Significance-Driven Logic Compression for Energy-Efficient Multiplier Design

Issa Qiqieh[†], *Student Member, IEEE,* Rishad Shafik[†], *Member, IEEE,* Ghaith Tarawneh[†], *Member, IEEE,* Danil Sokolov[†], *Member, IEEE,* Shidhartha Das[‡], *Member, IEEE,* and Alex Yakovlev[†], *Fellow, IEEE*

***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 an 128-bit multiplier, compared to an accurate equivalent. These gains are achieved with low accuracy losses estimated at less than 0.0028 mean relative error. Additionally, we demonstrate the performance-energy-quality (PEQ) trade-offs for different degrees of compression, achieved through configurable logic clustering. While evaluating the effectiveness of the proposed approach three case studies were set up. First, a Gaussian blur filter was designed, which demonstrated up to 80% energy reduction with a meagre loss of image quality. Second, we evaluate our approach in machine learning application using perceptron classifier, showed up to 74% energy reduction with negligible error rate. Third, the proposed multiplier designs were used in a power-constrained image processing application. We showed that SDLC can achieve 60x improvement in computation capability, with potential to be employed in ubiquitous systems.**

***Index Terms*—Parallel multipliers, approximate arithmetic, adaptive computing, power-constrained computing.**

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]–[3].

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 decreased 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 [4]. This can be leveraged as an opportunity for energy-efficient systems 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, minimizing cost and improving reliability in many of these applications. It has been largely present in computing systems using fixed-point and floating point operations [5], [6]. Multipliers are crucial arithmetic units in modern 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 [7].

Over the years, a number of approximate computing approaches have been proposed. These approaches aim to reduce the complexity of the circuits and systems in terms of their computation latency and energy consumption [8]. Approximations can be introduced at all design levels, starting from the circuit [9] via the logic [10] and the architecture [11], [12] to programming language [13] and algorithms [14], [15]. The common design techniques in approximate circuit designs include functional, timing (voltage over-scaling (VOS) and over-clocking) approximations and systematic design methodologies with summary of the related work, are described further in the next section.

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 close to the exact product than that of

[†]I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov and A. Yakovlev are with the School of Electrical and Electronic Engineering, Newcastle University, Newcastle upon Tyne, UK
(e-mail:{i.qiqieh1, rishad.shafik, ghaith.tarawneh, danil.sokolov, alex.yakovlev}@newcastle.ac.uk).

[‡]Shidhartha Das is with ARM, Cambridge, UK
(e-mail:{shidhartha.das}@arm.com).

Manuscript received June 4, 2018.

lower-significance bits. This can be exploited to progressively compress the product terms using bit-significance. The aim is to achieve substantial energy savings at low loss of accuracy. Inspired by our previous work [16], we 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 design-time 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 multiplier prototype design of different sizes (from 4-bit to 128-bit), and their synthesis using EDA tools to extensively analyze conflicting PEQ trade-offs, and
4) three real-application case studies demonstrating comparative advantages of the proposed approach.

To the best of our knowledge, this is the *first* demonstration of a systematic logic compression-based approximate multiplier design approach, implemented in real-application case studies. The rest of the paper is organized as follows. Section II surveys relevant research in the area of approximate multiplier design. Section III introduces the proposed approximate multiplier design. Section IV provides the error analysis associated with different bit-widths of the proposed multiplier. The experimental results and design trade-offs are described in Section V. Three real-application case studies are discussed in Section VI, Section VII and Section VIII showing comparative advantages of applying the proposed approach in Gaussian blur filter, machine learning application and power-constrained signal processing, receptively. Finally, Section IX concludes the paper.

II. Related Work

In this section, the related research efforts in the field of approximate multiplier design are discussed. These efforts can be largely categorized as modifications of either timing or functional behaviors. Firstly, timing behavior can be modified using aggressive supply voltage scaling techniques [17], [18]. 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 [19], [20]. 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 [21]. Secondly, functional modifications deal with logic reduction techniques and can be performed by relaxing the need for accurate Boolean equivalence of the specification and implementation. This can be done, to achieve energy-efficiency, accelerate computations, minimize the silicon area or optimize other system parameters.

Table I summarizes the key features and limitations of research efforts to date in the domain of approximate multipliers.

TABLE I
SUMMARY OF APPROXIMATE MULTIPLIER DESIGN APPROACHES.

Approach	Methodology	Features and Limitations
[17] [18]	Aggressive voltage scaling: lowering the supply voltage below its nominal value.	Unexpected time-induced errors, which normally impact the most significant bits.
[22] [23]	Truncation: eliminating partial products from the least significant columns.	As more columns are eliminated, the resulting errors are maximised.
[24] [25]	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.
[26]	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.
[27] [28]	Automated and evolutionary redesign: systematically reducing the complexity of circuits.	Greedy approach depending on circuit activity profile and output significance.
[29] [30]	Manual re-design: manual alteration of the functional behaviours of the structure.	Different ideas of redesigning the multiplier extend from architecture to transistor level.

For example, truncating multiplier product terms allows for the elimination of some of the least significant partial product terms [22], [23], [31], [32]. 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 [24], [25]. 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 [26] by obtaining the optimized set of partial product terms based on power-area-accuracy trade-offs. 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 [29], [30].

Automated design approaches [27], [28], [33]–[35] present design flows for synthesizing approximate circuits using circuit activity profiles and quality bounds. For example, the systematic methodology for automatic logic synthesis of approximate circuits (SALSA) [27] begins with an RT-level description of the accurate circuit and an error constraint. The approach introduces Q-function, which determines whether the quality constraints are satisfied, when comparing both approximate and original outputs. The Q-function outputs a single Boolean value. The main idea of the SALSA method is to iteratively modify the behavioral description of approximate circuit, *i.e.*, to reduce the complexity of approximate circuit, with keeping the output of the Q-function unchanged.

Recently, evolutionary circuit design shows some evolved implementations of target circuits can be considered as innovative. However, the evolutionary design approach fails in producing useful implementations for approximate complex circuits. A promising evolutionary design process was implemented within the ABC tool and extensively evaluated on functional approximation of multipliers [36].

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 target (for the first time) the energy-efficient multiplier design, using significance-driven logic compression. The proposed method can be easily applied in any multiplier architecture without the need for a special design, in contrast to related works. In addition, the error/energy/performance imposed by approximation depends on the configuration parameters, such as the size of the multiplier and depth of logic compression. Therefore, knowledge of the PEQ trade-offs permits the selection of the logic compression that maximizes the power and performance saving.

III. Proposed Approximate Multiplier Design

Without loss of generality, let us consider two N bit binary inputs for an $(N \times N)$ multiplier, the multiplicand $(A = a_{N-1}2^{N-1} + \cdots + a_0)$ and the multiplier $(B = b_{N-1}2^{N-1} + \cdots + b_0)$. The product P can be expressed as:

$$P = A \cdot B = p_{2N-1}2^{2N-1} + \cdots + p_0 = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} a_i b_j 2^{i+j} \quad (1)$$

In parallel multiplication design, computation of P is generally divided into three consecutive stages: i) partial product formation, ii) partial product accumulation, and iii) carry propagation adder. Fig. 1 shows the difference between the design stages in accurate and the proposed multiplication. First, N^2 AND gates are utilized in parallel to generate N^2 product terms of partial product bit-matrix (PPM). This matrix is then column-wise summed up by using different accumulation methods, such as *carry-save array*, *Wallace* [37] and *Dadda-tree* [38], which followed by carry-propagate adder to generate the final $2N$-bit product. The performance, hardware complexity and power consumption associated with multiplier design depend largely on the maximum height of the accumulation tree. The proposed approach decreased the number of vertical product terms in the PPM with the aim of reducing the height of the critical column in the accumulation tree. This can be achieved by following two major steps highlighted in Fig. 1 (b). In the first, lossy compression is carried out through logic clustering. The resulting compressed terms are then remapped using their commutative properties. These steps of the proposed design approach together with the variable compression method, are described below.

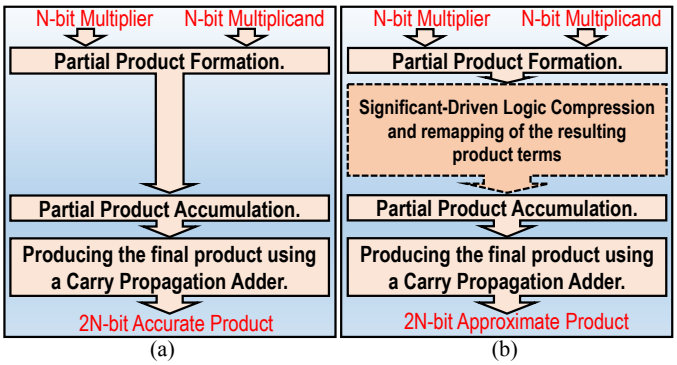

Fig. 1. Process chart showing the difference between the major stages in: (a) conventional multiplication, and (b) the proposed approach to multiplication.

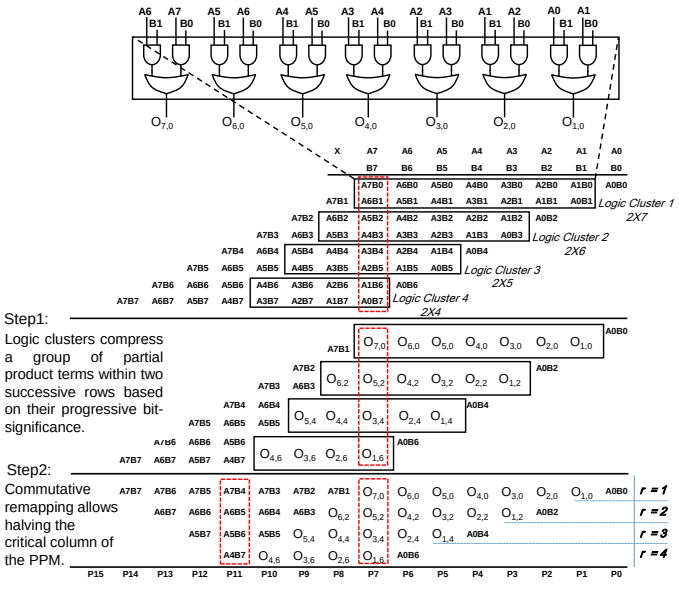

Fig. 2. Stylized demonstration of SDLC approach [16]: four different sizes of logic clusters used to compress partial products based on their progressive bit-significance in (8×8) parallel multiplier architecture.

