ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

*(A High Impact Factor, Monthly, Peer Reviewed Journal)*

*Website: www.ijareeie.com*

**Vol. 7, Issue 3, March 2018**

# Optimised Delay and Area Efficient Floating Point Arithmetic Unit

A.Arun [1], K.Rachel[2], S.Sathya[2] , C.Subbulakshmi[2], S.Varsha[2]

Assistant Professor, Dept. of ECE, Velammal Engineering College, Chennai, Tamilnadu, India[1]

UG Student, Dept. of ECE, Velammal Engineering College, Chennai, Tamilnadu, India[2]

**ABSTRACT:** In this paper we present a Hybrid floating-point (FP) implementations improve software FP performance without increasing the area overhead of hardware floating point units. The proposed implementations are synthesized in 65-nm CMOS and integrated into small fixed-point processors with a RISC-like architecture. Unsigned, shift carry, and leading zero detection (USL) support is added to a processor to increase the performance of the existing instruction set architecture and increase FP throughput. The hybrid implementations with USL support increase software FP throughput per core by 1.29× for multiplication and 3.07–4.05× for division and use 90.7–94.6% less area than dedicated hardware. Hybrid implementations with custom FP-specific hardware increase throughput per core over a FP software kernel by 1.22–2.03× for multiplication, 14.4× for division, and use 77.2–97% less area than dedicated hardware. The circuit area and throughput are found for 6 multiplication, 45 division designs. Index Terms—Arithmetic and logic structures, computer arithmetic, fine-grained system, floating point (FP).

**KEYWORDS:** Throughput, FPGA, Area, Floating point, Hybrid

## I. INTRODUCTION

FLOATING-POINT (FP) representation is the most commonlyused method for approximating real numbers. However, the large area and powerneeds of FP hardware limit many architectures to fixed point, picoChip. Small chip area is needed formulit-core architectures, since incurring area per core has reduced the number of cores that will fit on a chip die.Many methods have been proposed for increasing FPthroughput and low area overhead. Fused andcascade multiply–add FPUs results in more arithmetic, for example, SDR architectures,Blackfin microprocessors accuracy and provide speedup ; though they introduce large area and power overhead, which is not suitable forsimple fixed-point processors. If blocks of data having samemagnitudes, block FP (BFP) can be useful for increasing signal to noise ratioand dynamic range. Microoperations is beingused to create a virtual floating point unit, which uses existing fixed-pointhardware to enhance the FP datapath for a VLIW processor. The required hardware for FPdivision is also reduced by hardware prescaling and postscaling by shortening the exponent and mantissa. Custom FP instructions have also been proposedfor a FPGA to increase FP throughput with optimised areathan a dedicated hardware FPU.This paper proposes hybrid FP implementations, whichperforms FP usinga combination of fixed-point software instructions and hardware. Hybrid implementations offers area–throughput tradeoffs either through full software or hardware implementations.The main contributions of this paper are as follows.1) Two hybrid implementations with CFP hardware and six with USL support.2) Design and implementation of 6 multiplication and 45 division designs. These designs include full software kernels, full hardware modules, hybrid implementations with USL support and with CFP hardware. Three different algorithms for division are used.3) Evaluation of the proposed software kernels, hardware modules, and hybrid implementations, and FPUs.

## II.LITERATURE SURVEY

When hardware modules are used,it results in large area and high throughput.And when software kernels are used ,it results in low area and low throughput.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

*(A High Impact Factor, Monthly, Peer Reviewed Journal)*

*Website: www.ijareeie.com*

**Vol. 7, Issue 3, March 2018**

## III. FLOATING-POINT COMPUTATION BACKGROUND

A. Floating-Point Format:

This uses the IEEE-754 single-precision format for all FP arithmetic, with values on the normalized value interval ±[2−126, (2−2−23)×2127]. Round to nearest even, the IEEE-754 default rounding mode, and round toward zero are supported for all FP arithmetic.

