HIGH-SPEED COMPUTATION OF UNARY FUNCTIONS

M. Ohhashi+ and R.E. Schneider++

+ Toshiba Corporation, Kawasaki, 210 Japan
++ Lockheed Electronics Co., Plainfield, NJ 07061

ABSTRACT

This paper presents an architecture for fast evaluation of unary functions such as reciprocal, square root and reciprocal square root. The theory behind the architecture has been presented in [1]. The paper shows the results of extensive simulation that have allowed us to implement the architecture with minimum chip count and maximum accuracy. The accuracy is about 8 % error rate in the LSB of the chosen representation (IEEE 32-bit floating point format). This architecture allows the computation of unary functions in less than 200 nsec.

1.INTRODUCTION

In [1], a general algorithm is presented for calculating unary operations based on a required accuracy. This general algorithm however generates more hardware than necessary for the functions considered here, i.e. reciprocal, square root and reciprocal square root. It also does not discuss how the size of the memory output has been determined. Moreover, some of the hardware described is not commercially available, i.e. a 36 by 36 multiplier is required in [1]. The computer simulation discussed in this paper allows the examination of [1] base configuration by comparing the value calculated by the configuration with the value obtained by the computer function. By exercising the simulation, several different configurations have been tried. We will show a block diagram of the hardware to implement the series approximation that resulted in minimum error.

In the IEEE format, a single-precision floating point number is expressed in 32 bits comprised of three fields; a sign bit *s*, an 8-bit exponent *e* (biased by 127), and a normalized 24-bit fraction with the most significant fraction bit suppressed (which is always a 1 for a normalized number). The 24 bit number resides in the range of 1 to 2 - 2^{-23}. Thus the 23-bit mantissa is needed in the fractional part *f* to complete the 32 bit floating point word. A floating point number *v* would be expressed as

$$v = -1^s \times 2^{e-127} \times 1.f$$

The calculated functions have to be resolved to the same accuracy of a 24 bit mantissa including the hidden most significant bit.

The research was supported in part by Defense Advanced Research Projects Agency (DOD), monitored by the Air Force Avionics Laboratory under Contract F33615-81-K-1539, and Naval Electronic Systems Command under Contract N00039-85-C-0134, and in part by the Office of Naval Research under Contract N00014-80-C-0236, NR 048-659. This work has been done at Department of Computer Science of Carnegie-Mellon University, Pittsburgh, PA.

2.THEORY

A Taylor series expresses a function by an infinite sum of an exponential series weighted by nth derivatives of that function. Expressed mathematically,

$$f(z) = \sum_{n=0}^{\infty} y^n \times f^{(n)}(x)/n!$$

where $z = x + y$ and $f^{(n)}(x)$ is the nth derivative of f(x).

A finite sum of the Taylor series would result in the approximation of the function f(z). The number of terms in the series approximation is dictated by the required accuracy. In this work, the solution must be as close as possible to the exact value in the range of the 24 bit mantissa. Only the mantissa needs to be dealt with in the simulation. The exponent treatment results in its negation for reciprocal function or halving the exponent for the square root function, and can be handled separately. In the actual hardware implementation, some treatment of the mantissa based on the exponent will occur. This will be discussed later on in this paper.

A general conceptual diagram of the hardware is shown in Fig. 1. The argument z is separated into the two smaller quantities of x and y. The value of x is used as an address to ROMs that contain a table look-up of the derivatives used in the Taylor expansion. The value y is used both as a quantity and as an address to ROM. The contents of the ROMs are then multiplied and summed to approximate the desired function.

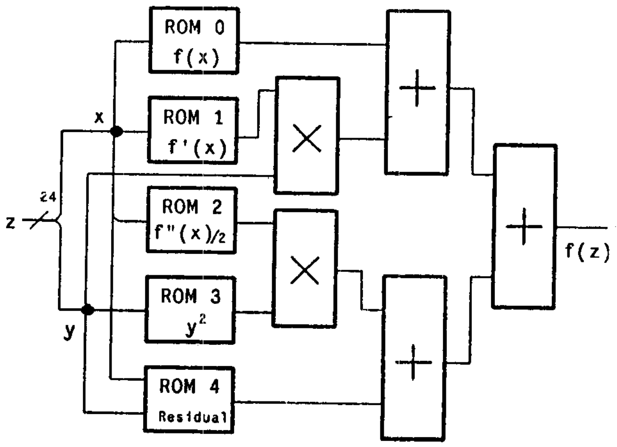

Figure 1: General conceptual diagram of Hardware

The simulation allows the modification of parameters, such as argument width. By this means, we can find the minimal widths of the Taylor terms that will yield results of acceptable accuracy to 24 bits for reciprocal, square root and reciprocal square root. The range of the argument z of the function is

