# BuzzSort: A Linear-Time, Event-Driven Data Conversion and Sorting Framework for Approximate Computing Architectures

Swagat Bhattacharyya, Linhao Yang, and Jennifer O. Hasler
*School of Electrical and Computer Engineering*
*Georgia Institute of Technology, Atlanta, GA 30332*
*Email: {sbhattac8,lyang319,ph67}@gatech.edu*

***Abstract*—Analog computational primitives, such as vector-matrix multipliers (VMMs), are foreseen to play a pivotal role in economizing computing; however, to improve the viability of general-purpose accelerators, there is a need for efficient data conversion and sorting during readout. This work introduces "BuzzSort," an event-driven framework that simultaneously converts and sorts data from analog systems. BuzzSort acquires analog data, retrieves sorting indices, and produces a sorted output vector in linear time. We experimentally demonstrate and characterize the efficacy of BuzzSort with a field-programmable analog array (FPAA) in a 350 nm process and a field-programmable gate array (FPGA).**

## 1. Universal Readout in Analog Computing

Analog computing platforms hold a distinct advantage for applications where low delay, power draw, and numerical error are crucial [1]. This advantage is exemplified by the pivotal role of analog vector-matrix multipliers (VMMs) as building blocks in emerging computational applications ranging from machine learning [2] to specialized tasks like beamforming [3]. In VMM readout, there is often a need to selectively prioritize or prune outputs to balance the time or energy cost of the digitization with numerical accuracy. VMM output priority is typically contingent on the corresponding values, effectively reducing the task of prioritization to that of sorting analog values.

VMM readout strategies are also application-specific; certain tasks, such as deep neural networks, may require A-D conversion of VMM outputs for subsequent computations [4], while other tasks, such as single-layer classifiers might only need the maximum output value from the VMM, often obtained through winner-take-all (WTA) circuits [5]. In specialized applications like beamforming [3], both output values and their sorted order are crucial. This diverse set of sorting-based readout and processing requirements is not unique to VMMs and extends to other analog cells, such as image sensors used in computer vision or Hopfield networks employed for energy surface optimization [6].

While catering to the differing demands poses considerable design challenge, an efficient readout mechanism that meets these requirements would catalyze the commercial adoption of various types of analog computational units, including VMMs. Any proposed readout solution should ideally be universally compatible (including on both mixed-signal and digital domains). To prevent performance bottlenecks, it is of utmost importance for the time complexity

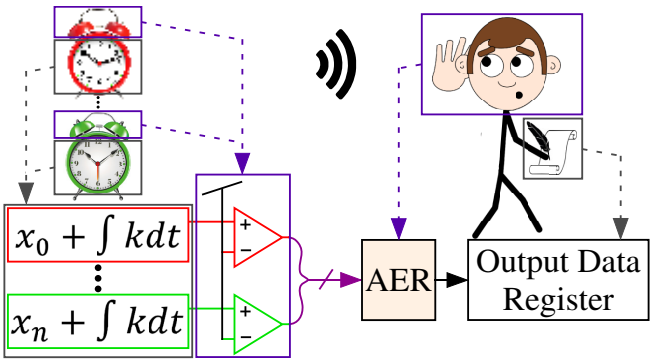


Figure 1. Human perception inherently sorts while observing physical phenomena; for example, one can determine the relative lag of unsynchronized alarm clocks with a preset alarm in linear time by observing the order in which the clocks "buzz." Such event-driven sorting approaches directly map to hardware, of which, one implementation is shown. Building on the event-driven paradigm in human perception, this work introduces a novel framework named BuzzSort to facilitate analog data conversion and sorting.

of the proposed readout and sorting algorithms to scale efficiently with the number of analog computational elements. Analog settling time often scales as $\mathcal{O}(M+N)$ (as in the case of VMMs, where $M$ and $N$ are the number of rows and columns, respectively), or faster in programmable analog hardware, where the bias currents can be flexibly scaled [1]. Although analog-digital converters (ADCs) can operate in linear time or faster [7], sorting outputs from VMMs or other analog cells in linear time or faster is difficult – this would require the formulation of parallel, comparison-free sorting algorithms adaptable to a diverse array of approximate computing platforms [8].

