Title

Array-based approximate arithmetic computing: A general model and applications to multiplier and squarer design

Authors

Shao, Botang

Li, Peng

Publication Date

2015

Peer reviewed

Array-Based Approximate Arithmetic Computing: A General Model and Applications to Multiplier and Squarer Design

Botang Shao and Peng Li, *Senior Member, IEEE*

***Abstract*—We propose a general model for array-based approximate arithmetic computing (AAAC) to guide the minimization of processing error. As part of this model, the Error Compensation Unit (ECU) is identified as a key building block for a wide range of AAAC circuits. We develop theoretical analysis geared towards addressing two critical design problems of the ECU, namely, determination of optimal error compensation values and identification of the optimal error compensation scheme. We demonstrate how this general AAAC model can be leveraged to derive practical design insights that lead to optimal tradeoffs between accuracy, energy dissipation and area overhead. To further minimize energy consumption, delay and area of AAAC circuits, we perform ECU design simplification by introducing logic don't cares. By applying this model and using a commercial 90 nm CMOS standard cell library, we propose an approximate 16 $\times$ 16 fixed-width Booth multiplier that consumes 44.85% and 28.33% less energy and area compared with theoretically the most accurate fixed-width Booth multiplier. Furthermore, it reduces average error, max error and mean squared error by 11.11%, 28.11%, and 25.00%, respectively, when compared with the most accurate reported approximate Booth multiplier and outperforms the same design significantly by 19.10% for the energy-delay-mean squared error product. Using the same approach, significant energy consumption, area and error reduction is achieved for a squarer unit. To further reduce error and cost by utilizing extra signatures and don't cares, we demonstrate a 16-bit fixed-width squarer that improves the energy-delay-max error product by 15.81%.**

***Index Terms*—Approximate arithmetic computing, multiplier, squarer.**

I. Introduction

As THE CMOS technology and VLSI design complexity scale, delivering desired functionalities while managing chip power consumption has become a first-class design challenge. To remedy this grand energy-efficiency challenge, array-based approximate arithmetic computing (AAAC) has been introduced as a promising solution to applications with inherent error resilience including media processing, machine learning, and neuromorphic systems. AAAC may allow one to trade off accuracy for significant reduction of energy consumption for such error tolerant applications.

Manuscript received August 29, 2014; revised November 19, 2014; accepted December 07, 2014. Date of current version March 27, 2015. This paper was recommended by Associate Editor F. Clermidy.

B. Shao was with the Department of Electrical and Computer Engineering, Texas A&M University, TX 77843-3128 USA, and is now with Freescale Semiconductor, Austin, TX 78735 USA (e-mail: jackieshao2011@gmail.com).

P. Li is with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: pli@tamu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2015.2388839

To this end, approximate multipliers and squarers have been a focus of a great deal of past and ongoing work. Two types of approximate multipliers exist: approximate AND-array multipliers, which utilize AND gates for partial product generation and approximate Booth multipliers, which use the modified Booth algorithm to reduce the number of partial products. Constant correction [1] and variable correction [2] schemes are proposed for approximate AND-array multipliers. However, since Booth multipliers are much more efficient than AND-array multipliers, approximate Booth multipliers have been intensively investigated [3]–[10]. In particular, statistical linear regression analysis [3], estimation threshold calculation [4], and self-compensation approach [5] have been utilized to compensate the truncation error. Accuracy of approximate multiplier designs is increased by using certain outputs from Booth encoders [6]–[9]. To decrease energy consumption, a probabilistic estimation bias (PEB) scheme [10] is presented. A series of approximate squarers have been proposed [11]–[13]. For instance, the designs of [11] and [12] compensate truncation error by utilizing constant and variable correction scheme, respectively. A LUT-based squarer [13] is proposed by employing a hybrid LUT-based structure.

While a diverse set of array-based approximate arithmetic unit designs exist, what is currently lacking is systemic design guidance that allows one to optimally trade off between error, area and energy. While the area and energy of a given design can often be easily reasoned or estimated, getting insights on error and thereby providing a basis for optimally trading off between error, area, and energy consumption appears to be challenging and not well understood.

To this end, the main contributions of this paper are two-fold.[1] First, we propose a general AAAC model for reasoning about different ways of controlling approximation errors and present optimal error compensation schemes under ideal design scenarios. The proposed model is general in the sense that it captures the key design structure that is common to a major class of array-based approximate arithmetic units (e.g., multipliers, squarers, dividers, adders/subtractors, and logarithmic function units). The proposed model offers critical insights of optimized error compensation schemes and the corresponding input signature generation logic that is the key to error compensation.

Compared to the preliminary work [14], we demonstrate detailed illustrations and sketches of proof for Theorems 1–4 in Section III and propose a method of further error and cost reduction for AAAC circuit designs by introducing extra signa-

[1] A preliminary version of this work appears in [14].

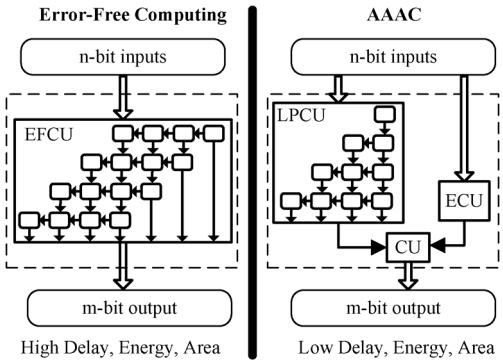


Fig. 1. The proposed AAAC model.

tures and don't cares, making our AAAC model more general and complete. Second, as two specific applications, by leveraging the design insights obtained from the proposed model, we present a new approximate Booth multiplier and squarer design that achieve noticeable reduction of error compared with existing designs while maintaining significant benefits in terms of delay, area and energy consumption due to the approximate nature of computation. Compared to the preliminary work [14], error and cost for the proposed 16-bit fixed-width squarer are further reduced by introducing extra signatures and don't cares.

When implemented using a commercial 90 nm CMOS standard cell library, the proposed approximate 16 × 16 fixed-width Booth multiplier consumes 44.85% and 28.33% less energy and area compared with theoretically the most accurate fixed-width Booth multiplier. Furthermore, it reduces average error, max error and mean square by 11.11%, 28.11%, and 25.00%, respectively, when compared with the most accurate reported approximate Booth multiplier and outperforms the same design significantly by 19.10% in terms of the energy-delay-mean squared error product (EDE_{ms}). For our approximate 16-bit fixed-width squarer, a 18.18%, 21.67%, and 31.25% reduction is achieved on average error, max error, and mean square, respectively, and more than 20.00% EDE_{ms} reduction is achieved, when compared with existing designs. By utilizing the method of introducing extra signatures and don't cares, the energy-delay-max error product (EDE_{max}) is further reduced by 15.81% for our proposed 16-bit fixed-width squarer. Additionally, when operated in the full-width mode, our multiplier and squarer have an even greater improvement of accuracy.

II. AAAC Model

Fig. 1 contrasts an error-free computing unit (EFCU) with n-bit inputs and an m-bit output (left) with its approximate counterpart modeled using the proposed AAAC model (right). The AAAC model consists of three units: low-precision computing unit (LPCU), error compensation unit (ECU), and combine unit (CU).

The LPCU in the AAAC circuit produces a low-precision approximate output, for example, based upon truncation or a fraction of the input bits, with lowered energy, delay, and/or area overheads compared with the error-free EFCU. To reduce the error produced by the LPCU, a *low-cost* ECU may be included for error compensation. Finally, the CU combines the error compensation produced by the ECU with the result outputted by the LPCU, generating the final output of the AAAC unit with reduced approximate error.

