Research report 164

VLSI DESIGN OF A PIPELINED CORDIC PROCESSOR

P CHOWN, D W WALTON, G R NUDD

(RR164)

In this report we discuss the VLSI realisation of a pipelined CORDIC arithmetic unit to perform stable matrix row operations for the solution of systems of linear equations. The algorithmic considerations of the CORDIC process are highlighted and chip level architecture is derived from those to implement algorithm in a pipelined manner. We then proceed to give details of the 2pm CMOS processor that has been designed to implement that architecture.

Department of Computer Science
University of Warwick
Coventry CV4 7AL
United Kingdom

October 1990

VLSI Design of a Pipelined CORDIC Processor

P. Chown, D.W. Walton, G.R. Nudd

Department of Computer Science,
University of Warwick,Coventry.

Abstract

In this report we discuss the VLSI realisation of a pipelined CORDIC arithmetic unit to perform stable matrix row operations for the solution of systems of linear equations. The algorithmic considerations of the CORDIC process are highlighted and a chip level architecture is derived from these to implement the algorithm in a pipelined manner. We then proceed to give details of the 2μm CMOS processor that has been designed to implement that architecture.

Algorithm

Introduction

The CORDIC operation has been widely used in the implementation of modern signal processing systems. This is due to the many functions that can be computed using the same hardware and the stability of the method compared to traditional approaches. It has been applied to many domains including the solution of linear equations [1,2,3,4], filtering [5], Single Value Decomposition [2,6] and direct matrix techniques such as the Faddeev Algorithm [7,8,9].

This report describes the design of a pipelined CORDIC processor to perform QR factorisation using Givens rotations. The Givens method performs a matrix row operation, replacing the traditional multiply and add (Gaussian elimination) with a two dimensional rotation. Traditionally an element of the matrix is reduced to zero by adding a multiple of another row to the row containing that element. The Givens method treats the two rows as a series of two-dimensional vectors formed by the pairs of points corresponding to a particular column. These vectors are then rotated in such a way that one of the coordinates in a particular vector becomes zero. All vectors are rotated by the same angle. This method has the property of increased numerical stability ; intuitively this is due to the fact that the length of the vector does not change under a rotation operation.

A more detailed description of the application of the Givens technique to signal processing can be found in the paper describing the iterative version of this processor [1]. We describe here the implementation of the pipelined design proposed in that paper and discuss the architectural problems encountered in the construction of such a processor from the systems level to the design of a CMOS circuit.

The Cordic Algorithm

The CORDIC (COordinate Rotation on a DIgital Computer) algorithm was first introduced by Volder [10] and provides a unified method of performing vector rotations and multiplication / division whilst performing only shift and add operations. Volder's algorithm was generalised by Walther [11] to include hyperbolic functions, exponentials and square roots.

The CORDIC method is based on the following set of iterative equations :-

$$x_{i+1} = x_i + \partial_i\, m\, y_i\, \mu_i$$
$$y_{i+1} = y_i - \partial_i\, x_i\, \mu_i$$
$$z_{i+1} = z_i - \partial_i\, \alpha_i$$

$$\alpha_i = (1/\sqrt{m}) \arctan(\mu_i\sqrt{m}) \quad : \quad \mu_i = 2^{-i}$$
$$\partial_i \in \{1, -1\} \quad : \quad m \in \{1, 0, -1\}$$

The equations perform a vector rotation operation in terms of a series of smaller imperfect rotations, as described in [12]. Each of the smaller rotations may be in either a clockwise or anticlockwise direction selected by the value of ∂_i. By allowing the direction to be chosen for each iteration separately a number of functions may be computed. The value of m is used to select either circular, linear or hyperbolic space. (1, 0, -1 respectively). The value of μ_i is chosen to be 2^{-i} with the result that each iteration can be performed using only simple shift and add operations.

On completion of the iteration various functions are retained in the x,y,z registers depending on the different values of ∂_i and m. The available functions are summarised in Figure 1.

	Rotate z to zero $\partial = \mathrm{sgn}(z)$	Rotate y to zero $\partial = \mathrm{sgn}(y)$ xor $\mathrm{sgn}(x)$
Circular m=1	$K(x \cos z + y \sin z)$ $K(y \cos z - x \sin z)$ 0	K x+y 0 $z - \tan (y/x)$
Linear m=0	x $y - xz$ 0	x 0 $z - (y/x)$
Hyperbolic m=-1	$K(x \cosh z - y \sinh z)$ $K(y \cosh z - x \sinh z)$ 0	K x+y 0 $z - \tanh (y/x)$

Figure 1 - CORDIC functions

The CORDIC processor that we describe in this paper performs the circular subset of the algorithm only. Since the implementation of Givens method requires only circular rotations the additional functionality is not required. Provision of the additional modes also carries an overhead due to the increased connectivity between data bits in successive stages of the pipeline. This will lead to both hardware cost and corresponding decreases in circuit speed.

K-factor Correction

Each of the iterations in the above equations introduces a scale factor error due to the imperfection of the rotation performed. These scale factors are multiplicative and accumulate as the computation proceeds. The total scale factor after N iterations is denoted by K and for the circular mode is equal to

$$K = \prod_{i=0}^{N-1} \frac{1}{\cos(\partial_i \arctan 2^{-i})}$$

$$= \prod_{i=0}^{N-1} \mathrm{sqrt}(1 + 2^{-2i}) = 1.646760...$$

The scale factor error inherent in the CORDIC process is an important aspect to consider and may be dealt with in a number of ways. The most direct method is to perform a division operation after each CORDIC operation, removing the scale factor directly. This can be expensive in terms of hardware and may severely limit performance. More popular methods solve the problem by incorporating a number of extra iterations into the algorithm in order to force K to be an integer power of the number radix (or unity). Once the scale factor has been modified in this way it may be removed by a simple shift operation.

Haviland [13] proposes that the additional scale factor iterations take the form of the following fixed multiples :

$$x = x_N - 2^{-i}x_N = x_N(1 - 2^{-i})$$
$$y = y_N - 2^{-i}y_N = y_N(1 - 2^{-i})$$

Each of these constant multiples may be formed by a single shift and subtract operation and has a value slightly less than one. K may be reduced to unity in this way by post multiplying the results by a suitable set of these multiples. The accuracy of the combined constant factor must be sufficient to avoid the introduction of errors into the results. For 32 bit integers 16 additional iterations are required to reduce the results to their correct values.

