Small-Area and Low-Power FPGA-Based Multipliers using Approximate Elementary Modules

Yi Guo[1], Heming Sun[2], Shinji Kimura[1]
[1] Graduate School of Information, Production and Systems, Waseda University, Kitakyushu, Japan
[2] Waseda Research Institute for Science and Engineering, Tokyo, Japan
Email: guoyi@fuji.waseda.jp

Abstract-Approximate multiplier design is an effective technique to improve hardware performance at the cost of accuracy loss. The current approximate multipliers are mostly ASIC-based and are dedicated for one particular application. In contrast, FPGA has been an attractive choice for many applications, because of its high performance, reconfigurability, and fast development. This paper presents a novel methodology for designing approximate multipliers by employing the FPGA-based fabrics. The area and latency are significantly reduced by cutting the carry propagation path in the multiplier. Moreover, we explore higher-order multipliers on architectural space by using our proposed small-size approximate multipliers as elementary modules. For different accuracy requirements, eight configurations for approximate 8×8 multiplier are discussed. In terms of mean relative error distance (MRED), the accuracy loss of the proposed 8×8 multiplier is low as 0.17%. Compared with the exact multiplier, our proposed design can reduce area by 43.66% and power by 20.36%. The critical path latency reduction is up to 27.66%. The proposed multiplier design has a better accuracy-hardware tradeoff than other designs with com-parable accuracy.

I. Introduction

Energy efficiency is a persistent challenge for emerging applications such as multimedia processing, data mining and machine learning. For most of these applications, a huge number of computations lead to large power consumptions. There are increasing demands of power- and area-efficient designs. On the other hand, many applications have inherently error-tolerant feature, where strictly exact outputs are not necessary. Therefore, approximate computing has been considered as a potential approach to achieve significant power reduction by exploiting the exactness relaxation in error-tolerant applications [1]. A lot of multiplications are included in applications from image processing to filtering with convolution neural networks. Therefore, approximate multiplier design has attracted more and more attentions.

FPGA has been a promising platform for lots of applications because of its high energy efficiency, capability of reconfiguration and fast development round. DSP blocks on modern FPGAs can provide high performance for many arithmetic operations, such as multiplication and division. However, the usage of DSP blocks might cause performance degradation on applications [2]. Some applications could exhaust the available DSP blocks for critical operations, while other applications executing concurrently on the same FPGA will use the LUT-based operations. Therefore, despite the availability of DSP blocks, Xilinx and Intel also provide logic-based soft multipliers [3][4]. It is always inevitable to have logic-based soft multipliers along with DSP blocks.

In the multiplier, adder and compressor are the basic units to accumulate the partial products, hence many works focus on the approximation of adder and compressor. In [5], a tree compressor is introduced to reduce partial products and a carry-maskable adder is explored to dynamically configurate on the final addition. Two approximate multipliers are discussed using inexact half-adder, full-adder and 4:2 compressors with less XOR gates in [6]. In [7], a bit significance-driven logic compression with OR gates is implemented, then it is used in approximate multipliers. In [8], two approximate 4:2 compressors are proposed and utilized in a Dadda tree multiplier. A recursive construction is another approach for the approximate multiplier built from small-size multipliers. In [9], an underdesigned multiplier (UDM) is constructed from inexact 2×2 multipliers, which is proposed by simplifying its Karnaugh-Map expression. Based on [9], several variants of approximate addition and multiplication units have been discussed in [10].

However, there is little study on FPGA-based approximate multiplier design. In [11], three approximate multipliers are introduced, where n rows of partial products are accumulated in the form of two layers in parallel. In [12], an inexact 4×2 multiplier is proposed by using four LUTs. Then, approximate 4×4 multiplier and 8×8 multiplier (Cc, Ca) are constructed from 4×2 multipliers.

Because the architectural differences between ASICs and FPGAs, the savings achieved by ASIC-based defined designs might not comparably translate to FPGA-based implementation. Moreover, it is common to employ FPGA as accelerator for many applications involving a huge number of multiplications. Therefore, it is expected to design low-cost FPGA-specific approximate multipliers.

In this paper, we focus on a low-cost FPGA-based approximate multiplier design, whereas most of previous works focus on ASIC-based approximate multipliers [5]-[10]. In general, there is always a tradeoff between accuracy loss and hardware savings in approximate computing. In order to achieve a good accuracy-hardware tradeoff, we discuss the 8×8 multiplier design on architectural space using a recursive construction. Our primary contributions are as follows:

1) We propose three types of approximate 4×4 multipliers implemented with LUTs and associated carry chain. The critical path is shortened by cutting the carry propagation in the multiplier.

