International Journal of Scientific Engineering and Technology (ISSN : 2277-1581)
Volume No.2, Issue No.8, pp : 776-779 1 Aug. 2013

# BIST Architecture and Implementation of 64-Bit Double Precision Floating Point Multiplier Using VHDL

**Anurag Sharma**

M.Tech VLSI, Suresh Gyan Vihar University, Jaipur.

Email ID- anuraagsharmaa@yahoo.com

## ABSTRACT

***In this paper a 64-bit double precision floating point multiplier is implemented. A BIST test pattern generator for double precision multiplier is proposed. Linear feedback shift registers are used to generate the test pattern. A comparator is used to compare the output response and the expected response. For the circuit to work correctly the output response must be the same as the expected response. Xilinx ISE is used to synthesize the circuit and ModelSim is used for simulation purpose.***

Keywords : BIST, floating, ModelSim, multiplier

## I. INTRODUCTION

BIST is a technique to test a circuit through built-in hardware functions. In BIST a part of the circuit is used to test the circuit itself. BIST helps in the testing and verification of the circuit without the requirement of any hardware verification language. Every device is required to go through testing to ensure proper working. The devices that are produced nowadays consist of heterogeneous components like processors, memories etc. So it's not easy to test them. Also the design of the devices is mostly core based. Therefore the internal structures cannot be accessed. The technology used to manufacture devices is deep submicron technology. If there is a fault in such a device, it's very complicated to detect.

BIST provides solution to all these problems.

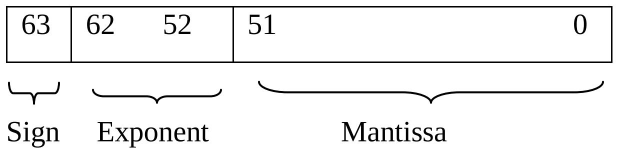
Floating point multiplier requires complex calculation because the operands used consists of 64-bits each. When the calculation involved is as large and complex as in the case of floating point multiplier then the proper working of the device cannot be ensured. The device might produce the output at the required time but the output cannot be verified. So even if there is a fault in the circuit of the device it might go unnoticed.

To solve this problem BIST architecture is used in the circuit. BIST architecture randomly generates a test pattern and applies it to the input of the device. Also the BIST architecture produces its own output, known as expected output, of those inputs. Both these outputs are compared with each other. To ensure proper working the outputs must match each other. If the outputs are same then a message is displayed regarding correct working of the floating point multiplier unit. In case the outputs are different, a message is displayed showing that the floating point multiplier is not working properly.

## II. 64-BIT DOUBLE PRECISION FLOATING POINT NUMBERS

Double precision floating point numbers are 64-bit binary numbers. The 64-bits are divided into 3 parts- sign, exponent and mantissa. The 52 least significant bits (LSBs) are used to represent the mantissa of the number. The next 11-bits are used to represent the exponent of the number. The most significant bit (MSB) of the number is used as a sign bit to represent the sign of the number.

- Sign bit '0' indicates positive number.
- Sign bit '1' indicates negative number.



## III. FLOATING POINT MULTIPLIER

Multiplication of two floating point numbers is a complex task and is carried out in a series of steps. Since a floating point number consists of 3 parts- sign, exponent and mantissa, calculations for all the parts are carried out separately.

## a) Calculation of Sign

The sign bit of the resultant is obtained by carrying out the ex-or operation of the sign bits of the two operands. Sign bit '0' represents a positive sign and sign bit '1' represents a negative sign.

## b) Calculation of Exponent:

The exponents of both the operands are represented in the IEEE 754 format, i.e., a bias of 1023 is added to both the exponents. To calculate the exponent of the resultant the bias of 1023 must be removed from the exponents. After removal of bias from the exponents, both are added to give the resultant exponent. This resultant exponent is in unbiased form. So to represent it in IEEE 754 format, it should be converted to the biased form by adding bias of 1023 to it.

