24 BIT CMOS GLEITKOMMA VEKTORARITHMETIK CHIP

D. Timmermann, B. Rix*, H. Hahn**, B.J. Hosticka**

** Fraunhofer-Institut für Mikroelektronische Schaltungen und Systeme*
Finkenstr. 61, 4100 Duisburg 1, ☎ (0203) 3783 219, FAX (0203) 3783 266
*** Consulting Services, 5883 Kierspe-Bollwerk*

Kurzfassung
Es wird eine CMOS Realisierung einer Vektorarithmetikeinheit zur Verarbeitung von Zahlen im IEEE-754 Gleitkommaformat (24 Bit Mantisse, 8 Bit Exponent) vorgestellt, der IMSCOR24. Dieser Chip wurde für Anwendungen in der digitalen Signalverarbeitung entwickelt, bei denen ein kontinuierlicher Datenstrom mit aufwendigen arithmetischen Operationen und möglichst hohem Durchsatz bearbeitet werden muß. Neben Multiplikations- und Divisionsbefehlen stehen u.a. die Funktionen Sinus, Kosinus, Hyperbelfunktionen, Quadratwurzeln, Logarithmen, Arcustangens, Vektorlänge, Vektorphase und Vektorrotation zur Verfügung. Alle bereitgestellten Funktionen werden mit einer normalisierten Durchsatzrate ("normalized peak performance") von bis zu 220 MFLOPS berechnet.

1. Einleitung

Anwendungsgebiete für den beschriebenen Chip sind zeitkritische Aufgaben z.B. in der

- Bildverarbeitung, Mustererkennung
- Grafikbeschleuniger
- Echtzeitsimulation, Robotik, Kinematik
- Signalverarbeitung, Daten- und Bildkompression, Signal- und Spektralanalyse
- Telekommunikation

In diesen Gebieten sind häufig die Algorithmen zur Lösung des jeweiligen Problems theoretisch bekannt, jedoch sehr rechenaufwendig. Bei der praktischen Umsetzung selbst mit neuesten kommerziell verfügbaren Signalprozessoren können dann die gewünschten Leistungsdaten, vor allem die Durchsatzrate, bei weitem nicht erreicht werden, wenn die verwendeten Algorithmen komplexere mathematische Funktionen beinhalten. In diesem Beitrag stellen wir eine Gleitkomma-Arithmetikeinheit vor, die auf dem COordinate Rotation DIgital Computer (CORDIC) Algorithmus basiert /1,2/ und eine außergewöhnliche Funktionalität aufweist. Obwohl der CORDIC Algorithmus in letzter Zeit zunehmend an Popularität gewinnt, wenn es um die Realisierung trigonometrischer oder hyperbolischer Funktionen geht, existieren bisher lediglich Implementierungen in Firmware oder nur eines Subsets des gesamten Funktionsumfangs /4,9/ für Spezialanwendungen. Demgegenüber realisiert der hier beschriebene Chip alle CORDIC Funktionen, im Gegensatz zu einer früheren rekursiven Festkomma-Lösung /5/ jedoch in einer Pipeline für Zahlen im Gleitkommafomat einfacher Genauigkeit nach IEEE-754 Standard, wodurch ein höherer funktionaler Durchsatz erreicht wird.

2. CORDIC Algorithmus

Die mathematische Basis des CORDIC Algorithmus bilden iterative Vektorrotationen /1,2/. Da die Iterationen weitgehend regulär und sehr leicht pipelinebar sind, eignen sie sich besonders für eine Integration. Außerdem erzielt man mit dem CORDIC Verfahren eine sehr hohe Funktionalität, verglichen mit der aufzuwendenden Chipfläche, da das Koordinatensystem besonders einfach geändert werden kann.
Der Iterationsteil der Schaltung implementiert die Iterationsgleichungen

