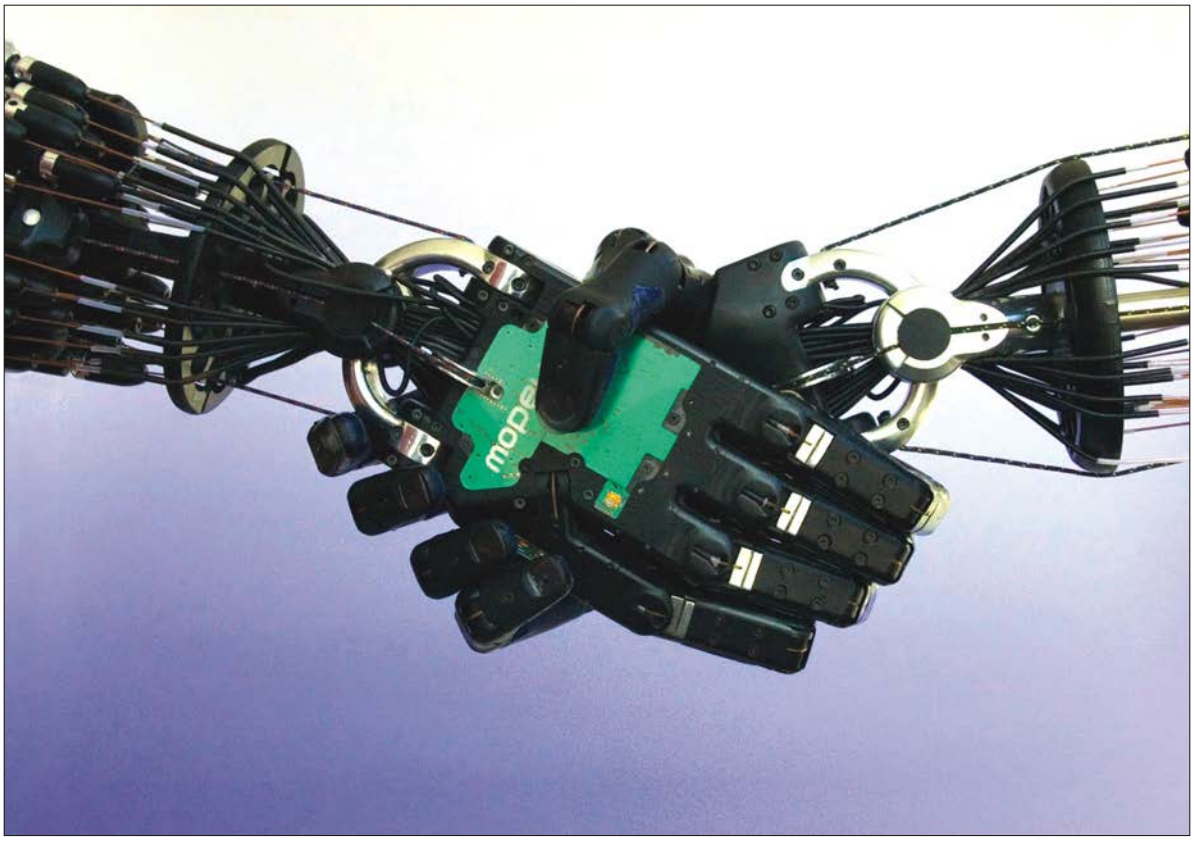

Journal of Artificial Intelligence

ISSN 1994-5450

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Artificial Intelligence 6 (2): 154-160, 2013
ISSN 1994-5450 / DOI: 10.3923/jai.2013.154.160

High Performance and Fault Tolerance Double Precision Floating Point Arithmetic Units

N. Vinothkumar, M.S. Ravi and Kittur Harish Maillikarju
India VIT University, India

Corresponding Author: N. Vinothkumar, India VIT University, India

ABSTRACT

The floating point arithmetic units are complex in their algorithms and many scientific problems require floating point units with high accuracy. Hence for increased performance and fault tolerance operations the double precision floating point arithmetic units adder, subtractor, multiplier and divider is designed which is enough for most System on Chip (SoC) applications and it also improves the accuracy during long chain of computations. The synthesized code results are verified and the complete layout is generated using backend flow.