978-1-7281-4123-7/20/$31.00 ©2020 IEEE

2) We provide a wide-range of approximate 8×8 multipliers by using our proposed approximate 4×4 multipliers as elementary modules. Eight configurations for approximate 8×8 multipliers are presented for different performance requirements.

This paper is organized as follows. Section II presents the preliminaries required to understand the proposed approach. Three types of approximate 4×4 multipliers are proposed in Section III. Section IV introduces the approximate 8×8 multiplier design built from proposed 4×4 multipliers. The evaluations of 4×4 multipliers and 8×8 multipliers are discussed in Section V, and Section VI concludes the paper.

II. Preliminaries

In this paper, we target the devices of Xilinx 7-series FPGA family. The proposed design can also be implemented on FPGAs from other vendors, which provide 6-input LUTs and carry chains.

The configurable logic blocks (CLBs) are the main logic resources for implementations of sequential as well as combinational circuits, where a CLB consists of two slices. Each slice has four 6-input look-up tables (LUTs), eight storage elements to register the outputs of LUTs, wide-function multiplexers, and a fast 4-bit carry chain [13]. A 6-input LUT can be configurated as one of the following two implementations. One implementation is a single 6-input combinational function with a single output O as shown in Fig. 1 (a), commonly referred as LUT6. Another implementation is named as LUT6_2, which has two 5-input combinational functions with $O5$ and $O6$ outputs as shown in Fig.1 (b).

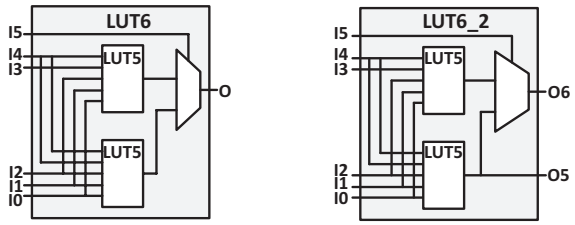
A LUT is instantiated with an INIT attribute, which describes the required truth table based on the input logic. An INIT attribute consists of 16 hexadecimal values (i.e. 64 binary values for 64 input combinations). The INIT value usually can be determined by creating a binary logic table of all input combinations and specifying the desired logic value of the output. It indicates that the logic value '1' occurs on the outputs for all 64 input combinations. For example, an INIT value of '0000000000000004' (hex) for LUT6 means that the output O is '1' for the input combination '000010'.

For combinational circuit design, the LUTs part and an associated 4-bit carry chain are generally used as shown in Fig. 2. The outputs of LUT drive the inputs of the carry chain which comprises multiplexers with bypass signals (AX/BX/CX/DX) and XOR gates. The carry chain usually implements a 4-bit carry-look ahead adder, where $O5$ is as carry-generate signal and $O6$ is as carry-propagate signal.

III. Proposed Approximate 4×4 Multipliers

To reduce the hardware consumptions of the 4×4 multiplier, we propose three novel approximate multipliers. Three designs provide different accuracy-hardware tradeoffs. The first multiplier with low-error feature is introduced in Section III-A. The second multiplier has an optimized structure on the first multiplier as introduced in Section III-B. The third multiplier aims to use only LUTs as discussed in Section III-C.

A. Approximate 4×4 multiplier 1 (AFM1)



(a) LUT6 (b) LUT6_2

Fig. 1. The structure of 6-input LUT [13].

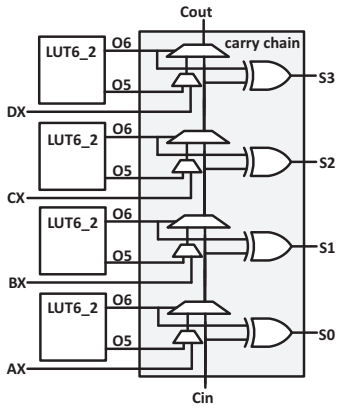


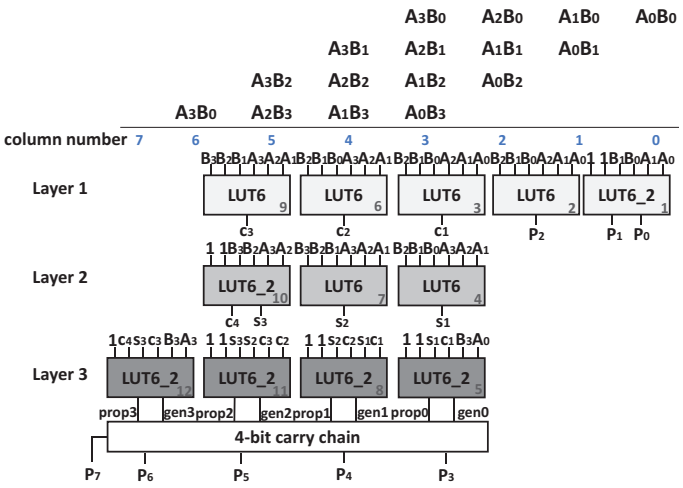
Fig. 2. The structure of carry chain [13].

