# Dual-Field Arithmetic Unit for $GF(p)$ and $GF(2^m)$*

Johannes Wolkerstorfer

Institute for Applied Information Processing and Communications,
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
`Johannes.Wolkerstorfer@iaik.at`, `http://www.iaik.at`

**Abstract.** In this article we present a hardware solution for finite field arithmetic with application in asymmetric cryptography. It supports calculation in $GF(p)$ as well as in $GF(2^m)$. Addition and multiplication with interleaved modular reduction are the main functionality of the unit. Additional functions—like shift operations and integer incrementation—allow the calculation of the multiplicative inverse and covering all operations required to implement Elliptic Curve Cryptography. Redundant number representation and efficient modular reduction make it ready for future cryptographic bitlengths and allow operation at high clock frequency on moderate hardware resources.

**Keywords:** Finite field arithmetic, multiplication, modular reduction, inversion, redundant number representation, hardware implementation.

## 1 Introduction

Finite field arithmetic is the backbone for nearly all public-key algorithms currently used. Widespread techniques like RSA encryption and Diffie-Hellman key agreement operate on finite fields with modular integer arithmetic. These algorithms have bitlengths up to 2048-bit to ensure information security for the next decade [3]. The calculation of these algorithms relies on exponentiation, which is computational intensive and demands dedicated hardware solutions when throughput is of concern. More recently, Elliptic Curve Cryptography (ECC) made the application of another type of finite fields popular: binary extension fields where elements can be represented as polynomials instead of integers. Binary fields $GF(2^m)$ are considered advantageous for hardware solutions because addition and modular reduction of polynomials are somewhat easier than those of integers.

ECC has the advantage of shorter bitlengths while offering the same level of security (163-bit up to 571-bit). That makes ECC attractive for application in constrained systems like smartcards where chip area is limited and the computational power of microprocessors is sparse. Applications of ECC are digital

---

* This work origins from the European Commission funded project *USB_CRYPT* established under contract IST-2000-25169 in the Information Society Technologies (IST) Program.

B.S. Kaliski Jr. et al. (Eds.): CHES 2002, LNCS 2523, pp. 500–514, 2003.

signature schemes, encryption schemes, and key agreement schemes [5,6,7]. The Elliptic Curve Digital Signature Standard [8,4] defines prime fields $GF(p)$ and binary fields $GF(2^m)$ as underlying fields for elliptic curves. For a full support of the standard, both type of fields have to be supported. This gives reason to search for hardware architectures that operate in both fields. Such a dual-field arithmetic unit can be realized and most of the hardware resources required for calculations in the prime field $GF(p)$ can be reused for operation in $GF(2^m)$. The cost of such an unified arithmetic unit for $GF(p)$ and $Gf(2^m)$ is only slightly higher than for a mere $GF(p)$-multiplier [15,16].

Former arithmetic units have focused on an efficient implementation of the multiplication and have neglected other operations. This is justified by the observation that the core operation of algorithms like RSA and Diffie-Hellman is modular exponentiation, which is calculated by repeated multiplications. The situation for ECC is slightly different. Although, the performance of ECC is also determined by multiplication, ECC requires besides multiplication and squaring also inversion, addition, and subtraction.

We will present a dual-field arithmetic unit that is capable to calculate all these operations in both types of fields: $GF(p)$ and $GF(2^m)$. The architecture takes low-power design considerations into account and assures a short critical path to enable operation at high clock frequencies. The intended applications of the arithmetic unit are systems with limited silicon area where both types of arithmetic are required and performance is not of utmost importance. The main motivation for the design was to develop an unit that is capable to perform all calculations of the Elliptic Curve Digital Signature Algorithm (ECDSA) and key-agreement protocols defined by the American National Standards Institute (ANSI) [4,5]. Further relevant ECC standards are published by the Institute of Electrical and Electronic Engineers (IEEE) [6], the International Standards Organization (ISO) [7], and the National Institute of Standards and Technology (NIST) [8].

The proposed architecture of the dual-field arithmetic unit focuses on an efficient implementation of operations in the finite fields $GF(p)$ and $GF(2^m)$. Operands are processed at full precision and most operations are executed within a single clock cycle. Multiplication is a multi-cycle operation with bitserial scheduling of the multiplier. Modular reduction is interleaved and uses quotient prediction for operation in $GF(p)$. Intermediate results of $GF(p)$-operations have a redundant number representation which permits to scale the architecture's precision without affecting the maximum clock frequency. The architecture even allows to calculate the Extended Euclidean Algorithm for inverting field elements. The architecture is highly regular and has only a small number of leaf cells—which is a desired property for a full-custom implementation.

The remainder of this article presents related work in §2. In §3 the mathematical background of operations in prime fields and in binary fields is covered. §4 presents the proposed architecture and discusses design considerations. Finally, we present results in §5 and draw conclusions in §6.