$$\begin{aligned} x_{i+1} &= x_i - m\sigma_i\, 2^{-S(m,i)}\, y_i \\ y_{i+1} &= y_i + \sigma_i\, 2^{-S(m,i)}\, x_i \\ z_{i+1} &= z_i - \sigma_i \alpha_{m,i} \end{aligned} \qquad (1)$$

wobei m das Koordinatensystem festlegt, σ_i die Drehrichtung der Rotation, $S(m,i)$ die Shiftfolge, $\alpha_{m,i}$ den Teildrehwinkel und i den Iterationsindex. Der Parameter m wird zu 1, 0, oder -1 gewählt, wodurch die entsprechenden Vektorrotationen als Drehungen auf einem Kreis, einer Geraden oder einer Hyperbel interpretiert werden können.

Die Konvergenz des Verfahrens wird durch die feste Shiftfolge *S*(m,i) bestimmt, die ihrerseits den Teildrehwinkel gemäß folgender Gleichung festlegt:

$$\alpha_{m,i} = \frac{1}{\sqrt{m}} \arctan(\sqrt{m}\ 2^{-S(m,i)}) = \begin{cases} \arctan(2^{-S(m,i)}) & \text{für } m = 1 \\ 2^{-S(m,i)} & \text{für } m = 0 \\ \operatorname{artanh}(2^{-S(m,i)}) & \text{für } m = -1 \end{cases} \quad (2)$$

Im Verlauf der Iteration wird entweder der Wert für *z* oder *y* gegen Null gezwungen, indem man die Drehrichtung entsprechend

$$\sigma_i = \text{sign}(z_i) \quad \text{oder} \quad \sigma_i = -\text{sign}(x_i)\text{sign}(y_i) \quad (3)$$

wählt, wobei sign(β) = 1 für β ≥ 0 und sign(β) = -1 für β < 0. Durch Vorgabe des Iterationsziels und des Koordinatensystems kann eines der sechs Funktionspaare in Tabelle I berechnet werden.

	$z_n \rightarrow 0$ (Rotation)	$y_n \rightarrow 0$ (Vectoring)
m = -1 hyperbolisch	$x_n = k_{-1}\,(x_0 \cosh(z_0) + y_0 \sinh(z_0))$ $y_n = k_{-1}\,(x_0 \cosh(z_0) + y_0 \sinh(z_0))$	$x_n = k_{-1}\sqrt{x_0^2 - y_0^2}$ $z_n = z_0 + \operatorname{artanh}(y_0 / x_0)$
m = 0 linear	$x_n = x_0$ $y_n = x_0\, z_0 + y_0$	$x_n = x_0$ $z_n = z_0 + y_0 / x_0$
m = 1 zirkular	$x_n = k_1\,(x_0 \cos(z_0) - y_0 \sin(z_0))$ $y_n = k_1\,(y_0 \cos(z_0) + x_0 \sin(z_0))$	$x_n = k_1\sqrt{x_0^2 + y_0^2}$ $z_n = z_0 + \arctan(y_0 / x_0)$

Tabelle I: CORDIC - Funktionen ($k_{-1,0,1}$ ≡ Skalierungsfaktoren)

Die Startwerte der Iteration und damit die Funktionsargumente sind mit x_0, y_0, und z_0 bezeichnet und k_m repräsentiert den sogenannten Skalierungsfaktor, eine unvermeidbare, algorithmusbedingte Verlängerung oder Verkürzung des Vektors während der Iteration. Da dieser Faktor im allgemeinen ungleich Eins ist, muß er nachträglich kompensiert werden.
Verschiedene Lösungsmöglichkeiten stehen zur Kompensation des Skalierungsfaktors zur Verfügung. Eine häufig anzutreffende Methode besteht darin, einige Iterationen zu wiederholen, um so einen besonders leicht zu kompensierenden Faktor zu erhalten (z.B. 2 oder 0,5), denn k_m ist mit der Shiftfolge *S(m,i)* durch die Beziehung

$$k_m = \prod_{i=0}^{n-1} \sqrt{1 + m2^{-2S(m,i)}}$$

verknüpft. Ein alternativer Ansatz beruht auf Doppelshifts, d.h. ein zusätzlicher Shift wird in die Iterationsgleichung eingearbeitet /3/. Die im vorliegenden Chip verwendete Kompensation verwendet eine hybride Lösung, indem man nur einige Iterationen verdoppelt und $1/k_m$ multiplikativ in möglichst wenig Faktoren (1 ± 2^{-j}) aufgespalten wird [8].

