Area Minimization of Redundant CORDIC Pipeline Architectures

Andreas Wassatsch, Steffen Dolling, Dirk Timmermann
University of Rostock
Department of Electrical Engineering and Information Technology
Institute of Applied Microelectronics and Computer Science
Richard-Wagner-Str. 31, D-18119 Rostock, Germany
wa11@e-technik.uni-rostock.de

Abstract

The CORDIC algorithm is used in many fields of signal processing for computation of elementary functions. Its main advantages are versatility and simplicity. When implemented in a word parallel pipeline it yields the highest possible throughput. However, this solution is accompanied with increased hardware complexity and chip area requirements. The goal of this paper is to develop redundant CORDIC pipeline architectures yielding very low chip area. The speed does not decrease at all when compared with other proposals. Our novel architectures result in the smallest redundant CORDIC implementation known to the authors. It also exhibits considerably less gate switching activity thus also reducing power consumption.

1. Introduction

For several applications CORDIC processing units have been shown to deliver improved performance compared to more conventional approaches. Because the CORDIC is especially suited to vector rotation operations, it can also be used for many other advanced algorithms, which can be interpreted as generalized vector rotations. The iteration equations of the unified CORDIC algorithm [4] are given by equation (1)-(3).

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

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

$$z_{i+1} = z_i - \sigma_i \alpha_{m,i} \quad (3)$$

Here m denotes the coordinate system, $S(m,i)$ the shift sequence, $\alpha_{m,i}$ the rotation angle, σ_i the rotation direction, and N the number of iterations necessary to obtain a precision of n bit. Using a redundant number representation, i.e. carry-save [6],[1] or signed-digit adders [8],[5],[10] results in the significant advantage of a very low addition time independent of n. Therefore, we subsequently consider only redundant structures. However, redundant architectures also exhibit different drawbacks, i.e. increased storage requirements per digit and a more complex determination of the sign. To be more precise, we can use an estimate of the sign to determine σ_i which is given by the most significant non-zero digit. Hereby, we can avoid the worst-case inspection of all digits to determine the sign exactly. The error which is introduced by this simple estimate can be compensated by doubling some iterations [6],[1],[8],[5] to avoid convergence violations. The amount of double iterations depends on the number of inspected MSD digits and the mode. Recently, a method called Differential CORDIC [2] has been proposed which avoids iteration repetitions at the cost of additional registers. Therefore we use the iteration doubling approach in our architecture.

2. Previous approaches for CORDIC chip area reduction

One important point is the way how scaling factor compensation is achieved, for example by integrating scaling into the iterations or by optimizing the special scaling operations [3]. Further work has been done on minimizing the amount of double iterations. In [8],[5] it is shown that for iterations $i > \frac{n}{2}$ the choice of $\sigma_i \in \{0, 1, -1\}$ does not affect the magnitude of the vector any longer. For $i > \frac{n}{4}$ and $\sigma_i = 0$ modified iterations instead of the standard iterations (1) and (2) avoid double iterations [10]. So iteration repetitions are only necessary in the first $\frac{n}{4}$ iterations, reducing the hardware requirements. About two third of the adders in the z-path can be omitted when applying the methods presented in [10] for rotation mode and [11] for vectoring mode. Further area reductions are feasible when applying Booth recoding of σ_i as has been demonstrated in [1],[10]. After reducing the area requirements by these algorithmic measures further reductions are possible on the bit-level by carefully investigating the required adder and register widths. First results have been reported in [11] for nonredundant architectures. These results have been applied in [9] to redundant CORDIC architectures. The main idea is to partly replace the full 4-2 redundant binary adder (RR), which can be implemented with 42 transistors [7], by a much simpler 3-2 redundant zero (RZ) cell, which is shown in Fig. 1. This can be done in most of the MSD positions, where zeros are added to the corresponding digit positions of x_i and y_i , respectively.

