Scaled Population Division for Approximate Computing

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

***Abstract*—In this paper we present an approximate division scheme for Scaled Population (SP) arithmetic, a technique that improves on the limitations of stochastic computing (SC). SP arithmetic circuits are designed (a) to perform all operations with a constant delay, and (b) they use scaling operations to help reduce errors compared to SC circuits. As part of this work, we also present a method to correlate two SP numbers with a constant delay. We compare our SP divider with SC dividers, as well as fixed-point dividers (in terms of area, power and delay). Our 512-bit SP divider has a delay (power) that is $0.08\times$ ($0.06\times$) that of the equivalent fixed-point binary divider. Compared to a equivalent SC divider, our power-delay-product is $13\times$ better.**

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

I. Introduction

Applications in signal processing [1], machine learning [2] and real-time systems [3] require fast arithmetic circuits and can tolerate small errors. Approximate computing techniques take advantage of this tolerance to errors, trading computational accuracy for more power-efficient operations. A popular approximate arithmetic scheme is Stochastic Computing (SC) [4] [5]. SC has seen uses in high-throughput decoding of LDPC codes [6], low area/power Deep Neural Networks [7].

Stochastic computing (SC) uses simple logic circuits for arithmetic operations. However, SC has several limitations. SC has a runtime complexity of O(Π) (Π is the length of the SC *bit stream*). SC accuracy also depends on the number and distribution of '1's and '0's in the input bit streams. The range of values that the bit streams are able to represent is limited to [0, 1]. The authors of [8] address these limitations of SC in a scheme called Scaled Population (SP) arithmetic. SP is a low area/power overhead scheme that has a constant delay (in terms of gates), wider range of values and smaller errors than SC. SP addition and multiplication schemes were presented in [8], and an SP subtraction scheme was presented in [9]. However, no technique for SP division has been reported to date. The key contributions of our work are:

- We exploit the correlation of the bits in the SP-based input operands to perform division with logarithmic delay. As part of this work, we also present a novel approach to correlate two SP bit vectors with a constant delay (using the notion of "strong correlation" which means maximum overlap of the '1' bits of the two SP numbers).
- We also compare our SP division method to fixed-point dividers and SC dividers with respect to the 4 metrics. Our 512-bit SP divider has a delay (power) that is $0.08\times$ ($0.06\times$) times that of the equivalent (same numerical precision) fixed-point binary divider, and compared to a equivalent SC divider, our power-delay-product is $13\times$ better.
- We evaluate the error performance of approximate dividers using a novel metric called ENOB (Estimated Number of Bits)

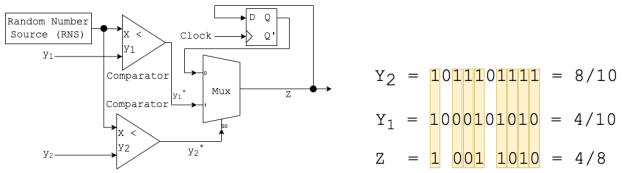
This work is partially supported by the RTML program of the National Science Foundation, project CCF-1937396.

II. Background and Previous Work

SP supports a wider range of numbers and mathematical operations are designed to have a constant delay. A scaling term (exponent) is used by SP to represent a wider ranger of numbers. SP numbers are expressed as a 2-term tuple: $\{\sigma, \pi\}$, where σ is a Σ-bit term (σ is a binary number) and π is a Π-bit vector [8]. Next, we present an overview of existing division techniques in SC.

A. Division in Stochastic Computing

The authors of [10] proposed the first scheme for dividing two SC numbers. Their method exploited the logic of a JK Flip-Flop to perform division. The accuracy of the method is dependant on the operands (better results are obtained when the dividend is significantly smaller than the divisor). The design was improved by introducing a feedback element, resulting in more accurate results. However, they still require significantly long bit streams, resulting in a large delay. The authors of [11] present CORDIV, a scheme that relies on the correlation of bits in the inputs to perform division. Compared to previous approaches, CORDIV achieves better accuracy but the delay is still proportional to the length of the operand bit streams. DFSM-DIV [12] and CBDIV [13] are improvements to the CORDIV technique. However, in addition to having a delay that is proportional to the length of the input bit streams, another drawback of these schemes is that CORDIV, DFSM-DIV and CBDIV require binary inputs, and the result for DFSM-DIV and CBDIV are also in binary. These dividers, therefore, cannot be directly cascaded with other SC operators. Next, we will look at CORDIV in more detail, and subsequently present our idea.