B. Fating intagrithm:

The FP operation algorithms are

1) Multiplication: Multiplication operation starts by multiplying the mantissas. The initial power is set by adding the operand exponents, the resultant product is normalized and
Rounded off, and the sign bit is given by XORing the sign bit of both operands.

2)Division: Three algorithms are implemented for division:long-division, nonrestoring, and Newton–Raphson .It is typically an infrequent operation ; therefore, little area must be assigned. The long-division and nonrestoring algorithms are chosen for their simplicity and low area optimisation, while the Newton–Raphson algorithm is chosen for its high throughput. Initially the magnitude of dividend and divisor is compared by the long division method. If the divisor is lesser than or equal to the dividend, it should be subtracted from the dividend to form a partial remainder, and 1 is right shifted as the next bit of the quotient. Or else, they are not subtracted and 0 is shifted. Then the partial remainder is left shifted by 1 bit and it is set as the new dividend. This process continues till all the quotient bits are verified. The division result is normalized and rounded off. The result is calculated by subtracting the input powers and adding back the bias. The nonrestoring division algorithm is same as the restoring algorithm except the restoring step for each loop iteration. The dividend is subtracted from the divisor. If the result is positive or negative a loop is executed and quotient is shifted.If the result is negative, the dividend is added to the partial result otherwise the least significant bit (LSB) of the quotient is set to 1 and the result gets subtracted from the dividend. This loop continues to iterate till all bits are checked for the quotient . The finally if the result is negative the result is restored. The final result is normalized and rounded off. The exponent is calculated similar to long division.For the Newton–Raphson division algorithm, the reciprocal of the divisor is checked repeatedly and then the dividend gets multiplied. The divisor and dividend are prescaled to a small interval. A linear approximation is used to calculate the reciprocal and reduce the maximum relative error of the final result. This calculation is then improved repeatedly. Once this reciprocal is verified, it is multiplied by the prescaled dividend to get the result,which is then extracted by computing residuals at a higher precision. XORing the sign bits of the operands,we get the results.


## IV. FULL SOFTWARE KERNELS

These are coded in AsAP instructions and forms a software library consisting of multiplication and division. They are full software because they use only GP fixed-point instructions. The platforms word size is 2 bytes and each value is received on chip as two words. To simplify evaluation, the words are split into four to store the following: the sign bit, exponent, high and low mantissa bits. As these kernels use only the platforms existing fixedpoint datapath, they do not add area.The programs for these kernels are large due to the lack of unsigned ALU instructions and the number of FP instructions required for emulating FP hardware. Computation time for software FP comprises primarily of operand comparisons, mantissa alignment, addition, normalization, and rounding.

1. Multiplication Kernel

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

*(A High Impact Factor, Monthly, Peer Reviewed Journal)*

*Website: www.ijareeie.com*

**Vol. 7, Issue 3, March 2018**

The instruction overheads for this kernel are used for calculating mantissa multiplication and rounding. The partial products are added using the MAC and aligned by the shifter.

2. A.Division Kernel Version 1

The kernel uses the long-division algorithm. The loop to determine the quotient requires the greatest number of instructions and involves several shift and subtract operations.

2. B. Division Kernel Version 2

This kernel uses the Newton-Raphson algorithm . The kernel starts with zero input detection and handling, followed by exponent calculation. The input is prepared for later calculations. The estimate of the reciprocal is calculated, followed by Newton-Raphson iterations. The initial input is then multiplied by the reciprocal of the second, and the result is normalized and rounded. Then, the LSB is corrected.

## V. FULL HARDWARE MODULES

These modules offer the highest throughput, but require the most area of the designs implemented. Full hardware modules are referred to as full hardware because all arithmetic is performed on dedicated FP hardware. As the target platform has a 16-bit datapath, the FP values are first loaded into FP registers. Each value is then stored as two 16-bit words. An entire FP operation is carried out by a single FP instruction and the results are read from the FP registers, 2 bytes at a time.

