IEICE TRANSACTIONS

on Fundamentals of Electronics, Communications and Computer Sciences

VOL. E102-A NO. 3
MARCH 2019

A PUBLICATION OF THE ENGINEERING SCIENCES SOCIETY
The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

IEICE TRANS. FUNDAMENTALS, VOL.E102–A, NO.3 MARCH 2019

PAPER

Design and Analysis of Approximate Multipliers with a Tree Compressor*

Tongxin YANG[†], ***Student Member***, **Tomoaki UKEZONO**[††], ***and*** **Toshinori SATO**[††a)], ***Members***

SUMMARY Many applications, such as image signal processing, has an inherent tolerance for insignificant inaccuracies. Multiplication is a key arithmetic function for many applications. Approximate multipliers are considered an efficient technique to trade off energy relative to performance and accuracy for the error-tolerant applications. Here, we design and analyze four approximate multipliers that demonstrate lower power consumption and shorter critical path delay than the conventional multiplier. They employ an approximate tree compressor that halves the height of the partial product tree and generates a vector to compensate accuracy. Compared with the conventional Wallace tree multiplier, one of the evaluated 8-bit approximate multipliers reduces power consumption and critical path delay by 36.9% and 38.9%, respectively. With a 0.25% normalized mean error distance, the silicon area required to implement the multiplier is reduced by 50.3%. Our multipliers outperform the previously proposed approximate multipliers relative to power consumption, critical path delay, and design area. Results from two image processing applications also demonstrate that the qualities of the images processed by our multipliers are sufficiently accurate for such error-tolerant applications.

key words: *approximate computing, approximate tree compressor, high speed multiplier, low power multiplier, small area multiplier*

1. Introduction

Recently emerging applications, such as those involving image processing and recognition, which are computationally demanding, have created challenges relative to power consumption. Most of these applications have an inherent tolerance for insignificant inaccuracies; therefore, there are unprecedented opportunities to reduce power consumption. Multiplication is a fundamental arithmetic function for many of these applications. By exploiting the inherent tolerance feature, approximate computing can trade accuracy for power. Currently, this trade-off plays an important role in such application domains [1].

A system level methodology that utilizes dynamic voltage scaling has been proposed [2]. This methodology truncates some least significant bits (LSBs) to release slack from critical paths and reduces the voltage to achieve lower power consumption. In this study, we focus on the structure of approximate multiplier designs. Approximate arithmetic focuses on basic arithmetic units, such as 8×8 approximate multipliers used in image signal processing applications and in custom accelerators to achieve lower power consumption and shorter critical path delay. In our preceding study [3], we proposed an approximate tree compressor (ATC) that reduces accumulation layers to simplify a complex circuit. It reduces the n-row partial product (PP) tree array by half. We introduced our approach as a term comprehending the power and accuracy requirements from simplifying the PP reduction component. An incomplete adder cell (iCAC), which is a basic cell of the compressor, is used to generate a vector to compensate accuracy and a sum in parallel. Pairs of signals, each of which is generated by an iCAC, can be accumulated and collected separately and simultaneously, thereby, reducing the length of the critical path. We qualitatively analyzed the accuracy of the ATC, finding that it delivered 0.0383% of the normalized mean error distance (NMED [4]**) in an 8×8 PP tree. Using the ATC we proposed two approximate multipliers [3]. In this paper, we improve the accuracy of both multipliers and thus propose additional two multipliers.

Since the ATC is a basic functional block, a variety of multipliers were constructed using the ATC [5]–[7]. In this paper, we focus on approximate multipliers that do not have dynamic configurability. We implement the four approximate multipliers using the ATC and a conventional multiplier in Verilog HDL using a 45-nm library and evaluate the power consumption, critical path delay, and design area. Power consumption is evaluated by two scenarios, one of which is considered for the first time in this paper. The added scenario reveals results are different from the published ones [3]. Comparison with the conventional Wallace tree multiplier demonstrates that one of the evaluated approximate multipliers reduces power consumption by 36.9% when the scenario of a random test pattern including 100,000 test vectors is used. We provide a crosswise comparison to demonstrate the superiority of our multipliers compared with existing approaches [4], [8] by implementing the three established approximate multipliers to evaluate power consumption, critical path delay, design area and accuracy. We also evaluate the qualities of all approximate multiplier designs in two real image processing applications.

The remainder of this paper is organized as follows. Section 2 introduces the related work on approximate multipliers. In Sect. 3, we explain how PP layer is simplified by the ATC and analyze its accuracy using a mathematical method.

Manuscript received September 3, 2018.

[†]The author is with Graduate School of Information and Control Systems, Fukuoka University, Fukuoka-shi, 814-0180 Japan.

[††]The authors are with Department of Electronics Engineering and Computer Science, Fukuoka University, Fukuoka-shi, 814-0180 Japan.

*An earlier version of this paper was presented at 35th IEEE International Conference on Computer Design, pp.89–96, November 2017 [3].

a) E-mail: tsato@fukuoka-u.ac.jp

DOI: 10.1587/transfun.E102.A.532

**NMED will be defined in Sect. 3.1.

We subsequently describe four approximate multipliers using the ATC. In Sect. 4, we discuss the implementations of our approximate multipliers, the conventional multiplier, and three established approximate multipliers, and after that we compare them in terms of power consumption, critical path delay, design area, and accuracy. In Sect. 5, we use two applications for image processing to evaluate the quality of our approximate multipliers. Conclusions are presented in Sect. 6.

2. Related Work

Adder is a basic cell in most Multipliers. Gupta et al. [9] discussed how to simplify the complexity of a conventional mirror adder cell at the transistor level by proposing five approximate adders, which reduce the number of transistors and the load capacitance. Mahdiani et al. [10] proposed Lower-Part-OR adder, which utilizes OR gates for lower-bit addition and accurate full adders for upper-bit addition. iCAC is similar to [10], because it uses an OR gate to generate the approximate sum. However, our method can obtain accurate results. Liu et al. [4] utilized an approximate adder to reduce carry propagation delay in PP accumulation and proposed a recovery vector to improve accuracy of the final product. Designers should select the bit width of the error recovery vector to satisfy accuracy requirements. The ATC also utilizes a recovery vector, but there is a difference in vector generation method. Liu et al.'s method [4] requires lower bit values, whereas ours does not.

Hashemi et al. [11] proposed a technique that reduces the size of the multiplier by detecting the leading one bit of the input operands and by selecting the following $(k-1)$ consecutive number of bits, where k is a designer-defined value of which specifies the bandwidth used in the core accurate multiplier. Yang et al. [8] proposed three 4-to-2 column approximate compressors for PP reduction in multipliers. Our approximate multipliers use their 4-to-2 compressor, but the utilization is limited. Furthermore, we exploit its asymmetric characteristics for accuracy improvement.

3. Approximate Multipliers

A typical multiplier consists of three parts: (i) PP generation using an AND gate, (ii) PP reduction using an adder tree, and (iii) addition of the final result using a carry propagation adder (CPA). Power consumption and circuit complexity are dominated by PP reduction [12]. We focus on the PP tree to reduce power consumption and analyze the results relative to accuracy.

This section is organized as follows. Section 3.1 explains how the PP layer is simplified by the ATC [3]. Section 3.2 presents the approximate multipliers that use the ATC. Section 3.3 introduces other ATC-based approximate multipliers and compares the ones proposed in Section 3.2.

3.1 Approximate Tree Compressor

First, we explain the iCAC. Figure 1(a) shows an accurate half adder, for which the following equation is obtained:

$$a + b = 2c + s = (c + s) + c,$$

where c and s denote a single binary digits carry and a sum, respectively, and the value of the sum in a multi-digit addition is $2c + s$.

