IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 29, 2019, accepted April 16, 2019, date of publication April 29, 2019, date of current version May 9, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2913743

Truncated SIMD Multiplier Architecture for Approximate Computing in Low-Power Programmable Processors

ROBERTO R. OSORIO AND GABRIEL RODRÍGUEZ
Department of Computer Engineering, CITIC, Universidade da Coruña, 15071 A Coruña, Spain

Corresponding author: Roberto R. Osorio (roberto.osorio@udc.es)

This work was supported in part by the Ministry of Economy and Competitiveness of Spain under Project TIN2016-75845-P (AEI/FEDER, UE), in part by the Xunta de Galicia and FEDER Funds of the EU under the Consolidation Program of Competitive Reference Groups under Grant ED431C 2017/04, and in part by the Centro Singular de Investigación de Galicia Accreditation 2016–2019 under Grant ED431G/01.

ABSTRACT Approximate computing has been exploited for many years in application-specific architectures. Recently, it has also been proposed for low-power programmable processors. However, this poses some challenges as, in a microprocessor, the energy consumed by fetching and decoding an instruction may be significantly higher than that of the execution itself. Therefore, approximate computing would be advisable only for those instructions, in which the execution stage is significantly expensive in terms of energy consumption. In this paper, we present new architectures for truncated SIMD multipliers able to calculate signed and unsigned products from 8×8 to 64×64 bits. Next, we analyze the precision loss incurred by truncation for all product sizes. We implement accurate and truncated architectures for both scalar and SIMD products and find that truncation allows area savings of up to 27%. The proposed design is experimentally evaluated in different scenarios, showing potential energy savings ranging from 29% to 42%. Finally, this paper analyzes the overall convenience of introducing truncated SIMD architectures with respect to accurate SIMD and scalar architectures.

INDEX TERMS Digital arithmetic, fixed-point arithmetic, approximate computing, approximate multiplier, low power.

I. INTRODUCTION

Approximate computing is a well-studied field of research that pursues to trade off computation accuracy in exchange for energy, time, and/or area savings. Some ideas for achieving these goals are using reduced precision in floating point arithmetic; reducing the accuracy of quotients, square roots, and scientific functions; or truncating sums and products. Traditionally, these ideas have been applied in application-specific architectures only. More recently, a number of papers have proposed to extend this reach, applying approximate computing to general purpose, low power, programmable processors [9], [12], [23]. Software and hardware techniques proposed in the past have to be repurposed for this more general application. This paper focuses on improving the precision and efficiency of approximate multiplications.

The associate editor coordinating the review of this manuscript and approving it for publication was Vyasa Sai.

Approximate computing should not be applied to any arbitrary instruction in a program. As an example, operations affecting control flow, such as computation of loop indices, must be executed accurately to avoid semantic errors. A possible solution to this problem is to implement hybrid units, in which truncated arithmetic is implemented using power gating [2]. In this way, the same circuit may implement accurate or approximate operations by switching power supply at the gate level, and rerouting signals appropriately through multiplexers. However, this presents disadvantages. First, if left connected to other signals, power gated sections increase circuit capacitance, affecting propagation delays. Second, switching between the approximate and precise operation modes takes time, causing delays in the execution of sections of code involving mixed arithmetic, and potentially canceling any power savings. These problems may be solved in the future by micro-electro-mechanical relays [10]. The inclusion of separate precise and approximate functional

units in microprocessors has been proposed to avoid the disadvantages of power gated designs [9]. The inclusion of different specialized arithmetic units is one possible approach to taking advantage of the extra transistors in the dark silicon era [8], [22]. However, having both units enabled at the same time implies a static power overhead which potentially cancels energy savings.

From a global perspective, the main limitation to exploit approximate computing on programmable processors comes from the fact that instruction fetch and control stages of the pipeline have been shown to consume more than 90% of the energy in simple arithmetic instructions such as addition and small products [13]. Provided that non-execution stages cannot be approximated, research should be focused on instructions that carry out heavy computations in the execution stage. In order to increase the share of energy dissipated in the execution stages compared to the control stages, this paper targets SIMD operations. The focus is put on multiplication as it is both a frequently used (as opposed to other more complex ones, such as division), and a relatively complex operation, offering opportunities for significant power savings (as opposed to simpler ones, such as addition). A scalar 32-bit multiplication consumes only 4.4% of the total instruction energy, which includes instruction cache and register access, as well as control [13]. We expect this share to increase to 7.8%, 12.7%, 18.5%, and 24.0% for 64-, 128-, 256-, and 512-bit vectors, respectively, assuming that the control and instruction cache energies remain constant, while register access and multiplication energies scale linearly with word size. This work focuses exclusively on integer multiplication. For floating point other solutions already exist. To reduce the energy overhead incurred by dual functional units, as well as to avoid the switching delay between precise and approximate modes in gated architectures, the proposed SIMD design can be used to implement accurate products of small word lengths, as described in Section IV. This work makes the following contributions:

- We analyze the error introduced by different-sized truncated multipliers, highlighting the effect of overflow in unsigned multipliers, and proposing a worst-case-based approach to evaluation in large multiplication sizes.
- An approach to the computation of correction factors (α and λ) of 32- and 64-bit truncated multipliers is proposed.
- A novel SIMD architecture for truncated approximate multipliers is presented. To the best of our knowledge, no other works have tackled this topic before. SIMD is crucial in our strategy to obtain energy savings when implementing approximate computing in programmable processors.
- A thorough experimental evaluation of the proposed architecture, including an estimation of: area savings; energy consumption in the multiplier unit itself and in the full datapath; and estimated energy savings for a real multimedia application, JPEG encoding and decoding.

This work is structured as follows. Section II introduces Booth multipliers, together with the basics of SIMD multiplication. In Section II-B a review of approximate multipliers is presented, with a focus on applying correction factors after truncation. Section III studies the error introduced by truncated multipliers. In Section IV, our architecture for truncated SIMD multiplication is explained in detail, which is then analyzed in Section V in terms of area and energy savings. Finally, Section VI concludes the paper.

II. MULTIPLICATION REVIEW

The Booth multiplication algorithm [3] is the most widely used in hardware implementations, as it can be used with both signed and unsigned numbers. The multiplicand is kept unchanged, while the multiplier is recoded into a signed digit format. The most common implementation is the radix-4 modified Booth algorithm, in which signed digits can take values within the set {-2, -1, 0, 1, 2}. The practical realization is quite simple and the number of partial products (*PPs*) is typically $\lfloor n/2 \rfloor$ for an $m \times n$ product. Figure 1(a) shows the basic implementation of a signed 8×8 multiplier, where PPs are labeled a to d. Each PP is sign extended, as it may be either positive or negative. Not taking into account sign extension, PPs are $(n+1)$ bits long as they may be multiplied by 2 or -2. Multiplying by -1 or -2 requires negating the multiplicand, which is accomplished by complementing its value and adding one unit. Those units are typically called *hot ones*, and they are labeled as x_{h-o} in the figure.

Sign extension can be avoided with some improvements in the algorithm [11], as shown in Figure 1(b). The prefix bits x_{cs} take either the value of the sign, or its complement depending on the PPs and their position. A simple rule allows to calculate the exact value for each bit. Implementing unsigned products is also possible by adding another PP, labeled e in Figure 1(c). The PPs and the hot ones are reduced by a redundant carry-save adder (CSA) using either a Wallace [24] or a Dadda [6] tree. Finally, a non-redundant adder produces the final result.

A. INTEGER SIMD MULTIPLIERS

Single-Instruction Multiple-Data instructions are supported in most modern processors targeting multimedia applications. Among SIMD instructions, integer multiplication is both power-hungry and extensively used. As such, we consider it a natural choice to test our approach.

The SIMD multiplier proposed by [7] is taken as a reference. It describes an architecture that computes full precision 8×8, 16×16, 32×32 and 64×64 signed and unsigned integer products. In all cases, 64-bit operands produce a 128-bit result. Pipelining is not considered, as the number of stages would depend on the characteristics of the targeted microprocessor. Although the area cost and energy consumption of pipeline registers cannot be ignored, it is reasonable to consider that the number of flip-flops is proportional to the number of logic gates for a given number of pipeline stages. As such, the energy and area characteristics

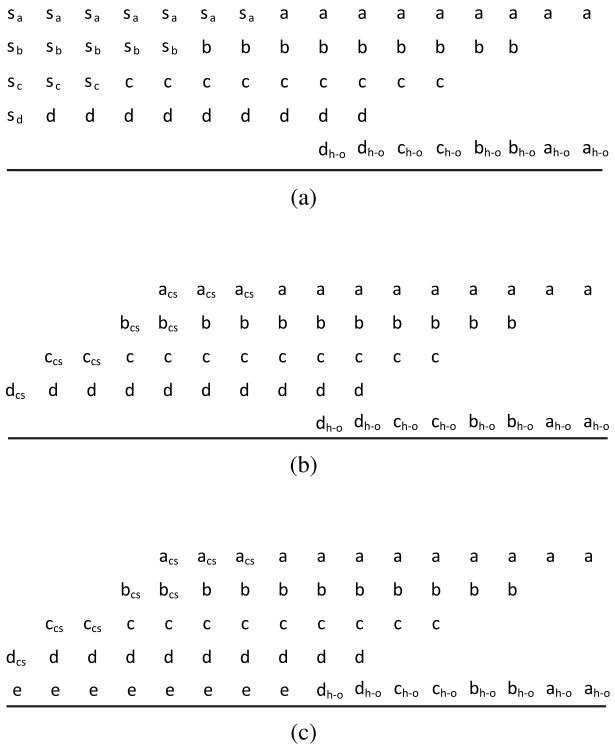

FIGURE 1. Different addition matrices for an 8 × 8 multiplier for signed and unsigned inputs: (a) original modified Booth with sign-extended partial products; (b) improved algorithm to avoid full sign extension; and (c) addition of a new PP to support unsigned multiplications.

for the non-pipelined design should therefore translate to the pipelined one without significant changes.