Multiplication Module

This module uses the FPMult instruction with a one-cycle execution latency to perform multiplication. Full HW Mult (32-bit I/O) is created for a 32-bit datapath and word size and uses the FPMult32 instruction to perform multiplication with a single-cycle execution time delay. Assuming operands are read from a processors local memory, a single instruction can perform multiplication.

DIVISION MODULE

This module performs the restoring division algorithm using FPDiv. This instruction has a 30-cycle execution latency.Full HW Div (32-bit I/O) is created for a 32-bit datapath and word size and uses the FPDiv32 instruction with a 30-cycle execution latency. A single instruction performs division when operands are read from a processors local memory.

## VI. PROPOSED HYBRID IMPLEMENTATIONS WITH UNSIGNED, SHIFT-CARRY, AND LEADING ZERO DETECTION

To determine the throughput and area achievable by increasing the instruction set, USL support is added to the target platforms ISA. Many ISA modifications are implemented, including adding unsigned operation support, leading zero detection, and additional shift-carry instructions. These extra shift instructions can set a carry flag when data are shifted out.

Multiplication Hybrid Implementation With USL Support

The unsigned multiply accumulate instructions decrease the overhead for partial product calculation. The additional shift instruction makes it easy for normalization and the unsigned addition instructions reduce the instruction count for rounding.

DIVISION HYBRID IMPLEMENTATION WITH USL SUPPORT VERSION 1

This uses the long-division algorithm . Unsigned addition/subtraction and added shift instructionsreduce the IC for exponent calculation, divisor and dividend mantissa subtraction, rounding, and normalization.

D. Division Hybrid Implementation With USL Support Version 2

This implementation uses the Newton-Raphson algorithm . The multiply accumulate, and additional shift instructions reduce the instruction count for calculating the exponent and initial estimate,executing the Newton-Raphson iterations, multiplying the input by the reciprocal, rounding, and then correcting the LSB.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

*(A High Impact Factor, Monthly, Peer Reviewed Journal)*

*Website: www.ijareeie.com*

**Vol. 7, Issue 3, March 2018**

## VII.PROPOSED HYBRID IMPLEMENTATIONS WITH CUSTOM FP-SPECIFIC HARDWARE

Hybrid implementations with CFP hardware is composed of fixed-point software and custom FP instructions operating together on FP workloads. These increase the throughput by reducing the difficulties of full software kernels and require less area than full hardware modules.CFP instructions perform operations on data which is stored in FP registers, and each value is stored as two 16-bit words.

A. Multiplication Hybrid Implementation With CFP Hardware Version 1

This implementation calculates mantissa multiplication and exponent and sign bit calculation using fixed-point software instructions. FPMult_NormRndCarry performs the rest of the operation and is given as follows.

1) FPMult_NormRndCarry: After mantissa multiplication and exponent calculation, the product is loaded into an FP register. The normalized and rounded mantissa is given back into an FP register, the 16 MSBs are returned, and the carry flag is set. If the carry flag is set, the exponent is incremented by 1 in software.

B .Multiplication Hybrid Implementation With CFP Hardware Version 2

This implementation performs mantissa multiplication in software using FP instructions. FPMult_NormRnd performsthe rest of the operation and is described as follows.

1) FPMult_NormRnd: Following software mantissa multiplication, the product, exponent, and sign bits are loaded into FP registers. The sign bit, exponent, and normalized and rounded product are then calculated.

Division Hybrid Implementation With CFP Hardware Version

The nonrestoring division algorithm is used to perform FP division with this implementation. The exponent and sign bit of the result are then determined. The instruction described in the following performs the rest of the operation.

1) FPDiv_LoopExpAdj: After the two inputs and the partially computed exponent are loaded into the FP registers,this instruction performs the division loop. The exponent is adjusted in the hardware following normalization and then rounding.