Ahmed [4] has devised a method in which a number of standard CORDIC iterations are repeated to force the value of K to be equal to 2. To attain the required accuracy in this case, again for 32 bit integers, an extra 28 iterations are required.

The system that we have adopted is that proposed by Delosme [14] which combines the ideas of Haviland and Ahmed. For 32 bit integers his method requires an additional nine iterations to reduce K to the value 2. Seven extra CORDIC iterations (i values of 0, 1, 3, 5, 6, 8, 14) are used in conjunction with two special iterations $(1 - 2^{-2})$ and $(1 + 2^{-6})$.

Recent years have seen the development of CORDIC implementations that make use of redundant number systems. These employ a redundant digit set for ∂ of (-1,0,1) allowing faster circuits to be implemented that make use of carry save addition/subtraction units. A consequence of the use of redundant number techniques is that the delta control circuitry must operate by making an *estimate* of the signs of the values from the previous iteration, with the additional option of deciding that the sign at this stage is zero, deferring the decision to a later iteration. Unfortunately this results in a value of K which is data dependent since the selection of zero for ∂ at a particular stage effectively removes the scale factor contributed by that stage. For the interested reader, some recent solutions to this problem have been published called the *double rotation method* [15], *correcting rotation method* [16], *Branching CORDIC* [12] and *online calculation* [2].

Application to Givens Rotations

Two distinct operations are involved in the application of a Givens rotation to a pair of matrix rows. The first operation is the calculate of the rotation angle using the relevant pair of values from the rows. The angle must be selected such that a prespecified member of the pair is reduced to zero. Given a vector <x,y> and rotating y to zero, we must rotate by a total angle, Z of

$$Z = \arctan(y/x)$$

In the second stage of the process this rotation is applied to the remainder of the elements in the rows. It has been noted by Deprettere [5] and Sung [17] that the sequence $\{ \partial_0 .. \partial_n \}$ is an equivalent representation of the rotation angle Z. Thus by storing the sequence of ∂ values calculated in the first stage, the second stage may be easily performed by 'replaying' those values. This considerably reduces the hardware requirements by not performing calculations using stored values of Z and α_i but by stored ∂ values. Due to the bit-recursive nature of the ∂ calculation each pipeline stage has only to calculate and retain a single ∂ value.

CHIP-LEVEL ARCHITECTURE

Overall Structure

The CORDIC algorithm described above is an inherently integer operation, and has been implemented as such by a number of institutions [7,8,9] including the University of Warwick [10]. This section will describe the architecture of the Warwick Pipelined CORDIC Processor which extends the operation of the algorithm to the IEEE 754 floating point number system, and pipelines the iterations of the algorithm to improve processor throughput. Using a readily available CMOS process technology, namely the 2μm process from ES2, it has been found possible to implement such a device in a silicon area small enough for commercial exploitation.

Floating point operation is achieved by making use of the fact that each individual step of the algorithm consists solely of additions and shifts. Therefore by aligning the mantissas of the two input numbers and storing the corresponding exponent it is possible to implement the pipeline as though it were a simple integer CORDIC system. The results of the computation must be post-normalised and recombined with the common exponent before they can be returned to the user as floating point values.

Previous implementations of the CORDIC algorithm have generally taken the form of registers for the x,y and z variables. When performing the computation barrel shifters are used to form the shifted versions of the x and y values which are added to the original ones to form new x,y values. The values of α needed in the z computations are typically stored in a ROM either on the same processor or as a separate unit. The algorithm is implemented by repeating the iterations on these registers until the computation on a set of values is complete. A number of institutions [1,5] have explored the possibility of pipelining the computation such that each stage of the pipeline performs a single iteration of the algorithm.

Several properties of the CORDIC algorithm have been used in the construction of our processor. The first is the use of stored ∂ values at each stage instead of the full value of z, as described earlier. Each stage of the pipeline computes a single ∂ value and stores this. In addition, the shifts at a particular stage of the iteration are fixed, and so they can be hard wired rather than being performed by a general shift unit.

Using these properties each pipeline stage performs a single iteration of the algorithm taking as inputs the values held in the registers of the previous stage. The shifting of the x and y values is performed by a series of point to point connections. This leads to space savings and allows rapid operation of the pipeline. Despite the above considerations some restrictions must still be imposed on the functionality of the unit in order to place a complete design on a realisable processor.

Floating Point Circuitry

From the above discussion, it can be seen that the design of the Pipelined CORDIC Processor falls naturally into two areas. These are the floating point manipulation area and the integer CORDIC pipeline. The floating point section of the CORDIC processor must carry out a number of functions. These functions are :

- Alignment of the mantissas of the two input numbers according to the values of their exponents.
- Storage of the exponent that corresponds to the integer values in each stage of the CORDIC pipeline (these values are stored in the Exponent Pipeline).
- Post-normalisation of the results emerging from the CORDIC pipeline and combination of the normalised numbers with the value obtained from the Exponent Pipeline.
- Detection and handling of error and overflow conditions arising from the CORDIC operation.

The mantissa alignment for the two input numbers is performed by subtracting the two exponent values obtained from the input numbers. The magnitude of this result is used to determine the right shift to be performed on the smaller mantissa. The sign of the difference in exponent values is used to select the correct mantissa for shifting. The sign of the exponent difference is also used to select the larger of the two exponent values for storage. The selected exponent is passed to the Exponent Pipeline where it keeps pace with the data passing through the CORDIC pipeline. When the results emerge from the end of the CORDIC Pipeline, the exponent corresponding to those results will be available at the end of the Exponent Pipeline ready for the post-normalisation operation.

The sign information contained within the floating point input numbers must be combined with the mantissas to form the signed integers needed by the CORDIC pipeline. This operation is not carried out until *after* the mantissa alignment operation has been performed. By keeping the sign information separate in this way, we can restrict the shift to unsigned numbers, simplifying the design of the barrel shifter.