An efficient implementation aims at reusing as many components as possible for all the word lengths. Reference [7] propose two possible architectures. The one called *shared segmentation* is more convenient, as it is easier to support the different word lengths in the CSA tree. We now summarize that architecture in order to support the explanations of the truncated SIMD architecture presented in Section IV.

The architecture by [7] is depicted in Figure 2. The PPs are arranged in 32 staggered rows, plus an extra row that holds the hot ones and enables unsigned multiplication. On top of this basic structure, other operands could be added to implement fused multiply-add instructions, or dot product. Data is distributed in such a way that each of the first 32 rows contains exactly one PP, independently of the word length. Also, each bit in the multiplicand is always allocated to the same position in each row, and all the bits in a given row are multiplied by the same Booth signed digit. Note that, depending on the word length, some of the bits in each row will be masked out.

As shown in Figure 1(b), PPs have also a small prefix which replaces full sign extension. As those prefix bits are located at different places depending on the word length, multiplexers are needed to select between prefix, or regular PP bits. In Figure 2, individual bits are represented by hollow dots. The solid ones are those that require a multiplexer, as they may be either regular, prefix, or hot one bits depending on the selected word length. As can be seen, multiplexed bits account for less than 20% of the total.

The last row consists of the hot one bits plus the PPs that enable unsigned multiplication. The Booth signed digit for this last row is 1 when the multiplication is unsigned, and the most significant bit of the second operand is 1. In all other cases the PP section of the row will consist of 0s. Note that this last row is the only one that may contain data from more than one subword.

The figure highlights the bits involved in computing 8×8, 16×16, and 32×32 products. The 33 rows must be added together using a CSA reduction tree plus a non-redundant adder. In a 64×64 multiplication all the bits in the figure are enabled. For a 32×32 product, almost one half are disabled. This proportion increases to almost seven out of eight for 8×8 operations. The reduction tree considers all the bits, but some of the full- and half-adders in it are capable of killing carry propagation. A selection signal is generated depending on the position of the adder in the tree and the selected word length. The final non-redundant adder must also implement carry-kill at selected places.

B. APPROXIMATE MULTIPLIERS

A large number of works have studied the design of approximate multipliers, covering different strategies with differing degrees of accuracy and energy efficiency. The most common approach to approximate multipliers is *truncation*, which consists in removing selected elements from the dot matrix. Some approaches further introduce a small number of *correction* terms, which seek to reduce the total error introduced by truncation. In this section we carry out a brief review of the related literature, highlighting the most promising strategies for the design of approximate SIMD multipliers.

Petra *et al.* [18] designed an unsigned multiplier with a signed extension. It is not based on the modified Booth algorithm, producing a large number of PPs, making this implementation less attractive than others. Reference [19] propose a modular approach for addition and multiplication. Large multipliers are built by combining smaller blocks, and truncation is implemented by removing some blocks. The proposed configurations, however, provide small power savings.

Albicocco *et al.* [2] leverage sleep transistor insertion to select between accurate and approximate modes by using power gating. The error introduced by truncation is not compensated in any way, producing an average error larger than other works. The same authors propose, in a different work, the use of *sloppy* operands [1]. The multiplier is recoded using the modified Booth algorithm, but recoding is approximate for the less significant columns. The introduced error is larger than that of a truncated multiplier.

Zervakis *et al.* [27] introduce partial product perforation, eliminating selected *P*Ps and achieving shallower adder trees. The approach is interesting, but difficult to extend to SIMD multipliers, as the number of perforated *P*Ps depends on the

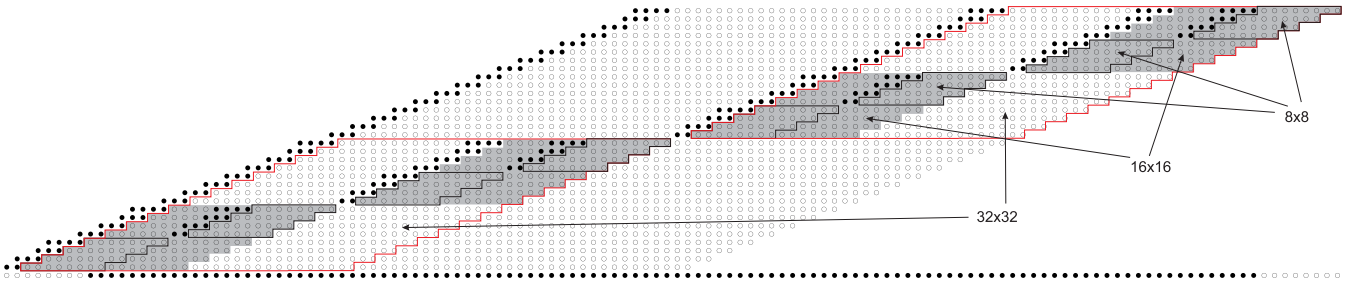

FIGURE 2. Dot matrix of the partial products for the SIMD multiplier architecture proposed by [7]. Each of the first 32 rows contains a different PP multiplied by the appropriate Booth signed digit. Different word lengths are enabled by masking out certain bits. E.g., the multiplier can be configured to compute eight 8 × 8, four 16 × 16, or two 32 × 32 multiplications by enabling only the highlighted sections of the dot matrix. Hollow dots represent data bits. Solid dots need to be multiplexed, as they contain data, sign extension prefixes, or hot ones, depending on the word length.

selected word length. Therefore, levels that could be perforated for 64 × 64 products, may be necessary for 8 × 8 ones.

Swartzlander [21] proposes a variable correction factor to reduce the error introduced by truncation. This idea is developed in a number of papers that represent the state of the art in the field. The PPs are divided into 2 parts: *major* and *minor* [5]. The bits in the minor part are not included in the reduction of the partial products, but some of them may be used to calculate a correction factor that is added to the major part.

Jiang *et al.* [17] perform a review over a large number of designs. The work by the same authors in [16] presents a more focused and meaningful review of a smaller set of multiplier designs. We highlight here three of them. In [15], a combination of truncation and approximate accumulation of the PPs is proposed. Reference [4] present a method to compute the correction factor for a given number of truncated bits. Reference [25] truncate products to $(n/2 + 1)$ bits, and introduce an algorithm to compute a correction factor up to 16 × 16 products. We simulated these approaches and found the latter to produce the lowest average error. For this reason, the design by Wang et al. is used as the basis of our SIMD approximate multiplier. The truncation and correction approach employed is further described in the next paragraphs. Interestingly, this work was discarded in the final evaluation by [17], for unclear reasons.

An $n \times n$ multiplier can be truncated to the $(n + 1)$ most significant bits incurring a maximum error of $\pm 2^8$ and $\pm 2^{17}$ units for 8 × 8 and 16 × 16 products, respectively [25]. In the design by Wang et al., the error is kept low by adding 2 correction factors, λ and α, starting at position $(n-1)$. This approach is detailed in Figure 3. The value of λ is obtained by using a simple logical function that involves the value of one of the Booth signed digits, a_0, and b_{n-1}, where a and b are the multiplicand and the multiplier, respectively. For more specific details, we refer the reader to the original paper [25]. The value of α is obtained as:

$$\alpha = \lfloor (m-1)/2 \rfloor \tag{1}$$

where m is the number of non-zero Booth signed digits for a particular product. Reference [25] propose to use a sorting network to compute m. This is more efficient than adding up the number of non-zero Booth digits. Figure 4 gives more details about the usage of sorting networks for the calculation of α. As can be seen in the figure, the complexity of the network to compute α increases rapidly with the word length, so that applying this scheme to 32 × 32 or 64 × 64 operations would be infeasible. For this reason, a different approach to computing α for large multiplications is proposed in Section III-B.

