Approximating the Sine Function With Combinational Logic

E. M. Schwarz
Computer Systems Laboratory
Stanford Univ.
Stanford, CA 94305-4055

M. J. Flynn
Computer Systems Laboratory
Stanford Univ.
Stanford, CA 94305-4055

Abstract

An algorithm which creates an implementation of trigonometric functions in combinational logic is presented. The algorithm provides an approximation of a trigonometric function with the latency of less than a 16 by 16 bit multiplication. Specifically, the derivation of the sine function for a fixed-point operand is shown. Traditionally, the sine function has been approximated by look-up tables, CORDIC methods, Taylor series, and Chebyshev polynomials. The algorithm presented will be shown to have faster implementations than Taylor series and Chebyshev polynomials for a similar precision, and not require any look-up tables.

1 Introduction

The proposed algorithm for trigonometric functions is based on several similar algorithms for division [15, 7, 8, 13, 14], square root [9], logarithms, and exponentials [10]. These studies describe the function to be approximated by a multiplication. The multiplication is expanded into a partial product array. The unknown operand is chosen with redundant digits such that there is no carry propagation between columns in this array. Each column forms a linear equation which is solved for a digit of the unknown operand. The unknown operand is determined by reducing the redundant digits into binary bits. The resulting formulations increase in complexity for each lesser significant digit, and therefore, only several of the most significant digits are formulated. This provides a good approximation to the function with combinational logic equations. The resulting implementation is fast and small and provides a reasonable (to be detailed) approximation.

This paper will present an algorithm for back-solving trigonometric functions, derive the formulations for the sine function, and compare them to other algorithms. The proposed algorithm will be shown to provide an approximation comparable to other algorithms with easily integrated combinational logic and with a fast latency.

2 Algorithm

The basic algorithm for trigonometric functions is:

1. Express the function or a related function as a multiplication. This involves the following sub-steps: (a) Express the binary operands as polynomials. (b) Take the derivative of the corresponding inverse trigonometric function with the operands expressed as polynomials. The resulting equation is a multiplication of polynomials.

2. Expand the multiplication into a partial product array.

3. Prohibit carry propagation between columns and solve for several of the most significant redundant digits of the unknown operand.

4. Express the resulting equations in a partial product array. Each digit of the unknown operand is weighted differently and its equation corresponds to a column in the array.

5. Reduce the array using Boolean and algebraic equivalencies.

The algorithm is specifically written for trigonometric functions but its basic steps are the same for other high-order arithmetic functions such as division, square root, logarithms, and exponentials.

The final array can be summed using Pezaris counters [11] which reduce three elements of known sign.
Let $Y$ equal the binary representation of the sine of an angle, and $\Theta$ be the binary representation of the angle. Then,

$$0 \leq Y \leq 1$$

$$Y = \sum_{i=0}^{N} y_i \cdot 2^{-i}$$

$$Y = y_0 \cdot 2^0 + y_1 \cdot 2^{-1} + \cdots + y_N \cdot 2^{-N}$$

$$Y(x) = \sum_{i=0}^{N} y_i \cdot x^i$$

$$Y(x) = y_0 \cdot x^0 + y_1 \cdot x^1 + \cdots + y_N \cdot x^N$$

And the angle is expressed by:

$$0 \leq \Theta \leq \frac{\pi}{2} \approx 1.57$$

$$\Theta = \sum_{i=0}^{N} \theta_i \cdot 2^{-i+1}$$

$$\Theta = \theta_0 \cdot 2^1 + \theta_1 \cdot 2^0 + \cdots + \theta_N \cdot 2^{-N-1}$$

$$\Theta(x) = 2 \cdot \left( \sum_{i=0}^{N} \theta_i \cdot x^i \right)$$

$$\Theta(x) = 2 \cdot (\theta_0 \cdot x^0 + \theta_1 \cdot x^1 + \cdots + \theta_N \cdot x^N)$$

The representation has been chosen with an extra bit for the angle and the sine ($\theta_0 = 0$ and $y_0 = 0$, given that $Y = 1$ is represented as $Y = 0.1111 \ldots$). The extra bit is needed since differentiation eliminates one bit from the equations. The polynomial representation is useful because it can be differentiated.

