Reduction-Free Multiplication for Finite Fields and Polynomial Rings

Samira Carolina Oliva Madrigal[1](✉), Gökay Saldamlı[1], Chen Li[2], Yue Geng[2], Jing Tian[2], Zhongfeng Wang[2], and Çetin Kaya Koç[3,4,5]

[1] San José State University, San José, USA
scolivamadrigal@gmail.com, gokay.saldamli@sjsu.edu
[2] Nanjing University, Nanjing, China
{MG21230068,181180030}@smail.nju.edu.cn, {tianjing,zfwang}@nju.edu.cn
[3] Iğdır University, Iğdır, Turkey
[4] Nanjing University of Aeronautics and Astronautics, Nanjing, China
[5] University of California, Santa Barbara, Santa Barbara, USA
cetinkoc@ucsb.edu

Abstract. The complexity of the multiplication operation over polynomial rings and finite fields drastically changes with the selection of the defining polynomial of the respective mathematical structure. Trinomials and pentanomials are the most natural choices for the best arithmetic. In this paper, we first present a study in which a special type of trinomial does not require any reduction steps. We then introduce two new algorithms, FIKO and RF-FIKO, fully interleaved bit-parallel Karatsuba-Ofman multipliers where the latter is only concerned with the three Karatsuba-Ofman terms and is free from the bipartite reduction circuits. All algorithms are implemented in FPGA and ASIC, and detailed implementation results are presented, showing significant improvements to existing methods.

Keywords: Cryptography · Finite field arithmetic · Polynomials rings · Karatsuba-Ofman multiplication · Polynomial bi-partite multiplication · Montgomery multiplication · Interleaved multiplication · Mersenne polynomials · Pseudoprimes · Equally-spaced polynomials · Reduction-free trinomials · Reduction-free multiplication

1 Introduction

The study of irreducible polynomials is particularly important in cryptography. For modern schemes that employ modular arithmetic such as, RSA and ECDSA, the multiplication operation is the most expensive. In particular, Post-quantum Cryptography (PQC) lattice-based schemes and Fully Homomorphic Encryption (FHE) systems would benefit from efficient finite field arithmetic. For practicality, security, and efficiency, we are concerned with binary fields in the polynomial basis (PB).

The most efficient bit-parallel multipliers known all make use of special trinomials to obtain time and space complexity speedups. Moreover, they make efforts to simplify modular reduction as a means to obtain faster implementations. In this paper we present a solution to the problem $A(t) \times B(t) \pmod{F(t)}$ that does not require reduction modulo $F(t)$ but only three multiplications followed by some shifts and additions in the last clock cycle, when $F(t)$ has a special form – A fully interleaved reduction-free modular multiplier.

2 Preliminaries

This section provides an overview of the notation used throughout in the paper, concerning modular reduction with finite rings and fields where the elements are represented in the polynomial basis.

Let $\mathbb{Z}_p[t]/(F(t))$ denote the set of polynomials in variable t with coefficients over p, defined by a monic $F(t)$ of degree n which forms a finite field, for p prime and $F(t)$ irreducible modulo p; and a ring, $\mathcal{R}_{q=p^n}$, otherwise. We express $F(t)$ as $\sum_{i=0}^{n} = f_n t^n + f_{n-1}t^{n-1} + f_{n-2}t^{n-2} + \cdots + f_2 t^2 + f_1 t + f_0 t^0$ where the f_i are the coefficients. The same notation is used for all other polynomials, strings, and data registers in hardware to refer to the ith coefficient or data bit. Let f_i^{-1} denote the multiplicative inverse of $f_i \pmod{p}$. An element $A(t)$ in any such structure is expressed as $A(t) = \sum_{i=0}^{n} = a_{n-1}t^{(n-1)} + a_{n-2}t^{(n-2)} + \cdots + a_1 t + a_0 t^0$. Then, the product of two elements is $A(t) \times B(t)$, is expressed as $C(t) = A(t) \times B(t) \pmod{p, F(t)}$. Arithmetic concerning coefficients of elements $A(t)$ and $B(t)$ conforms to $\pmod{p}$. Polynomial arithmetic conforms to $\pmod{F(t)}$.

Let $C_{max} = 2(n-1)$ denote the maximum degree of a product of two elements in $\mathcal{R}_q$. Let F_d be the degree of defining polynomial $F(t)$ and C_d the degree of an arbitrary product of two elements, $C(t) = A(t) \times B(t) \in \mathcal{R}_q$. Then, $\rho = C_d - (F_d - 1)$ denotes the number of reductions required to reduce a product of degree C_d to become representable in the field or ring.

For a hardware register or string holding $A(t)$ of depth $k = 2 \times r$, let $A(t)_{[j:i]}$ denote the bit range of data from j to i inclusive, from the jth most significant bit to the ith least significant bit. For a register A of width $k = 2 \times r$, let A_{UR} denote the upper register $A_{[k-1:r]}$ and A_{LR} denote the lower register $A_{[r-1:0]}$. Similarly, $||$ denotes concatenation of strings. For example, $C = A||B$, would refer to the equivalent SystemVerilog assignment $C = \{A, B\}$ for C of width k and A and B both of width r.

Let $GF(p^n)$ and $\mathbb{F}_q$ with $q = p^n$, be a finite field for p prime and defining polynomial $F(t)$ of degree n irreducible in modulo p. Our work is primarily concerned with Galois fields of the form $GF(2^n)$ and is easily adaptable to rings where the coefficients vary over $\mathbb{Z}_p$. We are interested in prime odd binary curves of degree n that define different binary fields $GF(2^n)$ for varying $F(t)$. We use the notation $GF(2^n)$, $(n+1)$-bit curve, and $k = (n+1)$-bit field interchangeably, where $(n+1)$-bit curve refers to $F(t)$ and k-bit field to $GF(2^n)$ which is defined by k-bit $F(t)$.

3 Reduction with Polynomial Rings

Efficient arithmetic over polynomial rings and fields $\mathbb{Z}_p[t]/F(t)$ has originated from usual modular arithmetic routines. If polynomial coefficients and words of multi-precision numbers are considered as the atomic units of their representations, polynomials enjoy carry-free arithmetic over these units. Multiplication involving creative representations make the arithmetic over polynomial rings quite interesting.

Modular multiplication involves multiplication and reduction steps; these are often implemented as separate algorithms or interleaved into one algorithm. Both type of implementations are characterized by the reduction technique employed. Modular reduction can applied from left-to-right, right-to-left, or both directions; corresponding examples to these would be Blakely, Montgomery, and Bipartite, respectively [3,16,25].

In this section, we go over the reduction techniques when given an already computed product $C(t) = A(t) \times B(t)$ with $A(t), B(t) \in \mathbb{Z}_p[t]/F(t)$ for some p and $F(t)$ a monic of degree n. We consider the product $C(t)$ of maximum degree $m = C_{max}$ and reduction $\pmod{q, F(t)}$.

3.1 Left-to-Right Reduction

Any algorithm that reduces a product $C(t)$ modulo $F(t)$ from the left falls into the left-to-right category, including the standard division algorithm.

Algorithm 1. Blakely Polynomial Reduction

```
Require: $C(t)$ of degree $m$
Ensure: $R(t) \equiv C(t) (\mathrm{mod} p, F(t))$
1: $R(t) = C(t)$
2: $j = m - n$
3: for $i = m$ downto $n$ do
4:     $q_j = r_i \bmod p$
5:     $R(t) = R(t) - q_j F(t) t^{i-n}$
6:     $j = j - 1$
7: end for
8: return $R(t)$
```

Algorithm 1 presents the Blakely reduction method from [3] which can be adapted to $GF(2^n)$ as in [20]. This is a bit-serial algorithm in which we loop $k = m - (n-1)$ times reducing the degree of $C(t)$ from the most significant position until $i = n$ and we obtain a residue of degree $\leq n-1$. In line 4 we compute the jth digit of the full quotient $Q(t)$ as the jth digit of $R(t)$ modulo p, starting from the most significant position. In line 5 we subtract an aligned multiple of the modulus, to continue reduction from left-to-right in each loop. The resulting residue satisfies the closed form of the division theorem, $R(t) = C(t) - F(t)Q(t)$.

3.2 Right-to-Left Reduction

Montgomery multiplication [25] demonstrates the unique example of right-to-left reduction which is shown in Algorithm 2.

