A POLYNOMIAL-BASED DIVISION ALGORITHM

Robert Hägglund, Per Löwenborg, and Mark Vesterbacka
Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden
roberth@isy.liu.se, perl@isy.liu.se, and markv@isy.liu.se

ABSTRACT

A polynomial-based division algorithm and a corresponding hardware structure are proposed. The proposed algorithm is shown to be competitive to other conventional algorithms like the *Newton-Raphson* algorithm for up to about 32 bits accuracy. For example, using *Newton-Raphson* with less than 12 bits accuracy of the initial approximation, requires 33% more general multiplications than the proposed algorithm, in order to achieve 24 bits accuracy.

1. INTRODUCTION

Division is an important operation in digital signal processing. However, it is more costly in terms of computational complexity and latency compared with multiplication and addition. Besides direct table look-up techniques, which are practical for low accuracy division, most division algorithms rely on recursion with relatively complex operations involved in the loop and, hence, the latency tends to be high [1], [2].

Many recursive division algorithms have been proposed. Some of them rely on subtractive methods to calculate the quotient, producing a linear convergence of one or a few bits in each iteration. The cost in terms of area and computational complexity of such algorithms is low but due to the large number of iterations required, the latency becomes high. To achieve lower latency, division algorithms that utilize multiplication can be used. Examples of such algorithms are the well known *Newton-Raphson* algorithm and algorithms using series expansion such as *Goldschmidt's algorithm* [1]. These methods converge quadratically towards the quotient when the number of iterations is increased.

In this paper an algorithm for approximating $1/X$ is proposed. The the algorithm is executed in two step. In the first step the operand X is preprocessed to yield the intermediate result Y. In the second step a polynomial approximation using X and Y is performed yielding an approximation of $1/X$. The algorithm produces a fast linear convergence rate of, typically between, four to ten bits for each multiplication used in the polynomial interpolation step, depending on the granularity of the initial step producing Y. This convergence rate is faster than that of subtractive methods but slower than the quadratic convergence rate of, e.g., the *Newton-Raphson* method. However as is illustrated in this paper, the proposed method is competitive to the methods with quadratic convergence for accuracy up to 32 bits, depending on the accuracy of the initial approximation. For more than 32 bits accuracy, algorithms with quadratic convergence will perform better.

For lower resolution the proposed algorithm can result in a lower latency compared with, e.g., the *Newton-Raphson* algorithm. Another use of the proposed algorithm is to find an initial approximation to other algorithms with faster convergence when higher accuracy is required.

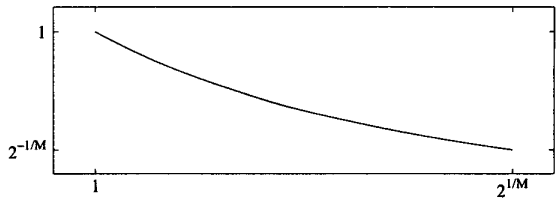

Figure 1. The function $F(Z) = 1/Z$.

Following this introduction the approximation of $1/X$ is derived in Section 2. In Section 3, implementation aspects are discussed and Section 4 shows some numerical examples.

2. DERIVATION OF THE APPROXIMATION

In order to derive the $1/X$ approximation, we start by considering the real valued function $F(Z)$, defined as

$$F(Z) = \frac{1}{Z}, \quad 1 \le Z < 2^{1/M} \tag{1}$$

where $M > 0$ is an integer constant. The function $F(Z)$ is plotted in Fig. 1. Next consider a polynomial $P(Z)$ of order N that approximates $F(Z)$ according to

$$P(Z) = \sum_{j=0}^{N} p_j Z^j \approx F(Z) \tag{2}$$

The polynomial coefficients p_j can be chosen in numerous ways. For example, $P(Z)$ could describe a series expansion of $F(Z)$ around some specific point belonging to the interval $1 \le Z < 2^{1/M}$. $P(Z)$ could also be found by minimizing some error criteria such as the Chebychev norm error, i.e., solving the optimization problem

$$\min_{p_j} \varepsilon = \max |P(Z) - F(Z)|, \quad 1 \le Z < 2^{1/M} \tag{3}$$

The approximation error that is made when substituting $F(Z)$ with $P(Z)$ will depend on the choice of polynomial coefficients in $P(Z)$, but it will also in general depend on the length of the interval $1 \le Z < 2^{1/M}$, i.e., on the constant M. For example, choosing $N = 1$ with p_0 and p_1 such that $P(1) = 1$ and $P(2^{1/M}) = 2^{-1/M}$ will define a linear approximation with a maximum approximation error $E_{max} = (1-2^{-1/(2M)})^2$ occurring at $Z = 2^{1/(2M)}$.

