Design of IEEE-754 Double Precision Floating Point Unit Using Verilog

Swathi. G. R
ECE Dept., SJCIT,
Chickballapur, Karnataka, India

Veena. R
Assistant Professor, ECE dept., SJCIT,
Chickballapur, Karnataka, India

***Abstract*-Floating point addition, subtraction and multiplication are widely used in large set of scientific and signal processing computation. Thus an IEEE-754 double precision floating point unit (FPU) is designed in this paper. The proposed design involves logarithmic approach for computing floating point numerical operations. It performs all the four basic arithmetic operations that also handle overflow, underflow, rounding and various exception conditions. The design is coded in Verilog hardware description language and simulated in Questasim.**

Keywords-IEEE 754, Double precision, Floating point unit (FPU), logarithmic approach.

I. INTRODUCTION

Real valued numbers are represented as floating point (FLP) numbers, which has its own standardizations and representations. Hence performing computations on these numbers are quite complicated and different from performing computations on ordinary numbers. The complexities in the computation are reduced by reducing the strength of operations, which can be done by having different number systems (as per Prahami's) [7] like logarithmic number systems (LNS).An IEEE 754 double precision floating point number format has been adopted in this paper. 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 '1' in the MSB of the significand and exponent is greater than zero and smaller than 1023.

Fig 1: IEEE-754 Double Precision Floating Point Number Format

The real numbers are represented by equations (1) and (2).

$$Z = (-1^{s}) * 2^{(E-bias)} * (1.M) \quad (1)$$

$$Value = \left(-1^{\text{sign bit}}\right) * 2^{(\text{Exponent } -1023)} * (1.\text{Mantissa}) \quad (2)$$

II BACKGROUND

A. *Related Works*

Lots of works have been proposed for FPU models, and all suggests some improvements in certain levels at various stages. An island-style with embedded FPU [11] is proposed by Beauchamp et al, and it gives a delay improvement than standard implementation for floating point applications.

A coarse grained FPUs where suggested by ho et al., and they presents a Hybrid architecture, where word blocks were used for computing simple operations. This also suggests improvement in delay as whole complex operations were moved to the coarse grained to reduce routing delays.

[13] Evan suggests a multiplier for performing single precision or either a dual precision floating point numbers, [14] Akkas further tunes Evan's multiplier and presents a quad precision multiplication .

In all these works done suggested the delay [11], [12] improvements and area reductions [13] - [15]. In our paper, we suggest an efficient model for performing the floating point operations by reducing the operation complexities by adopting the log conversion [2] of Sagunath Paul et al.

B. *Conventional Floating Point Computations*

Conventional floating point computation uses the similar computation architectures of integers with slight revision in computation logics. The representation of the floating point numbers itself got three individual parts; the computation has to be done on mantissa with respect to its sign and exponent parts. for example, the floating point adders uses any of the adder structures like carry look ahead adder or ripple carry adder or carry save adder, etc., to add the mantissa terms, but the addition have to be done in accordance with the sign and exponent bits. Let X1(sl,el,ml) and X2(s2,e2,m2) be two floating point numerals. Then X3(s3,e3,m3) be the addition result of X1 and X2, the computation is done basically in three steps as expressed in (3), (4) and (5).

$$s3 = s1 \odot s2 \quad (3)$$

$$e3 = e1 - e2 \quad (4)$$

$$m3 = \{m1\}_d \{\odot\}_{s3} \{m2\}_d \quad (5)$$

The sign bit chooses the operations whereas the exponent decides the operands or bits in mantissas to which be added. This shows the whole addition process apart from adding also includes some extra logics, which increases the complexity.

This will be worse in the case of multiplication in which the random multiplication patterns have to be done on the mantissa parts with respect to their exponent parts. Let X4(s4, e4, m4) be the output of product of floating point numbers X1 and X2, then the computation done as expressed in (6), (7) and (8).

$$s4 = s1 \odot s2 \quad (6)$$
$$e4 = e1 + s2 \quad (7)$$
$$m4 = \{m1\}e1 \odot \{m2\}e2 \quad (8)$$