The *generality* of the AAAC model lies in the fact it reflects the key computing principles behind a wide range of array-based arithmetic units, for example, approximate adders [15]–[17], approximate multipliers [1]–[10], and approximate squarers [11]–[13]. For instance, many approximate adders employ carry prediction from low input bits, which can be thought as a particular way of implementing the ECU. Similarly, error compensation is a common scheme in approximate multipliers and squarers.

Clearly, the **key AAAC design problem** is to develop an efficient LPCU and, in particular, an ECU so as to significantly reduce energy, delay, and/or area overhead while achieving a low degree of approximation error. While the area and energy of a given design can often be easily reasoned or estimated, the key challenge is to develop insights on error or error distribution so as to optimize the error compensation scheme, which we focus on in the following sections.

A. Error Metrics

We evaluate a given AAAC design with n-bit inputs by defining average error E_{ave}, maximum error E_{max} and mean squared error E_{ms}, respectively as

$$E_{ave} = \frac{1}{2^n \cdot N} \sum_{i=1}^{N} |O_{AAAC,i} - O_{EFCU,i}| \tag{1}$$

$$E_{max} = \frac{1}{2^n} \underset{i}{max} |O_{AAAC,i} - O_{EFCU,i}| \tag{2}$$

$$E_{ms} = \frac{1}{2^{2n} \cdot N} \sum_{i=1}^{N} (O_{AAAC,i} - O_{EFCU,i})^2 \tag{3}$$

where N, $O_{AAAC,i}$ and $O_{EFCU,i}$ denote the number of all possible input combinations, output of the AAAC, and output of EFCU (error-free result), respectively, for each input combination i. Note that the above error metrics are normalized with respect to the range of the output 2^{2n}. As shown in Fig. 1, for each input combination i, the ECU outputs error compensation, denoted by $Comp_i$. Hence the output of the AAAC circuit is: $O_{AAAC,i} = O_{LPCU,i} + Comp_i$, where $O_{LPCU,i}$ is the output of the LPCU. Importantly, the error of the LPCU, i.e., the error of the AAAC before compensation ($E_{BC,i}$) and after compensation ($E_{AC,i}$) is given simply by

$$E_{BC,i} = O_{EFCU,i} - O_{LPCU,i} \tag{4}$$

$$E_{AC,i} = |E_{BC,i} - Comp_i| \tag{5}$$

B. Model of Error Compensation Unit (ECU)

Ideally, a specific $Comp_i$ can be computed by the ECU to perfectly zero out the error for each input pattern i. However, this does not serve any purpose for approximate computing as we are essentially re-implementing the error-free operation. We present a practical yet general ECU model, which consists of a *Signature Generator* and a *K-to-1 Mux* as shown in Fig. 2(a). The process of error compensation is shown in Fig. 2(b). Conceptually, for a given input pattern i, the signature generator produces several signatures that encode certain essential information about the inputs. Based on the actual values of the extracted input signatures, this input pattern is classified into one of the K predetermined input classes with each having a predetermined

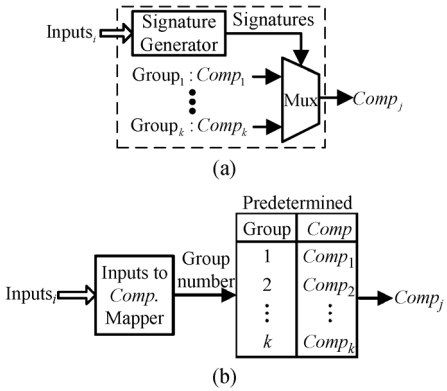


Fig. 2. ECU model: (a) ECU blocks, (b) error compensation process.

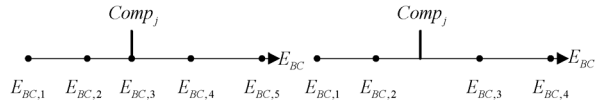


Fig. 3. Optimal E_{ave} -minimizing error compensation value: (left) N_{G_j} odd, (right) N_{G_j} even.

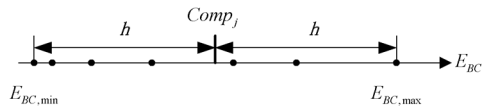


Fig. 4. Optimal E_{max}-minimizing error compensation value.

error compensation $Comp_j$ $(j = 1, 2, \ldots, K)$. The compensation for this input pattern is produced by using the signature values to select the constant compensation of its corresponding input group via the *K-to-1 mux*.

It is important to note that the structure of the ECU model may not immediately correspond to the specific logic implementation of the ECU. Nevertheless, it captures the general working principle of error compensation for AAAC.

III. Optimal Error Compensation and ECU Design

In the extreme case, if each input group has only one input pattern, then the optimal compensation for each group/input would be simply the corresponding $E_{BC,i}$ (4). However, in practical cases, we need to consider the $E_{BC,i}$ distribution within each group.

A. Optimal Error Compensation

Now it is evident that the **key ECU design problem** is to find an optimal signature generation scheme that minimizes one or more error metrics (i.e., E_{ave}, E_{max}, and E_{ms}) under a given set of cost constraints (e.g., area, delay, and energy). Note that the cost of the ECU often strongly correlates with the number of input groups K. We show several provable results for optimal selection of error compensation constants for a given compensation scheme. We also show an optimal error compensation scheme under an ideal scenario. We first denote the number of input patterns that fall in the j^{th} group by N_{G_j}.

Theorem 1: The optimal error compensation $Comp_j$ for the j^{th} group that minimizes E_{ave} is the median of $E_{BC,i}$ (4) of the group if N_{G_j} is odd; otherwise it can be any value that falls in the inclusive interval between the two medians of $E_{BC,i}$.

The above results are illustrated in the example of Fig. 3. For the j^{th} group, minimizing E_{ave} leads to minimization of the sum of distances from each $E_{BC,i}$ to $Comp_j$, which makes the value of $Comp_j$ the median of $E_{BC,i}$ of the group if N_{G_j} is odd. On the other hand, when N_{G_j} is even, $Comp_j$ can be any value that falls in the inclusive interval between the two medians of $E_{BC,i}$.

Theorem 2: The optimal error compensation $Comp_j$ for the j^{th} group that minimizes E_{max} is the mean of $E_{BC,min}$ and $E_{BC,max}$, where $E_{BC,min}$ and $E_{BC,max}$ are the minimum and maximum values of $E_{BC,i}$ in the group, respectively.

This result is illustrated by the example in Fig. 4, in which $h = (E_{BC,max} - E_{BC,min})/2$ is the minimum E_{max} value that can be achieved when $Comp_j$ is the mean of $E_{BC,min}$ and $E_{BC,max}$. Otherwise, either $(Comp_j - E_{BC,min})$ or $(E_{BC,max} - Comp_j)$ is greater than h.

Theorem 3: The optimal error compensation $Comp_j$ for the j^{th} group that minimizes E_{ms} is the mean of all $E_{BC,i}$ in this group.

To see how Theorem 3 may be proven, consider the j^{th} group, for which we have

$$E_{ms} = \sum_{i=1}^{N_{G_j}} (E_{BC,i} - Comp_j)^2 / N_{G_j}. \tag{6}$$

To minimize E_{ms}, let $\partial E_{ms}/\partial Comp_j = 0$, we have

$$Comp_j = \sum_{i=1}^{N_{G_j}} E_{BC,i} / N_{G_j}. \tag{7}$$

Equation (7) indicates that to minimize E_{ms} for one group, the best compensation is the average of all $E_{BC,i}$ in this group.

