Multiplierless Design of Very Large Constant Multiplications in Cryptography

Levent Aksoy, *Member, IEEE,* Debapriya Basu Roy, *Member, IEEE,* Malik Imran, *Student Member, IEEE,* Patrick Karl, and Samuel Pagliarini, *Member, IEEE*

Abstract—This brief addresses the problem of implementing very large constant multiplications by a single variable under the shift-adds architecture using a minimum number of adders/subtractors. Due to the intrinsic complexity of the problem, we introduce an approximate algorithm, called TÕLL, which partitions the very large constants into smaller ones. To reduce the number of operations, TÕLL incorporates graph-based and common subexpression elimination methods proposed for the shift-adds design of constant multiplications. It can also consider the delay of a multiplierless design defined in terms of the maximum number of operations in series, i.e., the number of adder-steps, while reducing the number of operations. High-level experimental results show that the adder-steps of a shift-adds design can be reduced significantly with a little overhead in the number of operations. Gate-level experimental results indicate that while the shift-adds design can lead to a 36.6% reduction in gate-level area with respect to a design using a multiplier, the delay-aware optimization can yield a 48.3% reduction in minimum achievable delay of the shift-adds design when compared to the area-aware optimization.

Index Terms—very large constant multiplication, shift-adds design, graph-based algorithms, common subexpression elimination, delay-aware optimization, cryptography.

I. INTRODUCTION

Multiplication of constant(s) by a variable is a ubiquitous operation in many applications, such as digital signal processing and cryptography. Since constants are determined beforehand in these applications and the implementation of a multiplier in hardware is expensive in terms of area and power consumption, the constant multiplication can be realized under the shift-adds architecture using only shifts and adders/subtractors [1]. Note that shifts by a constant value can be realized using only wires which represent no hardware cost. In cryptographic algorithms, such as elliptic curve cryptography (ECC) [2], [3] and supersingular isogeny key encapsulation (SIKE) [4], [5], prime numbers to be multiplied by a variable can respectively be 204-521 bits and 448-768 bits long due to security requirements. The parallel realization of

L. Aksoy, M. Imran, and S. Pagliarini are with the Department of Computer Systems, Centre for Hardware Security, Tallinn University of Technology, Tallinn, Estonia (e-mail: {levent.aksoy, malik.imran, and samuel.pagliarini}@taltech.ee.)

D. B. Roy and P. Karl are with the Technical University of Munich, Department of Electrical and Computer Engineering, Chair for Security in Information Technology, Munich, Germany (e-mail: {debapriya.basu-roy and patrick.karl}@tum.de.)

such constant multiplications is required for high-performance cryptographic designs [2]. Thus, the *very large constant multiplication* (VLCM) problem is defined as finding a minimum number of adders/subtractors which realize the multiplication of given very large constants by a variable. Similar to [6], this problem is NP-complete.

Techniques under the residue residue number system [7]–[9], that enable large constant multiplications to be realized using a set of small constant multiplications, have been introduced, but they require the logic for conversions between binary and residue number system. Many large integer multiplication architectures [10]–[12] have also been proposed, but both operands in these architectures are assumed to be variable. Moreover, prominent algorithms [13]–[16] have been developed for the shift-adds design of constant multiplications, but they are limited with the bit-width of constants. Furthermore, the VLCM problem has not been studied thoroughly. Hence, we introduce **the first approximate algorithm** TÕLL **proposed for the VLCM problem**, which is the main contribution of this brief. TÕLL divides the very large constants into small coefficients with a reasonable bit-width and re-defines these very large constants as linear equations in the form of summation of shifted versions of these small coefficients. It finds common partial products in a shift-adds design of these small coefficient multiplications using a prominent graph-based (GB) algorithm [14], [15]. It extracts common subexpressions among the linear equations using an efficient common subexpression elimination (CSE) algorithm [17], [18]. The performance of a design can be more critical than other characteristics and thus, an increase in area and power consumption can be compromised to meet the performance criterion. Hence, TÕLL can also consider the maximum number of operations in series, called the number of adder-steps, while reducing the number of operations. Experimental results show that shift-adds designs obtained by TÕLL have significantly less hardware complexity than those including generic multipliers and compressor trees, and delay-aware optimization leads to a significant reduction in minimum delay of a design with respect to area-aware optimization.