A. Significance-Driven Logic Compression (SDLC) Approach

As can be seen in Fig. 2, this step begins to generate all partial products using N^2 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 minimize the number of rows in the PPM, thereby achieving low-complexity hardware before proceeding to accumulation. To achieve lossy compression, we follow three key principles as follows.

1) Logic Clustering: 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.

Two adjacent partial product terms belonging to the same column can be compressed in a single term by using 2-input OR gate (see Fig. 2). Let us consider two of vertically aligned bits within two successive partial products $a_i b_j$ and $a_{i-1} b_{j+1}$ of the $(i+j)^{th}$ column in the PPM. O_{i+j}^{2-bit} is an output of a logic cluster, expressed as:

$$O_{i+j}^{2-bit} = a_i b_j \vee a_{i-1} b_{j+1} \quad . \quad (2)$$

For purposes of illustration, the logic cluster of size $(2 \times L)$ targets a group of consecutive partial product terms of a length of L columns within 2 rows. 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. Fig. 2 illustrates the utilization of four sizes of logic clusters in (8×8) 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.

2) Logic Compression: Using an array of OR gates in each logic cluster compresses the partial product terms by half (see Fig. 2). A reduced set of pre-processed partial product matrix is thus ready to be accumulated by applying any convenient scheme of multiplication. In theory, a two-input OR gate is sufficient to sum up two bits, *i.e.*, '0'+'1' = '1'+'0' = '0'∨'1' = '1'∨'0' = '1' and also '0'+'0' = '0'∨'0' = '0'. However, the OR gate fails to give an accurate sum if the two inputs are "ones", *i.e.*, '1'+'1' $\neq$ '1'∨'1', the difference value is '1' as the adder returns '10' and OR outputs '1'. So, the arithmetic sum of two successive partial products belonging to the $(i+j)^{th}$ column, can be approximated as:

$$a_i b_j + a_{i-1} b_{j+1} \simeq O_{i+j}^{2-bit} \quad . \tag{3}$$

By utilizing a parallel OR compressions through logic clusters, the number of product terms inside the PPM will decreased at the price of an error when the couple of partial product terms $a_i b_j$ and $a_{i-1} b_{j+1}$ are both high, *i.e.*, the error will be when $a_i b_j + a_{i-1} b_{j+1} \neq O_{i+j}^{2-bit}$. Under assumption that the input bits a_i and b_j are uniformly and independently distributed, the probability of having this error is given by: (1/16). However, the OR compression will not affect the more significant bits of the final product as demonstrated below, moreover error analysis is discussed in Section IV.

3) Progressive Cluster Sizing: Since the main goal is to design a power-efficient multiplier with negligible loss of accuracy, the length of the logic clusters L is decreased when going down in the PPM. 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. In general, the length of logic cluster in 2-bit compression used to produce L-bit array in the r^{th} row of the compressed PPM, is given by:

$$L_{2-bit}(r) = N - r \quad . \tag{4}$$

Despite using the same number of AND gates as the accurate multiplier, this approach will deterministically reduce the hardware complexity of partial product accumulation, *i.e.*, 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.

B. Commutative Remapping

In the logic compression step (Section III-A), the number of partial product terms is reduced. This can be leveraged to minimize number of rows in the PMM prior to the accumulation stage. For this purpose, the partial product terms resulted from the logic compression are remapped based on the commutative property of the bits, *i.e.*, bits with the same weight are gathered in the same column.

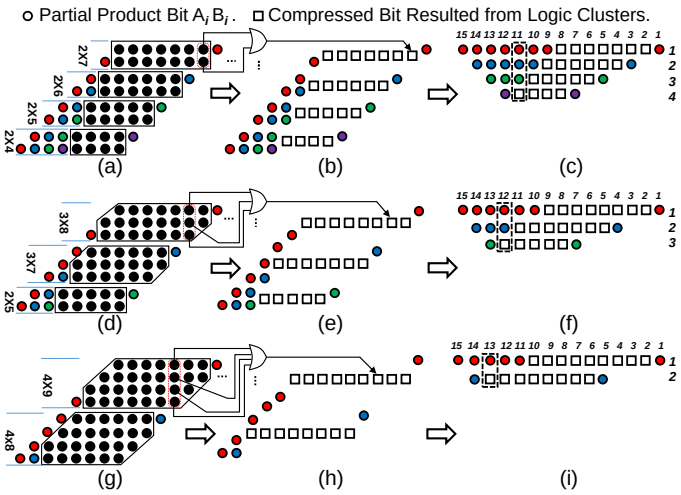

Fig. 3. Dot diagram showing the impact of increasing the depth of the logic clusters in the case of (8×8) multiplier: (a) clustering a group of bits within 2 successive rows in the partial product bit-matrix after bitwise multiplication; (b) generating a reduced set of product terms after targeting the depth of 2-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 3-bit logic compression; (g), (h) and (i) the same process when applying 4-bit logic compression. The dotted rectangles at the right indicate the heights of the critical columns which are further reduced compared to the accurate accumulation tree.

For example, in the case of $(2 \times L)$ logic cluster, the height of the critical column is reduced by half compared to the accurate accumulation tree. The compressed and remapped bit-matrix after applying commutative remapping of the bit sequence, is shown at the bottom of Fig. 2. Due to the reduced number of rows, the critical path delay is drastically shortened (see Section V). The height of the critical columns are further reduced with increased logic compression depth as seen below.

C. Variable Logic Cluster and Scalability Approach

The proposed approach is capable of achieving higher degrees of compression by increasing logic cluster depth. Comparing to the work in [16], the space of the product terms compressed by logic cluster has been modified, when the depth of logic compression spans over three or more rows in the PPM. This will provide scalability and improve the accuracy of the proposed SDLC approach. In this paper, the level of logic compression obtained from $(d \times L)$ logic cluster is denoted by d-bit logic compression, where d indicates the depth of logic cluster.

Fig. 3 demonstrates the impact of increasing depth from 2- to 3- and 4-bit logic cluster 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 and therefore more reductions in the hardware complexity and energy consumption of the multiplier design (see Section V). However, this can be achieved at a price of rational error grows when increasing the depth of logic clusters as will be demonstrated in the next section.

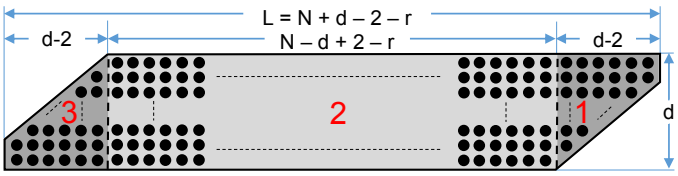

Fig. 4. Dot diagram showing the general space of targeted partial product terms compressed by a $(d \times L)$ logic cluster to produce array of L bits in r^{th} row of the reduced partial product matrix for $(N \times N)$ multiplier using SDLC approach with d-bit logic compression.

Fig. 4 presents a general space for d-bit logic cluster. In general, $(d \times L)$ logic cluster targets a group of consecutive partial product terms of a length of L columns within d rows in PPM. A $(d \times L)$ logic cluster is utilized to compress the product terms belonging to the same column into just single term using d-input OR gate. By following the same concept in (2), let us consider a depth of d successive product terms belonging to the $(i+j)^{th}$ column in the PPM $a_i b_j$, $a_{i-1} b_{j+1}$, $a_{i-2} b_{j+2}$, $\cdots$, $a_{i-(d-1)} b_{j+(d-1)}$. Thus, O_{i+j}^{d-bit} is an output term of logic cluster returned from logic compression in $(i+j)^{th}$ column and can be expressed as:

$$O_{i+j}^{d-bit} = \bigvee_{k=0}^{d-1} a_{i-k} b_{j+k} \quad . \tag{5}$$

The arithmetic sum of d successive partial products belonging to the $(i+j)^{th}$ column, can be approximated as:

$$\sum_{k=0}^{d-1} a_{i-k} b_{j+k} \simeq O_{i+j}^{d-bit} \quad . \tag{6}$$

In general, an N-row PPM resulted from accurate $(N \times N)$ multiplier can be compressed to $\lceil \frac{N}{d} \rceil$ rows, when utilizing SDLC approach with d-bit logic cluster. Let us consider r as the row index of the compressed and remapped partial product bit-matrix, in which r is ranging from 1 (the first row in the top), to $\lceil \frac{N}{d} \rceil$ (the last row at the bottom). The length of logic cluster L represents the number of columns targeted by logic compression, and equals to the length of bit array added into r^{th} row. L can be expressed by:

$$L_{d-bit}(r) = \begin{cases} (N+d-2)-r, & 1 \le r < \lceil \frac{N}{d} \rceil \\ (2N-3)-(d+1)(r-1), & r = \lceil \frac{N}{d} \rceil \end{cases} \tag{7}$$

The length of last row is separated in the second line of (7) to involve the length of the last logic cluster in case of $N \mod d \neq 0$, where the depth of the last cluster equals $N \mod d$. For instance, applying SDLC approach to (8×8) multiplier with 3-bit compression requires $\lceil \frac{8}{3} \rceil = 3$ logic clusters, the first two logic clusters compress group of product terms within 6 rows, *i.e.*, 3 rows each, leaving only two rows for the last logic cluster spans over 5 columns, see Fig. 3(d). Note that, the most significant part of PPM, where the height of the PPM is already lower or equal to $\lceil \frac{8}{d} \rceil$, logic compression is not required. In general, if i is the column index in the compressed and remapped PPM, (noting that i=1 indicates to the least significant column), the last column that can have an OR compression can be obtained as:

$$i_{last} = 2N - \lceil \frac{N}{d} \rceil - 1 \quad . \tag{8}$$

Algorithm 1 Generating the output bits of the logic cluster of the r^{th} row for the proposed $(N \times N)$ multiplier using d-bit logic clusters.

