High Speed IEEE 754 Floating Point Multiplier using Different Types of Adder

Anekant Jain* and Mr. Manish Saxena**
* M. Tech. Scholar, Department of ECE, BIST Bhopal, India
anekantjain18@gmail.com
** Asst. Professor, Department of ECE, BIST Bhopal, India
manish.saxena2008@gmail.com

Abstract: Due to advancement of new technology in the field of VLSI and Embedded system, there is an increasing demand of high speed and low power consumption processor. Speed of processor greatly depends on its multiplier as well as adder performance. In spite of complexity involved in floating point arithmetic, its implementation is increasing day by day. Due to which high speed adder architecture become important. Several adder architecture designs have been developed to increase the efficiency of the adder. In this paper, we introduce an architecture that performs high speed IEEE 754 floating point multiplier using carry select adder (CSA). Here we are introduced two carry select based design. These designs are implementation Xilinx Vertex device family.

Keywords: IEEE754, Single Precision Floating Point (SP FP), Double Precision Floating Point (DP FP), Binary to Execess-1 (), PAD1.

Introduction

The real numbers represented in binary format are known as floating point numbers. Based on IEEE-754 standard, floating point formats are classified into binary and decimal interchange formats. Floating point multipliers are very important in dsp applications. This paper focuses on double precision normalized binary interchange format. Figure 1 shows the IEEE-754 double precision binary format representation. Sign (s) is represented with one bit, exponent (e) and fraction (m or mantissa) are represented with eleven and fifty two bits respectively. For a number is said to be a normalized number, it must consist of'one' in the MSB of the significand and exponent is greater than zero and smaller than 1023. The real number is represented by equations (i) & (2).

$$Z = (-1^{s}) \times 2^{(E-Bias)} \times (1.M) \qquad (1)$$

$$Value = (-1^{signbit}) \times 2^{(Exponent-1023)} \times (1.Mantissa) \qquad (2)$$

Biasing makes the values of exponents within an unsigned range suitable for high speed comparison.
In computing, floating point describes a system for representing real numbers which supports a wide range of values. Numbers are in general represented approximately to a fixed number of significant digits and scaled using an exponent. The base for the scaling is normally 2, 10 or 16. The typical number that can be represented exactly is of the form:

$$\text{Significant digits} \times \text{base}^{\text{exponent}}$$

The term floating point refers to the fact that the radix point (decimal point, or, more commonly in computers, binary point) can "float"; that is, it can be placed anywhere relative to the significant digits of the number. This position is indicated separately in the internal representation, and floating-point representation can thus be thought of as a computer realization of scientific notation. Over the years, several different floating-point representations have been used in computers; however, for the last ten years the most commonly encountered representation is that defined by the IEEE 754 Standard.
The advantage of floating-point representation over fixed- point (and integer) representation is that it can support a much wider range of values. For example, a fixed-point representation that has seven decimal digits with two decimal places, can represent the numbers 12345.67, 123.45, 1.23 and so on, whereas a floating-point representation (such as the IEEE 754 decimal32 format) with seven decimal digits could in addition represent 1.234567, 123456.7,

0.00001234567,
1234567000000000, and so on. The floating-point format needs slightly more storage (to encode the position of the radix point), so when stored in the same space, floating-point numbers achieve their greater range at the expense of precision. The speed of floating-point operations is an important measure of performance for computers in many application domains. It is measured in FLOPS.

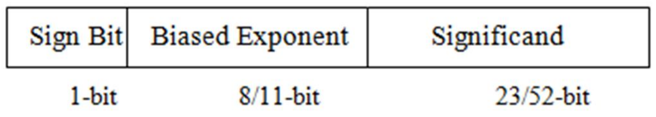


Figure 1: IEEE 754 Single Precision and Double Precision Floating Point Format

IEEE 754 STANDARD FLOATING POINT MULTIPICATION ALGORITHM
A brief overview of floating point multiplication has been explained below [5-6].

- Both sign bits S_1, S_2 are need to be Xoring together, then the result will be sign bit of the final product.
- Both the exponent bits E_1, E_2 are added together, then subtract bias value from it. So, we get exponent field of the final product.
- Significand bits Sig1 and Sig2 of both the operands are multiply including their hidden bits.
- Normalize the product found in step 3 and change the exponent accordingly. After normalization, the leading "1 "will become the hidden bit.

Above algorithm of multiplication algorithm is shown in Figure 2.

