Fuzzy-logic using Unary Bit-Stream Processing

Amir Hossein Jalilvand*, M. Hassan Najafi† and Mahdi Fazeli‡
*Computer Engineering Department, Iran University of Science and Technology, Tehran, Iran
†School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA, USA
‡Department of Computer Engineering, Bogazici University, Istanbul, Turkey
jalilvand_a@comp.iust.ac.ir, najafi@louisiana.edu, m_fazeli@boun.edu.tr

***Abstract*—There is a growing attention to the theory of fuzzy-logic and its applications. Efficient hardware design of the fuzzy-inference engine has become necessary for high-performance applications. Considering the facts that fuzzy-logic variables have truth values in the $[0, 1]$ interval and fuzzy controllers include minimum and maximum operations, this work proposes to apply the concept of unary processing to the platform of fuzzy-logic. In unary processing, data in the $[0, 1]$ interval is encoded as bit-stream with the value defined by the frequency of 1s. Operations such as minimum and maximum functions can be implemented using simple logic gates. Latency, however, has been an important issue in the unary designs. To mitigate the latency, the proposed design processes right-aligned bit-streams. A one-hot decoder is used for fast detection of the bit-stream with maximum value. Implementing a fuzzy-inference engine with 81 fuzzy-inference rules, the proposed architecture provides 82%, 46%, and 67% saving in the hardware area, power and energy consumption, respectively, and 94% reduction in the number of used LUTs compared to conventional binary implementation.**

***Index Terms*—Unary processing, Fuzzy-logic, bit-stream computing, low-cost design.**

I. INTRODUCTION

Today, hardware-based processing is bounded by some strict design constraints such as low power consumption, small circuit area, and reliability. Power and area costs, in particular, are the main concerns in designing embedded systems. Weighted binary radix has been the dominate format for representation of data in these systems. Computation on this representation is rather complex and hence costly as each bit has its own weight according to its position. Considering the complexity of conventional binary designs, unconventional design techniques are receiving more and more attention. *Unary computing* [12], [13] is one of such unconventional techniques that offers low-cost design [9], [11]. The paradigm has some common characteristics to *stochastic computing* (SC) [1], [5], [14]; however, it is deterministic and produces completely accurate results. The operations are mainly on the numbers in the $[0, 1]$ interval. Unlike the weighted binary format, all digits are weighted equally in this paradigm. Numbers are encoded uniformly by a sequence of 1s followed by a sequence of 0s (or vise versa). The streams in this format are called "*unary bit-streams*" [4], [11]. Independent of the length, the value of a unary bit-stream is determined by the frequency of 1's. For example, 1100 and 111000 are two unary bit-streams representing 0.5.

Simplicity of hardware design is the main advantage of unary computing. A standard `AND` gate fed with uncorrelated (independent) unary bit-streams implements the multiplication operation [8]–[10]. Low-cost design of minimum (Min) and maximum (Max) value functions based on correlated unary bit-streams are discussed in [11] for low-cost design of sorting network circuits. Nonetheless, some potential applications of unary computing are not well investigated yet. In this work, as the first study of its kind to the best of our knowledge, we apply the concept of unary processing to the platform of fuzzy-logic design.

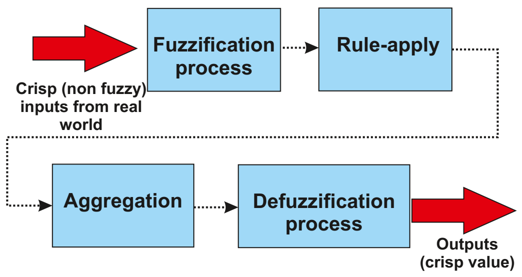

Fig. 1. The inference mechanism in fuzzy-logic applications.

The theory of fuzzy-logic [16] has been investigated in numerous applications including controlling systems, real-time embedded systems, robotics, security, image and signal processing, telecommunications, decision-making support systems, and chemical industry [2]. As illustrated in Fig. 1, most fuzzy-logic systems use the mechanism of inference which employs the following steps:

- Fuzzification process: converting crisp (non-fuzzy) inputs to fuzzy-linguistic values.
- Rule-apply: applying inference rules.
- Aggregation: aggregating the results of the rule-apply process.
- Defuzzification process: converting the fuzzy-linguistic value to crisp value of outputs.

