A Decimal Floating Point Arithmetic Unit for Embedded System Applications using VLSI Techniques

Rajeshwari Mathapati¹, Shrikant. K.Shirakol ²

¹Dept of E&CE, S.D.M. College of Engg. and Tech., Dharwad, Karnataka, India

Abstract: With the growing popularity of decimal computer arithmetic in scientific and commercial financial and internet based applications, hardware realization of decimal arithmetic algorithm is gaining more importance. Hardware decimal arithmetic unit serve as an integral part of some recently commercialized general purpose processors, where complex decimal arithmetic operations such as multiplication, have been realized by rather slow iterative hardware algorithms. The widely used IEEE standards, IEEE 754-1985 based binary floating point format and IEEE 754-2008 standard based decimal floating point (DFP) format uses high precision formats, which are generally useful for astronomical or scientific calculations. For simple applications, such a precision is not required and is an addition overhead. Therefore the another format named Immanuel Decimal Floating Point (IDFP) Format is presented and can be used for embedded system applications, which uses commonly used precision. IDFP can be represented either in binary or binary coded decimal (BCD). An arithmetic unit based on IDFP, BCD based number system is developed and is functionally simulated. The adder/subtractor based on BCD adder/subtractor and multiplier developed is a look up table (LUT) based BCD multiplier. The language used for programming is VHDL and is synthesized using Xilinx ISE 13.4 design suite. The results are obtained using Modelsim simulator.

Keywords—BCD, DFP, IDFP, LUT.

I. INTRODUCTION

Floating-point and fixed-point numbers are used to represent real numbers in digital systems. The advantage of floating-point over fixed-point representation is the wider range of values supported. There are different floating-point representations. One of the most use standards is the IEEE 754. This standard was issued in 1985 [2] and was updated in 2008 [3]. The IEEE Std 754-1985 defines the half (binary16), single (binary32) and double (binary64) formats. The Quadruple (binary128) is a new type, which was defined in IEEE Std 754-2008.

People predominantly perform math using decimal numbers. However, due to the speed and efficiency advantages of binary hardware, most computer systems support binary arithmetic rather than decimal arithmetic. Converting decimal numbers to binary and performing operations on them using binary hardware can lead to inaccuracies. For example, the binary floating-point (BFP) single-precision number closest to the decimal fraction 0.1 is actually equal to 0.100000001490116119384765625.

However, BFP suffers from the inability to accurately represent many fractional decimal numbers and to perform correct decimal rounding. The errors from using BFP for decimal data are unacceptable for many commercial and financial applications. DFP does not suffer from these problems, and it is specified in the IEEE 754-2008 Floating-point Standard (IEEE 754-2008). In certain applications, small errors like this can compound and lead to incorrect and unacceptable results [4]. One study by IBM estimates that binary representation and rounding errors of decimal numbers in a telephone billing application can accumulate to a yearly billing discrepancy of over $5 million dollars [5].

DFP number systems represent floating-point numbers using a radix (i.e., exponent base) of ten rather than two. DFP therefore exactly represents decimal numbers, provided those numbers fit in the format’s precision. DFP research has received renewed interest in recent years, and DFP solutions are now commercially available. However, DFP hardware support could be added with less area increase by sharing logic between DFP and BFP. We therefore investigate the area and performance of a combined DFP and BFP multiplier compared to separate DFP and BFP units. Multiplication is a
common operation in financial applications [6], such as in currency exchange, interest calculation, or stock and mutual fund valuation.

As DFP becomes more prevalent, there is an increasing need for accurate evaluation and understanding of both hardware and software DFP solutions. A benchmark suite that aids this evaluation would help to eliminate the gap in understanding of the tradeoffs between different DFP solutions and give insight into which operations could benefit most from hardware acceleration. Since hardware implementations are not widely available for all DFP formats, current analysis must use available software libraries [7].

