Raja Gopal Surineedi, G.S.Siva Kumar, P.Sunitha / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 5, September- October 2012, pp.1221-1225

Arithmetic Unit Implementation Using A FPGA IEEE-754-2008 Decimal64 Floating-Point

Raja Gopal Surineedi, G.S.Siva Kumar, P.Sunitha
M.Tech student, Associate professor, Associate professor,
Pragati engineering college, surampalem,

Abstract:

This paper describes the FPGA implementation of a Decimal Floating Point (DFP) ALU (Arithmetic logic unit). The design performs addition, subtraction, multiplication and division on 64-bit operands that use the IEEE 754-2008 decimal encoding of DFP numbers and is based on a fully pipelined circuit. The design presents a novel hardware for pre-signal generation stage and an enhanced version of previously published leading zero stage. The design can operate at a frequency of 130 MHZ on a CYCLONE-III with a latency of 8 cycles. The presented DFP adder/subtractor supports operations on the decimal64 format and it is easily extendable for the decimal128 format. To our knowledge, this is the first hardware FPGA design for Floating point arithmetic unit based on IEEE 754-2008 using decimal64 encoding.

Keywords - **About five key words in alphabetical order, separated by comma**

I. INTRODUCTION

Digital arithmetic operations are very important in the design of digital processors and application-specific systems. Arithmetic circuits form an important class of circuits in digital systems. With the remarkable progress in the very large scale integration (VLSI) circuit technology, many complex circuits, unthinkable yesterday have become easily realizable today. Algorithms that seemed impossible to implement now have attractive implementation possibilities for the future. This means that not only the conventional computer arithmetic methods, but also the unconventional ones are worth investigation in new designs. The notion of real numbers in mathematics is convenient for hand computations and formula manipulations. However, real numbers are not well-suited for general purpose computation, because their numeric representation as a string of digits expressed in, say, base 10 can be very long or even infinitely long. Examples include π, e, and 1/3. In this thesis an arithmetic unit based on IEEE standard for floating point numbers has been implemented on ALTERA cyclone -III FPGA Board. The arithmetic unit implmented has a 64-bit processing unit which allows various arithmetic operations such as, Addition, Subtraction, Multiplication, Division and, on floating point numbers. Each operation can be selected by a particular operation code. Synthesis of the unit for the FPGA board has been done using QUARTUS-II.

Implemented arithmetic unit is a part of a computer system specially designed to carry out operations on floating point numbers. Some systems (particularly older, microcode-based architectures) can also perform various transcendental functions such as exponential or trigonometric calculations, though in most modern processors these are done with software library routines.

II. Floating Point Formats

Several different representations of real numbers have been proposed, but by far the most widely used is the floating-point representation. Floating-point representations have a base b (which is always assumed to be even) and a precision p. If b = 10 and p = 3 then the number 0.1 is represented as 1.00 × 10-1

If b = 2 and p = 22, then the decimal number 0.1 cannot be represented exactly but is approximately 1.10011001100110011001×2-4. In general, a floating point number will be represented as ± d.dd… d × be, where d.dd… d is called the Significand and has p digits. More precisely ± d0 d1 d2 ... dp-1 × be represents the number. The term floating-point number will be used to mean a real number that can be exactly represented in the format under discussion. Two other parameters associated with floating-point representations are the largest and smallest allowable exponents, emax and emin. Since there are be possible significands, and emax - emin + 1 possible exponents, afloating-point number can be encoded in bits, where the final +1 is for the sign bit. The most common situation is illustrated by the decimal number 0.1. Although it has a finite decimal representation, in binary it has an infinite repeating representation. Thus when b = 2, the number 0.1 lies strictly between two floating-point numbers and is exactly representable by neither of them. A less common situation is that a real number is out of range, that is, its absolute value is larger than b ×bemax or smaller than 1.0 × bemin

III. IEEE 754 Standard for Binary Floating-Point Arithmetic:

The IEEE 754 Standard for Floating-Point Arithmetic is the most widely-used standard for floating-point computation, and is followed by many hardware (CPU and FPU) and software implementations.

