QUANTITATIVE ANALYSIS OF 64-BIT FLOATING POINT ARITHMETIC HARDWARE INTERPRETER ON FPGA BASED CUSTOM COMPUTING MACHINES

Renukadevi.S[1], Rajasekaran.S[2]

[1]Research Scholar, Mother Teresa Women's University, Kodaikanal, India.

[2]B.S.Abdur Rahman University, Chennai, India.

Email: [1]srajayaganesh@yahoo.com.sg

Abstract

Analog and mixed signal circuit simulation often employs the use of the so called LU decomposition method to solve a set of linear algebraic equations represented as Ax=b, where A is a square matrix. The LU method requires the factorization of A into two tri-diagonal matrices. Factorization is O(n3) time and dominates the execution time of the LU decomposition method. A number of approaches have been developed for reducing the execution time of LU factorization. One approach is to unroll the factorization algorithm and considering each resulting assignment statement to be a machine operation, interpret the instruction stream. If an interpreter is implemented in a special purpose hardware engine there may be efficiencies to be gained by using a uniquely developed floating point unit within the hardware interpreter. This paper documents the research in exploring alternatives in the design of a special purpose double precision floating point unit for a hardware interpreter to perform LU factorization using unrolled code. Alternatives explored were primarily looking at integer adder and multiplier units of the floating point unit to determine the speed and area for each integer unit that was considered. A floating point divider algorithm was also explored and studied. The result of this paper for the pipelined floating point unit gives the best performance with the Block Carry look-ahead integer adder, Booth-2 integer multiplier and the SRT integer divider units. One of the interesting aspects of this research was heavy use of rapid prototyping all models were implemented in Verilog to evaluate system level performance.

Keywords: IEEE-754, Floating point Unit, 64-bit FPU.

I. INTRODUCTION

The problem known as system of simultaneous linear algebraic equations with n unknown is one of the general topics in linear algebra. It may be written in terms of matrix operations such as Ax=b, where A is a matrix of size n*n and X and b are vectors each of size n. Once the mapping of A and the vector b are known there is this problem whether and when there exist a proper x obeying equation (1) and how to derive it. In particular, the LU decomposition method is used for solving such a system of linear equation. Matrix A is factorized as A=LU where L is the lower triangular matrix and U is the upper triangular matrix. After the two factors of the matrix L and U are calculated they are used to find the vector x by forward and backward substitution methods. Equation (1) which constitutes a system of linear equations are used in analog and mixed signal circuits. Generally, as the size of the matrix increases the time taken to simulate it also increases which leads to the need to reduce the time required to solve the set of linear equations.

The objective of this paper is to duty a FPGA based floating point method to be used in a hardware interpreter.

$$\begin{array}{cccccc} a_{11} & a_{12} & a_{13} \ldots & a_{1n} & x_1 & b_1 \\ a_{21} & a_{22} & a_{33} \ldots & a_{2n} & x_2 & b_2 \\ \ldots & \ldots & \ldots \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & a_{n3} \ldots & a_{nn} & x_n & b_n \end{array} \tag{1}$$

The interpreter which takes in a stream of variable length instructions representing the symbolic unrolled instructions in the LU factorization algorithm [26].The output of the interpreter is the sparse L and U factors where the matrix values are in IEEE-754 double precision format. The hardware model consists of a control unit and a floating point unit which produces one matrix element every cycle [25]. Since the area consumed by floating point execution units on FPGA is well known to be very large, several approaches to reduce the area and execution time of the FPU are investigated. The controller inputs and decodes instructions, controls data flow to and from the floating point unit and output data to the LU memory [25]. Thus the execution rate of the FPU is primarily determined by the speed and rate of data provided by the controller to the FPU and the rate of data that can be received by the controller from the FPU.

