International Journal of Wireless and Ad Hoc Communication

♦ ISSN: 2692-4056 Vol: 09, No. 02, 2025 ♦ PP. 08–13

Article AOI: Attps://doi.org/10.54216/IJWAC.090202



FULL LENGTH ARTICLE

FPGA Implementation of High Performance Accurate and Approximate Signed and Unsigned Multipliers using Structure of LUT Configurations

Saravanan V.[1,*] **Elarmathi S.**[2] **Rajalakshmi V. R.**[3]

[1] Associate Professor, Department of ECE, Knowledge Institute of Technology, Salem, Tamil Nadu, India
[2] Assistant Professor, Department of ECE, Knowledge Institute of Technology, Salem, Tamil Nadu, India
[3] PG Scholar, Department of ECE, Knowledge Institute of Technology, Salem, Tamil Nadu, India

Emails: vsece@kiot.ac.in · seece@kiot.ac.in · 2k22vlsi12@kiot.ac.in

Received: January 17, 2025 Revised: February 13, 2025 Accepted: March 14, 2025 ⋆ Corresponding author

ABSTRACT

A recent study examined the applications of multiplication and division in video and image manipulation, with attention also given to machine learning. DSP blocks that function as high-performance multipliers are provided by FPGA vendors. However, routing lag time and inefficiencies, particularly for lower bit-width multiplications, can emerge from the fixed placements and restricted number of FPGA multipliers, raising power consumption. FPGA companies offer soft IP cores made especially for multiplication to solve this problem. Even if these IP cores have improved over time, they can still be improved. This can be accomplished by creating low-latency, accurate, and core multiplier topologies that maximize FPGA space and take advantage of architectural characteristics such as rapid carry chains and look-up table structures. These architectures seek to improve overall efficiency by lowering critical path delay and multiplier resource consumption. This paper presents a method for building accurate and approximate signed and unsigned multipliers for an eight-bit configuration. This entails changing the LUT6 architecture to use one LUT5 with multiplexers in place of dual LUT5s with multiplexers. Using Xilinx software, the design was built in Verilog HDL and synthesized. At the conclusion of the process, variables including area, delay, and power were compared.

Keywords: FPGA ▪ Multipliers ▪ Power consumption ▪ LUT ▪ HDL

1. INTRODUCTION

A basic mathematical operation used frequently in digital image and signal processing is multiplication. Manufacturers of FPGAs such as Xilinx provide specific DSP blocks for effective multiplication. Even while these DSP blocks offer excellent performance, for some uses they might not always be the best option in terms of efficiency and resource use. Prior work compares critical path delays and lookup tables in two distinct implementations of JPEG and Reed–Solomon encoders on Virtex-7 series FPGAs with Xilinx Vivado. Implementing the DSP-based Reed–Solomon encoder can become more delayed as a result of routing delays caused by DSP block placement. In some cases, manual floor planning may be necessary for smaller applications. It is crucial to maximize FPGA resource allocation to increase application performance. However, in complex applications with conflicting resource requirements, manual optimization may not be feasible. For instance, when implementing a JPEG encoder, a significant portion of the accessible DSP blocks is utilized. This can deplete DSP blocks and limit their avail-

© 2025 The Author(s). Published by ASPG. This article is licensed under CC BY 4.0.

ability for critical operations in other applications running simultaneously on the same FPGA. This highlights the need for LUT-based multipliers to address resource constraints and improve overall application performance. Therefore, incorporating soft logic-based multipliers in addition to DSP blocks is crucial for achieving optimal performance across various implementation scenarios [1].

Methods that use a modular approach create larger FPGA-based multipliers from smaller blocks. While effective for small bit-width multipliers, these techniques can be resource-intensive for larger bit-width multipliers. For instance, implementing an 8×8 multiplier on a Virtex-7 FPGA using default Vivado synthesis settings requires 61 LUTs [2]. In contrast, a modular approach using 4×4 multipliers to create an accurate 8×8 multiplier consumes 82 LUTs. Researchers have investigated modified Booth algorithms for effective radix-4 multiplier designs on Altera and Xilinx FPGAs [7, 12]. Walters and Kumm used carry chains and six-input lookup tables in their Xilinx FPGA implementation to avoid partial-product trees, but faced significant critical-path delays. Parandeh-Afshar and colleagues employed Booth and Baugh–Wooley algorithms on Altera FPGAs for area-efficient multiplier designs, limiting adaptive logic module duration to reduce carry-chain lengths. However, this resulted in underutilization of FPGA resources and was not directly feasible for other FPGA vendors such as Xilinx.

