Double Duty: FPGA Architecture to Enable Concurrent LUT and Adder Chain Usage

Junius Pun*
Nanyang Technological University
JPUN001@e.ntu.edu.sg

Xilai Dai*
Cornell University
xd44@cornell.edu

Grace Zgheib
Altera
grace.zgheib@altera.com

Mahesh A. Iyer
Altera
mahesh.iyer@altera.com

Andrew Boutros
University of Waterloo
andrew.boutros@uwaterloo.ca

Vaughn Betz
University of Toronto
vaughn@ece.utoronto.ca

Mohamed S. Abdelfattah
Cornell University
mohamed@cornell.edu

***Abstract*—Flexibility and customization are key strengths of Field-Programmable Gate Arrays (FPGAs) when compared to other computing devices. For instance, FPGAs can efficiently implement arbitrary-precision arithmetic operations, and can perform aggressive synthesis optimizations to eliminate ineffectual operations. Motivated by sparsity and mixed-precision in deep neural networks (DNNs), we investigate how to optimize the current logic block architecture to increase its arithmetic density. We find that modern FPGA logic block architectures prevent the independent use of adder chains, and instead only allow adder chain inputs to be fed by look-up table (LUT) outputs. This only allows one of the two primitives—either adders or LUTs—to be used independently in one logic element and prevents their concurrent use, hampering area optimizations. In this work, we propose the Double Duty logic block architecture to enable the concurrent use of the adders and LUTs within a logic element. Without adding expensive logic cluster inputs, we use 4 of the existing inputs to bypass the LUTs and connect directly to the adder chain inputs. We accurately model our changes at both the circuit and CAD levels using open-source FPGA development tools. Our experimental evaluation on a Stratix-10-like architecture demonstrates area reductions of 21.6% on adder-intensive circuits from the Kratos benchmarks, and 9.3% and 8.2% on the more general Koios and VTR benchmarks respectively. These area improvements come without an impact to critical path delay, demonstrating that higher density is feasible on modern FPGA architectures by adding more flexibility in how the adder chain is used. Averaged across all circuits from our three evaluated benchmark set, our Double Duty FPGA architecture improves area-delay product by 9.7%.**

I. Introduction

Field-Programmable Gate Arrays (FPGAs) offer bit-level programmability at the price of performance when compared to Application-Specific Integrated Circuits (ASICs). This enables their flexible use in many application domains where deployment-time reconfigurability is warranted such as cloud networking [1]–[3], wireless communication [4]–[6], and machine learning [7]–[10]. Many mainstream machine learning workloads depend on high-performance dense matrix multiplications, pushing FPGA vendors to either bolster their FPGAs with tensor compute units [11], [12] or resort to heterogeneous FPGA-CGRA hybrid computing devices [13]. However, as machine learning becomes ever more prevalent, it is finding its way into a broad variety of applications that can take advantage of the flexibility of the FPGAs' soft logic. This is highlighted by the plethora of works to create new frameworks [14], methods [9], [15]–[17], benchmarks [18], and architectures [19], [20] to improve the efficiency of deep neural networks (DNNs) implemented using the FPGA fabric soft logic.

Most DNN accelerators struggle to leverage unstructured sparsity or arbitrary mixed precision for higher efficiency [21]. This is also true for hard blocks (DSPs and BRAM) that are available on modern FPGAs. For example, DSP units are often designed to optimize fused multiply-add operations that are 18, 27, or 32 bits wide [22], [23]. Despite many efforts to make FPGA hard blocks more reconfigurable [24]–[27], they are still fundamentally limited by their modes of operation, their fundamental multiply-accumulate operation, and their limited resource number on a given FPGA chip. In contrast, a soft-logic implementation of DNNs plays to the FPGA's strength of bit-level programmability by constructing exactly the circuit that is needed for an arithmetic operation. This includes the flexibility to optimize away ineffectual operations at the operand and bit levels [9], [18], and to utilize custom numerical formats [28], [29]. For instance, when synthesizing a multiplication operation where one operand is known at compile time, the operation can often be decomposed into a series of additions, allowing better utilization of FPGA adder chains. Furthermore, adder trees are important and common as they are needed to perform reduction in matrix multiplication.

Several prior works have explored arithmetic efficiency in FPGA soft logic. Compressor trees, constructed with generalized parallel counters (GPCs) [30]–[32], combine the use of both LUTs and dedicated adders to further boost resource utilization. Other approaches involve architectural changes to the logic block, such as packing more adders in one logic element [33] or including explicit XOR gates to improve compressor tree implementation [20]. However, current FPGA architectures present a fundamental limitation: the adder carry chain can only be driven by LUT outputs. Consequently, LUTs and adder chains must implement closely related logic to effectively utilize both resources. If a circuit is dominated by adder chains—as is often the case with matrix multiply reduction operations—the associated LUTs become unnecessarily occupied. Conversely, if LUTs implement unrelated logic functions, the adder chain remains inaccessible despite available inputs

* These authors contributed equally to this work.

and outputs within the very same logic block.

To address this shortcoming, we propose a modification to FPGA logic blocks to more flexibly enable the concurrent use of adder chains and LUTs. We fully implement our architecture—Double-Duty—in open-source FPGA CAD tools, from circuit and architectural modeling, to synthesis algorithms, to enable an extensive evaluation over three popular FPGA benchmark suites. Our results consistently show that the extra flexibility of enabling concurrent and independent use of LUTs and adder chains makes Double-Duty significantly more area-efficient compared to current FPGAs without compromising critical path delay. More concretely, we make the following contributions:

1) We propose the Double-Duty FPGA logic block architecture, which decouples the connections between LUTs and adders, enabling their independent and concurrent usage.
2) We quantify the area and critical path delay of the additional circuit components introduced by the Double-Duty architecture using SPICE-based simulation and transistor sizing with COFFE [34].
3) We integrate compressor tree algorithms into the open-source FPGA CAD tool Verilog-to-Routing (VTR) [35], significantly improving efficiency when synthesizing adder chains, ensuring a strong baseline on which to evaluate Double-Duty.
4) We evaluate our Double-Duty architecture on a suite of comprehensive benchmarks, including the VTR standard benchmarks, Koios ML benchmarks [36], and Kratos unrolled DNN benchmarks [18], achieving an average 9.7% improvement in area-delay product over all circuits, and up to 80% increase in packing density in stress tests.