The remainder of this brief is organized as follows: Section II introduces background concepts. TÕLL is described in detail in Section III. Experimental results are given in Section IV. Finally, Section V concludes the brief.

II. BACKGROUND

This section presents background concepts on the shift-adds design of constant multiplications. Since constants are mul-

tiplied by a common variable, the realization of constant multiplications corresponds to the realization of constants. For example, $3x = x \ll 1 + x = (1 \ll 1 + 1)x$ can be rewritten as $3 = 1 \ll 1 + 1$ by eliminating the variable x from both sides. These notations will be used interchangeably in this brief.

The straightforward digit-based recoding (DBR) technique [19] realizes the shift-adds design of constant multiplications in two steps: (i) define the constants under a particular number representation, e.g., binary or canonical signed digit (CSD)[1] [17]; (ii) for the nonzero digits in the representation of constants, shift the input variable according to digit positions and add/subtract the shifted variables with respect to digit values. Consider the multiple constant multiplication (MCM) block realizing $43x$ and $59x$ as an example. The decompositions of its constants under binary are given as follows:

$$43x = (101011)_{bin}x = x \ll 5 + x \ll 3 + x \ll 1 + x$$
$$59x = (111011)_{bin}x = x \ll 5 + x \ll 4 + x \ll 3 + x \ll 1 + x$$

which lead to a design with 7 operations in 4 adder-steps, as shown in Fig. 1(a).

Algorithms, that aim to maximize the sharing of partial products in the shift-adds design of constant multiplications, can be grouped in two categories based on the search space they explore: (i) The CSE methods [17], [18], [20]–[24] initially define the constants under a number representation. Then, in an iterative fashion, after all possible subexpressions that can be extracted from the nonzero digits in representations of constants, are identified, the "best" subexpression, generally, the most common one, is chosen to be shared among the constant multiplications. The exact CSE algorithm [20] uses a 0-1 integer linear programming (ILP)-based approach to maximize the sharing of subexpressions. (ii) The GB methods [13]–[16], [25]–[27], which are not restricted to any particular number representation, aim to find the "best" intermediate constants, generally, the ones that enable to realize the constant multiplications with a small number of operations. They consider a large number of possible realizations of a constant and obtain better solutions than CSE methods [15]. While the exact GB algorithm of [15] can explore the search space using breadth-first and depth-first search techniques, the exact GB algorithm of [16] uses a 0-1 ILP-based approach.

Returning to our simple MCM example, the exact CSE algorithm [20] finds a solution with 4 operations in 4 adder-steps when constants are defined under binary, sharing the common subexpressions $9x = (1001)_{bin}x$ and $41x = (101001)_{bin}x$ among the constant multiplications as shown in Fig. 1(b). On the other hand, the exact GB algorithm [15] obtains a solution with a minimum number of 3 operations in 3 adder-steps, finding the intermediate constant multiplication $5x$ to realize the constant multiplications as shown in Fig. 1(c).

In a shift-adds design of constant multiplications, the delay is generally defined as the number of adder-steps [28]. Note that the minimum adder-steps of a single constant c is computed as $mas_c = \lceil log_2 NZ(c) \rceil$, where $NZ(c)$ denotes the
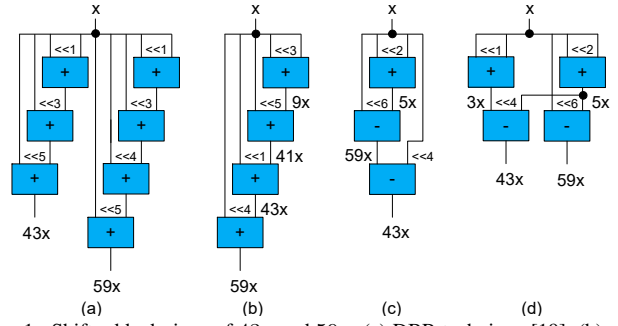

Fig. 1. Shift-adds designs of $43x$ and $59x$: (a) DBR technique [19]; (b) exact CSE method [20]; (c) exact GB method [15]; (d) approximate GB method under a delay constraint [30].

