Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.0000/ACCESS.2019.DOI

MACISH: Designing Approximate MAC Accelerators with Internal-Self-Healing

G.A. GILLANI[1], MUHAMMAD ABDULLAH HANIF[2], B. VERSTOEP[1], S.H. GEREZ[1], MUHAMMAD SHAFIQUE[2], AND A.B.J. KOKKELER[1]

[1]Faculty of EEMCS, University of Twente, Enschede 7500 AE, Netherlands
[2]Faculty of Informatics, Vienna University of Technology (TU Wien), Austria

Corresponding author: G.A. Gillani (e-mail: s.ghayoor.gillani@utwente.nl).

This work was conducted in the context of the ASTRON and IBM joint project, DOME, funded by the Netherlands Organization for Scientific Research (NWO), the Dutch Ministry of EL&I, and the Province of Drenthe.

ABSTRACT Approximate computing studies the quality-efficiency trade-off to attain a best-efficiency (e.g., area, latency, and power) design for a given quality constraint and vice versa. Recently, self-healing methodologies for approximate computing have emerged that showed an effective quality-efficiency trade-off as compared to the conventional error-restricted approximate computing methodologies. However, state-of-the-art self-healing methodologies are constrained to highly parallel implementations with similar modules (or parts of a datapath) in *multiples of two* and for square-accumulate functions through the pairing of *mirror* versions to achieve error cancellation. In this article, we propose a novel methodology for Internal-Self-Healing (ISH) that allows exploiting self-healing within a computing element *internally* without requiring a paired, parallel module, which extends the applicability to irregular/asymmetric datapaths while relieving the restriction of *multiples of two* for modules in a given datapath, as well as going beyond square functions. We employ our ISH methodology to design an approximate multiply-accumulate (xMAC), wherein the multiplier is regarded as an approximation stage and the accumulator as a healing stage. We propose to approximate a recursive multiplier in such a way that a near-to-zero average error is achieved for a given input distribution to cancel out the error at an accurate accumulation stage. To increase the efficacy of such a multiplier, we propose a novel 2×2 approximate multiplier design that alleviates the overflow problem within an $n \times n$ approximate recursive multiplier. The proposed ISH methodology shows a more effective quality-efficiency trade-off for an xMAC as compared to the conventional error-restricted methodologies for random inputs and for radio-astronomy calibration processing (up to 55% better quality output for equivalent-efficiency designs).

INDEX TERMS Approximate computing, approximate accelerators, approximate multiply-accumulate, approximate multiplier, internal-self-healing methodology, radio astronomy processing.

I. INTRODUCTION

Approximate Computing has shown high efficiency gains with regard to power, performance, and chip-area for error resilient applications [1], [2]. Such applications include machine-learning, multimedia digital signal processing, and scientific computing that can tolerate a quantified error within the computation while producing an acceptable output. The quantification of error tolerance is achieved by utilizing error-resilience analysis tools [3]–[8]. Approximate computing techniques exploit this error tolerance to optimize the computing systems at software-, architecture- and circuit-level to achieve the aforesaid efficiency gains [9]–[12].

The conventional approximate computing methodology suggests utilizing *fail-small*, *fail-rare*, or *fail-moderate* strategies [8], [13], wherein the errors are restricted as per their magnitudes and rates to avoid high loss in the output-quality. This is referred to as the *conventional methodology* in this article. The *fail-small* technique allows approximations within the computing system that can have high error rates with low error magnitudes [8]. On the other hand, the *fail-rare* technique refers to the introduction of approximations that introduce high error magnitudes with low error rates [8]. The *fail-moderate* technique allows moderate error magnitude with moderate error rate approximations [13]. An important drawback of the conventional methodology is a limited design-space, which excludes the approximations that

introduce high error magnitudes and high error rates. This limitation hinders the achievable efficiency gains for a given quality constraint and therefore limits the efficacy of the *quality-efficiency* trade-off [14], where a *high quality* means a low error at the output and a *high efficiency* means a low computational cost in terms of chip-area, latency, and power/energy.

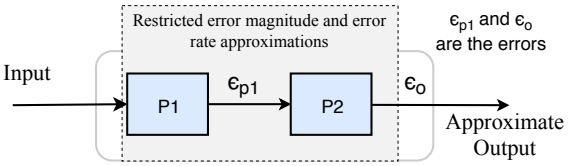
Recently proposed *fail-balanced* techniques for approximate computing have alleviated the aforesaid limitation in the design space. These techniques do not restrict the approximations based on their error profiles but provide an opportunity for the error cancellation to deliver an effective quality-efficiency trade-off [14], [15]. This is referred to as the *self-healing methodology* here. Consider an example of a computing architecture, composed of two computing elements: P1 and P2, as shown in Fig. 1. The input stream is fed to P1 while the output is obtained from P2. The conventional methodology suggests approximating both computing elements with controlled error rates and error magnitudes to avoid an unacceptable (high) loss in the output-quality; see Fig. 1a. On the other hand, the self-healing methodology considers P1 as an *approximation stage* and P2 as a *healing stage*. The approximations are applied at the *approximation stage* (approximate P1) in such a way that their corresponding error is canceled out (partially or fully) in the subsequent *healing stage* (accurate P2). To achieve this, a pair of approximate P1 elements is required with a *mirror error effect*, i.e., the error introduced by each P1 in a pair is an additive or multiplicative inverse of the other [14]; see Fig. 1b.

A serious limitation of the state-of-the-art self-healing methodology is that it can only be employed in parallel architectures that have similar computing elements (or parts of a datapath) in *multiples of two*, so that the *mirror error effect* is achieved by pairing the similar computing elements. However, in case of irregular/asymmetric datapaths that do not have similar elements in *multiples of two*, an approximation methodology is required that can provide the *mirror error effect* within a single computing element, as targeted in this article.

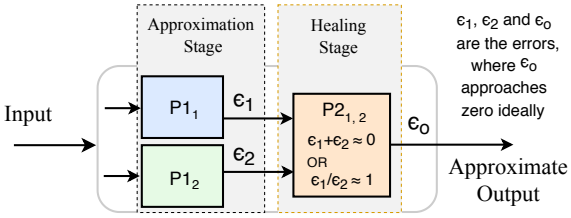
A. NOVEL CONTRIBUTIONS

The principal contribution of this work is a novel *Internal-Self-Healing (ISH)* methodology where the *approximation stage* (P1, see Fig. 1c) is designed for an internal *mirror error effect* without requiring a parallel paired computing element. To elaborate on the ISH methodology, the following is proposed in this article,

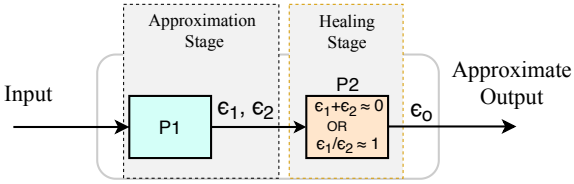
- The approximate multiply-accumulate (xMAC) concept with ISH methodology (Section III).
- Design of an $n \times n$ recursive multiplier with near-to-zero mean error and its efficacy for xMAC (Section III-A).
- Overflow handling scheme for near-to-zero mean error recursive multipliers and design of a novel approximate 2×2 multiplier that alleviates the overflow problem (Section III-B).



(a) Conventional approximate computing methodology.



(b) State-of-the-art self-healing approximate computing methodology [14].



(c) Proposed Internal-Self-Healing (ISH) approximate computing methodology.

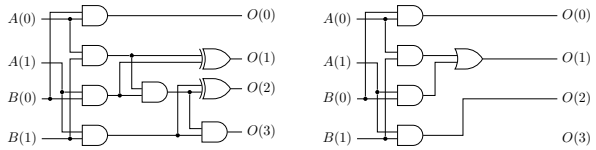
FIGURE 1: An overview of the conventional, the self-healing and the proposed approximate computing methodologies. The proposed ISH methodology does not require parallel computing elements but provides *mirror error effect* within a single approximate element (P1).

We also present a design space exploration based on a given input distribution (Section IV). We compare the conventional and the proposed ISH methodologies for chip-area and power optimized designs considering data with uniform and normal distributions and data obtained from a radio astronomy application (Section V).

II. BACKGROUND AND RELATED WORK

This section reviews the essential concepts concerning approximate multipliers, MAC, and the designs available in literature that correspond to the conventional and self-healing methodologies.

Approximate circuits for multipliers [14], [23]–[31] and adders [16]–[22] have been investigated for their pivotal role in digital signal processing architectures. Approximate recursive multipliers have been designed for their benefits of low power consumption and the possibility of fine-grained optimization based on the input distribution [14], [23]–[25]. An $n \times n$ recursive multiplier utilizes elementary 2×2 multiplier modules. An approximate 2×2 multiplier (M1) [23] features a lower complexity of the circuit (see Fig. 2b) as compared to the accurate design (M), see Fig. 2a. This brings a better chip-area, power and latency of M1 as compared to M. However, M1 brings one error case out of sixteen possible

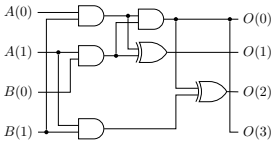


(a) Accurate (M).

(b) M1: 3*3↦7.

A / B	00	01	10	11
00	0000	0000	0000	0000
01	0000	0000	0010	0010
10	0000	0010	0100	0110
11	0000	0010	0110	1001

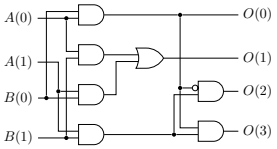
(c) Truth table of M2.



(d) M2.

A / B	00	01	10	11
00	0000	0000	0000	0000
01	0000	0001	0010	0011
10	0000	0010	0100	0110
11	0000	0011	0110	1011

(e) Truth table of M3.



(f) M3: 3*3↦11.

FIGURE 2: 2×2 Multiplier designs; M1 [23] and M2 [24] correspond to the conventional methodology, while M3 [14] corresponds to the self-healing methodology.

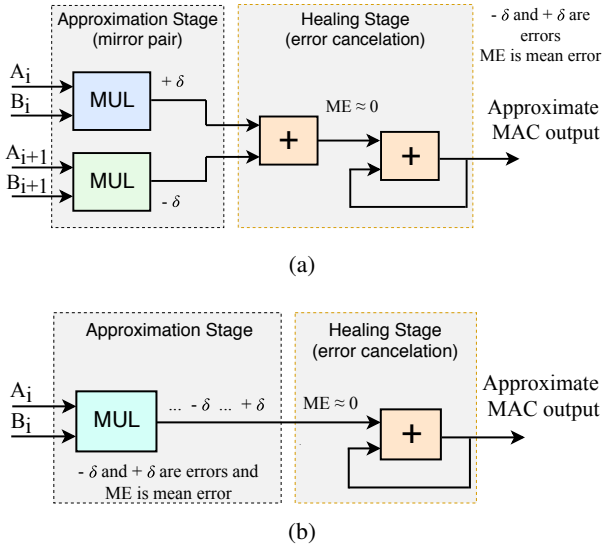


FIGURE 3: Approximate MAC designs, (a) utilizing the state-of-the-art self-healing methodology [14], (b) utilizing the proposed ISH methodology (MACISH), where approximation is achieved with $\pm\delta$ errors within a single multiplier module, which can be averaged out at the accumulator.

input combinations (3*3↦7), where the error_rate=1/16 and error_magnitude=2. Another approximate design, M2 [24], also provides a better efficiency as compared to M, while producing three error cases (Fig. 2c, 2d) with error_rate=3/16 and error_magnitude=1. M1 has a higher error magnitude and a lower error rate as compared to M2, therefore M1 can be regarded as a *fail-rare* design while M2 as a *fail-small* design, and M1 and M2 correspond to the conventional approximate computing methodology. To enable self-healing (*fail-balanced* design), [14] proposes M3 (Fig. 2e, 2f) that is a *mirror* of M1, i.e., it produces an error case ($\epsilon = +2$) which is an additive inverse of M1 ($\epsilon = -2$). Although, M3 requires more hardware as compared to M1, combining M1 and M3 in a pair has shown an overall effective quality-efficiency trade-off for square-accumulate architectures [14].

In case of approximate MAC (xMAC) accelerators, approximate multipliers that produce near-to-zero mean error provide the opportunity of error cancellation at the accumulation stage. A related approximate multiplier, DRUM, has been demonstrated for producing a near-to-zero mean error for uniformly distributed input, by optimizing the widths of input operands of a multiplier [26]. However, the applications that exhibit other input distributions (e.g., Gaussian) cannot utilize DRUM. On the other hand, the approximate recursive multipliers can be optimized based on the input distribution but they do not exhibit a near-to-zero mean error by original design [26]. Interestingly, we demonstrate in Section III that they can be re-designed to achieve a near-to-zero mean error profile while retaining their primary benefits.