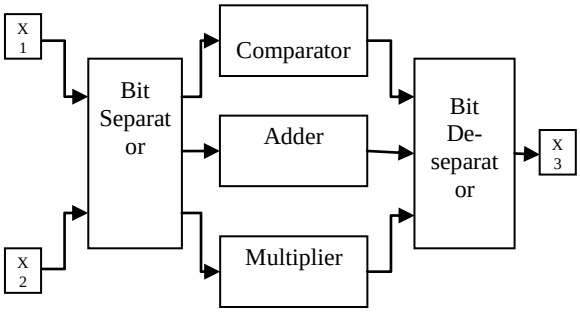

Fig 2: Conventional Floating Point Multiplier

C. LNS Representation

In contrast with FLP numbers, LNS includes neither an integer exponent nor separate linear mantissa. It is much simpler because it uses a single scaled exponent and can be represented by:

$$(-1)^s \times 2^{m.f} \quad (9)$$

where s, m and f indicate sign, integer and fractional bits respectively. Although there is no commonly accepted standard for the LNS format, the most widely used format is shown in Figure 3.

Sign (1-bit)	Fixed-Point Logarithmic Value	
	Integer (*m*-bit)	Fractional (*f*-bit)

Fig 3: Basic Components of Logarithmic Number Format

Typically, base-2 logarithms are used in LNS computations though in principle any base can be used. When the real numbers represented are signed, LNS has a maximum and minimum range between 2^-128 to ≈ 2^+128, ≈ 2.9E – 39 to 3.4E + 38. A special arrangement of bits is used to indicate the real number zero.

III.OUR APPROACH

A combination of two different data formats, including elements from both LNS and FLP systems, has been adopted in this design. These allow the multiply and divide operations to be rapidly computed using the LNS format, while addition and subtraction are processed efficiently in FLP representation. The concept of this hybrid number system design is shown below in figure 4.

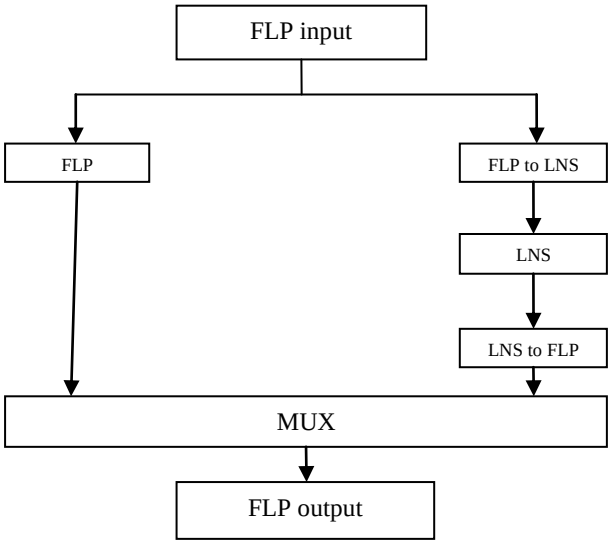

Fig 4: Concept of Hybrid Number System

An optimized architecture for Floating point unit is shown in Figure 5. As the computation of floating point numerals are very complicated, it demands a separate unit for its processing and this leads to the design of FPUs. By exploring the existing FPUs the phenomenon of arithmetic computations are still the same as the ordinary ALU operations, it just acts like an additional prop up for normal ALUs.

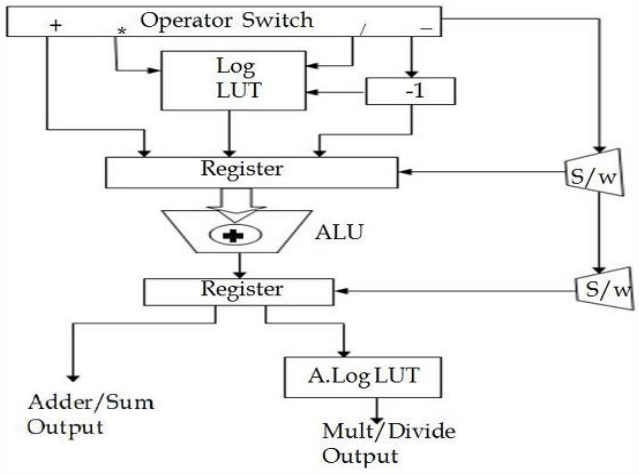

Fig 5: Proposed Model

A. Addition and Subtraction

Addition or subtraction computation phenomenon is done by adding or subtracting the mantissa bits according to the exponent and sign bits, and the steps involved in the computing are described as follows.

- The difference of the two exponents is calculated. If any, perform the mantissa shift and set the larger exponent as the tentative exponent of the result.

