Flächenreduzierte CORDIC-Architekturen

Steffen Dolling, Andreas Wassatsch, Dirk Timmermann

Universität Rostock

Fachbereich Elektrotechnik und Informationstechnik

Institut für Angewandte Mikroelektronik und Datentechnik

R.-Wagner-Str. 31, D-18119 Rostock

Tel.: 0381 498 3529, Fax.: 0381 498 3601

e-mail: dol@baltic.e-technik.uni-rostock.de

1 Motivation

Die hinsichtlich der Datendurchsatzrate optimale CORDIC-Implementierung bildet eine Pipelinestruktur mit pfad- und bitparalleler Arithmetik. Allerdings ist diese Lösung mit einer hohen Hardwarekomplexität verbunden. Das Ziel dieser Arbeit besteht entsprechend in der Entwicklung von CORDIC-Architekturen mit einer möglichst stark reduzierten Chipfläche ohne Geschwindigkeitsverlust.

2 Redundante CORDIC-Strukturen

Die Iterationsgleichungen für den CORDIC-Algorithmus lauten entsprechend /1/:

$$x_{i+1} = x_i - m\sigma_i 2^{-S(m,i)} y_i \qquad (1)$$

$$y_{i+1} = y_i + \sigma_i 2^{-S(m,i)} x_i \qquad (2)$$

$$z_{i+1} = z_i - \sigma_i \alpha_{m,i} \qquad i = 0,1,...,N-1 \qquad (3)$$

Dabei bezeichnet m das Koordinatensystem, $S(m,i)$ die Shiftfolge, $\alpha_{m,i}$ den Drehwinkel, σ_i die Drehrichtung, n die Genauigkeit und N die Anzahl der Iterationen.
Die Realisierung eines CORDIC-Arrays unter Verwendung einer redundanten Zahlendarstellung auf der Basis von Carry-Save-Addierern /2/ oder Signed-Digit-Addierern /3,4/ bietet den entscheidenden Vorteil einer sehr geringen und von der Wortlänge unabhängigen Rechenzeit pro Addiererstufe. Allerdings resultieren daraus auch verschiedene Nachteile, wie z.B. ein erhöhter Aufwand in Form mehrerer Bits zu Darstellung eines Digits oder eine im Gegensatz zur nichtredundanten Darstellung komplexere Bestimmung des Vorzeichens und damit von σ_i , welches durch das höchstwertige von Null verschiedene Digit festgelegt wird. Im worst-case-Fall müssen sämtliche Digits des Wortes vom MSD (Most-Significant Digit) bis zum LSD (Least-Significant Digit) untersucht werden. Eine praktikable Möglichkeit ist die Abschätzung des Vorzeichens aus einigen MSD's des entsprechenden Wortes. Ein dabei auftretender Fehler, der zu einer Verletzung der Konvergenzbedingung führen würde, kann durch Iterationswiederholungen kompensiert werden /2,3/. Die Iterationswiederholungen sind dabei abhängig von der Betriebsart, dem Koordinatensystem und der Anzahl t der untersuchten MSD-Stellen von z_i (Rotation) bzw. y_i (Vectoring). Eine weitere Möglichkeit ist die in /5/ vorgeschlagene Umformulierung der CORDIC-Iterationsgleichungen und die Berechnung von Absolutwerten, wodurch eine exakte σ_i – Bestimmung erreicht wird. Daraus resultiert eine reguläre VLSI-Struktur und die Vermeidung von Iterationswiederholungen, was aber eine nahezu verdoppelte Anzahl von Registern nach sich zieht.