Truncated multiplication in a MAC architecture has also been studied [32], [33], where the primary aim is to restrict the bit-width of multipliers and produce low error MAC computations by diminishing the effects of truncation. Other design approaches for approximate MAC utilize hybrid redundant adders [34], and an offset compensation to alleviate the inaccuracies of the approximate multiplier stage [35]. However, no exploitation of the self-healing methodology has been studied to the best of our knowledge.

III. DESIGNING AN APPROXIMATE MAC WITH THE INTERNAL-SELF-HEALING (ISH) METHODOLOGY

A MAC operation computes,

$$\sum_{i=1}^{N}(A_i * B_i) \tag{1}$$

where A and B are the input vectors of length N. To design an approximate MAC (xMAC) in compliance with the state-of-the-art self-healing methodology [14], the multiplication is considered as an *approximation stage* and the accumulation as a *healing stage*; see Fig. 3a. A pair of approximate multipliers is utilized such that they produce errors that are additive inverse of each other, i.e., $\epsilon_1 = +\delta$ and $\epsilon_2 = -\delta$, so that the expected value of the mean error approaches zero. This helps the accurate accumulator to cancel out the errors originated in the approximate multipliers. However, such a methodology is limited to architectures that have multiple MAC pairs in parallel, which is not always the case as discussed in Section I. Therefore, we propose an xMAC accelerator where an approximate multiplier can generate $+\delta$ and $-\delta$ errors internally, without requiring a parallel multiplier; see Fig. 3b. This relieves the restriction of *multiples of two* computing units. Moreover, the proposed xMAC

can also be utilized for parallel architectures by deploying a number of xMACs as per the desired level of parallelism. Our design is also well-suited for asymmetric datapaths, i.e., accelerators where the number of multipliers or MAC processing iterations are not a multiple of two.

A. APPROXIMATE MULTIPLIER FOR MAC

A key challenge in employing the ISH methodology for xMAC is to achieve an approximate multiplier that exhibits a near-to-zero mean error profile for a given input distribution, so that the subsequent accurate accumulator can average out the errors originated in the approximate multiplier. Here we discuss an approximate $n \times n$ unsigned recursive multiplier with the desired property, where n is the bit-width of input operands, $n \in \{2, 4, 8, 16, ...\}$.

An $n \times n$ recursive multiplier is constructed using $(n/2)^2$ elementary (2×2) multipliers [23]–[25]. These 2×2 multipliers generate partial products. Summation of the bit-shifted partial products produce the overall output of an $n \times n$ recursive multiplier. Fig. 4 shows cases of 4×4 ($O_{4\times4}$) and 8×8 ($O_{8\times8}$) recursive multiplication that are composed of four and sixteen 2×2 multipliers, respectively. Any number out of the set of 2×2 multipliers and/or adders can be approximated to achieve an approximate multiplier [23], [24]. However, in this work we only apply approximations in the 2×2 multipliers as in [14]. Therefore, any combination of approximate 2×2 multipliers, e.g., M1, M2 and M3 (Fig. 2), can be utilized to form an approximate $n \times n$ multiplier.

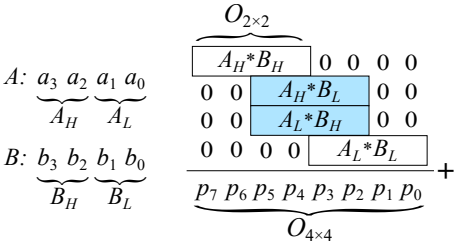
To achieve a near-to-zero mean error profile, the 2×2 *multipliers that have equal numerical weights (shown as same colored boxes in Fig. 4) can be approximated with* $+\delta$ *and* $-\delta$ *errors.* For example, in case of a 4×4 multiplier, the output ($O_{4\times4}$) can be expressed as follows (see Fig. 4a),

$$O_{4\times4} = A_L * B_L + 4(A_L * B_H) + 4(A_H * B_L) + 16(A_H * B_H) \quad (2)$$

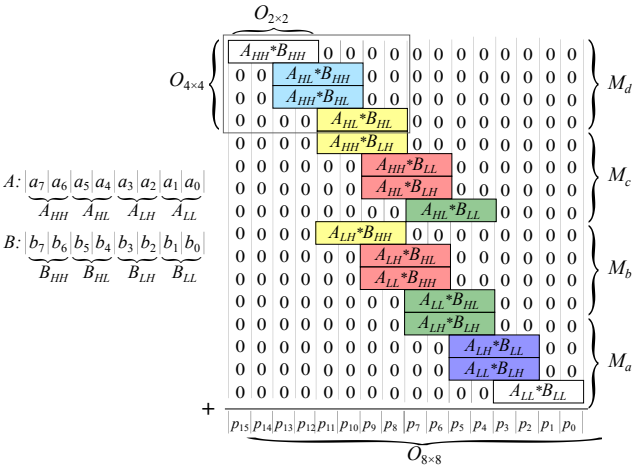
where the constants 4 and 16 are representing the shift factors. If M1 is deployed for $A_L * B_H$, M3 for $A_H * B_L$, and M for the other two, the expected mean error value of the multiplier (for uniformly distributed input vectors) is zero. Therefore, an xMAC utilizing such an approximate multiplier has an expected error value of zero for uniformly distributed input vectors. Likewise, near-to-zero expected error value configurations can be chosen for other input distributions.

B. OVERFLOW HANDLING

A challenge for designing an $n \times n$ recursive multiplier with near-to-zero mean error is the requirement of positive error ($\epsilon = +\delta$) 2×2 approximate multipliers like M3, which may result in the overall output exceeding the $2n$ bits. We define an *overflow configuration* as the configuration of an $n \times n$ multiplier consisting of any combination of 2×2 multipliers like M, M1, M2 and M3 that may overflow for any possible input combination. Here we discuss how to identify the *overflow configurations* in order to discard them



(a) 4×4 recursive multiplication requires four 2×2 multipliers.



(b) 8×8 recursive multiplication requires sixteen 2×2 multipliers.

FIGURE 4: Recursive $n \times n$ multiplication utilizes elementary 2×2 multipliers. The same colors show equal numerical weight 2×2 multipliers that can be approximated with $+\delta$ and $-\delta$ errors to enable ISH.

during design space exploration, and also propose a novel 2×2 approximate multiplier design that helps to alleviate the overflow problem.

1) Overflow Examples

Consider a 4×4 multiplication operation as shown in Fig. 4a. Let $A = (1111)_2$ and $B = (1111)_2$. This implies $A_H = A_L = B_H = B_L = (11)_2 = 3$, therefore Eq. (2) becomes,

$$O_{4\times4} = 3 * 3 + 4(3 * 3) + 4(3 * 3) + 16(3 * 3)$$

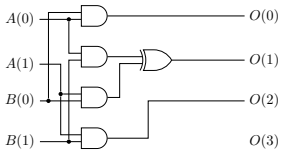
assuming M3 ($3*3 \mapsto 11$) is deployed for all 2×2 multipliers,

$$O_{4\times4} = 11 + 4(11) + 4(11) + 16(11)$$
$$= 275 = (1\ 0001\ 0011)_2$$

the output exceeds 8 bits ($2n$). Therefore the above example is an *overflow configuration* for a 4×4 multiplier, and is not desired. In case of a 4×4 multiplier, the overflow occurs as the value of the output is greater than 255, i.e., $2^{2n} - 1$. However, while constituting a higher order multiplier, say 8×8 multiplier, a 4×4 multiplier with an output value of less than 255 may also overflow the higher order multiplier. Note that 255 is still considerably larger than the maximum possible accurate output value of a 4×4 multiplier, which

B \ A	00	01	10	11
00	0000	0000	0000	0000
01	0000	0001	0010	0011
10	0000	0010	0100	0110
11	0000	0011	0110	0101

(a) Truth table of M4.



(b) M4: 3*3↦5.

FIGURE 5: A proposed 2×2 approximate multiplier for overflow compensation.

is 225 (i.e., $(2^n - 1)^2$). Consider an 8×8 multiplication (Fig. 4b), and let the constituting four 4×4 multiplications be represented by M_a, M_b, M_c and M_d such that the least significant multiplication is M_a while the most significant is M_d. The following expression represents the 8×8 computation,

$$O_{8\times8} = M_a + 16(M_b) + 16(M_c) + 256(M_d) \quad (3)$$

where the constants 16 and 256 are representing the shift factors. Let $A = (1111\ 1111)_2$ and $B = (1111\ 1111)_2$. Let M3 (3*3↦11), M3, M1 (3*3↦7) and M (3*3↦9) compute the $A_L{*}B_L$, $A_L{*}B_H$, $A_H{*}B_L$ and $A_H{*}B_H$ partial products respectively for each of the 4×4 multipliers. Therefore, each of the 4×4 multipliers will generate,

$$O_{4\times4} = 11 + 4(11) + 4(7) + 16(9) = 227$$

and Eq. (3) becomes,

$$\begin{aligned} O_{8\times8} &= 227 + 16(227) + 16(227) + 256(227) \\ &= 65603 = (1\ 0000\ 0000\ 0100\ 0011)_2 \end{aligned}$$

the output exceeds 16 bits (i.e., $2n$), therefore this is an *overflow configuration*. So, even in cases where none of the 4×4 multipliers lead to overflow, the resulting 8×8 multiplier can cause overflow. In general, any $n\times n$ multiplier configuration that is not an overflow configuration in itself but has a maximum output value of greater than $(2^n - 1)^2$, may overflow a higher order $2n \times 2n$ multiplier.

2) A Novel 2×2 Approximate Multiplier

To alleviate the overflow problem, we propose an approximate 2×2 multiplier design (M4), as shown in Fig. 5, which provides a larger negative error ($\epsilon = -4$) as compared to M1 ($\epsilon = -2$). Note that M4 can be balanced with two M2 ($\epsilon = +2$) in order to achieve the internal-self-healing. Noteworthy, M4 is useful in the design of near-to-zero mean error recursive multipliers as it reduces the maximum possible output value of an $n \times n$ multiplier. For instance, if M4 is employed to only $A_H * B_H$ in Eq. (2), it averts the possibility of overflow no matter which of the combination out of the given choices (M/M1/M2/M3/M4) is used for the other three 2×2 multipliers.

3) Overflow Handling Scheme

To identify the overflow configurations, we propose to assess each $n \times n$ multiplier configuration step-wise, from 4×4 constituting multipliers to an overall $n\times n$ multiplier. Without loss of generality, we elaborate on an 8×8 recursive multiplication operation. For each 8×8 configuration, firstly, we need to check an overflow for each of the four 4×4 multipliers,

$$\Gamma_4 = \max(O_{4\times4}) < 2^8 \quad (4)$$

where Γ_4 is the maximum possible output value of a 4×4 multiplier. If Eq. (4) fails for any of the four 4×4 multipliers, the configuration is discarded. Then we need to check the maximum possible output value of an overall 8×8 multiplier (Γ_8),

$$\Gamma_8 = \sum_{j=1}^{4}[\Gamma_4(j) * S(j)] < 2^{16} \quad (5)$$

which is essentially the summation of the products of maximum possible values of constituting 4×4 multipliers ($\Gamma_4(j)$) and their respective shift factors ($S(j)$). Likewise, additional steps can be added to identify overflow, or, to select non-overflow configurations for higher order recursive multipliers.

To automate overflow handling for an $n \times n$ approximate multiplier configuration, we propose to utilize a recursive function (see Section IV for details), where at each stage (n_r), the function checks the following condition for identifying valid configurations,

$$\Gamma_{n_r} < 2^{2n_r} \quad (6)$$

here n_r is the current recursive stage, $n_r \in \{4, 8, ..., n/2, n\}$. The related maximum possible output value (Γ_{n_r}) can be computed as,

$$\begin{aligned} \Gamma_{n_r} &= \sum_{j=1}^{4}[\Gamma_{(n_r/2)}(j) * S(j)] \\ &= \Gamma_{M_a} + 2^{n_r/2}\Gamma_{M_b} + 2^{n_r/2}\Gamma_{M_c} + 2^n\Gamma_{M_d} \end{aligned} \quad (7)$$

where $\Gamma_{M_a}, \Gamma_{M_b}, \Gamma_{M_c}$ and Γ_{M_d} are the maximum possible output values of the constituting $n_r/2 \times n_r/2$ multipliers (sub-multipliers).

C. COMPARISON OF THE PROPOSED ISH WITH THE CONVENTIONAL APPROXIMATE COMPUTING METHODOLOGY

1) Terminology and Notation

We follow the notation introduced in [11] and extend that to incorporate approximate recursive multipliers. Let $\mathbb{I}$ be a set of inputs that is mapped to $\mathbb{O}$ as the function f is executed in its exact form, i.e., $f : \mathbb{I} \mapsto \mathbb{O}$. Let $f^* : \mathbb{I} \mapsto \mathbb{O}^*$ and $f^{*\prime} : \mathbb{I} \mapsto \mathbb{O}^{*\prime}$ be the execution of the same function in approximate form by utilizing the conventional and the ISH methodologies, respectively. Let $\mathfrak{D}$ be the design space offered by an approximate computing methodology, which is essentially a set of all possible design configurations offered by the respective methodology, i.e.,

$$\mathfrak{D} = \{C_1, C_2, C_3, ..., C_g\} \quad (8)$$

here g is the number of design alternatives/configurations offered by the respective approximate computing methodology, and each C_i is a design configuration that characterizes a specific point: (e_i, q_i) in the quality-efficiency trade-off. Where e_i is efficiency and q_i is quality offered by C_i. We assume a high efficiency of design that offers a low computational cost (chip-area, power consumption or latency) and vice versa. Similarly, we assume a high quality of design that offers a low output error and vice versa.

