

# Energy-Efficient Approximate Wallace-Tree Multiplier using Significance-Driven Logic Compression

Issa Qiqieh\*, Rishad Shafik\*, Ghaith Tarawneh\*, Danil Sokolov\*, Shidhartha Das<sup>†</sup>, Alex Yakovlev\*

\*School of Electrical and Electronic Engineering, Newcastle University, Newcastle upon Tyne, NE1 7RU, UK,

<sup>†</sup>ARM, 110 Fulbourn Rd, Cambridge CB1 9NJ, Cambridge, UK

Emails: \*{i.qiqieh1, rishad.shafik, ghaith.tarawneh, danil.sokolov, alex.yakovlev}@newcastle.ac.uk, <sup>†</sup>Shidhartha.Das@arm.com

**Abstract**—In this paper, we propose an energy-efficient approximate multiplier design approach. Fundamental to this approach is configurable lossy logic compression, coupled with low-cost error mitigation. The logic compression is aimed at reducing the number of product rows using progressive bit significance, and thereby decreasing the number of reduction stages in Wallace-tree accumulation. This accounts for substantially lower number of logic counts and lengths of the critical paths at the cost of errors in lower significant bits. These errors are minimised through a parallel error detection logic and compensation vector. To validate the effectiveness of our approach, multiple 8-bit multipliers are designed and synthesized using Synopses Design Compiler with different logic compression levels. Post synthesis experiments showed the trade-offs between energy and accuracy for these compression levels, featuring up to 70% reduction in power-delay product (PDP) and 60% lower area in the case of a multiplier with 4-bit logic compression. These gains are achieved at a low loss of accuracy, estimated at less than 0.0554 of mean relative error. To demonstrate the impact of approximation on a real application, a case study of image convolution filter was extensively investigated, which showed up to 62% (without error compensation) and 45% (with error compensation) energy savings when processing image with a multiplier using 4-bit logic compression.

## I. INTRODUCTION

Approximate computing has been introduced as an efficient solution for achieving higher computational performance at low energy cost for imprecision-resilient applications. The basic premise of approximate computing is to replace traditional complex and energy-wasteful data processing blocks by low-complexity ones with reduced logic counts. As a result, effective chip area and energy consumption are reduced at the cost of imprecision introduced to the processed data [1].

Research has shown that the majority of modern applications could be ordered under the domain of approximate computing, such as digital signal processing, computer vision, robotics, multimedia and data analytics [2]. This can be leveraged as an opportunity for energy-efficient system design for current and future generations of application-specific systems.

Approximate arithmetic, such as approximate adders and multipliers, can be exploited as means of reducing energy requirements, increasing speed, reducing cost and increasing reliability in many of these applications [3] [4]. Multipliers are crucial arithmetic units in many of the aforementioned applications, for two major reasons. Firstly, they are characterized by complex logic design, being one of the most energy-

demanding data processing units in modern microprocessors. Secondly, compute-intensive applications typically exercise a large number of multiplication operations [5]. These factors have prompted approximate multiplier design research, since improvements made in the power/speed of a multiplier are expected to substantially influence the overall system power/performance trade-offs [6].

Recently reported multiplier design approaches can be largely categorized as modifications of either timing or functional behaviors. Timing behavior can be modified by lowering the supply voltage below its nominal value which allows for reductions in energy consumption at the cost of time-induced errors [7]. Since timing errors are caused by long carry chains, *i.e.*, impact the most significant bit of the final product, it is necessary to quantify the impact of timing violation by modifying the conventional multiplier to allow for graceful degradation [8].

Functional modifications deal with logic reduction techniques and can be performed by relaxing the need for accurate Boolean equivalence in favor of energy and circuit area reductions. For example, truncating multiplier product terms allows for the elimination of some of the least significant partial product terms [9]. As more columns are eliminated, further energy reduction is achieved; however, errors also increase. Large efficient multipliers using inaccurate small multiplier blocks is another effective technique [10] [11]; however, the hierarchical organization of small approximate blocks may not significantly reduce the critical path and also will eventually propagate more errors when increasing the size of multiplier. Automated design approaches [12]–[14] present design flows for generating approximate circuits using circuit activity profiles, quality bounds and evolutionary processes. The key principle of the above studies is to achieve reduced logic complexity, which is also the main aim of our work.

In this paper, we propose an energy efficient approximate multiplier using significance driven logic compression. Compared to our previous work in [15], we present the following new key contributions:

- 1) Incorporate a Wallace-tree accumulation method together with the significance-driven logic compression (SDLC) approach to shorten the reduction stages.
- 2) Add a parallel error detection and compensation method to minimise the impact of lossy compression.

