Energy-Efficient Approximate Multiplier Design using Bit Significance-Driven Logic Compression

Issa Qiqieh, Rishad Shafik, Ghaith Tarawneh, Danil Sokolov, and Alex Yakovlev
School of Electrical and Electronic Engineering, Newcastle University, Newcastle upon Tyne, NE1 7RU, UK
E-mails: {i.qiqieh1, rishad.shafik, ghaith.tarawneh, danil.sokolov, alex.yakovlev}@newcastle.ac.uk

Abstract—**Approximate arithmetic has recently emerged as a promising paradigm for many imprecision-tolerant applications. It can offer substantial reductions in circuit complexity, delay and energy consumption by relaxing accuracy requirements. In this paper, we propose a novel energy-efficient approximate multiplier design using a significance-driven logic compression (SDLC) approach. Fundamental to this approach is an algorithmic and configurable lossy compression of the partial product rows based on their progressive bit significance. This is followed by the commutative remapping of the resulting product terms to reduce the number of product rows. As such, the complexity of the multiplier in terms of logic cell counts and lengths of critical paths is drastically reduced. A number of multipliers with different bit-widths (4-bit to 128-bit) are designed in SystemVerilog and synthesized using Synopsys Design Compiler. Post-synthesis experiments showed that up to an order of magnitude energy savings, and reductions of 65% in critical delay and almost 45% in silicon area can be achieved for a 128-bit multiplier compared to an accurate equivalent. These gains are achieved with low accuracy losses estimated at less than 0.00071 mean relative error. Additionally, we demonstrate the energy-accuracy trade-offs for different degrees of compression, achieved through configurable logic clustering. In evaluating the effectiveness of our approach, a case study image processing application showed up to 68.3% energy reduction with negligible losses in image quality expressed as peak signal-to-noise ratio (PSNR).**

I. Introduction

There is a persistent demand for higher computational performance at low energy cost for emerging applications. It is unlikely that improvements from manufacturing processes alone, such as technology nodes or many-core system-on-chip, will be able to cope with this challenge. Thus there is a genuine need to develop disruptive design approaches to achieve transformational energy reductions. Approximate computing systems design is a promising approach to this end [1].

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. Research has shown that the majority of modern applications such as digital signal processing, computer vision, robotics, multi-media and data analytics have some level of tolerance to such imprecision [2]. This can be leveraged as an opportunity for energy-efficient systems design for current and future generations of application-specific systems.

Multipliers are crucial arithmetic units in many of these 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 to compute outcomes. These factors have prompted close attention in approximate multiplier design research, since improvements made in the power/speed of a multiplier are expected to substantially impact on overall system power/performance trade-offs [3].

TABLE I: Summary of approximate multiplier design approaches.

Approach	Methodology	Features and Limitations
[4] [5]	Aggressive voltage scaling: lowering the supply voltage below its nominal value.	Unexpected time-induced errors, which normally impact the most significant bits.
[6] [7]	Truncation: eliminating partial products from the least significant columns.	As more columns are eliminated, the resulting errors are maximised.
[8] [9]	Modular re-design: large efficient multipliers using inaccurate small multiplier blocks.	Scalability is not simple and this method may not significantly reduce the critical path.
[10]	S/W-based perforation: approximation of the generation of the partial products.	Decreasing the depth of the accumulation tree by utilizing a tool, and also real-time needs.
[11] [12]	Automated re-design: systematically reducing the complexity of circuits.	Greedy approach depending on circuit activity profile and output significance.
[13] [14]	Manual re-design: manual alteration of the functional behaviours of the structure.	Disparate ideas of redesigning the multiplier extend from architecture to transistor level.

Table I summarizes the key features and limitations of research efforts to date in the domain of approximate multipliers. These can be largely categorized as modifications of either timing or functional behaviors. Firstly, timing behavior can be modified using aggressive supply voltage scaling techniques [4], [5]. Operating below nominal voltage allows for reductions in energy consumption at the cost of time-induced errors. These errors cannot be rigorously bounded, and so extra error-compensation circuits need to be incorporated. Secondly, 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 [6], [7]. As more columns are eliminated, further energy reduction is achieved; however, errors also increase. Modular re-design with low-complexity combinational logic is another effective technique [8], [9]. This allows for building larger energy-efficient multipliers using small approximate ones; however, the hierarchical organization of small approximate blocks will eventually propagate errors which increase with the multiplier size. A software-based perforation technique has been proposed [10] by obtaining the optimized set of partial product terms based on power-area-accuracy trade-offs. Automated design approaches [11], [12] present design flows for generating approximate circuits using circuit activity profiles and quality bounds, and an evolutionary design process based on Cartesian Genetic Programming (CGP) has been utilized to implement approximate multipliers [15]. A number of power- and area-efficient multiplier redesign approaches have been proposed by changing the functional behavior. These changes extend from the architecture to transistor-level [13], [14]. The key principle of the above studies is to achieve reduced logic complexity, which is also the main aim of our work.

