Microprocessors and Microsystems 35 (2011) 23–33

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

An iterative logarithmic multiplier

Z. Babić [a], A. Avramović [a], P. Bulić [b,*]

[a] University of Banja Luka, Faculty of Electrical Engineering, Banja Luka, Bosnia and Herzegovina
[b] University of Ljubljana, Faculty of Computer and Information Science, Ljubljana, Slovenia

ARTICLE INFO

Article history:
Available online 21 July 2010

Keywords:
Computer arithmetic
Digital signal processing
Multiplier
Logarithmic number system
FPGA

ABSTRACT

Digital signal processing algorithms often rely heavily on a large number of multiplications, which is both time and power consuming. However, there are many practical solutions to simplify multiplication, like truncated and logarithmic multipliers. These methods consume less time and power but introduce errors. Nevertheless, they can be used in situations where a shorter time delay is more important than accuracy. In digital signal processing, these conditions are often met, especially in video compression and tracking, where integer arithmetic gives satisfactory results. This paper presents a simple and efficient multiplier with the possibility to achieve an arbitrary accuracy through an iterative procedure, prior to achieving the exact result. The multiplier is based on the same form of number representation as Mitchell's algorithm, but it uses different error correction circuits than those proposed by Mitchell. In such a way, the error correction can be done almost in parallel (actually this is achieved through pipelining) with the basic multiplication. The hardware solution involves adders and shifters, so it is not gate and power consuming. The error summary for operands ranging from 8 bits to 16 bits indicates a very low relative error percentage with two iterations only. For the hardware implementation assessment, the proposed multiplier is implemented on the Spartan 3 FPGA chip. For 16-bit operands, the time delay estimation indicates that a multiplier with two iterations can work with a clock cycle more than 150 MHz, and with the maximum relative error being less than 2%.

1. Introduction

Multiplication has always been a hardware-, time- and power-consuming arithmetic operation, especially for large-value operands. This bottleneck is even more emphasized in digital signal processing (DSP) applications that involve a huge number of multiplications [3,6–8,12–14,18,20,22,25]. In many real-time DSP applications, speed is the prime target and achieving this may be done at the expense of the accuracy of the arithmetic operations. Signal processing deals with signals distorted with the noise caused by non-ideal sensors, quantization processes, amplifiers, etc., as well as algorithms based on certain assumptions, so inaccurate results are inevitable. For example, a frequency leakage causes a false magnitude of the frequency bins in spectrum estimations. The signal-compression techniques incorporate quantization after a cosine or wavelet transform. When transform coefficients are quantized, instead of calculating high-precision coefficients and then truncating them, it is reasonable to spend less resources and produce less accurate results before the quantization. In many signal processing algorithms, which include correlation computations, the exact value of the correlation does not matter; only the maximum of the correlation plays a role. Additional small errors introduced with multipliers, as mentioned in the application described and others, do not affect the results significantly and they can still be acceptable in practice. Other applications that involve a significant number of multiplications are found in cryptography [4,5,10,11,19,26,27]. In applications where the speed of the calculation is more important than the accuracy, truncated or logarithm multiplications seem to be suitable methods [14,21].

1.1. Integer multiplication methods

The simplest integer multiplier computes the product of two n-bits unsigned numbers, one bit at a time. There are n multiplication steps and each step has two parts:

1. If the least-significant bit of the multiplicator is 1, then the multiplicand is added to the product, otherwise zero is added to the product.
2. The multiplicand is shifted left (saving the most significant bit) and the multiplicator is shifted right, discarding the bit that was shifted out.

A detailed implementation and description of this multiplication algorithm are given in [9]. Such an integer multiplication,

* Corresponding author. Tel.: +386 1 4768361; fax: +386 1 4264647.
E-mail address: patricio.bulic@fri.uni-lj.si (P. Bulić).

0141-9331/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.micpro.2010.07.001

where the least-significant bit of the multiplicator is examined, is known as the radix-2 multiplication.

To speed up the multiplication, we can examine k lower bits of the multiplicand in each step. Usually, the radix-4 multiplication is used, where two least-significant bits of the multiplicand are examined. A detailed explanation of the radix-4 multiplication can be found in [9].

Another way to speed up the integer multiplication is to use many adders. Such an approach typically requires a lot of space on the chip. The well-known implementation of such a multiplier is an array multiplier [9], where $n-2$ n-bits carry-save adders and one n-bits carry-propagate adder are used to implement the n-bits array multiplier.

1.2. Truncated multipliers

Truncated multipliers are extensively used in digital signal processing where the speed of the multiplication and the area- and power-consumptions are important. However, as mentioned before, there are many applications in DSP where high accuracy is not important. The basic idea of these techniques is to discard some of the less significant partial products and to introduce a compensation circuit to reduce the approximation error [13,21,23].

