A CORDIC Arithmetic Processor Chip

GENE L. HAVILAND AND AL A. TUSZYNSKI

Abstract—**A monolithic processor computes products, quotients, and several common transcendental functions. The algorithms are based on the well-known principles of "CORDIC," but recourse to a subtle novel corollary results in a scale factor of unity. Compared to older machines, the overhead burden is significantly reduced. Also, expansion of the functional repertoire beyond the circular domain, i.e., addition to the menu of hyperbolic and linear operations, is a relatively trivial matter, in terms of both hardware cost and execution time. A bulk CMOS technology with conservative layout rules is used for the sake of high reliability, low-power consumption, and good cycle speed.**

INTRODUCTION

THOUGH the concept of CORDIC Arithmetic is said to be quite old [1], [4], its implementations and applications continue to evolve. The acronym comes from Volder's Coordinate Rotations Digital Computer [1], developed in 1959 for air navigation and control instrumentation. An avuncular idea, particularly effective in decimal radix computations, was presented by Meggit in 1962 [2], under the label of "pseudo-division and pseudomultiplication." In 1971, Walther [3] generalized elegantly the mathematics of CORDIC's, showing that the implementation of a wide range of transcendental functions can be fully represented by a single set of iterative equations. Cochran [4] benchmarked, about the same time, various algorithms and found that CORDIC techniques surpass alternative methods in scientific calculator applications.

The pertinent effort of the Naval Ocean Systems Center (NOSC) culminates in the CORDIC Arithmetic Processor Chip (CAP Chip) of Fig. 1 that simplifies the architecture, boosts the speed, and reduces the power consumption of monolithic arithmetic modules. All computations are based on the execution of either

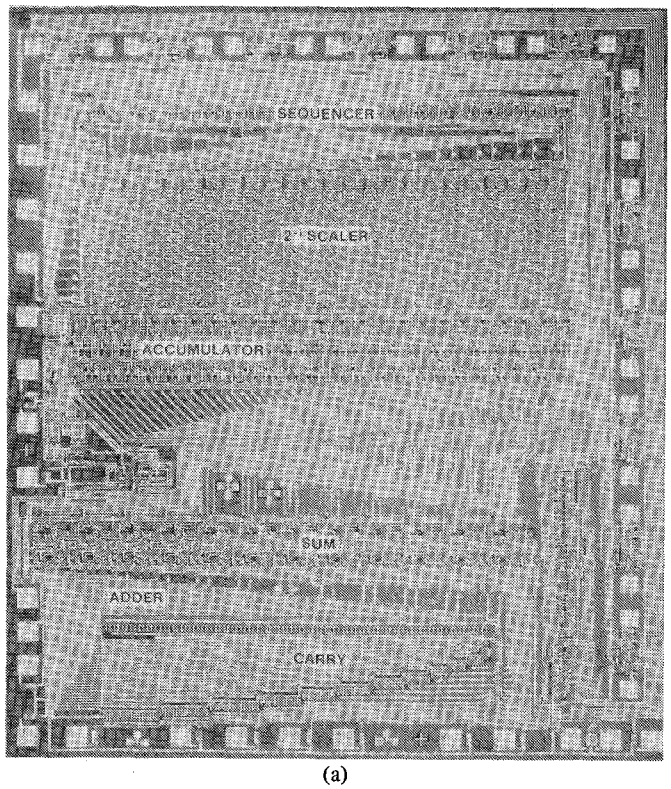
$$x_{i+1} = x_i \pm y_i 2^{-i} \tag{1a}$$

or

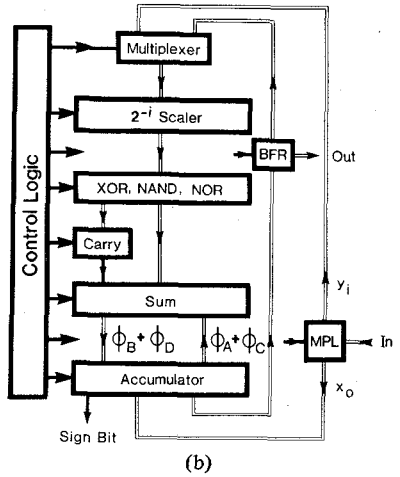
$$x_{(i+1),2} = (1 + \gamma 2^{-i})\, x_{(i+1),1}. \tag{1b}$$

While the first of these equations represents the regular CORDIC iterations, the second [5], [6] forces the scale factors of circular and hyperbolic functions to unity. ROM instructions govern the selection of either (1a) or (1b), but the ± option is executed by the sign bit of one of the operands.

This paper begins with multiplication and division, because the pertinent algorithm compares well, in its own right, with alternative techniques, especially in digital filter applications [7], [8], [17]. Moreover, the said algorithm is simple and transparent enough to project the feedback principle as the fundamental and common link of the CORDIC [1], [3],



(a)



(b)

Fig. 1. (a) The CAP chip. (b) Block diagram of the CAP chip.

Manuscript received January 4, 1979; revised June 26, 1979.

The authors are with the Microelectronic Circuit Design Branch, Naval Ocean Systems Center, San Diego, CA 92152.

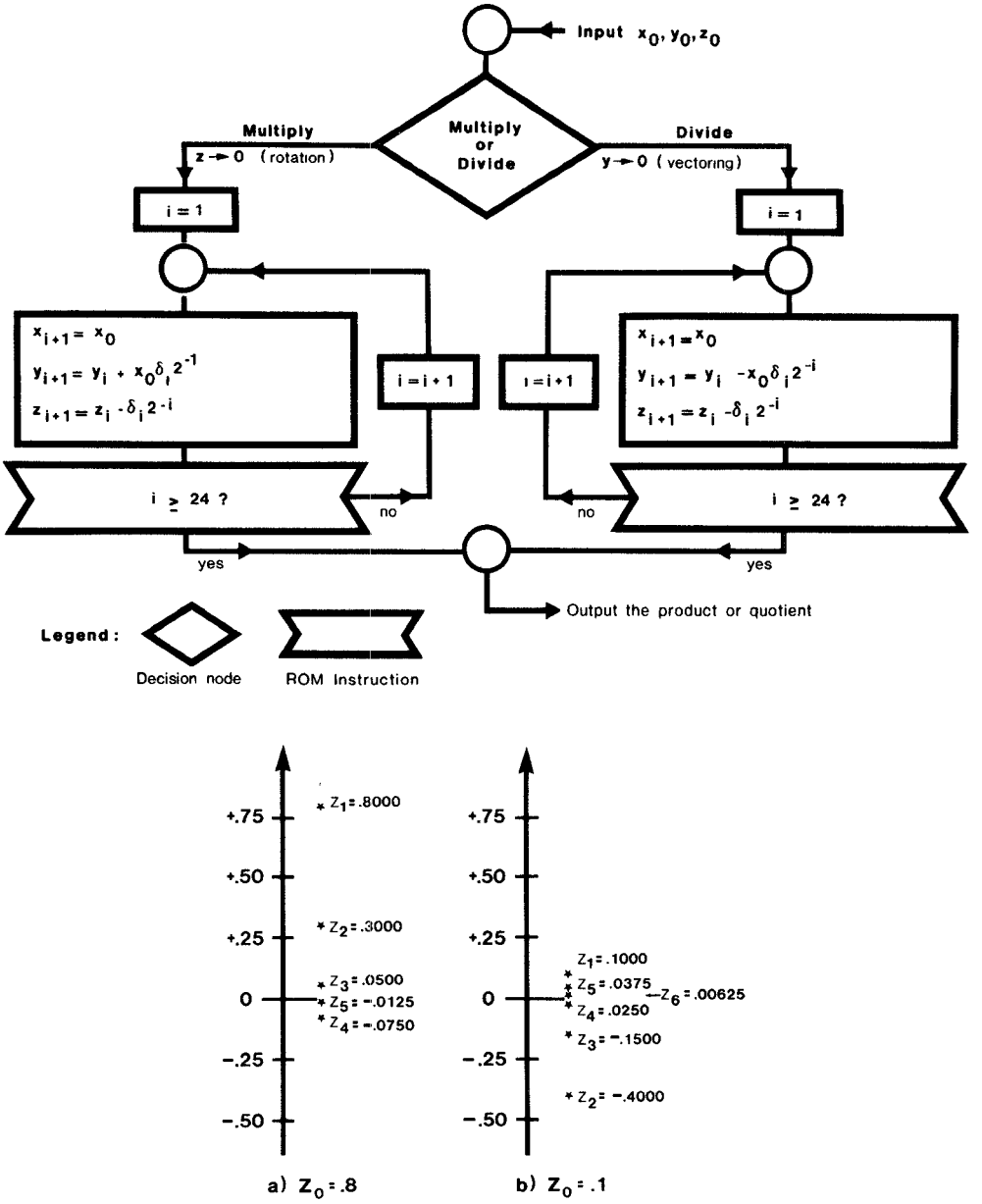


Fig. 2. (a) The multiply–divide algorithm. (b) Reduction to zero of operand z.

