Majority Logic-based Approximate Recording Adders for High-radix Booth Multipliers

Tingting Zhang [1], Honglan Jiang [2], Weiqiang Liu [3], Fabrizio Lombardi [4], Leibo Liu [5], and Jie Han [1]

***Abstract*— Approximate computing permits the improvement of hardware efficiency with a relaxation of accuracy for nanoscale technologies and devices. Instead of conventional Boolean logic, voter-based majority logic (ML) is widely applicable to many emerging nanotechnologies. High-radix Booth multipliers, such as radix-8 and radix-16 multipliers, suffer from high complexity when generating odd multiples of the multiplicand. In this paper, designs of approximate recording adders (ARAs) based on ML with no carry propagation are proposed to alleviate this issue. For calculating the triple and $5\times$ of the multiplicand, a 2-bit ARA and a 3-bit ARA are designed to compute the sum of $1\times$ and $2\times$ multiplicand, and the sum of $1\times$ and $4\times$ multiplicand, respectively. Moreover, a 4-bit ARA is especially developed for computing $7\times$ of the multiplicand as the addition of $-1\times$ and $8\times$ of the multiplicand. The proposed ARAs show advantages in hardware evaluated by delay and gate complexity, as well as in accuracy; for example, in a 16×16 radix-8 multiplier, the use of 2-bit ARAs achieves a reduction of 77% in the area-delay product with a normalized mean error distance of 7.51×10^{-4} for computing the triple multiplicand.**

I. Introduction

As Dennard scaling is expected to come to an end soon, it becomes more difficult to reduce power dissipation by scaling down the feature size of CMOS transistors. Emerging nanotechnologies have been investigated for low power and high speed in digital circuit design. These nanotechnologies often rely on majority logic (ML) as a computing primitive, such as quantum-dot cellular automata, spin-wave devices and nanomagnetic logic [1]. Moreover, minority logic-based nanotechnologies, such as single electron tunneling and tunneling phase logic, share similar logic synthesis for ML-based circuits. The widely used basic unit in ML is the 3-input majority gate (or voter), defined by the logic function, $F = M(A, B, C) = AB + BC + AC$.

*This work was supported by NSERC (RES0048688), and NSFC (62022041 and 62104127). The research of Fabrizio Lombardi was supported by NSF under CCF-1953961 and 1812467 grants. T. Zhang was supported by a Ph.D. scholarship from the China Scholarship Council.

[1] T. Zhang and J. Han are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada ttzhang@ualberta.ca, jhan8@ualberta.ca.

[2] H. Jiang is with the Department of Micro-Nano Electronic, Shanghai Jiao Tong University, Shanghai 200240, China honglan@sjtu.edu.cn.

[3] W. Liu is with College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China liuweiqiang@nuaa.edu.cn.

[4] F. Lombardi is with the Department of Electrical and Computer Engineering, Northeastern University, Boston MA 02115, USA lombardi@ece.neu.edu.

[5] L. Liu is with the School of Integrated Circuits, Tsinghua University, Beijing 100084, China liulb@tsinghua.edu.cn.

Approximate computing provides an emerging paradigm for energy efficient circuit design [2]. The conjunction of approximate computing and ML-based emerging nanotechnology can lead to low power integrated circuits and systems [3]. Approximate designs based on conventional Boolean logic for CMOS circuits have been extensively explored. However, a naive mapping method directly replacing the logic gates in CMOS-based circuits with ML does not fully utilize the specific properties of ML [4]. Therefore, to better adapt to emerging nanotechnologies, efficient approximate circuits must be specially designed by manipulating ML.

Prior ML-based approximate multiplier designs include unsigned array multipliers made of approximate partial product (PP) generators and compressors [5]–[7], signed radix-4 Booth multipliers using approximate PP generation and reduction [8], [9]. The Booth algorithm is commonly utilized for efficient and fast signed multiplication [10]. In the radix-4 Booth algorithm, PPs are generated by shift operations or 2's complementing. The high-radix Booth algorithms generate fewer PPs, which reduces the complexity of accumulating the PPs [11]. However, the drawback occurs in the generation of odd multiples of the multiplicand, which can not be done by means of simple shifts. They require additions with a long carry propagation and therefore, additional power and delay. A Boolean logic-based approximate recording adder (ARA) has been designed for generating the triple multiplicand in the radix-8 Booth algorithm [12]. However, research on ML-based high radix multiplier designs has not been pursued.

