Hardware Reciprocation using Degree-3 Polynomials but Only 1 Complete Multiplication

Arnaud Tisserand

Hardware Reciprocation using Degree-3 Polynomials but Only 1 Complete Multiplication

Arnaud Tisserand
ARITH Group, LIRMM, CNRS–Univ. Montpellier 2
161 rue Ada. F-34392 Montpellier, FRANCE
arnaud.tisserand@lirmm.fr

***Abstract*— This paper presents a dedicated operator for the reciprocal evaluation in hardware. It uses a degree-3 polynomial approximation that only requires one complete multiplication and a small number of additions or subtractions. The polynomial approximation is determined using a specific method for the choice of the coefficients and the splitting of the input interval. This leads to small and fast FPGA implementations.**

I. Introduction

Basic operations such as addition/subtraction or multiplication have always been implemented as high-performance dedicated operators or units in digital circuits [1]. Some recent applications, e.g. frequency demodulation, require fast evaluation of more complex operations or functions such as reciprocal $(1/x)$. Function approximation is often performed using polynomials in software as well as in hardware [2]. For instance, elementary functions (sine, cosine, exponential, logarithm, arc-tangent...) are often evaluated using polynomials [2]. Polynomial approximation to the reciprocal is often proposed for small accuracy [3]. Other algebraic functions, such as square root or square root reciprocal can be efficiently approximated using polynomials.

In hardware implementations, the size of the multipliers is a major concern. Several solutions have been investigated to limit their size [4]. For polynomial approximations, this constraint limits the degree of the polynomial. This paper presents an operator dedicated to the reciprocal evaluation using low-degree polynomial approximations with hardware reduced cost and small delay. A specific method for the selection of the polynomial coefficients allows to significantly reduce the quantity of computations for the approximation without a loss in accuracy.

This paper is organized as follows. Notations and background on function approximation are presented in Section II. Standard polynomial evaluation schemes are presented in Section III. Section IV presents the selection method for the polynomial coefficients. The implementation of the operator is presented in Section V. Section VI concludes this paper.

II. Notations and Background

The function to be evaluated is the reciprocal function $f(x) = 1/x$ for the argument x in the interval $[1, 2]$. This input interval corresponds to the range of a floating-point mantissa. Several range reduction schemes can be used to reduce to this interval. A complete summary of range reductions schemes may be found in [2]. The argument, the result and some intermediate values are represented using the standard unsigned binary fixed-point number system.

The degree-d *minimax* polynomial approximation to a function f on $[a, b]$ is the polynomial P^* that satisfies $||f - P^*||_\infty = \min_{P \in \mathcal{P}_d} ||f - P||_\infty$, where $\mathcal{P}_d$ is the set of polynomials with real coefficients and degree at most d, and the approximation error is $||f - P||_\infty = \max_{a \le x \le b} |f(x) - P(x)|$. The `numapprox` package in the Maple computer algebra system [5] provides the `minimax` function which numerically estimates polynomial approximations and the `infnorm` function which numerically estimates $||f - P^*||_\infty$. The polynomial used for the approximation here will be the degree-3 polynomial $P(x) = \sum_{i=0}^{3} p_i x^i$.

In the following, errors are expressed directly or as equivalent accuracy. The accuracy is the number of correct or significant bits. The relation between the error ϵ and the accuracy μ is $\mu = -\log_2 |\epsilon|$. For instance, the error $\epsilon = 0.0000107$ corresponds to an accuracy of $\mu = 16.5$ bits.

The minimax polynomial for $1/x$ with $x \in [1, 2]$ is:

$$2.871320 - 3.029870x + 1.392785x^2 - 0.235498x^3,$$

and it leads to a theoretical accuracy of 9.62 bits. The ranges of the coefficients require a different number of integer bits in the fixed-point representation. In order to avoid this scaling problem, we now consider $f(x) = 1/(1 + x)$ with $x \in [0, 1]$. Then the minimax polynomial is:

$$0.998737 + -0.950793x + 0.686291x^2 - 0.235498x^3.$$

The theoretical approximation error is the same since the change of variable $x = 1 + x$ does not modify the minimax polynomial quality. In practice one evaluate $1/(1 + x)$ for $x \in [0, 1]$, this leads to more efficient operators.

III. Polynomial Evaluation Schemes

The evaluation of $P(x)$ can be done using different evaluation schemes [2]. The direct, Horner and Estrin schemes are commonly used and they are summarized in Table I.

The direct scheme is not used due to its high operation cost and smaller accuracy (see [6] for accuracy analysis). The Horner scheme has the smallest operation cost but it is a sequential structure while the Estrin scheme allows some internal parallelism. The Horner scheme is implemented using basic blocks called *fused multiply and adds* (FMAs).

