Evaluation of the Back-End Design Overhead for ASIC Implementations of Large-Operand Multipliers Targeting Resource-Constrained Environments

Christoph Nagl[1,2], Michael Muehlberghuber[1], and Frank K. Gürkaynak[1]

[1] Integrated Systems Laboratory, ETH Zurich,
Gloriastrasse 35, 8092 Zurich, Switzerland
Email: cnagl@student.ethz.ch, {mbgh, kgf}@iis.ee.ethz.ch

[2] Institute for Applied Information Processing and Communications,
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria

***Abstract*—We investigate different ASIC hardware designs of binary-field multipliers with large operand sizes (>1000 bits). Our architectures target resource-constrained environments such as embedded smart cards or contactless-powered devices and evaluate their applicability. We present several different design strategies based on an iterative Karatsuba, a digit-serial, and a bit-serial multiplier. Based on a cryptographic pairing example application, a total of seven architectures are evaluated in terms of area requirement, speed, and power consumption. Each architecture was subject to a full back-end design flow providing detailed results for synthesis, post-layout, and for an "effective chip" giving realistic design-cost numbers. Providing detailed design flow results, we show that basically all designs exhibit a significant increase in area during their back-end design phase, which can certainly not be considered as negligible.**

***Keywords*—Large-operand multipliers, low-resource design, digit-serial, iterative Karatsuba, ASIP, Eta pairing.**

I. Introduction

Finite-field arithmetic represents one of the core-components of numerous cryptographic primitives, including public-key [1], elliptic-curve [2], [3], and pairing-based cryptography [4], [5]. Since in many security-related protocols, a significant amount of computation time is spent for finite-field operations, dedicated hardware architectures and accelerators for all kinds of applications have been proposed.

Due to its suitability for hardware implementations, the binary field is one of the favored choices of designers to serve as the underlying field for elliptic-curve-based cryptography. Although hardware accelerators can speed up implementations significantly, processing large operands still occupies a major part of many protocols' computation time. Especially for pairing-based architectures, the operand sizes of the respective finite fields grow up to 512 bits and far beyond and thereby contribute distinctly to the runtime of the whole system.

In this work, we present an application-specific instruction-set processor (ASIP), suitable to compute algorithms based on elliptic-curves over large binary finite fields. We evaluate two different types of base-field multipliers, optimized for low area. First, an iteratively-constructed Karatsuba multiplier [6] and second, a digit-serial multiplier based on the least-significant digit (LSD) first approach [7] are investigated. Moreover, we compare two Karatsuba multiplier designs based on different recursion depths with LSD designs of varying digit sizes and a least-significant bit (LSB) first multiplier in terms of speed and area. We present back-annotated results for seven different architectures including the required chip area, runtime, and energy consumption based on a 180 nm CMOS design flow.

Throughout the last decade, cryptographic pairing based protocols have emerged in literature, which allow the implementation of entirely new security services, including identity-based encryption [4], short signatures [5], and identity-based signatures [8]. Since the pairing computation is usually the most time-consuming part within these protocols, the need for hardware accelerators is obvious. While a number of hardware architectures targeting high-throughput implementations have been reported so far (e.g., [9]–[11]), only little attention has been paid to the development of pairing-accelerators in the field of resource-constrained environments such as embedded systems or smart cards. In order to explore the applicability of large operand arithmetic architectures in such environments all the way through until a successfully accomplished back-end design, we compute the η_T pairing [12] based on supersingular curves over the binary finite field $\mathbb{F}_{2^{1223}}$.[1] The presented architecture and multiplier variants can also be used for other cryptographic applications, where an arithmetic over large binary finite fields is required. We show that the resulting area overheads due to the late design steps of a digital VLSI circuit are significantly and therefore, should always be taken into consideration during development.

The paper is organized as follows. Section II provides an overview of related work. In Section III, basic preliminaries about the Karatsuba and digit-serial multiplication are given. Section IV introduces the proposed ASIP and discusses the architectures of the applied arithmetic units. In Section V, we present synthesis as well as post-layout results of our designs. Finally, we conclude our work in Section VI.

II. Previous Work

The efficiency of a finite-field arithmetic often depends heavily on the implementation of its multiplier. In general,

[1] We would like to note that recent advances in solving the discrete logarithm problem for finite fields of small characteristics reduce the initially assumed security level of the investigated pairing significantly and can therefore no longer be considered as secure. Nevertheless, we use it as an example application for our large-operand multipliers in this work.