In the following we assume that the operand X belongs to the interval $1 \le X < 2^{W_d}$, where W_d is a constant. This can be achieved by scaling with a power-of-two factor. In a fixed-point implementation, W_d could represent the number of bits used to

describe an integer X. In Appendix 1 the following Lemma is shown:

Lemma 1: Let $Z = X \cdot Y(X)$, where $Y(X)$ is defined as

$$Y(X) = 2^{-i/M}, \quad 2^{i/M} \le X < 2^{(i+1)/M} \quad i = 0, 1, ..., W_d M - 1 \tag{4}$$

Further define a polynomial $D(X, Y)$ according to

$$D(X, Y) = Y(X)P[X \cdot Y(X)] = \sum_{j=0}^{N} p_j Y(X)^{(j+1)} X^j \tag{5}$$

If $P(Z)$ approximates $F(Z)$ such that $|P(Z) - F(Z)| < \varepsilon$ then $D(X, Y)$ approximates $1/X$ on $1 \le X < 2^{W_d}$ with $|D(X, Y) - 1/X| < \varepsilon$

□

Using *Lemma* 1 an approximation of $1/X$ with an absolute approximation error less than ε is achieved by the polynomial $D(X, Y)$, defined for $1 \le X < 2^{W_d}$. Further the coefficients of $D(X, Y)$ are identical to those of $P(Z)$ and are found from the approximation problem (2). If the function $Y(X)$ can be achieved with sufficient accuracy, the approximation accuracy of $D(X, Y)$ is as least as large as the approximation accuracy of $P(Z)$ in (3).

The question now arises how to produce the function $Y(X)$ in (4). This will be accomplished by the use of an elementary function $S_0(X)$ defined as

$$S_0(X) = 2^{-\lfloor \log_2(X) \rfloor} = 2^{-i}, \quad 2^i \le X < 2^{(i+1)}, i \ge 0 \tag{6}$$

The reason for considering $S_0(X)$ is that it can easily and efficiently be implemented in hardware. In the following it is shown how $Y(X)$ is selected using $S_0(X)$. The proof is given in Appendix 1.

Lemma 2: $Y(X)$ can be constructed as

$$Y(X) = d \sum_{k=0}^{M-1} b_k S_0(b_k X) \tag{7}$$

with $S_0(X)$ according to (6),

$$d = 2^{1/M} - 1 \tag{8}$$

and

$$b_k = 2^{k/M}, \quad k = 0, 1, 2, ..., M - 1 \tag{9}$$

□

3. PROPOSED IMPLEMENTATION

The proposed implementation of the division algorithm is shown in Fig. 2 where the constants c_k is equal to db_k. Carry-save arithmetic can be used to achieve low latency in the multiplications by the constants b_k, the sum $Y(X)$ and inside the building block $YP(XY)$.

With the choice of $S_0(X)$ as in (6), $S_0(X)$ can be efficiently implemented as shown in Fig. 3. The output from $S_0(b_k X)$ is a W_d+1-bit binary number assuming that b_k is defined as in (9). $S_0(X)$ can be realized by the Boolean expressions $z_{j+1} = \mathrm{OR}(x_j, z_j)$ with $z_0 = 0$, and $s_{W_d - j} = \mathrm{AND}(x_j, \mathrm{NOT}(z_j))$, where x_j are bits of the input word X, z_j are bits of the intermediate word Z, and

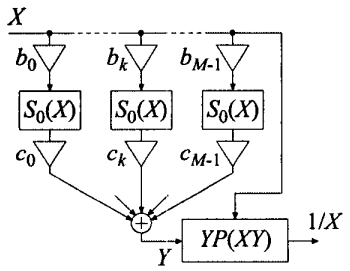

Figure 2. Implementation of the division algorithm.

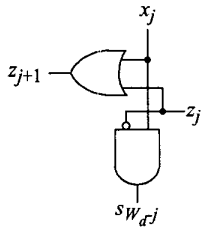

Figure 3. Implementation of $S_0(X)$

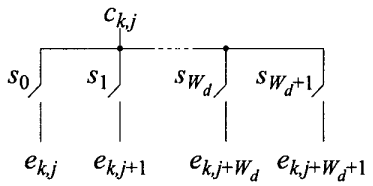

Figure 4. Implementation of bit j of the multiplexer realizing $c_k S_0(X)$.

