Unfolded Redundant CORDIC VLSI Architectures With Reduced Area and Power Consumption

Dirk Timmermann, Steffen Dolling
University of Rostock, R.-Wagner-Str. 31, D-18119 Rostock, Germany, email: dtim@baltic.e-technik.uni-rostock.de

Abstract

The CORDIC algorithm has been widely used as a powerful and flexible generic architecture to implement many algorithms involving non-trivial arithmetic. However, when using its fastest, i.e. unfolded, implementation it exhibits excessive silicon area demands. Exploiting some peculiarities of the algorithm permits simpler hardware structures in the unfolded case yielding a substantial area reduction. Thus, power consumption is decreased, too. These reductions have no speed penalty. The benefits of the improved architecture have been verified by developing VHDL models and synthesizing sample layouts for comparison purposes.

Keywords

VLSI architecture, CORDIC, Application specific systems, Computer arithmetic, Digital signal processing, Low Power Design, Parallel architectures

1. INTRODUCTION

For several applications CORDIC processing units have been shown to deliver superior performance when compared with more conventional approaches. This is due to the fact that many advanced algorithms can be interpreted as generalized vector rotations, for which CORDIC is especially suited to.

Considering the implementation part of signal processing systems, the main shortcoming of CORDIC-based pipeline or array architectures, i.e. so called unfolded architectures, is their increased hardware complexity. In this paper we address CORDIC´s hardware complexity and describe architectures with considerably reduced chip area requirements. It will be shown that this area shrinking is obtained with no loss in speed.

2. REDUNDANT CORDIC ALGORITHMS

The CORDIC algorithm is defined by recurrences in three coordinates /1/:

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

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

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

where m denotes the coordinate system (m=1 means circular, m=0 linear, m=-1 hyperbolic), $S(m,i)$ the shift sequence $S(0,i)=S(1,i)=0,1,..,n$ and $S(-1,i)=1,2,3,4,4,5,..,3i,3i+1,3i+1,3i+2,..n$, $\alpha_{m,i}$ is called the rotation angle, σ_i denotes the rotation direction, usually $\sigma_i \in \{-1,1\}$ in non-redundant number systems and $\sigma_i \in \{-1,01\}$ in redundant systems (but $\sigma_i = 0$ has to be avoided, cf. below). The precision in bit is given by n while N means the number of iterations with $N = n$ for $m = 0$ and $m = 1$ (for $m = -1$ some extra iterations are necessary, see $S(m,i)$). The rotation angle depends on $S(m,i)$ according to

$$\alpha_{m,i} = \frac{1}{\sqrt{m}}\tan^{-1}(\sqrt{m}\,2^{-S(m,i)}) = \begin{cases} \tanh^{-1}(2^{-S(-1,i)}) & \text{for } m=-1 \\ \tan^{-1}(2^{-S(1,i)}) & \text{for } m=1 \\ 2^{-S(0,i)} & \text{for } m=0 \end{cases} \tag{4}$$

Two operational modes are possible, rotation or vectoring. The rotation direction factor σ_i is determined by the following equation:

$$\sigma_i = \begin{cases} \text{sign}(z_i) & \text{for } z_i \to 0 \quad \text{(i.e. rotation)} \\ -\text{sign}(x_i\,y_i) & \text{for } y_i \to 0 \quad \text{(i.e. vectoring).} \end{cases} \tag{5}$$

where $\text{sign}(\lambda) = 1$ for $\lambda \geq 0$, else $\text{sign}(\lambda) = -1$. The algorithm converges for all input data inside the region of convergence, given by

$$C_m = \sum_{i=0}^{N-1}\alpha_{m,i} \geq \begin{cases} \left| \frac{1}{\sqrt{m}}\tan^{-1}(\sqrt{m}\,y_0/x_0) \right| & \text{for } y_n \to 0 \\ \left| z_0 \right| & \text{for } z_n \to 0 \end{cases} \tag{6}$$

The solution of the recurrences (1-3) is given in Table I with

$$k_m = \prod_{i=0}^{N-1}(1+m\,\sigma_i^2\,2^{-2S(m,i)})^{1/2} \tag{7}$$

being the scaling factor and x_0, y_0, and z_0 the starting values of the iterations.