In contrast to the two-valued logic in the binary sets (true or false), fuzzy-logic variables have truth values in the $[0, 1]$ interval, the acceptable range of data in unary processing. Previously, SC has been used in designing digital fuzzy-logic controller [3]. Lack of generality in the type of membership functions, inaccuracy of computations, and long processing time due to processing random bit-streams are the main limitations of the design in [3]. Song et al. [15] further develop a SC-based model to improve the evaluation efficiency of a fuzzy system. A fuzzy probability analysis is transformed into simple bit-wise operations on stochastic bit-streams. While their model accomplishes a faster convergence and requires a shorter runtime compared to the Monte Carlo simulation, still long random bit-streams of 100 to 1000 bits must be processed. In this work, we exploit the concept of unary processing in hardware-efficient design of fuzzy-logic systems. Our synthesis results show a significant improvement in the hardware area, power, and energy consumption compared to the conventional binary

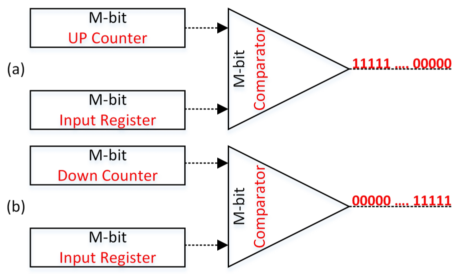

Fig. 2. Unary bit-stream generator: (a) left-aligned bit-stream, (b) right-aligned bit-stream.

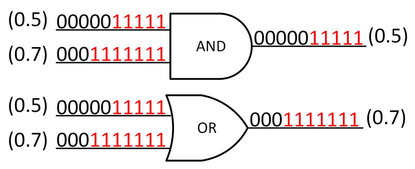

Fig. 3. Examples of Min (using `AND` gate) and Max (using `OR` gate) operations in unary computing.

implementation at no accuracy loss. The rest of this paper is organized as follows. Section II provides the necessary background on unary processing and fuzzy-logic systems. Section III discusses the proposed method and evaluates the proposed design. Section IV concludes the paper.

II. BACKGROUND

A. Unary processing

Processing data in the unary domain requires to convert the data from conventional binary to unary bit-stream representation. A stream of 2^M bits is required to demonstrate a real number with resolution of 2^{-M} [11]. Fig. 2 illustrates a left-aligned and a right-aligned unary bit-stream generator, each built from a counter and a comparator. Fig. 3 exemplifies the Min and Max value functions on unary bit-streams. As can be seen, a single `AND` gate implements the Min value function when it is fed with two same-length unary bit-streams. An `OR` gate, on the other hand, performs Max function when connected to two same-length unary bit-streams. Evidently, this implementation is independent of the precision of data; The same design can process input data with higher precision by processing longer bit-streams. Implementing the Min and Max value functions in the weighted binary domain is more complex, and dependent to the precision of data. As shown in Fig. 4, the conventional binary implementation of these functions requires an n-bit comparator and two n-bit multiplexers (MUXs), resulting in a higher hardware area and power cost.

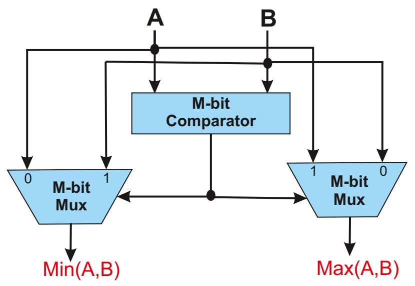

Fig. 4. Implementation of Min and Max value functions in the conventional binary domain.

B. Fuzzy-logic systems

The idea of Fuzzy-logic systems was first introduced by Zadeh in 1965 [16]. Designing a hardware-based fuzzy inference engine is the only solution when high-speed processing is desired [3]. Fuzzy-logic hardware has been designed in both analog and digital domains. Research findings confirm that analog hardware is rather simple. However, it lacks accuracy and reliability. Digital hardware, on the other hand, benefits from a higher accuracy and reliability. Nonetheless, it is often more complex and lacks adequate speed compared to analog design. The Generalized Modus Ponens (GMP) of fuzzy inference rules (single rule with single antecedent) are stated as below [7]:

$$\begin{array}{ll} \text{Rule:} & \text{If input is A then output is B} \\ \text{Fact:} & \text{Input is A'} \\ \hline \text{Consequence:} & \text{Output is B'} \end{array} \tag{1}$$