number of nonzero digits in the CSD representation of the constant. Thus, given a set of constants $C = \{c_1, c_2, \ldots, c_n\}$, the minimum adder-steps of multiple constants in the set C is computed as $mas_C = \max_{1 \leq i \leq n} \{mas_{c_i}\}$ [28]. There exist efficient CSE and GB algorithms introduced to optimize the number of operations in the multiplierless design where the delay constraint given in terms of the number of adder-steps is never violated [20], [28]–[30]. Returning to our example, Fig. 1(d) shows the realization of constant multiplications with a minimum number of adder-steps, i.e., 2, whose solution is obtained by the GB algorithm of [30] using 4 operations.

The proposed algorithms, except the DBR technique, are limited with the size of constants. This is simply because the number of possible partial products of a constant increases dramatically as its bit-width increases [15]. For example, the exact GB algorithm [13] developed for the shift-adds design of a single constant multiplication can handle a constant up to 32 bits. The approximate [14] and exact [15] GB algorithms can handle multiple constants up to 31 and 16 bits, respectively.

III. TÕLL - THE PROPOSED METHOD

TÕLL takes n large constants, i.e., $lc_1, lc_2, \ldots, lc_n$, in hexadecimal format and the number of bits in partition, i.e., p, as inputs and returns the multiplication of these large constants by an input variable under the shift-adds architecture described in Verilog as an output. Due to the limitations of algorithms proposed for the multiplierless design on the size of constants, the value of p is determined to be a multiple of 4 with a minimum and maximum value of 4 and 28, respectively. It initially partitions the large constants into p-bit coefficients[2] and defines each large constant as the summation of shifted p-bit coefficients, called a linear equation. Then, it applies a GB algorithm [14], [15] to these coefficients to find their multiplierless realization. Finally, it uses a CSE heuristic [17], [18] to extract common subexpressions in the linear equations and realizes the final linear equations using two-term subexpressions. It includes three stages: (i) partitioning; (ii) realization of coefficients; and (iii) realization of linear equations. It can also consider the delay of the multiplierless design while reducing the number of operations. In following, its stages are described under the area-aware optimization. Finally, details in the delay-aware optimization are given.

---

[1]An integer can be written in CSD using k digits as $\sum_{i=0}^{k-1} d_i 2^i$, where $d_i \in \{1, 0, -1\}$ with $0 \leq i \leq n - 1$. Under CSD, nonzero digits are not adjacent and a minimum number of nonzero digits is used.

[2]Partitioning of a k-bit large constant lc into p-bit coefficients can be written as $\sum_{i=1}^{\lceil k/p \rceil} lc[ip - 1 : (i-1)p]2^{(i-1)p} = \sum_{i=1}^{\lceil k/p \rceil} c_i 2^{(i-1)p}$.
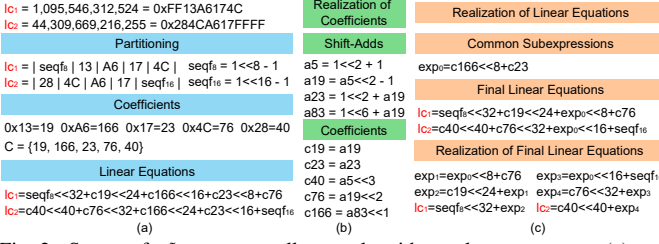
Fig. 2. Stages of TÕLL on a small example with two large constants: (a) partitioning; (b) realization of coefficients; (c) realization of linear equations.
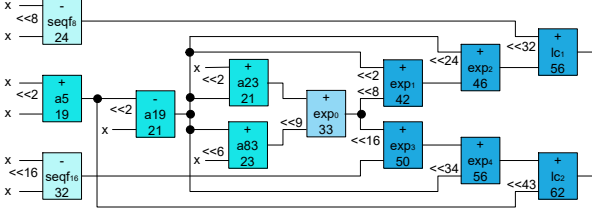

Fig. 3. Multiplierless realization of large constant multiplications in Fig. 2.