1-4244-1176-9/07/$25.00 ©2007 IEEE.

scheme	computations	# ±	# ×
direct	$p_0 + p_1x + p_2x^2 + p_3x^3$	3	5
Horner	$p_0 + (p_1 + (p_2 + p_3x)x)x$	3	3
Estrin	$p_0 + p_1x + (p_2 + p_3x)x^2$	3	4

TABLE I
STANDARD EVALUATION SCHEMES FOR DEGREE-3 POLYNOMIALS

A FMA computes $uv + w$ with more or less the same cost (delay and circuit area) than a single multiplication uv. The Horner scheme leads to a very efficient evaluation method for software implementations on processors providing FMA units but without internal parallelism (such as in floating-point units of several modern processors). For software implementation on a modern super-scalar or VLIW processor, the Estrin scheme leads to fast solutions due to parallelism. But for hardware implementation its higher operator cost (large area of multipliers), reduces its interest. Table II presents a scheduling example for the evaluation of a degree-3 polynomial using Horner and Estrin schemes assuming additions and multiplications require one cycle and two units are available for the Estrin scheme. This table clearly shows that Estrin scheme leads to faster polynomial evaluation even with a higher operation count than the Horner scheme.

This paper shows how to choose coefficients p_1 and p_3 such that multiplications p_1x and p_3x reduce to a small number of additions or subtractions in the case of the reciprocal evaluated using the Estrin scheme.

IV. COEFFICIENTS SELECTION METHOD

Table III presents the accuracy (number of significant bits) of minimax approximation polynomials for various degrees and the reciprocal function $1/(1 + x)$ on $[0, 1]$.

degree	1	2	3	4	5
accuracy [bits]	4.54	7.08	9.62	12.17	14.71

TABLE III
ACCURACY OF MINIMAX POLYNOMIALS FOR $1/(1 + x)$ ON $[0, 1]$

The best degree-3 polynomial approximation to $1/(1 + x)$ on $[0, 1]$ only leads to 9.62 bits of accuracy. In order to increase the accuracy without increasing the degree of the polynomial, we split the input interval into two sub-intervals and use one polynomial on each sub-interval. In order to simplify the sub-interval selection the decomposition is often $[0, 1] \longrightarrow ([0, 1/2], [1/2, 1])$. This decomposition leads to the accuracies and polynomials reported below:

- Sub-interval $[0, 1/2]$: 12.50 bits of accuracy with $0.999828178 - 0.987732997x + 0.859842408x^2 - 0.435419491x^3$
- Sub-interval $[1/2, 1]$: 14.98 bits of accuracy with $0.969776265 - 0.797738933x + 0.437900384x^2 - 0.109968557x^3$

The accuracy on the 2 sub-intervals is not balanced and most of the coefficients p_i are not very close to powers of 2.

In order to find values p_1 and p_3 closer to powers of two and to balance the accuracy on both sub-intervals we use another decomposition of the input interval $[0, 1]$. We use the sub-intervals $I_L = [0, m]$ and $I_H = [m, 1]$. This requires a full-width comparison for the sub-interval selection (implemented using a subtracter).

The search for the value m is performed using an manual exploration starting from $m = 1/2$. The value $m = 0.404$ leads to the accuracies and the polynomials reported below:

- Sub-interval I_L: 13.48 bits of accuracy with $P_L(x) = 0.999912796 - 0.992439460x + (0.895560967 - 0.500060549x)x^2$
- Sub-interval I_H: 13.74 bits of accuracy with $P_H(x) = 0.975990948 - 0.823384105x + (0.472178883 - 0.124858604x)x^2$

The search for the value m is performed by hand using several tests based on the `minimax` function in Maple. This value $m = 0.404$ leads to good results in practice. In the future, we plan to try to formalize this problem in order to provide an algorithm and an optimal value.

Using the decomposition $([0, 0.404], [0.404, 1])$, the coefficients p_3 are very close to powers of two (namely $-1/2$ for I_L and $-1/8$ for I_H). Then the multiplication p_3x is replaced by the term $-2^{-1}x$ for I_L and $-2^{-3}x$ for I_H. So the evaluation cost for sub-polynomial $S(x) = p_2 + p_3x$ reduces to only one subtraction. According to the Estrin evaluation scheme presented in Section III, $S(x)$ must be multiplied by x^2. For this operation we will use a specific squaring method recalled in Section V-A.

Now we have to replace the products $p_1 \times x$ on each sub-interval by a small number of additions/subtractions. We use quantification of the coefficient p_1. The quantification results are described in Table IV. Several numbers of non-zero bits are tested and the corresponding accuracies (total approximation error with quantified p_1) are reported. This shows that the product $p_1 \times x$ can be replaced by up to 5 additions/subtractions. The additions/subtractions involved in this computations are not on the critical path.