Algorithm 2. Polynomial Montgomery Reduction

```
Require: $C(t)$ of degree $m$
Ensure: $S(t) \equiv C(t)(\mathrm{mod} p, F(t))$
 1: $S(t) = C(t)$
 2: for $i = 0$ to $k - 1$ do
 3:     $q_i' = f_0^{-1} s_0 \bmod p$
 4:     $S(t) = (S(t) - q_i' F(t))/t$
 5: end for
 6: return $S(t)$
```

The residue is computed in a bit-serial fashion, where we loop $k = m-(n-1)$ times, reducing the product from the least significant position, one degree per loop. In line 3 we compute the ith quotient of the Montgomery quotient, q_i', as the multiplicative inverse of the least significant coefficient of $F(t)$ multiplied with the least significant digit of $S(t)$, modulo p. In line 4, subtracting $q_i' F(t)$ from $S(t)$ sets the constant coefficient, s_0, to zero and hence the division by t corresponds to a trivial division or a right-shift. This is the reason why Montgomery reduction is preferred in most repetitive multiply-reduce designs. The result is a residue $S(t) = C(t)t^{-k} \pmod{F(t)}$ with degree $\leq n-1$.

3.3 Bipartite Reduction

The bipartite modular multiplication (BMM) method introduced by Kaihara and Takagi in [16], presents a method of modular reduction in which a left-to-right and a right-to-left technique can be applied in parallel to reduce a product from both ends simultaneously. This method is presented in Algorithm 3; for completeness we simply combine Algorithms 1 and 2.

Algorithm 3 executes in a sequential fashion but it loops $\rho = \lfloor n/2 \rfloor = \frac{m-(n-1)}{2}$ half the number of times as Blakely and Montgomery which require $\rho = m - (n-1)$ reductions. In lines 4–5, we compute the standard and Montgomery quotients, respectively. In line 6, we apply Blakely reduction to $S(t)$ and in line 7 we apply Montgomery reduction. Lines 7–8 can be implemented as separate threads in software or functional units in hardware executing in parallel. When the coefficients are over $\mathbb{Z}_p$ we must account for the Montgomery domain and set the parameter R to be less than the modulus.

4 Interleaved Modular Reduction

This sections builds on the previous section by interleaving multiplication of a product and reduction using a simpler structure, $GF(2^n)$. We present the

Algorithm 3. Bipartite Polynomial Reduction

```
Require: $C(t)$ of degree $m$
Ensure: $S(t) \equiv C(t)(\mathrm{mod} p, F(t))$
 1: $S(t) = C(t)$
 2: $k = \lfloor n/2 \rfloor$
 3: for $i = 0$ to $\lfloor n/2 \rfloor - 1$ do
 4:     $q_k = s_k \bmod p$
 5:     $q'_i = f_0^{-1} s_0 \bmod p$
 6:     $S(t) = S(t) - q_k F(t) t^{k-n}$
 7:     $S(t) = (S(t) - q'_i F(t))/t$
 8:     $k = k - 1$
 9: end for
10: return $S(t)$
```

corresponding interleaved modular multiplication algorithms for Blakely, Montgomery, and BMM.

4.1 Blakely

Algorithm 4. Interleaved Blakely

```
Require: $k$-bit F(t), $(k-1)$-bit A(t) and B(t)
Ensure: $R(t) \equiv A(t) \times B(t)(\mathrm{mod} F(t))$
 1: $R(t) = 0$
 2: for $i = k - 2$ downto 0 do
 3:     $R(t) = R(t) \ll 1$
 4:     if $a_i$ then
 5:         $R(t) = R(t) \oplus B(t)$
 6:     end if
 7:     if $r_{k-1}$ then
 8:         $R(t) = R(t) \oplus F(t)$
 9:     end if
10: end for
11: return $R(t)$
```

Algorithm 4 shows the adapted version of the original interleaved algorithm in the binary basis [3, 20]. Multiplication and reduction are interleaved using the standard shift and add technique. In this case we observe the bits of $A(t)$ starting from the most significant bit; if the bit is set, we multiply or add $B(t)$ to $R(t)$. Similarly, if the most significant bit of $R(t)$ is set, we reduce $R(t)$ with $F(t)$. After a multiplication and a reduction, we shift out the degree that has been knocked down.

4.2 Montgomery

Algorithm 5 shows the interleaved binary version adapted from [2]. If we are only concerned with multiplication, we observe the bit of the multiplier from either end. However, when we are performing interleaved multiplication, these bits must be observed according to the technique; for Blakely and Montgomery it is according to the direction we are reducing from. In this case, we test if the least significant bit of the residue is set and reduce it with $F(t)$. Lastly, we shift out the knocked-down degree.

Algorithm 5. Interleaved Mongtomery

```
Require: $k$-bit F(t), $n = (k-1)$-bit A(t) and B(t)
Ensure: $R(t) \equiv A(t) \times B(t) \times 2^{-n} (\text{mod} F(t))$
 1: $R(t) = 0$
 2: for $i = 0$ to $k-2$ do
 3:     if $b_i$ then
 4:         $R(t) = R(t) \oplus A(t)$
 5:     end if
 6:     if $r_0$ then
 7:         $R(t) = R(t) \oplus F(t)$
 8:     end if
 9:     $R(t) = R(t) \gg 1$
10: end for
11: return $R(t)$
```

4.3 Bipartite Modular Multiplication

Algorithm 8 adapts BMM from [16] to $GF(2^n)$. In hardware, the intermediate Blakely $S(t)$ and Montgomery $T(t)$ residues are computed in parallel. The algorithm uses Algorithm 6 and Algorithm 7 to compute interleaved modular multiplication with $A(t)$ and the upper and lower words of $B(t)$. For k-bit curves, we can compute the bipartite residue in $k/2$ CC without dependencies. The Blakely residue requires $r-1$ reductions in Algorithm 6 while the Montgomery residue requires r reductions in Algorithm 7 since BH is one bit less. BMM ensures a residue $R(t) \equiv (A(t) \times BH(t) \pmod{F(t)}) + (A(t) \times BL(t) \times 2^{-\lceil \frac{k-1}{2} \rceil} \pmod{F(t)})$ $(\text{mod } F)(t)$. This residue is also implicitly expressed as $A(t) \times B(t) \times 2^{-r}$ $(\text{mod } F(t))$ indicating the $r = \lceil \frac{k-1}{2} \rceil$ Montgomery degrees knocked down.

Algorithm 6. ibBlakely

```
Require: $A(t)_{[k-2:0]}$, $BH(t)_{[r-2:0]}$, $F(t)_{[k-1:0]}$
Ensure: $S(t) \equiv A(t) \times BH(t) (\bmod F(t))$
 1: $S(t) = 0$
 2: for $i = r - 2$ downto $0$ do
 3:     $S(t) = S(t) \ll 1$
 4:     if $bh_i$ then
 5:         $S(t) = S(t) \oplus A(t)$
 6:     end if
 7:     if $s_{k-1}$ then
 8:         $S(t) = S(t) \oplus F(t)$
 9:     end if
10: end for
11: return $S(t)$
```

Algorithm 7. ibMontgomery

```
Require: $A(t)_{[k-2:0]}$, $BL(t)_{[r-1:0]}$, $F(t)_{[k-1:0]}$
Ensure: $T(t) \equiv A(t) \times BL(t) \times 2^{-r} (\bmod F(t))$
 1: $T(t) = 0$
 2: for $i = 0$ to $r - 1$ do
 3:     if $bl_i$ then
 4:         $T(t) = T(t) \oplus A(t)$
 5:     end if
 6:     if $t_0$ then
 7:         $T(t) = T(t) \oplus F(t)$
 8:     end if
 9:     $T(t) = T(t) \gg 1$
10: end for
11: return $T(t)$
```

Algorithm 8. BMM

```
Require: $k$-bit $F(t)$, $(k-1)$-bit $A(t)$, $B(t)$, $r = \lceil (\frac{k-1}{2}) \rceil$
Ensure: $R(t) \equiv A(t) \times B(t) \times 2^{-r} \pmod F(t)$
 1: $S(t) = ibBlakely(A(t), B(t)_{[k-2:r]}, F(t))$
 2: $T(t) = ibMontgomery(A(t), B(t)_{[r-1:0]}, F(t))$
 3: $R(t) = S(t) \oplus T(t) \oplus F(t)$
 4: return $R(t)$
```

5 Partially Interleaved Karatsuba-Ofman

Modular multipliers fall under one of two types: multiply and reduce or interleaved multiply and reduce. Varying implementations of Montgomery, BMM, and Mastrovito are among the most efficient interleaved modular multipliers introduced [14,24]. In general, interleaving fast multi-digit multipliers, such as Schönhage-Strassen or Fürer is difficult and application-specific [12,34]. For example, Fürer is intended for huge numbers in the order of 10^{82} and becomes efficient when the operands are in that order. In this section, we present the Karatsuba-Ofman Algorithm (KOA) and the work from [33], a Partially Interleaved Karatsuba-Ofman (PIKO) modular multiplier.