3 Bisherige Ansätze zur Chipflächenreduktion

Ein Ansatzpunkt zur Hardwareminimierung ist die Art der Kompensation des Skalierungsfaktors k_m, die z.B. durch die Integration der Skalierung in den Algorithmus und die Optimierung der Skalierungs-Iterationen vorgenommen werden kann /6/.
Die angesprochene Problematik der Bestimmung der Drehrichtungen σ_i und eine damit verbundene Reduzierung der notwendigen Iterationswiederholungen ist der Gegenstand weiterer Arbeiten. In /3/ wird gezeigt, daß für Iterationen mit dem Index $i > n/2$ der Skalierungsfaktor keinen Einfluß mehr auf die Länge des zu drehenden Vektors hat und entsprechend $\sigma_i = 0$ zugelassen werden kann. In /4/ wird gezeigt, daß für $i > n/4$ für den Fall $\sigma_i = 0$ durch modifizierte Iterationsgleichungen $x_{i+1} = x_i \left(1 + m2^{-2i-1}\right)$ bzw. $y_{i+1} = y_i \left(1 + m2^{-2i-1}\right)$ Iterationswiederholungen vermieden werden können. Ebenfalls in /4/ wird eine Methode zur parallelen Vorausberechnung der drehrichtungsbestimmenden σ_i in der Betriebsart Rotation vorgestellt, was zu einer Einsparung von 2/3 Iterationsstufen im *Z*-Pfad führt. Dieses Verfahren kann entsprechend /7/ auf den Vectoring-Modus adaptiert werden. Die Anwendung der genannten Methode ist mit einem weiteren Vorteil verbunden. So lassen sich durch eine Umkodierung der σ_i (z.B. Booth-Recoding) zwei Iterationen im *X*- bzw. *Y*-Pfad in eine Stufe integrieren, wobei die Auswahl der auszuführenden Iteration über zwei unterschiedliche Schiebeoperationen erfolgt /4/. Dieses Konzept wurde in /2/ für eine gemischte Basis-2-4-Architektur auf die Betriebsart Vectoring erweitert.
In /7/ werden Architekturen gezeigt, die auf der Basis einer nichtredundanten Zahlendarstellung zu einer Minimierung der CORDIC-Datenpfade und damit der benötigten Chipfläche führen. Diese Ausführungen werden in /8/ auf redundante Systeme übertragen und durch weitere Reduktionsmaßnahmen, die sich auf die kombinatorische Fläche beschränken, ergänzt. Zur Realisierung der Digitadditionen in den einzelnen *X*- bzw. *Y*-Pfaden kann ein mit 42 Transistoren günstig zu implementierender 4-2-Redundant-Redundant (RR) Addierer /9/ eingesetzt werden. Dabei macht eine Betrachtung der Gleichungen (1-3) deutlich, daß mit jeder Iteration, mit Ausnahme der Wiederholungen, die Anzahl der Nullen in den MSD-Positionen, die zu den korrespondierenden Digits x_i bzw. y_i addiert werden, ansteigt. Entsprechend wurde in /8/ eine Struktur angegeben, die an den genannten Positionen eine vereinfachte 3-2-Redundant-Zero (RZ) Addiererzelle (Bild 1) verwendet. Mit diesem Ansatz beläuft sich die Hardwareersparnis auf die ca. um den Faktor 2 kleineren 3-2 Zellen an den entsprechenden Positionen.

