Mobile Sicherheit durch effiziente Public-Key-Verschlüsselung

Hagen Ploog *Dirk Timmermann*

Universität Rostock, Institut für Angewandte Mikroelektronik und Datenverarbeitung
Richard-Wagner-Str. 31, 18119 Rostock
Hagen.Ploog@uni-rostock.de

Abstract. Public-Key-Kryptographie, insbesondere das RSA-Verfahren, spielt heute in fast allen wichtigen öffentlichen Transaktionen bei Identifizierung, Authentifizierung und Digitaler Signatur an Bankautomaten, Handys oder auch bei der seit kurzem eingeführten Möglichkeit zur elektronischen Steuererklärung (E.L.S.T.E.R), eine entscheidende Rolle.. Wir zeigen in diesem Beitrag, wie Public-Key-Kryptographie durch algorithmische Optimierungen in effiziente Hardware-Implementierungen überführt werden kann. Erst durch den Einsatz dedizierter Hardware wird eine minimale Verzögerungszeit garantiert und die Benutzung für den Anwender transparent.

Einführung

Verschlüsselungsverfahren haben in den letzen Jahren einen im wesentlichen verdeckten Einzug in unser tägliches Leben gehalten. Wir beschreiben in diesem Beitrag, welche Verfahren es gibt und wie sie eingesetzt werden können. Am Beispiel des RSA-Verfahrens zeigen wir, wie mit Hilfe von mathematischen Optimierungen eine schnelle Hardware-Lösung realisiert werden kann und sich damit mobile Sicherheit effizient verwirklichen lässt.

1. Kryptographische Verfahren

Es gibt drei unterschiedliche Prinzipien von Verschlüsselungsverfahren. Sie werden danach unterschieden, welche und wie die Schlüssel benutzt werden. Alle Algorithmen lassen sich dadurch in eine der folgenden Gruppen einordnen:

- symmetrische Systeme,
- asymmetrische Systeme und
- hybride Systeme

1.1. Symmetrische Systeme

Bei dem Einsatz symmetrischer Systeme wird auf der Sender und auf der Empfänger-Seite der gleiche Schlüssel benutzt. Das eigentliche Problem besteht in der Frage, wie der zu benutzende Schlüssel übertragen werden kann.

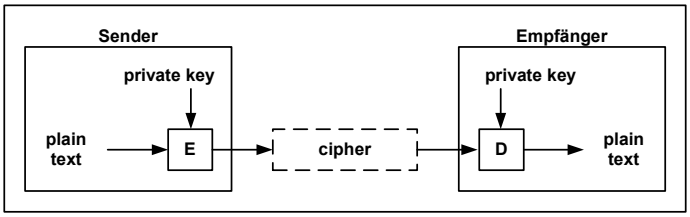

Bild 1: Symmetrische Verschlüsselung

1.2. Asymmetrische Systeme

Viele wichtige Funktionen (Identifikation, Authentifizierung, Unleugbarkeit, digitale Signatur usw.) lassen sich mathematisch nur mit Hilfe der asymmetrischen Systemen verwirklichen. Dabei werden auf beiden Seiten unterschiedliche Schlüssel benutzt. Die beiden Schlüssel sind eindeutig miteinander verknüpft. Je nach Anwendungsfall wird einer der beiden Schlüssel veröffentlicht, der jeweils andere geschützt.

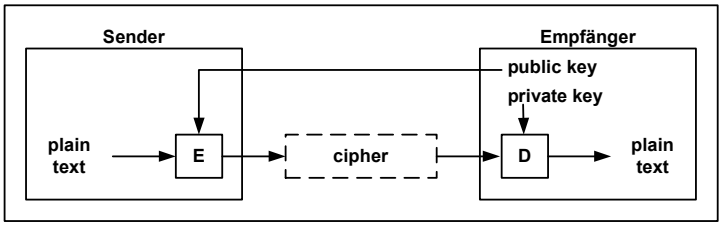

Bild 2 : Asymmetrische Verschlüsselung

1.3. Hybride Systeme

Um die Geschwindigkeitsvorteile der symmetrischen als auch die anwendungsspezifischen Vorteile der asymmetrischen Systeme nutzen zu können, wurden die hybriden Systeme eingeführt. Hier wird der Schlüssel mittels asymmetrischer Verfahren übertragen und anschließend zur Übertragung der Nutzdaten in den symmetrischen Modus umgeschaltet.