Partitioning: In TÕLL, two partitioning strategies are implemented. In the first one, called the *strict* partitioning, starting from the least significant bit, p-bit coefficients are generated from the hexadecimal digits of each large constant and stored as integers in set C without repetition. Shift values of these coefficients are computed based on the locations of hexadecimal digits and stored in set S. While partitioning large constants into p-bit coefficients, sequences of r 0s, where $r \geq p$ and $r \mod p = 0$, are found and ignored, since such a sequence requires no operations. Also, sequences of r 1s, where $r \geq p$ and $r \mod p = 0$, are identified and replaced by a subexpression, denoted as $seqf_r$, which needs only a single subtractor, i.e., $2^r - 1$. These sequence subexpressions are stored in set $Seqf$ without repetition. Finally, the realization of each large constant is written as a linear equation in the form of summation of coefficients in set C based on their shift values in set S and sequence expressions in set $Seqf$ based on their shift values in large constants. Fig. 2(a) shows the steps of the *strict* partitioning strategy when p is 8.

In the second one, called *common digit* partitioning, initially, all possible p-bit coefficients are identified and the ones, which occur more than once, are extracted from the large constant in an order of their number of occurrences iteratively, starting from the greatest one. Then, the remaining digits are divided based on the *strict* partitioning. The *common digit* partitioning aims to increase the sharing of common expressions while realizing the linear equations. For our example, common coefficients $0 \times A6$, 0×17, and $0 \times 4C$ are initially extracted.

Realization of Coefficients: Coefficients in the linear equations of each large constant determined at the partitioning stage and stored in set C are realized under the shift-adds architecture. TÕLL incorporates two prominent GB algorithms [14], [15]. For our example, Fig. 2(b) presents the solution of the exact algorithm [15] on coefficients in set C with 4 operations in 3 adder-steps.

Realization of Linear Equations: In TÕLL, common subexpressions in the linear equations obtained at the partitioning stage are identified and eliminated using a CSE heuristic. The developed CSE method is based on the CSE heuristics of [17], [18]. In this method, all subexpressions with two terms are found considering their shift values, their

number of occurrences is computed, and the one with the maximum number of occurrences greater than 1 is chosen to be eliminated. This process is iterated until there is no subexpression with a maximum number of occurrences greater than 1. For our example, Fig. 2(c) shows the realization of the common subexpression and the final version of linear equations after this subexpresion is eliminated. Then, till the number of coefficients and subexpressions in each final linear equation is less than or equal to 2, in an iterative fashion, two terms having the smallest bit-width value are determined, defined as a subexpression indicating the summation of these terms, and eliminated from the linear equations. The insight behind the selection of coefficients and subexpressions based on their bit-widths is to reduce design area. For our example, Fig. 2(c) also shows the realization of final linear equations which needs 6 operations in 3 adder-steps.

Observe from Fig. 2 that the multiplierless design requires a total number of 13 adders/subtractors, i.e., 2 for the sequence subexpressions, 4 for the shift-adds realization of coefficients, 1 for the common subexpression, and 6 for the realization of final linear equations, in 7 adder-steps. Fig. 3 presents this multiplierless design, where the *top*, *middle*, and *bottom* terms inside operations stand respectively for the operation type, operation output, and bit-width of the operation output, assuming that the input variable x is 16 bits long.

Delay-Aware Optimization: During the delay-aware realization of large constants under the shift-adds architecture, the coefficients of linear equations determined at the partitioning stage are realized using the algorithms of [14], [30] when the delay constraint is set to the mas_C value of multiple coefficients in set C. For our example, the coefficients are implemented using 5 operations in 2 adder-steps using the algorithm of [30]. Moreover, while choosing common subexpressions among the ones with the maximum number of occurrences to be eliminated in the linear equations, the one, that leads to the smallest increase in the number of adder-steps, is preferred. For our example, the subexpression exp_0 of Fig. 2 is also selected during the delay-aware optimization. Lastly, in the realization of final linear equations, the subexpressions are generated with a minimum number of adder-steps considering the bit-width of coefficients and subexpressions. For our example, the final linear equations are realized using 6 operations in 2 adder-steps. As an example, the final linear equation lc_1 is realized as $lc_1 = exp_2 + exp_1$, where $exp_1 = exp_0 << 8 + c76$ and $exp_2 = seqf_8 << 32 + c19 << 24$. We note that for our example, the delay-aware optimization leads to a design with a total number of 14 operations in 5 adder-steps.