The standard specifies:

- Basic and extended floating-point number formats
- Add, subtract, multiply, divide, square root, remainder, and compare operations
- Conversions between integer and floating-point formats
- Conversions between different floating-point formats
- When it comes to their precision and width in bits, the standard defines two groups: basic and extended format. [3,4,8]

A. Formats

The standard defines five basic formats, named using their base and the number of bits used to encode them. There are three binary floating-point formats (which can be encoded using 32, 64, or 128 bits) and two decimal floating-point formats (which can be encoded using 64 or 128 bits). The first two binary formats are the 'Single Precision' and 'DoublePrecision' formats of IEEE 754-1985, and the third is often called 'quad'; the decimal formats are similarly often called 'double' and 'quad'.

Table 2.1: Various basic formats of IEEE 754 standard

parameter → format name	*b* base	*p* (bits or digits)	*emax*
binary32	2	23+1 bits	+127
binary64	2	52+1 bits	+1023
binary128	2	112+1 bits	+16383
decimal64	10	16 digits	+384
decimal128	10	34 digits	+6144

B. Single Precision

The most significant bit starts from the left. The three basic components are the sign,exponent, and mantissa

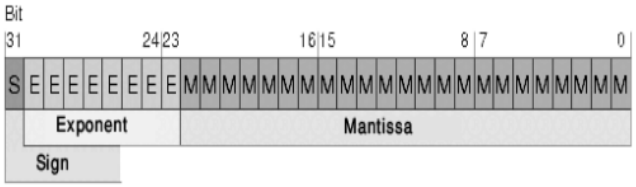

Fig 1.Floating representation of single precision

C. Double Precision

The double precision format helps overcome the problems of single precision floating point. Using twice the space, the double precision format has an 11-bit excess-1023 Precision exponent and a 53 bit mantissa (with an implied H.O. bit of one) plus a sign bit. This provides a dynamic range of about 10±308 and 14-1/2 digits of precision, sufficient for most applications. Double precision floating point values take the form shown in Figure2.2.[3]

Sign(s)	Exponent(e)	Fraction	Value
0	00000000	00000000000000000000000	+0 (positive zero)
1	00000000	00000000000000000000000	-0 (negative zero)
1	00000000	10000000000000000000000	$-2^{0-127} \times 0.(2^{-1}) = -2^{0-127} \times 0.5$
0	00000000	00000000000000000000001	$+2^{0-127} \times 0.(2^{-23})$ (smallest value)
0	00000001	01000000000000000000000	$+2^{1-127} \times 1.(2^{-2}) = +2^{1-127} \times 1.25$
0	10000001	00000000000000000000000	$+2^{129-127} \times 1.0 = 4$
0	11111111	00000000000000000000000	+ infinity
1	11111111	00000000000000000000000	- infinity
0	11111111	10000000000000000000000	Not a Number(NaN)

Table 2.2 Representation of Single Precision floating point numbers

In order to help ensure accuracy during long chains of computations involving double precision floating point numbers, Intel designed the extended precision format. The extended precision format uses 80 bits. Twelve of the additional 16 bits are appended to the mantissa; four of the additional bits are appended to the end of the exponent. Unlike the single and double precision values, the extended precision format does not have an implied H.O. bit which is always one. Therefore, the extended precision format provides a 64 bit mantissa, a 15 bit excess-16383 exponent, and a one bit sign. The format for the extended precision floating point value is shown in Figure 2.3.[3,4,8]

D. Exceptions

The IEEE standard defines five types of exceptions that should be signalled through a one bit status flag when encountered. Some arithmetic operations are invalid, such as a division by zero or square root of a negative number. The result of an invalid operation shall be a NaN (Not a number). There are two types of NaN, quiet NaN (QNaN) and signaling NaN (SNaN). They have the following format, where s is the sign bit:

QNaN = s 11111111 10000000000000000000000
SNaN = s 11111111 00000000000000000000001

The result of every invalid operation shall be a NaN string with a QNaN or SNaN exception. The SNaN string can never be the result of any operation, only

the SNaN exception can be signaled and this happens whenever one of the input operand is a SNaN string otherwise the QNaN exception will be signaled. The SNaN exception can for example be used to signal operations with uninitialized operands, if we set the uninitialized operands to SNaN.