Among the design options for multipliers, traditional shift-and-add, serial, and serial/parallel multipliers prioritize minimizing area but suffer from lengthy critical-path delays. Parallel multipliers such as Wallace and Dadda structures reduce delay through partial-product reduction, while FPGA-specific designs must also consider LUT structures, carry chains, routing overhead, and available DSP resources.

2. RELATED WORK

Recently, approximate computing has gained popularity as a viable method for designing digital systems with minimal energy use. The foundation of approximate computing is the tolerance of many systems and applications for a certain amount of error. This makes approximate arithmetic attractive in signal processing, image processing, machine learning, and other applications where perfect arithmetic accuracy may not always be necessary.

Kumm [4] developed a method for massive multiplier design for FPGAs that maximizes resource usage. These designs are made up of smaller multipliers, which can be logic-based multipliers. Pilipović and Bulić [8] developed a logarithmic multiplier design with radix-4 Booth encoding. These advantages come at the cost of decreased accuracy. Ullah et al. [6] developed approximate multipliers for FPGA-based hardware accelerators. Improved designs are required for high performance in FPGAs with soft multiplier IP cores. By taking advantage of lookup table structures and carry-chain resources, these designs improve area, delay, and power trade-offs.

Liang et al. [9] developed a binarized neural network for FPGAs, radically reducing hardware consumption while maintaining acceptable precision. Mittal [10] surveyed FPGA-based accelerators for convolutional neural networks and showed that integer and binary operations can benefit from optimized multiplier design. Faraone et al. [12] developed AddNet, using FPGA-optimized multipliers for neural networks. These studies demonstrate that multiplier optimization remains central to FPGA accelerator performance.

3. EXISTING SYSTEM

As a fundamental mathematical operation frequently employed in image and digital signal processing, multiplication plays a significant role. Multipliers may be quickly created with blocks of DSP, which FPGA manufacturers such as Xilinx supply. Although these DSP blocks have high performance, they may not always be an ideal option for resource efficiency and utilization.

To construct larger FPGA-based multipliers with smaller blocks, new techniques are being developed. These methods work better with smaller bit-width multipliers. Larger bit-width multipliers ultimately result in more FPGA resource consumption. Furthermore, a challenge with existing FPGA architectures is that carry chains and adaptive logic structures can limit portability and utilization efficiency across vendors.

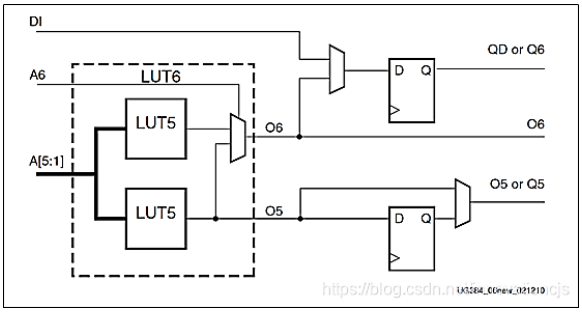


Figure 1. LUTs in subsequent ALMs.

Among the various options available for designing multipliers, traditional shift-and-add, serial, and serial/parallel multipliers are known for their ability to meet low-area requirements but suffer from high critical-path delays. Conversely, popular parallel multipliers reduce delay but often require more resources. This motivates optimized LUT-level architectures for FPGA multipliers.

