ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 1, Issue 2, November 2012

An Efficient Implementation of High Performance Floating Point Adder / Subtractor

Addanki Purna Ramesh, Siva jampani
Department of ECE, Sri Vasavi Engineering College, Pedatadepalli, Tadepalligudem, India

Abstract- ***Floating Point (FP) arithmetic is widely used in large set of scientific and signal processing computation. Adder / subtractor are one of the common arithmetic operations in these computations. This paper implements an efficient high performance floating-point adder/subtractor. The whole design was captured in Verilog Hardware description language (HDL) and targeted on a Virtex-6 Xc6vlx75t-3ff484 Field Programmable Gate Array (FPGA), with optimal area and high performance. In addition, the proposed design is in compliant with IEEE-754 format and handles overflow, underflow and various exception conditions. This design operates at 331.939 MHz frequency.***

Index Terms- **Adder/ Subtractor, FPGA, IEEE-754, Verilog, Floating Point, Double Precession.**

I. INTRODUCTION

The advantage of floating-point representation over fixed-point and integer representation is that it can support a much wider range of values. For many signal processing, and graphics applications, it is acceptable to trade off some accuracy (in the least significant bit positions) for faster and better implementations. A Floating point describes a method of representing real numbers in a way that can support a wide range of values. . The base for the scaling is normally 2, 10 or 16. Double precision floating point is an IEEE 754 standard for encoding binary or decimal floating point numbers in 64 bits. The IEEE 754 standard defines a double as 1 bit for sign,11 bits for Exponent,53 bits (52 explicitly stored) for Significant.The Double precision floating point format is shown in figure 1.

1 sign bit	11-bits exponent	52-bits mantissa

Fig .1 Double Precision Floating-Point Format

II. IMPLEMENTATION

The double precision floating point adder/subtractor performs addition, subtraction operations. The Black box view of double precision floating point adder and subtractor is shown in figure2 and 3 respectively.

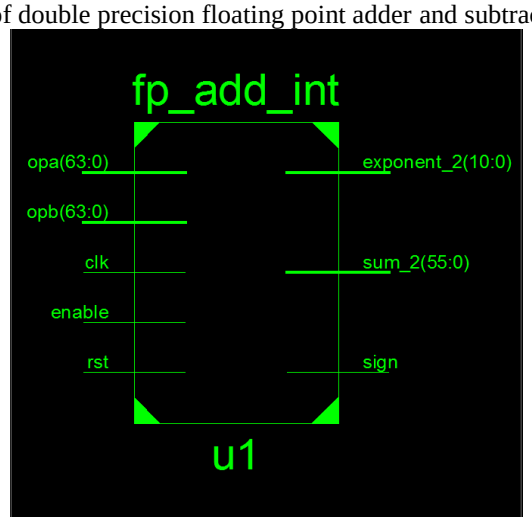

Fig .2 Black Box View Of Double Precision Floating Point Adder

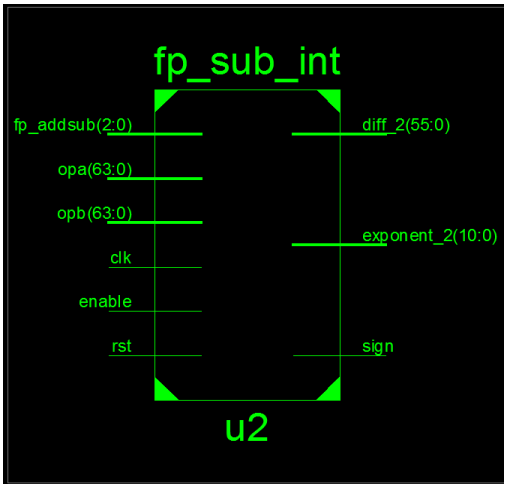

Fig .3 Black Box View Of Double Precision Floating Point Subtractor

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 1, Issue 2, November 2012

Table 1&2 shows the Rounding Modes selected for various bit combinations of rmode and Operations selected for various bit combinations of Fpu_op.