- 3) Use a real application to extensively investigate the conflicting design trade-offs between accuracy and implementation cost (power-delay product and area).

The rest of the paper is organized as follows. Section II introduces the proposed approximate multiplier design. Section III shows the error detection and compensation method used to reduce the errors. Section IV provides the error analysis associated with different levels of logic compressions. The experimental results and design trade-offs are described in Section V. Finally, Section VI concludes the paper.

## II. PROPOSED APPROXIMATE WALLACE MULTIPLIER

In this section, we introduce an approximate Wallace-tree multiplier design approach using hardware-based logic compression techniques. The proposed approach consists of two major steps. First, systematic lossy compression is carried out using SDLC approach [15] to generate a reduced number of partial product rows. Second, Wallace method is applied to reduce the number of these rows to the height of two to be then combined using a carry propagating adder. These steps together with different logic compression levels, are described below.

1) *Significance-Driven Logic Compression (SDLC)*: According to [15], SDLC approach generates all partial products in an  $(N \times N)$  multiplier using  $N^2$  AND gates, similar to conventional multiplication. Then various sizes of logic clusters are utilized to compress a group of vertically-aligned bits within a group of successive partial products based on their progressive bit significance. Each logic cluster is able to produce an approximate but acceptable row. After that a commutative remapping technique is used to reduce the number of rows in the partial product bit-matrix. This leads to decrease the number of reduction stages in the Wallace accumulation tree and therefore reduction in hardware complexity of Wallace multiplier. To achieve lossy compression in the SDLC approach, we follow two key principles as follows.

a. *Reduction of partial product terms*: The proposed multiplier organizes the partial product terms using different sizes of logic clusters. Each logic cluster targets a group of  $d$  consecutive rows of partial products. Each of the  $l$  columns of this cluster are reduced to a single bit thanks to compression operation using a  $d$ -input OR gate. To preserve the high-order bits of the final product,  $d$ -bit OR compression is not applied everywhere. In general, each  $(d \times l)$  logic cluster is responsible for two operations: i) generating  $(d \times l)$  partial product bits within  $d$  contiguous rows, by utilizing  $(d \times l)$  AND gates. Then, ii) compressing these bits by just a row of  $l$  bits using  $l$   $d$ -input OR gates. Figure 1 demonstrates the impact of increasing depth of the logic cluster from 2- to 3- and 4-bit in the case of  $(8 \times 8)$  parallel multiplier. The lined boxes refer to a group of bits targeted by different sizes of logic clusters. As can be seen, with increased depth of logic clusters further reduction in the partial product terms is achieved. Moreover, the size of the logic clusters is decreased when going down in the partial product matrix. This permits the most significant product terms

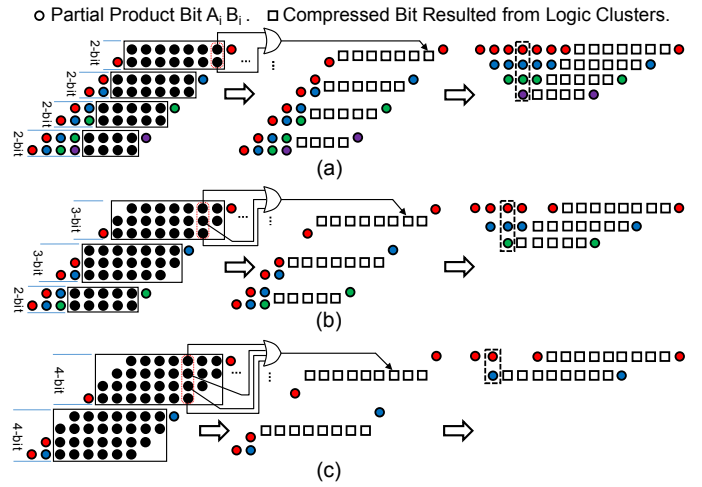


Figure 1: Stylized demonstrations of SDLC approach [15] in  $(8 \times 8)$  parallel multiplication using: (a) 2-bit, (b) 3-bit, (c) 4-bit logic compression levels.

to be accumulated on a carry-propagation basis as in the conventional multiplier, while bits with lower significance are compressed using the SDLC approach. Thus, the accuracy of the significant bits of the final product is less affected.

b. *Commutative remapping*: The reduction in the number of product terms, can be leveraged to decrease the number of rows prior to the accumulation stage. This can be achieved by remapping the partial product terms based on the commutative property of the bits, *i.e.*, bits with the same weight are gathered in the same column. For example, in the case of 2-bit logic cluster, the height of the critical column is reduced by half compared to the accurate accumulation tree. Figure 1 (the dot diagrams at right) shows the ordered bit-matrices after applying commutative remapping of the bit sequence resulting from the SDLC approach. The height of the critical columns are further reduced with increased logic compression depth.