Equation (5) is only valid for non-redundant addition schemes. As these are carry-dependent and, consequently, inherently slow, some recently described proposals use redundant adders like carry-save /2/ or redundant binary /3,4,5,6/ adders. However, while delivering superior speed due to their carry-independent nature, redundant addition schemes introduce certain algorithmic difficulties. The sign of

a number can not be easily derived from the sign bit in the leftmost bit position as in the two's complement number system. In fact, in redundant schemes the most significant digit could equal zero. This would suggest choosing $\sigma_i = 0$, but this choice is commonly prohibited due to the inevitable variation of the otherwise fixed scaling factor k_m, thus loosing all benefits of easy scaling factor compensation /6/. So further bits, at worst all bits, would have to be inspected for sign evaluation, loosing the speed advantage.

To overcome this conflict most authors /2,3,4,6/ utilize the approach found in fast SRT-division /7/: some most significant digits of the number are inspected and this estimate is used to determine an appropriate σ_i. Provided the absolute value of the estimate is above a specific margin, σ_i is set to +1 or -1, respectively, otherwise to 0. This choice can disturb the convergence behavior of the algorithm due to possibly wrong rotation direction. As has been shown in /2,3,4,6/ this can be compensated by simple iteration doubling after a specific number of microrotations.

	$z_n \rightarrow 0$ (rotation)	$y_n \rightarrow 0$ (vectoring)
$m = -1$	$x_n = k_{-1}(x_0\cosh(z_0)+y_0\sinh(z_0))$ $y_n = k_{-1}(x_0\cosh(z_0)+y_0\sinh(z_0))$	$x_n = k_{-1}\sqrt{x_0^2 - y_0^2}$ $z_n = z_0+\tanh^{-1}(y_0/x_0)$
$m = 0$	$x_n = x_0$ $y_n = x_0 z_0 + y_0$	$x_n = x_0$ $z_n = z_0 + y_0 / x_0$
$m = 1$	$x_n = k_1 (x_0 \cos(z_0)-y_0 \sin(z_0))$ $y_n = k_1 (y_0 \cos(z_0)+x_0 \sin(z_0))$	$x_n = k_1\sqrt{x_0^2 + y_0^2}$ $z_n = z_0+\tan^{-1} (y_0/x_0)$

Table I: CORDIC functions

The number of iteration doublings linearly depends on the number of inspected digits /3,4,6/. For p inspected digits each (p-1)th iteration has to be repeated.

An alternative approach to keep the scaling factor constant is given in /8/ by using a reformulation of the CORDIC algorithm and computing absolute values. This idea is an adaptation of an algorithm first described to speed up the add-compare-select loop in Viterbi decoders. Unfortunately, it nearly doubles the amount of registers in the pipeline, thus significantly increasing chip area and latency. Therefore, this idea can not be used when chip area is the primary issue. In the following we build upon the iteration doubling method /2,3,4,6/.

Due to possible finite word length effects, angle errors and overflow prevention, we need a unified machine word length of $w_{xy} = n+g_{xy}+o_{xy}+1 = n+\log_2(n)+3$ bit for the x and y datapath and $w_z = n+g_z+o_z+1= n+\log_2(n/3)+4$ bit for the z datapath. A more detailed discussion of CORDIC´s inherent quantization errors can be found in /9/.

3. SILICON AREA CONSIDERATIONS

A typical architecture that implements the recurrences (1-3) in an unfolded spatial array manner maps each iteration to one row of the array. As we assume $N \cong n$ iterations and three datapaths with an internal word length of at least n bit (see above) the area complexity of a CORDIC array is proportional to $3n^2$. As an example, the required silicon area of the chip described in /10/, a IEEE-754 single precision floating point CORDIC pipeline implemented with non-redundant adders, exceeds 150 mm^2 in a 1.5 µm CMOS technology. Two thirds of this area are devoted to the iteration execution. Other published implementations with similar hardware efforts include /11,12,13/. Obviously, any reduction in chip area due to algorithmic and architectural modifications would improve yield and decrease production cost. In addition it reduces the power consumption, which is considered to be of growing significance in wireless applications. The motivation for this work is, therefore, to reduce the chip area to allow the application of CORDIC array and pipeline architectures in die size critical areas, i.e. embedded control, but with no speed penalty.