978-1-4799-7244-9/14/$31.00 © 2014 IEEE

area cost of fully parallel multipliers scales quadratically with operand size. As a result, they are not feasible for large operand sizes. To implement large-operand multipliers, hybrid serial/parallel or fully serial multiplier designs can be used to reduce cost in terms of area. Karatsuba's fast multiplication algorithm applies the concept of *divide and conquer* to split a multiplication into smaller sized sub-multiplications [6]. The algorithm re-uses partial products and exploits the fact that the addition operation is generally less complex. Its sub-quadratic complexity of $O(n^{1.58})$ improves multiplication especially for large operands. The work in [13] demonstrates an iterative application of Karatsuba's algorithm in order to leverage circuit re-use for area reduction. While the iterative approach allows to balance area and computation time, it also demands for additional circuitry.

In [7], Song et al. presented a hybrid modular multiplier approach which is inherently scalable. By selecting digit size D, the design can be trimmed for low area or low computation time. The digit-serial multiplier performs the modular reduction for each digit multiplication step in an interleaved manner. As a consequence, a smaller register can be used for the intermediate multiplication results.

Ghosh et al. [10] presented an efficient FPGA implementation multiplying 1223-bit operands based on a two-step 306-bit Karatsuba multiplier taking nine cycles for a single base-field multiplication. Their design is able to compute an η_T pairing within 190 μs on a Virtex 6 FPGA consuming 15 167 slices and 54 681 LUTs. Adikari et al. [11] proposed a crypto-processor for computing the η_T pairing on supersingular elliptic curves over $\mathbb{F}_{2^{1223}}$ targeting both FPGA and ASIC platforms. Best-case corner synthesis results for a 65 nm CMOS technology show that their design requires approximately 500 kGE and calculates a single pairing within 80.6 µs. As opposed to our presented results, previous work on large-operand binary-fields multipliers focused primarily on high-throughput rather than low area or low power implementation.

III. Preliminaries

The multipliers presented in this work are benchmarked by computing the η_T pairing based on the supersingular elliptic curve $E : y^2 + y = x^3 + x + b$. We can compute the full pairing by providing the finite-field operations of modular multiplication, squaring, inversion, addition, and an XOR operation. The elements of $\mathbb{F}_{2^{1223}}$ are represented in polynomial basis and all operations are performed in $\mathbb{F}_{2^{1223}}$ modulo the irreducible trinomial $f(x) = x^{1223} + x^{255} + 1$. The applied algorithm uses tower extensions for elements of $\mathbb{F}_{2^{2m}}$ and $\mathbb{F}_{2^{4m}}$. Hence, arithmetic of these elements can be computed in the base field, using the quadratic extension of $D = d_0 + d_1 s \in \mathbb{F}_{2^{2m}}$ and quartic extension of $G = g_0 + g_1 s + g_2 t + g_3 st \in \mathbb{F}_{2^{4m}}$. In order to obtain a low-area implementation, we apply Fermat's little theorem to compute the finite-field inversion [14]. Consequently, the cost of a field inversion in $\mathbb{F}_{2^{1223}}$ is 15 multiplications and 1222 squarings. With this approach we do not require additional hardware for inversion and can compute the inversion by means of squarings and multiplications. The results presented in this paper are generally considering the full pairing including the final exponentiation step. The applied version of the pairing requires 4321 multiplications, 8579 squarings, and 15 952 additions to compute. For all presented architectures the multiplication is the most expensive computation in terms of both area and time.