In TÕLL, the design and verification process of the multiplierless realization of large constant multiplications is au-

Instance	Prime Width	$p = 8$			$p = 16$			$p = 24$		
		oper	step	time	oper	step	time	oper	step	time
anomalous [3]	220	19	13	4.5	15	12	4.5	15	10	4.8
anssifrp [3]	272	60	33	5.7	43	24	5.6	43	15	6.8
bn(2,254) [3]	268	39	22	5.5	32	16	5.4	29	18	5.7
brainpool256 [3]	268	58	35	5.6	44	27	5.5	41	20	7.3
brainpool348 [3]	400	80	51	8.1	61	33	7.9	54	27	10.4
sike610 [5]	640	69	39	12.4	51	28	12.5	47	24	14.3
sike751 [5]	768	87	50	14.7	62	36	14.7	55	27	17.3

TABLE II
GATE-LEVEL RESULTS OF DESIGNS REALIZING SINGLE PRIME NUMBER MULTIPLICATIONS.

| Instance | Generic Multiplier | | | Compressor Trees | | | Shift-Adds | | | | | | | | |
| | | | | | | | $p = 8$ | | | $p = 16$ | | | $p = 24$ | | |
	area	delay	power	area	delay	power	area	delay	power	area	delay	power	area	delay	power
anomalous [3]	3477	9846	849	4367	4742	161	2516	4234	677	2202	4480	606	2530	5922	823
anssifrp [3]	9525	23215	2630	12339	27401	526	8082	31183	2441	7988	25890	2524	9506	24351	2998
bn(2,254) [3]	6297	22150	1537	8585	11266	322	4929	9460	1335	4968	10646	1505	4957	10381	1474
brainpool256 [3]	10292	22956	2815	12356	27615	540	8268	32319	2583	8273	27506	2738	8779	23264	2906
brainpool348 [3]	14539	33738	4010	17184	37916	759	10942	43181	3367	11095	36695	3608	11280	31180	3777
sike610 [5]	11597	32475	3117	14217	32933	609	10476	33891	3163	10751	29473	3652	11562	31862	4100
sike751 [5]	14670	40180	4037	17636	38633	786	12757	40163	3974	12649	35587	4137	14453	39910	5255

TABLE III
GATE-LEVEL RESULTS OF DESIGNS WITH MINIMUM ACHIEVABLE DELAY.

| Instance | Area-Aware Optimization | | | Delay-Aware Optimization | | | | |
	area	delay	power	oper	step	area	delay	power
anomalous [3]	6296	1135	1304	17	6	5952	1116	1317
anssifrp [3]	22488	1982	5786	52	8	22696	1403	5152
bn(2,254) [3]	15069	1582	3868	34	7	12597	1258	2885
brainpool256 [3]	22937	2396	6487	49	8	21385	1404	5312
brainpool348 [3]	29394	2729	7889	69	8	30440	1410	6714
sike610 [5]	27444	2534	8324	64	8	29243	1528	7601
sike751 [5]	29736	3643	8355	71	8	33159	2167	8899

tomated. TÕLL can generate the behavioral description of the design in Verilog and the associated testbench for verification. It can also describe the large constant multiplications using multipliers in Verilog. TÕLL is available at *https://github.com/leventaksoy/vlcm*.

IV. EXPERIMENTAL RESULTS

As the first experiment set, the well-known cryptographic prime numbers are taken from [3], [5] and the related constants to be multiplied by a variable are computed. Table I presents these instances, each of which includes a single prime number, i.e., n is 1, and their bit-width values. Note that these elliptic curves are chosen because the underlying primes do not have any special form. Other elliptic curves, such as *Curve25519* and *NIST Curves*, are based on either pseudo-Mersenne or Solinas primes where modular reductions are performed using a small number of adders/subtractors [3]. Hence, modular reduction in elliptic curves given in Table I are performed using Montgomery reduction that involves constant multiplication. The results shown in this work focus only on the constant multiplication of Montgomery reduction, not the entire Montgomery reduction. Table I also shows the high-level results of shift-adds designs, where *oper*, *step*, and *time* denote the number of operations, the number of adder-steps, and the run-time of TÕLL in seconds, respectively. These results were obtained when the *strict* partitioning strategy is chosen, the area-aware optimization is used, the approximate GB