2) *Wallace-tree accumulation Method*: One of well-known fast multiplier is the column compression multipliers presented by Wallace [16]. This multiplier consists of three consecutive phases: partial product formation, accumulation, and carry propagating adder (CPA), as follows.

a. *Wallace tree with variable logic clusters*: In general,  $(N \times N)$  traditional Wallace multiplier begins to group  $N$  rows together in sets of three rows each. Any additional rows that are not a member of a group of three are transferred to the next level without modification. Within each group of three rows, (3, 2) counters (full adders) are applied to the columns containing three bits and (2, 2) counters (half adders) are applied to the columns containing two bits. Columns containing only a single bit are transferred to the next level unchanged. By doing so, the partial product bit-matrix is then column-wise accumulated to a height of two. These two rows are combined using a CPA.

Since the hardware complexity of Wallace multiplier depends on the number of partial product rows in accumulation tree [17], SDLC approach can be employed as an efficient way to reduce the height of the rows in partial product matrix. As such, the number of the reduction stages in Wallace

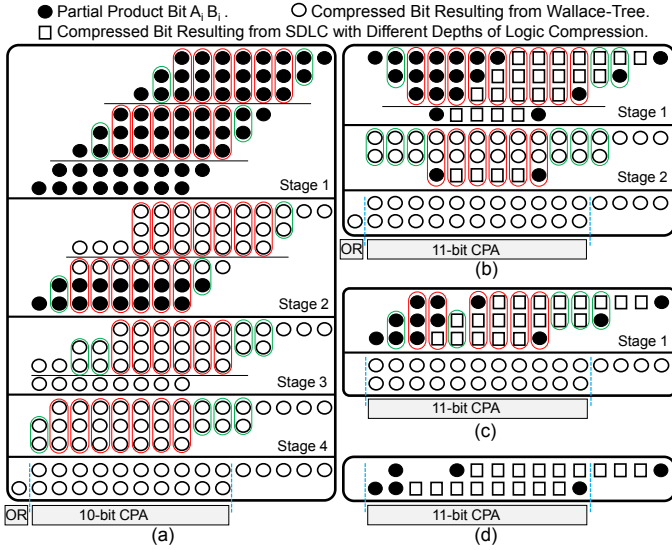


Figure 2: Reduction stages of an  $(8 \times 8)$  multiplier: (a) Traditional Wallace tree; (b), (c) and (d) Proposed Wallace tree coupled with SDLC approach for 2-bit, 3-bit and 4-bit logic compression levels.

accumulation method is deterministically reduced.

Figure 2 illustrates the impact of using different degrees of logic compression on the number of reduction stages needed by Wallace method in the case of  $(8 \times 8)$  Multiplier. As can be seen, four reduction stages are required in the case of traditional Wallace multiplier. Stage 1 reduces form 8 rows to 6 rows, then stage 2 from 6 rows to 4 rows, then 4 to 3 and finally, 3 to 2 in stage 4. This can be achieved by utilizing total of 38 (3, 2) counters, 15 (2, 2) counters, and a 10-bit CPA with 1 OR gate used to form the 16-bit product. The number of reduction stages required by Wallace accumulation method is reduced when applying the SDLC approach with different logic compression levels, thereby the hardware complexity in terms of (3, 2) and (2, 2) counters is substantially decreased. For instance, with 2-bit logic clusters, two reduction stages with matrix heights of 3 and 2, this requires total number of 15 (3, 2) counters and 9 (2, 2) counters. These numbers are further reduced to only 6 (3, 2) counters and 5 (2, 2) counters for 3-bit logic clusters. No reduction stages needed with 4-bit logic compression.

For an  $(N \times N)$  Wallace-tree multiplier, the height of the matrix in the  $k^{th}$  reduction stage,  $\alpha_k$  is given by the following recursive equations:

$$\alpha_0 = \begin{cases} N, & \text{for traditional Wallace-tree multiplier} \\ \lceil \frac{N}{d} \rceil, & \text{for SDLC with } d\text{-bit compression level} \end{cases}$$

$$\alpha_{k+1} = 2 \cdot \left\lfloor \frac{\alpha_k}{3} \right\rfloor + \alpha_k \bmod 3. \quad (1)$$

The reduction in hardware complexity achieved by SDLC leads to low switching capacitance and leakage reading as well as shortened critical paths (see Section V). Furthermore, different depths of logic compression can support the Wallace-tree multiplier with different energy-accuracy trade-offs (see Section IV).

