Out Of Order Floating Point Coprocessor For RISC V ISA

Vinayak Patil, Aneesh Raveendran, Sobha P M, A David Selvakumar, Vivian D
Centre for Development of Advanced Computing, Bangalore, INDIA
{ *vinayakp,raneesh, sobhapm, david, vivand} @ cdac.in*

***Abstract*— Floating point operations are important and essential part of many scientific and engineering applications. Floating point coprocessor (FPU) performs operations like addition, subtraction, division, square root, multiplication, fused multiply and accumulate and compare. Floating point operations are part of ARM, MIPS, and RISC-V etc. instruction sets. The FPU can be a part of hardware or be implemented in software. This paper details an architectural exploration for floating point coprocessor enabled with RISC-V [1] floating point instructions. The Floating unit that has been designed with RISC-V floating point instructions is fully compatible with IEEE 754-2008 standard as well. The floating point co-processor is capable of handling both single and double precision floating point data operands in out-of-order for execution and in-order commit. The front end of the floating point processor accepts three data operands, rounding mode and associated Opcode fields for decoding. Each floating point operation is tagged with an instruction token for in-order completion and commit. The output of the floating point unit is tagged with either single or double precision results along with floating point exceptions, if any, based on the RISC-V instruction set. The coprocessor for floating point integrates with integer pipeline. The proposed architecture for floating point coprocessor with out-of-order execution, in-order commit and completion/retire has been synthesized, tested and verified on Xilinx Virtex 6 xc6vlx550t-2ff1759 FPGA. A system frequency of 240MHz for single precision floating point and 180 MHz double precision floating point operations has been observed on FPGA.**

***Index Terms*—IEEE 754 floating point standard, floating point co-processor, RISC processor, RISC-V instruction set, Microprocessor.**

I. INTRODUCTION

Many scientific and engineering applications require floating point arithmetic. Application varies from climate modeling, supernova simulations, electromagnetic scattering theory, Computational Geometry and Grid Generation [2] to image processing, FFT calculation [3], matrix arithmetic, Eigen value calculation [4], Jacobi solver [5], and so on. To support and accelerate these applications, high performance computing devices which can produce high throughput are required. Floating point coprocessors drastically increase the performance of these high computation applications. In most modern general purpose computer architectures, floating point coprocessor is integrated within processor chip like in ARM, MIPS etc. FPU acts a kind of accelerator and work in parallel with the integer pipeline. It will offload large computational, high latency floating point instruction from main processor. If these instructions were executed by main processor, then it will slow down main processor as operations like floating point division, square root, multiply and accumulate takes large computation and long latency. Further out of order execution floating point coprocessor unit helps to speed up whole chip. The standardized methods to represent floating point numbers have been instituted by IEEE 754 standard through which the floating point operation can be carried out with modest storage requirements [6].

Three basic components in IEEE 754 are sign (S), exponent (E) and mantissa (M). Bit width and bit allocations are as shown in TABLE I.

TABLE I : *FLOATING POINT BIT REPRESENTATION*

Precision	Sign	Exponent	Mantissa	Bias
Single	1[31]	8[30:23]	23[22:0]	127
Double	1[63]	11[62:52]	52[51:0]	1023

Number = $(-1)^S*(H.M)^E$ where H = hidden bit

IEEE 754 2008 standard supports all floating point operations. It handles all special input like sNaN, qNaN, +Infinity, –Infinity positive zero and negative zero. It handles five exceptions namely overflow, underflow, invalid, inexact and division by zero. It supports five different rounding modes.

II. RELATED WORKS

High computational intensive algorithms and simulations such as Monte Carlo simulations [7] [8] [9] finite element analysis [10] [11], Eigen value calculation [4], Jacobi solver analysis [5] requires large amount of floating point operations. Many embedded processors, especially older designs, do not have hardware support for floating-point operations. In the absence of an FPU as hardware, many FPU functions can be emulated through software libraries [12], which save the added hardware cost of an FPU but are significantly slower in speed. In order to improve the performance, hardware implementation of floating point unit is needed.

Up to now many efforts has been taken to design of floating point coprocessor. But many of design do not have wide precision support. Some designs support single precision [13] [14]. Some designs have limited arithmetic operation support like only MAC [15], or only addition, subtraction, division [14].