The high complexity of the multiplier is usually caused by the carry propagation path. To reduce the complexity of the multiplier, we approximate the carry signal in 4×4 multiplier.

Figure 3 shows the structure and expression of approximate 4×4 multiplier 1 (AFM1), where three layers of LUTs are used to accumulate partial products and the carry chain is used to produce the final multiplication result. Layer 1 computes the carry result from the preceding column. For example, column 4 consists of six elements B_3, B_2, B_1, A_3, A_2 and A_1. LUT9 is fully used to compute the carry result from column 4, that is, six inputs of one LUT are completely employed for six elements. However, there are eight elements on column 3, which exceeds the input number of one 6-input LUT. Therefore, LUT6 inexactly computes the carry result from column 3 by ignoring one partial product of A_0B_3. In addition, LUT3 is used to compute the carry from column 0~2. When B_2, B_1, B_0, A_2, A_1 and A_0 on columns 1 and 2 all are '1', the exact carry result is 2-bit '10' (bin). In AFM1, LUT3 computes inexactly this carry result as 1-bit '1' (bin). Layer 2 generates the sum result for the current column, while Layer 3 produces the carry-propagate and carry-generate signals for the associated carry chain. Particularly, to fully utilize the LUT resource, a LUT6_2 with two outputs (i.e. LUT10) is employed in Layer 2 to generate both the carry result and the sum result from column 5. The total number of LUTs is 12, and the critical path is occupied by 2 LUTs and a 4-bit carry chain.

In AFM1, Layer 1 is approximate and other two layers are accurate. Table I illustrates the error occurrences for AFM1. The maximum error magnitude is '8' for all input combinations and the error probability is 0.0156 (= 4/256) for a uniform and independent distribution.

B. Approximate 4×4 multiplier 2 (AFM2)



LUT	INIT value (Hex)	Expression
LUT1	6AC06AC0A0A0A0A0	$P_0 = A_0B_0$ $P_1 = (A_1B_0)\oplus(A_0B_1)$
LUT2	1E665AAAB4CCF000	$P_2 = (A_1A_0B_1B_0)\oplus(A_2B_0)\oplus(A_1B_1)\oplus(A_0B_2)$
LUT3	E888A000C8000000	$c_1 = A_2A_1B_1B_0+A_2A_0B_2B_0+A_1A_0B_2B_1+A_1A_0B_1B_0$
LUT4	96665AAA3CCCF000	$s_1 = (A_3B_0)\oplus(A_2B_1)\oplus(A_1B_2)$
LUT5	8778877808800880	$prop_0 = (s_1\oplus c_1)\oplus(A_0B_3)$ $gen_0 = (s_1\oplus c_1)(A_0B_3)$
LUT6	E888A000C0000000	$c_2 = A_3A_2B_1B_0+A_3A_1B_2B_0+A_2A_1B_2B_1$
LUT7	96665AAA3CCCF000	$s_2 = (A_3B_1)\oplus(A_2B_2)\oplus(A_1B_3)$
LUT8	8778877808800880	$prop_1 = (s_2\oplus c_2)\oplus(s_1c_1)$ $gen_1 = (s_2\oplus c_2)(s_1c_1)$
LUT9	E888A000C0000000	$c_3 = A_3A_2B_2B_1+A_3A_1B_3B_1+A_2A_1B_3B_2$
LUT10	800080006AC06AC0	$s_3 = (A_3B_2)\oplus(A_2B_3)$ $c_4 = A_3B_2A_2B_3$
LUT11	936C936C20802080	$prop_2 = (s_3\oplus c_3)\oplus(s_2c_2)$ $gen_2 = (s_3\oplus c_3)(s_2c_2)$
LUT12	87777888F8888000	$prop_3 = (s_3c_3)\oplus(A_3B_3)\oplus c_4$ $gen_3 = (s_3c_3)\oplus(A_3B_3)c_4 + s_3c_3A_3B_3$

Fig. 3. The structure and expression of AFM1. Layer 1 computes the carry result from the preceding column while Layer 2 generates the sum result for current column. Layer 3 produces the carry-propagate and carry-generate signals for carry chain.

