JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

DESIGN AND IMPLEMENTATION OF POSIT MULTIPLIER

[1] Dudala.V.N.V.Pavan Raj Kumar ,[2] Y.Syamala,

1 M.TECH in VLSID &ES, Department of Electronics and Communication Engineering Author,[2]Associate professor,
[1] Department of Electronics and Communication Engineering,
[1] Seshadri Rao Gudlavalleru Engineering College,
Gudlavalleru, India

Abstract : In many applications, the floating numbers are generally represented with the use of IEEE floating point number system. With the advancement of the technology the IEEE floating numbers are replaced by posit number system. In general multipliers consume high power and delay in arithmetic operations. The posit number system reduces these parameters with the maximum bit width of the mantissa. These multipliers are usually used in MAC units, in DSP processors, signal processing, graphics and scientific computations. The posit number systems are designed for the reduction of the propagation time and reducing the complexity of the floating point multiplications. The posit encoder and decoder are designed in the processor registers to further reduction of the area overhead. In this paper implementing of Posit Multiplier using Virtex 7 FPGA family.

***IndexTerms*-:Floating point, Posit number system, Multiplier**

I. INTRODUCTION

In digital world most of the people requires the digital devices for low cost and requires high performance and the most important thing is less power consumption mostly for the hand held devices. To obtain those requirements the digital circuits need to achieve high performance on the limited area hence different techniques are proposed to reach their requirements. After comparing all the different designs for the complex multiplication process instead of IEEE floating point number systems, posit number system achieves better results with respect to bit width and high speed. In the floating point number systems division of the data is most important factor hence there is a separation of sign bit, mantissa and exponent. In posit number system along with that there is a one more part is added called ad regime bit due to this calculation is also made simpler for the complex calculations.

Posit, a new datatype designed as a direct replacement for IEEE Standard 754 floating-point numbers (floats). Unlike earlier forms of universal number (unum) arithmetic, posits do not require interval arithmetic or variable size operands; like floats, they round if an answer is inexact. They provide compelling advantages over floats, including larger dynamic range, higher accuracy, better closure, bitwise identical results across systems, simpler hardware, and simpler exception handling.

II. BACKGROUND

The Posit format is a young floating point format and derives from its predecessors Type I and II Unum which was introduced by John L. Gustafson [1]. Type I Unum was a superset of IEEE 754 and supported compatibility and had a more efficient bit representation. However, it had problems with values having dynamical sizes and inherited a lot of the disadvantages with IEEE 754. Type II Unum was an overhaul and supported reciprocals of the arithmetic operations (+ - x /) and also supported decimal representations. The disadvantages were that the format was not precise in certain operations like table look-up which was limited to 20 bits or less. In February 2017 Type III Unum also called Posits were introduced.

According to John L. Gustafson[2] Posits are possible to directly convert to IEEE 754 floats, it supports hardware implementations, it operates faster, the results are more accurate and is more cost efficient than the IEEE 754 in terms of energy consumption. Posits could essentially act as an drop-in replacement for the IEEE 754. The Posit standard definition uses a different approach to the construction of the floating point number compared to IEEE 754 [3]. Currently there are four Posit format ranging from 8-, 16-, 32- and 64-bit precision and a special format called Quire. Each format is defined by a set of properties.Unlike the conventional floating-point format, the bit-width of each component in posit format, as shown in Fig. 1, is dynamic (except the 1-bit sign). The Posit bit-string consists of four bit-fields called Sign, Regime, Exponent and Fraction.