The post-normalisation operation is performed separately on each of the two output values from the CORDIC pipeline. For each number the following operations are performed. Firstly the sign information is extracted from the integer value, and this is placed into the sign bit of the output floating point number. The resulting unsigned number is then left shifted in a data-dependant manner such that the left most bit of the shifted data is the most significant '1' in the original number. The resulting left adjusted number is then placed into the mantissa of the floating point number to be output, losing the initial '1' as this is supplied by default in the IEEE 754 standard. The number of bits that this data had to be shifted is subtracted from the exponent value supplied by the Exponent pipeline and becomes the output exponent.

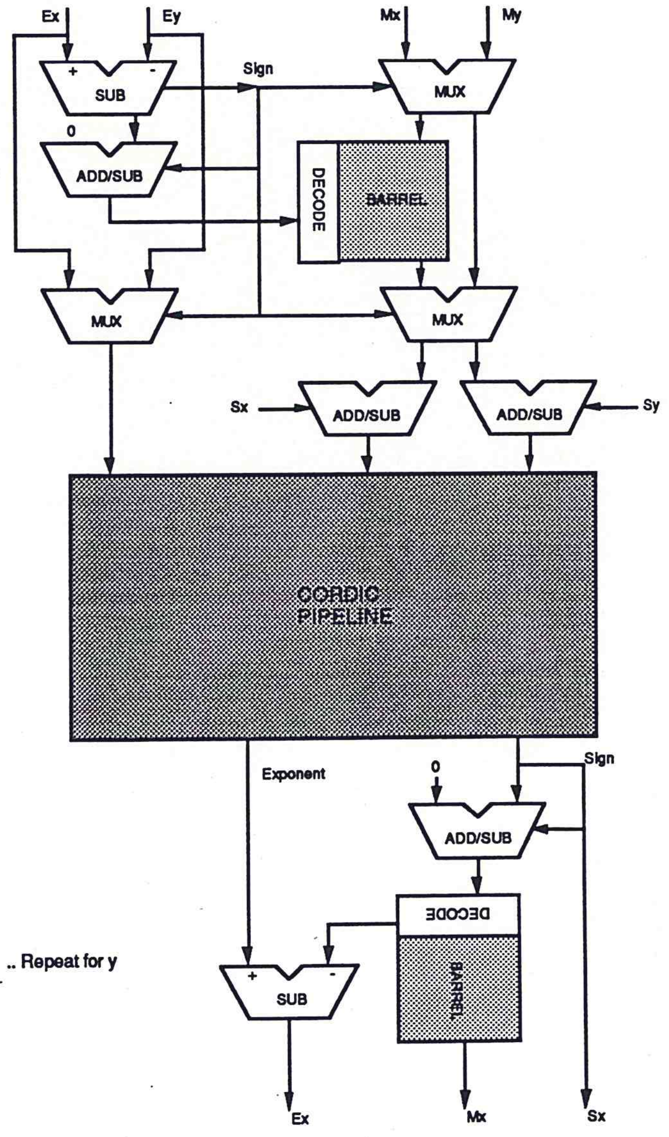


Figure 2 - Floating point circuitry

K-Factor Correction Circuitry

As detailed previously, several additional stages are needed within the CORDIC pipeline to eliminate the K-factor that arises in the course of the computation. These additional stages are similar in construction to the pipeline stage already presented. If this were not the case then the regular structure of the pipeline would be disturbed which may cause layout problems. Since the construction of these stages is not remarkably different from those already described, no further details will be given here.

Floor Plan

The major part of the silicon area of the Pipelined CORDIC processor is devoted to the integer pipeline and K-factor correction circuitry. The large number of iterations, each operating on two 32-bit integers, combined with the need to perform K-factor correction and the large size of the basic adder/subtractor cell, leads to this large area requirement. Given that the integer pipeline is by far the largest unit in the system this is the part of the design that will determine the placement of the remaining circuits. After some initial estimates of the size of the pipeline and the size of the cells that would be used to construct it, a rough floor plan of the cell was drawn, showing the flow of data through the IC and the placement of the larger units. This diagram is shown in Figure 5.

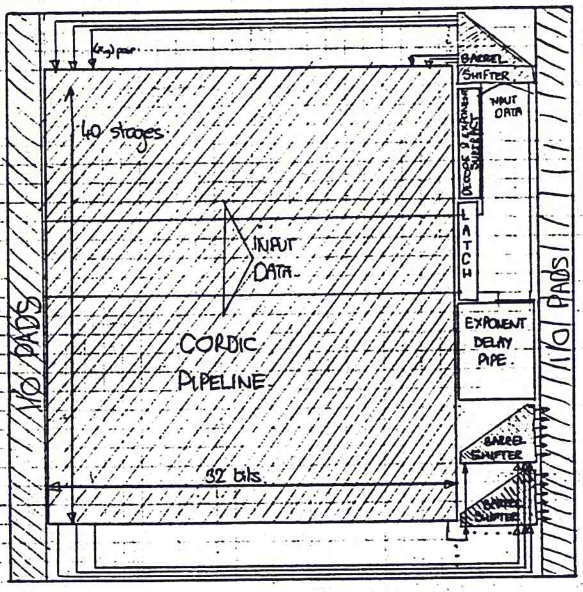


Figure 5 - Rough Floor Plan

Input and Output

The Pipelined CORDIC Processor may be packaged in a number of ways, dependant on the choice of input/output methods and control of the system The design of the pipeline is such that on each step of the pipeline clock, two floating point numbers are consumed and two are produced.

In order to provide a single I/O pin for each input or output signal 128 pins would be required. In addition other pins are required for power, ground and control signals, leading to an estimated total of 145 pins. By multiplexing the inputs and outputs to the processor it is possible to reduce the pin requirement, although this leads to less flexibility when interfacing the processor to a system and a reduction in the speed of the pipeline. By multiplexing all of the four input and output numbers onto a single 32 bit port a minimum figure of approx. 45 pins is obtained. The speed of the pipeline here is 1/4 that of the external bus. We have chosen an intermediate solution where both inputs are multiplexed onto one port and both outputs onto another port. As a compromise between a very large pin count and a very slow pipeline, this choice also allows a fair amount of flexibility in the architecture supporting the processor. This solution leads to 81 pins allocated as follows :

64 Data Pins
8 Power/GND pins
1 Clock input
1 Output Invalid pin
1 Pipeline halt pin?
1 Delta Set input
1 Chip enable pin
1 Output Enable
1 Input Latch Enable
1 Input register select
1 Output register select

~81 Pins Total

Cell Layout and Design

Overview

The design of the Warwick Pipelined CORDIC Processor was performed on a number of networked SUN Workstations using the MAGIC design tools from the University of California, Berkeley. These tools provide a number of useful features for the designer of VLSI circuits, notably the use of a symbolic design style and the provision of an interactive design rule checker that operates on designs as they are entered. The symbolic nature of the tools arises from the fact that only key layers are used to perform the design itself. Layers that can be derived from the basic layers are derived using a large number of rules given to the MAGIC package. For example where a p-fet is formed by passing polysilicon over p-diffusion, an n-well is automatically formed around the transistor to accommodate it. The additional layers are normally invisible unless viewed explicitly which helps to keep the design uncluttered.