#### b. Scalability of the proposed Wallace multiplier design:

The proposed approach is scalable for any  $(N \times N)$  multiplier, as shown in Algorithm 1. This algorithm forms all partial

product terms and apply SDLC approach to generate reduced and ordered partial product bit-matrix  $M$  as indicated in Lines (7), which can then be treated as an accumulation tree by Wallace method as indicated in Line (8). The two rows resulting from Wallace reduction stages are combined using carry propagating adder Line (9). The algorithm associated with the SDLC approach for any  $(N \times N)$  multiplier is detailed in [15].

**Algorithm 1**  $(N \times N)$  Wallace-tree multiplier using SDLC approach with  $d$ -bit logic clusters.

```

1: procedure APPROXIMATE-WALLACE( $P, A, B$ )
2:   Output:  $P[1, 2, \dots, 2N]$   $\triangleright$  Final Product bits
3:   Inputs:  $A[1, 2, \dots, N]$   $\triangleright$  Multiplicand bits
4:           $B[1, 2, \dots, N]$   $\triangleright$  Multiplier bits
5:   Initialize:  $M[1, 2, \dots, \lceil \frac{N}{d} \rceil][1, 2, \dots, 2N - 1]$   $\triangleright$  Reduced Matrix by SDLC
6:              $R[1, 2][1, 2, \dots, 2N - 1]$   $\triangleright$  Two rows combined by CPA

7:    $M \leftarrow SDLC(A, B, d)$   $\triangleright$  SDLC [15] with  $d$ -bit logic compression
8:    $R \leftarrow WallaceReduction(M)$   $\triangleright$  Reducing  $M$  to a height of two
9:    $P \leftarrow CarryPropagatingAdder(R)$   $\triangleright$  Final product is generated
10: end procedure

```

### III. ERROR COMPENSATION METHOD

The lossy compressions exercised by the logic clusters introduce errors in the final product. These errors are originated from using an array of OR gates to compress the partial product terms instead of adding them by exact adders. In theory, a two-input OR gate is sufficient to add two bits, *i.e.*,  $'0'+'1'='1'+'0'='0'+OR'1'='1'+OR'0'='1'$  and also  $'0'+'0'='0'+OR'0'='0'$ . However, the OR gate fails to give an accurate sum if the two inputs are “ones”, *i.e.*,  $'1'+'1' \neq '1'$  OR  $'1'$ , in such cases the error distance is ‘1’ as the adder returns ‘10’ and OR outputs ‘1’. When increasing the level of compression using 3- and 4- logic clusters, 3- and 4-input OR gate are also sufficient to add three and four bits together but with an increased probability of errors comparing to 2-bit depth of logic compression.

The proposed multiplier design adopts a systematic error compensation method to reduce the impact of error issued by the logic clusters. Figure 3 illustrates various logic structures used to detect such errors for 2-, 3- and 4-bit depths of logic compression. These logic structures have been designed to run in parallel with the logic clusters to generate the error signals. In the case of 2-bit logic clusters, AND gate is able to generate an error signal only when the OR gate fails to give an accurate sum, *i.e.*, both of its inputs are “ones”. Based on this situation, each error signal can be used as an error compensation bit to modify the final result. In the case of 3- and 4-bit logic clusters, 2-input OR followed by a pair of 2-input AND gates are responsible to detect the errors. Parallel

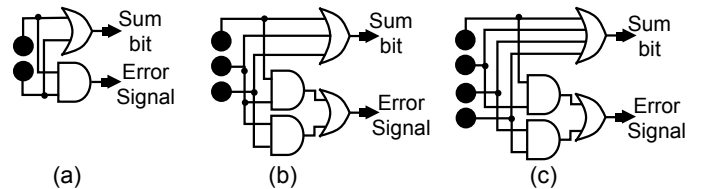


Figure 3: The error-detection logic circuit parallel with the logic clusters required by ECM method in: (a) 2-bit; (b) 3-bit; (c) 4-bit logic clusters.

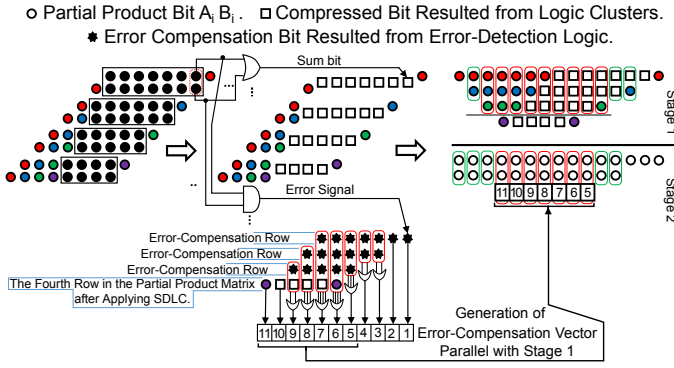


Figure 4: The proposed ECM method in 2-bit logic compression.

with logic clusters, the suggested implementations of error-detection logic can discover most of the cases that lead to error without causing any additional cost of delay.

