

# Fault-Tolerant High-Performance Cordic Processors

Jae-Hyuck Kwak<sup>§</sup>, Vincenzo Piuri<sup>\*1</sup>, Earl E. Swartzlander, Jr.<sup>§</sup>

<sup>§</sup> Department of Electrical and Computer Engineering, University of Texas at Austin  
Austin, TX 78712, USA

<sup>\*</sup> Department of Electronics and Information, Politecnico di Milano  
Piazza L. da Vinci 32, 20133 Milano, Italy

## Abstract

*This paper presents a low-cost approach to concurrent error detection in a high-performance CORDIC processor based on a conditional-sum scheme. The specific characteristics of the CORDIC computation and the processor allow fault detection at a low increase in circuit complexity and latency. The detection scheme is based on use of the AN codes for the arithmetic part and on duplication of the rotation direction generators. Granular-pipelining has been applied to provide a variety of different performance tradeoffs, all with the same fault detection capabilities.*

## 1: Introduction

The CORDIC algorithm is an effective iterative technique for vector rotation and for evaluating more than a dozen of elementary functions [1]. Several dedicated VLSI architectures have been proposed in the literature to achieve high performance in real-time signal-processing applications. Many solutions are based on use of the redundant data representations [2, 3]. These techniques can be effectively applied only if the rotation directions can be estimated in advance, so that the high performance of the arithmetic paths is preserved. To estimate the direction, sign detection is applied either to the angle (*rotation mode*) or to the y coordinate (*vectoring mode*), and it has been usually done by inspecting a few MSDs (most significant digits) [4]. However, such redundant systems have resulted either in an increased number of iterations [3], difficulties in pipelining [5], or high hardware complexity. An efficient solutions to these problems have been proposed, which were based on a conditional sum scheme to implement an effectively overlapped structure [6]. Granular pipelining can be effectively used for a good tradeoff in the conflicting performance goals (throughput, latency, and hardware complexity) [7, 8].

On the other hand, in many mission-critical applications (e.g., in aerospace and telecommunication), fault tolerance is mandatory. Whenever decisions, which are critical for the safety either of personnel or equipment, are taken on the basis of CORDIC processor outputs, these data must be correct or, at least, erroneous results must be identified so that proper actions can be taken to avoid them. Similar needs occur when maintenance is difficult or impossible. On-line checking of all results can be accomplished by concurrent error detection. Several techniques are available, based on hardware redundancy, data coding, or time redundancy [9]. In arithmetic data paths, arithmetic codes such as AN codes and residue codes have been proved effective [10].

<sup>1</sup> The portion of this research activity performed by Vincenzo Piuri has been supported by the MADESS II Project funded by the National Research Council of Italy.

In this paper, we present a family of high-performance CORDIC processors with concurrent error detection ability for vectoring. The proposed architectures are based on the unpipelined structure for redundant CORDIC and the novel prediction scheme introduced in [6]. In the present paper, the performance of the basic structure is enhanced by varying the granularity of pipelining. We introduce an efficient use of the  $3N$  code and modular redundancy for concurrent error detection in CORDIC processing, by adopting the single-bit error model to describe the effects of faults on the computation. Even though cascades of additions with reconvergent paths may lead to aliasing [11], we show that the specific operations performed by an unfolded CORDIC processor can still satisfy the conditions to detect any single error in the arithmetic data paths (including pipeline registers) by checking only the final processor outputs. On the other hand, since the basic structure is composed of carry-save adders and the carry-propagate adder in the final stage can be pipelined, longer coded operands do not reduce the throughput at all, i.e., error checking does not affect the performance. Only the rotation direction generators need to be duplicated since their operations are not protected by the  $3N$  code. Fault coverage of the proposed approach, hardware redundancy, and performance are finally evaluated.

