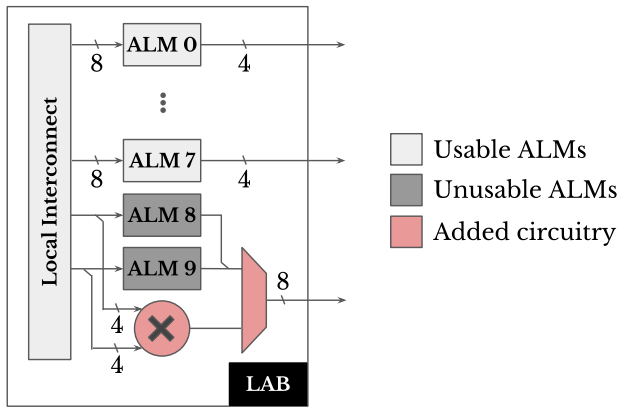

Fig. 8. Proposed 4-bit Shadow Multiplier architecture implementing a 4-bit unsigned multiplication. Highlighted in red is the 4-bit multiplier added to the LAB and the multiplexers choosing between the output of the multiplier and the shadowed ALMs.

architecture routing interfaces (discussed in Section 3.1), this shadow multiplier borrows its input and output pins from some ALMs in the cluster. Hence, the intra-cluster interconnect is not affected and no cluster pins are added. Figure 8 shows how the multiplier *shadows* some ALMs by making them unusable when the multiplier is in use and vice versa.

To implement registered multiplications efficiently using the Shadow Multiplier architecture without adding more area, we change the position of the output selection multiplexers in Figure 8 to be in front of the registers within the shadowed ALMs (see Figure 2)—instead of at the ALM outputs. Therefore, the shadow multiplier outputs can be registered directly without going through the general routing saving area and reducing delay. This modification has minor impact on the delay of the shadowed ALMs, since their output is still going through the same number of multiplexers (one multiplexer position has changed); while for the shadow multiplier a delay of a 2:1 multiplexer is added, which is insignificant compared to the delay of the shadow multiplier.

An N-bit × N-bit multiplier has $2N$ input pins and $2N$ output pins, while each ALM has eight input pins and only four output pins. Therefore, the number of ALMs that are shadowed by the hard multiplier is defined by the multiplier output bit-width and $\frac{N}{2}$ ALMs are unusable when the shadow multiplier is in use. The area overhead of adding a hard multiplier to the LAB consists of the area of the multiplier and the $2N$ 2:1 multiplexers added in front of the ALM registers. The area overhead of this architecture is larger than the other proposed architectures where we only added two adders and few 2:1 multiplexers. Further, the shadow multiplier area also depends on the size of the multiplier added to the cluster, with wider multipliers having larger area footprint. However, a large shadow multiplier results in a more efficient implementation of large multiplies in the soft logic. Therefore, the choice of shadow multiplier size is heavily influence by the multiplication bit-widths used by the target applications. We experiment with shadow multiplier sizes of 4, 6, and 9 bits and show the tradeoff between ALM savings and cluster area overhead in Section 4.

FPGA applications have diverse needs for multiplication: There will be a variety of multiplication precisions and a mix of signed and unsigned multiplications. To ensure our shadow multiplier is as flexible as possible, we propose a hard multiplier design shown in Figure 9(b) that modifies the conventional multiplier array shown in Figure 9(a). The modified multiplier enables efficient implementation of two's complement signed multiplications that are larger than the shadow multiplier size. The multiplier array consists of normal cells (see Figure 9(c)), invertible cells (see Figure 9(d)) and sign bits labelled with colored "S" symbols. The sign bits are added to implement

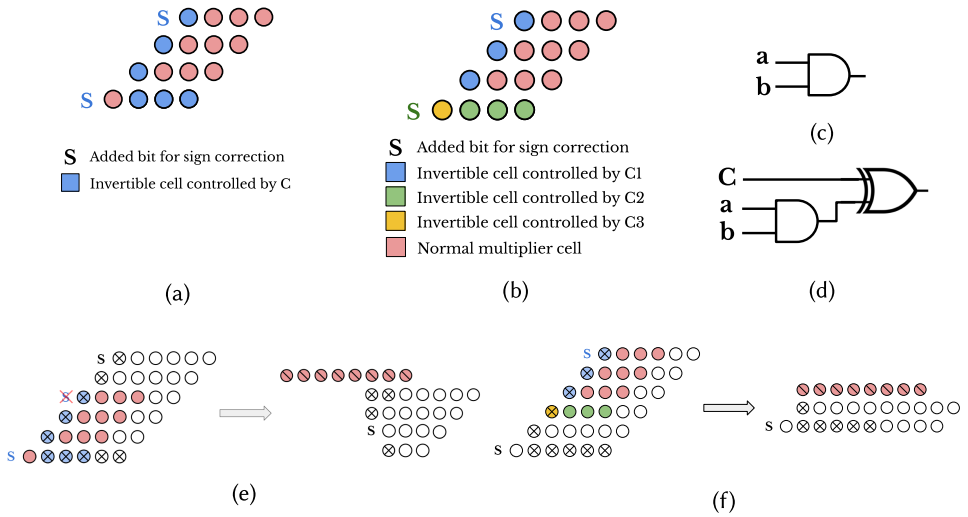

Fig. 9. Shadow multiplier: (a) Conventional hard multiplier design, (b) Modified hard multiplier design, (c) Normal multiplier cell, (d) Invertible multiplier cell, and 6-bit multiplication mapped to 4-bit hard multiplier (e) using conventional multiplier array and (f) using enhanced multiplier array with three distinct sign control signals. Cells with a cross sign are inverted and cells with a diagonal line represent hard multiplier outputs.

a Baugh–Wooley signed multiplication [3] where the "S" bits are 1 for a signed multiplication and 0 otherwise. The invertible cells in the modified multiplier are controlled by three control signals ($C1, C2$, and $C3$) instead of the single control signal in conventional multipliers. In addition, the sign bits are also controlled by two of those control signals (control signal with matching color in Figure 9(b)) instead of only one in the conventional multipliers.

Figure 9(e) illustrates how a 6-bit signed multiplication 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 multiplication. This results in five partial results to be reduced in the soft logic in addition to extra logic to correct the contamination of the misplaced "S" bit (marked by a red cross in Figure 9(e)). However, as shown in Figure 9(f) when using a multiplier array with three distinct control signals, we can invert the corner cell, disable the inversion of the bottom boundary cells and set the green S in Figure 9(b) to 0. This enables the hard multiplier to implement the top left corner of the multiplier array resulting in no contamination bits and only three partial results to reduce in the soft logic; therefore reducing both area and critical path delay of the multiplication.

4 EXPECTED GAINS AND HARDWARE COST

In this section, we quantify the expected gains from mapping multiplies and MACs to the proposed architectures. Then, we evaluate the hardware cost represented in the area and delay overheads caused by the added circuitry. Finally, using these area and delay results, we fully evaluating the architectures on a wider range of applications using a full CAD flow in Sections 5 and 6.

4.1 ALM Savings for Multiplies and MACs

We hand-map multiplies and MACs of sizes ranging from 4 to 9 bits to the ALMs of the baseline architecture as well as the three proposed ones. We verify the correctness and efficiency of our

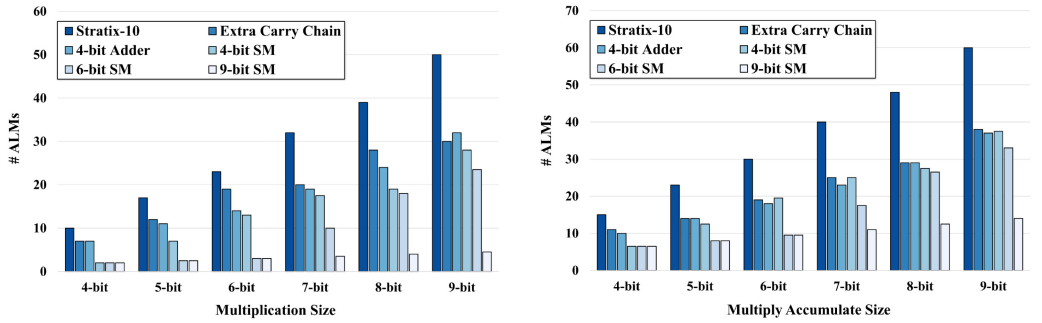

Fig. 10. Number of ALMs needed (used or made unusable) to implement 4- to 9-bit multiplies (a) and MACs (b) on the Stratix 10, Extra Carry Chain, 4-bit Adder, and 4-, 6- and 9-bit Shadow Multiplier architectures.

mapping for the baseline architecture by checking that they match the synthesis results from Quartus 17.1 targeting Stratix 10. 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 multiplication output; however, any other assumption leads to the same trend in results.

Figure 10 shows the number of ALMs required to implement 4- to 9-bit multiplies and MACs on the baseline ALM architecture, as well as our Extra Carry Chain and 4-bit Adder ALMs. The two variants of the 4-bit Adder architecture, discussed in Section 3.3, have the exact same ALM architecture for arithmetic ALMs and their architectures differ only on the LAB level. Therefore, in this section, where we are concerned only with ALM counts, we will refer to both of them as the 4-bit Adder architecture. Both the Extra Carry Chain and the 4-bit Adder architectures outperform the baseline across all multiplication and MAC precisions. For stand-alone multiplies, the Extra Carry Chain and the 4-bit Adder architectures reduce the geometric average ALM usage by 29% and 36% across this range of multiplication 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 by 36% and 38% relative to the baseline, respectively. The Extra Carry Chain architecture can implement the accumulation in the second carry chain of the ALMs implementing the last addition in the multiplier array. Therefore, its advantage over the baseline architecture is larger on MAC operations than on multiplies.