## c) Calculation of Mantissa:

Mantissa calculation is the most complex part of floating point multiplication. A 64-bit number contains 52-bit mantissa. The resultant mantissa is calculated by multiplying the mantissas of both the operands. But before the multiplication is carried out, the mantissas of both the operands need to be normalized. Normalization is done in order to ensure that either of the numbers to be multiplied is not zero. If any one of the numbers is zero than the resultant will be zero. If both the numbers are zero than the resultant will be undefined or Not a Number (NaN). Normalization is done by adding a '1' as the MSB of the mantissa. By adding a '1' as MSB, the possibility of the number being a zero is eliminated. After normalization, the number of bits in the mantissa is increased by one, so the normalized mantissa contains 53-bits.

The next step after normalization is multiplication of the normalized mantissas. Two 53-bits mantissas are multiplied and a resultant of 106-bits is obtained. There are several different algorithms which can be used to carry out the multiplication of the mantissas. As the size of the mantissa is very large it is convenient to use an algorithm rather than multiplying directly. This 106-bits resultant cannot be stored directly into the output because of its size. The output mantissa must contain only 52-bits. To obtain 52-bits mantissa, normalization of the 106-bits resultant is carried out. In normalized form the MSB of the number must be 1. Therefore all the '0' bits before the first '1' bit are discarded. Now the mantissa is in normalized form with a '1' as MSB.

Now to extract the final 52-bits, denormalization is carried out. This is done because the mantissa that is finally stored in IEEE 754 format is not in the normalized form as the integer part of the output is by default 1. So the first bit of the number, i.e. 1, is discarded and the next 52 bits are stored as the mantissa of the output, also discarding the remaining least significant bits.

## EXCEPTIONS

There are 5 exceptional cases defined by the IEEE 754 representation. Whenever any of these cases occur, it should be represented by a flag.

**i) Invalid Operation:**

There are some operations which are invalid such as square root of a negative number. The result of an invalid operation is Not a Number (NaN). Whenever any invalid operation occurs, it should be represented by NaN. A NaN value is represented by setting all the bits of exponent to '1' alongwith a non-zero mantissa.

**ii) Inexact Operation:**

When the result of an operation cannot be defined exactly by the available range of output, inexact operation is signaled. This may occur due to a large exponent that cannot be defined by the available bits.

**iii) Division by zero:**

During a division operation if the divisor is zero then the result is not defined. This condition is signaled by a exception flag.

**iv) Overflow:**

When the result of an operation exceeds the range that can be represented by the available exponent bits, an overflow is said to have occurred. It should be signaled by an overflow flag.

**v) Underflow:** When a result is too small to be represented in the floating point format, it is known as underflow condition. It should be signaled by an underflow flag.

## IV. BIST ARCHITECTURE

The general BIST architecture consists of a test pattern generator, a device under test (DUT) and an optical response analyzer (ORA). The test pattern generator generates several patterns of inputs to test the device. The patterns generated by the test pattern generator are provided to the device under test (DUT) and its response to the test patterns is analyzed by the optical response analyzer (ORA). The optical response analyzer compares the output of the DUT with the expected output and provides the results accordingly.

The test patterns generator generates patterns using a gate-level representation of the design netlist. These patterns are stored in tester memory and scanned into the circuit using parallel scan chains. The chip input/output plays a major role in determining the number of scan chains. Other than this there are also some factors like the tester channels and on-chip routing congestion which can have an impact on the number of scan chains.

BIST adds an on-chip pattern generator and an on-chip result compressor to the circuit. The former helps in feeding the scan chains and latter helps in compressing the scanned out responses. All those responses are compressed into a final signature.

When a test pattern is applied to a circuit, the pattern data is scanned out firstly. After scanning, clock cycles are applied to it. The clock cycles can be one or more than one. Finally the resultant data is scanned out.