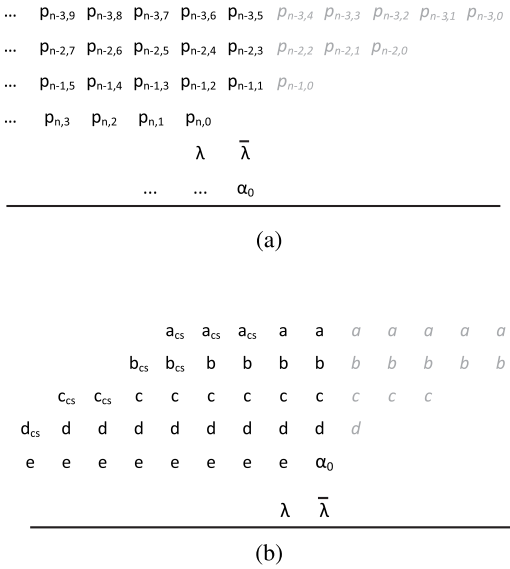

FIGURE 3. Truncated multipliers proposed by [25]: (a) generic version; and (b) 8 × 8 version. Grayed out bits are truncated (removed from the computation). λ and α are correction factors introduced to reduce the error incurred.

III. CHALLENGES OF SIMD TRUNCATION

Applying truncation to SIMD multipliers creates new challenges that need to be addressed to produce a viable design. In this section we deal with two sources of error which will require specific support in our architecture: the potentially large overflow in unsigned multiplications, and the computation of α for large multipliers without resorting to large sorting or addition networks. Then, we accurately characterize

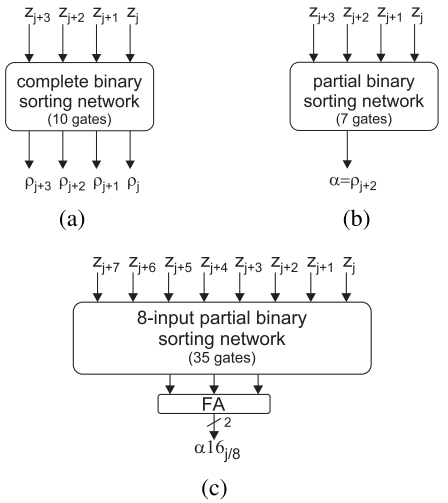

FIGURE 4. Schematics of the sorting networks proposed by [25] to calculate α. Subfigure (a) uses a full network (10 gates) to compute α for the 8 × 8 product. The value of α is, at most, 1 when the number of non-zero Booth digits is 3 or 4, or 0 in any other case. E.g., 0110 is sorted as 0011; and 1101 as 0111. The only interesting bit then becomes $\rho_{j+2} = \alpha$. Subfigure (b) shows a partial sorting network (7 gates) to compute α. Subfigure (c) exemplifies the use of a partial sorting network (35 gates) to compute $\alpha_1\alpha_0$ in the 16 × 16 multiplier. In this case, the maximum value of α is 3, and is obtained when there are 7 or 8 non-zero Booth digits.

truncation errors for small multipliers and propose a simulation strategy for extending these results for larger units.

A. OVERFLOW IN UNSIGNED MULTIPLICATIONS

Errors in truncated multipliers may switch the sign of the result. This happens when a positive operand, whose value is close to zero, is multiplied by a negative Booth digit. In this situation, the most significant bits in the resulting PP will be ones. In the accurate implementation, this is not a problem, as the hot one bits will switch them to zero. However, hot ones are truncated in the approximate design, and so spurious ones will remain. When dealing with signed products, this particular overflow is not catastrophic, as it only happens with values close to zero. Thus, the distance between the accurate and the obtained result is small.

A serious problem arises in the same situation with unsigned multiplications, as the inaccurate result represents a large unsigned number, instead of a small negative one. To the best of our knowledge, this issue has not been addressed in any related work. We propose the following approach: for each Booth digit, a number of the most significant bits of the multiplicand are checked. If those bits are zero, that Booth digit is turned to zero. The number of bits to check depends on the index of the Booth digit. As can be seen in Figure 3(b), each truncated PP makes use of a number of bits of the multiplicand. As such, we modulate the value of each of the Booth digits by inspecting only the bits in the multiplicand that can be affected by that digit. As an example, the least significant Booth digit will be forced to zero if the most significant bit of the multiplicand is also zero; and the next digit will be forced to zero if the three MSBs are zero. The implementation of this correction will be explained in more detail in Section IV and Figure 9.

Whereas a more complex analysis of the values of both operands may narrow the number of troublesome cases, checking the value of the multiplicand provides the simplest solution. In Figure 5, a small example computing the 8-bit product $01h \times 2Ah$ is shown. The multiplier value $2Ah$ is Booth-encoded as $01\overline{11}2$. In Figure 5(a), the accurate operation is shown, using 5 *PPs* with 2 *hot ones* each. In Figure 5(b), the 5 partial products are truncated so that the seven LSBs are discarded. The values of α and λ are 1 and 0, respectively, and $\overline{\lambda}$'s value is 1, marked in bold. The result is $FF80h$ instead of the expected $002Ah$. In the case of a signed product, the difference would be small ($42d$ vs $-128d$). In the unsigned case, however, the difference is unacceptably large ($42d$ vs $65\,408d$). In Figure 5(c), our proposed correction is applied. It sets all Booth digits to zero. The new result is close to the accurate one for both signed and unsigned products.

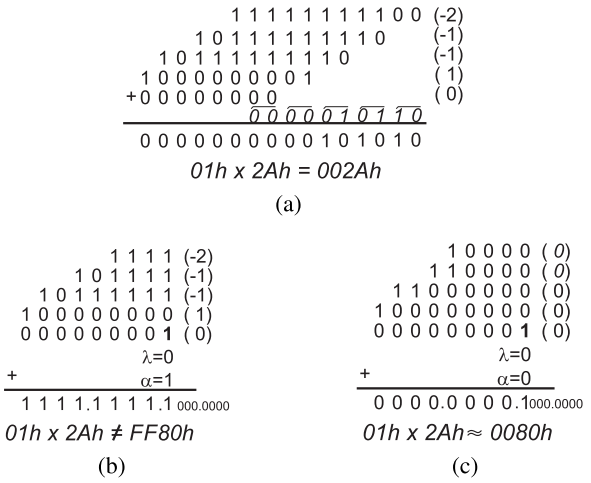

FIGURE 5. Example of overflow in unsigned products. By truncating the partial products, small negative values appear as large positive ones. Hence, the resulting error in (b) is too large. By correcting Booth's digits, the resulting error is within the limits (c).

B. COMPUTING α

As explained in Section II-B, a new scheme is needed to compute the correction factor α for large multipliers. In our planned SIMD implementation, we need to obtain either eight $\alpha 8$ (1 bit each), four $\alpha 16$ (2 bits each), two $\alpha 32$ (3 bits each), or one $\alpha 64$ (4 bits). Note that each α is given by Equation 1, which is nonlinear. As such, a value of $\alpha 16$, should not be obtained by adding two $\alpha 8$, as the maximum error would be increased by 33%. For this reason, we choose to compute $\alpha 16$ in a precise way. Instead of using partial sorting networks as seen in Figures 4(b) and 4(c), we choose instead to use full sorting networks for $\alpha 8$, as shown in Figure 4(a), and provide their outputs as inputs to a specially designed boolean network which takes as input two ordered sequences and outputs $\alpha 16$. This strategy is less expensive than the one proposed

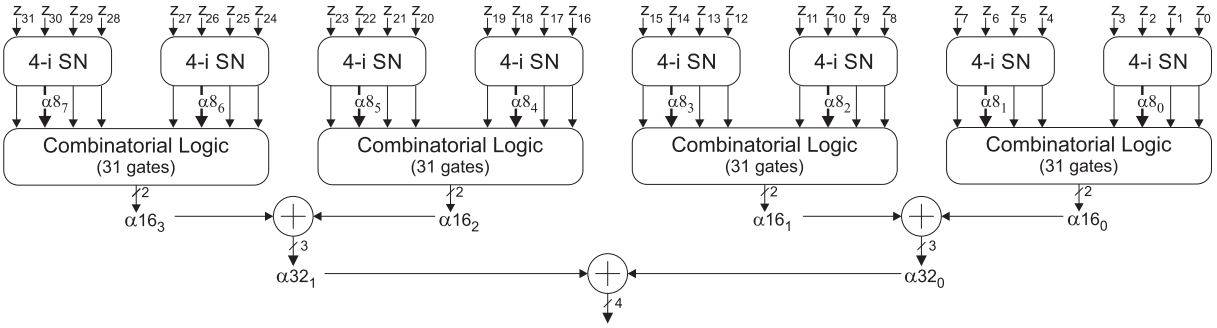

FIGURE 6. Circuit for computing the α values. At the top level there are eight sorting networks of four elements each to compute the $\alpha 8$ values (Figure 4(a)). The $\alpha 16$ values are obtained by using specially designed boolean networks which take the $\alpha 8$ values as input. The $\alpha 32$ values are approximated by adding together $\alpha 16$ values, and $\alpha 64$ is approximated by adding together the $\alpha 32$ values. Note that, for $n \times n$ multiplications, only the αn values are necessary.

by [25]. Computing $\alpha 32$, and $\alpha 64$ precisely would be slow and expensive. We propose to approximate $\alpha 32$ by the sum of two $\alpha 16$, and $\alpha 64$ as the sum of two $\alpha 32$. The maximum error is 14% and 20% respectively, which is acceptable compared to the maximum errors for these multipliers, as will be shown in Section III-C. The resulting circuits are shown in Figure 6.

