Eine ALU für die schnelle Berechnung der Kryptographie auf Basis elliptischer Kurven

Mathias Schmalisch, Marc-Sebastian Fiedler, Dirk Timmermann

Institut für Angewandte Mikroelektronik und Datentechnik, Universität Rostock
Richard-Wagner-Str. 31, 18119 Rostock
Tel.: +49 (381) 498 35 36
Fax: +49 (381) 498 36 01
Email: mathias.schmalisch@etechnik.uni-rostock.de

Abstract. In diesem Beitrag wird eine arithmetisch logische Einheit (Arithmetic Logic Unit – ALU) für Operationen im endlichen Körper von $GF(2^m)$ vorgestellt. Diese ALU dient als Grundbaustein für einen Prozessor zur Berechnung der Kryptographie auf Basis elliptischer Kurven. Dazu wird eine kurze Einführung in die Kryptographie auf Basis elliptischer Kurven gegeben und aufgezeigt, welche Berechnungen mit der ALU ausgeführt werden. Anschließend wird der Aufbau und die Funktionsweise der ALU erläutert.

Einführung

In einer Zeit, in der der Trend hin zum mobilen Datenaustausch mehr und mehr an Bedeutung gewinnt, wird die Frage nach der Sicherheit der zu übertragenen Daten immer häufiger gestellt. Die einzige Antwort auf diese Frage bietet die Kryptographie. Denn nur mit ihr ist die Signierung, Authentifizierung und verschlüsselte Datenübertragung möglich. Bei mobilen Geräten kommen aber in der Regel nur recht leistungsschwache Prozessoren zur Anwendung. Diese haben daher nicht die Leistungsfähigkeit, entsprechende kryptographische Algorithmen in ausreichend kurzer Rechenzeit auszuführen. Einen Ausweg aus diesem Dilemma stellen kryptographische Coprozessoren dar, die einen Großteil des Rechenaufwands übernehmen.

Dieser Betrag stellt einen ersten Schritt hin zu einem Coprozessor für Algorithmen auf Basis elliptischer Kurven vor. Für die Kryptographie auf Basis elliptischer Kurven werden vier Operationen im endlichen Körper benötigt, wobei die verwendeten Zahlen so breit wie die Schlüssellänge sind (heutzutage etwa 160 Bit). Das neue an der vorgestellten ALU ist die Berechnung des multiplikativen Inversen in einem eigenen Modul [FIE02]. Somit können Algorithmen angewendet werden, die im Gegensatz zu anderen Coprozessoren ohne Invertierung drei- bis viermal so schnell arbeiten.

Die ALU gliedert sich in drei große Bereiche. Zum ersten Bereich zählen die vier einzelnen Module, welche die Berechnungen im endlichen Körper ermöglichen. Dazu gehören die Addition, die Multiplikation, die Quadrierung und die Invertierung. Der zweite Bereich ist für die Speicherung der Daten verantwortlich. Er enthält eine Registerbank mit 16 Speicherplätzen, in der die Eingangsdaten und die berechneten Werte abgelegt werden können. Der letzte Bereich besteht aus der Steuerlogik für die ALU.

Der Beitrag ist wie folgt aufgebaut. Im ersten Abschnitt werden die Grundlagen der Kryptographie auf Basis elliptischer Kurven vorgestellt. Im zweiten Abschnitt wird der Aufbau der ALU und der einzelnen Funktionen erklärt. Daraufhin werden die Syntheseergebnisse für die ALU vorgestellt und ein Ausblick auf die Erweiterung der ALU zu einem Prozessor für Kryptographie auf Basis elliptischer Kurven gegeben. Im letzten Abschnitt werden die Ergebnisse des Beitrags noch einmal kurz zusammengefasst.

1. Kryptographie auf Basis elliptischer Kurven

Die Kryptographie auf Basis elliptischer Kurven (Elliptic Curve Cryptography - ECC) wurde von Neal Koblitz [KOB87] und Victor Miller [MIL86] unabhängig voneinander vorgestellt. Inzwischen wird diese Art der Kryptographie für verscheiden kryptographische System eingesetzt, z.B. für digitale Signaturen und Schlüsselaustauschprotokolle [IEE99].