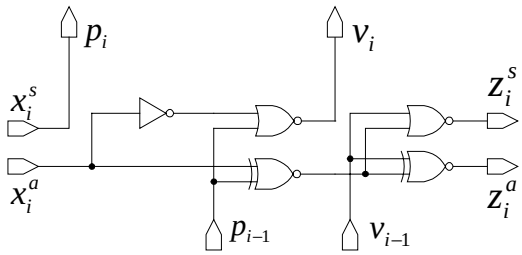

Bild 1 3-2 Redundant-Zero Adderzelle

4 Flächenreduktion

4.1 Eigenschaften der binären redundanten Addition

Ausgangspunkt der Betrachtung ist eine Digitvektoraddition $S = A + B$ mit den Vektorelementen $s_i, a_i, b_i \ (i = 0,1,...,n-1)$, wobei $s_i, a_i \in \{\bar{1}, 0, 1\}$ und $b_i = 0$ gelten soll. Vektor S wird bei dieser Operation implizit umkodiert, so daß für drei benachbarte Elemente $\{..., s_{i+2}, s_{i+1}, s_i, ...\} \neq \{..., 1, 1, 1, ...\}$ bzw. $\{..., s_{i+2}, s_{i+1}, s_i, ...\} \neq \{..., \bar{1}, \bar{1}, \bar{1}, ...\}$ gilt. Oder anders ausgedrückt: maximal zwei benachbarte Elemente s_i können 1 oder $\bar{1}$ sein. Diese Tatsache gilt auch, wenn in die Addition ein Carry-Digit $c = 1$ oder $c = \bar{1}$ in die niederwertigste Digitvektorposition einbezogen wird.

Betrachten wir Vektor $A=\{a_3,a_2,a_1,a_0\}$ als Ergebnis einer oben angegebenen Addition. Da $a_2\,a_1\,a_0\neq 111$ bzw. $a_2\,a_1\,a_0\neq\bar{1}\bar{1}\bar{1}$ gilt, hat dieser einen Wertebereich $-13\le A\le 13$. Eine Addition von A und $B=\{0,0,0,0\}$ sowie einem Carry $c\in\{\bar{1},0,1\}$ in der niederwertigsten Position ergibt entsprechend der Additionsvorschrift in /9/ einen Vektor $S=\{s_4,s_3,s_2,s_1,s_0\}$ mit einem Wertebereich $-14\le S\le 14$. Ein $s_4\neq 0$ kann nur in folgenden Fällen auftreten:

$$s_4=\bar{1}\qquad s_4\,s_3=\bar{1}1$$

$$s_4=1\ \begin{cases} s_4\,s_3=1\bar{1} \\ s_4\,s_3\,s_2=10\bar{1}\ \ und\ \ s_1\,s_0\neq 11 \end{cases}$$

Diese stellen somit Pseudoüberläufe dar, die sich durch $\bar{1}1\rightarrow 0\bar{1}$, $1\bar{1}\rightarrow 01$ und $10\bar{1}\rightarrow 011$ umkodieren lassen. Damit folgt $s_4=0$ und $s_2\,s_1\,s_0\neq 111$ bzw. $s_2\,s_1\,s_0\neq\bar{1}\bar{1}\bar{1}$.

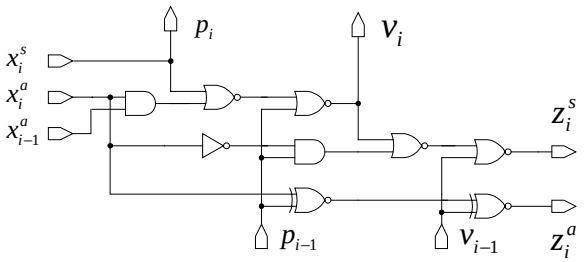

Bild 2 Redundant-Zero-0-Zelle

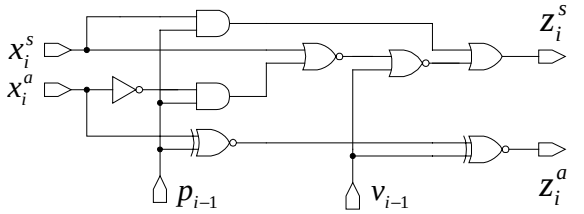

Bild 3 Carry-Absorber-Zelle

Die Addition $S=A+B+c$ kann mit einem 4-Digit-Addierer realisiert werden, der zur Verknüpfung der Digits mit den Indizes 1 und 0 und des Carry c in der niederwertigsten Position RZ-Addierer entsprechend Bild 1 verwendet. Die Addition der MSD's und die Vermeidung von Pseudoüberläufen wird mit einer Redundant-Zero-0-Zelle (RZ0) (Bild 2) und einer Carry-Absorber-Zelle (CAB) (Bild 3) realisiert.

