# Low Cost Sorting Network Circuits using Unary Processing

M. Hassan Najafi, *Student Member, IEEE,* David. J. Lilja *Fellow, IEEE,*
Marc D. Riedel, *Senior Member, IEEE,*, and Kia Bazargan, *Senior Member, IEEE,*

***Abstract*—Sorting is a common task in a wide range of applications from signal and image processing to switching systems. For applications that require high performance, sorting is often performed in hardware with ASICs or FPGAs. Hardware cost and power consumption are the dominant concerns. The usual approach is to wire up a network of compare-and-swap units in a configuration called a Batcher (or Bitonic) network. Such networks can readily be pipelined. This paper proposes a novel area- and power-efficient approach to sorting networks, based on "unary processing." In unary processing, numbers are encoded uniformly by a sequence of one value (say, 1) followed by a sequence of the other value (say, 0) in a stream of 0's and 1's with the value defined by the fraction of 1's in the stream. Synthesis results of complete sorting networks show up to 92% area and power saving compared to the conventional binary implementations. However, the latency increases. To mitigate the increased latency, the paper uses a novel time-encoding of data. The approach is validated with two implementations of an important application of sorting: median filtering. The result is a low-cost, energy-efficient implementation of median filtering with only a slight accuracy loss, compared to conventional implementations.**

***Index Terms*—Sorting networks, unary processing, time-encoding data, stochastic computing, median filtering, low-cost design.**

## I. Introduction

Sorting is an important task in applications ranging from data mining, to databases [21][20][29], to ATM and communication switching [14][1], to scientific computing [13], to scheduling [47], to artificial intelligence and robotics [7], to image [28], video [42][11] and signal processing [36]. For applications that require high performance, sorting is often performed in hardware with ASICs or FPGAs [12]. Based on the target applications, hardware sorting units vary greatly in the way that they are configured. The number of inputs can be as low as 9 for some image processing applications (e.g. median filtering) or as high as tens of thousands. The data inputs are sometimes binary values, integers or floating-point numbers ranging from 4 to 256-bit precision.

Hardware cost and power consumption are the dominant concerns with hardware implementations. The total chip area is limited in many applications. As fabrication technologies continue to scale, keeping chip temperatures low is an important goal since leakage current increases exponentially with temperature. Power consumption must be kept as low as possible. Developing low-cost, power-efficient hardware-based solutions to sorting is an important goal.

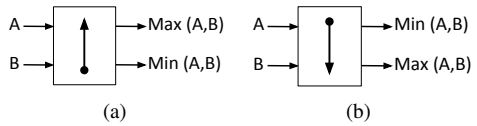



Fig. 1: The schematic symbol of a CAS block a) ascending b) descending

The usual approach is to wire up a network of compare-and-swap (CAS) units in a configuration called a Batcher (or Bitonic) network. Such networks can readily be pipelined. The parallel nature of hardware-based solutions allows them to outperform sequential software-based solutions. The hardware cost and the power consumption depend on the number of CAS blocks and the cost of each CAS block.

This paper proposes a novel area- and power efficient approach to sorting networks based on "unary processing." Data is encoded as serial bit streams, with values represented by the fraction of 1's in a stream of 0's and 1's. This is an evolution of prior work on stochastic processing. Our designs inherit the fault tolerance and low-cost design advantages of stochastic processing while producing completely accurate and deterministic results. As with stochastic processing, however, the approach is handicapped in term of latency. A serial representation is exponentially longer than a conventional binary positional representation.

To mitigate the long latency issue of unary processing, this paper adopts a mixed-signal time-encoding approach recently proposed in [30]. The approach is different to the work on continuous time mixed-signal designs of [22] and [48] in the sense that instead of converting data to (from) binary format by using costly analog to digital (digital to analog) converters and processing in binary domain, the data is encoded in time using low-cost analog to time converters and processed in unary domain. We represent the data with time-encoded pulse signals. The proposed approach is validated with two implementations of an important application of sorting networks: median filtering. Median filtering has been also used in [31] as a case study for processing time-encoded values but no result or discussion on the power consumption and energy efficiency of the designs is presented. Our synthesis results show up to 92% area and power savings compared to conventional weighted binary implementations. Time-encoding the data

A perliminary version of this paper appeared as [32].
M. H. Najafi, D. Lilja, M. Riedel, and K. Bazargan are with the Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities, MN 55455 USA (e-mail: najaf011@umn.edu; lilja@umn.edu; mriedel@umn.edu; kia@umn.edu).

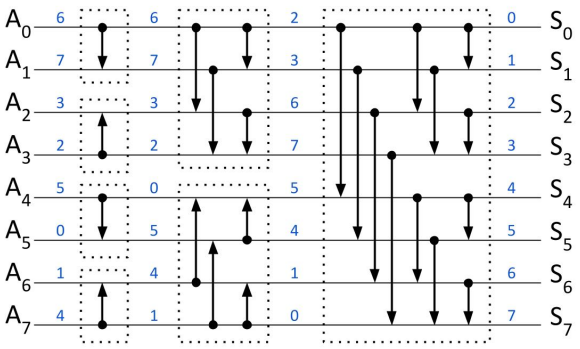



Fig. 2: The CAS network for an 8-input bitonic sorting [17].

provides a significant improvement in the latency and energy consumption with only a slight loss in accuracy.

## II. Background

### A. Sorting Networks

A sorting network is a combination of CAS blocks that sorts a set of input data. Each CAS block compares two input values and swaps the values at the output, if required. There are two variants: an "ascending" type and a "descending" type. Figure 1 shows their schematic symbols. In a conventional design, each CAS block consists of an $M$-bit comparator and two $M$-bit multiplexers, where $M$ is the data-width of the inputs.

Sorting networks are fundamentally different from software algorithms for sorting such as QuickSort, MergeSort, BubbleSort, etc., since the order of comparisons is fixed in advance; the order is not data dependent as is the case with software algorithms. The bitonic and odd-even merge sorting networks proposed by Batcher [6] are two popular configurations of sorting networks [24][26]. They have the lowest known latency for hardware-based sorting [17][2].

Bitonic Sort uses a key procedure called Bitonic Merge (BM). Given two equal size sets of input data, sorted in opposing directions, the BM procedure will create a combined set of sorted data. It recursively merges an ascending and a descending set of size N/2 to make a sorted set of size N [19]. Figure 2 shows the CAS network for an 8-input bitonic sorting network made up of ascending and descending BM units. The total number of CAS blocks in an N-input bitonic sorting is $N \times log_2(N) \times (log_2(N)+1)/4$. Thus, 8-input, 16-input, 32-input, and 256-input bitonic sorting networks require 24, 80, 240, and 4,608 CAS blocks, respectively [17].

An odd-even merge sorting network recursively merges two ascending sequences of length N/2 to make a sorted sequence of length N. Odd-even merge sorting units requires fewer CAS blocks than bitonic sorting units, but often have more complex wiring [17]. Due to their simpler structure, in this paper we will present designs based on bitonic sort networks. The proposed design approach, however, is applicable to any sorting network topology, including odd-even sorting networks; it will accrue the same advantages.

