

Implementation of an Algorithm to Calculate the Inverse Square Root Operation in a Microcontroller

L. Orozco, M. F. Rocha, C. A. Calva, M. R. Osnaya, M. I. Rocha, R. Navarrete, E. Andrade, and C. Solis

Abstract—At the present time, digital signal processing algorithms use elementary operations such as addition, subtraction, multiplication and division. Using the elementary operations more complex algorithms can be implemented. In the case of inverse square root operation ($a^{-1/2}$) there are direct instructions to calculate it in “C” language, however, the processing time of these instructions is very large compared with the processing time of basic instructions, because their algorithms are complicated. Another aspect to consider is the used hardware; in particular, the memory is consumed very much, for this reason it is necessary to design faster firmware to obtain optimal performance in digital signal processing algorithms.

In this paper, an algorithm to calculate the inverse square root operation in Fixed-Point arithmetic is implemented in a microcontroller using Newton-Raphson Method and Least Squares Method. The goal is to reduce the processing time compared with the required processing time used in Floating-Point arithmetic.

Index Terms—Firmware, fixed-point, inverse square root algorithm, microcontroller.

I. INTRODUCTION

The objective of this paper is to implement an algorithm to calculate the inverse square root (ISQRT) operation by means of Fixed-Point arithmetic for reducing the processing time used in Floating-Point arithmetic. The algorithm is implemented in an AVR microcontroller using “C” language. To talk about the algorithm it is necessary to talk about Floating-Point and Fixed-Point arithmetic.

Floating-Point is the representation of a real number with fractional and integer part. The operations such as addition subtraction, multiplication and division in Floating-Point need some special algorithms to calculate their results [4], for this reason, their handling is a bit complicated. Fig. 1 shows the multiplication algorithm, the algorithm shows that to solve an operation using Floating-Point some subroutines are required and therefore their processing time is large.

Manuscript received July 22, 2012; revised September 3, 2012. This work was supported in part by the Instituto Politécnico Nacional, Escuela Superior de Ingeniería Mecánica y Eléctrica and PIFI of Mexico City. (sponsor and financial support acknowledgment goes here).

Leonardo Orozco, Miguel Rocha, César A. Calva, Ma. Del Rosario Osnaya and Rafael Navarrete are with the Escuela Superior de Ingeniería Mecánica y Eléctrica, Instituto Politécnico Nacional, México (e-mail: leo_4_oro@hotmail.com; mrocha@ipn.mx).

M. Isabel Rocha is with the Department of G. F. O., D- I. E, Universidad Politécnica de Valencia, España (e-mail: marocga@doctor.upv.es).

E. Andrade and C. Solis are with the Department of Physics, Universidad Nacional Autónoma de México.

Moreover, to represent a real number in Fixed-Point, it is necessary a quantity of bits (word length) and a fixed point to separate the fractional part from the integer part of the number [3], this representation is shown in Fig. 2 where a quantity of bits is assigned to represent the fractional part and another quantity of bits for the integer part. The point is just an abstraction, that is to say, its position is imagined and the value of the generated code is interpreted.

Using bigger quantity of bits for the representation, the resolution of the number in Fixed-Point is better. Moreover, arithmetic operations in Fixed-Point are calculated by means of algorithms, however, to reduce processing time is possible to use bit shifting techniques for some special cases, and in this way avoiding the algorithms. For the division a special case is when an unsigned integer number (dividend) is divided by a power of 2, in other words, the divisor is of the form 2^n (where n is a positive integer number); to solve this operation quickly, it is possible right shift n bits, that is to say, every bit in the dividend is simply moved n bit positions, and the vacant bit-positions are filled in with zeros.

For the multiplication a special case is when a multiplicand is a power of 2 (2^n); to solve this operation quickly, it is possible left shift n bits in the other multiplicand, and the vacant bit-positions are filled in with zeros [2].