978-1-4799-1743-3/15/$31.00 ©2015 IEEE

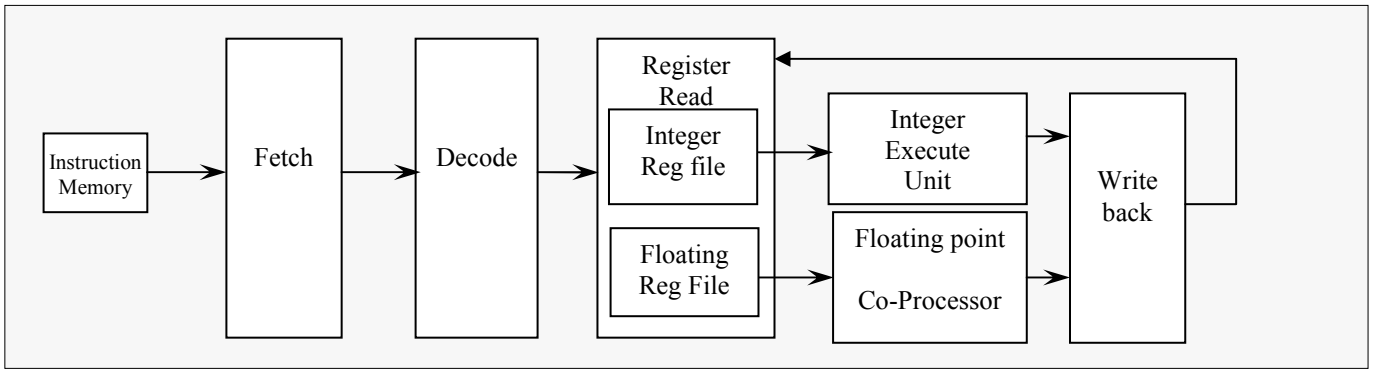

Fig. 1. Top Level Architecture For RISC V Processor

FPU with SPARC architecture has been implemented [16], which considers only addition, multiplication and division operations. Some efforts have been made for FPU with RISC architecture [17] but this paper considers only single precision operations, major operation like fused multiply and accumulate is missing, with system frequency of 70 Mhz.

Proposed paper implements floating point coprocessor which supports both single and double precision. It also has wide number of arithmetic operations like addition, subtraction, division, square root, multiplication, fused multiply and accumulates and compare. It is fully compatible with IEEE754 standard. One of the key benefits of this implementation is that it is enabled with an open source RISC V ISA [1].

III. Instruction Set Analysis

Before the selection of the RISC-V instruction set for implementation, an analysis on floating point instructions in the MIPS, ARM, MIPS, and RISC-V has been done.

TABLE II : *ISA Analysis:-RISC-V, ARM and MIPS*

Design Philosophy	RISC-V [1]	ARM [18]	MIPS[19]
ISA encoding scheme	Fixed and variable	Fixed	Fixed
Instruction size	16/32/64/ 128-bit	16/32/ 64-bit	32/64-bit
General purpose registers	32	31	31
Floating point registers	32	31	31
Is all rounding mode supported	Yes	Yes	Yes
Single precision floating point operations	26	33	15
Double precision floating point operations	26	29	15

IV. Pipeline Organization of RISC-V Processor

A. RISC Processor Architecture

Fig. 1. shows the top level architecture of the RISC processor. Although in this section whole processor environment has been discussed, this paper focuses on floating point co-processor implementation. Fig. 1. also shows how FPU coprocessor is going to integrate with main processor. The fetch unit fetches the instructions from the program memory based on the program counter value. Decoder decodes the instructions and passes to the register select unit. If the current instruction is integer related instruction, then it passes to the integer pipeline and if the instruction is related the floating point, then passes to the floating point co-processor pipeline. For floating point instructions, integer decoder just do partial decoding, full decoding of those instruction will takes place in FPU coprocessor. The data from register select module is given to the floating point and integer modules. The floating point co-processor will accepts input signals as three input operands (either single or double precision), opcode and round mode. It decodes instructions and executes them. Final result is written back to the registers through write back unit.

B. Out Of Order FPU Coprocessor

Fig. 2. shows the top level architecture of an in order issue out-of-order execution pipelined FPU co- processor.

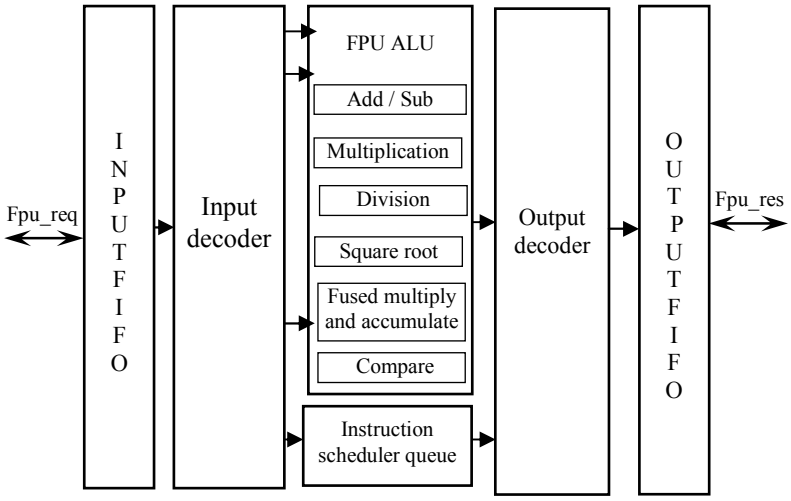

Fig. 2. Architecture of Out-of-order FPU Co-Processor

In order FPU cannot utilize full capability of design. In this type of design many cycles gets waste during stall of pipeline. To make use of these waste cycles, out of order FPU have been implemented. It receives in order instructions, performs out of order execution and in order commit.

