Cairo University

SUPPORT OF FMA IN OPEN-SOURCE PROCESSOR

By

Ahmed Ali Ismail Ali Mohamed

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
2016

SUPPORT OF FMA IN OPEN-SOURCE PROCESSOR

By
Ahmed Ali Ismail Ali Mohamed

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

Under the Supervision of

Prof. Dr. Hossam A. H. Fahmy

Professor,
Electronics and Communications
Engineering,
Faculty of Engineering, Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2016

SUPPORT OF FMA IN OPEN-SOURCE PROCESSOR

By
Ahmed Ali Ismail

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

____
Prof. Dr. Ibrahim Mohamed Qamar, Internal Examiner

____
Prof. Dr. Ashraf M. Salem, External Examiner
(Faculty of Engineering, Ain Shams University)

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2016

Engineer's Name:	Ahmed Ali Ismail Ali Mohamed
Date of Birth:	04/02/1987
Nationality:	Egyptian
E-mail:	Ahmed_Ismail@mentor.com
Phone:	01001708043
Address:	3 Ibn el Ekhsheed st., Dokki, Giza
Registration Date:	01/10/2010
Awarding Date:	2016
Degree:	Master of Science
Department:	Electronics and electrical communications

Supervisors:

Prof. Dr. Hossam A. H. Fahmy

Examiners:

Prof. Dr. Hossam A. H. Fahmy, Thesis main advisor
Prof. Dr. Ibrahim Mohamed Qamar, Internal examiner
Prof. Dr. Ashraf M. Salem, External examiner, Faculty of Engineering, Ain Shams University

Title of Thesis:

SUPPORT OF FMA IN OPEN-SOURCE PROCESSOR

Key Words:
FPU; FMA; Processor; ISA; Verification

Summary:
In this work, we have added the support of the Fused Multiply-Add (FMA) unit in OpenSparc T2 open-source processor. The FMA unit used supports both binary and decimal formats. The used FMA optimizes the area and power consumption by sharing most of the hardware between the binary and decimal operations.
The work done includes modifying the processor Instruction Set Architecture (ISA) to support the new operations, integrating the FMA unit inside the floating point unit of the processor, updating the processor to understand the new instructions and communicate correctly with the new unit. The work done also includes modifying the assembler to understand the assembly of the new instructions and generates the executable accordingly.
During our work we verified the FMA unit using Formal Verification technology and found and fixed many bugs in the implementation. We also proposed a methodology for verifying the floating point units using Formal Verification.

Acknowledgments

Praise be to Allah, Lord of the Worlds for all his blessings, and peace be upon prophet Mohamed and his companions.

I want to thank my family and wife for their invaluable support. Also thanks to all my friends for their help and support.

Finally, I would like to express my sincere gratitude to my advisor Prof. Hossam Fahmy for his support, patience and encouragement.

Table of Contents

List of Tables

List of Figures

Abstract

In this work, we have added the support of the Fused Multiply-Add (FMA) unit in OpenSparc T2 open-source processor. The FMA unit used supports both binary and decimal formats, allowing us to complete the support for the binary floating point operations in the aforementioned processor since it was missing the FMA operations as well as adding initial support for decimal floating point operations which were totally missing in the processor. The used FMA optimizes the area and power consumption by sharing most of the hardware between the binary and decimal operations.

The support of more functionality on the processor hardware helps in improving the overall processing time, compared to the software implementations of the same functionality where the unsupported hardware instruction is replaced by multiple simpler instructions. The area considerations for the new hardware support can be minimized by optimizing the hardware implementation and reusing the hardware units in different operations. Also using newer technology with smaller feature size can reduce the overall area needed.

The work done includes modifying the processor Instruction Set Architecture (ISA) to support the new operations, integrating the FMA unit inside the floating point unit of the processor, updating the processor to understand the new instructions and communicate correctly with the new unit. The work done also includes modifying the assembler to understand the assembly of the new instructions and generates the executable accordingly.

The new functionality of the processor is verified by updating the processor testing environment with new tests to exercise the new instructions, the old functionality of the processor is also verified in the different scenarios by using the processor available regression tests.

During our work we verified the FMA unit using Formal Verification technology and found and fixed many bugs in the implementation. We also proposed a methodology for verifying the floating point units using Formal Verification.

