Design of an ASIC-Based High Speed 32-bit Floating Point Adder

Debarshi Deka
Dept. of EEE
BITS Pilani Goa
Goa, India
f20170494@goa.bits-pilani.ac.in

Navaneeth Kumar
Dept. of EEE
BITS Pilani Goa
Goa, India
f20170977@goa.bits-pilani.ac.in

Dipankar Pal, SM-IEEE
Dept. of EEE
BITS Pilani Goa
Goa, India
dipankarp@goa.bits-pilani.ac.in

***Abstract*— Advancements in machine-learning algorithms made it necessary to explore fast algorithms for Floating Point operations, addition being most commonly used complex operation involving significant delay and power-consumption. Applications include high-performance computer vision, imaging and deep-learning functions accelerated using dedicated hardware accelerators. This paper proposes a 32-bit Floating Point Adder based on the 'Far-and-Close-Data-Path-Algorithm' with added optimizations to give a better implementation in terms of overall minimum latency and improved accuracy for certain input cases. The designs have been coded in Verilog, synthesized in Cadence Genus and physically verified with Cadence Innovus in GDSII under ASIC platform.**

Keywords—ASIC, Floating Point Adder, Far and Close Data Path Algorithm, Cadence Genus and Innovus, Kogge-Stone Adder, Barrel Shifter, Leading One Predictor (LOP), Compound Adder, Normalized, Denormalized and Mixed numbers

I. INTRODUCTION

Floating Point Adders form the basic block of many application-specific as well as general-purpose processors. Hence, performance of such systems largely depends on processing speed, area and power dissipation of the Floating Point adders. State-of-the-art Floating Point adders use the 'Far and Close Data Path Addition' as the basic algorithm and improvements have been made to its various components like rounding, normalization etc. over the years. This paper discusses the improvements over the 'Standard Floating Point Addition' (SA) [1,2], the baseline algorithm for floating-point addition to more advanced 'Far and Close Data Path Addition' (FCA) [3,4] and proposes an architecture that aims at improving the existing FCA with further parallelism and new functionalities.

To this end, a high-speed Floating Point Adder is implemented which makes two major improvements to the FCA. The first improvement exploits the concept of parallelism within the Floating Point adder to increase the overall speed of the design. The second improvement increases the accuracy of computation of 'mixed' type of inputs which is discussed in detail in subsequent Sections. Other optimizations such as improvement in shifter design, mantissa addition, rounding logic have also been implemented. It makes use of the FCA architecture to achieve high performance in case of 'normalized' type of operation. Since, the ASIC results of SA and FCA are not available they have been implemented on the ASIC platform according to the theory. The proposed design is validated and benchmarked against the mentioned SA and FCA designs giving the best results in terms of delay.

Section II gives the Floating Point representation technique. Section III gives the standard addition algorithm while Section IV presents the proposed addition-algorithm for Floating Point numbers. Section V discusses the results and gives a comparison with candidate designs. Lastly Section VI concludes the paper.

II. FLOATING POINT REPRESENTATION

The Institute of Electrical and Electronics Engineering (IEEE) issued standards for binary Floating Point arithmetic in 1985 [5]. Among these standards, Single-precision format uses 1-bit for sign bit (S), 8-bits for exponent (E) and 23-bits to represent the fraction/mantissa (F) as shown in Figure 1.

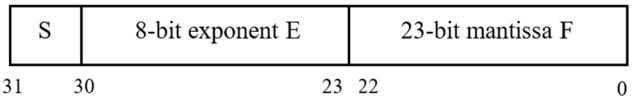

Fig.1. IEEE 754 Single Precision Format

The encoding of a single-precision Floating Point number to its decimal value is summarized in Table I.

TABLE I. IEEE 754 SINGLE PRECISION ENCODING

Sign (S)	Exponent (E)	Fraction (F)	Value	Description
S	0xFF	0x00000000	$(-1)^S \times \infty$	Infinity
S	0xFF	F≠0	NaN	Not a Number
S	0x00	0x00000000	0	Zero
S	0x00	F≠0	$(-1)^S \times 0.f \times 2^{-126}$	Denormalized
S	0x00<E<0xFF	F	$(-1)^S \times 1.f \times 2^{E-127}$	Normalized

For normalized and denormalized numbers, the first bit just left to decimal is the implicit bit and is not encoded in the Floating Point representation. It is set to '1' or '0' respectively during operations.

The rounding mode used in all the implementations mentioned in this paper is 'Round to nearest Even'. [5] In this mode, if the two nearest representable values are equally near, the one with its least significant bit equal to zero is chosen.