## 2 Related Work

E. Savaş et al. published in 2000 a unified multiplier for $GF(p)$ and $GF(2^m)$ which uses Montgomery multiplication for both fields [15]. Multiplication is done bitserial and the multiplicand is processed in blocks. Arbitrary precision multiplication is possible and precision is only constrained by memory. Their architecture is based on a pipeline of block-sized processing elements. The pipeline can have different configurations to trade area for speed. This approach has the smartness to process arbitrary precision numbers, which comes at the cost of a more complicated architecture that seems to be challenging for a full-custom implementation. Another restraint is the need of the Montgomery algorithm for precomputed constants and the need of transformations.

J. Großschädl's unified multiplier is bitserial too but processes the multiplicand in full precision [16]. Modular reduction is done by an interleaved quotient prediction and a conditional modulus subtraction, which does not require any pre-computations or transformations. Multiplication in $GF(2^m)$ takes $m$ cycles, whereas multiplication in $GF(p)$ takes between $\log_2 p$ and $2\log_2 p$ cycles due to a conditional extra modular reduction cycle. Intermediate results of $GF(p)$-operations are stored in redundant representation because partial-product accumulation is done with carry-save adders. A carry-propagate adder with lower wordsize converts redundant results iteratively into their binary representation. The proposed architecture is simple, requires little hardware resources and has a regular structure that is convenient for a full-custom implementation. The low $GF(p)$-performance is a disadvantage. It is caused by the reduction algorithm and the redundant-to-binary conversion.

J. Goodman et al. presented in [17] a VLSI implementation of a dual-field arithmetic unit. Their so-called Domain-Specific Reconfigurable Cryptographic Processor (DSRCP) is not a mere multiplier for $GF(p)$ and $GF(2^m)$. It can calculate all operations required for elliptic curve cryptography including inversion and comparisons. These operations are executed on an extensive datapath which is controlled by a microcoded control unit. Main components of the datapath are a carry-propagate adder for operation in $GF(p)$ that takes three cycles for an addition and a reconfigurable datapath for operation in $GF(2^m)$. An additional comparator allows comparisons of integers or polynomials. The Montgomery algorithm is used for multiplication in $GF(p)$. Multiplication in $GF(2^m)$ obeys an iterative MSB-first scheme with interleaved modular reduction.

## 3 Mathematical Background

This section describes the representation of prime field elements and binary field elements and presents operations on these elements.

Prime field elements $A \in GF(p)$ are integers in the set $\{0, 1, \ldots p-1\}$ where $p$ is prime. Binary field elements $A(x) \in GF(2^m)$ are polynomials of degree less than $m$ when a polynomial basis is used to represent the field elements. These polynomials have coefficients in the set $\{0, 1\}$. Both types of field elements can be represented with bitstrings as shown in (1) and (2). The binary representation of a prime field element needs $n = \lceil \log_2 p \rceil$ bits for storage. Elements of $GF(2^m)$ require $m$ bits to store all coefficients of the polynomial. The memory requirement to store both types of elements is $max(n, m)$ bits.

$$A \in GF(p): \qquad A = \sum_{i=0}^{n-1} a_i 2^i \quad \text{with} \quad n = \lceil \log_2 p \rceil, \quad a_i \in \{0,1\} \tag{1}$$

$$A(x) \in GF(2^m): \quad A(x) = \sum_{i=0}^{m-1} a_i x^i, \quad a_i \in \{0,1\} \tag{2}$$

Although elements of both fields are stored uniformly, their field operations differ. Addition of prime field elements is an integer addition with modular reduction, whereas addition of polynomials is done coefficient-wise without the need of modular reduction. Multiplication requires modular reduction in both cases because the result of an integer multiplication as well as the result of a polynomial multiplication could have double the bitlength of their operands. Surprisingly, an almost identical algorithm can calculate multiplication in $GF(p)$ and in $GF(2^m)$ which facilitates an unified hardware approach.

Not all cryptographic algorithms require the inversion of field elements. For instance, the RSA algorithm and the Diffie-Hellman key-exchange are based on exponentiation and require only multiplications and square operations. On the other hand, elliptic-curve cryptography and the digital signature algorithm require inversion too. The inverse of field elements can be calculated by exponentiation using the Fermat theorem or by applying the Extended Euclidean Algorithm (EEA) [1]. The latter has better running time but is more difficult to implement in hardware because it requires inconvenient operations like magnitude-comparisons of integers or bitlength-comparisons of polynomials.

### 3.1 Addition and Modular Reduction in $GF(p)$

Addition of two integers $A, B \in GF(p)$ is done by calculating the sum $A+B$ with carry propagation. In case, the sum $A + B$ exceeds $p - 1$, a modular reduction is necessary to obtain the result of $A + B \bmod p$ in the range $[0, p - 1]$.