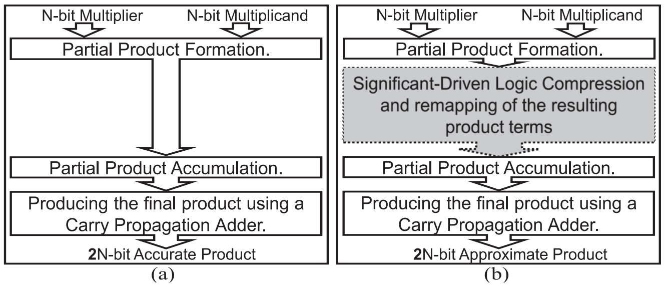
A typical $(N \times N)$ accurate multiplier generates N^2 product terms, which are then accumulated as a final product of size $2N$. The accuracy of this product depends largely on the significance of bits; preserving higher-significance bits is likely to generate an outcome closes to the exact product than that of lower-significance bits. This can be exploited to progressively compress higher order combinatorial terms systematically and to achieve substantial energy savings at low loss of accuracy. In our work, we leverage this opportunity to make the following key contributions:

1) We propose a novel energy-efficient approximate multiplier design approach using bit significance-driven logic compression (SDLC).
2) At the core of our approach is a configurable logic clustering of product terms appropriately chosen for a given energy-accuracy trade-off, followed by remapping using their commutative properties to reduce the resulting number of product terms.
3) We demonstrate the comparative gains (with up to an order of magnitude energy reduction) through the design and synthesis of multipliers of different sizes (from 4 to 128 bits). Furthermore, we implement the multiplier in a real case-study image processing application to highlight its key advantages.

To the best of our knowledge, this is the first demonstration of a systematic logic compression-based approximate multiplier design approach using a real application. The rest of the paper is organized as follows. Section II introduces the proposed approximate multiplier design. Section III provides the error analysis associated with different bit-widths of the proposed multiplier. The experimental results and design trade-offs are described in Section IV. Finally, Section V concludes the paper.

II. Proposed Approximate Multiplier Design

Our proposed approach consists of two major steps. In the first, lossy compression is carried out through logic clustering. The resulting compressed terms are then remapped using their commutative properties. These steps together with the variable compression method, are described below.



N-bit Multiplier
N-bit Multiplicand
Partial Product Formation.
Partial Product Accumulation.
Producing the final product using a Carry Propagation Adder.
2N-bit Accurate Product
(a)
N-bit Multiplier
N-bit Multiplicand
Partial Product Formation.
Significant-Driven Logic Compression and remapping of the resulting product terms
Partial Product Accumulation.
Producing the final product using a Carry Propagation Adder.
2N-bit Approximate Product
(b)

Figure 1: Process chart showing the difference between the major stages in: (a) conventional multiplication, and (b) the proposed approach to multiplication.

1) Logic Compression: Parallel multiplication design is generally divided into three consecutive stages: partial product formation, accumulation, and carry propagation adder. In an $(N \times N)$ multiplier, N^2 AND gates are utilized in parallel to generate the partial product bit-matrix. This matrix is then column-wise accumulated to generate the final product by using carry propagation adders.

The proposed approach begins by generating all partial products using the same number of AND gates, similar to conventional multiplication. Before proceeding to the accumulation stage, the number of bits in the partial product matrix is reduced by performing lossy logic compression. The aim is to reduce the number of rows in the partial product matrix, thereby achieving low-complexity hardware before proceeding to accumulation. Figure 1 shows the difference between the design stages in accurate and the proposed multiplication. The shaded box highlights the contribution in this paper. To achieve lossy compression, we follow three key principles as follows.