### B. Unary processing

Weighted binary radix has been the dominant format for representing numbers in the field of computer engineering since its inception. The representation is compact; however, computing on this representation is relatively complex, since each bit must be weighted according to its position. Also, the representation is very susceptible to noise: a flipped bit can introduce a large error (if it is a significant bit in the representation.)

Poppelbaum [39] and Gaines [18] introduced stochastic processing based on uniformly distributed random bit streams. All digits have the same weight in this computing paradigm. Numbers are limited to the [0, 1] interval and encoded by the probability of obtaining a one versus a zero in the stream. To represent a real number with a resolution of $2^{-M}$, a stream of $2^M$ bits is required. Beginning in 2001, Brown and Card, [8], [9], and in 2008, Qian et al. reintroduced the concept of stochastic processing to the computer engineering community, [41], [40].

Clearly, a stochastic representation is much less compact than conventional weighted binary; this translates to high latency. However, complex functions can be computed with remarkably simple logic, e.g. multiplication can be performed using a single AND gate. Also, the representation can tolerate high clock skew [33], timing errors [3], and soft logic errors (i.e., bit flips) [40][27][34].

A recent evolution of the idea of stochastic computing has been to perform the processing completely deterministically [23][30][31]. If properly structured, computation on deterministic bit streams can be performed with same circuits as are used in stochastic computing. The results are completely accurate with no random variations; furthermore, the latency is greatly reduced. The idea of unary (or burst) processing was first introduced in 1980s [38] [37] as a hybrid information processing technique that has characteristics common to both conventional binary and to stochastic processing. It is deterministic, but borrows the concept of averaging from stochastic methods. In this paper we apply unary processing to problem of desiging low-cost, power-efficient sorting networks.

**Unary streams.** In unary processing, numbers are encoded uniformly by a sequence of one value (say, 1) followed by a sequence of the other value (say, 0) (See Figure 3). This uniform sequence of bits is called a unary stream. In the literature, this method of encoding is also called pulse-width encoding [15]. As with stochastic streams, all the bits have equal weight. This property provides the immunity to noise. Multiple bit flips in a long unary stream produce small and uniform deviations from the nominal value. In stochastic processing, only real-valued

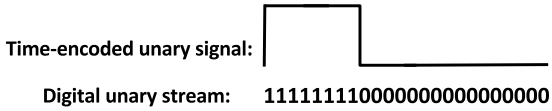



Fig. 3: Time-based vs. digital-stream unary representation.

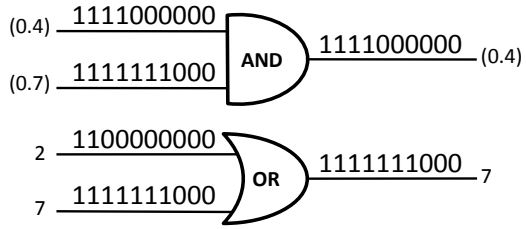



Fig. 4: Example of performing maximum and minimum operations on unary streams.

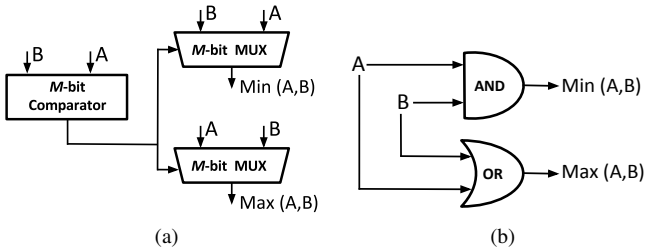



Fig. 5: Hardware implementation of a CAS block a) Conventional binary design b) Unary design.

numbers can be represented: numbers in the [0, 1] interval with the unipolar format and numbers in the [-1, 1] interval with the bipolar format. In contrast, with unary streams both real-valued and integer numbers can be represented. In representing real-value numbers, the number of ones divided by the length of stream determines the value. In representing integer values, the number of ones directly determines the value. For example, when using unary streams in the real domain, the streams 1000 and 11000000 are both representations of the value 0.25. In the integer domain, on the other hand, these streams represent 1 and 2, respectively. Similar to the bipolar format for stochastic streams, negative numbers can also be represented with unary streams using a simple linear transformation [4].

**Unary Operations.** The maximum (Max) and minimum (Min) value functions are two useful functions with simple and low cost unary implementation. In a weighted binary design, data-width dependent comparator and multiplexer units must be used to implement these functions. In unary processing, individual gates can synthesize these functions: an AND gate gives the minimum of two unary streams when two equal-length unary streams are connected to its inputs; an OR gate gives the maximum value when its inputs are fed with two equal-length unary streams. These gates showed a similar functionality when fed with correlated stochastic bit-streams [5]. Figure 4 shows an example of finding the minimum and maximum values in unary processing. An important advantage of unary processing is that synthesizing a function is independent of the resolution of data (length of streams). The same core logic is used for processing 128-bit unary streams that is used for processing 256-bit unary streams. While developing a general method for synthesizing all operations with unary processing is still a work in progress, recent work has shown absolute-value subtraction (using an XOR gate), comparison (using a D-type flip-flop) [31], and multiplication (using an AND gate) [23], [30] of unary streams.

**Time-based unary streams.** The representation of numbers in unary processing is not limited to purely digital bit streams. A time-based interpretation of numbers is also possible using pulse modulation of data [30]. Figure 3 shows both approaches. While both approaches can operate on the same unary logic, the time-based representation offers a seamless solution to the increasing number of time-based sensors and, as we will show, can be exploited in addressing the long latency problem of unary circuits.

## III. Complete Sort System

In this section we discuss hardware implementation of complete sort networks. We first discuss the conventional binary design of the complete sorting networks and then present the synthesis approach based on unary processing.

### A. Conventional Design

As discussed in Section II, sorting networks are made of CAS blocks. The hardware cost of a sorting network is therefore a direct function of the number of CAS blocks and the cost of each block. As shown in Figure 5a, in a weighted binary design with a data-width of $M$ bits, each CAS block consists of one $M$-bit comparator and two $M$-bit multiplexers. Thus, by increasing the resolution of data, the complexity of the design will also be increased. Increasing the complexity of the design directly affects the cost of the hardware implementation, latency, power, and as a result energy consumption. Another issue with the conventional binary design is noise immunity and fault tolerance. In a noisy environment, faults due to bit flips on high-order bits can produce large errors. Thus, additional fault-tolerance techniques must be used if the goal is to design a noise tolerant system.

### B. Unary Design

The essential operations in CAS blocks are maximum and minimum functions. This makes unary processing a good fit for hardware implementation of CAS blocks and sorting networks. As shown in Figure 5, instead of data-width dependent complex logic, one AND and one OR gate is sufficient to synthesize the CAS block in unary domain. The sorting networks can therefore be synthesized regardless of the resolution of the input data. While the synthesized circuit will be much less costly than the circuit synthesized in the binary approach, additional overhead must be incurred for conversion units which are required to convert the data between the binary and the unary fomart and a longer operation time due to performing the operation on $2^M$-bit long streams.