The value c is generated by a *AND* b and s is generated by a *XOR* b, so $(c + s)$ can be generated by a *OR* b. Based on the above, consider the basic logic cell shown in Fig. 1(b), for which the following equations are obtained:

$$p = c + s,$$
$$q = c,$$
$$\{c, s\} = a + b = p + q.$$

where { , } denotes concatenation. The last equation explains that a 2-bit output is given from the addition of two 1-bit inputs. This cell is named iCAC. Table 1 shows the truth tables for an accurate half adder and an iCAC. Note that the bit position of c and that of s, p, and q are different. As can be seen, q is equal to c. While p is not equal to s, the precise sum can be obtained by adding p and q, so the iCAC is not an approximate adder but an element of a precise adder.

By extending the above equation to m bits, the following equation is obtained:

$$\mathrm{S} = \mathrm{A} + \mathrm{B} = \mathrm{P} + \mathrm{Q}.$$

where A, B, P, and Q are m-bit values, each bit corresponds to a, b, p, and q, respectively. A row of eight iCACs, which is used for 8-bit inputs, is shown in Fig. 2.

Consider an example of an 8-bit adder with two inputs of A = 01011111 and B = 00110110. While the accurate sum S is 10010101, the row of eight iCACs produces P = 01111111 and Q = 00010110. Again, it is evident that the

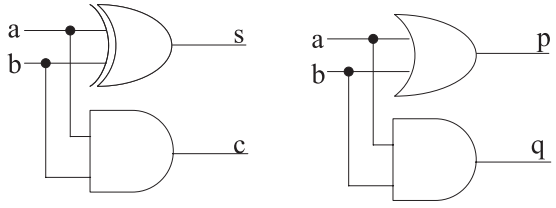

Fig. 1 (a) Accurate half adder, and (b) incomplete adder cell.

Table 1 Truth table for a half adder and an incomplete adder cell.

Inputs		Outputs			
		Accurate half adder		iCAC	
a	b	c	s	q	p
0	0	0	0	0	0
0	1	0	1	0	1
1	0	0	1	0	1
1	1	1	0	1	1

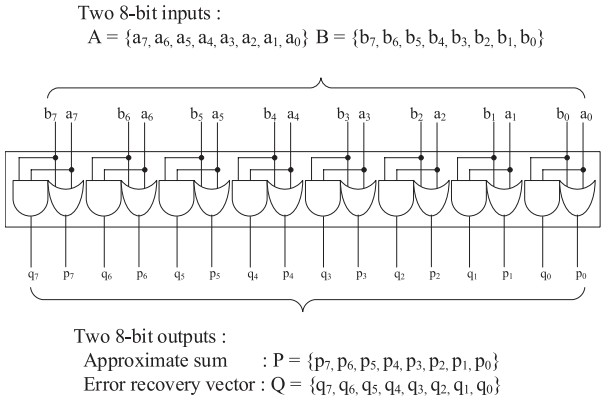

Fig. 2 A row of the incomplete adder cells with two 8-bit inputs.

following holds:

$$S = P + Q. \tag{1}$$

Although S is obtained from P and Q, we will use P as the approximation for S. Q can be used as an error recovery vector for the approximate sum P.

Next, we explain the ATC. Consider an n-bit multiplier with n rows, each of which has n PPs. Using the row of iCACs shown in Fig. 2, each pair of PP rows is replaced by one P and one Q. Now, we obtain $n/2$ Ps and $n/2$ Qs. If we use the sum of the $n/2$ Qs instead of the $n/2$ Qs themselves, the n PP rows are compressed to the $n/2$ P rows and the sum row, which comprise $(n/2) + 1$ rows. Remember that P is always greater than or equal to S and that Q is equal to C. By exploiting these features, we can use OR gates to generate the approximate sum of the $n/2$ Qs without significant loss of accuracy. Because the approximate sum of the Qs cannot completely recover accuracy, we call the approximate sum of Qs an accuracy compensation vector and use V for its designation.

This is the ATC. An ATC with n inputs is called an ATC-n, and the structure of an example ATC with eight inputs (ATC-8) is shown in Fig. 3. The rectangles represent rows of iCACs, with the number of iCACs in each row (rectangle) is dependent upon the bit width of the inputs. When there are eight m-bit inputs (D1, D2, $\cdots$, D8), four rows of m iCACs are required to build the m-bit ATC-8. The ATC-8 generates four approximate sums (P1, P2, P3, and P4) and four error recovery vectors (Q1, Q2, Q3, and Q4). OR gates generate the accuracy compensation vector, V. As a result, the eight inputs are reduced to five.

Then, we analyze the accuracy of the ATC quantitatively. Assume an n-bit approximate multiplier, where n is an even number, and $n/2$ error recovery vectors (denoted by Q1, Q2, $\cdots$, Q$(n/2)$) must be collected. The sum of these vectors is obtained as follows:

$$Q_i = Q1_i + Q2_i + \cdots + Q(n/2)_i. \tag{2}$$

The accuracy compensation vector, V, is defined as follows:

$$V_i = Q1_i \; OR \; Q2_i \; OR \; \cdots \; OR \; Q(n/2)_i, \tag{3}$$

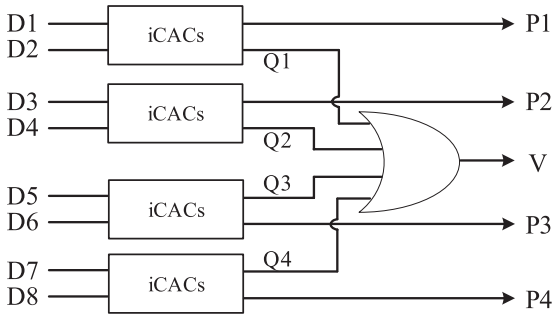

Fig. 3 Structure of an ATC-8.

where i denotes the PP position from the LSB.

In order to evaluate the performance of approximate arithmetic circuits, error distance (ED) and mean ED (MED) were proposed in [13] and normalized MED (NMED), relative ED (RED), and mean RED (MRED) were proposed in [4]. ED is defined as the arithmetic difference between the accurate output (S) and the approximate output (S') : ED=|S-S'|. MED is the average of EDs for a set of outputs. NMED is defined as NMED=MED/S_{max}, where S_{max} is the maximum magnitude of the output of an accurate circuit. RED is the ED divided by the accurate output (RED=|S-S'|/S), and MRED is the average of REDs that can be obtained similarly as MED.

Using ED, MED, and NMED, the accuracy of the ATC is discussed. From (1) and Fig. 3, the accurate sum of each bit position can be achieved by $P_i + Q_i$, and the approximate sum of each bit position can be achieved by $P_i + V_i$. ED_i is defined as the difference with the same meaning of each bit position; therefore, the ED and ED_i of the ATC are calculated as follows:

$$ED_i = (Q_i - V_i) \times f_i, \tag{4}$$

$$ED = \sum_{i=1}^{2n-1} ED_i, \tag{5}$$

where f_i is an accuracy sensitivity factor defined as follows:

$$f_i = 2^{i-1}. \tag{6}$$

The value of V_i differs from Q_i when two or more Q1, Q2, $\cdots$, Q$(n/2)$ are 1. The maximum value of V_i is 1; therefore, the maximum ED of each bit position ($(ED_i)_{max}$) and the maximum ED of ATC (ED_{max}) are calculated as follows:

$$(ED_i)_{max} = ((Q_i)_{max} - 1) \times f_i, \tag{7}$$

$$ED_{max} = \sum_{i=1}^{2n-1} (ED_i)_{max}. \tag{8}$$

$(Q_i)_{max}$ is defined as follows:

$$(Q_i)_{max} = \begin{cases} \frac{m}{2}, & \text{while } m \text{ is even number} \\ \frac{m-1}{2}, & \text{while } m \text{ is odd number} \end{cases}, \tag{9}$$

$$m = \begin{cases} i, & 1 \le i \le n \\ 2n - i, & n < i < 2n \end{cases}. \tag{10}$$

If an 8-bit multiplier is considered, it is easy to show that $(\mathrm{ED}_i)_{\max}$ is smallest (=0) when i = 15, 14, 13, 3, 2, and 1, and largest (=2048) when i = 12. $\mathrm{ED}_{\max}$ is 5208 when all bits of both inputs to the multiplier are 1. As seen, there are no errors at the positions of the three most significant bits (MSBs) and the three LSBs.

