Journal of VLSI Signal Processing 25, 107–112, 2000
© 2000 Kluwer Academic Publishers. Manufactured in The Netherlands.

The Story of Unified CORDIC

JOHN STEPHEN WALTHER
Design Technology Center of the Integrated Circuit Business Division of Agilent Technologies, Inc., 1501 Page Miller Road, Palo Alto, California 94304-1126

Abstract. The story is told of how the Unified CORDIC algorithm came to be developed, starting from an idea in the mind of a student and culminating in a product, a patent, a paper, a motorcade, and an award. Along the way, a unified algorithm for generating the elementary functions and the math co-processor industry in Silicon Valley were born.

This is the story of how the Unified CORDIC algorithm came to be developed. For me, the story began with an incident in 1965. I was studying elementary function evaluation for a college class in Fortran programming when it struck me that the trigonometric and hyperbolic functions are closely linked. It wasn't clear to me how at the time, but I was fascinated by this idea, and I mentally filed it away to look into later.

In 1966 I graduated from MIT and started to work at Hewlett-Packard Laboratories, which was just then being formed, in Palo Alto, California. After working on medical electronics projects for about two years, I became interested in work that was being done in the Electronics Research Lab on the 9100 desk-top calculator (see Fig. 1). The precursor to the 9100 was a breadboard prototype that was designed and built by Tom Osborne at home, probably around 1965. Tom approached HP with the idea of turning the invention into a product.

1. CORDIC in the 9100 Desk-top Calculator

The 9100 used CORDIC for trigonometric functions only. The logarithm, exponential, and square-root functions were calculated using algorithms called pseudo-division and pseudo-multiplication, published in 1962 by Meggit [1]. The hyperbolic functions were derived from the logarithm and exponential functions via well known identities. The CORDIC algorithms were used in a way that reduced the need for stored constants, although this caused the execution to be slower than methods that use more stored constants. However, this was considered to be a good trade-off since the read-only memory that was used for programs and constants was expensive (it consisted of discrete diodes soldered into a printed circuit board), and the calculator application did not require high performance.

The sine and cosine functions, for example, were not calculated directly, but rather, they were derived from the tangent function, which was calculated using CORDIC. To do this, the value 1.0 was loaded into a register X, 0 was loaded into a register Y, and the argument angle, a, was loaded into a register Z. The CORDIC rotation steps were then applied, which caused X_{after} to contain $K\cos(a)$, Y_{after} to contain $K\sin(a)$, and Z_{after} to contain 0. In these results the factor K, called a radius distortion factor, is present because the CORDIC rotation steps change the length of the X-Y vector somewhat from its original length of 1.0 in this example. However, this did not matter in the case of the 9100 since the next step was to calculate the tangent (by dividing Y_{after} by X_{after}) and this caused the K factor to cancel out. The sine function, for example, was calculated using the following sequence of operations, where T_1 through T_4 are intermediate results:

$$T_1 = \frac{Y_{\text{after}}}{X_{\text{after}}} \tag{1a}$$

$$T_2 = T_1 * T_1 \tag{1b}$$

$$T_3 = \frac{1}{T_2} \tag{1c}$$

$$T_4 = \sqrt{T_3 + 1} \tag{1d}$$

$$\text{SINE} = \pm\left(\frac{1}{T_4}\right) \tag{1e}$$

Figure 1. The Hewlett-Packard 9100 desktop calculator (Reprinted with kind permission of the HP museum of calculators).

A “restoring” form of CORDIC was used. That is to say, if when the Z register became negative after having the angle associated with a particular CORDIC rotation step subtracted from it, then the Z register would be restored to its previous value, and the rotation step in the X and Y registers would not be done. Since the 9100 was a decimal machine with 12 decimal digits for each of the internal registers, only six angle constants, $\arctan(10^{-k})$, for $k = 1, 2, \ldots, 6$, needed to be stored. Constants for decades 7 and up did not need to be stored because their mantissas are the same as for decade 6 when stored in normalized form (i.e., with the most significant digit being nonzero). Each angle constant might need to be subtracted from the Z register up to ten times for each digit position before the Z register became negative. The condition of Z becoming negative would trigger the restoring operation and the switch to the next angle constant. This process would then be repeated until all of the digits in the Z register were zeroed out.