II. Background

A. FPGA Basics

FPGAs can implement any logic function through programmable Lookup Tables (LUTs), organized into large clusters called Logic Blocks (LBs). Every LB contains multiple (typically 10) smaller units, called Adaptive Logic Modules (ALMs) or Fracturable Logic Elements (FLEs), connected to a programmable interconnect network via a local crossbar. Figure 2a shows the ALM design typical of Intel's Stratix 10 series FPGAs. Each ALM typically consists of four 4-input LUTs that can be combined with multiplexers to implement two 5-input truth tables or a single 6-input truth table. In addition to LUTs, each ALM contains two 1-bit full adders, whose inputs are directly connected to the output of 4-LUTs. This structure helps simplify logic before addition, and the LUT can be used in many addition algorithms [30], [32]. Moreover, the carry-in and carry-out signals are connected along multiple ALMs, forming a long carry chain, allowing fast and high-bit-width integer addition.

Despite this versatility, a key limitation of this conventional FPGA architecture is that the LUTs and adders are not independent. When the adders are in use, the LUT outputs provide inputs to the adder chain and thus cannot be used to implement other logic functions. This dependence limits the resource utilization in many arithmetic-heavy workloads. Some of the recent commercial FPGAs from Xilinx [37] have completely removed the 1-bit adders and instead use LUTs to generate carry propagate and generate signals, which further limits the ability of using arithmetic and logic resources independently.

B. CAD Tools

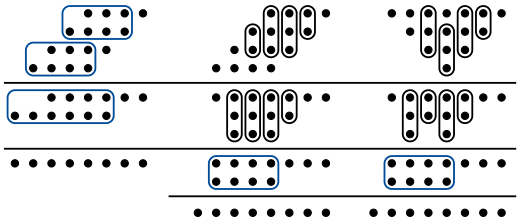
VTR Verilog-to-Routing (VTR) [35], [38] is an open-source FPGA CAD tool that takes Verilog design files with an FPGA architecture description file and performs synthesis, placement, routing, and timing analysis. By using a tree-structured XML architecture description file, VTR allows users to experiment with arbitrary FPGA architectures and quantitatively evaluate new architectures and CAD algorithms.

Parmys VTR has undergone several enhancements over time. Recently it changed the synthesis front-end from Odin II [39] to Yosys [40], a more flexible and advanced RTL synthesis tool that supports modern features like SystemVerilog generate statements. However, as the Yosys front-end is a general-purpose synthesis tool, VTR relies on a plugin called Parmys (Partial Mapper for Yosys) to handle FPGA-specific technology mapping [35]. Parmys has some significant limitations, particularly in the optimization of arithmetic operations including addition and multiplication. In this paper, we develop adder chain synthesis within Parmys to significantly improve its efficiency in synthesizing adder chains and compressor trees, to be able to investigate our proposed Double-Duty architecture using a strong CAD baseline.

COFFE 2 Circuit Optimization For FPGA Exploration (COFFE) [34] is an automated transistor-level modeling tool that provides accurate estimates of area, delay, and energy consumption for FPGA tiles, by utilizing HSPICE simulations and automated transistor sizing. Its successor, COFFE 2 [41], extends these capabilities to heterogeneous FPGA architectures, supporting complex logic blocks, fracturable LUTs, and custom DSP tiles. In this work, we use COFFE 2 to model our Double-Duty architecture for precise area and timing estimation.

C. Arithmetic Optimizations

Integer multiplications are the core of computation workloads, especially in machine learning. While DSPs are specialized units dedicated to multiplication, they are a limited resource and are usually better utilized with high bit width, and when both operands are unknown at compile time. However, a soft logic multiplier is still crucial to optimize resource utilization, particularly for custom and low-bitwidth arithmetic, and especially for unrolled DNNs [15], [17], [18] in which one of the operands is known at compile time—the DNN model parameters. This reduces a multiplication operation into multiple additions of partial products. For an n-bit by n-bit multiplication, n such rows are formed, which require n-1 summations. Using only full adders, an approach to reduce latency is to use binary adder trees, which perform as many summations as possible in parallel, requiring $O(\log n)$ time



(a) Cascade (b) Wallace (c) Dadda

Fig. 1: Conceptual diagram of a 4-bit multiplier using Cascade, Wallace, and Dadda algorithms. Pill shapes represent LUT-based compressors and rectangles represent full adder chains. Each layer represents a reduction stage.

to obtain the final result. This is currently the default and only approach taken by VTR to synthesize soft multipliers. To exploit the available LUTs in FPGAs, compressor trees [30]–[32] can also be used for efficient summation. Carry save logic, implemented with LUTs, is used to compress the initial n rows into 2 final rows that can be summed together with a fast ripple carry chain. Figure 1 illustrates the conceptual diagrams of two widely used algorithms, Wallace and Dadda, with Cascade—a simple algorithm that only uses adder chains—for comparison. Notably, compressor trees are currently not supported in Parmys/VTR.

III. Double-Duty Architecture and Circuit-Level Modeling

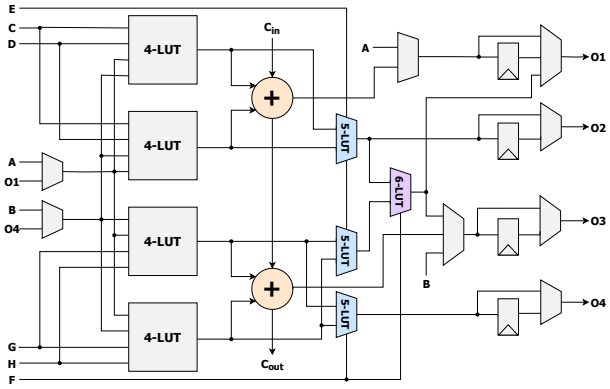
This section presents our new architecture, Double-Duty, that enables the concurrent use of logic block LUTs and adder chains, resulting in denser FPGA placement. We describe the architectural enhancements and variants, evaluate our proposal at the circuit level, and present the CAD synthesis changes that were needed to effectively and fairly assess our proposed architecture.

A. Double-Duty Logic Block Architecture

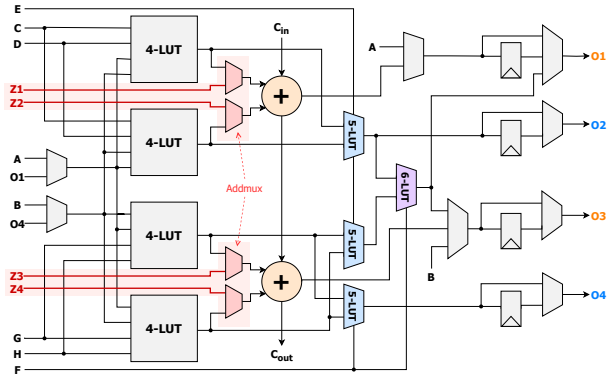
To enable independent and concurrent use of both LUTs and adders, we introduce two main changes to the ALMs and local routing as described below. Furthermore, we present two variants **DD5** and **DD6** which enable the use of 5-LUT and 6-LUT modes concurrently with the adders, respectively.

