A Pipelined Karatsuba-Ofman Multiplier over $GF(3^{97})$ Amenable for Pairing Computation

Nidia Cortez-Duarte[1], Francisco Rodríguez-Henríquez[1], Jean-Luc Beuchat[2] and Eiji Okamoto[2]

[1] Computer Science Departament, Centro de Investigación y Estudios Avanzados del IPN, Av. Instituto Politécnico Nacional No. 2508, México D.F.

[2] Graduate School of Systems and Information Engineering, University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki, 305-8573, Japan.

Abstract. We present a subquadratic ternary field multiplier based on the combination of several variants of the Karatsuba-Ofman scheme recently published. Since one of the most relevant applications for this kind of multipliers is pairing computation, where several field multiplications need to be computed at once, we decided to design a k-stage pipeline structure for $k = 1, \ldots, 4$, where each stage is composed of a 49-trit polynomial multiplier unit. That architecture can compute an average of k field multiplications every three clock cycles, which implies that our four-stage pipeline design can perform more than one field multiplication per clock cycle. When implemented in a Xilinx Virtex V XC5VLX330 FPGA device, this multiplier can compute one field multiplication over $GF(3^{97})$ in just 11.47ns.

keyword: Finite field arithmetic; Field Multiplier; cryptography

1 Introduction

Arithmetic over ternary extension fields $GF(3^m)$ has gained an increasing importance in several relevant cryptographic applications, quite especially, in (hyper) elliptic Curve Cryptography and pairing based cryptography. Those applications typically require efficient implementation of the basic arithmetic finite field operations such as field addition, subtraction, multiplication, division, exponentiation, cubing and cube root computation. Among the aforementioned arithmetic operations, multiplication is by far the one that has received more attention, mostly because of its crucial role in the aforementioned cryptographic schemes.

We formally define a ternary finite field as follows. Let $P(x)$ be an irreducible polynomial over *GF*(3). Then, the ternary extension field *GF*(3^m) of degree $m \in \mathbb{N}$ can be defined as,

$$\mathit{GF}(3^m) \cong \mathit{GF}(3)[x]/\left(P(x)\right) = \left\{\left(a_0 + a_1x + \cdots + a_{m-1}x^{m-1}\right) \mid a_i \in \mathit{GF}(3), i = 0, \ldots, m-1\right\}.$$

The ternary coefficients a_i are sometimes called *trits*. Addition and subtraction are the simplest arithmetic operations in $GF(3^m)$ as these two operations can be performed by adding/subtracting each trit in arithmetic modulo three. On the contrary, field multiplication is a more complicated operation that deserves much more attention.

Let $A(x), B(x)$ and $C'(x) \in GF(3^m)$ and $P(x)$ be the irreducible polynomial used to construct the field $GF(3^m)$. Multiplication in $GF(3^m)$ is defined as polynomial multiplication modulo the irreducible polynomial $P(x)$, i.e., $C'(x) = A(x)B(x) \bmod P(x)$. Hence, we can obtain the field product by first computing the polynomial product $C(x)$ of degree at most $2m-2$ as,

$$C(x) = A(x)B(x) = (\sum_{i=0}^{m-1} a_ix^i)(\sum_{i=0}^{m-1} b_ix^i)$$

Followed by a reduction step modulo $P(x)$ that can be obtained through addition and subtraction operations only. This implies that the reduction step can be accomplished at a linear complexity cost and therefore its cost is modest compared with that of the first step. Depending on the specific scheme selected, polynomial multiplication may have either super linear or quadratic complexity. Hence, it results convenient to classify parallel field multiplier schemes according to their space complexity as quadratic and subquadratic space complexity multipliers.

