# Fuzzy-logic Processing using Unary Bit-Streams

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
jalilvand_a@comp.iust.ac.ir, najafi@louisiana.edu, m_fazeli@iust.ac.ir

***Abstract*—there is a growing attention to the theory of fuzzy-logic and its applications. An efficient design of fuzzy-inference hardware has become the only solution in many high-performance applications. Considering the fact that fuzzy-logic variables have truth value in the $[0, 1]$ interval, this work proposes to apply the concept of processing unary bit-streams to the platform of fuzzy-inference systems. In unary processing, data is encoded as serial bit-streams with the value defined by the fraction of 1s in a stream of 1s and 0s. Long latency is a challenge with unary processing. To mitigate the long latency problem, we propose a novel idea which significantly reduces the latency at no accuracy loss. Synthesis results show that the proposed architecture provides up to 75.5% saving in hardware area, 40% reduction in power consumption, and 53% saving in energy consumption compared to traditional weighted binary implementation.**

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

## I. INTRODUCTION

Recent processing hardware is bounded by some strict design constraints such as low power consumption, small circuit area, and reliability. Power consumption and hardware area cost in particular are the important concerns in designing embedded systems. Considering the limitations of the conventional computing methods in hardware efficient design, unconventional design techniques are receiving more and more attention.

*Unary computing* is one of such unconventional techniques that first intorduced in 1980s [8]. The paradigm has some characteristics common to *stochastic computing* (SC) [2] but is deterministic and produces completely accurate results. The operations are mainly on the numbers in the $[0, 1]$ interval. Unlike weighted binary-radix, 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). We call the streams in this format "unary bit-streams". 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 can be used to multiply two numbers represented by unary bit-streams [5]. This particularly is useful in hardware efficient design of convolution engines for neural networks. Minimum (Min) and maximum (Max) value functions based on unary bit-streams are discussed in [7] 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 processing unary bit-streams to the platform of fuzzy-logic controller.

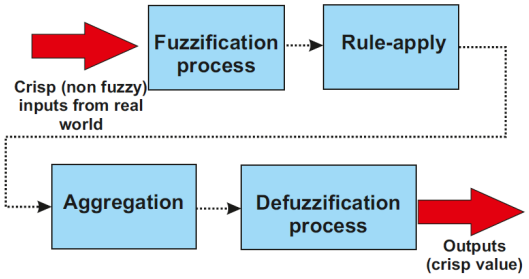


Fig. 1. Inference mechanism in fuzzy-logic applications.

There has been a growing attention to the fuzzy-logic application since its initial work in [9]. As the concept of fuzzy-logic is very close to human reasoning, fuzzy-logic controllers are easy to perception and design. The theory of fuzzy-logic 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. As illustrated in Fig. 1, most fuzzy-logic applications use the mechanism of inference which employs the following steps:

- Fuzzification process: converting crisp (non fuzzy) inputs to a fuzzy-linguistic value.
- 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 binary sets (true or false), fuzzy-logic variables have truth value in the $[0, 1]$ interval. We exploit the concept of processing unary bit-streams with the aim of improving the hardware design cost of fuzzy-logic systems. Our synthesis results show a significant improvement in the hardware area, power, and energy consumption compared to the conventional weighted binary-based implementation at no accuracy loss.

The rest of this paper is organized as follows. Section II provides the background on unary computing and fuzzy logic systems. Section III discusses the proposed method and provides the synthesis results. Section IV concludes the paper.

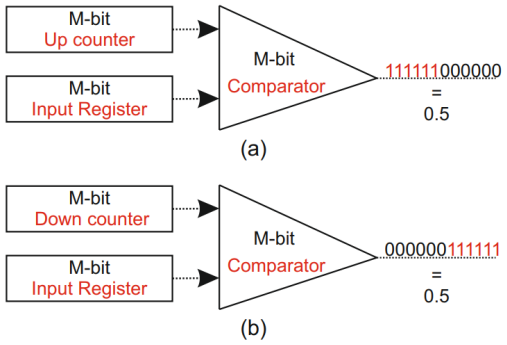


Fig. 2. Unary bit-stream generator: (a) left-aligned bit-stream, (b) right-aligned bit-stream.

