FPGA Based Double Precision Floating Point Multiplier enabled threshold comparator

Magilipally Sathya Laxmi, Konda Kiran Kumar, Gudipalli Kalyan

Abstract— **With the advent of technology, the demand of high-speed digital systems is on the rise and the multiplier is a ubiquitous unit in almost every digital system. compared to other operations in an arithmetic logic unit the multiplier consumes more time and power. Floating point units are widely used in a dynamic range of engineering and technology applications. This demands for the development of faster floating point arithmetic circuits.**

In this project we propose an architecture for a fast floating point multiplier compliant with the single precision IEEE 754-2008 standard. In this proposed architecture we are trying to minimize the carry propagation at every level possible.

The floating point multiplication is carried out in three parts: In the first part, we determine the sign of the product by performing a xor operation on the sign bits of the two operands. In the second part, the exponent bits of the operands are passed to an adder stage and a bias of 1023 is subtracted from the obtained output. In the third stage multiplication of mantissa is performed in 4 stages namely a) Partial product generator b) Partial product accumulator c) Final stage adder d) Normalization and Rounding.

The proposed architecture is simulated using Modelsim and synthesized using Xilinx ISE and it will be implemented on FPGA board for hardware implementation and testing. The Xilinx Chip scope tool will be used to test the FPGA inside results while the logic running on FPGA.

In addition to this a threshold comparator is developed which used to triggering the other circuits based on the threshold value.

Index Terms— **Double Precision Floating Point, Multiplier, FPGA, Xilinx, Modelsim.**

I. INTRODUCTION

The IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754) is the most widely used standard for floating-point computation, and is followed by many CPU and FPU implementations. The standard defines formats for representing floating-point number (including ±zero and denormals) and special values (infinities and NaN's) together with a set of floating-point operations that operate on these values. It also specifies four rounding modes and five exceptions. IEEE 754 specifies four formats for representing floating-point values: single-precision (32-bit), double-precision (64-bit), single-extended precision (≥ 43-bit, not commonly used) and double-extended precision (≥ 79-bit, usually implemented with 80 bits). Many languages specify that IEEE formats and arithmetic be implemented, although sometimes it is optional. For example, the C programming language, which pre-dated IEEE 754, now allows but does not require IEEE arithmetic (the C float typically is used for IEEE single-precision and double uses IEEE double-precision).

Floating point numbers are one possible way of representing real numbers in binary format; the IEEE 754 standard presents two different floating point formats, Binary interchange format and Decimal interchange format. Multiplying floating point numbers is a critical requirement for DSP applications involving large dynamic range. This paper focuses on double precision floating point binary interchange format. Figure 1 shows the IEEE 754 double precision floating point binary format representation; it consists of a one bit sign (S), an eleven bits exponent (E), and a fifty two bits fraction (M or Mantissa). An extra bit is added to the fraction to form what is called the significand1. If the exponent is greater than 0 and smaller than 2047, and there is 1 in the MSB of the significant then the number is said to be a normalized number, Significant is the mantissa with an extra MSB bit.

This paper presents a new floating-point multiplier which can perform a double-precision floating-point multiplication or simultaneous single precision floating-point multiplications. Since in single precision floating-point multiplication results are generated in parallel, the multiplier's performance is almost doubled compared to a conventional floating point multiplier.

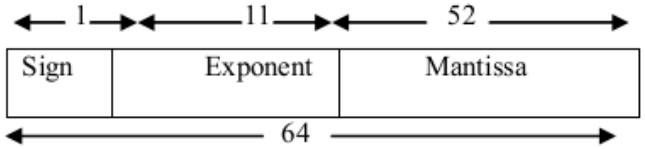

Fig.1 IEEE 754 Floating Point Representation

II. BLOCK DIAGRAM IMPLEMENTATION

A. *Review Stage*

In this project the implementation and simulation results of basic blocks required for Double precision floating point multiplier are presented.

The main blocks of the design are given as follows:

1) Exponent Adder.

2) Mantissa multiplier

3) Sign detector module (EX-or gate)

4) Normalization Unit

5) Threshold comparator

ISSN: 2278 – 7798

The block diagram of the implementation is given in the figure below.

