IEEE 754-2008 COMPLIANT DECIMAL FLOATING POINT DIVIDER

By

Ahmed Hamdy Ahmed Khalil

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
ELECTRONICS AND COMMUNICATIONS ENGINEERING

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2015

IEEE 754-2008 COMPLIANT DECIMAL FLOATING POINT DIVIDER

By

Ahmed Hamdy Ahmed Khalil

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science
in
Electronics and Communications Engineering

Thesis Advisor
Prof. Dr. Hossam A. H. Fahmy

Prof. of Computer Arithmetic, Faculty of Engineering, Cairo University

Faculty of Engineering, Cairo University

Giza, Egypt

2015

IEEE 754-2008 COMPLIANT DECIMAL FLOATING POINT DIVIDER

By

Ahmed Hamdy Ahmed Khalil

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the

Requirements for the Degree of

Master of Science
in

Electronics and Communications Engineering

Approved by the Examining Committee

_____

Prof. Dr. Hossam A. H. Fahmy, Thesis Main Advisor

_____

Associate Prof. Dr. Ahmed Nader Mohieldin, Member

_____

Prof. Dr. El-Sayed Mostafa Saad, Member

(Prof. of Electronic circuits, Faculty of Engineering, Helwan University)

Faculty of Engineering, Cairo University

Giza, Egypt

2015

Acknowledgements

First of all I must thank ALLAH for his great mercy supporting me all the way till the end. If it weren't for his help, I wouldn't have reached this point.

I would like to thank my advisor, Prof. Hossam A. H. Fahmy for giving me the opportunity to work in a fruitful research environment and for his continuous guidance and support, as well as for his successful discussion and encouragement.

Most importantly, I would like to thank my family for their continuous encouragement, and for helping me all through my work.

Contents

List of Figures

List of Tables

List of Abbreviations

AQA	Accurate Quotient Approximation
BFP	Binary floating point
BCD	Binary coded decimal
BID	Binary integer decimal
CSA	Carry save adder
CPA	Carry propagation adder
DFP	Decimal floating point
DPD	Densely packed decimal
FO4	Fan out of four
MSD	Most significant digit
MSB	Most significant bit
MUX	Multiplexer
NaN	Not a Number
LZV	Leading zeros value
PP	Partial product
qNaN	Quite Not a Number
sNaN	Signal Not a Number
SRT	Sweeney, Robertson, and Tocher division algorithm
ZF	Zero flag

Abstract

The binary numbering system is the most suitable for electronic computers, and the decimal numbering system is the most used in human life especially in commercial applications. But the binary system can not represent decimal fraction numbers accurately, so the IEEE 754-2008 standard defines the decimal floating numbers and decimal floating arithmetic operations. The decimal floating point division is one of these operations.

This thesis presents new design and implementation for decimal floating point divider based on Accurate Quotient Approximation algorithm which can be considered as very high radix decimal divider. Accurate Quotient Approximations is an iterative division algorithm based on look-up table that gives an approximation to the reciprocal of the divisor. The algorithm iterates by using the reciprocal to find an approximated value for the quotient. Then the approximated quotient is multiplied by the divisor and the result is subtracted from the partial remainder to find the new partial remainder. At first iteration the partial remainder is the dividend. The divider iterates until the quotient reaches the desired accuracy. This divider can produce three digits per iteration. The operation of this divider is compliant to IEEE 754-2008 standard. Also it supports the five rounding modes that are defined in the IEEE 754-2008 standard, and another two famous rounding methods. This divider is extended to execute decimal floating point division operations.

The functionality of the proposed decimal floating point divider is verified by around one million test cases that are designed to test decimal floating point units. Also the design is synthesized by using Synopsys design compiler tool working on 65nm technology which shows that the critical path delay is 48.2 FO4 and area equals 156842 NAND2 gate.

The main goal of this thesis is to design and implement new decimal floating point divider functionally correct. The proposed DFP divider can produce three quotient digits in each iteration, which can reduce the overall latency.

Chapter 1. Introduction

The decimal number system is the most widely used system for human calculations. So that at the beginning of computer machines age all machines were based on decimal system. Mechanical computers mimicked human numeration systems for scientific and commercial calculations, and the most used system was decimal system. At the start of electronic computer industry many computers used decimal system in arithmetic operations such as ENIAC [1] and IBM 650 [2].

However the nature of memory units and flip-flops is binary and it is required four binary bits to represent each decimal digit which consumes more memory than binary system. Furthermore the binary system arithmetic units are simpler and faster than decimal system arithmetic units. Therefore Burks, Goldstine and von Neumann [3] discussed the superiority of binary system over decimal system. They announced that using binary system for addressing, data storage and arithmetic operations is more suitable for electronic computers due to the nature of these computers which have two statuses for the signal. Therefore the binary system dominates most of computers nowadays and there are few computers based on decimal system.