1.3. Logarithmic multiplication methods

Logarithmic multiplication introduces an operand conversion from integer number system into the logarithm number system (LNS). The multiplication of the two operands N_1 and N_2 is performed in three phases, calculating the operand logarithms, the addition of the operand logarithms and the calculation of the antilogarithm, which is equal to the multiple of the two original operands. The main advantage of this method is the substitution of the multiplication with addition, after the conversion of the operands into logarithms. LNS multipliers can be generally divided into two categories, one based on methods that use lookup tables and interpolations, and the other based on Mitchell's algorithm (MA) [17], although there is a lookup-table approach in some of the MA-based methods [16]. Generally, MA-based methods suppressed lookup tables due to hardware-area savings. However, this simple idea has a significant weakness: logarithm and anti-logarithm cannot be calculated exactly, so there is a need to approximate the logarithm and the antilogarithm. The binary representation of the number N can be written as:

$$N = 2^k\left(1+\sum_{i=j}^{k-1}2^{i-k}Z_i\right) = 2^k(1+x) \tag{1}$$

where k is a characteristic number or the place of the most significant bit with the value of '1', Z_i is a bit value at the ith position, x is the fraction or mantissa, and j depends on the number's precision (it is 0 for integer numbers). The logarithm with the basis 2 of N is then:

$$\begin{aligned} log_2(N) &= log_2\left(2^k\left(1+\sum_{i=j}^{k-1}2^{i-k}Z_i\right)\right) = log_2(2^k(1+x)) \\ &= k + log_2(1+x) \end{aligned} \tag{2}$$

The expression $log_2(1+x)$ is usually approximated; therefore, logarithmic-based solutions are a trade-off between the time consumption and the accuracy.

This paper presents a simple iterative solution for multiplication with the possibility to achieve an arbitrary accuracy through an iterative procedure, based on the same form of numbers representation as Mitchell's algorithm. The proposed multiplication algorithm uses different error correction formulas than MA. In such a way, the error correction can be started with a very small delay after the main computation and can run almost in parallel with the main computation. This is achieved through pipelining.

The paper is organized as follows: Section 2 presents the basic Mitchell's algorithm and its modifications, with their advantages and weaknesses. Section 3 describes the proposed solution. In Section 4 the hardware implementations of the proposed algorithm are discussed. Section 5 gives a detailed error analysis and the experimental evaluation of the proposed solution. Section 6 shows the usability of the proposed multiplier and Section 7 draws a conclusion.

2. Mitchell's algorithm based multipliers

A logarithmic number system is introduced to simplify multiplication, especially in cases when the accuracy requirements are not rigorous. In LNS two operands are multiplied by finding their logarithms, adding them, and after that looking for the antilogarithm of the sum.

One of the most significant multiplication methods in LNS is Mitchell's algorithm [17]. An approximation of the logarithm and the antilogarithm is essential, and it is derived from a binary representation of the numbers (1).

The logarithm of the product is

$$\log_2(N_1\cdot N_2) = k_1 + k_2 + \log_2(1+x_1) + \log_2(1+x_2) \tag{3}$$

The expression $\log_2(1+x)$ is approximated with x and the logarithm of the two numbers' product is expressed as the sum of their characteristic numbers and mantissas:

$$\log_2(N_1\cdot N_2) \approx k_1 + k_2 + x_1 + x_2 \tag{4}$$

The characteristic numbers k_1 and k_2 represent the places of the most significant operands' bits with the value of '1'. For 16-bit numbers, the range for characteristic numbers is from 0 to 15. The fractions x_1 and x_2 are in range $[0,1)$.

The final MA approximation for the multiplication (where $P_{true} = N_1\cdot N_2$) depends on the carry bit from the sum of the mantissas and is given by:

$$P_{MA} = (N_1\cdot N_2)_{MA} = \begin{cases} 2^{k_1+k_2}(1+x_1+x_2), & x_1+x_2 < 1 \\ 2^{k_1+k_2+1}(x_1+x_2), & x_1+x_2 \geqslant 1 \end{cases} \tag{5}$$

The final approximation for the product (5) requires the comparison of the sum of the mantissas with '1'.

The sum of the characteristic numbers determines the most significant bit of the product. The sum of the mantissas is then scaled (shifted left) by $2^{k_1+k_2}$ or by $2^{k_1+k_2+1}$, depending on the x_1+x_2. If $x_1+x_2<1$, the sum of mantissas is added to the most significant bit of product to complete the final result. Otherwise, the product is approximated only with the scaled sum of mantissas. The proposed MA-based multiplication is given in Algorithm 1.

Algorithm 1 (Mitchell's algorithm).

1. N_1, N_2: n-bits binary multiplicands, $P_{MA} = 0{:}2$ n-bits approximate product
2. Calculate k_1: leading one position of N_1
3. Calculate k_2: leading one position of N_2
4. Calculate x_1: shift N_1 to the left by $n-k_1$bits
5. Calculate x_2: shift N_2 to the left by $n-k_2$ bits
6. Calculate $k_{12} = k_1 + k_2$
7. Calculate $x_{12} = x_1 + x_2$
8. IF $x_{12} \geqslant 2^n$ (i.e. $x_1+x_2 \geqslant 1$):
   (a) Calculate $k_{12} = k_{12} + 1$
   (b) Decode k_{12} and insert x_{12} in that position of P_{approx}
   ELSE:
   (a) Decode k_{12} and insert '1' in that position of P_{approx}
   (b) Append x_{12} immediately after this one in P_{approx}
9. Approximate $N_1\cdot N_2 = P_{MA}$