2. RSA

Das RSA-Verfahren ist der bekannteste Vertreter der Gruppe der asymmetrischen Systeme. Das Schlüsselset besteht aus drei Zahlen, dem öffentlichen Schlüssel *E*, dem privaten Schlüssel *D* und dem Systemmodul *N*. Dabei ist *N* das Produkt zweier Primzahlen (p, q) und um ausreichende Sicherheit zu gewährleisten ca. 310 Stellen lang (1024 bit). *E* kann zufällig gewählt werden, muss aber relativ prim zu φ(*N*) sein. Dadurch gilt: $\gcd(E,(p-1)\cdot(q-1))=1$. Für die Berechnung von D gilt dem entsprechend: $D = E^{-1} \bmod (p-1)\cdot(q-1)$.

Zum Verschlüsseln einer Nachricht *m*, wird $c = m^E \bmod N$ berechnet. Zum Dekodieren wird der verschlüsselte Text mit $m = c^D \bmod N$ wieder zurückgewandelt. In beiden Fällen wird eine Zahl *x* mit einer Zahl *y* exponentiert und anschließend mit einer Zahl *N* reduziert.

2.1. Modulo-Exponetierung

Die Modulo-Exponentierung kann als eine Folge von nacheinander ausgeführten Modulo-Multiplikationen verstanden werden ("square and multiply"). Dadurch werden die einzelnen Zwischenergebnisse nicht unnötig groß und die zum Speichern dieser Zahl notwendigen Register können in der Größenordnung von $\log_2(N)$ bleiben. In Algorithmus 1 stellt y_i den Wert des Schlüssels an der entsprechenden Bitposition dar.

```
Input:    x, y, N
Output: z = x^y mod N
1.  z = x
2.  For i = n-1 DOWNTO 0 DO
3.     z = z • z  mod N
4.     IF ( y_i == 1) z= z • m mod N
5.  END FOR
6.  RETURN z
```

Algorithmus 1: Square and Multiply

2.2. Modulo-Multiplikation

Die Modulo-Multiplikation $P = AB \bmod N$ kann wiederum auf die Modulo-Addition zurückgeführt werden. Anstatt das gesamte Zwischenprodukt zu berechnen und erst dann modulo N zu reduzieren, wird die Reduktion nach jedem Zwischenschritt durchgeführt, $P_i = 2P_{i-1} + Ab_i \bmod N$. Dadurch verbleibt P_i immer im Bereich von [0..3N[. Für $A, B < N$ und $B=b_{n-1}b_{n-2} \ldots b_0$ gilt:

$$\begin{aligned} AB \bmod N &= \sum_{i=0}^{n-1} Ab_i 2^i \bmod N \\ &= \sum_{i=n-1}^{0} Ab_i 2^i \bmod N \\ &= \left(\left(\left(\ldots\left(\left(Ab_{n-1} \bmod N\right)\cdot 2 + Ab_{n-2}\right)\bmod N\right)\cdot 2 + \ldots\right)\bmod N\right)\cdot 2 + Ab_0\Big) \bmod N \end{aligned}$$

Die Multiplikation Ab_i ist eine binäre Entscheidung, ob A zum Zwischenergebnis P_i addiert werden muss. Bei der Modulo-Addition zweier Zahlen mit $A, B < N$ gilt :

$$A + B \bmod N = \begin{cases} A + B, \; falls \; A + B < N \\ A + B - N, \quad sonst \end{cases}$$

3. Bewertung

Zur Bewertung einer möglichen Implementierung müssen Vergleichswerte geschaffen werden. Einen solchen Vergleichswert kann die zur Berechnung notwendige Taktanzahl und damit die benötigte Zeit darstellen. Zur Vereinfachung nehmen wir deshalb an, dass eine Modulo-Addition in einem Taktzyklus ausgeführt werden kann. Weiterhin nehmen wir an, dass im Mittel die Hälfte aller Bits des Schüssel mit 1 besetzt sind. Damit lässt sich die Modulo-Exponentierung im Mittel mit 1.5n-Modulo-Multiplikationen ausführen, jede dieser Modulo-Multiplikationen benötigt ihrerseits wieder n Modulo-Additionen. Die Latenzzeit dieser Architektur beträgt damit $1.5n^2$ Takte. Um diesen erheblichen Rechenaufwand zu reduzieren werden Verfahren eingesetzt, die die unnötigen Schritte während der Berechnung überspringen. In Tabelle 1 sind die zur Zeit aktuellen Werte angegeben.