Two prevailing DFP number encodings have emerged. One, originally proposed by Intel, encodes the significand of the DFP number as an unsigned binary integer, and is commonly called the binary integer decimal (BID) encoding. The other, originally proposed by IBM, encodes the significand as a compressed BCD integer, and is commonly called densely packed decimal (DPD). The IEEE 754-2008 Standard for Floating-point Arithmetic [8] specifies these DFP encodings and their operations, rounding, and exception handling. The standard allows DFP numbers to be encoded either in BID or DPD, and specifies arithmetic operations on 64-bit and 128-bit DFP numbers, known as decimal64 and decimal128, respectively.

The paper [9] adds to previous research by investigating the tradeoffs of sharing hardware for BFP and BID multiplication. The combined multiplier design we present is based on a BID multiplication algorithm presented in [10] and a basic BFP multiplication algorithm presented in [11]. This paper demonstrates that a hardware design for a combined BFP and BID multiplier is feasible and compelling.

II. STUDY OF IEEEEE FLOATING POINT


The standard defines

1) Arithmetic formats: sets of binary and decimal floating-point data, which consist of finite numbers (including signed zeros and subnormal numbers), infinities, and special "not a number" values (NaNs).

2) Interchange formats: encodings (bit strings) that may be used to exchange floating-point data in an efficient and compact form

3) Rounding rules: properties to be satisfied when rounding numbers during arithmetic and conversions

4) Operations: arithmetic and other operations on arithmetic formats

5) Exception handling: indications of exceptional conditions (such as division by zero, overflow, etc.)

A. Basic format

The standard provides for many closely related formats, differing in only a few details. Five of these formats are called basic formats and others are termed extended formats; three of these are especially widely used in computer hardware and languages:

1) Single precision: used to represent the "float" type in the C language family (though this is not guaranteed). This is a binary format that occupies 32 bits (4 bytes) and its significand has a precision of 24 bits (about 7 decimal digits).

2) Double precision: used to represent the "double" type in the C language family (though this is not guaranteed). This is a binary format that occupies 64 bits (8 bytes) and its significand has a precision of 53 bits (about 16 decimal digits).

3) Double extended: also called "extended precision" format. This is a binary format that occupies at least 79 bits (80 if the hidden/implicit bit rule is not used) and its significand has a precision of at least 64 bits (about 19 decimal digits).

Increasing the precision of the floating point representation generally reduces the amount of accumulated round-off error caused by intermediate calculations. [5]
Less common IEEE formats include:

1) Quadruple precision (binary128) is a binary format that occupies 128 bits (16 bytes) and its significand has a precision of 113 bits (about 34 decimal digits).

2) Double precision (decimal64) and quadruple precision (decimal128) decimal floating-point formats. These formats, along with the single precision (decimal32) format, are intended for performing decimal rounding correctly.

3) Half also called float16, a 16-bit floating-point value.

B. Rounding Rules

The standard defines five rounding rules. The first two rounds to a nearest value; the others are called directed roundings:

1) Rounding to nearest: Round to nearest, ties to even—rounds to the nearest value. Round to nearest, ties away from zero—rounds to the nearest value

2) Directed Roundings: Round toward 0—directed rounding towards zero (also known as truncation). Round toward +∞—directed rounding towards positive infinity (also known as rounding up or ceiling). Round toward −∞—directed rounding towards negative infinity (also known as rounding down or floor).

C. Operations

1) Required operations for a supported arithmetic format (including the basic formats) include:

2) Arithmetic operations (add, subtract, multiply, divide, square root, fused multiply–add, remainder, etc.)

3) Conversions (between formats, to and from strings)

4) Scaling and (for decimal) quantizing

5) Copying and manipulating the sign (abs, negate, etc.)

6) Comparisons and total ordering

7) Classification and testing for NaNs, etc.

8) Testing and setting flags

9) Miscellaneous operations

D. Exception handling

The five possible exceptions are:

1) Invalid operation (e.g., square root of a negative number) (returns qNaN by default).

