The Delay and Area Analysis of Different Single Precision Floating Point Arithmetic Operations

Navaneeth Prabhanandan
PG Student [VLSI Design], Dept. Of ECE
Amrita Vishwa Vidyapeetham University, Kerala, India
navaneethp89@gmail.com

Senthil Murugan
Assistant Professor, Dept. Of ECE
Amrita Vishwa Vidyapeetham University, Kerala, India
senthilmurugan@am.amrita.edu

Abstract— In this paper the performance of different single precision floating point arithmetic operations has been evaluated. The comparison have been made between Brent Kung, Kogge Stone and Ladner Fischer Decimal Floating point Arithmetic Operations, focusing on minimizing delay and area. In addition the proposed design is compliant with the IEEE 754-2008 floating point format. The Verilog Codes have been simulated in Modelsim and synthesized in Xilinx ISE software. The Simulations of proposed design show that the Decimal Floating point Arithmetic operation using kogge stone gives much faster response compared to Brent Kung and Ladner Fischer

Keywords—Decimal floating point; arithmetic; IEEE 754-2008 ; verilog ; synthesis ; xilinx ; single precision

I. INTRODUCTION

The IEEE 754-2008 is the revised version of IEEE 754-1985. This standard specifies formats and methods for floating -point arithmetic in computer system- standard and extended functions with single, double, extended and extendable precisions-- and recommends formats for data interchange.

Financial and business applications use decimal based arithmetic to perform arithmetic operations. These applications require accuracy. The IBM z900 introduced hardware for decimal arithmetic.IBM in general, have been making quite strong claims for the advantages of hardware support for decimal arithmetic: Initial benchmarks indicate that some applications spend 50% to 90% of their time in decimal processing, because software decimal arithmetic suffers a 100 to 1000 performance penalty over hardware. The need for decimal floating-point adder in hardware is urgent [7].

Besides the accuracy and the speed up factors, saving power is very important. A research paper estimates that power savings for the whole application due to the use of a dedicated hardware instead of a software layer are of the same order of magnitude as the time savings. It also indicates that the process normalized Energy Delay Product (EDP) metric, clearly shows that a hardware implementation for DFP units gives two to three orders of magnitude improvement in EDP as a conservative estimate if compared with software implementations.[13]

The decimal arithmetic seems to take the same road map of binary. After the domination of binary ALUs in processors, a common trend now is to include either separated Decimal (including DFP) ALUs besides their binary equivalents or to use combined binary and decimal ALUs.[14][15]

II. IEEE 754-2008 REPRESENTATION OF DECIMAL FLOATING POINT NUMBER [2]

There are many parameters that define a Floating-Point representation system. This resulted in a variety of Floating-Point processors with different representations, producing different results to the execution of the same program. In some cases, because of anomalies, the results might be very different. To avoid this the IEEE Floating -point Standard 754 was developed.

Representation and Formats

The representation of the floating-point number x consists of two components, the significand M_x^* (also called mantissa) and the exponent E_x , such that

$$x = M_x^* \times b^{Ex}$$

where b is a constant called the base. The sign of the number is determined by the sign of the significand. The exponent is a signed integer.

The signed significand can be represented using any representation system, such as sign-and-magnitude or two's complement. Today the most used representation is sign-and-magnitude. In such case, a floating-point number x is represented by a triple (S_x , M_x , E_x), that is

$$x = (-1)^S M_x^* \times b^{Ex}$$

The two parts of the representation are as follows :

First, the significand is in sign and magnitude representation. Consequently it is represented by two components:

- Sign S. One bit. S=1 if negative.
- Magnitude (also called significand).

Represented in radix 2 with one integer bit. That is , the normalized significant is represented by

1. F

Where F of f bits (depending on the format) is called the fraction and the most-significand 1 is the

hidden bit. The range of the(normalized) significand is

$$1 \leq 1.F \leq 2 - 2^{-f}$$

Second, the exponent is base 2 and in biased representation. The number of bits of the exponent field is e, depending on the format .The representation is biased with bias $B = 2^{e-1} - 1$.

The three components are packed into one word, in which the order of the field is S, E, F. This order makes comparison simpler.

The value zero, denormals, and the special values NAN and infinites are represented as follows:

- The representation of floating point zero is E=0 and F=0 .The sign S differentiates between positive and negative zero. Because of this representation and the hidden bit, the value $1.0 * 2^{-B}$ is not represented.
- The representation E=0 and F≠ 0 is used for denormals ; in this case the floating -point value is represented as $v=(-1)^S 2^{-(B-1)}(0.F)$
- The maximum exponent representation

(E=2^e -1=2B +1) is used to represent not a number (NAN) for F≠ 0and plus and minus infinity for F=0

The system has two formats: basic and extended. The basic format allows representation in single and double precision. In each case we give the three components with the number of bits in parentheses. We call v the value represented.