procedure LOGICCLUSTER(A, B, r)
  Output:$C[c_1, c_2, \cdots, c_L]$ //Output bits from logic cluster
  Inputs: $A[a_1, a_2, \cdots, a_N]$ //Multiplicand bits
    $B[b_1, b_2, \cdots, b_N]$ //Multiplier bits
    r //Row index of the compressed and remapped PPM

  if $\frac{N}{d} = \lceil \frac{N}{d} \rceil$ *or* $r \neq \lceil \frac{N}{d} \rceil$ **then**
//Forming (N+d-r-2)-bit output if (N mod d = 0)
    $\omega \leftarrow 1$ //initialize column index
    $length = N + d - 2 - r$
    for $x \leftarrow 1$ *to* d-2 **do**
      $C[\omega] \leftarrow (A[x+1] \wedge B[d(r-1)+1]) \vee (A[x] \wedge B[d(r-1)+2])$
      for $y \leftarrow 1$ *to* $x-1$ **do**
        $C[\omega] \leftarrow C[\omega] \vee (A[x-y] \wedge B[d(r-1)+2+y])$
      end for
      $\omega \leftarrow \omega + 1$
    end for
    for $x \leftarrow 1$ *to* $N-d+2-r$ **do**
      $C[\omega] \leftarrow (A[x-d+1] \wedge B[d(r-1)+1]) \vee (A[x+d-2] \wedge B[d(r-1)+2])$
      for $y \leftarrow 1$ *to* $d-2$ **do**
        $C[\omega] \leftarrow C[\omega] \vee (A[x+d-2-y] \wedge B[d(r-1)+2+y])$
      end for
      $\omega \leftarrow \omega + 1$
    end for
    for $x \leftarrow 1$ *to* d-2 **do**
      $C[\omega] \leftarrow (A[N-r-1] \wedge B[d(r-1)+1+x]) \vee (A[N-r] \wedge B[d(r-1)+2+x])$
      for $y \leftarrow 1$ *to* $d-2-x$ **do**
        $C[\omega] \leftarrow C[\omega] \vee (A[N-r-y] \wedge B[d(r-1)+2+x+y])$
      end for
      $\omega \leftarrow \omega + 1$
    end for

  else if $N \mod d = 1$ **then**
//Forming (2N-(d+1)(r-1)-3)-bit output if (N mod d = 1) in last row
    $length = 2N - (d+1)(r-1) - 3$
    for $x \leftarrow 1$ *to* $length$ **do**
      $C[x] \leftarrow (A[x+1] \wedge B[N])$
    end for

  else
//Forming (2N-(d+1)(r-1)-3)-bit output if (N mod d $\neq$ 1 $\neq$ 0) in last row
    $\omega \leftarrow 1$
    $length = 2N - (d+1)(r-1) - 3$
    $\acute{d} = N \mod d$
    for $x \leftarrow 1$ *to* $\acute{d}-2$ **do**
      $C[\omega] \leftarrow (A[x+1] \wedge B[d(r-1)+1]) \vee (A[x] \wedge B[d(r-1)+2])$
      for $y \leftarrow 1$ *to* $x-1$ **do**
        $C[\omega] \leftarrow C[\omega] \vee (A[x-y] \wedge B[d(r-1)+2+y])$
      end for
      $\omega \leftarrow \omega + 1$
    end for

    for $x \leftarrow 1$ *to* $length - 2(\acute{d}-2)$ **do**
      $C[\omega] \leftarrow (A[x+\acute{d}-1] \wedge B[d(r-1)+1]) \vee (A[x+\acute{d}-2] \wedge B[d(r-1)+2])$
      for $y \leftarrow 1$ *to* $\acute{d}-2$ **do**
        $C[\omega] \leftarrow C[\omega] \vee (A[x+\acute{d}-2-y] \wedge B[d(r-1)+2+y])$
      end for
      $\omega \leftarrow \omega + 1$
    end for

    for $x \leftarrow 1$ *to* $\acute{d}-2$ **do**
      $C[\omega] \leftarrow (A[N-r+1] \wedge B[d(r-1)+1+x]) \vee (A[N-r] \wedge B[d(r-1)+2+x])$
      for $y \leftarrow 1$ *to* $\acute{d}-2-x$ **do**
        $C[\omega] \leftarrow C[\omega] \vee (A[N-r-y] \wedge B[d(r-1)+2+x+y])$
      end for
      $\omega \leftarrow \omega + 1$
    end for
  end if
  return C
end procedure

The last column affected by OR approximation in the case of (8×8) multiplier is highlighted by dashed-line box in the Fig. 3 (c), (f) and (i)).

Algorithm 1 explains the process of forming L-bit array resulted from $(d \times L)$ logic cluster for any N-bit multiplier. According to (7), the length of the logic cluster depends on the row index of the compressed and remapped PPM. For all r, except the last one, the algorithm begins to generate $(N + d - 2 - r)$ bits to each row. To explain, the first $(d-2)$ less significant bits as indicated in Line (9 to 15) and highlighted in area 1 (see Fig. 4). In this area, the logic cluster compresses two partial product terms in the least significant column and increases the level of compression to $(d-1)$ partial products in the $(d-2)^{th}$ column. This is followed by forming $(N - d + 2 - r)$ bits of area 2 in Fig. 4, as referred in the Lines (16 to 22). Lastly, Lines (23 to 29) indicate generation of the more significant $(d-2)$ bits of area 3. However, when $(N \mod d) \neq 0$, the logic cluster returns $(2N-3)-(d+1)(r-1)$ bits at the last row. In such a case, if the value of $(N \mod d) = 1$, *i.e.*, logic compression in the last row is not required, the product terms generated by bitwise ANDing as the accurate PPM, see Lines (32 to 34), otherwise, when $(N \mod d) = \acute{d}$, and $\acute{d} \neq 1$ and $\acute{d} \neq 0$, the logic compression is required by using d'-bit logic cluster using the same steps, see Lines (36 to 62).

However, for purposes of illustration, each row in the compressed and remapped partial product bit-matrix can be subdivided into four parts (for instance, each row illustrated in the Fig. 3 (c), (f) and (i))

(i) the less-significant zero bits that represent the number of consecutive shifts at the beginning of each row, given by:

$$l_{zeros}(r) = d(r-1) \quad . \tag{9}$$

Then,

(ii) the first product term which is unaffected by the logic compression which can be formed as:

$$A(1) \wedge B(d(r-1)+1) \quad , \tag{10}$$

followed by,

(iii) the array of L bits returned from logic cluster given by (7) and demonstrated in Algorithm 1. Lastly,

(iv) the most significant part of the r^{th}, which is unaffected by the logic compression, the length of bits that compose the unaffected most significant part in the r^{th} is obtained as:

$$l_{MSB}(r) = \begin{cases} N - dr + 1, & 1 \leq r < \lceil \frac{N}{d} \rceil. \\ 1, & r = \lceil \frac{N}{d} \rceil. \end{cases} \tag{11}$$

The SDLC proposed approach using d-bit compression is scalable for any $(N \times N)$ multiplier, as shown in Algorithm 2. This algorithm generates a compressed and remapped partial product bit-matrix M Line (21), which can then be treated as an accumulation tree by any scheme of multiplication, such as *carry-save*, *Wallace* and *Dadda* accumulation methods. The main loop Lines (5 to 20) is responsible for forming and remapping product terms in $\lceil \frac{N}{d} \rceil$-row approximated matrix, as demonstrated in Fig. 3. Lines (7 to 10) indicate forming

Algorithm 2 Generating a reduced partial product matrix M for $(N \times N)$ multiplier using SDLC approach with d-bit logic clusters,$\forall\{\ d \in \{2, 3, ..., N\}$.

```
1:  procedure RPPM(M, A, B)
2:      Output: M = [ m_{1,1}  ...  m_{1,2N} ; ... ; m_{⌈N/d⌉,1}  ...  m_{⌈N/d⌉,2N} ]  //Compressed Matrix
3:      Inputs: A[a_1, a_2, ..., a_N]                                  //Multiplicand bits
4:              B[b_1, b_2, ..., b_N]                                  //Multiplier bits
    //Forming rows of M
5:      for r ← 1 to ⌈N/d⌉ do
6:          ρ ← 1                                          //initialize column index
    //Shifting by zeros at the beginning of row r
7:          for z ← 1 to d(r − 1) do
8:              M[r][ρ] ← "0"
9:              ρ ← ρ + 1
10:         end for
    //Forming unaffected first bit in row r
11:         M[r][ρ] ← A[1] ∧ B[d(r − 1) + 1]
12:         ρ ← ρ + 1
    //Forming outputs of logic cluster r
13:         M[r][ρ:ρ+length] ← LogicCluster(A, B, r)
14:         ρ ← ρ + length + 1
    //Forming unaffected MSBs in row r
15:         for k ← 1 to N−dr do
16:             M[r][ρ] ← A[N−r+1] ∧ B[rd+k−1]
17:             ρ ← ρ + 1
18:         end for
19:         M[r][ρ] ← A[N−r+1] ∧ B[N]
20:     end for
    //M is then treated as a reduced accumulation tree
21:     return M
22: end procedure
```

of the less-significant zero bits that represent the number of consecutive shifts at the beginning of row r, managed by (9). The first product term unaffected by the logic compression is formed in Line (11). This followed by generating L bits from the logic cluster using (7), this is shown in Line (13) by retrieving Algorithm 1. Lines (15 to 20) refer to forming the most significant part of r, which is unaffected by the logic compression, described by (11).