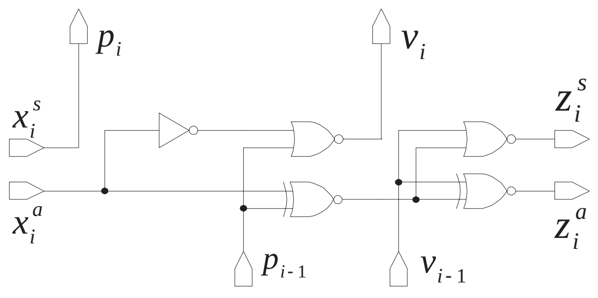
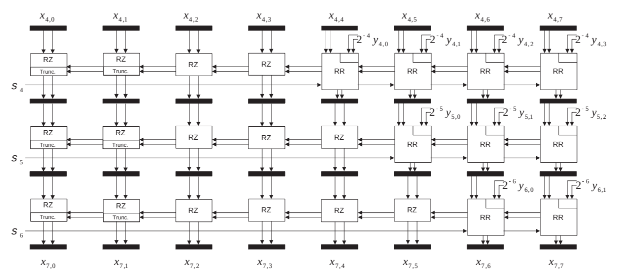


Figure 1. Redundant zero adder-cell (RZ)

Fig. 2 depicts the situation for 3 iterations. Hardware saving are possible as the RZ cells need about half the chip area of RR cells. This architecture is the starting point for our novel bit-level reductions, described in the following.



$x_{4,0}$ $x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$ $x_{4,7}$
$2^{-4}y_{4,0}$ $2^{-4}y_{4,1}$ $2^{-4}y_{4,2}$ $2^{-4}y_{4,3}$
$2^{-5}y_{5,0}$ $2^{-5}y_{5,1}$ $2^{-5}y_{5,2}$
$2^{-6}y_{6,0}$ $2^{-6}y_{6,1}$
RZ
Trunc.
RR
s_4
s_5
s_6
$x_{7,0}$ $x_{7,1}$ $x_{7,2}$ $x_{7,3}$ $x_{7,4}$ $x_{7,5}$ $x_{7,6}$ $x_{7,7}$

Figure 2. Previous reduced x-datapath

3. Area reduction

3.1. Properties of the redundant binary addition

Our consideration starts by the digit vector addition $S = A + B$, where the digits are $s_i, a_i \in \{\bar{1}, 0, 1\}$ and $b_i = 0$ with $i = 0, 1, \ldots, n-1$. When using the addition rule given in [7] the vector S is implicitly recoded during this operation so that three subsequent digits of the vector S are given by

$$\begin{aligned} \{\ldots, s_{i+2}, s_{i+1}, s_i, \ldots\} &\neq \{\ldots, 1, 1, 1, \ldots\} \\ \{\ldots, s_{i+2}, s_{i+1}, s_i, \ldots\} &\neq \{\ldots, \bar{1}, \bar{1}, \bar{1}, \ldots\} \end{aligned}$$

This means that three adjacent digits of the result S can not have the same value $\bar{1}$ or 1 as the maximum count of adjacent digits with the same value unequal zero is two. In the same way this statement is also true if the carry-digit of the lowest significant digit of the addition is unequal zero. When considering the vector $A = \{a_3, a_2, a_1, a_0\}$ as a result of such an addition, we obtain

$$\{(a_3, a_2, a_1), (a_2, a_1, a_0)\} \neq \{(111), (\bar{1}\bar{1}\bar{1})\}$$

and the range of values of A is $-13 \leq A \leq 13$. Computing $S = A + B + c_{in}$ with $B = \{0, 0, 0, 0\}$ and $c_{in} \in \{\bar{1}, 0, 1\}$ in the LSD-position results in $S = \{s_4, s_3, s_2, s_1, s_0\}$ and a range of values $-14 \leq S \leq 14$. The possible combinations are shown in Tab. 1. The marked "-" lines represent combinations of input value A which can not occur in a previous addition as shown before.

