IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 14 September 2023, accepted 4 October 2023, date of publication 17 October 2023, date of current version 25 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3325313

RESEARCH ARTICLE

Programmable All-in-One 4×8-/2×16-/1×32-Bits Dual Mode Logic Multiplier in 16 nm FinFET With Semi-Automatic Flow

NETANEL SHAVIT[1], (Graduate Student Member, IEEE), INBAL STANGER[1], (Graduate Student Member, IEEE), RAMIRO TACO[2], (Member, IEEE), ALEXANDER FISH[1], (Member, IEEE), AND ITAMAR LEVI[1], (Member, IEEE)

[1]EnICs Laboratories, Faculty of Engineering, Bar-Ilan University, Ramat Gan 5290002, Israel
[2]Department of Computer Engineering, Modeling, Electronics and Systems, University of Calabria (UNICAL), 87036 Rende, Italy

Corresponding author: Netanel Shavit (netanel.shavit@biu.ac.il)

This work was supported in part by the Israel Innovation Authority in the Frame of the GenPro Consortium and the Israel Science Foundation under Grant 2511/20.

ABSTRACT In this paper, an improved multiplier architecture, utilizing dual mode logic (DML) targeting single-instruction-multiple-data (SIMD)-*like* systems is proposed. The design introduces improvements at both the architecture and logic gate levels, by capitalizing on their synergistic combination. At the architecture level, the multiplier design is adapted to accommodate diverse computations based on the level of the input data parallelism. The main novelty is the incorporation of three different acceleration or bypass mechanisms jointly. The configurable multiplier has three variable precision configuration options: a 32 × 32-bit, two 16 × 16-bit, and four 8 × 8-bit multipliers. This bypassing architecture seamlessly integrates DML logic, which supports two modes of operation: a high-performance dynamic mode and a low-energy consumption static mode, with smooth mode switching capabilities. By optimizing the DML mode based on the multiplier's bit-width, the design enhances active computational block utilization, overall performance, and energy efficiency. In the dynamic mode, the DML implementation achieves an average performance improvement of 15% for the 32-bit, 8% for the 16-bit, and 7% for the 8-bit multipliers compared to the CMOS implementation. In the static mode, the DML implementation demonstrates an average energy reduction of 28%. When running in combined mode, where the 32-bit multiplier operates in dynamic mode for acceleration and the 8-bit multiplier operates in static mode for energy savings, the DML implementation exhibits an average overall performance gain of 15% and up to 18% lower energy consumption. The non-trivial semi-automation flow utilized for the complex implementation of the proposed architecture is also presented.

INDEX TERMS Alternative logic family, automation flow, configurable multipliers, design flow, dual mode logic (DML), dynamic logic, logic gates, multiplier, single instruction multiple data (SIMD).

I. INTRODUCTION

Multiplication is one of the most widely used arithmetic operations in a variety of applications, such as machine learning [1], [2], [3], [4], [5], [6], [7], image/video processing [8], [9], [10], [11], [12], and cryptographic operations [12], [13], [14], [15], [16]. Multipliers often cause computational bottlenecks, so that optimizing their efficiency is crucial [11], [17], [18], [19]. Embedded systems and high-performance computing devices incorporate a diverse range of multipliers, which vary in terms of their bit-length, internal architecture, and operational principles. Specifically, we consider the realm of high-parallelism in current processors/embedded-systems

The associate editor coordinating the review of this manuscript and approving it for publication was Harikrishnan Ramiah.

© 2023 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

where memory-level-parallelism (MLP) and instruction-level-parallelism (ILP) exist in almost all processors. In these cases, depending on the code/application, multiplications of various lengths may be required as well as a variable number of parallel multiplications.

The ability to *configure* or program a single block to efficiently execute *different operations with varying levels of parallelism* is of utmost importance [2], [5], [6], [7], [9], [10], [13], [20]. Configurable multipliers' instances are good candidates for hardware utilization efficiency, improved overall data-throughput, performance, and area reduction.

Dual mode logic (DML) is a circuit-level technique that endows logic gates with configurable characteristics [21]. These gates can operate in one *mode* which is more efficient in terms of performance (speed), termed the dynamic mode, and another *mode*, static, which allows for low-energy operation. Both modes trade off energy or performance to achieve their goal. The specificity of the DML feature is that this configurability can be performed generally per-gate and on-the-fly. This DML configurability is orthogonal to architectural improvements and provides additional value. This article illustrates a paradigm where our configurable and **parallel** multipliers leverage DML dual-modularity and architectural-level solutions to optimize overall system performance.

The goal of this paper is to present an enhanced multiplier architecture that specifically targets single-instruction-multiple-data (SIMD)-like systems. The multiplier has three configuration options, allowing for the selection of a 32 × 32-bit multiplier, two 16 × 16-bit multipliers, or four 8 × 8-bit multipliers. This architecture is implemented using DML logic and takes advantage of the unique DML capabilities that give it additional operational versatility. By dynamically adjusting the DML operation mode based on the multiplier's bit width, which affects information flow in the architecture, the design optimizes the utilization of active computational blocks, enhances performance, and reduces energy consumption.

Several DML-based multipliers can be found in the literature. In previous studies, such as [22] and [23], conventional multipliers without configurability were implemented, with specific configurations of 8 × 8-bit and 16 × 16-bit operands respectively. Configurable DML-based multipliers have also been reported [24], [25] and are discussed in detail later in this paper. In brief, [24] presented a DML-based tiny multiplier (8-bit operands maximum) with some level of configurability. However, this solution targeted small (number of bits) multipliers which are far simpler, and are designed for neural-network acceleration. Given the simplicity of this tiny multiplier, many architectural advantages and acceleration techniques are not applicable. In [25], the authors proposed a medium-size multiplier for 16-bit operands, that also harnessed DML efficiency and produced excellent results. However, their multiplier did not contain compression circuits, and partial product (PP) summation was performed per element. The absence of such architectural partitioning limits the modularity or acceleration possibilities in different stages of the computation. *In contrast, our work showcases larger granularity in acceleration with multiple SIMD-like resolutions, along with various bypass acceleration modes. These modes work in conjunction with the three different operation modes of the DML multiplier, offering enhanced flexibility and performance.*

We present a significant advance by the achievement of a larger bit-width in DML-based multipliers, handling a 32 × 32-bit multiplication for the first time. This milestone derives from the utilization of a semi-automatic tool, as detailed below, that was developed specifically to enhance the scalability of DML designs. In contrast to previous DML studies that have relied on custom designs and implementation, which impose limitations on device sizing and design scale-up, our approach overcomes these constraints. The proposed multiplier was implemented and fabricated using advanced 16 nm FinFET technology. The findings presented in this paper are based on the analysis of the resulting silicon implementation.

The proposed multiplier evidences significant improvements. For low energy consumption operation of the system, the multiplier is used in its static mode, to achieve an energy reduction of 27%,28%, and 29%, for the 32-bit, 16-bit, and 8-bit multipliers respectively. Boost acceleration of the system period, up to 15% higher performance, is achieved by operating in the DML dynamic mode. The DML mix-mode, which adapts the modes of DML operation to the multiplier configuration (i.e. the 32-bit multiplier operates in dynamic mode to boost acceleration, whereas the 8-bit multiplier operates in static mode to enhance energy efficiency), presents an improvement of 15% and 18% in both performance and energy, respectively. These improvements highlight the effectiveness and versatility of the DML approach in optimizing both performance and energy efficiency.

The remainder of this paper is organized as follows: Section II provides an introduction to DML technology. The architecture of the proposed configurable multiplier is presented in detail in Section III. Section IV describes the semi-automation flow for the DML implementation. The measurement and analysis results are reported in Section V. Section VI concludes the paper, and summarizes the key findings.

II. BACKGROUND

In this section, we briefly review literature related to the DML and we follow with discussing previously proposed DML-based multipliers architectures.

A. DUAL MODE LOGIC

DML is an alternative logic family to the dominant CMOS logic. Over the past 15 years, extensive research has been conducted to better understand and explore the potential advantages of DML in various architectures and technology nodes. The foundational work on DML is discussed

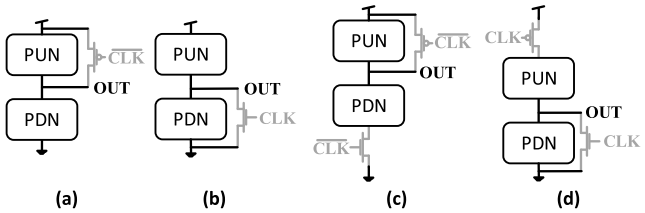


FIGURE 1. Basic DML gates topology (a) A-type (b) B-type (c) A-footer-type (d) B-header-type.

in [21], [26], and [27]. Research focusing on the implementation of DML in scaled nodes has been presented in [22] and [28], whereas the capability of DML to operate effectively in 16 nm FinFET technology, proving its unique dual-modality design intent, can be found in [23] and [24].