To address these challenges, we introduce an event-driven algorithm hereafter referred to as "BuzzSort." Inspired by human perception [Fig. 1], BuzzSort is an integrated data conversion and sorting algorithm optimized for analog cells related to sleep sort, spaghetti sort, and counting sort. Like sleep sort, spaghetti sort, and counting sort [9], BuzzSort yields a final sorted output vector and the sort indices in $\mathcal{O}(n)$ time. Our algorithm has been experimentally verified using an in-house System on Chip (SoC) field-programmable analog array (FPAA) implemented in a $350\,\mathrm{nm}$ process and a field-programmable gate array (FPGA), confirming its efficacy. To the best of our knowledge, this work is the first demonstration of sorting on an analog/mixed-signal platform and the first demonstration of sorting on a physical-computing platform in linear time.

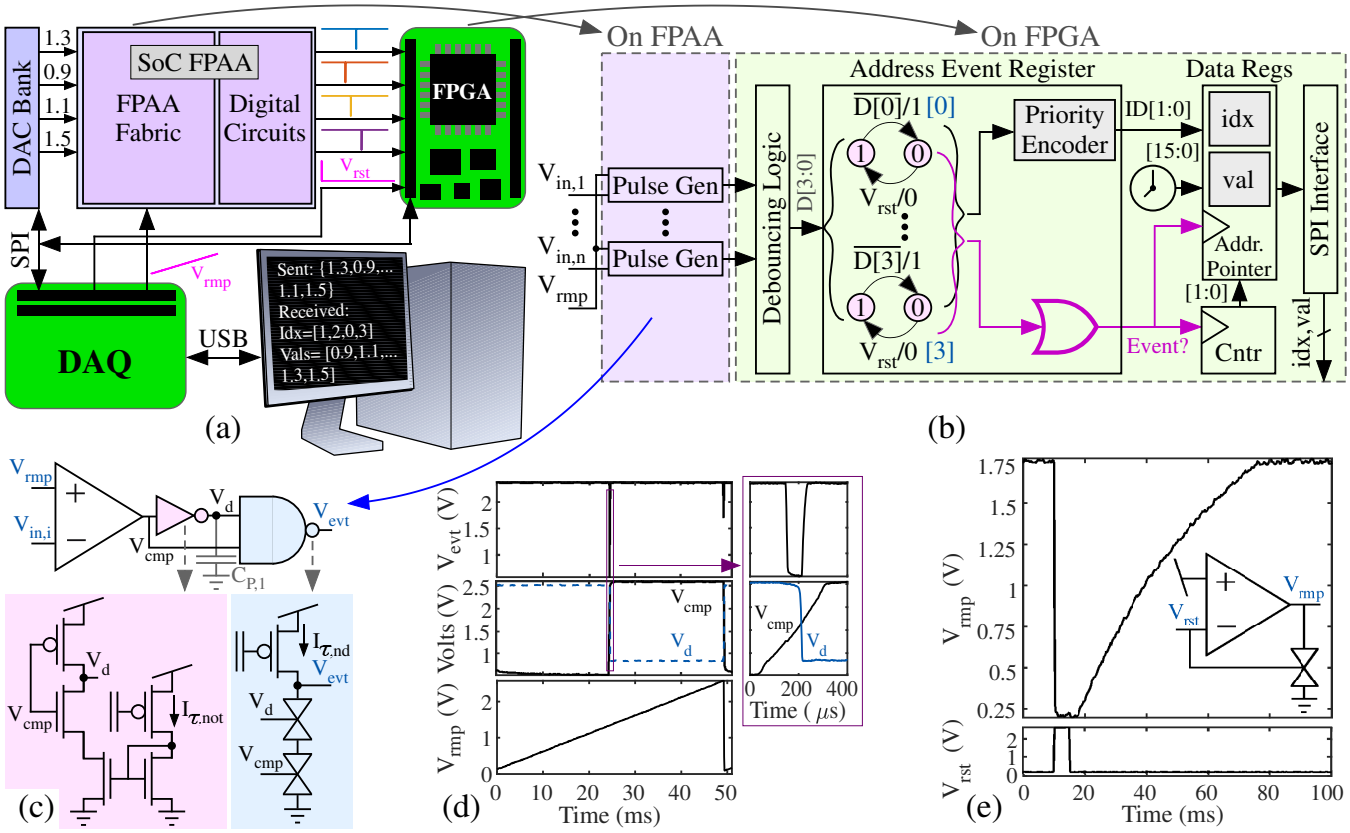


Figure 2. BuzzSort hardware implementation details and subcircuit characterization. Signal flow diagram of the (a) complete testbench utilized for algorithm verification and the (b) major subcircuits and their inter-connectivity. Pulse generator (c) schematic and (d) transient response as $V_{rmp}$ crosses $V_{in,i}$. (e) Ramp generator schematic and transient response. Both circuits in (c) and (e) can reside within a single CAB; $C_{P,1}$ is a parasitic capacitance. In (e), the ramp generator operates with a 0.1 nA bias current, yielding a slightly non-linear, albeit monotonic ramp, which suffices in our application.

The rest of this work is organized as follows. Section 2 motivates BuzzSort. Section 3 elaborates on hardware-level details and experimentally characterizes a mixed-signal implementation of BuzzSort. Section 4 showcases the performance of BuzzSort in a typical analog readout application and section 5 discusses how BuzzSort compares with contemporary approaches. Section 6 concludes the paper.

## 2. Foundations

BuzzSort uses strategies found in sleep, spaghetti and counting sorts, such as time evolution, parallel comparison and count-based storage. Consider the following gedanken-experiments in the context of physical-computing platforms:

- Consider sorting $n$ unsynchronized 24-hour clocks (preset to buzz at midnight) based on their current time. One could simply observe the sequence of buzzes.
- Imagine sorting $n$ finite-length spaghetti sticks by length. Standing the bundle on a table and obscuring the top using one's hand, one could slowly lower one's hand, observing the order in which the sticks appear.

The gedankenexperiments underscore that parallel dynamical evolution and comparison yield effective sorting, provided that values are bounded. This event-driven approach offers configurable trade-offs among sorting time, sorting accuracy, resolution, item count ($n$), and power draw, enabling application-specific optimization. These trade-offs are empirically examined in the following sections.

## 3. Circuit Architecture and Characterization

In our hardware testbench, shown in Fig. 2(a), analog circuits on an FPAA handle item evolution and event generation, while address-event register (AER)-based digital logic on an FPGA observes and records the event sequence [Fig. 2(b)]. In our demonstrations, we sort four voltages supplied from an digital-analog converter (DAC). All runtime control/communication of/with the DAC, FPAA, or FPGA is handled through a data-acquisition system (DAQ).

### 3.1. Analog-Domain Circuits

In the analog domain, our in-house FPAA, equipped with 98 programmable computational analog blocks (CABs) and a nonvolatile floating-gate (FG) pFET routing fabric, houses $n$=4 *pulse generators* and one *ramp generator*. Subcircuit bias currents, also generated with FG pFETs, can be programmed to 13-bit precision using hot-electron injection.

For reasons that will be subsequently elucidated, the AER dictates sorting performance in large-scale systems; thus, the analog circuit architecture is intricately tied to the application-specific design of the digital logic. Our event detector in this work is a pulse generator, which grants enhanced flexibility in configuring the AER subsystem. For