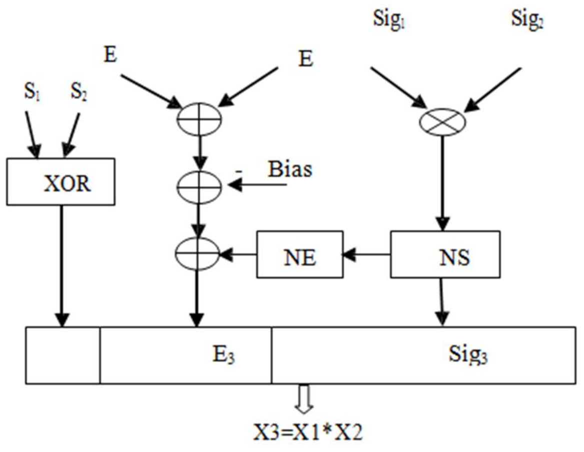


Figure 2: IEEE754 SP FP and DP FP Multiplier Structure, NE: Normalized exponent, NS: Normalized Significand

IEEE-754 standard is s standard representation established by IEEE and widely used standard for floating point computation. Single precession floating point represents computer format and it occupies 32-bits in a computer memory and represents a wide range of values by using a floating point. In IEEE 754-
2008, the 32-bit with base 2 formats [2][6] is referred as single precision or binary 32.
The standard basically has four types and they are

- Arithmetic format: it is a set of binary and decimal floating point numbers which has finite numbers that contains signed zero, subnormal and infinite numbers and special value called" not a number"(NaN).
- Interchange format: it is a bit string or encodings that are mainly used to exchange floating point data in a compact and efficient form.
- Rounding rules: properties should be satisfied while doing arithmetic operations and conversion of any numbers on arithmetic formats.
- Exception handling: It indicates any exceptional conditions like overflow, underflow etc., while doing operations.

Some other computer representations for non-integral numbers
Floating-point representation, in particular the standard IEEE format, is by far the most common way of representing an approximation to real numbers in computers because it is efficiently handled in most large computer processors. However, there are alternatives:

- Fixed-point representation uses integer hardware operations controlled by a software implementation of a specific convention about the location of the binary or decimal point, for example, 6 bits or digits from the right. The hardware to manipulate these representations is less costly than floating- point and is also commonly used to perform integer operations. Binary fixed point is usually used in special- purpose applications on embedded processors that can only do integer arithmetic, but decimal fixed point is common in commercial applications.
- Binary-coded decimal is an encoding for decimal numbers in which each digit is represented by its own binary sequence.
- Logarithmic number systems represent a real number by the logarithm of its absolute value and a sign bit. The value distribution is similar to floating-point, but the value-to- representation curve, i. e. the graph of the logarithm function, is smooth (except at 0). Contrary to floating-point arithmetic, in a logarithmic number system multiplication, division and exponentiation are easy to implement but addition and subtraction are hard.
- Where greater precision is desired, floating-point arithmetic can be implemented (typically in software) with variable- length significands (and sometimes exponents) that are sized depending on actual need and depending on how the calculation proceeds. This is called arbitrary-precision arithmetic.
- Some numbers (e.g., 1/3 and 0.1) cannot be represented exactly in binary floating-point no matter what the precision. Software packages that perform rational arithmetic represent numbers as fractions with integral numerator and denominator, and can therefore represent any rational number exactly. Such packages generally need to use "bignum" arithmetic for the individual integers.
- Computer algebra systems such as Mathematica and Maxima can often handle irrational numbers like or $3^{1/2}$ in a completely "formal" way, without dealing with a specific encoding of the significand. Such programs can evaluate expressions like " " exactly, because they "know" the underlying mathematics.

Ieee-754 Floating Point Format

IEEE is a technically used standard followed by many hardware and software implementation. Single precision floating point standard represents 32bits (4bytes) in computer number format, most significant bit will start from left this single precision is having three basic components sign, exponent and mantissa sign bit width is 1 bit, exponent of width is 8bits and mantissa of 24 bits out of which 23 bits are explicitly stored [6] and 1 bit is implicitly stored, sign bit is used represent sign of floating point number where sign(s=o positive numbers=1 negative number)

The Number representation of single precession [9]. Value = (-1) s *2E-127 * 1.M (normalized) when E>0 S= sign bit (0 for positive, 1 for negative) e=unbiased exponent; e=E-127(bias) E=exponent it can be signed or unsigned integer, 8-bit signed integer it ranges fromEi = - 128 to Eax =127(2'compliment) and for unsigned it ranges from ei =0 to Eax =255 which is the accepted biased form in IEEE 754 single precession. In this exponent with value 127 represents actual zero. The true mantissa will have 23 fraction bits to the right of binary point and implicit to left of binary point with value 1unless exponent bit is stored with zeros. In memory we can see only 23 fraction bits in mantissa even though it has 24bit.If E>0 and E.