algorithm [14] is selected for the shift-adds realization of coefficients, and the bit-width of the input variable is 16. Note that TÕLL was run on a PC including an Intel Core i5-10600K processing unit at 4.1 GHz with 16 GB memory.

Observe from Table I that the use of a high p value leads to a shift-adds design with a small number of operations and the multiplierless realizations are obtained in a reasonable time.

Table II presents the gate-level results of designs realizing prime number multiplications using a generic multiplier, compressor trees, and adders/subtractors under the shift-adds architecture. In this table, *area*, *delay*, and *power* stand for the total area in μm^2, delay in the critical path in ps, and total power dissipation in μW, respectively. Logic synthesis was performed by Cadence Genus using a commercial 65 nm cell library without a strict delay constraint aiming for area optimization. Designs are validated using 10,000 randomly generated inputs in simulation.

Observe from Table II that the shift-adds designs occupy less area when compared to those using a generic multiplier and compressor trees. The gain in area on the shift-adds design with respect to the one using a generic multiplier (compressor trees) reaches up to 36.6% (49.5%) on the *anomalous* instance when p is 16. Note also that as p increases, the hardware complexity of the shift-adds design tends to increase, although there are designs obtained when p is 16 (or 24) with less area when compared to those obtained when p is 8 (or 16). This is because as p increases, the sizes of operations, which have an impact on the hardware complexity, are increased.

In order to show the impact of the optimization techniques on the minimum achievable delay, the shift-adds designs obtained under the area- and delay-aware optimization when p is 16 are synthesized with timing constraints changed in a binary search manner till the minimum delay in the critical path is found without a negative slack. The initial lower and upper bounds of the timing constraint are taken as 0 and 80 ns, respectively. Table III shows the gate-level results of designs
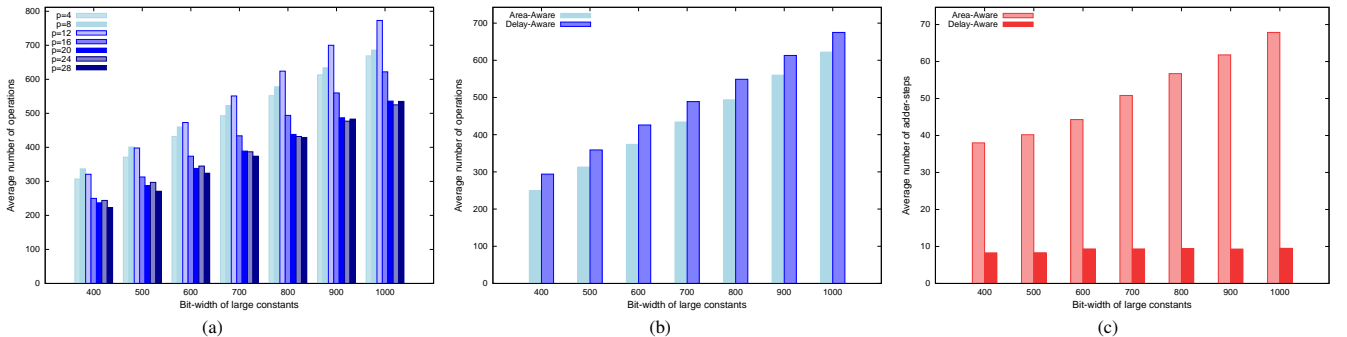

Fig. 4. Results on randomly generated instances when n is 5: (a) impact of p on the number of operations; (b)-(c) impact of the optimization technique on the number of operations and adder-steps when p is 16.

with the minimum achievable delay and also, the high-level results of designs when the delay-aware optimization is used.