	A	S		
		$c_i = 1$	$c_i = 0$	$c_i = \bar{1}$
	3210	43210	43210	43210
-	1111	10000	$1000\bar{1}$	$100\bar{1}0$
-	1110	$100\bar{1}1$	$100\bar{1}0$	$100\bar{1}\bar{1}$
-	$111\bar{1}$	$10\bar{1}10$	$10\bar{1}1\bar{1}$	$10\bar{1}00$
	1101	$10\bar{1}10$	$10\bar{1}1\bar{1}$	$10\bar{1}00$
	1100	$10\bar{1}01$	$10\bar{1}00$	$10\bar{1}0\bar{1}$
	$110\bar{1}$	$10\bar{1}00$	$10\bar{1}0\bar{1}$	$10\bar{1}\bar{1}0$
	$11\bar{1}1$	$1\bar{1}100$	$1\bar{1}10\bar{1}$	$1\bar{1}1\bar{1}0$
	$11\bar{1}0$	$1\bar{1}1\bar{1}1$	$1\bar{1}1\bar{1}0$	$1\bar{1}1\bar{1}\bar{1}$
	$11\bar{1}\bar{1}$	$1\bar{1}010$	$1\bar{1}01\bar{1}$	$1\bar{1}000$
	1011	$1\bar{1}100$	$1\bar{1}10\bar{1}$	$1\bar{1}1\bar{1}0$
	1010	$1\bar{1}1\bar{1}1$	$1\bar{1}1\bar{1}0$	$1\bar{1}1\bar{1}\bar{1}$
	$101\bar{1}$	$1\bar{1}010$	$1\bar{1}01\bar{1}$	$1\bar{1}000$
	1001	$1\bar{1}010$	$1\bar{1}01\bar{1}$	$1\bar{1}000$
	1000	$1\bar{1}001$	$1\bar{1}000$	$1\bar{1}00\bar{1}$
	$100\bar{1}$	$1\bar{1}000$	$1\bar{1}00\bar{1}$	$1\bar{1}0\bar{1}0$
	$10\bar{1}1$	$1\bar{1}000$	$1\bar{1}00\bar{1}$	$1\bar{1}0\bar{1}0$
	$10\bar{1}0$	$1\bar{1}0\bar{1}1$	$1\bar{1}0\bar{1}0$	$1\bar{1}0\bar{1}\bar{1}$
	$10\bar{1}\bar{1}$	$1\bar{1}\bar{1}10$	$1\bar{1}\bar{1}1\bar{1}$	$1\bar{1}\bar{1}00$
	$1\bar{1}11$	01000	$0100\bar{1}$	$010\bar{1}0$
	$1\bar{1}10$	$010\bar{1}1$	$010\bar{1}0$	$010\bar{1}\bar{1}$
	$1\bar{1}1\bar{1}$	$01\bar{1}10$	$010\bar{1}\bar{1}$	$01\bar{1}01$
	$1\bar{1}01$	$01\bar{1}10$	$01\bar{1}1\bar{1}$	$01\bar{1}00$
	$1\bar{1}00$	$01\bar{1}01$	$01\bar{1}00$	$01\bar{1}0\bar{1}$
	$1\bar{1}0\bar{1}$	$01\bar{1}00$	$01\bar{1}0\bar{1}$	$01\bar{1}\bar{1}0$
	$1\bar{1}\bar{1}1$	00100	$0010\bar{1}$	$001\bar{1}0$
	$1\bar{1}\bar{1}0$	$001\bar{1}1$	$001\bar{1}0$	$001\bar{1}\bar{1}$
-	$1\bar{1}\bar{1}\bar{1}$	00010	$0001\bar{1}$	00000
-	0111	01000	$0100\bar{1}$	$010\bar{1}0$
	0110	$010\bar{1}1$	$010\bar{1}0$	$010\bar{1}\bar{1}$
	$011\bar{1}$	$01\bar{1}10$	$01\bar{1}1\bar{1}$	$01\bar{1}00$
	0101	$01\bar{1}10$	$01\bar{1}1\bar{1}$	$01\bar{1}00$
	0100	$01\bar{1}01$	$01\bar{1}00$	$01\bar{1}0\bar{1}$
	$010\bar{1}$	$01\bar{1}00$	$01\bar{1}0\bar{1}$	$01\bar{1}\bar{1}0$
	$01\bar{1}1$	00100	$0010\bar{1}$	$001\bar{1}0$
	$01\bar{1}0$	$001\bar{1}1$	$001\bar{1}0$	$001\bar{1}\bar{1}$
	$01\bar{1}\bar{1}$	00010	$0001\bar{1}$	00000
	0011	00100	$0010\bar{1}$	$001\bar{1}0$
	0010	$001\bar{1}1$	$001\bar{1}0$	$001\bar{1}\bar{1}$
	$001\bar{1}$	00010	$0001\bar{1}$	00000
	0001	00010	$0001\bar{1}$	00000
	0000	00001	00000	$0000\bar{1}$
	$000\bar{1}$	00000	$0000\bar{1}$	$000\bar{1}0$
	$00\bar{1}1$	00000	$0000\bar{1}$	$000\bar{1}0$
	$00\bar{1}0$	$000\bar{1}1$	$000\bar{1}0$	$000\bar{1}\bar{1}$
	$00\bar{1}\bar{1}$	$00\bar{1}10$	$00\bar{1}1\bar{1}$	$00\bar{1}00$