where A, B, A', B' are fuzzy sets. The rule is described by the fuzzy implication function $R = A \rightarrow B$, where R is a fuzzy relation and, when the "min" operator is used, its membership functions (matching degree) are computed as

$$\mu_R(u,v) = \min(\mu_A(u), \mu_B(v)); \qquad u \in \cup, v \in \vee. \tag{2}$$

where $\mu_{\mathrm{R}}(u,v)$, $\mu_{\mathrm{A}}(u)$, $\mu_{\mathrm{B}}(v)$ are membership functions of R, A, and B, and $\cup$ and $\vee$ are universes of discourse of A and B, respectively. The *Consequence* in (1) is determined by the fuzzy composition B'=A'°R and, when the "max-min" operator is used, the membership function value of B' is computed as

$$\mu_{\mathrm{B'}}(v) = \max \ \min(\mu_{A'}(u), \mu_{\mathrm{R}}(u,v)) \tag{3}$$

where $\mu_{\mathrm{B'}}(v)$ and $\mu_{\mathrm{A'}}(u)$ are the membership functions of B' and A', respectively.

The following example demonstrates the fuzzy inference process in a basic tipping problem (see Fig. 5). The rules (with multiple antecedents) are stated as below:

Rule 1: **If** service is poor **And** food is rancid **then** tip is cheap.
Rule 2: **If** service is good **then** tip is average.
Rule 3: **If** service is excellent **Or** food is delicious **then** tip is generous.

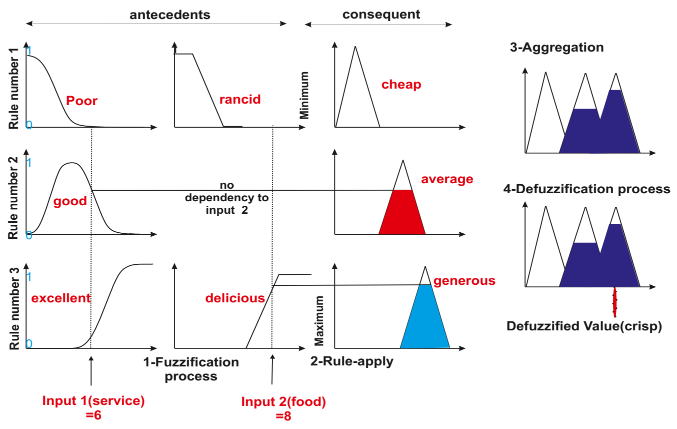

Fig. 5. Graphical representation of fuzzy inference process in a basic tipping problem.

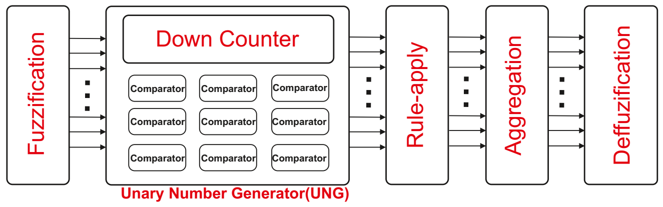

Fig. 6. The general structure of the proposed architecture.

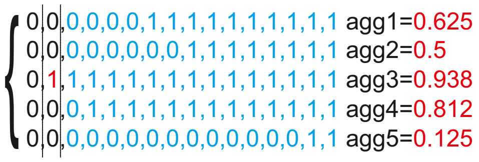
0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1 agg1=0.625
0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1 agg2=0.5
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1 agg3=0.938
0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1 agg4=0.812
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1 agg5=0.125

Fig. 7. An example of the defuzzification process.

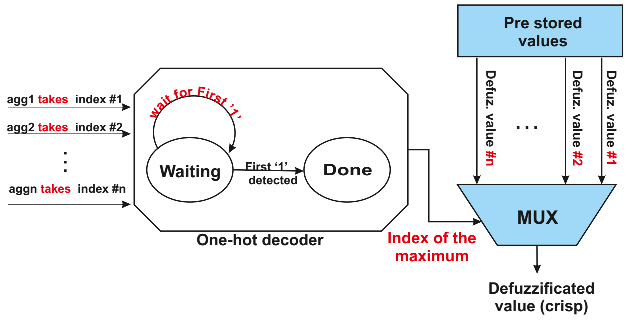

Fig. 8. Proposed architecture for the defuzzification process.