## VIII. RESULTS AND COMPARISONS

Every implementation is synthesized with a 65-nm CMOS standard cell library using Synopsys DC compiler with a 1.3 V operating voltage and 25 °C operating temperature and clock frequencies of 600, 800, 1000, and 1200 MegaHertz. For the need of accuracy and performance analysis, FPgen, a test suite for verifying FP datapaths is used to include test cases unlikely to be covered by pure random test generation.This testing is supplemented by using pseudorandomly generated FP values on the normalized value interval $\pm[2^{-126}, (2 - 2^{-23}) \times 2127]$. With the exception of the full software kernels, every design adds circuitry to the platform processor, the area for this circuitry is referred to as additional area.

A. Individual FP Designs Compared

Additional area is plotted versus cycles per FLOP times the clock period in ns. The designs are plotted on separate graphs according to the type of operation. We can determine the optimal design with respect to an area constraint by selecting an implementation that uses less area than the constraint and requires the fewest average cycles per FLOP. For example, consider an area constraint Amax equal to 10% of the target platform processor area. For this, more area is allocated for multiplication hardware because division is less frequent operations. The full hardware designs require the most area to achieve the highest throughput; however, none of these implementations meets the area constraint and the FMA is the largest implementation, increasing processor area by 33%. Except for multiplication, the hybrid implementations with USL support require the least area to improve throughput. They can also be used for general-purpose workloads because the USL instructions are non-FP specific. For the full software kernels, division requires less cycles per FLOP using the long-division and digit-by-digit algorithms, respectively. But, the division hybrid implementations with USL support require slightly less cycles per FLOP when using the Newton-Raphson algorithm.

B. Comparison When Combining FP Designs

To compare the throughput and area when adding multiple designs, the FP designs are combined into 38 functionally equivalent FPU implementations consisting of an addition/subtraction and multiplication unit. These designs are evaluated for performing Newton-Raphson division. These Newton-Raphson and FMA implementations of the divide and square root are mapped in a pipelined fashion and loops are unrolled to potentially provide high throughput . The

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

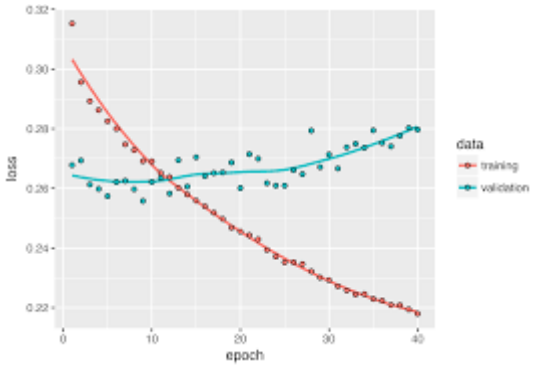
International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

*(A High Impact Factor, Monthly, Peer Reviewed Journal)*

*Website: www.ijareeie.com*

**Vol. 7, Issue 3, March 2018**

multiplication and division implementations are compared with full software, full hardware, and hybrid designs using the long-division, digit-by-digit, nonrestoring, or Newton-Raphson algorithm.




## IX.ADVANTAGES OF HYBRID APPROACHES

Implementing using software becomes the only option for floating point arithmetic when area cannot be improved. When the goal is maximum throughput, dedicated hardware unit becomes absolute. Hybrid designs are ideal than floating point hardware unit because they increase the throughput andrequire less area, when the area is constrained. They provide a procedure for fulfilling an area constraint that hardware units would contravene. By appending functions to prevailing hardware unit using hybrid designs with USL support it can simplify multiword operations. The hybrid implementation with CFP (Computing Forward Progress) can add custom hardware which performs specific steps of an floating point operation which can overshoot the performance of USL design support. Else,these techniques would require many fixed point instructions. The complete software design requires many operations on large multiword data values. Summing and carrying between partial products or carrying between words is the function required by multiword operation. The coder must neglect to use signed bit(bit 16 for target platform)and must manipulate carry flags and summation of partial product in software. Thus, fully utilized 16-bit words cannot be operated by signed hardware unit. Words must be divided into 15 bits each at most. The hybrid designs with USL assist provide unsigned hardware proficient handling of multiword values, increasing Newton-Raphson throughput. The methods like long-division and digit-by-digit are much less advantage, as they rely on more number of shifts. To discover the gain of different design approaches multiple hybrid designs with CFP are being implemented. Each class will differ in terms of which steps or the percentage of the floating point operation that is conducted in software. Based on the throughput increase and area overhead the steps that support hardware are justified. By supporting operand comparison in hardware Hybrid Add/Sub w/CFP Ver.3 improves the add/sub operation throughput the highest. Else, categorizing the operands to compare the exponent and multiword mantissa will require many instructions.For multiplication, by adding more hardware assist to Ver.1 Hybrid Mult w/CFP Ver.2 improves throughput the most. This execution will reduce the executed instruction count by determining if the outcome is zero and manipulating the sign bit and exponent in hardware. This secondary hardware circuitry increases the area.By implementing sign bit and exponent calculation the division and square root implementation will use lesser area in software than the dedicated hardware floating point unit and by executing the balance operation in hardware the throughput of these operations is improved.