(a) (b)

Fig. 1: CORDIV [11]

1) CORDIV: CORDIV [11] relies on *strong correlation* between two SC bit streams (the divisor (Y_2) and dividend (Y_1)) to perform division. Since we are dealing with SC numbers, a number is simply a bit stream. As shown in Figure 1a, the CORDIV unit reads one bit every clock cycle. Let $P(y_i)$ be the probability of observing a '1' at a given clock cycle in the SC bit stream Y_i. $P(y_1, y_2)$ is the joint probability of observing a '1' in both bit streams in the same clock cycle. From probability theory [14] we know that: $P(y_1|y_2) = \frac{P(y_1,y_2)}{P(y_2)}$. When $Y_1 < Y_2$ and both SC numbers are strongly correlated, the probability relation simplifies to $P(y_1|y_2) = \frac{P(y_1)}{P(y_2)}$. CORDIV achieves division by constructing a new SC bit stream Z, whose value is equal to $P(y_1|y_2)$, and therefore also equal to $\frac{P(y_1)}{P(y_2)}$. In Figure 1b illustrates the construction of output Z. From Y_1, we select only those bits whose corresponding bit in Y_2 is a '1' (the shaded columns represent Z, whose value is $\frac{y_1}{y_2}$). There are three major drawbacks that prevent CORDIV from being used in SP arithmetic:

1) The construction of Z violates the constant delay requirement of SP arithmetic.
2) Strong correlation negatively affects the error of addition and multiplication for SC or SP.
3) The output is of variable size (as shown in Figure 1a). The size of the output Z is determined by the number of '1's in Y_2.

In the next section, we present our SP design that builds on CORDIV, and overcomes these drawbacks.

III. An Overview of SP Division

In this section we will describe our SP divider. Figure 2 gives an overview of the SP division technique. Our method is divided into 4 *stages*. Each stage is designed to eliminate the drawbacks of CORDIV:

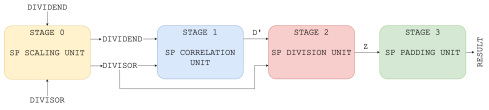


Fig. 2: SP Division

1) **Stage 0: The SP Scaling Unit** is used to modify the exponent of the input SP operands to ensure that the mantissa of the dividend $\pi_{dividend} \leq \pi_{divisor}$, the mantissa of the divisor.
2) **Stage 1: The SP Correlation Unit** is used to take the dividend and divisor SP bit streams as input, and output D', a SP bit vector which has the same value as the original dividend, but is now strongly correlated with the divisor.
3) **Stage 2: The SP Division Unit** takes as input D' and the divisor, and outputs Z, the variable length result.
4) **Stage 3: SP Padding Unit**. SP arithmetic requires the result to be of a fixed length. The SP Padding Unit takes the variable length Z and outputs a division result that has the same number of bits as the divisor and dividend.

In the next sections, we will study the design of each of the stages in more detail.

A. Stage 0: SP Scaling Unit

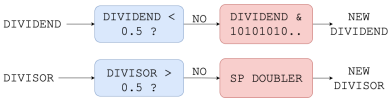


Fig. 3: Stage 0: SP Scaling Unit

CORDIV requires $\pi_{dividend} \leq \pi_{divisor}$. However, because SP has an exponent term, it is possible to have $dividend \leq divisor$, but $\pi_{dividend} \geq \pi_{divisor}$. We use stage 0 to adjust the π and σ of the 2 operands to ensure $\pi_{dividend} \leq \pi_{divisor}$. The logic of Stage 0 is described in Figure 3. SP arithmetic [8] has units that can compare numbers to fixed constants. We use these to implement Stage 0. If $\pi_{dividend} \geq 0.5$, we halve the value of the mantissa. This is easily achieved in constant time by using bit-wise logical AND as shown in Figure 3. To ensure that the value of the dividend is unchanged, the exponent is incremented by one if $dividend \geq 0.5$. Next, we want to modify the $\pi_{divisor}$ so that $\pi_{divisor} \geq 0.5$. To achieve this, we use SP population vector doublers from [9]. This unit doubles the numbers of ones in the π of its input. Therefore, we subtract one from the exponent, to keep the value of the divisor unchanged. The divisor needs to be doubled until $\pi_{divisor} \geq 0.5$. In the worst case, this process can take $(\log_2(\Pi) - 1)$ cycles. While all other stages are constant delay, this scaling of the divisor results in a logarithmic delay for our SP divider.