The above three theorems suggest the following important design guidance. For a given compensation scheme, the compensation $Comp_j$ for each input group can be optimally determined according to the results above to minimize the targeted error metric. Now we turn into the other design problem by presenting the optimal error compensation scheme under an ideal scenario.

Theorem 4: Assume $E_{BC,i}$ is uniformly and continuously distributed from $\overline{E}_{BC,min}$ to $\overline{E}_{BC,max}$, where $\overline{E}_{BC,min}$ and $\overline{E}_{BC,max}$ are the minimum and maximum values of $E_{BC,i}$, in the entire input range, then the optimal E_{ms}-minimizing error compensation scheme with K input groups partitions the entire $E_{BC,i}$ range into K non-overlapping equal-length intervals with one interval corresponding to a specific input group.

To provide an intuitive sketch of proof for Theorem 4 without loss of generality, let us consider a 4-group example in Fig. 5. First, it is not hard to see that with the number of input groups fixed, partitioning the error distribution into non-overlapping groups would not increase the overall E_{ms}. Now assume DP_1 and DP_5 are the lower and upper bounds of $E_{BC,i}$ and fixed for all inputs and the corresponding lower and upper bounds for the

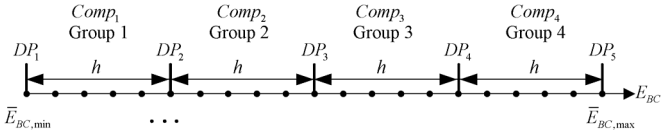


Fig. 5. Optimal E_{ms}-minimizing error compensation scheme.

four input groups are DP_1 and DP_2, DP_2 and DP_3, DP_3 and DP_4, DP_4 and DP_5.E_{ms} can be written as

$$E_{ms} = \left[\int_{DP_1}^{DP_2} (E_{BC,i} - Comp_j)^2 dE_{BC,i} + \int_{DP_2}^{DP_3} (E_{BC,i} - Comp_j)^2 dE_{BC,i} + \int_{DP_3}^{DP_4} (E_{BC,i} - Comp_j)^2 dE_{BC,i} + \int_{DP_4}^{DP_5} (E_{BC,i} - Comp_j)^2 dE_{BC,i} \right] / (DP_5 - DP_1)$$

where to minimize E_{ms}, according to Theorem 3, the optimal compensation for the j^{th} group is given by $Comp_j = (DP_j + DP_{j+1})/2$. Then, let $\partial E_{ms}/\partial DP_j = 0$ ($j = 2,\ 3,\ 4$), we have

$$DP_3 - DP_2 = DP_2 - DP_1$$
$$DP_4 - DP_3 = DP_3 - DP_2$$
$$DP_5 - DP_4 = DP_4 - DP_3$$

Therefore,

$$DP_2 - DP_1 = DP_3 - DP_2 = DP_4 - DP_3 = DP_5 - DP_4 \quad (8)$$

Equation (8) indicates that the four input groups are non-overlapping and in equal length with one compensation for each input group. Note that $E_{BC,i}$ is discrete and hence not continuously distributed in reality. This continuous assumption is a good approximation when the error is densely populated between $\overline{E}_{BC,min}$ and $\overline{E}_{BC,max}$.

B. Further Error and Cost Reduction

In practice, the above theoretical results can be used to come up a good error compensation scheme and the corresponding optimal compensation value for each input group while considering the logic implementation complexity. For a given application, this process may help us identify a highly compact set of signatures. With a good initial set of signatures chosen, to further reduce error, one effective way is to add extra signatures by directly considering certain input bits. Such signatures can further divide the K predetermined input classes into a larger number, say M, of smaller groups.

On the other hand, it is also important to minimize hardware implementation cost. For this, introducing don't care terms is a general approach to the reduction of logic complexity [18]. In logic synthesis, don't cares can be expressed using special non-Boolean values, such as "x" [19]. When having the design synthesized by synthesis tools such as Synopsys Design Compiler [20], we set constraints of minimizing power and area, so an optimal logic will be generated.

For the case of ECU design, we propose to set the compensation values of a subset of input groups to don't cares in order to reduce logic complexity. The issue now becomes how to determine which groups should be given a high priority to be set as don't cares. We rank all the groups based on their impact on the target error metric when they are set to be don't cares and adopt the following process to systematically do so. As shown in Algorithm 1 that targets E_{max}, assume that there are M groups and the compensation value has p bits. $E_{i,j}$ represents error before compensation of the i^{th} element in the j^{th} group. When targeting to minimize E_{max}, each j^{th} group is evaluated by considering the worst case $E_{max,j}$ ($\overline{E_{max,j}}$) when the compensation $Comp_j$ is set to be a don't care. $\overline{E_{max,j}}$ is computed to be the maximum $E_{max,j}$ when $Comp_j$ varies from 0 to $2^p - 1$. Then, the groups which have the smallest $\overline{E_{max,j}}$ are set as don't cares, while other groups are implemented precisely because they have a comparatively greater impact on E_{max}. In practice, the more don't cares we set, the less energy consumption and area the design can be achieved with the use of a logic synthesis tool such as Synopsys Design Compiler [20], but the greater the error. Therefore, the number of don't care groups should be chosen by jointly considering the specifications of energy, delay, area, and accuracy.

Algorithm 1 Don't care introduction while minimizing E_{max}

for $j = 1$ to M **do**
  $\overline{E_{max,j}} = 0$;
  for $Comp_j = 0$ to $2^p - 1$
    compute $E_{max,j} = \max_i |E_{i,j} - Comp_j|$;
    if $\overline{E_{max,j}} < E_{max,j}$
    $\overline{E_{max,j}} = E_{max,j}$; **then**
    end if
  end for
  end for
Set don't cares to a number of groups with the smallest $\overline{E_{max,j}}$;

To minimize E_{ave}, a process similar to Algorithm 1, where the j^{th} group is ranked by the worst case $E_{ave,sum,j}$ ($\overline{E_{ave,sum,j}}$), which is defined as the maximum value of $E_{ave,sum,j} = \sum_i |E_{i,j} - Comp_j|$, when $Comp_j$ ranges from 0 to $2^p - 1$. To minimize E_{ms}, we rank the groups by the worst case $E_{ms,sum,j}$ ($\overline{E_{ms,sum,j}}$), which is defined as the maximum value of $E_{ms,sum,j} = \sum_i (E_{i,j} - Comp_j)^2$, when $Comp_j$ ranges from 0 to $2^p - 1$.

C. Practical ECU Design Guidance

The above theoretical analysis provides optimal design strategies for minimizing a particular error metric. In practice, minimization of one error metric may often lead to near-optimal minimization of other error metrics. We summarize the practical ECU design guidance that is directly resulted from these results:

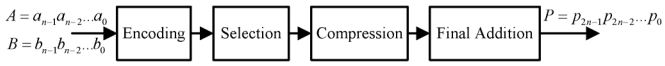


Fig. 6. Error-free booth multiplier blocks.

1) Hardware cost can be controlled by targeting an appropriate number of input groups;
2) Different input groups shall have no or little overlap on the E_{BC} axis to minimize approximation error;
3) The $E_{BC,i}$ spread of each group shall be largely of equal length;
4) Non-uniformity of $E_{BC,i}$ spread may be reduced by splitting groups with a large spread into smaller groups;
5) For a given compensation/grouping scheme, the optimal compensation values for all groups can be determined to minimize a given error metric according to Theorems 1–3;
6) Additional accuracy improvement may be achieved by directly introducing a subset of inputs as extra signatures to further divide large input groups. Energy and area overhead can be reduced by setting the compensation values of the groups that have a comparatively small impact on the overall error to don't cares.