Previous work on area reduction has focused on different parts of the algorithm. At first, many researchers investigated methods to reduce the hardware amount necessary for scaling factor compensation by incorporating the scaling into the iteration or by optimizing the number of scaling iterations /15/.

Further research concentrated on the optimization of constant scale factor algorithms emerging from the application of redundant adders, as mentioned before. Subsequently, some authors tried to reduce the number of iterations, i.e. /5/ by applying a prediction scheme in the rotation mode resulting in a z-path reduction to roughly one third. This method can be also extended to the vectoring mode, as has been demonstrated in /16/. Thus the hardware requirements of the z-path decrease considerably for rotation and vectoring mode.

The prediction of some σ_i in the rotation mode /5/ allows for partly recoding σ_i so as two iterations in x and y can be multipexed to one and selecting one of two different shifts. Recently, this concept has been generalized to vectoring by adapting radix-4 SRT-division operand-prescaling /17/ yielding a unified mixed radix 2-4 architecture with $n/4$-1 less microrotations than a pure radix-2 architecture /6/.

One observation which can be made is that the aforementioned modifications trade algorithmic and architectural simplicity of CORDIC for an overall reduction of microrotations. Therefore, for non-redundant addition schemes the impact of decreasing magnitudes of x_i y_i and z_i in vectoring and rotation mode, respectively, has been investigated in /16/, yielding a decrease in the widths of the corresponding datapaths and, consequently, lower silicon area. In the following, we investigate whether this approach can be extended to the redundant case, too.

4. AREA REDUCED ARCHITECTURES

4.1 Rotation mode x- and y- datapath

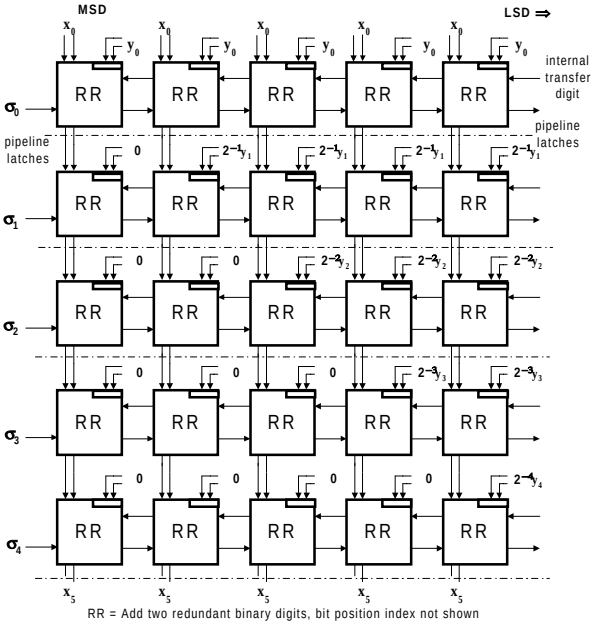

Fig. 1: Section of datapath for standard *x*-iterations

Figure 1 depicts the usual way to implement the *x* iterations (shown for *m*=1). The iterations for *y* are implemented correspondingly.
Here we chose 4-2 redundant binary adders for implementation. They can be realised in static CMOS circuit technique as given in /18/. The situation for carry-save adders is about the same. It should be noted that appropriate measures have to be taken to avoid pseudo overflows due to the redundant representation. This can be readily incorporated into the 4-2 redundant binary adder cell /19/ and will not be indicated here for sake of simplicity. For the same reason we do not consider any iteration repetition necessary for guaranteeing a constant scaling factor. Starting with the second iteration and the most significant digit an increasing number of zeros has to be added to the corresponding digits of x_i due to the shift sequence.
In these positions, it is sufficient for proper iteration results to take into account the previous value of x_i (y_i in the *y*-datapath) and the transfer digit from the next lower significant digit position. Hence, a much simpler redundant addition cell can be employed. Fig. 2 demonstrates the corresponding architecture. We refer to this adder as a 2-2 redundant binary adder (RBA). It can be noted that not only a simpler addition resulting from one fixed addend occurs.