a) Input FIFO

The floating point unit accepts FPU request packet and stores it into input FIFO which is further read by input decoder. Fig. 3. shows packet format. Packet contains three input operands, opcode, F7, F5, F3 and rounding mode.

119….88	87….56	55….24	23…18	17..11	10..6	5..3	2...0
Op1(SP/DP)	Op2(SP/DP)	Op3(SP/DP)	Opcode	F7	F5	F3	RM

Fig. 3. FPU request packet

Opcode, F7, F5 and F3 are specific for each floating point instruction. Rounding mode will be variable. It can have any value as mention in IEEE754 standard. Packet has three operands as op1, op2 and op3. These can be either single precision or double precision.

b) Input decoder

Input decoder in floating point pipeline reads packet from input FIFO. It decodes the operations depending on opcode, F7, F5 and F3 fields. If any of these fields is not matching with legal opcode or legal F7, F5, F3 for floating point instruction then decoder will give that as illegal instruction. If fields are matching then decoder enable corresponding FPU operation and it will pass op1, op2 and op3 data to FPU ALU. FPU coprocessor keeps the track of the instruction by putting token into an instruction scheduler queue.

On each clock cycle an instruction will be issued till instruction scheduler queue is not full. Since, latency for each operation is different; depth of this queue should be equal to maximum clock latency of all operations to operate without any bubble in pipeline. As instructions are issued on every clock cycle, they can execute in parallel and out of order. They may finish even out of order but they cannot commit out of order.

c) Floating Point ALU architecture

Fig. 4. shows the micro-architecture of the floating point co-processor ALU. There are six independent pipelines as shown in figure. These pipelines are floating point add/sub, divider, multiplier, square root, multiply and accumulate and compare. Each pipeline has its own clock latency. It receives three operands (either single precision or double precision) and rounding mode as input. It gives output of arithmetic operation and five exception as overflow, underflow, invalid, inexact and divbyzero. Opcode decoder will decode instruction and enables the required module. Internal module will perform operation on input data. Each module will take specific clock latency and gives output along with exceptions. There is special input detector which checks for special inputs like snan, qnan, infinity and zero. If any special input is there then instruction will get bypassed directly. It will give output for these special inputs as per IEEE754.

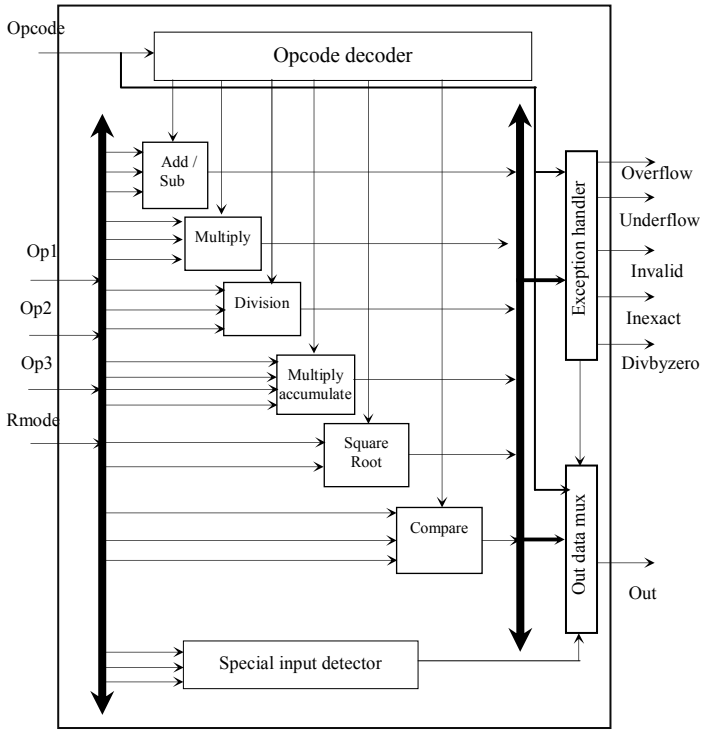

Fig. 4. Floating Point ALU Architecture

d) Output decoder

Output decoder logic only commits oldest instruction from pipeline. This ensures in order commit. It checks token from instruction scheduler queue and fetches oldest token and commits instruction only if that token matches with completed instruction. Output decoder takes corresponding output data with exception, forms packet and send it to output FIFO. Appropriate exceptions for the each operation are raised and processed according to the RISC-V instruction set. The floating point co-processor has 5 exception signals to raise exceptions in the operation. The exceptions are raised according to the IEEE 754 standard.

e) Output FIFO

Decoder stores output data along with exceptions in output FIFO. It sends this packet to write back stage of integer pipeline.