When mapping multiplies to the Shadow Multiplier architecture discussed in Section 3.4, there are often two ways of mapping an M-bit multiply to an architecture with an N-bit shadow multiplier. This happens when $M > N$, where the mapping could be either done by using one N-bit shadow multiplier followed by a series of additions, as shown in Figure 11(a), or by combining several shadow multipliers together and adding their outputs as in Figure 11(b). For all cases where $M > N$, we count the number of ALMs using the two mapping techniques and choose the more efficient mapping for each case. We found that when $M - N > 2$, it is more efficient to use the mapping with more shadow multipliers, and otherwise the mapping with one shadow multiplier is more efficient over the 4- to 9-bit multiplication range.

Figure 10 shows that Shadow Multiplier architecture reduces ALM count for multiplication and MAC operations more than the Extra Carry Chain and 4-bit Adder architectures, with the gains increasing for larger Shadow Multiplier architectures. For instance, when implementing a 9-bit multiplication using the 9-bit Shadow Multiplier architecture, only 4.5 ALMs are consumed, since

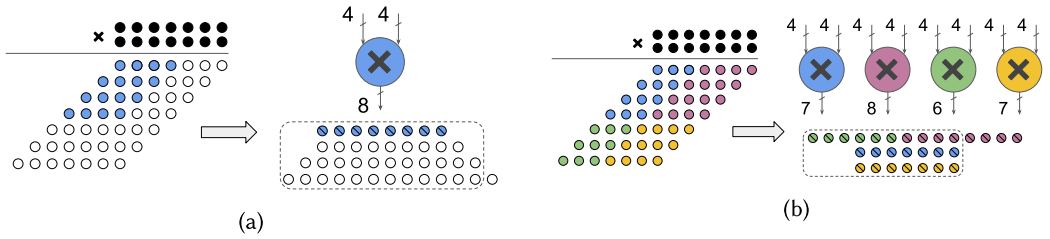

Fig. 11. Two ways of implementing a 7-bit multiplication on an architecture with a 4-bit shadow multiplier; (a) using one shadow multiplier, soft logic for some partial products and three soft logic additions and (b) using four shadow multipliers and two soft logic additions. Colored circles represent partial products generated and reduced by the matching color shadow multiplier, uncolored circles are partial products implemented using the soft logic, and colored circles with a diagonal line are shadow multiplier outputs. The circles enclosed by a dashed box are added together using soft logic to produce the final output of the multiplication.

they were made unusable by selecting the output of the multiplier (see Figure 8). On average for 4- to 9-bit multiplies, the reduction in ALMs used (in addition to ALMs made unusable) vs. the baseline varies from 56% for a 4-bit Shadow Multiplier to 87% for a 9-bit Shadow Multiplier. For MAC operations, the 4-bit Shadow Multiplier architecture yields a 43% reduction in ALMs while the 9-bit variant leads to a 70% reduction in ALMs. This architecture does not have any advantage over the baseline architecture when implementing pure additions, so it achieves slightly lower percentage ALM reductions on MACs than on stand-alone multiplication. Overall, Shadow Multiplier architectures reduce ALM usage more than the Extra Carry Chain or the 4-bit Adder architectures; however, they also add more area to the FPGA tile, which will be discussed in Section 4.3.

4.2 COFFE Flow: Extensions and Technology Node

Section 4.1 showed that the three proposed architectural changes will lead to considerable savings in the total number of ALMs needed to implement multiplies and MACs in 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; therefore, to evaluate the proposed architectures 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. Previously COFFE could only 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. COFFE previously supported fracturing a 6-LUT into two 5-LUTs; therefore, we extended COFFE to support fracturing a 6-LUT into four 4-LUTs or eight 3-LUTs. These deeper levels of fracturing are needed to model the LUTs feeding the carry chain in Stratix 10 and all the proposed architectures. 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 significantly affects both transistor sizing and delay, 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 that allow us to perform area and delay measurements for our three new architectures: the Stratix-10-like architecture, Stratix 10 with a second level of carry chain adders and finally the deeper fracturability design with 4 adders per ALM (as shown in Figure 2, 4 and 6, respectively).

Table 3. Routing and Tile Architecture Parameters

Parameter	Value	Parameter	Value
N (ALMs/LAB)	10	Fs	3
W (Channel width)	320	Fcin	0.15
L (Wire segment length)	4	Fcout	0.1
I (Inputs/LAB)	60	Fclocal	0.5

We believe these added 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 Multiplier architectures, we created a structural System Verilog implementation of our enhanced multiplier array with parameterized precision. DSP block multipliers in FPGAs are typically implemented with standard cells; 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 by $(\frac{22}{28})^2$ and input it to COFFE, along with the area of the input drivers and output select multiplexers (see Figure 8). 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. In addition, all shadow multipliers are designed to run at 1 GHz.

4.3 Tile Area and Delay Cost

To evaluate the area and speed of the logic and routing tile, that is, the core of an FPGA architecture, we need to specify more than the ALM and arithmetic structure. For all architectures evaluated, we use logic cluster parameters that match Stratix 10 and routing parameters for a length 4 routing wire architecture that prior work has shown to have good performance. These parameters are similar to those in Reference [11] and are summarized in Table 3. For each architecture, we use COFFE to size transistors and compute the resulting tile area and delays.

COFFE optimizes a user-specified cost function of area and delay. For this study we use a cost function of $area \cdot delay^2$ as it reflects the greater emphasis on delay compared to area, which is typical in high-performance FPGAs like Stratix 10. Since there can be several solutions with similar $area \cdot delay^2$ values, but different area vs. delay tradeoffs, we also guide COFFE's transistor sizing to keep the delay of the routing and other structures that have not been modified in this study nearly constant. This results in each architecture being optimized to a similar area vs. delay point, allowing easier comparison of the speed and area tradeoff for each of the evaluated architectures.

4.3.1 FPGA Tile Area. Figure 12(a) shows the area breakdown of the baseline, Extra Carry Chain, 4-bit Adder single-chain, 4-bit Adder double-chain, and 4- and 9-bit Shadow Multiplier architectures. The Extra Carry Chain and the 4-bit Adder double-chain architectures show small tile area increases over the baseline, of 2.4% and 2.9%, respectively. The major contributor to this increase is doubling the number of adders per ALM for both architectures, which leads to approximately 1.7% area increase. The remainder of the area increase is primarily due to the extra multiplexing required by these architectures. The 4-bit Adder Double Chain architecture adds six 2:1 multiplexers to its LUT compared to only two 2:1 multiplexers for the Extra Carry Chain architecture; therefore, it shows a slightly higher increase in area vs. the baseline. The 4-bit Adder single-chain architecture has the same number of adders per LAB as Stratix 10 and a simpler ALM

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

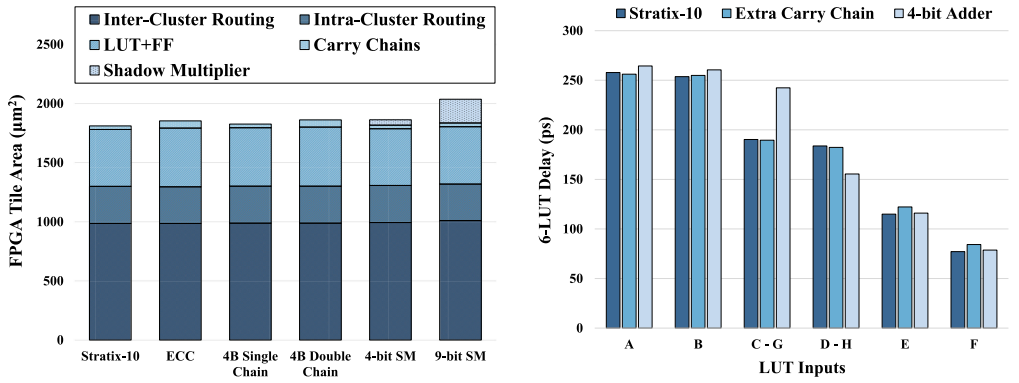

Fig. 12. (a) Area breakdown of the FPGA tile of Stratix 10, Extra Carry Chain, 4-bit Adder single-chain, 4-bit Adder double-chain, 4-bit Shadow Multiplier, and 9-bit Shadow Multiplier architectures. (b) LUT input delays of the baseline ALM as well as the modified ALMs of the Extra Carry Chain and 4-bit Adder architectures.

(no arithmetic) design for half of its ALMs; therefore it shows only 0.6% increase in tile area vs. the baseline.

As shown in Figure 12(a), the 4- and 9-bit shadow multipliers increase the FPGA tile area by 2.9% and 12.5%, respectively. The main contributor to this increase in area is the added shadow multiplier, which contributes a 2.5% and 11.1% area increase for the case of 4- and 9-bit Shadow Multipliers, respectively. The remaining 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 shadow multiplier's area. In addition, a 6-bit Shadow Multiplier architecture, which is considered an intermediate option between the 4- and 9-bit variants, shows a 6.0% increase in FPGA tile area vs. the baseline.

