High-Speed Cosine Generator

James A. McIntosh
National Instruments
6504 Bridge Point Parkway
Austin, Texas 78730

Earl E. Swartzlander, Jr.
Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, Texas 78712

Abstract

An 8-bit high-speed cosine generator circuit was designed and simulated. Speed and area estimates were made for 16-, 24-, 32-, 40-, 48-, 56-, and 64-bit designs. The basis of the cosine generator design is stages of a half-angle cosine function preceded by a low order Taylor series approximation of $\cos(\theta/2^N)$. *This turns out to be a decent approximation. The big difference between this and most other methods is that a lookup ROM for the first order approximations is avoided.*

1: Introduction

It is often necessary to calculate trigonometric functions such as sine and cosine in signal processing applications. The problem is to calculate these functions quickly so that the application using them can run quickly. Previously, people have used look-up tables with some form of interpolation to calculate the sine or cosine function [1], [2]. Another possibility would be to use a Taylor series expansion [3], but this method is generally too complex and too slow to be of practical use in a high-speed design. Also, because the coefficients are different for each stage, the design becomes very difficult. For these reasons, designs based on Taylor series expansion will not be considered any further here. Another idea is to use Chebyshev polynomials [4]. This idea was the starting point of this design, but the more general cosine half-angle formula was used as the basis of the design instead.

2: Approach

2.1: Main Idea

First, we will consider the Chebyshev polynomial [2]:

$$T_N(x) = \cos(N \cos^{-1} x) \quad (1)$$

and $T_{N+1}(x) = 2xT_N^2(x) - T_{N-1}(x),\ N \geq 1$ (2)

with $T_0(x) = 1$, and $T_1(x) = x$.

If we set $T_N(x) = \cos(y)$, from equation (1) we can see that

$$x = \cos(y/N) \quad (3)$$

However, we see here that we must be able to calculate $x = \cos(y/N)$ in order to calculate $z = \cos(y)$, the value of interest, from equation (2), the Chebyshev polynomial of order N.

Several previous approaches seem to have x obtained from a lookup table and then the result is calculated from the interpolating Chebyshev polynomial.

However, solving for T_2 from equation (2) gives:

$$T_2(x) = 2x^2 - 1 \quad (4)$$

If we set $x = \cos(y/2) = \cos(\alpha)$, then we get from equations (1) and (4):

$$T_2(x) = \cos(y) = \cos(2\alpha) = 2\cos^2\alpha - 1 \quad (5)$$

This is seen to be half-angle formula for cosines. The basis for this design is to design stages based in this formula rather than the Chebyshev polynomial.

Therefore,

$$F_N(\theta) = \cos(\theta) = \cos(2^N \cos^{-1} x) \quad (6)$$

where $x = \cos(\theta/2^N)$.

Also, $F_i(\theta) = z_{i+1} = 2z_i^2 - 1$ (7)

where z_i is the input into stage i, and z_{i+1} is the output from stage i.

For sufficiently large N, the second order Taylor series approximation for $\cos(\theta/2^N)$ becomes good.

$$\therefore \quad x = 1 - \frac{(\theta/2^N)^2}{2!} = 1 - \frac{\theta^2}{2^{2N+1}} \quad (8)$$

This will be the input into stage 1.

A block diagram of the design is shown in Figure 1.

1058-6393/95 $4.00

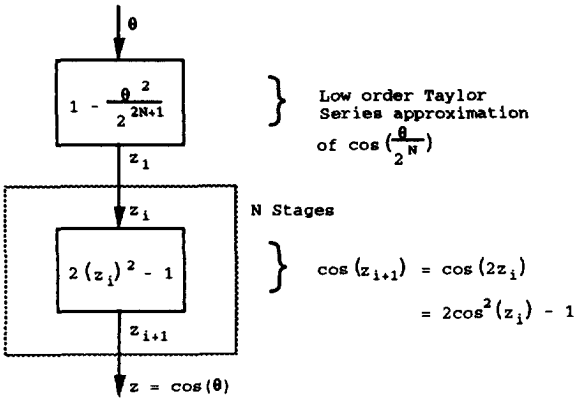

Figure 1. Block Diagram

2.2: Improvements for precision

To keep the precision throughout the various stages any additions to constants (in this case 1) are deferred until the last possible moment. To do this we consider:

$$1+\frac{z_{i+1}}{a_{i+1}} = 2(1+\frac{z_i}{a_i})^2 - 1 = 2 + 4\frac{z_i}{a_i} + 2\frac{z_i^2}{a_i^2} - 1$$

$$= 1 + \frac{4}{a_i}(z_i + \frac{z_i^2}{2a_i}) \qquad (9)$$

Thus $z_{i+1} = (z_i + \frac{z_i^2}{2a_i})$ and $a_{i+1} = \frac{a_i}{4}$.