a. Clustering a group of rows: The proposed multiplier organizes the partial product terms using different sizes of significant-driven logic clusters. Each logic cluster targets a group of columns containing two bits starting from the least significant bits in successive partial products. In general, each $2 \times L$ logic cluster is responsible for two operations: i) generating $2L$ partial product bits within two contiguous rows, *i.e.*, L pairs of vertically aligned bits, by utilizing $2L$ AND gates. Then, ii) minimizing these $2L$ bits by half using L OR gates. Figure 2 illustrates the utilization of four sizes of logic clusters in 8-bit parallel multiplier. The first 2×7 logic cluster forms 14 partial products by utilizing 14 AND logic gates and extracts 7-bit value by using an array of 7 OR logic gates. The second 2×6 logic cluster minimizes 12 partial products into 6 bits. In a similar way the third and fourth logic clusters use 2×5 and 2×4 to minimize 10 and 8 partial products into 5 and 4 bits respectively. By doing so, each logic cluster compresses a group of vertically aligned bits within two successive partial products based on their progressive bit significance.

b. Generation of a reduced set of product terms: Using an array of OR gates in each logic cluster compresses the partial product terms by half. A reduced set of preprocessed partial product matrix is thus ready to be accumulated by applying any convenient scheme of multipli-

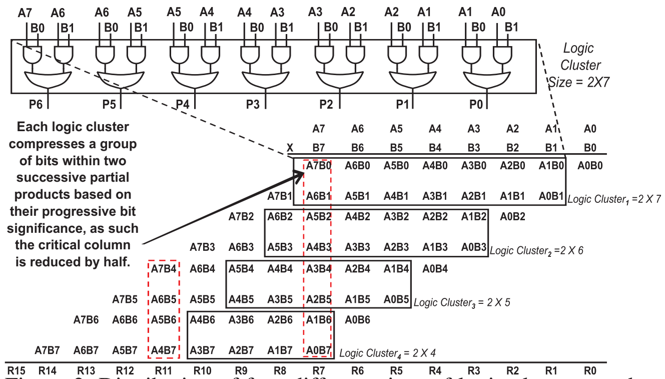

A7 A6 A6 A5 A5 A4 A4 A3 A3 A2 A2 A1 A1 A0
B0 B1 B0 B1 B0 B1 B0 B1 B0 B1 B0 B1 B0 B1
P6 P5 P4 P3 P2 P1 P0
Logic Cluster Size = 2X7
Each logic cluster compresses a group of bits within two successive partial products based on their progressive bit significance, as such the critical column is reduced by half.
X A7 A6 A5 A4 A3 A2 A1 A0
B7 B6 B5 B4 B3 B2 B1 B0
A7B0 A6B0 A5B0 A4B0 A3B0 A2B0 A1B0 A0B0
A7B1 A6B1 A5B1 A4B1 A3B1 A2B1 A1B1 A0B1
Logic Cluster1 = 2 X 7
A7B2 A6B2 A5B2 A4B2 A3B2 A2B2 A1B2 A0B2
A7B3 A6B3 A5B3 A4B3 A3B3 A2B3 A1B3 A0B3
Logic Cluster2 = 2 X 6
A7B4 A6B4 A5B4 A4B4 A3B4 A2B4 A1B4 A0B4
A7B5 A6B5 A5B5 A4B5 A3B5 A2B5 A1B5 A0B5
Logic Cluster3 = 2 X 5
A7B6 A6B6 A5B6 A4B6 A3B6 A2B6 A1B6 A0B6
A7B7 A6B7 A5B7 A4B7 A3B7 A2B7 A1B7 A0B7
Logic Cluster4 = 2 X 4
R15 R14 R13 R12 R11 R10 R9 R8 R7 R6 R5 R4 R3 R2 R1 R0

Figure 2: Distribution of four different sizes of logic clusters used to compress partial products based on their progressive bit-significance in (8×8) parallel multiplier architecture.

cation, such as carry-save array, Wallace and Dadda tree. In theory, a two-input OR gate is sufficient to sum up 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', the difference value is '1' as the adder returns '10' and OR outputs '1'.

c. Significance-driven progressive cluster sizing: Since the main goal is to design a power-efficient multiplier with negligible loss of accuracy, the size of the logic clusters is decreased when going down in the partial product matrix. The more significant bits are treated with progressively higher precision, while bits with lower significance are compressed using the SDLC approach. This permits the most significant product terms to be accumulated on a carry-propagation basis as in the conventional multiplier. Thus, the accuracy of the significant bits of the final product is less affected.