IV. Proposed Multiplier Design

The AAAC model is applied to approximate fixed-width and full-width Booth multiplier designs.

A. Fundamentals of Booth Multipliers

Booth multipliers are ideal for high speed applications and the Radix-4 Modified Booth multipliers are most widely applied [21], [22]. The main blocks of a Radix-4 Modified Booth multiplier are shown in Fig. 6. The encoding block applies the Radix-4 Booth Algorithm to encode the multiplier B, allowing the selection block to generate only half number of partial products needed for array multipliers with each product being one of the following: 0, A, 2A, -A, -2A. Then, the compressors (such as 2:2, 3:2 [23], and 4:2 [24] compressors) in the compression block compress the number of partial products to two [23], [24]. Finally, a $2n$-bit adder is used to generate the final product.

B. The Basic Idea

In this paper, we use nxn fixed-width Booth multipliers to refer to approximate Booth multipliers that operate on two n-bit inputs while outputting only an n-bit product [4]. For convenience of discussion, we assume the higher and lower n bits of the multiplicand and multiplier correspond to the integer and fractional parts of the inputs, respectively. In this regard, a fixed-width multiplier outputs, possibly in an approximate manner, the n-bit integer part of the exact product.

Fig. 7 shows the full 8-partial product array for a full-precision 16 × 16 Booth multiplier where each dot row (PP_0 to PP_7) is a partial product. The 16 dots (bits) in each PP_i are denoted by $pp_{i,15}pp_{i,14}\ldots pp_{i,0}$ from left to right, c_i is the correction constant required to generate the negative partial product, and s_i is sign of the i^{th} partial product. The vertical dashed line splits the array at the position of the binary (radix) point. A fixed-width multiplier outputs an integer output by approximating the carry-out produced by the fractional part of the array, which is also labeled as the *truncation part (TP)*. On the other hand, the contribution of the bits left of the binary point, i.e., ones in the *accurate part (AP)*, is not approximated.

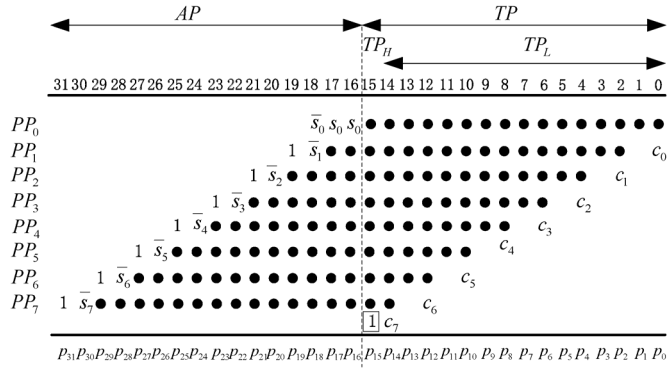


Fig. 7. Partial product diagram for fixed-width 16 × 16 booth multipliers (n = 16).

Direct-truncated Booth multipliers (DTM) [5], which are an extreme case of fixed-width multipliers, output an n-bit integer product by simply neglecting the bits in the *TP* part of the array without forming them in the first place, thus potentially producing a large error. As another extreme, post-truncated Booth multipliers (PTM) [3] form the complete partial product array, compress all the bits, compute with full precision, add an extra "1" to the $n-1^{th}$ column to exactly round the carry-out to the n^{th} column, and finally output the exact n-bit integer part of the final product (with rounding), as shown in Fig. 7. As such, PTMs are theoretically the *most accurate* fixed-width multipliers.

Our goal in approximate fixed-width multiplier design is to approach the accuracy of a PTM without incurring its high overhead that is commensurate with that of a full-precision multiplier. Under the AAAC model, we associate the accurate part (AP) and the truncation part (TP) of the array in Fig. 7 with the LPCU and ECU, respectively. More specifically, the bits in AP are processed by the LPCU while the effects of the ones in TP are approximated by the ECU in the form of error compensation. The exact product (EFCU output) is $O_{EFCU} = O_{LPCU} + S_{TP}$, where S_{TP} is the partial sum of TP, and O_{LPCU} is the LPCU output corresponding to AP. To reduce the amount of approximate error, we further divide TP into TP_H (i.e., the $n-1^{th}$ column) and TP_L (Fig. 7) and have [10]

$$S_{TP,H} = \frac{1}{2}SUM_{n-1}, S_{TP,L} = \frac{1}{4}SUM_{n-2} + \ldots + \left(\frac{1}{2}\right)^n SUM_0 \quad (9)$$

where $S_{TP,H}$ and $S_{TP,L}$ correspond to the partial sums of TP_H and TP_L, and SUM_i represents the sum of all bits in the i^{th} column, respectively. Now it is clear that $S_{TP} = S_{TP,H} + S_{TP,L}$. The main objective in the design of ECU is to well approximate $S_{TP} \approx O_{ECU}$ such that a fixed-width n-bit output is produced, i.e., $O_{EFCU} = O_{LPCU} + S_{TP} \approx O_{LPCU} + O_{ECU}$. Note again that the ECU of a PTM (most accurate fixed-width multiplier) produces as the output (with rounding)

$$O_{ECU,PTM} = int(S_{TP} + 1) = int(S_{TP,H} + S_{TP,L} + 1) \quad (10)$$

where $int(\cdot)$ returns the integer part of its argument. To approach the PTM, we design our ECU's output to be

$$O_{ECU} = int(S_{TP,H} + \tilde{S}_{TP,L} + 1) \quad (11)$$

where $\tilde{S}_{TP,L}$ is a good approximation to $S_{TP,L}$. In (11), only $\tilde{S}_{TP,L}$ is approximated by the ECU while $S_{TP,H}$ is computed

TABLE I
MODIFIED BOOTH ENCODING

Inputs			Partial Product	Booth Encoder Outputs				
b_{2i+1}	b_{2i}	b_{2i-1}	PP_i	n_i	t_i	o_i	z_i	c_i
0	0	0	0	0	0	0	1	0
0	0	1	$+A$	0	0	1	0	0
0	1	0	$+A$	0	0	1	0	0
0	1	1	$+2A$	0	1	0	0	0
1	0	0	$-2A$	1	1	0	0	1
1	0	1	$-A$	1	0	1	0	1
1	1	0	$-A$	1	0	1	0	1
1	1	1	-0(=0)	1	0	0	1	0

exactly. Regarding to (11), we denote the carry-out from TP_L to TP_H by θ

$$\theta = int(2 \cdot S_{TP,L}) \tag{12}$$

(10) can now be simplified to

$$O_{ECU,PTM} = int(S_{TP,H} + \frac{1}{2}\theta + 1) \tag{13}$$

Going back to (11), it is now clear that the main task of the ECU design is to well approximate θ. Since $S_{TP,H}$ is kept exact in (11), it is also natural to associate both AP and $S_{TP,H}$ with the LPCU, and process them with the LPCU's encoding and selection blocks. In this case, the ECU only produces an approximate θ.

C. Design of Error Compensation Unit

According to Section III-C, the key problem in ECU design is to classify all input patterns into largely equally sized groups with none or little overlap according to values of $E_{BC,i}$, which is the error before compensation (in this case θ).

To start, we first examine the standard Booth encoding that encodes each set of three consecutive bits of multiplier B into five signals and determines the corresponding partial product in terms of multiplicand A in Table I, where z_i signifies whether the partial product is zero or not, n_i specifies the sign of each partial product, and PP_i is the actual i^{th} partial product generated from the selection block. As in Fig. 7, it is worth noting that Booth encoding is applied across the entire partial product array including the TP part, which is associated with the error. By following the ECU design guidance in Section III-C, we identify a set of error compensation signatures of low cost from Table I to compensate for the error due to TP.