4. PROPOSED SYSTEM

Multiplication and division are now often used in a wide range of applications, including machine learning and image/video processing. High-performance DSP blocks with multipliers are available from FPGA suppliers; however, the quantity and arrangement of these blocks on FPGAs are constrained. This may result in inefficient lower bit-width multiplications and routing delays, which raise power consumption. FPGA companies offer soft IP cores made especially for multiplication to solve this problem. Even if the designs of these soft multiplier IP cores have evolved over time, more may be done to increase resource and performance efficiency. In this work, we suggest creating low-latency, approximate, accurate, and area-optimized soft-core multiplier architectures by utilizing FPGA architectural features including rapid carry chains and look-up table structures.

Our goal is to lower multiplier resource consumption and critical-path latency. Using a unique strategy, we build approximate and accurate signed and unsigned multipliers for

8-bit setups. Additionally, we propose reworking the LUT6 architecture by substituting a single LUT5 with multiplexers for dual LUT5s with multiplexers, as shown in Figure 2. The suggested designs were synthesized with Xilinx software and implemented in Verilog HDL. Area, delay, and power consumption were used to evaluate performance.

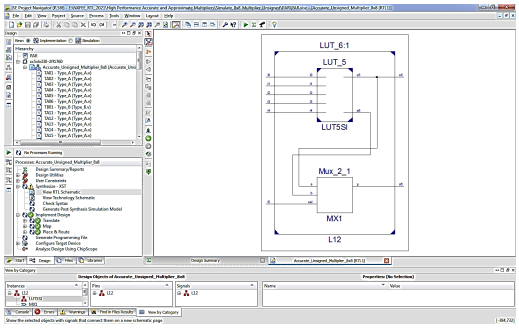


Figure 2. Proposed LUT6 slice structure.

The earlier discussed binary and ternary adders for adding partial products are included in the proposed design of a precise multiplier. Resource usage can be made more effective by using ternary adders. However, because each element in the carry chain depends on the carry-generate signal from the previous cell, the efficiency of a ternary adder is reduced. For instance, an 8-bit ternary adder with three operands written on a Virtex-7 FPGA using Vivado has greater critical-path latency than an 8-bit binary adder with two operands. To address this issue, we employ various approximation techniques to create high-performance and resource-efficient approximate multipliers, as shown in Figures 3 and 4.

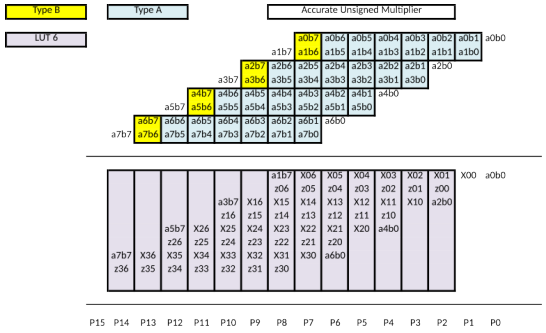


Figure 3. Proposed unsigned accurate and approximate multiplier.

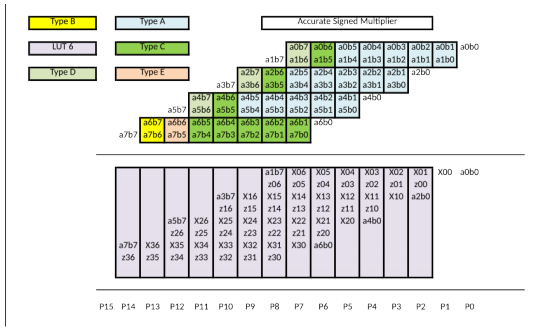


Figure 4. Proposed signed accurate and approximate multiplier.

4.1 Total Assistance of the Work

Precise unsigned multiplier architecture: This refers to a digital circuit design that performs multiplication of two unsigned binary numbers with full precision, producing the exact result without loss of accuracy while remaining area optimized.

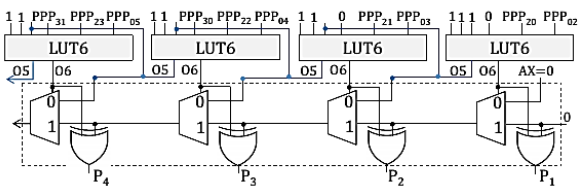


Figure 5. Ternary adder calculation of final product.