Key words: Floating point adder, subtractor, multiplier, divider, backend flow

INTRODUCTION

Scientific and engineering computations require high performance floating-point units (FPU). The invention of many SoC applications results in need for, high accuracy FPUs with increased performance. Over the last decade, a number of FPU designs have been presented (MSC, 2008). Among floating point numbers, the single precision floating point format is suitable for many applications, but this range is somewhat small so that in many financial, scientific and other applications faces some errors during computations (Ciricescu *et al*., 2003). Moreover, in serious chains of computations, the small range precision of the single precision format will results in complex error (Astrom and Wittenmark, 1995). In order to overcome this problem the double precision Floating point format with twice the range than single precision and it has an 11-bit excess 1023 exponent bit and a 52 mantissa bit and a sign bit. This helps to get a dynamic range of about 16 digits of precision, sufficient for most applications. Hence to improve accuracy during serious chains of computations with dynamic range double precision floating point numbers we have designed a double precision arithmetic units adder, subtractor, multiplier and divider that functions as ALU for various applications.

ADDER

Floating-point addition operation is the most frequent operation that is suitable for most of the scientific operation in mathematical processors, embedded processors and some data processing units. These processors require high stability and accuracy in their results (Hennessy and Patterson, 1996). To achieve this double precision floating point adder is designed. In the initial stage the exponent bits and the mantissa bits of both operands are separated and the exponent bits of both the operands are compared with each other to find the larger exponent operand and the mantissa of operand with smaller exponent value is right shifted by number of difference in the exponent value to make exponent value of both the operands equal before performing addition. If the exponents are equal then the mantissa bits

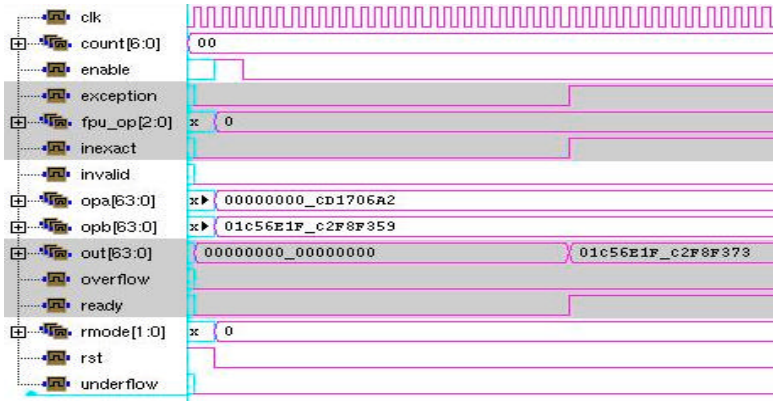

Fig. 1: Simulation results of adder

of both the operands are added directly. In order to improve the accuracy and overflow condition an extra one bit is included in the mantissa part and it is used for rounding purpose.

The infinity, underflow, overflow and invalid condition are checked during rounding. The rounding is carried out in four modes Round to nearest (mode = 00), Round to zero (mode = 01), Round to positive infinity (mode = 10),Round to negative infinity (mode=11) based on the mode specified during the input. Figure 1 shows the simulation results of adder. The signal opa (63:0) and opb (63:0) shows the 64 bit input operands A and B respectively in hexadecimal value. The out (63:0) signal shows the output of addition of operand A and B. The 64 bit double precision floating point format is shown in hexadecimal value. The signals invalid, inexact, overflow, underflow shows the exceptions conditions of adder.

SUBTRACTOR

Subtraction is similar to addition in that you need to calculate the difference in the exponents between the two operands and the mantissa bits of the operand with smaller exponent value is right shifted before subtracting the mantissa part. The definition of the subtraction operation is to take the number in operand B and subtract it from operand A (Standards Committee of the IEEE Computer Society, 1985). However, to make the operation easier and for high performance the smaller number will be subtracted from the larger number and if A is the smaller number, it will be subtracted from B and then the sign bit of the result is inverted. The numbers with non-zero exponent value is normalized numbers. An extra bit'1' is added to the mantissa part for normalized numbers and this extra bit is used during the rounding of results. The infinity, underflow, overflow and invalid condition are checked during rounding and the different rounding modes that carried out for subtraction same as the addition.