E. Range of Floating Point Numbers

By allowing the radix point to be adjustable, floating-point notation allows calculations over a wide range of magnitudes, using a fixed number of digits, while maintaining good precision. For example, in a decimal floating-point system with three digits, the multiplication that human would write as

$0.12 \times 0.12 = 0.0144$ Would be expressed as

$(1.2 \times 10^{-1}) \times (1.2 \times 10^{-1}) = (1.44 \times 10^{2})$

In a fixed-point system with the decimal point at the left, it would be $0.120 \times 0.120 = 0.014$ A digit of the result was lost because of the inability of the digits and decimal point to 'float' relative to each other within the digit string. The range of floating-point numbers depends on the number of bits or digits used for representation of the significand (the significant digits of the number) and for the exponent. On a typical computer system, a 'double precision' (64-bit) binary floating point number has a coefficient of 53 bits (one of which is implied), an exponent of 11 bits, and one sign bit. Positive floating-point numbers in this format have an approximate range of 10^{-308} to 10^{308} (because 308 is approximately $1023 \times \log10 (2)$, since the range of the exponent is [−1022, 1023]). The complete range of the format is from about -10^{308} through $+10^{308}$..

The number of normalized floating point numbers in a system F(B, P, L, U) (where B is the base of the system, P is the precision of the system to P numbers, L is the smallest exponent representable in the system, and U is the largest exponent used in the system) is:

$2 * (B - 1) * B^{(P-1)} * (U - L + 1) + 1$. The one is added because the number could be zero. There is a smallest positive normalized floating-point number, Underflow level = UFL = B^{L} which has a 1 as the leading digit and 0 for the remaining digits of the mantissa, and the smallest possible value for the exponent. There is a largest floating point number, Overflow level = OFL = $B^{(U + 1)} * (1 - B^{(-P)})$ which has B - 1 as the value for each digit of the mantissa and the largest possible value for the exponent. Any number larger than OFL cannot be represented in the given floating-point system and no number smaller than the UFL can be represented in the floating point system.

The IEEE standard goes further than just requiring the use of a guard digit. It gives an algorithm for addition, subtraction, multiplication, division and square root, and requires

IV. DESIGN OF ARITHMETIC UNIT

Decimal arithmetic plays a key role in many commercial and financial applications, which process decimal values and perform decimal rounding. However, current software implementations are prohibitively slow [1], prompting hardware manufacturers such as IBM to add decimal floating point (DFP) arithmetic support to their microprocessors [2]. Furthermore, the IEEE has developed the newly IEEE 754-2008 [3] standard for Floating-Point Arithmetic adding the decimal representation to the IEEE 754-1985 standard. There are several works focused on fixed-point addition [4, 5]. For example, in [4, 5], Busaba, and Haller propose combined decimal and binary adders using pre-sum and pre-selection logic. Only a few previous recent papers focused on decimal floating-point addition [6-8]. The proposed adder by Cohen et al. [8] and Bohlender et al. [6] have long latencies and produces one result digit each cycle. In [7] presents the first IEEE 754 decimal floating point adder. Nevertheless, recently appears the first publications in decimal arithmetic applications on FPGA [9]. Several hardware designs were synthesized using other platforms than FPGA [10], this is why it is believed that it can be one of the first implementation on FPGA for addition/subtraction using decimal64 encoding. A recent work [9] presents a DFP adder for FPGA but using Binary Integer Decimal (BID) encoding, whose results will be used in our comparisons. Because of the BID adder occupies less area [9] than our proposed design, it may be argued that the BID format is well suited for hardware implementation but considering the latency-area tradeoff.