## 2: The basic CORDIC architecture

The unified CORDIC algorithm [1,2] is defined by

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

where  $m$  denotes a coordinate system (+1: circular, 0: linear, or -1: hyperbolic),  $\sigma_i$  the rotation direction, and  $S(m,i)$  the iteration sequence.  $\alpha_{m,i}$  is the rotation angle defined as

$$\alpha_{m,i} = (1/\sqrt{m}) \cdot \tan^{-1}(\sqrt{m} \cdot 2^{-S(m,i)})$$

In rotation mode, the processor performs planary vector rotation, while in vectoring mode the magnitude and the angle of an initial vector are computed.  $\sigma_i$  is thus determined by

$$\sigma_i = \begin{cases} \text{sign}(z_i), & \text{for rotation mode } (z_n \rightarrow 0) \\ \text{sign}(-x_i \cdot y_i), & \text{for vectoring mode } (y_n \rightarrow 0) \end{cases} \quad (2)$$

Estimation of the rotation direction in vectoring mode is usually based on the parameter  $w_i$ , defined as  $w_i = 2^i \cdot y_i$  [2]. Therefore, the CORDIC iteration equations become

$$\begin{aligned} x_{i+1} &= x_i - \sigma_i \cdot 2^{-2i} \cdot w_i \\ w_{i+1} &= 2(w_i + \sigma_i \cdot x_i) \\ z_{i+1} &= z_i - \sigma_i \alpha_i \end{aligned} \quad (3)$$

To predict  $\sigma_i$ , the estimated value of  $w_i$ ,  $\hat{w}_i$ , is first produced by truncating  $w_i$  to  $t$  MSDs [4];  $\sigma_i$  is then given by the sign of  $\hat{w}_i$ .

In [6], an efficient CORDIC architecture is presented, which fully overlaps the computation along the arithmetic data paths and the rotation direction selection for maximum exploitation of the hardware throughput capabilities. Computation in the  $x$  and  $w$  data flows depends in each iteration on the rotation direction evaluated on the output of the previous iteration. Therefore, evaluation of the subsequent values for  $x$  and  $w$  should be performed alternately with the evaluation of the rotation direction. Circuits are thus used not continuously: arithmetic circuits along the data flows for one CORDIC iteration must wait for the rotation direction circuit and, then, the rotation direction circuit related to the subsequent iteration must wait for the completion of the data flow computation.

To enhance the overall performance, these waiting periods need to be removed. The architecture proposed in [6] achieves such a goal by adopting the basic principle of the conditional sum mechanism. One CORDIC computation on the data flows  $x$  and  $w$  is performed for each of the two possible values of the rotation direction. During the same time interval, the rotation direction is also generated. Finally, the direction is used to select the actual outputs of the data flows  $x$  and  $w$  to be delivered to the subsequent stage. To implement the addition/subtraction in each CORDIC stage, any adder/subtractor architecture can be used. To minimize the latency of the operation, in [6] redundant arithmetic is adopted; any detailed structure can be adopted to implement the full adders for the adder/subtractors (Conditional-Sum Adder/Subtractors, CSAS) in the data paths, provided that the detection ability of the  $3N$  code is preserved. A carry-propagate adder is needed at the end of CORDIC operations to propagate carries and transform the redundant representation in the conventional binary one.

The unfolded CORDIC architecture is shown in Figure 1: each CORDIC iteration is mapped onto a dedicated architecture stage to maximize the system performance and to guarantee the detection ability of the  $3N$  code. The design of one stage of such an approach is shown in Figure 2.

The intrinsic uniformity of this structure and the unidirectional data flow allow for introducing pipelining at different granularities to enhance the throughput. Full pipelining is achieved when data are latched at the output of every stage [8]. Meanwhile, in order to save the circuit complexity, some adjacent stages of the basic architecture can be combined into a single stage. The clock cycle must include the latency of the longest chains of adders and multiplexers in the stages plus the setup and hold time of the pipeline register as well as the clock skew. Pipelining with intermediate granularities between no pipelining and full pipelining gives intermediate tradeoffs and more alternatives to achieve a better match with the application. The finer the granularity, the higher the throughput, the circuit complexity and the latency.

### 3: The concurrent error detection approach

As shown in the previous section, the basic architecture of the CORDIC processor consists of arithmetic data paths computing  $x$ ,  $w$ , and  $z$ , and the rotation direction generators. The data paths perform cascades of addition/subtractions, with reconvergent data paths between the  $x$  and the  $w$  data flows. The rotation direction generators perform logic operations on the input data. The different nature of the operations performed by these parts requires different techniques to minimize the overheads (both in hardware and latency) and to assure high detection ability for the adopted error model. Since we aim to a high abstraction view of the system operation, we consider a behavioral error model for each component, by focusing on single-bit errors. This restriction is widely adopted in the literature and can be easily satisfied by an appropriate design of the individual components.