In order to improve the accuracy the mantissa part of the result is stored in register and a signal counts the number of 0's in the register before the leftmost '1'. And the exponent is reduced to no. of zeros and the leftmost bit '1' becomes the leading bit of the mantissa part of the result hence due to increasing the mantissa part the accuracy of the result is increased. The Fig. 2 shows the simulation results of subtractor. The signal opa (63:0) and opb (63:0) shows the 64 bit input operands A and B, respectively in hexadecimal value. The out (63:0) signal shows the output of subtraction of operand A and B. The 64 bit double precision floating point format is shown in hexadecimal value. The signals invalid, inexact, overflow, underflow shows the exceptions conditions of subtractor.

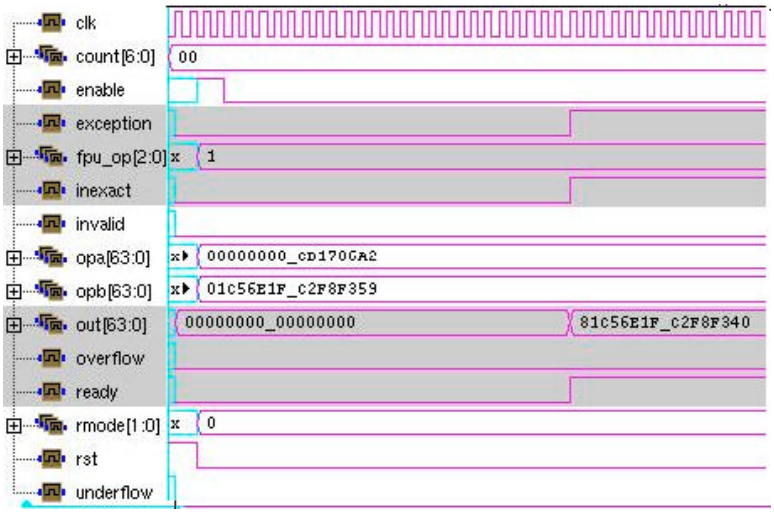

Fig. 2: Simulation results of subtractor

MULTIPLIER

The multiplication operation is performed depending on efficient advantage of the multiplier resources in the target FPGA device so that it increases the performance of device. The exponent bits and the mantissa bits of both operands are separated and the exponent bits of both the operands are compared with each other to find the larger exponent operand and to check whether the number is normalized or de normalized number. The mantissa bits of operand A and operand B and the leading '1' (for normalized numbers) are stored separately in the 53-bit register. Multiplying all 53 bits of A by 53 bits of B would result in a 106 bit product. The multiplier is designed based on the Xilinx Virtex 5 a device contains DSP48E slices with 25 by 18 twos complement Multipliers, which can perform a 24-bit by 17-bit unsigned multiply (Louca *et al.*, 1996). Hence for high performance the 53-bit by 53- bit floating point multiply is broken into smaller components is described as:

- Product_a = A (23:0)×B (16:0)
- Product_b = A (23:0)×B (33:17)
- Product_c = A (23:0)×B (50:34)
- Product_d = A (23:0)×B (52:51)
- Product_e = A (40:24)×B (16:0)
- Product_f = A (40:24)×B (33:17)
- Product_g = A (40:24)×B (52:34)
- Product_h = A (52:41)×B (16:0)
- Product_i = A (52:41)×B (33:17)
- Product_j = A (52:41)×B (52:34)

The products (a-j) are added together, with the appropriate offsets based on which part of the A and B arrays they are multiplying. The summation of the products is accomplished by adding one

