Design of a fully balanced ASIC coprocessor implementing complete addition formulas on Weierstrass elliptic curves

Niels Pirotte*, Jo Vliegen*, Lejla Batina† and Nele Mentens*
*ES&S and imec-COSIC/ESAT, KU Leuven
Wetenschapspark 21, 3590 Diepenbeek, Belgium
†Digital Security Group, Radboud University
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
Email: {jo.vliegen, nele.mentens}@kuleuven.be, lejla@cs.ru.nl

***Abstract*—This paper discusses the first design of an ASIC coprocessor for Elliptic Curve Cryptography (ECC) using the complete addition law of Renes et al. The main reason for using the complete addition law is the reduced vulnerability to side-channel analysis (SCA) attacks, since point addition and point doubling can be performed with the same addition formulas. Further, all inputs are valid, so there is no need for conditional statements handling special cases such as the point at infinity. The proposed hardware architecture is optimized for area efficiency, targeting applications such as smart cards and RFID tags. A bottom-up design approach is used, minimizing the total implementation area by optimizations in each abstraction layer. The design implements a full-word Montgomery Multiplier ALU (MMALU) with built-in adder functionality. Additionally, an exploration is done on the design parameters of the MMALU and the scheduling of the modular operations in order to minimize the size of the register file. For point multiplication, a Montgomery ladder is implemented with the option of randomizing the execution order of the point operations as a countermeasure against SCA attacks. The post-synthesis implementation results are generated using the open source NANGATE45 library.**

***Index Terms*—Elliptic Curve Cryptography (ECC), complete addition formulas, Montgomery ladder, balanced point operations, side-channel analysis (SCA), Montgomery multiplication, ASIC coprocessor**

I. Introduction

The amount of embedded systems used in everyday life grows rapidly in devices like mobile cell phones, RFID tags, IoT devices, etc. These applications introduce new challenges concerning the protection of data and communication. First of all, they have design constraints due to the very limited implementation surface and energy budget. The second challenge is the physical accessibility of the devices to potential malicious users, which makes the devices vulnerable to side-channel analysis attacks (SCA) [1]. The goal of a SCA attack is not to break the cryptographic algorithms mathematically, but to extract secret information from the physical implementation of the algorithms. For example, the processed information leaks through the power consumption, the timing behavior and the electromagnetic radiation of the device.

When embedded devices need public-key cryptography, Elliptic Curve Cryptography (ECC) [2] [3] is preferred, thanks to smaller key sizes compared to e.g. RSA [4]. This leads to a reduction of the power consumption and the computational resources. ECC was introduced in 1985 and 1987 independently by Victor Miller [2] and Neal Koblitz [3], respectively. They proposed the use of a group of points on an elliptic curve (EC) to create discrete-log based cryptosystems and defined the addition law for the resulting group structure. This addition law results in a different set of equations for a pair of identical points and for a pair of different points in the group. These equations are called point doubling and point addition, respectively. The most straightforward way of implementing an EC point multiplication, i.e. the basic operation in an EC-based cryptosystem, is through iterative conditional branching of point doublings and point additions. This is disadvantageous for SCA resistance, because the conditional branching reveals information on the executed operation through side channels, as shown by Örs et al. on an FPGA implementation [5].

In [6], Bernstein and Lange show the benefits of complete addition formulas, which use the same set of equations for point doubling and point addition. This leads to constant-time and exception-free implementations, mitigating the behavioral effects of branching. While the authors of [6] present their formulas for specific types of curves over binary extension fields, namely Edwards curves, Renes et al. are the first to propose complete addition formulas for the broadly used Weierstrass curves over prime fields. The authors of [7] also notice that in some cases, performance loss will be unnoticeable in comparison to traditional incomplete formulas.

Our contribution. We present the first ASIC implementation of the complete formulas introduced in [7]. In combination with the implementation of the Montgomery ladder algorithm for point multiplication, this results in an inherently balanced implementation (without dummy operations) of ECC over Weierstrass curves. The design is optimized for area in the interest of lightweight embedded applications in three ways. (1) A full-word Montgomery multiplier with integrated adder functionality is designed, eliminating the need for separate modular addition hardware. (2) A careful exploration of the design parameters and bounds of the Montgomery multiplier is done, in order to minimize the operation count in the point addition formula. (3) The size of the register file is minimized by optimizing the number of registers, by reducing

the number of writable registers (through the intelligent use of shift registers), and by optimizing the size of the input multiplexers (by limiting left-operand and right-operand accessibility). Important to note is that a completely balanced operation of the coprocessor mainly protects against simple power analysis (SPA) attacks. Therefore, the randomization of the point operations is integrated as well as a countermeasure against differential power analysis (DPA) attacks.

Organization. Section II gives a brief overview of the necessary preliminaries and used notations. In Sect. III, a summary of related literature on complete formula implementations is given. Next, Sect. IV discusses the design choices and the experimental setup. Finally, Sects. V and VI elaborate on the results and formulate a conclusion, respectively.

II. Preliminaries

ECC is based on a group structure on an EC. An EC is a set of points (x, y), which are solutions of a polynomial equation defined over a Galois field of the following form[(1)]:

$$y^2 = x^3 + ax + b \tag{1}$$

Prime fields, i.e. $GF(p)$, and binary extension fields, i.e. $GF(2^n)$, are the most commonly used finite fields in the context of ECC. In this paper, only prime fields are considered and therefore a and b in Eq. (1) are constants in $GF(p)$ and p is a large prime.