To provide fault tolerance for the arithmetic data paths, arithmetic codes can be efficiently used [10]. Residue codes are frequently preferred when high performance is required, but  $AN$  codes limit the circuit complexity increase. For our redundant CORDIC architecture the use of carry-save addition in each stage allows use of the  $AN$  codes without throughput reduction.

The irregularity of the structure of the rotation direction generators and the presence of non-arithmetic functions in them make duplication with comparison [9] more practical for these circuits. Note that the complexity of such circuits is very low with respect to the adders in the three arithmetic data paths. Consequently, duplication is a viable solution.

The resulting architecture with concurrent error detection ability is shown in Figure 3, for the case of full pipelining. It is designed to be able to detect single errors in the final outputs.

The structure resembles very much the basic architecture without fault tolerance features. The arithmetic data paths (including both the adder/subtractors, the multiplexers for the conditional-sum scheme, and the final carry-propagation adder) have the same basic structure without fault tolerance abilities, but have to deal with a higher number of operands' bits. The rotation direction generators are duplicated and compared for error checking. Comparators are assumed to be self-checking.

Due to the peculiar nature of the CORDIC computation, as discussed later, single errors in one arithmetic unit are propagated towards the final outputs of the arithmetic data paths. This does not happen in the general case of reconvergent data paths [12]. Checking the intermediate results of the arithmetic data paths can thus be avoided without impairing the overall detection abilities. Consequently, the hardware overhead for data path checking is greatly reduced.

The granularly pipelined architectures or the unpipelined structure are similar to the fully pipelined scheme, but have fewer pipeline stages. Error detection ability is identical.

#### 4: Protection of the arithmetic data paths

Let's now describe more in detail how the data paths  $x$ ,  $w$ , and  $z$  operate in the redundant scheme. To protect the adder/subtractors in each stage of Figure 2, we encode the  $x$ ,  $w$ , and  $z$  operands with the  $3N$  code.  $AN$  codes are non-systematic codes in which each nominal datum  $N$  is replaced by its coded representation  $C(N)$  via the linear transformation  $C(N) = A \times N$ , where the integer constant  $A$  is the *code generator*. Arithmetic units are not modified by the adoption of the coded data: the only difference with respect to the nominal structure is therefore the increased number of bits. The number of additional bits is  $\lceil \log_2 A \rceil$ , i.e., 2 bits for the  $3N$  code. The coding unit is a simple multiplier, but its structure can be minimized since  $A$  is a constant known in advance. It has been shown [10] that  $A=3$  gives satisfactory detection capacity for all arithmetic units in which single faults induce single errors that can be represented as  $\pm 2^j$ . Decoding of results involving only additions and subtractions can be performed by applying the linear inverse-transformation  $N = C(N)/A$ .

Let us analyze the error detection characteristics of the hardware components in the arithmetic data paths when the  $3N$  code is adopted, and the related hardware supports.

In the unfolded CORDIC architectures, the shift operations can be realized by hardwired shifters since the number of positions for which the operands must be moved is fixed at each CORDIC stage. Hardwired shifters do not contain any active device and, therefore, behave as any other interconnection when faults are envisioned.

Hardwired shifters as well as all other interconnection wires become 2-bit larger to accommodate the longer operands. Under the single error assumption, wiring in the arithmetic data flows is automatically protected whenever single faults appear as single-bit errors. These errors can in fact be viewed as errors in the connected units.

To support the  $3N$  code, two additional full adders must be introduced in each adder/subtractor of the CSAS blocks to deal with the 2 additional bits of the longer operands. The introduction of these full adders does not imply any delay in the pipeline stage because carries are not propagated in the carry-save structure. The half and full adders can be easily designed to guarantee the single-error property (e.g., see [11], although any other - possibly optimized - design which guarantees such a property can be used). Any single error either produced by a fault occurred within a stage or arriving from the previous stage appears as a single error in the output of at least one adder. Thus, any of these single errors can be detected by verifying that the outputs of the adders are divisible by 3.