The CORDIC algorithm fascinated me, and my study of it led me to read papers by Volder [2] and Liccardo [3]. I learned from Volder’s paper how to calculate the trigonometric functions, and I learned from Liccardo’s paper how to calculate the hyperbolic functions. I realized that the hyperbolic iteration equations described rotations in a hyperbolic coordinate system, as opposed to the circular coordinate system that applies to the trigonometric functions. I discovered that by writing the iteration equations as a set of difference equations in a form that has the coordinate system encoded as a parameter m, where $m = 1$ for circular and $m = -1$ for hyperbolic, that I could solve the set of difference equations to produce equations for the values in the X, Y, and Z registers after n arbitrary iterations. Furthermore, I discovered that I could derive equations for the angle constants, distortion constants, the domain of convergence, and produce a proof of convergence for all of the parameterized coordinate systems combined. Here, indeed, was a unified algorithm for the elementary functions.

2. Making CORDIC Faster

I wanted to see how fast I could calculate the elementary functions given practical amounts of hardware logic. I had in mind providing the HP-2116 computer, which had been recently developed in the Electronics Research Lab, with an elementary function calculator, analogous to providing a person with a desk-top calculator. Except now, speed would definitely be a concern. On the other hand, the cost of read-only memory was substantially less than when the 9100 was designed—now you could get one thousand bits of ROM in one integrated circuit package. So, I wrote a simulator in Fortran that simulated the X, Y, and Z registers, shifters, adders, a constant store, and a microprogram store. I set about the task of determining the performance of various hardware configurations.

The low cost of ROM meant that a modest number of constants relating to radius distortion could be stored. A 48-bit binary floating point format, consisting of 40 bits of mantissa and 8 bits of exponent, was chosen. The X, Y, and Z registers were each 48 bits long, which included 8 bits as guard bits in each register. The guard bits allow up to a few hundred round-off errors to accumulate without introducing more than a small fraction of a bit of error into the result’s 40-bit long mantissa. The mantissas of only 24 normalized constants relating to radius distortion needed to be stored, along with the mantissas of 24 normalized angle constants. Similar to the situation in the decimal case, the mantissas of the constants for bicades 25 and higher are the same as for bicade 24 when stored in normalized form, and, therefore, they do not need to be stored.

The modest number of constants relating to radius distortion is made possible by a property of the radius distortion for each CORDIC rotation step: the amount of the distortion for a rotation through a given angle in the negative direction is the same as the amount of distortion for a rotation through the same angle in the positive direction. This leads to the use of a "nonrestoring" method in which a rotation is always done at every step, except that at each step the direction of rotation might be negative or positive depending on whether the accumulated rotations have overshot or not overshot, respectively, the target angle. A restoring method would not allow using prestored constants relating to distortion because some steps would have a rotation and other steps would not, with the result that a different constant would have to be stored for each of the 2^{48} different rotate/don't-rotate patterns, which would be impractical.

Instead of dividing the result by the total radius distortion upon completion of the rotation steps, it is convenient, and faster, to calculate the sine function, for example, by loading the X register with the reciprocal of the total expected radius distortion $1/K$, so that at the end of the rotation steps the length of the X-Y vector is 1.0. Then the $\cos(a)$ and $\sin(a)$ can be read out directly from X_{after} and Y_{after}, respectively. It is actually these reciprocal distortion constants that are stored in read-only memory.

3. Handling Small and Large Arguments

For angle arguments that are much smaller than arctan (0.5), it is wasteful in time to start out with the courser incremental rotations. Instead, we start out with a small incremental rotation appropriate for the size of the angle argument. Because the mantissa of the angle argument is normalized, however, we still have to go through the sequence of the next 48 smaller incremental rotations. The reason that we store multiple reciprocal distortion constants is that a different one applies for each size of starting-out incremental rotation.

For angle arguments that are much larger that arctan (0.5), we use identities, called prescaling identities, to calculate a scaled-down argument which can be shown to give the same answer as the original argument, and then we apply the CORDIC rotation steps to the scaled-down argument.