3. Funktionaler Überblick

Bild 1 zeigt das Blockschaltbild des Chips. Der Eingangsdatenstrom wird auf die drei Eingangsports *x*, *y* und *z* gegeben, und nach einer Pipelinelatenz von insgesamt 44 Taktzyklen werden die Daten an den Ausgangsports *x* und *yz* ausgegeben (die Bezeichnungen *M* und *E* in Bild 1 stehen für Mantisse und Exponent der Gleitkommazahl). Der Chip erlaubt die Berechnung aller in Tabelle I spezifizierten Funktionen. Beispielsweise wird eine komplette Koordinatentransformation im kartesischen Koordinatensystem mit der Instruktion ROT durchgeführt. Die wichtigsten unterstützten Instruktionen sind in Tabelle II zusammengefaßt.

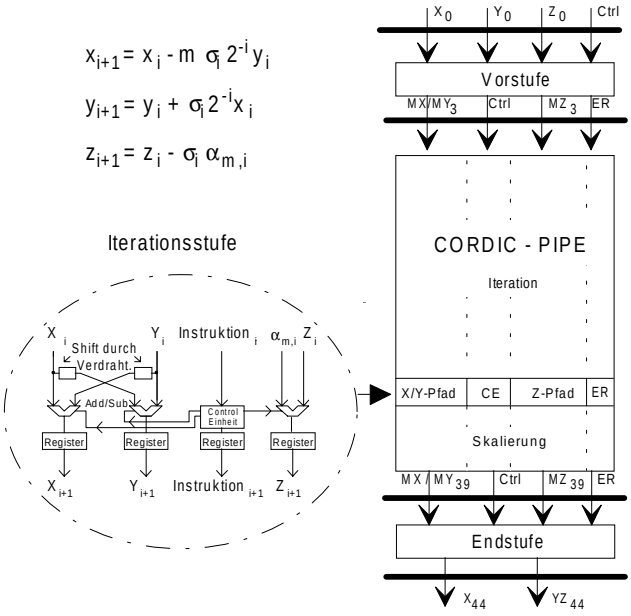


Bild 1: Blockdiagramm des IMSCOR24

Instruk-tion	Ergebnis	
	x	yz
DIVADD	x	z+y/x
DIVSUB	x	z-y/x
MULADD	x	y+z*x
MULSUB	x	y-z*x
HVECT	$(x^2-y^2)^{0.5}$	z+artanh(y/x)
HVECTM	$(x^2-y^2)^{0.5}$	z-artanh(y/x)
HROT	x*cosh z + y*sinh z	y*cosh z + x*sinh z
HROTM	x*cosh z - y*sinh z	y*cosh z - x*sinh z
VECT	$(x^2+y^2)^{0.5}$	z+arctan(y/x)
VECTM	$(x^2+y^2)^{0.5}$	z-arctan(y/x)
ROT	x*cos z - y*sin z	y*cos z + x*sin z
ROTM	x*cos z + y*sin z	y*cos z - x*sin z
IHROT#	x*cosh z' + y*sinh z'	y*cosh z' + x*sinh z'
IHROTM#	x*cosh z' - y*sinh z'	y*cosh z' - x*sinh z'
IROT#	x*cos z' - y*sin z'	y*cos z' + x*sin z'
IROTM#	x*cos z' + y*sin z'	y*cos z' - x*sin z'
FX....	wie oben, Festkomma	

Tabelle II: Funktionstabelle

Weitere Funktionen können durch die Wahl bestimmter Eingangswerte berechnet werden, wie in Tabelle III dargestellt.