After that, the array of error compensation bits associated with a logic cluster can be defined as error compensation row. Thus, 4, 3 and 2 error compensation rows can be generated by 2-, 3- and 4-bit logic clusters respectively. Adding any of these rows to the reduced partial product bit-matrix can mitigate the influence of the errors caused by SDLC approach. However, increasing the number of rows in Wallace accumulation tree leads to increase the hardware complexity and also the delay time for generating the final product. To this end, an efficient error compensation method (ECM) is adopted by the proposed design and described as follows.

In the case of 2-bit logic clusters, as shown in Figure 4, four rows of product terms are generated by the SDLC approach in parallel with four error compensation rows produced by the error-detection logic. Applying Wallace method to accumulate the four rows of product terms requires two reduction stages (see Section II-2). Within reduction stage 1, first three rows of product terms are accumulated and the fourth row is transferred to the second reduction stage without any modification. It is possible to exploit the time taken by the first reduction stage of Wallace tree to form one error compensation vector. This can be done by ORing all bits with the same weight through the first three error compensation rows with the existing fourth row of product terms. Then, instead of adding this vector as an additional row in Wallace tree, it is used to replace the fourth existing row within the second reduction stage, so this will not increase the number of reduction stages in Wallace tree. Also, to keep the critical delay of multiplier design less affected by the proposed ECM, just last seven most significant bits of the error compensation vector are accumulated in Wallace tree.

For the cases of 3- and 4-bit logic clusters, the last eight and nine of the most significant bits of the error compensation vector are included as additional row in the Wallace tree respectively. Compared to 2-bit logic cluster, this will rise the number of reduction stages in Wallace tree by one leads to increase the critical delay of the proposed multiplier design. However, the impact of the proposed ECM on the probability and mean of the errors introduced by different logic compression levels have remarkably decreased (see Section IV).

## IV. ERROR ANALYSIS

A number of simulations are carried out to examine the impact of error on the proposed approach for different degrees of logic compression when applying Wallace-tree scheme. Several error metrics have been discussed in [14] and [18] for evaluating the effectiveness and quantifying errors of approximate adders and multipliers. For any  $(N \times N)$  approximate multiplier, the error distance ( $ED$ ) is defined as the arithmetic difference between the accurate product ( $P$ ) and erroneous product ( $P'$ ), i.e.,  $ED = |P - P'|$ . The relative error distance ( $RED$ ) is the ratio of  $ED$  over the accurate output, i.e.,  $RED = \frac{ED}{P} = \frac{|P - P'|}{P}$ . The error probability ( $EP$ ) is defined as the ratio of incorrect outputs with respect to the total number of outputs. For any  $(N \times N)$  approximate multiplier, the mean  $RED$  ( $MRED$ ) is defined as [18]:

$$MRED = \frac{\sum_{i=0}^{2^{2N}-1} RED}{2^{2N}}. \quad (2)$$

The Mean Error Distance ( $MED$ ) is another useful error metric defined as the average of the  $ED$  values, i.e.,  $MED = \frac{\sum ED}{2^{2N}}$ . Also, the mean squared error ( $MSE$ ) is defined as the average of the squared  $ED$  values, i.e.,  $MSE = \frac{\sum ED^2}{2^{2N}}$ . For comparing multipliers of different sizes, the normalized  $MED$  ( $NMED$ ) is defined as [18]:

$$NMED = \frac{MED}{P_{max}} = \frac{\sum_{i=0}^{2^{2N}-1} ED}{2^{2N} P_{max}}, \quad (3)$$

where  $P_{max}$  is the maximum product that can be obtained from an  $(N \times N)$  accurate multiplier, i.e.,  $P_{max} = (2^N - 1)^2$ .

The simulations are performed in Matlab by incorporating a functional model of the SDLC approach with  $(8 \times 8)$  Wallace-tree accumulation. The response of all approximate multipliers are evaluated for all possible combinations of operands. Table I shows five error metrics using different depths of logic compression with  $(8 \times 8)$  Wallace accumulation. It can be seen that the proposed ECM improves the accuracy for all depths of logic compression. The  $MRED$  is improved more than 45% for 2- and 4-bit logic clusters and (up to 75%) for 3-bit logic clusters. Similar observation for  $NMED$  improvements (up to 76%) for 3-bit logic clusters. The increasing trend in the error rate is expected due to the increased bit-depth of logic cluster of the multiplier. This is because the growing likelihood of finding a pair of vertically aligned “ones” through two successive rows. In such cases, the corresponding OR gate will return an error (as detailed in Section III). However, such

TABLE I: ECM drastically reduces the errors across all metrics.