4. CORDIC in a Floating Point Co-Processor for the HP 2116

The Fortran simulations of CORDIC showed that impressive performance could be achieved with practical amounts of hardware logic. Word of the simulation results spread to the Data Systems Division, and they approached me about building a floating point co-processor for the 2116 computer. They had lined up a customer, Leasco Timesharing Company, who wanted to buy 200 units. I was offered the project leader position to lead a crash program to develop the new product.

However, I was faced with a difficult decision. At that time, I was working toward my Ph.D. at Stanford University, after having received the Master of Science degree in Electrical Engineering there. There would not be time to both take classes and lead the project at the same time. I chose the project.

The project team grew to 15 people. The microprogramming was done by Dave Cochran, who had written the microprogram for the 9100 and who later wrote the microprogram for the HP-35 calculator [4]. The hardware was ably put together by Ken Peterson, aided by Chung Tung, who later went on to become general manager of the Advanced Product Division, which manufactured HP's line of hand-held calculator products. A new detailed hardware simulator was written by Jim Duley, who later became director of the Electronics Research Lab. The power supply was designed by Greg Justice, who later adapted the design for use in the 2100 computer. The power supply was the first switching-regulated power supply used in an HP computer product.

The floating point processor, which was given model number 2152A (see Fig. 2), consisted of ten 8-inch by 8-inch printed circuit boards, each containing about 50 integrated circuit packages, in a box about 12 inches high, 16 inches wide, and 24 inches deep. It weighed about 80 pounds. The logic was constructed out of Complementary Transistor Logic (CTL), a kind of emitter-coupled logic, produced by Fairchild Semiconductor. This is the same logic family that was used in the 2116 computer. The registers were built out of simple latches, since you could get four latches to a package, but only one flip-flop to a package. The logic drew 30 amps from the power supply. Most of

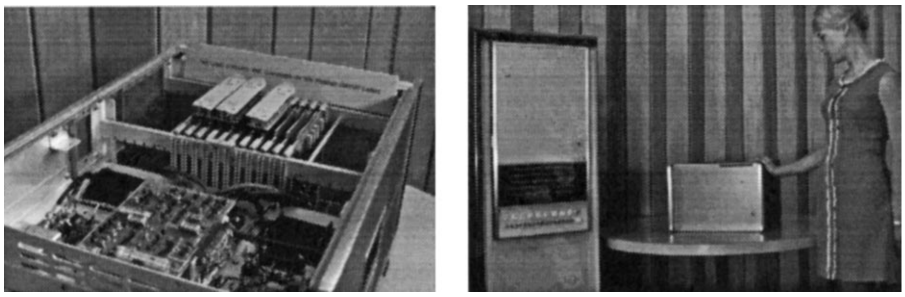


Figure 2. The Hewlett-Packard 2152A co-processor inside (left), and with the 2116 computer (right) (Hewlett-Packard Company, Reprinted with permission).

the power went into pull-down resistors, which were needed because in CTL there were active devices only for pulling up on signal lines, not down. The program memory was built out of thousand-bit ROM packages. The instruction register was 48 bits wide, containing control fields to independently control three identical arithmetic logic units, one each for the X, Y, and Z registers. The instruction register could be loaded from the 2116 computer in order to perform diagnostic tests to detect and diagnose logic faults. The diagnostic programs that were written were designed to detect and diagnose logic failures down to the gate level.

I applied for and was granted a patent [5] for the 2152A, with HP as the assignee. The patent, which was over 200 pages in length, contained all the information needed for someone skilled in logic design to recreate the hardware, including the microprogram. Although the patent was never used by HP in an infringement lawsuit, it has served as a reference for prior art in at least two lawsuits. I was told by one lawyer that, because of the patent, I had become known in patent-law circles as the father of the math co-processor industry that sprang up in Silicon Valley.

Nine months after the project began, Ken Peterson, Chung Tung, and I transferred to the Data Systems Division with the hardware in order to pass environmental tests and to help set up a production line. The environmental tests were completed successfully, and six production prototypes were made. Shortly after that, we learned that Leasco Timesharing Company had gone out of business, having succumbed to an economic recession that was occurring at that time. The 2152A project was canceled.

I do not know of any other HP product that has used the Unified CORDIC algorithm. The model 35 calculator (see Fig. 3) had similar design constraints as the 9100 calculator, except that it had low power as an added constraint, and so it used algorithms similar to those used in the 9100.