TABLE I. ERROR OCCURRENCES FOR AFM1

Input combination	Exact result	Approximate result	Difference
7×7	49	41	8
7×15	105	97	8
15×7	105	97	8
15×15	225	217	8

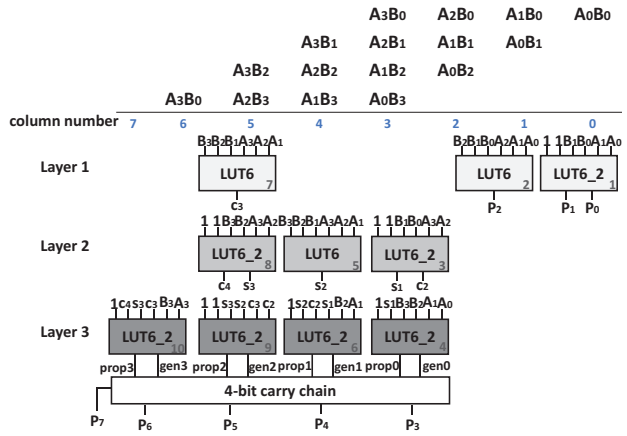


Fig. 4. The structure of AFM2.

To further reduce the area, the second design is proposed based on AFM1.

Figure 4 shows approximate 4×4 multiplier 2 (AFM2) which is optimized on AFM1. AFM2 has the similar structure with AFM1, where the carry result from preceding column and the sum result for the current column is computed by Layer 1 and Layer 2, respectively. Because the probability is low that the carry signal is generated from columns 1 and 2, the carry from columns 1 and 2 is omitted by eliminating LUT3 in AFM1. In addition, in AFM2, the carry result from column 3 is computed inexactly as c_2 by LUT3, hence LUT6 is AFM1 is omitted. Table II shows the input and output configurations for each LUT in AFM2. The total area is 10 LUTs, and the critical path involves 2 LUTs and a carry chain.

TABLE II. INPUT AND OUTPUT CONFIGURATIONS FOR LUTS IN AFM2

LUT	Input configuration						Output configuration		INIT value (Hex)
	I5	I4	I3	I2	I1	I0	O6	O5	
LUT1	1	1	B_1	B_0	A_1	A_0	P_1	P_0	6AC06AC0A0A0A0A0
LUT2	B_2	B_1	B_0	A_2	A_1	A_0	P_2		1E665AAAB4CCF000
LUT3	1	1	B_1	B_0	A_3	A_2	s_1	c_2	6AC06AC080008000
LUT4	1	s_1	B_3	B_2	A_1	A_0	$prop_0$	gen_0	953F6AC02A008000
LUT5	B_3	B_2	B_1	A_3	A_2	A_1	s_2		96665AAA3CCCF000
LUT6	1	s_2	c_2	s_1	B_2	A_1	$prop_1$	gen_1	807F7F8000808000
LUT7	B_3	B_2	B_1	A_3	A_2	A_1	c_3		E888A000C0000000
LUT8	1	1	B_3	B_2	A_3	A_2	c_4	s_3	800080006AC06AC0
LUT9	1	1	s_3	s_2	c_3	c_2	$prop_2$	gen_2	936c936c20802080
LUT10	1	c_4	s_3	c_3	B_3	A_3	$prop_3$	gen_3	87777888F8888000

C. Approximate 4×4 multiplier 3 (AFM3)

To further improve the latency saving in approximate multiplier, approximate 4×4 multiplier 3 (AFM3) is proposed by removing the associated carry chain. The result of 4×4 multiplier is computed by eight LUTs.

Figure 5 shows the structure of AFM3, which only consists of eight LUTs. In each LUT, the carry result from preceding column is computed inexactly by AND gates as shown as the shadow part in Fig. 5. The results of P_0, P_1,..., P_7 are computed in parallel and the critical path is shortened to 2 LUTs.

IV. PROPOSED APPROXIMATE 8×8 MULTIPLIERS BY USING 4×4 MULTIPLIERS AS ELEMENTARY MODULES

In general, a 2n×2n multiplier (denoted as $A \times B$) can be built from four n×n multipliers as described by

$$A \times B = (A_H \times 2^n + A_L) \times (B_H \times 2^n + B_L)$$
$$= A_H \times B_H \times 2^{2n} + (A_H \times B_L + A_L \times B_H) \times 2^n + A_L \times B_L. \quad (1)$$