The CORDIC processor consists of a small number of cells that are iterated over the majority of the silicon area, which is the reason that the circuitry could be implemented by a small design team in a short time. Most of the cells are fairly standard designs adapted for use in our application and so only brief notes are given here on their construction. Some circuit elements have required more work than is detailed here and where appropriate other documents have been written describing those details. References to these are given at the end.

The block diagram for the circuitry that is used to perform the floating point input and output operations is shown in Figure 2.

In addition to the circuitry used to perform the floating point to integer conversions for the CORDIC pipeline, it is also necessary to provide a certain amount of random logic that can detect overflow, underflow and error conditions within the processor. The outputs from this circuitry can be used either to activate an error flag that is output with the results or to cause particular values to be output in place of the calculated one. Typical conditions that need to be detected and handled are :

- Invalid number (NAN) given as an input
- Zero has been input on one or both inputs
- INF has been input on one or both inputs
- Number has been reduced to Zero
- Number has become INF

These are the major conditions that need to be detected. The actions that should be taken in each case can be deduced by examining the representations of numbers within the IEEE 754 standard and the behaviour of the CORDIC algorithm. The circuitry to perform these functions consists of a small amount of random logic that can be laid efficiently using a silicon compiler. The design of this section has not yet been completed although the ES2 Solo tools have been selected for the compilation, producing designs conforming to the ES2 2µm CMOS rules.

Structure of the CORDIC Pipeline

An integer pipeline stage from the rotation processor implements a single step in the CORDIC iteration described previously. This operation involves a single add/subtract for each of the x and y values, followed by a shift, preparing the input values for the next iteration of the algorithm. Since the shifts are constant at each stage of the algorithm, the shifts can be hard wired into the pipeline avoiding the need for a full barrel shifter at each stage. Since the add/subtract and register cells have been designed specifically without the use of the second metal layer, it is possible to use the same area to perform both the calculation and shifting in the same area. The shifts are implemented by taking the signals onto the the second metal layer above the pipeline, performing the shift and then bringing them back down for the next stage. The block diagram for a single pipeline stage is shown in Figure 3.

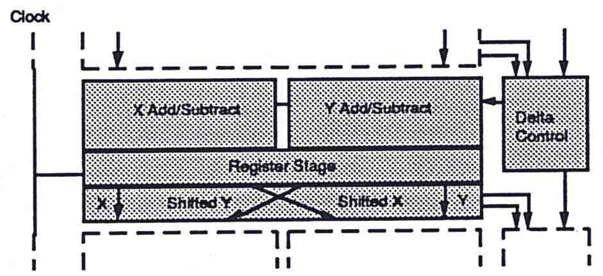


Figure 3 - Pipeline Stage Block Diagram

Our implementation consists of a fully parallel pipelined system and due to the large number of repeated stages in the pipeline each stage should occupy as small an area as possible. There remains a compromise between circuit complexity (i.e. area) and cycle time. Techniques such as redundant number systems reduce cycle time at the cost of dramatically increasing circuit area. Due to the limitations on processor area and design time, a simpler conventional method was used to implement the stages.

Each stage within the pipeline consists of two 32-bit adder / subtractors plus the associated delta control and clock driver circuitry. Although the mantissas contained in the floating point input numbers are only 24 bits long, 32 bit adders are used for the following reasons ; firstly in order that the truncation error arising from the right shift of the smaller mantissa does not corrupt the result, 6 bits are retained from the truncation, corresponding to 2^6 or 64 times the truncation error. Since there are only 39 additions taking place in the pipeline this error is easily accommodated. In addition there are two bits at the high end of the number to allow for expansion. Assuming that the input number is a binary fraction with a maximum value of 1, the maximum output value is given by the expression

$$max = 2 + 1 + 2^{-1} + 2^{-2} + ... \Rightarrow 4$$

which can be accommodated by two additional bits.

Delta Control Circuitry

The specification of the algorithm states that at each iteration, a decision has to be made about which direction the data should be rotated. This decision is represented in the CORDIC equations by the choice of the value of ∂, which is in turn determined by the values of x, y, z and the mode in which the CORDIC algorithm is being operated. As has already been discussed, the choice of ∂ has been simplified in our implementation by restricting the CORDIC algorithm to the circular mode of operation, and by using ∂-values obtained from one pair of input values to determine the rotations that are to be performed on the data that follows them.

The circuitry that performs the selection of the ∂ values, the Delta Control section, takes sign information from the data values of the previous stage and generates a single Add/Subtract signal. This signal is passed to all of the adder/subtractor cells in that stage of the pipeline. The Delta Control circuitry also makes use of a Compute/Apply signal that is also pipelined to keep pace with the data items to which it corresponds. Using this signal the control circuit will either generate a ∂-value from the signs of the input numbers or apply a previously calculated ∂-value (circuit shown in Figure 4). Since the Compute/Apply signal is also pipelined it is not necessary to wait for the pipeline to be flushed before the operation of the Delta circuits can be changed.

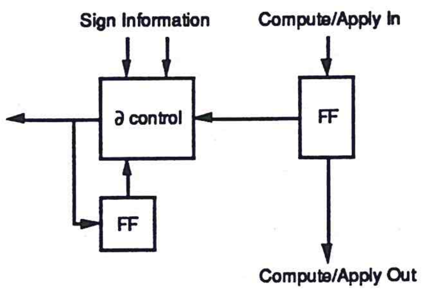


Figure 4 - Delta Control Block Diagram

Pipeline Adder

In choosing an adder implementation for any system a compromise must be made between speed and circuit size. In our application the adder cell is replicated a great number of times and so the size of the adder is the most important factor to be considered. Taking into account the size constraint we have chosen to make the Manchester Carry Adder the focus of our design. Once the basic design work had been carried out, the circuit was optimised for multibit addition by fine tuning transistor sizes.