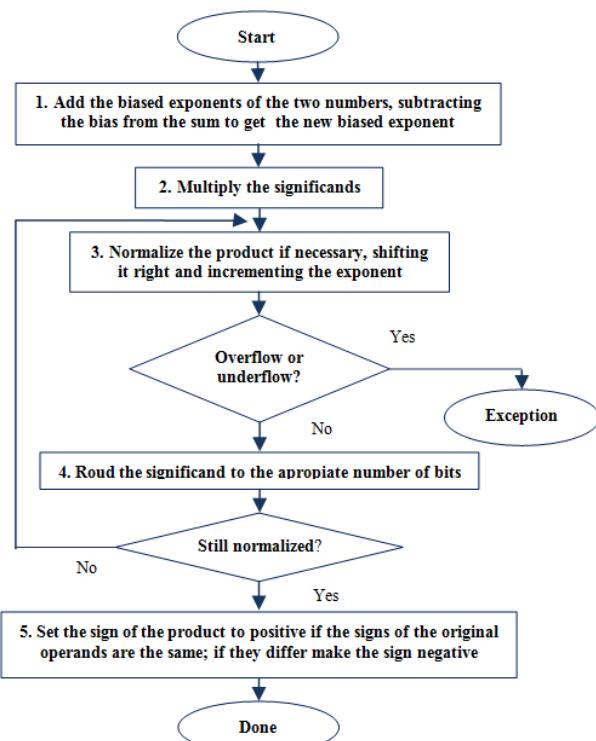


Fig. 1. Algorithm to multiply two numbers in floating-point.

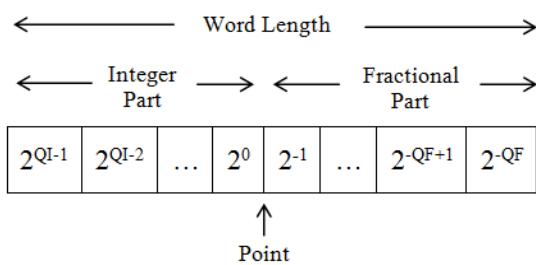


Fig. 2. Representation of a real number using fixed-point.

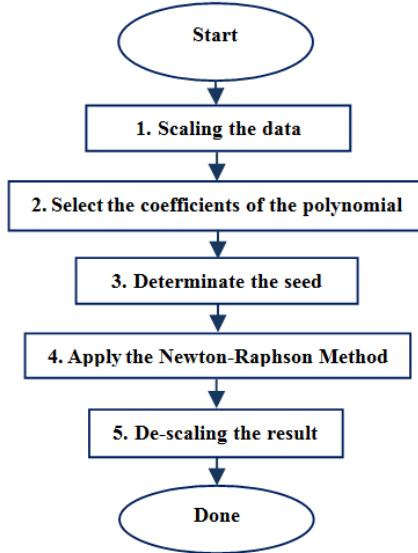


Fig. 3. Algorithm to calculate the inverse square root operation.

II. METHODOLOGY

The algorithm to calculate the ISQRT is shown in Fig. 3, it is a single algorithm and it is explained next. The goal is to solve the inverse square root operation of a positive real number using Newton-Raphson Method and Least Squares Method; consider the equation 1.

$$f(x) = \frac{1}{x^2} - a \quad (1)$$

The reason to use the equation 1 is because its root corresponds to the ISQRT, it is shown in equation 2.

$$x = \frac{1}{\sqrt{a}} \quad (2)$$

CodevisionAVR was used to program an ATMEGA-8 which is a 8-bit, RISC microcontroller and Harvard architecture [1], it defines 16 bites for an unsigned int variable [2]. We use an unsigned int variable and a Fixed-Point to represent the radicand, we use the point to assign 5 bits for integer part and 11 bits for fractional part. Equation 2 can be solved for any radicand greater than 0, but, because there are only 5 bits to represent the integer part, this implementation is limited to solving equation 2 for a radicand greater than 0 and less than 32.