The multiplexers are also extended to deal with the 2-bit longer data. Any single error generated by the multiplexer affects only one bit and, consequently, appears as a single

erroneous bit of the operand at the input of the subsequent arithmetic units. Such an operand will not be divisible by 3, thus propagating the error towards the final output as it was an error in the arithmetic unit itself. Each multiplexer operates independently on every bit: a fault at the selection input of the single-bit portion of the multiplexer may affect only that bit. If the wrong data input is selected, an error will be propagated only if the erroneous input bit is different from the expected one. Under the single fault assumption, only one selection input of a multiplexer can be faulty and, consequently, only one output bit of the multiplexers can be erroneous. This still satisfies the single error condition that assures the single error detection ability of the  $3N$  code. Any single error arriving at a multiplexer input is propagated to the output if that input is selected by the multiplexer control signal (if the other input is selected, the error on the first one is masked). Therefore, checking the divisibility by 3 of the outputs of all multiplexers detects not only possible single errors due to a fault in the multiplexers, but also errors propagated by the CSAS blocks connected to the multiplexers. Consequently, for single errors generated or propagated by the CSAS blocks and the multiplexers, we do not need to check the outputs both of the CSAS blocks and the multiplexers. However, note that not checking the CSAS outputs may increase the error latency. In fact, under the usual hypothesis of uniform distribution of the CORDIC inputs, the error latency typically lasts very few iterations. If the mean time between failures is long enough, we have sufficient time to detect the presence of the first error and avoid possible subsequent multiple-error masking.

At the end of the CORDIC computation, the redundant data representation due to the carry-save units used to speed up the computation needs to be transformed back into the conventional binary representation by propagating the carries. A carry-propagating adder is used in the arithmetic data paths  $x$ ,  $w$ , and  $z$ . These adders can be designed to ensure the single-bit error assumption (e.g., [11]). Consequently, from the point of view of error detection, these adders can be protected by the same  $3N$  code. In granularly-pipelined architectures high throughput is maintained by pipelining the carry-propagation adders.

Encoding a CORDIC input  $N$  in the  $3N$  code can be performed by addition as  $2N+N$  [10]. In our redundant CORDIC architecture, this can be accomplished without any additional hardware. In fact, since the first carry-save stage receives only the sum bits and no carry bit, encoding can be directly performed in the first stage by presenting the CORDIC inputs at the sum and carry of each CSAS block. Error detection abilities and operation for encoding coincide therefore with those of the arithmetic data paths in any other CORDIC stage, as discussed at the beginning of this section.

Final decoding and checking of the arithmetic data flows are performed by dividing the results by 3: the quotient and the remainder are the decoded result and the error syndrome, respectively. This operation can be implemented as a subtraction [10], and possibly, merged with the final carry propagation. Error checking is finally done by verifying that all the three most-significant bits generated by this divider are null: if at least one is not zero, an error is detected [13]. High throughput can be maintained by pipelining this subtractor [14].

Now let's consider the cascade of stages in the arithmetic data flows  $x$ ,  $w$ , and  $z$ . If these data flows are tree-like structured (i.e., without reconvergent paths or cycles), any single error that occurs in such data paths should be propagated to the final outputs of the CORDIC processor. The  $z$  path satisfies this constraint since it is linear and contains only adder/subtractors. However, this is not the case for the  $x$  and  $w$  data flows, since data are exchanged between them at each CORDIC iteration. According to [12], even without multiplications, the presence of reconvergent paths in a generic data flow makes aliasing possible for the adopted coding technique. To avoid aliasing, we need to check the data before they converge in an arithmetic unit. In the CORDIC processor, this could be accomplished with a checker for the  $3N$  code at each multiplexer (or pipeline register) output.

If no shifting is present along the  $x$  and  $w$  data paths, a single-bit error of  $\pm 2^f$  in the  $f$ -th bit of the  $x$  multiplexer output at the  $i$ -th CORDIC iteration will result in the same single-bit error in the output both of the  $x$  and  $w$  flows at the  $(i+1)$ -th iteration. This error will thus be present in both inputs of the  $(i+2)$ -th iteration. Consequently, the  $x$  and  $w$  outputs are affected by the single-bit error  $\pm 2^{f+1}$ . Such error will be propagated for any number of CORDIC iterations and then detected at the final outputs by the  $3N$  detection technique.