V. MICRO-ARCHITECTURE OF INTERNAL FPU ALU

A. Adder/subtractor

Architecture for floating point adder cum subtractor unit is as shown in Fig. 5. FPU adder has five stages. So clock latency for adder is 5 clocks. This is continuous pipelined structure. It receives 4 inputs. Two IEEE format operands, 3 bit round_mode for run time configuration, 1 bit operation signal as control signal for addition or subtraction. It gives addition result as an output in IEEE 754 format. Along with this, it will give four more exceptional signals namely overflow, underflow, invalid and inexact. Main functionality is divided

into three parts, as pre-processing unit, binary addition subtraction unit and post processing unit. Pre-processing stage performs unpacking of input operands into sign, mantissa and exponent. For addition or subtraction exponent of both operands should be same. Exponent unit calculates effective exponent and exponent difference. Larger operand exponent is considered as an output exponent.

Swap unit decides smaller and larger operand. Smaller operand's mantissa undergoes left shift by exponent difference value to make exponent of both operands same. Effective operation (EOP) is calculated by xor-ing sign of both operands (S1 and S2) and operation bit. Sign of final result is calculated by using EOP and larger operand. Three bits will be added to smaller mantissa as guard, round and sticky. These are called rounding bits which are useful at rounding. Smaller mantissa will through left shift by exponent difference value. This shift goes through rounding bits. Trailing zero detector (TZD) will calculate number of trailing zeros for smaller mantissa. So as per left shift and number of trailing zeros, sticky bit will update. If any non-zero bit losses out in shift then sticky bit will be set otherwise it will be reset. Binary addition subtraction unit takes larger mantissa and shifted smaller mantissa as inputs. It does binary addition or subtraction on both operand and gives result and carry to post processing unit.

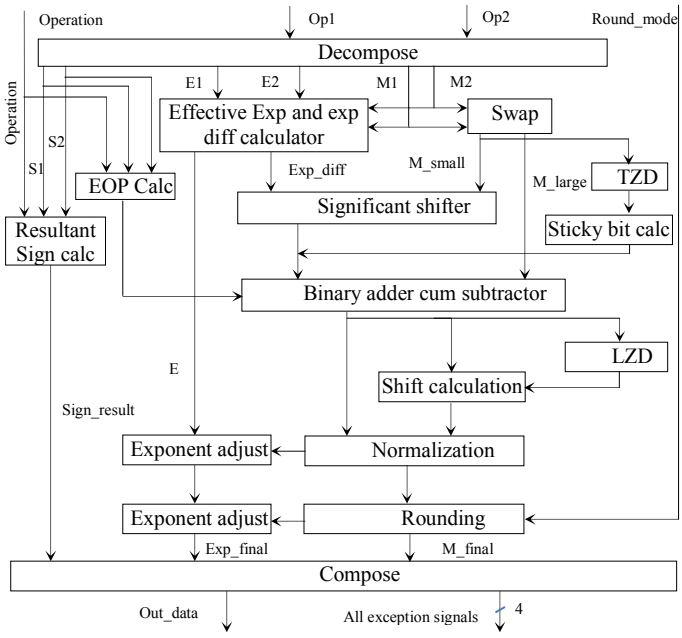

Fig. 5. Architecture for Floating Point Adder / Subtractor

Result of addition or subtraction may be de-normalized or it may have a carry bit generated. It has to be normalized if possible. Leading zeros detector (LZD) will provide number of leading zeros present in result. Shift calculation unit calculates amount and type of shift required. Leading zeros is amount of left shift needed. If there is carry from operation there will be one bit right shift. Normalization unit normalizes result according to type and amount of shift given by shift calculation unit. According to shift, there is adjustment in exponent. Normalized result will go through rounding as per rounding mode provided. If there is carry from rounding then again result is shifted and exponent will be updated. Rounded result, updated exponent and sign of result will be provided to compose unit which convert output to IEEE 754 standard format and also gives all exception signal.

B. Divider

Architecture for floating point divider unit is as shown in Fig. 6. FPU divider has 31 stages. So clock latency for divider is 31 clocks. This is continuous pipelined structure. It receives three inputs. Two IEEE format operands and three bit round_mode for run time configuration. It gives out_data as division result; in IEEE format. Along with this it will give five more exceptional signals as overflow, underflow, invalid, inexact and divbyzero. Design has three main stages as pre-processing unit, non-restoring binary divider and post-processing unit.