Log2(N)	Anzahl Takte
512	393216
1024	1572864

Tabelle 1 : Taktanzahl in Abhängigkeit der Schlüssellänge

4. Optimierung

Der Durchsatz kann erhöht werden, indem die Nullen im Multiplikator B übersprungen werden, denn nur für $b_i = 1$ muss tatsächlich eine Addition von A zum bisherigen Zwischenergebnis erfolgen. Die Anzahl der zu überspringenden Bits sei sp. Das Überspringen der Nullen erfolgt mittels eines k-stufigen Barrelshifters. Die maximale Verschiebeweite beträgt dann 2^k-1. Die Zwischenwerte berechnen sich dann:

$$P_i = \left(2^{sp} P_{i-1} + Ab_i\right) \bmod M$$

Häufig wird der Multiplikator B in eine binary-signed-digit Darstellung D_{SD2} mit $d_i \in$ {-1, 0, 1} umkodiert, so dass die Anzahl der Nicht-Null-Digits und damit die Anzahl der tatsächlich auszuführenden Operationen gesenkt wird. Da d_i jetzt auch den Wert "–1" annehmen kann, muss die Architektur neben der Addition auch die Subtraktion von A ermöglichen. P_i berechnet sich dadurch zu:

$$P_i = \left(2^{sp} P_{i-1} + d_i A\right) \bmod M$$

Es kann gezeigt werden [WU99], dass durch das Umkodieren einer *n*-bit Binärzahl die Anzahl der Nicht-Null-Digits im Mittel *n*/3 beträgt und dadurch die Abarbeitung von durchschnittlich drei Bit pro Operation ermöglicht wird. In realen Systemen wird die mögliche Anzahl der zu verschiebenden Bits u.a. durch die Anzahl der Stufen des Barrelshifters begrenzt, so dass die maximal zu erwartende durchschnittliche Verschiebeweite von drei Bit pro Operation nicht erreicht werden kann. Vom Zwischenergebnis werden noch, entsprechend der Verschiebeweite, Vielfache des Moduls $2^x M$ subtrahiert, so dass sich letztlich ergibt:

$$P_i = 2^{sp} P_{i-1} + d_i A \mp 2^x M$$

Die Addition erfolgt zweistufig. Zuerst wird mit einem Carry-Save-Addierer (CSA) die Summe gebildet, die dann mittels eines Carry-Propagate-Addierers (CPA) von der redundanten Repräsentation wieder in die binäre Darstellung überführt wird, wodurch die Anzahl der Register zum Speichern der einzelnen Werte gesenkt wird. Diese Veröffentlichung konzentriert sich auf die Beschleunigung des Multiplikations-look-ahead ohne auf die Beschleunigung des Reduktions-look-ahead einzugehen, da dieser unabhängig von der Multiplikation arbeitet.

5. SD-Recoding

Die Umformung einer Binärzahl *B* in eine binary-signed-digit Darstellung D_{SD2} mit $d_i \in \{-1, 0, 1\}$ reduziert die Anzahl der Nicht-Null-Werte im Mittel auf *n*/3 und im schlechtesten Fall auf *n*/2, wobei *n* die Anzahl der Bits von *B* darstellt. Die Anzahl der auszuführenden Operationen während der Multiplikation wird durch die Anzahl der Nicht-Null-Digits von D_{SD2} bzw. durch die Länge der '0' und '1'-Ketten in *B* bestimmt. Reitwiesner gibt in [REI60] einen Algorithmus für die Umkodierung einer Binärzahl in eine SD-Zahl von *rechts-nach-links* an. Da die Multiplikation aber von *links-nach-rechts* ausgeführt wird, benutzt Sedlak [SED86a] deshalb für die Umkodierung Tabelle 2, die auf den folgenden Regeln beruht:

1. $\left(\langle 1\rangle^a\right)_2 \mapsto \left(1, \langle 0\rangle^{a-1}, \bar{1}\right)_{SD2} \quad \forall a \geq 2$
2. $\left(\bar{1}, 1\right)_{SD2} = \left(0, \bar{1}\right)_{SD2}$

Für das Umkodieren wird eine zusätzliche Variable *bc* benötigt, die anzeigt, ob gerade ein Block von Einsen oder Nullen übersprungen wird. In Tabelle 2 sind alle möglichen Fälle zusammengefasst. Als Eingabe dient dabei das aktuelle *bc* und die nächsten drei zu untersuchenden Bits von *B*. Als Ausgabe erhält man das entsprechende d_i und den nächsten Wert für *bc*.