The rest of this chapter is organized as follows: Section1.1 explains the increasing importance of decimal arithmetic. Section1.2 discusses the decimal floating point standard format with its arithmetic operations. And finally section 1.3 surveys the recently published hardware implementations for different decimal floating point operations.

1.1 The Need for Decimal Floating Point Arithmetic

Although binary based computers dominate the world, decimal computations can't be ignored. Decimal numeration system is essential for many applications [4]. Databases belong to 51 commercial and financial organizations were surveyed and investigated, these databases include many financial applications such as banking, billing, inventory control, financial analysis, taxes, and retail sales. There were more than 456,420 columns which contained numeric data and were investigated to extract statistic information. This survey reported that 55% were decimal, and that further 43.7% were integer types which could have been stored as decimal numbers [4]. The results of these applications are required to be accurate and rounded correctly to be committed by human manual calculations and law.

For binary based computers, decimal numbers will be converted to/from binary numbers. Decimal numbers may not be converted exactly, due to the lack of binary system accuracy and finite precision hardware. Most of fraction numbers are not converted to binary numbers properly [5], let the decimal number $X_{dec} = 0.6$, to convert this decimal number to binary number it will be $X_{bin} = 0.10011001100110011001100110 \ldots \ldots$ that requires infinite number of bits to be represented exactly in binary which is not available so this number will be approximated, the stored value will be $X_{dec}^{stored} = 0.599999999999999999913263$, so any operation using this number will produce inaccurate results although the arithmetic operation is correct. The decimal to/from binary conversion is implemented using software programs with high delays.

In addition to the accuracy problem there is another problem caused by binary arithmetic is the removal of trailing fraction zeroes. For example, binary system can't distinguish between

1.5 and 1.50 because of the normalization nature of binary system. The trailing fraction zeros are essential in the calculation, they are very important for physics measurement for example if it is reported that, the mass of a body is 10.7 kg versus 10.700 kg the two measures are not the same as the first one is accurate for 0.1 kg but the second one is accurate for 0.001 kg. Hence binary arithmetic units can't be used directly for financial application and decimal arithmetic operations as they produce results not compatible with law and human requirements [5].

To avoid the drawbacks of using binary arithmetic units for decimal calculations, researchers and companies made a lot of efforts to overcome these drawbacks. Finally we can find two methods to deal with decimal calculations the first, is to keep decimal numbers without conversion and software libraries will execute the decimal calculations using binary floating point (BFP) arithmetic units. The most widely used libraries are Sun's BigDecimal for Java [6], IBM's decNumber library [7], and Intel's Decimal Floating-Point Math library [8]. But the performance of software libraries is very bad, as the performance can reach 1000 time slower than implementing dedicated hardware [9].

The other method is to implement pure hardware for decimal operations to overcome the software bad performance, IBM presented Z900 as one of the first microprocessor that contains decimal integer arithmetic unit [10], although this unit is limited on decimal integers, decimal fixed point operations can be executed by scaling the operand using software libraries.

Fixed point and integer arithmetic units can generate accurate results, but the scaling of floating-point numbers, and rounding of the final result limit their usage. The width of these units is limited by the format precision, so when using fixed point and integer arithmetic units for floating point operations the input operands will be scaled to convert them to integer numbers this scaling operation is implemented using software programs, this process is difficult to be executed and can result in errors specially when using very large values and very small values for the same operation [4].

Although rounding is very important for financial and commercial applications, fixed point and integer arithmetic units don't implement round process in hardware but it is required to use software to do that. Using software for scaling and rounding processes impose high delay on the operation, also consumes more power than using dedicated hardware for all operation [4].

Therefore it is required to implement pure hard ware for decimal floating point arithmetic to avoid drawbacks of binary floating point arithmetic units and to save time and power of software used to adjust the operation of fixed point and integer arithmetic units.

1.2 IEEE 754-2008 Standard

IEEE 754-1985 is the most widely used standard for binary floating point arithmetic, and is implemented for many microprocessor designs and software libraries [11]. But this is a binary floating-point standard so it cannot be used for decimal arithmetic. Then IEEE published floating point standard IEEE 854-1987 [12], which was radix independent, and this standard was designed mainly for scientific and engineering applications, so this standard didn't find a way for commercial needs. Due to booming of decimal floating point arithmetic

and there was not any standard to identify decimal floating point (DFP) arithmetic, IEEE published the IEEE 754-2008 in August 2008 [13].