Hardware interpreter

The hardware interpreter is a dedicated hardware engine as shown in fig.1, to generate the L and U factors from the instruction stream given by the preprocessor [26]. An unrolled LU factorization code for a given sparse matrix is given in an encoded form to the control unit. The control unit [25] of the hardware interpreter then decodes the instructions and the memory which form the integral part of the hardware interpreter

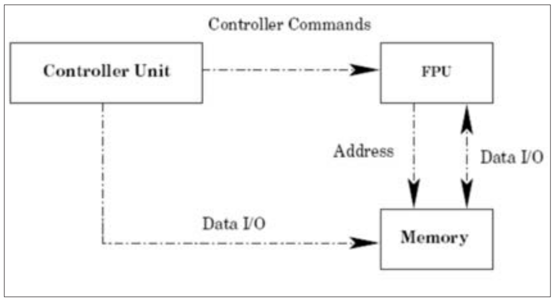


Fig. 1. Block Diagram of the Hardware Interpreter

The control unit gives two 64-bit input operands to the FPU core which are of IEEE-754 format. The FPU writes back the output matrix element, which is of 64-bits, to the memory. The memory exchanges data I/O with the control unit and the FPU core.

II. ARCHITECTURE OF THE FLOATING POINT UNIT

During every clock cycle, two 64-bit operands are given either to the divider or to the multiply-add unit. Note that in fig.2, the output of the divider is one of the inputs to the multiplier and the output of the multiplier is one of the inputs to the subtracter. There are temporary registers which are the supporting elements of the FPU core that act as a buffer. Temporary1 register stores the output of the divider (temp1 register), temp2 register stores the input to the FP multiplier and the latency of the FPU core elements.

Each operation consists of four parts: tag, opcode, operand and the address. The tag (set of binary numbers whose length depends on the number of operations) is the same for a set of divider and multiply-add operations, hence it helps to connect the multiply –add to the performed and accordingly the inputs are sent to the corresponding FPU core. Signals Complete and done indicate the completion of the operation by the divider and the multiply-add units respectively. There are two outputs from the FPU i.e. the output from the divider and the multiply-add unit which constitute the matrix elements. The control unit recognizes these signals and writes-back the output to the memory [25].

The memory used in this paper is single-ported in nature that means it can handle either a read or a write every cycle. The study of different kinds of memory which can handle more than one output of the FPU and also provide input to the FPU at the same time is a totally different research problem and has been handled in this paper.