Thus the circuit for one digit position reduces from a 4-2 RBA, sign inverting logic, and two latches to one 2-2 RBA and two latches. In Fig. 3 a fully static CMOS version of a 4-2 RBA requiring 42 transistors is shown /18/. The redundant binary numbers are coded in sign and amplitude (value) format, indicated by indices *s* and *a*, respectively. The additional steering and inverting logic amounts to one inverter and one 2-1 multiplexer, adding at least 6 transistors when employing transmission gate logic. In summary, we need 48 transistors. The signals *p* and *q* represent the internal transfer digit. Dotted areas indicate AOI-gates.

Fig. 4 exhibits a 3-2 RBA cell, which was reported in /18/. It requires 22 transistors.

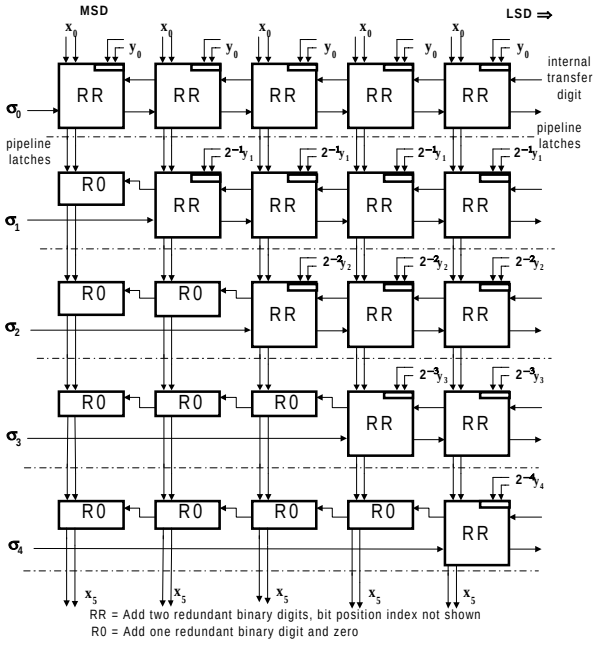

Fig. 2: Reduced datapath for standard *x*-iterations

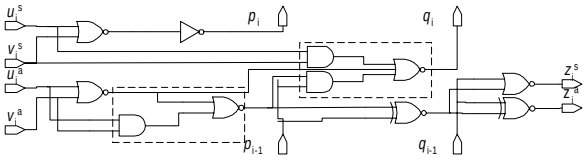

Fig. 3: Static CMOS 4-2 RBA cell /18/

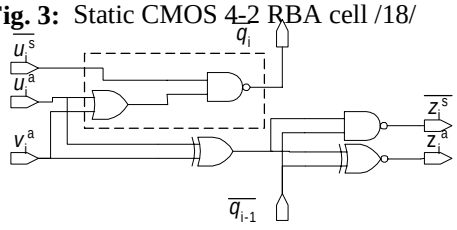

Fig. 4: CMOS 3-2 RBA cell /18/

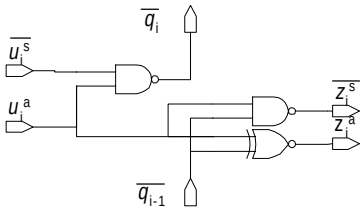

Fig. 5: CMOS 2-2 RBA cell

On the other hand, the 2-2 RBA can be directly derived from this circuit, yielding the circuitry shown in Fig. 5, requiring only 14 transistors.