B. Stage 1: SP Correlation Unit

Given two SP numbers Y_1 and Y_2, where $Y_1 < Y_2$, we need to transform Y_1 such that its '1's are maximally overlapped (correlated) with the '1's in Y_2. The SP Correlation Unit is described in Algorithm 1. We next describe its steps.

1) Step 1 ($r = Y_1 \wedge Y_2$): If we perform a bit-wise logical AND operation between the SP numbers Y_1 and Y_2, then the result r will be strongly correlated to Y_2. However, numerically, $Y_1 \geq r$. The further r is from Y_1, the greater will be the error in our computation. For example, if $Y_1 = 11000$ and $Y_2 = 10011$, then $r = 10000$.

2) Step 2 (Fixing the Error): In the above example, the reason $r \leq Y_1$ is because Y_1 can have '1's in positions that Y_2 does not. We need to "collect" these ones and "insert" them back into to r. We achieve this with the help of the bit-wise difference operator ('\') or bit-wise AND-NOT operation ($a \setminus b \Rightarrow a \wedge (\neg b)$). Let $t_1 = Y_1 \setminus r$. Continuing our example,

$t_1 = 01000$. t_1 represents the '1's in Y_1 that were "left behind" in step 1. Next, we shuffle (using hardwired connections to randomly transpose the bits) the bits of t_1 to obtain t_1'. We want to shuffle the bits in order to try and align the "left behind" '1's with '1's in Y_2. We obtain t_2 using: $t_2 = Y_2 \wedge t_1'$. The '1's in t_2 will always overlap with the '1's in Y_2. We now insert the '1's of t_2 back to r using the operation: $r = r \vee t_2$. We repeat Step 2 a certain number of times ($Rounds$) until $r \approx Y_1$. When $Rounds = 1$, we perform only Step 1. For $Rounds = 2$, we perform Step 1 once, and Step 2 once. The loop represents Step 2. In the circuit implementation, we unroll this loop to improve the circuit delay.

Algorithm 1 SP Correlation Unit

Require: $Y_2 \geq Y_1$, $\{Y_1, Y_2\} \in SP\ Numbers$
  $r \leftarrow bit\ vector\ of\ all\ 0s$ ▷ Output, Initially 0
  $Rounds \leftarrow From\ Circuit\ Designer$
  $r \leftarrow Y_1 \wedge Y_2$ ▷ Step 1
  for $i \in [2....Rounds]$ **do** ▷ Step 2
    $t_1 = Y_1 \setminus r$
    $t_1' = shuffle(t_1)$
    $t_2 = Y_2 \wedge t_1'$
    $r = r \vee t_2$
  end for

C. Stage 2: SP Division Unit

Stage 1 provides us two operands that are now strongly correlated. In Stage 2 we perform our CORDIV-based division but with a key improvement over [11]: we will construct the same output, but with a constant delay. The logic for Stage 2 is depicted in Figure 4. In Figure 4 the SP bit vectors are of length 10 ($\Pi = 10$). The divisor (Y_2) has a numerical value of $\frac{8}{10}$, while the dividend (Y_1) has a numerical value of $\frac{4}{10}$. Since the numbers are strongly correlated, we can find the result of division by selecting the bits Y_1^i, whose corresponding i^{th} bit in Y_2, Y_2^i is a '1'. We accomplish this using a set of Π MUXes. As shown in the figure, the select line for the i^{th} MUX is the Y_2^i bit (indicted by the blue connections), and the input selected when the select line is high is the Y_1^i (indicated by the red connections). The connection of the other input of the MUX will be discussed in the next section. The output of each MUX forms the Z^i bits. In Figure 4, the 8 bits of Z, corresponding to the 8 '1' bits in Y_2, are assigned the bit value from Y_1 via the MUXes. If we consider just these 8 bits, then the result of division is as expected ($\frac{4/10}{8/10} = \frac{4}{8} = \frac{1}{2}$). However, in SP, we cannot have variable length outputs, and the remaining bits need to be "padded" or filled in, with minimal affect to the number represented by Z. We present our scheme of doing this in the next section.

