Scaled Population Subtraction for Approximate Computing

Kunal Bharathi*, Jiang Hu† and Sunil P. Khatri‡
Department of ECE, Texas A&M University
College Station, TX, USA
Email: *kunal-bharathi@tamu.edu, †jianghu@tamu.edu, ‡sunilkhatri@tamu.edu

***Abstract*—In this paper we present Scaled Population Subtraction to fill a void in Scaled Population arithmetic. Scaled population (SP) arithmetic is a scheme that is inspired by stochastic computing (SC), a non-conventional approximate computing method that is well known for its simplicity, area efficiency and resilience to bit errors. SP arithmetic reduces the numerical errors compared to SC and also solves the serialization limitation of SC, since it is designed to have a O(1) gate delay. Previously, SP was limited to only addition and multiplication and did not have a way to perform subtraction. This paper introduces the basic SP subtraction idea, followed by a detailed study of several ways that the basic design can be improved to reduce the computational error. Our best SP design significantly improves the error compared to our basic SP subtraction idea (reducing it by 32.3%). We also study the trade-off between design complexity of the SP subtractor against output error. Also, our implementation of the SP subtractor exhibits an improved delay, power and area compared to fixed point realizations with the same size.**

***Index Terms*—Approximate Arithmetic, Stochastic Computing, Computer Arithmetic**

I. Introduction

Approximate computing is an approach that trades off accuracy against power, delay, area or reliability. Approximate computing is often used in applications where approximate results are tolerable. Examples of applications that are not always sensitive to the occasional small errors include signal processing [1], machine learning [2], scientific computing [3], and real-time systems [4]. For example, in a perceptron, we find a weighted sum and compare it with a threshold. Small errors do not introduce any problems as long as they don't change the value of the weighted sum with respect to the threshold. Voltage over-scaling [5], precision scaling [6], and inexact or faulty hardware [7] are some popular approximate computing techniques. Stochastic computing [8] is a non-conventional arithmetic scheme for area-efficient implementation of approximate computing.

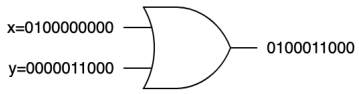


Fig. 1: Stochastic Computing - Addition $\frac{1}{10} + \frac{2}{10} = \frac{3}{10}$

In stochastic computing (SC), values are represented by bit vectors, and the arithmetic operations are processed by simple logic circuits, such as OR/AND gates for addition and multiplication, respectively. In Figure 1, we see how an OR gate can be used for addition. The numbers are represented as a vector of 1s and 0s. If the length of the vector is Π and the number of ones is π, the value the vector represents is π/Π. The two bit vectors are fed to each of the two inputs of the OR gate. The result is the bit vector at the output of the OR gate. Assuming that the input bit vector is random with a uniform distribution of 1s, the output bit vector will approximately have as many ones as the sum of the two inputs. One can see that such an adder design is much simpler and therefore has much lower circuit area compared to conventional adders. Moreover, the impact of a single-bit error in SC is much less than that of an MSB error in conventional binary numbers.

However, classical stochastic computing, which we will refer to as SC in this paper, has its own limitations. First, the accuracy of SC depends on the density and the randomness of the 1's in the binary bit vector. Second, the range of values that can be represented by the bit vectors is limited to $[0, 1]$. Third, SC has a runtime complexity of O(Π), where Π is the length of the bit vector. These weaknesses limit the applicability of SC in real world applications.

Scaled Population (SP) arithmetic [9] addresses the above limitations and achieves fast, approximate computing with a low area/power overhead and improved accuracy over SC. SP arithmetic uses some of the basic ideas of SC, but with following improvements:

- The inherent serialization of SC is avoided. A key design goal of SP arithmetic is that each operation is computed using O(1) gate delays (as opposed to clock cycles). SP never allows any operation that requires a serial traversal of the Π bits of any operand. Therefore, The SP arithmetic achieves a dramatic speedup over SC.
- The errors of SC are significantly reduced by providing a scaling (exponent) term in SP arithmetic. SP arithmetic is also more tolerant to bit errors than traditional fixed-point or floating-point computation.
- The range of numbers that can be represented by SP is much larger than what is possible in SC, due to the use of an exponent field in SP.

While SP [9] results in a significant improvement over SC, it does not provide a method for subtraction. For example, a matrix inner product with positive as well as negative elements cannot be realized using existing SP due to this limitation.

In this paper, we present an efficient technique for SP subtraction to fill this void in SP arithmetic. The key contributions of our research are:

- We present a novel idea of how subtraction can be performed using the SP arithmetic approach. We also present several improvements to the basic SP subtraction idea and present the best SP subtractor in terms of the accuracy.
- We conduct a thorough study of the relationship between design complexity and computational error.
- We further improve some of the computational units previously presented in [9]. For example, we present Skewed* addition, which is a more accurate version of the Skewed addition method presented in [9]. We also improve the density check units of [9]. These improvements are tailored to give us better subtraction accuracy and they are expected to lead to improved additional and multiplication as well.

This work is partially supported by the RTML program of the National Science Foundation, project CCF-1937396.

- We implement our approach in circuits of 45nm technology, and show improved area, delay and power compared to fixed-point arithmetic.

The rest of this paper is organized as follows: Section II provides background information on SC and SP. In Section III we will describe the basic SP subtraction idea, and in Section IV make several improvements to the basic idea. In Section V, we present results for an ASIC implementation of the best SP subtraction circuit from Section IV. Finally, in Section VI, we will present our conclusions.