Table 1: Rounding Modes Selected For Various Bit Combinations of Rmode

Bit combination	Rounding Mode
00	round_nearest_even
01	round_to_zero
10	round_up
11	round_down

Table 2: Operations Selected For Various Bit Combinations of Fpu_Op

Bit combination	Operation
000	Addition
001	Subtraction

A. Addition

The input operands are separated into their mantissa and exponent components, and the larger operand goes into "mantissa large" and "exponent large", with the smaller operand populating "mantissa small" and "exponent small". The comparison of the operands to determine which is larger only compares the exponents of the two operands, so in fact, if the exponents are equal, the smaller operand might populate the "mantissa large" and "exponent large" registers. This is not an issue because the reason the operands are compared is to find the operand with the larger exponent, so that the mantissa of the operand with the smaller exponent can be right shifted before performing the addition. If the exponents are equal, the mantissas are added without shifting.

B. Subtraction

The input operands are separated into their mantissa and exponent components, and the larger operand goes into "mantissa large" and "exponent large", with the smaller operand populating "mantissa small" and "exponent small". Subtraction is similar to addition in that you need to calculate the difference in the exponents between the two operands, and then shift the mantissa of the smaller exponent to the right before subtracting. The definition of the subtraction operation is to take the number in operand B and subtract it from operand A. However, to make the operation easier, 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 will be inverted of the result.

C. Rounding

Rounding takes a number regarded as infinitely precise and, if necessary, modifies it to fit in the destination's format while signalling the inexact exception, underflow, or overflow when appropriate. Except where stated otherwise, every operation shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that result according to one of the attributes in this clause.

An infinitely precise result with magnitude at least $b\, e^{max}\, (b - \tfrac{1}{2}\, b\, 1{-}p)$ shall round to ∞ with no change in sign; here e^{max} and p are determined by the destination format as

- RoundTiesToEven, the floating-point number nearest to the infinitely precise result shall be delivered; if the two nearest floating-point numbers bracketing an unrepresentable infinitely precise result are equally near, the one with an even least significant digit shall be delivered
- RoundTiesToAway, the floating-point number nearest to the infinitely precise result shall be delivered; if the two nearest floating-point numbers bracketing an unrepresentable infinitely precise result are equally near, the one with larger magnitude shall be delivered.

Three other user-selectable rounding-direction attributes are defined, the directed rounding attributes

- RoundTowardPositive, the result shall be the format's floating-point number (possibly $+\infty$) closest to and no less than the infinitely precise result
- RoundTowardNegative, the result shall be the format's floating-point number (possibly $-\infty$) closest to and no greater than the infinitely precise result
- RoundTowardZero, the result shall be the format's floating-point number closest to and no greater in magnitude than the infinitely precise result.
-

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 1, Issue 2, November 2012

Fpu_round module performs the rounding operation. The black box view is shown in figure 4.