Supposing an $n \times n$ recursive multiplier, the function f corresponds to multiplication operation. Let $\mathfrak{D}^*$ and $\mathfrak{D}^{*\prime}$ be the design space offered by the conventional and the ISH approximate computing methodologies respectively. The conventional approximate computing methodology utilizes the conventional error-restricted elementary (2×2) multipliers (M1, M2) along with the accurate version (M). Let $\mathfrak{K}^*$ be a set of elementary multipliers utilized by the conventional approximate computing methodology, i.e., $\mathfrak{K}^* = \{\text{M, M1, M2}\}$. On the other hand, the proposed ISH methodology utilizes the conventional and the proposed self-healing based elementary multipliers, $\therefore\ \mathfrak{K}^{*\prime} = \{\text{M, M1, M2, M3, M4}\}$, where $\mathfrak{K}^{*\prime}$ is a set of elementary multipliers utilized by the ISH methodology. It can be noted that all elements of $\mathfrak{K}^*$ are included in $\mathfrak{K}^{*\prime}$, i.e., $\mathfrak{K}^* \subset \mathfrak{K}^{*\prime}$. Therefore,

$$\mathfrak{D}^* \subset \mathfrak{D}^{*\prime} \tag{9}$$

2) Comparison

To compare the trade-offs offered by two methodologies, we define *effectivity* ($\mathfrak{E}$), such that $\mathfrak{E}$ is a function of quality and efficiency. A design methodology (with an effectivity of $\mathfrak{E}_1$) is considered to be more effective than the other (with the effectivity of $\mathfrak{E}_2$), i.e., $\mathfrak{E}_1 > \mathfrak{E}_2$, if and only if it provides a better efficiency for a given output quality, and a better quality for a given efficiency. As shown in Eq. (9), the design alternatives offered by the proposed ISH methodology include the design alternatives offered by the conventional methodology, and at the top of that, the ISH methodology also offers new designs that help error cancellation. Consequently, the proposed ISH methodology provides a quality-efficiency trade-off that is always more effective (or at least equally effective in the worst case) as compared to that of the conventional methodology counterpart, i.e.,

$$\mathfrak{E}_{(f^{*\prime}:\mathbb{I}\mapsto\mathbb{O}^{*\prime})} \geq \mathfrak{E}_{(f^{*}:\mathbb{I}\mapsto\mathbb{O}^{*})} \tag{10}$$

Besides the overall trade-off, it is also important to analyze the error bounds of an approximate circuit that affect its feasibility for a target application. Marzek et al. [29] formalized the Worst Case Error (WCE) of a recursive multiplier as,

$$\text{WCE}_n = \text{WCE}_{M_a} + 2^{n/2}\text{WCE}_{M_b} + 2^{n/2}\text{WCE}_{M_c} + 2^{n}\text{WCE}_{M_d} \tag{11}$$

where WCE_n is the worst case error of an $n \times n$ recursive multiplier, and WCE_{M_a}, WCE_{M_b}, WCE_{M_c} and WCE_{M_d} represent the worst case errors of the four constituting $(n/2 \times n/2)$ multipliers (sub-multipliers) respectively. In case of an approximate multiplier that is designed in a conventional way, Eq. (11) represents the WCE that occurs when a worst case input triggers the error cases of all the approximate sub-multipliers. On the other hand, consider M_b and M_c are mirrored, such that they have error magnitudes that are additive inverse of each other, i.e., utilizing the proposed ISH methodology. If an input triggers an error case for each sub-multiplier, the second and third terms in Eq. (11) cancel out. In fact, the WCE for such an ISH based approximate multiplier occurs when one of the M_b or M_c does not have an error triggering input and is given as,

$$\text{WCE}_n = \text{WCE}_{M_a} + 2^{n/2}\text{WCE}_{(M_b,M_c)} + 2^{n}\text{WCE}_{M_d} \tag{12}$$

where $\text{WCE}_{(M_b,M_c)}$ is the worst case error of M_b and M_c, which occurs when only one of them introduces an error, and the error has a same direction (sign) as that of M_a and M_d. Hence, the worst case error (WCE_n) of the ISH methodology can never be greater than that of the conventional methodology. Keeping in view the design space relation in Eq. (9), and the worst case errors for the conventional (see Eq. (11)) and the ISH (see Eq. (12)) methodologies, we have,

$$\text{WCE}_{(f^{*\prime}:\mathbb{I}\mapsto\mathbb{O}^{*\prime})} \leq \text{WCE}_{(f^{*}:\mathbb{I}\mapsto\mathbb{O}^{*})} \tag{13}$$

From Eq. (10) and Eq. (13), it can be concluded that it is always beneficial to employ the proposed ISH methodology as compared to the error restricted conventional approximate computing methodology. Moreover, we quantify the benefits offered by the proposed ISH methodology in the subsequent sections.

IV. DESIGN SPACE EXPLORATION METHODOLOGY

To quantify the gains offered by the ISH methodology as compared to the conventional methodology, we need to find the best (optimal/near-optimal) quality-efficiency designs for each. In this section, we present our design space exploration methodology that leads us to such approximate multiplier configurations for an approximate MAC unit. These designs are referred to as the *pareto-optimal* designs/configurations in this article. The methodology is designed such that it allows us to explore the design space in a reasonably small amount of time while using limited computational and memory resources.

A. HUGE DESIGN SPACE - A CHALLENGE

Fig. 4 shows a 4×4 and an 8×8 multiplier built using 2×2 elementary modules. As can be seen, the number of elementary multipliers increases rapidly with the increase in the number of bits per input (operand). The number of 2×2 elementary modules required for an $n \times n$ multiplier can mathematically be given as: $(n/2)^2$.

The total number of possible configurations for an approximate multiplier directly depends on the number of elementary multipliers and the number of types that each can

TABLE 1: Number of configurations for a few example scenarios with different bit-widths (n) of multipliers and types of elementary 2×2 designs (m).

S. No.	n	m	No. of Configurations
1.	8	3	4.3×10^{7}
2.	8	5	1.53×10^{11}
3.	16	3	3.43×10^{30}
4.	16	5	5.42×10^{44}

have. Assuming m as the number of types of elementary multipliers, the total number of possible configurations for an $n \times n$ multiplier can mathematically be given as:

$$No.\ of\ \ configurations = m^{(n/2)^2} \tag{14}$$

As can be inferred from Eq. (14), the number of configurations grows rapidly both with m and n. To further highlight the requirement of a systematic design space exploration methodology, Table 1 presents the number of possible configurations for a few example cases with different m and n values. It can be seen that a huge design space has to be explored for a 16×16 multiplier case with only 5 options for elementary 2×2 designs (S. No. 4). To tackle such an enormous design space, we propose a heuristic that prunes the search space in order to find the pareto-optimal configurations effectively.

B. PROPOSED METHODOLOGY FOR DESIGN SPACE EXPLORATION

Our design space exploration methodology employs a recursive algorithm with intermediate pruning for fast exploration. The intermediate pruning is employed to prune less-effective parts of the overall design space at each intermediate stage to reduce the design space for the next subsequent stage. The overall flow is illustrated in Fig. 6 and the related algorithms are given in Appendix A. The main steps of the methodology are as follows.

Initialization

In this step, we define a variable $E\_Configs$ which stores the error and cost characteristics as well as the identities (IDs) of the elementary (2×2) multipliers. The error characteristics are stored in the form of an error map (E_Maps), which contains the output error for each possible input combination of a 2×2 multiplier. An example illustration of an E_Map is shown in Fig. 7. The cost characteristics include the area and/or power values of the elementary multipliers.

Step 1

Given the probability distributions of the input operands, i.e., ρ_x and ρ_y, the first step involves input probability computation of all the individual elementary (i.e., 2×2 in our case) multipliers in an $n \times n$ multiplier. The input probability distribution of all the elementary modules is stored in a matrix ρ, where each entity of the matrix represents the input probability distribution of a single elementary multiplier and has a cumulative sum of 1.

To compute the input probability of all the elementary multipliers, we first independently compute the probability distribution of the pairs of bits of the input operands x and y which are the inputs to these elementary multipliers. Similar to [25], the probability distribution of a pair of consecutive bits of the input operand x can be given as:

$$P_x\{i\}(k) = \sum_{q=0}^{2^{n-2i-2}-1} \sum_{p=0}^{2^{2i}-1} \rho_x(q \times 2^{2i+2} + k \times 2^{2i} + p) \tag{15}$$

where i defines the pair of bits in the input operand, i.e., i^{th} pair consists of the bits at locations $2i$ and $2i+1$, and k defines the combined decimal value of the bits ($i \in \{0, 1, 2, ..., n/2-1\}$ and $k \in \{0, 1, 2, 3\}$). Similarly, the probability distribution of a pair of consecutive bits (defined by j) of the input operand y can be given as:

$$P_y\{j\}(l) = \sum_{r=0}^{2^{n-2j-2}-1} \sum_{s=0}^{2^{2j}-1} \rho_y(r \times 2^{2j+2} + l \times 2^{2j} + s) \tag{16}$$

where $j \in \{0, 1, 2, ..., n/2-1\}$ and $l \in \{0, 1, 2, 3\}$. Using Eq. 15 and Eq. 16, and assuming x and y as independent random variables, the input probability distribution of a specific 2×2 multiplier (represented by $\rho\{i, j\}$) can be computed using the following equation:

$$\rho\{i, j\}(k, l) = P_x\{i\}(k) \times P_y\{j\}(l) \tag{17}$$

As an example, consider a 4×4 multiplier that consists of four 2×2 multipliers. The probability distributions of the four 2×2 multipliers are given as $\rho\{0, 0\}$, $\rho\{0, 1\}$, $\rho\{1, 0\}$, and $\rho\{1, 1\}$, where the probability distribution of each 2×2 multiplier, say $\rho\{0, 1\}$, has a probability-value for each input combination, i.e., $\rho\{0, 1\}(0, 0)$, $\rho\{0, 1\}(0, 1)$, $\rho\{0, 1\}(0, 2)$, $\rho\{0, 1\}(0, 3)$, $\rho\{0, 1\}(1, 0)$, ..., $\rho\{0, 1\}(3, 3)$.

Step 2

The ρ computation step is followed by a recursive step where at each call the $n_r \times n_r$ multiplier is divided into four $n_r/2 \times n_r/2$ sub-multiplier units (see Fig. 4a for an example of a 4×4 multiplier) and for each sub-multiplier Step 2 is called again with the corresponding multiplier size and input distribution (Step 2a). Note that for the very first call to Step 2 (i.e., while moving from Step 1 to 2) the variable n_r is initialized with n where n_r represents a local variable that defines the bit-width of the inputs of the multiplier at a particular recursive stage. From each intermediate stage, the step returns at maximum X number of highly-efficient configurations given a defined multiplier size and input probability distribution. Here X represents a parameter which defines the maximum number of representative configurations that can be selected from an intermediate recursive stage. It should