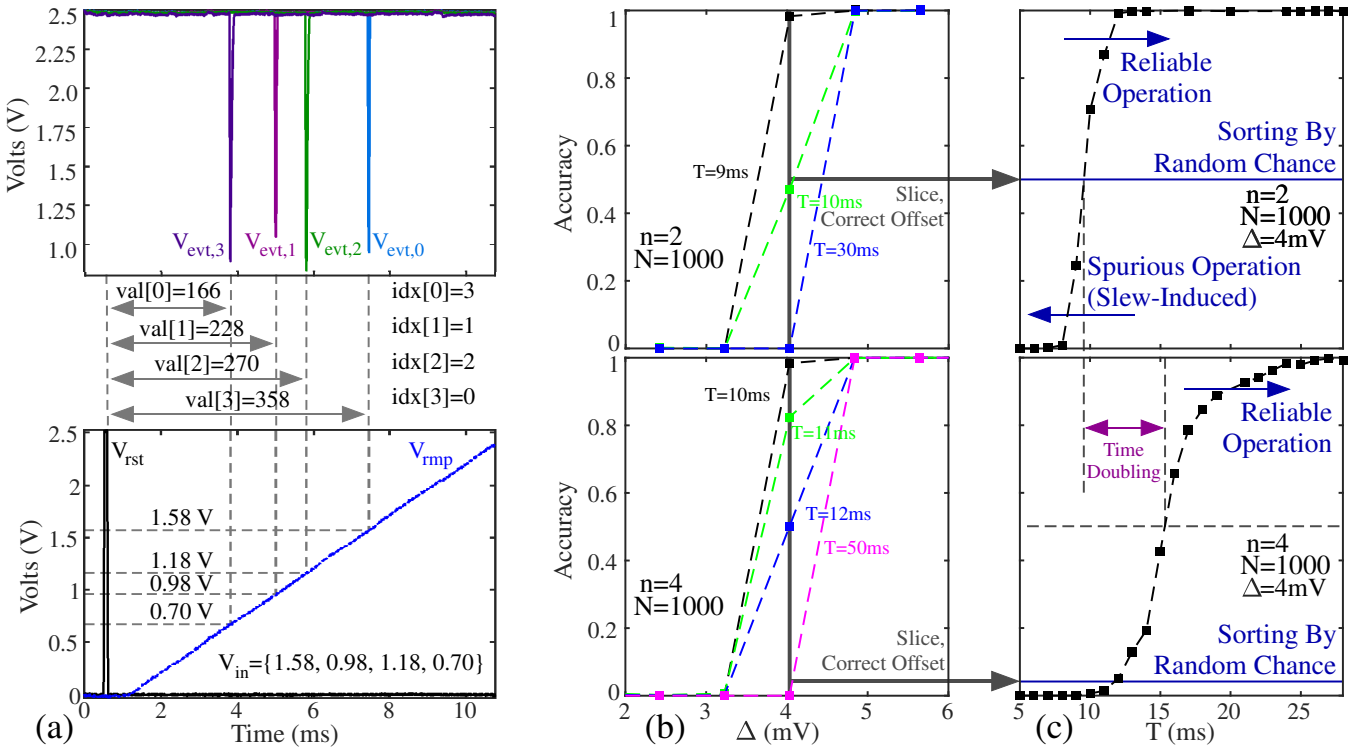


Figure 3. (a) Temporal evolution of $V_{evt,:}$ and resultant sorted outputs given a representative input array and control signals. BuzzSort accuracy over $N = 1000$ trials versus (b) the voltage differential $\Delta$ between input voltages $V_{in,i}$ for several $V_{rmp}$ periods $T$ and (c) $T$ for a fixed $\Delta$ =4 mV. In the top row of subplots, $n = 2$, with $V_{in,2} - V_{in,1} = \Delta$; in the bottom row, $n = 4$, with $V_{in,2} - V_{in,1} = \Delta$ and $V_{in,1} - V_{in,0} = V_{in,3} - V_{in,2} = \Delta/2$. As elucidated in the body, these plots highlight the crucial influence of comparator offsets and timing jitter on BuzzSort performance.

example, in applications with low resolution or speed requirements, the state machine in the AER may be bypassed in favor of a simpler configuration comprising a multi-input OR gate and a priority encoder; in systems subjected to rigorous latency constraints, latency can be minimized by bypassing the delay elements in the pulse generator.