[4], the Meggitt [2] and the Chen [16] procedures. Also, once established, it can be easily expanded to trigonometric and hyperbolic functions.

A Digital Feedback Loop

Take three numbers, x_0, y_0, and z_0, z_0 being restricted to the range

$$0 \leqslant z_0 \leqslant 1. \tag{2}$$

Perform the following iterations:

$$y_{i+1} = y_i + x_0 \delta_i 2^{-i} \tag{3a}$$

$$= y_0 + x_0 \Sigma(\delta_i 2^{-i}) \quad \text{for } i = 1 \text{ through } n \tag{3b}$$

and

$$z_{i+1} = z_i - \delta_i 2^{-i} \tag{3c}$$

$$= z_0 - \Sigma(\delta_i 2^{-i}) \tag{3d}$$

but

$$x_{i+1} = x_i \tag{3e}$$

$$= x_0. \tag{3f}$$

To the δ_i operator assign the values of either plus one or minus one, depending on the polarity of z_i. In other words, let

$$\delta_i = \begin{cases} +1 & \text{if } z_i \geqslant 0 \\ -1 & \text{if } z_i < 0 \end{cases}. \tag{4}$$

A partial flow diagram of the above operation is given in Fig. 2(a), and a few steps of the z_i iteration are developed in Fig. 2(b). Note that

$$|z_{i+1}| < 2^{-i}, \tag{5}$$

although the magnitude of z_{i+1} is not necessarily smaller than that of z_i. The absolute value of z is gradually reduced towards zero, but the reduction may proceed zig-zag fashion.

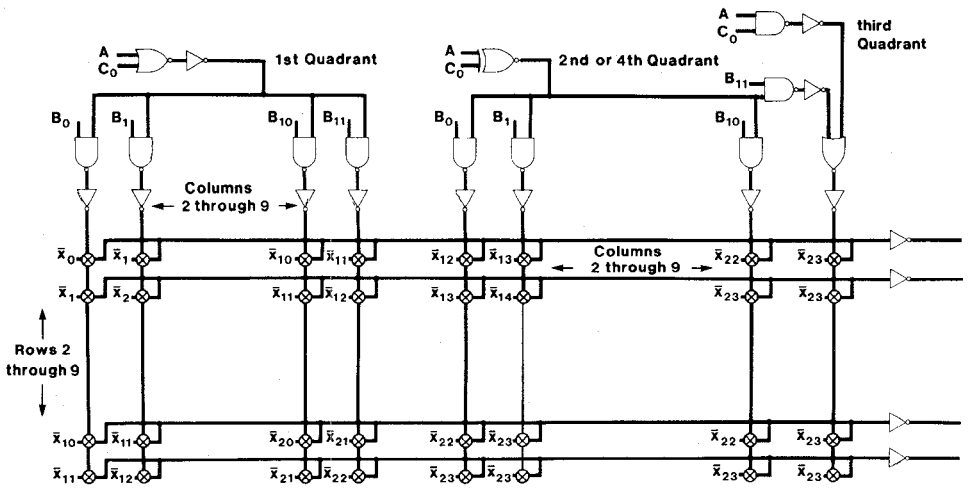


Fig. 3. Schematic diagram of the 2^{-i} scaler.

What we have here is an arrangement which amounts to an autonomous feedback loop of attractive simplicity. The zero-seeking mechanism of the loop is controlled entirely by the sign bit of z; the sign bit determines δ_i and that operator implements, in turn, the crucial add-subtract option of (3) and (4).

Equation (5) implies that a sufficiently high i, say $i = n$, will justify the approximation

$$z_{n+1} = z' \simeq 0 \tag{6a}$$

and will, therefore, lead to the expression

$$\sum_{i=1}^{n} (\delta_i 2^{-i}) \simeq z_0 \tag{6b}$$

and hence, by substitution into (3b), to the product equation

$$y_{n+1} = y' \simeq y_0 + x_0 z_0. \tag{6c}$$

It goes without saying that we could have reduced to zero the operand y rather than z. Exercising that option, one arrives at

$$y_{n+1} = y' \simeq 0 \tag{7a}$$

or

$$x_0 \left[\sum_{i=1}^{n} (\delta_i 2^{-i}) \right] \simeq -y_0 \tag{7b}$$

and, therefore, at the quotient equation

$$z' = z_0 + \frac{y_0}{x_0}. \tag{7c}$$

Equations (6) and (7) demonstrate that the digital feedback algorithm, defined by (2)–(4), leads to practical implementations of functions germane to multiplication and division [3]. Compared to alternative techniques [9], digital feedback looks good in division and, as we shall soon see, it becomes even more attractive when circular and hyperbolic functions are considered.

The Chip

Block diagram particulars and layout details of the CAP chip are shown in Fig. 1(a) and (b), respectively. There are but three major circuit blocks: the all important 2^{-i} scaler [11], a 12-bit two's complement adder, and a 24-bit accumulator of the shift-register variety. The narrow block at the top of the chip is the "i" counter, called the "sequencer." The I/O buffers are distributed around the periphery of the chip, but all multiplexers are merged with the appertaining functional blocks.

The 24-bit data are processed in two 12-bit steps. The lower byte of the word held by the accumulator is released into the adder-subtractor by the local clock, an intermediate step of addition or subtraction is performed, and the result is returned to the accumulator. The upper byte is subjected to similar treatment, beginning with the release of data that now includes the carry generated by the lower byte, and terminating with the acceptance of the result by the accumulator.

The scaler takes up a large part of the chip's surface and a sizable fraction of the cycle time. This is both understandable and acceptable considering its function, namely, the two's complement multiplication of every δ_i and every $x_0\delta_i$ by 2^{-i}. The scaler is indeed the centerpiece of CORDIC hardware; the present implementation is distinctly faster than its shift-register counterparts. The circuitry is really quite simple, owing to the highly efficacious transmission gates of the CMOS technology [12]. A matrix of such gates, arranged as shown in Fig. 3, propagates the sign bit while it shifts the data by "i" bits. The signal flow matrix of the scaler is square (Fig. 4) with 24 columns for bit locations and as many rows for cycle numbers. However, since there is some redundance in Fig. 4, the physical matrix need only be half as large as is its model, and that is why we have in Fig. 3 a matrix of 12 rows for 12 exponents and 24 columns for as many bits.

