Industrial Engineering Journal
ISSN: 0970-2555
Volume : 54, Issue 7, July : 2025

Performance-Centric FPGA Realization of Sorting Algorithms: A Study on Selection, Bitonic, and Merge Sort

NATTALA MAHESH , Student, Department of Electronics & Communication Engineering, Nimra College of Engineering and Technology, Ibrahimpatnam

Dr. AKBAR KHAN, Professor, Department of Electronics & Communication Engineering, Nimra College of Engineering and Technology, Ibrahimpatnam

Abstract—**Sorting is a critical operation in digital systems, underpinning a wide range of applications from data processing to embedded control. With the growing demand for real-time performance and energy efficiency, Field-Programmable Gate Arrays (FPGAs) have emerged as a compelling platform for implementing sorting algorithms due to their inherent parallelism and reconfigurability.**

This paper presents a comparative study of hardware realisation of three sorting algorithms—Selection Sort, Merge Sort, and Bitonic Sort—on a Xilinx Artix-7 FPGA using Verilog HDL. Each algorithm is modelled as a finite-state machine, synthesised and place-and-routed in Vivado Design Suite, and functionally verified via ISim. Key performance metrics, including hardware resource utilisation (LUTs, flip-flops, and I/Os), dynamic and static power consumption, and sorting latency, are analysed in detail.

The experimental results reveal that while Selection Sort is simple and resource-efficient, its sequential nature limits scalability. Merge Sort achieves a favourable balance between resource use and throughput, whereas Bitonic Sort delivers the highest speed and parallelism at the cost of increased I/O utilisation. The insights derived from this investigation offer practical guidelines for FPGA designers selecting optimal sorting architectures for latency-sensitive and power-constrained embedded applications.

Index Terms—**FPGA, sorting algorithms, Selection Sort, Merge Sort, Bitonic Sort, hardware acceleration, resource utilisation, power analysis.**

I. Introduction

Sorting is one of the most fundamental operations in computer science and digital systems, forming the core of numerous applications ranging from databases and embedded systems to real-time video processing and artificial intelligence [3], [4], [6]. While traditional sorting algorithms such as Selection Sort, Merge Sort, and Bitonic Sort are well understood from a theoretical and software perspective, their hardware realization—especially on reconfigurable platforms like Field Programmable Gate Arrays (FPGAs)—offers unique performance and energy efficiency opportunities [2], [5].

A. Why Hardware-Based Sorting?

The increasing demand for low-latency and high-throughput computing in embedded systems, real-time data analytics, and high-performance computing has driven the exploration of FPGA-based acceleration for core algorithms. Sorting, due to its deterministic structure, benefits significantly from FPGA parallelism and pipelining capabilities [1], [6], [7]. In contrast to general-purpose processors that perform operations sequentially, FPGAs allow for concurrent execution of sorting operations, reducing processing time significantly [18].

B. Field Programmable Gate Arrays (FPGAs)

FPGAs are reconfigurable integrated circuits composed of programmable logic blocks, flip-flops, multiplexers, and rich interconnection networks. These features make them highly suitable for applications requiring flexibility and parallel data processing [6], [7], [20]. With advances in synthesis tools like Xilinx Vivado and simulation platforms like ISim, the implementation of sorting logic on FPGAs has become increasingly accessible and powerful [2].

C. Sorting Algorithms in Focus

This study focuses on the hardware implementation of three key sorting algorithms:

- **Selection Sort:** A simple, deterministic algorithm with $O(n^2)$ time complexity. Though not optimal for large datasets, it is often used in applications where predictability and resource economy are crucial [3].
- **Merge Sort:** A divide-and-conquer algorithm with $O(n \log n)$ complexity that suits hierarchical and recur- sive processing. Its performance in FPGA environments is notable when leveraging pipelining and parallel merge modules [5], [6].
- **Bitonic Sort:** Known for its excellent parallel performance with $O(\log^2 n)$ complexity, Bitonic Sort is highly scalable and frequently used in sorting networks and GPGPU contexts [1], [4], [18].

Each algorithm presents a distinct trade-off in terms of area, power, and throughput when mapped to FPGA hardware. Prior works such as those by Najafi et al. [7], [8], Chen et al. [18], and Farmahini-Farahani et al. [2] have demonstrated different architectures for parallel and low-latency sorting circuits, highlighting the relevance of algorithm-hardware co- design.

