Unary Positional Computing

McKenzie van der Hagen and Marc Riedel
vand0955@umn.edu, mriedel@umn.edu
Department of Electrical and Computer Engineering, University of Minnesota

***Abstract*—This paper introduces a novel representation that is a hybrid of unary and positional systems. It builds upon unary computation, a recent evolution in the field of stochastic computing. In contrast to the stochastic approach, the unary approach is completely accurate, with no random fluctuations. It requires less area and has much lower latency. However, compared to a conventional binary approach, the latency is still unacceptably high for many applications. The unary positional system proposed in this paper reduces the latency exponentially, with only a modest increase in the hardware complexity. Constructs for arithmetic operations such as addition and multiplication are presented and their performance is evaluated.**

I. Introduction & Background

Stochastic computing allows complex operations to be performed with simple logic, but it suffers from high latency and poor precision [1], [2], [3], [4], [5]. Furthermore, the results are always somewhat inaccurate due to random fluctuations. The random or pseudorandom sources required to generate the representation are costly, consuming a majority of the circuit area (and diminishing the overall gains in area).

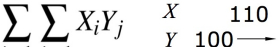
Prior work showed that randomness is *not*, in fact, a requirement for this computational paradigm [6]. If properly structured, the same arithmetical constructs can operate on *deterministic* bit streams, with the data represented uniformly by the fraction of ones versus zeroes.

Conceptually, an operation such as multiplication in stochastic logic works by randomly *sampling* the inputs. This is achieved by randomizing the input bit streams and then intersecting them. This approach is easy to understand but incurs a lot of overhead. Creation of the random bit streams, say with constructs such as LFSRs, is costly. Furthermore, one must do a *lot* of sampling. Indeed, in order to obtain a result that is equivalent in precision to n binary bits, one must sample 2^{2n} bits. Randomness requires, in effect, "oversampling" to get a statistically accurate result [7].

But is such random sampling necessary? Why not simply intersect two deterministic bit streams? In [6], the authors showed that all the constructs that work on stochastic bit streams can operate on deterministically-generated bit streams by utilizing convolution. Illustrated by Figures 1 and 2 convolution intuitively consists of three operations: slide, multiply, and sum. If we implement this operation on uniform deterministic bit streams, the result will be the completely accurate equivalent to a stochastic operation. Furthermore, it will be completely absent of random fluctuations.

Without the requirement of randomness, bit streams can be generated inexpensively. More importantly, the latency is reduced by a factor of approximately $1/2^n$, where n represents the number of bits necessary to achieve the desired precision

$$\sum_{i=1}^{L}\sum_{j=1}^{L} X_i Y_j$$



(a) (b)

Fig. 1: Discrete Convolution. (a) Mathematical operation on two bit streams, X and Y. (b) Intuition: convolution is equivalent to sliding one bit stream past the other.

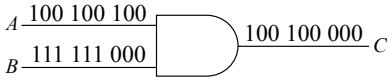


Fig. 2: Multiplication via Convolution, by Clock Dividing a Signal.

in a traditional binary representation. (For example, for the equivalent of 10 binary bits of precision, the bit stream length is reduced from 2^{20} to only 2^{10}.) As is the case with stochastic bit streams, all bits in the deterministic streams are weighted equally. Accordingly, as is the case with stochastic circuits, the deterministic circuits have a high degree of tolerance to soft errors. Throughout this paper, the approach in [6] will be referred to as the **unary** approach.

This paper introduces a novel representation that is a hybrid of the unary and positional systems. We note that an attempt was made to introduce positional computation into stochastic systems [8]. However, beyond the similarities of the underlying approach, the technical details in this paper are completely different, seeing as we target the unary representation.

This paper is structured as follows: Section II discusses the unary positional representation. Section III compares and contrasts the representation to plain unary, as well as to stochastic and binary representations. Section IV presents constructs for performing arithmetic operations on the new representation. Section V evaluates the time and area complexities of the constructs in Section IV. Finally, VI presents conclusions and discusses future directions.

II. Unary Positional Representation

In traditional positional representations such as decimal and binary, numbers are expressed as the sum of values at adjacent positions weighted accordingly by increasing powers of the base (10 for decimal; 2 for binary). The raw value at each position is represented by a unique symbol from a predefined set (from $\{0,\ldots,9\}$ for decimal; from $\{0,1\}$ for binary).