In general, the result of a modular reduction of an integer $I \bmod p$ is the remainder of the integer division $\frac{I}{p}$. The remainder can be calculated using (3).

$$I \bmod p = I - q \cdot p \quad \text{with} \quad q = \left\lfloor \frac{I}{p} \right\rfloor \tag{3}$$

Equation (3) is not very practical because it determines the quotient $q$ by division of large integers. Division can be avoided when the reduction result may exceed the desired interval $[0, p - 1]$. In this case, it possible to estimate a quotient $\hat{q}$ by comparing $I$ with a number $N$ in the magnitude of the modulus $p$. A good choice is $N = 2^{\lfloor \log_2 p \rfloor}$ where only the most significant bit of $p$ is set. In case $p$ is a generalized Mersenne prime, this estimation is very close.

### 3.2 Multiplication and Squaring in $GF(p)$

Multiplication $A \cdot B$ is a heavyweight operation compared to addition. The product of two large integers cannot be calculated in a single step. The product is calculated by accumulating partial products $A \cdot b_i$ iteratively—this algorithm is known as the *double-and-add* algorithm. Bitserial multiplication obeys the double-and-add algorithm. It scans all bits $b_i$ of the multiplier $B$ iteratively. If the actual multiplier bit $b_i = 1$, the multiplicand $A$ is accumulated to the intermediate result as done in (4). Two different schemes are possible to scan the multiplier bits: the LSB-first scheme and the MSB-first scheme. The LSB-first scheme starts to scan multiplier bit $b_0$ and ends with $b_{n-1}$. The MSB-first scheme operates in the opposite direction.

$$C = A \cdot B = A \cdot \left( \sum_{i=0}^{n-1} b_i 2^i \right) = \sum_{i=0}^{n-1} (A \cdot b_i) 2^i, \quad n = \lceil \log_2 B \rceil \tag{4}$$

Bitserial multiplication can easily be extended to modular multiplication mod $p$. Extending Equation (4) by an interleaved modular reduction step will reduce the intermediate result in each iteration and yield Algorithm 1. As mentioned above, exact modular reduction would require the calculation of the quotient $q = \lfloor \frac{C}{p} \rfloor$ which involves division. Algorithm 1 avoids division by estimating the quotient $\hat{q}$. The estimation simplifies the algorithm substantially but the result $C$ may exceed the desired range $[0, p-1]$. To obtain a fully reduced result, the modulus $p$ has to be added up to two times.

**Algorithm 1** Multiplication in $GF(p)$ with interleaved modular reduction

```
Input: $A, B \in [0, p-1], \quad 2^{n-1} \le p < 2^n$
Output: $C = A \cdot B \bmod p$
$C \Leftarrow 0$
for $i = n-1$ to $0$ do
    $C \Leftarrow 2 \cdot C + A \cdot b_i$
    $\hat{q} \Leftarrow \text{Q\_ESTIM}(C)$
    $C \Leftarrow C - \hat{q} \cdot p$
end for
while $C < 0$ do
    $C \Leftarrow c + p$
end while
return $C$
```

Squaring is closely related to multiplication because $A^2 \bmod p = A \cdot A \bmod p$. Systems that operate with wordsizes smaller than the bitlength of the operands—like microprocessors—usually have an extra square function to exploit common sub-expressions on wordsize-level. This could save nearly 50 percent of the required wordsize multiplications. For systems that operate on full-length operands there is not such a short cut and squaring is done most efficiently by multiplication.

### 3.3 Inversion in $GF(p)$

Inversion calculates the multiplicative inverse $A^{-1}$ of an element $A$. The inverse has the property that $A \cdot A^{-1} \bmod p = 1$. There are two different methods to calculate the inverse. One is based on the theorem of Fermat that states $A^{p-1} \bmod p = 1$ and implies that $A^{p-2} \bmod p = A^{-1} \bmod p$ [1]. Using this theorem, the inverse is calculated by exponentiation which requires about $1.5 \log_2 p$ multiplications—an expensive operation. The other method to calculate the inverse is the Extended Euclidean Algorithm (EEA). It solves the equation $A \cdot X + p \cdot Y = D$ for $X$, $Y$ and $D = gcd(A, p)$. The EEA's procedure to calculate the inverse is given in Algorithm 2. It is a slightly modified version of the algorithm given in [13].

**Algorithm 2** Inversion in $GF(p)$: Extended Euclidean Algorithm (EEA)

```
Input: $A \in [0, p-1]$,  $p$ prime
Output: $A^{-1} \bmod p$
$Y \Leftarrow A$, $D \Leftarrow p$, $B \Leftarrow 1$, $X \Leftarrow 0$
while $Y \neq 0$ do
    while $y_0 = 0$ do
        $Y \Leftarrow Y/2$, $B \Leftarrow (B + b_0 p)/2$
    end while
    while $d_0 = 0$ do
        $D \Leftarrow D/2$, $X \Leftarrow (X + x_0 p)/2$
    end while
    if $Y \geq D$ then
        $Y \Leftarrow Y - D$, $B \Leftarrow (B - X) \bmod p$
    else
        $D \Leftarrow D - Y$, $X \Leftarrow (X - B) \bmod p$
    end if
end while
return $X$
```