5. The Paper on Unified CORDIC

Word of the Unified CORDIC algorithm might never have reached the world if it were not for a fateful event that occurred in 1970. Barney Oliver, then the director of HP Labs, asked me to host Nathaniel Macon on a tour of the labs. I didn't know who Nathaniel

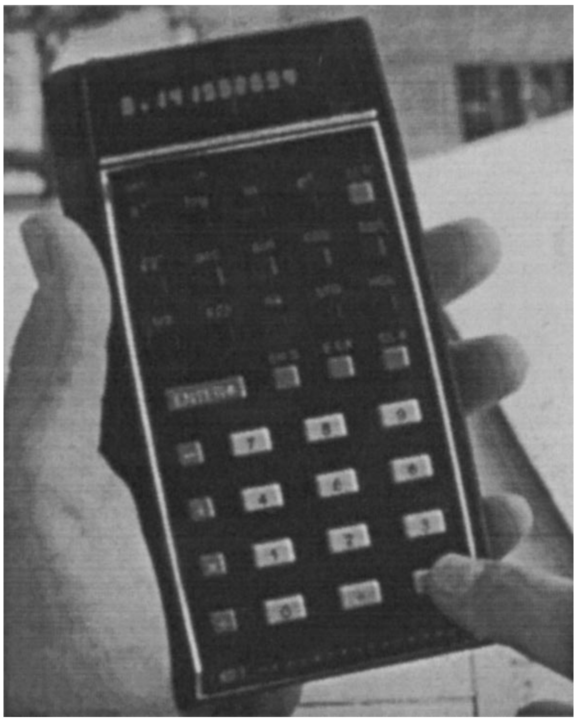


Figure 3. The Hewlett-Packard 35 calculator (Hewlett-Packard Company. Reprinted with permission).

Macon was, but, anyway, I arranged the tour, arranged lunch at the Palo Alto Hills Country Club, and invited Barney to join us for lunch. We had a nice tour, including a tour of the 2152A project, and at lunch Nat invited me to write a paper about my work on CORDIC for the Spring Joint Computer Conference, which was sponsored by the American Federation of Information Processing Societies. As it turned out, Nat was the technical program chairman for the conference. I said yes, and so I spent my Christmas vacation writing the paper [6]. A few months later I learned that I had won the conference's best paper award.

I went to the conference with my wife to present my paper and to attend the awards luncheon to receive my award. Imagine our astonishment when our ride to the awards luncheon turned out to be in a limousine in a motorcade with multiple motorcycles as escorts. Of course, I realized later, after Senator Sam Ervin gave the luncheon address, that the motorcade was for the senator, but that made it no less thrilling. As I proudly received my award, I realized that my 15 minutes of fame had come to me on that day.

The response to the paper was amazing. It was included in a book called "The Best Computer Papers of 1971" by Auerbach Publishers [7], and it was included in the textbook "Computer Arithmetic" by Hwang [8]. I received word that it was being taught in Computer Science classes on computer arithmetic, and for years afterward I received letters about the paper.

6. Making CORDIC Even Faster

A few years later, around 1975, when I was working on the IEEE Floating Point Standards Committee, I came to the realization that CORDIC algorithms could be sped up further through the use of multipliers and that it was becoming practical to put fully combinational multipliers in hardware. To see how the speed-up works for the sine function, for example, consider combining the sum-of-angles formula for the sine function,

$$\sin(A+d) = \sin(A)\cos(d) + \cos(A)\sin(d)\,, \tag{2}$$

with the Taylor series expansions for $\cos(d)$ and $\sin(d)$,

$$\cos(d) = 1 - \frac{1}{2}d^2 + \frac{1}{24}d^4 + ... \tag{3a}$$

$$\sin(d) = d\left(1 - \frac{1}{6}d^2 + \frac{1}{120}d^4 + ...\right) \tag{3b}$$

with the condition that the d^2 terms (and, therefore, also the higher terms) are small enough to be ignored. Then if we make the substitution, $B = A + d$, we get,

