ANALYSIS OF FPGA BASED 32-BIT FLOATING POINT ARITHMETIC UNIT

[1]D KIRTANA , [2] P KIRANMAYEE, [3]V VIJAYA KUMAR
[1,2,3]Assistant professor, ECE Department, St.Martin's Engineering College,Sec

ABSTRACTS
In a wide range of DSP applications includes processing of sensor array processing, audio and speech signal processing, control of systems, radar and sonar signal processing, spectral estimation, digital image processing, seismic data processing, biomedical signal processing, statistical signal processing, signal processing for communications, Filter designing & many high accuracy based operations. Floating point operations are used due to its huge dynamic range, high accuracy and straightforward operation rules. With the current trends in system requirementsand available FPGAs, floating-point implementations arebecoming more common and designers are increasinglytaking advantage of FPGAs as a platform for floating-pointimplementations. The rapid advance in Field-ProgrammableGate Array (FPGA) technology makes such devicesincreasingly attractive for implementing floating-pointarithmetic. Compared to Application Specific IntegratedCircuits, FPGAs offer reduced development time and costs.Moreover, their flexibility enables field upgrade andadaptation of hardware to run-time conditions. This paper describes the process of building a general floating point arithmetic unit using Verilog HDLbased on FPGA. The floating point arithmetic unit can perform addition and subtraction operations of a couple of double precision floating point numbers or two couple of single precision floating point numbers. At the end of this paper, the features and calculation correctness are proved through simulation and hardware experiments.

1. INTRODUCTION

The implementation of the floating point arithmetic has been appropriate within the floating point high level languages; however the execution of the arithmetic by hardware is difficult task. With the expansion of the very large scale integration (VLSI) technology have become the most effective choices for implementing floating hardware arithmetic units due to their high integration density, high performance, low worth and versatile applications needs for prime precious operation. The IEEE 754 standard presents two completely different floating point formats, Binary interchange format and Decimal interchange format. This section focuses solely on single precision normalized binary interchange format. Figure 1 shows the IEEE 754 single precision binary format representation, it consists of a one bit sign (S), an eight bit exponent (E), and a twenty three bit fraction (M) or Mantissa.

In digital signal processing, image processing, voice communications, wireless communications and many other fields, a large number of data with different precision and high requirements of real-time need to be processed. Floating point arithmetic has the characteristics of high precision. But compared to integers arithmetic , floating- point arithmetic occupies more hardware resources so that it isimplementedbysoftwareinmanysystems.Asares ult,the operationspeedofthisfloatingpointarithmeticisver yslow. And although hardware floating-point arithmetic can increase the speed of computation, the floating-point arithmetic for multiple precision requires many floating- point units, which occupies a large amount of hardware resources. So if a floating-point unit which can achieve different precision processing is designed, hardware costs willbereduced.AndtherapiddevelopmentofFPGA makes itpossible.Themulti-precisionfloating-pointarithmeticunit, proposedinthispaper,canchangetheinternalconfig uration of the circuit (when operating), according to the calculation precision, in order to achieve single or double precision calculation with minimum hardwareresources.

Theimplementation of the floating point arithmetic hasbeen very easy and convenient in the floating point high level languages, but the implementation of thearithmetic by hardware has been very difficult. Withthe development of the very large scale integration(VLSI) technology, a kind of devices like FieldProgrammable Gate Arrays (FPGAs) have becomethe best options for implementing floating hardwarearithmetic units because of their high integrationdensity, low price, high performance and flexibleapplications requirements for high precious operation.Floating-point implementation on FPGAs has beenthe interest of many researchers. The use of customfloating-point formats in FPGAs has beeninvestigated in a long series of work [3], [4], [8]. Inmost of the cases, these formats are shown to beadequate for some applications that requiresignificantly less area to implement than IEEEformats [6] significant

speedups for certain chosenapplications. The earliest work on IEEE floating-point[7] focused on single precision although found to befeasible but it was extremely slow. Eventually, it wasdemonstrated [8] that while FPGAs wereuncompetitive with CPUs in terms of peak FLOPs,they could provide competitive sustained floatingpointperformance. Since then, a variety of work [2],[5],[9]-[10] has demonstrated the growing feasibilityof IEEE compliant, single precision floating pointarithmetic and other floating-point formats ofapproximately same complexity. In [2], [5], thedetails of the floating-point format are varied tooptimize performance. The specific issues ofimplementing floating-point division in FPGAs havebeen studied [10].