II. Related Previous Work and Background

Stochastic Computing (SC) is a scheme for approximate computing on fractional numbers. In SC, a number x is represented by a $\Pi-$bit vector π, where $|\pi| \leq \Pi$ bits are randomly chosen to be 1, so that $x = \frac{|\pi|}{\Pi} \in [0, 1]$. Here $|\pi|$ is the number of bits of Π that are 1. SC has been shown to have a very low area cost for implementing common arithmetic operations like multiplication. Circuit implementations of SC can be found in [8] and [10]. The accuracy of the method depends on the location of 1s and 0s in the bit vectors. In [10], the absolute value of subtraction is realized by using MUX, but general subtraction is not handled. SC division is described in [11]. The three major drawbacks of SC include: (a) a performance bottleneck due to the serial transformation from a binary number to its corresponding SC bit vector and a serial computation of the result, (b) restricted operand range, since numbers are constrained to be between 0 and 1, and (c) a strong dependence of the error on operand values.

Scaled Population (SP) arithmetic is an approach designed to handle a wider range of numbers and perform mathematical operations in constant time [9], while reducing errors compared to SC. SP uses a scaling term (exponent) that enables the representation of numbers beyond the range [0, 1]. The SP representation of a number x is an M-bit tuple $x = \{\sigma, \pi\}$, where σ is a Σ-bit exponent term and π is a Π-bit population vector such that $M = \Sigma + \Pi$. The numerical value of a number using the SP representation is given by the following expression:

$$(|\pi|/\Pi) * 2^{\sigma - \Sigma_0}$$

In the above expression, $|\pi|$ is the number of ones in the SP representation of Π and Σ_0 is a constant. For example, the number 2.8 can be represented as $\{\sigma, \pi\} = \{110, 1011110101\}$, assuming $\Sigma_0 = 4$. The SP representation described above only handle non-negative numbers. However, augmenting SP to handle signed computation can be easily accomplished by adding a sign bit.

SP provides us an addition scheme called skewed addition, which is illustrated in Figure 2. We use 2 Π-bit masks m_x and m_y, where m_x has the left $\frac{\Pi}{2}$ bits set to 1, and m_y has the right $\frac{\Pi}{2}$ bits set to 1. The result will be $\pi_z = (\pi_x \& m_x)|(\pi_y \& m_y)$, with $\sigma_z = \sigma_x + 1 = \sigma_y + 1$. In *skewed* addition, it is guaranteed that no matter what the density[1] of the operands' population vectors is, the 1s in the two operands will never be aligned at the same bit position. This avoids the main source of error in SC addition.

SP also provides a method for approximate multiplication. The authors of [9] provide a set of supporting operations that they use in their SP addition and multiplication techniques. All operations take constant time. Details for these operations are presented in [9]. We utilize (and improve, as described) some of these operations:

[1]The density of a population vector π is defined as $|\pi|$.

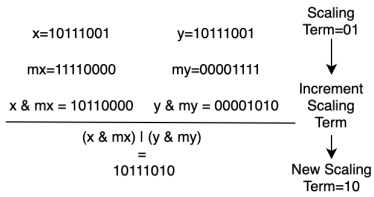


Fig. 2: SP-based *Skewed* Addition

- Generator: The generate operation converts a conventional binary number to the SP format.
- Shuffle Unit: Provides a 2-level LFSR based shuffler that can be used to permute the bits of the population vector. This helps achieve a more random distribution of 1s in a population vector. We improve the shuffler as described in the sequel.
- Density Check Unit: This is used to determine the number of ones in the population vector. We improve the Density Check Unit as described in the sequel.
- Scaling Unit: This is used to adjust the density of the population vector by modifying the exponent term.

In the next section we will explore our SP based subtraction scheme.

III. Overview of Scaled Population Subtraction

III.1. The Basic Subtraction Idea

Before describing the subtraction idea, we shall quickly describe a few key supporting facts.

III.1.1. Scaling Prior to Subtraction: Given two numbers, A and B, in SP representation, if the exponent terms (σ) are equal, we are free to manipulate the π terms in order to compute $A - B$. SP uses a *scaling unit* to adjust the σ and π terms for A and B without altering the value of the numbers. The scaling unit can double (or halve) the value of π while decrementing (or incrementing) σ by 1, to keep the values unchanged. We assume that the SP representation of A and B are scaled so as to have the same exponent term σ prior to subtraction. Henceforth, A or B will refer to the population vector π_A or π_B of A or B, respectively (unless otherwise specified).

III.1.2. Complement of a Number: Given the SP population vector of a number A, finding $1 - A$ is done by flipping all the bits of A. This can easily be realized using a set of inverters, where the number of inverters required is equal to Π. The amount of time it takes to compute $1 - A$ is a constant that depends on the delay of a single inverter. We use this to perform SP subtraction.

III.1.3. SP-based Subtraction: Given two SP numbers A and B, our goal is to compute $B - A$. We can do this as follows:

- First compute $S1 = (1 - B)$. This can be computed easily using inverters as previously described.
- Next, compute $S2 = A + S1$. This step can be computed using the addition scheme in SP arithmetic.
- Finally, compute $1 - S2$ to obtain the goal $B - A$. Effectively, the above steps have performed the computation $1 - (A + (1 - B))$, which is $B - A$. Thus SP subtraction can basically be achieved using only complementation and SP addition.

However, there are limitations to the basic SP subtraction approach which need to be addressed.

III.1.4. The Saturation Problem: Given SP numbers A and B, if we follow the procedure described above to compute $B - A$, we will run into a problem when $A > B$. As previously stated, signed numbers in SP are handled using a signed

bit, and SP computation cannot naturally result in a negative number. For example, assume that $A = 4/10$, represented by a population vector of size 10, with four bits set to 1. Let $B = 1/10$, represented by a population vector of size 10, with one bit set to 1. Once the subtraction starts, we cannot change the exponent term σ of A or B until the end. The first step in finding $B - A$ is to find $S1 = (1 - B) = 9/10$. Next, we find $S2 = A + S1 = 4/10 + 9/10 = 13/10$. This is clearly a problem, since 13/10 cannot be represented using just the population vector in SP, and the maximum value that the population vector can represent is 1. This is a situation that we call *saturation*. If the resulting value of subtraction is negative, then we are guaranteed that S2 will saturate and lead to an incorrect result.