Step 1.b differentiates the arc sine function of the polynomial operands. This is expressed as follows:

$$\Theta(x) = \text{ArcSin}(Y(x))$$

$$\frac{d\Theta(x)}{dx} = \frac{1}{\sqrt{1 - Y(x)^2}} \frac{dY(x)}{dx}$$

$$Y(x)' = (1 - Y(x)^2) \cdot \Theta(x)'$$

$$Y(x)'^2 = (1 - Y(x)^2) \cdot (\Theta(x))'^2$$

Thus, step 1 of the algorithm is complete. Equation 1 represents a related function to the sine as a multiplication.

Step 2 expands all the terms of the multiplication and creates a partial product array. Each of the polynomials are expanded. The following is the expansion of the angle's polynomial form:

$$\Theta(x)' = 2 \theta_1 + 4 \theta_2 x + 6 \theta_3 x^2 + \ldots$$

3 Derivation of Sine Formulations

Prior to deriving the formulations for the sine function, the notation and the ranges of the operands are detailed. This study assumes the angle is represented in a fractional fixed point notation and has been pre-rotated to be between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$ radians. The sign of the angle in this domain is directly equal to the sign of the sine of the angle. For a sign magnitude notation the sign is calculated independent of the magnitude. Thus, the domain of the sine function can be limited to 0 to $\frac{\pi}{2}$ radians. The range of the sine function for this domain is between 0 and 1.

The derivation of sine formulation involves five steps as described in the algorithm section. The first step is to express the function or a related function as a multiplication. For trigonometric functions a related function is used which is the derivative of the corresponding inverse trigonometric function. For the sine function, the derivative of arc sine is used. Before differentiating this function, the operands are expressed as polynomials as suggested in step 1.a.
Figure 2: First 8 Columns of Partial Product Array of \((1 - Y(x)^2) \cdot (\Theta(x)^2) = (Y(x))^2\)

Thus, Equation 2 is set equal to Equation 3 and expanded into a partial product array as shown in Figure 2. Each column of the array is weighted by a different power of \(x\) rather than a different power of 2 as in a binary partial product array. The first eight columns are shown but this array can easily be expanded for more or less bits.

Step 3 back-solves each column of the array for a digit of the sine, \(Y\). The following equations are the result of back-solving 8 digits of the sine function:

\[
y_0 = 0 \\
y_1 = 2\theta_1 \\
y_2 = 2\theta_2 \\
y_3 = 2\theta_3 - (4\theta_1)/3 \\
y_4 = 2\theta_4 - 4\theta_2 \\
y_5 = 2\theta_5 + (4\theta_1)/15 - 4\theta_2 - 4\theta_3 \\
y_6 = 2\theta_6 - 8\theta_1\theta_2 - 4\theta_4 - (4\theta_1)/3 + (4\theta_1\theta_2)/3 \\
y_7 = 2\theta_7 - (8\theta_2)/3 - (8\theta_1)/3 - 3\theta_2 - 4\theta_3 \\
\]

Step 4 forms a partial product array from these formulations. The fractional terms are rounded to the number of bits in the array and represented in minimal redundant form (except if the term would overflow the array, see \(\theta_1\)). For a 12 bit estimate the array formed has a total of 113 elements and a maximum column height of 18 elements.
Step 5 reduces this array using Boolean and algebraic equivalencies (see [12, 14] for specific equivalencies used). The resulting array is shown in Figure 3. The elements with overbars are inverted. The first row indicates the digit of the sine located in the column below it, the second row indicates the weight of the column, and the third row indicates the number of elements in the column. Also, the third row last column contains the total element count. There has been a reduction from 113 to 87 total elements and from 18 to 12 elements in the maximum column.

This array can be implemented in the same way as a multiplier except Pezaris counters[11, 5, 13] are used instead of Carry-Save Adders (sometimes called CSAs or full adders), and a two's complement adder is used to subtract the outputs of the counter tree. The array for a 12-bit estimate is smaller than that of a 12 bit multiplier. The multiplier's array has 23 columns and has 12 elements in the maximum column. Thus, the sine function can be implemented with the same latency as a 12 bit multiplier and with a slightly smaller amount of area.