III. STANDARD FLOATING POINT ADDITION ALGORITHM

Following steps (reference, Figure 2) gives the basic steps followed in standard Floating Point addition algorithm. [1]

- Initially, both the numbers are decomposed to get their respective sign, exponent, and mantissa. The extra implicit bit is also appended.

2021 International Conference on Applied Electronics (AE) | 978-80-261-0973-0/21/$31.00 ©2021 10.23919/AE51540.2021.9542881

ISBN 978-80-261-0973-0, © University of West Bohemia, 2021

- The exponents are then compared and the difference between the two is obtained. This difference is used to adjust the mantissa of the lesser number to obtain the same exponent value for both the numbers. The adjusted mantissa is added or subtracted based on the sign bit.
- The result obtained is normalized if necessary and rounding is performed accordingly.
- This result is combined with the larger exponent to get the final output.

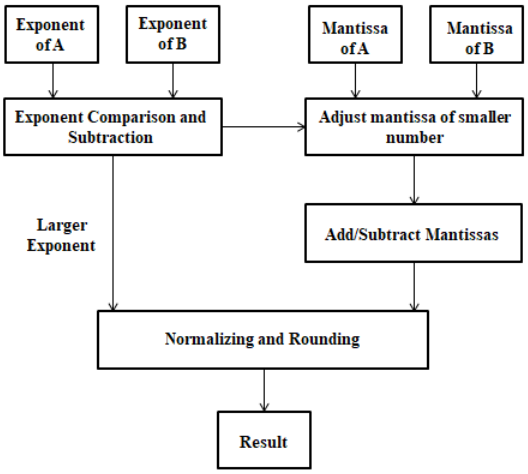

Fig. 2. Standard Floating Point Algorithm

IV. Proposed Floating Point Addition

Proposed design can be visualized from an algorithm point of view as shown below in Figure 3.

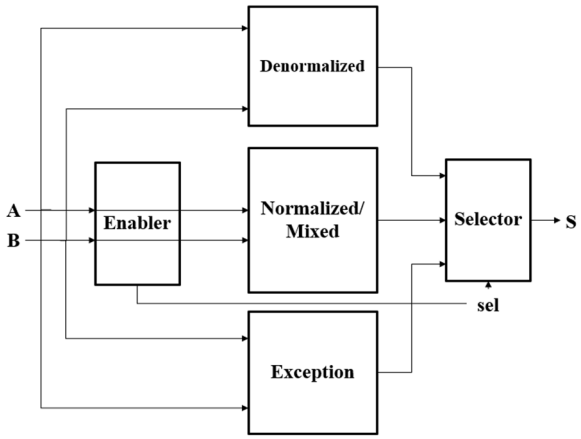

Fig.3. Basic Layout and Flow Diagram

This layout mainly takes advantage of the concept of parallelism and improves the accuracy of computation of certain input cases. The algorithm, implementation of each block, optimizations performed, analysis, comparison and final results are discussed in the following sections. The main flow of the system is as follows [6].

- 'Enabler' block takes the two inputs and identifies the type of operation i.e. input case.
- The output is calculated assuming the inputs to be 'Normalized', 'Denormalized', 'Mixed' and 'Exception' and the necessary computations are carried out in the individual blocks.
- Based on the type of operation identified, the selector block outputs the correct case calculated in the previous step.

For this design, we have categorized all input cases into the following types of operation:

- Normalized: Both inputs are normalized numbers
- Denormalized: Both inputs are denormalized numbers
- Mixed: One input is normalized and the other is denormalized
- Exception: Handling exceptional cases as defined by IEEE 32-bit single precision Floating Point format.

A. Enabler

This block identifies the input case as described earlier and based on this, generates the select line 'sel' and the control signals 'outa' and 'outb' to indicate the type of Floating Point input.

B. Exception

This block outputs the results directly for exceptional cases (Table II). This path avoids the extra delays incurred due to utilization of entire hardware for simple computations (for example, addition of a number with zero)

TABLE II. IMPLEMENTED EXCEPTIONAL CASES

Input 1	Input 2	Result
zero	number	number
infinity	number	infinity
infinity	infinity	Infinity (if same sign)
infinity	infinity	NaN (if different sign)
NaN	NaN	NaN

C. Denormalized

This block computes the sum assuming denormalized case. Here, mantissas of the two numbers are added directly without any shifting since the exponents are zero. A 24-bit Kogge-Stone adder is implemented to carry out this addition. The LSB of the exponent is also appended as MSB of the input, to account for overflow of output into normalized number. [7]

D. Normalized /Mixed

This block computes the sum of the Floating Point inputs assuming Normalized/Mixed operation. This is the most complex block that has been divided two major stages which are discussed below:

1) Preparation: Following figure shows the 'Preparation' stage which is connected to the 'Computation' stage.