- Shift the mantissa of the smaller exponent to right by the bits of difference in the exponents.
- According the sign bit perform addition (if equal) or subtraction (if unequal) on mantissas, to get the tentative mantissa for result.
- Check for exception. If yes, set the signal exception.
- If no, normalize and round off the mantissa result. If there is an overflow due to rounding, shift right and increment exponent by 1 bit.
- Have the highest of sign bits as the sign bit of the result.

B. Multiplication and Division

Multiplication or division computation procedure involved in this design simplifies the overall data-path of the FPU, as there is mere computation with mapping only involved and this simplifies the overall stages involved in computation. The stages are described as follows.

- Map the mantissa of the input data to the corresponding logarithmic number in the LUT. Its value is given by $(-1)^s \times 2^{m.f}$, which provides a similar representation range to FLP numbers.
- Add/Subtract the logarithms, if any overflow shift to the right, and map with antilogarithm LUT to obtain the mantissa of the result.
- The exponent of the result is obtained by adding the exponent bits.
- The sign bit of the result is obtained by the xor of both sign bits.

IV. RESULTS AND DISCUSSIONS

This section shows the simulated results of the proposed design coded in Verilog. The proposed design is verified using the Questasim simulator version 6.4c.Snapshots of the addition, subtraction, multiplication and division are shown below.