Next, an additive Abelian group can be obtained by defining an addition law $(+)$ on an EC that is nonsingular, requiring $4a^3 + 27b^2 \neq 0 \mod p$. The elements in the group are the points on the EC, along with an additional point called the point at infinity. The point at infinity is denoted by $\mathcal{O}$ and can be expressed as $\mathcal{O} = (x, \infty)$. The obtained Abelian group can be represented as follows:

$$\big(\{(x, y) \mid x, y \in GF(p) \text{ satisfying Eq. } (1)\} \cup \mathcal{O}, +\big) \tag{2}$$

The addition law can be implemented using the following operations modulo p: addition, subtraction, multiplication and inversion. In hardware, inversions modulo p are very costly operations. Nevertheless, they can be avoided when embedding the elliptic curve in the projective plane, i.e. $\mathbb{P}^2(GF(p))$. In order to do so, every point (x, y) is mapped to $(x : y : 1)$ and the point at infinity is mapped to $(0 : 1 : 0)$. The projections $(x : y : z)$ and $(\lambda x : \lambda y : \lambda z)$ are equal, and thus $(\frac{x}{z} : \frac{y}{z} : 1)$ equals $(x : y : z)$. When embedding the EC in the projective space, Eq. (1) becomes

$$y^2 z = x^3 + axz^2 + bz^3. \tag{3}$$

A series of point additions is called a point multiplication or scalar multiplication and is defined as follows:

$$mP = \underbrace{P + P \cdots + P}_{\text{m times}}, \tag{4}$$

with P an element of an EC group structure and m a positive integer. For negative m, Eq. 4 becomes $m(-P)$. Point multiplication is used in ECC schemes, because the security relies on the elliptic curve discrete logarithm problem (ECDLP). The ECDLP states that it is unfeasible to calculate m, such that $Q = mP$, when the points P and Q are known.

A commonly used technique to design implementations of such ECC schemes, is a bottom-up design approach, wherein each abstraction level uses the operations of the abstraction level below. Fig. 1 depicts the different abstraction levels for ECC applications. The remainder of this section discusses the different abstraction levels up to the point multiplication and introduces the algorithms we use in our design.

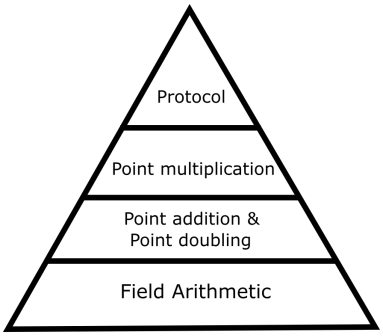

Fig. 1: Abstraction levels in the design of an ECC coprocessor.

A. Field Arithmetic

At the lowest level are the field arithmetic operations. When using the projective representation of the EC, only modular addition, modular subtraction and modular multiplication suffice to implement the addition laws. Modular multiplication modulo p is significantly more complex than modular addition and subtraction, due to the required trial divisions. Therefore, a variety of techniques were designed to speed up or scale down this operation in hardware. For modular multiplication, Montgomery multiplication is a popular technique with a small chip area, a low power consumption and a high throughput in mind. The Montgomery Modular Multiplication (MMM) algorithm was introduced by Peter Montgomery in 1985 [8]. In 1999, Colin Walter suggested an improvement, which made the final reduction at the end of the original algorithm unnecessary by introducing modified input bounds as well as a lower bound for the Montgomery parameter [9]. This improvement is advantageous for SCA resistance since it leads to a time-constant implementation. The Montgomery multiplication algorithm, which we use in our design, is given in Algorithm 1.

Algorithm 1: Montgomery Modular Multiplication [9]

Input : $A = (a_{n-1}, \ldots, a_0)_r, B = (b_{n-1}, \ldots, b_0)_r, p = (p_{n-1}, \ldots, p_0)_r, R$ (with $RR^{-1} = 1 \mod p$)
Output: $S = \text{MMM}(A, B) = ABR^{-1} \mod p$

```
S := 0;
for i ← 0 to n − 1 do
    q_i := (s_0 + a_i b_0)(−p_0^{-1}) mod r;
    S := (S + a_i B + q_i p) div r;
end
Return S;
```

(1) This holds for short Weierstrass elliptic curves, which are isomorphic with every possible elliptic curve.

In Algorithm 1, a_i stands for the i-th digit of the word A. When A is an n-digit number expressed in base r, A can be written as $A = \sum_{i=0}^{n-1} a_i r^i$. B and p are represented in a similar way. The power of the Montgomery algorithm is the ability to calculate a modular operation by replacing costly trial divisions by a prime with divisions by a power of 2. The latter comes for free in hardware, by implementing logical shift operations. MMM, working on the input operands A and B, computes $ABR^{-1} \mod p$, where p is the modulus and R^{-1} is an element of the finite field satisfying $RR^{-1} - pp' = 1$ or identically $RR^{-1} = 1 \mod p$[2][3]. R is a power of two, i.e. r^n, such that $R = r^n > p$; R is also called the Montgomery parameter. The algorithm becomes useful when the operands are first transformed to Montgomery representation, mapping A and B to respectively $A_{Mont} = AR \mod p$ and $B_{Mont} = BR \mod p$. This way, Montgomery multiplication can be used to calculate modular multiplications, because the algorithm ensures $S_{Mont} = SR \mod p = \text{MMM}(A_{Mont}, B_{Mont})$, with $S = AB \mod p$. At the end of the calculations in the Montgomery domain, the result needs to be converted back. As a consequence, Montgomery multiplication requires two additional steps. However, this overhead becomes negligible when a large series of consecutive operations is performed, which is the case in the context of scalar multiplication.