Parallel multiplier schemes exhibiting quadratic space complexity have been profusely reported in the literature. Some famous examples are, the schoolbook method, interleaving multipliers, matrix-vector multipliers, Montgomery multipliers and Horner's rule-based multipliers, along with many other variants of them. Curiously, there exist much less literature on subquadratic multiplier schemes. A non exhaustive list of those schemes includes, the Toeplitz matrix-vector product, DFT Multipliers, Chinese Remainder Theorem (CRT) multipliers, and Divide-and-Conquer multipliers such as the Winograd Multiplier, the Toom-Cook Multiplier discovered in 1963 and the Karatsuba-Ofman Multiplier (KOM) [9].

Due to the subquadratic space complexity enjoyed by KOM, which is supported by closed complexity formulae bounds, this scheme has become a very attractive design option for implementing fully or semi parallel field multipliers over prime extension fields in the cryptographic range of interest (i.e., extension degrees $m \in [70, 560]$). Till now several fast hardware and software KOM implementations have been reported [5, 13, 16], all of them over binary extension fields *GF*(2^m). In [7] a digit-serial LFSR multiplier combined with KOM techniques over *GF*(3^m) was reported.

In this paper we present the design specifics of a Karatsuba-Ofman field multiplier over *GF*(3^m) that borrows ideas of the KOM variants proposed in [4] and [6]. Further, since the main applications of a ternary field multiplier require the batch computation of a relatively large number of products, we decided to utilize a k-stage pipeline structure for $k = 1, \ldots, 4$. Our architecture is able to perform k field multiplications every three clock cycles. This way, the fastest design reported in this research work can compute one field multiplication over *GF*(3^{97}) in 11.47 ns, whereas our most efficient multiplier achieves a Speed Per Area (SPA) factor of $15,442$ when implemented in a Xilinx Virtex V FPGA device. [1]

The rest of this paper is organized as follows. In §2, the Karatsuba-Ofman multiplier algorithm and its variants are explained. Then, in §3 we describe the multiplier design architecture and we give in §4 a summary of the implementation results achieved, and a comparison with other ternary field multipliers recently reported. Final remarks are drawn in §5.

2 Karatsuba-Ofman Multiplication

Let the field *GF*(3^m) be constructed using the polynomial $P(x)$ irreducible over *GF*(3), with $m > 1$ an odd number and $n = \lceil \frac{m}{2} \rceil$. Let A, B be two elements in

[1] The Speed Per Area (SPA) factor is defined as $SPA = (slices \cdot latency)^{-1}$, where $slices$ refers to the number of XILINX FPGA slices required by the design and $latency$ is the time elapsed till one computation is ready.

$GF(3^m)$. Then, both elements can be represented in the polynomial basis as,

$$A = \sum_{i=0}^{m-1} a_i x^i \;=\; \sum_{i=n}^{m-1} a_i x^i + \sum_{i=0}^{n-1} a_i x^i = x^n \sum_{i=0}^{n-2} a_{i+n} x^i + \sum_{i=0}^{n-1} a_i x^i \;=\; x^n A^H + A^L;$$

$$B = \sum_{i=0}^{m-1} b_i x^i \;=\; \sum_{i=n}^{m-1} b_i x^i + \sum_{i=0}^{n-1} b_i x^i = x^n \sum_{i=0}^{n-2} b_{i+n} x^i + \sum_{i=0}^{n-1} b_i x^i \;=\; x^n B^H + B^L.$$

The Karatsuba-Ofman Multiplier (KOM) is based on the observation that the polynomial product $C = A \cdot B$ can be written as,

$$\begin{aligned} C &= x^{2n} A^H B^H + (A^H B^L + A^L B^H) x^n + A^L B^L \\ &= x^{2n} A^H B^H + A^L B^L + \left[(A^H + A^L)(B^L + B^H) - (A^H B^H + A^L B^L) \right] x^n \end{aligned} \tag{1}$$