For z_1 we use $\phi = \theta / 4$ or $\theta = \phi 2^2$ to normalize.

Thus, $x = 1 - \frac{\theta^2}{2^{2N+1}} = 1 - \frac{\phi^2}{2^{2N-3}} = 1 + \frac{z_1}{a_1}$ (10)

with $z_1 = -\phi^2$ and $a_1 = 2^{2N-3} \Rightarrow a_i = 2^{2(N-i)-1}$.

2.3: Implementation of $-\phi^2$

From [1], we see that we can use the symmetry of the multiplication matrix to simplify the calculation of any x^2. From [5], we see that

$$x^2 = x_{n-1}2^{2n-2} - 2\sum_{i=0}^{n-2} x_{n-1}x_i 2^{n-1+i} + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} x_i x_j 2^{i+j} \qquad (11)$$

Also, $$-2\sum_{i=0}^{n-2} x_{n-1}x_i 2^{n-1+i} = -2^{2n-1} + \sum_{i=0}^{n-2} \overline{x_{n-1}x_i} 2^{n+i} + 2^n \qquad (12)$$

and $$\sum_{i=0}^{n-2}\sum_{j=0}^{n-2} x_i x_j 2^{i+j} = \sum_{i=0}^{n-2} x_i 2^{2i} + \sum_{i=0}^{n-3}\sum_{j=i+1}^{n-2} x_i x_j 2^{i+j} \qquad (13)$$

Finally, we get:

$$x^2 = -2^{2n-1} + \sum_{i=0}^{n-1} x_i 2^{2i} + \sum_{i=0}^{n-2} \overline{x_{n-1}x_i} 2^{n+i} + \sum_{i=0}^{n-3}\sum_{j=i+1}^{n-2} x_i x_j 2^{i+j} + 2^n \qquad (14)$$

Normally, we would ignore the -2^{2n-1} term, but it will be important when we perform the calculation for $\frac{z_i^2}{2^{2(N-i)}} + z_i$.

By performing the two's compliment on equation (14), we get:

$$-x^2 = -2^{2n-1} + \sum_{i=0}^{n-1} \overline{x_i} 2^{2i} + \sum_{i=0}^{n-2} x_{n-1}x_i 2^{n+i} + \sum_{i=0}^{n-3}\sum_{j=i+1}^{n-2} \overline{x_i x_j} 2^{i+j} + \sum_{i=0}^{n-1} 2^i \qquad (15)$$

The following is the bit product matrix for the $-\phi^2$ design for an eight-bit cosine generator.

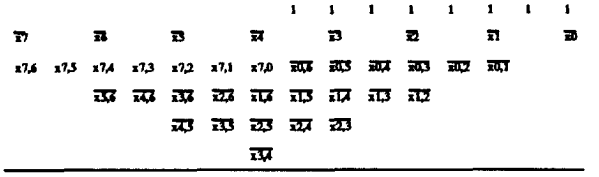

Figure 2. $-\phi^2$ bit product matrix - Eight Bits

2.4: Implementation of $\frac{z_i^2}{2^{2(N-i)}} + z_i$

Note that for any m:

$$-2^{2n-1} = -2^m + \sum_{i=2n-1}^{m-1} 2^i \qquad (16)$$

From equations (14) and (16), we get:

$$\frac{x^2}{2^{2(N-k)}} + x = -2^{2n-1} + \sum_{i=0}^{n-1} x_i 2^{2i-2(N-k)} + \sum_{i=0}^{n-2} \overline{x_{n-1}x_i} 2^{n+i-2(N-k)} + \sum_{i=0}^{n-3}\sum_{j=i+1}^{n-2} x_i x_j 2^{i+j-2(N-k)} + \sum_{i=2n-1-2(N-k)}^{2n-2} 2^i + 2^{2-2(N-k)} + \sum_{i=0}^{2n-1} 2^i \qquad (17)$$

This operation will perform the shift and the add. Any constants (for rounding or the 1.0 in the last stage) are also added as necessary. All the items in the multiplication matrix are reduced and added as one entity, similar to the merged arithmetic method discussed in [6]. A simple addition of 1 to the bit below the one wanted as the least significant bit (LSB) is used to round to the "nearest" value.