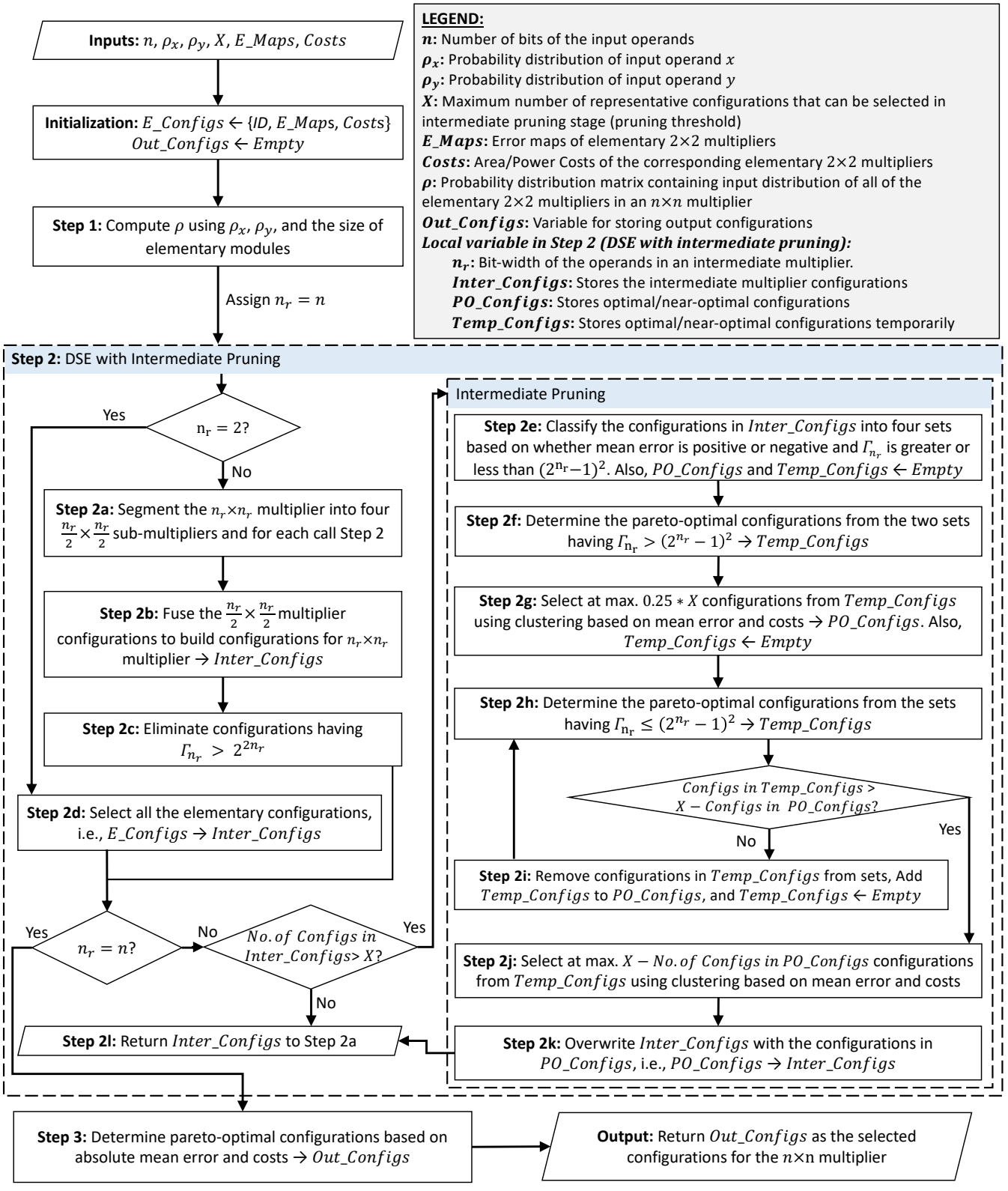


FIGURE 6: The proposed design space exploration methodology for approximate recursive multipliers. Our methodology takes into account the bit-widths and the probability distributions of inputs, pruning threshold (X), and error & cost characteristics of the elementary multipliers to return the best (pareto-optimal/near-optimal) configurations while using limited computational and memory resources.

B \ A	$0=(00)_2$	$1=(01)_2$	$2=(10)_2$	$3=(11)_2$
$0=(00)_2$	0	0	0	0
$1=(01)_2$	0	0	0	0
$2=(10)_2$	0	0	0	0
$3=(11)_2$	0	0	0	-4

FIGURE 7: An example of E_Map (error map) for M4 elementary multiplier shown in Fig. 5.

be noted that a large number of representatives results in a big design space when combined for larger multipliers and thus the design space exploration consumes more time and computational resources.

The received configurations for all the four sub-multipliers are then combined to generate the possible configurations for the $n_r \times n_r$ multiplier which are stored in $Inter\_Configs$ (Step 2b). At the same step, the mean error values, the maximum possible output values, and the costs of the generated configurations are also computed using the error and cost characteristics of the corresponding sub-multipliers. The mean error of a configuration of an $n_r \times n_r$ multiplier composed of four $n_r/2 \times n_r/2$ multipliers can be computed as,

$$ME_{n_r} = ME_a + 2^{n_r/2} ME_b + 2^{n_r/2} ME_c + 2^{n_r} ME_d \quad (18)$$

where ME_{n_r} represents the mean error of the $n_r \times n_r$ multiplier and ME_a, ME_b, ME_c and ME_d represent the mean error of the M_a, M_b, M_c and M_d sub-multipliers, respectively (see Fig. 4b for an example of an 8×8 multiplier). Similarly, the maximum possible output value of a configuration of an $n_r \times n_r$ multiplier can be computed using Eq. (7).

To compute the area and power costs of the generated configurations, we have utilized the model that will be discussed in Section V-B. The model estimates the costs of an overall multiplier by adding the costs of the corresponding sub-multipliers together with their contribution to the adder trees. Once the configurations have been generated and all the required characteristics have been computed, the configurations are then checked for the maximum possible output value to avoid overflow conditions (Step 2c), as mentioned in Section III-B. All the configurations having a maximum possible output value greater than or equal to 2^{2n_r} (for an $n_r \times n_r$ multiplier) are removed from the $Inter\_Configs$. At this point, the value of n_r is compared with n and if it is equal, all the configurations are forwarded to Step 3. However, if n_r is not equal to n, the remaining number of configurations is checked and intermediate pruning (Step 2e-2k) is applied if it is greater than a pre-specified threshold, i.e., X. This is mainly done to reduce the number of possible configurations at the preceding higher stage such that the design space exploration can be performed using limited computational and memory resources and in a time-efficient manner.

The recursive function keeps on calling itself unless the size of the sub-multipliers is equivalent to 2×2 (i.e., $n_r = 2$), which acts as the termination point for the recursive calling. At this point, Step 2d is performed where $Inter\_Configs$ is initialized with $E\_Configs$ (i.e., all the elementary multiplier configurations) along with their mean errors and maximum possible output values. The mean error for each elementary multiplier is computed by taking the dot product of the E_Map of the corresponding elementary multiplier with the input probability distribution matrix. Note that Step 2d is called for each elementary module location in an $n \times n$ multiplier and the probability matrix used for computing the mean errors of the elementary multipliers is the one which contains the input probability distribution of that particular location. After initializing the $Inter\_Configs$, it is checked for the total number of configurations and returned to Step 2a if the number of configurations is less than X; otherwise intermediate pruning is performed on it. To avoid any confusion, it is important to highlight that $Inter\_Configs$ in Step 2 is a local variable.

Intermediate Pruning (Step 2e - 2k)

Whenever the number of intermediate configurations in $Inter\_Configs$ (after Step 2c or Step 2d) is greater than X and $n_r \neq n$, the intermediate pruning is called for choosing a subset of X effective configurations which can be used as the representatives of the complete design space of the sub-multiplier. To achieve this, at Step 2e, we classify the configurations into four sets based on the mean error and maximum possible output value of the configurations. Set 1 contains configurations having mean error > 0 and maximum output value $> (2^{n_r} - 1)^2$, Set 2 contains configurations having mean error $<= 0$ and maximum output value $> (2^{n_r} - 1)^2$, Set 3 contains configurations having mean error > 0 and maximum output value $<= (2^{n_r} - 1)^2$, and Set 4 contains configurations having mean error $<= 0$ and maximum output value $<= (2^{n_r} - 1)^2$. The configurations are divided into these four sets because of two main reasons: 1) So that different number of configurations can be selected from different sets based on their importance, for example, the configurations having maximum output value greater than $(2^{n_r} - 1)^2$ may result in overflow as mentioned in Section III-B and, therefore, should be given less importance; and 2) The configurations having positive and negative mean error should be given equal importance, as only in case configurations with both positive and negative mean error are available, the internal self-healing can be utilized to generate approximate configurations for larger multipliers that result in zero/near-to-zero mean error.

To select a subset of effective configurations, we first find pareto-optimal configurations from sets 1 and 2 based on absolute mean error and cost and store temporarily in $Temp\_Configs$ (Step 2f). Then, in Step 2g, we check the number of pareto-optimal configurations. If it is greater than 25% of X, we first select the two extreme values from the pareto-optimal configurations, i.e., configurations having

minimum and maximum absolute mean error, and then apply k-means clustering to find $0.25 * X - 2$ clusters using the rest of the pareto-optimal configurations based on mean error and cost. Here, k-means [36] is applied to group configurations offering nearby error-cost points in the quality-efficiency trade-off. The configuration closest to the cluster centroid is then selected from each cluster as its representative. The selected configurations are then stored in a local variable $PO\_Configs$. Moreover, if the number of pareto-optimal configurations in Step 2f is less than $0.25 * X$, all the configurations are selected and stored in $PO\_Configs$. Also, in the same step the $Temp\_Configs$ is re-initialized to empty.

The remaining configurations are selected from set 3 and 4 using Step 2h, 2i and 2j. In Step 2h, we find the pareto-optimal configurations from the sets based on the absolute mean error and the cost of the configurations and store them in $Temp\_Configs$. If the selected number of configurations from these sets is greater than the remaining number of configurations (i.e., greater than $X-$ No. of configurations in $PO\_Configs$), we perform Step 2j to find the remaining required configurations from $Temp\_Configs$ using clustering (similar to Step 2g). However, if it is less, we perform Step 2i where we remove the configurations from the sets that are present in $Temp\_Configs$, and we add the configurations of $Temp\_Configs$ to $PO\_Configs$. Finally, we re-initialize $Temp\_Configs$ to empty before moving back to Step 2h. Then, Step 2h is performed again using the modified sets to find near-optimal points. This cycle (Step 2h $\rightarrow$ Step 2i $\rightarrow$ Step 2h) is repeated until the condition is satisfied (or the sets are empty). This procedure ensures that we select the most effective (optimal/near-optimal) configurations from the sets as much as allowed by the X parameter. Afterwards, Step 2j is performed and the selected configurations are added to $PO\_Configs$. The resultant configurations are forwarded to Step 2k where $Inter\_Configs$ is overwritten with the configurations in $PO\_Configs$. Then the intermediate pruning function is returned to Step 2l, where the $Inter\_Configs$ are forwarded to the higher stage (Step 2a) for generating configurations of larger multipliers.

Step 3

From the received configurations the pareto-optimal configurations are found using their absolute mean error and the area/power cost characteristics. The resultant configurations are then returned as the final configurations for the $n \times n$ multiplier.

C. VIABILITY OF OUR APPROACH

We utilized the above design space exploration methodology for finding the pareto-optimal designs for 4-bit, 8-bit and 16-bit multipliers. Table 2 shows the runtime of the simulations (with $X = 60$) using MATLAB (2017a) on an Intel Core i5-6600 CPU with 16 GB of RAM. We have also simulated the first case ($n = 8$ and $m = 3$) exhaustively which resulted in a simulation runtime of 43 seconds on our system. Interestingly, the pareto-optimal configurations for the aforesaid exhaustive simulation are exactly the same as of our algorithm at $X = 60$, which has a simulation runtime of 5 seconds. Moreover, our methodology enables us to explore a huge design space ($n = 16$ and $m = 5$) in less than four minutes using a general purpose computer system as a simulation platform.

TABLE 2: Simulation runtime for the design space exploration of multipliers. While using a general purpose simulation platform, our methodology explores a huge design space (S. No. 4) in less than four minutes.

S. No.	n	m	No. of Configurations	Simulation Time (Seconds)
1.	8	3	4.3×10^7	5
2.	8	5	1.53×10^{11}	7
3.	16	3	3.43×10^{30}	138
4.	16	5	5.42×10^{44}	209

V. EXPERIMENTAL RESULTS

To study the quality-efficiency trade-off for approximate MAC accelerators and to compare the proposed internal-self-healing (ISH) methodology with the conventional methodology, we have performed a design space exploration for area- and power-optimization for uniform and normal input distributions. As 8-bit architectures are widely used in the signal processing applications [38]–[42], our experiments are mainly focused on 8-bit designs. However, we also compare 4-bit and 16-bit designs to test the scalability of our methodology.

A. EXPERIMENTAL SETUP

A quality analysis was performed using function-accurate behavioral implementations of accurate and approximate $n \times n$ recursive multipliers, and a hardware efficiency analysis was performed utilizing Synopsys Design Compiler and Power Compiler for the TSMC 40nm Low Power (TCBN40LP) technology library, as shown in Fig. 8. To fix the latency budget of all the synthesized designs, a fixed operating frequency of 1 GHz has been utilized for hardware efficiency analysis. This legitimates the area and power comparison of various design alternatives to ensure a fair comparison. We have utilized the *compile_ultra* command for synthesizing all designs. Questasim has been utilized for functional verification and to generate the switching activity for power estimation. For normally distributed inputs, the following mean (μ) and standard deviation (σ) values have been considered, 4-bit case: ($\mu = 8$, $\sigma = 1.5$), 8-bit case: ($\mu = 128$, $\sigma = 22.5$), and 16-bit case: ($\mu = 32768$, $\sigma = 6553$).