IV. Error Analysis

In the the previous sections, for $(N \times N)$ multiplier, generating compressed and remapped PPM using $(d \times L)$ logic clusters has been discussed. In this section, the impact of the error derived from the proposed SDLC approach, for different sizes of multipliers, is examined. Several error metrics have been discussed in [39] and [40] 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.*, $ED = |P - P'|$. Since the output of the SDLC approach is under-approximated, *i.e.*, the final product derived from the proposed approach is always lower than or equal to the product produced by accurate multiplier, ED can be expressed as, $P - P'$. The Mean Error Distance (MED) is defined as the average of the ED values and obtained as:

$$MED = \frac{\sum_{i=0}^{2^{2N}-1} ED}{2^{2N}} \quad . \tag{12}$$

Also, the mean squared error (MSE) is defined as the average of the squared ED values, which given by:

$$MSE = \frac{\sum_{i=0}^{2^{2N}-1} ED^2}{2^{2N}} \quad . \tag{13}$$

TABLE II
ERROR METRICS FOR VARYING SIZES OF PROPOSED MULTIPLIER USING 2-BIT LOGIC CLUSTER.

Bit-Width	EP(%)	MRED	NMED	NMSE	Max(RED)
4-bit	19.5	0.0277	0.0106	8.35E-04	0.31111111
6-bit	35	0.0266	0.0064	2.25E-04	0.32804233
8-bit	49.1	0.0199	0.0035	5.95E-05	0.33202614
12-bit	70.7	0.0082	0.001	3.92E-06	0.33325193
16-bit	83.9	0.0028	0.0002	2.48E-07	0.3332519

The relative error distance (RED) is another useful error metric defined as 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 [39]:

$$MRED = \frac{\sum_{i=0}^{2^{2N}-1} RED}{2^{2N}} \quad . \tag{14}$$

For comparing multipliers of different sizes, the normalized MED ($NMED$) can be expressed, using (12), as [39]:

$$NMED = \frac{MED}{P_{max}} \quad , \tag{15}$$

where P_{max} is the maximum product that can be obtained from an ($N \times N$) accurate multiplier, *i.e.*, $P_{max} = (2^N - 1)^2$. In this paper, we add normalized MSE ($NMSE$), defined, using (13), as:

$$NMSE = \frac{MSE}{P_{max}^2} \quad . \tag{16}$$

Using NMSE metric for analyzing error derived from approximate multipliers has the following advantages: i) avoids bias towards different approximate multipliers that under-approximate and over-approximate, and ii) helps to explain the impact of using approximate multipliers for a set of applications, when the user-experience metrics depends on MSE, such as the peak signal-to-noise ratio (PSNR) (see Section VI).

The 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 if $N < 16$. Monte Carlo approach is used to simulate the functional model of the proposed SDLC if the size of operands are 16-bit or more, where the simulation is running very slow. Using Monte Carlo approach provides

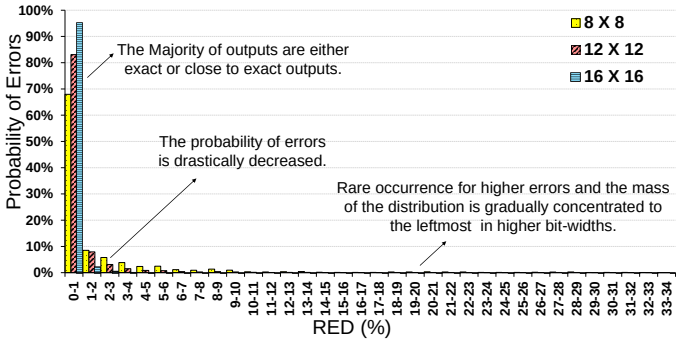

Fig. 5. Error percentage distribution for 8-bit, 12-bit and 16-bit proposed multiplier after applying 2-bit depth compression.

TABLE III
ERROR METRICS FOR DIFFERENT DEPTHS OF LOGIC COMPRESSION IN THE PROPOSED (16×16) MULTIPLIER.

Cluster-Depth	EP(%)	MRED	NMED	NMSE	Max(RED)
2-bit	83.9	2.89E-03	0.0002	2.48E-07	0.333
3-bit	94.4	1.21E-02	0.0012	5.81E-06	0.428
4-bit	97.8	3.47E-02	0.0055	1.09E-04	0.466
5-bit	98.6	4.12E-02	0.0058	1.17E-04	0.484
6-bit	99.1	5.83E-02	0.0111	4.39E-04	0.490
7-bit	99.3	5.79E-02	0.0100	3.44E-04	0.495
8-bit	99.6	1.08E-01	0.0247	2.00E-03	0.497

different distribution functions to represent all input variables, *i.e.*, multiplicand and multiplier combinations involved in the simulations. For simplicity, the inputs for the multiplier discussed in Algorithm 2 are obtained as follows. First, the size of the multiplier N and the depth of logic cluster d are selected, then both operands of multiplicand and multiplier are both assumed to be a random variables with uniform distributions for the all values between 0 and $2^N - 1$. The simulations are repeated for 2^{20} input vectors in the case of $N \geq 16$. Note that, the equations (12) to (14) are applicable when evaluating the proposed SDLC approach for all possible combinations of operands. However, for Monte Carlo simulation, the term 2^{2N} in the denominator is replaced by the number of multiplication times performed by each simulation, which is equivalent to 2^{20} in this work.

Table II shows four error metrics using varying sizes of the proposed multiplier in the case of 2-bit logic compression. 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 probability (EP) 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 III-A2.

The error Probabilities in Table II can be misleading, as the eventual impact of error is reflected in error distance metrics, such as MRED and NMED [41], [42]. Also, the readings of MAX(RED) would not denote severe degradation of the final output because the occurrence of these errors is rare. This can be seen in Fig. 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 (16×16) multiplier. As expected, increased depth leads to higher error rates (up to 99.6%) when clustering with 8-row logic compression. However, results for the MRED metric are

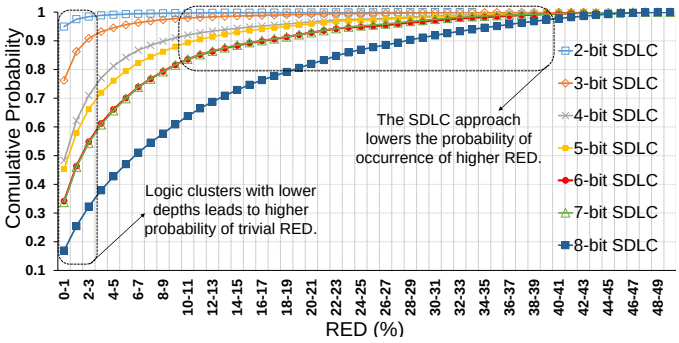

Fig. 6. Cumulative probability distribution for the error induced by different logic compression levels of the proposed SDLC approach in the case of (16 × 16) multiplier.

only marginally higher when compared with logic compression with 2- to 5-bit logic clusters. Similar observations can be made in the case of the NMED metric.

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. Fig. 6 shows the cumulative probability distribution for all possible relative errors resulting from (16 × 16) multiplier for different sizes of logic clusters. The proposed multiplier does not sacrifice the precision of the more significant bits. This is observed in the sharp rise of cumulative probability of errors towards 1, especially for lower depth of logic compression, such as 2-, 3- or 4-bit SDLC. The SDLC approach tends to produce results that are closer to the exact outputs when using lower depth of logic clusters. This is because the error occurrence increases as well due to the growing of likelihood of errors returned by higher depth logic cluster. According to (6), the probability of having inequality between the output of logic cluster and the accurate arithmetic addition is increased with higher depth logic clusters. This can be observed when the cumulative probability distributions reach to 1 faster than SDLC approach with higher depth of logic clusters, such as 6- to 8-bit SDLC. For example, in the cases of 2-, 3- and 4-bit SDLC, the probability of having errors with less than 1% RED, *i.e.*, RED of 0%-1%, is 0.95, 0.76 to 0.49 respectively. The cumulative probability exceeds 0.9 when the RED is lower than 16% for all logic cluster depths, except the case of 8-bit SDLC, where the likelihood of same RED range is just 75%. Furthermore, the cumulative probability in the cases of 6- and 7-bit logic clusters are almost identical. This can be explained using (8), as the number of the last column that can have an OR compression is the same in both cases, leading to have the same number of more significant bits unaffected by logic compression. The impact of increased degree of compression in the SDLC approach is further investigated in the application case-studies in Sections VI-VIII.

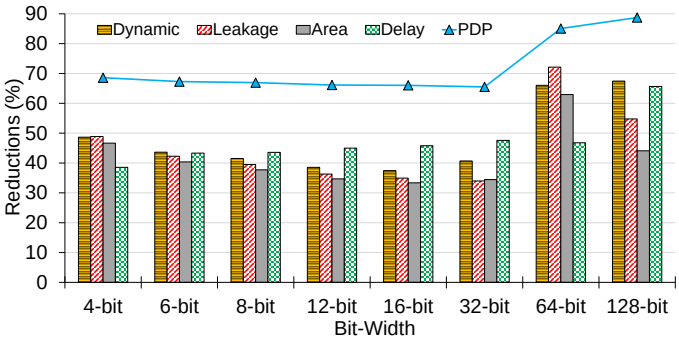

Fig. 7. Dynamic/leakage power, area, delay and PDP trade-offs for different bit-widths of the (2-bit SDLC) proposed multiplier.