4 Comparison

The proposed method creates implementations in combinational logic which are similar to a small direct multiplier in area, latency, and actual implementation. Thus, the latency of the proposed method is much less than the delay of a large multiplier (i.e. 53 bits) and provides a reasonable approximation. Other methods for the sine function include look-up tables, CORDIC methods, Taylor series, and Chebyshev polynomials. Look-up tables are fast but require large amounts of area and in some technologies are difficult to implement on the same chip with combinational logic. CORDIC methods [17, 1, 16] require very little hardware and can produce a very precise result, but they require a large latency. Typically, N iterations are required to get N bits of precision. Taylor series expansion methods [5] are comparable to the proposed method since they require few iterations and no look-up tables. They can also be implemented with many look-up tables [3] but this variation will not be compared since this requires a large amount of area. Chebyshev polynomials [4, 6] also are interesting because they provide an approximation with no look-up tables. Regular Taylor series expansions and Chebyshev polynomials will be compared to the proposed method since none use look-up tables and all provide an approximation with a low latency.

Table 1 shows a numerical comparison of the proposed method for an 8-bit, 12-bit, and 16-bit estimate, to several Taylor series expansions and Chebyshev polynomials. Also, a graphical comparison of 12-bit proposed method, Taylor series, and Chebyshev polynomials is given in Figure 4. These methods were simulated exhaustively for the most significant 16 bits. The results are summarized for the average number of bits correct and minimum number of bits correct (number of bits correct is determined by taking the log base two of the absolute error each test-case). The table also shows the size of the array of the proposed method in terms of the maximum number of elements in a column and the total number of elements in the array. The maximum column height directly corresponds to the size of the columns of an equivalent size multiplier. The latency of all the methods is given in the table in terms of the delay of a large multiplication and an addition. The 12-bit proposed method is comparable to a third order Taylor series or a third order Chebyshev in both average and minimum bits correct and it requires substantially less latency. The third order Taylor and Chebyshev methods require three multiplications and one addition and
the proposed method requires the latency of less than one large multiplication. Thus, the proposed method provides an approximation comparable to a third order Taylor series or Chebyshev polynomial with less latency.

5 Conclusion

The algorithm presented in this study creates an implementation in combinational logic of an approximation of the sine function. This algorithm also can be applied to many other trigonometric functions. The latency is very small in comparison to a similar approximation by a Taylor series or a Chebyshev polynomial. The proposed method is good for an application which needs a fast approximation of a trigonometric function or as part of a higher precision algorithm. Thus, an algorithm for the sine function has been presented which is fast and easily integrated with combinational logic.

Table 1: Statistics of Proposed, Taylor Series, and Chebyshev Polynomial Method for Sine Function

<table>
<thead>
<tr>
<th>Method</th>
<th>Bits per Order</th>
<th>Bits Correct</th>
<th>Avg.</th>
<th>Max.</th>
<th>Total</th>
<th>Latency</th>
<th>May</th>
<th>Add</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prop.</td>
<td>12th</td>
<td>10.34</td>
<td>5</td>
<td>24</td>
<td></td>
<td>&lt; 1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Prop.</td>
<td>15th</td>
<td>15.82</td>
<td>33</td>
<td>20</td>
<td></td>
<td>&lt; 1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Taylor</td>
<td>1st ORD</td>
<td>5.61</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Taylor</td>
<td>2nd ORD</td>
<td>10.66</td>
<td>6.79</td>
<td>20</td>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Taylor</td>
<td>3rd ORD</td>
<td>11.76</td>
<td>7.66</td>
<td>30</td>
<td></td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cheby</td>
<td>1st ORD</td>
<td>8.81</td>
<td>7.79</td>
<td>20</td>
<td></td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Cheby</td>
<td>2nd ORD</td>
<td>14.85</td>
<td>13.94</td>
<td>40</td>
<td></td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 4: Number of Correct Bits in Proposed, Taylor, and Chebyshev Method for Sine Function

References