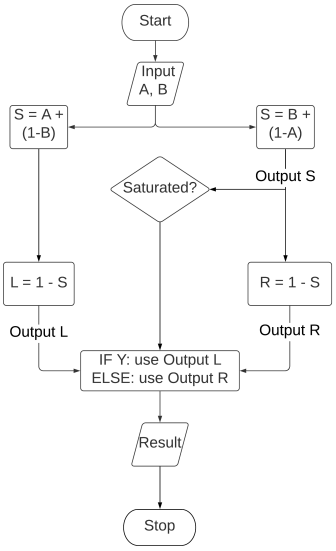


Fig. 3: Solving the Saturation Problem

However, there is a relatively easy fix for this problem, shown in Figure 3. We use 2 paths of computation, to compute both $B - A$ (the left branch of Figure 3) as well as $A - B$ (the right branch of Figure 3). If $B > A$, the value *Output S* computed on the right branch will saturate. We check if the *Output S* has saturated (consists of all 1s). If it has, we will use the subtraction result from the left branch (which is guaranteed to be non-negative) and subsequently set the sign bit of the result. If no saturation is detected, we use the output of the right branch and the sign bit is not set. Therefore, we can handle the saturation issue with additional logic without affecting the delay of our SP subtractor, assuming that the left and right branches execute in parallel. Alternately, serializing the computation of the branches yields a smaller area at the cost of delay.

The design in Figure 3 is the most basic SP subtractor idea. Next we will describe the different ways in which this idea is enhanced for improved error performance.

III.1.5. Summary of Enhancements and Evaluation Methodology: We explored enhancing the SP subtractor design along three independent axes:

- The first axis focused on the impact of different SP adders on the proposed SP subtraction unit. We compared three different types of adders: Regular, Skewed and Skewed*.
- The second axis involved enhancements to the population vector doubler. We will describe the need for a doubler in the sections that follow. Two doublers are explored: Ideal Doubler and Enhanced Density Check Doubler.
- The third axis explored input rescaling. Again, we will describe the need for input scaling later. Two SP subtractor designs are explored: Without-Rescaling and With-Rescaling.

The sections that follow explore the different options of each of the axis. The experimental data in each step will be used to determine the best design and eliminate the lesser ones from consideration in subsequent steps. In the next section, we will describe our experimental setup.

III.2. Evaluation Setup

The comparisons of the various enhancements over the basic SP subtraction framework are conducted using an experimental evaluation framework, which is described below.

- Different designs of the SP subtractor were tested for different sizes of population vectors Π. In the discussion that follows, we have used 3 different population vector sizes, $\Pi \in \{40, 100, 200\}$.
- The experiment was performed using 10000 pairs of values (A, B) randomly chosen such that $|A - B|$ was a fixed value. 10 values of $|A - B|$ were chosen, such that $(|A - B|) \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$.
- From these resulting experimental runs, we report the following metrics: The output value of every experiment is denoted as δ_i, where i ranges from 1 to N, N being the number of experiments.
  1) Actual result versus Exact result: For a given $T = |A-B|$ value, let $\mu = \Sigma\delta_i/N$. We plot μ versus the exact value for μ, which is T. This plot demonstrates if and when the actual results converge to the exact results.
  2) RMSE: For a given $T = |A - B|$ value, the Mean Square Error (MSE) is defined as $\Sigma(\delta_i - T)^2/N$. The RMSE is $\sqrt{MSE}$. RMSE gives us an idea of how far the actual result is from the exact result.
  3) Error%: We compute the Error% across all experiments for a given $T = |A - B|$. This metric gives context to the error of the actual result in relation to the exact result. We compute this as $Error\% = ((\mu - T)/T) * 100$.

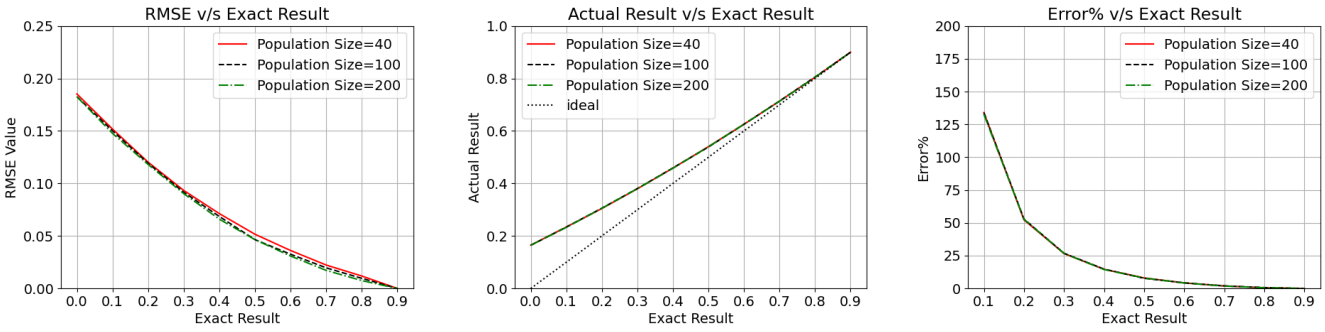
IV. Details of Proposed SP Subtraction

IV.1. Subtractor Design With Regular Addition

IV.1.1. Regular Addition: In this paper, Regular Addition refers to the SC-based method of performing addition using an OR gate. The only modification is that we perform this operation on the entire population vector at once to ensure a constant addition time (and not proportional to the size of the population vector Π). For example, consider 2 10-bit SP population vectors $A = 2/10 = 1000010000$ and $B = 2/10 = 0010000001$. To find $A + B$, we simply perform the bitwise OR of the 2 population vectors. In other words, $A + B \equiv A|B = 1010010001 = 4/10$. As expected, the errors in this method occur when either A or B are densely packed with 1s, and the 1s in the two operands start to align with each other. More specifically, the error is equal to $|\pi_A \& \pi_B|/\Pi$, where $|\cdot|$ indicates the number of 1s in a bit vector. This fact is what makes this technique approximate.