Here, assume that two inputs to an n-bit multiplier are distributed uniformly. The ATC then takes two PP rows as a group to generate a row of Q and a row of P. Therefore, the following hold:

(a) There are $n/2$ groups in total.
(b) Error occurs when $\mathrm{Q}_i > 1$.
(c) The number of combinations is $(2^2 \times 2^2 - 1)^{((\mathrm{Q}_i)_{\max} - \mathrm{Q}_i)}$ when $\mathrm{Q1}_i, \mathrm{Q2}_i, \cdots$, and $\mathrm{Q}(n/2)_i$ have values.
(d) The number of combinations is $(2^2 \times 2^2)^{(\frac{n}{2} - (\mathrm{Q}_i)_{\max})}$ when $\mathrm{Q1}_i, \mathrm{Q2}_i, \cdots$, and $\mathrm{Q}(n/2)_i$ do not have values.

If we define error type (ET) as $\mathrm{Q}_i = 2, 3, 4, \cdots, (\mathrm{Q}_i)_{\max}$, we can also calculate the number of combinations of ET for each bit position as follows:

$$\mathrm{ET}_{i,\mathrm{Q}_i} = \begin{cases} {}_{(\mathrm{Q}_i)_{\max}}\mathrm{C}_{\mathrm{Q}_i} \times 16^{(\frac{n}{2} - (\mathrm{Q}_i)_{\max})} \times 15^{((\mathrm{Q}_i)_{\max} - \mathrm{Q}_i)}, \\ \qquad 1 < \mathrm{Q}_i \le (\mathrm{Q}_i)_{\max} \\ 0, \qquad 0 < \mathrm{Q}_i \le 1 \end{cases} \tag{11}$$

Note the C in ${}_{(\mathrm{Q}_i)_{\max}}\mathrm{C}_{\mathrm{Q}_i}$ indicates a mathematical combination.

The sum of the average ED of each bit position is equivalent to the MED defined in [13]. The MED of each bit position (MED_i) and the MED of ATC are defined as follows:

$$\mathrm{MED}_i = \frac{\sum_{\mathrm{Q}_i=2}^{(\mathrm{Q}_i)_{\max}} (\mathrm{ET}_{i,\mathrm{Q}_i} \times (\mathrm{Q}_i - 1) \times f_i)}{2^{2n}}, \tag{12}$$

$$\mathrm{MED} = \sum_{i=1}^{2n-1} \mathrm{MED}_i. \tag{13}$$

The NMED can be expressed as follows:

$$\mathrm{NMED} = \frac{\mathrm{MED}}{(2^n - 1)^2}. \tag{14}$$

The results of $\mathrm{ET}_{i,\mathrm{Q}_i}$ and MED_i for an 8×8 ATC are shown in Table 2. As a result, the NMED is 0.0383%.

3.2 Approximate Multipliers

We refer to the approximate multiplier that utilizes the ATC as ATCM. The overall structure of our approach is shown in Fig. 4. We divide the PP reduction part of an 8-bit multiplier into three stages (i.e., Stage1, Stage2, and Stage3). Note that PP generation and the CPA are not shown. PP reduction is only described. The LSB is the bit position 0, and the MSB is the bit position 14. There are eight rows in Stage1, and $\mathrm{PP}_{(x,y)}$ represents the PP in row x and at bit position y. For example, $\mathrm{PP}_{(1,0)}$ represents the PP in the first row and at bit position 0, and $\mathrm{PP}_{(2,8)}$ represents the PP in the second row and at bit position 8. The solid rectangle in Stage1 represents an ATC-8, and the dashed rectangles represent rows of seven iCACs. Note that every row of the iCACs includes PPs that are not processed. For example, $\mathrm{PP}_{(1,0)}$ and $\mathrm{PP}_{(2,8)}$ in the row of iCACs are not processed.

In Stage1, ATC-8 compresses the number of PP rows by half. In this case, eight rows of PPs are reduced to four rows (P1, P2, P3, and P4) and to one accuracy compensation vector, V.

In Stage2, approximate 4-to-2 column compressors without carry propagation compress the halved rows and the compensation vector. In this study, we use ACCI2 [8]

Table 2 Results of $\mathrm{ET}_{i,\mathrm{Q}_i}$ and MED_i for an 8×8 ATC.

i	$\mathrm{ET}_{i,\mathrm{Q}_i}$ 2	3	4	$(\mathrm{Q}_i)_{\max}$	f_i	MED_i
1	0	0	0	0	1	0.00
2	0	0	0	1	2	0.00
3	0	0	0	1	4	0.00
4	256	0	0	2	8	0.03
5	256	0	0	2	16	0.06
6	720	16	0	3	32	0.37
7	720	16	0	3	64	0.73
8	1350	60	1	4	128	2.88
9	720	16	0	3	256	2.94
10	720	16	0	3	512	5.88
11	256	0	0	2	1024	4.00
12	256	0	0	2	2048	8.00
13	0	0	0	1	4096	0.00
14	0	0	0	1	8192	0.00
15	0	0	0	0	16384	0.00

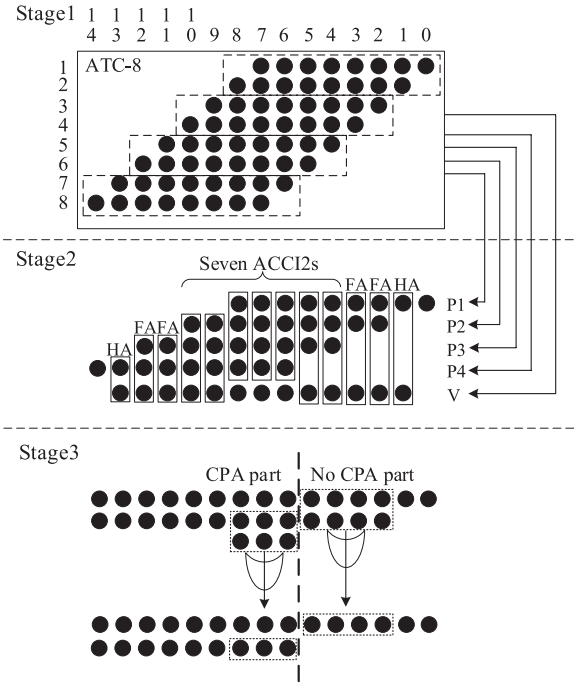

Fig. 4 Overall structure of the 8-bit approximate multiplier with OR logic accumulation in the CPA part in Stage3.

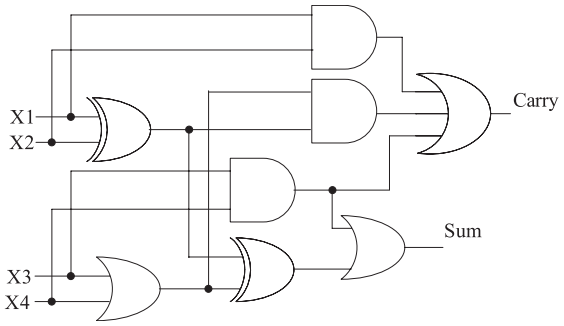

Fig. 5 ACCI2 schematic.

Table 3 Truth table for ACCI2 [8].

