Synthesis of Approximate Parallel Prefix Adders

Apostolos Stefanidis, Ioanna Zoumpoulidou, Dionysios Filippas, Giorgos Dimitrakopoulos, and Georgios Ch. Sirakoulis

Abstract—**Approximate computation has evolved recently as a viable alternative for maximizing energy efficiency. One aspect of approximate computing involves the design of hardware units that return a sufficiently accurate result for the examined occasion, rather than computing an accurate result. As long as the hardware units are allowed to compute approximately, they can be designed with multiple new ways. In this work, we focus on the synthesis of approximate parallel prefix adders. Instead of exploring specific architectures, as done by state-of-the-art approaches, the introduced synthesizer can produce every solution that meets the designer's criteria, resulting in adders with various delay, area and error trade-offs. This automatic design space exploration allows approaching in several cases optimal solutions that could have not been designed with any other known parallel prefix architecture. The synthesized adders, when compared with state-of-the-art, achieve 27%–36% better error frequency on average for random inputs and improve image quality metrics by 8%–42% for image filtering. These results are achieved with the proposed adders requiring the same or marginally more hardware area or energy. On the contrary, in split-accuracy configurations, more than 30% of hardware area/energy can be saved for the same classification accuracy for a neural network application.**

Index Terms—**Approximate adders, Parallel Prefix Adders, Synthesis, Energy efficiency**

I. Introduction

Approximate computing explores a fundamental energy-quality tradeoff by allowing hardware components to compute possibly inaccurate results with the goal of saving as much energy as possible [1]. The design of approximate hardware units is not straightforward and opens up a whole new space of design choices [2]. For instance, a hardware component can become inaccurate by removing selected transistors from specific cells [3]. Also, approximation can be achieved by removing certain gates or flip-flops from a design's netlist [4].

In this work, we focus on the design of approximate adders. An adder is the fundamental component in every datapath and its performance in terms of accuracy-vs-energy efficiency is expected to be crucial for the overall quality of the approximate hardware units.

Out of a wide range of adder organizations, parallel prefix adders have been proven to be both efficient and versatile and can meet multiple design criteria and constraints. Parallel prefix structures can model multiple adder forms spanning from ripple carry adders, carry increment adders, and fast adders of logarithmic logic depth as well as everything in between [5]. Therefore, the parallel prefix formulation of addition can form the base for the design of multiple forms of approximate adders. On the other hand, optimizing the performance and area of parallel prefix adders is not a trivial task and has been studied extensively in the literature [6], [7], [8], [9].

Approximate parallel prefix adders have been designed by shortening the carry chain [10], [11] or pruning lower levels of the adder [12]. In a similar vein, in [13] logic gates are removed from the cells of the parallel prefix adder for the lower order bits, following a split-accuracy configuration [14], [15]. Similar techniques have been extended to FPGA chips [16] including also error-correction circuits. The work of [17] expands approximate parallel prefix adder design to support configurable accuracy. Although such state-of-the-art approaches identify efficient approximate parallel prefix designs, they follow a specific architecture in each case, thus exploring only part of the design space.

Our goal is to *automatically synthesize all possible approximate parallel prefix adders* given a maximum allowed carry chain length, the available number of prefix levels as well as the maximum allowed internal fanout. The maximum carry chain length can be less than the bitwidth of the adder and determines the accuracy of the adder's result. The carry chain length, the maximum allowed number of prefix levels and fanout constraints determine the area/power of the design and its delay characteristics.

To enable this constrained synthesis approach, we automatically enumerate all possible parallel-prefix trees, both accurate and approximate, following an approach similar to the one presented by Roy et al. in [7] only for accurate parallel prefix adders. Approximate parallel prefix trees include arbitrary smaller carry chain lengths and possible number of prefix levels. The extra freedom in carry-chain lengths, increases significantly the number of parallel prefix adders that can be designed. To tackle this state explosion, we utilize solution pruning that allowed us to explore the design space in a reasonable runtime even for large bit widths.

This enumeration of approximate parallel prefix adders enabled the identification of new architectures that are presented for the first time in open literature and offer significant accuracy and area/power benefits when compared to state-of-the-art approximate parallel prefix adders [10], [11]. The proposed approach can also synthesize approximate parallel prefix adders of split accuracy that compare favorably to efficient split-accuracy parallel prefix architectures [13].

In overall, the contributions of this work can be summarized as follows:

- It allows for the first time –to the best of our knowledge– the enumeration of approximate parallel prefix adder topologies given a set of design constraints. Appropriate

Apostolos Stefanidis, Ioanna Zoumpoulidou, Dionysios Filippas, Giorgos Dimitrakopoulos and Georgios Ch. Sirakoulis are with the Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece, (e-mail: apstefan@ee.duth.gr, ioanzoub1@ee.duth.gr, dfilippa@ee.duth.gr, dimitrak@ee.duth.gr, gsirak@ee.duth.gr)

pruning based on design constraints enables the enumeration to complete in reasonable runtime.

- It generates many candidate prefix graph structures for a given set of constraints that can be evaluated for their performance in physical implementation and numerical accuracy. Also, the introduced synthesis engine is available in [18] under a permissive open-source license, which makes it fully analyzable and freely extensible and reusable.
- The proposed approach can synthesize multiple forms of approximate adders that compare favorably to state-of-the-art including also split-accuracy alternatives. The proposed adders have been thoroughly evaluated both for their hardware complexity and their accuracy using synthetic inputs as well as real applications. For instance, when compared with state-of-the-art approximate parallel prefix adders in image filtering applications, they improve well-known image quality metrics by 8%–42% for the same or marginally more hardware area/energy. Also, in split-accuracy configurations, more than 30% area/energy can be saved for the same classification accuracy of a neural network.

The rest of the paper is organized as follows: Section II revisits the basics of parallel prefix addition and introduces the possible approximations supported by parallel prefix adders. Section III introduces the proposed adder synthesis approach. Experimental results are given in Section IV, and related work discussing the design of non-parallel-prefix approximate adders is given in Section V. Finally, conclusions are drawn in the last Section.

II. Accurate and Approximate Parallel Prefix Addition

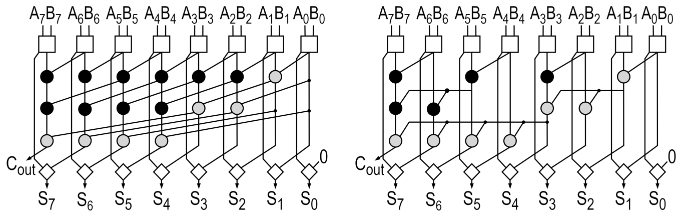
When adding two n-bit binary numbers $A = A_{n-1}A_{n-2}\ldots A_0$ and $B = B_{n-1}B_{n-2}\ldots B_0$, the sum bit S_i at the i-th bit position is computed by combining the modulo-2 sum (exclusive OR) of bits A_i and B_i, i.e., $H_i = A_i \oplus B_i$ (XOR), and the carry C_{i-1} computed in the previous bit position:

$$S_i = H_i \oplus C_{i-1}. \tag{1}$$

For computing the sum bits of the following bit positions, the incoming carry C_{i-1} needs to propagate to the next position. The carry out of the i-th bit position C_i is computed using the local carry generate and propagate bits $G_i = A_i\, B_i$ (AND) and $P_i = A_i + B_i$ (OR) and the fundamental recursive carry propagation formula

$$C_i = G_i + P_i\, C_{i-1}. \tag{2}$$

For short bit widths, ripple-carry adder structures [19] offer compact implementations. For increased bit widths, carry-lookahead adders improve the linear growth of carry chain delay [20]. In this case, carries are computed in parallel and addition is implemented in logarithmic logic depth.



A_7B_7 A_6B_6 A_5B_5 A_4B_4 A_3B_3 A_2B_2 A_1B_1 A_0B_0
C_{out}
0
S_7 S_6 S_5 S_4 S_3 S_2 S_1 S_0

(a) Kogge-Stone (b) Ladner-Fischer

Fig. 1. The (a) Kogge-Stone [21] and (b) Ladner-Fischer [22] parallel prefix carry-propagate adders.

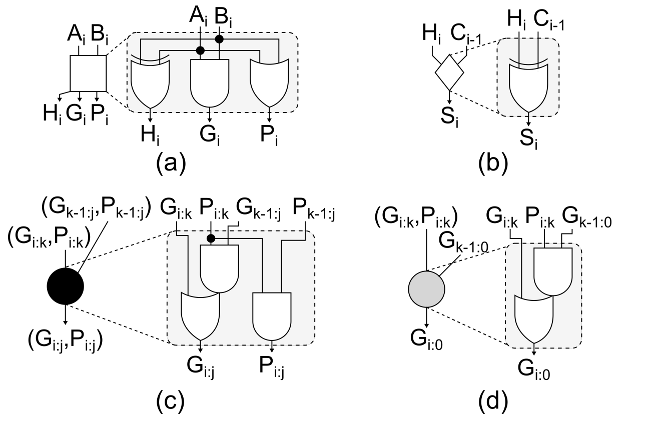


Fig. 2. The blocks used in the (a) first and (b) in the last stage of a parallel-prefix adder. (c) The logic-level implementation of the prefix operator $\circ$. (d) The simplified operator $\circ$ used in the last node of each column.