Our key idea is to use *encoded sign* and *magnitude* information of the partial products to classify the input patterns into largely equally sized non-overlapping groups according to the $E_{BC,i}$ value. In the following, we first present a set of error signatures for each partial product, and then compress them for the entire ECU.

1) Signatures for Each Partial Product Row: The first signature to be chosen is z_i. It is effective since a non-zero value of z_i signifies the zero-valued corresponding partial product PP_i, thereby classifying the inputs into two $E_{BC,i}$ (zero vs. non-zero error) groups independently of the multiplicand A.

Starting from these two input groups, we select our second signature to be n_i, which encodes the sign of PP_i, allowing us to further partition the large non-zero $E_{BC,i}$ input group into two smaller groups of positive vs. negative error.

To further reduce the approximation error, we introduce the third signature to split the large signed error input groups by using the magnitude information of each partial product. For this, we count the number of non-zero bits in multiplicand A: $n_{nza} = \sum_{i=0}^{n-1} a_i$.

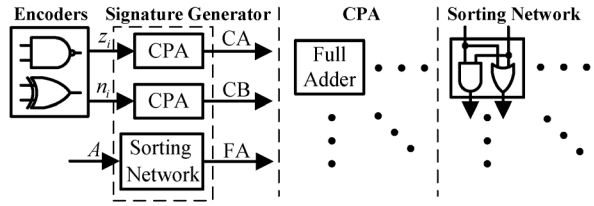


Fig. 8. Blocks and schematics of signature generator.

2) Compressed Signatures for ECU: Note that n_i and z_i are defined for each partial product and there are $n/2$ partial products for nxn bits multiplication. In addition, n_{nza} ranges from 0 to n. Utilizing these signatures for the ECU would create a huge number of input groups and lead to significant area and energy overhead. To simplify the design of the signature generator, we first sum up n_i and z_i to produce CA and CB, respectively, and then introduce a Boolean variable FA that indicates whether n_{nza} is above $n/2$ or not

$$CA = \sum_{i=0}^{\frac{n}{2}-1} z_i,\ CB = \sum_{i=0}^{\frac{n}{2}-1} n_i, FA = \begin{cases} 0, & \sum_{i=0}^{n-1} a_i < \frac{n}{2} \\ 1, & \sum_{i=0}^{n-1} a_i \geq \frac{n}{2} \end{cases} \tag{14}$$

CA, CB, and FA are the final set of compressed signatures we use for the ECU. These signatures can be implemented with low-cost in hardware and possess the desirable properties outlined in Section III-C based on the proposed general AAAC model. Fig. 8 illustrates the design of the proposed signature generator (left) that consists of two carry propagation adders (CPAs) (middle) for generating CA, CB and an n-input odd-even sorting network [25] (right) for FA generation. Note that n_i and z_i are already computed by the encoders in the LPCU.

D. The Complete Fixed-Width Multiplier

With the selected signatures and classified input groups, next, we need to determine the actual error compensation for each group, i.e., an approximate to θ in (12). As discussed in Section III-A, one can follow Theorems 1–3 to choose a fixed compensation for each input group to minimize a targeted error metric. For example, to minimize E_{ms}, the optimal compensation is the average $int(2 \cdot S_{TP,L})$ value for each group ($\overline{\theta}$). The complete proposed fixed-width Booth multiplier is shown in Fig. 9. The ECU is designed to run in parallel with the selection block and part of compression block so that it causes little extra delay during runtime.

We take 16 × 16 fixed-width Booth multiplier design as an example to illustrate the signature and compensation generation schemes, and additional possible simplifications. To further simplify the ECU, we consider different ranges and combinations of the signature values in Table II, where $\wedge$ denotes AND operation. The goal is to identify a smaller set of refined input groups with controlled error spread. $\overline{S}_{TP,L}$ is the average of $S_{TP,L}$ in each input group. To minimize E_{ms}, the optimal integer error compensation $\overline{\theta}$ is set to be the average of (12) in the group.

To further simplify, as shown in Table III, Case 2 and Case 3 are merged to form Group 1 (G1). Group 2 (G2) consists of Cases 1 and 4. Finally, Case 5 is Group 3 (G3). Each merged

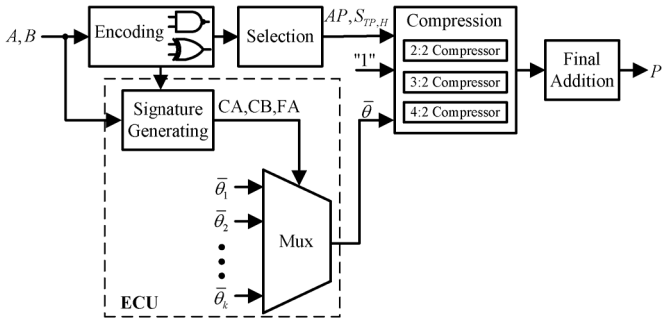


Fig. 9. Proposed fixed-width booth multiplier architecture.

TABLE II
COMPENSATION FOR INPUT GROUPS OF 16 × 16 MULTIPLIER

CA range	Case	Condition	$\overline{S}_{TP,L}$	$\overline{\theta}$
[0,1]	1	$CA=1 \wedge CB<3 \wedge FA=0$	0.9853	1
	2	The rest when CA in [0,1]	1.1259	2
[2,5]	3	$CA=2 \wedge CB>3 \wedge FA=0$ $CA=2 \wedge CB<3 \wedge FA=1$	1.0188	2
	4	The rest when CA in [2,5]	0.8580	1
[6,8]	5	CA in [6,8]	0.4001	0

TABLE III
COMPENSATION FOR THE REFINED INPUT GROUPS

Group	Case	$\overline{\theta}$	$\overline{S}_{TP,L}$
1	2,3	2	1.1153
2	1,4	1	0.8608
3	5	0	0.4001

group has the same $\overline{\theta}$ and error selection is realized by a simple 3-to-1 mux.

E. Proposed Full-Width Booth Multiplier

Approximate full-width multipliers, i. e., ones that approximate accurate $n \times n$ Booth multipliers by outputting a full-width $2n$-bit approximate product, are also useful for many practical applications.

The presented fixed-width design can be readily extended to facilitate full-width operation with the difference being that in this case we would like to approximate $S_{TP,L}$ by $\tilde{S}_{TP,L}$ as in (11). Again, to minimize E_{ms}, for instance, the optimal compensation for each input group would be the average of $S_{TP,L}$, denoted by $\overline{S}_{TP,L}$, in that group. For n=16, we show the values of $\overline{S}_{TP,L}$ for the same three input groups in the last column of Table III and we have

$$O_{ECU} = S_{TP,H} + \overline{S}_{TP,L} \tag{15}$$

Similar to the proposed fixed-width multiplier, since $S_{TP,H}$ is kept exact in (15), it is also natural to associate both AP and $S_{TP,H}$ with the LPCU, and process them with the LPCU's encoding and selection blocks. In this case, the ECU only produces an approximate $\overline{S}_{TP,L}$.

V. PROPOSED SQUARER DESIGN

A. The Basic Squarer Design

We demonstrate the application of the AAAC model to approximate fixed-width and full-width squarer designs.