2) Division by zero (an operation on finite operands gives an exact infinite result, e.g., 1/0 or log (0)) (returns ±infinity by default).

3) Overflow (a result is too large to be represented correctly) (returns ±infinity by default (for round-to-nearest mode)).

4) Underflow (a result is very small (outside the normal range) and is inexact) (returns a denormalized value by default).

5) Inexact (returns correctly rounded result by default).

III. OVERVIEW OF IDFP FORMAT

A modified decimal floating point format named as Immanuel decimal floating point, (I-DFP) [1], format is proposed here. In this modified decimal format a number can be represented either in binary format or in BCD format. The format of a floating point number represented using the I-DFP is as follows

<table>
<thead>
<tr>
<th>Meta Number</th>
<th>Number In Bcd/Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT8</td>
<td>± (0 to 2^8 - 1)</td>
</tr>
<tr>
<td>INT16</td>
<td>± (0 to 2^16 - 1)</td>
</tr>
<tr>
<td>INT32</td>
<td>± (0 to 2^32 - 1)</td>
</tr>
</tbody>
</table>

The meta number is an 8 bit number. The meta number indicates the sign (+/-) bit of the number, whether the number is represented in binary / bcd, data format type in use and the number of decimal places in the number. The number can be indicated in the general form as ± (0 to (2N – 1)) * 10^E, where the exponential E is in the range 0 to 14. Four data format types are defined after IDFP, they are:
An example of number representation in I – DFP, BCD format is as follows. If the number, -42.9496 7295 is represented in FLOAT_INT32 data type then, -42.9496 7295 can be rewritten as -42 9496 7295 x 10^-8, which when represented in I-DFP BCD format is, 1110 1000 0100 0010 1001 0100 1001 0110 0111 0010 1001 0101.

1) In the above example, first 8 bits will represent the Meta number.

2) In Meta number, first bit represent the sign bit (+/-), where ‘1’ holds for negative and ‘0’ for positive sign. Second and third bit represent the data format (binary/BCD), where ‘1’ holds for BCD and ‘0’ for binary format. Rest of five bits represents the number of decimal places in number. In the above example its ‘8’ (01000). Other bits of IDFP format represent the number either in binary or BCD format.

IV. IDFP ARITHMETIC UNIT

The IDFP arithmetic unit is formed by making use of BCD Adder/ Subtractor and a LUT based multiplier. In this model the Adder/Subtractor is used of 40-bits. Among the 40-bits 8-bits are used for the Meta number and rest all 32-bits for the BCD/Binary representation of the IDFP number. The multiplier used here for the multiplication of two IDFP numbers is LUT multiplier of 40-bits.

BCD or Binary Coded Decimal is that number system or code which has the binary numbers or digits to represent a decimal number. In case of BCD the binary number formed by four binary digits, will be the equivalent code for the given decimal digits and we can use the binary number from 0000-1001 only, which are the decimal equivalent from 0-9 respectively. It is possible to perform addition in BCD by first adding in binary, and then converting to BCD afterwards. Conversion of the simple sum of two digits can be done by adding 6 (that is, 16 – 10) when the five-bit result of adding a pair of digits has a value greater than 9 as shown in fig 1. This technique can be extended to adding multiple digits by adding in groups from right to left, propagating the second digit as a carry, always comparing the 5-bit result of each digit-pair sum to nine.

BCD Subtraction is done by adding the ten's complement of the subtrahend. To represent the sign of a number in BCD, the number 0000 is used to represent a positive number, and 1001 is used to represent a negative number. The remaining 14 combinations are invalid signs.

The LUT multiplier is used for multiplying two multi-bit binary operands to produce a binary result by means of a lookup table containing all possible products of said operands, reduction of the total amount of memory required to store the table is obtained by segmenting one operand into a plurality of non-overlapping bit groups and constructing lookup tables for the bit groups, in which each lookup table contains products of its associated bit group and the other, non-partitioned operand. Multiplication is accomplished by generating partial products from the lookup tables, shifting the partial products to account for the relative significance of their associated bit groups, and adding the partial products to provide the resultant product.