5.1 Karatsuba-Ofman Multiplication

KOA is a recursive divide and conquer algorithm based on the observation by Babbage that two n-digit numbers can be expressed as binomials and multiplied out likewise requiring four multiplications [1,19]. In 1962, Karatsuba and Ofman observed that the middle term could actually be computed in one multiplication with some additions and subtractions from already computed terms requiring only three multiplications total. KOA is intended for very large numbers in the order of several thousand digits ranging from 10^3 to 10^4 digits with complexity $O(n^{log_2(3)})$ in the input size.

The algorithm is quite generic that it can easily be adapted to polynomial multiplication for binary fields where operands range from a few thousand bits to a few hundred thousand bits. In practice, KOA is used in conjunction with reduction routines such as Blakely or Montgomery [3,25]. Recursion can be set to any desired level. However, further recursion and improved KOA must account for platform constraints and is completely application-specific.

Now, consider two arbitrary polynomials $A(t) = a_p t^m + \cdots + a_2 t^2 + a_1 t + a_0$ and $B(t) = b_q t^n + \cdots + b_2 t^2 + b_1 t + b + 0$ of degrees m and n respectively, and without loss of generality, let $m \geq n$ and $r = \lfloor m/2 \rfloor$. For simplicity, let's say they are both expressed as the closest power of two and are split into half-size equal words. Let:

$$\begin{aligned}A(t) &= A_1(t) \cdot t^r + A_0(t),\\ B(t) &= B_1(t) \cdot t^r + B_0(t)\end{aligned}$$

where, $A_1(t)$ and $A_0(t)$ represent the upper and lower words of the polynomial $A(t)$, each of degree r. Standard multiplication of $A(t)$ and $B(t)$ can expressed as follows:

$$\begin{aligned}C(t) &= A(t) \cdot B(t)\\ &= (A_1(t)t^r + A_0(t))(B_1(t)t^r + B_0(t))\\ &= (A_1(t)B_1(t))t^{2r} + (A_1(t)B_0(t) + A_0(t)B_1(t))t^r\\ &\quad + A_0(t)B_0(t)\\ &= C_2(t)t^{2r} + C_1(t)t^r + C_0(t).\end{aligned}$$

Notice that, the above calculation requires four different polynomial multiplications with operands of degree r. This has quadratic complexity in operand size. KOA however, achieves the same computation with only three multiplications as follows:

$$\begin{aligned}
C_0(t) &= A_0(t) \cdot B_0(t) \\
C_2(t) &= A_1(t) \cdot B_1(t) \\
C_1(t) &= (A_0(t) + A_1(t))(B_0(t) + B_1(t)) - C_0(t) - C_2(t) \\
C(t) &= C_2(t)t^{2r} + C_1(t)t^r + C_0(t).
\end{aligned}$$

5.2 Interleaving Karatsuba-Ofman and Bipartite Reduction

In [33] Saldamlı et al. present PIKO, consisting of Karatsuba-Ofman multiplication and bipartite reduction circuits. The algorithm considers only the first layer of recursion requiring half-size words. In this section, we give a general overview of the algorithm as follows.

Let $F(t) = x^n + x^{\lfloor \frac{n}{2} \rfloor} + 1$ the defining polynomial of degree n for some odd prime curve with coefficients defined over $GF(2)$. Now, consider two n-bit elements $A(t)$ and $B(t)$ in the field, both of maximum degree $n - 1$. First, we prefix the elements to consists of $k = n + 1$ bits. This allows all operands $F(t)$, $A(t)$, and $B(t)$ to be split into equal half-size words of size $r = \frac{k}{2}$ bits. Note that the maximum degree for the lower word of an element is $d = r - 1$ but for the upper word it is $r - 2$. For uniformity, because we are working with r-bit registers, the operands are decomposed as follows.

$$\begin{aligned}
F(t) &= F_1(t) \cdot t^d + F_0(t), \\
A(t) &= A_1(t) \cdot t^d + A_0(t), \\
B(t) &= B_1(t) \cdot t^d + B_0(t), \\
Q(t) &= Q_1(t) \cdot t^d + Q_0(t), \\
Q'(t) &= Q'_1(t) \cdot t^d + Q'_0(t).
\end{aligned}$$

where $Q(t)$ and $Q'(t)$ represent the Blakely and Montgomery quotients.

Figure 1 illustrates the PIKO algorithm. The algorithm is the same as BMM except that multiplication is done using KOA. In the upper part, we compute the Karatsuba-Ofman terms $C_0(t)$, $C_1(t)$, and $C_2(t)$; the middle term is partially computed. In the middle part, we compute bipartite terms consisting of bipartite quotients and bipartite products. The quotients $Q'_0(t)$ and $Q_1(t)$ are computed using fully interleaved Montgomery and Blakely algorithms applied to the products $C_0(t)$ and $C_2(t)$ which are reduced with $F_0(t)$ and $F_1(t)$, respectively.

The reduction terms are then $(Q'_0(t)F_0(t))$, $(Q'_0(t) + Q_1(t))(F_0(t) + F_1(t))$, and $(Q_1(t)F_1(t))$. These terms are applied to $C_0(t)$, $C_1(t)$, and $C_2(t)$ to produce reduced products $C'_0(t)$, $C'_1(t)$, and $C'_2(t)$. Lastly, the final sum can be computed in different ways. In Fig. 1, the final sum is simply $S(t) = C'_{2LR}(t)||C'_{0UR}(t) + C'_1(t) + C'_{2LR}(t)||C'_{0UR}(t)$. This gives us a residue $A(t) \times B(t) \times t^{-r} \pmod{F(t)}$. The term t^{-r} is equivalently expressed as $(t^r)^{-1} = 2^{-r} = (2^r)^{-1}$ since in the binary base, t^{-r} is a shift right by r indicating the Montgomery degrees knocked down.

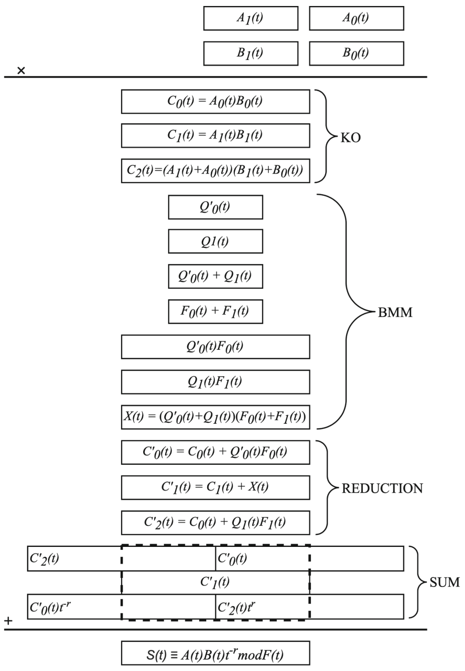
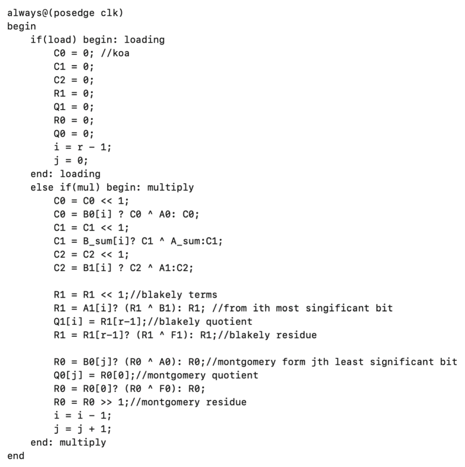


Fig. 1. Partially interleaved Karatsuba-Ofman algorithm.