### 3.4 Addition in $GF(2^m)$

Addition in $GF(2^m)$ is done coefficient-wise as shown in Equation (5).

$$A(x) + B(x) = \sum_{i=0}^{m-1} a_i x^i + \sum_{i=0}^{m-1} b_i x^i = \sum_{i=0}^{m-1} (a_i + b_i) x^i = \sum_{i=0}^{m-1} (a_i \text{ xor } b_i) x^i \quad (5)$$

Coefficients of polynomials are elements of $GF(2) = \mathbb{Z}_2$ and therefore, addition of coefficients is done modulo 2 which corresponds to the Boolean XOR-function. Multiplication of coefficients matches the Boolean AND-function. Subtraction in $GF(2^m)$ is identical with addition because the additive inverse of an element is its identity: $A(x) + A(x) = 0$.

### 3.5 Multiplication in $GF(2^m)$

Multiplication in $GF(2^m)$ calculates the product of two polynomials and applies modular reduction. Although, polynomial multiplication is completely different from integer multiplication, the resulting algorithm for multiplication in $GF(2^m)$ is very similar to Algorithm 1 for multiplication in $GF(p)$. This property allows building an efficient unified multiplier that supports both fields.

Multiplication in $GF(2^m)$ can also use the double-and-add approach used for multiplication in $GF(p)$ which accumulates partial products as shown in (6). Partial products have to be aligned to the intermediate result which is indicated in (6) by a multiplication by $x^i$. Multiplication by $x^i$ can easily be computed by shifting the binary representation of the partial product $i$ positions to the left.

$$A(x) \cdot B(x) = A(x) \cdot \left( \sum_{i=0}^{m-1} b_i x^i \right) = \sum_{i=0}^{m-1} \big(A(x) b_i\big) \cdot x^i \tag{6}$$

A modular reduction step after the polynomial multiplication assures that the result is an element of $GF(2^m)$ with an degree less than $m$. Alternatively, the reduction can be done during the accumulation of partial products as shown in Algorithm 3.

**Algorithm 3** Multiplication in $GF(2^m)$ with interleaved modular reduction

```
Input: $A(x), B(x) \in GF(2^m)$,    irreducible polynomial $P(x)$ of degree $m$
Output: $C(x) = A(x) \cdot B(x) \bmod P(x)$
$C(x) \Leftarrow 0$
for $i = m - 1$ to 0 do
    $C(x) \Leftarrow C(x) \cdot x + A(x) b_i$
    $C(x) \Leftarrow C(x) + c_m P(x)$
end for
return $C(x)$
```

The modular reduction $A(x)$ mod $P(x)$ in $GF(2^m)$ is done modulo an irreducible polynomial $P(x)$. This operation calculates in principle the remainder of the polynomial division $A(x)/P(x)$. Efficient implementations avoid division by iterated subtraction of the product $P(x) \cdot x^i$. During bitserial multiplication with interleaved modular reduction the intermediate result $C(x)$ can not have higher degree than $m$. Thus, modular reduction is only necessary when $C(x)$ has degree $m$. This condition is indicated by $c_m = 1$.

### 3.6 Squaring in $GF(2^m)$

In contrast to $GF(p)$, squaring in $GF(2^m)$ has lower complexity than multiplication. One reason for this is, that $A(x)^2$ mod $P(x)$ is a linear operation in $GF(2^m)$. Based on this observation one could square efficiently by calculating $A(x)^2 = \sum_{i=0}^{m-1} a_i x^{2i}$. A subsequent modular reduction will yield the desired result. This method is used in software implementations like [14]. Hardware implementations can exploit this feature when the extension degree $m$ and the irreducible polynomial $P(x)$ are fixed.

### 3.7 Inversion in $GF(2^m)$

The inverse of an element $A(x) \in GF(2^m)$ can be calculated by the exponentiation $A(x)^{2^m-2} \bmod P(x)$ or by the Extended Euclidean Algorithm for polynomials (EEA). An improvement of the EEA algorithm is the Modified Almost Inverse Algorithm presented in [14]. Algorithm 4 is a slightly modified version of this. Almost all calculations of the algorithm operate on polynomials but comparisons of polynomials are replaced by integer subtractions and sign testing to avoid additional circuitry in a hardware implementation. In Algorithm 4, polynomials are multiplied by $x^{-1}$ which is a simple shift-right operation.