IV.1.2. SP Subtraction using Regular Addition: In Figure 3 we use the 2 Regular Adders, to compute S in the left and right branches. The errors in computation are due to the approximate nature of the Regular Adder result. Using the experimental setup described previously, we generated the RMSE (Figure 4a), Actual v/s Exact Result (Figure 4b) and the Error% (Figure 4c) plots, for SP subtraction using Regular Adders.

The average RMSE value across all experiments is 0.0723. An interesting trend is that the errors are larger when the



(a) RMSE (b) Actual v/s Exact Value (c) Error%

Fig. 4: SP Subtraction with Regular Addition

exact result is small. We will address this issue later in the paper. Next, we will look at SP subtraction using Skewed and Skewed* addition.

IV.2. SP Subtraction with Skewed and Skewed Adders*

In order to perform SP subtraction with a Skewed or Skewed* Adder, we need a population vector doubler. In this section we first discuss the need for this circuit, and compare the different variants of this circuit. Finally we use the best variant in the SP subtraction design with the Skewed and Skewed* Adders.

IV.2.1. Population Vector Doubler: To improve the error performance of SP subtraction with Regular Addition we will explore the Skewed Addition scheme (see Figure 2). Skewed Addition concatenates the left half of one vector with the right half of the other. This eliminates the errors due to 1s in the same position of both operands in Regular Addition. A consequence of using Skewed Addition is that the value of the result is half the exact value. One way to resolve this issue is by incrementing the exponent by 1 (effectively doubling the value of the Skewed Addition result). However, when we use Skewed Addition for SP subtraction, we cannot change this exponent term until the end of the subtraction procedure because a complementation operation follows the generation of *Output S* on both branches of Figure 3. The correct way to double the result of Skewed Addition (i.e. *Output S* in both branches of Figure 3) is to double the number of ones in the population vector *Output S* of the adder result (see Figure 9). The unit that performs this operation is a Population Vector Doubler. We henceforth refer to this unit as *doubler* for convenience. Next we will see different types of doublers and their effect on the error performance. The doubler is applied only for numbers no greater than 0.5.

IV.2.1.1. ***Ideal Doubler:*** The Ideal Doubler is a tool that helps us understand the error behavior of the SP subtractors better. It is not a design that can be realized in practice with a constant delay, but is used to compare the performance of our doubler candidates. When the input to the Ideal Doubler is less than 0.5, the output is twice the input. The output population vector consists of twice as many ones as the input. If the input is 0.5 or greater, the output population vector is a vector of all 1s (because that is the maximum value that can be represented).

IV.2.1.2. ***SP Doubler:*** SP uses a population vector doubler for some of its operations [9]. This is a doubler that can actually be realized with a constant gate delay. The bit operations used by this doubler are[2]:

[2]Here $Sh(*)$ refers to the shuffle operation, | refers to bit-wise OR and & refers to bit-wise AND.

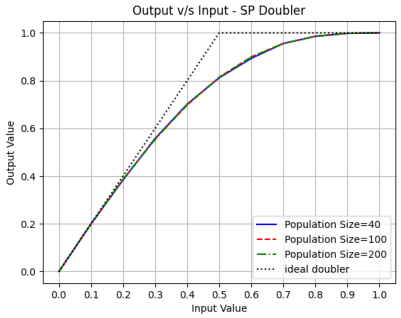


Fig. 5: The SP Doubler

- $X = Sh(Input)$
- $A = Input|X$
- $B = Input\&X$
- $Output = A|Sh(B)$

The performance of this doubler can be seen in Figure 5. The dotted line is the response of the Ideal Doubler that we desire. Next, we will look at the Density Check Doubler, an attempt to get a more ideal input-output curve.

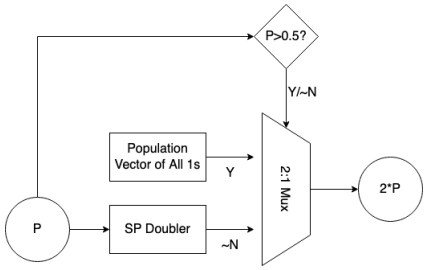


Fig. 6: The Density Check Doubler - Design

IV.2.1.3. ***Density Check Doubler:*** Figure 6 shows the design of the Density Check Doubler (DCD). The ideal doubler outputs a vector of all 1s when the input is greater than or equal to 0.5. SP provides us with a practical method to check if the value of the input is greater than or equal to 0.5. This unit is represented by the decision block in the figure (this block is the *density check*). Therefore, the density check doubler will select one of two outputs: A vector of all 1s when the input is greater than or equal to 0.5, otherwise it will select the output of the SP doubler. The input-output graph for the density check doubler is shown in Figure 7. As we can see from the graph, there is only marginal improvement compared to the SP doubler. Next, we will see an improved version of the Density Check Doubler.