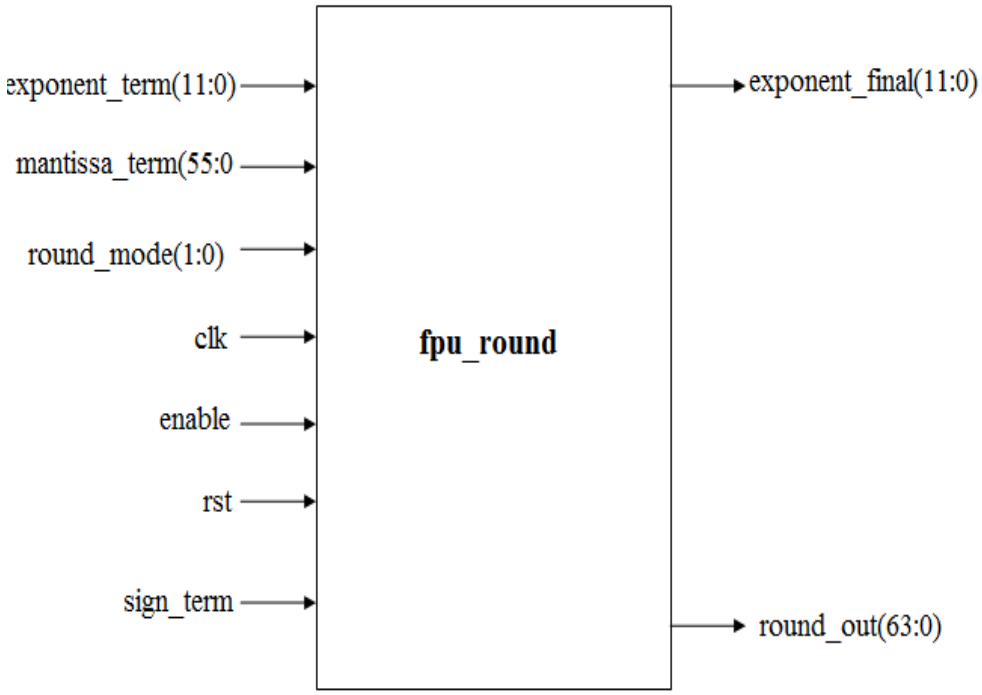

Fig.4 Black Box View Of Rounding Module

The inputs to the (fpu_round) module from the previous stage (addition, subtraction, multiply, or divide) are sign (1 bit), mantissa_term (56 bits), and exponent_term (12 bits). The mantissa_term includes an extra '0' bit as the MSB, and two extra remainder bits as LSB's, and in the middle are the leading '1' and 52 mantissa bits. The exponent term has an extra '0' bit as the MSB so that an overflow from the highest exponent (2047) will be caught; if there were only 11 bits in the register, a rollover would result in a value of 0 in the exponent field, and the final result of the fpu operation would be incorrect. There are 4 possible rounding modes. Round to nearest (code = 00), round to zero (code = 01), round to positive infinity (code = 10), and round to negative infinity (code = 11).

For round to nearest mode, if the first extra remainder bit is a '1', and the LSB of the mantissa is a '1', then this will trigger rounding. To perform rounding, the mantissa_term is added to the signal (rounding_amount). The signal rounding_amount has a '1' in the bit space that lines up with the LSB of the 52-bit mantissa field. This '1' in rounding_amount lines up with the 2 bit of the register (mantissa_term); mantissa_term has bits numbered 55 to 0. Bits 1 and 0 of the register (mantissa_term) are the extra remainder bits, and these don't appear in the final mantissa that is output from the top level module, fpu_double.

For round to zero modes, no rounding is performed, unless the output is positive or negative infinity. This is due to how each operation is performed. For multiply and divide, the remainder is left off of the mantissa, and so in essence, the operation is already rounding to zero even before the result of the operation is passed to the rounding module. The same occurs with add and subtract, in that any leftover bits that form the remainder are left out of the mantissa. If the output is positive or negative infinity, then in round to zero mode, the final output will be the largest positive or negative number, respectively.

For round to positive infinity mode, the two extra remainder bits are checked, and if there is a '1' in both bit, and the sign bit is '0', then the rounding amount will be added to the mantissa_term, and this new amount will be the final mantissa. Likewise, for round to negative infinity mode, the two extra remainder bits are checked, and if there is a '1' in either bit, and the sign bit is '1', then the rounding amount will be added to the mantissa_term, and this new amount will be the final mantissa. The output from the fpu_round module is a 64-bit value in the round_out register. This is passed to the exceptions module.