B. Point Addition & Point Doubling

The next abstraction layer involves the point operations, namely point addition and point doubling. As mentioned in Sect. I, point doubling and point addition use a different set of equations in most addition laws. When implemented in hardware or software, this leads to different execution times and power traces for both, inevitably leaking information on the scalar m in the computation of the point multiplication mP. Evidently, this side-channel leakage needs to be avoided to avert weakening the implemented ECC scheme.

Therefore, we use the complete addition law proposed in [7], which uses the same addition formulas for point addition and point doubling. More specifically, we use Algorithm 7 in [7], since it has a minimal number of operations and no input restrictions. This algorithm targets short Weierstrass curves with $a = 0$, i.e. j-invariant 0 curves. These curves are also used in practice, such as the *secp256k1* curve used in the Bitcoin Protocol [10].

The concerned formulas of Algorithm 7 in [7], giving new point coordinates $(X_3 : Y_3 : Z_3)$ dependent on input points $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$, are

$$\begin{aligned} X_3 =& (X_1Y_2 + X_2Y_1)(Y_1Y_2 - 3bZ_1Z_2) \\ & - 3b(Y_1Z_2 + Y_2Z_1)(X_1Z_2 + X_2Z_1), \\ Y_3 =& (Y_1Y_2 + 3bZ_1Z_2)(Y_1Y_2 - 3bZ_1Z_2) \\ & + 9bX_1X_2(X_1Z_2 + X_2Z_1), \\ Z_3 =& (Y_1Z_2 + Y_2Z_1)(Y_1Y_2 + 3bZ_1Z_2) \\ & + 3X_1X_2(X_1Y_2 + X_2Y_1). \end{aligned} \tag{5}$$

C. Point Multiplication

The most straightforward algorithm for point multiplication is the double-and-add algorithm, consisting of consecutive point additions and point doublings. To overcome the side-channel leakage caused by the conditional branches in the algorithm, a better solution is to use the Montgomery ladder algorithm for point multiplication. The Montgomery ladder, as presented by Peter Montgomery in [11], is shown in Algorithm 2. In contrast with the double-and-add method, each iteration executes the same operations, namely one point addition and one point doubling.

Algorithm 2: Montgomery ladder [11]

Input : P, $m = (m_{t-1}, \ldots, m_0)_2$ with $m_{t-1} = 1$
Output: $R = mP$

$R_0 := P$; $R_1 := 2P$;
for $i \leftarrow t-2$ **to** 0 **do**
  if $m_i = 1$ **then**
    $R_0 := R_0 + R_1$; $R_1 := 2R_1$;
  else
    $R_1 := R_0 + R_1$; $R_0 := 2R_0$;
  end
  $\{R_1 - R_0$ *remains invariant*$\}$
end
$R := R_0$;
Return R;

When using the Montgomery ladder in Algorithm 2, first a point addition and then a point doubling is performed, independent of m_i. In [12], the authors introduce additional randomization by using a random bit which decides on the execution order of both operations. This order introduces an additional uncertainty independent of the key, further complicating SCA attacks. The resulting Montgomery ladder is shown in Algorithm 3 and is applied in our ASIC design.

III. Related Work

The first work to introduce a fully balanced ECC implementation was by Batina et al. [13]. The authors modified the non-complete addition formulas over binary extension fields in order to make them balanced. For point multiplication, the Montgomery ladder algorithm was used. The implementation was implemented on an FPGA and the resistance against an SPA attack was evaluated.

Examples of elliptic curves with complete addition laws are Edwards curves [6], twisted Edwards curves [14] and twisted Hessian curves [15]. They all operate over binary extension fields. The work of Renes et al. [7] was the first to propose

[2] Bézout's identity ensures the existence of R^{-1} [8] [9].
[3] R^{-1} and p' can be calculated using the extended Euclidean algorithm.

Algorithm 3: Montgomery ladder for point multiplication with random order execution [12]

Input : P, $m = (m_{t-1}, \ldots, m_0)_2$ with $m_{t-1} = 1$, random bits $r_{t-2}, \ldots, r_0$
Output: $R = mP$

1 $R_0 := P$;
2 $R_1 := 2P$;
3 **for** $i \leftarrow t-2$ **to** 0 **do**
4 **if** $m_i = 1$ **then**
5 **if** $r_i = 0$ **then**
6 $T_0 := R_0 + R_1$; $T_1 := 2R_1$;
7 $R_0 := T_0$; $R_1 := T_1$;
8 **else**
9 $T_1 := 2R_1$; $T_0 := R_0 + R_1$;
10 $R_0 := T_0$; $R_1 := T_1$;
11 **end**
12 **else**
13 **if** $r_i = 0$ **then**
14 $T_1 := R_0 + R_1$; $T_0 := 2R_0$;
15 $R_0 := T_0$; $R_1 := T_1$;
16 **else**
17 $T_0 := 2R_0$; $T_1 := R_0 + R_1$;
18 $R_0 := T_0$; $R_1 := T_1$;
19 **end**
20 **end**
21 $\{R_1 - R_0$ *remains invariant*$\}$
22 **end**
23 $R := R_0$;
24 Return R;

complete addition formulas on Weierstrass curves over prime fields.

In [16], Massolino et al. present the first FPGA implementation of the formulas in [7]. The result is a competitive design emphasizing parallelization possibilities. The design executes a number of field operations simultaneously by using two up to six processors, which increases throughput, but also silicon area. In [17], Chmielewski et al. implement and evaluate three FPGA implementations of the formulas in [7]: a non-protected architecture and two architectures protected by randomization countermeasures for DPA protection.