Betrachten wir eine Pipelinestruktur zur Realisierung der Iteration $A_{i+1}=A_i+2^{-(i+4)}B_i$, $i=(0,1,...,N-1)$. A_i und B_i sind dabei Digitvektoren mit den Vektorelementen $a_{i,j},b_{i,j}\in\{\bar{1},0,1\}$, $(j=0,1,...,n-1)$. Für die höchstwertigsten Digits des Eingangsvektors A_0 gilt dabei $a_{0,n-1}\,a_{0,n-2}\,a_{0,n-3}\neq 111$ bzw. $a_{0,n-1}\,a_{0,n-2}\,a_{0,n-3}\neq\bar{1}\bar{1}\bar{1}$. Für die Addition der Digits $a_{i,n-(i+1)}\,a_{i,n-(i+2)}\,a_{i,n-(i+3)}\,a_{i,n-(i+4)}$ werden in den einzelnen Stufen die oben beschriebenen 4-Digit-Addierer und für die weiteren niederwertigen Stellen 4-2-RR-Zellen eingesetzt. So ergibt sich ein Schema gemäß Bild 4, indem neben *A* und *B* auch der Ausgangsvektor *S* des 4-Digit-Addierers dargestellt ist.

	$a_{0,n-1}$	$a_{0,n-2}$	$a_{0,n-3}$	$a_{0,n-4}$	$a_{0,n-5}$	$a_{0,n-6}$
+	0	0	0	0	$b_{0,n-1}$	$b_{0,n-2}$
$s_{0,4}$	$s_{0,3}$	$s_{0,2}$	$s_{0,1}$	$s_{0,0}$		
	$\boldsymbol{a_{1,n-1}}$	$a_{1,n-2}$	$a_{1,n-3}$	$a_{1,n-4}$	$a_{1,n-5}$	$a_{1,n-6}$
+		0	0	0	0	$b_{1,n-1}$
	$s_{1,4}$	$s_{1,3}$	$s_{1,2}$	$s_{1,1}$	$s_{1,0}$	
	$\boldsymbol{a_{2,n-1}}$	$\boldsymbol{a_{2,n-2}}$	$a_{2,n-3}$	$a_{2,n-4}$	$a_{2,n-5}$	$a_{2,n-6}$
+			0	0	0	0
		$s_{2,4}$	$s_{2,3}$	$s_{2,2}$	$s_{2,1}$	$s_{2,0}$
	$\boldsymbol{a_{3,n-1}}$	$\boldsymbol{a_{3,n-2}}$	$\boldsymbol{a_{3,n-3}}$	$a_{3,n-4}$	$a_{3,n-5}$	$a_{3,n-6}$

Bild 4 Additions-Schema

Betrachten wir die Iteration mit dem Index 1, so wird deutlich, daß das Element $a_{1,n-1}$ nicht durch einen Übertrag aus den niederwertigen Stellen verändert werden kann, da $s_{1,4}=0$ gilt. Daraus resultiert für alle Iterationsstufen mit dem Index $i\ge 1$ $a_{i,n-1}=a_{1,n-1}$ und entsprechend können die Addierer in den Positionen $a_{i,n-1}$ entfallen. Mit steigendem Iterationsindex werden somit die Digits von A vom MSD her beginnend konstant, was einer wachsenden Einsparung von Addiererzellen entspricht.