AddMux In each ALM, we add extra multiplexers and four additional ALM inputs (Z1–Z4), as highlighted in Figure 2b. These modifications allow inputs to bypass the LUTs and connect directly to the adder chain, enabling the use of adders and LUTs concurrently and independently. In this revised architecture, the ALMs support independent operation in 5-LUT mode: two output pins are allocated for adder outputs (O1 and O3), while the remaining two output pins can be used for the 5-LUT outputs (O2 and O4).

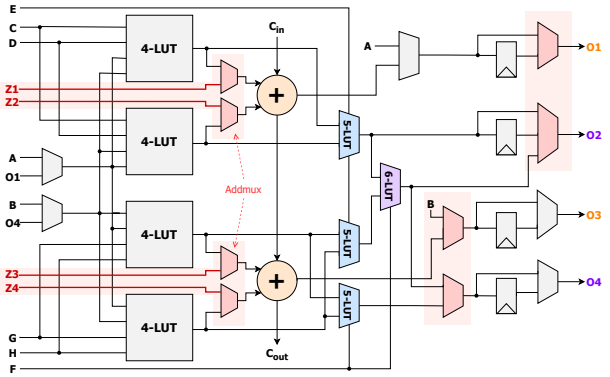
AddMux Crossbar To accommodate the additional ALM inputs (Z1–Z4), we introduce a secondary local interconnect, which is sparsely populated and draws inputs from the existing LB inputs as shown in Figure 3. Importantly, the total number of input pins to the LB remains unchanged—this modification leverages the same inputs of the original LB, and therefore



(a) Baseline Stratix-10 ALM.



(b) Double-Duty ALM. (DD5)



(c) Double-Duty ALM with concurrent LUT6 mode. (DD6)

Fig. 2: Baseline (a) and Double-Duty ALM architectures. We show two variants: the DD5 (b) supports concurrent 5-LUT and adders usage while the DD6 (c) supports concurrent 6-LUT and adder usage.

does not require any modifications to global routing. Notably, the wires that our new crossbar connects to have connections from LB-to-LB direct links [42]. Specifically, Stratix-10 architectures include 40 LB-to-LB wires, 20 from the east LB and 20 from the west LB. These enable direct connections between adjacent LBs without needing to access the global interconnect. As shown in Figure 3, these direct LB-to-LB

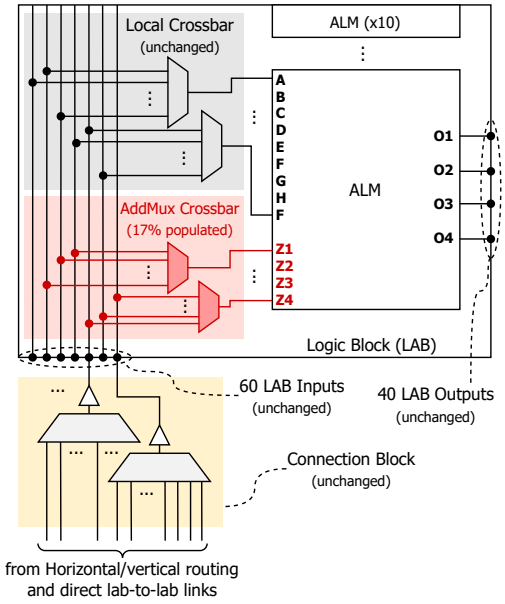


Fig. 3: A new AddMux Crossbar connects existing LB inputs to the new adder chain direct paths (Z1–Z4).

connections are multiplexed together with global routing into the connection block multiplexers. These feed 60 LB inputs, 40 of which can be connected to LB-to-LB inputs. Our new AddMux Crossbar connects to 10 out of those 40 LB inputs, making this new crossbar only 17% populated ($\frac{10}{60}$ inputs). In contrast, the existing local crossbar is more than 50% populated [42].

DD6 The AddMux and AddMux Crossbar allow the concurrent usage of 5-LUTs and adders; however, DD5 does not support concurrent 6-LUT usage with the adders. To address this, we propose an extended version, DD6, which modifies the multiplexing of LUT and adder signals to the outputs, enabling fully independent 6-LUT and adder operation, as shown in Figure 2c. However, our preliminary analysis indicates that the additional flexibility provided by DD6 yields marginal benefits in practice. A detailed evaluation of these trade-offs is presented in Section V.

B. Circuit-Level Modeling of Double-Duty

To model the delay and area overheads introduced by our architectural modifications, we build upon the framework established by Eldafrawy et al. [33], leveraging COFFE 2 [41] to replicate the Stratix-10-like baseline architecture used in their work. In particular, we create SPICE models for AddMux, as well as the additional AddMux Crossbar. COFFE 2 is then used to automatically size the transistors, and the resultant area and delay characteristics are used in the VTR model for the benchmark-level evaluations. Table I shows the area and delay costs of each added component. Overall, our modifications increase the tile area by 3.72% as compared to the baseline architecture. Table II shows the delay impact of the added components on several paths within the architecture. Although these multiplexers cause a LUT-to-adder delay increase, the delay of any signal feeding the adders directly is almost cut

TABLE I: Area and delay of added circuit components. The area data are all shown per ALM.

Circuit	Area (MWTAs)	Delay (ps)
AddMux	1.698	68.77
Baseline Crossbar	289.6	72.61
AddMux Crossbar	77.91	77.05
Baseline ALM	2,167.3	-
DD5 ALM	2,366.6 (**+3.72%**)	-

TABLE II: Delay impact of added circuits in the Double-Duty architecture variants on data paths.

Architecture	Path	Delay (ps)
Baseline	LB input → ALM inputs A-H ❶	72.61
	ALM inputs A-H → Adder input ❷	133.4
Double-Duty	LB input → ALM inputs Z_1-Z_4	77.05 (+6.11% vs. ❶)
	ALM inputs A-H → Adder input	202.2 (+51.6% vs. ❷)
	ALM inputs Z_1-Z_4 → Adder input	68.77 (-48.4% vs. ❷)

in half, as now the signal does not need to go through the LUT anymore. In addition, the delay of LUT output to ALM output remains almost the same as the output circuitry is largely unchanged. Note that Table II shows that the AddMux Crossbar has higher delay than the existing Local Crossbar. This may seem counterintuitive, as the AddMux Crossbar is much smaller; however, this additional delay is an artifact of transistor sizing in COFFE 2. COFFE 2 tries to optimize all the paths going through an ALM, and the Z ALM inputs have a much lower delay to the output of the ALM compared to the conventional LUT inputs; therefore, COFFE 2 aggressively optimizes the Local Crossbar for delay but the AddMux Crossbar can afford to be much slower and thus use smaller transistors.