V. 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. Accurate ripple adders were used in both accurate and approximate multipliers to accumulate the partial product rows within the accumulation stage (see Fig. 2). A generic SystemVerilog code was used to generate synthesizable modules for all accurate and approximate versions. These modules have been parametrized and configured differently at design time according to the bit-width of multiplier. Run-time reconfigurability of logic cluster using cost vs. degree of approximation (*i.e.* logic compression depth) trade-off is being considered for future work. The generated codes were implemented and synthesized 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 synthesizing all sizes of accurate and proposed multipliers when mapping the circuits to the Faraday's 90nm technology library [43] and evaluating for power, area, delay and energy in terms of power-delay-product (PDP). For the synthesis process, 1-V was set as nominal supply voltage for the given technology library. All designs are synthesized as combinational logic blocks with no clock constraints. This is done to allow the synthesis tool to optimize power and area freely in favor of delay. Note that introducing clock constraints can skew the synthesis tools' optimization algorithm (*e.g.* produce more delay savings at the cost of power/area optimization). However, this does not introduce overall changes between the proposed approximate multipliers and baseline designs.

Fig. 7 depicts a comparison of dynamic/leakage power, area, delay and energy trade-offs for all eight sizes of 2-bit SDLC multipliers, when compared with conventional accurate multiplier. As seen, there are significant improvements in all design trade-offs. This is basically because SDLC approach reduces the complexity of multiplier implementation by minimizing the number of rows in the accumulation tree (see

TABLE IV
Design Trade-offs for Different Bit-Widths of the Accurate Multiplier Used to Obtain Comparative Analysis in Fig. 7.

Bit-Width	Power		Area (um^2)	Delay (ns)	Energy (pJ)
	Dynamic (uW)	Leakage (uW)			
4-bit	6.4	0.7	292	0.96	0.007
6-bit	21.1	1.7	740	1.57	0.036
8-bit	47	3.2	1388	2.18	0.109
12-bit	140.7	7.5	3287	3.4	0.504
16-bit	300	13.8	5989	4.63	1.453
32-bit	1843	58	25856	9.9	18.82
64-bit	17563.6	566.9	194194	18.6	337.228
128-bit	146159	2684.9	832734	62.35	9280.417

TABLE V
NUMBER OF LIBRARY CELLS INSTANTIATED TO FORM DIFFERENT BIT-WIDTHS OF THE (2-BIT SDLC) PROPOSED MULTIPLIER.

No. of cells	4-bit	6-bit	8-bit	12-bit	16-bit	32-bit	64-bit	128-bit
Accurate	56	132	240	552	992	4.2K	27.2K	102.2K
Proposed	30	76	142	342	626	2.6K	10.7K	76.9K

Section III-B). 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, run-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. 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 minimized to 33.4%-62.9%, and energy consumed is substantially reduced in terms of PDP by 65.5%-88.7%. For all bit-widths of the proposed multiplier, the absolute readings of the synthesized designs can be obtained by combining the reduction percentages in Fig. 7 with the absolute readings of the accurate multiplier in Table IV. For instance, in the case of 32-bit proposed multiplier, the PDP readings of 5.9 pJ is derived using (dynamic/leakage) power consumptions of (1093.2 uW/38.3 uW), respectively and propagation delay of 5.19 ns, while the silicon area is 16949.3 um^2. 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. Table V lists the number of the logic cells utilized to build all sizes of the conventional and proposed multiplier designs. As expected, the SDLC approach can be employed to minimize number of cells required to implement all sizes of the proposed multiplier.

The proposed multiplier utilizes different sizes of logic

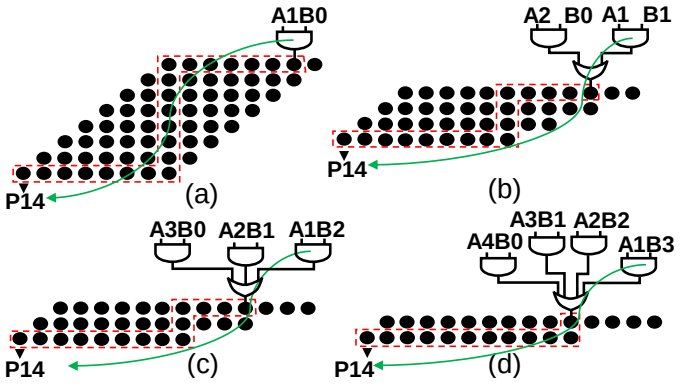

Fig. 8. Dot diagram highlights the impact of increasing depth of logic clusters on the critical path of (8×8) multiplier: (a) partial product bit-matrix of the conventional multiplier; (b) after 2-bit SDLC; (c) 3-bit SDLC; and (d) 4-bit SDLC. The dotted polygons indicate the maximum propagation path for summing up the accumulation tree. Higher degrees of compression minimize the propagation delay associated with accumulation tree (such as (b) and (c)), while a further reduction in (d), since a carry propagation adder is just needed to generate the product (no extra delay required for accumulation tree). The curved lines identify the critical paths for each multiplier (from A1 to P14).

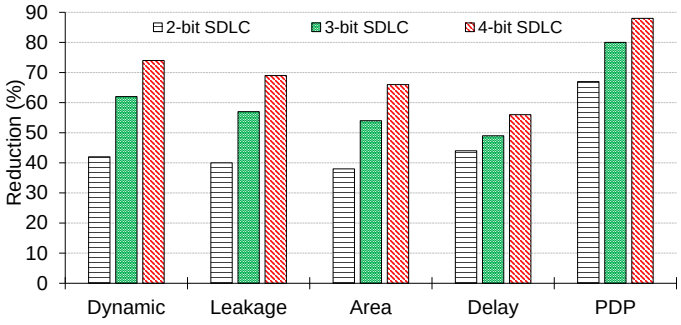

Fig. 9. Dynamic power, leakage power, delay, area and energy trade-offs for different degrees of logic compression of (8×8) multiplier.

clusters. Each logic cluster contains an array of OR gates used to compress a set of product terms. The logic clusters have no carry propagation and can also work in parallel. Hence, the delay required to generate the compressed partial product matrix is: 2-input AND gate delay (for parallel forming N^2 product terms) + d-input OR gate delay (for parallel logic compression). Then, the compressed partial product matrix is ready to be accumulated by applying any convenient scheme of multiplication, as shown in 8. Generally, the critical delay of the proposed approach depends on: (i) the size of multiplier (*i.e.* higher bit-widths increase the number of product terms and therefore the propagation delay associated with accumulation tree is also increased), and (ii) the level of logic compression (*i.e.* increased depth of the logic clusters leads to lower number of rows in accumulation tree and also lower height of the critical column). Note that, the critical path of the entire design depends also on the method of summing up the accumulation tree returned from the SDLC approach, such as carry-save array or Wallace method [10], *etc*. According to the experimental work described at the beginning of this section, the critical delay of $(N \times N)$ proposed multiplier is identified when any change in the input A(1) resulting in a change in the output of P$(2N-2)$ of the final product. See Fig. 8 as an example illustration of critical path in the case of (8×8) multiplier.

Fig. 9 illustrates the dynamic/leakage power, delay, area and energy trade-offs 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 (see Section III-C).

Fig. 10 shows comparative area, power, delay and energy advantages of our approach (with 2-bit SDLC) for different bit-widths. The comparisons are carried out with the following two existing approaches: Kulkarni [24] and ETM [44], chosen for their direct relevance to our work. In the Kulkarni approach, a large multiplier is produced using small approximate units as building blocks. The design approach of $(N \times N)$ ETM follows truncation principles by dividing the multiplier into accurate and approximate parts. Similar to N-bit fixed-width multiplier, only the most significant N bits of the final product are generated using accurate multiplications, whereas in the approximate part, a probabilistic bit manipulation is used to generate the least significant N bits of the product terms.

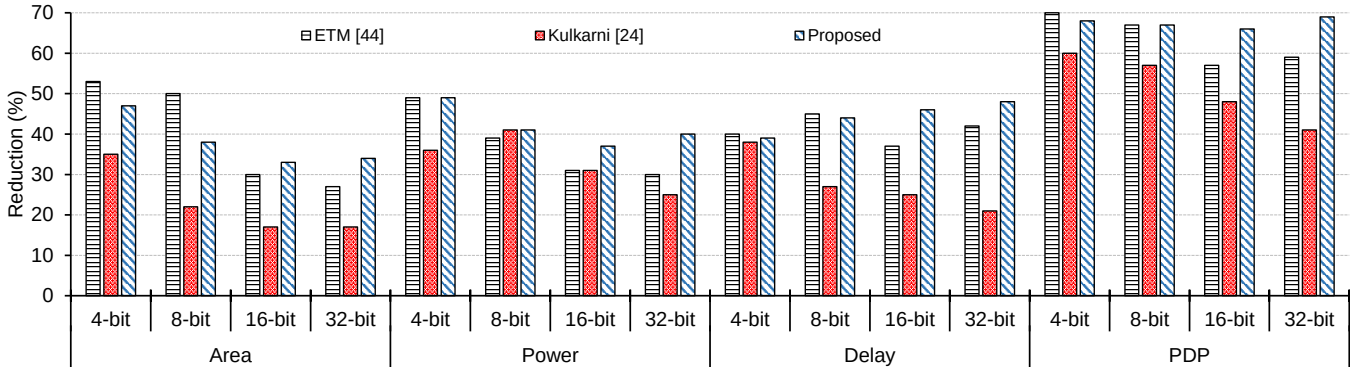

Fig. 10. Area, power, delay and PDP trade-offs for various scalable approximate multipliers (for absolute readings, combine with Table IV).

Our approach produces better results as the bit-width of the multiplier is increased. This is seen with the 16- and 32-bit multiplier. For example, in the case of 32-bit approximate multiplier, the (area and power) savings obtained from both ETM and Kulkarni are (27.2% and 30.1%) and (17.1% and 25.2%), respectively, the area and power trade-offs of the proposed multiplier is (34.5% and 40.5%), while the (delay and PDP) savings are (41.6% and 59.2%), (21.2% and 41.1%) and (47.6% and 68.8%) for ETM, Kulkarni and proposed, respectively. In such a case, the proposed approach outperforms both approaches in terms of power, area, delay and PDP savings. This is expected as the number of product rows is halved (with 2-bit clustering) and commutative remapping is used to reduce the parallel accumulation complexity. Noting that, applying higher depths of logic cluster will further increase the design gains of the propose multiplier, thereby making the proposed multiplier outperform even for lower bit-widths, such as (8×8) multiplier (see Fig. 9).