With a computational cost of three n-trit polynomial multiplications and 4 additions/subtractions. By applying this strategy recursively, in each iteration each degree polynomial multiplication is transformed into three polynomial multiplications with their degrees reduced to about half of its previous value. The algorithm ends after $\lceil \log_2(m) \rceil$ iterations, where all the polynomial operands collapse into single coefficients. Nevertheless, practice has shown that is better to truncate the KOM algorithm earlier, with the aim of performing the underlying multiplications using alternative techniques that are more compact and/or faster for small operands (typically the so-called school book method has been selected [13, 16]). For example, if $m = rs$, with $r = 2^k$, k a positive integer, we can halt the KOM recursion after k iterations and then compute the underlying s-trit polynomial multiplications using the school-book method. It has been shown that in this case the space and time complexities of that hybrid scheme are upper bounded as [13, 16, 6],

$$\begin{aligned}
\#\ \text{Additions/subtractions} \in GF(3) &\leq \left(\frac{m}{s}\right)^{\log_2 3}(s^2+6s-1)-8m+2\ ;\\
\#\ \text{Multiplications} \in GF(3) &\leq \left(\frac{m}{s}\right)^{\log_2 3} s^2; \qquad (2)\\
\text{Latency} &\leq T_{Mult} + (\log_2 s + 3\log_2 r)T_{Add}\ .
\end{aligned}$$

Where T_{Mult} and T_{Add} represents the time delay of one multiplication and one addition over *GF*(3), respectively. Some design disadvantages associated to KOM schemes include the redundant computation of several multiplications when the extension degree m is not a power of two; and also the fact that the KOM scheme has an associated time delay higher than other bit-parallel multiplier schemes. Because of these drawbacks, several KOM variants that try to remedy these issues have been proposed [5, 13, 4, 11, 6]. In [6] a novel KOM proposal was presented, where instead of dividing the operands in upper and lower halves, the operands are split according to the parity of the powers of the variable x, i.e., in odd and even coefficients. Hence, if $A, B \in GF(3^m)$ they can be split as,

$$A = \sum_{i=0}^{m-1} a_i x^i \ = \ \sum_{i=0}^{n-1} a_{2i}x^{2i} + x\cdot\sum_{i=0}^{n-2} a_{2i+1}x^{2i} \ = \ \sum_{i=0}^{n-1} a_{2i}y^i + x\cdot\sum_{i=0}^{n-2} a_{2i+1}y^i = A_e(y) + xA_o(y);$$
$$B = \sum_{i=0}^{m-1} b_i x^i \ = \ \sum_{i=0}^{n-1} b_{2i}x^{2i} + x\cdot\sum_{i=0}^{n-2} b_{2i+1}x^{2i} \ = \ \sum_{i=0}^{n-1} b_{2i}y^i + x\cdot\sum_{i=0}^{n-2} b_{2i+1}y^i = B_e(y) + xB_o(y).$$

Where as before $n = \lceil \frac{m}{2} \rceil$ and $y = x^2$. This partition leads to the following Karatsuba-Ofman like multiplication equation,

$$\begin{aligned}
C = {} & [A_e(y) + xA_o(y)) \cdot (B_e(y) + xB_o(y)] = A_e(y)B_e(y) + \\
& + yA_o(y)B_o(y) + x\left[(A_e(y)+A_o(y))(B_e(y)+B_o(y)) - (A_e(y)B_e(y) + A^L B^L))\right]
\end{aligned} \qquad (3)$$

which improves the KOM latency of Eq.2 to $T_{Mult} + (\log_2 s + 2\log_2 r)T_{Add}$.

In this work we present the design of a Karatsuba-Ofman field multiplier over *GF*(3^m) that borrows some of the ideas already proposed in [4] and [6]. The interested reader is referred to Appendix A where a brief description of the KOM variant proposed in [4] is given. A pipelined architecture realization of such multiplier is explained in the next Section.

3 A Pipelined Karatsuba-Ofman Multiplier over $GF(3^m)$

We describe here the design of a KOM operating in the ternary extension field *GF*(3^m), with $m = 97$, which was constructed using the irreducible trinomial $P(x) = x^{97} + x^{12} - 1$. This ternary extension field is a popular choice for pairing computations and that is the reason why we selected it. However, the same design principles to be explained in this Section can be applied for other field selections.