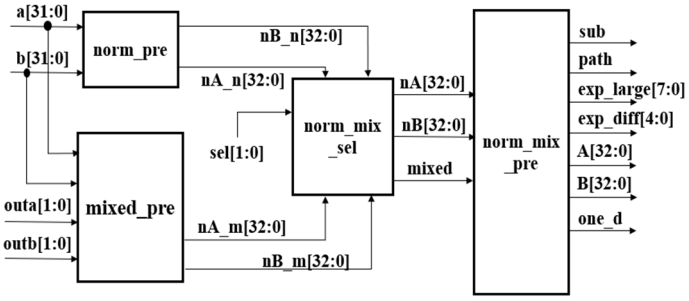

Fig.4. Preparation Stage

- **'norm_pre'** adds the implicit bit '1' at the end of exponent and beginning of mantissa resulting in the output of two 33-bit numbers.
- **'mixed_pre'** prepares the inputs for the 'mixed' case. Initially, normalized and denormalized numbers are identified based on the control signals generated in 'Enabler' and the implicit bit is added i.e. '1' for normal and '0' for denormal number. The denormal number is partially normalized by left shifting (done using Barrel Shifter) the mantissa such that the MSB is '1'. The amount left shifted (counted using an LOD i.e. Leading One Detector [8]) is set as the exponent, but mathematically it will be considered as a negative bias. However, it violates the IEEE format as negative biased exponents don't exist as seen in Table I. This step is essential for addition in mixed case to achieve higher accuracy [6]. Its significance at a hardware level is illustrated in the 'Results' section. The outputs of this block are the normalized and the partially normalized number (originally denormal).
- **norm_mix_sel** selects the required 'prepared inputs' and the 'mixed' flag is set to '1' in case of mixed inputs, else it is '0' for normal inputs. This mixed flag is used to indicate a negative biased exponent in further calculations.
- **norm_mix_pre** performs final preparation of inputs. Determines effective operation, identify far or close path for computation stage, resulting exponent difference, larger exponent and larger number to determine the output sign bit.

2) Computation: This stage is based on the 'Far-and-Close-Datapath-Algorithm' for Floating Point Addition. It calculates the addition output based on the 'prepared inputs' generated in 'Preparation' stage and the path identified. It has been briefly discussed as follows.

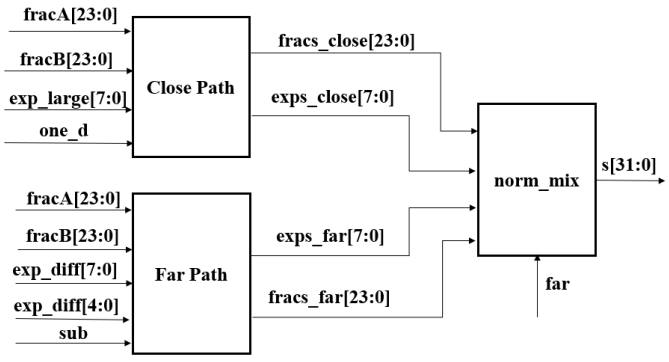

Fig.5. Computation Stage

- **Close Path:** Close Path output is taken if exponent difference is '0' or '1' and effective operation is subtraction. Following optimizations/ modifications to existing 'Close Path' architecture have been carried out: Determination of 'leading one' using an LOP (Leading One Predictor [9]) to reduce delay, implementation of the Compound Adder, Shifters designed using Barrel Shifter [10] design, rounding logic incorporated into Compound Adder to remove redundant calculations.
- **Far Path:** Far Path output is taken if exponent difference is more than one. Optimizations to the Compound Adder and shifters are same as that of Close Path

Addition in Far path and Close Path is carried out by 24 -bit Kogge-Stone adder which has 5 stages. Only black cells are used in this adder so that 'Group Propagate' and 'Group Generate' are obtained in parallel [11]. These results are used to compute 'Sum' and 'Sum+1' as follows:

$$S_i = P_i \oplus G_{i-1:0}$$
$$S1_i = S_i \oplus P_{i-1:0}$$

E. Selector

Based on the select line generated from 'Enabler' block, it chooses the desired output based on the type of input case.