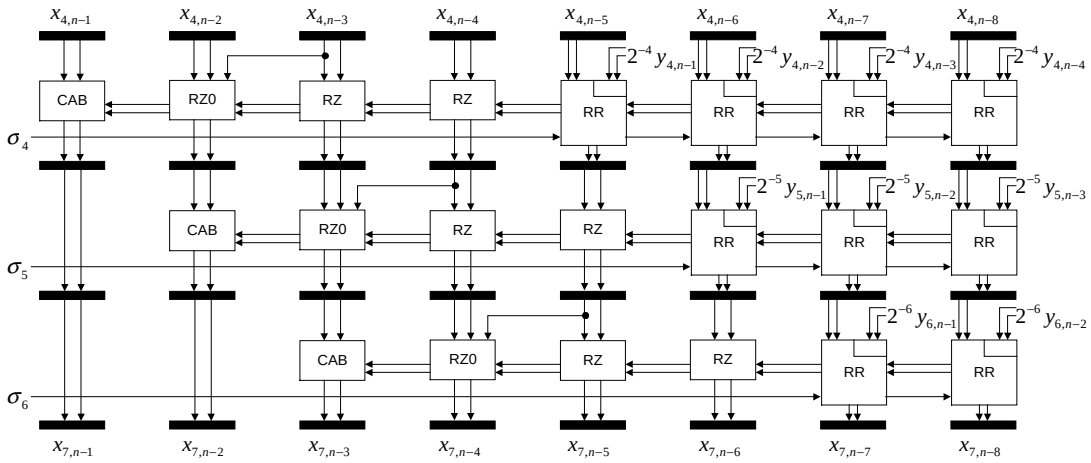

Bild 5 Reduzierter *X*-Datenpfad mit speziellen Addiererzellen

4.2 Betriebsart Rotation

Die genannte Architektur läßt sich in den *X*- und *Y*-Datenpfad integrieren. Bild 5 zeigt den prinzipiellen Aufbau von drei aufeinanderfolgenden Iterationsstufen. Daran wird deutlich , daß in den MSD-Positionen des *X*- bzw. *Y*-Pfades zunehmend die Addiererzellen entfallen können. Die Latenzzeit τ_I der so entstandenen Addiererreihe ist gleich der einer RR-Zelle ($\tau_I = t_{RR}$).

Eine Besonderheit ist bei Iterationswiederholungen zu beachten, da hier *X* bzw. *Y* um den gleichen Wert 2^{-i} wie in der vorhergehenden Iteration verschoben werden und damit das beschriebene Prinzip nicht mehr anwendbar ist. Hierfür existieren zwei Lösungsmöglichkeiten. Einerseits kann in den Array-Stufen, die eine Iterationswiederholung ausführen, eine vollständige Addiererreihe entsprechend /8/ implementiert werden. Andererseits kann die Instanziierung der speziellen 4-Digit-Adder auch um *k* Iterationen später erfolgen, wobei *k* die Anzahl der Iterationswiederholungen repräsentiert.

Eine Alternative zu dem beschriebenen 4-Digit-Addierer ist ein 3-Digit-Addierer. Das Prinzip ist in Abbildung 6 bei einem Einsatz in 3 CORDIC-Iterationsstufen dargestellt und beruht auf folgender Überlegung. Gegeben ist ein Digit-Vektor $A=\{a_2, a_1, a_0\}$ mit dem Wertebereich $-6 \le A \le 6$. Dieser läßt sich derart umkodieren, so daß für den Teilvektor $A_1 = \{a_{1,1}, a_{1,0}\}$ ein Wertebereich von $-2 \le A_1 \le 2$ gilt. Bilden wir nun den Vektor $A_2 = \{a_{2,2}, a_{2,1}, a_{2,0}\}$ mit der Zuordnung $a_{1,1} = a_{2,2}, a_{1,0} = a_{2,1}$ und $a_{2,0} \in \{\bar{1}, 0, 1\}$. Dieser hat einen Wertebereich von $-5 \le A_2 \le 5$ und bei der Addition eines Carry-Digits $c \in \{\bar{1}, 0, 1\}$ entsprechend einen Wertebereich $-6 \le A_2 \le 6$. Damit kann mit diesem Verfahren eine Carry-Absorbierung aus einer niederwertigen Digitposition und eine gleichzeitige Umkodierung des Ergebnisvektors erfolgen, um in einem weiteren Schritt erneut einen Übertrag zu absorbieren. Die Realisierung kann mit einer Recoding-Einheit (REC) und einer CAB-Zelle realisiert werden. Der Vorteil dieser Architektur besteht darin, daß die Einsparung von Addiererzellen bereits eine Iteration früher als mit dem 4-Digit-Addierer erfolgen kann. Nachteilig dagegen ist die Vergrößerung der Rechenzeit einer Iterationsstufe, da die Latenzzeit τ_{IEC} größer der einer RR-Zelle ist ($\tau_{REC} > t_{RR}$). Für die Behandlung von Iterationswiederholungen gelten die oben gemachten Aussagen. Ein ähnliches Prinzip wird in /3/ für eine On-the-Fly-Konvertierung der MSD-Digits in eine binäre Darstellung verwendet.