4.3.2 Logic and Routing Delays. While the small area increases of the Extra Carry Chain and the two variants of the 4-bit Adder architectures mean that the routing wire delays are not significantly impacted, some LUT delays are. The delay of each LUT inputs from the local interconnect of the LAB to the 6-LUT output is shown in Figure 12(b) for each ALM architecture. Note that the ALM has eight inputs where 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 almost the same delay compared to the baseline architecture, while inputs E and F have experienced a 6.3% and 9.2% increase in delay, respectively. In this architecture a connection is added from both these inputs to the second-level adders in the ALM, increasing the capacitive loading of these inputs; hence, increasing their delay. Overall the Extra Carry Chain architecture leads to very little change in the key delay paths.

The 4-bit Adder architecture has more impact on LUT delay and shows an increase in delay for all of its inputs except D and H. As Figure 6 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 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 2.6% delay increase for inputs A and B. Input C's delay increases by 27% as it is also impacted by the extra 2:1 multiplexers (see Figure 6) added to ensure all the necessary signals can reach the 3-LUTs when in multiplication or

addition modes. The buffering added after the 3-LUT output is also the reason for the reduction in delay of inputs D–H. As shown in Figure 6, those inputs are connected to the selection pin of the multiplexer choosing between the output of two 3-LUTs. This is now preceded by the LUT output buffer; therefore, the voltage at the input of this multiplexer swings from rail to rail (instead of only between 0 V and V_{dd}-V_{th} at this point in the original ALM), leading to a smaller delay though the multiplexer.

The LUT delays of the Shadow Multiplier architectures are almost the same as the baseline. However, the increase in tile area caused by the shadow multiplier also leads to some slowdown in the programmable routing due to longer wirelengths. For the 4-bit Shadow Multiplier architecture, the delay of a direct (nearest neighbor) connection and a length 4 routing wire have increased by 2.3% and 2.5%, respectively. A 9-bit Shadow Multiplier increases routing delay more: by 7.4% for a direct connection and 11.3% for a length 4 routing wire.

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 (RCP) delay [22]. We analyzed the VPR results on our Stratix-10-like architecture and found that on average, across the large VTR benchmarks, the critical path delay is divided into 73% inter-cluster routing delay, 12% intra-cluster routing delay, 12% logic delay and 3% carry chain delay. Using these ratios we can combine the COFFE delay numbers for each architecture into the RCP delay as a single delay metric. The Extra Carry Chain architecture shows an increase in the RCP delay of 1.1%, mainly due to small increases in the routing delays. The 4-bit Adder Double Chain architecture has a 2.3% increase in RCP delay caused by an increase in routing delays in addition to the increase in LUT delays discussed above. The single-chain variant of the architecture is less affected by this delay increase, showing only 1.1% increase in RCP delay. The Shadow Multiplier architectures have larger RCP delays due to increases in the inter- and intra-LAB routing delays that are proportional to the shadow multiplier sizes. This RCP delay increase varies from 1.3% for a 4-bit Shadow Multiplier architecture to 3.6% and 5% for the 6-bit and 9-bit Shadow Multiplier architectures, respectively.

5 APPLICATION-LEVEL EVALUATION

The prior section showed that all of the proposed architectures show a significant reduction in the number of ALMs needed for multiplication operations. The area and delay impact of these proposed architectures on the FPGA tile ranges from very small (1–3%) for the ALM arithmetic changes and small Shadow Multipliers to moderate (up to 13% area and 5% delay) for large Shadow Multipliers. This suggests these architectures are likely to produce a net gain on complete applications, particularly those that are multiplication intensive. In this section, we compile arithmetic kernels and entire benchmark applications through a full CAD flow for the various architectures to evaluate the net impact of each proposed change.

5.1 VTR Architecture Files

We use the VTR CAD flow to quantify the gains of the proposed architectures by implementing multiplication, MAC, and machine learning kernels, as well as complete application benchmarks, on each architecture. We wrote architecture files in the VTR format that describe the ALM and the LAB of each of our proposed architectures. The delays of all paths within the LAB and the routing architecture come from the COFFE runs discussed in Section 4.3. In addition, the architecture and routing parameters match those used to run COFFE as shown in Table 3.

The LAB has 60 input pins divided into four groups of 15 pins each, where each group can reach only 50% of the ALM inputs to implement a 50% depopulated crossbar, which matches the partial crossbar style of Intel FPGA architectures [28]. Further, each pair of ALM outputs have

muxing that makes them swappable (O1, O2 and O3, O4 in Figure 2). As mentioned in Section 2.2, each LAB in the Stratix 10 architecture has nearest neighbour connections to the LAB on its right and left as well as feedback connections to itself. Since each ALM has two pairs of swappable outputs, we choose to connect only 50% of the outputs (two non-equivalent outputs from each ALM) to each of these possible direct connections. This ensures that most of the time there are enough routing possibilities for signals within a LAB or connected to adjacent LABs to use these fast direct connects. In addition, it reduces the hardware cost by 50% vs. having each ALM output feed a connection block mux to an input pin in its own LAB and in each of the two adjacent LABs. This results in each LAB having 20 internal feedback connections, as well as 20 direct connects to the LAB to its immediate left and another 20 to the LAB to its immediate right.

Since we are also evaluating general benchmarks on those architectures, we need a full definition of the FPGA architecture; therefore, DSP blocks and BRAMs are added to the architecture files. To match Stratix 10, we use 20 Kb BRAMs and DSP blocks that can implement either a single 27×27 or two 18×19 multipliers. For both blocks we define the key modes of operation available in Stratix 10 and extract the relevant delays within those blocks using Quartus Prime Pro 17.1 targeting an Arria 10 FPGA. The DSP and BRAM architectures of Arria 10 are very similar to Stratix 10. In addition, Arria 10 is a 20-nm chip, while Stratix 10 is 14 nm; therefore, we use the delays of Arria 10, since it is the architecture with the closest technology node to the 22-nm technology node used when generating our logic and routing circuits in COFFE as discussed in Section 2.4.

5.2 VTR Flow: Synthesis and Packer Enhancements

In the following sections, we detail the VTR CAD enhancements we made to support each of our target architectures and to ensure high-quality implementation of multiplies and MACs.

5.2.1 Soft Multiplier Synthesis. To evaluate the proposed architectures on multiply and MAC benchmarks, we modified ODIN II to support implementing multiplies using soft logic. The user provides an input parameter to ODIN II defining the minimum multiplier size to be implemented using DSPs. ODIN II then implements all multiplications with both operands having bitwidths smaller than this number using soft logic. The partial products are synthesized as 2-input ANDs and the partial product reduction is implemented as a balanced adder tree for minimum critical path delay. We also modified ODIN II to take a parameter that specifies the size of the shadow multiplier in the architecture and implement small multiplies (<10 bits) using shadow multipliers and soft logic following the mapping discussed in Section 4.1.

5.2.2 Packing Carry Chains. To integrate the baseline and the proposed architectures in the VTR flow, some modifications to VPR are needed. Since, all the architecture changes are on the LAB level, all the modifications required in VPR are in and around the packer stage. The packer takes a netlist of *atoms* (i.e., LUTs, adders, registers, etc.) and groups them to produce a netlist of logic clusters (i.e., LABs). The packer tries to group connected logic together, with timing-critical connections given extra weight as capturing them within a cluster allows them to be connected with fast intra-cluster routing. Absorbing nets within a cluster also reduces demand for the general interconnect, saving power and making successful routing easier.

The packing of adder atoms is particularly important, as they need to be packed and placed together to ensure that the Cout-to-Cin connections between adjacent adder atoms are always routable through the fast carry links within and between the LABs as shown in Figures 1 and 2. For this purpose, Luu et al. [29] introduced a pre-packing stage in VPR 7.0. The user labels carry chain links in the architecture file with a *pack pattern* construct that instructs the pre-packer to capture structures in the atom netlist that match this pattern and group them together to build what VPR calls a *molecule*; all the atoms in a molecule are packed together in a logic cluster (see Figure 13).

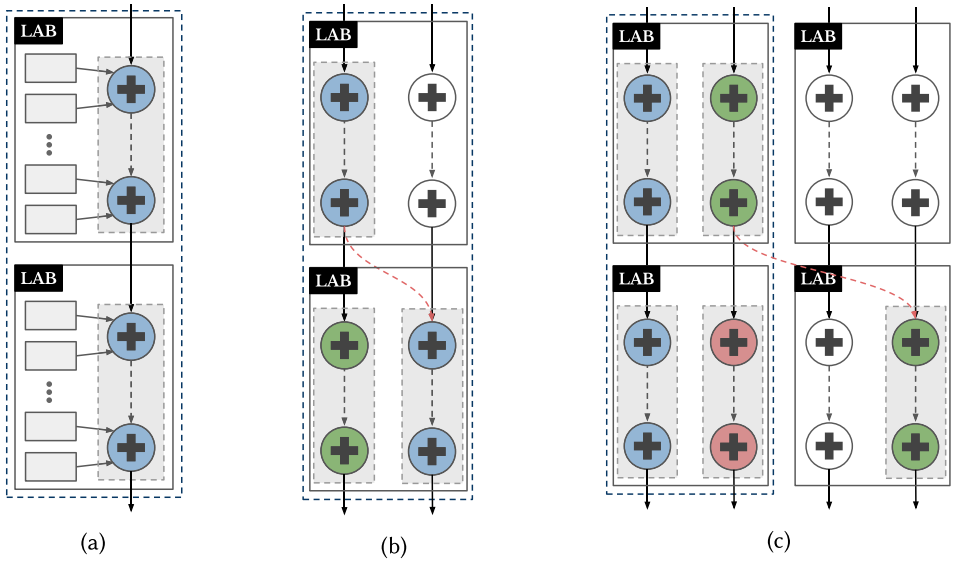

Fig. 13. (a) Packing a long addition that spans two LABs on an architecture with one carry chain per LAB where dashed box surrounding the LABs represent a placement macro and dashed box within the LABs represent a molecule. (b) Packing a long addition that spans two LABs on an architecture with two adder chains per LAB without keeping track of chain ID during packing. (c) Packing two long additions spanning two LABs on an architecture with two carry chains per LAB without the one long chain per LAB constraint. Curved dashed arrows represent Cout-to-Cin connection that are impossible to route with the given packing.