IV. VTR CAD Enhancements

Soft multiplication synthesis in VTR hasn't been fully optimized, which results in redundant and/or wasteful resource usage. To tackle these shortcomings, we integrated several optimization techniques into VTR's synthesis. This ensured a strong CAD baseline in the evaluation of our Double-Duty architecture.

Unrolled Multiplication General multiplication treats each partial product as a unique signal, since its value is unknown during compilation. However, in the case of unrolled multiplication, the rows are shifted duplicates of the multiplicand, and are included in the summation if the corresponding bit of the known operand is '1'. We define this corresponding bit as the "selector bit" of the row. When forming partial sums from these rows, there can therefore be adder chains that have identical input signals but sum different rows. Instead of synthesizing duplicate adder chains, which is the current VTR behaviour, a single adder chain can be used instead, with its output signals fanned out as inputs for future adder chains. Multiplicand rows can also be excluded if their selector bits are '0', reducing the initial number of rows to sum. For a sample 8-bit multiplication with a $(01010101)_2$ constant,

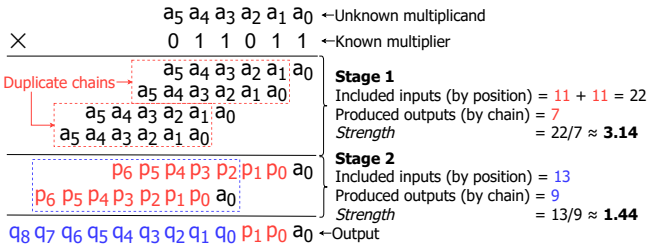


Fig. 4: A sample 6-bit by 6-bit unrolled multiplication. With known multipliers, it is possible to have duplicate adder chains in reduction stages. The *strength* of each reduction stage is computed with input signals unique *by position*, and output signals unique *by chain*.

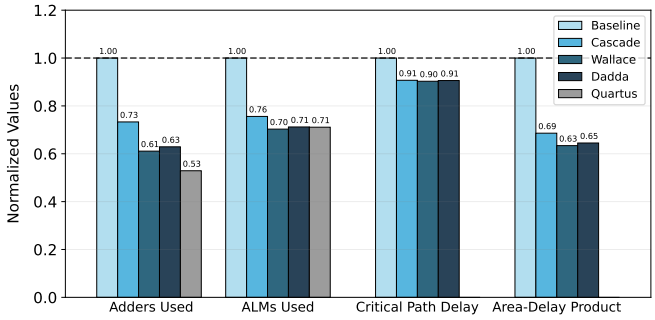


Fig. 5: Normalized adders, ALMs, critical path delay, and area-delay product (ADP) comparison between baseline VTR and our improved Cascade, Wallace, and Dadda adder synthesis algorithms on the Kratos benchmark set. Quartus area results are also displayed for comparison.

baseline VTR uses $2.85\times$ more full adders than in the optimal case which exploits adder chain redundancy.

Improved Binary Adder Tree Synthesis For n rows to sum, it is possible to insert $\lfloor n/2 \rfloor$ parallel adder chains at stage 1 of the addition operation, whose k rows of outputs form the next k rows to reduce in stage 2. Adder chains need not necessarily be formed from adjacent rows, which presents many possible adder chain combinations. To find the best combination at every stage, a *strength heuristic* is defined to reduce the ratio of included signals by the adder chains to the number of output signals generated in this stage. Included signals are counted by position, meaning that a duplicate input signal in different rows can be counted multiple times. The heuristic thus rewards adder chain placements with duplicate adder chains that can be replaced with a single adder chain. Figure 4 demonstrates this concept. To determine the optimal placement, a dynamic programming approach (Algorithm 1) is used to find a placement that gives maximum strength for each stage.

Compressor Tree Synthesis Instead of relying solely on adders, compressor trees implement reduction operations using both LUTs and adder chains. Current compressor tree implementations on FPGAs rely on generalized parallel counters (GPCs) specific to each architecture [32]. To implement compressor trees for any arbitrary architecture, each output signal of a compressor can be viewed as the result of a boolean equation of the GPC's input signals. After inserting compressors at each reduction stage, the final two rows are summed together with an adder chain. The intermediate combinational logic can then be optimized as part of logic synthesis, and then packed into LUTs. ABC synthesis [43] is responsible for this logic packing within the VTR suite, and thus only the initial boolean logic mapping from the soft multiplier's input signals to the final adder chain's input signals needs to be implemented in VTR. To achieve this, we make use of two compressor tree algorithms, the Proposed-Wallace (PW) and Dadda trees [44]. PW and Dadda have been empirically shown to minimize and maximize the number of full adders required in the final adder chain, respectively. Both of these trees use only full and half adders as compressors, for which the boolean relationships between their inputs and outputs are well-known. Whenever the algorithm requires the use of a compressor, it is inserted as its logically equivalent combinational logic, constructed with boolean gates. The final compression logic is thus a combinational circuit that can be optimized as a whole and then synthesized into LUTs by ABC.

CAD Improvement Validation After implementing the proposed synthesis algorithms and integrating them into VTR, we validated the changes using the Kratos benchmark suite. Figure 5 summarizes the geometric mean of adder usage, ALM usage, critical path delay, and area-delay product across various benchmarks and operand widths on the Stratix 10 FPGA. For reference, we also include the results of adder and ALM utilization from Intel Quartus Prime Pro 23.1 to compare resource efficiency with a commercial-grade tool. The goal is not to outperform Quartus which uses more advanced optimizations that we have not implemented. Instead, it is

Algorithm 1 Adder row selection for maximum strength.

$C_S \leftarrow$ empty collection ▷ empty solution cache
procedure BESTPLACEMENT(R_n) ▷ n-subset of rows
  if $n = 2$ **then return** R_n
  end if
  if $C_S(R_n)$ exists **then return** $C_S(R_n)$
  end if
  $S_{\text{best}} \leftarrow$ null ▷ best solution
  if n is even **then**
    for $p \leftarrow$ all pairs in R_n **do**
      $S_{R_{n-2}} \leftarrow$ BESTPLACEMENT($R_{n-2} \leftarrow R_n \backslash \{p\}$)
      $A_p \leftarrow$ adder chain with p as inputs
      $I_p, O_p, I_S, O_S \leftarrow$ input, output signals of A_p, $S_{R_{n-2}}$
      $I_S \leftarrow I_S + I_p$
      if A_p is not used in $S_{R_{n-2}}$ **then** $O_S \leftarrow O_S + O_p$
      end if
      if $H_S = I_S/O_S > H_{S_{\text{best}}}$ or S_{best} is null **then**
        $S_{\text{best}} \leftarrow S\{A_p, S_{R_{n-2}}\}$
      end if
    end for
  else if n is odd **then**
    for $r \leftarrow R_n$ **do** ▷ for each row
      $S_{R_{n-1}} \leftarrow$ BESTPLACEMENT($R_{n-1} \leftarrow R_n \backslash \{r\}$)
      if $H_{S_{R_{n-1}}} > H_{S_{\text{best}}}$ or S_{best} is null **then** $S_{\text{best}} \leftarrow S_{R_{n-1}}$
      end if
    end for
  end if
  Add S_{best} to C_S, **return** S_{best}
end procedure

TABLE III: Key statistics of our three benchmark suites, measured on the baseline Stratix 10 architecture with VTR.