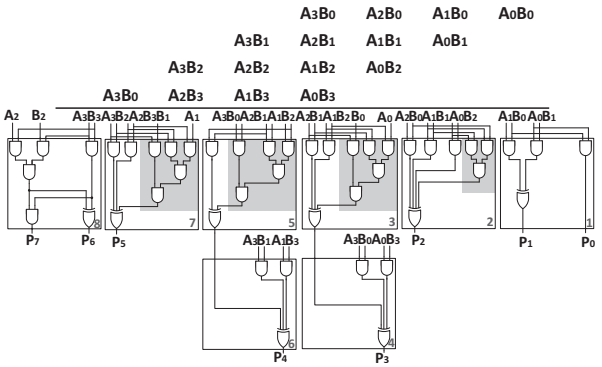


Fig. 5. The structure of AFM3. The result is computed without carry chain. The shadow part in LUT indicates the carry from preceding column.

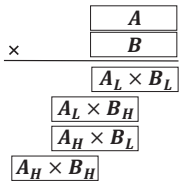


Fig. 6. Building higher-order multiplier from four small-size multipliers.

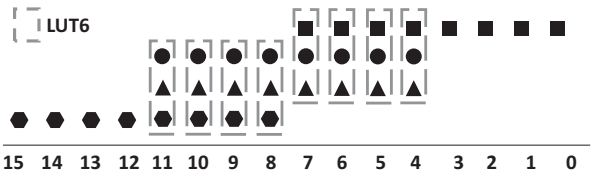


Fig. 7. Inexact addition implemented by eight LUTs generates the results of columns 4 to 11. The dots of ■, ●, ▲ and ⬣ indicates the result from $A_L \times B_L$, $A_L \times B_H$, $A_H \times B_L$, and $A_H \times B_H$, respectively.

Figure 6 shows the construction for a higher-order multiplier from four small-size multipliers. We utilize this recursive approach to explore 8×8 multiplier design from 4×4 multipliers. Firstly, four 8-bit products are generated from four 4×4 multipliers. Then, additions are used to sum four 8-bit products. The exact addition for summing four 8-bit products consists of nine LUTs and three associated carry chains [12]. The latency of exact addition is large because of the serial carry propagation path. To lower the hardware consumptions of the addition, especially latency, we propose an inexact addition to compute the results of each column in parallel as shown in Fig. 7. Approximation in the proposed inexact addition is cutting the carry propagation between two adjacent columns. OR operation is used to compute the inexact sum result. The results of columns 4 to 11 are computed by eight LUTs, where each LUT is configurated as OR operation. The INIT value for each LUT is 'FEFEFEFEFEFEFEFE' (hex). The latency is efficiently reduced by cutting the carry propagation.

V. Experiment Results and Discussion

A. Experiment setup

To clarify the contributions of the proposed multipliers, the proposed design was compared with the Vivado default exact multiplier, Xilinx multiplier IP (exact) [4], and approximate multipliers in [9] (UDM), [10] (C3), [11] (SMA3), [12] (Cc, Ca).

For accuracy analysis, approximate multipliers were evaluated in terms of mean error distance (MED), mean relative error distance (MRED) and error rate (ER). The error distance (ED) is the arithmetic difference between the exact result (Y) and the inexact result ($\tilde{Y}$). MED is the average value of EDs for all input combinations. The relative error distance (RED) is defined as $\mathrm{RED} = \mathrm{ED}/\mathrm{Y}$ and the average value of REDs is MRED. ER is the percentage of the erroneous result produced by inexact multipliers among all results. The functional models of propsed multipliers were implemented using Matlab and an exhaustive simulation was performed.

For evaluation of area, latency and power, the proposed design was implemented in Verilog and synthesized using Xilinx Vivado 18.3 for XC7VX485T device of Virtex-7 family. We synthesized approximate multipliers UDM, C3, SMA3, Cc and Ca using the open-source codes provided by [10]-[12], respectively. All multipliers were synthesized and optimized in the same environment with default options. To precisely evaluate power, we reported power results with switching activity interchange format file (.saif) from post-implementation functional simulation.