The shifting operation moves the single-bit error  $\pm 2^f$  to different positions in the adder/subtractor inputs of the CORDIC iteration after the fault occurs, thus making aliasing possible. Thus, if the single error affects the  $x$  data path in the  $i$ -th CORDIC iteration, it still appears in the  $(i+1)$ -th iteration as a single error  $\pm 2^f$  at the  $x$  output and  $\pm \sigma_{i+1} 2^{f-2i-2}$  at the  $w$  output. Then, at the  $(i+2)$ -th iteration, the multiple error  $\pm 2^f \mp \sigma_{i+2} \sigma_{i+1} 2^{f-4i-6}$  affects the  $x$  output, while the  $w$  output contains the multiple error  $\pm \sigma_{i+2} 2^{f-2i-4} \pm \sigma_{i+1} 2^{f-2i-1}$ . It is possible to verify that there is always at least one of these two outputs which is not divisible by 3 at any CORDIC iteration (and at the final CORDIC outputs). In other words, even if the reconvergent paths transform single-bit error into a multiple-bit error, the error is still detectable by checking only the final output of the CORDIC processor for the  $x$  and  $w$  data flows. This is due to the specific nature of the CORDIC computation.

## 5: Protection of the control structure

The internal structure of the rotation direction generator consists of some adders and combinatorial circuits [6]. The complexity of this part is usually so small with respect to the arithmetic data paths  $x$ ,  $w$ , and  $z$ , that hardware duplication with comparison can be used to protect the rotation direction generator of each stage. Operations are completely duplicated and results are compared at each CORDIC iteration step.

This approach requires twice the complexity of the basic generator plus a one-bit comparator. Generation of the directions is accommodated within the same time interval of a single CORDIC iteration. It is worth noting that the modular redundancy with comparison covers at least the same error model adopted in the arithmetic data paths.

## 6: Architecture evaluation

The additional circuit complexity of the proposed approach to concurrent error detection is due to the higher number of operand bits in the arithmetic data paths (encompassing adder/subtractors, multiplexers, carry-propagation adders), the decoder/checkers, and the duplicated rotation direction generators with comparators. In pipelined architectures, also the larger registers contribute to the higher circuit complexity.

The latency is increased by the final carry-propagation adders and by the decoder/checkers. The carry-propagation adder is slower since its operands have larger size (2 bits more than in the uncoded operands). The whole latency of the decoder/checker must be added to the system operation. If the application receives the information about CORDIC output correctness with some delay, we can include a pipeline stage between the unpipelined CORDIC structure and the decoder/checker. This makes the result available without any latency increase due to the decoder/checker, even though the related error signal will be delivered in parallel to the subsequent CORDIC output. In pipelined solutions, both the final adder and the decoder/checker are pipelined so that the latency increase is one pipeline clock cycle.

The throughput is marginally affected by concurrent error detection. In the non-pipelined structure, the throughput is reduced by the slower operation of the final carry-propagation

adder and by the presence of the decoder/checker. However, if a pipeline register is introduced between the CORDIC processor and the decoder/checkers as said above, the throughput is fully preserved, since CORDIC processing and decoding/checking are fully overlapped in time. The throughput is not affected by concurrent detection also in the pipelined solutions.

Table I shows the relative circuit complexity and the relative latency with respect to the architecture having similar component structures but without concurrent error detection, for different operand sizes. These figures have been evaluated for the fully pipelined architecture; similar values can be obtained for the granularly-pipelined and unpipelined solutions. Evaluation has been performed by adopting the standard gate count approach [15].

## 7: Conclusions

A family of architectures for redundant CORDIC processing in vectoring mode has been presented, which has concurrent error detection abilities as well as different performance tradeoffs. Low latency is obtained by fully overlapping the redundant sum computation and generation of the rotation direction within each stage. Granular pipelining makes several speed options available, with corresponding circuit complexity and latency.