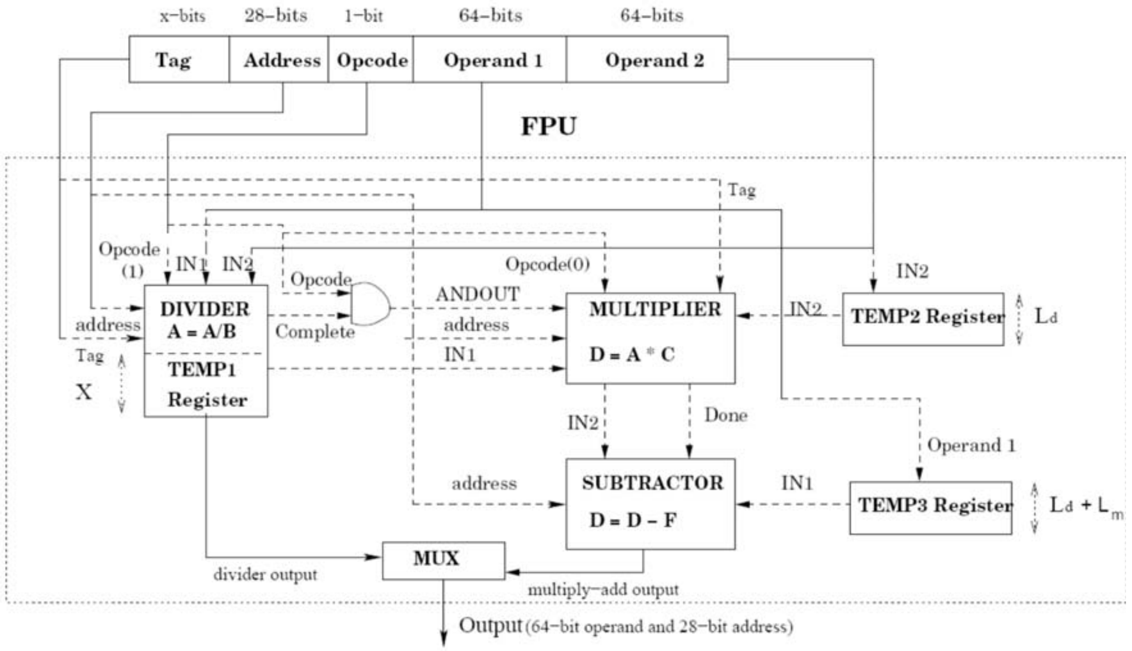
Design of the data path

The data path consists of a FPU core and supporting elements. Each core element is pipelined to increase its throughput. In this study, initially, the numbers of pipeline stages were equal to the number of tasks for each core element algorithm. The number of stages was modified if needed to improve the performance or area. During the study, the initial fully partitioned staged circuits were examined to identify where stages could be combined with a corresponding increase in execution speed or reduced circuit area.

The linear pipeline which is a cascade of processing stages each consisting of the combinational circuit that performs the arithmetic operations on the data flowing through the pipe. Stages are separated by high speed interface latches that are fast registers to hold the intermediate results. The pipe is under the control of a common clock. Once a linear pipelining is filled up, it produces one result per clock cycle, independent of the number of stages it contains. The whole design has been implemented in Verilog HDL. The data and control flow of the interpreter is shown in fig.2.

III. IMPLEMENTATION USING VERILOG

Verilog can be used to represent several different abstraction levels [2]. The level chosen to represent the 64-bit floating-point adder includes a mixture of register-level and logic-level logic block. To describe the register transfer level and the data flow through the adder, Verilog constructs called processes provide the sequential instruction that manipulate the data. Each process contains a sensitivity list that triggers the actions within a process. [14, 16] Provides idea on how to code more efficiently to use the synthesis tool capabilities.



L_d - Latency of the divider, L_m - Latency of the multiplier, X - Average number of multiply-add operations for the respective divide operations

Fig. 2. Architecture of the floating point unit

A. Floating-Point Addition

The pipelined design of the 64-bit floating point adder has a latency of seven clock cycles. Once the pipeline has been filled, the adder can generate a result each clock cycle so long as new operands are given every clock cycle thereafter. The design follows the traditional algorithm except that the stage has been split into multiple stages to allow the adder to run at a slightly faster clock speed. Basically, by having more stages with fewer logic levels, the pipeline can be run at faster clock speeds.

The following section describes the algorithm used to partition the different stages of manipulation and the Verilog constructs used to synthesize the FPU. The adder design pipelines the steps to achieve a summation every clock cycle. Each pipeline stage performs operations independent of others. Input data to the adder continuously streams in from the multiplier. The following sub-sections describe each of the adder pipeline stages in more detail as shown in fig.3.

Stage 1: Unpacking operands

The two sign, exponent and mantissa bits for operand A and operand B are latched in registers which are 1-bit, 11-bit, and 52-bit in length. The inputs are checked for special values like Not a number, Zero and the appropriate flags are set which is passed on through all stages.

Stage 2: Calculate Exponent and Sign

The second stage in the adder used comparator logic to place the larger of the two operands as operand A. The combinational Verilog process compares the exponents. If the exponents are equal, the logic then compares the mantissa values. The comparator is left to the synthesis tool. Sign bits of the two operands are XOR'ed. Sign bits do not affect the comparison. The registered process handles insertion of the implied leading-once for each new mantissa value. The leading-one insertion operation always takes place regardless of the operand values.