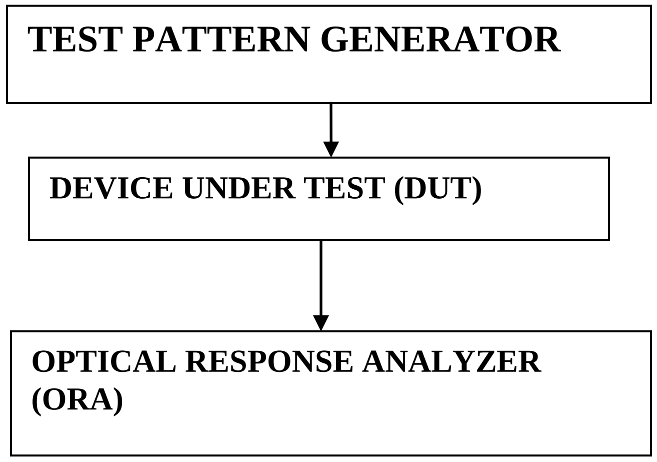


Fig 1 BIST architecture

## V. SYNTHESIS AND SIMULATION RESULTS

The BIST architecture is implemented on 64- bit double precision floating point multiplier using XILINX ISE . It has been mapped and routed on XILINX FPGA circuit of family Spartan 3E XC3S500E. ModelSim is used for simulation purpose. The implementation results are shown below. Fig.2 shows the RTL schematic of the BIST architecture. Fig.3 shows the RTL schematic of the circuit without implementing BIST architecture.

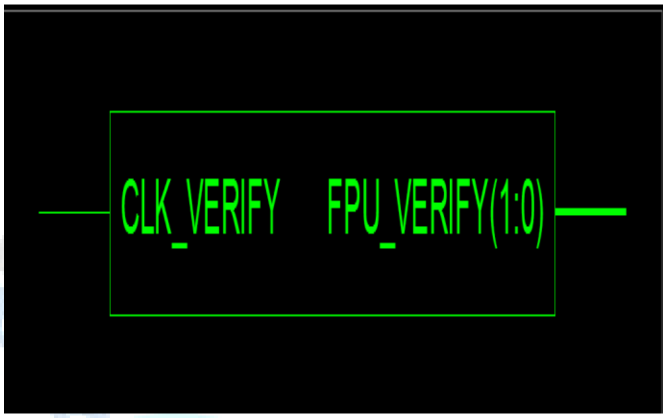


Fig.2 RTL schematic of BIST architecture

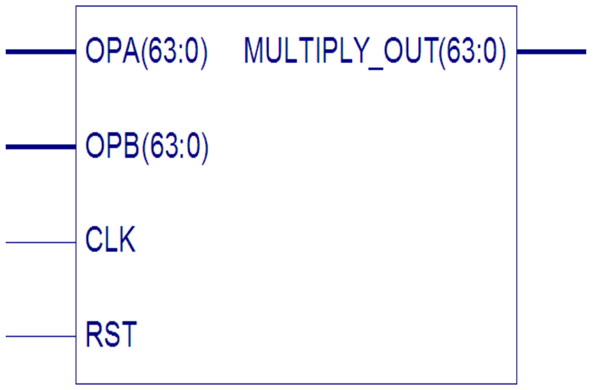
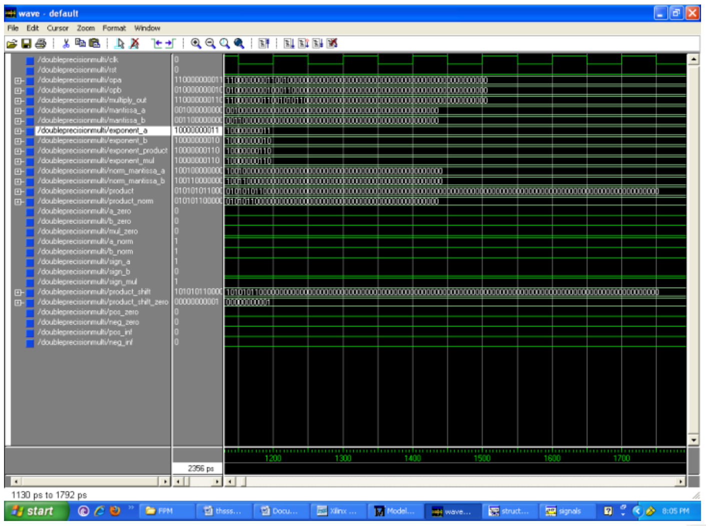