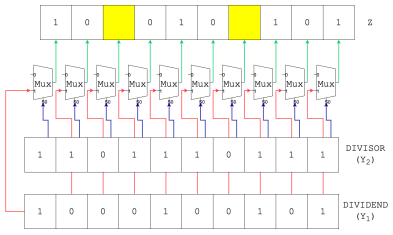


Fig. 4: Stage 2 - Unrolled CORDIV

D. Stage 3: SP Padding Unit

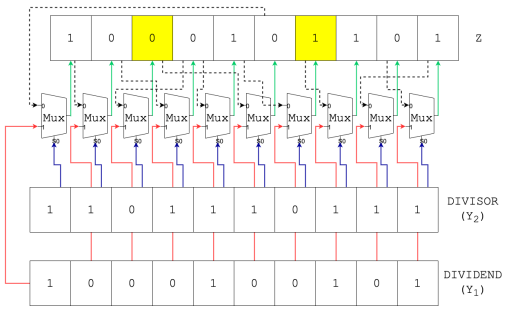


Fig. 5: Stage 2 + Stage 3 (Padding Circuit)

The easiest way to pad Z, is to fill the yellow entries of Figure 4 randomly with '1' bits or '0' bits. But, doing so will alter the value represented by the SP number Z. We want to fill the yellow entries of Figure 4 with a bit that has a probability $\frac{Y_1}{Y_2}$ of being a 1. If we sample a bit from an SP bit vector at random, the probability of sampling a '1' bit is equal to the value represented by the SP number. We use this property to pad the missing (yellow) bits of Z in Figure 4. In Figure 5, the previously unused MUX connections are randomly hard-wired using the dashed black lines. Now, if a bit is missing (yellow) from Z (the corresponding bit in Y_2 is a '0'), then this circuit *randomly* (with a probability $\approx \frac{Y_1}{Y_2}$ of being 1) selects a bit from a different part of Z. This way, we pad the result of stage 2, without affecting the value of the final padded result appreciably. In Figure 5, the "padded" bits are indicated using a yellow color. Therefore, the padded result Z (final result) now has a value $\frac{5}{10} = \frac{1}{2}$. We construct the hardwired feedback connections in a manner that guarantees the absence of cyclic dependencies whenever at most $\frac{\Pi}{2}$ divisor bits are '0'. Since the logic of Stage 0 ensures that $\pi_{divisor} \geq 0.5$, we avoid any cyclical dependencies in Stage 3. In the next sections we will study the performance of our divider using different metrics obtained via experimental simulations.

IV. Simulation Experiments and Results

A. Experimental Methodology

In Section IV-A1 we discuss our method of computing error, while Section IV-A2 describes our circuit implementation approach.

1) ENOB: The Effective-Number-of-Bits (ENOB) is used in analog circuits like ADCs or DACs [15], and represents the precision of any analog circuit scheme in terms of binary bits. We adopt this metric in the context of SP. Every Π-bit SP number has a precision that is at most $\log_2 \Pi$ binary bits. Therefore, the maximum ENOB for a Π bit SP number is $\log_2 \Pi$. The approximate nature of SP computations introduce errors in the output. If the RMSE (Root-Mean-Square Error) of the SP arithmetic operation is ϵ, then the number of SP bits affected by the error is $\epsilon * \Pi$. Therefore, the effective number of accurate SP bits available after the SP arithmetic operation is $\Gamma = \Pi - \epsilon * \Pi$. The new ENOB is therefore: $\log_2 \Gamma = \log_2(\Pi - \epsilon * \Pi)$. In the discussion that follows, we

use the ENOB as one of the metrics to quantify the errors due to the approximate nature of our divider. The total number of *whole* bits of accuracy after accounting for approximation errors is $ENOB' = \lfloor (ENOB) \rfloor$.