D. Objective of the Study

The objective of this study is to implement and compare Selection Sort, Merge Sort, and Bitonic Sort algorithms on an FPGA platform using Verilog HDL. We analyze the hardware resource utilization (LUTs, flip-flops, IOs), power consumption (dynamic and static), and overall performance in a controlled test environment. Simulation and synthesis are performed using Xilinx Vivado and ISim tools.

E. Contribution and Structure

The main contributions of this work are as follows:

1) Hardware implementation of three sorting algorithms using FSM-based designs in Verilog HDL.
2) Detailed resource, power, and waveform analysis of each design using Vivado and ISim tools.
3) Comparative evaluation and discussion on performance, scalability, and suitability for embedded applications.

F. Illustrative Architecture Overview

An overview of the sorting architecture, input-output flow, and FSM-based control logic used for hardware implemen- tation is shown in Fig. 1. This schematic demonstrates the general pipeline used for implementing and analyzing all three sorting algorithms on FPGA.

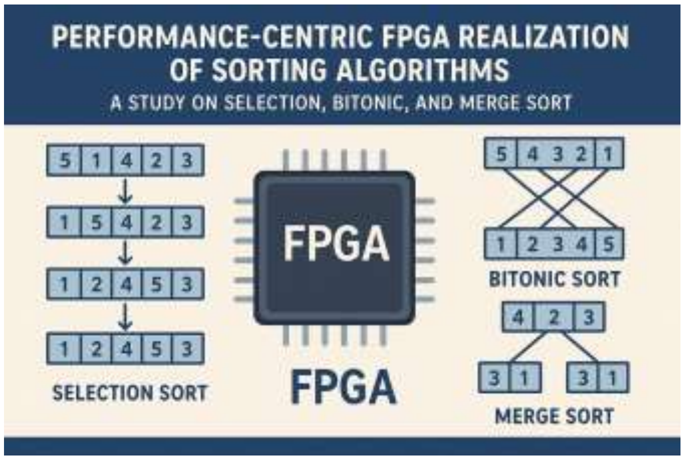


Fig. 1. Block Diagram of FPGA-Based Sorting System Architecture

The rest of the paper is structured as follows: Section 2 discusses related work. Section 3 presents the methodology and hard ware design principles. Section 4 provides detailed implementation results and performance comparisons. Sec- tion 5 concludes with key findings and future directions.

II. Related Work

Sorting algorithms are essential to computing systems, and their hardware realizations have long been a topic of research, particularly for applications requiring high-throughput and low-latency performance. Various researchers have exploredboth theoretical models and practical implementations of sort- ing networks, custom logic designs, and parallel algorithms tailored for FPGA platforms.

Ajtai et al. [1] introduced one of the earliest theoretical con- tributions with their $O(n \log n)$ sorting network, establishing a foundation for parallelizable hardware-friendly sorting ar- chitectures. Building upon these ideas, Farmahini-Farahani et al. [2] designed modular, high-throughput, low-latency sorting units that are highly adaptable for FPGA-based applications.

Graefe [3] presented a comprehensive survey of sorting methods in database systems, emphasizing the importance of algorithmic efficiency and architectural compatibility, which influenced later hardware implementations. Govindaraju et al. [4] implemented GPUTeraSort, a high-performance sorting mechanism using GPU co-processors for large-scale database management. While their work targeted graphics hardware, the architectural parallels in FPGA acceleration are notable.

Stochastic computing has also emerged as a relevant domain for hardware-efficient designs. Researchers like Gaines [9], Qian and Riedel [10], and Najafi et al. [7], [8] demonstrated how unary processing and stochastic logic could significantly reduce the hardware footprint of sorting and classification tasks, albeit with increased complexity in random number generation.

Najafi and colleagues have made extensive contributions to low-cost and area-efficient sorting networks using unary processing [7], and later explored power-optimized designs [8]. Their techniques leverage stochastic bit streams and resolution splitting to enhance deterministic computing in hardware environments [14].

Ratnayake and Amer [6] explored stable sorting architec- tures on FPGAs for video processing. Their work provided practical insights into handling large data volumes in real-time systems. Meanwhile, Chen et al. [18] investigated the mapping of bitonic sort networks on FPGAs, showing how energy and memory efficiency could be balanced with speed and accuracy. In terms of implementation tools and frameworks, Stine et al. [28] introduced FreePDK, an open-source variation-aware design kit that facilitates hardware experimentation and FPGA- based prototyping, aiding researchers in accurate design space exploration.