$$1 \leq z \leq 2 - 2^{-23}$$

The number of terms needed for a certain accuracy is determined by the convergence of the Taylor terms. Note that $y = z - x$. The smaller y is, the faster the series will converge, since the derivatives are well behaved. Since z is a 24 bit number and x is a (b + 1) bit number, y would be a (23 - b) bit number. The resulting bounds would be:

$$1 \leq x \leq (2^{b+1} - 1) / 2^b$$

$$0 \leq y < 2^{-b}$$

The tradeoff of making the representation of y smaller results in that x be more accurate, that is more bits in its representation. Referring to the conceptual diagram of the hardware shown in Fig. 1, x will be used as an address to ROMs containing the derivatives of the functions. The more bits used for x, the larger the memory requirement. In general, it is the bit size of x that characterizes the accuracy of the result.

Generally, the exponent and the mantissa should be handled separately. This is true for the computer simulation, however this strategy has to be changed in the hardware implementation. As an example, when the square root of *v* is to be taken, the exponent needs to be halved. A problem occurs when the exponent is odd, which results in a fractional exponent. A fractional exponent cannot be represented. A solution is to pre-process the mantissa and scale it by two and compensate the scaling in the exponent, that is diminish it by 1.

3.SIMULATION RESULTS

The simulation reflects the hardware configuration of Taylor series approximation as shown in Fig.1. The truncated Taylor expansion is

$$\begin{array}{lll} f(z) \approx & f(x) + & ;\text{ROM0} + \\ & y \times f^{(1)}(x) + & ;y \times \text{ROM1} + \\ & y^2 \times f^{(2)}(x)/2 + & ;\text{ROM3} \times \text{ROM2} + \\ & y^3 \times f^{(3)}(x)/6 + y^4 \times f^{(4)}(x)/24 & ;\text{ROM4} \\ z = x + y & & \end{array}$$

The right side shows the corresponding hardware configuration. The program calculates each value of the Taylor series expansion up to the 5th Taylor term. Each term can be selected to be used or not used to see its effective contribution. This approach can directly check the hardware configuration.

The simulation program is written in the C language under dUnix (4.1 BSD) on a VAX-11/780 machine. All calculations are performed in double floating point format (64 bit) of VAX-11/780 to avoid the data accuracy loss. These quantities are then truncated to a 32-bit mantissa accuracy. Note that in Fig.1 the additions are done with 32 bits. Also in the error routine, the comparison will be between 32 bit mantissas. This double floating point format inside the VAX 11/780 is always carefully considered for bit manipulation.

The *union* structure in the C language provides a way to overlay different kinds of data in a single area of storage. With *union* the value z can be utilized either as a floating point quantity or as two 32 bit integer quantities. Using *union* allows the floating point number's mantissa to be converted into an integer and perform bit operations on it and then return the quantity back to floating point.

The input variable z has a range from 1 to $2\text{-}2^{-23}$. A better description of z is to show its hex representation:

$$800000_{16} \leq z \leq \text{FFFFFF}_{16}$$

The fixed point is located after the MSB bit in this case. For any input variable, an equivalent normalized IEEE 32 bit floating point number can be assumed. Each variable can be regarded as a number which has an effective exponent part 0 and a mantissa having the same range as z.

In the program, truncation routines are used for every input and output variables of each Taylor term calculation. The routine can truncate any floating point number to the desired width which is equivalent to the accuracy level. The routine accomplishes truncation by bit manipulation and not by mathematical operations so no rounding occurs and accuracy of the floating point value can be preserved. Truncation allows the simulation to correspond to the appropriate bit sizes of the hardware.