In our unary positional representation, each position is represented by a separate bit stream of length n, the base. We assume that n is a power of 2. The value of each stream is simply the number of ones that it contains, so we have a unary (or *uniform*) representation in this respect. However, the value of the complete number is the sum of the values of the streams weighted according to their position, so it is positional in this respect. With k positions, the weighting at position p is

n^p, for $p = 0, \ldots, k-1$. We can represent a total of n^k distinct numbers.

We note that, with streams of length n, we could in principle represent numbers from 0 to n, so $n+1$ distinct numbers with each stream. However, for technical reasons, the last bit in each stream is always 0. Accordingly, each stream represents values from 0 to $n-1$, so n distinct values. This remains consistent with the definition of a base-n positional representation.

A. Conversion

Conversion from conventional binary to the unary positional representation is straightforward. Bits in each stream are filled in expanding groups, based on the original binary bits: first 1 bit, then 2 bits, then 4 bits, and so on, from right to left. If the binary bit is one, then the group is filled with ones; otherwise it is filled with zeroes. The n^{th} bit in each stream is always set to zero. A stream is created for each group of $\log_2 n$ bits in the original binary representation. An example with 3 streams each of length 8 is shown in Figure 3.

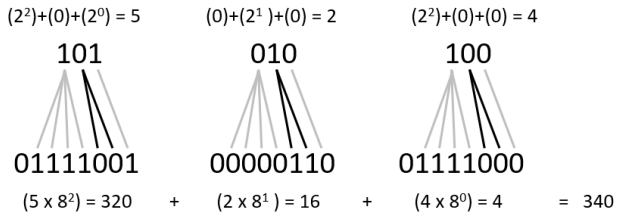


Fig. 3: Conversion of the decimal value 340 from a traditional binary representation to a base-8 Unary Positional representation where n = 8 and k = 3.

III. Comparison of Representations

We compare the representation to conventional binary, to the unary method from [6], and to a stochastic representation. Comparisons are presented in terms of n, the base, and k, the number of positions, in the representation.

TABLE I
Number of bits necessary to represent n^k distinct numbers in various representations where n is the unary positional base and k is the number of positions.

Stochastic	Unary	Unary Positional	Binary
n^{2k}	n^k	nk	$(\log_2 n)k$

A. Resolution

Table I summarizes the number of bits necessary to encode values in the various representations. The number of bits translates directly into the latency of the computation. Due to the inefficiencies of random sampling, a stochastic representation requires very long streams to represent a given value: approximately n^{2k} random bits to represent n^k distinct values. The unary approach proposed in [6] reduces this requirement to n^k bits, while the unary positional method presented here shrinks it even further to the linear relationship of nk. A conventional binary representation improves upon this yet again, reducing the number of bits to $(\log_2 n)k$ bits, the optimal number. The four methods represent an exponential sliding scale (with stochastic computing at the disastrous left end, in our opinion).

B. Range

The stochastic representation produces fractional numbers in the range from $[0, \ldots, 1]$, with the unipolar interpretation; or fractional numbers in the range $[-1, \ldots, 1]$ with the bipolar representation [1]. Although it is possible to map these values to other ranges, the representation is naturally limited to these fractional values.

The unary positional representation that we are proposing here incorporates weighting. Depending on the assigned weighting, the range be can extended to positive integer values of arbitrary size. Furthermore, if we include a radix point, mixed whole and fractional numbers can be represented. This extension is straightforward; we do not explore it further here. Note that the unary positional system can also be used to represent negative numbers via a two's complement representation. An example is given in Figure 4.

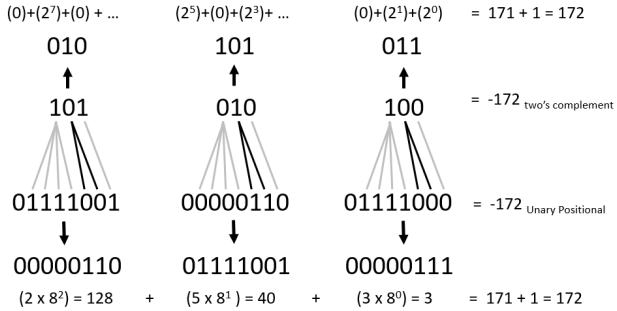


Fig. 4: Conversion and interpretation of -172 in Two's Complement and Unary Positional.