1) A .Floating Point Number System

Every real number can be approximated by a floating point number in the IEEE 754 standard [ANSI 85] as long as that number is within specific range. The floating point number format is based on scientific notation with limited size for each field. For a normalized floating point number in the IEEE 754 single precision standard where the integer part is always equals to 1, the sign bit is 1 bit in size. The integer part is omitted as it is always equals to 1. The size of fraction part is 23 bit and the size of exponent is 8 bit. The base is always equal to 2 and the total size of a single precision floating point number is 32 bits. In general, an IEEE 754 floating point number F can be expressed as follows: where s stands for the sign bit, f stands for the fraction and e stands for the biased exponent. In order to express a negative exponent, there is an exponent bias b associated with the exponent field. The actual exponent is the value of the exponent field minus the bias. The value of bias depends on the size of exponent e size as in equation 2.2. The term significand represents 1.f in which integer field and fraction field are packed together.

Flow Summary	
Flow Status	Successful - Thu Aug 23 16:21:23 2012
Quartus II Version	11.0 Build 208 07/03/2011 SP 1 SJ Web Edition
Revision Name	qa
Top-level Entity Name	Floating_top
Family	Cyclone III
Total logic elements	8,938 / 15,408 (58 %)
Total combinational functions	8,612 / 15,408 (56 %)
Dedicated logic registers	3,745 / 15,408 (24 %)
Total registers	3745
Total pins	43 / 169 (25 %)
Total virtual pins	0
Total memory bits	0 / 516,096 (0 %)
Embedded Multiplier 9-bit elements	28 / 112 (25 %)
Total PLLs	0 / 4 (0 %)
Device	EP3C16F256C6
Timing Models	Final

Fig 2.Area utilization report.

2) Arithmetic operations

The outline of the paper is as follows. In the next Section, it is described the background information on decimal floating-point. It describes the challenge of adding decimal64 encoded numbers, and presents the technique and theory for addition and subtraction. Section 4 presents synthesis results for our proposed adder and comparisons with the adder from [9]. Finally, Section 5 presents our conclusions.

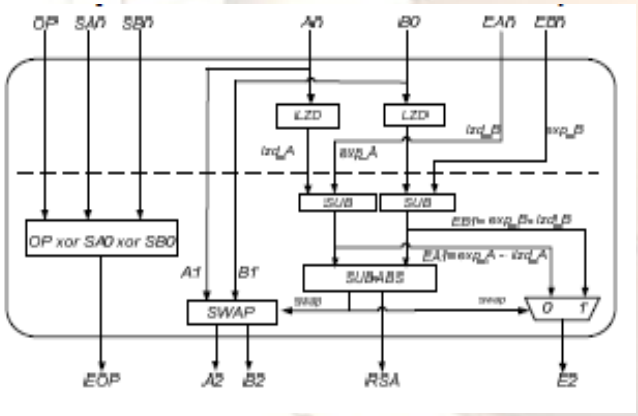

Fig 3.Architecture of FLU

In the rest of this paper, AX and BX are the significands and EAX, EBX andF EX are the exponents respectively. X is a digit that denotes the outputs of different units. The symbol (N)Z T“ refers to Tth bit of the Zth digit in a number N, where the least significant bit and the least significant digit have index 0.