V. Results

The adder modules are implemented using Verilog HDL, synthesized using Cadence Genus and physically verified using Cadence Innovus in GDSII under ASIC platform. The proposed design is compared with 'Standard Floating Point Adder (SA)' and 'Far and Close Path Floating Point Adder (FCA)'. Table III shows the synthesis results obtained in Genus. Table IV provides the Physical Verification (Post Layout-Routing Optimization) Innovus result of the proposed design.

TABLE III. Cadence Genus Synthesis Results

Design	Timing (ps)	Power		
		Leakage (nW)	Dynamic Power (nW)	Total Power (nW)
Proposed	6347	110.539	145097.841	145208.380
FCA	7596	64.630	113154.262	113218.892
SA	8892	38.984	90261.969	90300.953
Design	**Area**			
	Cells Count	**Total Area Sq. nm.**	**Logic (%)**	**Inverter (%)**
Proposed	2190	3639.156	93.9	6.1
FCA	1248	2306.584	95.2	4.8
SA	818	1446.796	95.6	4.4

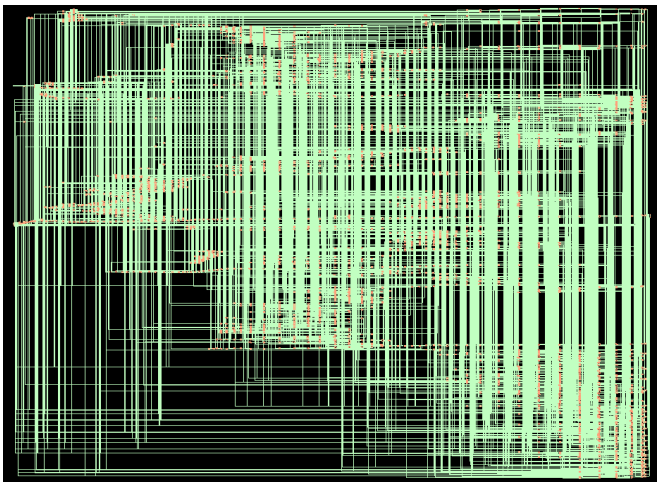

Fig.6. Schematic of Proposed Design (Courtesy Genus)

TABLE IV. POST IMPLEMENTATION RESULTS

Internal Power (mW)	0.1746
Switching Power (mW)	0.2623
Total Power (mW)	0.4371
Leakage Power (mW)	0.0001345
Capacitance (F)	6.01E-012

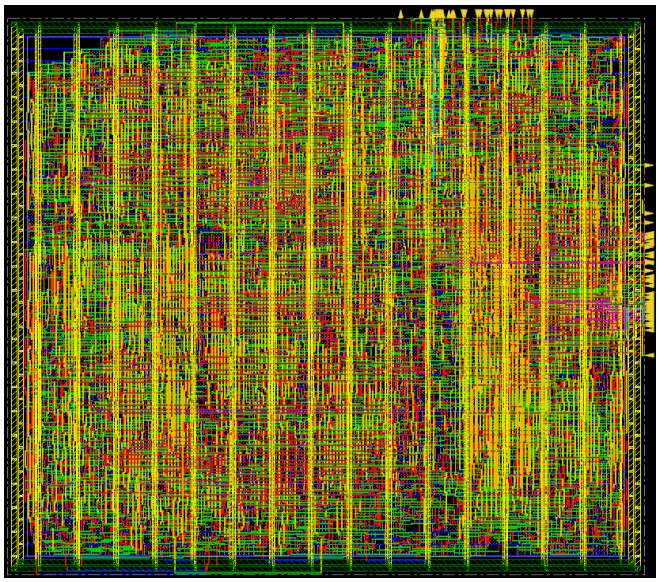

Fig.7. Physical Verification of Proposed Layout (Courtesy Innovus)

The simulation results are calculated for each input case using 'ISim' Simulator and displayed below where 'a' and 'b' are the inputs and 's' is the output. It should be noted in Fig. 8 that for the 3rd case (Mixed), the result 's' would have been input 'b' itself, i.e., 02081cea without the 'Mixed' logic. Normally, these special cases are taken care of by software, but the proposed design can take care of such special cases at the hardware level itself, improving accuracy by introducing new functionality.