Benchmark	Num. Circuits	ALMs ($\times 10^3$)		Adder Percent		Avg. Fmax (MHz)
		Avg.	Max	Avg.	Max	
VTR [45]	19	10.2	59.7	19.5%	47.7%	109.5
Koios [36]	20	64.3	360.7	22.5%	50.2%	70.9
Kratos [18]	7	59.6	208.2	61.4%	70.7%	103.7

to validate whether our CAD flow generates realistic FPGA resource utilization and is representative of a commercial FPGA CAD flow. We omit Quartus timing results as it uses a different timing model, and the open-source Stratix-10-like architecture in VTR does not provide accurate timing data. Overall, our updates to VTR improve upon the baseline by ∼37% in terms of area-delay product, and closely match Quartus in terms of resource utilization. This provides a solid foundation for fair and realistic comparisons in our end-to-end evaluations presented in the next sections.

V. Evaluation

A. Experimental Setup

VTR Setup The baseline FPGA architecture used for evaluation is modeled after Intel's Stratix 10 architecture, with a 20-nm technology node for delay and area estimation. This model is directly available from the Verilog-to-Routing (VTR) repository and was developed in prior work [33]. We build upon this Stratix-10-like architecture by modeling the AddMux, AddMux Crossbar, and different output multiplexing to get the DD5 and DD6 architectures. Timing models for these components are obtained from COFFE 2. The number of ALMs per LB is fixed at 10, as in the original architecture, and all other FPGA components remain unchanged. We fix the channel width at 400, and we set the VTR `target_ext_pin_util` to 0.9 for both inputs and outputs—this allows the VTR packer to use up to 90% of the inputs and outputs of a logic block. The adder synthesis algorithm is set to use "Wallace" as this gives the best results in all metrics, as shown in Figure 5. The placement and routing are all timing-driven, and we run every experiment three times with three different seeds and take the average to get representative results.

Benchmark Selection To assess the impact of our architectural modifications, we evaluate our designs across the three benchmark suites shown below. Table III also summarizes key information about each benchmark.

- Kratos: A specialized set of circuits designed for unrolled deep neural networks (DNNs) [18]. Kratos allows customized circuit sizes and varying computation sparsity levels, making it ideal for studying the effects of our optimizations. In this evaluation , we used the same size of the "small-size" set of benchmarks as in the original Kratos paper.
- Koios: A benchmark suite focusing on machine learning designs implemented on FPGAs [36]. This benchmark set helps to assess how well our architecture adapts to typical ML workloads beyond Kratos.
- VTR Standard: A general-purpose FPGA benchmark suite from the VTR repository [45]. This benchmark set ensures that our modifications do not degrade the versatility of the FPGA in broader application scenarios.

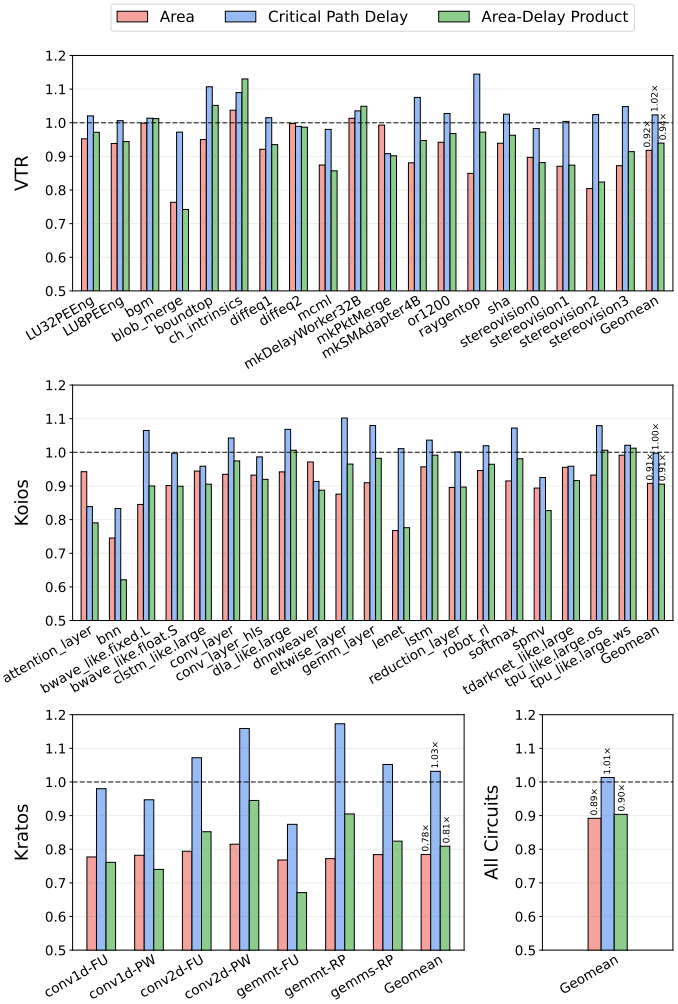


Fig. 6: Normalized ALM area usage, critical path delay, and area-delay product of Double-Duty DD5 Architecture, evaluated on the Koios, VTR, and Kratos benchmarks.

B. Experiment Results

Area and Delay Evaluation Figure 6 presents the normalized results of our proposed DD5 architecture compared to the baseline Stratix 10 architecture across the Koios, VTR, and Kratos benchmark suites. In terms of ALM area usage, DD5 consistently reduces resource consumption, achieving an average reduction of 10.9%. This improvement is particularly substantial in the Kratos benchmark suite, where arithmetic operations dominate, showing an average of 21.6% area savings. While the average of critical path delay remains the same level as the baseline, a few circuits do exhibit an increase of up to 16%. These cases are likely due to suboptimal packing, increased routing congestion, and routing heuristics that are not yet optimized for our modified architecture. Lastly, when considering the area-delay product (ADP), DD5 demonstrates an average improvement of 9.7%, indicating the effectiveness of our new Double-Duty Architecture.