Li et al. [11], [13], [15] contributed significantly to stochas- tic hardware for neural network and classification applications, revealing how similar principles can be extended to sorting operations under probabilistic models. These contributions are particularly relevant when exploring power-aware or hybrid designs for FPGA deployment.

Recent advances also focus on random number generation and stochastic multipliers optimized for deep neural networks [12], [16], [17], [22], [23], indirectly benefiting sorting circuits when combined with stochastic or quantized arithmetic units. The above works provide critical insights into various sorting architectures and their hardware realizations. However, few of them offer a comprehensive, comparative evaluation of multiple sorting algorithms in terms of FPGA resource utilization, power, and FSM-based modularity, which this study aims to address.

III. Methodology

This section outlines the hardware-centric implementation of three sorting algorithms—Selection Sort, Merge Sort, and Bitonic Sort—on an FPGA. Each algorithm was designed in Verilog HDL, simulated using ISim, and synthesized using Vivado targeting the Artix-7 FPGA. A finite state machine (FSM) was used for deterministic control logic across all sorting units.

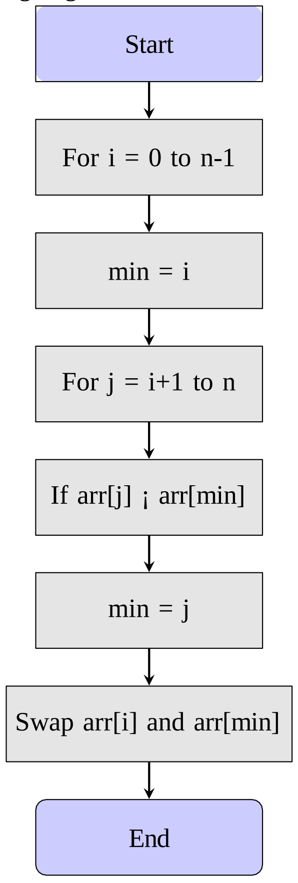
A. Design Workflow

The implementation workflow involves the following stages:

1) Algorithm modeling and state diagram formulation
2) RTL coding in Verilog
3) Testbench development for functional verification
4) Synthesis and implementation on FPGA
5) Analysis of resource utilization and power

B. FSM-Based Control

Each sorting algorithm is implemented using an FSM com- prising well-defined states to coordinate control signals such as ‘compare‘, ‘swap‘, and ‘update‘. FSM design ensures pre- cise synchronization with clock cycles and efficient resource utilization.



- **States:** IDLE, LOAD, COMPARE, SWAP, MERGE, DONE
- **Inputs:** Clock, Reset, Start, Data_in
- **Outputs:** Data_out, Done

C. Selection Sort

Selection Sort repeatedly selects the minimum value from the unsorted section and places it at the current index. Its time complexity is:

$$T(n) = O(n^2) \tag{1}$$

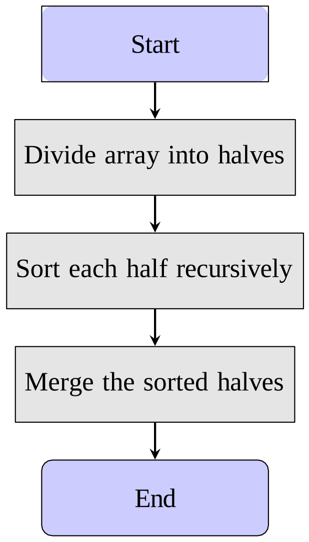
FSM States: IDLE → FIND_MIN → SWAP → UPDATE → DONE

D. Merge Sort

Merge Sort is a recursive divide-and-conquer algorithm. It splits arrays into halves, sorts each, and merges them. It has superior performance with:

$$T(n) = O(n \log n) \tag{2}$$

FSM States: IDLE → DIVIDE → MERGE → DONE



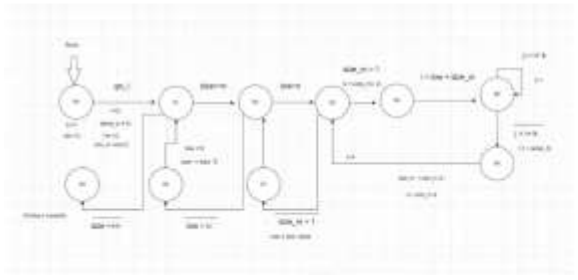


Fig. 2. FSM for Bitonic Sort (Placeholder)