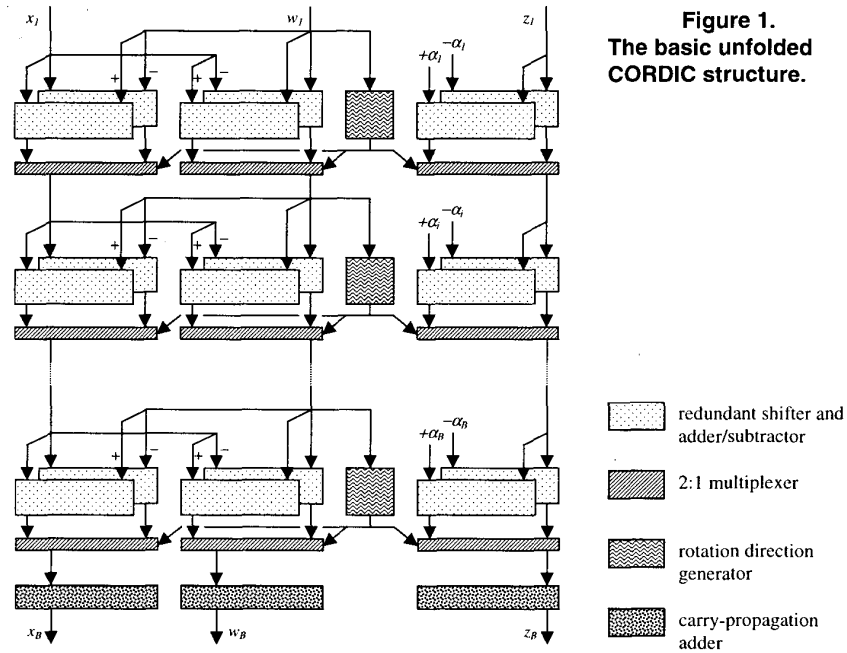
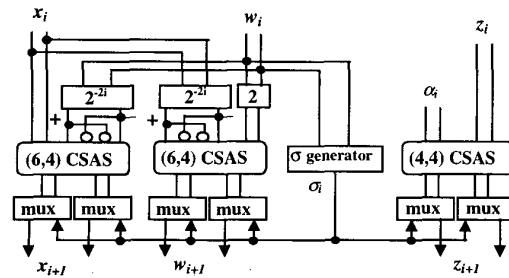
The high regularity of the structure, the use of carry-save adders in each pipeline stage, and the peculiar characteristics of the CORDIC algorithm make viable the use of the  $3N$  code for concurrent error detection in the arithmetic data paths at a minimum cost in circuit complexity and latency. For example, in the case of 16-bit operands, the increases of circuit complexity and latency are about 15.1% and 6.2%, respectively.

## References

- [1] J. S. Walther, "A Unified Algorithm for Elementary Functions," *Proc. of Spring Joint Computer Conference*, pp. 379-385, 1971.
- [2] M. D. Ercegovac and T. Lang, "Redundant and On-line CORDIC: Application to Matrix Triangularization and SVD," *IEEE Trans. on Comput.*, vol. 39, pp. 725-740, 1990.
- [3] N. Takagi, T. Asada, and S. Yajima, "Redundant CORDIC Methods with a Constant Scale Factor for Sine and Cosine Computation," *IEEE Trans. on Comput.*, vol. 40, pp. 989-995, 1991.
- [4] J.-A. Lee and T. Lang, "Constant-Factor Redundant CORDIC for Angle Calculation and Rotation," *IEEE Trans. on Comput.*, vol. 41, pp. 1016-1025, 1992.
- [5] D. Timmermann, H. Hahn, and B.J. Hosticka, "Low Latency Time CORDIC Algorithms," *IEEE Trans. on Comput.*, vol. 41, pp. 1010-1014, 1992.
- [6] J.-H. Choi, J.-H. Kwak, and E. E. Swartzlander, Jr., "High-Speed CORDIC Architecture Based on Redundant Sum Formation and Overlapped Sigma-Selection," *IEEE Proc. of the International Conference on Computer Design*, 1999.
- [7] S. Wang, V. Piuri, and E. E. Swartzlander, Jr., "Hybrid CORDIC Algorithm," *IEEE Trans. on Comput.*, vol. 46, pp. 1202-1207, 1997.
- [8] S. Wang and V. Piuri, "A Unified View of CORDIC Processor Design," in *Application Specific Processing*, Boston: Kluwer Publishing, pp. 121-160, 1997.
- [9] D.P. Siewiorek and R.S. Swarz, *Reliable computer systems: design and evaluation*, Burlington, MA: Digital Press, 1992.
- [10] T.R.N. Rao, *Error Coding for Arithmetic Processors*, New York: Academic Press, 1974.
- [11] R. Stefanelli and M. Annaratone, "A multiplier with multiple error correction capability", *Proc. ARITH-6*, 1983.
- [12] A. Antola, V. Piuri, and M. Sami, "A High-Level Synthesis Approach to Optimum Design of Self-Checking Circuits," *Proc. European Design Automation Conference EURO-DAC'96*, Geneva, Switzerland, pp. 382-387, 1996.
- [13] V. Piuri and R. Stefanelli, "Efficient use of the  $3N$  code for concurrent error detection in parallel multipliers," in *Computer Arithmetic, Scientific Computation and Mathematical Modelling*, J.C. Baltzer AG, Scientific Publishing Co., pp. 69-88, 1991.
- [14] L. Dadda and V. Piuri, "Pipelined adders," *IEEE Trans. on Comput.*, vol. 45, pp. 348-356, 1996.
- [15] C. Mead, L. Conway, *Introduction to VLSI System*, Addison-Wesley, 1980.