{C,S}		X1X2	00	01	11	10
X3X4	00		00	01	10	01
	01		01	10	11	10
	11		**11**	11	**11**	11
	10		01	10	11	10

shown in Fig. 5. The halved rows and the compensation vector are connected to the four inputs of ACCI2s (X1, X2, X3, and X4) from the top to the bottom. For example, consider the ACCI2 at the bit position 10. $P2_{10}$, $P3_{10}$, $P4_{10}$, and V_{10} are connected to X1, X2, X3, and X4, respectively. There is no P1 at the bit position 10. V_6, V_7, and V_8 are not compressed by the ACCT2. This is due to prevent these path delays from increase. In these bit positions 6 to 8, there are five bits to be compressed. Table 3 shows the truth table for an ACCI2. Because of the difference in bits in each column, Stage2 consists of seven ACCI2s, four accurate full adders, and two accurate half adders, as shown in Fig. 4.

In Stage3, to reduce the carry propagation of the CPA, lower columns from the bit positions 0 to 5 are excluded from the CPA. We refer to them as a "No CPA part", which uses OR gates to reduce columns with two bits to one bit. The CPA part is further divided into two groups of columns, i.e., three bits and two bits. We propose two techniques to reduce the former to two bits. The first technique processes the columns by OR gates. Here, the columns with two bits in the CPA part are left as they are. We refer to the approximate multiplier that uses this technique as ATCM1. It is suitable for designs that prioritize power saving over accuracy. The other technique uses accurate full adders, as shown in Fig. 6. A carry is propagated to columns with two bits in the CPA part; therefore, they are processed by half adders. We refer to the approximate multiplier that uses this technique as ATCM2. This technique is suitable for designs where accuracy loss is not desirable. As a result, a 9-bit CPA is required for the final addition in the 8-bit ATCMs.

The ATCM1 and ATCM2 were proposed in [3] and have already been evaluated. In the following, in order to improve their accuracy, we propose two modifications in Stage2. Since the ATCM1 and ATCM2 are different only in the CPA parts of Stage3, the modifications can be used in both ATCM1 and ATCM2.

The first modification changes the connection of ACCI2

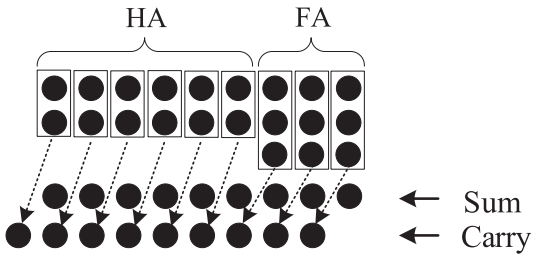

Fig. 6 The CPA part of ATCM2.

at bit position 10. Consider the column at bit position 10 in Stage1: $P2_{10}$, $P3_{10}$, $P4_{10}$, and V_{10} are generated from $PP_{(4,10)}$, $PP_{(5,10)}$, $PP_{(6,10)}$, $PP_{(7,10)}$, and $PP_{(8,10)}$, respectively, with two iCACs as follows:

$P2_{10} = PP_{(4,10)}$;
$P3_{10} = PP_{(5,10)}$ *OR* $PP_{(6,10)}$;
$P4_{10} = PP_{(7,10)}$ *OR* $PP_{(8,10)}$; and
$V_{10} = (PP_{(5,10)}$ *AND* $PP_{(6,10)})$ *OR* $(PP_{(7,10)}$ *AND* $PP_{(8,10)})$.

Remember that at the bit position 10 of ATCM1 and ATCM2, $P2_{10}$, $P3_{10}$, $P4_{10}$, and V_{10} are connected to X1, X2, X3, and X4 of the ACCI2, respectively. From Table 3, there are two approximate results in an ACCI2, with one of the approximate results of an ACCI2 occurring when X1 = X2 = 0 and X3 = X4 = 1. When $PP_{(4,10)}$, $PP_{(5,10)}$, and $PP_{(6,10)}$ are 0 and $PP_{(7,10)}$ and $PP_{(8,10)}$ are 1, $P2_{10}$ and $P3_{10}$ will be 0 and $P4_{10}$ and V_{10} will be 1. This means that the result of the ACCI2 at the bit position 10 will be approximate under that condition. To improve the accuracy of ATCM1 and ATCM2, we modify the connection between $P2_{10}$, $P3_{10}$, $P4_{10}$, and V_{10} and X1, X2, X3, and X4 of the ACCI2 at the bit position 10. This is because when $PP_{(5,10)}$, $PP_{(6,10)}$, $PP_{(7,10)}$, and $PP_{(8,10)}$ are all 0, $P3_{10}$, $P4_{10}$, and V_{10} will be 0. We modify the connection to X1 = $P4_{10}$, X2 = $P3_{10}$, X3 = $P2_{10}$, and X4 = V_{10}, so that when $PP_{(5,10)}$, $PP_{(6,10)}$, $PP_{(7,10)}$, and $PP_{(8,10)}$ are all 0, X1 = X2 = 0 will have occurred, and V_{10} (X4) = 0; therefore, X3 = X4 = 1 will be impossible. V_5, and V_9 are inputs of the ACCI2 at the bit positions 5 and 9, respectively, but both of them are generated by three iCACs. For example, V_9 is 0 only when $P2_9$, $P3_9$, and $P4_9$ are 0. This is not a condition of error nonoccurrence. The connection of ACCI2 at bit position 4 does not need to be modified, because $P1_4$, $P2_4$, $P3_4$, and V_4 are already connected to X1, X2, X3, and X4, respectively. Although changing the connection will cause the switching activities of the nets (X1, X2, X3, and X4 of the ACCI2) to be different, we believe that this minor modification will not seriously increase the design area, power consumption, or critical path delay.

The second modification replaces XOR gates in the full adders at the bit positions 2 and 3 with OR gates. This replacement causes the sum of the adder to be approximate, whereas its carry does not be changed. In the No CPA part in Stage3, sums and carries from Stage2 are processed by OR gates to generate the final output approximately. This means that even if the sums and carries from Stage2 are accurate, the final output will be approximate. For example, if the sums from ACCI2 at the bit position 4 (S_4) and full adders at

the bit positions 3 and 2 (S_3, S_2) are 0, 1, and 0, respectively, and the carries from the full adders at the bit positions 3 and 2 (C_3, C_2) and the half adder at the bit position 1 (C_1) are 0, 1, and 0, respectively, the final approximate output processed by OR gates in the No CPA part will be {0,1,0} at the bit positions 4, 3, and 2, respectively, whereas the accurate output should be {1,0,0}. It is important to remember that carries are shifted to the left by one; therefore, C_3, C_2, and C_1 are at the bit positions 4, 3, and 2, respectively. C_2 = 1 and S_2 = 0 indicates that two of three inputs to the full adder at bit position 2 are 1; therefore, if we replace the XOR gates at the bit positions 2 and 3 with OR gates, the final approximate output at the bit positions 4, 3, and 2 will become {0,1,1}, respectively, which is considerably close to the accurate output ({1,0,0}). Additionally, improvements in power consumption and design area can be expected by using this modification.

The modified multipliers, which are improved in accuracy from ATCM1 and ATCM2 by using the two modifications, are referred to as ATCM1_i and ATCM2_i, respectively.

3.3 Comparison with other ATC-Based Multipliers

We have investigated other ATC-based approximate multipliers [5]–[7] as well as ATCM1 and ATCM2 [3]. They are mainly differentiated in configurability. While ATCMs do not have it, the remaining ones [5]–[7] have it and can dynamically trade accuracy for power.

Another difference is in their targets. ATCMs prioritize accuracy over power consumption. On the contrary, the one [7] prioritizes power over accuracy. Between them, the other one [5], [6] is designed to balance power and accuracy. On the condition of the same power consumption, the accuracy of ATCM2_i is the highest. On the other hand, on the condition of the same accuracy, the power consumed by the one [7] is the lowest.