A. Accurate Parallel Prefix Adders

Mapping carry-lookahead adders to parallel-prefix structures maximize their efficiency and increase their placement and wiring regularity [5], [6], [7]. Carry computation is transformed to a prefix problem [23] by using the associative operator $\circ$ which associates pairs of generate and propagate bits as follows:

$$(G, P) \circ (G', P') = (G + P\,G', P\,P').$$

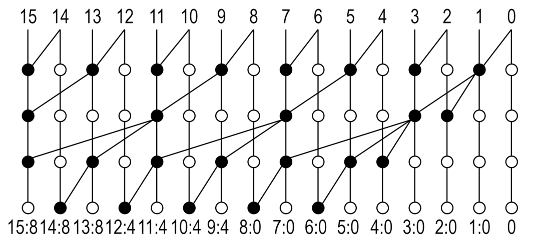
In a series of consecutive associations of generate and propagate pairs, $(G_{k:j}, P_{k:j})$ denotes the group generate and propagate term produced out of bits $k, k-1, \ldots, j$,

$$(G_{k:j}, P_{k:j}) = (G_k, P_k) \circ (G_{k-1}, P_{k-1}) \circ \ldots \circ (G_j, P_j), \tag{3}$$

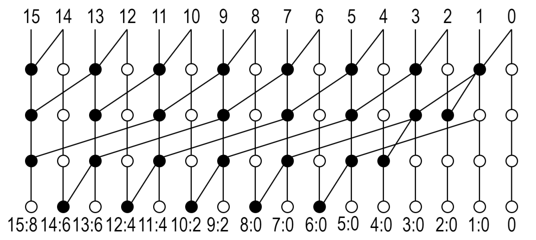
and carry C_i corresponds to $G_{i:0}$. This condition for carry C_i makes the adder *accurate*.

Fig. 1 depicts two parallel-prefix carry-propagate adders and Fig. 2 highlights the logic-level implementation of their basic blocks. Each adder is organized in three stages. The pre-processing stage computes bits G_i, P_i and H_i. Parallel prefix carry computation is done in the second stage using $\log_2 n$ prefix levels. The last node of each bit column requires a simpler implementation of the $\circ$ operator since only a group generate term $G_{i:0}$ needs to be computed. The last stage computes the sum bits according to (1).

Parallel prefix adders can take many forms depending on the number of prefix levels used and the internal fanout allowed. This affects directly the number of operators needed



(a)



(b)

Fig. 3. Two examples of approximate 16-bit parallel prefix adders. Solution (a) requires less operators than solution (b) but offers a smaller minimum carry chain length. In solution (a) the carry of bit position 9 checks only 5 less significant bit positions, making the carry chain 6 bits long, while in solution (b) the carry chains of all inaccurate bit positions is at least 8 bits long.

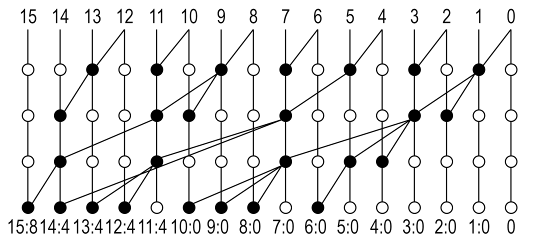


Fig. 4. An example adder synthesized by the proposed approach that combines the benefits of both worlds. It achieves the minimum carry-chain length of the adder in Fig. 3(b) and requires a similar number of operators and maximum fanout as the more area efficient adder depicted in Fig. 3(a).

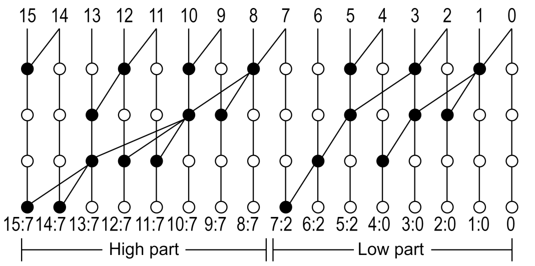


Fig. 5. An example of a split-accuracy adder synthesized by the proposed approach. The prefix tree of each part is designed separately and the two parts are attached together to form the approximate parallel prefix adder.

to complete accurate carry computation. By allowing more prefix levels than the minimum possible, offers more freedom that can be translated to reduced number of operators [5], [6], [7]. For large input wordlengths sparse parallel prefix adders are preferred, since the wiring and area of the design are significantly reduced without sacrificing delay [24]. Parallel prefix formulation of addition has been expanded to other forms of carries, such as Ling carries [25], and even to sum-propagate adders [26].

B. Approximate Parallel Prefix Adders

In this work, without loss of generality, *approximate carry computation* means that the parallel prefix trees are not obliged to associate every carry C_i with the full group generate term $G_{i:0}$ but it may use a group generate term of smaller length, i.e., $G_{i:m}$, with $m > 0$.

Fig. 3 depicts two representative examples of approximate 16-bit parallel prefix adders, designed according to the methodology presented in [10]. In both cases, the nine low-order carry bits are computed accurately while the rest are computed approximately using the minimum number of prefix levels that corresponds to a 16-bit adder. The accuracy achieved is not the same for all bit positions. For instance, in the case shown in Fig. 3(a) the carry chain of bit position 9 is only 6 bits long while the carry chain length of bit position 15 is 8 bits.

On the contrary, the adder that is shown in Fig. 3(b) achieves a minimum chain length of 8 bits in inaccurate bit positions 9, 13. To achieve this goal the second adder utilizes three more prefix operators, i.e. 28 operators are used by the adder of Fig. 3(b) and 25 operators are used by the adder in Fig. 3(a). The more accurate carry computation achieved by the adder of Fig. 3(b) is expected to translate to better numerical accuracy of the addition operations with the cost of marginally increased area and power. Which approach should be selected is totally application dependent.

However, other parallel prefix structures, like the one shown in Fig. 4 that was synthesized using the proposed approach can achieve the benefits of both adders shown in Fig. 3 under the same prefix levels. It achieves a minimum carry-chain length of 8 to all bit positions as done by the adder of Fig. 3(b), having the same or larger carry chain for every bit position, but needs only 26 operators and a maximum fanout of 3, similarly to the adder of Fig. 3(a).

The approximate parallel prefix topologies may refer to the whole adder width or they can be designed separately for the low and the high-order bits following a split-accuracy configuration [13], [15]. For instance, Fig. 5 depicts a split-accuracy parallel prefix adder designed by the proposed method, where the carry chains of the two parts do not interfere or share any operators, i.e., all carry chains of the most significant part stop at bit position 7.

III. Enumeration of Approximate Parallel Prefix Adders

The proposed approach for synthesizing approximate parallel prefix adders extends the enumerating approach of [7] to include also approximate solutions that satisfy the given design constraints. Building an approximate parallel-prefix adder of $n + 1$ bits involves adding recursively prefix operators to an adder of n bits only to the most significant column of the $(n + 1)$-bit adder. The n less significant bit positions are kept unmodified. This process is applied by using every possible

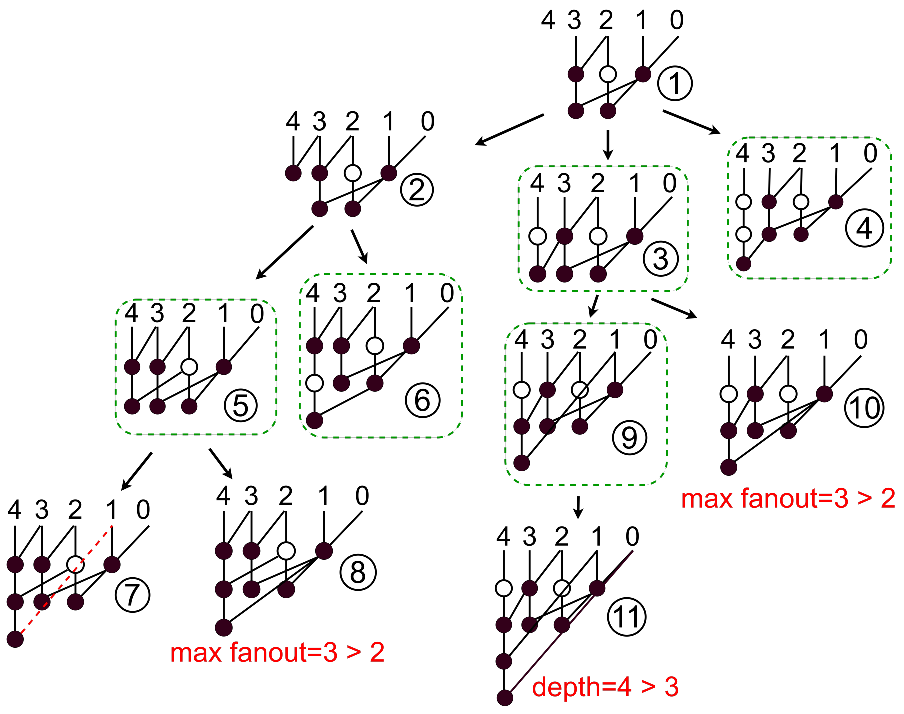


Fig. 6. Illustrative example of recursively generating 5 bit adders starting from a 4 bit adder. Every solution with an accuracy of $k = 3$ or higher is saved.

valid n-bit adder as the starting point for generating $(n+1)$-bit adders.

A. Designer constraints

The size of the solution space increases rapidly since an n bit adder can lead to multiple $(n+1)$-bit approximate adders. Therefore, some possible solutions are skipped.

A solution is considered invalid when it violates the maximum allowed number of prefix levels (depth) and maximum fanout set by the designer.

The designer must also specify the approximation target for the adders. This is quantified by a single number k, and means that each carry chain in the adder should be at least k bits long for bit positions larger than k, and fully accurate for every bit position smaller than k. This constraint affects which adders are considered complete and will be passed on to the construction of the next bit width adders. Any adder at least as accurate as the requested accuracy constraint is acceptable, even if it is of higher accuracy than requested.

In split-accuracy configuration the designer sets a minimum carry chain length k separately for each part.

B. Recursive Enumeration of Prefix Trees

The recursive procedure used for generating approximate parallel prefix adders will be described via the example shown in Fig. 6. In this example we arbitrarily start from the 4-bit adder depicted as ① in Fig. 6. The full application of the proposed approach would start from every possible valid 4-bit adder as a root solution. Our goal is to generate all valid 5-bit adders stemming from this initial solution that satisfy our design constraints. Let's assume that the design constraints are: 5-bit adders with a maximum fan-out of 2, maximum depth of 3 and minimum accuracy of $k = 3$. This means that the last operator on column 4 will need to look back at least until bit position 2.