The corresponding error comparisons of these approaches are shown Fig. 11, demonstrating comparative errors (in terms of MRED, NMED and also ER) using the proposed (8×8) multiplier (with 2-bit SDLC). As expected, our approach outperforms both approaches in terms of MRED, NMED due to its bit significance-driven logic compression. For example, the MRED equals 0.252 and 0.139 for the cases of the (8×8) multiplier proposed by ETM [44] and Kulkarni [24], respectively, whereas, the MRED of the proposed multiplier is just 0.0199. As the proposed approach progressively preserves the high-order bits, it is expected to exhibit significantly lower errors for multipliers with higher bit-widths.

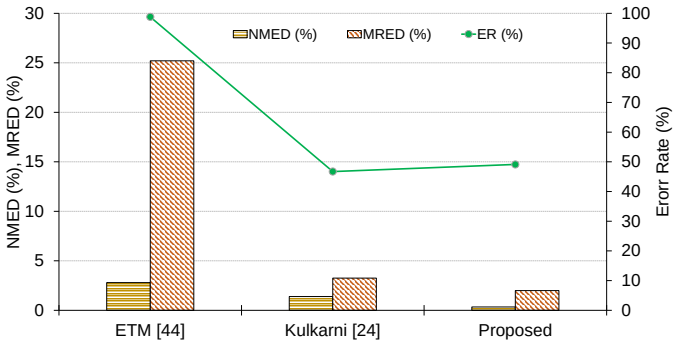

Fig. 11. Comparative errors in terms of MRED, NMED and also ER for various scalable (8×8) approximate multipliers.

For 2's complement multiplication, the SDLC approach is feasible for the case of signed operands. First, a reorganized partial-product array is created as in Baugh-Wooley multiplication [45]. Fig. 12 shows the partial product matrix in the case of (8×8) signed multiplier. Some of the product terms indicated as $q_{i,j}$ (highlighted in blue), are obtained as the NAND of the operands bits A_i and B_j. The proposed SDLC approach is then applied to minimize the height of the accumulation tree as discussed in Section III. The use of the signed multiplication does not reveal any significant degradation of the results obtained in the previous sections. An example of leveraging the proposed SDLC approach into signed multiplication is discussed in [12].

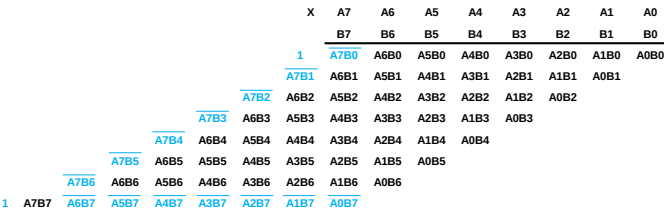

Fig. 12. Partial product matrix of (8×8) signed multiplier as in Baugh-Wooley method.

Note that combining Booth-encoding along with the proposed SDLC approach may not be as feasible. This is due to two reasons; the first is due to variable number of add/subtract operations and the second is because of long delays with isolated 1s, *e.g.* bits 001010101(0) recoded as 011111111, requiring 8 instead of 4 partial products. Also, the advantage of implementing higher radix Booth encoders to generate a reduced number of partial products comes at the expense of increased hardware complexity. However, the proposed multiplier can do the same task (*i.e.* reducing the number of partial products at low loss of accuracy) without using Booth schemes.

VI. Case Study 1: Gaussian Blur Filter

We evaluate the efficiency of the proposed technique on a Gaussian blur filter application. The application consists of additions and multiplications using key multipliers as building

blocks. Our analysis considers the Gaussian blur filter [46] 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 8-bit and 16-bit of the proposed multiplier together with the Gaussian blur algorithm. All modules are implemented in Matlab covering 2-, 3- and 4-bit depth clustering in the case of (8×8) multiplication and from 2- to 8-bit depth clustering for (16×16) multiplication. 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 and 16-bit gray-scale input images of size of (500×500) pixels. We approximate Gaussian blur by replacing the standard multiplication in the Gaussian filter with the aforementioned approximate 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:

$$PSNR = 10 \log_{10} \left(\frac{255^2}{MSE} \right) \quad , \tag{17}$$

where MSE is the mean squared-error measured with respect to the reference pixel. To calculate the consumed energy in the multiplier unit required to process the input image, we follow this equation:

$$energy = Power * Delay * N \quad , \tag{18}$$

where Power and Delay are obtained for one multiplier design from the synthesis tool. N is the number of multiplications necessary to treat the input image by Gaussian filter. The energy savings are then calculated compared to the conventional accurate multiplier. Fig. 13 and Fig. 14 demonstrate the impact of different bit-depth clustering on the image quality after applying the Gaussian blur filter. The standard multiplier and number of different levels of approximation for the proposed (8×8) and (16×16) multipliers are used.

As can be seen, the use of the SDLC approach can yield fruitful results. The PSNR for the case of 2-, 3- and 4-bit logic clustering for (8×8) SDLC are 50.2dB, 39dB, 30dB respectively, whereas the PSNR values are 70.2dB, 61.3dB, 51.4dB, 42.8dB, 39.3dB, 83.2dB, 30.2dB, when treating the images using 2- to 8-bit logic clustering for (16×16) SDLC. The values of PSNR are computed compared to the image resulting after applying Gaussian blur filtering with the case of accurate multiplication. Thus, the proposed approach can provide a significant dynamic energy saving up to 80.1% with acceptable quality of output image, especially when utilizing smaller bit depth clusters such as 2- and 3-bit for (8×8) SDLC and 2- to 6-bit for (16×16) SDLC. As can be observed from Fig. 14, the proposed (16×16) multiplier allows for more levels of energy/quality trade-offs ,comparing to the output quality of (8×8) multiplier in Fig. 13.

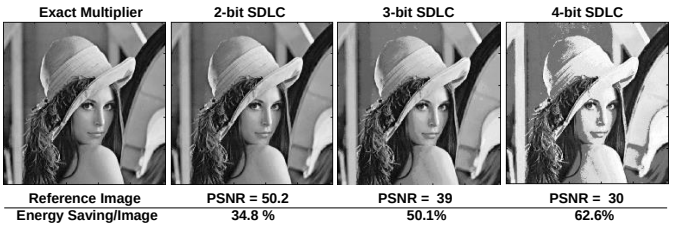

Fig. 13. Output quality after applying Gaussian blur filtering for different degrees of logic compression of the proposed (8×8) multiplier.

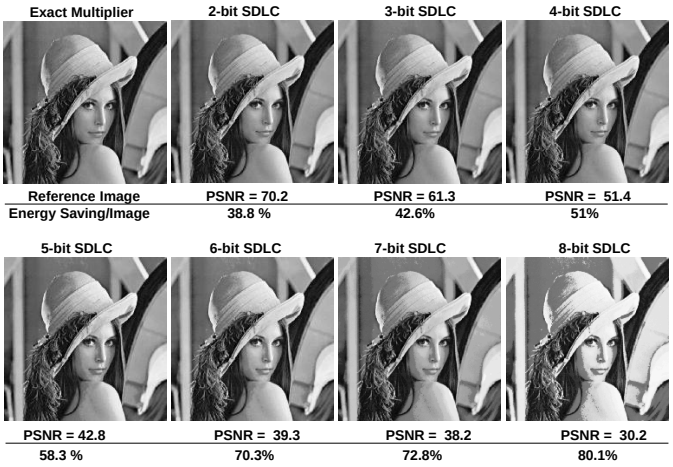

Fig. 14. Output quality after applying Gaussian blur filtering for different degrees of logic compression of of the proposed (16×16) multiplier.

VII. Case Study 2: Perceptron Classifier

We evaluate the proposed multiplier against a perceptron classifier. Perceptron classifiers are widely used in machine learning applications. We exploited perceptron for learning a binary classifier, *i.e.* a function which takes the inputs $x_1, x_2, ..., x_m$ and produces an output value y. The output y is a single binary value, expressed as:

$$y = \begin{cases} +1, & w \cdot x + b > 0, \\ -1, & \text{otherwise}, \end{cases} \tag{19}$$

where w is a vector of real-valued weights, which vary over runtime depending on the number of training input samples and the training rate, $w \cdot x$ is the dot product, *i.e.* $\sum_{i=1}^{m} w_i \cdot x_i$, m is the number of inputs to the perceptron and b is the bias (0 used in our example). We used (19) to classify patterns that are linearly separable [47].

For perceptron learning algorithm, a training set is used to train the perceptron to classify inputs correctly. This is accomplished by adjusting the connecting weights and the bias to properly handle linearly separable sets. All input sets are randomly generated and independently distributed from 0 to 65535 to allow for using 16-bit multiplication. Then, we evaluate the classifier against test set of 1000 two-dimensional points that belong to two classes [-1,+1], see (19). The approximate multipliers are used to multiply the perceptron inputs by the weights vectors. The error rate (ER) is the ratio of mismatch between classified class and the actual output. Fig. 15 demonstrates the comparison of the classification problem with accurate (16×16) multiplier and the proposed

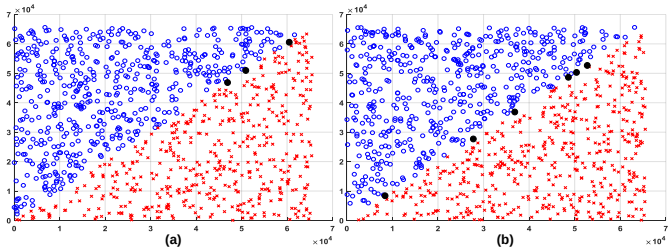

Fig. 15. The test set perceptron classification using; (a) accurate multiplier; (b) 2-bit SDLC proposed multiplier. (blue and red points represent two classes -1 and +1, black dots for mismatch classification points.)