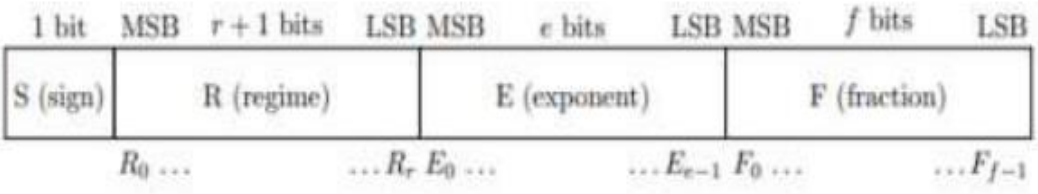

Figure 1: Binary fields of a Posit floating point

III.DESIGN OF POSIT MULTIPLIER ARCHITECTURE

In posit number system the bit width is varied with the sign bit, regime bit, mantissa and exponent. The mantissa doesn't require specific bit length and the most of the bits are used in the regime bits. The calculation of the posit multiplier is mainly done on the regime bits. The mantissa bits are filled with zeros, and these bits are inverted to ones, the partial products of the booth multiplier are reduced with the modified propagations. Hence, these circuits are accumulating the bit positions, and the final adder is still toggling. This may cause the wastage of power consumption and delay.

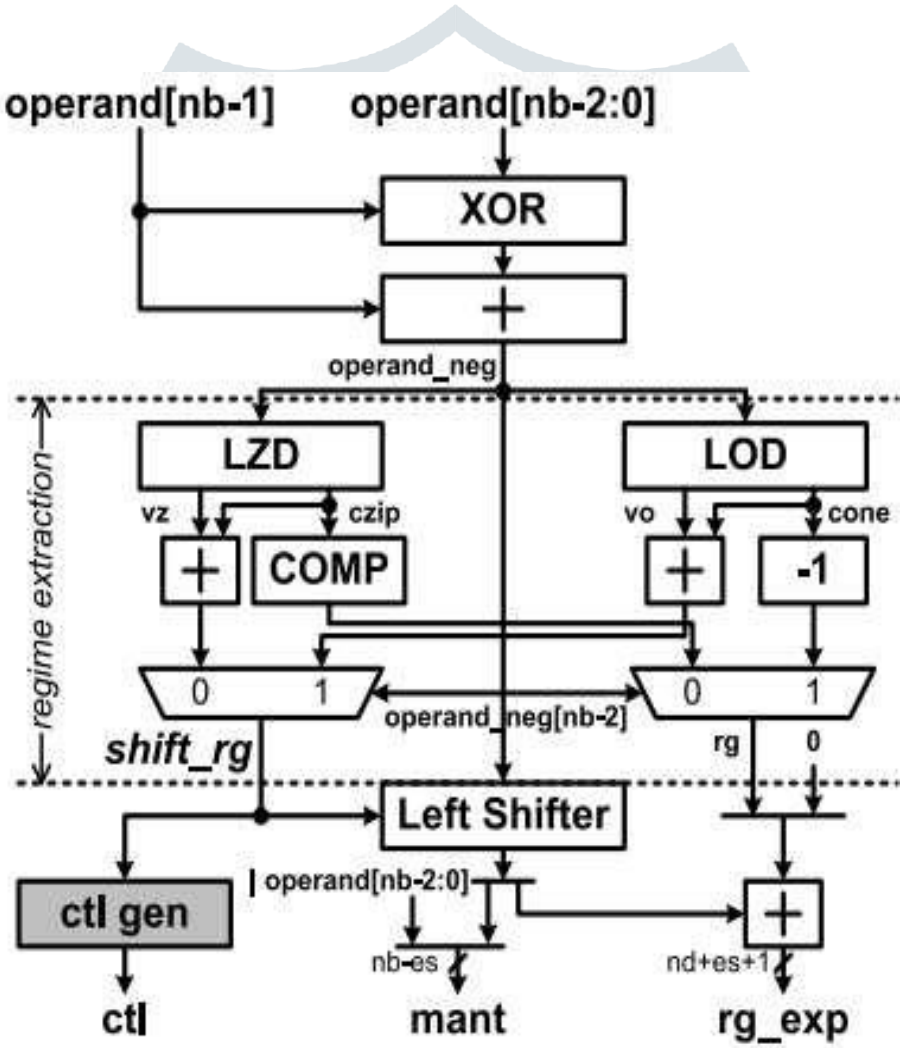

Figure 2: Posit component extraction in hardware arithmetic unit.

The circuit shown in Fig. 2 (except the grey module) is commonly used to extract each component of a posit number.In the posit multiplier architecture, two different changes are made for signal toggling and to reduce the power consumption. Initially the control signalis used to control the multiplication of the mantissa and most of the necessary part of the multiplier is avoided to improve the performance of the multiplier.