Representable numbers, conversion and rounding

By their nature, all numbers expressed in floating-point format are rational numbers with a terminating expansion in the relevant base (for example, a terminating decimal expansion in base-10, or a terminating binary expansion in base-2).

Irrational numbers, such as π or $\sqrt{2}$, or non-terminating rational numbers, must be approximated. The number of digits (or bits) of precision also limits the set of rational numbers that can be represented exactly. For example, the number 123456789 clearly cannot be exactly represented if only eight decimal digits of precision are available. When a number is represented in some format (such as a character string) which is not a native floating-point representation supported in a computer implementation, then it will require a conversion before it can be used in that implementation. If the number can be represented exactly in the floating-point format then the conversion is exact. If there is not an exact representation then the conversion requires a choice of which floating-point number to use to represent the original value. The representation chosen will have a different value to the original, and the value thus adjusted is called the rounded value.

Whether or not a rational number has a terminating expansion depends on the base. For example, in base-10 the number 1/2 has a terminating expansion (0.5) while the number 1/3 does not (0.333...). In base-2 only rationals with denominators that are powers of 2 (such as 1/2 or 3/16) are terminating. Any rational with a denominator that has a prime factor other than 2 will have an infinite binary expansion. This means that numbers which appear to be short and exact when written in decimal format may need to be approximated when converted to binary floating-point. For example, the decimal number 0.1 is not representable in binary floating-point of any finite precision; the exact binary representation would have a "1100" sequence continuing endlessly:

e = −4; s = 1100110011001100110011001100110011..., where, as previously, s is the significand and e is the

exponent.
When rounded to 24 bits this becomes e = −4; s = 110011001100110011001101,
which is actually 0.100000001490116119384765625 in decimal.
As a further example, the real number π, represented in binary as an infinite series of bits is

11.0010010000111111011010101000100010000101101000110000100011010011... but is 11.0010010000111111011011 when approximated by rounding to a precision of 24 bits. In binary single-precision floating-point, this is represented as s = 1.10010010000111111011011 with e = 1.
This has a decimal value of 3.1415927410125732421875, whereas a more accurate approximation of the true value of π
Is 3.1415926535897932384626433832795...

The result of rounding differs from the true value by about 0.03 parts per million, and matches the decimal representation of π in the first 7 digits. The difference is the discretization error and is limited by the machine epsilon.
The arithmetical difference between two consecutive representable floating-point numbers which have the same exponent is called a unit in the last place (ULP). For example, if there is no representable number lying between the representable numbers 1.45a70c22hex and 1.45a70c24hex, the ULP is 2×16−8, or 2−31. For numbers with an exponent of 0, a ULP is exactly 2−23 or about 10−7 in single precision, and about 10−16 in double precision. The mandated behavior of IEEE-compliant hardware is that the result be within one-half of a ULP.

Literature Review

Soumya Havaldar et al. [1], gives an FPGA Based High Speed IEEE-754 Double Precision Floating Point Multiplier Using Verilog. This paper has implemented DPFP Multiplier using parallel Adder. A high speed floating point double precision multiplier is implemented on a Virtex-6 FPGA. In addition, the proposed design is compliant with IEEE-754 format and handles over flow, under flow, rounding and various exception conditions. The design achieved the operating frequency of 414.714 MHz with an area of 648 slices.

Ragini Parte et al. [2], IEEE point number-crunching has an immense application in DSP, advanced PCs, robots because of its capacity to speak to little numbers and huge numbers and in addition marked numbers and unsigned numbers. Disregarding unpredictability included in gliding point number juggling, its usage is expanding step by step. Here we break down the impacts of utilizing three unique sorts of adders while figuring the single accuracy and twofold exactness skimming point increase. We additionally exhibit the increase of significand bits by disintegration of operands strategy for IEEE 754 standard.

Ross Thompson et al. [3], IEEE-754 determines trade and number juggling positions also, routines for paired and decimal drifting point number juggling in PC programming world. The execution of a skimming point framework utilizing this standard should possible completely in programming, or in equipment, or in any blend of programming and equipment. This venture propose VHDL execution of IEEE - 754 Floating point unit .In proposed work the pack, unload and adjusting mode was actualized utilizing the VHDL dialect and reenactment was checked.
In this proposition work, DPFP Multiplier alongside SPFP Multiplier has been actualized with four ordinary Adders (PA, CSKA, CSA, and CSABEC). Think about their Results and CSA is known not the speediest snake among every single customary viper. Be that as it may, CSA involves more territory as it has two parallel circuits to include the same bits yet with diverse convey data. As CSA figures the whole without sitting tight for the transitional conveys to spread stage by stage. Finally it is the obligation of multiplexer to pick and give the last right yield. CSABEC is an adjusted adaptation of CSA in which one of the parallel circuits is supplanted by the arrangement of Binary to Excess-1 Converters circuit (BECs). It is turned out to be an awesome way to deal with decrease the territory.