The transmission gates are driven by a sequencer with outputs A, B, and C (Fig. 5). Output A enables either the upper or the lower byte and output C picks the first or the second

Cycle #	Bit # 23 22 21 20 19 18 17 16 15 14 13 12	11 10 09 08 07 06 05 04 03 02 01 00
00	23 22 21 20 19 18 17 16 15 14 13 12	11 10 09 08 07 06 05 04 03 02 01 00
01	23 23 22 21 20 19 18 17 16 15 14 13	12 11 10 09 08 07 06 05 04 03 02 01
02	23 23 23 22 21 20 19 18 17 16 15 14	13 12 11 10 09 08 07 06 05 04 03 02
03	23 23 23 23 22 21 20 19 18 17 16 15	14 13 12 11 10 09 08 07 06 05 04 03
04	23 23 23 23 23 22 21 20 19 18 17 16	15 14 13 12 11 10 09 08 07 06 05 04
05	23 23 23 Quadrant II 20 19 18 17	16 15 14 Quadrant I 08 07 06 05
06	23 23 23 21 20 19 18	17 16 15 09 08 07 06
07	23 23 23 23 23 23 23 23 22 21 20 19	18 17 16 15 14 13 12 11 10 09 08 07
08	23 23 23 23 23 23 23 23 23 22 21 20	19 18 17 16 15 14 13 12 11 10 09 08
09	23 23 23 23 23 23 23 23 23 23 22 21	20 19 18 17 16 15 14 13 12 11 10 09
10	23 23 23 23 23 23 23 23 23 23 23 22	21 20 19 18 17 16 15 14 13 12 11 10
11	23 23 23 23 23 23 23 23 23 23 23 23	22 21 20 19 18 17 16 15 14 13 12 11
12	23 23 23 23 23 23 23 23 23 23 23 23	23 22 21 20 19 18 17 16 15 14 13 12
13	23 23 23 23 23 23 23 23 23 23 23 23	23 23 22 21 20 19 18 17 16 15 14 13
14	23	23 14
15	23	23 15
16	23 Quadrant III:	23 Quadrant IV: 16
17	23 Same as Row 11	23 Same as 17
18	23 of Quadrant II	23 Quadrant II 18
19	23	23 19
20	23	23 20
21	23	23 21
22	23 23 23 23 23 23 23 23 23 23 23 23	23 23 23 23 23 23 23 23 23 23 23 22
23	23 23 23 23 23 23 23 23 23 23 23 23	23 23 23 23 23 23 23 23 23 23 23 23

Fig. 4. Shifting of x_i and y_i.

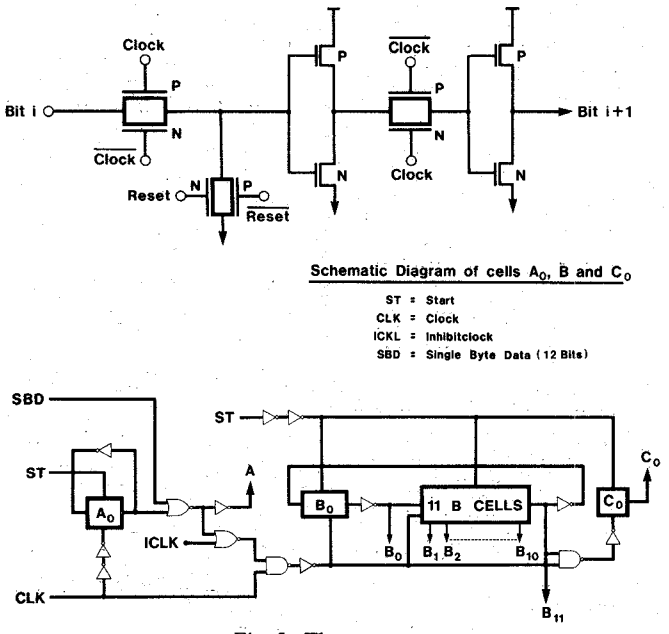


Fig. 5. The sequencer.

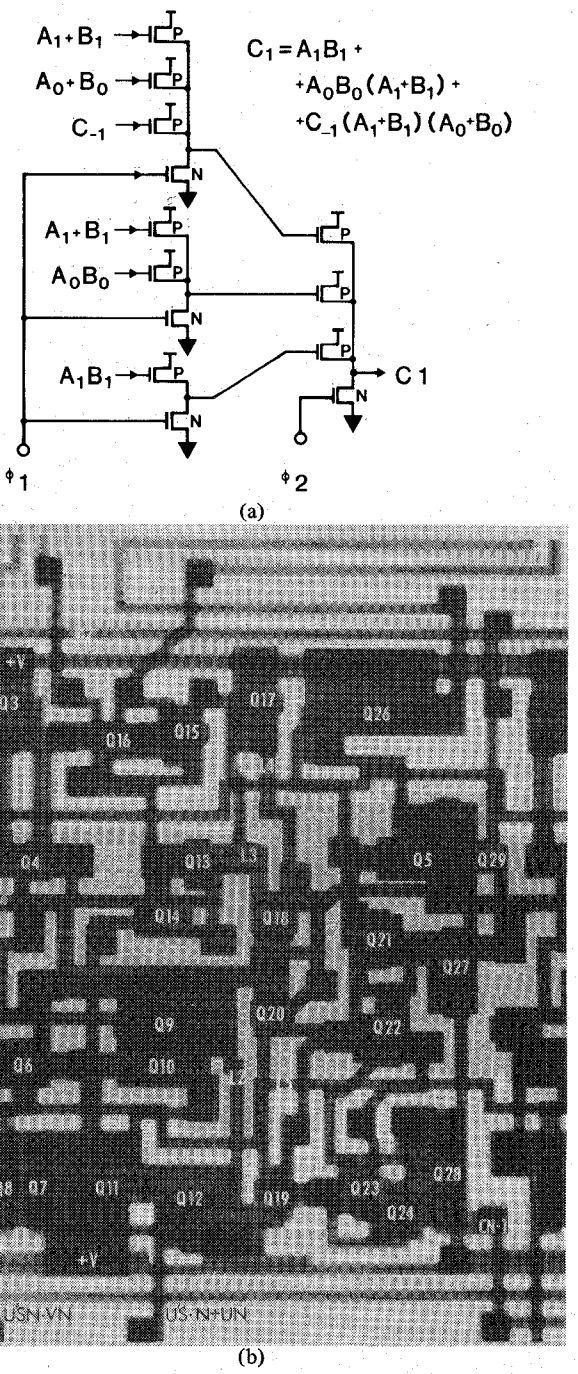


Fig. 6. (a) Dynamic CMOS, carry 1. (b) Fragment of the adder.

quadrant, while outputs B select one out of the 12 pertinent columns. Only regular CORDIC cycles are counted by the sequencer. Clock signals which pace the "double cycle" and the "scale factor" operations are inhibited by status bits outputted by the instruction ROM (Fig. 10).

The adder has a configuration which resembles conventional look ahead logic, but its circuitry is unique. Selected fragments of our "dynamic CMOS" circuits are shown in Fig. 6(a) and (b). Whereas there are $2n$ transistors in a conventional n-input CMOS gate, the complementary CMOS configuration of Fig. 6(a) has only $n + 1$. The resultant savings in surface area are most welcome in the CARRY module which has a total of 98 ports in the C_{12} gate. The precharge clocks ϕ_1 and ϕ_2 are, of course, synchronized with the clock which controls the timing of the lower and upper byte add-subtract operations.

The adder gate logic utilizes a combination of a XOR-AND-OR element and a HALF-ADDER. Various gate configurations, including the conventional CMOS NOR and the Floating XOR [13], are employed, but the whole thing adds up to only 26 transistors. The chip layout for this section of logic is shown in Fig. 6(b). 10 000 μ^2 of surface area are consumed if metal-gate bulk-CMOS with 8 μm spacing is employed.