	$0\bar{1}11$	00000	$0000\bar{1}$	$000\bar{1}0$
	$0\bar{1}10$	$000\bar{1}1$	$000\bar{1}0$	$000\bar{1}\bar{1}$
	$0\bar{1}1\bar{1}$	$00\bar{1}10$	$00\bar{1}1\bar{1}$	$00\bar{1}00$
	$0\bar{1}01$	$00\bar{1}10$	$00\bar{1}1\bar{1}$	$00\bar{1}00$
	$0\bar{1}00$	$00\bar{1}01$	$00\bar{1}00$	$00\bar{1}0\bar{1}$
	$0\bar{1}0\bar{1}$	$00\bar{1}00$	$00\bar{1}0\bar{1}$	$00\bar{1}\bar{1}0$
	$0\bar{1}\bar{1}1$	$0\bar{1}100$	$0\bar{1}10\bar{1}$	$0\bar{1}1\bar{1}0$
	$0\bar{1}\bar{1}0$	$0\bar{1}1\bar{1}1$	$0\bar{1}1\bar{1}0$	$0\bar{1}1\bar{1}\bar{1}$
-	$0\bar{1}\bar{1}\bar{1}$	$0\bar{1}010$	$0\bar{1}01\bar{1}$	$0\bar{1}000$
-	$\bar{1}111$	00000	$0000\bar{1}$	$000\bar{1}0$
	$\bar{1}110$	$000\bar{1}1$	$000\bar{1}0$	$000\bar{1}\bar{1}$
	$\bar{1}11\bar{1}$	$00\bar{1}10$	$00\bar{1}1\bar{1}$	$00\bar{1}00$
	$\bar{1}101$	$00\bar{1}10$	$00\bar{1}1\bar{1}$	$00\bar{1}00$
	$\bar{1}100$	$00\bar{1}01$	$00\bar{1}00$	$00\bar{1}0\bar{1}$
	$\bar{1}10\bar{1}$	$00\bar{1}00$	$00\bar{1}0\bar{1}$	$00\bar{1}\bar{1}0$
	$\bar{1}1\bar{1}1$	$0\bar{1}100$	$0\bar{1}10\bar{1}$	$0\bar{1}1\bar{1}0$
	$\bar{1}1\bar{1}0$	$0\bar{1}1\bar{1}1$	$0\bar{1}1\bar{1}0$	$0\bar{1}1\bar{1}\bar{1}$
	$\bar{1}1\bar{1}\bar{1}$	$0\bar{1}010$	$0\bar{1}01\bar{1}$	$0\bar{1}000$
	$\bar{1}011$	$0\bar{1}100$	$0\bar{1}10\bar{1}$	$0\bar{1}1\bar{1}0$
	$\bar{1}010$	$0\bar{1}1\bar{1}1$	$0\bar{1}1\bar{1}0$	$0\bar{1}1\bar{1}\bar{1}$
	$\bar{1}01\bar{1}$	$0\bar{1}010$	$0\bar{1}01\bar{1}$	$0\bar{1}000$
	$\bar{1}001$	$0\bar{1}010$	$0\bar{1}01\bar{1}$	$0\bar{1}000$
	$\bar{1}000$	$0\bar{1}001$	$0\bar{1}000$	$0\bar{1}00\bar{1}$
	$\bar{1}00\bar{1}$	$0\bar{1}000$	$0\bar{1}00\bar{1}$	$0\bar{1}0\bar{1}0$
	$\bar{1}0\bar{1}1$	$0\bar{1}000$	$0\bar{1}00\bar{1}$	$0\bar{1}0\bar{1}0$
	$\bar{1}0\bar{1}0$	$0\bar{1}0\bar{1}1$	$0\bar{1}0\bar{1}0$	$0\bar{1}0\bar{1}\bar{1}$
	$\bar{1}0\bar{1}\bar{1}$	$0\bar{1}\bar{1}10$	$0\bar{1}\bar{1}1\bar{1}$	$0\bar{1}\bar{1}00$
	$\bar{1}\bar{1}11$	$\bar{1}1000$	$\bar{1}100\bar{1}$	$\bar{1}10\bar{1}0$
	$\bar{1}\bar{1}10$	$\bar{1}10\bar{1}1$	$\bar{1}10\bar{1}0$	$\bar{1}10\bar{1}\bar{1}$
	$\bar{1}\bar{1}1\bar{1}$	$\bar{1}1\bar{1}10$	$\bar{1}1\bar{1}1\bar{1}$	$\bar{1}1\bar{1}00$
	$\bar{1}\bar{1}01$	$\bar{1}1\bar{1}10$	$\bar{1}1\bar{1}1\bar{1}$	$\bar{1}1\bar{1}00$
	$\bar{1}\bar{1}00$	$\bar{1}1\bar{1}01$	$\bar{1}1\bar{1}00$	$\bar{1}1\bar{1}0\bar{1}$
	$\bar{1}\bar{1}0\bar{1}$	$\bar{1}1\bar{1}00$	$\bar{1}1\bar{1}0\bar{1}$	$\bar{1}1\bar{1}\bar{1}0$
-	$\bar{1}\bar{1}\bar{1}1$	$\bar{1}0100$	$\bar{1}010\bar{1}$	$\bar{1}01\bar{1}0$
-	$\bar{1}\bar{1}\bar{1}0$	$\bar{1}01\bar{1}1$	$\bar{1}01\bar{1}0$	$\bar{1}01\bar{1}\bar{1}$
-	$\bar{1}\bar{1}\bar{1}\bar{1}$	$\bar{1}0010$	$\bar{1}001\bar{1}$	$\bar{1}0000$