Despite using the same number of AND gates as the accurate multiplier, this approach will deterministically reduce the hardware complexity of partial product accumulation, *e.g.*, the count of the compressor cells needed in column compression multiplication for Wallace and Dadda cases, and also the number of half and full adders in the carry-save array will be decreased since the number of bits in the accumulation tree is minimized.

2) Commutative Remapping: The logic compression step (Section II-1) reduces the number of partial product terms. This reduction can be leveraged to reduce 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. Due to the reduced number of rows, the critical path delay is drastically reduced (see Section IV). Figure 3 demonstrates how the size of the partial product bit-matrix in the case of an (8×8) multiplier is reduced using the SDLC approach. The lined boxes refer to a group of bits targeted by different sizes of logic clusters in which the height of the critical column is reduced by half.

The proposed approach is scalable for any $(N \times N)$ multiplier, as shown in Algorithm 1. This algorithm generates a reduced and ordered partial product bit-matrix, which can

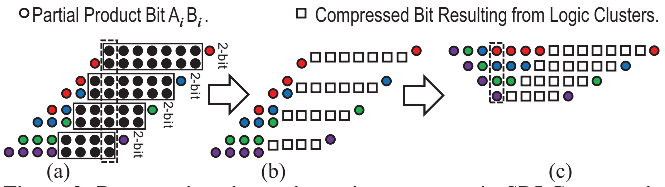

○ Partial Product Bit A_iB_i.
□ Compressed Bit Resulting from Logic Clusters.
2-bit
2-bit
2-bit
2-bit
(a)
(b)
(c)

Figure 3: Dot notation shows the major two steps in SDLC approach in the case of (8×8) multiplier: (a) clustering a group of rows in the partial product bit-matrix after bitwise multiplication; (b) generating a reduced set of product terms after applying logic compression; (c) ordered matrix after applying commutative remapping of the bit sequence resulting from the SDLC approach. The dotted rectangles indicate the height of the critical column which is reduced by half compared to the accurate accumulation tree.

then be treated as an accumulation tree by any scheme of multiplication. Line (9) indicates how partial product bits are compressed using logic clusters. The main loop (lines 6 to 17) is responsible for remapping product terms in an ordered bit-matrix, as demonstrated in Figure 3.

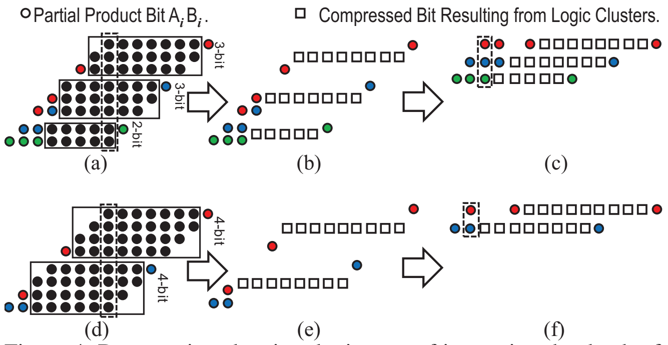
3) Variable Logic Cluster Approach: The proposed approach is capable of achieving higher degrees of compression by increasing logic cluster depth. Figure 4 demonstrates the impact of increasing depth to 3 and 4 bits in the case of (8×8), showing the key steps in logic compression and commutative remapping. As can be seen, with increased depth we can achieve further reduction in the partial product terms, leading to fewer rows for final accumulation.

Algorithm 1 Generating a reduced partial product matrix M using the Significant-Driven Logic Clusters (SDLC) approach for $(N \times N)$ multiplier, $\{\forall\ N \in \{2\sigma : \sigma \in \mathbb{N}^*\}\}$

```
1:  procedure SDLC($M, A, B$)
2:      Output:  $M[1, 2, ..., \frac{N}{2}][1, 2, ..., 2N-1]$          ▷ Reduced Matrix
3:      Inputs:  $A[1, 2, ..., N]$                                    ▷ Multiplicand bits
4:               $B[1, 2, ..., N]$                                    ▷ Multiplier bits
5:      Initialize:$\rho \leftarrow 1$                                ▷ Bit Position
6:      for $i \leftarrow 1\ to\ \frac{N}{2}$ do                      ▷ Forming rows of $M$
7:          $M[i][\rho] \leftarrow A(0) \wedge B(2i-2)$               ▷ First bit in each row of M
8:          for $j \leftarrow 1\ to\ N-i$ do                          ▷ Forming outputs of each logic cluster
9:              $M[i][j+\rho] \leftarrow (A(j) \wedge B(2i-2)) \ \vee\ (A(j-1) \wedge B(2i-1))$
10:         end for
11:         Initialize:$\delta \leftarrow 1$                         ▷ Bit Position
12:         for $k \leftarrow 2i-1\ to\ N-1$ do                       ▷ Forming unaffected MSBs
13:             $M[i][\rho + (N-i) + \delta] \leftarrow A(N-i) \wedge B(k)$
14:             $\delta \leftarrow \delta + 1$
15:         end for
16:         $\rho \leftarrow \rho + 2$                                ▷ Shift left by 2 (next row)
17:     end for
18:     return $M$                         ▷ $M$ is then treated as a reduced accumulation tree
19: end procedure
```