Our paper realizes the first ASIC implementation of the complete formulas of Renes et al., optimized towards minimal silicon area.

IV. Design

This section elaborates on the functionality and design choices of the ASIC implementation in a bottom-up way, following the abstraction layers in Fig. 1. After giving an overview of the design parameters of full-word Montgomery multipliers, the Montgomery Modular ALU (MMALU) is discussed, which enables modular multiplication, addition and subtraction. Hereafter, the design choices for the control logic, implementing the complete point addition formulas in Eq. 5, are explained. Finally, the Montgomery ladder implementation for point multiplication is discussed.

A. Design Parameters of the MMM

This paragraph discusses the influence of certain design parameters on the function of Algorithm 1, inspired by the doctoral thesis of Lejla Batina [18]. Especially important is the relation between the input bounds and the Montgomery parameter R. This relationship influences speed and area, because it is directly linked to the required number of iterations of the algorithm and the size of the input and output registers. To understand the influence of these parameters, the operation of the algorithm is explained below. For simplicity, it is assumed that all data are expressed in binary form and therefore the base r is 2.

1) Working principle: The correctness of the MMM algorithm depends on two statements, both easily verified via induction. The first statement,

$$0 \leq S < p + B, \tag{6}$$

holds during each operation of the for loop in Algorithm 1 and ensures the output remains bounded. After n iterations, $S = ABR^{-1} \mod p$ with $R = r^n$. This is verifiable with the help of the following statement, in which $Q = (q_{n-1}, \ldots, q_0)$ (cfr. Alg. 1):

$$RS = AB + Qp \Rightarrow S = ABR^{-1} \mod p \tag{7}$$

2) Divisibility by 2: In Algorithm 1, addition with $q_i p$ always ensures an outcome divisible by 2 in the last step of the for loop. In other words, the least significant bit (LSB) of the outcome on line 4 is 0. This is easily validated by replacing q_i with the expression on line 3 in Algorithm 1. As a result, the LSB of S becomes:

$$\begin{aligned} & s_0 + a_i b_0 + q_i p_0 \\ = \; & s_0 + a_i b_0 + \big((s_0 + a_i b_0)(-p_0^{-1})\big) p_0 \\ = \; & s_0 + a_i b_0 + (s_0 + a_i b_0)(-1) \\ = \; & 0 \end{aligned}$$

3) Upper Bound on Q: Q cannot be bigger than $R - 1$, because Q has maximum n bits and therefore has a maximum value of $r^n - 1 = R - 1$.

4) Lower Bound on the Inputs: A design parameter with significant influence on the implementation is the bound on the inputs; it directly determines the size of the inputs, the output and the intermediate registers in the implementation. Let us assume that the inputs are bounded by a multiple k of the modulus p:

$$A, B < kp \overset{\text{Eq. } 6}{\Rightarrow} \; S < p + kp = (k+1)p$$

Thus, when the inputs are smaller than k times the modulus p, the input registers need to store $\lceil log_r(k) \rceil$ more bits than the number of bits needed to represent p. Further, the intermediate result needs $\lceil log_r(k+1) \rceil$ bits more than p.

5) Lower Bound on the Montgomery Parameter: The Montgomery parameter ($R = r^n$) determines the number of iterations (n) to obtain the result. We assume that the lower bound on the Montgomery parameter is a multiple l of p.

Parameter	Bound	# bits
Prime (p)	-	$\boldsymbol{x}$
Montgomery parameter ($R = r^n$)	$> \boldsymbol{lp}$	$n+1$
Inputs	$< \boldsymbol{kp}$	$x + \lceil log_2(k) \rceil$
# iterations	$= n$	$\lceil log_2(n) \rceil$
Intermediate result (after shift)	$< (k+1)p$	$x + \lceil log_2(k+1) \rceil$
Intermediate result (before shift)	$< 2(k+1)p$	$x + \lceil log_2(k+1) \rceil + 1$
Output	$< (\frac{k^2}{l} + 1)p$	$x + \lceil log_2(\frac{k^2}{l} + 1) \rceil$

TABLE I: Influence of the boundaries of the MMM design parameters on register sizes

6) Lower Bound on the Output: The relation between the Montgomery parameter and the input bound has an important effect on the output of the MMM. Assuming that both the inputs and the Montgomery parameter are bounded by a multiple ($< kp$ and $> lp$, respectively) of p, we obtain the following bound on S:

$$\begin{aligned} & A, B < kp \text{ and } R > lp \\ \overset{\text{Eq. 7}}{\Rightarrow}\ & S = \frac{AB}{R} + \frac{Qp}{R} \\ \Rightarrow\ & S < \frac{k^2p^2}{R} + \frac{(R-1)p}{R} \\ \Rightarrow\ & S < \frac{k^2p^2}{lp} + \frac{(R-1)p}{R} \\ \Rightarrow\ & S < \frac{k^2}{l}p + p - \frac{p}{R} \\ \Rightarrow\ & S < \big(\frac{k^2}{l} + 1\big)p \end{aligned}$$

As an example, assume that the input bounds are $A, B < 2p$ and the Montgomery parameter is larger than $4p$. Then the output remains smaller than $2p$, in accordance with the original result in the paper of Walter, enabling MMM without final subtraction [9].

7) Summary: Table I gives an overview of the design parameters of the MMM for a prime of x bits. By carefully selecting the lower bound of the Montgomery parameter (lp) and the upper bound of the inputs (kp), the number of iterations (n) and the bound on the output and the intermediate values can be determined. These bounds determine the register sizes in the design and are therefore important design parameters.