Zur Implementierung des *Z*-Pfades wird Gleichung (3) entsprechend /3/ zu $z_{i+1} = 2\left(z_i - \sigma_i 2^{S(m,i)} \alpha_{m,i}\right)$ modifiziert.

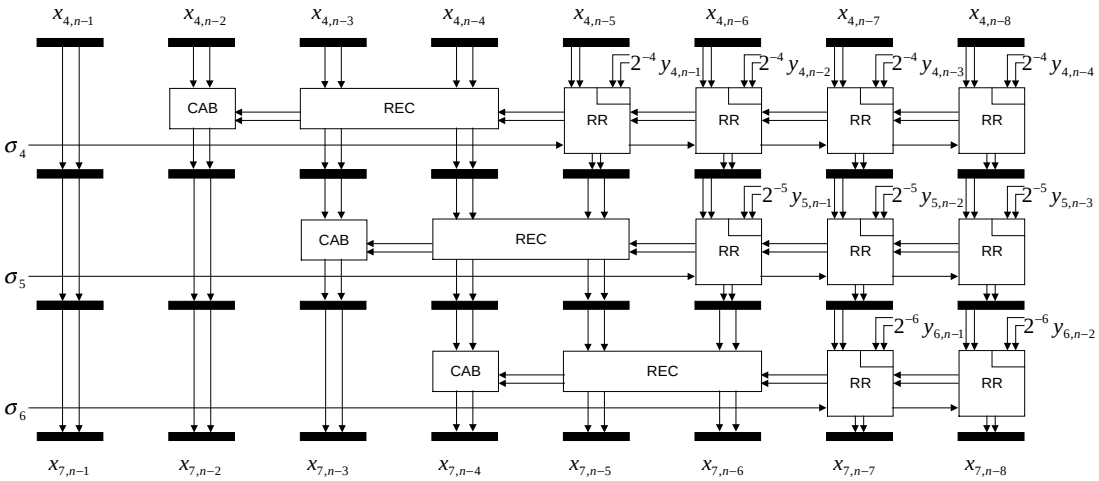

Bild 6 Reduzierter *X*-Pfad mit 3-Digit-Addierer

Die Ermittlung der drehrichtungsbestimmenden σ_i erfolgt damit in jeder Iteration an der gleichen Position über $t = 4$ führende Digit-Stellen. Entsprechend /8/ kann der *Z*-Pfad mit 3-2-Redundant-Binary (RB) Addierer/Subtrahierer Zellen und vom LSD her beginnend zunehmend reduzierten Addiererreihen ausgeführt werden. Ab der Iteration mit dem Index $i = (n + 1)/3$ kann die Berechnung in *Z* abgebrochen werden, wobei sich die restlichen σ_i aus dem zuletzt ermittelten z_i ergeben. Entsprechend /7/ sind die Iterationswiederholungen dann bis zu diesem Zeitpunkt auszuführen. Damit ergibt sich eine Chipfläche gemäß Bild 7.