| | |
|---|---|
| FULL HW MULT | FPMULT |
| FULL HW MULT (32-bit I/O) | FPMULT32 |
| FULL HW DIV | FPDIV |
| FULL HW DIV (32-bit I/O) | FPDIV32 |

## X. RELATED WORK AND COMPARISON

Since this work gives single- precision floating point operations,we equate our outcomes with other work that improves single-precision floating point throughput with lesser area than a dedicated hardware implementation and we do not equate with designs using reduced floating point word size or Binary Floating Point(BFP).Our outcomes comprises

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

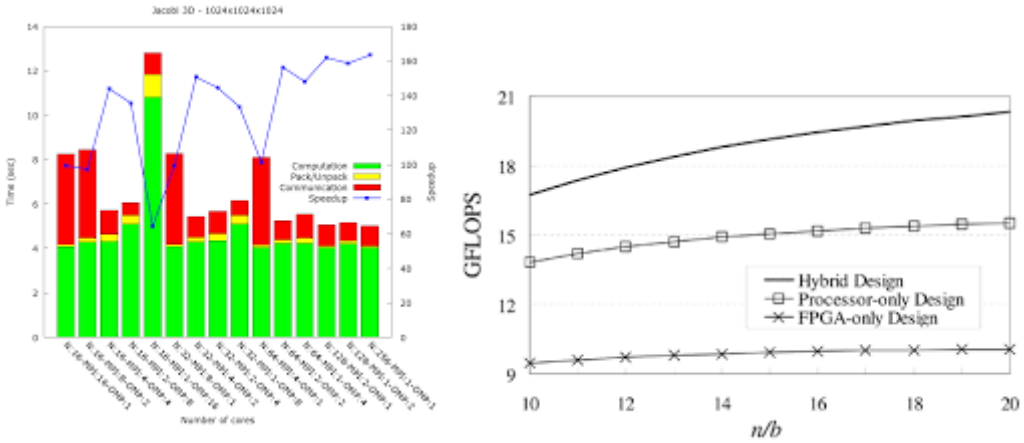
International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

*(A High Impact Factor, Monthly, Peer Reviewed Journal)*

*Website: www.ijareeie.com*

**Vol. 7, Issue 3, March 2018**

implementations with a 16-bit and 32-bit word size and datapath where everything is implemented using single-precision floating point.Since not every task reports area data;to make a constant evaluation,the area overhead of each design is compared against dedicated hardware implementation reported in respective task. This paper describes about the area overhead for assisting each floating point operation independently.Thus, area is reported under floating point operation classes for which cycle counts are reported whereas other work do not issue area for particular operations.An unmerged operation is being performed for multiply-add by our operations,except for the FMA design.The work by Gilani et al andViitanen et al. did not investigate modular implementations.With differing amount of modular implementation Hockert and Compton investigated modular design,but did not examine the overhead for assisting independent floating point operations.Granting more adaptability across a wide range of area constraints for improving floating point throughput,our work presents a larger range ofarea overheads.For both divide and square operation,our designs need less area than FMA and also offerlowest cycles per FLOP.Our 32-bit I/O implementations has achieved the lowest cycles per FLOP for all operation classes when equated with other work that decrease floating point area overhead compared with dedicated floating point hardware.