Joye und Yen haben in [JOY00] gezeigt, dass die so erzeugte SD_2-Zahl bezüglich des minimalen Hamming-Gewichts von D_{SD2} optimal ist. Für die Umwandlung wird eine führende NULL benötigt. Der Algorithmus startet mit *bc*=0 und b_n=0.

	b_i	b_{i-1}	b_{i-2}	d	bc'	Aktion
bc=0	0	0	X	0	z0	skip
	0	1	0	0	0	
	0	1	1	1	1	Halt + Addition
	1	0	X	1	0	
	1	1	X	X	X	skip
bc=1	1	1	X	0	1	
	1	0	1	0	1	
	1	0	0	-1	0	Halt + Subtraktion
	0	1	X	-1	1	
	0	0	X	X	X	skip

Tabelle 2 : Optimales SD-recoding (links nach rechts)

5.1. Wirkung und Auswirkung

Der Algorithmus verschiebt den Multiplikator *B* nun solange, bis entweder die durch den *k*-stufigen Barrelshifter bedingte maximale Verschiebeweite von 2^k-1 erreicht ist oder eine die Ausführung einer Operation bedingende Eingangskombination der zu untersuchenden Bits vorliegt. Bei einem zweistufigen Barrelshifter wird in ca. 52% aller Fälle eine Operation allein durch den Umkodierungsvorgang ausgelöst. Dieser Wert beträgt bei einem dreistufigem Barrelshifter schon 96%. In dieser Konfiguration wird nur noch in 4% der Fälle die maximale Verschiebeweite erreicht, ohne dass gleichzeitig ein Auslösen durch die Umkodierung erfolgt. Für diesen Fall muss *d* auf NULL und *sp* auf 2^k-1 gesetzt werden können.

Bild 3 zeigt die relative Häufigkeit eines ausschließlich durch das Umkodieren bedingten Halts an einer bestimmten Bitposition *i* für 10000 modularen Multiplikationen zweier 4096 Bit langer Zufallszahlen unter Benutzung eines drei- bzw. vier-stufigen Barrelshifters.

Die maximal mögliche Verschiebeweite wird dabei in ca. 95% der Fälle nicht genutzt (Halt vor Bitposition 2^k-1). Die einfache Erweiterung der Stufenanzahl des Barrelshifters führt also nicht zur durchschnittlich schnelleren Modulo-Multiplikation. Die beiden dargestellten Kurven unterscheiden sich geringfügig, was darin begründet liegt, dass '1'- und '0'-Ketten, deren Länge größer als 2^k-1 ist, in mehreren Schritten abgearbeitet werden müssen. Es gilt:

$$l_{l>2^k-1} = u\left\lfloor \frac{l}{2^k-1} \right\rfloor + l \bmod \left(2^k-1\right)$$

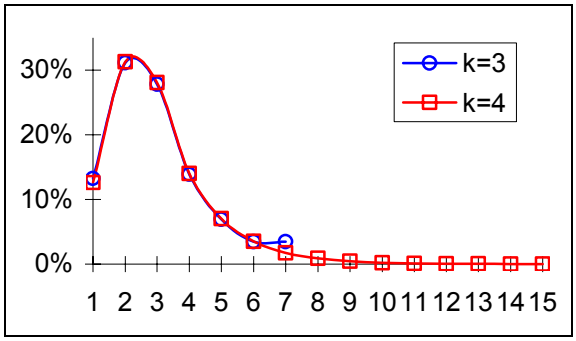

Bild 3 : Abbruchwahrscheinlichkeit an Bitposition *i*

6. Beschleunigung

Um die Modulo-Multiplikation zu beschleunigen, wird zuerst der Fall analysiert, der die Leistung der Modulo-Muliplikation begrenzt. Enthält ein Eingabewert jeweils abwechselnd '0' und '1', also *B* = ..101010101... , dann wird in Abhängigkeit von *bc* entweder