**Algorithm 4** Inversion in $GF(2^m)$: Modified Almost Inverse Algorithm

```
Input: $0 \neq A(x) \in GF(2^m)$,    irreducible polynomial $P(x)$ of degree $m$
Output: $A(x)^{-1} \bmod P(x)$
$Y(x) \Leftarrow A(x)$, $D(x) \Leftarrow P(x)$, $B(x) \Leftarrow 0$, $X(x) \Leftarrow 1$
loop
  while $y_0 = 0$ do
    $Y(x) \Leftarrow Y(x) \cdot x^{-1}$, $X(x) \Leftarrow (X(x) + x_0 P(x)) \cdot x^{-1}$
  end while
  if not $(1 - Y < 0)$ then {comparison $Y(x) = 1$ by integer subtraction}
    return $X(x)$
  end if
  if $Y - D < 0$ then {comparison $\deg Y(x) < \deg D(x)$ by integer subtraction}
    $Y(x) \Leftarrow Y(x) + D(x)$, $X(x) \Leftarrow X(x) + B(x)$
    $D(x) \Leftarrow D(x) + Y(x)$, $B(x) \Leftarrow B(x) + X(x)$
  else
    $Y(x) \Leftarrow Y(x) + D(x)$, $X(x) \Leftarrow X(x) + B(x)$
  end if
end loop
```

## 4 Architecture

The Elliptic Curve Digital Signature Algorithm (ECDSA) [8] is the target application of the dual-field arithmetic unit. The desired functionality of the unit can be clearly derived from this application. Off course, multiplications in $GF(p)$ and $GF(2^m)$ are the most important functions, but addition and subtraction are required too. In order to calculate the inverse, it is necessary to increment integers, to check whether integer values are negative, and to shift values one position to the left or to the right. Operations like holding the result or clearing the result are obviously useful. The required operations can be summarized in the following categories: integer arithmetic, modular integer arithmetic, modular polynomial arithmetic, and comparisons.

Functionality is one important aspect of a hardware module. Other quality aspects of a circuit are its size, its speed, and its energy consumption. These factors cannot be optimized independently because they influence each other. Energy efficiency was a prime objective in the design of the arithmetic unit, so the unit was not optimized for lowest gate-count or for a high degree of parallelism. High throughput is achieved by keeping the critical path short to enable operation at high clock frequencies. The architecture is scalable for the maximum bitlength of integers respectively polynomials. Adjusting the bitlength to the requirements of the application (e.g. 192-bit) keeps the gate-count low. It is possible to process smaller integers/polynomials by pre-shifting them in order to align them to the physical dimension of the unit.

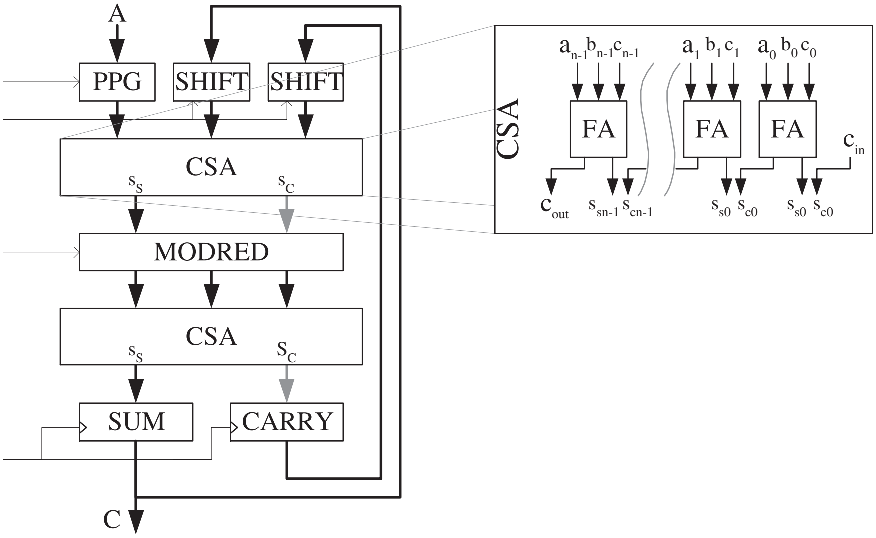

**Fig. 1.** Architecture of the dual-field arithmetic unit

Figure 1 shows the architecture of the dual-field arithmetic unit. The unit has three major components: a partial-product generator (PPG), a modular reduction unit (MODRED), and a shift unit (SHIFT). During bitserial multiplication, which is done in the MSB-first scheme at full precision, the PPG masks the input $A$ with the actual multiplier bit $b_i$ to generate the partial product $A \cdot b_i$. The partial product is generated the same way for $GF(p)$ and for $GF(2^m)$. Note, that the circuit which serializes the multiplier $B$ is left out in the Figure 1. An adder (CSA) accumulates the partial product to the intermediate result stored in the registers *SUM* and *CARRY*. Prior to this addition the intermediate result is shifted by the *Shift* unit one position to the left to align the last accumulation result. The *MODRED* unit inserts a modular reduction step by subtracting $\hat{q} \cdot p$ in case of $GF(p)$-operation or $q \cdot P(x)$ in case of $GF(2^m)$-operation. A datapath cycle is finished when the new intermediate result is stored in the registers *SUM* and *CARRY.*