Multiple inference rules can be operated simultaneously with a sample of crisp input value. Due to the fact that most fuzzy-logic systems need to characterize their crisp output, they pursue specific defuzzification strategies in their process.

III. The Proposed Approach

In this section, we first introduce our proposed architecture for the fuzzy-inference process. We then evaluate the proposed design, and present the simulation and synthesis results.

A. Proposed System Architecture

Fig. 6 illustrates the general structure of the proposed architecture for the fuzzy-inference engine. First, the input value (crisp value) is converted to a linguistic value based on the input membership function. The numbers in the range of $[0, 1]$ are converted to unary bit-streams using a unary number generator (UNG) unit. A single down-counter is shared in generating all unary bit-streams to minimize the bit-stream generation overhead. *Rule-apply* and *Aggregation* processes are Min and Max-based functions and hence, implemented using simple AND and OR logic gates.

We implement the Middle-of-Maxima (MOM) [6] as the defuzzification method. Fig. 7 represents an example of the deffuzification process. agg1, agg2, agg3, agg4 and agg5 are the aggregated values of the fuzzy inference process. The defuzzification process needs to find the index of the aggregated value with maximum value. In the example shown in Fig 7, agg3 has the maximum value and hence its index (i.e., 3) must be sent to the output. Due to using a down-counter in generating bit-streams, the aggregated values are converted to right-aligned unary bit-streams (first all 0s and then all 1s). When processing some right-aligned unary bit-streams, the bit-stream that generates the first "1" is the bit-stream with maximum value. So, there is actually no need to generate and process full length bit-streams (i.e., 2^n-bit bit-streams for n-bit precision data) to find the index of the input with maximum value. In practice, this index is found in a shorter time than 2^n cycles. For example, for the set of inputs given in Fig. 7, the index of the input with maximum value is found after only two clock cycles.

Fig. 8 demonstrates the proposed architecture for the defuzzification process. The outputs of the *Aggregation* process, namely agg1, agg2, ..., and agg$n$, take index1, index2, ..., and Index$n$, respectively. The unary bit-streams corresponding to the aggregated values are transmitted to a one-hot decoder to detect the bit-stream that generate the first 1 (i.e., the bit-stream with maximum value.) As soon as the one-hot decoder detects one of the patterns (1,0,0,0,0), (0,1,0,0,0), (0,0,1,0,0), (0,0,0,1,0), or (0,0,0,0,1), the corresponding index (e.g., 3 in the example of Fig. 7) will be sent to the MUX unit to select the corresponding defuzzification value.

TABLE I
Average Number of Processing Cycles for the Proposed Design

Design Methodology	Input Precision (M)					
	2	3	4	5	6	7
Conventional Binary	1	1	1	1	1	1
Proposed **5Inp./Out.**	1.51	2.22	3.67	6.92	12.73	24.82
Proposed **7Inp./Out.**	1.61	2.23	3.78	6.52	12.44	23.94
Proposed **9Inp./Out.**	1.26	1.90	2.81	4.68	8.77	16.78

Table I reports the average number of processing cycles for the proposed fuzzy-inference architecture for three different number of input/output membership values when processing 20 randomly selected sets of input data. As can be seen, the average number of processing cycles with the proposed unary designs are significantly smaller than 2^n, the required bit-stream length to accurately represent an n-bit precision data in the unary domain [11].

B. Design Evaluation

We implemented three different fuzzy-inference controllers with 25, 49, and 81 inference rules (five, seven, and nine input/output fuzzy-linguistic values) for a system with two input/output variables that controls two velocities. Table II illustrates the fuzzy rule table for the case of seven input/output fuzzy-linguistic values. Fig. 9 demonstrates the graphical representation of the input/output membership function for this case. NL, NS, Z, PS, PM, PL are the abbreviations for Negative Large, Negative Small, Zero, Positive Small, Positive Medium and Positive Large, respectively. For example, if input$_1$ is PM and input$_2$ is NS, the output is PM.