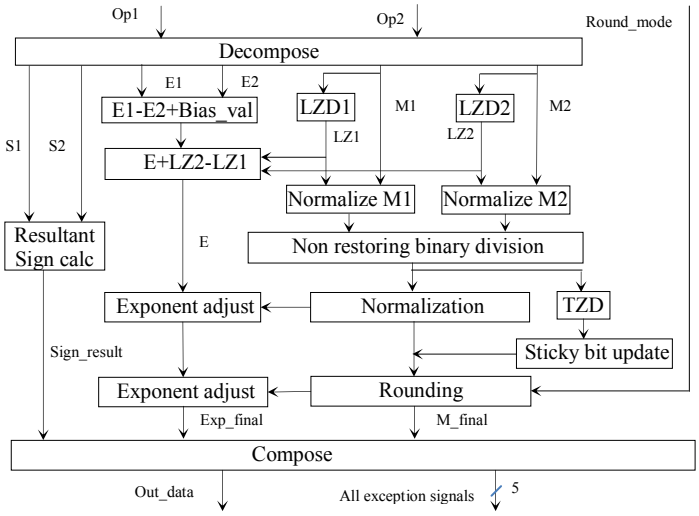

Fig. 6. Architecture for Floating Point Divider

This is continuous pipelined structure. It receives three inputs. Two IEEE format operands and three bit round_mode for run time configuration. It gives out_data as division result; in IEEE format. Along with this it will give five more exceptional signals as overflow, underflow, invalid, inexact and divbyzero. Design has three main stages as pre-processing unit, non-restoring binary divider and post-processing unit.

In pre-processing, decompose unit unpacks two IEEE-754 encoded numbers into corresponding sign bits, biased binary exponents, and mantissa. Sign of final result is calculated by using sign of both operands. Resultant exponent will be calculated by E1 – E2. Bias has to be added to this value (E1-E2+Bias_value). If there are leading zeros in mantissa, then normalization of mantissa is required. This will increase precision in result. So leading zero detectors will calculate number of leading zeros in mantissa part of both operands. Then, there will be left shift for normalization. According to this shift, resultant exponent will be updated (E+LZ1-LZ2). Normalized mantissas will go through non-restoring binary division (NRBD) algorithm [20]. It gives one bit of quotient on every clock cycle. This result may not be normalized.

Maximum of 1 bit of left shift is required for normalization. If results are too tiny then there is need of right shift to make exponent in specified limit. Trailing zeros detector will check number of trailing zeros and if right shift is required then sticky bit will be updated. After this normalization, it will undergo rounding as per rounding mode mentioned. If there is carry from rounding then again result is shifted and exponent will be updated. Rounded result, updated exponent and sign of result will be provided to compose unit. Compose unit converts output to IEEE 754 standard format and also gives all exception signal.

C. Architecture for floating point square root

Architecture for floating point square root unit is as shown in Fig. 7. FPU square root has 31 stages. So clock latency for square root is 31 clocks. It receives 2 inputs, operand in IEEE 754 format and 3 bit round_mode for run time configuration. It gives out_data as square root. This will also be in IEEE format. Along with this it will give two more exceptional signals as invalid and inexact.

IEEE-754 encoded numbers (Operand) are unpacked into their corresponding sign bits (S), biased binary exponents (E), and mantissa (M). Resultant exponent will be calculated by input operand exponent with following formulae.

If (E is even) Ec = (EA + 1022) / 2;

Else (E is odd) Ec = (EA + 1023) / 2;

If there are leading zeros in mantissa, left shift is needed to normalize mantissa. This will increase precision in result.

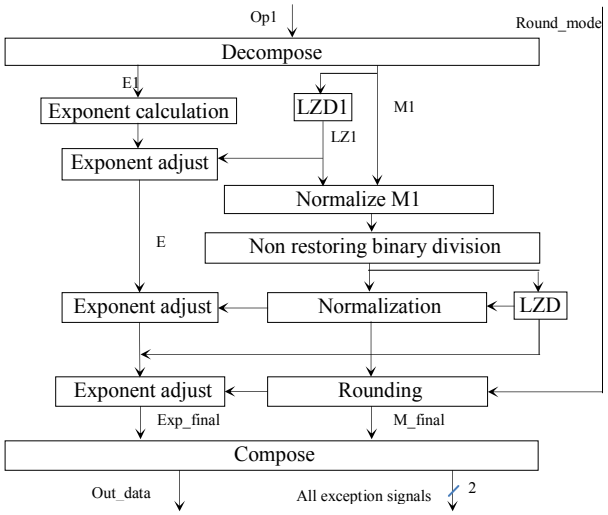

Fig. 7. Architecture For Square Root

So leading zero detectors will calculate number of leading zeros in mantissa part operand. According to that mantissa will be shifted. According to this shift for normalization resultant exponent is modified. Normalized mantissas will go through non-restoring square root calculation (NRSC) algorithm. On every clock cycle NRSC gives 1 bit of result. Result has to be normalized. Result can have maximum of 1 bit left shift. If results are too tiny then there is need of right shift to make exponent in specified limit. After this normalization it will through rounding as per rounding mode mentioned. If there is carry from rounding then again result is shifted and exponent will be updated. Rounded result, updated exponent and sign of result will be provided to compose unit which convert output to IEEE 754 standard format and also gives invalid and inexact exception signals.

