Reciprocal, Division and Square Root Algorithms and Implementation

ECE 583: High Speed Computer Arithmetic

Qiao Gao

CWID: A20282211

12/3/2012

Abstract

This paper provides an overview on digital arithmetic algorithms for reciprocal, division and square root operations which have been widely used in contemporary VLSI architectures. Moreover, some implementation examples of these algorithms will also be provided and shortly described in this paper as well as some algorithm improvements that are made to speed up specific applications and make it easier to adjust to a new structure.

The algorithms that will be introduced in this article include: SRT algorithm, Newton-Raphson(NR) Iteration and Goldschmidt method for division and square root and NR algorithm for reciprocal calculation.

Meanwhile, the implementations for SRT and NR method will be provided as instances for these algorithms in actual usage. Then, the speed up method for Goldschmidt algorithm will be served as an example of algorithm optimization.

In addition to the introduction of algorithms and implementations, and purpose of this paper also focuses on the thought of improvement when we make a design depending on these algorithms.

Keywords: reciprocal, division, square root, SRT, Newton-Raphson, Goldschmidt.

1 Introduction

With the high-speed development of the electronics industry, electronic devices with high performance are widely used in many respects of our daily lives, both in usage of profession and entertainment. For example, as a higher speed of wireless network are increasingly needed because of larger amount of data is being transported to satisfy the needs for online entertainment, a faster digital signal process should be developed depending on a more efficient algorithm and hardware structure. In order to achieve this, an improvement on basic arithmetic operations such as addition, subtraction, multiplication and division is necessary since even all the algorithms are depending on these operations.

In this paper, some design methods will be involved in terms of reciprocal, division and square root operation. Though less frequently used in algorithm implementation, these operations do take more time than other basic operations. This is due to the pact that, these operations are much more complicated than the other operations which make them a critical part on some specific applications. For instance, in graphic process, division and square root reciprocal should be calculated in a very high speed, which make it essential to optimize both the algorithms and structures of them.

The algorithms for these operations can be divided into two categories: digit recurrence method which generate results digit by digit and iterative method which calculate the approximate results by iterative some internal variables step by step. In this paper, in terms of digit recurrence

method, we introduce SRT method [1] for division and square root and serve an implementation for it from Burgess and Hinds [2]. Moreover, in terms of iterative methods, we introduce Newton-Raphson algorithm and implementation from Kucukkabak and Akkas [3] as well as Goldschmidt algorithm [4] and its improvement method for division and square root made by Ercegovac and Imbert [5].

These methods provide great examples on how to deal with the algorithms and how to improve our design by changing structures.

2 Description of algorithms

1.1 SRT algorithms:

The conventional SRT division method [l] is a digit-recurrence algorithm which is developed by Sweeney, Robertson, and Tocher in 1950s depending on their independent investigations. In their research, they assume that the divider value is constrained from 1/2 to 2 which means the first digit if the fraction section should be 1 to make the design more efficiency, since we can transfer both variables to this range by just simple shift operations. More significantly, the main thought of SRT algorithm is that decreasing the number of adders in the design by just shift the remainder when it is in the range of -1/2 to 1/2. Detail explanation will be shown below.

In traditional non-store division algorithms [6], there is no possibility for the bits of quotient to be zero since we just give it the value of 1 or -1 as the result of judging the sign between the sign of the dividend and the remainder in every step. To solve this problem, in SRT method they set q_i to zero when the remainder is in the range of -1/2 to 1/2. As a result, we could just do shift operation in this situation and we get the zero possibility at the same time. Moreover, we get the correct answer so that we do not need correctness operation the final step, which make the design with smaller area consumption.