The following steps are executed for each solution:

1) Find the LSB of the latest node inserted in column $n + 1$. The LSB is the least significant bit included in the computation in column $n + 1$. If no operators have been added yet, the LSB is equal to $n + 1$.
2) Create the next set of solutions by inserting the new operator in column $n + 1$ and connecting it to the node directly above it and to every possible node in column $LSB - 1$.
3) Proceed to the next set of solutions and repeat the process. A solution does not generate new solutions either if it's fully accurate or if it already violates some design constraint.

We execute the above steps for the example of Fig. 6, starting with solution ①. Since a 5-bit adder is being constructed, operators will be added only on column 4. Since no operator has been inserted yet, $LSB = 4$ (step 1). For step 2, we need to identify all the valid positions in column $LSB - 1 = 3$ for the new operator to be connected.

The first valid connection is the input of column 3. This connection produces solution ②. The other two solutions connect the new operator to the already existing operators of column 3. The first one is placed in the first prefix level producing the group generate and propagate term 3:2, and the second one in the second prefix level computing the group term 3:0. Connecting the new operator to be added to these positions will generate solutions ③ and ④, respectively. Solution ③ has a minimum accuracy of 3 so it's accepted as a final solution. Solution ④ is fully accurate, so it is also accepted as a final solution. On the other hand solution ② is a valid solution but doesn't meet the minimum required accuracy, so it is not added to the list of final solutions.

The process is repeated recursively for each new solution. For instance, for solution ② the LSB is equal to 3 since the last operator on column 4 computes the bits 4:3. Therefore the new operator added should be connected to column 2. The choices are to either connect it to input bit 2, or to the operator that generates the group term 2:0. This expansion of the prefix tree leads to solutions ⑤ and ⑥, respectively. Solution ⑤ has an accuracy of 3 and solution ⑥ is fully accurate, so they are both accepted as final solutions.

Even though solution ⑤ is accurate enough to be accepted as a final solution, it is not fully accurate. Therefore any solution stemming from it should still be examined, since they can lead to a better QoR solution later on. In this case, $LSB = 2$, and thus the new operator will be connected either to input bit 1 or to operator that generates group term 1:0 in column 1.

It has been observed that connecting new operators to input bits will potentially grant a worse QoR compared to connecting them to another operator, since the adder topology is used less efficiently. The first new operator added on each column (e.g. the operator added to generate solution 2) is excluded from this, since connecting it to an input bit is the only way to insert it on the first logic level, therefore no inefficiency is observed. On the other hand, connecting to an input bit instead of other operators can help generate solutions with smaller maximum fan-out, in cases where this constraint is tight. Our experiments have shown that connecting an operator to an input bit for two consecutive steps will only grant suboptimal solutions, consuming runtime without generating a high quality adder with small fan-out. Therefore no two consecutive operators should be connected to an input bit. After generating solution ⑦ by connecting the new operator to input bit 1, it is observed that the previous operator is also connected to an input bit (bit 2). Therefore solution ⑦ will be rejected for using the topology inefficiently. Solution ⑧ is also rejected since it violates the maximum fanout constraint on operator 1:0.

By continuing the recursive process, five final solutions are generated: solutions ③, ⑤ and ⑨ are approximate, whereas solutions ④ and ⑥ are fully accurate. Each one of these solutions will be passed on to the algorithm as a root solution (just like solution 1 is for this example) to generate one-bit larger 6-bit adders.

C. Solution pruning criteria

To avoid excessive runtime, two more pruning techniques are applied. First, we only keep a fixed number of solutions that have the same operator number, maximum level and maximum fanout. This pruning is applied when it is time to accept a solution as final. If there are more than 500 solutions with exactly the same characteristics, the new solution is rejected. This pruning of a valid solution is not expected to ruin the design-space exploration efficiency of the proposed approach since solutions that have the same characteristics most probably already explore a similar part of the design space. For instance, the 7-bit adders Ⓐ and Ⓑ of Fig. 7, have exactly the same characteristics in terms of prefix levels, number of operators and maximum fanout. Keeping both of these adders would most probably lead to synthesizing almost-similar adders and also paying an unnecessary runtime overhead. Therefore only a specific number of such adders will be passed on to the next level of the algorithm, e.g., for generating the 8-bit adders using these 7-bit adders as root.

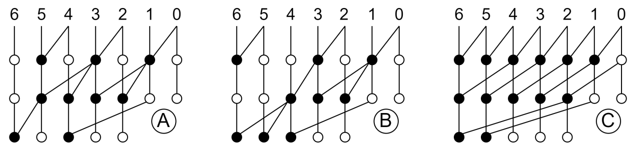


Fig. 7. Three 7-bit adders synthesized by our method. Adders A and B have similar characteristics and only a certain number of such adders will be kept, whereas adder C is much larger than A and B and can be possibly rejected for tight pruning criteria.

Second, a new valid solution that is much larger than the smallest valid solution already identified for the same bitwidth is automatically rejected. Specifically, if the smallest n-bit solution has s operators, every n-bit solution with more than $s + \Delta$ operators will be rejected. For example in Fig. 7, solutions Ⓐ and Ⓑ have 9 operators, whereas solution Ⓒ has 13 operators. For a value of Δ smaller than 4, solution Ⓒ would have been rejected. High values of Δ can produce a larger set of solutions, e.g., solutions with tight depth and maximum fanout constraints, whereas smaller values of Δ will greatly improve the runtime if the designer selected relaxed design constraints for the approximate adder under design.

We noticed experimentally that Δ should be linearly dependent to the adder's bit width. Therefore, we empirically set $\Delta = 0.7n$ where n is the adder's bit width. For instance, for $n = 16$ bits $\Delta = 11$, and for $n = 32$ bits $\Delta = 22$. This pruning is applied as new solutions get generated: if any new solution (final or intermediate) has more operators than the minimum adder size + Δ for this specific bit width, it is rejected.

D. Supporting Split Accuracy

Another broad category of adders that can be synthesized by our method are adders of split accuracy. In this case, the adder is split in two parts that do not share any operator. Also, each part is characterized by its own minimum carry chain length requirement, while both parts share the same constraints with respect to the maximum number of prefix levels and fanout. To support the synthesis of split-accuracy adders, the proposed method does not require any significant modification to its main recursive algorithm.

To be more specific, let's assume that we need to design an n-bit adder that consists of an m-bit low-accuracy part and $(n-m)$-bits high-accuracy part. Each part is characterized by its own minimum carry length requirement. At some recursive step, the algorithm would have generated all valid m-bit adders. Taking such adders as root solutions and extending

TABLE I
ENUMERATION RESULTS FOR 16 BIT APPROXIMATE AND FULLY ACCURATE ADDERS.

Carry Chain	depth	req max f.o.	Smallest solution # nodes	Smallest solution max f.o.	# solutions	Runtime (mins)
16	4	8	31	8	11807	0.20
		6	32	6	8418	0.18
		4	34	4	2238	0.10
		2	42	2	13	0.01
11	4	8	29	5	51110	0.35
		6	29	5	36937	0.27
		4	30	4	20382	0.16
		2	39	2	1516	0.02
8	4	8	26	3	144773	0.67
		6	26	3	127487	0.53
		4	26	3	82801	0.33
		2	28	2	8922	0.06
8	3	8	41	2	1	0.01
		6	41	2	1	0.01
		4	41	2	1	0.01
		2	41	2	1	0.01

them by one bit, means that we cross the border between the low-accuracy and the high-accuracy part. When this happens, the process effectively restarts and the new operators added in the high-accuracy part are not allowed to connect to any operator of the lower part. In this way, the solutions derived so far for the lower-accuracy part remain unchanged. Also, for the rest of the enumeration procedure, the minimum carry chain constraint of the high-accuracy part is only taken into account.

IV. Evaluation

The evaluation of the proposed approach is presented in four steps. In the first step, the relation between the designer's constraints and the number of derived solutions as well as the structure of the best choices is investigated. In the second step, we compare the proposed approach to state-of-the-art in terms of numerical accuracy on random inputs and hardware complexity. Next, we evaluate the accuracy of the proposed adders on an image filtering application and a neural-network classifier. In the last step, split-accuracy adders synthesized by the proposed method are compared to relevant state-of-the-art for hardware complexity and accuracy for the same benchmark applications.

A. Design Space Exploration Efficiency

In this section, we highlight the effectiveness of the enumeration algorithm with respect to various examined design constraints. Tables I and II depict the number of operators and the maximum achieved fanout of the smallest solution derived (in terms of number of operators) for 16 and 32-bit adders and for various design constraints. Also, we report the execution time of each run and the total number of solutions that are derived in each case. The design constraints include various combinations of minimum allowed carry-chain length, and the maximum allowed number of prefix levels (depth) and internal fanout ('req max f.o.'). The selected design constraints are reported in the first three columns of Tables I and II.

In both Tables I and II, the first group of rows refers to fully-accurate parallel prefix adders. Although there are many choices of possible solutions as shown in column '# solutions', the smallest ones match the number of operators of state-of-the-art parallel prefix adders available in open literature for various maximum fanout choices [6], [7]. This shows that our algorithm performs correctly at least for the fully accurate adder case, where optimal prefix structures are already known.

TABLE II
ENUMERATION RESULTS FOR 32 BIT APPROXIMATE AND FULLY ACCURATE ADDERS.