TABLE III. Accuracy Comparison of 4×4 Multipliers

Designs	MED	MRED (%)	ER (%)
AFM1	**0.125**	**0.14**	**1.56**
AFM2	1.500	2.94	17.19
AFM3	11.250	13.53	32.81
UDM [9]	3.125	2.61	19.14
C3 [10]	4.688	13.97	46.48
SMA3 [11]	10.750	12.34	35.94
Ca [12]	0.188	0.24	2.34

TABLE IV. Area, Latency and Power of 4×4 multipliers

Designs	Area [LUTs]	Latency (ns)	Power (W)
Exact	16	6.412	0.311
Xilinx Multiplier IP	15	6.765	0.308
AFM1	12	6.763	0.306
AFM2	10	6.804	0.304
AFM3	**8**	**5.650**	0.294
UDM [9]	13	6.451	0.298
C3 [10]	15	7.013	**0.290**
SMA3 [11]	**8**	6.214	0.293
Ca [12]	12	7.058	0.307

B. Evaluation of 4×4 multipliers

Table III shows the accuracy comparison of approximate 4×4 multipliers. The proposed multiplier AFM1 achieves the lowest accuracy loss in terms of MED, MRED and ER. The hardware performance is shown in Table IV. The proposed AFM3 has the smallest area and the shortest latency. Combining with the results in Table III and IV, it is can be observed that

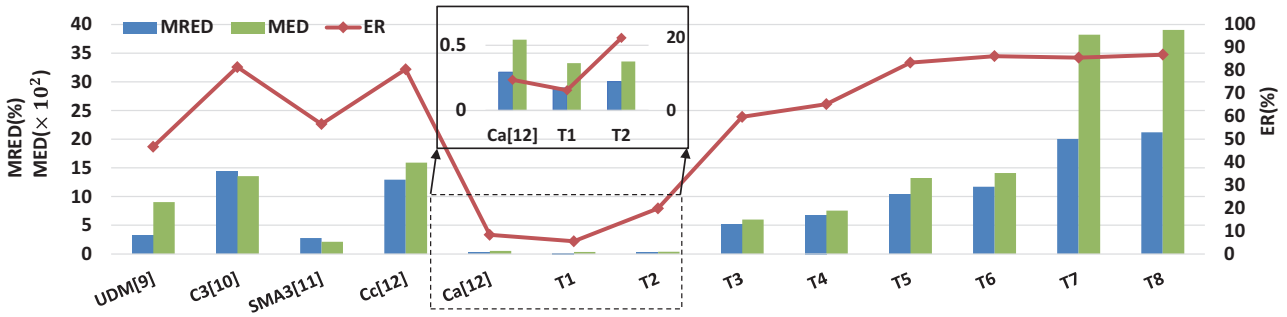
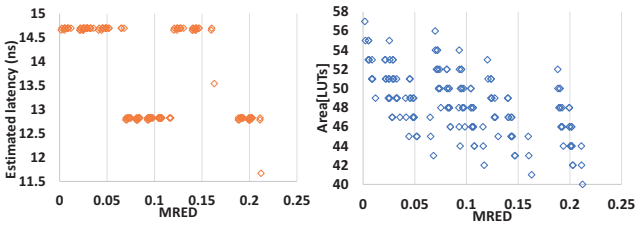


Fig. 10. Accuracy comparison of approximate 8×8 multipliers

TABLE V. AREA AND LATENCY OF EXACT AND INEXACT ADDITONS

Designs	Area [LUTs]	Latency (ns)
Exact addition	9	7.892
Inexact addition	8	6.020



(a) (b)

Fig. 9. The estimated latency and area of all configurations for the 8×8 multiplier. (a) Estimated latency vs. MRED. (b) Area vs. MRED.

TABLE VI. EIGHT CONFIGURATIONS FOR PROPOSED 8×8 MULTIPLIER

Designs	Configuration				
	$A_H \times B_H$	$A_H \times B_L$	$A_L \times B_H$	$A_L \times B_L$	addition
T1	AFM1	AFM1	AFM1	AFM1	exact
T2	AFM1	AFM1	AFM1	AFM2	exact
T3	AFM2	AFM2	AFM3	AFM3	exact
T4	AFM2	AFM3	AFM3	AFM3	exact
T5	AFM2	AFM1	AFM3	AFM1	inexact
T6	AFM2	AFM3	AFM3	AFM3	inexact
T7	AFM3	AFM1	AFM3	AFM3	inexact
T8	AFM3	AFM3	AFM3	AFM3	inexact

the proposed 4×4 design outperforms other approximate multipliers with comparable accuracy. For example, for AFM1 and Ca, the number of LUTs are both 12. The latency and power of AFM1 is smaller than those of Ca. In addition, the accuracy loss of AFM1 is lower than that of Ca.

C. Evaluation of 8×8 multipliers

For the design of 8×8 multipliers, we discuss all possible configurations building from 4×4 multipliers. Firstly, we evaluate the latency and area of exact/inexact addition. Then, the latency and area of all configurations are estimated. Finally, eight configurations for 8×8 multiplier are presented.