D. Exceptions

Exception is an event that occurs when an operation on some particular operands has no outcome suitable for a reasonable application. That operation might signal one or more exceptions by invoking the default or, if explicitly requested, a language-defined alternate handling. Note that event, exception, and signal are defined in

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 1, Issue 2, November 2012

diverse ways in different programming environments. The exceptions module is shown in figure5.All of the special cases are checked for, and if they are found, the appropriate output is created, and the individual output signals of underflow, overflow, inexact, exception, and invalid will be asserted if the conditions for each case exist.

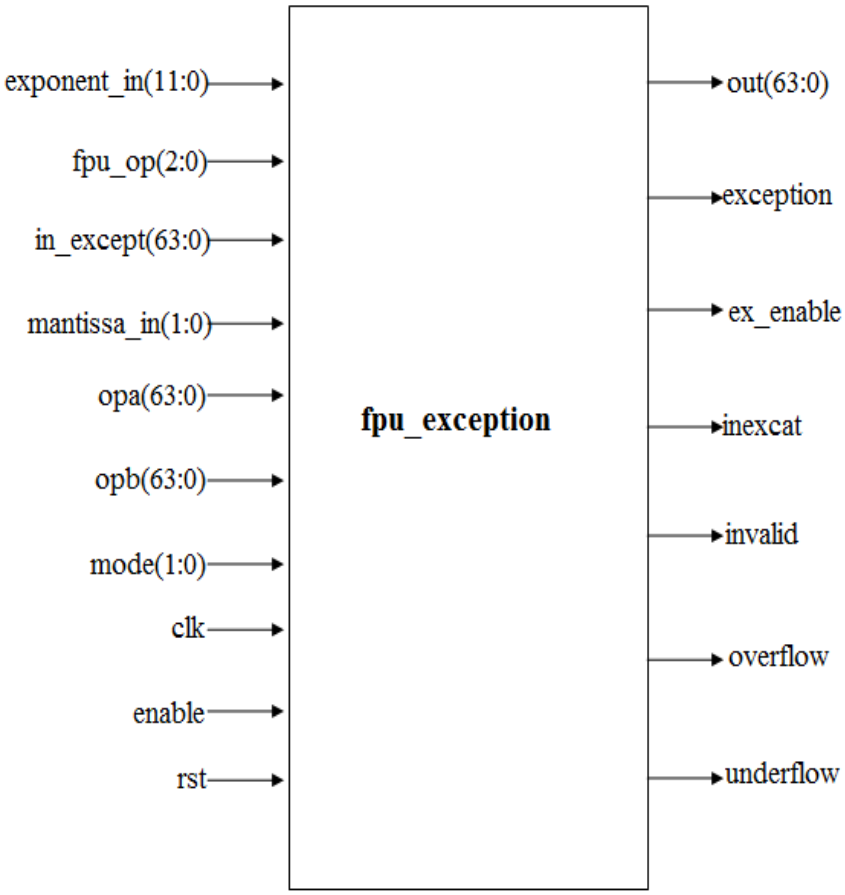

Fig. 5 Black box view of Exception Module

The exception signal will be asserted for the following special cases.

1. Divide by 0: Result is infinity, positive or negative, depending on sign of operand A
2. Divide 0 by 0: Result is SNaN, and the invalid signal will be asserted
3. Divide infinity by infinity: Result is SNaN, and the invalid signal will be asserted
4. Divide by infinity: Result is 0, positive or negative, depending on the sign of operand A the underflow signal will be asserted
5. Multiply 0 by infinity: Result is SNaN, and the invalid signal will be asserted
6. Add, subtract, multiply, or divide overflow: Result is infinity, and the overflow signal will be asserted
7. Add, subtract, multiply, or divide underflow: Result is 0, and the underflow signal will be asserted
8. Add positive infinity with negative infinity: Result is SNaN, and the invalid signal will be asserted
9. Subtract positive infinity from positive infinity: Result is SNaN, and the invalid signal will be asserted
10. Subtract negative infinity from negative infinity: Result is SNaN, and the invalid signal will be asserted
11. One or both inputs are QNaN: Output is QNaN
12. One or both inputs are SNaN: Output is QNaN, and the invalid signal will be asserted
13. If either of the two remainder bits is '1': Inexact signal is asserted.