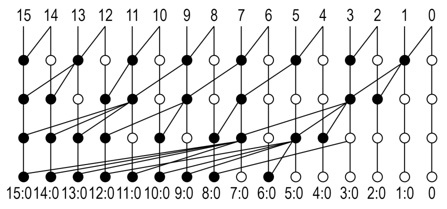
Carry Chain	depth	req max f.o.	Smallest solution # nodes	Smallest solution max f.o.	# solutions	Runtime (mins)
32	5	16	75	16	41789	48.77
		12	79	12	22859	36.72
		10	82	10	20825	28.50
		8	88	8	11645	22.28
		6	91	6	2977	13.90
		4	107	4	201	9.40
22	5	16	72	12	279291	57.62
		12	72	12	209618	45.41
		10	73	10	196464	37.48
		8	75	8	150398	29.22
		6	78	5	113160	20.18
		4	85	4	57830	13.33
16	5	16	64	7	1321248	118.27
		12	64	7	1128069	77.16
		10	64	7	454208	60.49
		8	64	6	436093	44.72
		6	66	6	303434	32.62
		4	68	4	176845	21.81
16	4	16	113	3	1	0.33
		12	113	3	1	0.33
		10	113	3	1	0.33
		8	113	3	1	0.33
		6	113	3	1	0.29
		4	113	3	1	0.17

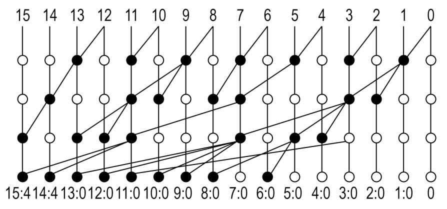
The rest rows refer to various forms of approximate adders. For instance, Fig. 8 depicts the smallest solution derived for the constraints of rows 3, 7 and 11 respectively, each one including 34, 30 and 26 prefix operators as highlighted in Table I.

In overall, as the maximum fanout constraint increases, i.e. gets more relaxed, it is observed that the adder size decreases or stays the same. This is expected since a relaxed fanout constraint drives our algorithm to more efficient solutions. By tightening the maximum fanout constraint, the number of final valid solutions is also reduced. It can also be observed that the smallest adder doesn't always utilize all of the allowed maximum fanout for the case of approximate adders. As the carry chain becomes smaller, the number of operators involved in each chain also decreases. This property reduces naturally the fanout requirements.

The maximum allowed depth has a direct effect on the size of the final solution. It can be observed that as the maximum depth constraint becomes more tight, i.e. approaching the minimum number of $\log_2 n$ levels for a n-bit adder, the size of the smallest solution increases considerably.

The properties of synthesized 32-bit adders, shown in Table II, follow a similar trend. For example, comparing the fully accurate solutions with the approximate ones, we observe that the number of operators is significantly reduced as the accuracy reduces for the same maximum fan-out and number of prefix levels. The total number of solutions found also increases as the accuracy constraint decreases, since


(a) Accurate


(b) Min. Carry Chain 11 bits

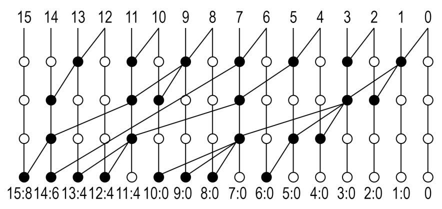

(c) Min. Carry Chain 8 bits

Fig. 8. Accurate and approximate 16-bit parallel prefix adders synthesized by the proposed approach using a maximum of 4 prefix levels and equivalent maximum fanout constraints.

more adders that fit the carry chain length criterion can be constructed.

The measured runtime is not prohibitive, even if enumeration covers thousands of solutions in many cases: It is always less than 1 minute for 16-bit adders and less than 2 hours for 32-bit adders. These runtimes were taken on a generic laptop with a 2.2 GHz Intel Core i7 CPU and 8 GB of RAM for a single-threaded C++ implementation. Since each solution starts recursively from a different root solution, it is is straight forward to implement this method in a multithreaded fashion. In this case, the runtime is expected to reduce significantly. In overall, it can be observed that experiments with a higher number of valid solutions, which are the cases with more relaxed constraints, also have a higher runtime. Moreover, since we retain a fixed number of solutions per operator number, level and maximum fanout, relaxing any of those constraints further increases the runtime. In most cases, the runtime usually reduces in half as we tighten the required maximum fanout.

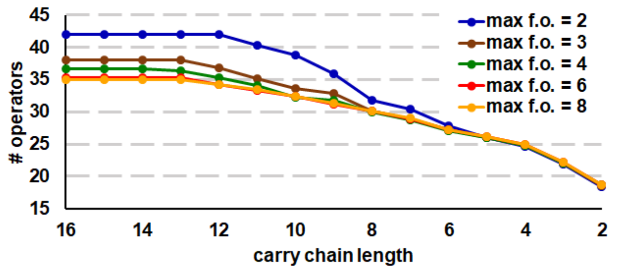


Fig. 9. The average number of operators synthesized by the proposed approach for 16-bit adders assuming various carry chain lengths and maximum fanout (‘max f.o.’) constraints. In all cases, the number of prefix levels is set to four.

Besides the properties of the smallest solution derived for each set of constraints, it is interesting to explore the properties of all approximate adders synthesized by the proposed method. In this direction, Fig. 9 depicts how the average number of operators of the derived 16-bit adders change according to the minimum carry chain length requested, for various maximum fanout requirements. In all cases, the maximum allowed number of prefix levels is set to four.

When minimum carry chain length is close to the adder’s bitwidth only a very small amount (if any) of operations can be saved. In Fig. 9, the number of operators reduces when dropping the requirement of minimum carry chain length below 12 bits. The second observation is that for carry chain lengths of 8 bits (half the total bit width of the adder) or less, the maximum fanout constraint is not critical any more, since all results converge to the same number of operators. This is expected since, as the minimum carry chain required is lowered, less operators will belong on the same chain, and there are less opportunities for an operator to reach its maximum fanout constraint. In overall, as the accuracy constraint is relaxed, the maximum fanout constraint becomes increasingly less important and eventually for extremely relaxed cases doesn’t impact the result at all.

B. Accuracy comparisons with state-of-the-art approximate parallel prefix adders

In this section, we compare state-of-the-art approximate parallel-prefix adders [10] with equivalent parallel-prefix adders derived by the proposed synthesis algorithm for the case of 16-bit adders. The adders of [10] represent a generic family of efficient approximate parallel adders and can be considered a superset of other published solutions [11], [12], [17]. The adders of [10] include also an error correction mechanism. This is out of the scope of this work, and it is orthogonal to the proposed approach. So in order to ensure a fair comparison we excluded it from the adders we recreated.

Both the proposed adders and the ones presented in [10] are shown in Fig. 10. The approach of [10] creates hybrid accuracy adders by defining a maximum and a minimum carry chain. For this reason, for each adder of [10] that we compare against, we present two of our own solutions. The first one, named Proposed I, has a minimum accuracy equal to the maximum carry chain length of the adders under comparison. For an apple-to-apple comparison Proposed I adders have the same maximum fanout as the maximum fanout of the adders of [10]. On the contrary, the second solution, named Proposed II, is derived by setting the minimum carry chain length between the minimum and maximum carry chain of the adders we are comparing against and allow for higher fanout to improve operator sharing.

To quantify the accuracy of the approximate parallel prefix adders we drive each adder with 50,000 random inputs and check the addition result against the fully accurate one. Random inputs are derived from two probability distribution functions. The first is a purely uniform distribution (i.e., $u = 1$ with u being the probability of a sample belonging to the uniform distribution), where the samples are uniformly distributed between $[-2^{n-1}, 2^{n-1}-1]$ with n being the adder’s bit width. The second one follows a mixed structure of a half uniform

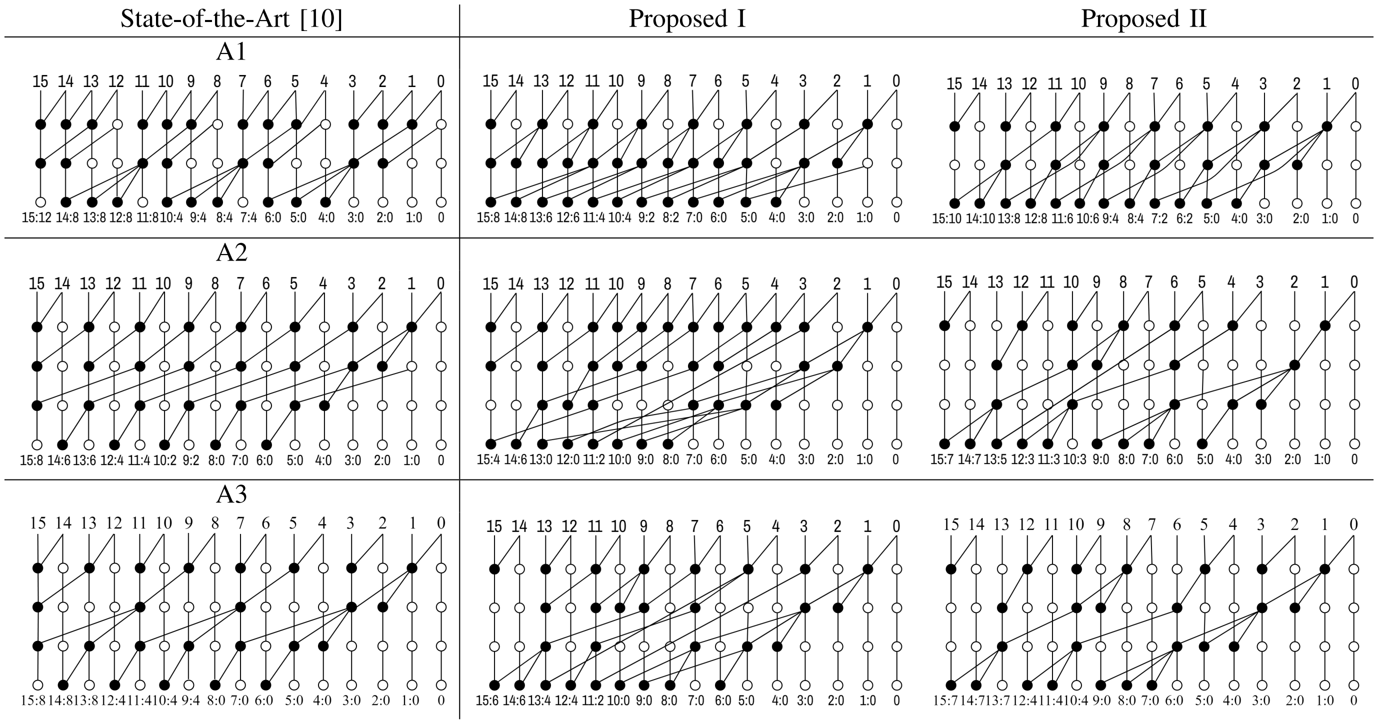


Fig. 10. The example adders used for the comparison to state-of-the-art. For each adder proposed in [10] on the left, we use two adders synthesized by the proposed approach. The two used solutions can help us understand better the design space that can be covered by the proposed enumerative approach.