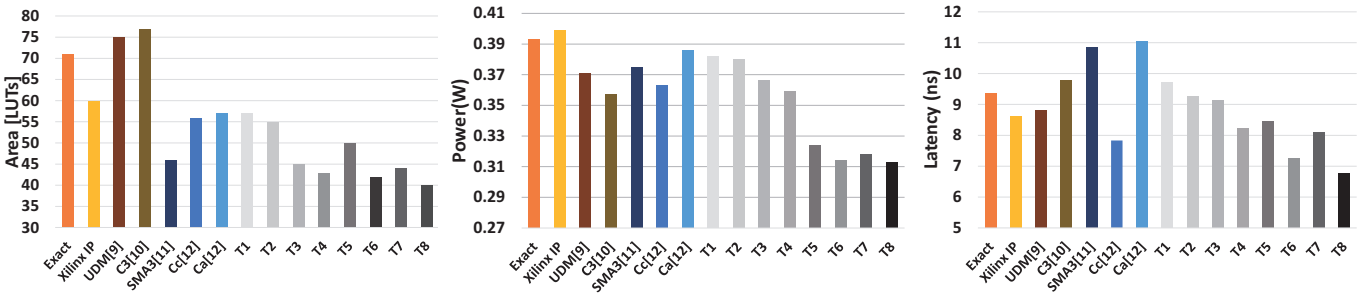
The latency and area of exact and proposed inexact additions are shown in Table V. The latency of inexact addition is smaller than that of exact one. By using the structure shown in Fig. 6, the area of the 8×8 multiplier can be calculated as the summation of total LUTs of four 4×4 multipliers with the area of addition. The latency of the 8×8 multiplier is estimated as the summation of the largest latency among four 4×4 multipliers with the latency of the addition. Figure 9 shows the estimated latency and area of all configurations for the 8×8 multipliers. The operation on $A_H \times B_H$ is most significant, which determines the dot position of each configuration in Fig. 9. The different accuracy-hardware can be achieved by different configurations on 8×8 multiplier. We select eight configurations for the proposed 8×8 multiplier as illustrated in Table VI.

Figure 10 shows the accuracy comparison for the approximate 8×8 multipliers. The proposed design has the wide-range of accuracy, which provides several choices for applications with different accuracy requirements. The proposed T1 has the lowest accuracy loss among all approximate designs, which employs AFM1 as 4×4 elementary multiplier. The MRED of T1 is low as 0.17% and the ER is 5.49%.

The area, power and latency of the exact multipliers and approximate multiplier with 8-bit input are shown in Fig. 11. The proposed multiplier T8 achieves the lowest area, power and latency among all multipliers. Compared with the exact multiplier, T8 has the area saving of 43.66%, power saving of 20.36% and latency saving of 27.66%. UDM and C3 both focus on logic gates, whose area are even larger than the exact multiplier on FPGA-based implementation. T1 and Ca have the similar accuracy loss, yet the latency of Ca is larger than that of T1. Overall, the proposed design T3 and T5 are suggested for error-tolerant applications, where T3 has advantage for applications with high-accuracy requirements and T5 is better for area- and power-efficient designs.

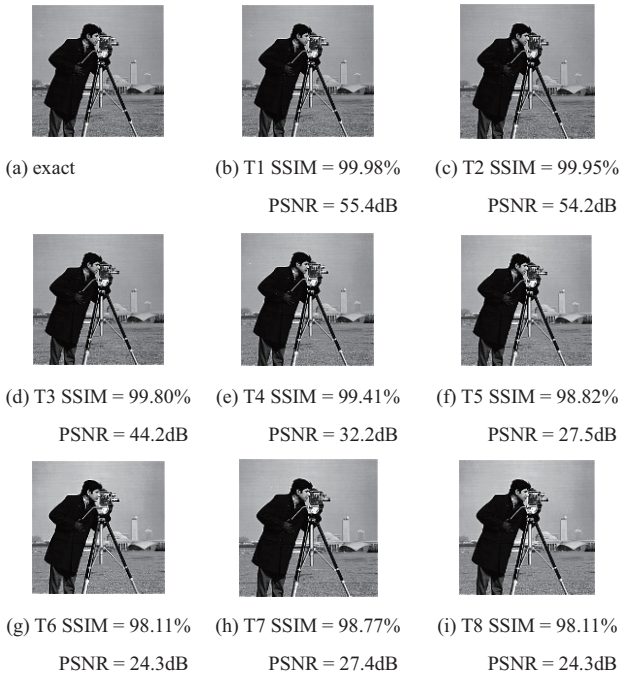
D. Image processing application

The image sharpening algorithm [14] is widely utilized to evaluate approximate multipliers. The peak signal-to-noise ratio (PSNR) is a metric to measure the quality of processed images as defined in [8]. In addition, a well-established metric structural similarity index (SSIM) is another metric and we used the Matlab function *ssim* to calculate it. Figure 12 shows the processed images from the exact multiplier and proposed approximate multipliers. The difference is imperceptible among the images processed by exact multiplier and the proposed multiplier. The PSNR and SSIM results are also



(a) (b) (c)

Fig. 11. Hardware performance for the exact 8×8 multiplier and approximate 8×8 multipliers. (a) Area (b) Power (c) Latency.