C. SIMULATION

Several metrics have been proposed in the literature in order to evaluate the error introduced by approximation. Reference [17] use the maximum and average absolute errors. In the same paper, using the percentage of non-accurate products is also proposed. We found this latter metric of little interest, as error rates are always close to 100%. As such, we focus on the former ones.

Evaluating the error for 8×8 and 16×16 truncated multipliers only requires minutes of CPU time. We have extensively checked all the combinations for those cases, including signed and unsigned products. However, it is not possible to exhaustively check the 2^{64} combinations for a 32×32 multiplier, or the 2^{128} combinations for a 64×64 one.

In order to overcome this challenge we first focused on the 32×32 multiplier. We initially tried several thousands of millions of random operands, but the maximum error was significantly lower than expected. Next, we selected random values for one of the operands, and tried all the values for the second one. As the maximum error was still low, we analyzed the worst cases in 8×8 and 16×16 multipliers, trying to extract a pattern for the problematic inputs. We discovered that the worst cases are those in which the five least significant bits of the multiplicand are zero in the 8×8 case, and when the 10 LSBs are zero for 16×16 multiplications. As such, we focused on simulating operations with multiplicands having 15 or more of their LSBs equal to zero. The obtained maximum errors were in line with our estimations. Next, we took the values of the multiplier that produced the larger errors in the 8×8 and 16×16 cases and used them with all possible combinations of the multiplicand.

TABLE 1. Error characterization of truncated multipliers. Values have been divided by 2^n for normalization purposes.

	max. positive error	max. negative error	average abs. error
8×8	1.000	0.668	0.297
16×16	2.000	1.667	0.357
32×32	4.500	3.478	0.460
64×64	9.000	8.000	1.816

Table 1 shows the maximum and average error values obtained by applying the techniques presented in this section. Values are normalized by dividing by 2^n, which allows a more intuitive assessment of the error, considering that in some fields, such as multimedia, the result of $n \times n$ products are often right-shifted by n bits. After shifting, the maximum error for 8×8 and 16×16 products are, respectively, 1 and 2 units. However, the average error is significantly lower. The results obtained for 8×8 and 16×16 products conform with those reported by [25]. For the 32×32 and 64×64 operations the error seems to grow at a slightly larger rate. However, it must be taken into account that these figures were obtained not by exhaustive simulation, as in the smaller word lengths, but by testing suspected worst cases. Note that, although the normalized error basically doubles with the size of the multiplier, it becomes less significant, as the maximum size of the result grows at an even larger pace.

IV. ARCHITECTURE OF THE TRUNCATED SIMD MULTIPLIER

The proposed architecture is a combination of truncated multipliers of four different sizes. Truncation is implemented by removing bits in the PPs which are not used in any product configuration, as shown in Figure 7. In total, 24% less bits are featured in the CSA network. Additionally, bits which are unused for a given word length n are permanently set to zero during $n \times n$ operations, so that they do not draw current, dissipating static power only. For example, when operating the multiplier in 8×8 mode, 82% of the bits are fixed to zero. Area savings in the multiplier design are due to truncation.

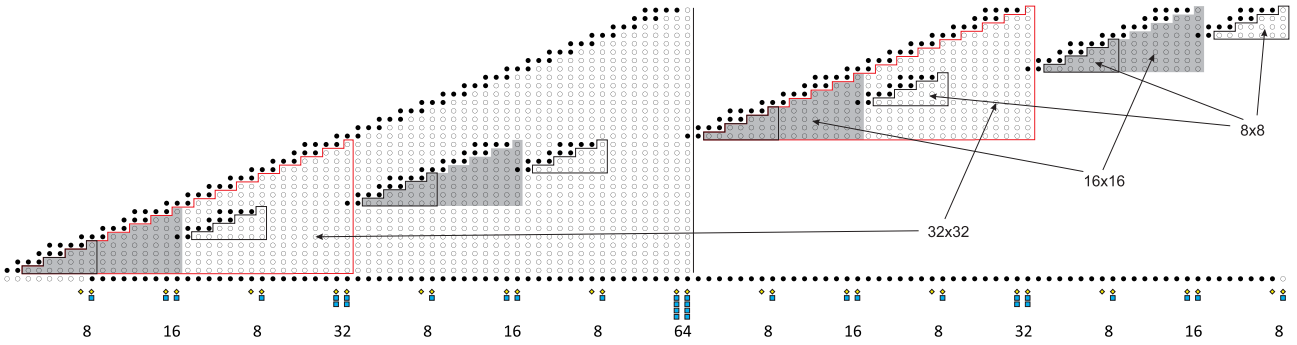

FIGURE 7. Dot matrix of the partial products for the truncated SIMD architecture. The floorplan is similar to that in Figure 2, but some bits have been truncated, and only bits 7 to 127 are computed. Correction bits are calculated as detailed in Section II-B and III-B, and added at the bottom PP (λ bits are depicted as diamonds, and α bits as squares). Hot ones are truncated.

Energy savings are larger than area savings, as the former are due to a combination of truncation and bit selection. These numbers include the α and λ correction factors, which are shown at the bottom of Figure 7 as shaded dots. The 33^{rd} PP is required to implement unsigned multiplication but, contrarily to accurate SIMD multipliers [7], does not include hot one bits, which are truncated as well.

Note that the left half of Figure 7 can be easily adapted to perform one accurate 32×32 product, two 16×16, or four 8×8 ones. The added cost would be reduced to extend a small number of multiplexers. Having an easily reconfigurable accurate/approximate unit allows to efficiently execute full sections of code, including approximate data computation and precise indexing without incurring unit duplication nor expensive switching delays.

Starting from the right to the left, Figure 7 details the footprints of 8×8, 16×16, 32×32, and 64×64 multipliers, highlighted in the same way as in Figure 2. As will be shown in Section V-A, area savings are significant, but smaller than in fixed-width multipliers. This is evidently a result of supporting 64×64 multiplication, as almost half of the bits in the circuit are not used by any other word length. As such, we considered to forgo support for 64×64 products in order to reduce area. In that case, the footprint will consist of two copies of the right half of the figure. Section V includes an evaluation of the area and energy savings provided by such an architecture, which turn out to be only slightly larger than the full 64-bit SIMD unit to which the remainder of this section is devoted.

Multiplying $X \times Y$ starts by obtaining the Booth digits b_i to compute each partial product $PP_i = X \times b_i$. Each Booth digit is encoded using 3 bits (*neg*, *two* and *one*), as shown in Figure 8. Note how supporting different word lengths in a SIMD multiplier requires to insert a "virtual" zero bit between sub-word boundaries. As such, multiplexers are used for selecting the appropriate value when computing b_k such that k is a multiple of 4.

In order to reduce the error in unsigned multiplications, Booth digits are subject to the correction detailed in Section III-A. Detecting which digits should be set to zero

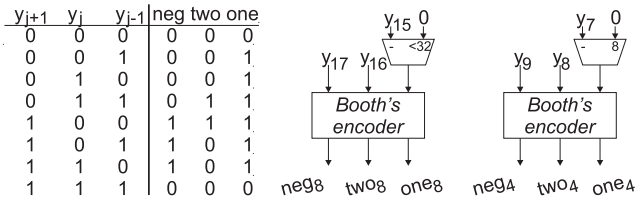

FIGURE 8. Effect of sub-word boundaries in the calculation of Booth digits. The bits of the i^{th} digit, b_i, are obtained from the multiplier bits ($y_{2i+1}y_{2i}y_{2i-1}$), or from ($y_{2i+1}y_{2i}$0) if bit 2i is the start of a new word for the selected word length. E.g., b_4 is computed from the values of ($y_9y_8y_7$) if the word length is greater than 8 bits, and ($y_9y_8$0) otherwise; b_8 is computed from ($y_{17}y_{16}y_{15}$) if the word length is greater than 16 bits, or from ($y_{17}y_{16}$0) otherwise.

requires a tree of *OR* gates that must consider sub-word boundaries for all the word lengths. Figure 9 details the correction circuit. However, the correspondence between the control bits and Booth digits depends on the word length. Therefore, the control bits are sorted in 4 different ways, and the multiplexer at the bottom of Figure 9 selects the 32 bits that will enable or disable the 32 Booth digits. Overall, this circuit takes significant area, as it will be analyzed in Section V, and it possibly introduces considerable delay.

After calculating the corrected Booth digits, partial products are computed by multiplying the sub-words in the multiplicand X by the corresponding b_i. However, as explained in Section II, each PP has a prefix that substitutes for sign extension. This aspect is common to both accurate and truncated architectures. Each prefix consists of 2, 3, or 4 bits, depending on the index of the PP. The prefix bits are computed using one Booth digit plus the most significant bit (MSB) and the sign of the corresponding sub-word, as detailed in Figure 10. Note that the prefix bits are computed in the same way as in a precise multiplier, as they are intended only to avoid full sign extension. However, the roles each bit plays in this computation change depending on the selected word length. Figure 11 details how to select the MSB for each PP.