2) Hardware Implementation: In order to find the circuit area, power, delay and energy of our SP divider design, we expressed the logic of our divider in Verilog [16]. The logic for all 4 stages consists of only combinational circuits (no flip-flops/latches). The design was then synthesized using Synopsys DC using the ASAP7 [17] technology library. Prior to synthesis, we allow Synopsys DC to optimize the logic of the top-module if it can. We used "compile_ultra" since we have tight timing constraints (we want fast circuits, since speed is a key driving philosophy of SP arithmetic). The area reported by DC is the cell area. The ASAP7 technology library has a well known 4x scale factor [18] and we apply this correction for all designs before reporting all our results. In order to compute the power numbers, we first create a switching activity file using Synopsys VCS for every design using 10000 test vectors. The activity file is then used by Synopsys DC to report the final power numbers. The delay is obtained from the timing report generated by Synopsys DC. The power report provides us with the total power consumption data.

B. Experimental Results

1) SP Divider Error and Hardware Implementation: In this section we will study the error performance of the entire SP divider and the hardware implementation of the SP divider logic.

Error Simulations: In order to study the error of the entire SP divider, we implemented the logic in python, and performed 100,000 simulations. For each simulation we pick a *dividend* randomly from a uniform distribution with range (0, 1) and a *divisor* randomly from a uniform distribution with range (dividend, 1).

ENOB		ROUNDS					Ideal ENOB
		1	2	3	4	5	
Π	**8**	2.52	2.62	2.68	2.71	2.71	3
	16	3.59	3.69	3.75	3.79	3.80	4
	32	4.63	4.72	4.79	4.83	4.84	5
	64	5.65	5.74	5.81	5.85	5.86	6
	128	6.66	6.75	6.81	6.86	6.88	7
	256	7.66	7.75	7.82	7.86	7.88	8
	512	8.66	8.75	8.82	8.87	8.89	9

TABLE I: ENOB for SP Divider

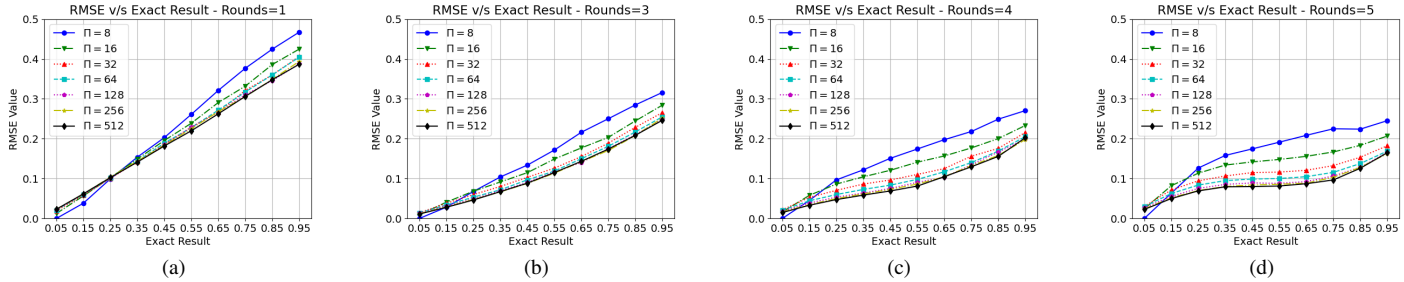
We perform these simulations for $\Pi = \{8, 16, 32, 64, 128, 256, 512\}$. For each Π, we simulated division for $Rounds \in \{1, 2, 3, 4, 5\}$. We omit the plots for $Rounds = 2$ for brevity.

Figure 6 plots the RMSE v/s Exact result for different Πs and $Rounds$ values. The Exact Result here refers to the result assuming 0 errors, in other words, it is the division result calculated using an exact (not approximate) method. We use IEEE-754 floating point arithmetic for our exact results. In each of the figures, for a given population size, we plot 9 data points. Consider the last black diamond ($\Pi = 512$) in Figure 6a. This data point indicates that in our experiments, for the exact values in the range (0.9, 1], we obtained an average RMSE of 0.38. The behavior of these figures is as expected. The error performance improves with more $Rounds$ (due to more accurate correlation) and also improves with Π (due to more mantissa bits). The only outliers are the plots of $\Pi = 8$, which we attribute to noise due to a very short mantissa. Observe that for $\Pi = 8$ the error is 0 when the exact result is close to 0 (which is much better than the error of larger Π). This is because the smallest value that can be represented by an SP number with $\Pi = 8$ is 0.125. Therefore, any division that is expected to result in a value less than 0.125 will always result in 0. For the same reason, any error in the range [0, 0.125), results in 0 error. An important point to note is that we are reporting average RMSE values, and approximate computations will always have certain inputs for which the errors will be significantly greater. However, in applications like ML, with millions of computations, the average error is more important than a few outliers.