Generation and addition of one-step partial products: In unsigned binary multiplication, one-step partial products are generated by ANDing each bit of the multiplier with the entire multiplicand. Each partial product is shifted left based on the position of the corresponding multiplier bit. After generating the partial products, they are added to obtain the final product.

Effective partial-product reduction tree: A key element of high-speed multiplier design is the effective partial-product reduction tree, which decreases the number of partial products to two final rows that can be added with a fast adder. Wallace tree and Dadda tree methods are common techniques for this purpose. Equation 6 summarizes the number of steps required to find the exact product.

$$No.\ of\ stages\ =\ \left\lceil log_3(\frac{M}{2})\right\rceil + 1 \qquad (1)$$

Figure 6. Equation for exact product computation steps.

Signed accurate multiplier: A signed accurate multiplier performs multiplication of signed binary numbers while maintaining full precision in the result. The multiplier handles positive and negative numbers, typically represented in two's-complement format. The main challenge is handling sign bits correctly while maintaining efficiency and speed.

Unsigned approximate 4×2 **multiplier:** This is a foundational element for higher-order approximate multipliers. The six inputs of a LUT in advanced FPGAs are fully utilized by the suggested 4×2 approximate multiplier. An asymmetric approximate 4×4 multiplier is built by applying FPGA-specific optimizations to minimize output errors.

Estimated ternary adder: A ternary approximate adder is used for summation of approximate partial products. Our approximate and accurate topologies reduce the total number of LUTs used by up to 26% compared with the area-optimized Xilinx LogiCORE IP. The suggested approximation architecture reduces multiplier critical-path delay by up to 51%. The quality metrics used for error characterization are the total number of errors, highest error magnitude, average relative error, and total number of maximum errors.

4.2 Slice Structure of Xilinx FPGA

Modern FPGAs, such as those made available by Intel and Xilinx, create sequential as well as combinational circuits using six-input LUTs. All designs mentioned have been implemented using Xilinx FPGAs. Nonetheless, the suggested approach is universal and may be used with FPGAs from different suppliers, such as Intel, which likewise employs carry chains and factorable six-input LUTs. The Xilinx FPGA slice structure is a fundamental building block that typically contains look-up tables, flip-flops, multiplexers, carry logic, arithmetic logic, and wide multiplexers. Key features are configurability, the ability to split six-input LUTs into two five-input LUTs, independent use of LUTs and flip-flops, ded-

icated routing, and dedicated carry chains for fast arithmetic operations. The carry chain implements carry-lookahead behavior using $O5$ as the carry-generate signal and $O6$ as the carry-propagate signal.

$$S_i = P_i \oplus C_i \tag{2}$$

$$C_{i+1} = G_i + P_i \cdot C_i \tag{3}$$

Figure 7. Slice structure of a Xilinx FPGA.

4.3 Baugh–Wooley's Multiplication Algorithm

In contrast to unsigned multiplication, accurate product computation in signed multiplication requires precise sign extension of all partial products. The Baugh–Wooley multiplication algorithm encodes sign information in the generated partial products, eliminating the requirement for explicit sign-extension bit computation and communication.

$$\begin{aligned} A &= -a_{N-1}2^{N-1} + \sum_{n=0}^{N-2} a_n 2^n \\ B &= -b_{M-1}2^{M-1} + \sum_{m=0}^{M-2} b_m 2^m \end{aligned} \tag{4}$$

$$\begin{aligned} P = {} & a_{N-1}b_{M-1}2^{N+M-2} + \sum_{n=0}^{N-2}\sum_{m=0}^{M-2} a_n b_m 2^{n+m} \\ & - 2^{N-1}\sum_{m=0}^{M-2} a_{N-1}b_m 2^m - 2^{M-1}\sum_{n=0}^{N-2} b_{M-1}a_n 2^n \end{aligned} \tag{5}$$

$$\begin{aligned} P = {} & a_{N-1}b_{M-1}2^{N+M-2} + \sum_{n=0}^{N-2}\sum_{m=0}^{M-2} a_n b_m 2^{n+m} \\ & + 2^{N-1}\sum_{m=0}^{M-2} \overline{a_{N-1}b_m} 2^m + 2^{M-1}\sum_{n=0}^{N-2} \overline{b_{M-1}a_n} 2^n \\ & + 2^{N-1} + 2^{M-1} + 2^{N+M-1} \end{aligned} \tag{6}$$