$s_{W_d - j}$ are bits of the output word S_0. The least significant bits is x_0, z_0, and $s_{W_d + 1}$.

To reduce the amount of hardware, multiple-constant techniques [3] can be used to realize b_k. Furthermore, since $S_0(X)$ only attains powers of two, the multiplication of the constants c_k in Fig. 2 can be realized as a W_d+1:1 multiplexer for each output bit of $S_0(X)$ as shown in Fig. 4.

The implementation of $YP(XY)$ can for example be implemented as follows.

1) Use one general multiplier to multiply X and Y.

2) Compute $(XY)^i$, $i = 2, 3, ..., N$ with N–1 general multipliers.

3) Evaluate $P(XY)$. Only multiplications by constants are needed.

4) Multiply Y and $P(XY)$ using one general multiplier.

This implementation of the polynomial utilizes N+1 general multipliers in sequence to generate an Nth order polynomial approximation of $1/X$.

4. NUMERICAL EXAMPLES

In this section numerical examples are given that illustrate the properties of the proposed algorithm. Three cases are considered. In all three cases a polynomial $P(Z)$ is selected that interpolates $F(Z)$ in such a way that (3) is minimized with equiripple maximum errors ε. Further, the input signal X is assumed to be represented as a fixed point binary integer of wordlength $W_d = 12$. One

could for some applications expect the number of bits used to represent the input data to be equal to the number of bits used to represent the output data. This case is not considered here for practical reasons due to the vast number of possible input combinations using for example $W_d = 32$. It should be noted that when non-integer values, $X \geq 1$ are considered the approximation error is at most ε according to *Lemma* 1. However when X is an integer the maximum approximation error is usually much smaller than ε since the points of maximum approximation error of $D(X,Y)$ are non-integers, in general. Also, due to *Lemma* 1, the approximation error decreases with $|Y(X)|$ as X increases. The achievable accuracy using, for example $W_d = 32$ could therefore be lower (but still higher than ε) than what can be achieved with $W_d = 12$, but in practice the difference would be small if noticeable at all. The three cases studied are:

Case 1: $N = 1$, $M = 41$. $\varepsilon = 7.08\times10^{-5}$
$p_0 = 1.983236$, $p_1 = -0.983236$

Case 2: $N = 2$, $M = 40$. $\varepsilon = 2.49\times10^{-7}$
$p_0 = 2.974154$, $p_1 = -2.948458$, $p_2 = 0.974303$

Case 3: $N = 3$, $M = 41$. $\varepsilon = 8.76\times10^{-10}$
$p_0 = 3.966384$, $p_1 = -5.899493$, $p_2 = 3.899834$,
$p_3 = -0.966724$

Figure 5 shows the accuracy that is achieved for different polynomial orders N as a function of the number of parallel branches M. Both the minimum accuracy guaranteed to be achieved as well as the accuracy for an integer valued X with $W_d = 12$ is shown. It can be seen that the maximum approximation error for *Case* 1 is 9.07×10^{-6} corresponding to 16.46 bits. The corresponding values for *Case* 2 and *Case* 3 are 4.30×10^{-8} corresponding to 24.17 bits and 1.27×10^{-10} corresponding to 32.59 bits, respectively.

With $P(Z)$ of order N the number of general multiplications needed in $D(X,Y)$, besides the N+1 multiplications for the fixed coefficients $p_0, p_1, \ldots, p_N$ is N+1. The computational cost and latency for a multiplication where one input is fixed can be substantially reduced compared with that of a general multiplication. Therefore the number of general multiplications needed to produce the output is used as a relevant measure of implementation cost. Other measures could be total latency, power consumption or die area required to achieve a certain accuracy. However, since such measures are hard to utilize without thoroughly specifying the hardware on a transistor level, the number of general multiplications is used in this paper. Thus, the number of general multiplication for *Case* 1, *Case* 2, and *Case* 3 are 2, 3 and, 4, respectively.

The *Newton-Raphson* algorithm starts with an initial approximation of $1/X$ corresponding to a certain number of bits. This initial approximation is usually found from tabulated values. Then the accuracy is improved by an iterative procedure. For each iteration the execution of 2 general multiplications needs to be performed. To achieve 32 bits of accuracy with *Newton-Raphson* would require 2 iterations (4 general multiplications) with 2^7 addresses. To achieve 24 bits of accuracy one would need 1 iteration and 2^{12} addresses or 2 iterations and 2^5 addresses. 16 bits is achieved with 1 iteration and 2^8 addresses. The proposed method could therefore be advantageous in terms of, e.g. reduced latency, for 24 bit accuracy since it requires the implementation one less general multiplication (*Case* 2).