TABLE III
COMPARISON OF THE PROPOSED 16-BIT ADDERS AGAINST STATE-OF-THE-ART APPROXIMATE PARALLEL PREFIX ADDERS [10].

Adder	Carry Chain Length		max fanout	nodes	Uniform		Mixed	
	min	max			EF	MRE	EF	MRE
A1	4	7	3	29	0.088	2.89	0.176	28.64
Prop. I	7	8	3	33	0.012	0.95	0.075	33.87
Prop II	5	6	3	27	0.058	0.96	0.159	18.85
A2	8	9	3	28	0.006	1.93	0.071	71.27
Prop. I	9	14	2	36	0.001	1.86	0.018	68.94
Prop. II	8	10	3	26	0.004	1.16	0.037	39.89
A3	6	9	3	25	0.015	0.96	0.074	27.66
Prop. I	9	11	3	29	0.002	0.81	0.018	35.98
Prop. II	7	10	3	25	0.007	0.70	0.038	22.82
Avg. Savings Prop. I	-	-	-	-19%	86%	37%	65%	8%
Avg. Saving Prop. II	-	-	-	4%	36%	51%	27%	36%

and half Gaussian distribution. Mixed distribution means that the chance of a sample belonging to the uniform distribution is 50%, i.e. $u = 0.5$. For the Gaussian distribution we assume $\mu = 0$ and $\sigma = 256$.

We use two metrics for evaluating the results. The first metric, named Error Frequency (EF), quantifies how often an adder returns an inaccurate result irrespective on the amount of error:

$$EF = \frac{\text{total erroneous samples}}{\text{total samples}}$$

An erroneous sample is a set of inputs for which the adder returns the wrong sum.

The second metric, named Mean Relative Error (MRE), represents how far the erroneous sum is from the correct result for all samples:

$$MRE = \frac{\sum_i RE_i}{\text{total samples}}.$$

Relative error RE_i refers to the error of the ith sample:

$$RE_i = \left| \frac{\text{sum} - \text{expected sum}}{\text{expected sum}} \right|$$

with the expected sum being the correct result and sum being the actual result returned by the adder for the ith input sample.

The obtained results are highlighted in Table III. For the Proposed I adders, we notice that the operator number is always higher. This is expected, since we are requesting for more accurate adders by construction for all bit positions than the adders [10]. On average, the Proposed I adders have 19% more operators while achieve 86% lower error frequency on

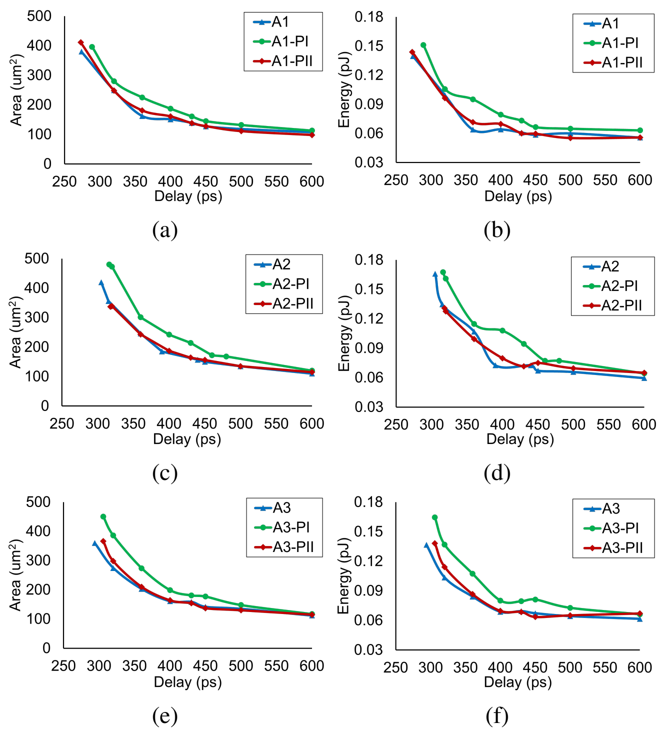


Fig. 11. The Area-Delay and Energy-Delay curves for the adders under comparison for various delay targets when synthesized at 40nm/0.8V.

random uniform inputs and 65% on random inputs derived from the mixed distribution.

The Proposed II adders achieve a better error frequency than the adders [10] and also marginally decrease the number of operators. For instance, the Proposed II adders achieve 36% lower error frequency on average on random uniform inputs, and 27% on inputs derived from a mixed distribution function. This improvement comes at no cost since the operator number decreases by 4% on average, as shown in the last row of Table III. Proposed II adders represent a more balanced solution in overall, where both hardware resources are saved, and approximate addition is performed with more accuracy than [10]. This conclusion is also supported by the performance of the proposed adders with respect to MRE.

In overall, the proposed adders given an erroneous result less often relative to [10], as shown by their performance with respect to EF, and the magnitude of the error is also smaller as shown by the MRE.

C. Hardware Complexity Comparisons

The adders under comparison shown in Fig. 10 and used in Table III have been synthesized from Verilog RTL using Cadence's digital implementation flow and a commercial-grade 40 nm standard-cell library under a mixed V_T configuration at 0.8V. All adders were synthesized for a variety of delay targets, starting from their minimum delay point up to a maximum delay target of 600ps. The area and energy of adders A1, A2 and A3 together with their equivalent Proposed I and II solutions shown in Fig. 10 are depicted in Fig. 11. Figs. 11(a), (c), (e) on the left column show how the area of each adder scales in respect to the delay target. Similarly, Figs. 11(b), (d), and (f) show the corresponding scaling of their energy. The energy values were derived from the same inputs for which we calculated the error frequency and mean relative error metrics.

The minimum achievable delay between the adders compared in each case is very similar. This is expected since the Proposed I and II adders were designed to have the same number of prefix levels as the adders presented in [10] and also have a similar fanout distribution on the critical paths.

After a closer examination, we see that Proposed I adders always have a slightly higher area or energy for the same delay target, which eventually converges to the same area/energy as Proposed II and the state-of-the-art adders for more relaxed delay targets. This is easily explained by the fact that Proposed I adders have higher operator count, in exchange for more accurate behavior. The Proposed II adders have a very similar area-energy behavior as the adders from [10], occasionally becoming marginally better or worse, but still achieve a significantly better error behavior as seen in Table III.

D. Application-level Performance Comparisons

The examined approximate adders are also tested on (a) an image filtering application that employs a mean filter on grayscale images and (b) a neural-network classifier trained for the MNIST dataset.

1) Mean Filter: Mean filtering is applied for smoothing 8-bit grayscale images by calculating the mean value of the pixels in a window around each pixel. In this example, we used a 5×5 window, and thus the mean value of 25 neighbors are computed using the 16-bit approximate adders under comparison. An example of the application of mean filtering to a selected image is shown in Fig. 12. This example includes the result of mean filtering using a fully-accurate adder, as well as the result of mean filtering when employing three variants of the examined approximate adders.

To quantify the overall impact of approximate addition to the performance of mean filtering relative to the result taken from a fully-accurate 16-bit adder, we used two well-known metrics: the Peak Signal-to-Noise Ratio (PSNR) and the Mean Structural Similarity (MSSIM) [27]. PSNR quantifies the amount of noise present in an image. PSNR is a logarithmic quantity in the decibel (dB) scale that has low values for noisy images, while its value increases with the reduction of noise. The MSSIM is a metric that quantifies the similarity between two images and its values range from -1 to 1.

Table IV reports the PSNR and MSSIM achieved by the examined approximate adders for each test image, together with their average values for all examined cases. The input images are publicly available and taken from [28]. In all cases, Proposed I adders outperform the adders of each category, since they improve PSNR and MSSIM by 42% and 30% on average. The Proposed II adders follow a similar trend to the one highlighted in Table III for random inputs. Even if they are worse than the Proposed I adders, they outperform in almost all cases the adders of [10] by 15% and 8% on average for PSNR and MSSIM metrics. As expected, the adders of the category A1 exhibit the lower performance since the corresponding approximate adders have smaller chains relative to A2 and A3.

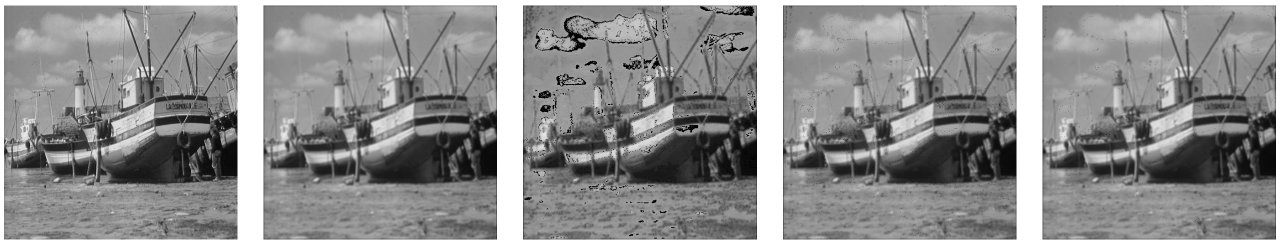

(a) Original (b) Accurate (c) A1 (d) A1 Prop. I (e) A2 Prop. II

Fig. 12. (a) The reference 8-bit grayscale image and (b) the result of mean filtering when using a fully-accurate 16-bit adder. The result of mean filtering when using (c) the A1 adder, (d) the Proposed I adder of the A1 category and (e) the Proposed II adder of the A2 category.

TABLE IV
THE PERFORMANCE OF THE APPROXIMATE ADDERS UNDER COMPARISON FOR A MEAN FILTERING APPLICATION RELATIVE TO A FULLY-ACCURATE RESULT USING THE PEAK SIGNAL-TO-NOISE RATIO (PSNR) AND THE MEAN STRUCTURAL SIMILARITY (MSSIM) QUALITY METRICS.