Chapter 1 : Introduction

1.1. Floating point arithmetic

The floating point arithmetic is used in many applications that require complex calculations and accurate results with large dynamic range. The fixed point arithmetic although much simpler and can use the integer units in the processor, but it supports very small range of numbers. For the same number of bits, the fixed point numbers have a choice of either precision or supporting large numbers while floating point numbers can support both. Taking an eight bits number as an example, only 256 different numbers can be represented in either fixed or floating point numbers, the selection of the fixed point location will limit both the range and precision of the number to a fixed value. Assuming the point position is selected to be 2 bits from the right, then the maximum fixed point number is 64 and the precision is 0.25. On the other side if we defined a floating point number with 2 bits to define the point position within the least significant 6 bits then we can reach the same maximum value but with higher precision of 0.0625. The floating point benefits will come with the cost of adding extra complexity in the calculations which turns into extra delay and larger hardware area.

Floating point operations can be done on any processor even if the processor has no floating point support on the hardware. However, the usage of the software libraries to perform the floating point operations slows down the computation. A dedicated floating point unit (FPU) is supported in many processors today since doing the operation on hardware saves both time and power [1].

Benchmarking for the support of decimal floating point (DFP) in hardware versus the support in software has been done in [2], authors have concluded that large improvement in the DFP applications is achieved when having the support in hardware. The benchmark results showed that more than 75% of the execution time is spent in DFP functions if evaluated in software. The hardware support speedup ranges from 1.3 to 31.2 on different benchmarks. In [3] the energy-delay product improvement due to the use of hardware support was reported over 500.

1.2. Binary floating point arithmetic

The binary floating point (BFP) units have been available in commercial computers since 1950’s [4]. The numbers in BFP format are represented by three parts: sign, exponent and mantissa. The mantissa is similar to the integer representation and therefore can use the same integer units or techniques for the mantissa calculations. In fact in some processors such as the OpenSparc T2 processor, as we will explain in more details in Chapter 4, the integer and binary floating point multiplication and division are sharing the same units.

1.3. Decimal floating point arithmetic

The main limitation for the BFP arithmetic is the ability to handle the common fractions accurately. The common fraction 0.1 as an example cannot be described accurately using BFP number using finite number of bits. This limitation may cause a large errors in some of the financial applications causing large loss for the companies due to truncation error [5]

Therefore the increasing demand on DFP arithmetic is more obvious in military and financial applications.

1.4. IEEE standard for floating point arithmetic

The floating point arithmetic standard (IEEE 754) was published in 1985 and updated in 2008 (IEEE 754-2008) [6]. The standard was defined to make sure that the results are correct and consistent if the operation is done through hardware unit, software library, or combination of both. The software development can be compatible across different machines if the operations are following the standard. The standard specifies binary and decimal formats for the floating point numbers. The standard specifies five basic formats which are three binary formats with encodings of lengths 32, 64, and 128 bits (also known as single, double and quad precisions) and two decimal formats with encodings in lengths of 64 and 128 bits. The standard also specifies possible extensions to these formats.

The floating point numbers are defined in the following form: $(-1)^s$ x b^e x m, where s is the sign and can take values 0 or 1, b is the radix and can be either 2 for binary and 10 for decimal, e is the exponent and can be any integer between emin and emax (the emin and emax varies from one format to another but will always follow the rule that emin = 1 – emax), and m is the significand of the number. The number of bits in the significand is the precision (p) and the values of each digit in the significand is between 0 and b. The standard defines +ve and –ve zeros. Beside that the standard specifies four more floating point values which are two infinities (+∞ and −∞) and two Not a Number (NaNs) which are qNaN (quiet) and sNaN (signaling).

1.4.1. Binary floating point numbers representation

The binary floating point numbers have the radix of 2. The basic binary floating point formats defined in the standard are represented in Table 1.1

Table 1.1: Binary floating point formats

Parameter	Binary 32	Binary 64	Binary 128
Precision (p)	24	53	113
Emax	127	1023	16383
exponent field width	8	11	15

The encoding for the binary number in each format is unique, i.e. each number can be represented in only one possible encoding. The binary numbers encoding is shown in

Figure 1.1 where the most significant bit (MSB) represents the sign, the next w bits are representing the biased exponent, and the least significant p-1 bits are used for the trailing significand. The biased exponent is defined as E = e + bias where bias is fixed number for every binary format which is equal to emax. The MSB of the significand is hidden so the total number of bits for the significand is p. The hidden bit can be either 0 or 1 according to the exponent value, those are called normal and subnormal numbers respectively.