Since $2 \equiv -1 \bmod 3$, it results convenient to encode a trit $a \in GF(3)$ using a positive bit a^+ and a negative bit a^- such that $a = a^+ - a^-$. Multiplication of two elements $a, b \in GF(3)$ is now defined by [3],

$$a \cdot b = \big((1-b^-)b^+a^+ \; V \, b^-(1-b^+)(1-a^+)\big) - \big((1-b^-)b^+a^- \; V \, b^-(1-b^+)(1-a^-)\big)$$

and requires two 3-input LUTs or one 6-input/two-output LUT. Thus, the fanout of the array multiplier is equal to m. Additionally, trit negation can be accomplished by just bit swapping. An arbitrary field element in *GF*(3^{97}) is therefore represented as a 97-trit word that occupies 194 bits of memory. Hence, a parallel field adder/subtracter in *GF*(3^{97}) is composed of 97 adders/subtracters in *GF*(3).

Since we are considering the case $m = 97$, then $\lceil \frac{97}{2} \rceil = 49$, and we have that a 97-trit polynomial multiplication can be accomplished via Karatsuba-Ofman strategy by invoking three times a 49-trit polynomial multiplier unit along with four 49-trits adders/subtracters. The 49-trit polynomial multiplier unit was designed as shown in Fig.5, following the non redundant KOM variant introduced in [4] (See Appendix A for a brief description of this variant). We also utilized the even-odd partition introduced in [6] in all the sub-modules shown in Fig.5. The overlapping module was accomplished according to Eqs. 1 and 3, whereas the reduction module was implemented using the reduction equation $x^{m+i} = x^i - x^{12+i}$ for $i \geq 0$, dictated by the irreducible trinomial, $P(x) = x^{97} + x^{12} - 1$.

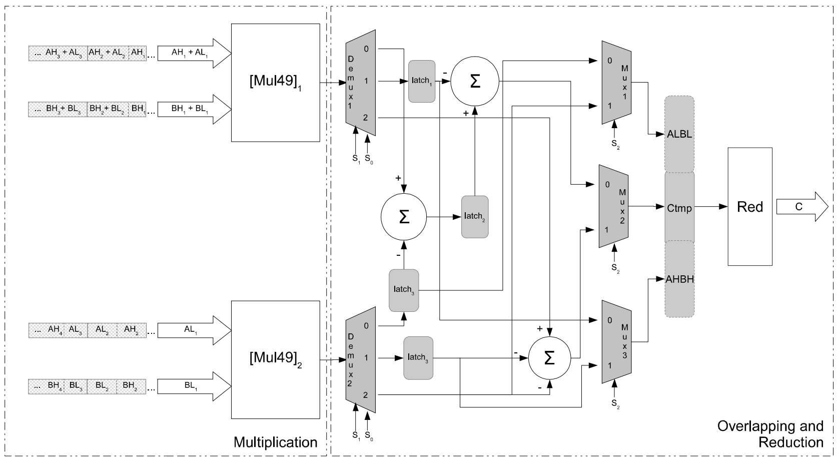

Fig. 1. Two-Stage Pipeline Architecture of a $GF(3^{97})$ KOM Multiplier

Having designed a 49-trit polynomial multiplier unit we proceeded to use it as a stage for building a pipelined multiplier architecture. This can be useful for applications where multiple field multiplications need to be computed, such as