The targets make their structures also different. ATCMs have a stack consisting of only one ATC and also utilize the approximate 4-2 compressors [8] and the precise full adders. In contrast, the remaining ones [5]–[7] have a stack consisting of three ATCs. Differences can be also found in how to generate the accuracy compensation vector, V. ATCMs and the one [7] utilizes OR gates and the other one [5], [6] utilize the precise full adders.

The above differences are summarized in Table 4. Simulation-based comparisons between them except for the ones proposed in this paper (ATCM1_i and ATCM2_i) can be found in [7].

Table 4 Comparison among ATC-based multipliers.

	This paper	[5], [6]	[7]
Configurability	No	Yes	Yes
ATC	1 round	3 rounds	3 rounds
V	OR gates	Full adders	OR gates

4. Experimental Results

Here, we evaluate our multipliers in terms of power consumption, critical path delay, design area, and computational accuracy. To clarify the contributions to power savings of the multipliers, we implement and evaluate the conventional Wallace tree multiplier and three previously proposed approximate multipliers [4], [8]. Two different approximate multipliers using 10-bit and 8-bit recovery vectors [4] are referred to as AMER_10b and AMER_8b, respectively, and the approximate multiplier using ACCI2 [8] is referred to as ACCI_M2. We implement ATCM1, ATCM1_i, ATCM2, ATCM2_i, AMER_8b, AMER_10b, ACCI_M2, and the Wallace tree multiplier. Every CPA used in all multipliers is implemented as a ripple carry adder.

4.1 Power, Delay, and Area Results

All seven approximate multipliers and the conventional Wallace tree multiplier are coded using Verilog HDL. We use Synopsys VCS to simulate the designs and generate the value change dump (vcd) files to precisely evaluate power consumption. We use the Synopsys Design Compiler to synthesize the multipliers with the NanGate 45 nm Open Cell Library [14] to evaluate power consumption at 0.5 GHz frequency. We also synthesize them by the timing constraints from 0.5 GHz (2 ns) to 2.0 GHz (0.5 ns) in 0.1 ns steps. The operating condition for synthesis employs typical conditions (the process factor is 1.00, the power supply is 1.1 V, and the operating temperature is 25°C). All designs are synthesized and are optimized using default compile options. All the multipliers are eight bits.

To estimate power consumption, we use the Synopsys Power Compiler with switching activity interchange format files generated from the vcd files. We prepare a random test pattern with 100,000 test vectors to objectively estimate power consumption. Considering that two inputs (A and B) of an 8-bit multiplier, the total possibilities of the input sets are 65,536. We prepare two test patterns for the multipliers to generate the vcd file. Test pattern 1 is the same as that in [3] and is generated by using nested loops in ascending order, i.e., A and B values of (0, 0), (0, 1), (0, 2), $\cdots$, (0, 511), (1, 0), (1, 1), (1, 2), $\cdots$, (1, 511), $\cdots$, (511, 0), (511, 1), (511, 2), $\cdots$, (511, 511), with the number being 65,536 in total. In this paper, we add another scenario for further investigations. Test pattern 2 is generated randomly, with the number of patterns 100,000 in total.

The results of power consumption for the multipliers are shown in Fig. 7, where the x-axis is the names of the test patterns and the y-axis is power consumption. The eight bars (from left to right) in each test pattern represent ATCM1, ATCM1_i, ATCM2, ATCM2_i, AMER_8b, AMER_10b, ACCI_M2, and Wallace. Note that test patterns do not affect static power consumption. The static power consumption is less than 10uW for every multiplier.

First, we discuss the power consumption of the multi-

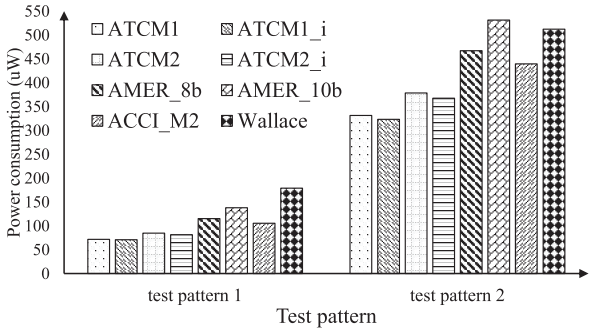

Fig. 7 Power consumption results.

pliers when the test pattern 1 is used. ATCM1_i delivers the lowest power consumption among all multipliers. Compared with the Wallace tree multiplier, it delivers 60.5% power savings. The power consumption of ATCM1 is slightly larger than that of ATCM1_i, and ATCM2_i. ATCM2 delivers 54.5% and 52.7% power savings as compared with the Wallace tree multiplier, respectively. AMER_10b consumes ∼2-fold more power than ATCM1_i. ACCI_M2 shows good result relative to power consumption with its power savings larger than that of AMER_8b.

The multipliers consume more power in the case of the test pattern 2 than in the case of the test pattern 1. This is because the differences between two consecutive inputs in the test pattern 1 are much smaller than those in the test pattern 2 and thus the switching activity in the case of the pattern 1 is also much smaller than that in the case of the pattern 2. As expected, ATCM1_i delivers the lowest power consumption and reduces power consumption by 36.9% as compared with the Wallace tree multiplier. Additionally, ATCM1_i and ATCM2_i consume less power than ATCM1 and ATCM2, respectively. There results show that the modifications achieve the improvements in power consumption. The order of power consumption of the multipliers in the case of the test pattern 2 from small to large is ATCM1_i, ATCM1, ATCM2_i, ATCM2, ACCI_M2, AMER_8b, Wallace, and AMER_10b. That order is almost the same to the case of the test pattern 1, except for AMER_10b. The power consumption of AMER_10b becomes larger than that of the Wallace multiplier, when the test pattern changes from 1 to 2.

ATCM1_i consumes ∼4.6-times larger power and the other approximate multipliers consume ∼4-times larger power, when the test pattern changes from 1 to 2. Overall, the increasing rates of the power consumption are ∼4-fold in the approximate multipliers. In contrast, the increasing rate is ∼2.9-fold in Wallace tree multiplier. The reason why the increase in power from the case using pattern 1 to that using 2 is larger in the approximate multipliers than in Wallace tree multiplier is as follows. All evaluated approximate multipliers reduce PP by using OR gates to replace some XOR gates in adders. In general, switching activity is higher for XOR gates than for OR gates. A switching transition at an input of an XOR gate always results in a transition at its output, unless there is a simultaneous transition at another input. On the other hand, an OR gate has a controlling input of 1, resulting in the insensitized at another input. Once an intermediate value turns to 1, it masks the following switching transitions. As explained above, inputs of the test pattern 1 are delivered in an ascending order, that means switching transitions at inputs of the multipliers between two consecutive test vectors are rare. This affects OR gates in the PP reduction stage to be less switching transitions at their outputs. In contrast, even if transitions at inputs are rare, XOR gates in Wallace tree have more transitions in their outputs than OR gates in the approximate multipliers due to the above reason. Using the test pattern 2 increases the transitions both in the approximate multipliers and in Wallace tree multiplier with the same tendency. Because the switching activity is higher in Wallace tree multiplier than in the approximate multipliers when the pattern 1 is used, the increase rate in the activity from the cases using the pattern 1 to that using 2 is larger in the approximate multipliers than in Wallace tree multiplier. The switching activity closely affects power consumption since static power is constant, so that the increase in power between the cases using the two patterns is larger in the approximate multipliers than in Wallace tree multiplier. To demonstrate this, we use Power Compiler to report the switching activities of all nets of the multipliers. Table 5 shows the total switching activities of ATCM1_i and Wallace as an example. Here, we assume that each net in the two multipliers has the same capacity; therefore, the increasing rate of power consumption is roughly estimated by total switching activity. As shown, the total switching activity for test pattern 2 is 4.8-fold that of the test pattern 1 for ATCM1_i, and the activity for the test pattern 2 was 2.9-fold that of test pattern 1 for Wallace tree multiplier.