Look-Up Table multipliers are simply a block of memory containing a complete multiplication table of all possible input combinations. The large table sizes needed for even modest input widths make these impractical for FPGAs.

The following table is the contents for a 6 input LUT for a 3 bit by 3 bit multiplication table.
TABLE II: Truth Table of LUT Multiplier

<table>
<thead>
<tr>
<th></th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>001</td>
<td>000001</td>
<td>000010</td>
<td>000011</td>
<td>000100</td>
<td>000101</td>
<td>000110</td>
<td>001011</td>
<td>001111</td>
</tr>
<tr>
<td>010</td>
<td>001000</td>
<td>001010</td>
<td>001100</td>
<td>001110</td>
<td>010000</td>
<td>010001</td>
<td>010110</td>
<td>011110</td>
</tr>
<tr>
<td>011</td>
<td>010000</td>
<td>010010</td>
<td>010100</td>
<td>010110</td>
<td>011000</td>
<td>011001</td>
<td>011110</td>
<td>100110</td>
</tr>
<tr>
<td>100</td>
<td>100000</td>
<td>100010</td>
<td>100100</td>
<td>100110</td>
<td>101000</td>
<td>101001</td>
<td>101110</td>
<td>111110</td>
</tr>
<tr>
<td>101</td>
<td>101000</td>
<td>101010</td>
<td>101100</td>
<td>101110</td>
<td>110000</td>
<td>110001</td>
<td>110110</td>
<td>111110</td>
</tr>
<tr>
<td>110</td>
<td>110000</td>
<td>110010</td>
<td>110100</td>
<td>110110</td>
<td>111000</td>
<td>111001</td>
<td>111110</td>
<td>111111</td>
</tr>
<tr>
<td>111</td>
<td>111000</td>
<td>111010</td>
<td>111100</td>
<td>111110</td>
<td>111001</td>
<td>111010</td>
<td>111111</td>
<td>111111</td>
</tr>
</tbody>
</table>

The IDFP arithmetic unit (as shown in Fig. 2) consists of a multiplexer which is used to select one of the operations (addition/ subtraction/ multiplication). By making use of a select line of a Mux we perform the particular operation with two IDFP numbers. Whenever the select line is ‘0’ the BCD addition is performed. The data format is of 40 bit MSB 8 bits represents the Meta number and remaining 32 bits represents the data in BCD format. While performing the BCD addition the data and exponent part is added as per BCD addition. The first 4 bits of Meta number perform the XOR operation. The same procedure is followed for the BCD subtraction when the select line is ‘1’. Multiplication is done when the select line is ‘2’, the Meta number is added according to BCD addition and only the data (LSB 32 bits) is multiplied with LUT multiplier as shown in Fig 3.

V. CONCLUSION AND FUTURE SCOPE

A. Conclusion

This dissertation discusses the overall contribution of the project.

1) The IDFP floating point format is implemented.
2) The schematic and simulation result of one bit adder, four bit adder, BCD adder, inverter, 9’s complement, BCD Subtractor, multiplexer, LUT multiplier, top module of the project are designed and implemented.
3) The 1474 number of LUT’s is used with 5% of utilization. Number of bounded IOB’s used are 128 with 26% of utilization.
4) The IDFP arithmetic unit is designed for 32-bits. The addition and subtraction of IDFP numbers is accomplished using 32-bit BCD adder/subtractor. The multiplier is designed using 32-bit LUT multiplier.
5) The arithmetic unit is designed and implemented in Xilinx technology of version – 13.4 (Higher Version) using Xilinx ISE and Xilinx XST Tools.

B. Future Work

1) The design in this dissertation can be upgraded to give better performance.
2) Resolution can be increased up to 64 to 128 bits in order to increase the size of a unit.
3) We can design and implement IDFP divider for 32-bit format.
4) Using different multipliers we can design a better IDFP multiplier.
REFERENCES