Table 1: $S = A + B + c_{in}$

The case $s_4 \neq 0$ is only possible for pseudo-overflows, which can be recoded with $\bar{1}1 \to 0\bar{1}$, $1\bar{1} \to 01$, $\bar{1}01 \to 0\bar{1}\bar{1}$, $10\bar{1} \to 011$ and so on. From this it follows that in all possible combinations $s_4 = 0$ and $\{(s_3, s_2, s_1), (s_2, s_1, s_0)\} \neq \{(111), (\bar{1}\bar{1}\bar{1})\}$.

The addition $S = A + B + c_{in}$ can be done with a 4-digit-adder, wich uses an RZ-adder for the processing of the LSD digits and c_{in} corresponding to Fig. 1. The addition of the MSD's and suppression of pseudo-overflows can be done with the redundant-zero-0-cell (RZ0), shown in Fig. 3, and a carry-absorber-cell (CAB) (Fig. 4). Now we can use these cells in a pipeline structur for the implementation of the iteration equation $A_{i+1} = A_i + 2^{(i+4)} B_i$

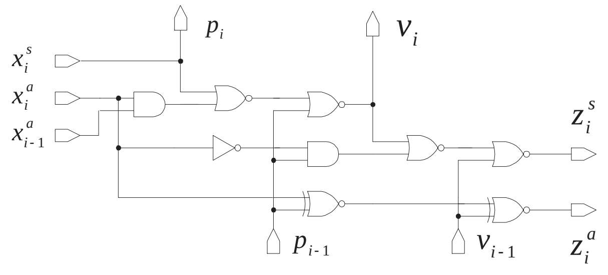


Figure 3. Redundant zero 0 cell (RZ0)

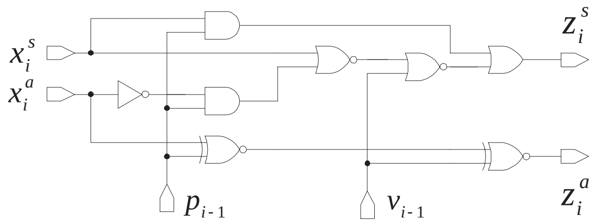


Figure 4. Carry absorber cell (CAB)

with $i = (0, 1, \ldots, N-1)$. A_i and B_i are the digit-vectors with the vector elements $a_{i,j}, b_{i,j} \in \{\bar{1}, 0, 1\}$ and $j = 0, 1, \ldots, n-1$. For the most significant digits of the input vector A_0 we know that $a_{0,n-1}a_{0,n-2}a_{0,n-3} \neq \{(111), (\bar{1}, \bar{1}, \bar{1})\}$. To perform the required addition of the most significant digits $a_{i,n-1-i}a_{i,n-2-i}a_{i,n-3-i}a_{i,n-4-i}$ we use the above mentioned 4-digit adder and for the following less significant digits the ordinary 4-2-RR-cell. This can be done in a scheme given in Fig. 5. In the iteration