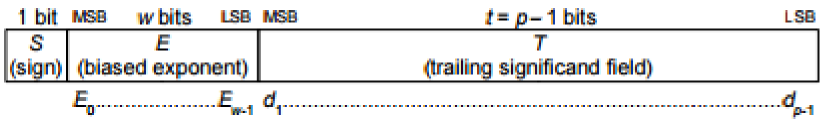

Figure 1.1: Binary floating point encoding

The exponent for normal binary floating point numbers is in the range 1 to $2^w - 2$, the remaining two values for the exponent which are 0 and $2^w - 1$ are reserved for the following special representations:

1. E = 0 is used to encode ± 0 and the subnormal numbers
2. E = $2^w - 1$ is used to encode $\pm\infty$ and the NaNs

The normal binary floating point numbers have a hidden 1 in the significand and are represented as $(-1)^s$ x 2^ex $1.$significand, the largest number that can be represented in this format is $(-1)^s$ x 2^{2w-2}x 1.2^{p-1} while the smallest normal binary floating point number is represented by E=1 and trailing significand (T) = 0 and is equivalent to $(-1)^s$ x $2^{1-\text{bias}}$. The numbers smaller than the smallest normal values are called subnormal and have leading hidden 0, with the exponent bits are all zeros. The maximum subnormal number is $(-1)^s$ x 2^{-bias}x 0.2^{p-1}.

Because of the hidden 1 in the normal binary numbers, the binary operations requires normalization step at the end to bring the result back to the normal form in case the result is not subnormal, this is not always needed in the decimal operations since the result can be un-normalized as shown in next section.

The biased exponent E = $2^w - 1$ is used to represent special values as shown in Table 1.2

Table 1.2: Binary special values encodings

Significand	Special value
= 0	$\pm\infty$
$\neq 0$	qNaN, or sNaN

The 0 binary number is represented by the encoding of E = 0 and T = 0. The standard supports ± 0 which is useful in case of division by zero to identify of the result is +ve or –ve ∞.

1.4.2. Decimal floating point numbers representation

The decimal floating point numbers have the radix of 10. The decimal floating point numbers are more convenient in some applications like the financial and military applications where the error impact can be very large. The decimal floating point numbers are more familiar to the human since it is used in the their normal operations, the decimal floating point numbers can also specify some numbers that the binary cannot specify accurately in finite number of bits such as the number 0.1.

The IEEE 754-2008 added support for the decimal floating point arithmetic, the standard specifies two basic encodings for the decimal formats as explained in Table 1.3.

Table 1.3: Decimal floating point formats

Parameter	**Decimal 64**	**Decimal 128**
Precision (p)	16	34
emax	384	6144
combination field width in bits	13	17

The decimal encoding -unlike the binary one- allows multiple representation for the value, all the representations for the same value are called cohort. The different encodings for the same decimal number allows the system to maintain the precision of the result, for example the two numbers 5 x 10^{-2} and 50 x $10^{-3}$are equivalent but the precision in the second number is greater by 1 digit. The number of available cohorts for each values varies according to the number of trailing zeros in the value as well as the difference between exponent and the maximum and minimum exponents. The maximum number of cohorts for decimal floating point number is equal to the number of digits in the significand of this number. The standard specifies the preferred exponent -out of all the available cohorts- of the number for each operation to make sure that results are consistent across the different implementations.

The decimal numbers encoding is shown in Figure 1.2, the MSB of the number is the sign bit, the next w+5 bits (G) are representing the exponent and the last t trailing bits are representing the trailing significand (T).

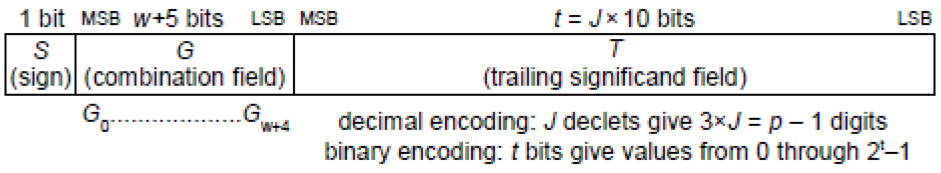
1 bit
MSB
w+5 bits
LSB
MSB
t = J × 10 bits
LSB
S (sign)
G (combination field)
T (trailing significand field)
G0 .... Gw+4
decimal encoding: J declets give 3×J = p – 1 digits
binary encoding: t bits give values from 0 through 2^t–1

Figure 1.2: Decimal floating point encoding

The standard specifies two ways to encode the significand, the first one is the decimal encoding using densely-packed-decimal encoding, the other way is to use binary encoding and consider all the t significand bits as one integer value with range from 0 to $2^t - 1$. The binary encoding can be used efficiently if the decimal floating point operations are done on the software since the operations can reuse the integer execution