The benefits of utilizing DML gates and design techniques stem from their ability to operate in two distinct modes, offering flexibility and adaptability to meet specific system requirements, workloads, and energy budgets. The first DML operation mode is the *static* mode, which resembles conventional static logic families like CMOS. In this mode, the gate operates in a similar manner, consuming less power but resulting in a slower frequency due to the unique transistor sizing employed in DML. The second mode is the *dynamic* mode, which works in two phases, as seen in other dynamic logic families [29], [30]. During the first phase, the pre-charge, the gate output is pre-charged, regardless of its correct logic value. In the second phase, the evaluation, the gate output either discharges or retains its level based on the logic function. The dynamic mode allows for faster operation at the expense of higher power consumption. It is worth noting that DML achieves these features while minimizing the impact on area, and in many cases, by even providing area savings. This is made possible through the special transistor sizing techniques employed in DML, which strike a balance between power consumption, frequency, and area utilization [31].

The architecture of a DML gate (Fig. 1) is based on a standard CMOS gate, which consists of two complementary pull-up and pull-down networks (PUN/PDN). In addition to these networks, the DML gate includes an extra transistor connected in parallel to one of the networks, with its gate connected to a clocked signal (*CLK*). There are two main types of DML gates: the A-type (Fig. 1(a)) and the B-type (Fig. 1(b)). In A-type gates, the additional transistor is connected in parallel to the PUN, whereas in B-type gates, it is connected in parallel to the PDN. The purpose of this additional transistor is to enable the dynamic mode of operation in DML. To disable the dynamic operation in the static mode, the *CLK* signal is grounded, to disable the operation of the additional transistor. Note that the A-type gates and the B-type gates are connected to complementary signals, according to their structure. In the dynamic mode, the gate evaluation occurs through only one of the pull-up or pull-down networks, which determines the gate delay. This allows for optimizing the critical delay network for timing by sizing up the transistors accordingly, while the other network can be sized for efficient energy consumption.

The utilization of different types of DML gates has certain implications in terms of their arrangement and operation. To ensure a proper pre-charge phase, the output of each gate needs to be connected to the opposite gate type. This is done by arranging the gates in an alternating fashion, such as A-type followed by B-type, and so on. By connecting the gate outputs in this manner, the output of the first pre-charged gate closes the anti-pre-charged network of the next gate, which serves as the evaluation network. This arrangement prevents conflicts and enables a successful pre-charge phase. During the evaluation phase, all the pre-charge devices are turned off, and the evaluation signal propagates through the logic, similar to the operation of Domino logic [29]. In addition, in the design of DML circuits, there is an aim to maximize the number of gates whose evaluation networks are predominantly composed of parallel-connected devices, as discussed in [21] and [31]. This configuration allows for high-performance evaluation with less area. That is because the sizing of devices in the evaluation network is done in a specialized manner to ensure optimal performance during the evaluation phase of operation, and it needs to be larger in serial-connected transistors.

DML differs from other dynamic logic families in terms of the availability of the static mode. This mode is facilitated by the presence of full PUNs and PDNs that operate in parallel to the pre-charge devices. These networks are not involved in the dynamic evaluation process and are specifically included to enable functional static operation. During the static mode, the emphasis is on energy reduction rather than performance. As a result, all the transistors that are not involved in the dynamic operation, whether the PUN in the A-type gates or the PDN in the B-type gates, can be sized to a minimum, thus significantly reducing energy consumption. On the other hand, in the dynamic mode, the minimal sizing of the static mode network actually contributes to achieving high performance. This is because of the minimized input and output capacitance and increased resistance of these networks, which reduced interaction with the complementary evaluation network. All these lead to improved performance and reduced competition between the two networks.

The DML gates exist in other versions, known as A-footer-type (Fig. 1(c)) and B-header-type (Fig. 1(d)) gates. These footer and header gates play a crucial role in the seamless connection between non-DML and DML domains. During pre-charge, the evaluation network is disconnected in these gates, ensuring a straightforward pre-charge process. These footer and header gates also serve as separators between two similar-type gates when such concatenation is necessary. Their presence allows for efficient organization and arrangement of the gates, ensuring proper functionality and avoiding conflicts.

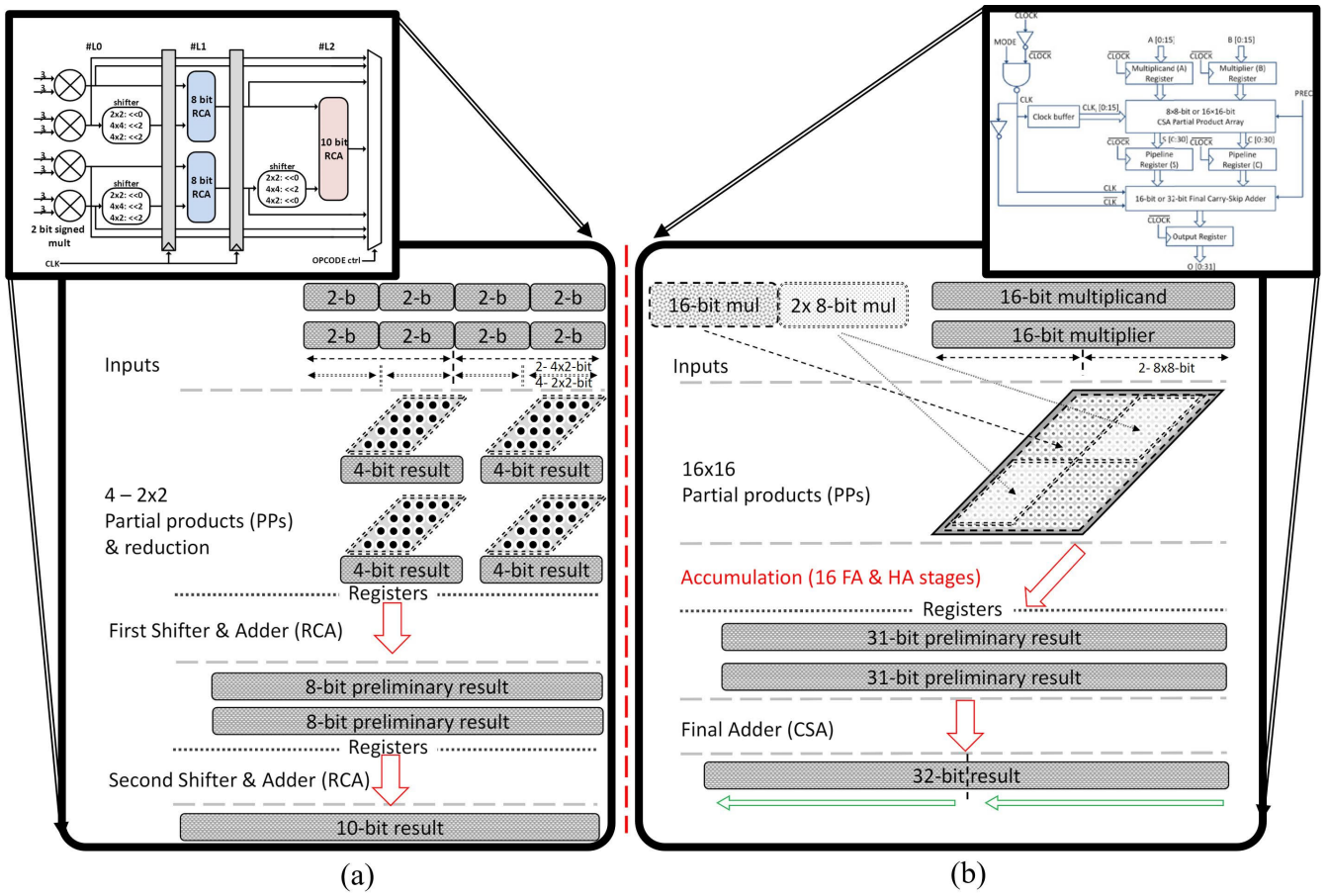


FIGURE 2. Illustrated architecture of previous DML configurable multipliers in two view versions: the original figure (in the small box), and the compatible regenerated abstract architecture. (a) oriented for AI acceleration - FlexDML tiny architecture [24], and (b) Double-prediction carry-save adder-based array multiplier [25].

B. PREVIOUS DML MULTIPLIERS

Numerous architectural solutions have been presented in the literature that aim to enhance the efficiency of DML in different arithmetic circuits. These solutions focus on optimizing energy consumption and improving overall performance by employing various mode-control granularity tactics. These works present a range of strategies and techniques that enable fine-tuning of the system's energy-delay requirements, hence allowing for better customization and optimization of DML-based designs.