The Manchester Carry Adder [18] differs from a standard ripple adder through the way in which the carry information is passed along the calculation. Consider one stage of a Manchester Carry adder. Under differing input conditions, the 'carry in' signal may be allowed to pass unchanged to the next stage, or be forced to a '1' or a '0'. The central idea behind the Manchester Carry adder is to detect the condition where the carry input is equal to the carry output and in this case feed it directly through a pass gate. By doing this we attempt to speed the propagation of a carry signal which is the limiting factor in a conventional adder.

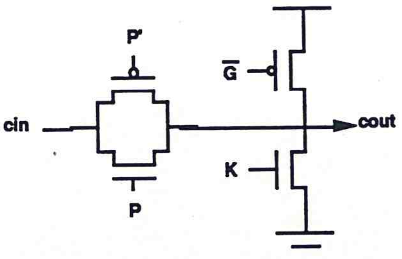
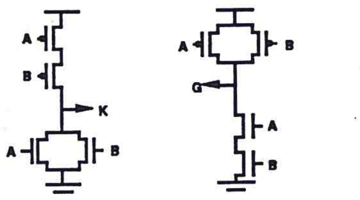


Figure 6: Carry Propagation

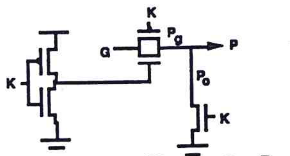
The circuit to implement the carry chain of a Manchester adder is shown in Figure 6. We have used the concepts of kill (K), generate (G) and propagate (P) in addition to the carry in and carry out signals. If we analyse the logic tables describing a full adder then it can be seen that when the two inputs are both low the carry output will always be low and this represents killing the carry. Similarly when both inputs are high a high carry output will always be generated. When the two inputs are different the carry is simply propagated to the next stage. The K,G and P signals are high when the respective conditions are asserted.



B	A	K	G	cout
0	0	1	1	0
0	1	0	1	Z
1	0	0	1	Z
1	1	0	0	1

Figure 7 - Generate and Kill signals

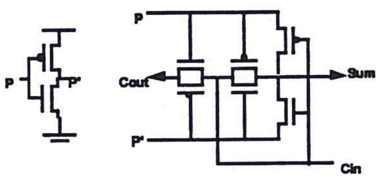
The generate and kill control signals are produced by the simple NOR and NAND circuits shown in Figure 7. The propagate signal can be logically derived from the input signals by making use of an XOR gate but is produced more efficiently from the kill and generate signals. A structure similar to the classic 6 transistor XOR cell is used but with only 5 transistors, shown in Figure 8.



B	A	K	G	Pg	Po	P
0	0	1	1	Z	0	0
0	1	0	1	1	Z	1
1	0	0	1	1	Z	1
1	1	0	0	0	Z	0

Figure 8 - Propagate signal

The last piece of circuitry contained in the adder is that which deals with carry propagation and sum generation. In the case of a multibit adder the total calculation time is made up of two components. Initially the kill, generate and propagate signals for all of the adders are calculated, taking a small constant time. The majority of the calculation time is taken up by the propagation of the longest carry through the adder chain. It is therefore the propagation path between the carry in and carry out signals that should be optimised first to achieve the most benefit.



Cin	P	Sum	Cout
0	0	0	Z
0	1	1	0
1	0	1	Z
1	1	0	1

Figure 9 - Carry and Sum Circuitry

The sum at a particular stage is formed from the propagate and carry in signals by a pass gate and two extra transistors. The carry out signal is simply the carry in passing though a pass gate controlled by the propagate signal, as shown in Figure 9. For a carry signal to be passed through a single stage of the Manchester Carry adder, the following circuit elements must be either charged or discharged :

Two transistor gates
Two conducting transistor channels
Two output capacitances of OFF transistors

The complete Manchester Carry adder is constructed using 23 transistors and is layed out in a horizontal brick shape as shown in Figure 10. In order to provide both add and subtract operations, a single XOR gate is positioned at one of the inputs to provide a selective inversion of that input. By inverting the input and altering the carry into bit 0 respectively, subtraction is obtained. The basic pipeline cell thus contains an adder, an XOR gate and a flip flop to buffer the data between pipeline stages. This pipeline cell is duplicated to obtain a structure with the functionality shown in the block diagram of Figure 3.

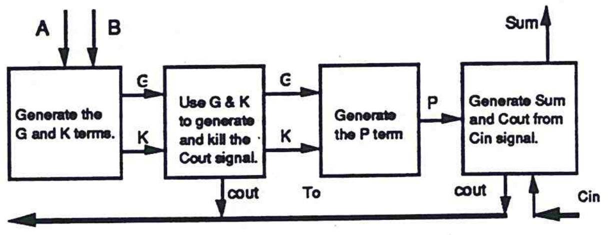

A
B
Sum
Generate the G and K terms.
G
K
Use G & K to generate and kill the Cout signal.
G
K
Generate the P term
P
Generate Sum and Cout from Cin signal.
cout
To
cout
Cin

Figure 10 - basic shape of adder

When a long carry chain occurs in a Manchester Carry adder of the type described above the capacitance of the carry path becomes large as does the number of transistors through which the signal must pass. This results in a significant reduction in speed or even incorrect operation of the circuit. In order to solve this problem buffers have to be inserted into the carry path at regular intervals to restore signal levels. If there was sufficient room for the additional hardware then a mechanism known as carry skip could be implemented which detects a long propagation path and provides a shorter alternative route for the carry signal.

Shift Connections

The shift operation is performed within the pipeline by providing point to point hard wired connections using the second metal layer available in the ES2 2µm process. To implement a shift structure as shown in schematic form in Figure 3 would not be practical because the wires from the two registers are forced to cross over each other. Since we are attempting to restrict ourselves to the second metal layer only, this approach is not suitable. In order to cross, vias are needed to the first metal layer which would interfere with the pipeline stages beneath.