Assuming that the input data is given in binary format and the result must again be in binary, a unary stream generator is required to convert the data from binary to unary and a counter is required to count the number of ones in the final unary stream and convert the result back into binary. Figure 6 shows the design of a unary stream generator responsible for converting the data from binary to unary. For

TABLE I: Synthesis results of complete bitonic sort networks (Non-Pipelined).

| # of inputs and outputs | # of CAS units | Data width | Area ($\mu m^2$) | | Critical Path ($ns$) | | Power (@max f) — (@50MHz) ($mW$) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Conven. | Unary | Conven. | Unary | Conven. | Unary | Conven. | Unary |
| 8 | 24 | 8-bit | 3,086 | 2,194 | 1.85 | 0.74 | 1.30 | 3.26 | 0.12 | 0.13 |
| | | 16-bit | 6,865 | 4,531 | 2.05 | 0.75 | 2.63 | 5.59 | 0.27 | 0.23 |
| | | 32-bit | 14,868 | 9,456 | 2.41 | 0.77 | 4.90 | 10.1 | 0.62 | 0.44 |
| 16 | 80 | 8-bit | 10,534 | 4,511 | 2.73 | 0.87 | 3.66 | 5.30 | 0.49 | 0.25 |
| | | 16-bit | 22,920 | 8,901 | 3.42 | 0.89 | 6.61 | 8.94 | 1.17 | 0.44 |
| | | 32-bit | 49,812 | 17,274 | 3.80 | 0.93 | 13.4 | 15.9 | 2.63 | 0.83 |
| 32 | 240 | 8-bit | 32,508 | 9,235 | 4.06 | 1.07 | 8.86 | 8.40 | 1.75 | 0.49 |
| | | 16-bit | 68,621 | 17,643 | 5.05 | 1.13 | 16.6 | 13.8 | 4.18 | 0.86 |
| | | 32-bit | 149,669 | 27,811 | 5.90 | 1.12 | 31.8 | 25.4 | 11.3 | 1.52 |
| 64 | 672 | 8-bit | 90,691 | 19,028 | 5.71 | 1.33 | 19.8 | 13.4 | 5.48 | 0.96 |
| | | 16-bit | 191,174 | 29,259 | 7.03 | 1.35 | 39.2 | 22.5 | 13.6 | 1.60 |
| | | 32-bit | 431,182 | 56,598 | 8.00 | 1.37 | 78.5 | 41.2 | 33.1 | 3.03 |
| 128 | 1,792 | 8-bit | 242,049 | 33,916 | 7.49 | 1.62 | 44.4 | 21.4 | 15.7 | 1.80 |
| | | 16-bit | 523,565 | 60,686 | 9.27 | 1.63 | 89.8 | 37.1 | 41.1 | 3.19 |
| | | 32-bit | 1,047,646 | 115,835 | 10.14 | 1.63 | 165.7 | 69.1 | 85.4 | 6.05 |
| 256 | 4,608 | 8-bit | 586,456 | 74,719 | 9.71 | 1.91 | 88.7 | 36.5 | 42.2 | 3.64 |
| | | 16-bit | 1,239,154 | 126,804 | 11.79 | 1.94 | 181.3 | 62.1 | 102 | 6.40 |
| | | 32-bit | 2,560,803 | 234,957 | 12.89 | 1.97 | 367.7 | 113 | 221 | 12.0 |

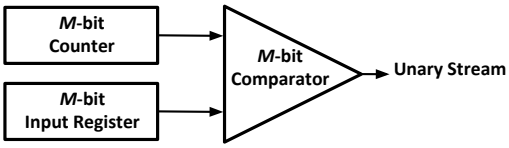



Fig. 6: Unary stream generator.

each input data, one unary stream generator and, for each output, one counter is required. A significant cost saving in implementing the CAS blocks, particularly for large-scale sorting circuits, will compensate for the overhead of converters in unary designs. Note that while the converters are data-width dependent, the CAS blocks synthesized with the unary approach are independent of data resolution.

### C. Design Evaluation

In order to evaluate the costs and benefits of the proposed design approach, we developed Verilog hardware descriptions of complete bitonic sorting networks for 8, 16, 32, 64, 128, and 256 data inputs, for both the conventional binary and for the proposed unary approach. For the unary approach, the architectures include the required conversion units from/to binary. The developed designs are synthesized using the Synopsys Design Compiler vH2013.12 and a 45nm standard-cell library. We report synthesis results for three different data widths of 8, 16, and 32 bits. In order to find the minimum hardware cost and also the maximum speed of the developed architectures we synthesized a non-pipelined and also a pipelined version of each architecture.

*1) Non-Pipelined Design:* Table I shows the synthesis results for the non-pipelined implementations. As can be seen, the unary approach could save the hardware cost of the implemented sort networks up to 91%. For small networks like the 8-input sort networks, the cost overhead of unary stream generators and output converters was comparable to the saving due to using a low-cost CAS implementation and so lower savings are achieved. By increasing the number of inputs and so the number of CAS blocks, the savings dominate the overheads and a hardware area saving of around 91% is achieved when implementing the 256-input sorting network with the unary approach.

The total (dynamic plus static) power consumption of the synthesized designs at the maximum feasible working frequency of each architecture, and also at a constant working frequency of 50 MHz, are presented in Table I. The static power or leakage is the dominant power when the system operates at low frequencies. It is directly proportional to the hardware cost and so a sort network with a lower hardware cost will have a lower leakage power. When a system works at its maximum frequency, dynamic power, which is an increasing function of the working frequency, is the dominant one. Thus, although the unary designs would have a much lower power consumption at low speeds, due to a lower critical path latency and so higher maximum working frequency, the power numbers reported for unary implementation of the 8 and 16 input sorting networks are greater than the power numbers reported for their corresponding binary implementations. As shown in Table I, for larger sorting networks (32-input and above), the simplicity of the unary design has led to even a lower power consumption at the maximum working frequency than the power consumption of the binary implementation.

Due to a simpler architecture, the critical path (CP) latency of the designs synthesized with the unary approach is lower than that of the conventional binary designs. However, the total latency of the unary approach which is the product of the CP latency and $2^M$ (the number of clock cycles the system must operates to generate and process the unary stream), is much more than the latency of the conventional design (one clock cycle $\times$ CP latency). Although the longer latency of the unary approach is still acceptable for many applications, a more important issue is the energy consumption. Energy consumption is evaluated by the product of the processing time and the total power consumption. Although the unary implementations of the sorting networks have often shown a

TABLE II: Synthesis results of complete bitonic sort networks (Pipelined).