Observe from Tables I and III that the delay-aware optimization yields significant reduction in the number of adder-steps with an increase in the number of operations. Thus, the designs obtained using the delay-aware optimization have significantly improved delay over those obtained using the area-aware optimization, reaching up to a 48.3% reduction on the *brainpool348* instance. However, those designs have larger area than the ones obtained under the area-aware optimization, e.g., the *anssifrp* and *brainpool348* instances. It is also observed from the results obtained during the binary search of minimum delay that the delay-aware optimization can generate designs with less area and delay when compared to the design obtained under the area-aware optimization with the minimum delay, e.g., the *bn(2,254)*, and *brainpool256* instances.

As the second experiment set, we used randomly generated multiple constants whose bit-width ranges from 400 to 1000 in a step of 100, i.e., a total of 7 categories. We generated 30 instances for the same bit-width of constants when the number of constants, i.e., n, is 5, a total of $7 \times 30 = 210$ instances. In this experiment, the algorithm of [14] is used to realize the multiplierless design of coefficients and the *strict* partitioning strategy is chosen.

Fig. 4(a) presents the average number of operations obtained by TŌLL when the area-aware optimization is considered. Observe that the use of a high p value for partitioning the hexadecimal digits decreases the required number of operations, simply because it reduces the number of terms in linear equations. Interestingly, less number of operations can be obtained when p is decreased, because the number of common subexpressions in the linear equations is increased in this case. Although it is clear that an increase in p decreases the required number of operations, the prominent GB algorithms [14], [15] are limited with the size of coefficients and the reduction of the number of operations does not always lead to a design with a small area as shown in Tables I and II.

Figs. 4(b)-(c) show the average number of operations and adder-steps obtained by TŌLL when the area- and delay-aware optimizations are used and p is 16. Note that the delay-aware optimization can reduce the number of adder-steps of a shift-adds design significantly, but with an increase in the number of operations. Note that while the maximum increase in the number of operations is $1.17\times$, the maximum decrease in the number of adder-steps is $7.14\times$ in the delay-aware optimization with respect to the area-aware optimization.

V. Conclusions

This brief introduced TŌLL, the first approximate algorithm proposed for the VLCM problem. Our method is equipped with both area and delay optimization techniques, including previously proposed algorithms used to reduce the number of operations and adder-steps of a shift-adds design. Experimental results clearly indicated that TŌLL can lead to a significant reduction on the circuit area when compared to that of a design with a multiplier or compressor trees. It can also generate alternative designs which may help a designer to choose the best fit for the design requirements in a given application.

References

[1] H. Nguyen and A. Chatterjee, "Number-Splitting with Shift-and-Add Decomposition for Power and Hardware Optimization in Linear DSP Synthesis," *IEEE TVLSI*, vol. 8, no. 4, pp. 419–424, 2000.

[2] D. B. Roy and D. Mukhopadhyay, "High-Speed Implementation of ECC Scalar Multiplication in GF(p) for Generic Montgomery Curves," *IEEE TVLSI*, vol. 27, no. 7, pp. 1587–1600, 2019.

[3] D. J. Bernstein and T. Lange. SafeCurves: Choosing Safe Curves for ECC. [Online]. Available: https://safecurves.cr.yp.to

[4] D. B. Roy, T. Fritzmann, and G. Sigl, "Efficient Hardware/Software Co-Design for Post-Quantum Crypto Algorithm SIKE on ARM and RISC-V based Microcontrollers," in *ICCAD*, 2020, pp. 1–9.

[5] D. Jao. Supersingular Isogeny Key Encapsulation. [Online]. Available: https://sike.org/files/SIDH-spec.pdf

[6] P. Cappello and K. Steiglitz, "Some Complexity Issues in Digital Signal Processing," *IEEE Tran. on Acoustics, Speech, and Signal Processing*, vol. 32, no. 5, pp. 1037–1041, 1984.

[7] R. Chaves and L. Sousa, "Improving Residue Number System Multiplication with More Balanced Moduli Sets and Enhanced Modular Arithmetic Structures," *IET*, vol. 1, no. 5, pp. 472–480, 2007.

[8] J. Y. S. Low and C.-H. Chang, "A New Approach to the Design of Efficient Residue Generators for Arbitrary Moduli," *IEEE TCAS*, vol. 60, no. 9, pp. 2366–2374, 2013.