Für die Berechnung in Hardware eignen sich elliptischer Kurven über den endlichen Körper $GF(2^m)$ am besten. In diesem Fall hat die Gleichung für die elliptische Kurve folgende Form:

$$E/K: \quad y^2 + xy = x^3 + ax^2 + b \qquad y, x, a, b \in K \tag{1}$$

Damit auf dieser Kurve die Punktoperationen definiert sind, darf diese Kurv nicht singulär werden. Dazu ist die Diskriminante Δ zu bestimmen. Wenn die Diskriminante $\Delta = 0$ ist, habe wir es mit einer singulären Kurve zu tun. Für eine elliptische Kurve nach Gleichung(1), hat die Gleichung für die Diskriminante folgende Form:

$$\Delta = b \tag{2}$$

Nähere Informationen können in [MEN93] nachgelesen werden.

1.1. Skalarmultiplikation

Sämtliche kryptographischen Systeme, die der ECC beruhen, basieren auf der Skalarmultiplikation *k*P*. Dazu wird eine Zahl *k* mit einem Punkt *P* auf der elliptischen Kurve multipliziert. Da auf einer elliptischen Kurve nur die Punktaddition und die Punktverdopplung möglich sind, muss die Skalarmultiplikation auf diesen beiden Operationen abgebildet werden. Dazu gibt es mehrer Algorithmen. Der einfachste Algorithmus ist der sogenannte „Double and Add“ Algorithmus.

Algorithmus 1: Double and Add
INPUT: Integer k > 0 und Punkt P
OUTUT: Q = k * P
$k \leftarrow (k_{n-1}, \ldots, k_1, k_0)_2$
Q ← P
For i from (n - 2) downto 0 do
    Q ← 2 * Q
    If k_i = 1 then
        Q ← Q + P
EndFor
Return Q

Ein weiterer Algorithmus zur Berechnung der Skalarmultiplikation stammt von Peter Montgomery und wird in [MON87] und [LOP99] näher beschrieben und hier kurz vorgestellt. Dieser Algorithmus hat den Vorteil, dass er immer dieselbe Anzahl an Punktoperation benötigt, egal wie das Hamminggewicht (Anzahl der Einsen in *k*) von *k* aussieht. Wohingegen der "Double and Add" Algorithmus abhängig vom Hamminggewicht ist. Ein weiterer Vorteil besteht darin, dass beim Montgomery Algorithmus die Berechnung der *y*-Koordinate eingespart werden kann. Das liegt daran, dass immer zwei benachbarte Punkte berechnet werden. Denn Peter Montgomery hat festgestellt, dass sich aus den *x*-Koordinaten zweier benachbarter Punkte die *y*-Koordinaten berechnen lassen.

Algorithmus 2: Montgomery
INPUT: Integer k > 0 und Punkt P
OUTUT: Q = k * P
$k \leftarrow (k_{n-1}, \ldots, k_1, k_0)_2$
$P_1 \leftarrow P$; $P_2 \leftarrow 2 * P$
For i from (n - 2) downto 0 do
    If $k_i = 1$ then
        $P_1 \leftarrow P_1 + P_2$; $P_2 \leftarrow 2 * P_2$
    Else
        $P_1 \leftarrow 2 * P_1$; $P_2 \leftarrow P_1 + P_2$
EndFor
Return ($Q = P_1$)

1.2. Punktoperationen

Auf einer elliptischen Kurve ist die Punktaddition definiert, wenn die Kurve nicht singulär ist. In diesem Fall können aus den Koordinaten zweier Punkte $P = (x_1, y_1)$ und $Q = (x_2, y_2)$ ein dritter Punkt $R = (x_3, y_3)$ berechnet werden. Dazu wird eine Gerade durch die Punkte P und Q gelegt. Diese Gerade schneidet die elliptische Kurve in einem dritten Punkt $-R$, wie es in Bild 1 dargestellt ist. Aus diesem Punkt lässt sich dann der Punkt R bestimmen.