D. Floating point multiplier

Floating point multiplier unit accepts either single or double precision data operands and performs either single or double precision floating point multiplication based on the *opcode* for the unit. A 3-bit rounding mode is used as input for providing the rounding mode. Given two floating point numbers X_1= {S_1, E_1, M_1} and X_2 = {S_2, E_2, M_2} the floating point multiplication result can be computed using,

S_{OUTPUT} = S_1 XOR S_2

E_{OUTPUT} = $E_1 + E_2 - bias$

1. M_{OUTPUT} =1.M_1 * 1.M_2

The architecture of the floating point multiplier is shown in Fig. 8. A divide and conquer approach is used to design the proposed floating point multiplier design. The multiplication sign can be calculated by performing the xor- operation on sign of the input operands. The 8-bit, 11-bit adder for exponent addition and 24x24-bit, 53x53-bit multiplier for mantissa multiplication are design as individual units. Based on the operation corresponding bias (127 for single precision and 1023 for double precision) are subtracted from the added exponent result. The multiplier result is rounded by using the input rounding mode. After rounding, to make the hidden bit as one, proposed design calculates the leading one position in the rounded result and forms the floating point output. The single precision floating point multiplier is having 17-clock cycles latency and double precision floating point multiplier is having 9 clock cycles latency for the operation.

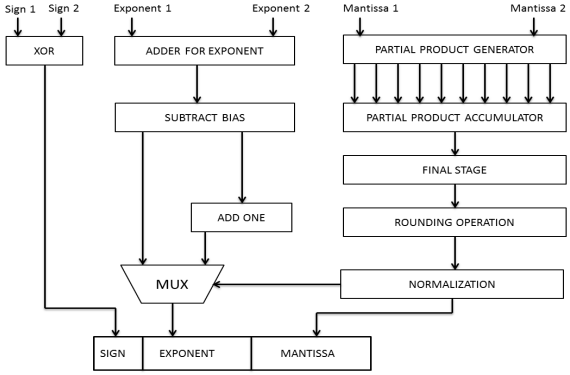

Fig. 8. Micro-Architecture of Floating Point Multiplier

E. Architecture for fused multiply and accumulator

Architecture for fused multiply and accumulate (MAC) module is as shown in Fig. 9. MAC has 10 clocks latency. This is continuous pipelined structure. It receives 3 IEEE 754 format input operands. It gives out data as MAC result and 4 bit exception signals as an output.

It takes three IEEE 754 operands decompose it into sign mantissa and exponent. MAC is divided into two parts, multiplication of first two operand and then added or subtracted result with third operand. In first part, mantissa of first and second operand gets multiplied. Result of this will undergoes normalization process as per leading zero detector. Resultant exponent is calculated by E1+E2-Bias. Normalized result and exponent is passed to second part.

Second part takes multiplication result and third operand as input. It will find out smaller operands and shifts it to make both operands exponent same. Then it will add or subtracts both mantissas. Then this result will undergo normalization. Normalized result will be rounded as per rounding mode given. Exponent will be updated after both normalization and rounding if needed. Resultant sign, mantissa and exponent will be packed into IEEE 754 form.