TABLE II
Fuzzy Rule Table in the Case of Seven Input/Output Fuzzy-Linguistic Values

Inp$_1$ \ Inp$_2$	PL	PM	PS	Z	NS	NM	NL
PL	PL	PL	PL	PL	PL	PS	PS
PM	PL	PL	PM	PM	PM	PS	PS
PS	PS	PS	PS	Z	Z	PS	PS
Z	Z	Z	Z	Z	Z	Z	Z
NS	NS	NS	Z	Z	NS	NS	NS
NM	NL	NL	NM	NM	NM	NS	NL
NL	NL	NL	NL	NL	NL	NM	NM

TABLE III
AREA, POWER, DELAY, ENERGY CONSUMPTION, AND HARDWARE USAGE (# OF LUTS) COMPARISON

Design	M	Area ($\mu m^2 \times 1000$)		Power (mW) (@max freq)		Critical Path (ns)		Energy (pJ)		Hardware Usage (# of LUTs)	
		Conv.	Prop.	Conv.	Prop.	Conv.	Prop.	Conv.	Prop.	Conv.	Prop.
5Inp./Out.	2	1.873	0.811	0.289	0.232	1.37	0.95	0.396	0.355	96	34
	3	2.736	1.108	0.395	0.304	3.33	0.98	1.316	0.667	235	38
	4	4.446	1.482	0.506	0.376	3.50	0.99	1.770	1.413	326	39
	5	5.785	1.685	0.509	0.351	3.92	1.14	1.996	2.616	395	42
	6	7.072	2.000	0.596	0.401	4.63	1.18	2.761	5.890	534	47
	7	8.350	2.392	0.583	0.380	5.62	1.25	3.28	11.379	808	62
7Inp./Out.	2	3.047	1.123	0.496	0.425	2.71	1.14	1.344	0.734	182	65
	3	5.578	1.559	0.407	0.285	5.02	1.20	2.043	0.763	403	72
	4	7.516	2.079	0.494	0.345	5.93	1.27	2.931	1.616	606	75
	5	9.534	2.35	0.474	0.305	7.88	1.37	3.740	2.895	643	78
	6	11.439	2.807	0.460	0.284	9.15	1.49	4.212	5.395	896	79
	7	13.515	3.317	0.532	0.323	9.85	1.61	5.244	12.906	1342	83
9Inp./Out.	2	5.205	1.478	0.499	0.415	3.72	1.32	1.857	0.691	268	104
	3	8.795	1.991	0.513	0.371	5.51	1.33	2.830	0.939	690	113
	4	12.230	2.669	0.536	0.387	7.16	1.41	3.842	1.537	1043	115
	5	15.798	3.030	0.552	0.319	8.51	1.52	4.701	2.271	1283	117
	6	19.406	3.615	0.638	0.365	8.60	1.54	5.491	4.938	1546	120
	7	22.856	4.265	0.703	0.380	10.65	1.67	7.496	10.648	2321	139

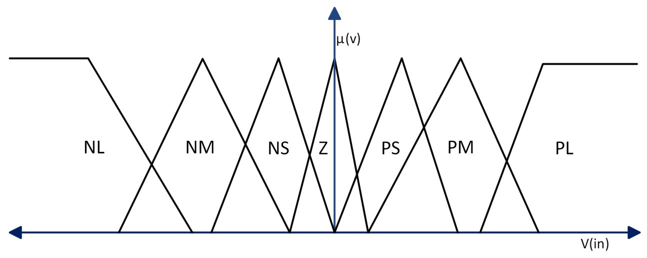

Fig. 9. Graphical representation of the input/output membership function for the case of seven input/output linguistic values.

For performance evaluation, we implemented both the proposed unary bit-stream-based and the conventional design of the fuzzy-inference controller in MATLAB. For hardware cost comparison, we developed RTL VHDL description for the proposed and the conventional binary design. The designs were synthesized using the Synopsys Design Compiler with the 45nm FreePDK library and also with Xilinx Vivado v.2018.2 on a Virtex-7 FPGA. We report the synthesis results for different data bit-widths (i.e., $M = 2, 3, 4, 5, 6,$ and 7.)