Stage 3: Shift Mantissa Stage

In order to add two floating-point values in scientific notation, the two values must have the same exponent in both sign and magnitude. The adder must perform this operation by shifting one of the operands and making adjustments to the operand exponent value. Stage 1 of

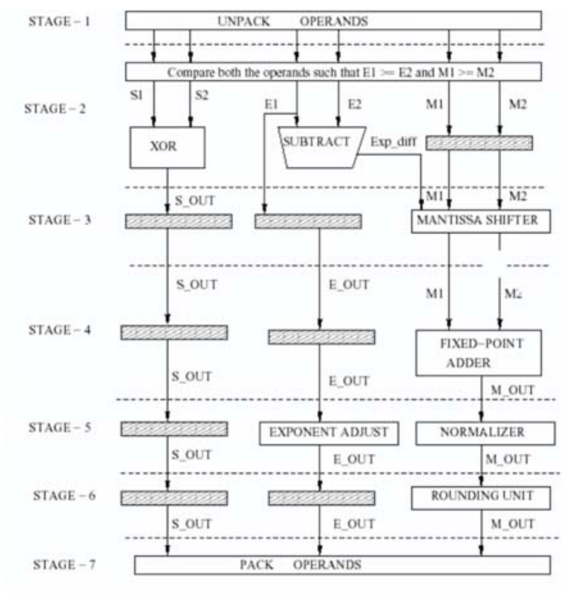


Fig. 3. Pipeline stages of floating point adder

the opening takes the difference of the two operand exponents to determine how many shifts are needed on operand B. By shifting to the right, the operand stands to lose only lower significant bits. The maximum number of shifts needed is 53. Even though barrel shifter is simple in its design, each input bit is directly connected to 53 or more output lines, which needs large number of connections. Once alternative is a three-level combinatorial shifter to shift 53 bits of the mantissa by having the first-level shift the bits by 0,1,2 or 3 bit positions, and letting the second-level shift the bits by multiples of 4 (i.e. 0,4,8,12) and the third-level by 16 (i.e.0,16,32,48,64). This way, shifts from length 0 to 53 can be performed.

Stage 4: Mantissa Addition Stage

Stage 4 of the adder pipeline performs the addition of the two mantissa integer values. Note that since operand A is greater than operand B, a borrow cannot happened in subtraction, and thus, the carry-out bit of the result is cleared. The carry-out bit becomes important in the next stage of the pipeline which may indicate the result needs neither further normalization nor exponent adjustment. If an addition took place with a carry-out, an immediate adjustment to the exponent must be done prior to the normalization stage since the bit does not take part in the 53-bit mantissa result vector. To do so, the stage must shift the result vector to the right by once to accommodate the carry-out bit as the new leading-one. This is the only stage which gives us scope to study different fixed-point adders to determine which gives the best performance. The different kinds of integer adder used for comparative study are Ripple carry adder(RCA), Carry look -ahead adder(CLA), Carry Select adder(CSA) and Carry Save adder. All adders are implemented at the logic-level and the exponent and sign bits are stored in delay registers. Overflow and underflow are checked and appropriate flags are set.

Stage 5: Normalization Shift Calculation stage

Stage 5 determines the number of shifts required to normalize the resulting mantissa value after the addition takes place. Comparator logic is used here to find the first leading-one digit from the MSB. A counter maintains the number of comparisons made which is the equal to the number of shifts needed. The shift value is used to normalize the mantissa such that the leading one in the mantissa resides in the most significant bit location. This stage also uses the shift value to adjust the exponent to the number of shifts required. Shifter in this stage is left to the synthesis tool. Normalized mantissa, the exponent, and sign are passed onto the next stage in the pipeline.

Stage 6: Rounding the result

The mantissa after normalization is rounded for precision and accuracy. This research deals with only one mode of rounding which adheres to IEEE-754 standard, which is rounding to the nearest, where the round bit is calculated with the combination of guard (g), sticky (s) and round(r) bits of the mantissa. If the combination of the three bits, i.e. 'gsr' is greater than, which is '101' in binary, then actual mantissa which is from 0 to 52 bits is increased by one, else if the value of 'gsr' is lesser than 5 then there is no change to the mantissa. A 3-bit comparator or is used to compare the 'gsr' bits the resultant mantissa sign and exponent are passed onto the final stage.