When constructing the partial products in the SIMD multiplier each bit may be either a prefix bit, a normal product bit, or a masked out zero. Figure 12 shows the way in which the j^{th}

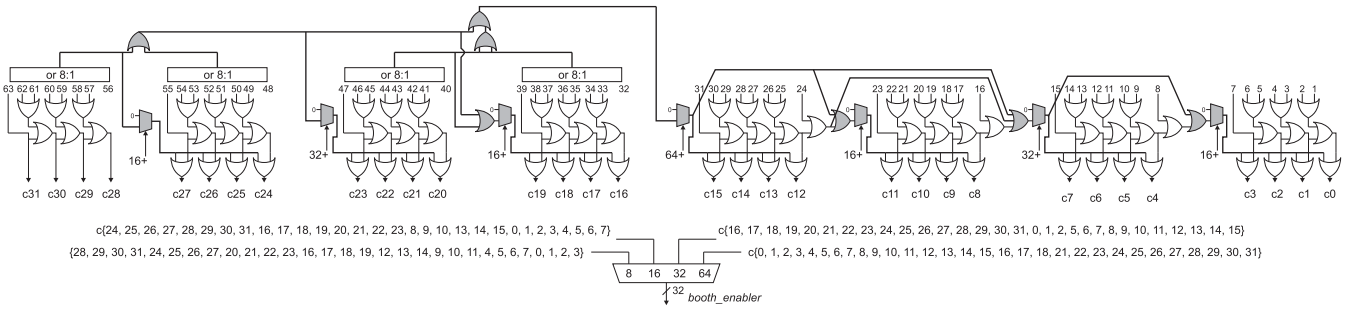

FIGURE 9. Correction of Booth digits for truncated unsigned multiplication. At the top of the figure, 63 bits of the multiplicand enter the circuit, which are used to compute 32 control bits, ($c0$ to $c31$). Those are basically computed as a chain of *OR* gates, with two modifications. First, the delay is reduced by *OR*-ing the most significant bits using a tree structure. Second, at the sub-word boundaries, multiplexers (actually implemented as *AND* gates) select whether the values for upper bits must propagate or not.

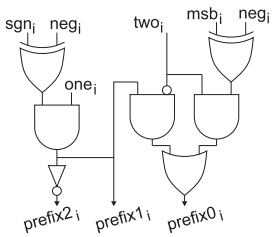

FIGURE 10. Prefix calculation for each partial product. The first PP of each product has the following 4 bit prefix: {$prefix2_i$, $prefix1_i$, $prefix1_i$, $prefix0_i$}; the last PP uses: {$prefix2_i$, $prefix0_i$}; and the ones in between use: {1, $prefix2_i$, $prefix0_i$}.

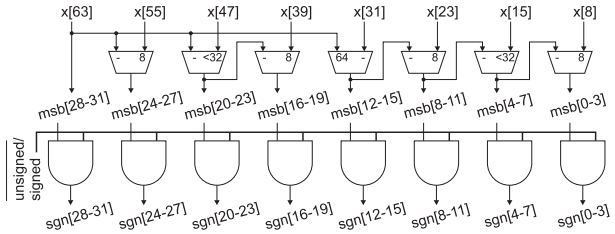

FIGURE 11. Selection of the MSB and sign bits for each PP. The multiplexers select different bits depending on the word length. E.g., in 64 × 64 mode, bit x_{63} is the MSB for all PPs; in 32 × 32, x_{63} and x_{31} are selected from each of the two sub-words. Sign bits are obtained from the MSB by masking them to zero if the product is unsigned.

bit of the i^{th} partial product ($PP_{i,j}$) is computed. First, it may be multiplied by two by selecting x_j or x_{j-1} using two_i. Next, the selected bit will be complemented if b_i is negative. Then, some bits at the PPs are prefix bits for a given word length, and regular bits for the other ones. This is implemented at the bottom multiplexer in the figure. Only some bits (solid dots in Figure 7) require that last multiplexer. Most of them are visible in the left side of the figure. Finally, the resulting bit will be set to zero under two circumstances: either because its Booth digit is zero, or because it is masked out for the current word length.

Depending on the word length, the values of α and λ are computed in different ways, and they are located at different positions in the bottommost PP. However, they never overlap, that is, no bit of αi is located in the same position of a bit of

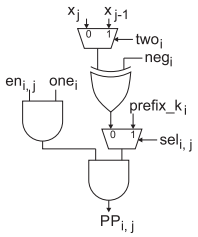

FIGURE 12. Value selection for each partial product bit. Depending on its position in the dot matrix, it might be a data bit (multiplied by the appropriate Booth digit) or a prefix bit. Besides, it might be enabled or disabled depending on the selected word length.

αj as long as $i \neq j$. The same happens with λ. As such, those values are always computed in the same way, and they are enabled or disabled, depending on the current word length, through an *AND* gate. This can be seen at the bottom of Figure 7, where α and λ bits are represented using square and diamond shapes, respectively. Parameter λ plus 1 is expressed as $\lambda\bar{\lambda}$, which explains why each occurrence is 2-bit wide. For 8×8 and 16×16 products, α is 1- and 2-bit wide, respectively. For the other modes, α is computed by adding one or more 2-bit numbers.

Finally, all the bits in the PPs are reduced to two terms by means of a tree of carry-save adders. In this implementation, 3:2 compressors are used, organized as a Dadda tree [6]. Using 4:2, or even larger, compressors could provide some area and energy savings. However, the 3:2 option was chosen for two reasons: 1) adhering to the original Dadda's algorithm was preferred; and 2) the best implementations of 4:2 and larger compressors make use of transmission gates, that are not standard logic components. Note that the impact of this choice is minimal, since the same type of compressor will be used in accurate and truncated architectures during the experimental evaluation, providing a fair comparison. In Figure 13, the implementation of the rightmost 8 × 8 and 16 × 16 truncated products (bits 7 to 31) is shown.

V. ARCHITECTURE EVALUATION

In this section, truncated architectures are evaluated against their accurate counterparts in terms of area (Section V-A)

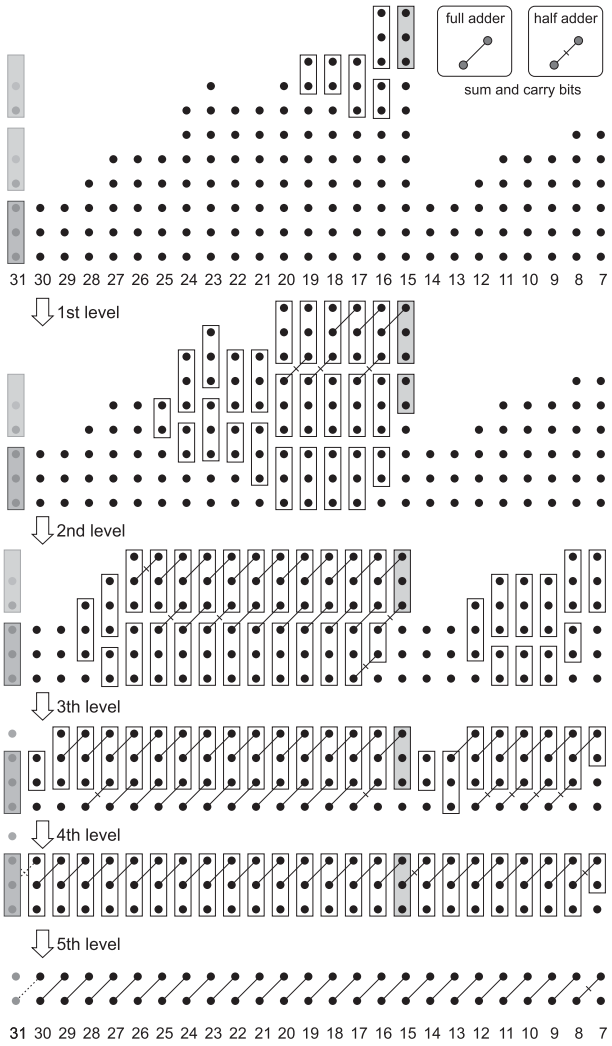

FIGURE 13. Dadda's adder tree for the right-most 8 × 8 and 16 × 16 truncated products. In this limited example, operand bits are reduced by five levels of adders. The operands of full and half adders are enclosed by rectangles, and the sum and carry bits produced by the same adder are shown connected by a line. Two 8 × 8 products are computed using columns 7 to 15 and 23 to 31. One 16 × 16 product is implemented using columns 15 to 31. Column 31 could be used as part of a 32 × 32 product, but it is not fully depicted due to its size. Different sub-word lengths are supported by setting some incoming bits to zero, as already explained. Additionally, carry propagation between sub-words must be prevented. Hence, shaded rectangles represent adders that implement conditional *carry–kill* at the boundary of the 8 × 8 product. This means that, whereas they always produce sum bits, the carry bit can be forced to zero.

and energy consumption (Section V-B). All word lengths, plus SIMD, are explored for signed and signed/unsigned modes. The signed/unsigned architectures include a mode selection input bit that specifies whether the operands must be considered as signed or unsigned numbers. Furthermore, 32-bit-only versions of the SIMD architectures, as discussed in Section IV, have been simulated, and the results presented for analysis. The energy cost of multiplication instructions is analyzed in Section V-C. In order to estimate energy savings in a real application, Section V-D analyzes the potential savings for JPEG compression. Finally, Section V-E discusses the results in depth.