Table I. Relative Circuit Complexity and Latency.

| Number of Operands' Bits    | 8      | 16     | 32     | 64     |
|-----------------------------|--------|--------|--------|--------|
| Relative Circuit Complexity | 130.2% | 115.1% | 107.6% | 103.8% |
| Relative Latency            | 112.5% | 106.2% | 103.1% | 101.6% |

Figure 1.  
The basic unfolded  
CORDIC structure.Figure 2.  
The detailed structure  
of a single stage in  
CORDIC iteration [8].  
The  $(m,n)$  CSAS  
means an  $m$ -input,  $n$ -  
output carry-save  
adder/subtractor.

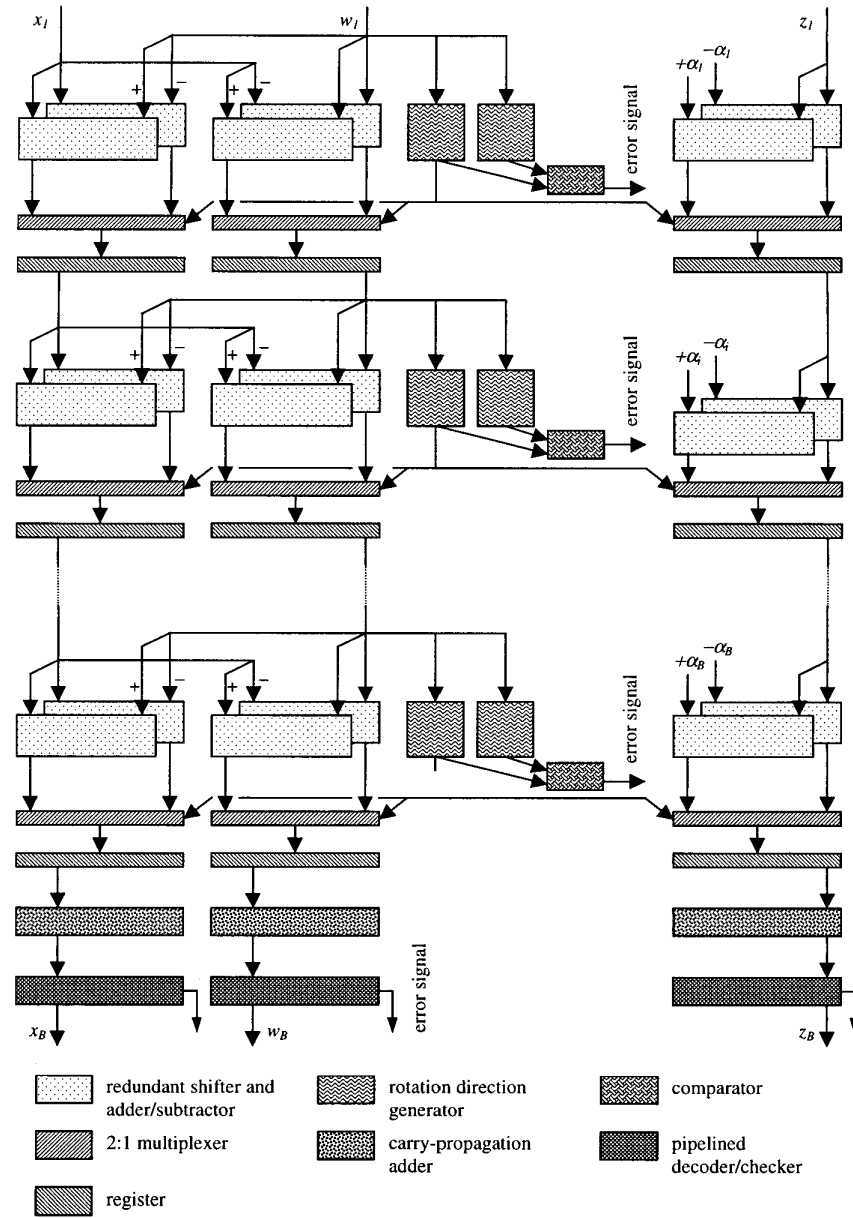


**Figure 3. The fully pipelined CORDIC structure with concurrent error detection ability.**