TABLE VI
ERROR RATE RESULTS AND ENERGY SAVINGS FOR PERCEPTRON CLASSIFIER

16-bit Multiplier	ER%	Energy savings%
Accurate	0.3	–
Proposed (2-bit SDLC)	0.6	40.2
Proposed (3-bit SDLC)	2.2	68.7
Proposed (4-bit SDLC)	12.7	74.2
ETM [44]	12.1	38.2
Kulkarni [24]	6.7	27.4

2-bit SDLC design. Compared to the accurate multiplier, the proposed SDLC multiplier classifies six points, from the 1000 points in the testing set, as class 1 by mistake. Note that, even the design that uses the accurate multiplier cannot classify all points correctly (three mismatched points). Table VI shows the comparative error rates and energy advantages of our approach and various (16×16) multipliers. The energy savings is calculated by (18). The proposed approach outperforms the other designs and can provide energy saving of up to 74.2% with acceptable error rates, especially when utilizing lower bit depth clusters such as 2- and 3-bit SDLC. However, for 4-bit SDLC, the increased ER in Table VI, is expected. This is because higher depth of logic clusters affects the accuracy when multiplying the inputs with their associated weights. When compared with existing approaches, such as ETM [44] and Kulkarni *et al.* [24], the 4-bit SDLC produces a solution with comparable ER.

VIII. Case Study 3: Power-Constrained Signal Processing

This case study serves as an exemplar of how the PEQ trade-offs of different approximate multiplier configurations (Section V) can be leveraged to extract maximal data processing capability in power-constrained applications. The main idea is to design multiple data processing units with different power and quality elasticity, provided by configurable logic compression feature of our proposed SDLC approach. When more power is available, accurate processing units can be used; however, when the power is limited data processing units with progressively higher approximation can be suitably selected. The aim is to deliver the best possible data processing capability, mitigating low-power situations where traditional accurate logic circuits cannot operate.

Fig. 16 shows the simplified block diagram using signal convolution as an example. As shown, the application is power-constrained as no direct energy storage is available, *i.e.*, the incoming harvested power is fed into the target logic circuit

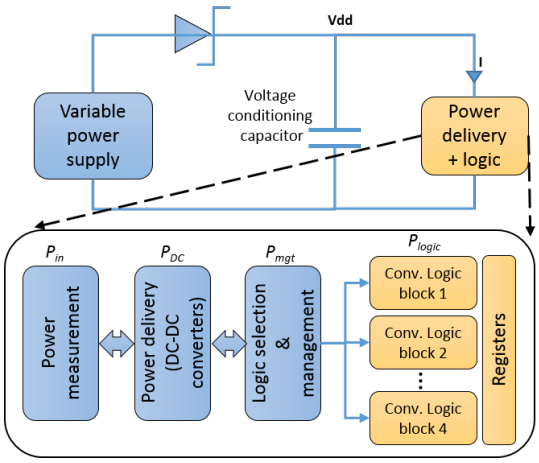

Fig. 16. Power-constrained image convolution circuitry

through a voltage protection and conditioning unit. The logic block consists of power measurement subsystem using shunt resistance network (not shown for simplicity), followed by three different DC-DC converters to enable variable voltage-current supply into the main logic circuit. Based on the incoming power, one of the four convolution logic blocks with suitable approximation is chosen such that incoming power budget is not violated, while also maximizing the quality of data processing functionality.

The circuit in Fig. 16 was designed with simulated power scavenging with the assumption that the incoming power does not vary faster than the period of synchronous logic clock. Since multipliers constitute the bulk proportion of the convolution logic blocks, they were designed with 16-bit multipliers of four different approximations: accurate multiplier, approximate multiplier with 2-, 3- and 4-bit logic clusters (see Section III-C). All four convolution configurations used precise carry-propagation adders organized in array of multiply-accumulate (MAC) units.

Table VII shows the synthesized power, delay, area and PDP comparisons of the different MAC units including the four convolution circuits in our proposed approach. As can be seen, the accurate MAC (row 2) has significantly higher power consumption and delay compared with the approximate MAC implementations (rows 3-5). As the logic compression level is increased using 2-bit to 3-bit and 4-bit logic clusters, the critical path is incrementally cut down in favor of reduced dynamic and leakage power, coupled with latency. As a result, up to 5.5x energy efficiency expressed in terms of PDP, can be achieved in the case of 4-bit SDLC MAC. Note that the energy reductions are achieved at the cost of reduced quality (see Section IV) and 140% increased overall area.

TABLE VII
POWER, DELAY AND AREA COMPARISONS OF DIFFERENT LOGIC COMPONENTS

Circuit	P_{dyn} (uW)	P_{leak} (uW)	Delay (ns)	Area (um^2)	PDP (fJ)
Accurate	58.19	4.23	2.63	1417.47	174.27
2-bit SDLC	36.24	2.97	2.11	904.56	76.86
3-bit SDLC	28.90	2.40	1.73	672.31	49.66
4-bit SDLC	23.41	2.01	1.35	501.37	32.32
Proposed	–	–	–	*3495.71*	–

Fig. 17 presents the simulation results of the application (Fig. 16). In a power-constrained scenario, the system requires the signal convolution tasks to be completed in units of 2700 MACs (300-sample packets being convoluted by a signal with 9 samples). No deadline is imposed for the number of packets to be processed over a given time in this example. Fig. 17(a) depicts the input power (P_{in}) scavenged using a simulated source, together with the effective logic power (P_{logic}) for consecutive time intervals of $50ms$ each. The logic selection and management subsystem (Fig. 16) estimates P_{budget} for the convolution circuit discounting the losses in power delivery. The P_{logic} determines which logic mode can be operated to ensure each signal packet is processed with the available power. The excess power (P_{loss}) is bypassed through an RC network parallel to the logic block (not shown). As can be seen, when higher power is available initially, the accurate MACs are selected for the convolution task, allowing up to 2 packets to be processed (Fig. 17(b)). However, when scavenged power is low, logic modes (such as 2-bit, 3-bit or 4-bit SDLC MACs) are selected. As the power becomes lower than that required for processing a packet, the computation is skipped to the next interval. The selection of logic modes has a direct impact on the quality of the outcomes as shown in Fig. 17(c). Processing convolution using 4-bit SDLC MACs causes the SNR to degrade to as low as 19 dB. Note that the effective SNR for a given packet can still vary despite having the similar logic mode selection. This is because the error introduced by the approximate logic block is dependent on the signal values. Signals with higher numeric values (*i.e.* higher '1's in the significant parts of the logic) can incur more errors than those with less numeric values (see Section IV).

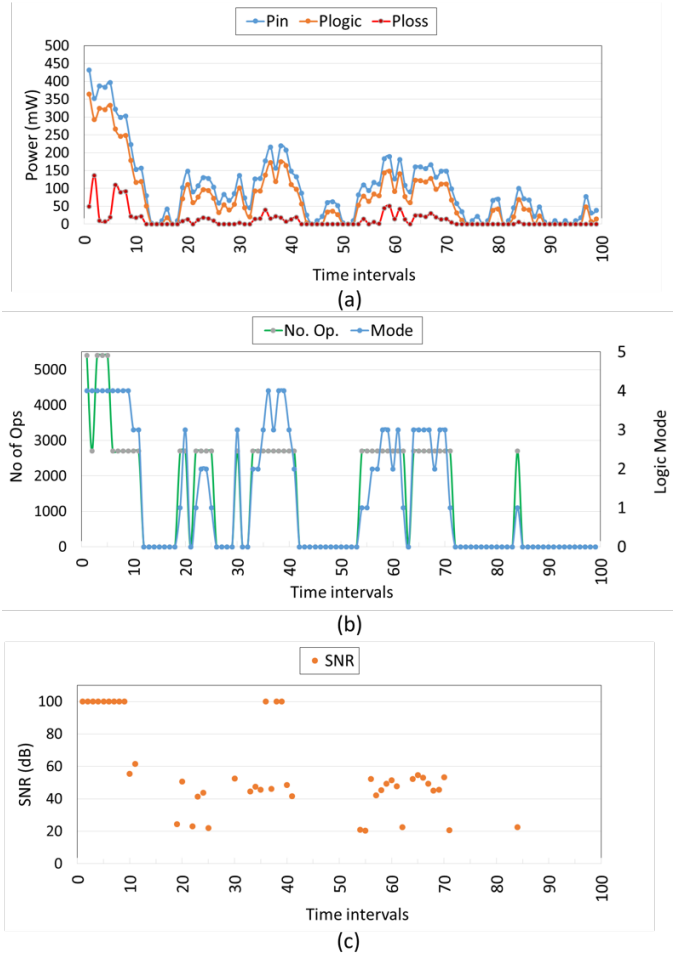

Fig. 17. (a) Input power (P_{in}) and effective logic power (P_{logic}), together with the logic bypass losses (P_{loss}), (b) logic mode selection [0: none selected, 1: 4-bit logic cluster MACs, 2: 3-bit logic cluster MACs, 3: 2-bit logic cluster MACs, and 4: accurate MACs] and the total number of MAC operations performed for the power budget for logic subsystem determined from (a), and (c) valid signal-to-noise (SNR) points for the selected convolution tasks.

The PEQ elasticity offered by SDLC multipliers substantially improves the data processing capability. For example, the accurate MAC (mode 4) can only be used to complement up to 2.7k MACs as opposed to 150k MACs (about 60x) produced by the combination of circuits (mode 0-4), including our SDLC multipliers. This is a major advantage of systems design with approximate circuits of variable precision.