Image	Metric	A1	A1 Prop. I	A1 Prop. II	A2	A2 Prop. I	A2 Prop. II	A3	A3 Prop. I	A3 Prop. II
boat	PSNR	16.65	30.83	24.96	29.19	35.88	34.29	30.57	35.21	31.00
	MSSIM	0.5204	0.8216	0.6068	0.8699	0.9646	0.9236	0.8396	0.9547	0.8721
bridge	PSNR	20.22	30.66	24.13	32.41	42.60	34.64	30.06	42.35	31.02
	MSSIM	0.6675	0.8569	0.6254	0.9280	0.9963	0.9389	0.8340	0.9912	0.8770
couple	PSNR	19.86	30.22	24.34	31.71	40.40	33.86	28.72	40.64	30.55
	MSSIM	0.5005	0.7622	0.5156	0.8379	0.9883	0.8922	0.7141	0.9836	0.8302
house	PSNR	21.05	30.35	24.41	33.39	42.70	34.54	28.13	42.34	31.49
	MSSIM	0.5372	0.7118	0.5454	0.7982	0.9970	0.9040	0.6892	0.9898	0.8627
female	PSNR	19.87	29.63	25.35	30.57	40.95	33.31	28.94	40.71	30.06
	MSSIM	0.5185	0.7492	0.5820	0.8250	0.9918	0.8991	0.7332	0.9891	0.8556
mandril	PSNR	15.42	28.14	21.87	28.39	35.92	31.61	27.11	35.49	28.45
	MSSIM	0.5013	0.7911	0.5346	0.8525	0.9768	0.9063	0.7663	0.9698	0.8389
peppers	PSNR	16.29	30.43	23.54	29.85	36.46	33.99	30.13	36.74	30.66
	MSSIM	0.4961	0.8006	0.5645	0.8842	0.9750	0.9184	0.8168	0.9813	0.8767
Average	PSNR	18.48	30.04	24.09	30.79	39.27	33.75	29.09	39.06	30.46
	MSSIM	0.5345	0.7848	0.5678	0.8565	0.9843	0.9118	0.7705	0.9799	0.8590

To better highlight the tradeoff between application-level performance and hardware complexity, Fig. 13 depicts in the same diagram the combined performance of two figures-of-merit: The energy-delay product at 450 ps and the average PSNR computed after applying mean filtering on all images using the examined approximate adders. In this case, adders of the A2 and A3 categories exhibit the best overall performance. The Proposed I adders of those two categories, i.e., A2-PI and A3-PI, achieve the largest average PSNR with the highest energy. On the contrary, The Proposed II adders, i.e., A2-PII and A3-PII, exhibit a more balanced performance. In those cases, PSNR is improved relative to the adders of [10], with either better energy-delay product, as in the case of A3-PII, or slightly increased for A2-PII. From the A1 category of adders only the Proposed I adder A1-PI shows a notable performance, since it achieves an average PSNR close to the average PSNR of A2 and A3 with the same energy-delay profile.

2) Neural Network: Next, we designed a Multi-Layer Perceptron (MLP) with 784 neurons in the input layer, 500 in the hidden layer and 10 in the output layer, trained for digit recognition. The MLP was trained in PyTorch 2.0.1 [29] using 50K 28×28 images from the MNIST dataset [30]. Training was performed with the goal to identify the quantized 8-bit weights that minimize classification error.

The remaining 10K images of MNIST dataset were used for the inference stage where we evaluate the accuracy of the classification using accurate and approximate adders. For inference, we assume fully accurate 8×8 multipliers that

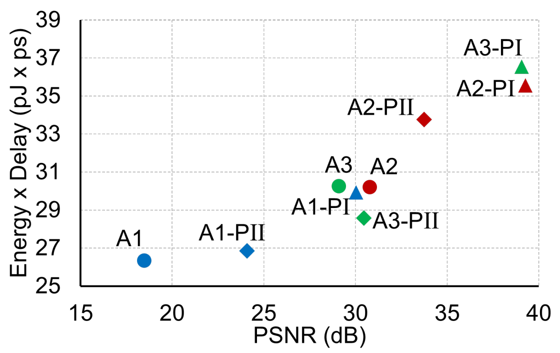


Fig. 13. The combined behavior of the examined approximate adders with respect to Energy×Delay at 450 ps and the average PSNR achieved for each case after applying mean filtering to all test images.

produce 16-bits wide products. Addition is performed at 32 bits using fully accurate and several forms of approximate adders. In the future, we plan to extend this study using also approximate multipliers [31], [32].

The reference accuracy of the MLP using accurate adders is 96.82%. The accuracy achieved for a variety of approximate adders is summarized in Fig. 14. The performance for the A2 and A3 adders depicted as straight lines in Fig. 14 corresponds to the performance achieved by the 32-bit counterparts of the 16-bit A2 and A3 adders shown in Fig. 10. A3 adder that has a minimum carry chain length of 10 bits and a maximum carry chain length of 24 bit achieves an accuracy of 96.72%.

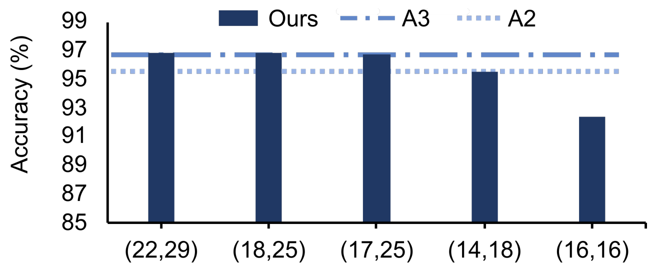


Fig. 14. Classification accuracy of the proposed approximate parallel prefix adders with different carry chain lengths compared to state-of-the-art 32-bit approximate adders.

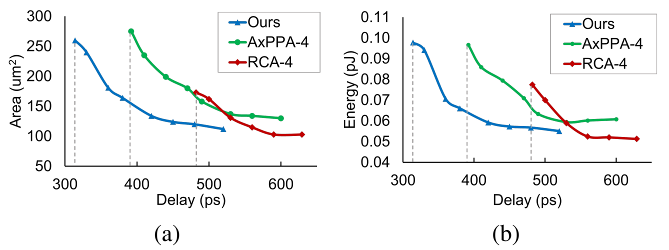


Fig. 15. The (a) Area-Delay and (b) Energy-Delay curves for split-accuracy 16 bits approximate adders when synthesized at 40nm/0.8V for different delay targets. The width of the low-accuracy part is selected to be 4-bits wide in all cases.

Similarly, A2 with a minimum carry length of 16 bits and a maximum carry length of 17 bits achieves an accuracy of 95.55%.

For the proposed adders we considered five cases, each one characterized by a different minimum and maximum carry chain length. For instance, solution (22,29) means that the corresponding 32-bit adder has a minimum carry chain length of 22 bits and maximum one of 29 bits. All 32-bit adders consist of 5 prefix levels and have a maximum allowed fanout of 8. The first three adders synthesized by the proposed method achieve accuracy greater than 96%. Accuracy degradation starts to appear on smaller carry chain lengths. The (14,18) configuration achieves a performance equal to the A2 adder.

E. Evaluation of split-accuracy adders

To complete the evaluation, we compare the split-accuracy parallel prefix adders synthesized by our method to state-of-the-art split-accuracy parallel prefix adders [13]. The adders of [13] consist of an accurate parallel prefix tree on the high-accuracy part of the design and very simplified logic on the low-accuracy part. The prefix tree of the high-accuracy part can follow any prefix topology. At the border between the high and the low-accuracy part a carry-generate bit is produced that enters the high-accuracy part as a pseudo carry-in signal through an additional carry increment stage.

In this comparison we include also an approximate ripple-carry adder that follows also the split-accuracy paradigm [15], as a baseline outside the family of parallel prefix adders. In this case, the low-accuracy part does not involve any carry generation or propagation and the sum bits are computed directly from the input bits of the same column. On the contrary, the high-accuracy part is a complete ripple-carry adder driven by a carry-in signal of 0.

1) Hardware Complexity: Fig 15 depicts the area-vs-delay and energy-vs-delay of three 16-bit split-accuracy adders. For the approximate ripple-carry adder (RCA) and the split-accuracy parallel prefix adder of [13] (AxPPA) we assume that the low-accuracy part consists of 4 bits and the high-accuracy part covers 12 bit positions. The high-accuracy part of AxPPA is built following a Ladner-Fischer topology. The proposed adder has a low accuracy part of 4 bits too, and the maximum carry chain length at each part is equal to the width of the corresponding part. For an apple-to-apple comparison, both parallel prefix adders have the same number of prefix levels and maximum fanout requirements.

From Fig. 15 it is evident that the proposed designs are significantly faster than any of the two split-accuracy approaches. AxPPA adders are slower due to the extra carry-increment stage needed for including the pseudo carry in signal. Also, this speed benefit does not come with an area or energy cost. Under equal delay, e.g., at 400 ps, the proposed designs require 41% less area and 31% less energy than AxPPA [13]. The split-accuracy parallel prefix trees synthesized by the proposed method are by-construction more area/energy efficient since effectively they consist of two separate smaller adders attached together.

Approximate RCA appears to be area/energy efficient but is significantly slower than the parallel prefix architectures. In a low-power setup, such parallel prefix adders can tradeoff their improved timing slack for voltage reduction and thus save considerable amounts of energy relative to RCAs.

2) Numerical Accuracy: To compare the examined split-accuracy adders with respect to their numerical accuracy, we utilized the same mean filtering and neural network application described in Section IV-D.

TABLE V
THE AVERAGE PSNR AND MSSIM COMPUTED AFTER APPLYING MEAN FILTERING TO THE IMAGES OF TABLE IV USING THE EXAMINED SPLIT-ACCURACY ADDERS.