$$
\begin{array}{ccccccc}
 & a_{0,n-1} & a_{0,n-2} & a_{0,n-3} & a_{0,n-4} & a_{0,n-5} & a_{0,n-6} \\
+ & 0 & 0 & 0 & 0 & b_{0,n-1} & b_{0,n-2} \\
\hline
s_4 & s_3 & s_2 & s_1 & s_0 & & \\
 & \mathbf{a_{1,n-1}} & a_{1,n-2} & a_{1,n-3} & a_{1,n-4} & a_{1,n-5} & a_{1,n-6} \\
+ & & 0 & 0 & 0 & 0 & b_{1,n-1} \\
\hline
 & s_4 & s_3 & s_2 & s_1 & s_0 & \\
 & \mathbf{a_{1,n-1}} & \mathbf{a_{2,n-2}} & a_{2,n-3} & a_{2,n-4} & a_{2,n-5} & a_{2,n-6} \\
+ & & & 0 & 0 & 0 & 0 \\
\hline
 & & s_4 & s_3 & s_2 & s_1 & s_0 \\
 & \mathbf{a_{1,n-1}} & \mathbf{a_{2,n-2}} & \mathbf{a_{3,n-3}} & a_{3,n-4} & a_{3,n-5} & a_{3,n-6}
\end{array}
$$

Figure 5. Addition scheme (bold face = fixed values)

index 1 the element $a_{1,n-1}$ can not be influenced from a overflow out of the less significant digits because $s_4 = 0$. Therefore all following iterations can use $a_{i,n-1} = a_{1,n-1}$. Correspondingly all the adders in position $a_{i,n-1}$ can be omitted. The digits of vector A will become fixed in the same way beginning from the MSD with increasing iteration index. This results in a growing saving of adder cells.

3.2. Rotation mode

The above mentioned structure is usable in the datapath of X and Y and results in a considerable simplification of the architecture given in [9]. The basic construction of three subsequent iteration stages is shown in Fig. 6. Obviously the adder cells for the MSD's in the X- and Y-path can be reduced beginning with iteration index 4. The RZ0 recodes

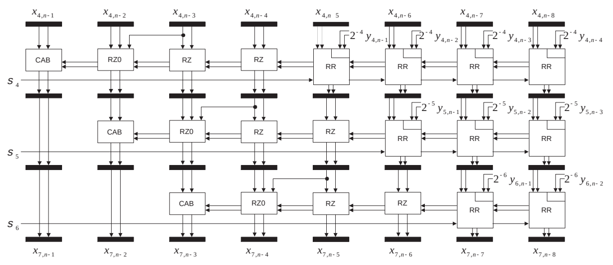


Figure 6. Novel area reduced x-datapath by using special adder cells

the value of x_i and y_i at the specific digit positions and CAB absorbs any possible carry out from these positions. Thus, in the leading digit positions the adders can be omitted completely, resulting in a significant overall hardware reduction. As the addition time of these special cells is not slower than for a RR cell no speed penalty has to be paid.

Special precaution has to be taken in double iterations, because we have to shift the X- and Y-vectors by the same value 2^{-i} as in the previous iteration. With this we face the problem that the above outlined principle can not be applied. There are two ways to solve this problem. We can use in the double iteration stages the entire adder rows according to [9]. Or we can delay the first use of the special 4-digit adder cells by k iterations, where k is the number of double iterations.

As an alternative for the 4-digit adder we have developed a 3-digit adder which serves for the same purpose. The main idea is demonstrated in Fig. 7 for three CORDIC-iteration stages. It is based upon the following consideration. The digit vector A is given by $A = \{a_2, a_1, a_0\}$ with a value $-6 \leq A \leq 6$. The vector A is recodeable in such a way that the value of subvector $A_1 = \{a_{1,2}, a_{1,1}\}$ is given by $-2 \leq A_1 \leq 2$. Now we create the vector $A_2 = \{a_{2,2}, a_{2,1}, a_{2,0}\}$ with the assignment $a_{1,2} = a_{2,2}$, $a_{1,1} = a_{2,1}$ and $a_{2,0} \in \{\bar{1}, 0, 1\}$. This vector A_2 has a value $-5 \le A_2 \le 5$. After the addition of a carry-digit $c_{in} \in \{\bar{1}, 0, 1\}$ to the vector A_2 in the LsD position it has a value $-6 \le A_2 \le 6$. With this procedure it is possible to perform the carry absorbtion from a less significant digit position and simultaneously recoding the result vector for preparing the carry absorbtion in the next iteration. This can be implemented with a recoding unit (REC) and the same carry-absorber cell (CAB) as before. The advantage of this architecture is the saving of RR-adder cells beginning with the iteration index 3 (Fig. 7). A disadvantage is a small increase in computing time compared with the first method. The handling of double iterations is the same as before. A similar method has been shown in [5] for on-the-fly-conversion of the most significant digits into a binary representation.