If the output is positive infinity, and the rounding mode is round to zero or round to negative infinity, then the output will be rounded down to the largest positive number (exponent = 2046 and mantissa is all 1's). Likewise, if the output is negative infinity, and the rounding mode is round to zero or round to positive infinity, then the output will be rounded down to the largest negative number. The rounding of infinity occurs in the exceptions module, not in the rounding module. QNaN is defined as Quiet Not a Number. SNaN is defined as Signaling

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 1, Issue 2, November 2012

Not a Number. If either input is a SNaN, then the operation is invalid. The output in that case will be a QNaN. For all other invalid operations, the output will be a SNaN. If either input is a QNaN, the operation will not be performed, and the output will be a QNaN. The output in that case will be the same QNaN as the input QNaN. If both inputs are QNaNs, the output will be the QNaN in operand A.

III. SIMULATION RESULTS

The simulation results of double precision floating point adder/subtractor (Addition, Subtraction) are shown in figures 6, 7, 8, and 9 respectively. Figure 9 shows the Device utilization summery for the floating point adder/subtractor.

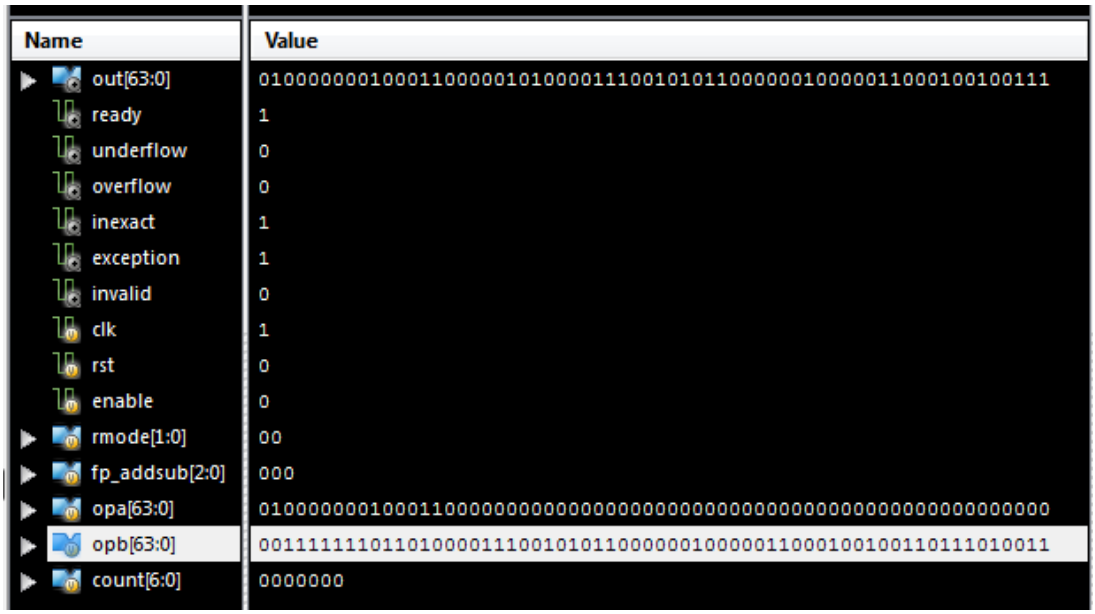

Fig. 6 Simulation Results for Floating Point Addition