Table 5 Total toggle rate of ATCM1_1 and Wallace.

Multipliers	Test pattern	Total switching activity (sum of all nets)
ATCM1_i	1	12.30
	2	63.63
Wallace	1	32.02
	2	93.68

The comparison results for critical path delay and design area for the different multipliers are shown in Fig. 8, where the y-axis represents the percentages of reduction from the conventional Wallace tree multiplier. Larger values represent better results. The two bars (from left to right) in each multiplier show the percentages of reduction in critical path delay and design area. As shown, the results for the critical path delay and design area for ATCM1_i and ATCM2_i are better than those for ATCM1 and ATCM2, respectively. The improvements in design area for ATCM1_i and ATCM2_i are a consequence of replacing XOR gates with OR gates. By analyzing critical path delay reports in detail, one difference in critical path delay between ATCM1_i and ATCM1 is found from the generation circuit of $PP_{(4,10)}$ (X3), V_{10} (X4), and the internal OR gate with X3 and X4 inputs of the ACCI2 at the bit position 10. These circuits are optimized

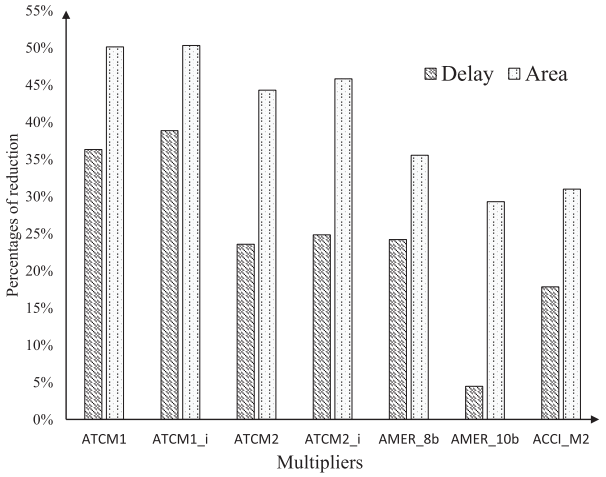

Fig. 8 Reduction rates (%) in delay and area.

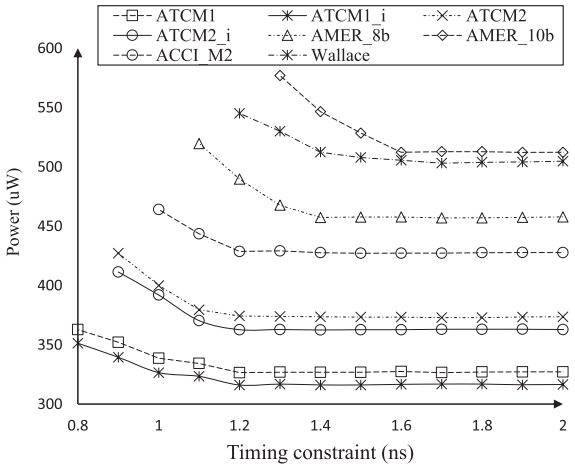

Fig. 9 Power results using different timing constraints in synthesis.

as an "AOIxx" gate while maintaining an independent AND gate in the path of ATCM1. Note that there is one module in each design of ATCM1 and ATCM1_i, and that generated netlists are optimized by design area rather than path delay as a default; therefore, this is an accidental result.

ATCM1_i delivers the largest delay reduction of 38.9% among the seven approximate multipliers. This improvement is achieved along with a 50.3% reduction in design area. Note that these results are compared to those of the Wallace tree multiplier. ATCM2_i mitigates the reduction rates of critical path delay and design area, delivering 24.8% and 45.8% reductions, respectively. AMER_8b delivers 35% savings in design area, whereas the improvement in critical path delay is comparable to that of ATCM2. The improvements in critical path delay and in design area with ACCI_M2 are the smallest among the multipliers, except for AMER_10b.

The results of power consumption and design area in the cases of the different timing constraints for synthesis are shown in Figs. 9 and 10, respectively. In the evaluations, the test pattern 2 is used. The results of synthesis with timing violations (a minus value of slack) are not plotted. As shown, the design areas of ATCM1_i and ACTM1 are highly similar and smaller than others, and the power consumption of ATCM1_i is the lowest among all multipliers. Additionally, ATCM1_i and ATCM1 can be synthesized without timing violations, with even the timing constraint lowered to 0.8ns. From this perspective, it is observed that the critical paths of ATCM1_i and ATCM1 are the shortest among all eight multipliers. Although different timing constraints give us different results for design area and power consumption, the differences are small. Overall, our approximate multipliers deliver the largest improvements in power consumption, critical path delay, and design area in all cases of different timing constraints. As expected, as the timing constraints for each multiplier becomes smaller (stricter), both of the design areas and power consumption become larger.

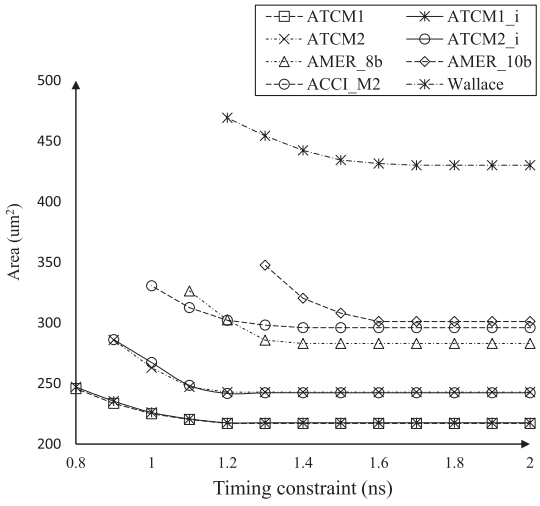

Fig. 10 Area results using different timing constraints in synthesis.

4.2 Accuracy Results

Three metrics of NMED, MRED, and error rate (ER) are used to evaluate the multipliers. ER is the percentage of inaccurate outputs among all outputs generated from all combinations of inputs. We use Synopsys VCS to evaluate the numerical outputs of all multipliers. Because 8-bit multipliers were evaluated, the total number of test patterns was 65,536 (the previous test pattern 1).

Table 6 compares the accuracy results. Both the NMED and MRED results with ACCI_M2 are the smallest among the five approximate multipliers. ATCM1_i and ATCM2_i deliver better NMED and MRED results than ATCM1 and ATCM2 do, respectively. Except for ACCI_M2, ATCM2_i delivers the smallest NMED and the second smallest MRED results among the multipliers. As expected, the results of AMER_10b are better than AMER_8b. ATCM1 delivers the largest NMED and MRED among the multipliers; however, they are very close to those of ATCM2 and AMER_8b and

Table 6 Accuracy comparison.

Multipliers	NMED (%)	MRED (%)	ER (%)
ATCM1	0.28	1.64	55.44
ATCM1_i	0.25	1.43	57.14
ATCM2	0.21	1.21	47.33
ATCM2_i	0.17	0.98	50.01
AMER_8b	0.24	1.16	51.42
AMER_10b	0.20	0.62	31.59
ACCI_M2	0.04	0.62	72.29

Table 7 Range of input values each interval.

Interval	Range of A	Interval	Range of A
1	0 ~ 31	5	128 ~ 159
2	32 ~ 63	6	160 ~ 191
3	64 ~ 95	7	192 ~223
4	96 ~ 127	8	224 ~ 255

may be sufficiently accurate for error-tolerant applications. The probability of error occurrence of ATCM1 is much less than that of ACCI_M2. AMER_10b delivers the best result relative to ER.