III. Error Analysis

A number of simulations are carried out to examine the impact of error on the proposed approach for different sizes of multiplier. Several error metrics have been discussed [16] and [17] 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 rate (ER) is


Partial Product Bit $A_i B_i$.
Compressed Bit Resulting from Logic Clusters.
3-bit
3-bit
2-bit
4-bit
4-bit
(a)
(b)
(c)
(d)
(e)
(f)
Figure 4: Dot notation showing the impact of increasing the depth of the logic clusters in the case of (8×8) multiplier: (a) clustering a group of bits within three successive rows in the partial product bit-matrix after bitwise multiplication; (b) generating a reduced set of product terms after targeting the depth of 3-row logic compression; (c) ordered matrix after applying commutative remapping of the bit sequence resulting from the SDLC approach; (d), (e) and (f) the same process when applying 4-bit logic clusters. The dotted rectangles indicate the heights of the critical columns which are further reduced compared to the accurate accumulation tree.

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 [17]:

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

The Mean Error Distance (MED) is another useful error metric defined as the average of the ED values , *i.e.*, $MED = \frac{\sum ED}{2^{2N}}$. For comparing multipliers of different sizes, the normalized MED (NMED) is defined as [17]:

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

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

Exhaustive simulations are performed in Matlab by implementing a functional model of the SDLC approach. The response of all approximate multipliers are evaluated for all possible combinations of operands. Table II shows four error metrics using varying sizes of the proposed multiplier. It can be seen that MRED and NMED fall drastically as the size of the multiplier is increased from 4 to 16-bit. The increasing trend in the error rate is expected due to the increased bit-width of the multiplier. This is because the error occurrence increases as well due to 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 II-1.

However, such error rates can be misleading, as the eventual impact of error is reflected in error distance metrics such as MRED and NMED [18]. Also, the readings of MAX(RED)

TABLE II: Error metrics for varying sizes of proposed multiplier.

Bit-Width	MRED	NMED	ER (%)	MAX(RED) (%)
4-bit	2.77313	0.010556	19.53	31.1111
6-bit	2.65879	0.006393	34.96	32.8042
8-bit	1.98826	0.003527	49.11	33.2026
12-bit	0.00824	0.000952	70.68	33.3308
16-bit	0.00071	0.000084	78.72	33.3325

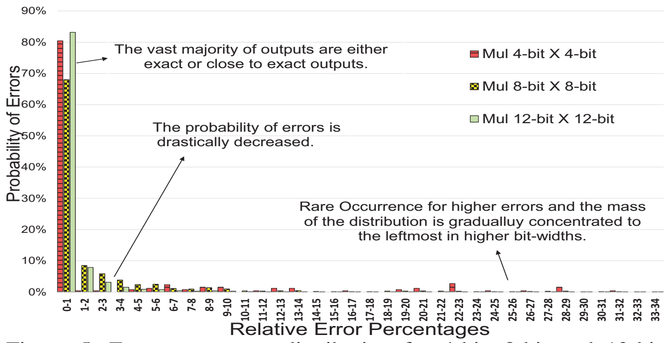


Figure 5: Error percentage distribution for 4-bit, 8-bit and 12-bit proposed multiplier after applying 2-bit depth compression.

would not denote severe degradation of the final output because the occurrence of these errors is regarded as very rare. This can be seen in Figure 5, which demonstrates the probability distribution for all relative errors resulting from three different sizes of multipliers using the SDLC approach. The probability distribution shows that the proposed approach tends to produce exact or close to exact results. This is seen in the sharp decline of the probability of errors with higher REDs, *e.g.*, the MAX(RED) listed in Table II. Furthermore, as the bit-width of the multiplier is increased, the mass of the distribution is gradually concentrated at a lower error distance. This is because the proposed approach does not sacrifice the precision of the more significant bits when using significance-driven logic compression.