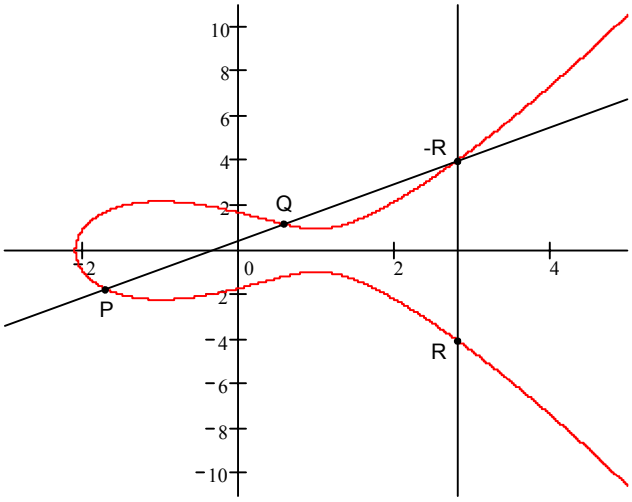

Bild 1: Punktaddition

Die Punktverdopplung ist ein spezieller Fall der Punktaddition, bei dem $Q = P$ ist. Somit ist die Gerade eine Tangente an der elliptischen Kurve im Punkt P. Zur Berechnung der Koordinaten für den Punkt R wird die Steigung λ der Gerade benötigt. Für die elliptische Kurve nach Gleichung (1) hat die Gleichung für die Steigung λ folgende Form:

$$\lambda = \begin{cases} \dfrac{y_1 + y_2}{x_1 + x_2} & P \neq Q \\ x_1 + \dfrac{y_1}{x_1} & P = Q \end{cases} \tag{3}$$

Die Koordinaten für $R = P + Q$ können dann mit folgender Gleichung bestimmt werden:

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$$
$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \tag{4}$$

Für die Punktverdopplung kann die Gleichung (4) auf folgende Form vereinfacht werden:

$$x_3 = \lambda^2 + \lambda + a$$
$$y_3 = x_1^2 + \lambda x_3 + x_3 \tag{5}$$

1.3. Operationen im endlichen Körper $GF(2^m)$

Eine Skalarmultiplikation wird auf die Punktaddition und Punktverdopplung auf der elliptischen Kurve abgebildet. Diese wiederum werden mit Hilfe der Addition, Multiplikation und Division im endlichen Körper berechnet. Eine Division lässt sich im endlichen Körper am besten über die Multiplikation mir dem multiplikativen Inversen berechnen:

$$\frac{a}{b} = a \cdot b^{-1} \tag{6}$$

Des Weiteren lässt sich die Quadrierung als Sonderform der Multiplikation schneller berechnen, als die Multiplikation selbst. Somit werden vier Funktionen im endlichen Körper $GF(2^m)$ benötigt. Das sind die Addition, Multiplikation, Quadrierung und Invertierung, wie sie in Bild 2 dargestellt sind.