Der Chip verwendet intern eine Festkomma-Pipeline, die festverdrahtete Additions-und-Schiebeoperationen enthält. Die Festkomma-Pipeline führt die CORDIC Iterationsgleichungen und die anschließende Skalierungsfaktorkompensation aus. Jede Iterationsstufe implementiert eine Iteration des Algorithmus. Der 29-stufigen Festkomma-Pipeline geht eine Vorstufe voraus, die die Eingangsdaten vom externen Gleitkommaformat nach IEEE-754 (24 Bit Mantisse, 8 Bit Exponent) in das interne Festkommaformat (32 Bit, siehe Bild 2) umsetzt. Die Berechnung selbst erfolgt mit einer modifzierten Gleitkommaversion des CORDIC Algorithmus, die ein Verschieben der Mantissen und Exponentberechnungen innerhalb der inneren Festkommapipeline vermeidet /6/. Prinzipiell wird dabei nach dem gleichen Schema wie bei Gleitkomma-Addierern vorgegangen: 1) Bestimmung eines gemeinsamen Exponenten (Referenzexponent) für die Argumente, 2) binäres Schieben der Mantissen in Abhängigkeit von der Differenz zwischen dem Exponent des jeweiligen Exponenten und des Referenzexponents, 3) Durchführung der Berechnung mit den geschobenen Mantissen, 4) Normalisierung der Mantissen und entsprechende Berechnung des zugehörigen Exponenten aus dem Referenzexponenten. Im Falle von CORDIC stellt sich dies komplizierter dar, weil hier prinzipiell mit drei Argumenten gearbeitet wird.

Zunächst wird, wie in Bild 3 skizziert, aufgrund der in IEEE-754 vorgeschriebenen Vorzeichen-Betragsdarstellung der Zahlen das Zweierkomplement der drei Mantissen gebildet (im Block TCOMPL) und ein Bias von 127 subtrahiert (BIAS), um den 10 Bit breiten Referenzexponenten zu erhalten. Dieser wird gemäß Tabelle IV gebildet und unverändert in der Festkomma-Pipeline in speziellen Registern mitgeführt. Mehrere Zwischenergebnisse legen fest, um wieviel die Mantissen zu schieben sind. An den Iterationsteil der Festkomma-Pipeline schließen sich 8 Skalierungsstufen an, die den Skalierungsfaktor k_m kompensieren.

Auf die Festkomma-Pipeline folgt eine Endstufe (Bild 4), in der die Mantissen normalisiert und gerundet werden und daraus die Ergebnisexponenten berechnet werden, so daß die Ausgabedaten dem IEEE Format genügen. Dies bedingt eine Umwandlung in Vorzeichen/ Betragsdarstellung (TCOMPL). Im Block LZDC wird für die Mantissen die Anzahl der führenden Nullen hinter dem Dezimalpunkt bestimmt und damit die Barrelshifter zur Normalisierung angesteuert. Mit diesen Ergebnissen und dem in der Vorstufe berechneten Referenzexponenten können die drei Ergebnisexponenten bestimmt werden. Als Rundungsverfahren findet das Sticky-Bit Runden Anwendung für die Mantissen, wodurch kein Übertrag generiert wird und der Rundungsfehler auf 1 LSB begrenzt wird. Der Datenfluß wird durch das 6 Bit breite Instruktionswort kontrolliert sowie durch das Iterationsziel und die Wahl des Koordinatensystems. Zusätzlich zum Gleitkommaformat wird auch ein 24 Bit Festkommaformat unterstützt. Ein spezieller Bit-Pipelining Modus ist vorgesehen, um bestimmte Matrixalgorithmen erheblich zu beschleunigen, z.B. QR Zerlegung und Givens Rotation. Bei derartigen Anwendungen berechnet man zunächst die Phase eines Vektors und dreht folgende Vektoren um diese Phase. Um die Pipelinelatenz von 44 Zyklen zu vermeiden, sind spezielle Instruktionen (in Tabelle II mit # gekennzeichnet) vorhanden, die Rotationen um den Winkel der letzten Phasenberechnung durchführen. Angenommen, ein kontinuierlicher Datenstrom von Vektoren mit Endkoordinaten (a_2,b_2),