The second change has to be done by the mantissa multiplier and small portion of the multiplier is controlled by the control signal. The posit multiplier is used to change the modifications of the leading zeroand the leading one detection. The bit-widths of other components are also shown in Fig. 3.

Figure. 3 shows the mantissa multiplier is divided into different groups, the input is divided into the mantissa and the mantissa is propagated to the inputs of the mantissa multiplier. The sign bit and the exponent is obtained by using the regime bit and the sign bit. The control signal is used to control the mantissa multiplier, the output is propagating to the carry propagate adder and the inputs are taken from the exponent and the mantissa and these are processed to the normalization. The outputs from the normalization has to be done by the posit component packing and the separated bits are combined to achieve the posit number including the sign bit. The final result is obtained by using the rounding technique and the final product is taken after rounding of the final posits number system.

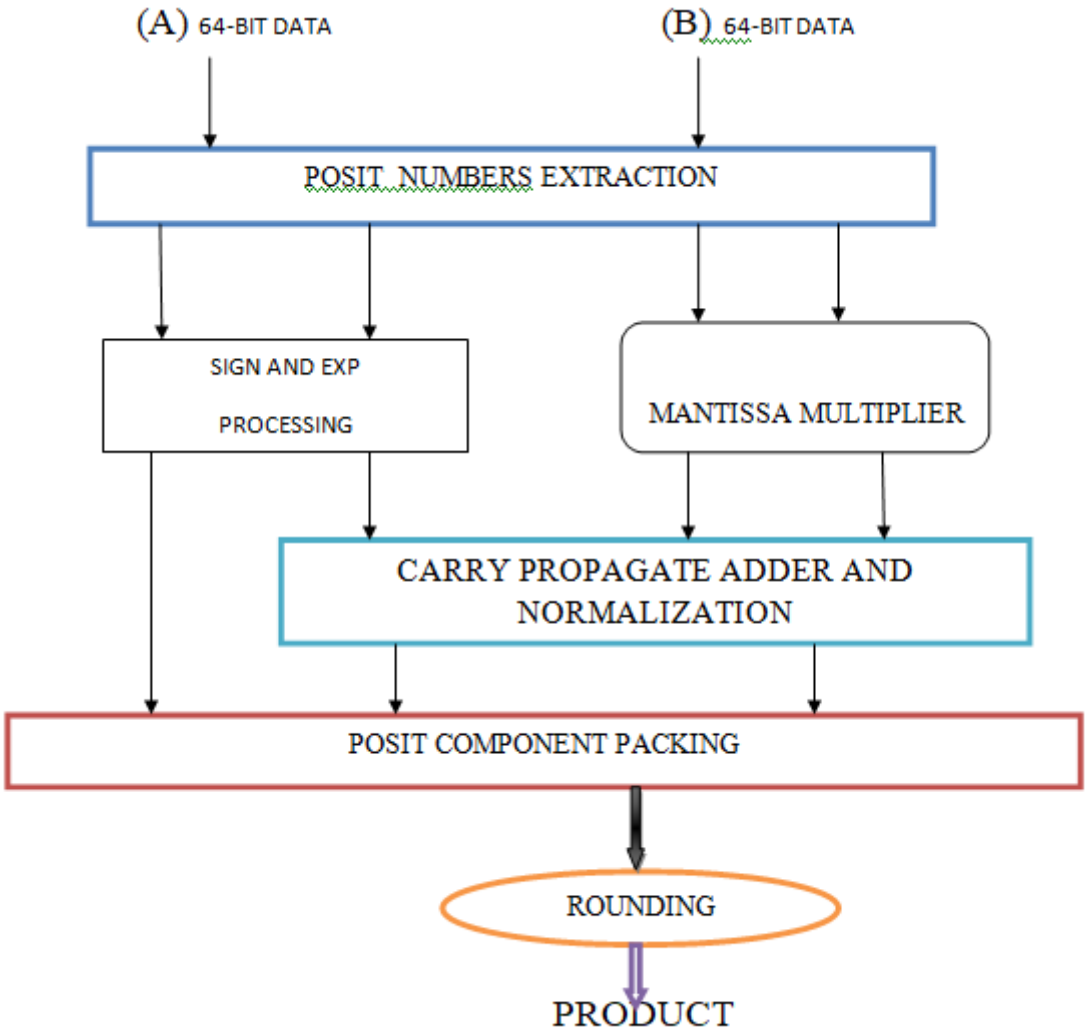

Figure 3: Data path implementation of posit multiplier.