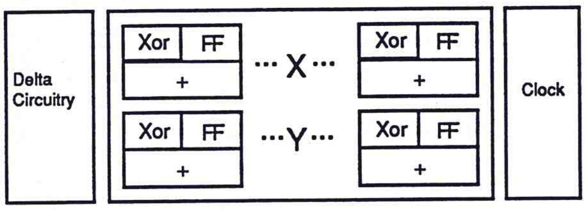


Figure 11 - Schematic of a single pipeline stage

In order to solve this problem the two adders corresponding to the x and y values from the CORDIC iteration are put on top of each other instead of next to each other. The crossing of the data lines is then performed locally in metal one. A single complete pipeline stage is shown in Figure 11. It contains two arrays of cells each of which contains an adder, XOR gate and a flip flop. All control signals for a particular pipeline stage are routed between the two adders of that stage.

The delta control circuitry is placed at the left edge of the stage and the clock circuitry is placed at the right edge. Vias are placed on the output of each flip flop to facilitate the hard wired shift connections that will be made when the pipeline is assembled. The interconnect that performs the shift is routed between this row of flip flop vias and the next stage in the pipeline. Two identical shifts per pipeline stage are implemented in this fashion, corresponding to the shifted versions of the x and y values in the CORDIC equations.

Delta Control Circuitry

This circuitry provides the value of ∂ required in the CORDIC iterations and passes it to the processing elements of a single pipeline stage. It is passed to the pipeline in the form of a delta control signal. This signal controls the operation of the pipeline adders, performing either an addition or a subtraction by means of the XOR gate provided. The circuit that derives the delta control signal is shown in Figure 12. The circuit comprises a flip flop to store the compute/apply bit, a flip flop to store a computed delta bit and a multiplexor. The multiplexor selects either the precomputed delta or a newly computed value which and passes this to the flip flop to be stored for the next cycle. As has already been described, the value of delta is obtained in our implementation by taking the XOR of the sign bits of the two previous data values.

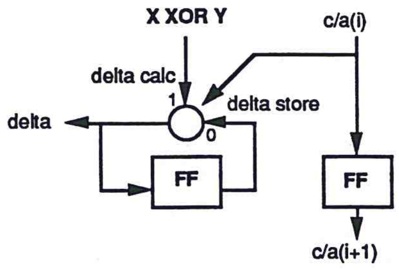


Figure 12 - Main delta control circuitry

A large buffer is also included in the delta control circuit to provide enough drive for the delta control line. Since the shifted versions of data values must be sign extended, two more buffers are added to boost the sign bits arriving at each pipeline stage and pass them to the most significant bits of the current stage. A block outline of the delta cell is shown in Figure 13. The sign bits and compute/apply signal run vertically and the delta signal runs horizontally across the middle of the cell.

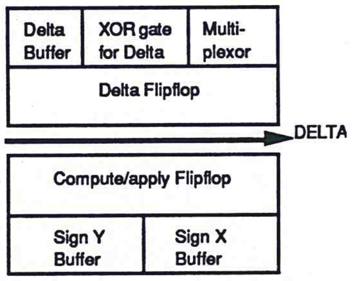


Figure 13 - Block diagram of delta cell

Barrel Shifter

A complete description of the options currently available for the design of Barrel Shifters and details of the implementation chosen for our application is described elsewhere [18]. We summarise here the results of that paper and describe in addition the design of the decoder circuits that are required in a complete design.

A single level switching network is used as the core of the shifter, routing the data path through a right angled turn while performing the shift operation. The shifter operates in a precharged manner, the entire network being precharged to +5V and then selectively discharged under control of the input values and the shift control signals. The use of a precharged scheme both speeds the operation of the circuit and allows the array to be designed using only n type switching elements. If a precharged scheme were not used then it would be necessary to use transmission gates as the switching elements since both high and low voltage levels need to be passed. This would lead to a much larger and slower shift network. The circuit for the main data path through the array is given in Figure 14.

The shift operation is performed within the pipeline by providing point to point hard wired connections using the second metal layer available in the ES2 2μm process. To implement a shift structure as shown in schematic form in Figure 3 would not be practical because the wires from the two registers are forced to cross over each other. Since we are attempting to restrict ourselves to the second metal layer only, this approach is not suitable. In order to cross, vias are needed to the first metal layer which would interfere with the pipeline stages beneath.

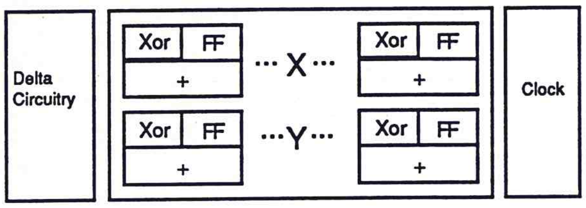


Figure 11 - Schematic of a single pipeline stage

In order to solve this problem the two adders corresponding to the x and y values from the CORDIC iteration are put on top of each other instead of next to each other. The crossing of the data lines is then performed locally in metal one. A single complete pipeline stage is shown in Figure 11. It contains two arrays of cells each of which contains an adder, XOR gate and a flip flop. All control signals for a particular pipeline stage are routed between the two adders of that stage.

The delta control circuitry is placed at the left edge of the stage and the clock circuitry is placed at the right edge. Vias are placed on the output of each flip flop to facilitate the hard wired shift connections that will be made when the pipeline is assembled. The interconnect that performs the shift is routed between this row of flip flop vias and the next stage in the pipeline. Two identical shifts per pipeline stage are implemented in this fashion, corresponding to the shifted versions of the x and y values in the CORDIC equations.

Delta Control Circuitry

This circuitry provides the value of ∂ required in the CORDIC iterations and passes it to the processing elements of a single pipeline stage. It is passed to the pipeline in the form of a delta control signal. This signal controls the operation of the pipeline adders, performing either an addition or a subtraction by means of the XOR gate provided. The circuit that derives the delta control signal is shown in Figure 12. The circuit comprises a flip flop to store the compute/apply bit, a flip flop to store a computed delta bit and a multiplexor. The multiplexor selects either the precomputed delta or a newly computed value which and passes this to the flip flop to be stored for the next cycle. As has already been described, the value of delta is obtained in our implementation by taking the XOR of the sign bits of the two previous data values.