(a_3,b_3), ... sei um arctan(b_1/a_1) zu drehen. Die Instruktionssequenz dafür lautet: VECT(a_1,b_1), IROT(a_2,b_2), IROT(a_3,b_3) etc. und die Berechnung wird mit der vollen Durchsatzrate durchgeführt.

Instruk-tion	Eingabe			Ergebnis	
	x	y	z	x	yz
ROT	x	0	z	x cos(z)	x sin(z)
HROT	x	0	z	x cosh(z)	x sinh(z)
HVECT	x+1	x-1	0	$2\sqrt{x}$	½ ln(x)
HROT	x	x	z	x e^z	x e^z
HVECT	x	1	0	$\sqrt{x^2-1}$	$\coth^{-1}(x)$
HVECT	x+¼	x-¼	0	$\sqrt{x}$	ln(¼/x)
HVECTM	1	y	π/2	$\sqrt{y^2+1}$	$\cot^{-1}(y)$
HVECT	x+y	y-x	0	$2\sqrt{x \cdot y}$	½ ln(y/x)

Tabelle III: Zusätzliche Funktionen

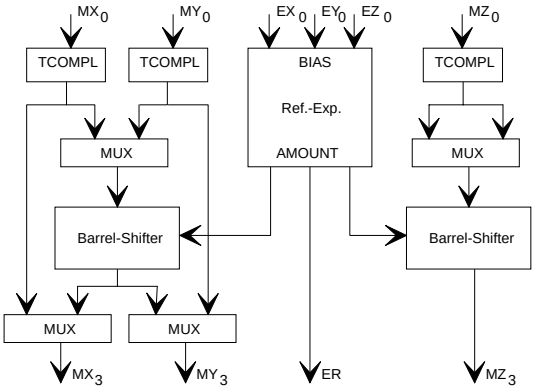


Bild 3: Vorstufe

m	ER, Vectoring	ER, Rotation
1	max (EY_0,EX_0)	max (EY_0,EX_0)
0	max (EZ_0, EY_0-X_0)	max (EY_0, EX_0+Z_0)
-1	max (EY_0,EX_0)	max (EY_0,EX_0)