Different Types of Adder

Parallel Adder

Parallel adder can add all bits in parallel manner i.e. simultaneously hence increased the addition speed. In this adder multiple full adders are used to add the two corresponding bits of two binary numbers and carry bit of the previous adder. It produces sum bits and carry bit for the next stage adder. In this adder multiple carry produced by multiple adders are rippled, i.e. carry bit produced from an adder works as one of the input for the adder in its succeeding stage. Hence sometimes it is also known as Ripple Carry Adder (RCA). Generalized diagram of parallel adder is shown in figure 3.

Table 1: Summary of Literature Review

Entitle of paper	Approached used	Software	Parameter	Published Year
Design of Vedic IEEE 754 Floating Point Multiplier	IEEE 754 floating point multiplier using Vedic Multiplier	Xilinx 12.1 i	Slice = 305, LUT= 598	IEEE 2016
Analysis of Effects of using Exponent Adders in IEEE- 754 Multiplier by VHDL	IEEE 754 floating point multiplier using Carry select adder	Verilog using the Model SIM SE 6.3	Slice = 353, LUT= 610	IEEE 2015
An IEEE 754 Double-Precision Floating-Point Multiplier for Denormalized and Normalized Floating-Point Numbers	IEEE 754 floating point multiplier using floating number	Xilinx Spartan 3E xa3s250c-4vqg100	Slice = 453, LUT= 678	IEEE 2015

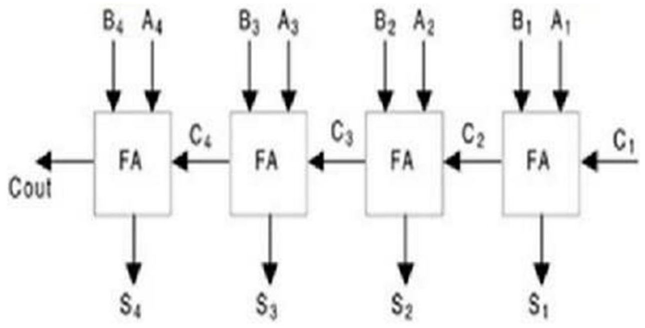


Figure 3: Parallel Adder (n=7 for SPFP and n=10 for DPFP)

An n-bit parallel adder has one half adder and n-1full adders if the last carry bit required. But in 754 multiplier’s exponent adder, last carry out does not required so we can use XOR Gate instead of using the last full adder. It not only reduces the area occupied by the circuit but also reduces the delay involved in calculation. For SPFP and DPFP multiplier’s exponent adder, here we Simulate 8 bit and 11 bit parallel adders respectively as show in figure 4.

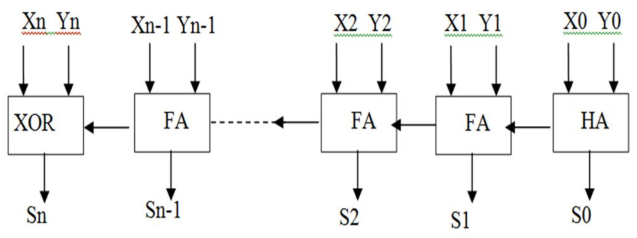


Figure 4: Modified Parallel Adder (n=7 for SPFP and n=10 for DPFP)

Carry Skip Adder

This adder gives the advantage of less delay over Ripple carry adder. It uses the logic of carry skip, i.e. any desired carry can skip any number of adder stages. Here carry skip logic circuitry uses two gates namely "and gate" and "or gate". Due to this fact that carry need not to ripple through each stage. It gives improved delay parameter. It is also known as Carry bypass adder. Generalized figure of Carry Skip Adder is shown in figure 5.