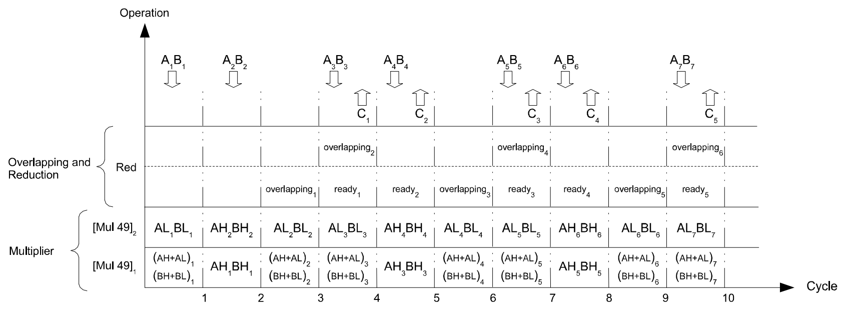
Fig. 2. *GF*(3^{97}) Field Multiplier Dataflow when Using a Two-Stage Pipeline Architecture

when computing field exponentiations and/or pairings. A concrete application of this is the evaluation-interpolation scheme introduced by Gorla et al. in [15] to perform multiplication over *GF*(3^{6m}) by means of five multiplications over *GF*(3^{2m}). In turn, each multiplication over *GF*(3^{2m}) can be computed with three multiplications over *GF*(3^{m}). Thus, the scheme proposed in [15] to multiply two elements in *GF*(3^{6m}) involves 15 multiplications over *GF*(3^{m}). The pipeline multiplier architecture proposed here appears to be well suited for an efficient computation of the *GF*(3^{6m}) multiplication algorithm just described.

In this work we designed a pipeline architecture using up to four 49-trit multiplier stages. However, due to space constraints, only the architecture of the two-stage pipeline multiplier will be fully explained.

Figures 1 and 2 show the general architecture and the dataflow associated to a two-stage pipeline architecture. As it is shown in Figure 1 our structure is divided into two main blocks, namely, the block composed of two polynomial multiplier units and the overlapping & reduction block. We recall that according to Eq.1, one full *GF*(3^{97}) multiplication requires three 49-trit polynomial multiplications. Therefore, we decided to use the first 49-trit multiplier unit

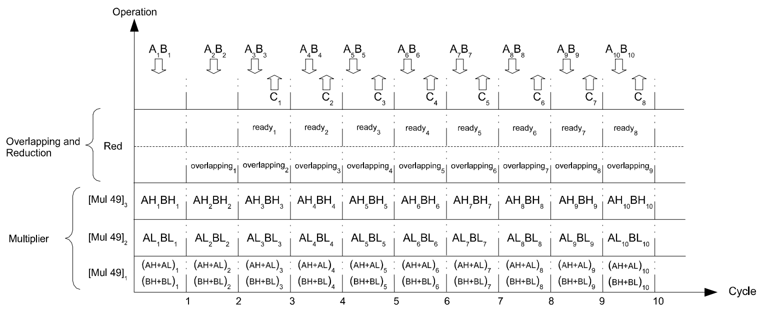

Fig. 3. *GF*(3^{97}) Field Multiplier Dataflow when Using a Three-Stage Pipeline Architecture

for computing the partial products $(A_i^H + A_i^L)(B_i^L + B_i^H)$, and $A_i^H B_i^H$ for $i \geq 1$ in the first and second clock cycles of a new computation, respectively. [2] Meanwhile, the second multiplier unit is used to compute the products $A_i^L B_i^L$ and $A_{i+1}^H B_{i+1}^H$ in the first and second clock cycles, respectively. The outputs of the multiplier unit are coupled to two demultiplexer blocks that either store the computed values in latches or, if the dataflow permits, directly add/subtract the multiplier's output with other products previously computed. Notice that the products $A_i^L B_i^L$ and $A_i^H B_i^H$ are also needed to feed the overlapping module. This is accomplished by using the multiplexer blocks 1 and 3, respectively. The multiplexer 2, on the other hand, is in charge of collecting the partial product $(A_i^H + A_i^L)(B_i^L + B_i^H)$, which can come from two different adder blocks. The control signals S_0, S_1 and S_2 are provided by the control unit which orchestrates the dataflow in the way illustrated in Fig. 2. As it is shown in Fig. 2, starting from the third clock cycle, the two-stage pipelined multiplier can compute two products every three clock cycles.