4.2 Rotation mode z-datapath

For redundant adders, the original *z*-iteration according to Eq. 3 is modified to $z_{i+1} = 2(z_i - \sigma_i\, 2^{S(m,i)}\alpha_{m,i})$ /4/. In this way, the critical bits to be examined for sign estimation are fixed at the same position for all iterations. While this is mandatory for recursive implementations it also improves the regularity of the physical implementation in array and pipeline structures, as shown in Fig. 6.

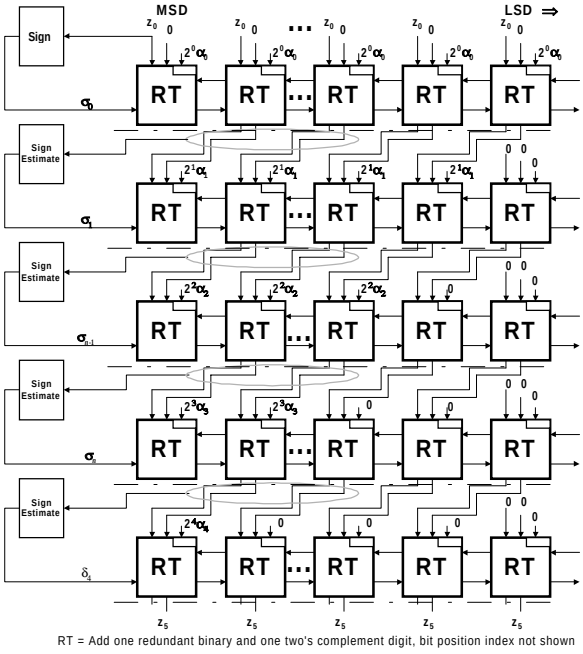

Fig. 6: Section of datapath for standard *z*-iterations

In this figure we identify the lower right triangle of adders (a RT-adder is a 3-2 RBA with sign steering logic) which do not contribute to the sign estimation, as they are adding zeros. Based on this observation we can diagonally prune this array of adders to the structure given in Fig. 7 with no loss in precision or speed. Neglecting the hardware necessary for sign estimation, the gate count has been roughly halved.

4.3 Vectoring mode x- and y-datapath

For redundant adders, the original iterations for *x* and *y* according to Eqs. 1 and 2 are usually modified to

$$x_{i+1} = x_i - m\,\sigma_i\, 2^{-2S(m,i)}\, y_i \qquad (8)$$

$$y_{i+1} = 2(y_i + \sigma_i\, x_i) \qquad (9)$$

Let us consider the *x*-path and take into account that the absolute value of y_i is kept constant to a specific degree by Eq. (9). Then it can be deduced that in the *x*-path about the same situation occurs as for *x* in the rotation mode (Fig. 1). However, with each iteration the amount of zeroes which have to be added to the most significant digit positions increases by two. Thus, referring to Fig. 1, in each iteration two (instead of one) more 4-2 RBA cells can be substituted by 2-2 RBA cells (called R0) beginning from higher order digit positions Thus, even more hardware savings than in the rotation mode are possible. Furthermore, the *x*-iteration can be halted after *n/2*+1 iterations as *n*-bit precision has been achieved /16,20/.