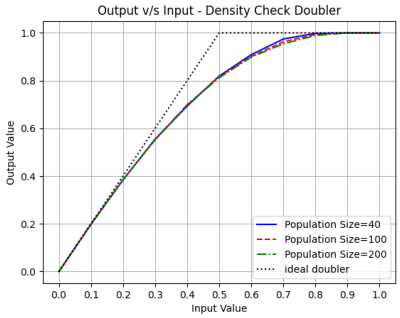


Fig. 7: The Density Check Doubler

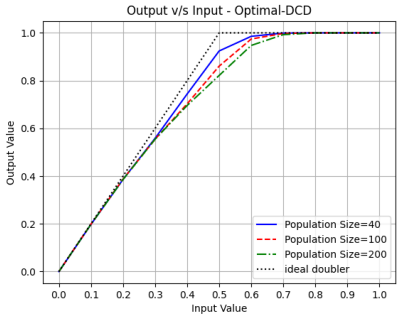


Fig. 8: The E-DCD

IV.2.1.4. ***Enhanced Density Check Doubler****:* Observe the curve of the DCD in Figure 7. For input values greater than 0.5, our experiments show that the DCD curve is below the expected ideal line. This suggests that the density check in the design is not working as expected. It fails to accurately detect when an input value is greater than 0.5. Another observation from our experiment is that the deviation of the DCD output from the ideal output is one-sided. The density check is more likely to conclude that a number is less than 0.5. We now try to correct this inaccuracy.

The realization of the density check is straightforward. Given an input population vector P, it first computes the intermediate population vector X.

$$X = P|(P <<< 1)|(P <<< 2) \quad (1)$$

Here "$<<< k$" represents a k-bit circular shift. If X is a vector of all 1s, then the input is deemed to have been greater than 0.5 [9]. Clearly our experiments (see Figure 7) suggest that the above bit operations are not populating all 1s in X when they should be. Therefore, we explored several different methods of generating X, inspired by the above equation. For each method, we re-generated the input-output graph of the DCD to determine the best approach. We found that the best way to generate X is to use the following bit operations, where Sh(A) represents shuffled version of population vector A (the shuffles breaks a long sequence of 1s or 0s (or "run") in the population vector, giving a more uniform distribution):

$$\begin{aligned} X \quad = \quad & P|Sh(P)|Sh(P <<< 1)|Sh(P <<< 2) \quad (2) \\ & |Sh(P <<< 3)|Sh(P <<< 4) \end{aligned}$$

We call the DCD that uses this new density check as Enhanced Density Check Doubler (or E-DCD). The input-output relation for the E-DCD is shown in Figure 8. The error in the output of this doubler is dependent on the probability of runs of 1s or 0s in the input population vector. Longer bit vectors will have a larger likelihood of containing a run of 1s or 0s. Therefore E-DCD performs better for a population size of 40 compared to a population size of 200. The performance of this E-DCD is better than any of the previous methods. Therefore, this is the design that we choose in practice. Next, we will present SP subtraction design using Skewed Addition (and the E-DCD).

IV.2.2. SP Subtraction using Skewed Addition: Having designed the doubler, we study the use of Skewed Addition for SP subtraction. In Figure 9 we use Skewed Addition to compute S in the left and right branches of the design in Figure 9. In the right branch, Skewed Addition will be used to first compute $S = B + (1 - A)$. Similarly, in the left branch we compute $S = A + (1 - B)$. Recall that, the result of Skewed Addition is half its true value. To correct this we need to double the result of Skewed Addition. The output of the doubler is indicated as *Output S* in the Figure 9. In this section we will look first at the results using the Ideal Doubler followed by results using the E-DCD.

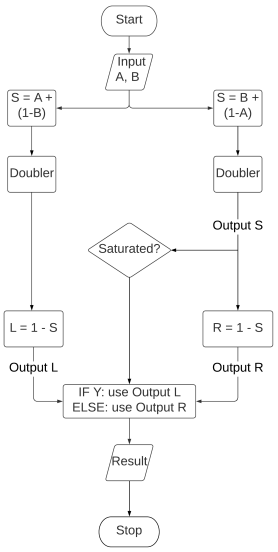
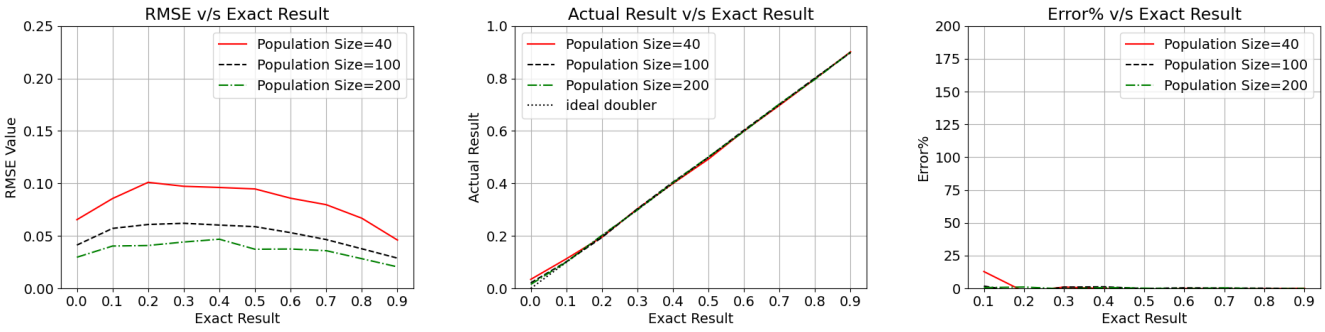


Fig. 9: SP Subtraction and Doublers

IV.2.2.1. ***SP Subtraction using Skewed Addition with Ideal Doubler****:* In this section we present the results for SP Subtraction using Skewed Adders with the Ideal Doubler. Using the experimental setup described previously, we generated the RMSE (Figure 10a), Actual v/s Exact Result (Figure 10b) and the Error% (Figure 10c) plots. Similar to Regular addition, the errors for small exact differences are larger than those for exact differences. However, since we are using an ideal doubler, the results look a lot more promising than those of the Regular Adder design (see Figures 4a, 4b and 4c).