| # of inputs and outputs | CAS units | Pipeline Stages | Data width | Area ($\mu m^2$) | | Critical Path ($ns$) | | Power (@max freq) | | (@50MHz) ($mW$) | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Conven. | Unary | Conven. | Unary | Conven. | Unary | Conven. | Unary |
| 8 | 24 | 6 | 8-bit | 6,926 | 2,659 | 0.42 | 0.39 | 19.5 | 8.1 | 0.46 | 0.16 |
| | | | 16-bit | 14,383 | 5,024 | 0.49 | 0.42 | 35.6 | 11.6 | 0.97 | 0.27 |
| | | | 32-bit | 25,066 | 9,916 | 0.53 | 0.49 | 66.5 | 17.2 | 1.88 | 0.48 |
| 16 | 80 | 10 | 8-bit | 19,338 | 5,834 | 0.42 | 0.40 | 67.9 | 17.0 | 1.50 | 0.37 |
| | | | 16-bit | 39,554 | 10,323 | 0.48 | 0.44 | 126 | 23.3 | 3.17 | 0.56 |
| | | | 32-bit | 83,102 | 18,065 | 0.52 | 0.50 | 241 | 33.9 | 6.64 | 0.94 |
| 32 | 240 | 15 | 8-bit | 57,900 | 13,095 | 0.42 | 0.41 | 213 | 38.5 | 4.68 | 0.86 |
| | | | 16-bit | 118,202 | 17,029 | 0.50 | 0.46 | 381 | 48.2 | 9.95 | 1.16 |
| | | | 32-bit | 248,129 | 29,682 | 0.53 | 0.50 | 748 | 70.0 | 21.0 | 1.88 |
| 64 | 672 | 21 | 8-bit | 161,934 | 25,248 | 0.42 | 0.44 | 602 | 83.0 | 13.3 | 1.92 |
| | | | 16-bit | 329,787 | 37,726 | 0.50 | 0.47 | 1105 | 104 | 28.7 | 2.61 |
| | | | 32-bit | 718,216 | 63,144 | 0.52 | 0.50 | 2201 | 149 | 61.9 | 4.02 |
| 128 | 1,792 | 28 | 8-bit | 431,062 | 59,579 | 0.42 | 0.47 | 1625 | 182 | 36.0 | 4.53 |
| | | | 16-bit | 901,206 | 84,646 | 0.49 | 0.50 | 3070 | 221 | 78.8 | 5.90 |
| | | | 32-bit | 1,834,850 | 134,746 | 0.52 | 0.52 | 5990 | 310 | 167 | 8.70 |
| 256 | 4,608 | 36 | 8-bit | 1,107,998 | 140,006 | 0.42 | 0.49 | 4228 | 407 | 93.0 | 10.6 |
| | | | 16-bit | 2,294,989 | 189,903 | 0.49 | 0.51 | 7859 | 489 | 204 | 13.3 |
| | | | 32-bit | 4,714,805 | 289,723 | 0.54 | 0.54 | 15024 | 648 | 437 | 18.9 |

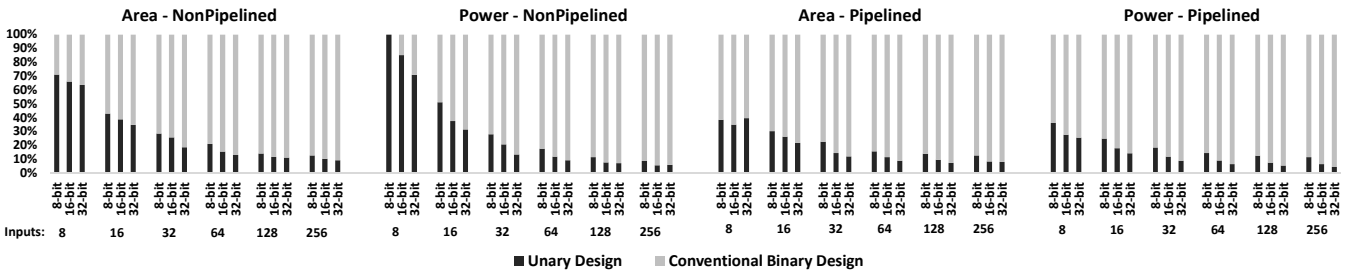



Fig. 7: Normalized area and power (@50MHz) cost numbers reported for the non-pipelined and pipelined structures of the implemented complete sort networks.

lower power consumption for a fixed frequency, a very long processing time would lead to a higher energy consumption than their conventional binary counterparts. We will address the long latency and high energy consumption problem of unary designs in the next section.

*2) Pipelined Design:* Table II shows the synthesis results for a fully pipelined structure (only one CAS block between pipeline registers) of the developed designs. Although due to using a large number of pipeline registers, the fully pipelined structure is significantly more costly than the non-pipelined structure, a higher working frequency is achieved with the pipelined one. Designing the sorting network with only one CAS block between pipeline registers leads to a higher latency and total area than the case with more number of CAS blocks between pipeline registers. However, the one CAS block approach (fully pipelined) results in a higher sorting throughput [17]. Thus, choosing the number of CAS blocks between pipeline registers is a trade-off between the total area and latency, and the throughput, and is a design decision.

As can be seen in Table II, the hardware area cost of the pipelined unary designs are 61% to 92% lower than the hardware cost of the pipelined binary designs. Observing a high saving in the area of the small-scale sorting circuits, such as the 8-input sorting network (61% for 8-bit data), is due to using simpler pipeline registers (1-bit instead of $M$-bit) in the pipelined unary design compared to the pipelined binary design. Figure 7 shows normalized diagrams for area and power cost numbers of the synthesized architectures. In each configuration, the results are normalized to the value of the conventional design with that configuration.

Critical path latency of the unary design in the pipelined structure of small sorting networks was slightly lower than that of the binary designs. The reason was a simpler CAS block between the pipeline registers in the unary approach. For large networks (e.g. 128-input, 256-input), however, the CP latency of binary design was lower than the unary implementation. Although in these designs still the CAS blocks of the unary approach are simpler, a more complex unary stream generator and a larger output counter limit the performance of the circuit and increase the CP. The total processing time of the pipelined binary design is the product of the CP latency and the number of pipeline stages. The throughput, however, is higher than the non-pipelined binary design because at each cycle a new set of inputs can enter the system and a set of sorted numbers is leaving the system. For pipelined unary designs, the total latency is the CP latency $\times$ number of pipeline stages $\times 2^M$, where $M$ is the data-width. Thus, similar to the non-pipelined structure, the total latency of the pipelined unary implementations is much higher than the total latency of their conventional binary counterparts. This long latency further

makes the total energy consumption higher than the energy consumption of the binary designs. We will address this issue in the next section by time-encoding of data using a mixed-signal design of sorting network-based median filtering.

## IV. Highly Efficient Median Filters

A median filter is a popular non-linear filter widely used in image, speech, and signal processing applications. It replaces each input data with the median of all the data in a local neighborhood. This results in filtering out impulse noise and smoothing of the image while preserving important properties such as the edge information [35]. In real-time image and video applications, the digital image data are affected by noise resulting from image sensors or transmission of images. A hardware implementation of the median filter is therefore required for denoising. The high computational complexity of median filters, however, makes their hardware implementation expensive and inefficient for many applications. In this section we first propose a low-cost implementation of median filters similar to the unary sorting networks introduced in Section III. We then exploit a time-based representation of input data using pulse-width modulation to address the long latency problem of the implemented circuits.

### A. Circuit Design

There are a variety of methods for hardware implementation of median filters [44], [25]. Sorting network-based architectures [10] consisting of a network of CAS blocks are one of the most common approaches. The incoming data is sorted as it passes the network. The middle element of the sorted data is the median. As the sorting network can be easily pipelined, the approach provides the best performance [35]. The local neighborhood in median filtering is often a 3x3 or 5x5 window with the target input data at the center. Figures 8 and 9 show the sorting networks for a 3x3 and a 5x5 median filter, respectively. We developed a non-pipelined and a pipelined structure of these median filters with both the conventional binary and the proposed unary design approach with 8-bit input data resolution. The CAS blocks presented in Figure 5 were used in the developed architectures. A separate unary stream generator was used for converting each input data and a counter was used for converting the output median stream back to binary form in the unary designs.