The recurrence equation for SRT division is:

$$R_{i+1} = 2(R_i - D*q_i)$$

This is same with the conventional non-store devision: we repeat this step for several times to get the final remainder and every bit of quotient we want. The steps of SRT algorithm can be summarized below:

(1) Set the dividend X as R_0

(2) Determine q_i, with the rule below:

$$q_i = \begin{cases} 1 & R_i \geq \frac{1}{2} \\ 0 & |R_i| < \frac{1}{2} \\ \bar{1} & R_i \leq -\frac{1}{2} \end{cases}$$

(3) Then follow the recurrence equation:

$R_{i+1} = 2(R_i - D*q_i)$

(4) Loop from (2) till all the n bits of q are generated.

Finally, we get the answer: $Q = \sum_{i=0}^{n-1} q_i$ and remainder R_n.

This is the SRT algorithm for division operations. We can just adjust the same method on square root by changing the recurrence equation to [2]:

$$R_{i+1} = 2(R_i - D*q_i) - q_i^2 * 2^{-i-1}$$

As described by McCann and Pippenger [7], in this method, at least one bit of precision is generated per iteration of the algorithm, regardless of the values of D or R_0, which means the more loops we operate, more precise the answer will be. However, the area and timing consumption may greatly increase at the time. That's why we have to hold the remainder, which may make it possible to check the error and make correctness for external circuits.

1.2 Newton-Raphson algorithms:

Another popular algorithm is Newton-Raphson method, which is an approximate method depending on Taylor series. For each iterative, the result gets closer the correct point by going one level deeper in Taylor series. Actually, we can run more than one level in one iterative step which will result in much more complex circuit and more area consumption. However, we do not need the result to be so accurate for most conditions so we just do one level iterative which is known as Newton-Rapson iterative algorithm [3].

The principle of Newton-Raphson iterative algorithm can be summarized as below:

$$X[j+1]=x[j]-f(x[j])/f'(x[j])$$

This equation provides the basic iterative method of Newton-Raphson, with no limitation of operation categories. In this paper, however, we just provide the algorithm of reciprocal depending on NR method.

More specifically, for reciprocal operation, f(R)=1/R-d. So the equation above can be changed to:

$$R[j+1]=R[j](2-R[j]d)$$

This is the iterative formula for reciprocal operation. As we can see from this equation, two multiplications and one subtraction should be completed during one iterative and with $R[i]^2$ inside the representation, the accuracy of bit doubles in every iterative.

Then how many iterative operations should we take? This depends on the firth approximate value we choose and the final accuracy we want. However, there are so many ways for us to choose the first R according to d that we can even choose any value we like. This value must be easy to get and as close as possible to the accurate result. One normal solution is adding Table Look-up values at this point [3]. In this method, we modify d and use it to find an approximate initial value in a table. In this process, we can finish the initial part without additional calculations.

1.3 Goldschmidt algorithms:

Goldschmidt algorithm is a division method which was discovered by Goldschmidt in 1960s [4]. This method is also known as multiplicative normalization method and being widely used among AMD CPUs. The algorithm can be expended to calculate square root and reciprocal. We can summarize this algorithm below:

$$Q = \frac{X}{D} = \frac{X \cdot K_1 K_2 K_3 \dots K_n}{D \cdot K_1 K_2 K_3 \dots K_n} = \frac{q_n}{r_n}$$

If we can control K to make r_n get closer to 1 for every multiplication, then q_n will become increasingly closer to Q. This is the basic though of Goldschmidt method. As a result, the key point of Goldschmidt algorithm focuses on how to choose the first and how to generate K step by

step. Obviously, the best K_1 is 1/D, so we should find approximate value for this. The same with the initial part of Newton-Raphson algorithm, we can depend this step on Look-up Table (LUT) with modified D. We can summarize the steps of Goldschmidt method as below [5]:

Let $D=1:d_1d_2...d_{n-1}$ and define $\widehat{D}=1:d_1d_2...d_p$, where $p<<n$. Typical values here are $n=53$ and $p\approx10$. From an optimal reciprocal table with p-bits-in and p 2-bits-out that uses $\widehat{D}$ as entry, obtain a p+2-bit approximation K_1 to 1/D. Define $q_0=X$, $r_0=D$. We then compute

$$r_i=r_{i-1}K_i;$$
$$q_i=q_{i-1}K_i;$$
$$K_{i+1}=2-r_i$$

The method of choosing K_{i+1} is depending on 2's complement of r_i actually. The accuracy of this method is doubled in every step. To prove this, we define the error $e=1-K_1D$, then $r_1=1-e$, $K_2=1+e$. As a result, $r_2=1-e^2$. If we keep doing this step, we can get:

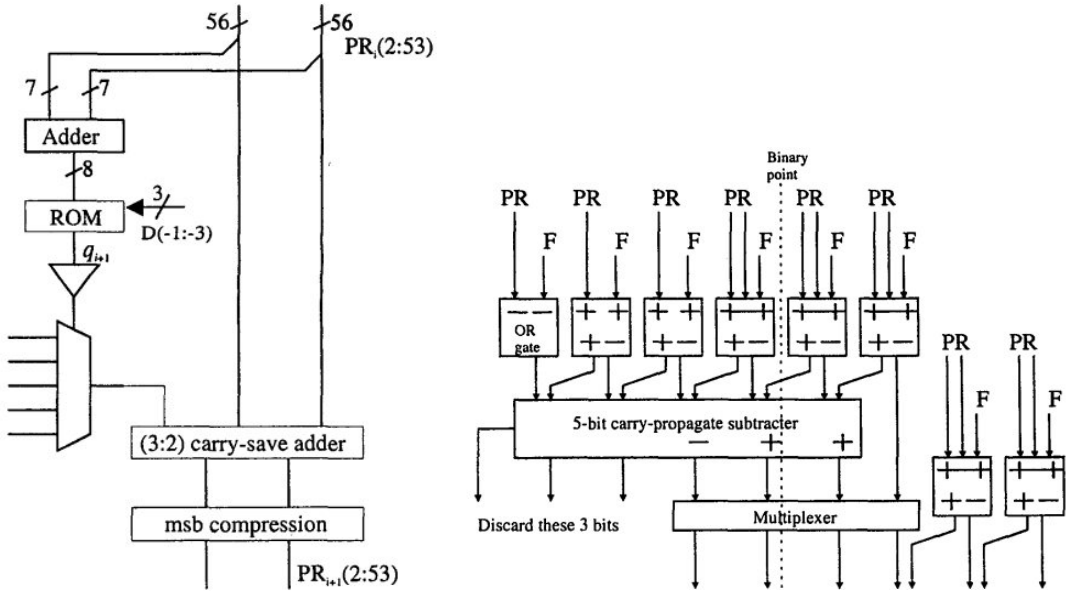
$$Q=\frac{X}{D}=\frac{q_n}{1-e^{2^{n-1}}}$$

So we make r_i very close to 1 by these steps in which e is smaller than 1 and determined by LUT.

3 Implementations

After introduce some algorithms that are frequently used in digital circuit design, some implementation examples will be provided in this part.

In terms of SRT division algorithm, Burgess and Hinds [2] provided their implementation which is shown in Figure 1:



(a) (b)

Figure 1: (a) iterative structure, (b) msb Compression of partial remainder

This is a Radix-4 implementation that generate more than one bit at each iterative and its recurrence equation can be concluded as:

$$R_{i+1} = rR_i - 2D*q_i$$

In which r is the radix number [1]. This implementation can provide more accuracy than conventional method with faster speed depending on LUT which is the ROM part in Figure 1 (a).