A. Karatsuba Multiplier

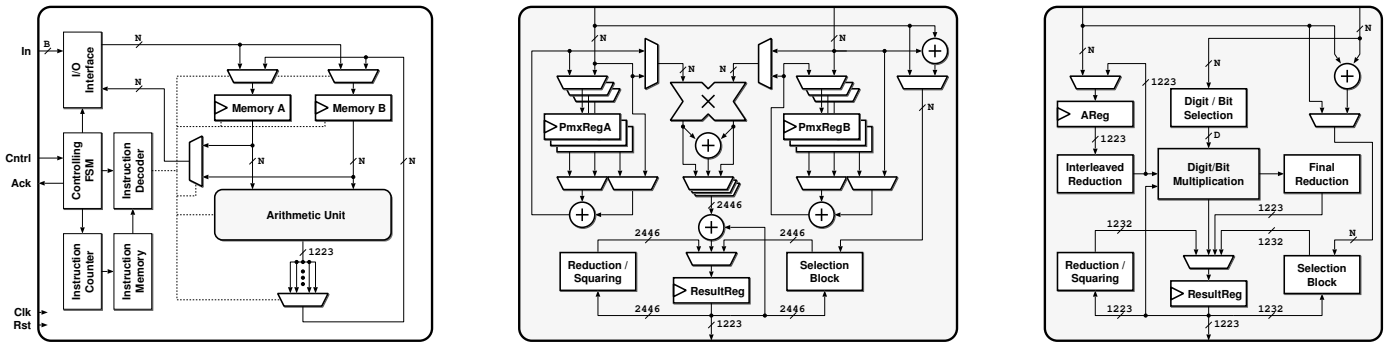
Karatsuba's algorithm [6] is a fast multiplication algorithm, which applies the concept of *divide and conquer*. By splitting the multiplication into sub-multiplications, it allows to use a multiplier of smaller size and to benefit from reusing partial products. In contrast to the classic multiplication, Karatsuba's algorithm requires just three multiplications, four additions, and two subtractions. As the addition operation is generally less expensive than a multiplication, saving a multiplication for extra addition operations provides asymptotically better performance. Let A and B be two elements in $\mathbb{F}_{2^m}$. Using a polynomial representation, A and B can be described by $A = \sum_{i=0}^{m-1} a_i x^i = A^H x^{m/2} + A^L$ and $B = \sum_{i=0}^{m-1} b_i x^i = B^H x^{m/2} + B^L$, respectively [15]. Based on this representation, we can write a single-step multiplication according to Karatsuba as:

$$C = A^H B^H x^m + (A^H B^H + A^L B^L + (A^H + A^L)(B^L + B^H))x^{m/2} + A^L B^L.$$

Repeated application allows to reduce the multiplier width at the expense of more partial products and additions. As there are three multiplications per applied iteration, 3^N partial products of width $\lceil m/(2^N) \rceil$ need to be computed for an N-step application. The base level multiplication is performed using a fully parallel multiplier circuit. As fully parallel multiplier circuits are generally large, we try to trade area for computation time by reusing a single core multiplier for the three multiplications required in each iteration in an iterative manner. However, the serial application requires additional circuitry prior and after the core multiplier, which reduces the benefits from a smaller core multiplier width.

B. Least-Significant Digit First Multiplier

Digit-serial multipliers [7] allow to balance between area and computation time by selecting an appropriate digit size. With a digit size D, an $m \times m$-bit multiplication requires $\lceil \frac{m}{D} \rceil$ digit multiplications of $m \times D$ width. Other than two-step multipliers, where multiplication and reduction are done in two separate steps (cf. Karatsuba-based multiplication), digit-serial multipliers allow to perform the modular reduction in an interleaved manner, where each digit multiplication result of length $m + D - 1$ is reduced by D bits. For the given irreducible polynomial the reduction step can be performed in a single step, which allows to use combinational networks for both the multiplication step and the interleaved reduction step. The reduction circuit is implemented with an XOR network, combining the multiplicand A with $D - 1$ bits defined by the irreducible polynomial $f(x)$. The propagation delay through this network can be minimized by structuring the XOR-gates as a binary tree of height $\lceil log_2(D) \rceil$, which results in the maximum propagation delay of the reduction circuit. Algorithm 1 illustrates, the three main parts of a least-significant digit first multiplier, namely the digit multiplication, the interleaved reduction, and the final reduction [16].



(a) Top-level architecture (b) Karatsuba-based arithmetic unit (c) Digit/Bit-serial arithmetic unit

Figure 1: Architectures of the application-specific instruction-set processor (ASIP) for pairing-based applications.

IV. Application Specific Instruction-set Processor

Figure 1a shows an overview of the top level of our ASIP. The proposed architecture contains a control unit which controls the execution of the finite-field operations, an arithmetic unit which is capable of calculating finite-field operations, two data RAMs to store temporary values during algorithmic computation, and an instruction memory holding the instruction words for the instruction set supported by the control unit.