Table III shows the synthesis results for the developed architectures. For now let us ignore the rows representing Unary-Time-based designs, they will be discussed in Section IV-B2. The overhead in pipelined designs includes pipeline registers and for unary designs include the required converters from/to binary. Similar to the results reported for the complete sort networks, the unary implementation of the median filters significantly improves the hardware cost, up to 90% for the 5x5 median filter architecture. The pipelined implementations have a higher working frequency and a higher throughput. Comparing the power consumption of the pipelined implementations show that, for the same working frequency, the unary designs have a significantly lower power consumption. For applications in which hardware cost and power consumption

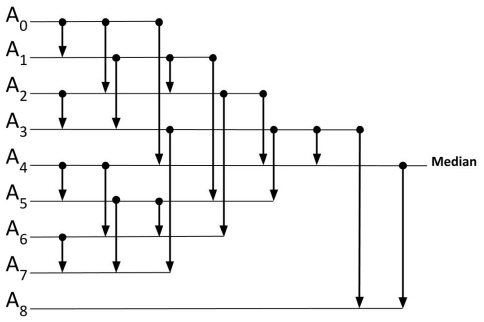



Fig. 8: The CAS network for a 3x3 Median Filter made of 19 CAS blocks [28].

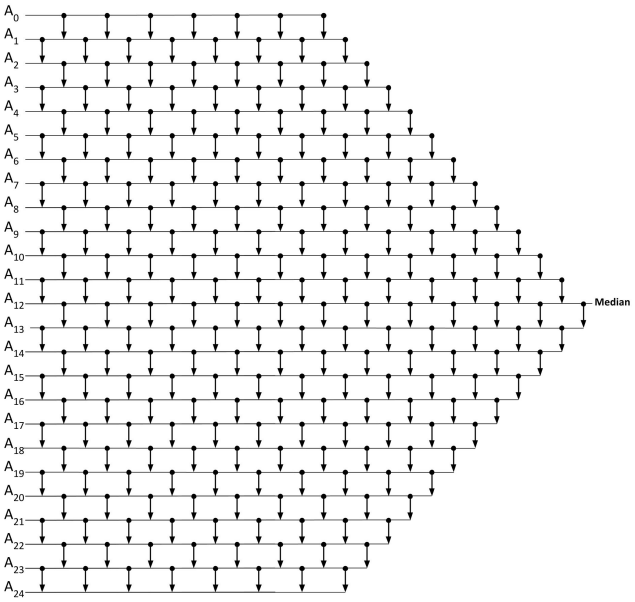



Fig. 9: The CAS network for a 5x5 Median Filter made of 246 CAS blocks [46].

are the main priorities, the proposed unary designs outperform the conventional weighted binary designs. However, for high-performance low-energy applications the binary design can be a better choice. In the following section we exploit the concept of near sensor processing and time-based representation of data to improve the latency and energy consumption of the unary-based median filtering designs at the cost of a slight accuracy loss.

### B. Time-based unary design

*1) Overview:* Image sensors convert the light intensity to an analog voltage/current. The conventional approach for processing these sensed data is to first convert the analog data to digital binary form using a conventional analog-to-digital (ADC) and then process the binary data using digital logic. In unary processing, this binary data is first converted to a unary bit stream and then processed using unary circuits. Processing of image pixels with 8-bit resolution requires running the unary circuit for 256 cycles. Even with a higher working frequency, due to a large number of clock cycles running the

TABLE III: Synthesis results of the sorting network-based median filters for data-width=8.

| Median Filter | Design Approach | Area ($\mu m^2$) | | | Latency ($ns$) | | Power ($mW$) (@max freq) | Energy ($pJ$) |
|---|---|---|---|---|---|---|---|---|
| | | CAS Logic | Overhead | Total | CP | Total | | |
| 3x3 | Binary-NonPipelined | 2,167 | - | 2,167 | 2.10 | 2.10 | 1.03 | 2.1 |
| | Binary-Pipelined (8-stage) | 2,167 | 3,384 | 5,551 | **0.43** | 3.44 | 15.56 | 6.6 |
| | Unary-NonPipelined | 79 | 917 | **996** | 0.70 | 179.2 | 0.95 | 170.2 |
| | Unary-Pipelined (8-stage) | 79 | 1,292 | 1,371 | 0.40 | 102.4 | 3.08 | 315.3 |
| | Unary-Time-based | 79 | 776 | **855** | 0.39 | **0.39** | 1.78 | **0.69** |
| 5x5 | Binary-NonPipelined | 32,772 | - | 32,772 | 6.77 | 6.77 | 5.76 | 38.9 |
| | Binary-Pipelined (26-stage) | 32,772 | 28,208 | 60,980 | **0.43** | 11.18 | 219 | 94.1 |
| | Unary-NonPipelined | 1,051 | 1,988 | **3,039** | 1.07 | 273.9 | 0.93 | 254.7 |
| | Unary-Pipelined (26-stage) | 1,051 | 6,377 | 7,428 | 0.40 | 102.4 | 19.68 | 2015.2 |
| | Unary-Time-based | 1,051 | 1,960 | **3,011** | 0.78 | **0.78** | 2.71 | **2.11** |

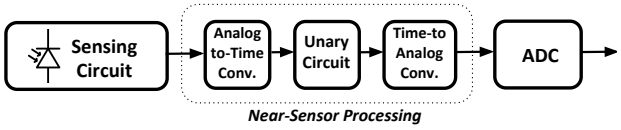



Fig. 10: Near Sensor Processing with unary circuits.

circuit, the total latency of the processing using unary circuit is more than that of processing with the binary design.

Near sensor image processing (NSIP) [16] is an interesting concept that suggests integrating some of the processing circuits (i.e. median filter circuit) with the sensing circuit. This can potentially improve the power consumption, size, and costs of vision chips. With more and more sensors providing time-encoded outputs and ways to convert signals from voltage or current to time signals [43], the sensed data in the form of time-encoded signals can directly be fed to unary circuits. Inspired from the NSIP concept and based on the idea of time-encoding data introduced in [30], we time-encode the sensed input data to address the long latency of processing using unary circuits. Figure 10 depicts a simple flow of the method. Assuming that the output of the sensing circuit is in voltage or current form, an analog-to-time converter (ATC) (i.e. low-cost circuit shown in Figure 11) is used to convert the sensed data to a time-encoded pulse signal. The converted signal is processed using the unary circuit and the output is converted back to a desired analog format using a time-to-analog converter (TAC) (i.e. a voltage integrator).

*2) Evaluation:* Table III shows the area, latency, power, and energy consumption of the implemented median filtering circuits synthesized with the conventional binary, digital bit-stream based unary, and the proposed time-based unary approach. The low-cost pulse-width modulator shown in Figure 11 was used as the ATC and a Gm-C active integrator [45] was used as the TAC to convert the output signal back to analog form in the time-based unary designs. While a pulse-width modulator generates a periodic signal with a specific duty cycle and frequency, only one period of the generated signal will be sufficient for processing the data using the unary designs [31]. The duty cycle of the generated signal is determined by the DC level of the sensed data. The hardware cost and the energy consumption of the implemented ATC and TAC are a function of the target working frequency. We extracted the area and energy numbers from [30] and

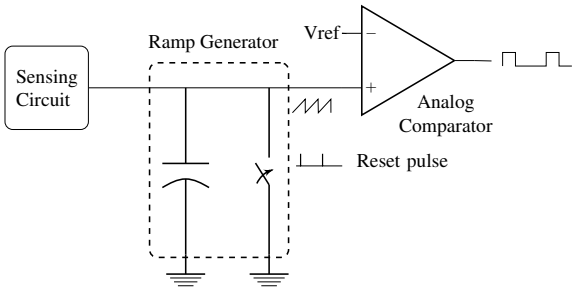



Fig. 11: A low-cost Analog-to-Time converter proposed in [30]. The Reset pulse defines the frequency of the output signal and is generated using the clock signal.

report them as the overhead of the time-based unary design in Table III.