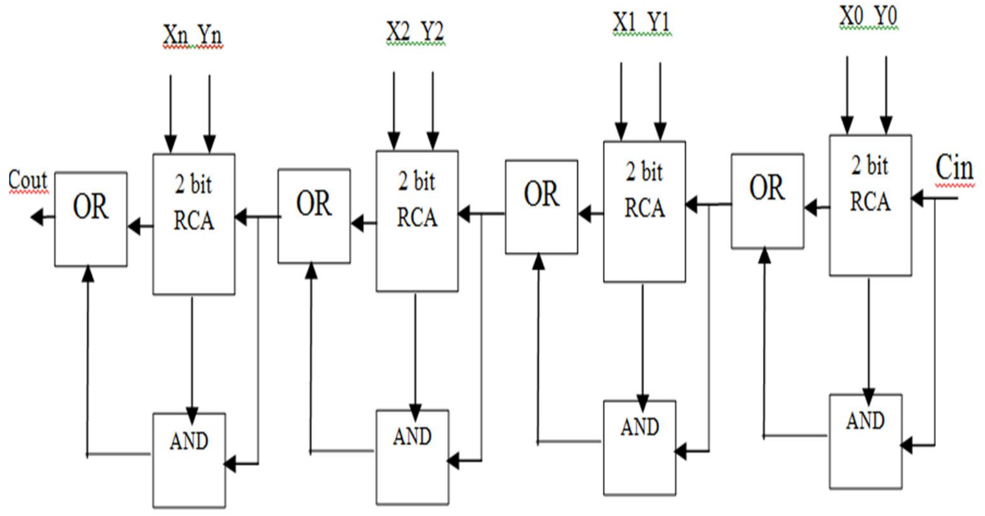


Figure 5: Carry Skip Adder

Carry Select Adder

Carry select adder uses multiplexer along with RCAs in which the carry is used as a select input to choose the correct output sum bits as well as carry bit. Due to this, it is called Carry select adder. In this adder two RCAs are used to calculate the sum bits simultaneously for the same bits assuming two different carry inputs i.e. '1' and '0'. It is the responsibility of multiplexer to choose correct output bits out of the two, once the correct carry input is known to it. Multiplexer delay is included in this adder. Generalized figure of Carry select adder is shown in figure 3.9. Adders are the basic building blocks of most of the ALUs (Arithmetic logic units) used in Digital signal processing and various other applications**.** Many types of adders are available in today's scenario and many more are developing day by day. Half adder and Full adder are the two basic types of adders. Almost all other adders are made with the different arrangements of these two basic adders only. Half adder is used to add two bits and produce sum and carry bits whereas full adder can add three bits simultaneously and produces sum and carry bits.

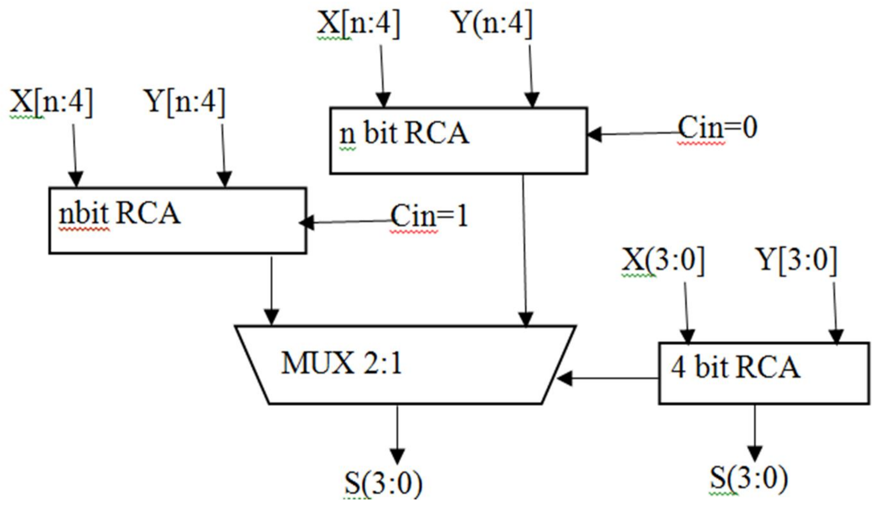
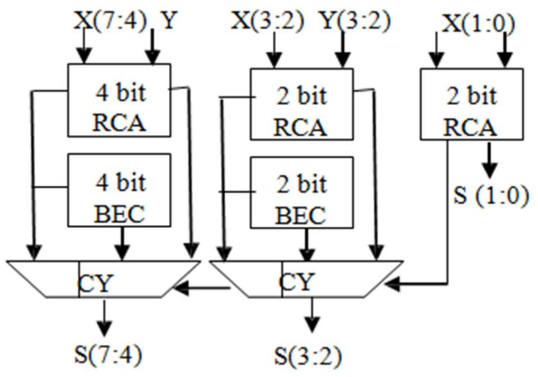


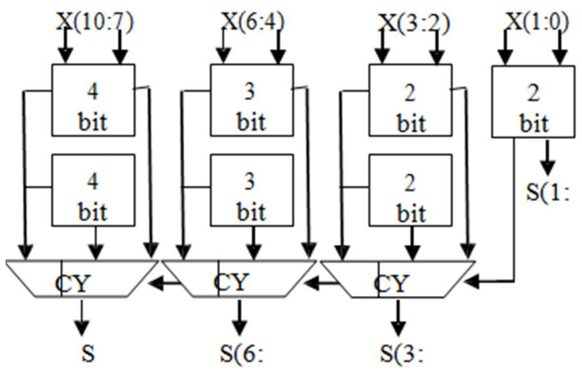
Figure 6: Carry Select Adder

Carry Select Adder with Binary to Excess-1Converter (CSABEC)