Table III depicts the error trade-off with increased degree of compression achieved through higher depths of logic clusters in (8×8) multiplier. As expected, increased depth leads to higher error rates (up to 78%) when clustering with 4-row logic compression. However, results for the MRED metric are only marginally higher when compared with logic compression with 2- or 3-bit logic clusters. Similar observations can be made in the case of the NMED metric. The impact of increased degree of compression is further investigated in the application case-study in Section IV.

IV. Experimental Results and Design Trade-offs

To demonstrate the proposed approach, we applied it on eight different sizes of widely known multipliers ranging from 4-bit to 128-bit. For the purpose of fair comparison, accurate ripple adders were used in both accurate and approximate multipliers to accumulate the partial product rows within the accumulation stage (see Figure 1). A generic SystemVerilog code was used to generate synthesizable modules for all accurate and approximate versions. These modules have been parametrized and configured differently during instantiation according to the bit-width of multiplier. The generated codes were implemented and synthesised using two different off-the-shelf tools: Mentor Graphics Questa Sim was used to compile

TABLE III: MRED, NMED, error rate and maximum RED for different depths of logic compression in (8×8) multiplier.

Cluster-Depth	MRED	NMED	ER (%)	MAX(RED) (%)
2-bit	1.9883	0.0035	49.11	33.2
3-bit	4.6847	0.0101	65.73	42.69
4-bit	10.5836	0.0327	77.57	46.48

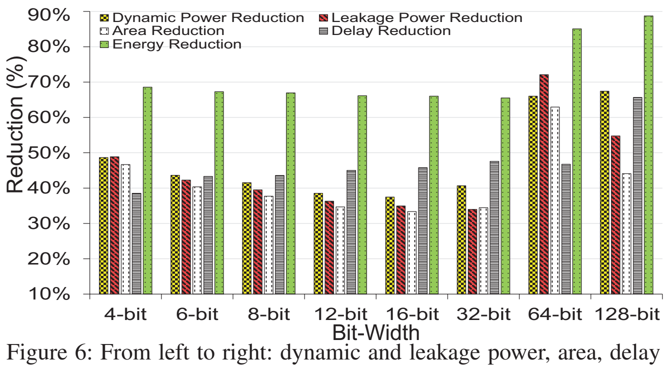


Figure 6: From left to right: dynamic and leakage power, area, delay and energy reductions for different bit-widths of proposed multiplier.

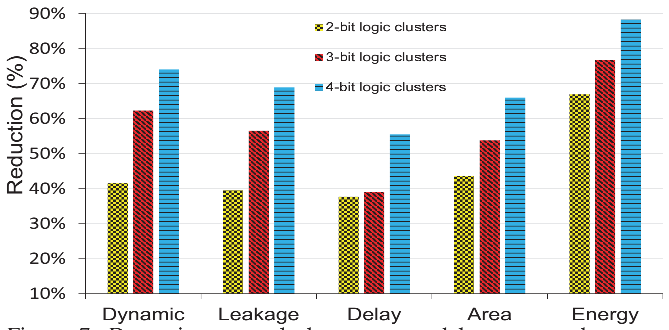


Figure 7: Dynamic power, leakage power, delay, area and energy savings for different degrees of logic compression of 8-bit multiplier.

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 presents a comparison of dynamic/leakage power, area, delay and energy reductions for all eight sizes of proposed multipliers when compared with a conventional accurate multiplier (Figure 1(a)). As seen, there are significant improvements in all design trade-offs. This is basically because SDLC approach reduces the complexity of multiplier implementation by reducing the number of rows in the accumulation tree. Furthermore, this reduction in hardware complexity leads to low switching capacitance and leakage readings as well as shortened critical paths.

The experiments show noteworthy reductions in terms of power consumption, runt-time and also silicon area used. For dynamic and leakage power, the reductions obtained from applying the SDLC approach range from 37.5%-67.4% and 34%-72.1% respectively when the bit-width ranges from a 4-bit to 128-bit multiplier. Furthermore, the range of savings in the operating delay for the same sizes of the proposed multiplier is from 38.5%-65.6%. The reduction in complexity also leads to silicon area to be reduced by 33.4%-62.9%, and energy consumed is substantially reduced by 65.5%-88.74%. The non-linear trend of the bars in some cases is attributed to the inconsistency of the ratio of the array of additions in the accumulation tree between the approximate and the accurate multiplier.