In the field of multipliers, few studies on DML multipliers have already been presented, and they are different from each other by the technology nodes, bit-length, architecture, and configurable features. In [22] a 8 × 8-bit multiplier was presented. It was implemented in 28 nm FD-SOI technology, with column-bypassing PP reduction tree architecture. The work in [23] demonstrated a 16 × 16-bit multiplier, in 16 nm FinFET technology, based on radix-4 booth-encoded Wallace tree architecture. Both of them did not have configurability features.

Configurable DML-based multipliers have also been reported. In [24], a tiny multiplier (4×4-bit) was implemented in the 16 nm FinFET technology, in a unique architecture directed towards neural-network acceleration with a high level of flexibility. The computation starts by processing with small building blocks over small operands (i.e., 2 × 2-bit multiplier), then according to the configuration, escalates to outputs that depend on all or a partial set of the operands, e.g., results can be added or simply broadcast as outputs in accordance with the configuration. It was implemented in 3 pipeline stages, each stage had the ability to be configured to the static or dynamic mode of DML, allowing high flexibility to attend to the changeable requirements of the system. Fig. 2(a) illustrates the architecture of this work in two views: first the original figure from [24] in a small version, and second *abstract-view* version. This version was designed with the aim to adapt the architecture to a multiplier structure of inputs–PP–reduction–result, for better comparison ability with the proposed multiplier in this article. In this version, it can be seen that the multiplier in [24] has 8-bit operand inputs, PP and reduction of each 2 × 2-bit separately, and final adder (of two stages of ripple carry adder) for the final result.

The multiplier in [25] is a DML configurable multiplier as well. It was implemented in the 65 nm technology, with carry-save adder-based array multiplier architecture, and in two

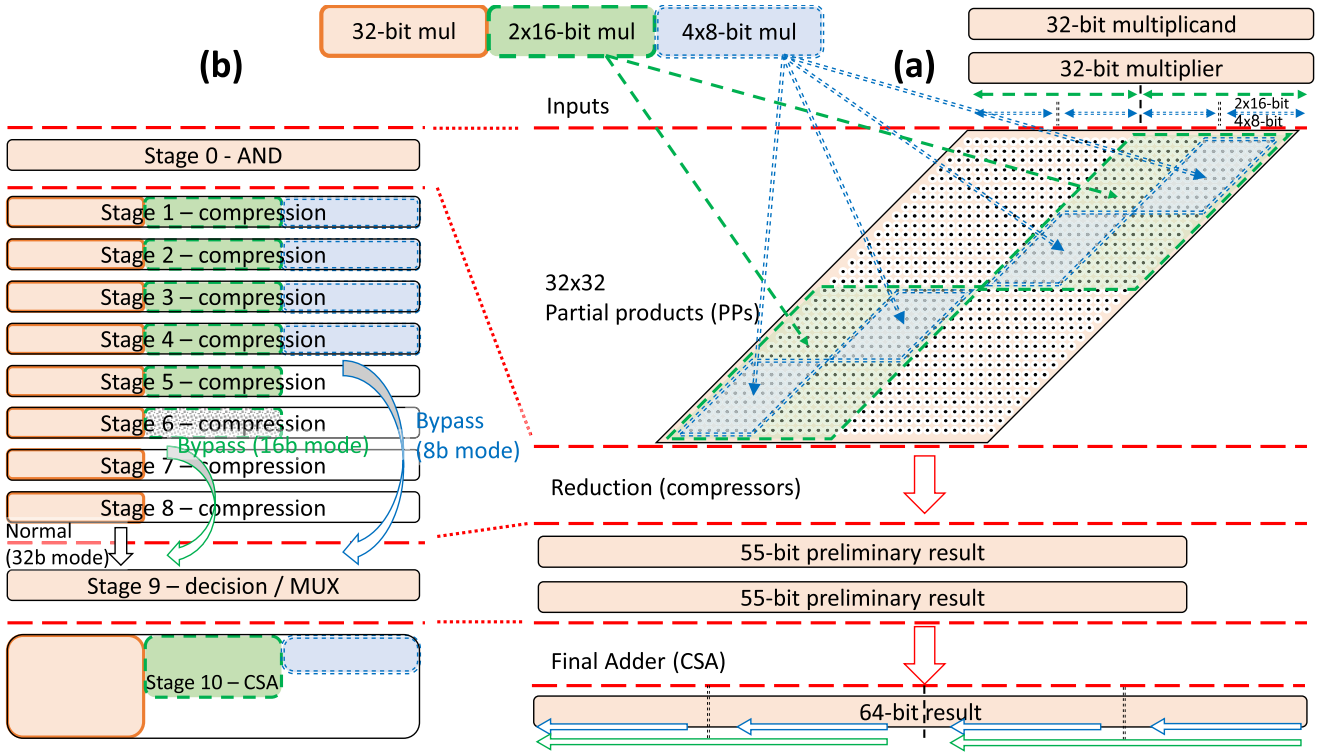


FIGURE 3. The proposed flexible and accelerated multiplier architecture: (a) data flow, and (b) hardware architecture.

pipeline stages. Its configurability has two precision levels: a 16 × 16-bit multiplier that works in the dynamic mode of DML, or an 8 × 8-bit multiplier that works in the static mode of DML. This architecture is also illustrated in Fig. 2(b), in similar two versions, the original and the regenerated. In the regenerated version, the 16-bit operand inputs are shown, followed by PP and accumulation of the data to two final vectors, and a final adder (of carry skip adder) for the final result.

The next section presents the architecture of the configurable multiplier implemented in this study. A similar figure to Fig. 2, with the same abstract illustration, is presented to create a better understanding.

III. THE CONFIGURABLE MULTIPLIER ARCHITECTURE

In this study, a configurable multiplier was implemented to support different modes of execution: a 32 × 32-bit multiplier (i.e., full-word), two 16 × 16-bit multipliers (two half-words), or four 8 × 8-bit multipliers (four quarter-words), *all on the same hardware*. The output of the multiplication is 64-bit wide, representing either a single 64-bit result, two 32-bit results, or four 16-bit results, depending on the selected mode. In comparison to a *non-configurable* 32-bit architecture, performing four 8-bit operations would require four sequential invocations of the 32-bit element. This results in significant idle time for the combinatorial logic, with approximately 95% of the logic remaining unused.

This section provides a comprehensive explanation of the architecture of the configurable multiplier. The overview highlights the primary advantages of the overall architecture and outlines its distinguishing factors in comparison to prior works in the field. It then provides a detailed examination of the underlying blocks including their specific details and functionalities.

A. WALLACE TREE ARCHITECTURE

The multiplier architecture is based on the Wallace tree multiplier [32], which has three main segments: **(1)** a parallel PP generator, **(2)** a compression/reduction stage, and **(3)** a final adder. The entire multiplier is constructed using combinatorial logic, from the input operands to the final output result.

The architecture of the **configurable** multiplier is presented in Fig. 3, where Fig. 3(a) illustrates the three fundamental segments of the Wallace tree architecture, and Fig. 3(b) provides a representation of the compatible hardware implementation.

The main novelty lies in its three distinct acceleration and bypass mechanisms within the same architecture. These mechanisms support variable precision operations, which in turn enable optimal area utilization of the *active* logic:

1) **Input Sequencing to PP Tree Parallelism:** The operands input sequence is dynamically parsed to handle different configurations, such as processing two 32-bit words, 4 × 16-bit words, or 8 × 8-bit words. This allows for the implementation of SIMD-like parallelism in the system where the multiplier is