When an addition is wider than the maximum addition width per LAB, the addition is divided into multiple *sub-additions*. The width of the sub-additions is equal to the maximum addition width per LAB except for the last sub-addition, which could be smaller (i.e., remainder of addition width divided by the maximum addition width per LAB). Each sub-addition is packed into a LAB and the dedicated carry links between LABs are used for the Cout-to-Cin connections between sub-additions. These links are only available between vertically adjacent LABs, adding constraints on the placement of these LABs after packing. Therefore, VPR uses *placement macros*, which are similar to the concept of molecules, but on the cluster level. As shown in Figure 13(a), a placement macro groups all the LABs building a long addition together to ensure they are always placed in the right position relative to each other. Placement macros are built by VPR after packing and before placement by searching for used carry links between LABs to identify the LABs that need to be grouped together.

5.2.3 Logic Clusters with Multiple Carry Chains. The 4-bit Adder Double Chain architecture has two independent carry chains per LAB. However, the pre-packer in VPR always assumes one carry chain per LAB, similarly to Figure 1(a) or (b). Therefore, if an architecture has more than one carry chain, the VPR's packer would recognize the first one and completely ignore the others. Hence, we modify the packer to identify multiple chains within the LAB and label each of them with a unique ID. We also enhance the pre-packer to identify whether those chains are independent from one another as in the 4-bit Adder Double Chain architecture (see Figure 1(c)) or represent a single structure similar to the Extra Carry Chain architecture where the first chain feeds the second one as shown in Figure 4.

As shown in Figure 13(b), for architectures with multiple independent carry chains, such as the two-chain architecture of Figure 1(c), placement-macros alone are not sufficient to ensure

routability of the Cout-to-Cin connections between sub-additions. We must also ensure that an addition is implemented entirely using chain ID 1 or entirely using chain ID 2 in all LABs. Therefore, we modify the packer to keep track of the carry chain ID used to implement the first sub-addition in a long addition that spans multiple LABs. The packer then recognizes the other sub-additions of this addition and forces them to be packed at the same chain ID within their LABs. This will ensures that the Cout pin of one sub-addition will always reach the Cin pin of its adjacent sub-addition, as long as these LABs are aligned using placement-macros. Further, we add a constraint where a LAB can only be part of one long (multi-LAB) addition. This is to avoid potential routability problems (see Figure 13(c)) where the *Cout1* and the *Cout2* of the LAB are connected to two different LABs making it impossible to route both Cout-to-Cin connections (Cout pins of a LAB are connected only to the Cin pins of the LAB below it).

5.2.4 Logic Clusters with Multiple ALM Types. The 4-bit Adder single-chain architecture (see Figure 1(b)) has two types of ALMs; with and without adder chains. Therefore, additions must be packed only into the ALM type that has adder primitives, while LUT and register primitives can use either ALM type. To minimize the chance of running out of locations that can accept arithmetic, it is useful for the packer to prefer to use the simpler ALMs to implement LUTs and registers when there is a choice. To achieve this in a generic way, we modify the packer to recognize different ALM types within the LAB and quantify their flexibility depending on the number of primitives each ALM type contains. Thus, if an atom can be packed in two types of ALMs but one of them has more primitives than the other, then the packer will favour packing the atom in the ALM with fewer primitives. To avoid negatively affecting the behaviour of the packer, which has a complex timing and routability-aware gain function, this condition is only used for tie-breaking. Therefore, for the 4-bit Adder single-chain architecture, LUTs and registers that are not directly connected to additions prefer to be packed in the ALM type that has no adder primitive, improving the packing density of the architecture. This modification led to a 5% reduction in the number of LABs used by the 4-bit Adder single-chain architecture when implementing multiplies and MACs.

5.2.5 Deferred Legality Packing. As shown in Figure 7, when implementing multiplies the 4-bit Adder architectures use the 3-LUTs in front of the adders to implement the two-input AND gates generating partial products that are feeding those adders. The 4-bit Adder architectures can pack eight two-input LUTs and 4 bits of addition in one ALM compared to the baseline, which only packs four two-input LUTs and 2 bits of addition per ALM. Although we showed in Section 3.3 that the high level of input sharing between those LUTs allows this packing while using only seven of eight ALM input pins, the packer in VPR was not able to achieve this dense packing until we enhanced it as detailed below.

To understand why the packer fails to achieve this packing, consider the first level of addition in a 5-bit multiplication as shown in Equation (3).

$$\begin{array}{cccccc} & A_4 & A_3 & A_2 & A_1 & A_0 \\ \times & B_4 & B_3 & B_2 & B_1 & B_0 \\ \hline & I_{04} & I_{03} & I_{02} & I_{01} & I_{00} \\ + \; I_{14} & I_{13} & I_{12} & I_{11} & I_{10} & \\ \iddots & & \vdots & & & \end{array} \tag{3}$$

When packing this addition, the packer tries to pack the adder molecule first, then brings in the partial product LUTs one after another. For the 4-bit Adder architectures, pure addition uses all

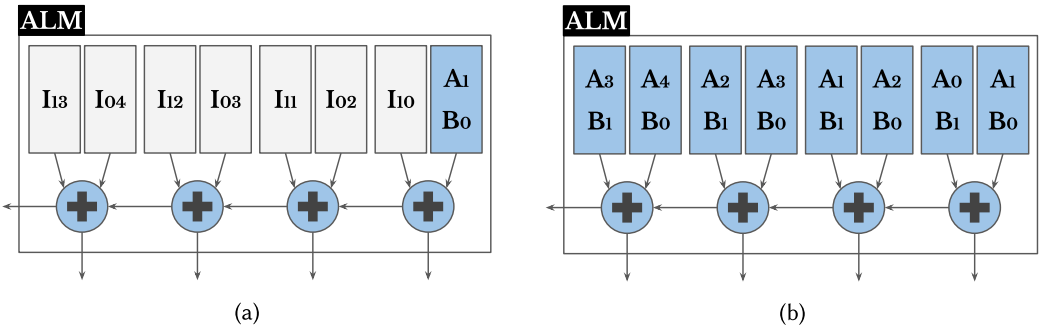

Fig. 14. Two steps of packing the addition surrounded by a dashed box in Equation (3) in the arithmetic ALM of the 4-bit Adder architecture. (a) After packing only one partial product LUT, the ALM needs nine distinct inputs, while (b) packing all the partial product LUTs reduces the number of required inputs to 7.

the ALM inputs, since there are 4 bits of addition per ALM. The packer then tries to bring in a two-input LUT feeding one of the adders as illustrated in Figure 14(a). Although this LUT will replace one of the used ALM input pins (one of the adder inputs is now connected to the output of this LUT), it will add two extra inputs, increasing the number of needed ALM inputs to 9. The packer will then reject this placement, since the ALM now needs more inputs than there are available. The key problem is that the VPR packer checks if a cluster is legal and routable after each atom is added. For the 4-bit adder architecture, however, packing only some of the partial product logic (LUTs) in front of an addition will result in an ALM with more than eight inputs. However, as shown in Figure 14(b) packing all the partial products possible will bring the input count back down to 7 as some signals are re-used.

We solve this problem with two changes. First, we specify a larger pack pattern in the VPR architecture file that includes not only the carry chain links between adders but also the links between the 3-input LUTs (partial product generators) and the adders. This ensures VPR tries to pack the partial products and adders as a unit (molecule) when possible. Second, we change the VPR packing legality check for this type of molecule. Instead of adding one atom at a time and checking legality, we first check if an entire molecule is legal, and if it is not legal, then we gradually remove atoms until it becomes legal. This new technique correctly handles the case where only a complete molecule packing is legal, but also still gracefully shrinks a molecule if it is too large to be legal. We use this technique for all the proposed ALM architectures as well as the baseline to achieve the densest packing possible for all of them and for a fair comparison. This modification has led to reducing the number of ALMs used by the baseline to implement multiplies by 15%, while for the 4-bit Adder architectures the number of ALMs was reduced by 50% and 41% for the single-chain and double-chain variants, respectively.

5.2.6 Efficient Adder Trees Implementation. The Extra Carry Chain architecture enhances efficiency by capturing two levels of an adder tree within one LAB. The molecule creation stage for this architecture affects how often we can use these two levels of addition (instead of just one level) within a LAB. For example, Figure 15 shows the accumulation of five 20-bit inputs (A–E) into a 20-bit SUM. In Figure 15(a) the creation of molecules leads to 1 LAB with two levels of addition used and 2 LABs with only one level used, while the molecules in Figure 15(b) leads to only 2 LABs with two levels of addition used, reducing area by one LAB. We enhanced VPR to always start creating molecules from the top of adder tree structures for this architecture. This maximizes the use of the two level adder hardware in a LAB, reducing both area and delay.