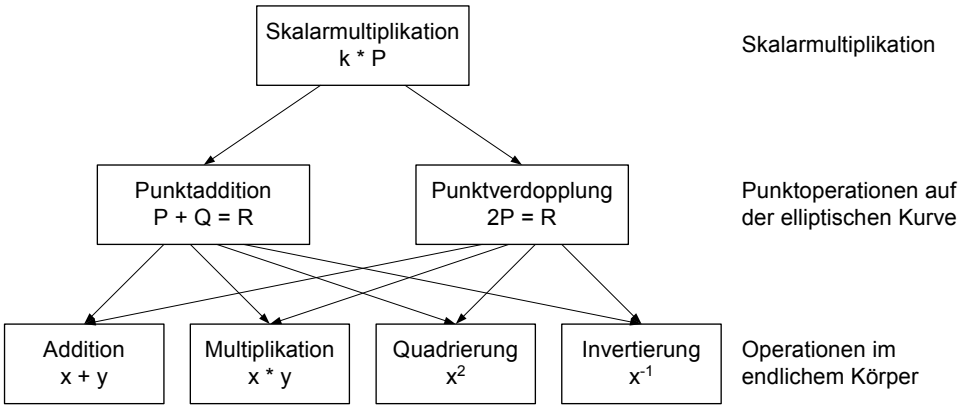

Bild 2: Skalarmultiplikation

2. Aufbau der ALU

Die ALU gliedert sich in drei große Bereiche. Zum ersten Bereich zählen die vier Operationen, welche die Berechnungen im endlichen Körper $GF(2^m)$ auf Polynombasis ermöglichen. Dazu gehören die Addition, die Multiplikation, die Quadrierung und die Invertierung. Der zweite Bereich ist für die Speicherung der Daten verantwortlich. Er enthält eine Registerbank mit 16 Speicherplätzen, in der die Eingangsdaten und die berechneten Werte abgelegt werden können. Der letzte Bereich ist die Steuerlogik für die ALU. Sie besteht aus einem endlichen Zustandsautomaten.

Für die Realisierung der ALU wurde der endliche Körper $GF(2^{167})$ mit dem irreduziblen Polynom $x^{167} + x^6 + x^0$ gewählt.

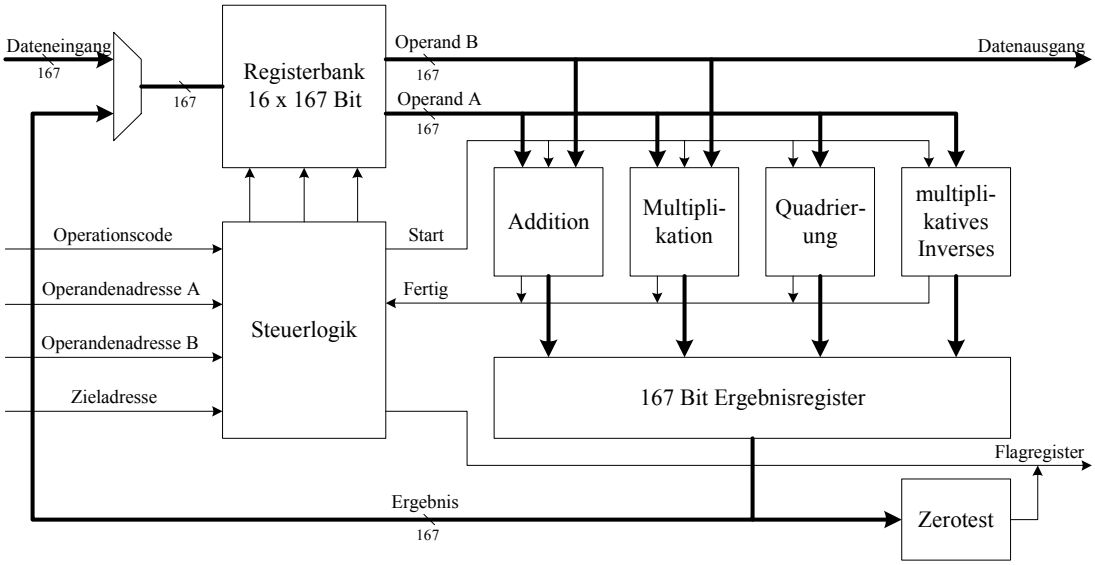

Bild 3: Architektur der ALU

Angesteuert wird die ALU durch ein 15 Bit Steuerwort, welches sich aus einem 3 Bit breiten Operationscode und drei 4 Bit breiten Adressen zusammensetzt. Mit den drei Adressen werden die entsprechenden Speicherplätze aus der Registerbank ausgewählt. Dabei sind zwei Adressen für die Auswahl der Operanden A und B zuständig und die dritte Adresse wählt das entsprechende Zielregister in der Registerbank aus, in die das berechnete Ergebnis geschrieben wird.

Da die Berechnung des multiplikativen Inversen von Null nicht möglich ist, war es notwendig einen Zerotest zu implementieren. Dazu wurde das Ergebnisregister auf Null geprüft. Das Resultat des Tests wurde mit in das Flagregister aufgenommen, was außerdem ein weiteres Bit enthält, welches anzeigt ob die ALU gerade beschäftigt ist oder ob sie sich im Ruhezustand befindet.