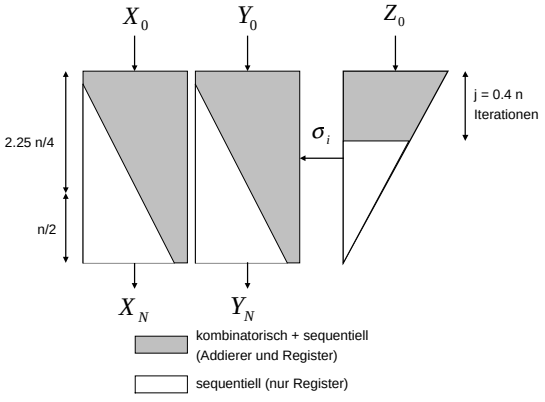

Bild 7 Chipfläche Rotation

4.3 Betriebsart Vectoring

Für den Einsatz redundanter Addierer werden (1) und (2) zu $x_{i+1} = x_i - m\sigma_i 2^{-2S(m,i)} y_i$ und $y_{i+1} = 2\left(y_i + \sigma_i x_i\right)$ modifiziert. Entsprechend werden im *Z*-Pfad sich stetig vermindernde Drehwinkel $\alpha_{m,i}$ zu z_i addiert, womit die gleichen Bedingungen wie im *X*- bzw.*Y*-Pfad des Rotation-Modus gegeben sind. Damit kann unter Verwendung der speziellen Addiererzellen gemäß dem Prinzip in Abbildung 5 eine nahezu dreiecksförmige Architektur des *Z*-Pfades bzgl. der kombinatorischen Fläche gebildet werden. Dabei ist zu beachten, daß anstelle der 4-2-RR-Zellen die 3-2-RB Addierer/Subtrahierer Zellen eingesetzt werden. Ebenso ist das genannte Prinzip der Minimierung der Addiererreihen im *X*-Pfad anwendbar, wobei gegenüber der Betriebsart Rotation zwei Besonderheiten bestehen. Einerseits erhöht sich die Anzahl der Nullen, die in den MSD-Positionen addiert werden, mit jeder Iteration um zwei, womit sich ebenfalls die Anzahl der gesparten Addiererzellen erhöht. Andererseits können die Berechnungen im *X*-Pfad nach $n/2$ Iterationen abgebrochen werden, da der *X*-Wert dann *n*-Digit-Genauigkeit erreicht hat. Die Situation im *Y*-Pfad ist vergleichbar mit der im *Z*-Pfad des Rotation-Modes. Allerdings ist hier die Gesamtzahl der Iterationen auszuführen und als Addiererzellen sind die 4-2-RR-Adder zu implementieren. Somit ergibt hier eine vom LSD her wachsende Reduzierung der Addiererreihen eine Dreiecksstruktur hinsichtlich der kombinatorischen als auch der sequentiellen Fläche.

5 Bewertung

Die Bewertung basiert auf der Annahme, das $m = 1$ gilt und jede vierte Iteration wiederholt wird. Weiterhin soll eine 4-2-RR Zelle, eine RZ-0 Zelle bzw. eine CAB-Zelle der Fläche von zwei Volladdierern (VA) und eine 3-2-RB-Zelle bzw. eine RZ-Zelle der Fläche von einem VA entsprechen. Ebenso verzichten wir zur Vereinfachung auf die Betrachtung der Skalierungsfaktorkompensation, der Vorzeichenbestimmung und der Guard- und Overflow-Digits. Im Rotation-Mode benötigen wir für den *X*- und *Y*-Pfad 2 * (2 * (2.25 n/4 * n/2 * 3/2 + n/2 * n/2 * 1/2)) = 2.19 n^2 und im Z-Pfad n^2/3 äquivalente VA, also insgesamt 2.52 n^2. Dazu kommen 2 * (2 * (2.25 n^2/4 + n^2/2)) + (2.25 n^2/4 + n^2/2) = 5.31 n^2 Latches. Im Vectoring-Mode benötigen wir für *X* (2.25 n/4 * n * 1/2) * 2 = 0,56 n^2 , für *Y* (4.25 n/4 * 1/2) * 2 = 1,06 n^2 und für *Z* 1/2 n^2 VA, entsprechend insgesamt 2,20 n^2 VA. Zusätzlich sind 5,31 n^2 Latches nötig. Tabelle 1 zeigt einen Vergleich zur Absolutwertmethode in /5/ und gemischten Basis 2-4-Architektur in /2/. Es sei darauf hingewiesen, daß wir eine reine Basis 2 Architektur verwenden. Die Basis 2-4 ist möglich und führt zu weiteren Reduzierungen auf Kosten der Regularität. Die Werte sind in der Tabelle mit angegeben.