Early implementations eitherinvolved multiple FPGAs for implementing IEEE 754single precision floating-point arithmetic, or theyadopted custom data formats to enable a single-FPGAsolution. To overcome device size restriction,subsequent single-FPGA implementations of IEEE754 standard employed serial arithmetic or avoidedfeatures, such as supporting gradual underflow, whichare expensive to implement.

In this paper, a high-speed IEEE754-compliant 32-bit floating point arithmetic unit designed usingVHDL code has been presented and all operations ofaddition, subtraction, multiplication and division gottested on Xilinx and verified successfully along withthat all the exceptions of floating point numbers arestudied in detail. The simulation results of addition,subtraction, multiplication and division in Modelsimwave window.

2. LITERATURE REVIEW

In 2010, Kuang proposed a power-efficient 16x16 multiple precision multiplier. Using the slightly different divide-and-conquer technique similar to, four small 8x8 modified Booth multipliers are employed for the generation of partial products for the higher and lower portions of the computation. The resulting partial products are reorganized and fed to a large reduction tree followed by a fast adder for the generation of the final results. The basic idea is illustrated where CV is the correction vector required for Booth multiplication. They also investigated the potential of saving power when single lower precision operation or operations with truncation were required. A dynamic range detector with supplement shutdown circuit was presented to handle such power-saving scenarios.

In 2008, Akkas presented a technique capable of modifying an IEEE adder architecture to a new dual-mode one that allows one operation of native precision or two parallel additions on half of the native precision (e.g., one double or two single). The author showed the detail designs for a 5-stage pipelined dual-mode double precision adder with improved single-path algorithm, and a 3-stage dual-mode quadruple precision adder with the two-path algorithm. Both designs support only normalized numbers. The implementation (0.11 um CMOS) area and latency overhead of the dual-mode double precision adder is around 26% and 10%, while those for the dual-mode quadruple adder is 13% and 18%.

Even et al. proposed a dual precision IEEE floating-point multiplier that can compute one single-precision result in 2 clock cycles or one double-precision product in 3 cycles, supporting all IEEE-compliant rounding modes. The half-size multiplication array (e.g., 27×53) is used in the first clock cycle for single precision, or the first two cycles for double precision, with the following cycle allocated for the final addition and rounding/normalization. Therefore, there will be one stall cycle after a double precision operation.

Floating-point representation provides better dynamic range support, thus is more useful for scientific computations. In 2008, Akkas presented a technique capable of modifying an IEEE adder architecture to a new dual-mode one that allows one operation of native precision or two parallel additions on half of the native precision (e.g., one double or two single). The author showed the detail designs for a 5-stage pipelined dual-mode double precision adder with improved single-path algorithm, and a 3-stage dual-mode quadruple precision adder with the two-path algorithm. Both designs support only normalized numbers. The implementation (0.11 um CMOS) area and latency overhead of the dual-mode double precision adder is around 26% and 10%, while those for the dual-mode quadruple adder is 13% and 18%.

This paper describes the architecture and implementation, from both the standpoint of target applications as well as circuit design, of an FPGA DSP Block that can efficiently support both fixed and floating-point (FP) arithmetic. Most contemporary FPGAs embed DSP blocks that provide simple multiply-add-based fixed-point arithmetic cores. Current floating-point arithmetic FPGA solutions make use of these hardened DSP resources, together with embedded memory blocks and soft logic resources, however, larger systems cannot be efficiently implemented due to the routing and soft logic limitations on the devices, resulting in significant area, performance, and power consumption penalties compared to ASIC implementations. In this paper we analyze earlier proposed embedded floating-point implementations, and show why they are not suitable for a production FPGA. We contrast these against our solution – a unified DSP Block – where (a) the FP multiplier is overlaid on the fixed point constructs, (b) the FP Adder/Subtracter is integrated as a separate unit; and (c) the multiplier and adder can be combined in a way that is both arithmetically useful, but also efficient in terms of

FPGA routing density and congestion. In addition, a novel way of seamlessly combining any number of DSP Blocks in a low latency structure will be introduced. We will show that this new approach allows a low cost, low power, and high density floating point platform on current 20nm FPGAs.