A separate ATC is used for time-encoding each input data (9 ATCs for 3x3 median filter circuit). For each time-based unary design, the reported overhead numbers are for a working frequency equal to the inverse of the critical path latency of the circuit. Assuming that the clock signal that drives the ATC is available in the system, a lower working frequency translate to a lower area and energy overhead. As can be seen in Table III, the total area of the time-based designs including the overhead of ATCs and TAC is lower than the area cost of the digital bit stream-based non-pipelined version of the unary design. The total latency and the energy consumption of the time-based unary designs are better than those of the pipelined and non-pipelined structure of the unary design and also lower than those of the binary designs. A lower CP latency in the time-based unary designs in comparison to the non-pipelined unary design is due to not using unary stream generator and counter in the time-based approach.

The down-side of the time-based unary design, however, is a slight accuracy loss. The working frequency of the ATC affects the effective number of bits in representing and processing data, hence the accuracy of computation. To evaluate the performance of the median filtering unary designs when working with time-encoded input signals, we developed SPICE netlists of both 3x3 and 5x5 median filtering circuits and simulated their operation on a $128 \times 128$ noisy soldier image. The sample input image is shown in Figure 12. Simulations were carried

TABLE IV: Average error rate of processing the sample image using the time-based unary circuits.

| Median Filter Time-based Unary | | Length of input signals (1/freq.) | | | |
|---|---|---|---|---|---|
| | | CP | 1ns | 2ns | 5ns |
| 3x3 | Ideal ATC | 2.09% | 0.84% | 0.45% | 0.19% |
| | ATC of [30] | 2.65% | 1.05% | 0.56% | 0.21% |
| 5x5 | Ideal ATC | 4.70% | 3.33% | 1.83% | 0.94% |
| | ATC of [30] | 4.86% | 3.66% | 1.90% | 1.01% |

out using a 45-nm standard cell library in HSPICE. Table IV shows the average output error rates for the images produced using the time-based unary designs. Image pixel intensities were converted to pulse signals using the ATC shown in Figure 11 and also using the HSPICE built-in pulse generator (an ideal ATC). In Table IV, these two methods correspond to the rows "ATC of [30]" and "Ideal ATC", respectively. Comparing the output images with the expected output image (produced using a software-based implementation of the algorithm in Matlab), the mean of the output error rates was calculated as follows:

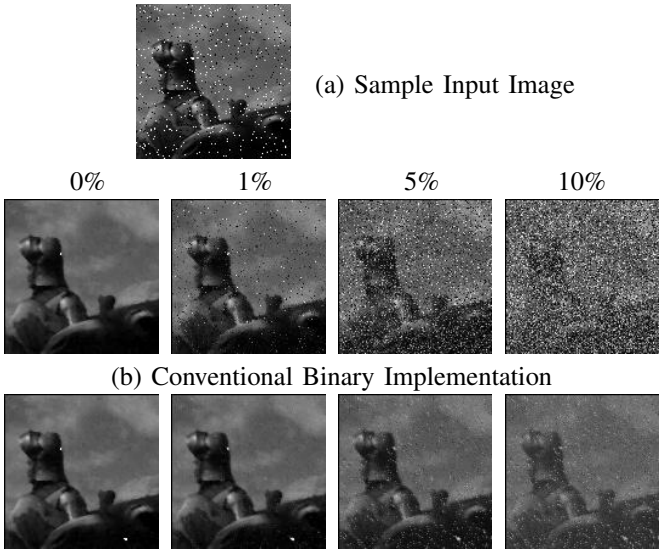
$$AverageErrorRate = \frac{\sum_{i=1}^{W}\sum_{j=1}^{H}|P_{i,j} - E_{i,j}|}{255.(W \times H)} \times 100$$

where $E_{i,j}$ is the expected value for location $(i, j)$ in the output image, $P_{i,j}$ is the pixel value for the same location produced using the circuit, and $W$ and $H$ are the dimensions of the image. As can be seen in Table IV, increasing the length of the input signal (a lower working frequency) leads to a higher accuracy in the time-based approach. An average error rate of less than 1 percent is achieved in the 3x3 median filtering circuit with 1ns and in the 5x5 circuit with 5ns processing time. The inherent inaccuracy in converting the values with the ATC of [30] resulted in a slightly higher error rates when comparing to the error rates where using ideal ATC.

*3) Sources of inaccuracy:* Error in generating pulse signals (analog value to time conversion), error in measuring the output signal (time to analog conversion), and error due to skew noise [30] are the main sources of errors in the time-based unary processing. A different gate delay for AND and OR gates, particularly, can be a main source of skew in the unary sorting networks. Such a skew is negligible for small sorting networks (e.g. 3x3 median filtering). However, for large sorting networks (e.g. 5x5 median filtering) the skew in each stage is propagated to the next stage, resulting a considerable skew error. With careful gate sizing and adjusting gate delays, or simply increasing the length of the input signals we can mitigate this source of inaccuracy in the time-based unary design.

## V. Noise-Tolerant Behavior

To evaluate the noise-tolerance of the proposed unary designs in comparison to that of the corresponding conventional binary implementations, we randomly injected soft errors, i.e., bit flips, for 0%, 1%, 5%, and 10% noise injection rates on the inputs of CAS blocks of the 3x3 median filtering circuits and measured the corresponding average output error rates. A noise injection rate of 10% means that 10% of the total bits in the inputs of CAS blocks are randomly chosen and flipped. The sample image shown in Figure 12 was used as the input to the circuits. For the conventional binary implementation the data-width was fixed at 8 bits and bit streams of length 256 were used to represent values in the unary designs. Figure 12 shows the performance of the implemented circuits at various noise injection rates. As can be seen, the proposed unary implementation has shown a higher noise-tolerance compared to the conventional binary implementation. For injection rates higher than 1%, the quality of the output image produced by the binary design degrades drastically leading to a useless image for injection rates higher than 5%. This noise-immunity observed in the unary design is mainly due to its data encoding approach, a common property between the unary and the stochastic processing. Bits are equally weighted in unary streams and so bit flips produce small and uniform deviation from the nominal value.




(a) Sample Input Image

(b) Conventional Binary Implementation

(c) Proposed Unary Implementation

Fig. 12: (a) Sample input image, and comparison of the noise-tolerance capability of (b) the conventional binary vs. (c) the proposed unary implementation for the 3x3 median filtering circuit for different noise injection rates.