2.1. Addition

Eine Besonderheit der Mathematik in endlichen Körpern von $GF(2^m)$ besteht in der Tatsache, dass bei der Addition kein Überlauf auftritt und somit auch nicht an die nächste Stelle übergeben werden muss. Der endliche Körper von $GF(2^m)$ besteht aus einem m Bit breiten Vektor des endlichen Körpers $GF(2)$. In diesem endlichem Körper gibt es nur die Elemente 0 und 1, wenn durch die Addition eine größeres Elemente entsteht, wird dieses modulo 2 reduziert:

$$a + a = 2a \bmod 2 = 0 \tag{7}$$

Somit ist die Addition im Vergleich zur herkömmlichen Addition deutlich schneller und kleiner in Hardware zu realisieren. Sie enthält nur 167 parallel liegende XOR-Verknüpfungen. Und da kein Übertrag durchgereicht werden muss, kann sie in nur einem Takt durchgeführt werden.

2.2. Multiplikation

Die Multiplikation zweier Zahlen X = A * B kann nach folgender Gleichung durchgeführt werden:

$$X = A \cdot B = \sum_{i=0}^{166} A \cdot b_i 2^i \qquad\qquad B = \left(b_{166}, \ldots, b_0\right)_2 \tag{8}$$

Es werden also die einzelnen Partialprodukte $A * b_i 2^i$ berechnet und dann aufaddiert. Wobei die Addition einfache eine XOR-Verknüpfung darstellt, wie im vorigen Abschnitt erläutert. Dieser serielle Algorithmus lässt sich sehr einfach in Hardware umsetzen, wobei die Partialprodukte nach jedem Takt aufsummiert werden. Dazu wird der Multiplikator B in ein Linksschieberegister geladen, es wird also immer das jeweilige MSB (Most Significant Bit) b_{166} ausgegeben. Immer wenn $b_{166} = 1$ ist, wird der Multiplikand A zu dem Wert im Zwischenregister addiert. Wenn $b_{166} = 0$ ist, bleibt der Wert im Zwischenregister unverändert. Das Zwischenregister ist durch ein Linksschieberegister mit einer Breite von 333 Bit realisiert. Darin wird das aktuelle aufsummierte Partialprodukt gespeichert. In jedem Takt wird nun um eine Stelle geschoben, der Multiplikand evtl. aufaddiert und das Ergebnis ins Zwischenregister geschrieben. Dieser Algorithmus wird auch „Shift and Add“ Methode genannt, er kann sowohl von links nach rechts als auch von rechts nach links durchgeführt werden. Damit ist die einzelne Multiplikation beendet. Bei dieser Möglichkeit müsste nun noch die Reduzierung des Ergebnisses mit dem irreduziblen Polynom erfolgen

In dieser ALU wurde die eben vorgestellte Multiplikation modifiziert und mit der Reduzierung zusammengefasst. Das bringt denn Vorteil mit, dass das Zwischenergebnis nicht größer als 168 Bit werden kann und dass für die Reduktion keine zusätzlichen Taktzyklen benötigt werden. Die Koeffizienten des Zwischenergebnisses dürfen nicht größer werden als x^{167}. Da das MSB des Zwischenergebnisregisters genau diesem Koeffizient entspricht, muss immer wenn das MSB = 1 ist eine Reduktion mit dem irreduziblen Polynom durchgeführt werden. Somit lässt sich die serielle Multiplikation in 167 Takten durchführen.

2.3. Quadrierung

Eine weitere Besonderheit in GF(2^m) stellt die Quadrierung da. Denn es gilt:

$$(a+b)^2 = a^2 + 2ab + b^2 = a^2 + 0ab + b^2 = a^2 + b^2 \tag{9}$$

Es wird bei der Quadrierung von Polynomen jedes Element einzeln quadriert, was zur Folge hat, dass nur Elemente mit geradzahligen Potenzen im Ergebnis entstehen können. Die eigentliche Quadrierung lässt sich dabei sehr einfach realisieren. Jeder ungerade Koeffizient ist Null und die geraden Koeffizienten werden einfach, aus dem zu quadrierendem Wert, nach folgendem Schema übernommen.