A. Scaling

Scaling is the first step in the algorithm, consider the equation 3 which is an equivalent form of equation 2.

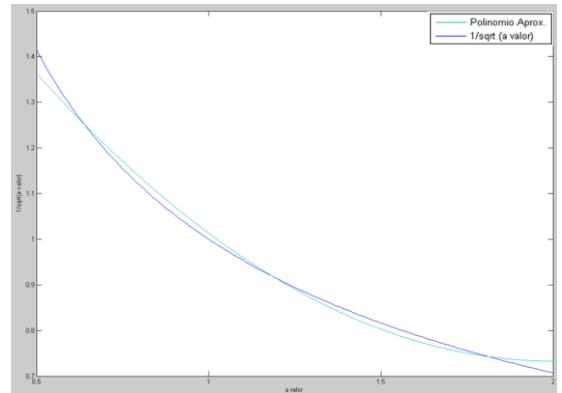


Fig. 4. Differences between polynomial approach and inverse square root function. Purple line is the inverse square root function and blue line is the polynomial approach.

The radicand of equation 3 is the original radicand of equation 2 (a) but it is divided by a power of two, this division is called scaling and it consist in finding an even value for n to make that the new radicand of equation 3 to be contained in the interval from 0.5 to 2. The value of n is positive when the radicand of equation 2 is more than 2, but, it is negative when this radicand is less than 0.5.

$$x = \frac{1}{2^{\frac{n}{2}}} \sqrt{\frac{a}{2^n}} \quad (3)$$

$$\text{where } 0.5 < \frac{a}{2^n} < 2$$

We use the Least Squares Method to obtain a quadratic polynomial approach which approximates the inverse square root function. If the interval is short, then a lower error is obtained between polynomial approach and inverse square root function. For this reason we prefer to use the interval from 0.5 to 2 for the radicand of equation 3. However, the interval of the original radicand of equation 2 remains the same ($0 < a < 32$).

B. Select the coefficients of the polynomial approach

Fig. 4 shows a notable difference between the polynomial approach and inverse square root function. To reduce the difference between both curves, the interval from 0.5 to 2 was divided to obtain smaller intervals and therefore new quadratic polynomials approach, in this way the interval that contains to the radicand obtained from scaling is which determines the coefficients of the polynomial approach to use.

C. Determine the seed

Newton-Raphson Method allows obtaining the root of equation 1, that is to say, it allows calculating the result of the ISQRT operation which is showed in the equation 2. The iterative function of Newton-Raphson Method is shown in equation 4.

$$x_{n+1} = x_n - \frac{f(x_n)}{\frac{df(x_n)}{dx}} \quad (4)$$

Equation 4 needs a starting value x_0 , this initial value is called seed and it is essential to assure that the Newton-Raphson Method converges to the solve.

To obtain the seed, the radicand obtained in the scaling process (radicand of equation 3) is evaluated in the selected polynomial approach.

D. Apply the Newton-Raphson Method

The equation 4 is difficult to implement in firmware. The main problem is the derivative of the function $f(x)$ and another problem is the function divided by its derivative; to solve these operations it is necessary to use especial algorithms. To avoid these problems, the equation 4 is modified using the equation 1 and its derivative to obtain the equation 5.

$$x_{n+1} = \frac{x_n}{2} (3 - ax_n^2) \quad (5)$$

Newton-Raphson Method consists in evaluating the seed (x_0) in the equation 5 and obtaining the result called x_1 , making once iteration, a new result called x_2 will be obtained by means of x_1 ; making more iterations the new result will approach more to the result of the ISQRT operation until both are the same. In other words, to execute the Newton-Raphson Method allows calculating the value of ISQRT operation for the radicand obtained in the scaling process.

Another important point is the division in equation 5; it can be executed quickly by means of bit shifting techniques.