Tabelle IV: Referenzexponent ER

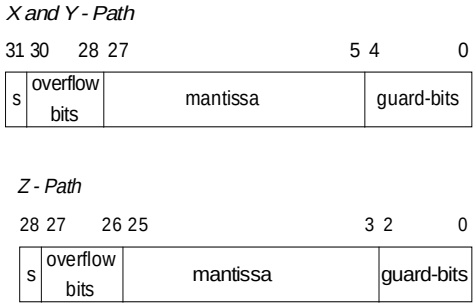


Bild 2: Internes Datenformat

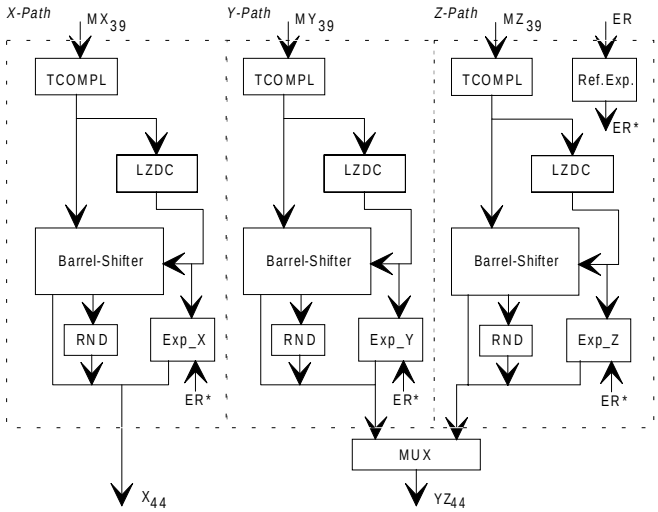


Bild 4: Endstufe

4. Design

Die Realisierung der CORDIC Gleichungen nach (1) bedingt drei Datenpfade. Aufgrund des größeren Ergebnisbereichs der berechneten Funktionen und der Add-and-shift Charakteristik des Algorithmus sind zusätzliche Überlauf und Schutzbits bereitzuhalten. Für das Gleitkomma IEEE-754 Format einfacher Genauigkeit sind zwei interne 32-Bit Pfade (1 Vorzeichen, 3 Überlauf-, 23 Nachkomma- und 5 Schutzbits) für *x* und *y* und ein 29 Bit Datenpfad (1 Vorzeichen, 2 Überlauf-, 23 Nachkomma- und 3 Schutzbits) für *z* erforderlich. Zusammen mit der 29-stufigen Festkomma-Pipeline und den 8 Skalierungsstufen und 10 Registerbits für den Referenzexponenten summiert sich dies zu einen 103 Bit breiten × 37 Stufen tiefen internen Datenpfad, so daß die Chipfläche von zentraler Bedeutung wird.

Zur Durchführung der CORDIC Gleichungen müssen die *x* und *y* Mantissen jeweils geschoben und überkreuz addiert/subtrahiert werden (vgl. Glg. (1) und Bild 1). Die direkte Umsetzung dieses Datenflusses in Silizium würde zu sehr irregulären Leitungskreuzungen und einer Mindestlänge der Verdrahtung horizontal über 32 Bit führen. Aus diesem Grund sind der *x* und *y* Pfad im Layout bitweise nebeneinander plaziert, wodurch sich die Mindestverdrahtungslänge auf horizontal 2 Bit reduziert (jeweils 1 Bit für *x* und *y*).

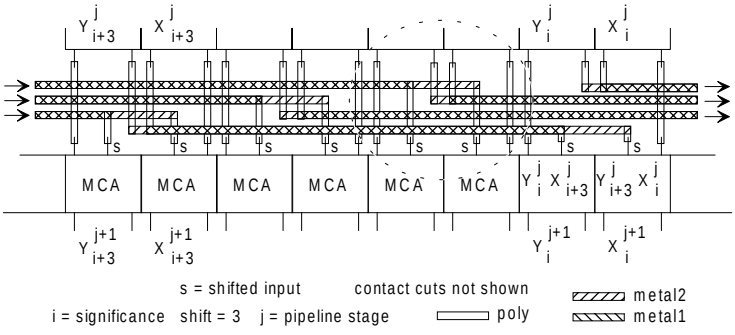
Die Plazierung und Verdrahtung der Festkomma-Pipeline hängt von der verwendeten Shiftfolge $S(m,i) = S(i)$ ab, die in Tabelle V dargestellt ist. Diese wurde in Computersimulationen durch Optimierung der Iterationsanzahl und der resultierenden Fehlerverteilung gefunden /8/.