TABLE I: IEEE 754 single precision format [1]

S	8 bit Exponent- E	23 bit Fraction -F

1. Basic single (32 bits) and double (64 bits)
   - Single : S (1) ,E(8) , F(23)
   a) If 1 < E < 254 ,then $v=(-1)^S 2^{-(B-1)}(1.F)$ (normalized fp number).
   b) If E =255 and F≠ 0,then NAN (not a number).
   c) If E =255 and F= 0, then $v=(-1)^S \infty$ (plus and minus infinity).
   d) If E =0 and F≠0, then $v==(-1)^S 2^{-(126}(0.F)$ (denormal, gradual underflow).
   e) If E=0 and F=0, then $v==(-1)^S 0$ (positive and negative zero)
   - Double : S(1),E(11),F(52)
   - similar representation to single, replacing 255 by 2047 ,and so on.
2. Extended : single (at least 43 bits = S(1),E(11),F(31)) and double (at least 79 bits = S(1),E(15),F63)).

TABLE II: Single and Double precision format summary [1]

Format	Precision (*p*)	E_{max}	E_{min}	Exponent bias	Exponent width	Format width
Single	24	+127	-126	127	8	32
Double	53	+1023	-1022	1023	11	64

Rounding

Rounding modes are:

- Default : Rounding to nearest, to even when tie
- Directed : Round towards plus infinity; Round towards minus infinity; and Round toward 0 (truncate)

Operations

Operations include:

- Numerical : Add , Sub ,Mult
- Conversions : Floating to integer ; Binary to decimal (integer) ; Binary to decimal (floating)
- Miscellaneous : Change formats; Compare and set condition code

Excepetions

The IEEE standard defines the following five exceptions. By default these exceptions set a flag and the computation continues. The implementation can include a trap handler for each exception that, when enabled, is called when exception occurs.

- Overflow (when rounded value is too large to be represented). Result is set to ± infinity.
- Underflow (when rounded value is too small to be represented).
- Division by zero.
- In exact result (result is not an exact floating point number), Infinite precision result different from floating - point t number.
- Invalid .This flag is set when NAN result is produced.

TABLE III: Exceptions for Operations in DFP [1]

Sign	Exponent	Fraction	Value	Description
S	0×FF	0×00000000	$(-1)^S$	Infinity
S	0×FF	F ≠ 0	NaN	Not a Number
S	0×00	0×00000000	0	Zero
S	0×00	F ≠ 0	$(-1)^s \times 2^{(E-126)} \times (0.F)$	Denormalized Number
S	0×00< E< 0×FF	F	$(-1)^s \times 2^{(E-127)} \times (1.F)$	Normalized Number

III. FLOATING-POINT ADDITION

In this Project addition/subtractions are the more complicated operations. Basically the addition is based on the carry look ahead adder. There are four important steps to do the floating -point addition.

1. Adjust and Alignment

Here the comparison is occurring. The exponent bits of two operands are compared to find the highest operand. Let *a_original* be the 8 bits exponent of the first operand and *b_original* be the second operand, comparing these two operands and assigning the highest one as the *a_original,* first operand. After this, breaking the original operand to exponent and fraction. Then inserting the hidden bits to the fractional part .Now aligning the exponent part of the operands and making the sizes of fractional part same by finding the difference between the exponent parts. Here we are taking the *aexp* as the exponent part of the first operand and *bexp* is the exponent part of the second operand. *afraction* is the fractional part of the first operand and *bfraction* is the fractional part of the second operand.

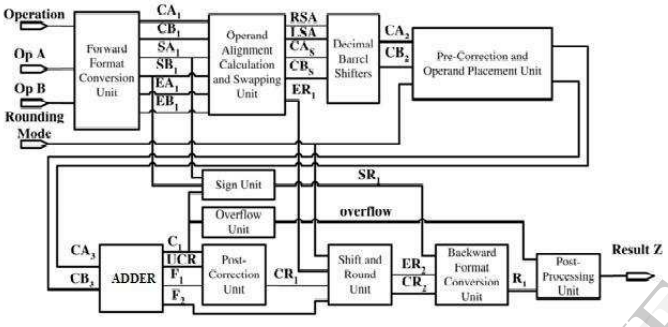

Fig I: Proposed Decimal Floating -Point Adder [6]

2. Add fractions

Binary addition is most primitive and commonly used applications in computer arithmetic. A large variety of adder circuits are available, such as Ripple carry adder, Carry look ahead adder. The speed of a carry look ahead adder improves by reducing the amount of time required to determine carry bits. A Ripple Carry Adder is a very area-efficient adder design. Unfortunately it is also slow. So for obtaining higher operating speed we are using Parallel prefix Adder tree structures such as kogge stone, Ladner Fischer and Brent Kung adders. The basic digital blocks of all the three adders are the same.