For each Π and $Rounds$ value, we use the average RMSE across the entire simulation to compute the ENOB values, using the formulas discussed in Section IV-A1. Table I summarizes the ENOB for all our simulations. We note that a reasonable value of $Rounds$ beyond which the ENOB doesn't appreciably increase is 2 or 3.

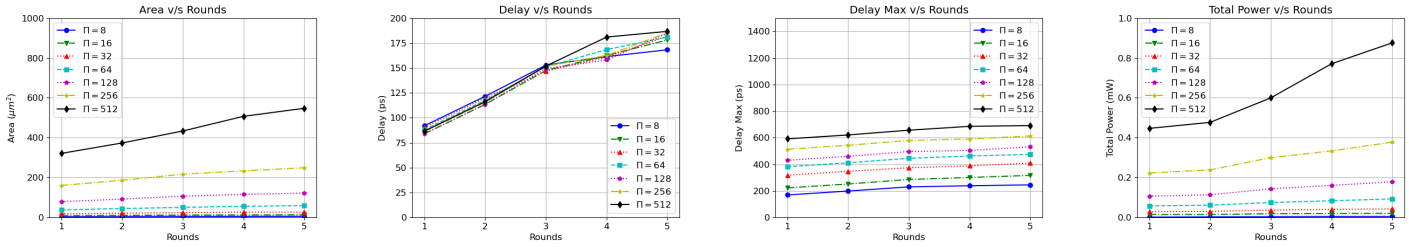
Circuit Implementation: In the previous section we described the error performance for different configurations of the SP divider. Now we will study the area, power and delay of our SP divider's circuit implementation. Figure 7a plots the area performance of our SP divider. The graph behaves as expected, where the designs with larger Π and/or larger $Rounds$ have a larger area footprint. The user can use the data of Figures 6 and 7 to choose a design based on the area-error trade-off. The circuits with larger area have a smaller error, as indicated by the corresponding ENOB value ($\uparrow$ ENOB $\Rightarrow$ $\downarrow$ Error) in Table I. The delay performance of the circuit is shown in Figure 7b. Our SP divider has a logarithmic delay due to Stage 0. In Figure 7b, we report the delay assuming that the scaling of the divisor requires a single iteration. With this assumption, the delay numbers should be independent of Π, and only depend on the $Rounds$. In Figure 7b we see that this holds true for $Rounds = \{1, 2, 3\}$. For $Rounds = \{4, 5\}$ this property is no longer precisely valid. This can be explained based on our implementation methodology. Since we allow Synopsys DC to optimize the overall Verilog of our design, it finds optimizations and chooses slightly different cells for the designs in question, leading to slightly different delays. Therefore, we observe some variations. Figure 7c reports the delay assuming the worst case scenario of requiring $(\log_2(\Pi) - 1)$ cycles to scale the divisor (we call this *Delay Max*). Observe that the plots for different Π no longer overlap due to the logarithmic dependency of the delay on Π. Again, based on these plots and the ENOB values (Table I), a user can choose the best design based on the delay-error (or Delay Max-error) trade-offs.

Figure 7d plots the power performance of our SP divider. The total power consumption of the circuit is shown. This graph behaves as expected, where the designs with larger



(a) (b) (c) (d)

Fig. 6: RMSE v/s Exact Result



(a) (b) (c) (d)

Fig. 7: Circuit Implementation of SP Divider

II and/or larger $Rounds$ have a larger power consumption. The user can use this data along with the data in Figure 6, to choose a design based on the power-error trade-off. The circuits with larger power have a smaller error, as indicated by the corresponding ENOB value (Table I).

Note that the incremental ENOB gains beyond $Rounds = 2$ or 3 is small (Table I), and the area and power increase linearly with $Rounds$. This suggests that a practical value of $Rounds$ is 2 or 3.

In the next section, we will see how our SP divider compares with other approximate and non-approximate dividers, reported in the literature.

2) Comparisons with Alternate Methods: In this section we will study how our SP divider performs relative to other approximate and non-approximate (precise) dividers. First, we will benchmark the area, power and delay of the SP divider against a fixed-point precise divider. Next, we compare the SP divider against dividers in the SC domain.