[2] Here and in the rest of this discussion the notation A_i^H, A_i^L and B_i^H, B_i^L is used to indicate the upper and lower halves of the operands A_i and B_i, respectively for $i \geq 1$.

Following the same strategy, we designed a three and four-stage pipeline multiplier. Figures 3 and 4 show the field multiplier dataflow when using a three and a four-stage structure, respectively. It can be seen that starting from the second clock cycle, those multipliers can compute three and four field multiplications every three clock cycles, respectively. This implies that the four-stage pipelined multiplier enjoys a super scalar computation capability, in the sense that it can compute more than one product per clock cycle.

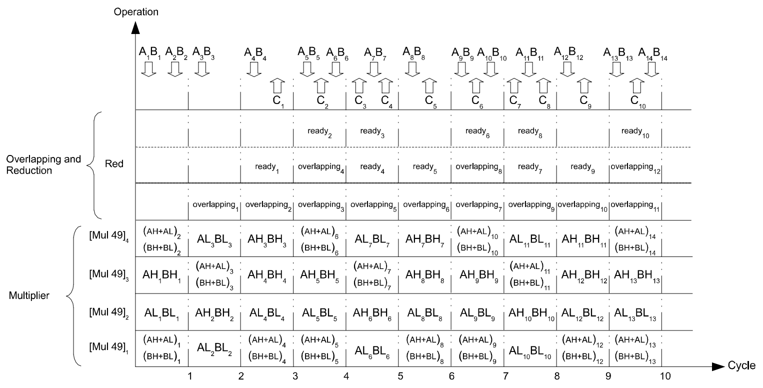

Fig. 4. $GF(3^{97})$ Field Multiplier Dataflow when Using a Four-Stage Pipeline Architecture

4 Results and Comparison

The fastest design reported in this research work, the four-stage pipeline structure, can compute one field multiplication over $GF(3^{97})$ in 11.467ns, whereas our most efficient multiplier, the three-stage pipeline structure achieves a Speed Per Area (SPA) factor of $15,442.0$ when implemented in a Xilinx Virtex V

Table 1. Hardware Cost of several $GF(3^{97})$ Multipiers Specifically Designed for Pairings

Multiplier	**Platform**	**Cycles**	**Time (ns)**	**Latency (ns)**	**Frequency (MHz)**	**Area (Slices)**	**Efficiency**
Beuchat et al. [2]	Cyclone II	33	6.711	221.46	149	700	6450.0
Shokrollahi et al. [15]	Virtex II	97	3.33	323.3	300	327	9458.0
	Virtex II	14	9.0	126.12	111	2954	2684.0
	Virtex II	7	13.9	97.2	72	4006	2568.2
Ronan et al. [14]	Virtex II Pro	7	16.23	113.6	61.6	$3,737$	2355.6
Grabher et al. [8]	Virtex II Pro	28	6.67	186.6	150	946	5664.9
Kerings et al. [10]	Virtex II Pro	25	34.129	853.22	29.3	$1,821$	644.0
Bertoni et al. [1]	Virtex II Pro	7	10.6	74.15	94.4	$3,561$	3787.2
Here_Pipe1	Virtex V	3	11.939	35.817	83.759	$2,337$	11947.0
Here_Pipe2	Virtex V	$\frac{3}{2}$	14.219	21.328	70.328	$4,163$	11262.0
Here_Pipe3	Virtex V	1	13.229	13.229	75.591	$4,895$	15442.0
Here_Pipe3	Virtex II Pro	1	17.895	17.895	56.786	$9,041$	6180.9
Here_Pipe4	Virtex V	$\frac{3}{4}$	15.289	11.467	65.407	$7,740$	11267.2

XC5VLX330 FPGA device. The synthesis and place-and-route steps were performed using Xilinx ISE 9.2i and ModelSimSE 6.2c design tools.