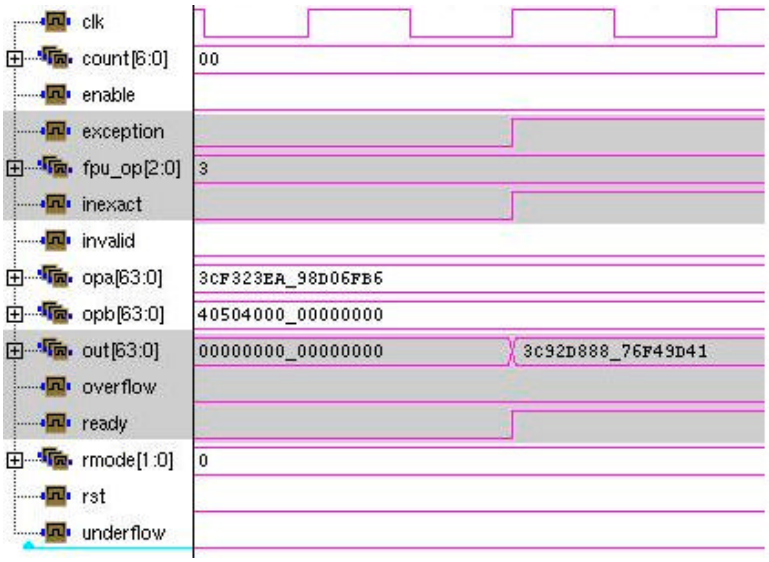

Fig. 3: Simulation results of a multiplier

product result to the previous product result instead of adding all 10 products (a-j) together in one summation so that it takes the advantage of the adders available in the FPGA so that the performance is increased. The summation gives the mantissa part of the result and the exponents of both the operand A and B are added together and to get the exponent part of the result. Finally the 106 mantissa bits are rounded to get 52 mantissa bit. The Fig. 3 shows the simulation results of multiplier. The signal opa (63:0) and opb (63:0) shows the 64 bit input operands A and B respectively in hexadecimal value. The out (63:0) signal shows the output of multiplication of operand A and B. The 64 bit double precision floating point format is shown in hexadecimal value. The signals invalid, inexact, overflow, underflow shows the exceptions conditions of multiplier.

DIVIDER

For high performance and Fault tolerance the divide operation is performed in long hand style, with one bit of the quotient calculated each clock cycle based on a comparison between the dividend and the divisor. The exponent bits and the mantissa bits of both operands are separated and the exponent bits of both the operands are compared with each other to find the larger exponent operand and to check whether the number is normalized or de normalized number. Mantissa of operand A is the dividend and mantissa of operand B is the divisor. If the dividend is greater than the divisor, the quotient bit is '1' and then the divisor is subtracted from the dividend, this difference is shifted one bit to the left and it becomes the dividend for the next clock cycle. If the dividend is less than the divisor, the dividend is shifted one bit to the left and then this shifted value becomes the dividend for the next clock cycle (Vangal *et al*., 2006). The divide operation takes 54 clock cycles to complete, as it takes 1 clock cycle to calculate each of the 54 bits (53 mantissa bits and 1 extra bit for rounding) of the quotient. The exponent for the divide operation is calculated from the exponent fields of operands A and B. The exponent of operand A is added to 1023 and then the exponent of operand B is subtracted from this sum. Finally, the exponent bits and the mantissa bits are rounded to obtain the 64 double precision floating point format. The rounding is carried out in different modes as explained in section II. Figure 4 shows the simulation results

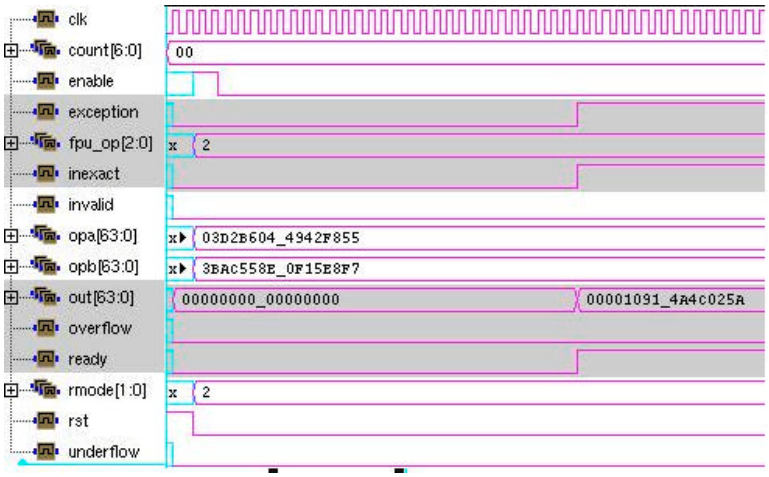

Fig. 4: Simulation results of a divider

Table 1: Cell area of logic components

Component	No. of cells	Cell area (μm²)
Adder	1341	50145
Subtractor	2639	69991
Multiplier	11298	333092
Divider	3988	103391
Total	19266	556619

Table 2: Power consumption of logic components

Component	Leakage power (nW)	Dynamic power (μW)
Adder	31.780	193.233
Subtractor	287.805	8963.453
Multiplier	1616.187	49579.587
Divider	405.410	12562.053
Total	2341.182	71298.328

of divider. The signal opa (63:0) and opb (63:0) shows the 64 bit input operands A and B, respectively in hexadecimal value. The out (63:0) signal shows the output of division of operand A and B. The 64 bit double precision floating point format is shown in hexadecimal value. The signals invalid, inexact, overflow, underflow shows the exceptions conditions of divider.