B. DESIGN SPACE EXPLORATION OF THE PROPOSED ISH METHODOLOGY

We have performed design space exploration as discussed in Section IV to obtain the best designs offered by the ISH methodology. These best designs are referred to as the *pareto-optimal* configurations/designs, and the line joining

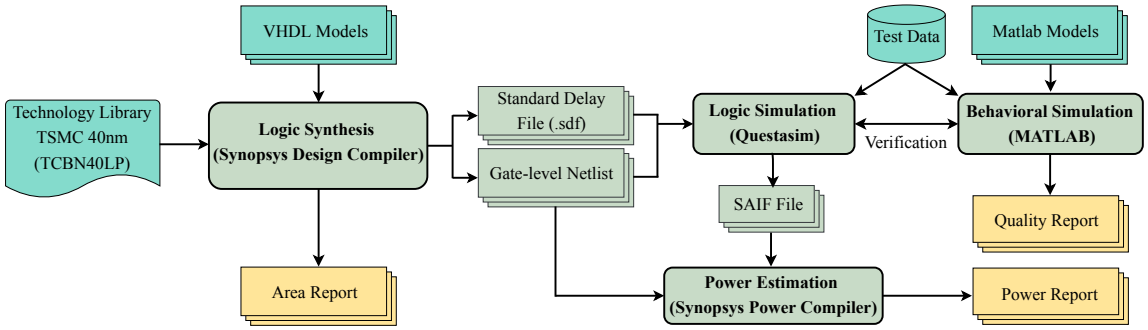


FIGURE 8: Experimental setup utilized for the quality-efficiency trade-off study [14].

TABLE 3: 2×2 multiplier cost (conversely: efficiency) estimation for TSMC 40nm Low Power library at 1 GHz. The estimation also includes the costs related to the adder trees within a higher order multiplier.

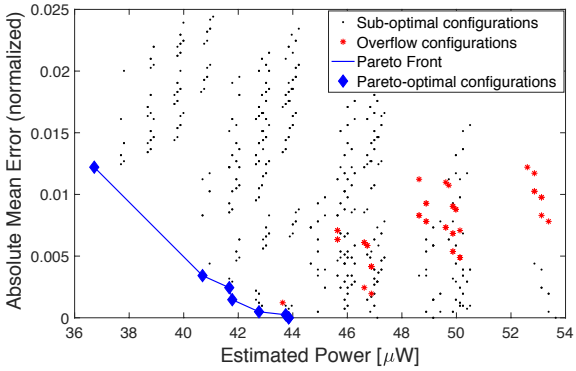
Design Type	4×4 **Multiplier**		8×8 **Multiplier**	
	Area (μm^2)	*Power* [a]	*Area* (μm^2)	*Power* [a]
M	21.52	13.41	32.43	27.59
M1	13.29	9.18	25.20	22.34
M2	19.17	10.16	31.11	22.06
M3	19.17	13.15	31.21	27.47
M4	16.76	10.27	27.36	22.66

[a]Power (μW) estimates based on uniformly distributed input.

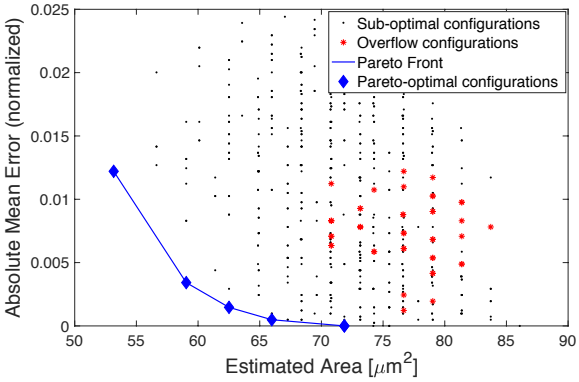
the pareto-optimal points in the quality-efficiency trade-off is regarded as the *pareto front*.

One way of estimating the hardware costs of an $n \times n$ recursive multiplier is to add up the costs of the constituting sub-multipliers [14], [29]. However, this ignores the hardware costs related to adder trees within an $n \times n$ multiplier. Therefore, the cost estimation proposed in [14], [29] is useful for ranking purpose only, and has an underlying assumption that the costs of adder trees will follow the same trend as that of the sub-multipliers. In this work, we have utilized a more effective way of cost estimation that also includes the cost contributions of the adder trees related to the sub-multipliers. Firstly, we obtain the cost of an $n \times n$ multiplier composed of multiples of a unique 2×2 multiplier, say M1, using the Synopsys tool flow. Then we divide the cost of an $n \times n$ multiplier by the number of total 2×2 designs constituting an $n \times n$ multiplier. This includes the area/power costs of the 2×2 multipliers along with the related adders, and therefore provides a plausible estimation of hardware costs, or conversely: the hardware efficiency.

Table 3 shows the estimated hardware costs of the considered 2×2 multipliers that are utilized for estimating the costs of design configurations during the design space exploration. Note that an 8×8 multiplier needs more adders as compared to a 4×4 multiplier to add the partial products. Therefore, each of the 2×2 multipliers in Table 3 has a lower cost estimate while constituting a 4×4 multiplier as compared to while constituting an 8×8 multiplier.



(a) Design space exploration for power optimization.



(b) Design space exploration for area optimization.

FIGURE 9: Quality-efficiency trade-off study of a 4×4 multiplier optimized for uniformly distributed inputs.

Fig. 9 shows the complete design space of a 4×4 recursive multiplier utilizing the five 2×2 multiplier options (M, M1, M2, M3, and M4), optimized for uniformly distributed input. The absolute mean error shown at the y-axis (in all our results) is normalized to the output range of the multiplier, i.e., 2^{2n}, where n is the bit-width of the input operands. Red asterisks represent the overflow configurations that are identified and discarded (using Eq. (6)) while choosing pareto-optimal configurations. Table 4 shows the pareto-optimal configurations for a 4×4 recursive multiplier based on uniformly

TABLE 4: Pareto-optimal configurations for a 4×4 recursive multiplier based on uniformly distributed input.

Power Optimization				Area Optimization			
LSM*	→		MSM*	LSM	→		MSM
M1	M1	M1	M1	M1	M1	M1	M1
M1	M1	M1	M3	M1	M1	M1	M3
M1	M2	M1	M3	M1	M4	M1	M3
M1	M4	M1	M3	M1	M4	M4	M3
M1	M4	M2	M3	M4	M2	M4	M3
M2	M4	M2	M3	-	-	-	-
M4	M4	M2	M3	-	-	-	-

*LSM and MSM are the least significant and the most significant 2×2 multipliers respectively.

distributed input. The left column shows the power optimized pareto-optimal configurations and the right column shows the area optimized ones. Each configuration contains four 2×2 multipliers, e.g., M1 M1 M1 M1, where the left-most 2×2 multiplier is the Least Significant Multiplier (LSM) and the right-most one is the Most Significant Multiplier (MSM) of a 4×4 configuration. The hardware efficiency increases and the output-quality decreases as we go from the bottom row to the top row. It can be seen that most of the pareto-optimal configurations include the self-healing based designs like M3 [14] and M4. This substantiates the importance of M3 and M4 designs, where M4 is a novel 2×2 multiplier design proposed in this work.

C. SCALABILITY AND COMPARISON OF ISH WITH THE CONVENTIONAL METHODOLOGY

To compare the proposed ISH and the conventional approximate computing methodologies, we compare their pareto fronts for area- and power-optimized designs based on each input distribution (uniform and normal). As discussed in Section III-C, all four approximate designs (M1, M2, M3 and M4) are considered as 2×2 multiplier options for the proposed ISH methodology. However, only the conventional low error-rate (M1) and low error-magnitude (M2) approximate designs are considered for the conventional methodology.

Fig. 10 shows the pareto fronts for 4×4 recursive multipliers. It can be seen that the proposed ISH methodology presents many designs that have better efficiency for a given quality constraint and vice versa as compared to the conventional methodology counterparts. It should be noted that the additional design points (shown in Fig. 10b and 10d) are not worse as compared to the conventional methodology because they increase design options in the pareto front. However, such designs may be ignored as they do not provide much efficiency benefits as compared to their decreased quality. Moreover, it should be noted that Fig. 10 shows pareto-optimal configurations based on exhaustive search (without intermediate pruning algorithm), as the design space is small enough for a 4×4 multiplier case.

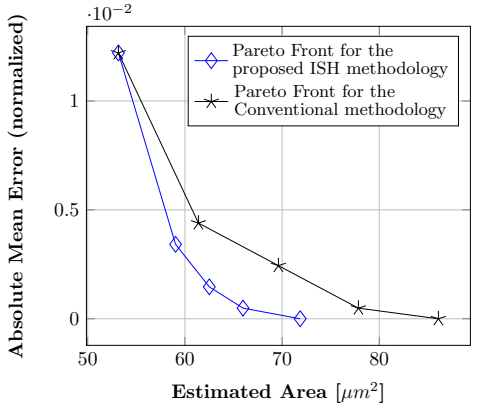
To verify the scalability of the proposed methodology, we also present a comparison for 8×8 and 16×16 recursive multipliers as shown in Fig. 11. Here the y-axis is shown in logarithmic scale to clearly illustrate the widely spread designs for comparison. It is to be noted that we have performed an exhaustive search for the 8×8 conventional methodology case, where the rest of the simulations (for Fig. 11) have been performed by utilizing the intermediate-pruning technique discussed in Section IV. It can be seen that the proposed ISH methodology clearly outperforms the conventional methodology for all considered input lengths by providing many designs that have better efficiency for a given quality constraint and vice versa.

In the case of the ISH methodology, it is noteworthy that the error drops relatively faster for increasing area/power costs in the beginning. This is because of error balancing that helps to reduce the error without using the accurate modules. However, at a certain stage, when the error is already very low, the rate of error drop (with respect to area/power) decreases (e.g., see Fig. 11b, designs: C6 and C7). This is because the usage of accurate modules is necessary to further reduce the error beyond this stage.

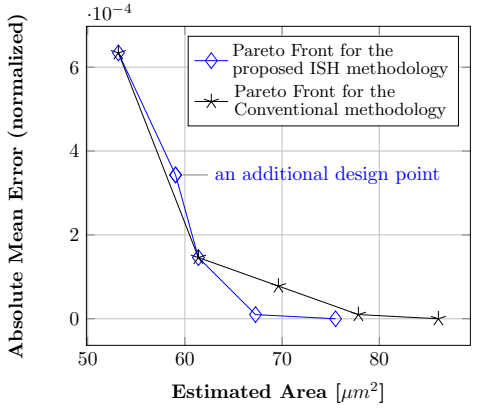
As discussed earlier, for 4×4 and 8×8 cases, exhaustive simulation is utilized to obtain pareto-optimal designs based on the conventional methodology, which means no conventional design can be better than them. Although there are some approximations involved within the intermediate-pruning algorithm that may provide near-optimal (instead of optimal) designs in a rare case, it generates better ISH designs as compared to the conventional exhaustively searched designs. This substantiates the fact that the ISH methodology performs better than the conventional error-restricted methodology, including the higher order input cases. Therefore we can conclude that the proposed ISH methodology provides a more effective quality-efficiency trade-off as compared to the conventional approximate computing methodology due to internal self-healing of the errors within the approximate modules, and this is independent of the target hardware efficiency (e.g., area or power), input width (e.g., 4-bit/8-bit/16-bit), and input distribution (e.g., uniform or normal).

D. CASE STUDY: RADIO ASTRONOMY CALIBRATION PROCESSING

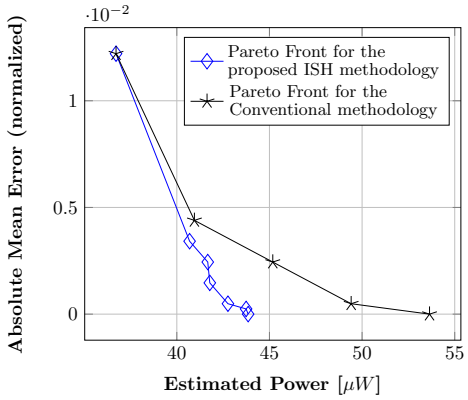
So far, we have shown results for general input distributions. Here we present the improvements offered by the ISH methodology for an application. Radio astronomy calibration estimates complex antenna gains (G) within a radio telescope by utilizing an iterative method, known as StEFCal [37]. For a given configuration, i.e., the number of antenna elements and receiving channels in a radio telescope, StEFCal estimates G by utilizing current visibilities (V) and the model visibilities (M). Considering a hardware accelerator design, it has three dominant kernels: complex-input element-wise product, complex-input square-accumulate and complex-input multiply-accumulate (MAC) [6], [14]. Here we focus on the quality-efficiency trade-off of the complex-



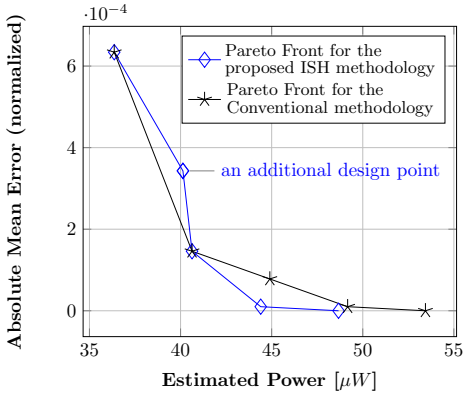
(a) Area optimization for uniformly distributed input.