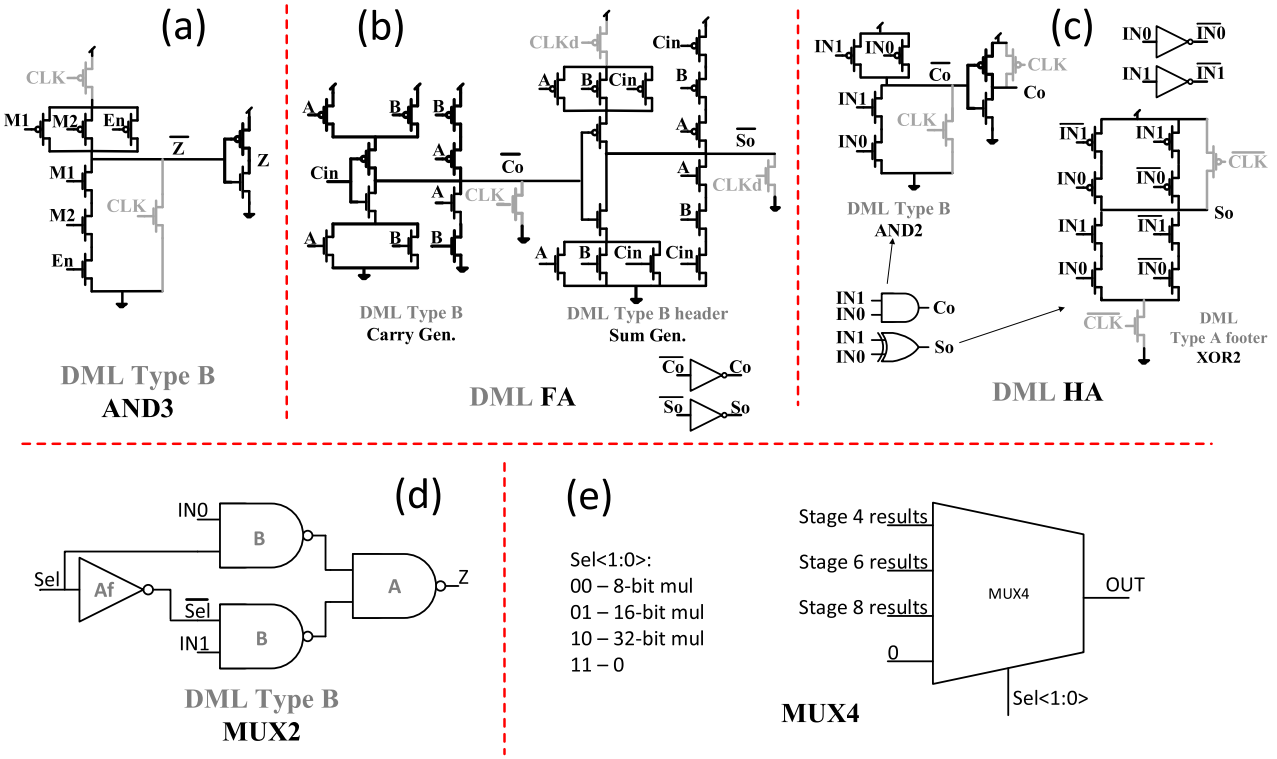


FIGURE 4. Schema of the gates (in BLACK the basic CMOS gate, in GRAY the additional transistors for the DML design): (a) AND3 (b) FA (c) HA (d) MUX2 (e) MUX4.

integrated. This parallelism can be leveraged in various systems, including multi-core processors, high memory level parallelism architectures, or ASIC designs. One of the key differences from previous works such as [24] and [25] is that they focused on smaller operations and had limited precision capabilities. Our implementation targets larger operations and offers three precision levels, as illustrated in Fig. 3.

2) **Accelerating the Compression Tree:** One of the main advantages and sources of significant performance saving is the inclusion of an accelerated compression tree (as elaborated below). The compression and reduction tree takes-up most of the computation effort reducing from (generally) 1024 PPs to (e.g.,) 9-bit vector; therefore, if embedded, architectural acceleration-tricks have more potential. This feature, not included in previous architectures, allows for gradually increased performance when operating on smaller operands in parallel. Specifically, two compression stages can be saved in the 16-bit mode, and four stages can be bypassed in the 8-bit mode. Note that the architecture in [25] (Fig. 2(b)) does not incorporate any compression, since PPs are directly accumulated. Therefore, implementing savings in their implementation and integrating bypasses in their accumulation is more challenging when DML circuits are embedded, given the various operation modes and gate-types involved.

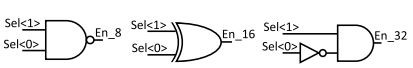
3) **Acceleration and Parallelism of the Final Long Adder:** The architecture chunks up the carry skip adder (CSA) into subchains. Specifically, in the 16-bit mode, two subchains operate in parallel, while in the 8-bit mode, four subchains are parallelized. This design decision enhances efficient utilization of the hardware and contributes to improved performance. A similar feature, albeit on a smaller scale, was also presented in [25] (Fig. 2(b)).

The following subsections provide a comprehensive explanation of the three segments comprising the Wallace Tree multiplier, explained in terms of their underlying concepts, data flow mechanisms, hardware structures, and hardware implementation through logic gates.

B. PARTIAL PRODUCT

The PP segment of the multiplier stretches out, in a parallelogram shape of weights, the 1024 1 × 1-bit multiplications between the 32-bit multiplicand and 32-bit multiplier, as depicted in Fig. 3(a). In the case of a 32-bit multiplication, each multiplicand and multiplier represents a single 32-bit number. For 16-bit multiplication, they represent two 16-bit numbers, and in the 8-bit multiplication mode, they represent four 8-bit numbers. In the 32-bit multiplication mode, all the PPs are utilized. However, in the 16-bit multiplication mode, only two 16 × 16 diagonal parallelograms are used, and in the 8-bit multiplication mode, only four

TABLE 1. Configuration table for the enable signals of the different PP parallelograms.

Configuration	Sel<1>	Sel<0>	8-bit parallelograms Enable	16-bit parallelograms Enable	32-bit parallelograms Enable
4 x 8-bit multipliers	0	0	1	0	0
2 x 16-bit multipliers	0	1	1	1	0
32-bit multiplier	1	0	1	1	1
Reset	1	1	0	0	0
Implementation					

8×8 diagonal parallelograms are employed. These distinctions are illustrated in Fig. 3(a) with different markers.

The hardware implementation of the PP segment is represented as stage 0 in Fig. 3(b). Each PP bit is generated by an AND3 gate, which takes inputs from the multiplicand, multiplier, and enable signal. This gate structure is depicted in Fig. 4(a). In the smaller multiplication modes, the unused portions of the PP segment are enabled to generate zeros, ensuring they do not impact the final result. The calculation mode is controlled by a 2-bit signal, which determines the enable signals, as illustrated in Table 1. In this table, "8-bit parallelograms enable" refers to the parallelograms used only in the 8-bit multiplication, whereas the "16-bit parallelograms enable" and the "32-bit parallelograms enable" mean the *overhead PPs* of the 16-bit (or the 32-bit) multiplications on the 8-bit (and the 16-bit) multiplications.

C. REDUCTION

The second segment of the multiplier involves the reduction of the PPs. This reduction transforms the 1024 PP bits into two 55-bit vectors that feed into the final adder, as well as a 9-bit vector that represents the final result.

The hardware implementation of the reduction process involves 8 stages of compression, followed by a decision stage in the ninth stage. In the case of the 32-bit multiplier, all 8 stages are utilized to generate the two final vectors. However, for the 16-bit and 8-bit multipliers, only 6 and 4 stages are required, respectively. As a result, these smaller multipliers enable a **bypass mechanism** that directs the compressed results to the ninth stage, as shown in Fig. 3(b).

In compression stages, 1-8, three types of gates are utilized. The primary components for the compression process are 3:2 compressors, implemented as Full Adders (FAs) as shown in Fig. 4(b). In addition, 2:2 compressors, implemented as Half Adders (HAs), shown in Fig. 4(c), are also employed. To prevent data leakage between sub-parallelograms in the 16-bit and 8-bit multipliers, a few multiplexers (MUX) are incorporated at the junctions. The MUX gates, illustrated in Fig. 4(d), are implemented using NAND2 gates, which are part of the HA gate depicted in Fig. 4(c).

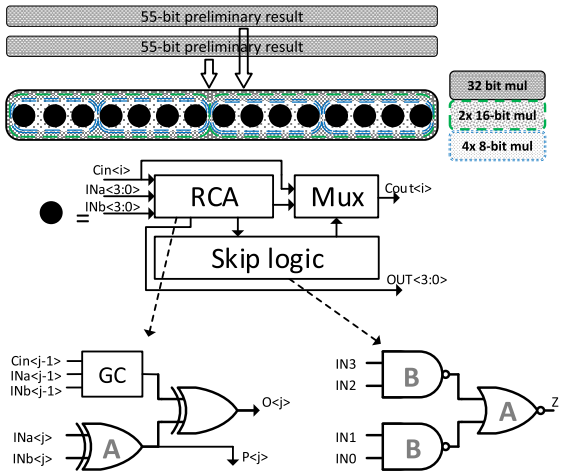


FIGURE 5. The final CSA architecture, which can be divided into 3 different length adders as a function of the multiplier operation.

In stage 9, a group of MUX4 gates (Fig. 4(e)) is employed to direct the appropriate vectors to the final adder. Each MUX4 gate selects the input according to the control signal bits, enabling the standard data flow for the 32-bit multiplier or choosing one of the bypass inputs for the smaller multipliers. The MUX4 gate is constructed using MUX2 gates, which are depicted in Fig. 4(d). This configuration ensures that the final adder receives the correct vectors for the addition in stage 10.

D. FINAL ADDER

The final adder takes two 55-bit vectors as inputs and computes the final result of the multiplier. It was implemented using a CSA, as presented in Fig. 5.