E. De-scaling the result

The modification made in equation 3 to the original radicand from the equation 2 during the scaling process should be compensated by means of de-scaling process. The de-scaling process consists in a division; that is to say, the result obtained in the application of the Newton-Raphson Method is divided by $2^{(n/2)}$; the value of n is the same that was used in scaling process.

III. RESULTS AND DISCUSSION

CodevisionAVR in "C" language was used to implement the algorithm in an ATMEGA-8 microcontroller adjusted to a frequency of 4 MHz. The used variables were unsigned int, and they are defined in CodevisionAVR as 16 bits variables, therefore, word length is defined as 16. Using the point is possible to define 5 bits for integer part and 11 bits for fractional part. In this way, the minimum number to represent is zero (00000.0000000000_2) and the maximum number is 31.99951172 (11111.1111111111_2) but it is an interpreted value because it is really a binary number without point (11111111111111_2) and its equivalent decimal code is 65535, remember that the point is just an abstraction.

After the implementation of the algorithm, a proof value was proposed to test the algorithm. The proof value was an interpreted value of 2.0, which is really a decimal code of 4096; the algorithm calculated the ISQRT for the proof value, the result was a decimal code of 1448, using the abstraction of the point is possible to interpret the decimal code as 0.70703125.

The processing time of the algorithm was measured and the obtained value was 322.5 microseconds, it is shown in Fig. 5. The algorithm used 10.3 % of the available space in the flash memory.

Using "C" language and the math.h library, the direct instructions to calculate the ISQRT operation using Floating-Point were implemented in a program. A proof value of 2.0 was proposed to test the program, the result was 0.70710678. The program used 9.6% of the available space in the flash memory. Fig. 6 shows the used processing time in Floating-Point; in this case it was 1341.75 microseconds.

Comparing both results is possible to appreciate that the algorithm in Fixed-Point is faster than the program in Floating-Point, but the cost is the precision, it is better in Floating-Point; in Fixed-Point, precision is not the best because the fractional part is short to represent the number.

After that, one hundred tests were made, some results of these tests are shown in Table I and Table II; tests consist in generate a random value in decimal code for the radicand, this value is in the interval from 1 to 65535, using the abstraction of the point, this interval can be interpreted as an interval from 0.000488281 to 31.99951172.

Processor	
Name	Value
Program Counter	0x000011D
Stack Pointer	0x045F
X pointer	0x0B4F
Y pointer	0x0158
Z pointer	0x05A7
Cycle Counter	1290
Frequency	4.0000 MHz
Stop Watch	322.50 us
SREG	0000H S V N Z C
Registers	

Fig. 5. Processing time of the algorithm to solve the inverse square root operation in the ATMEGA-8 microcontroller, using fixed-point arithmetic.

Processor	
Name	Value
Program Counter	0x0000077
Stack Pointer	0x045F
X pointer	0x5E2E
Y pointer	0x015C
Z pointer	0x04F3
Cycle Counter	5367
Frequency	4.0000 MHz
Stop Watch	1341.75 us
SREG	0000H S V N Z C
Registers	

Fig. 6. Processing time of the instructions of the math library to calculate the inverse square root operation in an ATMEGA-8 microcontroller, using floating-point arithmetic.

TABLE I: RESULTS OF ISQRT OPERATION IN FIXED-POINT.

Radicand	Radicand	Result of ISQRT (algorithm)	Result of ISQRT (algorithm)
Interpreted value	Decimal code	Interpreted value (Fixed-Point)	Decimal code
1.4140625	2896	0.84082031	1722
3.51269531	7194	0.53369141	1093
12.375	25344	0.28417969	582
5.03515625	10312	0.44580078	913
0.5	1024	1.4140625	2896
22.0986328	45258	0.21289063	436
27.1293945	55561	0.19189453	393
8.63671875	17688	0.34033203	697
30.1967773	61843	0.18212891	373
0.00097656	2	overflow	overflow