(8x8) Wallace Multiplier	$EP$ (%)	$MED$	$MSE$	$MRED$ (%)	$NMED$ (%)
2-bit SDLC [15]	49.11	229.38	251733.8	1.9883	0.3527
2-bit SDLC_Modified_ECM	36.75	167.18	204883.93	1.0762	0.2571
<b>Improvements(%)</b>	<b>25.17</b>	<b>27.12</b>	<b>18.61</b>	<b>45.87</b>	<b>27.11</b>
3-bit SDLC [15]	65.73	654.94	1590278.5	4.6847	1.0072
3-bit SDLC_ECM	43.19	162.52	187754.3	1.1725	0.2499
<b>Improvements(%)</b>	<b>34.29</b>	<b>75.19</b>	<b>88.19</b>	<b>74.97</b>	<b>75.19</b>
4-bit SDLC [15]	77.57	2127.78	15309286	10.5835	3.2723
4-bit SDLC_ECM	69.45	1111.45	4743255	5.5382	1.7093
<b>Improvements(%)</b>	<b>10.47</b>	<b>47.76</b>	<b>69.02</b>	<b>47.67</b>	<b>47.76</b>



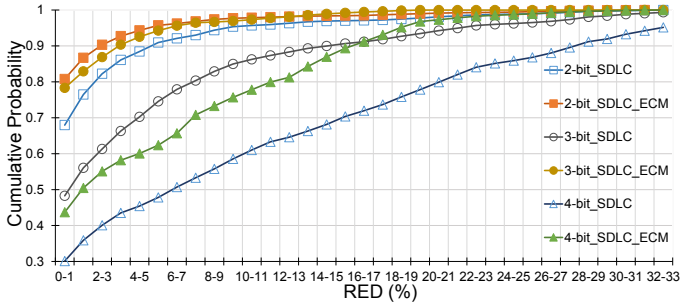


Figure 5: Cumulative probability distribution for the error induced by different logic compression levels coupled with the proposed ECM.

probability of error can be misleading, as the eventual impact of error is reflected in error distance metrics, *i.e.*, MRED and NMED [19].

The majority of these errors would not denote severe degradation of the final output because the occurrence of the higher errors is regarded as very rare. This can be seen in Figures 5 which demonstrates the cumulative probability distribution for the relative errors resulting from Wallace-tree multiplier for different sizes of logic clusters coupled with ECM. The proposed multiplier does not sacrifice the precision of the more significant bits when using SDLC approach. This can be observed in the sharp rise of the cumulative probability of errors towards 1, especially for lower depth of logic compression, such as 2- or 3-bit SDLC. Furthermore, incorporating ECM together with SDLC approach tends to produce results that are closer to the exact outputs. This is seen when the cumulative probability distributions reach to 1 faster than SDLC approach without ECM. The proposed ECM increases the probability of trivial errors; however, it lowers the probability of occurrence of higher RED. For example, in the case of 2-bit SDLC, the probability of having errors with less than 1% RED, *i.e.*, RED of 0%-1%, is increased from 0.68 to 0.81 when applying ECM, while the likelihood of RED of the range 9%-10% is decreased from 0.02 to 0.002 for the same case. Similar observations can be made in the case of 3 and 4-bit logic clusters. The impact of increased degree of compression coupled with ECM is further investigated in the application case-study in Section V.

## V. EXPERIMENTAL RESULTS AND DESIGN TRADE-OFFS

To demonstrate the proposed approach, we applied it on different ( $8 \times 8$ ) parallel multiplier designs. A SystemVerilog code was used to generate synthesizable modules for Wallace-tree accumulation structure coupled with 2-bit, 3-bit and 4-bit logic clusters. Accurate ripple adders were used in the last phase for adding the resulting two rows after Wallace accumulation phase. The generated codes were implemented and synthesised using two different off-the-shelf tools: Mentor Graphics Questa Sim was used to compile the SystemVerilog codes and run the associated test benches; and Synopsys Design Compiler was utilized for synthesising all sizes of accurate and proposed multipliers when mapping the circuits to the Faraday's 90nm technology library and evaluating for power, delay and area.

Figure 6 illustrates the impact of proposed ECM in terms of dynamic/leakage power, delay, area and PDP savings with increased degree of logic compression. As seen, there are significant improvements in all design trade-offs. This is basically because SDLC approach decreases the number of reduction stages in Wallace accumulation phase (see Section II-2). Furthermore, this reduction in hardware complexity leads to low switching capacitance and leakage reading as well as shortened critical paths. In the case of 2-bit logic clusters, slight decreasing of critical delay and power consumption comparing to 3-bit and 4-bit logic compressions. This is because the error compensation vector is utilized by replacing the fourth existing row without increasing the number of reduction stages (see Section III).