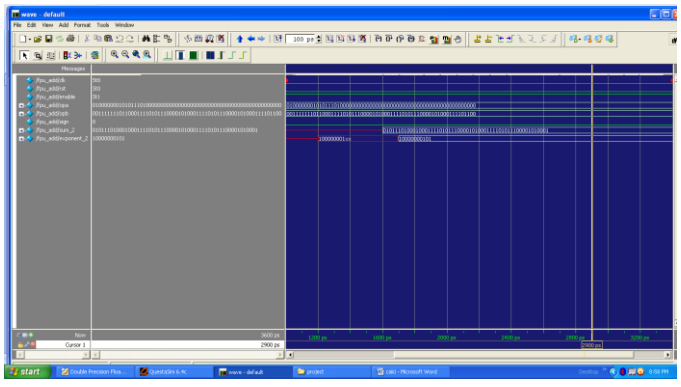

Fig 6: Snapshot of Floating point adder adding two numbers

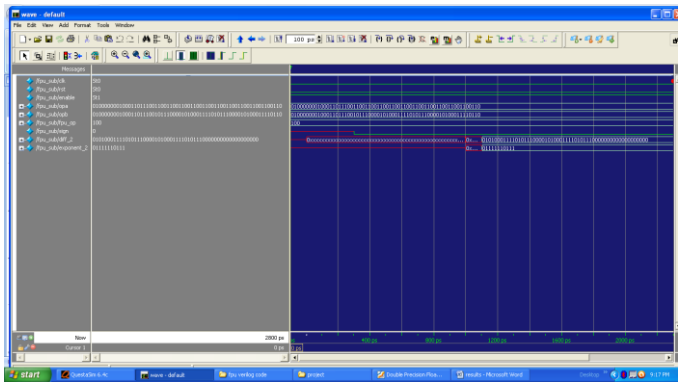

Fig 7: Snapshot of Floating point subtractor subtracting two numbers

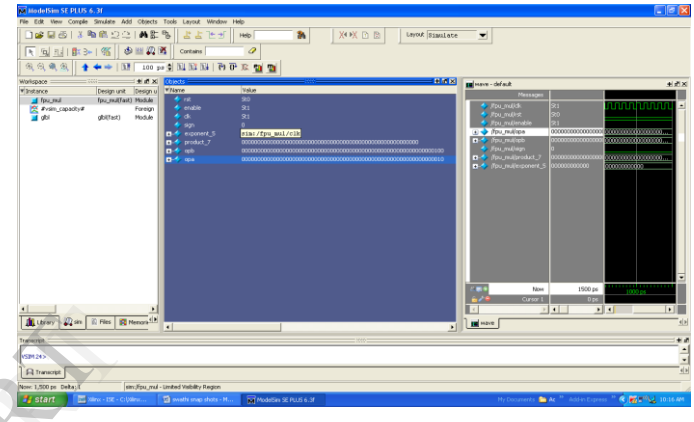

Fig 8: Snapshot of Floating point multiplier multiplying two numbers

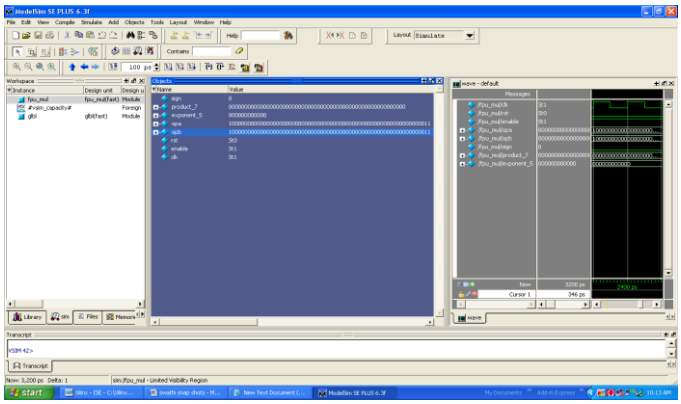

Fig 9: Snapshot of Floating point divider dividing two numbers

V. CONCLUSION

This paper presents an IEEE-754 double precision floating point unit capable of performing double precision addition, subtraction, multiplication and division.The use of logarithmic approach for multiplication and division reduces the complexity of computation and simplifies the overall data-path of the FPU. The main requirement for this model is the LUTs and its memory occupancies lead to the latency in the model.

REFERENCES

[1] ChiWai Yu,Alastair M. Smith, Wayne Luk, Philip H.W. Leong and Steven J.E.Wilton, "optimizing floating point units in Hybrid FPGAs,. IEEE trans. on Very Large Scale Integr. (VLSI) systems, vol. 20, no.7,pp 45- 65, July 20 12.

[2] Suganth Paul, Nikhil Jayakumar, and Sunil P. Khatri," A fast hardware approach for approximate, Efficient logarithm and antilogarithm computations", IEEE trans. on very large scale integr.(VLSI) systems, vol. 17, no. 2, February 2009, pp. 269-277.

[3] Yee Jern Chong and Sri Parameswaran," configurable multimode embedded floating-point units for FPGAs ", IEEE trans. on very large scale integr.(VLSI) systems, vol. 19, no. II,pp.2033-44, Nov 20 II.

[4] G. Govindu, S. Choi, V. K. Prasanna, V. Daga, S. Gangadharpalli, and V. Sridhar, "A high-performance and energy efficient architecture for floating-point based LU decomposition on FPGAs," in Proc. I II" Reconfigurable Arch.Workshop (RAW), Santa Fe, NM, Apr. 2004,pp. 149-153

[5] M. de Lorimer and A DeHon," Floating point sparse matrix-vector multiply for FPGAs,"in Proc. ACM Int. Symp. Field Program.Gate Arrays, Monterey, CA, Feb. 2005, pp. 75-85.

[6] K. H. Abed and R. E. Siferd,"CMOS VLSI implementation of a low power logarithmic converter, " IEEE Trans. Computers, vol. 52, no. II, pp. 1421-1433, Nov. 2003.

[7] IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754, 1985.

[8] Y.J.Chong and S. Parameswaran, "Flexible multi-mode embedded floating-point unit for field programmable gate arrays," in Proc. FPGA, 2009, pp. 171-180.

[9] S. S. Demirsoy and M. Langhammer, "Cholesky decomposition using fused datapath synthesis," in Proc. FPGA, 2009, pp. 241-244.

[10] M. Langhammer and T. VanCourt, "FPGA floating point datapath compiler," in Proc. FCCM, 2009, pp. 259-262. [II] M. J. Beauchamp, S. Hauck, and K. S. Hemmert, "Embedded floating point units in FPGAs," in Proc. IEEE Symp. Field Program. Gate Arrays (FPGA), 2006, pp. 12-20.

[12] C. H. Ho, C. W. Yu, P. H. W. Leong, W. Luk, and S. 1. E. Wilton, "Domain-specific hybrid FPGA: Architecture and floating point applications," in Proc. Int. Con! Field Program. Logic Appl. (FPL). 2007.

[13] G. Even, S. M. Mueller, and P.-M. Seidel, "A dual precision IEEE floating-point multiplier," Integr.'. VLSI J, voL 29, no. 2, pp. 167-180, 2000

[14] A Ye and J. Rose, "Using Bus-Based Connections to Improve Field programmable Gate Array Density for Implementation Datapath Circuits", IEEE Trans. VLSI, vol. 14, no. 5, pp. 462-473, 2006

[15] Xilinx, Inc., San Jose, CA, "Virtex-ll platform FPGAs:Completedatasheet,"2005.[Online].Available:http://direct.xilinx .combvdocs/publications/ds031.pdf.