Für das Layout der internen Festkomma-Pipeline wurde ein flächenoptimierender Modulgenerator entwickelt. In Bild 5 ist das Layout- und Verdrahtungsschema des Generators zusammen mit dem bitweise geschachtelten *x* und *y* Pfad und der Shiftverdrahtung dargestellt (MCA steht für Manchester Carry Chain Addierer).

i	$S(i)$	i	$S(i)$	i	$S(i)$	i	$S(i)$	i	$S(i)$	i	$S(i)$
1	1	6	3	11	7	16	12	21	16	26	21
2	2	7	4	12	8	17	13	22	17	27	22
3	2	8	5	13	9	18	13	23	18	28	23
4	2	9	6	14	10	19	14	24	19	29	24
5	2	10	6	15	11	20	15	25	20	-	-

m	$a(m,1)$	$a(m,2)$	$a(m,3)$	$a(m,4)$	$a(m,5)$	$a(m,6)$	$a(m,7)$	$a(m,8)$
1	-2	4	-5	6	0	17	-20	0
0	0	0	0	0	0	0	0	0
-1	2	4	0	6	-6	0	-20	-21

Tabelle V: Shiftfolge und Skalierungsparameter [8]



Prozeß	Double-metal CMOS
Design Rule	1.6µm
Chipfläche	13.3 x 14.2 mm
Aktive Fläche	8.2 x 13.4 mm
Transistoren	210 000+
Gehäuse	280 Pin PGA
# Pipe Stufen	(3+37+4)= 44
Max. Clock	10 MHz
Datenformat:	
extern	24b mant., 8b exp. (IEEE-754 single prec.) oder 24b Festkomma
intern	32b mant., 10b exp.
Instruktion	6bit

Tabelle VI: Technische Daten

Bild 5: Shiftverdrahtung

Jede Bitzelle enthält zwei identische Ausgänge am linken und rechten Zellenrand. Zusammen mit einer asymmetrischen Anordnung der zu schiebenden Eingänge (gestrichelter Bereich) und Verdrahtung der längeren verschobenen *x* Eingänge in Metal2 und der verschobenen *y* Eingänge in Metal1 wird ein Verdrahtungskanal eingespart. Insgesamt werden im ungünstigsten Fall maximal 16 Kanäle belegt und die gesamte Shiftverdrahtung nimmt nur 18% der Chipfläche der Festkomma-Pipeline ein. Der Generator plaziert und verdrahtet die interne Pipeline in 25 s (VAX 11-8550) und ist flexibel bezüglich der Design-Regeln und der Algorithmusparameter. Er berechnet die gerundete binäre Darstellung der Drehwinkel $\alpha_{m,i}$ und programmiert die Eingänge des dem *z* Addierer vorgeschalteten 3-zu-1 Multiplexers an VDD oder VSS. Aus Chipflächengründen entschieden wir uns für einen Ripple-Carry Addierer, aufgebaut als Manchester Carry Chain mit 4 Bit Slices und Carry Buffern nach 2 Bit. Die kapazitive Belastung durch Sign-Extension der zu schiebenden MSBs wird durch einen optimierten MSB Buffer aufgefangen. Der Generator wurde so entwickelt, daß auch schnelle Algorithmen mit redundanten Addierern (redundant binär /4/ oder carry save) realisiert werden können. Allerdings ist dann von einer mehr als verdoppelten Chipfläche auszugehen. Die Vor- und Endstufe wurden manuell layoutet.

Zur Bestimmung der Drehrichtung muß σ_i bekannt sein. Dazu werden in einer Kontrolleinheit zwischen dem *xy* und *z* Pfad die Vorzeichen der drei Pfade ausgewertet und entsprechend σ_i nach Gleichung (3) bestimmt. Im Bit-Pipelining Modus wird das berechnete σ_i festgehalten und für die nächste VECT Instruktion benutzt /3/. Die Skalierungsfaktorkompensation erfolgt durch Multiplikationen von *x* und *y* mit $(1+\text{sign}(a(m,i))2^{-|a(m,i)|})$, wobei $a(m,i)$ durch Tabelle V definiert ist. Dies resultiert in einer Vektorkontraktion (m = 1, $1/k_1$ = 0.784039965) oder -expansion (m = -1, $1/k_{-1}$ = 1.327798882). Dazu kann die gleiche Hardware wie bei der Iteration verwendet werden, nur die Shiftverdrahtung und die Kontrolleinheit ändert sich. Die unterschiedlichen, von *m* abhängigen Bitshifts in jeder Skalierungsstufe werden durch einen von *m* gesteuerten 3-zu-1 Multiplexer und entsprechende Programmierung von dessen Eingängen realisiert.