Moreover, Kucukkabak and Akkas [3] provided their solution on reciprocal implementation depends on Newton-Raphson algorithm. The structure is shown below (Figure 2):

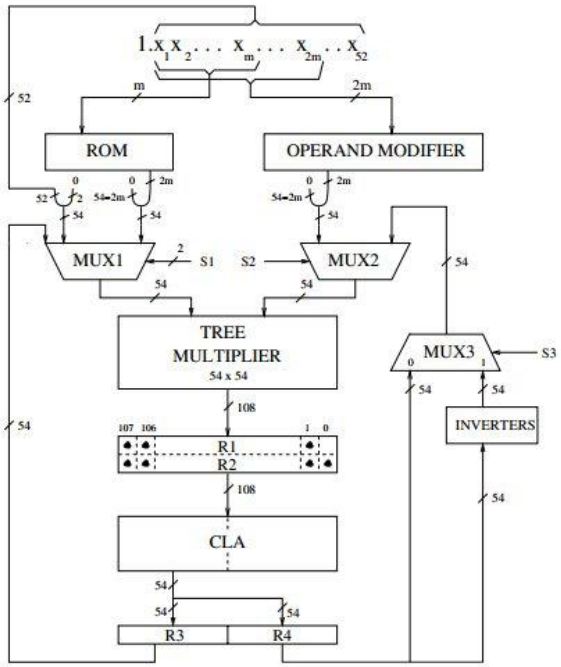


Figure 2: Reciprocal Unit Implementation

In this structure, we can see clearly that at the initial part, a ROM LUT and an operand modifier are used, as we have described in part two. The core part of this structure is a tree multiplier that is used for all the multiplications in this algorithm and the operands of it are controlled by several MUXs. This is a very efficient structure with smaller area consumption and faster speed than traditional design.

4 Result and discussion

For all these implementations, the designers not only just modified their design structures to the algorithms that are already exist, but also change some details of the algorithms which make it easier to be implemented in hardware domain. As what we have described in the implementations [2][3], we can put part of our calculations in LUTs to make the process faster and expend our algorithms to more the radix-2 to achieve more accuracy. These are all important methods to improve our designs depending on specific algorithm.

5 Conclusions

This paper covered three digital arithmetic algorithms: the SRT recurrence method for division and square root, the Newton-Raphson iterative method for reciprocal operation and Goldschmidt division method, which are widely used in today's digital circuit design such as VLSI. Moreover, two implementations of division and reciprocal operation are provided as the examples of modern digital circuit design.

6 Future recommendations

Though we already have very good designs for basic arithmetic operations, faster and lower cost

designs are still needed due to the high-speed development of electronic industry. As a result, we still have a long way to go to improve our designs. We can modify the algorithms with smaller LUTs and decrease the complexity for designs with lower accuracy demand. But more significantly, obviously, some brand new algorithms may greatly improve the implementation of arithmetic operations to a much higher level.

7 References

[1] M.D. Ercegovac and T. Lang, "Division and square root: digit recurrence algorithms and implementations", Boston: Kluwer Academic Publishers, 1994.

[2] N. Burgess and C. Hinds, "Design Issues in Radix-4 SRT Square Root and Divide Unit,"Proc. 35th Asilomar Conf. Signals, Systems and Computers (Asilomar '01), pp. 1646-1650, 2001.

[3] U. Kucukkabak and A. Akkas, "Design and implementation of reciprocal unit using table look-up and Newton-Raphson iteration", in Proc. of Euromicro Symposium on Digital System Design, 31 Aug.- 3 Sept. 2004, pp. 249 - 253.

[4] R.E. Goldschmidt, "Applications of Division by Convergence", MSc dissertation, Massachusetts Inst. of Technology, June 1964.

[5] M.D. Ercegovacet et al., "Improving Goldschmidt Division, Square Root, and Square Root Reciprocal," IEEE Trans. Computers, vol. 49, no. 7, pp. 759-763, July 2000.

[6] M.D. Ercegovac and T. Lang, Digital Arithmetic. Morgan Kaufmann, 2004.

[7] Mccann, M. and Pippenger, N., "SRT division algorithms as dynamical systems", Computer Arithmetic, 2003. Proceedings. 16th IEEE Symposium on, pp. 46-53, 2003