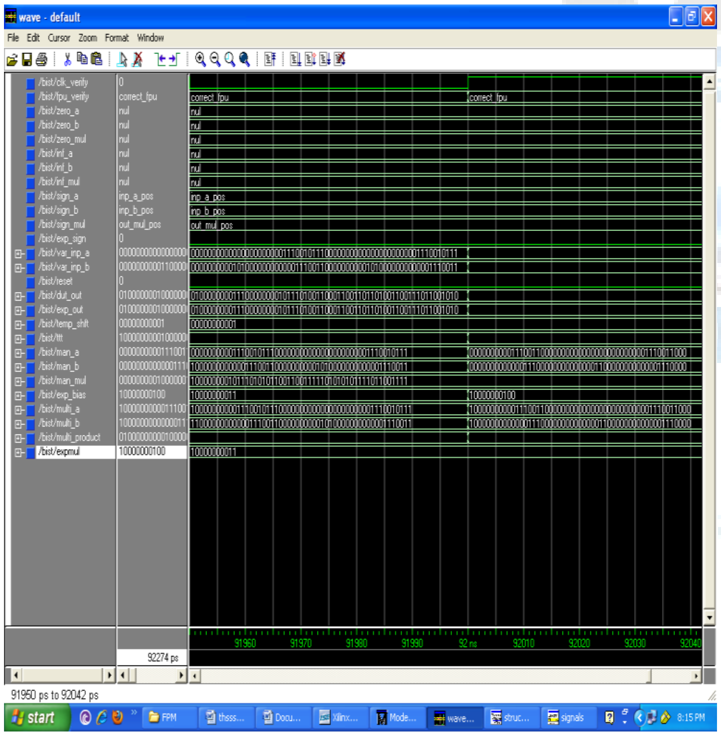
Fig.3 RTL of 64-bit floating point multiplier

Fig.4 shows the simulation waveform of floating point multiplier unit without implementing the BIST architecture and Fig.5 shows the simulation waveform of the complete unit with BIST architecture.



```
wave - default
/doubleprecisionmulti/clk                  0
/doubleprecisionmulti/rst                  0
/doubleprecisionmulti/opa                  110000000011
/doubleprecisionmulti/opb                  010000000010
/doubleprecisionmulti/multiply_out         110000000110
/doubleprecisionmulti/mantissa_a           001000000000
/doubleprecisionmulti/mantissa_b           001100000000
/doubleprecisionmulti/exponent_a           10000000011
/doubleprecisionmulti/exponent_b           10000000010
/doubleprecisionmulti/exponent_product     10000000110
/doubleprecisionmulti/exponent_mul         10000000110
/doubleprecisionmulti/norm_mantissa_a      100100000000
/doubleprecisionmulti/norm_mantissa_b      100110000000
/doubleprecisionmulti/product              010101011000
/doubleprecisionmulti/product_norm         010101100000
/doubleprecisionmulti/a_zero               0
/doubleprecisionmulti/b_zero               0
/doubleprecisionmulti/mul_zero             0
/doubleprecisionmulti/a_norm               1
/doubleprecisionmulti/b_norm               1
/doubleprecisionmulti/sign_a               1
/doubleprecisionmulti/sign_b               0
/doubleprecisionmulti/sign_mul             1
/doubleprecisionmulti/product_shift        101010110000
/doubleprecisionmulti/product_shift_zero   00000000001
/doubleprecisionmulti/pos_zero             0
/doubleprecisionmulti/neg_zero             0
/doubleprecisionmulti/pos_inf              0
/doubleprecisionmulti/neg_inf              0
2356 ps
1130 ps to 1792 ps
```