The CSA is constructed with 4-bit ripple carry adder (RCA) blocks. Each RCA block computes its carry-out and generates a selection bit for the skip logic. The skip logic determines whether the carry-in of the entire block should be propagated to the carry-out or not. A MUX gate is used to choose between the RCA carry-out and the carry-in of the RCA block, based on the selection bit. The carry only ripples through one 4-bit RCA block and the necessary MUX gates, depending on the parallelism of the specific calculation.

The CSA architecture is well-suited for the configurable multiplier given the presence of built-in MUX gates [25]. This enables partitioning between different regions of the partial multipliers and facilitates parallelization. In the CSA, the 32-bit multiplier utilizes 14 groups of RCA to perform the summation of 55-bit inputs, with 9-bits of the final result already generated. The 16-bit multiplier only generates 7-bits of the final result through the reduction process, thus requiring 7 RCA blocks for the first multiplier and 8 RCA blocks for the second. Similarly, the 8-bit multiplier produces 5 final result bits in the reduction, necessitating 3 RCA blocks for the first multiplier and 4 RCA blocks for the remaining 3 multipliers.

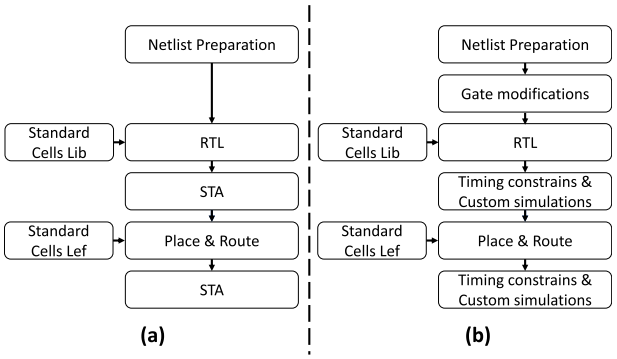


FIGURE 6. Implementation design flow for (a) CMOS (b) DML.

IV. DML CONFIGURABLE MULTIPLIER

The configurable multiplier was implemented twice, employing two different logic family gates: the conventional CMOS and the novel DML architecture. The following section outlines the step-by-step process involved in implementing the DML architecture in an automated flow, starting from the initial design idea and progressing to the final tape-out stage.

A. SEMI-AUTOMATIC FLOW

First, the standard automation flow of CMOS gates is described briefly, as a starting point to the DML flow.

The CMOS design flow is shown in Fig. 6(a). The multiplier design was implemented using a Python script [33], which automated the generation of the gate-level netlist. Synthesis was not used here because the Python generates a synthesizer's multiplier structure, and sizing is determined on the STA iterative step. This script produced gate-level netlists in both the Verilog and Spice formats. Using Cadence Innovus, the gate-level netlist served as the basis for generating the register transfer level (RTL) representation by CMOS standard cells, from the standard cell library *.lib*. Static Timing Analysis (STA), with iterative gate sizing optimization, was performed to validate the timing constraints of the RTL design, leveraging the information from the *.lib* file. Subsequently, place and route operations were executed using the CMOS standard cells library *.lef* to create a physical implementation of the design. After completing the physical implementation, STA was once again employed to verify the timing aspects of the physical layout. Finally, the design was ready for the tape-out process.

The DML automation flow adheres to the standard flow with certain modifications. These modifications are illustrated in Fig. 6(b), in comparison to standard CMOS flow, and are detailed in the following paragraphs.

In the gate-level netlist preparation step, the critical constraint in the DML design involves the specific concatenation requirement of the cell types: A-type – B-type – A-type – B-type (and vice-versa), as detailed in Section II-A. To seamlessly integrate the DML into the automated flow, the Python script, running the multiplier design, was partitioned into cells consisting of gates arranged in a B-type – A-type – B-type – A-type sequence. Then, the CMOS cells were smoothly substituted by the DML cells, while maintaining the architecture netlist identical to the CMOS design. The next subsection explores the internal gate-level modifications implemented to adapt the DML cells to the prescribed structure.

The next step involves generating an RTL representation using a standard cell *.lib* file, followed by performing STA to verify its timing. However, while creating a DML *.lef* file is feasible, the *.lib* file is not compatible with the two modes of DML operation. To address this challenge, a degenerated *.lib* file was employed, without timing constraints for individual cells. This decision was justified because the macros are full mixed-signal blocks, where timing verification is extensively conducted in an analog environment (e.g., Virtuoso simulations) rather than relying on automatic STA. Iterations involving modifications to the standard cell RTL and timing simulations are performed until the design satisfies the constraints. The automated generation of the circuitry representation netlist is a significant challenge for DML, but is effectively tackled by this automation approach.

Then the automation flow proceeds with the automatic place and route step, utilizing the DML standard cell *.lef* file. The generation of the DML *.lef* file was carried out in coordination with the CMOS standard cell *.lef* files. After the place and route step, comprehensive timing simulations were conducted in the full-custom domain to validate the final design outcome before proceeding with the tape-out process.

B. GATE-LEVEL ADAPTATIONS

As mentioned in the preceding subsection, the DML gates were structured as cells consisting of paired gates arranged in a predetermined sequence from B-type to A-type. *This design approach inherently ensures the proper cascading functionality*. This subsection describes the adaptations at each stage of the multiplier's hardware implementation, depicted in Fig. 3(b), to ensure compatibility with the cell structure.

The initial stage, referred to as stage 0, incorporates an AND3 gate design comprising a NAND3-DML-B-header gate combined with an INVERTER gate. The AND3 DML gate structure is depicted in Fig. 4(a), which shows the additional transistors (in gray) alongside the standard CMOS cell (in black) to facilitate DML pre-discharge.

The next stages, referred to as stages 1-8, incorporate FA (see Fig. 4(b)), HA (Fig. 4(c)), and MUX (Fig. 4(d)) gates. The primary modification challenge was encountered in the FA gates, which were implemented using a mirror FA architecture. The FA gate comprises three logic levels in its *sum* output: *Cout-not*, *Sum-not*, and *Sum*. To extract an A-type output from the cell (*Cout* and *Sum*), both *Cout-not* and *Sum-not* gates must be of B-type. However, this conflicts with the requirement of *Cout-not* to be an input to the *Sum-not* level, with different gate types. This presents a

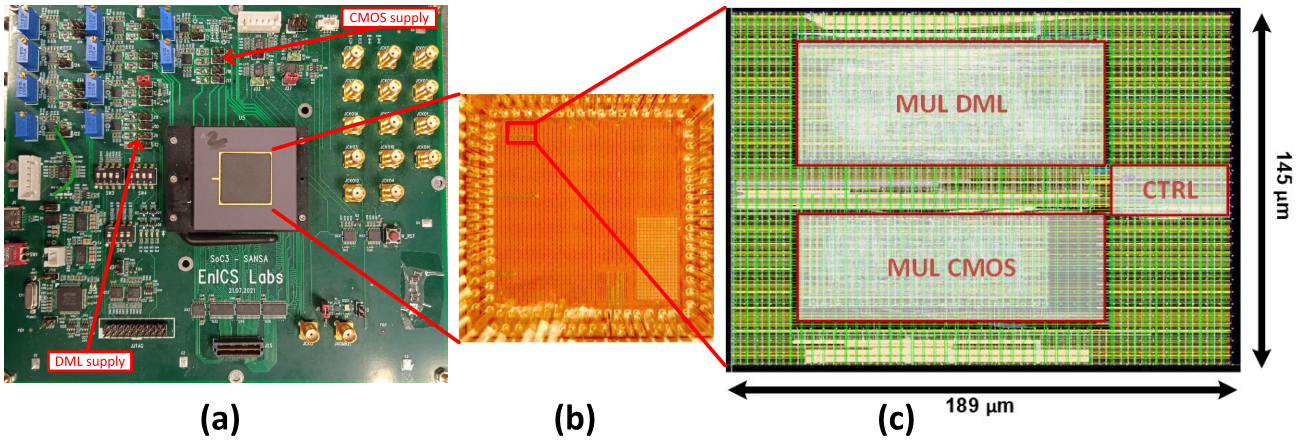


FIGURE 7. (a) The measurement board, (b) the chip micrograph, and (c) the final layout of the CMOS and the DML multipliers.

trapped inverter challenge, as described in previous works such as [34]. In this implementation, the challenge was overcome by utilizing the header features. The FA gate was structured as B-type – B-header-type – A-type, as depicted in Fig. 4(b), allowing seamless integration into the overall flow. This FA structure requires two clock stages, where the *CLKd* signal is delayed from the *CLK* signal, ensuring that the headed gate is delayed to remain in the pre-charge phase until all its inputs are stable. Notably, in previous works [22], [23], [25], the mirror-FA was implemented in an A-type – A-footer-type – B-type structure, which does not meet our concatenation requirements.