A. AREA COST

Area has been estimated using two different metrics. First, it has been assessed in terms of the number of adders in the Dadda tree. Table 2 shows the results for multipliers able to implement both signed and unsigned products. Table 3 shows similar results for multipliers supporting signed multiplication only. While admittedly the CSA tree represents only part of the total area, some interesting conclusions can be derived from these results. As can be seen in both tables, the benefits of truncation increase with word length. SIMD architectures, however, must implement all word lengths, and therefore the achieved area reduction is comparable to that of the smaller word length supported, 8 × 8. Furthermore, supporting unsigned products has a significant cost, as an additional PP must be added. The difference is more pronounced in small multiplications, but extends also to the SIMD architecture. Table 4 shows the increase in the number of adders when unsigned products are supported, which is higher in truncated multipliers.

TABLE 2. Number of half adders (HAs) and full adders (FA) in the Dadda reduction tree for different words lengths of signed/unsigned multipliers. The area cost of implementing half adders (HA), full adders (FA), and carry-kill FAs is different. As such, the total number of adders is normalized to the total necessary area in terms of FAs in the *Total* column. The number of carry-kill FAs is always small (below 30 for 64 × 64 units) and omitted. The rightmost column shows area savings in terms of the total number of adders.

	Accurate			Truncated			
	HA	FA	Total	HA	FA	Total	Total Savings
8 × 8	5	27	29	7	20	23	21%
16 × 16	13	119	124	12	74	78	37%
32 × 32	29	495	505	23	276	284	44%
64 × 64	61	2015	2037	43	1063	1078	47%
SIMD 64	61	2018	2058	75	1482	1523	26%
SIMD 32	29	497	512	39	331	349	32%

TABLE 3. Number of adders in the Dadda tree of exclusively signed multipliers. The columns share the same meaning as those in Table 2.

	Accurate			Truncated			
	HA	FA	Total	HA	FA	Total	Total Savings
8 × 8	10	15	19	5	11	13	32%
16 × 16	22	91	99	10	53	57	42%
32 × 32	46	435	451	21	230	238	47%
64 × 64	94	1891	1925	41	970	985	49%
SIMD 64	61	1981	2021	73	1308	1348	33%
SIMD 32	29	476	491	35	309	325	34%

TABLE 4. Adders overhead for supporting unsigned multiplications.

Word length	8	16	32	64	SIMD 64	SIMD 32
Accurate	34%	20%	11%	5%	2%	4%
Truncated	43%	27%	16%	9%	11%	7%

The second area metric we present is the number of transistors. All the analyzed architectures have been implemented using Verilog and synthesized using Yosys Open Synthesys Suite [26]. Whereas commercial synthesis tools are more effective when optimizing a given implementation, Yosys provides a clean output in terms of basic logic gates that can be easily analyzed. The estimated transistor count for each architecture is presented in Tables 5, 6, and 7.

TABLE 5. Transistor count for signed and unsigned multipliers.

	Accurate	Truncated	Total Savings
8×8	3224	2402	25%
16×16	10536	6692	36%
32×32	37194	21494	42%
64×64	140746	72850	48%
SIMD 64	154700	111896	27%
SIMD 32	42392	29184	31%

TABLE 6. Transistor count for exclusively signed multipliers.

	Accurate	Truncated	Total Savings
8×8	2506	1618	35%
16×16	9214	5432	41%
32×32	33644	19612	41%
64×64	134074	71116	46%
SIMD 64	148520	103596	30%
SIMD 32	40322	27410	32%

TABLE 7. Transistor overhead for supporting unsigned multiplications (%).

	Word length					
	8	16	32	64	SIMD 64	SIMD 32
Accurate	22	13	10	5	4	5
Truncated	33	19	9	2	7	6

Comparing the area results obtained with both metrics, we see that they are consistent, except for small multipliers. In those cases, the transistor count reduction is slightly larger than the one observed for adders, as the CSA tree represents a smaller proportion of the total in smaller circuits. More importantly, it becomes clear that, for 8×8 and 16×16 multipliers, supporting both signed and unsigned multiplications implies a large area overhead.

Area savings for truncated SIMD architectures are significant, but modest compared to fixed-width multipliers. The reason for this becomes obvious by observing Figures 2 and 7, as the number of bits to calculate and sum up is only slightly lower than those in accurate architectures. Even if only some of those bits are used at any given moment, the reduction tree has to be designed to support all the word lengths. The convenience of dropping 64×64 support must be carefully considered if larger area savings are required. Eventually, truncation must be evaluated based on energy consumption. As mentioned in Section IV, energy savings are expected to be larger than area savings, as they will be able to leverage masked out bits, which must be present for the computation of large word lengths, but may be disabled for smaller ones.

TABLE 8. Energy savings for truncated fixed-width multipliers (%).

	8×8	16×16	32×32	64×64
Only Signed	31	40	41	47
Also Unsigned	18	24	37	43

B. ENERGY DISSIPATED BY THE MULTIPLIERS

All the architectures have been simulated using large sets of random data in order to evaluate energy consumption. In total, 160 000 bytes are used for each operation mode. The switching of every gate when changing circuit inputs on the Verilog output provided by Yosis was evaluated. We measured both the total number of switches and those which draw current from source. The results were weighted by the fan-out of each gate. The figures presented in this section represent savings in dynamic energy dissipation with respect to accurate architectures, as absolute power figures cannot be obtained with our methodology.

Due to the different capabilities of each type of multiplier, different experiments have been carried out. As a preliminar step, two types of simulations were performed for each word length using (exclusively) signed fixed-width multipliers. The first simulation performed N consecutive products. The second intercalated the product of 0×0 in between every valid product, resulting in 150% more switches. The purpose of this latter simulation is to check that the accurate and truncated architectures do not behave differently depending on how many bits change from one set of operands to the next one. The difference was found to be under 3%, and therefore we can confirm that performing just the first kind of simulation is sufficient.

1) FIXED WIDTH MULTIPLIERS: SIGNED VS UNSIGNED

We performed three sets of experiments: i) products with exclusively signed multipliers; ii) products in unsigned mode; and iii) alternating signed and unsigned products. The difference when alternating modes is negligible (less than 1%), and therefore only the average is displayed in Table 8.

2) SIMD MULTIPLIERS

For the SIMD architectures, signed products were tried first. The data set was processed first in 8×8 mode, followed by the other modes and, eventually, the simulation was repeated alternating word lengths. The results are shown in Tables 9 and 10 for the signed architecture plus the signed/unsigned one working in each of those modes.

In all cases, with two exceptions, the truncated architecture outperforms the accurate one with savings ranging from 33 to 57%. However, it is interesting to notice that the 64-bit architecture that supports unsigned multiplication experiences a dramatic energy savings drop when word lengths are alternated. A thorough study on the nets that switch values more

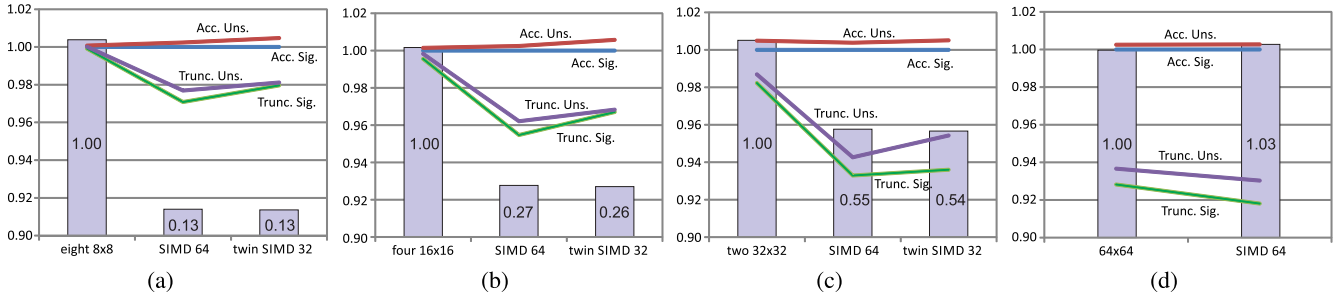

FIGURE 14. Energy dissipation including instruction processing for four word lengths and equivalent workloads. Bars represent the total energy dissipation of the accurate architectures, and are normalized to the dissipation of eight 8 × 8 multipliers. Actual values are overlaid on each bar. Lines represent the total savings in the instruction pipeline by using the proposed 64 × 64 truncated SIMD architecture. Note that the left axis is truncated to 0.9 to increase the readability of the results.

TABLE 9. Energy savings for 64-bit SIMD (%).

	8 × 8	16 × 16	32 × 32	64 × 64	Alternating
Only Signed	41	49	53	46	41 (54)
Signed	34	42	47	40	10 (42)
Unsigned	33	42	46	40	10 (41)

TABLE 10. Energy savings for 32-bit SIMD (%).