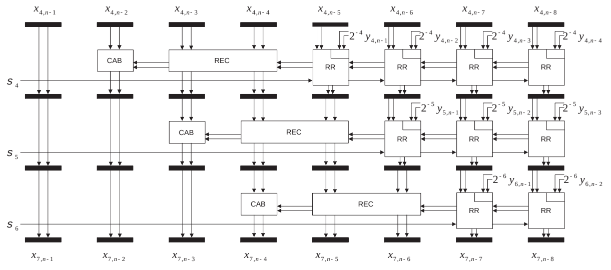


Figure 7. Novel area reduced x-datapath by using 3 digit adder cells

In the standard architecture dynamic power is consumed due to unnecessary switching activity at the outputs of the RR cells in the leading digit positions. This is caused by the transfer digits from the lower digit positions. The CAB cells absorb all transfer digits so this undesirable switching activity is suppressed. A dynamic power reduction proportional to the number of saved adder cells is obtained. Static power consumption is decreased as well.

In the z-datapath Equ.(3) is modified to $z_{i+1} = 2(z_i - \sigma_i 2^{S(m,i)} \alpha_{m,i})$ [5]. Thus the sign estimate can be performed by inspecting four leading MSDs of z_i. The addition requires 3-2 redundant binary (RB) adder cells. As $\alpha_{m,i}$ is shifted one position to the left in each iteration we do not need full-width adders. In fact, each z-iteration needs one less adder and register cell, starting from the least significant digit. Any iteration exeeding $i = \frac{n+1}{3}$ needs only registers to store $z_{(n+1)/3}$ as the σ_i can be predicted from the bits of $z_{(n+1)/3}$ [10]. The novel rotation mode architecture results in the least chip area consuming CORDIC pipeline architectures known to the authors. The significant reduction of adder cells results in a situation where the impact of pipeline registers dominates the chip area. Fig. 8 depicts the overall chip area reductions.

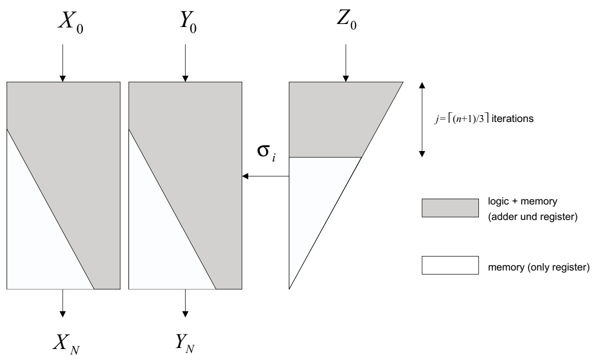

X_0
Y_0
Z_0
σ_i
$j = \lceil (n+1)/3 \rceil$ iterations
logic + memory (adder und register)
memory (only register)
X_N
Y_N

Figure 8. Chip area in rotation mode

3.3. Vectoring mode

In order to use redundant adders and the sign estimation technique, equations (1) and (2) are modified according to [5], yielding

$$x_{i+1} = x_i - m\sigma_i 2^{-2S(m,i)} y_i \quad (4)$$

$$y_{i+1} = 2(y_i + \sigma_i x_i) \quad (5)$$

Correspondingly, constantly decreasing rotation angles $\alpha_{m,1}$ are added to z_i, yielding the same situation as for the x- and y-path in rotation mode. Consequently, the same approach for eliminating an increasing number of leading adder cells in each iteration can be employed. In the x-path, y_i is shifted two digit positions to the right in each iteration. So the iteration can be stopped for $j \ge \lceil \frac{n}{2} \rceil$ and only registers are required in the remaining x-iterations. In addition, the hardware savings in the leading digit positions are larger than in the rotation mode due to the larger shift. The situation in y resembles that of z in rotation mode due to an increasing number of unnecessary adders and registers starting from the LSD.