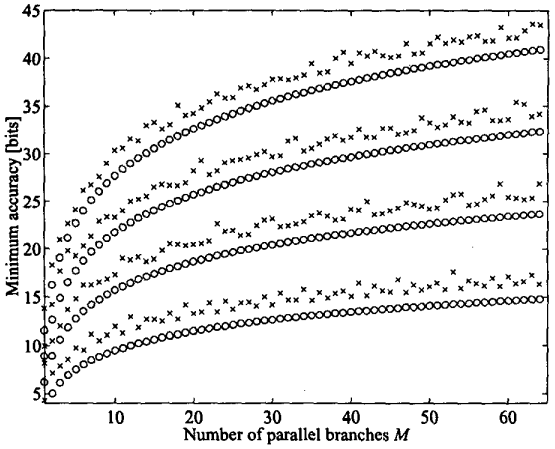

Figure 5. Minimum accuracy in bits as a function of the number of branches M for the three cases. (Lower plot $N = 1$ and upper plot $N = 4$.) Plots with 'o' show the minimum achievable accuracy, ε. Plots with 'x' shows the achieved accuracy for an integer valued X with $W_d = 12$.

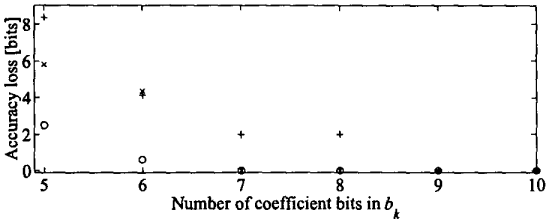

Figure 6. Number of coefficient bits required for b_k for the three different cases. *Case* 1 is plotted as 'o', *Case* 2 as 'x', and *Case* 3 as '+', respectively.

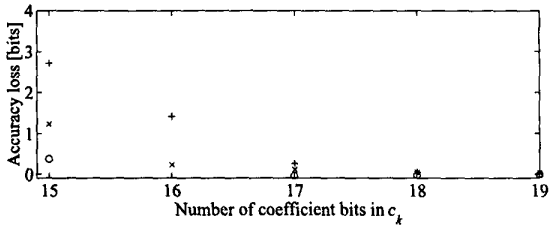

Figure 7. Number of coefficient bits required for c_k for the three different cases. *Case* 1 is plotted as 'o', *Case* 2 as 'x', and *Case* 3 as '+', respectively.

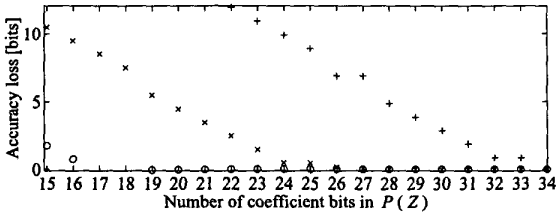

Figure 8. Number of coefficient bits required for the polynomial coefficients p_i for the three different cases. *Case* 1 is plotted as 'o', *Case* 2 as 'x', and *Case* 3 as '+', respectively.

This is provided that the computational cost and latency of computing Y, requiring 40 parallel branches, is not too high compared to that of achieving 5 initial bits for the *Newton-Raphson* algorithm. Considering 16 bits of accuracy, 8 bit initial solution is required using *Newton-Raphson* with 2 general multiplications. In *Case* 1, 16 bits is achieved with 2 general multiplications and 41 parallel branches.

Figure 6 and 7 show the loss in accuracy as a function of the number of coefficient bits for the coefficients b_k and c_k. It can be seen that between 7 to 9 and 16 to 18 bits are required for b_k and c_k, respectively, in order to preserve the accuracy. In Fig. 8 the accuracy loss for using a finite number of bits in the polynomial coefficients of $P(Z)$ is plotted. It can be seen that typically a few more bits than the required accuracy of the output is required.

5. CONCLUSION

A polynomial-based division algorithm and a corresponding hardware structure are proposed. The algorithm consists of two steps, a preceding step which preprocess the input and a second step where a polynomial approximation is performed. Any kind of interpolation or series expansion can be used. The accuracy attained is determined by the granularity of the preceding step as well as the accuracy of the final polynomial approximation. Numerical examples illustrates the properties of the proposed algorithm including effects of finite coefficient wordlength. The proposed method can be competitive to other conventional method like the *Newton-Raphson* algoritm for up to about 32 bits of accuracy. For higher accuracy the proposed algorithm can be used to create an initial solution to other conventional algorithms.