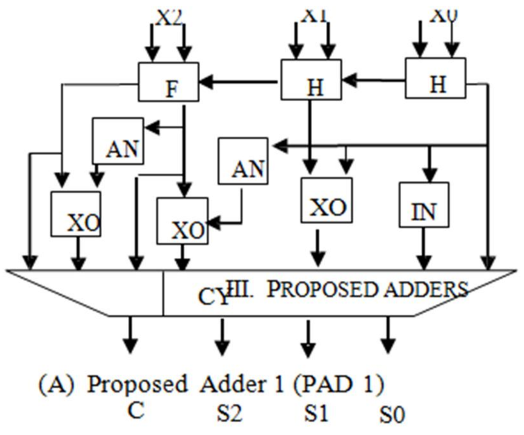
The basic design of CSA has two parallel and same type of circuits of RCA (Ripple Carry Adder) which add the same bits but with different carry inputs. In this paper we have introduced a modified version of CSA with the objective to reduce area occupied by the circuit and delay involved in calculation. In this circuit, one of the parallel circuit of CSA is replaced be Binary to Excess-1 Converter (BEC). We have implemented these adders in exponent addition of IEEE 754 floating point multiplication. 8 bit, 11 bit and 3 bit CSABEC adders has been shown in figure 5 [a], [b] and [c] respectively.



(a) 8 bit CSABEC

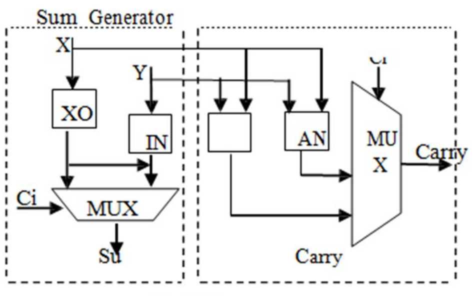


(b) 11 bit CSABEC



(c) 3 bit CSABEC

Figure 5 : Carry Select Adder Binary to Excess-1 Conveter



(a) CBL

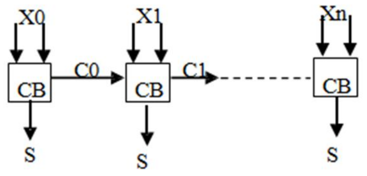
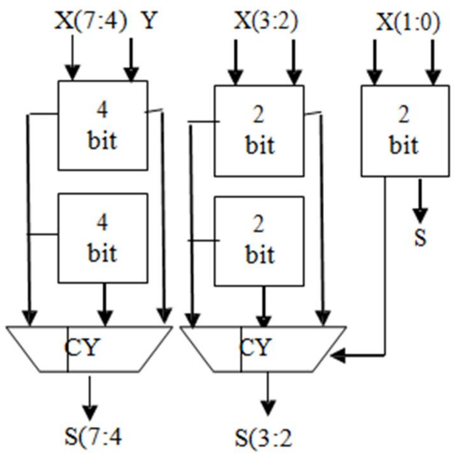


FIGURE 7: (b)n bit PAD 1 (n=7 and n=10 for



(a) 8 bit MCBL

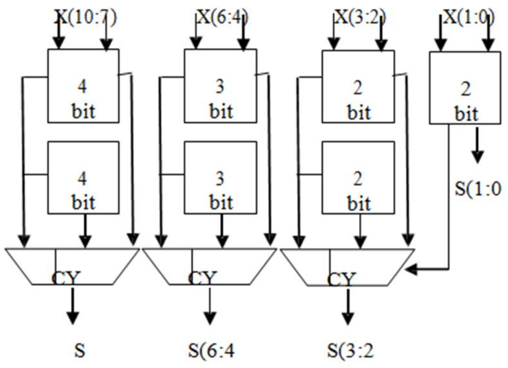


FIGURE 6 : Carry Select Adder with Common Boolean

Manju-et al explains 16-bit Carry Select Adder with Common Common Boolean Logic (CBL), here we have proposed a new adder and call it as Modified Common Boolean Logic Adder (PAD 1). In this adder as shown in figure, we have basic building block of a XOR Gate, a AND Gate, a OR Gate, an inverter and two 2:1 multiplexers. For every two bit addition, required so we need 8 such blocks. We can understand a single block in two parts, a sum bit generator part and a carry bit generator part as shown in figure 6(a). As the carry out of last bit is not required, we can remove the carry generator part of the last block used. It not only reduces the gate count but also reduces the delay involved in calculation. Likewise in DPFP Multiplier 11 blocks are applied, last block only with sum bit generator. This adder recognized as PAD 1 in this paper [5]. Detailed structure of CBL has been shown in figure 7 [a]