	8 × 8	16 × 16	32 × 32	Alternating
Only Signed	39	46	57	49
Signed	41	48	43	39
Unsigned	42	50	45	40

TABLE 11. Energy savings for mixed signed/unsigned modes in SIMD architectures (%).

	8 × 8	16 × 16	32 × 32	64 × 64	Alternating
SIMD 64	33	41	46	39	10 (39)
SIMD 32	43	43	44		40

often reveals that the circuit which performs Booth digits correction in Figure 9 is responsible for this behavior. A large number of nets change values in response to changes in the word length, and the effect is more noticeable in the 64-bit architecture. New simulations were conducted switching the word length every two, four, and eight operations, showing that energy savings improve rapidly. The results for alternating every four operations are shown between parentheses.

The signed/unsigned SIMD architectures were simulated switching between signed and unsigned modes after every operation. The results are shown in Table 11 and, by comparing them with the ones in Tables 9 and 10, it can be seen that switching mode has little impact.

Finally, we computed the power/delay characteristics of the truncated SIMD architectures as compared to the precise ones. The results are presented in Table 12. An average improvement of over 50% is observed. The improvement in delay characteristics could be leveraged to reduce the supply voltage of the truncated multipliers, improving energy savings quadratically.

TABLE 12. Improvement in the power-delay product of the truncated SIMD architectures with respect to the precise ones (%).

		8	16	32	64	Totals
SIMD 64	Only Signed	49	56	58	54	52
	Signed/Unsigned	44	51	51	47	48
SIMD 32	Only Signed	49	55	65	–	57
	Signed/Unsigned	52	54	52	–	54
Totals		49	54	57	50	53

C. ENERGY DISSIPATED BY MULTIPLICATION INSTRUCTIONS

The former analysis allowed us to study a variety of configurations and cases. However, it does not include the cost of fetching and decoding the instructions. Based on the energy breakdown provided by [13], an approximate estimation has been performed. Figure 14 shows the estimated energy dissipation for each architecture and word length in both exclusively signed and signed/unsigned versions. The twin 32-bit SIMD architecture consists of two 32-bit circuits executing the same instruction on different data, which is equivalent to a 64-bit SIMD circuit without support for 64 × 64 products. The data set is the same used in the previous experiments. All the architectures carry out the same workload for each word length. Plotting absolute values is not possible due to the large difference between scalar and SIMD architectures. For this reason, information is represented in two ways:

- The bars represent the energy dissipated just by the accurate exclusively signed architectures. It can be seen that the SIMD architectures consume less energy than the fixed-width ones. The difference is more noticeable for 8 × 8 products and null for 64 × 64 ones. The explanation is that the same work is performed executing a single SIMD instruction in 16-bit mode than executing four 16 × 16 individual instructions.
- The lines represent the relative difference with respect to the accurate signed architecture. Figure 14 shows that the advantage of using truncated multipliers increases with word length.

Whereas savings between 2% and 8% may appear small, they are quite significant, as they are relative to the total cost

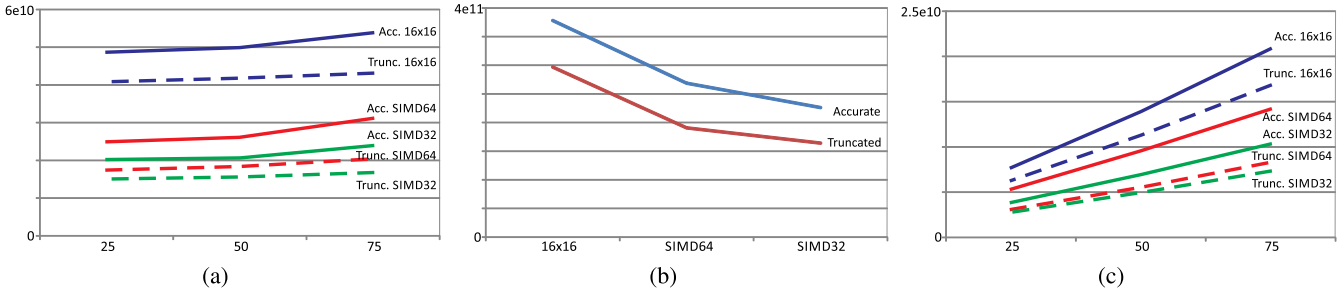

FIGURE 15. Energy dissipation in JPEG: (a) quantization, (b) forward DCT, and (c) inverse DCT. Energy dissipation figures indicate the number of gate switches that draw power from the source times the fan-out of each gate.

of fetching, decoding and executing the instructions. Two conclusions may be drawn from the plots:

- Implementing exclusively signed products provides significant savings, as detailed in Section V-B. However, the plots in Figure 14 reveal that the difference is greatly reduced when the total cost of the instruction is considered. As such, supporting also unsigned products adds an extra cost of 1% total energy, at most.
- The difference between using a full 64-bit SIMD architecture vs two 32-bit ones is also very small in most cases. Thus, supporting 64 × 64 products may not be as expensive as expected.

D. CASE STUDY: ENERGY DISSIPATION WITH JPEG

In this section, we will focus on the specific case of encoding and decoding JPEG images using the implementation by the [14]. At the encoder, two functions make extensive use of multiplication: quantization and DCT. We did not consider color conversion, as truncation errors introduce significant degradation in quality and, moreover, conversion is usually implemented using table look-up. At the decoder, only the inverse DCT is considered, as inverse quantization works mainly on byte-sized data, and therefore using accurate 8 × 8 products is more convenient. Other tasks in JPEG encoding/decoding are significantly less demanding of computational resources.

All the multiplications are performed as 16 × 16 signed products. Figure 15 shows an estimation of energy consumption due to multiplications in the three aforementioned functions for all the architectures. Data was obtained processing the 6 Mpixel image `hdr.ppm` from the Image Compression Benchmark [20] for three levels of quality (25, 50 and 75). Processing the other images in the repository produces comparable results. Figure 15(a) shows that energy consumption is significantly lower for SIMD multipliers than for scalar ones. Also, truncated multipliers clearly outperform their accurate counterparts. The results in Figure 15(b) are independent of the level of quality, as forward DCT always works on the original pixels of the image. Again, truncation allows up to 25% savings, and SIMD architectures beat the scalar ones. Figure 15(c) shows the large influence of quality settings. This is due to the fact that, at the decoder, whole arrays of data may skip inverse DCT if their content is zero

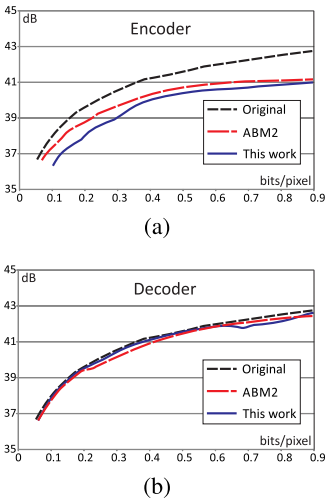

FIGURE 16. PSNR vs bit-rate for `hdr.ppm`. Quality settings range from 25 to 90. Three series of data are depicted using different 16 × 16 multipliers: the original accurate one, the truncated one from this work, and the ABM2-R15 from [15].

(which often happens for low quality levels). The results are, in any case, consistent with the previous ones.

Overall, these results are coherent with those in Figure 14(b), but show that truncated multipliers perform better on a real application such as JPEG compression that on synthetic data. The twin 32-bit SIMD is, on average, 15% more efficient than the full 64-bit SIMD.

Lossy image compression can be evaluated through two metrics: image quality and file size. The balance between both metrics can be analyzed using a PSNR vs bit-rate plot (Peak Signal-to-Noise Ratio, in dB vs bits per pixel), as in Figure 16. The results for three different 16 × 16 signed multipliers are shown in the figure: the original accurate one; our truncated multiplier taken from [25]; and the best implementation from [15], called ABM2-R15. The latter one implements radix-8 Booth's algorithm with approximated ±3 partial products and truncated final addition.

In Figure 16(a), the results are shown for the encoder. Approximate multipliers have been used for computing DCT and quantization, producing a perceptible degradation only at high quality levels (above 0.5 bps). In Figure 16(b), the results for the decoder are shown. In this case, the image has been compressed using the accurate multiplier, and then decoded using different multipliers.

Both approximate multipliers offer results close to the accurate case. At the encoder, ABM2 performs slightly better than ours, in line with the results presented in [15]. At the decoder, our work introduces less degradation, but the differences with the accurate multiplier are minimal. This is specially important because images are usually encoded just once, but decoded many times. In this sense, obtaining good results at the decoder is more significant than at the encoder. By analyzing Figure 16(a) it is noticed that, for a desired PSNR, more data are produced when using approximate multipliers. Alternatively, lower PSNR is achieved for a target file size. As it is expected that storing and/or transmitting those extra bits increases energy consumption, it is not clear whether approximate multipliers produce net energy savings for all the range of application of JPEG encoding. Decoding, however, looks more promising. For the sake of completeness, we have also tried encoding with a straightforward truncated multiplier without any kind of correction. We observed a significant quality degradation, equivalent to a 12% bit-rate increase, which clearly justifies using an advanced truncation algorithm.