In this paper, ML-based approximate recording adders are designed for high-radix Booth multipliers to generate the odd multiples of the multiplicand. 2-bit, 3-bit, and 4-bit ARAs that incur no carry propagation are, respectively, designed for calculating the triple, $5\times$ and $7\times$ of the multiplicand in radix-8 and radix-16 Booth algorithms. The ARAs are then employed to generate the lower significant bits of the odd multiples, thus overcoming the long delay issue in radix-8 and radix-16 schemes. To assess the proposed ARAs, an analytical investigation of the hardware complexity and errors is presented.

The remainder of this paper is organized as follows. Section II proposes ML-based AFA designs for radix-8 and radix-16 Booth multipliers. Section III reports the results for error and hardware metrics. Section IV concludes the paper.

II. Approximate Recording Adder Designs

A. An ARA Design for Radix-8 Booth Multipliers

For an $n\times n$ signed multiplier, let $A = a_{n-1}a_{n-2}...a_2a_1a_0$ be the multiplicand and $B = b_{n-1}b_{n-2}...b_2b_1b_0$ be the

multiplier. The most significant bits of A and B are the sign bits. In the radix-8 Booth algorithm, the multiplicand is encoded into 0, $\pm A$, $\pm 2A$, $\pm 3A$, or $\pm 4A$ to generate the PP array depending on four adjacent multiplier bits. The addition for implementing $A + 2A$ is required to calculate the triple multiplicand (i.e., $3A$), as in Fig. 1 (a). Consider each 2-bit adder as a component, it computes the addition (denoted by R) by

$$\begin{aligned} R &= 2^{i+2}C_{out} + 2^{i+1}S_{i+1} + 2^{i}S_i \\ &= 2^{i+1}a_{i+1} + 3 \times 2^{i}a_i + 2^{i}a_{i-1} + 2^{i}C_{in}, \end{aligned} \quad (1)$$

where a_{i+1}, a_i, a_{i-1} are three adjacent bits in A, C_{in} is the carry-in from the addition of lower significance, S_{i+1} and S_i are the first and second bits of the sum, and C_{out} is the carry out of the addition (omitted in Fig. 1).

(1) can be reformulated as

$$R = 2^{i+2}a_i + 2^{i+1}a_{i+1} + 2^{i}(C_{in} + a_{i-1} - a_i). \quad (2)$$

Let $\tilde{R}$ be the approximate R for the 2-bit ARA design, given by $\tilde{R} = 2^{i+2}\tilde{C}_{out} + 2^{i+1}\tilde{S}_{i+1} + 2^{i}\tilde{S}_i$, where $\tilde{C}_{out}$, $\tilde{S}_{i+1}$ and $\tilde{S}_i$ denote approximate C_{out}, approximate S_{i+1} and approximate S_i, respectively. By comparing (2) with (1), $\tilde{C}_{out}$ and $\tilde{S}_{i+1}$ can be given by

$$\tilde{C}_{out} = a_i, \quad (3)$$

$$\tilde{S}_{i+1} = a_{i+1}. \quad (4)$$

Then, we use a 1-bit $\tilde{S}_i$ to approximate $C_{in} + a_{i-1} - a_i$. Let r and $\tilde{r}$ be the exact and approximate results of $a+b-c$, respectively. As shown in Table I, there are two cases that r can not be represented by a 1-bit binary number. Then, by rounding the results in these two cases to their nearest binary numbers, $\tilde{r}$ is obtained by using a MV, as

$$\tilde{r} \cong a + b - c \cong a\&\bar{c}|a\&b|\bar{c}\&b = M(a, b, \bar{c}). \quad (5)$$

Let e be the difference between $\tilde{r}$ and r, computed by

$$\begin{aligned} e &= \tilde{r} - r = M(a, b, \bar{c}) - (a + b - c) \\ &= \begin{cases} 1, & \{a, b, c\} = \{0, 0, 1\} \\ -1, & \{a, b, c\} = \{1, 1, 0\} \\ 0, & others \end{cases}. \end{aligned} \quad (6)$$