CFPD (Combinational-Fixed-Point-Divider)			
Bit Width	**Delay(ps)**	**Area(μm^2)**	**Total Power(mW)**
2	76.43	0.73	0.42
3	201.73	2.08	1.06
4	505.10	3.66	2.11
5	749.43	5.92	3.53
6	1012.58	9.20	5.46
7	1348.88	12.16	7.56
8	1785.82	17.50	10.81
9	2099.32	21.51	13.17

TABLE II: CFPD Circuit Implementation

Fixed-Point Divider: In order to compare our divider against a "precise" divider, we chose a fixed point divider design from the Synopsys DesignWare Library. The division logic used in these precise dividers relies on the Newton-Raphson [19] approximation technique. As a result, a purely combinational divider design will require cascaded CPAs (Carry-Propagation-Adders) and the area, power and delay numbers increase super-linearly with operand bit-width. Synopsys therefore provides both combinational and sequential dividers, and recommends the use of sequential dividers for operand bit-width ≥ 16.

For a fair comparison, we compare dividers of *equal precision*. The precision of an SP divider in terms of binary bits is determined by ENOB', and the precision of a precise divider is determined by its bit-width. The highest ENOB value across our designs is 8.89 (referring to the values in Table I). Therefore, we synthesize fixed-point dividers with operand bit-widths of ≤ 9, and use the Combinational-Fixed-Point-Divider (*CFPD*) component in the Synopsys DesignWare library, to compare our work with.

Table II reports the delay, area and total power for the CFPD for input operand bit-widths in the range [2, 9]. Table III presents the ratio the area, power and delay for SP dividers with an ENOB' of 8, with the corresponding CFPD design (with a bit-width 8). Each row in the table corresponds to a SP divider design (with a different $Round$ value). The values reported are ratios of the quantity for the SP divider to the corresponding quantity for the CFPD. The SP divider is clearly much faster than the CFPD, on average having a delay (delay max) that is just $0.08\times$ ($0.36\times$) that of the CFPD delay. The CFPD has a large delay because of a long critical path through the CPAs in the design. This speed-up is at the cost of higher area than the CFPD, with the SP divider being on average $25\times$ larger. A key reason for this increase is that the SP design is required to have an $O(1)$ delay (in terms of gates) per round. The power for the SP divider is again much better ($0.06\times$ on average) than the power for the CFPD. The main reason for such a large power difference is the large dynamic power component required in the CFPD.

SP Divider / CFPD Performance Ratios (Equivalent ENOB)						
Π	ENOB'	Rounds	Delay Ratio	Delay Max Ratio	Area Ratio	Power Ratio
512	8	1	0.05	0.33	18.37	0.04
512	8	2	0.07	0.35	21.36	0.04
512	8	3	0.09	0.37	24.79	0.06
512	8	4	0.10	0.38	29.00	0.07
512	8	5	0.10	0.39	31.30	0.08
Average:			**0.08**	**0.36**	**24.97**	**0.06**

TABLE III: SP Divider v/s CFPD (for $ENOB' = 8$)

CBDIV				
Binary Input Size	Area(um^2)	Delay(ps)	Latency(ps)	Total Power (mW)
4	4.58	74.83	1.20E+03	0.15
8	11.39	87.38	2.24E+04	0.25
16	20.76	104.83	6.87E+06	0.31
32	44.89	127.60	5.48E+11	0.49
64	92.22	154.69	2.85E+21	0.79

TABLE IV: CBDIV Circuit Implementation

SC Dividers: Next, we compare the performance of our SP divider against an approximate divider from the SC domain. We chose CBDIV [13], a very recent work that builds on CORDIV in the SC domain, as our benchmark circuit. Using the same methodology as before, we implemented the circuits for CBDIV, in order to get the area, delay and power numbers. The result of our synthesis is shown in Table IV. The inputs to CBDIV are binary numbers (they convert from binary to SC as part of the design). Therefore, the first column in the table refers to the size of input operands in the binary number system. The corresponding SP divider is one with an ENOB' that matches the binary input size. CBDIV is a sequential circuit that uses an FSM to perform the correlation that is required for the underlying CORDIV logic. *Latency* here refers to the total amount of time required to generate the result of division. Latency is calculated using the formula: $Delay * Cycles$. The number of cycles required by the sequential circuit of CBDIV is 2^N, where N is the number of binary bits. Table V compares our SP divider with CBDIV (for $ENOB' = 8$). Comparing equivalent CBDIV and SP division circuits we see that SP division has much better delay max and power-delay-max-product numbers, and CBDIV has better area and power numbers. This is expected, as CBDIV is designed to be a serial circuit that achieves small area (and power) at the cost of long computation time. Note that the CBDIV design includes the circuitry to convert binary numbers to SC numbers and back. Chaining together successive division operations in CBDIV incurs expensive conversions to-and-from the binary number system.