In the simulation, the VAX is assumed to calculate an accurate result of the considered function to a 64 bit floating point quantity. The difference between the VAX value and the Taylor series is examined in two stages. The first stage is to calculate the absolute error. This error is then quantized in steps of 2^{-32} and a count of each quantized level of error is made. This count relates the number of bit errors. The second stage is to make a bit by bit comparison of the 24 bit mantissa, and determine whether the two quantities are identical. From this error analysis, a configuration can be determined; the appropriate sizes of the table look-up ROMs can be revealed.

Exhaustive simulations, stepping z from 800000_{16} to FFFFFF_{16} in steps of 000001_{16}, have been done on the three unary functions. Note that the reciprocal function has the worst convergence rate for the Taylor series approximation. The approximated 32 bit fraction data of intermediate step has only small absolute error.

When the 32 bit mantissa data is reduced into the 24 bit mantissa of IEEE format, a bit error occurs in the LSB of the 24 bit mantissa when compared with the 24 bit correct value. For these fraction size reductions, several roundoff procedures are available[1]. Checking these procedures with the simulation indicates the truncation approach is best because the truncation approach requires no additional hardware and it was clarified that normal rounding could not reduce the LSB error rate. As shown in Table 1, 10 bit input for ROM0, ROM1 and ROM2 for each function causes acceptable LSB error rate in the range of 2.2 % to 4.6 % and 9 bit input for these ROMs has an error rate in the range of 4.0 % to 8.3 %.

Generally, if individual errors between the correct value and the approximation value are more evenly distributed over the full range, it is statistically better for the approximation. Therefore in the simulation for the LSB error, positive and negative error occurrences are counted.(when the correct value is larger than approximated value, we define an error as *positive*) For the function of reciprocal square root and square root, truncation to ROM2 output which is consistent with reducing value of $f^{(2)}(x)$ causes the slightly better symmetric error characteristics. For example, when ROM2 output is reduced from 8 bit to 7 bit accuracy in square root and reciprocal square root (that means LSB is always 0 for the latter case),*positive/negative* error rate becomes from 1.0 / 7.3 (%/%) to 2.8 / 4.9 (%/%) for reciprocal square root and from 4.1 / 0.3 (%/%) to 2.9 / 1.1 (%/%) for square root.

It is worth noting from these results that ROM 4, which is used for the 4th and 5th Taylor terms, is not effective for increasing the 24 bit data accuracy range in these functions.

4. IMPLEMENTATION

The simulation determined what memory size was required to achieve the solution to 24 bit fraction range with acceptable accuracy. This was incorporated into a configuration that would yield the complete floating point result. This configuration would compute all three unary functions, i.e., square root, reciprocal and reciprocal square root. This was achieved by using larger memory that would contain the derivatives of the corresponding functions. The memory can be considered partitioned into three areas; each area dealing with a particular function.

Some treatment must be done to each function such that the result is in IEEE floating point format and also to preserve the exponent, as already mentioned in section 2.

Consider the reciprocal function, $f(z) = 1/z$ then

$$1 \leq z < 2$$
$$.5 < f(z) \leq 1$$

The result is not in IEEE format. The result has to be scaled by 2 and compensated for in the exponent. Such treatment must be considered for each function. Furthermore, in the functions dealing with square root the result must be scaled by square root of 2 when the exponent is odd. Such post treatment is accomplished by enlarging the memory and using a control signal called parity. Parity is a control line that reflects the parity of the exponent and also the status of the v being a special case.

The special cases of zero and infinity are calculated by the exponent ROM since each have the same mantissa.

$$\sqrt{+0} = 0$$
$$\sqrt{+\infty} = \infty$$
$$\sqrt{z<0} = NaN$$

$$1/(\pm 0) = \pm\infty$$
$$1/(\pm\infty) = \pm 0$$

$$1/(\sqrt{+0}) = \infty$$
$$1/(\sqrt{+\infty}) = 0$$
$$1/(\sqrt{z<0}) = NaN$$

Nan (Not-a-Number)

Any operations on NaN and denormalized numbers will yield NaN.