In the RMSE plot, notice that the curves have a "hump". This behavior is related to the ideal doubler's output. When the input to the doubler is greater than 0.5, the output is always one, in other words, the doubler saturates. When the doubler saturates, the result of SP subtraction is always 0. Figure 11 shows how often the doubler saturated during our experiments. The doubler saturates most often when the exact result is 0. However this is does not lead to an error since the actual result is 0 anyway. For other small but non-zero exact results (eg: 0.1, 0.2), saturation leads to an increasing error, because the actual result is 0 while the exact result is non-zero. As the value of the exact result increases further saturation is no longer an issue and the error reduces. This error behavior manifests as the humps in the RMSE plot. Also, the larger the size of the population vector, the lesser the saturation frequency (see Figure 11). This is also reflected in the RMSE plot in Figure 10a, where the larger populations have a flatter curve and lower RMSE values. In a subsequent section we will see



(a) RMSE (b) Actual v/s Exact Value (c) Error%

Fig. 10: SP Subtraction with Skewed Addition and Ideal Doubler

how to eliminate this saturation problem altogether. Next, we present the results for SP subtraction using Skewed Addition and an E-DCD.

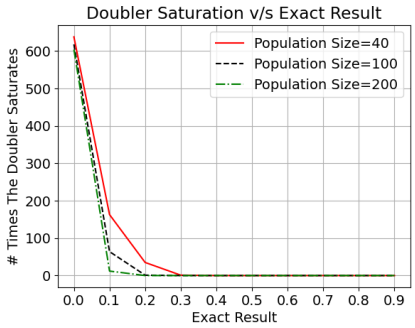


Fig. 11: Doubler Saturation v/s Exact Result

IV.2.2.2. ***SP Subtraction using Skewed Addition with E-DCD****:* In this section we report the results for SP Subtraction using Skewed Adder with the E-DCD. Using the experimental setup described previously, we generated the RMSE (Figure 12a), Actual v/s Exact Result (Figure 12b) and the Error% (Figure 12c) plots. From the plots we can see that the "humps" are still an issue in the RMSE plot. In addition, the E-DCD doubler introduces some additional errors due to its non-ideal nature. The average RMSE value across all experiments is 0.0848. This is 1.17x the RMSE error of SP subtraction with Regular Addition. However, note that the Actual v/s Exact Result and Error% (Figures 12b and 12c) are much better than those for Regular Addition (Figures 4b and 4c).

Next we introduce Skewed* Addition in SP Subtraction and compare it with the Regular and Skewed approaches.

IV.2.3. SP Subtraction using Skewed Addition:*

IV.2.3.1. ***Skewed* Addition****:* Skewed* Addition is modification to the Skewed Addition method, specifically designed to prevent doubler overflow during SP subtraction. The doubler will never saturate as long as the input is less than 0.5. In Skewed* Addition, to add X to Y, we take the entire population vector of X and concatenate it with the entire population vector of Y. The length of the result is twice as long as any one of its inputs. An example is shown in Figure 13. Since the result is twice as long, the result of Skewed* Addition is half the true sum, and hence requires a doubler just like Skewed Addition. Increasing the number of bits in the result also ensures that the addition step in Skewed* Addition introduces no errors due to 1s appearing in the same bit positions of the two inputs.

Given two SP population vectors A and B, where $B > A$, we have the following relation:$A-B<0$, $A+(1-B)<1$ If we use Skewed* Addition for this step, we have $A+(1-B)<0.5$, because the result will be half the true value. This forms the input to the doubler, which will ideally never saturate since its input never exceeds 0.5.

For subsequent SP operations on the result of SP subtraction with Skewed* Addition, the vector will be decimated back to Π bits.

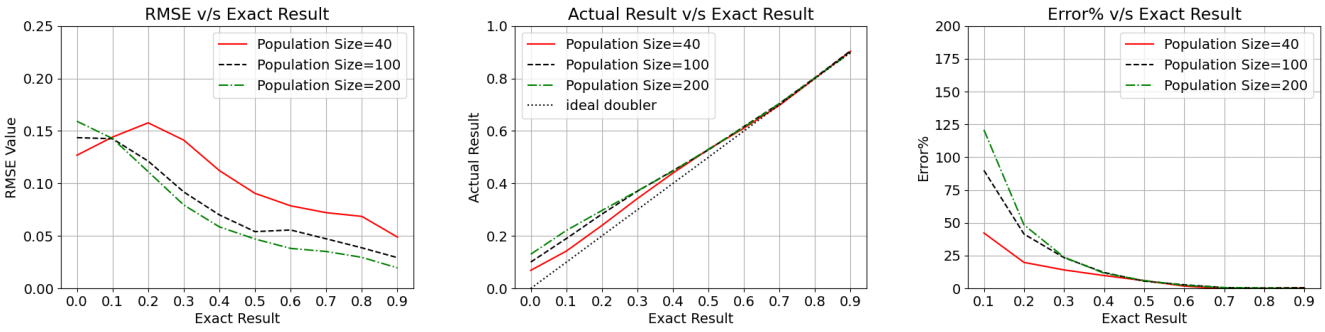
IV.2.3.2. ***SP Subtraction using Skewed* Addition with E-DCD****:* We next show the results for Skewed* Adder with the E-DCD. Using the experimental setup described previously, we generated the RMSE (Figure 14a), Actual v/s Exact Result (Figure 14b) and the Error% (Figure 14c) plots. The average RMSE across all experiments is 0.0576. This is 0.67x the RMSE of the SP Subtraction with Skewed Addition with E-DCD, and 0.79x the RMSE of SP Subtraction with Regular Addition. Additionally, if we compare the plots for SP Subtraction with Skewed* Addition and E-DCD (Figures 14a, 14c, 14b) with the plots for SP Subtraction using Skewed Addition and E-DCD (Figures 12a, 12c, 12b), we see that the use of Skewed* Addition has solved the "humps" in the RMSE graphs. Similarly, comparing Figures 14a, 14b and 14c with the plots for SP Subtractor with Regular Addition (Figures 4a, 4b and 4c) we observe a better RMSE as well as Error% for SP Subtraction with Skewed* Addition and E-DCD, and so we conclude that it is the preferred method among all the methods we have developed. All schemes exhibit the trend where the errors are larger for smaller exact differences. We will try and resolve this issue in the next section.