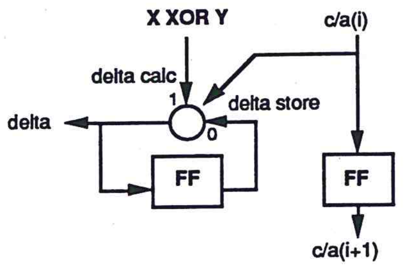


Figure 12 - Main delta control circuitry

A large buffer is also included in the delta control circuit to provide enough drive for the delta control line. Since the shifted versions of data values must be sign extended, two more buffers are added to boost the sign bits arriving at each pipeline stage and pass them to the most significant bits of the current stage. A block outline of the delta cell is shown in Figure 13. The sign bits and compute/apply signal run vertically and the delta signal runs horizontally across the middle of the cell.

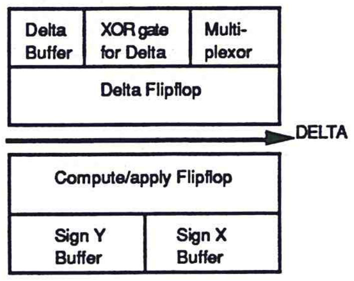


Figure 13 - Block diagram of delta cell

Barrel Shifter

A complete description of the options currently available for the design of Barrel Shifters and details of the implementation chosen for our application is described elsewhere [18]. We summarise here the results of that paper and describe in addition the design of the decoder circuits that are required in a complete design.

A single level switching network is used as the core of the shifter, routing the data path through a right angled turn while performing the shift operation. The shifter operates in a precharged manner, the entire network being precharged to +5V and then selectively discharged under control of the input values and the shift control signals. The use of a precharged scheme both speeds the operation of the circuit and allows the array to be designed using only n type switching elements. If a precharged scheme were not used then it would be necessary to use transmission gates as the switching elements since both high and low voltage levels need to be passed. This would lead to a much larger and slower shift network. The circuit for the main data path through the array is given in Figure 14.

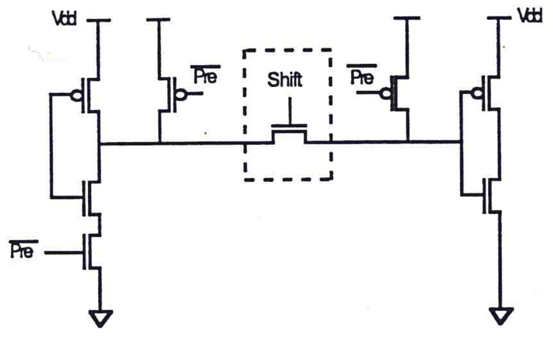


Figure 14 - Barrel Shifter Data Path

The design of the switch cells that comprise the barrel shifter network passed through a number of revisions from a full transmission gate to a single transistor. The final version of this cell is shown in Figure 15. There are several important points to note from this design. Firstly the shift control lines are run in polysilicon. Although this causes the delay for charging the shift lines to be increased relative to delay for metal, the charging of the shift lines can be performed in parallel with the precharging of the data lines. The second point to note is that only one of the two metal layers is used rather than both. This decision both reduces the area needed for the cell and allows routing to be performed in the second metal layer over the top of the barrel shifter. The support circuits are also designed without using the second metal layer for this reason. Thirdly, since only one transistor is used in the design, it is not necessary to include both types of substrate. The area savings arising from this are very large.

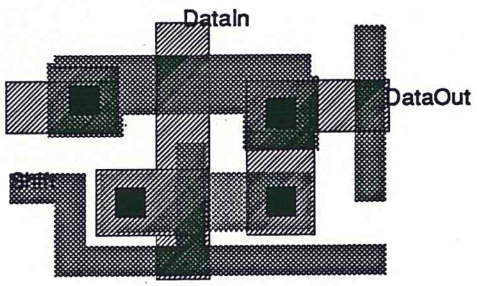


Figure 15 - Barrel Shifter Cell

In order to complete the design of the barrel shifter, it is necessary to provide signals for the Shift Control lines. This is achieved by performing some sort of decode operation on another piece of data to produce a set of data signals in which at most one signal is high at any particular time. In the case of the barrel shifter at the input of the processor the data to be decoded is the unsigned difference between the two exponents. In the case of the output barrel shifters the signal is the data itself, the shift lines being decoded in such a manner that the most significant '1' in the input number is shifted to the left hand end. In the latter case a binary number must also be produced indicating the number of bit positions by which the input was shifted. The circuitry to perform these functions is described in the next section.

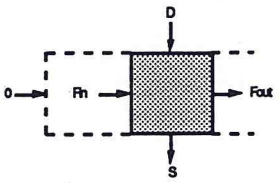
Shift Decode Circuitry

The shift signals that control the flow of data through the barrel shifters take the form of a number of data lines, only one of which may be set to a '1' at any particular time. The selection of the control signal that must be activated is made by converting from other data generated within the processor. For the mantissa alignment circuitry, the data determining the shift is the unsigned difference between the two input exponents. For the post normalisation shifters, the control information is derived from the data itself such that the most significant '1' is shifted to the most significant digit position.

For the input decoder, the binary input from the exponent subtract circuitry is converted to 1 of N form by making use of an AND gate for each shift control line. Both the true and inverted forms of each input bit are supplied to this set of gates, forming a direct 1 of N decoder. The circuit that we have implemented is probably at the limits of this sort of approach, as the pull up logic of an AND gate consists of a direct chain of p-transistors. Since the exponent difference is represented by five bits, we have five transistors in series which drastically slows the circuit. This has been slightly compensated for by making the pull up transistors wider but this cannot be continued indefinitely.

The decoder for the post normalisation shifter is a little more complicated. The selection of the most significant digit in a number is essentially a carry chain type of operation, as the information about more significant digits must be passed down before any particular digit can make the decision about whether or not it is the most significant one. This circuitry is likely to be slower than that of the input decoder for those reasons. Similar techniques to those applied to the improvement of carry chain performance in adders could be applied here also, but the circuit presented here satisfied our performance requirements and so no such addition was necessary.

By considering a number of approaches to the implementation of this operation, we arrived at a solution in which each bit position makes a decision based on its own data bit (D), and a flag from the next most significant digit that indicates whether or not a '1' has been found (Fin). Using this information, a Shift control bit (S) is produced and a value Fout is passed to the next bit position. Diagrammatically this is shown in figure 16.