$$D_{SD} = \ldots 101010101\ldots \quad (bc = 0) \text{ oder}$$
$$D_{SD} = \ldots 0\bar{1}0\bar{1}0\bar{1}0\bar{1}0\ldots \quad (bc = 1)$$

erzeugt. Dadurch wird die Verschiebeweite auf lediglich zwei Bits pro Operation reduziert. Das ist tatsächlich der ungünstigste Fall für die SD-Kodierung. Es erscheint daher sinnvoll, jeweils zwei dieser Nicht-Null-Werte in geeigneter Weise zusammenzufassen. Sobald durch den Multiplikations-look-ahead eine Aktion ausgelöst werden soll, wird deshalb nicht nur das aktuelle Digit, sondern auch das nächste und übernächste Digit von D_{SD2} bestimmt. Dabei muss eine Verringerung der Verschiebeweite auch weiterhin immer möglich sein (Abbruch an Bitposition 2^k-1, bzw. 2^k-2). Die modifizierte SD-Umwandlung generiert deshalb nicht nur die nächsten drei Digits von D_{SD}, sondern auch die nächsten drei möglichen Werte für *bc*.

Tabelle 3 fasst alle möglichen Werte für $bc_{i+1}=0$ zusammen. Für den Fall, dass $bc_{i+1}=1$ ist, sieht die Tabelle identisch aus, mit dem Unterschied, dass alle Werte für *b* invertiert und für *d* negiert werden.

Durch diese Erweiterung muss die Architektur nun allerdings in der Lage sein, die Werte ±2*A*, ±3*A*, ±4*A*, ±5*A* und ±6*A* zu P_i zu addieren. Falls die erweiterte Verschiebung nur teilweise bzw. nicht ganz ausgeführt werden kann, müssen weiterhin auch die Werte ±*A* und ±2*A* addiert werden können.

b_i	b_{i-1}	b_{i-2}	b_{i-3}	b_{i-4}	$d_i \overset{bc_i'}{\rightarrow} d_{i-1} \overset{bc_{i-1}'}{\rightarrow} d_{i-2} \overset{bc_{i-2}'}{\rightarrow}$
0	0	0	0	X	$0 \overset{0}{\rightarrow} 0 \overset{0}{\rightarrow} 0 \overset{0}{\rightarrow}$
0	0	0	1	0	$0 \overset{0}{\rightarrow} 0 \overset{0}{\rightarrow} 0 \overset{0}{\rightarrow}$
0	0	0	1	1	$0 \overset{0}{\rightarrow} 0 \overset{0}{\rightarrow} 1 \overset{1}{\rightarrow}$
0	0	1	0	X	$0 \overset{0}{\rightarrow} 0 \overset{0}{\rightarrow} 1 \overset{0}{\rightarrow}$
0	0	1	1	X	$0 \overset{0}{\rightarrow} 1 \overset{1}{\rightarrow} 0 \overset{1}{\rightarrow}$
0	1	0	0	X	$0 \overset{0}{\rightarrow} 1 \overset{0}{\rightarrow} 0 \overset{0}{\rightarrow}$
0	1	0	1	0	$0 \overset{0}{\rightarrow} 1 \overset{0}{\rightarrow} 0 \overset{0}{\rightarrow}$
0	1	0	1	1	$0 \overset{0}{\rightarrow} 1 \overset{0}{\rightarrow} 1 \overset{1}{\rightarrow}$
0	1	1	0	0	$1 \overset{1}{\rightarrow} 0 \overset{1}{\rightarrow} \overline{1} \overset{0}{\rightarrow}$
0	1	1	0	1	$1 \overset{1}{\rightarrow} 0 \overset{1}{\rightarrow} 0 \overset{1}{\rightarrow}$
0	1	1	1	X	$1 \overset{1}{\rightarrow} 0 \overset{1}{\rightarrow} 0 \overset{1}{\rightarrow}$
1	0	0	0	X	$1 \overset{0}{\rightarrow} 0 \overset{0}{\rightarrow} 0 \overset{0}{\rightarrow}$
1	0	0	1	0	$1 \overset{0}{\rightarrow} 0 \overset{0}{\rightarrow} 0 \overset{0}{\rightarrow}$
1	0	0	1	1	$1 \overset{0}{\rightarrow} 0 \overset{0}{\rightarrow} 1 \overset{1}{\rightarrow}$
1	0	1	0	X	$1 \overset{0}{\rightarrow} 0 \overset{0}{\rightarrow} 1 \overset{0}{\rightarrow}$
1	0	1	1	X	$1 \overset{0}{\rightarrow} 1 \overset{1}{\rightarrow} 0 \overset{1}{\rightarrow}$
1	1	X	X	X	(vorher Abbruch)