$$\sin(B) \approx \sin(B-d) + d\cos(B-d). \tag{4}$$

The application of this to CORDIC is as follows. To calculate $\sin(B)$ we now load a reciprocal distortion constant into the X register, as before, except that now we use a constant that accounts for the total radius distortion for just the first half of the CORDIC rotation steps. As before, we load 0 into the Y register, and the angle B into the Z register. Now we perform just the first half of the CORDIC rotation steps. At this point, the angle d in the Z register has been reduced in size to where d^2 is less than the significance of the least significant guard bit, and so d^2 can be ignored. The actual angle that has been rotated through is just $B-d$, so that means that $\sin(B-d)$ is sitting in the Y register, and $\cos(B-d)$ is sitting in the X register. So now we can calculate the sine function, without doing any further iterations, as,

$$SINE = Y_{\text{after}} + (Z_{\text{after}} * X_{\text{after}})\,. \tag{5}$$

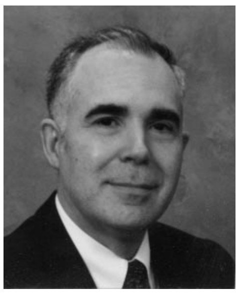
7. Conclusion

The investigation on using multipliers with CORDIC was the last work that I did on CORDIC. Around 1980, John Young, CEO of HP, put out an open call to HP employees to increase the quality of HP products by a factor of ten in ten years. In 1982, I decided to heed that call, and I set my career on the path of developing test methodologies and tools to support fast-time-to-market and high quality for the integrated circuits used in HP products, and that is what I have been doing since. Still, when I look back to my work on CORDIC, I am pleased that the idea that had perked my interest in 1965 as I was studying for a Fortran class had blossomed into such interesting results.

References

1. J.E. Meggitt, "Pseudo Division and Pseudo Multiplication Processes," *IBM Journal*, April 1962, pp. 210–226.
2. J.E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, pp. 330–334.
3. M.A. Liccardo, "*An Interconnect Processor with Emphasis on CORDIC Mode*," Masters Thesis, EE Dept., University of California at Berkeley, September 1968.
4. D.S. Cochran, "Algorithms and Accuracy in the HP-35," *Hewlett-Packard Journal*, vol 23, no. 10, 1972, pp. 10–11.

5. J.S. Walther, US patent 3766370: Elementary Floating Point CORDIC Function Processor and Shifter, filing date May 14, 1971, issue date Oct 16, 1973.
6. J.S. Walther, "A Unified Algorithm for Elementary Functions," *Conference Proceedings, Spring Joint Computer Conference*, May 1971, pp. 379–385.
7. J.S. Walther, "A Unified Algorithm for Elementary Functions," in *The Best Comuter Papers in 1971*, Orlando R. Petrocelli (Ed.), Auerbach Publishers 1972, pp. 69–81.
8. K. Hwang, *Computer Arithmetic Principles: Architecture and Design*. New York: Wiley, 1979.



John Stephen Walther John Stephen "Steve" Walther was born in 1944 in Honolulu, Hawaii. He graduated from Punahou High School in Honolulu in 1962. He received a BSEE degree from MIT in 1966 and an MSEE degree from Stanford in 1969. Steve had some interesting jobs while still in school. In 1960 at Electronics Laboratories he designed an agricultural application of "sonar in air" for the Libby Company. In 1963 at Brookhaven National Laboratory he worked on an experiment at a synchrotron to determine the gyromagnetic ratio of the Lambda-zero meson. In 1965 he worked as a vector-cardiograph operator at Boston City Hospital. From 1964 to 1966 at the MIT Instrumentation Laboratory he worked on the guidance system of the MX missile. Steve joined the Electronics Research Laboratory (ERL) of HP Labs in 1966 when HP Labs was being formed. He first worked in the medical section and helped to design an ear oximeter. He then joined the computer section, where he designed and was project leader for a math co-processor for the 2116 and 2100 computers. He invented the unified-cordic algorithm, for which he received a best paper award from the American Federation of Information Processing Societies. He worked on the HP model 35 calculator and also on the 45, 65, and 80. Steve was Associate Director of ERL from 1971 to 1973. In 1982, in response to a quality initiative at HP, Steve joined the Test Team at ERL. Since then he has contributed to the design-for-test and automatic test generation for over 100 integrated circuits, including IC's in computers, laser-jet printers, and ink-jet printers. Steve is currently Technical Contributor on the Test Team at the Design Technology Center of the Integrated Circuit Business Division at Agilent Technologies.
steve_walther@agilent.com