V. Conclusion

In this paper we presented an approximate division technique called SP division. Prior to this work, there was no division scheme in SP arithmetic. We provide the user/designer different SP division design variants, allowing the designer to chose based on the specific area-power-delay-error tradeoffs of the target application. We have shown that on average, and SP divider with ENOB' of 8, is $12\times$ faster, and consumes $16\times$ less power, than the corresponding Fixed-Point-Combinational-Divider. We also developed a constant delay method to correlate two SP numbers.

SP Divider / CBDIV Performance Ratios (Equivalent ENOB)							
Π	ENOB'	Rounds	Delay Ratio	Delay Max Ratio	Area Ratio	Power Ratio	Power Delay Max Product (PDP) Ratio
512	8	1	<0.01	0.03	28.22	1.79	0.053
512	8	2	0.01	0.03	32.81	1.91	0.057
512	8	3	0.01	0.03	38.09	2.40	0.072
512	8	4	0.01	0.03	44.55	3.09	0.092
512	8	5	0.01	0.03	48.08	3.51	0.105
Average:			**0.01**	**0.03**	**38.35**	**2.54**	**0.075**

TABLE V: SP Divider v/s CBDIV (for $ENOB' = 8$)

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*. IEEE, 2015, pp. 554–566.

[3] Z. Wang, S. Mohajer, and K. Bazargan, "Low latency parallel implementation of traditionally-called stochastic circuits using deterministic shuffling networks," in *ASPDAC*, 2018, pp. 337–342.

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

[5] A. Alaghi and J. P. Hayes, "Survey of Stochastic Computing," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, may 2013. [Online]. Available: https://doi.org/10.1145/2465787.2465794

[6] S. Sharifi Tehrani, W. Gross, and S. Mannor, "Stochastic decoding of LDPC codes," *IEEE Communications Letters*, vol. 10, no. 10, pp. 716–718, 2006.

[7] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, "VLSI Implementation of Deep Neural Network Using Integral Stochastic Computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2688–2699, 2017.

[8] 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.

[9] K. Bharathi, J. Hu, and S. P. Khatri, "Scaled Population Subtraction for Approximate Computing," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*, 2020, pp. 348–355.

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

[11] S.-I. Chu, "New Divider Design for Stochastic Computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 1, pp. 147–151, 2020.

[12] N. Temenos and P. P. Sotiriadis, "Deterministic Finite State Machines for Stochastic Division in Unipolar Format," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[13] S. Yu, Y. Liu, and S. X.-D. Tan, "Approximate Divider Design Based on Counting-Based Stochastic Computing Division," in *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*, 2021, pp. 1–6.

[14] F. Dekking, C. Kraaikamp, H. Lopuhaä, and L. Meester, *A Modern Introduction to Probability and Statistics: Understanding Why and How*, ser. Springer Texts in Statistics. Springer, 2005. [Online]. Available: https://books.google.com/books?id=XLUMIlombgQC

[15] K. Scott and S. P. Khatri, "Flash-based Digital to Analog Conversion," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, 2022.

[16] "IEEE Standard Verilog Hardware Description Language," *IEEE Std 1364-2001*, pp. 1–792, 2001.

[17] L. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm FinFET predictive process design kit," *Microelectronics*, vol. 53, pp. 105–115, Jul. 2016.

[18] L. T. Clark, V. Vashishtha, D. M. Harris, S. Dietrich, and Z. Wang, "Design flows and collateral for the ASAP7 7nm FinFET predictive process design kit," in *2017 IEEE International Conference on Microelectronic Systems Education (MSE)*, 2017, pp. 1–4.

[19] Wikipedia contributors, "Newton's method — Wikipedia, the free encyclopedia," 2022, [Online; accessed 19-May-2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Newton%27s_method&oldid=1087753056