Recall that a two's complement number is interpreted as negative if it has a leading one. Accordingly, a unary positional number is interpreted as negative when at least half of the bits in the highest weighted position are one. As usual, the magnitude of a negative number can be determined by flipping all of the bits and adding one. (Note that the leftmost zero in each position should remain, as this is acting as a filler.)

C. Fault Tolerance

Any entirely uniform representation, including both stochastic and unary, provides a high degree of tolerance to soft errors (i.e., bit flips). With all bits weighted equally, a single bit flip changes the value by only a single increment or decrement of the precision of the representation. In contrast, conventional binary provides poor tolerance to soft errors. A single bit flip in the most significant bit changes the value by $\frac{1}{2}n^k$

The unary positional representation presented here falls in between these two extremes in terms of tolerance to soft errors. The most impactful bit flips occur in the highest weighted position. Here a single bit flip can change the value by n^{k-1} times the resolution. Table II summarizes the soft-error tolerance of the various representations.

TABLE II Comparison of Fault Tolerance:
Maximum error introduced with a single bit flip. Expressed in terms of the unary positional values of n, the base, and k, the number of positions.

Unary	Unary Positional	Binary
1	n^{k-1}	$(1/2)n^k$

IV. Computation

We discuss constructs for performing basic arithmetic operations with the representation.

A. Carryover

As with any positional representation, the challenge with arithmetic is the carryover between positions. Intuitively, carryover in our unary positional representation occurs when we have a full group of n bits in a given stream. This translates to a single bit in the adjacent stream with the next highest weighting.

Note that, within each stream, the representation is unary: the value is solely dependent on the number of ones. Thus, to recognize when a stream contains a full set of n ones, an n-bit shift register is used. The output of the desired arithmetic operation is used as the enable signal of this register, with the data-in hardcoded as one. This way, the ones from the result are captured and condensed while the zeroes are discarded. Accordingly, when the last bit of the register fills with a one, this indicates that the operation has produced a full set of n ones, and a carryover should be generated: a one is generated as the input to the next position, and the register for the current position is reset to all zeroes.[1] These operations happen simultaneously in a single clock cycle. The hardware block for the Carry Unit is shown in Figure 5.

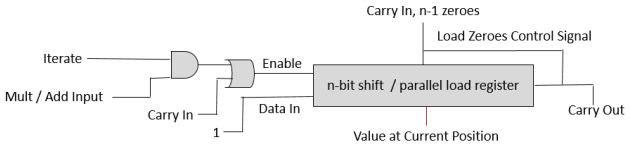


Fig. 5: Carry Unit.

The output capture and carryover propagation is implemented by separate carry units at each position. Thus, as with conventional binary operations, registers must pause while carry propagation is taking place. This behavior is controlled by an "iterate" signal. This signal is the NOR of all of the carry-out signals.

B. Addition

With a unary encoding, addition can be achieved through simple concatenation: the sum of two quantities, each represented by the number of ones in a stream, is simply the total number of ones. To retain the positional aspects, this concatenation is first performed at each position. The results are then condensed and carryovers are appropriately propagated to higher positions.

Figure 6 shows the architecture for 2-input addition. In addition to controlling the shift registers in the carry units, the "iterate" signal also holds the concatenated sums at each position in place until the carryover propagation is complete.

[1]Note that multiple positions can receive carryovers simultaneously. So, in fact, positions are parallel-loaded with the carry-in value and $n-1$ zeroes, rather than all zeroes.

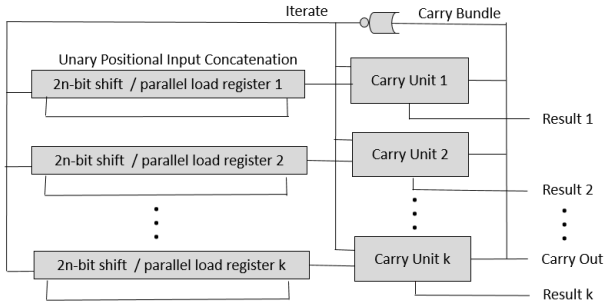


Fig. 6: Unary Positional addition architecture for 2 base-n inputs, with k positions.