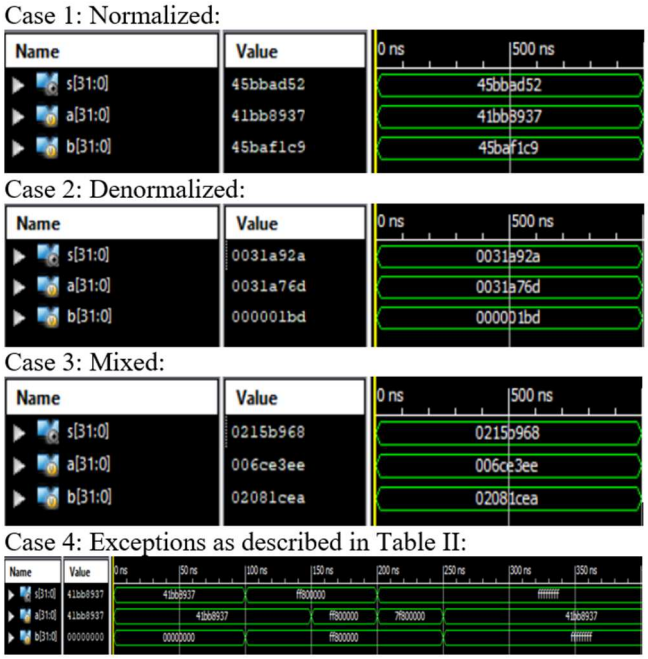

Fig.8. Simulation waveforms for all input cases

VI. Conclusion

The proposed Floating Point adder modifies upon existing designs to further reduce the delay and increases the accuracy of computation by introducing the 'mixed' block. The delay reduction is largely implemented by using parallel paths for operation of different input cases which demand different hardware requirement. This leads to an increase in the area and power. However, we gain additional functionality to handle 'mixed' numbers directly at a hardware level thereby increasing accuracy. In addition to this, a further reduction in delay is achieved by other optimizations such as the usage of a parallel prefix adder for faster summing and rounding operations, and using a state-of-the-art Barrel Shifter. The operations on mixed numbers seem to violate the IEEE format by having a negative exponent bias as described earlier but this assists in the calculations and results in a more accurate computation as shown in the simulation waveforms (Case 3). Coming to timing performance, after synthesis using Cadence Genus, it is found that the proposed design is 28.62% is faster than the SA adder and 16.44% faster than the FCA adder making it a high-speed 32-bit Floating Point Adder.

References

[1] Djordje Pesic and Ivan Ratkovic, "An efficient FPGA implementation of Floating Point addition," *2015 23rd Telecommunications Forum Telfor (TELFOR)*, Belgrade, Serbia, pp. 1-3, 24-26 Nov. 2015

[2] P M Drusya and Vinodkumar Jacob, "Area efficient fused Floating Point three term adder," *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, Chennai, India, pp. 1-4, 3-5 March 2016

[3] Ali Malik and Seok-bum Ko, "A Study on the Floating-Point Adder in FPGAS," *2006 Canadian Conference on Electrical and Computer Engineering*, Ottawa, Ont., Canada, pp. 2-4, 7-10 May 2006

[4] Ali Malik, "*Design Tradeoffs Analysis of Floating-Point Adder in FPGAs*," PhD thesis, University of Saskatchewan, Department of Electrical Engineering, August 2005

[5] IEEE Standard Board and ANSI, "*IEEE Standard for Binary Floating-Point Arithmetic*," 1985, IEEE Std 754-1985.

[6] Castillo, Arturo Barrabés, "*Design of single precision float adder (32-bit numbers) according to IEEE 754 standard using VHDL*." PhD thesis, Bratislava, April 25th 2012

[7] E. M. Schwarz, M. Schmookler and S. D. Trong, "Hardware implementations of denormalized numbers," Proceedings 2003 16th IEEE Symposium on Computer Arithmetic, Santiago de Compostela, Spain, pp. 70-78, 2003

[8] V. G. Oklobdzija, "An Algorithmic and Novel Design of a Leading Zero Detector Circuit: Comparison with Logic Synthesis." IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 124-128, Vol. 2, No. 1., 1994

[9] J. D. Bruguera and T. Lang, "Leading-One Prediction with Concurrent Position Correction," IEEE Transactions on Computers, pp. 1083–1097, vol. 48, no.10., 1999

[10] Matthew Rudolf Pillmeier, "Barrel shifter design, optimization, and analysis," PhD thesis, Lehigh University, 2001

[11] Neil Weste and David Harris. 2010. CMOS VLSI Design: A Circuits and Systems Perspective (4th. ed.). Addison-Wesley Publishing Company, USA. pp. 46