Table 1 shows several field multipliers over $GF(3^m)$ recently reported in the open literature. Since most of these designs were implemented in Xilinx Virtex II Pro, for comparison purposes we decided to implement our three-stage pipeline multiplier in the XC2VP70 device, which belongs to the same family.

5 Conclusion

In this paper we have presented a field multiplier in characteristic three, whose design is based on a combination of two variants recently proposed of the well-known Karatsuba-Ofman multiplication algorithm. In an effort to obtain a faster multiplier without increasing the critical path of the algorithm, we introduced a pipelined architecture which can be useful in those applications were several or many field multiplications need to be computed in batch. To the best of our knowledge, this work presents the first $GF(3^{97})$ field multiplier with super scalar computation capability, i.e., a multiplier that is able to compute in average more than one field multiplication per clock cycle. Taking advantage of the subquadratic space complexity of the Karatsuba-Ofman algorithm along with the pipelined architecture just described we obtained place-and-route simulation results showing that our multipliers are among the most efficient designs recently published. In particular, the four-stage pipeline architecture appears to be the fastest ternary field multiplier reported in the open literature.

References

1. G. Bertoni, J. Guajardo, S. S. Kumar, G. Orlando, C. Paar, and T. J. Wollinger. Efficient $GF(p^m)$ arithmetic architectures for cryptographic applications. In M. Joye, editor, *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 158–175. Springer, 2003.
2. J.-L. Beuchat, N. Brisebarre, J. Detrey, and E. Okamoto. Arithmetic operators for pairing-based cryptography. In P. Paillier and I. Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 2007.
3. J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, M. Shirase, and T. Takagi. Algorithms and arithmetic operators for computing the η_t pairing in characteristic three. Cryptology ePrint Archive, Report 2007/417, 2007. http://eprint.iacr.org/.
4. N. S. Chang, C. H. Kim, Y.-H. Park, and J. Lim. A non-redundant and efficient architecture for Karatsuba-Ofman algorithm. In J. Zhou, J. Lopez, R. H. Deng, and F. Bao, editors, *ISC*, volume 3650 of *Lecture Notes in Computer Science*, pages 288–299. Springer, 2005.
5. S. S. Erdem and Ç. K. Koç. A less recursive variant of karatsuba-ofman algorithm for multiplying operands of size a power of two. In *IEEE Symposium on Computer Arithmetic*, pages 28–35. IEEE Computer Society, 2003.
6. H. Fan and M. A. Hasan. A new approach to subquadratic space complexity parallel multipliers for extended binary fields. *IEEE Trans. Computers*, 56(2):224–233, 2007.

7. E. Gorla, C. Puttmann, and J. Shokrollahi. Explicit formulas for efficient multiplication in $\mathbb{F}_{3^{6m}}$. In C. M. Adams, A. Miri, and M. J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 173–183. Springer, 2007.
8. P. Grabher and D. Page. Hardware acceleration of the tate pairing in characteristic three. In Rao and Sunar [12], pages 398–411.
9. A. Karatsuba and Y. Ofman. Multiplication of Multidigit Numbers on Automata. *Soviet Phys. Doklady (English Translation)*, 7(7):595–596, January 1963.
10. T. Kerins, W. P. Marnane, E. M. Popovici, and P. S. L. M. Barreto. Efficient hardware for the tate pairing calculation in characteristic three. In Rao and Sunar [12], pages 412–426.
11. P. L. Montgomery. Five, six, and seven-term karatsuba-like formulae. *IEEE Trans. Computers*, 54(3):362–369, 2005.
12. J. R. Rao and B. Sunar, editors. *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*. Springer, 2005.
13. F. Rodríguez-Henríquez and Ç K. Koç. On fully parallel karatsuba multipliers for $GF(2^m)$. In *Proceedings of International Conference on Computer Science and Technology CST*, pages 405–410, May 19-21 2003. Acta Press, Cancún México.
14. R. Ronan, C. Murphy, T. Kerins, C. O.hEigeartaigh, and P. S. Barreto. A flexible processor for the characteristic 3 η_T pairing. *Int. J. High Performance Systems Architecture*, 1(2):79–88, 2007.
15. J. Shokrollahi, E. Gorla, and C. Puttmann. Efficient FPGA-based multipliers for $\mathbb{F}_{3^{97}}$ and $\mathbb{F}_{3^{6\cdot 97}}$. *CoRR*, abs/0708.3022, 2007.
16. J. v. z. Gathen and J. Shokrollahi. Efficient FPGA-based Karatsuba multipliers for polynomials over $\mathbb{F}_2$. In *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers*, volume 3897 of *Lecture Notes in Computer Science*, pages 359–369. Springer-Verlag, 2006.