Fig. 10 shows the full 8-partial squaring array (PS_0 to PS_7) for a full-precision 16-bit squarer, where the input is denoted by $A(a_{n-1} \ldots a_0)$ [11]. Here, we use the method in [11] to

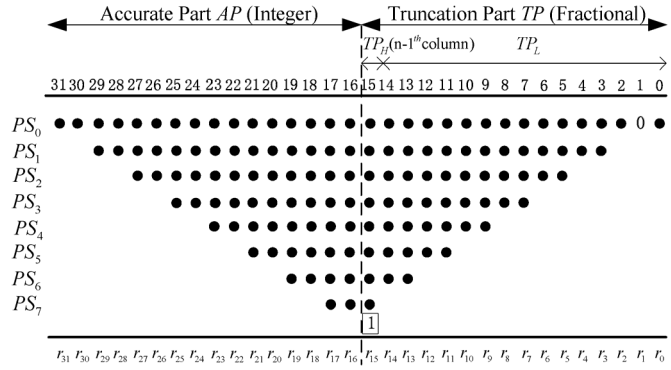


Fig. 10. Squaring diagram for 16-bit fixed-width squarers.

TABLE IV
COMPENSATION FOR INPUT GROUPS OF 16-BIT SQUARER

Group	Condition	$\overline{\theta}$	$\overline{S}_{TP,L}$
1	$CA=0$	0	0.2197
	$CA=1 \wedge CB=0$		0.4539
2	$CA=1 \wedge CB=1$	1	0.6210
	$CA=2 \wedge CB=0$		0.8133
3	$CA=2 \wedge CB=1$	2	1.0134
	$CA=3$		1.2902
4	$CA=4$	3	1.6966
5	$CA=5$	4	2.1278
6	$CA=6$	5	2.5838
7	$CA=7$	6	3.0645

implement squarers instead of using Booth algorithm as applied to multiplier design in Section IV because squarers implemented by using the method in [11] are more energy-efficient and faster since most partial products bits are implemented by simple AND operation of two input bits instead of more complex Booth encoding and selection blocks.

The squarer design process is very similar to the one presented for the proposed multipliers, e.g., based on (9)–(15). Again, the key problem is to design an ECU to well approximate θ. Briefly, by following the ECU design guidance in Section III-C, we consider the signals on the $n-2^{th}$ column as signatures since they have the highest weight on TP_L and include all input bits which contribute to TP_L. To simplify the design of the signature generator, we sum up the signals on the $n-2^{th}$ column to produce the first signature CA. To further reduce the approximation error, we introduce one input bit as the second signature (CB) to further split the large input groups formed by CA. Accordingly, input bit a_6 is chosen as the second signature CB for the proposed 16-bit squarer.

The final input groups of the 16-bit squarer classified by CA and CB are show in Table IV. CA is generated by an odd-even sorting network [25], which has a low hardware overhead, and CB is selected directly from the input A.

The error compensations $\overline{\theta}$ for the fixed-width squarer are shown in the second last column of Table IV. The values of $\overline{S}_{TP,L}$ (error compensation) of different input groups for the full-width squarer are shown in the last column of Table IV.

B. Further Error and Cost Reduction for Fixed-Width Squarer

As described in Section III-B, we may introduce certain input bits as extra signatures to sub-divide each of the large input classes into a smaller group to further reduce one or more error metrics. To further simplify the logic, thus decreasing energy

consumption and area overhead of ECU, don't cares are set for certain input groups.

We demonstrate the application of the above process to the proposed 16-bit fixed-width squarer while targeting E_{max}. To give an overall evaluation of design choices, an energy-delay-max error product is defined as EDE_{max}.

First, we introduce extra signatures. In the proposed 16-bit fixed-width squarer design, input bit a_6 is utilized as one signature because it can further divide some of the large eight classes formed by signature CA into smaller groups. Specifically, according to the design guidance in Section III that an efficient signature should be able to divide input cases into groups largely of equal length (sub-dividing large input classes formed by the first signature CA in this case), we evaluate each input bit from a_0 to a_{15} as a candidate extra signature. The simulation results indicate that selecting a_6 as a signature can divide more large original classes formed by the first signature CA and achieve a better overall accuracy performance than any other input bit from a_0 to a_{15}. Using the same method, additional input bit signatures are chosen one at a time subsequently. Detailed results should be discussed in the result section.

Second, to further reduce energy consumption and area, as illustrated in Section III-B, don't cares are introduced to some of the input groups. The ECU with signatures of CA and the extra signatures selected according to the above process has certain logic complexity. In order to simplify ECU design and trade off between cost and error, we set compensation values of the input groups which have the least worst case impacts on the overall error to be don't cares. For example, to minimize overall E_{max}, we may rank the groups based on their worst case $E_{max,j}$ ($\overline{E_{max,j}}$), which is the biggest E_{max} that the group can reach when $\overline{\theta}$ is any value in its range. Since $\overline{\theta}$ has three bits in this case, it ranges from 0 to 7. After the groups are ranked by $\overline{E_{max,j}}$, those that have the lowest $\overline{E_{max,j}}$ are given a higher priority to be set as don't cares.

VI. Experimental Results

The proposed 16-bit fixed-width Booth multiplier and squarers are designed in Verilog HDL, synthesized using Synopsys Design Compiler [20] with a commercial 90 nm CMOS technology and standard cell library. From Synopsys Design Compiler synthesis (Design Vision) reports, we get the pre-layout delay, dynamic power, leakage power, and area. We also implement five additional fixed-width Booth multipliers: DTM (direct truncated Booth multiplier) [5], PEBM (with probabilistic estimation bias compensation) [10], ZSM (uses sum of $\overline{z_i}$ as signatures) [7], PTM (post-truncated Booth multiplier—most accurate/expensive fixed-width multiplier) [3] and MLCP (uses a multilevel conditional probability estimator to make error compensation) [19]. Four additional squarers are implemented: DTS (direct truncated squarer), CCS (with a constant compensation) [11], VCS (the signals on the $n-2^{th}$ column as the compensation) [12], and PTS (post-truncated squarer − most accurate/expensive fixed-width squarer).

For all Booth multiplier and squarer designs implemented in this paper, partial products are generated and then compressed to two partial products using 2:2, 3:2 [23], and 4:2 [24] compressors. 2:2, 3:2, and 4:2 compressors provide an efficient method for compressing the number of partial products to two, which are optimized for speed and are well incorporated in silicon compilation and logic synthesis tools. Finally, the two partial products are added up by a carry propagation adder (CPA) to produce final results.

TABLE V
Comparison of 16×16 Fixed-width Multipliers

Multiplier	Area (um^2)	Delay (ns)	Energy (pJ)	E_{ms}	EDE_{ms} ($pJ \cdot ns$)
DTM	2,645	2.61	2.24	9.85	57.59
PTM	5,239	3.72	6.51	0.08	1.94
PEBM	2,937	2.79	2.73	0.35	2.67
ZSM	3,256	2.99	3.32	0.20	1.99
MLCP	3,250	2.89	2.83	0.22	1.80
Proposed	3,755	2.99	3.59	0.15	1.61

A. Comparison of Different Multipliers

In Table V, six fixed-width multipliers are compared for area, delay, energy which is the product of delay and power (sum of dynamic power and leakage power), and an energy-delay-mean squared error product (EDE_{ms}). The energy consumption and area of the proposed multiplier are slightly larger than PEBM and ZSM, but are much smaller than PTM, with a 44.85% and 28.33% reduction respectively. The proposed design has a significantly reduced EDE_{ms}, with 19.10% and 10.56% reduction compared with ZSM and MLCP. This indicates that our design delivers much improved accuracy with a small overhead.