Finally, we have studied the total energy savings attained by the proposed approximate units for the full JPEG encoder. The share of integer multiplications which can be performed in an approximate way, i.e., those performed during quantization and DCT, is 99.9%, and so the savings observed in Figure 15 translate directly to the execution stage. The total energy saved depends on how instruction energy is distributed across the pipeline stages in each particular architecture, depending on the complexity of the control stages of the pipeline, issue width, and degree of speculation. These results suggest that approximate units are more appropriate for specific-purpose architectures, were no complex instruction scheduling takes place. Furthermore, we observe that integer multiplication instructions represent approximately 5% of the total instructions executed by the compressor. The proposed SIMD truncated multipliers are a first step into the design of approximate SIMD units. A wider range of units will be necessary in order to leverage the full potential of SIMD approximate units.

E. ANALYSIS

A number of conclusions can be drawn from the results presented in this section:

- Truncated architectures provide significant area and energy savings in all the tested configurations and scenarios.
- SIMD architectures dissipate less energy than fixed-width ones for the same amount of work, except for 64×64 products, in which they are equivalent.
- According to Table 7, the overhead due to supporting unsigned multiplications is large for small multipliers. This overhead may affect static power dissipation. However, the difference is diluted for large or SIMD multipliers. As such, it is not clear that a low power architecture should renounce to support both operation modes.
- Savings in area and dynamic energy dissipation are dissimilar in small multipliers, but they are correlated in large and SIMD multipliers. This is particularly true when the multipliers support unsigned operations. In light of this, achieving area reductions in small truncated multipliers may not imply comparable energy savings in all applications.
- There are differences between the energy dissipation calculated using random data in Figure 14(b), and those obtained in a real application as in Figure 15. This is due to the fact that, in real multimedia applications, data items processed in sequence have many bits in common, mainly the most significant ones. Hence, not only energy savings may vary for different applications, but also the behavior of a given architecture with respect to the others.
- Without disregarding the previous conclusion, Figure 15 shows that using a twin 32-bit SIMD unit instead of its 64-bit counterpart may provide up to 20% energy savings. Considering that 64-bit multiplications are relatively uncommon, removing support for those operations should be considered.

VI. CONCLUDING REMARKS

A number of architectures have been implemented and analyzed for both accurate and truncated multiplication. The problem of implementing unsigned truncated multiplications using Booth's algorithm has been addressed and solved. The error introduced by truncation has been characterized for all product sizes, proposing a methodology for those cases in which full simulation is not tractable. A truncated 64-bit SIMD architecture has been proposed, from which a twin 32-bit one was derived. Additionally, the most significant half of the truncated architecture has the capability of computing accurate products. Area savings have been assessed for all fixed-width and SIMD architectures, showing the benefits of truncation. Afterwards, the reduction in energy consumption has been analyzed, first considering only the multiplier, and then adding the cost of fetching and decoding the instructions. Moreover, the case of image encoding and decoding using the standard JPEG has been studied. As we have not found other SIMD approximate multipliers in the literature, some comparisons have been drawn with other scalar approximate multipliers, showing similar levels of accuracy for JPEG encoding and decoding. Overall, the results show that SIMD architectures provide significant savings in energy dissipation, that supporting both signed and unsigned products has little impact, and that supporting 64×64 products has a significant impact on energy consumption that could be avoided by resorting to a twin 32-bit SIMD architecture.

REFERENCES

[1] P. Albicocco, G. C. Cardarilli, A. Nannarelli, M. Petricca, and M. Re, "Imprecise arithmetic for low power image processing," in *Proc. 46th Asilomar Conf. Signals, Syst. Comput. (ASILOMAR)*, Pacific Grove, CA, USA, Nov. 2012, pp. 983–987.

[2] P. Albicocco, G. C. Cardarilli, A. Nannarelli, M. Petricca, and M. Re, "Truncated multipliers through power-gating for degrading precision arithmetic," in *Proc. Asilomar Conf. Signals, Syst. Comput.*, Pacific Grove, CA, USA, Nov. 2013, pp. 2172–2176.
[3] A. D. Booth, "A signed binary multiplication technique," *Quart. J. Mech. Appl. Math.*, vol. 4, no. 2, pp. 236–240, Jan. 1951.
[4] Y.-H. Chen and T.-Y. Chang, "A high-accuracy adaptive conditional-probability estimator for fixed-width Booth multipliers," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59, no. 3, pp. 594–603, Mar. 2012.
[5] K.-J. Cho, K.-C. Lee, J.-G. Chung, and K. K. Parhi, "Design of low-error fixed-width modified Booth multiplier," *IEEE Trans. Very Large ScaleIEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 5, pp. 522–531, May 2004.
[6] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, pp. 349–356, Jan. 1965.
[7] A. Danysh and D. Tan, "Architecture and implementation of a vector/SIMD multiply-accumulate unit," *IEEE Trans. Comput.*, vol. 54, no. 3, pp. 284–293, Mar. 2005.
[8] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. 38th Annu. Int. Symp. Comput. Archit. (ISCA)*, San Jose, CA, USA, Jun. 2011, pp. 365–376.
[9] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *Proc. 17th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, London, U.K., Mar. 2012, pp. 301–312.
[10] H. Fariborzi *et al.*, "Analysis and demonstration of MEM-relay power gating," in *Proc. IEEE Custom Integr. Circuits Conf.*, San Jose, CA, USA, Sep. 2010, pp. 1–4.
[11] M. J. Flynn and S. F. Oberman, *Advanced Computer Arithmetic Design*. Hoboken, NJ, USA: Wiley, 2001.
[12] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proc. 18th IEEE Eur. Test Symp. (ETS)*, Avignon, France, May 2013, pp. 1–6.
[13] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *ISSCC Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2014, pp. 10–14.
[14] *Independent JPEG Group,Release 9.C*. Accessed: Apr. 2019. [Online]. Available: http://www.ijg.org
[15] H. Jiang, J. Han, F. Qiao, and F. Lombardi, "Approximate radix-8 Booth multipliers for low-power and high-performance operation," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2638–2644, Aug. 2016.
[16] H. Jiang, C. Liu, N. Maheshwari, F. Lombardi, and J. Han, "A comparative evaluation of approximate multipliers," in *Proc. IEEE/ACM Int. Symp. Nanosc. Architectures (NANOARCH)*, Beijing, China, Jul. 2016, pp. 191–196.
[17] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han, "A review, classification, and comparative evaluation of approximate arithmetic circuits," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 4, Aug. 2017, Art. no. 60.
[18] N. Petra, D. De Caro, V. Garofalo, E. Napoli, and A. G. M. Strollo, "Truncated binary multipliers with variable correction and minimum mean square error," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 6, pp. 1312–1325, Jun. 2010.
[19] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, J. Henkel, and J. Henkel, "Architectural-space exploration of approximate multipliers," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Austin, TX, USA, Nov. 2016, pp. 1–8.
[20] *Image Compression Benchmark*. Accessed Apr. 2019. [Online]. Available: http://imagecompression.info/test_images
[21] E. E. Swartzlander, "Truncated multiplication with approximate rounding," in *Proc. 33rd Asilomar Conf. Signals, Syst., Comput.*, Pacific Grove, CA, USA, Oct. 1999, pp. 1480–1483.
[22] M. B. Taylor, "Is dark silicon useful? Harnessing the four horsemen of the coming dark silicon apocalypse," in *Proc. DAC Design Automat. Conf.*, San Francisco, CA, USA, Jun. 2012, pp. 1131–1136.
[23] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proc. 52nd ACM/EDAC/IEEE Design Automat. Conf. (DAC)*, San Francisco, CA, USA, Jun. 2015, pp. 1–6.
[24] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, no. 1, pp. 14–17, Feb. 1964.
[25] J.-P. Wang, S.-R. Kuang, and S.-C. Liang, "High-accuracy fixed-width modified booth multipliers for lossy applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 1, pp. 52–60, Jan. 2011.
[26] C. Wolf. *Yosys Open Synthesis Suite*. Accessed: Apr. 2019. [Online]. Available: http://www.clifford.at/yosys/
[27] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris, and K. Pekmestzi, "Design-efficient approximate multiplication circuits through partial product perforation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 10, pp. 3105–3117, Oct. 2016.

ROBERTO R. OSORIO received the Ph.D. degree in physics from the University of Santiago de Compostela, Spain, in 1999. In 2000, he joined IMEC v.z.w., where he contributed to MPEG-21. Since 2003, he was a Researcher with the Ramón y Cajal Program and an Associate Professor, since 2008. In 2010, he joined the University of A Coruña. His main research interests include the areas of application specific circuits, and image and video coding. His homepage is http://gac.udc.es/~roberto.

GABRIEL RODRÍGUEZ is currently an Associate Professor of computer engineering with the Universidade da Coruña, where he is also a member of the Computer Architecture Group. His main research interests include the field of optimizing compilers, architectural support for high-performance computing, and power-aware computing. His homepage is http://gac.udc.es/~gabriel.