	Ours	AxPPA-4	RCA-4
Avg. PSNR	36.33	37.35	29.85
Avg. MSSIM	0.9144	0.9541	0.9108

Table V reports the average PSNR and MSSIM achieved by each approximate adder for all test images of Table IV. AxPPA adder and the one synthesized by the proposed approach show almost the same performance in terms of average PSNR, while AxPPA is better in terms of average MSSIM. Both parallel prefix solutions outperform the approximate RCA since they offer better accuracy at the lower part of the design. In overall, the proposed design exhibits the best overall performance since it saves significant amounts of area and energy relative to AxPPA [13], without degrading much the examined image quality metrics.

Lastly, we tested the performance of the same split-accuracy adder architectures for computing the inference of the MLP model used in Section IV-D. Multiplications are considered accurate and addition is performed approximately at 32 bits. The results gathered are reported in Fig. 16. For each split-accuracy adder we tried several alternatives derived after

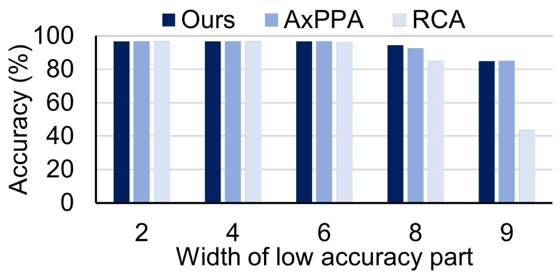


Fig. 16. Inference performance of an MLP trained for the MNIST dataset using accurate multipliers and split-accuracy 32-bit adders of varying width for their low-accuracy part.

increasing the width of the low-accuracy part. As long as the low-accuracy part is 6 bits or less, all architectures behave equivalently and very close to the fully-accurate result. When increasing further the width of the lower part the classification accuracy is degraded for all designs. In the last case that uses a low-accuracy part of 9 bits the approximate RCA affects significantly the inference accuracy of the MLP.

V. Related Work

Approximate adder design spans many different design choices that go beyond the scope of this paper that focuses on parallel prefix adder design.

For instance, approximate adders can be designed by utilizing circuit blocks organized in a slit-accuracy configuration. In this case [14], the adder is divided in two parts: the accurate one is responsible for adding the most significant bits and consists of a fully accurate adder, and the approximate one, which adds the least significant bits and consists of simple OR/AND gates in place of addition blocks. Follow up work [33] utilizes error correction to reduce the error frequency of [14]. Similarly, in [34], the adder is divided in three parts. The lowest part is implemented by simpler gates without producing any carry signals, the middle part is an approximate carry look-ahead adder and the higher part is an accurate adder. Another gate level approximation is presented in [4], where a tool for automatic gate pruning is proposed.

Logic-level approximations can be applied at the transistor level as well. The approximate mirror adder [3] emerges from addition cells with fewer transistors. Also, approximate full adders using are presented in [35] and [36].

Another broad category of methods for approximating addition focuses on the higher level architecture of the adder. Early efforts construct approximate adders by shortening the carry chain. For instance, in [37] and later in [38], the Almost Correct Adder (ACA) and the Variable Latency Speculative Adder (VLSA) are proposed, respectively. ACA consists of shorter carry generators that operate in parallel, while VLSA utilizes ACA's architecture and reduces the area overhead by sharing some computational blocks. Error detection and recovery circuits are also implemented in VLSA. Error compensation has been also recently used in [39].

In [40] and [41], addition is separated in two parts: the accurate, where the addition is performed accurately, and the inaccurate one, where an approximate technique is utilized. In [42], the same approach is enhanced by adding a selector circuit that determines how the bits will be divided into the two parts (the accurate and the inaccurate one). The Error-Tolerant Constant Adder (ETCA) proposed in [43] replaces the inaccurate part of the adder with a constant output. ETCA has smaller area and lower power properties, while maintaining acceptable levels of accuracy. The accuracy-configurable adder [44] includes an error detection and correction circuit.

Approximate carry-select and carry-skip adders have also been proposed. For example, the speculative carry-select adder in [45] is divided into n/k smaller adders (n being the bit-width of the adder and k the target carry-chain propagation length). Each sub-adder consists of two k-bit adders as needed by carry-select operation. Splitting the adder to smaller blocks has been also used in [46] to design approximate carry-skip adders.

Similarly, the gracefully-degrading accuracy-configurable adder [47] is split into k-bit smaller adders, each one connected to a multiplexer that selects an appropriate carry-in connection. This design has been extended in [48] to include also error correction. In [49] a speculative adder is divided into smaller adders that include, among other units, a carry predictor. The consistent carry approximate adder proposed in [50] follows a similar approach but carry prediction depends on a wider set of input bits. The carry cut-back adder presented in [51], utilizes a feedback technique to cut the carry chain and therefore prevent the activation of the critical path. The cut signal is generated by a carry propagate block. Finally, the simple accuracy reconfigurable adder [52] and the block-based carry speculative adder [53] rely also on smaller adder blocks and carry prediction.

VI. Conclusions

Parallel prefix computation of addition offers a versatile framework for designing adders of various area-power-delay characteristics. The automatic synthesis of parallel prefix adders allows the designer to explore deeply the available design space under the selected constraints. In this work, we generalize the synthesis of parallel prefix adders to include also approximate parallel prefix trees that compute carry chains of shorter length, under strict maximum fanout and maximum prefix level constraints. To limit the huge design space, heuristic pruning techniques keep the runtime under control without degrading the final quality-of-result.

The proposed synthesis approach allows to derive new approximate parallel prefix adders that have not appeared in open literature supporting uniform or split accuracy. The derived adders offer a two-fold benefit: They either offer significantly better accuracy for a minimum area/power overhead or allow for accuracy improvements with modest area and power savings making them especially beneficial for applications with power limitations like ambient assisted living systems developed in domestic environments.

Most importantly, the introduced synthesis engine is available under a permissive open-source license, which allows end-users to test it and extend it for future research.

Future work, is planned to explore the interplay between approximation criteria and how they affect the characteristics of the synthesized parallel prefix adders under Process-Voltage-Temperature (PVT) variations [54].

Acknowledgements

This work is supported by the project ”Study, Design, Development and Implementation of a Holistic System for Upgrading the Quality of Life and Activity of the Elderly” (MIS 5047294) implemented under the Action ”Support for Regional Excellence”, funded by the Operational Programme ”Competitiveness, Entrepreneurship and Innovation” (NSRF 2014-2020) and co-financed by Greece and the EU (European Regional Development Fund).

References

[1] P. Stanley-Marbell, A. Alaghi, M. Carbin, E. Darulova, L. Dolecek, A. Gerstlauer, G. Gillani, D. Jevdjic, T. Moreau, M. Cacciotti, A. Daglis, N. E. Jerger, B. Falsafi, S. Misailovic, A. Sampson, and D. Zufferey, “Exploiting Errors for Efficiency: A Survey from Circuits to Applications,” *ACM Computing Surveys*, vol. 53, no. 3, 2020.

[2] M. Pashaeifar, M. Kamal, A. Afzali-Kusha, and M. Pedram, “A Theoretical Framework for Quality Estimation and Optimization of DSP Applications Using Low-Power Approximate Adders,” *IEEE Trans. on Circuits and Systems I*, vol. 66, no. 1, pp. 327–340, 2019.

[3] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, “Low-Power Digital Signal Processing Using Approximate Adders,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, 2013.

[4] J. Schlachter, V. Camus, K. V. Palem, and C. Enz, “Design and applications of approximate circuits by gate-level pruning,” *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1694–1702, 2017.

[5] R. Zimmermann, “Binary Adder Architectures for Cell-Based VLSI and their Synthesis,” Ph.D. dissertation, ETHZ, 1998.

[6] S. Knowles, “A Family of Adders,” in *Proc. of the IEEE Symp. on Computer Arithmetic*, April 1999, pp. 30–34.

[7] S. Roy, M. Choudhury, R. Puri, and D. Z. Pan, “Towards Optimal Performance-Area Trade-Off in Adders by Synthesis of Parallel Prefix Structures,” in *Design Automation Conference (DAC)*, 2013, pp. 1–8.

[8] Y. Ma, S. Roy, J. Miao, J. Chen, and B. Yu, “Cross-Layer Optimization for High Speed Adders: A Pareto Driven Machine Learning Approach,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 12, p. 2298–2311, 2019.

[9] R. Roy, J. Raiman, N. Kant, I. Elkin, R. Kirby, M. Siu, S. Oberman, S. Godil, and B. Catanzaro, “PrefixRL: Optimization of Parallel Prefix Circuits Using Deep Reinforcement Learning,” in *Design Automation Conference (DAC)*, 2021, p. 853–858.

[10] D. Esposito, D. De Caro, and A. G. M. Strollo, “Variable Latency Speculative Parallel Prefix Adders for Unsigned and Signed Operands,” *IEEE Trans. on Circuits and Systems I*, vol. 63, no. 8, pp. 1200–1209, 2016.

[11] D. Esposito, D. De Caro, E. Napoli, N. Petra, and A. G. M. Strollo, “Variable Latency Speculative Han-Carlson Adder,” *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 62, no. 5, pp. 1353–1361, 2015.

[12] A. Cilardo, “A New Speculative Addition Architecture Suitable for Two’s Complement Operations,” in *Design, Automation Test in Europe (DATE)*, 2009, pp. 664–669.

[13] M. M. A. da Rosa, G. Paim, P. U. L. da Costa, E. A. C. da Costa, R. Soares, and S. Bampi, “AxPPA: Approximate Parallel Prefix Adders,” *IEEE Trans. on Very Large Scale Integration Systems*, vol. 31, no. 1, pp. 17–28, 2023.

[14] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, “Bio-Inspired Imprecise Computational Blocks for Efficient VLSI Implementation of Soft-Computing Applications,” *IEEE Trans. on Circuits and Systems I*, vol. 57, no. 4, pp. 850–862, 2010.

[15] P. Balasubramanian, C. Dang, D. L. Maskell, and K. Prasad, “Approximate ripple carry and carry lookahead adders—a comparative analysis,” in *2017 IEEE 30th International Conference on Microelectronics (MIEL)*. IEEE, 2017, pp. 299–304.