The pulse generator, elucidated in Fig. 2(c), is a level-crossing detector that compares the item value $V_{in,i}$ against a rising reference $V_{rmp}$, generating a brief active-low pulse when the two values cross. This functionality is achieved by feeding the comparator output $V_{cmp}$ into a NOT gate to generate a delayed and inverted copy $V_d$ and by using a NAND gate to compare $V_d$ to $V_{cmp}$. The NOT gate is current-starved so the high-low transition time is adjustable, thereby allowing tunability of the NAND gate output pulse width. As shown in Fig. 2(d), the biasing in this work leads to a 120 µs delay between the beginning of a comparator output transient and the output pulse generation (due to comparator slew) and an output pulse width of 75 µs. Delays do not affect performance as long as they are consistent (as ensured through programmable biases) — a consistent delay corresponds to a systematic shift in the values after A-D conversion and does not change the sorted order.

The ramp generator [Fig. 2(e)] produces a rising output $V_{rmp}$ when its input $V_{rst}$ is low, and resets to a low state when $V_{rst}$ is high. By utilizing an operational-transconductance amplifier (OTA) integrator and a T-gate reset, our design can fit within a single CAB alongside a pulse generator. Although separate ramp generators for each input item would mitigate bus delays in large systems, a single ramp generator can be employed in this work given the small problem size. Our energy-efficient design integrates using bussline parasitics; for rise times of 100 ms on a VDD=2.5 V process, the OTA can be biased with roughly 0.1 nA. Note that $V_{rmp}$ is externally supplied during characterization of the integrated system [as in Fig. 3].

### 3.2. Digital-Domain Circuits

Our digital-domain components [Fig. 2(b)] are realized on a Tang Nano 9K (Gowin GW1NR-9) FPGA. These circuits observe pulse events generated by the FPAA circuitry and subsequently infer the order and voltages of the analog inputs $V_{in,i}$. Incoming event pulses are first routed through debouncing logic to ensure noise-free transitions. The debounced inputs $D[3:0]$ are then channeled into an AER-like structure partially comprising four 2-state Mealy machines. Given an event on its debounced bit, each Mealy machine outputs a '1' and undergoes a $1 \rightarrow 0$ state transition, latching to the '0' state until the reset signal $V_{rst}$ is applied on the onset of the next conversion $V_{rmp}$. Mealy machine outputs are processed by a priority encoder to produce an "ID" corresponding to the input event address. Concurrently, a multi-input OR gate generates an "Event Flag."

The "Event Flag" serves a dual purpose: clocking a counter tallying the number of observed events, and triggering write operations to two output data registers: the sorting index register 'idx' and the sorted value register

'val.' The pre-update event count defines the current register write addresses. Here, 'idx' stores the input event addresses ascertained by the AER, and 'val' stores elapsed times from $V_{rst}$ to the respective event. From the perspective of 'val,' our architecture essentially functions like a comparator bank cascaded with time-digital converters and resembles parallel ramp ADCs, which are theoretically explored in [10]. Data retrieval from the FPGA output data registers is facilitated through a serial peripheral interface (SPI).

## 4. Results

During typical BuzzSort operation [Fig. 3(a)], FPAA analog circuits and FPGA digital logic consume $0.26\,\mu\mathrm{W}$ and $0.98\,\mathrm{mW}$, respectively. The maximum sorting speed is constrained by the power budget; however, reducing power is more complex than simply decreasing $T$, as the analog power is predominated by the static draw of the nine-transistor OTA used as the comparator, rather than switching (dynamic) power. In a custom implementation, both analog and digital power could be reduced by orders of magnitude. Notably, given the infrequency of level crossings, alternate comparator designs, like strong-arm latches (which draw near-zero static power) could perform better.