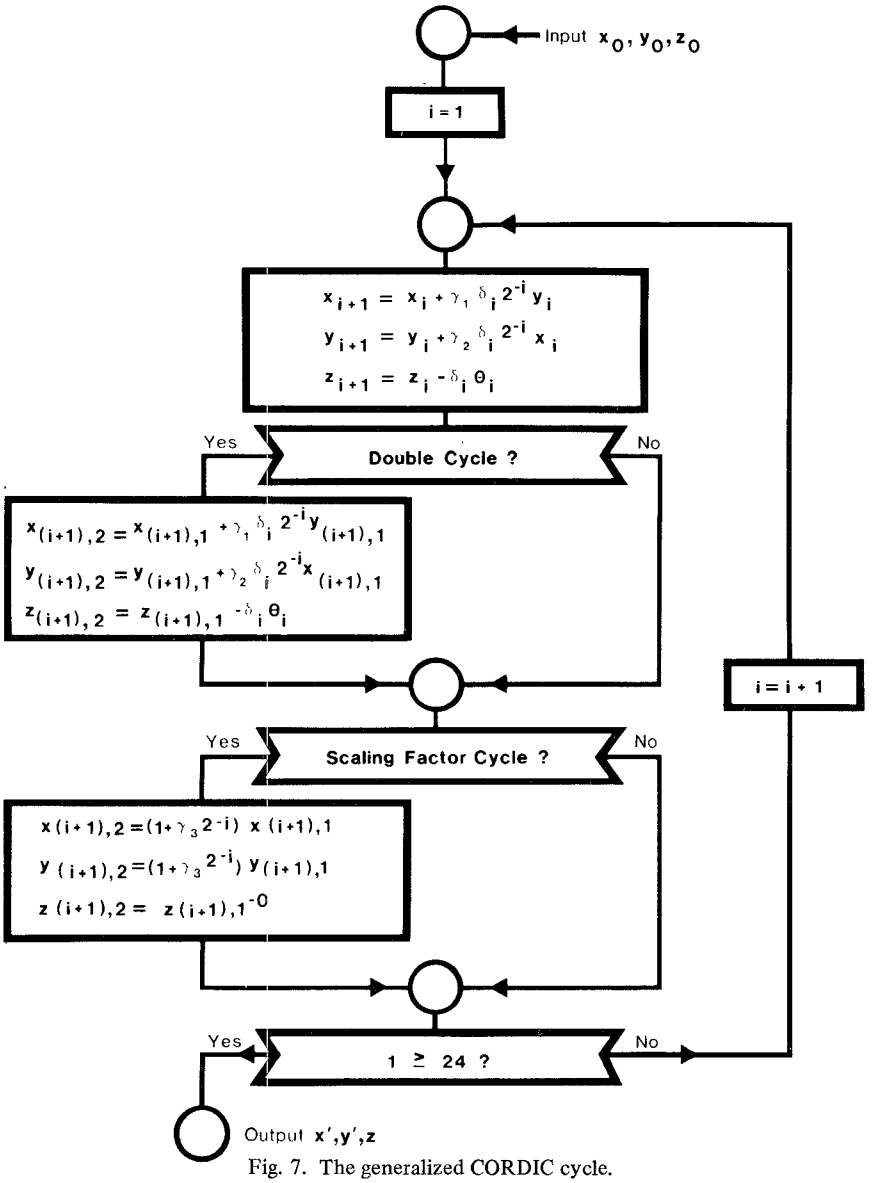


Fig. 7. The generalized CORDIC cycle.

General CORDIC Equations

Written in conventional format, the CORDIC equations look as follows:

$$x_{i+1} = x_i + \delta_i 2^{-i} y_i \tag{9a}$$

$$y_{i+1} = y_i - \delta_i 2^{-i} x_i \tag{9b}$$

and

$$z_{i+1} = z_i - \delta_i \theta_i. \tag{9c}$$

The CAP chip executes the above and two supplementary sets of equations:

$$x_{(i+1),2} = x_{(i+1),1} + \delta_i 2^{-i} y_{(i+1),1} \tag{10a}$$

$$y_{(i+1),2} = y_{(i+1),1} - \delta_i 2^{-i} x_{(i+1),1} \tag{10b}$$

$$z_{(i+1),2} = z_{(i+1),1} - \delta_i \theta_i \tag{10c}$$

and

$$x_{(i+1),2} = (1 + \gamma 2^{-i})\, x_{(i+1),1} \tag{11a}$$

$$y_{(i+1),2} = (1 + \gamma 2^{-i})\, y_{(i+1),1} \tag{11b}$$

$$z_{(i+1),2} = z_{(i+1),1} + 0. \tag{11c}$$

The flow diagram of the generalized instruction cycle is given in Fig. 7. Equations (10) and (11) represent the "double cycle" and the "scaling factor K" operations, respectively. The *raison d'etre* of these operations is explained below.

Let us focus our attention on the variable "i" in (9). We have already come across the relationship

$$\theta_i = 2^{-i} \tag{12a}$$

and will yet tackle the functions

$$\theta_i = \arctan(2^{-i}) \tag{12b}$$

and

$$\theta_i = \operatorname{arctanh}(2^{-i}). \tag{12c}$$

Implied in (6), as well as in the concept of feedback itself, is the convergence relationship:

TABLE I
THE K SCALING FACTOR FOR TRIGONOMETRIC FUNCTIONS

"i"	COSINE	PRODUCT
0	.7071067811865	.7071067811865
1	.8944271909999	.6324555320337
2	.9701425001453	.6135719910779
3	.9922778767137	.6088339125177
4	.9980525784829	.6076482562562
5	.9995120760871	.6073517701413
6	.9998779520347	.6072776440935
7	.9999694838188	.6072591122989
8	.9999923706928	.6072544793325
9	.9999980926568	.6072533210899
10	.9999995231632	.6072530315291
11	.9999998807907	.6072529591389
12	.9999999701977	.6072529410414
13	.9999999925494	.6072529365170
14	.9999999981374	.6072529353859
15	.9999999995343	.6072529351031
16	.9999999998836	.6072529350324
17	.9999999999709	.6072529350147
18	.9999999999927	.6072529350103
19	.9999999999982	.6072529350092
20	.9999999999995	.6072529350089
21	.9999999999999	.6072529350089
22	1.0000000000000	.6072529350088
23	1.0000000000000	.6072529350088
24	1.0000000000000	.6072529350088

"i"	K CYCLE	CORRECTION	MULTIPLIER	DIFFERENCE
1	0	.5000000000000	1.0000000000000	.3927470649912
2	2	.7500000000000	.7500000000000	.1427470649912
3	3	.8750000000000	.6562500000000	.0489970649912
4	4	.9375000000000	.6152343750000	.0079814399912
5	0	.9687500000000	.6152343750000	.0079814399912
6	0	.9843750000000	.6152343750000	.0079814399912
7	7	.9921875000000	.6104278564453	.0031749214365
8	8	.9960937500000	.6080433726311	.0007904376222
9	0	.9980468750000	.6080433726311	.0007904376222
10	10	.9990234375000	.6074495802750	.0001966452662
11	0	.9995117187500	.6074495802750	.0001966452662
12	12	.9997558593750	.6073012771548	.0000483421460
13	0	.9998779296875	.6073012771548	.0000483421460
14	14	.9999389648438	.6072642104265	.0000112754176
15	0	.9999694824219	.6072642104265	.0000112754176
16	16	.9999847412109	.6072549443100	.0000020093011
17	0	.9999923706055	.6072549443100	.0000020093011
18	0	.9999961853027	.6072549443100	.0000020093011
19	19	.9999980926514	.6072537860631	.0000008510542
20	20	.9999990463257	.6072532069407	.0000002719319
21	21	.9999995231628	.6072529173798	-.0000000176290

Required Product of Cosines = .6072529350088
K Multiplier for Cosines = .6072529173798
Error at 24 cycles of 40 bits = .0000000176290

$$\left| z_0 - \sum_{i=1}^{n} (\delta_i \theta_i) \right| \leqslant \theta_n. \tag{13}$$

This inequality is fulfilled by (12a) and (12b) but not (12c), for the simple reason that

$$2^{-(i+1)} = (\tfrac{1}{2})(2^{-i}), \quad \text{i.e., term } i+1 = (\tfrac{1}{2}) \text{ of term } i \tag{14a}$$

and

$$\arctan [2^{-(i+1)}] > (\tfrac{1}{2}) \arctan (2^{-i}),$$
$$\text{i.e., term } i+ > \tfrac{1}{2} \text{ of term } i, \tag{14b}$$

but

$$\operatorname{arctanh} [2^{-(i+1)}] < (\tfrac{1}{2}) \operatorname{archtanh} (2^{-i}),$$
$$\text{i.e., term } i+1 < \tfrac{1}{2} \text{ of term } i. \tag{14c}$$

This is why we need the "double pass" operation when hyperbolic functions are being processed. Tables I-III illustrate the point at issue for the specific case of $n = 24$, showing that

$$\left[\operatorname{arctanh} (2^{-1}) - \sum_{i=2}^{24} \operatorname{arctanh} (2^{-i}) \right] \gg \operatorname{arctanh} (2^{-24}), \tag{15a}$$

but

$$\left\{ \operatorname{arctanh} (2^{-1}) - \left[\sum_{i=2}^{24} \operatorname{arctanh} (2^{-i}) + \sum_{k} \operatorname{arctanh} (2^{-k}) \right] \right\} < \operatorname{arctanh} (2^{-24}), \tag{15b}$$

if

$$k = 3, 4, 7, 12, 13, 18, 19, \text{and } 21. \tag{15c}$$

So much for the double pass capability. Simply put, some CORDIC operations are run twice, in order to comply with inequality (13).