[16] G. Thakur, H. Sohal, and S. Jain, “FPGA-Based Parallel Prefix Speculative Adder for Fast Computation Application,” in *International Conference on Parallel, Distributed and Grid Computing (PDGC)*, 2020, pp. 206–210.

[17] F. Ebrahimi-Azandaryani, O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram, “Accuracy configurable adders with negligible delay overhead in exact operating mode,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 28, no. 1, Jan. 2023.

[18] IC-Lab-DUTH Repository. (2023) Synthesis approximate parallel prefix adders. [Online]. Available: https://github.com/ic-lab-duth/ApproximatePrefix

[19] N. Weste and D. Harris, *CMOS VLSI Design a Circuits and Systems Perspective*. Addison Wesley (3rd Edition), 2010.

[20] B. Lee and V. Oklobdzija, “Improved CLA Scheme with Optimized Delay,” *Journal VLSI Signal Processing Systems*, no. 3, p. 265–274, 1991.

[21] P. M. Kogge and H. S. Stone, “A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations,” *IEEE Trans. on Computers*, vol. C-22, pp. 786–792, Aug. 1973.

[22] R. E. Ladner and M. J. Fisher, “Parallel Prefix Computation,” *Journal of The ACM*, vol. 27, no. 4, pp. 831–838, Oct. 1980.

[23] R. P. Brent and H. T. Kung, “A Regular Layout for Parallel Adders,” *IEEE Trans. on Computers*, vol. 31, no. 3, pp. 260–264, Mar. 1982.

[24] A. Beaumont-Smith and C. C. Lim, “Parallel-Prefix Adder Design,” in *Proc. of the IEEE Symp. on Computer Arithmetic*, Apr. 2001, pp. 218–225.

[25] G. Dimitrakopoulos and D. Nikolos, “High-Speed Parallel-Prefix VLSI Ling Adders,” *IEEE Trans. on Computers*, vol. 54, no. 2, pp. 225–231, 2005.

[26] G. Dimitrakopoulos, K. Papachatzopoulos, and V. Paliouras, “Sum propagate adders,” *IEEE Trans. on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1479–1488, 2021.

[27] C. Liu, J. Han, and F. Lombardi, “An analytical framework for evaluating the error characteristics of approximate adders,” *IEEE Transactions on Computers*, vol. 64, no. 5, pp. 1268–1281, 2014.

[28] USC Viterbi - Signal and Image Processing Institute. USC-SIPI Image Database. [Online]. Available: https://sipi.usc.edu/database/database.php

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” *Advances in neural information processing systems*, vol. 32, 2019.

[30] L. Deng, “The mnist database of handwritten digit images for machine learning research [best of the web],” *IEEE signal processing magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[31] A. G. M. Strollo, E. Napoli, D. De Caro, N. Petra, G. Saggese, and G. Di Meo, “Approximate multipliers using static segmentation: Error analysis and improvements,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 6, pp. 2449–2462, 2022.

[32] M. Ahmadinejad and M. H. Moaiyeri, “Energy-and quality-efficient approximate multipliers for neural network and image processing applications,” *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 2, pp. 1105–1116, 2021.

[33] H. Seo, Y. Yang, and Y. Kim, “Design and Analysis of an Approximate Adder with Hybrid Error Reduction,” *Electronics*, vol. 9, p. 471, 03 2020.

[34] T. Ban, B. Wang, and L. Naviner, “Design, synthesis and application of a novel approximate adder,” in *International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2018, pp. 488–491.

[35] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, “Approximate XOR/XNOR-based Adders for Inexact Computing,” in *International Conference on Nanotechnology (IEEE-NANO)*, 2013, pp. 690–693.

[36] D. Nanu, P. Roshini, D. Sowkarthiga, and K. S. A. Ameen, “Approximate Adder Design Using CPL Logic for Image Compression,” *International journal of innovative research and development*, 2014.

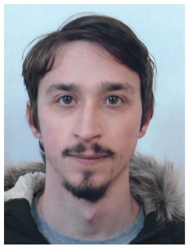
[37] S.-L. Lu, “Speeding Up Processing with Approximation Circuits,” *Computer*, vol. 37, no. 3, pp. 67–73, 2004.

[38] A. K. Verma, P. Brisk, and P. Ienne, “Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design,” in *Design, Automation and Test in Europe (DATE)*, 2008, pp. 1250–1255.

[39] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, “Design of Voltage-Scalable Meta-Functions for Approximate Computing,” in *Design, Automation and Test in Europe (DATE)*, 2011, pp. 1–6.

[40] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo, and Z. H. Kong, “Design of Low-Power High-Speed Truncation-Error-Tolerant Adder and Its Application in Digital Signal Processing,” *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 8, pp. 1225–1229, 2010.

[41] N. Zhu, W. Goh, and K. S. Yeo, “An Enhanced Low-Power High-Speed Adder for Error-Tolerant Application,” in *Intern. Symp. on Integrated Circuits (ISIC)*, 2010, pp. 69–72.
[42] N. Zhu, W. L. Goh, and K. S. Yeo, “Ultra Low-Power High-Speed Flexible Probabilistic Adder for Error-Tolerant Applications,” in *Intern. SoC Design Conference*, 2011, pp. 393–396.
[43] H. Seo, Y. S. Yang, and Y. Kim, “An energy-efficient imprecise adder with a lower-part constant approximation,” in *International SoC Design Conference (ISOCC)*, 2020, pp. 143–144.
[44] A. B. Kahng and S. Kang, “Accuracy-Configurable Adder for Approximate Arithmetic Designs,” in *Design Automation Conference (DAC)*, 2012, pp. 820–825.
[45] K. Du, P. Varman, and K. Mohanram, “High Performance Reliable Variable Latency Carry Select Addition,” in *Design, Automation Test in Europe (DATE)*, 2012, pp. 1257–1262.
[46] Y. Kim, Y. Zhang, and P. Li, “An Energy Efficient Approximate Adder with Carry Skip for Error Resilient Neuromorphic VLSI Systems,” in *Intern. Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 130–137.
[47] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, “On Reconfiguration-Oriented Approximate Adder Design and Its Application,” in *Intern. Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 48–54.
[48] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, “A low latency generic accuracy configurable adder,” in *Design Automation Conference (DAC)*, 2015, pp. 1–6.
[49] I.-C. Lin, Y.-M. Yang, and C.-C. Lin, “High-Performance Low-Power Carry Speculative Addition with Variable Latency,” *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 9, pp. 1591–1603, 2015.
[50] L. Li and H. Zhou, “On Error Modeling and Analysis of Approximate Adders,” in *Intern. Conference on Computer-Aided Design (ICCAD)*, 2014, pp. 511–518.
[51] V. Camus, J. Schlachter, and C. Enz, “A Low-Power Carry Cut-Back Approximate Adder with Fixed-Point Implementation and Floating-Point Precision,” in *Design Automation Conference (DAC)*, 2016, pp. 1–6.
[52] W. Xu, S. S. Sapatnekar, and J. Hu, “A simple yet efficient accuracy-configurable adder design,” *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 6, pp. 1112–1125, 2018.
[53] F. Ebrahimi-Azandaryani, O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram, “Block-Based Carry Speculative Approximate Adder for Energy-Efficient Applications,” *IEEE Trans. on Circuits and Systems II*, vol. 67, no. 1, pp. 137–141, 2020.
[54] K. Papachatzopoulos and V. Paliouras, “Path-based delay variation models for parallel-prefix adders,” *IEEE Transactions on Emerging Topics in Computing*, 2023.



Apostolos Stefanidis received the Diploma in Electrical and Computer Engineering from the Democritus University of Thrace, Xanthi, Greece, in 2017, where he is currently pursuing his Ph.D. degree.

His current research interests include electronic design automation, with emphasis in machine learning applications on autonomous timing and power optimization.

Ioanna Zoumpoulidou received the Diploma in Electrical and Computer engineering from the Democritus University of Thrace, Xanthi, Greece, in 2022.

Her current research interests include electronic design automation, with high-level synthesis technologies.

Dionysios Filippas received the Diploma degree in electrical and computer engineering and the M.Sc. degree in computer engineering from Democritus University of Thrace, Xanthi, Greece, in 2019 and 2021, respectively, where he is currently working toward the Ph.D. degree.

His research interests include energy-efficient machine-learning accelerators, high-level synthesis, floating-point arithmetic and fault-tolerant systems.

Giorgos Dimitrakopoulos received the B.S., M.Sc., and Ph.D. degrees in Computer Engineering from the University of Patras, Patras, Greece, in 2001, 2003, and 2007, respectively.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece. He is interested in the design of digital integrated circuits, energy-efficient data-parallel accelerators, functional safety architectures, and the use of high-level synthesis for agile chip design.

He received two Best Paper Awards at the Design Automation and Test in Europe (DATE) Conference in 2015 and 2019, respectively. Also, he received the HIPEAC Technology Transfer Award in 2015.

Georgios Ch. Sirakoulis (Member, IEEE) received the Dipl.Eng. and Ph.D. degrees in electrical and computer engineering from the Democritus University of Thrace (DUTh), Greece, in 1996 and 2001, respectively. He is a Professor and the Head of the Department of Electrical and Computer Engineering, DUTh. He has published more than 340 technical papers. He is the co-editor/co-author of nine books, the coauthor of 28 book chapters, and a guest editor of 15 special issues. His current research interests include complex electronic systems, future and emergent electronic devices, circuits, models, and architectures, unconventional computing memristors, cellular automata, quantum cellular automata, bioinspired computation/biocomputation and bioengineering, and modeling and simulation. He is an Editor of IEEE TRANSACTIONS ON NANOTECHNOLOGY, IEEE NANOTECHNOLOGY MAGAZINE, Microelectronics, Scientific Reports, Plos ONE, Microelectronics Journal and Integration, VLSI Journal, Journal of Cellular Automata, International Journal of Unconventional Computing, Parallel Processing Letters, etc.