Within stages 1-8, the HA, illustrated in Fig. 4(c), is composed of an AND2 gate (comprising a NAND2-B-type gate and an INVERTER-A-type gate) and a XOR2 gate. The XOR gate presents a challenge since its inputs involve two signals and their inverses, leading to the recurring challenge of the trapped inverter. To resolve this problem the XOR gate was implemented as an A-footer-type gate.

The final gate in stages 1-8 is the MUX gate (shown in Fig. 4(d)), constructed by concatenating NAND2 gates. This configuration resembles the NAND2 DML gate discussed in detail for the HA gate (depicted in Fig. 4(c)), where the first NAND gate is the B-type and the second NAND gate is the A-type. In stages 9 and 10, as depicted in Fig. 4(e) and Fig. 5, all the components are constructed using the previously described gates.

V. MEASUREMENTS AND RESULTS

The multipliers were fabricated using TSMC 16 nm FinFET technology. The measurement board with different voltage supplies for the two multipliers is depicted in Fig. 7(a). The physical chip is captured in the micrograph displayed in Fig. 7(b). In Fig. 7(c) the layout of both the CMOS and DML multipliers is presented.

The multipliers underwent extensive simulation and measurement processes to evaluate their worst-case timing

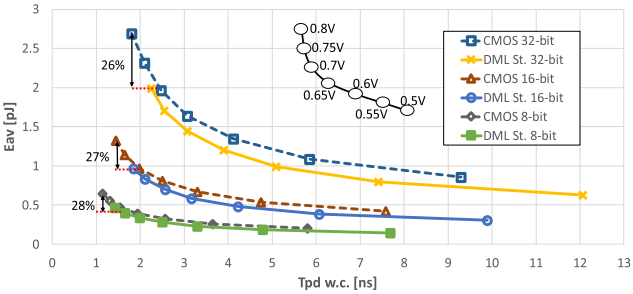


FIGURE 8. Static DML vs. CMOS Tpd and Energy Results.

propagation delay (Tpd) and average energy consumption (Eav). These evaluations were performed for different multiplier configurations and various voltage levels ranging from 800mV to 500mV.

Fig. 8 compares the CMOS multiplier to the static mode operation of the DML multiplier. This comparison highlights the well-known characteristics of the DML static mode, which exhibits energy consumption improvement at the expense of performance degradation. Specifically, the average energy improvement achieved by the DML static mode, for a range of supply voltages, is 27%, 28%, and 29% for the 32-bit, 16-bit, and 8-bit multipliers, respectively. However, the static DML mode incurs a delay overhead of 25%, 28%, and 29% on average for the corresponding multiplier configurations. A similar comparison of the dynamic mode of the DML multiplier and the CMOS multiplier is presented in Fig. 9. As anticipated, the DML dynamic mode demonstrates improvements in Tpd accompanied by an increase in energy consumption. On average, the DML dynamic mode achieves a Tpd improvement of 15% for the 32-bit multiplier, 8% for the 16-bit multiplier, and 7% for the 8-bit multiplier. It is well-known [28] that DML's advantages become more pronounced with deeper logic depth. Clearly, if only Tpd is important one might think that it is therefore worth-while to always operate in the 32-bit mode even with 8-bits operands. However, owing to parallelism the computation power reduces in a factor of

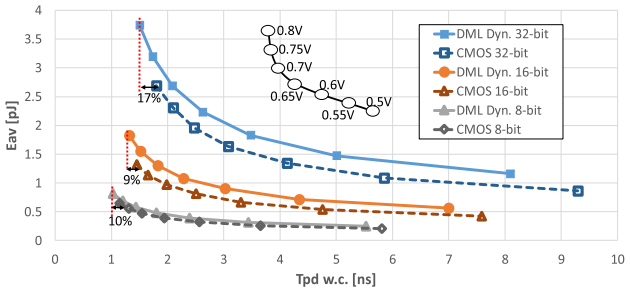


FIGURE 9. Dynamic DML vs. CMOS Tpd and Energy Results.

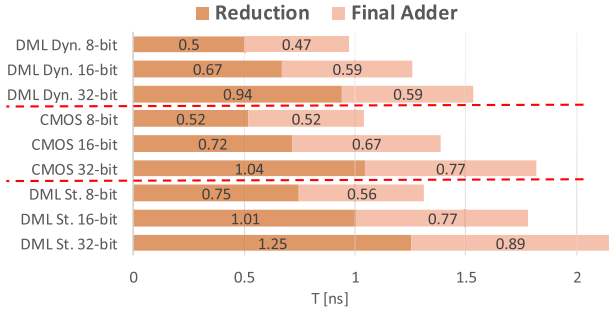


FIGURE 10. Tpd divided into parts of the configurable multiplier, the reduction, and the final adder.

four. This feature may aid in compile-time decisions striking a balance between work-load and overall performance. On the other hand, the energy consumption in the DML dynamic mode is higher, with average increases of 37%, 35%, and 23% for the corresponding multiplier configurations. These findings provide valuable insights into the trade-off between energy consumption and timing performance in the different operation modes of the DML multiplier compared to its CMOS counterpart.

Fig. 10 provides insights into the acceleration of the multiplier when changing its configuration, regardless of the operating modes. The internal propagation delay of the multiplier was analyzed through simulations, specifically examining the contribution of different parts of the multiplier to the overall delay. The reduction part, encompassing stages 0-9 in Fig. 3, and the final adder (stage 10) were evaluated separately. The acceleration observed in the reduction part is attributed to the availability of the bypass option. While the 32-bit multiplier utilizes all 10 stages, the 16-bit multiplier bypasses stages 7 and 8, resulting in 8 stages. Similarly, the 8-bit multiplier only passes through 6 stages and bypasses stages 5-8. In the final adder part, the acceleration is attributed to the shorter rippling path. Although the RCA unit calculates the skip logic of all blocks in parallel for all configurations, the difference lies in the number of MUX gates the data path needs to traverse. For the 32-bit, 16-bit, and 8-bit multipliers, the maximum number of MUXs encountered is 14 blocks, 8 blocks, and 4 blocks, respectively (as depicted in Fig. 5). These variations in the number of MUX gates contribute

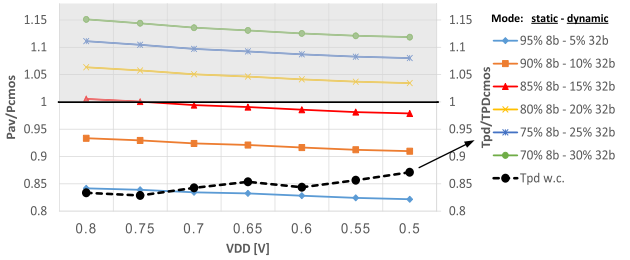


FIGURE 11. Mix mode of DML with different usage percentages of 32-bit and 8-bit configurations.

to the observed acceleration in the final adder part of the multiplier. Overall, these findings demonstrate the impact of configuration changes on the internal propagation delay of the multiplier, thus highlighting the acceleration achieved in the reduction part and the final adder part for different multiplier configurations.

The maximum efficiency of the DML multiplier can be achieved by leveraging a mixed mode of operation that combines the strengths of both the dynamic and static modes, based on the specific requirements of the system. This study investigated the feasibility of using a mixed mode configuration where the 32-bit multiplier operates in the dynamic mode to accelerate slower operations, while the 8-bit multiplier operates in the static mode to conserve power consumption during common operations. In this configuration, the timing degradation of the static mode does not impact the overall performance of the multiplier, and the energy consumption in the dynamic mode becomes negligible due to the power-saving nature of the static mode configuration. Fig. 11 illustrates the improvements achieved for different utilization percentages of the multipliers, in both performance and energy consumption, compared to the same percentages in the CMOS implementation. The power improvement is shown on the left Y-axis, while the timing improvement is shown on the right Y-axis. The results indicate an average timing improvement of approximately 15% across the entire range of supply voltages. This timing improvement is related to the fact that while the worst case operation of the CMOS implementation is the 32-bit multiplier, the DML implementation's worst case is the 32-bit multiplier in the faster dynamic mode. When using the 32-bit multiplier for less than 15% of the operations, the DML multiplier demonstrates power improvement. The power improvement becomes more significant at lower supply voltages. These findings highlight the benefits of using a mixed-mode configuration in the DML multiplier, since it allows for optimized power consumption and improved timing performance.

Finally, a comparison to previous works is provided in Table 2. The comparison includes smaller DML multipliers from previous publications, with extrapolations to 32 × 32-bit multipliers [23], [24], [25], as well as state-of-the-art (non-DML) 32 × 32-bit multipliers reported in the

TABLE 2. Comparison Results.