If we compute all terms in PIKO in a bit-parallel fashion, bit-by-bit per clock cycle, it becomes fully interleaved. Then, a residue of k-bit operands can be computed in $\frac{k}{2}$ clock-cycles. We call this version the Fully Interleaved Karatsuba-Ofman (FIKO) algorithm. This follows by noting that when we compute a product $A \times B$, the bits of a multiplier B, can be observed from either the least significant or most significant position. In this manner, we can compute the Blakely and Montgomery quotients from opposite ends. Close attention to the half-size products, reveals no dependencies in this approach. For example, the product of two upper words $A_1(t) \times B_1(t)$ will always have $C_{max} = 2(r-2)$ because they are prefixed. Consequently, the most significant $(k-1) - C_{max}$ bits of $C(t)_2$ will always be zero. The upper bits of the product $C_2(t)$ become fixed as the amount by which we shift $A_1(t)$ by decreases.

The remaining multiplications such as $Q_1(t)F_1(t)$, can all be accomplished using shifts and adds, according to $F(t)$. The three types of $F(t)$ used are discussed in Sect. 7. Figure 2 shows the core of the FIKO bit-parallel algorithm in SystemVerilog. $C0$, $C1$, $C2$ correspond to KOA terms; $R1$ and $R0$ are the Blakely and Montgomery residues; and $Q1$ and $Q0$ are the Blakely and Montgomery quotients.



```
always@(posedge clk)
begin
    if(load) begin: loading
        C0 = 0; //koa
        C1 = 0;
        C2 = 0;
        R1 = 0;
        Q1 = 0;
        R0 = 0;
        Q0 = 0;
        i = r - 1;
        j = 0;
    end: loading
    else if(mul) begin: multiply
        C0 = C0 << 1;
        C0 = B0[i] ? C0 ^ A0: C0;
        C1 = C1 << 1;
        C1 = B_sum[i]? C1 ^ A_sum:C1;
        C2 = C2 << 1;
        C2 = B1[i] ? C2 ^ A1:C2;

        R1 = R1 << 1;//blakely terms
        R1 = A1[i]? (R1 ^ B1): R1; //from ith most singificant bit
        Q1[i] = R1[r-1];//blakely quotient
        R1 = R1[r-1]? (R1 ^ F1): R1;//blakely residue

        R0 = B0[j]? (R0 ^ A0): R0;//montgomery form jth least significant bit
        Q0[j] = R0[0];//montgomery quotient
        R0 = R0[0]? (R0 ^ F0): R0;
        R0 = R0 >> 1;//montgomery residue
        i = i - 1;
        j = j + 1;
    end: multiply
end
```

Fig. 2. Fully interleaved Karatsuba-Ofman algorithm core.

6 Reduction-Free FIKO

In this section we develop the FIKO algorithm into a reduction-free version. First, we present the special form of the defining polynomial which allows us to accomplish this.

6.1 Equally Spaced Polynomials

In [14], Koç provides a concise definition of Equally Spaced Polynomials (ESPs) and Equally Spaced Trinomials (ESTs). An ESP with degree $n = \delta k$ has form $x^{\delta k} + x^{\delta(k-1)} + \cdots + x^{\delta} + 1$ and is necessarily of even degree with all non-zero terms equally spaced by $\delta - 1$ zero terms. For example, $x^{\delta k} + x^{\delta(k-1)} + x^{\delta(k-2)} = x^6 + x^3 + x^0 = 1001001_2$ for $\delta = 3$ and $k = 2$, the terms are equally spaced by $\delta - 1 = 2$ zero terms. Similarly, $x^4 + 1$ is an ESP. A special case is the All-One-Polynomial (AOP), in which case $\delta = 1$. For example, $x^{(k-1)} + x^{(k-2)} + x^{(k-3)} + x^{(k-4)} + x^{(k-5)} + x^{(k-6)} = 1111111_2$ with $k = 6$. An EST is a trinomial with all non-zero terms equally spaced and necessarily of even degree; for example $x^4 + x^2 + 1 = 10101_2$.

6.2 Reduction-Free Trinomials

Ideally, we would like to work with an intermediary binomial or an EST where we can reduce a product from both ends in parallel without dependencies. However, for our work, ESTs cannot be used as they have an even degree, cannot be equally split, and have security considerations. Now, when the defining polynomial is an ESP-like trinomial of form $x^n + x^r + 1$ with n odd and $r = \lceil n/2 \rceil$, we can enjoy reduction-free multiplication. We define this special polynomial as a Reduction-Free Trinomial (RFT). Such polynomials can be split into equal half-size r-bit registers and are of necessarily odd prime degree. This trinomial is easily characterized as one whose lower register is a 1 and whose upper register is an ESP of desired binomial form. For symmetry, we zero-extend the input operands and work with r-bit registers.

When we work with half-size operands and an RFT, careful observation of the computations and results in FIKO reveal that the bipartite reduction circuits can be removed. The following example shows this.

Example 1. For simplicity, consider $GF(2^9)$, $F(t)$ an RFT that defines the field, and two elements in the field:

$$F(t) = t^9 + t^5 + 1$$
$$A(t) = t^8 + t^7 + t^6 + t^5 + t^2 + 1$$
$$B(t) = t^8 + t^5 + t^3 + t + 1.$$

The corresponding half-size words with the prefixed elements are given in Table 1.

Table 1. Half-size parameters

Operand	Upper register	Lower register
$F(t)$	$F_1(t) = 10001$	$F_0(t) = 00001$
$A(t)$	$A_1(t) = 01111$	$A_0(t) = 00101$
$B(t)$	$B_1(t) = 01001$	$B_0(t) = 01011$

If we compute FIKO as usual, we would obtain all Karatsuba-Ofman products, bipartite terms, reduced terms, and final sum. However, since $F_0(t) = 1$, we can observe that the lower word of the Montgomery quotient is just the lower register of $C_0(t)$ since $a \pmod 1$ is always a. Similarly, we observe that the Blakely quotient for the entire product $C_2(t)$ is just the most significant $[k-2:r-1]$ bits of $C_2(t)$. Note that the Blakely quotient will always fit in less than r-bits. Because we are working with half-size words and we are only concerned with the upper register of the standard quotient $Q(t)$, we can easily see that this is just $C_2(t)[k-2:r-1]$. Because $A_1(t)$ and $B_1(t)$ are prefixed and their product is of degree at most $2(r-2)$, it can be observed that the $(k-1)-2(r-2)$ bits of $C_2(t)$ will always be zero and that the upper register of the quotient is found in the specified bits. However, the full upper register of $C_2(t)_{[k-1:r]}$ can be taken as $Q_1(t)$ if we multiply it by t or shift it left by one. This is illustrated in Fig. 3;

the left side shows RF-FIKO quotients taken directly from $C_0(t)$ and $C_2(t)$ and the right side shows the quotients we obtain after we apply bipartite reduction. They are the same.

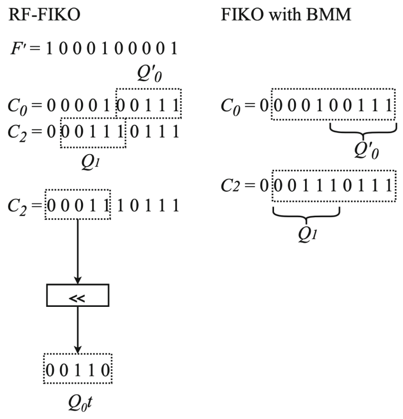


Fig. 3. RFFIKO and FIKO with a RFT.

There is no need for the Blakely and Montgomery reduction circuits, we can just take the quotients from the Karatsuba-Ofman terms $C_0(t)$ and $C_2(t)$. RF-FIKO, illustrated in Fig. 4, reduces to a total of three distinct multiplications, namely those to compute the Karatsuba-Ofman products $C_0(t)$, $C_1(t)$, and $C_2(t)$. The bipartite products and reduction terms are all computed with shifts and additions on the last clock cycle. In Example 1, our desired answer $C(t) = (C_2' t^{2r} + C_0' + C_1' t^r + C_0' t^{2r} + C_2')t^{-r}$ is 1000010100. We can easily obtain the result in the standard domain by re-adding the r Montgomery zeroes (degrees we knocked down) to this bipartite residue and employing standard reduction. Our result is $C(t)t^{-r} \pmod{F(t)} \equiv A(t)B(t)(t^r)^{-1} \pmod{F(t)}$. Now, from closer inspection of Fig. 4, we can see that $Q_0'(t)F_0(t) = Q_0'(t)$ since $F_0(t) = 1$.