Thus, two errors are introduced to $\tilde{r}$, with one negative and one positive, compared with r. Then, $\tilde{S}_i$ is given by

$$\tilde{S}_i = M(C_{in}, a_{i-1}, \overline{a}_i). \quad (7)$$

The error E is defined as the difference between $\tilde{R}$ and R, as $E = \tilde{R} - R$. Based on (2)-(7), E is obtained as

$$\begin{aligned} E &= 2^{i}\{M(C_{in}, a_{i-1}, \overline{a}_i) - (C_{in} + a_{i-1} - a_i)\} \\ &= \begin{cases} 2^{i}, & \{C_{in}, a_{i-1}, a_i\} = \{0, 0, 1\} \\ -2^{i}, & \{C_{in}, a_{i-1}, a_i\} = \{1, 1, 0\} \\ 0, & others \end{cases}. \end{aligned} \quad (8)$$

For two possible values of a_{i+1}, thus, the 2-bit ARA design introduces two negative and two positive errors with $|E| = 2^i$ in all input assignments.

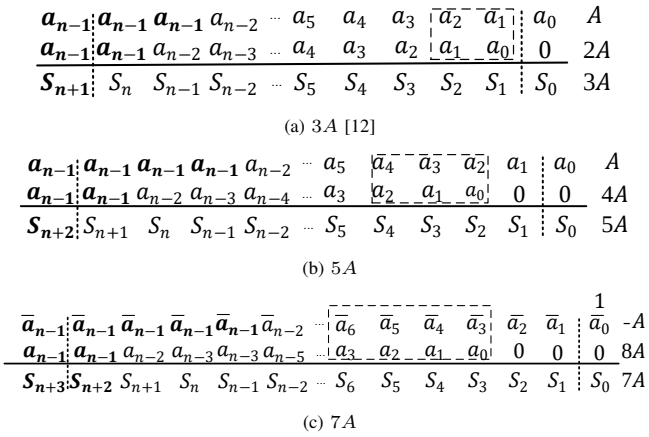


Fig. 1: Generations of odd multiples of the multiplicand for high-radix Booth multipliers.

TABLE I: The Truth Table of r and $\tilde{r}$ to Perform the Function $a + b - c$

a	b	c	r	$\tilde{r}$	e
0	0	0	0	0	0
0	0	1	**-1**	**0**	**1**
0	1	0	1	1	0
0	1	1	0	0	0
1	0	0	1	1	0
1	0	1	0	0	0
1	1	0	**2**	**1**	**-1**
1	1	1	1	1	0

B. ARA Designs for Radix-16 Booth Multipliers

The radix-16 algorithm encodes the multiplicand into 0, $\pm A$, $\pm 2A$, $\pm 3A$, $\pm 4A$, $\pm 5A$, $\pm 6A$, $\pm 7A$ or $\pm 8A$ depending on five adjacent multiplier bits. $3A$ can be approximately calculated by using the ARA design discussed in Section II-A and $6A$ can be implemented by left shifting $3A$ by one bit. Therefore, we consider the ARAs for calculating $5A$ and $7A$ next.

1) A 3-bit ARA Design for Computing $5A$: To compute $5A$, the addition of $4A$ and A is performed. As shown in Fig. 1 (b), except for the addition of the two least significant bits, let each 3-bit addition as a unit, the sum is given by

$$\begin{aligned} R = & \quad 2^{i+3}C_{out} + 2^{i}S \\ = & \quad 2^{i+3}C_{out} + 2^{i+2}S_{i+2} + 2^{i+1}S_{i+1} + 2^{i}S_i \\ = & \quad 2^{i+2}a_{i+2} + 2^{i+1}a_{i+1} + 2^{i+1}a_{i-1} \\ & + 5 \times 2^{i}a_i + 2^{i}a_{i-2} + 2^{i}C_{in}, \end{aligned} \quad (9)$$

where a_{i+2}, a_{i+1}, a_i, a_{i-1}, and a_{i-2} are five adjacent bits in A, C_{in} is the carry-in, S ($S_{i+2}S_{i+1}S_i$) is the 3-bit sum and C_{out} is the 1-bit carry out.