As our design aims at resource-constrained environments and therefore area is of utmost priority, our first design decision was with regard to the employed memories. We decided to build our design based on single port register macro cells, which require by far less chip area compared to their two-port and flip-flop based counterparts. Beside the two operand memories, the arithmetic unit represents the most important block within our architecture and will be discussed in the subsequent sections. Apart from that, a hardcoded instruction memory realized by a combinational network holds the actual instructions to be carried out in order to compute the algorithm. Moreover, a serial to parallel I/O interface allows to read in the coordinates of the elliptic-curve input points respectively to write out the pairing result. The I/O interface uses a full handshake protocol for communication and exploits a byte-write feature of the used memory macrocells, allowing to save chip area and keeping a low number of input/output pins of the circuit. The I/O time to import inputs and to export results from the circuit is less than 3 % of the total computation time. The time for I/O can be easily balanced, for instance, for algorithms with short computation times or when the circuit is integrated in a system-on-chip, by applying more data pins (e.g., integer divisors of the memory's data width).

Algorithm 1 Least Significant Digit (LSD) Multiplier

Input: $A = \sum_{i=0}^{m-1} a_i x^i$ with $a_i \in \mathbb{F}_2$
Output: $C \equiv A \cdot B \bmod f(x)$.
1: $C \leftarrow 0$
2: **for** $i \leftarrow 0$ to $\lceil \frac{m}{D} \rceil - 1$ **do**
3: $\quad C \leftarrow B_i A + C$ ▷ Digit Multiplication
4: $\quad A \leftarrow Ax^D \bmod f(x)$ ▷ Interleaved Reduction
5: **end for**
6: **return** $C = C \bmod f(x)$ ▷ Final Reduction

The architecture is designed to be flexible with respect to redesigns of the arithmetic unit and changes in the algorithm itself. This flexibility is attained by following a microcoded processor approach instead of a traditional finite state machine-based implementation. From an algorithmic perspective, a microcoded approach complies better with the rather irregular instruction sequence inherent in the applied algorithms. Moreover, the numerous computational steps within the pairing algorithm are hard to maintain if implemented as classic finite-state machines. Using a microcoded approach has significant advantages as it makes changes at algorithmic level simpler. The microcode is realized as a lookup table providing the instructions to the arithmetic unit. Programming is done by an assembler-like metacode which is then transformed into the actual microcode. Due to this transformation, tasks such as algorithmic re-sequencing or rescheduling of computational blocks to get a minimal memory footprint are easier feasible.

A. Algorithmic Logic Unit

The most important component of the presented ASIP is its algorithmic logic unit (ALU), which carries out all the required arithmetic in the underlying binary finite field $\mathbb{F}_{2^{1223}}$. Since we operate in a characteristic two finite field, addition and squaring come almost for free in hardware. Moreover, the inversion is performed using Fermat's little theorem, which allows to compute the inversion by means of squarings and multiplications. Therefore, we focus on the implementation of the base-field multiplication. Based on our choice regarding the single-port memories, we now present two different ALU architectures, one based on an iteratively used Karatsuba multiplier [13] and the other one using a digit/bit-serial multiplier. The block diagrams of the arithmetic units are depicted in Figure 1b and 1c, where N is the width of the corresponding data path.

1) Karatsuba-Based Architecture: For the given case of a 1223×1223 bit multiplication, we use a three-step and a four-step approach, which allows us to have data segments of 153 and 77 bits, respectively. In the following, we denote these two architectures by K153 and K77. The three-step iterative approach uses a 153-bit core multiplier which requires 39 kGE. Accomplishing an additional iteration step of the Karatsuba