Stage 7: Write the result back to register

Finally sign, exponent and mantissa re concatenated to form the final result. The special condition flags are checked and if any of the flags are set high, then the result various accordingly. The result is stored back in a 64-bit register and passed as an output from the FPU.

B. Floating-Point Multiplication

The 64-bit FP adder has latency of six clock cycles as depicted in fig. 4. The multiply mantissa stage is a focus of study in this research. Different fixed-point multipliers used in this stage are either non-pipelined or fully pipelined. A comparative study of such architecture gives the tradeoffs between performance and area. The following section describes the algorithm used to portion the different stages of and the Verilog construct used.

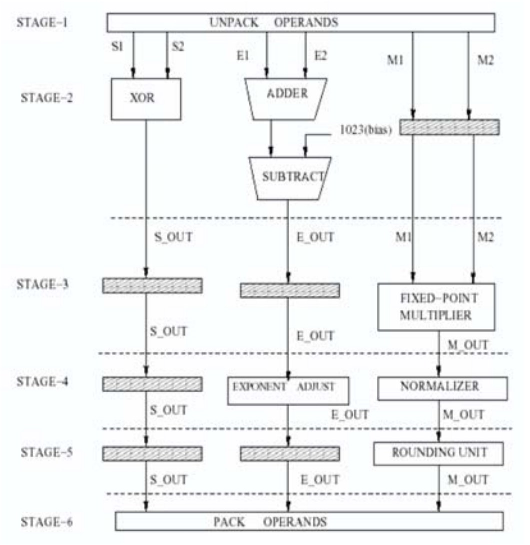


Fig. 4. Pipeline stages of floating point multiplier

Stages 1, 4 and 6 are done same like adder

Stage 2: Calculate exponent and sign

The exponents of operand A and operand B are added together and the result is subtracted from the bias, 1023 which gives the resultant biased exponent. The sign bits of the two operands are XOR'ed to give the sign bit. The sign and exponent output are passed on through all stages.

Stage 3: Multiply mantissa

The mantissa fields of operand A and operand B are multiplied. The output of the fixed point multiplier is double the mantissa length. The different kinds of fixed-point multipliers used for comparative study are shift-add multiplier, Booth-2 multiplier and Booth-3 multiplier 3. Both non-pipelined and fully-pipelined fixed-point multipliers have been studied. Hence the numbers of pipeline stages vary from 6 for a non-pipelined to 56 stages for a fully-pipelined FP multiplier. The exponent and sign bits are stored in delay resisters.

Stage 5: Rounding

Rounding of the mantissa is similar to the operation done in FP adder except that the lower 53 bits of the mantissa is used for rounding. Using the Verilog construct the sticky bit which ranges from (0-51) bits of the mantissa to 1-bit can be reduced. 52nd bit of the output is the round bit. Accordingly if either the round bit is '1' or both round bit and sticky bit are '1' then the mantissa is incremented by '1' else it remains the same.

C. Floating-Point Division

The 64-bit FP divider has a latency of six clock cycles as depicted in fig.5. Only one fixed point divider has been studied. The following section describes the algorithm used to partition the different stages of the Verilog constructs used.

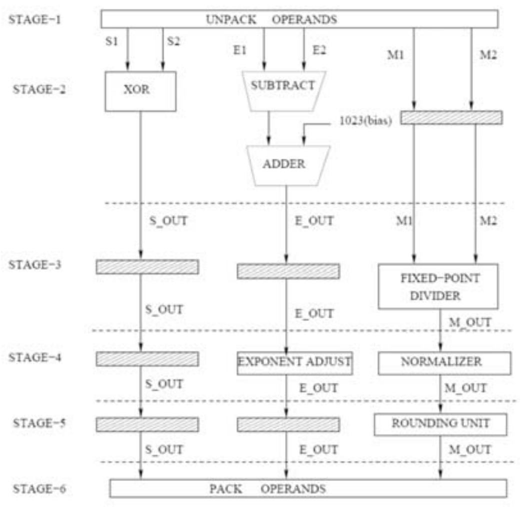


Fig. 5. Pipeline stages of floating point Divider

Stages 1, 4, 5 and 6 are done same like adder.