Fig.4 Simulation result of floating point multiplier



```
wave - default
/bist/clk_verify       0
/bist/fpu_verify       correct_fpu
/bist/zero_a           nul
/bist/zero_b           nul
/bist/zero_mul         nul
/bist/inf_a            nul
/bist/inf_b            nul
/bist/inf_mul          nul
/bist/sign_a           inp_a_pos
/bist/sign_b           inp_b_pos
/bist/sign_mul         out_mul_pos
/bist/exp_sign         0
/bist/var_inp_a        0000000000000000
/bist/var_inp_b        0000000000110000
/bist/reset            0
/bist/dut_out          0100000001000000
/bist/exp_out          0100000001000000
/bist/temp_shft        00000000001
/bist/ttt              1000000000100000
/bist/man_a            0000000000111001
/bist/man_b            0000000000000111
/bist/man_mul          0000000001000000
/bist/exp_bias         10000000100
/bist/multi_a          1000000000011100
/bist/multi_b          1000000000000011
/bist/multi_product    0100000000010000
/bist/expmul           10000000100
92274 ps
91950 ps to 92042 ps
```

Fig.5 Simulation result of BIST architecture

## VI. CONCLUSION

64-bit double precision floating point multiplier has been successfully implemented using VHDL. The code is synthesized using Xilinx ISE tool and Model Sim is used for simulation purpose. The output of the floating point multiplier has been verified with the help of BIST architecture. The output of the device under test (DUT) is the same as the expected output and the floating point multiplier unit is working correctly. Various flags have been introduced in the design to represent exceptional conditions such as positive infinity, negative infinity, zero etc.

## REFERENCES :

i. *D.Bakalistx. Kavousianos, H. T. Vergos D. Nikolos And G. Piq. Alexiou, "Low Power Built-In Self-Test Schemes for Array and Booth Multipliers" Vlsi Design 2001, Vol. 12, No. 3, Pp. 431-448*

ii. *B. Fagin and C. Renard, "Field Programmable Gate Arrays and Floating Point Arithmetic," IEEE Transactions on VLSI, vol. 2, no. 3, pp. 365–367, 1994.*

iii. *Zhiquan Zhang Zhiping Wen ; Lei Chen "BIST Approach for testing Embedded Memory Blocks in System-on-Chips" Testing and Diagnosis, 2009. ICTD 2009. IEEE Circuits and Systems International Conference 28-29 April 2009, pg no 1-3.*

iv. *N. Shirazi, A. Walters, and P.Athanas, "Quantitative Analysis of Floating Point Arithmetic on FPGA Based Custom Computing Machines,"Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM"95), pp.155–162, 1995.*

v. *Jutman, A.; Tsertov, A.; Ubar, R. "Calculation of LFSR Seed and Polynomial Pair for BIST Applications" Design and Diagnostics of Electronic Circuits and Systems, 2008. DDECS 2008. 11th IEEE Workshop16-18 April 2008.page no 1-4.*

vi. *Serdar S. Erdem ,Çetin K. Koç, "A Less Recursive Variant of Karatsuba-Ofman Algorithm for Multiplying Operands of Size a Power of Two", 16th IEEE Symposium on Computer Arithmetic, 2003*

vii. *Shianling Wu; Furukawa, H.; Boryau Sheu; Laung-Terng Wang; Hao-Jan Chao; Lizhen Yu; Xiaoqing Wen; Murakami, M. "Practical Challenges in Logic BIST Implementation – Case Studies" Asian Test*

viii. *Rohit Sreerama, Paidi Satish, K Neelima. "An Algorithm for variable precision based floating point multiplication", proc International Conference on Advances in Information Technology and Mobile Communication, AIM 2012, page no-238-242*

ix. *John G. Proakis and Dimitris G. Manolakis (1996), "Digital Signal Processing: Principles,.Algorithms and Applications", Third Edition.*

x. *Xilinx Application Note by Peter Alfke "Efficient Shift Registers, Lfsr Counters, And Long Pseudo-Random Sequence Generators" July 7,1996 (Version 1.1).*