The extra cost induced by the ECM method translates in cost in area, delay and power. For example, while the area was divided by 2 with the 3-bit SDLC compared to the traditional Wallace multiplier, it is only reduced by 30% with the 3-bit SDLC-ECM (first item of Figure 6-b). This is due to increasing the number of reduction stages when including the error compensation vector to the accumulation tree as additional row for the cases 3- and 4-bit SDLC approach (see Section III). For dynamic and leakage power, the reductions obtained from applying error compensation method range from 25.3%-40.8% and 34.5%-51.5% respectively. Furthermore, the range of savings in the operating delay for the proposed multiplier is from 8.9%-17.1%. The reduction in complexity also leads to silicon area to be reduced by 29.7%-42.7%, and energy is reduced as a PDP by 32.4%-51.4%.

We evaluate the efficiency of the proposed technique on a real life image-processing application. Such an application consists of additions and multiplications using key multipliers as building blocks. Our analysis considers the Gaussian blur filter since it is widely used in graphics software, typically to

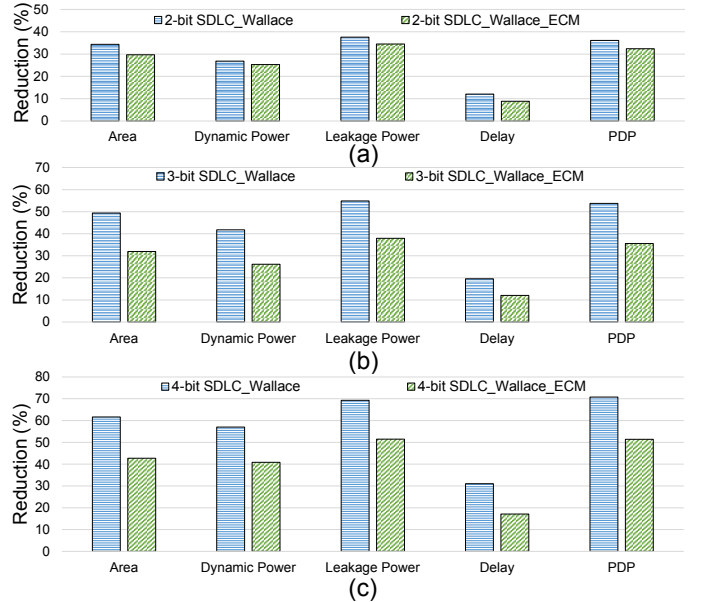


Figure 6: The impact of the proposed ECM on the ( $8 \times 8$ ) approximate Wallace multiplier with: (a) 2-bit, (b) 3-bit and (c) 4-bit logic compression levels.








Exact Wallace Multiplier	2-bit Clustering	2-bit Clustering with ECM	3-bit Clustering	3-bit Clustering with ECM	4-bit Clustering	4-bit Clustering with ECM
						
Reference Image	PSNR = 50.2	PSNR = 56.4	PSNR = 39	PSNR = 51.5	PSNR = 30	PSNR = 35.8
Energy Saving/Image	34.8 %	29.3 %	50.1 %	31.4 %	62.6 %	45.9 %

Figure 7: Evaluating the impact of proposed error compensation method on the output quality after applying Gaussian blur filtering.

reduce image noise and detail by acting as a low-pass filter. This filter involves the convolution of a ‘kernel’, described by a Gaussian function, with the pixels of the image. The values of a given pixel in the output image are calculated by multiplying each kernel value by the corresponding input image pixel values; then all the obtained values are added and the result will be the value for the current pixel that overlaps with the centre of the kernel.

To illustrate the effect of the proposed error compensation method on different logic compression levels, different versions of an 8-bit approximate Wallace-tree multiplier together with the Gaussian blur algorithm are implemented in Matlab covering 2-, 3- and 4-bit depth clustering. The Gaussian kernel is  $(3 \times 3)$  with a 1.5 standard deviation value and it uses 8-bit fixed point arithmetic and is applied to 8-bit grayscale input image size  $(512 \times 512)$  pixels. We approximate Gaussian blur by replacing the standard multiplication in the Gaussian filter with the aforementioned approximate  $(8 \times 8)$  multipliers. The peak signal-to-noise ratio (PSNR) is a fidelity metric used to measure the quality of the output images, expressed as:

$$PSNR = 10 \log_{10} \left( \frac{255^2}{MSE} \right), \quad (4)$$

where  $MSE$  is the mean squared error measured with respect to the reference pixel. Figure 7 demonstrates the impact of the proposed error compensation method on the image quality after applying the Gaussian blur filter. As seen, the PSNR in the case of 2-, 3- and 4-bit depth clustering are 50.2 dB, 39 dB, 30 dB respectively. By employing the proposed error compensation method, the PSNR values are increased to 56.4 dB, 51.5 dB, 35.8 dB respectively. This is because the error compensation vector capable to decrease the mean and probability of the majority of the errors introduced by the lossy compression. Thus, the proposed approach can provide a significant dynamic energy saving up to 68.3% with acceptable quality of output image, especially when utilizing smaller bit depth clusters with the proposed error compensation method.