We also evaluate NMED, MRED, and ER using the test pattern 2. We observe small differences in NMED (< 0.02%), MRED (< 0.2%), and ER (< 1%) for each multiplier between the cases of the test patterns 1 and 2, demonstrating that the coverage of possible patterns of the test pattern 2 is sufficient, and that the results of power consumption evaluated using the test pattern 2 are reliable.

The accuracy of approximate multipliers should meet the quality requirements of applications. Ranges of input values in a multiplier will be different for different program phases and/or for different applications. This means that the accuracy of an approximate multiplier should be investigated dependent upon the ranges of input values for different program phases and/or for different applications. We analyze the distribution of error occurrence using RED, which is intuitive enough to reflect relationships between accurate and approximate products for comparing the accuracy of the approximate multipliers in detail. Consider an 8×8 multiplier, with two inputs, A and B, and output C. The inputs range from 0 to 255, and output ranges from 0 to 65,025 (unsigned values), with a possible number of outputs at 65,536. We categorize the input A in eight intervals, with the range of input B from 0 to 255 for each interval. For example, the ranges of input values for A and B are from 0 to 31 and 0 to 255, respectively, for interval one; the ranges of input values for A and B are from 32 to 63 and 0 to 255, respectively, for interval two; and the ranges of input values for A and B are from 224 to 255 and 0 to 255, respectively, for interval eight. Table 7 shows the ranges of input values for each interval, with their average RED results calculated and shown in Fig. 11. The x-axis represents the intervals from one to eight, and the y-axis shows the average RED, with the seven bars (from left to right) in each interval representing ATCM1, ATCM1_i, ATCM2, ATCM2_i, AMER_8b, AMER_10b, and ACCI_M2. Lower values (bars) represent better results.

We can see that that ATCM1_i and ATCM2_i deliver better results in accuracy than ATCM1 and ATCM2, highlighted by the improvements in intervals seven and eight. ACCI_M2 gives us the smallest average RED values from intervals three to eight, however, its results are not good for intervals one and two. Although the average RED for AMER_10b is the second smallest from intervals three to eight, it shows a balanced result. The difference in the average RED between AMER_10b and ACCI_M2 is small from intervals three to eight. ATCM2_i is not good for interval four; however, its average RED is the smallest for interval two. Except for ATCM1 and ATCM2, although ATCM1_i obtains the largest average RED from intervals two to eight, it is better than AMER_8b for interval one.

5. Image Processing

We use an image sharpening application [15] and an image multiplication application [12] to evaluate the multipliers. Both applications are popular in the evaluation of approximate multipliers. All input images for the two applications are 512×512 grayscale bitmap images with 8-bit pixels, and only the multiplications are approximate, whereas other operations (addition, subtraction, and division) used in the two applications are accurate.

The processed image quality was measured by peak signal noise ratio (PSNR), which is usually used to measure the quality of a reconstructive process involving information loss and is defined by the mean square error (MSE) [12]. The MSE and PSNR were defined in [12] as:

$$\mathrm{MSE} = \frac{1}{mp}\sum_{i=0}^{m-1}\sum_{j=0}^{p-1}[L(i,j) - K(i,j)]^2, \tag{15}$$

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{MAX_L^2}{\mathrm{MSE}}\right), \tag{16}$$

where $L(i, j)$ and $K(i, j)$ are the correct and obtained values, respectively, of each pixel, m and p are the image dimensions, and MAX_L represents the maximum value of each pixel (255 here, as the images are 8-bit).

In an image sharpening application [15], assume I is the original image, S is the processed image, and x and y are the coordinates. The sharpening algorithm is defined as:

$$S(x,y) = 2I(x,y) - \frac{1}{273}\sum_{i=-2}^{2}\sum_{j=-2}^{2} G(i+3, j+3)I(x-i, y-i), \tag{17}$$

where G is a 5×5 kernel, given as:

$$G = \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

In an image multiplication application [12], two images

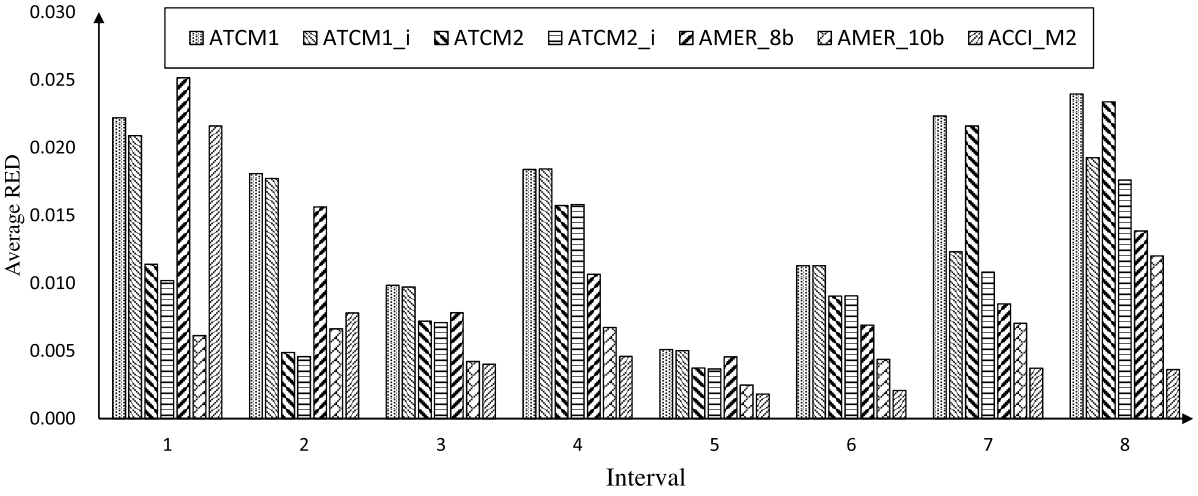

Fig. 11 Average RED distribution in approximate multipliers.

Table 8 PSNR results of approximate multipliers, in dB.

Image No.	Application	Description	ATCM1	ATCM1_i	ATCM2	ATCM2_i	AMER_8b	AMER_10b	ACCI_M2
1	Image sharpening	Lena	45.8	46.02	48.69	49.02	45.15	52.38	48.13
2		Some peppers	50.65	51.28	52.39	53.2	46.48	57	51.11
3		A bridge	51.64	52.36	52.65	53.54	46.76	56.84	50.89
4		A domed palace	48.66	49.02	51.94	52.53	45.35	55.31	49.15
5		A truck on the grassland	48.52	49.02	52.54	53.34	49.38	57.27	51.35
6		A house and a car	48.76	49.32	52.15	53.03	46.83	56.72	50.99
7	Image Multiplication	A tank on the grassland	50.42	50.5	52.82	52.95	45.14	45.73	58.41
8		Four hyenas							
9		A fighter aircraft	47.77	47.81	51.56	51.65	51.38	52.42	59.95
10		The moon							
11		A F-16 fighter	43.25	48.45	43.25	49.26	44.89	45.4	56.55
12		Small ripples							

Fig. 12 Image multiplication, (a) image No. 7, (b) image No. 8, (c) processed image by accurate multiplier.

are multiplied on a pixel by pixel basis, with the obtained 16-bit result divided by 255 to meet the range of an 8-bit pixel; therefore, the result blends the two images into a single output image, as shown in Fig. 12.

Twelve images collected from the Internet are used for image sharpening and multiplication. Table 8 shows the PSNR results in dB. Larger values represent better quality images.