Tabelle 3 : Kodierung für drei Bits ($bc_{i+1}=0$)

6.1. Berechnung der Beschleunigung

D_{SD} enthält eine bestimmte Anzahl von Nicht-Null-Werten, die jeweils den Multiplikations-look-ahead zu einem HALT und dadurch die Ausführung einer Operation veranlassen. Im folgenden werden wir nun die Anzahl der durch die Modifikation am Multiplikations-look-ahead eingesparten Operationen berechnen. Bild 4 zeigt einen Moore-Automaten (FSM), der die Umwandlung von einer Binärdarstellung in eine SD-Darstellung ausführt. Wegen der Symmetrie kann diese FSM entlang der gestrichelten Linie in zwei bezüglich der Übergangswahrscheinlichkeit äquivalente Hälften geteilt werden. Man erkennt dann, dass z.B. S_1 äquivalent zu ($\neg S_7$) ist. Zur Vereinfachung benutzen wir diese Symmetrie im weiteren und setzen: $S_1 = S_7$, $S_2 = S_8$, $S_3 = S_9$, $S_4 = S_{10}$, $S_5 = S_{11}$ und $S_6 = S_{12}$.

Befindet sich die FSM nun in irgendeinem der HALT-Zustände (S_1, S_2, S_3 (S_7, S_8, S_9)), dann folgt mit der Wahrscheinlichkeit $P(add|S_x)$ in einer der nächsten beiden Stellen ein weiterer HALT:

$$
\begin{aligned}
P(add \mid S_1) &= P(S_1 \mid 1) + P(S_1 \mid 01) + P(S_1 \mid 00) \\
&= 0.5 + 0.25 + 0.25 \\
&= 1 \\
P(add \mid S_2) &= P(S_2 \mid 00) = 0.25 \\
P(add \mid S_3) &= P(S_3 \mid 11) = 0.25
\end{aligned}
$$

Als nächstes berechnen wir nun die Wahrscheinlichkeit P_{STOP}, dass die FSM, von einem beliebigen Zustand startend, in einem bestimmten Zustand S_x anhält, denn dieser HALT-Zustand ist der nächste Zustand von dem aus die FSM startet. Dazu müssen wir den Abstand zwischen zwei HALT-Zuständen berechnen.

Zuerst wird von S_3 aus gestartet. Mit der Eingangsfolge "11", oder "011" bzw. "(0)^{a}11" halten wir in S_2.

$$
\begin{aligned}
P(S_3 \mapsto S_2) &= \tfrac{1}{2}\cdot\tfrac{1}{2} + \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + \ldots \\
&= \tfrac{1}{4}\sum_{i=0}^{\infty} 2^{-i} = \tfrac{1}{4}\cdot\frac{1}{1-\frac{1}{2}} = \tfrac{1}{2}
\end{aligned}
$$

Die Wahrscheinlichkeit, dass bei einem Start von S_3 aus der nächste Halt in S_2 liegt, beträgt demnach 50%. Formell erhalten wir folgende Übergangswahrscheinlichkeiten:

$$
\begin{aligned}
P(S_1 \mapsto S_2) &= \tfrac{1}{2} \\
P(S_1 \mapsto S_1) &= \tfrac{1}{2}\cdot\tfrac{1}{2} = \tfrac{1}{4} \\
P(S_1 \mapsto S_3) &= \tfrac{1}{2}\cdot\tfrac{1}{2} = \tfrac{1}{4} \\
P(S_2 \mapsto S_2[S_8]) &= \tfrac{1}{2}\cdot\tfrac{1}{2} + \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + \ldots \\
&= \tfrac{1}{4}\sum_{i=0}^{\infty} 2^{-i} = \tfrac{1}{4}\frac{1}{1-\frac{1}{2}} = \tfrac{1}{2} \\
P(S_2 \mapsto S_1[S_7]) &= \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + \ldots \\
&= \tfrac{1}{8}\sum_{i=0}^{\infty} 2^{-i} = \tfrac{1}{8}\frac{1}{1-\frac{1}{2}} = \tfrac{1}{4} \\
P(S_2 \mapsto S_3[S_9]) &= \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + \ldots \\
&= \tfrac{1}{8}\sum_{i=0}^{\infty} 2^{-i} = \tfrac{1}{8}\frac{1}{1-\frac{1}{2}} = \tfrac{1}{4} \\
P(S_3 \mapsto S_2) &= \tfrac{1}{2}\cdot\tfrac{1}{2} + \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + \ldots \\
&= \tfrac{1}{4}\sum_{i=0}^{\infty} 2^{-i} = \tfrac{1}{4}\frac{1}{1-\frac{1}{2}} = \tfrac{1}{2} \\
P(S_3 \mapsto S_1) &= \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + \ldots \\
&= \tfrac{1}{8}\sum_{i=0}^{\infty} 2^{-i} = \tfrac{1}{8}\frac{1}{1-\frac{1}{2}} = \tfrac{1}{4} \\
P(S_3 \mapsto S_3) &= \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + \ldots \\
&= \tfrac{1}{8}\sum_{i=0}^{\infty} 2^{-i} = \tfrac{1}{8}\frac{1}{1-\frac{1}{2}} = \tfrac{1}{4}
\end{aligned}
$$

Geht man von einem beliebigen Startpunkt aus, so hält die FSM mit $P_{STOP} = 0.5$ in $S_2(/S_8)$, mit $P_{STOP} = 0.25$ in $S_3(/S_9)$ und mit $P_{STOP} = 0.25$ in $S_1(/S_7)$. Damit kann jetzt die durchschnittliche Wahrscheinlichkeit P_{H2} dafür angegeben werden, dass in einem der beiden nächsten Schritte ein weiterer Halt erfolgt:

$$\begin{aligned} P_{H2} &= P(HALT \mid S_1)\cdot P(S_1 \mid add) \\ &\quad + P(HALT \mid S_2)\cdot P(S_2 \mid add) \\ &\quad + P(HALT \mid S_3)\cdot P(S_3 \mid add) \\ &= \tfrac{1}{4}\cdot 1 + \tfrac{1}{2}\cdot\tfrac{1}{4} + \tfrac{1}{4}\cdot\tfrac{1}{4} \\ &= \tfrac{7}{16} \end{aligned}$$

Die hier beschriebene Modifikation des SD-Recoding beschleunigt die Modulo-Multiplikation *nicht*, falls keine zusätzliche Addition in den nächsten beiden Bits auszuführen ist. Ist aber eine Addition auszuführen, so wird aufgrund der zwei gleichzeitig ausgeführten Additionen eine Beschleunigung von 100% erreicht. Die durchschnittliche Geschwindigkeit ergibt sich also zu:

$$(1 - P_{H2})\cdot 3\,\tfrac{bit}{operation} + P_{H2}\cdot 2\cdot 3\,\tfrac{bit}{operation} = 4.3\,\tfrac{bit}{operation}$$