Table III reports the synthesis results in terms of hardware footprint area, power consumption at maximum working frequency, critical path latency, energy consumption, and the number of LUTs used to implement the fuzzy-logic designs. The energy consumption of the proposed design is calculated by finding power $\times$ critical path latency $\times$ average number of cycles where the average number of cycles is extracted from Table I. In case of implementing the fuzzy-logic controller with 25 inference rules (5Inp./Out.) up to 72% saving in the hardware area, 35% reduction in the power consumption, 50% improvement in the energy consumption, and 92% saving in the occupied LUTs are observed. As can be seen in the reported numbers, by increasing the number of inference rules a higher saving in the hardware design cost can be achieved. For instance, for the case of implementing a fuzzy-inference engine with 49 inference rules (7Inp./Out.), up to 75, 39, 63, and 93 percent improvements in the hardware area, power, energy, and occupied LUTs are observed. Also, for the case of implementing a fuzzy-inference controller with 81 inference rules (9Inp./Out.), up to 82 percent improvement in the hardware area is achieved. Power and energy consumption are reduced up to 46 and 67 percent, respectively. For the FPGA-based design for this case the number of occupied LUTs is reduced up to 94% with the proposed design compared to the conventional binary design. As can be seen, the rate of energy saving decreases by increasing the data-width as the number of processing cycles in the proposed design increases by increasing the precision of data. Therefore, the proposed design offers a significant saving in the hardware area and power costs for all data-widths, and an improved energy consumption for low data bit-widths.

IV. CONCLUSION

This work applies the concept of unary computing to the platform of fuzzy-inference engine. A low-cost and high-performance unary-stream-based fuzzy-inference controller is developed. The processing is deterministic and produces the same output as the conventional binary design. An overhead is the cost of converting data from weighted binary to unary bit-streams using a unary number generator unit. The more inference rules, the higher the saving in the hardware cost. Synthesis results for the case of implementing a fuzzy-inference controller with 81 inference rules show up to 82% saving in area, 46% reduction in power, 67% saving in energy consumption, and 94% reduction in the LUT usage compared to the conventional binary design.

References

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

[2] Zeungnam Bien and Kyung Chan Min. *Fuzzy Logic and its Applications to Engineering, Information Sciences, and Intelligent Systems*, volume 16. Springer Science & Business Media, 2012.

[3] F. Colodro, A. Torralba, and L. G. Franquelo. A digital fuzzy-logic controller with a simple architecture. In *Proceedings of IEEE International Symposium on Circuits and Systems - ISCAS '94*, volume 2, pages 101–104 vol.2, May 1994.

[4] S. R. Faraji and K. Bazargan. Hybrid Binary-Unary Hardware Accelerators. In *2019 24th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2019.

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

[6] Hans Hellendoorn and Christoph Thomas. Defuzzification in fuzzy controllers. *J. Intell. Fuzzy Syst.*, 1(2):109–123, March 1993.

[7] Donald L. Hung and William F. Zajak. Design and implementation of a hardware fuzzy inference system. *Information Sciences - Applications*, 3(3), 1995.

[8] Devon Jenson and Marc Riedel. A Deterministic Approach to Stochastic Computation. In *Proceedings of the 35th International Conference on Computer-Aided Design*, ICCAD '16, New York, NY, USA, 2016.

[9] M. Hassan Najafi, S. Rasoul Faraji, Kia Bazargan, and David Lilja. Energy-Efficient Pulse-based Convolution for Near-Sensor Processing. In *IEEE Intern. Symp. on Circuits and Systems (ISCAS)*, May 2020.

[10] M. Hassan Najafi, S. Jamali-Zavareh, D. J. Lilja, M. D. Riedel, K. Bazargan, and R. Harjani. Time-Encoded Values for Highly Efficient Stochastic Circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(5):1644–1657, May 2017.

[11] M. Hassan Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan. Low-Cost Sorting Network Circuits Using Unary Processing. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 26(8):1471–1480, Aug 2018.

[12] W.J. Poppelbaum. Burst processing: A deterministic counterpart to stochastic computing. In *Proceedings of the 1st International Symposium on Stochastic Computing and its Applications*. 1978.

[13] W.J. Poppelbaum, A. Dollas, J.B. Glickman, and C. O'Toole. Unary processing. In *Advances in Computers*, volume 26, pages 47 – 92. Elsevier, 1987.

[14] Weikang Qian, Xin Li, M.D. Riedel, K. Bazargan, and D.J. Lilja. An Architecture for Fault-Tolerant Computation with Stochastic Logic. *Computers, IEEE Trans. on*, 60(1):93–105, Jan 2011.

[15] X. Song, Z. Zhai, P. Zhu, and J. Han. A stochastic computational approach for the analysis of fuzzy systems. *IEEE Access*, 5:13465–13477, 2017.

[16] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965.