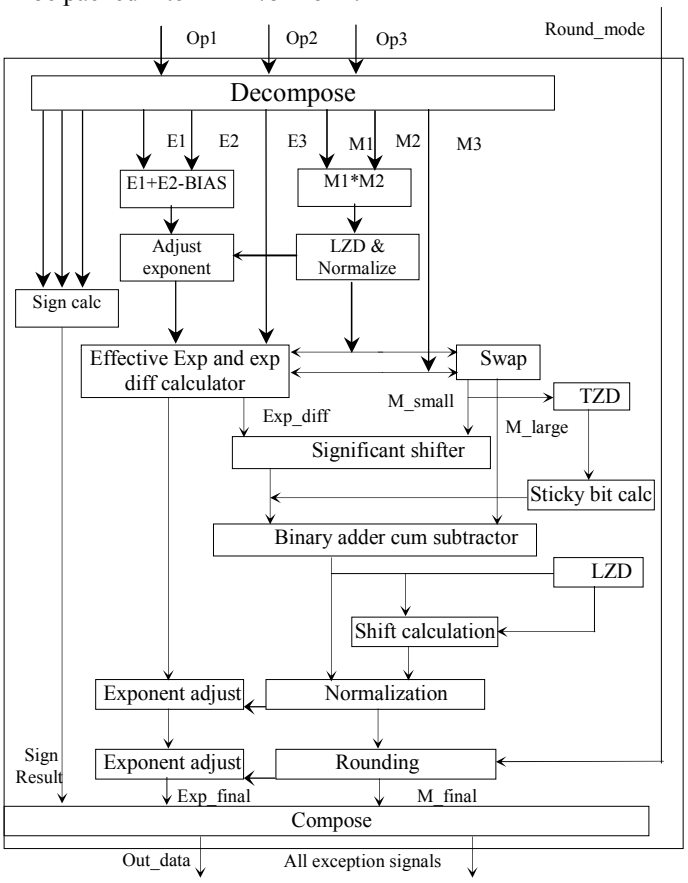

Fig. 9. Fused Multiply and Accumulate

F. Architecture for comparator

Architecture for comparator module is as shown in Fig. 10. FPU comparator has three clocks latency. This is continuous pipelined structure. It receives 2 inputs IEEE format operands. It gives 4 bit comparison result as an output. Decompose unit will separate both operands into corresponding sign (S1, S2), exponent (E1, E2) and mantissa (M1, M2). Then exponent comparator will compare both exponent and give 3 bit result as E_gr (if E1 > E2), E_ls (if E1 < E2) and E_eq (if E1=E2). Similarly mantissa comparator will give 3 bit result as M_gr, M_ls and M_eq. Now according to sign of both operands and results from both binary comparators; comparator logic unit will give results. If input operands are invalid then unordered bit will set.

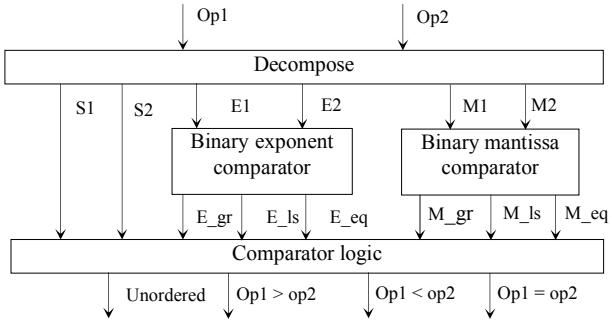

Fig. 10. Architecture For Comparator

VI. EXPERIMENT RESULTS

A. Verification and Testing

In order to verify proposed design, an extensive simulation and tests on FPGA have been done. ModelSim DE 10.0a by Mentor Graphics is used for simulation. Hardware is verified on PICO M503 FPGA board equipped with a Xilinx Virtex 6 xc6vlx550t. Results are verified and tested against IBM test suit [21] for single precision (as this is the only test suit available from IBM). Apart from these, a million random test cases have been generated. Software floating point unit using Matlab version 7.6.0.324 (R2008a) has been developed. One million generated test cases have been verified against this software implementation.

TABLE III: *Performance analysis*

Slice registers	LUTs	Frequency
40,193	55,661	180 Mhz

B. Accuracy Assessment

The experiment results show that accurate results are obtained for addition, multiplication, division and square root operations. Maximum error of 1 unit in last position (ULP) against IBM and software implementation results is observed for certain cases. This is because of difference in rounding strategies.

C. Analysis and benchmarking

A floating point coprocessor with an open source RISC V ISA has been implemented from scratch and verified with large number of test cases. It may not be fair to do one to one comparison for earlier implemented designs, as specification of design like precision, supported operations, platform are different. FPVC [22] gives throughput of 25 MFLOPs. CalmRISC32 [17] gives design frequency of 70 MHz. GRFPU [23] gives peak performance of 100 MFLOPs. By using the continuous pipelined architecture for the presented design, the FPU unit achieved a frequency of 240 Mhz for single precision and 180 Mhz for double precision operations on FPGA. So if inputs are supplied continuously on every clock cycle then

implemented design gives throughput up to 180 MFLOPS on FPGA itself.

VII. CONCLUSION

This paper presents design and implementation of an out of order floating point coprocessor enabled with an open source RISC V ISA. It is compatible with IEEE754-2008 standard. It supports common floating point arithmetic operations along with runtime reconfigurable rounding mode. It is capable of handling all exceptions mentioned in the standard. Individual floating point units uses state of the art algorithms for the implementation to achieve effective resource utilization on FPGA. By using the continuous pipelined architecture for the present design, the FPU units achieved a frequency of 240 Mhz for single precision and 180 Mhz for double precision operations on Xilinx Virtex-6 FPGA. Future work includes, exploring other ways to improve flexibility, such as an adaptive multi-mode floating point unit or reconfiguring the computation components within floating point units to perform other functions. Furthermore implementation of enhanced current design algorithms will be considered to achieve more throughputs. Currently only pluggable FPU coprocessor with RISCV ISA has been implemented, in future full processor compatible with RISC V ISA will be explored.