## II. BACKGROUND

### A. Unary processing

Weighted binary radix has been the dominate format for representation of numbers in the field of computer engineering. This representation is compact. Nonetheless, computation on this representation is rather complex as each bit has its own weight according to its position. Moreover, this format is not immune to noise: flipping the most significant bit (MSB) results in a huge error in the computation.

In SC, numbers in the $[0, 1]$ interval are encoded by the probability of getting a 1 versus a 0 in a stream of interleaved 0s and 1s. A stream of $2^M$ bits is required to demonstrate a real number with resolution of $2^{-M}$ [7]. All digits have the same weight with a stochastic representation (there is no most/least significant bit). Noise immunity, therefore, increases compared to the weighted binary radix. The fact that representation of numbers in SC is less compact than the weighted binary leads to a significantly higher computation time and energy consumption. However, complex arithmetic functions can be implemented using simple logic, e.g., an AND gate can implement the multiplication operation.

Unary computing is a hybrid technique that has characteristics common to both binary radix and to SC. In unary computing, numbers are encoded uniformly by a sequence of one value (say 1) followed by a sequence of the other value (say 0). Fig 2 illustrates a left-aligned and a right-aligned unary bit-stream generator. Similar to stochastic bit-streams, all bits have the same weight in unary bit-streams. Thus, the bit-streams have a similar immunity to noise. Unlike SC, computation on unary bit-streams are deterministic and completely accurate.

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's fed with two unary bit-streams. An OR gate, on the other hand, performs Max function on unary bit-streams. Evidently, the implementation is independent of the precision of data. As shown, both real-valued and integer numbers can be represented and processed. Implementing the Min and Max functions in the weighted binary domain is more complex and dependent to the precision of data. As can be seen in Fig. 4, the conventional binary

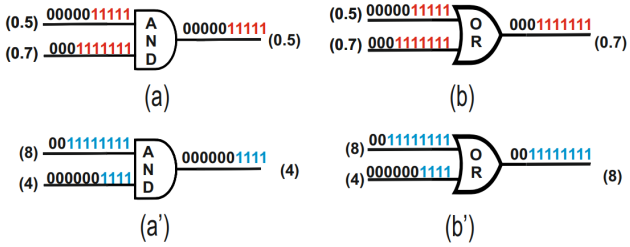


Fig. 3. minimum and maximum operation in unary computing on real-valued and integer numbers.

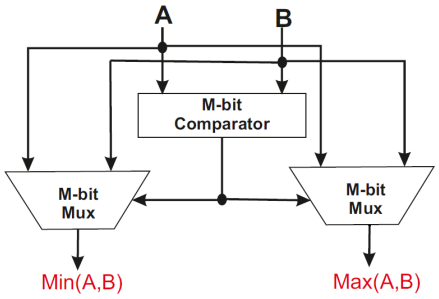


Fig. 4. Implementation of Min and Max value functions in weighted binary radix domain.

implementation of these arithmetic functions requires an $n$-bit comparator and two $n$-bit multiplexers (MUXs) which results in a larger circuit area and a higher power consumption.

In a recent application of unary processing, Najafi et al. proposed a novel area and power-efficient synthesis approach based on unary processing for low-cost implementation of sorting network circuits [6], [7]. Their synthesis results show a significant reduction in the power consumption and hardware area footprint compared to the conventional binary radix-based implementation.

### B. Fuzzy-logic systems

The idea of Fuzzy-logic systems was first introduced by Zadeh in 1965 [9]. Designing a hardware-based fuzzy inference engine is the only solution when the processing speed expectation reaches a certain level [1]. 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 implementation. The Generalized Modus Ponens (GMP) of fuzzy inference rules (single rule with single antecedent) are stated as below [4]:

| | | |
|---|---|---|
| Rule: | If input is A then output is B | |
| Fact: | Input is A' | (1) |
| Consequence: | Output is B' | |

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 $\mu_R(u,v)$, $\mu_A(u)$, $\mu_B(v)$ 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

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

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

The following example demonstrates the fuzzy inference process in a basic tipping problem (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.