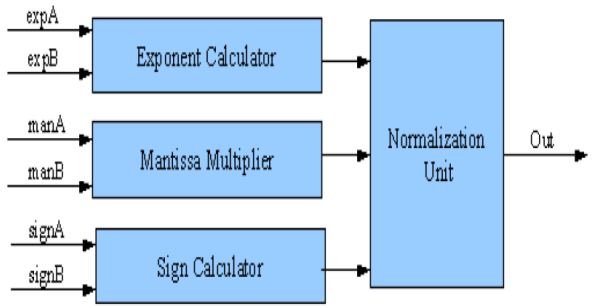

Fig.2 Floating Point Multiplier High Level Block Diagram

B. Exponent Adder

An exponent adder is used to add two input exponents. In this case we add 11bit input exponents using a generic unsigned binary adder. The std_logic_vector's are converted to signed data type and simply added by using ' + ' operator. The ' + ' operator makes use of adder logic given in std_logic_arith package of IEEE library. As the exponent fields (Ea and Eb) are biased, the bias must be removed in order to do the addition. And then, we must to add again the bias, to get the value to be entered into the exponent field of the result (Er).

$$Er = (Ea-1023) + (Eb-1023) + 1023$$
$$= Ea + Eb – 1023$$

For double precision multiplier the Bias is equal to 1023. The exponent sum should be subtracted with the bias value. This is called as Bias adjustment. In this exponent adder we also have an input called exp_adju which comes from the mantissa multiplier. This is for the Normalized result after multiplication. If the significand is unnormalized then we convert it into normalized form by incrementing or decrementing the exponent.

C. Mantissa multiplier

The mantissas of two operands will be multiplied by using mantissa multiplier. Normally mantissa for double precision floating point number is of 52 bits, so a 52x52 binary multiplier is used to compute this value. The multiplier output holds 104 bits of data. The same is transferred to normalization unit.

$$M_r = M_a \times M_b$$

Where M_r is the Mantissa resultant, M_a is the Mantissa of opearand A and M_b is the Mantissa of operand B.

D. Sign detector

This module generates sign of the resultant. In general, if sign of both operands is either positive or negative then the resultant sign will be positive. If one has positive and the other is negative then the resultant sign is negative. This is achieved simply by doing XOR operation for both operands. The sign bit represented as '1' if the resultant has negative sign otherwise '0'.

E. Normalization Unit

This module collects data from exponent adder, mantissa multiplier and sign detector. It normalizes the data bits as per final bias from exponent adder and adjusts the data so as to fit in IEEE standard format which is 1 sign bit, 11 exponent bits and 52 mantissa bits.

F. Threshold comparator

This module is used to generate a flag by comparing the double precision floating point multiplication result with given threshold which is also a double precision value. This generates a comparator flag based on the threshold value and multiplier result.

There are two flags generated, one is comparator out and the other is comparator fail. Comparator out flag is high when the floating point multiplier result crosses threshold value otherwise it is low. Comparator fail is the flag used to indicate the floating point result is exactly same as the threshold; it is simply alternative to equal flag. This helps in simply triggering the other circuits whenever required. This comparison is performed for every new resultant and with the threshold. The user can also change the threshold value at any point of time.

III. Simulation Results

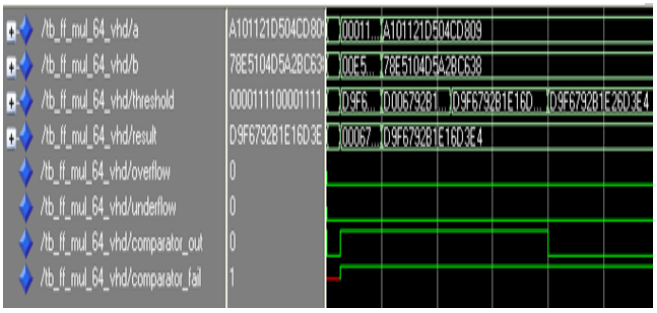

Fig.3 double precision floating point multiplier results in hexadecimal

The above figure shows final result of the double precision floating point multiplier. The first two signals are operands A, B. Next signal is threshold which is extra logic included in the design for the threshold comparison. The next signal is result which is the final required value after doing multiplication. The signals comparator out and comparator fail are used for threshold indication. This is used for triggering the other circuits based on the result value.

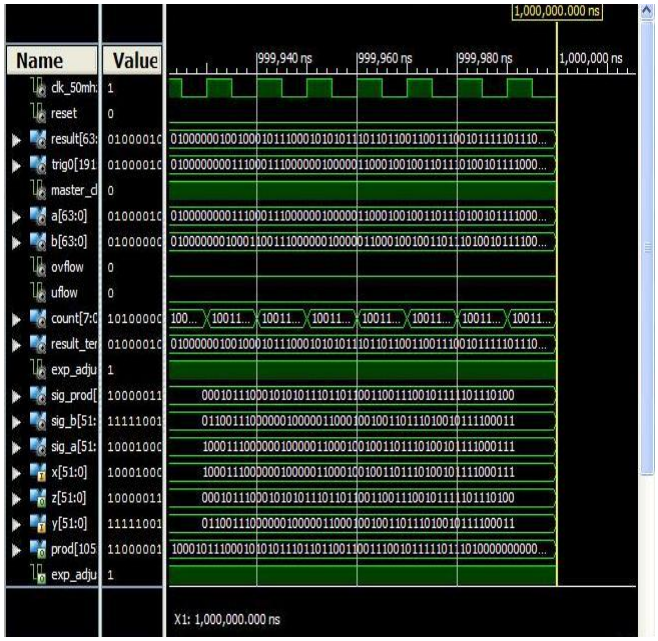

Fig.4 double precision floating point multiplier results in binary

The above figure shows the internal signals and their values which are used in the design.