As $5 \times 2^{i}a_i = 2^{i+3}a_i - 2^{i+1}a_i - 2^{i}a_i$. (9) is first reformulated as

$$\begin{aligned} R = & \quad 2^{i+3}a_i + 2^{i+2}a_{i+2} + 2^{i+1}(a_{i+1} + a_{i-1} - a_i) \\ & + 2^{i}(a_{i-2} + C_{in} - a_i). \end{aligned} \quad (10)$$

Then, similar to the 2-bit ARA design, based on the

approximation methods in (3)-(5), the addition using the 3-bit ARA is approximately given by

$$\tilde{C}_{out} = a_i, \tag{11}$$

$$\tilde{S}_{i+2} = a_{i+2}, \tag{12}$$

$$\tilde{S}_{i+1} = M(a_{i+1}, a_{i-1}, \overline{a}_i), \tag{13}$$

$$\tilde{S}_i = M(a_{i-2}, C_{in}, \overline{a}_i), \tag{14}$$

where $\tilde{S}_{i+2}\tilde{S}_{i+1}\tilde{S}_i$ and $\tilde{C}_{out}$ are the approximate 3-bit S and C_{out}, respectively.

Let $\tilde{R} = 2^{i+3}\tilde{C}_{out} + 2^{i+2}\tilde{S}_{i+2} + 2^{i+1}\tilde{S}_{i+1} + 2^i\tilde{S}_i$. Then, the error E for the 3-bit ARA design is given by

$$\begin{aligned} E &= 2^{i+1}\{M(a_{i+1}, a_{i-1}, \overline{a}_i) - (a_{i+1} + a_{i-1} - a_i)\} \\ &\quad +2^i\{M(a_{i-2}, C_{in}, \overline{a}_i) - (a_{i-2} + C_{in} - a_i)\} \\ &= \begin{cases} 2^{i+1} + 2^i, & \{a_{i+1}, a_i, a_{i-1}, a_{i-2}, C_{in}\} \\ & = \{0,1,0,0,0\} \\ 2^{i+1}, & \{a_{i+1}, a_i, a_{i-1}\} = \{0,1,0\}\ and \\ & \{a_{i-2}, C_{in}\} \neq \{0,0\} \\ 2^i, & \{a_i, a_{i-2}, C_{in}\} = \{1,0,0\}\ and \\ & \{a_{i+1}, a_{i-1}\} \neq \{0,0\} \\ -2^{i+1} - 2^i, & \{a_{i+1}, a_i, a_{i-1}, a_{i-2}, C_{in}\} \\ & = \{1,0,1,1,1\} \\ -2^{i+1}, & \{a_{i+1}, a_i, a_{i-1}\} = \{1,0,1\}\ and \\ & \{a_{i-2}, C_{in}\} \neq \{1,1\} \\ -2^i, & \{a_i, a_{i-2}, C_{in}\} = \{0,1,1\}\ and \\ & \{a_{i+1}, a_{i-1}\} \neq \{1,1\} \\ 0, & others \end{cases} \end{aligned} \tag{15}$$

Thus, this 3-bit ARA introduces errors in 28 out of 64 possible cases, including two negative and two positive errors with $|E| = 2^{i+1}+2^i$, six negative and six positive errors with $|E| = 2^{i+1}$, as well as six negative and six positive errors with $|E| = 2^i$.

2) A 4-bit ARA Design for Computing $7A$: As shown in Fig. 1 (c), the addition of $8A$ and $-A$ in 2's complement is performed to generate $7A$, where the 2's complement of $-A$ is implemented by inverting each bit of A and then adding 1. Taking each 4-bit addition as a unit, R is given by

$$\begin{aligned} R = & \quad 2^{i+4}C_{out} + 2^i S \\ = & \quad 2^{i+4}C_{out} + 2^{i+3}S_{i+3} + 2^{i+2}S_{i+2} + 2^{i+1}S_{i+1} + 2^i S_i \\ = & \quad 2^{i+3}\bar{a}_{i+3} + 2^{i+3}a_i + 2^{i+2}\bar{a}_{i+2} + 2^{i+2}a_{i-1} \\ & +2^{i+1}\bar{a}_{i+1} + 2^{i+1}a_{i-2} + 2^i\bar{a}_i + 2^i a_{i-3} + 2^i C_{in}, \end{aligned} \tag{16}$$

where a_{i+3}, a_{i+2}, a_{i+1}, a_i, a_{i-1}, a_{i-2}, and a_{i-3} are seven adjacent bits in A, C_{in} is the carry-in, S ($S_{i+3}S_{i+2}S_{i+1}S_i$) and C_{out} are the sum and the carry out, respectively.