The arithmetic unit uses Carry-Save Adders (CSA) to eliminate carry-propagation delay during $GF(p)$-operation. Carry propagation in conventional adders would cause significant delay. CS-adders prevent this by a redundant representation of the output sum. The redundant sum consists of two integers $S_S$ and $S_C$, which are stored in the registers SUM and CARRY. The number of storage bits used for this is twice the amount to store the sum in a binary representation. The additional hardware resources are justifiable because they allow to add integers of arbitrary length in constant time. Carry-save adders are based on Formula (7) and are implemented with conventional full-adder cells. The delay of an adder of arbitrary width is equal to the delay of a single full-adder cell.

$$A + B + C = CSA(A, B, C) = S_S + 2S_C \tag{7}$$
$$\text{with} \quad s_{Si} = a_i \text{ xor } b_i \text{ xor } c_i \quad \text{and} \quad s_{Ci} = a_i b_i \text{ or } a_i c_i \text{ or } b_i c_i$$

Carries are only required for addition in $GF(p)$. Addition in $GF(2^m)$ does not use them because polynomials are added coefficient-wise with the Boolean XOR-function. The XOR-function is a sub-function of an CS-adder as Formula (7) reveals: Keeping input $C = 0$, makes the sum-component $S_S = A$ xor $B$. The carry-component will be $S_C = A$ and $B$ in this case. This property of carry-save adders is used to configure the arithmetic unit for $GF(2^m)$-operation. By forcing the carry-output of both CS-adders to zero, all carries are eliminated and in turn the CS-adders will have the desired functionality of an $n$-bit XOR-gate.

The input and the output of the dual-field arithmetic unit is restricted to binary $n$-bit values (input $A$, output $C$). Therefore, it is necessary to convert integers stored internally in a redundant representation into their binary representation before output. The conversion of a redundant number into a binary number requires addition with carry propagation. Usually, such a conversion is done with a carry-propagation adder implicating the performance problems mentioned above. Carry-save adders offer another possibility to do the conversion. The observation that the result of an repeated carry-save addition of an redundant number ($S = S_S + 2S_C$) and zero is the binary number $S$ after $\log_2 \log_2 S$ iterations on average, leads to Algorithm 5. Addition, subtraction and multiplication in $GF(p)$ require this conversion before output. Both CS-adders of the arithmetic unit can be used to execute this operation. Hence, the expected running time of an $n$-bit architecture is halved to $0.5 \log_2 n$ cycles. During conversion, the PPG and MODRED unit have to output 0 in order to keep the third CSA input 0. Conversion is finished when $S_C = 0$. An $n$-bit NOR-gate reports this condition.

The MODRED unit calculates the correction term for the interleaved modular reduction. It has different functionality for $GF(p)$ and $GF(2^m)$-operations. During $GF(2^m)$-operations, the functionality of MODRED is simple. Whenever the most-significant bit $s_{Sm}$ of the intermediate result $S_S$ is set, MODRED has to output the irreducible polynomial $P(x)$, otherwise 0. The subsequent CS-adder

**Algorithm 5** Redundant-to-binary conversion with carry-save adders

```
Input: redundant number $S = (S_S + 2S_C)$
Output: binary number $S$
while $S_C \neq 0$ do
    $(S_S, S_C) \Leftarrow CSA(0, S_S, 2S_C)$
end while
return $S_S$
```

will execute the reduction by adding the correction term: $S_S$ mod $s_{Sm}P(x)=$ $S_S$ mod $P(x)$. The reduction works for arbitrary irreducible polynomials and is not restricted to a special kind of polynomials like trinomials or pentanomials.

Modular reduction by a prime integer is more complicated. The quotient is estimated and causes a non-perfect reduction. The intermediate result can exceed the bitlength of the modulus $n = \lceil \log_2 p \rceil$. Therefore, the datapath is chosen to be $n + 2$ bits wide. The quotient estimation works as follows: first the magnitude of the intermediate result is estimated by adding the three highest bits of the redundant intermediate result with carry propagation $\hat{S} = (s_{Sn+1}, s_{Sn}, s_{Sn-1})+(s_{Cn+1}, s_{Cn}, s_{Cn-1})$. Then the quotient $\hat{q} \in \{-2, -1, 0, 1, 2\}$ is determined by table-lookup. The table entries are chosen such that the desired result of the reduction is in the range $[0, -(p-1)]$. As a consequence of this reduction algorithm the reduced intermediate result $S - \hat{q} \cdot p$ is usually negative and hence the datapath must be capable to handle signed numbers. This reduction scheme works for arbitrary moduli $p$ and is not restricted to generalized Mersenne primes. To ensure that the final result of an operation is fully reduced—or in other words is $\in [0, p-1]$—, the modulus $p$ may have to be added up to two times until the result is positive. Positive results are indicated by a cleared sign bit of $S_S$ and $S_C = 0$. The sign bit is also used for integer comparison, which are based on integer subtractions.

Signed numbers enable the calculation of subtractions. The arithmetic unit can calculate a subtraction $A - B$ by loading $A$ in one cycle and adding $-B$ in the next cycle. $-B$ is calculated by the PPG and a CSA: The PPG generates the one's-complement $\bar{B}$ of $B$ where $\bar{b_i}$ = not $b_i$ by inverting all bits. The CSA can turn $\bar{B}$ into the two's-complement $-B$ by incrementation. Incrementation is simply achieved by setting the lowest carry bit $s_{C0} = 1$ that is usually 0.