(b) Area optimization for normally distributed input.



(c) Power optimization for uniformly distributed input.



(d) Power optimization for normally distributed input.

FIGURE 10: Comparison of the pareto-optimal designs of 4×4 multipliers based on ISH and the conventional approximate computing methodologies. The proposed ISH methodology outperforms for all considered optimization targets by providing better (or at least equal) efficiency designs for a given quality constraint and vice versa.

input MAC operation, which computes,

$$\sum_{j=1}^{N} \{Z_j * V_j\} \qquad Z, V \in \mathbb{C} \tag{19}$$

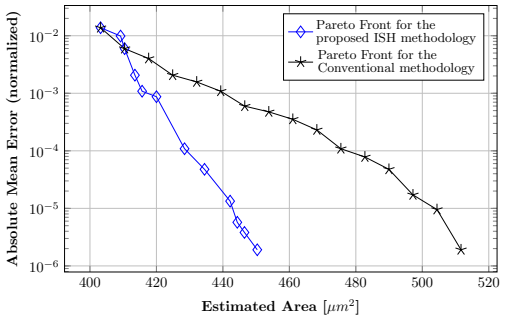
where Z represents an element-wise product of model visibility (M) and gain computed in the last iteration (G_{i-1}) [14]. We assume $N = 496$, which is the vector size for a radio telescope configuration of 124 antenna elements and 4 channels. It is to be noted that each complex multiplication requires four real-input multiplications. Therefore, in order to study the quality-efficiency trade-off, we have utilized pareto-optimal multipliers of the ISH and the conventional methodologies for all the four multiplications. Keeping in view the feasibility of 8-bit architectures in radio astronomy processing [39], we considered the MAC operation utilizing 8×8 multipliers. As shown in Eq. (10), the ISH methodology always provides better (or at least equivalent as a worst case) designs as compared to the conventional methodology. Therefore, here we present the cases that quantify the maximum benefits offered by the ISH methodology. Table 5 shows equivalent-efficiency designs for the area and power optimization. It can be seen that the area-optimized design of the ISH methodology (ISH_A) brings 27% improvement of the Mean Square Error (MSE) as compared to the equivalent-efficiency conventional methodology design (Conven_A). For power optimized designs, ISH methodology (ISH_P) offers 55% improvement in quality as compared to the conventional methodology counterpart (Conven_P).

TABLE 5: Employing equivalent-efficiency approximate MAC alternatives in radio astronomy calibration. The proposed ISH designs exhibit up to 55% better quality as compared to the conventional methodology counterparts.

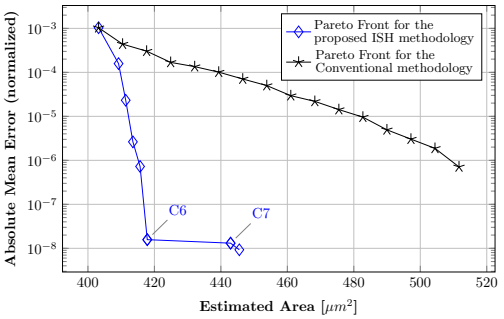
Design Alternatives	**MAC Error (MSE)**	**Hardware Cost***
Accurate	0	A = 519, P = 439
Conven_A	1.96e-02	A = 447 (P^s =389)
ISH_A	1.44e-02	A = 447 (P^s =384)
Conven_P	2.01e-02	P = 383 (A^s =445)
ISH_P	9.07e-03	P = 383 (A^s =472)

* A and P are Area (μm^2) and Power (μW) estimates (respectively) of each multiplier in a complex-input MAC accelerator.
^{s} These A and P costs are not the primary optimization targets.

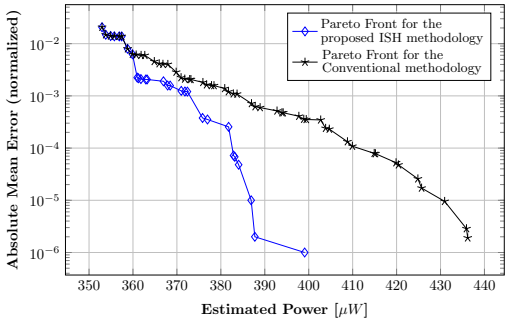
Table 5 also shows the power costs of area-optimized



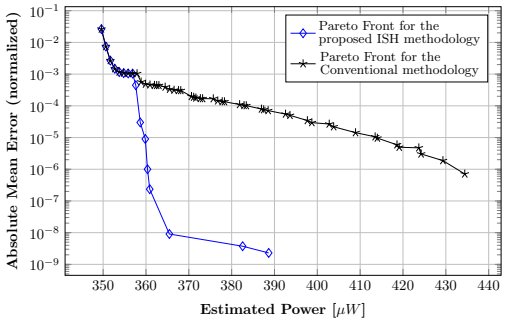
(a) Area optimization for uniformly distributed input (8×8).



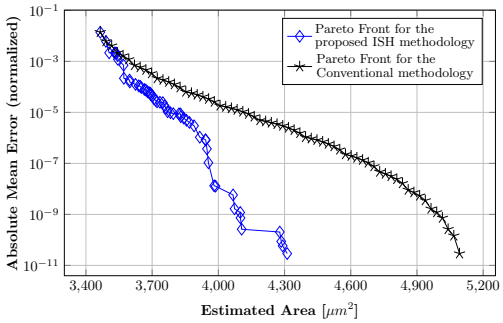
(b) Area optimization for normally distributed input (8×8).



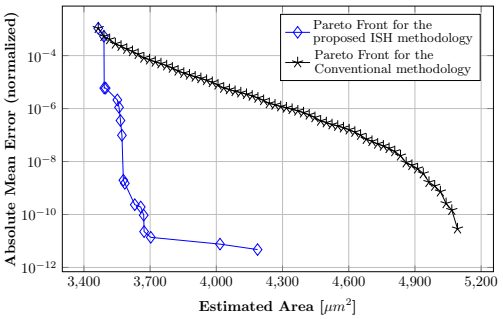
(c) Power optimization for uniformly distributed input (8×8).



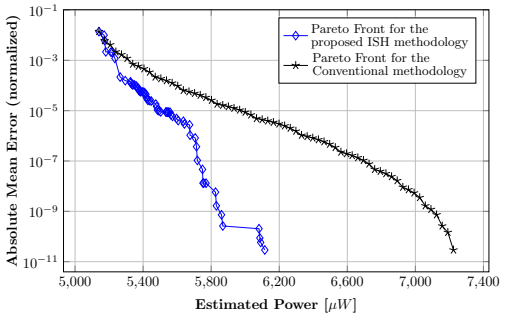
(d) Power optimization for normally distributed input (8×8).



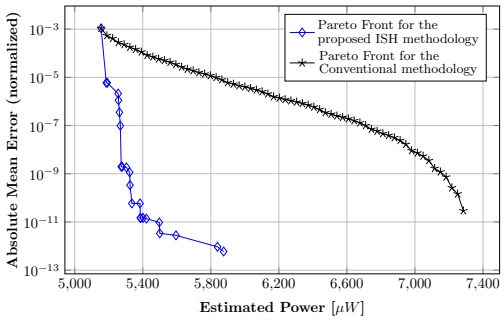
(e) Area optimization for uniformly distributed input (16×16).



(f) Area optimization for normally distributed input (16×16).



(g) Power optimization for uniformly distributed input (16×16).



(h) Power optimization for normally distributed input (16×16).

FIGURE 11: Comparison of the pareto-optimal designs of 8×8 ((a)-(d)) and 16×16 ((e)-(h)) recursive multipliers based on the ISH and the conventional approximate computing methodologies. The proposed ISH methodology outperforms for all considered optimization targets.

TABLE 6: Synthesis based comparison of a few pareto-optimal configurations for an 8×8 recursive multiplier. The proposed ISH methodology provides more effective quality-efficiency designs, e.g., ISH_1 shows 18% better area and 14% better power as compared to Conven_2, also with a better quality output.

Design Alternatives	**Pareto-optimal multiplier configurations optimized for normally distributed input** **LSM*** → **MSM***																**Quality-Efficiency** **Error***	**Area***	**Power***
Accurate	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	0	519	439
Conven_1	M1	M1	M1	M	M	M1	M	M1	M	M	M1	M1	M	M	M	M1	2.95e-5	455	387
Conven_2	M	M	M	M	M	M1	M	M	M	M	M	M	M	M	M	M1	1.87e-6	507	414
ISH_1	M4	M1	M1	M1	M1	M1	M4	M1	M1	M1	M1	M1	M3	M4	M1	M4	1.57e-8	415	356
ISH_2	M4	M1	M1	M	M4	M4	M3	M1	M1	M	M4	M1	M	M1	M3	M1	9.26e-9	464	373

* Unit of Area is μm^2. Power (μW) is obtained utilizing switching activity based on normally distributed input. Normalized absolute mean error is considered as an Error. LSM and MSM are the least significant and the most significant 2×2 multipliers respectively.

designs (Conven_A and ISH_A), and area costs for power-optimized designs (Conven_P and ISH_P). Although these power and area costs are not the primary optimization targets, it is important to note that they can introduce an additional trade-off. For instance, the conventional methodology design Conven_A consumes slightly more power as compared to the ISH counterpart (ISH_A); on the other hand, the conventional methodology design Conven_P requires less area as compared to the ISH counterpart (ISH_P). Moreover, it is to be noted that in case of the ISH methodology, the power-optimized designs (e.g., ISH_P) tend to utilize more M2 multipliers as they are cheap in power, on the other hand, they require more area (see Table 3). Keeping in view the aforesaid comparisons, it is important to optimize a design based on the efficiency target defined by the (application-specific) design specifications. In cases, where both chip-area and power consumption are equally important, a 3D trade-off graph can be plotted to identify the pareto-optimal points, where the x, y and z axis represent area, power and error respectively. This case study shows that utilizing the ISH methodology brings better quality for a given hardware cost (or efficiency target) as compared to the conventional methodology. However, in order to employ approximate modules optimized for radio astronomy processing, a comprehensive study of error resilience and input distribution of representative data sets is required, which is beyond the scope of this paper.

E. SYNTHESIS BASED COMPARISON

As yet, we have shown quality-efficiency improvements offered by the proposed ISH methodology based on estimated hardware cost models discussed in Section V-B. Here we present the results for comparable quality-efficiency designs to quantify hardware improvements based on the synthesis of complete units. For instance, we synthesize a few of the best 8×8 multiplier pareto-optimal designs that are optimized for normally distributed input. Table 6 shows the area and power of the considered designs for the conventional (Conven_1 and Conven_2) and the ISH (ISH_1 and ISH_2) methodologies. As can be expected, hardware efficiency increases as we compromise on the quality of output. For example, Conven_1 is more efficient (requires less area and power) as compared to Conven_2 and Accurate designs. On the other hand, Conven_2 and Accurate designs have a better quality of output as compared to Conven_1. Similarly, the efficiency of ISH_1 is higher than ISH_2.

To verify the validity of the cost *estimation* model discussed in Section V-B, we compare it with the *synthesis* based hardware cost. Fig. 12 shows the differences in area and power costs that vary from 0-4% for the considered designs. This shows that the estimation model provides a viable approach to rank the designs cost-wise in order to find the pareto-optimal configurations during the design space exploration process, especially because it's not possible to synthesize the millions of designs to obtain the synthesis-based hardware costs.

Table 6 also shows the design configurations of the considered designs. It can be seen that the designs based on the proposed ISH methodology tend to utilize M3 and M4 as approximate 2×2 multiplier options, which helps for error cancellation and avoiding overflow situation at the same time. It can also be noted that the approximate 8×8 designs tend to utilize an approximate 2×2 multiplier (M1/M4) at the most significant position. This is because of a very low probability of error for the considered normally distributed input ($\mu = 128$, $\sigma = 22.5$), i.e., for the aforesaid characteristics of input, it is hardly likely that both inputs of a 2×2 multiplier (placed at the most significant position) are 3. Interestingly, while comparing quality and efficiency of designs for the proposed ISH and the conventional methodologies (see Table 6), ISH_1 exhibits 18% better area and 14% better power as compared to Conven_2, and at the same time, it provides a better quality of output (less error) due to the error cancellation mechanism intrinsic to the ISH methodology.

VI. CONCLUSION

A novel Internal-Self-Healing (ISH) methodology is presented for approximate MAC accelerators, which utilizes the proposed approximate recursive multipliers with a near-to-zero mean error profile. In contrast to the state-of-the-art self-healing methodologies, the proposed ISH methodology has shown an opportunity for error cancellation within the approximate circuits, without requiring similar computing elements (or parts of a datapath) in *multiples of two*.