Design	[23]			[24]			[25]			This work			[19]	[18]	
Technology	16 nm FinFET			16 nm FinFET			65 nm CMOS			16 nm FinFET			65 nm CMOS	65 nm CMOS	
Architecture	Radix-4 Booth Multiplier Scaled Data[a]			Bit Fusion Multiplier Scaled Data[a]			Carry-Save Multiplier Scaled Data[a]			Wallace-Tree Multiplier			Wallace Multiplier with CLA (post-layout)	NR-SO Wallace Multiplier (pre-layout)	Baseline Multiplier (pre-layout)
	DML_s	DML_d	CMOS	DML_s	DML_d	CMOS	DML_s	DML_d	CMOS	DML_s	DML_d	CMOS			
V_{DD} [V]	0.8			0.8			1.2			0.8			0.9	1	
Latency [cyc]	1			3			2			1			1	1	
Freq. [MHz]	357.5	470	402.5	491.16	936.33	558.91	297.62	416.67	312.5	444.85	667.27	556.06	434.78	109.9	76.9
Energy/Op [pJ]	3.88	6.16	5.12	11.53	16.53	13.18	10.16	13.76	12.24	1.99	3.75	2.69	7.66	16.2	32.7
Area [μm^2]	3,294.8		3,789	17,517.8		11,214	N.A.			4,447.9		3,043.8	14,665	21,002.8	32,765.2
Test Chip	✓[b]			✓[c]			-			✓			-	-	

[a] Scaled data for a 32x32-bit multiplier is extrapolated according to equations in [22]
[b] Test chip for 16x16-bit multiplier
[c] Test chip for 4x4-bit multiplier

literature [18], [19]. Our multiplier demonstrated the best performance and energy consumption trade-off, while also occupying a smaller area. As compared to [23], which is also fabricated in 16 nm FinFET and operates at 0.8 V supply voltage, we achieve 24%/42%/38% higher frequencies, in parallel to energy consumption saving of 49%/39%/48%, in the static DML, dynamic DML and CMOS, respectively. Notably, the multipliers in [24], which are 16 nm FinFET and 0.8 V as well, are arranged in a 3-stage pipeline, enabling higher frequency operation, but at the expense of increased latency. In our implementation, there is a degradation of 9%/29%/1% in the frequency, with a significant energy saving of 83%/78%/80% less energy consumption, in the same respective operation modes. As compared to [25], note that it was fabricated in a 65 nm technology (therefore with a 1.2V nominal supply voltage), we show 49%/60%/78% performance improvement, with 80%/73%/78% energy consumption savings. This makes sense even considering that voltage ($X0.66$) and technology ($X0.25$) scaling lead to lower energy consumption with performance improvement along with more complex and accelerated design. Similarly, as compared to [19] (also 65 nm, 0.9 V supply voltage), our CMOS results indicate 28% higher maximum frequency with 65% less energy consumption, and in comparison to the two outstanding architectures in [18] (65 nm and 1 V), about 400%/600% with 84%/92% improvement in performance and energy is achieved.

Generally, DML design can be area efficient as compared to CMOS blocks. However, this depends on the application and designer. For example, in [22], [23] the DML multiplier is more area efficient than CMOS; however, in [24] and in this work the opposite is true. The reason lies in the fact that the novel sizing of DML gates becomes more efficient when the standard CMOS transistor sizing (i.e., up-sizing Xi factors) is not minimal. In our comparative design, for simplicity (and as a worst-case), the CMOS implementation used minimum gate sizing (as the DML-based design), so that due to the additional overhead of the DML control, the total area is impacted. The area footprint of our design is $4447.9\mu m^2$ for the DML implementation and $3043.8\mu m^2$ for the CMOS, slightly larger than [23]. This is reasonable since we incorporate a more complex compression circuitry and control and bypass cost with this negligible overhead. The other designs consume much more area, due to either more flexibility [24] or older technology nodes [18], [19].

VI. CONCLUSION

We presented an improved multiplier on the architecture level and the logic gate level. At the architecture level, we implemented a configurable multiplier, which can be programmed dynamically to three configuration options: a 32 × 32-bit multiplier, two 16 × 16-bit multipliers, and four 8 × 8-bit multipliers. All the configurations are on the same hardware, with high area utilization and bypass mechanisms for acceleration. Even this innovative multiplier shows significant improvements in comparison to previous publications in the literature. In addition, at the gate level, we used the DML logic with two modes of operation: the fast dynamic mode and the low energy static mode. Each mode of operation presents its own familiar characteristics when operating separately: high frequency in the dynamic mode, and low energy consumption in the static mode. Furthermore, when the DML's mode of operation is adapted to the multiplier's configuration option, the data showed improvement in both frequency performance and energy consumption, of 15% and 18%, respectively. Finally, we presented the non-trivial semi-automation flow utilized for the complex implementation of the architecture. This semi-automation flow is likely to be useful for future complex DML design.

ACKNOWLEDGMENT

The authors would like to thank Or Maltababashi for his devoted assistance with the CMOS's Python script, and also would like to thank Ido Assaf for his efforts with the DML's Python script generation and the DML standard cells layout.

REFERENCES

[1] T. Abtahi, C. Shea, A. Kulkarni, and T. Mohsenin, "Accelerating convolutional neural network with FFT on embedded hardware," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 9, pp. 1737–1749, Sep. 2018.

[2] D. Wu, X. Fan, W. Cao, and L. Wang, "SWM: A high-performance sparse-winograd matrix multiplication CNN accelerator," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 5, pp. 936–949, May 2021.

[3] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing the convolution operation to accelerate deep neural networks on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 7, pp. 1354–1367, Jul. 2018.

[4] W. Huang, H. Wu, Q. Chen, C. Luo, S. Zeng, T. Li, and Y. Huang, "FPGA-based high-throughput CNN hardware accelerator with high computing resource utilization ratio," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 8, pp. 4069–4083, Aug. 2022.

[5] C. Ding, Y. Huan, L. Zheng, and Z. Zou, "Dynamic precision multiplier for deep neural network accelerators," in *Proc. IEEE 33rd Int. Syst.-Chip Conf. (SOCC)*, Sep. 2020, pp. 180–184.

[6] P.-H. Kuo, Y.-H. Huang, and J.-D. Huang, "Configurable multi-precision floating-point multiplier architecture design for computation in deep learning," in *Proc. IEEE 5th Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Jun. 2023, pp. 1–5.

[7] W. Mao, K. Li, Q. Cheng, L. Dai, B. Li, X. Xie, H. Li, L. Lin, and H. Yu, "A configurable floating-point multiple-precision processing element for HPC and AI converged computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 2, pp. 213–226, Feb. 2022.

[8] D. Guevorkian, A. Launiainen, V. Lappalainen, P. Liuha, and K. Punkka, "A method for designing high-radix multiplier-based processing units for multimedia applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 5, pp. 716–725, May 2005.

[9] D. J. Moni and P. E. Sophia, "Design of low power and high speed configurable booth multiplier," in *Proc. 3rd Int. Conf. Electron. Comput. Technol.*, vol. 6, Apr. 2011, pp. 338–342.

[10] R. Ramya and S. Moorthi, "Design and implementation of accuracy configurable multi-precision multiplier architecture for signal processing applications," in *Proc. IEEE Recent Adv. Intell. Comput. Syst. (RAICS)*, Dec. 2018, pp. 89–93.

[11] B. Boro, K. M. Reddy, Y. B. N. Kumar, and M. H. Vasantha, "Approximate radix-8 booth multiplier for low power and high speed applications," *Microelectron. J.*, vol. 101, Jul. 2020, Art. no. 104816.

[12] M. M. A. Basiri, S. C. Nayak, and N. M. Sk, "Multiplication acceleration through quarter precision Wallace tree multiplier," in *Proc. Int. Conf. Signal Process. Integr. Netw. (SPIN)*, Feb. 2014, pp. 502–505.

[13] L. Nan, X. Zeng, Q. Ding, W. Li, Y. Du, and L. Chen, "Research of special instructions for finite field x multiplications of cryptographic algorithms," in *Proc. IEEE 3rd Adv. Inf. Technol., Electron. Autom. Control Conf. (IAEAC)*, Oct. 2018, pp. 1608–1613.

[14] L.-M. Nan, X.-Y. Zeng, W. Li, C. Lin, Y.-R. Du, and Z.-B. Dai, "Research of special instructions for composite field multiplications in symmetric cryptographic algorithms," in *Proc. 14th IEEE Int. Conf. Solid-State Integr. Circuit Technol. (ICSICT)*, Oct. 2018, pp. 1–4.

[15] S. Bayat-Sarmadi, M. Mozaffari Kermani, R. Azarderakhsh, and C.-Y. Lee, "Dual-basis superserial multipliers for secure applications and lightweight cryptographic architectures," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 61, no. 2, pp. 125–129, Feb. 2014.

[16] D. Zoni, A. Galimberti, and W. Fornaciari, "Flexible and scalable FPGA-oriented design of multipliers for large binary polynomials," *IEEE Access*, vol. 8, pp. 75809–75821, 2020.