As shown, in both the image sharpening and image multiplication applications, PSNR results for ATCM2_i and ATCM1_i are larger than those for ATCM2 and ATCM1, respectively. These results demonstrate that the improvements in accuracy are also effective at the application level. The PSNR result for AMER_10b in each row in the image sharpening application is the best, whereas ATCM2_i and ATCM2 obtained the second and third best PSNR results in each row in the image sharpening application. Except for the PSNR result for image No. 5, the result for ATCM1 is better than that for AMER_8b, and AMER_8b gives us the worst PSNR result. This result is consistent with the average RED results for interval one (Fig. 11). Although the range of input values for interval one is from 0 to 31 and the largest value of the 5×5 kernel was 41, the largest value appears only once, with the rest of the 24 values all within a range from 0 to 31. This result demonstrates that accuracy analysis for an approximate multiplier using MRED is insufficient, and that it is necessary to analyze accuracy using different ranges of input values for such approximate applications. The range of inputs for the multipliers for the image multiplication application are from interval one to eight, which differs from that for the image sharpening algorithm. As expected, ACCI_M2 delivers the best PSNR results in the image multiplication application. For the approximate multipliers, we observe that the PSNR results for all multipliers were >43 dB. This indicates that all of the approximate multipliers are sufficiently accurate, because the PSNR values over 40 dB are considered good in lossy image compression [16].

The processed images following image sharpening (image No. 1) and image multiplication (images No. 7 and No. 8) are shown in Figs. 13 and 14. As shown, all approximate multipliers result in visually indistinguishable images from

Fig. 13 Processed images using the image sharpening algorithm, (a) Wallace (accurate), (b) ATCM1, (c) ATCM1_i, (d) ATCM2, (e) ATCM2_i, (f) AMER_8b, (g) AMER_10b, and (h) ACCI_M2.

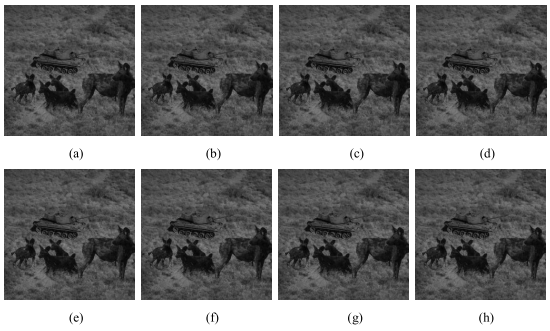

Fig. 14 Processed images using the image multiplication, (a) Wallace (accurate), (b) ATCM1, (c) ATCM1_i, (d) ATCM2, (e) ATCM2_i, (f) AMER_8b, (g) AMER_10b, and (h) ACCI_M2.

the accurately processed ones.

6. Conclusions

This study designed and analyzed four approximate multipliers, two of which were proposed in this paper. All multipliers achieved lower power consumption and smaller critical path delays than the conventional multiplier. First, we introduced iCAC, which plays an important role in maintaining high levels of accuracy. Second, we built an ATC using the iCAC, with the complexity of the multiplier circuit greatly reduced through the use of an innovative ATC. We then constructed the approximate multipliers, to exploit the positive characteristics of ATC. Last, we evaluated our multipliers at the circuit and application levels, with experimental results demonstrating that the proposed multipliers delivered significant power savings and speed-up while maintaining circuit areas smaller than those of the conventional Wallace tree multiplier. Furthermore, the best results of the proposed multipliers showed that they often improved power, delay, and area with the accuracy not inferior to the state-of-the-art approximate multipliers. We also found that the quality of the evaluated applications was enough tolerable to the human sense.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number JP17K00088 and by funds (No.175007 and No.177005) from the Central Research Institute of Fukuoka University. This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc.

References

[1] S. Venkataramani, V.K. Chippa, S.T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp.1–12, Dec. 2013. doi:10.1145/2540708.2540710

[2] B. Moons and M. Verhelst, "DVAS: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing," IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), pp.237–242, July 2015. doi:10.1109/ISLPED.2015.7273520

[3] T. Yang, T. Ukezono, and T. Sato, "Low-power and high-speed approximate multiplier design with a tree compressor," 35th IEEE International Conference on Computer Design (ICCD), pp.89–96, Nov. 2017. doi:10.1109/ICCD.2017.22

[4] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery, design," Automation & Test in Europe Conference & Exhibition (DATE), March 2014. doi:10.7873/DATE.2014.108

[5] T. Yang, T. Ukezono, and T. Sato, "A low-power high-speed accuracy-controllable approximate multiplier design," 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), pp.605–610, Jan. 2018. doi:10.1109/ASPDAC.2018.8297389

[6] T. Yang, T. Ukezono, and T. Sato, "Design and analysis of a low-power high-speed accuracy-controllable approximate multiplier," IEICE Trans. Fundamentals, vol.E101-A, no.12, pp.2244–2253, Dec. 2018.

[7] H. Baba, T. Yang, M. Inoue, K. Tajima, T. Ukezono, and T. Sato, "A low-power and small-area multiplier for accuracy-scalable approximate computing," IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp.569–574, July 2018. doi:10.1109/ISVLSI.2018.00109

[8] Z. Yang, J. Han, and F. Lombardi, "Approximate compressors for error-resilient multiplier design," IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), pp.183–186, Oct. 2015. doi:10.1109/DFT.2015.7315159

[9] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol.32, no.1, pp.124–137, Jan. 2013. doi:10.1109/TCAD.2012.2217962

[10] H.R. Mahdiani, A. Ahmadi, S.M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications," IEEE Trans. Circuits Syst. I, Reg. Papers, vol.57, no.4, pp.850–862, April 2010. doi:10.1109/TCSI.2009.2027626

[11] S. Hashemi, R.I. Bahar, and S. Reda, "DRUM: A dynamic range unbiased multiplier for approximate applications," IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp.418–425, Nov. 2015. doi:10.1109/ICCAD.2015.7372600

[12] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," IEEE Trans. Comput., vol.64, no.4, pp.984–994, April 2015. doi:10.1109/TC.2014.2308214

[13] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of

approximate and probabilistic adders," IEEE Trans. Comput., vol.62, no.9, pp.1760–1771, Sept. 2013. doi:10.1109/TC.2012.146
[14] NanGate, Inc. NanGate FreePDK45 Open Cell Library, http://www.nangate.com/?page_id=2325, 2008
[15] M.S. Lau, K.V. Ling, and Y.C. Chu, "Energy-aware probabilistic multiplier: Design and analysis," 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, pp.281–290, Oct. 2009. doi:10.1145/1629395.1629434
[16] D. Bull, Communicating Pictures - A Course in Image and Video Coding, Academic Press, 2014.

Tongxin Yang received B.S. degree from Harbin University of Science and Technology, China and joined Logic Research Co., Ltd, Japan in 2008. Since 2016, he has been working toward the Ph.D. degree in Information and Control Systems, Fukuoka University, Japan. His research interests include approximate computing systems and low power circuits and systems.

Tomoaki Ukezono graduated from the School of Information Science, Japan Advanced Institute of Science and Technology (JAIST). He received the Ph.D. degree from JAIST in 2010. Tomoaki joined Center for Highly Dependable Embedded Systems Technology, JAIST as researchers in 2010. Tomoaki joined School of Information Science, JAIST as assistant professors in 2011. Currently, Tomoaki is with Department of Electronics Engineering and Computer Science, Fukuoka University as assistant professors from 2015. His current research interests include computer architecture and operating system. Tomoaki is a member of the Institute of Electronics, Information and Communication Engineers (IEICE) of Japan.

Toshinori Sato received the B.S., M.S., and Ph.D. degrees in electronic engineering from Kyoto University in 1989, 1991, and 1999, respectively. From 1991 to 1999, he was with Toshiba Corporation, where he was engaged in the research on multicore processors and the developments of embedded processors. He is currently a professor of Department of Electronics Engineering and Computer Science at Fukuoka University. He previously served on the faculty of Department of Artificial Intelligence at Kyushu Institute of Technology and of System LSI Research Center at Kyushu University. His research interests include microprocessor architecture and design methodology. He is a senior member of ACM and IEEE and a member of IEICE and IPSJ.