We have presented a quantitative analysis based on a design space exploration of 4-bit, 8-bit and 16-bit designs

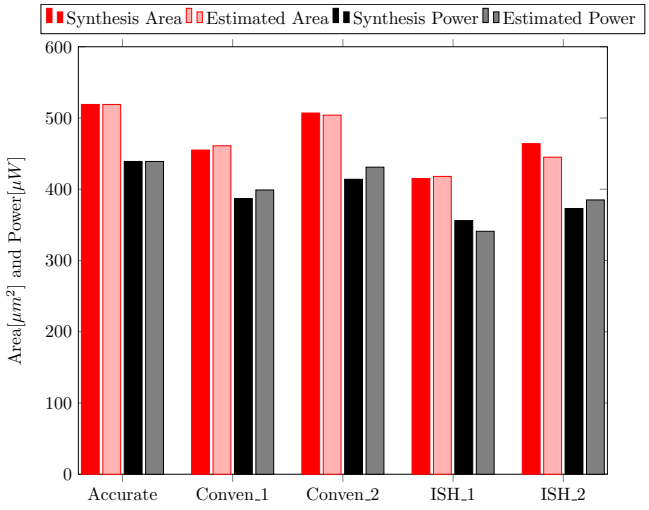


FIGURE 12: Comparison of the model based estimated cost and the synthesis based cost. The difference varies from 0-4% for both area and power costs.

for area and power optimization, considering uniform and normal input distributions, and radio astronomy calibration processing. Our results showed a more effective quality-efficiency trade-off offered by the ISH methodology as compared to the conventional approximate computing methodology. By definition, the proposed ISH methodology also considers the conventional designs in addition to the novel designs (for error cancellation) during the design space exploration. Therefore, it always generates better or at least equivalent designs with a higher *effectivity* (based on quality-efficiency trade-off) and a lower *worst-case error* as compared to the conventional methodology.

.

APPENDIX A PSEUDO-CODES FOR DESIGN SPACE EXPLORATION METHODOLOGY

To ease the reproducibility of our work, we present the pseudo-codes (Algorithm 1 and Algorithm 2) for our design space exploration methodology explained in Section IV; see Fig. 6 for the flow diagram. The definitions of the variables and functions utilized in the algorithms are given in Table 7.

The main algorithm, Algorithm 1, considers the probability distribution of the input operands and the types of elementary (2×2) multipliers in terms of their error maps (E_Maps) and costs ($Costs$) for searching the pareto-optimal configurations. The DEFINE_E_Configs function defines an array of structures containing different characteristics of the elementary configurations. The DSE function is the main recursive function used for performing the design space exploration whereas the DEFINE_E_Configs function is just to convert the characteristics of the elementary configurations to a well defined array of structures (a data type).

Algorithm 2 details the INTERMEDIATE_PRUNING function, which restricts the intermediate design space to a limited number of points. The aforesaid function utilizes the

TABLE 7: Definition of variables and functions used in Algorithm 1 and Algorithm 2.

Variables/Functions	Definition
n	Number of bits of the input operands.
n_r	Bit-width of the operands in an intermediate multiplier.
ρ_x and ρ_y	Probability distributions of the input operands of the $n \times n$ multiplier.
X	Maximum number of representative configurations that can be selected in an intermediate stage (pruning threshold).
E_Maps	An array of matrices containing error maps of the elementary 2×2 multipliers.
AO_Maps	A matrix containing the accurate output values of a 2×2 multiplier.
$Costs$	An array containing the area/power costs of the elementary 2×2 multipliers.
$E\_Configs$	An array of structures where each element contains the identifier of one of the elementary configurations, the maximum possible output value that it can generate, its area/power cost, and its error map.
$Out\_Configs$	An array of structures which contains the final output of the design space exploration, i.e., the pareto-optimal configurations of the $n \times n$ multiplier along with their error and area/power characteristics.
$Temp\_Configs$	A variable for temporarily storing the configurations. The type of the variable is the same as the $Out\_Configs$.
ρ	An array of matrices which contains the input probability distribution of each of the 2×2 elementary multiplier in an $n \times n$ multiplier.
$\rho\{x_1 : x_2, y_1 : y_2\}$	A block of ρ matrix which is defined by row indexes x_1 to x_2 and column indexes y_1 to y_2. For example, in Algorithm 1 in the DSE function the ρ matrix has to be divided into four blocks which is represented as $\rho\{(i-1)*n_r/4+1 : i*n_r/4, (j-1)*n_r/4+1 : j*n_r/4\}$ on line#24 of the algorithm. This term means that the represented block spans the rows from $(i-1)*n_r/4+1$ to $i*n_r/4$ and columns from $(j-1)*n_r/4+1$ to $j*n_r/4$.
$Max\_Nominal\_Output$	Maximum output expected from an accurate multiplier, i.e., $(2^n-1)^2$ for an $n \times n$ multiplier.
$Inter\_Configs$	Stores the intermediate multiplier configurations.
$PO\_Configs$	Stores the intermediate pareto-optimal multiplier configurations related to a specific recursive stage (n_r).
$Config\_set_1$	Set of configurations from $Inter\_Configs$ having $Mean\_Error > 0$ and maximum possible output value, i.e., $\Gamma > Max\_Nominal\_Output$.
$Config\_set_2$	Set of configurations from $Inter\_Configs$ having $Mean\_Error <= 0$ and maximum possible output value, i.e., $\Gamma > Max\_Nominal\_Output$.
$Config\_set_3$	Set of configurations from $Inter\_Configs$ having $Mean\_Error > 0$ and maximum possible output value, i.e., $\Gamma <= Max\_Nominal\_Output$.
$Config\_set_4$	Set of configurations from $Inter\_Configs$ having $Mean\_Error <= 0$ and maximum possible output value, i.e., $\Gamma <= Max\_Nominal\_Output$.
$Y.z$	z field of the structure Y. It can be a scalar, vector, or a matrix. For example, $E\_Configs\{i\}.E\_Map$ means the E_Map of the i^{th} elementary configuration.
$dot\_product(Y, Z)$	Dot product of the entities Y and Z.

CONFIGURATION_SELECTION function, which selects a limited number of points from a set of pareto-optimal configurations by applying k-means clustering [36]. To avoid the local minima problem, k-means operation is repeated multiple (M) times on the same set of points, each time with different initialization [36]. For each repetition of k-means, the loss ($Loss_{new}$) is computed and compared with the loss from the previous instance. If $Loss_{new}$ is less than that of the previous instance, the new set of clusters ($Clusters_{new}$) is selected as the optimal set otherwise the previous set is retained.

REFERENCES

[1] S. Venkataramani, S.T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in Proc. of the 52nd Annual DAC ACM/EDAC/IEEE, San Francisco, CA, USA, 2015, pp. 120.

Algorithm 1 Pseudo-code for design space exploration.

Input:
n, ρ_x, ρ_y, X, E_Maps and $Costs$
Initialize:
$E\_Configs$, $Out\_Configs$ and ρ = $Empty$

$E\_Configs$ = DEFINE_E_Configs(E_Maps, $Costs$)
Find ρ using **Eq. 17**
$Inter\_Configs$ = DSE(n, n_r, X, $E\_Configs$, ρ)
$Out\_Configs$ = Pareto-optimal configurations from $Inter\_Configs$ based on $Cost$ vs. absolute $Mean\_Error$
return $Out\_Configs$

function DEFINE_E_CONFIGS(E_Maps, $Costs$)
  for i=1:No. of maps in E_Maps **do**
    $E\_Configs\{i\}.ID = i$
    $E\_Configs\{i\}.E\_Map$ = $E\_Map\{i\}$
    $E\_Configs\{i\}.Cost$ = $Costs\{i\}$
    $E\_Configs\{i\}.\Gamma = max(AO\_Maps + E\_Map\{i\})$
  end for
  return $E\_Configs$
end function

function DSE(n, n_r, X, $E\_Configs$, ρ)
  if $n_r == 2$ **then**
    $Inter\_Configs$ = $E\_Configs$
    for i = 1 : no. of configurations in $Inter\_Configs$ **do**
      $Inter\_Configs\{i\}.Mean\_Error$ = dot_product($Inter\_Configs\{i\}.E\_Map$, ρ)
    end for
  else
    for $i = 1 : 2$ **do**
      for $j = 1 : 2$ **do**
        $Configs\{i,j\}$ = DSE(n, $n_r/2$, X, $E\_Configs$, $\rho\{(i-1)*n_r/4+1 : i*n_r/4, (j-1)*n_r/4+1 : j*n_r/4\}$)
      end for
    end for
    $Inter\_Configs$ = All possible configurations of an $n_r \times n_r$ multiplier using $Configs$
    Delete the configurations having $Inter\_Configs.\Gamma \geq 2^{2*n_r}$ from $Inter\_Configs$
    if ($n_r \neq n$) & (No. of elements in $Inter\_Configs > X$) **then**
      $Inter\_Configs$ = INTERMEDIATE_PRUNING(n_r, X, $Inter\_Configs$)
    end if
  end if
  return $Inter\_Configs$
end function

Algorithm 2 Pseudo-code for intermediate pruning.

function INTERMEDIATE_PRUNING(n_r, X, $Inter\_Configs$)
  $PO\_Configs \leftarrow Empty$
  if $Inter\_Configs \neq Empty$ **then**
    $Max\_Nominal\_Output = (2^{n_r} - 1)^2$
    Generate $Config\_set_1$ from $Inter\_Configs$ (see Table 7)
    Generate $Config\_set_2$ from $Inter\_Configs$ (see Table 7)
    Generate $Config\_set_3$ from $Inter\_Configs$ (see Table 7)
    Generate $Config\_set_4$ from $Inter\_Configs$ (see Table 7)
    $Configs_1$ = Pareto-optimal configurations from $Config\_set_1$ based on cost vs. absolute mean error
    $Configs_2$ = Pareto-optimal configurations from $Config\_set_2$ based on cost vs. absolute mean error
    $Temp\_Configs = \{Configs_1, Configs_2\}$
    if No. of configurations in $Temp\_Configs >$ floor(X*0.25) **then**
      $PO\_Configs$ = CONFIGURATION_SELECTION($Temp\_Configs$, floor(X*0.25))
    else
      $PO\_Configs$ = $Temp\_Configs$
    end if
    while ($\{Config\_set_3, Config\_set_4\} \neq Empty$) & ($X >$ no. of configurations in $PO\_Configs$) **do**
      $Configs_3$ = Pareto-optimal configurations from $Config\_set_3$ based on cost vs. absolute mean error
      $Configs_4$ = Pareto-optimal configurations from $Config\_set_4$ based on cost vs. absolute mean error
      $Temp\_Configs = \{Configs_3, Configs_4\}$
      if No. of configurations in $Temp\_Configs > X$ - No. of configurations in $PO\_Configs$ **then**
        $Temp\_Configs$ = CONFIGURATION_SELECTION($Temp\_Configs$, X - no. of configurations in $PO\_Configs$)
        Add $Temp\_Configs$ to $PO\_Configs$
      else
        Add $Temp\_Configs$ to $PO\_Configs$
        Remove configurations in $Temp\_Configs$ from $Config\_set_3$ and $Config\_set_4$
      end if
    end while
  end if
  return $PO\_Configs$
end function

function CONFIGURATION_SELECTION($Inter\_Configs$, N)
  $O\_Configs \leftarrow Empty$
  if $N == 1$ **then**
    $O\_Configs \leftarrow$ Configuration from $Inter\_Configs$ having minimum absolute mean error
  else
    $O\_Configs \leftarrow$ Configurations from $Inter\_Configs$ having minimum and maximum absolute mean error
    if $N > 2$ **then**
      Remove the configurations in $O\_Configs$ from $Inter\_Configs$
      $Loss = Infinite$
      $Clusters = Empty$
      for i = 1 : M **do**
        $\{Clusters_{new}, Loss_{new}\}$ = Use k-means to find $N - 2$ clusters of configurations in $O\_Configs$
        if $Loss_{new} < Loss$ **then**
          $Clusters = Clusters_{new}$
          $Loss = Loss_{new}$
        end if
      end for
      $O\_Configs$ = Configurations closest to the cluster centroids in $Clusters$
    end if
  end if
  return $O\_Configs$
end function

[2] Q. Xu, M. Todd, and S.K. Nam, "Approximate computing: A survey," IEEE Design & Test, vol. 33, no. 1, pp. 8–22, 2016.
[3] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard, "Quality of service profiling," in Proc. of the 32nd International Conference on Software Engineering ACM/IEEE, Cape Town, South Africa, 2010, pp. 25-34.
[4] A.K. Mishra, R. Barik, and S. Paul, "iACT: A software-hardware framework for understanding the scope of approximate computing," in Workshop on Approximate Computing Across the System Stack (WACAS), Utah, USA, 2014.
[5] P. Roy, J. Wang and W.F. Wong, "PAC: program analysis for approximation-aware compilation," in Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems IEEE, Amsterdam, Netherlands, 2015, pp. 69-78.
[6] G.A. Gillani and A.B.J. Kokkeler, "Improving error resilience analysis methodology of iterative workloads for approximate computing," in Proc. of the Computing Frontiers Conference ACM, Italy, 2017, pp. 374–379.
[7] M.A. Hanif, R. Hafiz, and M. Shafique, "Error resilience analysis for systematically employing approximate computing in convolutional neural networks.," in Proc. of the DATE IEEE Dresden, Germany, 2018, pp. 913-916.
[8] V.K. Chippa, S.T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate

computing," in Proc. of the 50th Annual DAC ACM/EDAC/IEEE, USA, 2013, pp. 113.