The supplementary operations called out in (11) force the scale factor K to converge toward unity. While the regular iterations cross-link x and y, the scale factor K is adjusted by

TABLE II
COMPUTATION AND LISTING OF DOUBLE PASS CYCLES FOR HYPERBOLIC FUNCTIONS

"i"	Arctanh(2^{-i})	$\sum_{k=i+1}^{24} (\theta_k)$	Difference	"i"	DOUBLE PASS CYCLES	THETA	REMAINDER
24	.0000000596046		.0000000596046	1	0	.5493061443341	.0431429745373
23	.0000001192093	.0000000596046	.0000000596046	2	0	.2554128118830	.0431429745373
22	.0000002384186	.0000001788139	.0000000596046	3	0	.1256572141405	.0431429745373
21	.0000004768372	.0000004172325	.0000000596046	4	0	.0625815714770	.0431429745373
20	.0000009536743	.0000008940697	.0000000596046	5	1	.0312601784907	.0118827960466
19	.0000019073486	.0000018477440	.0000000596046	6	0	.0156262717521	.0118827960466
18	.0000038146973	.0000037550926	.0000000596046	7	1	.0078126589515	.0040701370951
17	.0000076293945	.0000075697899	.0000000596046	8	1	.0039062698684	.0001638672267
16	.0000152587891	.0000151991844	.0000000596046	9	0	.0019531274835	.0001638672267
15	.0000305175781	.0000304579735	.0000000596047	10	0	.0009765628104	.0001638672267
14	.0000610351563	.0000609755516	.0000000596047	11	0	.0004882812888	.0001638672267
13	.0001220703131	.0001220107079	.0000000596052	12	0	.0002441406299	.0001638672267
12	.0002441406299	.0002440810210	.0000000596088	13	1	.0001220703131	.0000417969136
11	.0004882812888	.0004882216509	.0000000596379	14	0	.0000610351563	.0000417969136
10	.0009765628104	.0009765029397	.0000000598707	15	1	.0000305175781	.0000112793354
9	.0019531274835	.0019530657501	.0000000617334	16	0	.0000152587891	.0000112793354
8	.0039062698684	.0039061932337	.0000000766347	17	1	.0000076293945	.0000036499409
7	.0078126589515	.0078124631021	.0000001958495	18	0	.0000038146973	.0000036499409
6	.0156262717521	.0156251220536	.0000011496984	19	1	.0000019073486	.0000017425923
5	.0312601784907	.0312513938057	.0000087846850	20	1	.0000009536743	.0000007889180
4	.0625815714770	.0625115722963	.0000699991807	21	1	.0000004768372	.0000003120808
3	.1256572141405	.1250931437733	.0005640703671	22	1	.0000002384186	.0000000736622
2	.2554128118830	.2507503579138	.0046624539692	23	0	.0000001192093	.0000000736622
1	.5493061443341	.5061631697968	.0431429745373	24	1	.0000000596046	.0000000140576

TABLE III
SUM OF ARCTANH (2^{-i}) INCLUDING DOUBLE PASS CYCLES

"i"	Double Pass Cycles	Arctanh(2^{-i})	$\sum_{k=i+1}^{24} (\theta_k)$	$\theta_i - \sum_{k=i+1}^{24} (\theta_k)$
24	0	.0000000596046		.0000000596046
24	1	.0000000596046	.0000000596046	
23	0	.0000001192093	.0000001192093	-.0000000000000
22	0	.0000002384186	.0000002384186	-.0000000000000
22	1	.0000002384186	.0000004768372	-.0000002384186
21	0	.0000004768372	.0000007152557	-.0000002384186
21	1	.0000004768372	.0000011920929	-.0000007152557
20	0	.0000009536743	.0000016689301	-.0000007152557
20	1	.0000009536743	.0000026226044	-.0000016689301
19	0	.0000019073486	.0000035762787	-.0000016689301
19	1	.0000019073486	.0000054836273	-.0000035762787
18	0	.0000938146973	.0000073909760	-.0000035762787
17	0	.0000076293945	.0000112056732	-.0000035762787
17	1	.0000076293945	.0000188350677	-.0000112056732
16	0	.0000152587891	.0000264644623	-.0000112056732
15	0	.0000305175781	.0000417232513	-.0000112056732
15	1	.0000305175781	.0000722408295	-.0000417232513
14	0	.0000610351563	.0001027584076	-.0000417232513
13	0	.0001220703131	.0001637935639	-.0000417232508
13	1	.0001220703131	.0002858638770	-.0001637935639
12	0	.0002441406299	.0004679341902	-.0001537935603
11	0	.0004882812888	.0006520748200	-.0001637935312
10	0	.0009765628104	.0011403561088	-.0001637932984
9	0	.0019531274835	.0021169189192	-.0001637914357
8	0	.0039062698684	.0040700464028	-.0001637765344
8	1	.0039062698684	.0079763162712	-.0040700454028
7	0	.0078126589515	.0118825861396	-.0040699271880
7	1	.0078126589515	.0196952450911	-.0118825861396
6	0	.0156262717521	.0275079040426	-.0118816322906
5	0	.0312601784907	.0431341757947	-.0118739973040
5	1	.0312601764907	.0743943542854	-.0431341757947
4	0	.0625815714770	.1056545327760	-.0430729612990
3	0	.1256572141405	.1682361042530	-.0425788901126
2	0	.2554128118830	.2938933183935	-.0384805065105
1	0	.5493061443341	.5493061302765	.0000000140576

separate, though identical, manipulations of x and y. For example, setting the gammas in (11) to plus 1 for $i = 2, 4$, one multiplies both output variables by 1.32812:

$$x^* = (1 + 2^{-2})(1 + 2^{-4})x$$
$$= 1.32812x \tag{16a}$$

and

$$y^* = (1 + 2^{-2})(1 + 2^{-4})y$$
$$= 1.32812y. \tag{16b}$$

The role of the scale factors in the realization of circular and hyperbolic functions will be discussed in a later section.

CIRCULAR FUNCTIONS

Prominent among the functions used in servo control is the resolver operation defined by (17) [10]. The search for "solid-state" resolver hardware is lively and likely to continue for some time to come. Mathematically, however, one deals with the old and commonplace rotation of axes depicted in Fig. 8. When a pair of rectangular axes is rotated anticlockwise by an angle θ, then the coordinates of a point P transform

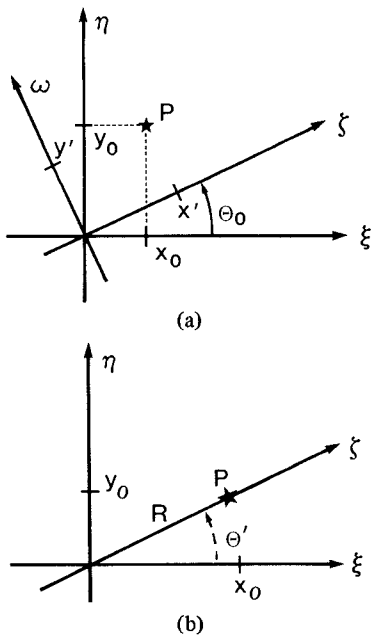


Fig. 8. (a) "Rotation," given x_0, y_0, and θ_0 find x' and y'. (b) "Vectoring," given x_0 and y_0 find R and θ'.

from x_0, y_0 to x, y in accordance with the equations:

$$x = x_0 \cos\theta + y_0 \sin\theta \tag{17a}$$

$$y = -x_0 \sin\theta + y_0 \cos\theta. \tag{17b}$$

Interesting, from the viewpoint of implementation, is the fragmentation property of θ: Theta can be split up into an arbitrary number of other angles. For example, if

$$\theta = \theta_1 + \theta_2, \tag{18a}$$

then

$$x_1 = x_0 \cos\theta_1 + y_0 \sin\theta_1 \tag{18b}$$

$$y_1 = -x_0 \sin\theta_1 + y_0 \cos\theta_1 \tag{18c}$$

and consequently,

$$x = x_1 \cos\theta_2 + y_1 \sin\theta_2 \tag{19a}$$

$$= x_0 (\cos\theta_1 \cos\theta_2 - \sin\theta_1 \sin\theta_2) + y_0 (\sin\theta_1 \cos\theta_2 + \sin\theta_2 \cos\theta_1) \tag{19b}$$

$$= x_0 \cos(\theta_1 + \theta_2) + y_0 \sin(\theta_1 + \theta_2). \tag{19c}$$

Interpretation of this result in terms of multiple fragmentations leads to a set of recursive formulas, which read as follows:

$$x_{i+1} = x_i \cos\delta_i\theta_i + y_i \sin\delta_i\theta_i \tag{20a}$$

$$y_{i+1} = -x_i \sin\delta_i\theta_i + y_i \cos\delta_i\theta_i \tag{20b}$$

and

$$z_{i+1} = z_i - \delta_i\theta_i. \tag{20c}$$

There are no restrictions on the various θ's, other than those considered in (10)-(14), in connection with the double cycle operation. For that matter, (20c) is exactly the same as (9c), though there are significant discrepancies between the other members of sets (9) and (20). These discrepancies will now be eliminated as much as possible, for the sake of hardware simplicity.