IX. 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 reduced 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. Three case studies were set up. The first case study shows the performance-energy-quality (PEQ) trade-offs of SDLC multiplier applied in an image processing application. The second case study evaluates the SDLC approach in machine learning application using perceptron classifier. The third case study takes advantage of PEQ elasticity of approximate multiplier configurations. It was shown that, such elasticity can substantially improve the data processing capability when operating with real-world power budgets in ubiquitous systems. 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. Run-time configurable approximate multiplier design is being considered for future research.

Acknowledgment

The authors would like to thank MOHE (Jordan), Al-Balqa Applied University (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 Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, 2016.

[2] V. De, "Energy-efficient computing in nanoscale cmos," *IEEE Design Test*, vol. 33, no. 2, pp. 68–75, 2016.

[3] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Cross-layer approximate computing: From logic to architectures," in *DAC*, 2016, pp. 1–6.

[4] L. Sekanina, "Introduction to approximate computing: Embedded tutorial," in *DDECS*, 2016, pp. 1–6.

[5] M. D. Ercegovac, "On approximate arithmetic," in *Asilomar Conference on Signals, Systems and Computers*, 2013, pp. 126–130.

[6] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini, "A transprecision floating-point platform for ultra-low power computing," in *DATE*, 2018, pp. 1051–1056.

[7] H. Jiang, C. Liu, N. Maheshwari, F. Lombardi, and J. Han, "A comparative evaluation of approximate multipliers," in *International Symposium on NANOARCH*, 2016, pp. 191–196.

[8] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *ETS*, 2013, pp. 1–6.

[9] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "Impact: Imprecise adders for low-power approximate computing," in *ISLPED*, 2011, pp. 409–414.

[10] I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov, S. Das, and A. Yakovlev, "Energy-efficient approximate wallace-tree multiplier using significance-driven logic compression," in *IEEE International Workshop on Signal Processing Systems (SiPS)*, 2017, pp. 1–6.

[11] K. Al-Maaitah, G. Tarawneh, A. Soltan, I. Qiqieh, and A. Yakovlev, "Approximate adder segmentation technique and significance-driven error correction," in *PATMOS*, 2017, pp. 1–6.

[12] D. Esposito, A. G. M. Strollo, and M. Alioto, "Low-power approximate MAC unit," in *PRIME*, 2017, pp. 81–84.

[13] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate data types for safe and general low-power computation," in *PLDI*, 2011, pp. 164–174.

[14] K. Jain and V. V. Vazirani, "Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation," *JACM*, vol. 48, no. 2, pp. 274–296, 2001.

[15] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," *SIGPLAN Not.*, vol. 47, no. 4, pp. 301–312, 2012.

[16] I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov, and A. Yakovlev, "Energy-efficient approximate multiplier design using bit significance-driven logic compression," in *DATE*, 2017, pp. 7–12.

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

[18] 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, no. 4, pp. 517–526, 2010.

[19] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner, "Razor: circuit-level correction of timing errors for low-power operation," *IEEE Micro*, vol. 24, no. 6, pp. 10–20, 2004.

[20] S. Ghosh, S. Bhunia, and K. Roy, "CRISTA: A new paradigm for low-power, variation-tolerant, and adaptive circuit synthesis using critical path isolation," *IEEE TCAD/ICAS*, vol. 26, no. 11, pp. 1947–1956, 2007.

[21] K. Shi, D. Boland, E. Stott, S. Bayliss, and G. A. Constantinides, "Datapath synthesis for overclocking: Online arithmetic for latency-accuracy trade-offs," in *DAC*, 2014, pp. 1–6.

[22] 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 TCAS-I: Regular Papers*, vol. 57, no. 6, pp. 1312–1325, 2010.

[23] J. E. Stine and O. M. Duverne, "Variations on truncated multiplication," in *DSD*, 2003, pp. 112–119.

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

[25] C. H. Lin and I. C. Lin, "High accuracy approximate multiplier with error correction," in *ICCD*, 2013, pp. 33–38.

[26] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris, and K. Pekmestzi, "Design-efficient approximate multiplication circuits through partial product perforation," *IEEE Transactions VLSI Systems*, vol. 24, no. 10, pp. 3105–3117, 2016.

[27] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: Systematic logic synthesis of approximate circuits," in *DAC*, 2012, pp. 796–801.

[28] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "Macaco: Modeling and analysis of circuits for approximate computing," in *ICCAD*, 2011, pp. 667–673.

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

[30] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE TCAD/ICAS*, vol. 32, pp. 124–137, 2013.

[31] E. E. Swartzlander, "Truncated multiplication with approximate rounding," in *Proc. of 33rd Asilomar Conference on Signals, Systems, and Computers*, vol. 2, 1999, pp. 1480–1483.

[32] J. M. Jou, S. R. Kuang, and R. D. Chen, "Design of low-error fixed-width multipliers for DSP applications," *IEEE TCAS-II: Analog and Digital Signal Processing*, vol. 46, no. 6, pp. 836–842, 1999.

[33] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, "Abacus: A technique for automated behavioral synthesis of approximate computing circuits," in *DATE*, 2014, pp. 361:1–361:6.

[34] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *DATE*, 2013, pp. 1367–1372.

[35] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, "ASLAN: Synthesis of approximate sequential circuits," in *DATE*, 2014, pp. 1–6.

[36] M. Ceska, J. Matyas, V. Mrazek, L. Sekanina, Z. Vasicek, and T. Vojnar, "Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished," in *ICCAD*, 2017, pp. 416–423.

[37] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-13, no. 1, pp. 14–17, 1964.

[38] L. Dadda, "Some schemes for parallel multipliers," *Alta frequenza*, vol. 34, no. 5, pp. 349–356, 1965.

[39] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Transactions on Computers*, vol. 62, no. 9, pp. 1760–1771, 2013.

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

[41] I. Chong, H.-Y. Cheong, and A. Ortega, "New quality metric for multimedia compression using faulty hardware," in *VPQM for Consumer Electronics*, 2006, pp. 267–272.

[42] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *DAC*, 2013, p. 113.

[43] "Faraday technology corporation," http://www.faraday-tech.com, accessed: 2017-12-12.

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

[45] C. R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Transactions on Computers*, vol. C-22, no. 12, pp. 1045–1047, 1973.

[46] C. Solomon and T. Breckon, "Fundamentals of digital image processing : a practical approach with examples in matlab." Wiley-Blackwell, 2011, ch. 4, pp. 95–96.

[47] Y. Freund and R. E. Schapire, "Large margin classification using the perceptron algorithm," *Machine learning*, vol. 37, no. 3, pp. 277–296, 1999.

Issa Qiqieh received the B.S. degree and M.Sc. degree from Hijjawi Faculty for Engineering Technology, Yarmouk University, Irbid, Jordan, in 2005 and 2007 respectively. He is currently pursuing Ph.D. degree from the School of Engineering, Newcastle University, Newcastle upon Tyne, UK. He was a full-time lecturer in the School of Engineering, Al-Balqa' Applied University, Al-Salt, Jordan, from 2007 to 2014. His current research interests include approximate circuits, power-adaptive computing and energy-efficient arithmetic designs.

Rishad Shafik (MIET, MIEEE) is a Lecturer in Electronic Systems within the School of Engineering, Newcastle University, UK. Dr. Rishad received his Ph.D., and M.Sc. (with distinction) degrees from Southampton in 2010, and 2005; and B.Sc. (with distinction) from the IUT, Bangladesh in 2001. He is one of the editors of the book "Energy-efficient Fault-tolerant Systems," published by Springer USA. He is also author/co-author of 85+ IEEE/ACM journal and conference articles, with three best paper nominations. He has recently co-chaired 30th DFT2017 (www.dfts.org) at Cambridge, UK. His research interests include energy-efficiency and adaptability aspects of embedded computing systems.

Ghaith Tarawneh is a Research Associate at Newcastle University. He finished his PhD in 2013, specializing in metastability, synchronization and clock domain crossing. Since then, he has worked in various digital design areas including on-chip parameter sensing, network on chip, neuromorphic computing, formal verification and EDA tool development. He is currently a member of the Partially Ordered Event Triggered Systems (POETS) project that is developing a novel architecture for massively parallel computing.

Danil Sokolov is a Senior Research Associate in the School of Engineering, Newcastle University, UK. He received PhD from Newcastle University in 2006. In the period 2007-2010 he has been employed as a consultant by an EDA start-up Elastix Corp to lead R&D in the area of ultra low power synthesis. Dr. Sokolov is the author of 14 journal articles, over 40 pair-reviewed conference papers and a chapter in a textbook System On Chip: Next Generation Electronics. His research expertise is in modelling of self-timed heterogeneous systems, synthesis methods for energy-efficient circuits, development of CAD tools and their integration into industry-level design flows. He is the lead architect and developer of Workcraft – a toolset for capture, simulation, synthesis and verification of interpreted graph models.

Shidhartha Das (MIET, MIEEE) is currently the Principal R&D Engineer at ARM, and the recipient of the ARM Inventor of the Year award in 2016. He received the B.Tech degree from the IIT, Bombay in 2002 and the M.S and Ph.D degrees from the University of Michigan, Ann Arbor in 2005 and 2009. His research interests include emerging non-volatile memory technologies, micro-architectural circuit and systems design. He is the recipient of multiple best paper awards; his research also featured in popular IEEE magazines. Dr. Das serves on the technical program committee of several leading international conferences.

Alex Yakovlev (FIET, FIEEE) is a Professor of Computer Engineering, who founded and leads the MicroSystems Research Group, and co-founded the Asynchronous Systems Laboratory at Newcastle University. He was awarded an EPSRC Dream Fellowship in 2011–13. He has published 8 edited and co-authored monographs and more than 300 papers in IEEE/ACM journals and conferences, in the area of concurrent and asynchronous systems, with several best paper awards and nominations. He has chaired organizational committees of major international conferences. He has been principal investigator on more than 30 research grants and supervised 40 PhD students. Most recently, he has been elected to the fellowship of Royal Academy of Engineering in the UK.