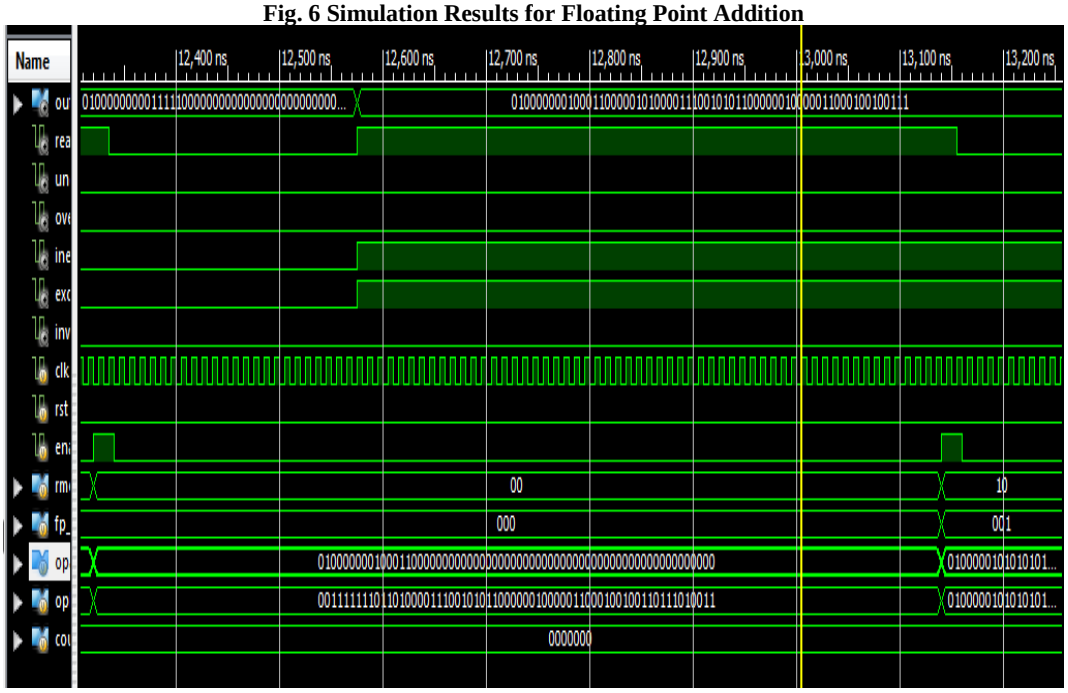

Fig. 7 Simulation Results for Floating Point Addition

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 1, Issue 2, November 2012

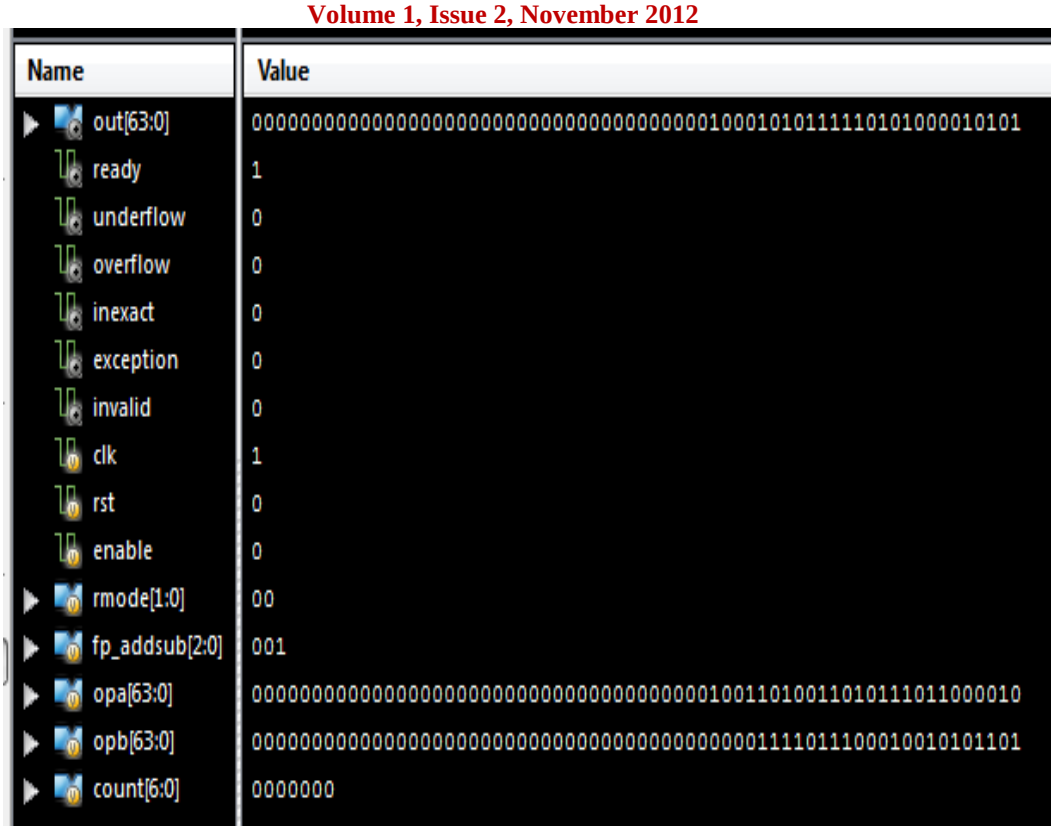

Fig. 8 Simulation Results for Floating Point Subtraction

Logic Utilization	Used	Available
Number of Slice Registers	2423	93120
Number of Slice LUTs	2367	46560
Number of fully used LUT-FF pairs	1447	3343
Number of bonded IOBs	206	240
Number of BUFG/BUFGCTRLs	1	32

Fig . 9 Device Utilization Summery For Floating Point Adder/Subtractor

IV. CONCLUSION

The double precision floating point adder/subtractor (addition, subtraction) supports the IEEE 754 binary interchange format. The whole design was captured in Verilog Hardware description language (HDL) and targeted on a Xilinx Virtex-6 xc6vlx75t-3ff484 FPGA. It achieved 331.939 MFLOPs.

REFERENCES

[1] Manish Kumar Jaiswal and Ray C.C. Cheung, "High Performance FPGA Implementation of Double Precision Floating Point Adder/Subtractor," in International Journal of Hybrid Information Technology, Vol. 4, No. 4 October, 2011.

[2] A. Malik and S.-B. KO, "A Study on the Floating-Point Adder in FPGAs," in Canadian Conference on Electrical and Computer Engineering (CCECE-06), May 2006, pp. 86–89.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 1, Issue 2, November 2012