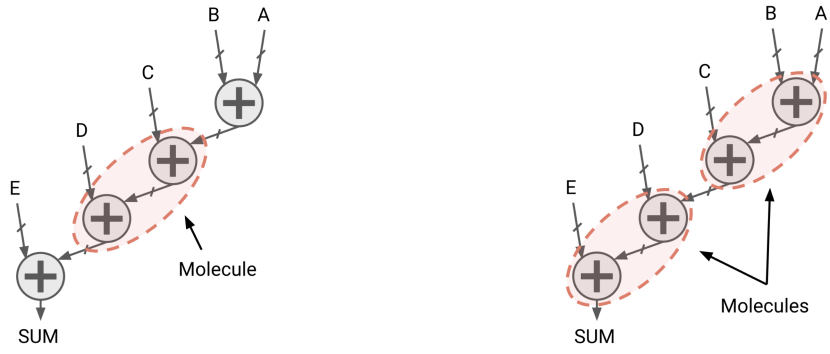

Fig. 15. Implementing the accumulation of five 20-bit inputs (A-E) into a 20-bit output using (a) 3 LABs and (b) 2 LABs depending on the molecule creation efficiency in the pre-packing stage.

5.3 Micro-Benchmarks: Multiplies and Multiply-Accumulate

We use VTR, with the CAD enhancements detailed in the prior sections, to implement multiplication and MAC benchmarks on the baseline and the proposed architectures to evaluate their post-routing area and speed. We evaluate multiplies over the same 4- to 9-bit precision range as Section 4.1; however, for completeness we evaluate all the operand precision combinations (4-bit × 4-bit, 4-bit × 5-bit to 9-bit × 9-bit) resulting in 36 multiplication benchmarks. The same approach is used to create 36 MAC benchmarks where the accumulation bit-width is chosen to match the multiplication output bit-width similarly to the convention followed in Section 4.1. We also register all the inputs and outputs of the multiplies and MACs to evaluate their critical path delays.

We use ALM and LAB counts to evaluate the area needed to implement each of the benchmarks. Although VPR tries to pack logic densely into clusters, the number of LABs can still be misleading for these micro-benchmarks as some LABs can be partially empty but would be filled with other logic in a larger design incorporating multiplication or MAC operations. Therefore, we also report the total number of ALMs used to implement the benchmarks. In addition, we modified the packer to report the number of *logic ALMs* (L-ALMs), which are the ones used to implement at least one LUT or adder or are made unusable by a shadow multiplier. Hence, the number of logic ALMs is the number of used ALMs without counting those used solely to implement registers. The logic ALM count is needed to compare the VPR results with our previous hand-mapping of Section 4.1, which did not consider registers. The logic ALM count is also useful, because the register density of our micro-benchmarks is fairly high as we register all inputs and outputs; often in a complete design the register density will be lower so some of these registers could go into other logic blocks. For routability and speed comparisons, we report the average routed wirelength and critical path delay achieved by the benchmarks on each architecture.

Figure 16(a) and (b) show the average number of LABs, ALMs and logic ALMs used to implement the multiplication and MAC benchmarks for the 4-bit Adder single-chain, 4-bit Adder double-chain, Extra Carry Chain, 4-, 6-, and 9-bit Shadow Multiplier architectures. All the results in this section and throughout the rest of the article are geometrically averaged over all the benchmarks and normalize the results of the Stratix-10-like baseline architecture. For multiplication, the two variants of the 4-bit Adder architecture achieve similar reductions in logic ALMs, 43% and 42% for the single- and double-chain variants, respectively. This is slightly better than the 36% reduction achieved by the hand-mapping, since the hand-mapping counted partially filled ALMs (e.g., the bottom ALM in Figure 7) as a used ALM while VPR continues packing logic into these ALMs

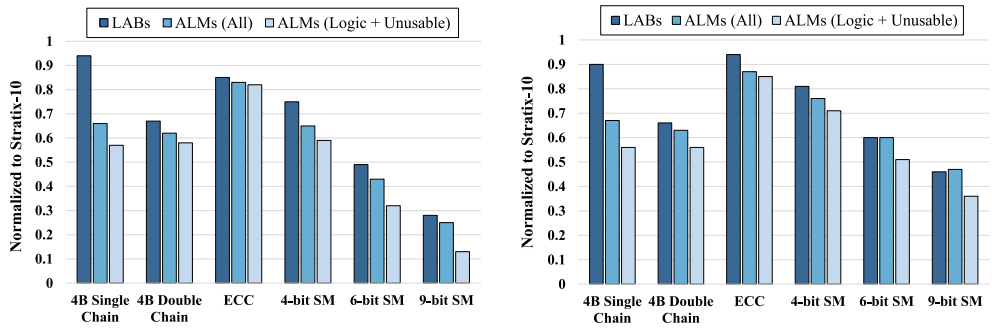

Fig. 16. Average VPR resource utilization results normalized to the Stratix-10-like baseline architecture for implementing (a) multiplication and (b) MAC benchmarks on the proposed architectures.

when possible, resulting in more area reduction. On the LAB level the 4-bit Adder single-chain architecture shows a reduction of only 6% as it is constrained by the fact that only half the ALMs can perform addition; hence, it can implement the same number of addition bits per lab as the basline. The double-chain variant achieved a much larger 33% reduction in LABs, as it allows every ALM to perform 4 bits of addition. For MACs, both architectures achieve a 44% logic ALM reduction, while the single- and double-chain variants achieve LAB reductions of 10% and 34%, respectively.

The Extra Carry Chain architecture resource utilization for multiplications and MACs are given by the ECC columns in Figures 16(a) and (b), respectively. While the Extra Carry Chain architecture achieved nearly as large ALM reductions as the 4-bit Adder architectures in our hand mapping results, it achieves smaller reductions of 15% to 18% in logic ALMs when run through the full VTR CAD system. The basic reason is that the Extra Carry Chain architecture is a more difficult target for CAD tools. For example, to achieve the best hand mapping of multiplications to the Extra Carry Chain architecture, we optimized both the order of additions and the grouping of additions into one ALM for each multiplier size individually. However, in VPR we use the heuristic of Section 5.2.6, which does not always achieve optimal results. When hand-mapping MACs, we pack the accumulation into the second carry chain in the ALMs implementing the last stage of the multiplication adder tree. This optimization requires packing-aware re-synthesis of the multiplication logic, in which ”pass-through” addition cells are added to allow some internal multiply signals to feed the second level of adders, enabling them to implement the accumulation. We have not implemented this complex optimization; therefore, our results for this architecture are not as good as our hand mapping—we leave this out for future work.

As shown in Figure 16(a) and (b), for multiplication and MAC benchmarks, increasing the size of the shadow multiplier results in larger resource count reductions. For multiplication benchmarks, the reduction in LABs is 25% for the 4-bit variant and reaches 72% for the 9-bit Shadow Multipliers architecture. At the ALM level, logic ALMs (including unusable ALMs) reductions vs. the baseline range from 42% to 87% for the 4- and 9-bit variants, respectively. The logic ALM reductions achieved by the Shadow Multiplier architectures are smaller but still significant for MAC benchmarks; they range from 19% to 54% LAB reduction and 29% to 65% logic ALM reduction for the 4- and 9-bit variants, respectively. The Shadow Multiplier architectures have no advantage over the baseline when implementing stand-alone additions. Therefore, for MAC benchmarks the extra addition at the multiplication output dilutes the resource reductions compared to those of the multiplication benchmarks.

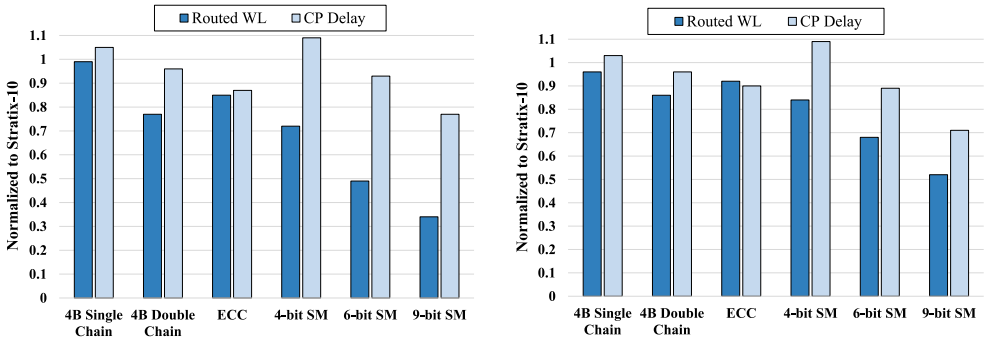

Fig. 17. Average VPR routed wirelength and critical path delay results normalized to the Stratix-10-like baseline architecture for implementing (a) multiplication and (b) MAC benchmarks on the proposed architectures.

The reductions in logic ALMs achieved by the full CAD flow are somewhat smaller than those of the hand-mapping of Section 4.1. This results from a slight difference between the hand-mapping and ODIN II when implementing multiplications that are larger than the shadow multiplier size. ODIN II always splits this multiplication into four multiplications, as shown in Figure 11(b), and then chooses which ones to implement using shadow multipliers and which ones to implement using soft logic according to the criteria discussed in Section 4.1. The hand-mapping instead takes a more global view when combining soft logic and a shadow multiplier to create a larger array, as shown in Figure 11(a). It creates the entire partial product reduction tree at once, which often saves logic ALMs vs. the two-step automated solution.

As shown in Figure 17(a) and (b), all the proposed architectures show reductions in the routed inter-LAB wirelength and most reduce the critical path delay vs. the baseline when implementing multiplies and MACs. By reducing the ALM counts needed and capturing many connections within the more powerful ALMs and logic blocks, wirelength has been reduced and speed is often enhanced. The gains are most substantial for the architectures with the largest ALM count reductions; the 9-bit Shadow Multiplier architecture reduces wirelength by 66% on multiplications and 48% on MACs, respectively.