Figure 7 illustrates the dynamic/leakage power, delay, area and energy savings with increased degree of logic compression. Higher depth of clustering achieves considerable savings in all design trade-offs since by increasing the depth of logic clusters, the hardware complexity associated with lower numbers of product rows is also decreased.

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 [19] since it is widely used in graphics software, typically to 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 variable logic clusters in the proposed approach, different versions of an 8-bit approximate multiplier together with the Gaussian blur algorithm are implemented in Matlab covering 2-, 3- and 4-bit depth clustering. The Gaussian kernel is (3×3) 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 (200×200) pixels. We approximate Gaussian blur by replacing the standard multiplication in the Gaussian filter with the aforementioned approximate (8×8) multipliers. The peak signal-to-noise ratio (PSNR) is a fidelity metric used to measure the quality of the output images. PSNR is expressed as:

$$PSNR = 10 \log_{10} \left(\frac{255^2}{MSE} \right) \quad , \qquad (3)$$

where MSE is the mean squared-error measured with respect to the reference pixel. Figure 8 demonstrates the impact of different bit-depth clustering on the image quality after applying the Gaussian blur filter. The standard (8×8) multiplier and three different levels of approximation for the proposed (8×8) multiplier are used. In fact, the utility of the proposed approach yields fruitful results. The PSNR for the case of 2-, 3- and 4-bit depth clustering are 50.2 dB, 39 dB, 30 dB respectively. The values of PSNR are computed compared to the image resulting after applying Gaussian blur filtering with the case of exact multiplication. 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

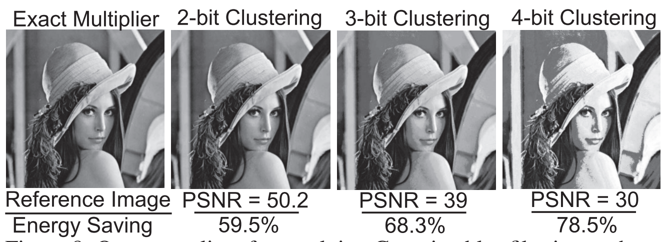


Figure 8: Output quality after applying Gaussian blur filtering to three different versions of the proposed (8×8) multiplier.

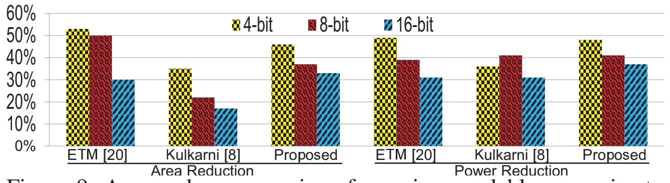


Figure 9: Area and power savings for various scalable approximate multipliers.

such as 2- and 3-bit. Figure 9 shows comparative power and area advantages of our approach for different bit-widths. The comparisons are carried out with the following two existing approaches: Kulkarni [8] and ETM [20], chosen for their direct relevance to our work. In the Kulkarni approach, a large multiplier is produced using small approximate multipliers as building blocks. The design approach in the ETM follows truncation principles by dividing the multiplier into accurate and approximate parts. In the approximate part, probabilistic bit manipulation is used to generate the product terms. Our approach produces better results as the bit-width of the multiplier is increased. This is seen with the 16-bit multiplier, where our approach outperforms both approaches in terms of power and area. This is because the number of product rows is halved (with 2-bit clustering) and commutative remapping is used to reduce the parallel accumulation complexity. The corresponding error comparisons of these approaches are shown in Table IV, demonstrating comparative errors (in terms of MRED, NMED and also ER) using the proposed (8×8) multiplier (with 2-bit clustering). As expected, our approach outperforms both approaches in terms of MRED, NMED due to its bit significance-driven logic compression. As the proposed approach progressively preserves the high-order bits, it is expected to exhibit significantly lower errors for multipliers with higher bit-widths.

TABLE IV: Comparative errors in terms of MRED, NMED and also ER for various scalable (8×8) approximate multipliers.