A. Carry Look Ahead Adder.

Given two Single precision operands A and B.

Pre-processing: This step involves computation of Propagate and Generate bits corresponding to each pair of bits in A and B. Define the functions P_i (Propagate) and G_i (Generate) at each bit position

$$P_i = A_i \text{ xor } B_i$$
$$G_i = A_i \text{ and } B_i$$

Carry Look Ahead network : This block differentiate the adder circuits. The long distance propagation of carries embody this intermediate stage. This will differentiate the adder and its performance. There are lots of ways to develop these intermediate stages.

$$P_{i:j} = P_{i:k+1} \text{ and } P_{k:j}$$
$$G_{i:j} = G_{i:k+1} \text{ or } (P_{i:k+1} \text{ and } G_{k:;j})$$

Post- processing: This is the final step and is common to all adders of this family.(carry look ahead) It involves computation of sum bits. Sum bits are computed by the logic given below.

$$S_i = P_i \text{ xor } C_i$$

B. Different Adder Circuits.

BC Block: The black cell takes two pairs of generate and propagate signals (gi, pi) and (gj, pj) as input and compute a pair of generate and propagate signals as output. The expressions for the output signals g,p generated by the black cell are given by

$$g = g_i + p_i * g_j$$
$$p = p_i * p_j$$

GC Block: The grey cell takes two pairs of generate and propagate signals (g_i, pi) and (g_j, pj) as input and computes a generate signal g as output which is shown below.

$$g = g_i + p_i * g_j$$

Buffer: The Buffer takes a pair of the generate and propagate signals as input and passes the same signals to the output .It is shown below.

$$g = g_i$$
$$p = p_i$$

a) Brent – Kung Adder

The Brent – Kung tree computes prefixes for 2-bit groups. These are used to find prefixes for 4 –bit groups, which in turn are used to find prefixes for 8 – bit groups and so forth. The prefixes then fan back down to compute the carries-in to each bit. The tree requires $2\log 2^N - 1$ stages. The fan-out is limited to 2 at each stage .The basic blocks used in this case are grey and black cells. [9]

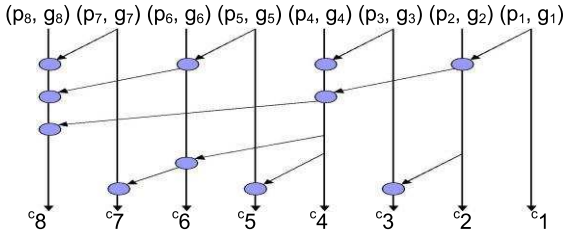

Fig II: Logic design of a Brent Kung adder [11]

b) Kogge –Stone Adder

The Kogge stone adder is built from generate and propagate blocks. It calculates carries corresponding to every bit with the help of black and grey cells. Here the logic level is given by $\log_2 N$.The fan-out is limited to 2. [9]

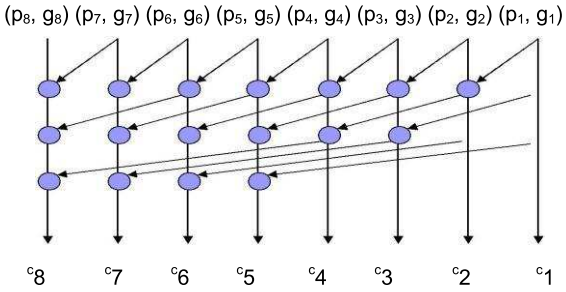

Fig III: Logic design of a Kogge Stone Adder [11]

c) Ladner – Fischer Adder

It is a parallel prefix adder. Ladner-Fischer has minimum logic depth but it has large fan-out required up to n/2.Ladner-Fischer adder has carry operator nodes. [11].