## VI. CONCLUSIONS

In this paper, a novel approximate Wallace multiplier design is proposed using significance-driven logic compression (SDLC) approach. This design approach utilizes an algorithmic and configurable lossy compression based on bit significance to form a reduced number of partial product rows. This is then accumulated using Wallace-tree scheme. Different  $(8 \times 8)$  multiplier designs in SystemVerilog and their post synthesis results demonstrate the following key advantages. Firstly, The Wallace-tree modification with reduced length

of carry propagating adder improve the power efficiency for 2-bit logic cluster in SDLC. Secondly, the proposed error compensation method significantly reduce the impact of errors resulted from variable sizes of logic clusters at low power and area overheads. These trade-offs are further substantiated by a case study of convolution filter used in image processing.

## ACKNOWLEDGMENT

The authors would like to thank MOHE (Jordan), BAU (Jordan) and EPSRC PRiME project EP/K034448/1(UK) for their funding and support.

## REFERENCES

- [1] S. Mittal, “A survey of techniques for approximate computing,” *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, p. 62, 2016.
- [2] L. Sekanina, “Introduction to approximate computing: Embedded tutorial,” in *IEEE DDECS*, pp. 1–6, 2016.
- [3] J. Han and M. Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” in *ETS*, pp. 1–6, 2013.
- [4] M. D. Ercegovac, “On approximate arithmetic,” in *2013 Asilomar Conference on Signals, Systems and Computers*, pp. 126–130, 2013.
- [5] S. H. Nawab *et al.*, “Approximate signal processing,” *The Journal of VLSI Signal Processing*, vol. 15, no. 1, pp. 177–200, 1997.
- [6] H. Jiang *et al.*, “A comparative evaluation of approximate multipliers,” in *NANOARCH, IEEE/ACM*, pp. 191–196, 2016.
- [7] Y. Liu *et al.*, “Computation error analysis in digital signal processing systems with overscaled supply voltage,” *IEEE on VLSI systems*, vol. 18, no. 4, pp. 517–526, 2010.
- [8] K. Shi *et al.*, “Datapath synthesis for overclocking: Online arithmetic for latency-accuracy trade-offs,” in *DAC*, pp. 1–6, ACM, 2014.
- [9] T. A. Drane *et al.*, “On the systematic creation of faithfully rounded truncated multipliers and arrays,” *IEEE Transactions on Computers*, vol. 63, no. 10, pp. 2513–2525, 2014.
- [10] P. Kulkarni *et al.*, “Trading accuracy for power with an underdesigned multiplier architecture,” in *2011 24th International Conference on VLSI Design*, pp. 346–351, 2011.
- [11] C. H. Lin and I. C. Lin, “High accuracy approximate multiplier with error correction,” in *ICCD*, pp. 33–38, 2013.
- [12] S. Venkataramani *et al.*, “Salsa: Systematic logic synthesis of approximate circuits,” in *DAC*, pp. 796–801, 2012.
- [13] R. Venkatesan *et al.*, “Macaco: Modeling and analysis of circuits for approximate computing,” in *Conf. on CAD*, pp. 667–673, IEEE, 2011.
- [14] V. Mrazek *et al.*, “Evoapproxsb: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods,” in *DATE*, pp. 258–261, 2017.
- [15] I. Qiqieh *et al.*, “Energy-efficient approximate multiplier design using bit significance-driven logic compression,” in *DATE*, pp. 7–12, 2017.
- [16] C. S. Wallace, “A suggestion for a fast multiplier,” *IEEE Transactions on electronic Computers*, no. 1, pp. 14–17, 1964.
- [17] W. J. Townsend *et al.*, “A comparison of dadda and wallace multiplier delays,” in *Optical Science and Technology, SPIE*, pp. 552–560, International Society for Optics and Photonics, 2003.
- [18] J. Liang *et al.*, “New metrics for the reliability of approximate and probabilistic adders,” *IEEE Transactions on Computers*, vol. 62, pp. 1760–1771, Sept 2013.
- [19] I. S. Chong *et al.*, “New quality metrics for multimedia compression using faulty hardware,” in *International Workshop on Video Processing and Quality Metrics for Consumer Electronics*, pp. 267–272, 2006.