APPENDIX 1

Proof of Lemma 1: With $1 \le Z < 2^{1/M}$ and $Y > 0$ according to (4), $F(Z/Y) = Y/Z = 1/X$, for $2^{i/M} \le X < 2^{(i+1)/M}$ where $i = 0,1,\ldots W_dM-1$. That is $F(X) = 1/X$ for $1 \le X < 2^{W_d}$.

Further, since $|P[XY(X)] - 1/[XY(X)]| < \varepsilon$ the inequality $|Y(X)P[XY(X)] - 1/X| < \varepsilon |Y(X)| \le \varepsilon$ holds.

Proof of Lemma 2: The function $S_0(b_kX)$ equals

$$S_0(b_kX) = 2^{-i}, \quad 2^i/b_k \le X < 2^{i+1}/b_k, i \ge 0 \tag{10}$$

with b_k according to (9). The right hand side of (7) can be expressed as

$$RHS = d\sum_{k=0}^{M-1}\{2^{k/M}2^{-i}, \quad 2^{i-k/M} \le X < 2^{(i+1-k/M)}\}$$

$$= \begin{cases} d2^{-i}\left(\sum_{k=0}^{M-l-1} 2^{\frac{k}{M}} + \frac{1}{2}\sum_{k=M-l}^{M-1} 2^{\frac{k}{M}}\right), \\ 2^{i+\frac{l}{M}} \le X < 2^{i+\frac{l+1}{M}}, \begin{cases} i = 0, 1, \ldots, W_d - 1 \\ l = 0, 1, \ldots, M-1 \end{cases} \end{cases} \tag{11}$$

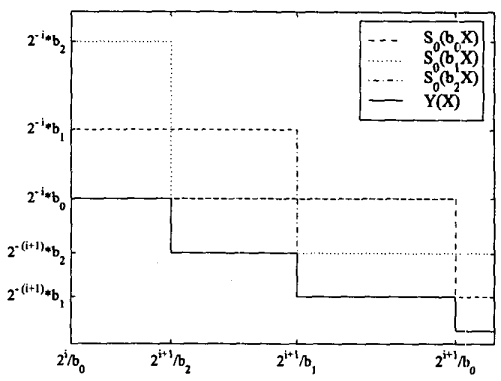

Figure 9. Creating $Y(X)$ from $S_0(X)$.

The equation (11) can be found by sketching the individual terms in the summation which are shown, for the case where we have 3 branches, in Fig. 9. Note that the normalization constant d, defined as in (8), is not included.

From Fig. 9 we can compute the function $Y(X)$ in the interval $2^i/b_0 \le X < 2^{i+1}/b_0$ according to

$$Y(X) = \begin{cases} d2^{-i}\sum_{k=0}^{2} b_k, & \frac{2^i}{b_0} \le X < \frac{2^{i+1}}{b_2} \\ d2^{-i}\left(\sum_{k=0}^{1} b_k + \frac{1}{2}b_2\right), & \frac{2^{i+1}}{b_2} \le X < \frac{2^{i+1}}{b_1} \\ d2^{-i}\left(b_0 + \frac{1}{2}\sum_{k=1}^{2} b_k\right), & \frac{2^{i+1}}{b_1} \le X < \frac{2^{i+1}}{b_0} \end{cases} \tag{12}$$

Equation (11) can be rewritten to (12) by using b_k from (9). Further simplification of (11) yields

$$RHS = \begin{cases} 2^{-j/M}, & 2^{j/M} \le X < 2^{(j+1)/M} \\ j = 0, 1, \ldots, W_dM - 1 \end{cases} \tag{13}$$

where $iM + l$ has been substituted by j. The result of (13) is equal to (4) and thereby the equality in (7) holds.

REFERENCES

[1] Oberman, S.F. and Flynn, M.J.: 'Division algorithms and implementations,' *IEEE Trans. on Computers*, Vol. 46, No. 8, Aug. 1997, pp. 833-854.

[2] Soderquist, P. and Leeser, M.: 'Division and square root choosing the right implementation,' *IEEE Micro*, Vol. 17, No. 4, July-Aug. 1997, pp. 56-66.

[3] Dempster A.G. and Macleod M.D: 'Use of minimum-adder multiplier blocks in FIR digital filters,' *IEEE Trans. on Circuits and Systems II*, 42, 1995, pp. 569-577.