## XI. CONCLUSION

In this paper, two hybrid implementations with CFP hardware and hybrid implementations with USL support are presented for a fixed-point processor. These implementations increase the throughput of FP operations by adding USL support instructions to the ISA, and custom FP instructions. The area overhead is kept low by utilizing the existing fixedpoint functional units. The circuit area and throughput are found for 6 multiplication, 45 division designs. This paper presents designs that improve FP throughput versus a baseline software implementation and require less area overhead compared with an FMA than other works. Many examples demonstrate how to determine the optimal FP designs for a given area constraint. Hybrid implementation is an effective design method for increasing FP throughput and require up to 97% less area than a traditional FMA.

## ACKNOWLEDGMENT

The authors would like to thank STMicroelectronics for donating the chip fabrication.

## REFERENCES

[1] J.-M. Muller et al., Handbook of Floating-Point Arithmetic, 1st ed.Basel, Switzerland: Birkhäuser, 2009.
[2] S. Z. Gilani, N. S. Kim, and M. Schulte, "Energy-efficient floatingpoint arithmetic for software-defined radio architectures," in Proc. IEEE Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP), Sep. 2011, pp. 122–129.
[3] S. M. Shajedul Hasan and S. W. Ellingson, "An investigation of the Blackfin/uClinux combination as a candidate software radio processor,"Dept. Elect. Comput. Eng., Virginia Polytechn. Inst. State Univ.,Blacksburg, VA, USA, Tech. Rep. 2, 2006.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

*(A High Impact Factor, Monthly, Peer Reviewed Journal)*

*Website: www.ijareeie.com*

**Vol. 7, Issue 3, March 2018**

[4] Floating Point Arithmetic on the PicoArray, accessed on Jun. 5, 2015. [Online]. Available: https://support.picochip.com/picochip-resourcefolder/Nexu5utu/4598jf4897f/floatingpoint.pdf/download
[5] C. Iordache and P. T. P. Tang, "An overview of floating-point support and math library on the Intel XScale architecture," in Proc. 16th IEEE Symp. Comput. Arithmetic, Jun. 2003, pp. 122–128.
[6] Z. Yu et al., "AsAP: An asynchronous array of simple processors," IEEE J. Solid-State Circuits, vol. 43, no. 3 pp. 695–705, Mar. 2008.
[7] D. N. Truong et al., "A 167-processor computational platform in 65 nm CMOS," IEEE J. Solid-State Circuits, vol. 44, no. 4, pp. 1130–1144, Apr. 2009.
[8] N. Hockert and K. Compton, "Improving floating-point performance in less area: Fractured floating point units (FFPUs)," J. Signal Process. Syst., vol. 67, no. 1, pp. 31–46, Apr. 2012.
[9] IEEE Standard for Floating-Point Arithmetic, IEEE Standard 754-2008, 2008.
[10] D. R. Lutz and C. N. Hinds, "Novel rounding techniques on the NEON floating-point pipeline," in Proc. 39th Asilomar Conf. Signals, Syst. Comput., Oct./Nov. 2005, pp. 1342–1346.
[11] A. H. Karp and P. Markstein, "High-precision division and square root," ACM Trans. Math. Softw., vol. 23, no. 4, pp. 561–589, Dec. 1997.
[12] M.-B. Lin, Digital System Designs and Practices: Using Verilog HDL and FPGAs. New York, NY, USA: Wiley, 2008.
[13] S. F. Oberman and M. J. Flynn, "Design issues in division and other floating-point operations," IEEE Trans. Comput., vol. 46, no. 2,pp. 154–161, Feb. 1997.
[14] Z. Jin, R. N. Pittman, and A. Forin, "Reconfigurable custom floatingpoint instructions," Microsoft Res., Tech. Rep. MSR-TR-2009-157, Aug. 2009.
[15] S. F. Oberman and M. Flynn, "Division algorithms and implementations,"