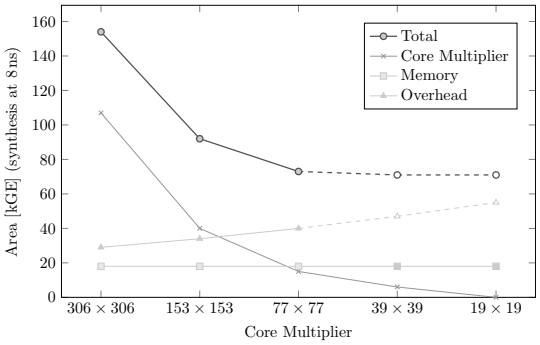


Figure 2: Cost of iteratively decomposed Karatsuba multiplier

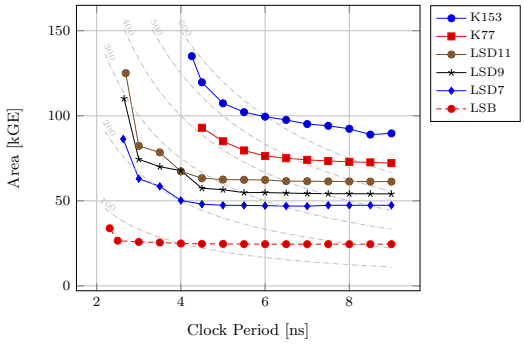


Figure 3: Area consumption of multiplier architectures

algorithm, we use a 77-bit core multiplier, requiring only 15 kGE. In order to use the core multiplier in an iterative manner, we need to multiply a specific set of operand segments and accumulations thereof to compute the partial products. To compute a partial product, the corresponding segments have to be applied to the core multiplier. Beside the segments which are directly passed from the memory to the core multiplier, there are segments which require accumulation of certain segments prior to multiplication. To accumulate these segments, we require at least one register at each core multiplier port. However, as there are 81 partial products to compute in the four-step iterative case, we add additional pre-multiplication XOR (PMX) registers (cf. Figure 1b). In order to keep the resulting circuit area low, while ensuring a fair comparison between the two iteration levels, we try to find a similar core multiplier saturation as in the single register case. For the K153 case we use one PMX register pair and obtain 65 % core multiplier saturation (41 cycles). The K77 architecture is equipped with three PMX register pairs which results in 61 % saturation of the core multiplier (132 cycles). Effectively reusing previously calculated segments is crucial in order to obtain a good performance of the Karatsuba multiplier. The results of a partial product are then added segment-wise to a large 2446-bit wide result register. A partial product generally has $2N-1$ bits, which is then organized in a upper half P^H, a lower half P^L, and the sum of these parts $P^C = P^H \oplus P^L$.

Iterative application of Karatsuba's algorithm allows to reduce the core multiplier size which dominates the area requirement for low iteration degrees. Figure 2 shows the area decomposition of Karatsuba-based multipliers according to the core-multiplier width for synthesis results for an 8 ns clock constraint. We extrapolate the architecture overhead for iteration degrees higher than four, applying a linear increase of PMX registers to compensate the otherwise significantly slower computation time due to the exponential increase in partial products. The extrapolated cost overhead of iterative decomposition suggests that iterative decomposition beyond the four-step Karatsuba-architecture does not provide a noteworthy smaller multiplier. Additionally, computing the $3^5 = 243$ partial products of the five-fold step requires significantly more computation cycles which prolongs overall pairing computation time and energy consumption.

2) Digit-Serial Architecture: Digit-serial multipliers can generally be divided into two subtypes, namely those processing the multiplier from its most significant to its least significant bit (MSB multipliers) and those processing the multiplier from its least significant to its most significant bit (LSB multipliers). For our pairing architecture, we consider a digit-serial $\mathbb{F}_{2^m}$ multiplier, as it is scalable in terms of area and time complexity and allows to do interleaved reduction so that the number of required registers to store the product can be reduced from $2m-1$ to $m+D-1$ bits. However, we need an additional shift-register to hold the $m-1$ bit multiplicand A for the interleaved reduction. Integrating an LSD multiplier mandates to decide on the digit size to be applied. In order to minimize the integration overhead and to avoid generation of extra macro cells, the same infrastructure was used for the LSD multiplier architectures as with the Karatsuba multiplier designs. While we need to load the multiplicand to a distinct register A for interleaved reduction, we do not need an additional register for the multiplier and can directly read multiplier digits B_i from the SRAM data outputs. To avoid wasting computation cycles by digit sizes other than the prime factors of 77, which would result in additional registers and surplus computation time, we chose from digit sizes of 11, 7, or 1 bits. In order to avoid excessive area consumption, we integrate the multiplier in the arithmetic unit by sharing the result register with other arithmetic modules. In Figure 1c a simplified block diagram of the arithmetic unit is given. The result register is shared between the addition, squaring, and multiplication circuits requiring a multiplexer structure. We also consider the special case for $D = 1$, i.e., the bit-serial multiplier version, for our design evaluation.

V. Results

We used VHDL to describe our architectures and synthesized them with Synopsys Design Compiler version 2010.03 for a 180 nm CMOS technology by UMC. Standard cells from Faraday Technologies under typical case conditions were used for synthesis. Back-end design for all seven different architectures was done using Cadence Design Systems Encounter Digital Implementation tool version 8.1.4. Testvectors for post-layout simulations have been obtained from a reference model based on the RELIC toolkit [17].