[17] P. Patali and S. Thottathikkulam Kassim, "Efficient modular hybrid adders and radix-4 booth multipliers for DSP applications," *Microelectron. J.*, vol. 96, Feb. 2020, Art. no. 104701.

[18] L. M. G. Rocha, M. Macedo, G. Paim, E. Costa, and S. Bampi, "Improving the partial product tree compression on signed radix-2^m parallel multipliers," in *Proc. 18th IEEE Int. New Circuits Syst. Conf. (NEWCAS)*, Jun. 2020, pp. 182–185.

[19] B. Ramkumar and H. M. Kittur, "Faster and energy-efficient signed multipliers," *VLSI Des.*, vol. 2013, p. 13, 2013.

[20] S.-R. Kuang and J.-P. Wang, "Design of power-efficient configurable booth multiplier," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 3, pp. 568–580, Mar. 2010.

[21] I. Levi and A. Fish, *Dual Mode Logic: A New Paradigm for Digital IC Design*. Cham, Switzerland: Springer, 2021.

[22] R. Taco, I. Levi, M. Lanuzza, and A. Fish, "An 88-fJ/40-MHz [0.4 V]–0.61-pJ/1-GHz [0.9 V] dual-mode logic 8×8 bit multiplier accumulator with a self-adjustment mechanism in 28-nm FD-SOI," *IEEE J. Solid-State Circuits*, vol. 54, no. 2, pp. 560–568, Feb. 2019.

[23] N. Shavit, I. Stanger, R. Taco, M. Lanuzza, and A. Fish, "A 0.8-V, 1.54-pJ/940-MHz dual-mode logic-based 16×16-b booth multiplier in 16-nm FinFET," *IEEE Solid-State Circuits Lett.*, vol. 3, pp. 314–317, 2020.

[24] I. Stanger, N. Shavit, R. Taco, M. Lanuzza, L. Yavits, I. Levi, and A. Fish, "FlexDML: High utilization configurable multimode arithmetic units featuring dual mode logic," *IEEE Solid-State Circuits Lett.*, vol. 6, pp. 73–76, 2023.

[25] R. De Rose, P. Romero, and M. Lanuzza, "Double-precision dual mode logic carry-save multiplier," *Integration*, vol. 64, pp. 71–77, Jan. 2019.

[26] I. Levi, A. Kaizerman, and A. Fish, "Low voltage dual mode logic: Model analysis and parameter extraction," *Microelectron. J.*, vol. 44, no. 6, pp. 553–560, Jun. 2013.

[27] I. Levi and A. Fish, "Dual mode logic: Design for energy efficiency and high performance," *IEEE access*, vol. 1, pp. 258–265, 2013.

[28] N. Shavit, R. Taco, and A. Fish, "Efficiency of dual mode logic in nanoscale technology nodes," in *Proc. IEEE Int. Conf. Sci. Electr. Eng. Isr. (ICSEE)*, Dec. 2018, pp. 1–4.

[29] Y. Sun and V. Kursun, "Carbon nanotubes blowing new life into NP dynamic CMOS circuits," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 2, pp. 420–428, Feb. 2014.

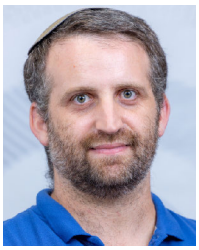
[30] B. Majji and K. Ragini, "Design and implementation of RNB multiplier using NP domino logic," in *Proc. Int. Conf. Recent Trends Microelectron., Autom., Comput. Commun. Syst. (ICMACC)*, Dec. 2022, pp. 270–275.

[31] I. Levi, A. Belenky, and A. Fish, "Logical effort for CMOS-based dual mode logic gates," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 5, pp. 1042–1053, May 2014.

[32] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, no. 1, pp. 14–17, Feb. 1964.

[33] H. Marinberg, E. Garzón, T. Noy, M. Lanuzza, and A. Teman, "Efficient implementation of many-ported memories by using standard-cell memory approach," *IEEE Access*, vol. 11, pp. 94885–94897, 2023.

[34] V. Yuzhaninov, I. Levi, and A. Fish, "Design flow and characterization methodology for dual mode logic," *IEEE Access*, vol. 3, pp. 3089–3101, 2015.



NETANEL SHAVIT (Graduate Student Member, IEEE) received the B.Sc. (summa cum laude) and M.Sc. degrees in electrical engineering from Bar-Ilan University, Ramat Gan, Israel, in 2017 and 2019, respectively, where he is currently pursuing the Ph.D. degree.

His current research interests include low-power and high-frequency logic families and digital circuit design, in wide abstraction layers: from the transistors level to the architecture level, from full custom to EDA tools, from design, layout, and simulations to chip measurements.

INBAL STANGER (Graduate Student Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical engineering from Bar-Ilan University, Ramat Gan, Israel, in 2018 and 2020, respectively, where she is currently pursuing the Ph.D. degree.

Her current research interests include the design of low-power and high-frequency digital circuits under extreme process, voltage and temperature variations, and specifically the design of unique logic families under cryogenic temperatures.

RAMIRO TACO (Member, IEEE) received the M.S. degree in electrical engineering from University San Francisco de Quito, Quito, Ecuador, in 2012, and the Ph.D. degree in electrical engineering from the University of Calabria, Rende, Italy, in 2017. In 2017, he joined the Emerging Nanoscaled Integrated Circuits and Systems Laboratories, Bar-Ilan University, Ramat Gan, Israel, as a Postdoctoral Fellow. In 2020, he joined Universidad San Francisco as the Head of the Institute of Micro and Nanoelectronics, Quito. He led the first integrated circuit designed in Ecuador. In 2023, he joined DIMES as a Senior Researcher. His research interests include high-speed and energy-efficient mixed-signal VLSI designs and energy-efficient IC memories.

ALEXANDER FISH (Member, IEEE) received the B.Sc. degree in electrical engineering from the Technion—Israel Institute of Technology, Haifa, Israel, in 1999, and the M.Sc. and Ph.D. (summa cum laude) degrees from Ben-Gurion University (BGU), Israel, in 1999 and 2002, respectively. He was a Postdoctoral Fellow with the ATIPS Laboratory, University of Calgary, Canada, from 2006 to 2008. In 2008, he joined as a Faculty Member with the Electrical and Computer Engineering Department, BGU. He founded the Low Power Circuits and Systems (LPC&S) Laboratory, specializing in low power circuits and systems. In July 2011, he was appointed as the Head of the VLSI Systems Center, BGU. In October 2012, he joined the Faculty of Engineering, Bar-Ilan University, as an Associate Professor, and the Head of the Nanoelectronics Track. In March 2015, he founded the Emerging Nanoscaled Integrated Circuits and Systems (ENICS) Laboratories. Currently, he is a Full Professor and the Co-Director of the EnICS Impact Center. His research interests include power reduction methodologies for high speed digital and mixed signal VLSI chips, energy efficient SRAM and eDRAM memory arrays, CMOS image sensors and biomedical circuits, systems and applications, and cryogenic CMOS circuits. He has authored over 190 scientific papers in journals and conferences. He also submitted more than 30 patent applications of which 22 were granted. He has published three book chapters and two books as an editor.

He founded and served as the Editor-in-Chief for the *Journal of Low Power Electronics and Applications* (JLPEA) (MDPI), from 2012 to 2018, and he was an associate editor of IEEE various journals. He is an Associate Editor of IEEE Access journal, *Microelectronics Journal* (Elsevier), and *Integration, the VLSI Journal* (Elsevier). He also served as the program chair and the chair of different tracks of various IEEE conferences. He was a co-organizer of many special sessions at IEEE conferences, including IEEE ISCAS, IEEE Sensors, and IEEEI conferences. He is a member of the Technical Committee of the European Solid-State Circuits Conference. He is also a member of the VLSI Systems and Applications and Bio-Medical Systems Technical Committees of IEEE Circuits and Systems Society.

ITAMAR LEVI (Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical and computer engineering from Ben-Gurion University, in 2012 and 2013, respectively, and the Ph.D. degree from Bar-Ilan University (BIU), Ramat Gan, Israel, in 2017.

He was a Research-Associate with the UCLouvains Crypto-Group, UCLouvain, Belgium, until 2019. He is currently a Computer-Engineering Faculty Member with BIU, where he is also a member of the Emerging Nanoscale Circuits and Systems Laboratories (EnICS). His current research interests include digital circuit design and acceleration, embedded systems security, security evaluation analysis for cryptographic devices, side-channel and fault-injection attacks and countermeasures, and cryptographic implementations. He has (co)authored over 60 journal articles and international conference papers and seven patent applications, he coauthored a book on *Dual-Mode-Logic: A New Paradigm for Digital IC Design* and serves in various technical committees of IEEE CAS Society and Hardware Security journals and conferences.