For an RFT, the cross term is simply computed as the sum of the quotients with multiplication by $(F_1(t) \oplus F_0(t))$ implemented as a shift left by $r-1$. Hence, we have a fully interleaved reduction-free modular multiplier with a total of three half-size word multiplications and six sums (two k-bit sums, three r-bit sums, and one 1-bit sum). The sums can be implemented differently, for example to reduce an r-bit sum to a 1-bit sum at the cost of space. In either case, the metrics are the same and we kept the original implementation with three k-bit sums and three r-bit sums.

7 Test Inputs

We consider finite fields of the form $GF(2^n)$ where the degree of the defining polynomial corresponds to an ECDSA binary field or a Mersenne exponent [6,13,

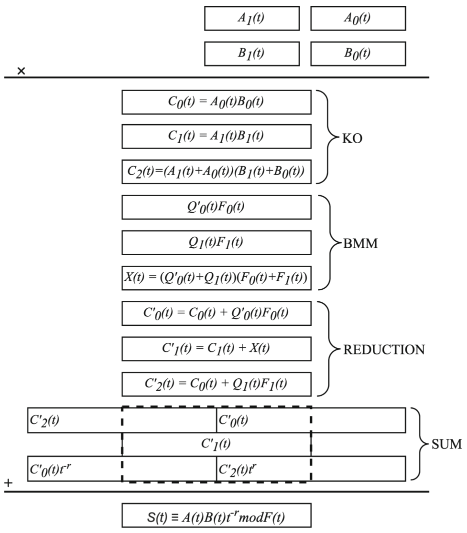


Fig. 4. Reduction-free fully interleaved Karatsuba-Ofman (RF-FIKO) algorithm.

29,38]. We consider three different types of trinomials, namely, RFTs of special form $x^n + x^{\lceil n/2 \rceil} + 1$, special trinomial type 1 of form $x^n + x^{\lfloor n/2 \rfloor} + 1$ and special trinomial type 2 of form $x^n + x + 1$.

Table 2. Test curves

Curve type	Exponents
ECDSA	163, 233, 283, 409, 571
Mersenne	107, 127, 521, 607, 1279, 2203, 2281, 3217

Table 2 shows the curves used and Table 3 shows our test field groups. For example, test group $F2$ from Table 3 consists of finite fields of the form $GF(2^n)$ where n varies over the ECDSA exponents listed in table Table 2 and the defining polynomial for all such fields is an RFT. More explicitly, the $F2$ group consists of fields: $GF(2^{163})$, $GF(2^{233})$, $GF(2^{283})$, $GF(2^{409})$, and $GF(2^{571})$ each of which is defined by a corresponding RFT $F(t) = t^{163} + t^{82} + 1$, $F(t) = t^{233} + t^{117} + 1$, $F(t) = t^{283} + t^{142} + 1$, $F(t) = t^{409} + t^{205} + 1$, and $F(t) = t^{571} + t^{286} + 1$, respectively. RF-FIKO is completely defined by an RFT and hence, we only test it with finite fields in groups $F1$ and $F2$. Blakely, Montgomery, FIKO, and BMM are independent of the defining polynomials and can be tested with all test groups.

Table 3. Test groups

Group	Curves	$F(t)$
$F1$	Mersenne	RFT
$F2$	ECDSA	RFT
$F3$	Mersenne	Type 1
$F3'$	ECDSA	Type 1
$F4$	Mersenne	Type 2
$F4'$	ECDSA	Type 2

8 Hardware Implementation

In this section, we present two bit-parallel hardware implementations in $GF(2^n)$. The algorithms implemented were Interleaved Montgomery, Interleaved Blakely, BMM, FIKO, and RF-FIKO. We implemented in Verilog and SystemVerilog and prototyped on a Virtex-7 FPGA. The ASIC design synthesis was done using TSMC 28-nm CMOS technology.

8.1 Non-recursive Decomposition

Figure 5 details the microarchitecture for RF-FIKO corresponding to our algorithm in Fig. 4. A direct and naïve implementation of Fig. 4 would yield a residue in k-CC with the KOA terms computed in r-CC followed by the bipartite reduction products in r-CC. However, a bit-parallel implementation allows for a fully interleaved implementation with all terms computed in parallel, one bit per clock cycle, in r-CC. Moreover, close inspection of Fig. 4 and the bipartite terms, allows us to be concerned only with the KOA products of $C_0(t)$, $C_1(t)$, and $C_2(t)$ since the bipartite and reduction terms can all be computed with shifts and additions. The critical delay path is then the multiplication of the three KOA terms for which there exist various techniques for improvement.

The fastest bit-parallel implementations are concerned with special trinomials and exploiting structure to compute the product $A(t) \times B(t) \mod F(t)$ while making efforts to simplify the reduction. We have explored the mathematical structure of the RFT together with noted observations so that we can solve the problem $A(t) \times B(t) \mod F(t)$ with only three multiplications, $C_0(t)$, $C_1(t)$, and $C_2(t)$, and obtain a true reduction-free residue. Because of the nature of the RFT, we obtain reduction-free quotients and compute the remaining terms with some shifts and adds.

This work sets forth an initial presentation of RF-FIKO concerned with the upper layer of recursion of Karatsuba-Ofman to compute the three multiplications using half-size operands. Further KOA layers imply more space and platform constraints. This decomposition was selected as an initial step to improve PIKO in time and space complexity and hence, obtain improvement over the

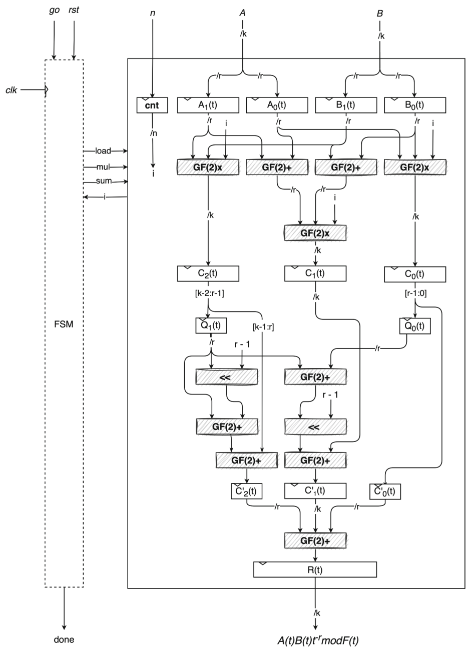


Fig. 5. RF-FIKO microarchitecture

bipartite reduction technique by completely removing it. Moreover, implementations that apply non-recursive KOA (only the upper layer), usually use 1/4 less space compared to the fastest implementations [8–10, 15, 22, 35–37]. Further recursion and incorporation with other techniques is reserved for future work.

8.2 Register-Transfer Level Design

Close observation of Fig. 5 shows that the KOA terms are computed in a bit-parallel fashion. The data path is controlled by a four state Moore machine. In state zero all data registers are loaded with the value of zero and the down counter, *cnt*, is loaded with $r - 1$. In state two, the multiplication control signal controls the bit-parallel multiplication of $C_0(t)$, $C_1(t)$, and $C_2(t)$ to compute them one bit per clock cycle. Sketched rectangles highlight the three different arithmetic operations; the symbols $GF(2)+$, $GF(2)x$, and $\ll$ correspond to $GF(2)$ addition, $GF(2)$ multiplication, and shift left by $r - 1$. These operations are implemented as XORs, shift and add multiplication, and arithmetic left shift to multiply by a power of two. When *cnt* reaches zero, the multiplication is complete and the final *sum* and *done* signals are generated by the FSM. The remaining RTL details how the final sum is computed; all such registers are assigned inside a procedural block.

The r-bit quotients are assigned as $Q_1(t) = C_2(t)_{[k-2:r-1]}$ and $Q_0(t) = C_0(t)_{[r-1:0]}$. The reduced terms $C'_0(t)$ and $C'_2(t)$ are implemented as r-bits. The final sum only concerns the upper register of $C'_0(t)$ and hence, there is no need to compute the full $C'_0(t)$ term. Moreover $Q'_0(t)F_0(t)$ reduces to $Q'_0(t)$ for $F_0(t) = 1$ which does not affect the upper register of $C'_0(t)$. We let $C'_0(t) = C_0(t)_{[k-1:r]}$. Similarly for $C'_2(t)$, we implement $Q_1(t)F_1(t)$ as a left shift by $r-1$ added with Q_1, and take only the lower register of $C'_2(t) = C_2(t) \oplus (Q_1(t) \ll (r-1)) \oplus Q_1(t)$. For $C'_1(t)$ we require the full term, and implement the cross-term $X(t)$ as $(Q_1(t) \oplus Q_0(t)) \ll (r-1)$ since multiplication by $(F_1(t) \oplus F_0(t))$ reduces to a shift left by $r-1$. Finally, our sum of concatenated terms, is $\Sigma = C'_0(t)||C'_2(t) \oplus C'_1(t) \oplus C'_2(t)||C'_0(t)$ which corresponds to the final output in Fig. 5, $A(t)B(t)t^{-r} \pmod{F}(t)$.