3. A VARIABLE PRECISION FIXED- AND FLOATINGPOINT LIBRARY FOR RECONFIGURABLE HARDWARE

In variable precision floating-point library (VFloat) that supports general floating-point formats as well as IEEE standard formats. optimum reconfigurable hardware implementations could need the utilization of arbitrary floating-point formats that don't essentially adjust to IEEE standard sizes. Most antecedently printed floating-point formats to be used with reconfigurable hardware square measure subsets of our format. Custom data paths with optimum bit widths for every operation may be designed mistreatment the variable exactitude hardware modules within the VFloat library, enabling a better level of similarity. The VFloat library includes three varieties of hardware modules for format management, arithmetic operations, and conversions between fixed-point and floating-point formats. The format conversions gives hybrid fixed- and floating-point operations during a single style [1].

ALGORITHMS FOR FLOATING POINT ARITHMETIC UNIT

The algorithms using flow charts for floating point addition/subtraction, multiplication and division have been described in this section, that become the base for writing VHDL codes for implementation of 32-bit floating point arithmetic unit.

3.1 Floating Point Addition / Subtraction

The algorithm for floating point addition is explained. While adding the two floating point numbers, two cases may arise. Case I: when both the numbers are of same sign i.e. when both the numbers are either +ve or –ve. In this case MSB of both the numbers are either 1 or 0. Case II: when both the numbers are of different sign i.e. when one number is +ve and other number is –ve. In this case the MSB of one number is 1 and other is 0.

Case I: - When both numbers are of same sign

Step 1:- Enter two numbers N1 and N2. E1, S1 and E1, S2 represent exponent and significant of N1 and N2 respectively.

Step 2:- Is E1 or E2 =“0”. If yes; set hidden bit of N1 or N2 is zero. If not; then check if E2 > E1, if yes swap N1 and N2 and if E1 > E2; contents of N1 and N2 need not to be swapped.

Step 3:- Calculate difference in exponents d=E1-E2. If d= “0” then there is no need of shifting the significant.

If d is more than “0” say “y” then shift S2 to the right by an amount “y and fill the left most bits by zero.

Shifting is done with hidden bit.

Step 4:- Amount of shifting i.e. “y” is added to exponent of N2 value. New exponent value of E2=(previous E2) + “y”. Now result is in normalize form because E1 = E2.

Step 5:- Check if N1 and N2 have different sign, if “no”;

Step 6:- Add the significant of 24 bits each including hidden bit S=S1+S2.

Step 7:- Check if there is carry out in significant addition. If yes; then add “1” to the exponent value of either E1 or new E2. After addition, shift the overall result of significant addition to the right by one by making MSB of S as “1” and dropping LSB of significant.

Step 8:- If there is no carry out in step 6, then previous exponent is the real exponent.

Step 9:- Sign of the result i.e. MSB = MSB of either N1 or N2.

Step 10:- Assemble result into 32 bit format excluding 24th bit of significant i.e. hidden bit.

Case II: - When both numbers are of different sign

Step 1, 2, 3 & 4 are same as done in case I.

Step 5:- Check if N1 and N2 have different sign, if “Yes”;

Step 6:- Take 2“s complement of S2 and then add it to S1 i.e. S=S1+ (2“s complement of S2).

Step 7:- Check if there is carry out in significant addition. If yes; then discard the carry and also shift the result to left until there is “1” in MSB and also count the amount of shifting say “z”.