Rotation	/5/	/2/	Basis 2	Basis 2-4
VA	5 n^2	3.5 n^2	2.52 n^2	2.27 n^2
Latch	7.75 n^2	4.62 n^2	5.31 n^2	4.62 n^2
Vectoring				
VA	5 n^2	3.5 n^2	2.21 n^2	*2.09* n^2
Latch	n^3/6 +9.5 n^2	4.62 n^2	5.31 n^2	4.62 n^2

Tabelle 1 Vergleich

Zur Evaluierung wurden VHDL-Modelle für einen Basis 2 Standard-CORDIC und die vorgeschlagenen Architekturen für den Rotation-Mode entwickelt, simuliert und auf eine Standardzellenbibliothek synthetisiert. Auf dieser Basis wurden Chip-Layouts für eine 1.0 µ CMOS-Technologie und Datenpfade mit einer Genauigkeit von extern 15 Bit realisiert. Durch die algorithmischen Modifikationen konnte gegenüber einem redundanten Standard-CORDIC eine effektive Flächeneinsparung von 25% bei gleicher Latenzzeit und Durchsatzrate und damit die bisher kleinste bekannte Realisierung einer CORDIC-Pipeline erreicht werden.

/1/ J.S. Walther, "A unified algorithm for elementary functions", Proc. of Spring Joint Computer Conf., S. 379-385, 1971

/2/ E. Antelo, J.D. Brugera, E.L. Zapata, "Unified mixed radix 2-4 redundant CORDIC processor", IEEE Trans. on Comp., B. 45, Nr. 9, S. 1068-1073, Sept. 1996

/3/ J.-A. Lee and T. Lang, "Constant-factor redundant CORDIC for angle calculation and rotation", IEEE Trans. on Comp., B. 41, Nr. 8, S. 1016-1025, Aug. 1992

/4/ D. Timmermann, H. Hahn, B.J. Hosticka, "Low latency time CORDIC algorithms", IEEE Trans. on Computers, B. 41, Nr. 8, S. 1010-1015, Sept. 1992

/5/ H. Dawid, H. Meyr, "The differential CORDIC algorithms: constant scale factor redundant implementation without correcting iterations", IEEE Trans. on Comp., Nr. 3, S. 307-318, März 1996

/6/ G. Schmidt, et al., "Parameter optimization of the CORDIC-algorithm and implementation in a CMOS-chip", Proc. EUSICO-86, B. 2, S. 1219-1222, Hague, Netherlands, 1986

/7/ D. Timmermann, I. Sundsbø, "Area and latency efficient CORDIC architectures", Proc. ISCAS'92, S. 1093-1096, San Diego, Mai 1992

/8/ D. Timmermann, S. Dolling, "Unfolded Redundant CORDIC VLSI Architectures With Reduced Area and Power Consumption", VLSI'97, Gramado, Brasilien, Aug. 1997

/9/ S. Kuninobu, et.al., "Design of high speed MOS multiplier and divider using redundant binary representation", Proc. 8th Symp. Computer Arithmetic, S. 80-86, New York, 1987