C. Multiplication

Fundamentally, multiplication is performed by adding together as many copies of the multiplicand as specified by the multiplier. With a unary encoding, multiplication can be performed with an AND gate by matching every bit of the first operand with every bit of the second [6]. This can be achieved by holding each bit of the first operand constant, while rotating through all the bits of the second operand. (This is termed the "clock division" method.) Details of the approach are found in [6].

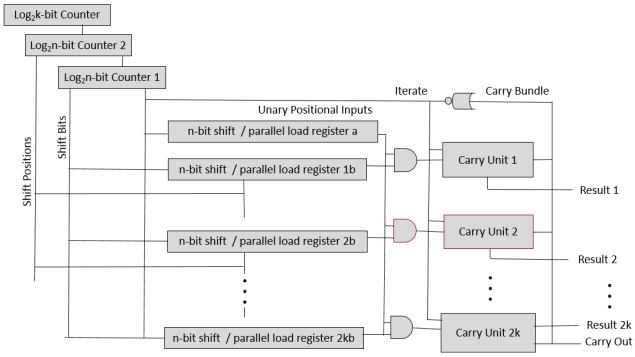


Fig. 7: Full Unary Positional multiplication architecture for two base-n inputs with k positions each. The multiplier is indicated as a and the multiplicand as b.

For a unary positional representation, appropriately aligned inputs are multiplied with an AND gate at each position and carryover is accounted for as previously described. Figure 7 shows the architecture to support these operations. At the outset, the operands are loaded in in the unary positional format. The k positions of the multiplicand are loaded into registers $1b$ through kb while the first position of the multiplier is loaded in to register a. To perform the operation, register a is continuously rotated, exposing each bit of the current position of the multiplier, as a counter increments. When this counter reaches n, indicating that a full rotation is complete, the multiplicand registers are shifted by one, and the process repeats. To account for the positional encoding, the action of shifting the multiplicand also increments a counter. When this counter reaches n, the next position of the multiplier is loaded and each portion of the multiplicand is moved up one position. This process continues for all k positions of the multiplier. When computation completes, the final product is realized

TABLE III
Complexity of Multiplication with Two Operands, with base n and k positions

	Binary	Stochastic	Unary	Unary Positional
Time	$O([(\log_2 n)k]^2)$	n^{4k}	n^{2k}	kn^2
Area	$O([(\log_2 n)k]^2)$	$4(\log_2 n)k$	$2(\log_2 n)k$	$2\log_2 n+\log_2 k+2k+2k(2n+3)+2n$

in the shift registers of each carry unit. Notice in Figure 7 that $2k$ copies of the hardware are needed to account for 2-input multiplication with k positions. Although the product is condensed back to the original format, this expanded hardware is necessary to account for the increased magnitude of the product relative to the size of the inputs.

V. Evaluation of Methodology

The designs in Section IV were implemented in Verilog and simulated to ensure correctness and to evaluate performance. We compare the time-space complexity and scalability of the proposed designs with existing methods.

A. Time-Space Complexity

The latency and area (in terms of gate count) required to implement multiplication for various representations is summarized in Table III.

Binary multiplication is well studied with many complex variations and optimizations. However, it is generally accepted that both the time and space complexities of a straightforward implementation are on the order of $[(\log_2 n)k]^2$.

In both the unary and stochastic methods, multiplication itself can be performed with a single AND gate, but this ignores the cost of generating the bit streams. The area values in Table III include this cost which is proportional to $(\log_2 n)k$. Addition operations in these methods have comparable complexities.

For the unary positional method discussed here, the area complexity directly reflects the circuit components in Figure 7: two $\log_2 n$-bit counters, a $\log_2 k$-bit counter, $2k$ fan-in NOR gate, 2 n-bit registers and three additional gates at each position, as well as 2 n-bit registers to hold the multiplier and final carry out. The time complexity reflects the time necessary for each of the n bits of the multiplier to see each of the n bits of the multiplicand at each of the k positions. Similar examination of the addition architecture in Figure 6 yields an area complexity of $3nk+n+k$ and a time complexity of $2n$.