In order to minimize the contained parallel core multiplier size, we use a three-fold and a four-fold iterative Karatsuba, which results in data segments of 153 and 77 bits, respectively. Based on these widths, we utilized 153 and 77-bit wide memory macrocells. Starting from the Karatsuba architectures, we developed comparable digit-serial multiplier designs. Thus, for

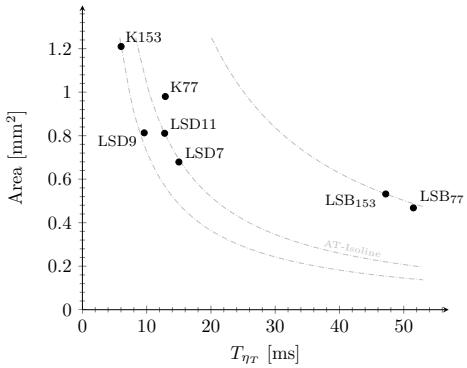
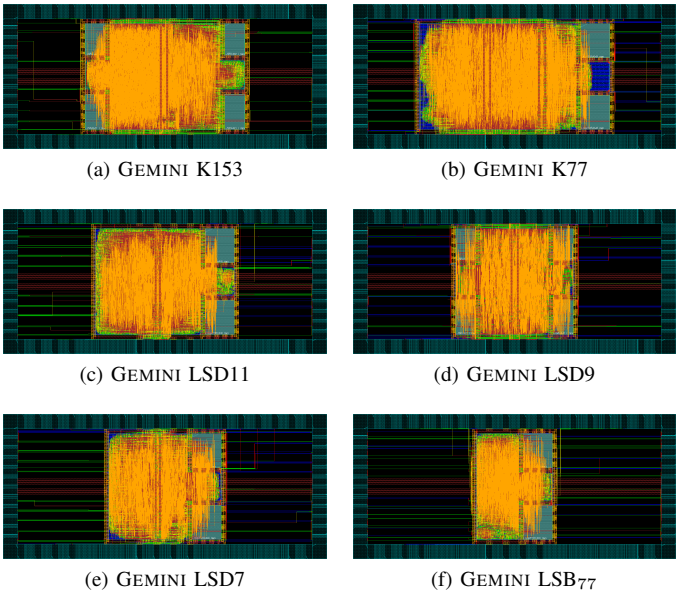


Figure 4: AT synthesis results for 8 ns



(a) GEMINI K153 (b) GEMINI K77

(c) GEMINI LSD11 (d) GEMINI LSD9

(e) GEMINI LSD7 (f) GEMINI LSB_{77}

Figure 5: Routing results after back-end design flow

the 77-bit memory interface, we decided to use a digit size of 11 and 7 bits, referred to as the LSD11 and LSD7 architectures. Factorization of the 153-bit memory interface, i.e., $3^2 \cdot 17$, does not reveal such a suitable bit width. Therefore, we decided to use a digit size of 9 bits, resulting in the LSD9 design. Moreover, we provide results for implementations based on bit-serial multipliers for both the 77 and the 153-bit memory interface versions, denoted by LSB_{77} and LSB_{153}, respectively. Area results are provided in terms of gate equivalents (GE), where one GE is equal to the area of a 2-input NAND gate, which in turn corresponds to 9.3744 μm^2 for the utilized standard cell library.

Area requirements for the multiplier architectures at the synthesis level are illustrated in Figure 3 and for the top-level designs as an AT plot in Figure 4. Figure 3 illustrates the synthesis results for each multiplier architecture over a range of clock constraints. The top-level designs where constrained with 8 ns to keep all multiplier architectures in their low-area domain. The AT-isolines in Figure 4, fitted to the designs K153, LSD11, and LSB_{153}, demonstrate the equivalence of three architecture groups based on either a common datapath width, or bit-serial multiplication. In addition to synthesis results, we also provide area numbers for the post-layout design step in Table I. Moreover, we list an "effective area" value, which is required if the architectures are actually fabricated as a single system-on-chip (including the core power ring). Consequently, these values represent realistic figures for chip area cost. Six of the seven resulting chip layouts of the architectures, called GEMINI, are presented in Figure 5, each being placed in the same pad ring. To find reasonable values for the effective chip area, the core area was successively constrained from both sides, while the pad ring was held constant. The presented layouts were constrained up to a point where optimization and routing became exceedingly complicated and time-consuming. While further back-end design iterations may provide even more compact and tailored layouts, these results provide a solid basis for estimating design integration costs.