9 Results

In this section we provide results for both FPGA and ASIC hardware implementations for sample ECDSA and Mersenne curves. In both implementations, Blakely and Montgomery form our baseline for comparison. Bipartite reduction is the fastest reduction technique prior to this work. Our main targets for comparison are BMM and FIKO for half-size words. A brief comparison with some of the fastest bit-parallel multipliers in the field is also provided along with estimates for FHE curves.

9.1 FPGA Results

Table 4 shows the FPGA results for sample curves in different groups–namely, the clock cycle (CC) count, LUT count, slices, frequency (MHz), and the latency in clock cycles $\times$ clock period (μs). For each sample field $GF(2^n)$, we list the defining polynomial $F(t)$ and the results for each algorithm. Blakely and Montgomery show some slight variance in all metrics. In comparison to the other three algorithms, these approximately double in the execution time but half the space used. FIKO outperforms BMM for $GF(2^{107})$ and $GF(2^{163})$ curves using several tens to hundreds more LUTs and slices. For the non-RFT curves, BMM and FIKO behave similar. As expected, RF-FIKO in turn outperforms FIKO which was our expected goal. For $GF(2^{107})$, RF-FIKO is 1.026 times faster than FIKO and 1.071 times faster than BMM using approximately half the LUTs. For $GF(2^{163})$ RF-FIKO computes the residue 1.064 times faster than FIKO and 1.076 times faster than BMM. This is more easily observed in Fig. 6 which shows the execution time for all algorithms for sample fields.

9.2 ASIC Results

Table 5 shows the results from our ASIC implementation. As expected, the results are significantly faster with the largest curve attaining a period of 0.41 ns for Montgomery. For 108-bit and 164-bit curves, FIKO outperforms BMM in

Table 4. FPGA sample results

Algorithms	CC	LUT	$Slices$	$f\ (MHz)$	$Latency\ (\mu s)$
$F1 : F(t) = t^{107} + t^{54} + 1$					
Blakely	107	179	64	299.4	0.357
Montgomery	107	177	61	277.8	0.385
BMM	54	344	114	327.9	0.165
FIKO	54	400	150	342.5	0.158
RF-FIKO	54	235	108	350.8	0.154
$F2 : F(t) = t^{163} + t^{82} + 1$					
Blakely	163	260	78	268.1	0.608
Montgomery	163	262	73	268.8	0.606
BMM	82	477	142	286.5	0.286
FIKO	82	768	259	289.9	0.283
RF-FIKO	82	512	212	308.6	0.266
$F_3 : F(t) = t^{521} + t^{260} + 1$					
Blakely	521	819	218	218.3	2.39
Montgomery	521	813	230	216.0	2.41
BMM	261	1492	435	219.3	1.19
FIKO	261	2,685	922	203.7	1.28
$F_4' : F(t) = t^{571} + t + 1$					
Blakely	571	893	238	215.1	2.66
Montgomery	571	895	264	207.9	2.75
BMM	286	1,632	491	215.1	1.33
FIKO	286	2,772	942	199.7	1.432

time but not space. For 108-bit and 164-bit curves RF-FIKO outperforms BMM in time and space. For 108-bit curves, BMM computes a residue in 54 CC $\times$ 0.36 ns $\times$ 10^{-3} = 0.01944 µs requiring 5,512 gates. RF-FIKO computes the residue in 54 CC $\times$ 0.32 ns $\times$ 10^{-3} = 0.01728 µs with 4,876 gates. RF-FIKO differs by 2.16 ns and is more optimal in space with 636 fewer gates. In terms of execution time, RF-FIKO is 19.44 ns / 17.28 ns = 1.125 times faster than BMM when working in $GF(2^{107})$. For 164-bit curves, BMM computes a residue in 0.03116 µs at the cost of 6551 gates and RF-FIKO in 0.02788 µs with 7059 gates. In this case RF-FIKO requires 508 more gates but is 31.16 ns / 27.88 ns = 1.118 times faster than BMM when working in $GF(2^{163})$. Figure 7 shows the latency for 108 and 164-bit curves in clock cycle count by clock cycle time in μs for all algorithms. For other curves, FIKO behaves similar to BMM.

Table 5. ASIC sample results

$Algorithm$	CC	$Time(ns)$	$Gate\#$	$Area * time$
$F1 : F(t) = t^{107} + t^{54} + 1$				
Blakely	107	0.33	1,941	640.53
Montgomery	107	0.37	2,039	754.43
BMM	54	0.36	5,512	1,984.32
FIKO	54	0.34	7,812	2,656.08
RF-FIKO	54	0.32	4,876	1,560.32
$F2 : F(t) = t^{163} + t^{82} + 1$				
Blakely	163	0.34	3,115	1,059.1
Montgomery	163	0.38	3,835	1,457.3
BMM	82	0.38	6,551	2,489.38
FIKO	82	0.34	12,402	4,216.68
RF-FIKO	82	0.34	7,059	2,400.06
$F_3 : F(t) = t^{521} + t^{260} + 1$				
Blakely	521	0.4	9,240	3,696
Montgomery	521	0.41	11,785	4,831.85
BMM	261	0.39	20,690	8,069.1
FIKO	261	0.41	41,007	16,812.87
$F_4' : F(t) = t^{571} + t + 1$				
Blakely	571	0.4	11,031	4,412.4
Montgomery	571	0.41	12,399	5,083.59
BMM	286	0.39	22,370	8,724.3
FIKO	286	0.41	43,346	17,771.86

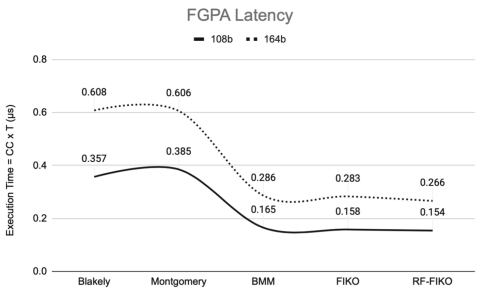


Fig. 6. FPGA execution time in CC $\times$ T (μs) for 108 and 164-bit curves

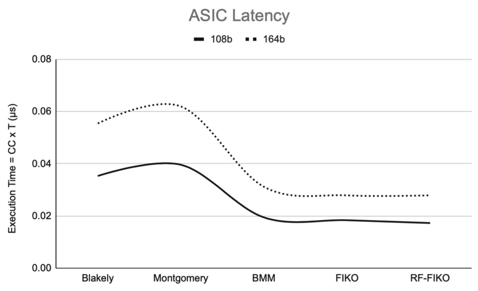


Fig. 7. ASIC execution time in CC $\times$ T (μs) for 108 and 164-bit curves

RF-FIKO removes the reduction part in modular multiplication simplifying the solution to the problem $A(t) \times B(t) \pmod{F(t)}$ to three multiplications followed by some shifts and additions on the last clock cycle; the results presented showed improvement in time and space complexity for specific curves. Further optimization can be obtained through experimentation and incorporation with other techniques such as, varying parameters, Mastrovito matrix for KOA terms, higher radices (e.g. 2^2, 2^3, etc.), refined and combined versions of KOA (e.g., with Toom-Cook), and pipelining.

Both FPGA and ASIC results show that for k-bit fields, Montgomery and Blakely can compute the residue in k CC, our main metric for time. BMM, FIKO, and RF-FIKO compute in $k/2$ CC. The focus of this work was to highlight the impact of our modulus polynomial. Hence, for comparison and additional consideration noted earlier, all algorithms were implemented in the same base and in a similar fashion. Results from state-of-the-art implementations, such as [16] and [17] for BMM, confirm similar results with respect to clock cycle count. For example in [17], for a radix-4 pipelined implementation, the residue is computed in $\frac{n}{2}+4$ CC for n-digit operands. A strict comparison in terms of time and space for state-of-the-art implementations would require reproducing such works and is considered for future work as there are different optimization levels that can be applied. Moreover, besides the several variants and considerations for comparison, such works also conform to particular modulus polynomials [21,23].