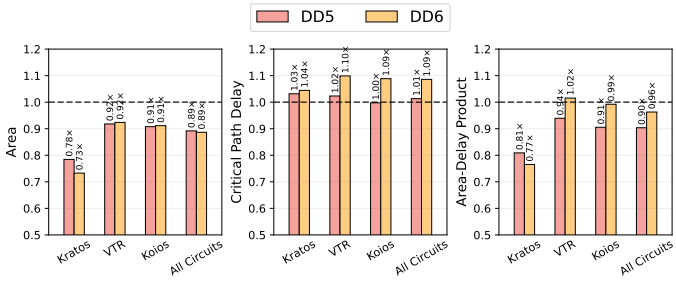


Fig. 7: Normalized ALM area used, critical path delay, and area-delay product (ADP) comparison between DD5 and DD6 architectures on the geometric means of 13 VTR standard benchmarks, 18 Koios benchmarks and 7 Kratos benchmarks set at data width of 6 and sparsity of 50%.

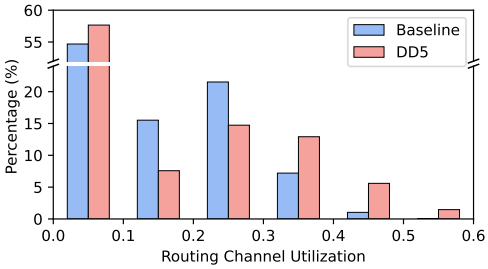


Fig. 8: Histogram of routing channel utilization averaged over the Kratos benchmark set. DD5 increases routing congestion on average as the diagram shows a shift of the histogram to higher channel utilization ranges.

DD5 vs. DD6 We additionally evaluate the second variant DD6, which allows concurrent and independent usage of 6-LUT and adders, and should bring more flexibility compared to DD5. Figure 7 shows the comparison between DD5 and DD6. While DD6 provides minor additional area savings in Kratos benchmarks, no noticeable gains are observed in Koios or VTR benchmarks. This can be attributed to two factors: (1) the current CAD tools are not optimized for architectures supporting concurrent 6-LUT usage, and (2) the utilization of 6-LUTs is inherently low. On average only 7% of ALMs across these benchmarks make use of 6-LUTs, making the opportunity for concurrent usage limited. Additionally, the added complexity of the output multiplexers in DD6 leads to higher critical path delays, with an average frequency penalty of approximately 8%. Consequently, the area-delay product also increases, suggesting that the added flexibility of DD6 does not offer practical benefits for general workloads.

Routing Congestion To analyze the routing impact of the proposed DD5 architecture, we examined the distribution of routing channel utilization using the Kratos benchmark set, which showed the largest area saving (22%), and the highest critical path delay penalty (3%) on average. The result is shown in Figure 8. While the portion of underutilized channels slightly increased, the majority of the channel utilization shifted towards a higher range, especially the portion between 0.3 and 0.6. This pattern suggests that by packing the same amount of logic into fewer logic blocks (LBs), routing channels become more congested, leading to denser usage. For all the circuits in these benchmarks, none failed to route, and the critical path delay impact is not too high, suggesting that this additional routing congestion is well within the range of what the FPGA's routing network can handle.

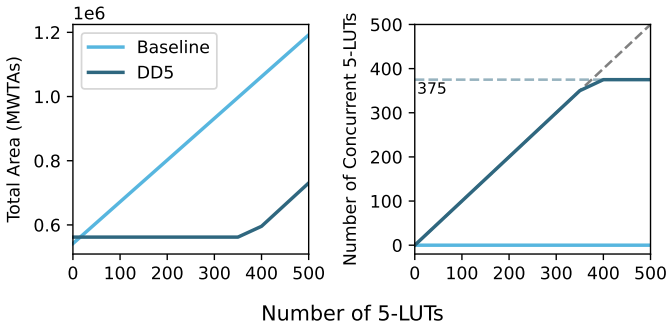


Fig. 9: Results of Packing Stress test. It starts with a circuit of 500 adders, and we pack an increasing number of LUTs into the circuit. The x-axis is the number of LUTs packed which saturates at 375 concurrent LUTs.

Packing Stress Test To quantify the limits of concurrent 5-LUT usage in DD5, we performed an *artificial* packing stress test. In this test, we constructed a synthetic circuit with 500 adders and incrementally added 5-LUT logic to the circuit, up to 500 LUTs in total. Under perfect concurrency conditions, this would be the maximum number of LUTs that could be absorbed into ALMs, concurrently with adders, without increasing utilization. We enabled the "allow unrelated clustering" option in VPR to encourage maximum packing density at the cost of ignoring timing, hence the "artificial" nature of this test. Figure 9 shows the results. The left plot presents total area usage in minimum-width transistor areas (MWTAs). When only adders are present, the baseline architecture has a slight area advantage since the DD5 architecture has a small area overhead in each ALM. However, when LUTs are added, the DD5 architecture shows a significant advantage as it can absorb LUTs into existing ALMs with adders, keeping the area the same until the ALMs are saturated. The right plot shows the number of concurrently packed 5-LUTs, which saturates at 375, corresponding to 75% of the theoretical maximum. This indicates that up to 75% concurrency between LUTs and adders is achievable, before becoming constrained by other factors like the global and local interconnect.

End-to-End Stress Test In this *realistic* stress test, we aim to evaluate how effectively the Double-Duty architecture improves packing density under constrained FPGA resources. The procedure is as follows: we implement a circuit from the Kratos benchmarks to determine the FPGA size needed for a successful implementation. Then, keeping this FPGA size fixed, we incrementally add instances of a small SHA circuit from the VTR standard benchmark suite, until VTR can no longer complete placement and routing. This is indeed how new FPGA architectures are stress-tested in industry [46]. We conducted this experiment using both the baseline architecture and the DD5 architecture on three different Kratos circuits, and

TABLE IV: Results of the End-to-End stress tests. The table compares the maximum number of additional SHA instances packed into a fixed-size FPGA for three Kratos circuits using both the baseline and the DD5 architecture. The DD5 architecture enables significant improvements in packing density and slight reductions in critical path delay.

Base Kratos Circuit	**conv1d-FU-mini**			**conv2d-FU-mini**			**gemmt-FU-mini**		
Architecture	**Base**	**DD5**	**Δ%**	**Base**	**DD5**	**Δ%**	**Base**	**DD5**	**Δ%**
Maximum SHA instances	5	9	+80.0%	3	5	+66.7%	11	13	+18.2%
Adders ($\times 10^3$)	37.55	38.79	+3.29%	18.86	19.48	+3.28%	14.35	14.97	+4.31%
5-LUTs ($\times 10^3$)	9.21	15.90	+72.5%	5.30	8.82	+66.5%	19.74	23.54	+19.2%
Concurrent 5-LUTs	0	4397	+27.7%*	0	2458	+27.9%*	0	3790	+16.1%*
Critical Path Delay (ns)	15.81	15.05	-4.81%	16.07	15.01	-6.60%	16.27	15.16	-6.82%
ALM Count ($\times 10^3$)	27.93	27.08	-3.04%	14.16	13.79	-2.64%	18.45	18.04	-2.27%
Logic Block Count	2947	2946	-0.03%	1474	1498	+1.63%	1895	1870	-1.32%
Total ALM Area (MWTAs $\times 10^6$)	60.54	60.88	+0.57%	30.69	30.99	+0.98%	40.00	40.55	+1.37%