Table I: Area requirement of implemented architectures (all designs have been constrained for a clock period of 8 ns)

Arch	Synthesis			Post-layout			Effective Area		
	mm^2	kGE	%	mm^2	kGE	%	mm^2	kGE	%
K153	1.191	126.9	100	1.243	132.6	+4	1.77	188.8	+48
K77	0.983	104.9	100	1.030	109.8	+5	2.10	223.9	+104
LSD11	0.811	86.5	100	0.848	90.4	+5	1.55	165.3	+81
LSD9	0.813	86.8	100	0.848	90.4	+4	1.36	144.7	+67
LSD7	0.679	72.4	100	0.708	75.6	+4	1.29	137.6	+90
LSB_{153}	0.532	56.7	100	0.560	59.8	+5	0.96	102.2	+80
LSB_{77}	0.468	49.9	100	0.494	52.7	+6	0.90	95.9	+92

Table I and Figure 5 indicate that the routing overhead as well as the power planning significantly contribute to the effective area of the overall design. Moreover, due to the extensive routing as well as the placement of the RAM macrocells, the core densities with regard to the standard cells vary greatly between the designs. For instance, while the K77 design offers a moderate density of only 58%, the K153 architecture reaches a standard cell density of 81%. Our fastest design (K153) needs about 750 kcycles while the slowest and also smallest architecture (LSB_{77}) requires approximately 6 500 kcycles. All implementations run at a supply voltage of 1.8 V and consume between 0.6 mW/MHz (LSB_{77}) and 2 mW/MHz (K153). Since the designs do not incorporate particular low-power design techniques such as operand isolation, there is still potential to improve their power budget. For the power consumption figures given in Table II, we limited power analysis to the actual computation period and ignored the time for input and output where I/O pad cells are active. As the target application is an integration to a system-on-chip or to a crypto-processor module, this approach provides more meaningful results. Power analysis is based on actual stimuli provided by a pairing calculation to the back-annotated circuits. Compared to the internal and switching power, the amount of leakage power is negligible. It is interesting to note that, even though the Karatsuba-based design K153 is consuming the largest amount of power on average, it is also the most energy efficient due to its fast computation time. In a general case, the LSD-based designs offer better overall-performance regarding area, time, and energy cost.

Based on these results, the digit-serial based architectures are to be favored in terms of area and time optimality as they provide a good balance between area requirement and multiplier latency. The bit-serial architectures offer minimal requirements in chip area at the expense of higher computation time and energy consumption.

In Table III, we provide a comparison of the presented ASIC design implementations. While the usually published results solely provide synthesis values, we list the post-layout gate counts, which are more accurate numbers since they include any required buffers and are based on realistic wire-load models. As shown in Table III, a full η_T pairing implementation can be designed using roughly 50 kGE while keeping the computation time in the magnitude of around 50 ms. Since all of our designs were subject to the back-end design step targeting a clock period of 8 ns (in order to provide a fair comparison among the different architectures), some of the smaller designs definitely allow faster clock rates while keeping the circuit size constant.

Hardware results published in open literature usually restrict themselves to synthesis results, as actual chip layout is complex, costly, and time consuming. To simulate a circuit's actual power consumption it is mandatory to have realistic parasitics which can only be extracted from an actual layout. However some chip design tools provide features to extract power consumption figures at the synthesis stage. It is important to note that these figures are basically very rough estimates as they lack layout information due to standard-cell placement and signal routing of a chip.

VI. Conclusion

By consequently designing for low chip-area and focusing on efficient implementations of the finite-field multiplication, we are able to present several low-area hardware multiplier solutions which are able to compute complex algorithms such as cryptographic pairings in reasonably short time. We show that the Karatsuba multiplier, which has so far mostly been used for high-throughput implementations, might also represent a suitable alternative to area-saving multiplier types like the digit-serial approach. The four-step iterative architecture K77 was significantly more complex in the back-end design stage than all the other designs. Its numerous 77-bit multiplexers and disperse result registers make signal routing and timing fixes hard to achieve. This suggests that higher degrees of iteration have adverse effects on the design size and its feasibility.

We present seven different architectures using different memory/multiplier constellations based on a mature CMOS ASIC technology. Providing actual back-annotated design results for chip area, computation time, as well as power and energy consumption, this work provides a substantial basis for future design decisions to implement systems utilizing large-scale binary-field arithmetic.