B. Accuracy Analysis for Multipliers

In this section, we provide a more detailed accuracy comparison among different approximate multipliers. Error reduction of the proposed design to existing designs are defined as $(|E_{existing} - E_{proposed}|/E_{existing}) \times 100\%$, where $E_{existing}$ represents one of error metrics (E_{ave}, E_{max}, and E_{ms}) of the compared existing design and $E_{proposed}$ is defined as the same error metric of the proposed design.

1) Fixed-Width Booth Multipliers: We evaluate the accuracies of the five different designs in terms of E_{ave}, E_{max} and E_{ms} ((1)–(3)) in Section II-A) as a function of bit width ($n = 8,\ 12,\ 16,\ and\ 32$), where exact error analysis is conducted for all possible input combinations when $n = 8,\ 12,$ and 16. While for $n = 32$, since the number of input combinations is very large, exact error analysis becomes extremely time-consuming. To alleviate the computational challenge while getting decent error estimates, we evaluate E_{ave} and E_{ms} by averaging over a large number of input combinations as follows. For instance, for E_{ave} evaluation, we first randomly generate one data set of 400 million input combinations and calculate E_{ave} for this set. To give a more decent and accurate error estimates, we randomly generate 10 such data sets in total, with 400 million input combinations for each set, and calculate the average E_{ave} of the ten sets and get the final E_{ave} result. While for the final E_{max} result, we choose the maximum E_{max} of the ten sets. Fig. 11(a) shows the error reductions of the proposed fixed-width multipliers over DTM, PEBM, and ZSM. For example, our 16-bit design significantly reduces E_{ave}, E_{max}, and E_{ms} by 11.11%, 28.11%, and 25.00%, respectively, when compared with ZSM. Note that Fig. 11(a) also presents that with the increase of bit width, the proposed multiplier has a more significant increase

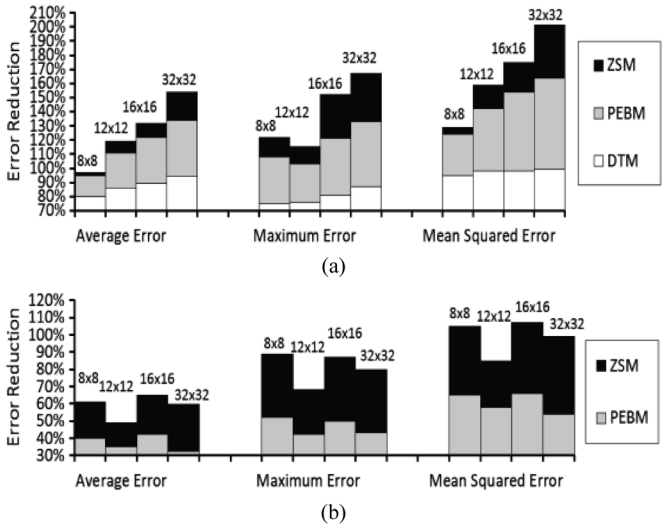


Fig. 11. Error reductions of proposed booth multipliers over DTM, PEBM and ZSM: (a) fixed-width, (b) full-width.

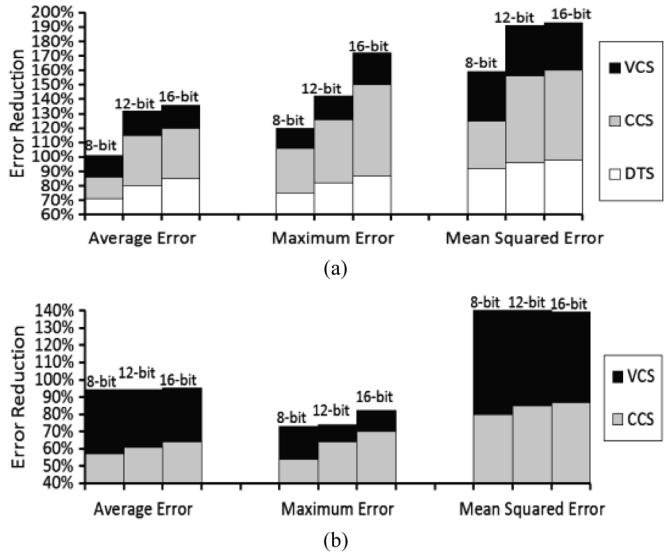


Fig. 12. Error reductions of proposed squarers over DTS, CCS and VCS: (a) fixed-width, (b) full-width.

TABLE VI
COMPARISON OF 16-BIT FIXED-WIDTH SQUARERS

Squarer	Area (um^2)	Delay (ns)	Energy (pJ)	E_{ms}	EDE_{ms} ($pJ \cdot ns$)
DTS	1,566	2.21	0.80	4.34	7.67
PTS	3,016	2.93	2.05	0.08	0.48
CCS	1,891	2.22	0.98	0.28	0.61
VCS	1,997	2.34	1.08	0.16	0.40
Proposed	2,090	2.45	1.18	0.11	0.32

in terms of E_{ms} reduction. To further illustrate the accuracy advantages of the proposed multiplier, we compare the proposed 16×16 fixed-width design with MLCP, which is a more recent design. A 15.79%, 37.60%, and 31.82% reduction is achieved in terms of E_{ave}, E_{max}, and E_{ms}.

2) Full-Width Booth Multipliers: We further compare the accuracies of the five different multipliers in full width in Fig. 11(b). The proposed 16×16 full-width Booth multiplier achieves 24.14%, 37.50%, and 46.15% reduction on E_{ave}, E_{max}, and E_{ms} ((1)–(3)in Section II-A), respectively, when compared with ZSM and outperforms the most accurate fixed-width PTM with an error reduction of 12.00% and 12.50% for E_{ave} and E_{ms}, when $n = 16$.

C. Comparison of Different Squarers

According to the results in terms of area, delay, energy which is product of delay and power (sum of dynamic power and leakage power), and $Energy \cdot Delay \cdot E_{ms}$ (EDE_{ms}) in Table VI, the proposed squarer consumes 42.43% and 30.70% less energy and area than PTS. Despite slightly more energy consumption and area than CCS and VCS, the proposed 16-bit fixed-width squarer reduces EDE_{ms} significantly, with 20.00% reduction compared with VCS.

D. Accuracy Analysis for Squarers

1) Fixed-Width Squarers: Fig. 12(a) shows the error reductions of the proposed fixed-width squarers over DTS, CCS and VCS. For instance, our 16-bit design has a significant 18.18%, 21.67%, and 31.25% reduction in terms of E_{ave}, E_{max}, and E_{ms}, when compared with VCS.

2) Full-Width Squarers: More significant error reduction is achieved when the proposed squarers operate in full width. As shown in Fig. 12(b), the proposed 16-bit full-width squarer significantly reduces E_{ave}, E_{max}, and E_{ms} by 31.58%, 11.69%, and 50.00%, respectively, when compared with VCS and outperforms the most accurate fixed-width PTS with an error reduction of 48.00% and 62.50% for E_{ave} and E_{ms}, respectively, when $n = 16$.

E. Further Reduction of Error and Cost for the 16-bit Fixed-Width Squarer

As illustrated in Section V-B, we choose extra signatures from input bits (a_0 to a_{15}) and set don't cares to some of the groups to further reduce EDE_{max}, the energy-delay-max error product that is an overall performance measure.

As described in Section V-B, we choose a number of extra signatures directly from the input bits one at a time. Table VII shows the optimal extra signatures from input bits, given the number of extra signatures. We limit the number of extra signatures to seven as giving beyond this would lead to rapid increase of area and energy overheads. Corresponding to the last row of the table, when introducing seven extra signatures, to trade off between energy consumption and E_{max}, we set 110 groups as don't cares. Further optimizing the squarer by introducing seven extra signatures and don't cares decreases EDE_{max} significantly, achieving a reduction of 15.81%, when compared with the proposed 16-bit fixed-width squarer design with only one extra signature and no don't cares (the second row).