IV.RESULTS

The proposed multiplier is implemented in Xilinx tool with the help of Verilog HDL programming language. The implantation are synthesized by using the xilinx syntheis tool and implemented in the Virtex 7 family. Then the implementation and simulation results are shown below.

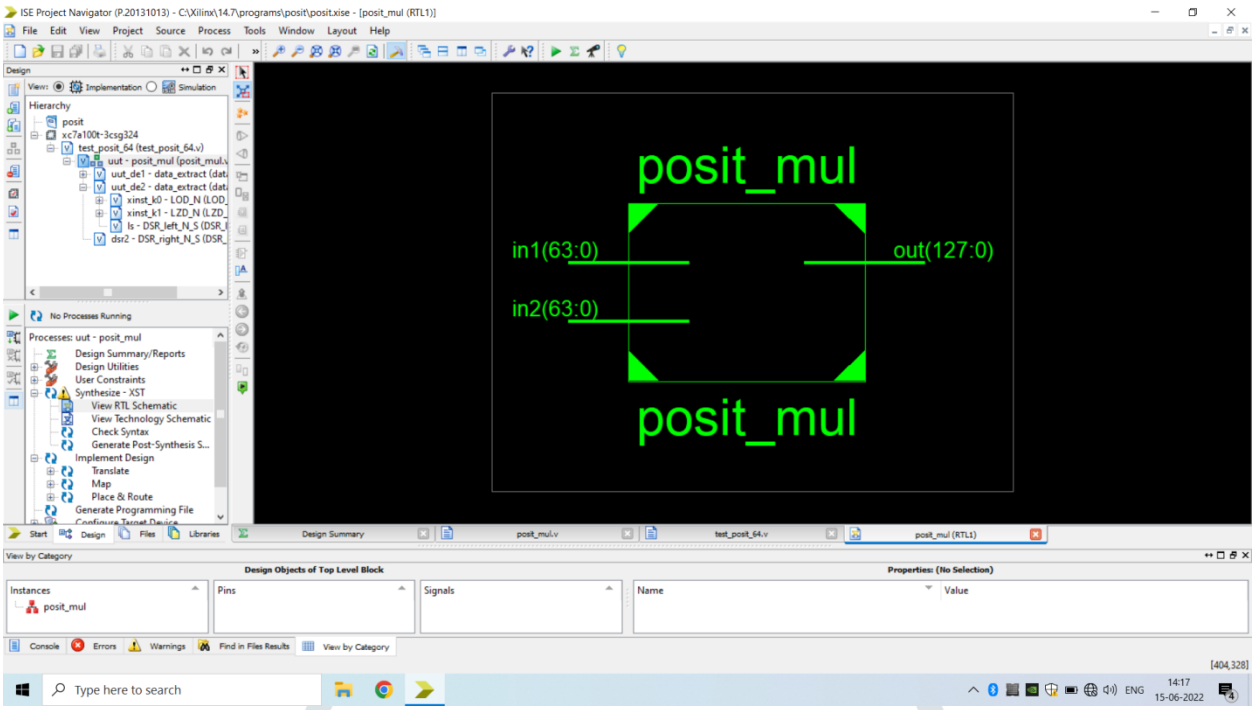

Figure 4: RTL Schematic of 64-bit posit multiplier

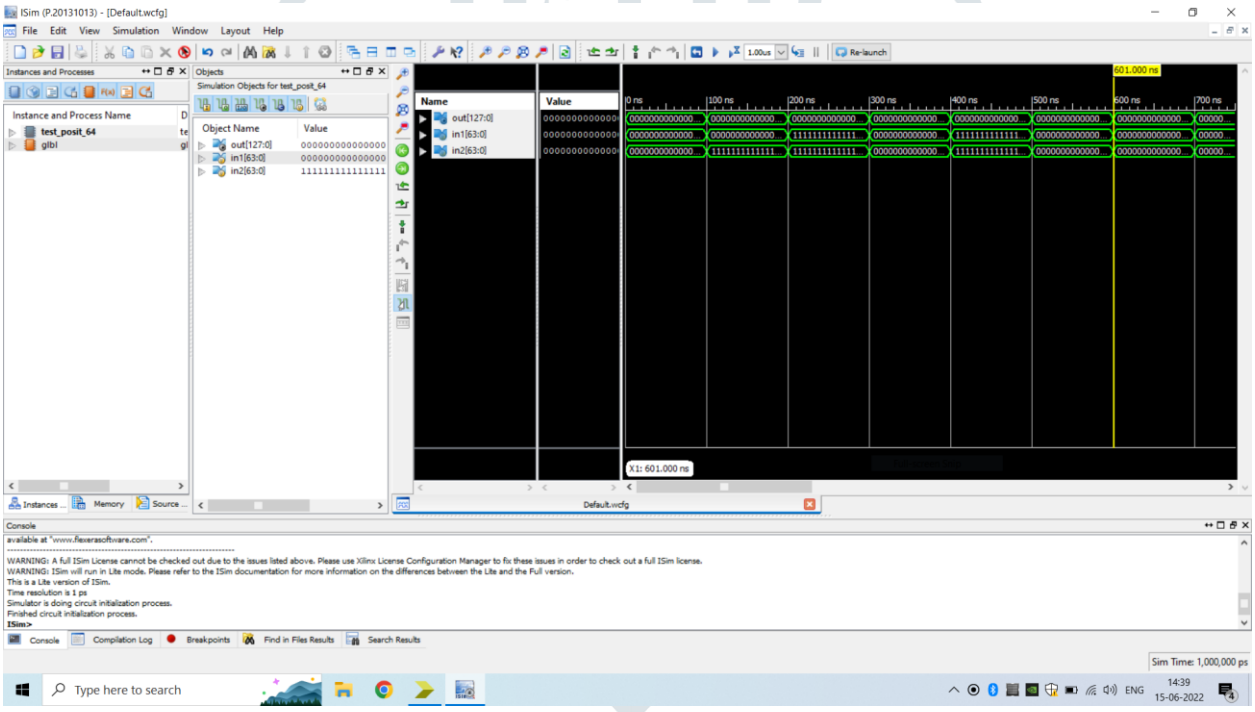

Figure 5: Simulation Waveform of 64-bit posit multiplier

```
MUXCY:CI->O           1    0.023   0.000  Madd_out_cy<0>_120 (Madd_out_cy<0>121)
MUXCY:CI->O           1    0.023   0.000  Madd_out_cy<0>_121 (Madd_out_cy<0>122)
MUXCY:CI->O           1    0.023   0.000  Madd_out_cy<0>_122 (Madd_out_cy<0>123)
MUXCY:CI->O           1    0.023   0.000  Madd_out_cy<0>_123 (Madd_out_cy<0>124)
MUXCY:CI->O           1    0.023   0.000  Madd_out_cy<0>_124 (Madd_out_cy<0>125)
MUXCY:CI->O           0    0.023   0.000  Madd_out_cy<0>_125 (Madd_out_cy<0>126)
XORCY:CI->O           1    0.370   0.279  Madd_out_xor<0>_126 (out_127_OBUF)
OBUF:I->O                 0.000          out_127_OBUF (out<127>)
----------------------------------------
Total                    32.029ns (18.660ns logic, 13.369ns route)
                                  (58.3% logic, 41.7% route)
```

Figure 6: Delay report of 64-bit posit multiplier

V.CONCLUSION

IN THIS PAPER IMPLEMENTED THE 64-BIT POSIT MULTIPLIER USING VIRTEX 7 XILINX FPGA FAMILY. POSIT NUMBER SYSTEMS ARE THE BEST SUITABLE NUMBER SYSTEM FOR THE REPLACEMENT OF THE IEEE 754 FLOATING POINT NUMBER SYSTEM DUE TO THE IMPROVEMENT IN ACCURACY AND REDUCTION IN THE POWER CONSUMPTION AND DELAY. FINALLY, CONCLUDE IMPLEMENTING POSIT MULTIPLIER IN VIRTEX 7 FAMILY IS BETTER RESULTS AS COMPARED TO OTHER OTHER FPGA XILINX FAMILES.