Appendix A Non Redundant Karatsuba-Ofman Multiplier

Authors in [4], presented a non redundant KOM algorithm that strives for avoiding the redundancy of the KOM terms when computing polynomial products with trit-length m not a power of two. Let $A(x), B(x) \in GF(3^m)$, with $m = r + s$, $0 < s < r = 2^k$, k a positive integer. Then the polynomial product $C = A \cdot B$ can be computed using an unbalanced splitting as follows,

$$A(x) = \sum_{i=0}^{m} a_i x^i \longrightarrow A^L = \sum_{i=0}^{r-1} a_i x^i;\ A^H = x^r \cdot \sum_{i=0}^{s-1} a_{i+r} x^i.$$
$$B(x) = \sum_{i=0}^{m} b_i x^i \longrightarrow B^L = \sum_{i=0}^{r-1} b_i x^i;\ B^H = x^r \cdot \sum_{i=0}^{s-1} b_{i+r} x^i.$$

thus,

$$A^H + A^L = \sum_{i=0}^{s-1} (a_i + a_{i+r}) x^i + \sum_{i=s}^{m-1} a_i x^i;$$
$$B^H + B^L = \sum_{i=0}^{s-1} (b_i + b_{i+r}) x^i + \sum_{i=s}^{m-1} b_i x^i.$$

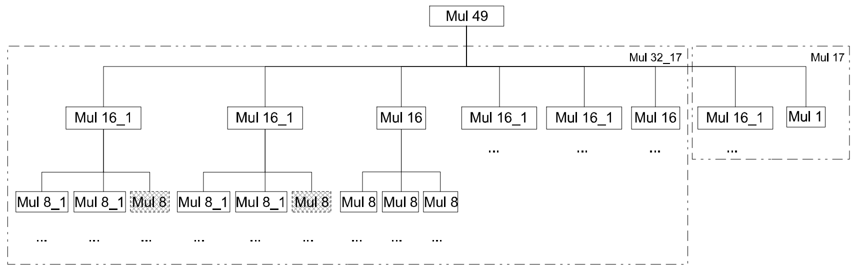

Fig. 5. A Non-Redundant 49-Trit Polynomial Multiplier Unit

Authors in [4] observed that the upper $m - s$ trits of the terms $(A^H + A^L)$, A^L are identical. The same occurs with the terms $(B^H + B^L)$, B^L. Therefore, if the KOM partial multiplication $(A^L \cdot B^L)$ and $(A^H + A^L) \cdot (B^H + B^L)$ are carefully computed, the upper $m - s$-trit product can be shared, thus computing it just once and not twice. Because of practical reasons, authors in [4] save this computation only when the extra condition $s < 2^{k-1}$ is satisfied. They do this by using two specialized functions, namely, NRKOA, and NRHKOA, that separates into two parts the redundant and non-redundant factors that appear from the KOM dataflow. Figure 5 shows a non redundant 49-trit KOM, where the shadowed MUL8 blocks can be shared as explained above.