[3] K. Hemmert and K. Underwood, "Open Source High Performance Floating-Point Modules," in 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM-06), April 2006, pp. 349–350.

[4] K. Underwood and K. Hemmert, "Closing the gap: CPU and FPGA trends in sustainable floating-point BLAS performance," in 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM- 2004), April 2004, pp. 219–228.

[5] P. Belanovic and M. Leeser, "A Library of Parameterized Floating-Point Modules and Their Use," in 12th International Conference on Field-Programmable Logic and Applications (FPL-02). London, UK: Springer-Verlag, Sep. 2002, pp. 657–666.

[6] V. Oklobdzija, "An algorithmic and novel design of a leading zero detector circuit: comparison with logic synthesis," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 2, no. 1, pp. 124–128, Mar 1994.

[7] Sateesh Reddy and Vinit T Kanojia, " Unified Reconfigurable Floating-Point Pipelined Architecture," in (IJAEST) International Journal Of Advanced Engineering Science And Technologies, Vol No. 7, Issue No. 2, 2011, 271 – 275.

[8] Dhiraj Sangwan & Mahesh K. Yadav," Design and Implementation of Adder/Subtractor and Multiplication Units for Floating-Point Arithmetic," in International Journal of Electronics Engineering, 2010, pp. 197-203.

[9] Per Karlstr¨om, Andreas Ehliar and Dake Liu, "High Performance, Low Latency FPGA based Floating Point Adder and Multiplier Units in a Virtex 4".