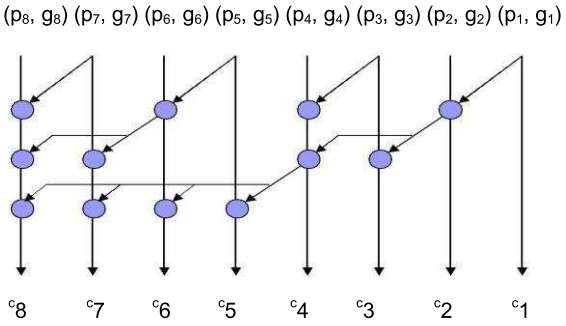

Fig IV: Logic design of a Ladner Fischer Adder [11]

3. Post normalize

The processing of this step is based on the carry bit, which obtained from the addition of fractional bits.

4. Check for exponent overflow

In this step checking exponent bits whether it is exceeding the maximum limit by comparing it with the maximum size.

RESULT AND CONCLUSION

Three Different Algorithms used in IEEE 754-2008, where compared with respect to their delay and frequency. All the Program simulated in Modelsim and synthesized using Xilinx ISE tool. The synthesis results for the Single precision Adders are obtained for the Virtex 5 FPGA. Here a clock of 20ns is used. The input and output signals, clock are assumed to be ideal. The design is optimized for the delay. From the synthesized result Single precision Decimal Floating point Adder using Kogge Stone algorithm is much faster than others. The below table compare the delay.

TABLE IV: FPGA Synthesized result comparison

DFP unit with Different Adder	Brent Kung	Kogge stone	Ladner Fischer
Delay(ns)	6.283	1.498	1.890
Frequency(MHZ)	159.160	667.557	529.101

ACKNOWLEDGEMENT

This work was supported by the Amrita School of Engineering. The author is very grateful to those persons who showed their helpful mind for this project.

REFERENCES

[1] IEEE 754-2008,IEEE Standard for Floating-Point Arithmetic,2008

[2] Milos D. Ercegovac and Tomas Lang,Text Book " Digital Arithmetic ", pp 414-417.

[3] Purna Ramesh Addanki,Venakata Nagaratna Tilak Alapati and Mallikarjuna Prasad Avana,"An FPGA Based High Speed IEEE-754 Double Precision Floating Point Adder/Subtractor and Nultiplier Using Verilog", in International Journal of Advance Science and Technology ,vol 52 (2013).

[4] Severance, Charles (20 Feb 1998)."An Interview with the Old Man of Floating-Point" (http://www.eecs.berkeley.edu/~wkahan/ieee754status/754story.html).

[5] General Decimal Arithmetic (http://speleotrove.com/decimal/)

[6] Liang-Kai Wang, MJ Schulte, JD Thompson, and N Jairam. "Harware Designs for Decimal Floating-Point Addition and Related Operations". IEEE Transactions on Computer. 2009; 58:pp 322-335.

[7] Cowlishaw, M.F." Decimal floating-point: algorism for computers", Computer Arithmetic, 2003. Proceedings. 16th IEEE Symposium on jun 2003,pp 104 – 111.

[8] EM Schwarz, JS Kapernick, and MF Cowlishaw. "Decimal floating-point suppoert on the IBM z10 processor". IBM Journal of Research and Development.2009;53: pp 231-239.

[9] Deepa Yagain, Vijaya Krishna A and Akansha Baliga, " Design of High-Speed Adders for Efficient Digital Design Blocks", Vol 2012.

[10] R. Ladner and M. Fischer,"Parallel prefix computation", Journal of ACM.La.Jolla CA,Vol.27,pp 831-838,ct 1980

[11] Harware Algorithm for Arithmetic Modules (www.aoki.ecei.tohoku.ac.jp/arith/mg/algorithm.html)

[12] P. Chaitanya kumari and R. Nagendra ," Design of 32 bit Parallel Prefix Adders", in IOSR Journal of Electronics and Communication Engineering,Vol 6,Issue 1,pp 01-06.

[13] H. Fahmy, R. Raafat, A. Abdel-Majeed, R. Samy, T. ElDeeb and Y. Farouk ,"Energy and Delay Improvement via Decimal Floating Point Units", Computer Arithmetic, 2009. ARITH 2009. 19th IEEE Symposium on jun 2009.pp 221-224.

[14] C. Webb, "IBM z10: The next-generation mainframe microprocessor," Micro, IEEE,Vol. 28, Mar-Apr 2008,pp 19-29

[15] E.Schwarz and S.Carlough,"Power6 decimal divide," in Application-specific Systems,Architectures and Processors,2007.ASAP.IEEE International Conference on Jul 2007.pp 128-133