$$\begin{aligned} P_8 = {} & a_7 b_7 2^{14} + \sum_{n=0}^{6}\sum_{m=0}^{6} a_n b_m 2^{n+m} + 2^7 \sum_{m=0}^{6} \overline{a_7 b_m} 2^m \\ & + 2^7 \sum_{n=0}^{6} \overline{b_7 a_n} 2^n + 2^8 + 2^{15} \end{aligned} \tag{7}$$

Figure 8. Baugh–Wooley multiplication equations and partial-product representation.

For an $N \times M$ signed multiplier, the corresponding operands are specified in two's-complement representation. The signed partial products are generated to determine the final product P. By using Baugh–Wooley's multiplication method, the negative partial-product terms are rewritten, eliminating the requirement for explicit sign-extension bits. In the equation, $a_x b_y$ stands for the one's complement of the corresponding partial-product term for $x \in [0, N-1]$ and $y \in [0, M-1]$.

5. RESULTS AND DISCUSSION

5.1 Signed Accurate Multiplier

The implemented accurate signed 8×8 multiplier uses LUT6 elements, employed using single LUT5s and multiplexers. The logic used for execution includes sign extension, Booth encoding, partial-product generation, partial-product accumulation, final addition, and sign correction, as shown in Figure 9.

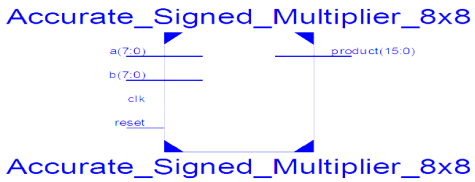


Figure 9. Signed accurate multiplier.

```
Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 14962 / 16
-------------------------------------------------------------------------
Delay:               13.818ns (Levels of Logic = 16)
  Source:            a<2> (PAD)
  Destination:       product<15> (PAD)

  Data Path: a<2> to product<15>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             17   0.694   0.736  a_2_IBUF (a_2_IBUF)
     LUT4:I0->O             2   0.086   0.666  TA12/Mxor_o6_Result1 (x11)
     LUT6:I2->O             2   0.086   0.905  L3/LUT5SI/Mxor_x1_Result1 (L3/LUT5SI/x1)
     LUT6:I0->O             4   0.086   0.613  L3/LUT5SI/Mxor_x3_Result1 (L3/si)
     LUT4:I1->O             1   0.086   0.901  n089 (n0)
     LUT6:I0->O             4   0.086   0.500  n11 (n1)
     LUT4:I2->O             1   0.086   0.412  n279 (n2)
     LUT6:I5->O             4   0.086   0.914  n389 (n3)
     LUT6:I0->O             5   0.086   0.505  n4 (n4)
     LUT4:I2->O             3   0.086   0.496  n5 (n5)
     LUT4:I2->O             4   0.086   0.914  n6 (n6)
     LUT6:I0->O             5   0.086   0.837  L17/LUT5SI/o51 (y16)
     LUT6:I1->O             4   0.086   0.425  L18/LUT5SI/o51 (y17)
     LUT6:I5->O             2   0.086   0.666  L19/LUT5SI/o51 (y18)
     LUT6:I2->O             1   0.086   0.286  Mxor_product<15>_Result (product_15_OBUF)
     OBUF:I->O                  2.144          product_15_OBUF (product<15>)
    ----------------------------------------
    Total                      13.818ns (4.042ns logic, 9.776ns route)
                                        (29.3% logic, 70.7% route)
```

Figure 10. Timing report of signed accurate multiplier.

5.2 Timing Report of Signed Accurate Multiplier

The timing report generated by synthesis and place-and-route tools contains detailed information specific to the design and target device. This includes gate delay and net delay. Gate delays refer to propagation delays associated with elements such as LUT6s, LUT5s, and multiplexers used in multiplier implementation. Each LUT has a propagation delay from inputs to output, depending on the logic function and internal LUT structure.