Proposed Adder 2 (PAD 2)

Kogge-Stone Adder has named after Peter M. Kogge and Harold S. Stone who has proposed this adder. It is also known by the name parallel prefix adder. This adder provides carry signal in very less time [6]. We have proposed an efficient design of Kogge-stone adder which shows very significant reduction in area occupied and delay involved in summation. Eventually, proposed KSA provides very efficient method to calculate exponent bits in IEEE 754 multiplication. We have introduced as Proposed Adder 2 (PAD2), which uses the logic of Kogge-stone adder. We have designed this adder by using cascaded arrangement of half adders and XOR Gates as shown in figure [8]. We have simulated a 8-bit KSA for IEEE 754 SPFP multiplication and a 11-bit KSA for IEEE 754 DPFP multiplication.

For exponent addition in IEEE 754 multiplication, we have checked the results of applying conventional adders, CSA and its modified versions (CSABEC) .Also we have checked the performances of PAD 1 and PAD 2 which has been proven to be the better options among all. Multiplication of Mantissa bits has been done with the splitting off the operands method because of the large size of the Mantissa bits.

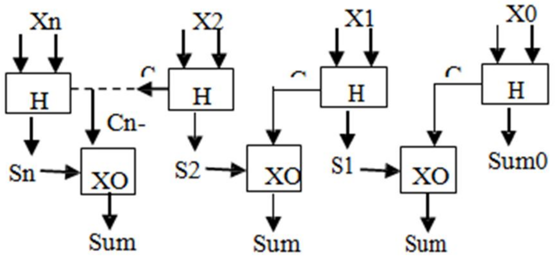


FIGURE 8: (b) PAD1 (n=7 for SPFP and n=10 for DPFP)

Significand Bit Calculation

As per the format defined by IEEE 754, significand bits of SPFP and DPFP are 23 bits and 52 bits which are the normalized bits and excludes the initial ‘1’ known as the hidden bit. While calculating the resultant significand bits of the multiplication of two SPFP or DPFP numbers, significand bits needs to multiplied but considering the hidden one along with the significand bits. So now the significand bits of SPFP and DPFP becomes 24 bits and 53 bits respectively. This large size of Mantissa bits has made the multiplication procedure more complex. Here we use the technique of operand decomposition to reduce the complexity and ensure the accuracy. 24 and 53 bits are need to be break down into groups. Each group of the first operand is multiplied with each one group of the other operand. Let the first operand is grouped into n groups and second operand is grouped into m groups then eventually we get nm multiplication terms which are the partial product terms of the final multiplication of the significant bits. These partial product terms have to added together very carefully by shift and add method. Shifting has to done according to which group of the first operand is multiplied by which group of the second operand. Final multiplication resultant bits are of size 48 and 106-bits of SPFP and DPFP respectively. These bits need to be normalized first so that we get the least most ‘1’ which will become the hidden bit and not appear in the result shown in the IEEE 754 floating point formats. Such a long size of the multiplication result makes it unsuitable to store in IEEE 754 floating point format, so now these bits are truncated and round off to 23 bits and 52 bits.

Proposed Design

In IEEE754 standard floating point representation, 8 bit Exponent field in single precision floating point (SP FP) representation and 11 bit in double precision floating point (DP FP) representation are need to add with another 8 bit exponent and 11 bit exponent respectively, in order to multiply floating point numbers represented in IEEE 754 standard as explained earlier. Ragini et al. [10] has used parallel adder for adding exponent bits in floating point multiplication algorithm. We proposed the use of 8-bit modified CSA with dual RCA and 8-bit modified CSA with RCA and BEC for adding the exponent bits. We have found the improved area of 8-bit modified Carry select adder with RCA and BEC over the 8-bit modified CSA with dual RCA.

Sign bit calculation

To calculate the sign bit of the resultant product for SP FP and DP FP multiplier, the same strategy will work. We just need to XOR together the sign bits of both the operands. If the resultant bit is ‘1’, then the final product will be a negative number. If the resultant bit is ‘0’, then the final product will be a positive number.

Exponent bit calculation

Add the exponent bits of both the operands together, and then the bias value (127 for SPFP and 1023 for DPFP) is subtracted from the result of addition. This result may not be the exponent bits of the final product. After the significand multiplication, normalization has to be done for it. According to the normalized value, exponents need to be adjusted. The adjusted exponent will be the exponent bits of the final product.