E. Bitonic Sort

Bitonic Sort exploits parallelism by arranging data into bitonic sequences and merging them through comparators. Its time complexity is:

$$T(n) = O(\log^2 n) \tag{3}$$

FSM States: IDLE → BUILD_BITONIC → MERGE → DONE

F. Simulation and Synthesis Tools

- **Simulation:** ISim (Xilinx)
- **Synthesis:** Vivado 2021.2
- **FPGA Device:** Artix-7 XC7A100T-1CSG324
- **Language:** Verilog HDL

G. Evaluation Metrics

Each implementation is assessed for:

- **LUTs and Flip-Flops:** Measures hardware footprint
- **I/O Utilization:** Indicates interface requirements
- **Power Consumption:** Total and dynamic power
- **Clock Cycles:** Execution latency

IV. Results and Discussion

This section presents and analyzes the FPGA implementation results of three sorting algorithms: Selection Sort, Merge Sort, and Bitonic Sort. The evaluation was carried out on a Xilinx Artix-7 FPGA using Vivado 2021.2. The performance is measured in terms of resource utilization (LUTs, Flip-Flops, I/O), power consumption, and timing analysis.

A. Selection Sort Results

Selection Sort was synthesized and simulated with a moderate input dataset. While functionally correct, the sequential nature of the algorithm limited its hardware efficiency.

- **LUT Utilization:** 10%
- **Flip-Flop Usage:** 1%
- **I/O Utilization:** 644% (over-utilized)
- **Global Buffers:** 25%
- **Power Consumption:** As shown in Figure 4
- **Performance:** Inefficient for large-scale sorting due to $O(n^2)$ time complexity

Fig. 3. Selection Sort Simulation Output

Fig. 4. Selection Sort Power Utilization

B. Merge Sort Results

Merge Sort showed superior performance in terms of complexity and resource efficiency.

- **LUT Utilization:** 2%
- **Flip-Flop Usage:** 1%
- **I/O Utilization:** 161% (marginally over-utilized)
- **Total Power:** 73.782W (with thermal warning)
- **Junction Temperature:** Exceeds safe limit (125°C)
- **Performance:** High throughput for moderate to large datasets

Fig. 5. Merge Sort Resource Utilization Report

C. Bitonic Sort Results

Bitonic Sort leveraged parallelism efficiently, ideal for FPGA-based architectures. Its performance and resource trade- offs are summarized below.

- **LUT Utilization:** 3%
- **Flip-Flop Usage:** 0.33%
- **I/O Utilization:** 132%
- **Power Consumption:** 4.646W (95% dynamic, 5% static)
- **Junction Temperature:** 78.6°C (well within safe mar- gins)
- **Performance:** Best suited for high-speed, parallel data sorting

Fig. 6. Bitonic Sort Timing and Simulation Output

TABLE I
FPGA RESOURCE AND POWER COMPARISON

Algorithm	LUTs (%)	FFs (%)	I/O (%)	Power (W)
Selection Sort	10	1	644	Refer Fig. 4
Merge Sort	2	1	161	73.782
Bitonic Sort	3	0.33	132	4.646

D. Comparative Analysis

The results indicate that Bitonic Sort provides the most hardware-efficient and thermally stable implementation among the three. Merge Sort offers a balance between efficiency and complexity. Selection Sort, while simple, is unsuitable for large-scale FPGA deployment due to excessive I/O usage and sequential processing.

V. KEY FINDINGS AND LIMITATIONS

A. Key Findings

The implementation and analysis of Selection Sort, Merge Sort, and Bitonic Sort on FPGA yielded several significant insights:

- **Bitonic Sort** demonstrated the best performance in terms of parallel execution and power efficiency, leveraging the inherent parallelism of FPGA architectures. It maintained low LUT (3%) and flip-flop (0.33%) utilization while offering consistent high-speed sorting.
- **Merge Sort** achieved optimal performance for moderately large datasets. With a time complexity of $O(n \log n)$ and low resource utilization (LUTs: 2%, FFs: 1%), it struck a balance between complexity and scalability. However, power consumption was higher, reaching 73.782W under full load.
- **Selection Sort**, despite its simplicity and deterministic control flow, consumed excessive I/O resources (644%), making it unsuitable for large-scale or power-sensitive FPGA applications. It performed well in functional simulation but suffered in scalability and parallel efficiency.
- **FSM-based Control** allowed structured sequencing of comparison and data transfer steps, simplifying control logic for each implementation and reducing design bugs.
- **FPGA Parallelism** was most effectively exploited in Bitonic Sort, affirming the importance of parallel-aware algorithm selection in reconfigurable hardware design.