B. Montgomery Modular ALU

Most literature on Montgomery multipliers in hardware is focused on fast implementations of the MMM algorithm, often at the expense of area efficiency. For these fast implementations, systolic arrays (e.g. [19]), pipelining (e.g. [20]) and high-radix approaches (e.g. [21]) are very popular. However, this paper focuses on low silicon area implementations. As a result, a full-word implementation of the Montgomery multiplication algorithm without final subtraction [9], inspired by the Modular ALU (MALU) design of Sakiyama et al. [22], was chosen.

	Upper limit	Number of bits
Prime	p	k
Montgomery parameter	$2^{(k+4)} (> 16p)$	$k+5$
Inputs	$4p$	$k+2$
# iterations	$k+4$	$\lceil log_2(k+4) \rceil$
Intermediate result (after shift)	$5p$	$k+3$
Intermediate result (before shift)	$10p$	$k+4$
Output	$2p$	$k+1$

TABLE II: Necessary recourses of the MMALU, with corresponding upper bounds for $R > 16p$

As discussed in Sect. II-A, the MMM has a bounded output dependent on the upper bound of the inputs and the lower bound of the Montgomery parameter R. When the inputs of the MMM are limited to numbers smaller than $4p$, the intermediate results of Algorithm 1 are numbers smaller than $5p$ (after the shift operation). If we choose the Montgomery parameter to be larger than $16p$, the output is bounded by $2p$. Table II summarizes the previous reasoning.

The functionality of the MMALU can be selected using two control signals, cmd and sub. This can also be seen in Fig. 2. When cmd is set to 0, a MMM is performed on the input operands. When operating in MMM mode, the sub control signal selects either the MMM of the two input operands, i.e. when sub equals 0, or otherwise the MMM of one input operand with the second input operand equal to 1. The second feature can be used to scale down the input operand to ensure an outcome smaller than or equal to p, because

$$\begin{aligned} & A < 4p \text{ and } B = 1 \text{ and } R > 16p \\ \overset{\text{Eq. 7}}{\Rightarrow}\ & RS = AB + Qp \\ \Rightarrow\ & RS < 4p \cdot 1 + (R-1)p \\ \Rightarrow\ & RS < (R+3)p \\ \Rightarrow\ & S < p + \frac{3p}{16p} \\ \Rightarrow\ & S \leq p \end{aligned}$$

Normally, the MMM of an input operand and 1 transforms the input from the Montgomery representation back to the original domain. However, when performed on all coordinates of a point on the EC, this operation only results in a scaling. Remember that the complete formulas to be implemented have inputs and an output in the projective space. Consequently, the formulas yield the same output for scalar multiples of either of the input points, i.e. the output remains unaffected when loading a set of inputs (X, Y, Z) or $(\lambda X, \lambda Y, \lambda Z)$ for either input point. It should also be noted that this property renders transformation of the input coordinates, i.e. $\{X_1, Y_1, Z_1, X_2, Y_2, Z_2\}$, to their Montgomery representation and transformation of the output coordinates back to the original domain unnecessary. This follows directly from $(X : Y : Z) = (RX : RY : RZ)$.

Fig. 3 gives a conceptual view of the MMALU when cmd equals 0. This is a straightforward implementation of Algorithm 1 with two ripple carry adders (RCA).

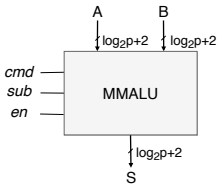

Fig. 2: Interface of the MMALU functional block.

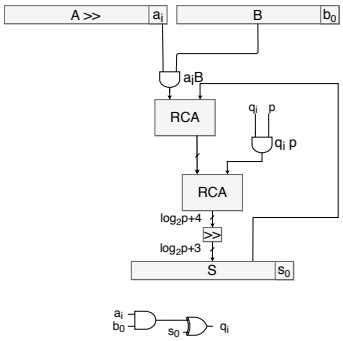

Fig. 3: Architecture of MMALU when $cmd = 0$.

Fig. 4 shows the functionality of the MMALU when $cmd = 1$. In this mode, the integrated adder functionality is selected. As a result, two field element inputs can be added or subtracted depending on the value of sub. When sub equals 0, the adder adds two $2p$ inputs to an output of maximum $4p$. Additionally, subtraction is implemented when sub equals 1. The same output range is available, due to the addition of $2p$ with use of the second RCA, after subtracting with the first RCA. Without this extra addition of $2p$, the output would be larger than $-2p$ and smaller than $2p$. When $sub = 1$, all bits of B are inverted and the carry input of the RCA is set to 1 to obtain the 2's complement representation of B.

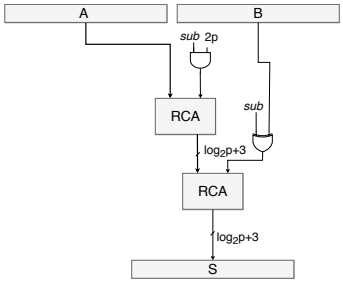

Fig. 4: Architecture of MMALU when $cmd = 1$.