Consider $2^i\bar{a}_i = 2^{i+3}a_i - 2^{i+3}a_i - 2^i a_i + 2^i$. (16) can be reformulated as

$$\begin{aligned} R = & \quad 2^{i+4}a_i + 2^{i+3}(\bar{a}_{i+3} - a_i) + 2^{i+2}(\bar{a}_{i+2} + a_{i-1}) \\ + & \quad 2^{i+1}(\bar{a}_{i+1} + a_{i-2}) + 2^i(C_{in} + a_{i-3} - a_i + 1). \end{aligned} \tag{17}$$

Then, for the terms in the brackets in (17) to be compatible with the format in (5), consider $2^{i+2}a_{i-1} = 2^{i+3}a_{i-1} - 2^{i+2}a_{i-1}$, $2^{i+1}a_{i-2} = 2^{i+2}a_{i-2} - 2^{i+1}a_{i-2}$, $2^i a_{i-3} = 2^{i+1}a_{i-3} - 2^i a_{i-3}$ and $\bar{a}_i$ (logic operation) is equivalent to $-a_i + 1$ (arithmetic operation), (17) can further be reformulated as

$$\begin{aligned} R = & \quad 2^{i+4}a_i + 2^{i+3}(\bar{a}_{i+3} + a_{i-1} - a_i) \\ & +2^{i+2}(\bar{a}_{i+2} + a_{i-2} - a_{i-1}) + 2^{i+1}(\bar{a}_{i+1} + a_{i-3} \\ & -a_{i-2}) + 2^i(C_{in} + \bar{a}_i - a_{i-3}). \end{aligned} \tag{18}$$

Thus, the addition using the 4-bit ARA is computed by

$$\tilde{C}_{out} = a_i \tag{19}$$

$$\tilde{S}_{i+3} = M(\bar{a}_{i+3}, a_{i-1}, \bar{a}_i), \tag{20}$$

$$\tilde{S}_{i+2} = M(\bar{a}_{i+2}, a_{i-2}, \bar{a}_{i-1}), \tag{21}$$

$$\tilde{S}_{i+1} = M(\bar{a}_{i+1}, a_{i-3}, \bar{a}_{i-2}), \tag{22}$$

$$\tilde{S}_i = M(C_{in}, \bar{a}_i, \bar{a}_{i-3}), \tag{23}$$

where $\tilde{S}_{i+3}\tilde{S}_{i+2}\tilde{S}_{i+1}\tilde{S}_i$ and $\tilde{C}_{out}$ are the approximate 4-bit S and C_{out}, respectively.

Let $\tilde{R} = 2^{i+4}\tilde{C}_{out} + 2^{i+3}\tilde{S}_{i+3} + 2^{i+2}\tilde{S}_{i+2} + 2^{i+1}\tilde{S}_{i+1} + 2^i\tilde{S}_i$. Thus, E for the 4-bit ARA design is given by

$$\begin{aligned} E &= 2^{i+3}\{M(\bar{a}_{i+3}, a_{i-1}, \bar{a}_i) - (\bar{a}_{i+3} + a_{i-1} - a_i)\} \\ &+2^{i+2}\{M(\bar{a}_{i+2}, a_{i-2}, \bar{a}_{i-1}) - (\bar{a}_{i+2} + a_{i-2} - a_{i-1})\} \\ &+2^{i+1}\{M(\bar{a}_{i+1}, a_{i-3}, \bar{a}_{i-2}) - (\bar{a}_{i+1} + a_{i-3} - a_{i-2})\} \\ &\quad +2^i\{M(C_{in}, \bar{a}_i, \bar{a}_{i-3}) - (C_{in} + \bar{a}_i - a_{i-3})\}. \end{aligned} \tag{24}$$

The errors are analyzed using a method similar to (15). Thus, errors are introduced in 176 combinations of inputs.