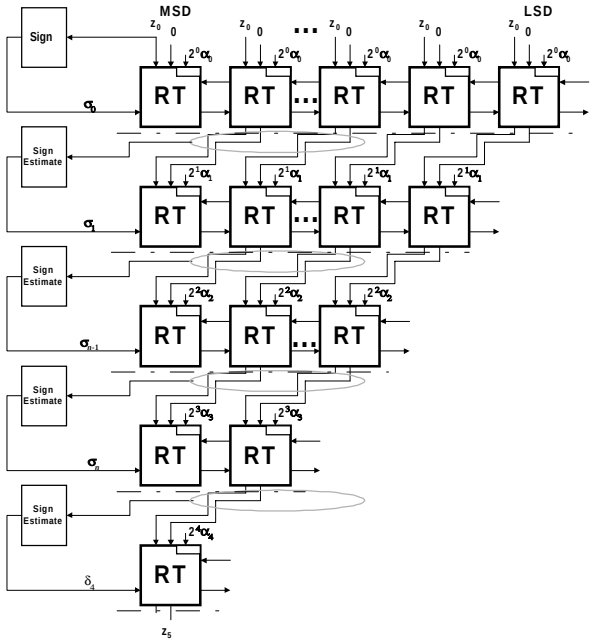

Fig. 7: Reduced datapath for standard *z*-iterations

Considering the *y*-path (Eq. 9) about the same situation as for the *z*-path in rotation mode exists. However, 4-2 RBA's have to be employed. An increasingly number of zeroes can be added, starting from the least significant digit position. The error resulting from not adding the whole length of x_i and y_i does not affect the overall iteration as it is hidden in the guard digits. So we can omit the lower right triangle of 4-2 RBA cells, yielding a reduction in gate count by one half of the standard architecture.

4.4 Vectoring mode *z*-datapath

The *z*-path and the corresponding iteration equation (3) resembles the *x*- and *y*-path in rotation mode. That is, we add constantly decreasing values of the rotation angle $\alpha_{m,i}$ to z_i. The resulting architecture is similar to Fig. 2, yielding an upper right triangle of 3-2 RBA cells (Fig. 4) and a lower left triangle of 2-2 RBA

cells (Fig. 5). Again, the hardware savings are not only due to the simpler adder cells, but also caused by the partly dispensable sign steering logic.

5. EVALUATION

As speed (latency as well as throughput) is not affected by our reductions we concentrate on a discussion of the area requirements. It should be noted that the described modifications are not only fully applicable to redundant constant scale factor methods which employ correcting iterations /3,4,6/ but also partly applicable to the „absolute value“ method given in /8/. Therefore, as unmodified references for comparison we assume the mixed radix 2-4 architecture /6/ and the „absolute value“ method /8/.
As our discussion is based on the assumption that we use correcting iterations we have to establish a base for comparison. We assume that we assimilate enough higher significant digits to guarantee that only each fourth iteration has to be repeated. For a detailed discussion of this topic see, for example, /4,6/. Also, no scaling factor compensation, no sign steering logic, and $m = 1$ is assumed and no guard and overflow digits in order to keep the comparison simple. The standard architecture then requires $1.25n$ iterations with full length datapaths.
A 4-2 RBA, 3-2 RBA, and 2-2 RBA occupy an area equal to 2, 1, and 2/3 full adders, respectively. This assumption can be proved when comparing the transistor counts of the corresponding figures 3,4, and 5.
The improved correcting iterations method /6/ and our architectures employ $\sigma_i \in \{-1,1\}$ for $i \le n/4$, $\sigma_i \in \{-1,0,1\}$ for $n/4 < i \le n/2$ using the approach given in /5/ which avoids correcting iterations, and radix-4 iterations for $i < n/2$.
Our architectures are also supposed to use only one third of the *z*-path and a parallel prediction of all further σ_i in the rotation mode /16,21/. In vectoring mode, as $\alpha_{m,i} = 2^{-S(m,i)}$ for $S(m,i) \ge n/3$ a single summing of the bits $\sigma_i 2^{-S(m,i)}$ is necessary, reducing two third of the *z*-path to one redundant subtraction /16/.

5.1 Rotation mode

The „absolute value“ method needs $7.75n^2$ latches and $5n^2$ equivalent full adders /8/. The mixed radix 2-4 architecture /6/ requires $n/4 * 1{,}25 + n/4 + n/4 = 0.8125n$ iterations. In the first half of the iterations it uses three full length datapaths, in the second half only two datapaths are necessary by using the prediction method. Thus $(3 * 2 * 2.25n/4 + 2 * 2 * n/4) * n = 4.375n^2$ latches (note: two latches per digit) and $(2.25n/4 * (2 * 2 + 1) + n/4 * (2+1))\ n = 3.5625\ n^2$ equivalent full adders are necessary.