In summary, the MMALU can perform addition and subtraction without dedicated modular addition hardware, which ensures outputs between 0 and $4p$ when handling inputs smaller than $2p$. Therefore, an output of the MMALU in MMM mode, which takes $4p$ inputs and returns a $2p$ output, can be used as input of a subsequent addition. More generally, a series of additions, subtractions and multiplications becomes possible, when the input bounds of each operation are respected. This also means that a series of subsequent additions is not always possible without intermediate scaling or MMM, which can increase the number of operations of the respective algorithm. Nevertheless, other properties of the MMM are useful for performing such a series of operations. For example, three outputs of Montgomery multiplications of values smaller than $2p$ can be added resulting in an output smaller than $4p$. This is because an MMM with two $2p$ input values results in the following output bound:

Functionality	Input	Output
add/subtract	$2 \times 2p$	$1 \times 4p$
add/subtract	$3 \times \frac{5}{4}p$	$1 \times 4p$
add	$2 \times p$	$1 \times 2p$
multiply	$2 \times 4p$	$1 \times 2p$
multiply	$2 \times 2p$	$1 \times \frac{5}{4}p$
scale	$1 \times 4p$	$1 \times p$

TABLE III: Bounds of the inputs and outputs of the MMALU

$$
\begin{aligned}
& A, B < 2p \text{ and } R > 16p \\
\overset{\text{Eq. 7}}{\Rightarrow}\ & S = \frac{AB}{R} + \frac{Qp}{R} \\
\Rightarrow\ & S < \frac{4 \cdot p^2}{R} + \frac{(R-1)p}{R} \\
\Rightarrow\ & S < \frac{4 \cdot p^2}{16 \cdot p} + \frac{(R-1)p}{R} \\
\Rightarrow\ & S < \big(\frac{1}{4} + 1\big)p - \frac{p}{R} \\
\Rightarrow\ & S < \frac{5}{4}p
\end{aligned}
$$

Table III specifies all functions of the MMALU. All calculations must respect these upper bounds on the input, otherwise a correct operation of the MMALU is not guaranteed. As a consequence of the optimizations made, not all algorithms will be directly compatible with this design, i.e. without appending scaling operations.

C. Point Addition

Moving one abstraction level up in Fig. 1, the next step is to implement the complete point addition using Algorithm 7 in [7], given in Eq. 5, using the previously designed MMALU. An overview of common architectures for ECC processor designs is given in [23]. We implement a finite state machine (FSM) to implement the rules for point addition in combination with a register file to store the intermediate results. The output of the FSM consists of a write enable (WE) signal, a write address (Address) and two read adresses (LO and RO) for the register file. It also contains an OPCODE field to control the operation of the MMALU.

In [7], the authors already present an implementation algorithm for the proposed addition law in the form of consecutive field operations. Algorithm 4 is a modified version of this sequence of operations, taking into account the restrictions given in Table III and the optimizations explained in the following paragraph, resulting in a minimization of the number of registers and the size of the multiplexers.

Originally, Algorithm 7 in [7] presumes 14 registers, i.e. 6 input, 3 output and 5 temporary registers. Our design utilizes

[(4)]Register data of t_1 is shifted into t_0 ($t_0 \leftarrow t_1$)

Algorithm 4: Complete, projective point addition for prime order j-invariant 0 short Weierstrass curves $E/F_p : y^2 = x^3 + b$

Require: $P = (X_1 : Y_1 : Z_1)$, $Q = (X_2 : Y_2 : Z_2)$ on $E : Y^2Z = X^3 + bZ^3$ and $b3 = 3 \cdot b$

Ensure: $(X_3 : Y_3 : Z_3) = P + Q$

1. $t_1 \leftarrow X_1 \cdot X_2$	2. $t_1 \leftarrow Y_1 \cdot Y_2$ (4)	3. $t_2 \leftarrow Z_1 \cdot Z_2$
4. $t_3 \leftarrow X_1 + Y_1$	5. $t_4 \leftarrow X_2 + Y_2$	6. $t_3 \leftarrow t_3 \cdot t_4$
7. $t_4 \leftarrow t_0 + t_1$	8. $t_3 \leftarrow t_3 - t_4$	9. $t_4 \leftarrow Y_1 + Z_1$
10. $Y_2 \leftarrow Y_2 + Z_2$	11. $Y_1 \leftarrow t_4 \cdot Y_2$	12. $Y_2 \leftarrow t_1 + t_2$
13. $t_4 \leftarrow Y_1 - Y_2$	14. $Y_2 \leftarrow X_1 + Z_1$	15. $Z_1 \leftarrow X_2 + Z_2$
16. $Z_1 \leftarrow Y_2 \cdot Z_1$	17. $Y_1 \leftarrow t_0 + t_2$	18. $Y_1 \leftarrow Z_1 - Y_1$
19. $X_1 \leftarrow t_0 + t_0$	20. $t_1 \leftarrow X_1 + t_0$ (4)	21. $Y_2 \leftarrow b3 \cdot t_2$
22. $Z_1 \leftarrow t_0 + Y_2$	23. $t_1 \leftarrow t_0 - Y_2$ (4)	24. $Y_1 \leftarrow b3 \cdot Y_1$
25. $X_1 \leftarrow t_4 \cdot Y_1$	26. $Y_2 \leftarrow t_3 \cdot t_1$	27. $X_1 \leftarrow Y_2 - X_1$
28. $Y_1 \leftarrow Y_1 \cdot t_0$	29. $Y_2 \leftarrow t_1 \cdot Z_1$	30. $Y_1 \leftarrow Y_2 + Y_1$
31. $t_1 \leftarrow t_0 \cdot t_3$ (4)	32. $Z_1 \leftarrow Z_1 \cdot t_4$	33. $Z_1 \leftarrow Z_1 + t_1$
34. $X_1 \leftarrow scale(X_1)$	35. $Y_1 \leftarrow scale(Y_1)$	36. $Z_1 \leftarrow scale(Z_1)$

only 11 registers, reusing 3 input registers (X_1, Y_1 and Z_1) as output registers. In addition, the algorithm ensures only 3 address bits to select the left and right operand for the MMALU and to select the write address. This is accomplished by organizing the formulas in Algorithm 4 such that maximum 8 registers need to be accessible as left or right operand. In order to reduce the number of writable registers, register t_0 is made not directly writable; instead, when writing to t_1, the value of t_1 is shifted to t_0. This is incorporated in Algorithm 4, which writes to t_0 and t_1 alternately. Figure 5 gives the resulting architecture implementing the point addition.