TABLE II: RESULTS OF ISQRT OPERATION IN FIXED-POINT AND FLOATING-POINT

Radicand	Result of ISQRT (math library)	Result of ISQRT (algorithm)	Percent Relative error
Interpreted value	Floating Point	Interpreted value (Fixed-Point)	%
1.4140625	0.84094133	0.84082031	0.014391016
3.51269531	0.53355569	0.53369141	0.025436895
12.375	0.284267622	0.28417969	0.030932755
5.03515625	0.445649603	0.44580078	0.033922839
0.5	1.41421356	1.4140625	0.010681555
22.0986328	0.212724395	0.21289063	0.078145703
27.1293945	0.191990593	0.19189453	0.050035161
8.63671875	0.340271528	0.34033203	0.017780492
30.1967773	0.181978341	0.18212891	0.082739956
0.00097656	32.00004096	overflow	-

The implemented algorithm solved the ISQRT operation for the radicand expressed in decimal code, the result was expressed in decimal code too. Using the abstraction of the point is possible to interpret the result as a real number (fixed-Point). Table I shows the interpreted value of the radicand, the radicand expressed in decimal code, the result of the algorithm implemented to solve the ISQRT operation in decimal code and its interpreted value.

Table II shows the interpreted value of the radicand, the result of the ISQRT operation in floating point using math library, the interpreted value of the result in Fixed-Point (result obtained from the algorithm to solve the ISQRT operation) and finally the Percent relative error between the result in floating point and the result in fixed-Point.

IV. CONCLUSIONS

If you need a small processing time, you could use Fixed-Point because it is faster than Floating Point. Use Fixed-Point arithmetic allows reducing the processing time. However, it generates an error because the resolution to represent a real number is limited by the defined word length.

Moreover, the seed value is essential to assure that the Newton-Raphson Method converges to the solution. If the value of the seed is near to the root of the function, then, less iteration is needed to find this root. In this implementation, the correct seed value is not a problem because it was obtained with a quadratic polynomial approach.

The implementation of the algorithm to solve the inverse square root in Fixed-Point is approximately 4 times faster than the implementation of direct instructions using Floating-Point; however, the memory consumed by the algorithm is slightly bigger than the memory consumed for the direct instructions of the math library.

ACKNOWLEDGMENT

IPN-PIFI for the facilities to do this paper.

Centro de Investigaciones Avanzadas del IPN, unidad Gdl. for the facilities to do this paper.

Roberto Leyva Hernández for his collaboration in the implementation of the algorithm and for his cooperation in research.

REFERENCES

- [1] ATMEL ATMEGA-8 Datasheet. [Online]. Available: <http://www.atmel.com/Images/doc2486.pdf>
- [2] R. H. Barnett, S. Cox, and L. O'Cull, *Embedded C programming and the Atmel AVR*, 2nd edition, Thomson Delmar Learning, 2007, vol. 13, pp. 5.
- [3] J. G. Proakis, *Tratamiento Digital De Señales, Principio, Algoritmos Y Aplicaciones*, 3rd edition, Prentice Hall, 1998.
- [4] J. M. Muller, *Elementary Functions, Algorithms and Implementation*, 2nd edition, Birkhäuser. Boston 2006.



Leonardo Orozco was born in Mexico City on October 22th 1990. He has got his technical degree in telecommunications at Centro de Estudios Científicos y Tecnológicos “Carlos Vallejo Márquez”, Ciudad de México, México, in 2008. At the present time, he is studying his engineering degree in Communications and Electronics at Escuela Superior de Ingeniería Mecánica y Eléctrica of Instituto Politécnico Nacional, Ciudad de México, México. He has participated in two scientific research stays at Centro de Investigaciones Avanzadas, IPN unidad Gdl, and Universidad Politécnica de Puebla, México. He has presented some papers in his country and recently in Dubai, UAE. His research interests are in the areas of electronics embedded systems and telecommunications.