A block diagram of the final architecture is shown in Fig.2. The number in the circle shows the DIP counts. In total, 44 DIPs are necessary. Each ROM contains data related to 4 functions, square root, reciprocal and reciprocal square root plus one spare location for future development. The ROMs associated with the Taylor terms have additional areas depending on the parity of the exponent.

The propagation delay through the circuit is less than 180 nsec. typical, 300 nsec maximum, utilizing 35 nsec ROMs, 50 nsec. multipliers, and 45 nsec 32 bit adders. There is also some added delay because of the set up time of the multipliers.

5. CONCLUSION

This paper presents the results of experiments on a Taylor series approach presented in [1]. The results achieved through the extensive simulation, describe a practical high speed circuit that utilizes off-the-shelf TTL compatible components that can perform the square root, reciprocal and reciprocal square root on an IEEE 32-bit floating point number. There is a maximum 8% error rate of the LSB between the circuit value and the value of the function implemented on a VAX 11/780, accurate out to 24-bit mantissa, throughout the entire range of the IEEE floating point number. The calculation of these functions can be achieved in less than 180 nsec. (typical).

Presently this circuit as described here is being implemented into a programmable systolic array cell designed at Carnegie Mellon University. A simplified version that utilizes the function reciprocal square root is being built at Lockheed Electronics Company, Inc., New Jersey. It will be used in a systolic array cell configured in a processor to execute a Givens rotation.

6. ACKNOWLEDGEMENT

We would like to extend our gratitude to the joint effort of Prof. H.T.Kung of Carnegie-Mellon University and Gerry Mersten and John SantaPietro of Lockheed Electronics Company who made the work on this project possible. In addition, we would like to thank Marco Annaratone of Carnegie-Mellon University for his guidance and technical support.

7. REFERENCES

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

2. Cavanagh, J.J.F., *Digital Computer Arithmetic*, McGraw-Hill Book Co., ComputerScience Series.

Table 1: Mantissa LSB error rate vs. ROM size for 3 functions

ROM size [INPUT BIT / OUTPUT BIT]					OCCURRENCE of MANTISSA LSB ERROR for 3 functions (in %)					
					$1/Z$		$1/\sqrt{Z}$		$\sqrt{Z}$	
ROM0	ROM1	ROM2	ROM3	ROM4	total	positive / negative	total	positive / negative	total	positive / negative
9	9	9	8	—			7.7	2.8	4.0	2.9
32	16	7	8					4.9		1.1
9	9	9	8	—	7.8	2.8	8.3	1.0	4.4	4.1
32	16	8	8			5.0		7.3		0.3
10	10	10	8	—	4.6	2.2	4.1	0.8	2.2	0.2
32	16	8	6			2.4		3.3		2.0

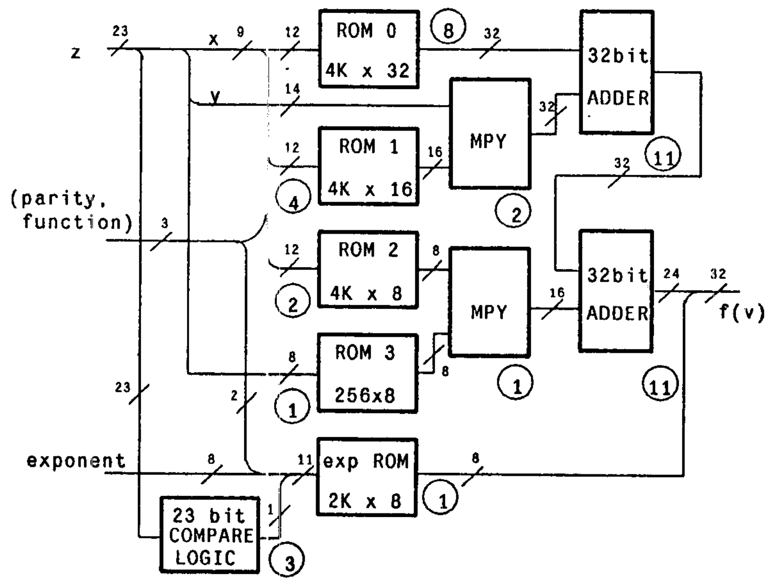

Figure 2: Blockdiagram of Unary Arithmetic Unit