D. Point Multiplication

The next design stage is implementing point multiplication with a Montgomery ladder as described in Algorithm 2. Important to note is that this algorithm assumes the MSB of m is not 0, therefore cutting the key space in half and making brute force attacks more feasible. This assumption was made to avoid addition with the point at infinity, which was not possible whilst using the typical addition formulas. However, this is not the case for their complete counterparts. Therefore, the algorithm can be altered, restoring all possibilities in the key space. In order to do so, R_0 and R_1 are initialized to $\mathcal{O}$ and P respectively and the loop must iterate through all bits of m, i.e. from $t - 1$ to 0.

It should be noted that by using the Montgomery ladder instead of the double-and-add method, a possible speedup and reduction of resources is not feasible. The reason is that the output of a point addition is directly written in the input registers. In other words, the output of a point addition will automatically be the input for the next point addition without the need for additional loading. In the double-and-add method, one input remains unmodified. Consequently, only one input should be reloaded during the next iteration. However, the benefits of a balanced ladder method outweigh the lost speed gain and resource reduction, in spite of the lost opportunity.

V. Results

All blocks in the design were implemented in VHDL and simulated with ModelSim PE. The ASIC area, expressed in gate equivalents (GE), and the maximum clock frequency are calculated by Synopsis Design Compiler 2016 using the open source NANGATE45 library [24]. For testing of correctness of operation, test vectors are constructed in Magma [25]. The test bench also implements the *secp160k1* and *secp256k1* curve used in the Bitcoin protocol.

For the implementation without randomization in the Montgomery ladder, Table IV shows the utilized silicon area in kilo gate equivalents and the maximum clock frequency of the design with respect to the size of the prime. For an easy comparison to future designs, the silicon area is given with and without the inclusion of the register file. Also, for the *secp160k1* and the *secp256k1* curves, which are described in [26], the duration of a single scalar multiplication is simulated at maximum frequency. The results show that the size of the prime and the area of the implementation are linearly correlated. Each increase of 32 bits for the prime, results in an increase in implementation size of approximately 6.5 kGE. In contrast, the maximum clock frequency decreases with an increasing prime size. This decline is slower than linear decrease. For completeness, Table V gives the results for an implementation with the randomization of the order of the point operations. Note that the speed is unaffected. However, the area increases due to the extra temporary registers in the Montgomery ladder.

VI. Conclusion and Future Work

This work presents the first ASIC coprocessor design implementing ECC with complete formulas in $GF(p)$. The design consists of a fully balanced implementation, targeting protection against SPA attacks. As a first step towards DPA protection, the randomization of the execution order of the point operations was incorporated in the Montgomery ladder.

# bits	Area (kGE)	Area w/o reg. file (kGE)	max. Freq. (MHz)	Scalar mult. (ms)
64	17.77	10.03	333.33	-
96	26.30	14.77	250.00	-
128	34.12	18.64	166.67	-
160	42.48	23.22	166.67	5.52 (secp160k1)
192	51.02	27.96	142.86	-
224	59.12	32.45	111.11	-
256	66.51	36.06	100.00	23.06 (secp256k1)

TABLE IV: Results generated using Design Compiler 2016 with the NANGATE45 library, with and without (w/o) randomization of operations in the Montgomery ladder.

# bits	Area (kGE)	Area w/o reg. file (kGE)
64	21.58	13.84
96	31.91	20.38
128	41.51	26.03
160	51.08	31.81
192	61.35	38.29
224	71.59	44.92
256	81.89	51.45

TABLE V: Results generated using Design Compiler 2016 with the NANGATE45 library, with and without (w/o) randomization of operations in the Montgomery ladder.

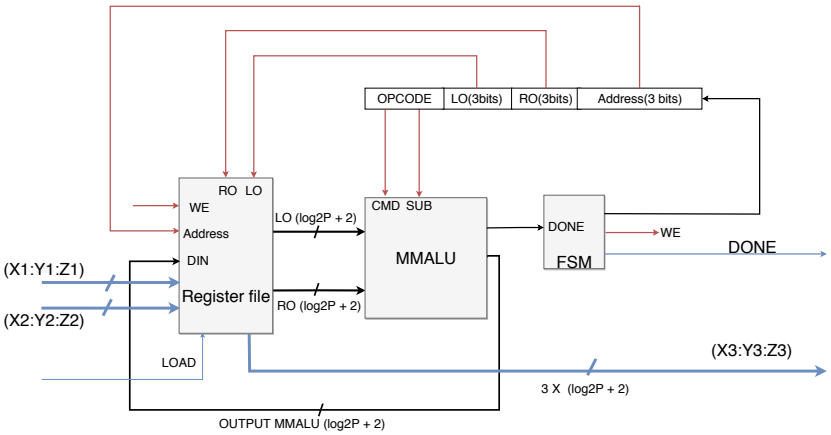

Fig. 5: Architecture of the point addition module.

Additionally, the design was optimized for minimal silicon area by (1) using complete formulas for short Weierstrass curves with $a = 0$, i.e. j-invariant 0 curves; (2) performing datapath optimizations in the Montgomery modular ALU (MMALU) with integrated adder functionality; (3) minimizing the size of the register file by exploring the design parameters of the MMALU and intelligent scheduling of the modular operations. The silicon area, the maximum operating frequency and the scalar multiplication execution time are evaluated using Synopsys Design Compiler 2016.