Stage 2: Calculate the exponent and sign

The exponents of operand A and operand B are subtracted and the result is added to the bias, 1023 for double-precision number. The sign bits of the two operands are XOR'ed. The resultant sign and exponent is passed through all stages.

Stage 3: Divide Mantissa

The mantissa fields of operand A and operand B are divided. The integer divider studied used the SRT Radix-2 algorithm. The exponent and sign bits are stored in delay registers.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

FPU performance is assessed by considering each component in turn that is, the floating point adder, floating point multiplier and the floating point divider. The objective was to determine the efficient FPU with the optimal number of pipeline stages that produces results every clock cycle. The design was targeted on a Xilinx VIRTEX FPGA XCV1000.The number of CLB slices available on the XCV1000 is 12,288 and hence the device utilization is specified as percent of the total number of slices present. This section describes all the experiments that were performed and their results. From these results, significant conclusions are made about the performance of the data. There are three different floating point units-adder, multiplier and the divider each of which has different fixed-point unit to perform addition, multiplication and division respectively. Each unit is pipelined to give maximum performance. Random binary test vectors were generated of length 64-bit, and were used to verify the output of each FP core element. The vectors used for testing were pre-selected such that it included all exceptions. The floating-point numbers included zero, maximum/minimum positive and negative numbers. Special values include positive and negative infinity and Not a Number. The output of the FP core elements are of IEEE-754 format. Appropriate flags are set for the special values and on an occurrence of overflow/underflow. The simulation results were verified both after the behavioral design and the structural design. The results for each fixed-point unit algorithms were also verified with the standard simulator.

Floating-Point Adder

A Floating point (FP) adder is used to subtract two IEEE-754 double precision numbers. FP adder is analyzed for various configurations, each differing only by the fixed point adder used. Thus the primary focus is on the performance of the fixed-point adders. Obviously, the highest performance fixed-point adder results in the fastest FP adder. The fixed point adders considered in this study are:

1. Ripple Carry Adder (RCA)
2. Carry Look Ahead Adder (CLA)
3. Carry Select Adder (CSA)
4. Block Carry Look Ahead Adder (BCLA)

Analysis of FP adder

The section analyses the performance of the FP adder for each of the fixed-point adders considered. The Table-I below shows the area timing results of the FP adder after place and route on VIRTEX FPGA. After analyzing the results in Table I, it is concluded that floating point adder using RCA occupies the maximum area and is the slowest and occupies the maximum area. The FP adder using CLA, CSA and BCLA run at the same speed of 9.1 MHz and take the same time to produce a result, but of the three, the one using BCLA as the fixed-point adder has the advantage of occupying the minimum area compared to CLA and CSA. Thus based on the performance, FP adder using BCLA as the fixed point adder has the highest performance and minimum area.

Table 1. Summary of FP Adder Performance for each Fixed-Point Adder

FPA (fixed-point adder used)	Latency (No.of pipeline stages)	Total Area		Clock Cycle Length (in ns)	Frequency of FPA (in Mhz)
		% of Slices	No. of Slices (out of 12,288)		
FPA (RCA)	4	7.1	867	149.4	6.7
FPA (CLA)	6	6.2	764	110.4	9.1
FPA (CSA)	6	6.4	787	110.4	9.1
FPA (BCLA)	6	6.1	750	110.4	9.1

B.Floating-Point Multiplier

Floating-Point multiplier is to multiply two IEEE-754 double precision numbers. Architectures used for fixed-point multipliers are

Shift Add multiplier (SAM)

Note that the FP adder using CLA, CSA and BCLA have the same speed of 9.1 MHz and latency of 6 cycles. If a FP multiplier is used with non-pipelined fixed point multiplication stage then B-2 gives the best performance whereas if the pipelined fixed point multiplier is used, then FP multiplier using any of the fixed-point algorithms which executes at a speed of 13.3 MHz but if it were to be traded off with area, then shift-add multiplier gives the best performance with a high latency (of 56 cycles). The FP divider, it executes at a speed of 1.3 MHz with a latency of 6 cycles.