(a) exact
(b) T1 SSIM = 99.98% PSNR = 55.4dB
(c) T2 SSIM = 99.95% PSNR = 54.2dB
(d) T3 SSIM = 99.80% PSNR = 44.2dB
(e) T4 SSIM = 99.41% PSNR = 32.2dB
(f) T5 SSIM = 98.82% PSNR = 27.5dB
(g) T6 SSIM = 98.11% PSNR = 24.3dB
(h) T7 SSIM = 98.77% PSNR = 27.4dB
(i) T8 SSIM = 98.11% PSNR = 24.3dB

Fig. 12. Processed imaged by exact and approximate multipliers.

presented in Fig. 12. The SSIM results of T1, T2, T3, and T4 are higher than 99.0%, because the exact addition is used to sum the results from four small-size multipliers. In general, a PSNR of 20dB can be regarded as acceptable [15], the images processed by the proposed multipliers all are sufficiently exact for error-tolerant application.

VI. Conclusion

In this paper, we propose a FPGA-based approximate 8×8 multiplier design to achieve lower hardware consumptions than exact multiplier. We firstly propose approximate 4×4 multipliers with a shorter critical path and small number of LUTs. Then, the design of 8×8 multipliers is explored by using proposed 4×4 multipliers as elementary modules. A wide-range of approximate 8×8 multipliers is provided for different accuracy-hardware requirements. The experimental results demonstrate that the proposed multiplier design delivers more hardware reduction than previous designs with comparable accuracy.

Acknowledgments

This work was partly executed under the cooperation of organization between Waseda University and KIOXIA Corporation (former Toshiba Memory Corporation). The work was supported in part by Grants-Aid for Scientific Research from JSPS and a research fund from NEC. The work of Y. Guo was supported by the China Scholarship Council scholarship. The authors convey their sincere gratitude.

References

[1] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proceedings of the IEEE European Test Symposium (ETS)*, pp. 1-6, 2013.

[2] I. Kuon, and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 2, pp. 203-215, 2007.

[3] Intel, Integer Arithmetic IP Cores User Guide, https://www.altera.com /en_US/pdfs/literature/ug/ug_lpm_alt_mfug.pdf, 2017.

[4] Xilinx, LogiCORE IP Multiplier v11.2., https://www.xilinx.com/ support/documentation/ip_documentation/mult_gen_ds255.pdf, 2017.

[5] T. Yang, T. Ukezono, and T. Sato, "A low-power high-speed accuracy-controllable approximate multiplier design," in *Asia and South Pacific Design Automation Conference (ASP-DAC),* pp. 605-610, 2018.

[6] S. Venkatachalam and S. B. Ko, "Design of power and area efficient approximate multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 25, no. 5, pp. 1782-1786, 2017.

[7] I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov, and A. Yakovlev, "Energy-efficient approximate multiplier design using bit significance-driven logic compression," in *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*, pp. 7-12, 2017.

[8] A. Momeni, J. Han, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 984-994, 2015.

[9] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *International Conference on VLSI Design* , pp. 346-351, 2011.

[10] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel, "Architectural-space exploration of approximate multipliers," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp.1-8, 2016.

[11] S. Ullah, S. S. Murthy, and A. Kumar, "SMApproxlib: library of FPGA-based approximate multipliers," in *ACM/ESDA/IEEE Design utomation Conference (DAC)*, pp. 1-6, 2018.

[12] S. Ullah, S. Rehman, B. S. Prabakaran, F. Kriebel, M. A. Hanif, M. Shafique, and A. Kumar, A, "Area-optimized low-latency approximate multipliers for FPGA-based hardware accelerators," in *Proceedings of the 55th Annual Design Automation Conference (DAC)*, Article No.159, 2018.

[13] Xilinx, 7 Series FPGAs Configurable Logic Block User Guide, https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf, 2016.

[14] M. S. K. Lau, K. V. Lin, and Y. C. Chu, "Energy-aware probabilistic multiplier: design and analysis," in *Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems*, pp. 281-209, 2009.

[15] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, "RoBA multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 2, pp.393-401, 2017.