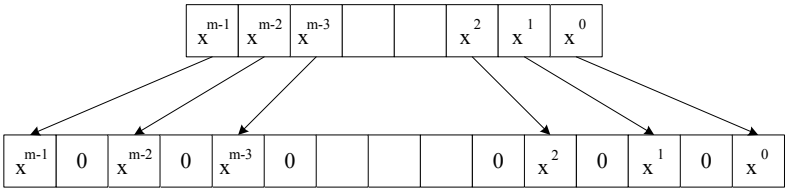

Bild 4: Quadrierung

Somit wird liegt ein Zwischenergebnis mit einer Bitbreite von 333 Bit vor, dieses muss dann noch mit dem irreduziblen Polynom reduziert werden. In unserer Lösung wird die Reduktion seriell durchgeführt. Dabei kann jedes zweite Bit des Zwischenergebnisses übersprungen werden, da dies Null ist. Das funktioniert aber nur bis zu den letzen 6 Bits. Denn durch das irreduzible Polynom kann ab dann jedes Bit eine Eins aufweisen. Somit werden für die Reduktion 86 Takte benötigt. Diese Tatsache geht auf die Arbeit [WU99] zurück und die Anzahl Takte lassen sich durch die Gleichung $(m + k – 1)/2$ bestimmen, wobei m und k durch das irreduzible Polynom $x^m + x^k + x^0$ bestimmt werden.

2.4. Invertierung

Für die Invertierung wurde auf die Arbeit von [BRU93] zurückgegriffen. In dieser Hardwarelösung wird das multiplikative Inverse mit Hilfe des erweiterten Euklidischen Algorithmus berechnet. Der Vorteil dieser Lösung besteht darin, dass sie sich genau so schnell berechnen lässt, wie eine serielle Multiplikation. Außerdem kann dieser

Hardwareimplementierung auch weiter parallelisiert werden, so dass sie auch für eine schnellere ALU geeignet ist. In Bild 5 ist ein schematischer Aufbau der Hardwareinvertierung dargestellt.