	ETM [20]	Kulkarni [8]	Proposed
MRED (%)	25.2	3.25	1.99
NMED(%)	2.8	1.39	0.335
ER(%)	98.8	46.73	49.11

V. Conclusions

In this paper, a novel approximate multiplier design is proposed using significance-driven logic compression (SDLC). This design approach utilizes an algorithmic and configurable lossy compression based on bit significance to form a reduced set of partial product terms. This is then reorganized and accumulated using various schemes of parallel multiplication. On a statistical basis, the results of NMED and MRED metrics show how the impact of error is alleviated when the size of the multiplier is increased. Additionally, the error distributions show high right-skewness for error probabilities, indicating that the proposed multiplier gives close to exact products for most inputs. The results obtained after synthesis have shown a substantial decrease in run-time, power consumption and even in silicon area. We demonstrate energy-accuracy trade-offs for different levels of approximations achieved through configurable logic clustering. To illustrate the effect of variable logic clusters, case study of an image-processing application shows that the proposed approach can provide significant energy and area savings with negligible loss in output quality, especially when utilizing smaller bit depth clusters. We believe that the proposed approach can be used with already existing low-power compute units to extract manifold benefits with a minimal loss in output quality.

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] J. Han and M. Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” in *2013 18th IEEE European Test Symposium (ETS)*, pp. 1–6, May 2013.

[2] L. Sekanina, “Introduction to approximate computing: Embedded tutorial,” in *2016 IEEE 19th DDECS*, pp. 1–6, April 2016.

[3] H. Jiang, C. Liu, N. Maheshwari, F. Lombardi, and J. Han, “A comparative evaluation of approximate multipliers,” in *2016 IEEE/ACM International Symposium on NANOARCH*, pp. 191–196, July 2016.

[4] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, “Design of voltage-scalable meta-functions for approximate computing,” in *2011 DATE*, pp. 1–6, March 2011.

[5] Y. Liu, T. Zhang, and K. K. Parhi, “Computation error analysis in digital signal processing systems with overscaled supply voltage,” *IEEE Transactions on VLSI Systems*, vol. 18, pp. 517–526, April 2010.

[6] N. Petra, D. D. Caro, V. Garofalo, E. Napoli, and A. G. M. Strollo, “Truncated binary multipliers with variable correction and minimum mean square error,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, pp. 1312–1325, June 2010.

[7] J. E. Stine and O. M. Duverne, “Variations on truncated multiplication,” in *DSD, 2003*, pp. 112–119, Sept 2003.

[8] P. Kulkarni, P. Gupta, and M. Ercegovac, “Trading accuracy for power with an underdesigned multiplier architecture,” in *2011 24th Internatioal Conference on VLSI Design*, pp. 346–351, Jan 2011.

[9] C. H. Lin and I. C. Lin, “High accuracy approximate multiplier with error correction,” in *2013 IEEE 31st ICCD*, pp. 33–38, Oct 2013.

[10] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris, and K. Pekmestzi, “Design-efficient approximate multiplication circuits through partial product perforation,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, pp. 3105–3117, Oct 2016.

[11] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, “Salsa: Systematic logic synthesis of approximate circuits,” in *DAC, 2012 49th ACM/EDAC/IEEE*, pp. 796–801, June 2012.

[12] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, “Macaco: Modeling and analysis of circuits for approximate computing,” in *2011 IEEE/ACM ICCAD*, pp. 667–673, Nov 2011.

[13] K. Bhardwaj, P. S. Mane, and J. Henkel, “Power- and area-efficient approximate wallace tree multiplier for error-resilient systems,” in *Fifteenth ISQED*, pp. 263–269, March 2014.

[14] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, “Low-power digital signal processing using approximate adders,” *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 32, pp. 124–137, 2013.

[15] Z. Vasicek and L. Sekanina, “Evolutionary approach to approximate digital circuits design,” *IEEE Transactions on Evolutionary Computation*, vol. 19, pp. 432–444, June 2015.

[16] C. Liu, J. Han, and F. Lombardi, “A low-power, high-performance approximate multiplier with configurable partial error recovery,” in *2014 DATE*, pp. 1–4, March 2014.

[17] J. Liang, J. Han, and F. Lombardi, “New metrics for the reliability of approximate and probabilistic adders,” *IEEE Transactions on Computers*, vol. 62, pp. 1760–1771, Sept 2013.

[18] I. S. Chong, H. yeon Cheong, and A. Ortega, “New quality metric for multimedia compression using faulty hardware,” in *In Intl Workshop on Video Processing and Quality Metrics for Consumer Electronics*, 2006.

[19] C. Solomon and T. Breckon, *Fundamentals of digital image processing : a practical approach with examples in Matlab*. Wiley-Blackwell, 2011.

[20] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, “Low-power high-speed multiplier for error-tolerant application,” in *EDSSC*, pp. 1–4, 2010.