Table II: Power dissipation obtained by stimuli-based power analysis in mW (f_{clk}=125 MHz)

Architecture	Internal Power	Switching Power	Leakage Power	Total
K153	112.50	151.20	0.011	263.7
K77	86.45	136.70	0.007	223.1
LSD11	94.22	81.89	0.006	176.1
LSD9	110.00	72.40	0.009	182.4
LSD7	77.49	58.56	0.007	136.0
LSB_{153}	76.91	28.90	0.008	105.8
LSB_{77}	51.14	25.57	0.005	80.7

Table III: Comparison of ASIC implementations (based on post-layout simulations)

Arch	Tech. nm	f_{max} MHz	Power$^{\alpha}$ mW	Energy mJ/Comp.	Time ms	Area kGE	ATE
K153	180	125	264	1.52	6.02	133	1.0
K77	180	125	223	2.93	12.90	110	3.4
LSD11	180	125	176	2.19	12.80	90	2.1
LSD9	180	125	182	1.87	9.62	90	1.3
LSD7	180	125	136	1.99	15.01	76	1.9
LSB_{153}	180	125	106	4.78	47.19	60	11.1
LSB_{77}	180	125	81	4.08	51.48	53	9.1

$^{\alpha}$ @ f_{max}, 1.8 V

References

[1] W. Diffie and M. Hellman, "New Directions in Cryptography," *Information Theory, IEEE Trans. on*, vol. 22, no. 6, pp. 644–654, Nov 1976.

[2] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[3] V. S. Miller, "Use of Elliptic Curves in Cryptography," in *Advances in Cryptology-CRYPTO'85 Proceedings*. Springer, 1986, pp. 417–426.

[4] D. Boneh and M. Franklin, "Identity-Based Encryption from the Weil Pairing," in *CRYPTO 2001*, ser. LNCS, J. Kilian, Ed. Springer Berlin Heidelberg, 2001, vol. 2139, pp. 213–229.

[5] D. Boneh et al., "Short Signatures from the Weil Pairing," in *ASIACRYPT 2001*, ser. LNCS. Springer, 2001, vol. 2248, pp. 514–532.

[6] A. Karatsuba and Y. Ofman, "Multiplication of Multidigit Numbers on Automata," in *Soviet Physics Doklady*, vol. 7, 1963, p. 595.

[7] L. Song and K. K. Parhi, "Low-Energy Digit-Serial/Parallel Finite Field Multipliers," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 19, no. 2, pp. 149–166, 1998.

[8] K. Paterson, "ID-based Signatures from Pairings on Elliptic Curves," *Electronics Letters*, vol. 38, no. 18, pp. 1025–1026, 2002.

[9] J.-L. Beuchat et al., "A Comparison between Hardware Accelerators for the Modified Tate Pairing over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$," in *Pairing 2008*, ser. LNCS. Springer Berlin Heidelberg, 2008, vol. 5209, pp. 297–315.

[10] S. Ghosh et al., "High Speed Cryptoprocessor for η_T Pairing on 128-bit Secure Supersingular Elliptic Curves over Characteristic Two Fields," in *CHES 2011*, ser. LNCS. Springer, 2011, vol. 6917, pp. 442–458.

[11] J. Adikari et al., "Towards Faster and Greener Cryptoprocessor for Eta Pairing on Supersingular Elliptic Curve over $\mathbb{F}_{2^{1223}}$," in *Selected Areas in Cryptography*, ser. LNCS. Springer, 2013, vol. 7707, pp. 166–183.

[12] P. Barreto et al., "Efficient Pairing Computation on Supersingular Abelian Varieties," *Designs, Codes and Cryptography*, vol. 42, pp. 239–271, 2007.

[13] Z. Dyka and P. Langendoerfer, "Area Efficient Hardware Implementation of Elliptic Curve Cryptography by Iteratively Applying Karatsuba's Method," in *DATE'05*, march 2005, pp. 70 – 75 Vol. 3.

[14] T. Itoh and S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in GF(2^m) Using Normal Bases," *Information and Computation*, vol. 78, no. 3, pp. 171–177, 1988.

[15] F. Rodrıguez-Henrıquez and Ç. Koç, "On fully parallel Karatsuba Multipliers for GF(2^m)," in *Proc. International Conference on Computer Science and Technology-CST*, 2003, pp. 405–410.

[16] S. Kumar et al., "Optimum Digit Serial GF(2^m) Multipliers for Curve-Based Cryptography," *Computers, IEEE Transactions on*, vol. 55, no. 10, pp. 1306–1311, 2006.

[17] D. F. Aranha and C. P. L. Gouvêa, "RELIC is an Efficient LIbrary for Cryptography," http://code.google.com/p/relic-toolkit.