Future work includes a side-channel analysis of the proposed architecture. We expect that the design will be resistant against SPA attacks and some DPA attacks. More countermeasures need to implemented, however, to provide a fully protected implementation. We also expect that a further reduction of the silicon area is possible by using scalable approaches to decrease the size of the MMALU.

References

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology – Proceedings of CRYPTO*, ser. LNCS, M. Wiener, Ed., no. 1666. Springer-Verlag, 1999, pp. 388–397.

[2] V. Miller, "Uses of elliptic curves in cryptography," in *Advances in Cryptology – Proceedings of CRYPTO*, ser. LNCS, H. C. Williams, Ed., no. 218. Springer-Verlag, 1985, pp. 417–426.

[3] N. Koblitz, "Elliptic curve cryptosystem," *Mathematics of Computation*, vol. 48, pp. 203–209, 1987.

[4] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[5] S. B. Örs, E. Oswald, and B. Preneel, "Power-analysis attacks on an FPGA – first experimental results," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, C. D. Walter, Ç. K. Koç, and C. Paar, Eds., no. 2779. Springer, 2003, pp. 35–50.

[6] D. J. Bernstein and T. Lange, "Faster addition and doubling on elliptic curves," in *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, ser. LNCS, K. Kurosawa, Ed., no. 4833. Springer, 2007, pp. 29–50.

[7] J. Renes, C. Costello, and L. Batina, "Complete addition formulas for prime order elliptic curves," in *35th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, ser. LNCS, M. Fischlin and J.-S. Coron, Eds., no. 9665. Springer, 2016, pp. 403–428.

[8] P. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[9] C. Walter, "Montgomery exponentiation needs no final subtractions," *Electronics Letters*, vol. 35, no. 21, pp. 1831–1832, 1999.

[10] "Bitcoin," https://github.com/bitcoin/bitcoin/tree/master/src/secp256k1, 2017.

[11] P. L. Montgomery, "Speeding the pollard and elliptic curve methods of factorization," *Mathematics of computation*, vol. 48, no. 177, pp. 243–264, 1987.

[12] L. Batina, J. Hogenboom, N. Mentens, J. Moelans, and J. Vliegen, "Side-channel evaluation of FPGA implementations of binary Edwards curves," in *17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*. IEEE Publishing, 2010, pp. 1248–1251.

[13] L. Batina, N. Mentens, B. Preneel, and I. Verbauwhede, "Balanced point operations for side-channel protection of elliptic curve cryptography," *IEE Proceedings-Information Security*, vol. 152, no. 1, pp. 57–65, 2005.

[14] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters, "Twisted Edwards curves," in *International Conference on Cryptology in Africa (AFRICACRYPT)*, ser. LNCS, S. Vaudenay, Ed., no. 5023. Springer, 2008, pp. 389–405.

[15] D. J. Bernstein, C. Chuengsatiansup, D. Kohel, and T. Lange, "Twisted hessian curves," in *International Conference on Cryptology and Information Security in Latin America (LATINCRYPT)*, ser. LNCS, K. Lauter and F. Rodríguez-Henríquez, Eds., no. 9230. Springer, 2015, pp. 269–294.

[16] P. M. C. Massolino, J. Renes, and L. Batina, "Implementing complete formulas on Weierstrass curves in hardware," in *International Conference on Security, Privacy, and Applied Cryptography Engineering (SPACE)*, ser. LNCS, C. Carlet, M. A. Hasan, and V. Saraswat, Eds., no. 10076. Springer, 2016, pp. 89–108.

[17] Ł. Chmielewski, P. M. C. Massolino, J. Vliegen, L. Batina, and N. Mentens, "Completing the complete ECC formulae with countermeasures," *Journal of Low Power Electronics and Applications*, vol. 7, no. 1, 2017.

[18] L. Batina, "Arithmetic and architectures for secure hardware implementations of public-key cryptography," PhD thesis, 2005.

[19] S. B. Örs, L. Batina, B. Preneel, and J. Vandewalle, "Hardware implementation of a Montgomery modular multiplier in a systolic array," in *International Parallel and Distributed Processing Symposium*. IEEE, 2003.

[20] N. Mentens, K. Sakiyama, B. Preneel, and I. Verbauwhede, "Efficient pipelining for modular multiplication architectures in prime fields," in *17th ACM Great Lakes symposium on VLSI*. ACM, 2007, pp. 534–539.

[21] F. Bernard, "Scalable hardware implementing high-radix Montgomery multiplication algorithm," *Journal of Systems Architecture*, vol. 53, no. 2, pp. 117–126, 2007.

[22] K. Sakiyama, L. Batina, N. Mentens, B. Preneel, and I. Verbauwhede, "Small-footprint ALU for public-key processors for pervasive security," in *Workshop on RFID Security*, vol. 12, 2006.

[23] H. Marzouqi, M. Al-Qutayri, and K. Salah, "Review of elliptic curve cryptography processor designs," *Microprocessors and Microsystems*, vol. 39, no. 2, pp. 97–112, 2015.

[24] NanGate Inc, "NanGate FreePDK45 Open Cell Library," 2018.

[25] "Magma computational algebra system," 2018. [Online]. Available: http://magma.maths.usyd.edu.au/magma/

[26] Certicom Corp., "SEC 2: Recommended elliptic curve domain parameters," 2000. [Online]. Available: http://www.secg.org/SEC2-Ver-1.0.pdf