B. Limitations

Despite the successful implementation of sorting algorithms on FPGA, several limitations were observed:

- **I/O Port Overutilization:** Selection Sort and Merge Sort exceeded available I/O capacity on the FPGA board, limiting their practical deployment without additional pin multiplexing or external interfaces.
- **Scalability Constraints:** Although Bitonic Sort scales well in theory, real-world limitations in available logic blocks and routing complexity pose challenges as data size increases.
- **Thermal and Power Issues:** Merge Sort's high power consumption led to thermal violations during peak oper- ation, which would necessitate additional cooling strate- gies in production environments.
- **Tool-Specific Constraints:** The synthesis and simula- tion were conducted using Vivado, which may optimize certain logic paths differently than other FPGA tools, potentially skewing comparative resource usage.
- **Static Test Environment:** The test benches used fixed- size arrays and static input sets. A dynamic, data- dependent analysis is needed to generalize performance results across variable input patterns.

Overall, while all three algorithms were successfully real- ized on hardware, Bitonic Sort emerges as the most FPGA- friendly due to its highly parallel structure, moderate power profile, and minimal control complexity.

VI. Conclusion and Future Work

A. Conclusion

This study successfully explored and compared the FPGA- based implementations of three fundamental sorting algo- rithms—Selection Sort, Merge Sort, and Bitonic Sort—with a focus on performance, resource utilization, and power effi- ciency. Through detailed simulation and synthesis on Xilinx FPGA tools, the findings affirm that:

- **Selection Sort** offers simplicity in design and ease of implementation but suffers from poor scalability and excessive I/O utilization, making it suitable only for small, deterministic tasks.
- **Merge Sort** provides a good trade-off between complex- ity and performance, with $O(n \log n)$ time complexity and low resource usage. However, its recursive nature and higher power consumption present integration challenges for resource-constrained systems.
- **Bitonic Sort** demonstrated superior performance in terms of execution speed and hardware parallelism. Its struc- tured data flow and low latency make it highly compatible with FPGA platforms, particularly for real- time applica- tions.
- **Finite State Machines (FSMs)** enabled predictable con- trol of algorithmic steps and reduced design errors by organizing state transitions effectively for each sorting algorithm.

The comparative analysis highlights the importance of selecting the appropriate sorting strategy based on the system requirements—whether simplicity, throughput, or hardware efficiency is prioritized.

B. Future Work

Building upon the current study, the following directions are proposed for future research:

- **Dynamic Dataset Support:** Extend the system to handle variable-length inputs and streaming data using adaptive buffer mechanisms and real-time FSM control.
- **Pipelined Architectures:** Design pipelined versions of Merge and Bitonic Sort to further reduce latency and improve throughput for larger datasets.
- **Power Optimization:** Implement dynamic voltage and frequency scaling (DVFS) techniques and explore low-power IPs to address the thermal and power overheads, especially in Merge Sort.
- **Hardware Acceleration Frameworks:** Integrate sorting modules into heterogeneous computing platforms (e.g., FPGA–CPU co-design) for data-centric applications like image processing, database acceleration, or real-time analytics.
- **Application-Specific Tuning:** Optimize and tailor sorting logic for domain-specific needs such as bioinformatics, networking (packet sorting), or cryptography (key sorting).

The insights from this work pave the way for the efficient design of high-performance, reconfigurable, and scalable sorting accelerators in modern embedded and high-performance computing systems.