Unlike most of the architectures, the 4-bit Adder single-chain and the 4-bit Shadow Multiplier increase the critical path delay for both multiplies and MACs. For the 4-bit Adder single-chain architecture, routing delays increase the critical path delay for multiplies by 5% and MACs by 3%. Only half the ALMs of this architecture support arithmetic operations. Therefore, for such arithmetic-intensive benchmarks the LAB output pins connected to this half become highly utilized, stressing the routing and increasing routing delays. The 4-bit Shadow Multiplier architecture also shows an increase in critical path delay compared to the baseline of 9% for both multiplication and MAC benchmarks. As mentioned in Section 2.4, all the shadow multipliers have internal path delays of 1 ns or less. However, 4-bit soft multiplies implemented on the baseline architecture can achieve a delay of less than 1 ns; therefore, achieving higher speed for multiplication and MAC benchmarks compared to the 4-bit Shadow Multiplier architecture.

5.4 Machine Learning Kernel

Due to the limited Verilog language coverage of ODIN II, it was not possible to use full ML accelerator benchmarks to evaluate the proposed architectures. Therefore, in this section, we use a 4 ×

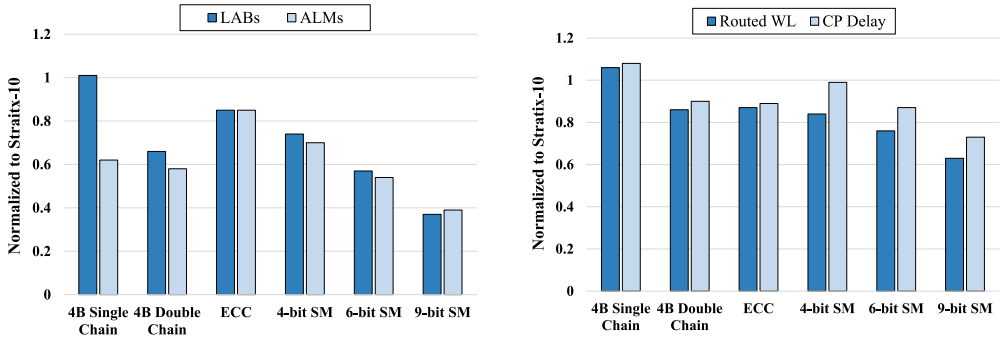

Fig. 18. Average VPR (a) resource utilization, (b) routing wirelength and critical path delay results normalized to the Stratix-10-like baseline architecture for implementing 4 × 4 fully pipelined matrix multiplication benchmarks on the proposed architectures.

4 matrix multiplication benchmark to resemble a ML kernel with high multiplication and addition densities. We vary the operands precision from 4 to 9 bits to generate a total of six 4 × 4 matrix multiplication benchmarks and use them to evaluate our proposed architectures. All operations within the matrix multiplication are carried out in parallel resulting in 64 multiplications and 48 two-input additions for each benchmark. The benchmarks are also fully pipelined, which is typical for an ML kernel, to maximize throughput. For a a 4 × 4 matrix multiplication with 4-bit precision, VPR reports a max frequency of 318 MHz for the Stratix-10-like baseline architecture in a 22-nm technology.

Figure 18(a) shows the average resource utilization of the proposed architectures when implementing the 4 × 4 matrix multiplication benchmarks, normalized to the results of the baseline architecture. All architectures show reductions in LAB counts except for the 4-bit Adder single-chain architecture. These benchmarks have very high arithmetic density; therefore, similarly to the multiplication benchmarks results, the 4-bit Adder single-chain architecture is again constrained by the number of addition bits per LAB and shows a small increase of 1% in LAB count vs. the baseline. However, the 4-bit Adder single-chain architecture shows a promising reduction in ALMs of 38% indicating that the LABs are still partially empty and LAB reductions could be achieved with a design that has more non-arithmetic logic to fill these LABs.

The 4-bit Adder double-chain architecture, achieves a 34% reduction in LABs and 42% reduction in ALMs vs. the baseline. This closely matches the resource reductions achieved when implementing multiplication benchmarks. This shows that the 4-bit Adder double-chain architecture can maintain its area efficiency for larger arithmetic heavy designs. The Extra Carry Chain architecture achieves only 15% reduction in ALMs and LABs. This also matches its resource reductions on the multiplication benchmarks, so it has a consistent, but lesser, efficiency.

For the Shadow Multiplier architectures the area efficiency increases with the shadow multiplier size; the reduction in LABs vs. the baseline varies from 26% to 63% for the 4- and 9-bit Shadow Multipliers, respectively. A 6-bit Shadow Multiplier shows a 46% reduction in ALMs, which is very close to the 42% achieved by the 4-bit Adder double-chain architecture. However, the 4-bit Adder double chain has a smaller FPGA tile area overhead as discussed in Section 4.3.

Figure 18(b) shows the critical path delay and (inter-cluster) routed wirelength averaged over the 4 × 4 multiplication benchmarks for the various architectures. The 4-bit Adder single-chain architecture shows an 8% degradation in critical path delay and a 6% increase in routed wirelength

vs. the baseline. This is aligned with the results achieved by the 4-bit Adder single-chain architecture on multiplication and MAC benchmarks and is unsurprising given that it did not reduce LAB count for the matrix multiplication kernels. The 4-bit Adder double-chain and the Extra Carry Chain architectures reduce critical path delay by 10% and 11% and routed wire length by 14% and 13%, respectively. This is also logical as the LAB count and the number of connections between LABs have been reduced. Large shadow multipliers (9-bit) also reduce critical path delay (by 27%) and routed wirelength (by 37%), while smaller shadow multipliers show smaller gains in these metrics. Overall, the 4-bit double-chain architecture and the larger Shadow Multiplier architectures achieve the greatest LAB count, routed wirelength and critical path delay gains on these benchmarks.

5.5 General Application Benchmarks

So far we have evaluated the area and delay efficiency of the proposed architectures when implementing arithmetic-intensive applications and shown they can achieve significant gains in this target application area. However, we still need to ensure that these architecture changes do not negatively impact the performance of other applications. We use the nine larger benchmarks[2] (>10K primitives) from the VTR 7.0 heterogeneous benchmark suite, introduced and detailed in Table II in Reference [29], as our general application benchmark suite. Those benchmarks come from a variety of real applications including finance, medical, physics, and math. Three of those benchmarks have no multiplications, four have a maximum of 32 multiplications and two have high multiplication densities of 152 (stereovision1) and 564 (stereovision2) multiplications. Therefore, to evaluate the impact of our architecture changes on general application benchmarks, we implement those benchmarks on each architecture and evaluate their post-routing area and speed vs. the baseline. For these designs we let ODIN II use hard multiplication vs. soft logic with its usual heuristics; all multiplies larger than than 2-bit × 2-bit are implemented with hard multipliers. Note that the Shadow Multiplier architectures have hard multipliers not only in DSP blocks, but also in the LABs. Therefore the VPR packer considers both choices for implementing each multiplication requested while giving the shadow multipliers higher priority when implementing small multiplies (that can fit into the shadow multiplier), since they have smaller area footprint than DSP.

Figure 19(a) shows the LAB and ALM resource usage of each architecture variant, geometrically averaged over the 9 benchmark circuits and normalized to the baseline architecture. It also shows the average *grid size* required by each architecture; this is the total number of grid locations at which blocks can be placed (rows × columns) in an FPGA with enough LABs, DSPs, RAM, and IO blocks to implement a benchmark. For all the results in this section, the DSP utilization of the proposed architectures did not increase compared to the baseline architecture. Therefore, reductions in LAB and ALM resources directly reflect the better resource efficiency of the proposed architectures. The two variants for the 4-bit Adder architecture show the same reductions in LABs and ALMs vs. the baseline of 9% and 11%, respectively. This area reduction comes from the capability of the 4-bit Adder architectures to implement 4 bits of addition per ALM compared to only 2 bits of addition per ALM in the baseline. Adders are common, so these general benchmarks benefit significantly from the new addition capabilities. While the 4-bit Adder double-chain architecture has a superior LAB resource reduction for multiplication kernels, the 4-bit Adder single-chain architecture, which provides only half as much addition per LAB, is equally efficient in these general applications. In addition, both architectures show similar reductions in grid size of

[2]The benchmarks are arm core, bgm, blob merge, LU8PEEng, LU32PEEng, mcml, stereovision0, stereovision1, and stereovision2.

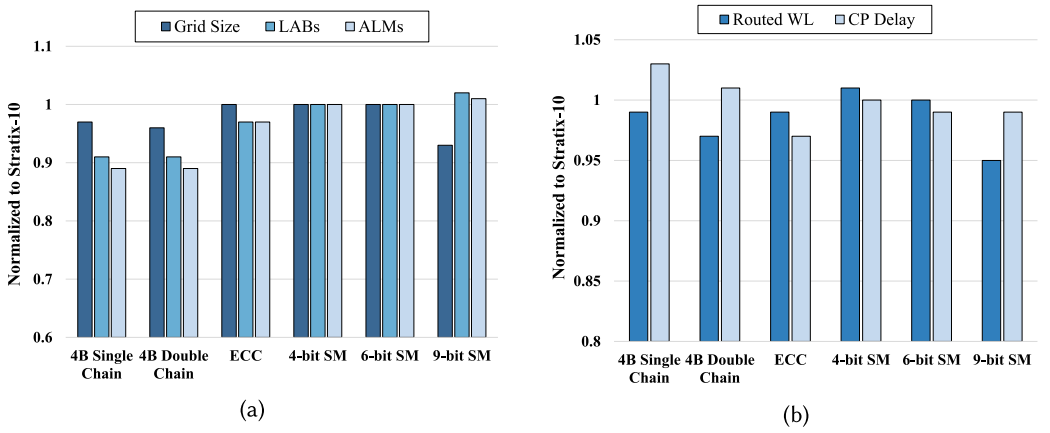

Fig. 19. Average VPR (a) resource utilization, (b) routing wirelength and critical path delay results normalized to the Stratix-10-like baseline architecture for implementing large (>10K primitives) VTR benchmarks.