III. Error and Hardware Evaluation

A. Evaluation Metrics

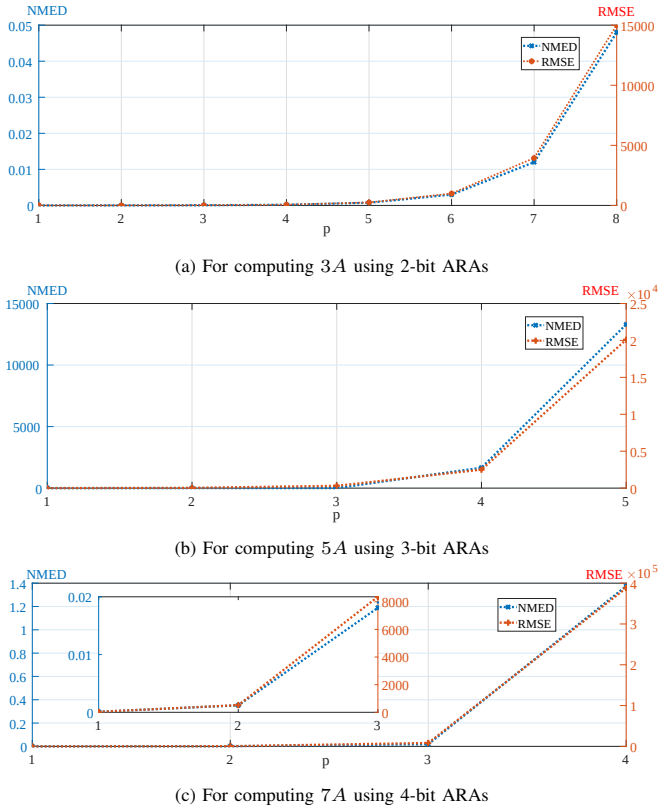
Two error metrics are considered to evaluate the approximate designs, the normalized mean error distance ($NMED$), and the root-mean-square error ($RMSE$) [13]. The $NMED$ is the normalized average of the absolute differences between the approximate and the exact results. The $RMSE$ is the root of the average squared differences between the approximate and the exact results. To adapt to different ML-based nanotechnologies, the circuit characteristics are analytically evaluated by the number of utilized majority voters (MVs), the critical path delay (D), and the area-delay product (ADP, given by $MV \times D$). The critical path delay is measured by the number of MVs on the critical path because the delay of inverters is often very small for ML-based nanotechnologies [14]. The ADP is used to assess the overall hardware efficiency.

In this section, the approximate ARA designs are compared with exact ML-based ripple carry adders (RCAs) (implemented by cascading full adders). Let the inputs be a, b and c, and the outputs be c_{out} and s. An ML based full adder performs $2c_{out}+s = a+b+c$, where $c_{out} = M(a,b,c)$ and $s = M(\bar{c}_{out}, M(a,b,\bar{c}), c)$ [15]. Thus, the ML-based full adder consists of three MVs and two inverters. There are one and two majority gate delays, respectively for generating c_{out} and s. An m-bit ML-based RCA requires $3m$ MVs with $D = m + 1$.

B. Simulation Results and Analysis

For an $n \times n$ signed Booth multiplier, let the approximate factor p denote the number of ARA blocks applied to generate odd multiples of A. As shown in Fig. 1 (a), for the radix-8 Booth algorithm, $2p$-bit addition can approximately be computed by using p blocks of 2-bit ARAs, where $0 < p \leq \lfloor \frac{n+1}{2} \rfloor$, except for the least significant bit. Compared with an exact RCA, reduction rates of $\frac{5p}{3(n+1)}$ for the number of MVs and $\frac{2p}{n+2}$ for D are obtained. In the radix-16 Booth algorithm, using p blocks of 3-bit ARAs to generate $5A$ leads to $3p$ approximate least significant bits, where $0 < p \leq \lfloor \frac{n+1}{3} \rfloor$, except for the two least significant bits, as per Fig. 1 (b). The utilization of 3-bit ARAs achieves decrease rates of $\frac{7p}{3(n+1)}$ for the number of MVs and $\frac{3p}{n+2}$ for D than using an RCA. The use of p blocks of 4-bit ARAs computes $7A$ with $4p$ approximate least significant bits, where $0 < p \leq \lfloor \frac{n+1}{4} \rfloor$, except for the three least significant bit addition, as per Fig. 1 (c). Moreover, $8p$ MVs and $4p$ units of D are saved.

For $n = 16$, Fig. 2 presents the error results measured by $NMED$ and $RMSE$ due to the use of ARAs in the generation of the odd multiples of A. When $p = 6$, 4, 3, errors for computing $3A$, $5A$ and $7A$ significantly increase. Thus, for $n = 16$, $p = 5$, 3, 2 are preferred when using 2-bit, 3-bit and 4-bit ARAs, respectively. Five 2-bit ARAs reduce the ADP by 77% with an $NMED$ of 7.51×10^{-4} and an $RMSE$ of 247.31. When using three 3-bit ARAs, a reduction of 70% in the ADP is achieved with an $NMED$ of 6.34×10^{-4} and an $RMSE$ of 314.36. Moreover, the use of two 4-bit ARAs reduces the ADP by 55% with an $NMED$ of 1.2×10^{-3} and an $RMSE$ of 522.61.