.

REFERENCES

[1] L. van Dam, "Enabling High Performance Posit Arithmetic Applications Using Hardware Acceleration", Master's thesis, Delft University of Technology, the Netherlands, Sep. 17, 2018, isbn: 9789461869579.

[2]J. L. Gustafson, The End of Error: Unum Computing. CRC Press, Feb. 5, 2015, vol. 24, isbn: 9781482239867.

[3]A. A. D. Barrio, N. Bagherzadeh, and R. Hermida, "Ultra-low-power adder stage design for exascale floating point units", ACM Trans. Embed. Comput. Syst., vol. 13, no. 3s, 150:1–150:24, Mar. 2014. doi: 10.1145/2567932.

[4]J. L. Gustafson. (Oct. 10, 2017). Posit Arithmetic, [Online]. Available: https ://posithub.org/docs/Posits4.pdf (visited on Mar. 13, 2019).

[5]J. L. Gustafson, "A Radical Approach to Computation with Real Numbers", Supercomputing Frontiers and Innovations, vol. 3, no. 2, pp. 38–53, Sep. 2016. doi: 10.14529/jsfi160203.

[6]Posit Working Group. (Jun. 23, 2018). Posit Standard Documentation, [Online]. Available: https://posithub.org/docs/posit_standard.pdf (visited on Apr. 30, 2019).

[7]R. Munafo. (2018). Survey of Floating-Point Formats, [Online]. Available: http://www.mrob.com/pub/math/floatformats.html (visited on Feb. 9, 2019).

[8]D. Goldberg, "What every computer scientist should know about floating-point arithmetic", ACM Computing Surveys (CSUR), vol. 23, no. 1, pp. 5–48, Mar. 1991. doi: 10.1145/103162.103163.

[9]IEEE Computer Society Standards Committee and American National Standards Institute, "IEEE Standard for Binary Floating-Point Arithmetic", ANSI/IEEE Std 754-1985, 1985. doi: 10.1109/ieeestd.1985.82928.

[10]"IEEE Standard for Floating-Point Arithmetic", IEEE Std 754-2008, pp. 1–70, 2008. doi: 10.1109/ieeestd.2008.4610935.

[11]W. Kahan and J. D. Darcy, "How Java's floating-point hurts everyone everywhere", in ACM 1998 workshop on Java for High–Performance Network Computing, Stanford University, 1998, pp. 1–81.

[12]J. L. Gustafson and I. T. Yonemoto, "Beating Floating Point at its Own Game: Posit Arithmetic", Supercomputing Frontiers and Innovations, vol. 4, no. 2, pp. 71–86, Jun. 2017. doi: 10.14529/jsfi170206.

[13]M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org, [Online]. Available: https://www.tensorflow.org/.

[14]S. van der Linde, "Posits alsvervanging van floating-points: Eenvergelijking van Unum Type III Posits met IEEE 754 Floating Points met Mathematica en Python", Bachelor's Thesis, Delft University of Technology, Sep. 26, 2018.

[15]S. H. Leong. (2018). SoftPosit-Python, [Online]. Available: https://gitlab.com/ cerlane/SoftPosit-Python (visited on Apr. 20, 2019).

[16]K. Mercado. (2017). PySigmoid, [Online]. Available: https : / / github . com / mightymercado/PySigmoid (visited on Feb. 10, 2019).

[17]Speed Go Computing. (2018). NumPy (on top of SoftPosit), [Online]. Available: https://github.com/ xman/numpy-posit (visited on Apr. 20, 2019).