3% and 4% for the single- and double-chain variants, respectively—they reduce the FPGA size for LAB-constrained circuits that had many additions. The Extra Carry Chain architecture achieves smaller resource reductions, as it can only benefit adder trees rather than all additions. Therefore, this architecture only shows a 3% reduction in ALMs and LABs vs. the baseline and has no impact on the FPGA grid size.

Smaller shadow multipliers (4- and 6-bit) have no impact on ALM and LAB counts or on the grid size—there are simply not enough small multiplications in the VTR benchmarks for these shadow multipliers to be used much. The 9-bit shadow multipliers, however, are used by some of the benchmarks and lead to less than 2% increase in both ALM and LAB counts. This reduces DSP block usage and enables a 7% reduction in grid size—benefiting from DSP-constrained benchmarks.

Figure 19(b) shows the inter-block routed wirelength and the post-routing critical path delay, both geometrically averaged over all the benchmarks and normalized to the baseline architecture. The single and double-chain variants of the 4-bit Adder architecture slightly reduce the routed wirelength vs. the baseline by 1% and 3%, respectively. They also slightly increase critical path delay by 3% for the single chain and 1% for the double chain. This is logical as each architecture has slightly increased some delays within the LAB and routing, as detailed in Section 4.3. The Extra Carry Chain architecture also slightly reduces routed wirelength (by 1%) and improves critical path delay (by 3%) as it has captured some timing-critical connections within adder trees into its enhanced ALMs. Architectures with small (4- and 6-bit) shadow multipliers cause little change in routed wirelength and critical path delay, again as their shadow multipliers are not used much by these benchmarks. The 9-bit Shadow Multiplier architecture reduces wirelength (by 5%) and critical path delay (by 1%); it appears that the wider availability of hard multipliers (in both LABs and DSPs) has enabled a placement with less wiring for benchmarks with 9-bit or smaller multiplications.

We also experimented with changing the VTR synthesis settings to map 9-bit or smaller multiplications to the soft fabric for benchmarks whose grid size is set by their DSP block requirements with the default ODIN II settings. There are two large (>10K primitives) DSP-constrained benchmarks in the VTR benchmark suite. We implement those benchmarks on the baseline, 4-bit Adder double chain, and 6-bit Shadow Multiplier architectures while instructing ODIN II to implement 9-bit or smaller multiplies using soft logic or shadow multipliers. Note that the 9-bit Shadow Multiplier architecture is already capturing these small multiplies into its hard shadow multipliers, so

Table 4. Grid Size and Critical Path Delay Results of Implementing the Two Large (> 10K Primitives) DSP-constrained Benchmarks from the VTR Benchmark Suite while Implementing 9-bit Multiplies or Smaller Using Soft Logic

	Stratix-10-like		4B Double Chain		6-bit SM		9-bit SM	
	Grid Size	**CP Delay**	**Grid size**	**CP delay**	**Grid Size**	**CP delay**	**Grid size**	**CP delay**
Stereovision1	1.10×	0.83×	0.86×	0.91×	0.78×	0.72×	0.56×	0.60×
Stereovision2	0.69×	0.96×	0.69×	0.94×	0.88×	1.06×	0.88×	1.05×
Geomean	**0.88×**	**0.89×**	**0.77×**	**0.93×**	**0.87×**	**0.87×**	**0.70×**	**0.79×**

All results are normalized to the baseline architecture with default synthesis settings.

its results remain the same as in Figure 19(a). Compared to the default ODIN II setting, all proposed architectures reduce the FPGA grid size for those benchmarks. Table 4 shows the grid size required and critical path delay achieved by each architecture with these synthesis settings, all normalized to the results of the baseline architecture with default synthesis settings. The baseline architecture achieves a 12% average grid size reduction by mapping small multiplies to the fabric, and interestingly this does not slow down the circuits. The enhanced architectures show larger gains: Twenty-three percent grid size reduction for the 4-bit Adder double-chain architecture and 13% for the 6-bit and 30% for the 9-bit Shadow Multiplier architectures. All these enhanced architectures also have smaller average critical path delays than the baseline architecture with default synthesis settings.

6 RESULTS SUMMARY

When implementing the benchmarks on the proposed architectures using the VTR flow, we used COFFE's reported delays for each architecture to define the delays of logic and routing paths within the architectures for VTR to calculate the critical path delays for each benchmark circuit. However, in the prior sections we presented only resource counts for implementing the benchmarks, rather than silicon area. Since our architectural changes increase the area of the FPGA logic and routing tile, we need to combine the resource usage results with COFFE's tile area results from Section 4.3.

For multiplies and MACs we compute the silicon area of the used logic ALMs (which includes their associated routing), as these micro-benchmarks are small enough that using LAB utilization results in significant round off effects. For the matrix multiplication kernels we use the silicon area of the used LAB tiles, since these kernels are larger and round off to the logic block count is less significant. For the VTR benchmarks we show area of the used LABs and of the entire FPGA grid; both metrics are important as increased use of LAB-based multiplication can reduce DSP block usage and shrink the grid. We conservatively use ODIN II's default synthesis settings when mapping the VTR benchmarks; as detailed in Section 5.5, further gains in grid size are possible when we more aggressively use the soft fabric for small multiplies in DSP-constrained designs.

Table 5 summarizes these areas and critical path delays of the proposed architectures vs. the baseline for various types of benchmarks from multiplies to entire applications. The 4-bit Adder double-chain architecture performs very well across all the test cases. It achieves a large reduction in area at the ALM level for stand-alone multiplies and MACs (40% to 42%), which translates to large area reductions at the LAB level (by 32%) for the matrix multiplication kernels. It also reduces the used LAB area and total FPGA grid area for the VTR benchmarks by 6% and 1%, respectively. It has minimal impact on the critical path delay of general applications and speeds up multiplication and matrix multiplication kernels by 4% to 10%. The single-chain variant is not as consistent; it

Table 5. Overall Area and Delay Results Normalized to the Stratix-10-like Baseline Architecture for Implementing Various Benchmarks on the Proposed Architectures

		4-bit Adder Single Chain	4-bit Adder Double Chain	Extra Carry Chain	4-bit Shadow Multiplier	6-bit Shadow Multiplier	9-bit Shadow Multiplier
Multiplication	**Area (L-ALM)**	0.57×	**0.60×**	0.84×	0.61×	0.34×	0.15×
	CP Delay	1.05×	**0.96×**	0.87×	1.09×	0.93×	0.77×
Multiply-Accumulate	**Area (L-ALM)**	0.56×	**0.58×**	0.87×	0.73×	0.54×	0.40×
	CP Delay	1.03×	**0.96×**	0.90×	1.09×	0.89×	0.71×
Matrix Multiplication	**Area (LAB)**	1.02×	**0.68×**	0.87×	0.76×	0.60×	0.42×
	CP Delay	1.08×	**0.90×**	0.89×	0.99×	0.87×	0.73×
VTR Benchmarks	**Area (Grid)**	0.98×	**0.99×**	1.02×	1.03×	1.06×	1.05×
	Area (LAB)	0.92×	**0.94×**	0.99×	1.03×	1.06×	1.15×
	CP Delay	1.03×	**1.01×**	0.97×	1.00×	0.99×	0.99×

performs well at the ALM level, but the fact that only half the ALMs in a LAB have carry chains means it has no area improvement at the LAB level for the matrix multiplication kernels.

The Extra Carry Chain architecture achieves smaller, but consistent, area reductions of 13% to 16% across multiplication, MAC and matrix multiplication kernels. Since our hand mapping of multiplications and MACs can achieve somewhat better area reductions than this, it is possible CAD improvements could lead to further area gains. However, those CAD algorithms would involve complex interactions between synthesis and packing and, even in the best case, would not fully close the gap with the 4-bit Adder double-chain architecture.

The Shadow Multiplier architectures also show consistent area gains across the multiplication, MAC and matrix multiplication kernels, with the gains increasing as larger shadow multipliers are used. The architecture with the largest shadow multiplier (9-bit) achieves 85%, 60%, and 58% area reductions for multiplication, MAC and matrix multiplication kernels, respectively. However, shadow multipliers are more costly in silicon area and are not heavily used in general (VTR) benchmarks, so they also have an area penalty on the VTR benchmarks, ranging from 3% (LAB and grid) for a 4-bit Shadow Multiplier to 15% used LAB area and 5% total grid area at 9 bits. The 9-bit Shadow Multiplier architecture also significantly reduces the delay of multiplications (by 23%), MACs (by 29%) and matrix multiplication kernels (by 27%).

Considering all the results, the most compelling architectures are the 4-bit Adder double-chain and 6- and 9-bit Shadow Multiplier architectures. The 4-bit Adder double-chain architecture improves multiplication and MAC density by ~1.7× and matrix multiplication density by 1.5×, while simultaneously increasing their speed. It also *reduces* the used LAB area of general benchmarks by 8% and the total grid area by 3%, which means that this architecture change is a win in every application domain. The 6- and 9-bit Shadow Multiplier architectures can increase multiplication density even more, by 2.9× and 6.7×, respectively, which lead to density increases for matrix multiplication kernels of 1.7× and 2.4×, respectively. These shadow multipliers do come at a cost, however; the 6- and 9-bit Shadow Multipliers architectures increase the used LAB area of general (VTR) benchmarks by 6% and 15%, respectively and the grid area by 6% and 5%, respectively.