These gate delays are typically reported along with the specific logic elements and paths where they occur. Net delays refer to delays associated with interconnect routing between logic elements. Interconnect delays depend on the physical routing of signals in the FPGA or ASIC device, including wire length, fan-out, and capacitive loading. Net delays can vary significantly depending on placement and routing.

The power report of the signed accurate multiplier and the output waveform of the signed accurate multiplier are shown in Figures 11 and 12.

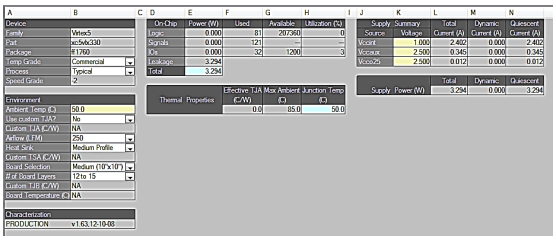


Figure 11. Power report of signed accurate multiplier.

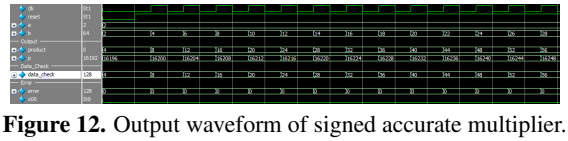


Figure 12. Output waveform of signed accurate multiplier.

5.3 Unsigned Accurate Multiplier

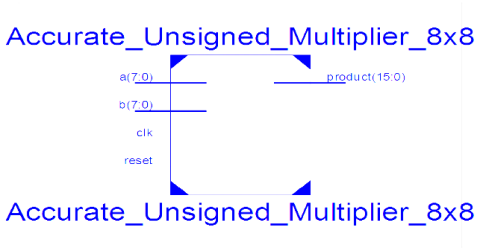


Figure 13. Unsigned accurate multiplier.

```
Timing Detail:
--------------
All values displayed in nanoseconds (ns)

================================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 11 / 2
--------------------------------------------------------------------------------
Delay:               4.110ns (Levels of Logic = 3)
  Source:            i5 (PAD)
  Destination:       o6 (PAD)

  Data Path: i5 to o6
                                Gate     Net
    Cell:in->out      fanout   Delay    Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             1   0.694    0.901  i5_IBUF (i5_IBUF)
     LUT6:I0->O            1   0.086    0.286  o61 (o6_OBUF)
     OBUF:I->O                 2.144           o6_OBUF (o6)
    ----------------------------------------
    Total                      4.110ns (2.924ns logic, 1.186ns route)
                                       (71.1% logic, 28.9% route)

================================================================================
```

Figure 14. Timing report of unsigned accurate multiplier.

The timing report of the unsigned accurate multiplier indicates the timing behavior of the implemented design. The power report of the unsigned accurate multiplier and output waveform are shown in Figures 15 and 16.

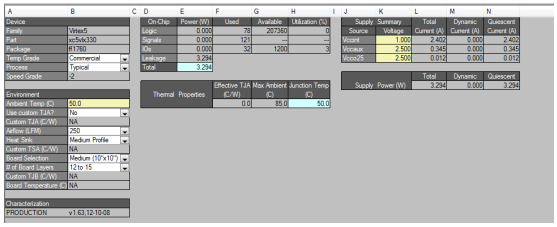


Figure 15. Power report of unsigned accurate multiplier.

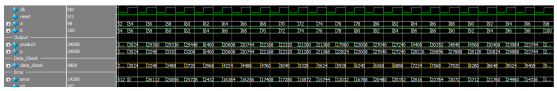


Figure 16. Output waveform of unsigned accurate multiplier.

6. CONCLUSION AND FUTURE SCOPE

This work provides area-optimized, high-performance soft-core exact and approximate multipliers that use LUT6 tables and supporting sum and carry chains in current FPGAs. It is recommended that LUT6 with multiplexers be modified to use one LUT5 with multiplexers rather than dual LUT5s with multiplexers. The proposed multiplier architectures reduce FPGA resource consumption, critical-path delay, and power consumption while preserving useful accuracy for approximate computing applications.