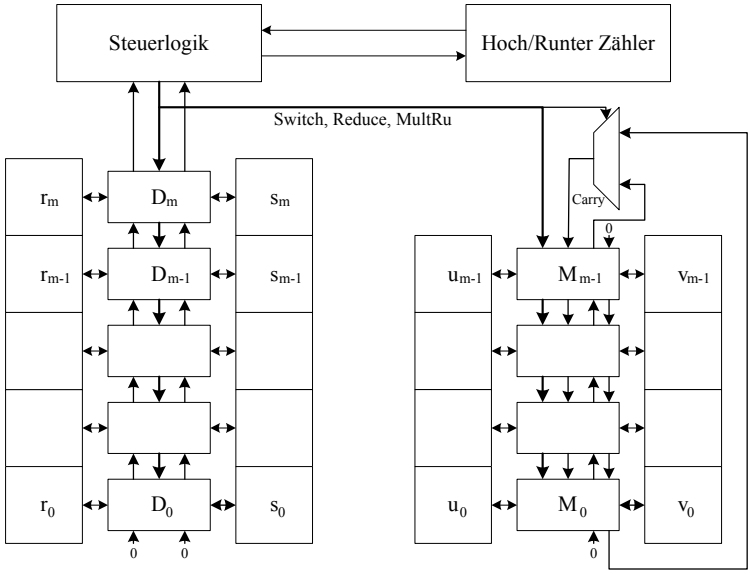

Bild 5: Invertierung

Die Schaltung besteht aus den vier Registern *r*, *s*, *u*, und *v*. Zwischen den *r* und *s* Registern sind identische D-Zellen (Division) und zwischen den *u* und *v* Registern die identischen M-Zellen (Modulo Division) eingefügt worden. Die Steuersignale Switch, Reduce und MultRu werden in der Steuerlogik generiert und dann in die D- und M-Zellen geschaltet. Anfangs wird in das *s* Register das irreduzible Polynom und in das *r* Register der zu invertierende Wert geladen. Das *u* Register wird auf Null und das *v* Register wird auf Eins gesetzt. Nun wird der zu invertierende Wert in *m* Durchläufen mit dem irreduziblen Polynom reduziert, dann enthält das *r* Register nur noch Nullen. Parallel zu diesen Schritten wird im *u* und *v* Register mit den M-Zellen die gleiche Reduktion durchgeführt. Dabei enthält das *u* Register dann nach *m* Schritten das multiplikative Inverse zu dem Anfangswert im *r* Register. Für unseren Fall liegt das multiplikative Inverse somit nach 167 Takten vor.

3. Ergebnisse und Ausblick

Nach der Synthese dieser ALU für eine Xilinx XC400E-8 habe wir folgende Ergebnisse erzielt:

Taktfrequenz	Flipflops	LUT	Gatteräquivalente
51,3 MHz	1419 (14%)	5116 (53%)	63049

Tabelle 1: Ergebnisse

Durch umfangreiche Simulationen wurde die korrekte Funktionsweise der ALU überprüft.

Als nächste Arbeit soll diese ALU zu einem kompletten Prozessor zur Berechnung der Skalarmultiplikation ausgebaut werden. Dazu wird die ALU um eine Schicht zur Berechnung der Punktaddition und Punktverdopplung erweitert. Auf dieser wird wiederum um eine Schicht für die Skalarmultiplikation aufgesetzt. Dieser beiden Schichten sollen durch Mikrocodesteuerung realisiert werden. Somit können auch neuere Algorithmen noch im Nachhinein umgesetzt werden.

4. Zusammenfassung

In diesem Beitrag wurde eine ALU für Berechnungen im endlichen Körper $GF(2^m)$ vorgestellt. Das besondere an dieser ALU stellt die Implementierung des multiplikativen Inversen dar. Denn diese Möglichkeit wurde in andere Implementierungen für die Skalarmultiplikation nicht berücksichtiget. Daher können mit dieser ALU Algorithmen zur Berechnung der Skalarmultiplikation angewendet werde, die bis zu 4-mal schneller sind als Algorithmen, ohne schnelle Invertierung auskommen müssen.

Literatur

[BRU93] Brunner, H.; Curiger, A.; Hofstetter, M.:
On Computing Multiplicative Inverses in $GF(2^m)$
IEEE Transactions on Computation, vol. 42, pp. 1010-1015, 1993

[FIE02] Fiedler, M.-S.:
Entwicklung und Implementierung einer ALU für Operationen im endlichen Körper GF(2m)
Universität Rostock, Bachelorarbeit, 2002

[IEE99] IEEE P1363:
Standard Specifications for Public Key Cryptography
Draft Version 13, 1999

[KOB87] Koblitz, N.:
Elliptic Curve Cryptosystems
Mathematics of Computation, vol. 48, no. 177, pp. 203-209, 1987

[LOP99] Lopez, J.; Dahab, R.:
Fast Multiplication on Elliptic Curves over GF(2m) without Precomputation
Workshop on Cryptographic Hardware and Embedded Systems CHES '99, LNCS 1717, Springer Verlag, 1999

[MEN93] Menezes, A. J.:
Elliptic Curve Public Key Cryptosystems
Kluwer Academic Publishers, 1993

[MIL86] Miller, V.:
Use of elliptic curves in cryptography
Advances in Cryptology CRYPTO '85, LNCS 218, pp. 417 426, Springer Verlag, 1986

[MON87] Montgomery, P.:
Speeding the Pollard and Elliptic Curve Methods of Factorization
Mathematics of Computation, vol. 48, pp. 243-264, 1987

[WU99] Wu, H.:
Low Complexity Bit-Parallel Finite Field Arithmetic Using Polynomial Basis
Workshop on Cryptographic Hardware and Embedded Systems CHES '99, LNCS 1717, Springer Verlag, 1999