As elucidated by Fig. 3(b-c), we now show that BuzzSort scalability (given requirements on sorting accuracy, number of items, resolution, and timing) is not bottlenecked by the analog circuits in a reconfigurable, programmable platform. While comparator offsets and timing jitter significantly impact sorting performance, in programmable systems, offsets can be largely mitigated (e.g., through use of FG differential pairs, such as in FGOTAs [11]). Fig. 3(b) shows that when the difference among the input voltages $\Delta$ sufficiently exceeds comparator offsets, sorting accuracy is high and insensitive to the $V_{rmp}$ period $T$ and the number of items $n$. This makes BuzzSort a "plug-and-play" solution for sorting tasks requiring below 12 effective-number-of-bits (ENOB) of resolution, which is compliant with the specifications of many approximate computing applications.

If $\Delta$ approximates the inherent comparator offset, the system tends enters a metastable regime, as evidenced around $\Delta = 4\,\mathrm{mV}$ in Fig. 3(b) and further analyzed in Fig. 3(c). In this regime, sorting accuracy has a pronounced sensitivity to timing jitter and comparator slew mismatch. The metastable regime also exhibits undesirable scaling behavior; not only is the sorting accuracy a strong function of $T$, but the minimum $T$ required for accurate sorting also increases strongly with $n$. To avoid operating in this regime, the input resolution should be kept coarser than the largest comparator offset, which in turn ensures that BuzzSort scales favorably. Comparator offset is caused by differential pair mismatch, which can be improved by increasing transistor sizes. Consequently, comparator mismatch bounds the input resolution, considering the $\mathcal{O}(n)$ time and space scaling required from BuzzSort.

Specifically, comparisons occur in $\mathcal{O}(1)$ time outside the metastable regime, irrespective of the number of items $n$ and determined solely by $T$, which in-turn can be tuned over a wide range (according to application requirements) without detriment to accuracy. Since the "analog NAND" output stage within the pulse generator can be analyzed similarly to a common-source amplifier, and since the FG pFET in the "analog NAND" is biased in the subthreshold regime, the output stage sustains a constant power-delay product (PDP) [1]. Consequently, if $n$ grows (proportionally scaling the output bus capacitance), the output stage bias can be power-scaled proportionally to hold bus communication delays constant. These features allow our analog circuits to scale with $\mathcal{O}(1)$ time complexity for design sizes feasible within chip die dimensions.

TABLE 1. COMPARISON OF BUZZSORT TIME AND SPACE COMPLEXITY TO CONTEMPORARY SORTING ALGORITHMS

| Algorithm | Time Complexity | Space Complexity |
|---|---|---|
| **BuzzSort** | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ |
| Quicksort [12] | Average: $\mathcal{O}(n \log n)$ | $\mathcal{O}(1)$ |
| Merge Sort [13] | $\mathcal{O}(n \log n)$ | $\mathcal{O}(n)$ |
| Counting Sort [14] | $\mathcal{O}(n+k), k = range$ | $\mathcal{O}(k)$ |
| Spaghetti Sort [9] | Average: $\mathcal{O}(n)$ | $\mathcal{O}(n)$ |

Hence, BuzzSort scalability is not bottlenecked by the analog circuitry, but rather by the AER paradigm. The AER is responsible for resolving and recording a sequence of events, thereby introducing a trade-off between time and voltage resolution. Thus, analog dynamics may need to be slowed to achieve high sorting accuracy while sorting a large number of items or targeting finer voltage resolution. The distributed Mealy machines within the AER operate locally (and hence with constant time complexity).

The AER also includes a priority encoder and a multi-input OR gate which are implemented as trees of logic gates, resulting in routing-constrained time and space complexities of $\mathcal{O}(n)$ [1]. Additionally, assuming each output data register write operation by the AER completes within a fixed window and occurs at most $n$ times during a full conversion cycle (filling the $n$ 'idx' and 'val' registers), the worst-case time and space complexities of BuzzSort are both $\mathcal{O}(n)$.

## 5. Discussion

Though it is easiest to prove the impact of a work by comparing it to existing approaches, due to the severe scarcity of literature on analog sorting, with the only known exception of spaghetti sort [1, 9], we separately benchmark our algorithms and analog design against classical sorting methods and other analog event detection applications, respectively.