Figure 8 plots time-area complexity for 2-input multiplication for the unary and unary positional methods. The length of the unary bit streams creates an exponential relationship between n, the base, and k, the number of positions, that rapidly compounds into an overwhelming time delay. In contrast, the complexities for the unary positional representation reflect a linear relationship between n and k that are influenced by only constant exponential factors.

From the positional perspective, Figure 9 focuses on the area complexity comparison between binary and unary positional multiplication, which quickly favors the latter for large values of k.

Multiplication with i inputs, for $i > 2$ is not discussed here due to length restrictions. Its time-space complexity is summarized in Table IV.

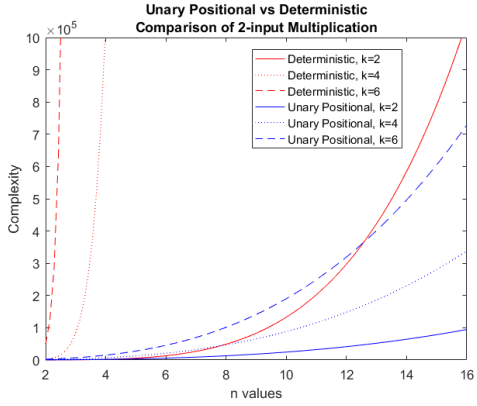


Fig. 8: Graphical comparison of time-space complexities for 2-input multiplication with unary (red) and unary positional (blue) representations for varying values of n, the base, and k, the number of positions.

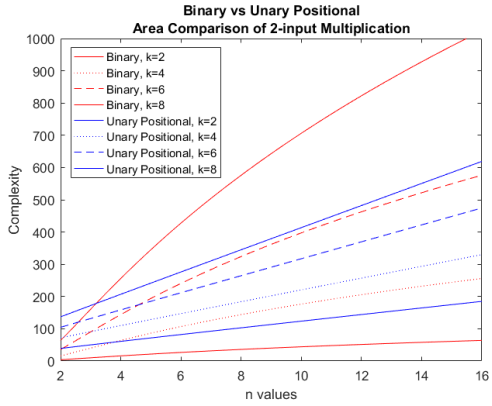


Fig. 9: Graphical comparison of space complexities for 2-input multiplication with binary (red) and unary positional (blue) representations for varying values of n, the base, and k, the number of positions.

TABLE IV
Complexity of i-input multiplication

	Stochastic	Unary	Unary Positional
Time	n^{2ki}	n^{ki}	$k(i-1)n^2$
Area	$(\log_2 n)ki^2$	$(\log_2 n)ki$	$2\log_2 n+\log_2 k+ki+ki(2n+3)+2n$

VI. Future Directions

Comparing stochastic methods, to the unary method in [6], to the unary positional method proposed here, to conventional binary, we see a sliding scale of tradeoffs: the stochastic and unary methods require very long latency, but require very simple hardware. Conventional binary is the most compact representation, but operating on it requires relatively complex hardware. The unary positional system proposed here hits an attractive sweet spot, having much lower latency than stochastic and unary, yet still smaller hardware complexity than conventional binary. In future work, we will apply the method to more complex functions, such as polynomials, encountered in fields such as image and signal processing.

REFERENCES

[1] B. R. Gaines, *Stochastic Computing Systems*, ser. Advances in Information Systems Science. Springer US, 1969.

[2] B. D. Brown and H. C. Card, "Stochastic neural computation i: Computational elements," *IEEE Transactions On Computers, vol. 50, no. 9, pages 891-905*, 2001.

[3] W. Qian and M. D. Riedel, "Synthesizing logical computation on stochastic bit streams," *Proceedings of the Design Automation Conference*, 2009.

[4] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Transactions on Computers*, vol. 60, pp. 93–105, 2011.

[5] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Transaction on Embedded Computing*, vol. 12, 2013.

[6] D. Jenson and M. Riedel, "A deterministic approach to stochastic computation," *ICCAD*.

[7] W. Qian, "Digital yet deliberately random: Synthesizing logical computation on stochastic bit streams," Ph.D. dissertation, Ph.D., University of Minnesota, 2011.

[8] Y. Zhu, P. Suo, and K. Bazargan, "Binary stochastic implementation of digital logic," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2014, pp. 171–180.