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

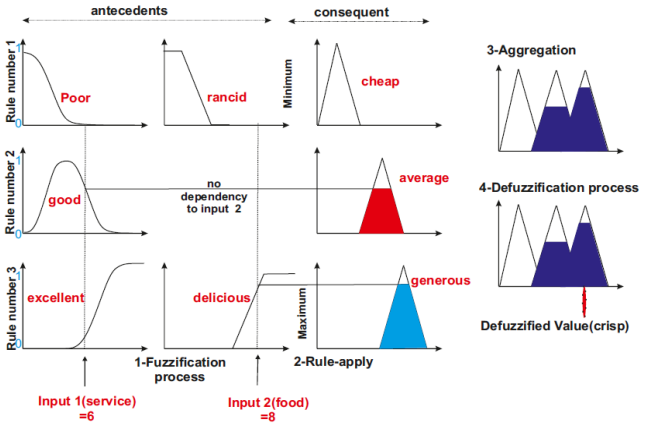


Fig. 5. Graphical representation of fuzzy inference process in a basic tipping problem.

Bit-stream processing based on SC was previously applied to digital fuzzy-logic controller design in [1]. The design of [1] uses a module that computes the membership function of the input variable. The main challenges, however, are the lack of generality in type of membership functions and a long processing time due to processing long bit-streams.

## III. THE PROPOSED APPROACH

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

### A. Proposed System Architecture

As discussed earlier, simple standard AND and OR gates implement the Min and Max value functions in the unary domain. This provides the opportunity to design a low-cost unary stream-based fuzzy-inference engine. The challenge, however, is the long latency of processing unary bit-streams which further translates to high energy consumption. For instance, for 6- and 8-bit precision unary design the system needs to run for 64 and 256 clock cycles, respectively. We mitigate this long latency issue in our developed architecture and propose a novel high-performance platform for fuzzy-inference systems.

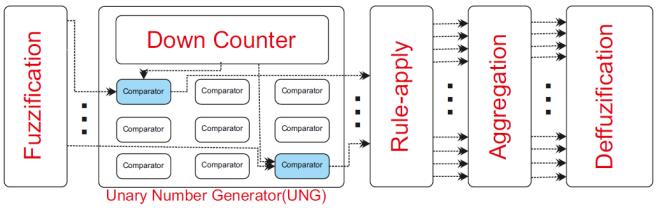


Fig. 6. The general structure of the proposed architecture

Fig. 6 illustrates the general structure of our proposed architecture. 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 by using a unary number generator (UNG) unit. A single down counter is shared in generating all unary bit-streams. *Rule-apply* and *Aggregation* processes will be done using simple logic gates.

For evaluation purposes, we select Middle-of-Maxima (MOM) [3] 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 with values 0.625, 0.5, 0.938, 0.812 and 0.125. These correspond to bit-streams $(0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1)$, $(0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1)$, $(0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)$, $(0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1)$, and $(0,0,0,0,0,0,0,0,0,0,0,0,0,1,1)$, respectively. As can be seen, due to using a down counter in generating bit-streams the generated unary streams are all right-aligned.

While prior unary designs were interested in finding the maximum value between some input data, in the implemented fuzzy-inference engine we are interested in finding the index of the input data with the maximum value. Consider the example shown in Fig 7. agg3 has the maximum value and its index must be returned as the output. When processing right-aligned unary bit-streams the bit-stream that generates the first "1" is the bit-stream with maximum value. So, in most cases, there is actually no need to generate and process bit-streams for $2^n$ cycles. In practice, the index of the bit-stream with maximum value is found in a smaller number of cycles. For example, for the inputs provided in Fig. 7, instead of processing the bit-streams for 16 clock cycles, the index of the input with maximum value (i.e., 3) is found after only two clock cycles.

Fig. 8 demonstrates the defuzzification process in our proposed architecture. The outputs of the *Aggregation* process agg1, agg2, ..., and agg$n$, take index1, index2, ..., and


Fig. 7. An example of the deffuzification process

Index$n$, respectively. These bit-streams are transmitted to a one-hot decoder to find the index of the bit-stream with the first "1". 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), and (0,0,0,0,1), the corresponding index (e.g., index3 in the case of Fig. 7) will be sent to the MUX unit to select the corresponding defuzzification value.

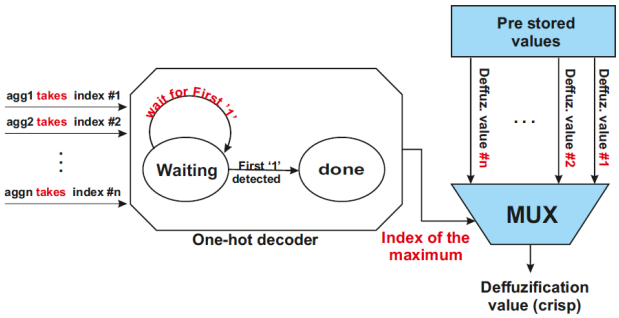


Fig. 8. General architecture of deffuzification process

Table I reports the average number of processing cycles required by the proposed architecture for 20 randomly selected sets of input data.