The numerical quality of the approximation to a function using a polynomial deals with two components: the *approximation error* and the *round-off error*. The approximation error measures the distance between the approximated mathematical function and the function used for the approximation, here the polynomial p. Here the approximation error is provided by the `minimax` Maple function. The approximation error is also called the *method error*. One can deal with the approximation error by using more accurate approximations. This mainly corresponds to use higher degree minimax polynomials. In practice bounding the approximation error is not really difficult using tools such as the `numapprox` package in Maple.

The *round-off error* or *rounding error* due to the discrete nature of the intermediate and final values adds up to the approximation error. It is the difference between the calculated approximation of a number and its exact mathematical value.

Cycle	Horner			Estrin			
	unit	# $\pm$	# $\times$	unit1	unit2	# $\pm$	# $\times$
1	$p_3 \otimes x$	0	1	$x \otimes x$	$p_3 \otimes x$	0	2
2	$p_2 \oplus p_3 x$	1	0	$p_1 \otimes x$	$p_2 \oplus p_3 x$	1	1
3	$(p_2 + p_3 x) \otimes x$	0	1	$p_0 \oplus p_1 x$	$(p_2 + p_3 x) \otimes x^2$	1	1
4	$p_1 \oplus (p_2 + p_3 x)x$	1	0	$p_0 + p_1 x \oplus (p_2 + p_3 x)x^2$	—	1	0
5	$\big(p_1 + (p_2 + p_3 x)x\big) \otimes x$	0	1	—	—	0	0
6	$p_0 \oplus \big(p_1 + (p_2 + p_3 x)x\big)x$	1	0	—	—	0	0
	total	3	3		total	3	4

TABLE II

SCHEDULING OF HORNER AND ESTRIN SCHEMES FOR DEGREE-3 POLYNOMIALS ($\oplus$ AND $\otimes$ OPERATIONS ARE COMPUTED AT CURRENT CYCLE)

interval	p_1	accuracy [bits]
I_L	1	8.35
	$1 - 2^{-7}$	11.91
	$1 - 2^{-7} + 2^{-12}$	**12.47**
	$1 - 2^{-7} + 2^{-12} + 2^{-17}$	12.48
I_H	1	2.49
	$1 - 2^{-3}$	4.27
	$1 - 2^{-3} - 2^{-4}$	6.55
	$1 - 2^{-3} - 2^{-4} + 2^{-7}$	8.44
	$1 - 2^{-3} - 2^{-4} + 2^{-7} + 2^{-9}$	9.97
	$1 - 2^{-3} - 2^{-4} + 2^{-7} + 2^{-9} + 2^{-10}$	**12.95**

TABLE IV

ACCURACY FOR SEVERAL QUANTIFICATIONS OF COEFFICIENT p_1

This error is small for one single operation, i.e. a fraction of the weight of the least significant bit (LSB). But during a sequence of operations, these small errors may accumulate themselves and significantly degrade the accuracy of the final result. Bounding the round-off error is a very complex task. In digital-signal processing some methods have been proposed to model round-off errors using noise (see [7]).

The GAPPA software, presented in [8], allows to evaluate and to produce a proof of mathematical properties on numerical codes. The main useful characteristic of GAPPA used in this word is its capability to tightly bound round-off errors and prove these bounds are below some threshold. GAPPA can generate a file for the Coq proof assistant (see [9]).

We use GAPPA to get the total accuracy (including approximation and round-off errors) of our operator for each subinterval (I_L and I_H). These accuracy results for our operator are reported in Table V.

interval	$I_L = [0, 0.404]$	$I_H = [0.404, 1]$
accuracy [bits]	11.35	11.58

TABLE V

ACCURACY RESULTS FOR OUR OPERATOR

V. OPERATOR IMPLEMENTATION

The operator is decomposed into several parts. Its overall architecture is presented on Figure 1. The critical path is indicated using a dotted line on this figure. As in FPGA additions and subtractions are performed using dedicated logic using the fast carry lines, the critical path is the delay of the squarer plus the delay of the multiplier plus finally the delay of the last adder.