9.3 Bit-Parallel Multipliers and Fully Homomorphic Encryption

This subsection provides a brief comparison against some of the fastest bit-parallel modular multipliers and estimates for FHE curves. Table 6 lists the total gate count for similar works in the PB and Shifted PB. RF-FIKO space complexity consists of 7059 total gates for $GF(2^{163})$ which is six times less gates compared to [9,36,37] and four times less gates than [35]. The estimated time delay as a function of the signal propagation delay through total AND-gates (T_A) and XOR-gates (T_X) is expected to be significant.

Table 6. Bit-parallel multipliers in PB and SPB

$GF(2^{147})$		
Work	*Gate#*	*Time*
Sunar & Koç, Wu [36,37]	43,217	$T_A + 10T_X$
Elia et al. [8]	33,045	$T_A + 11T_X$
Fan et al. [9]	43,217	$T_A + 9T_X$
Negre [28]	49,000	$T_A + 8T_X$
Li et al. [35]	29,154	$T_A + 8T_X$

On the other hand, the degrees of defining polynomials for FHE are much larger than those of PQC, demanding more parallelism. Several functions with fully-homomorphic properties work with polynomial rings of the form $\mathbb{Z}_q[t]/(t^n + 1)$. These rings are applied to the FHE algorithms such as, CKKS and RLWE BVG [4,7]. For RLWE BGV, the moduli can take values in the intervals $q \in [2^{15}, 2^{500}]$ and $n \in [2^9, 2^{14}]$. CKKS works with polynomial rings of the form $\mathbb{Z}[t]/\phi(t)_m$ for which the defining polynomial $\phi(t)_m$ is the m-th cyclotomic polynomial for $m \in \mathbb{Z}^+$ and a power of 2. It also works with a ring $\mathbb{Z}_p[t]/\phi(x)_m$. Table 7 lists estimates for RF-FIKO for sample n for $F(t)$ that define rings for RLWE BVG, CKKS, and similar functions such as, FV and BFV [5,11]. The estimates for total gate count are based on the results for RF-FIKO for $GF(2^{163})$, as $\lceil n/163 \rceil \times 7059$. The CC is $n/2$ as noted previously. We can easily see that for $GF(2^{512})$, BVG would require 22,174 gates which is approximately half of the gate count for three of the fastest bit-parallel multipliers listed in Table 6 for a much smaller field.

10 Applications

Ring and finite field multiplication forms the fundamental operation in cryptographic schemes. Public-key cryptography, symmetric key cryptography, FHE,

Table 7. Estimates for FHE curves [4,5,7,11]

n	*CC*	*Gate#*
RLWE BVG		
2^9	256	22,174
2^{14}	8,192	709,538
CKKS and similar		
2^{15}	16,384	1,419,076
2^{16}	32,768	2,838,152
2^{17}	65,536	5,676,303

and primitives based on these, such as KEMs and HMACs, all employ modular arithmetic. These schemes are particularly interested is fast modular multiplication with polynomial rings and fields defined over different bases [18]. When working in higher radices, intermediate operations may be performed in $GF(2^n)$ for efficiency, such as eliminating carry propagations. In the modern regime, our work is applicable to schemes, such as RSA and ECC (e.g. ECDSA) [29]. In the quantum regime, NIST PQC Third Round Lattice-based finalists which share similar construction based on structured lattices, would benefit from our work. FHE uses higher degrees than PQC and requires a higher level of parallelism [4,26].

The arithmetic in Lattice-based schemes consists of matrix algebra and finite field and ring arithmetic. Cryptographic hashing (SHA-3 and XOFs), randomness generation, and ring multiplication are among the most expensive computations in these schemes [30]. The Number Theoretic Transform (NTT) is employed in all Lattice-based schemes for fast ring and field multiplication except in Saber and NTRU. Tables 8 and 9 list sizes for keys and moduli used in modern and PQC schemes. Table 10 lists rings used in different PQC schemes. Further details regarding the schemes and varying instances can be obtained by accessing the specification documents of each submission [31].

10.1 NTT-Unfriendly Rings

The polynomial arithmetic techniques applied to Lattice-based schemes can be grouped into three categories, namely, NTT-friendly, NTT-unfriendly, and combinations of Karatsuba and Toom-N. Karatsuba-Ofman multiplication is particularly suitable when the defining polynomial of the ring has degree above 16 and within 256. Variants of Karatsuba and Toom-N are more efficient when the degree is above 256. These variants are particularly suited for NTT-unfriendly rings where the moduli are a power of two, $\mathbb{Z}_{2^m}[t]$ (e.g. Saber and NTRU).

NTTs can be adapted for NTT-unfriendly rings to obtain significant speed-up through new implementations of the schemes and techniques (e.g., layering) [18]. However, improving NTTs by reducing the number of modular reductions is a sought venue for improvement [27]. Moreover, NTTs may not be applicable in all use cases (e.g. compression in Saber). Depending on the implementation, platform, and techniques applied, a speed-up may not be possible [31]. For example, in AVX2, a software speedup was not possible for $n = 509$ with NTT of length-1024 due to selected strategy and vector layout [18]. A fast hardware implementation uses schoolbook multiplication and highlights the difficulties of implementing recursive structures in hardware, such as Toom-N [32].

10.2 Number Theoretic Transforms

Kyber, Dilithium, and Falcon use NTT-friendly rings. NTRU-HRSS is flexible and the latest specification allows for variants that use a prime q allowing for security and size trade-offs not present when q is a power of two [31]. Falcon, based on NTRU lattices, uses a prime modulus $q = 12289$ of special form

$q = ((k \times 2n) + 1)$ that makes it suitable for NTTs [31]. Hardware acceleration of primitives through platform-specific ISA extensions and cryptographic processors such as, single-cycle multiplication and vectorized NTTs on target platforms, such as ARM Cortex-M4 and Intel AVX2 would improve all Lattice-based schemes.

Table 8. Sample modern schemes [29]

Modern schemes	Keys/Moduli (bits)
AES	128, 192, 256-bit keys
RSA	2048, 4096, 7680, 15,360-bit moduli
DH	2048-bit modulus
ECC	160–233, 224–255, 256–383, 384–511, 512+ moduli

The reduction-free property of RFTs is an extension from binomials. For simplicity, if we consider a small field of the form $\mathbb{Z}_2[t]/(x^8+1)$, we can easily see that with this binomial, elements can be split into even half-size words of d-bits or 8/2 bits. $F(t)$ can be split evenly by allowing the upper register to hold the d most significant bits and the lower register can be truncated to d-bits since it is 1 and reducing with $\pmod{00001}$ is equivalent to reducing with $\pmod{0001}$ or just 1. We can easily observe that the Blakely quotient will just be the $d-1$ most significant bits of C_2 and Montgomery is just the least significant d-bits of C_0. Modular multiplication in finite fields and rings can apply the reduction-free property when the defining polynomial is a binomial or trinomial that allows it.

When the coefficients of the polynomials are elements in a ring of field, such as $\mathbb{Z}_q$, such as a ring of the form $\mathbb{Z}_q[t]/(x^n+1)$, the reduction-free property can be explored with respect to modulo q. For example, for $q = 8192_{10} = 10000000000000_2$ and intermediate multiplications and additions of elements in $\{0, 1, .., q-1\}$ may be done in the binary base and we may exploit reduction-free properties and split q into $q_1 = 1000000$ and $q_0 = 0000000$.

Table 9. Sample PQC keys [31]

PQC schemes	Key size (Bytes)	Security level
Kyber768	$sk = 2400(32^{\dagger}), pk = 1182$	3
FireSaber-KEM	$sk = 3040(1760^{\ddagger}), pk = 1312$	5
NTRUhrss701	$sk = 1452, pk = 1138$	$1, 3^{*}$
Dilithium5	$sk = 2592, pk = 4595$	5
Falcon-1024	$pk = 1793, \sigma = 1280$	5

† indicates option for only 32 bytes of randomness with trade-offs.

‡ indicates option to use compression to reduce the key size to 384 bytes. ∗ 1 for non-local models, 3 for local.