IEEE 754-2008 standard defines decimal floating-point data formats and operations as follow,

1.2.1 DFP Formats and Encoding

The standard specifies three formats for floating point decimal numbers, two are basic formats (decimal64 and decimal128) and one is storage format (decimal32), as shown in Table 1.1. The basic formats are used for DFP arithmetic operations, and the storage format is not used by DFP arithmetic operations. These formats can represent positive and negative values within the range of the used precision format, $\pm 0, \pm \infty$, and NaN (Not a Number).

Decimal format	Decimal 32	Decimal 64	Decimal 128
Total storage width	32	64	128
Combinational field	11	13	17
Trailing significand field	20	50	110
Total significand digits	7	16	34
Maximum Exponent	96	384	6144
Minimum Exponent	-95	-383	-6143
Exponent Bias	101	398	6176
Exponent width	8	10	14

Table 1.1: Decimal formats defined in IEEE 754-2008

The representation of decimal floating-point number is:

$$(-1)^S \times \mathrm{C} \times 10^{\mathrm{q}} \tag{1.1}$$

Where S is the sign, q is the exponent, $\mathrm{C} = (\mathrm{d}_{\mathrm{p}-1}\mathrm{d}_{\mathrm{p}-2} \ldots\ldots \mathrm{d}_0)$ is the significand, or coefficient, where $\mathrm{d}_\mathrm{i} \in \{0,1,2,3,4,5,6,7,8,9\}$, and p is the precision. There are two restrictions on C and q [13]:

- C must be an unsigned integer between 0 and $(10^{\mathrm{p}} - 1)$. Notice that in the standard, C is not required to be normalized. So we can find different representation for the same number, let 500×10^3 and 5×10^5 have same value, and both belong to the same cohort.
- q must be an integer satisfying equation, $1 - \mathrm{E}_{\min} \leq \mathrm{q} + \mathrm{p} - 1 \leq \mathrm{E}_{\max}$.

Figure 1.1 shows the decimal interchange floating-point format adopted in the IEEE 754-2008 standard which includes,

S Sign Field	G Combination Field	T Trailing Significand Field

Figure 1.1: Decimal interchange floating-point format

Sign bit S represents the sign of the decimal number, it has two statuses one or zero then the decimal number will be negative or positive respectively.

The second part is the combination field G; it can be divided into two subfields. The first subfield consists of the most significant five bits of G, is used to encode the MSD of the significand, the most significant two bits of the exponent, and is used to indicate NaN and infinity cases as shown in table 1.2. The second subfield consists of the remaining bits of the biased exponent, as the exponent is encoded in binary excess-code and the bias value differs according to the precision format as shown in table 1.1.

The remaining part is the trailing significand field T this field consists of the remaining fifteen digits of the significand. For trailing significand field value there are two types of encoding formats defined in the standard. Densely Packed Decimal "DPD" is introduced by IBM. DPD format encodes every three decimal digits using 10 bits [14]. DPD format is efficient in memory storage and data buses usage than Binary Coded Decimal "BCD" which uses 4 bits for each single digit. But it is not easy to use DPD directly for arithmetic operations, so there are need to convert from DPD format to/from BCD format [5]. Conversion process can be implemented using dedicated circuits which can cost low delay. Intel presented Binary Integer Decimal "BID"; here the entire trailing significand field value is converted from decimal form to binary form to be stored [15]. The BID encoding format is suitable for software implementation, to use the binary integer arithmetic logic unit in the current microprocessors [5].

Special values are encoded as follow:

- If the most significant five bits of the combination filed are 11110, then we have $\pm\infty$, according to the sign bit.
- If the most significant five bits of the combination filed are 11111, then we have NaN case, and if the most significant sixth bit is 1 hence we have sNaN otherwise we have qNaN, and all the other bits are ignored. sNaN represents values for uninitialized variables or missing data samples. qNaN results from any invalid operations or operations that involve qNaN as operand.
- Overflow occurs when the absolute value of the result is greater than the maximum available value$(10^{p} - 1) \times 10^{E_{max}-p+1}$.
- Underflow occurs when the result absolute value is smaller than $10^{E_{min}-p+1}$ and not equals the zero.
- Subnormal number has a value between $10^{E_{min}}$ and $10^{E_{min}-p+1}$. Subnormals fill the underflow gap around zero in DFP arithmetic.

G's MSD(5-bits)	type	Exponent's MSB (2-bits)	Significand's MSD
$a_i a_{i-1} a_{i-2} a_{i-3} a_{i-4}$	$D_0 \leq 7$	$a_i a_{i-1}$	$0 a_{i-2} a_{i-3} a_{i-4}$
$11 a_{i-2} a_{i-3} a_{i-4}$	$D_0 > 7$	$a_{i-2} a_{i-3}$	$100 a_{i-4}$
$11111 - \begin{matrix}0\\1\end{matrix}$	qNaN sNaN	-	-
11110	Infinity	-	-

Table 1.2: Decoding combination field