*Denotes the percentage of 5-LUTs in which LUTs and adders are used concurrently. This is impossible in the baseline architecture.

all experiments are performed with timing-driven placement and routing to better reflect a realistic scenario. Table IV shows the results, indicating that DD5 enables significant improvements in packing density. For instance, the maximum number of SHA instances that can be implemented in the same FPGA area increased by 80% for `conv1d-FU-mini` and by 66.7% for `conv2d-FU-mini`, compared to the baseline. This is achieved through significant 5-LUT concurrency: 27.7% for `conv1d-FU-mini`, 27.8% for `conv2d-FU-mini`, and 16.1% for `gemmt-FU-mini`. Additionally, the critical path delay also shows slight improvements; this is because the critical path lies in the multiple adder region, and our AddMux in DD5 architecture reduced the delay from ALM input to the adder, resulting in a lower overall critical path delay. These results demonstrate that DD5 not only allows for denser packing but can also improve timing in certain stress scenarios.

VI. Discussion

Related Works There are two closely-related works targeting the arithmetic efficiency of the FPGA's soft logic. Eldafrawy et al. [33] proposed modifications to the ALM microarchitecture with an additional adder chain. However, their work's main focus is on low-bit multiply-accumulate (MAC) only, and lacks end-to-end evaluations on circuit benchmarks, making it challenging to compare directly to their work. Furthermore, our approach is synergistic with this prior work as we can bypass the LUTs to access one or more adder chains within the ALM. Similarly, LUXOR [20] achieves higher arithmetic efficiency of compressor trees by improving the expressiveness of each ALM through an extra XOR gate. This approach is also orthogonal to ours, as LUXOR focuses on gate-level function enhancement, whereas Double-Duty focuses on improving data path flexibility and resource utilization. Therefore, both enhancements can be applied to the same FPGA simultaneously.

Compatibility with Other Architectures Our architectural exploration and evaluation are based on an Intel Stratix-10-like FPGA, as its open-source architectural models are readily available [33]. However, modern FPGA families—such as the AMD Versal series [37]—feature different design paradigms: they use LUTs as full adders and include dedicated carry-lookahead logic, fundamentally different from the Stratix 10. To adapt the Double-Duty architecture to these FPGAs, several function unit augmentations would be necessary, including reintroducing explicit 1-bit full adders. While this requires some additional logic, the relative area cost of full adders in modern technologies is negligible and unlikely to cause substantial overhead. As a result, we believe our Double-Duty architecture is adaptable to modern devices with minimal design changes and without compromising its core efficiency benefits. We plan to more accurately compare to these alternative logic block architectures in future work.

VII. Conclusion

We presented Double-Duty, a novel FPGA logic block architecture that enables the concurrent and independent usage of adders and LUTs within the same ALM, significantly improving arithmetic density with minimal area overhead and a negligible impact on critical path delay. Together with the architectural innovations, we enhanced the VTR CAD toolchain [35] with optimized synthesis algorithms for adder chains, substantially improving its efficiency and providing us with a baseline that is representative of commercial CAD tools like Quartus. We evaluated Double-Duty across three diverse benchmark suites, achieving an average 9.7% improvement in area-delay product and up to 21.6% area savings on the arithmetic-heavy Kratos benchmarks [18]. Although denser packing may introduce routing pressure, we found the increase in routing congestion to be relatively low and well within the capabilities of modern FPGA routing fabrics. Lastly, two stress tests show the potential of Double-Duty in achieving higher density and resource utilization.

Acknowledgment

This project is supported by Intel Corporation, the National Science Foundation under Grant No. 2303626, and the CN Yang Scholars Programme, Nanyang Technological University. We would like to thank Sergey Gribok, Ilya Ganusov, Martin Langhammer, Sadegh Yazdanshenas, and Susanne Balle for discussion and feedback.

REFERENCES

[1] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung *et al.*, "Azure accelerated networking: SmartNICs in the public cloud," in *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[2] N. Tarafdar, T. Lin, E. Fukuda, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "Enabling flexible network FPGA clusters in a heterogeneous cloud data center," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017.

[3] J. Weerasinghe, R. Polig, F. Abel, and C. Hagleitner, "Network-attached FPGAs for data center applications," in *2016 International Conference on Field-Programmable Technology (FPT)*, 2016.

[4] T. Havinga, X. Jiao, W. Liu, and I. Moerman, "Accelerating FPGA-based WI-FI transceiver design and prototyping by high-level synthesis," in *Proceedings of the 31st IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2023.

[5] G. Li, S. Wu, C. You, W. Zhang, G. Shang, and X. Zhou, "Cell-free massive MIMO-OFDM: Asynchronous reception and performance analysis," *IEEE Internet of Things Journal*, vol. 11, no. 7, 2023.

[6] M. W. Hassan, H. Ltaief, and S. A. Fahmy, "High throughput massive MIMO signal decoding using multi-level tree search on FPGAs," in *Proceedings of the 32nd IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2024.

[7] M. S. Abdelfattah, D. Han, A. Bitar, R. DiCecco, S. O'Connell, N. Shanker, J. Chu, I. Prins, J. Fender, A. C. Ling *et al.*, "DLA: Compiler and FPGA overlay for neural network inference acceleration," in *Proceedings of the 29th International Conference on Field programmable Logic and Applications (FPL)*, 2018.

[8] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang *et al.*, "Flightllm: Efficient large language model inference with a complete mapping flow on fpgas," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2024.

[9] E. Wang, J. J. Davis, G.-I. Stavrou, P. Y. Cheung, G. A. Constantinides, and M. Abdelfattah, "Logic shrinkage: Learned FPGA netlist sparsity for efficient neural network inference," in *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2022.

[10] Q. Wang, L. Zheng, Z. An, S. Xiong, R. Wang, Y. Huang, P. Yao, X. Liao, H. Jin, and J. Xue, "A scalable, efficient, and robust dynamic memory management library for HLS-based FPGAs," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2024.

[11] M. Langhammer, E. Nurvitadhi, B. Pasca, and S. Gribok, "Stratix 10 NX architecture and applications," in *Proceedings of the 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021.

[12] A. Arora, S. Mehta, V. Betz, and L. K. John, "Tensor slices to the rescue: Supercharging ML acceleration on FPGAs," in *Proceedings of the 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021.

[13] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer, "Xilinx adaptive compute acceleration platform: Versal architecture," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019.