### 5.1. BuzzSort vs Contemporary Sorting Algorithms

While optimizing analog readout performance, it is essential to employ algorithms with a time complexity of $\mathcal{O}(n)$ or better. This rules out all sequential, comparison-based sorting algorithms, such as merge sort and quicksort [15], which have a worst-case time complexity of $\mathcal{O}(n \log n)$.

An alternative lies with comparison-free sorting algorithms like counting sort [14] or sleep sort. Counting sort leverages bounded integer keys, counting occurrences to produce a sorted output array. The worst-case time and space complexities of counting sort are $\mathcal{O}(n+k)$ and $\mathcal{O}(k)$, respectively, with $k$ denoting the input value/key range [16].

TABLE 2. COMPARISON OF BUZZSORT HARDWARE METRICS TO CONTEMPORARY EVENT DETECTOR IMPLEMENTATIONS

| **Application** | **Power** (μW) | **Area** ($\mathbf{mm^2}$) | **Bandwidth** (kHz) |
|---|---|---|---|
| BuzzSort (FPAA Only) | 0.26/Ch | 0.10 [19] | 0-0.1 |
| Vehicle Classification [20] | 46.7 | 2.25 | 0.02-100 |
| Level Crossing ADC [21] | 3.9 | 2.00 | 0.02-20 |
| Voice Detection [18] | 3.49 | 0.25 | 0-16 |
| Wireless Sensor Node [22] | 214 | 2.25 | 0.5-1.4 |

Sorting algorithms that parallelize comparisons, such as spaghetti sort [9], can also achieve a worst-case time complexity of $\mathcal{O}(n)$. Although historically considered impractical, recent developments suggest spaghetti sort can be implemented on analog platforms using WTA circuits [17]. Table 1 presents a detailed time and space complexity comparison of contemporary sorting algorithms to BuzzSort; the worst-case time and space complexity of BuzzSort mirrors the average case of spaghetti sort.

### 5.2. BuzzSort Circuit vs Other Event Detectors

Table 2 compares the performance of BuzzSort's analog event detector with other event detectors operating near the audio frequency range. The analog circuits used in BuzzSort exhibit several-fold lower runtime power draw than the other circuits in Table 2. Moreover, the estimated die area required for implementing the analog-domain BuzzSort circuits is lower than that of the other approaches. This superior level of performance is facilitated by the FPAA, which permits selective utilization of only the essential analog blocks. Note that each pulse generator used in BuzzSort occupies only a miniscule fraction of an FPAA CAB.

However, the low power draw of BuzzSort comes with an operating frequency penalty; to increase operating frequency, FG pFETs must be biased for higher drain currents, consequently elevating power draw. Nevertheless, unlike other event detector applications, achieving a high operating frequency is not as critical for most sorting algorithm applications. For instance, a voice detection circuit [18] must have an input bandwidth that at least encompasses the frequency range of human speech. BuzzSort is instead designed primarily for high spatial bandwidth, that is, sorting a large input item volume, rather than high time-domain bandwidth.

## 6. Conclusion

BuzzSort is a facilitating advancement in the readout of analog cells; by uniquely integrating data conversion and sorting, BuzzSort fulfills diverse requirements across different application scopes and various computing platforms. We conduct hardware-level experiments exploring the efficacy of BuzzSort using an FPAA in a $350\,\mathrm{nm}$ process and a commercially available FPGA. Our results provide strong empirical validation for the unparalleled performance scaling of BuzzSort in approximate computing platforms.

## Acknowledgments

This material is based on work partially supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-2039655. The authors thank Pranav O. Mathews and Afolabi Ige for helpful discussions.

## References

[1] J. Hasler, "Analog architecture complexity theory empowering ultra-low power configurable analog and mixed mode SoC systems," *JLPEA*, vol. 9, no. 1, 2019.

[2] P. Xiao, C. Bennett, B. Feinberg, S. Agarwal, and M. Marinella, "Analog architectures for neural network acceleration based on non-volatile memory," *Appl Phys Rev*, vol. 7, no. 3, 2020.