The reduction method chosen was the reduced area multiplier described by [7]. This method uses slightly less hardware and a narrower carry-lookahead adder (CLA) as

the well-known Wallace or Dadda methods. Because the number of reduction levels is the same as either of these other methods, the smaller CLA should make the reduced area multiplier slightly faster as well. Dot diagrams for each of the reduction stages of the eight-bit design are shown in the following figures.

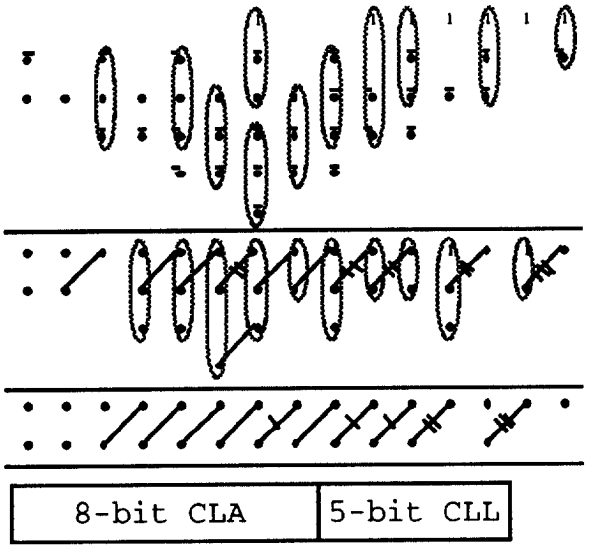

Figure 3. Stage 0: $-x^2$ - Eight Bits

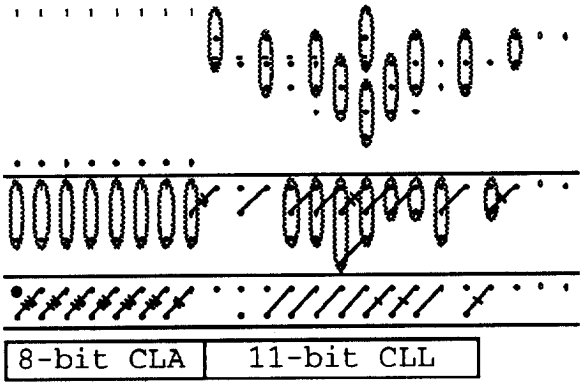

Figure 4. Stage 1: $\frac{z_1^2}{2^8} + z_1$ - Eight Bits

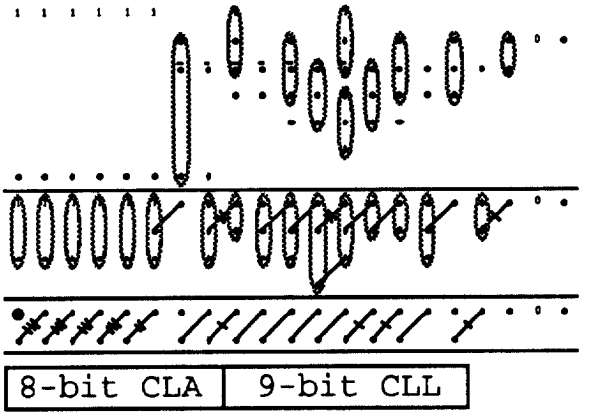

Figure 5. Stage 2: $\frac{z_2^2}{2^6} + z_2$ - Eight Bits

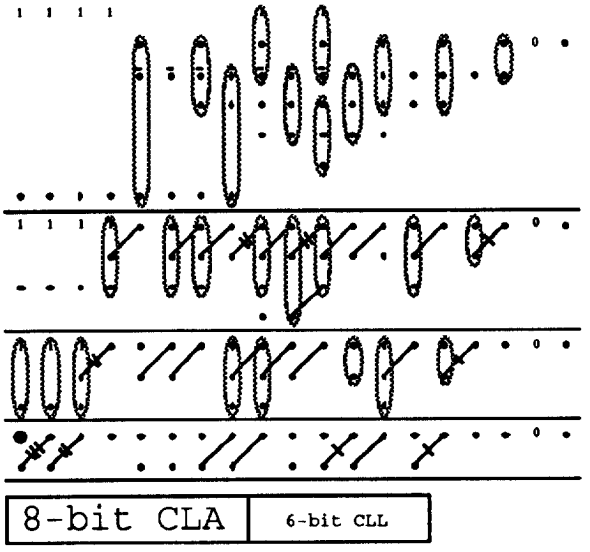

Figure 6. Stage 3: $\frac{z_3^2}{2^4} + z_3$ - Eight Bits

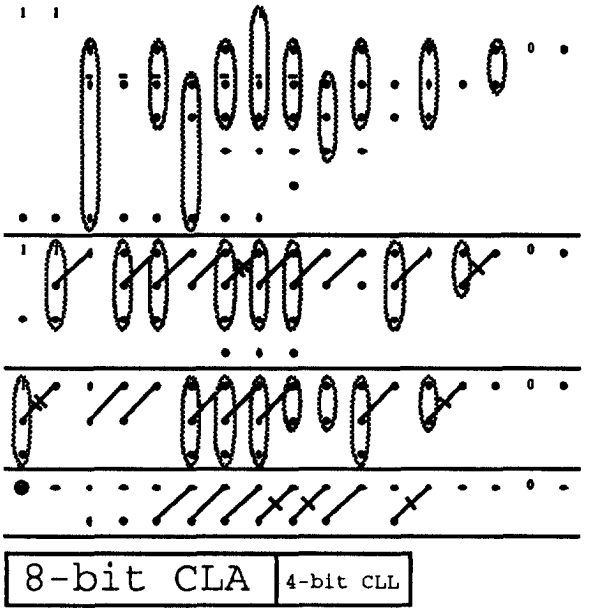

Figure 7. Stage 4: $\frac{z_4^2}{2^2} + z_4$ - Eight Bits

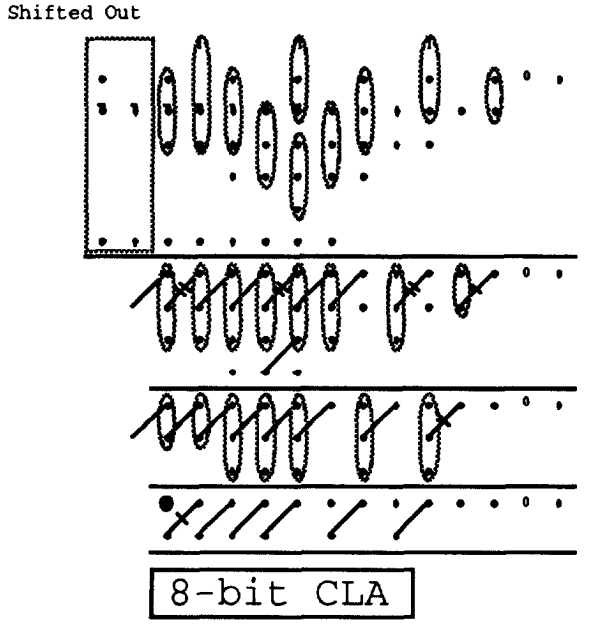

Figure 8. Stage 5: $1.0 + 2^3(z_5^2 + z_5)$ - Eight Bits

2.5: Calculating the maximum number of stages

Because of the way the bits are shifted in the final stage, and because each stage shifts the bits of the bit product matrix two more bits relative to the number to which it is being added, at some point, adding more stages accomplishes nothing. At this point, the bit product matrix is insignificant because it is shifted more places than there are significant bits. Upon examining equations (9) and (10), we can see that n/2+1 half-angle cosine stages is the maximum number of useful stages. In addition, there is the low order Taylor series approximation stage in front. This gives a maximum total of n/2+2 useful stages. This is the number of stages used in this analysis.

3: Results

Here are complexity and speed estimates for cosine generators of various sizes.

Assumptions about complexity and speed:

It was assumed that all gates (including 2- to 4-input ANDs and ORs, 2-input NANDs, and inverters) have unit area and delay, except the XOR and XNOR that have an area of 4 units and a delay of 3 units. All other components are built from these primitives.

Complexity and speed by word size:

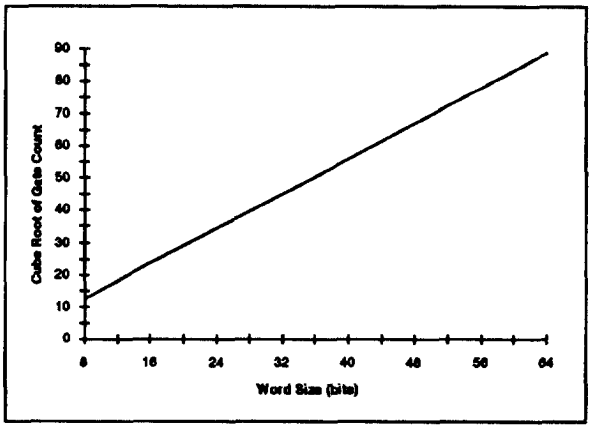

Figure 9. Gate Count vs. Word Size

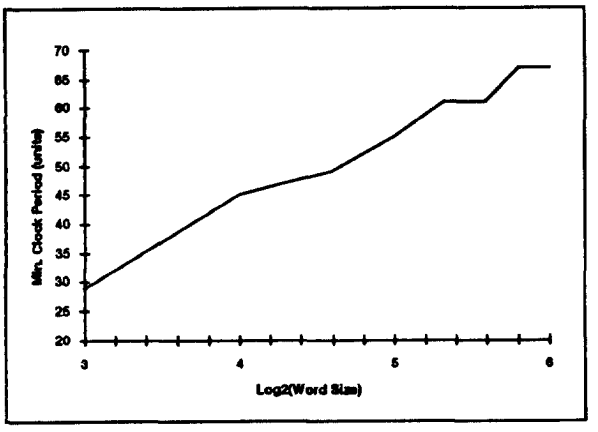

Figure 10. Throughput vs. $Log_2(n)$

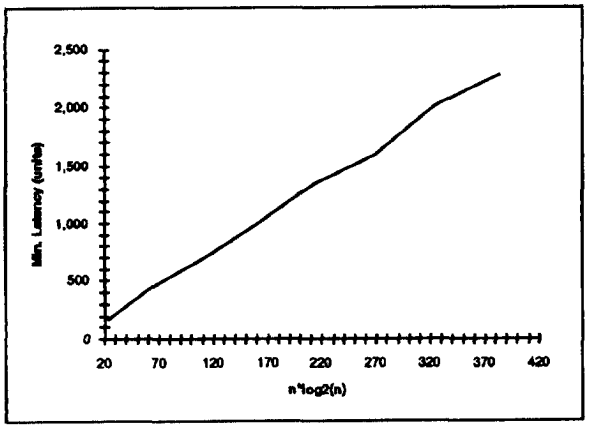

Figure 11. Latency vs. $n*Log_2(n)$

Complexity seems to be approximately on the order of n^3. Speed, however, seems to be approximately on the order of log(n) for the minimum clock period and n•log(n) for the minimum latency.

4: Conclusions

There are advantages and disadvantages to this design. One advantage is that it avoids lookup tables. If the technology is such that logic is smaller and faster than ROM, this will be better. Otherwise, the speed and complexity are comparable to existing methods for calculating the cosine function. There are other topics to study based on this analysis of the cosine generator. One would be a study of the accuracy of this design, by number of stages up to the maximum number of useful stages based on the given word size, and compared to the accuracy of other cosine generation methods. Another topic would be to investigate different methods for rounding the numbers. There have already been papers devoted to this topic. However, the method of rounding to the "nearest" and a simpler method of truncation are strong candidates based on their simplicity. The error the simpler methods introduce compared to other methods would have to be examined and a reasonable engineering trade-off would have to be made.

5: References

[1] Michael J. Schulte, *Algorithms and Hardware Designs for Parallel Elementary Function Generation*, Masters Thesis for the University of Texas at Austin, December 1992.

[2] Kai Hwang, H. C. Wang, and Z. Xu, "Evaluating Elementary Functions with Chebyshev Polynomials on Pipeline Nets," *Proceedings of the 8th Symposium on Computer Arithmetic*, pp. 121-128, 1987.

[3] P. Michael Farmwald, "High Bandwidth Evaluation of Elementary Functions," *Proceedings of the 5th Symposium on Computer Arithmetic*, pp. 139-142, 1981.

[4] Stanley A. White, "A Simple High-Performance Sine/cosine Reference Generator," *Proceedings of the 27th Asilomar Conference on Signals, Systems, and Computers*, pp. 612-616, 1993.

[5] Charles R. Baugh and Bruce A. Wooley, "A Two's Complement Parallel Array Multiplication Algorithm," *IEEE Transactions on Computers*, Vol. C-22, pp. 1045-1047, 1973.

[6] Earl E. Swartzlander, Jr., "Merged Arithmetic," *IEEE Transactions on Computers*, Vol. C-29, No. 10, pp. 946-950, October 1980.

[7] K'Andrea C. Bickerstaff, Michael J. Schulte, and Earl E. Swartzlander, Jr., "Reduced Area Multipliers," *Proceedings of the International Conference on Application-Specific Array Processors*, pp. 478-489, 1993.