[14] J. Duarte *et al.*, "Fast inference of deep neural networks in FPGAs for particle physics," *JINST*, vol. 13, no. 07, p. P07027, 2018.

[15] E. Wang, J. J. Davis, P. Y. K. Cheung, and G. A. Constantinides, "LUT-Net: Rethinking inference in FPGA soft logic," in *IEEE International Symposium on Field-Programmable Custom Computing Machines*, 2019.

[16] M. Andronic and G. A. Constantinides, "PolyLUT: Learning piecewise polynomials for ultra-low latency FPGA LUT-based inference," in *2023 International Conference on Field Programmable Technology (ICFPT)*, 2023, pp. 60–68.

[17] S. Tridgell, M. Kumm, M. Hardieck, D. Boland, D. Moss, P. Zipf, and P. H. W. Leong, "Unrolling ternary neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 4, oct 2019.

[18] X. Dai, Y. Chen, and M. S. Abdelfattah, "Kratos: An FPGA benchmark for unrolled DNNs with fine-grained sparsity and mixed precision," in *Proceedings of the 35th International Conference on Field programmable Logic and Applications (FPL)*, 2024.

[19] A. Boutros, M. Eldafrawy, S. Yazdanshenas, and V. Betz, "Math doesn't have to be hard: Logic block architectures to enhance low-precision multiply-accumulate on FPGAs," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019.

[20] S. Rasoulinezhad, Siddhartha, H. Zhou, L. Wang, D. Boland, and P. H. Leong, "LUXOR: An FPGA logic cell architecture for efficient compressor tree implementations," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2020.

[21] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, and J. S. Vetter, "Nvidia tensor core programmability, performance & precision," in *2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, 2018.

[22] Intel Corporation, *Intel Arria Native Fixed Point DSP IP Core User Guide: Functional Description*, Intel Corporation.

[23] AMD, *Versal Adaptive SoC Technical Reference Manual (AM011) Digital Signal Processor*, AMD, 2025.

[24] A. Boutros, S. Yazdanshenas, and V. Betz, "Embracing diversity: Enhanced DSP blocks for low-precision deep learning on FPGAs," in *Proceedings of the 29th International Conference on Field programmable Logic and Applications (FPL)*, 2018.

[25] A. Arora, A. Bhamburkar, A. Borda, T. Anand, R. Sehgal, B. Hanindhito, P.-E. Gaillardon, J. Kulkarni, and L. K. John, "CoMeFa: Deploying compute-in-memory on FPGAs for deep learning acceleration," in *Proceedings of the 30th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022.

[26] Y. Chen and M. S. Abdelfattah, "BRAMAC: Compute-in-BRAM architectures for multiply-accumulate on FPGAs," in *Proceedings of the 31st IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2023.

[27] Y. Chen, J. Dotzel, and M. S. Abdelfattah, "M4BRAM: Mixed-precision matrix-matrix multiplication in FPGA block RAMs," in *Proceedings of the 34th International Conference on Field programmable Logic and Applications (FPL)*, 2023.

[28] M. Langhammer, G. Baeckler, and S. Gribok, "Fractal synthesis: Invited tutorial," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 202–211.

[29] Z. Carmichael, H. F. Langroudi, C. Khazanov, J. Lillie, J. L. Gustafson, and D. Kudithipudi, "Deep positron: A deep neural network using the posit number system," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 1421–1426.

[30] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Efficient synthesis of compressor trees on FPGAs," in *2008 Asia and South Pacific Design Automation Conference*, 2008, pp. 138–143.

[31] M. Kumm and P. Zipf, "Pipelined compressor tree optimization using integer linear programming," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, 2014, pp. 1–8.

[32] K. Hoßfeld, H. J. Damsgaard, J. Nurmi, M. Blott, and T. B. Preußer, "High-efficiency compressor trees for latest AMD FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 17, no. 2, apr 2024.

[33] M. Eldafrawy, A. Boutros, S. Yazdanshenas, and V. Betz, "FPGA logic block architectures for efficient deep learning inference," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, no. 3, jun 2020.

[34] C. Chiasson and V. Betz, "COFFE: Fully-automated transistor sizing for FPGAs," in *International Conference on Field-Programmable Technology (FPT)*, 2013, pp. 34–41.

[35] M. A. Elgammal, A. Mohaghegh, S. G. Shahrouz, F. Mahmoudi, F. Koşar, K. Talaei, J. Fife, D. Khadivi, K. Murray, A. Boutros, K. B. Kent, J. Goeders, and V. Betz, "VTR 9: Open-source CAD for fabric and beyond FPGA architecture exploration," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, May 2025. [Online]. Available: https://doi.org/10.1145/3734798

[36] A. Arora, A. Boutros, S. A. Damghani, K. Mathur, V. Mohanty, T. Anand, M. A. Elgammal, K. B. Kent, V. Betz, and L. K. John, "Koios 2.0: Open-source deep learning benchmarks for FPGA architecture and CAD research," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 3895–3909, 2023.

[37] AMD, *Versal Adaptive SoC Configurable Logic Block Architecture Manual (AM005)*, AMD, 2024. [Online]. Available: https://docs.amd.com/r/en-US/am005-versal-clb/CLB-Architecture

[38] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. ElDafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, and V. Betz, "VTR 8: High performance CAD and customizable FPGA architecture modelling," *ACM Trans. Reconfigurable Technol. Syst.*, 2020.

[39] P. Jamieson, K. B. Kent, F. Gharibian, and L. Shannon, "Odin ii - an open-source verilog HDL synthesis tool for CAD research," *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 149–156, 2010.

[40] C. Wolf, "Yosys open synthesis suite," 2012. [Online]. Available: https://yosyshq.net/yosys/

[41] S. Yazdanshenas and V. Betz, "COFFE2: Automatic modelling and optimization of complex and heterogeneous FPGA architectures," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, pp. 3:1–3:27, January 2019.

[42] K. T. Khoozani, A. A. Dehkordi, and V. Betz, "Titan 2.0: Enabling open-source CAD evaluation with a modern architecture capture," in *Proceedings of the 34th International Conference on Field programmable Logic and Applications (FPL)*, 2023.

[43] R. Brayton and A. Mishchenko, "ABC: an academic industrial-strength verification tool," in *Proceedings of the 22nd International Conference on Computer Aided Verification*, ser. CAV'10. Berlin, Heidelberg: Springer-Verlag, 2010, p. 24–40.

[44] S. Asif and Y. Kong, "Low-area wallace multiplier," *VLSI Design*, vol. 2014, pp. 1–6, 05 2014.

[45] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson, "The VTR project: architecture and CAD for FPGAs from verilog to routing," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. New York, NY, USA: Association for Computing Machinery, 2012, p. 77–86.

[46] I. Ganusov, Altera Corp. San Jose, CA. Private communication. March 2025.