D	Fin	S	Fout
0	0	0	0
0	1	0	1
1	0	1	1
1	1	0	1

$\Rightarrow$ $S = D.\overline{Fin}$

$Fout = Fin + D$

Figure 16 - Normalisation Decoder

By using a transmission gate to attempt to speed the operation of the carry chain, the following circuits have been designed that implement the truth table shown in figure 16 above. There is also inverters to produce the normal and inverted forms of the values D and Fin that are required, which are not shown. Since it is possible for the F signal to be passed through a number of stages, level restoring buffers are placed after every four decode units.

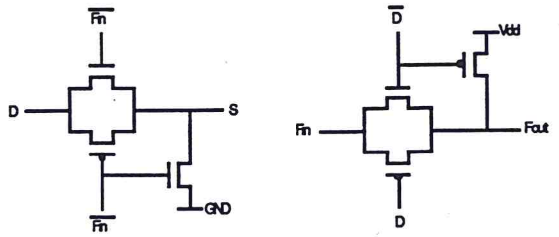


Figure 17 - Normalisation Decoder Circuit

Exponent Pipeline

The exponent pipeline is simply an eight bit wide storage path made up of the same number of pipeline stages as the main CORDIC pipeline so that the exponent keeps pace with the data to which it corresponds. The units that make up the pipeline are simply flip flops, of the same design as that used in the main pipeline so they will not be described in any more detail here.

Input and Output Circuits

The input and output circuitry was relatively simple to design as the pad layouts were supplied by ES2 and were simply used in that form to provide the external interface. The signals that were taken from the inputs pads and sent to the output pads have to be multiplexed due to our choice of interface schemes. This has been accomplished through the use of standard flip flops and multiplexers.

Putting It All Together

With the design of the major cells completed, it was necessary to combine them all into one design that performed the complete CORDIC algorithm as originally conceived. A first version of the processor has been produced by laying in the interconnect by hand which is necessary through the majority of the circuit as the constraints on the dimensions of the circuit are too tight to enable automatic routers to be used. With hindsight a fair amount of the larger scale routing through the Floating Point area of the chip could have been done with an automatic routing tool. The layout of the complete circuit has shown that the floating point circuitry is far more sparsely packed than it could be. This would probably be remedied by redesigning the barrel shifters to perform the shift operation in a straight through topology rather than the right angle turn that is currently implemented. The pipeline itself has been designed with only a small amount of unused space due to the routing of the shifts above the pipeline circuitry.

The full CORDIC circuit has not been tested completely due to the lack of a means of obtaining a correspondence between a logic level simulation and the silicon layout level. Even a full logic level simulation of the processor would be a difficult task. It is intended that a small scale test chip will be produced that will test the functionality of the basic building blocks described in this paper. Work will also proceed on a logic level simulation of the full processor, using those elements, that will verify that our implementation of the algorithm is correct under all circumstances. The full layout of the CORDIC processor is shown in Figure 18.

References

1. *K.Lismore, G.J.Vaudin*, "Design and Application of a CORDIC Processor for Real Time Signal Processing", Warwick University Dept. of Computer Science, 1988.

2. *M.Ercegovac, T.Lang*, "Redundant and Online CORDIC: Application to Matrix Triangularisation and SVD", Computer Science Dept., UCLA, 1987.

3. *K.Jainandunsing, E.Deprettere*, "Solving Sets of Linear Equations for Real Time Signal Processing", Proc EUSIPCO-86.

4. *H.M..Ahmed*, "Signal Processing Algorithms and Architectures", Ph.D. Thesis, Dept. of Electrical Eng, Stanford University, 1982.

5. *F.Deprettere, P.Dewilde*, "Pipelined CORDIC Architectures for fast VLSI Filtering and Array Processing", Proc. ICASSP-84, pp 41.A.6.1-41.A.6.4, 1984

6. *J.R.Cavallaro, F.T.Luk*, "CORDIC Arithmetic for a SVD Processor", 8th Symposium on Computer Arithmetic, pp 215-22, 1987.

7. *J.Nash, S.Hansen*, "Modified Faddeev Algorithm for matrix Manipulation", SPIE Real Time Signal Processing VII, pp 39-46, 1984.

8. *J.H. Moreno, T.Lang*, "On Partitioning the Faddeev Algorithm", Proc. Int. Conf. Systolic Arrays, pp125-134, 1988.

9. *D.W.J.Walton*, "Increased stability of the Faddeev Algorithm using Semi-CORDIC operations", Computer Science Dept, Warwick University, Research Report.

10. *J.E.Volder*, "The CORDIC Trigonometric computing Technique", IRE Trans on EC, Vol EC-8,No3, 330-334.

11. *J.S.Walther*, "A unified Algorithm For Elementary Functions",Proc. Spring Joint Computer Conf. pp 379-385.

12. *J.Duprat, J.M.Muller*, "The CORDIC Algorithm: New Results for Fast VLSI Implementation", Laboratoire LIP, Ecole Normale Superieure de Lyon, Janvier 1990.

13. *G.L.Haviland, A.A.Tuszynski*, "A CORDIC Arithmetic Processor Chip", IEEE Trans Comp. C29, pp 68-79, 1980.

14. *J.M.Delosme*, "VLSI Implementation of Rotations in Pseudo Euclidean Spaces", IEEE Int. Conf. Acoustics, Speech & Signal Processing 2, pp 927-930, Boston, 1983.

15. *N.Takagi, T.Asada, S.Yajima*, "A Hardware Algorithm for Computing Sign and Cosine using Redundant Binary Representation", Systems and Computares in Japan, Vol 18, No 8, pp 1-9, Aug 1987.

16. *N.Takagi, T.Asada, S.Yajima*, "Redundant CORDIC Methods with a constant scale factor", Submitted to IEEE Trans. on Computers.

17. *T.Y.Sung, Y.H.Hu, H.J.Yu*, Doubly Pipelined CORDIC array for Digital Signal Processing", Proc. ICASSP, pp 22.6.1-22.6.4.

18. *P.M.Chown*, "Design of a Barrel Shifter for the Warwick Pipelined CORDIC Processor", Warwick University Dept. of Computer Science, Internal Report, 1990.