Our architecture as depicted in Fig. 8 (t=2.25n/4) has the same number of iterations as the mixed radix approach. We need 2*[2 * (2.25n/4 * n/2 * 3/2 + n/2 * n/4 * ½) + 2/3 * (2.25n/4 * n/2 * ½ + n/2 * n/4 * 3/2)] = 2.375n^2 equivalent full adders for x and y. The z-datapath contains j=n/3+0.25n/4 ≅ 0.4n iteration stages with (normally) 0.4n^2 3-2 RBA's. However, 1/3n * 0.4n * ½ of these adders can be omitted, yielding n^2/3 equivalent full adders for z. On the whole, the suggested architecture needs 2.7n^2 equivalent full adders.
The latch count is the same as in the mixed radix case minus the savings in z, thus 2*(2 * 2.25n^2/4 + 2 * n^2/4 + 0.4n^2 - 1/3n * 0.4n * ½) = 3.9n^2 latches.

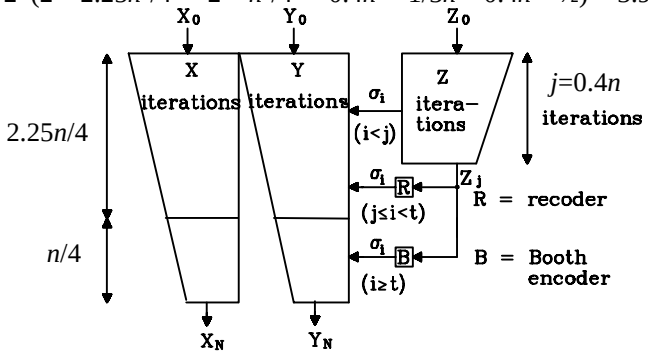

Fig. 8: Combinational area of proposed architecture (latches in x and y must have full length)

5.2 Vectoring mode

The absolute value method needs n^3/6+9.5n^2 latches and 5n^2 equivalent full adders /8/. The mixed radix 2-4 architecture /6/ requires the same hardware as in the rotation mode as it represents a unified architecture. The determination of σ_i is much more complicated due to the necessary prescaling of x and y. To simplify the comparison we build upon the same approach, but do not take into account the necessary hardware.
In our proposal, the x-iteration stops after 2.25n/4 iterations. So the x-datapath can be implemented with 2.25n/4 * n * ½ 4-2 RBA's and 2-2 RBA's each, resulting in 2 * 2.25/4 * ½ + 2/3 * 2.25/4 * ½) = 0.75n^2 equivalent full adders.
The y-path requires 4.25n/4 iterations and on the average n/2 4-2 RBAs yielding about n^2 full adders. In the z-path we have 0.4n iterations with 0.2n^2 3-2 and 2-2 RBS each. So we have n^2/3 full adders and in summary for all datapaths slightly more than 2n^2 full adders.
The latch count is 2 * 2.5n^2 = 5n^2 .

6. CONCLUSION

The results are summarized in Table II.

	/8/		/6/		this proposal	
	adders	latches	adders	latches	adders	latches
Rot.	$5n^2$	$7.75\ n^2$	$3.5\ n^2$	$4.375\ n^2$	$2.7n^2$	$3.9n^2$
Vect.	$5n^2$	$n^3/6+$ $9.5n^2$	$3.5n^2$	$4.375n^2$	$2n^2$	$5n^2$

Table II: Comparison

As can be seen significant savings have been achieved. To assess this results we are developing parametrized VHDL models of these architectures. Currently, we have modeled a rotation mode standard radix-2 redundant addition scheme with full length datapaths and all iterations and a radix-2 architecture with reduced *x*- and *y*- datapaths according to section 4.1 and a *z*-path containing about *n*/3 iterations and the parallel prediction. These architectures have been verified by extensive simulation and synthesized for an external wordlength of 16 bit and mapped onto a 1.0µ CMOS standard cell design.