[9] P. Patronik and S. J. Piestrak, "Design of Residue Generators with CLA/Compressor Trees and Multi-Bit EAC," in *LASCAS*, 2017, pp. 1–4.

[10] A. A. Karatsuba and Y. Ofman, "Multiplication of Multidigit Numbers on Automata," *Soviet Physics Doklady*, vol. 7, pp. 595–596, 1963.

[11] P. L. Montgomery, "Modular Multiplication without Trial Division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[12] C. Rafferty, M. O'Neill, and N. Hanley, "Evaluation of Large Integer Multiplication Methods on Hardware," *IEEE Tran. on Computers*, vol. 66, no. 8, pp. 1369–1382, 2017.

[13] J. Thong and N. Nicolici, "A Novel Optimal Single Constant Multiplication Algorithm," in *DAC*, 2010, pp. 613–616.

[14] Y. Voronenko and M. Püschel, "Multiplierless Multiple Constant Multiplication," *ACM Tran. on Algorithms*, vol. 3, no. 2, 2007.

[15] L. Aksoy, E. Gunes, and P. Flores, "Search Algorithms for the Multiple Constant Multiplications Problem: Exact and Approximate," *Elsevier MICPRO*, vol. 34, no. 5, pp. 151–162, 2010.

[16] M. Kumm, "Optimal Constant Multiplication Using Integer Linear Programming," *IEEE TCAS II*, vol. 65, no. 5, pp. 567–571, 2018.

[17] R. Hartley, "Subexpression Sharing in Filters Using Canonic Signed Digit Multipliers," *IEEE TCAS II*, vol. 43, no. 10, pp. 677–688, 1996.

[18] A. Hosangadi, F. Fallah, and R. Kastner, "Reducing Hardware Complexity of Linear DSP Systems by Iteratively Eliminating Two-Term Common Subexpressions," in *ASP-DAC*, 2005, pp. 523–528.

[19] M. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann, 2003.

[20] L. Aksoy *et al.*, "Exact and Approximate Algorithms for the Optimization of Area and Delay in Multiple Constant Multiplications," *IEEE TCAD*, vol. 27, no. 6, pp. 1013–1026, 2008.

[21] Y.-H. Ho *et al.*, "Global Optimization of Common Subexpressions for Multiplierless Synthesis of Multiple Constant Multiplications," in *ASP-DAC*, 2008, pp. 119–124.

[22] V. Lefevre, "Multiplication by an Integer Constant," Institut National de Recherche en Informatique et en Automatique, Tech. Rep., 2001.

[23] I.-C. Park and H.-J. Kang, "Digital Filter Synthesis Based on Minimal Signed Digit Representation," in *DAC*, 2001, pp. 468–473.

[24] M. Potkonjak, M. Srivastava, and A. Chandrakasan, "Multiple Constant Multiplications: Efficient and Versatile Framework and Algorithms for Exploring Common Subexpression Elimination," *IEEE TCAD*, vol. 15, no. 2, pp. 151–165, 1996.

[25] A. Dempster and M. Macleod, "Constant Integer Multiplication Using Minimum Adders," *IEE Proc. - Circuits, Devices and Systems*, vol. 141, no. 5, pp. 407–413, 1994.

[26] ——, "Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters," *IEEE TCAS II*, vol. 42, no. 9, pp. 569–577, 1995.

[27] O. Gustafsson, "A Difference Based Adder Graph Heuristic for Multiple Constant Multiplication Problems," in *ISCAS*, 2007, pp. 1097–1100.

[28] H.-J. Kang, H. Kim, and I.-C. Park, "FIR Filter Synthesis Algorithms for Minimizing the Delay and the Number of Adders," in *ICCAD*, 2000, pp. 51–54.

[29] A. Dempster, S. Demirsoy, and I. Kale, "Designing Multiplier Blocks With Low Logic Depth," in *ISCAS*, 2002, pp. 773–776.

[30] L. Aksoy *et al.*, "Optimization of Area and Delay at Gate-Level in Multiple Constant Multiplications," in *DSD*, 2010, pp. 3–10.