Table 10. Sample PQC fields and ring parameters [31]

Scheme	$\mathbb{Z}_q[t]/F(t)$
Saber	$\mathbb{Z}_{3329}[t]/(t^{256}+1)$
Kyber	$\mathbb{Z}_{3329}[t]/(t^{256}+1)$
NTRUhrss701	$\mathbb{Z}_{8192}[t]/(t^{701}+1)$
Dilithium	$\mathbb{Z}_{8380417}[t]/(t^{256}+1)$
Falcon-1024	$\mathbb{Z}_{12289}[t]/(t^{1024}+1)$

Improvements of Karatsuba and Toom-N that explore reduction-free $(\bmod\ q, F(t))$ is applicable for cases where NTT cannot be applied. If NTT can be adapted, there is no reduction $(\bmod\ F(t))$ and we are only concerned with reduction $(\bmod\ q)$ in which case, we can obtain reduction-free NTT if q can be expressed accordingly. The polynomial arithmetic and techniques applicable depend on the implementation type, software or hardware. In Saber, the reference software implementation notes that 50–70% of the time is spent on generating pseudorandomness [31]. Recent work also shows that optimization of polynomial multiplication in Lattice-based schemes, controls computation time to a large-scale [18]. Our work is specifically applicable to hardware implementations which are optimized through principled design. RF-FIKO can be designed in different bases, radices, incorporated with other techniques (pipelining, parallelism, refined KOA) and algorithms (e.g. Toom-N), and transformed into other domains (e.g. NTT) to obtain a significant speed up in modern, PQC, and FHE schemes.

Highly optimized software and specialized hardware implementations have paramount applications on the Internet and computing systems in general. These include embedded firmware (e.g. TPMs), cryptographic libraries (e.g. OpenSSL) to secure the cloud and VPNs through transport layer security (TLS) and IPSec implemented in the OS code on hosts and gateway routers, and devices in general such as, cryptographic cores and modules (e.g., secure enclaves on SoCs). Moreover, blockchain technology which is highly dependent on PKC, namely digital signatures, is faced with protecting against quantum attacks. To remain secure and practical in the quantum regime, blockchains must implement PQC schemes efficiently. General adaption of PQC must also be applied in a timely manner [26]. Being able to operate on encrypted data efficiently is also highly desired for FHE applications such as, zk-SNARKs.

11 Conclusions

Efficient implementations of modern PKC and lattice-based schemes are sought in both software and hardware. In this paper, we introduced two new algorithms (FIKO and RF-FIKO) which are based on fully interleaved bit-parallel Karatsuba-Ofman multipliers without the bipartite reduction circuits. Their

FPGA and ASIC implementations were faster than FIKO and BMM and showed promising results for PQC and FHE implementations. Moreover, further analysis of a complete system that applies cryptographic primitives must account for software and hardware attacks, such as side-channels. In this case, because RF-FIKO is reduction-free, it eliminates timing leakage via modular reductions without the need to recourse to alternative algorithms. This conforms to constant-time implementation requirements.

Further optimization of RF-FIKO through incorporation with other techniques or transforming NTT into reduction-free NTT, merits further research and consideration.

Acknowledgements. As the first author, I would like to thank all the beautiful people in my life.

References

1. Babbage, C.: Passages from the Life of a Philosopher. Longman, London (1864)
2. Bajard, J.C.: Useful Arithmetic for Cryptography, July 2013
3. Blakley, G.R.: A computer algorithm for the product AB modulo M. IEEE Trans. Comput. **32**(5), 497–500 (1983)
4. Brakerski, Z., Garg, S., Tsabary, R.: FHE-based bootstrapping of designated-prover NIZK. IACR ePrint Archive, Paper 2020/1168 (2020)
5. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 505–524. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_29
6. Brent, R.P., Zimmermann, P.: The great trinomial hunt. Not. Am. Math. Soc. **58**(2), 233–239 (2011)
7. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 409–437. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_15
8. Elia, M., Leone, M., Visentin, C.: Low complexity bit-parallel multipliers for $GF(2^m)$ with generator polynomial $x^m + x^k + 1$. Electron. Lett. **35**, 551–552 (1999)
9. Fan, H., Dai, Y.: Fast bit-parallel $GF(2^n)$ multiplier for all trinomials. IEEE Trans. Circuits Syst. I Regul. Pap. **54**(4), 485–490 (2005)
10. Fan, H., Hasan, M.A.: Fast bit parallel-shifted polynomial basis multipliers in $GF(2^n)$. IEEE Trans. Circuits Syst. I Regul. Pap. **53**(12), 2606–2615 (2006)
11. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR ePrint Archive (2012)
12. Fürer, M.: Faster Integer Multiplication, April 2007
13. Gallardo, L.H., Rahavandrainy, O.: On (unitary) perfect polynomials over $\mathbb{F}_2$ with only Mersenne primes as odd divisors (2019)
14. Halbutoğulları, A., Koç, Ç.K.: Mastrovito multiplier for general irreducible polynomials. IEEE Trans. Comput. **49**(5), 503–518 (2000)
15. Hariri, A., Reyhani-Masoleh, A.: Bit-serial and bit-parallel montgomery multiplication and squaring over $GF(2^m)$. IEEE Trans. Comput. **58**(10), 1332–1345 (2009)

16. Kaihara, M.E., Takagi, N.: Bipartite modular multiplication. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 201–210. Springer, Heidelberg (2005). https://doi.org/10.1007/11545262_15
17. Kaihara, M.E., Takagi, N.: Bipartite modular multiplication method. IEEE Trans. Comput. **57**(2), 157–164 (2008)
18. Kannwischer, M.J.: Polynomial multiplication for post-quantum cryptography. Ph.D. thesis, Radboud Universiteit Nijmegen (2022)
19. Karatsuba, A., Ofman, Y.: Multiplication of multidigit numbers by automata. Soviet Physics-Doklady **7**, 595–596 (1963)
20. Koç, Ç.K.: High-speed RSA implementation. Technical report. TR 201, RSA Laboratories, 73 pages, November 1994
21. Li, Y., Chen, Y.: New bit-parallel Montgomery multiplier for trinomials using squaring operation. IACR ePrint Archive, Paper 2014/268 (2014)
22. Li, Y., Ma, X., Zhang, Y., Qi, C.: Mastrovito form of non-recursive Karatsuba multiplier for all trinomials. IEEE Trans. Comput. **66**(9), 1573–1584 (2017)
23. Li, Y., Zhang, Y.: An Efficient CRT-based Bit-parallel Multiplier for Special Pentanomials. IACR ePrint Archive (2020)
24. Mastrovito, E.: VLSI architectures for computation in Galois fields, Ph.D. dissertation. Ph.D. thesis, Linkoping University (1991)
25. Montgomery, P.L.: Modular multiplication without trial division. Math. Comput. **44**(170), 519–521 (1985)
26. Moody, D.: The beginning of the end: the first NIST PQC standards. Technical report, NIST (2022)
27. Navas, J.A., Dutertre, B., Mason, I.A.: Verification of an optimized NTT algorithm. In: Christakis, M., Polikarpova, N., Duggirala, P.S., Schrammel, P. (eds.) NSV/VSTTE -2020. LNCS, vol. 12549, pp. 144–160. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63618-0_9
28. Negre, C.: Efficient parallel multiplier in shifted polynomial basis. J. Syst. Architect. **53**, 109–116 (2007)
29. NIST: Digital Signature Standard (DSS). Technical report. Federal Information Processing Standards Publications (FIPS PUBS) 186-4, U.S. Department of Commerce, July 2013
30. NIST: SHA-3 standard: permutation-based hash and extendable-output functions. Technical report. Federal Information Processing Standards Publications (FIPS PUBS) 202, U.S. Department of Commerce, August 2015
31. NIST: Post-Quantum Cryptography, Round 3 Submissions (2020)
32. Roy, S.S., Basso, A.: High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(4), 443–466 (2020)
33. Saldamli, G., Baek, Y., Ertaul, L.: Partially interleaved modular Karatsuba-Ofman multiplication. IJCSNS **15**(5), 503–518 (2015)
34. Schönhage, A., Strassen, V.: Schnelle multiplikation großer zahlen. Computing **7**(3), 281–292 (1971)
35. Sun, J., Li, Y., Zhang, Y., Guo, X.: Efficient nonrecursive bit-parallel Karatsuba multiplier for a special class of trinomials. VLSI Des. (2018)
36. Sunar, B., Koç, C.: Mastrovito multiplier for all trinomials. IEEE Trans. Comput. **48**(5), 522–527 (1999)
37. Wu, H.: Bit-parallel finite field multiplier and squarer using polynomial basis. IEEE Trans. Comput. **51**(7), 750–758 (2002)
38. Zierler, N.: Primitive trinomials whose degree is a Mersenne exponent. Inf. Control **15**(1), 67–69 (1969)