[9] S. Mittal, "A survey of techniques for approximate computing," ACM Computing Surveys (CSUR), vol. 48, no. 4, pp. 62, 2016.

[10] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Invited: Cross-layer approximate computing: From logic to architectures," in Proc. of the 53rd Annual DAC ACM/EDAC/IEEE, Austin, TX, USA, 2016.

[11] P. Stanley-Marbell et al., "Exploiting Errors for Efficiency: A Survey from Circuits to Algorithms," arXiv preprint, arXiv:1809.05859 (2018).

[12] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han, "A review, classification, and comparative evaluation of approximate arithmetic circuits," ACM Journal on Emerging Technologies in Computing Systems (JETC), vol. 13, no. 4, article no. 60, 2017.

[13] E. Nogues, D. Menard, and M. Pelcat, "Algorithmic-level approximate computing applied to energy efficient hevc decoding," IEEE Transactions on Emerging Topics in Computing, 2016, DOI: 10.1109/TETC.2016.2593644.

[14] G.A. Gillani, M.A. Hanif, M. Krone, S.H. Gerez, M. Shafique, and A.B.J. Kokkeler, "SquASH: Approximate Square-Accumulate with Self-Healing," IEEE Access, DOI 10.1109/ACCESS.2018.2868036, vol. 6, pp. 49112-49128, 2018.

[15] S. Mazahir, O. Hasan and M. Shafique, "Adaptive Approximate Computing in Arithmetic Datapaths," IEEE Design & Test, 35(4), 2018.

[16] A. Verma, P. Brisk, and P. Ienne, "Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design," in Proc. of the DATE, Munich, Germany, 2008, pp. 1250–1255.

[17] A. Kahng and S. Kang, "Accuracy-Configurable Adder for Approximate Arithmetic Designs," in Proc. of the 49th Annual DAC ACM/EDAC/IEEE, San Francisco, CA, USA, 2012, pp. 820–825.

[18] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-Power Digital Signal Processing using Approximate Adders," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 32, no. 1, pp. 124–137, 2013.

[19] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A Low Latency Generic Accuracy Configurable Adder," in Proc. of the 52nd Annual DAC ACM/EDAC/IEEE, San Francisco, CA, USA, 2015.

[20] M.K. Ayub, O. Hasan and M. Shafique, "Statistical error analysis for low power approximate adders," in Proc. of the 54th Annual DAC ACM/EDAC/IEEE, Austin, TX, USA, 2017, pp. 1–6.

[21] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique and J. Henkel, "Probabilistic Error Modeling for Approximate Adders," IEEE Transactions on Computers, vol. 66, no. 3, pp. 515–530, 2017.

[22] B.S. Prabakaran, S. Rehman, M.A. Hanif, S. Ullah, G. Mazaheri, A. Kumar, and M. Shafique, "DeMAS: An efficient design methodology for building approximate adders for FPGA-based systems," in Proc. of the DATE, Dresden, Germany, 2018, pp. 917-920.

[23] P. Kulkarni, P. Gupta, M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in Proc. of the VLSI Design IEEE, Chennai, India, 2011, pp. 346–351.

[24] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel, "Architectural-space exploration of approximate multipliers," in Proc. of the ICCAD, Austin, TX, USA, 2016, pp. 1–8.

[25] S. Mazahir, O. Hasan, R. Hafiz, and M. Shafique, "Probabilistic Error Analysis of Approximate Recursive Multipliers," IEEE Transactions on Computers, vol. 66, no. 11, pp. 1982-1990, Nov. 1 2017.

[26] S. Hashemi et al., "DRUM: A Dynamic Range Unbiased Multiplier for Approximate Applications," in Proc. of the ICCAD, IEEE/ACM, 2015.

[27] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," IEEE Transactions on Computers, vol. 64, no. 4, pp. 984–994, Apr. 2015.

[28] H. Jiang, J. Han, F. Qiao, and F. Lombardi, "Approximate radix-8 booth multipliers for low-power and high-performance operation," IEEE Transactions on Computers, vol. 65, no. 8, pp. 2638–2644, Aug. 2016.

[29] V. Mrazek, Z. Vasicek, L. Sekanina, H. Jiang, and J. Han, "Scalable Construction of Approximate Multipliers With Formally Guaranteed Worst Case Error," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26, no. 11, pp. 2572–2576, Nov. 2018.

[30] S. Ullah, S. Rehman, B.S. Prabakaran, F. Kriebel, M.A. Hanif, M. Shafique, and A. Kumar, "Area-optimized low-latency approximate multipliers for FPGA-based hardware accelerators," in Proc. of the 55th Annual DAC ACM/EDAC/IEEE, San Francisco, CA, USA, 2018, pp. 159.

[31] V. Leon, G. Zervakis, D. Soudris, and K. Pekmestzi, "Approximate hybrid high radix encoding for energy-efficient inexact multipliers," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26, pp. 421-430, March 2018.

[32] S.R. Kuang, and J.P. Wang, "Low-error configurable truncated multipliers for multiply-accumulate applications," Electronics Letters, vol. 42, no. 16, pp. 904–905, 2006.

[33] N. Petra, D. De Caro, V. Garofalo, E. Napoli, and A.G. Strollo, "Truncated binary multipliers with variable correction and minimum mean square error," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 57, no. 6, pp. 1312–1325, 2010.

[34] S. Dutt, A. Chauhan, R. Bhadoriya, S. Nandi, and G. Trivedi, "A high-performance energy-efficient hybrid redundant MAC for error-resilient applications," in Proc. of VLSI Design IEEE, Bangalore, India, 2015, pp. 351–356.

[35] D. Esposito, A.G. Strollo, and M. Alioto, "Low-power approximate MAC unit," in Proc. of the PRIME, Giardini Naxos, Italy, 2017, pp. 81–84.

[36] A.K. Jain, "Data clustering: 50 years beyond K-means," Pattern Recognition Letters, 31(8), pp. 651-666, 2010.

[37] S. Salvini, and S.J. Wijnholds, "Fast gain calibration in radio astronomy using alternating direction implicit methods: Analysis and applications," Astronomy & Astrophysics, vol. 571, pp. A97, 2014.

[38] R.C. Gonzalez and R.E. Woods, "Digital Image Processing," Prentice-Hall, 3rd ed. Englewood Cliffs, NJ, USA, 2008.

[39] M.L. Schmatz, R. Jongerius, G. Dittmann, A. Anghel, T. Engbersen, J. van Lunteren, and P. Buchmann, "Scalable, efficient ASICS for the square kilometre array: From A/D conversion to central correlation," in Proc. of Acoustics, Speech and Signal Processing (ICASSP) IEEE, Florence, Italy, 2014, pp. 7505-7509.

[40] V.T. Lee, A. Alaghi, R. Pamula, V.S. Sathe, L. Ceze, and M. Oskin, "Architecture Considerations for Stochastic Computing Accelerators," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 11, pp. 2277-2289, 2018.

[41] D. Bankman and B. Murmann, "An 8-bit, 16 input, 3.2 pJ/op switched-capacitor dot product circuit in 28-nm FDSOI CMOS," in Proc. of the Asian Solid-State Circuits Conference (A-SSCC) IEEE, Toyama, Japan, 2016, pp. 21-24.

[42] N.P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in Proc. of the 44th Annual International Symposium on Computer Architecture (ISCA) ACM/IEEE, Toronto, ON, Canada, 2017, pp. 1-12.

G.A. GILLANI received the B.Sc. degree in electrical engineering (with honors) from University of Engineering and Technology (UET), Taxila, Pakistan, and the Master of Engineering (M.Eng.) degree in computer science and technology from Northwestern Polytechnical University, Xián, China. He is currently pursuing the Ph.D. degree in the faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) at the University of Twente, The Netherlands.

His research interests include energy-efficient computing, low-power signal processing, machine learning and the non-conventional computing paradigms like near-memory computing, brain-inspired computing, and approximate computing. He is also a recipient of the Chinese Govt. Scholarship for M.Eng. studies.

MUHAMMAD ABDULLAH HANIF is a University Assistant at the Institute of Computer Engineering, Department of Informatics, Vienna University of Technology (TU Wien), Austria. He currently holds a M.Sc. degree in electrical engineering with specialization in Digital Systems and Signal Processing (DSSP) from School of Electrical Engineering and Computer Science (SEECS), National University of Sciences and Technology (NUST), Islamabad, Pakistan, and a B.Sc. degree in electronic engineering from GIKI, Pakistan.

His research interests are in brain-inspired computing, machine learning, approximate computing, computer architecture, energy-efficient design, robust computing, system-on-chip design, and emerging technologies. In the past, he has also worked as a Research Associate in Vision Processing (VISpro) lab, Information Technology University (ITU), Pakistan, and as a Lab Engineer in Ghulam Ishaq Khan Institute of Engineering Sciences and Technology (GIKI), Pakistan. He is also a recipient of President's Gold Medal for his outstanding academic performance during his MS degree.

B. VERSTOEP received the B.Sc. degree in electrical engineering from University of Twente, Enschede, The Netherlands, in 2018. He did his B.Sc. research assignment at Computer Architecture for Embedded Systems (CAES) group in the faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS).

His research interests are in the field of electronics design, more specifically FPGA design optimization.

S.H. GEREZ received the M.Sc. degree (with honors) in electrical engineering in 1984 and the Ph.D. degree in applied sciences in 1989, both from the University of Twente, The Netherlands. He has been an assistant professor at the University of Twente since 1990 (part-time starting from 2001), focusing on research and education in the fields of implementation of digital signal processing, digital integrated-circuit design, and design automation.

He is the author of the book Algorithms for VLSI Design Automation (Wiley, 1998). From 2001 to 2009, he was employed by the Cordless Telephony Division of National Semiconductor (called Sitel Semiconductor after 2005 and currently part of Dialog Semiconductor). Since 2009 he runs his business Bibix which offers consultancy services in his mentioned fields of interest.

MUHAMMAD SHAFIQUE (M'11–SM'16) is a full professor at the Institute of Computer Engineering, Department of Informatics, Vienna University of Technology (TU Wien), Austria. He is directing the Group on Computer Architecture and Robust, Energy-Efficient Technologies (CARE-Tech). He received his Ph.D. in computer science from Karlsruhe Institute of Technology (KIT), Germany in January 2011. Before, he was with Streaming Networks Pvt. Ltd. where he was involved in research and development of advanced video coding systems for several years.

His research interests are in computer architecture, power- and energy-efficient systems, robust computing covering various aspects of dependability and fault-tolerance, hardware security, emerging computing trends like neuromorphic and approximate computing, neurosciences, emerging technologies and nanosystems, self-learning and intelligent/cognitive systems, FPGAs, MPSoCs, and embedded systems. His research has a special focus on cross-layer analysis, modeling, design, and optimization of computing and memory systems covering various layers of the hardware and software stacks, as well as their integration in application use cases from Internet-of-Things (IoT), Cyber-Physical Systems (CPS), and ICT for Development (ICT4D) domains.

Dr. Shafique received the prestigious 2015 ACM/SIGDA Outstanding New Faculty Award, six gold medals in educational career, and several best paper awards and nominations at prestigious conferences like DATE, DAC, ICCAD and CODES+ISSS, Best Master Thesis Award, and Best Lecturer Award. He has given several Invited Talks, Tutorials, and Keynotes. He has also organized many special sessions at premier venues (like DAC, ICCAD, DATE, IOLTS, and ESWeek) and served as the Guest Editor for IEEE Design and Test Magazine (D&T) and IEEE Transactions on Sustainable Computing (T-SUSC). He has served as the TPC co-Chair of ESTIMedia and LPDC, General Chair of ESTIMedia, and Track Chair at DATE and FDL. He has served on the program committees of several IEEE/ACM conferences like ICCAD, ISCA, DATE, CASES, FPL, and ASPDAC. He is a senior member of the IEEE and IEEE Signal Processing Society (SPS), and a member of ACM, SIGARCH, SIGDA, SIGBED, and HiPEAC. He holds one US patent and over 200 papers in premier journals and conferences.

A.B.J. KOKKELER has worked more than 6 years at Ericsson as a system engineer and 8 years at the Netherlands foundation for research in astronomy (ASTRON) as a scientific project manager. In 2003 he joined the University of Twente where he currently is appointed as Associate Professor.

He has a background in telecommunication, mixed signal design and signal processing. Currently, his main interest lies in the area of the design of low-power architectures for telecommunications and computationally intensive applications. He is involved in research projects, sponsored by the Dutch and European governments and industry.