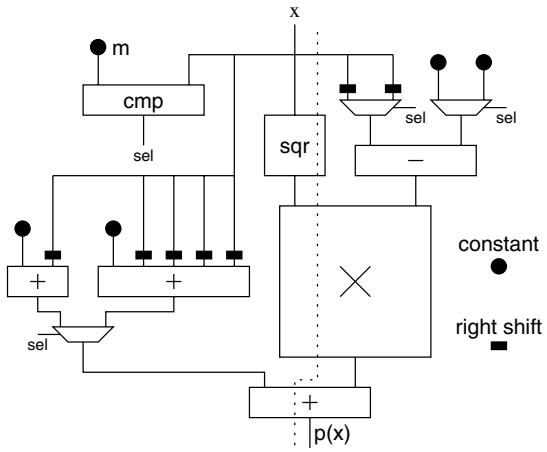

Fig. 1. Reciprocal operator decomposition

A. Square

The x^2 operation is implemented using an optimized method that can be found in computer arithmetic books such as [1]. The partial products $x_i x_j$ for all i and j in $\{0, 1, \ldots, n-1\}$ are reduced using several identities:

- the term $x_i x_i$ reduces to x_i;
- the sum of the two terms $x_i x_j + x_j x_i$ in a given column can be replaced by $2x_i x_j$ in the next column (of higher weight);
- another simplification is possible using $x_i x_{i-1} + x_i = 2x_i x_{i-1} + x_i \bar{x}_{i-1}$, i.e. two partial products in a column

are replaced a by one partial product in the same column and another one in the next column (of higher weight).

Based on these simplifications, a dedicated square operator is significantly faster and smaller than a standard multiplier.

B. Sub-Polynomial $S(x) = p_2 + p_3 x$

Due to the very simple value of $p_3 = -2^{-1}$ or $p_3 = -2^{-3}$, the sub-polynomial $S(x)$ only requires one subtraction.

C. Complete Multiplication $S(x) \times x^2$

The only complete or full-with multiplication is the one for the product of $S(x)$ by x^2. A large part of the operator circuit is devoted to this operation.

D. Final Multi-operand Addition

Due to the quantification of the coefficient p_1 using several signed bits, several terms must be added in a final addition. For each multi-operand addition/subtraction the 1 is not considered as an input because it only impacts the very most significant bits.

E. FPGA Implementation

The operator presented in this work has been implemented on a Spartan3 FPGA (xc3s250e-ft256-4) using ISE CAD tools from Xilinx for synthesis and place/route (average effort for each step). The implementation results are reported in Table VI.

synthesis effort	area	speed
area [slices]	96	103
delay [ns]	17.7	15.2

TABLE VI
IMPLEMENTATION RESULTS ON SPARTAN3 FPGA

F. Comparison with Previous Works

Several polynomial approximations to the reciprocal have been proposed such as [1, page 373] or [10]. But these works do not include implementation results. Our results have been compared to the results reported in [3] for the same accuracy (11 bits) on the same FPGA device and using the same implementation constraints (synthesis effort and place and route effort).

The operator proposed in [3] leads to an area of 178 slices and a delay of 11.9 ns. The operator presented here is 40% smaller but 50% slower. This is due to the fact that the operator presented in [3] is only based on a table and an addition (it uses a degree-1 polynomial). The speed of our operator can be significantly increased using intermediate registers.

VI. CONCLUSION

We have presented a dedicated operator for reciprocation in hardware. Using some specific non-standard decomposition of the domain and quantification of some polynomial coefficients it leads to small and fast hardware implementations. It uses a degree-3 polynomial approximation that only requires one complete multiplication and a small number of additions or subtractions.

Future prospects include the design of pipelined versions of our operator to reach smaller delay and higher throughput. We also plan to study the impact of a larger number of sub-intervals on the coefficients and the accuracy. Finally, we also plan to try to extend this method to other functions.

REFERENCES

[1] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann, 2003.

[2] J.-M. Muller, *Elementary Functions: Algorithms and Implementation*, 2nd ed. Birkhäuser, 2006.

[3] M. D. Ercegovac, J.-M. Muller, and A. Tisserand, "Simple seed architectures for reciprocal and square root reciprocal," in *Proc. 39th Asilomar Conference on Signals, Systems and Computers*. Pacific Grove, California, U.S.A.: IEEE, Oct. 2005, pp. 1167–1171.

[4] A. Tisserand, "High-performance hardware operators for polynomial evaluation," *Int. J. High Performance Systems Architecture*, vol. 1, no. 1, pp. 14–23, Mar. 2007.

[5] Maplesoft, "The maple computer algebra."

[6] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. SIAM, 2002.

[7] D. Ménard and O. Sentieys, "Automatic evaluation of the accuracy of fixed-point algorithms," in *Proc. Design, Automation and Test in Europe (DATE)*, Mar. 2002, pp. 529–537.

[8] G. Melquiond, "GAPPA: génération automatique de preuves de propriétés arithmétiques," http://lipforge.ens-lyon.fr/www/gappa/, 2006, LIP, ENS-Lyon.

[9] The Coq Development Team, "The Coq proof assistant," http://coq.inria.fr/, 2004, INRIA.

[10] M. Ito, N. Takagi, and S. Yajima, "Efficient initial approximation for multiplicative division and square root by a multiplication with operand modification," *IEEE Transactions on Computers*, vol. 46, no. 4, pp. 495–498, Apr. 1997.