## VI. Conclusions and Future Work

Batcher sorting networks have been widely used in different applications. Their regular structure makes them popular for signal processing systems and communication switching networks. However, a conventional weighted binary-based implementation of a large sorting networks is costly considering the large number of compare-and-swap (CAS) units that such a network entails. The VLSI cost increases significantly with increasing resolution of the input data. The high hardware cost and the high power consumption of such networks restrict their application.

This work proposes an area and power efficient implementation of sorting networks based on unary processing. The core processing logic consists of simple gates and is independent

of the resolution of data. The only overhead in the approach, the cost of converting data from/to binary, is small. More than 90% area and power savings are observed when compared to the costs of a conventional weighted binary implementation.

The penalty is latency. Processing digital unary streams, requires a relatively long running time, e.g., more than 100ns to process each set of input data. Although this is a $100\times$ increase in latency over conventional weighted binary, this increase may be tolerable for many applications. For example, ten gray-scale HD ($1280 \times 720$) images or four gray-scale Full HD ($1920 \times 1080$) images can be processed per second with the proposed scheme for a task such as median filtering, when operating on 256-bit long unary streams. In spite of the latency, a 90% decrease in power consumption might often make this a winning proposition.

To mitigate the latency of the approach, we further developed a time-based unary design approach in which the input data is encoded in time and represented with pulse signals. The result is a significant improvement in the latency and energy consumption, at the cost of a slight loss in accuracy. For example, more than 1000 gray-scale HD images or 400 gray-scale Full HD images can be processed per second with the proposed time-based unary implementation of the 3x3 median filtering at the cost of only 1% loss in accuracy.

In future work we will explore other applications of sorting based on unary processing, for instance in hardware implementations of weighted and adaptive median filters. We will also explore applications in communications and coding.

## ACKNOWLEDGMENT

This work was supported in part by National Science Foundation grant no. CCF-1408123. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. A perliminary version of this paper appeared as [32].

## REFERENCES

[1] J. P. Agrawal. Arbitrary size bitonic (asb) sorters and their applications in broadband atm switching. In *Conference Proceedings of the 1996 IEEE Fifteenth Annual International Phoenix Conference on Computers and Communications*, pages 454–458, Mar 1996.

[2] S. W. Al-Haj Baddar and B. A. Mahafzah. Bitonic sort on a chained-cubic tree interconnection network. *J. Parallel Distrib. Comput.*, 74(1):1744–1761, Jan. 2014.

[3] A. Alaghi, W.-T. J. Chan, J. P. Hayes, A. B. Kahng, and J. Li. Trading accuracy for energy in stochastic circuit design. *J. Emerg. Technol. Comput. Syst.*, 13(3):47:1–47:30, Apr. 2017.

[4] A. Alaghi and J. P. Hayes. Survey of stochastic computing. *ACM Trans. Embed. Comput. Syst.*, 12(2s):92:1–92:19, 2013.

[5] A. Alaghi, C. Li, and J. Hayes. Stochastic circuits for real-time image-processing applications. In *DAC, 2013 50th ACM/EDAC/IEEE*, pages 1–6, May 2013.

[6] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 307–314, New York, NY, USA, 1968. ACM.

[7] V. Brajovic and T. Kanade. A vlsi sorting image sensor: global massively parallel intensity-to-time processing for low-latency adaptive vision. *IEEE Transactions on Robotics and Automation*, 15(1):67–75, Feb 1999.

[8] B. Brown and H. Card. Stochastic neural computation. i. computational elements. *Computers, IEEE Transactions on*, 50(9):891–905, Sep 2001.

[9] B. D. Brown and H. C. Card. Stochastic neural computation. II. Soft competitive learning. *IEEE Transactions on Computers*, 50(9):906–920, Sep 2001.

[10] C. Chakrabarti. Sorting network based architectures for median filters. *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, 40(11):723–727, 1993.

[11] C. Chakrabarti and L.-Y. Wang. Novel sorting network-based architectures for rank order filters. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):502–507, Dec 1994.

[12] R. Chen and V. K. Prasanna. Computer generation of high throughput and memory efficient sorting designs on fpga. *IEEE Transactions on Parallel and Distributed Systems*, 28(11):3100–3113, Nov 2017.

[13] A. Colavita, E. Mumolo, and G. Capello. A novel sorting algorithm and its application to a gamma-ray telescope asynchronous data acquisition system. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 394(3):374 – 380, 1997.

[14] A. A. Colavita, A. Cicuttin, F. Fratnik, and G. Capello. Sortchip: a vlsi implementation of a hardware algorithm for continuous data sorting. *IEEE Journal of Solid-State Circuits*, 38(6):1076–1079, June 2003.

[15] K. Cushon, C. Leroux, S. Hemati, S. Mannor, and W. J. Gross. A min-sum iterative decoder based on pulsewidth message encoding. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 57(11):893–897, Nov 2010.

[16] J. E. Eklund, C. Svensson, and A. Astrom. Vlsi implementation of a focal plane image processor-a realization of the near-sensor image processing concept. *IEEE TVLSI*, 4(3):322–335, Sept 1996.

[17] A. Farmahini-Farahani, H. J. D. III, M. J. Schulte, and K. Compton. Modular design of high-throughput, low-latency sorting units. *IEEE Transactions on Computers*, 62(7):1389–1402, July 2013.

[18] B. Gaines. Stochastic computing systems. In *Advances in Information Systems Science*, Advances in Information Systems Science, pages 37–172. Springer US, 1969.

[19] B. Gedik, R. R. Bordawekar, and P. S. Yu. Cellsort: High performance sorting on the cell processor. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 1286–1297, 2007.

[20] N. Govindaraju, J. Gray, R. Kumar, and D. Manocha. Gputerasort: High performance graphics co-processor sorting for large database management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 325–336, New York, NY, USA, 2006. ACM.

[21] G. Graefe. Implementing sorting in database systems. *ACM Comput. Surv.*, 38(3), Sept. 2006.

[22] N. Guo, Y. Huang, T. Mai, S. Patil, C. Cao, M. Seok, S. Sethumadhavan, and Y. Tsividis. Energy-efficient hybrid analog/digital approximate computation in continuous time. *IEEE Journal of Solid-State Circuits*, 51(7):1514–1524, July 2016.

[23] D. Jenson and M. Riedel. A deterministic approach to stochastic computation. In *Proceedings of the 35th International Conference on Computer-Aided Design*, ICCAD '16, pages 102:1–102:8, New York, NY, USA, 2016.

[24] K. Kantawala and D. L. Tao. Design, analysis, and evaluation of concurrent checking sorting networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 5(3):338–343, Sept 1997.

[25] M. Karaman, L. Onural, and A. Atalar. Design and implementation of a general-purpose median filter unit in cmos vlsi. *IEEE Journal of Solid-State Circuits*, 25(2):505–513, Apr 1990.

[26] S. Y. Kuo and S. C. Liang. Design and analysis of defect tolerant hierarchical sorting networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1(2):219–223, June 1993.

[27] B. Li, M. H. Najafi, and D. J. Lilja. Using Stochastic Computing to Reduce the Hardware Requirements for a Restricted Boltzmann Machine Classifier. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pages 36–41, New York, NY, USA, 2016. ACM.

[28] P. Li, D. Lilja, W. Qian, K. Bazargan, and M. Riedel. Computation on stochastic bit streams digital image processing case studies. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 22(3):449–462, 2014.