The future scope is to increase accuracy and further decrease the number of critical-path delays. This would improve power consumption and area, which play important roles in circuit design and are promising for image processing, machine learning, and FPGA-based hardware accelerator applications.

REFERENCES

[1] A. Kakacak, A. E. Guzel, O. Cihangir, S. Gören, and H. F. Ugurdag, "Fast Multiplier Generator for FPGAs with LUT based Partial Product Generation and Column/Row Compression," *Integr. VLSI J.*, vol. 57, pp. 147–157, 2017.

[2] J. Beuchat and J. Muller, "Automatic Generation of Modular Multipliers for FPGA Applications," *IEEE Trans. Comput.*, vol. 57, no. 12, pp. 1600–1613, Dec. 2008.

[3] N. Brunie, F. de Dinechin, M. Istoan, G. Sergent, K. Illyes, and B. Popa, "Arithmetic Core Generation Using Bit Heaps," in *Proc. 23rd Int. Conf. Field Program. Log. Appl. (FPL)*, Porto, Portugal, 2013, pp. 1–8.

[4] M. Kumm, "Optimal constant multiplication using integer linear programming," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 65, no. 5, pp. 567–571, May 2018.

[5] P. Paz and M. Garrido, "Efficient Implementation of Complex Multipliers on FPGAs Using DSP Slices," *J. Signal Process. Syst.*, vol. 95, pp. 543–550, Apr. 2023.

[6] S. Ullah, S. Rehman, M. Shafique, and A. Kumar, "High Performance Accurate and Approximate Multipliers for FPGA-Based Hardware Accelerators," presented at the 2021 Design, Autom. Test Eur. Conf. Exhib. (DATE), Grenoble, France, Feb. 2021.

[7] H. Parandeh-Afshar and P. Ienne, "Measuring and Reducing the Performance Gap between Embedded and Soft Multipliers on FPGAs," in *Proc. 21st Int. Conf. Field Program. Log. Appl. (FPL)*, Chania, Greece, 2011, pp. 225–231.

[8] R. Pilipovic and P. Bulic, "The Logarithmic Multiplier Design with Radix-4 Booth Encoding," presented at the 2020 Int. Conf. Comput. Design (ICCD), Hartford, CT, USA, Oct. 2020.

[9] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "FP-BNN: Binarized Neural Network on FPGA," presented at the 2018 Int. Conf. Field Program. Technol. (FPT), Naha, Japan, Dec. 2018.

[10] S. Mittal, "A Survey of FPGA-Based Accelerators for Convolutional Neural Networks," *Neural Comput. Appl.*, vol. 32, pp. 1109–1139, Jan. 2020.

[11] R. Pilipovic, P. Bulic, and W. E. G. Walters, "Array Multipliers for High Throughput in Xilinx FPGAs with 6-Input LUTs," *Computers*, vol. 5, no. 4, Dec. 2020.

[12] J. Faraone, M. Kumm, and H. W. Leong, "AddNet: Deep Neural Networks Using FPGA-Optimized Multipliers," presented at the 2019 Int. Conf. Field Program. Technol. (FPT), Tianjin, China, Dec. 2019.

[13] B. Thiyaneswaran, S. Kumarganesh, M. Dharmalingam, P. N. Palanisamy, L. Vasanth, and A. Immanuvel, "Environmental Pollution and Weather Data Monitoring Using LoRa Low Power VLSI Solution," presented at the 9th Int. Conf. Sci. Technol., 2023.

[14] S. P. Anthoniraj, A. K. Anguraj, S. Kumarganesh, and B. Thiyaneswaran, "Development of Keyless Biometric Authenticated Vehicle Ignition System," *Mater. Today: Proc.*, vol. 81, no. 2, pp. 464–469, 2021.

[15] D. Kalaiyarasi and M. Saraswathi, "Design of an Efficient High Speed Radix-4 Booth Multiplier for Both Signed and Unsigned Numbers," presented at the 2016 Int. Conf. Adv. Electron. Comput. Commun. (ICAECC), Bangalore, India, Oct. 2016.