IV.3. Input Rescaling

In the previous sections we have observed that smaller exact differences yield larger errors. Figure 15 shows one approach to curb this trend. The basic idea is that if we multiply or *rescale* the inputs prior to subtraction, then the exact difference is also rescaled by the same amount. A larger exact difference would mean smaller RMSE errors. SP subtraction with rescaling works as follows:

- Given 2 SP numbers A and B, perform SP subtraction using Skewed* Addition and E-DCD.
- If the result of the subtraction is less than some threshold D, double the values of A and B and repeat SP subtraction. Track the number of rescale operations, call it E.
- Repeat this process until the result exceeds the threshold, D or one of the inputs to the SP subtractor exceed 0.5.
- At the end of the subtraction process, subtract E from the exponent σ.

The rescaling, or doubling of A and B can be easily achieved using the population vector doublers that we have previously



(a) RMSE (b) Actual v/s Exact Value (c) Error%

Fig. 12: SP Subtraction with Skewed Addition and E-DCD Doubler

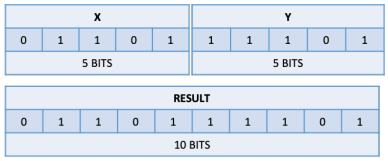


Fig. 13: Skewed* Addition

discussed. The optimal threshold D was determined experimentally to be 0.5.

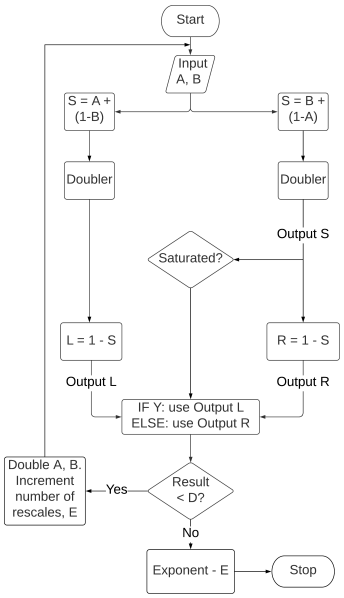


Fig. 15: SP Subtraction with Input-Rescaling

IV.3.1. SP subtraction with Skewed Addition, E-DCD and Input Rescaling:* In this section we explore SP Subtraction with Skewed* Addition with E-DCD and Input Rescaling. The optimal threshold for rescaling was previously determined to be 0.5. Using the experimental setup described previously, we generated the RMSE (Figure 16a), Actual v/s Exact Result (Figure 16b) and the Error% (Figure 16c) plots. The average RMSE across all experiments is 0.0489. This is 0.85x the RMSE of SP Subtraction using Skewed* Addition with E-DCD but no Input Rescaling. In addition, the Actual v/s Exact Result and the Error% plots exhibit better results as well. Therefore, we can conclude that SP subtraction using Skewed* Addition with E-DCD and Input Rescaling is the best performing design.

We have examined 4 practically realizable SP subtractor designs:

- SP Subtractor with Regular Addition (method RA).
- SP Subtractor with Skewed Addition using E-DCD (method SK).
- SP Subtractor with Skewed* Addition using E-DCD (method SK*).
- SP Subtractor with Skewed* Addition using E-DCD and input rescaling (method SK*I).

The bar chart in Figure 17 shows a comparison of the RMSE values for each of methods mentioned above. As we can see, the best method (smaller RMSE) is method SK*I, which reduces RMSE by 32.3% compared to RA. Although SK has worse errors than RA, it is an important intermediate step for developing SK* and SK*I.

IV.4. Error Comparison Summary

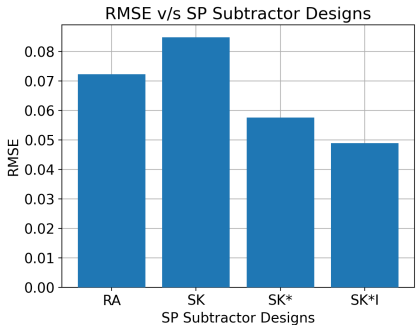
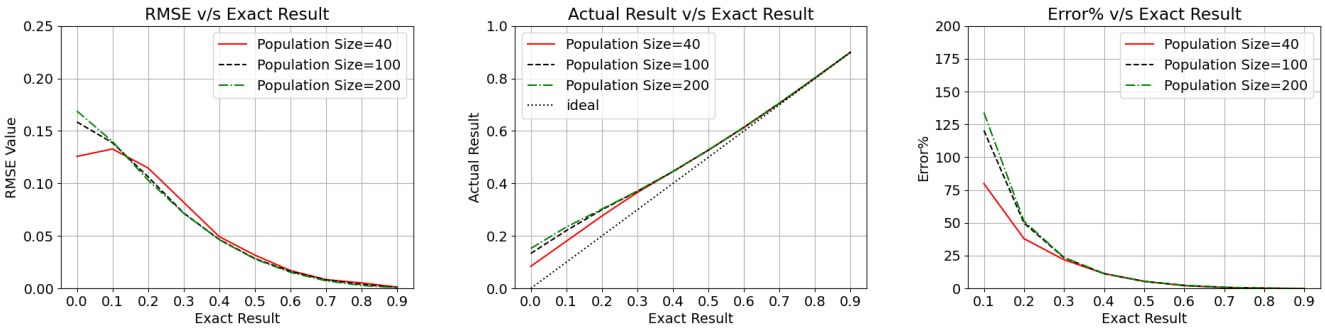