Step 8:- Subtract “z” from exponent value either from E1 or E2. Now the original exponent is E1-“z”. Also append the “z” amount of zeros at LSB.

Step 9:- If there is no carry out in step 6 then MSB must be “1” and in this case simply replace “S” by 2“s complement.

Step 10:- Sign of the result i.e. MSB = Sign of the larger number either MSB of N1or it can be MSB of N2.

Step 11:- Assemble result into 32 bit format excluding 24th bit of significant i.e. hidden bit.

In this algorithm three 8-bit comparators, one 24-bit and two 8-bit adders, two 8-bit subtractors, two shift units and one swap unit are required inthe design.

3.2 Floating Point Multiplication

The algorithm for floating point multiplication isexplained through flow chart in Figure 3. Let N1 andN2 are normalized operands represented by S1, M1,E1 and S2, M2, E2 as their respective sign bit,mantissa (significant) and exponent. Basicallyfollowing four steps are used for floating pointmultiplication.

1. Multiply significant, add exponents, and determinesignM=M1*M2E=E1+E2-BiasS=S1XORS2

2. Normalize Mantissa M (Shift left or right by 1) andupdate exponent E

3. Rounding the result to fit in the available bits
4. Determine exception flags and special values foroverflow and underflow.

Sign Bit Calculation: The result of multiplicationis a negative sign if one of the multiplied numbersis of a negative value and that can be obtained byXORing the sign of two inputs.

Exponent Addition is done through unsigned adderfor adding the exponent of the first input to theexponent of the second input and after that subtractthe Bias from the addition result (i.e. E1+E2- Bias). The result of this stage can be called asintermediate exponent. Significant Multiplicationis done for multiplying the unsigned significant and placing the decimal point in the multiplicationproduct. The result of significant multiplicationcan be called as intermediate product (IP). Theunsigned significant multiplication is done on 24bit. The result of the significant multiplication(intermediate product) must be normalized to havea leading "1" just to the left of the decimal point(i.e. in the bit 46 in the intermediate product).Since the inputs are normalized numbers then theintermediate product has the leading one at bit 46or 47. If the leading one is at bit 46 (i.e. to the leftof the decimal point) then the intermediate productis already a normalized number and no shift isneeded. If the leading one is at bit 47 then theintermediate product is shifted to the right and theexponent is incremented by 1.

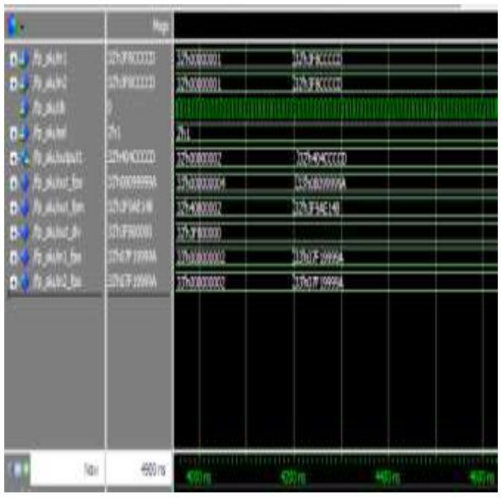

Fig. 1:Simulation result of decimal inputs 2.5& 4.75 for adder in modelsim wave window

3.3 Floating Point Division

The algorithm for floating point multiplication isexplained through flow chart in Figure 1. Let N1 andN2 are normalized operands represented by S1, M1,E1 and S2, M2, E2 as their respective sign bit,mantissa (significant) and exponent. If let us say weconsider x=N1 and d=N2 and the final result q hasbeen taken as "x/d". Again the following four stepsare used for floating point division.

1. Divide significant, subtract exponents, anddetermine sign M=M1/M2 E=E1-E2 S=S1XORS2
2. Normalize Mantissa M (Shift left or right by 1) andupdate exponent E
3. Rounding the result to fit in the available bits
4. Determine exception flags and special valuesThe sign bit calculation, mantissa division, exponentsubtraction (no need of bias subtraction here),rounding the result to fit in the available bits andnormalization is done in the similar way as has beendescribed for multiplication.