The divider is the slowest of all and makes an impact only if the deal is with small matrices. In this paper large matrices of size unto 300*300 are dealt, where the occurrence of a divide operation is rare. Here the speed of the multiply-add unit dominates the speed of the FPU. The FPU can produce a maximum of unto two results per clock cycle depending on the size of the matrix, due to the pipelined structure.

1. Shift Add Multiplier (SAM)
2. Booth-2 multiplier (B-2M)
3. Booth-3 multiplier (B-3M)

The FP multipliers which are studied here differ only in the fixed point multiplier used in its implementation. Hence focus is on the performance of the fixed point multipliers. The fixed point multipliers can be non-pipelined, fully pipelined or partially pipelined.

Analysis of FP multiplier

The Table 2. presents the summary of results for FP multiplier with pipelined fixed point multiplier units. FP multiplier using SA, B-2 and B-3 execute at the same frequency of 13.292 MHz as the normalization of the mantissa stage dominates the speed of the unit. Studying the other performance parameters being area and latency, it is to be noted that SA has the minimum area but maximum latency of 57 cycles whereas B-2 has more area compared to SA but produces result every 32 cycle whereas B-3 occupies the maximum area. Based on the results, FP multiplier using B-2 as the fixed point multiplier has the best performance.

Table 2. Summary of FF Adder Performance for each non-pipelined Fixed-Point Multiplier

FPM (multiplier used)	Latency (No.of pipeline stages)	Total Area of FPM		Clock Cycle Length (in ns)	Frequency of FPA (in Mhz)
		% of Slices	No. of Slices (out of 12,288)		
FPM(SA)	3	41.3	5081	420.9	2.4
FPM(B-2)	3	39.3	4728	306.8	3.3
FPM(B-3)	3	38.1	4678	474.5	2.1

Floating-Point Divider

FP divider is used to divide two IEEE-754 64-bit numbers. The only algorithm considered for the fixed point divider part of the FP divider is the SRT algorithm (Radix-2). Thus a single (SRT) non-pipelined divider meeting the requirement of the FPU was investigated.

Analysis of FP divide

Only one fixed point algorithm for the divider, namely the SRT algorithm (radix-2) has been investigated. Depending on the fixed point unit the FP divider executes at a speed of 1.3 MHz and occupies an area of 65.1 % of total single-chip slices available. The performance of the FP diver is shown below in Table 3.

V. CONCLUSION

Note that the FP adder using CLA, CSA and BCLA have the same speed of 9.1 MHz and latency of 6 cycles. If a FP multiplier is used with non-pipelined fixed point multiplication stage then B-2 gives the best performance whereas if the pipelined fixed point multiplier is used, then FP multiplier using any of the fixed-point algorithms which executes at a speed of 13.3 MHz but if it were to be traded off with area, then shift-add multiplier gives the best performance with a high latency (of 56 cycles). The FP divider, it executes at a speed of 1.3 MHz with a latency of 6 cycles.

Table 3. Performance of FP Divider using SRT Fixed-Point Divider

Stage	Pipeline Stages	Device Utilization		Design Statistics	
		% of Slices	No. of Slices (out of 12,288)	Min Period (in ns)	Max Freq. (in Mhz)
Stage 1	Latch data	0.6	74	23.3	42.8
Stage 2	Exponent_Sign	0.5	67	29.2	34.2
Stage 3	Divider	60.6	7448	718.7	1.3
Stage 4	Mantissa(SRT)	2.5	303	58.1	17.2
Stage 5	Normalize Mantissa	0.5	67	52.1	19.2
Stage 6	Write Result	0.4	43	23.3	42.9

The divider is the slowest of all and makes an impact only if the deal is with small matrices. In this paper large matrices of size unto 300*300 are dealt, where the occurrence of a divide operation is rare. Here the speed of the multiply-add unit dominates the speed of the FPU. The FPU can produce a maximum of unto two results per clock cycle depending on the size of the matrix, due to the pipelined structure.