(a) For computing $3A$ using 2-bit ARAs

(b) For computing $5A$ using 3-bit ARAs

(c) For computing $7A$ using 4-bit ARAs

Fig. 2: Errors (in $NMED$ and $RMSE$) vs. p when $n = 16$.

IV. Conclusion

In this paper, approximate recording adders (ARAs) based on ML are proposed to compute the odd multiples of the multiplicand in high-radix Booth multipliers. In particular, 2-bit, 3-bit, and 4-bit ARAs are designed for calculating the triple, $5\times$, and $7\times$ of the multiplicand, respectively. They are then employed to generate the lower significant bits of odd multiples of the multiplicand. Due to the advantage of no carry propagation in these ARAs, the critical path is significantly shortened. The performance of the proposed ARAs has been analyzed using hardware and error metrics. For a 16-bit multiplication, the use of ARAs reduces up to 77% of the ADP when generating the odd multiples of the multiplicand, compared with exact computing.

References

[1] B. Parhami, D. Abedi, and G. Jaberipur, "Majority-Logic, its applications, and atomic-scale embodiments," Comput. Elect. Eng., vol. 83, no.106562, 2020.

[2] J. Han, and M. Orshansky, "Approximate computing: an emerging paradigm for energy-efficient design," in IEEE Eur. Test Symp., pp. 1-6, 2013.

[3] T. Zhang, W. Liu, E. McLarnon, M. O'Neill and F. Lombardi, "Design of majority logic (ML) based approximate full adders," in Proc. IEEE Int. Symp. Circuits Syst., pp. 1-5, 2018.

[4] G. Meuli, V. Possani, R. Singh, S. Y. Lee, A. T. Calvino et al., "Majority-based design flow for AQFP superconducting family," in Des. Autom. Test Eur. Conf. Exhib., 2022.

[5] W. Liu, T. Zhang, E. McLarnon, M. O'Neill, P. Montuschi and F. Lombardi, "Design and analysis of majority logic based approximate adders and multipliers," IEEE Trans. Emerg. Top. Comput., vol. 9, no. 3, pp. 1609-1624, 2019.

[6] F. Sabetzadeh, M. H. Moaiyeri and M. Ahmadinejad, "A majority-based imprecise multiplier for ultra-efficient approximate image multiplication," IEEE Trans. Circuits Syst. I, Regular Papers, vol. 66, no. 11, 2019.

[7] M. H. Moaiyeri, F. Sabetzadeh and S. Angizi, "An efficient majority-based compressor for approximate computing in the nano era," Microsyst. Technol., vol. 4, pp. 1-13, 2017.

[8] T. Zhang, W. Liu, J. Han and F. Lombardi, "Design and analysis of majority logic Based approximate radix-4 Booth encoders," in IEEE/ACM Int. Symp. Nanoscale Archit., pp. 1-6, 2019.

[9] T. Zhang, H. Jiang, H. Mo, W. Liu, F. Lombardi, L. Liu and J. Han, "Design of majority logic-based approximate Booth multipliers for error-tolerant applications," IEEE Trans. Nanotechnol., vol. 21, pp. 81-89, 2022.

[10] W. Yeh and C. Jen, "High-speed Booth encoded parallel multiplier design," IEEE Trans. Comput., vol. 49, no. 7, pp. 692-701, 2000.

[11] R. Riedlinger, R. Arnold, L. Biro, B. Bowhill, J. Crop et al., "A 32 nm, 3.1 billion transistor, 12 wide issue itanium processor for mission-critical servers," IEEE J. Solid-State Circuits, vol. 47, no. 1, pp. 177–193, 2012.

[12] H. Jiang, J. Han, F. Qiao and F. Lombardi, "Approximate radix-8 booth multipliers for low-power and high-performance operation," IEEE Trans. Comput., vol. 65, no. 8, pp. 2638-2644, 2015.

[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, 2013.

[14] V. Pudi, K. Sridharan and F. Lombardi, "Majority logic formulations for parallel adder designs at reduced delay and circuit complexity," IEEE Trans. Comput., vol. 66, no. 10, pp. 1824-1830, 2017.

[15] H. Cho and E. E. Swartzlander, "Adder and multiplier design in quantum-dot cellular automata," IEEE Trans. Comput., vol. 58, no. 6, pp. 721-727, 2009.