Shift operations are executed in the SHIFT unit. The SHIFT unit can output either 0, its input $I$, $I$ shift-left 1, or $I$ shift-right 1.

## 5 Results

The different functionalities of all datapath components can be combined into useful instructions of the whole arithmetic unit. The evolving instructions can be summarized in four categories: load operations, shift operations, addition, and multiplication. Load operations can either load the constants 0, 1 or the values $A$, $\bar{A}$, or $-A$. The shift operations can shift the stored value one position to the left or to the right. In the addition category are the operations XOR, integer addition, integer subtraction, incrementation, and integer addition/subtraction with modular reduction.

All the operations listed so far can be executed in one clock cycle. Multiplication takes exactly $n$ clock cycles and can calculate the product of two integers smaller than $2^{n/2}$, or the product of two integers modulo $p$, or the product of two polynomials mod $P(x)$.

When the datapath is configured for $GF(p)$-operation and the MODRED unit is inactive, the *hold* operation will convert redundant results into their binary representation. On average, 192-bit numbers will be converted in 3.8 cycles, 224-bit numbers in 3.9 cycles, and 256-bit numbers in 4.0 cycles. Control flags indicate whether the result is binary or it is negative. They are always evaluated and are reused for comparisons.

Inversion is a compound operation that has to be controlled from outside. Table 1 lists the estimated number of clock cycles for Algorithm 2 and Algorithm 4. These algorithms are about four times faster than calculating the inverse by exponentiation. The clock-cycle ratio of inversion to multiplication is about 70 for $GF(p)$ and 70 for $GF(2^m)$. This gives reason to avoid inversion when possible and advises to use projective coordinates when implementing elliptic curve cryptography. Table 1 also lists expected running times for a elliptic-curve scalar-multiplication using projective coordinates. All estimates are conservative and include the transformation to affine coordinates.

**Table 1.** Estimited cycles for inversion and ECC scalar multiplication

| $GF(p)$ | INV (Alg. 2) [cycles] | ECC proj. [cycles] |
|---|---:|---:|
| 192-bit | 14,000 | 720,000 |
| 224-bit | 16,500 | 900,000 |
| 256-bit | 19,400 | 1,150,000 |

| $GF(2^m)$ | INV (Alg 4) [cycles] | ECC proj. [cycles] |
|---|---:|---:|
| 163-bit | 11,000 | 490,000 |
| 233-bit | 16,200 | 905,000 |
| 283-bit | 20,700 | 1,405,000 |

The dual-field arithmetic unit requires only a few hardware resources. Four $n+2$-bit register are needed to store the modulus, the multiplier, and the result in redundant representation. The two SHIFT units can be implemented with $2n+4$ 4-to-1 multiplexers. The PPG unit is built of $n+2$ AND-gates and the same amount of XOR-gates. The MODRED unit has the same complexity as PPG plus $n+2$ 2-to-1 multiplexers. Two instances of CS-adders require $2n+4$ full-adder cells and $2n+4$ AND-gates which eliminate carries during $GF(2^m)$-operation. Table 2 lists the gate count for different bitlengths and gives a rough estimation of the area requirements of a standard-cell implementation on the 0.35 $\mu m$ CMOS process from Austriamicrosystems. The arithmetic unit is also well suited for a full-custom implementation. The regular part of the datapath is composed of only half a dozen of different gates. It should be of no difficulty to find a bitslice architecture for that part and to design leaf-cells for the gates in an appropriate logic style in order to obtain a sound full-custom layout. The datapath does not need sophisticated control because most instructions are executed in a single cycle. Only multiplication consumes more cycles. The control unit for bitserial multiplication can be used both for $GF(p)$ and for $GF(2^m)$-operation because the same double-and-add algorithm in the MSB-first scheme is used.

**Table 2.** Gate count and estimated area on a 0.35 $\mu m$ CMOS process

| Size | AND | XOR | MUX2 | MUX4 | FA | REG | area on 0,35 $\mu m$ |
|---|---|---|---|---|---|---|---|
| 163-bit | 660 | 330 | 165 | 330 | 330 | 660 | 0.57 $mm^2$ |
| 224-bit | 904 | 452 | 226 | 452 | 452 | 904 | 0.78 $mm^2$ |
| 283-bit | 1140 | 570 | 285 | 570 | 570 | 1140 | 0.99 $mm^2$ |

Most of the design decisions for the dual-field arithmetic unit were guided by low-power considerations. Especially, the design on the algorithmic level and the architectural level of a digital circuit offer promising options to save power [9]. Contrary to low-power measures on logic level, they are difficult to estimate. Therefore, a qualitative reasoning will be given. One design goal was to keep the critical path short. This implicates on one hand a high clock frequency and gives on the other hand the possibility to scale the supply voltage *VDD* of CMOS circuits. Lowering *VDD* is an effective technique to save power as it contributes quadratically to the dynamic power consumption [9]. A lowered supply voltage will also slow down the circuit: VDD can be decreased until the critical path delay reaches the clock period. Short critical paths have another advantage for low-power circuit design: The probability of undesired signal transitions (glitches) is lowered. Glitches will occur more frequently when the combinational logic-depth is high. To ensure a short critical path of the arithmetic unit, CS-adders were chosen. The critical path spans the partial product generator PPG, the modular reduction unit MODRED and two CS-adders.