First off, one can factorize $\cos\delta_i\theta_i$ in (20a) and (20b):

$$x_{i+1} = \cos\delta_i\theta_i(x_i + y_i \tan\delta_i\theta_i) \tag{21a}$$

$$= \cos\theta_i(x_i + \delta_i y_i \tan\theta_i) \tag{21b}$$

and

$$y_{i+1} = \cos\theta_i(y_i - \delta_i x_i \tan\theta_i). \tag{21c}$$

Next, one can make the arbitrary, but highly convenient, substitution

$$\theta_i = \arctan(2^{-i}) \tag{22}$$

in order to arrive at

$$x_{i+1} = \cos\theta_i(x_i + \delta_i 2^{-i} y_i) \tag{23a}$$

$$y_{i+1} = \cos\theta_i(y_i - \delta_i 2^{-i} x_i) \tag{23b}$$

$$z_{i+1} = z_i - \delta_i \arctan(2^{-i}). \tag{23c}$$

Finally, one can compare the end results (x^*, y^*, z^*) of iterations (23) with the end results (x', y', z') of iterations (9) and conclude that

$$x^* = x'\left[\prod_{i=0}^{n} \cos(\arctan 2^{-i})\right] \tag{24a}$$

$$= x'\left[\prod_{i=0}^{n} (1 + 2^{-2i})^{-1/2}\right] \tag{24b}$$

$$= K_n x'. \tag{24c}$$

That spells out the overall dependence between the two sets of numbers as

$$x^* = K_n x' \tag{24d}$$

and

$$y^* = K_n y' \tag{24e}$$

but

$$z^* = z'. \tag{24f}$$

Since it depends on "n" only, the scale factor K_n is a machine constant. Consequently, given x' and y', one can realize x^* and y^* by many simple methods, including ROM look-up tables and regular combinatorial logic, but the scaling factor K technique, spelled out in (24) and Fig. 7, is particularly attractive because it offers a host of advantages such as speed, real estate economy, and conceptual simplicity.

Initialization

While the inverse tangent of 2^{-1} is only 26°, the processor must accommodate angles as large as ±180°. This does not present any great difficulty, but for the sake of compatibility with other functions, it is convenient to implement the range extension in two special "initialization" cycles (Fig. 9). The first shifts θ by 90°:

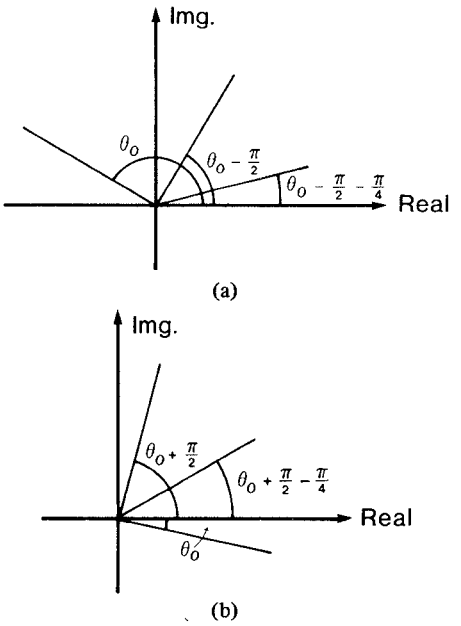


Fig. 9. (a) Initialization with $\theta_0 = 150°$. (b) Initialization with $\theta_0 = -15°$.

$$x = x_0 \cos(90°) + y_0 \sin\delta(90°) \tag{25a}$$

$$= \delta y_0 \tag{25b}$$

$$y = -x_0 \sin\delta(90°) + y_0 \cos(90°) \tag{25c}$$

$$= -\delta x_0 \tag{25d}$$

and

$$z = z_0 - \delta(90°). \tag{25e}$$

The second cycle executes the 45° shift,

$$x = x \cos(45°) + y \sin\delta(45°) \tag{26a}$$

$$= \frac{1}{\sqrt{2}}(x + \delta y) \tag{26b}$$

$$y = \frac{1}{\sqrt{2}}(y - \delta x) \tag{26c}$$

and

$$z = z - \delta(45°). \tag{26d}$$

The $1/\sqrt{2}$ multiplier in (26) had been actually anticipated and incorporated into the geometric K_n, when (24c) was written as

$$K_n = \sum_{i=0}^{n} (1 + 2^{-2i})^{-1/2}. \tag{27}$$

Equation (26) can be normalized, therefore, to read

$$x = x + (\delta) y \tag{28a}$$

$$y = y - (\delta) x \tag{28b}$$

and

$$z = z - (\delta)(45°). \tag{28c}$$

Hyperbolic Functions

When they are written in vector format, (17) read as follows:

$$\begin{vmatrix} x^* \\ y^* \end{vmatrix} = \begin{vmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{vmatrix} \begin{vmatrix} x_0 \\ y_0 \end{vmatrix}. \tag{29}$$

The coefficient matrix is orthogonal, and so are the three germane matrices shown below:

$$A_1 = \begin{vmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{vmatrix} \tag{30}$$

$$A_2 = \begin{vmatrix} \cos\theta & -j\sin\theta \\ -j\sin\theta & \cos\theta \end{vmatrix} \tag{31}$$

$$A_3 = \begin{vmatrix} \cos\theta & -j\sin\theta \\ -j\sin\theta & \cos\theta \end{vmatrix}. \tag{32}$$

Any one of these matrices can be used in expressions equivalent to (29) but, naturally enough, a unique geometrical interpretation must be associated with any particular matrix. For example, taking A_3 and relating it to the imaginary angle

$$\theta = j\phi, \tag{33}$$

one gets the hyperbolic relationship:

$$\begin{vmatrix} x^* \\ y^* \end{vmatrix} = \begin{vmatrix} \cos j\phi & -j\sin j\phi \\ -j\sin j\phi & \cos j\phi \end{vmatrix} \begin{vmatrix} x_0 \\ y_0 \end{vmatrix} \tag{34a}$$

$$= \begin{vmatrix} \cosh\phi & \sinh\phi \\ \sinh\phi & \cosh\phi \end{vmatrix} \begin{vmatrix} x_0 \\ y_0 \end{vmatrix} \tag{34b}$$

$$= \cosh\phi \begin{vmatrix} 1 & \tanh\phi \\ \tanh\phi & 1 \end{vmatrix} \begin{vmatrix} x_0 \\ y_0 \end{vmatrix}. \tag{34c}$$

The last of these equations leads directly to the iterative formulas

$$x_{i+1} = x_i + (\delta_i 2^{-i}) y_i \tag{35a}$$

$$y_{i+1} = y_i + (\delta_i 2^{-i}) x_i \tag{35b}$$

$$z_{i+1} = z_i - \delta_i \tanh(2^{-i}) \tag{35c}$$

and to the scale factor

$$K_n = \prod_{i=1}^{n} \cosh(2^{-i}) \tag{36a}$$

$$= \prod_{i=1}^{n} (1 - 2^{-2i})^{-1/2}. \tag{36b}$$

The hyperbolic routine does not require initialization, but it does call for the "double cycle" operations of (10). Tables II and III, drawn for $n = 24$, show 9 double cycles and 11 scale factor operations; the former produce an overall shift in θ of

$$\theta(\max) = z_0(\max) = 1.09 \tag{37}$$

while the latter generate the scale factor

$$K_{24} = 1.205. \tag{38}$$

Operations which reduce z to zero (rotations) implement the transformation

$$x^* = x_0 \cosh \alpha + y_0 \sinh \alpha \tag{39a}$$

and

$$y^* = x_0 \sinh \alpha + y_0 \cosh \alpha \tag{39b}$$

where

$$\alpha \simeq \theta_0 \tag{39c}$$

or alternatively, generate the functions sinh α, cosh α, e^{α}, $e^{-\alpha}$, etc., when appropriate values are assigned to the input variables x_0 and y_0 [2]. Vector operations, on the other hand, produce

$$z^* = z_0 + \operatorname{arctanh}(x_0/y_0) \tag{40a}$$

and

$$x^* = (x_0^2 - y_0^2)^{-1/2} \tag{40b}$$

as well as perspicuous mutations of these functions.