4. FLOATING-POINT UNIT IN THE SYNERGISTIC PROCESSOR

The floating-point unit (FPU) within the synergistic processor part (SPE) of a CELL processor may be a absolutely pipelined 4-way single-instruction multiple-data (SIMD) unit designed to accelerate media and information streaming with 128-bit operands. It supports 32-bit single-precision floating-point and 16-bit number operands with 2 completely different latencies, six-cycle and seven-cycle, with eleven FO4 delay per stage. The FPU optimizes the performance of vital single-precision multiply-add operations. Since precise rounding error, exceptions, and de-norm range handling don't seem to be vital to multimedia system applications, IEEE correctness on the single-precision floating-point numbers is sacrificed for performance and easy style. It employs fine-grained clock gating for power saving. the planning has 768K transistors in one.3 mm2, made-up SOI in 90-nm technology. Correct operations are ascertained up to five.6 GHz with one.4 V and 56°C, delivering forty four.8 GFlops. design, logic, circuits, and integration square measure co-designed to satisfy the performance, power, and space goals.

CONCLUSION Improvement in Floating-point operations by minimizing the time consumed for FPU operations, power consumed in floating point operations and space utilization which will enhance the working of digital signal processing & other many operations. Existing floating point operations have limitations that it can implement on only one type of hardware either 32 bits, 64 bits & 128 bits. The VHDL code written for complete 32-bit floatingpoint arithmetic unit has been implemented and testedon Xilinx. The designed arithmetic unit operates on32-bit operands. It can be designed for 64- bitoperands to enhance precision. It can be extended tohave more mathematical operations like trigonometric,logarithmic and exponential function.

REFERENCES

[1] X. Wang and M. Leeser, "Vfloat: A variable precision fixed- and floating point library for reconfigurable hardware," ACM Trans. Reconfigurable Technol. Syst., vol. 3, no. 3, pp. 16:1–16:34, Sep. 2010.

[2] K. S. Hemmert and K. D. Underwood, "Fast, efficient floating-point adders and multipliers for

FPGAs," ACM Trans. Reconfigurable Technol.Syst., vol. 3, no. 3, pp. 11:1– 11:30, Sep. 2010.
[3] A. Baluni, F. Merchant, S. K. Nandy, and S. Balakrishnan, "A fully pipelined modular multiple precision floating point multiplier with vector support," in Proc. ISED, 2011, pp. 45–50.
[4] K.Manolopoulos, D. Reisis, and V. Chouliaras, "An efficient multiple precision floating-point multiplier," in Proc. 18th IEEE Int. Conf. Electron.,Circuits Syst., 2011, pp. 153–156.
[5] A. Isseven and A. Akkas, "A dual-mode quadruple precision floatingpoint divider," in Proc. 40th ACSSC, 2006, pp. 1697–1701.
[6] A. R. Lopes, A. Shahzad, G. A. Constantinides, and E. C. Kerrigan, "More flops or more precision Accuracy parameterizable linear equation solvers for model predictive control," in IEEE Symposium on Field Programmable Custom Computing Machines, Napa, California, 2009.
[7] J. Maciejowski, "Predictive Control with Constraints,"Prentice Hall, Pearson Education Limited, Harlow, UK, 2001.
[8] R. Strzodka and D. G¨oddeke, "Pipelined mixed precision algorithms on FPGAs for fast and accurate PDEsolvers from low precision components," in IEEE Symposiumon Field-Programmable Custom ComputingMachines (FCCM 2006), Apr. 2006, pp. 259–268.
[9] A. Buttari, J. Dongarra, J. Kurzak, P. Luszczek, and S. Tomov, "Using mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy," ACM Trans. Math.Softw.,vol. 34, no. 4, pp. 1–22, 2008.
[10] J. Sun, G. Peterson, and O. Storaasli, "High performance mixed-precision linear solver for fpgas,"IEEE Trans. on Computers, vol. 57, no. 12, pp. 1614–1623, 2008.