IMPLEMENTATION OF RESULTS

The double precision floating point arithmetic units are synthesized with 90 nm CMOS standard-cell technology library. The arithmetic units results in time slack of 1.23 nsec and Table 1 shows the cell area of each module and the total area of the design and Table 2 shows the power leakage power and the dynamic power consumed by different modules of arithmetic units. The power and the area consumed by the single precision floating point format (Graillat, 2009) is less than the double precision floating point format but when it is compared with the accuracy and performance it has the added advantage (Umar *et al*., 2004). The complete backend design of the arithmetic units is carried out using the cadence encounter.

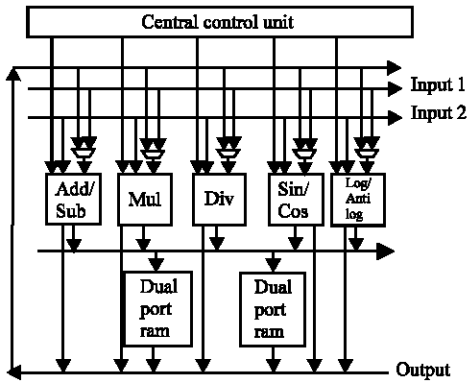

Fig. 5: Architecture of adaptive central processing unit for SoC applications

Figure 5 shows the proposed architecture with the central control unit, Floating point arithmetic units and Dual port RAMs. The SRAMS are used to store the values from the different computations so that the parallel operations are also possible in case of long chain of computations which increases the performance of the processor.

CONCLUSIONS AND FUTURE WORKS

Thus the high performance and increased accuracy than single precision floating point is obtained from the double precision floating point numbers and it is implemented in the Virtex 5 FPGA. Our future work is to design an adaptive programmable central core with the arithmetic operations that can adapt to various applications by changing the set of instructions in the control unit which finds various SoC applications such as in Active Structural Acoustic Control (ASAC), Active Noise Control (Anc), Active Vibration Control (AVC), Structural Health Monitoring (SHM) and Structural Health Control (SHC) etc.

REFERENCES

Astrom, K.J. and B. Wittenmark, 1995. Adaptive Control. 2nd Edn., Addison-Wesley, New York, USA.

Ciricescu, S., R. Essick, B. Lucas, P. May, K. Moat, J. Norris, M. Schuette and A. Saidi, 2003. The reconfigurable streaming vector processor (RSVP™). Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, December 3-5, 2003, Motorola Inc., Schaumburg, IL, USA., pp: 141-150.

Graillat, S., 2009. Accurate floating-point product and exponentiation. IEEE Trans. Comput., 58: 994-1000.

Hennessy, J.L. and D.A. Patterson, 1996. Computer Architecture a Quantitative Approach. 2nd Edn., Morgan Kaufmann Publishing Co., San Francisco, CA.

Louca, L., T.A. Cook and W.H. Johnson, 1996. Implementation of IEEE single precision floating point addition and multiplication on FPGAs. Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, April 17-19, 1996, Napa Valley, CA., pp: 107-116.

MSC, 2008. IEEE standard for floating-point arithmetic. Technical Reports, Microprocessor Standards Committee of the IEEE Computer Society, New York, USA., August 2008.

Standards Committee of the IEEE Computer Society, 1985. EEE standard for binary floating-point arithmetic. ANSI/IEEE Std. 754-1985 pp: 2355. http://kfe.fjfi.cvut.cz/~klimo/nm/ieee754.pdf

Umar, A., M.M. Al-Akaidi, S.A. Khan, S. Khattak and M. Assadullah, 2004. Performance evaluation of a hiperlan type 2 standard based on arithmetic formats. Inform. Technol. J., 3: 1-5.

Vangal, S.R., Y.V. Hoskote, N.Y. Borkar and A. Alvandpour, 2006. A 6.2-GFlops floating-point multiply-accumulator with conditional normalization. IEEE J. Solid-State Circuits, 41: 2314-2323.