A low-power driven design decision on the architectural level is the modular reduction unit MODRED. From the functional point of view, it would be possible to omit the MODRED unit and to cover its functionality by an enlarged partial product generator PPG. Thereby, a former single-cycle operation with an interleaved modular reduction would require two clock cycles: One cycle for the operation itself and one for the modular reduction step. This would certainly increase the energy-delay product. Furthermore, such an architecture would have negative impact on the signal activity of the input $A$ of the arithmetic unit: During multiplication, this bus would always change between the multiplicand $A$ and the modulus $p$ or $P(x)$. The insertion of an extra MODRED trades increased area-demands for lower power-consumption and helps to avoid wasteful signal activity.

## 6 Conclusion

In this article we presented a dual-field arithmetic unit that offers all instructions to implement the elliptic curve digital signature standard over prime fields $GF(p)$ and binary extension fields $GF(2^m)$. Therefore, the unit can calculate shift-operations, increments, and comparisons besides addition and multiplication. These operations enable to calculate the inverse with the extended Euclidean algorithm.

A design objective for the unit was energy efficiency, which yielded a low-power architecture that can be realized on moderate silicon area. The unit requires only little more hardware resources than a mere $GF(p)$-multiplier. The $GF(2^m)$-functionality and some other useful operations come at almost no additional cost. The use of carry-save adders guarantees a short critical path that allows operation at high clock frequencies—independent of the chosen datapath precision. The simplicity of the architecture with its inherent regularity and its limited number of leaf cells makes it well suited for a full-custom implementation.

## References

1. A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, New York, 1997.
2. R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*, Cambridge University Press, Cambridge, 1986.
3. A. K. Lenstra and E. R. Verheul, *Selecting Cryptographic Key Sizes*, Journal of Cryptology, vol. 14, pp. 255–293, 2001.
4. ANSI X9.62, *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, ANSI standard, 1999.
5. ANSI X9.63, *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*, ANSI standard, 2001.
6. IEEE P1363, *Standard Specifications for Public-Key Cryptography*, IEEE standard, 2000.
7. ISO / IEC 15946, *Information Technology - Security Techniques - Cryptographic Techniques Based on Elliptic Curves*, Committee Draft (CD), 1999.
8. National Institute of Standards and Technology, *Digital Signature Standard*, FIPS Publication 186–2, Feb. 2000.
9. J. M. Rabaey, M. Pedram (ed.), *Low Power Design Methodologies*, Kluwer Academic Publishers, 1996.
10. E. D. Mastrovito, *VLSI Architectures for Computations in Galois Fields*, PhD thesis, Linköping University, Linköping, Sweden, 1991.
11. C. Paar, *Efficient VLSI Architectures for Bit Parallel Computation in Galois Fields*, PhD thesis, Universität Essen, 1994.
12. Ç. Koç, C.Y. Hung, *A Fast Algorithm for Modular Reduction*, IEE Proceedings: Computers and Digital Techniques 145(4), pp. 265–271, July 1998.
13. M. Brown, D. Hankerson, A. Menezes, *Software Implementation of the NIST Elliptic Curves over Prime Fields*, Proceedings of CT-RSA 2001, LNCS 2020, pp. 250–265, Springer Verlag, 2001.

14. D. Hankerson, J. L. Hernandez, A. Menezes, *Software Implementation of Elliptic Curve Cryptography over Binary Fields*, Cryptographic Hardware and Embedded Systems - CHES 2000, LNCS 1965, pp. 1–24, Springer Verlag, Berlin, 2000.
15. E. Savaş, A. Tenca, Ç. Koç, *A Scalable and Unified Multiplier Architecture for Finite Fields* $GF(p)$ *and* $GF(2^m)$, Cryptographic Hardware and Embedded Systems - CHES 2000, LNCS 1965, pp. 281–296, Springer Verlag, Berlin, 2000.
16. J. Großschädl, *A Bitserial Unified Multiplier Architecture for Finite Fields* $GF(p)$ *and* $GF(2^m)$, Cryptographic Hardware and Embedded Systems - CHES 2001, LNCS 2162, pp. 206–223, Springer Verlag, 2001.
17. J. Goodman, A. P. Chandrakasan, *An Energy-efficient Reconfigurable Public-Key Cryptography Processor*, IEEE Journal of Solid-State Circuits, pp. 1808–1820, November 2001.