The overall area reduction obtained is 20%. It should be noted that this result has been achieved at the first run and no optimized library for the improved RBA's was available. We expect further reductions by refining the model and the library. The improvements are valid for implementations in both correcting iteration and absolute value architectures

REFERENCES

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

/2/ T. Noll, „Carry-save architectures for high speed signal processing," Journal of VLSI Signal Processing, vol. 3, pp. 121-140, 1991

/3/ N. Takagi, T. Asada, and S. Yajima, „Redundant CORDIC methods with a constant scale factor for sine and cosine computation," IEEE Trans. on Computers, vol. 40, no. 9, pp. 989-995, Sept. 1991

/4/ J.-A. Lee and T. Lang, „Constant-factor redundant CORDIC for angle calculation and rotation," IEEE Trans. on Computers, vol. 41, no. 8, pp. 1016-1025, Aug. 1992

/5/ D. Timmermann, H. Hahn, B.J. Hosticka, "Low latency time CORDIC algorithms," IEEE Trans. on Computers, vol. 41, no. 8, pp. 1010-1015, 1992

/6/ E. Antelo, J.D. Bruguera, and E.L. Zapata, „Unified mixed radix 2-4 redundant CORDIC processor," IEEE Trans. on Computers, vol. 45, no. 9, pp. 1068-1073, Sept. 1996

/7/ M.D. Ercegovac and T.Lang, „Division and Square Root: Digit-Recurrence, Algorithms and Implementations, Kluwer Academic, 1994

/8/ H. Dawid, H. Meyr, „The differential CORDIC algorithms: constant scale factor redundant implementation without correcting iterations," IEEE Trans. on Computers, vol. 45, no. 3, pp. 307-318, March 1996
/9/ Y.H. Hu, „The quantization effects of the CORDIC algorithm," IEEE Trans. on Circuits and Systems, vol. 40, no. 4, pp. 834-844, 1992
/10/ D. Timmermann, B. Rix, H. Hahn, B.J. Hosticka, "A CMOS floating-point vector-arithmetic unit", IEEE Journal of Solid-State Circuits, vol. 29, no. 5, pp. 634-639, May 1994
/11/ H. Yoshimura, T. Nakanishi, and H. Yamauchi, "50-MHz CMOS geometrical mapping processor," IEEE Trans. on Circuits and Systems, vol. 36, pp. 1360-1364, 1989
/12/ R. Künemund, et. al., „CORDIC processor with carry-save architecture", Proc. ESSCIRC'90, pp. 193-196, 1990
/13/ A. A. J. de Lange, et.al., „An optimal floating-point pipeline CMOS CORDIC processor," Proc. ISCAS'88, Espoo, Finland, pp. 2043-2047, 1988
/14/ A.M. Despain, "Fourier transform computers using CORDIC iterations," IEEE Trans. on Computers, vol.23, no. 10, pp. 993-1001, 1974
/15/ G. Schmidt, et.al., "Parameter optimization of the CORDIC-algorithm and implementation in a CMOS-chip," Proc. EUSIPCO-86, Pt. 2, pp. 1219-1222, Hague, Netherlands, 1986
/16/ D. Timmermann and I. Sundsbø, "Area and latency efficient CORDIC architectures", Proc. ISCAS´92, pp. 1093-1096, San Diego, May 1992
/17/ M.D. Ercegovac and T.Lang, „Simple radix 4 division with operand scaling," IEEE Trans. On Computers, vol. 39, no. 9, pp. 1204-1208, Sept. 1990
/18/ S. Kuninobu, et.al., „Design of high speed MOS multiplier and divider using redundant binary representation", Proc. 8th Symp. Computer Arithmetic, pp. 80-86, New York, 1987
/19/ D. Timmermann, B.J. Hosticka, "Overflow effects in redundant binary number systems," IEE Electronics Letters, vol. 29, no. 5, pp. 440-441, March 1993
/20/ D. Timmermann, H. Hahn, B.J. Hosticka, "Modified CORDIC algorithm with reduced iterations," *Electronics Letters*, vol. 25, no. 15, pp. 950-951, July 1989
/21/ E. Antelo, J.D. Bruguera, J. Villalba, and E.L. Zapata, „Redundant CORDIC rotator based on parallel prediction," Proc. 12th Symp. Computer Arithmetic, pp. 172-179, 1995