Overview

Figs. 10 and 11 show a complete set-up for the execution of rotation and vectoring operations in the linear, circular, and hyperbolic modes. There are six modules, namely, three CAP chips, a 16 X 512 ROM, an ROM ADDRESS counter, a clock, and an I/O box. The I/O box loads the data into the processor and returns the results to the bus; it also sets the two most significant bits of the ROM address. These two bits ($A9$ and $A8$) select one of the three sectors of the ROM beginning at address zero for linear operations, address 128 for circular functions, and 256 for hyperbolics. The counter, which generates the other 7 address bits, is first reset and then allowed to advance one bit per block cycle. The operation of the CAP chips is under the control of the instruction section of the ROM: Status bits from the ROM cause the execution of either a regular CORDIC cycle, or a "double" cycle, or a scaling factor K operation; they also signal arrival at the "last" cycle, that is, completion of the computation. The sign bit of either y or z controls the add/subtract options of all three chips.

The external instruction which activates the processor must include three function selection bits: one for either rotation or vectoring and two for either linear or trigonometric or hyperbolic functions. These three bits take care of the two decisions which start-off the signal flow diagram of Fig. 11. Once the function to be executed has been identified, the operation of the calculator is paced along by the local clock. Linear processing is purely "CORDIC," but the trigonometric routines add scaling factor K cycles to the menu, while hyperbolic algorithms use both the double cycle and scaling factor K supplements. The execution time of circular functions is slightly longer than that of linear functions, and the execution time of hyperbolic functions is longer still, but the implementation

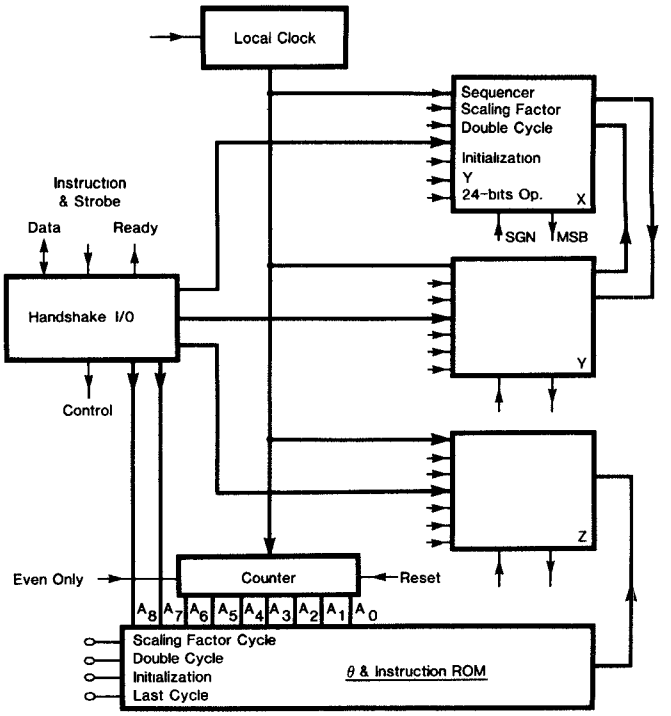


Fig. 10. System organization.

of all three classes of functions is equally simple. Simplicity, of both architecture and circuitry, may indeed be the most striking and important feature of the CAP chip implementation of the CORDIC concept. Where reliability is at a premium, nothing scores higher than well-founded simplicity.

Conclusion

Whereas the performance of a chip depends on its architecture, circuit design, and processing, one may want to separate processing from the other two factors when attempting to assess the quality of a device. It is understood that performance is always technology limited. A faster technology will invariably bring about higher speed and, possibly, reduce power dissipation at the same time. For a given circuit schematic, conversion from conservatively laid out metal-gate CMOS to tightly spaced poly-gate SOS will produce spectacular improvement. For that reason, speed alone is hardly a satisfactory measure of circuit design quality; to compare different embodiments of an idea, one must speak of minimum cycle times, expressed in multiples (n) of "typical" gate delays. The propagation delay of a gate sums up the quality of the technology, while "n" gives an estimate of the combined quality of the architecture and the circuit design. Naturally, one needs a definition of the "typical" gate. Physical dimensions present no difficulty—one simply picks a "minimum size" device—but the typical configuration may be open to dispute. We use an inverter with a fan out of three.

SPICE analysis [15] of the CAP chip suggests $n = 13$ as the minimum cycle time of a two-byte (24-bit) operation. The "gate" delay is roughly 100 ns. More than half of the overall delay is attributable to the 2^{-i} scaler. This is understandable,