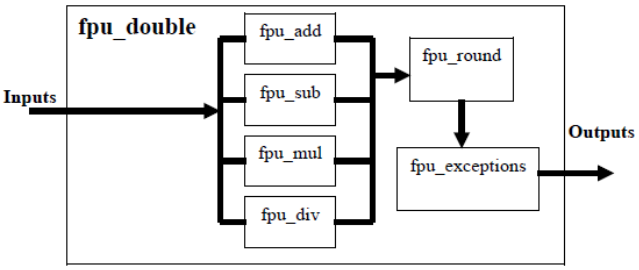

Fig 4.Basic modules of arithmetic operation

V.CONCLUSION

Several different representations of real numbers have been proposed, but by far the most widely used is the floating-point representation. Floating-point representations have a base b (which is always assumed to be even) and a precision p. Finally we implement all basic arithmetic operations involved in any DSP applications in Floating point .With the help of this FPU core we can do any kind of DSP applications. Here we achieve high speed of 130MHz by various pipelining stages.

REFERENCES:

[1] M. F. Cowlishaw, “Decimal floating-point: algorism for computers,” in *Proc. 16th IEEE Symp. Computer Arithmetic*, 2003, pp. 104–111.

[2] E. M. Schwarz, J. S. Kapernick, and M. F. Cowlishaw, “Decimal floating-point support on the IBM System z10 processor,” 2009, iBM Journal of Research and Development.

[3] “IEEE Standard for Floating-Point Arithmetic,” pp. 1–58, 2008, iEEE Std 754-2008.

[4] F. Y. Busaba, C. A. Krygowski, W. H. Li, E. M. Schwarz, and S. R. Carlough, “The IBM z900 decimal arithmetic unit,” in *Proc. Conf Signals, Systems and Computers Record of the Thirty-Fifth Asilomar Conf*, vol. 2, 2001, pp. 1335–1339.

[5] *Xilinx Inc. Virtex-5 Libraries Guide for VHDL design*, v12.1 ed., Xilinx Inc., June 2009. [Online]. Available: http://www.xilinx.com

[6] Xilinx Inc, *DS335: Floating-Point Operator v5.0*, June 2009.

[7] J. Thompson, N. Karra, and M. J. Schulte, “A 64-bit decimal floating-point adder,” in *Proc. IEEE Computer society Annual Symp. VLSI*, 2004, pp. 297–298.

[8] M. S. Cohen, T. E. Hull, and V. C. Hamacher, “CADAC: A Controlled-Precision Decimal Arithmetic Unit,” no. 4, pp. 370–377, 1983.

[9] A. Farmahini-Farahani, C. Tsen, and K. Compton, “FPGA implementation of a 64-Bit BID-based decimal floatingpoint

adder/subtractor," in *Proc. Int. Conf. Field-Programmable Technology FPT 2009*, 2009, pp. 518–521.

[10] C. Tsen, S. Gonzalez-Navarro, and M. Schulte, "Hardware design of a Binary Integer Decimal-based floating-point adder," pp. 288–295, 2007, computer Design, 2007. ICCD 2007. 25th International Conference on.

[11] C. Minchola and G. Sutter, "A FPGA IEEE-754-2008 Decimal64 Floating-Point Multiplier,in *Proc. Int. Conf. Reconfigurable Computing and FPGAs ReConFig '09*, 2009, pp. 59–64.

[12] L.-K. Wang and M. J. Schulte, "Decimal Floating-Point Adder and Multifunction Unit with Injection-Based Rounding,in *Proc. 18th IEEE Symp. Computer Arithmetic ARITH '07*, 2007, pp. 56–68.

[13] M. Vazquez, G. Sutter, G. Bioul, and J. P. Deschamps, "Decimal Adders/Subtractors in FPGA: Efficient 6-input LUT Implementations," in *Proc. Int. Conf. Reconfigurable Computing and FPGAs ReConFig '09*, 2009, pp. 42–47.

[14] *Xilinx Inc. XST User Guide 12.1*, v12.1 ed., Xilinx Inc., June 2009. [Online]. Available: http://www.xilinx.com

[15] *Xilinx Inc. Xilinx ISE Design Suite 12.1 Software Manuals*, v12.1 ed., Xilinx Inc., June 2009. [Online]. Available: http://www.xilinx.com