IV. SPARTAN 3E

A. *Spartan 3E Board*

Fig. 5 Spartan 3E board

B. *FPGA Structure*

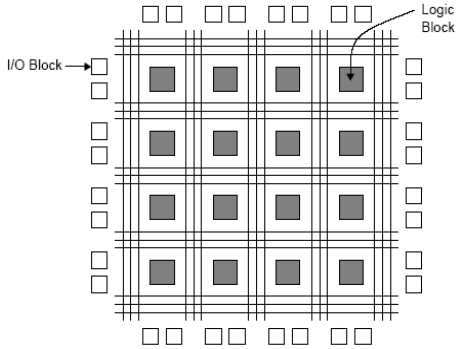

Fig. 6 FPGA structure

V. CHIPSCOPE RESULTS

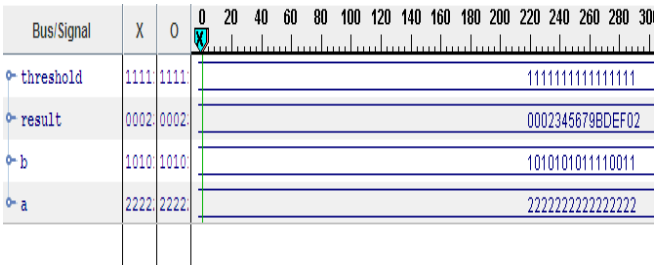

Fig.7 chip scope results for floating point multiplier

VI. TABLE

Logic utilization	used	available	utilization
Number of slice flipflops	779	9312	8%
Number of 4 input LUT's	954	9312	10%
Number of occupied slices	778	4656	16%
Number of bonded IOB's	66	232	28%

VII. CONCLUSION

This paper presents an implementation of a floating point multiplier that supports the IEEE 754-2008 binary interchange format. This design holds not only floating point multiplier as well as a threshold comparator which is useful to trigger the other circuits automatically. The design is implemented on a Xilinx Spartan3E XC3S500E FPGA. Simulation results are well verified using simulator modelsim 6.2c and synthesis results are validated.

REFERENCES

[1] IEEE standards board, IEEE standard for floating-point arithmetic, 2008

[2] Paschalakis, S., Lee, P., "Double Precision Floating-Point Arithmetic on FPGAs", In Proc. 2003, 2nd IEEE International Conference on Field Programmable Technology (FPT '03), Tokyo, Japan, Dec. 15-17, pp. 352-358, 2003

[3] Hamacher, Carl, Vranesic, Zvonko, Zaky, Safwat, "Computer Organization" Fifth Edition, pp. 367-390

[4] Hamid, L.S.A., Shehata, K., El-Ghitani,H., ElSaid,M.,"Design of Generic Floating Point Multiplier and Adder/Subtractor Units" , 12'h International Conference on Computer modelling and Simulation, 2010, pp.615-618.

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

[6] F.de Dinechin and B.Pasca. Large multipliers with fewer DSP blocks.In Field Pro- grammable Logic and Applications. IEEE, Aug. 2009.

ISSN: 2278 – 7798

[7] IEEE 754-2008, IEEE Standard for Floating-Point Arithmetic, 2008.
[8] Patterson, D. & Hennessy, J. (2005), computer Organization and Design : The Hardware/software Interface , Morgan Kaufmann.
[9] B. Lee and N. Burgess, "Parameterisable Floatingpoint Operations on FPGA," Conference Record of the Thirty-Sixth Asilomar Conference on Signals, Systems, and Computers, 2002
[10] A. Jaenicke and W.Luk, "Parameterized FloatingPoint Arithmetic on FPGAs", Proc. of IEEE ICASSP, 2001, vol. 2, pp. 897-900.

Authors:

MOGILIPALLY SATYA LAXMI pursuing M.Tech in VLSI SYSTEM DESIGN at Nagarjuna Institute of Technology & Sciences.

KONDA KIRAN KUMAR, working as an Assistant Professor at Nagarjuna Institute of Technology & Sciences.

G. KALYAN is presently working as Application Engineer in Unistring Tech Solutions Pvt Ltd, Hyderabad, Telangana, India.