REFERENCES

[1] Heshan A1-Twaijry and Michael J. Flynn,1995, “Performance/area tradeoffs in booth multipliers Technical report”, Stanford University.

[[2] P.L.Brown, W.S. Richman, 1969, “The choice of base”, Communications of the ACM, Vol 12, No. 10.

[3] Xilinx Datasheet, 2001, “Virtex 2.5v field programmable gate arrays”. Technical report, Xilinx Inc., April.

[4] Stuart F, Oberman David L, Harris and Mark A. Horowitz, 1997, “Srt division architectures and implementations”. In Proceeding of the 13th IEEE Symposium on Computer Arithmetic.

[5] Stuart F.Oberman and Michael J.Flynn, 1997, “Division algorithms and implementations”. IEEE Transactions on Computers, VOL 4, No. 8.

[6] Chen-Ying Hsu, 1998, “Variable precision arithmetic processor in fpgas”, Thesis, University of Toronto.

[7] Kai Hwang, 1979, “Computer Arithmetic-Principles, Architecture, and Design”. John Wiley and Sons.

[8] J.A.Hidalgo, V.Moreno-Vergara, O.Oballe, A, Daza, M.J.Martin-Vazuez, and A.Gago, 1998, “ A radix-8 multiplier unit design for specific purpose”,

In XIII Conference of Design of Circuits and Integrated Systems (DCIS'98) Madrid. Dept. de Electronic, E.T.S.I. Industrials.

[9] J.Kahan, W.Palmer, 1979, "On a proposed floating-point standard. Technical report". SIGNUM Newsletter, Special Issue.

[10] Weng Fook Lee, 2000, "VHDL-Coding and Logic Synthesis with SYNOPSYS". Academic Press,

[11] Allison L. Walters, 1998, "A scaleable fir filter implementation using 32-bit floating point complex arithmetic on a FPGA based custom computing platform". Master's thesis, Virginia Polytechnic Institute and State University.

[12] Al Walters Nabeel Shirazes and Peter Athena's,1995, "Quantitative analysis of floating point arithmetic on FPGA based custom computing machines". In presented at the 5th International Workshop on Field Programmable Logic and Applications. Virginia Polytechnic Institute and State University.

[13] New York NY, 1985, "IEEE Standard for Binary Floating-Pont Arithmetic", Institute of Electrical and Electronics Engineers, ANSI/IEEE STD 754.

[14] Stuart F. Oberman and Michael J.Flynn, 1995, "Implementing division and other floating-point operations A system perspective". In proceedings of SCAN-95, International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numeric.

[15] Stuart F. Oberman and Michael J.Flynn, 1997, "Design issues in division and other flatting-point operations". IEEE Transactions on Computers.

[16] Stuart Franklin Oberman, 1996, " Design Issues in High Performance Floating Point Arithmetic Units". PhD thesis, Stanford University.

[17] Jan Ogrodzki, 1994, Circuit Simulation Methods and Algorithms. CRC Press.

[18] Akber Syed, 2002, "A hardware interpreter for sparse matrix Lu factorization". Master's thesis, University of Cincinnati.

[19] Sanjeev Thiyagarajan, 2001, "Reducing memory space for completely unrolled Lu factorization of sparse matrces", Master's thesis, university of Cincinnati.

[20] Gary W.Bewick, 1994, Fast Multiplication algorithms and implementation .PhD thesis, Stanford University.

[21] Hong Zhang, 1998, " An evolution of complete loop unrolling technique for solving sparse linear system of equations using direct methods", Master's thesis, University of Cincinnati.

Renuka Devi.S graduated from Meenakshi College for Women, Chennai during the year 1997 in Mathematics. She obtained her Master's degree during the year 1999 from Annamalai University, Chidambaram. She is doing her research work in the field of Computational Mathematics. Currently she is the Assistant Professor, in the Department of Science and Humanities, of Sri Ramanujar Engineering College, Chennai-600 048. She has published about nine papers to her credit in National and International Conferences.