In this study, we focused on implementing low-precision multiplications using the FPGA soft fabric. We proposed six architectures that can increase the area efficiency of implementing low-precision multiplications using logic blocks. Since, we focused on a precision range from 4 to 9 bits, we need to discuss the area-efficiency of implementing higher precision multiplications on the proposed architectures. The 4-bit Adder architectures double the number of adders per arith-

metic ALM compared to Stratix 10. Therefore, regardless of the operands precision, the 4-bit Adder architectures will always be able to pack twice as much additions per LAB compared to Startix 10 and therefore will show consistent area gains for all multiplication precisions. The Extra Carry Chain architecture adds a second level of carry chain to the ALMs and therefore can implement adder trees efficiently. This is also independent of the operand precision, and, therefore, the area gains will also be consistent for high-precision multiplications. However, the Shadow Multiplier architectures add a fixed size low-precision multiplier to the FPGA logic blocks. Shadow Multipliers can still be combined together with ALMs to implement multiplications of higher precision than the Shadow Multiplier size (see Figure 11). However, the gains are higher when implementing multiplications that can fit entirely in the shadow multiplier—having the same or smaller precision than the shadow multiplier.

7 CONCLUSION

In this article, we proposed three types of FPGA architectural changes and six architectures that can achieve promising area and delay savings vs. a Stratix-10-like logic architecture for applications using low-precision multiplication. Two of our changes are on the ALM level where we double the number of adders in the ALM while the third is on the LAB level where we add a low-precision (4-, 6-, and 9-bit) shadow multiplier to the LAB that shares the input and output LAB interface with some ALMs within the LAB. We evaluate each change in detail, using COFFE for transistor sizing, area estimation, and delay extraction of the logic and routing of each architecture. Then, we use VTR to implement four sets of benchmarks that cover both arithmetic heavy applications and general applications to produce post-routing area and speed results.

The 4-bit Adder double-chain architecture performs well in every application domain. It achieves a 1.7× area reduction for multiplication benchmarks and a 1.5× area reduction for matrix multiplication benchmarks compared to the Stratix 10 architecture. In addition, it also shows speed improvements between 4% and 10% over the baseline architecture for all arithmetic-heavy benchmarks. By making additions more efficient, this architecture also reduces the LAB area of general benchmarks by 6% and causes only a 1% increase in critical path delay on these general designs. The 6- and 9-bit Shadow Multiplier architectures achieve even greater gains vs. the baseline on multiplication-intensive designs: 2.9× and 6.7× area reduction for multiplication benchmarks and 1.7× and 2.4× area reduction for matrix multiplication benchmarks, respectively. However, these high gains come at a cost for general applications, increasing used LAB area by 6% and 15% for the 6- and 9-bit Shadow Multiplier architectures, respectively.

Given the large and growing importance of deep learning and therefore of multiplication and MAC operations, we believe future FPGAs should adopt one of these more efficient architectures.

REFERENCES

[1] Y. Cao. 2018. Predictive Technology Model (PTM). Retrieved from 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. *IEEE Trans. VLSI Syst.* 12, 3 (2004), 288–298.
[3] C. Baugh and B. Wooley. 1973. A two's complement parallel array multiplication algorithm. *IEEE Trans. Comput.* C-22, 12 (1973), 1045–1047.
[4] V. Betz and J. Rose. 1997. Cluster-based logic blocks for FPGAs: Area-efficiency vs. input sharing and size. In *Proceedings of the Custom Integrated Circuits Conference*. 551–554.
[5] V. Betz and J. Rose. 1998. How much logic should go in an FPGA logic block. *IEEE Design & Test of Computers*, 15, 1 (1998), 10–15.
[6] A. Boutros et al. 2018. Embracing diversity: Enhanced DSP blocks for low-precision deep learning on FPGAs. In *Proceedings of the International Conference on Field Programmable Logic and Applications*. 1–8.
[7] A. Boutros et al. 2018. You cannot improve what you do not measure: FPGA vs. ASIC efficiency gaps for convolutional neural network inference. *ACM Trans. Reconfig. Technol. Syst.* 11, 3 (2018), 1–23.

[8] A. Boutros et al. 2019. Math doesn't have to be hard: Logic block architectures to enhance low-precision multiply-accumulate on FPGAs. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 94–103.
[9] M. Burich. 2012. Conference workshop: FPGAs in 2032, challenges and opportunities in the next 20 years, convergence of programmable solutions. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*.
[10] S. Chandrakar et al. 2015. Enhancements in UltraScale CLB architecture. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 108–116.
[11] C. Chiasson and V. Betz. 2013. COFFE: Fully-automated transistor sizing for FPGAs. In *Proceedings of the International Conference on Field-Programmable Technology*. 34–41.
[12] Intel Corporation. 2005. *Stratix GX Transeiver User Guide*.
[13] Xilinx Corporation. 2007. *Virtex-II Platform FPGA User Guide*.
[14] Xilinx Corporation. 2007. *Virtex-II Pro and Virtex-II Pro X FPGA User Guide*.
[15] M. Deo et al. 2019. Intel Stratix 10 MX devices solve the memory bandwidth challenge. Intel Whitepaper.
[16] J. Fowers et al. 2018. A configurable cloud-scale DNN processor for real-time AI. In *Proceedings of the International Symposium on Computer Architecture*, 1–14.
[17] S. Han et al. 2016. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of the International Symposium on Computer Architecture*. 243–254.
[18] K. He et al. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*. 770–778.
[19] Intel Corporation. 2017. Intel Stratix 10 logic array blocks and adaptive logic modules user guide (UG-S10LAB).
[20] P. Jamieson and J. Rose. 2006. Enhancing the area-efficiency of FPGAs with hard circuits using shadow clusters. In *Proceedings of the International Conference on Field Programmable Technology*. 1–8.
[21] A. Krizhevsky et al. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1097–1105.
[22] I. Kuon and J. Rose. 2011. Exploring area and delay tradeoffs in FPGAs with architecture and automated transistor design. *IEEE Trans. VLSI Syst.* 19, 1 (2011), 71–84.
[23] M. Langhammer et al. 2019. Fractal synthesis: Invited tutorial. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 202–211.
[24] M. Langhammer and B. Pasca. 2015. Floating-point DSP block architecture for FPGAs. In *Proceedings of the International Symposium on Field Programmable Gate Arrays*. 117–125.
[25] Guy Lemieux and David Lewis. 2001. Using sparse crossbars within LUT. In *Proceedings of the International Symposium on Field Programmable Gate Arrays*. 59–68.
[26] D. Lewis et al. 2005. The Stratix II logic and routing architecture. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 14–20.
[27] D. Lewis et al. 2016. The Stratix 10 highly pipelined FPGA architecture. In *Proceedings of the International Symposium on Field Programmable Gate Arrays*. 159–168.
[28] D. M. Lewis et al. 2003. The StratixTM routing and logic architecture. In *Proceedings of the International Symposium on Field Programmable Gate Arrays*. 12–20.
[29] J. Luu et al. 2014. VTR 7.0: Next generation architecture and CAD system for FPGAs. *ACM Trans. Reconfig. Technol. Syst.* 7, 2 (2014), 1–30.
[30] A. Mishra et al. 2017. WRPN: Wide reduced-precision networks. arXiv preprint arXiv:1709.01134.
[31] Kevin E. Murray et al. 2020. VTR 8: High performance CAD and customizable FPGA architecture modelling. *ACM Trans. Reconfig. Technol. Syst.* 0, ja, 1
[32] E. Nurvitadhi et al. 2016. Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC. In *Proceedings of the International Conference on Field-Programmable Technology*. 77–84.
[33] E. Nurvitadhi et al. 2017. Can FPGAs beat GPUs in accelerating next-generation deep neural networks? In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 5–14.
[34] J. et al. Qiu. 2016. Going deeper with embedded FPGA platform for convolutional neural network. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 26–35.
[35] E. Real et al. 2018. Regularized evolution for image classifier architecture search. arXiv preprint arXiv:1802.01548.
[36] J. Rose et al. 1993. Architecture of field-programmable gate arrays. *IEEE J. Solid-State Circ.* 81, 7 (1993), 1013–1029.
[37] V. Rybalkin et al. 2018. FINN-L: Library extensions and design trade-off analysis for variable precision LSTM networks on FPGAs. In *Proceedings of the International Conference on Field Programmable Logic and Applications*. 1–8.
[38] S. M. Trimberger. 2015. Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology. *Proc. IEEE*, 318–331.
[39] Mike W. et al. 2019. Virtex UltraScale+ HBM FPGA: A revolutionary increase in memory performance. *Xilinx Whitepaper*.

[40] Xiaowei X. et al. 2018. Scaling for edge inference of deep neural networks. *Nat. Electron.* 1, 4 (2018), 216–222.
[41] S. Yazdanshenas and V. Betz. 2017. Automatic circuit design and modelling for heterogeneous FPGAs. In *Proceedings of the International Conference on Field Programmable Technology*. 9–16.
[42] S. Yazdanshenas and V. Betz. 2019. COFFE 2: Automatic modelling and optimization of complex and heterogeneous FPGA architectures. *ACM Trans. Reconfig. Technol. Syst.* 12, 1 (2019), 1–27.

Received October 2019; revised March 2020; accepted April 2020