VIII. ACKNOWLEDGMENT

Authors sincerely thank Department of Electronics and Information Technology (DEITY), MCIT, GoI, and Dr. Sarat Chandra Babu, Executive Director, C-DAC, Bangalore for the continuous support, encouragement and the opportunity to work on the topic of this paper.

IX. REFERENCES

[1] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic. The RISC-V Instruction Set Manual,Volume I: Base User-Level ISA. Technical Report UCB/EECS-2011 -62, EECS Department, University of California, Berkeley, May 2011.

[2] Bailey D H., "High-precision floating-point arithmetic in scientific computation." Computing in Science and Engineering, 2005, 7(3):54-61.

[3] K. Scott Hemmert, Keith D. Underwood. "An Analysis of the Double-Precision Floating-Point FFT on FPGAs." Sandia National Laboratories, In proceeding of 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2005.

[4] Miaoqing Huang, Miaoqing Huang. "Reaping the processing potential of FPGA on double-precision floating-point operations: an eigenvalue solver case study ", Computer Science & Computer Engineering Department, 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, 2010

[5] Gerald R. Morris and Viktor K. Prasanna, "An FPGA-based floating-Point Jacobi Iterative Solver", department of electrical engineering, In Proceedings of the 8th International Symposium on Parallel Architectures, Algorithms and Networks, 2005.

[6] IEEE Standard for Binary Floating-Point Arithmetic,IEEE Std.754-1985.

[7] Joy, C., P. P. Boyle, et al. (1996). "Quasi-Monte Carlo Methods in Numerical Finance." Management Science 42: 926-938.

[8] Hull, J.C. Option, futures, and other derivatives. Prentice Hall, Upper Saddle River, 2000.

[9] Jäckel, P. Monte Carlo Methods in Finance. John Wiley & Sons, LTD, 2002.

[10] J. Wang, Y.-K. Yong and T. Imai, Finite element analysis of piezoelectric vibrations of quartz plate resonators with higherorder plate theory, Int. J. of Solids and Struct. 36, pp. 2303 -2319, 1999

[11] P. C. Y. Lee and Y.-K. Yong, Frequency-temperature behavior of thickness vibrations of double rotated quartz plates affected by plate dimensions and orientations, J. Appl. Phys. 60(7), 2327-2342, 1986.

[12] Marius Cornea, Cristina Anderson, Charles Tsen, "Software Implementation of the IEEE 754R Decimal Floating-Point Arithmetic", Software and Data Technologies Communications in Computer and Information Science Volume 10, 2008, pp 97-109.

[13] Jainik Kathiara, Miriam Leeser "An Autonomous Vector/Scalar Floating Point Coprocessor for FPGAs", IEEE International Symposium on Field-Programmable Custom Computing Machines, 2011

[14] Haibing HU, Tianjun Jin, Xianmiao Zhang, Zhengyu LU, Zhaoming Qian, "A Floating-point Coprocessor Configured by a FPGA in a Digital Platform Based on Fixed-point DSP for Power Electronics", Power Electronics and Motion Control Conference, 2006.

[15] ChrisN.Hinds, "An Enhanced Floating Point Coprocessor for Embedded Signal Processing and Graphics Applications", Conference on Signals, Systems, and Computers, 1999.

[16] Xunying Zhang, Xubang Shen, "A Power-Efficient Floating-point Co-processor Design", 2008 International Conference on Computer Science and Software Engineering

[17] C. Jeong, S Kim, M Lee "In-Order Issue Out-of-Order Execution Floating-point Coprocessor for CalmRISC32". 15th IEEE Symposium on Computer Arithmetic, 2001

[18] ARM Ltd. ARMv6-M Architecture Reference Manual, 2008.

[19] Opencores.org. Open-RISC Architecture Reference Manual, 2003.

[20] Kihwan Jun ; Swartzlander, E.E. "Modified non-restoring division algorithm with improved delay profile and error correction" Conference on Signals, Systems and Computers (ASILOMAR), 2012 Conference Record of the Forty Sixth Asilomar

[21] FPgen Team, "Floating-Point Test-Suite for IEEE", IBM Labs in Haifa

[22] Jainik Kathiara, Miriam Leeser "An Autonomous Vector/Scalar Floating Point Coprocessor for FPGAs" IEEE International Symposium on Field-Programmable Custom Computing Machines, 2011

[23] J. Gaisler, The LEON-2 Processor User's Manual Version 1.0.13. Goteborg, Sweden: Gaisler Research, 2003