Scanpfadregister verbinden die Vor- und Endstufe mit der internen Pipeline. Aufgrund deren hohen Regularität und des unvertretbaren Chipflächenbedarfs wurde auf Scanpfade in der Festkommapipeline verzichtet. Die wesentlichen technischen Daten des Chips (Bild 6) sind in Tabelle VI zusammengefaßt. Es erreicht eine normalisierte Spitzenleistung von 220 MFLOPS.

5. Zusammenfassung

Es wurde über eine auf dem CORDIC Verfahren basierende Pipeline-Arithmetikeinheit berichtet, die erstmals alle durch dieses Verfahren möglichen Funktionen berechnet. Sie unterstützt das IEEE-754 Format für Gleitkommazahlen einfacher Genauigkeit, wodurch hohe Funktionalität und Durchsatz erreicht werden konnten. Dazu wurde ein hochreguläres und modulares Konzept für Algorithmus, Architektur und Layout

entwickelt. Mit einem u.a. in der Wortbreite parametrisierbaren Modulgenerator wurde das Layout der internen Pipeline vollständig generiert. Dadurch ist ein problemloser Übergang auf kundenspezifische Wortbreiten sowie skalierte Technologien und damit ermöglichte leistungsstärkere Berechnungsverfahren gesichert, verbunden mit einer Leistungssteigerung um das Zehnfache. Weitere Chipflächeneinsparungen und Leistungssteigerungen sind mit den in /7/ und /10/ beschriebenen Techniken möglich.

Bild 6: Chipfoto

Literatur

/1/ J.E. Volder, "The CORDIC trigonometric computing technique", *IRE Transactions on Electronic Computers*, Band EC-8, Nr.3, S. 330-334, Sept. 1959

/2/ J.S. Walter, "A unified algorithm for elementary functions", *Proceedings of Spring Joint Computer Conference*, (SJCC), AFIPS Proc., Bd. 38, S. 379-385, 1971

/3/ E.F. Deprettere et.al., "Pipelined CORDIC architectures for fast VLSI filtering and array processing", ICASSP'84, S. 41 A6.1-6.4, San Diego, März 1984

/4/ H. Yoshimura et. al., "A 50 MHz CMOS geometrical mapping processor", Digest of Techn. Papers, IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, S. 162-163, Febr. 1988

/5/ D. Timmermann, H. Hahn, B.J. Hosticka, und G. Schmidt, "A programmable CORDIC chip for digital signal processing applications", IEEE Journal of Solid-State Circuits, Band 26, Nr. 9, S. 1317-1321, Sept. 1991

/6/ B. Yang, "Untersuchung zu einem 32-Bit-Gleitkomma CORDIC-Prozessor", Diplomarbeit, Institut für Informationstechnik, Ruhr-Universität Bochum, Juli 1986

/7/ D. Timmermann und I. Sundsbø, "Area and latency efficient CORDIC architectures", IEEE International Symposium on Circuits and Systems (ISCAS´92), S. 1093-1096, San Diego, Mai 1992

/8/ D. König und J.F. Böhme, "Optimizing the CORDIC algorithm for processors with pipeline architecture", Proc. EUSIPCO-90, S. 1391-1394, Barcelona, Sept. 1990

/9/ E.F. Deprettere, A.A.J. de Lange, und P. Dewilde, "The synthesis and implementation of signal processing applications specific VLSI CORDIC arrays", Proc. ISCAS-90, S. 974-977, New Orleans, Mai 1990

/10/ D. Timmermann, H. Hahn, und B.J. Hosticka, "Low latency time CORDIC algorithms", IEEE Transactions on Computers, S. 1010-1015, August 1991