TABLE I
AVERAGE NUMBER OF PROCESSING CYCLES FOR THE PROPOSED DESIGN

| Design Methodology | Input Precision (M) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Conventional Unary | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
| **Proposed** | 1.8 | 2.8 | 4.8 | 9.2 | 17.6 | 34.5 | 68.3 | 132.3 |

### B. Design Evaluation

We implemented a two-input two-output fuzzy inference system with 49 fuzzy-inference rules (seven input/output fuzzy-linguistic values) to control two velocities. For performance evaluation, we implemented both the proposed unary stream-based design and the conventional fuzzy-inference system in MATLAB. For hardware cost evaluation, we developed RTL VHDL description of the fuzzy-inference system for both the proposed and the conventional binary designs. The designs are synthesized using the Synopsys Design Compiler with a 45nm standard-cell library. We report the synthesis results for different data bit-width ($M = 2, 3, 4, 5, 6, 7$ and $8$.)

Table II reports the synthesis results in terms of hardware area, power consumption at maximum working frequency, critical path latency, and energy consumption. As can be seen, a significant improvement (up to 75.5 percent) in the hardware area footprint is achieved with the proposed design. Power and energy consumption are also reduced up to 40 and 53 percent, respectively, compared to the conventional binary radix-based design. Note that the energy consumption of the proposed design is measured by power $\times$ critical path latency $\times$ the average number of cycles from Table I.

TABLE II
SYNTHESIS RESULTS FOR AREA, POWER, DELAY AND ENERGY CONSUMPTION

| M | Area ($mm^2$) | | Power($mW$) (@max freq) | | Critical path ($ns$) | | Energy ($pJ$) | |
|---|---|---|---|---|---|---|---|---|
| | Conv. | Pro. | Conv. | Pro. | Conv. | Pro. | Conv. | Pro. |
| 2 | 3.05 | 1.12 | 0.50 | 0.42 | 2.71 | 1.14 | 1.34 | 0.87 |
| 3 | 5.58 | 1.55 | 0.41 | 0.29 | 5.02 | 1.2 | 2.04 | 0.96 |
| 4 | 7.51 | 2.07 | 0.49 | 0.35 | 5.93 | 1.27 | 2.93 | 2.12 |
| 5 | 9.53 | 2.35 | 0.47 | 0.30 | 7.88 | 1.37 | 3.74 | 3.86 |
| 6 | 11.44 | 2.80 | 0.46 | 0.28 | 9.15 | 1.49 | 4.21 | 7.49 |
| 7 | 13.51 | 3.31 | 0.53 | 0.32 | 9.85 | 1.61 | 5.24 | 17.94 |
| 8 | 15.37 | 3.79 | 0.51 | 0.30 | 11.76 | 1.66 | 6 | 34.77 |

## IV. CONCLUSION

This work applies the concept of unary computing to the platform of fuzzy-inference systems. A low-cost and high-performance unary-stream-based fuzzy-inference controller is developed. The processing is deterministic with no accuracy loss in the computation. The only overhead is the cost of converting data from weighted binary to unary bit-streams using a unary number generator unit, and the cost of a defuzzification controller. The more inference rules, the higher the saving in the hardware cost. Synthesis results for the case of a fuzzy inference system with 49 rules show up to 75.5% saving in area, 40% reduction in power, and 53% saving in energy consumption compared to the conventional binary radix-based implementation.

## REFERENCES

[1] 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.

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

[3] H. Hellendoorn and C. Thomas. Defuzzification in fuzzy controllers. *J. Intell. Fuzzy Syst.*, 1(2):109–123, Mar. 1993.

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

[5] 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 Transactions on Very Large Scale Integration (VLSI) Systems*, 25(5):1644–1657, May 2017.

[6] M. H. Najafi, D. J. Lilja, M. Riedel, and K. Bazargan. Power and area efficient sorting networks using unary processing. In *2017 IEEE Intern. Conf. on Computer Design (ICCD)*, pages 125–128, Nov 2017.

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

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

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