Significand bit calculation

Significand bits including the one hidden bit are need to be multiply, but the problem is the length of the operands. Number of bits of the operand will become 24 bits in case of SP FP representation and it will be 53 bits in case of DP FP representation, which will result the 48 bits and 106 bits product value respectively. In this paper we use the technique of break up the operands into different groups then multiply them. We get many product terms, add them together carefully by shifting them according to which part of one operand is multiplied by which part of the other operand. We have decomposed the significand bits of both the operands ain four groups. Multiply each group of one operand by each group of second operand. We get 16 product terms. Then we add all of them together very carefully by shifting the term to the left according to which groups of the operands are involved in the product term.

Simulation Result

All the designing and experiment regarding algorithm that we have mentioned in this paper is being developed on Xilinx 6.2i updated version. Xilinx 6.2i has couple of the striking features such as low memory requirement, fast debugging, and low cost. The latest release of ISETM (Integrated Software Environment) design tool provides the low memory requirement approximate 27 percentage low. ISE 6.2i that provides advanced tools like smart compile technology with better usage of their computing hardware provides faster timing closure and higher quality of results for a better time to designing solution.

These designs were compared with IEEE-754 floating point multiplier architecture proposed by Ragini et al. [2] to show for the improvements obtained.

So Ragini et al. [2] architecture is best in all these architectures. Implementing the Ragini et al. [2], proposed architecture IEEE-754 floating point design has been captured by VHDL and the functionality is verified by RTL and gate level simulation. To estimate the number of slice, number of 4-i/p LUTs and maximum combinational path delay (MCPD). By using this technique of break up the operands into different groups then multiply them I reduce all parameter compare to my base paper.

Conclusion

IEEE754 standardize two basic formats for representing floating point numbers namely, single precision floating point and double precision floating point. Floating point arithmetic has vast applications in many areas like robotics and DSP. Delay provided and area required by hardware are the two key factors which are need to be consider Here we present single precision floating point multiplier by using two different adders namely modified CSA with dual RCA and modified CSA with RCA and BEC.

Among all two adders, modified CSA with RCA and BEC is the least amount of Maximum combinational path delay (MCDP). Also, it takes least number of slices i.e. occupy least area among all two adders.

Refrences

[1] Soumya Havaldar, K S Gurumurthy, “Design of Vedic IEEE 754 Floating Point Multiplier”, IEEE International Conference On Recent Trends In Electronics Information Communication Technology, May 20-21, 2016, India.

[2] Ragini Parte and Jitendra Jain, “Analysis of Effects of using Exponent Adders in IEEE- 754 Multiplier by VHDL”, 2015 International Conference on Circuit, Power and Computing Technologies [ICCPCT] 978-1-4799-7074- 2/15/$31.00 ©2015 IEEE.

[3] Ross Thompson and James E. Stine, “An IEEE 754 Double-Precision Floating-Point Multiplier for Denormalized and Normalized Floating-Point Numbers”, International conference on IEEE 2015.

[4] B. Fagin and C. Renard, "Field Programmable Gate Arrays and Floating Point Arithmetic," IEEE Transactions on VLS1, vol. 2, no. 3, pp. 365-367, 1994.

[5] N. Shirazi, A. Walters, and P. Athanas, "Quantitative Analysis of Floating Point Arithmetic on FPGA Based Custom Computing Machines," Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM"95), pp.155-162, 1995.

[6] Malik and S. -B. Ko, “A Study on the Floating-Point Adder in FPGAs”, in Canadian Conference on Electrical and Computer Engineering (CCECE-06), (2006) May, pp. 86–89.

[7] D. Sangwan and M. K. Yadav, “Design and Implementation of Adder/Subtractor and Multiplication Units for Floating-Point Arithmetic”, in International Journal of Electronics Engineering, (2010), pp. 197-203.

[8] L. Louca, T. A. Cook and W. H. Johnson, “Implementation of IEEE Single Precision Floating Point Addition and Multiplication on FPGAs”, Proceedings of 83rd IEEE Symposium on FPGAs for Custom Computing Machines (FCCM‟96), (1996), pp. 107–116.

[9] [9] Jaenicke and W. Luk, "Parameterized Floating-Point Arithmetic on FPGAs", Proc. of IEEE ICASSP, vol. 2, (2001), pp. 897-900.

[10] Lee and N. Burgess, “Parameterisable Floating-point Operations on FPGA”, Conference Record of the Thirty- Sixth Asilomar Conference on Signals, Systems, and Computers, (2002).

[11] M. Al-Ashrafy, A. Salem, W. Anis, “An Efficient Implementation of Floating Point Multiplier”, Saudi International Electronics, Communications and Photonics Conference (SIECPC), (2011) April 24-26, pp. 1-5.