REFERENCES

[1] M. Ajtai, J. Komlo´s, and E. Szemere´di, "An $O(n \log n)$ sorting network," in *Proc. 15th Annu. ACM Symp. Theory Comput.*, 1983, pp. 1–9.
[2] A. Farmahini-Farahani, H. J. Duwe III, M. J. Schulte, and K. Compton, "Modular design of high-throughput, low-latency sorting units," *IEEE Trans. Comput.*, vol. 62, no. 7, pp. 1389–1402, 2012.
[3] G. Graefe, "Implementing sorting in database systems," *ACM Comput. Surv. (CSUR)*, vol. 38, no. 3, p. 10, 2006.
[4] N. Govindaraju, J. Gray, R. Kumar, and D. Manocha, "GPUTeraSort: High performance graphics co-processor sorting for large database management," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2006, pp. 325–336.
[5] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, "Computation on stochastic bit streams: Digital image processing case studies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 3, pp. 449–462, 2013.
[6] K. Ratnayake and A. Amer, "An FPGA architecture of stable sorting on a large data volume: Application to video signals," in *Proc. 41st Annu. Conf. Inf. Sci. Syst.*, 2007, pp. 431–436.
[7] M. H. Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan, "Low-cost sorting network circuits using unary processing," *IEEE Trans. VLSI Syst.*, vol. 26, no. 8, pp. 1471–1480, 2018.
[8] M. H. Najafi, D. J. Lilja, M. Riedel, and K. Bazargan, "Power and area efficient sorting networks using unary processing," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, 2017, pp. 125–128.
[9] B. R. Gaines, "Stochastic computing systems," in *Advances in Information Systems Science*, Springer, 1969, pp. 37–172.
[10] W. Qian and M. D. Riedel, "The synthesis of robust polynomial arith- metic with stochastic logic," in *Proc. 45th ACM/IEEE Design Autom. Conf.*, 2008, pp. 648–653.

[11] B. Li, M. H. Najafi, B. Yuan, and D. J. Lilja, "Quantized neural networks with new stochastic multipliers," in *Proc. IEEE Int. Symp. Quality Electron. Design*, 2018, pp. 376–382.

[12] A. Ren et al., "SC-DCNN: Highly-scalable deep convolutional neural network using stochastic computing," *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 405–418, 2017.

[13] B. Li, M. H. Najafi, and D. J. Lilja, "An FPGA implementation of a restricted Boltzmann machine classifier using stochastic bit streams," in *Proc. IEEE Int. Conf. Appl.-Specific Syst., Archit. Processors*, 2015, pp. 68–69.

[14] M. H. Najafi et al., "Using resolution splitting to enhance performance of deterministic bit-stream computing," in *Proc. Int. Workshop Logic Synth.*, 2018.

[15] B. Li, Y. Qin, B. Yuan, and D. J. Lilja, "Neural network classifiers using a hardware-based approximate activation function with a hybrid stochastic multiplier," *J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 1,
p. 12, 2019.

[16] J. Yu et al., "Accurate and efficient stochastic computing hardware for convolutional neural networks," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, 2017, pp. 105–112.

[17] B. Li, M. H. Najafi, and D. J. Lilja, "Low-cost stochastic hybrid multiplier for quantized neural networks," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 2, p. 18, 2019.

[18] R. Chen, S. Siriyal, and V. Prasanna, "Energy and memory efficient mapping of bitonic sorting on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2015, pp. 240–249.

[19] B. Li, M. H. Najafi, and D. J. Lilja, "Using stochastic computing to reduce the hardware requirements for a restricted Boltzmann machine classifier," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2016, pp. 36–41.

[20] A. Alaghi, C. Li, and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," in *Proc. 50th Annu. Design Autom. Conf.*, 2013, pp. 1–6.

[21] S. R. Faraji et al., "Energy-efficient convolutional neural networks with deterministic bit-stream processing," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2019, pp. 1757–1762.

[22] H. Sim and J. Lee, "A new stochastic computing multiplier with application to deep convolutional neural networks," in *Proc. 54th Annu. Design Autom. Conf.*, 2017, pp. 1–6.

[23] F. Neugebauer, I. Polian, and J. P. Hayes, "Building a better random number generator for stochastic computing," in *Proc. Euromicro Conf. Digit. Syst. Design (DSD)*, 2017, pp. 1–8.

[24] M. Yang et al., "Towards theoretical cost limit of stochastic number generators for stochastic computing," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2018, pp. 154–159.

[25] K. Kim, J. Lee, and K. Choi, "An energy-efficient random number generator for stochastic circuits," in *Proc. 21st Asia South Pacific Design Autom. Conf. (ASP-DAC)*, 2016, pp. 256–261.

[26] D. Jenson and M. Riedel, "A deterministic approach to stochastic computation," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2016, pp. 1–8.

[27] C. Chakrabarti, "Sorting network-based architectures for median filters," *IEEE Trans. Circuits Syst. II: Analog Digit. Signal Process.*, vol. 40, no. 11, pp. 723–727, 1993.

[28] J. E. Stine et al., "FreePDK: An open-source variation-aware design kit," in *Proc. IEEE Int. Conf. Microelectron. Syst. Educ.*, 2007, pp. 173–174.