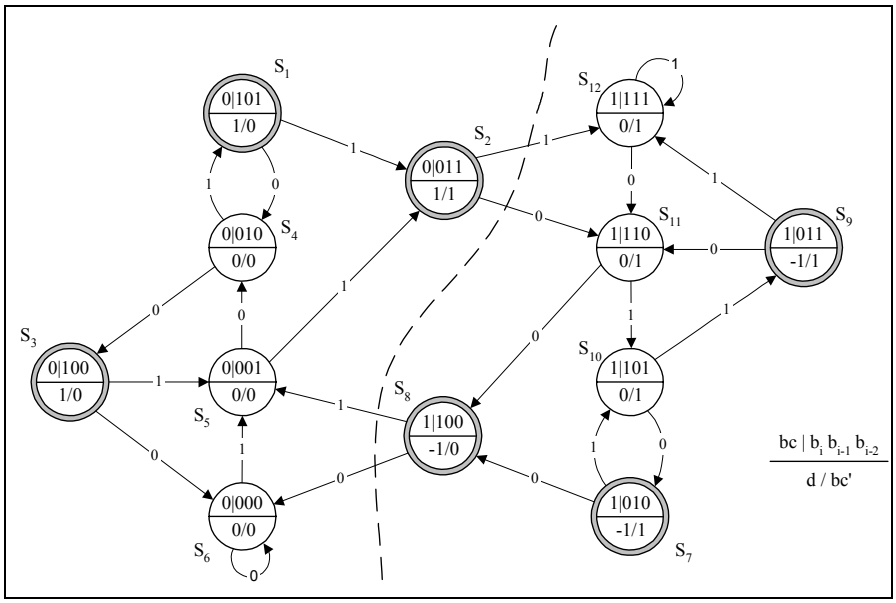

Bild 4 : Zustandsmaschine zum Überführen einer Binärzahl in eine SD-Zahl

7. Implementierung

Statt durch Multiplikation kann das entsprechende Vielfache von *A* über einen zusätzlichen, binär gewichteten Eingang am CS-Addierer gebildet werden. Durch eine entsprechende Aufteilung liegen am höherwertigem Eingang des Addierers die Werte 0, ±2*A* und ±4*A* an. Am niederwertigen Eingang liegen die Werte 0, ±*A* und ±2*A* an. Durch die vorgeschlagene Änderung erhöht sich die Laufzeit des eingesetzten CS-Addieres um eine Volladdiererlaufzeit T_{ADD}. Der Wert *A* wird über einen zusätzlichen Multiplexer an die entsprechenden Eingänge des Addieres geführt. Der Multiplexer kann z.B. als Tri-State-Bus oder als Transmission-Gate implementiert werden. Die gesamte zusätzliche Latenzzeit der Addiereinheit beträgt dadurch lediglich $T_{ADD}+T_{MUX}$. Bild 5 zeigt den Blockaufbau einer Implementierung der hier vorgestellten Modifikation am SD-Recoding. Die dazugekommenen Elemente sind grau hinterlegt.

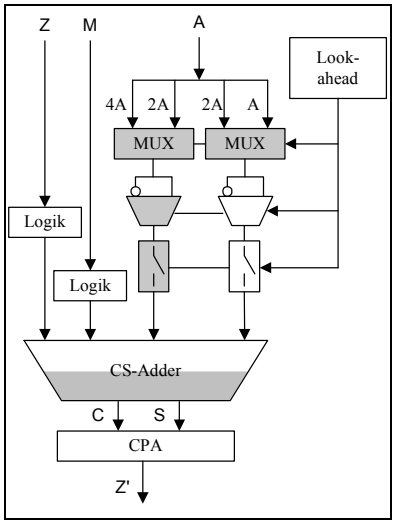

Bild 5 : Erweiterung an der Addierer-Einheit

8. Zusammenfassung

Die Modulo-Multiplikation kann bezüglich der Anzahl der notwendigen Operationen effizient mit einer seriellen Architektur realisiert werden. Theoretische Überlegungen zeigen, dass der obere Grenzwert bisher bei durchschnittlich maximal drei Bit pro Operation lag. Durch eine einfache Modifikation am Kodierungsverfahren kann der Grenzwert nun auf 4.3 Bit pro Operation verschoben werden. Dadurch wird im Mittel eine Leistungssteigerung von bis zu 43% erreicht. Die Auswirkungen der dafür zusätzlich erforderlichen Hardware sind eher gering, da alle wesentlichen Komponenten bereits vorhanden sind. Die Latenzzeit für die modifizierte Architektur berechnet sich zu $T_{LZ} = 1.5 \cdot n \cdot \frac{n}{4.3}$. Für n=1024 ergibt sich damit eine Taktanzahl von 365782 Takten. Vergleicht man die erreichte Geschwindigkeit nun mit der eingangs vorgeschlagenen Lösung, erzielt man eine Verbesserung um den Faktor 4.3.

Literatur

[SED86] Sedlak, H.:
Ein Public-Key-Code Kryptographie-Prozessor
2. E.I.S-Workshop, GMD, Bonn, 1986

[WU99] Wu, H.; Hasan, M.A.:
Closed-Form Expression for the Average Weight of Signed Digit Representations
IEEE Transactions on Computers, Vol. 48, No. 8, August 1999

[Rei60] Reitwiesner, G.W.:
Binary Arithmetic
Advances in Computers, Vol.1, S.231-308, 1960

[SED86a] Sedlak, H.:
Patentschrift DE 3631992 C2, 1986

[Joy00] Joye, M.; Yen, S.M.:
Optimal Left-to-Right binary Signed-Digit Recoding
IEEE Transactions on Computers, Vol. 49, No. 7, July 2000