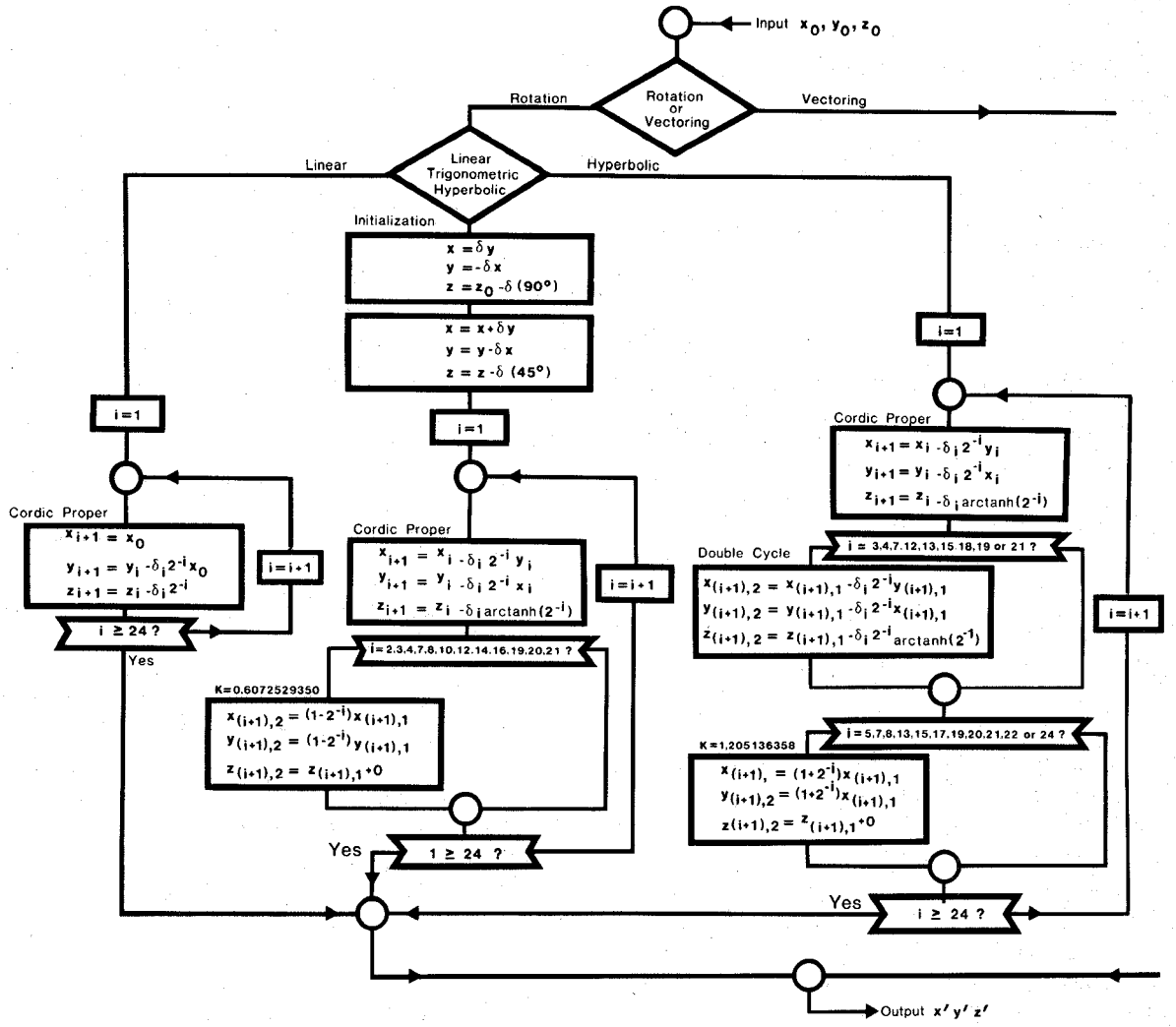


Fig. 11. Flow diagram of the CORDIC system.

considering the size of the structure in Fig. 3. Next in order of nuisance ratings comes the carry circuit $C12$, whose layout is shown in Fig. 6(b). Taken together, the scaler and the carry determine, just about, the effective "n" of the system. Further improvements in "n" will have to come either from modifications of these elements, or from conversion to single-byte operation. The former approach must await inventive contributions, but the latter is feasible right now. The entire system can be implemented in single-byte format by recourse to three micron layout rules; an "n" of 7.5 can thus be realized without changes in circuitry. Furthermore, even an early vintage edition of submicron CLOSED COSMOS [14] will accommodate a complete single-byte system on just one chip. What we have then, in addition to a system which executes transcendental functions in 40 μs, is a good candidate for submicron phototyping.

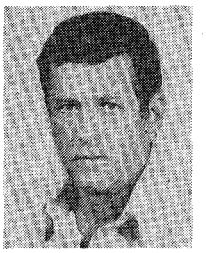
Acknowledgment

The authors acknowledge their indebtedness to C. A. Bass, Associate Head for Systems, Systems/Aerospace Surveillance Department, Naval Ocean Systems Center, San Diego, CA.

References

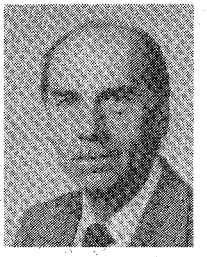
[1] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. EC-8, pp. 330–334, Sept. 1959.
[2] J. E. Meggitt, "Pseudo division and pseudo multiplication processes," *IBM J.*, pp. 210–226, Apr. 1962.
[3] J. S. Walther, "A unified algorithm for elementary functions," in *1971 Proc. Joint Spring Comput. Conf.*, pp. 379–385, 1971.
[4] D. S. Cochran, "Algorithms and accuracy in the HP-35," *Hewlett-Packard J.*, pp. 10–11, June 1972.
[5] C. A. Bass, "A flow-diagram technique for time-varying coordinate systems," NRL Rep. 7199, June 26, 1970.
[6] C. A. Bass and J. F. Kohne, Jr., "Computer program design specification for the AN/WSC-2 digital control system," NELC Tech. Note #2953, NOSC, San Diego, CA, May 30, 1975.
[7] A. Habibi and P. A. Wintz, "Fast multipliers," *IEEE Trans. Comput.*, vol. C-19, pp. 153–157, Feb. 1970.
[8] M. Davis and G. Bioul, "Fast parallel multiplication," *Philips Res. Rep.*, vol. 32, no. 1, pp. 44–70, 1977.
[9] H. Schmid, *Decimal Computation.* New York: Wiley, 1974.
[10] S. A. Davis and B. K. Ledgerwood, *Electromechanical Components for Servomechanisms.* New York: McGraw-Hill, 1961.
[11] D. G. Steer and S. R. Penstone, "Digital hardware for sine-cosine function," *IEEE Trans. Comput.*, vol. C-26, pp. 1283–1286, Dec. 1977.
[12] E. A. Torrero, "Focus on IC analog switches and multiplexers," *Electron. Des.*, pp. 64–72, Sept. 1975.

[13] RCA Part #CD4030.
[14] A. G. R. Dingwall and R. E. Sticker, "C^2L: A new high-speed high-density bulk CMOS technology," *IEEE J. Solid-State Circuits*, vol. SC-12, pp. 344-349, Aug. 1977.
[15] L. W. Nagel, "Spice 2: A computer program to simulate semiconductor circuits," Univ. of California, Berkeley, ELR, 1975.
[16] T. C. Chen, "Automatic computation of exponentials, logarithms, ratios and square roots," *IBM J. Res. Develop.*, pp. 380-388, July 1972.
[17] W. H. Specker, "A class of algorithms for ln x, exp x, sin x, cos x, $\tan^{-1} x$," *IEEE Trans. Comput.*, vol. C-19, pp. 153-157, Feb. 1970.



Gene L. Haviland received the B.S.E.E. degree from the California Polytechnic University, Pomona, in 1962.

In 1962, he joined the Applied Research Laboratory, Guidance and Control Division, Litton, IN, where he was actively involved in circuit design and hybrid computer development for inertial navigation systems. In 1967 he joined the Semiconductor Division of Union Carbide and subsequently became responsible for linear IC design and development. Since 1972 he has headed the Microelectronic Circuit Design Branch, Naval Ocean Systems Center, San Diego, CA, responsible for design techniques based on large-scale integration primarily in the area of digital systems.



Al A. Tuszynski received the B.S. degree from the University of London, London, England, the M.S. and D.Eng. Sc. degrees from the New Jersey Institute of Technology, Newark, NJ.

After 20 years with various system and semiconductor companies, he joined the Department of Electrical Engineering, University of Minnesota, Minneapolis, in 1970, to explore and teach techniques for the design and testing of large monolithic circuits. His consulting association with the Microelectronics Division of the Naval Ocean Systems Center, San Diego, CA, dates back to 1974.

Dedicated LSI for a Microprocessor-Controlled Hand-Carried OCR System

MICHEL C. RAHIER, MEMBER, IEEE, AND PAUL G. A. JESPERS, SENIOR MEMBER, IEEE

Abstract—**The binary picture processing and recognizing stages of an optical character recognition (OCR) system have been designed using both flexibility of available microprocessors and speed of peripheral custom-designed integrated circuits. A dedicated large-scale integrated (LSI) processor performs edge detection and thinning of a 32 × 24 digitized one-piece pattern. The output signal—a set of 3 bit vectors describing the skeletonized character contour—feeds a microprocessor which controls the character recognition algorithm including pattern segmentation, filtering, feature extraction, and classification decision. This low-cost equipment is especially suitable for hand-carried OCR systems where well-formed printed alphanumerics are to be read. However, continously deformed patterns like carefully handprinted characters are recognized as well. A system reading speed of 100 characters/s (or 30 cm/s) can be achieved.**

Manuscript received May 29, 1979; revised September 11, 1979. This work was supported by an IRSIA fellowship. This work was presented at the 1979 International Solid-State Circuits Conference, Philadelphia, PA.

The authors are with the Microelectronics Laboratory, Catholic University of Louvain, Louvain-la-Neuve, Belgium.

I. Introduction

THE steadily growing use of computers in industrial and business environments involves a huge need for data-entry devices with document direct-reading capability without use of tedious high-cost keyboarding operations.

Currently available optical character recognition (OCR) equipments can roughly be divided into three main classes according to their complexity level. In the first one arbitrary controlled source documents, using bar codes or magnetic-ink supports [1], lead to fairly simple recognition schemes and related hardwares, but the algorithms can generally not be extended for real-world character reading. At the other end of the available OCR systems, large machines have been built in an attempt to recognize handwriting. For example, automatic mail sorting [2] was successfully achieved with handwritten postal zip-code classification even though the large variety of character shapes encountered and the random background noise often mixed with the numerals. This generally results in

0018-9200/80/0200-0015$00.75