VII. CONCLUSION

A general model is presented for array-based approximate arithmetic computing to guide the design of approximate Booth multipliers and squarers. To shed light on the design of ECU, which is the key of AAAC design, we develop four theorems to address two critical design problems of the ECU design, namely,

TABLE VII
EDE_{max} OF 16-BIT FIXED-WIDTH SQUARERS WITH EXTRA INPUT SIGNATURES

Number of Extra Signatures	Extra Signatures	Area (um^2)	Delay (ns)	Power (mW)	Energy (pJ)	E_{max}	EDE_{max} ($pJ \cdot ns$)
1	a_6	2,090	2.45	0.48	1.18	0.94	2.72
2	a_6a_7	2,095	2.45	0.48	1.18	0.92	2.66
3	$a_6a_7a_{13}$	2,129	2.45	0.48	1.18	0.85	2.46
4	$a_6a_7a_{13}a_0$	2,137	2.45	0.48	1.18	0.84	2.43
5	$a_6a_7a_{13}a_0a_4$	2,156	2.46	0.48	1.18	0.82	2.38
6	$a_6a_7a_{13}a_0a_4a_5$	2,202	2.48	0.48	1.19	0.80	2.36
7	$a_6a_7a_{13}a_0a_4a_5a_8$	2,295	2.49	0.49	1.22	0.79	2.40
7 (After introducing don't cares)	$a_6a_7a_{13}a_0a_4a_5a_8$	2,167	2.46	0.48	1.18	0.79	2.29

determination of optimal error compensation values and identification of the optimal error compensation scheme. To further reduce energy consumption and area, we introduce don't cares for ECU logic simplification. The presented experimental results show that the proposed approach has led to significant performance improvements for a number of approximate multiplier and squarer designs.

REFERENCES

[1] M. J. Schulte and E. E. Swartzlander, "Truncated multiplication with correction constant," *Proc. Workshop VLSI Signal Processing VI*, pp. 388–396, 1993.

[2] E. J. King and E. E. Swartzlander, "Data-dependent truncation scheme for parallel multipliers," *Conf. Rec. 31st Asilomar Conf. Signals, Syst. Comput.*, 1997.

[3] S.-J. Jou, M.-H. Tsai, and Y.-L. Tsao, "Low-error reduced-width Booth multipliers for DSP applications," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 50, no. 11, pp. 1470–1474, 2003.

[4] S. O. N. G. Min-An, V. A. N. Lan-Da, and K. U. O. Sy-Yen, "Adaptive low-error fixed-width booth multipliers," *IEICE Trans. Fundam. Electron., Commun., Comput. Sci.*, vol. 90, no. 6, pp. 1180–1187, 2007.

[5] H.-A. Huang, Y.-C. Liao, and H.-C. Chang, "A self-compensation fixed-width booth multiplier and its 128-point FFT applications," in *Proc. 2006 IEEE Int. Symp. Circuits Syst. (ISCAS)*.

[6] K.-J. Cho *et al.*, "Design of low-error fixed-width modified booth multiplier," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 5, pp. 522–531, 2004.

[7] J.-P. Wang, S.-R. Kuang, and S.-C. Liang, "High-accuracy fixed-width modified booth multipliers for lossy applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 1, pp. 52–60, 2011.

[8] Y.-H. Chen and T.-Y. Chang, "A high-accuracy adaptive conditional-probability estimator for fixed-width booth multipliers," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59, no. 3, pp. 594–603, Mar. 2012.

[9] Y.-H. Chen, "An accuracy-adjustment fixed-width booth multiplier based on multilevel conditional probability," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 1, pp. 203–207, 2014.

[10] C.-Y. Li *et al.*, "A probabilistic estimation bias circuit for fixed-width Booth multiplier and its DCT applications," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 58, no. 4, pp. 215–219, 2011.

[11] E. G. Walters, III, M. J. Schulte, and M. G. Arnold, "Truncated squarers with constant and variable correction," *Proc. SPIE*, vol. 5559, pp. 40–50, 2004.

[12] E. G. Walters, III and M. J. Schulte, "Efficient function approximation using truncated multipliers and squarers," in *Proc. 17th IEEE Symp. Comput. Arith. (ARITH-17 2005)*.

[13] V.-P. Hoang and C.-K. Pham, "Low-error and efficient fixed-width squarer for digital signal processing applications," in *Proc. 2012 4th Int. Conf. Commun. Electron. (ICCE)*, pp. 477–482.

[14] B. Shao and P. Li, "A model for array-based approximate arithmetic computing with application to multiplier and squarer design," *Proc. 2014 Int. Symp. Low Power Electron. Design (ISLPED '14)*, pp. 9–14.

[15] K. Du, P. Varman, and K. Mohanram, "High performance reliable variable latency carry select addition," *Proc. Design, Autom., Test Eur. Conf. Exhib. (DATE) 2012*, pp. 1257–1262.

[16] N. Zhu, W. L. Goh, and K. S. Yeo, "An enhanced low-power high-speed adder for error-tolerant application," in *Proc. Int. Symp. Integr. Circuits (ISIC)*, Dec. 2009, pp. 69–72.

[17] K. Yongtae, Y. Zhang, and P. Li, "An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems," *Proc. 2013 IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, pp. 130–137.

[18] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. New York: Springer, 2006.

[19] R. A. Bergamaschi *et al.*, "Efficient use of large don't cares in high-level and logic synthesis," in *Proc. 1995 IEEE/ACM Int. Conf. Comput.-Aided Design*.

[20] Synopsys. Inc, "Design compiler user guide," [Online]. Available: http://www. synopsys. com 2010

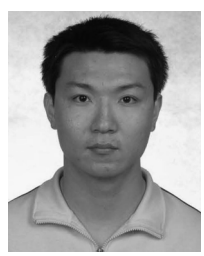
[21] N. H. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Reading, MA, USA: Addison-Wesley, 2010.

[22] E. de Angel and E. E. Swartzlander, Jr., "Low power parallel multipliers," in *Proc. IEEE VLSI Signal Process. Workshop IX*, 1996, pp. 199–208.

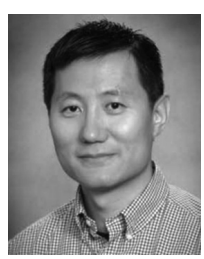
[23] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, no. 1, pp. 14–17, Feb. 1964.

[24] V. G. Oklobdzija, D. Villeger, and S. S. Liu, "A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach," *IEEE Trans. Comput.*, vol. 45, no. 3, pp. 294–306, 1996.

[25] K. E. Batcher, "Sorting networks and their applications," in *Proc. Apr. 30–May 2, 1968, ACM Spring Joint Comput. Conf.*, pp. 307–314.



Botang Shao received the M.S. degree in Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, USA, in 2014. He is currently a design engineer at Freescale Semiconductor, Austin, TX, USA. His research interests are in the areas of low-power arithmetic circuits and VLSI design.



Peng Li (S'02–M'04–SM'09) received the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2003. He is currently an associate professor with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, USA. His research interests are in the general areas of circuits and systems, electronic design automation, aspects of parallel computing and computational neuroscience. He received four best paper awards from major IEEE/ACM conferences and an NSF Career Award. He edited two books and has authored or co-authored over 160 publications.