Fig. 17: RMSE v/s Designs

V. Circuit Implementation

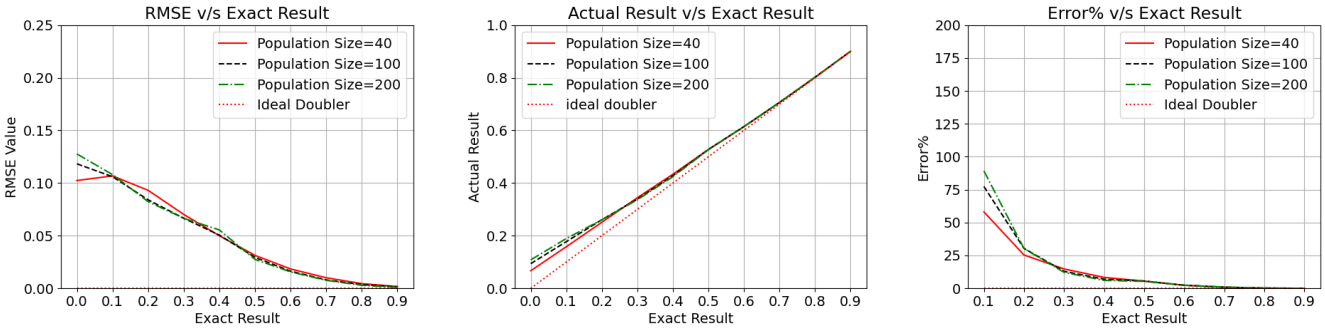
Our most accurate method SK*I (SP subtraction using Skewed* Addition with the E-DCD and Input Rescaling) was realized using synopsys DC in 45nm Nangate technology [12]. The circuit area, power and performance are compared with conventional fixed point subtractors for binary numbers. SC does not have a complete solution to subtraction and therefore cannot be compared. The conventional fixed point subtractor implemented in this experiment uses a Carry Look Ahead (CLA) adder.

In the SK*I design according to Figure 15, there are two parallel computational paths which compute both $A - B$ and $B - A$ at the same time. We implement only one of them in the circuit, thereby serializing the design. The $A - B$ path is implemented, and if during the process we detect that $A < B$, we will repeat the process with the operands swapped. The delay estimate assumes the situation where the circuit is executed thrice, which accounts for one Input Rescaling operation. SP subtraction is designed to be part of an SP arithmetic module [9], where units like the doubler already



(a) RMSE (b) Actual v/s Exact Value (c) Error%

Fig. 14: SP Subtraction with Skewed* Addition and E-DCD Doubler



(a) RMSE (b) Actual v/s Exact Value (c) Error%

Fig. 16: SP Subtraction with Skewed* Addition, E-DCD Doubler and Input Rescaling

exist. Therefore, the area of these units are not counted again in our subtractor area estimation.

TABLE I: Circuit characteristics of different subtractor implementations.

Design	Bitwidth	Area(μm^2)	Delay(ns)	Power(μW)
Fixed	32	425	0.82	820
Fixed	64	782	0.94	1563
SK*I	32	183	0.47	261
SK*I	64	358	0.47	503
SK*I	128	713	0.47	995

The results are shown in Table I. We note that the delay for our proposed SP subtractor (SK*I) is always less than the delay of the fixed point subtractor. We can confirm that as was theoretically stated, the delay through the SP subtractor is independent of the size of the operands. As the width of the SP subtractor increases, the area grows because we have to realize a larger number of gates. The 32 and 64-bit wide SP subtractors also have a better area footprint and power performance when compared to the fixed point designs. However, the precision of the 32 and 64-bit wide SP subtractors is lower than that of the fixed point designs of the same bitwidth.

VI. Conclusion

In this paper we presented a technique to perform SP subtraction, an arithmetic operation that was previously missing in SP. Experimental results show that our best subtractor has 0.67x the RMSE error compared to the basic SP subtractor design. Our SP subtractor design can be simplified by eliminating rescaling, or using a simpler population vector doubler, at the cost of computational error. We also demonstrate that our best SP subtractor design has reduced area, delay and power as compared with a traditional fixed-point binary subtractor of the same width, with improved tolerance to bit errors in the operands. Our design allows SP arithmetic to now be used in applications that require subtraction operations.

References

[1] R. St Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger, "General-purpose code acceleration with limited-precision analog computation," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 505–516, 2014.

[2] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, "Rumba: An online quality management system for approximate computing," in *ISCA*, 2015, pp. 554–566.

[3] B. Grigorian, N. Farahpour, and G. Reinman, "Brainiac: Bringing reliable accuracy into neurally-implemented approximate computing," in *HPCA*, 2015, pp. 615–626.

[4] Y. Wang, H. Li, and X. Li, "Real-time meets approximate computing: An elastic CNN inference accelerator with adaptive trade-off between QoS and QoR," in *DAC*, 2017, pp. 1–6.

[5] J. S. Vetter and S. Mittal, "Opportunities for nonvolatile memory systems in extreme-scale high-performance computing," *Computing in Science & Engineering*, vol. 17, no. 2, pp. 73–82, 2015.

[6] G. Keramidas, C. Kokkala, and I. Stamoulis, "Clumsy value cache: An approximate memoization technique for mobile GPU fragment shaders," in *Workshop on Approximate Computing*, 2015.

[7] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *DAC*, 2012, pp. 820–825.

[8] B. R. Gaines, "Stochastic computing," in *Proceedings of the Joint Computer Conference*, 1967, pp. 149–156.

[9] H. Zhou, S. P. Khatri, J. Hu, and F. Liu, "Scaled population arithmetic for efficient stochastic computing," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, pp. 611–616.

[10] B. R. Gaines, "Stochastic computing systems," in *Advances in Information Systems Science*. Springer, 1969, pp. 37–172.

[11] T.-H. Chen and J. P. Hayes, "Design of division circuits for stochastic computing," in *ISVLSI*, 2016, pp. 116–121.

[12] Wdavis, *NCSU EDA Wiki*, 29 May 2014 (accessed June 11, 2020). [Online]. Available: https://www.eda.ncsu.edu/wiki/NCSU_EDA_Wiki