[3] J. Dugger, P. Smith, M. Kucic, and P. Hasler, "An analog adaptive beamforming circuit for audio noise reduction," in *IEEE ICASSP*, 2012, pp. 5293–5296.

[4] C. Schlottmann and P. Hasler, "A highly dense, low power, programmable analog vector-matrix multiplier: The FPAA implementation," *IEEE JETCAS*, vol. 1, pp. 403–411, Oct 2011.

[5] J. Lazzaro, S. Ryckebusch, M. Mahowald, and C. Mead, "Winner-take-all networks of O(n) complexity," in *Advances Neu Info Proc Syst*, vol. 1, 1988.

[6] P. Mathews and J. Hasler, "Physical computing for hopfield networks on a reconfigurable analog IC," in *IEEE ISCAS*, 2023, pp. 1–5.

[7] M. Pelgrom, *Linear and Time-Based Conversion*. Cham, Switzerland: Springer Intl, 2022, pp. 795–812.

[8] S. Abdel-hafeez and A. Gordon-Ross, "A comparison-free sorting algorithm," in *Intl. SoC Design Conference*, 2014, pp. 214–215.

[9] A. Dewdney, "On the spaghetti computer and other analog gadgets for problem solving," *Sci American*, vol. 250, no. 6, pp. 19–26, 1984.

[10] M. Kucic, "Analog programmable filters using floating-gate arrays," Master's thesis, Georgia Insitute of Technology, Dec 2000.

[11] F. Adil and P. Hasler, "Offset removal from floating gate differential amplifiers and mixers," in *IEEE MWSCAS*, vol. 1, 2002, pp. I–251.

[12] C. Hoare, "Algorithm 64: Quicksort," *Comm ACM*, vol. 4, no. 7, p. 321, Jul 1961.

[13] H. Goldstine and J. Von Neumann, "Planning and coding of problems for an electronic computing instrument, Part II, Volume 2," 1963.

[14] H. Seward, "Information sorting in the application of electronic digital computers to business operations," Master's thesis, Massachusetts Institute of Technology, Digital Computer Laboratory, 1954.

[15] P. Bhalchandra, N. Deshmukh, S. Lokhande, and S. Phulari, "A comprehensive note on complexity issues in sorting algorithms," *Advances Comp Research*, vol. 1, no. 2, pp. 1–9, 2009.

[16] R. Babu, M. Khalid, S. Soni, S. Chowdary Babu, and Mahesh, "Performance analysis of counting sort algorithm using various parallel programming models," *IJCSIT*, vol. 2, pp. 2284–2287, Jan 2011.

[17] J. Hasler and E. Black, "Physical computing: Unifying real number computation to enable energy efficient computing," *Journal of Low Power Electronics and Applications*, vol. 11, no. 2, p. 14, Mar 2021.

[18] H. Noguchi, T. Takagi, M. Yoshimoto, and H. Kawaguchi, "An ultra-low-power VAD hardware implementation for intelligent ubiquitous sensor networks," in *IEEE Wrkshp Sig Proc Sys*, 2009, pp. 214–219.

[19] S. George, S. Kim, S. Shah, J. Hasler, M. Collins *et al.*, "A programmable and configurable mixed-mode FPAA SoC," *IEEE TVLSI*, vol. 24, pp. 1–9, 01 2016.

[20] S. Bunaiyan and F. Al-Dirini, "Real-time analog event-detection for event-based synchronous sampling of sparse sensor signals," in *IEEE MWSCAS*, 2021, pp. 1053–1057.

[21] C. Weltin-Wu and Y. Tsividis, "An event-driven clockless level-crossing ADC with signal-dependent adaptive resolution," *IEEE JSSC*, vol. 48, no. 9, pp. 2180–2190, 2013.

[22] B. Rumberg, D. Graham, V. Kulathumani, and R. Fernandez, "Hibernets: Energy-efficient sensor networks using analog signal processing," *IEEE JETCAS*, vol. 1, no. 3, pp. 321–334, 2011.