4. Conclusion

The paper deals with CORDIC pipeline area reduction without loss of throughput. We first studied the carry behavior of additions of two operands where one operand is increasingly shifted to the LSD. Thus we were able to devise special adder cells which result in incremental reduction of adders required in each iteration. This yields significant area savings due to the reduced number of adders. We have developed VHDL models and synthesized different layouts (Fig. 9) to assess our architectures. Using a

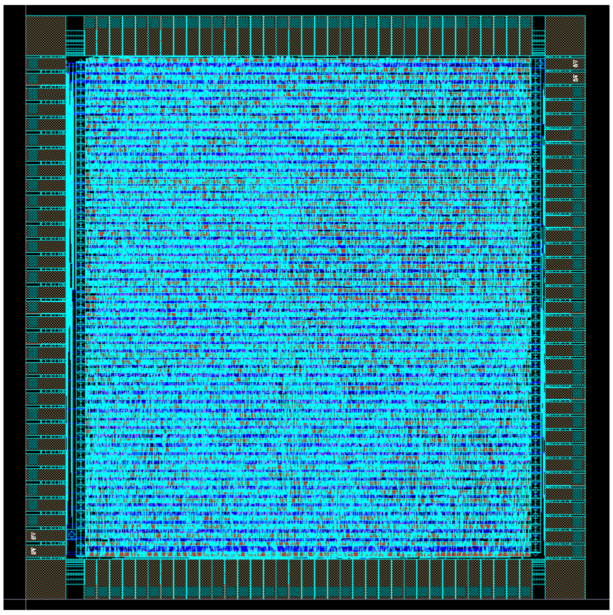


Figure 9. Layout with es2 standard cells of our redundant CORDIC (rotation-mode)

1.0μm CMOS standard cell technology and an external precision of 15 bit we obtained area reductions by more than 12% compared with previously proposed architectures. Because the RZ0, RZ, CAB and RR cells have been assembled from logic cells found in the library the results can be greatly improved when using optimized full custom cells.

implementation	over all area
redundant CORDIC	$50.03mm^2$
reduced architecture[9]	$41.18mm^2$
new reduced architecture	$36.58mm^2$

Table 2: Comparison of results

References

[1] E. Antelo, J. Brugera, and E. Zapata. Unified mixed radix 2-4 redundant cordic processor. *IEEE Trans. on Computers*, 45(9):1068–1073, Sept. 1996.

[2] H. Dawid and H. Meyr. The differential cordic algorithms: constant scale factor redundant implementation without correcting iterations. *IEEE Trans. on Computers*, 45(3):307–318, March 1996.

[3] G. Schmidt, et al. Parameter optimization of the cordic-algorithm and implementation in a cmos-chip. In *Proc. EUSICO-86, B. 2*, pages 1219–1222, Hague, Netherlands, 1986.

[4] J.S.Walther. A unified algorithm for elementary functions. In *Proc. of Spring Joint Computer Conference*, pages 379–385, 1971.

[5] J.-A. Lee and T. Lang. Constant-factor redundant cordic for angle calculation and rotation. *IEEE Trans. on Computers*, 41(8):1016–1025, Aug. 1992.

[6] T. Noll. Carry-save architectures for high speed signal processing. *Journal of VLSI Signal Processing*, 3:121–140, 1991.

[7] S. Kuninobu et.al. Design of high speed mos multiplier and divider using redundant binary representation. In *Proc. 8th Symp. Computer Arithmetic*, pages 80–86, New York, 1987.

[8] N. Takagi, T. Asada, and S. Yajima. Redundant cordic methods with a constant scale factor for sine and cosine computation. *IEEE Trans. on Computers*, 40(9):989–995, Sept. 1991.

[9] D. Timmermann and S. Dolling. Unfolded redundant cordic vlsi architectures with reduced area and power consumption. In *VLSI'97*, Gramado, Brasilien, Aug. 1997.

[10] D. Timmermann, H. Hahn, and B. Hosticka. Low latency time cordic algorithms. *IEEE Trans. on Computers*, 41(8):1010–1015, Sept. 1992.

[11] D. Timmermann and I. Sundsbø. Area and latency efficient cordic architectures. In *Proc. ISCAS'92*, pages 1093–1096, San Diego, May 1992.