[29] B. A. Mahafzah. Performance assessment of multithreaded quicksort algorithm on simultaneous multithreaded architecture. *The Journal of Supercomputing*, 66(1):339–363, Oct 2013.

[30] M. H. Najafi, S. Jamali-Zavareh, D. J. Lilja, M. D. Riedel, K. Bazargan, and R. Harjani. Time-Encoded Values for Highly Efficient Stochastic Circuits. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 25(5):1–14, 2017.

[31] M. H. Najafi and D. J. Lilja. High-Speed Stochastic Circuits using Synchronous Analog Pulses. In *2017 22nd ASP-DAC*, pages 481–487, Jan 2017.

[32] M. H. Najafi, D. J. Lilja, M. Riedel, and K. Bazargan. Power and Area Efficient Sorting Networks using Unary Processing. In *Computer Design (ICCD), 2017 IEEE 35th International Conference on*, Nov 2017.
[33] M. H. Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan. Polysynchronous Clocking: Exploiting the Skew Tolerance of Stochastic Circuits. *IEEE Transactions on Computers*, 66(10):1734–1746, Oct 2017.
[34] M. H. Najafi and M. E. Salehi. A Fast Fault-Tolerant Architecture for Sauvola Local Image Thresholding Algorithm Using Stochastic Computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(2):808–812, Feb 2016.
[35] E. Nikahd, P. Behnam, and R. Sameni. High-speed hardware implementation of fixed and runtime variable window length 1-d median filters. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 63(5):478–482, May 2016.
[36] D. S. K. Pok, C. I. H. Chen, J. J. Schamus, C. T. Montgomery, and J. B. Y. Tsui. Chip design for monobit receiver. *IEEE Transactions on Microwave Theory and Techniques*, 45(12):2283–2295, Dec 1997.
[37] W. Poppelbaum. Burst processing: A deterministic counterpart to stochastic computing. In *Proceedings of the 1st International Symposium on Stochastic Computing and its Applications*. 1978.
[38] W. Poppelbaum, A. Dollas, J. Glickman, and C. O'Toole. Unary processing. In *Advances in Computers*, volume 26, pages 47 – 92. Elsevier, 1987.
[39] W. J. Poppelbaum, C. Afuso, and J. W. Esch. Stochastic computing elements and systems. In *Proceedings of the Joint Computer Conference*, AFIPS '67 (Fall), pages 635–644, New York, NY, USA, 1967. ACM.
[40] W. Qian, X. Li, M. Riedel, K. Bazargan, and D. Lilja. An architecture for fault-tolerant computation with stochastic logic. *Computers, IEEE Trans. on*, 60(1):93–105, Jan 2011.
[41] W. Qian and M. Riedel. The synthesis of robust polynomial arithmetic with stochastic logic. In *45th ACM/IEEE Design Automation Conference, DAC'08*, pages 648–653, 2008.
[42] K. Ratnayake and A. Amer. An fpga architecture of stable-sorting on a large data volume : Application to video signals. In *2007 41st Annual Conference on Information Sciences and Systems*, pages 431–436, March 2007.
[43] V. Ravinuthula, V. Garg, J. G. Harris, and J. A. B. Fortes. Time-mode circuits for analog computation. *Intern. Journal of Circuit Theory and Applications*, 37(5):631–659, 2009.
[44] D. S. Richards. VLSI median filters. *IEEE Trans. on Acoustics, Speech, and Signal*, 38(1):145–153, Jan 1990.
[45] W. Sansen. Analog design essentials, ser. the international series in engineering and computer science, 2006.
[46] J. Scott. Analysis of two-dimensional median filter hardware implementations for real-time video denoising. MS thesis, Penn State University, December 2010.
[47] D. C. Stephens, J. C. R. Bennett, and H. Zhang. Implementing scheduling algorithms in high-speed networks. *IEEE Journal on Selected Areas in Communications*, 17(6):1145–1158, Jun 1999.
[48] Y. Tsividis. Continuous-time digital signal processing. *Electronics Letters*, 39(21):1551–1552, Oct 2003.

**M. Hassan Najafi** received the B.Sc. degree in computer engineering from University of Isfahan, Isfahan, Iran, and the M.Sc. degree in computer architecture from University of Tehran, Tehran, Iran, in 2011 and 2014, respectively. He is currently working toward the Ph.D. degree as a research assistant at ARCTiC Labs in the Department of Electrical and Computer Engineering, University of Minnesota, Twin cities. His research interests include stochastic and approximate computing, computer-aided design of integrated circuits, low power design, and designing fault tolerant systems. In recognition of his research, he received the Doctoral Dissertation Fellowship at the University of Minnesota and the Best Paper Award at the 2017 35th IEEE International Conference on Computer Design.

**David J. Lilja** (F06) received the B.S. degree in computer engineering from Iowa State University in Ames, IA, USA, and the M.S. and Ph.D. degrees in electrical engineering from the University of Illinois at Urbana-Champaign in Urbana, IL, USA. He is currently the Schnell Professor of Electrical and Computer Engineering at the University of Minnesota in Minneapolis, MN, USA, where he also serves as a member of the graduate faculties in Computer Science, Scientific Computation, and Data Science. Previously, he served ten years as the head of the ECE department at the University of Minnesota, and worked as a research assistant at the Center for Supercomputing Research and Development at the University of Illinois, and as a development engineer at Tandem Computers Incorporated in Cupertino, California. He was elected a Fellow of the Institute of Electrical and Electronics Engineers (IEEE) and a Fellow of the American Association for the Advancement of Science (AAAS). His main research interests include computer architecture, parallel processing, computer systems performance analysis, approximate computing, and storage systems.

**Marc D. Riedel** (SM12) received the B.Eng. degree in electrical engineering from McGill University, Montreal, QC, Canada, and the M.Sc. and Ph.D. degrees in electrical engineering from the California Institute of Technology (Caltech), Pasadena, CA, USA. He is currently an Associate Professor of electrical and computer engineering with the University of Minnesota, Minneapolis, MN, USA, where he is a member of the Graduate Faculty of biomedical informatics and computational biology. From 2004 to 2005, he was a Lecturer of computation and neural systems with Caltech. He was with Marconi Canada, CAE Electronics, Toshiba, and Fujitsu Research Labs. Dr. Riedel was a recipient of the Charl H. Wilts Prize for the Best Doctoral Research in Electrical Engineering at Caltech, the Best Paper Award at the Design Automation Conference, and the U.S. National Science Foundation CAREER Award.

**Kia Bazargan** (SM07) received the B.Sc. degree in computer science from Sharif University, Tehran, Iran, and the M.S. and Ph.D. degrees in electrical and computer engineering from Northwestern University, Evanston, IL, USA, in 1998 and 2000, respectively. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, USA.

Dr. Bazargan was a recipient of the US National Science Foundation Career Award in 2004. He was a Guest Co-Editor of the ACM Transactions on Embedded Computing Systems Special Issue on Dynamically Adaptable Embedded Systems in 2003. He was on the technical program committee of a number of the IEEE/ACM-sponsored conferences, including Field Programmable Gate Array, Field Programmable Logic, Design Automation Conference (DAC), International Conference on Computer-Aided Design, and Asia and South Pacific DAC. He was an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS from 2005 to 2012. He is a Senior Member of the IEEE Computer Society.